# *JEUS EJB 안내서*

Document info

Document name: “JEUS EJB 안내서”

Document date: 2005-02-01

Manual release version: 1

Software Version: JEUS 5

# 차 례

# 그림 목차

# 표 목차

# 매뉴얼에 대해서

## 매뉴얼의 대상

이 책은 JEUS 시스템 관리자와 JEUS 위에서 J2EE Enterprise JavaBeans(EJB)를 개발, 배치(Deployment)하려는 개발자를 대상으로 하고 있다.

EJB 개발자들은 필요한 부분을 이 문서에서 찾을 수 있을 것이다.

## 매뉴얼의 전제 조건

이 책을 읽기 전에 독자는 JEUS 시스템에 대하여 전반적으로 깊이 이해하고 있어야 한다. 우선 먼저 JEUS Server 안내서를 읽기 권한다.

이 책에는 J2EE 스펙에 대한 설명을 담고 있지 않기 때문에 독자는 EJB 개념에 대해서 먼저 선수 학습이 되어 있어야 한다. 이 책의 거의 모든 정보들은 JEUS에 관련된 것들을 위주로 다루고 있으므로, 그 외의 필요한 지식들은 독자들이 이 책을 읽기 전에 습득해야 한다.

## 매뉴얼의 구성

본 매뉴얼은 JEUS의 메인 설정 파일인 JEUSMain.xml의 계층 구조를 따라서 구성되었다. 실제 이 구성은 JEUS의 구조와 비슷하다.

다른 JEUS 매뉴얼처럼 본 매뉴얼도 다양한 방법으로 읽을 수 있다.

- 일반 책 처럼처음부터 끝까지 통독할 수 있다.
- 각 장은 서로 독립적인 내용이므로 필요한 부분만 정독할 수 있다.
- 핸드북처럼 필요한 부분을 찾아서 볼 수 있다. 그러나 이렇게 하기 전에 먼저 JEUS에 대해서 충분히 이해하길 권한다.

JEUS EJB 안내서는 다음처럼 15 장으로 구분되어 있다

1. 소개
2. 따라하기
3. JEUS EJB의 개요
4. JEUS EJB Engine
5. JEUS EJB Module
6. JEUS의 일반적인 EJB들
7. Enterprise JavaBeans의 보안
8. Enterprise JavaBeans의 상호운용
9. Enterprise JavaBeans 클러스터링
10. Session Enterprise JavaBeans
11. Entity Enterprise JavaBeans
12. Message-dirven Enterprise JavaBeans
13. EJB Timer Service
14. EJB 클라이언트
15. 결론

또, 자세한 정보나 콘솔 툴, XML 설정 파일, API등을 소개하는 14개의 부록을 포함하고 있다. 목록은 다음과 같다

A ejbadmin 콘솔 툴 레퍼런스

B JEUS EJB 환경 변수C JEUS EJB 기본 Java Type과 DB  Field 매핑

D EJBMain.xml XML 설정 레퍼런스

E jeus-ejb-dd.xml XML Configuration Reference

F Instant EJB QL API 레퍼런스

이 문서는 top-down방식으로 구성되어 있다. 가장 포괄적인 개념을 먼저 기술하고, 그 아래 주요 컴포넌트들을 중심으로 세부적인 정보들을 설명하는 방식을 따르고 있다.

가장 중요한 1장부터 15장까지는 JEUS를 처음 접하는 관리자들이 반드시 읽어봐야 한다. 이 장들은 교과서와 같은 형식으로 되어 있으므로 앞 장부터 순서적으로 읽어나가야 한다.

문서의 맨 뒤에 있는 부록에는 앞 장에서 기술된 정보들을 좀 더 집약적이며 간결한 기술적인 예시로 설명되어 있다. 그러므로, 부록 문서들은 경험이 많은 JEUS Manager들에게 기술적 참조를 위한 좋은 리소스가 될 수 있을 것이다.

## 관련된 문서들

다음의 문서들은 이 책에서 발견할 수 있는 정보들을 포함하고 있는 문서들이다.

- JEUS Server 안내서
- JEUS 클라이언트 어플리케이션 안내서
- Sun Microsystems, Inc.의 EJB 2.1 Specification

# JEUS 5.0 에서 변경된 사항

이 절은 JEUS 버전 4.x과 비교해서 JEUS 버전 5.0 의 EJB의 가장 큰 변화에 대해서 언급한다. 이 절에서는 JEUS 4.x EJB를 이미 사용하고 있는 사용자를 위한 것으로, JEUS 5.0 EJB 매뉴얼에 대한 새로운 정보를 빠르게 찾을 수 있도록 한다. 아래에서 소개하는 변경된 내용을 제외하고는, 이전 매뉴얼과 동일하다.

- deployment 방식 : 기존의 4.x대에 있던 DIR, JAR, EAR 방식이 모두 융합되어 단일한 application deployment 방식으로 변경되었다. 또한 classloader 구조도 application deployment 방식에 맞추어 변경되고 deployment descriptor의 이름도 변경되었다. 변경된 방식을 이해하기 위해서는 JEUS Server 안내서의 deployment 부분을 참고하고 5장을 보기 바란다.

- webservice endpoint 지원 : EJB 2.1 spec에 맞추어 Stateless session bean의 client view interface에 webservice endpoint가 추가되었다. 실제로 EJB를 이용해 webservice를 제공하는 과정은 JEUS webservice 안내서에 자세히 나와 있다.

- EJB Timer Service : EJB 2.1 spec의 Timer service가 JEUS 5.0 EJB에 추가되었다. 이에 대한 설명은 13장을 참고하기 바란다.

- jeus-ejb-dd.xml과 EJBMain.xml 변경 : 설정 xml 파일들이 DTD 기반에서 XML schema 기반으로 변경되었다. 또한 여러가지 새로운 설정이 추가되고 이전의 설정 형식이 변경된 것도 있다. 이는 부록 D, E 를 참고하기 바란다.

- ejbadmin: deploy 방식 등의 변경으로 인해 ejbadmin의 명령어 형식에 많은 변화가 있으며 이 사항에 대해서는 부록 A를 보기 바란다.

JEUS 5.0의 또 다른 수정 내용과 새로운 기능에 대한 정보는 다른 JEUS 5.0 매뉴얼의 이 장을 참조하기 바란다.

# 일러두기

| 표기 예 | 내용 |
|---|---|
| 텍스트 | 본문, 12포인트, 바탕체 Times New Roman |
| *텍스트* | 본문 강조 |
| CTRL+C | CTRL와 동시에 C를 누름 |
| `public class myClass { }` | Java 코드 |
| `<system-config>` | XML 문서 |
| **참조: / 주의:** | 참조 사항과 주의할 사항 |
| **Configuration** 메뉴를 연다 | GUI의 버튼 같은 컴포넌트 |
| JEUS_HOME | JEUS가 실제로 설치된 디렉토리<br>예)c:\jeus50 |
| `jeusadmin nodename` | 콘솔 명령어와 문법 |
| [ 파라미터 ] | 옵션 파라미터 |
| < xyz > | '<'와 '>' 사이의 내용이 실제 값으로 변경됨. 예)<node name>은 실제 hostname으로 변경해서 사용 |
| \| | 선택 사항. 예) A\|B: A나 B 중 하나 |
| … | 파라미터 등이 반복되어서 나옴 |
| ?, +, * | 보통 XML 문서에 각각 "없거나, 한 번", "한 번 이상", "없거나, 여러 번"을 |

| 표기 예 | 내용 |
|---|---|
| | 나타낸다. |
| . . . | XML이나 코드 등의 생략 |
| *<<FileName.ext>>* | 코드의 파일명 |
| *그림 1.* | 그림 이름이나 표 이름 |

## OS 에 대해서

본 문서에서는 모든 예제와 환경 설정을 Microsoft Windows™의 스타일을 따랐다. 유닉스같이 다른 환경에서 작업하는 사람은 몇 가지 사항만 고려하면 별무리 없이 사용할 수 있다. 대표적인 것이 디렉토리의 구분자인데, Windows 스타일인 "\"를 유닉스 스타일인 "/"로 바꿔서 사용하면 무리가 없다. 이외에 환경 변수도 유닉스 스타일로 변경해서 사용하면 된다.

그러나 Java 표준을 고려해서 문서를 작성했기 때문에, 대부분의 내용은 동일하게 적용된다.

## 용어 설명

다음에 나오는 용어들은 이 책에서 주로 사용되는 것들이다. 이해하지 못하거나 의미가 불분명한 것들은 이 리스트에서 제공된 설명을 통하여 이해하도록 하자.

| 용어 | 정의 |
|---|---|
| **Activation** | *passivation* 의 반대 개념. 빈 인스턴스를 제 2차 저장장소에서 주 메모리로 이동하는 것을 의미한다. |

| 용어 | 정의 |
|---|---|
| **Active management** | JEUS에서, 비정상적인 상황에서 자동적으로 email이 발송되는 것을 의미한다. |
| **Bean pool** | EJB 인스턴스의 풀. |
| **Bean type** | J2EE의 기본 빈 타입이거나 (예, stateless, stateful, entity and message-driven bean) 개발자가 정의한 빈 (기본J2EE빈의 하위 빈). |
| **BMP** | Bean-managed persistence entity bean. |
| **클러스터링** | 여러 EJB engines 에 분산되어 있는 EJB 빈의 설정. 클라이언트 입장에선 하나의 빈처럼 보이지만 내부적으로는 안정성과 성능을 향상하기 위해 부하분산이 이루어 진다. 9장을 참조하기 바란다. |
| **CMP** | Container-managed persistence entity bean. |
| **커넥션 풀** | EJB Objects의 풀. 이 objects 는 클라이언트와 EJB간에 연결 및 통신을 책임진다. |
| **DD** | deployment descriptor의 약자. |
| **EJB** | Enterprise JavaBeans |
| **EJB client** | EJB의 메소드를 사용하는 컴포넌트(예를 들면, 독립적인 Java application, Servlet 또는 다른 EJB). |
| **EJB engine** | J2EE 스펙에서 “EJB container”. 이것은 EJB컴포넌트의 기반 환경을 제공한다. |

| 용어 | 정의 |
|---|---|
| **EJB Module** | EJB JAR안에 있는 하나 혹은 그 이상의 EJB패키지. EJB engine에 Deploy 되는 EJB는 EJB modules을 사용한다. 5장을 참조 하라. |
| **Entity 캐시** | entity 빈에서 비활성화된 빈을 유지하기 위해 사용되는 특별한 캐시. 이 캐시가 채워졌을 때만 빈이 제 이차 저장장소로 비활성화 된다. 이것은 성능 향상을 위해 사용 된다. |
| **Fail over** | 같은 EJB어플리케이션을 실행하는 여러 engine을 사용함으로써 안정성을 높일 때 사용. |
| **부하 분산** | 같은 EJB어플리케이션을 실행하는 engine에 클라이언트의 요청을 공정히 분배함으로써 성능을 향상시킬 때 사용. |
| **MDB** | Message-driven bean. |
| **Object management** | 이 용어는 *connection pool* 과 *bean pool* 환경을 참조하기 바란다. |
| **Passivation** | EJB인스턴스를 다시 요청이 있을 때까지 주 메모리에서 제 이차 저장 장소로 옮겨놓는 것이다. |
| **주 메모리** | *runtime memory*라고도 한다. 컴퓨터 내부의 실제 가용 메모리 (RAM). |
| **Round robin** | 큐 데이타 구조와 같다. 큐에 들어온 순서대로 선택 된다. 이것은 컴포넌트 별로 공정한 부하분산을 보장한다. |
| **이차 메모리** | 영구적인 기억장소로써 유지를 위해 전원을 요구하지 않는다(예로는 하드 디스크가 있다). |

| 용어 | 정의 |
|---|---|
| **SF** | Stateful session bean. |
| **SL** | Stateless session bean. |
| **Thread ticket pool** | “access ticket”의 풀. JEUS에서 EJB에 접근하기 위해선 클라이언트는 반드시 thread ticket pool로부터 “thread ticket”을 얻어야 한다. |

# 연락처

**Korea**
Tmax Soft Co., Ltd
18F Glass Tower, 946-1, Daechi-Dong, Kangnam-Gu
Seoul 135-708
South Korea
Email: info@tmax.co.kr
Web (Korean):http://www.tmax.co.kr

**USA**
Tmax Soft, Co.Ltd.
2550 North First Street, Suite 110
San Jose, CA 95131
USA
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com

**Japan**
Tmax Soft Japan Co., Ltd.
6-7 Sanbancho, Chiyoda-ku,
Tokyo 102-0075
Japan
Email: info@tmaxsoft.co.jp
Web (Japanese): http://www.tmaxsoft.co.jp

**China**
Beijing Silver Tower, RM 1507, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
Tel: 86-10-64106148 Fax: 86-10-64106144
E-mail : info@tmaxchina.com.cn
Http://ww.tmaxchina.com.cn

# 1 소 개

Sun Microsystems Inc.에서 정의한 EJB 2.1 스펙에 따르면 Enterprise JavaBeans (“EJB”) 을 다음과 같이 정의 할 수 있다.

*“EJB 아키텍처는 객체지향 분산 엔터프라이즈 어플리케이션의 개발 및 분산 배치를 위한 컴포넌트이다. EJB로 작성된 어플리케이션은 확장성, 트랜잭션 처리 그리고 동시 사용자 처리에 안전하다. EJB 스펙을 준수한 어플리케이션은 한번 작성되면 어떠한 서버 플랫폼에도 배치 된다.”*

JEUS WAS는 위에서 언급한 “서버 플랫폼”으로서 본 매뉴얼은 JEUS에서 EJB아키텍처를 어떻게 구현했는지 기술되어 있다.

본 매뉴얼은 다른 JEUS 매뉴얼과 같이, 모든 스펙에 대해서 설명하는 것이 아니고, JEUS WAS에서 구현된 스펙과 특징에 대해서만 설명한다. 이런 관점에서 볼 때 이 매뉴얼은 EJB스펙과 구현된 스펙간에 간격을 줄이는데 도움이 될 것으로 본다.

# 2 따라하기

아래의 간단한 단계들은 JEUS EJB의 시작부터 발전된 단계로 갈 수 있도록 구성 되었다.

아래에 있는 예제에 따르면, JEUS 노드 이름은 "johan"이다. 이 이름을 보게 되면 컴퓨터 이름으로 대체하면 된다(이것은 JEUS node name과 같다).

1. 먼저, JEUS의 정상적인 설치와 환경 변수를 확인한다(특별히 시스템 PATH에 JEUS_HOME\bin\ 디렉토리가 설정되어 있는지 확인한다). JEUS설치에 대한 정보는 JEUS 설치 안내서를 참조하기 바란다.

   참고: 이 매뉴얼을 통해서"JEUS_HOME"을 보면 실제 JEUS의 설치 디렉토리로 생각하면 된다 (예 "c:\jeus50").

2. "JEUS_HOME\config\johan\JEUSMain.xml"에 적어도 하나의 EJB Engine이 설정되어 있어야 한다. 아래의 예제에 굵은 글자 부분을 확인 하기 바란다.

<<*JEUSMain.xml*>>

```
<?xml version="1.0"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <node>
        <name>johan</name>
        <engine-container>
            <name>mycontainer</name>
            <engine-command>
                <type>ejb</type>
                <name>engine1</name>
            </engine-command>
            <engine-command>
                <type>servlet</type>
                <name>engine1</name>
            </engine-command>
        </engine-container>
    </node>
```

```
</jeus-system>
```

3. JEUSMain.xml 에 상응한 EJB engine 디렉토리가 있는지 확인한다. JEUS_HOME\config\johan\johan_ejb_engine1 디렉토리가 존재 해야 한다. 여기서 “engine1”이라는 것은 JEUSMain.xml에 EJB engine 이름이 engine1이기 때문이다.

4. JEUS_HOME\config\johan\johan_ejb_engine1\ 디렉토리상에, EJBMain.xml 파일이 존재해야 한다. 이것은 각각의 EJB engine의 설정 파일이며 아래와 같은 부분이 포함되어 있어야 한다.

<<*EJBMain.xml*>>

```
<?xml version="1.0"?>
<ejb-engine xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
</ejb-engine>
```

5. 위의 사항을 확인 후, command prompt (터미널 창)를 열고 ‘jeus’라고 입력한다. 그러면 JEUS Manager가 기동 된다.

6. 다른 command prompt를 열고 ‘jeusadmin johan’이라고 입력한다(다시 확인하는데, “johan” 은 각자의 컴퓨터 이름이다).

7. 관리자의 username 과 password 를 입력한다(이것은 설치 때 설정한다).

8. jeusadmin에 prompt가 보이면 ‘boot’라고 입력한다. JEUS node가 부팅되며 container와 engine이 기동 된다.

9. ‘help’라고 입력하면 ejb engine에 대한 모니터링과 관리에 대한 명령어들을 볼 수 있다.

10. 다른 command prompt 를 열고 ‘ejbadmin johan_mycontainer’ 을 입력한다(“johan” 은 각자의 컴퓨터 이름이고 , “mycontainer” 는 JEUSMain.xml에 설정된 container 이름이다).

11. username과 password를 입력한다.

12. prompt에 ‘help’라고 입력하면 EJB engine에 있는 EJB와 EJB모듈을 모니터링하고 관리할 수 있는 목록을 볼 수 있다 (만약 새롭게 설치된 것이라면, EJB와 모듈이 없을 것이다).

13. ‘ejbadmin’ tool을 빠져 나오려면 prompt에‘exit’을 입력한다.

14. ‘jeusadmin’ tool을 빠져 나오려면 prompt에 ‘down’, 그 다음 ‘jeusexit’ 그리고 최종으로 ‘exit’을 입력한다.

15. 전체 JEUS system 이 종료 된다.

“따라하기” 단계는 JEUS의 사용법을 간단히 보여준 것이고, 매뉴얼의 나머지 부분에서 위의 각 단계와 환경에 대해 자세히 설명한다.

만약 위의 단계에 문제가 있으면 JEUS 설치 안내서와 JEUS Server 안내서에 있는 정확한 환경 세팅을 참고 하기 바란다.

다음 장에서 JEUS EJB의 개괄을 살펴 봄으로써 EJB를 살펴 보도록 하겠다 .

# 3 JEUS EJB 의 개요

## 3.1 소개

J2EE Enterprise JavaBeans (이하 “EJB”) 스펙은 복잡하고, 중요한 고성능의 비즈니스 로직을 개발할 때 개발자의 노력을 최소화할 수 있는, 이식성이 뛰어난 비즈니스 컴포넌트 작성을 위한 구조를 기술하고 있다.

## 3.2 JEUS 에서 지원하는 J2EE EJB 스펙

JEUS 5.0 은 Sun Microsystems, Inc 사의 EJB 2.1 스펙을 모두 구현함으로써 완벽히 호환된다.

또한, J2EE 1.4 스펙도 준수하고 있다.

더 자세한 스펙 구현 정보들은 JEUS Server 안내서를 참조한다.

## 3.3 JEUS Server 에서 JEUS EJB

[그림 1] 은 JEUS Server 안내서에 언급된 내용으로서, 전체 JEUS 시스템 중 EJB의 위치를 나타낸다.

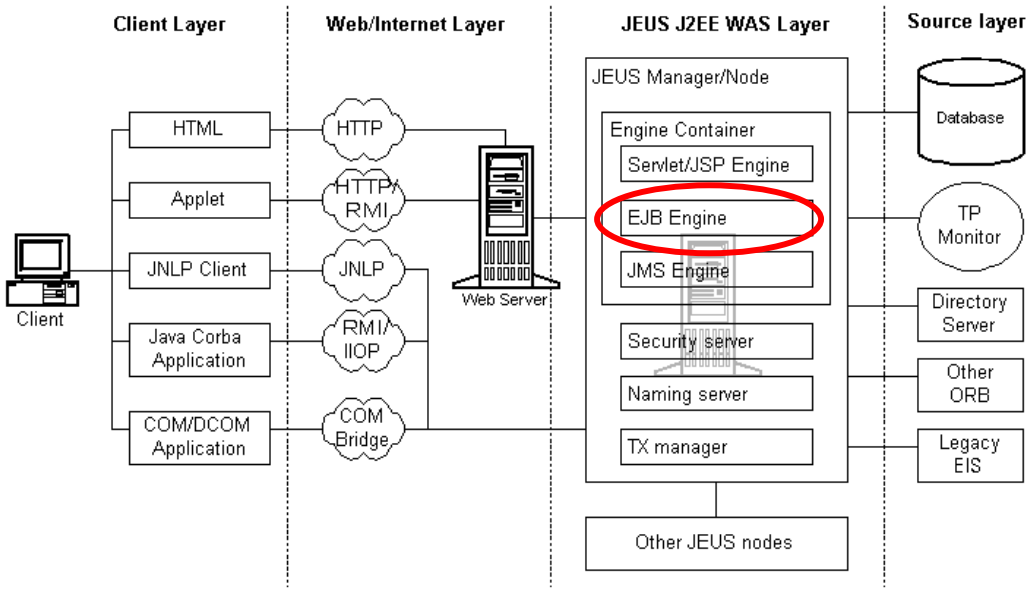


*그림 1 JEUS 에서의 EJB 파트*

[그림 1]은 전체 JEUS의 논리적인 engine 구성에서 EJB engine이 속해 있는 위치와 상호 연관 관계를 보여준다.

**참고**: 위의 그림은 물리적인 JEUS의 구조를 표현한다기 보다는 논리적인 구성을 나타낸다. 실제 시스템에서는 JEUS Manager와 engine container는 별도의 JVM과 주소영역에서 실행된다.

## 3.4 JEUS EJB Components

아래의 [그림 2]는 EJB engine(EJB Container 또는 EJB Server라고도 함)에서 설정할 수 있는 주요 컴포넌트들을 보여주고 있다.

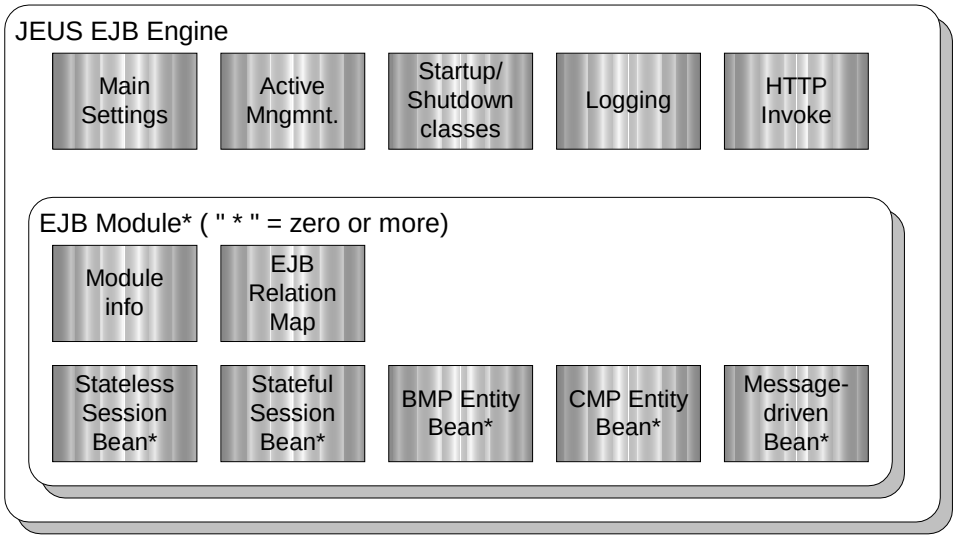


*그림 2 EJB 아키텍처의 개관*

위의 [그림 2] 에서 가장 중요한 컴포넌트들은 다음과 같다.

- **EJB engine**은 EJB 최상위의 컴포넌트로 Deploy된 EJB 모듈들과 EJB 들의 실행환경을 제공하는 책임을 맡고 있다. EJB engine은 4장에서 더 자세히 설명한다.

- **EJB module** 은 EJB를 그룹화하고 관리할 수 있는 단위이며 이 안에는 한 개 이상의 EJB를 포함하고 있다. JEUS 에서 EJB를 조립하고 Deploy하고 컨트롤할 때, EJB module을 단위로 한다. 따라서 모든 EJB는 반드시 어느 하나의 EJB Module 에 포함되어 있어야 한다. 5장에서는 이 EJB module에 대해 더 자세히 다룬다.

- **Enterprise JavaBeans**은 EJB module 내에 속해 있으며 실제 비즈니스 컴포넌트를 일컫는다. 모두 5개의 종류가 있으며, 다음에 이들을 설명한다(5개의 종류들은 [그림 2]의 아랫부분에 보이고 있다).

## 3.5 5 개의 Bean 종류

JEUS EJB는 다음과 같이 5개의 EJB 종류를 지원한다.

1. Stateless session bean (10장)
2. Stateful session bean (10장)

3. BMP entity bean (11장)
4. CMP 1.1 & CMP 2.0 entity bean (11장)
5. MDB message-driven bean (12장)

추가로, 6장에서는 각 EJB 종류마다 다른 특성들에 대해 설명한다.

이러한 Bean들의 Deploy와 관리는 EJB module의 단위로 이루어진다. 5장에서는 EJB module에 대해 설명한다.

## 3.6 클러스터링을 통한 성능과 안정성

성능과 안정성을 더하기 위하여 JEUS는 표준 외에 두 가지 추가 기능을 제공한다.

- 성능개선을 위한 부하 분산 기능
- 안정성 확장을 위한 Failover 기능

이 두 기능들을 가능하게 하려면 두 개 이상의 EJB engine에 EJB가 Deploy 되어 있어야 한다. EJB의 클러스터링은 9장에서 설명한다.

## 3.7 EJB 시스템 구조

이 장의 나머지 부분에서는 JEUS EJB에 관련된 주요 물리적 요소들을 살펴보기로 한다. 이러한 요소들은 EJB에 관련된 물리적 디렉토리 구조와 EJB 설정파일들, EJB 조합을 위해 디자인된 소프트웨어 툴들, 그리고 마지막으로 JEUS EJB를 관리할 수 있는 Deploy 툴 등으로 구성된다. 또한 우리는 EJB 시스템에 영향을 주는 전역 시스템 변수들과 특성들도 살펴보겠다.

다음과 같은 항목들은 좀 더 상세히 살펴보기로 한다.

1. JEUS EJB의 디렉토리 구조
2. JEUS EJB XML 설정파일의 구조
3. EJB 툴들과 유틸리티들
4. JEUS EJB를 위한 시스템 환경 변수들

# 3.8 JEUS EJB 에 관련된 디렉토리 구조

[그림 3]은 JEUS EJB에 관련된 디렉토리들과 파일들을 보여주고 있다.

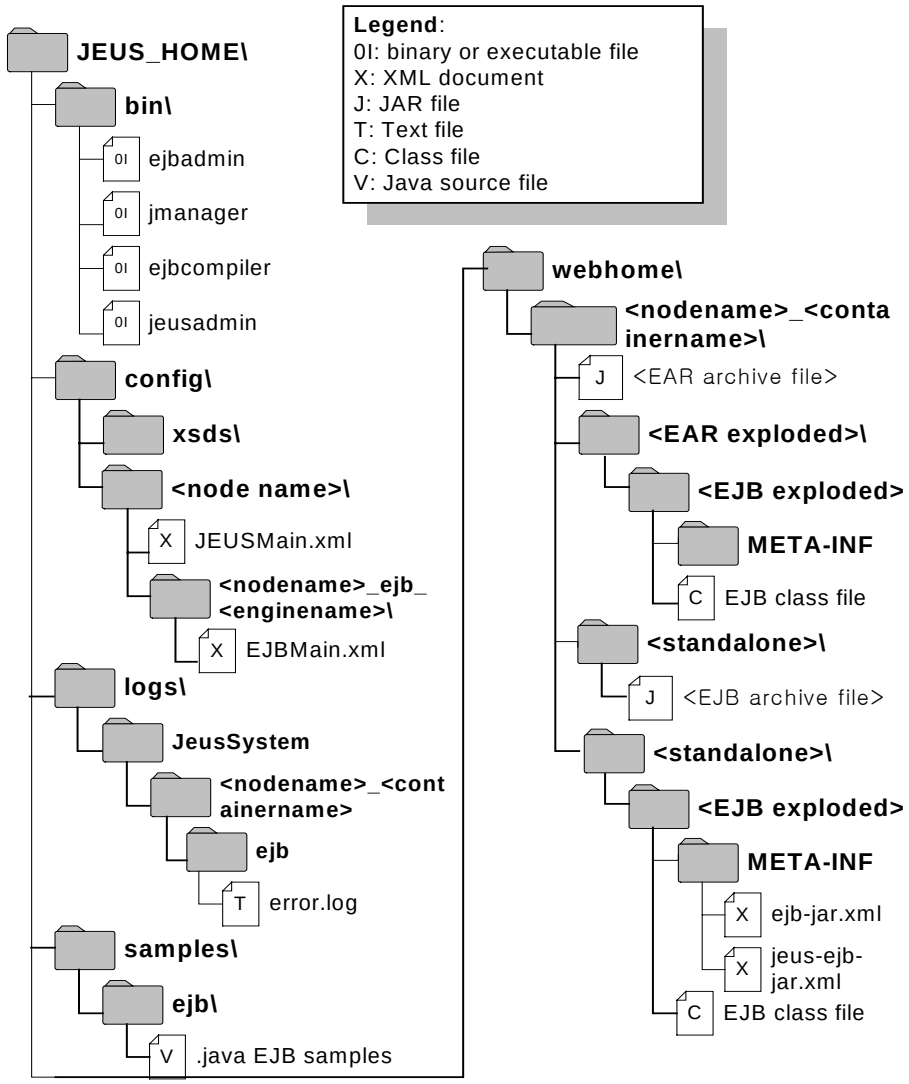


*그림 3 JEUS EJB 관련 디렉토리와 파일*

[그림 3]의 디렉토리들과 파일들은 다음과 같은 목적을 가진다.

- **bin** 디렉토리는 관리자가 EJB를 관리할 때 사용할 수 있는 실행 스크립트들이 담겨져 있다. 여기에는 jeusadmin, ejbadmin, appcompiler 등이 있다. 좀 더 상세한 정보는 다음 절에서 살펴보자.

- **config** 디렉토리는 다음과 같은 하위 디렉토리를 가진다.

- **xsds**\ 디렉토리는 EJB에 관련된 모든 XML schema 파일들이 들어 있다.

- **<node name>**\ 디렉토리는 현재의 JEUS 시스템의 주 설정 디렉토리이다. 이 디렉토리에는 EJB engine을 시스템에 추가할 수 있는 JEUSMain.xml이 포함되어 있다.

- **<node name>_ejb_<engine name>**\ 는 <node name>\ 아래에 위치하고 한 개의 EJB engine을 설정하는 디렉토리이다. 이 디렉토리에는 주 EJB engine 설정파일인 EJBMain.xml 파일이 포함되어 있다.

- **logs\JeusSystem\<node_name>_<containername>\ejb** 디렉토리는 EJB engine에서 발생하는 모든 로그 파일들이 생성되는 곳이다.

- **samples\ejb** 디렉토리는 여러 종류의 EJB들을 구현 해 놓은 예제 코드들과 하위 디렉토리들이 포함되어 있다.

- **webhome\<node_name>_<containername>** 디렉토리는 <node_name>_<containername> 의 container에서 사용하는 application들의 archive file이나 directory가 존재하는 directory이다. application의 archive file이나 directory에는 EJB 구현 클래스들과 helper 클래스들, EJB stub과 skeleton들이 포함되어 있다.

중요: **webhome\<node_name>_<containername>** 디렉토리는 application을 Deploy하는 방법에 따라 archive file이나 directory를 가질 수 있다. 이들의 차이점에 대해서는 JEUS Server 안내서와 이 문서 5장에서 다룬다.

위의 리스트에서 사용된 “<”, “>” 표현(예 "<node name>\”)은 그 사이에 사용된 문자들은 실제 시스템 설정 값으로 대체되어야 한다. 그러므로, 예를 들어, 실제 시스템이 “johan”이라 한다면 JEUS 노드는 “johan”이라고 값이 설정되어야 한다. 그러므로, “<node name>\” 표현은 실제 시스템에서 “johan \” 디렉토리로 대체되어야 한다.

## 3.9 EJB XML 설정 파일

다음 XML 설정 파일들은 JEUS EJB의 관리와 설정에 관련된 것들이다 [표 1].

*표 1.JEUS EJB 의 XML 설정 파일*

| 파일명<br>(DTD 파일명) | 위치 | 목적 | 자세한 설명 |
|---|---|---|---|
| **JEUSMain.xml**<br>(jeus-main.xsd) | JEUS_HOME\config\<br><node name>\ | EJB engine의 등록은 이 파일에서 설정해야 한다. | JEUS Server 안내서와 본 매뉴얼의 4장 |
| **EJBMain.xml**<br>(ejb-main.xsd) | JEUS_HOME\config\<br><nodename>\<nodename>_ejb_<engine name>\ | 주 EJB engine 설정 파일. | 본 매뉴얼의 4장 |
| **ejb-jar.xml** (ejb-jar_2_1.xsd) | META-INF\<br>표준 jar 파일내의 구조 | J2EE 표준 EJB module DD | J2EE EJB 2.1 스펙 |
| **jeus-ejb-dd.xml**<br>(jeus-ejb-dd.xsd) | META-INF\<br>표준 jar 파일내의 구조 | JEUS EJB 모듈을 위한 JEUS DD | 본 매뉴얼의 5장 |

XML schema 파일들은 JEUS_HOME\config\xsds\ 디렉토리에 있다.

위 파일들의 내용은 반드시 표준, 즉 JEUS에서 정의된 XML 헤더로 시작되어야 한다. 또한 root element는 JEUS XML schema의 namespace를 기존 namespace로 지정해야 한다. 각 파일에 사용된 헤더는 다음과 같다.

<<*JEUSMain.xml*>>

```
<?xml version="1.0"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

<<*EJBMain.xml*>>

```
<?xml version="1.0"?>
<ejb-engine xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

<<*jeus-ejb-dd .xml*>>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

*<<ejb-jar.xml >>*

```
<?xml version="1.0"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
```

(위에서 마지막 헤더는 EJB 2.1 스펙에 포함되어 있다.)

위 파일들의 예제가 이 문서에서 사용될 때에는 표준 헤더는 주로 생략되고 있다. 실제 XML 설정파일에는 이 헤더들이 포함되어있다는 것을 기억하기 바란다.

**중요**: 매뉴얼을 통해 모든 태그는 XML 설정 파일에 있는 순서대로 제공된다. 태그 순서는 부록 D와 E에서 확인할 수 있다. 그러나 실제 사용할 때는 항상 그렇게 할 수는 없는 법이다. 실제 JEUS에서는 태그 순서에 관계없이 XML file을 읽을 수 있다.

## 3.10 JEUS EJB 에서 사용된 툴과 유틸리티들

다음 툴들은 EJB 모듈들과 EJB, EJB engine을 관리하기 위하여 사용된다:

- **jeusadmin**: 이 콘솔 툴은 EJB engine을 컨트롤 하기 위해 사용된다. 자세한 내용은JEUS Server 안내서를 참고한다.
- **ejbadmin**: 이 콘솔 툴은 EJB 모듈을 Deploy하고 모니터링하기 위해 사용된다. 상세한 내용은 부록 A를 참고한다.
- **appcompiler**: 이 콘솔 유틸리티는 각 모듈의 stub과 skeleton 클래스들을 작성한다. 상세한 내용은 JEUS 서버 안내서를 참고한다.
- **웹 관리자**: JEUS의 그래픽 관리 툴로서, 위의 콘솔 툴에서 가능한 작업은 모두 할 수 있다. 이 툴에 관해서는 JEUS 웹 관리자 안내서를 통해 자세한 내용을 볼 수 있다.
- 텍스트 편집기 또는 XML 편집기: XML 설정파일은 수작업을 통하여 수정하기 위해 필요하다. 이러한 툴들은 JEUS 제품군에 포함되어 있지 않다.

## 3.11 JEUS EJB 에 관련된 시스템 환경 변수들

[표 2]에서는 사용자가 직접 정의할 수 있는 EJB에 관련된 시스템 환경 변수들이 나열되어 있다. 이 변수들은 JEUS_HOME\bin\ 디렉토리에 있는 다양한 JEUS 스크립트들 안에 "-D" Java Property로 등록된다 (예. jeusadmin, ejbadmin, appcompiler 스크립트들).

*표 2. EJB 시스템의 환경 변수*

| 환경 변수 | 의미 | 예 |
|---|---|---|
| JEUS_HOME | JEUS가 설치된 디렉토리. 이 디렉토리는 시스템이 EJB engine의 설정파일을 읽어 들일 수 있도록 설정되어야 한다. | C:\jeus50 (윈도우 환경 에서는 기본 설치경로가 C 드라이브로 제시된다.) |
| JAVA_HOME | 일부 툴과 스크립트에서 Java 2 SDK 가 사용된다. (appcompiler 스크립트에서 사용된다). | c:\jdk1.4(반드시 설정되어야 한다) |
| JEUS_BASEPORT | JEUS Server에서 사용하는 Port Number | 9736 (반드시 설정되어야 한다. 설정되지 않을 경우 기본값은 9736이다.) |

OS 환경에 따라 어떤 식으로 이 변수들을 설정할지 달라진다. 사용자의 OS 매뉴얼을 참조하여 더 상세한 시스템 환경변수 설정법을 살펴보길 바란다.

**참고**: 대부분의 이 변수들은 JEUS가 설치될 때 자동으로 설정이 된다. 일반적으로 사용할 때에는 그 설정 그대로 사용해도 무방하다.

## 3.12이 매뉴얼 구조에 대한 설명

이 장에서 제공된 정보와 같이 본 매뉴얼의 구성은 [그림 4]와 같다.

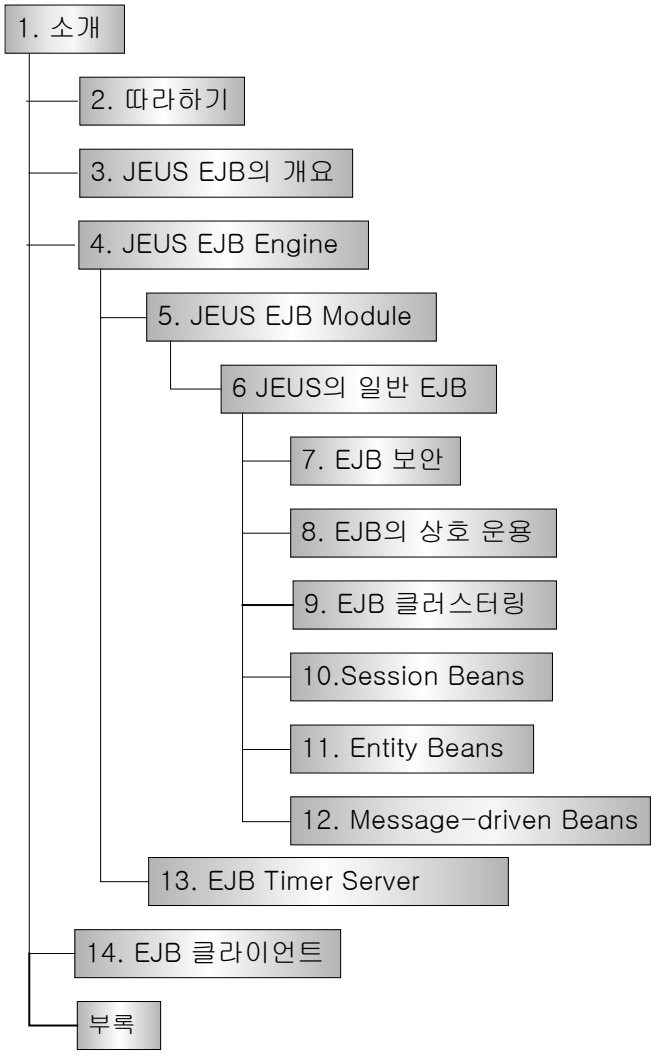


*그림 4 매뉴얼의 계층 구조*

위의 그림은 이 문서의 상 하위 구성을 확연히 보여주고 있다. 우선 맨 위의 EJB engine부터 살펴볼 것이며, 차례로 그 하위 컴포넌트들(EJB 모듈과 EJB)을 살펴볼 것이다.

## 3.13결론

이상으로 JEUS EJB에 대한 일반적인 설명을 마친다.

이 장에서는 JEUS EJB의 주요 컴포넌트들에 대한 것을 설명하였고, 이들이 JEUS 시스템에서 어디에  위치하는지 살펴보았으며, EJB 디렉토리 구조와 EJB 관련된 XML 설정 파일들과 툴들을 살펴보았다.

마지막으로, 앞에서 주어진 정보를 기반으로 하여 어떻게 이 문서가 구성되어 있는지도 살펴보았다.

다음 장에서는 JEUS EJB의 가장 최상위 컴포넌트인 EJB engine에 대해 설명하겠다.

# 4 JEUS EJB engine

## 4.1 소개

JEUS EJB engine은 Enterprise JavaBeans의 운영 환경을 제공하는 주요 소프트웨어 컴포넌트이다. J2EE 스펙에서는 “EJB container” 와 “EJB server”라는 용어가 사용되고 있으며, 이 문서에서 사용될 “EJB engine”과 동일한 개념이다.

“EJB container”에 대한 상세한 정보는 상위 레벨의 백서나 유사 정보를 참조하기 바란다.

앞 장과 마찬가지로, EJB engine에 대한 간략한 개념적 소개를 시작으로 EJB engine의 설정, 컨트롤, 모니터링 그리고 성능튜닝 등을 순서대로 설명한다

EJB모듈, EJB Deploy에 대한 상세한 정보는 5,장과 6장에서 더 다룬다 .

## 4.2 EJB Engine 에 대하여

### 4.2.1 소개

[그림 5]은 EJB engine과 이의 하위 컴포넌트들을 도식화한 것이다. 이 절에서는 그림의 상위 레벨에 그려진 항목들(active management, logging, EJB engine의 주요 설정 )만 살펴보겠다. 다음 절에서는 EJB 모듈과 EJB에 대하여 설명한다

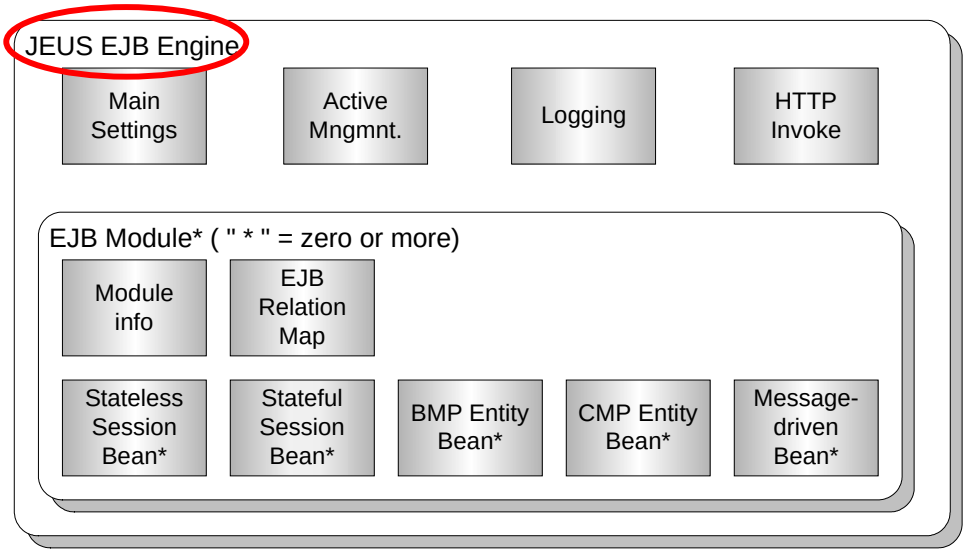


*그림 5 EJB engine 과 서브 컴포넌트*

### 4.2.2 EJB engine 과 EJB 시스템

JEUS EJB engine은 JEUS의 engine container 내에서 설정되고 실행된다(JEUS Server 안내서 참조). 그리고, 각각의 EJB engine들은 각각의 JVM에서 실행된다. 즉, 한 container당 하나의 JVM을 뜻한다. 단 한 개의 EJB engine만이 각 engine container에 존재할 수 있기 때문에 각 EJB engine은 별도의 유일한 JVM안에서 운영된다고 해도 무리가 없을 것이다.

일반적으로 여러 개의 머신이나 CPU 위에  여러 개의 engine container와 EJB engine이 설정되어 실행된다. 이러한 설정을 engine의 클러스터링이라고 부른다. 이는 시스템의 성능향상과 높은 안정성 및 보안성을 유지시켜주기 때문에 효율적인 시스템 구조라고 할 수 있다.

뒤에서 보겠지만 JEUS EJB에서 클러스터링은 EJB engine 레벨에서 간접적으로만 구성된다. 우리가 실제적으로 JEUS에서 클러스터링 할 수 있는 것은 EJB 들 뿐이다.

EJB 클러스터링에 대한 것은 9장에서 자세히 다룬다.

### 4.2.3 EJB engine 기본 설정 및 기능

EJBMain.xml(EJB의 주 설정파일)에는 기본적인 설정 기능이 여러 가지가 있다. 무엇보다도, 이 설정에는 engine의 resolution time-period 등의 여러가지 설정이 있다.

이 모든 기본적인 설정들은 다음 절에서 자세하게 다룰 것이다(4.3절).

다음 절들에서는 EJB engine의 중요한 설정 요소인, Logging, Active Management에 대해서 알아본다.

### 4.2.4 EJB Engine Logging

EJB engine이 생성할 수 있는 로그에는 두 가지 종류가 있다.

- 시스템 로그
- 사용자 로그

시스템 로그

시스템 로그는 EJB engine에서 발생하는 대부분의 비정상적 이벤트들을 로깅한다. 이 로그에는 시스템 차원의 예외상황과 문제 뿐만 아니라 어플리케이션 상의 예외상황도 포함된다. (예, EJB 코드 내에서 발생한 예외상황)

logger의 handler에 file-handler가 지정되면 시스템 로그는 JEUS_HOME\logs\JeusSystem\<node name>_<containername>\ejb 아래에 파일로 남게 되고, 실제 파일은 error_yyyymmdd.log의 포맷으로 남게 된다. 여기서 “yyyymmdd”는 로그 파일이 생성된 날짜를 나타낸다. EJB engine이 서비스를 제공하는 동안 날짜 별로 다른 로그 파일이 생성되는 것이다.

뿐만 아니라, console-handler를 사용하면 모든 로그 메시지를 화면에 남길 수도 있다. 일반적으로 이런 상황에서는 로그 메시지가 파이프를 통하여 JEUS manager가 시작된 터미널 창에 출력된다.

그 외에 user가 만든 user-handler를 등록할 수 있다. 자세한 설명은 JEUS Server 안내서의 logging 장을 참조하기 바란다.

사용자 로그

사용자 로그는 시스템 로그의 서브 셋 메시지들을 로그로 생성하고, 추가로 logging API를 사용하여 프로그래머가 남기게 한 메시지들을 로그로 생성한다. JEUS Server 안내서에 더 자세한 JEUS 특정 사용자 로깅 API정보가 설명되어 있다.

이 두가지 logging의 설정은 JEUSMain.xml의 <engine-command>와 <enginecontainer> 에 정의된다. 더 자세한 정보는 JEUS Server 안내서에 설명되어 있다.

### 4.2.5 Active Management

Active management의 의미는 EJB 모듈에 문제가 발생한 경우에 EJB engine이 전자우편으로 자체적으로 통지를 보내주는 것을 의미한다. 이런 문제의 한 예로는, EJB 클래스의 버그로 인하여 EJB 의 Thread block을 오랫동안 유지한 채 시스템 리소스를 낭비하고 시스템 성능을 저하시키고 있는 경우를 들 수 있다.

JEUS EJB에서는 오류 정책(또는 조건)을 설정할 수가 있어서 조건에 맞으면 그 비정상적인 현상에 대한 전자우편을 보내 통지하고 EJB engine을 자동으로 재시작 할 수 있도록 하고 있다.

실제적인 Active Management에 대한 자세한 설정은 다음 절(4.3)에서 설명된다.

### 4.2.6 HTTP Invoke

클라이언트가 JNDI 서비스를 통해서 EJB인스턴스를 찾을 때 클라이언트는 원격으로 EJB 메소드를 호출 할 수 있는 EJB RMI 스텁을 받는다. 기본적으로, 스텁과 EJB간의 원격통신은 스텁으로부터 RMI runtime으로 java.net.Socket 이 연결되어 이루어진다.

이 연결은 firewall이 중간에 있을 경우와 직접 RMI runtime port로 접근 할 수 없는 환경에선 문제가 될 수 있다. 이 경우 특별한 통신 모드가 필요한데 이것이 *HTTP invocation mode*이다.

이 모드를 사용할 경우 클라이언트가 stub을 통해 메소드를 호출할 때 RMI 요청을 HTTP로 포장해서 웹서버로 보내고 웹서버는 RMI handler servlet(jeus.rmi.http.ServletHandler)으로 요청을 전달 한다. handler servlet은 HTTP 헤더를 제거하고 RMI runtime으로 요청을 전달 한다.

**참고**: RMI runtime은 반드시 웹서버와 Handler Servlet이 운영되는 Web container와 같은 머신에 있어야 된다.

요청에 대한 결과도 HTTP 응답에 포함되어 스텁에 전달된다.

HTTP invoke 모드는 방화벽을 넘어 EJB를 호출하기 위해 HTTP를 사용하는 것이다.

HTTP invoke 모드는 두 곳에서 설정할 수 있다:

- EJBMain.xml

- EJB module 의 jeus-ejb-dd.xml.

EJBMain.xml에 HTTP invoke모드가 설정 되면 EJB engine 내의 모든 모듈에 적용된다. EJB DD파일에 세팅되면, 세팅된 특정 EJB에만 적용 된다.(EJB DD가 EJBMain.xml에 앞선다)

더 상세한 내용은 6장에서 설명한다.

### 4.2.7 EJB engine 의 디렉토리들과 파일들

[그림 6] 은 EJB engine을 관리할 때 사용하게 되는 디렉토리와 파일들을 보여준다

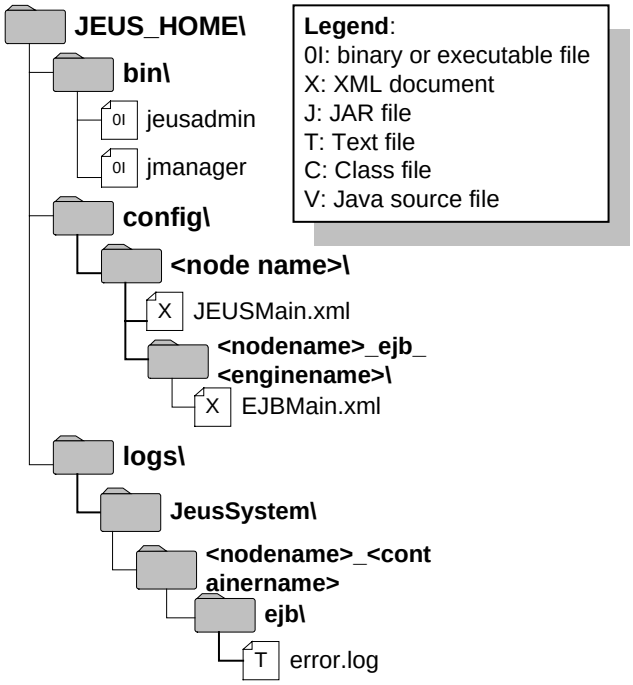


*그림 6 EJB engine 관련 디렉토리와 파일들*

위의 그림에서 다음을 주목하기 바란다.

- **EJBMain.xml:** 이 파일은JEUS의 EJB engine의 주 설정 파일이다.
- **EJB engine 로그 파일:** 이 파일은 다음과 같은 포맷으로 명명된다. error_yyyymmdd.log, 여기서 “yyyymmdd” 는 로그 파일이 생성된 날짜를 의미한다.
- **jeusadmin** 과 **웹 관리자는** EJB engine을 컨트롤하고 모니터링 하기 위하여 사용된다.
- **JEUSMain.xml** 파일은 JEUS시스템에 새로운 EJB engine을 추가하고 싶을 때 사용된다.

위 파일들과 디렉토리에 대한 상세한 정보는 이 매뉴얼의 3장을 참고한다.

### 4.2.8 결론

이것으로 JEUS EJB에 대한 포괄적인 설명을 마무리한다.

우리는 EJB engine의 주요 설정 컴포넌트들을 살펴보았다(주요설정, engine 로깅, Active Management).

그리고, 마지막으로 JEUS EJB engine에서 사용하는 디렉토리 구조를 살펴보았다.

이제 EJB engine의 설정에 관련된 상세한 사항들을 한 번 알아보자.

## 4.3 EJB engine 설정

### 4.3.1 소개

JEUS 시스템의 EJB engine을 추가하고 설정하기 위해서는 두 개의 파일에 설정 데이터를 입력해야 한다.

- **JEUSMain.xml 파일**: JEUS의 주 설정 파일로서 JEUS_HOME\config\<nodename>에 위치하고 있다. 이 파일에는 몇 가지의 태그만 추가해 주면 된다.

- **EJBMain.xml파일:** JEUS_HOME\config\<nodename>\<nodename>_ejb_<enginename> 디렉토리 아래에 위치한다. 이 파일은 EJB engine의 실제 정보를 모두 담고 있지만 JEUSMain.xml 파일에서는 단순히 EJB engine의 존재여부와 logging 설정 정도를 선언하고 있다.

다음 두 절에서 두 파일에 대한 필수설정과 옵션설정에 대해 알아 보겠다.

### 4.3.2 JEUSMain.xml 에 EJB engine 추가

JEUSMain.xml에 EJB engine을 추가하는 방법은 JEUS Server 안내서에 잘 설명되어 있다. 여기서는 EJB engine을 추가하는 간단한 예를 보여준다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    . . .
    <node>
                . . .
        <engine-container>
```

```
            . . .
            <engine-command>
                <type>ejb</type>
                <name>engine1</name>
                <system-logging>
                  <handler>
                    <console-handler>
                      <name>consoleHandler</name>
                    </console-handler>
                  </handler>
                </system-logging>
            </engine-command>
                  . . .
        </engine-container>
         . . .
    </node>
    . . .
</jeus-system>
```

위의 예는 “engine1”이라는 EJB engine을 추가하고 있다. 다른 태그에 대한 정보는 JEUS Server 안내서를, 웹 관리자를 사용하여 engine을 설정하는 방법에 대해서는 JEUS 웹 관리자 안내서를 참조하라.

그리고, 필수 XML 헤더가 위에 생략되어 있음을 알린다. 3장에 포함되어야 할 헤더의 정보들이 설명되어 있다.

위의 <engine-command>태그를 JEUS 부트 시 정상적으로 작동하게 하려면, JEUS_HOME\config\<nodename>\<nodename>_ejb_<enginename> 디렉토리가 반드시 존재해야 한다. 여기서 “nodename”은 JEUS의 노드 이름이고, 이는 로컬 머신의 호스트 이름을 의미한다. “enginename”은 JEUSMain.xml의 <engine-command> 태그 중 <name> 태그의 값을 가지고 있어야 한다. 위의 예에서 호스트 이름이 “johan”이라고 가정하면, engine 이름은 “johan_ejb_engine1”이 된다.

JEUS_HOME\config\<nodename>\<nodename>_ejb_<enginename>의 아래 에는 반드시 EJBMain.xml이 존재해야 한다. 이 파일은 다음 절에서 설명한다.

**중요**: Engine Container에서는 EJB engine을 하나만 가질 수 있다.

### 4.3.3 EJBMain.xml 의 설정

EJBMain.xml은 EJB engine의 모든 운영 설정들을 포함하고 있으며, engine의 홈 디렉토리에 들어 있다("engine 디렉토리"라고 일컫는다). 앞에서 설명 했듯이 이 디렉토리는 다음과 같은 경로를 가져야 한다.

JEUS_HOME\config\<nodename>\<nodename>_ejb_<enginename>

경로 포맷에 대한 정보에 대해서는 앞 절에서 설명하였다.

EJBEngine.xml 파일에는 <ejb-engine> 태그 아래에 다음과 같은 EJB engine 설정들이 포함되어 있다.

- **Resolution**. 여기에 설정된 시간의 간격마다 Passivation 과 garbage collection 이 수행된다. EJB engine의 bean이 더 이상 클라이언트로부터 요청을 받지 않을 경우, engine은 그 bean을 비 활성화 시킨다. 추가적으로, engine은 주기적으로 사용하지 않는 자원을 정리하기 위하여 garbage collection을 수행한다. 여기에 설정된 값은 millisecond 단위로 이 값이 해당 작업의 주기가 된다. 그러므로, 이 태그는 EJB engine의 "Heart Beat Rate"라고 할 수 있겠다.

- **WS (FTP) port**. EJB 에서는, stub 파일과 같은 파일들은 서버와 클라이언트 또는 서버들 사이에서 옮겨 다닌다. 이러한 파일들의 전송을 위하여 FTP 서버가 필요하다. JEUS는 이런 목적을 위하여FTP 서버를 포함하고 있다. 이 FTP 서버는 "Class FTP" 서버라 불린다. WSPort 태그는 FTP 서버의 포트 번호를 일컫는다. 디폴트로 JEUS 는 Class FTP 서버를 위하여 9739 포트를 이용한다

- **user notification enabling**. 이 옵션을 설정하면 EJB exception이 JEUSMain.xml에서 설정한 user log에 남게 된다. (자세한 것은JEUS Server 안내서를 참조한다).

- **Active management**. EJB engine을 모니터링하고, 오류를 처리하며 전자우편 통지를 보내는데 관련된 설정을 한다(다음의 하위 절 참조).

- **HTTP invoke** (나중에 설명한다).

- **Timer service** (13장에서 설명한다)

다음은 EJBMain.xml 의 설정 예이다.

*<<EJBMain.xml>>*

```
<ejb-engine>
    <resolution>100000</resolution>
    <ws-port>9988</ws-port>
    <enable-user-notify>true</enable-user-notify>
    <module-list>accountejb</module-list>
    <active-management>
        . . .
    </active-management>
    <invoke-http>
        . . .
    </invoke-http>
</ejb-engine>
```

다음은 EJB engine의 주요 하위 컴포넌트들을 어떻게 설정하는지에 대하여 설명한다.

### 4.3.4 System Logging 설정

EJB engine의 **system logging과 user logging**은 JEUSMain.xml에서 설정된다. 이 설정은 EJB engine 뿐만 아니라 다른 모든 engine에게도 적용되는 공통적인 설정이므로 Jeus Server 매뉴얼을 참조하기 바란다.

### 4.3.5 Active Management 설정

Active Management 설정은 두 부분으로 나뉘어져 있다. engine 재시작 조건들과 전자우편 통보 기능이 그것이다. engine 재시작 조건은 EJB engine이 재시작 하기 전까지의 허용 가능한 최대 블록된 EJB thread 수로 결정된다.

설정 사항으로는 다음과 같은 것이 있다.

- **max blocked thread**. EJB engine이 재시작 하기 전까지 허용할 수 있는 블록된 EJB thread의 최대 개수이다.

- **max idle time.** 지정된 시간 동안 thread 가 block 된 채 요청을 받지 않고 idle 상태에 있으면 “블록된 thread 리스트”로 추가된다. 이 설정은 engine에서 block 된 thread로 판단하는 기준이 된다. 단위는 millisecond이다.

- **email notification.** 재시작 조건에 따라 EJB engine이 시작될 경우에 어디로 전자우편 통보를 보내야 할 지 설정할 수 있는 전자우편 통보

기능이 있다. 이 설정은 jeus-common.xsd file의 smtp-handlerType을 따른다. 주요 설정들은 아래와 같다:

- **SMTP host address.** 메시지를 보낼 때 사용할 SMTP의 주소. 이 주소는 호스트 이름이나 IP 주소로 설정되어 있어야 한다.
- 메일 송수신자의 메일 주소인 to, from, cc, bcc가 있다.

EJBMain.xml 의 Active Management 설정의 예는 다음과 같다.

*<<EJBMain.xml>>*

```
<ejb-engine>
    . . .
    <active-management>
        <max-blocked-thread>200</max-blocked-thread>
        <max-idle-time>180000</max-idle-time>
        <email-notify>
            <name>activeHandler</name>
            <smtp-host-address>mail.foo.com</smtp-host-address>
            <from-address>jeus@foo.com</from-address>
            <to-address>admin@foo.com;admin2@foo.com</to-address>
            <cc-address>admin@bar.com</cc-address>
            <bcc-address>admin2@foo.com</bcc-address>
        </email-notify>
    </active-management>
    . . .
</ejb-engine>
```

### 4.3.6 HTTP Invoke 환경 설정

EJB engine에서 HTTP invocation 이 필요하면 EJBMain.xml에서 <invoke-http>를 추가한다. EJBMain.xml의 설정은 모든 모듈에 적용되는데, 각각의 모듈의 DD파일에 <invoke-http>가 있다면 이것이 EJBMain.xml을 의 설정을 무시하게 된다.

두 개의 하위 태그가 있다.

- **URL**: 반드시 HTTP-RMI스텁으로부터 호출되는 RMI handler Servlet 의 URI (jeus.rmi.http.ServletHandler)를 입력한다. 이 URI에서는 프로토콜, IP, 포트를 제외한 Servlet 요청 경로만을 넣는다. 프로토콜은 “HTTP” 로 IP는 RMI runtime과 같은 주소로 간주된다. 이 말은 HTTP-RMI요

청을 받은 웹 서버와 Web container가 RMI runtime과 같은 머신에 있어야 된다는 것이다. 그러면 RMI runtime 의 주소는 RMI 스텁에게 알려지게 된다. 웹서버의 포트는 반드시 다음 "HTTP port"에 설정 해야 한다.

- **HTTP port**: HTTP-RMI요청을 받고 처리할 웹 서버 또는 Web Container를 설정해야 한다. 이 웹서버/Web container 는 반드시 RMI handler Servlet이 Deploy되어 실행되고 있어야 한다.

예)

<<*EJBMain.xml*>>

```
</ejb-engine>
    . . .
    <invoke-http>
        <url>
            /mycontext/RMIHandlerServlet
        </url>
        <http-port>
            80
        </http-port>
    </invoke-http>
</ejb-engine>
```

HTTP invoke가 정상적으로 작동하기 위해선 RMI handler Servlet이 Deploy 되어 있어야 한다. RMI handler Servlet을 구현한 클래스는 jeus.rmi.http.ServletHandler이다. jeus.rmi.http.ServletHandler은 context내에 <invoke-http>에 설정된 URL과 같게 Deploy 되어야 한다. Deploy에 대한 것은 JEUS Web Container 안내서를 참조 한다.

### 4.3.7 –D 설정

EJB engine의 XML 설정의 추가로, jeus.proeprties 파일 (또는 JEUSMain.xml 의 `<engine-container><command-option>` 태그) 의 JAVA_ARGS에 -D설정을 추가하여 System Property를 설정할 수 있다.

이 설정에 대해서는 부록 B를 참조하라.

### 4.3.8 결론

이 장에서는 JEUS EJB engine 설정에 관련된 전반적인 것을 설명하였다. 어떻게 JEUSMain.xml에 EJB engine을 추가하며 EJBMain.xml을 설정하는지 등을 살펴보았다.

다음 절에서는 설정된 EJB engine을 컨트롤 하고 운영하는 방법에 대해 살펴보겠다.

## 4.4 EJB engine 관리

### 4.4.1 소개

EJB engine을 제어하는 것은 다른 JEUS engine(Servlet 또는 JMS engine)을 제어하는 것과 많은 유사점을 가진다. jeusadmin콘솔 툴을 이용하던가 웹 관리자를 사용할 수도 있다. 이 툴들을 어떻게 사용하는지에 대해서는 JEUS Server 안내서의 “Engine 장”에 자세히 다루고 있다

여기서는 콘솔 툴의 사용 방법만 간단하게 설명한다.

### 4.4.2 콘솔 사용

여기서는 jeusadmin 콘솔 툴을 이용하여 EJB engine을 컨트롤 하는 방법에 대한 몇 가지 간단한 예를 들어보겠다. 아래의 예에서는 JEUSMain.xml 과 EJBMain.xml이 johan 이라는 이름의 시스템에서 작성되었다고 가정한다.

먼저, jeusadmin 을 시작하고 JEUS 노드에 연결시킨다( 이 예에서 노드의 이름은 “johan”이다).

```
c:\> jeusadmin johan
```

그 다음으로 jeusadmin 프롬프트가 나타나면 사용자이름과 암호를 입력한다.

만약에 JEUS 노드가 부트되지 않았으면 지금 다음과 같이 한다.

```
johan> boot
```

EJB engine을 포함한 모든 설정된 engine들이 자동적으로 시작된다.

“johan_ejb_engine1”이라는 EJB engine이 이미 기동되었다고 가정하면 다음과 같은 명령어로 정지 시킬 수 있다.

```
johan> downeng johan_ejb_engine1
```

EJB engine이 정지되었다.

다시 정지된 EJB engine을 시작하려면

```
johan> starteng johan_ejb_engine1
```

EJB engine이 시작된다.

위 명령들은 EJB engine과 연관된 아주 간단한 명령어이다. JEUS Server 안내서를 보면 모든 명령어 정보들을 얻을 수 있다.

**참고**: EJB engine 자체를 관리하려면 ejbadmin이 아닌 jeusadmin을 사용해야 한다.

### 4.4.3 결론

jeusadmin을 사용하여 어떻게 EJB engine을 컨트롤(start, down) 하는지 살펴보았다.

## 4.5 EJB Engines 모니터링

### 4.5.1 소개

모니터링이란 특정 EJB engine의 실행환경의 정보와 상태 정보를 수집하는 것을 뜻한다. 이번 절에서 이 주제를 다룬다.

**참고**: 웹 관리자를 사용해서 EJB engine을 모니터링할 것을 추천한다. 이유는 웹 관리자가 console tool에 비해 자세하고 완벽한 engine 상태 정보를 제공하기 때문이다. 웹 관리자에 대한 것은 JEUS 웹 관리자 안내서를 참조 하기 바란다.

### 4.5.2 Console 사용하기

EJB engine 정보를 보기 위한 명령어는 “jeusadmin”과 “ejbadmin”두 가지가 있다. 이들을 통해 기본적인 실행환경 정보를 얻을 수 있다. JEUS Server 안내서와 부록 A를 보면 더 상세한 정보를 얻을 수 있다.

### 4.5.3 결론

이 절에선 EJB engine을 모니터링하기 위한 핵심적인 내용을 설명했다. 다음 절에선 EJB engine의 최적화를 위한 튜닝에 대해 설명한다.

# 4.6 EJB Engine 튜닝

## 4.6.1 소개

각의 JEUS EJB engine들의 전체적인 성능향상을 위해서 설정을 변경할 때가 있다.

이 절에서는 EJB engine(EJBMain.xml에 설정된)의 성능관련 설정을 간략하게 살펴보겠다.

참고: EJBMain.xml의 정보나 팁에 대해서는 부록 D 에서 참조할 수 있다. 부록 D 에 적힌 XML/DTD중 "P"라고 적힌 것은 성능과 관련된 것이다.

다음은 튜닝을 위해서 필요한 사항들이다.

- Resolution 설정 튜닝
- Fast-deploy 기능 사용
- 최적성능을 위해 시스템 로그 설정
- active management 사용하지 않기
- HTTP invoke mode사용

## 4.6.2 Resolution 설정 튜닝

Resolution은 EJB engine의 "심장박동"이라고 할 수 있다. Resolution 에 설정된 시간이 지나면 EJB engine은 모든 하위 컴포넌트들(예, bean pools)을 점검하고, bean 을 비활성화 하고, garbage collection을 수행한다.

이 값을 변경함으로써 생기는 영향은 매우 중대한 수준이고, 따라서 바르게 지정되어야 한다.

이 값이 클수록 시스템 메모리나 기타 리소스의 회수 주기가 길어져 자원 활용률은 떨어지지만 이에 대한 작업 수행이 덜 일어나므로 engine의 성능은 좋아진다. 이 값을 작게 하면 engine은 최신 상태를 유기 하겠지만 전체적인 성능은 내려 간다.

## 4.6.3 Fast Deploy Option 사용

Fast deploy option은 EJBMain.xml에서 설정하는게 아니라 application별로 설정하는 것이지만 성능에 큰 영향을 미치는 것이기에 여기서 설명한다. engine 부팅 시에 Deploy되어야 할 EJB 모듈들이 이미 컴파일되어 RMI stub

과 skeleton이 있다면, application의 jeus-ejb-dd.xml이나 jeus-application-dd.xml에서 fast-deploy 옵션을 “true”로 설정 해야 한다. 이는 engine이 EJB 모듈 Deploy시에 RMI 클래스를 생성하지 않도록 하는 것이다.

EJB 모듈과 Deploy에 관련된 보다 자세한 정보는 5장을 보기 바란다.

### 4.6.4 최대성능을 위한 시스템 로그 설정

시스템 로그는 성능 개선을 위해 3가지 방법으로 조정 가능하다.

- Logger의 handler를 설정하지 않으면 기본적으로 console handler가 사용된다. 가능하면 file handler를 사용해서 logging이 빠르게 이루어지도록 하는게 좋다.
- File handler의 버퍼 크기를 크게 설정하라.
- 로그 레벨을 “SEVERE”로 설정하라.

물론 위의 제안은 안정적인 운영환경에서만 적용되어야 한다. 개발 환경에서는 “반대”의 값들이 설정되어야 한다(즉, console handler를 사용하고, 작은 버퍼 크기를 사용하고, “FINE”로그 레벨을 사용하는 게 개발을 편이하게 해준다).

### 4.6.5 Active Management 사용하지 않기

EJB engine 레벨에 Active Management 를 사용하는 것이 꼭 필요한 것은 아니고, 어떤 경우에는 성능에 저하를 가져올 수 있다. 대신에, Servlet engine에 정의된 Active Management에 의지하는 것이 더 낫다(JEUS Web Container 안내서 참고).

그러므로, 대부분의 경우 EJBMain.xml에서 Active Management 설정을 생략하는 경우가 많다.

### 4.6.6 HTTP Invoke Mode 사용

HTTP invoke mode 를 사용하면  성능을 향상 시킬 수 있다.

### 4.6.7 결론

이것으로 설명을 마무리한다. 더 자세한 정보는 부록 D 와 다음 장을 참고한다.

Engine Container의 튜닝과 같은 상위 개념에 대한 설명은 JEUS Server 안내서에서 자세히 설명하고 있다.

## 4.7 결론

지금까지 EJB engine에 대한 기초적인 사항과 JEUS EJB의 최상위 레벨의 개념인 구조, 설정, 운영, 모니터링과 튜닝에 대해 살펴보았다.

다음 장은 각각의 Enterprise JavaBeans의 Deploy 단위인 EJB 모듈에 대한 중요한 개념을 설명한다.

# 5 JEUS EJB Module

## 5.1 소개

EJB 모듈은 Enterprise JavaBeans들을 표현하고 그룹화 하기 위한 개념이다. EJB 모듈은 JEUS EJB engine에 Deploy 할 수 있는 가장 작은 단위를 말한다. 그러므로, 한 개의 Enterprise JavaBeans이 Deploy 된다 하더라도, 한 개의 EJB 모듈로 패키지화 되어야 한다.

이 장은 JEUS 안에서 EJB 모듈을 패키징, Deploy, 컨트롤, 모니터링 및 관리하기 위하여 알아야 할 모든 것들을 설명하고 있다. 이 장은 JEUS에서 EJB Deploy를 어떻게 하는지 그 기초를 설명하는 매우 중요한 부분이다.

**참고**: 이 장은 EJB 모듈에 대해서만 설명하고 있다. EJB 모듈 내에서 Enterprise JavaBeans를 설정하는 실제적인 설명은 10장부터 12까지에서 잘 설명되어 있다.

## 5.2 EJB 모듈의 개념

### 5.2.1 소개

[그림 7] 는 JEUS EJB engine상에서 EJB 모듈이 개념적으로 어떤 위치에 있는지를 나타낸다. 여러 개의 EJB 모듈들이 EJB engine에 존재할 수 있으며, 각 모듈들은 EJB 클라이언트가 사용하는 Enterprise JavaBeans들을 담고 있다.

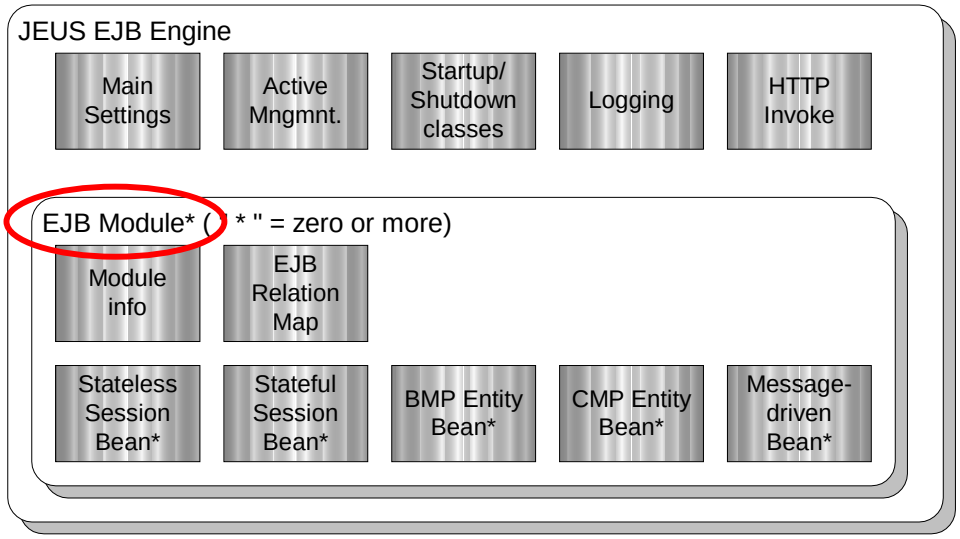


*그림 7 JEUS EJB system 과 관련된 EJB module*

이 장은 EJB 모듈의 하위 모듈들과 주요 설정기능들에 대한 설명을 하고, EJB 모듈의 조립, Deploy, 컨트롤과 모니터링에 대한 개념을 소개한다. 더 자세한 정보는 이 장의 후반부를 참조해야 한다.

### 5.2.2 Module Info 설정

위의 [그림 7] 에서는 “Module info” 블록이 있다. 그 블록은 전체적인 JEUS EJB 모듈 설정을 표현하고 있다. 이 설정들과 다음에 나올 두 하위 절에서 설명하고 있는 설정들은 모두 JEUS EJB 모듈 deployment descriptor에서 설정한다.

모듈 정보 설정은 EJB 모듈 내의 보안 역할 매핑을 결정한다. 상세한 이 변수들의 설정은 5.3절을 참조하라.

### 5.2.3 EJB Relation Map

EJB relation map은 EJB CMP 2.0의 Relation을 정의하기 위해 사용된다. 이 태그가 여러 개의 EJB들에 연관되어 영향을 미치므로 모듈 레벨에서 정의된다.

Relation Mapping에 대한 자세한 설명은 Entity EJB 에서 자세히 다룬다 (11장).

### 5.2.4 Enterprise JavaBeans

어떤 EJB 모듈이든 가장 중요한 부분은 Enterprise JavaBeans 그 자체이다. JEUS에서는 bean들이 EJB 1.1이나 EJB 2.0, EJB 2.1 스펙에 준하여 작성된 것이어야 한다. Enterprise JavaBeans는 EJB 클라이언트들이 이용할 실제 업무 로직과 method들을 구현해야 한다.

EJB 모듈의 모든 Enterprise JavaBeans들은 표준 J2EE EJB DD와 JEUS 특정 EJB 모듈 DD, 이 두 개의 deployment descriptor에 의해 설정된다.

다양한 Enterprise JavaBenas의 설정 방법은 10장부터 시작된다

### 5.2.5 JEUS 내에서의 EJB 모듈 관리

[그림 8] 은 JEUS 내부에서 EJB 모듈을 관리하기 위한 4가지 주요 작업들을 소개한다

- assembly
- deployment
- control (undeploy, reload, suspend, resume)
- monitoring

5.3, 5.4, 5.5, 5.6절에서 이 과정들에 대한 상세한 정보가 설명되어 있다.

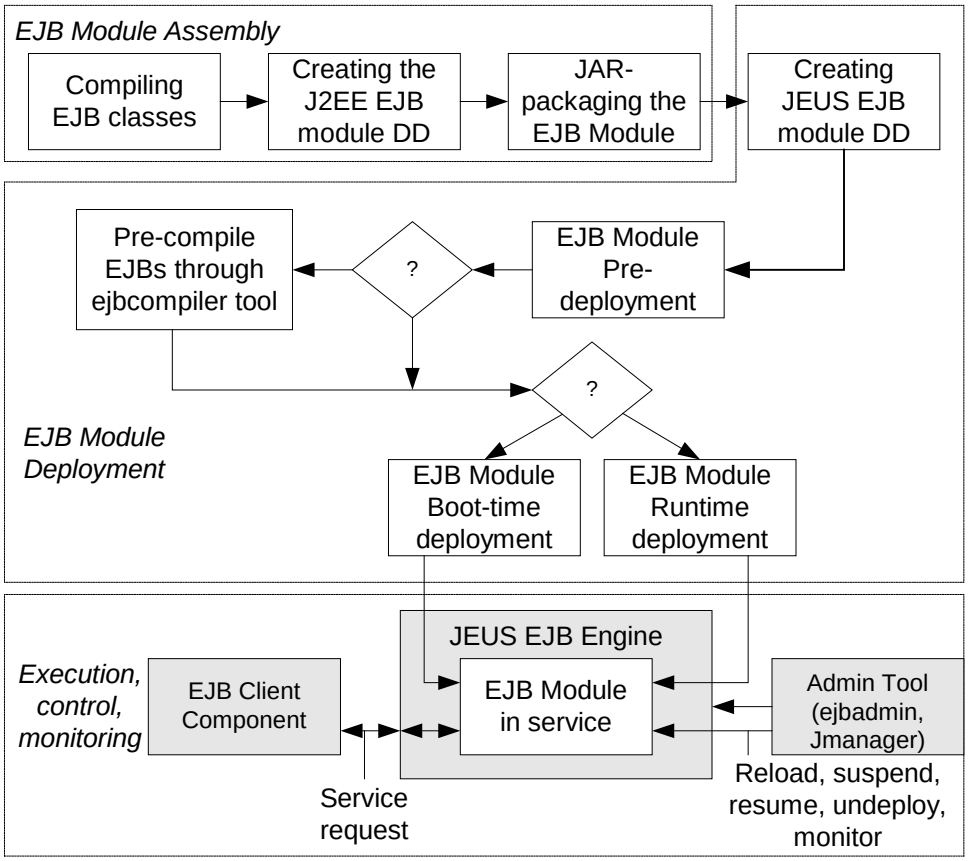


*그림 8 JEUS EJB engine 에서 J2EE EJB module deploy 순서도*

.

다음 절들에서는 위의 그림에서 도식화된 것처럼 EJB 모듈의 관리와 관련된 주요 컴포넌트들, 소프트웨어 모듈 그리고 절차들을 설명한다. 우리가 살펴볼 것들은 아래와 같다.

- 표준 J2EE EJB 모듈 JAR 파일, 그 내용과 구조
- 두 가지의 JEUS 특정 DD 파일들
- 세 가지 Deploy 모드 (pre-deploy, boot-time deploy, runtime deploy)
- appcompiler 사용하기
- pre-deployed EJB 모듈과 연관된 디렉토리 구조
- Deploy된 EJB 모듈을 컨트롤링하기
- 마지막으로, Deploy된 EJB 모듈을  리로딩 하는 것에 대한 간단한 설명

### 5.2.6 J2EE EJB JAR 파일과 그 내용

아래의 [그림 9] 에서는 예상할 수 있는 표준 J2EE EJB 모듈 JAR 파일의 내용을 볼 수 있다.

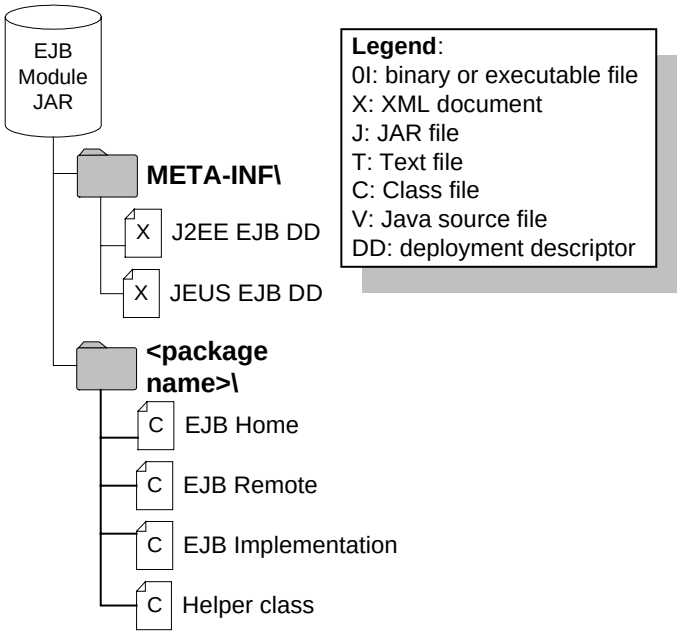


*그림 9 J2EE EJB module JAR 파일의 구조와 내용*

.

EJB JAR의 내용은 다음과 같다:

- **package directory**(들)은 EJB 클래스들을 포함하고 있다. EJB 소스 코드에 정의되어 있는 것과 같이 Java 패키지 구조를 반영하는 구조로 되어 있어야한다. 예를 들어 한 EJB 가 “mypackage”라는 패키지에 속해 있다고 선언되어 있으면 EJB의 클래스는 반드시 EJB JAR의 “mypackage”라는 디렉토리 아래에 위치해야 한다.

- **EJB Implementation**은 프로그래머가 실제 구현한 EJB이다.

- **EJB remote** (or local) **interface** 는 클라이언트가 bean으로 (remote로 또는 local로) 접근하기 위한 인터페이스이다.

- **EJB home interface**는 클라이언트가 bean의 레퍼런스를 얻기 위한 (때로는 새로운 bean을 생성하기 위한) 인터페이스이다.

- **EJB helper classes**는 bean이 사용하는 라이브러리 클래스들을 말한다.

- **J2EE EJB DD** 는 실제적인 표준 EJB 모듈 deployment descriptor이다. 다음 절을 참고한다.

- **JEUS EJB DD** 는 JEUS에서 EJB를 deploy할 때 필요한 EJB 모듈 deployment descriptor이다. 다음 절을 참고한다.

### 5.2.7 두 개의 Deployment Descriptor 들

EJB 모듈을 Deploy하기 전에 두 개의 DD가 준비되어야 한다.

- **J2EE 표준 EJB 모듈 DD**: 이 파일은 “ejb-jar.xml”로 명명되고, EJB모듈의 \META-INF\ 디렉토리 안에 위치한다. J2EE EJB DD의 내용에 대한 정보는 Sun사의 EJB 2.1 스펙을 참고하라.

- **JEUS 특정 EJB 모듈 DD**: 이 파일은 “jeus-ejb-dd.xml” 이고EJB모듈의 \META-INF\ 디렉토리 안에 위치한다. 이 파일의 내용에 대한 정보는 이 장의 5.3절을 참조하라.

### 5.2.8 Pre-deployment, Boot-time Deployment 그리고 Runtime Deployment

[그림 8] 에서 봤듯이 JEUS에서는 세 가지 방법으로 EJB 모듈을 Deploy 할 수 있다.

- **Pre-deployment**: 이 Deploy 에서는 모듈이 실제로 EJB engine에 Deploy 되기 전의 준비과정이 실행된다. Pre-deployment는 EJB engine이 멈추고 모듈이 Undeploy 되더라도 모든 설정들이 JEUS 시스템에 영구적으로 저장됨을 의미한다.

- **Boot-time deployment**: 이 모드는 pre-deployed된 EJB 모듈이 EJB engine이 시작할 때마다 자동적으로 Deploy되도록 한다.

- **Runtime deployment:** 이 모드는 EJB engine이 시작되고 난 후에 EJB 모듈을 Deploy 한다(즉, EJB engine이 운영되고 있을 때).

위의 세 Deploy에 공통적으로, EJB stub과 skeleton 클래스를 형성하기 위해 appcompiler를 이용할 수 있다 (이 방법은 EJB engine 부팅과 Deploy하는 시간을 줄일 수 있다).

각 Deploy 모드는 5.4절에서 다룬다

### 5.2.9 appcompiler 콘솔 툴 이용하기

appcompiler 툴은 pre-deployment가 끝난 후에도 필요에 따라 RMI stub과 skeleton 클래스들을 생성하기 위해 사용할 수 있다. 이 툴은 또한 클라이언트가 Deploy된 bean과 원격통신을 EJB 클라이언트 JAR파일을 생성한다.

appcompiler 툴은 boot-time deployment 또는 runtime-deployment 가 수행될 때 자동적으로 실행된다. 이는 시간이 비교적 많이 소모되는 과정이고 engine 부팅 시에 많은 bean들을 Deploy 할 때에는 더 더욱 그러하다.

EJB engine의 부팅 시간을 줄이고자 할 때(또는 runtime-deployment를 짧게 하고 싶을 때)에는 EJB engine을 시작하기 전(또는 수동으로 runtime-deployment를 하기 전에)에 이 툴을 사용하면 된다. 그리고 application에 포함된 jeus-ejb-dd.xml이나 jeus-application-dd.xml에 fast-deploy option이 true로 설정되어 있는지 확인한다. 또한 이런 설정 없이 ejbadmin 툴의 deploy 명령에 –f 옵션을 주어도 fast deploy 방식으로 deploy가 된다. Fast deploy 옵션을 “true”로 설정하면 appcompiler가 자동적으로 기동 되지 않는다. 자세한 설명은 4장을 보라.

5.4절에서 appcompiler의 사용법을 설명할 것이다.

### 5.2.10 Pre-Deployed EJB 모듈의 디렉토리 구조

EJB 모듈이 pre-deploy된 후에 클래스 파일들과 설정파일들은 [그림 10] 에서와 같은 방법으로 JEUS 설치 디렉토리에 배포된다 .

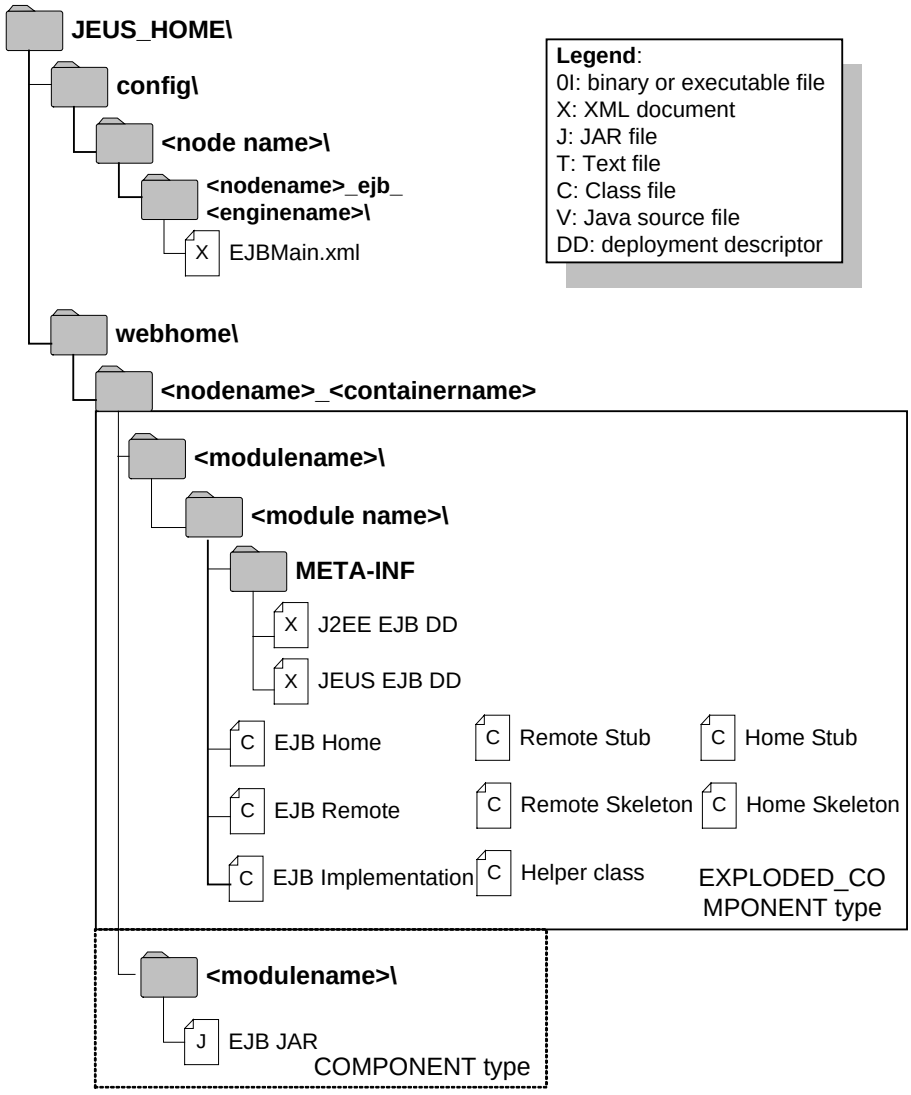


*그림 10 . pre-deploy 된 JEUS EJB module 에 포함된 파일과 디렉토리 구조*

.위의 디렉토리와 파일들에는 다음과 같은 설명이 따른다.

- **JEUS_HOME\webhome\<node_name>_<container_name>** 디렉토리는 <node_name>_<container_name>에 해당하는 container에 deploy된 application들이 존재하는 directory이다. 여기에는 Deploy된 EJB 모듈의 EJB 구현 클래스들 뿐만 아니라 helper클래스들과 EJB stub, skeleton들이 놓이게 된다.

EAR의 한 부분으로 EJB archive file (JAR file)이 deploy되는 경우에는 해당 ear file이 이 directory 아래에 놓이게 된다. 이런 방식을 EAR deployment 방식이라고 한다. 만약 standalone application으로 jar file이 단독으로 deploy될 경우에는 jar file 이름으로 되어 있는 directory 아래에 jar file이 놓이게 된다. 이 경우는 standalone deployment 방식이라고 한다. 위의 그림에서는 standalone deployment 방식만 나타나 있다.

이 directory에는 archive file 뿐만 아니라 archive file을 풀어놓은 directory도 존재할 수 있다. 즉, 위에서 설명한 archive file 자리에 archive file을 풀어놓은 directory가 놓일 수도 있다. EAR archive file을 directory로 풀어놓는 방식을 exploded EAR deployment 방식이라고 하고 jar file을 풀어둔 방식을 exploded standalone deployment 방식이라고 한다. JEUS에서는 archive file이 존재하는 standalone deployment를 COMPONENT type, 풀어놓은 것을 EXPLODED_COMPONENT type이라고 부른다.

이후의 설명에서는 standalone application에 대해서만 설명하겠다. EAR deployment 방식이나 deploy 전반에 대한 상세한 설명은 Jeus Server 매뉴얼의 deploy 부분을 참고하기 바란다.

### 5.2.11 EJB 모듈 컨트롤과 모니터링

Deploy된 EJB모듈은 다음과 같은 네가지 방법으로 조작가능하다.

- **Reload** 는 먼저 ‘undeploy”를 하고 다시 ‘deploy’를 하는 명령어이다. ‘reload’의 대한 상세설명은 다음 절에서 제공된다.
- **Suspend** 는 클라이언트의 요청이 한시적으로 EJB 모듈을 접근 불가능의 상태로 만든다. **Resume** 명령어로 다시 EJB모듈이 접근 가능한 상태로 되돌려진다.
- **Undeploy** 는 EJB engine의 운영 메모리에서 EJB 모듈을 내려 EJB 클라이언트가 Enterprise JavaBeans에 접근 불가능한 상태로 만든다. 하지만 물리적으로 파일을 삭제하지 않은 채 pre-deployment상태로 유지시킨다.
- **Monitoring** 의 의미는 Deploy되고 활성화된 EJB모듈로부터 상태와 운영 정보를 수집하는 것을 말한다.

상위 세 번째까지가 어떻게 실행되는지는 5.5절에서 설명될 것이고, 네 번째 항목은 5.6절에서 설명될 것이다.

### 5.2.12 EJB 모듈 리로딩하기

EJB모듈을 리로딩할 때는 다음 사항을 주의하기 바란다.

- **'reload'** 명령어는 효과적으로 EJB를 Undeploy하고 Deploy하는 방법이다.
- Undeployment가 실행되고 그리고 다음으로 다시 로딩이 수행될 때 undeployed된 EJB 모듈의 모든 트랜잭션 상태는 잃어 버리게 된다. 이는 Undeploy되거나 Reload되는 모듈과 연관있는 모든 EJB 트랜잭션이 롤백된다는 것을 뜻한다.
- 리로딩은 ejbadmin 콘솔 툴에서 'reload' 명령어를 실행시키거나 웹 관리자를 이용할 수 있다. 더 자세한 설명은 다음 절에서 한다.

### 5.2.13 결론

지금까지 JEUS EJB 모듈에 대한 주요 설정, 하위 컴포넌트와 디렉토리 구조들을 살펴보았다.

또한, 네 가지의 EJB모듈 관리상 중요한 내용들(assembly, deployment, control, monitoring)도 살펴보았다. 각각의 개념들은 다음 절에서 더 자세히 살펴볼 것이다.

**참고**: EJB모듈을 튜닝 하는 관점에 대해서는 그리 많이 언급할 것이 없다. 튜닝은 engine과 bean 차원에서 수행된다.

이제부터는 J2EE EJB모듈을 어떻게 조립(assembling)하는지 살펴보기로 하자.

## 5.3 EJB 모듈 조립(Assembling)

### 5.3.1 소개

여기에서는 J2EE WAS의 EJB engine에 상관없이 공통되는 EJB 모듈을 조립하여 Deploy 할 수 있는 과정의 모든 것을 배워보기로 하자[그림 11]. 여기서는 어떻게 한 모듈을 조립하는지, 그리고 다음에는 JEUS에 어떻게 Deploy 되는지 보여준다.

이 장은 JEUS에 한정된 내용은 아니지만, 완벽을 기하기 위하여 알아보도록 한다.

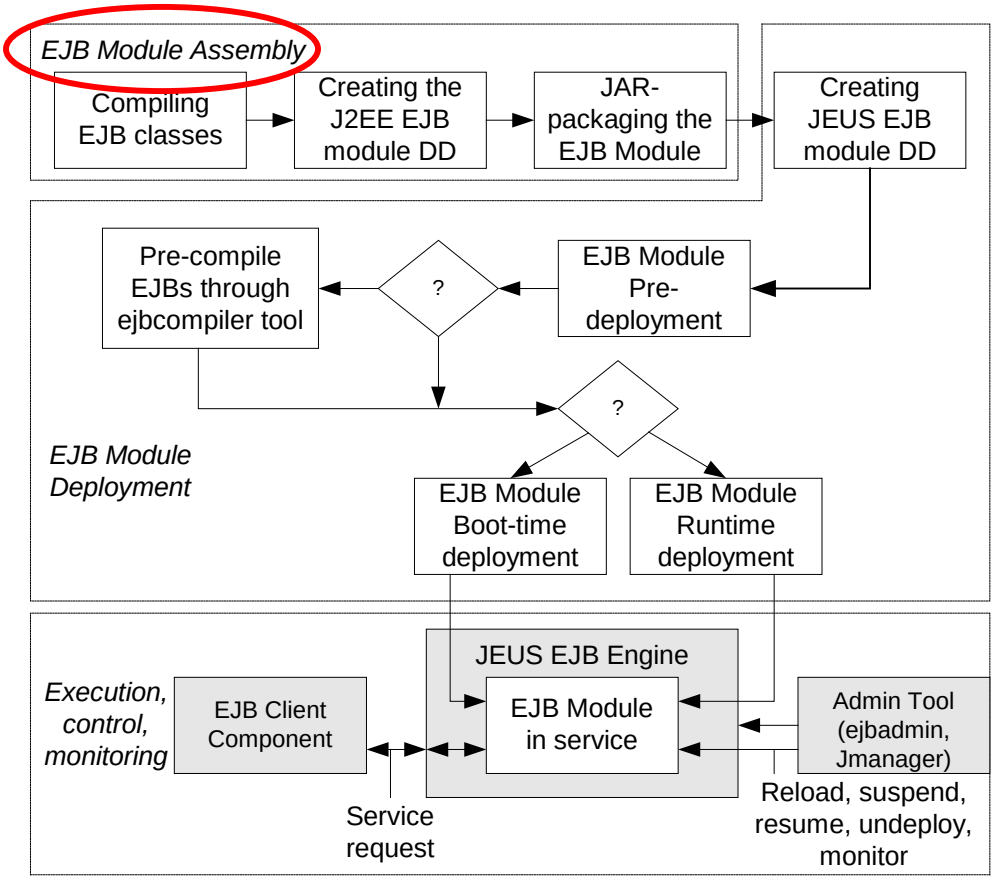


*그림 11 . EJB module 조립*

[그림 11] 에서와 같이 EJB모듈을 조립할 때 세 단계를 거친다.

1. **EJB 클래스 컴파일**. EJB의 .java 소스 파일이 이미 존재한다고 가정한다. EJB프로그래밍에 대한 정보는 관련 책자나 이 문서의 “EJB장”을 참조한다.

2. **J2EE EJB deployment descriptor를 작성한다**. DD의 포맷은 EJB 2.1 스펙에 잘 나와 있다.

3. JEUS EJB deployment descriptor를 작성한다.

4. **JAR 압축파일에 EJB클래스와 DD 패키징하기**. JAR 파일은 물리적이고, 운반 가능하지만 아직Deploy되지 않은 EJB모듈을 일컫는다.

위의 세 단계의 세부 내용은 다음 절에서 자세히 다룬다.

간단한 “counter” 예제를 통하여 이 과정을 살펴보기로 하자. “counter” 는 JEUS_HOME\samples\ejb\stateful\ 디렉토리에 있는 Stateful 세션 빈이다. 여기에는 다음과 같은 세 개의 Java 소스 파일이 있다.

Counter.java (remote interface)

CounterHome.java (home interface)

CounterEJB.java (bean implementation)

Stateful 세션 빈이나 다른 네 가지 종류의 빈에 대한 자세한 정보는 이 문서의 후반부에 있는 10,11,12장을 참고하기 바란다.

### 5.3.2 EJB 클래스 컴파일

EJB 소스 파일들로부터 EJB모듈을 조립하는 첫 번째 과정은 소스파일들을 컴파일 하는 일이다. 이는 Java 2 JDK의 javac 툴을 이용한다. classpath로는 JEUS_HOME\lib\system\jeus.jar 파일을 설정하고, 다른 헬퍼(helper) 클래스가 있다면, 이 또한 설정해 준다.

아래의 예제는 “counter” 빈에서 사용될 EJB Java 파일들을 윈도우 환경에서 컴파일 하는 것을 보여준다. 세 개의 소스 파일들은 “counter”라는 디렉토리에 존재한다는 가정을 하자 (왜냐하면 EJB패키지의 이름이 “counter”이기 때문이다).

```
> javac –classpath c:\jeus50\lib\system\jeus.jar *.java
```

위 명령의 실행으로 세 개의 .class파일들이 생성되었을 것이다. (bean, home, remote 클래스) 이제 표준 J2EE EJB deployment descriptor 를 생성할 수 있는 준비가 되었다.

**참고**: appcompiler 유틸리티는 기본적인 EJB 클래스들을 컴파일 하는데 사용될 수 없다. 이 툴은 RMI stub과 skeleton클래스 그리고 EJB client JAR파일을 생성하기 위해서만 사용된다.

### 5.3.3 표준 J2EE EJB Deployment Descriptor 생성

J2EE EJB deployment descriptor는 수동으로 또는 JEUS Builder를 사용하여 생성할 수 있다. 먼저 앞의 방법을 설명하겠다.

표준 J2EE descriptor를 수동으로 생성할 때는 일반적인 텍스트 편집기를 열어 새 파일을 생성한다. “ejb-jar.xml”이라고 이름짓고, 임의의 디렉토리에다 가져다 놓는다.

다음으로는 EJB 스펙을 준수한 XML 정보를 입력한다. 아래는 “counter”빈의 간단한 J2EE EJB deployment descriptor의 예이다. 당연히, deployment descriptor는 빈의 종류와 특수 요구사항에 따라 확연히 달라진다. J2EE deployment descriptor는 단순히 단일 빈을 나타내는 것이 아니라 전체 EJB 모듈을 나타내고 있다는 것을 명심한다.

<<*ejb-jar.xml*>>

```
<?xml version="1.0"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
    <display-name>CounterBeanModule</display-name>
    <enterprise-beans>
        <session>
            <display-name>CounterBean</display-name>
            <ejb-name>counter</ejb-name>
            <home>counter.CounterHome</home>
            <remote>counter.Counter</remote>
            <ejb-class>counter.CounterEJB</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Bean</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor/>
    <ejb-client-jar>counterbeanclient.jar</ejb-client-jar>
</ejb-jar>
```

위에 나오는 모든 태그들은 J2EE 표준에 준하는 것이기 때문에 이들의 사용법을 설명하지 않겠다. EJB 2.1 스펙이나, 이에 관련된 주제를 다루고 있는 책을 참고하기 바란다.

JEUS Builder를 이용한 방법은 JEUS Builder 안내서를 참조 하기 바란다.

### 5.3.4 JEUS EJB Module Deployment Descriptor 생성

JEUS에서 EJB 모듈 Deploy 시에 중요한 것 중의 하나는 JEUS EJB 모듈 DD이다. JEUS EJB 모듈 DD를 작성하는 이유는 JEUS 에 맞게 EJB 모듈을 설정하고 J2EE DD에서 발견된 외부 레퍼런스들을 실제적인 시스템 서비스들과 컴포넌트들에 매핑시키기 위함이다(주로 JNDI를 통하여 한다). 풀링과 엔티티 빈, 데이터베이스 테이블과의 매핑정보를 설정하는 정보들도 여기에 들

어있다. 이러한 설정들은 EJB모듈을 JEUS 시스템 및 그 주변 환경들에 맞추어 Deploy하기 위해서는 반드시 필요한 작업들이다.

JEUS EJB DD는 기본 또는 표준 J2EE DD 위에 놓이는 추가적인 설정으로 볼 수 있다. J2EE DD와 마찬가지로 JEUS EJB DD도 Enterprise Java Beans(즉, 단일 EJB 모듈)에 적용된다.

JEUS EJB DD파일은 반드시 다음과 같은 명명 규칙을 가져야 한다.

jeus-ejb-dd.xml

XML파일의 주요 부분과 설정은 다음과 같다.

- <module-info> EJB module name과 role assignment를 포함한다. Role assignment에 관련된 정보는 7장의 “보안”을 참고하기 바란다.
- <beanlist> EJB 모듈 내의 모든 EJB들을 리스팅 한다. 이 하위 태그의 이름과 내용은 빈의 종류(session , entity 종류)에 따라 다양한 모습을 가진다.

  **중요**: bean list와 그것의 sub-elements는 이번 장에서 다루지 않는다. 6장, 10장부터 12장을 통해 자세한 설명을 하겠다.
- CMR 2.0 종류의 빈들을 위한 CMR 매핑을 정의한 EJB relation mapping 설정이 있다 ( 이 태그는11장에서 설명한다).

다음은 JEUS EJB 모듈 DD 템플릿 파일의 예이다.

<<*jeus-ejb-dd.xml*>>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
        <role-permission>
            . . . <!-- See chapter 7-->
        </role-permission>
    </module-info>
    <beanlist>
        <jeusbean>
            . . . <!-- See chapters 6 and 10 -->
        </jeusbean>
        <jeusbean>
```

```
            . . . <!-- See chapters 6 and 10 -->
        </jeusbean>
            . . .
    </beanlist>
    <ejb-relation-map>
        <!-- See chapter 11 -->
    </ejb-relation-map>
</jeus-ejb-dd>
```

### 5.3.5 EJB JAR 파일 작성 (Packaging)

EJB JAR파일을 작성하는데 있어 두 가지의 옵션이 있다. JEUS Builer 또는 jar유틸리티를 이용하는 옵션이 있다. 수동으로 작성하는 방법을 여기서 설명하고 JEUS builder를 사용하는 방법은 JEUS Builder 안내서를 참조 하기 바란다.

먼저 다음 사항을 가정한다.

- “counter” 디렉토리에 counter EJB 클래스 파일들이 존재한다.
- 앞 절에서 우리가 생성한 ejb-jar.xml파일과 jeus-ejb-dd.xml이 “META-INF”디렉토리에 존재한다.
- 위의 두 디렉토리는 “counter” 라는 부모 디렉토리 아래에 존재하는 것을 가정으로 한다.

그럼 먼저 필수인 client JAR파일을 생성해보도록 하자( 앞 절에 ejb-jar.xml에 의하여 “counterbeanclient.jar”라고 생성해야 한다). 콘솔 창에서 명령어를 수행한다.

```
> jar cf counterbeanclient.jar counter\CounterHome.class
counter\Counter.class
```

다음으로는 EJB 클래스들과, ejb-jar.xml, client JAR를 포함하고 있는 실제 EJB JAR파일을 생성하기로 하자.

```
C:\temp\counter> jar cf countermod.jar counter meta-inf cou
nterbeanclient.jar
```

이 것으로 ‘countermod.jar”라는 J2EE EJB 모듈을 완성하였다.

### 5.3.6 결론

이것으로 JEUS에서 J2EE EJB 모듈을 조립하는 간단한 예를 살펴보았다.

위에서 본 것과 같이 조립과정은 네 가지 주요 과정(컴파일, J2EE EJB DD 생성, JEUS EJB DD 생성, JAR 패키징)으로 나뉜다.

다음 절에서는 JEUS에 J2EE에 준하는 EJB 모듈을 Deploy 하는 방법을 소개한다.

## 5.4 EJB 모듈 Deploy

### 5.4.1 소개

이 절에서는 어떻게 JEUS EJB deployment descriptor를 작성하고 JEUS EJB engine에 Deploy 하는지 알려준다[그림 12].

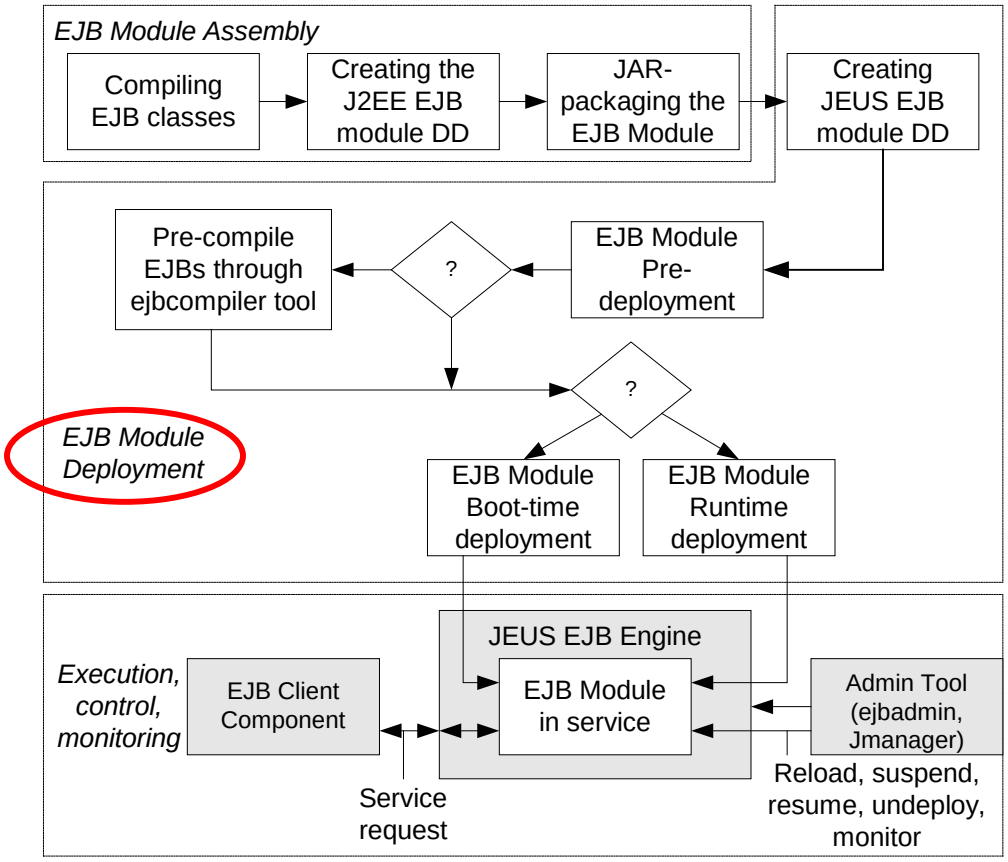


*그림 12. JEUS 에서 EJB module deployment*

.

EJB 모듈을 Deploy하는 과정은 다음과 같다.

- Pre-deployment
- 선택적인 EJB stub 과 skeleton 컴파일 (pre-compile)
- Boot-time deployment
- Runtime deployment (boot-time deployment시에는 사용되지 않음)

여기에서는 앞 절에서 설명했던 EJB 클래스와 표준 J2EE DD파일, JEUS EJB DD 파일이 포함된 EJB모듈이 이미 조립된 상태임을 가정한다. 이 절을 읽기 전에 준비해야 할 것은 J2EE DD, JEUS DD와 함께 패키지 된 EJB JAR모듈이다.

Deploy는 ejbadmin이나 웹 관리자에서 할 수 있다. 이 장의 맨 마지막 부분을 제외한 나머지 부분에서는 ejbadmin과 다른 표준 콘솔 툴을 이용하여 “수동”으로 Deploy 하는 방법을 설명한다.

아래의 예에서 사용될 EJB JAR파일은 “countermod.jar”이고 EJB 모듈 이름은 “countermod”라고 가정한다.

### 5.4.2 EJB Module Deploy

EJB 모듈을 Deploy 하기 전에 다음과 같은 부분들이 반드시 존재해야 한다.

- **EJB 모듈 JAR** 파일 (예. “countermod.jar).
- **표준 J2EE EJB DD** “ejb-jar.xml” 과 **JEUS EJB DD** “jeus-ejb-dd.xml” 이 파일들은 반드시 EJB 모듈 JAR파일의 META-INF 디렉토리 안에 존재해야 한다.

이 부분들의 설정이 끝나면, EJB engine에 실제 Deploy를 할 단계이다. 앞에서 본 바와 같이, EJB모듈을 Deploy 하는 데에는 세가지 모드가 있다:

- Pre-deployment
- Boot-time deployment
- Runtime deployment

각 Deploy 방법은 다음과 같다.

### 5.4.3 EJB 모듈 Pre-deployment

Pre-deployment는 “semi-deployment”이다. 즉, 이 과정은 EJB archive file이나 이 file의 내용을 가진 directory가 deploy될 container의 directory에 존재한다는 의미를 가진다. 그러나 EJB모듈은 실행 시에 EJB engine에 로딩되지 않는다. 그러므로, pre-deployment는 클라이언트에서 실제로 사용하기 전에 EJB 모듈 설정과 준비 과정에서 이루어진다.

먼저 archive file을 그대로 사용할 때의 pre-deployment 과정은 다음과 같다.

1. Deploy하고자 하는 container의 이름이 <node_name>_<container_name>일 때 archive file을 webhome\<node_name>_<container_name>의 archive file name으로 된 directory 내에 둔다. 즉, archive file이 countermod.jar이고 container 이름이 johan_container1인 경우 이 file이 복사되어야 할 directory는 webhome\johan_container1\countermod\countermod.jar가 된다.

만약archive file을 풀어서 사용할 경우의pre-deployment 과정은 다음과 같다.

2. Deploy하고자 하는 container의 이름이 <node_name>_<container_name>일 때 archive file을 webhome\<node_name>_<container_name>의 archive file name으로 된 directory 내에서 풀어둔다. 즉, archive file이 countermod.jar이고 container 이름이 johan_container1인 경우 이 file의 내용이 webhome\johan_container1\countermod\countermod 디렉토리에 풀리게 된다.

위의 과정을 거치면 pre-deployment를 마치게 된다. 이는 빈이 실제적으로 EJB engine에 Deploy되지 않았지만 영구적으로는 JEUS시스템에 설치되었음을 의미한다. 마침내, 서비스가 가능한 “counter” EJB를 언제든지 runtime Deploy할 수 있는 충분한 준비가 완료되었다.

### 5.4.4 appcompiler 툴 이용

위에서 설명하였듯이, appcompiler툴은 EJB모듈 pre-deployment와 boot-time/runtime deployment사이에서 선택적으로 사용될 수 있다. 이 툴을 사용함으로써 EJB engine에 EJB모듈을 Deploy 하는 속도를 빠르게 할 수 있다.

여기에서는 “johan”노드에서 pre-deployed된 “countermod” EJB 모듈의 stub와 skeleton 그리고 클라이언트 JAR파일들을 생성하기 위하여 appcompiler를 사

용하는 예를 보여준다. 예에서 실행 directory는 jar file이 존재하는 directory 이다.

```
appcompiler countermod.jar
```

수행 후에는 jar file 내에 stub과 skeleton파일들이 생성된다.

만약 archive를 directory로 풀어둔 경우에도 동일하게 사용한다. 다음 예에서 실행 directory는 archive를 풀어둔 directory가 존재하는 directory이다.

```
appcompiler countermod
```

수행 후에는 이 directory 내에stub과 skeleton파일들이 생성된다. appcompiler 툴에 대한 자세한 정보는 JEUS 서버 안내서를 참조하라.

**참고**: appcompiler를 사용할 경우에는 jeus-ejb-dd.xml이나 jeus-application-dd.xml의 fast-deploy 옵션을 사용하거나 ejbadmin 툴의 deploy 명령어에서 –f 옵션을 사용하도록 한다. 이 옵션이 사용되지 않을 경우에는 appcompiler는 효과가 없다. EJB engine에 대해서는 4장을 참고하라. 아래에 ejbadmin 툴에 대한 더 상세한 정보가 소개되어 있다.

### 5.4.5 EJB 모듈의 Boot-time Deployment

Pre-deployment가 끝나면 pre-deployed된 모듈들이 engine이 부팅할 때 마다 EJB engine에 Deploy 되도록 설정할 수 있다.

이렇게 하기 위하여, JEUSMain.xml파일 안에 <application> 태그를 이용해 EJB application을추가해 줘야 한다. 다음 예는 johan_container1의 directory에 pre-deployed된 “countermod” EJB 모듈을 추가하는 방법을 보여준다. 여기에 대한 자세한 설명은 JEUS Server 안내서의 deployment 항목을 참고하기 바란다.

*<<JEUSMain.xml>>*

```
<jeus-system>
  . . .
    <application>
    <name>countermod</name>
    <path>countermod</path>
    <deployment-target>
```

```
        <engine-container-name>johan_container1</engine-container-name>
      </deployment-target>
      <deployment-type>COMPONENT</deployment-type>
      <ejb-component>
        <uri>countermod.jar</uri>
      </ejb-component>
  </application>   . . .
</jeus-system>
```

여러 개의 <application> 태그들이 하나의 JEUSMain.xml파일에 추가될 수 있다. 각각의 <application> 태그가 각 pre-deployed된 모듈을 대표하며 engine 부팅 시에 Deploy됨을 명시한다. JEUS Server 메뉴얼에 JEUSMain.xml 설정 파일에 대한 정보가 설명되어 있다.

JEUSMain.xml 파일에 application을 추가하고 나서, countermod 모듈이 Deploy 되도록 EJB engine을 재시작 한다.

### 5.4.6 EJB 모듈의 Runtime Deployment

Runtime-deployment는 실행되고 있는 EJB engine에 EJB 모듈을 Deploy하는 것을 의미한다(이는 EJB engine이 시작할 때만 발생하는 boot-time Deploy와는 반대되는 개념이다).

Runtime-deployment를 할 수 있는 두 가지 방법이 있다. 하나는 ejbadmin 콘솔을 이용하는 방법이고, 다른 하나는 웹 관리자 를 이용하는 것이다.

Pre-deployment만이 이루어져 있다면 ejbadmin툴의 ‘deploy’명령어를 이용하여 실행되고 있는 EJB engine에 모듈을 설치할 수 있다.

“countermod” EJB모듈이 어떻게 Deploy되는지 다음 예에서 설명하고 있다. ejbadmin을 이용한 설명은 부록 A에 모두 설명되어 있다. 아래의 예에서는 JEUS의 노드이름이 “johan”이고, “johan_ejb_engine1”이 JEUS서버에 이미 부팅되어 있는 상태임을 가정한다. 자세한 정보는 4장을 참고하기 바란다.

```
c:\> ejbadmin johan_ejb_engine1
```

로그인명과 패스워드를 입력한다.

```
johan_ejb_engine1> deploy countermod
```

“countermod” EJB 모듈이 Deploy 되었다.

Deploy를 확인하기 위하여 다음 명령어를 수행시킨다.

```
johan_ejb_engine1> modulelist
```

“countermod”모듈이 포함된 모든 Deploy된 모듈들이 표시된다.

EJB 모듈에 포함된 EJB들은 이제 어떤 클라이언트에게도 서비스될 수 있는 상태가 되었다.

‘deploy’ 명령을 실행하기 전에, stub과 skeleton을 생성하기 위하여 appcompiler 스크립트를 실행시키면, ejbadmin에서는 ‘deploy’명령과 함께 –f 옵션을 사용할 수 있다. 이렇게 하면 stub과 skeleton클래스들이 재생성되지 않기 때문에 Deploy 속도를 빠르게 할 수 있다.

‘deploy’ 명령어와 –per(“permanent”) 파라미터를 사용하면 Deploy된 EJB모듈은 JEUSMain.xml의 <application> 태그로 추가된다.

부록 B에서는 “-Djeus.ejb.sourcegenerate.classpath”를 이용하는 것이 설명되어 있다. 이것은 Deploy 중에 appcompiler가 추가적으로 classpath를 사용할 수 있게 한다.

**참고**: 4.x대의 classloader 방식은 DIR과 JAR classloading 방식에 따라 각각 다르게 구현되어 있다. 5.0에서는 다른 WAS의 방식과 비슷한 isolated classloading 방식을 지원한다. 별도의 classloading 설정이 없다면 4.x대의 JAR mode와 같은 shared classloading 방식을 적용해서 EJB 모듈을 deploy한다. 각각의 classloading 방식에 대한 설명과 적용 상황은 JEUS Server 안내서의 JEUS classloader hierarchy 항목을 참조하기 바란다.

### 5.4.7 결론

이것으로 JEUS EJB 모듈 Deploy에 대한 설명을 마무리 한다.

이상으로 다음과 같이 5가지 중요한 과정을 설명하였다.

- JEUS EJB DD생성하기
- pre-deployment
- EJB pre-compilation
- boot-time deployment
- runtime deployment

다음 절에서는 어떻게 Deploy된 EJB 모듈이 컨트롤되는지 살펴보기로 하자.

# 5.5 EJB 모듈 컨트롤링

## 5.5.1 소개

컨트롤이란 Deploy된 EJB모듈을 suspend, resume, reload 명령어들을 적용함으로써 EJB 모듈 내의 EJB의 실행 상태를 변경하는 것을 의미한다 [그림 13].

Deploy된 EJB모듈을 컨트롤하는 방법에는 크게 두 가지가 있다. 하나는 ejbadmin 콘솔 툴을 이용하는 것이고 다른 하나는 웹 관리자를 이용하는 것이다.

ejbadmin을 사용하는 것부터 시작해 보자.

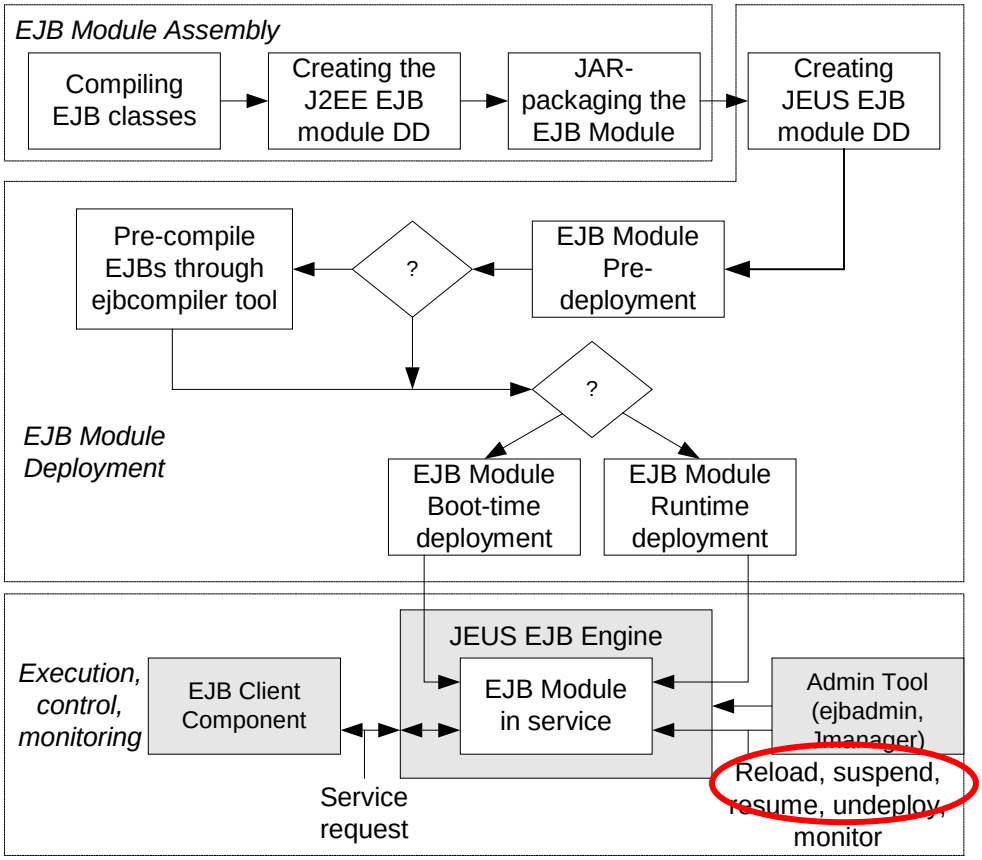


*그림 13. EJB modules deploy control 화면.*

### 5.5.2 Console 의 사용

ejbadmin 콘솔 툴에는 다음과 같은 컨트롤 명령어들이 존재한다:

- reload <module name> – 지정한 EJB모듈을 다시 로딩한다. 이 의미는 현재 활성화 되어 있는 EJB 모듈을 EJB 실행환경에서 Undeploy시켰다가 다시 디스크로부터 읽어 Deploy시키는 것을 의미한다.
- suspend <module name> – 일시적으로 선택된 EJB 모듈을 중지시킨다.
- resume <module name> – suspend되었던 EJB 모듈을 다시 활성화시킨다.
- undeploy <module name> – 선택된 모듈을 undeploy시킨다 (EJB engine의 실행 메모리에서 제거시킨다).

위에서 제시한 모든 명령어들은 EJB 모듈 이름을 파라미터로 받는다.

ejbadmin과 컨트롤 명령어에 대한 상세한 정보는 부록 A를 참조하라.

여기 ejbadmin 콘솔 툴을 사용하여 EJB 모듈을 undeploy하는 방법을 설명하는 예가 있다.

```
c:\> ejbadmin johan_ejb_engine1
```

사용자 이름과 패스워드를 입력한다.

```
johan_ejb_engine1> undeploy countermod
```

“countermod” EJB 모듈이 undeploy 되었다.

undeploy된 것을 확인하기 위하여 다음과 같은 명령을 수행한다.

```
johan_ejb_engine1> modulelist
```

“countermod”모듈을 제외한 모든 Deploy된 모듈들이 나열된다.

EJB모듈 안에 포함되었던 EJB들은 클라이언트의 호출에 응할 수 없게 되었다(반면, 모든 EJB관련된 클래스들과 설정들은 그대로 존재한다).

### 5.5.3 결론

여기 이 요약 절에서는 EJB engine내의 Deploy된 EJB모듈을 컨트롤하는 방법에 대하여 알려주었다.

다음 절에선 EJB 모듈의 모니터링에 대해서 알아 보겠다.

# 5.6 EJB 모듈의 모니터링

## 5.6.1 소개

EJB모듈의 모니터링이란 모듈이나 그 내부 빈의 상태를  보는 것이다[그림 14].

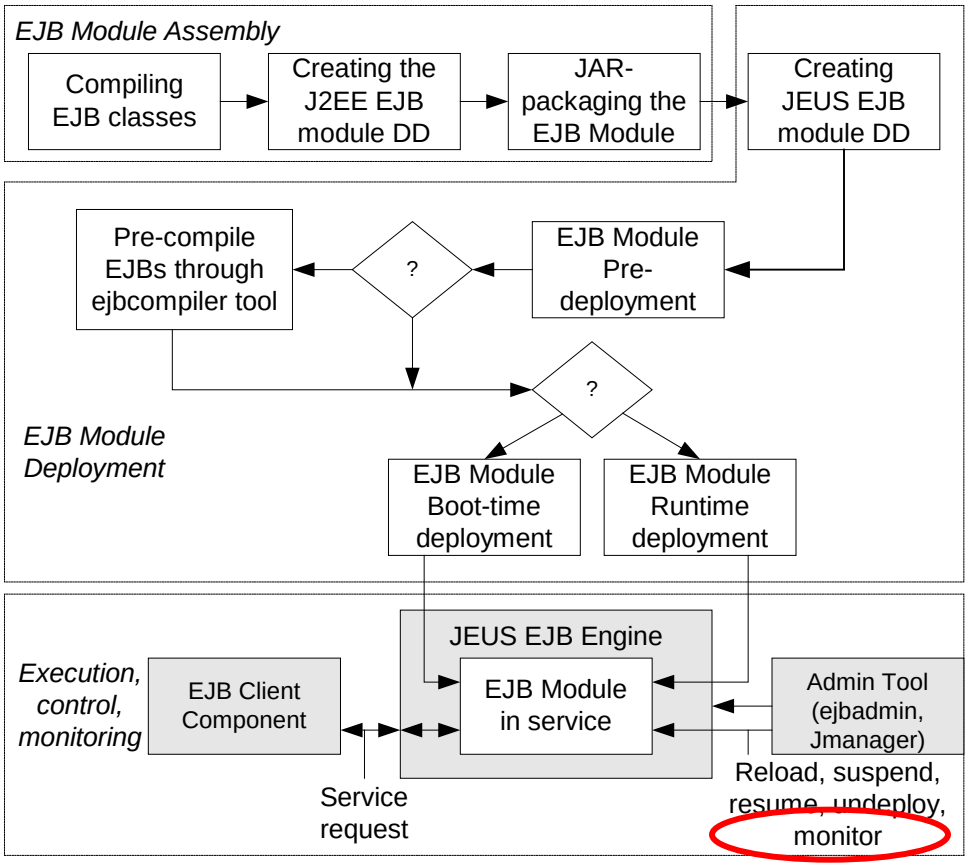


*그림 14. EJB module 모니터링*

ejbadmin로는 EJB모듈의 기본적인 사항만 볼 수 있다. 그러나 웹 관리자는 모듈에 대한 더 많은 정보를 제공한다.

## 5.6.2 Console 의 이용

ejbadmin 콘솔 툴을 이용할 때 두 가지의 명령어가 있다.

- modulelist – 이 명령어는 EJB engine에서 실행중인 EJB모듈을 볼 수 있다. 이 명령어는 주로 EJB모듈이 Deploy 되었는지 확인 하는데 사용된다.

- beanlist <module name> – 이 명령어는 “module name”에 포함된 EJB의 목록을 보여준다.
- moduleinfo <module name> - 모듈에 대한 정보를 보여준다.

ejbadmin의 전체 내용을 보려면 부록 A를 참조하길 바란다.

### 5.6.3 결론

EJB engine내의 Deploy된 EJB모듈을 모니터링하는 방법에 대하여 알아보았다.

## 5.7 결론

이것으로써 JEUS의 EJB 모듈에 대한 소개를 마치고자 한다. EJB 모듈이 어떻게 조립되고, Deploy되며, 컨트롤되고, 모니터링되는지 확인하였다.

이 문서의 나머지 부분에서는 EJB의 핵심 컴포넌트(Enterprise JavaBeans)에 대한 모든 것에 대해 설명한다. Enterprise JavaBeans은 EJB 모듈 내에 포함된 업무 로직을 가지고 있는 컴포넌트이고 EJB engine이 실행시킨다.

# 6 JEUS 의 일반적인 EJB 들

## 6.1 소개

이 장에서는 JEUS WAS 환경에서의 일반적인 Enterprise JavaBeans에 대하여 소개하고자 한다.

이 장에서는 EJB의 일반적인 사항에 대하여 설명할 것이다. 즉 모든 공통적인 EJB의 특성들이 다뤄질 것이며, EJB의 특정한 종류들에 대한 추가적인 정보들, 예를 들어 CMP, BMP, MDB들은 10장부터 12장까지에서 다뤄질 것이다 .

**참고**: 이 장에서는 일반적인 EJB 프로그래밍과 표준 설정법들은 설명하지 않는다. 그러한 것들은 J2EE EJB1.1과 2.0, 2.1 스펙에서 자세히 설명하고 있다. 이 장에서는 JEUS시스템에서 실행할 수 있는 J2EE 표준규격의 EJB를 만들기 위한 추가적인 정보들만 제공한다.

실제적인 Enterprise JavaBeans의 deploy에 대해 언급하지 않을 것이다. 빈들은 항상 EJB모듈의 한 부분으로 deploy 되기 때문에, deploy에 관한 것들은 이미 EJB 모듈 장(5장)에서 다루었다. 그러므로, 이 장을 포함하여 다음 장들을 읽기 전에 반드시 5장의 내용에 대하여 숙지하고 있어야 한다.

## 6.2 Enterprise JavaBeans 의 개요

### 6.2.1 소개

아래의 [그림 15]은 다섯 가지 기본적인 빈들을 보여준다. 이 장에서는 이 빈들이 가지는 특성들을 설명할 것이다(그림에서 "CMP" 타입은 실제로는 다르지만 매우 밀접하게 관계가 있는 "CMP1.1"과 "CMP 2.0"을 의미한다).

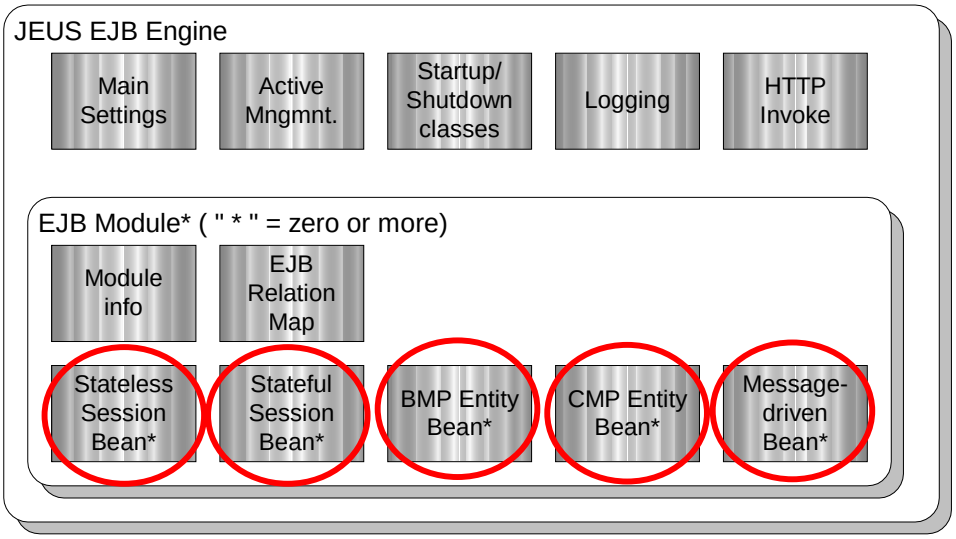


*그림 15. EJB engine 과 EJB 모듈에 포함된 다섯 가지 기본적인 EJB 타입들*

### 6.2.2 JEUS EJB DD(Deployment Descriptor)

JEUS EJB DD는 "jeus-ejb-dd.xml로 명명된 일반적인 XML파일이고 속하는 application의 archive file이나 해당 directory 내에 존재한다.

JEUS EJB DD는 5장에서 언급을 했었다. 이 장에서는 이 설정 파일에 대하여 자세히 설명하기로 한다.

### 6.2.3 여섯 가지 EJB 종류

JEUS는 이 문서 처음의 "개요" 에서 언급했듯이 아래에 나열된 EJB들을 지원하고 있다( J2EE 1.3 스펙에 엄격히 준한다).

1. Stateless session bean (10장)
2. Stateful session bean (10장)
3. BMP entity bean (11장)
4. CMP 1.1 entity bean (11장)
5. CMP 2.0 entity bean (11장)
6. MDB message-driven bean (12장)

위의 목록에는 어떤 장에서 추가적인 상세 정보를 설명하고 있는지 나타내고 있다. 이 장에서는 모든 빈들에 적용되는 공통적인 특징과 설정에 대하여 설명한다.

### 6.2.4 각 EJB 타입에 지원되는 설정

아래 [표 3] 에서는 JEUS EJB의 XML로 설정 가능한 기능들과 컴포넌트들의 간략한 정보를 보여주고 있다. 표의 왼쪽에 나열된 항목들은 실제 설정 가능한 JEUS EJB DD 의 XML 태그들이다. 표의 나머지 6개의 열은 EJB종류에 따른 설정 사항들을 보여주고 있다.

*표 3. JEUS EJB 의 설정 가능한 특징들과 컴포넌트들.*

| 설정 가능한 개체 | Stateless | Stateful | BMP | CMP1.1 | CMP2.0 | MDB |
|---|---|---|---|---|---|---|
| 1. EJB name | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2. Export name | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 3. Local export name | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 4. Export port | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 5. Export IIOP switch | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 6. Single VM switch | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 7. Local invocation opt. | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 8. fast deploy | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 9. use-access-control | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 10. **Run-as identity** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 11. **Security CSI Interop**. | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 12. Env. Refs | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 13. EJB refs | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 14. Resource Refs | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 15. Resource env. Refs | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 16. Thread ticket pool settings | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 17. **Clustering settings** | ✓ | ✓ | ✓ | ✓ | ✓ | |
| 18. HTTP Invoke | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 19. Pooling bean switch | | ✓ | | | | |
| 20. File DB session store | | ✓ | | | | |
| 21. Object management | | ✓ | ✓ | ✓ | ✓ | |
| 22. Persistence Optimize | | | ✓ | ✓ | ✓ | |
| 23. CM persistence opt. | | | | ✓ | ✓ | |

| 설정 가능한 개체 | Stateless | Stateful | BMP | CMP1.1 | CMP2.0 | MDB |
|---|---|---|---|---|---|---|
| 24. Schema info | | | | ✓ | ✓ | |
| 25. Relationship map | | | | | ✓ | |
| 26. Connect. fact. name | | | | | | ✓ |
| 27. MDB resource adaptor | | | | | | ✓ |
| 28. Destination | | | | | | ✓ |
| 29. Max messages | | | | | | ✓ |
| 30. Acknowledge mode | | | | | | ✓ |
| 31. Durable switch | | | | | | ✓ |
| 32. Message selector | | | | | | ✓ |
| 33. JNDI SPI settings | | | | | | ✓ |
| 34. Durable timer service | ✓ | | ✓ | ✓ | ✓ | ✓ |

위의 표는 아래처럼 나뉘어 진다.

- 항목 1부터 18까지는 6개의 EJB종류 모두에 공통으로 적용되는 사항이다(MDB의 2~8과 17는 제외). 이 모든 항목들에 대해서 이 장에서 다룬다.
- 항목 10(run-as identity)과 11(security interoperability), 17(clustering)는 모든 EJB 종류에 공통이지만, 바로 이 다음 장(7, 8, 9장)에서 별도로 다룰 것이다.
- 항목 19와 20은 stateful session bean 에 대해서만 적용되고 10장에서 자세히 다룬다.
- 항목 21(object management)는 stateful session beans와 세 가지의 entity bean(BMP, CMP1.1, CMP2.0)과 공유된다. 이 항목은 session bean를 설명하는 장(10장)에서 다룬다.
- 항목 22부터 25까지는 entity bean에만 적용된다. 이들은 11장에서 다룬다.
- 항목 26부터 33까지는 message-driven bean(MDB)에만 적용된다. 이들은 12장에서 다룬다.

- 항목 34는 EJB 2.1 spec에서 timer service를 사용할 수 있는 bean (stateful session bean을 제외한 모든 bean)에 사용할 수 있다. 이는 13장에서 다룬다.

### 6.2.5 기본적인 JEUS 고유의 EJB 설정

모든 EJB는 jeus-ejb-dd.xml파일에 기본적인 JEUS 전용 태그들을 설정해야만 한다. 이 기본 설정들은 [표 3] 의 항목 1부터 9까지 해당된다. 이 설정들은 다음과 같다.

- ejb-jar.xml에 주어진 이름과 동일한 EJB name.
- JNDI export name.
- EJB가 방화벽이나 그 비슷한 소프트웨어를 통하여 접근될 때 사용되는 RMI export port.
- 최적화 설정(local invocation optimize)은 local interface의 대체용으로 사용될 수 있다.
- Bean 별로 fast deploy 여부를 설정할 수 있다.
- 기타.

이 설정에 대한 자세한 정보는 다음 절에서 다시 볼 수 있다.

### 6.2.6 EJB Run-as Identity

EJB run-as identity는 다른 빈에서 method를 호출할 때 그 빈과 연관된 빈의 식별자를 일컫는다. 그러므로 run-as identity는 method호출 시에 전달되어 다른 빈들이 이 빈의 특정한 method를 호출하게 할 지 결정하도록 한다.

JEUS EJB DD에서는 표준 J2EE EJB DD(ejb-jar.xml)에 선언된 run-as identity *roles* 과 실제 시스템 *principals*을 매핑 시켜줘야 한다.

Run-as identity는 다른 EJB 보안 관련 사항들과 함께 7장에서 설명한다.

### 6.2.7 EJB 보안 상호 운용

JEUS EJB의 보안 상호운용은 EJB engine이나 빈이 다른 벤더의 WAS와 상호작용 할 수 있게 한다.

보안 상호운용은 8장에서 논의 한다.

### 6.2.8 Thread Ticket Pool

[그림 16]은 EJB thread ticket pool (“TTP”)의 개념에 대하여 보여주고 있다.

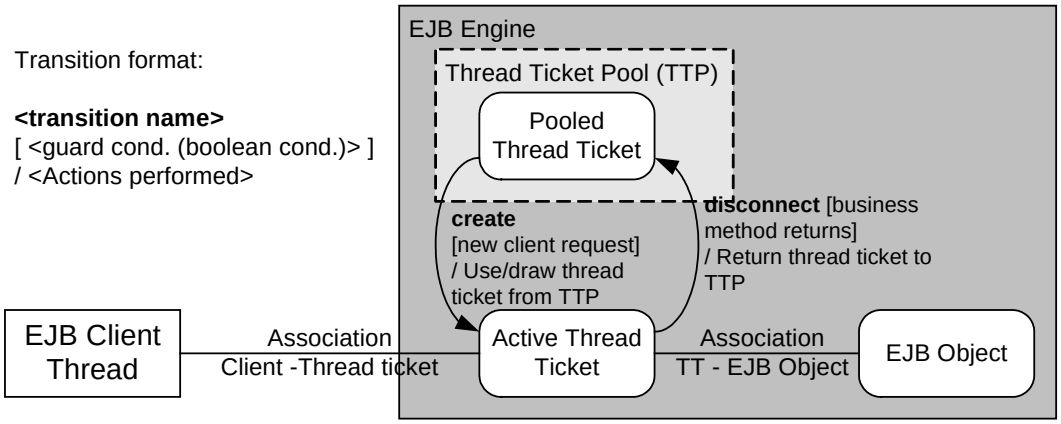


*그림 16. thread ticket/thread ticket pool 상태 차트: thread ticket pool은 클라이언트 요청을 받으면 thread ticket을 클라이언트 요청에 부여한다. 그런 후 클라이언트는 EJB 객체와 연관을 맺게 된다*

[그림 16]에서 볼 수 있듯이 thread ticket pool은 thread ticket (“TT”)을 가지고 있다. 한 개의 TT는 하나의 클라이언트 thread가 EJB engine에 접근할 수 있도록 허가한다. 클라이언트의 요청이 도착하면, 하나의 TT가 TTP에서 얻어지고 클라이언트에게 부여된다. 이는 각 클라이언트 thread는 실제 EJB 인스턴스에게 요청을 전달할 한 개의 EJB Object에게 할당된다는 것을 의미한다 (그러므로 EJB Object는 EJB engine 안에 존재하는 한 Object라고 볼 수 있고 EJB 클라이언트와 실제 EJB 인스턴스와의 연결을 관리한다).

thread ticket pool에 있는 TT의 숫자 보다 많은 클라이언트의 요청이 올 경우, 새로운 요청은, TT가 free되어 pool로 반환 될 때까지 기다려야 한다. EJB 처리가 끝나거나(stateless session과 MDB bean) connection timeout이 발생하여 클라이언트가 연결을 잃을 경우에 TTP로 TT가 반환된다.(10, 11, 12장)

thread ticket pool은 5개의 기본 EJB 타입 각각에 적용되는 pool일 뿐이다. 각 실제 EJB는 EJB engine 내부에 고유의 thread ticket pool을 가진다.( 즉, EJB당 하나의 TTP를 설정한다).

Stateful bean과 entity bean은 두 가지의 pool들이 설정될 수 있다. 하나는 connection pool이고 다른 하나는 bean pool이다. 다음 장에서 이 두 pool들에 대해 더 상세히 다루겠다(10장과 11장).

Stateless bean과 MDB에서는 thread ticket pool이 설정 가능한 유일한 pool이다. 이런 bean에서는 thread ticket이 실제의 bean instance를 나타낸다.(10장과 12장 참조).

**참고**: 위의 그림은 UML과 유사한 표기법을 사용한 가상의 상태 차트이다. 그림 안의 각 상자는 어느 한 종류의 실제 인스턴스와 인스턴스의 상태를 모두 표현한다. 상태 전이(state transition)로는 표준 UML 표기법을 이용하여 화살표로, transition name은 굵은 글자로, guard condition(모두 참이어야 transition이 발생할 수 있는 Boolean 조건들)은 대괄호로, 그리고 전이 중에 실행되는 action의 리스트가 표현되어 있다. 이 문서에서는 비슷한 주제에서는 계속 이 표기법을 사용하므로 익숙해지기 바란다.

### 6.2.9 시스템의 Entity 레퍼런스의 매핑

JEUS EJB DD의 중요한 역할 중의 하나는 ejb-jar.xml파일에 정의된 참조와 실제 시스템에서 사용되고 있는 JNDI export name들과의 매핑이다. 이 매핑은 다음과 같은 시스템 자원들에 적용되어야 한다.

- EJB 환경 변수들.
- EJB 참조.
- Resource 참조.
- 관리되고 있는 object들에 대한 참조.

이 모든 참조자들은 표준 ejb-jar.xml파일에 정의되어 있다. 그러나 JEUS EJB DD에 있는 실제 시스템 JNDI 이름들에 매핑되어야 한다. 이 매핑은 EJB 종류마다 다르게 논리적으로 구성된다.

### 6.2.10 EJB 클러스터링

EJB를 클러스터링 함으로써 여러 개의 EJB engine에 동일한 EJB를 deploy할 수 있다. 이로 인해 성능 개선을 위한 부하분산과 안정성을 높이기 위한 fail over 기능이 가능하다.

EJB클러스터링에 대해서는 9장에서 자세히 다룬다.

### 6.2.11 HTTP Invoke

클라이언트가 JNDI서비스를 통해서 EJB인스턴스를 찾을 때 클라이언트는 원격으로 EJB 메소드를 호출할 수 있는 EJB RMI 스텁을 받는다. 기본적으로,

스텁과 EJB간의 원격 통신은 스텁으로부터 RMI runtime으로 java.net.Socket 이 연결되어 이루어진다.

그러나 이 연결은 방화벽이 중간에 있을 경우나 직접 RMI runtime port로 접근할 수 없는 환경에선 문제가 될 수 있다. 이 경우 특별한 통신 모드가 필요한데 이것이 HTTP invocation mode이다.

이 모드를 사용할 경우 클라이언트가 stub을 통해 메소드를 호출할 때 RMI 요청을 HTTP로 포장해서 Web server로 보내고 Web server는 RMI handler Servlet (jeus.rmi.http.ServletHandler)으로 요청을 전달 한다. handler Servlet은 HTTP 헤더를 제거하고 RMI runtime으로 요청을 전달 한다.

**참고**: RMI runtime은 반드시 Web server와 handler Servlet에 운영되는 Web container와 같은 머신에 있어야 된다.

요청에 대한 결과도 HTTP 응답에 포함되어 stub에 전달된다.

HTTP invoke 모드는 방화벽을 넘어 EJB를 호출하기 위해 HTTP를 사용하는 것이다.

HTTP invoke 모드는 두 곳에서 설정 할 수 있다:

- EJBMain.xml
- EJB module 의 jeus-ejb-dd.xml

EJBMain.xml에 HTTP invoke모드가 설정 되면 EJB engine 내의 모든 모듈에 적용된다. EJB DD파일에 설정되면 설정된 특정 EJB에만 적용 된다.(EJB DD가 EJBMain.xml에 앞선다)

이 내용은 4장에서 설명하였다.

### 6.2.12 결론

이것으로 JEUS EJB에 대한 소개를 마무리한다. JEUS에서 지원하는 6개의 EJB 종류를 살펴보았다. 설정 가능한 bean 특성들의 목록과 어떻게 이 특성들이 특정 bean 종류에 매핑되는지 보았다. 또한 6개의 bean종류에서 공통적으로 적용 가능한 공통 설정들도 확인하였다. 기본 bean설정, run-as identity, thread ticket pool, 참조 매핑, 보안 연동성, 클러스터링 그리고 HTTP invocation이 그 것이다.

다음 절에서는 기본 설정에 대한 자세한 사항들에 대해서 알아본다.

# 6.3 Enterprise JavaBeans 설정

## 6.3.1 소개

모든 EJB 설정들은 JEUS EJB 모듈 DD 파일인 “jeus-ejb-dd.xml”파일 안에 설정된다.

상위 XML 태그는 bean의 종류에 관계 없이 <jeus-bean>을 사용한다.

다음 절에서는 위의 5종의 EJB XML 부모 태그들에 적용될 수 있는 설정들을 설명한다. <beanlist> 밖에 있는 다른 태그들은 5장에서 이미 설명하였다. 특정 XML 부모 태그(bean 종류)에 대한 설명은 이 문서의 후반부에 나오는 해당 장들을 참고하라.

## 6.3.2 기본적인 환경 설정

다음의 “기본” 설정들은 5종의 모든 Enterprise JavaBeans들에 적용된다. 이 설정들은 모두 JEUS 전용 설정들이고, JEUS EJB모듈 DD파일(jeus-ejb-dd.xml)에 선언되어 있다. 이 파일은 5장에서 설명하였다.

- **<ejb-name>** 표준 ejb-jar.xml DD파일에서 선택한 이름과 동일한 이름이다.
- <**export-name**> JNDI에 등록될 시스템 내부의 유일한 이름을 가져야 한다.
- <**local-export-name**> JNDI에 등록될 시스템 내부의 유일한 이름을 가져야 한다.
- <**export-port**> RMI service listening port를 명시적으로 설정할 필요가 있을 때에 이 설정을 사용할 수 있다. 방화벽을 사용하고 있어서 특정 port로만 시스템접근을 허용할 경우에 유용하다. 이 태그의 설정은 방화벽 설정의 “allowed”로 설정된 포트와 동일한 것이어야 한다.
- **<export-iiop>**스위치는 활성화되어 있을 경우에 bean의 interface가 IIOP stub과 skeleton으로서 COS naming server에 export될 수 있도록 한다. 이는 IIOP로 접근 가능한 모든 클라이언트가 bean에 접근 가능하도록 한다.
- <**single-vm-only**> “true”로 설정되어 있으면 JNDI server는 bean이 실행되고 있는 JVM에서만 bean이름이 export될 수 있도록 범위를 한정시킨다. 이 의미는 해당 bean에 접근 할 수 있는 클라이언트는 동일한

JVM내에서 운영되고 있는 Servlet과 bean들 뿐이라는 것이다.(즉, 그 bean은 현재의 engine container에서만 볼 수 있다는 것이다). 이 옵션은 같은 bean들이 다른 engine container에도 deploy 되어 있을 경우에 유용하게 사용될 수 있다. 일반적으로, bean들이 같은 JNDI export name을 가지고 있을 경우에는 이 이름들이 export될 때 서로를 JNDI naming server에서 덮어 쓰려한다. 이 옵션을 사용함으로써, 각 bean들은 bean을 운영하고 있는 JVM에서 자체적으로 인식되고, 범위가 한정되므로 export name을 덮어쓰지 않는다.

- <**local-invocation-optimize**> 이 Boolean설정은 "EJB local interface"의 대안으로 할 수 있는 설정이다. 활성화 되면(기본값) 같은 EJB engine에서 진행되는 모든 method 호출은 표준 RMI를 사용하지 않고 더 빠른 표준 Java method invocation을 사용하게 된다.

  **경고**: local invocation optimization이 사용될 때에는 호출이 call-by-reference로 일어난다(표준 RMI는 call-by-value이다). 그러므로 이 사실을 이용해서 call-by-reference 형식으로 EJB코딩을 해서는 안 된다. 모든 method 호출은 call-by-value인 것처럼 취급한다.

- <**fast-deploy**> 이 Boolean설정은 이 bean의 ObjectImpl, HomeImpl 클래스와 이들의 RMI stub, skeleton 클래스가 deploy 전에 이미 생성되어 있는지의 여부를 지정한다. true로 설정되면 이들 클래스를 deploy시에 따로 생성하지 않는다. 기본값은 false이다. 이 값이 설정되어 있지 않더라도 jeus-application-dd.xml이나 ejbadmin에서의 –f option이 설정되어 있다면 fast deploy를 수행하게 된다.

- **<use-access-control>** 이 Boolean 설정은 bean method들을 수행할 때 J2EE의 principal에 따라 method의 수행이 J2SE security manager의 감시를 받도록 할것인지를 결정한다. 자세한 것은 JEUS security manual과 JACC specification을 참조하기 바란다. 기본값은 false이다.

다음의 예는 어떻게 위의 설정들이 Bean Managed Persistence Entity Bean (BMP)에 적용되는지 보여준다.

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
    . . .
    <beanlist>
```

```
        <jeus-bean>
            <ejb-name>account</ejb-name>
            <export-name>ACCOUNTEJB</export-name>
            <local-export-name>
                LOCALACCOUNTEJB
            </local-export-name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            <single-vm-only>true</single-vm-only>
            <local-invoke-optimize>true</local-invoke-optimize>
            <fast-deploy>true</fast-deploy>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 6.3.3 Run-as Identity 환경 설정

<run-as-identity> XML 태그 내에 설정되는 run-as-identity는 7장에서 설명 한다.

### 6.3.4 보안 상호운용

보안 상호운용은 8장에서 설명 한다.

### 6.3.5 Thread Ticket 설정

EJB의 thread ticket은 bean의 최상위 XML 태그의 <thread-max>에서 설정한다. 이 값은 각 EJB를 수행하는 thread의 최대 허용 개수, 즉 최대 thread ticket의 수를 의미한다. 대기하고 있는 클라이언트의 요청수가 이 값보다 클 경우에는 새로운 요청들은 먼저 진행중인 작업이 끝나고, pool로 thread ticket이 반환될 때까지 대기해야 한다.

다음은 BMP bean에 위의 설정을 적용시킨 예이다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
```

```
            . . .
            <thread-max>200</thread-max>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 6.3.6 External Reference 설정과 매핑

ejb-jar.xml에 선언된 모든 external reference들은 실제 시스템 JNDI export-name과 매핑되어야 한다. 이 매핑은 JEUS EJB DD 내부의 네 가지 다른 XML 태그로 설정된다[표 4].

*표 4. ejb-jar.xml 과 JEUS EJB DD 파일에서 참조 태그들과의 관계*

| ejb-jar.xml XML 태그 이름 | 관련된 JEUS EJB DD XML 태그 | 설명 |
|---|---|---|
| <env-entry> | <env> | EJB 환경 설정을 정의하거나 무시 |
| <ejb-ref> | <ejb-ref> | EJB reference를 실제 export name로 binding |
| <resource-ref> | <res-ref> | external resource manage reference를 실제 export name에 binding |
| <resource-env-ref> | <res-env-ref> | 관리되고 있는 object reference를 실제 export name에 binding |

모든 <env-entry> 태그는 다음과 같은 하위 태그들을 가진다.

- <**name**> env 의 이름. ejb-jar.xml에 정의된 <env-entry-name> 태그의 와 같은 값을 사용한다.

- **<type>** 값의 자료형에 해당하는 완전한 Java 클래스 이름 (기본 자료형은 wrapper 클래스를 사용한다).

- **<value>** env 의 실제 값을 선언한다.

다른 세 태그들(<ejb-ref>, <res-ref>, <res-env-ref>)은 여러 개의 <jndi-info> 태그를 가진다. 이 태그들은 또 JNDI export name에 레퍼런스를 매핑하기 위해서 다음과 같은 하위 태그들을 가진다.

- **<ref-name>** EJB 소스 코드와 ejb-jar.xml에서 정의된 참조의 이름. ejb-jar.xml 에서 <ejb-ref> 와 동격의 태그는 <ejb-ref> 이다. <resource-ref> 와 동격은 <res-ref> 이고, <resource-env-ref> 와 동격은 <resource-env-ref> 이다.

- **<export-name>** JEUS JNDI 서버에 entity또는 object가 export될 때 사용되는 시스템 의존적인 이름.

"<ref-name>"과 "<export-name>" 태그는 반드시 ejb-jar.xml에서 주어진 이름과 같은 값이어야 한다.

아래는 모두 네 개의 external reference 가 JNDI 매핑된 XML의 예이다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>ejb/AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
            </ejb-ref>
            <res-ref>
```

```
                <jndi-info>
                    <ref-name>jdbc/AccountDB</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
            </res-ref>
            <res-env-ref>
                <jndi-info>
                    <ref-name>jms/StockQueue</ref-name>
                    <export-name>STOCKQUEUE</export-name>
                </jndi-info>
            </res-env-ref>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 6.3.7 클러스터링 환경 설정

클러스터링 환경은 9장에서 논의한다.

### 6.3.8 HTTP Invoke 환경 설정

EJB모듈 중 특정 EJB에 HTTP invocation을 설정하고 사용하기 위해선 jeus-ejb-dd.xml의 <jeus-bean>하위에 <invoke-http> 을 설정 해야 한다.

EJB engine에서 HTTP invocation이 필요하면 EJBMain.xml에서 <invoke-http>를 추가한다(4장에서 언급). EJBMain.xml의 설정은 모든 모듈에 적용되는데, 각 모듈의 DD파일에 <invoke-http>가 있다면 DD파일의 설정을 사용한다.

설정에는 두 개의 하위 태그가 있다:

- **URL**: 반드시 HTTP-RMI스텁으로부터 호출되는 RMI handler Servlet의 URI (jeus.rmi.http.ServletHandler)를 입력한다. 이 URI에서는 프로토콜, IP, 포트를 제외한 Servlet 요청 경로만을 넣는다. 프로토콜은 “HTTP”로 IP는 RMI runtime과 같은 주소로 간주된다. 이 말은 HTTP-RMI요청을 받은 웹 서버와 Web container가 RMI runtime과 같은 머신에 있어야 된다는 것이다. 그러면 RMI runtime 의 주소는 RMI 스텁에게 알려지게 된다. 웹서버의 포트는 반드시 다음 “HTTP port”에 설정 해야 한다.

- **HTTP port**: HTTP-RMI요청을 받고 처리할 웹 서버 또는 Web Container를 설정해야 한다. 이 웹서버/Web container 는 반드시 RMI handler Servlet이 Deploy되어 실행되고 있어야 한다.

예제:

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <invoke-http>
                <url>
                    /mycontext/RMIServletHandler
                </url>
                <http-port>
                    80
                </http-port>
            </invoke-http>
            . . .
        </jeus-bean>
    </beanlist>
    . . .
</jeus-ejb-dd>
```

HTTP invoke가 정상적으로 작동하기 위해선 RMI handler Servlet이 deploy 되어 있어야 한다. RMI handler Servlet을 구현한 클래스는 jeus.rmi.http.ServletHandler이다. jeus.rmi.http.ServletHandler은 context내에 <invoke-http>에 설정된 URL과 같게 deploy 되어야 한다. deploy에 대한 것은 JEUS Web Container 안내서를 참조 한다.

### 6.3.9 결론

이것으로 모든 EJB의 종류에 공통인 기본 설정에 대한 설명을 마친다. 기본 설정에는 run-as identity, thread ticket pool, external reference mapping, security interoperability, bean clustering이 있다( 몇몇 항목은 나중에 더 자세히 다룬다).

다음 절에선 deploy된 EJB를 모니터링 하는 방법을 알아보겠다.

# 6.4 Enterprise JavaBeans 모니터링

### 6.4.1 소개

개별 EJB를 제어할 방법은 없다 (5장에서도 언급 했듯이 EJB 모듈을 제어해야 한다). 그러나 ejbadmin이나 웹 관리자를 이용해서 모니터링은 가능 하다 (웹 관리자를 이용하는 것을 권한다).

### 6.4.2 Console 이용

ejbadmin에선 개별 EJB의 정보를 볼 수 있는 명령어는 “beanlist” 하나 밖에 없다. ejbadmin이 기동되어 있고 “countermod”라는 EJB가 deploy 되어 있다고 가정하면, 아래와 같은 명령을 내릴 수 있다.

```
johan_ejb_engine1> beanlist countermod
```

결과로 EJB모듈에 등록되어 있는 개별 빈들의 상태와 이름을 볼 수 있다.

다음 명령으로 EJB 모듈내의 각 빈들의 이름과 상태를 모니터링 할 수 있다.

```
johan_ejb_engine1>beaninfo mybean
```

부록 A 에 자세한 설명이 나와 있으니, 참고하기바란다.

### 6.4.3 결론

이 절에선 EJB의 모니터링에 관한 핵심을 소개했다.

다음 절에선 JEUS에서 EJB 성능에 관계된 부분을 알아보겠다.

# 6.5 Enterprise JavaBeans 튜닝

### 6.5.1 소개

모든 EJB 종류에 공통으로 EJB 운영 성능을 향상(또는 시스템 리소스의 낭비를 막기 위해)시키기 위해 적용시킬 수 있는 몇 가지 설정이 있다:

- 같은 EJB engine에 존재하는 모든 bean들의 JEUS EJB DD 태그 중 <local-invocation-optimize>를 “true”로 설정한다.
- Thread ticket pool의 max 값을 적절히 수정한다.
- 여러 개의 EJB engine에 bean들을 클러스터링해서 설정한다. 9장을 참고하라.

- 특정 bean-instance에 적용되지 않는 “static” 형태의 method들은 home method를 사용하라(일반적인 프로그래밍 팁).
- JDBC connection pool을 바로 설정한다. JEUS Server 안내서 참고.
- engine 부팅과 EJB 모듈 deploy 시간을 절약하기 위해 fast-deploy 옵션과 EJB compiler를 사용한다. 5장을 참고하라.

위의 기본적인 최적화 옵션뿐만 아니라 빈의 종류에 따라 수 십 개의 튜닝 팁이 존재한다. 특히 entity bean에 해당하는 이야기라고 할 수 있다. 그러므로, 각 EJB 종류에 따른 튜닝과 최적화 하는 방법은 해당하는 각 장에서 찾아보기 바란다.

### 6.5.2 Thread Ticket Pool 설정 튜닝

thread ticket pool은 올바르게 설정되어야 한다. 다음과 같은 원칙이 적용될 수 있다.

Max pool 크기를 크게 잡을수록 더 많은 thread ticket들이 생성될 수 있고, 따라서 더 많은 클라이언트 요청을 수행할 수 있게 된다. 그러나 크게 잡힌 max pool 크기는 당연히 많은 메모리 자원을 소모하게 된다(많은 클라이언트들이 EJB engine에 접속한 경우에).

**중요**: max pool 크기는 thread ticket pool에서 가장 중요한 파라미터이다. 그러므로, 성능 개선을 위해서는 이 값을 가장 먼저 조절하고 튜닝해야 한다.

특정한 시간대에 bean의 서비스 사용을 요청하는 클라이언트가 급격히 증가할 경우에는 step 값을 증가시켜 새로운 thread ticket들이 “batch”로 생성될 수 있도록 설정한다.

Thread ticket pool 설정은 특정한 EJB에 대하여 접근 가능한 양을 조절할 수 있는 밸브로 작용한다.

### 6.5.3 결론

여기까지 JEUS에서 EJB를 튜닝할 때 필요한 기본적인 것들을 살펴보았다. 다음 장에서는 특정 EJB 에 따른 튜닝 상세 정보를 다룬다.

## 6.6 결론

JEUS의 일반적인 Enterprise JavaBeans에 대한 설명은 다 했다.

EJB의 기본적인 설정 특성들은 모두 살펴보았고 EJB종류에 상관없이 어떻게 모니터링하며 튜닝하는지도 살펴보았다.

이 문서의 나머지 부분에서는 EJB에 관련된 특정 기술적인 요소들에 대해 살펴보고, 나중에는 각 EJB 종류(session, entity, MDB bean )에 따른 특수한 요구사항들과 쟁점에 대해 설명하겠다.

다음 장은 JEUS에서의 기본적인 EJB보안 설정에 대해 알아본다. 그리고 이전에 나왔던 run-as identity 항목에 대해서도 알아본다.

# 7 Enterprise JavaBeans 의 보안

## 7.1 소개

이 장에서는 JEUS 에서 어떻게 EJB를 위한 보안 설정을 할 수 있는지 살펴본다.

여기에서 말하는 보안은 인증(사용자와 암호)과 권한부여(Access Control Lists)를 말한다.

JEUS 보안에 대한 상세한 정보는 JEUS Server 안내서를 참조하라.

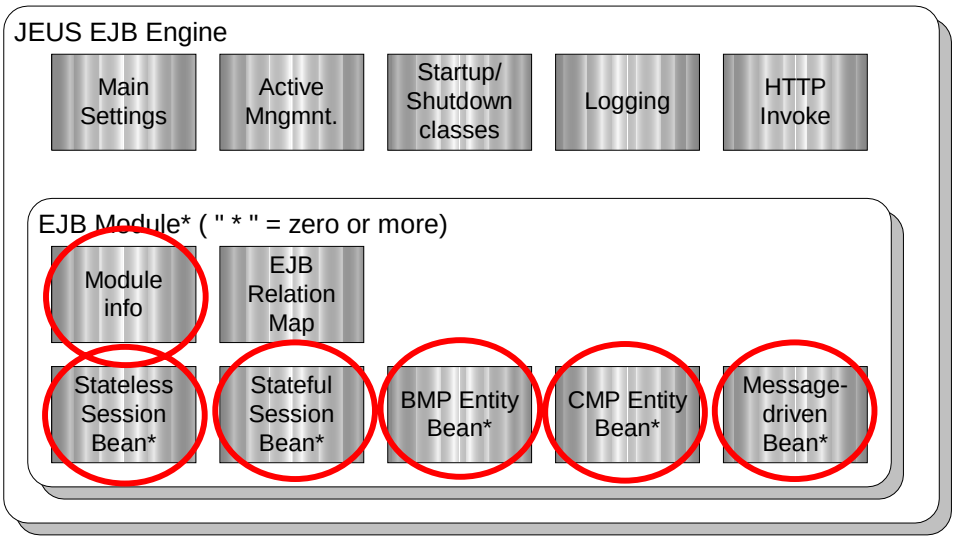


*그림 17 EJB 모듈의 보안관련 설정의 위치*

.

## 7.2 EJB 보안 개요

### 7.2.1 소개

JEUS EJB의 보안 설정에 관련된 세 가지 항목을 살펴보기로 하자.

- 한 EJB 모듈에 한정된 role-to-user 매핑.

- run-as identity 설정.
- JEUS security 설정

### 7.2.2 EJB Module Role Assignment

위에서, EJB모듈과 설정을 설명할 때, 모든 JEUS EJB DD의 최상위에서 설정한 role assignment를 볼 수 있다.

이 role assignment 는 표준 ejb-jar.xml파일에 개발자가 정의한 역할과 실제 JEUS 보안 도메인의 주체와 그룹 이름을 매핑하는 것이다 (아래의 JEUS security 설정을 참조).

### 7.2.3 Run-as Identity 설정

jeus-ejb-dd.xml DD 파일에는 <run-as-identity>라 불리는 XML 설정 태그가 있다. 이 태그는 ejb-jar.xml의 <run-as-specified-identity>에서 개발자가 정의한 역할 이름과 JEUS 보안 도메인 설정에서 지정한 실제 Principal의 매핑을 제공한다 (예, subjects.xml 파일에서). 이 Principal은 일반적으로 간단한 사용자 이름을 사용한다.

### 7.2.4 JEUS security 보안 영역 설정 파일

JEUS Security 안내서에서는 JEUS의 보안 설정을 설명하고 있다. 이 설정 중 EJB에 대해 가장 중요한 설정이 subjects.xml이다. (이 파일은 JEUS_HOME\config\security\<security domain name>\에서 찾을 수 있다) 이 파일은 실제 JEUS Principal과 Role의 mapping 리스트를 가지고 있다.

위에서 언급했듯이, 이 파일에 지정된 모든 사용자 이름과 그룹들은 JEUS EJB DD를 통하여 ejb-jar.xml에서 개발자가 정의한 역할들과 매핑이 되어 있어야 한다.

**참고**: 물론 Database에도 security xml file들에 포함되는 내용을 저장할 수 있다. 그러나 이번 장에서는 subjects.xml에 저장했다고 가정한다. JEUS Security 안내서에서 자세한 설명을 볼 수 있다.

### 7.2.5 결론

JEUS에서 EJB 보안을 설정할 때 세 가지 주요 항목들을 살펴보았다. EJB module role assignment, run-as identity 매핑, 그리고 JEUS 보안 영역 설정파일인 subjects.xml이 그것이다.

다음에서는 이 항목들에 대한 자세한 설정을 설명한다.

# 7.3 EJB 보안 설정

## 7.3.1 소개

이 장에서는 다음 세 항목의 설정에 관련된 자세한 내용들을 알아본다.

- EJB 모듈의 역할 할당(role assignment)
- EJB run-as identity 설정
- JEUS 보안 영역 설정

## 7.3.2 역할 할당(Role Assignment) 설정

EJB 모듈의 역할 할당 설정은 jeus-ejb-dd.xml 파일의 <module-info> 아래 <role-permission> 태그에서 한다. 두 개의 필수 최하위 태그는 다음과 같다.

- **<role>** ejb-jar.xml 파일에 지정된 것과 동일한 role name이 설정되어야 한다.
- **<principal>** subjects.xml에서 정의된 principal name과 동일해야 한다. "role name-user name"쌍이 개발자가 정의한 역할과 JEUS 주체간의 매핑을 정의한다. 하나의 역할 이름에 복수 개의 사용자 이름을 설정할 수 있다.

예)

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    <module-info>
        <role-permission>
            <role>manager</role-name>
            <principal>peter</user>
        </role-permission >
    </module-info>
    <beanlist>
       . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

**참고**: 이 설정은 bean이 deploy되었을 경우에만 유효하다.

### 7.3.3 Run-as Identity 설정

Run-as identity설정은 jeus-ejb-dd.xml에서 bean 설정 수준과 같은 <run-as-identity> 태그에서 설정된다. 이 태그의 하위 태그는 다음과 같다.

- **<principal-name>** ejb-jar.xml 파일의 <run-as-specified-identity> 태그에서 주어진 role에 mapping하고자 하는 principal name으로 subjects.xml에서 설정되어 있어야한다.

예)

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 7.3.4 완전한 보안 설정의 예

다음의 XML 문서의 일부분들은 세 개의 파일들(ejb-jar.xml, jeus-ejb-dd.xml, subjects.xml)에서 가져온 것이다. 이 파일들에서 서로 동일해야 하는 부분들은 굵은 글자로 강조하였다.

<<*ejb-jar.xml*>>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
    <enterprise-beans>
        <session>
```

```
            <ejb-name>customer</ejb-name>
            <home>customer.CustomerHome</home>
            <remote> customer.Customer </remote>
            <ejb-class>customer.CustomerEJB </ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>container</transaction-type>
            . . .
            <security-role-ref>
                <role-name>customer</role-name>
                <role-link>customerMgmt</role-link>
            </security-role-ref>
            <security-identity>
                <use-caller-identity />
            </security-identity>
            . . .
        </session>
        <session>
            <ejb-name>EmployeeService</ejb-name>
            <security-identity>
                <run-as-specified-identity>
                    <role-name>admin</role-name>
                </run-as-specified-identity>
            </security-identity>
        </session>
        . . .
</enterprise-beans>
<assembly-descriptor>
    <security-role>
        <role-name>customerMgmt</role-name>
    </security-role>
    <method-permission>
        <role-name>customerMgmt</role-name>
        <method>
            <ejb-name>customer</ejb-name>
            <method-name>*</method-name>
            <method-params />
        </method>
   </method-permission>
   . . .
```

```
    </assembly-descriptor>
    <ejb-client-jar>client.jar</ejb-client-jar>
</ejb-jar>
```

위의 XML 예에서는 두 개의 session bean들을 선언하였다. 첫 번째 bean(“customer”)은 “customerMgmt” 역할에 연결되는 보안 역할을 가지고 있다. 두 번째 session bean(”EmployeeService”)은 “admin”이라는 <run-as-specified-identity>역할을 선언하고 있다. “customerMgmt”와 ”admin” 역할은 EJB Deploy시에 실제 시스템 주체들과 매핑되어야 한다. 이 매핑은 다음 XML 문서에 예시되어 있다.

<<*jeus-ejb-dd.xml*>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
        . . .
        <role-permission>
            <principal>customer</principal>
            <role>customerMgmt</role>
        </role-permission>
    </module-info>
    <beanlist>
        <jeus-bean>
            <ejb-name>EmployeeService</ejb-name>
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

위의 예에서는 어떻게 “customerMgmt”와 “admin” 역할이 실제 시스템 주체 이름들(각각 “customer”와 “peter”)에 매핑되는지 볼 수 있다. 다음 예에서는 어떻게 이 사용자들이 subjects.xml에 설정되는지 살펴본다.

<<*subjects.xml*>>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<subjects xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

```
    <subject>
        <description>No description</description>
        <principal>
            <classname>jeus.security.resource.PrincipalImpl</clas
sname>
            <name>customer</ name>
        </principal>
        <credential>
            <classname>jeus.security.resource.PasswordFactory</cl
assname>
            <property>
                <name>password</name>
                <value>dds674mf==</value>
            </property>
        </credential>
    </subject>
    <subject>
        <description>No description</description>
        <principal>
            <classname>jeus.security.resource.PrincipalImpl</clas
sname>
            <name>peter</name>
        </principal>
        <credential>
            <classname>jeus.security.resource.PasswordFactory</cl
assname>
            <property>
                <name>password</name>
                <value>amV1czEyMw==</value>
            </property>
        </credential>
    </subject>
</subjects>
```

그러므로, 위의 예에서 “customer”과 “peter” 라는 사용자들의 정의를 볼 수 있다. 이 사용자들은 jeus-ejb-dd.xml 파일을 통하여 ejb-jar.xml 파일의 “customerMgmt”와 “admin” 역할과 연관되어 있다.

### 7.3.5 결론

여기서는 역할 할당, run-as identity,그리고 해당하는 subjects.xml 파일을 살펴보았다. 앞의 두 태그에서의 설정은 subjects.xml에서 지정된 설정과 일치해야 하며 표준 EJB DD(ejb-jar.xml)와도 일치해야 한다.

## 7.4 결론

이로써, 기본적인 JEUS EJB의 보안 설정에 대한 설명을 마무리 짓는다. 우리는 EJB 모듈 역할 할당, run-as identity 그리고 subjects.xml파일의 주요 컴포넌트들을 살펴보았다. 그들이 어떤 것인지, 어떻게 연관되어 있고 어떻게 설정되는지도 살펴보았다.

다시 한 번 말하지만, JEUS 보안에 대해서는 JEUS Security 안내서를 참고하라.

다음 장에서는 특별한 종류의 보안 환경설정에 대해 알아 보겠다. 즉, JEUS EJB engine이 다른 벤더의 J2EE 플랫폼과 연동시의 보안 상호 운용 설정에 대해 알아보겠다.

# 8 Enterprise JavaBeans 의 상호 운용

## 8.1 소개

EJB 2.1에서 EJB container들은 EJB에 RMI-IIOP프로토콜로 접근 하는 것을 허용해야 한다. 또한 접근 보안, 트랜잭션 등을 위해 여러 상호운용 서비스가 지원되어야 한다.

## 8.2 EJB 상호 운용의 개요

EJB 2.1에서는 다른 벤더들의 상호 운용성을 위해 CORBA를 적용했다 JEUS EJB container의 서비스 중에, 트랜잭션 상호 운용을 위한 CORBA Object Transaction Service (OTS) 와 보안 상호 운용을 위한 CORBA Common Secure Interoperability (CSI)를 여기서 논의 하겠다.

### 8.2.1 Transaction 상호 운용

JEUS EJB Container는 CORBA Object Transaction Service와의 트랜잭션 상호 운용을 지원하지 않는다. EJB container로부터 시작된 Global Transaction Propagation은 다른 벤더들로부터 거절당한다. 이것에 대한 것은 EJB2.1스펙을 참조하라.

### 8.2.2 Security 상호 운용

CSI 에 의한 Security 상호운용은 JEUS EJB container에서 모두 지원한다. EJB home의 Interoperable Object Reference (IOR)가 JEUS JNDI server에 등록이 되면, CSI 를 위한 보안 정보가 첨부된다. 이 정보를 바탕으로, 클라이언트 측의 ORB와 JEUS container가 IIOP 프로토콜의 보안 레벨에서 negotiation을 처리한다. 또한 JEUS EJB Container는 CSI서비스를 이용해서 IIOP를 통해 다른 EJB빈을 호출할 경우 클라이언트로 동작한다. 다음 절에서 이 설정에 대해 설명할 것이다. client-side ORB와 JEUS EJB Container간의 상호 운용에 대해선 CSI v2 specification of CORBA를 참조 하기 바란다.

# 8.3 EJB 상호운용 설정

## 8.3.1 EJB 보안 상호운용 모듈의 설정

JEUS EJB 보안 상호운용 모듈을 활성화 시키기 위해서, JEUSMain.xml의 <engine-container> 태그의 <enable-interop> 값을 true로 해야 한다. 만약 모듈이 활성화되지 않았다면, CSI 프로세스는 작동하지 않을 것이다.

CSI스펙에 있는 프로토콜을 수행하기 위해서 두 가지 추가적인 보안 환경 파일과 JEUS security manager에 있는 정보를 이용한다. 두 파일 중 keystore 라고 불리는 파일은 X.509를 위한 저장 파일이고 Sun Microsystems에 의해서 공급된 X.509 keystore 구현 파일이다. Secure Socket Layer (SSL) 이 호출 됐을 때 이 파일이 클라이언트에게 보내 진다. truststore 로 불리는 다른 파일은 X.509클라이언트 측 증명 설정 파일이고 형식은 Keystore와 같다. [표 5]에서 보는 것은 이를 위한 옵션이다. 물론 이 값은 –D를 이용해서 설정된다.

*표 5. keystore 와 truestore 파일들에 관련된 JVM –D 파라미터들.*

| 파라미터 | 설명 |
|---|---|
| jeus.ssl.keystore | keystore 파일까지 절대 경로. 설정하지 않을 경우, 기본값은 'JEUS_HOME/config/nodename/keystore' 이다. |
| jeus. ssl.keypass | Keystore파일에 주어진 password 값. 정의 되지 않았으면 'jeuskeypass'이다. |
| jeus. ssl.truststore | truststore 파일까지 절대 경로. 정의되지 않았으면, 기본값은 'JEUS_HOME/config/nodename/truststore' 이다. |
| jeus. ssl.trustpass | truststore 파일에 주어진 password. 설정되지 않았으면 , 기본 값은 'jeustrustpass'이다. |

호출자의 authentication 과 authorization 이 신뢰하는 노드 사이에서는 불필요 할 때도 있다. jeus.ejb.csi.trusthosts 값에 IP주소를 설정함으로써 신뢰하는 노드의 불필요한 보안 점검을 피할 수 있다. JEUS security manager는 불특정 접근자에게 anonymous principal을 나타내는 'guest'을 사용한다.

### 8.3.2 EJB Bean 에 IOR 설정

EJB 빈은 CSI정보를 IOR에 더함으로써 클라이언트에게 보안에 대한 수준을 전달할 수 있다. 정보는 각 빈의 <security-interop> 아래 설정된다. [표 6]을 보면 IOR에 있는 정보와 jeus-ejb-dd.xml의 태그의 매핑 정보를 볼 수 있다. IOR의 환경은 CSIv2 specification을 참조 한다. JEUS용 태그는 부록 E를 참조 한다.

*표 6. IOR 비트에서 JEUS XML 태그로의 매핑.*

| IOR 의 설정 비트 | JEUS 태그 |
|---|---|
| TLS_SEC_TRANS 의 무결성(intergrity)과 신뢰성(confidentity) 비트 | integrity-confidentiality |
| TLS_SEC_TRANS | trust-in-client |
| AS_ContextSec | client-auth |
| SAS_ContextSec | identity-assertion |

### 8.3.3 Client-side Principal 설정

JEUS의 어플리케이션에서 다른 벤더의 EJB Container로 요청을 보낼 때, principal 은 보안 상호운용을 사용한다. 더 나아가, 보안 상호운용을 위한 principal 은 Java Authentication Authorization Service (JAAS)를 이용한다. JEUS에 의한 javax.security.auth.spi.LoginModule 구현은 jeus.ejb.csi.login.EJB CSILoginModule에서 되어 있다. 어플리케이션의 보안 환경 통합에 대해서 JAAS 스펙을 참조하라.

증명서(Credential)에는 두 가지 타입이 있다. 하나는 username 과 password 이다. 서버 측 보안 시스템이 보안 도메인(Security Domain)들을 관리하는 Realm이 여러 개라면, username을 포함하는 Realm의 이름을 지정해야 한다. 다른 하나의 타입은 X.509 credential이다 .

아래의 [표 7] 은 위 파일을 위해서 사용되는 JVM option 을 보여 준다.–D와 같이 사용된다.

*표 7. 두 가지 credential 타입들과 관련된 JVM 파라미터들.*

| **Credential 타입** | **-D 파라미터** | **설명** |
|---|---|---|
| Username-Password | jeus.ejb.csi.username | username의 문자열 값 |
| | jeus.ejb.csi.realm | realm의 문자열 값 |
| | jeus.ejb.csi.password | password의 문자열 값 |
| X.509 Credential | jeus.ejb.csi.certalias | Keystore내의 X.509 credential의 별칭 문자열값. |

## 8.4 결론

JEUS EJB container는 IIOP 프로토콜을 이용해서 EJB의 상호 운용성을 지원한다. 트랜잭션 관련한OTS는 지원하지 않는다. CSI이용을 위해서는 EJB 보안 상호운용 모듈이 활성화 되어 있어야 하며 두 가지의 환경 정보 파일이 있어야 한다. 각 CSI와 관련된EJB는 보안 상호 운용을 위한 설정 태그를 갖고 있어야 한다.

# 9 Enterprise JavaBeans 클러스터링

## 9.1 소개

EJB에 fail over와 부하 분산기능을 주기 위해서는 각 bean들을 여러 EJB engine에 deploy되어 클러스터링을 형성해야 한다. 이 장에서는 이러한 개념에 대하여 설명하고자 한다.

클러스터링은 컴포넌트 레벨(개별 bean)에서 수행되고 SL, SF, entity bean에서 수행 된다. 즉 MDB는 해당 되지 않는다 [그림 18].

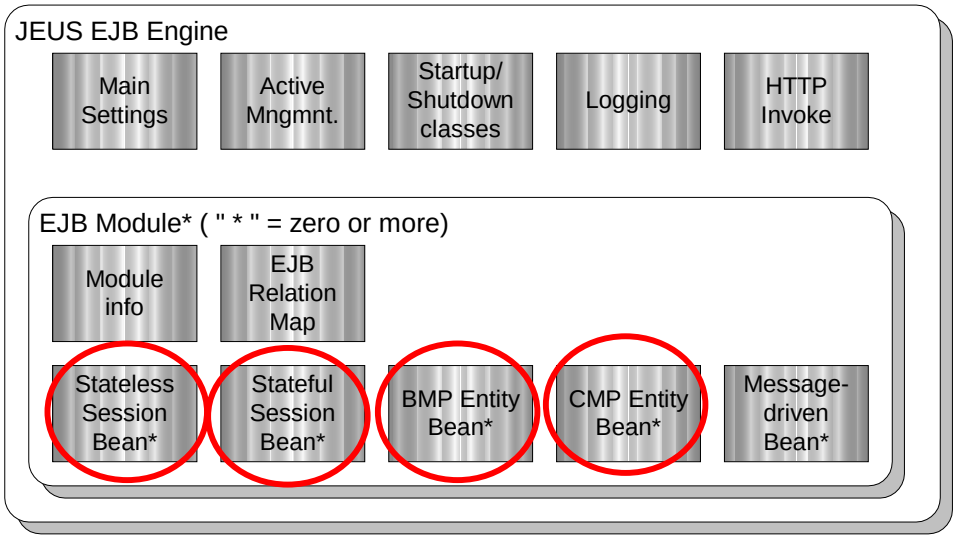


*그림 18. EJB 모듈에서 클러스터링 위치 정보*

## 9.2 EJB 클러스터링의 개요

### 9.2.1 소개

JEUS EJB 클러스터링에는 두 가지의 논점이 대두된다.

1. **Fail over**: bean의 method 실행 전과 실행 중에 한 EJB 인스턴스나 EJB engine이 멈추면 다른 EJB engine의 EJB 인스턴스에서 모든 요청이 처리된다.

2. **부하 분산**: round-robin 부하 분산 알고리즘을 이용하여 클러스터링 된 bean의 전체적인 성능과 응답속도가 향상될 수 있다.

위의 두 기능과 관련된 “idempotent” bean method의 존재에 대하여 설명할 필요가 생긴다.

먼저, EJB 클러스터링의 기본적인 구조적 항목들을 살펴보자.

### 9.2.2 EJB 클러스터링 아키텍처

[그림 19] 는 JEUS에서 어떻게 EJB 클러스터링이 동작하는지 보여준다.

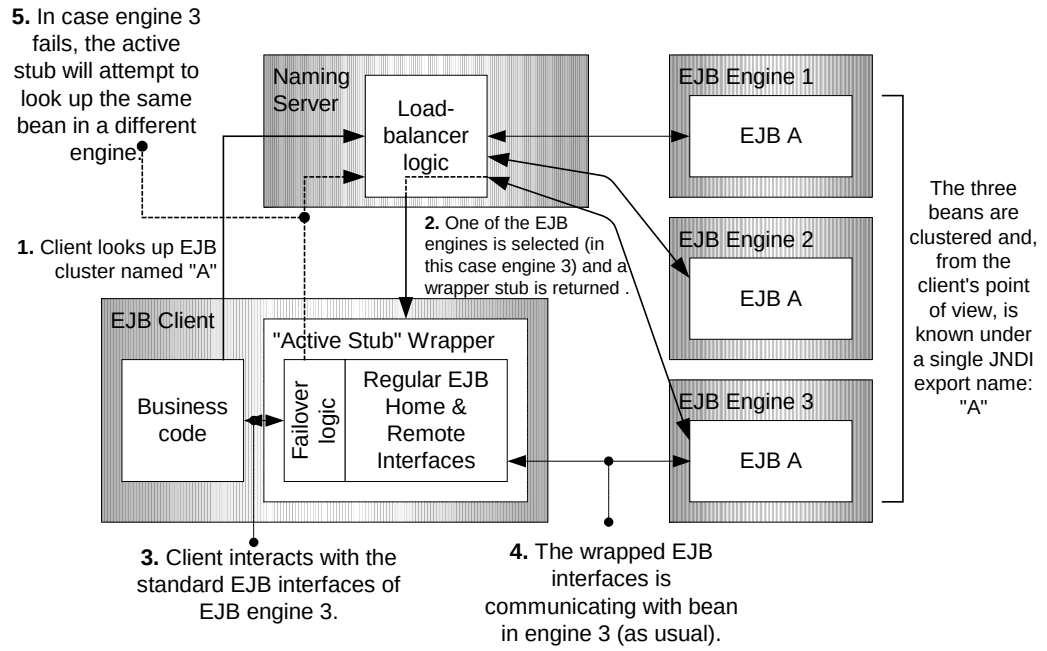


*그림 19. 개념적인 EJB 클러스터링 아키텍처.*

개념적인 EJB 클러스터링 아키텍처.이 그림은 두 가지 EJB클러스터링의 주요 기능인 부하 분산과 fail over를 설명한다.

### 9.2.3 부하 분산

[그림 19 ]에서와 같이 부하 분산은 실제적으로 JNDI Naming Server 내에서 수행되고 있다. 클라이언트에게는 클러스터링된 모든 bean들이 같은 이름으로 인식된다 (예에서는 “A”). JNDI Naming Server는 실제적으로 그 이름이 서로 다르지만 클러스터링 되어 서로 다른 EJB engine 내부에 존재하는 연결된 EJB 라는 것을 알고 있다.

클라이언트가 “A” bean을 요청할 때, Naming Server는 세 EJB engine에 존재하는 세 개의 bean 중 하나를 선택하여 반환한다. bean은 round-robin 방식으로 선택된다.

클라이언트는 그 후부터는 선택된 EJB engine에서, 전달된 EJB home과 remote 인터페이스를 통하여 bean과 일반적인 방법으로 연동하게 된다 ().

위의 이야기는 세 개의 bean들은 공평하게 같은 method호출 요청을 받고, 그럼으로써 잠재적으로 한 개의 engine이 모든 요청에 대해 서비스하는 것보다 무려 세 배의 시스템 성능 향상을 기대할 수 있게 된다 (부하 분산 시 발생하는 작은 자원소모를 계산하지 않을 때).

### 9.2.4 Fail over (EJB 복구)

Fail over는 한 EJB 서비스의 장애가 발생하더라도 정상적인 서비스를 제공하는 것을 의미한다 (예, OS 장애, 네트워크 중단 또는 EJB engine 장애). JEUS 시스템이 처리할 수 있는 장애 복구에는 두 가지가 있다.

- 클라이언트의 요청이 도착했을 때 접근 불가한 bean에게 요청을 보내지 않고, 처리되지 않은 요청을 다른 bean에게 다시 보내는 방법.
- 실행중인 bean이 어떤 이유로 runtime 오류를 가지고 있다면 진행중인 요청을 다른 bean에게 다시 보내는 방법.

위의 두 시나리오의 차이는 언제 오류 상황이 발견되었느냐에 있다. Remote EJB 업무 method를 호출하기 전인가 아니면 bean이 요청을 처리하고 있는 중인가의 예가 있을 수 있다.

첫 번째의 방식은 간단하다. 단순히 문제가 발생한 EJB engine을 제외하고 부하 분산 알고리즘을 실행하면 된다. 새로운 클라이언트의 요청을 처리하기 위해서는 사용 가능한 engine의 bean을 선택하면 된다.

JEUS에서는 fail over의 re-routing이 클라이언트 쪽의 EJB 스텁에서 처리된다. 이 스텁을 “active stub”이라고 부르고 또는 표준 EJB interface를 둘러 싼 wrapper라고도 부른다. 이러한 wrapper를 사용할 때의 다른 점은 현재 연결되어 있는 bean 또는 EJB engine이 문제가 있는지 판단할 수 있는 로직을 가지고 있는 지이다. 이러한 문제점이 발견되면, 살아있는 스텁이 자동으로 JNDI Server에 접속하고 작동하고 있는 EJB engine을 대신하여 새로운 스텁을 요청한다([그림 19 ] 상단의 5번). 이 처리 과정은 클라이언트에게 투명하게 처리된다.

반면에, 두 번째 복구 방법은 약간의 문제가 있다. Bean이 요청을 처리하고 있을 때 문제가 발견되면 bean이 얼만큼 요청을 처리하고 있었는지도 모르고 문제가 발생했을 때 어떤 부작용들을 발생시키고 있었는지도 모른다. 단순하게 클러스터에 존재하는 다른 bean의 같은 method를 호출하는 것은 같은 부작용을 조장할 수 있기 때문에 자칫하면 위험한 결과를 가져올 수도 있다.

이 문제를 명백히 하기 위하여, DB 필드를 1씩 증가시키는 method를 가지는 bean 인스턴스 “A”를 고려해보자. 1이 증가 된 바로 후에 문제가 발생했다. 이럴 경우 단순하게 다른 bean “B”에 있는 같은 업무 method를 다시 호출하면 1이 다시 한 번 증가하게 된다. 결과적으로 DB에 일괄적이지 못하고 잘못된 값을 넣게 된다. 이 결과는 수용할 수 없는 것이다. 이 문제를 해결할 수 있는 방법은 Idempotent method를 선언하는 것이다. 이는 다음 절에서 설명한다

### 9.2.5 Idempotent Method 를 통한 EJB 복구

Idempotent method는 부작용이 없는 “getter” method이다. 이는 우리가 Idempotent method를 호출하면서도 method의 수행 중에 어떠한 상태도 (예, instance 변수, DB 필드 등) 변경되지 않는 것이 보장됨을 의미한다.

앞 절에서의 두 번째 복구 문제를 해결하기 위하여 어떻게 Idempotent method 가 사용되는지는 간단히 확인할 수 있다. 한 method가 Idempotent method 라면 첫 번째 시도 했던 method호출이 실행 중에 실패했더라도 안전하게 다시 호출 가능하다.

그러나, 그 method가 Idempotent method 가 아니라면, 이런 상황에서 대책이 없다. 잘못된 메소드를 계속 실행시키는 것보다 Exception을 던지는 것이 차라리 낫다. 그러므로, Idempotent method를 많이 사용할수록 EJB fail over는 더 잘 작동된다.

한 method가 Idempotent method인지 아닌지 판단하는 공식은 없다. 그러므로 각 EJB업무 method의 상태를 정확히 식별하고, JEUS와 통신하도록 하는 것은 프로그래머나 EJB Deployer에게 달려있다. 다음 장에서 이에 대한 설명을 한다.

### 9.2.6 결론

EJB 클러스터링으로 가능한 두 가지 주요 기술들을 살펴보았다. 향상된 성능을 위한 부하 분산과 안정성을 위한 fail over가 그것이다.

또한, 효과적인 fail over 작업을 위해 “Idempotent method”에 대해서도 개념적으로 살펴보았다.

다음 절에서는 어떻게 이 두 가지를 설정하는지 알아본다.

# 9.3 EJB 클러스터링 설정

## 9.3.1 소개

EJB 클러스터링을 설정하기 위해서는 두 가지를 설명할 필요가 있다.

- 클러스터링으로 구성 될 각 EJB를 위한 JEUS EJB DD의 설정
- 클러스터링으로 구성된 여러 EJB들의 일정한 설정

첫 번째 논점인 각각의 bean들이 클러스터링으로 구성되기 위해 어떻게 설정되는지부터 살펴보기로 한다.

## 9.3.2 단일 EJB DD (각각의 Bean)에 EJB 클러스터링 설정

JEUS EJB 모듈 DD파일(jeus-ejb-dd.xml)에는 클러스터링에 참여하는 각각의 bean들을 위해서 <clustering> 태그 아래에 다음과 같은 설정을 적용할 수 있다.

- **<home-clustered>** EJB home interface의 클러스터링을 전체적으로 활성화 또는 비활성화시킨다.
- **<ejb-home-fail-over>** EJB home interface의 fail over를 활성화시킬 스위치 역할을 한다. “true”로 설정되면 일반 Stub이 아닌, Active Stub이 생성되고 JNDI Server로부터 전달된다.
- **<ejb-home-idempotent-method>** home interface들이 idempotent method들이라는 것을 선언한다(앞 절 참조).
- **<ejb-remote-fail-over>** EJB remote interface의 fail over를 활성화하는 스위치 역할을 한다. “true”로 설정되면 일반 Stub이 아닌, Active Stub이 생성되고 JNDI Server로부터 전달된다.
- **<ejb-remote-idempotent-method** remote interface들이 idempotent method들이라는 것을 선언한다 (이전 절 참고).

예)

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
        . . .
    <beanlist>
        . . .
        <bean-managed>
             . . .
            <export-name>ACCOUNT</export-name>
            . . .
            <clustering>
                <home-clustered>true</home-clustered>
                <ejb-home-failover>true</ejb-home-failover>
                <ejb-home-idempotent-method>
                  <method_name>foo</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
                </ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-method>
                  <method_name>bar</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
                </ejb-remote-idempotent-method>
            </clustering>
            . . .
        </bean-managed>
         . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 9.3.3 EJB 클러스터링 설정 분배 (여러 개의 Bean 들)

위에서 지정한 bean 클러스터링이 작동하기 위해서는 다음의 내용을 주의해야 한다.

- 클러스터링에 참여하는 모든 bean들은 <clustering>하위의 모든 정보가 동일해야 한다.

- 동일한 export-name 을 가지는 bean 은 자동으로 클러스터링에 포함된다. 따라서, 클러스터링으로 구성하기 위해서는 원하는 bean 의 export-name 을 모두 동일하게 설정한다.

다음 디스크립터의 한 부분은 세 개의 BMP bean의 설정을 표현한 것이다. 각각 “ACCOUNT”, “ACCOUNT”, “ACCOUNT” 이다. 비록 서로 다른 EJB engine에 deploy 되어있기는 하지만 실제의 EJB들은 동일한 것이다 (즉, 실제의 bean들은 동일한 interface와 구현 클래스들을 가지고 있다).

*<<EJB engine 1 에 deploy 된 accountmod.jar 의 jeus-ejb-dd.xml >>*

```
<jeus-ejb-dd>
    <module-info>
        . . .
    </module-info>
    <beanlist>
        . . .
        <bean-managed>
            . . .
            <export-name>ACCOUNT</export-name>
            . . .
            <clustering>
                <home-clustered>true</home-clustered>
                <ejb-home-failover>true</ejb-home-failover>
                <ejb-home-idempotent-method>
                  <method_name>foo</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
                </ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-method>
                  <method_name>bar</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
```

```
                    </method-params>
                </ejb-remote-idempotent-method>
            </clustering>
            . . .
        </bean-managed>
         . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

*<<EJB engine 2 에 deploy 된 accountmod.jar 의 jeus-ejb-dd.xml >>*

```
<jeus-ejb-dd>
    <module-info>
        . . .
    </module-info>
    <beanlist>
        . . .
        <bean-managed>
            . . .
            <export-name>ACCOUNT</export-name>
            . . .
            <clustering>
                <home-clustered>true</home-clustered>
                <ejb-home-failover>true</ejb-home-failover>
                <ejb-home-idempotent-method>
                  <method_name>foo</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
                </ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-method>
                  <method_name>bar</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
```

```
                </ejb-remote-idempotent-method>
            </clustering>
            . . .
        </bean-managed>
         . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

*<<EJB engine 3 에 deploy 된 accountmod.jar 의 jeus-ejb-dd.xml >>*

```
<jeus-ejb-dd>
    <module-info>
        . . .
    </module-info>
    <beanlist>
        . . .
        <bean-managed>
            . . .
            <export-name>ACCOUNT</export-name>
            . . .
            <clustering>
                <home-clustered>true</home-clustered>
                <ejb-home-failover>true</ejb-home-failover>
                <ejb-home-idempotent-method>
                  <method_name>foo</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
                </ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-method>
                  <method_name>bar</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
```

```
                </ejb-remote-idempotent-method>
            </clustering>
            . . .
        </bean-managed>
         . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 9.3.4 결론

지금까지 JEUS의 bean 클러스터링에 대해 설명하였다.

단일 bean EJB DD에 클러스터링 설정을 어떻게 정의하는지 살펴보았고, 이 같은 정보가 클러스터링에 참가하는 각 bean에 어떻게 설정되는지도 살펴보았다.

## 9.4 결론

이것으로 JEUS의 EJB 클러스터링에 대한 소개를 마친다.

클러스터링은 두 가지 중요한 사항을 만족시킨다. 하나는 향상된 성능이고 다른 하나는 더 좋은 안정성이다. 그러므로, 가능하다면 bean을 클러스터링하여 사용하기를 권장한다.

다음 장에서는 JEUS의 session bean에 대하여 알아본다.

# 10 Session Enterprise JavaBeans

## 10.1 소개

이 장에서는 두 가지의 Session Bean들을 자세히 살펴보기로 한다. Session Bean에는 stateless bean 와 stateful bean이 있다(각각 “SL”, “SF”). 이 장에서는 주로 SF bean들에 대하여 이야기할 것이다. SL 은 앞의 6장에서 설명한 설정 외에는 특별한 것이 없기 때문이다.

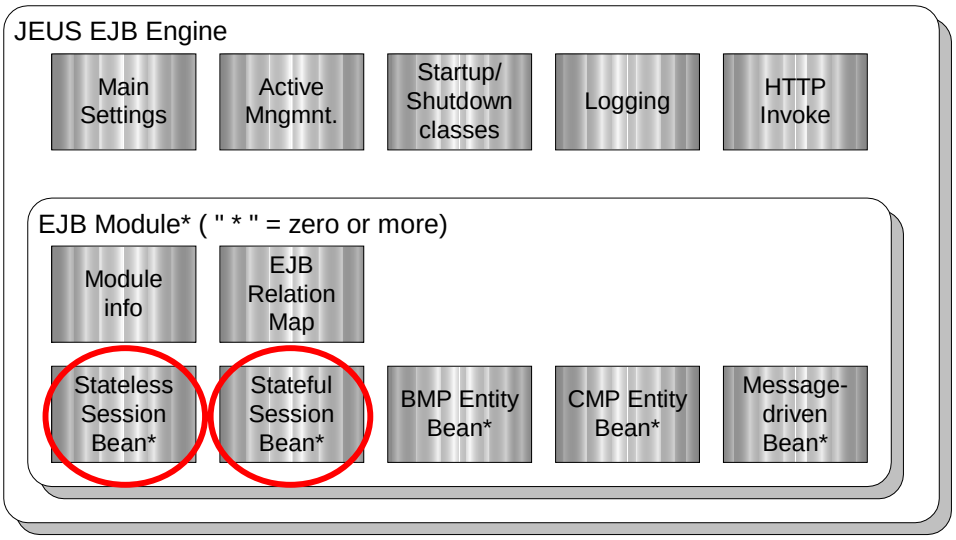


*그림 20. EJB engine 에서의 Session Bean.*

## 10.2 Session EJB 의 개요

### 10.2.1 소개

Stateful Session Bean은 앞서 6장에서 설명했던 공통적인 EJB설정을 포함하고 있고, 추가로 다음과 같은 설정과 컴포넌트들을 가진다.

- Object Management 설정 : Bean Instance Pool 과 Connection Pool.
- Bean Pooling 옵션.
- File DB나 Session Manager를 통한 상태 유지 메커니즘.

다음 세부 절에서는 위의 세 항목들에 대해 자세히 알아본다.

위에서 언급했듯이 Stateless Session Bean은 6장에서 설명했던 일반적인 EJB 설정 외에 추가적인 설정이 필요하지 않다. 다음 절을 제외한 이 장의 나머지 부분에서는 Stateful Session Bean의 추가적인 요구사항들만 설명한다.

### 10.2.2 Stateless Bean Thread Ticket Pool (Stateless Bean 에 해당)

6장에서 설명했듯이, Thread ticket pool은 EJB 클라이언트의 요청을 수락하는 데 사용하는 쓰래드 티켓을 가지고 있다. 이 Thread Ticket의 문법은 Stateless Session Bean에서 다소 다르게 표현된다.

Stateless Session Bean에서 Thread Ticket내의 쓰래드 티켓은 단순히 티켓이 아니라 실제 Session Bean Instance이다. 나중에 설명되겠지만, 이 사실 때문에 Stateless Session Bean에서는 Connection Pool이나 Bean Pool등을 설정할 필요가 없다. Stateless Session Bean의 pooling은 직접 Thread Ticket에 의해 관리된다 [그림 21].

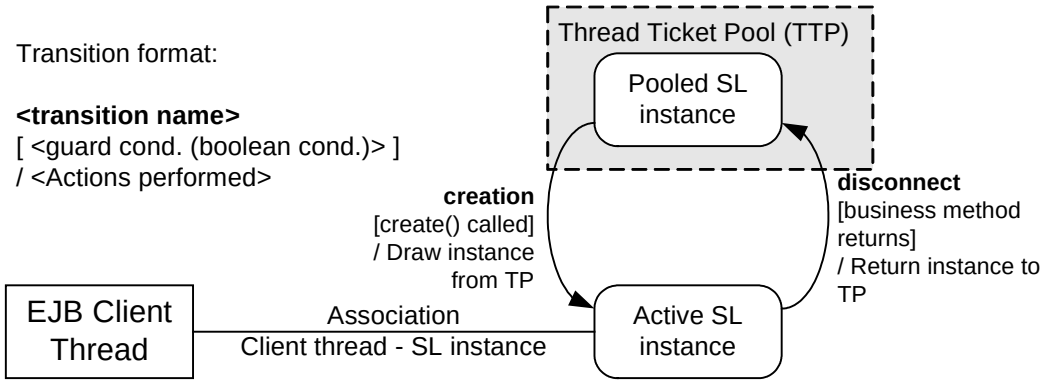


*그림 21. Thread ticket pool 은 SL bean pool 을 포함하고 클라이언트 요청에 이들을 할당한다*

Thread ticket pool 은 SL bean pool을 포함하고 있으며, 클라이언트 요청에 이들을 할당한다 그러므로 클라이언트가 Stateless Session Bean의 method를 호출하면 Thread Ticket에서 하나의 SL instance(그림에서 “SL”)가 꺼내져 오고 요청에 응할 수 있도록 할당된다. SL instance가 처리를 끝냈을 때, 쓰래드는 Thread Ticket로 반환되고 다음 클라이언트 요청을 기다리게 된다.

### 10.2.3 Stateless Bean 과 Webservice endpoint

EJB 2.1 spec에 의해 stateless session bean은 webservice endpoint를 interface로 가질 수 있다. 이때 설정해야 될 것과 실제 구현 예는 JEUS Webservice 안내서를 참고하기 바란다.

이제 Stateful Bean을 사용할 때 Connection Pool과 Bean Pool이 Thread ticket pool과 함께 어떻게 사용되는지 살펴보기로 한다.

### 10.2.4 Stateful Bean Thread Ticket Pool 과 Object Management 설정 (Stateful Bean 에 해당)

[그림 22]은 JEUS SF bean의 완벽한 “state-chart”를 보여준다.

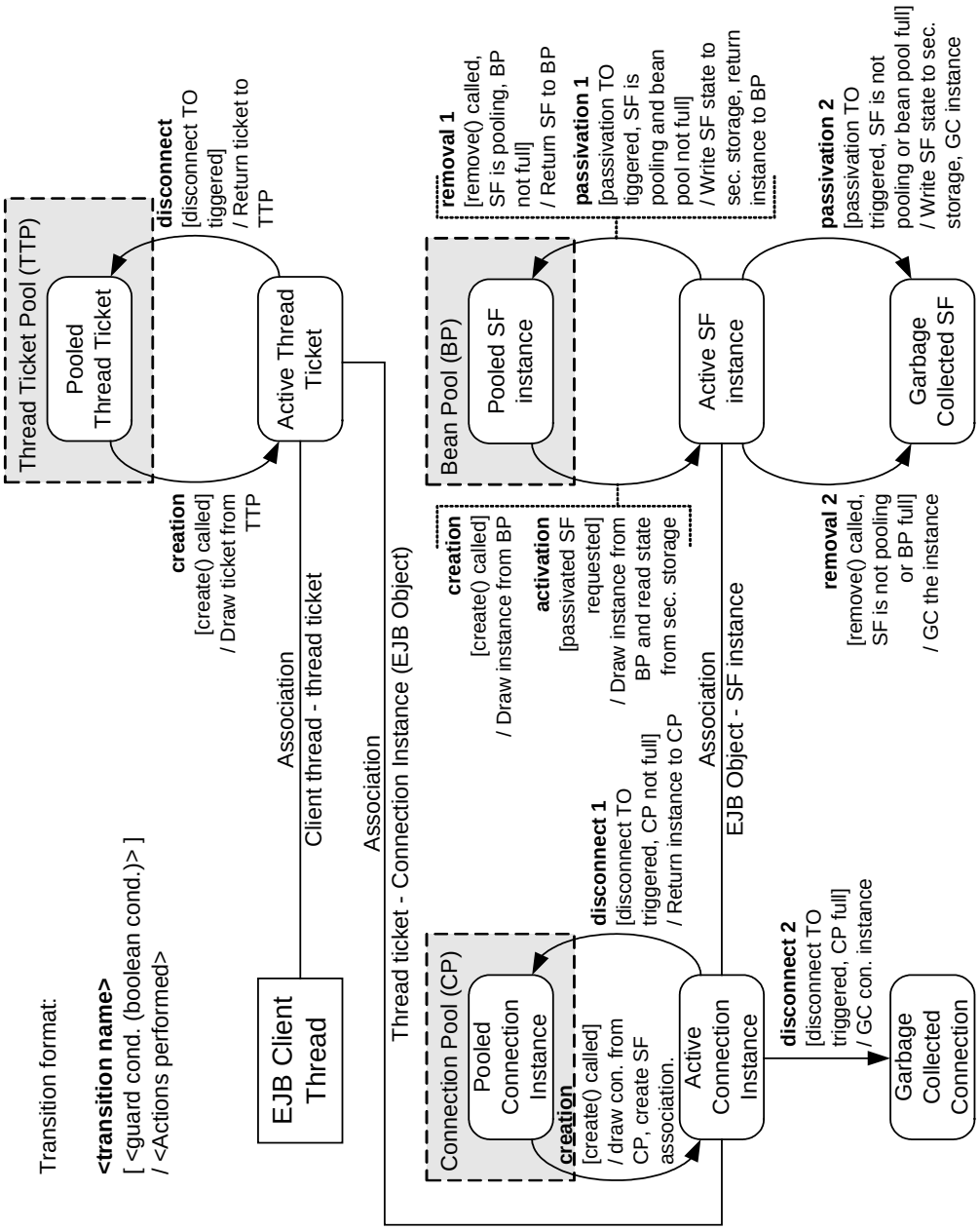


*그림 22. JEUS EJB engine 의 SF bean 에 속하는 Object Management 와 pool*

이름이 암시하듯이 Stateful Bean은 클라이언트의 요청간에 상태 정보를 지속적으로 유지하고 있어야 하기 때문에 간단한 Thread ticket pool만을 가지고는 설정이 불가능하다. Stateful Session Bean은 Connection Pool("CP")과 Bean (Instance) Pool을 설정하고 활용해야 한다([그림 22 ]참조). Entity Bean에도 동일하게 적용된다(11장).

JEUS EJB engine에서 이 pool들이 동작하는 모습들을 살펴보자([그림 22] 참고). 클라이언트의 요청이 Thread ticket pool에 도착하면 Thread ticket pool에서 쓰래드 티켓("TT")을 받는다. TT는 클라이언트의 요청을 Connection Pool에서 받은 EJB object (“EO”)와 연결시킨다. 클라이언트를 대신하여 EO는 Bean Pool(Instance Pool)에서 받은 실제 Stateful Bean Instance("SF")와 연결한다.

클라이언트 요청의 초기화가 끝나면 EJB object는 클라이언트와 선택된 Stateful Session Bean Instance(Stateful Session Bean의 기본 속성)와의 연결을 지속시킨다.

그러나, 만약에 클라이언트가 SF에게 지정된 시간( *passivation timeout* )내에 어떤 요청도 보내지 않으면 시스템 자원의 확보를 위하여 SF instance는 passivate된다. 이 의미는 SF bean의 상태는 2차 저장 장소에 기록되고 Bean Instance는 소멸되거나 Bean Pool로 반환된다는 것이다.

Passivation(그림에서 passivation 1)이 된 후에 bean이 Bean Pool로 반환되는 조건은 bean이 *pooling bean* 으로 설정되고, Bean Pool이 이미 차 있지 않은 조건(max pool size 값 이상이 되지 않는 조건)이어야 한다. 이 두 조건 중의 하나라도 만족시키지 못하면 SF bean instance는 passivation(그림에서 passivation 2) 후에 소멸된다.

클라이언트가 passivate된 SF bean의 서비스를 요청하면 bean은 다시 *activate* 된다 (그림에서 Activation Transition). 이는 새로운 SF instance가 Bean Pool에서부터 발급되고 2차 저장소에서 받은 데이터로 채워진다.

클라이언트가 *disconnect timeout* 에 지정된 시간만큼 동안 SF bean을 접근하지 않으면 클라이언트와의 연결을 관리하는 SF bean의 EJB object가 connection pool로 반환(disconnect 1)되거나 또는 Connection Pool이 차 있으면 소멸(disconnect 2) 된다. 두 경우 모두 EJB client와 SF bean과의 연결을 영구적으로 끊어버린다. 연결을 잃어 버린 후 bean에게 새로운 요청이 들어오면 클라이언트에게 exception이 떨어진다.

다음에 나오는 장들에서 설명하겠지만, Object Management 관련 설정은 모든 종류의 Entity Bean들에게도 적용된다.

### 10.2.5 Pooling Session Bean (Stateful Bean 에 해당)

Stateful Session Bean을 설정할 때에는 이 bean에만 해당되는 유일한 설정이 적용된다. 바로 *pooling* 설정이다.

J2EE EJB스펙에 따르면, 실제 Stateful Session Bean의 instance는 다른 클라이언트 세션에 재사용될 수 없다. 그러나 이 요구 사항은 별로 유용한 것이 아니라서, JEUS는 성능개선을 위하여 이 부분만 스펙을 "위반"하는 방법을 제공한다. 이렇게 함으로써, Pooling 설정을 활성화하여 모든 Stateful Session Bean들이 보관되었다가 클라이언트 세션에 재사용하여 자원 낭비를 줄일 수 있는 결과를 얻는다.

전 절에서 설명하였듯이, 이 기능이 활성화되면, EJB engine은 실제로 모든 passivate 된 SF Bean instance들을 재사용하기 위하여 Bean Pool에 반환한다.

그러나 기본적으로는 스펙을 준수하기 위해서, Stateful Session Bean은 이 방식으로 저장되지 않고, passivation 된 후에 소멸된다. 이에 대한 설정은 나중에 설명한다.

### 10.2.6 상태 유지 메커니즘 (Stateful Bean 에 해당)

이름에서 암시하듯이, Stateful Session Bean은 상태를 지니고 다닌다. 이 상태는 클라이언트 세션 동안 반드시 유지되어야 한다 (즉, 같은 클라이언트로부터 오는 모든  다른 요청에도 상태는 지속되어야 한다). 이것이 Stateful Session Bean의 기본 특성이다.

일반적으로, 런타임시의 상태는 instance 변수로 저장된다. 그러나, 위에서 설명한 것과 마찬가지로 Stateful Session Bean이 passivate 상태로 진입하면 시스템 자원을 보존하기 위하여 운영 환경에서 이 instance변수들을 제거해야 한다. 그러나 bean이 다시 재활성화되면 상태를 복구해야 하기 위해서 어딘가에서 데이터를 가져와야 한다. 이게 바로 2차 저장소가 필요한 이유이다.

이 2차 저장소는 다음과 같은 두 방식이 있다.

- **file-database** (file DB). File DB가 사용될 때에는 모든 Stateful Session Bean 의 상태가 passivate 되었을 때 로컬 파일 시스템에 파일로 저장이 된다. 로컬 머신에서 운영되는 EJB engine에만 이 파일의 접근 권한이 있으므로 클러스터링으로 구성된 Stateful Session Bean에서는 File DB를 사용할 수 없다. 하지만, File DB는 다음에 설명될 Session Manager보다 더 효율적이고 빠르다.

- **remote session manager.** JEUS Manager에서는 "Session Manager"를 설정할 수 있다. Session Manager는 클러스터링으로 구성된 Servlet과 Stateful Session Bean의 세션 데이터를 저장하는 서버처럼 동작한다. 와 같이 작동한다. Session Server는 JEUSMain.xml 파일에서 설정된다. Stateful Session Bean을 위해 Session Manager를 사용하는 것이 File

DB를 사용하는 것보다 비효율적이기는 하지만 Stateful Bean들이 클러스터링으로 구성될 때에는 반드시 사용되어야 한다. Session Manager를 사용할 때, bean이 Passivation timeout으로 인해 passivate 되거나 Transaction Commit이 성공적으로 수행된다면 Session Manager로 데이터가 전달된다.

**참고**: JEUSMain.xml에 Session Manager는 "removal-to" (removal timeout)이라는 설정이 있다. 이 값이 "passivation-to"의 값보다 더 커야 한다.

### 10.2.7 결론

위에서는 JEUS의 Session Bean에 대하여 다음과 같은 주제들을 설명하였다.

SL bean을 위한 설정은, "일반적인 EJB 장"(6장)에서 다루었던 항목들을 제외하곤 없다. Thread ticket pool은 thread ticket으로 사용되는 것 보다는. SL bean의 실제 Instance를 사용하기 위해서 사용된다.

SF bean 에 대해서는, Object Management 관련 설정(connection pool, bean pool)을 알아봤다. 그리고, SF pooling 옵션과 두 개의 상태 유지 메커니즘(File DB와 Remote Session Manager)에 대하여 설명하였다.

다음은 SF bean의 항목을 구체적으로 설정하는 방법에 대하여 설명한다.

## 10.3 Session EJB 설정

### 10.3.1 소개

여기서는 실제로 어떻게 SF bean을 설정하는지 설명한다. 살펴볼 항목들은 다음과 같다.

- Object Management 설정: Bean Pool과 Connection Pool.
- Bean Pooling 옵션.
- File DB나 Session Manager를 이용한 상태 유지 메커니즘

이 모든 항목들은 JEUS EJB 모듈 DD의 `<jeus-bean>` 태그 아래에 설정된다.

SL bean설정은 6장에도 설명되어 있으니 참고하기 바란다.

### 10.3.2 Object Management 관련 설정(Stateful Bean 에 해당)

SF bean 의 Object Management 설정을 위해서는 다음과 같은 설정들이 제공되어야 한다.

- **<bean-pool>** 설정은 bean instance pool의 작동방식을 결정한다. 하위 태그들은 다음과 같다.
  - **<pool-min>**: Pool에 유지하려는 최소 bean instance의 개수.
  - **<pool-max>:** Pool에 유지하려는 최대 bean instance의 개수
  - **<resizing-period**>:Pool의 크기를 재조정하는 시간. “Resizing” 은 사용되지 않는 bean instance들이 pool의 최조값까지 제거됨을 의미한다.
- **<connect-pool>** 설정은 몇 개의 EJB Remote의 커넥션을 몇 개까지 유지 할지를 설정한다. 하위 태그들은 다음과 같다.
  - **<pool-min>:** Pool 내에 유지할 EJB connection object instance의 최조값이며 최소값이다.
  - **<pool-max>:** Pool에서 유지할 최대 EJB connection object instance 의 개수이다.
  - **<resizing-period**>:Pool의 크기를 재조정하는 시간. “Resizing” 은 사용되지 않는 bean instance들이 pool의 최조값까지 제거됨을 의미한다.
- **<capacity>**설정은 EJB engine이 효율적으로 동작하기 위한 힌트를 제공한다. 이 값은 생성될 bean instance들의 최대값을 추측하는데 사용된다. 이 값은 Hashtable에 저장된 SF meta data의 효과적인 구성을 위해 사용된다.
- **<passivation-timeout>** 시간 동안 클라이언트의 요청을 받지 않는 bean을 passivate하는 데 사용된다. 그러므로, 여기에 설정된 시간을 초과하는 동안 클라이언트의 요청이 없으면 그 bean은 passivation의 대상이 되는 것이다. Passivation이 실행되면, 메모리에서 그 bean instance가 제거되고 그 상태는 파일이나 DB에 저장된다.
- **<disconnect-timeout>** 이 태그는 여기에 설정된 시간 동안 클라이언트의 요청을 받지 못하면 클라이언트와 bean사이의 연결을 취소하는 데

사용된다. 그렇게 되면 EJB object connection instance는 영구적으로 runtime 메모리에서 제거된다.

**중요**: passivation-timeout 과 disconnect-timeout에 사용되는 시간은 bean instance에 액세스했던 마지막 시점부터 측정된다. 그러므로 disconnect-timeout 을 passivation- timeout 보다 길게 설정해 줘야 한다.

**참고**: 위의 설정들은 EJB마다 각각 적용된다. 이 의미는 EJB engine안에서 모든 EJB들은 고유의 pool과 timeout 설정 값을 가진다는 것이다.

위의 설정들은 Entity Bean (BMP, CMP 1.1와 CMP 2.0)에도 같이 적용된다. 이 문서의 후반부에서Entity Bean의 설정을 설명할 때 이 절을 참조하기 바란다.

다음은 위의 설정을 포함한 XML 샘플이다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <object-management>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </bean-pool>
                <connect-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </connect-pool>
                <capacity>2000</capacity>
                <passivation-timeout>10000</passivation-timeout>
                <disconnect-timeout>1800000</disconnect-timeout>
            </object-management>
        <jeus-bean>
        . . .
    <beanlist>
    . . .
<jeus-ejb-dd>
```

### 10.3.3 Bean Pooling 설정(Stateful Bean 에 적용)

Stateful Session Bean을 “pooling” bean으로 전환 하기 위해서는 간단히 jeus-ejb-dd.xml의 <pooling-bean> 값을 “true”로 설정해준다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>teller</ejb-name>
            <export-name>TELLEREJB</export-name>
            <local-export-name>
                LOCALTELLEREJB
            </local-export-name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            <single-vm-only>true</single-vm-only>
            <local-invoke-optimize>true</local-invoke-optimize>
            <pooling-bean>true</pooling-bean>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 10.3.4 세션 데이터 유지 메커니즘 설정 (Stateful Bean 에 해당)

전 절에서 본 것처럼, 두 가지의 세션 데이터 저장 메커니즘인 File DB와 Remote Session Manager 중 한 가지를 선택할 수 있다. 클러스터링을 사용하지 않는다면 다음과 같은 설정이 적용될 수 있고 `<file-db-info>` 태그 아래에 < local-file-db >태그에 아래의 내용을 설정한다.

- **<file-db-path>** 로컬 파일 시스템의 절대 경로이다. 이 경로는 메인과 백업 DB파일들이 저장되는 경로를 지정한다.

- **<file-db-name>** File DB의 이름이다. 실제 파일 이름은 이 태그에서 주어진 이름과 숫자, 확장자 “.fdb”가 조합되어 사용 된다 (확장자명을 따로 기입하면 않도록 한다

- **<min-hole>** 과 **<packaging-rate>** 최적화를 위한 설정이다.

Stateful Bean을 클러스터링한다면 JEUSMain.xml에서 구성한 Session Manager를 사용해야 한다. 그리고, <file-db-info> 태그 아래에 <remote-file-db>태그에 아래의 내용을 설정한다.

- **<remote-primary-file-DB>** JEUSMain.xml에 선언되어 있는 주 Session Manager의 JNDI export name.
- **<remote-backup-file-DB>** JEUSMain.xml 에 설정되어 있는 백업 Session Manager의 JNDI export name.
- **<conn-pool-size>** Session Manager와 EJB engine 간의 Connection Pool 크기.

다음은 XML 예이다. 실제 설정에서는 <is-local-file-db> 태그의 참 또는 거짓만 선택적으로 설정하고 그에 따른 설정을 해 주면 된다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <file-db-info>
                <local-file-db>
                    <file-db-path>c:\temp</file-db-path>
                    <file-db-name>teller</file-db-name>
                    <min-hole>5000</min-hole>
                    <packing-rate>0.4</packing-rate>
                </local-file-db>
                <remote-file-db>
                    <remote-primary-file-db>
                        MYSESSIONSERVER
                    </remote-primary-file-db>
                    <remote-backup-file-db>
                        MYSESSIONBACKUP
```

```
                    </remote-backup-file-db>
                    <conn-pool-size>50</conn-pool-size>
                </remote-file-db>
            </file-db-info>
        </jeus-bean>
        . . .
    <beanlist>
    . . .
<jeus-ejb-dd>
```

### 10.3.5 결론

SF bean에 관련하여 세 가지 태그를 어떻게 설정하는지 살펴보았다 (Object의 Pooling 설정, pooling, 세션 데이터 저장).

SL bean과 기본 SF 설정에 대한 정보는 6장을 참고한다.

다음 절에서는 SF bean(그리고 Entity bean) 을 튜닝할 때 필요한 내용을 설명한다

## 10.4 Session (그리고 Entity) EJB 튜닝

### 10.4.1 소개

6장에서 본 성능 튜닝 정보 외에 다음과 같은 사항들도 Stateful Session Bean을 deploy 할 때 반드시 확인되어야 한다(Stateless Session Bean은 6장에서 주어진 정보 외에 다른 것은 필요하지 않다).

- pooling 옵션 사용하기.
- File DB 와 Session Manager중 선택.
- File DB 설정 조절.
- Object Management 의 capacity 태그 설정 바로 하기.
- Object Management 의 timeout 값 설정.

처음 세 항목은 SF bean에만 적용되는 것이고 나머지 두 개는 SF와 Entity Bean에도 적용된다.

### 10.4.2 Pooling 옵션 사용

사용된 Stateful Session Bean은 재사용 되어서는 안 된다는 스펙의 요구사항을 반드시 지켜야 하는 이유가 없다면 pooling bean옵션은 활성화시켜 놓는다. <pooling-bean> 태그의 값을 “true”로 하여 pooling을 활성화한다.

### 10.4.3 File DB 와 Session Manager 중 선택

Stateful Session Bean을 클러스터링 설정할 필요가 없을 경우에는 세션 데이터 저장소로 File DB를 사용하도록 한다. Bean이 클러스터링으로 되어 있는 경우에만 Remote File DB(Session Manager)를 사용하도록 한다.

### 10.4.4 File DB 설정 조절

최적의 성능을 내기 위해서는 min hole과 packaging rate의 두 File DB설정을 튜닝해야 한다.

일반적으로, 두 값이 높게 설정되어있으면, File DB의 최적화가 자주 일어나지는 않는다. 반면, 시스템 자원 낭비가 발생할 수도 있고 File DB 접근 시간이 길어질 수 있다.

### 10.4.5 Object Management 의 capacity 태그의 올바른 설정

Object Management 의 capacity 태그를 정확히 설정해야 한다. 많은 instance들의 생성과 자주 사용하는 bean에는 높은 값을 설정한다. 자주 사용하지 않는 bean은 낮게 설정한다. 이 숫자는 동시에 생길 SF instance들의 대략적인 최대 값을 반영한 값이다.

메모리 관리가 중요한 사항일 경우에 신중하게 조절해야 한다.

### 10.4.6 Object Management 의 timeout 값 설정

Object Management 을 설정할 때, passivation timeout과 disconnection timeout을 천분의 일초 단위로 설정할 수 있다. 설정할 때에는 다음과 같은 주의를 요한다.

- timeout값을 길게 주면 오랜 시간(대략 십여 분 이상 또는 timeout이 중지의 경우) 동안 메모리 안에 많은 instance가 활성화된 상태로 머물러 있다. 그러므로 시스템 자원 낭비를 초래한다.
- 너무 짧은 timeout값은 (수 초) passivation, activation, removal등의 작업이 너무 자주 일어나므로 성능을 저하시킬 수 있다.

disconnection timeout 기간이 항상 passivation timeout보다 길어야 한다는 것에 주의 한다.

### 10.4.7 결론

이것으로 SF/Entity Bean 튜닝에 대한 것을 마무리 한다.

이 절에서는 다음 사항들에 대해 알아보았다.

- pooling 옵션.
- File DB 와 Session Manager간의 선택.
- File DB 설정 조절.
- Object Management 의 capacity 태그의 올바른 설정.
- Object Management 의 timeout 값 설정.

## 10.5 결론

이 장에서는 SF Session Bean의 기본적인 컴포넌트와 설정법 등을 살펴보았다.

다음 장에서는 세 종류의 Entity EJB, 즉 BMP, CMP1.1, CMP2.0에 대해 알아보겠다.

# 11 Entity Enterprise JavaBeans

## 11.1 소개

JEUS EJB engine은 세 가지의 Entity EJB를 지원한다. BMP (bean-managed persistence), CMP 1.1 (container-managed persistence) 그리고 CMP 2.0이 그것이다.

이 장은 JEUS에서 이러한 EJB들을 설정하고 튜닝하는데 필요한 모든 정보를 다룬다.

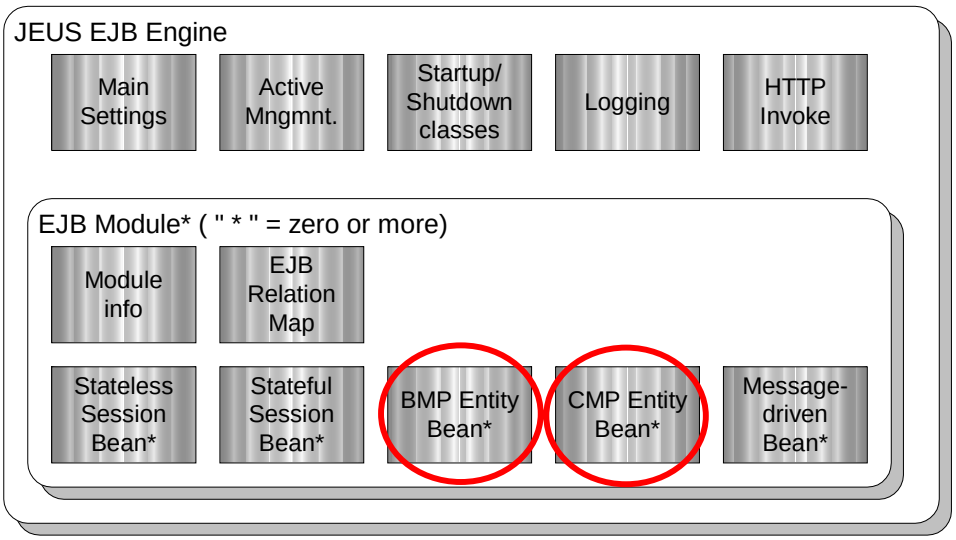


*그림 23. EJB engine 의 Entity bean*

.

## 11.2 Entity EJB 의 개요

### 11.2.1 소개

개요 부분에서는 다음과 같은 주제에 대하여 설명한다.

- 세 가지의 JEUS Entity EJB
- JEUS entity EJB의 설정할 수 있는 특성들(개요)

- Entity EJB Object management
- 일반적인 persistence 최적화
- CMP- persistence 최적화
- CMP 스키마 설정
- CMP 2.0 Relation
- default primary key 클래스
- JEUS 의EJB QL 추가
- instant EJB QL API
- 자동 Primary Key 생성

**참고**: JEUS에서는 성능 튜닝이 EJB entity bean 설정의 가장 중요한 부분이다. 그러므로, 이 장에서는 engine 모드와 서브 모드와 같이 성능 관련 부분에 대하여 많은 설명을 한다. 이 장의 후반부의 Entity bean 튜닝 부분(11.7절)은 이 절에서 설명한 내용을 좀 더 간략하게 다시 정리한 것이다

### 11.2.2 JEUS 에서 지원하는 세 가지의 Entity EJB

J2EE EJB 스펙에 준하여, JEUS EJB engine은 다음과 같은 세가지 entity bean 들을 지원한다.

- Bean Managed Persistence Entity Bean (지금부터 약자로 “BMP”를 사용한다).
- EJB 1.1 스펙에 따른 Container Managed Persistence Entity Bean (약자로 “CMP 1.1”를 사용한다).
- EJB 2.1 스펙에 따른 Container Managed Persistence Entity Bean (약자로 “CMP 2.0”를 사용한다).

세 개의 bean 종류에 따라 세 개의 설정이 존재한다.

- 세 종류에 모두 적용되는 공통된 설정들 (“모든 entity bean에 적용”가 표시될 것이다).

- **CMP 1.1 and CMP 2.0** 에만 적용되고 BMP에는 적용되지 않는 설정들 ("CMP 1.1/2.0에 해당" 이 표시될 것이다).
- **CMP 2.0** 에만 적용되는 설정들 ("CMP 2.0에 해당" 이 표시될 것이다).

### 11.2.3 각 Entity Bean 종류 별 설정의 개요

[표1]은 하위 entity bean 종류의 설정 특성들을 보여주고 있다.

*표 8. JEUS 내의 BMP, CMP 1.1 그리고 CMP 2.0 bean 들의 구성 요소.*

| 설정 가능한 Entity 옵션 | BMP | CMP 1.1 | CMP 2.0 |
|---|---|---|---|
| 1. EJB name | ✓ | ✓ | ✓ |
| 2. Export name | ✓ | ✓ | ✓ |
| 3. Local export name | ✓ | ✓ | ✓ |
| 4. Export port | ✓ | ✓ | ✓ |
| 5. Export IIOP switch | ✓ | ✓ | ✓ |
| 6. Single VM switch | ✓ | ✓ | ✓ |
| 7. Local invocation opt. | ✓ | ✓ | ✓ |
| 8. Fast deploy | ✓ | ✓ | ✓ |
| 9. use-access-control | ✓ | ✓ | ✓ |
| 10. Run-as identity | ✓ | ✓ | ✓ |
| 11. Security CSI Interop. | ✓ | ✓ | ✓ |
| 12. Env. refs | ✓ | ✓ | ✓ |
| 13. EJB refs | ✓ | ✓ | ✓ |
| 14. Resource Refs | ✓ | ✓ | ✓ |
| 15. Resource env. Refs | ✓ | ✓ | ✓ |
| 16. Thread ticket pool settings | ✓ | ✓ | ✓ |
| 17. Clustering settings | ✓ | ✓ | ✓ |
| 18. HTTP invoke | ✓ | ✓ | ✓ |
| 19. Object management | ✓ | ✓ | ✓ |
| 20. Persistence Optimize | ✓ | ✓ | ✓ |
| 21. CM persistence opt. | | ✓ | ✓ |
| 22. Schema info | | ✓ | ✓ |
| 23. Relationship map | | | ✓ |

위의 표에서와 같이 항목 1부터 16까지는 모든 EJB에 공통으로 적용된다. 이 항목들은 6장에서 모두 설명하였다

항목 19부터 23까지는 (회색으로 구분된 부분) entity bean에만 해당하는 항목들이다. 19항목은 예외인데 이는 stateful session bean에도 해당된다. 이 내용에 대해서는 10장에서 이미 다루었다. 20에서 23 항목까지는 이 장에서 자세히 다룬다

위에서 회색에서 구분된 항목 외에도 다음과 같은 JEUS entity EJB에 연관된 주제에 대해서도 설명한다.

- default primary key 클래스.
- EJB QL 언어에 추가된 JEUS 특정항목.
- JEUS instant EJB QL API.

### 11.2.4 세 Entity Bean 들의 공통된 특성들

종류에 상관없이 JEUS의 entity bean들은 모두 6장에서 설명한 기능과 설정 컴포넌트들을 공유한다.

그러므로, 이 장을 읽고 JEUS EJB engine에서 entity bean을 설정하고 사용하기 전에 6장의 내용에 대해 깊은 이해를 하고 있어야 한다.

### 11.2.5 Entity bean Object 관리 및 Entity Cache (모든 종류에 해당)

[그림 24]은 EJB engine에서 entity bean 객체가 어떻게 관리되는지 보여준다.

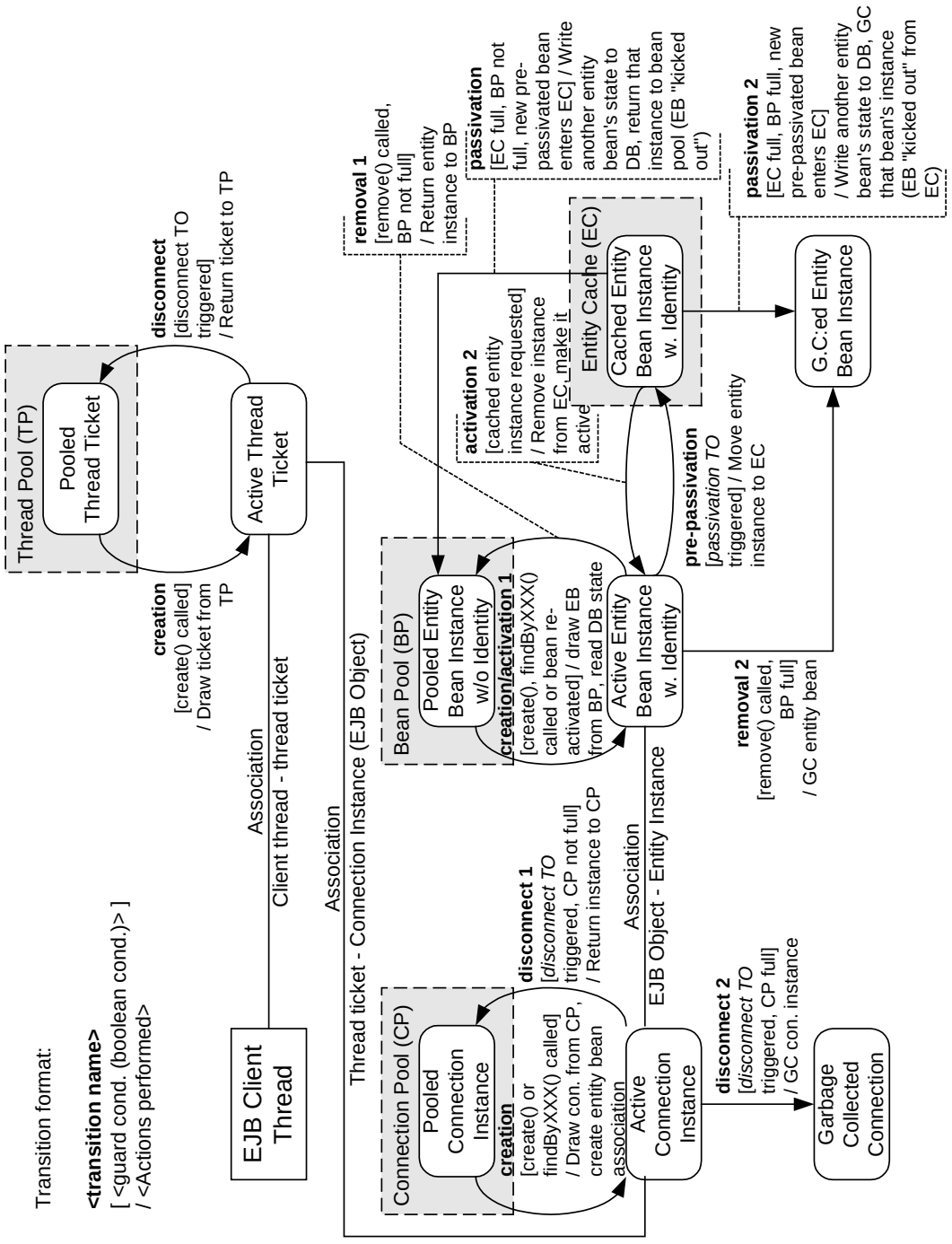


*그림 24. JEUS EJB engine 에서 Entity bean 의 object 와 instance 관리.*

"object management" 는 EJB engine의 내부적인 bean instance pool과 connection pool을 의미한다. 이 pool들은 각 bean마다 설정되고 성능향상과 시스템 자원의 효율적인 관리를 위해 사용된다.

Object management 의 개념은 기본적으로 모든 entity bean들과 Stateful session bean에 대하여 공통적으로 반영될 수 있다. 그러므로, object management와 entity bean에 대한 정보가 있는 10장을 참고하라

그러나, SF(Stateful session bean)와 entity bean의 object management 에는 한 가지의 중요한 차이 점이 있다. Entity bean은 표준 pool외에 entity-cache를 사용한다. Entity cache는 [그림 24]의 오른쪽에 그려져 있다. 이는 다음과 같이 작동한다.

Entity EJB가 passivate되려 할 때(passivation timeout이 지났을 때) 그 bean은 실제로는 바로 passivate되지 않는다. (SF bean의 경우에는 passivate된다) 대신에 이는 entity cache로 전달된다. 이 cache에서는 모든 passivate된 entity bean들이 활성화 상태로 저장되어 있다 (즉, 식별될 수 있고, 유효한 상태를 유지한다). 다른 pool들과 마찬가지로 이 entity cache도 runtime 메모리에 유지되고, 따라서 활성화 상태와 passivate 사이의 존재하지 않는 상태에서 runtime cache로 작동된다.

그렇다면 언제 entity bean이 실제적으로 passivate되는가? Entity cache pool이 꽉 찰 때 passivate가 된다. 새로운 entity bean이 cache로 들어오려 할 때 그 cache가 이미 passivate된 bean들로 가득 차 있다고 한다면 cache내에 있는 하나의 bean이 선택되어 강제로 밀려 나가게 된다. 이렇게 강제로 밀려 나간 entity bean은 passivate되고 그 상태는 database에 쓰여지며 그 instance는 bean pool로 반환되거나 버려진다(bean pool이 가득 차 있는 상태라면). cache에 막 들어오려는 entity bean은 cache에 진입하여 결과적으로 강제로 밀려난 bean instance의 자리를 차지하게 된다.

이러한 cache를 사용함으로써 얻는 장점은 시간이 많이 소모되는 passivation 관리를 효과적으로 만들 수 있다는 것이다. Cache를 사용함으로써 passivate 된 entity bean을 다시 활성화 시킬 때 EJB database에 접근할 필요가 없어지고 cache에 있는 instance(만약에 존재하면)를 사용함으로써 해결할 수 있게 된다.

Object pool 의 표준 passivation timeout 값이 시간적인 제약을 관장하는 passivation 값이라고 한다면, entity cache는 메모리의 제약을 관장하는 passivation 설정 값이라고 할 수 있다. 이것이 entity cache를 사용하는 주요 동기이다.

**주의:** entity cache 설정은 실제로 “object management”의 설정이 아니라 일반적인 entity bean persistence 최적화 설정이다. 이것은 이 장의 후반부의 설정 부분에서 더 자세히 다룬다.

### 11.2.6 Engine Mode Selection 을 통한 ejbLoad()Persistence 최적화 (모든 entity bean 에 적용)

Entity bean에 대하여 어느 시기에 ejbLoad() method를 호출할 것인지 결정하는 것은 EJB engine(container)에 달려있다. 이 method는 database의 상태정보와 bean instance의 runtime 상태정보를 동기화 시키기 위해 BMP와 CMP에서 사용하는 것이다. 본질적으로, ejbLoad() method는 database에서 한 레코드를 읽고 그 결과를 bean의 내부 특성 값들에 설정하기 위해 사용된다.

ejbLoad() method는 entity bean의 생명주기 동안 적어도 한 번은 호출된다. 어떤 경우에는 bean instance의 호출 때마다 바로 전에 ejbLoad() method를 호출할 필요가 생기기도 한다. 왜 그런가? Database에서 bean의 상태정보를 읽어야 하는 시나리오에는 기본적으로 크게 두 가지가 있다.

- Entity bean 이 클러스터링되어 있을 때. Bean이 여러 개의 EJB engine에 걸쳐 클러스터 되어 있을 때에 그 중 하나의 bean이 database의 한 레코드를 변경시킬 수 있다. 이 의미는 특정 entity bean instance에 있는 상태 정보가 가장 최신의 것이라는 것을 보장할 수 없다는 의미이다 (클러스터링된 entity bean들이 서로 통신할 수 없기 때문이다).

- 어떤 외부 요소(작업자 또는 시스템)가 EJB와 연동되어 있는 database를 변경할 경우가 있다. 이 시나리오에서 EJB engine의 특정 EJB instance는 database의 내용을 제대로 반영하지 못한 상태가 된다.

[그림 25]에 두 개의 시나리오가 표현되어 있다.

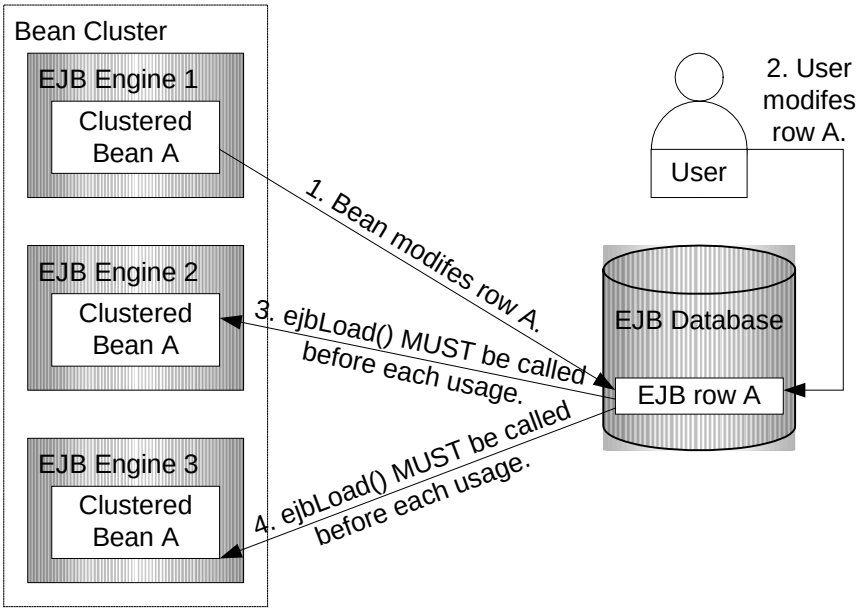


*그림 25. Entity bean 이 클러스터링되어 있을 때나 외부에서 bean 의 DB 행을 수정하였을 때, ejbLoad 는 주기적으로 호출되어야 한다.*

위의 두 시나리오의 상황을 피할 수만 있다면(클러스터링을 사용하지 않고, 어떤 외부요소도 EJB의 DB데이터를 변경하지 않는다고 보장한다면) entity bean의 ejbLoad() 동작을 최적화 할 수 있게 된다. “EXCLUSIVE_ACCESS” engine모드를 bean에 적용하면 최적화가 가능하다. 이 모드를 사용하면, ejbLoad()는 bean이 인스턴스화될 때 단 한번만 호출된다. 이렇게 함으로써 DB의 접근량을 약 50%가량 줄일 수 있다. [그림 26] 는 EXCLUSIVE_ACCESS 모드가 사용될 때의 시나리오를 보여주고 있다.

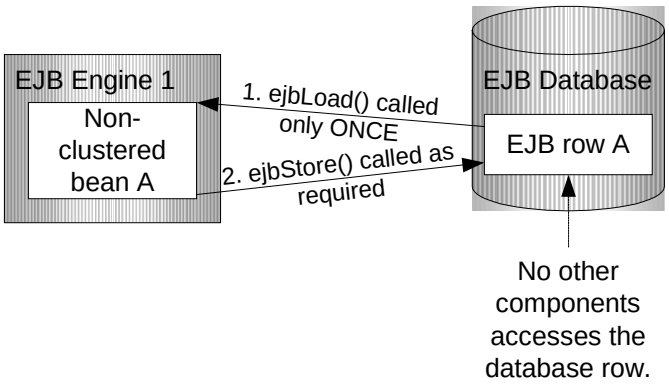


*그림 26. EXCLUSIVE_ACCESS 는 단지 하나의 bean 이 database 행을 사용하고 대부분의 ejbLoad() 는 최적화된다*

Bean을 클러스터링 시켜야 하거나 외부 요소가 EJB데이터를 변경시켜야 한다면 “SINGLE_OBJECT”또는 “MULTIPLE_OBJECT” engine 모드를 사용해야 한다. 두 모드 중 하나가 사용되면, EJB engine은 EJB 클라이언트의 요청이 도착 할 때마다 ejbLoad() method를 호출한다 [그림 26]. 이렇게 하면 각 entity bean의 성능이 저하되지만 모든 상태정보가 database에 의해 관리되기 때문에 다른 engine과 bean들에게 부하를 분산시킬 수 있다 [그림 26].

그렇다면 SINGLE_OBJECT와 MULTIPLE_OBJECT 모드 사이의 다른 점은 무엇인가? SINGLE_OBJECT 에서는 각 EJB engine의 단 한 개의 bean instance만이 모든 클라이언트의 요청을 처리한다. 즉, 첫 요청이 처리되고 있는 동안 도착한 요청들은 대기하고 있어야 한다 [그림 27] .MULTIPLE_OBJECT 모드에서는 이러한 제약이 적용되지 않고 EJB engine이 새로운 entity bean instance를 생성하여 새로운 EJB 클라이언트 요청을 처리한다.

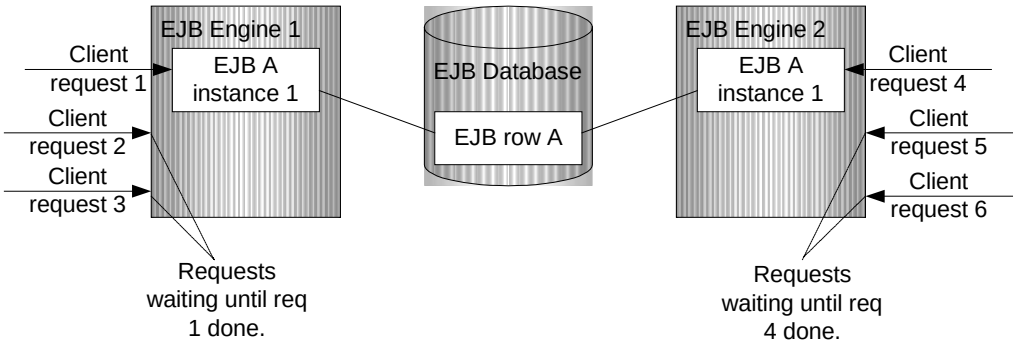


*그림 27. SINGLE_OBJECT 엔진 모드의 새로운 요청은 대기해야 한다.*

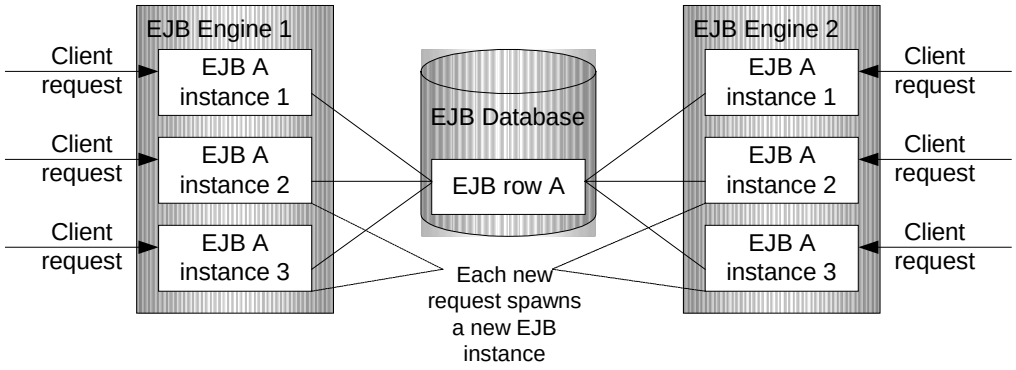


*그림 28. MULTIPLE_OBJECT 엔진 모드에서는 새로운 EJB instance 가 할당된다*

.

[표2]는 각 engine 모드를 선택할 때의 모든 장점과 단점들을 나열해 놓고 있다.

*표 9. 세가지 엔진모드의 단점과 장점.*

| | **EXCLUSIVE** | **SINGLE** | **MULITPLE** |
|---|---|---|---|
| 클러스터링을 허용하는가? | No | Yes | Yes |
| 외부 entity가 DB에 접근 가능한가? | No | Yes | Yes |
| Database connection이 효율적으로 사용 되는 가?(ejbLoad() 호출빈도). | Yes | No | No |
| 장시간의 transaction의 경우 적당한 옵션 지원하는가?. | No | No | Yes |
| 하나의 엔진에 적합한 단일 처리의 효과적인 모드는? | Not applicable | Yes | No |
| 하나의 엔진에서 EJB를 동시에 처리하기 위해 요구되는 효과적인 다중처리 모드는? | Not applicable | No | Yes |

이 모드들 중에 적합한 모드를 선택하기 위한 기준은 다음과 같다.

- 많은 양의 요청이 entity bean에 요청되고, 많은 수의 EJB engine들이 설정되어 있다면 SINGLE_OBJECT모드를 선택하고 engine들에 bean들을 클러스터링 구성한다.
- 많은 양의 요청이 entity bean에 요청되지만, 한정된 EJB engine이 설정되어 있으면 MULTIPLE_OBJECT 모드를 선택하고 engine들에 bean들을 클러스터링 구성한다.

- MULTIPLE_OBJECT과 SINGLE_OBJECT을 선택할 때 고려해야 할 중요한 사항 중의 하나는 한 트랜잭션을 수행하기 위해 필요한 총 시간이다. 이 시간이 많이 걸리는 트랜잭션이라면 당연히 MULTIPLE_OBJECT이 더 좋은 선택일 것이다.

- 많지 않은 동시 요청이 entity bean에 들어온다고 한다면 EXCLUSIVE_ACCESS모드를 선택하고 하나의 EJB engine에 bean을 deploy한다.

EJB클러스터링에 대한 자세한 정보는 9장의 앞 부분을 살펴보라.

### 11.2.7 Non-modifying Methods 에 의한 ejbStore()Persistence 최적화 (BMP & CMP 1.1 에 해당)

전 절에서는 ejbLoad() 호출이 어떻게 최적화될 수 있는지에 대해 살펴보았다. 이 절에서는 어떻게 ejbStore() 호출이 최적화 될 수 있는지에 대해 살펴보겠다. 이것은 BMP와 CMP1.1 bean들에만 적용된다. CMP2.0은 다른 방법으로 이 문제에 대해 접근한다.

ejbStore()는 database에 bean의 상태정보가 유지되어야 한다고 EJB engine에 의해 판단될 때마다 호출된다. 일반적으로, 이 database의 접근은 각 트랜잭션의 commit또는 bean instance가 passivate되기 전에 시도된다.

그러나, 만약에 트랜잭션의 중간에 bean의 상태가 바뀌지 않으면 ejbStore()가 호출될 이유가 없다. 현재, JEUS EJB engine은 bean의 상태가 변경되었는지 자동적으로 예측할 수 없다. 그러므로, 개발자나 deployer가 언제 ejbStore()를 호출해야 하는지 EJB engine이 판단할 수 있도록 힌트를 제공해야 한다.

이 힌트는 non-modifying method들의 리스트로 제공된다. non-modifying method는 근본적으로 읽기만 허용하는 method(“getter”)로서 entity bean의 database 내부 상태를 어떤 식으로도 변경하지 않는다.

non-modifying method의 리스트를 보고 EJB engine은 ejbStore() method를 호출할지 말지 결정한다. 트랜잭션 중 또는 bean이 활성화된 상태일 때 non-modifying method만 실행되면 EJB engine은 ejbStore() 호출이 불필요하다고 판단하고 그 과정을 건너뛴다. 이는 전체적인 성능이 향상됨을 뜻한다.

### 11.2.8 ejbLoad(), ejbFind() CM Persistence 최적화 (CMP 1.1/2.0 에 해당)

전 절에서는 언제 ejbLoad()와 ejbStore() method들이 호출되어야 할지에 대하여 설명하였다. 이 설명은 BMP와 CMP 들이 ejbLoad()와 ejbStore() method들을 사용하기 때문에 BMP와 CMP entity bean에만 적용되었다.

이 절에서는 성능향상을 위해 어떻게 engine과 연관된 ejbLoad()와 ejbFind() method가 최적화 될 수 있는지에 대해 설명한다. CMP는 실제로 어떻게 ejbLoad(), ejbFind()가 구현되는지는 EJB engine에 달려있기 때문에 이 설명은 CMP bean(1.1과 2.0버전)에만 국한된다. BMP에서 이 method의 구현과 작동방식은 전적으로 개발자의 의향에 달려있다.

EJB engine이 CMP ejbLoad(), ejbFind()를 호출하면 다음과 같은 engine내의 하위모드를 선택하여 작동 방식을 최적화 할 수 있다.

- **ReadLocking:**. ejbLoad() 호출 시 EJB engine이 database의 shared lock을 가질 수 있게 한다. 이 종류의 lock은 다른 bean들이 같은 레코드를 읽을 수 있지만 쓰는 것은 금지하게 한다.

- **WriteLocking:**. ejbLoad() 호출 시 EJB engine이 database의 exclusive lock을 가질 수 있게 한다. 다른 bean들이 database에 읽고 쓰기를 모두 금지 시킨다.

- **WriteLockingFind:**. ejbLoad(), ejbFind() 호출시 EJB engine이 database의 exclusive lock을 가질 수 있게 한다. 다른 bean들이 database에 읽고 쓰기를 모두 금지 시킨다.

위의 세 가지 모드의 자세한 기술적 설명은 하지 않는다. 그 대신에 어느 상황에서 위의 모드들이 사용되어야 할지를 알아본다.

- CMP bean이 쓰기 작업보다는 읽기 작업을 더 빈번히 수행하면 항상 ReadLocking 모드를 선택한다.

- 반면, 읽기 보다는 쓰기 작업을 더 많이 수행하면 WriteLocking 모드나 WriteLockingFind 모드를 선택한다.

CMP에서는 engine 모드와 앞 절에서 설명한 non-modifying method와 이 절에서 설명한 engine내 하위모드가 규칙에 맞게 조합되어 설정되어야 한다. 이 모든 파라미터들의 설정 요약은 튜닝 부분을 살펴보라.

CMP bean 에서는 또한, `java.sql.ResultSet`의 Fetch Size를 크기나 EJB engine이 부팅할 때 캐싱되어야 할지나 EJB bean instance가 미리 인스턴스화 되어야 할지를 결정하는 “init caching”설정을 할 수 있다.

다음 절에서 더 자세하게 설명할 것이다.

### 11.2.9 Entity Bean 스키마 정보 (CMP 1.1/2.0 에 해당)

EJB engine이 bean의 persistence를 관리하는 책임을 가지고 있다면(CMP), entity bean instance 필드와 실제 database의 테이블과 컬럼들과의 매핑을 선언적으로 정의해주는 방법이 있어야 한다. 또한 database source도 지정해 주어야 할 것이다.

이 정보는 EJB XML 환경에 “schema info”로 기술한다.

CMP 1.1의 이 태그에는 EJB engine이 CMP 1.1 finder method를 생성할 때 기본적으로 사용될 SQL문도 들어있다.

다음 절에서 더 많은 설명을 할 것이다.

### 11.2.10 Entity Bean Relationship Mapping (CMP 2.0 에 해당)

EJB 2.1 스펙에는 database에 있는 관계(relation)를 지원하기 위하여 CMP relationship의 개념이 소개되어 있다.

CMP 2.0 bean들과 이 relationship들을 사용할 때에는 deployer가 JEUS EJB DD에 추가정보를 제공해줘야 한다.

다음 절에 더 많은 설명을 할 것이다.

### 11.2.11 Default JEUS Primary Key 클래스

`jeus.ejb.bean.objectbase.DefaultPK` 클래스는 JEUS에서 default primary key 로 사용되고 있다.

이 클래스는 ejb-jar.xml에 분명한 primary key 필드나 클래스가 없을 때 사용된다. 이 경우에 deployer는 반드시 ejb-jar.xml의 `<prim-key-class>` 태그에 `jeus.ejb.bean.objectbase.DefaultPK` 를 추가하여 준다

### 11.2.12 JEUS 의 EJB QL 추가 (CMP 2.0 에 해당)

CMP 2.0 bean은 ejb-jar.xml의 EJB-QL 문장과 연계되어 있는 home interface의 findByXXXX() method를 가지고 있어야 한다. JEUS는 표준 EJB-QL언어에 몇 가지 추가 사항을 더 지원한다. 이들은 11.4절에 기술되어 있다.

이 추가 지원 사항들은 ejb-jar.xml에 사용되거나 Instant EJB QL 요청에 사용될 수 있다. (다음절에서 자세히 설명한다). 11.4절에는 위의 예가 포함된 완전한 ejb-jar.xml의 예가 포함되어 있다.

### 11.2.13 Instant EJB QL (CMP 2.0 에 해당)

클라이언트 코드 내에서 CMP bean의 집합을 찾으려 하면 bean의 home interface에 선언되어 있는 `findByXXXX()` 메소드들에 의존 할 수 밖에 없다.

이 찾기 방법이 복잡한 경우에는 `findByXXXX()` 만으로는 부족할 수가 있다. 이러한 경우를 대비하여 JEUS에서는 클라이언트 코드에서 EJB-QL select 질의문들을 정의할 수 있는 비표준 인터페이스를 제공한다. 이 인터페이스는 `jeus.ejb.bean.objectbase.EJBInstanceFinder` 라고 불리며, JEUS EJB DD에 bean의 instant QL을 활성화시켜 놓았을 경우, entity bean의 home interface에서 구현한다.

이 인터페이스는 사용자의 EJB-QL을 home interface로 넘겨주는 메소드인 `findWithInstantQL(String ejbQLString)`을 가지고 있다. 이 메소드는 질의에 해당하는 EJB interface의 `java.util.Collection` 객체를 반환한다. 11.6절에서 이 메소드의 예가 있다.

부록 F에서 이 API에 대한 설명을 하고 있다.

### 11.2.14 자동 Primary Key 생성 (CMP 2.0 에 해당)

어떤 경우는 특정 primary key없이 EJB instance를 생성할 때도 있다. 이 경우 JEUS EJB engine에서 고유한 primary key를 생성해서 EJB 인스턴스를 만들 수도 있다. JEUS EJB engine은 데이터 베이스와 연동하여 이러한 기능을 제공한다.

11.6에 자세히 설명 된다.

### 11.2.15 결론

지금까지 세 종류의 entity bean과 관련된 JEUS의 기능을 살펴보았다. 세 종류가 모두 공통적으로 가지는 특성들인 object pool, engine 모드를 사용한 persistent optimization과 non-modifying 메소드, 그리고 engine 서브 모드를 통한 CM persistence optimization, CMP schema, CMP relationship, default primary key 클래스, instant EJB QL API에 대하여 살펴보았다.

다음 절에서는 어떻게 이 기본 특성들이 JEUS EJB DD에 설정되는지 자세히 살펴본다.

# 11.3 Entity EJB 설정

## 11.3.1 소개

다음과 같은 JEUS entity EJB의 설정법을 살펴보겠다.

- 기본적이고 공통적인 설정들(모든 entity bean에 적용가능).
- Object Pool (모든 entity bean에 적용가능).
- entity cache 설정을 포함한 Persistence 최적화(모든 entity bean에 적용가능).
- CM persistence optimization(CMP에만 해당).
- Schema (CMP에만 해당).
- Relationships (CMP 2.0에만 해당).
- Instant EJB QL (CMP 2.0에만 해당).

모든 이 특성들은 JEUS EJB모듈 DD (jeus-ejb-dd.xml) `<beanlist>` 태그 아래의 `<jeus-bean>`에 설정된다.

## 11.3.2 Entity EJB 의 기본 공통 설정항목 설정 (모든 entity bean 에 해당)

JEUS 에서 지원하는 모든 bean 종류에 적용 가능한 설정에 대해서는 6장에서 언급했었다. 이 설정들은 모든 종류의 entity bean들에게도 적용할 수 있다.

복습을 위하여 BMP entity bean의 기본 XML 태그의 예를 제공한다:

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>account</ejb-name>
            <export-name>ACCOUNTEJB</export-name>
            <local-export-name>
                LOCALACCOUNTEJB
            </local-export-name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
```

```
            <single-vm-only>true</single-vm-only>
            <local-invoke-optimize>true</local-invoke-optimize>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 11.3.3 Object Management 설정(모든 종류에 해당)

object Pool 설정에 대한 설명은 10장을 참고하라.

Entity cache에 대한 설정은 다음 절에서 설명한다.

### 11.3.4 ejbLoad/ejbStore Persistence 최적화 설정 (모든 종류에 해당)

Persistence method 호출의 최적화는 jeus-ejb-dd.xml의 각 bean descriptor에 설정되어 있다. 설정해야 할 세 가지 정보는 다음과 같다.

- **<engine-type>** EXCLUSIVE_ACCESS, SINGLE_OBJECT, MULTIPLE_OBJECT 중의 하나로 설정되어야 한다. 전 절에서 이 세 종류에 대한 차이점을 살펴보았었다.

- **<non-modifying-method> 의 리스트**. 이 리스트는 ejbStore()가 호출되어야 하는지에 대한 힌트를 EJB engine에 제공한다. 이것 또한 전 절에서 설명하였다. 이 태그는 CMP 2.0 bean에서는 적용되지 않는다.

- **<entity-cache-size>** Passivate되어야 할 entity bean instance들을 위한 내부 cache의 크기를 결정한다. Cache에 담을 수 있는 entity bean instance의 최대 크기 만큼이 주어진다. 크기가 클수록 좋은 성능을 가질 수 있지만 시스템 자원(주 메모리)은 그 만큼 더 많이 사용된다.

- **<update-delay-till-tx>**: 이 옵션은 Boolean 값을 가지며, EJB 데이터의 update, insert가 트랜잭션이 commit될 때까지 지연될지 여부를 나타낸다. 이 값이 "false"이면 모든 update, insert 작업은 즉각 실행므로, 동일한 트랜잭션 내에서는 commit()이 호출되기 전에도 변경 사항을 볼 수가 있다. 그러나 이렇게 하면 성능에 악영향을 미치게 된다. 기본값은 "true"이다.

다음은 object management 설정을 포함한 예이다.

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            <object-management>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </bean-pool>
                <connect-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </connect-pool>
                <capacity>2000</capacity>
                <passivation-timeout>10000</passivation-timeout>
                <disconnect-timeout>180000</disconnect-timeout>
            </object-management>
            <persistence-optimize>
                <engine-type>SINGLE_OBJECT</engine-type>
                <non-modify-method>
                  <method-name>myBusinessMethod</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                    <method-param>double</method-param>
                  </method-params>
                </non-modify-method>
                <non-modify-method>
                  <method-name> myBusinessMethod2</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
                    <method-param>int</method-param>
                  </method-params>
                </non-modify-method>
                <update-delay-till-tx>
                   false
```

```
                </update-delay-till-tx>
                <entity-cache-size>100</entity-cache-size>
            </persistence-optimize>
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 11.3.5 ejbLoad, ejbFind CM Persistence 최적화 설정 (CMP 1.1/2.0 에 해당)

CMP 1.1과 2.0 bean에는 ejbLoad() persistence 최적화를 설정할 수 있다. (11.2.8에서 설명하였다.)

최적화를 가능하게 하는 세 가지의 설정은 다음과 같다.

- **<subengine-type>** ReadLocking, WriteLocking 또는 WriteLockingFind 로 설정되어야 한다.
- **<fetch-size>** ResultSet으로 한 번에 가져올 수 있는 레코드의 수를 결정한다.
- **<init-caching>** 이 Boolean 스위치 값이 활성화되면 database 테이블의 각 레코드에 대하여 EJB engine이 미리 인스턴스화된 EJB entity instance를 생성한다. 이 작업은 engine이 시작될 때 수행된다. 비활성화가 되어 있으면 EJB instance들은 home interface의 create(), findByXXXX() 또는 비슷한 method 호출에 의해서만 생성된다.

예)

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <cm-persistence-optimize>
                <subengine-type>WriteLocking</subengine-type>
                <fetch-size>80</fetch-size>
```

```
                <init-caching>true</init-caching>
            </cm-persistence-optimize>
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 11.3.6 DB 스키마 정보 설정 (CMP 1.1/2.0 에 해당)

CMP 1.1/2.0에서는 database 테이블과 컬럼에 대응하는 EJB instance 필드와의 매핑을 포함하고 있는 DB 스키마 정보를 설정해야 한다. jeus-ejb-dd.xml에 다음과 같은 정보를 제공해야 한다.

- **<table-name>** 현재 EJB에 매핑되어야 할 관계형 database 테이블의 이름이다. 만약에 설정이 되어 있지 않으면 다음과 같은 테이블 이름이 임의로 설정된다. <EJB module name> + <EJB name>. 일부 DBMS들이 가지는 테이블 명 길이의 제한 때문에 15자 이내의 테이블 이름만이 허용된다.

- **<creating-table>** 이 스위치가 활성화 되어 있으면 EJB engine이 부팅될 시점에 테이블이 존재하지 않으면 지정해준 테이블 이름으로 테이블이 생성된다. 테이블을 생성하는 방식을 하위 태그로 지정해 주는데, <use-existing-table>이 지정되어 있다면 EJB module이 deploy될 시에 DB에 지정된 테이블이 없다면 새로 만들고 있다면 그 테이블을 그대로 사용한다. <force-creating-table>이 지정되어 있다면 DB에 테이블이 존재하더라도 그 table을 지우고 새로운 table을 생성한다. 또한 <use-existing-table>을 사용하고 DB에 테이블이 존재하거나 <creating-table>이 설정이 되어 있지 않으면 특수한 “schema check”가 활성화된다. 이 체크는 EJB engine이 부팅될 때 수행되고 존재하는 DB 테이블과 <schema-info> 태그의 스키마 설정이 맞는지 확인한다. 관리자는 engine container의 JVM 파라미터에 "-Djeus.ejb.checktable=false"를 지정하여 이 작업을 수행하지 않게 할 수 있다.

- **<deleting-table>** 이 스위치가 활성화되면 EJB module이 undeploy될 때 지정된 table이 DB에서 제거된다. 이 옵션은 실제 상황에서 아주 조심스럽게 쓰여야 하므로 실수를 방지하기 위해 JEUS의 system property로 별도의 지정을 해야 동작한다. 즉, JEUSMain.xml의 Command Option에 “-Djeus.ejb.enable.configDeleteOption=true”로 설정할 때만 동작한다. 또한 <creating-table>의 설정이 없는데 <deleting-

table> 설정을 사용하는 경우는 거의 없으므로 <creating-table>의 설정이 없을 때에는 <deleting-table> 설정이 실행되지 않도록 구현되어 있다.

- container managed field 매핑은 각각의 컬럼과 필드의 관계를 위해 존재한다.
    - **<field>** 는 컬럼에 매핑되어야 하는 EJB 필드의 이름을 지정한다.
    - **<column-name>** 은 주어진 database테이블에 존재하는 것과 같은 것이다.(지정되어 있지 않으면, EJB필드의 이름을 사용한다.)
    - **<type>** Database 컬럼에 정의된 데이터 형을 기준으로 한다. (예. “VARCHAR(25)”, “NUMERIC” 등.) 이 태그가 지정되어 있지 않으면, 디폴트 타입이 사용된다.(사용될 타입은 DB에 따라 다를 수 있다.) 이 매핑은 부록 C에서 다룬다. Oracle DBMS의 경우 “BLOB” (Binary Large Object) 과 “CLOB (Character Large Object)를 다룰 수 있다. 이 타입들은 이미지 같이 큰 자료 형에 적합하다. "CLOB" 타입의 경우 java.lang.String 에 적합하고 "BLOB"의 경우 serializable된 필드에 적합하다.
    - **<exclude-field>** 이 Boolean 옵션이 활성화 되어 있으면, 위에서 지정한 필드들을 매핑하지 않고, persistence management에 포함되게 한다. 이 옵션은 CMP 2.0 bean 에서만 작동한다. 이 옵션은 이전(migration)을 위한 목적으로 사용된다.
- **<primary-key-field>** 이 리스트는 bean 과 DB 레코드의 primary field를 생성하여 모든 EJB 필드의 이름을 선언한다. 만약 지정되지 않으면 ejb-jar.xml에 있는 primary key 클래스의 내용을 찾아서 그 모든 public 필드들이 primary key(PK)를 형성한다고 가정한다. 이 태그는 primary key 클래스가 사용되어야만 유효하다.
- CMP 1.1에서는 EJB engine에서 구현되는 finder method들을 위하여 SQL 문장들이 지정되어 있어야 한다. 이 SQL들을 정의하기 위하여 두 가지가 설정되어야 한다.
    - SQL 문장이 사용될 finder method의 이름과 파라미터들.

- **SQL** 문장, 이 문장은 finder method를 생성할 때 사용된다. 지정된 문장에는 “where” 절 다음에 나오는 부분만을 포함하고 있어야 한다. 다른 요소는 자동으로 주어 진다. 여기에는 “?” 문자를 지정할 수 있고 이는 finder method의 파라미터 값들을 대신하게 된다. “<”같은 특수문자도 쓸 수 있다.(이 SQL은 WHERE에 포함된다).

- **<include-updates> 태그** : 이 설정이 "true"로 설정이 되면, finder 메소드 호출 동안 변경된 사항이 데이터베이스에 업데이트 되므로 finder 메소드에서 변경된 사항을 볼 수 있다.

- **<db-vendor>** 태그는 이 bean을 위해 사용되는 database의 벤더를 지정한다. 벤더 이름은 벤더 특정 최적화를 하기 위하여 EJB engine이 사용하게 된다. 여기서 지정된 값이 실제로 성능에 얼마나 많은 영향을 미칠 수 있는지는 확인된 바 없다. 사용 가능한 벤더들의 이름은 부록 F에 지정되어 있다. 일반적인 예는 “oracle”을 들 수 있다. 이 값은 또한 Java 객체와 벤더가 정한 database 필드 타입과의 정확한 매핑을 가려내기 위하여 사용된다.

- database와 연결을 맺을 때 사용되는 database connection pool의 JNDI export name(data source name)이 있다. 이 connection pool은 JEUSMain.xml에 설정되어 있고 JEUS의 관리자 JVM에서 실행되고 있다.

- 마지막으로, <**auto-key-generator**> CMP 2.0에서 자동으로 primary key를 생성하기 위해 사용하며 하위 태그들이 있다. 하위태그들은 11.6절을 참조하기 바란다.

- **<jeus-query>** 태그는 JEUS 확장 EJB-QL을 포함한 EJB-QL을 query 메소드에서 지정할 수 있게 해준다. 이것은 ejb-jarxml의 <query> 태그와 비슷하다. 11.4절에서 자세하게 설명한다. 이 설정은 BEA Weblogic에서 JEUS 5.0 으로의 마이그레이션을 간단하게 하기 위해서 제공하는 기능이다.

예)

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
    . . .
    <beanlist>
```

```
        . . .
        <jeus-bean>
            . . .
            <schema-info>
                <table-name>ACCOUNT</table-name>
                <creating-table>
                  <use-existing-table/>
                </creating-table>
                <deleting-table>true</deleting-table>
                <cm-field>
                    <field>id</field>
                    <column-name>ID</column-name>
                    <type>NUMERIC</type>
                    <exclude-field>false</exclude-field>
                </cm-field>
                <cm-field>
                    <field>customer_addr</field>
                    <column-name>customer_address</column-name>
                    <type>VARCHAR(30)</type>
                    <exclude-field>false</exclude-field>
                </cm-field>
                <prim-key-field>
                    <field>id</field>
                </prim-key-field>
                <find-method>
                  <method>
<method-params>findByAddress</method-name>
                      <method-params>
                        <method-param>
                          java.lang.String
                        </method-param>
                      </method-params>
                  </method>
                  <sql>customer_address=?</sql>
                </find-method>
                <db-vendor>oracle</db-vendor>
                <data-source-name>MYDB</data-source-name>
                <auto-key-generator> <!-- See section 11.6 -->
                    <generator-type>
```

```
                        Oracle
                    </generator-type>
                    <generator-name>
                        my_generator
                    </generator-name>
                    <key-cache-size>
                        20
                    </key-cache-size>
                </auto-key-generator>
                <jeus-query>
                  <query-method>
                   <method-name>findByTitle</method-name>
                   <method-params>
                    <method-param>
                       java.lang.String
                    </method-param>
                   </method-params>
              </query-method>
              <jeus-ql>
  SELECT OBJECT(b) FROM Book b WHERE b.title = ?1 ORDERBY b.price
              </jeus-ql>
            </jeus-query>
          </schema-info>
             . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 11.3.7 Relationship Mapping 설정 (CMP 2.0 에 해당)

이 절은 one-to-one/one-to-many와 many-to-many relationship의 두 부분으로 나뉘어져 있다.

#### One-to-one/one-to-many Relationship 매핑

두 bean간에 one-to-one/one-to-many relationship을 설정할 때, 다음과 같은 정보를 제공해 줘야 한다:

- **<relation-name>** ejb-jar.xml 파일에 정의되어 있다.

- 두 개의 JEUS relationship의 역할(relationship의 각 방향별로 하나). 각 relationship의 역할에 필요한 정보는 다음과 같다.

    - **<relationship-role-name>** ejb-jar.xml에 정의되어 있다. (<ejb-relationship-role-name> 태그에서 설정함)

    - **<column-map>** 두 개의 하위 태그인 <**foreign-key-column**> 과 <**target-primary-key-column**>을 가진다. 전자는 relationship에서 foreign key의 역할을 하는 컬럼의 이름이고, 후자는 foreign key 컬럼이 매핑되어야 할 테이블의 primary key 필드이다. Key가 여러 개의 컬럼으로 구성되어 있으면 복수 개의 컬럼 대 컬럼 매핑을 지정할 수 있다.

아래에는 “employee-manager”라는 관계를 가지는 두 bean의 many-to-one(= one-to-many)관계의 예를 보여주고 있다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <ejb-relation-map>
        <relation-name>employee-manager</relation-name>
        <jeus-relationship-role>
            <relationship-role-name>
                employee-to-manager
            </relationship-role-name>
            <column-map>
                <foreign-key-column>
                    manager-id
                </foreign-key-column>
                <target-primary-key-column>man-id</target-
primary-key-column>
            </column-map>
        </jeus-relationship-role>
    </ejb-relation-map>
    . . .
</jeus-ejb-dd>
```

위의 XML예에서는 employee 테이블의 “manager-id”라는 foreign key 컬럼이 employee-to-manager의 many-to-one관계를 맺기 위하여 manager 테이블의 primary key 필드와 매핑되어 있다.

One-to-one 매핑도 이와 비슷한 방법으로 설정된다. 한 가지 기억할 것은 one-to-one 관계에서는 primary key 컬럼과 foreign key 컬럼이 테이블 어디에 존재해도 상관없으며, One-to-many에서는 반드시 "many" 쪽의 테이블이 foreign key 컬럼을 가지고 있다는 것이다.

### Many-to-many Relationship 매핑

두 bean간에 many-to-many 관계를 맺기 위해서는 one-to-one의 관계에서 설정된 정보 외에 추가로 다음의 정보가 반드시 설정되어야 한다.

- Many-to-many 관계의 join 테이블 이름을 지정한다.
- 두 **JEUS relation role**에서 Join 테이블의 두 foreign key 컬럼을 두 bean이 사용하는 두 테이블의 primary key 컬럼에 매핑한다.

아래에는 many-to-many 관계 설정을 나타내는 "student-course"의 예 이다:

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
        . . .
    <ejb-relation-map>
        <relation-name>student-course</relation-name>
        <table-name>STUDENTCOURSEJOIN</table-name>
        <jeus-relationship-role>
            <relationship-role-name>
                student
            </relationship-role-name>
            <column-map>
                <foreign-key-column>
                    student-id <!--This column in join table-->
                </foreign-key-column>
                <target-primary-key-column>
                    stu-id <!--This column in student table-->
                </target-primary-key-column>
            </join-column-map>
        </jeus-relationship-role>
        <jeus-relationship-role>
            <relationship-role-name>
                course
```

```
            </relationship-role-name>
            <column-map>
                <foreign-key-column>
                    course-id <!--This column in join table-->
                </foreign-key-column>
                <target-primary-key-column>
                    cou-id <!--This column in course table-->
                </target-primary-key-column>
            </column-map>
        </jeus-relationship-role>
    </ejb-relation-map>
        . . .
</jeus-ejb-dd>
```

위의 예에서 어떻게 두 JEUS relationship role이 사용되고 있는지 관찰해보기 바란다. 각 role은 Join 테이블의 두 컬럼 중 한 개와 실제 bean이 사용하는 테이블의 primary key 컬럼이 매핑하도록 선언하고 있다.

### 11.3.8 Instant EJB QL 의 설정 (CMP 2.0 만 해당)

클라이언트 쪽에 instant EJB QL 질의를 가능하게 하려면(부록 F와 11.4절을 보라), JEUS EJB DD의 `<enable-instant-ql>` 태그에 “true” 값을 설정해 준다.

예)

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <enable-instant-ql>true</enable-instant-ql>
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 11.3.9 Database Insert Delay 구성 (CMP Only)

CMP에서 EJB가 생성될 때 EJB 데이터가 backend 데이터베이스에 기록되는 시점을 설정할 수 있다.

- ejbCreate : ejbCreate()를 실행 후, ejbPostCreate()를 실행하기 전에 새로운 EJB 데이터를 삽입한다.
- ejbPostCreate : ejbCreate()와 ejbPostCreate()를 완성한 후에 새로운 EJB 데이터를 삽입한다.

이 설정은 jeus-ejb-dd.xml에서 <jeus-bean> 태그 아래의 <database-insert-delay> 태그 안에서 구성된다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <database-insert-delay>
                ejbCreate
            </database-insert-delay>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

### 11.3.10 결론

이것으로 JEUS의 entity bean 설정에 대한 설명을 모두 마친다.

지금까지 다음과 같은 설정들을 살펴보았다.

- 기본 및 공통 설정들 (모든 entity bean에 적용).
- Object management (모든 entity bean에 적용).
- Persistence optimization (모든 entity bean에 적용).

- CM persistence optimization (CMP 에 해당).
- Schema (CMP 에 해당).
- Relationship (CMP 2.0 에 해당).
- Instant EJB QL (CMP 2.0 에 해당).

다음 절에서는 JEUS CMP 2.0의 instant EJB QL API를 이용하는 클라이언트 어플리케이션이 어떻게 프로그램되는지 간략하게 살펴보자.

## 11.4 CMP 2.0 을 위한 JEUS EJB QL 추가사항

### 11.4.1 소개

CMP 2.0 bean의 findByXXX() method와 ejbSelectXXX() method에서는 그 home interface와 ejb-jar.xml의 EJB-QL문장이 연계되어 있어야 한다. JEUS는 표준 EJB-QL 언어에 몇 가지 사항들을 추가한다. 이 추가 사항들은 다음 절에서 설명한다.

이 추가 사항들은 ejb-jar.xml 파일이나 EJB QL에서 사용된다. 11.5절에서는 이 기능이 사용된 ejb-jar.xml의 예를 보여준다.

**주의**: 여기에서 제공된 EJB QL extension들은 표준 EJB 2.1 QL에서 지원하지 않는 항목들로서, 이것을 이용하여 작성된 EJB어플리케이션은 J2EE 표준에 준하지 않으므로 이식성이 떨어진다. 이 의미는 여기서 제공된 확장 기능을 사용하여 만든 EJB 컴포넌트는 다른 J2EE 서버에 deploy될 수 없다는 것이다.

JEUS의 EJB QL extension은 대부분 표준 SQL처럼 작동한다. 그러므로, 더 상세한 정보는 SQL 을 참고하기 바란다.

이제부터 다음과 같은 것들에 대하여 살펴보겠다.

- JEUS EJB QL extension 키워드 추가사항
- JEUS EJB QL extension Subquery
- JEUS EJB QL extension ResultSet
- JEUS EJB QL extension의 기본 집합(Aggregation) 함수들
- JEUS EJB QL extension의 추가적인 집합 함수들

- JEUS EJB QL extension의 Dynamic Query
- JEUS EJB QL extension GROUP BY 키워드

### 11.4.2 JEUS EJB QL extension 키워드 추가사항

JEUS에서는 다음과 같은 비표준 keyword들이 EJB QL 문장에 사용될 수 있다.

- **ORACLEHINT** keyword. “oraclehint<표준 Oracle hint string>”의 문법을 가진다. 이 keyword는 Oracle의 문서를 참고하라.

```
SELECT OBJECT(b) ORACLEHINT '/*+ALL_ROWS*/'
FROM Book b
```

### 11.4.3 JEUS EJB QL extension Subquery

JEUS EJB QL subquery는 EJB QL 질의의 하나로 WHERE절에 포함된 또 다른 질의절을 말한다. 이것은 SQL 질의와 subquery의 관계와 비슷하다.

다음의 예에서는 중첩된 EJB QL 질의의 한 예로서 평균 이상의 월급을 받는 고용인들을 질의한다.

```
SELECT OBJECT(e)
FROM EmployeeBean AS e
WHERE e.salary >
  (SELECT AVG(e2.salary)
   FROM EmployeeBean AS e2)
```

다음 하위 절에서는 subquery들의 세부 사항들을 설명한다.

#### Subquery 리턴 타입들

JEUS EJB QL 질의의 return type들로 다음과 같은 것들이 올 수 있다.

- Single value 또는 <cmp-field>에 대응하는 Collection (위의 예에서 e.salary와 같은 값들)
- 집합 함수를 통해서 만들어진 한 개의 return 값 (예, MAX(e.salary)).
- 간단한 primary key(compound primary key가 아닌)를 포함한 cmp-bean. (예: SELECT OBJECT(emp))

### 비교 연산자로서의 Subquery

JEUS EJB QL은 subquery를 비교 연산자의 피연산자로 사용할 수 있다.

JEUS EJB QL은 비교 연산자는 다음과 같다.

- [NOT] IN, [NOT] EXISTS, [NOT] UNIQUE )
- <, >, <=, >=, =, <>
- 위 연산자와 ANY, ALL, SOME의 조합

다음의 예는 매니저가 아닌 직원을 추출한다.

```
SELECT OBJECT(employee)
FROM EmployeeBean AS employee
WHERE employee.id NOT IN
  (SELECT employee2.id
   FROM ManagerBean AS employee2)
```

다음은 subquery 결과값이 없는지 테스트 하는 **NOT EXISTS** 연산자의 예이다. ("NOT"은 값이 있는지를 테스트하기 위해 제거될 수 있다.):

```
SELECT OBJECT(cust)
FROM CustomerBean AS cust
WHERE NOT EXISTS
  (SELECT order.cust_num
   FROM OrderBean AS order
   WHERE cust.num = order.cust_num)
```

위 질의의 결과는 주문을 하지 않은 모든 고객의 명단이 된다.

마지막 예는 상호 연관된(correlated) 질의라고 할 수 있는데 그 이유는 subquery의 결과가 상위 질의의 파라미터로 반영이 되어야 하기 때문이다. 의존 관계가 없는 질의를 비상호연관(uncorrelated) 질의라고 한다.

다음의 예는 **UNIQUE** 연산자의 사용을 보여주고 있다. 이 연산자는 result set의 row 중 중복된 것이 있는지를 확인한다. 0개 또는 한 개의 row이거나, subquery의 결과 내에 있는 모든 레코드가 유일하다면 TRUE를 리턴한다. 이 연산자에 NOT을 붙이면 반대의 결과를 얻게 된다.

```
SELECT OBJECT(cust)
FROM CustomerBean AS cust
```

```
WHERE UNIQUE
  (SELECT cust2.id FROM CustomerBean AS cust2)
```

다음은 “>” 연산자의 예로서, 다른 연산자와 마찬가지로 subquery의 결과값과 사용될 수 있다.

```
SELECT OBJECT(employee)
FROM EmployeeBean AS employee
WHERE employee.salary >
  (SELECT AVG(employee2.salary)
   FROM EmployeeBean AS employee2)
```

위의 JEUS EJB QL 질의는 평균 월급보다 많이 받는 모든 고용인의 결과를 리턴한다.

참고로, ANY, ALL, SOME이 사용되지 않는 경우에는 반드시 한 개의 값만이 subquery의 결과로 리턴되어야 한다. (이 경우에서는 평균 월급)

ANY, ALL, SOME 연산자는 하나의 값보다 더 많은 값을 리턴하는 subquery에 사용한다. 이 연산작업은 다음과 같이 해석될 수 있다.

- <Operand> <arithmetic operator> **ANY** <subquery>: subquery의 결과 중 적어도 한 개의 값이 “true”를 리턴하면 “true”를 리턴한다.
- <Operand> <arithmetic operator> **SOME** <subquery>: “ANY”와 같은 의미를 가진다.
- <Operand> <arithmetic operator> **ALL** <subquery>: subquery의 결과 모두가 “true”를 리턴해야 “true”를 리턴한다.

**중요**: XML DD파일에 “>”와 “<” 문자를 넣으려 할 때는 CDATA 부분에 포함시켜줘야 한다. 그렇지 않으면 XML 파서가 혼동을 일으키고 XML 태그 지정문자로 인식한다.

예)

*<<ejb-jar.xml>>*

```
. . .
<query>
    <description>Method finds large orders</description>
    <query-method>
        <method-name>findLargeOrders</method-name>
```

```
        <method-params></method-params>
    </query-method>
    <ejb-ql>
  <![CDATA[SELECT OBJECT(o) FROM Order o WHERE o.amount > 1000]]>
    </ejb-ql>
</query>
. . .
```

### 11.4.4 JEUS EJB QL extension ResultSet

JEUS EJB engine은 복수 컬럼인 java.sql.ResultSet 객체를 반환하는 ejbSelect() 질의를 지원한다. 그러므로 ejbSelect method의 SELECT문 안에 콤마로 구분된 타깃 필드의 리스트를 지정할 수 있다.

```
SELECT employee.name, employee.id, employee.salary
FROM EmployeeBean AS employee
```

위의 질의는 모든 EmployeeBean 인스턴스의 name, id, salary로 구성된 java.sql.ResultSet을 생성한다. 이 ResultSet은 “name”, “id”, “salary”의 컬럼으로 구성된 레코드를 포함하게 된다. 이 ResultSet의 각 레코드는 해당하는 entity EJB 인스턴스를 표현하게 된다.

ResultSet들은 cmp-field 값(bean이나 relationship filed가 아닌)만을 반환한다.

**중요**: ejbSelect() method를 사용하여 java.sql.ResultSet종류의 객체를 얻어와 모든 작업을 마쳤을 때에는 ResultSet.close()를 호출해 줘야한다. 이 close() method는 EJB engine에 의해 자동적으로 호출되지 않는다.

### 11.4.5 JEUS EJB QL extension Dynamic Query

JEUS EJB QL은 EJB QL 질의를 프로그램적으로 구현할 수 있도록 지원한다. 이것은 Instant EJB QL 기능으로 가능한 것이다. 이 절에서는 JEUS CMP 2.0에서 Instant EJB QL을 프로그래밍 할 때 필요한 기본적인 사항들을 알아본다.

다음 코드에서는 CMP 2.0 home interface의 instant EJB QL 사용법을 보여주고 있다. 이 코드는 순수하게 클라이언트 쪽의 것이다. 이 코드가 작동되도록 하려면 instant EJB QL을 사용하도록 CMP bean에 설정되어 있어야 한다 (이 부분에 대해서는 11.3에서 설명하고 있다).

```
import jeus.ejb.bean.objectbase.EJBInstanceFinder;
import java.util.Collection;
. . .
```

```
Context initial = new InitialContext();
Object objref = initial.lookup("emp");
EmpHome home = (EmpHome)
   PortableRemoteObject.narrow(objref, EmpHome.class);
EJBInstanceFinder finder = (EJBInstanceFinder) home;
java.util.Collection foundBeans =
   finder.findWithInstantQL("<EJB QL query sentence>");
// Use foundBeans collection…
. . .
```

위 예에서의 "<EJB QL query sentence>"는 파라미터 없는 완전한 EJB QL 문장으로 대체되어야 한다("?"가 허용되지 않는다).

이 API에 대해서는 부록 F를 참고하라.

### 11.4.6 JEUS EJB QL extension GROUP BY 키워드

JEUS EJB QL 은 **GROUP BY** 와 **GROUP BY..HAVING** 절을 지원한다. 이들은 다음과 같이 작동한다.

- <SELECT statement> GROUP BY <grouping column(s)>: SELECT 문장이 실행되면 레코드들이 생성되고, GROUP BY 문장은 <grouping column(s)>에 지정된 컬럼(들)의 셋을 검색한다. 같은 값을 가지는 이 컬럼(들) 내의 모든 레코드들은 하나의 레코드로 합쳐지고 중복되는 것들은 버려진다.

  ```
  SELECT employee.departmentName
  FROM EmployeeBean AS employee
  GROUP BY employee.departmentName
  ```

  위 질의는 EmployeeBean의 유일한 모든 department 이름의 리스트를 결과 값으로 가진다.

- <SELECT statement> GROUP BY <grouping column(s)> HAVING <condition>: 이런 형식의 문장은 바로 전의 것과 같은 방식으로 작동하지만, <condition>에 설정된 것을 만족시키는 컬럼 그룹만이 포함된다.

  ```
  SELECT employee.departmentName
  FROM EmployeeBean AS employee
  GROUP BY employee.departmentName
  HAVING COUNT(employee.departmentName) >= 3
  ```

위의 질의는 두 개 이상의 EmployeeBean 인스턴스가 발견된 모든 유일한 department 이름들의 리스트를 반환한다 (즉, 3 이상의 employee 들을 가진 department 의 리스트).

예)

*표 10. EmployeeBean instance 을 위한 EJB 필드들*

| **id** | **departmentName** |
|---|---|
| johnsmith | R&D |
| peterwright | R&D |
| lydiajobs | R&D |
| catherinepeters | Marketing |

[표 10]의 데이터에 EJB QL 질의를 실행시키면, [표 11]과 같은 결과가 나온다.

*표 11. 결과 ResultSet*

| **departmentName** |
|---|
| R&D |

이 결과는 HAVING 조건이 적어도 세 개의 레코드를 가진 컬럼 그룹을 포함시켜야 한다고 명시되었기 때문이다.

더 자세한 GROUP BY, GROUP BY … HAVING 절에 대한 정보는 SQL 참고 문서들을 살펴보기 바란다.

**참고**: JEUS EJB QL에서는 GROUP BY와 GROUP BY HAVING 문장을 subquery 또는 최상위의 질의에서 사용할 수 있다. 후자의 경우에는 java.lang.String 객체의 집합이 리턴된다. 최상위의 GROUP BY와 GROUP BY HAVING 문장은 bean 객체를 리턴할 수 없다.

### 11.4.7 결론

지금까지 우리는 JEUS CMP에 관련된 EJB QL extension에 대하여 살펴보았다. 이 확장자들은 새로운 keyword들과 subquery들, ejbSelect() method에 의해 리턴되는 ResultSet, 집합함수, 다양한 EJB QL 질의 그리고, GROUP BY keyword들을 살펴보았다.

다음 장에서는 이 장에서 설명한 EJB QL 확장자들이 포함되어 있는 완전한 EJB CMP 2.0 entity bean 설정 예를 살펴본다.

## 11.5 완전한 CMP 2.0 Entity Bean 예

### 11.5.1 소개

어떻게 EJB코드, 표준 EJB DD, JEUS DD가 함께 설정되고 작동될 수 있는지 보여주기 위하여, 이번 절에서는 완전한 CMP 2.0 bean의 예를 제시한다. 이 예제는 책을 개념으로 모델화한 Book EJB이다.

이 절에서는 기본적으로 모든 예에 주석을 달지 않았고 CMP 2.0 bean이 jeus-ejb-dd.xml에 어떻게 설정되는 지만 보여준다

**참고**: JEUS 의 ejb-jar.xml에서 EJB QL 사용 예를 잘 살펴보기 바란다.

### 11.5.2 Remote Interface

*<<Book.java>>*

```
package test.book;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Book extends EJBObject {

    public String getTitle() throws RemoteException;
    public void setTitle(String title) throws RemoteException;
    public String getAuthor() throws RemoteException;
    public void setAuthor(String author) throws
                                          RemoteException;
    public double getPrice() throws RemoteException;
    public void setPrice(double price) throws RemoteException;
    public String getPublisher() throws RemoteException;
```

```
    public void setPublisher(String publisher) throws
                                                    RemoteException;
    public String toBookString() throws RemoteException;
}
```

### 11.5.3 Home Interface

*<<BookHome.java>>*

```
package test.book;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import javax.ejb.EJBHome;
import java.util.*;

public interface BookHome extends EJBHome {

    public Book create(String code, String title, String  author,
                       double price, String publisher) throws
                                CreateException, RemoteException;
    public Book findByPrimaryKey(String code) throws
                                FinderException, RemoteException;
    public Collection findByTitle(String title) throws
                                FinderException, RemoteException;
    public Collection findInRange(String from, String to) throws
                                FinderException, RemoteException;
    public Collection findAll() throws FinderException,
                                RemoteException;
}
```

### 11.5.4 Bean Implementation

*<<BookEJB.java>>*

```
package test.book;

import javax.ejb.EntityBean;
import javax.ejb.EntityContext;

public abstract class BookEJB implements EntityBean {
```

```
    public String ejbCreate(String code, String title,
        String author, double price, String publisher) {
        setCode(code);
        setTitle(title);
        setAuthor(author);
        setPrice(price);
        setPublisher(publisher);
        return null;
    }

    public void ejbPostCreate(String code, String title, String
                author, double price, String publisher) {}

    public abstract String getCode();
    public abstract void setCode(String code);
    public abstract String getTitle();
    public abstract void setTitle(String title);
    public abstract String getAuthor();
    public abstract void setAuthor(String author);
    public abstract double getPrice();
    public abstract void setPrice(double price);
    public abstract String getPublisher();
    public abstract void setPublisher(String publisher);

    public String toBookString() {
        return getCode() + "- [" + getTitle() + "] by " +
                    getAuthor() + " | " + getPrice() + " | " +
                    getPublisher();
    }

    public BookEJB() {}
    public void setEntityContext(EntityContext ctx) {}
    public void unsetEntityContext() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove(){}
}
```

### 11.5.5 J2EE EJB DD

*<<ejb-jar.xml>>*

```
<?xml version="1.0"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee" version="2.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
   <enterprise-beans>
      <jeus-bean>
         <ejb-name>BookBean</ejb-name>
         <home>test.book.BookHome</home>
         <remote>test.book.Book</remote>
         <ejb-class>test.book.BookEJB</ejb-class>
         <persistence-type>Container</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
         <primkey-field>code</primkey-field>
         <reentrant>False</reentrant>
         <cmp-version>2.x</cmp-version>
         <abstract-schema-name>Book</abstract-schema-name>
         <cmp-field><field-name>code</field-name></cmp-field>
         <cmp-field><field-name>title</field-name></cmp-field>
         <cmp-field><field-name>author</field-name>
         </cmp-field>
         <cmp-field><field-name>price</field-name></cmp-field>
        <cmp-field><field-name>publisher</field-name></cmp-field>
         <query>
            <query-method>
            <method-name>findByTitle</method-name>
            <method-params>
               <method-param>
               java.lang.String
               </method-param>
            </method-params>
         </query-method>
         <ejb-ql>
  SELECT OBJECT(b) FROM Book b WHERE b.title = ?1 ORDERBY b.price
         </ejb-ql>
         </query>
```

```
        <query>
           <query-method>
              <method-name>findInRange</method-name>
              <method-params>
                 <method-param>
                 java.lang.String
                 </method-param>
                 <method-param>
                 java.lang.String
                 </method-param>
              </method-params>
           </query-method>
           <ejb-ql>
  SELECT OBJECT(b) FROM Book b WHERE b.title BETWEEN ?1 AND ?2
           </ejb-ql>
        </query>
        <query>
           <query-method>
           <method-name>findAll</method-name>
           <method-params/>
           </query-method>
           <ejb-ql>
  SELECT OBJECT(b) ORACLEHINT '/*+ALL_ROWS*/' FROM Book b
           </ejb-ql>
         <query>
      </jeus-bean>
   </enterprise-beans>
   <assembly-descriptor>
       <container-transaction>
           <method>
               <ejb-name>BookBean</ejb-name>
               <method-name>*</method-name>
               <method-params/>
           </method>
           <trans-attribute>Required</trans-attribute>
       </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

**참고**: 굵은 글자의 3줄에 주목하기 바란다. 이들은 비표준의 JEUS 전용 EJB QL이다. (“ORDERBY”, “BETWEEN”, “ORACLEHINT”).

### 11.5.6 JEUS EJB DD

*<<jeus-ejb-dd.xml>>*

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
   <module-info/>
   <beanlist>
      <jeus-bean>
         <ejb-name>BookBean</ejb-name>
         <export-port>0</export-port>
         <export-name>book</export-name>
         <export-iiop>false</export-iiop>
         <local-invoke-optimize>false</local-invoke-optimize>
         <schema-info>
            <table-name>Booktable</table-name>
            <creating-table>
                  <use-existing-table>
            </creating-table>
            <deleting-table>true</deleting-table>
            <cmfield><field>code</field></cmfield>
            <cmfield><field>title</field></cmfield>
            <cmfield><field>author</field></cmfield>
            <cmfield><field>price</field></cmfield>
            <cmfields><field>publisher</field></cmfields>
            <db-vendor>oracle</db-vendor>
            <data-source-name>datasource1</data-source-name>
         </schema-info>
         <persistence-optimize>
            <engine-type>EXCLUSIVE_ACCESS</engine-type>
            <non-modifying-method>
                  <method-name>getTitle</method-name>
            </non-modifying-method>
            <non-modifying-method>
                  <method-name>getAuthor</method-name>
            </non-modifying-method>
            <non-modifying-method>
                  <method-name>getPrice</method-name>
```

```
            </non-modifying-method>
            <non-modifying-method>
                  <method-name> getPublisher </method-name>
            </non-modifying-method>
            <non-modifying-method>
                  <method-name>toBookString</method-name>
            </non-modifying-method>
         </persistence-optimize>
         <enable-instant-ql>true</enable-instant-ql>
      </jeus-bean>
   </beanlist>
</jeus-ejb-dd>
```

### 11.5.7 결론

이번 장에선 CMP 2.0 “Book”의 예를 통해 JEUS EJB QL을 살펴보았다.

이것을 deploy하려면 패키징 후에 EJB로 deploy하면 된다. 이것은 5장을 참조하라

다음 장에서는 CMP 2.0에서 primary key 자동생성에 대해서 알아 보겠다.

## 11.6 자동 Primary Key 생성 (CMP 1.1 & CMP 2.0)

### 11.6.1 소개

경우에 따라서, 개발자들은 primary key 생성 없이 EJB인스턴스 만들기를 원할 때도 있다. 이 경우 JEJS EJB engine이 새 EJB 인스턴스에게 할당할 고유의 primary key를 생성하도록 설정 할 수 있다. JEUS EJB engine은 이 기능을 데이터베이스와 연동하여 제공한다.

JEUS의 경우, 자동 primary key 생성 기능이 CMP1.1과 CMP2.0에 적용 된다.

### 11.6.2 자동 Primary Key 생성의 사용법과 적용법

이번 절에선 JEUS에서 자동 primary key 생성 기능의 사용법에 대해 기술 한다.

1. 개발자들은 습관적으로 CMP1.1/CMP2.0을 파라미터 없이 create()를 호출한다. 이것은 create() 메소드가 EJB의 primary key가 되는 파라미터를 갖지 않는다는 뜻이다.

예)

```
InitialContext ctx = new InitialContext();
BookHome bHome = (BookHome) ctx.lookup("bookApp");
Book b = bHome.create(“JEUS EJB Guide”, ”Park Sungho”);
// Start using “b”.
. . .
```

위의 예제에서 EJB 홈인 BookHome를 찾고 EJB 홈에서 파라미터가 두 개인 create()를 호출 한다(“JEUS EJB Guide” 와 “Park Sungho”). 이 파라미터는 새로운 책의 제목과 저자이름을 뜻한다. 생각해보면 이 파라미터들을 primary key로 이용하는 것은 적합 하지 않다(같은 제목의 책이 있을 수 있고, 여러 저자가 같은 제목의 책을 낼 수도 있기 때문이다).

이 기능을 지원하기 위해, EJB engine은(“A”라고 부른다) 반드시 고유의 primary key를 생성해서 새 책에 할당 해야 한다. 그리고 여러 engine들(“B”, “C”)이 클러스터링 된 상태에서도 같은 일이 동시에 벌어 질 수 있다는 것을 고려 해야 한다.

광범위한 클러스터링 중에도 각 engine들은  고유한 primary key을 얻을 수 있어야 하기 때문에 하나의 중앙 저장 장치를 필요로 한다. 이 저장 장치는 하나의 컬럼과 하나의 데이터로 이루어진 데이터베이스로 구현 된다. 이 하나의 값은 EJB engine과 연동하면서 계속 증가하는 primary key counter(또는 primary key generator)이다. 이 방법으로 데이터베이스는 항상 클러스터링된 EJB engine이 읽을 수 있는 새롭고 고유한 primary key를 유지 할 수 있다.

2. primary key가 없는  create()가 호출 되고 나면, EJB engine “A” 는  중앙 저장 장치(primary key database)로 접근한다. 이 저장 장치로부터, 정수로 된 primary key를 받아오고 Book 인스턴스에 할당한다.

3. EJB engine “A”는 저장장치에 있는 미리 결정된 값으로 primary key를 증가 시킨다. 미리 결정된 값은 기본적으로 “1”이다 그러나 1이상을 쓰는 것을 권장한다(“20”). 미리 결정된 값은 “key cache size” 로 불린다. 만약 key cache size가 “20”이라면, EJB engine “A”는 “19” (20-1)를 할당한다. 다음에 primary key를 읽는 다른  EJB engine (“B” 와 “C”) 은 가장 최근에 primary key를 읽어간 것보다 20이 큰 값을 읽게 된다.

   이것은 EJB engine “A”는 19개까지는 외부의 primary key저장 장치에 접근 없이 EJB 인스턴스를 만들 수 있다는 것을 의미한다. 따라서 성

능향상에 큰 기여를 한다. engine “A”는 할당 받은 primary key의 counter를 유지 함으로써, 이 값을 다 소비 했을 때만 외부에서 새로운 primary key를 읽어 오면 된다. 내부적인 counter는 물론 primary key 를 생성 하는데 쓰이게 된다.

자동 primary key 생성의 몇 가지 특징:

- Oracle 과 MS SQL Server 는 자동으로 primary key sequence를 제공한다. 11.6.3과 11.6.4에서 자동 primary key생성 설정에 대해서 논의 할 것이다.
- Non-Oracle 과 non-MS SQL Server 는 직접 primary key 테이블을 작성해야 한다. 11.6.5에서 논의 할 것이다.
- 이 기능을 사용하기 위해선 데이터베이스는 반드시 Transaction Isolation Level TRANSACTION_SERIALIZABLE (isolation level) 을 지원해야 한다. 여기서는 지원한다고 가정한다.
- primary key 는 EJB에서 반드시 정의 되어야 한다. primary key는 반드시 java.lang.Integer 타입이고, 복합키가 되면 안되며, EJB deployment descriptor 에 선언되어야 한다.

**중요**: 자동 primary key 생성 기능은 반드시 TRANSACTION_SERIALIZABLE (isolation level)을 지원해야만 한다. 즉, Oracle, MS SQL Server 외에는 모두 이 isolation level이 지원되는지 확인 해야만 한다.

이제 데이터베이스에 설정하는 법을 보자.

### 11.6.3 Oracle DB 에서 Primary Key 생성 설정

Oracle 데이터베이스에서 자동 primary key 생성은 Oracle에서 사용하는 “sequence 객체”를 이용한다. Oracle 데이터베이스에서는 자동 primary key생성 기능을 설정하기 전에, sequence 객체가 생성되어 있어야 한다. Sequence 를 설정 하기 위해선 Oracle문서를 참조하기 바란다.

아래 와 같이 jeus-ejb-dd.xml을 설정 해야 한다.

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
    . . .
    <beanlist>
```

```
        . . .
        <jeus-bean>
            . . .
            <schema-info>
                . . .
                <data-source-name>MYORACLEDB</data-source-name>
                <auto-key-generator>
                    <generator-type>
                        Oracle
                    </generator-type>
                    <generator-name>
                        my_generator
                    </generator-name>
                    <key-cache-size>
                        20
                    </key-cache-size>
                </auto-key-generator>
            </schema-info>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

위 예제에서, primary key생성 타입을 “Oracle”로 했으며, generator이름을 “my_generator”로 하고 key cache size를 “20”으로 설정 했다.

**중요**: key cache 크기는 Oracle의 sequence 객체의 SEQUENCE INCREMENT 값과 일치해야 한다.

마지막으로 Oracle 데이터베이스는 <data-source-name>에서 선택된다 (여기서는 “MYORACLEDB”가 사용되었는데, JEUSMain.xml의 DB connection pool JNDI 이름이다. JEUS Server 안내서를 참조하라).

### 11.6.4 MS SQL Server 에서 자동 Primary Key 생성 설정

MS SQL Server 에서 자동primary key 생성은 간단하다. MS SQL Server는 자동으로 고유의 IDENTITY column에 primary key를 유지한다.

MS SQL Server 의 설정은 아래의 jeus-ejb-dd.xml을 참조한다.

*<<jeus-ejb-dd.xml>>*

```
<jeus-ejb-dd>
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <schema-info>
                . . .
                <data-source-name>MSSQLDB</data-source-name>
                <auto-key-generator>
                    <generator-type>
                        MSSQL
                    </generator-type>
                </auto-key-generator>
            </schema-info>
        </jeus-bean>
        . . .
    </beanlist>
</jeus-ejb-dd>
```

<generator-type> 에 “MSSQL”만 넣으면 된다. MS SQL Server는 <data-source-name>에서 선택된다.

### 11.6.5 Other DB 의 자동 Primary Key 생성

만약 Oracle 과 MS SQL Server 외에 다른 데이터베이스를 이용해서 자동 primary key를 생성한다면, 아래의 절차를 반드시 지켜야 한다.

데이터베이스에 테이블을 하나 만든다. 테이블은 하나의 컬럼과 하나의 열을 갖고 있어야 한다. 값은 “0”으로 설정되어야 한다. [표 12].

*표 12. primary key 생성테이블의 예제.*

Primary key generator table *PrimKeyTable*

| | *PrimKeyGeneratorColumn* |
|---|---|
| Row 1 | 0 |

위의 테이블은 두 개의 SQL로 이루어 진다.

```
CREATE table PrimKeyTable (PrimKeyGeneratorColumn int);
INSERT into PrimKeyTable VALUES (0);
```

위의 것을 수행한 후 jeus-ejb-dd.xml을 다음과 같이 수정한다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <schema-info>
                . . .
                <data-source-name>MYDB</data-source-name>
                <auto-key-generator>
                    <generator-type>
                        USER_KEY_table
                    </generator-type>
                    <generator-name>
                        PrimKeyTable
                    </generator-name>
                    <sequence-column>
                        PrimKeyGeneratorColumn
                    </sequence-column>
                    <key-cache-size>
                        20
                    </key-cache-size>
                </auto-key-generator>
            </schema-info>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

위 예제에서, 타입이 “USER_KEY_table”이고, 테이블 이름이 “PrimKeytable”이고 컬럼 이름이 “PrimKeyGeneratorColumn”이고 key cache 크기가 “20”으로 설정된 것이다.

데이터베이스를 사용하기 위해서 설정하는 <data-source-name> 태그의 값은 JEUSMain.xml의 DB connection pool의 export-name과 일치 해야 한다 ( JEUS Server 안내서를 참조하라).

### 11.6.6 결론

지금까지 JEUS에서 CMP 1.1 과 CMP 2.0에서 자동 primary key 생성 기능 설정과 사용법을 보았다. 이 기능을 사용하면 primary key에 신경 쓰지 않고 손쉽게 EJB를 개발 할 수 있다.

다음 장에서는 entity EJB 의 튜닝 포인트를 보겠다.

## 11.7 Entity EJB 튜닝

### 11.7.1 소개

이 절에서는 JEUS의 entity EJB 성능 튜닝에 대해서 자세히 알아보도록 하겠다.

다음과 같은 것들을 살펴보겠다.

- engine main mode 선택하기 (모든 entity bean에 적용).
- non-modifying method 사용하기 (모든 entity bean에 적용).
- entity cache 설정(모든 entity bean에 적용).
- engine sub-mode 선택 (CMP).
- DB fetch size 설정 (CMP).
- initial entity bean caching 설정 (CMP).
- DB vendor property 설정(CMP).
- instant EJB QL 사용 (CMP 2.0).

**참고**: 여기서 제공하는 튜닝 팁들은 6장과 10장의 “object management”에서 제공한 팁들과 함께 entity bean에 적용할 수 있다.

### 11.7.2 Engine main mode 선택과 클러스터링 적용 여부 (모든 Entity Bean 에 해당)

[표13] 에서는 <engine-type>을 선택할 때와 몇 개의 EJB engine에 EJB를 클러스터링할지를 선택할 때, 주요 판단 기준으로 삼을 수 있는 규칙을 정리했다.

*표 13. Entity bean 엔진 타입 선택과 클러스터링 사용 여부.*

| EJB engine 수 / CPU 수 / bean 당 요청 수 | **많은 EJB engine 다수의 CPU** | **적은 EJB engine 1개 이상의 CPU** | **EJB engine 한 개 CPU 한 개** |
|---|---|---|---|
| **자주 사용** | EJB를 클러스터링하고, SINGLE_OBJECT를 사용한다. | EJB를 클러스터링 하고, MULTIPLE_OBJECT를 사용한다. | 더 많은 EJB engine을 사용하도록 한다. 그렇지 않으면, 클러스터링 하지 말고, MULITPLE_OBJECT를 사용한다. |
| **보통으로 사용** | EJB를 클러스터링 하고, SINGLE_OBJECT를 사용한다. | EJB를 클러스터링 하고, MULTIPLE_OBJECT를 사용한다. | 클러스터링 하지 말고, MULITPLE_OBJECT를 사용한다. |
| **조금 사용** | EJB를 클러스터링 하지 말고, EXCLUSIVE_ACCESS를 사용한다. | EJB를 클러스터링 하지 말고, EXCLUSIVE_ACCESS를 사용한다. | EJB를 클러스터링 하지 말고, EXCLUSIVE_ACCESS를 사용한다. |

물론, entity bean의 database 데이터가 외부의 WAS가 아닌 컴포넌트에 의해 변경되는 환경에서는 SINGLE_OBJECT나 MULTIPLE_OBJECT를 사용해야만 한다.

그리고 fail over를 위해서는 bean을 클러스터링해야 한다.

SINGLE_OBJECT 와 MULTIPLE_OBJECT 모드 중 하나를 선택 할 때 마지막으로 고려해야 할 사항은 대상 bean(s)이 긴 트랜잭션 처리 시간을 가지느냐는 것이다 (즉, bean내부에서 트랜잭션이 시작하고 commit되어 끝날 때까지의 평균 시간). 그 시간이 아주 길면, MULTIPLE_OBJECT 모드를 선택하기를 권장하고, 이 모드에서 사용 가능한 동시성을 사용해야 한다. 그 시간이 반대로 짧으면 SINGLE_OBJECT모드 사용을 권장한다.

만약에 위의 "규칙"이 준수되면, EJB engine은 최적의 성능으로 ejbLoad() 호출을 할 수 있을 것이다.

**참고**: 위의 테이블에서 사용된 "자주 사용되는", "보통으로 사용되는", "조금 사용되는"의 의미를 명확하게 정의할 수 없다. 약간의 직관력과 많은 테스트가 주어진 환경에 최적의 설정을 찾는 방법일 것이다.

### 11.7.3 Non-modifying Method 등록(BMP & CMP 1.1 용)

Database에 표현된 상태 정보를 변경하지 않는 entity bean의 모든 업무 method들은 JEUS EJB 모듈 DD의 **<non-modify-method>** 태그에 등록되어야 한다. 이렇게 함으로써 EJB engine이 ejbStore() 를 호출하지 않아 효율적으로 운영될 수 있게 된다.

이 튜닝은 BMP와 CMP 1.1 bean에게만 적용할 수 있다.

### 11.7.4 Entity Cache 크기 설정 (모든 타입)

Entity cache는 passivate된 entity bean instance들의 저장소 역할을 한다. Instance들은 entity bean이 다시 activate가 되면 cache에서 나오게 된다.

**<entity-cache-size>** 크기를 크게(수 백 개의 instance가 수용되도록) 설정하면 passivate 된 bean들이 다시 activate될 때 새로운 entity bean instance들이 생성되지 않아도 되므로 성능이 좋아질 것이다.

그러나, 메모리 관리가 가장 중요한 사항이라면 이 값을 적게 주거나 아예 사용하지 않도록 한다(값이 "0"이면 entity cache를 사용하지 않는다).

### 11.7.5 Engine Sub-mode 선택 (CMP1.1/2.0 해당)

CMP bean을 사용할 때 **<subengine-type>** 을 선택하는 주요 기준은 다음과 같다.

- CMP bean이 읽기 작업을 쓰기 작업보다 더 많이 할 것이라 예상되면 ReadLocking 모드를 사용한다.

- 반면, CMP bean이 쓰기 작업을 읽기 작업보다 더 많이 할 것이라 예상되면 WriteLocking 또는 WriteLockingFind 모드를 사용한다.

이렇게 함으로써 EJB engine이 ejbLoad(), ejbFind() method를 최적화하여 사용할 수 있게 된다.

### 11.7.6 Fetch Size 설정 (CMP1.1/2.0 해당)

CMP bean의 **<fetch-size>** 태그를 시스템 메모리의 과용을 감수하더라도 높은 성능을 원한다면 높게 설정되어야 한다. 큰 값을 설정하면 매 호출마다 많은 양의 데이터를 읽어오기 때문에, 네트워크를 효과적으로 사용하게 된다. 그러나, 이렇게 할 때 EJB engine은 많은 양의 데이터를 캐싱해야 하고, 따라서 시스템 자원을 많이 소모하게 되어 각 호출에 대하여 대기 시간을 증가시키는 결과를 낳는다.

반대의 경우 만약 시스템 메모리의 절약이 높은 우선 순위를 가지고 느린 네트워크 속도를 감수할 수 있다면 이 값을 상대적으로 낮게 설정한다.

기본 fetch 값은 “10”이다. 대부분의 경우 “100”은 높은 값이고 “10”보다 낮은 값은 낮은 값으로 생각할 수 있다.

### 11.7.7 Initial Caching 설정(CMP1.1/2.0 해당)

높은 운영 성능을 위해 많은 시스템 메모리 사용을 감수할 수 있다면, CMP bean의 **<init-caching>** 값을 “true”로 설정한다. 이렇게 설정하면 EJB engine이 부팅할 때 모든 DB의 레코드들이 읽혀져 entity bean instance로 전달된다. 결과적으로 engine 부팅 시간이 오래 걸리기는 하겠지만, 처음의 접근 시간을 훨씬 줄일 수 있다.

반대의 경우 만약 시스템 메모리의 절약 또는 빠른 engine 부팅 시간이 우선이라면 이 값은 “false”로 설정되어야 한다.

만약 database 테이블이 EJB에 아주 많이 매핑되어 있는 경우(수백, 수천 레코드)에는 초기 caching 태그 사용이 자제되어야 할 것이다.

그러나, EJB가 *read-only*(ejbLoad()가 한 번만 호출되는 것을 암시)라면 이 옵션을 강력히 추천한다

### 11.7.8 DB Vendor 설정(CMP1.1/2.0 해당)

**<db-vendor>** 태그를 올바르게 설정한다. 모든 경우는 아니지만 어떤 상황에서는 Oracle과 같은 벤더의 제품을 위해서 EJB engine이 DB와의 연결을 최적화시킨다.

부록 E에는 이 태그에 사용할 수 있는 DBMS 벤더의 이름 값들이 나열되어 있다.

### 11.7.9 EJB QL 의 적절한 사용 (CMP 2.0 해당)

EJB QL이 약간은 비효율적이기 때문에 일반적으로 이의 사용은 권장되지 않는다. 그러므로 간단히 사용하고 말 경우에만 사용하도록 한다.

### 11.7.10 결론

최상의 성능과 효과적인 시스템 메모리 자원을 관리하기 위한 일곱 가지의 entity bean 튜닝 방법을 설명하였다.

더 자세한 내용은 부록 E를 참조하라.

## 11.8 결론

이것으로 JEUS에서의 entity EJB에 대한 설명을 매듭짓는다. Entity EJB의 JEUS 고유의 특성들을 살펴보았고, 어떻게 설정하는지, 어떻게 instant EJB QL을 프로그램하고, 성능을 위해 튜닝하는지도 살펴보았다. 마지막으로 간단하지만 완벽한 CMP 2.0 bean의 예를 제시함으로써 마무리를 지었다.

다음 장에서는 JEUS의 message-driven bean에 대하여 설명하겠다.

# 12 Message-driven Enterprise JavaBeans

## 12.1 소개

이 장에서는 JEUS에서의 message-driven bean에 대하여 간략하게 설명한다.

대부분의 MDB에 관련된 사항들은 EJB 2.1 스펙에 명시되어 있다. 그러므로, 본 매뉴얼에서는 이런 bean들을 JEUS EJB engine에서 사용할 때 유의해야 할 사항들만을 설명한다.

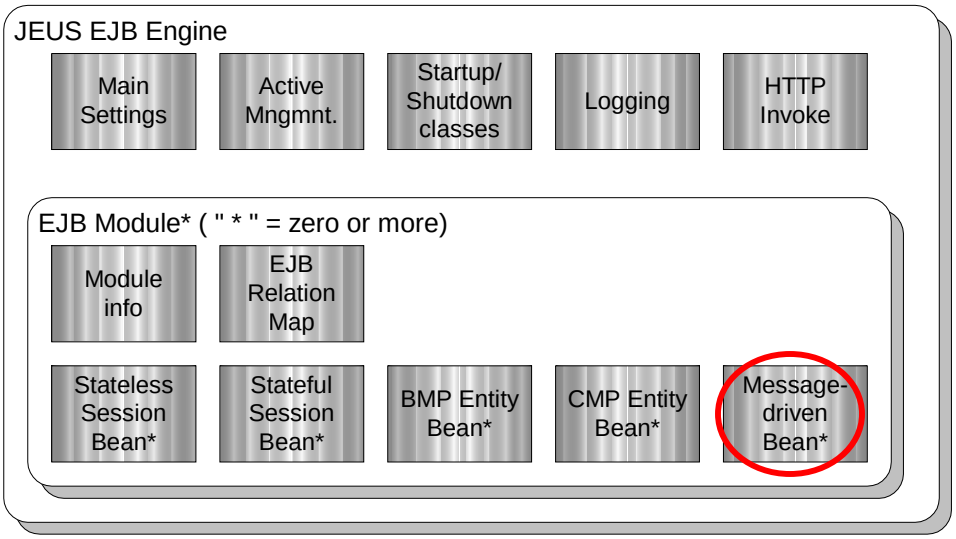


*그림 1. EJB engine 내의 MDB bean 들.*

## 12.2 MDB EJB 의 개요

### 12.2.1 소개

이 장에서는 MDB에 관련된 JEUS의 특정 항목들을 소개한다.

### 12.2.2 MDB 동시처리와 Thread Ticket Pool

EJB engine은 message-driven bean instance들을 동시에 여러 개 실행시킬 수 있으며, 또한 message stream을 처리할 수도 있다. EJB engine은 MDB 클래스

에 도착하는 메시지들의 순서를 보장하지 않으므로 받은 순서대로 메시지를 전달하지 못하고 임의로 메시지를 전달하게 된다. MDB도 처리 순서를 보장하지 않으므로, 설계 시에 이에 대한 고려를 충분히 해야 한다. 예를 들어 예약을 하는 메시지가 도착하기 전에 예약 취소하는 메시지가 먼저 올 수 있기 때문이다.

이런 동시처리는 Thread ticket pool이 MDB에서 instance를 가져와서 Request에 할당함으로써 처리된다. 이는 SL bean의 동작법과 동일하다(10장 참조).

MDB에서는 “object management”에 관련된 환경 설정이 없다.

### 12.2.3 MDB 만의 환경 설정

EJB engine이 MDB bean을 실행하기 위해서는 약간의 세팅만 해주면 된다. 다음 절에서 설명되겠지만, 이 설정은 MDB를 잘 알고 있는 이들에게는 따로 설명할 필요가 없다.

MDB만의 설정 외에 MDB는 JEUS의 다른 EJB와 공통되는 설정도 가지고 있다. 자세한 정보는 6장을 참조한다.

### 12.2.4 JNDI SPI 설정

기본적으로 JEUS MDB는 JEUS Naming Service로 JMS의 커넥션을 lookup해서 사용한다.

때로는 다른 Naming Service로부터 얻어야  될 때도 있다. 이 경우는 다른 JMS Service(IBM MQ나 SONIC MQ)를 이용할 때이다.

이 경우는 JMS service를 포함한 외부 naming service 를 사용하는 MDB를 설정할 수 있다.

다음 절에서 자세한 환경 설정을 볼 수 있다.

### 12.2.5 결론

이상으로 기본적인 JEUS MDB의 기본 사항들을 잠시 살펴보았다.

다음 절에서는 JEUS에서 MDB가 어떻게 설정되는지 설명한다.

# 12.3 MDB EJB 설정

### 12.3.1 소개

이 절에서는 다음 항목을 간략하게 설명한다.

- JEUS EJB DD의 기본 MDB 설정 (“일반적인EJB”장에서 설명한 설정들의 하위 항목)
- JEUS EJB DD 파일에 설정되어 있는 JEUS MDB의 JMS 전용 설정.

### 12.3.2 기본 환경 설정

MDB는 다른 EJB 종류와 몇 가지 기본 설정들을 공유한다. 각 MDB에 설정할 수 있는 것들은 EJB name, run-as identity, security interoperability, reference 와 thread ticket pool이 있다.

다음은 MDB의 기본 설정 예제이다(굵은 글씨로 표기된 항목 중 두 번째를 다음 절에서 설명한다).

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
        . . .
    <beanlist>
        . . .
        <jeus-bean>
            <ejb-name>order</ejb-name>
            <!--JMS settings goes here (see next sub-section-->
            <run-as-identity>
                . . .
            </run-as-identity>
            <security-interop>
                . . .
            </security-interop>
            <env>
                . . .
            </env>
            <ejb-ref>
                . . .
            </ejb-ref>
            <res-ref>
```

```
                . . .
            </res-ref>
            <res-env-ref>
                . . .
            </res-env-ref>
            <!-- No clustering of MDB -->
        </jeus-bean>
        . . .
    </beanlist>
       . . .
</jeus-ejb-dd>
```

### 12.3.3 JMS 설정

MDB를 위한 주요 MDB/JMS 특정 설정들 중 꼭 설정되어야 하는 것은 EJB 2.1의 방식을 따를 때에는 JMS connection factory의 export name이고 J2EE 1.4의 connector를 사용하여 메세징 시스템과 연결할 때에는 해당 resource adaptor의 설정이다.

- **<connection-factory-name>** JMS connection factory가 사용할 JNDI export-name
- **<mdb-resource-adaptor>** JEUS Connector를 통해 메세징 시스템과 연결될 경우 사용하는 resource-adaptor에 대한 설정

다음 설정들은 ejb-jar.xml에 설정되어 있는 것을 overriding하고자 하는 경우에 사용한다.

- **<destination>**
- **<max-messages>**
- **<ack-mode>** “auto-acknowledge” 또는 “dups-ok-acknowledge” 로 설정될 수 있다.
- **<durable>** “true” 또는 “false”로 설정될 수 있다.
- **<message-selector>**

위의 설정은 아래에 나열된 문서들에서 더 자세한 정보를 볼 수 있다.

- JEUS JMS 안내서

- JEUS Connector 안내서
- JMS 스펙
- EJB 스펙 (“MDB contract” 장)

다음은 XML 설정 문서의 일부이다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
        . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <connection-factory-name>
                ConnectFactory
            </connection-factory-name>
            <destination> … </destination>
            <max-message> … </max-message>
            <ack-mode> … </ack-mode>
            <durable>true</durable>
            <msg-selector> … </msg-selector>
            . . .
        </jeus-bean>
             . . .
    </beanlist>
      . . .
</jeus-ejb-dd>
```

### 12.3.4 JNDI SPI 환경 설정

MDB가 JMS service를 찾기 위해 naming service를 설정할 때 각각의 MDB에 <jndi-spi>를 설정 할 수 있다.

하위 절에 다음과 같은 태그가 있다.

- **MQ vendor** name: MDB가 커넥션을 맺을 MQ/JMS의 vender의 이름 (JNDI naming 서비스를 통해 설정). “IBMMQ” 와 “SONICMQ”가 공식적인 지원 vender이다.

- **Initial context factory**: Naming Server를 위한 JNDI initial context factory 클래스의 이름을 사용해서 JMS Service에 연결된다.
- **Provider URL**: 요청에 의해서 Naming Service에 접속할 때 사용하는 URL이다.

예)

```
<jeus-ejb-dd>
        . . .
    <beanlist>
        . . .
         <jeus-bean>
            . . .
            <jndi-spi>
                <mq-vendor>SONICMQ</mq-vendor>
                <initial-context-factory>
                    acme.jndi.ACMEContextFactory
                </initial-context-factory>
                <provider-url>
                    protocol://localhost:2345
                </provider-url>
            </jndi-spi>
            . . .
        </jeus-bean>
             . . .
    </beanlist>
      . . .
</jeus-ejb-dd>
```

### 12.3.5 Conclusion

이상으로 JEUS MDB 설정의 간단한 설명을 마무리 짓는다.

## 12.4 MDB EJB 튜닝

일반적인 MDB튜닝에 대해서는 6장을 참조한다.

EJB와 JMS 스펙 및 JEUS JMS 안내서도 참조한다.

## 12.5 결론

이것으로 JEUS MDB에 대한 간단한 설명을 마친다. 다른 참고 문서들도 살펴보고 앞에서 명시한 문서들도 참고하여 JEUS MDB에 대한 이해를 더 해보도록 한다.

다음 장에서는 JEUS의 EJB Timer Service에 대해 설명한다.

# 13 EJB Timer Service

## 13.1 소개

EJB Timer Service는 EJB가 특정한 시간 또는 주기적으로 timer callback 을 받을 수 있도록 하는 서비스이다. 기본적인 사용 방법은 EJB 2.1 spec에 자세히 나와 있으므로 여기서는 JEUS EJB에서 제공하는 Timer 서비스와 이를 사용하기 위한 설정에 대해서 설명하겠다.

## 13.2 Timer Service 의 설정

### 13.2.1 소개

JEUS EJB Timer 서비스는 기본적으로 spec을 따르지만 persistence하게 Timer를 관리하는 기능은 성능과 사용자의 필요에 따라 선택적으로 사용할 수 있도록 구현되었다. 이를 위해 설정 항목이 두가지 종류로 나누어져 있다.

- Timer service를 사용하는 모든 bean에 적용되는 공통적인 성질과 Persistence한 Timer 서비스를 가능하게 하는 설정으로 EJBMain.xml에 설정한다.
- 각 bean이 deploy되거나 undeploy될 때 Persistent Timer들을 어떻게 관리하는가에 대한 설정으로 jeus-ejb-dd.xml에서 변경한다.

### 13.2.2 Persistent timer service 설정 (EJBMain.xml)

Persistent timer service는 <ejb-engine>의 하위 항목인 <timer-service>를 사용하여 설정된다. 다음은 한 예이다.

*<<EJBMain.xml>>*

```
<ejb-engine>
    . . .
    <timer-service>
        <min-delivery-interval>7000</min-delivery-interval>
```

```
        <max-redelivery-count>1</max-redelivery-count>
        <redelivery-interval>4000</redelivery-interval>
        <thread-pool>
            <min></min>
            <max></max>
            <period></period>
        </thread-pool>
        <durable-setting>
            <db-vendor>oracle</db-vendor>
            <data-source-name>jdbc/DB1</data-source-name>
            <engine-type>SINGLE_OBJECT</engine-type>
            <table-name>jeusTimerTable</table-name>
        </durable-setting>
    </timer-service>
    . . .
</ejb-engine>
```

<timer-service> 항목의 하위 항목 중 timer service의 공통적인 설정에 대한 설명은 다음과 같다.

- <min-delivery-interval> 이 값은 특정 timer가 생성된 후에 timer event가 발생하기까지의 최소 시간이다. 이 값이 작을수록 EJB 엔진에 부하를 줄 수 있다. 단위는 ms이며 기본값은 7000이다.

- <max-redelivery-count> 이 값은 timer callback 함수에서 exception이 발생하거나 transaction이 rollback되어 재전송이 발생하는 상황에서 최대 재전송 횟수를 나타낸다. 기본값은 1이다.

- <redelivery-interval> 이 값은 재전송을 해야 할 상황이 발생했을 때부터 재전송하기까지의 시간이다. 단위는 ms이고 기본값은 5000이다.

<thread-pool> 이 값은 Timer service가 timeout() method를 실행할때 사용하는 thread pool에 대한 설정이다. 하위 항복에 대한 자세한 설명은 다음과 같다.

- <min> 이 값은 pooling되는 thread의 최소값이다.

- <max> 이 값은 pooling되는 thread의 최대값이다.

- <period> 이 값은 pooling되는 thread를 정리하는 시간이다.

<durable-setting> 항목은 Persistent timer service를 사용한다면 꼭 있어야 할 항목이다. Persistent timer는 DB를 사용하기 위해 내부적으로 CMP bean을 사용하므로 이 CMP bean이 <durable-setting>의 하위 항목들을 통해 설정된다. 따라서 각 항목은 CMP bean의 schema 설정과 같다. 자세한 설명은 다음과 같다.

- <db-vendor> 이 값은 Timer CMP bean이 사용하는 DB의 vendor를 지정하는 것이다.
- <data-source-name> 이 값은 Timer CMP bean이 사용하는 datasource resource의 이름을 지정한다.
- <engine-type> 이 값은 Timer CMP bean이 사용하는 엔진 type을 지정한다. 자세한 것은 CMP 설정을 참고하기 바란다.
- <table-name> 이 값은 Timer CMP bean이 사용할 DB table 이름을 지정한다. 기본값은 Jeus_Timer이다.

### 13.2.3 Persistent timer 의 처리 (jeus-ejb-dd.xml)

Persistent timer service를 사용하도록 EJB 엔진에 설정되어 있을 때, 사용자는 각 EJB bean 별로 Persistent timer service를 사용할지, deploy하기 전에 DB에 남아있는 persistent timer를 사용할 지 제거할 지 등의 동작을 설정할 수 있다. 이는 jeus-ejb-dd.xml을 통해서 이루어진다.

다음은 이러한 설정을 한 예이다.

<<*jeus-ejb-dd.xml*>>

```
<jeus-ejb-dd>
        . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <durable-timer-service>
                  <enable-durable-timers>
                        true
                  </enable-durable-timers>
                  <ignore-durable-timers-at-deploy>
                        false
                  </ignore-durable-timers-at-deploy>
```

```
                    <delete-durable-timers-at-deploy>
                          false
                    </delete-durable-timers-at-deploy>
                    <delete-durable-timers-at-undeploy>
                          false
                    </delete-durable-timers-at-undeploy>
               </durable-timer-service>
               . . .
          </jeus-bean>
               . . .
     </beanlist>
       . . .
</jeus-ejb-dd>
```

다음은 하위 항목들의 설명이다.

- <enable-durable-timers> 이 값은 이 bean이 사용하는 timer를 persistent하게 만들지의 여부를 결정한다.
- <ignore-durable-timers-at-deploy> 이 값은 이 bean이 deploy될 때 DB의 Timer table에 존재하는 이 bean에 대한 timer들을 살리지 않고 무시하고 싶을 때 사용한다. 기본값은 false이다.
- <delete-durable-timers-at-deploy> 이 값은 이 bean이 deploy될 때 DB의 Timer table에 존재하는 이 bean에 대한 timer들을 deploy후에 지워버리고 싶을 때 사용한다. <ignore-durable-timers-at-deploy>과 같이 true로 설정되어야 의미가 있다. 기본값은 false이다.
- <delete-durable-timers-at-undeploy> 이 값은 이 bean이 undeploy될 때 현재 존재하는 timer들을 DB의 Timer table에서 제거하고싶을 때 사용한다. 제거하지 않으면 다음 deploy때 DB에 저장된 timer들이 다시 생성될수 있다. 기본 값은 true이다.

# 13.3Timer Service 사용시 주의사항

### 13.3.1 Persistent timer 와 JDBC connection

Persistent Timer는 DB에 저장되고 이 Timer들을 관리하는 CMP bean은 EJBMain.xml에 설정된 datasource resource를 사용한다. 따라서 Persistent timer를 사용하는 bean이 포함된 transaction은 Timer CMP bean이 사용하는

datasource도 관리하게 된다. 따라서 이를 고려해서 datasource를 LocalXAResource로 사용할지 XAResource로 사용할지를 결정해야 한다.

Datasource에 대한 자세한 설명은 JEUS Server 안내서를 참고하기 바란다.

## 13.4 결론

이 장에서는 JEUS의 EJB Timer service의 설정과 사용시의 주의사항에 대해 살펴보았다.

다음 장은 EJB를 사용하는 클라이언트 프로그램을 작성하는데 필요한 여러 가지 정보를 소개한다.

# 14 EJB 클라이언트

## 14.1 소개

EJB 클라이언트는 EJB engine에서 실행되고 있는 enterprise bean의 업무 method를 호출하는 모든 종류의 어플리케이션을 일컫는다. 그러므로 EJB클라이언트는 Servlet, Applets, 다른 EJB, 단순 Java 프로그램 등이 이에 속한다.

이 장에서는 EJB클라이언트를 작성하고 실행시키는 데 기본적으로 필요한 정보를 제공하며, 여기서는 일반적인 Java 프로그램을 살펴보기로 하다.

다른 종류의 EJB클라이언트에 대해서는 JEUS Client Application 안내서를 참조한다.

## 14.2 EJB 클라이언트 개요

### 14.2.1 소개

이 절에서는 JEUS EJB클라이언트의 설정과 실행에 대한 기본적인 사항들을 알아본다.

### 14.2.2 JEUS 에서 클라이언트 어플리케이션

일반적인 클라이언트 어플리케이션에 대해서는 JEUS Client Application 안내서를 참조한다. 그 안내서는 모든 클라이언트 어플리케이션(일반 Java 어플리케이션, CAS 클라이언트, Applet 클라이언트, JNLP) 들에 대해서 설명한다.

이 절에서는 일반적인 Java 프로그램에 대한 사항들만 설명한다

### 14.2.3 JEUS 에서 EJB 클라이언트 어플리케이션

JEUS의 EJB 클라이언트 어플리케이션을 프로그래밍할 때에는 주의해야 할 사항이 거의 없다고 해도 과언이 아니다. 표준 J2EE API를 그대로 적용하면 되기 때문이다.

그러나, JEUS EJB 클라이언트를 실행시킬 때에는 실제로 두 가지 설정이 필요하다.

- 정확한 JNDI InitialContext instance가 설정되기 위해서는 환경 변수들이 제공되어야 한다.

- 두 개의 중요한 JAR 파일을 포함하도록 EJB 클라이언트를 실행시킬 JVM의 클래스경로가 설정되어야 한다. 두 개의 파일이란, EJB stub을 포함하는 클라이언트 JAR와 JEUS의 J2EE 클래스들을 포함하는 clientcontainer.jar 이다.

### 14.2.4 JNDI InitialContext

EJB를 사용할 때마다 필요한 중요 컴포넌트는 InitialContext이다. InitialContext는 JNDI namespace의 모든 명명된 객체를 찾을 때 사용하는 객체이다. EJB에서는 이 객체를 사용하여 EJB home interface를 찾고 그 결과로 EJB engine의 실제 bean instance의 레퍼런스를 찾게 된다.

JEUS의 InitialContext instance를 가져오기 위해서는 특정한 환경 값들을 제공해야 한다. 이 환경 변수들은 다음 절에서 설명한다.

### 14.2.5 EJB 클라이언트 실행과 CLASSPATH

JEUS에서 EJB 클라이언트를 실행시키려면 먼저 InitialContext에게 속성을 전달해야 하고, 다음으로는 JVM 운영 환경의 정확한 클래스경로를 잡아줘야 한다.

이 장의 마지막 부분에서는 JVM을 호출할 때 어떻게 클래스경로를 결정하고 설정하는지에 대해서 설명한다.

### 14.2.6 결론

지금까지 JEUS의 EJB 클라이언트 어플리케이션에 필요한 기본 속성들에 대해 알아 보았다. EJB 클라이언트를 실행시킬 때 먼저 JNDI환경을 설정해야 하고 클래스 경로를 바로 잡아야 한다는 것을 알았다.

다음 절에서는 EJB 클라이언트의 작성 방법에 대해 자세히 알아본다.

## 14.3 EJB 접근을 위한 클라이언트 프로그래밍

다음 샘플 코드는 “HelloApp”라는 export name을 가지는 EJB를 찾고 “sayHello()”라는 method를 호출하는 일반적인 클라이언트 어플리케이션의 구현을 보여준다.

*<<HelloClient.java>>*

```
package hello;

import java.rmi.*;
import javax.ejb.*;
import javax.naming.*;

public class HelloClient {
    HelloHome home = null;
    Hello obj = null;
    private void run() {
        try {

            //Receive JEUS initial context
            InitialContext ctx = new InitialContext();
            //Receive EJB home object
            object ref=ctx.lookup("HelloApp");
            home = HelloHome)PortableRemoteObject.narrow
                    (ref,HelloHome.class);
            //Load/create EJB object
            obj = (Hello)home.create();
            //Business Method invocation
            String s = obj.sayHello();

        } catch (Exception e) {
            // Exception handling.
        }
    }

    public static void main(String args[]) {
        HelloClient hclient = new HelloClient();
        hclient.run();
    }
}
```

위 예는 EJB 클라이언트 프로그래밍에 대한 새로운 것들을 발견할 수는 없지만, JEUS 고유의 설정과 클라이언트의 Deployer가 알아야 할 것을 보여주기 위해 제시하였다.

위의 예에서 InitialContext가 얻어지는 굵은 글자체로 된 부분을 주시하라. 그 부분이 제대로 동작하기 위해서는 InitialContext의 속성을 다음 절에 있는 것처럼 설정해줘야 한다.

# 14.4 Initial Context 설정

## 14.4.1 소개

InitialContext를 얻기 전에 다섯 개의 naming context 속성 중 적어도 한 개는 지정해 줘야 클라이언트 어플리케이션이 JEUS naming server(JNS)와 통신할 수 있게 된다.

## 14.4.2 다섯 개의 JNDI 속성들

아래에는 JEUS의 InitialContext 객체를 얻기 전에 설정해야 할 다섯 개의 지원되는 naming context 환경 속성들을 열거하고 있다.

**`INITIAL_CONTEXT_FACTORY`** `(required)`

이 naming context 속성은 “jeus.jndi.JEUSContextFactory”의 값을 가져야 한다. 이 값은 naming context를 만드는데 사용하는 factory의 클래스 이름이다.

**`URL_PKG_PREFIXES`**

이 속성은 jeus.jndi.jns.url로 설정되어 JEUS InitialContext의 URL scheme을 이용하여 객체들을 찾는다.

**`PROVIDER_URL`**

이 속성은 JEUS naming server(JNS Server) IP 주소를 설정한다. 기본값은 127.0.0.1(“localhost”)이다.

**`SECURITY_PRINCIPAL`**

이 속성은 JEUS naming service의 인증 과정 중에 사용되는 사용자 정보를 설정하기 위해 사용한다. 실행 쓰래드에 사용자 정보가 등록되어 있지 않으면 기본으로 guest로 인증된다. 만약에 나중에 지정이 되면 JEUS security realm에 정의된 username을 가지는 사용자 정보가 사용된다.

**`SECURITY_CREDENTIALS`**

인증과정 중에 사용될 사용자의 패스워드를 설정한다. 인증이 실패하게 되면 그 전의 사용자 정보가 그 전처럼 유지된다. 성공하면 새로운 사용자 정보가 사용된다.

위에서 제시한 기본 속성들 외에 추가적인 특성들이 더 설정될 수 있다. 그 추가 속성들은 JEUS Server 안내서의 "JNDI naming 장"에 기술되어 있다.

다음 두 절은 위의 속성들을 가지고 설정할 수 있는 두 가지 방법들에 대하여 설명한다. 한 방법은 JVM의 "-D" 속성을 사용하는 것과 다른 방법은 Hashtable을 사용하는 것이 있다.

### 14.4.3 JVM 속성을 이용한 Naming 속성 값 설정

위의 절에서 설명한 속성들은 JVM interpreter의 "-D" 스위치를 사용하여 설정할 수 있다. 이 스위치들은 EJB 클라이언트 어플리케이션을 시작하기 위해 사용하는 "java"명령어에 추가하여야 한다.

아래에는 각 naming 속성이 "-D" 스위치로 설정되는 것을 보여준다.

**INITIAL_CONTEXT_FACTORY** (required)

```
-Djava.naming.factory.initial=jeus.jndi.JEUSContextFactory
```

**URL_PKG_PREFIXES**

```
-Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
```

**PROVIDER_URL**

```
-Djava.naming.provider.url=<host_address>
```

**SECURITY_PRINCIPAL**

```
-Djava.naming.security.principal=<username>
```

**SECURITY_CREDENTIALS**

```
-Djava.naming.security.credentials=<password>
```

아래의 예제에서는 클라이언트 호출 시에 "-D" 옵션을 사용하여 EJB 클라이언트의 naming context를 설정하는 것을 보여준다. "peter"라는 사용자이름과 "golf42"라는 패스워드가 192.168.10.10 주소로 등록된 JEUS naming server를 이용하여 EJB stub을 찾는 EJB 클라이언트에 어떻게 사용되는가를 보여준다.

```
C:\> java –cp <classpath>
-Djava.naming.factory.initial=jeus.jndi.JEUSContextFactory
-Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
-Djava.naming.provider.url=192.168.10.10
-Djava.naming.security.principal=peter
```

```
-Djava.naming.security.credentials=golf42
<EJB application class name>
```

(위의 내용은 전체가 한 라인으로 구성된다.)

### 14.4.4 Hashtable 를 이용한 Naming 속성 설정

Naming context 속성을 설정하는 다른 방법은 Hashtable를 사용하여 클라이언트 코드 내의 naming attribute data를 직접 Hashtable에 넣는 것이다. 이 방법은 값들을 hard-coding하기 때문에 권장되지는 않는다.

다음 코드는 이를 어떻게 구현하는지 보여준다.

...

```
Context ctx = null;
Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY,
“jeus.jndi.JEUSContextFactory”);
ht.put(Context.URL_PKG_PREFIXES, “jeus.jndi.jns.url”);
ht.put(Context.PROVIDER_URL, “192.168.10.10”);
ht.put(Context.SECURITY_PRINCIPAL, “peter”);
ht.put(Context.SECURITY_CREDENTIALS, “golf42”);
try {
ctx = new InitialContext(ht);
. . .
```

위의 예에서, Hashtable이 InitialContext의 constructor 파라미터로 전달되는 것에 주목한다.

### 14.4.5 결론

지금까지 JNDI InitialContext 객체가 JVM의 “-D” 속성(권장사항)이나 EJB 클라이언트 코드의 Hashtable를 통하여 설정되는 것을 살펴보았다.

다음 절에서는 EJB 클라이언트를 어떻게 호출하는지 살펴본다.

## 14.5 클라이언트 어플리케이션의 실행

EJB 클라이언트 어플리케이션을 실행시킬 때 어떤 클래스들은 클래스 경로를 통하여 접근 가능해야 한다. 이 클래스들은 다음과 같은 두 개의 JAR파일들에 모두 들어있다.

- **JEUS_HOME\client\<module name>.jar** 파일은 EJB 모듈 내의 EJB 들을 접근하고 사용할 때 사용되는 모든 클래스들이 포함되어 있다. 이 클래스로는 home과 remote stub 등이 있다.

- **CLIENT_HOME\clientcontainer.jar** 파일은 JEUS와 상호 작용하기 위한 많은 수의 런타임 클래스들이 들어 있다 (CLIENT_HOME은 기본으로 "JEUS_HOME\webhome\client_home"으로 설정되어 있다).

그러므로 EJB 클라이언트 어플리케이션을 실행시키려면 위의 두 JAR파일들은 클라이언트 JVM에 의해 접근 가능해야 한다. 이 요구 사항은 물론 naming context 속성들을 설정하는 자리에 추가 해줘야 한다.

아래에 제공된 예에서는 "AccountClient"라는 main 메소드를 가진 EJB 클라이언트 어플리케이션이 "account"라고 불리는 EJB 모듈을 사용한다

```
C:\> java –classpath %JEUS_HOME%\client\account.jar;
%CLIENT_HOME%\clientcontainer.jar;c:\mylib\acclient.jar
-Djava.naming.factory.initial=jeus.jndi.JEUSContextFactory
-Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
-Djava.naming.provider.url=192.168.10.10
-Djava.naming.security.principal=peter
-Djava.naming.security.credentials=golf42
AccountClient
```

위의 텍스트는 한 줄로 만들어져 있고, 스크립트 파일로 만들어 사용할 수도 있다.

위의 예는 시스템 환경 변수들인 "JEUS_HOME"과 "CLIENT_HOME"이 적절한 위치에 설정되어 있음을 가정한다. 그리고, 클래스경로에 클라이언트 어플리케이션 자체가 속한 클래스들을 포함해야 한다.

## 14.6 결론

이 장에서는 JEUS의 EJB 클라이언트에 대해 간략하게 살펴보았다.

이 문서의 나머지 부분들은 JEUS EJB의 부록들로 구성한다.

# 15 결론

이것으로 JEUS의 EJB 안내서를 마무리한다.

본 안내서에서는 다음과 같은 주제들에 대해 설명하였다.

- JEUS EJB 의 개요
- JEUS EJB engine의 설정, 운영, 튜닝
- JEUS EJB 모듈의 조립, Deploy, 사용
- 일반적인 EJB: JEUS EJB DD와 EJB thread ticket pool과 같은 JEUS 전용의 설정들
- JEUS EJB의 보안
- 다른 J2EE EJB engine과 상호 보안
- JEUS EJB의 클러스터링
- Session bean에 해당하는 항목들 (stateful와 stateless)
- Entity bean에 해당하는 항목들 (BMP, CMP 1.1과 CMP 2.0)
- Message-driven bean에 해당하는 항목들 (MDB)
- Timer Service 설정과 사용
- EJB Client Application과 JEUS 내의 구성 설정에 대한 대략적인 설명

JEUS EJB에 대한 추가적인 설명이 필요하다면 JEUS Server 안내서와 JEUS Client Application 안내서 그리고 JEUS를 설치할 때 생기는 JEUS_HOME\samples 디렉토리 내의 EJB 예들을 참조한다.

다음은 JEUS EJB에 대한 참고 문서인 부록들로 구성되어 있다.

# A ejbadmin 콘솔 툴 레퍼런스

## A.1 소개

이 문서는 EJB 엔진에서 실행되고 있는 EJB 모듈을 컨트롤하는데 사용되는 ejbadmin 콘솔 툴의 목적과 수행을 설명한다.

ejbadmin 스크립트는 JEUS_HOME\bin\에서 찾을 수 있다.

## A.2 목적

ejbadmin 콘솔 툴의 목적은, 사용자에게 Deploy된 EJB모듈을 간단하고 빠르게 접근하고 컨트롤 할 수 있는 방법을 제시하는 것이다. ejbadmin 툴은 그래픽 디스플레이를 사용할 수 없는 환경에 적합하다.

ejbadmin 툴은 이미 조립된 EJB모듈들을 Deploy하고 컨트롤링하는 기본적인 작업들을 수행하도록 만들어져 있다. 그러나 이 툴을 사용해서 EJB모듈을 조립하거나 EJB엔진이나 Deploy된 모듈들의 상세한 정보를 얻거나 할 수는 없다. 조립을 위해서는 JEUS Builder를, 상세한 모니터링을 위해서는 웹 관리자를 사용해야한다.

**참고**: ejbadmin 콘솔 툴은 EJB 모듈들을 컨트롤하도록 되어 있다. 각각의 EJB 또는 EJB 엔진 자체를 조작할 수는 없다. 각각의 EJB들을 대상으로 작업하려면 웹 관리자를 사용하고, EJB 엔진을 대상으로 작업하려면 jeusadmin 툴이나 웹 관리자를 사용하여야 한다( JEUS 웹 관리자 안내서에서 자세히 설명이 되어 있다).

## A.3 호출

ejbadmin을 실행 할 때 다음과 같은 문법을 적용한다.

```
ejbadmin [-verbose] container_name
        [-u<username> -p<password>] [<command>]
```

위의 옵션들은 다음과 같은 의미을 가질 수 있다.

`-verbose`

콘솔 윈도우에 더 많은 정보를 보이도록 한다.

`container_name`

이 필수 파라미터는 연결하려는 EJB엔진이 실행되고 있는 container의 이름을 명시한다. container의 이름을 짓는 규칙은 일반적으로 <nodename>_ <containername>이고 <nodename>과 <containername>은 실제 이름들로 바뀌어야 한다.

`-U<username> -P<password>`

ejbadmin을 interactive 모드로 실행하고 싶지 않거나 로그인 과정을 거치지 않으려면 유효한 사용자이름과 패스워드를 –U와 –P를 사용하여 부여해 줘야 한다.

`<command>`

여기에 명령어를 넣으면 ejbadmin 명령프롬프트에서 입력하지 않고도 바로 명령을 실행 시킬 수 있다. 이 옵션은 –U와 –P 를 추가로 설정해 줘야한다. 이 옵션이 사용되면 해당 명령이 수행되고 바로 exit한다 (예, non-interactive 모드).

예)

```
johan> ejbadmin –verbose johan_container1
```

“johan” 머신에서 실행된 이 명령어는 “johan_container1”에 ejbadmin을 연결시킨다. “container1”은 JEUSMain.xml에 지정된 엔진 컨테이너의 이름이다 ( JEUS Server 안내서 참조 ). 이 경우 ejbadmin은 이 엔진 컨테이너에서 실행중인 EJB 엔진에 대한 관리를 할 수 있다.

ejbadmin 이 EJB엔진과 연결되면 사용자는 사용자 이름과 패스워드를 넣어야 한다. ejbadmin을 사용하기 위해서는 사용자가 반드시 administrator의 권한을 가지고 있어야 한다.

사용자가 성공적으로 인증되면 다음과 같은 명령창이 나타난다. 이는 ejbadmin이 다음 절에 나오는 모든 명령을 수행할 준비가 되었다는 것을 나타낸다.

```
johan_container1>
```

## A.4 기본 ejbadmin 명령어

ejbadmin 명령프롬프트에 입력할 다음 명령어들은 ejbadmin 자체를 컨트롤 한다.

`exit`

ejbadmin을 끝내고 EJB 엔진과의 연결을 끊는다.

`help`

사용 가능한 모든 명령어를 보여준다.

## A.5 EJB 모듈을 컨트롤하는 명령어

다음의 명령어들은 EJB엔진에 연결된 EJB모듈들을 ejbadmin 프롬프트에서 실행시킬 수 있다.

**`distribute/deploy [-f] [-per] [-isolated] [-absolute-path] [-auto] [-localAppDD] [-writeAppDD] [-class-ftp-unit [|JAR] [|CLASS]] [-keep] modulename`**

연결된 container의 EJB 엔진에 해당 EJB모듈을 Deploy 한다. 이 명령이 실행되려면 그 모듈의 archive file이나 directory가 해당 container의 application directory에 존재해야 한다 (-absolute-path 옵션을 사용할 경우는 제외). modulename은 그 archive file의 확장자를 제외한 file name이거나 directory 이름이 된다.

**Deploy** 명령은 deploy를 하고 EJB 모듈을 곧바로 시작하는데 비해 distribute는 deploy만 하고 EJB 모듈의 서비스를 시작하지는 않는다. 이 경우에는 시작하고자 할 때 resume이나 start 명령을 사용해야 한다.

**-f**옵션은 툴이 “fast deploy”를 하도록 명령한다. 이 때는 home과 remote stub과 skeleton클래스들이 재생성되지 않는다. 처음 Deploy를 하거나 모듈의 클래스 파일들이 변경되었을 때에는 이 옵션을 사용해서는 안 된다. 그러나, appcompiler툴을 사용하여 필요한 클래스들을 생성했을 때에는 사용할 수 있다( JEUS 서버 안내서 참조).

**-per** 옵션은 모듈이 JEUSMain.xml 의 <application> 태그로 추가되서,

엔진이 부팅할 때 그 모듈이 로딩되도록 한다.

**-isolated** 옵션은 deploy 하는 EJB 모듈의 classloader 를 다른 EJB 들과 분리시킬 때 사용한다.

**-absolute-path** 옵션은 deploy 를 할 archive file 이나 directory 가 존재하는 절대경로를 설정한다. 따라서 container 의 application directory 에 application 이 존재하지 않아도 이 옵션을 사용해서 deploy 할 수 있다. 주의할 것은 여기의 path 는 ejbadmin 이 실행되는 node 의 path 가 아니라 container 가 실행중인 node 에서의 path 라는 것이다.

**-auto** 옵션은 deploy 하는 module 이 auto deploy 옵션을 사용하도록 한다. Auto deploy 에 대해서는 JEUS Server 안내서를 참고하기 바란다.

**-localAppDD** 옵션은 deploy 를 할 때 application archive 에 존재하는 jeus-application-dd.xml 을 사용하게 하는 옵션이다. jeus-application-dd.xml 에 대해서는 JEUS Server 안내서를 참고하기 바란다.

**-writeAppDD** 옵션은 deploy 할 때 사용하는 jeus-application-dd.xml 를 container 의 해당 application archive 내에 복사해 두도록 하는 옵션이다. 이 옵션을 사용하고 나면 deploy 할 때 사용했던 –absolute-path 나 –auto 옵션등을 다음 deploy 할때에 –localAppDD 를 통해서 그대로 사용할 수 있게 된다.

**-class-ftp-unit** 은 deploy 된 EJB 모듈의 stub class 를 client 에서 class-ftp 를 통해 가져올 때 JAR file 단위로 보낼지 class file 단위로 보낼지를 결정하는 옵션이다. 기본값은 JAR file 단위이다.

예)

```
johan_container1> deploy –f mymodule
```

위의 명령은 “mymodule”이라는 EJB 모듈을 “johan_ejb_engine1” EJB 엔진에 Deploy하면서 “fast deploy”를 하도록 설정하고 있다.

**-keep**은 deploy과정중 Engine으로부터 생성되는 home과 remote의 stub과 skeleton, 그리고 추가적으로 생성되는 클래스들에 대한 source file을 삭제하지 않게 하는 옵션이다.

**`reload modulename`**

지정한 모듈을 다시 로딩한다. 이 옵션은 “deploy” 명령과 같은 동작을 하지만, 현재 동작중인 EJB모듈을 Undeploy 하고 Deploy 하도록 한다. 다양한 option을 사용하고 싶다면 undeploy 명령과 deploy 명령을 이용하도록 한다.

**`suspend/stop [-a] modulename`**

지정한 모듈을 멈추고, 다른 컴포넌트들은 이 EJB들을 접근할 수 없게 된다.

-a 옵션은 modulename을 지정하지 않은 경우 모든 module을 대상으로 명령을 수행할 때 사용한다.

**`resume/start [-a] modulename`**

지정한 모듈을 다시 활성화하고, 포함된 모든 EJB들이 다시 서비스 가능한 상태로 되돌린다. 이 명령어는 “suspend”명령어의 수행 후에 실행되어야 한다.

-a 옵션은 modulename을 지정하지 않은 경우 모든 module을 대상으로 명령을 수행할 때 사용한다.

**`undeploy [-per] [-a] modulename`**

지정한 모듈을 EJB 엔진에서 Undeploy한다. 이 명령어가 수행될 때 어떤 파일들도 제거되지않는다.

-per 옵션이 사용되면 JEUSMain.xml의 해당하는 <application> 태그가 제거된다. -a 옵션은 modulename을 지정하지 않은 경우 모든 module을 대상으로 명령을 수행할 때 사용한다.

## A.6 EJB 를 모니터링하는 명령어

Deploy된 모듈과 bean 들의 정보를 얻는데 두 기본 명령어가 사용된다.

`modulelist [modulename]`

modulename이 생략되면 Deploy된 모든 EJB모듈의 이름을 출력한다. modulename을 주면 주어진 modulename에 해당하는 EJB 모듈의 이름이 출력된다. modulename에는 regular expression이 지원된다. Regular expression에 대해서는 J2SE 1.4 API의 java.util.regex.Pattern 클래스의 javadoc을 참고하기 바란다.

`beanlist modulename`

지정한 모듈에 포함된 모든 bean의 이름을 출력한다. 이 modulename에도 regular expression이 지원된다.

`moduleinfo <modulename>`

모듈에 대한 모니터링을 해준다.

`beaninfo <bean name>`

빈에 대한 모니터링을 해준다.

# B JEUS EJB 환경 변수

## B.1 소개

[표9]에 JEUS EJB와 관련있는 모든 Java 시스템 프로퍼티를 정리했다.

이 속성들은 ‘jeus’ 스크립트 파일에 설정되어 있지만 ‘ejbadmin’과 ‘appcompiler’ 스크립트들에도 적용할 수 있다. 그리고 이 속성들은 JEUSMain.xml (JEUS Server 안내서 참조)의 <engine-container> 태그 아래 <command-option> 태그에서도 설정할 수 있다.

**중요**: 이 속성들은 ‘jeus’ 스크립트 내부에만 적용시키기를 권장한다. 경험이 많은 JEUS 엔지니어들만이 엔진 컨테이너와 EJB엔진의 속성들을 수정하기 바란다.

이 환경 변수들은 “-D” 옵션과 같이 사용하고 환경 변수 이름 뒤에 ‘=’ 문자 다음에 값을 입력해야 한다.

디폴트로 ‘jeus’ 스크립트에서는 JVM 속성 중 일부는 시스템 환경 변수 (“jeus.home” 속성이 JEUS_HOME 으로부터 할당받은 것처럼)에서 값을 할당받는다.

## B.2 환경 변수 레퍼런스

*표 14. EJB Java 시스템 속성*

| JEUS Manager JVM 변수들 | 의미 | 예 / 값 |
|---|---|---|
| `jeus.home` | JEUS가 설치된 홈 디렉토리. | c:\jeus50 |
| `jeus.ejb.operation Timeout` | EJB 엔진의 operation timeout을 선언한다. 한 EJB가 그 EJB 엔진의 JVM 외부의 method를 호 | 60000(milliseconds) |

| JEUS Manager JVM 변수들 | 의미 | 예 / 값 |
|---|---|---|
| | 출할 때 bean내부의 코드에서 이 시간만큼이 허용된다. | |
| jeus.ejb.checkTable | 'false'로 설정되어 있으면 EJB 엔진은 테이블 이름과 필드가 entity EJB의DD에 선언된 스키마 정보와 같은지 확인하지 않는다. 이 설정은 성능을 향상시킬 수 있다. 'true'로 설정되면 확인작업을 수행한다 (만약에 EJB엔진이 테이블을 직접 만들지 않는다면). | true (default)<br><br>false |
| jeus.ejb.enable.configDeleteOption | false로 설정되어 있으면 <deleting-table>의 설정이 무시된다. | true<br><br>false(default) |
| jeus.server.classpath | 이 변수는 EJB 실행환경을 위해 추가될, 부가적인 클래스패스 정보를 명시한다. 여기에 명시된 클래스 패스는 엔진내부의 모든 EJB 모듈에 적용되며 ClientContainer 클래스의 파라미터로서 EJB 클라이언트 애플이케이션에서 사용될 수도 있다. 이것은 EJBMain.xml의 <user-class-path> 태그와 동일한 것이다. | c:\user\MyLib;d:\classes |
| jeus.ejb.sourcegenerate.classpath | EJB 컴파일러에서 EJB 의 구현 클래스를 컴파일할 때 사용할 클래스패스를 추가한다. | c:\dev\myejb\ |

# C JEUS EJB 기본 Java Type 과 DB Field 매핑

## C.1 소개

이 문서에서는JEUS 특정인 Java 필드 종류와 JEUS에서 지원하는 주요 DB벤더의 database 컬럼 종류와의 기본 매핑을 나열하고 있다.

이 기본 매핑은 CMP bean의 `<schema-info><cm-field><type>` 가 jeus-ejb-dd.xml에 지정되어 있지 않을 때 사용된다. 이런 경우에는 EJB 시스템은 bean의 CMP 필드 중 Java 필드를 검색하여 Java 타입을 얻어 오고 <type>은 다음 절에 나오는 기본 매핑에 지정된 것과 같다고 여긴다.

다음에 나오는 테이블에서는 세 가지의 컬럼들을 보여 주고 있다.

- **Java 필드 종류**: 검색을 통하여 발견된 EJB CMP필드의 종류.
- **SQL 종류**: `java.sql.Types` 클래스에서 정의된 일반적인 SQL 종류.
- **DB 컬럼 종류**: 발견된 Java 필드 종류에 기반 하여 새로운 테이블을 생성할 때JEUS가 사용할 DB 컬럼 종류의 이름.

## C.2 Oracle 필드-컬럼 타입 매핑

*표 15. Oracle 을 위한 EJB CMP 필드-DB 컬럼 타입 매핑.*

| **Java 필드타입** | **SQL 타입** (java.sql.Types) | **Oracle DB 컬럼 타입** |
|---|---|---|
| `java.lang.String` | `VARCHAR` | `VARCHAR(255)` |
| `java.math.BigDecimal` | `NUMERIC` | `NUMERIC(15,5)` |
| `java.lang.Boolean` | `BIT` | `SMALLINT` |

| **Java 필드타입** | **SQL 타입** (java.sql.Types) | **Oracle DB 컬럼 타입** |
|---|---|---|
| Boolean | BIT | SMALLINT |
| java.lang.Byte | TINYINT | SMALLINT |
| Byte | TINYINT | SMALLINT |
| java.lang.Character | CHAR | CHAR(4) |
| Char | CHAR | CHAR(4) |
| java.lang.Short | SMALLINT | SMALLINT |
| Short | SMALLINT | SMALLINT |
| java.lang.Integer | INTEGER | INTEGER |
| Int | INTEGER | INTEGER |
| java.lang.Long | BIGINT | NUMERIC(22,0) |
| Long | BIGINT | NUMERIC(22,0) |
| java.lang.Float | REAL | REAL |
| Float | REAL | REAL |
| java.lang.Double | DOUBLE | DOUBLE PRECISION |
| Double | DOUBLE | DOUBLE PRECISION |
| byte[] | LONGVARBINARY | LONG RAW |
| java.sql.Date | DATE | DATE |
| java.sql.Time | TIME | DATE |
| java.sql.Timestamp | TIMESTAMP | DATE |
| <Java object> | JAVA_OBJECT | LONG RAW |

## C.3 Sybase 필드-컬럼 타입 매핑

*표 16. Sybase 를 위한 EJB CMP 필드-DB 컬럼 타입 매핑.*

| **Java 필드 타입** | **SQL 타입** (java.sql.Types) | **Sybase DB 컬럼 타입** |
|---|---|---|
| java.lang.String | VARCHAR | VARCHAR(255) |
| java.math.BigDecimal | NUMERIC | NUMERIC(15,5) |
| java.lang.Boolean | BIT | BIT |
| boolean | BIT | BIT |
| java.lang.Byte | TINYINT | TINYINT |
| byte | TINYINT | TINYINT |
| java.lang.Character | CHAR | CHAR(4) |
| char | CHAR | CHAR(4) |
| java.lang.Short | SMALLINT | SMALLINT |
| short | SMALLINT | SMALLINT |
| java.lang.Integer | INTEGER | INT |
| int | INTEGER | INT |
| java.lang.Long | BIGINT | NUMERIC(22,0) |
| long | BIGINT | NUMERIC(22,0) |
| java.lang.Float | REAL | REAL |
| float | REAL | REAL |
| java.lang.Double | DOUBLE | DOUBLE PRECISION |
| double | DOUBLE | DOUBLE PRECISION |

| Java 필드 타입 | SQL 타입 (java.sql.Types) | Sybase DB 컬럼 타입 |
|---|---|---|
| byte[] | LONGVARBINARY | IMAGE |
| java.sql.Date | DATE | DATETIME |
| java.sql.Time | TIME | DATETIME |
| java.sql.Timestamp | TIMESTAMP | Not supported |
| <Java object> | JAVA_OBJECT | IMAGE |

## C.4 MSSQL 필드-컬럼 타입 매핑

*표 17. MSSQL 을 위한EJB CMP 필드-DB 컬럼 타입 매핑.*

| Java 필드 타입 | SQL 타입 (java.sql.Types) | MSSQL DB 컬럼 타입 |
|---|---|---|
| java.lang.String | VARCHAR | VARCHAR(255) |
| java.math.BigDecimal | NUMERIC | NUMERIC(15,5) |
| java.lang.Boolean | BIT | BIT |
| boolean | BIT | BIT |
| java.lang.Byte | TINYINT | TINYINT |
| byte | TINYINT | TINYINT |
| java.lang.Character | CHAR | CHAR(4) |
| char | CHAR | CHAR(4) |
| java.lang.Short | SMALLINT | SMALLINT |
| short | SMALLINT | SMALLINT |

| Java 필드 타입 | SQL 타입 (java.sql.Types) | MSSQL DB 컬럼 타입 |
|---|---|---|
| java.lang.Integer | INTEGER | INT |
| int | INTEGER | INT |
| java.lang.Long | BIGINT | NUMERIC(22,0) |
| long | BIGINT | NUMERIC(22,0) |
| java.lang.Float | REAL | REAL |
| float | REAL | REAL |
| java.lang.Double | DOUBLE | FLOAT |
| double | DOUBLE | FLOAT |
| byte[] | LONGVARBINARY | IMAGE |
| java.sql.Date | DATE | DATETIME |
| java.sql.Time | TIME | DATETIME |
| java.sql.Timestamp | TIMESTAMP | DATETIME |
| <Java object> | JAVA_OBJECT | IMAGE |

## C.5 DB2 필드-컬럼 타입 매핑

*표 18. DB2 를 위한 EJB CMP 필드-DB 컬럼 타입 매핑.*

| Java 필드 타입 | SQL 타입 (java.sql.Types) | DB2 DB 컬럼 타입 |
|---|---|---|
| java.lang.String | VARCHAR | VARCHAR(255) |
| java.math.BigDecimal | NUMERIC | DECIMAL(15,5) |

| **Java 필드 타입** | **SQL 타입** (java.sql.Types) | **DB2 DB 컬럼 타입** |
|---|---|---|
| java.lang.Boolean | BIT | SMALLINT |
| boolean | BIT | SMALLINT |
| java.lang.Byte | TINYINT | SMALLINT |
| byte | TINYINT | SMALLINT |
| java.lang.Character | CHAR | CHARACTER(4) |
| char | CHAR | CHARACTER(4) |
| java.lang.Short | SMALLINT | SMALLINT |
| short | SMALLINT | SMALLINT |
| java.lang.Integer | INTEGER | INTEGER |
| int | INTEGER | INTEGER |
| java.lang.Long | BIGINT | BIGINT |
| long | BIGINT | BIGINT |
| java.lang.Float | REAL | REAL |
| float | REAL | REAL |
| java.lang.Double | DOUBLE | DOUBLE |
| double | DOUBLE | DOUBLE |
| byte[] | LONGVARBINARY | VARCHAR(255) |
| java.sql.Date | DATE | DATE |
| java.sql.Time | TIME | TIME |
| java.sql.Timestamp | TIMESTAMP | TIMESTAMP |

| Java 필드 타입 | SQL 타입 (java.sql.Types) | DB2 DB 컬럼 타입 |
|---|---|---|
| <Java object> | JAVA_OBJECT | LONG VARCHAR |

## C.6 Cloudscape 필드-컬럼 타입 매핑

*표 19. Cloudscape 를 위한 EJB CMP 필드-DB 컬럼 타입 매핑.*

| Java 필드 타입 | SQL 타입 (java.sql.Types) | Cloudscape DB 컬럼 타입 |
|---|---|---|
| java.lang.String | VARCHAR | VARCHAR(255) |
| java.math.BigDecimal | NUMERIC | DECIMAL(15,5) |
| java.lang.Boolean | BIT | BOOLEAN |
| boolean | BIT | BOOLEAN |
| java.lang.Byte | TINYINT | TINYINT |
| byte | TINYINT | TINYINT |
| java.lang.Character | CHAR | CHAR(4) |
| char | CHAR | CHAR(4) |
| java.lang.Short | SMALLINT | SMALLINT |
| short | SMALLINT | SMALLINT |
| java.lang.Integer | INTEGER | INTEGER |
| int | INTEGER | INTEGER |
| java.lang.Long | BIGINT | LONGINT |
| long | BIGINT | LONGINT |

| **Java 필드 타입** | **SQL 타입** (java.sql.Types) | **Cloudscape DB 컬럼 타입** |
|---|---|---|
| java.lang.Float | REAL | REAL |
| float | REAL | REAL |
| java.lang.Double | DOUBLE | DOUBLE PRECISION |
| double | DOUBLE | DOUBLE PRECISION |
| byte[] | LONGVARBINARY | LONG BIT VARYING |
| java.sql.Date | DATE | DATE |
| java.sql.Time | TIME | TIME |
| java.sql.Timestamp | TIMESTAMP | TIMESTAMP |
| <Java object> | JAVA_OBJECT | LONG BIT VARYING |

## C.7 Informix 필드-컬럼 타입 매핑

*표 20.Informix 를 위한 EJB CMP 필드-DB 컬럼 타입 매핑.*

| **Java 필드 타입** | **SQL 타입** (from java.sql.Types) | **Informix DB 컬럼 타입** |
|---|---|---|
| java.lang.String | VARCHAR | VARCHAR(255) |
| java.math.BigDecimal | NUMERIC | DECIMAL(15,5) |
| java.lang.Boolean | BIT | VARCHAR |
| Boolean | BIT | VARCHAR |
| java.lang.Byte | TINYINT | SMALLINT |
| Byte | TINYINT | SMALLINT |

| **Java 필드 타입** | **SQL 타입** (from java.sql.Types) | **Informix DB 컬럼 타입** |
|---|---|---|
| java.lang.Character | CHAR | CHAR |
| Char | CHAR | CHAR |
| java.lang.Short | SMALLINT | SMALLINT |
| Short | SMALLINT | SMALLINT |
| java.lang.Integer | INTEGER | INTEGER |
| Int | INTEGER | INTEGER |
| java.lang.Long | BIGINT | SERIAL8 |
| Long | BIGINT | SERIAL8 |
| java.lang.Float | REAL | REAL |
| Float | REAL | REAL |
| java.lang.Double | DOUBLE | DOUBLE PRECISION |
| Double | DOUBLE | DOUBLE PRECISION |
| byte[] | LONGVARBINARY | VARCHAR(255) |
| java.sql.Date | DATE | DATE |
| java.sql.Time | TIME | DATETIME HOUR TO SECOND |
| java.sql.Timestamp | TIMESTAMP | DATETIME YEAR TO SECOND |
| <Java object> | JAVA_OBJECT | BYTE |

# D EJBMain.xml XML 설정 레퍼런스

## D.1 소개

본 부록의 레퍼런스는 JEUS EJB 엔진의 메인 설정 파일인 EJBMain.xml 의 모든 태그에 대해서 설명하고 있다. 이 파일의 XSD 파일은 "JEUS_HOME\config\xsds" 디렉토리의 "ejb-main.xsd" 파일이다.

본 레퍼런스는 3부분으로 나눠져 있다.

1. **XML Schema/XML tree는**, XML구성 파일의 모든 엘리먼트의 계층적인 목록이다. 각 엘리먼트 트리의 형식은 다음과 같다.

    a. **index number** (예 (11))는, 태그 레퍼런스로 각 태그마다 인덱스를 붙여 놓은 것이다. 태그 레퍼런스는 이 번호순서대로 설명한다.

    b. XML Schema에서 정의한 XML 태그 명을 <**tag name**> 형식으로 표시한다.

    c. XML Schema 에서 정의한 Cardinality를 표시한다. "?" = 0개나 1개의 element, "+" = 1개 이상의 element, "*" = 0개 이상의 element, (기호가 없음) = 정확히 1개의 element.

    d. 몇몇 태그에는 "P" 문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.

2. **태그 레퍼런스**: 트리에 있는 각 XML 태그를 설명한다.

    a. **Description**: 태그에 대한 간단한 설명

    b. **Value Description**: 입력하는 값과 타입

    c. **Value Type**: 값의 데이터 타입. 예) String

d. **Default Value**: 해당 XML을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 XML 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위해서 추천하는 값

h. **Child Elements**: 자신의 태그 안에 사용하는 하위 태그

3. **Example XML 파일**: "EJBMain.xml"에 대한 완전한 예제

## D.2 XML Schema/XML 트리

```
(1) <ejb-engine>
   (2) <resolution>? P
   (3) <enable-user-notify>? P
   (4) <active-management>?
      (5) <max-blocked-thread>? P
      (6) <max-idle-time>? P
      (7) <email-notify>?
         (8) <smtp-host-address>
         (9) <from-address>
         (10) <to-address>
         (11) <cc-address>?
         (12) <bcc-address>?
   (13) <invoke-http>?
      (14) <url>
      (15) <http-port> P
   (16) <timer-service>?
      (17) <min-delivery-interval>? P
      (18) <max-redelivery-count>? P
      (19) <redelivery-interval>? P
      (20) <thread-pool>?
         (21) <min>? P
         (22) <max>? P
         (23) <period>? P
```

```
(24) <durable-setting>?
   (25) <db-vendor>
   (26) <data-source-name>
   (27) <engine-type>
   (28) <table-name>? P
```

# D.3 Element Reference

**(1) <ejb-engine>**

| | |
|---|---|
| *Description* | [en] This is the root element of EJBMain.xml. |
| *Example* | `<ejb-engine> ... </ejb-engine>` |
| *Child Elements* | (2)resolution?<br>(3)enable-user-notify?<br>(4)active-management?<br>(13)invoke-http?<br>(16)timer-service? |

**(2) <ejb-engine> <resolution>**

| | |
|---|---|
| *Description* | EJB 의 비활성화와 Garbage Collection 은 이 element 에서 정의된 시간 간격으로 시도된다. EJB 엔진에서 서비스 중인 어떤 빈이 클라이언트로부터 더 이상의 요청이 없을 때 EJB 엔진은 이 빈을 비활성화 한다. 이 작업의 수행 주기를 설정한다. |
| *Value Description* | millisecond |
| *Value Type* | nonNegativeLongType |
| *Value Type Description* | 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다. |
| *Default Value* | 300000 |
| *Example* | `<resolution>100000</resolution>` |

**(3) <ejb-engine> <enable-user-notify>**

| | |
|---|---|
| *Description* | 이 특성이 활성화된다면 EJB exception 은 JEUSMain.xml 의 엔진 컨테이너에 정의된 user log 에 기록된다. |

*Value Type* boolean

*Default Value* false

*Example*
```
<enable-user-notify>true</enable-user-notify>
```

### (4) <ejb-engine> **<active-management>**

*Description* EJB 엔진을 모니터링하고 오류를 처리하며 그 결과를 전자우편을 통해 관리자에게 통지한다.

*Value Description* millisecond

*Example* `<resolution>100000</resolution>` [Performace Recommendation]: 일반적으로 사용자는 `EJBMain.xml` 의 `active-management` 보다는 `Servlet` 엔진에 포함된 `active-management` 을 사용한다( `JEUS` 웹서버 가이드 참조 ).

*Child Elements*
```
(5)max-blocked-thread?
(6)max-idle-time?
(7)email-notify?
```

### (5) <ejb-engine> <active-management> **<max-blocked-thread>**

*Description* 이 element 는 블럭된 thread 의 최대 개수를 설정한다. 이 설정 값 보다 EJB 에 block 된 Thread 개수가 많을 경우에 Container 를 restart 시킨다. 이 값이 작게 설정 되어 있다면 EJB 엔진이 너무 자주 재시작될 수도 있기 때문에 주의가 필요하다.

*Value Description* thread 개수

*Value Type* off-intType

*Value Type Description* 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

*Default Value* -1

*Defined Value* -1
이 값은 블럭된 thread 개수에 대한 제한이 없음을 의미한다. 즉 이렇게 설정하면 EJB 엔진은 절대로 블럭된 thread 때문에 재시작

되지는 않는다.

*Example* `<max-blocked-thread>200</max-blocked-thread>`

`(6) <ejb-engine> <active-management> <max-idle-time>`

*Description* 이 element 는 EJB thread 가 블럭되었다고 간주되는 상태를 정의한다. 이 element 의 값은 시간을 의미하며 만약 어떤 thread 가 이 시간을 초과되도록 idle 상태를 유지한다면 이 thread 는 블럭되었다고 간주된다.

*Value Description* millisecond

*Value Type* nonNegativeLongType

*Value Type Description* 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

*Default Value* 300000

*Example* `<max-idle-time>180000</max-idle-time>`

`(7) <ejb-engine> <active-management> <email-notify>`

*Description* active management 나 혹은 다른 비정상적인 상태에 의해서 엔진이 재시작 한다면 이 element 에서 정의된 곳으로 전자 메일을 보내서 상황을 알린다.

*Child Elements*
```
(8)smtp-host-address
(9)from-address
(10)to-address
(11)cc-address?
(12)bcc-address?
```

`(8) <ejb-engine> <active-management> <email-notify> <smtp-host-address>`

*Description* email 을 보낼 smtp server 의 주소를 지정한다.

*Value Type* token

`(9) <ejb-engine> <active-management> <email-notify> <from-address>`

*Description* email 을 보내는 사람의 address 를 지정한다.

*Value Type* token

```
(10) <ejb-engine> <active-management> <email-notify> <to-address>
```

*Description* email 을 받는 사람의 address 를 지정한다.

*Value Type* token

```
(11) <ejb-engine> <active-management> <email-notify> <cc-address>
```

*Description* email 을 참조로 받는 사람의 address 를 지정한다.

*Value Type* token

```
(12) <ejb-engine> <active-management> <email-notify> <bcc-address>
```

*Description* email 을 숨은 참조로 받는 사람의 address 를 지정한다.

*Value Type* token

```
(13) <ejb-engine> <invoke-http>
```

*Description* 이 기능을 설정하면 클라이언트 측의 EJB stub 과 원격지의 RMI 실행환경은 HTTP-RMI 요청 ( Request )으로 통신한다. 이것은 방화벽을 사이에 두고 EJB 에 접근할 때 사용된다. 이 모드( HTTP 호출 모드 )를 사용할 때 클라이언트가 EJB stub 에서 메소드를 호출하면 HTTP-RMI 요청 ( Request )은 이것을 웹 컨테이너로 보낼 웹서버로 발송된다. 그리고 이것은 RMI Handler Servlet (jeus.rmi.http.ServletHandler )으로 보내지고 여기서 Handler Servlet 은 요청( Request )으로부터 HTTP 헤더를 제거한 뒤 이것을 RMI 실행환경으로 전송한다. 이 element 가 설정되기 앞서 jeus.rmi.http.ServletHandler Servlet 은 반드시 JEUS 웹 컨테이너에 Deploy 되어 있어야만 한다( JEUS 웹 가이드를 참고한다. ).

*Performance Recommendation* HTTP 호출 모드를 사용함으로써 약간의 성능 향상을 기대할 수 있다.

*Child Elements*
```
(14)url
(15)http-port
```

```
(14) <ejb-engine> <invoke-http> <url>
```

*Description* HTTP-RMI stub 에 의해 호출될 RMI Handler Servlet (jeus.rmi.http.ServletHandler) 의 URI 경로가 반드시 설정되어야 한다. 이 URL 은 프로토콜, 웹 서버 IP, 포트번호를 제외하고 오직 Servlet 요청 경로만을 설정해야 한다. 프로토콜은 HTTP, RMI 실행환경과 웹 서버는 같은 IP 주소를 가지고 있다고 가정 한다(이것은 웹 서버와 웹 컨테이너는 반드시 HTTP-RMI 요청을 같은 머신에서 받는다는 것을 의미한다). 포트번호는 다음에 설명할 element(HTTP-port)에서 설정한다.

*Value Description* RMI Handler Servlet 을 명시한 Servlet 컨텍스트 경로

*Value Type* token

*Example* `<url>/mycontext/RMIHandlerServlet</url>`

(15) <ejb-engine> <invoke-http> **<http-port>**

*Description* HTTP-RMI 요청을 받고 처리할 웹 서버의 포트번호를 설정한다. 이 웹 서버/웹 컨테이너에서는 반드시 RMI Handler Servlet 이 Deploy 되어 있고 이미 실행 중이어야만 한다.

*Value Description* HTTP-RMI stub 가 연결할 웹 서버의 포트번호.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 80

(16) <ejb-engine> **<timer-service>**

*Description* 이 EJB engine 이 제공하는 Timer service 에 대한 설정을 한다.Timer Service 를 persistence 하게 관리하려면 이 설정이 있어야 한다.

*Child Elements*
(17)min-delivery-interval?
(18)max-redelivery-count?
(19)redelivery-interval?
(20)thread-pool?
(24)durable-setting?

(17) <ejb-engine> <timer-service> **<min-delivery-interval>**

*Description* 이 값은 특정 timer 가 생성된 후에 timer event 가 발생하기까지의 최소 시간이다.

*Value Description* millisecond 단위

*Value Type* nonNegativeLongType

*Value Type Description* 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

*Default Value* 7000

*Performance Recommendation* 이 값이 작을수록 EJB 엔진에 부하를 줄 수 있으므로 사용하는 timer 의 interval 에 맞게 크게 설정하면 좋다.

(18) <ejb-engine> <timer-service> **<max-redelivery-count>**

*Description* 이 값은 timer callback 함수에서 exception 이 발생하거나 transaction 이 rollback 되어 재전송이 발생하는 상황에서 최대 재전송 횟수를 나타낸다.

*Value Description* 재전송 횟수

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 1

(19) <ejb-engine> <timer-service> **<redelivery-interval>**

*Description* 이 값은 재전송을 해야 할 상황이 발생했을 때부터 재전송하기까지의 시간이다.

*Value Description* millisecond 단위이다.

*Value Type* nonNegativeLongType

*Value Type Description* 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

*Default Value* 4000

(20) <ejb-engine> <timer-service> **<thread-pool>**

*Description* Timer service 가 timeout() method 를 실행할때 사용하는 thread pool 에

대한 설정이다.

*Child Elements*

```
(21)min?
(22)max?
(23)period?
```

```
(21) <ejb-engine> <timer-service> <thread-pool> <min>
```

*Description* pooling 되는 객체의 최소값을 지정한다.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 2

```
(22) <ejb-engine> <timer-service> <thread-pool> <max>
```

*Description* pooling 되는 객체의 최대값을 지정한다.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 30

```
(23) <ejb-engine> <timer-service> <thread-pool> <period>
```

*Description* pooling 되는 객체를 정리하는 시간을 지정한다.

*Value Type* long

*Default Value* 3600000

*Performance Recommendation* 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(24) <ejb-engine> <timer-service> <durable-setting>
```

*Description* Timer service 가 timer 들을 persistence 하게 관리하기 위해 필요한 설정이다. 이 설정을 하지 않으면 timer 가 persistence 하게 관리되지 않는다.

*Child Elements*

(25)db-vendor
(26)data-source-name
(27)engine-type
(28)table-name?

(25) <ejb-engine> <timer-service> <durable-setting> **<db-vendor>**

*Description*

Timer service 가 persistence 를 위해 사용하는 DB 의 vendor 설정이다.값은 jeus-ejb-dd.xml 의 <schema-info>의 <db-vendor>와 같다.

*Value Type*

db-vendorType

*Defined Value*

oracle
Oracle DBMS 인 경우

informix
Informix DBMS 인 경우

db2
IBM DB2 DBMS 인 경우

mssql
Microsoft SQL Server DBMS 인 경우

sybase
Sybase DBMS 인 경우

hsql
HSQL DBMS 인 경우

cloudscape
Cloudscape DBMS 인 경우

(26) <ejb-engine> <timer-service> <durable-setting> **<data-source-name>**

*Description*

Timer service 가 persistence 를 위해 사용하는 DB 의 datasource 의 export name 설정이다. 값은 jeus-ejb-dd.xml 의 <schema-info>의 <data-source-name>와 같다.

*Value Type* token

(27) <ejb-engine> <timer-service> <durable-setting> **<engine-type>**

*Description* Timer service 가 DB 에 접근하는 방식을 설정한다. 값은 jeus-ejb-dd.xml 의 <persistence-optimize>의 <engine-type>와 같다.

*Value Type* engineType

*Defined Value*

EXCLUSIVE_ACCESS
각각의 Entity Bean 인스턴스는 그것이 나타내는 데이터베이스의 열과 일대일 관계를 가지고 그에 대해서 독점적인 접근을 한다. 이것은 하나의 인스턴스가 살아있는 동안 다른 Entity Bean 은 데이터에 접근하거나 변경할 수 없다는 사실을 의미하기 때문에 EJB 엔진은 ejbLoad() 호출을 모두 생략할 수 있다는 것을 나타낸다. 엔진은 빈이 생성될 때 한 번 ejbLoad() 호출을 실행하고 빈이 살아있는 동안 데이터베이스에 더 이상의 어떤 변화도 없다고 가정하고 ejbLoad() 호출을 하지 않는다. 이 빈은 이것이 클러스터링된 빈의 일부일 때 사용할 수 없다. 데이터베이스는 WAS 에 의해서만 변경이 허락된다는 점을 반드시 기억하기 바란다( WAS 가 아닌 접근은 허용되지 않는다).

SINGLE_OBJECT
element 의 값을 이것으로 설정한다면 다른 EJB 엔진의 여러 EJB 들이 같은 데이터베이스 열에 매핑될 수 있다. ejbLoad()는 Entity Bean 으로 요청이 들어오기 앞서서 항상 호출된다. SINGLE_OBJECT 와 MULTIPLE_OBJECT 는 같은 종류의 EJB 를 클러스터링할 때 필요하다. SINGLE_OBJECT 와 MULTIPLE_OBJECT 의 차이점은 SINGLE_OBJECT 인 경우 각각의 EJB 엔진 안에서 오직 하나의 EJB Entity Bean 이 EJB 클라이언트의 모든 요청을 처리한다. 즉 같은 EJB 엔진으로 다른 EJB 클라이언트의 요청이 도착한다면 먼저 연결을 맺고 있는 다른 클라이언트의 요청이 끝날 때까지 대기상태에 있어야 한다.

MULTIPLE_OBJECT
element 의 값을 이것으로 설정한다면 다른 EJB 엔진의 여러 EJB 들이 같은 데이터베이스 열에 매핑될 수 있다. ejbLoad()는 Entity

Bean 으로 요청이 들어오기 앞서서 항상 호출된다. SINGLE_OBJECT 과 MULTIPLE_OBJECT 는 같은 종류의 EJB 를 클러스터링할 때 필요하다. SINGLE_OBJECT 와 MULTIPLE_OBJECT 의 차이점은 MULTIPLE_OBJECT 인 경우 각각의 EJB 엔진 안에서 모든 EJB 클라이언트의 요청을 동시에 처리할 여러 EJB Entity Bean 인스턴스가 생성된다.즉 SINGLE_OBJECT 모드와 달리 EJB 클라이언트의 요청은 먼저 처리하고 있는 다른 요청이 끝날 때까지 대기하지 않아도 된다.

(28) <ejb-engine> <timer-service> <durable-setting> **<table-name>**

*Description* Timer service 가 사용하는 DB 의 Table 이름을 설정한다.

*Value Type* token

*Default Value* Jeus_Timer

# D.4 샘플 EJBMain.xml 파일

<<EJBMain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-engine xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
   <resolution>300000</resolution>
   <enable-user-notify>false</enable-user-notify>
   <active-management>
      <max-blocked-thread>-1</max-blocked-thread>
      <max-idle-time>300000</max-idle-time>
      <email-notify>
         <smtp-host-address>mail.foo.com</smtp-host-address>
         <from-address>admin@mail.foo.com</from-address>
         <to-address>admin@mail.foo.com</to-address>
         <cc-address>admin@mail.foo.com</cc-address>
         <bcc-address>admin@mail.foo.com</bcc-address>
      </email-notify>
   </active-management>
   <invoke-http>
      <url>/mycontext/RMIHandlerServlet</url>
```

```
      <http-port>80</http-port>
   </invoke-http>
   <timer-service>
      <min-delivery-interval>7000</min-delivery-interval>
      <max-redelivery-count>1</max-redelivery-count>
      <redelivery-interval>4000</redelivery-interval>
      <thread-pool>
         <min>2</min>
         <max>30</max>
         <period>3600000</period>
      </thread-pool>
      <durable-setting>
         <db-vendor>oracle</db-vendor>
         <data-source-name>datasource</data-source-name>
         <engine-type>EXCLUSIVE_ACCESS</engine-type>
         <table-name>Jeus_Timer</table-name>
      </durable-setting>
   </timer-service>
</ejb-engine>
```

# E jeus-ejb-dd.xml XML Configuration Reference

## E.1 소개

본 부록의 레퍼런스는 JEUS EJB 모듈의 Deployment Descriptor 파일인 jeus-ejb-dd.xml의 모든 태그에 대해서 설명하고 있다. 이 파일의 XSD 파일은 “JEUS_HOME\config\xsds” 디렉토리의 “jeus-ejb-dd.xsd” 파일이다.

본 레퍼런스는 3부분으로 나눠져 있다.

4. **XML Schema/XML tree는**, XML구성 파일의 모든 엘리먼트의 계층적인 목록이다. 각 엘리먼트 트리의 형식은 다음과 같다.

    a. **index number** (예 (11))는, 태그 레퍼런스로 각 태그마다 인덱스를 붙여 놓은 것이다. 태그 레퍼런스는 이 번호순서대로 설명한다.

    b. XML Schema에서 정의한 XML 태그 명을 <**tag name**> 형식으로 표시한다.

    c. XML Schema 에서 정의한 Cardinality를 표시한다. “?” = 0개나 1개의 element, “+” = 1개 이상의 element, “*” = 0개 이상의 element, (기호가 없음) = 정확히 1개의 element.

    d. 몇몇 태그에는 “P” 문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.

5. **태그 레퍼런스**: 트리에 있는 각 XML 태그를 설명한다.

    a. **Description**: 태그에 대한 간단한 설명

    b. **Value Description**: 입력하는 값과 타입

    c. **Value Type**: 값의 데이터 타입. 예) String

d. **Default Value**: 해당 XML을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 XML 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위해서 추천하는 값

h. **Child Elements**: 자신의 태그 안에 사용하는 하위 태그

6. **Example XML 파일**: "EJBMain.xml"에 대한 완전한 예제

## E.2 XML Schema/XML 트리

```
(1) <jeus-ejb-dd>
   (2) <module-info>?
      (3) <keep-generated>? P
      (4) <fast-deploy>? P
      (5) <role-permission>*
         (6) <principal>+
         (7) <role>
         (8) <actions>?
         (9) <classname>?
         (10) <excluded>?
         (11) <unchecked>?
      (12) <unspecified-method-permission>?
         (13) <role>*
         (14) <excluded>
         (15) <unchecked>
      (16) <unspecified-container-transaction>?
   (17) <beanlist>
      (18) <jeus-bean>*
         (19) <ejb-name>
         (20) <export-name>?
         (21) <local-export-name>?
         (22) <export-port>? P
         (23) <export-iiop>?
```

```
   (24) <only-iiop> P
(25) <single-vm-only>? P
(26) <local-invoke-optimize>? P
(27) <use-access-control>? P
(28) <run-as-identity>?
   (29) <principal-name>
(30) <security-interop>?
   (31) <integrity-confidentiality>? P
   (32) <trust-in-client>? P
   (33) <client-auth>? P
   (34) <identity-assertion>? P
(35) <env>*
   (36) <type>
   (37) <name>
   (38) <value>
(39) <ejb-ref>?
   (40) <jndi-info>*
      (41) <ref-name>
      (42) <export-name>
(43) <res-ref>?
   (44) <jndi-info>*
      (45) <ref-name>
      (46) <export-name>
(47) <res-env-ref>?
   (48) <jndi-info>*
      (49) <ref-name>
      (50) <export-name>
(51) <service-ref>?
   (52) <service-client>*
      (53) <service-ref-name>
      (54) <port-info>*
         (55) <service-endpoint-interface>?
         (56) <wsdl-port>?
         (57) <stub-property>*
            (58) <name>
            (59) <value>
         (60) <call-property>*
            (61) <name>
            (62) <value>
```

```
(63) <security>?
   (64) <request-sender>?
      (65) <action-list>
      (66) <password-callback-class>?
      (67) <user>
      (68) <timeToLive>?
      (69) <userNameToken>?
         (70) <passwordType>?
         (71) <userTokenElements>?
      (72) <signature-infos>?
         (73) <signature-info>+
            (74) <signatureParts>?
            (75) <keyIdentifier>
            (76) <keystore>
               (77) <key-type>
               (78) <keystore-password>
               (79) <keystore-filename>
      (80) <encryption-infos>?
         (81) <encryption-info>+
            (82) <encryptionParts>?
            (83) <encryptionSymAlgorithm>?
            (84) <encryptionUser>?
            (85) <keyIdentifier>
            (86) <keystore>?
               (87) <key-type>
               (88) <keystore-password>
               (89) <keystore-filename>
            (90) <embeddedKey>?
               (91) <embeddedKeyCallbackClass>
               (92) <key-name>
   (93) <response-receiver>?
      (94) <action-list>
      (95) <password-callback-class>?
      (96) <timeToLive>?
      (97) <decryption>?
         (98) <keystore>
            (99) <key-type>
            (100) <keystore-password>
            (101) <keystore-filename>
```

```
                  (102) <signature-verification>?
                     (103) <keystore>
                        (104) <key-type>
                        (105) <keystore-password>
                        (106) <keystore-filename>
      (107) <service-impl-class>?
      (108) <wsdl-override>?
      (109) <service-qname>?
      (110) <call-property>*
         (111) <name>
         (112) <value>
(113) <thread-max>? P
(114) <clustering>?
   (115) <home-clustered>? P
   (116) <ejb-home-failover>? P
   (117) <ejb-home-idempotent-method>*
      (118) <method-name>
      (119) <method-params>?
         (120) <method-param>*
   (121) <ejb-remote-failover>? P
   (122) <ejb-remote-idempotent-method>*
      (123) <method-name>
      (124) <method-params>?
         (125) <method-param>*
(126) <invoke-http>?
   (127) <url>
   (128) <http-port> P
(129) <pooling-bean>? P
(130) <object-management>?
   (131) <bean-pool>?
      (132) <pool-min>? P
      (133) <pool-max>? P
      (134) <resizing-period>? P
   (135) <connect-pool>?
      (136) <pool-min>? P
      (137) <pool-max>? P
      (138) <resizing-period>? P
   (139) <capacity>? P
   (140) <passivation-timeout>? P
```

```
   (141) <disconnect-timeout>? P
(142) <file-db-info>?
   (143) <local-file-db>?
      (144) <file-db-path>?
      (145) <file-db-name>?
      (146) <min-hole>? P
      (147) <packing-rate>? P
   (148) <remote-file-db>?
      (149) <remote-primary-file-db>
      (150) <remote-backup-file-db>?
      (151) <conn-pool-size>? P
(152) <persistence-optimize>?
   (153) <engine-type>? P
   (154) <non-modifying-method>*
      (155) <method-name>
      (156) <method-params>?
         (157) <method-param>*
   (158) <entity-cache-size>? P
   (159) <update-delay-till-tx>? P
   (160) <include-update>? P
(161) <schema-info>?
   (162) <table-name>?
   (163) <cm-field>*
      (164) <field>
      (165) <column-name>?
      (166) <type>?
      (167) <exclude-field>? P
   (168) <creating-table>?
      (169) <use-existing-table>?
      (170) <force-creating-table>?
   (171) <deleting-table>? P
   (172) <prim-key-field>*
      (173) <field>
      (174) <column-name>?
      (175) <type>?
      (176) <exclude-field>? P
   (177) <find-method>*
      (178) <query-method>
         (179) <method-name>
```

```
            (180) <method-params>?
               (181) <method-param>*
         (182) <sql>?
         (183) <include-updates>?
      (184) <jeus-query>*
         (185) <query-method>
            (186) <method-name>
            (187) <method-params>?
               (188) <method-param>*
         (189) <sql>?
         (190) <include-updates>?
      (191) <db-vendor>
      (192) <data-source-name>
      (193) <auto-key-generator>?
         (194) <generator-type>
         (195) <generator-name>?
         (196) <sequence-column>?
         (197) <key-cache-size>? P
   (198) <database-insert-delay>? P
   (199) <cm-persistence-optimize>?
      (200) <subengine-type>? P
      (201) <fetch-size>? P
      (202) <init-caching>? P
   (203) <enable-instant-ql>? P
   (204) <connection-factory-name>?
   (205) <mdb-resource-adapter>?
      (206) <resource-adapter-name>
      (207) <activation-config>?
         (208) <description>*
         (209) <activation-config-property>+
            (210) <activation-config-property-name>
            (211) <activation-config-property-value>
   (212) <destination>?
   (213) <max-message>? P
   (214) <ack-mode>?
   (215) <durable>?
   (216) <msg-selector>?
   (217) <jndi-spi>?
      (218) <mq-vendor>
```

```
            (219) <initial-context-factory>
            (220) <provider-url>?
         (221) <durable-timer-service>?
            (222) <enable-durable-timers>? P
            (223) <ignore-durable-timers-at-deploy>? P
            (224) <delete-durable-timers-at-undeploy>? P
   (225) <ejb-relation-map>*
      (226) <relation-name>
      (227) <table-name>?
      (228) <jeus-relationship-role>*
         (229) <relationship-role-name>
         (230) <column-map>*
            (231) <foreign-key-column>
            (232) <target-primary-key-column>
   (233) <message-destination>?
      (234) <jndi-info>*
         (235) <ref-name>
         (236) <export-name>
```

# E.3 Element Reference

**(1) <jeus-ejb-dd>**

| | |
|---|---|
| *Description* | 단일 JEUS EJB 모듈의 최상위 element. 각각의 jeus-ejb-dd.xml 파일에는 이 태그가 반드시 존재한다. |
| *Child Elements* | (2)module-info?<br>(17)beanlist<br>(225)ejb-relation-map*<br>(233)message-destination? |

**(2) <jeus-ejb-dd> <module-info>**

| | |
|---|---|
| *Description* | EJB 모듈 전체에 적용되는 포괄적인 정보를 설정한다. |
| *Child Elements* | (3)keep-generated?<br>(4)fast-deploy?<br>(5)role-permission*<br>(12)unspecified-method-permission?<br>(16)unspecified-container-transaction? |

(3) <jeus-ejb-dd> <module-info> **<keep-generated>**

*Description* 이 module 에 대해서 생성된 java source file 을 남길지의 여부를 지정한다. 이 설정이 true 일때는 jeus-application-dd.xml 의 설정에 우선한다. false 인 경우에는 jeus-application-dd.xml 의 keep-generated 설정에 따른다.

*Value Type* boolean

*Default Value* false

(4) <jeus-ejb-dd> <module-info> **<fast-deploy>**

*Description* 이 module 이 deploy 전에 ejb module compiler 에 의해 compile 되어 fast deploy 가 가능할때 설정한다. 이 설정이 true 일때는 jeus-application-dd.xml 의 설정에 우선한다. false 인 경우에는 jeus-application-dd.xml 의 fast deploy 설정에 따른다.

*Value Type* boolean

*Default Value* false

(5) <jeus-ejb-dd> <module-info> **<role-permission>**

*Description* 이 EJB module 에서의 user principal 과 ejb-jar.xml 에서 사용하는 role 의 관계를 설정한다.

*Child Elements*
(6)principal+
(7)role
(8)actions?
(9)classname?
(10)excluded?
(11)unchecked?

(6) <jeus-ejb-dd> <module-info> <role-permission> **<principal>**

*Description* role 에 해당하는 user principal 을 지정한다.

*Value Description* security 의 subjects.xml 에서 지정되어 있는 principal 이름

*Value Type* token

(7) <jeus-ejb-dd> <module-info> <role-permission> **<role>**

*Description* principal 들에게 부여할 role 이름을 지정한다.

*Value Type* token

(8) <jeus-ejb-dd> <module-info> <role-permission> **<actions>**

*Description* 이 role permission 객체에 대한 action 을 정의한다. default 로 사용되는 role permission 은 정해진 action 이 없다.

*Value Type* token

(9) <jeus-ejb-dd> <module-info> <role-permission> **<classname>**

*Description* 사용할 role permission class name 을 지정한다. 지정하지 않으면 JEUS 에서 기본적으로 제공하는 class 가 사용된다.

*Value Type* token

(10) <jeus-ejb-dd> <module-info> <role-permission> **<excluded>**

*Description* role 을 사용하지 못하도록 만든다.

*Example*
```
<excluded/>
```

(11) <jeus-ejb-dd> <module-info> <role-permission> **<unchecked>**

*Description* 아무런 체크없이 role 을 사용가능하도록 만든다.

*Example*
```
<unchecked/>
```

(12) <jeus-ejb-dd> <module-info> **<unspecified-method-permission>**

*Description* ejb-jar.xml 에서 method permission 이 지정되어 있지 않은 method 에 대한 설정을 한다.

*Child Elements*
```
(13)role*
(14)excluded
(15)unchecked
```

(13) <jeus-ejb-dd> <module-info> <unspecified-method-permission> **<role>**

*Description* ejb-jar.xml 에서 method permission 이 지정되어 있지 않은 method 의 permission 을 여기에 설정된 role 에게 부여한다. 다른 role 의 principal 은 이 method 들을 호출할 permission 을 얻지 못한다.

*Value Description* ejb-jar.xml 의 <assembler-description>에서 지정된 role name

*Value Type* token

```
(14) <jeus-ejb-dd> <module-info> <unspecified-method-permission>
<excluded>
```

*Description* ejb-jar.xml 에서 method permission 이 지정되어 있지 않은 method 를 exclude 시킨다. 따라서 어떤 principal 은 이 method 들을 호출할 permission 을 얻지 못한다.

```
(15) <jeus-ejb-dd> <module-info> <unspecified-method-permission>
<unchecked>
```

*Description* ejb-jar.xml 에서 method permission 이 지정되어 있지 않은 method 를 unchecked 로 간주한다. 따라서 모든 principal 은 이 method 들을 호출할 permission 을 얻는다.

```
(16) <jeus-ejb-dd> <module-info> <unspecified-container-transaction>
```

*Description* ejb-jar.xml 에서 container transaction 이 지정되어 있지 않은 method 에 대한 설정을 한다. 이 값의 default 는 -Djeus.ejb.transaction.attribute.default 로 설정할수 있다. 이 값이 지정되어 있지 않으면 Supports 가 default 로 사용된다.

*Value Type* trans-attributeType

*Defined Value* NotSupported

Supports

Required

RequiresNew

Mandatory

Never

```
(17) <jeus-ejb-dd> <beanlist>
```

*Description* 각 bean 의 설정을 하는 element 이다.

*Child Elements*
```
(18)jeus-bean*
```

### (18) <jeus-ejb-dd> <beanlist> **<jeus-bean>**

*Description* 각 bean 에 대한 jeus specific 설정을 한다. ejb-jar.xml 의 각 bean 마다 이 설정이 되어야 한다. bean 의 종류마다 설정해야 할 element 가 다르므로 JEUS EJB 메뉴얼을 참고해서 설정하기 바란다.

*Child Elements*
```
(19)ejb-name
(20)export-name?
(21)local-export-name?
(22)export-port?
(23)export-iiop?
(25)single-vm-only?
(26)local-invoke-optimize?
(27)use-access-control?
(28)run-as-identity?
(30)security-interop?
(35)env*
(39)ejb-ref?
(43)res-ref?
(47)res-env-ref?
(51)service-ref?
(113)thread-max?
(114)clustering?
(126)invoke-http?
(129)pooling-bean?
(130)object-management?
(142)file-db-info?
(152)persistence-optimize?
(161)schema-info?
(198)database-insert-delay?
(199)cm-persistence-optimize?
(203)enable-instant-ql?
(204)connection-factory-name?
(205)mdb-resource-adapter?
(212)destination?
(213)max-message?
(214)ack-mode?
```

```
(215)durable?
(216)msg-selector?
(217)jndi-spi?
(221)durable-timer-service?
```

**(19) <jeus-ejb-dd> <beanlist> <jeus-bean> <ejb-name>**

*Description* ejb-jar.xml 에 지정된 ejb-name 을 가리킨다.

*Value Type* token

*Example*
```
<ejb-name>teller</ejb-name>
```

**(20) <jeus-ejb-dd> <beanlist> <jeus-bean> <export-name>**

*Description* JNDI Naming System 에 등록될 유일한 이름이다. 클러스터링에 참여하는 모든 빈은 같은 export-name 을 가져야 한다.

*Value Description* 임의로 지정할 수 있고 JNDI Naming System 에서 반드시 유일한 이름이어야만 한다.

*Value Type* token

*Example*
```
<export-name>TELLEREJB</export-name>
```

**(21) <jeus-ejb-dd> <beanlist> <jeus-bean> <local-export-name>**

*Description* 빈이 로컬 빈 인터페이스일 때 사용하는 JNDI 이름이다.

*Value Description* 임의로 지정할 수 있고 JNDI Naming System 에서 반드시 유일한 이름이어야만 한다.

*Value Type* token

*Example*
```
<local-export-name>LOCALTELLEREJB</local-export-
name>
```

**(22) <jeus-ejb-dd> <beanlist> <jeus-bean> <export-port>**

*Description* 이 element 는 이 빈이 서비스하게 될 RMI Listener Port 를 명시한다. 이 설정은 클라이언트와 EJB 간에 방화벽이 있을 때 사용되기도 한다. 이 element 는 관리자가 RMI 통신을 허용하는 포트 번호를 제공하는 경우에만 사용할 수 있다.

*Value Description* 포트번호

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 0

*Example*
```
<export-port>7654</export-port>
```

(23) <jeus-ejb-dd> <beanlist> <jeus-bean> **<export-iiop>**

*Description* 이 element 를 설정하면 빈의 인터페이스 가 IIOP stub 과 skeleton 으로서 COS Naming Server 에 export 될 수 있도록 한다. 이는 IIOP 로 접근 가능한 모든 클라이언트가 빈에 접근 가능하도록 한다.

*Child Elements* (24)only-iiop

(24) <jeus-ejb-dd> <beanlist> <jeus-bean> <export-iiop> **<only-iiop>**

*Description* 이 EJB 의 Home 을 IIOP 이외에 RMI Stub 도 같이 등록할지를 결정한다. 같이 등록된다면 CosNaming 에는 IIOP stub 이, JEUS JNDI 에는 RMI stub 이 등록된다.

*Value Type* boolean

*Default Value* true

(25) <jeus-ejb-dd> <beanlist> <jeus-bean> **<single-vm-only>**

*Description* "true"로 설정되어 있으면 JNDI 서버는 빈이 실행 되고 있는 JVM 에서만 빈 이름이 export 될 수 있도록 범위를 한정시킨다. 이 의미는 해당 빈을 접근 할 수 있는 클라이언트는 동일한 JVM 내에서 운영되고 있는 Servlet 과 빈들뿐이라는 것이다(즉 그 빈은 현재의 Engine Container 에서만 볼 수 있다는 것이다). 이 옵션은 같은 빈이 다른 Engine Container 에도 Deploy 되어 있을 경우에 유용하게 사용될 수 있다. 일반적으로 빈이 같은 JNDI export 이름을 가지고 있을 경우에는 이 이름들이 export 될 때 서로를 JNDI Naming Server 에서 overwrite 한다. 이 옵션을 사용함으로써 각 빈들은 빈을 운영하고 있는 JVM 에서만 인식되고 범위가 제한됨으로 export 이름이 overwrite 되지 않는다.

| | |
|---|---|
| *Value Type* | boolean |
| *Default Value* | false |
| *Example* | `<single-vm-only>true</single-vm-only>` |

**(26) <jeus-ejb-dd> <beanlist> <jeus-bean> <local-invoke-optimize>**

| | |
|---|---|
| *Description* | 이 element 는 두 개의 EJB 가 서로 통신하고 같은 JVM 안에서 실행된다면 메소드 호출을 할지 RMI 통신을 할지를 결정한다. 이 옵션이 활성화되었을 때 메소드 호출로 통신한다(call-by-reference). 이 사실을 이용해서 EJB 프로그래밍을 하지 않을 것을 권장한다. 모든 메소드 호출은 call-by-value 인 것처럼 취급하라. |
| *Value Type* | boolean |
| *Default Value* | true |
| *Example* | `<local-invoke-optimize>true</local-invoke-optimize>` |
| *Performance Recommendation* | 성능을 향상시키기 위해서 두 개의 EJB 가 같은 JVM 에 존재할 때 이 값을 true 로 설정한다. |

**(27) <jeus-ejb-dd> <beanlist> <jeus-bean> <use-access-control>**

| | |
|---|---|
| *Description* | EJB method 를 호출하는 중에 method 를 호출한 principal 을 대상으로 EJB method 가 사용하는 resource 에 대해 J2SE Security 에서 제공하는 access-control 을 사용할 것인지를 지정한다. 이 기능이 동작하기 위해서는 JEUS 를 기동할 때 -Djava.security.manager 를 설정해서 security manager 를 활성화시켜야 한다. |
| *Value Type* | boolean |
| *Default Value* | false |
| *Example* | `<use-access-control>false</use-access-control>` |
| *Performance Recommendation* | access control 을 check 하지 않을거라면 false 로 지정하는 것이 성능에 도움이 된다. |

**(28) <jeus-ejb-dd> <beanlist> <jeus-bean> <run-as-identity>**

*Description* 이 element 는 ejb-jar.xml 에 정의된 run-as-specified-identity role 이름을 실세 사용자이름(principal)으로 매핑을 정의한다.

*Child Elements* `(29)principal-name`

`(29) <jeus-ejb-dd> <beanlist> <jeus-bean> <run-as-identity>` **`<principal-name>`**

*Description* ejb-jar.xml 안에서 run-as-specified-identity 의 role 로 사용되는 principal 이름.

*Value Description* security 의 subjects.xml 에서 지정되어 있는 principal 이름

*Value Type* token

`(30) <jeus-ejb-dd> <beanlist> <jeus-bean>` **`<security-interop>`**

*Description* 이 element 는 IIOP/CSI 를 사용하고 EJB 엔진에서 사용가능할 때 즉 enable-interop element 가 true 일 때 선언된다. 다음의 설정에 대한 정보가 필요하면 CSI 스펙을 참고하라.

*Child Elements*
`(31)integrity-confidentiality?`
`(32)trust-in-client?`
`(33)client-auth?`
`(34)identity-assertion?`

`(31) <jeus-ejb-dd> <beanlist> <jeus-bean> <security-interop>` **`<integrity-confidentiality>`**

*Description* 이 element 는 CSI 스펙에 정의된 "Integrity" 비트와 "Confidentiality" 비트를 매핑한다. 설정은 여기에서 정의된 대로 두 개의 비트 모두에 대해서 설정을 적용된다.

*Value Type* security-interopElementType

*Default Value* NotSupported

*Defined Value* NotSupported
"TLS_SEC_TRANS.target_supports" 비트배열의 "Integrity" 과 "Confidentiality" 비트를 0 으로 설정한다(disable).

Supports

"TLS_SEC_TRANS.target_supports" 비트배열의 "Integrity" 과 "Confidentiality" 비트를 1 로 설정한다(enable).

Requires
"TLS_SEC_TRANS.target_requires" 비트배열의 "Integrity" 과 "Confidentiality" 비트를 1 로 설정한다(enable). 그리고 "TLS_SEC_TRANS. target_supports" 비트배열의 "Integrity" 과 "Confidentiality" 비트를 1 로 설정한다(enable).

*Example*

```
<integrity- confidentiality> Requires</integrity-confidentiality>
```

(32) <jeus-ejb-dd> <beanlist> <jeus-bean> <security-interop> **<trust-in-client>**

*Description* 이 element 는 CSI 스펙에 정의된 "Trust in client" 비트를 매핑한다.

*Value Type* security-interopElementType

*Default Value* NotSupported

*Defined Value*

NotSupported
"TLS_SEC_TRANS.target_supports" 비트배열의 "Client authentication"비트를 0 으로 설정한다 (disable).

Supports
"TLS_SEC_TRANS.target_supports" 비트배열의 "Client authentication"비트를 1 로 설정한다 (enable).

Requires
"TLS_SEC_TRANS.target_requires" 비트배열의 "Client authentication"비트를 1 로 설정한다 (enable). 그리고 "TLS_SEC_TRANS.target _supports" 비트배열의 "Client authentication"비트를 1 로 설정한다(enable).

*Example*

```
<trust-in-client>Requires</trust-in-client>
```

(33) <jeus-ejb-dd> <beanlist> <jeus-bean> <security-interop> **<client-auth>**

*Description* 이 element 는 CSI 스펙에 정의된 "Client authentication" 비트를 매핑한다.

*Value Type* security-interopElementType

*Default Value* NotSupported

*Defined Value* NotSupported
"AS_ContextSec.target_supports" 비트 배열의 "Client authentication" 비트를 0 으로 설정한다 (disable).

Supports
"AS_ContextSec.target_supports" 비트 배열의 "Client authentication" 비트를 1 로 설정한다 (enable).

Requires
"AS_ContextSec.target_requires" 비트 배열의 "Client authentication" 비트를 1 로 설정한다(enable). 그리고
"AS_ContextSec.target_supports" 비트 배열의 "Client authentication" 비트를 1 로 설정한다(enable).

*Example*
```
<client-auth>Requires</client-auth>
```

(34) <jeus-ejb-dd> <beanlist> <jeus-bean> <security-interop> **<identity-assertion>**

*Description* 이 element 는 CSI 스펙에 정의된 "Identity assertion" 비트를 매핑한다.

*Value Type* security-interopElementRestrictedType

*Default Value* NotSupported

*Defined Value* NotSupported
"SAS_ContextSec.target_supports" 비트 배열의 "Client authentication" 비트를 0 으로 설정한다 (disable).

Supports
"SAS_ContextSec.target_supports" 비트 배열의 "Client authentication" 비트를 1 로 설정한다 (enable).

*Example* `<integrity- confidentiality> Requires</integrity-confidentiality>`

(35) <jeus-ejb-dd> <beanlist> <jeus-bean> **<env>**

*Description* 표준 EJB DD 정의된 <env-entry>태그에 추가되거나 Override 된다.

*Child Elements*
(36)type
(37)name
(38)value

(36) <jeus-ejb-dd> <beanlist> <jeus-bean> <env> **<type>**

*Description* 환경 변수의 자바 타입.

*Value Description* 다음의 자바 타입 중 하나를 선택해야 한다. java.lang.Boolean, java.lang.String, java.lang.Integer, java.lang.Double, java.lang.Byte, java.lang.Short, java.lang.Long, java.lang.Float, java.lang.Character.

*Value Type* token

*Example* `<type>java.lang.Integer</type>`

(37) <jeus-ejb-dd> <beanlist> <jeus-bean> <env> **<name>**

*Description* 코드에서 사용하는 환경 변수의 이름.

*Value Type* token

*Example* `<name>minAmount</name>`

(38) <jeus-ejb-dd> <beanlist> <jeus-bean> <env> **<value>**

*Description* 이 값은 대응하는 wrapper 클래스 생성자의 파라미터로서 사용 된다.

*Value Type* token

*Example* `<value>100</value>`

(39) <jeus-ejb-dd> <beanlist> <jeus-bean> **<ejb-ref>**

*Description* 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다.

*Child Elements*
(40)jndi-info*

(40) <jeus-ejb-dd> <beanlist> <jeus-bean> <ejb-ref> **<jndi-info>**

*Description* 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다. 예를 들면 실제 JNDI 이름이 "ACCEJB"인 account EJB 를 코드상에서 "ejb/account"으로 lookup 할 수 있다.

*Child Elements* (41)ref-name
(42)export-name

(41) <jeus-ejb-dd> <beanlist> <jeus-bean> <ejb-ref> <jndi-info> **<ref-name>**

*Description* 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수 있다.

*Value Description* 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

*Value Type* token

*Example* <ref-name>ejb/AccountEJB</ref-name>

(42) <jeus-ejb-dd> <beanlist> <jeus-bean> <ejb-ref> <jndi-info> **<export-name>**

*Description* JEUS DD 에 정의된 실제 JNDI 이름.

*Value Type* token

*Example* <export-name>ACCEJB</export-name>

(43) <jeus-ejb-dd> <beanlist> <jeus-bean> **<res-ref>**

*Description* 이 element 는 소스코드 상에서 사용할 수 있는 외부 자원(예: 데이터베이스) 참조 이름을 선언할 수 있다.

*Child Elements* (44)jndi-info*

(44) <jeus-ejb-dd> <beanlist> <jeus-bean> <res-ref> **<jndi-info>**

*Description* 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다. 예를 들면 실제 JNDI 이름이 "ACCEJB"인 account EJB 를 코드상에서 "ejb/account"으로 lookup 할 수 있다.

*Child Elements* (45)ref-name
(46)export-name

(45) <jeus-ejb-dd> <beanlist> <jeus-bean> <res-ref> <jndi-info> **<ref-name>**

*Description* 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수 있다.

*Value Description* 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

*Value Type* token

*Example* <ref-name>ejb/AccountEJB</ref-name>

(46) <jeus-ejb-dd> <beanlist> <jeus-bean> <res-ref> <jndi-info> **<export-name>**

*Description* JEUS DD 에 정의된 실제 JNDI 이름.

*Value Type* token

*Example* <export-name>ACCEJB</export-name>

(47) <jeus-ejb-dd> <beanlist> <jeus-bean> **<res-env-ref>**

*Description* 이 element 는 코드에서 사용하는 외부자원 참조를 실제 JNDI 이름으로 bind 한다. 이 element 는 표준 EJB DD 의 <resource-env-ref>에 대응한다.

*Child Elements* (48)jndi-info*

(48) <jeus-ejb-dd> <beanlist> <jeus-bean> <res-env-ref> **<jndi-info>**

*Description* 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다. 예를 들면 실제 JNDI 이름이 "ACCEJB"인 account EJB 를 코드상에서 "ejb/account"으로 lookup 할 수 있다.

*Child Elements* (49)ref-name
(50)export-name

(49) <jeus-ejb-dd> <beanlist> <jeus-bean> <res-env-ref> <jndi-info> **<ref-name>**

*Description* 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수 있다.

*Value Description* 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

*Value Type* token

*Example* `<ref-name>ejb/AccountEJB</ref-name>`

`(50) <jeus-ejb-dd> <beanlist> <jeus-bean> <res-env-ref> <jndi-info> **<export-name>**`

*Description* JEUS DD 에 정의된 실제 JNDI 이름.

*Value Type* token

*Example* `<export-name>ACCEJB</export-name>`

`(51) <jeus-ejb-dd> <beanlist> <jeus-bean> **<service-ref>**`

*Description* JEUS 웹 서비스 클라이언트 설정 문서의 루트 엘리먼트(root element).

*Child Elements* `(52)service-client*`

`(52) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> **<service-client>**`

*Description* 배치되는 웹서비스 클라이언트를 위한 설정들을 표시한다.

*Child Elements*
`(53)service-ref-name`
`(54)port-info*`
`(107)service-impl-class?`
`(108)wsdl-override?`
`(109)service-qname?`
`(110)call-property*`

`(53) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> **<service-ref-name>**`

*Description* WSDL 파일에서 관련된 웹서비스 엔드포인트 이름이다. <service-ref-name>은 표준 배치 서술자 web.xml 혹은 ejb-jar.xml 의 <service-ref-

name>에 상응한다.

*Value Type* token

```
(54) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info>
```

*Description* 배치되는 웹서비스 클라이언트가 호출하는 웹서비스 포트 정보를 표시한다.

*Child Elements*
```
(55)service-endpoint-interface?
(56)wsdl-port?
(57)stub-property*
(60)call-property*
(63)security?
```

```
(55) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <service-endpoint-interface>
```

*Description* WSDL port 의 서비스 엔드포인드 인터페이스를 나타내는 클래스를 표시한다. <service-ref>에서 <port-component-ref>의 <service-endpoint-interface>에 상응한다.

*Value Type* token

```
(56) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <wsdl-port>
```

*Description* <port-info>와 연결된 WSDL port 정의를 표시한다.

*Value Type* QName

```
(57) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <stub-property>
```

*Description* 특정 port 에서 사용하는 javax.xml.rpc.Stub 객체에 설정하는 property 들을 표시한다.

*Child Elements*
```
(58)name
(59)value
```

```
(58) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <stub-property> <name>
```

*Description* javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

*Value Type* string

```
(59) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <stub-property> <value>
```

*Description* javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

*Value Type* string

```
(60) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <call-property>
```

*Description* 특정 port 에서 사용하는 javax.xml.rpc.Call 객체에 설정하는 property 들을 표시한다.

*Child Elements*
(61)name
(62)value

```
(61) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <call-property> <name>
```

*Description* javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

*Value Type* string

```
(62) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <call-property> <value>
```

*Description* javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

*Value Type* string

```
(63) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security>
```

*Description* 웹서비스의 보안(WS-Security)을 위한 웹 서비스 클 라이언트 설정이다.

*Child Elements* (64)request-sender?
(93)response-receiver?

```
(64) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender>
```

*Description* 웹서비스를 호출하는 메시지에 보안을 적용하기 위한 설정이 다.

*Child Elements* (65)action-list
(66)password-callback-class?
(67)user
(68)timeToLive?
(69)userNameToken?
(72)signature-infos?
(80)encryption-infos?

```
(65) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <action-list>
```

*Description* 어떤 보안을 적용할 것인지를 String 으로 나열한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

*Value Type* string

```
(66) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <password-callback-
class>
```

*Description* 패스워드를 설정하는 콜백 클래스의 풀 패키지 명이다.

*Value Type* string

```
(67) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <user>
```

*Description* UsernameToken 에 들어갈 이름과 서명에 들어갈 키의 별칭 을 설정한다.

*Value Type* string

```
(68) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <timeToLive>
```

*Description* 보내게 될 메시지의 유효기간을 초 단위로 설정한다. 기본값 은 300 초이다.

*Value Type* string

```
(69) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <userNameToken>
```

*Description* UsernameToken 을 설정한다.

*Child Elements*
```
(70)passwordType?
(71)userTokenElements?
```

```
(70) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <userNameToken>
<passwordType>
```

*Description* UsernameToken 에 사용될 패스워드의 타입 설정이다.
"PasswordDigest" 혹은 "PasswordText"를 사용할 수 있다.

*Value Type* passwordTypeType

*Defined Value*
PasswordDigest
UsernameToken 에 설정되는 암호가 base64 encoding 된 상태 로 메시지에 포함된다.

PasswordText
UsernameToken 에 설정되는 암호가 평이한 텍스트로 메시지에 포함된다.

```
(71) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <userNameToken>
<userTokenElements>
```

*Description* UsernameToken 에 추가될 엘리먼트의 리스트이다. 각 항목은 공백으로 분리된다. "nonce" 혹은 "created"가 사용될 수 있다. passwordType 이 "PasswordText"일 경우에 사용가능하다.

*Value Type* string

```
(72) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
```

```
client> <port-info> <security> <request-sender> <signature-infos>
```

*Description* 메시지에 서명을 하기 위한 설정이다.

*Child Elements* (73)signature-info+

```
(73) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <signature-infos>
<signature-info>
```

*Description* 메시지의 서명을 위한 설정히다. 복수 설정이 가능하다.

*Child Elements*
(74)signatureParts?
(75)keyIdentifier
(76)keystore

```
(74) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <signature-infos>
<signature-info> <signatureParts>
```

*Description* 메시지의 특정 부분을 서명하고자 할 때 사용한다. "{}{http://schemas.xmlsoap.org/soap/envelope/}Body; Token"과 같은 방식으로 열거할 수 있다. 기본적으로 설정하지 않았을 경우에는 SOAP 몸체 전체를 서명하게 되어 있다.

*Value Type* string

```
(75) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <signature-infos>
<signature-info> <keyIdentifier>
```

*Description* 서명에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier 중의 하나를 사용한다.

*Value Type* sigKeyIdentifierType

*Defined Value* IssuerSerial
X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference

X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참조하는 방식이다.

SKIKeyIdentifier
Subject Key Idenfication 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 X509 인증서를 포함하고 서명 검증을 위해 사용하도 록 한다.

```
(76) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <request-sender> <signature-infos> <signature-info> <keystore>
```

*Description* 메시지의 서명을 위한 개인키를 저장하고 있는 키스토어의 설정이다.

*Child Elements*
(77)key-type
(78)keystore-password
(79)keystore-filename

```
(77) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <request-sender> <signature-infos> <signature-info> <keystore> <key-type>
```

*Description* 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

*Value Type* string

```
(78) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <request-sender> <signature-infos> <signature-info> <keystore> <keystore-password>
```

*Description* 키 스토어에 접근하기 위한 암호 설정이다.

*Value Type* string

```
(79) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <request-sender> <signature-infos> <signature-info> <keystore> <keystore-filename>
```

*Description* 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로 에서 찾게 된다.

*Value Type* string

```
(80) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
```

*Description* 메시지를 암호화 하기 위한 설정이다.

*Child Elements*
```
(81)encryption-info+
```

```
(81) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info>
```

*Description* 배치되는 웹서비스 클라이언트를 위한 설정들을 표시한다.

*Child Elements*
```
(82)encryptionParts?
(83)encryptionSymAlgorithm?
(84)encryptionUser?
(85)keyIdentifier
(86)keystore?
(90)embeddedKey?
```

```
(82) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <encryptionParts>
```

*Description* 특정 부분을 암호화 하기 위한 설정이다.
"{mode}{ns}{localname};{mode}{ns}{localname};…" 과 같은 형식이다.
기본 mode 값은 content 이다.
예:{Content}{http://example.org/payment}CreCard;{Element}{}UserName

*Value Type* string

```
(83) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <encryptionSymAlgorithm>
```

*Description* 암호화에 사용하는 알고리즘이다. AES_128, AES_256,

TRIPLE_DES, AES_192 를 지원한다.

*Value Type* string

```
(84) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <encryptionUser>
```

*Description* 암호화에 사용되는 키의 별칭이다.

*Value Type* string

```
(85) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <keyIdentifier>
```

*Description* 암호화에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier EmbeddedKeyName 중의 하나를 사용한다.

*Value Type* encKeyIdentifierType

*Defined Value*

IssuerSerial
X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference
X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참조하는 방식이다.

SKIKeyIdentifier
Subject Key Idenfication 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 암호화에 사용된 X509 인증서를 포함한다.

EmbeddedKeyName
웹서비스와 웹서비스 클라이언트가 공유하는 세션키를 사용할 때 사용한다. 웹서비스와 클라이언트는 키의 이름만을 주고 받음으로써

어떤 키를 사용했는지를 알 수 있다.

```
(86) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <keystore>
```

*Description* 암호화에 사용될 키의 저장소 설정이다.

*Child Elements*
(87)key-type
(88)keystore-password
(89)keystore-filename

```
(87) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <keystore> <key-type>
```

*Description* 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

*Value Type* string

```
(88) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <keystore> <keystore-password>
```

*Description* 키 스토어에 접근하기 위한 암호 설정이다.

*Value Type* string

```
(89) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <keystore> <keystore-filename>
```

*Description* 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

*Value Type* string

```
(90) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <request-sender> <encryption-infos>
<encryption-info> <embeddedKey>
```

*Description* 웹서비스와 웹서비스 클라이언트가 공유하고 있는 키를 설정한다.

keyIdentifier 가 "EmbeddedKeyName"으로 설정되어야 사용할 수 있다.

*Child Elements*

```
(91)embeddedKeyCallbackClass
(92)key-name
```

```
(91) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <request-sender> <encryption-infos> <encryption-info> <embeddedKey> <embeddedKeyCallbackClass>
```

*Description* 세션 키를 사용하려 할 경우, 키의 바이트 정보를 가지고 있 는 콜백 클래스를 설정한다.

*Value Type* string

```
(92) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <request-sender> <encryption-infos> <encryption-info> <embeddedKey> <key-name>
```

*Description* 세션 키의 이름을 설정한다.

*Value Type* string

```
(93) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <response-receiver>
```

*Description* 웹서비스 응답 메시지가 보안 적용이 되어있을 경우, 처리하 기 위한 설정이다.

*Child Elements*

```
(94)action-list
(95)password-callback-class?
(96)timeToLive?
(97)decryption?
(102)signature-verification?
```

```
(94) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <port-info> <security> <response-receiver> <action-list>
```

*Description* 받게 되는 메시지가 어떤 보안이 적용되어 있어야 하는지 설정한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

*Value Type* string

```
(95) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <password-callback-
class>
```

*Description* 패스워드 콜백 클래스의 이름을 풀 패키지 이름으로 입력한다.

*Value Type* string

```
(96) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <timeToLive>
```

*Description* 받은 메시지의 유효기간을 설정한다(초단위). 기본값은 생성 시간으로 부터 300 초 동안이다.

*Value Type* string

```
(97) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <decryption>
```

*Description* 받는 메시지의 암호화 된 부분을 해독하기 위한 설정이다.

*Child Elements* (98)keystore

```
(98) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <decryption>
<keystore>
```

*Description* 메시지의 암호를 해독하기 위한 키 스토어의 설정이다.

*Child Elements* (99)key-type
(100)keystore-password
(101)keystore-filename

```
(99) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <decryption>
<keystore> <key-type>
```

*Description* 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

*Value Type* string

```
(100) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
```

```
client> <port-info> <security> <response-receiver> <decryption>
<keystore> <keystore-password>
```

*Description* 키 스토어에 접근하기 위한 암호 설정이다.

*Value Type* string

```
(101) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <decryption>
<keystore> <keystore-filename>
```

*Description* 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

*Value Type* string

```
(102) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <signature-
verification>
```

*Description* 받는 메시지의 서명을 검증하기 위한 설정이다.

*Child Elements* (103)keystore

```
(103) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <signature-
verification> <keystore>
```

*Description* 서명을 검증하기 위한 키 스토어 설정이다.

*Child Elements* (104)key-type
(105)keystore-password
(106)keystore-filename

```
(104) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <signature-
verification> <keystore> <key-type>
```

*Description* 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

*Value Type* string

```
(105) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
```

```
client> <port-info> <security> <response-receiver> <signature-
verification> <keystore> <keystore-password>
```

*Description* 키 스토어에 접근하기 위한 암호 설정이다.

*Value Type* string

```
(106) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <port-info> <security> <response-receiver> <signature-
verification> <keystore> <keystore-filename>
```

*Description* 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

*Value Type* string

```
(107) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <service-impl-class>
```

*Description* 웹서비스 클라이언트를 위한 서비스 구현체를 표시한다. 배치 시에 자동 생성되므로 웹서비스 배치자가 설정할 필요가 없다.

*Value Type* token

```
(108) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <wsdl-override>
```

*Description* <service-ref>의 <wsdl-file>을 대체하기 위한 WSDL 파일의 위치를 표시한다. 표시된 위치는 유효한 URL 이어야 한다.

*Value Type* string

```
(109) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <service-qname>
```

*Description* WSDL 의 WSDL service 정의를 표시한다.

*Value Type* QName

```
(110) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-
client> <call-property>
```

*Description* WSDL service 에서 사용하는 모든 javax.xml.rpc.Call 객체에 설정하는

property 들을 표시한다.

*Child Elements* (111)name
(112)value

```
(111) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <call-property> <name>
```

*Description* javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

*Value Type* string

```
(112) <jeus-ejb-dd> <beanlist> <jeus-bean> <service-ref> <service-client> <call-property> <value>
```

*Description* javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

*Value Type* string

```
(113) <jeus-ejb-dd> <beanlist> <jeus-bean> <thread-max>
```

*Description* EJB 엔진이 클라이언트의 요청을 받고 처리하는 쓰레드의 최대 갯수를 설정한다. 이 값만큼의 thread 가 이미 사용되고 있다면 그 다음의 request 들은 다른 thread 들이 사용가능하게 될때까지 기다리게 된다.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 100

```
(114) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering>
```

*Description* 클러스터링 설정은 장애 대처와 로드 분산 기능을 제공한다. 이것은 기본적으로 하나의 EJB 가 다른 여러 EJB 엔진에 모두 설치되었을 때 사용 가능하다. 이것은 JEUS 노드간의 클러스터링 방법과 설정이 유사하다. 클러스터링에 참여하는 모든 EJB 는 같은 export-name 을 가져야 한다.

*Child Elements* (115)home-clustered?

(116)ejb-home-failover?
(117)ejb-home-idempotent-method*
(121)ejb-remote-failover?
(122)ejb-remote-idempotent-method*

**(115) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <home-clustered>**

*Description* 이 element 를 true 로 설정하면 클러스터링 기능이 활성화된다.

*Value Type* boolean

*Default Value* false

*Example*
```
<home-clustered>true</home-clustered>
```

**(116) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-home-failover>**

*Description* 이 element 를 "true"로 설정하면 failover 는 클러스터링된 빈의 홈 메소드를 호출할 때 발생한다. 이것은 에러가 발생하기 전이나 빈의 홈 메소드가 실행도중이면 다른 빈의 홈 메소드가 선택되어 실행된다는 것을 의미한다.

*Value Type* boolean

*Default Value* false

*Example*
```
<ejb-home-failover>true</ejb-home-failover>
```

**(117) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-home-idempotent-method>**

*Description* 이 element 는 "idempotent" 홈 메소드라는 것을 선언한다. 이 메소드는 수행 중에 어떤 상태 즉 빈 그 자신이나 데이터베이스의 필드 내용 등이 변경되지 않음을 보장한다. idempotent 메소드는 기본적으로 어떤 업데이트나 설정의 변화 없이 결과값을 반환하는 getter 메소드이다. 이런 메소드를 명시하는 이유는 어떤 메소드가 Idempotent 메소드라면 첫 번째 시도 했던 메소드 호출이 실행 중에 실패했더라도 안전하게 다시 호출이 가능하다. 그러나 그 메소드가 Idempotent 메소드가 아니라면 이런 경우 해결책이 없다. 같은

부작용을 가지게 되는 두 번의 연속적이고 중복적인 메소드 호출로 불일치 상태에 이르는 위험을 감수해야 하며 이 경우 exception 을 던지는 것이 더 합리적이다. 따라서 Idempotent method 를 많이 사용할수록 EJB failover 는 효율적으로 작동될 것이다.

*Child Elements* `(118)method-name`
`(119)method-params?`

`(118) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-home-idempotent-method> **<method-name>**`

*Description* method 의 이름을 지정한다.

*Value Type* token

*Example* `foo`

`(119) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-home-idempotent-method> **<method-params>**`

*Description* method 의 parameter 들을 순서대로 지정한다.

*Child Elements* `(120)method-param*`

`(120) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-home-idempotent-method> <method-params> **<method-param>**`

*Description* method 의 parameter 의 fully qualified class name 을 지정한다.

*Value Type* token

*Example* `java.lang.String`

`(121) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> **<ejb-remote-failover>**`

*Description* 이 element 가 "true"로 설정되면 클러스터링된 EJB 빈의 remote/business 메소드에 대해서 failover 기능이 수행된다. 이것은 에러가 발생하기 전이나 빈의 홈 메소드가 실행도중이면 다른 빈의 홈 메소드가 선택되어 실행된다는 것을 의미한다.

*Value Type* boolean

*Default Value* false

*Example* `<ejb-remote-failover>true</ejb-remote-failover>`

(122) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> **<ejb-remote-idempotent-method>**

*Description* 이 element 는 "idempotent"리모트 메소드라는 것을 선언한다. 이 메소드는 수행 중에 어떤 상태 즉 빈 그 자신이나 데이터베이스의 필드 내용 등이 변경되지 않음을 보장한다. idempotent 메소드는 기본적으로 어떤 업데이트나 설정의 변화없이 결과값을 반환하는 getter 메소드이다. 이런 메소드를 명시하는 이유는 어떤 메소드가 Idempotent 메소드라면 첫 번째 시도 했던 메소드 호출이 실행 중에 실패했더라도 안전하게 다시 호출이 가능하다. 그러나 그 메소드가 Idempotent 메소드가 아니라면 이런 경우 해결책이 없다. 같은 부작용을 가지게 되는 두 번의 연속적이고 중복적인 메소드 호출로 불일치 상태에 이르는 위험을 감수해야 하며 이 경우 exception 을 던지는 것이 더 합리적이다. 따라서 Idempotent method 를 많이 사용할수록 EJB failover 는 효율적으로 작동될 것이다.

*Child Elements* `(123)method-name`
`(124)method-params?`

(123) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-remote-idempotent-method> **<method-name>**

*Description* method 의 이름을 지정한다.

*Value Type* token

*Example* `foo`

(124) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-remote-idempotent-method> **<method-params>**

*Description* method 의 parameter 들을 순서대로 지정한다.

*Child Elements* `(125)method-param*`

(125) <jeus-ejb-dd> <beanlist> <jeus-bean> <clustering> <ejb-remote-idempotent-method> <method-params> **<method-param>**

*Description* method 의 parameter 의 fully qualified class name 을 지정한다.

*Value Type* token

*Example* `java.lang.String`

(126) <jeus-ejb-dd> <beanlist> <jeus-bean> **<invoke-http>**

*Description* 이 기능을 설정하면 클라이언트 측의 EJB stub 과 원격지의 RMI 실행환경은 HTTP-RMI 요청(Request)으로 통신한다. 이것은 방화벽을 사이에 두고 EJB 에 접근할 때 사용된다. 이 모드 ( HTTP 호출 모드 )를 사용할 때 클라이언트가 EJB stub 에서 메소드를 호출하면 HTTP-RMI 요청 ( Request )은 이것을 웹 컨테이너로 보낼 웹 서버로 발송된다. 그리고 이것은 RMI Handler Servlet(jeus.rmi.http.ServletHandler)으로 보내지고 여기서 Handler Servlet 은 요청(Request )으로부터 HTTP 헤더를 제거한 뒤 이것을 RMI 실행환경으로 전송한다. 이 element 가 설정되기에 앞서 jeus.rmi.http.ServletHandler Servlet 은 반드시 JEUS 웹 컨테이너에 Deploy 되어 있어야만 한다( 더 자세한 정보는 JEUS 웹 서버 가이드를 참고하라. ).

*Performance Recommendation* HTTP 호출 모드를 사용함으로써 약간의 성능 향상을 기대할 수 있다.

*Child Elements* `(127)url`
`(128)http-port`

(127) <jeus-ejb-dd> <beanlist> <jeus-bean> <invoke-http> **<url>**

*Description* HTTP-RMI stub 에 의해 호출될 RMI Handler Servlet (jeus.rmi.http.ServletHandler) 의 URI 경로가 반드시 설정되어야 한다. 이 URL 은 프로토콜, 웹 서버 IP, 포트번호를 제외하고 오직 Servlet 요청 경로만을 설정해야 한다. 프로토콜은 HTTP, RMI 실행환경과 웹 서버는 같은 IP 주소를 가지고 있다고 가정 한다(이것은 웹 서버와 웹 컨테이너는 반드시 HTTP-RMI 요청을 같은 머신에서 받는다는 것을 의미한다). 포트번호는 다음에 설명할 element(HTTP-port)에서 설정한다.

*Value Description* RMI Handler Servlet 을 명시한 Servlet 컨텍스트 경로

| | |
|---|---|
| *Value Type* | token |
| *Example* | `<url>/mycontext/RMIHandlerServlet</url>` |

`(128) <jeus-ejb-dd> <beanlist> <jeus-bean> <invoke-http>` **`<http-port>`**

| | |
|---|---|
| *Description* | HTTP-RMI 요청을 받고 처리할 웹 서버의 포트번호를 설정한다. 이 웹 서버/웹 컨테이너에서는 반드시 RMI Handler Servlet 이 Deploy 되어 있고 이미 실행 중이어야만 한다. |
| *Value Description* | HTTP-RMI stub 가 연결할 웹 서버의 포트번호. |
| *Value Type* | nonNegativeIntType |
| *Value Type Description* | 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다. |
| *Default Value* | 80 |

`(129) <jeus-ejb-dd> <beanlist> <jeus-bean>` **`<pooling-bean>`**

| | |
|---|---|
| *Description* | stateful session bean 에 대해서만 적용되는 element 로 이 element 를 true 로 설정하면 상태유지 세션 빈 인스턴스는 비활성화(클라이언트 요청을 모두 처리하고 빈 풀에 인스턴스가 반환된 상태)된 후에 재사용된다. |
| *Value Type* | boolean |
| *Default Value* | false |
| *Performance Recommendation* | 상태유지 세션 빈의 재사용을 금지시킬 특별한 이유가 없다면 전체적인 성능 최적화를 위해서 이 기능을 true 로 설정한다. |

`(130) <jeus-ejb-dd> <beanlist> <jeus-bean>` **`<object-management>`**

| | |
|---|---|
| *Description* | 오직 상태유지 세션 빈과 엔티티 빈에만 적용되는 객체 관리 기능은 이 빈에 대한 빈 인스턴스 풀링 작업을 제어한다. |
| *Child Elements* | `(131)bean-pool?`<br>`(135)connect-pool?`<br>`(139)capacity?`<br>`(140)passivation-timeout?`<br>`(141)disconnect-timeout?` |

(131) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management> **<bean-pool>**

*Description* 빈 풀은 EJB 빈의 구현 클래스 인스턴스를 가지고 있다. 인스턴스들은 클라이언트의 요청을 처리하기 위해 EJB Context 와 빈 skeleton 구현 클래스에 연결되었을 때 풀에서 나와서 서비스를 시작한다. 풀에 더 이상의 인스턴스가 남지 않았을 때 새로운 인스턴스가 생성되고 그것은 빈 풀에 추가된다.

*Child Elements*
(132)pool-min?
(133)pool-max?
(134)resizing-period?

(132) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management> <bean-pool> **<pool-min>**

*Description* 풀에 담을 수 있는 빈 인스턴스의 최소 개수.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 0

(133) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management> <bean-pool> **<pool-max>**

*Description* 풀에 있는 빈 인스턴스의 최대 개수.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 100

(134) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management> <bean-pool> **<resizing-period>**

*Description* 빈 풀의 사이즈가 재조정되는 시간 간격. 이 시간마다 사용되지 않는 인스턴스를 풀에서 제거한다.

*Value Type* nonNegativeLongType

*Value Type Description* 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

*Default Value* 60000

```
(135) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management>
<connect-pool>
```

*Description* 커넥션 풀은 클라이언트와 빈 풀에서 가져온 EJB 인스턴스를 중개하는 EJB 리모트 인터페이스 구현 클래스(EJB objects)를 가지고 있다. 이 풀에서 커넥션 인스턴스를 호출해서 클라이언트의 요청과 연결을 맺는다.

*Child Elements*
```
(136)pool-min?
(137)pool-max?
(138)resizing-period?
```

```
(136) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management>
<connect-pool> <pool-min>
```

*Description* 풀에 담을 수 있는 빈 인스턴스의 최소 개수.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 0

```
(137) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management>
<connect-pool> <pool-max>
```

*Description* 풀에 있는 빈 인스턴스의 최대 개수.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 100

```
(138) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management>
<connect-pool> <resizing-period>
```

*Description* 빈 풀의 사이즈가 재조정되는 시간 간격. 이 시간마다 사용되지 않는 인스턴스를 풀에서 제거한다.

*Value Type* nonNegativeLongType

*Value Type Description* 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

*Default Value* 60000

(139) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management> **<capacity>**

*Description* 이것은 생성될 것으로 예상되는 빈 인스턴스의 최대 개수를 의미한다. 이 값은 EJB 와 연계될 내부 클라이언트 세션 데이터의 효율적인 구성을 위해 사용된다.

*Value Description* bean 의 갯수

*Value Type* positiveIntType

*Value Type Description* 기본적으로 non-negative int 형이지만 0 이상의 값만 허용한다.

*Default Value* 10000

(140) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management> **<passivation-timeout>**

*Description* 이 element 에서 명시한 특정 기간 동안 클라이언트의 요청을 받지 않은 빈을 EJB 엔진에서 비활성화 한다. 비활성화 된 빈은 메모리로부터 제거되고 그 상태는 파일이나 DB 에 저장된다.

*Value Description* millisecond 단위이다.

*Value Type* off-longType

*Value Type Description* 기본적으로 unsigned long 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

*Default Value* 300000

(141) <jeus-ejb-dd> <beanlist> <jeus-bean> <object-management> **<disconnect-timeout>**

*Description* 이 element 는 여기서 설정된 시간 동안 클라이언트 의 요청을 받지 못하면 클라이언트와 빈 인스턴스 사이의 연결을 취소하는데 사용된다. 그렇게 되면 커넥션 인스턴스(EJB object)는 영구적으로

런타임 메모리에서 제거된다.

*Value Description* millisecond 단위이다.

*Value Type* off-longType

*Value Type Description* 기본적으로 unsigned long 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

*Default Value* 3600000

(142) <jeus-ejb-dd> <beanlist> <jeus-bean> **<file-db-info>**

*Description* EJB 엔진은 상태유지 세션 빈이 비 활성화될 때 빈의 상태를 File DB 에 저장하고 후에 다시 필요할 때 다시 그 상태 값을 복구한다. 이때 사용하는 File DB 에 대한 설정을 한다.

*Child Elements*
(143)local-file-db?
(148)remote-file-db?

(143) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> **<local-file-db>**

*Description* 이 element 는 비 활성화되는 상태유지 세션 빈의 상태를 저장하는 로컬 파일 데이터베이스를 명시한다.

*Performance Recommendation* 클러스터링을 사용하지 않는다면 원격 File DB 대신 이 element 를 사용하는게 성능에 좋다.

*Child Elements*
(144)file-db-path?
(145)file-db-name?
(146)min-hole?
(147)packing-rate?

(144) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> <local-file-db> **<file-db-path>**

*Description* 이 element 는 상태유지 세션 빈의 상태를 저장할 File DB 를 생성할 디렉토리 경로를 명시한다.

*Value Description* 경로는 절대경로를 사용한다.

*Value Type* token

*Example* `<file-db-path>c:\temp</file-db-path>`

(145) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> <local-file-db> **<file-db-name>**

*Description* 여기에 File DB 의 이름을 명시한다. 이 이름 뒤에는 숫자가 덧붙여지고 .fdb 라는 확장자를 사용한다. 숫자는 실제 File DB 와 백업 File DB 를 구별하기 위해서 사용된다. 예를 들어 File DB 이름이 "teller"인 경우 "teller1.fdb"와 "teller2.fdb"가 생성된다.

*Value Description* 확장자가 없는 임의의 파일 이름.

*Value Type* token

*Example* `<file-db-name>teller</file-db-name>`

(146) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> <local-file-db> **<min-hole>**

*Description* File DB 에 있는 빈을 재활성화할 때 그 빈이 사용하는 File DB 에 "hole"이 생성된다. 너무 많은 "hole"이 있을 때 그 File DB 는 재구성될 수도 있다. min-hole 과 packing-rate 상태가 일치하면 File DB 는 재구성된다. 이 element 에 명시된 값은 재구성이 요구되는 최소한의 "hole" 개수를 의미한다.

*Value Description* "hole"의 최소개수.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 1000

*Example* `<min-hole>5000</min-hole>`

*Performance Recommendation* File DB 의 재구성은 시스템 자원의 낭비를 가져올 수 있다. 따라서 이 값은 최대한 높게 설정하라.

(147) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> <local-file-db> **<packing-rate>**

*Description* 이 값은 0 과 1 사이에 있는 값이다. min-hole 상태가 일치하고

"hole"에 의해 사용되는 File DB 의 총 비율이 이 값과 일치할 때 File DB 는 재 구성된다.

*Value Type* fractionType

*Value Type Description* 0 과 1 사이의 float 형으로 비율을 나타낸다.

*Default Value* 5.0E-1

*Example*
```
<packing-rate>0.4</packing-rate>
```

*Performance Recommendation* File DB 의 재구성은 시스템 자원의 낭비를 가져올 수 있다. 따라서 이 값은 최대한 높게 설정하라.

### (148) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> **<remote-file-db>**

*Description* 이 element 는 로컬 File DB 의 대안이다. 만약 이 값이 설정된다면 빈의 상태는 JEUSMain.xml 에 설정된 세션 매니저에 저장되고 JEUS 매니저에 의해 실행될 것이다. 상태유지 세션 빈이 클러스터링 되기를 원한다면 반드시 이 element 를 설정해야 한다. 클러스터링 되지 않은 환경에서는 이 값이 사용될 이유가 없다.

*Performance Recommendation* 성능상의 이유로 클러스터링을 하지 않는다면 이 값을 사용하지 않기를 권장한다.

*Child Elements*
```
(149)remote-primary-file-db
(150)remote-backup-file-db?
(151)conn-pool-size?
```

### (149) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> <remote-file-db> **<remote-primary-file-db>**

*Description* 이 element 는 상태 영속성을 위해 사용될 주 Session Manager 를 선택한다.

*Value Description* Session Manager 의 JNDI 이름.

*Value Type* token

*Example*
```
<remote-primary-file-db>MYSESSIONSERVER</remote-primary-file-db>
```

(150) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> <remote-file-db> **<remote-backup-file-db>**

*Description* 이 element 는 상태 영속성을 위해 사용될 백업 Session Manager 를 선택한다. Backup Session Manager 는 Primary Session Manager 가 다운 등의 이유로 사용이 불가능하게 되었을 때 사용한다.

*Value Description* Session Manager 의 JNDI 이름.

*Value Type* token

*Example*
```
<remote-backup-file-db>MYSESSIONBACKUP</remote-backup-file-db>
```

(151) <jeus-ejb-dd> <beanlist> <jeus-bean> <file-db-info> <remote-file-db> **<conn-pool-size>**

*Description* 원격 세션 서버와의 커넥션을 캐시할 때 사용하는 풀의 사이즈를 설정한다.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 20

*Example*
```
<conn-pool-size>50</conn-pool-size>
```

(152) <jeus-ejb-dd> <beanlist> <jeus-bean> **<persistence-optimize>**

*Description* 이 element 는 entity bean 에서 ejbLoad 와 ejbStore 메소드를 호출될 때를 결정하는 EJB 엔진 규칙을 결정하고 최적화 하기 위한 설정을 포함한다. 이 메소드들을 더 적게 호출될수록 성능은 보다 효율적이 된다.

*Performance Recommendation* 이 element 의 하위 element 들은 성능에 큰 영향을 끼치는 것이므로 정확히 설정되어야 한다.

*Child Elements*
```
(153)engine-type?
(154)non-modifying-method*
(158)entity-cache-size?
(159)update-delay-till-tx?
(160)include-update?
```

**(153) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> <engine-type>**

*Description*

이것은 EJB 엔진에 의해서 ejbLoad() 호출이 발생할 때 일어나는 행위를 결정하고 최적화시킨다.

*Value Type*

engineType

*Default Value*

EXCLUSIVE_ACCESS

*Defined Value*

EXCLUSIVE_ACCESS
각각의 Entity Bean 인스턴스는 그것이 나타내는 데이터베이스의 열과 일대일 관계를 가지고 그에 대해서 독점적인 접근을 한다. 이것은 하나의 인스턴스가 살아있는 동안 다른 Entity Bean 은 데이터에 접근하거나 변경할 수 없다는 사실을 의미하기 때문에 EJB 엔진은 ejbLoad() 호출을 모두 생략할 수 있다는 것을 나타낸다. 엔진은 빈이 생성될 때 한 번 ejbLoad() 호출을 실행하고 빈이 살아있는 동안 데이터베이스에 더 이상의 어떤 변화도 없다고 가정하고 ejbLoad() 호출을 하지 않는다. 이 빈은 이것이 클러스터링된 빈의 일부일 때 사용할 수 없다. 데이터베이스는 WAS 에 의해서만 변경이 허락된다는 점을 반드시 기억하기 바란다( WAS 가 아닌 접근은 허용되지 않는다).

SINGLE_OBJECT
element 의 값을 이것으로 설정한다면 다른 EJB 엔진의 여러 EJB 들이 같은 데이터베이스 열에 매핑될 수 있다. ejbLoad()는 Entity Bean 으로 요청이 들어오기 앞서서 항상 호출된다. SINGLE_OBJECT 와 MULTIPLE_OBJECT 는 같은 종류의 EJB 를 클러스터링할 때 필요하다. SINGLE_OBJECT 와 MULTIPLE_OBJECT 의 차이점은 SINGLE_OBJECT 인 경우 각각의 EJB 엔진 안에서 오직 하나의 EJB Entity Bean 이 EJB 클라이언트의 모든 요청을 처리한다. 즉 같은 EJB 엔진으로 다른 EJB 클라이언트의 요청이 도착한다면 먼저 연결을 맺고 있는 다른 클라이언트의 요청이 끝날 때까지 대기상태에 있어야 한다.

MULTIPLE_OBJECT
element 의 값을 이것으로 설정한다면 다른 EJB 엔진의 여러 EJB

들이 같은 데이터베이스 열에 매핑될 수 있다. ejbLoad()는 Entity Bean 으로 요청이 들어오기 앞서서 항상 호출된다. SINGLE_OBJECT 과 MULTIPLE_OBJECT 는 같은 종류의 EJB 를 클러스터링할 때 필요하다. SINGLE_OBJECT 와 MULTIPLE_OBJECT 의 차이점은 MULTIPLE_OBJECT 인 경우 각각의 EJB 엔진 안에서 모든 EJB 클라이언트의 요청을 동시에 처리할 여러 EJB Entity Bean 인스턴스가 생성된다.즉 SINGLE_OBJECT 모드와 달리 EJB 클라이언트의 요청은 먼저 처리하고 있는 다른 요청이 끝날 때까지 대기하지 않아도 된다.

*Example* `<engine-type>SINGLE_OBJECT</engine-type>`

*Performance Recommendation* 만약 빈이 클러스터링 되지 않고 데이터베이스의 열에 접속할 다른 요소가 없다면 항상 EXCLUSIVE_ACCESS 를 사용하라.

`(154) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> <non-modifying-method>`

*Description* Non-modifying 메소드란 빈과 연결된 데이터베이스에 어떤 변화도 주지 않는 메소드를 의미한다(예 : non-modifying methods = "getter"/"read only" 메소드 ). 빈의 모든 read-only 메소드는 ejbStore() 메소드 호출을 보다 효율적으로 사용하기 위해서 이 element 에 명시된다.

*Performance Recommendation* 최적의 성능을 위해서 모든 read-only 메소드는 이 리스트에 등록되는 것이 좋다.

*Child Elements*
`(155)method-name`
`(156)method-params?`

`(155) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> <non-modifying-method> <method-name>`

*Description* method 의 이름을 지정한다.

*Value Type* token

*Example* `foo`

`(156) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> <non-`

modifying-method> **<method-params>**

*Description* method 의 parameter 들을 순서대로 지정한다.

*Child Elements* (157)method-param*

(157) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> <non-modifying-method> <method-params> **<method-param>**

*Description* method 의 parameter 의 fully qualified class name 을 지정한다.

*Value Type* token

*Example* java.lang.String

(158) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> **<entity-cache-size>**

*Description* 이 element 는 내부 캐시 메모리 안에 남아있는 Entity Bean 인스턴스의 최대 개수를 명시한다. 최적의 성능을 위해 캐시 메모리 안에 비활성화된 빈 인스턴스를 가지고 있다.

*Value Description* 이 element 는 내부 캐시 메모리 안에 남아있는 Entity Bean 인스턴스의 최대 개수.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 2000

*Defined Value* 0
비활성화된 Entity Bean 에 대한 캐싱을 하지 않는다.

*Example* <entity-cache-size>100</entity-cache-size>

*Performance Recommendation* 많은 시스템 메모리 사용을 감수한다면 이 값은 충분히 높게 설정할 때 보다 최적의 성능을 기대할 수 있다. 반대로 시스템 메모리가 충분하지 못하다면 이 값을 낮게 설정하라.

(159) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> **<update-delay-till-tx>**

*Description* 이 옵션을 “false”로 설정했다면 생성된 EJB 데이터베이스의 삽입과 갱신이 EJB setXXX()가 호출될 때 실행이 된다. 일반적으로 이 옵션을 “true”로 설정을 하며, “true”일 경우에는 EJB 데이터의 삽입과 갱신은 트랜젝션이 커밋되었을 때 업데이트 된다. 개발자들이 원하는 사항은 다음과 같은 것들이 있다. 1. 트랜젝션을 시작한다. 2. EJB 데이터 갱신을 한다. 3. EJB 데이터를 읽어들인다. 4. 트렌젝션을 커밋한다. 설정된 값이 “false”이면, 스탭 3. 에서 읽힌 데이터는 스탭 2. 에서 기록된 데이터일 것이다. 설정된 값이 “true”이면, 스탭 3 가 아닌 스탭 4. 에서 커밋후 보여지는 값을 읽어 스탭 2. 에서 기록한 데이터이다. 이 옵션은 CMP2.0 에서만 사용된다.

*Value Type* boolean

*Default Value* true

*Example*
```
<update-delay-till-tx>true</update-delay-till-tx>
```

*Performance Recommendation* “false”로 세팅하면 EJB 의 insert, update 작업의 성능이 악영향을 미친다. 모든 insert, update 작업이 각각 실행되므로, 성능 저하를 가져온다. 성능은 위해서는 “true”로 사용하는 것이 좋다.

### (160) <jeus-ejb-dd> <beanlist> <jeus-bean> <persistence-optimize> <include-update>

*Description* <schema-info>의 각 <find-method>마다 설정되는 <include-updates>값의 default 값을 여기서 지정할 수 있다. 즉, <find-method>에 <include-update>가 지정되어 있지 않은 경우 여기에 지정된 값으로 설정된다. “true”일 경우 finder 메소드 호출되는 동안에 생성된 update 가 commit 되므로, finder 메소드가 실행될 동안 update 된 정보를 볼 수 있다.

*Value Type* boolean

*Default Value* false

*Example*
```
<include-updates>true</include-updates>
```

*Performance Recommendation* find method 가 자신이 수행되는 transaction 에서 변경된 내용을 반영해야 할 필요가 없다면 false 로 설정되어 있는게 성능에 도춤이

된다.

(161) <jeus-ejb-dd> <beanlist> <jeus-bean> **<schema-info>**

*Description* 이 element 는 데이터베이스의 컬럼과 EJB 필드의 매핑을 정의한다. 이 element 는 오직 CMP Entity Bean 에서만 사용한다.

*Child Elements*
(162)table-name?
(163)cm-field*
(168)creating-table?
(171)deleting-table?
(172)prim-key-field*
(177)find-method*
(184)jeus-query*
(191)db-vendor
(192)data-source-name
(193)auto-key-generator?

(162) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<table-name>**

*Description* EJB 와 매핑되는 관계형 데이터베이스의 테이블 이름. Default Value: EJB 모듈이름 + EJB 빈 이름의 마지막 15 문자.

*Value Type* token

*Example*
```
<table-name>ACCOUNT</table-name>
```

(163) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<cm-field>**

*Description* Container 가 관리하는 데이터베이스의 각 컬럼과 매핑하는 필드를 명시한다.

*Child Elements*
(164)field
(165)column-name?
(166)type?
(167)exclude-field?

(164) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <cm-field> **<field>**

*Description* 데이터베이스의 컬럼과 매핑할 EJB 필드의 이름.

*Value Type* token

*Example* `<field>id</field>`

(165) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <cm-field> **<column-name>**

*Description* <field> 태그에 명시된 EJB 필드와 매핑하게 될 데이터베이스 테이블의 컬럼이름을 명시한다.

*Value Type* token

*Example* `<column-name>ID</column-name>`

(166) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <cm-field> **<type>**

*Description* 이것은 데이터베이스 입장에서 테이블 컬럼의 데이터 타입을 의미한다 ( 예 : "VARCHAR(20)", "NUMERIC" ). 만약 이 element 가 명시되지 않는다면 기본값을 사용한다. 오라클 DB 의 경우에 "CLOB"과 "BLOB" 타입을 사용할 수도 있다. "CLOB" 타입은 EJB 의 java.lang.String 필드에 대응되고 "BLOB" 은 직렬화된 객체 필드에 대응된다.

*Value Type* token

*Example* `<type>NUMERIC</type>`

(167) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <cm-field> **<exclude-field>**

*Description* 이 element 를 "true"로 설정하면 위에서 지정된 필드에 대한 accessor method (set, get method)가 EJB server 가 생성하는 concrete bean class 에 생기지 않는다. 즉, 이 bean 을 사용하는 client 가 이 field 를 사용할수 없게 만든다. 이는 ejb-jar.xml 에서 지정된 cmp-field 를 무시하는 결과가 된다. 이 설정은 오직 CMP 2.0 EntityBean 에서만 작동된다.

*Value Type* boolean

*Default Value* false

*Example* `<exclude-field>true</exclude-field>`

(168) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<creating-table>**

*Description* 만약 이것을 활성화시키면 EJB 엔진이 부팅할 때 데이터베이스에 테이블이 없다면 생성한다. 만약 이 element 의 값이 "false" 라면 특별히 "schema check" 가 활성화된다. 이것은 EJB 엔진이 부팅할 때 작동하고 schema-info element 에 정의된 DB 스키마가 올바른지 검사한다. Engine Container 의 JVM 파라미터를 명시함으로써 이것을 Override 할 수도 있다( 예 : -Djeus.ejb.checktable=false" ) 하위 element 에 따라 table 을 생성하는 방식이 달라진다.

*Child Elements*
```
(169)use-existing-table?
(170)force-creating-table?
```

(169) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <creating-table> **<use-existing-table>**

*Description* 만약 DB 에 지정된 이름의 table 이 이미 존재한다면 이를 그대로 사용하고 없는 경우에만 table 을 생성한다.

(170) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <creating-table> **<force-creating-table>**

*Description* 만약 DB 에 지정된 이름의 table 이 이미 존재한다면 이를 지우고 다시 table 을 생성한다.

(171) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<deleting-table>**

*Description* 이 것을 활성화시키면 EJB 엔진이 종료될 때 명명된 데이터베이스의 테이블을 삭제한다. 의도하지 않은 DB table 삭제를 막기 위해 system property 가 설정되어 있고 <creating-table> 설정이 존재해야 table 을 지운다.

*Value Type* boolean

*Default Value* false

*Example*
```
<deleting-table>true</deleting-table>
```

(172) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<prim-key-field>**

*Description* ejb-jar.xml 에 <prim-key-class>가 명시되어 있을 경우에만 사용된다. <prim-key-class>의 field 중 EJB 빈과 DB 테이블의 열에서 Primary Key 를 구성하기 위해 사용되는 모든 EJB 필드 이름을 명시한다.하위 element 중 <field>만 지정하면 된다. Default Value: 만약 이 값이 명시되지 않는다면<prim-key-class>의 모든 public 필드를 가지고 Primary Key 를 구성한다.

*Example*
```
<prim-key-field>id</prim-key-field>
```

*Child Elements*
```
(173)field
(174)column-name?
(175)type?
(176)exclude-field?
```

**(173) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <prim-key-field> <field>**

*Description* 데이터베이스의 컬럼과 매핑할 EJB 필드의 이름.

*Value Type* token

*Example*
```
<field>id</field>
```

**(174) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <prim-key-field> <column-name>**

*Description* <field> 태그에 명시된 EJB 필드와 매핑하게 될 데이터베이스 테이블의 컬럼이름을 명시한다.

*Value Type* token

*Example*
```
<column-name>ID</column-name>
```

**(175) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <prim-key-field> <type>**

*Description* 이것은 데이터베이스 입장에서 테이블 컬럼의 데이터 타입을 의미한다 ( 예 : "VARCHAR(20)", "NUMERIC" ). 만약 이 element 가 명시되지 않는다면 기본값을 사용한다. 오라클 DB 의 경우에 "CLOB"과 "BLOB" 타입을 사용할 수도 있다. "CLOB" 타입은 EJB 의 java.lang.String 필드에 대응되고 "BLOB" 은 직렬화된 객체 필드에

대응된다.

*Value Type* token

*Example*
```
<type>NUMERIC</type>
```

(176) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <prim-key-field> **<exclude-field>**

*Description* 이 element 를 "true"로 설정하면 위에서 지정된 필드에 대한 accessor method (set, get method)가 EJB server 가 생성하는 concrete bean class 에 생기지 않는다. 즉, 이 bean 을 사용하는 client 가 이 field 를 사용할수 없게 만든다. 이는 ejb-jar.xml 에서 지정된 cmp-field 를 무시하는 결과가 된다. 이 설정은 오직 CMP 2.0 EntityBean 에서만 작동된다.

*Value Type* boolean

*Default Value* false

*Example*
```
<exclude-field>true</exclude-field>
```

(177) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<find-method>**

*Description* CMP 1.1 Entity Bean 의 경우에 finder 메소드에 대해서 반드시 필요한 SQL 문장을 명시해야 한다.CMP 2.0 의 경우에는 ejb-jar.xml 에 지정된 EJB-QL 을 overriding 할 수 있다. 적용되는건 jeus-query 와 같다.

*Child Elements*
```
(178)query-method
(182)sql?
(183)include-updates?
```

(178) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <find-method> **<query-method>**

*Description* Find 메소드의 이름과 파라미터.

*Child Elements*
```
(179)method-name
(180)method-params?
```

(179) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <find-method> <query-method> **<method-name>**

*Description* method 의 이름을 지정한다.

*Value Type* token

*Example* `foo`

```
(180) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <find-method>
<query-method> <method-params>
```

*Description* method 의 parameter 들을 순서대로 지정한다.

*Child Elements* `(181)method-param*`

```
(181) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <find-method>
<query-method> <method-params> <method-param>
```

*Description* method 의 parameter 의 fully qualified class name 을 지정한다.

*Value Type* token

*Example* `java.lang.String`

```
(182) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <find-method>
<sql>
```

*Description* EJB 1.1 에서 finder 메소드를 생성할 때 사용하는 SQL 문장의 일부분. EJB 2.0 에서는 이 element 를 사용하지 않는다. 특수한 문자( “<” 같은)를 사용할 때는 <![CDATA[sql]]> 를 사용해야 한다.

*Value Description* 이것은 단지 where 절의 키워드를 명시한다. 이 SQL 문장에서 "?"는 finder 메소드가 호출될 때 finder 메소드의 파라미터의 값으로 순서대로 대체된다. ?뒤에 숫자를 넣을수도 있는데 이는 몇번째 파라미터인지를 나타낸다. 즉, "?1"은 첫번째 파라미터의 값이 들어가는 자리이다. 이때 ? 뒤에 숫자를 넣는 형식과 ?만 사용하는 형식은 혼용될 수 없다.

*Value Type* token

*Example* `<sql>customer_address=?</sql>`

```
(183) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <find-method>
```

**<include-updates>**

*Description* "true"일 경우 finder 메소드 호출되는 동안에 생성된 update 가 commit 되므로, finder 메소드가 실행될 동안 update 된 정보를 볼 수 있다. Default Value: <persistence-optimize>의 <include-update>에 지정된 값이 default 값이다.

*Value Type* boolean

*Example* `<include-updates>true</include-updates>`

(184) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<jeus-query>**

*Child Elements*
(185)query-method
(189)sql?
(190)include-updates?

(185) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <jeus-query> **<query-method>**

*Description* Find 메소드의 이름과 파라미터.

*Child Elements*
(186)method-name
(187)method-params?

(186) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <jeus-query> <query-method> **<method-name>**

*Description* method 의 이름을 지정한다.

*Value Type* token

*Example* `foo`

(187) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <jeus-query> <query-method> **<method-params>**

*Description* method 의 parameter 들을 순서대로 지정한다.

*Child Elements* (188)method-param*

(188) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <jeus-query> <query-method> <method-params> **<method-param>**

*Description* method 의 parameter 의 fully qualified class name 을 지정한다.

*Value Type* token

*Example* `java.lang.String`

(189) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <jeus-query> **<sql>**

*Description* EJB 1.1 에서 finder 메소드를 생성할 때 사용하는 SQL 문장의 일부분. EJB 2.0 에서는 이 element 를 사용하지 않는다. 특수한 문자( “<” 같은)를 사용할 때는 <![CDATA[sql]]> 를 사용해야 한다.

*Value Description* 이것은 단지 where 절의 키워드를 명시한다. 이 SQL 문장에서 "?"는 finder 메소드가 호출될 때 finder 메소드의 파라미터의 값으로 순서대로 대체된다. ?뒤에 숫자를 넣을수도 있는데 이는 몇번째 파라미터인지를 나타낸다. 즉, "?1"은 첫번째 파라미터의 값이 들어가는 자리이다. 이때 ? 뒤에 숫자를 넣는 형식과 ?만 사용하는 형식은 혼용될 수 없다.

*Value Type* token

*Example* `<sql>customer_address=?</sql>`

(190) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <jeus-query> **<include-updates>**

*Description* “true”일 경우 finder 메소드 호출되는 동안에 생성된 update 가 commit 되므로, finder 메소드가 실행될 동안 update 된 정보를 볼 수 있다. Default Value: <persistence-optimize>의 <include-update>에 지정된 값이 default 값이다.

*Value Type* boolean

*Example* `<include-updates>true</include-updates>`

(191) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<db-vendor>**

*Description* jeus-query 태그는 query 메소드(findXXX)에서 EJB-QL 과 JEUS EJB-QL 확장을 사용할 수 있도록 한다. 이것은 ejb-jar.xml 의 query 태그와

비슷하다. 이 태그의 주 목적은 BEA WebLogic 어플리케이션 서버를 JEUS 4.2 로 마이그레이션할 때 용이하게 하기 위함이다. 적용되는건 find-method 와 같다.

*Value Type* db-vendorType

*Defined Value*

oracle
Oracle DBMS 인 경우

informix
Informix DBMS 인 경우

db2
IBM DB2 DBMS 인 경우

mssql
Microsoft SQL Server DBMS 인 경우

sybase
Sybase DBMS 인 경우

hsql
HSQL DBMS 인 경우

cloudscape
Cloudscape DBMS 인 경우

### (192) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<data-source-name>**

*Description* 데이터베이스와 연결할 때 사용하는 데이터베이스 커넥션 풀의 JNDI 이름. 이 커넥션 풀은 일반적으로 JEUSMain.xml 에 설정되고 JEUS 매니저 JVM 에 의해 실행된다.

*Value Description* JEUS DB 커넥션 풀의 JNDI 이름.

*Value Type* token

*Example*

```
<data-source-name>MYDB</data-source-name>
```

(193) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> **<auto-key-generator>**

*Description* 이 element 는 Primary Key 를 설정하지 않고 create()를 호출할 때 Primary Key 를 자동으로 생성해 주는 외부 소스를 지정한다. 이 외부 소스는 단일한 Primary Key 를 생성할 필요가 있는 여러 EJB 엔진이 공유하는 하나의 데이터베이스이어야만 한다. EJB 엔진이 데이터베이스로부터 Primary Key 를 가져오고 난 뒤 DB 에 있는 Primary Key 의 값은 항상 유일하고 다른 EJB 엔진에 의해 사용될 수 있도록 값이 증가된다. Primary Key 값은 데이터베이스 안에서 항상 "int" 형이며 빈 안에서는 반드시 java.lang.Integer 타입이어야만 한다.

*Child Elements*
```
(194)generator-type
(195)generator-name?
(196)sequence-column?
(197)key-cache-size?
```

(194) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <auto-key-generator> **<generator-type>**

*Description* 이 element 는 Primary Key 를 가지고 있는 데이터베이스의 타입과 벤더를 명시한다.

*Value Type* generator-typeType

*Defined Value*

ORACLE
Oracle DB 인 경우

MSSQL
MS SQL DB 인 경우

USER_KEY_TABLE
Oracle 과 MSSQL 을 제외한 다른 DB 인 경우

*Example*
```
<generator-type>USER_KEY_TABLE</generator-type>
```

(195) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <auto-key-generator> **<generator-name>**

*Description* Oracle 과 USER_KEY_TABLE 값을 사용하는 경우 사용한다.

| | |
|---|---|
| | Oracle 인 경우 SEQUENCE 이름을 명시한다. USER_KEY_TABLE 인 경우 Primary Key 를 가지고 있는 테이블 이름을 명시한다. |
| *Value Description* | Oracle 인 경우 SEQUENCE 이름, USER_KEY_TABLE 인 경우 테이블 이름. |
| *Value Type* | token |
| *Example* | `<generator-name>MYKEYTABLE</generator-name>` |

`(196) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <auto-key-generator>` **`<sequence-column>`**

| | |
|---|---|
| *Description* | USER_KEY_TABLE 인 경우에만 사용한다. 이 element 는 Primary Key 를 가지고 있는 컬럼 이름을 명시한다. |
| *Value Description* | 컬럼 이름. |
| *Value Type* | token |
| *Example* | `<sequence-column>PRIMARYKEYCOLUMN</sequence-column>` |

`(197) <jeus-ejb-dd> <beanlist> <jeus-bean> <schema-info> <auto-key-generator>` **`<key-cache-size>`**

| | |
|---|---|
| *Description* | 이 element 는 EJB 엔진에 할당될 유일한 키의 개수를 설정한다. 이것은 Primary Key 데이터베이스로부터 키를 넘겨받은 뒤 그 Primary Key 의 값을 여기에 선언된 값만큼 증가시킨다. 이것은 그만큼 Primary Key 에 접속할 필요가 없어지므로 성능 향상에 도움을 준다. |
| *Value Description* | 로컬 EJB 엔진에 할당될 Primary Key 의 개수. Oracle 데이터베이스인 경우 이 값은 반드시 SEQUENCE 의 SEQUENCE INCREMENT 값과 일치해야 한다 |
| *Value Type* | nonNegativeIntType |
| *Value Type Description* | 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다. |
| *Default Value* | 1 |

*Example* `<key-cache-size>20</key-cache-size>`

*Performance Recommendation* 성능 최적화를 위해 이 값을 적당히 올린다.

(198) <jeus-ejb-dd> <beanlist> <jeus-bean> **<database-insert-delay>**

*Description* 이 옵션은 EJB 가 생성될 때, 언제 새로운 EJB 데이터가 DB 에 저장될지를 결정한다. 현재는 두 가지 방법이 있다. ejbCreate() 메소드 완료 후와 ejbPostCreate() 메소드 완료 후가 있다.

*Value Type* database-insert-delayType

*Default Value* ejbPostCreate

*Defined Value* ejbCreate
ejbCreate 메소드 후에 EJB 데이터를 DB table 에 추가한다.

ejbPostCreate
ejbPostCreate 메소드 후에 EJB 데이터를 DB table 에 추가한다.

*Example* `<database-insert-delay>ejbCreate</database-insert-delay>`

*Performance Recommendation* “false”로 세팅하면 EJB 의 insert, update 작업의 성능이 악영향을 미친다. 모든 insert, update 작업이 각각 실행되므로, 성능 저하를 가져온다. 성능은 위해서는 “true”로 사용하는 것이 좋다.

(199) <jeus-ejb-dd> <beanlist> <jeus-bean> **<cm-persistence-optimize>**

*Description* 이 element 의 사용은 EJB 엔진에 의해 생성되는 ejbLoad()와 ejbStore() 메소드의 성능을 향상시키기 위해서 사용하는 어떤 속성들을 정의한다. 이 element 는 오직 CMP Entity Bean 에서만 사용된다.

*Child Elements*
(200)subengine-type?
(201)fetch-size?
(202)init-caching?

(200) <jeus-ejb-dd> <beanlist> <jeus-bean> <cm-persistence-optimize> **<subengine-type>**

| | |
|---|---|
| *Description* | 이 element 는 데이터베이스 테이블의 열에 접근할 때 ejbLoad() 메소드가 가지는 데이터 베이스 락(lock)의 타입을 선언한다. 이 설정은 빈의 성격에 따라 설정될 수 있다( 예: 빈이 쓰기보다 읽기 작업을 더 많이 수행하는 경우 아니면 그 반대로 읽기보다 쓰기 작업을 더 많이 수행하는 경우 ). Oracle DB 에 대해서는 WriteLock 을 사용하는 경우 DB Isolation 이 Serialization 으로 설정되는 효과를 얻을 수 있다. |
| *Value Type* | subengine-typeType |
| *Default Value* | ReadLocking |
| *Defined Value* | ReadLocking<br>생성된 ejbLoad()는 데이터베이스 테이블의 열에 대해서 항상 "shared lock"을 가진다.<br><br>WriteLocking<br>생성된 ejbLoad()는 데이터베이스 테이블의 열에 대해서 항상 "exclusive lock"을 가진다.<br><br>WriteLockingFind<br>생성된 ejbLoad()와 ejbFind() 는 데이터베이스 테이블의 열에 대해서 항상 "exclusive lock"을 가진다. |
| *Example* | `<subengine-type>WriteLocking</subengine-type>` |
| *Performance Recommendation* | 만약 EJB 빈이 DB 테이블의 열에 대해서 쓰기보다 읽기 작업이 더 많다면 "ReadLocking"을 사용한다. 반대로 읽기보다 쓰기 작업이 더 많다면 "WriteLocking"을 사용한다. |

### (201) <jeus-ejb-dd> <beanlist> <jeus-bean> <cm-persistence-optimize> <fetch-size>

| | |
|---|---|
| *Description* | 이 element 는 DB 에서 매우 큰 사이즈의 ResultSet 를 리턴 받을 때 한번에 얼마나 많은 열을 가지고 올 것인지를 명시한다. |
| *Value Type* | nonNegativeIntType |
| *Value Type Description* | 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다. |

*Default Value* 10

*Example*
```
<fetch-size>80</fetch-size>
```

*Performance Recommendation* 이 값을 높게 설정하면 시스템 메모리가 보다 많이 필요하겠지만 데이터베이스 "select" 요청을 보다 효율적으로 실행한다.

**(202) <jeus-ejb-dd> <beanlist> <jeus-bean> <cm-persistence-optimize> <init-caching>**

*Description* 이 element 를 "true"로 설정하면 EJB 엔진은 매핑하는 DB 테이블의 열에 대해서 EJB Entity Bean 을 미리 초기화한다. 이 작업은 엔진이 부팅될 때 실행된다. 이 element 를 "false"로 설정하면 EJB 인스턴스는 create(), findByPrimaryKey() 또는 이와 같은 기능을 하는 홈 인터페이스의 메소드를 통해서 생성된다.

*Value Type* boolean

*Default Value* false

*Example*
```
<init-caching>true</init-caching>
```

*Performance Recommendation* 이 값을 "true"로 설정하면 보다 많은 시스템 메모리가 필요하지만 전체적인 성능은 향상된다. 시스템 메모리가 부족하다고 판단되면 이 값은 "false"로 설정한다.

**(203) <jeus-ejb-dd> <beanlist> <jeus-bean> <enable-instant-ql>**

*Description* 이 값을 "true"로 설정 한다면 이 빈의 홈 인터페이스는 부가적으로 JEUS 특정적인 인터페이스 ( "jeus.ejb.bean.objectbase.EJBInstanceFinder" )를 구현한다. 이 인터페이스는 다음과 같은 메소드를 포함한다. findWithInstantQL(java.lang.String qlSentence). 이 메소드는 클라이언트의 소스 코드에서 바로 임의의 EJB QL 쿼리를 명시할 수 있도록 해준다. 이것은 finder 메소드가 충분하지 않을 경우 임시적인 해결책이 될 수 있다. 참고: "qlSentence" 문자열 파라미터는 오직 파라미터가 없는, 즉 "?" 이 없는 EJB QL 문장만을 지원한다는 사실을 유의하라. 이것은 오직 CMP 2.0 Entity Bean 에서만 작동한다.

*Value Type* boolean

*Default Value* false

*Example*
```
<enable-instant-ql>true</enable-instant-ql>
```

*Performance Recommendation* "findWithInstantQL(java.lang.String qlSentence)" 메소드 사용은 비효율적이다. 아주 특별한 상황이 아니라면 사용하지 않는 것이 좋다.

(204) <jeus-ejb-dd> <beanlist> <jeus-bean> **<connection-factory-name>**

*Description* EJB 2.0 style 의 Message-Driven Bean 에서만 사용하는 element 로 이 MDB 가 사용할 JMS connection factory 의 JNDI 이름을 설정한다.

*Value Type* token

(205) <jeus-ejb-dd> <beanlist> <jeus-bean> **<mdb-resource-adapter>**

*Description* Connector 와 연동되는 Message-Driven Bean 에서만 사용하는 element 로 이 MDB 가 사용할 resource adapter 를 설정한다.

*Child Elements*
```
(206)resource-adapter-name
(207)activation-config?
```

(206) <jeus-ejb-dd> <beanlist> <jeus-bean> <mdb-resource-adapter> **<resource-adapter-name>**

*Description* MDB 가 사용할 resource adapter 의 이름을 설정한다. 이 이름은 해당 resource adapter 의 jeus-connector-dd.xml 에 지정되어 있는 module name 이다.

*Value Type* token

(207) <jeus-ejb-dd> <beanlist> <jeus-bean> <mdb-resource-adapter> **<activation-config>**

*Description* resource adapter 를 설정할 activation config 를 기록한다. 이 설정은 ejb-jar.xml 의 activation config 를 override 할 수 있다.

*Child Elements*
```
(208)description*
(209)activation-config-property+
```

(208) <jeus-ejb-dd> <beanlist> <jeus-bean> <mdb-resource-adapter> <activation-config> **<description>**

*Description* 이 activation config 에 대한 설명을 적을 수 있다.

*Value Type* string

```
(209) <jeus-ejb-dd> <beanlist> <jeus-bean> <mdb-resource-adapter>
<activation-config> <activation-config-property>
```

*Description* 각 activation config property 를 지정한다.

*Child Elements* (210)activation-config-property-name
(211)activation-config-property-value

```
(210) <jeus-ejb-dd> <beanlist> <jeus-bean> <mdb-resource-adapter>
<activation-config> <activation-config-property> <activation-config-
property-name>
```

*Description* 지정하고자 하는 activation config property 의 이름이다.JMS MDB 의 경우에는 acknowledgeMode, messageSelector, destinationType, subscriptionDurability 의 기본적으로 인식된다.

*Value Type* token

```
(211) <jeus-ejb-dd> <beanlist> <jeus-bean> <mdb-resource-adapter>
<activation-config> <activation-config-property> <activation-config-
property-value>
```

*Description* 지정하고자 하는 activation config property 의 값을 설정한다.

*Value Type* token

```
(212) <jeus-ejb-dd> <beanlist> <jeus-bean> <destination>
```

*Description* EJB 2.0 style 의 Message-Driven Bean 에서만 사용하는 element 로 이 MDB 가 사용할 JMS Destination 의 JNDI 이름을 설정한다. 자세한 것은 JEUS JMS 메뉴얼과 JMS spec 을 참고하기 바란다.

*Value Type* token

```
(213) <jeus-ejb-dd> <beanlist> <jeus-bean> <max-message>
```

*Description* EJB 2.0 style 의 Message-Driven Bean 에서만 사용하는 element 로 이 MDB 가 사용하는 JMS Session 에 주어지는 최대 message 개수를 지정한다. 자세한 것은 JEUS JMS 메뉴얼과 JMS spec 을 참고하기

바란다.

*Value Type* nonNegativeIntType

*Value Type Description* 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

*Default Value* 10

(214) <jeus-ejb-dd> <beanlist> <jeus-bean> **<ack-mode>**

*Description* EJB 2.0 style 의 Message-Driven Bean 에서만 사용하는 element 로 이 MDB 가 사용하는 JMS Session 의 Ackndowledge mode 를 설정한다. 자세한 것은 JEUS JMS 메뉴얼과 JMS spec 을 참고하기 바란다.

*Value Type* ack-modeType

*Defined Value* Auto-acknowledge

Dups-ok-acknowledge

(215) <jeus-ejb-dd> <beanlist> <jeus-bean> **<durable>**

*Description* EJB 2.0 style 의 Message-Driven Bean 에서만 사용하는 element 로 이 MDB 를 JMS 의 durable subscriber 로 지정할지를 결정한다.자세한 것은 JEUS JMS 메뉴얼과 JMS spec 을 참고하기 바란다.

*Value Type* durableType

*Defined Value* Durable

NonDurable

(216) <jeus-ejb-dd> <beanlist> <jeus-bean> **<msg-selector>**

*Description* EJB 2.0 style 의 Message-Driven Bean 에서만 사용하는 element 로 이 MDB 가 사용할 message selector 를 설정한다.자세한 것은 JEUS JMS 메뉴얼과 JMS spec 을 참고하기 바란다.

*Value Type* token

(217) <jeus-ejb-dd> <beanlist> <jeus-bean> **<jndi-spi>**

*Description* EJB 2.0 style 의 Message-Driven Bean 에서만 사용하는 element 로 만약 MDB 가 기본값(jeus.jndi.JEUSContextFactory) 이 아닌 다른 JNDI 이름 서비스에 등록되어 있는 JMS 서비스를 사용한다면 이 element 를 사용한다. 이 element 는 JEUS MDB 를 IBM MQ 나 SONIC MQ 같은 JEUS JMS 서비스 이외의 것과 연결할 때 사용한다.

*Child Elements*
```
(218)mq-vendor
(219)initial-context-factory
(220)provider-url?
```

### (218) <jeus-ejb-dd> <beanlist> <jeus-bean> <jndi-spi> **<mq-vendor>**

*Description* 다음 element 에서 설정할 JNDI 이름 서비스를 통해서 MDB 과 연결을 맺을 MQ/JMS 벤더의 이름.

*Value Type* mq-vendorType

*Defined Value*

SONICMQ
Sonic MQ 를 사용하는 경우

IBMMQ
IBM MQ 를 사용하는 경우

### (219) <jeus-ejb-dd> <beanlist> <jeus-bean> <jndi-spi> **<initial-context-factory>**

*Description* JMS 서비스와 연결할 때 JEUS Naming Service 를 사용할 때 필요한 initial context factory 의 클래스 이름.

*Value Type* token

*Example*
```
<initial-context-
factory>acme.jndi.ACMEContextFactory</initial-
context-factory>
```

### (220) <jeus-ejb-dd> <beanlist> <jeus-bean> <jndi-spi> **<provider-url>**

*Description* JNDI 이름 서비스와 연결할 때 사용하는 URL 주소와 포트번호.

*Value Type* anyURI

*Example*
```
<provider-url>
```

```
protocol://localhost:2345</provider-url>
```

(221) <jeus-ejb-dd> <beanlist> <jeus-bean> **<durable-timer-service>**

*Description* 이 EJB 가 Timer Service 를 사용하는 경우 persistence timer 에 대한 동작을 설정한다.

*Child Elements*
```
(222)enable-durable-timers?
(223)ignore-durable-timers-at-deploy?
(224)delete-durable-timers-at-undeploy?
```

(222) <jeus-ejb-dd> <beanlist> <jeus-bean> <durable-timer-service> **<enable-durable-timers>**

*Description* 이 EJB 가 timer 를 persistence 하게 관리하게 할것인지를 결정한다. 이 값이 true 라고 하더라도 EJBMain.xml 에 durable timer 설정이 되어 있지 않으면 persistence timer 를 사용할 수가 없다.

*Value Type* boolean

*Default Value* true

(223) <jeus-ejb-dd> <beanlist> <jeus-bean> <durable-timer-service> **<ignore-durable-timers-at-deploy>**

*Description* 이 EJB 가 deploy 될 때에 이전에 제거되지 않은 persistence timer 를 무시할지의 여부를 지정한다. 만약 true 로 무시되면 이 timer 들은 deploy 시에 제거된다.

*Value Type* boolean

*Default Value* false

(224) <jeus-ejb-dd> <beanlist> <jeus-bean> <durable-timer-service> **<delete-durable-timers-at-undeploy>**

*Description* timer 는 code 상에서 명시적으로 remove()가 호출되지 않으면 일반적으로 제거되지 않는다. 이 EJB 가 undeploy 될 때에 아직 제거되지 않은 persistence timer 가 있다면 이를 제거하도록 하는 option 이다.

*Value Type* boolean

*Default Value* false

(225) <jeus-ejb-dd> **<ejb-relation-map>**

*Description* 이 element 는 CMP 2.0 Entity Bean 간의 Relation 을 정의할 때 사용한다. ejb-relation-map element 는 ejb-jar.xml 에 선언된 각각의 Relation 마다 하나씩 존재한다.

*Child Elements*
```
(226)relation-name
(227)table-name?
(228)jeus-relationship-role*
```

(226) <jeus-ejb-dd> <ejb-relation-map> **<relation-name>**

*Description* 이 element 는 표준 ejb-jar.xml 에 정의된 EJB 2.0 Relation 의 이름을 명시한다.

*Value Type* token

*Example*
```
<relation-name>student-course</relation-name>
```

(227) <jeus-ejb-dd> <ejb-relation-map> **<table-name>**

*Description* EJB Relation 이 다 대 다( M:M ) 관계라면 이 데이터 베이스 안에서 이 다 대 다( M:M ) Relation 을 표현하는 "join-table"이라는 것이 있다. 이 element 는 다 대 다( M:M ) Relation 을 표현하는 "join-table"의 이름을 명시한다.

*Value Type* token

*Example*
```
<table-name>studentcoursejoin</table-name>
```

(228) <jeus-ejb-dd> <ejb-relation-map> **<jeus-relationship-role>**

*Description* 이 element 는 하나의 EJB 와 다른 EJB 사이에 Relation 을 명시한다. 각각의 element 는 단방향으로 Relation 을 의미한다. 다대다 Relation ( M:M )인 경우 반드시 두 개의 jeus-relationship-role element 가 필요하다. 각각의 element 는 "join-table"의 각각의 Foreign Key 와 EJB 의 실제 Primary Key 를 매핑된다.

*Child Elements*
```
(229)relationship-role-name
(230)column-map*
```

(229) <jeus-ejb-dd> <ejb-relation-map> <jeus-relationship-role> **<relationship-role-name>**

*Description* 이것은 ejb-jar.xml 파일의 ejb-relationship-role-name element 에 정의된 relationship role 의 이름이다.

*Value Type* token

*Example*
```
<relationship-role-name>student-to-
course</relationship-role-name>
```

(230) <jeus-ejb-dd> <ejb-relation-map> <jeus-relationship-role> **<column-map>**

*Description* 이 element 는 하나의 테이블의 Foreign Key 를 다른 EJB 의 Primary Key 와 매핑한다. 이 매핑은 단방향으로 Relation 을 정의한다. 만약 Foreign Key 나 Primary Key 가 여러 컬럼이나 EJB 필드를 복합하여 사용되었을 경우 이것들은 모두 column-map element 에 정의해야 한다.

*Child Elements*
```
(231)foreign-key-column
(232)target-primary-key-column
```

(231) <jeus-ejb-dd> <ejb-relation-map> <jeus-relationship-role> <column-map> **<foreign-key-column>**

*Description* 이 <column-map>이 선언된 EJB 의 DB table 에 존재하는 Foreign Key DB Column 의 이름이다. 이 키는 target-primary-key-column element 에 정의된 상대 EJB 의 Primary Key DB Column 의 값이 매핑된다. 만약 many-to-many relation 인 경우는 relation table 에 존재하는 foreign key 컬럼의 이름이다.

*Value Type* token

(232) <jeus-ejb-dd> <ejb-relation-map> <jeus-relationship-role> <column-map> **<target-primary-key-column>**

*Description* Foreign Key 컬럼에 매핑될 상대 EJB 의 Primary Key 의 DB Column 이름이다.

*Value Type* token

(233) <jeus-ejb-dd> **<message-destination>**

*Description* ejb-jar.xml 의 <message-destination>에 선언된 message destination 와 JNDI 에 등록된 실제 Destination 객체를 매핑한다.

*Child Elements* (234)jndi-info*

(234) <jeus-ejb-dd> <message-destination> **<jndi-info>**

*Description* 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다. 예를 들면 실제 JNDI 이름이 "ACCEJB"인 account EJB 를 코드상에서 "ejb/account"으로 lookup 할 수 있다.

*Child Elements* (235)ref-name
(236)export-name

(235) <jeus-ejb-dd> <message-destination> <jndi-info> **<ref-name>**

*Description* 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수 있다.

*Value Description* 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

*Value Type* token

*Example* <ref-name>ejb/AccountEJB</ref-name>

(236) <jeus-ejb-dd> <message-destination> <jndi-info> **<export-name>**

*Description* JEUS DD 에 정의된 실제 JNDI 이름.

*Value Type* token

*Example* <export-name>ACCEJB</export-name>

# E.4 Sample jeus-ejb-dd.xml File

*<<jeus-ejb-ddxml>>*

```
<jeus-ejb-dd>
    <module-info>
        <role-permission>
```

```
            <role>manager</role>
            <principal>peter</principal>
        </role-permission>
    </module-info>
    <beanlist>
        <jeus-bean>
            <ejb-name>teller</ejb-name>
            <export-name>TELLEREJB</export-name>
            <local-export-name>LOCALTELLEREJB</local-export-name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            <single-vm-only>true</single-vm-only>
            <local-invoke-optimize>true</local-invoke-optimize>
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            <security-interop>
                <integrity-confidentiality>Requires</integrity-
confidentiality>
                <trust-in-client>Requires</trust-in-client>
                <client-auth>Requires</client-auth>
                <identity-assertion>Supports</identity-assertion>
            </security-interop>
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>ejb/AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
            </ejb-ref>
            <res-ref>
                <jndi-info>
                    <ref-name>jdbc/AccountDB</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
```

```
            </res-ref>
            <res-env-ref>
                <jndi-info>
                    <ref-name>jms/StockQueue</ref-name>
                    <export-name>STOCKQUEUE</export-name>
                </jndi-info>
            </res-env-ref>
            <thread-max>200</thread-max>
            <clustering>
                <home-clustered>true</home-clustered>
                <ejb-home-failover>true</ejb-home-failover>
                <ejb-home-idempotent-
method>foo(java.lang.String,int)</ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-
method>foo(java.lang.String,int)</ejb-remote-idempotent-method>
            </clustering>
            <invoke-http>
                <url>/mycontext/RMIHandlerServlet</url>
                <http-port>80</http-port>
            </invoke-http>
        </jeus-bean>
        <jeus-bean>
            <ejb-name>shoppingcart</ejb-name>
            <export-name>SHOPPINGCARTEJB</export-name>
            <local-export-name>LOCALSHOPPINGCARTEJB</local-
export-name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            <single-vm-only>true</single-vm-only>
            <local-invoke-optimize>true</local-invoke-optimize>
            <pooling-bean>true</pooling-bean>
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            <security-interop>
                <integrity-confidentiality>Requires</integrity-
confidentiality>
                <trust-in-client>Requires</trust-in-client>
```

```
                <client-auth>Requires</client-auth>
                <identity-assertion>Supports</identity-assertion>
            </security-interop>
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>ejb/AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
            </ejb-ref>
            <res-ref>
                <jndi-info>
                    <ref-name>jdbc/AccountDB</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
            </res-ref>
            <res-env-ref>
                <jndi-info>
                    <ref-name>jms/StockQueue</ref-name>
                    <export-name>STOCKQUEUE</export-name>
                </jndi-info>
            </res-env-ref>
            <thread-max>200</thread-max>
            <clustering>
                <home-clustered>true</home-clustered>
                <ejb-home-failover>true</ejb-home-failover>
                <ejb-home-idempotent-
method>foo(java.lang.String,int)</ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-
method>foo(java.lang.String,int)</ejb-remote-idempotent-method>
            </clustering>
            <invoke-http>
                <url>/mycontext/RMIHandlerServlet</url>
                <http-port>80</http-port>
```

```
            </invoke-http>
            <object-management>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                    <resizing-period>1800000</resizing-period>
                </bean-pool>
                <connect-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </connect-pool>
                <capacity>2000</capacity>
                <passivation-timeout>10000</passivation-timeout>
                <disconnect-timeout>180000</disconnect-timeout>
            </object-management>
            <file-db-info>
                <local-file-db>
                    <file-db-path>c:\temp</file-db-path>
                    <file-db-name>teller</file-db-name>
                    <min-hole>5000</min-hole>
                    <packing-rate>0.4</packing-rate>
                </local-file-db>
                <remote-file-db>
                    <remote-primary-file-
db>MYSESSIONSERVER</remote-primary-file-db>
                    <remote-backup-file-
db>MYSESSIONBACKUP</remote-backup-file-db>
                    <conn-pool-size>50</conn-pool-size>
                </remote-file-db>
            </file-db-info>
        </jeus-bean>
        <jeus-bean>
            <ejb-name>account</ejb-name>
            <export-name>ACCOUNTEJB</export-name>
            <local-export-name>LOCALACCOUNTEJB</local-export-
name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            <single-vm-only>true</single-vm-only>
```

```
            <local-invoke-optimize>true</local-invoke-optimize>
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            <security-interop>
                <integrity-confidentiality>Requires</integrity-
confidentiality>
                <trust-in-client>Requires</trust-in-client>
                <client-auth>Requires</client-auth>
                <identity-assertion>Supports</identity-assertion>
            </security-interop>
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>ejb/AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
            </ejb-ref>
            <res-ref>
                <jndi-info>
                    <ref-name>jdbc/AccountDB</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
            </res-ref>
            <res-env-ref>
                <jndi-info>
                    <ref-name>jms/StockQueue</ref-name>
                    <export-name>STOCKQUEUE</export-name>
                </jndi-info>
            </res-env-ref>
            <thread-max>200</thread-max>
            <clustering>
                <home-clustered>true</home-clustered>
                <ejb-home-failover>true</ejb-home-failover>
```

```
                <ejb-home-idempotent-
method>foo(java.lang.String,int)</ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-
method>foo(java.lang.String,int)</ejb-remote-idempotent-method>
            </clustering>
            <invoke-http>
                <url>/mycontext/RMIHandlerServlet</url>
                <http-port>80</http-port>
            </invoke-http>
            <object-management>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                    <resizing-period>1800000</resizing-period>
                </bean-pool>
                <connect-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </connect-pool>
                <capacity>2000</capacity>
                <passivation-timeout>10000</passivation-timeout>
                <disconnect-timeout>180000</disconnect-timeout>
            </object-management>
            <persistence-optimize>
                <engine-type>SINGLE_OBJECT</engine-type>
<non-modify-method>
                  <method-name>myBusinessMethod</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
  <method-param>int</method-param>
  <method-param>double</method-param>
                  </method-params>
                </non-modify-method>
                <entity-cache-size>100</entity-cache-size>
            </persistence-optimize>
        </jeus-bean>
        <jeus-bean>
            <ejb-name>account</ejb-name>
```

```
            <export-name>ACCOUNTEJB</export-name>
            <local-export-name>LOCALACCOUNTEJB</local-export-
name>
            <export-port>7654</export-port>
            <export-iiop>true</export-iiop>
            <single-vm-only>true</single-vm-only>
            <local-invoke-optimize>true</local-invoke-optimize>
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            <security-interop>
                <integrity-confidentiality>Requires</integrity-
confidentiality>
                <trust-in-client>Requires</trust-in-client>
                <client-auth>Requires</client-auth>
                <identity-assertion>Supports</identity-assertion>
            </security-interop>
            <schema-info>
                <table-name>ACCOUNT</table-name>
                <creating-table>true</creating-table>
                <deleting-table>true</deleting-table>
                <cm-field>
                    <field>id</field>
                    <column-name>ID</column-name>
                    <type>NUMERIC</type>
                    <exclude-field>true</exclude-field>
                </cm-field>
                <prim-key-field>
                    <field>id</field>
</prim-key-field>
<find-method>
<query-method>
                    <method-name>findByAddress</method-name>
  <method-params>
<method-param>
  java.lang.String
</method-param>
   </method-params>
  </query-method>
```

```
                  <sql>customer_address=?</sql>
                </find-method>
                <db-vendor>oracle</db-vendor>
                <data-source-name>MYDB</data-source-name>
                <auto-key-generator>
                    <generator-type>USER_KEY_TABLE</generator-
type>
                    <generator-name>MYKEYTABLE</generator-name>
                    <sequence-column>PRIMARYKEYCOLUMN</sequence-
column>
                    <key-cache-size>20</key-cache-size>
                </auto-key-generator>
            </schema-info>
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>ejb/AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
            </ejb-ref>
            <res-ref>
                <jndi-info>
                    <ref-name>jdbc/AccountDB</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
            </res-ref>
            <res-env-ref>
                <jndi-info>
                    <ref-name>jms/StockQueue</ref-name>
                    <export-name>STOCKQUEUE</export-name>
                </jndi-info>
            </res-env-ref>
            <thread-max>200</thread-max>
            <clustering>
                <home-clustered>true</home-clustered>
```

```
                <ejb-home-failover>true</ejb-home-failover>
                <ejb-home-idempotent-
method>foo(java.lang.String,int)</ejb-home-idempotent-method>
                <ejb-remote-failover>true</ejb-remote-failover>
                <ejb-remote-idempotent-
method>foo(java.lang.String,int)</ejb-remote-idempotent-method>
            </clustering>
            <invoke-http>
                <url>/mycontext/RMIHandlerServlet</url>
                <http-port>80</http-port>
            </invoke-http>
            <object-management>
                <bean-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                    <resizing-period>1800000</resizing-period>
                </bean-pool>
                <connect-pool>
                    <pool-min>10</pool-min>
                    <pool-max>200</pool-max>
                </connect-pool>
                <capacity>2000</capacity>
                <passivation-timeout>10000</passivation-timeout>
                <disconnect-timeout>180000</disconnect-timeout>
            </object-management>
            <persistence-optimize>
                <engine-type>SINGLE_OBJECT</engine-type>
                <non-modify-method>
                  <method-name>myBusinessMethod</method-name>
                  <method-params>
                    <method-param>java.lang.String</method-param>
  <method-param>int</method-param>
  <method-param>double</method-param>
                  </method-params>
                </non-modify-method>
                <entity-cache-size>100</entity-cache-size>
            </persistence-optimize>
            <cm-persistence-optimize>
                <subengine-type>WriteLocking</subengine-type>
```

```
                <fetch-size>80</fetch-size>
                <init-caching>true</init-caching>
            </cm-persistence-optimize>
            <enable-instant-ql>true</enable-instant-ql>
        </jeus-bean>
        <jeus-bean>
            <ejb-name>order</ejb-name>
            <connection-factory-
name>jms/QueueConnectionFactory</connection-factory-name>
            <destination>MDB_QUEUE</destination>
            <max-message>20</max-message>
            <durable>Durable</durable>
            <jndi-spi>
                <mq-vendor>SONICMQ</mq-vendor>
                <initial-context-
factory>acme.jndi.ACMEContextFactory</initial-context-factory>
                <provider-
url>protocol://localhost:2345</provider-url>
            </jndi-spi>
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            <security-interop>
                <integrity-confidentiality>Requires</integrity-
confidentiality>
                <trust-in-client>Requires</trust-in-client>
                <client-auth>Requires</client-auth>
                <identity-assertion>Supports</identity-assertion>
            </security-interop>
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>ejb/AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
```

```
            </ejb-ref>
            <res-ref>
                <jndi-info>
                    <ref-name>jdbc/AccountDB</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
            </res-ref>
            <res-env-ref>
                <jndi-info>
                    <ref-name>jms/StockQueue</ref-name>
                    <export-name>STOCKQUEUE</export-name>
                </jndi-info>
            </res-env-ref>
            <thread-max>200</thread-max>
            <clustering>
                <home-clustered></home-clustered>
                <ejb-home-failover></ejb-home-failover>
                <ejb-home-idempotent-method></ejb-home-
idempotent-method>
                <ejb-remote-failover></ejb-remote-failover>
                <ejb-remote-idempotent-method></ejb-remote-
idempotent-method>
            </clustering>
            <invoke-http>
                <url>/mycontext/RMIHandlerServlet</url>
                <http-port>80</http-port>
            </invoke-http>
        </jeus-bean>
    </beanlist>
    <ejb-relation-map>
        <relation-name>student-course</relation-name>
        <table-name>studentcoursejoin</table-name>
        <jeus-relationship-role>
            <relationship-role-name>student-to-
course</relationship-role-name>
            <column-map>
                <foreign-key-column>manager-id</foreign-key-
column>
```

```
                <target-primary-key-column>man-id</target-
primary-key-column>
            </column-map>
        </jeus-relationship-role>
    </ejb-relation-map>
</jeus-ejb-dd>
```

# F Instant EJB QL API 레퍼런스

## F1 소개

EJB QL API는 EJB 클라이언트 어플리케이션 개발자가 클라이언트 쪽 코드에 직접 EJB QL 질의를 지정하여 EJB finder 메소드들이 가지는 제약점을 극복할 수 있도록 해준다.

**참고:** 이 API의 사용은 아주 극단적인 상황에서만 사용되어야 한다. 이 API는 표준 EJB 엔티티 finder 메소드보다 고정적이고 다소 비효율적이다.

이 API 는 단 한 개의 인터페이스와 하나의 메소드로 구성되어 있다.

## F.2 The EJBInstanceFinder Interface

```
interface jeus.ejb.bean.objectbase.EJBInstanceFinder
```

```
public abstract interface EJBInstanceFinder extends Remote
```

이 인터페이스는 jeus-ejb-dd.xml 파일의 enable-instant-ql element가 "true"로 설정되어 있는 환경에서 CMP 2.0 엔티티 빈의 home 인터페이스에 의해서 구현된다.

이 인터페이스는 클라이언트 코드 내에 임의의 EJB QL 질의를 직접 넣을 수 있도록 한다.

**Methods**

```
java.util.Collection findWithInstantQL (String ejbQlQuery)
```

*설명*

- "ejbQlQuery" 파라미터로 표현된 EJB QL 질의에 해당하는 빈의 EJB 집합을 리턴한다.

*파라미터*

- ejbQlQuery string: 유효한 EJB QL 문장으로 “?”이 없는 것이어야 한다. 이 문법은 JEUS에서 정의한 EJB QL 세 가지 추가 사항의 대상 중 하나이다(11장을 참고하라).

*리턴값*

- java.util.Collection: 질의에 답에 해당하는 빈 인터페이스의 집합.

*예외*

- FinderException
- RemoteException

# G JEUS EJB Ant Task 레퍼런스

## G.1 소개

JEUS 기반으로 EJB 컴포넌트를 개발할 때 작업을 편리하게 수행할 수 있도록 몇 가지의 Ant task를 제공한다. EJB 개발에 필요한 Ant task에는 deploy 관련 task가 있는데 이에 대해서는 JEUS Deployment 안내서를 참고하기 바란다.

EJB 와 관련된 task은 다음과 같다.

- appcompiler
- ejbddinit

여기서는 이 task 들에 대한 설명과 개발자가 명시할 수 있는 속성들을 제공한다. Ant 사용법과 설정에 대한 보다 자세한 정보는 JEUS Deployment 안내서를 참고하라. 그 외로 JEUS 서버를 시작하고 종료하는데 사용할 수 있는 Ant task에 대한 정보는 JEUS Server 안내서를 참고하라.

JEUS에서 제공하는 ant task를 사용하기 위해서는 다음과 같이 task definition을 build.xml에 추가해야 한다.

```
<taskdef resource="jeus/util/ant/jeusant.properties">
   <classpath>
      <path refid="jeus.libraries"/>
   </classpath>
</taskdef>
```

## G.2 appcompiler

“appcompiler” Ant task는 pre-deployment 작업 후 EJB 모듈 혹은 개별적인 EJB 빈에 대해서 필요한 RMI stub 과 skeleton 클래스를 임의로 생성하고 싶을 때 사용한다. 또한 이것은 클라이언트에서 원격지에 있는 Deploy된 빈과 통신할 때 필요한 클래스 파일들 즉 EJB client JAR를 생성할 수도 있다.

“appcompiler” task에서 사용하는 속성은 [표 9]에 열거되어 있다.

*표 21. 'appcompiler' Ant task 속성.*

| Attribute | 설명 | 선택여부 |
|---|---|---|
| jeushome | JEUS 홈 디렉토리. | 필수 |
| target | ejbcompile을 하려는 EJB 모듈의 archive file 또는 이를 풀어 둔 directory | 필수 |
| keep | appcompiler할 때에 생성되는 소스 파일을 유지할지의 여부 | 선택(기본값은 false) |
| ear | target이 EAR 모듈인지의 여부 | 선택(기본값은 false) |
| ejbjar | ejb-jar.xml의 path | 선택(기본값은 target의 META-INF/ejb-jar.xml) |
| jeusejbdd | jeus-ejb-dd.xml의 path | 선택(기본값은 target의 META-INF/jeus-ejb-dd.xml) |

예)

*<<Ant build script>>*

```
<appcompiler jeushome="${jeus.jeushome}" target="${modulename}"/>
```

# G.3 ejbddinit

‘ejbddinit’ Ant task는 EJB 클래스를 기반으로 ejb-jar.xml과 jeus-ejb-dd.xml 파일을 생성한다. 속성 파일을 만들어서 선택적인 설정이 가능하다( JEUS Builder 안내서에 있는 DD 생성 유틸리티에 관한 부록을 참고하라).

‘ejbddinit’ task에서 사용하는 속성은 [표 22]에 열거되어 있다.

*표 22. 'ddinit' Ant task 속성.*

| Attribute | 설명 | 선택여부 |
|---|---|---|
| target | EJBDDInit을 실행하기 위한 jar archive file 이나 이를 풀어둔 directory | 필수 |
| jeusHome | JEUS 홈 디렉토리. | 필수 |
| property | EJBDDInit을 실행할 때 참조할 property file. 자세한 사항은 아래에서 설명한다. | 선택 |
| logginglevel | EJBDDInit이 사용할 logging level을 지정한다. 이 Level은 J2SE logging API의 level을 따른다. | 선택(기본값은 INFO) |

또한 <classpath> 항목을 하위 항목으로 가질 수 있다. 이 classpath는 지정된 EJB 모듈이 사용하는 classpath를 지정할 수 있다.

예)

```
<ejbddinit logginglevel="FINEST" jeushome="d:/jeus/jeus5"
target="ejb.jar">
      <classpath>
<pathelement path="${jeus.home}/lib/application/tsharness.jar"/>
      </classpath>
</ejbddinit>
```

이 경우 descriptor를 만들 때 필요한 여러가지 정보를 property file로 제공할 수 있다. 이 property 파일의 각 속성은 다음 표와 같다.

| 프로퍼티 | 설명 | 기본값 |
|---|---|---|
| vendor | DB 벤더명 | N/A |
| datasource-name | CMP에서 사용할 데이터소스 | N/A |
| creating-table | CMP의 테이블 생성 여부 | false |

| 프로퍼티 | 설명 | 기본값 |
|---|---|---|
| deleting-table | CMP의 테이블 삭제 여부 | false |
| engine-type | Entity Bean의 엔진 타입 | EXCLUSIVE_ACCESS |
| subengine-type | CMP의 서브 엔진타입 | ReadLocking |
| fetch-size | ResultSet으로 한 번에 가져올 수 있는 레코드의 수 | 10 |
| enable-instant-ql | JEUS의 Instance QL 사용 여부 | false |
| local-optimize | Local Invoke Optimize 사용 여부 | true |
| logging-level | 로그 레벨 | fatal |
| export-port | RMI Listener Port | 0 |
| export-iiop | COS Naming 사용 여부 | false |
| single-vm-only | 해당 컨테이너의 JNDI에만 객체를 bind한다. | false |
| thread-pool-max | 최대 thread 개수 | 100 |
| bean-pool-min | EJB bean의 최소 개수 | 0 |
| bean-pool-max | EJB bean의 최대 개수 | 100 |
| capacity | EJB bean 인스턴스의 capacity | 10000 |
| passivation-timeout | EJB가 Passivate되는 시간 | -1 |
| disconnect-timeout | EJB의 인스턴스가 완전히 사라지는 시 | -1 |

| 프로퍼티 | 설명 | 기본값 |
|---|---|---|
| | 간 | |
| connect-pool-min | EJB 커넥션 풀의 최소 개수 | 0 |
| connect-pool-max | EJB 커넥션 풀의 최대 개수 | 100 |
| jms-connection | JMS connection의 export name | N/A |
| mail-connection | Mail 리소스의 export name | N/A |
| url-connection | URL 리소스의 export name | N/A |
| init-caching | Entity Bean을 미리 초기화할 지 여부 | false |

예) 속성 파일

<<*ddinit.properties*>>

```
#JEUS DD Generation Option
#Sat Mar 15 15:03:04 KST 2003
save-dir=C:/Jeus50/sample/src/ejb/basic/beanManaged/ddinit
module-name=beanmanaged_product
base-dir=C:/Jeus50/sample/out/ejb/basic/beanManaged/build
logging-level=debug
local-optimize=false
```

# 색 인

F



H



I



J