

EJB 안내서

JEUS 9.1

TMAXSOFT

저작권 공지

Copyright 2026. TmaxSoft Co., Ltd. All Rights Reserved.

회사 정보

(주)티맥스소프트

주소 : 경기도 성남시 분당구 정자일로 45, 티맥스소프트타워

기술 서비스 센터: 1544-8629

홈페이지: <https://www.tmaxsoft.com>

제한된 권리

이 소프트웨어(JEUS®) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

Java, Solaris는 Oracle Corporation 및 그 자회사, 관계회사의 등록 상표입니다.

Microsoft, Windows, Windows NT는 Microsoft Corporation의 등록 상표 또는 상표입니다.

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

Noto는 Google Inc.의 상표입니다. Noto 글꼴은 오픈 소스입니다. 모든 Noto 글꼴은 SIL Open Font License, 버전 1.1에 따라 게시됩니다. (<https://www.google.com/get/noto/>)

기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상호, 상표, 또는 등록 상표입니다.

본 사용설명서에 기재된 회사, 시스템, 제품 이름 등에 반드시 상표 표시(™, ®)를 하지는 않습니다.

오픈 소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다.: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. :

`${INSTALL_PATH}/license/oss_licenses`

유지 보수

구분	지원항목	서비스 내용
제품지원	패치 & 업그레이드	무상 패치 서비스 제공 메이저 버전 업그레이드 시 할인 혜택 웹 지원을 통한 패치 내역 제공
기술 지원 - 기본 서비스	장애 지원	장애 발생 시 원인 분석 및 조치 Service Desk팀 → 기술팀 → R&D의 3단계 장애 분석 및 조치
	일상 지원(온라인 지원)	E-mail, 전화, 원격, 웹 사이트 등 온라인 자원을 통한 질의 응답 서비스
	고객 맞춤 지원(방문 지원)	고객의 요청으로 수행하는 방문 지원 서비스
기술 지원 - 옵션 서비스	예방 지원	정기 점검을 통한 시스템 운영현황 보고 및 장애 예방 ◦ 관리자 또는 운영자의 요구사항 수렴 ◦ 운영 현황(시스템, 엔진 운영) 보고서 제공 ◦ 필요 시 시스템 개선 권장 사항 보고
유지 보수 비용 및 기간	계약 시 별도 협의	계약 시 EOL/EOS 문서 제공

안내서 이력

제품 버전	안내서 버전	발행일	비고
JEUS 9.1.1	3.4.1	2026-06-08	Fix 릴리스
JEUS 9.1	3.3.1	2025-09-30	GA 버전
JEUS 9 Fix#1	3.2.2	2025-07-10	JDK 21 지원 내용 추가
JEUS 9 Fix#1	3.2.1	2025-06-30	RC 버전
JEUS 9	3.1.2	2025-01-03	-

제품 버전	안내서 버전	발행일	비고
JEUS 9	3.1.1	2024-09-27	-

목차

용어 해설	1
1. EJB 소개	3
1.1. 개요	3
1.2. 구성 요소	3
1.3. EJB 환경 및 설정	5
1.3.1. 디렉터리 구조	5
1.3.2. XML 설정 파일	6
1.3.3. 관련 툴	8
1.4. EJB 기본 설정	8
2. EJB 엔진	11
2.1. 개요	11
2.2. 주요 기능	11
2.3. EJB 엔진 디렉터리 구조	12
2.4. EJB 엔진 설정	13
2.4.1. Basic 설정	13
2.4.2. Active Management 설정	15
2.4.3. Timer Service 설정	16
2.5. 시스템 로그 설정	17
2.6. EJB 엔진 제어 및 모니터링	17
2.7. EJB 엔진 튜닝	17
2.7.1. Resolution 설정 튜닝	17
2.7.2. Fast Deploy	18
2.7.3. 최대 성능을 위한 시스템 로그 설정	18
2.7.4. Active Management 사용하지 않기	18
2.7.5. HTTP Invoke 모드 사용	18
3. EJB 모듈	19
3.1. 개요	19
3.2. EJB 모듈 관리	19
3.3. EJB 모듈 조립	21
3.3.1. EJB 클래스 컴파일	22
3.3.2. Deployment Descriptors(DD) 작성	22
3.3.3. EJB JAR 파일 패키징	25
3.4. EJB 모듈 Deploy	25
3.4.1. Deploy	26
3.4.2. Deploy된 EJB 모듈의 디렉터리 구조	27
3.5. EJB 모듈 제어 및 모니터링	28
3.5.1. EJB 모듈 제어	29
3.5.2. EJB 모듈 모니터링	30
4. EJB의 공통 특성	33

4.1. 개요	33
4.2. EJB 설정	35
4.2.1. 기본 환경설정	35
4.2.2. Thread Ticket 설정	37
4.2.3. External Reference 설정과 매핑	38
4.2.4. HTTP Invoke 환경설정	41
4.2.5. EJB 보안 설정	42
4.2.5.1. 보안 설정	42
4.2.5.2. 보안 설정 예제	43
4.3. EJB 모니터링	46
4.4. EJB 튜닝	47
4.4.1. Thread Ticket Pool 설정 튜닝	47
5. EJB 상호 운용성 및 RMI/IIOP	49
5.1. 개요	49
5.1.1. 트랜잭션 상호 운용(OTS)	49
5.1.2. 보안 상호 운용(CSiv2)	50
5.2. 상호 운용 설정	50
5.2.1. COS Naming Service 설정	50
5.2.2. 상호 운용성 활성화 설정	50
5.2.3. CSiv2 보안 상호 운용 설정	50
5.2.4. EJB RMI/IIOP 설정	51
5.3. RMI/IIOP 클라이언트	52
5.3.1. JEUS Managed Server	52
5.3.2. 다른 벤더 WAS	53
5.3.3. Standalone 클라이언트	53
5.4. 알려진 문제점(Known Issues)	54
6. EJB 클러스터링	56
6.1. 개요	56
6.2. 주요 기능	57
6.2.1. Load Balancing	57
6.2.2. Failover(EJB 복구)	58
6.2.3. Idempotent 메소드를 통한 EJB 복구	59
6.2.4. Session Replication	59
6.3. EJB 클러스터링 설정	60
6.3.1. Annotation을 통한 클러스터링 설정	60
6.3.2. xml을 통한 클러스터링 설정	62
6.4. EJB Failover 제한	63
7. Session Bean	65
7.1. Stateless Session Bean	65
7.1.1. Thread Ticket Pool(TTP)과 Object Management	65
7.1.2. Web Service Endpoint	66

7.2. Stateful Session Bean	66
7.2.1. Thread Ticket Pool(TTP)과 Object Management	66
7.2.2. Pooling Session Bean	67
7.2.3. Bean Pool 설정	68
7.2.4. 세션 데이터 유지 메커니즘 설정	68
7.3. 공통 설정	69
7.3.1. Object Management 관련 설정	69
8. Entity Bean	72
8.1. 개요	72
8.2. 주요 기능	73
8.2.1. 공통 기능	73
8.2.1.1. Entity Bean Object 관리 및 Entity Cache	73
8.2.1.2. 엔진 모드 Selection을 통한 ejbLoad() Persistence 최적화	75
8.2.2. BMP & CMP 1.1	77
8.2.2.1. Non-modifying 메소드에 의한 ejbStore() Persistence 최적화	78
8.2.3. CMP 1.1/2.0	78
8.2.3.1. ejbLoad(), ejbFind(), CM Persistence 최적화	78
8.2.3.2. Entity Bean 스키마 정보	79
8.2.3.3. 자동 Primary Key 생성	79
8.2.4. CMP 2.0	83
8.2.4.1. Entity Bean Relationship Mapping	83
8.2.4.2. JEUS EJB QL Extension	83
8.2.4.3. Instant EJB QL	83
8.3. Entity EJB 설정	84
8.3.1. 공통 설정	84
8.3.1.1. 기본 공통 항목 설정	84
8.3.1.2. Object Management 관련 설정	85
8.3.1.3. ejbLoad()와 ejbStore() Persistence 최적화 설정	87
8.3.2. CMP 1.1/2.0	89
8.3.2.1. ejbLoad()와 ejbFind() CM Persistence 최적화 설정	89
8.3.2.2. DB 스키마 정보 설정	90
8.3.3. CMP 2.0	94
8.3.3.1. Relationship Mapping 설정	94
8.3.3.2. Instant EJB QL의 설정	97
8.3.3.3. JEUS EJB QL Extension	97
8.3.4. DB Insert Delay 설정(CMP Only)	102
8.4. Entity EJB 튜닝	103
8.4.1. 공통	103
8.4.2. BMP & CMP 1.1	104
8.4.3. CMP 1.1/2.0	104
8.4.4. CMP 2.0	105

8.5. 완전한 CMP 2.0 Entity Bean 예제	105
9. Message Driven Bean(MDB)	110
9.1. 개요	110
9.2. MDB 설정	110
9.2.1. 기본 환경설정	110
9.2.2. JMS 설정	111
9.2.3. JNDI SPI 환경설정	114
10. EJB Timer Service	116
10.1. Timer Service 설정	116
10.1.1. Persistent Timer Service 설정(EJB 엔진)	116
10.1.2. Persistent Timer 처리(jeus-ejb-dd.xml)	117
10.1.3. Cluster-Wide Timer Service 설정	118
10.2. Timer 모니터링	118
10.3. Timer Service 사용 주의사항	119
11. EJB 클라이언트	120
11.1. 개요	120
11.2. EJB 접근을 위한 클라이언트 프로그래밍	120
11.3. InitialContext 설정	121
11.3.1. JVM 속성을 이용한 Naming 속성값 설정	122
11.3.2. Hashtable을 이용한 Naming 속성 설정	123
12. 부가 기능	124
12.1. WorkArea 서비스	124
12.1.1. UserWorkArea 인터페이스	124
12.1.2. PropertyMode 타입	125
12.1.3. 예외	125
12.1.4. Nested UserWorkArea	125
12.1.5. UserWorkArea를 사용하는 응용 프로그램 개발	126
Appendix A: 기본 Java 타입과 DB 필드 매핑	130
A.1. 개요	130
A.2. Tibero 필드 - 컬럼 타입 매핑	130
A.3. Oracle 필드 - 컬럼 타입 매핑	131
A.4. Sybase 필드 - 컬럼 타입 매핑	132
A.5. Microsoft SQL Server 필드 - 컬럼 타입 매핑	133
A.6. DB2 필드 - 컬럼 타입 매핑	134
A.7. Cloudscape 필드 - 컬럼 타입 매핑	134
A.8. Informix 필드 - 컬럼 타입 매핑	135
Appendix B: Instant EJB QL API Reference	137
B.1. 개요	137
B.2. EJBInstanceFinder Interface	137
B.3. EJBInstanceFinder Method	137

용어 해설

2차 저장 장소

영구적인 기억장소로써 유지를 위해 전원을 요구하지 않는다(예: 하드 디스크).

부하 분산

클러스터링에서 작업이 한쪽으로 치우치지 않도록 하는 기술이다. 같은 EJB 애플리케이션을 실행하는 엔진에 클라이언트의 요청을 공정히 분배함으로써 성능을 향상시킬 때 사용한다.

주 메모리

Runtime Memory라고도 한다. 컴퓨터 내부의 실제 가용 메모리(RAM)이다.

Activation

Passivation의 반대 개념이다. Bean Instance를 제 2차 저장 장소에서 주 메모리로 이동하는 것을 의미한다.

Active Management

비정상적인 상황일 때 JEUS에서 자동으로 이메일(E-mail)을 발송하는 것을 의미한다.

Bean Pool

EJB Instance의 Pool이다.

Bean Type

Jakarta EE의 기본 Bean 타입이거나(예: Stateless, Stateful, Entity and Message Driven Bean) 개발자가 정의한 Bean이다(기본 Jakarta EE Bean의 하위 Bean).

BMP

Bean Managed Persistence Entity Bean이다.

CMP

Container Managed Persistence Entity Bean이다.

Connection Pool

EJB Objects의 Pool이다. 이 Objects는 클라이언트와 EJB 간에 연결 및 통신을 책임진다.

DD

Deployment Descriptor의 약어이다.

EJB

Enterprise Java Beans이다.

EJB Clustering

여러 EJB 엔진에 분산되어 있는 EJB Bean의 설정이다. 클라이언트 입장에서는 하나의 Bean처럼 보이지만 내부적으로는 안정성과 성능을 향상하기 위해 부하 분산이 이루어진다.

EJB Client

EJB의 메소드를 사용하는 컴포넌트이다(예를 들면, 독립적인 Java application, Servlet 또는 다른 EJB).

EJB Engine

Jakarta EE 스펙에서 “EJB container”이다. 이것은 EJB 컴포넌트의 기반 환경을 제공한다.

EJB Failover

같은 EJB 애플리케이션을 실행하는 여러 엔진을 사용함으로써 안정성을 높일 때 사용한다.

EJB Module

EJB JAR 안에 있는 하나 혹은 그 이상의 EJB 패키지이다. EJB 엔진에 deploy되는 EJB는 EJB 모듈을 사용한 다.

Entity Cache

Entity Bean에서 비활성화된 Bean을 유지하기 위해 사용되는 특별한 Cache이다. 이 캐시가 채워졌을 때만 Bean이 제 2차 저장 장소로 비활성화된다. 이것은 성능 향상을 위해 사용된다.

MDB

Message Driven Bean의 약어이다.

Object Management

Connection Pool과 Bean Pool 환경을 참고한다.

Passivation

EJB 인스턴스를 다시 요청이 있을 때까지 주 메모리에서 제 2차 저장 장소로 옮겨놓는 것이다.

Round Robin

Queue의 데이터 구조와 같다. Queue에 들어온 순서대로 선택된다. 이것은 컴포넌트별로 공정한 부하 분산을 보장한다.

SF

Stateful Session Bean이다.

SL

Stateless Session Bean이다.

Thread Ticket Pool

Access Ticket의 Pool이다. JEUS에서 EJB에 접근하기 위해서 클라이언트는 반드시 Thread Ticket Pool로부터 Thread Ticket을 얻어야 한다.

1. EJB 소개

본 장에서는 EJB의 구성 요소와 특성, 그리고 간단한 설치와 환경설정 방법에 대해서 설명한다.

1.1. 개요

JEUS는 JSR 345 Jakarta™ Enterprise Beans 4.0(이하 EJB 4.0)을 지원하고 있다. EJB 4.0 표준(이하 EJB 표준)은 복잡하고 중요한 고성능의 비즈니스 로직을 개발할 때 개발자의 노력을 최소화하면서도 이식성이 뛰어난 비즈니스 컴포넌트를 작성할 수 있는 구조를 기술하고 있다.



Jakarta EE 9부터 EJB 4.0이 적용된다. 단, 기존 EJB와는 다른 새로운 버전이 아니라 EJB 2.x와 3.x의 기능을 지원해야 함을 명시하고 있으므로, 실질적으로 사용자는 EJB 2.x와 3.x를 계속 사용하게 된다. 따라서 본 안내서에서는 EJB 4.0에 대한 추가 설명 없이 EJB 2.x와 EJB 3.x의 동작 방식만을 안내한다.

EJB 4.0에 따르면 EJB의 정의는 다음과 같다.

"EJB 아키텍처는 객체지향 분산 엔터프라이즈 애플리케이션의 개발 및 분산 배치를 위한 컴포넌트이다. EJB로 작성된 애플리케이션은 확장성, 트랜잭션 처리 그리고 동시 사용자 처리에 안전하다. EJB 스펙을 준수한 애플리케이션은 한번 작성되면 어떠한 서버 플랫폼에도 배치된다."

JEUS는 위에서 언급한 "서버 플랫폼"의 역할을 담당한다. 본 안내서에서는 JEUS에서 구현한 EJB 구현체와 엔터프라이즈 환경에 필수적인 부가 기능에 대해서 설명한다.



더 자세한 스펙 구현 정보들은 "JEUS 소개"를 참고한다.

JEUS EJB는 클러스터링을 통한 성능과 안정성을 더하기 위해 JEUS는 표준 외에 2가지 추가 기능을 제공한다.

- 성능 개선을 위한 Load Balancing 기능
- 안정성 확장을 위한 Failover 기능

이 기능들을 사용하려면 2개 이상의 Managed Server(이하 MS)가 클러스터로 구성되어 있어야 한다.

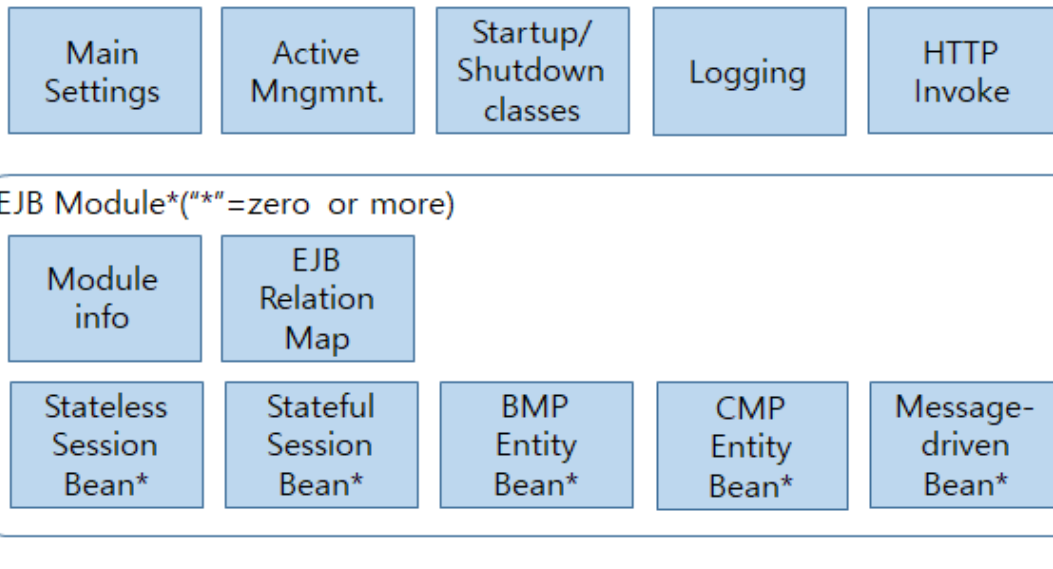


EJB 클러스터링에 대한 자세한 내용은 [EJB 클러스터링](#)을 참고하고, MS 클러스터링에 대한 자세한 내용은 JEUS Domain 안내서의 "JEUS 클러스터링"을 참고한다.

1.2. 구성 요소

다음은 EJB 구현체의 주요 구성 요소에 대한 그림이다.

EJB Engine



EJB 구현체의 주요 구성 요소

• EJB 엔진

EJB 4.0에서 설명하는 EJB 컨테이너에 해당하는 것으로 deploy된 EJB 모듈의 실행 환경을 제공한다. 자세한 내용은 [EJB 엔진](#)을 참고한다.

• EJB 모듈

각각의 EJB 컴포넌트를 그룹화하고 관리할 수 있는 단위이다. JEUS에서는 EJB 컴포넌트를 deploy하고 제어할 때 EJB 모듈을 기본 단위로 하므로, EJB 모듈은 반드시 하나 이상의 EJB 컴포넌트를 포함하고 있다. 자세한 내용은 [EJB 모듈](#)을 참고한다.

• EJB 컴포넌트

EJB 모듈 내에 속해 있으며 실제 비즈니스 컴포넌트를 의미한다. 각 EJB 컴포넌트의 공통 특성은 [EJB의 공통 특성](#)을 참고한다.

EJB 컴포넌트는 다음과 같이 5가지 종류로 구분된다. 각 컴포넌트에 대한 자세한 내용은 각 장을 참고한다.

- Stateless Session Bean : [Session Bean](#)
- Stateful Session Bean : [Session Bean](#)
- BMP Entity Bean : [Entity Bean](#)
- CMP 1.1 & CMP 2.x Entity Bean : [Entity Bean](#)
- MDB : [Message Driven Bean\(MDB\)](#)



Entity Bean은 EJB 3.0부터 JPA로 대체되었으므로 이에 대해서는 "JEUS JPA 안내서"를 참고한다.

1.3. EJB 환경 및 설정

JEUS EJB에 관련된 환경설정 요소들은 EJB에 관련된 디렉터리 구조와 EJB 설정 파일들, 각종 툴 등으로 구성된다. 또한 EJB와 관련된 시스템 프로퍼티들과 그 특성들도 설명한다.

1.3.1. 디렉터리 구조

다음은 JEUS EJB에 관련된 디렉터리들과 파일에 대한 설명이다.

```
{JEUS_HOME}
|--bin
|   |--appcompiler
|   |--jeusadmin
|--domains
|   |--<domain_name>
|   |   |--config
|   |   |   |--domain.xml
|   |   |--servers
|   |       |--<server_name>
|   |           |--logs
|   |               |--error.log
|   |--.application
|       |--<EAR_archive_file>
|       |   |--APP-INF
|       |   |--Lib
|       |   |--META-INF
|       |   |--<EAR_archive_file>_jar__
|       |       |--META-INF
|       |       |--EJB class file
|       |--<EJB_archive_file>
|       |   |--<EJB_archive_file>_jar__
|       |       |--META-INF
|       |           |--ejb-jar.xml
|       |           |--jeus-ejb-jar.xml
|--lib
|   |--schemas
|       |--jakartaee
|       |--jeus
|--sample
|   |--ejb
|       |--.java EJB sample
```

bin

관리자가 EJB를 관리할 때 사용할 수 있는 실행 스크립트들이 저장된다. 여기에는 jeusadmin, appcompiler 등이 있다.

domains\<domain name>\config

현재의 JEUS 시스템의 주요 설정 디렉터리이다. 이 디렉터리에는 domain.xml 파일이 위치한다.

domains\<domain name>\servers\<server name>\logs

EJB 엔진의 로그 파일이 위치한다. EJB 로그의 파일 핸들러를 별도로 지정한 경우 별도의 파일로 생성된다. 핸들러가 없는 경우에는 서버의 로그 설정을 따른다.

domains\<domain name>\.application\

도메인에서 사용하는 애플리케이션들의 Archive 파일이 복사되고 이 Archive 파일을 풀어 놓은 디렉터리가 위치한다. 애플리케이션의 Archive 파일과 풀린 디렉터리에는 EJB 구현 클래스들과 helper 클래스들이 포함되어 있다.

파일	설명
<EAR archive file>	EAR application-id 이름의 디렉터리 아래에 EAR 애플리케이션의 구성 파일이 그대로 복사된다.
<EJB archive file>	EJB module-id 이름의 디렉터리 아래에 EJB 모듈을 구성하는 파일이 그대로 복사된다.

lib\schemas

EJB와 관련된 스키마 파일이 다음과 같이 디렉터리별로 위치한다.

하위 디렉터리	설명
jakartaee	EJB와 관련된 모든 Jakarta EE XML 스키마 파일들이 위치한다.
jeus	EJB와 관련된 모든 JEUS XML 스키마 파일들이 위치한다.

samples\ejb

여러 종류의 EJB들을 구현해 놓은 예제 코드들과 하위 디렉터리들이 포함되어 있다.



디렉터리 목록에서 사용된 "<", ">" 표현(예: "<server name>/")은 그 사이에 사용된 문자들은 실제 시스템 설정 값으로 대체되어야 한다. 예를 들어 실제 시스템이 "server1"이라면 JEUS MS는 "server1"이라고 값이 설정되어야 한다. 그러므로 "<server name>/" 표현은 실제 시스템에서 "server1/"이라는 이름의 디렉터리로 대체되어야 한다.

1.3.2. XML 설정 파일

다음은 JEUS EJB의 관리와 설정에 관련된 XML 설정 파일이다.

XML 설정 파일들의 내용은 반드시 표준, 즉 JEUS에서 정의된 XML 헤더로 시작되어야 한다. 각 설정 파일의 XML 스키마 파일은 JEUS_HOME/lib/schemas/jeus/supportLocale/ko/ 디렉터리에 위치한다. 또한 Root Element는 JEUS XML 스키마의 namespace를 기존 name-space로 지정해야 한다.

- **domain.xml** (jeus-domain.xsd)

EJB 엔진에 대한 설정을 한다. 자세한 설명은 "JEUS Server 안내서"와 본 안내서의 [EJB 엔진](#)을 참고한다.

- 위치

JEUS_HOME/domains/<domain name>/config

- domain.xml의 XML 헤더

XML 헤더 : <domain.xml>

```
<?xml version="1.0"?>
<domain version="9.1" xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

- **ejb-jar.xml** (ejb-jar_4_0.xsd)

주 EJB 엔진 설정 파일이다. 자세한 설명은 EJB 4.0 스펙을 참고한다.

- 위치

META-INF\ 표준 JAR 파일 내의 구조

- ejb-jar.xml의 XML 헤더

ejb-jar.xml의 XML 헤더는 EJB 표준에 포함되어 있다.

XML 헤더 : <ejb-jar.xml>

```
<?xml version="1.0"?>
<ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd">
```

- **jeus-ejb-dd.xml** (jeus-ejb-dd.xsd)

JEUS EJB 모듈을 위한 JEUS Deployment Descriptors(이하 DD) 정보를 설정한다. 자세한 설명은 [EJB 모듈](#)을 참고한다.

- 위치

META-INF\ 표준 JAR 파일 내의 구조

- jeus-ejb-dd.xml의 XML 헤더

XML 헤더 : <jeus-ejb-dd.xml>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9.1">
```



본 안내서에서 사용되는 모든 태그 순서는 XML 스키마의 설정 순서대로 작성되어 있다. 태그

순서는 "JEUS XML Reference"를 참고한다. 그러나 실제 사용할 때는 순서를 제대로 지키기가 쉽지 않으므로 JEUS에서는 태그 순서를 자동으로 정렬해주는 기능을 제공한다. 그러므로 XML 설정 파일을 작성할 때 순서를 정확하게 지키지 않아도 무방하다.

위 파일들의 예제가 본 안내서에서 사용될 때에는 표준 헤더가 편의상 생략되어 사용되고 있다. 실제 XML 설정 파일에는 이 헤더들이 포함되어 있다는 것에 주의한다.

1.3.3. 관련 툴

다음은 EJB 엔진, EJB 모듈 그리고 EJB 컴포넌트를 관리하기 위해 사용되는 툴이다.

- 콘솔 툴(jeusadmin)

JEUS 콘솔 툴은 EJB 엔진을 제어하고 모니터링하기 위해 사용된다. 자세한 내용은 JEUS Reference 안내서의 "EJB 엔진 관련 명령어"를 참고한다.



EJB 2.x를 위한 appcompiler에 대한 설명은 JEUS Reference 안내서의 "appcompiler"를 참고한다.

1.4. EJB 기본 설정

본 절에서는 JEUS에서 EJB를 사용하기 위한 설치 단계를 설명한다. 다음은 server1이라는 이름의 서버에 EJB 엔진을 설정하는 과정에 대한 설명이다.

1. JEUS가 제대로 설치되었는지 확인하고, 환경변수를 확인한다.
2. Command 화면을 실행하고 JEUS_HOME/bin에서 다음과 같이 입력하면 MASTER가 기동된다.

```
$ startMasterServer -domain domain1 -u administrator -p password1 -verbose
. . .
[2016.08.06 21:33:15][2] [adminServer-1] [SERVER-0248] The JEUS server is STARTING.
. . .
[2016.08.06 21:33:16][3] [adminServer-1] [Security-0082] The JEUS security manager started.
. . .
[2016.08.06 21:33:16][3] [adminServer-1] [SERVER-0173] The JNDI naming server started.
. . .
[2016.08.06 21:33:28][2] [launcher-10] [Launcher-0034] The server[adminServer] initialization
completed successfully[pid : 2380].
[2016.08.06 21:33:28][0] [launcher-1] [Launcher-0040] Successfully started the server. The server
state is now RUNNING.
```



위에 보이는 [3]은 로그 레벨 중 CONFIG를 의미한다. 따라서 CONFIG 이상의 로그 레벨이 설정되어 있을 경우에만 확인 가능하다.

3. domain.xml EJB 엔진에 대해 설정할 수 있다. 설정을 하지 않는 경우 기본값으로 동작된다.



JEUS에서 제공되는 툴을 통한 MS의 설정 변경은 MASTER가 기동되어 있는 상태에서 가능하다.

4. 다른 Command 화면에서 다음과 같이 명령어를 실행한다(호스트 정보는 MASTER의 정보이다).

```
$ jeusadmin -host localhost:9736
User name:
```

5. 설치할 때 설정한 관리자의 사용자 이름과 패스워드를 입력한다.

```
User name: administrator
Password:
```

6. Command 화면에서 다음과 같이 입력하고 실행하면 서버가 기동(Booting)된다.

```
$ startManagedServer -domain domain1 -server server1 -u administrator -p password1 -verbose
...
[2016.08.07 14:10:10][2] [server1-1] [SERVER-0248] The JEUS server is STARTING.
...
[2016.08.07 14:10:17][2] [server1-1] [SERVER-0248] The JEUS server is RUNNING.
```

7. help 명령어를 입력하면 EJB 엔진에 대한 모니터링과 관리에 대한 명령어들을 조회할 수 있다.

```
[MASTER]domain1.adminServer>help -g EJB
[ EJB] -----
cancel-ejb-timer      Cancels active EJB timers.
ejb-timer-info        Lists the active EJB timers. If no options are
                      specified, a list of all EJB modules that
                      contain timers will be displayed.
modify-active-management Modify the active management configuration.
modify-check-resolution Set the check resolution of the EJB engine on the
                      server.
show-active-management Shows the active management of the EJB engine on
                      the server.
show-check-resolution Shows the check resolution of the EJB engine on
                      the server.
To show detailed information for a command, use 'help [COMMAND_NAME]'.
ex) help connect
```

8. 종료하려면 Command 화면에 stop-server(서버 종료) → local-shutdown(Master 종료) → exit(툴 종료)를 순서대로 입력한다.

```
[MASTER]domain1.adminServer>stop-server server1
Server [server1] was successfully stopped.
[MASTER]domain1.adminServer>local-shutdown
The server [adminServer] has been shut down successfully.
```

```
offline>exit
```

9. 전체 JEUS 시스템이 종료된다.



설치와 설정에 대한 자세한 내용은 "JEUS 설치 및 시작하기"와 "JEUS Server 안내서"를 참고한다.

2. EJB 엔진

본 장에서는 EJB 엔진에 대한 기초적인 사항과 JEUS EJB의 최상위 레벨의 개념인 구조, 설정, 운영, 모니터링과 튜닝에 대해 설명한다.

2.1. 개요

EJB 엔진은 EJB의 운영 환경을 제공한다. 본 안내서에서 사용한 **EJB 엔진**은 EJB 표준에서 사용하는 EJB 컨테이너라는 용어와 동일한 개념이다. EJB 모듈, EJB deploy에 대한 상세한 정보는 각각 [EJB 모듈](#)과 [EJB의 공통 특성](#)을 참고한다.

2.2. 주요 기능

다음은 EJB 엔진의 주요 기능에 대한 설명이다.

- **EJB 엔진과 Managed Server(MS)**

하나의 MS에는 하나의 EJB 엔진이 존재한다. 그러나 하나의 도메인에는 여러 개의 MS가 존재할 수 있기 때문에 하나의 도메인에 여러 개의 EJB 엔진이 존재할 수 있다.

일반적으로 여러 개의 머신이나 CPU 위에 여러 개의 MS에 EJB 엔진이 설정되고, Master에 의해 MS는 클러스터로 묶이는데 이러한 설정을 **EJB 클러스터링**이라고 한다. 이 구조는 시스템의 성능 향상과 높은 안정성 및 보안성을 유지해주는 효율적인 시스템 구조이다. EJB 클러스터링에 대한 자세한 내용은 [EJB 클러스터링](#)을 참고한다.

- **EJB 엔진 기본 설정**

EJB 엔진은 **domain.xml**의 **<ejb-engine>**에서 설정한다. 자세한 내용은 [EJB 엔진 설정](#)을 참고한다.

- **EJB 엔진 Logging**

domain.xml의 설정에 따라 MS에 로그를 남길 수 있지만 특별히 EJB 엔진의 로그(jeus.ejb)만 별도로 남길 수 있다. 이때 EJB 엔진 로그를 별도로 남기더라도 MS 로그에는 EJB 엔진 로그가 남는다.

Logger의 핸들러에 파일 핸들러가 지정되면 지정한 파일명으로 로그 파일이 생성된다. 또한, 콘솔 핸들러를 사용하면 모든 로그 메시지를 화면에 남길 수도 있다. 일반적으로 이런 상황에서는 로그 메시지가 파이프를 통하여 MS가 시작된 Command 화면에 출력된다.



그 외에 사용자가 생성한 사용자 핸들러를 등록할 수 있다. 자세한 설명은 JEUS Server 안내서의 "Logging"을 참고한다.

- **Active Management**

Active Management는 EJB 모듈에 문제가 발생한 경우에 EJB 엔진이 자체적으로 이메일(e-mail)로 통지를 보내주는 기능이다.

예를 들어 Bean 클래스의 잘못된 구현으로 인해서 무한 루프에 빠지거나 Deadlock과 같은 심각한 오류가 발생했을 때 EJB 엔진이 이를 감지하여 통지를 보내주도록 되어 있다. 부가적으로 오류 처리 정책을 설정할 수 있어서 비정상적인 현상이 발생한 경우에 이메일 통지뿐만 아니라 해당 EJB 엔진이 동작하는 MS를 재시작할 수 있도록 권고메시지를 출력할 수 있다. Active Management에 대한 자세한 설정은 [EJB 엔진 설정](#)을 참고한다.

• HTTP Invoke

원격 클라이언트가 JNDI 서비스를 통해서 EJB 인스턴스를 찾을 때 클라이언트는 EJB 메소드를 호출할 수 있는 RMI Stub을 받는다. 기본적으로 Stub은 RMI 런타임을 통해서 EJB와 원격 통신이 이루어지며, RMI 통신은 TCP 소켓을 기반으로 하고 있다. 이 방식은 RMI 통신 포트를 별도로 필요로 하기 때문에 방화벽이 있는 환경에서는 문제가 될 수 있다. 이 경우 특별한 통신 모드가 필요한데 이것이 **HTTP Invoke 모드**이다. 이 모드를 사용할 경우 원격 클라이언트의 RMI 요청을 HTTP로 감싸서 웹 엔진으로 보내고 웹 엔진은 RMI 요청을 다루는 서블릿(jeus.rmi.http.ServletHandler)으로 요청을 전달한다. 이 서블릿은 RMI 런타임으로 요청을 전달해서 실제 EJB 메소드를 호출한 뒤 그 결과를 HTTP 응답으로 감싸서 원격 클라이언트로 전달한다. HTTP Invoke 모드는 EJB 엔진 또는 EJB 컴포넌트별로 설정할 수 있다.

domain.xml의 <invoke-http>에 HTTP Invoke 모드가 설정되면 EJB 엔진 내의 모든 모듈에 적용된다. EJB 모듈의 jeus-ejb-dd.xml에서는 특정 EJB 컴포넌트에만 적용할 수 있다. 이때 jeus-ejb-dd.xml의 설정이 domain.xml의 설정보다 우선한다. domain.xml과 jeus-ejb-dd.xml의 설정에 대한 자세한 내용은 [HTTP Invoke 환경설정](#)을 참고한다.

2.3. EJB 엔진 디렉터리 구조

다음은 EJB 엔진을 관리할 때 사용하게 되는 디렉터리와 파일의 목록이다.

```
{JEUS_HOME}
|--bin
|   |--[01]appcompiler
|   |--[01]jeusadmin
|--domains
|   |--<domain_name>
|       |--config
|       |   |--[X]domain.xml
|       |--servers
|       |   |--<server_name>
|       |       |--logs
|       |       |--[T]error.log
|--lib
|   |--schemas
|       |--jakartaee
|       |--jeus
```

* Legend

- [01]: binary or executable file
- [X] : XML document
- [J] : JAR file
- [T] : Text file
- [C] : Class file
- [V] : java source file
- [DD] : deployment descriptor

기본 디렉터리에 대한 내용은 [디렉터리 구조](#)와 동일하고, 다음은 EJB 엔진에서 주로 사용하는 디렉터리에 대한 설명이다.

bin

EJB 엔진을 관리하는 툴이 위치한 디렉터리이다.

EJB 엔진 관리 툴	설명
appcompiler	EJB를 deploy하기 위해 필요한 클래스를 생성하고 컴파일해서 Fast Deploy를 수행한다.
jeusadmin	EJB 엔진을 제어하고 모니터링하기 위해 사용된다.

domains/<domain name>/servers/<server name>/logs

EJB 엔진의 로그 파일이 위치한 디렉터리이다. EJB 로그의 파일 핸들러를 별도로 지정한 경우 별도의 파일로 생성된다. 핸들러가 없는 경우에는 서버의 로그 설정을 따르게 된다.

2.4. EJB 엔진 설정

본 절에서는 domain.xml을 사용하여 EJB 엔진을 설정하는 방법을 설명한다.

EJB 엔진 설정은 크게 다음의 3가지 설정으로 나눌 수 있다. 설정된 내용은 JEUS_HOME/domains/<domain name>/config에 위치한 domain.xml 파일에 저장된다.

- [\[Basic\]](#)
- [\[Active Management\]](#)
- [\[Timer Service\]](#)



하나의 MS에는 하나의 EJB 엔진이 존재한다. MS를 추가하는 방법에 대한 자세한 내용은 JEUS Server 안내서의 "JEUS 설정"을 참고한다.

2.4.1. Basic 설정

다음은 domain.xml을 사용한 EJB 엔진의 Basic 설정 방법 예시이다.

EJB 엔진의 Basic 설정: <domain.xml>

```
<ejb-engine>
  <resolution>1000</resolution>
  <use-dynamic-proxy-for-ejb2>true</use-dynamic-proxy-for-ejb2>
  <enable-user-notify>false</enable-user-notify>
  <timer-service>
    <support-persistence>true</support-persistence>
    <max-retrial-count>1</max-retrial-count>
    <retrial-interval>5000</retrial-interval>
    <thread-pool>
      <min>2</min>
      <max>30</max>
      <period>3600000</period>
    </thread-pool>
  </timer-service>
</ejb-engine>
```

```

        </thread-pool>
    </timer-service>
    <async-service>
        <thread-min>0</thread-min>
        <thread-max>10</thread-max>
        <access-timeout>30000</access-timeout>
    </async-service>
</ejb-engine>

```

다음은 기본 정보 및 고급 선택사항의 영역별 설정 항목에 대한 설명이다.

• 기본 정보

항목	설명
Use Dynamic Proxy For Ejb2	기존 RMI Stub 방식 대신 Dynamic Proxy 방식을 사용한다.
Resolution	Active Management의 체크 주기와 Passivation 체크 주기를 설정한다. 즉, block된 스레드를 감지하는 주기와 Passivation 타임아웃 동안 클라이언트로부터 요청을 받지 않은 Bean을 감지하는 주기이다.

• 고급 선택사항

◦ Ejb Engine

항목	설명
Enable User Notify	옵션을 설정하면 EJB Exception이 서버에 설정한 user log에 남게 된다. user log에 대한 설명은 JEUS Server 안내서의 "Logging"을 참고한다.

◦ Async Service

Asynchronous Invocation Service를 위한 설정이다.

항목	설명
Thread Min	유지할 스레드 개수의 최솟값을 설정한다.
Thread Max	유지할 스레드 개수 최댓값을 설정한다.
Access Timeout	<p>Async 메소드가 수행이 완료된 후 일정 시간이 지나도 클라이언트에서 get을 하지 않으면 Future 객체를 삭제하는 시간이다.</p> <p>이는 클라이언트의 실수로 get을 하지 않는 경우 Memory Leak의 발생을 방지하기 위한 설정이다.</p>

◦ Invoke Http

EJB RMI Stub이 RMI 런타임 포트를 접근할 수 없는 상황일 경우에 설정한다.

항목	설명
Http Port	HTTP-RMI 요청을 받을 포트 번호를 설정한다. 웹 서버 및 웹 엔진에는 반드시 RMI 핸들러 서블릿이 deploy되어 실행되고 있어야 한다.
Url	<p>RMI Stub으로부터 호출되는 RMI 핸들러 서블릿의 URI를 입력한다.</p> <ul style="list-style-type: none"> ◦ URI에는 프로토콜, IP 주소, 포트를 제외한 서블릿 요청 경로만을 넣는다. 즉, rmiHandlerServlet.war의 jeus-web-dd.xml에 설정한 <context-path>와 web.xml에 설정한 <url-pattern>을 이어서 입력한다. jeus-web-dd.xml을 별도로 생성하지 않으면 <context-path>의 기본값인 war 이름으로 예에서는 rmiHandlerServlet이 된다. ◦ 프로토콜은 "HTTP"로 IP는 RMI 런타임과 동일한 주소로 간주된다. HTTP-RMI 요청을 받은 웹 서버와 웹 엔진이 RMI 런타임과 같은 머신에 있어야 한다. 그러면 RMI 런타임의 주소는 RMI Stub에게 알려지게 된다. 웹 서버의 포트는 반드시 다음 <http-port>에 설정해야 한다. ◦ JEUS에서 제공하는 rmiHandlerServlet.war에는 jeus-web-dd.xml이 포함되어 있지 않다. 따라서 기본적으로 컨텍스트는 모듈 이름과 같은 rmiHandlerServlet을 사용하게 된다. 또한 web.xml에는 <url-pattern>이 서블릿 핸들러가 설정되어 있다. 기본값 외의 설정을 사용하고 싶은 경우에는 jeus-web-dd.xml을 생성하고 web.xml을 수정한 후 rmiHandlerServlet.war를 deploy한다.

2.4.2. Active Management 설정

Active Management 설정은 엔진 재시작 조건들과 이메일 통보 기능으로 나뉘어진다. 엔진 재시작 조건은 EJB 엔진이 재시작하기 전까지의 허용 가능한 최대 block된 EJB 스레드 수로 결정된다.

다음은 domain.xml을 사용한 EJB 엔진의 Active Management 설정 방법 예시이다.

```
<ejb-engine>
  <resolution>1000</resolution>
  <use-dynamic-proxy-for-ejb2>true</use-dynamic-proxy-for-ejb2>
  <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>gw.tmaxsoft.com</smtp-host-address>
      <sender-id>sender@tmaxsoft.com</sender-id>
      <sender-password>1234567</sender-password>
      <from-address>sender@tmaxsoft.com</from-address>
      <to-address>receiver@tmaxsoft.com</to-address>
      <property>
        <key>mail.smtp.port</key>
        <value>587</value>
      </property>
      <property>
        <key>mail.smtp.auth</key>
        <value>true</value>
      </property>
    </email-notify>
  </active-management>
</ejb-engine>
```

```

    <property>
      <key>mail.smtp.starttls.enable</key>
      <value>true</value>
    </property>
  </email-notify>
</active-management>
</ejb-engine>

```

다음은 기본 정보 및 고급 선택사항의 영역별 설정 항목에 대한 설명이다.

• 기본 정보

항목	설명
Max Blocked Thread	<p>EJB 엔진이 재시작 권고하기 전까지 허용할 수 있는 block된 EJB 스레드의 최대 개수이다. 이 값이 작게 설정되어 있다면 EJB 엔진 재시작 권고가 너무 자주 될 수도 있기 때문에 주의해야 한다.</p> <p>기본값으로 설정하면 block된 스레드 개수에 대한 제한이 없음을 의미한다. 즉, 기본적으로 EJB 엔진은 block된 스레드 때문에 재시작 권고를 하지 않는다.</p>
Max Idle Time	<p>지정된 시간 동안 스레드가 block된 상태로 요청을 받지 않고 Idle 상태에 있으면 "block된 스레드 리스트"로 추가된다.</p> <p>이 설정은 엔진에서 block된 스레드로 판단하는 기준이 된다.</p>

• 고급 선택사항

◦ Email Notify

재시작 권고 조건에 따라 EJB 엔진이 시작될 경우에 이메일 통보를 보내기 위한 정보를 설정한다.

항목	설명
Smtp Host Address	메시지를 전송할 때 사용할 SMTP의 주소로 이 주소는 호스트 이름이나 IP 주소로 설정해야 한다.
Sender Id	SMTP 주소를 통해 인증 받을 ID를 설정한다.
Sender Password	SMTP 주소를 통해 인증 받을 ID의 암호를 설정한다.
From Address	메일 송신자의 주소를 설정한다.
To Address	메일 수신자의 주소를 설정한다.
Cc Address	메일을 참조로 받을 수신자의 주소를 설정한다.
Bcc Address	메일을 숨은 참조로 받을 수신자의 주소를 설정한다.
Property	기본적인 SMTP property 외에 추가로 필요한 javaMail API에서 지원하는 property가 있다면 key, value의 형태로 설정한다.

2.4.3. Timer Service 설정

EJB Timer Service는 EJB가 특정한 시간 또는 주기적으로 callback을 받을 수 있도록 하는 서비스이다. 기본적인

사용 방법은 EJB 스펙에 설명되어 있으므로 JEUS EJB에서 제공하는 Timer Service와 이를 사용하기 위한 설정에 대해서만 설명한다. 자세한 내용은 [EJB Timer Service](#)를 참고한다.

2.5. 시스템 로그 설정

System Logging과 User Logging 설정은 EJB 엔진뿐만 아니라 다른 모든 엔진에도 적용되는 공통적인 설정이므로 자세한 내용은 JEUS Server 안내서의 "Logging"을 참고한다.

2.6. EJB 엔진 제어 및 모니터링

EJB 엔진을 제어하는 것은 다른 JEUS 엔진(서블릿 또는 JMS)을 제어하는 것과 유사하다. 콘솔 툴을 사용하여 EJB 엔진의 실행 환경 정보와 상태 정보를 모니터링할 수 있다. 자세한 내용은 JEUS Reference 안내서의 "EJB 엔진 관련 명령어"를 참고한다.

2.7. EJB 엔진 튜닝

EJB 엔진의 전체적인 성능 향상을 위해 설정을 변경할 수 있다. 본 절에서는 EJB 엔진의 성능 관련 설정을 간략하게 설명한다.

튜닝을 위해 필요한 사항은 다음과 같다.

- Resolution 설정 튜닝
- Fast Deploy 기능 사용
- 최대 성능을 위한 시스템 로그 설정
- Active Management 사용하지 않기
- HTTP Invoke 모드 사용



EJB 엔진의 정보나 팁에 대해서는 "JEUS XML Reference"의 "11. domain.xml EJB 엔진 설정"을 참고한다. XML/Schema 중 "P"라고 표시된 것은 성능과 관련된 것이다.

2.7.1. Resolution 설정 튜닝

Resolution은 EJB 엔진의 상태를 체크하는 주기로 2가지 주기로 사용된다.

- block된 스레드 개수가 몇 개인지 검사한 후 EJB 엔진을 재시작하는 Active Management의 체크 주기를 나타낸다.
- 모든 하위 컴포넌트들(예: Bean pool)을 점검하고, 각 Bean이 비활성화 대상인지 검사하는 passivation의 체크 주기를 나타낸다.

Resolution 값이 클수록 시스템 메모리나 기타 리소스의 회수 주기가 길어져 자원 활용률은 떨어지지만 이에 대한 작업 수행이 덜 발생하므로 엔진의 성능은 향상된다. 이 값을 작게 하면 엔진은 최신 상태를 유지하겠지만 전체적인

성능은 저하된다. 따라서 메모리 크기나 용도에 따라서 Resolution 값을 적절하게 설정하는 것이 매우 중요하다.



Resolution 값의 설정에 따라 <passivation-timeout>이나 <disconnect-timeout>이 원하는 때에 발생하지 않을 수 있으므로 주의해야 한다.

2.7.2. Fast Deploy

Fast Deploy 옵션은 EJB에 설정하지 않고 애플리케이션별로 설정하는 것이지만 성능에 큰 영향을 미친다. 엔진이 기동될 때 deploy되어야 할 EJB 모듈들이 이미 컴파일되어 RMI Stub과 Skeleton이 있다면 **deploy** 명령어를 사용할 때 -fast 옵션을 추가한다. 이는 엔진이 EJB 모듈을 deploy할 때 RMI 클래스를 생성하지 않도록 하는 것이다. EJB 모듈과 deploy에 대한 자세한 내용은 [EJB 모듈](#)을 참고한다.

2.7.3. 최대 성능을 위한 시스템 로그 설정

시스템 로그는 성능 개선을 위해 3가지 방법으로 조정 가능하다.

- 가능하면 파일 핸들러를 사용해서 Logging이 빠르게 이루어지도록 하는게 좋다.
- 파일 핸들러의 버퍼 크기를 크게 설정한다.
- 로그 레벨을 'SEVERE'로 설정한다.



위의 제안은 안정적인 운영 환경에서만 적용되어야 한다. 개발 환경에서는 "반대"의 값들이 설정되어야 한다. 즉, 콘솔 핸들러를 사용하고, 작은 버퍼 크기를 사용하고, 'FINE' 로그 레벨을 사용하는 것이 개발을 용이하게 한다.

2.7.4. Active Management 사용하지 않기

EJB 엔진 레벨에 Active Management의 사용이 반드시 필요한 것은 아니다. 설정에 따라 성능 저하를 가져올 수 있기 때문에 기본적으로는 사용하지 않는다. 대신 서블릿 엔진에 정의된 Active Management를 사용하는 것이 편리하다. 그러므로 일반적으로 Active Management 설정은 생략하는 경우가 많다.

2.7.5. HTTP Invoke 모드 사용

클라이언트가 많은 경우 HTTP Invoke 모드를 사용하면 성능 향상 효과가 나타난다. 웹 서버가 JEUS로의 요청을 일정하게 조절해주기 때문에 클라이언트의 요청대로 스레드를 생성시키는 RMI에 비해 connection의 개수가 적게 생성된다. 그러나 보통의 경우에 HTTP 프로토콜로 변화하는 것은 오히려 성능 저하를 가져올 수 있다는 것을 고려해야 한다.

3. EJB 모듈

본 장에서는 EJB 모듈의 구조와 관리 방법에 대해서 설명한다.

3.1. 개요

EJB 모듈은 EJB 엔진에 deploy할 수 있는 기본 단위이다. 또한 EJB 컴포넌트들을 그룹화하고 이들의 설정 정보를 DD로 표현할 수 있도록 한 것이다. EJB를 deploy할 경우에는 단 하나의 EJB 컴포넌트를 deploy하더라도 반드시 EJB 모듈로 패키징해야 한다.



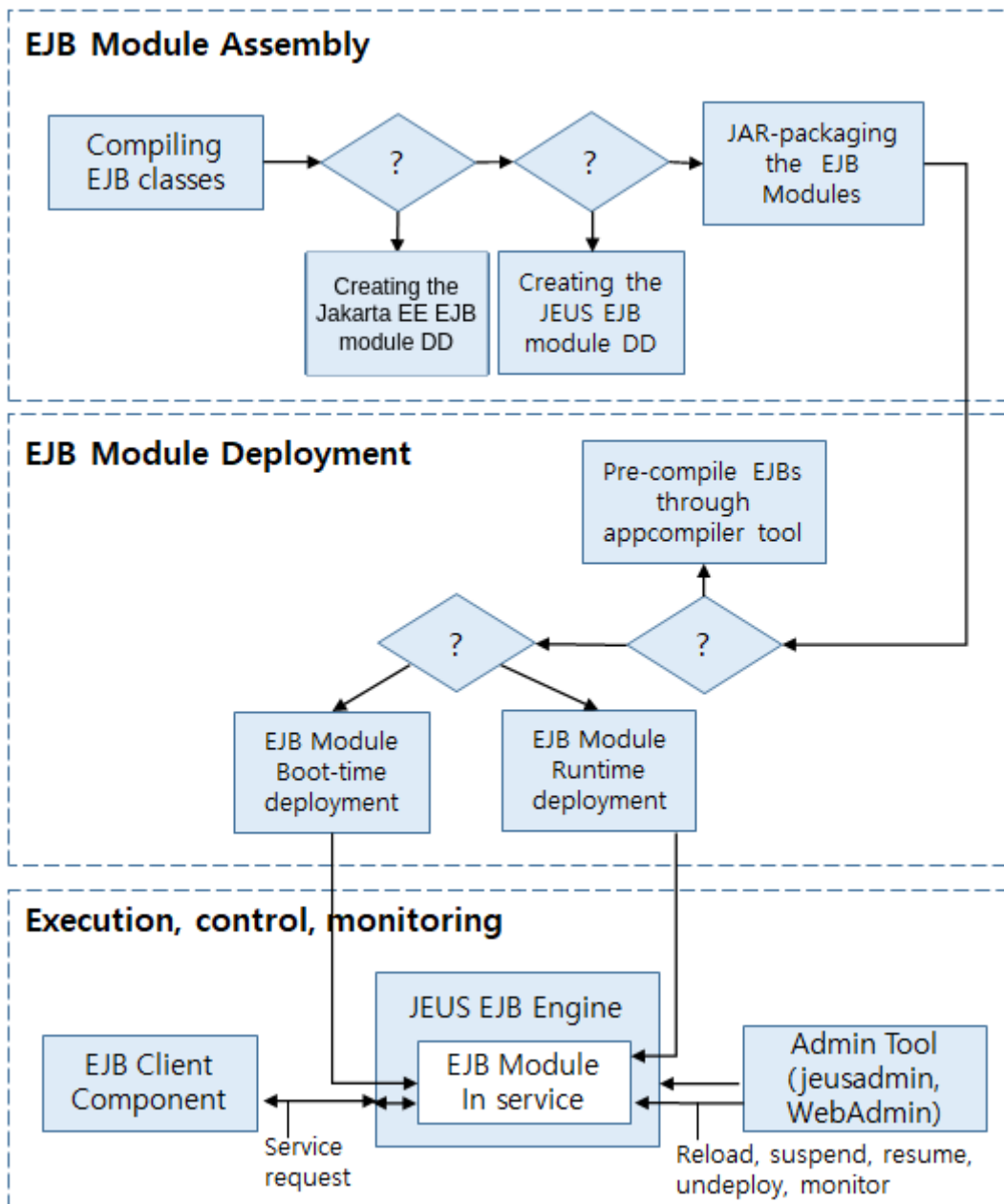
EJB 2.1 이하의 컴포넌트인 경우에는 EJB 모듈에 반드시 DD(ejb-jar.xml)가 있어야 하지만 3.0 이상의 컴포넌트부터는 Annotation을 지원하게 되면서 DD는 선택 사항으로 변경되었다.

3.2. EJB 모듈 관리

다음은 JEUS 내부적으로 EJB 모듈을 관리하기 위한 4가지 주요 작업으로 각 작업에 대한 상세한 설명은 세부 절을 참고한다.

- assembly : [EJB 모듈 조립](#)
- deployment : [EJB 모듈 Deploy](#)
- control(undeploy, redeploy, stop-application, start-application) : [EJB 모듈 제어](#)
- monitoring : [EJB 모듈 모니터링](#)

EJB 모듈의 관리 순서는 다음과 같다.



EJB 모듈 관리 순서도

EJB 모듈 JAR 파일 구조

다음은 표준 EJB 모듈 JAR 파일의 구조에 대한 설명이다.

```
EJB Module JAR
|--META-INF
|   |--[X]ejb-jar.xml
|   |--[X]jeus-ejb-dd.xml
|--<package_name>
|   |--[C]Business Interface
|   |--[C]EJB Implementation
|   |--[C]Helper class
|--[J]Helper library
```

* Legend

- [01]: binary or executable file
- [X] : XML document

- [J] : JAR file
- [T] : Text file
- [C] : Class file
- [V] : java source file
- [DD] : deployment descriptor

META-INF

다음은 하위 파일에 대한 설명이다.

파일	설명
ejb-jar.xml	실제적인 표준 EJB DD 파일이다. Annotation으로 설정 정보를 표현한 경우에는 존재하지 않을 수 있다. 단, EJB 2.x 스타일의 경우에는 반드시 필요하다.
jeus-ejb-dd.xml	JEUS에서 EJB를 deploy할 때 필요한 DD 파일이다. (선택 사항)

<package_name>

EJB 클래스들을 포함하고 있다. EJB 소스 코드에 정의되어 있는 것과 같이 Java 패키지 구조를 반영하는 구조로 되어 있어야 한다. 예를 들어 하나의 EJB가 "mypackage"라는 패키지에 속해 있다고 선언되어 있으면 EJB의 클래스는 반드시 EJB JAR의 "mypackage"라는 디렉터리 아래에 위치해야 한다.

다음과 같은 EJB 컴포넌트를 포함한다.

EJB 컴포넌트	설명
인터페이스	클라이언트가 접근하기 위한 EJB 2.x 스타일의 인터페이스, 3.0 이상의 인터페이스들이다.
엔터프라이즈 Bean 클래스	인터페이스에 맞춰서 실제 업무 로직을 구현한 Bean 클래스들을 의미한다.

Helper library

MANIFEST.MF Class-Path Entry에 명시한 라이브러리에 해당한다.

3.3. EJB 모듈 조립

EJB 모듈 관리 순서도와 같이 EJB 모듈을 조립(Assembling)할 때는 다음의 과정을 수행한다.

1. EJB 클래스 컴파일

개발자가 구현한 EJB 소스 파일들을 컴파일한다.

2. DD 작성

- ejb-jar.xml DD 작성
- jeus-ejb-dd.xml DD 작성

3. EJB JAR 파일 패키징

JAR 파일로 EJB 클래스와 DD를 패키징한다.

다음 절부터는 "counter" 예제를 통해 EJB 모듈 조립 과정을 설명한다. "counter"는 JEUS_HOME/samples/ejb/basic/statefulSession/ 디렉터리에 위치한 Stateful Session Bean으로, 다음과 같은 2개의 Java 소스 파일이 갖는다.

- Counter.java(business interface)
- CounterEJB.java(bean implementation)



Stateful Session Bean이나 다른 종류의 Bean에 대한 자세한 정보는 [Session Bean](#), [Message Driven Bean\(MDB\)](#)을 참고한다.

3.3.1. EJB 클래스 컴파일

EJB 소스 파일들로부터 EJB 모듈을 조립하는 첫 번째 과정은 개발자가 구현한 EJB 소스 파일들을 컴파일하는 것이다. 이는 JDK의 **javac** 컴파일러를 이용한다. 클래스 패스는 기본적으로 JEUS_HOME/lib/system/jakarta.ejb-api.jar를 사용하며, jeus-ejb-dd.xml에 대응되는 Annotation을 사용했다면 JEUS_HOME/lib/system/jeusapi.jar를 추가하고, 또 다른 helper 클래스가 있다면 모두 추가한다.

다음은 "counter" Bean에서 사용될 EJB Java 파일들을 UNIX 환경에서 컴파일하는 명령어를 실행하는 예이다.

```
$ javac -classpath JEUS_HOME/lib/system/jakarta.ejb-api.jar *.java
```

위 명령어를 실행하면 클래스 파일들이 생성된다.

3.3.2. Deployment Descriptors(DD) 작성

EJB 모듈을 deploy할 때 사용되는 DD는 다음과 같이 2가지로 나누어진다.

- 표준 EJB DD

ejb-jar.xml로 명명되고, EJB 모듈의 "META-INF/" 디렉터리에 위치한다. 개발자가 EJB 컴포넌트를 작성할 때는 설정 정보의 일부 또는 전부를 Annotation으로 작성할 수도 있다. 만약 Annotation 정보를 무시하고 DD의 설정만을 적용하고 싶은 경우에는 ejb-jar.xml에 <metadata-complete>를 true로 설정한다. Annotation과 DD를 혼용해서 사용한 경우는 DD가 Annotation에 우선한다.

- JEUS EJB DD

jeus-ejb-dd.xml로 명명되고, EJB 모듈의 "META-INF/" 디렉터리에 위치한다. 이 파일의 내용에 대한 정보는 아래의 [jeus-ejb-dd.xml DD 작성](#)을 참고한다.

표준 EJB DD(ejb-jar.xml) 작성

DD의 포맷은 EJB 표준에 따른다. 대부분 Annotation으로 가능하지만 필요한 경우 DD를 사용한다.

```
package ejb.basic.statefulSession;
```

```
@Remote
public interface Counter
{
    public void increase();
    ...
}
```

```
package ejb.basic.statefulSession;
@Stateful
@EJB(name="counter")
@TransactionManagement( TransactionManagementType.BEAN)
public class CounterEJB implements Counter {
    private int count = 0;

    public void increase()
    {
        count++;
    }
    ...
}
```

다음 예제는 Annotation을 "counter" Bean의 간단한 EJB 표준 DD(ejb-jar.xml)를 나타낸 것이다.

EJB 표준 DD : <ejb-jar.xml>

```
<?xml version="1.0"?>
<ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd">
    <enterprise-beans>
        <session>
            <ejb-name>counter</ejb-name>
            <business-remote>ejb.basic.statefulSession.Counter</business-remote>
            <ejb-class>ejb.basic.statefulSession.CounterEJB</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Bean</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor/>
</ejb-jar>
```



EJB 3.0 이상부터는 ejb-jar.xml의 모든 내용은 Annotation으로 표현 가능하므로 선택적으로 사용한다.

JEUS EJB DD(jeus-ejb-dd.xml) 작성

JEUS EJB DD가 필요한 이유는 EJB 엔진에 deploy하기 위한 기본 설정 및 External Reference들을 JEUS 서비스에 매핑시키기 위함이다. 그 외에도 인스턴스 Pooling과 Entity Bean, DB 테이블과의 매핑 설정도 jeus-ejb-dd.xml에서 한다.



이러한 설정들은 EJB 모듈을 JEUS에 맞추어 deploy하기 위해서 필요한 작업들이지만 일부는 Annotation으로 설정 가능하고 jeus-ejb-dd.xml이 존재하지 않아도 기본값으로 설정되기 때문에 별도로 환경설정을 해야 하는 경우에만 이 작업을 수행한다.

JEUS EJB DD는 표준 EJB DD에 추가적인 설정으로 볼 수 있다. 표준 EJB DD와 마찬가지로 JEUS EJB DD도 EJB 모듈에 적용된다.

다음은 JEUS EJB DD의 전반적인 구성이다.

JEUS EJB DD : <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 "4.2.5. EJB 보안 설정" -->
      . . .
    </role-permission>
  </module-info>
  <beanlist>
    <jeusbean>
      <!-- See chapters "제4장 EJB의 공통 특성"
            and "제7장 Session Bean" -->
      . . .
    </jeusbean>
    <jeusbean>
      <!-- See chapters "제4장 EJB의 공통 특성"
            and "제7장 Session Bean" -->
      . . .
    </jeusbean>
    . . .
  </beanlist>
  <ejb-relation-map>
    <!-- See chapter "제8장 Entity Bean" -->
  </ejb-relation-map>
  <message-destination>
    <jndi-info>
      <ref-name>ejb/AccountEJB</ref-name>
      <export-name>ACCEJB</export-name>
    </jndi-info>
  </message-destination>

  <library-ref>
    <library-name>jsf</library-name>
    <specification-version>
      <value>1.2</value>
    </specification-version>
    <implementation-version>
      <value>1.2</value>
    </implementation-version>
  </library-ref>

</jeus-ejb-dd>
```


JEUS EJB DD의 주요 요소들은 다음과 같다.

항목	설명
<module-info>	EJB 모듈 전체에 적용되는 포괄적인 정보를 설정한다. Role Assignment에 관련된 정보는 EJB 보안 설정 을 참고한다.
<beanlist>	모듈에 포함되는 EJB들을 선언한다. 이 하위 태그의 이름과 내용은 Bean의 종류(Session Bean, Entity Bean, MDB)에 따라 세부 설정이 달라진다. <beanlist>와 그것의 하위 태그는 EJB의 공통 특성 , Session Bean 부터 Message Driven Bean(MDB) 을 통해 자세히 설명한다.
<ejb-relation-map>	CMP 2.0 Bean들을 위한 CMR 매핑을 정의한 EJB Relation Mapping 설정이다. CMP는 더 이상 사용을 권장하지 않으므로 JPA를 사용한다.
<message-destination>	ejb-jar.xml의 <message-destination>에 선언된 message destination과 JNDI에 등록된 실제 Destination 객체를 매핑한다.
<library-ref>	애플리케이션에서 사용할 공유 라이브러리(shared library) 정보를 설정한다.

3.3.3. EJB JAR 파일 패키징

EJB JAR 파일을 패키징할 때에는 **jar** 유틸리티를 사용하고, 다음의 사항을 가정한다.

- ejb/basic/statefulSession 디렉터리에 "counter" EJB 클래스 파일들이 존재한다.
- 클래스에 Annotation으로 설정하여 ejb-jar.xml과 jeus-ejb-dd.xml은 존재하지 않는다.

EJB JAR 파일의 패키징 과정은 다음과 같다.

1. 다음과 같이 EJB 클래스들을 하나의 JAR 파일로 묶는다.

```
$ jar cf countermod.jar ejb/basic/statefulSession
```

이것으로 countermod.jar라는 Jakarta EE EJB 모듈을 완성한다.

2. 클라이언트에서 이 EJB를 사용하기 위해서는 인터페이스 파일이 필요하다. 클라이언트에서 사용하기 위해 배포할 counterbeanclient.jar는 다음과 같이 생성한다.

```
$ jar cf counterbeanclient.jar ejb/basic/statefulSession/Counter.class
```



counterbeanclient.jar는 원격 클라이언트에서 사용하기 위한 배포용 라이브러리이다.

3.4. EJB 모듈 Deploy

본 절에서는 EJB 모듈의 deploy에 대한 내용을 설명한다.

3.4.1. Deploy

일반적으로는 deploy할 때 EJB 클래스를 컴파일하지만 **appcompiler**를 사용해서 EJB 클래스를 미리 컴파일하면 deploy할 때의 속도를 향상시킬 수 있다. 또한 Auto Deploy 기능을 사용해서 손쉽게 EJB 모듈을 deploy할 수도 있다. Deploy에 대한 자세한 내용은 "JEUS Applications & Deployment 안내서"를 참고한다.

EJB 모듈 관리 순서도에서 JEUS는 다음과 같은 방법으로 EJB 모듈을 deploy한다.

- appcompiler 툴 이용

appcompiler 툴을 사용하면 Stub과 Skeleton 파일들을 미리 생성하여 Deploy 속도를 빠르게 할 수 있다.

- EJB 모듈의 Boot-time Deploy

EJB 엔진이 시작할 때마다 EJB 모듈이 자동으로 deploy되도록 한다.

- EJB 모듈의 Runtime Deploy

EJB 엔진이 시작되고 난 후에 즉, EJB 엔진이 운영되고 있을 때 EJB 모듈을 deploy한다. 콘솔 툴을 사용해서 EJB 모듈을 EJB 엔진에 deploy한다.

appcompiler 툴 이용

EJB 2.x 모듈의 경우 appcompiler 툴을 사용함으로써 Stub과 Skeleton 파일들을 미리 생성하여 Deploy 속도를 빠르게 할 수 있다. appcompiler를 사용할 경우에는 콘솔 툴의 deploy 명령어를 실행할 때 -fast 옵션을 사용한다. 이 옵션이 사용되지 않을 경우에는 appcompiler는 효과가 없다. appcompiler 툴에 대한 자세한 정보는 JEUS Reference 안내서의 "appcompiler"를 참조한다.

EJB 모듈의 Boot-time Deploy

MS가 기동(Booting)될 때 deploy되는 것을 Boot-time Deploy라고 하는데, MASTER를 통해 한 번 deploy된 애플리케이션은 MS를 기동할 때마다 Boot-time Deploy가된다. 애플리케이션의 Deploy 설정에 대한 자세한 내용은 JEUS Applications & Deployment 안내서의 "애플리케이션 작성 및 Deploy"를 참고한다.

하나의 도메인에 여러 개의 애플리케이션을 등록할 수 있다.

EJB 모듈의 Runtime Deploy

Runtime deploy는 실행되고 있는 EJB 엔진에 EJB 모듈을 deploy하는 것을 의미한다. 콘솔 툴의 **deploy** 명령어를 사용해서 실행되고 있는 EJB 엔진에 모듈을 설치할 수 있다.

"countermod" EJB 모듈의 deploy 과정을 설명한다. 모듈을 deploy하기 위해 다음의 사항을 가정한다.

- JEUS의 adminServer이라는 MASTER와 server1이라는 MS가 이미 기동(Booting)되어 있다.
- countermod 모듈이 JEUS_HOME/apphome 하위의 countermod.jar나 countermod 디렉터리 형태로 존재한다.

EJB 모듈의 Runtime Deploy는 콘솔 툴을 사용할 수 있다.

• 콘솔 툴 사용

콘솔 툴을 사용하여 deploy하는 과정은 다음과 같다.

1. 콘솔 툴을 실행한다. 콘솔 툴에 대한 자세한 설명은 JEUS Reference 안내서의 "Local 명령어"를 참고한다.

```
$ jeusadmin -host localhost:9736
```

2. 사용자 이름과 패스워드를 입력한다.

```
User name: administrator
Password:
Attempting to connect to localhost:9736.
The connection has been established to Domain Administration Server adminServer
in the domain domain1.
JEUS8 Administration Tool
To view help, use the 'help' command.
[MASTER]domain1.adminServer>
```

3. **deploy** 명령어를 실행해서 "countermod" EJB 모듈을 deploy한다.

```
[MASTER]domain1.adminServer> deploy countermod -servers server1
```

4. deploy를 확인하기 위해 다음과 같이 **application-info** 명령어를 실행하면 "countermod" 모듈이 포함된 모든 deploy된 애플리케이션들이 출력된다.

```
[MASTER]domain1.adminServer> application-info
```

3.4.2. Deploy된 EJB 모듈의 디렉터리 구조

EJB 모듈이 deploy된 후에 클래스 파일들과 설정 파일들은 다음과 같은 구조로 JEUS 설치 디렉터리에 배포된다.

```
{JEUS_HOME}
|--domain/<domain_name>/servers/<server_name>
|  |--.workspace/deployed
|    |--<application_id>
|      |--<EJB_archive_file>_jar__
|      |  |--META-INF
|      |    |--[X]Jakarta EJB DD
|      |    |--[X]JEUS EJB DD
|      |    |--[C]Bussiness Interfaxe(Home Interface, Component Interface)
|      |    |--[C]EJB Implementation
|      |    |--[C]Helper calss
|      |--[J]<EJB_archive_file>
|      |--<EAR_archive_file>_ear__
|      |--APP-INF
```

```

|--lib
|
|--META-INF
|
|--<EJB_archive_file>_jar__
|
|   |--META-INF
|   |
|   |   |--[X]Jakarta EJB DD
|   |   |--[X]JEUS EJB DD
|   |   |--[C]Bussiness Interfaxe(Home Interface, Component Interface)
|   |   |--[C]EJB Implementation
|   |   |--[C]Helper calss
|   |
|   |--[J]<EJB_archive_file>

```

* Legend

- [01]: binary or executable file
- [X] : XML document
- [J] : JAR file
- [T] : Text file
- [C] : Class file
- [V] : java source file
- [DD] : deployment descriptor

JEUS_HOME/domains/<domain_name>/servers/<server_name>/workspace/deployed/ 디렉터리는 <server name>에 해당하는 MS에 deploy된 애플리케이션들이 존재하는 디렉터리이다. 여기에는 deploy된 EJB 모듈의 EJB 구현 클래스들뿐만 아니라 helper 클래스들이 위치한다.

Deploy 방식은 다음과 같이 나누어지며, 방식에 따라 deploy되는 파일이 위치하는 디렉터리가 달라진다.

Deploy 방식	설명
EAR Deploy 방식	EAR 파일이 deploy되는 경우에는 해당 EAR 파일 이름으로 생성된 디렉터리 아래에 EAR에 포함된 모듈이 각각 놓인다.
Standalone Deploy 방식	JAR 파일이 deploy된 경우에는 해당 JAR 파일 이름으로 생성된 디렉터리에 "<jar 파일 이름>_jar__" 디렉터리 아래에 놓인다.
Exploded EAR Deploy 방식	EAR 파일을 Archive 형태가 아닌 풀어놓은 상태에서 deploy하는 방식이다.
Exploded Standalone Deploy 방식	JAR 파일을 Archive 형태가 아닌 풀어놓은 상태에서 deploy하는 방식이다.

Exploded EAR Deploy 방식과 Exploded Standalone Deploy 방식은 webhome 디렉터리로 복사되지 않고 원래의 위치를 참조한다. JEUS에서는 Archive 파일이 존재하는 standalone Deployment를 **COMPONENT Type**, 풀어놓은 것을 **EXPLODED COMPONENT Type**이라고 부른다.



이후의 설명에서는 Standalone 모듈에 대해서만 설명한다. EAR Deploy 방식이나 Deploy 전반에 대한 상세한 설명은 "JEUS Applications & Deployment 안내서"의 Deploy 관련 부분을 참고한다.

3.5. EJB 모듈 제어 및 모니터링

Deploy된 EJB 모듈에 대해 콘솔 툴을 사용해서 실행 상태를 제어하거나 모니터링할 수 있다. ([EJB 모듈 관리 순서도](#) 참고)

3.5.1. EJB 모듈 제어

Deploy된 EJB 모듈을 콘솔 툴의 stop-application, start-application, redeploy-application, undeploy 명령어를 사용해서 EJB 모듈 내의 실행 상태를 제어할 수 있다.

콘솔 툴 사용

Deploy된 EJB 모듈을 stop-application, start-application, redeploy-application, undeploy 명령을 통해 EJB 모듈 내의 실행 상태를 제어할 수 있다. 모든 명령어는 EJB 모듈의 ID를 파라미터로 받는다. 콘솔 툴에 대한 상세한 정보는 JEUS Reference 안내서의 "jeusadmin"을 참고한다.

- **redeploy-application**

redeploy-application은 지정한 EJB 모듈을 다시 로딩하는 명령어로 현재 활성화되어 있는 EJB 모듈을 EJB 실행 환경에서 undeploy했다가 다시 디스크로부터 읽어 deploy하는 것을 의미한다.

undeploy가 실행되고 다시 로딩이 수행될 때 undeploy된 EJB 모듈의 모든 트랜잭션 상태는 잃어버린다. 이는 undeploy되거나 redeploy되는 모듈과 연관있는 모든 EJB 트랜잭션이 rollback된다는 것을 의미한다.

```
redeploy-application <application-id>
```

- **stop-application**

stop-application 명령어는 선택된 EJB 모듈을 일시적으로 중지시켜 클라이언트의 요청으로부터 한시적으로 EJB 모듈이 접근이 불가능한 상태가 된다. start-application 명령어로 다시 EJB 모듈을 접근 가능한 상태로 되돌린다.

```
stop-application <application-id>
```

- **start-application**

stop-application 명령어로 중지되었던 EJB 모듈을 다시 활성화한다.

```
start-application <application-id>
```

- **undeploy**

undeploy 명령어는 선택된 모듈을 EJB 엔진의 실행 메모리에서 제거해서 EJB 엔진의 운영 메모리에서 EJB 모듈을 내려 EJB 클라이언트가 EJB에 접근 불가능한 상태가 된다. 그러나 파일을 물리적으로 삭제하지는 않는다.

```
undeploy <application-id>
```

다음은 콘솔 툴을 사용하여 EJB 모듈을 undeploy하는 과정에 대한 예이다.

1. 콘솔 툴을 실행한다.

```
$ jeusadmin -host localhost:9736
```

2. 사용자 이름과 패스워드를 입력한다.

```
User name: administrator
Password:
Attempting to connect to localhost:9736.
The connection has been established to Domain Administration Server adminServer
in the domain domain1.
JEUS8 Administration Tool
To view help, use the 'help' command.
[MASTER]domain1.adminServer>
```

3. 다음과 같이 **undeploy** 명령어를 실행하고 성공하면 "countermod" EJB 모듈이 undeploy된다.

```
[MASTER]domain1.adminServer> undeploy countermod
```

4. undeploy된 것을 확인하기 위해 다음과 같이 **application-info** 명령어를 실행한다. "countermod" 모듈을 제외한 deploy된 모든 모듈들이 조회된다.

```
[MASTER]domain1.adminServer> application-info
```

3.5.2. EJB 모듈 모니터링

콘솔 툴을 사용하여 EJB 모듈을 모니터링할 수 있다. 콘솔 툴에서 **application-info** 명령어를 실행하면 다음과 같이 deploy되고 활성화된 EJB 모듈로부터 상태와 운영 정보를 조회할 수 있다. application-info 명령어에 대한 자세한 내용은 JEUS Reference 안내서의 "EJB 엔진 관련 명령어"를 참고한다.

- 모듈 정보 조회

해당 모듈에 대한 정보를 조회한다.

```
[MASTER]domain1.adminServer>application-info -server server1 -id countermod -detail
General information about the EJB module [countermod].
=====
+-----+-----+
| Module Name | Unique Module Name |
+-----+-----+
```

```
| countermod | countermod |
+-----+-----+
=====
```

Beans

```
=====
+-----+-----+-----+-----+
| Bean Name |      Type      | Local Export Name | Remote Export Name |
+-----+-----+-----+-----+
| Count     | StatelessSessionBean |                   | Count               |
+-----+-----+-----+-----+
=====
```

• Bean 조회

해당 모듈("application-id")에 포함된 EJB의 목록을 조회한다.

```
[MASTER]domain1.adminServer>application-info -server server1 -id countermod -bean Count
```

Module name : countermod

Bean name : Count

```
=====
+-----+-----+-----+-----+-----+
|      Name      | (Count) | WaterMark(High:Low | Bound(Upper:L | Time(Max:Min:T |
|                 |         | :Cur)             | ower)         | otal)         |
+-----+-----+-----+-----+-----+
| create         | times(0) |                   |               |               |
+-----+-----+-----+-----+-----+
| comitted       | transactio |                   |               |               |
|                 | n(0)       |                   |               |               |
+-----+-----+-----+-----+-----+
| total-remote-t |           | thread(100:100:100) |               |               |
| hread          |           |                   |               |               |
+-----+-----+-----+-----+-----+
| timed-rb       | transactio |                   |               |               |
|                 | n(0)       |                   |               |               |
+-----+-----+-----+-----+-----+
| remove         | times(0) |                   |               |               |
+-----+-----+-----+-----+-----+
| active-bean    |           | bean(0:0:0)        |               |               |
+-----+-----+-----+-----+-----+
| request        | request(0) |                   |               |               |
+-----+-----+-----+-----+-----+
| total-bean     |           | bean(0:0:0)        |               |               |
+-----+-----+-----+-----+-----+
| rolledback     | transactio |                   |               |               |
|                 | n(0)       |                   |               |               |
+-----+-----+-----+-----+-----+
| active-thread  |           | thread(0:0:0)      |               |               |
+-----+-----+-----+-----+-----+
| MethodReadyCou |           | bean(0:0:0)        |               |               |
| nt             |           |                   |               |               |
+-----+-----+-----+-----+-----+
```



EJB 모듈의 모니터링은 콘솔 툴을 사용할 경우에 자세한 정보를 조회할 수 있다. 모니터링하는 자세한 방법은 각각 JEUS Server 안내서의 "바인딩된 객체 확인"과 본 안내서의 [Timer 모니터링](#)을 참고한다.

4. EJB의 공통 특성

본 장에서는 JEUS에서의 EJB 공통 특성에 대해 설명한다.

EJB 표준에 언급된 내용은 설명하지 않고, JEUS에서 EJB를 사용하기 위해 필요한 추가적인 정보들만 제공한다. 또한 EJB deploy에 대해 언급하지 않으므로 [EJB 모듈](#)의 내용을 먼저 참고한다.

4.1. 개요

본 절에서는 EJB의 종류와 각 종류별 DD 설정에 대해서 설명한다.

EJB 종류

JEUS는 EJB 스펙에 따라 다음의 EJB들을 지원한다.

- Stateless Session Bean : [Session Bean](#)
- Stateful Session Bean : [Session Bean](#)
- BMP Entity Bean : [Entity Bean](#)
- CMP 1.1 & CMP 2.x Entity Bean : [Entity Bean](#)
- Message-Driven Bean : [Message Driven Bean\(MDB\)](#)

본 장에서는 모든 Bean들에 적용되는 공통적인 특징과 설정에 대해서 설명한다. 각 EJB에 대한 추가적인 상세 정보는 위에 나열된 해당 장을 참고한다.



Entity Bean은 EJB 3.0부터는 JPA로 대체되었으므로 이에 대해서는 "JEUS JPA 안내서"를 참고한다. 따라서 Entity Bean의 사용을 권장하지 않지만 하위 호환성을 위해 계속 지원하고 있다.

EJB 종류별 DD 설정

JEUS EJB DD(Deployment Descriptors)는 jeus-ejb-dd.xml로 명명된 XML 파일이며, EJB JAR 파일의 META-INF 하위에 존재한다. JEUS EJB DD에 대한 간단한 내용은 [EJB 모듈](#)을 참고한다.

다음은 JEUS EJB DD로 설정 가능한 기능들과 컴포넌트들의 간략한 정보를 나타낸다("@" 표시는 Annotation으로도 설정 가능한 개체이다). 표의 왼쪽에 나열된 항목들은 실제 설정 가능한 JEUS EJB DD의 XML 태그들이다. 표의 나머지 6개 열은 EJB 종류에 따른 설정 사항들을 나타낸다.

설정 가능한 개체	Stateless	Stateful	BMP	CMP1.1	CMP2.0	MDB
1. EJB name @	√	√	√	√	√	√
2. Export name @	√	√	√	√	√	
3. Local export name @	√	√	√	√	√	

설정 가능한 개체	Stateless	Stateful	BMP	CMP1.1	CMP2.0	MDB
4. Export port	✓	✓	✓	✓	✓	
5. Export IIOP switch	✓	✓	✓	✓	✓	
6. use-access-control	✓	✓	✓	✓	✓	✓
7. Run-as Identify	✓	✓	✓	✓	✓	✓
8. Security CSI Interop.	✓	✓	✓	✓	✓	✓
9. Env. Refs @	✓	✓	✓	✓	✓	✓
10. EJB refs @	✓	✓	✓	✓	✓	✓
11. Resource Refs @	✓	✓	✓	✓	✓	✓
12. Resource env. Refs @	✓	✓	✓	✓	✓	✓
13. Thread ticket pool settings	✓	✓	✓	✓	✓	✓
14. Clustering settings @	✓	✓	✓	✓	✓	
15. HTTP Invoke	✓	✓	✓	✓	✓	
16. Pooling Bean Switch		✓				
17. Object management	✓	✓	✓	✓	✓	✓
18. Persistence Optimize			✓	✓	✓	
19. CM persistence opt.				✓	✓	
20. Schema info				✓	✓	
21. Relationship map					✓	
22. Connect. fact. name						✓
23. MDB resource adaptor						✓
24. Destination @						✓
25. JNDI SPI settings						✓
26. Max messages						✓
27. Activation Config @						✓
28. Durable Timer Service	✓		✓	✓	✓	✓

위의 표는 다음과 같이 나누어 진다.

- 항목 1부터 16까지는 6개의 EJB 종류 모두에 공통으로 적용되는 사항이다(MDB의 2~5과 14~16은 제외). 이 모든 항목들에 대해서 본 장에서 설명한다.
- 항목 7(Run-as Identify)과 8(security interoperability), 14(clustering)은 모든 EJB 종류에 공통이지만, 각각 [EJB 보안 설정](#), [EJB 상호 운용성 및 RMI/IIOP](#), [EJB 클러스터링](#)에서 별도로 설명한다.
- 항목 17은 Stateful Session Bean에 대해서만 적용되며, [Session Bean](#)에서 자세히 설명한다.
- 항목 18(object management)은 6개의 EJB 종류 모두에 공통으로 적용되는 사항이나 Bean의 종류에 따라 조금씩 다르므로 기본적인 것은 [Session Bean](#)에서 설명하고 각 장에서 차이점을 설명한다.

- 항목 19부터 22까지는 Entity Bean에만 적용된다. 이들은 [Entity Bean](#)에서 설명한다.
- 항목 23부터 28까지는 MDB에만 적용된다. 이들은 [Message Driven Bean\(MDB\)](#)에서 설명한다.
- 항목 29는 EJB 스펙에서 제공하는 타이머 서비스를 사용할 수 있는 Bean(Stateful Session Bean을 제외한 모든 Bean)에 사용할 수 있다. 이는 [EJB Timer Service](#)에서 설명한다.

4.2. EJB 설정

모든 EJB 설정들은 JEUS EJB DD 파일인 jeus-ejb-dd.xml 파일 내에 설정된다. 상위 XML 태그는 Bean의 종류에 관계없이 <jeus-bean>을 사용한다.

다음 절에서는 위의 5종류의 EJB XML 부모 태그들에 적용될 수 있는 설정들을 설명한다. <beanlist> 밖에 있는 다른 태그들은 [EJB 모듈](#)에서 이미 설명하였다. 특정 XML 부모 태그(Bea 종류)에 대한 설명은 이 문서의 후반부에 나오는 해당 장들을 참고한다.



클러스터링 환경설정은 [EJB 클러스터링](#)을 참고한다.

4.2.1. 기본 환경설정

다음은 주로 사용되는 JEUS 전용 설정들이며 jeus-ejb-dd.xml에 선언되어 있다. 이 파일에 대해서는 [EJB 모듈](#)을 참고한다.

- **<ejb-name>**

- 표준 ejb-jar.xml에서 <ejb-name> 또는 구현 클래스의 @Stateless, @Stateful, @MessageDriven의 "name" 값과 동일한 이름이다.

- **<export-name>**

- 글로벌 JNDI에 등록될 시스템 내부에서 유일한 이름을 가져야 한다.
- DD 파일에 이 element가 없는 경우 Annotation @Stateless나 @Stateful의 mappedName attribute 값이 적용되거나, ejb-jar.xml의 <mapped-name>값이 적용된다.

이 모든 설정이 없는 경우는 기본값으로 Remote Business Interface name이 사용된다. 단, Remote Business Interface가 하나만 존재하고 Remote Home Interface가 존재하지 하는 경우나 Remote Business Interface가 존재하지 않고 Remote Home Interface가 존재하는 경우에만 기본값이 적용되고 아닌 경우에는 JNDI name을 결정할 수 없어 deploy가 실패한다.

- 순위는 다음과 같다.

1. jeus-ejb-dd.xml의 <export-name>
2. ejb-jar의 <mapped-name>
3. @EJB의 mappedName

- **<local-export-name>**

- JNDI에 등록될 시스템 내부의 유일한 이름을 가져야 한다.
- DD 파일에 이 element가 없는 경우 Annotation @EJB mappedName attribute 값에 적용된다. 만약

Remote Business Interface가 존재하는 경우에는 이 값에 '_Local'이 붙는다.

둘 다 없는 경우는 기본값으로 Local Business Interface name이 사용된다. 단, Local Business Interface가 하나만 존재하고 Local Home Interface가 존재하지 않는 경우나 Local Business Interface가 존재하지 않고 Local Home Interface가 존재하는 경우에만 기본값이 적용되고 아닌 경우에는 JNDI name을 찾을 수 없어 Exception이 발생한다.

순위는 다음과 같다.

1. jeus-ejb-dd.xml의 <local-export-name>
2. Remote Business Interface가 존재하면 ejb-jar.xml의 <mapped-name>+ _Local
Remote Business Interface가 존재하지 않으면 ejb-jar.xml의 <mapped-name>
3. Remote Business Interface가 존재하면 @EJB의 mappedName + _Local
Remote Business Interface가 존재하지 않으면 @EJB의 mappedName

• <export-port>

- RMI 서비스 포트를 명시적으로 설정할 필요가 있을 때에 이 설정을 사용한다.
- 방화벽을 사용하고 있어서 특정 포트만 시스템 접근을 허용할 경우에 유용하다. 이 태그의 설정은 방화벽 설정의 "allowed"로 설정된 포트와 동일한 것이어야 한다. 값을 지정하지 않는 경우는 시스템 프로퍼티에 따라 일괄적으로 포트가 적용되는데 정리하면 다음과 같다.

항목	설명
jeus.ejb.export Port	jeus-ejb-dd에 특정 EJB에 대한 <export-port>가 지정되어 있지 않는 나머지 EJB의 export port를 결정한다.
jeus.rmi.usebaseport	<ul style="list-style-type: none">◦ true: Managed Server base port를 사용한다.◦ false: Managed Server base port + 7을 사용한다.

• <export-iiop>

- 활성화되어 있을 경우에 Bean의 인터페이스가 IIOP Stub과 Skeleton으로서 COS Naming Server에 export될 수 있도록 한다. 이는 IIOP로 접근 가능한 모든 클라이언트가 Bean에 접근 가능하도록 한다.

• <use-access-control>

- Boolean 설정은 Bean 메소드들을 수행할 때 Jakarta EE의 principal에 따라 메소드의 수행이 Java SE Security Manager의 감시를 받도록 할 것인지를 결정한다. 자세한 내용은 "JEUS Security 안내서"와 JACC specification을 참고한다. (기본값: false)

다음은 위의 설정들이 Stateful Session Bean class와 DD에 적용된 예로 대응되는 Annotation과 DD의 element는 굵게 표시했다.

Stateful Session Bean class와 DD : <CounterEJB.java>

```
package ejb.basic.statefulSession;

@Stateful(name="counter", mappedName="counterEJB")
public class CounterEJB implements Counter, CounterLocal
{
```

```

private int count = 0;

public void increase()
{
    count++;
}
...
}

```

Stateful Session Bean class와 DD : <jeus-ejb-dd.xml>

```

<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>counter</ejb-name>
            <export-name>counterEJB</export-name>
            <local-export-name>counterEJB_Local</local-export-name>
            <export-port>7654</export-port>
            <export-iiop/>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>

```

4.2.2. Thread Ticket 설정

다음은 EJB Thread Ticket Pool(이하 TTP)의 개념을 나타낸 그림이다.

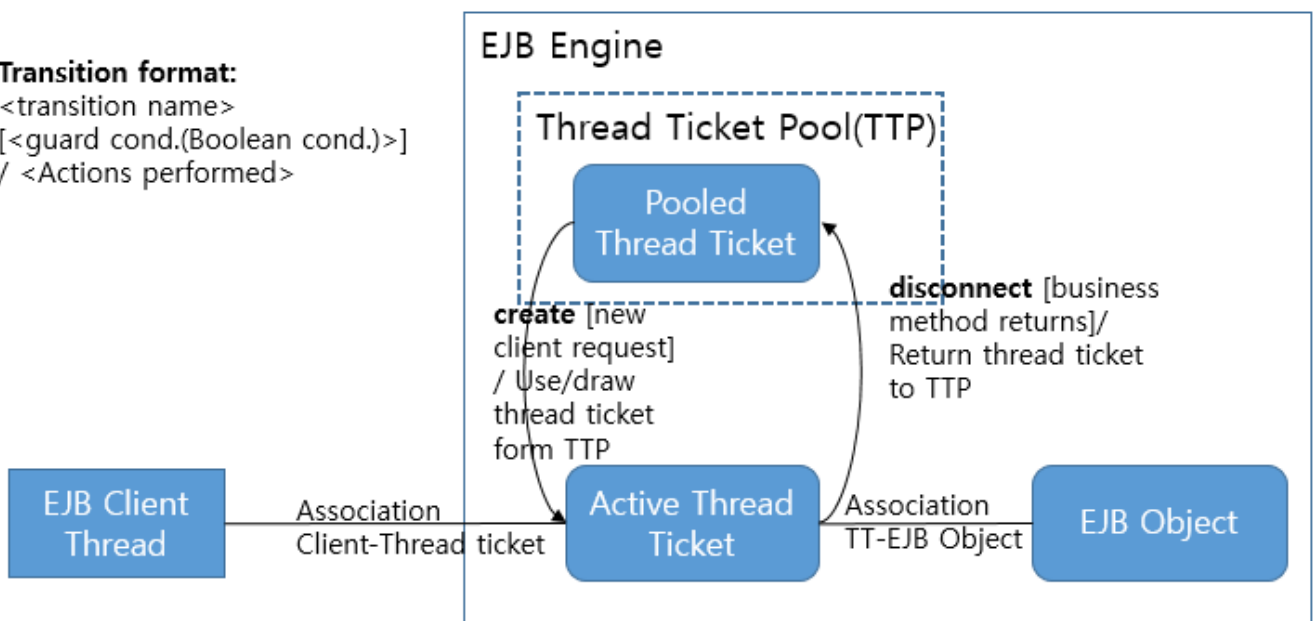
기본적인 동작은 클라이언트로부터 요청을 받으면 새로운 RMI 스레드를 클라이언트 요청에 부여하는 것이다. 이때 부여된 RMI 스레드의 수가 설정된 최댓값보다 많은 경우에는 해당 요청에 스레드를 부여하지 않고 대기한다.

Transition format:

```

<transition name>
[<guard cond.(Boolean cond.)>]
/ <Actions performed>

```



TTP 상태 전이도

위 그림에서 볼 수 있듯이 TTP는 Thread Ticket(이하 TT)을 가지고 있다. 한 개의 TT는 하나의 클라이언트 스레드가 EJB 엔진에 접근할 수 있도록 허가한다. 클라이언트의 요청이 도착하면, 하나의 TT가 TTP에서 얻어지고 클라이언트에게 부여된다. 이는 각 클라이언트 스레드는 실제 EJB Instance에게 요청을 전달할 한 개의 EJB Object에게 할당된다는 것을 의미한다. 그러므로 EJB Object는 EJB 엔진 내에 존재하는 하나의 Object라고 볼 수 있고 EJB 클라이언트와 실제 EJB Instance와의 연결을 관리한다.

TTP에 있는 TT의 숫자보다 많은 클라이언트의 요청이 오는 경우 새로운 요청은 TT가 Free되어 Pool로 반환될 때까지 기다려야 한다. 클라이언트의 요청이 TT를 기다린지 10분이 초과할 경우 시간 초과로 RemoteException을 던진다.

EJB 처리가 끝나거나(Stateless Session Bean) Connection 타임아웃이 발생하여 클라이언트가 연결을 잃을 경우에 TTP로 TT가 반환된다. 그리고 TTP는 원격 호출이 가능한 EJB 컴포넌트마다 하나씩 가지고 있다. 이 외에도 Connection Pool과 Bean Pool이 있다. MDB에서는 TTP가 아닌 Bean Pool을 설정한다.



Connection Pool과 Bean Pool에 대한 자세한 내용은 [Session Bean](#)과 [Entity Bean](#)을 참고한다. MDB에서 Bean Pool 사용에 대한 자세한 내용은 [Message Driven Bean\(MDB\)](#)을 참고한다.

EJB의 TT은 Bean의 최상위 XML 태그의 **<thread-max>**에서 설정한다. 이 값은 각 EJB를 수행하는 스레드의 최대 허용 개수, 즉 최대 TT의 수를 의미한다. 기본은 100이다. 대기하고 있는 클라이언트의 요청 수가 이 값보다 클 경우에는 새로운 요청들은 먼저 진행 중인 작업이 끝나고, Pool로 TT가 반환될 때까지 대기한다. 대기한지 10분이 지날 경우 시간 초과로 RemoteException을 던진다.

다음은 BMP Bean에 위의 설정을 적용한 예이다.

BMP Bean 설정 : <jeus-ejb-dd.xml>

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

4.2.3. External Reference 설정과 매핑

Bean이 사용하는 External Reference는 Annotation을 통해 설정하거나 또는 ejb-jar.xml에서 사용한 reference 이름과 매핑된 jeus-ejb-dd.xml의 JNDI export name을 사용하여 설정할 수 있다. 선언된 모든 External Reference들은 JNDI export name과 매핑되어야 한다. Annotation이나 ejb-jar.xml에 선언된 External Reference는 각 Reference에 해당하는 jeus-ejb-dd.xml의 태그에 실제 시스템 JNDI export name으로 매핑되어야 한다.

다음은 ejb-jar.xml과 JEUS EJB DD 파일의 참조 태그에 대한 설명이다.

ejb-jar.xml 태그	jeus-ejb-dd.xml 태그	설명
<env-entry>	<env>	EJB 환경설정을 정의하거나 무시한다.
<ejb-ref>	<ejb-ref>	EJB reference를 실제 export name으로 binding한다.
<resource-ref>	<res-ref>	External resource manage reference를 실제 export name에 binding한다.
<resource-env-ref>	<res-env-ref>	관리되고 있는 object reference를 실제 export name에 binding한다.
<message-destination-ref>	<message-destination-ref>	Message destination의 reference를 실제 export name에 binding한다.
<service-ref>	<service-ref>	Webservice endpoint의 reference를 실제 export name에 binding한다.

모든 <env> 태그는 다음의 하위 태그를 설정해야 한다.

태그	설명
<name>	env의 이름으로 ejb-jar.xml에 정의된 <env-entry-name> 태그의 값과 같은 값을 사용한다.
<type>	값의 자료형에 해당하는 완전한 Java 클래스 이름으로 기본 자료형은 wrapper 클래스를 사용한다.
<value>	env의 실제 값을 선언한다.

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

태그	설명
<ref-name>	EJB 소스 코드와 ejb-jar.xml에서 정의된 참조 이름으로 ejb-jar.xml에서 <ejb-ref>와 동격의 태그는 <ejb-ref>이다. <resource-ref>와 동격은 <res-ref>이고, <resource-env-ref>와 동격은 <res-env-ref>이다.
<export-name>	JNDI에 export될 때 사용되는 이름이다.



<ref-name>과 <export-name> 태그는 반드시 ejb-jar.xml에 설정된 이름과 같은 값이어야 한다.

다음은 External Reference를 JNDI로 매핑하는 방법으로, Annotation과 XML의 예이다.

External Reference를 JNDI로 매핑 : <CounterEJB.java>

```
@Stateful

@EJB(name="AccountEJB", beanInterface=ejb.Account.class, mappedName="ACCEJB")
@Resource(name="jdbc/DBDS", type=javax.sql.DataSource.class, mappedName="ACCOUNTDB")
public class CounterEJB implements Counter{
    @Resource(name="minAmount")
}
```

```

private int minAccount = 100;

. . .
}

```

External Reference를 JNDI로 매핑 : <ejb-jar.xml>

```

<ejb-jar.xml>
. . .
<enterprise-beans>
  <session>
    . . .
    <env-entry>
      <env-entry-name>minAmount</env-entry-name>
      <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
    <ejb-ref>
      <ejb-ref-name>AccountEJB</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <remote>ejb.Account</remote>
    </ejb-ref>
    <resource-ref>
      <res-ref-name>jdbc/DBDS</res-ref-name>
      <res-ref-type>javax.sql.DataSource</res-ref-type>
    </resource-ref>
    . . .
  </session>
</enterprise-beans>
. . .
</ejb-jar.xml>

```

External Reference를 JNDI로 매핑 : <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>AccountEJB</ref-name>
        <export-name>ACCEJB</export-name>
      </jndi-info>
    </ejb-ref>
    <res-ref>
      <jndi-info>
        <ref-name>jdbc/DBDS</ref-name>
        <export-name>ACCOUNTDB</export-name>
      </jndi-info>
    </res-ref>
    . . .
  </jeus-bean>

```



```

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

```

4.2.4. HTTP Invoke 환경설정

EJB 모듈 중 특정 EJB에 HTTP invocation을 설정하고 사용하기 위해서는 jeus-ejb-dd.xml의 <jeus-bean> 하위에 <invoke-http>를 설정해야 한다. EJB 엔진에서 HTTP invocation이 필요하면 domain.xml의 ejb-engine에 <invoke-http>를 추가한다([EJB 엔진](#)에서 언급). 이 설정은 모든 모듈에 적용되는데, 각 모듈의 DD 파일에 <invoke-http>가 있다면 파일의 설정을 사용한다.

다음은 jeus-ejb-dd.xml의 HTTP invoke 환경설정 예제이다.

HTTP Invoke 환경설정 : <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>

```

<invoke-http> 설정에는 2개의 하위 태그가 존재한다.

태그	설명
<url>	<p>반드시 HTTP-RMI Stub으로부터 호출되는 RMI 핸들러 서블릿(jeus.rmi.http.ServletHandler)의 URI를 입력한다. 이 URI에서는 프로토콜, IP, 포트를 제외한 서블릿 요청 경로만을 입력한다.</p> <p>프로토콜은 "HTTP"로 IP는 RMI 런타임과 동일한 주소로 간주된다. 즉, HTTP-RMI 요청을 받은 웹 서버와 웹 엔진이 RMI 런타임과 같은 머신에 있어야 된다. 그러면 RMI 런타임의 주소는 RMI Stub에게 알려지게 된다. 웹 서버의 포트는 반드시 다음 <http-port>에 설정해야 한다.</p>
<http-port>	HTTP-RMI 요청을 받을 포트 번호를 설정한다. 웹 서버 및 웹 엔진에는 반드시 RMI 핸들러 서블릿이 deploy되어 실행되고 있어야 한다.



HTTP invoke가 정상적으로 작동하기 위해선 RMI 핸들러 서블릿이 deploy되어 있어야 한다. RMI 핸들러 서블릿을 구현한 클래스는 jeus.rmi.http.ServletHandler이다. jeus.rmi.http.ServletHandler는 context 내에 <invoke-http>에 설정된 URL과 동일하게 deploy되어야 한다. deploy에 대한 자세한 내용은 "JEUS Web Engine 안내서"를 참고한다.

4.2.5. EJB 보안 설정

본 절에서는 JEUS에서 EJB를 위한 보안 설정 방법에 대해 설명한다. 여기에서 사용된 보안은 인증과 권한 부여를 의미한다.

JEUS EJB의 보안 설정을 위해 다음의 항목을 설정한다.

- EJB 모듈의 역할 할당(Role Assignment)
- EJB Run-as Identify 설정



EJB 설정에서 사용하는 실제 JEUS 사용자 목록은 accounts.xml에 정의되어 있다. 이에 대한 설정은 "JEUS Security 안내서"를 참고한다.

4.2.5.1. 보안 설정

다음은 상황별 보안 설정하는 방법에 대한 설명이다.

역할 할당(Role Assignment) 설정

EJB 모듈의 역할 할당 설정(Role Assignment)은 표준 ejb-jar.xml 파일에 개발자가 정의한 역할과 실제 JEUS 보안 도메인의 principal와 group name을 매핑하는 것이다. 이는 jeus-ejb-dd.xml 파일의 <module-info>의 <role-permission> 태그에서 설정한다.

다음은 역할 할당을 설정한 XML의 예이다.

역할 할당(Role Assignment) 설정 : <jeus-ejb-dd.xml>

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

다음은 설정 하위 필수 태그에 대한 설명이다.

태그	설명
<role>	EJB에 설정된(Annotation이나 ejb-jar.xml을 통해) 논리적인 role name을 지정한다.
<principal>	accounts.xml에서 정의된 실제 user name을 지정한다. role name-user name 쌍이 개발자가 정의한 role과 JEUS principal 간의 매핑을 정의한다. 하나의 역할 이름에 여러 개의 사용자 이름을 설정할 수 있다.

Run-as Identify 설정

Run-as Identify 설정은 ejb-jar.xml의 <run-as>에서 개발자가 정의한 역할 이름과 JEUS 보안 도메인 설정에서 지정한 실제 principal의 매핑을 제공한다(예: accounts.xml 파일에서).

이 principal은 일반적으로 간단한 사용자 이름을 사용한다. 이는 jeus-ejb-dd.xml에서 Bean 설정 수준과 같은 **<run-as-identity>** 태그에서 설정한다.

다음은 Run-as Identify를 설정한 XML 예이다.

Run-as Identify 설정 : <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>
```

태그	설명
<principal-name>	ejb-jar.xml 파일의 <run-as> 태그에서 주어진 role에 매핑할 user name으로 accounts.xml에 설정되어 있어야 한다.

4.2.5.2. 보안 설정 예제

다음은 role-permission의 role과 run-as의 role이 실제 JEUS 사용자에서 매핑되는 예이다.

Java 클래스와 XML 내용의 일부분들은 3개의 설정 파일들(ejb-jar.xml, jeus-ejb-dd.xml, accounts.xml)에서 가져온 것이다. 이 파일들에서 서로 동일해야 하는 부분들은 굵은 글자로 강조하였다.

보안 설정 : <CustomerBean.java>

```
@Stateless(name="CustomerEJB")
@RolesAllowed("CUSTOMER")
```

```
public class CustomerBean implements Customer {
    ...
}
```

보안 설정 : <EmployeeServiceBean.java>

```
@Stateful(name="EmployeeService")
@RunAs("ADMIN")
public class EmployeeServiceBean implements EmployeeService{
    ...
}
```

보안 설정 : <ejb-jar.xml>

```
<?xml version="1.0"?>
<ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/ ejb-jar_4_0.xsd">
    <enterprise-beans>
        <session>
            <ejb-name>CustomerEJB</ejb-name>
            . . .
        <security-role-ref>
            <role-name>CUSTOMER</role-name>
        </security-role-ref>
        <security-identity>
            <use-caller-identity />
        </security-identity>
    </session>
    <session>
        <ejb-name>EmployeeService</ejb-name>
        . . .
        <security-identity>
    </security-identity>
    <run-as>
        <role-name>ADMIN</role-name>
    </run-as>
    </security-identity>
    </session>
    . . .
    </enterprise-beans>
    <assembly-descriptor>
    <security-role>
        <role-name>CUSTOMER</role-name>
    </security-role>
    <method-permission>
        <role-name>CUSTOMER</role-name>
        <method>
            <ejb-name>CustomerEJB</ejb-name>
            <method-name>*</method-name>
            <method-params />
        </method>
    </method-permission>
    . . .
    </assembly-descriptor>
    . . .
</ejb-jar>
```

```
</ejb-jar>
```

위의 예에서는 2개의 Session Bean을 선언하였다.

- 첫 번째 Session Bean("CustomerEJB")은 "CUSTOMER" 역할에 연결되는 보안 역할을 가지고 있다.
- 두 번째 Session Bean("EmployeeService")은 "ADMIN"이라는 <run-as> 역할을 선언하고 있다.
"CUSTOMER"와 "ADMIN" 역할은 EJB deploy할 때 실제 시스템 principal들과 매핑되어야 한다.

다음의 XML DD 파일은 "CUSTOMER"와 "ADMIN" 역할이 실제 시스템 principal 이름들(각각 "customer"와 "peter")에 어떻게 매핑되는지 보여주는 예이다.

보안 설정 : <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>CUSTOMER</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>
```

다음은 이 사용자들이 accounts.xml에 어떻게 설정되는지를 확인할 수 있는 예이다.

보안 설정 : <accounts.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>customer</ name>
      <password>{SHA}bLSUPYpdjb8QDcq+ozfbIEZx5oY=</name>
      <group>test</group>
    </user>
    <user>
      <name>peter</ name>
      <password>{SHA}McbQlyhI3yi0G1HGTg8DQVWkyhg=</name>
      <group>test</group>
    </user>
  </users>
</accounts>
```

위의 예에서 "customer"와 "peter"라는 사용자들의 정의를 확인할 수 있다. 이 사용자들은 jeus-ejb-dd.xml 파일을 통하여 ejb-jar.xml 파일의 "CUSTOMER"와 "ADMIN" 역할과 연관되어 있다.

4.3. EJB 모니터링

JEUS에서는 EJB 모듈에 포함된 개별 EJB를 제어하는 방법은 제공하지 않는다. 그러나 콘솔 툴을 사용한 Bean별 모니터링은 가능하다.

콘솔 툴에서는 **application-info** 명령어를 사용하여 Bean Name, Export Name 및 상태 등 개별 EJB의 정보를 조회할 수 있다. 다음의 예제에서는 countermod라는 EJB가 deploy되어 있다고 가정한다. application-info 명령어에 대한 자세한 내용은 JEUS Reference 안내서의 "application-info"를 참고한다.

다음과 같이 명령어를 실행하면 EJB 모듈 내의 각 Bean Name과 Export Name을 확인할 수 있다.

```
[MASTER]domain1.adminServer>application-info -server server1 -id countermod -detail
General information about the EJB module [countermod].
```

```
=====
+-----+-----+
| Module Name | Unique Module Name |
+-----+-----+
| countermod | countermod         |
+-----+-----+
=====
```

Beans

```
=====
+-----+-----+-----+-----+
| Bean Name | Type           | Local Export Name | Remote Export Name |
+-----+-----+-----+-----+
| Count     | StatelessSessionBean |                   | Count               |
+-----+-----+-----+-----+
=====
```

다음과 같이 명령어를 실행하면 지정한 EJB 모듈에 등록되어 있는 개별 Bean의 상태를 확인할 수 있다.

```
[MASTER]domain1.adminServer>application-info -server server1 -id countermod -bean Count
```

Module name : countermod

Bean name : Count

```
=====
+-----+-----+-----+-----+-----+
| Name      | (Count) | WaterMark(High:Low:Cur) | Bound(Upper:Lower) | Time(Max:Min:Total) |
+-----+-----+-----+-----+-----+
| create    | times(0) |                          |                      |                      |
+-----+-----+-----+-----+-----+
| comitted  | transactio|                          |                      |                      |
|           | n(0)      |                          |                      |                      |
+-----+-----+-----+-----+-----+
| total-remote-t| thread(100:100:100)|                          |                      |
| hread       |           |                          |                      |                      |
+-----+-----+-----+-----+-----+
| timed-rb   | transactio|                          |                      |                      |
+-----+-----+-----+-----+-----+
```

	n(0)			
remove	times(0)			
active-bean		bean(0:0:0)		
request	request(0)			
total-bean		bean(0:0:0)		
rolledback	transactio			
	n(0)			
active-thread		thread(0:0:0)		
MethodReadyCou		bean(0:0:0)		
nt				

4.4. EJB 튜닝

모든 EJB 종류에 공통으로 EJB 운영 성능을 향상(또는 시스템 리소스의 낭비를 방지하기 위해)하기 위해 적용할 수 있는 몇 가지 설정은 다음과 같다.

- TTP의 최대값을 적절히 수정한다.
- 여러 개의 EJB 엔진에 Bean들을 클러스터링해서 설정한다. [EJB 클러스터링](#)을 참고한다.
- 적절한 JDBC Connection Pool을 설정한다. 설정에 대한 자세한 내용은 "JEUS Server 안내서"를 참고한다.
- EJB 2.x인 경우 EJB 모듈 Deploy 시간을 절약하기 위해 deploy 명령어에 -fast 옵션과 appcompiler를 사용한다. 자세한 내용은 [EJB 모듈](#)을 참고한다.

위의 기본적인 최적화 옵션뿐만 아니라 Bean의 종류에 따라 수십 개의 튜닝 팁이 존재한다. 그러므로, 각 EJB 종류에 따른 튜닝과 최적화하는 방법은 해당하는 각 장을 참고한다.

4.4.1. Thread Ticket Pool 설정 튜닝

EJB가 최상의 성능을 내기 위해서는 다음의 원칙을 적용해서 TTP가 올바르게 설정되어야 한다.

- max pool 크기를 크게 설정할수록 더 많은 TT들이 생성될 수 있고, 따라서 더 많은 클라이언트 요청을 수행할 수 있다. 그러나 크게 설정한 max pool 크기는 많은 클라이언트들이 EJB 엔진에 접속한 경우에 많은 메모리 자원을 소모하게 된다.
- 특정한 시간대에 Bean의 서비스 사용을 요청하는 클라이언트가 급격히 증가할 경우에는 max pool 크기에서 min pool 크기를 뺀 값을 증가시켜 새로운 TT들이 "batch"로 생성될 수 있도록 설정한다.

TTP의 설정은 특정한 EJB에 대하여 접근 가능한 양을 조절할 수 있는 밸브의 역할을 한다.



max pool 크기는 TTP에서 가장 중요한 파라미터이다. 그러므로 성능 개선을 위해서는 이

값을 가장 먼저 조절하고 튜닝해야 한다. 이 값은 <thread-max>에 설정한다.

5. EJB 상호 운용성 및 RMI/IIOP

본 장에서는 EJB의 상호 운용성 및 RMI/IIOP에 대해서 설명한다.

5.1. 개요

JEUS는 다른 WAS나 이기종 시스템 간에 원격 EJB 호출을 지원하기 위해 RMI/IIOP 프로토콜을 지원한다.

RMI/IIOP는 OMG(<http://www.omg.org/>)에서 정한 CORBA(Common Object Request Broker Architecture)에 기반한 것이며 EJB 상호 운용성(Interoperability)을 위해서 EJB 스펙에서 정한 표준 방식이다. JEUS 간에 또는 다른 WAS로부터 원격 호출을 지원해야 한다면 RMI/IIOP 상호 운용 기능을 사용할 것을 고려할 수 있다. RMI/IIOP 원격 호출 방식은 다른 WAS나 이기종 시스템을 지원하는 방식 중의 하나이다. 다른 WAS로부터 JEUS의 EJB를 호출하고 트랜잭션을 연동하는(또는 반대로) 다른 방법도 존재하지만 본 장에서는 RMI/IIOP 방식에 대해서만 설명한다.

상호 운용은 JEUS의 EJB를 다른 시스템에서 호출하는 것과 다른 WAS의 EJB를 JEUS에서 호출하는 것 2가지로 나눌 수 있다. 즉, JEUS가 RMI/IIOP 서버, 클라이언트 또는 둘 다 될 수 있다. 모든 경우에 마찬가지로 이를 지원할 수 있는 ORB(Object Request Broker) 런타임이 필요하며 트랜잭션과 보안의 상호 운용 또한 지원해야 한다.

JEUS에는 CORBA Naming Service를 지원하기 위한 COS Naming Server 및 Stub 클래스 없이도 Dynamic Stub을 메모리에서 자동으로 생성해 주는 ORB 런타임을 내장하고 있으며, 트랜잭션의 상호 운용을 위해 **OTS(Object Transaction Service)** 스펙과 보안 상호 운용을 위해 **CSiv2(Common Secure Interoperability version 2)** 스펙을 구현하고 있다.

CORBA 및 RMI/IIOP에 대해 이해하려면 다음의 관련 링크를 참조한다.

- [CORBA 2.3.1 Specification](#)
- [Glossary of Java IDL Terms](#)
- [Java RMI over IIOP](#)
- [OTS 1.2.1 Specification](#)
- [CSiv2 Specification](#)

5.1.1. 트랜잭션 상호 운용(OTS)

트랜잭션의 상호 운용은 OTS(Object Transaction Service) 스펙을 기반으로 이루어진다. Object는 트랜잭션에 영향을 받는 CORBA Object를 의미한다.

OTS를 사용하게 될 경우 ORB에 트랜잭션을 처리할 수 있는 Interceptor(일종의 Listener)를 추가하게 된다. 특별히 추가된 Interceptor가 IIOP 프로토콜 헤더의 트랜잭션 관련 부분을 처리한다. 이를 통해서 클라이언트 또는 서버 역할을 하는 JEUS는 다른 원격 시스템(다른 WAS 또는 JEUS) 간의 트랜잭션 전파(Transaction propagation)를 지원한다.



현재 지원하는 OTS의 스펙 버전은 1.2이다.

5.1.2. 보안 상호 운용(CSIv2)

CSIv2(Common Secure Interoperability version 2)는 상호 운용에 있어 보안을 보장해주기 위한 스펙이다. CSI를 사용하면 앞서의 OTS와 마찬가지로 보안처리가 가능한 Interceptor가 ORB에 추가되며, Interceptor가 보안에 관한 헤더 및 그 밖의 처리를 수행한다.

보안 상호 운용은 EJB Reference의 IOR(Interoperable Object Reference)에 CSI를 위한 보안 정보가 첨부됨으로써 시작된다. Naming Service에 IOR이 등록되면 이를 사용하려는 클라이언트의 ORB와 JEUS MS가 IIOP 프로토콜의 보안 레벨에서 협상(negotiation)을 처리한다. 또한 JEUS MS는 다른 EJB Bean을 호출할 경우 CSI 서비스가 필요하다면 클라이언트로 동작한다.



보안 상호 운용에 대한 자세한 내용은 CORBA의 CSIv2 스펙을 참고한다.

5.2. 상호 운용 설정

기본적으로 상호 운용은 설정되지 않았기 때문에 이를 사용하려면 반드시 관련된 서버 설정을 해야 한다.

JEUS의 EJB를 RMI/IIOP로 노출시키기 위해서는 MS의 Enable Interop 설정을 활성화해야 하고, deploy되는 EJB에 IIOP export 관련 설정이 있어야 한다. 반대로 JEUS를 RMI/IIOP 클라이언트로만 사용하는 경우에는 해당 MS의 Enable Interop 설정만 활성화하면 된다. 상호 운용과 관련된 사항은 domain.xml을 사용하여 설정할 수 있다.

5.2.1. COS Naming Service 설정

MS에서 제공하는 기본적인 JNDI Naming Service 외에 CORBA 객체를 위한 COS Naming Service를 추가할 수 있다. COS Naming Service에는 RMI/IIOP로 노출되는 EJB의 Reference(Stub)가 저장되고 이를 EJB 클라이언트에서 찾아서(lookup) 사용하게 된다. COS Naming Service는 필요한 시점에 자동으로 시작되며, MS JVM에 구동되어 '**BASEPORT + 4(예: 9740)**'을 서비스 포트로 사용한다.

5.2.2. 상호 운용성 활성화 설정

RMI/IIOP 클라이언트와 서버의 상호 운용 기능을 사용하려면 상호 운용성 속성을 활성화해야 한다. 이 속성이 활성화되어야 ORB가 초기화된다.

상호 운용성 속성을 활성화하려면 domain.xml에서 <enable-interop/>를 설정한다. 자세한 내용은 JEUS XML Reference 안내서의 "domain.xml 도메인 설정"을 참고한다.

CSIv2 기능을 사용하려면 좀 더 자세한 설정이 필요한데 이는 [CSIv2 보안 상호 운용 설정](#)을 참고한다.

5.2.3. CSIv2 보안 상호 운용 설정

CSIv2 기능을 수행하기 위해서 2가지 추가적인 보안 환경 파일과 JEUS Security Manager의 정보를 이용한다. 설정할 추가적인 보안 환경 파일은 **Keystore**과 **Truststore** 파일이다. 이 파일들의 경로와 필요한 정보는 시스템

프로퍼티를 사용해서 설정할 수 있다.

보안 환경 파일	설명
Keystore 파일	X.509를 위한 Key를 저장하고 Oracle 사에 의해서 공급된 X.509 keystore를 구현한 파일이다. Secure Socket Layer(SSL)가 호출됐을 때 이 파일이 클라이언트에게 보내진다.
Truststore 파일	X.509 클라이언트 측 증명 설정 파일이며, 형식은 Keystore와 같다.

CSIV2 관련 사항은 domain.xml에서 <interop-ssl-config>태그에서 설정한다. 자세한 내용은 JEUS XML Reference 안내서의 "domain.xml 도메인 설정"을 참고한다.

CSIV2 설정 예시 : <domain.xml>

```
<enable-interop>
  <interop-ssl-config>
    <keystore-path>/jeus/domains/domain1/ssl</keystore-path>
    <keystore-alias>changeit</keystore-alias>
    <keystore-password>changeit</keystore-password>
    <keystore-keypassword>{DES}FQrLbQ/D801l</keystore-keypassword>
    <truststore-path>/jeus/domains/domain1/ssl</truststore-path>
    <truststore-password>changeit</truststore-password>
  </interop-ssl-config>
</enable-interop>
```

CSIV2 관련 설정은 실행 스크립트의 시스템 프로퍼티를 통해서도 가능하다. 그러나 시스템 프로퍼티보다 **domain.xml 설정이 우선된다.**

다음은 실행 스크립트 옵션 "-D"를 이용해서 설정되는 값에 대한 설명이다.

파라미터	설명
jeus.ssl.keystore	Keystore 파일까지 절대 경로이다. (기본값: DOMAIN_HOME/config/keystore)
jeus.ssl.keypass	Keystore 파일에 주어진 패스워드값이다. (기본값: jeuskeypass)
jeus.ssl.truststore	Truststore 파일까지 절대 경로이다. (기본값: DOMAIN_HOME/config/truststore)
jeus.ssl.trustpass	Truststore 파일에 주어진 패스워드값이다. (기본값: jeustrustpass)

신뢰하는 노드 사이에서는 호출자의 authentication과 authorization이 불필요할 때도 있다.

jeus.ejb.csi.trusthosts 시스템 프로퍼티에 IP 주소를 설정하면 신뢰하는 노드의 불필요한 보안 점검을 피할 수 있다. JEUS Security Manager는 불특정 접근자에게 anonymous principal을 나타내는 'guest'를 사용한다.

5.2.4. EJB RMI/IIOP 설정

EJB를 RMI/IIOP로 노출시키려면 jeus-ejb-dd.xml에 EJB별로 <export-iiop>를 설정해야 한다. <export-iiop>가 설정되면 EJB Home Reference가 COS Naming Service에 주어진 <export-name>으로 등록된다. 이렇게 등록되어야만 외부에서 COS Naming Service를 통해 EJB Reference를 가져와서 사용할 수 있다.

다음은 EJB RMI/IIOP을 설정한 jeus-ejb-dd.xml의 예이다.

EJB RMI/IIOP 설정 : <jeus-ejb-dd.xml>

```
...
<jeus-bean>
  <ejb-name>CalcEJB</ejb-bean>
  <export-name>ejb/Calc</export-name>
  <export-iiop>
    <only-iiop>true</only-iiop>
  </export-iiop>
</jeus-bean>
...
```

태그	설명
<only-iiop>	EJB를 RMI와 RMI/IIOP로 각각 노출할지 아니면 RMI/IIOP만 노출할지를 설정한다. 각각 노출한다면 COS Naming Server에는 RMI/IIOP Stub이 등록되고, JEUS Naming Server에는 일반 RMI Stub이 등록된다.



현재는 EJB Home과 EJB Object Reference만 RMI/IIOP로 노출할 수 있다. EJB 3.0 비즈니스 뷰는 RMI/IIOP로 노출할 수 없다.

5.3. RMI/IIOP 클라이언트

RMI/IIOP 클라이언트는 다른 JEUS나 다른 벤더의 WAS 또는 Standalone 클라이언트가 될 수 있다. 본 절에서는 각각 어떤 방식으로 EJB를 사용할 수 있는지 설명한다.

5.3.1. JEUS Managed Server

JEUS MS에서 운용되는 애플리케이션에서 JEUS나 다른 벤더의 WAS 위에 있는 RMI/IIOP EJB를 호출하기 위한 방법은 다음과 같다.

먼저, COS Naming Server에서 EJB Home Reference(Stub)를 찾아와야 한다. 직접 찾아오는 경우에는 JNDI API의 corbaname URL을 사용해서 얻어올 수 있다.

corbaname lookup 사용

```
InitialContext ctx = new InitialContext();
Object obj = ctx.lookup("corbaname:iiop:1.2@192.168.11.22:9740#ejb/Calc");

CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
Calc calcRef = home.create();
```

URL은 "corbaname:iiop:1.2@<host>:<port>#<name>" 형식으로 <host>와 <port>는 COS Naming Server의 주소이다. 또는 직접 COS Naming Server를 PROVIDER URL로 지정하여 InitialContext를 얻어서 사용할 수도 있다.

다음의 예와 같이 PROVIDER URL을 지정하는 방식은 여러 가지가 있기 때문에 이 중에서 한 가지 방식으로

지정한다.

PROVIDER URL 사용

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "iiop://192.168.11.22:9740");
// env.put(Context.PROVIDER_URL, "iiopname://192.168.11.22:9740");
// env.put(Context.PROVIDER_URL, "corbaname:iiop:1.2@192.168.11.22:9740");
// env.put(Context.PROVIDER_URL, "corbaloc:iiop:1.2@192.168.11.22:9740");

InitialContext ctx = new InitialContext(env);
Object obj = ctx.lookup("ejb/Calc");

CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
Calc calcRef = home.create();
```

Dependency Injection을 사용하거나 "java:comp/env/" 형식의 논리적인 JNDI 이름을 사용하는 클라이언트의 경우에는 jeus-web-dd.xml과 같은 애플리케이션 JEUS DD 파일에서 <export-name>을 위의 corbaname URL로 매핑한다.

Servlet EJB Injection

```
@EJB(name="ejb/Calc")
private CalcHome calcHome;
```

RMI/IIOP EJB 매핑 : <jeus-web-dd.xml>

```
<ejb-ref>
  <jndi-info>
    <ref-name>ejb/Calc</ref-name>
    <export-name>corbaname:iiop:1.2@192.168.11.22:9740#ejb/Calc</export-name>
  </jndi-info>
</ejb-ref>
```

5.3.2. 다른 벤더 WAS

다른 벤더의 WAS에서 운용되는 애플리케이션에서 JEUS의 RMI/IIOP EJB를 호출하기 위한 방법은 [JEUS Managed Server](#)의 방식과 거의 동일하다. 이 경우 JEUS처럼 별도의 ORB 설정이 필요한지 확인하고, 필요하다면 설정을 해야 한다. 자세한 내용은 해당 제품의 관련 문서를 참고한다.

5.3.3. Standalone 클라이언트

Standalone 클라이언트는 별도의 ORB 런타임 및 관련된 설정이 되어 있지 않기 때문에, 먼저 기본적으로 필요한 jclient.jar 클라이언트 라이브러리 외에도 JEUS 시스템 라이브러리에 위치한 corba-omgapi.jar와 corba-orb.jar를 클래스 패스에 추가하고 jeus.client.interop=true 시스템 프로퍼티를 추가해야 한다.

EJB Home Reference(Stub)를 얻어오는 방식은 [JEUS Managed Server](#)와 비슷하다. 단지, JEUS에서 제공하는 JNDI 프로바이더를 추가적으로 지정한다.

```

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "jeus.jndi.JNSContextFactory");
env.put(Context.PROVIDER_URL, "iiop://192.168.11.22:9740");

InitialContext ctx = new InitialContext(env);
Object obj = ctx.lookup("ejb/Calc");

CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
Calc calcRef = home.create();

```

5.4. 알려진 문제점(Known Issues)

알려진 문제점들은 다음과 같다.

- 서버로 접속할 수 없을 때 재접속을 1분 동안 무한 시도하는 현상

JEUS에 내장된 ORB에서 기본적으로 서버로 연결할 수 없을 때 재접속을 60초 동안 재시도하는데, 이때 별도의 sleep time 없이 진행된다. 따라서 수많은 로그 메시지가 찍히거나 CPU가 과점되는 문제가 있다. 이를 방지하기 위해서는 다음과 같은 옵션을 클라이언트에서 시스템 프로퍼티로 설정해야 한다. (기본값: 60000, 단위: ms)

```
com.sun.corba.ee.transport.ORBCommunicationsRetryTimeout=1
```

- **PortableRemoteObject.narrow()** 호출 후에 **NullPointerException**이 발생하는 현상

클라이언트에서 다음과 같이 호출을 하는 경우 home 객체가 null이 리턴되면서 NullPointerException이 발생하는 경우가 있다.

NullPointerException이 발생하는 경우

```

CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
// null is returned
Calc calcRef = home.create(); // NullPointerException

```

이 문제는 CORBA Stub 클래스를 찾지 못한 경우 발생한다. JEUS MS에서 **Enable Interop** 설정이 제대로 되어 있지 않은 경우이다. 해당 설정이 있으면 올바른 ORB가 초기화되어 Stub을 동적으로 메모리에 자동 생성한다. 외부 클라이언트의 경우에는 동적 Stub을 생성하는 기능이 없는 경우에 이런 에러가 발생할 수 있다.

Standalone 클라이언트의 경우에는 시스템 프로퍼티에 다음과 같이 설정되어 있는지를 확인한다.

```
jeus.client.interop=true
```

- WebLogic에서 JEUS에 **dot(.)**이 포함된 이름의 EJB를 찾지 못하는 현상

"com.acme.CalcHome"과 같이 dot(.)이 포함된 형식으로 <export-name>을 지정한 경우 WebLogic에서는 lookup이 실패한다. 이는 WebLogic에서 dot(.)을 slash(/)와 동일하게 취급하여 내부적으로

"com/acme/CalcHome"과 같이 요청을 보내기 때문이다. 이런 경우에는 dot(.)이 포함된 이름을 사용하지 않도록 한다.

6. EJB 클러스터링

본 장에서는 EJB 클러스터링 개념과 주요 기능 설정 방법에 대해 설명한다.

6.1. 개요

EJB의 Failover와 Load Balancing 기능을 사용하기 위해서 각 Bean들은 여러 EJB 엔진에 deploy되어 클러스터링을 형성해야 한다. 클러스터링은 컴포넌트 레벨(개별 Bean)에서 수행되고 Stateless/Stateful Session Bean과 Entity Bean에서 이용할 수 있다. Message Driven Bean(MDB)은 클러스터링 대상에 해당 되지 않는다.

JEUS EJB 클러스터링은 크게 다음과 같은 2가지 기능을 갖는다.

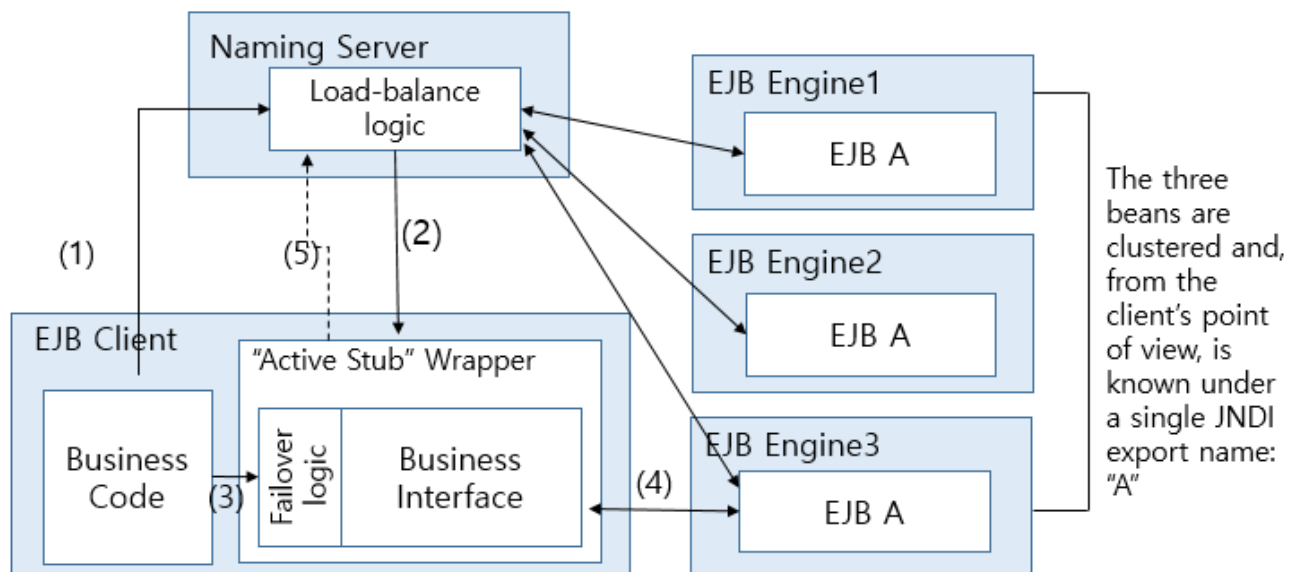
- Load Balancing

EJB에 대한 요청을 분산시켜 Bean의 전체적인 응답 속도를 향상시킨다.

- Failover

Bean의 메소드 실행 전과 실행 중에 하나의 EJB 인스턴스나 EJB 엔진이 멈추면 다른 EJB 엔진의 EJB 인스턴스에서 요청이 처리된다.

다음 그림은 EJB 클러스터링의 주요 기능인 Failover와 Load Balancing의 동작 방식이다.



(1) Client looks up EJB cluster named "A"

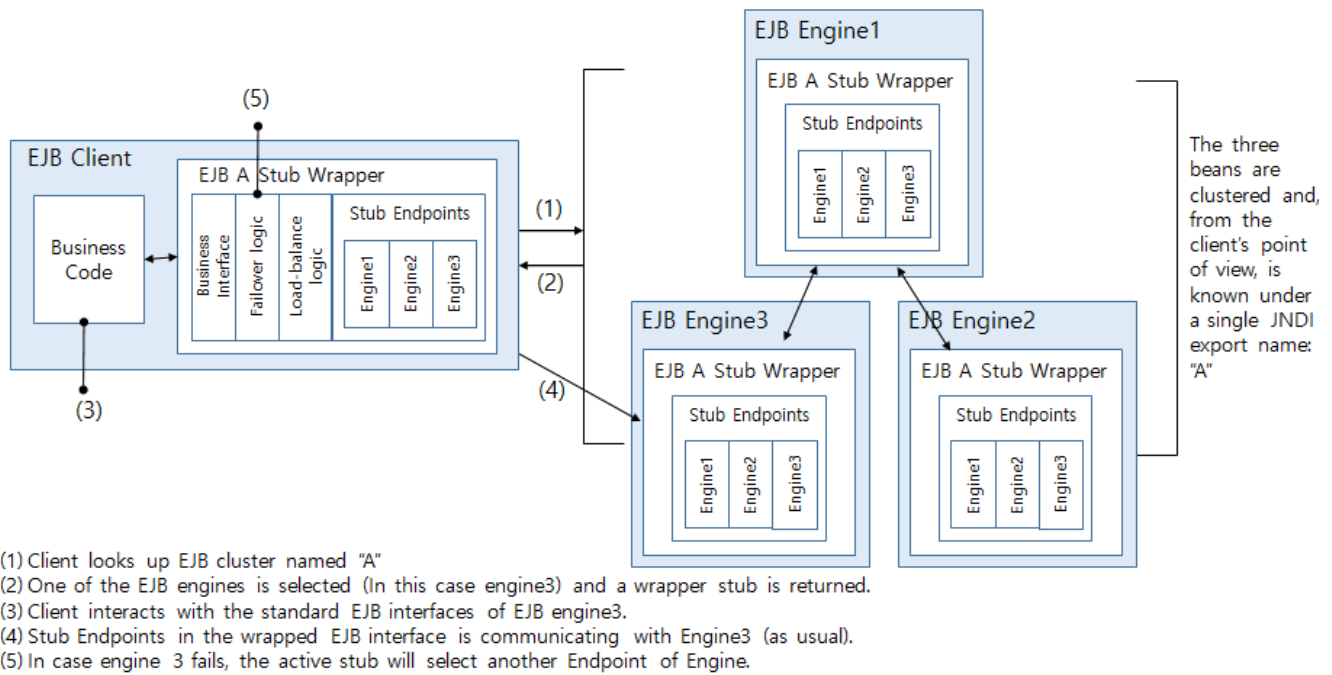
(2) One of the EJB engines is selected (In this case engine3) and a wrapper stub is returned.

(3) Client interacts with the standard EJB interfaces of EJB engine3.

(4) The Wrapped EJB interfaces is communicating with bean in engine 3(as usual).

(5) In case engine 3 fails. The active stub will attempt to look up the same bean in a different engine.

기본 클러스터링 아키텍처



EJB 3 stateless 클러스터링 아키텍처

클러스터링을 원하는 모듈을 deploy하면 Naming Server에 모두 같은 이름으로 바인드된다. 클라이언트는 그 하나의 이름으로 수행해도 Load Balancing과 Failover가 가능하다. 따라서 같은 모듈이더라도 Naming Server에 바인딩할 이름을 다르게 하여 deploy하면 해당 모듈은 클러스터링되지 않음에 주의한다.



Stateful Session Bean의 경우 Failover를 위해 JEUS 분산 세션 매니저(Session Manager)를 사용한다. 세션 매니저는 EJB 엔진당 하나만 존재하기 때문에 클러스터링할 Bean의 클러스터링 범위가 각 Bean별로 다르다면 안 된다. 예를 들어 Bean A는 EJB engine1과 EJB engine2로 묶고 Bean B는 EJB engine1과 engine3으로 묶었다면 세션 매니저는 Bean A와 Bean B가 EJB engine1, EJB engine2, EJB engine3에 클러스터링되어 있다고 잘못 확인하게 된다.

6.2. 주요 기능

다음은 EJB 클러스터링의 주요 기능에 대한 설명이다.

6.2.1. Load Balancing

클라이언트가 lookup이나 injection에 의해 Bean A를 요청하면 Naming Server는 3개의 EJB 엔진에 존재하는 3개의 Bean 중 하나를 선택하여 반환한다. 이는 3개의 Bean들은 동일하게 같은 메소드 호출 요청을 받게 되고 잠재적으로 한 개의 엔진이 모든 요청에 대해 서비스하는 것보다 무려 3배의 시스템 성능 향상을 기대할 수 있다는 것을 의미한다(Load Balancing의 경우에 발생하는 작은 자원소모를 계산하지 않을 때).

클라이언트는 Bean의 종류에 따라 아래와 같이 동작한다.

- Stateless
 - EJB 3.x

Method invoke 마다 Load Balancing을 수행한다(단, 첫 번째 호출은 lookup할 때에 선택된 EJB 엔진으로 가게 된다).

JEUS 7 이하에서는 lookup을 통해서만 Load Balancing을 하였으므로 한번 lookup해 온 EJB 엔진으로만 호출할 수 밖에 없었다. 하지만 JEUS 8부터는 lookup해 온 Bean 안에 클러스터링에 속한 다른 EJB 엔진과 통신할 수 있는 End Point들이 존재한다 ([EJB 3 stateless 클러스터링 아키텍처](#) 참조). 이 End Point들을 통해서 method invoke level의 Load Balancing이 가능하게 된다.

아래는 설정 가능한 Load Balancing 정책이다.

구분	설명
RoundRobin	클러스터에 속한 서버를 돌아가면서 한번씩 선택한다.
Random	랜덤하게 서버를 선택한다.
LocalLinkPreference	같은 서버를 우선적으로 선택한다.

Load Balancing 정책은 system property (jeus.ejb.cluster.selection-policy) 를 통해 설정할 수 있다. 자세한 사항은 JEUS Reference 안내서의 "EJB 시스템 프로퍼티"를 참고한다.

- EJB 2.x

jeus.jndi.clusterlink.selection-policy 프로퍼티를 이용 lookup할 때 EJB 엔진을 선택한다. 이때 선택된 EJB 엔진을 계속 호출한다.

- Stateful

jeus.jndi.clusterlink.selection-policy 프로퍼티를 이용 lookup할 때 EJB 엔진을 선택한다. 이때 선택된 EJB 엔진을 계속 호출한다.

아래의 표는 위의 EJB 버전 및 타입에 따른 Load Balancing 여부를 표로 정리한 것이다.

구분	new InitialContext()	lookup	method invoke
EJB 2.x, EJB 3.x - Stateful	O (Random)	O	X
EJB 3.x - Stateless	O (Random)	X	O

만약 성능보다 동작방식을 우선하여 'EJB 3.x - Stateless'를 EJB 2.x과 같은 방식의 Load Balancing으로 동작시키려면 [EJB 클러스터링 설정](#)을 참조하여 JEUS 7 이하 버전의 클러스터링 방식을 사용해야 한다.

6.2.2. Failover(EJB 복구)

Failover는 하나의 EJB 서비스에 장애가 발생해도 서비스를 정상적으로 제공하는 것을 의미한다. (예: OS 장애, 네트워크 중단 또는 EJB 엔진 장애)

JEUS 시스템이 처리할 수 있는 장애 복구에는 다음의 2가지가 있다.

- 클라이언트의 요청이 도착했을 때 접근 불가능한 Bean에게 요청을 보내지 않고, 처리되지 않은 요청을 다른 Bean에게 다시 보내는 방법

단순히 문제가 발생한 EJB 엔진을 제외하고 Load Balancing 알고리즘을 실행한다. 새로운 클라이언트의 요청을 처리하기 위해서는 사용 가능한 엔진의 Bean을 선택한다.

JEUS에서는 Failover의 Rerouting이 클라이언트 측의 EJB Stub에서 처리된다. 이 Stub을 Active Stub이라 부르고 또는 표준 EJB 인터페이스를 둘러싼 wrapper라고도 부른다. 이러한 wrapper를 사용할 때의 다른 점은 현재 연결되어 있는 Bean 또는 EJB 엔진이 문제가 있는지 판단할 수 있는 로직을 가지고 있는지 여부이다. 이러한 문제점이 발견되면, 기동 중인 Stub이 자동으로 JNDI 서버에 접속해서 작동하고 있는 EJB 엔진을 대신하여 새로운 Stub을 요청한다([기본 클러스터링 아키텍처](#) 또는 [EJB 3 stateless 클러스터링 아키텍처](#)의 5 번).

- **실행 중인 Bean이 어떤 이유로 런타임 오류를 가지고 있다면 진행 중인 요청을 다른 Bean에게 다시 보내는 방법**

이러한 상황에서의 복구 방법은 한계가 있다. Bean이 요청을 처리하고 있을 때 문제가 발견되면 얼마큼 요청을 처리하고 있었는지도 모르고 문제가 발생했을 때 어떤 런타임 에러를 발생시켰는지도 모른다. 단순히 클러스터에 존재하는 다른 Bean의 같은 메소드를 호출하는 것은 또 다른 부작용을 조장할 수 있기 때문에 자칫하면 위험한 결과를 가져올 수도 있다.

이 문제를 명백히 하기 위해 DB 필드를 1씩 증가시키는 메소드를 가지는 Bean Instance A를 고려해보자. 1이 증가된 후에 바로 문제가 발생하는 경우 단순히 다른 Bean "B"에 있는 같은 Business 메소드를 다시 호출하면 1이 다시 한 번 증가한다. 결과적으로 DB에 일괄적이지 못하고 잘못된 값을 전송하게 된다. 이런 경우 Idempotent 메소드를 이용하면 안전하게 복구될 수 있다. Idempotent 메소드를 통한 EJB 복구 방법에 대한 자세한 내용은 [Idempotent 메소드를 통한 EJB 복구](#)를 참고한다.

위의 두 시나리오의 차이는 오류 상황이 발견되는 시점이다. 즉, Remote Business 메소드를 호출하기 전인지 또는 Bean이 요청을 처리하고 있는 중인지의 예가 있을 수 있다.

6.2.3. Idempotent 메소드를 통한 EJB 복구

Idempotent 메소드는 부작용이 없는 getter 메소드이다. 이는 메소드의 수행 중에 어떠한 상태(예: instance 변수, DB 필드 등)도 변경되지 않는 것을 보장한다.

따라서 [Failover\(EJB 복구\)](#)에서의 두 번째 복구 방법이 지닌 한계는 Idempotent 메소드로 극복할 수 있다. 그러나 Idempotent 메소드가 아니라면 역시 대책이 없다. 런타임 에러가 발생한 메소드를 다시 실행시키는 것보다는 Exception을 던지는 것이 차라리 낫다. 그러므로 Idempotent 메소드를 많이 사용할수록 EJB Failover는 더 잘 작동된다. 메소드가 Idempotent 메소드인지 아닌지 판단하는 공식은 없다. 그러므로 Business 메소드의 상태를 정확히 식별하고 설정해야 한다.

6.2.4. Session Replication

Stateful Session Bean의 경우 세션의 백업을 위해서 JEUS 세션 매니저를 사용한다. 일반적으로 Business 메소드 호출 단위로 세션의 상태가 변화하기 때문에 JEUS에서는 메소드 호출이 발생하고 결과가 리턴되는 시점마다 JEUS 세션 매니저에 세션 백업을 요청한다. 이러한 백업 작업을 다른 용어로 **세션 복제(Session Replication)**라고 한다.

JEUS 세션 매니저는 세션을 동기적(Sync) 또는 비동기적(Async)으로 복제할 수 있다. JEUS에서는 이를 복제 모드(Replication Mode)라고 한다.

- 동기적(Sync) 복제 모드

세션 백업이 완료될 때까지 Bean이 기다려야 한다. 따라서 Failover를 해야 할 때 항상 최신의 세션이 복제되어 있다는 장점이 있다. 하지만 세션 매니저에서 네트워크 장애 등의 원인으로 타임아웃이 발생해서 진행이 안 될 경우 Bean도 그만큼 기다려야 하는 문제가 있다.

- 비동기적(Async) 복제 모드

퍼포먼스 문제는 없어지지만 세션 복제가 바로 발생하지 않고 나중에 이루어지기 때문에 그 사이에 Failover를 한다면 최신의 세션을 얻을 수 없게 된다.

2가지 방법은 서로 장단점이 있으므로 JEUS에서는 Bean과 각 Business 메소드 특성에 따라 사용자가 설정할 수 있다. 또한 세션 복제를 하지 않아도 되는 메소드가 있을 수 있으므로 이 역시 설정할 수 있다. 자세한 설정 방법은 [EJB 클러스터링 설정](#)을 참고한다.



JEUS에서는 클러스터링에 참여한 Stateful Session Bean이라면 기본적으로 동기적(Sync) 복제 모드로 세션 복제가 이루어진다. 사용자는 설정에 의해 이를 조정할 수 있다.

6.3. EJB 클러스터링 설정

EJB 클러스터링은 Bean 클래스에 Annotation으로 설정하거나 jeus-ejb-dd.xml에 설정할 수 있다. 설정할 사항은 클러스터링으로 구성될 Bean, 그 Bean의 Idempotent 메소드 그리고 Bean 또는 각 메소드의 세션 복제 모드이다.

본 절에서는 예를 통해 Annotation과 DD(xml) 파일에 클러스터링을 설정하는 방법에 대해 설명한다.

6.3.1. Annotation을 통한 클러스터링 설정

클러스터링에 참여하는 Bean 클래스 또는 메소드에 다음과 같은 Annotation을 이용하여 설정한다.

Annotation을 통한 클러스터링 설정 : <CounterEJB.java>

```
package ejb.basic.statelessSession;

import jakarta.ejb.Stateful;
import jeus.ejb.Clustered;
import jeus.ejb.Replication;
import jeus.ejb.ReplicationMode;

@Stateful(name="counter", mappedName="COUNTER")
@Clustered(useDlr = true)
@Replication(ReplicationMode.SYNC)
@CreateIdempotent
public class CounterEJB implements Counter, CounterLocal {
    private int count = 0;

    public int increaseAndGet() {
        return ++count;
    }

    @Replication(ReplicationMode.NONE)
```

```

public void doNothing(int a, String b) {
}

@Idempotent
public int getResult() {
    return count;
}

@Idempotent
@jeus.ejb.Replication(ReplicationMode.ASYNC)
public int getResultAnother() {
    return count;
}
...
}

```

다음은 각 클래스별 설정에 대한 설명이다.

클래스	설명
@jeus.ejb.Clustered	<p>Bean의 클러스터링을 전체적으로 활성화 또는 비활성화시킨다.</p> <p>useDir=true로 설정하면 JEUS 8에 새로 들어간 속도를 개선한 클러스터링을 사용하지 않게 된다. JEUS 8 클러스터링의 Failover 방식이 이전과는 다르게 작동하기 때문에 호환성을 위한 옵션이다. JEUS 8에서는 EJB 3.x stateless에만 적용된다. (기본값: false)</p>
@jeus.ejb.Idempotent	<ul style="list-style-type: none"> Bean 또는 Business 메소드가 Idempotent인지 선언한다. value를 false로 하면 Idempotent하지 않은 경우이다. 원격 Bean 객체를 생성하는 중 예외 상황이 발생할 경우 Stateless Session Bean에 대해서는 Idempotent로 항상 다시 시도하고 Stateful Session Bean의 경우에는 non-idempotent operation으로 애플리케이션에게 예외를 던진다.
@jeus.ejb.CreateIdempotent	<ul style="list-style-type: none"> Session Bean을 생성할 때 Idempotent하게 할지 선언한다. value를 false로 하면 Idempotent하지 않은 경우이다. 만약 이 Annotation을 기술하지 않는다면 Stateless Session Bean에 대해서는 항상 Idempotent로 간주한다.
@jeus.ejb.Replication	<ul style="list-style-type: none"> Bean별 또는 각 Business 메소드별로 세션 복제 모드를 설정할 수 있다. 여기에는 jeus.ejb.ReplicationMode라는 enum 타입의 클래스를 사용해야 한다. 이 클래스에 대한 것은 JEUS API 문서(javadoc)를 참고한다. 이 Annotation은 @Replication(ReplicationMode.SYNC)와 같은 식으로 사용이 가능하다. 사용자는 Bean별로 설정할 수 있으며 아무런 설정을 하지 않을 경우 디폴트는 ReplicationMode.SYNC이다. 메소드는 아무런 설정을 하지 않았을 경우 기본적으로 Bean에 설정된 값을 따른다. 단, @Idempotent인 경우는 기본적으로 ReplicationMode.NONE이다. 그리고 비즈니스 홈에 정의되는 create()는 Bean의 설정을 따른다.

6.3.2. xml을 통한 클러스터링 설정

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

xml을 통한 클러스터링 설정 : <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
  . . .
  <beanlist>
    . . .
    <jeus-bean>
      <ejb-name>counter</ejb-name>
      <export-name>COUNTER</export-name>
      . . .
      <clustering>
        <enable-clustering>true</enable-clustering>
        <use-dlr>true</use-dlr>
        <ejb-remote-idempotent-method>
          <method-name>getResult</method-name>
        </ejb-remote-idempotent-method>
        <ejb-remote-idempotent-method>
          <method-name>getResultAnother</method-name>
        </ejb-remote-idempotent-method>
        <create-idempotent>true</create-idempotent>
        <replication>
          <bean-mode>sync</bean-mode>
          <methods>
            <method>
              <method-name>doNothing</method-name>
              <method-params>
                <method-param>int</method-param>
                <method-param>java.lang.String</method-param>
              </method-params>
              <mode>none</mode>
            </method>
            <method>
              <method-name>getResultAnother</method-name>
              <method-params>
                <method-param>void</method-param>
              </method-params>
              <mode>async</mode>
            </method>
          </methods>
        </replication>
      </clustering>
      . . .
    </jeus-bean>
    . . .
  </beanlist>
  . . .
</jeus-ejb-dd>
```

다음은 **<clustering>**의 하위 설정 태그에 대한 설명이다.

태그	설명
<enable-clustering>	Bean의 클러스터링을 전체적으로 활성화 또는 비활성화시킨다.
<use-dlr>	EJB 클러스터링을 사용할 때 DynamicLinkRef를 사용할지 여부를 선택한다. <ul style="list-style-type: none"> ◦ true : JEUS 8에 새로 들어간 속도를 개선한 클러스터링을 사용하지 않게 된다. JEUS 8 클러스터링의 Failover 방식이 이전과는 다르게 작동하기 때문에, 호환성을 위한 옵션이다. JEUS 8 Fix#0에서는 EJB 3.x stateless에만 적용된다. ◦ false : JEUS 8에 새로 들어간 속도를 개선한 클러스터링을 사용한다. (기본값)
<ejb-remote-idempotent-method>	Bean 메소드 중에 Idempotent 메소드들을 선언한다(Annotation을 통한 클러스터링 설정 의 @jeus.ejb.Idempotent 설명 참조).
<ejb-remote-idempotent-exclude-method>	Bean 메소드 중에 Idempotent 메소드들로 선언한 것 중 제외하고 싶은 메소드를 선언한다. <ejb-remote-idempotent-method>보다 우선순위는 높고, 사용법은 동일하다.
<ejb-home-idempotent-method>	2.x 스타일의 홈 인터페이스에 정의된 메소드 중에 Idempotent 메소드들을 선언한다 (Annotation을 통한 클러스터링 설정 의 @jeus.ejb.CreateIdempotent 설명 참조). 사용법은 <ejb-remote-idempotent-method>와 동일하다.
<ejb-home-idempotent-exclude-method>	2.x 스타일의 홈 인터페이스에 정의된 메소드 중에 Idempotent 메소드들로 선언한 것 중 제외하고 싶은 메소드를 선언한다. <ejb-home-idempotent-method>보다 우선된다. 사용법은 <ejb-remote-idempotent-method>와 동일하다.
<create-idempotent>	Session Bean을 생성할 때 Idempotent하게 할지 선언한다.
<replication>	Bean 레벨의 세션 복제 모드 또는 메소드별 복제 모드를 설정한다. 자세한 내용은 Session Replication 과 xml을 통한 클러스터링 설정 : <jeus-ejb-dd.xml> 를 참고한다.

위에서 지정한 Bean 클러스터링이 작동하기 위해서는 다음의 내용을 주의해야 한다.

- 클러스터링에 참여하는 모든 Bean들은 <clustering> 하위의 모든 정보가 동일해야 한다.
- 클러스터링으로 구성하기 위해서는 원하는 Bean의 <export-name>을 모두 동일하게 설정한다.

6.4. EJB Failover 제한

매번 클라이언트에서 EJB Reference(Stub)를 lookup하지 않고, lookup한 EJB Reference를 Cache하여 계속 사용하는 경우(Injection을 사용하는 경우도 포함), 살아 있는 다른 EJB Endpoint가 있음에도 불구하고 Failover가 되지 않는 경우가 있을 수 있다. 이는 lookup하는 시점에 존재했던 EJB Endpoint 리스트에 해당하는 MS들이 모두 살아 있지 않고 그 이후에 구동된 새로운 MS만 존재하는 경우에 발생할 수 있다.

Failover를 해야하는 시점에서 내부적으로 lookup하여 새로운 EJB Reference를 클라이언트에게 전달한다. 그러나 이때 사용 중이던 EJB Reference가 lookup될 때 deploy되어 있던 MS들이 모두 다운되면 현재 사용 중이던 EJB Reference가 lookup될 당시 deploy되지는 않았지만 그 후에 deploy되어 Failover하는 시점에서 서비스 가능한 새로운 EJB Endpoint가 존재해도 Failover가 되지 않는다.

여기서 기존 MS들이 모두 다운되었다는 것은 비정상 종료되거나, 다운되거나, EJB Endpoint가 undeploy되어서 기존의 모든 EJB Endpoint의 서비스가 불가능한 것을 의미한다. 또한 새로운 EJB Endpoint가 deploy되는 경우는

뒤늦게(EJB Reference를 lookup하여 이미 사용 중일 때) 새로운 MS가 클러스터에 포함되거나, 뒤늦게 다운했던 MS를 재시작했거나 undeploy했던 EJB Endpoint를 뒤늦게 redeploy를 하는 경우가 포함된다.

2개의 MS로 Active/Backup 클러스터링을 구성하는 경우에 이런 현상이 발생할 수 있다. 또는 다음과 같은 시나리오에서 이런 경우가 발생할 수 있다.

예를 들어 A, B, C라는 MS가 있는데 클라이언트가 처음 lookup을 하여 계속 Cache를 하는 경우이다.

1. A, B, C에 모두 deploy되어 있고 처음 lookup한 결과, A의 EJB를 받는다.
2. A의 EJB를 사용하다 A가 비정상 종료되어 내부적으로 B의 EJB를 lookup하여 계속 서비스받는다.
3. B의 EJB가 undeploy되어서 C의 EJB를 lookup해서 사용한다.
4. 이때 다시 B의 EJB가 deploy되고, 곧이어 C가 비정상 종료되었다.

이 경우 C의 EJB Reference가 lookup될 때 B의 EJB Endpoint는 undeploy되어 있었고 C의 EJB Reference를 lookup한 다음에 deploy되었으므로 C가 다운되었을 때 B는 운용 중이었지만 Failover는 되지 않는다. 그러나 이때 B가 deploy나 undeploy된 것이 아니라 비정상 기동나 비정상 종료가 되었다면 정상적으로 Failover된다. 비정상 종료의 경우에는 Failover 시점에서 재기동 여부를 검사하기 때문에 가능하다.

이렇게 Failover가 제대로 수행되지 않는 상황에는 다시 lookup해서 새로운 EJB Reference를 가져온다. 그러면 새로운 Endpoint 목록을 가져오기 때문에 이런 문제를 피할 수 있다.

7. Session Bean

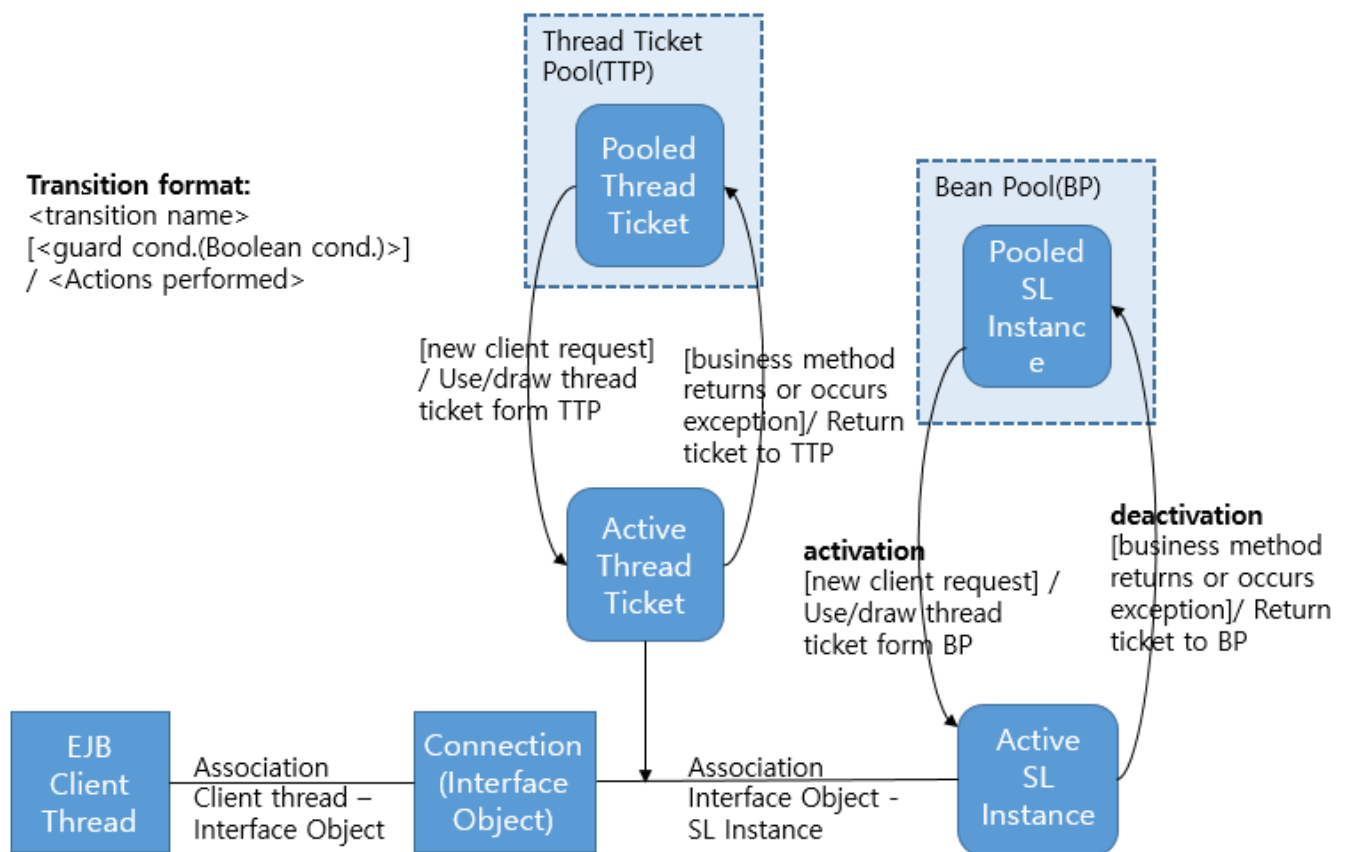
본 장에서는 Stateless Session Bean과 Stateful Session Bean에 대해서 자세히 설명한다. Stateless Session Bean은 사실상 EJB의 공통 특성에서 설명한 내용 외에는 특별한 것이 없다. 따라서 본 장에서는 주로 Stateful Session Bean들에 대해 설명한다.

7.1. Stateless Session Bean

본 절에서는 Session Bean의 TTP(Thread Ticket Pool)에 대해 설명한다.

7.1.1. Thread Ticket Pool(TTP)과 Object Management

Stateless Session Bean은 클라이언트의 요청에 관련된 상태 정보를 가지고 있지 않기 때문에 모든 클라이언트가 connection을 공유해서 Connection Pool은 설정할 필요가 없다. 그러나 상태 정보를 가지고 있지 않은 이유로 Bean Instance를 재활용할 수 있기 때문에 Bean Pool을 설정할 수 있다. 다음은 Stateless Session Bean의 TTP와 Bean Pool의 관계를 나타낸 그림이다.



Stateless Session Bean의 TTP과 Bean Pool

1. create 시점에 클라이언트와 연결을 맺은 connection으로 요청을 하면 TTP에서 TT(Thread Ticket)를 발급받는다. TT를 발급받지 못하면 TT를 발급받을 때까지 기다린다. 기다리는 시간이 10분을 초과하면 RemoteException이 발생하여 요청을 수행하지 못한다. (EJB의 공통 특성 참고)
2. TT를 발급받은 클라이언트의 요청을 수행하기 위해 실제 Bean instance를 Bean Pool로부터 할당받아 connection과 연결을 맺는다.

3. Stateless Session Bean Instance가 처리를 끝냈을 때 TT는 TTP로 반환되고 Bean Instance는 Bean Pool로 반환되어 다음 클라이언트 요청을 기다린다. 즉, 매 요청마다 TT와 Bean Instance를 새로 할당받고 요청이 끝나면 반환된다.

Bean Pool 개수의 의미가 TTP 개수의 의미와 다른 점은 동시 수행될 수 있는 로컬 클라이언트와 리모트 클라이언트의 요청의 개수와 관련된다는 점이다. 즉, Remote call은 TTP에서 TT를 할당받은 후 Bean Pool에서 Instance를 할당받을 수 있고, Local call은 클라이언트 스레드에서 바로 수행되므로 TT를 발급받을 필요가 없고 Bean Pool로부터 Instance만 할당받는다.

따라서 jeus-ejb-dd.xml의 <thread-max>의 값을 적게 설정하면 설정값 이상의 Remote call은 받을 수 없고 Bean Pool의 최댓값은 무한대이기 때문에 Local call은 처리할 수 있다.

7.1.2. Web Service Endpoint

EJB 2.1 이후부터 Stateless Session Bean은 웹 서비스 형태로 Export될 수 있다.



이에 대한 자세한 정보는 "JEUS Web Service 안내서"를 참고한다.

7.2. Stateful Session Bean

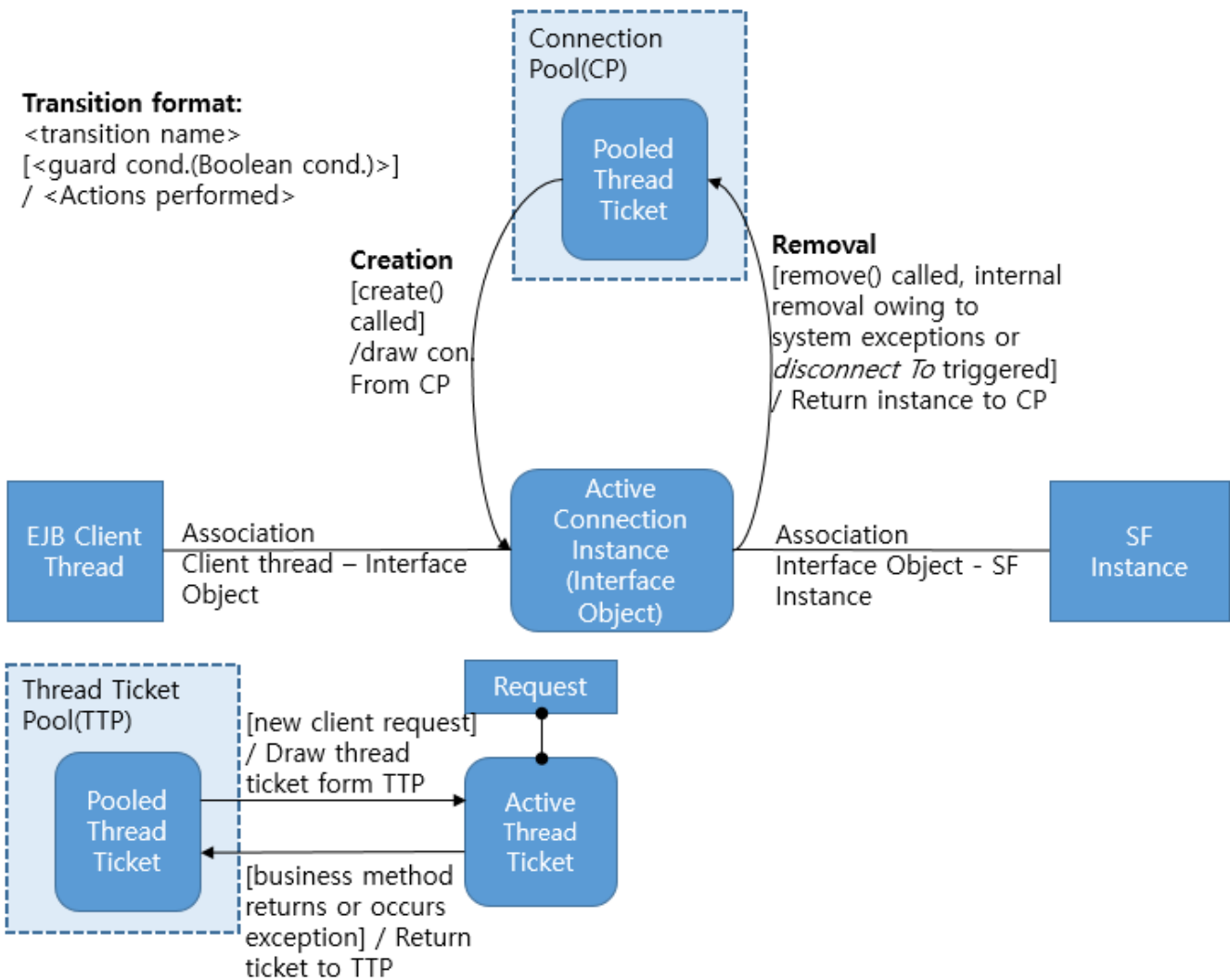
본 절에서는 다음의 Stateful Session Bean의 추가적인 설정들의 개념에 대해 설명한다.

- Object Management 설정 : Bean Instance Pool과 Connection Pool
- Bean Instance Pooling 옵션
- Session Manager를 통한 상태 유지 메커니즘

7.2.1. Thread Ticket Pool(TTP)과 Object Management

Stateful Session Bean은 클라이언트의 요청에 관련된 상태 정보를 지속적으로 유지하고 있어야 하기 때문에 Stateless Session Bean과는 달리 Connection Pool을 설정하고 활용해야 한다. 이는 Entity Bean에도 동일하게 적용된다. 자세한 내용은 [Entity Bean](#)을 참고한다.

다음은 Stateful Session Bean의 Connection Pool과 TTP, Bean Pool의 관계를 나타낸 그림이다.



Stateful Session Bean의 Connection Pool과 TTP, Bean Pool

클라이언트가 Stateful Session Bean을 생성하면 새로운 Bean Instance(SF Instance)가 만들어지고 Connection Pool에서 connection을 꺼내 2개가 연결된다. Bean Instance와 연결된 connection을 클라이언트에게 넘겨주면 이 connection은 현 클라이언트에게 할당되고 다른 클라이언트와 공유하지 않는다. 따라서 클라이언트가 제거하지 않는 한 다시 Connection Pool로 반환되지 않는다. Connection Pool로 반환될 때에는 연결되었던 Bean Instance와의 연결을 끊는다.

해당 connection으로 요청을 하면 TTP에서 TT를 받는다. TT를 받은 요청만 처리한다. Stateless Session Bean과 달리 매 요청마다 새로운 Bean Instance가 Bean Pool로부터 할당되는 것이 아니라 해당 클라이언트를 위한 Bean Instance가 고정되어 있다.

앞에서 설명했듯이 이 Bean Instance는 클라이언트가 Bean을 제거했을 때 connection이 Connection Pool로 반환되면서 connection과의 연결을 잃으면서 삭제된다. 기본적으로 Stateful Session Bean은 상태가 있기 때문에 Bean Instance를 재활용하는 Bean Pool을 사용하지 않는다.

7.2.2. Pooling Session Bean

EJB 표준에 따르면, 실제 Stateful Session Bean의 Instance는 다른 클라이언트 Session에서 재사용할 수 없다. 그러나 Bean이 제거될 때 초기화가 제대로 이루어진다면 Bean Instance를 재사용할 수도 있다. JEUS는 Bean Instance를 매번 생성하지 않는 점에서 성능 개선과 자원 낭비를 줄이기 위해 이를 재사용하는 방법을 제공한다.

Bean Pool을 사용하면 Bean이 제거될 때 해당 Bean Instance가 Bean Pool로 반환된다. Bean의 제거는 클라이언트가 명시적으로 remove를 부르거나 오랫동안 요청이 없어 타임아웃이 된 경우에 발생한다. 이때 PRE DESTROY callback이 불리게 되는데 여기서 Bean의 초기화를 잘 구현한 경우에만 사용하도록 권장한다.

7.2.3. Bean Pool 설정

Stateful Session Bean을 Pooling Bean으로 전환하려면 간단히 jeus-ejb-dd.xml의 <pooling-bean> 값을 true로 설정한다.

Bean Pool 설정 : <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>
    <object-management>
      <pooling-bean>true</pooling-bean>
      <bean-pool>
        <pool-min>10</pool-min>
        <pool-max>200</pool-max>
        <resizing-period>1800000</resizing-period>
      </bean-pool>
      . . .
    </object-management>
    . . .
  </jeus-bean>
  . . .
</beanlist>
. . .
</jeus-ejb-dd>
```

7.2.4. 세션 데이터 유지 메커니즘 설정

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 런타임할 때의 상태는 Instance 변수로 저장된다. 그러나 위에서 설명한 것과 마찬가지로 Stateful Session Bean이 passivate 상태로 진입하면 시스템 자원을 보존하기 위해 운영환경에서 이 Instance 변수들을 제거해야 한다. 그러나 Bean이 다시 재활성화되면 상태를 복구하기 위해서 어딘가에서 데이터를 가져와야 한다. 즉, 2차 저장소가 필요하다.

JEUS에 포함된 분산식 Session Manager를 2차 저장소로 사용한다. 기본적으로는 passivate될 때 클러스터링으로 구성된 Stateful Session Bean의 경우는 Failover를 위해 트랜잭션의 commit이 성공적으로 수행될 때로 Bean의 정보가 Session Manager로 전달된다. Session Manager는 MASTER에서 설정된다.

7.3. 공통 설정

본 절에서는 Stateless Session Bean과 Stateful Session Bean의 공통 설정 항목에 대해서 설명한다. 이 모든 항목들은 JEUS EJB 모듈 DD의 **<jeus-bean>** 태그 아래에 설정된다.

7.3.1. Object Management 관련 설정

Stateless Session Bean은 상태가 없어서 하나의 connection만 사용해도 무방하기 때문에 Bean Pool만 사용할 수 있고, Stateful Session Bean은 상태가 있기 때문에 Connection Pool과 Bean Pool을 모두 사용할 수 있다. 따라서 다음과 같이 제공되는 설정을 통해 Pool을 이용하면 매번 Instance를 생성하는 부하를 줄일 수 있다.

다음은 Object Management 관련 설정한 XML 예제이다.

Object Management 설정 : <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>
      <passivation-timeout>10000</passivation-timeout>
      <disconnect-timeout>1800000</disconnect-timeout>
    </object-management>
  </jeus-bean>
. . .
</beanlist>
. . .
</jeus-ejb-dd>
```

다음은 <object-management>의 하위 설정 태그에 대한 설명이다.

- **<bean-pool>**
 - EJB Bean Instance Pool의 작동 방식을 결정한다.
 - 하위 태그들은 다음과 같다.

태그	설명
<pool-min>	Pool을 초기화할 때 생성해두는 초기 Bean Instance 개수이다. (기본값: 0)
<pool-max>	Instance의 사용이 완료된 후 Pool에 저장 가능한 최대 Bean Instance 개수이다. (기본값: 100)

태그	설명
<resizing-period>	Pool의 크기를 재조정하는 시간으로 "resizing"은 사용되지 않는 Bean Instance들이 Pool의 최솟값까지 제거됨을 의미한다. (기본값: 5분, 단위: ms)

• <connect-pool>

- 클라이언트와 Bean Instance를 연결하는 connection을 몇 개까지 유지할지를 설정한다(해당 옵션은 JEUS 6 Fix#8부터 Session Bean에서는 사용하지 않는다).
- 하위 태그들은 다음과 같다.

태그	설명
<pool-min>	Pool을 초기화할 때 생성해두는 초기 connection 개수이다. (기본값: 0)
<pool-max>	connection 사용이 완료된 후 Pool에 저장 가능한 최대 connection 개수이다. (기본값: 100)
<resizing-period>	Pool의 크기를 재조정하는 시간으로 "resizing"은 사용되지 않는 Bean Instance들이 Pool의 최솟값까지 제거됨을 의미한다. (기본값: 5분, 단위: ms)

• <passivation-timeout>

- 지정된 시간 동안 클라이언트의 요청을 받지 않은 Stateful Session Bean을 passivate할 때 사용된다. 그러므로 여기에 설정된 시간을 초과하는 동안 클라이언트의 요청이 없으면 그 Bean은 passivation의 대상이 된다.

어떤 Bean이 passivation 대상이 되는지를 검사하는 주기는 EJB 엔진에 설정한 resolution을 따른다. Passivation이 실행되면 메모리에서 해당하는 Bean Instance가 제거되고 Instance의 상태는 파일에 저장된다. 내부적으로 분산 세션 서버를 사용한다.

- 이 설정은 여러 곳에서 설정이 가능하고 우선순위는 다음과 같다.
 1. 특정 Session Bean에만 적용 : jeus-ejb-dd.xml의 <passivation-timeout>
 2. 모든 Stateful Session Bean에 적용 : 시스템 프로퍼티 jeus.ejb.stateful.passivate
 3. 모든 EJB Bean에(Entity Bean과 Session Bean 모두) 적용 : 시스템 프로퍼티 jeus.ejb.all.passivate
 4. 모든 EJB Bean에(Entity Bean과 Session Bean 모두) 적용 : Master의 <node>/<session-router-config>/<session-router>/<file-db>/<passivation-to>

위의 모든 설정이 없으면 기본값으로 설정된다. (기본값: 300000(5분), 단위: ms)

• <disconnect-timeout>

- 지정된 시간 동안 클라이언트의 요청을 받지 못하면 클라이언트와 Stateful Session Bean Instance 사이를 연결하던 connection을 끊을 때 사용된다. 그렇게 되면 connection은 각각의 클라이언트 및 Instance와 맺고 있던 연결을 끊고 Connection Pool로 반환된다.

따라서 클라이언트는 이 connection으로 더 이상 요청을 할 수 없고, 사용 중이던 Bean Instance는 삭제되거나 Bean Pool을 사용 중이면 Bean Pool로 반환된다.

- 이 설정은 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.

1. 특정 Session Bean에만 적용(순서대로 우선순위)
 - a. ejb-jar.xml의 <stateful-timeout>
 - b. @StatefulTimeout
 - c. **(Deprecated)** jeus-ejb-dd.xml의 <disconnect-timeout>
2. 모든 Stateful Session Bean에 적용 : 시스템 프로퍼티 jeus.ejb.stateful.disconnect
3. 모든 EJB Bean에(Entity Bean과 Session Bean 모두) 적용 : 시스템 프로퍼티 jeus.ejb.all.disconnect

위의 모든 설정이 없으면 시스템 프로퍼티 jeus.ejb.all.disconnect의 기본값으로 설정된다.

(기본값: 3600000(1시간), 단위: ms)



<passivation-timeout>과 <disconnect-timeout>에 사용되는 시간은 Bean Instance에 액세스했던 마지막 시점부터 측정된다. 그러므로 <disconnect-timeout>을 <passivation-timeout>보다 길게 설정해야 한다. 또한 <passivation-timeout>은 EJB 엔진에 설정되는 resolution보다는 커야한다.

타임아웃 값을 길게 설정하면 오랜 시간(대략 십여 분 이상 또는 타임아웃이 중지된 경우) 동안 메모리 내의 많은 Instance가 활성화된 상태로 머물러 있다. 그러므로 시스템 자원 낭비를 초래한다. 너무 짧은 타임아웃 값은(수 초) passivation, activation 등의 작업이 너무 자주 발생하므로 성능을 저하시킬 수 있고 disconnect 작업으로 인해 세션 유실 가능성이 있다.

8. Entity Bean

Entity Bean은 EJB 3.0부터는 JPA로 대체되었다. 따라서 Bean을 새롭게 개발하는 경우는 JPA 사용을 권장한다. 그러나 기존 사용자를 위해 JEUS EJB 엔진은 Entity Bean을 지원한다.

본 장에서는 JEUS EJB 엔진에서 Entity Bean을 설정하고 튜닝하는 데 필요한 모든 정보에 대해 설명한다.

8.1. 개요

EJB 스펙에 따라 JEUS EJB 엔진은 다음과 같은 3가지 Entity Bean을 지원한다.

- Bean Managed Persistence Entity Bean (BMP)
- EJB 1.1 스펙에 따른 Container Managed Persistence Entity Bean(CMP 1.1)
- EJB 2.0 스펙에 따른 Container Managed Persistence Entity Bean(CMP 2.0)

종류에 상관없이 JEUS의 Entity Bean들은 모두 [EJB의 공통 특성](#)에서 설명한 기능과 설정 컴포넌트들을 공유하므로 JEUS EJB 엔진에서 Entity Bean을 설정하고 사용하기 전에 관련 장의 내용에 대한 이해를 필요로 한다.

3개의 Bean 종류에 따라 3개의 설정이 존재한다.

- 3가지 종류에 모두 적용되는 공통된 설정
- **CMP 1.1**과 **CMP 2.0**에만 적용되고 BMP에는 적용되지 않는 설정
- **CMP 2.0**에만 적용되는 설정

다음은 각 Entity Bean 종류별 설정 특성을 나타낸 표이다.

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. use-access-control	✓	✓	✓
7. Run-as identity	✓	✓	✓
8. Security CSI Interop.	✓	✓	✓
9. Env. refs	✓	✓	✓
10. EJB refs	✓	✓	✓
11. Resource Refs	✓	✓	✓
12. Resource env. Refs	✓	✓	✓
13. Thread ticket pool settings	✓	✓	✓

설정 가능한 Entity 옵션	BMP	CMP 1.1	CMP 2.0
14. Clustering settings	√	√	√
15. HTTP invoke	√	√	√
16. Object management	√	√	√
17. Persistence Optimize	√	√	√
18. CM persistence opt.		√	√
19. Schema info		√	√
20. Relationship map			√

위의 표에서와 같이 항목 1부터 15까지는 모든 EJB에 공통으로 적용된다. 이 항목들은 [EJB의 공통 특성](#)에서 모두 설명하였다.

16번 항목은 Session Bean과 Entity Bean에 모두 해당되지만 사용법은 차이가 있어서 그 차이점을 중점적으로 설명한다. 항목 17부터 20까지는 Entity Bean에만 해당하는 항목들로 본 장에서 자세히 설명한다.

이 외에도 다음과 같은 JEUS Entity EJB에 연관된 주제에 대해서도 설명한다.

- Default Primary Key 클래스
- EJB QL 언어에 추가된 JEUS 특정 항목
- JEUS Instant EJB QL API



JEUS에서는 성능 튜닝이 EJB Entity Bean 설정의 가장 중요한 부분이다. 그러므로 본 장에서는 엔진 모드와 서버 모드와 같이 성능 관련 부분에 대해서 자세히 설명한다. 본 장 후반부의 [Entity EJB 튜닝](#)은 본 절에서 설명한 내용을 좀 더 간략하게 다시 정리한 것이다.

8.2. 주요 기능

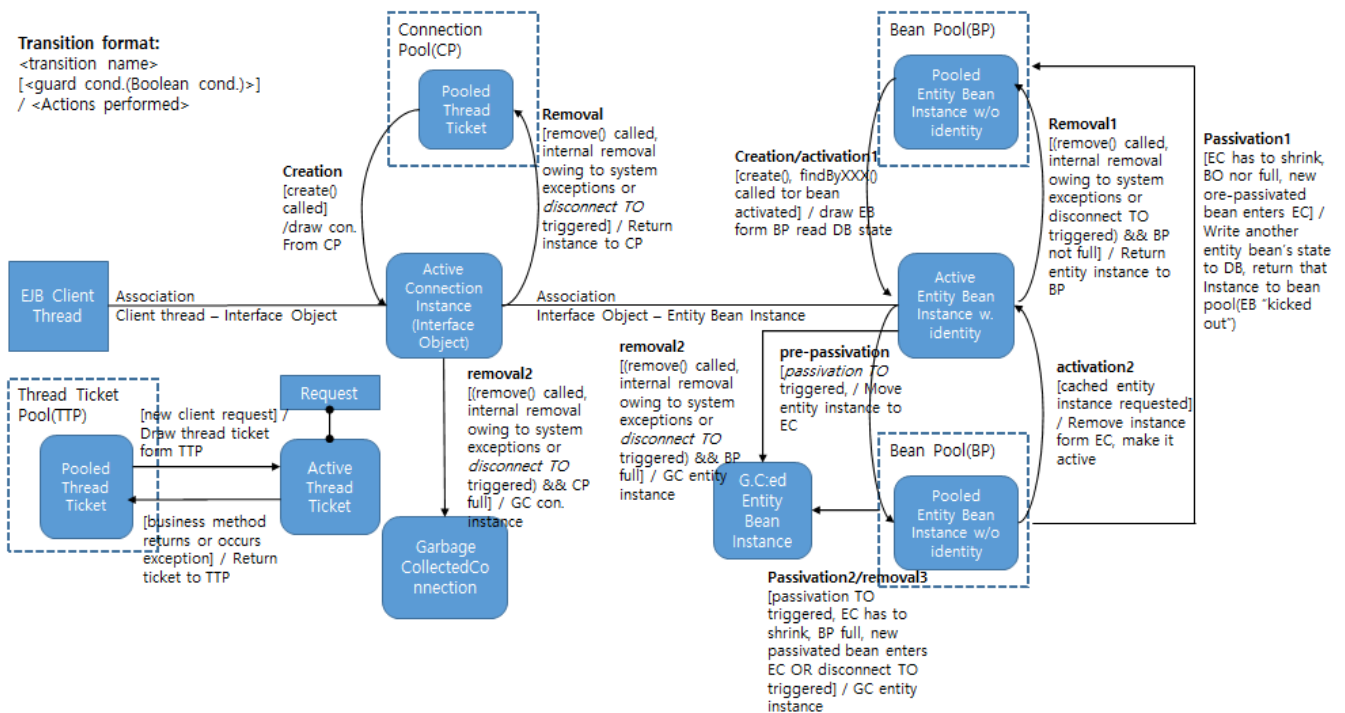
본 절에서는 Entity Bean의 공통 기능과 각 Bean의 주요 기능에 대해서 설명한다.

8.2.1. 공통 기능

3가지 Entity Bean들은 종류에 상관없이 JEUS Entity Bean의 공통 기능을 갖는다.

8.2.1.1. Entity Bean Object 관리 및 Entity Cache

다음은 EJB 엔진에서 Entity Bean Object가 어떻게 관리되는지 보여준다.



JEUS EJB 엔진에서 Entity Bean의 Object와 Instance 관리

"object management"는 EJB 엔진의 내부적인 Bean Instance Pool과 Connection Pool을 의미한다. 이 Pool들은 각 Bean마다 설정되고 성능 향상과 시스템 자원의 효율적인 관리를 위해 사용된다. 기본적으로 Stateful Session Bean과 비슷하게 동작하므로 본 절에서는 차이점만 언급하고 자세한 내용은 [Session Bean](#)을 참고한다.

Entity Bean은 Connection Pool 외에 Entity Cache를 사용한다. Entity EJB가 passivate되려 할 때(passivation timeout이 지났을 때) 그 Bean Instance는 실제로는 바로 passivate되지 않는다(Stateful Session Bean의 경우에는 passivate된다). 대신에 이는 Entity Cache로 전달된다. 이 Cache에서는 모든 passivate된 Entity Bean들이 활성화 상태로 저장되어 있다(즉, 식별이 가능하고, 유효한 상태를 유지한다). 다른 Pool들과 마찬가지로 이 Entity Cache도 런타임 메모리에 유지되고, 따라서 활성화 상태와 passivate 사이의 존재하지 않는 상태에서 Runtime Cache로 작동된다.

Entity Bean이 실제로 passivate되는 때는 다음과 같다.

Entity Cache가 사이즈를 줄여야 할 때 passivate가 된다. 새로운 Entity Bean이 Cache로 들어오려 할 때 그 Cache에 이미 passivate된 Bean들이 많이 있을 경우 Cache 내의 Bean들이 강제로 밀려 나간다. 이렇게 강제로 밀려 나간 Entity Bean은 passivate되고 그 상태는 DB에 쓰여지며 그 Instance는 Bean Pool로 반환되거나 Bean Pool이 가득 차 있는 상태라면 버려진다. Cache에 막 들어오려는 Entity Bean은 Cache에 진입하여 결과적으로 강제로 밀려난 Bean Instance의 자리를 차지하게 된다.

Cache 크기는 <persistence-optimize>/<entity-cache-size>로 조절 가능하다. 반드시 설정한 크기만큼 찼을 때 Cache 크기를 줄이는 것은 아나라 힌트로 사용된다. 또한 강제로 밀려나가는 Bean Instance를 선택하는 방법 또한 Cache에 오래 머문 순이 아니다.

이러한 Cache를 사용하면 시간이 많이 소모되는 passivation을 효과적으로 관리할 수 있다는 장점이 있다. Cache를 사용함으로써 passivate된 Entity Bean을 다시 활성화시킬 때 DB에 접근할 필요가 없어지고 Cache에 있는 Instance(만약에 존재하면)를 사용함으로써 해결할 수 있다.

Object Pool의 표준 passivation timeout 값이 시간적인 제약을 관장하는 passivation 값이라면 Entity Cache는 메모리의 제약을 관장하는 passivation 설정값이라고 할 수 있다. 이것이 Entity Cache를 사용하는 주요 동기이다.



Entity Cache 설정은 실제로 "object management"의 설정이 아니라 일반적인 Entity Bean persistence 최적화 설정이다. 이에 대한 자세한 내용은 [Entity EJB 설정](#)에서 설명한다.

8.2.1.2. 엔진 모드 Selection을 통한 ejbLoad() Persistence 최적화

Entity Bean에 대해 어느 시기에 ejbLoad() 메소드를 호출할 것인지 결정하는 것은 EJB 엔진(컨테이너)이다. 이 메소드는 DB의 상태 정보와 Bean Instance의 런타임 상태 정보를 동기화하기 위해 BMP와 CMP에서 사용한다. 본질적으로 ejbLoad() 메소드는 DB에서 하나의 레코드를 읽고 그 결과를 Bean의 내부 특성 값들에 설정하기 위해 사용된다.

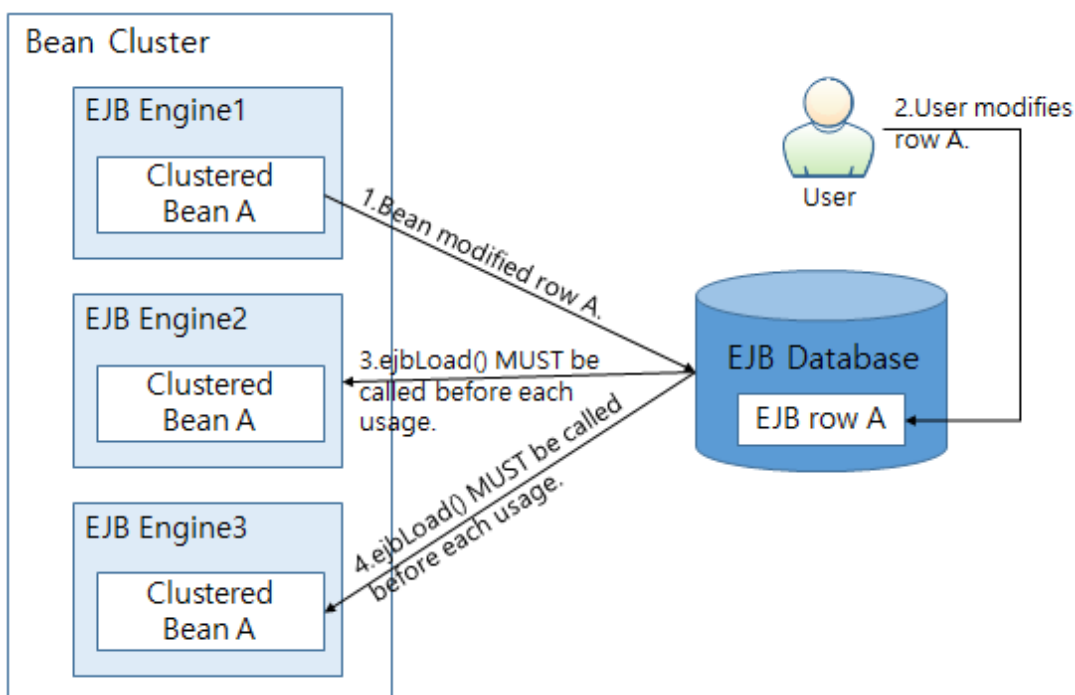
ejbLoad() 메소드는 Entity Bean의 생명주기 동안 적어도 한 번은 호출된다. 어떤 경우에는 Bean Instance의 호출할 때마다 바로 전에 ejbLoad() 메소드를 호출할 필요가 생기기도 한다.

DB에서 Bean의 상태 정보를 읽어야 하는 시나리오에는 기본적으로 크게 2가지가 있다.

- Entity Bean이 여러 개의 EJB 엔진에 걸쳐 클러스터되어 있을 때 그 중 하나의 Bean이 DB의 한 레코드를 변경시킬 수 있다. 즉, 클러스터링된 Entity Bean들이 서로 통신할 수 없기 때문에 특정 Entity Bean Instance의 상태 정보가 가장 최신의 것이라는 것을 보장할 수 없다.
- 어떤 외부 요소(작업자 또는 시스템)가 EJB와 연동되어 있는 DB를 변경할 경우가 있다.

이 시나리오에서 EJB 엔진의 특정 EJB Instance는 DB의 내용을 제대로 반영하지 못한 상태가 된다.

다음 그림은 Entity Bean이 클러스터링되어 있을 때나 외부에서 Bean의 DB 행을 수정했을 때 ejbLoad()가 주기적으로 호출되는 2개의 시나리오이다.

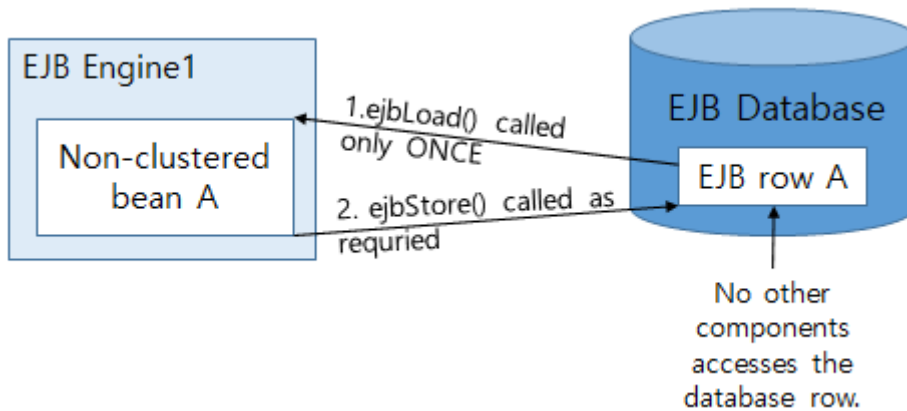


ejbLoad() 주기적 호출 시나리오

위의 두 시나리오의 상황을 피할 수만 있다면(클러스터링을 사용하지 않고, 어떤 외부 요소도 EJB의 DB 데이터를 변경하지 않는다고 보장한다면) Entity Bean의 ejbLoad() 동작을 최적화할 수 있다.

EXCLUSIVE_ACCESS 엔진 모드를 Bean에 적용하면 최적화가 가능하다. 이 모드를 사용하면, ejbLoad()는 Bean이 Instance화될 때 단 한 번만 호출된다. 이렇게 함으로써 DB의 접근량을 약 50%정도 줄일 수 있다.

다음 그림은 EXCLUSIVE_ACCESS 모드가 사용될 때의 시나리오를 보여주고 있다. EXCLUSIVE_ACCESS는 단지 하나의 Bean이 DB 행을 사용하고 대부분의 ejbLoad()는 최적화된다.



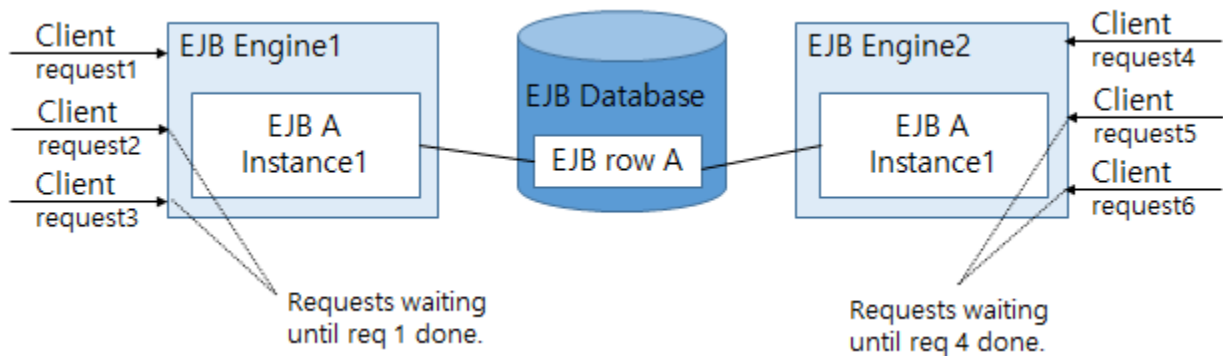
EXCLUSIVE_ACCESS 모드의 시나리오

Bean을 클러스터링해야 하거나 외부 요소가 EJB 데이터를 변경시켜야 한다면 SINGLE_OBJECT 또는 MULTIPLE_OBJECT 엔진 모드를 사용해야 한다. 두 모드 중 하나가 사용되면, EJB 엔진은 EJB 클라이언트의 요청이 도착할 때마다 ejbLoad() 메소드를 호출한다. 이렇게 하면 각 Entity Bean의 성능이 저하되지만 모든 상태 정보가 DB에 의해 관리되기 때문에 다른 엔진과 Bean들에게 부하를 분산시킬 수 있다.

다음은 SINGLE_OBJECT와 MULTIPLE_OBJECT 모드의 차이점에 대한 설명이다.

- SINGLE_OBJECT 엔진 모드

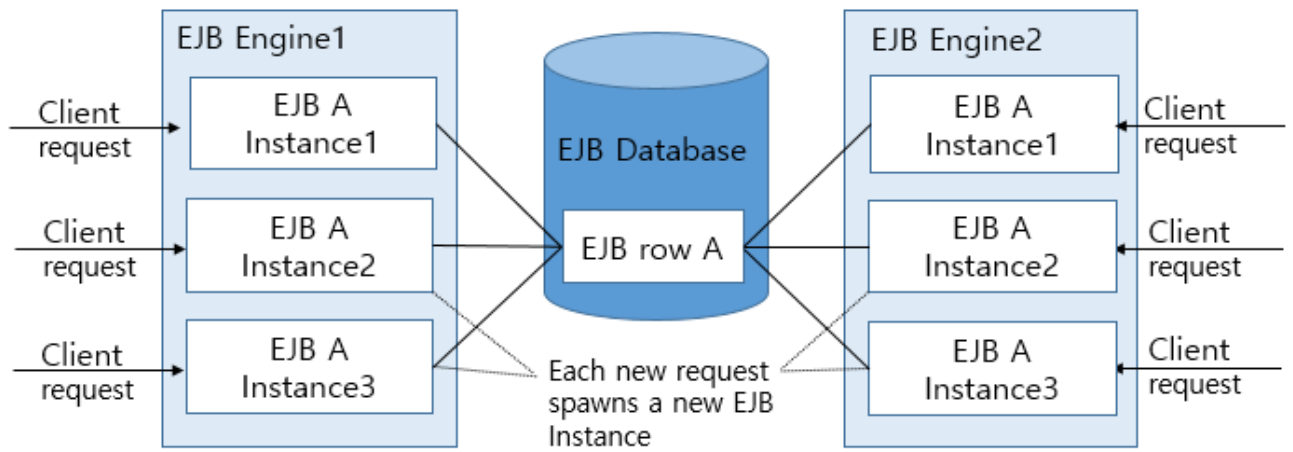
SINGLE_OBJECT에서는 각 EJB 엔진의 단 한 개의 Bean Instance만이 모든 클라이언트의 요청을 처리한다. 즉, 첫 요청이 처리되고 있는 동안 도착한 요청들은 대기하고 있어야 한다.



SINGLE_OBJECT 엔진 모드

- MULTIPLE_OBJECT 엔진 모드

MULTIPLE_OBJECT 모드에서는 이러한 제약이 적용되지 않고 EJB 엔진이 새로운 Entity Bean Instance를 생성하여 새로운 EJB 클라이언트 요청을 처리한다. 즉, MULTIPLE_OBJECT 엔진 모드에서는 새로운 EJB Instance가 할당된다.



MULTIPLE_OBJECT 엔진 모드

다음은 각 엔진 모드를 선택할 때의 장점과 단점을 비교한 표이다.

구분	EXCLUSIVE	SINGLE	MULTIPLE
클러스터링을 허용하는가?	No	Yes	Yes
외부 Entity가 DB에 접근 가능한가?	No	Yes	Yes
DB connection이 효율적으로 사용되는가?(ejbLoad() 호출빈도)	Yes	No	No
장시간의 트랜잭션의 경우 적당한 옵션을 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

3가지 엔진 모드 중 적합한 모드를 선택하기 위한 기준은 다음과 같다.

- 많은 양의 요청이 Entity Bean에 요청되고, 많은 수의 EJB 엔진들이 설정되어 있다면 SINGLE_OBJECT 모드를 선택하고 엔진들에 Bean을 클러스터링으로 구성한다.
- 많은 양의 요청이 Entity Bean에 요청되지만, 한정된 EJB 엔진이 설정되어 있으면 MULTIPLE_OBJECT 모드를 선택하고 엔진들에 Bean을 클러스터링으로 구성한다.
- MULTIPLE_OBJECT와 SINGLE_OBJECT를 선택할 때 고려해야 할 중요한 사항 중의 하나는 한 트랜잭션을 수행하기 위해 필요한 총 시간이다. 수행 시간이 많이 소요되는 트랜잭션이라면 당연히 MULTIPLE_OBJECT를 선택할 것을 권장한다.
- 많지 않은 동시 요청이 Entity Bean에 들어온다면 EXCLUSIVE_ACCESS 모드를 선택하고 하나의 EJB 엔진에 Bean을 deploy한다.

EJB 클러스터링에 대한 자세한 정보는 [EJB 클러스터링](#)의 설명을 참고한다.

8.2.2. BMP & CMP 1.1

본 절에서는 ejbStore() 호출을 최적화하는 방법에 대해 설명한다.



본 절에서 설명하는 ejbStore() 호출의 최적화 방법은 BMP와 CMP 1.1 Bean들에만 적용되고, CMP 2.0은 다른 방법으로 접근해야 한다.

8.2.2.1. Non-modifying 메소드에 의한 ejbStore() Persistence 최적화

ejbStore()는 EJB 엔진에 의해 DB의 Bean 상태 정보가 유지되어야 한다고 판단될 때마다 호출된다. 일반적으로 해당 DB로의 접근은 각 트랜잭션의 commit 또는 Bean Instance가 passivate되기 전에 시도된다.

그러나 만약에 트랜잭션의 중간에 Bean의 상태가 바뀌지 않으면 ejbStore()가 호출될 이유가 없다.

JEUS EJB 엔진은 Bean의 상태 변경을 자동적으로 예측할 수 없다. 그러므로 개발자나 deployer는 ejbStore()를 호출하는 시점을 EJB 엔진이 판단할 수 있도록 힌트를 제공해야 한다. 이 힌트는 Non-modifying 메소드들의 목록으로 제공된다. Non-modifying 메소드는 근본적으로 읽기만 허용하는 메소드(getter)로서 Entity Bean의 DB 내부 상태를 변경하지 않는다.

Non-modifying 메소드의 목록을 보고 EJB 엔진은 ejbStore() 메소드를 호출할지 여부를 결정한다. 트랜잭션 중 또는 Bean이 활성화된 상태일 때 Non-modifying 메소드만 실행되면 EJB 엔진은 ejbStore() 호출이 불필요하다고 판단하고 그 과정을 생략한다. 이는 전체적인 성능이 향상됨을 의미한다.

8.2.3. CMP 1.1/2.0

본 절에서는 CMP 1.1/2.0의 주요 기능에 대해서 설명한다.

8.2.3.1. ejbLoad(), ejbFind(), CM Persistence 최적화

엔진 모드 Selection을 통한 ejbLoad() Persistence 최적화와 Non-modifying 메소드에 의한 ejbStore() Persistence 최적화에서 ejbLoad()와 ejbStore() 메소드의 호출 시점에 대해서 설명하였다. BMP와 CMP는 ejbLoad()와 ejbStore() 메소드를 사용하기 때문에 BMP와 CMP Entity Bean에만 적용되었다.

본 절에서는 성능향상을 위해 ejbLoad()와 ejbFind() 메소드를 최적화하는 방법에 대해서 설명한다. ejbLoad(), ejbFind()의 구현 방법은 EJB 엔진에 영향을 받기 때문에 CMP Bean(1.1과 2.0 버전)에 국한해서 설명한다. BMP에서 해당 메소드의 구현과 작동 방식은 개발자에 의해 결정된다.

EJB 엔진이 CMP ejbLoad(), ejbFind()를 호출하면 다음과 같은 엔진의 하위 모드를 선택하여 작동 방식을 최적화할 수 있다. 3가지 모드에 대한 자세한 기술적 설명은 하지 않는다.

모드	설명
ReadLocking	ejbLoad()를 호출할 때 EJB 엔진이 DB의 Shared Lock을 가질 수 있게 한다. 이 종류의 Lock은 다른 Bean들이 같은 레코드를 읽을 수 있지만 쓰는 것은 금지한다.
WriteLocking	ejbLoad()를 호출할 때 EJB 엔진이 DB의 Exclusive Lock을 가질 수 있게 한다. 다른 Bean들이 DB에 읽고 쓰는 것을 모두 금지한다.
WriteLockingFind	ejbLoad()와 ejbFind()를 호출할 때 EJB 엔진이 DB의 Exclusive Lock을 가질 수 있게 한다. 다른 Bean들이 DB에 읽고 쓰는 것을 모두 금지한다.

위의 3가지 모드가 사용되어야 하는 상황은 다음과 같다.

- CMP Bean이 쓰기 작업보다는 읽기 작업을 더 빈번히 수행하면 항상 ReadLocking 모드를 선택한다.
- CMP Bean이 읽기 보다는 쓰기 작업을 더 많이 수행하면 WriteLocking 모드나 WriteLockingFind 모드를 선택한다.

CMP에서는 엔진 모드와 앞 절에서 설명한 Non-modifying 메소드, 그리고 본 절에서 설명한 엔진 내 하위 모드가 규칙에 맞게 조합되어 설정되어야 한다. 이 모든 파라미터들의 설정 요약은 [Entity EJB 튜닝](#)의 설명을 참고한다.

또한 CMP Bean에서는 java.sql.ResultSet의 Fetch Size, EJB 엔진이 기동(Booting)할 때 Caching되어야 하는지의 여부와 EJB Bean Instance가 미리 Instance되어야 할지 등을 결정하는 <init caching> 설정을 할 수 있다.

8.2.3.2. Entity Bean 스키마 정보

EJB 엔진이 Bean의 persistence를 관리하는 책임을 가지고 있다면(CMP), Entity Bean Instance 필드와 실제 DB의 테이블과 컬럼들과의 매핑을 선언적으로 정의해주는 방법이 있어야 한다. 또한 DB 소스도 지정해주어야 한다. 이 정보는 EJB XML 환경에 <schema info>로 기술한다.

CMP 1.1의 이 태그에는 EJB 엔진이 CMP 1.1 finder 메소드를 생성할 때 기본적으로 사용될 SQL 문도 포함된다.

8.2.3.3. 자동 Primary Key 생성

경우에 따라 개발자들은 Primary Key 생성 없이 EJB Instance를 생성해야 할 때가 있다. 이 경우 JEUS EJB 엔진이 새로운 EJB Instance에게 할당할 고유의 Primary Key를 생성하도록 설정할 수 있다.

JEUS EJB 엔진은 이 기능을 DB와 연동하여 제공한다. 이러한 자동 Primary Key 생성 기능에 대해 JEUS EJB 엔진은 DB와 연동하여 제공하고, CMP 1.1과 CMP 2.0에 적용된다. 이 기능을 사용하면 Primary Key에 신경쓰지 않고 손쉽게 EJB를 개발할 수 있다.

다음은 자동 Primary Key 생성 기능의 사용 방법과 적용 과정에 대한 설명이다.

1. 개발자들은 습관적으로 CMP1.1/CMP2.0을 파라미터 없이 create()를 호출한다. 이것은 create() 메소드가 EJB의 Primary Key가 되는 파라미터를 갖지 않는다는 의미이다.

```
InitialContext ctx = new InitialContext();
BookHome bHome = (BBookHome) ctx.lookup("bookApp");
Book b = bHome.create("JEUS EJB Guide", "Park Sungho");
// Start using "b".
. . .
```

위의 예제에서 EJB Home인 BookHome을 찾고 EJB Home에서 파라미터가 2개인 create()를 호출한다("JEUS EJB Guide"와 "Park Sungho"). 이 파라미터는 새로운 책의 제목과 저자 이름을 뜻한다. 저자는 다르나 동일한 제목의 책이 있을 수 있기 때문에 이 파라미터들을 Primary Key로 이용하는 것은 적합하지 않다.

이 기능을 지원하기 위해 EJB 엔진은("A") 반드시 고유의 Primary Key를 생성해서 새 책에 할당해야 한다. 그리고 여러 엔진들("B", "C")이 클러스터링된 상태에서도 같은 일이 동시에 발생할 수 있다는 것을 고려해야

한다.

광범위한 클러스터링 중에도 각 엔진들은 고유한 Primary Key를 얻을 수 있어야 하기 때문에 하나의 중앙 저장장치를 필요로 한다. 이 저장장치는 하나의 컬럼과 하나의 데이터로 이루어진 DB로 구현된다.

이 하나의 값은 EJB 엔진과 연동하면서 계속 증가하는 Primary Key Counter(또는 Primary Key Generator)이다. 이 방법으로 DB는 항상 클러스터링된 EJB 엔진이 읽을 수 있는 새롭고 고유한 Primary Key를 유지할 수 있다.

2. Primary Key가 없는 create()가 호출되면, EJB 엔진 "A"는 중앙 저장장치(Primary Key DB)로 접근한다. 이 저장장치로부터 정수로 된 Primary Key를 받아오고 Book Instance에 할당한다.
3. EJB 엔진 "A"는 저장장치에 있는 미리 결정된 값으로 Primary Key를 증가시킨다. 미리 결정된 값(Key Cache Size)은 기본적으로 "1"이다. 그러나 1 이상을 사용할 것을 권장한다("20").

만약 Key Cache Size가 "20"이라면, EJB 엔진 "A"는 "19" (20-1)를 할당한다. 다음에 Primary Key를 읽는 다른 EJB 엔진("B"와 "C")은 가장 최근에 Primary Key를 읽어간 것보다 20이 큰 값을 읽게 된다.

이것은 EJB 엔진 "A"는 19개까지는 외부의 Primary Key 저장장치에 접근 없이 EJB Instance를 생성할 수 있다는 것을 의미한다. 따라서 성능 향상에 큰 기여를 한다. 엔진 "A"는 할당받은 Primary Key의 Counter를 유지함으로써 이 값을 모두 소비했을 때에만 외부에서 새로운 Primary Key를 읽어 온다. 내부적인 Counter는 물론 Primary Key를 생성하는 데 사용된다.

다음은 자동 Primary Key 생성의 특징에 대한 설명이다.

- Oracle DB와 Microsoft SQL Server는 자동으로 Primary Key Sequence를 제공한다. 자동 Primary Key 생성 설정에 대한 자세한 내용은 [Oracle DB에서 Primary Key 생성 설정](#)과 [Microsoft SQL Server에서 자동 Primary Key 생성 설정](#)을 참고한다.
- Non-Oracle DB와 Non-Microsoft SQL Server는 직접 Primary Key 테이블을 작성해야 한다. 자세한 내용은 [Other DB의 자동 Primary Key 생성](#)을 참고한다.
- 자동 Primary Key 생성 기능을 사용하기 위해서는 DB가 반드시 Transaction Isolation Level - TRANSACTION_SERIALIZABLE을 지원해야 한다. 본 안내서에서는 지원한다고 가정한다.
- Primary Key는 EJB에서 반드시 정의되어야 한다. Primary Key는 반드시 java.lang.Integer 타입이고, 단일 Key여야 하며, EJB DD에 선언되어야 한다.



자동 Primary Key 생성 기능은 반드시 TRANSACTION_SERIALIZABLE(Isolation Level)을 지원해야만 한다. 즉, Oracle DB, Microsoft SQL Server 외에는 모두 이 Isolation Level이 지원되는지 확인해야 한다.

다음은 각 DB에 설정하는 방법에 대해 설명한다.

Oracle DB에서 Primary Key 생성 설정

Oracle DB에서 자동 Primary Key 생성은 Oracle에서 사용하는 **Sequence 객체**를 이용한다. 자동 Primary Key 생성 기능을 설정하기 전에 Sequence 객체가 생성되어 있어야 한다. Sequence 설정에 관련한 내용은 Oracle 문서를 참고한다.

다음은 Oracle DB에서 Primary Key를 생성하는 jeus-ejb-dd.xml의 예이다.

Oracle DB에서 Primary Key 생성 설정 : <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 Key 생성 타입을 "Oracle"로 설정했으며, <generator-name>을 "my_generator"로 하고 <key-cache-size>를 "20"으로 설정했다.



<key-cache-size>는 Oracle의 Sequence 객체의 SEQUENCE INCREMENT 값과 일치해야 한다.

Oracle DB는 <data-source-name>에서 선택된다(여기서는 "MYORACLEDB"가 사용되었는데, JEUSMain.xml의 DB Connection Pool JNDI 이름이다). 설정에 대한 자세한 내용은 "JEUS Server 안내서"를 참고한다.

Microsoft SQL Server에서 자동 Primary Key 생성 설정

Microsoft SQL Server에서 자동 Primary Key 생성은 간단하다. Microsoft SQL Server는 자동으로 고유의 IDENTITY 컬럼에 Primary Key를 유지한다.

다음은 Microsoft SQL Server에서 자동 Primary Key 생성을 설정한 jeus-ejb-dd.xml의 예이다.

Microsoft SQL Server에서 자동 Primary Key 생성 설정 : <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"을 입력한다. Microsoft SQL Server는 <data-source-name>에서 선택된다.

Other DB의 자동 Primary Key 생성 설정

만약 Oracle DB와 Microsoft SQL Server 외에 다른 DB를 이용해서 자동 Primary Key를 생성한다면, 다음의 절차를 반드시 지켜야 한다.

1. 다음과 같이 DB에 테이블을 하나 생성한다. 테이블은 하나의 컬럼과 하나의 열을 갖고 있어야 하고, 값은 '0'으로 설정되어야 한다.

다음은 Primary Key 값을 관리할 테이블이다.

	PrimKeyGeneratorColumn (Primary Key 값이 저장되는 컬럼)
Row 1	0 (Primary Key 값의 초기값)

위의 테이블은 다음의 SQL로 생성된다.

```

CREATE table PrimKeyTable (PrimKeyGeneratorColumn int);
INSERT into PrimKeyTable VALUES (0);

```

2. 테이블을 생성하고 jeus-ejb-dd.xml을 다음과 같이 수정한다.

Other DB의 자동 Primary Key 생성 설정 : <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 타입은 'USER_KEY_table'이고, 테이블 이름이 'PrimKeyTable'이다. 컬럼 이름은 'PrimKeyGeneratorColumn'이고 Key Cache 크기는 '20'으로 설정했다.

DB를 사용하기 위해서 설정하는 <data-source-name> 태그의 값은 JEUSMain.xml의 DB Connection Pool의 <export-name>과 일치해야 한다.



설정에 대한 자세한 내용은 "JEUS Server 안내서"를 참고한다.

8.2.4. CMP 2.0

본 절에서는 CMP 2.0의 주요 기능에 대해서 설명한다.

8.2.4.1. Entity Bean Relationship Mapping

EJB 2.0 스펙에는 DB에 있는 관계(Relationship)를 지원하기 위해 CMP Relationship의 개념이 소개되어 있다. CMP 2.0 Bean들과 이 Relationship들을 사용할 때에는 deployer가 JEUS EJB DD에 추가 정보를 제공해야 한다.

8.2.4.2. JEUS EJB QL Extension

CMP 2.0 Bean은 ejb-jar.xml의 EJB QL 문장과 연계되어 있는 home interface의 findByXXXX() 메소드를 가지고 있어야 한다. JEUS는 표준 EJB QL 언어에 몇 가지 추가 사항을 더 지원한다. 이 추가 지원 사항은 ejb-jar.xml에 사용되거나 Instant EJB QL 요청에 사용될 수 있다.

[JEUS EJB QL Extension](#)에는 위의 예가 포함된 완전한 ejb-jar.xml이 포함되어 있다.

8.2.4.3. Instant EJB QL

클라이언트 코드 내에서 CMP Bean의 집합을 찾으려면 Bean의 home interface에 선언되어 있는 findByXXXX()

메소드들에 의존할 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.EJBInstanceFinder라고 불리며, JEUS EJB DD에 Bean의 Instant QL을 활성화시킨 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 인터페이스의 java.util.Collection 객체를 반환한다.

이 메소드의 예제는 [자동 Primary Key 생성](#)을 참고하고, 해당 API에 대한 설명은 Appendix B의 [Instant EJB QL API Reference](#)를 참고한다.

8.3. Entity EJB 설정

본 절에서는 다음과 같은 JEUS Entity EJB의 설정 방법을 설명한다.

- 모든 Entity Bean에 적용 가능한 설정
 - 기본적이고 공통적인 설정들
 - Object Pool
 - Entity Cache 설정을 포함한 Persistence 최적화
- CMP에만 적용 가능한 설정
 - CM persistence optimization 최적화
 - 스키마 정보 설정
- CMP 2.0에만 적용 가능한 설정
 - Relationships 매핑 설정
 - Instant EJB QL 설정

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

8.3.1. 공통 설정

다음은 모든 Entity Bean에 적용 가능한 기본 설정에 대한 설명이다.

8.3.1.1. 기본 공통 항목 설정

[EJB의 공통 특성](#)에서 설명한 JEUS에서 지원하는 모든 Bean 종류에 적용 가능한 설정은 모든 종류의 Entity Bean들에게도 적용할 수 있다.

다음은 BMP Entity Bean의 기본 XML 태그의 예이다.

Entity EJB의 기본 공통 항목 설정 : <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>
    . . .
  </jeus-bean>
  . . .
</beanlist>
. . .
</jeus-ejb-dd>

```

8.3.1.2. Object Management 관련 설정

다음은 Object Management 관련 설정이 된 XML 예제로 **<jeus-bean>**의 **<object-management>** 태그 내에서 구성된다.

Object Management 관련 설정 : <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>5000</capacity>
        <passivation-timeout>10000</passivation-timeout>
        <disconnect-timeout>1800000</disconnect-timeout>
      </object-management>
    </jeus-bean>
    . . .
  </beanlist>
  . . .
</jeus-ejb-dd>

```

다음은 설정 태그에 대한 설명이다.

- **<bean-pool>**
 - EJB Bean Instance Pool의 작동 방식을 결정한다.
 - 하위 태그들은 다음과 같다.

태그	설명
<pool-min>	Pool을 초기화할 때 생성해두는 초기 Bean Instance 개수와 Pool에 유지하려는 최소 Bean Instance의 개수(resizing 작업할 때 Pool에 남기는 Instance의 개수)이다. (기본값: 0)
<pool-max>	Instance의 사용이 끝난 후 Pool에 저장 여부를 결정하는 Pool이 가질 수 있는 최대 Bean Instance 개수이다. Session Bean은 <pool-max> 설정값까지만 생성할 수 있으므로 그 이상의 요청이 한 번에 오면 EJB Exception이 발생한다. 반면, Entity Bean은 Bean Instance 생성에는 제한이 없고 사용 후에 Pool에 보관할 개수를 제한한다. (기본값: 100)
<resizing-period>	Pool의 크기를 재조정하는 시간으로 "resizing"은 사용되지 않는 Bean Instance들이 Pool의 최솟값까지 제거됨을 의미한다. (기본값: 5분, 단위: ms)

• <connect-pool>

- 클라이언트와 Bean Instance를 연결하는 connection을 몇 개까지 유지할 것인지를 설정한다. 해당 옵션은 Session Bean에서는 사용하지 않는다.
- 하위 태그들은 다음과 같다.

태그	설명
<pool-min>	Pool에 유지하려는 최소 connection의 개수로 resizing 작업할 때 Pool에 남기는 Instance의 개수이다. (기본값: 0)
<pool-max>	connection 사용이 끝난 후 Pool에 저장 여부를 결정하는 Pool이 가질 수 있는 최대 connection 개수이다. (기본값: 100)
<resizing-period>	Pool의 크기를 재조정하는 시간으로 "resizing"은 사용되지 않는 connection들이 Pool의 최솟값까지 제거됨을 의미한다. (기본값: 5분, 단위: ms)

• <capacity>

- 생성될 것으로 예상되는 Bean Instance의 최대 개수를 의미하며, Entity Bean에만 사용된다. 이 값은 EJB와 연계될 내부 클라이언트 세션 데이터의 효율적인 구성을 위해 사용된다. (기본값: 10000)

• <passivation-timeout>

- 지정된 시간 동안 클라이언트의 요청을 받지 않은 Bean을 passivate할 때 사용된다.

여기에 설정된 시간을 초과하는 동안 클라이언트의 요청이 없는 Bean Instance가 passivate 대상이 된다. passivate 대상인 Bean Instance가 메모리에 남아 있는 개수를 <persistence-optimize>/<entity-cache-size>로 조절할 수 있다.

- Stateful Session Bean의 경우는 Entity Cache라는 것이 없어 설정한 시간 동안 요청이 없으면 passivate되지만 Entity Bean의 경우는 Entity Cache에 머물러 있다가 passivate된다. 어떤 Bean이 passivate되는지 검사하는 주기는 EJB 엔진에 설정한 resolution을 따른다. (기본값: 5분)

passivate가 실행되면 메모리에서 해당하는 Bean Instance가 제거된다.

- 이 설정은 다음과 같이 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.

1. 특정 Session Bean에만 적용 : jeus-ejb-dd.xml의 <passivation-timeout>
2. 모든 Entity Bean에 적용 : 시스템 프로퍼티 jeus.ejb.entity.passivate
3. 모든 EJB Bean에(Entity Bean과 Session Bean 모두) 적용 : 시스템 프로퍼티 jeus.ejb.all.passivate

위의 모든 설정이 없으면 기본값으로 설정된다. (기본값: 300000(5분), 단위: ms)

• <disconnect-timeout>

- 지정된 시간 동안 클라이언트의 요청을 받지 못하면 클라이언트와 Bean Instance 사이를 연결하던 connection을 끊을 때 사용된다. 그렇게 되면 connection은 각각의 클라이언트와 인스턴스가 맺고 있던 연결을 끊고 Connection Pool로 반환된다.

따라서 클라이언트는 이 connection으로 더 이상 요청을 할 수 없고, 사용 중이던 Bean Instance는 삭제되거나 Bean Pool을 사용 중이면 Bean Pool로 반환된다.

- 이 설정은 다음과 같이 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.

1. 특정 Session Bean에만 적용 : jeus-ejb-dd.xml의 <disconnect-timeout>
2. 모든 Entity Bean에 적용 : 시스템 프로퍼티 jeus.ejb.stateful.disconnect
3. 모든 EJB Bean에(Entity Bean과 Session Bean 모두) 적용 : 시스템 프로퍼티 jeus.ejb.all.disconnect

위의 모든 설정이 없으면 시스템 프로퍼티 jeus.ejb.all.disconnect의 기본값인 3600000(1시간)으로 설정된다. (단위: ms)



<passivation-timeout>과 <disconnect-timeout>에 사용되는 시간은 Bean Instance에 액세스했던 마지막 시점부터 측정된다. 그러므로 <disconnect-timeout>을 <passivation-timeout>보다 길게 설정해야 한다. 또한 <passivation-timeout>은 EJB 엔진에 설정되는 resolution보다는 커야 한다.

타임아웃 값을 길게 설정하면 오랜 시간(대략 십여 분 이상 또는 타임아웃이 중지된 경우) 동안 메모리 안에 많은 Instance가 활성화된 상태로 머물러 있다. 그러므로 시스템 자원이 낭비된다. 타임아웃 값이 수 초정도로 너무 짧으면 passivation, activation 등의 작업이 자주 발생되므로 성능을 저하시킬 수 있고 disconnect 작업으로 인해 세션의 유실 가능성이 있다.

8.3.1.3. ejbLoad()와 ejbStore() Persistence 최적화 설정

Persistence 메소드 호출의 최적화는 jeus-ejb-dd.xml의 각 Bean DD에 설정되어 있다.

다음은 ejbLoad(), ejbStore() 메소드의 Persistence 최적화 설정된 XML 예제로 <jeus-bean>의 <persistence-optimize> 태그 내에서 구성된다.

ejbLoad()와 ejbStore() Persistence 최적화 설정 : <jeus-ejb-dd.xml>

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

```

<jeus-bean>
    . . .
    <persistence-optimize>
        <engine-type>SINGLE_OBJECT</engine-type>
        <non-modifying-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-modifying-method>
        <non-modifying-method>
            <method-name>myBusinessmethod2</method-name>
            <method-params>
                <method-param>
                    java.lang.String
                </method-param>
                <method-param>int</method-param>
            </method-params>
        </non-modifying-method>
        <entity-cache-size>500</entity-cache-size>
        <update-delay-till-tx>
            false
        </update-delay-till-tx>
        <include-update>
            true
        </include-update>
    </persistence-optimize>
</jeus-bean>
    . . .
</beanlist>
    . . .
</jeus-ejb-dd>

```

다음은 설정 태그에 대한 설명이다.

태그	설명
<engine-type>	<p>다음 중에 하나로 설정되어야 한다. 각 설정에 대한 차이는 주요 기능의 설명을 참고한다.</p> <ul style="list-style-type: none"> ◦ EXCLUSIVE_ACCESS (기본값) ◦ SINGLE_OBJECT ◦ MULTIPLE_OBJECT
<non-modifying-method>	<p>ejbStore()가 호출되어야 하는지에 대한 힌트를 EJB 엔진에 제공한다. 이 태그는 CMP 2.0 Bean에서는 적용되지 않는다.</p>
<entity-cache-size>	<p>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 리소스(주 메모리)는 그 만큼 더 많이 사용된다. (기본값: 2000)</p>

태그	설명
<update-delay-till-tx>	<p>Boolean 값을 가지며 EJB 데이터의 update, insert가 트랜잭션이 commit될 때까지 지연될지를 나타낸다.</p> <p>이 값이 false이면 모든 update, insert 작업은 즉각 실행되므로, 동일한 트랜잭션 내에서는 commit()이 호출되기 전에도 변경 사항을 확인할 수가 있지만 성능에 악영향을 미친다. (기본값: true)</p>
<include-update>	<p><schema-info><jeus-query><include-updates>의 기본값을 지정한다.</p> <p>이 값이 true일 경우 finder 메소드가 호출되는 동안에 생성된 update가 commit되므로, finder 메소드가 실행될 동안 업데이트된 정보를 조회할 수 있다. (기본값: false)</p>

8.3.2. CMP 1.1/2.0

다음은 CMP 1.1/2.0의 설정에 대한 설명이다.

8.3.2.1. ejbLoad()와 ejbFind() CM Persistence 최적화 설정

CMP 1.1과 2.0 Bean에는 ejbLoad() persistence 최적화를 설정할 수 있다. 자세한 내용은 [ejbLoad\(\)와 ejbStore\(\) Persistence 최적화 설정](#)을 참고한다.

다음은 ejbLoad, ejbFind CM Persistence 최적화 설정한 XML의 예로 <jeus-bean>의 <cm-persistence-optimize> 태그 내에 설정된다.

ejbLoad()와 ejbFind() CM Persistence 최적화 설정 : <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>

```

다음은 설정 태그에 대한 설명이다.

태그	설명
<subengine-type>	다음의 값으로 설정한다. <ul style="list-style-type: none"> ◦ ReadLocking (기본값) ◦ WriteLocking ◦ WriteLockingFind
<fetch-size>	ResultSet으로 한 번에 가져올 수 있는 레코드의 수를 결정한다. (기본값: 10)
<init-caching>	Boolean 스위치 값이 활성화되면 DB 테이블의 각 레코드에 대하여 EJB 엔진이 미리 인스턴스화된 EJB Entity Instance를 생성한다. 이 작업은 엔진이 시작될 때 수행된다. 비활성화되어 있으면 EJB Instance들은 home interface의 create(), find-ByXXXX() 또는 비슷한 메소드 호출에 의해서만 생성된다. (기본값: false)

8.3.2.2. DB 스키마 정보 설정

CMP 1.1/2.0에서는 DB 테이블과 컬럼에 대응하는 EJB Instance 필드와의 매핑을 포함하고 있는 DB 스키마 정보를 설정해야 한다.

다음은 DB 스키마 정보를 설정한 XML 예제로 <jeus-bean>의 <schema-info> 태그 내에 구성된다.

DB 스키마 정보 설정 : <jeus-ejb-dd.xml>

```

<jeus-ejb-dd>
  . . .
  <beanlist>
    . . .
    <jeus-bean>
      . . .
      <schema-info>
        <table-name>ACCOUNT</table-name>
        <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>
        <creating-table>
          <use-existing-table/>
        </creating-table>
        <deleting-table>true</deleting-table>
        <prim-key-field>
          <field>id</field>
        </prim-key-field>
        <jeus-query>
          <method>

```

```

        <method-name>findByAddress</method-name>
        <method-params>
            <method-param>java.lang.String</method-param>
        </method-params>
    </method>
    <sql>customer_address=?</sql>
</jeus-query>
<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>
<db-vendor>oracle</db-vendor>
<data-source-name>MYDB</data-source-name>
</schema-info>
    . . .
</jeus-bean>
    . . .
</beanlist>
    . . .
</jeus-ejb-dd>

```

다음은 설정 태그에 대한 설명이다.

- **<table-name>**

- 현재 EJB에 매핑되어야 할 관계형 DB 테이블의 이름이다.
- 만약에 설정되어 있지 않으면 EJB module name, EJB name을 이용해 테이블 이름이 임의로 설정된다.
- 일부 DBMS들이 가지는 테이블 이름의 길이제한 때문에 15자 이내의 테이블 이름만 허용된다.

- **<cm-field>**

- Container Managed Field 매핑은 각각의 컬럼과 필드의 관계를 위해 존재한다.
- 하위에 다음의 태그를 설정한다.

태그	설명
<field>	DB 컬럼에 매핑되어야 하는 EJB 필드의 이름을 지정한다.
<column-name>	주어진 DB 테이블에 존재하는 것과 같은 것이다. 지정되어 있지 않으면 EJB 필드의 이름을 사용한다.

태그	설명
<type>	<p>DB 컬럼에 정의된 데이터 타입(예: VARCHAR(25), NUMERIC 등)을 기준으로 한다. 태그가 지정되어 있지 않으면 디폴트 타입이 사용된다(사용될 타입은 DB에 따라 다를 수 있다). 이 매핑은 Appendix A의 기본 Java 타입과 DB 필드 매핑에서 설명한다.</p> <p>Oracle DBMS의 경우 다음의 값을 다룰 수 있다. 이 타입들은 이미지같이 큰 자료 타입에 적합하다.</p> <ul style="list-style-type: none"> ◦ CLOB(Character Large Object): java.lang.String에 적합하다. ◦ BLOB(Binary Large Object): serializable된 필드에 적합하다.
<exclude-field>	<p>true로 설정하면 <field> 태그로 지정된 필드에 대해 accessor 메소드가 생성되지 않아 클라이언트에서 해당 필드를 접근할 수 없다. 이 옵션은 CMP 2.0 Bean에서만 작동한다. 이 옵션은 이전(migration)을 위한 목적으로 사용된다.</p> <p>(기본값: false)</p>

• <creating-table>

- Bean이 start되는 시점에 이 Bean이 사용할 테이블이 DB에 존재하는지를 검사하여 하위 태그에 따라 기존의 테이블을 사용하거나 새로 생성하거나 에러를 발생한다.
- 검사하는 방법은 <schema-info>/<table-name>과 같은 이름을 가지고, <schema-info>/<cm-field>에 명시한 필드들이 존재하며, relation을 관리하는 Bean이라면 <ejb-relation-map>/<jeus-relationship-role>/<column-map>에 명시한 Foreign Key들도 존재하는지 검사한다.
- 다음의 하위 태그를 설정한다.

태그	설명
<use-existing-table>	DB에 이미 테이블이 있으면 그대로 사용하고 없는 경우에만 테이블을 생성한다.
<force-creating-table>	DB에 테이블이 존재해도 그 테이블을 삭제하고 새로운 테이블을 생성한다.

하위 태그를 지정하지 않을 경우 테이블이 이미 존재하면 에러를 발생하고 발생하고 없는 경우는 테이블을 생성한다.

- 태그가 없는 경우는 MS JVM 파라미터에 "-Djeus.ejb.checktable" 설정에 따라 테이블의 검사 여부를 결정한다.

설정값	설명
true	<p>테이블 검사를 하는데 해당 테이블이 존재하지 않으면 에러가 발생한다.</p> <p>(기본값)</p>
false	Bean이 start하는 시점에 테이블 검사를 하지 않기 때문에 실제 테이블을 사용할 때 에러가 발생한다.

• <deleting-table>

- 스위치가 활성화되면 EJB 모듈이 undeploy될 때 지정된 테이블이 DB에서 제거된다.

- 이 옵션은 실제 상황에서 아주 조심스럽게 사용되어야 하므로 실수를 방지하기 위해 JEUS의 시스템 프로퍼티로 별도의 설정을 해야 동작한다. 즉, JEUSMain.xml의 Command 옵션에 "-Djeus.ejb.enable.configDeleteOption=true"로 설정할 때만 동작한다.

또한 <creating-table> 설정이 없는데 <deleting-table> 설정을 사용하는 경우는 거의 없으므로, <creating-table> 설정이 없을 때에는 <deleting-table> 설정이 실행되지 않는다.

• <prim-key-field>

- 복합 Key를 Primary Key로 사용하는 경우에 선택적으로 사용된다. 즉, ejb-jar.xml의 <prim-key-class>에 복합 Key를 위해 사용자가 구현한 Primary Key 클래스를 지정한 경우에 선택적으로 사용할 수 있다. 기본적으로는 <prim-key-class>에 선언된 모든 public 필드가 Primary Key를 구성하지만, 사용자가 필드를 선택하여 Primary Key를 구성하고 싶은 경우에 이 목록을 설정한다.
- 다음의 element를 혼동하지 않도록 한다.

▪ 복합 Key

Element	설명
ejb-jar.xml/<prim-key-class>	사용자가 구현한 Primary Key 클래스이다. (필수 항목)
ejb-jar.xml/<primkey-field>	선언할 수 없다. <prim-key-class>에 선언한 타입은 여러 필드를 포함하는 타입이므로 하나의 필드 이름과 매핑시킬 수가 없다. (선택 항목)
jeus-ejb-dd.xml/<prim-key-field>	Primary Key 클래스의 public 필드 모두를 Primary Key로 구성하지 않고 필드를 선택하고 싶은 경우 그 목록을 선언한다. (선택 항목)

▪ 단일 Key

Element	설명
ejb-jar.xml/<prim-key-class>	java.lang.String, java.lang.Integer 등 ejb-jar.xml/<primkey-field>의 타입이다. (필수 항목)
ejb-jar.xml/<primkey-field>	Primary Key에 해당하는 Entity Bean의 필드 이름이다. <primkey-field>에 선언한 필드 이름과 <prim-key-class>에 선언한 타입이 Entity Bean 클래스에 선언된 필드 이름, 타입과 매핑되어야 한다. (선택 항목)
jeus-ejb-dd.xml/<prim-key-field>	여러 필드가 있을 경우 선택할 때 필요한 것이기 때문 선언할 필요가 없다. (선택 항목)

• <jeus-query>

- CMP 1.1 Entity Bean의 경우 finder 메소드에 대해서 필요한 SQL 문장을 반드시 명시해야 한다.
- CMP 2.0의 경우에는 ejb-jar.xml에 지정된 EJB QL을 overriding할 수 있다.

<jeus-query>를 설정하면 Query 메소드(findXXX)에서 EJB QL과 JEUS EJB QL 확장을 사용할 수 있다.

이것은 ejb-jar.xml의 <query> 태그와 비슷하다. [JEUS EJB QL Extension](#)에서 자세하게 설명한다. 이 태그는 BEA WebLogic 애플리케이션 서버를 JEUS 4.2로 마이그레이션할 때 용이하게 하는 것이 주요 목적이다.

- SQL들을 정의하기 위해 SQL 문장이 사용될 finder 메소드의 이름과 파라미터 그리고 SQL 문장이 설정되어야 한다. 이 문장은 finder 메소드를 생성할 때 사용된다.

지정된 문장에는 WHERE 절 다음에 나오는 부분만을 포함하고 있어야 한다. 다른 요소는 자동으로 주어진다. 여기에는 "?" 문자를 지정할 수 있고 이는 finder 메소드의 파라미터 값들을 대신하게 된다. ANSI SQL의 WHERE 절 문법을 사용할 수 있다(이 SQL은 WHERE에 포함된다).

- <include-updates>가 true로 설정되면 finder 메소드 호출 동안 변경된 사항이 DB에 Commit되므로 finder 메소드에서 변경된 사항을 확인할 수 있다.

- **<db-vendor>**

- Bean을 위해 사용되는 DB의 벤더를 지정한다.

벤더 이름은 벤더 특정 최적화를 위해 EJB 엔진이 사용하게 된다. 여기서 지정된 값이 실제로 성능에 얼마나 많은 영향을 미칠 수 있는지는 확인된 바 없다. 사용 가능한 벤더의 종류는 Appendix A의 [기본 Java 타입과 DB 필드 매핑](#)을 참고한다. 일반적인 예는 "oracle"을 들 수 있다.

- 이 값은 또한 Java 객체와 벤더가 정한 DB 필드 타입과의 정확한 매핑을 가려내기 위하여 사용된다.

- **<data-source-name>**

- DB와 연결할 때 사용하는 DB Connection Pool의 JNDI 이름을 지정한다. Connection Pool은 일반적으로 도메인에 설정되고, MS JVM에 의해 실행된다.

- **<auto-key-generator>**

- CMP 2.0에서 자동으로 Primary Key를 생성하기 위해 사용하며 하위 태그들이 있다. 하위 태그에 대한 자세한 설명은 [자동 Primary Key 생성](#)을 참고한다.

8.3.3. CMP 2.0

본 절에서는 CMP 2.0 설정에 대해서 설명한다.

8.3.3.1. Relationship Mapping 설정

2개의 Bean 사이에 Relationship Mapping을 설정하는 방법은 다음과 같이 경우에 따라 달라진다.

- [One-to-one/One-to-many Relationship Mapping](#)
- [Many-to-many Relationship Mapping](#)

One-to-one/One-to-many Relationship Mapping

2개의 Bean 간에 One-to-one/One-to-many Relationship을 설정할 때 <ejb-relation-map> 태그 내에 설정해야 한다.

다음은 'employee-manager'라는 관계를 가지는 2개의 Bean 간에 Many-to-one(= One-to-many) 관계를 설정한 예로 **<ejb-relation-map>** 태그에 설정한다.

One-to-one/One-to-many Relationship 설정 : <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>
```

다음은 설정 태그에 대한 설명이다.

- **<relation-name>**

ejb-jar.xml 파일에 정의되어 있다.

- **<jeus-relationship-role>**

2개의 JEUS Relationship의 역할을 Relationship의 각 방향별로 하나씩 설정한다. 각 Relationship의 역할에 필요한 정보는 다음과 같다.

- **<relationship-role-name>**

ejb-jar.xml의 <ejb-relationship-role-name> 태그에 정의되어 있다.

- **<column-map>**

2개의 하위 태그인 <foreign-key-column>과 <target-primary-key-column>을 가진다.

태그	설명
<foreign-key-column>	Relationship에서 Foreign Key의 역할을 하는 컬럼의 이름이다.
<target-primary-key-column>	Foreign Key 컬럼이 매핑되어야 할 테이블의 Primary Key 필드이다. Key가 여러 개의 컬럼으로 구성되어 있으면 여러 개의 컬럼 대 컬럼 매핑을 지정할 수 있다.

위의 XML 예제에서는 employee 테이블의 manager-id라는 Foreign Key 컬럼이 employee-to-manager의 Many-to-one 관계를 맺기 위해 manager 테이블의 Primary Key 필드와 매핑되어 있다.

One-to-one Mapping도 이와 비슷한 방법으로 설정된다. 단, One-to-one 관계에서는 Primary Key 컬럼과 Foreign Key 컬럼이 테이블 어디에 존재해도 상관없으며, One-to-many에서는 반드시 “many” 측의 테이블이

Foreign Key 컬럼을 가지고 있다는 것을 유념한다.

Many-to-many Relationship Mapping

2개의 Bean 간에 Many-to-many 관계를 맺기 위해서는 One-to-one의 관계에서 설정된 정보 외에 추가로 <table-name>과 <jeus-relationship-role>의 정보가 반드시 설정되어야 한다.

다음은 Many-to-many 관계 설정을 나타내는 'student-course'의 예이다.

Many-to-many Relationship Mapping 설정 : <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>
    </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>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<table-name>	Many-to-many 관계의 Join 테이블 이름을 지정한다.
<jeus-relationship-role>	2개의 <jeus-relationship-role>에서 각 Join 테이블 2개의 Foreign Key 컬럼을 Bean이 사용하는 테이블의 Primary Key 컬럼에 매핑한다.

위의 예에서 두 JEUS Relationship Role이 어떻게 사용되고 있는지 확인할 수 있다. 각 role은 Join 테이블의 두 컬럼 중 한 개와 실제 Bean이 사용하는 테이블의 Primary Key 컬럼이 매핑하도록 선언하고 있다.

8.3.3.2. Instant EJB QL의 설정

클라이언트에 Instant EJB QL 질의를 가능하게 하려면, JEUS EJB DD의 **<enable-instant-ql>** 태그를 true로 설정한다. 자세한 내용은 Appendix B의 [Instant EJB QL API Reference](#)를 참고한다.

Instant EJB QL 설정 : <jeus-ejb-dd.xml>

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

8.3.3.3. JEUS EJB QL Extension

본 절에서는 JEUS CMP에 관련된 EJB QL Extension에 대해 설명한다.

JEUS CMP에 관련된 EJB QL Extension은 다음과 같다.

- JEUS EJB QL Extension 키워드
- JEUS EJB QL Extension Subquery
- JEUS EJB QL Extension ResultSet
- JEUS EJB QL Extension Dynamic Query
- JEUS EJB QL Extension GROUP BY 키워드

CMP 2.0 Bean의 findByXXX() 메소드와 ejbSelectXXX() 메소드에서는 그 home interface와 ejb-jar.xml의 EJB QL 문장이 연계되어 있어야 한다. JEUS는 표준 EJB QL 언어에 몇 가지 사항들을 추가한다. 이 추가 사항들은 ejb-jar.xml 파일이나 EJB QL에서 사용된다. 이 기능이 사용된 ejb-jar.xml의 예는 [완전한 CMP 2.0 Entity Bean 예제](#)를 참고한다.



본 안내서에서 제공된 EJB QL Extension들은 표준 EJB 2.1 QL에서 지원하지 않는 항목들로서 이것을 이용하여 작성된 EJB 애플리케이션은 Jakarta EE 표준에 준하지 않으므로 이식성이 떨어진다. 이 의미는 여기서 제공된 확장 기능을 사용하여 생성한 EJB 컴포넌트는 다른 Jakarta EE 서버에 deploy될 수 없다는 것이다. JEUS의 EJB QL Extension은 대부분 표준 SQL처럼 작동한다. 그러므로 더 상세한 정보는 SQL을 참고한다.

JEUS EJB QL Extension 키워드 추가사항

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

- **ORACLEHINT** 키워드

"oraclehint<표준 *Oracle hint string*>"의 문법을 가진다.

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

이 키워드에 대한 상세한 정보는 Oracle의 문서를 참고한다.

JEUS EJB QL Extension Subquery

JEUS의 EJB QL Subquery는 EJB QL 질의의 하나로 WHERE 절에 포함된 또 다른 질의절을 의미한다. 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)
```

- 리턴 타입

JEUS EJB QL 질의에 대해 다음과 같은 리턴 타입들이 올 수 있다.

- Single value 또는 <cmp-field>에 대응하는 Collection(위의 예에서 e.salary와 같은 값들)
- 집합 함수를 통해서 생성된 한 개의 리턴값(예: MAX(e.salary))
- 간단한 Primary Key(Compound Primary Key가 아닌)를 포함한 cmp-Bean(예: SELECT OBJECT(emp))

- 비교 연산자

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

다음은 매니저가 아닌 직원을 추출하는 예이다.

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

```
FROM ManagerBean AS employee2)
```

다음은 Subquery 결과값이 없는지 테스트하는 **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 반영되어야 하기 때문이다. 의존 관계가 없는 질의를 비상호 연관(uncorrelated) 질의라고 한다.

다음의 예는 **UNIQUE** 연산자의 사용을 보여주고 있다. 이 연산자는 result set의 row 중 중복된 것이 있는지를 확인한다. 0개 또는 1개의 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 경우에는 평균 월급). ANY, ALL, SOME 연산자는 하나의 값보다 더 많은 값을 리턴하는 Subquery에 사용한다. 이 연산 작업은 다음과 같이 해석할 수 있다.

- <Operand> <arithmetic operator> **ANY** <subquery>

subquery의 결과 중 적어도 한 개의 값이 true를 리턴하면 true를 리턴한다.

- <Operand> <arithmetic operator> **SOME** <subquery>

"ANY"와 같은 의미를 가진다.

- <Operand> <arithmetic operator> **ALL** <subquery>

Subquery의 결과 모두가 true를 리턴해야 true를 리턴한다.



XML DD 파일에 ">"와 "<" 문자를 넣으려 할 때는 CDATA 부분에 포함시켜야 한다. 그렇지 않으면 XML Parser가 혼동을 일으키고 XML 태그 지정문자로 인식한다.

XML DD 파일에 ">"와 "<" 문자 삽입 : <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>
...
```

JEUS EJB QL Extension ResultSet

JEUS EJB 엔진은 복수 컬럼인 java.sql.ResultSet 객체를 반환하는 ejbSelect() 질의를 지원한다. 그러므로 ejbSelect 메소드의 SELECT 문 안에 콤마(,)로 구분된 Target 필드의 목록을 지정할 수 있다.

```
SELECT employee.name, employee.id, employee.salary
FROM EmployeeBean AS employee
```

위의 질의는 모든 EmployeeBean Instance의 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 Bean Instance를 표현한다. ResultSet들은 <cmp-field> 값(Bea인나 relationship filed가 아닌)만을 반환한다.



ejbSelect() 메소드를 사용해서 java.sql.ResultSet 종류의 객체를 얻어와 모든 작업을 마쳤을 때에는 ResultSet.close()를 호출해야 한다. 이 close() 메소드는 EJB 엔진에 의해 자동적으로 호출되지 않는다.

JEUS EJB QL Extension Dynamic Query

JEUS EJB QL은 EJB QL 질의를 프로그램적으로 구현할 수 있도록 지원한다. 이것은 Instant EJB QL 기능으로 가능한 것이다. 본 절에서는 JEUS CMP 2.0에서 Instant EJB QL을 프로그래밍할 때 필요한 기본적인 사항들을 설명한다.

다음 코드에서는 CMP 2.0 Home Interface의 Instant EJB QL 사용법을 보여주고 있다. 이 코드는 순수하게

클라이언트 측의 것이다. 이 코드가 작동되도록 하려면 Instant EJB QL을 사용하도록 CMP Bean에 설정되어 있어야 한다. 이 부분에 대해서는 [Entity EJB 설정](#)을 참고한다.

```
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 Appendix B의 [Instant EJB QL API Reference](#)를 참고한다.

JEUS EJB QL Extension GROUP BY 키워드

JEUS EJB QL은 **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 갖는 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 employee Bean의 유일한 모든 departmentName의 목록을 결과값으로 가진다.

- **<SELECT statement> GROUP BY <grouping column(s)> HAVING <condition>**

이런 형식의 문장은 이전에 설명한 예제와 같은 방식으로 동작하지만, <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 2개 이상의 EmployeeBean Instance가 발견된 모든 유일한 departmentName들의 목록을 반환한다(즉, 3 이상의 employee들을 가진 department의 목록).

예를 들어 **[Table 1]**의 데이터에 EJB QL 질의를 실행시키면 **[Table 2]**와 같은 결과가 출력된다.

• [Table 1]

id	departmentName
johnsmith	R&D
peterwright	R&D
lydiajobs	R&D
catherinepeters	Marketing

• [Table 2]

departmentName
R&D

이 결과는 HAVING 조건이 적어도 3개의 레코드를 가진 컬럼 그룹을 포함시켜야 한다고 명시되었기 때문이다. 더 자세한 GROUP BY, GROUP BY HAVING 절에 대한 정보는 SQL 문서들을 참고한다.



JEUS EJB QL에서는 GROUP BY와 GROUP BY HAVING 문장을 Subquery 또는 최상위의 절의에서 사용할 수 있다. 후자의 경우에는 java.lang.String 객체의 집합이 리턴된다. 최상위의 GROUP BY와 GROUP BY HAVING 문장은 Bean 객체를 리턴할 수 없다.

8.3.4. DB Insert Delay 설정(CMP Only)

CMP에서 EJB가 생성될 때 EJB 데이터가 Backend DB에 기록되는 시점을 설정할 수 있다.

메소드	설명
ejbCreate	ejbCreate()를 실행 후 ejbPostCreate()를 실행하기 전에 새로운 EJB 데이터를 삽입한다.
ejbPostCreate	ejbCreate()와 ejbPostCreate()를 완성한 후에 새로운 EJB 데이터를 삽입한다. 기본값으로 사용된다.

DB Insert Delay 설정은 jeus-ejb-dd.xml에서 <jeus-bean>의 <database-insert-delay> 태그 내에서 구성된다.

DB Insert Delay 설정 : <jeus-ejb-dd.xml>

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

8.4. Entity EJB 튜닝

본 절에서는 JEUS의 Entity EJB 성능 튜닝 방법에 대해서 자세히 설명한다. 튜닝 방법들은 [EJB의 공통 특성과 Session Bean](#)의 "object management"에서 제공한 방법들과 함께 Entity Bean에 적용할 수 있다.

8.4.1. 공통

모든 Entity Bean의 성능 튜닝에 공통적으로 필요한 설정은 다음과 같다.

- 엔진 메인 모드 선택과 클러스터링 적용 여부
- Entity Cache 크기 설정
- DB 벤더 설정

다음은 모든 Entity Bean의 성능 튜닝을 위한 설명이다.

엔진 메인 모드 선택과 클러스터링 적용 여부

다음은 <engine-type>을 선택할 때와 몇 개의 EJB 엔진에 EJB를 클러스터링할지를 선택할 때 주요 판단 기준이 될 수 있는 규칙을 정리한 표이다.

Bean당 요청 수	많은 EJB 엔진 다수의 CPU	적은 EJB 엔진 1개 이상의 CPU	EJB 엔진 1개 CPU 1개
자주 사용	EJB를 클러스터링하고, SINGLE_OBJECT를 사용한다.	EJB를 클러스터링하고, MULTIPLE_OBJECT를 사용한다.	더 많은 EJB 엔진을 사용하도록 한다. 그렇지 않으면, 클러스터링하지 않고, MULTIPLE_OBJECT를 사용한다.
보통으로 사용	EJB를 클러스터링하고, SINGLE_OBJECT를 사용한다.	EJB를 클러스터링하고, MULTIPLE_OBJECT를 사용한다.	EJB를 클러스터링하지 않고, EXCLUSIVE_ACCESS를 사용한다.
조금 사용	EJB를 클러스터링하지 않고, EXCLUSIVE_ACCESS를 사용한다.	EJB를 클러스터링하지 않고, EXCLUSIVE_ACCESS를 사용한다.	EJB를 클러스터링하지 않고, EXCLUSIVE_ACCESS를 사용한다.

물론 Entity Bean의 DB 데이터가 외부의 WAS가 아닌 컴포넌트에 의해 변경되는 환경에서는 SINGLE_OBJECT나 MULTIPLE_OBJECT를 사용해야 한다. 그리고 Failover를 위해서는 Bean을 클러스터링해야 한다.

SINGLE_OBJECT와 MULTIPLE_OBJECT 모드 중 하나를 선택할 때 마지막으로 고려해야 할 사항은 대상 Bean의 트랜잭션 처리시간이다(즉, Bean 내부에서 트랜잭션이 시작하고 commit되어 끝날 때까지의 평균시간). 그 시간이 아주 길면 MULTIPLE_OBJECT 모드를 선택하기를 권장하고, 이 모드에서 사용 가능한 동시성을 사용해야 한다. 그 시간이 반대로 짧으면 SINGLE_OBJECT 모드 사용을 권장한다.

만약에 위의 "규칙"이 준수되면 EJB 엔진은 최적의 성능으로 ejbLoad()를 호출할 수 있다.



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.

Entity Cache 크기 설정

Entity Cache는 passivate된 Entity Bean Instance들의 저장소 역할을 한다. Instance들은 Entity Bean이 다시 Activate가 되면 Cache에서 나오게 된다.

<entity-cache-size> 크기를 크게(수백 개의 Instance가 수용되도록) 설정하면 passivate된 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).

DB 벤더 설정

<db-vendor> 태그를 올바르게 설정한다. 모든 경우는 아니지만 어떤 상황에서는 Oracle과 같은 벤더의 제품을 위해서 EJB 엔진이 DB와의 연결을 최적화시킨다. 이 태그에 사용할 수 있는 DBMS 벤더의 이름 값들은 "JEUS XML Reference"의 "제11장.domain.xml EJB 엔진 설정"을 참고한다.

8.4.2. BMP & CMP 1.1

다음은 BMP와 CMP 1.1 튜닝에 대한 설명이다.

Non-modifying 메소드 등록

DB에 표현된 상태 정보를 변경하지 않는 Entity Bean의 모든 업무 메소드들은 JEUS EJB 모듈 DD의 **<non-modifying-method>** 태그에 등록되어야 한다. 이렇게 하면 EJB 엔진이 ejbStore()를 호출하지 않아 효율적으로 운영될 수 있다. 이 튜닝은 BMP와 CMP 1.1 Bean에게만 적용할 수 있다.

8.4.3. CMP 1.1/2.0

다음은 CMP 1.1/2.0 튜닝에 대한 설명이다.

엔진 Sub-mode 선택

CMP Bean을 사용할 때 **<subengine-type>**을 선택하면 EJB 엔진이 ejbLoad(), ejbFind() 메소드를 최적화해서 사용할 수 있다.

<subengine-type>을 선택하는 주요 기준은 다음과 같다.

- CMP Bean이 읽기 작업을 쓰기 작업보다 더 많이 할 것이라 예상되면 ReadLocking 모드를 사용한다.

- CMP Bean이 쓰기 작업을 읽기 작업보다 더 많이 할 것이라 예상되면 WriteLocking 또는 WriteLockingFind 모드를 사용한다.

Fetch Size 설정

시스템 메모리의 과용을 감수하더라도 높은 성능을 원한다면 CMP Bean의 **<fetch-size>** 태그를 시스템 메모리의 과용을 감수하더라도 높게 설정해야 한다. 큰 값을 설정하면 매 호출마다 많은 양의 데이터를 읽어오기 때문에 네트워크를 효과적으로 사용할 수 있다. 그러나 EJB 엔진은 많은 양의 데이터를 Caching해야 하기 때문에 시스템 자원을 많이 소모하게 되어 각 호출에 대한 대기시간이 증가한다.

반대의 경우 만약 시스템 메모리의 절약이 높은 우선순위를 가지고 느린 네트워크 속도를 감수할 수 있다면 이 값을 상대적으로 낮게 설정한다. 기본값은 "10"이다. 대부분의 경우 "100"은 높은 값이고 "10"보다 낮은 값은 낮은 값으로 생각할 수 있다.

Initial Caching 설정

높은 운영 성능을 위해 많은 시스템 메모리 사용을 감수할 수 있다면, CMP Bean의 **<init-caching>** 값을 true로 설정한다. 이렇게 설정하면 EJB 엔진이 기동(Booting)할 때 모든 DB의 레코드들이 읽혀져 Entity Bean Instance로 전달된다. 결과적으로 엔진 기동(Booting) 시간이 오래 걸리겠지만, 처음의 접근시간을 훨씬 줄일 수 있다. 반대의 경우 만약 시스템 메모리의 절약 또는 빠른 엔진 기동(Booting)시간이 우선이라면 이 값은 false로 설정해야 한다. 만약 DB 테이블이 EJB에 아주 많이 매핑되어 있는 경우(수백, 수천 레코드)에는 **<init-caching>**가 false로 설정되어야 할 것이다. 그러나 EJB가 read-only(ejbLoad())가 한 번만 호출되는 것을 암시)라면 이 옵션을 강력히 추천한다.

8.4.4. CMP 2.0

CMP 2.0 튜닝을 위해서 EJB QL 사용하는데 EJB QL은 효율적이지 않으므로 간단한 경우에만 사용하도록 한다.

8.5. 완전한 CMP 2.0 Entity Bean 예제

EJB 코드, 표준 EJB DD, JEUS DD가 함께 설정되고 작동될 수 있는지 보여주기 위해 본 절에서는 완전한 CMP 2.0 Bean의 예를 제시한다. 이 예제는 책을 개념으로 모델화한 Book EJB이다. 기본적으로 모든 예에 주석을 달지 않았고 CMP 2.0 Bean이 jeus-ejb-dd.xml에 설정하는 방법만 보여준다. 예제를 deploy하려면 패키징 후에 EJB로 deploy한다. 자세한 내용은 [EJB 모듈](#)을 참고한다. JEUS의 ejb-jar.xml에서 EJB QL 사용 예는 본 절의 [Jakarta EE EJB DD](#)를 참고한다.

Remote Interface

Remote Interface : <Book.java>

```
package test.book;

import java.rmi.RemoteException;
import jakarta.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;
}

```

Home Interface

Home Interface : <BookHome.java>

```

package test.book;

import java.rmi.RemoteException;
import jakarta.ejb.CreateException;
import jakarta.ejb.FinderException;
import jakarta.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;
}

```

Bean Implementation

Bean Implementation : <BookEJB.java>

```

package test.book;

import jakarta.ejb.EntityBean;
import jakarta.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() {}
}

```

Jakarta EE EJB DD

Jakarta EE EJB DD : <ejb-jar.xml>

```

<?xml version="1.0"?>
<ejb-jar version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/ ejb-jar_4_0.xsd">
    <enterprise-beans>
        <entity>
            <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>
            <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>
<primkey-field>code</primkey-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>
</entity>
</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>

```



굵은 글자의 3줄이 비표준의 JEUS 전용 EJB QL이다("ORDERBY", "BETWEEN",

"ORACLEHINT").

JEUS EJB DD

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-name>book</export-name>
      <export-port>0</export-port>
      <export-iiop><only-iiop>false</only-iiop></export-iiop>
      <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>
      <schema-info>
        <table-name>Booktable</table-name>
        <cm-field><field>code</field></cm-field>
        <cm-field><field>title</field></cm-field>
        <cm-field><field>author</field></cm-field>
        <cm-field><field>price</field></cm-field>
        <cm-field><field>publisher</field></cm-field>
        <creating-table>
          <use-existing-table/>
        </creating-table>
        <deleting-table>true</deleting-table>
        <db-vendor>oracle</db-vendor>
        <data-source-name>datasource1</data-source-name>
      </schema-info>
      <enable-instant-ql>true</enable-instant-ql>
    </jeus-bean>
  </beanlist>
</jeus-ejb-dd>
```

9. Message Driven Bean(MDB)

본 장에서는 EJB 엔진에서 Message Driven Bean(MDB)를 사용할 때 유의해야 할 사항들을 설명한다.

9.1. 개요

EJB 엔진은 Message Driven Bean(이하 MDB) Instance들을 동시에 여러 개 실행시킬 수 있으며, 또한 message stream을 처리할 수도 있다. EJB 엔진은 MDB 클래스에 도착하는 메시지들의 순서를 보장하지 않으므로 받은 순서대로 메시지를 전달하지 못하고 임의로 메시지를 전달한다.

MDB도 처리 순서를 보장하지 않는다. 예를 들어 예약을 하는 메시지가 도착하기 전에 예약을 취소하는 메시지가 먼저 올 수 있기 때문에 설계할 때 처리 순서에 대해 고려해야 한다.

이런 동시 처리는 Bean Pool이 MDB에서 Instance를 가져와서 Request에 할당함으로써 처리된다. 이는 Stateless Session Bean의 Thread Ticket Pool(TTP)의 동작법과 거의 동일하다. 따라서 MDB에서는 <thread-max> 값이 아닌 <pool-max> 값을 사용해야 한다. 이에 대해서는 [Session Bean](#)을 참조한다.

9.2. MDB 설정

EJB 엔진이 MDB를 실행하기 위해서는 약간의 설정이 필요하다. MDB는 고유의 설정 외에 JEUS의 다른 EJB와 공통되는 설정도 가지고 있다.

본 절에서는 다음 항목을 간략하게 설명한다.

- JEUS EJB DD의 기본 MDB 설정([EJB의 공통 특성](#)에서 설명한 설정들의 하위 항목)
- JEUS EJB DD 파일에 설정되어 있는 JEUS MDB의 JMS 전용 설정



MDB 설정은 MDB를 잘 알고 있는 이들에게는 별도로 설명할 필요가 없다.

9.2.1. 기본 환경설정

MDB는 다른 EJB 종류와 유사한 기본 설정을 사용한다. 각 MDB에 설정할 수 있는 것들은 <ejb-name>, <run-as identity>, <security-interop> 등이 있다.

다음은 MDB의 기본 설정 예제이다.

기본 환경설정 : <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
  . . .
  <beanlist>
    . . .
    <jeus-bean>
      <ejb-name>order</ejb-name>
      <!--JMS settings goes here (see "9.2.2. JMS 설정")-->
    </jeus-bean>
  </beanlist>
</jeus-ejb-dd>
```

```

        <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>

```

9.2.2. JMS 설정

MDB가 처리하는 메시지에 따라 성격이 달라진다. JMS 메시지를 처리하는 경우에는 JMS Connection Factory의 export name이 필요하고 Connector 메시지를 처리하는 경우에는 리소스 어댑터가 필요하다.

다음은 MDB 클래스의 Annotation으로 설정한 것과 그에 대응되는 XML 문서의 일부이다.

JMS 설정 : <MyMDB.class>

```

@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(propertyName="destinationType",
                                   propertyValue="jakarta.jms.Queue"),
        @ActivationConfigProperty(propertyName="acknowledgeMode",
                                   propertyValue="Auto-acknowledge")
    },
    mappedName = "jms/QUEUE1"
)
public class MyMDB implements MessageListener {
    ...
}

```

JMS 설정 : <jeus-ejb-dd.xml>

```

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

```

```

<jbus-bean>
    . . .
    <ejb-name>MyMDB</ejb-name>
    <connection-factory-name>
        QueueConnectionFactory
    </connection-factory-name>
    <destination>jms/QUEUE1</destination>
    <max-message>15</max-message>
    <activation-config>
        <activation-config-property>
            <activation-config-property-name>
                destinationType
            </activation-config-property-name>
            <activation-config-property-value>
                jakarta.jms.Queue
            </activation-config-property-value>
        </activation-config-property>
        <activation-config-property>
            <activation-config-property-name>
                acknowledgeMode
            </activation-config-property-name>
            <activation-config-property-value>
                Auto-acknowledge
            </activation-config-property-value>
        </activation-config-property>
    </activation-config>
    . . .
</jbus-bean>
    . . .
</beanlist>
    . . .
</jbus-ebb-dd>

```

다음은 설정 태그에 대한 설명이다.

- **<connection-factory-name>**

- JMS Connection Factory가 사용할 JNDI export name을 설정한다.
- 이 설정은 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.
 1. jeus-ebb-dd.xml/<jbus-bean>/<connection-factory-name>
 2. jeus-ebb-dd.xml/<jbus-bean>/<activation-config> property name : jeus.connectionFactoryName
 3. @ActivationConfigProperty property name : jeus.connectionFactoryName

- **<mdb-resource-adapter>**

- JEUS Connector를 통해 메시징 시스템과 연결될 경우 사용하는 리소스 어댑터를 설정한다. 해당 리소스 어댑터의 jeus-connector-dd.xml에 지정되어 있는 모듈의 ID를 설정하여 사용한다.

- **<destination>**

- JMS Destination의 JNDI 이름을 설정한다.
- 이 설정은 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.
 1. jeus-ebb-dd.xml/<jbus-bean>/<destination>

2. ejb-jar.xml의 <message-driven>/<mapped-name>
3. @MessageDriven의 mappedName
4. ejb-jar.xml의 <message-driven>/<message-destination-link>

- **<jndi-spi>**

- MDB가 기본값(jeus.jndi.JNSContextFactory)이 아닌 다른 JNDI 이름 서비스에 등록되어 있는 JMS 서비스를 사용할 때 즉, JEUS MDB를 IBM MQ나 SONIC MQ같은 JEUS JMS 서비스 이외의 것과 연결할 때 사용한다.

- **<max-messages>**

- 여러 개의 JMS 세션을 등록해서 처리할 경우 한 세션에 메시지를 할당하는 최대 개수를 지정할 때 사용한다.
- 큐에 쌓인 메시지 수가 설정된 값보다 적으면 하나의 세션이 계속 처리하며 쌓인 메시지가 이 값을 넘어가면 그때 다른 세션을 사용한다. 즉, 이 값이 10일 경우 메시지 수가 10개를 넘을 때까지는 하나의 세션만이 메시지를 처리한다. 이는 load를 줄이기 위한 것이며 자세한 것은 JMS 스펙을 참고한다.

- **<activation-config>**

- JMS나 리소스 어댑터를 설정할 activation config로 ejb-jar.xml의 activation config를 override하는 경우에 사용한다.
- JMS MDB의 경우에는 acknowledgeMode, messageSelector, destinationType, subscriptionDurability의 기본적으로 인식된다. 추가적으로 위에서 말한 jeus.connectionFactoryName과 jeus.clientId, jeus.subscriptionName을 사용할 수 있다. jeus.clientId와 jeus.subscriptionName은 Topic의 Durable Subscription을 위해 제공되고 만약 이 값들이 설정되지 않으면 "모듈 이름"/"Bean 이름"으로 설정된다.
- 이 설정은 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.
 1. jeus-ejb-dd.xml의 <activation-config>
 2. ejb-jar.xml의 <activation-config>
 3. @ActivationConfig

- **<ack-mode>**

- JMS 세션의 Acknowledge 모드를 설정한다.
- 이 설정은 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.
 1. jeus-ejb-dd.xml/<jeus-bean>/<ack-mode>
 2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : acknowledgeMode
 3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : acknowledgeMode
 4. ejb-jar.xml/<message-driven-bean>/<acknowledge-mode>
 5. @ActivationConfigProperty property name : acknowledgeMode

- **<durable>**

- JMS의 Durable Subscriber를 설정한다.
- 이 설정은 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.
 1. jeus-ejb-dd.xml/<jeus-bean>/<durable>
 2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : subscriptionDurability

3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : subscriptionDurability
4. ejb-jar.xml/<message-driven-bean>/<subscription-durability>
5. @ActivationConfigProperty property name : subscriptionDurability

- **<msg-selector>**

- JMS의 message selector를 설정한다.
- 이 설정은 여러 곳에서 설정 가능하고 우선순위는 다음과 같다.
 1. jeus-ejb-dd.xml/<jeus-bean>/<msg-selector>
 2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : messageSelector
 3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : messageSelector
 4. ejb-jar.xml/<message-driven-bean>/<message-selector>
 5. @ActivationConfigProperty property name : messageSelector



위의 설정에 대한 자세한 내용은 "JEUS MQ 안내서", "JEUS JCA 안내서", JMS 표준, EJB 표준을 참고한다.

9.2.3. JNDI SPI 환경설정

기본적으로 JEUS MDB는 JEUS Naming Service로 JMS의 connection을 Lookup해서 사용한다.

만약 다른 JMS 서비스(IBM MQ나 SONIC MQ)를 이용하는 경우 다른 서비스에서 connection을 얻어야 한다. 이런 경우는 JMS 서비스를 포함한 외부 Naming Service를 사용하는 MDB를 설정할 수 있다. MDB가 JMS 서비스를 찾기 위해 각각의 MDB에 **<jndi-spi>**를 설정한다.

다음은 JNDI SPI 환경설정에 대한 예제이다.

JNDI SPI 환경설정: <jeus-ejb-dd.xml>

```
<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>
```

다음은 <jndi-spi> 하위의 설정 태그에 대한 설명이다.

태그	설명
<mq-vendor>	MDB가 connection을 맺을 MQ/JMS의 벤더의 이름이다(JNDI naming 서비스를 통해 설정). 'IBMMQ'와 'SONICMQ'가 공식적인 지원 벤더이다.
<initial-context-factory>	Naming Server를 위한 JNDI InitialContext Factory 클래스의 이름을 사용해서 JMS 서비스에 연결된다.
<provider-url>	요청에 의해서 Naming Service에 접속할 때 사용하는 URL이다.

10. EJB Timer Service

EJB Timer Service는 EJB가 특정한 시간 또는 주기적으로 callback을 받을 수 있도록 하는 서비스이다. 기본적인 사용 방법은 EJB 스펙에 설명되어 있으므로 본 장에서는 JEUS EJB에서 제공하는 Timer Service와 이를 사용하기 위한 설정에 대해서 설명한다.

10.1. Timer Service 설정

JEUS EJB Timer Service는 기본적으로 스펙을 따르지만 persistence하게 Timer를 관리하는 기능은 성능과 사용자의 필요에 따라 선택적으로 사용할 수 있다.

Timer Service는 다음 2개의 설정을 한다.

- EJB 엔진의 Timer Service

Timer Service를 사용하는 모든 Bean에 적용되는 공통적인 설정과 persistence한 Timer Service를 가능하게 하는 설정을 한다.

- jeus-ejb-dd.xml

각 Bean이 deploy/undeploy될 때 Persistent Timer들을 어떻게 관리하는지에 대한 설정을 한다.

10.1.1. Persistent Timer Service 설정(EJB 엔진)

domain.xml을 사용하여 다음 예시와 같이 EJB 엔진의 Persistent Timer Service를 설정할 수 있다.

EJB 엔진의 Persistent Timer Service 설정: <domain.xml>

```
<ejb-engine>
  <resolution>1000</resolution>
  <use-dynamic-proxy-for-ejb2>true</use-dynamic-proxy-for-ejb2>
  <enable-user-notify>false</enable-user-notify>
  <timer-service>
    <support-persistence>true</support-persistence>
    <max-retrial-count>1</max-retrial-count>
    <retrial-interval>5000</retrial-interval>
    <thread-pool>
      <min>2</min>
      <max>30</max>
      <period>3600000</period>
    </thread-pool>
    <database-setting>
      <data-source-id>tibero_XA1</data-source-id>
      <db-vendor>tibero</db-vendor>
    </database-setting>
  </timer-service>
</ejb-engine>
```

고급 선택사항에서 Persistent Timer Service의 각 설정 항목 대한 설명은 다음과 같다.

• Thread Pool

Timer Service가 timeout() 메소드를 실행할때 사용하는 Thread Pool에 대한 설정이다.

항목	설명
Min	Pooling되는 스레드의 최솟값이다.
Max	Pooling되는 스레드의 최댓값이다.
Period	Pooling되는 스레드를 정리하는 시간이다.

• Database Setting

Persistent Timer Service를 사용한다면 반드시 설정해야 하는 항목이다. Persistent Timer는 DB를 사용하기 위해 내부적으로 CMP Bean을 사용하므로 이 CMP Bean이 아래 항목들을 통해 설정된다. 따라서 각 항목은 CMP Bean의 스키마 설정과 같다.

항목	설명
Db Vendor	Timer CMP Bean이 사용하는 DB의 벤더를 지정한다.
Data Source Id	Timer CMP Bean이 사용하는 데이터소스 리소스의 이름을 지정한다.

10.1.2. Persistent Timer 처리(jeus-ejb-dd.xml)

Persistent Timer Service를 사용하도록 EJB 엔진에 설정되어 있을 때 사용자는 각 EJB Bean별로 Persistent Timer Service를 사용할지, deploy하기 전에 DB에 남아있는 Persistent Timer를 사용할지 제거할지 등의 동작을 설정할 수 있다. 이는 jeus-ejb-dd.xml을 통해서 설정한다. Persistent Timer Service의 설정은 domain.xml에서 가능하다. domain.xml에서의 설정 방법에 대한 자세한 내용은 [Persistent Timer Service 설정\(EJB 엔진\)](#)을 참고한다.

다음은 jeus-ejb-dd.xml에 Persistent Timer 처리를 설정한 예제로, <jeus-bean>의 <timer-service> 태그 내에 설정한다.

Persistent Timer의 처리 : <jeus-ejb-dd.xml>

```
<jeus-ejb-dd>
. . .
<beanlist>
. . .
<jeus-bean>
. . .
<timer-service>
<support-persistence>
true
</support-persistence>
</timer-service>
. . .
</jeus-bean>
. . .
</beanlist>
. . .
</jeus-ejb-dd>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<support-persistence>	해당 Bean의 Timer를 persistent하게 관리할 것인지를 결정한다.

10.1.3. Cluster-Wide Timer Service 설정

클러스터의 경우에는 모든 EJB 엔진이 동일한 EJB Timer Service를 사용해야 하므로 공통된 설정을 해야 한다. 따라서 각 MS에 설정된 값은 무시된다.

다음은 domain.xml을 사용하여 Cluster-Wide Timer Service를 설정하는 과정에 대한 예시이다.

Cluster-Wide Timer Service 설정: <domain.xml>

```
<cluster>
  <cluster-wide-timer-service>
    <database-setting>
      <data-source-id>tibero_XA1</data-source-id>
      <db-vendor>tibero</db-vendor>
    </database-setting>
  </cluster-wide-timer-service>
</cluster>
```

각 설정 항목에 대한 설명은 [Persistent Timer Service 설정\(EJB 엔진\)](#)의 '**Database Setting**' 항목과 동일하다.

10.2. Timer 모니터링

콘솔 툴을 사용하여 동작 중인 EJB Timer에 대한 모니터링 및 동작 취소가 가능하다.

콘솔 툴 사용

Timer의 모니터링 및 동작 취소는 콘솔 툴을 사용할 수도 있다.

• 모니터링

ejb-timer-info 명령어를 사용해서 특정 서버의 EJB Timer 정보를 확인할 수 있다.

```
ejb-timer-info -server <server-name>
```

• 동작 취소

cancel-ejb-timer 명령어를 사용해서 특정 서버의 EJB Timer를 취소할 수 있다.

```
cancel-ejb-timer -server <server-name>
```



위의 2가지 명령어 및 EJB 엔진 관련 명령어에 대한 자세한 내용은 JEUS Reference 안내서의 "EJB 엔진 관련 명령어"를 참고한다.

10.3. Timer Service 사용 주의사항

다음은 Timer Service를 사용하는 경우 고려해야할 주의사항에 대한 설명이다.

- **Persistent Timer와 JDBC Connection**

Persistent Timer는 DB에 저장되고 이 Timer들을 관리하는 CMP Bean은 EJB 엔진의 Time Service - Database Setting - Data Source Id 값을 사용한다.

Persistent Timer를 사용하는 Bean이 포함된 트랜잭션은 Timer CMP Bean이 사용하는 데이터소스도 관리하게 된다. 따라서 이를 고려해서 데이터소스를 LocalXAResource로 사용할지 XAResource로 사용할지를 결정해야 한다.



데이터소스에 대한 자세한 설명은 "JEUS Server 안내서"를 참고한다.

11. EJB 클라이언트

본 장에서는 EJB 클라이언트 프로그래밍 예제와 InitialContext 설정 방법에 대해 설명한다.

11.1. 개요

EJB 클라이언트는 EJB 엔진에서 deploy된 EJB 컴포넌트의 업무 로직을 호출하는 모든 종류의 애플리케이션을 일컫는다. 그러므로 EJB 클라이언트는 서버릿, 애플릿, 다른 EJB, Standalone Java 프로그램 등이 이에 속한다.

여기서는 Standalone Java 프로그램으로 EJB 클라이언트를 작성하고 실행할 때 기본적으로 필요한 정보를 제공한다.



클라이언트 컨테이너 애플리케이션에 대해서는 "JEUS Application Client 안내서"를 참고한다.

11.2. EJB 접근을 위한 클라이언트 프로그래밍

다음은 "HelloApp"라는 export name을 갖는 EJB를 찾고 sayHello() 메소드를 호출하는 일반적인 클라이언트 애플리케이션의 구현을 보여주는 예제이다.

EJB 접근을 위한 클라이언트 프로그래밍 : <HelloClient.java>

```
package hello;
import java.rmi.*;
import jakarta.ejb.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class HelloClient {
    private void run()
    {
        try {
            //JEUS InitialContext 얻기
            InitialContext ctx = new InitialContext();

            //EJB 얻기
            Hello hello = (Hello)ctx.lookup("HelloApp");
            //Business Method 호출
            String s = hello.sayHello();
        }
        catch (Exception e)
        { // Exception handling. }
    }

    public static void main(String args[])
    {
        HelloClient hclient = new HelloClient();

        hclient.run();
    }
}
```



```
}
```

위의 예에서 InitialContext를 얻는 **"InitialContext ctx = new InitialContext();"** 부분이 제대로 동작하기 위해서는 InitialContext의 속성을 설정해야 한다. InitialContext의 설정에 대한 내용은 [InitialContext 설정](#)을 참고한다.

11.3. InitialContext 설정

InitialContext는 JNDI namespace의 모든 명명된 객체를 찾을 때 사용하는 객체로 EJB를 사용할 때마다 필요한 중요 컴포넌트이다.

EJB에서는 이 객체를 사용하여 EJB 엔진의 실제 Bean Instance의 Reference를 찾는다. JEUS의 InitialContext Instance를 가져오기 위해서는 InitialContext를 얻기 전에 5개의 Naming Context 속성 중 적어도 1개는 지정해야 클라이언트 애플리케이션이 JEUS Naming Service Server(JNSServer)와 통신할 수 있다.

JNDI InitialContext 객체가 JVM의 "-D" 속성(권장사항)이나 EJB 클라이언트 코드의 Hashtable를 통해 설정되는 것을 살펴보고 각각의 경우 EJB 클라이언트를 어떻게 호출하는지 설명한다.

다음은 JEUS에서 지원하는 InitialContext 객체를 얻기 전에 설정해야 할 5개의 Naming Context 환경 속성이다.

- **INITIAL_CONTEXT_FACTORY**(required)

Naming Context 속성은 "jeus.jndi.JNSContextFactory"의 값을 가져야 한다. 이 값은 Naming Context를 생성할 때 사용하는 factory의 클래스 이름이다.

- **URL_PKG_PREFIXES**

jeus.jndi.jns.url로 설정되어 JEUS InitialContext의 URL scheme을 이용하여 객체들을 찾는다.

- **PROVIDER_URL**

JEUS Naming server(JNS Server) IP 주소를 설정한다. (기본값: 127.0.0.1 ("localhost"))

- **SECURITY_PRINCIPAL**

JEUS Naming Service의 인증 과정 중에 사용되는 사용자 정보를 설정하기 위해 사용한다.

실행 스레드에 사용자 정보가 등록되어 있지 않으면 기본으로 guest로 인증된다. 만약에 나중에 지정되면 JEUS Security realm에 정의된 username을 갖는 사용자 정보가 사용된다.

- **SECURITY_CREDENTIALS**

인증 과정 중에 사용될 사용자의 패스워드를 설정한다. 인증이 실패하면 그 전의 사용자 정보가 그 전처럼 유지된다. 성공하면 새로운 사용자 정보가 사용된다.



위에서 제시한 기본 속성들 외에 추가적인 특성들을 더 설정할 수 있다. 그 추가 속성들에 대한 내용은 JEUS Server 안내서의 "JNDI Naming Server"를 참고한다.

11.3.1. JVM 속성을 이용한 Naming 속성값 설정

위에서 설명한 속성들은 JVM interpreter의 "-D" 스위치를 사용하여 설정할 수 있다. 이 스위치들은 EJB 클라이언트 애플리케이션을 시작하기 위해 사용하는 Java 명령어에 추가해야 한다.

다음은 각 Naming 속성이 "-D" 스위치로 설정하는 예이다.

- **INITIAL_CONTEXT_FACTORY**(required)

```
-Djava.naming.factory.initial=jeus.jndi.JNSContextFactory
```

- **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>
```

다음의 예제는 클라이언트를 호출할 때 "-D" 옵션을 사용하여 EJB 클라이언트의 Naming Context를 설정하는 것이다. 'user1'이라는 사용자명과 'password1'이라는 패스워드가 192.168.10.10 주소로 등록된 JEUS Naming Server를 이용하여 EJB Stub을 찾는 EJB 클라이언트에 어떻게 사용되는지를 보여준다.

```
$ java -classpath
    ${JEUS_HOME}/lib/client/jclient.jar
    -Djava.naming.factory.initial=jeus.jndi.JNSContextFactory
    -Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
    -Djava.naming.provider.url=192.168.10.10
    -Djava.naming.security.principal=user1
    -Djava.naming.security.credentials=password1
    HelloClient
```



JEUS에 deploy한 EJB가 2.x 스타일인 경우에는 별도로 클라이언트용 라이브러리를 가지고 있어야 한다. 이 라이브러리는 보통 appcompiler로 EJB를 컴파일할 때 생성할 수 있다.

클래스 FTP 서버를 사용한다면 클라이언트가 자동으로 필요한 클래스들을 다운받아서 사용한다. EJB 3.0 이상인 경우에는 이 사항을 고려할 필요가 없다.

11.3.2. Hashtable을 이용한 Naming 속성 설정

Naming Context 속성을 설정하는 다른 방법은 Hashtable를 사용하여 클라이언트 코드 내의 Naming Attribute Data를 직접 Hashtable에 넣는 것이다.

다음의 예제는 이를 어떻게 구현하는지 보여준다.

```
Context
    ctx = null; Hashtable ht = new Hashtable();
    ht.put(Context.INITIAL_CONTEXT_FACTORY, jeus.jndi.JNSContextFactory");
    ht.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
    ht.put(Context.PROVIDER_URL, "192.168.10.10");
    ht.put(Context.SECURITY_PRINCIPAL, "user1");
    ht.put(Context.SECURITY_CREDENTIALS, "password1");
    try { ctx = new InitialContext(ht); . . .
```

위의 예에서 Hashtable이 InitialContext의 constructor 파라미터로 전달되는 것에 주목한다.

다음 예제는 클라이언트 클래스에 Naming 속성을 설정한 경우에 EJB 클라이언트를 실행하는 방법이다.

```
$ java -classpath
    ${JEUS_HOME}/lib/client/jclient.jar HelloClient
```



Hashtable을 이용한 방법은 값들을 hard-coding하기 때문에 권장하지는 않는다.

12. 부가 기능

본 장에서는 JEUS에서 EJB를 개발할 때 사용할 수 있는 부가 기능들에 대해 설명한다.

12.1. WorkArea 서비스

WorkArea 서비스는 프로그램이 어떤 묵시적(implicit)인 컨텍스트를 계속 전달할 때 사용할 수 있는 기능이다. 마치 Security 컨텍스트나 트랜잭션 컨텍스트와 같이 별도의 인자로 전달하지 않아도 로컬 또는 원격 메소드를 호출할 때 계속 전달된다. 어떤 컨텍스트를 전달해야 하는 경우나 메소드의 인자수가 많아지는 경우에 WorkArea를 사용할 수 있다.

WorkArea는 일종의 사용자 저장 공간으로 name-value pair 형태의 맵(Map)으로 저장된다. WorkArea를 새로 시작하면 현재의 스레드와 함께 이동하기 때문에 호출된 메소드나 EJB와 같은 컴포넌트에서 계속 사용할 수가 있다. 또한, 원격 EJB를 호출하는 경우에도 전달(propagation)되는 특성이 있다.

WorkArea 서비스를 이용하기 위해 UserWorkArea 인터페이스인 jeus.workarea.UserWorkArea를 사용한다.



1. 자세한 API 정보는 JEUS API Javadoc 문서의 jeus.workarea 패키지를 참조한다.
2. 현재 원격 WorkArea 전달(propagation) 기능은 EJB 2.0과 3.0 모두 지원한다. JEUS 기본 RMI 호출이 아닌 IIOP를 사용하는 경우에도 전달된다. 호출을 하는 경우에는 전달되지 않는다.

12.1.1. UserWorkArea 인터페이스

UserWorkArea 인터페이스는 UserWorkArea의 시작, 종료 및 각 조작에 필요한 모든 메소드를 정의하며 다음과 같다.

UserWorkArea 인터페이스

```
public interface UserWorkArea {
    public void begin(String name);
    public void complete()
        throws NoWorkAreaException, NotOriginatorException;
    public String getName();
    public String[] retrieveAllKeys();
    public void set(String key, java.io.Serializable value)
        throws NoWorkAreaException, NotOriginatorException,
            PropertyReadOnlyException;
    public void set(String key, java.io.Serializable value, PropertyModeType mode)
        throws NoWorkAreaException, NotOriginatorException,
            PropertyReadOnlyException;
    public java.io.Serializable get(String key);
    public PropertyModeType getMode(String key);
    public void remove(String key)
        throws NoWorkAreaException, NotOriginatorException,
            PropertyReadOnlyException;
}
```



메소드 정의에 대한 자세한 정보는 Javadoc을 참고한다.

12.1.2. PropertyMode 타입

WorkArea에 저장되는 값들은 각각의 PropertyMode를 갖는다.

각 PropertyMode 타입은 다음과 같다.

타입	설명
NORMAL	등록된 값에 대해 수정과 삭제가 모두 가능하다.
READ_ONLY	UserWorkArea에 등록된 값에 대해 수정과 삭제가 모두 불가능하다.

12.1.3. 예외

UserWorkArea에서 정의된 예외는 다음과 같다.

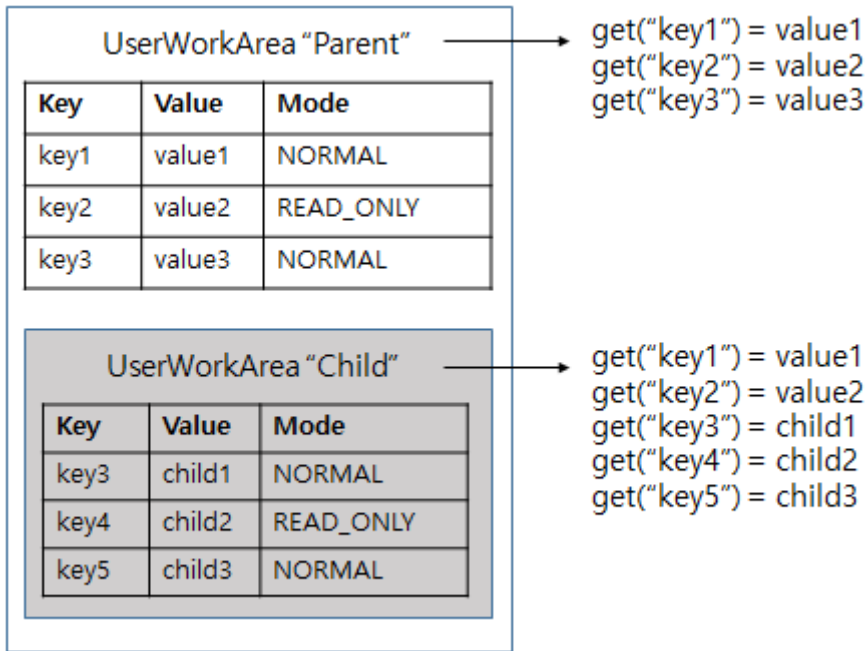
예외	설명
WorkAreaException	WorkArea에서 발생하는 Exception의 최상위 Exception이다.
NotOriginatorException	작업 영역을 시작한 스레드가 아닐 경우 값을 수정하거나 삭제 또는 작업 영역을 닫으려고 할 경우 발생한다.
PropertyReadOnlyException	PropertyMode가 READ_ONLY로 설정된 값을 변경하는 경우 발생한다.

12.1.4. Nested UserWorkArea

UserWorkArea는 중첩되어 사용될 수 있다. 이미 UserWorkArea가 존재하는 상황에서 새로운 UserWorkArea를 시작하면 그 UserWorkArea는 기존 UserWorkArea에 내포되게 된다.

내포된 UserWorkArea는 상위 UserWorkArea에 저장된 값들을 그대로 이용할 수 있으며 새로운 값들을 추가할 수 있다. 내포된 UserWorkArea에서 추가된 값들은 단지 그 UserWorkArea에서만 값을 갖게 되며 내포된 UserWorkArea가 완료되면 사라진다.

중첩된 UserWorkArea에서의 등록 정보는 다음 그림과 같다.



중첩된 UserWorkArea에서의 등록 정보

12.1.5. UserWorkArea를 사용하는 응용 프로그램 개발

본 절에서는 실제로 UserWorkArea를 사용하는 EJB 예제를 작성하며 UserWorkArea 인터페이스를 사용하는 방법을 설명한다.

EJB 예제는 UserWorkAreaSampleSender와 UserWorkAreaSampleReceiver 2개의 EJB로 이루어져 있다. Sender에서 UserWorkArea를 생성하여 데이터를 전달하고, Receiver에서 UserWorkArea에 설정된 값을 이용하여 메시지를 생성하여 원래 Sender에서 리턴하는 예제로 과정은 다음과 같다.

1. UserWorkArea 액세스

UserWorkArea를 사용하기 위해서는 먼저 JNDI에서 UserWorkArea를 lookup해야 한다.

다음은 JNDI에서 UserWorkArea를 lookup하는 방법에 대한 예이다.

UserWorkArea 액세스 : <UserWorkAreaSampleSenderBean.java>

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            //UserWorkArea를 JNDI에서 가져온다.
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
        } catch (NamingException e) {
            // Do Something...
        }
        return message;
    }
}
```

2. 새 UserWorkArea 시작

JNDI에서 처음 lookup해온 UserWorkArea는 아무런 정보가 없기 때문에 새로운 UserWorkArea를 시작해야 한다. 만약, UserWorkArea의 이름이 null이면 NullPointerException이 발생한다.

새 UserWorkArea 시작 : <UserWorkAreaSampleSenderBean.java>

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            // 새로운 UserWorkArea를 시작한다.
            userWorkArea.begin("UserWorkArea1");
        } catch (NamingException e) {
            // Do Something...
        }
        return message;
    }
}
```

3. WorkArea에 등록 정보 설정

새로 시작한 UserWorkArea에 등록 정보를 설정한다. 등록 정보는 <key, value, mode>로 이루어진다. 'key'는 String이며 'value'는 직렬화(Serialization)가 가능한 객체이다.

값을 설정할 때 현재 UserWorkArea가 없으면 NoWorkAreaException이 발생한다. 시작한 UserWorkArea가 아니면 NotOriginatorException이 발생하며, 이미 READ_ONLY로 설정된 값을 수정할 경우 PropertyReadOnlyException이 발생한다.

WorkArea에 등록 정보 설정 : <UserWorkAreaSampleSenderBean.java>

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            userWorkArea.begin("UserWorkArea1");
            // userWorkArea에 값을 NORMAL로 설정한다.
            userWorkArea.set("name", "johan");
            // userWorkArea에 값을 READ_ONLY로 설정한다.
            userWorkArea.set("company", "TmaxSoft", PropertyModeType.READ_ONLY);
            UserWorkAreaSampleReceiver receiver
                = (UserWorkAreaSampleReceiver) ic.lookup("receiver");
            message = receiver.createMessage();
        } catch (NamingException e) {
            // Do Something...
        } catch (NoWorkAreaException e) {
            // Do Something...
        } catch (PropertyReadOnlyException e) {
            // Do Something...
        }
    }
}
```

```

        // Do Something...
    } catch (NotOriginatorException e) {
        // Do Something...
    }

    return message;
}
}

```

4. WorkArea에 설정된 등록 정보 가져오기

Receiver에서 전파된 UserWorkArea에서 설정된 등록 정보를 가져와 메시지를 생성한다. 저장된 정보를 가져올 때 UserWorkArea에 존재하지 않는 key를 사용하면 null이 리턴된다.

WorkArea에 설정된 등록 정보 가져오기 : <UserWorkAreaSampleReceiverBean.java>

```

public class UserWorkAreaSampleReceiverBean implements UserWorkAreaSampleReceiver {
    public String createMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            // UserWorkArea에 저장한 값을 가져온다.
            String name = (String)userWorkArea.get("name");
            String company = (String)userWorkArea.get("company");
            message = "Hello " + name + " from " + company;

        } catch (NamingException e) {
            // Do Something...
        }

        return message;
    }
}

```

5. UserWorkArea 완료하기

시작한 UserWorkArea를 완료한다. 완료하는 것은 반드시 시작된 Originator에서만 가능하며 그 이외에서 완료하려면 NotOriginatorException이 발생한다.

UserWorkArea 완료하기 : <UserWorkAreaSampleSenderBean.java>

```

public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {
    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            userWorkArea.begin("UserWorkArea1");
            userWorkArea.set("name", "user1");
            userWorkArea.set("company", "TmaxSoft", PropertyModeType.READ_ONLY);
            UserWorkAreaSampleReceiver receiver

```



```

        = (UserWorkAreaSampleReceiver) ic.lookup("receiver");
        message = receiver.createMessage();
        userWorkArea.complete(); // UserWorkArea를 종료한다.

    } catch (NamingException e) {
        // Do Something...
    } catch (NoWorkAreaException e) {
        // Do Something...
    } catch (PropertyReadOnlyException e) {
        // Do Something...
    } catch (NotOriginatorException e) {
        // Do Something...
    }

    return message;
}
}

```

Appendix A: 기본 Java 타입과 DB 필드 매핑

본 부록에서는 JEUS 특징인 Java 필드 종류와 JEUS에서 지원하는 주요 DB 벤더의 DB 컬럼 종류와의 기본 매핑을 설명한다.

A.1. 개요

이 기본 매핑은 CMP Bean의 <schema-info><cm-field><type>이 jeus-ejb-dd.xml에 지정되어 있지 않을 때 사용한다. 이런 경우에는 EJB 시스템은 Bean의 CMP 필드 중 Java 필드를 검색하여 Java 타입을 얻어 오고 <type>은 다음의 각 절에 나오는 기본 매핑에 지정된 것과 같다고 여긴다.

다음의 테이블에서는 3가지의 컬럼들을 보여주고 있다.

- **Java 필드 타입**

검색을 통해 발견된 EJB CMP 필드의 종류

- **SQL 타입**

java.sql.Types 클래스에서 정의된 일반적인 SQL 종류

- **DB 컬럼 타입**

발견된 Java 필드 종류에 기반하여 새로운 테이블을 생성할 때 JEUS가 사용할 DB 컬럼 종류의 이름

A.2. Tibero 필드 - 컬럼 타입 매핑

다음은 Tibero를 위한 EJB CMP 필드와 DB 컬럼 타입 매핑 정보이다.

Java 필드 타입	SQL 타입(java.sql.Types)	Tibero DB 컬럼 타입
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	SMALLINT
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

Java 필드 타입	SQL 타입(java.sql.Types)	Tibero DB 컬럼 타입
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

A.3. Oracle 필드 - 컬럼 타입 매핑

다음은 Oracle를 위한 EJB CMP 필드와 DB 컬럼 타입 매핑 정보이다.

Java 필드 타입	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
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 필드 타입	SQL 타입(java.sql.Types)	Oracle DB 컬럼 타입
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

A.4. Sybase 필드 - 컬럼 타입 매핑

다음은 Sybase를 위한 EJB CMP 필드와 DB 컬럼 타입 매핑 정보이다.

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
byte[]	LONGVARBINARY	IMAGE
java.sql.Date	DATE	DATETIME
java.sql.Time	TIME	DATETIME

Java 필드 타입	SQL 타입(java.sql.Types)	Sybase DB 컬럼 타입
java.sql.Timestamp	TIMESTAMP	Not supported
<Java object>	JAVA OBJECT	IMAGE

A.5. Microsoft SQL Server 필드 - 컬럼 타입 매핑

다음은 Microsoft SQL Server를 위한 EJB CMP 필드와 DB 컬럼 타입 매핑 정보이다.

Java 필드 타입	SQL 타입(java.sql.Types)	Microsoft SQL Server 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	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

A.6. DB2 필드 - 컬럼 타입 매핑

다음은 DB2를 위한 EJB CMP 필드와 DB 컬럼 타입 매핑 정보이다.

Java 필드 타입	SQL 타입(java.sql.Types)	DB2 DB 컬럼 타입
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
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 object>	JAVA OBJECT	LONG VARCHAR

A.7. Cloudscape 필드 - 컬럼 타입 매핑

다음은 Cloudscape를 위한 EJB CMP 필드와 DB 컬럼 타입 매핑 정보이다.

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 필드 타입	SQL 타입(java.sql.Types)	Cloudscape DB 컬럼 타입
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.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

A.8. Informix 필드 - 컬럼 타입 매핑

다음은 Informix를 위한 EJB CMP 필드와 DB 컬럼 타입 매핑 정보이다.

Java 필드 타입	SQL 타입(java.sql.Types)	Informix DB 컬럼 타입
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	VARCHAR
Boolean	BIT	VARCHAR
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHAR

Java 필드 타입	SQL 타입(java.sql.Types)	Informix DB 컬럼 타입
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

Appendix B: Instant EJB QL API Reference

본 부록에서는 Instant EJB QL API Reference로 메소드와 인터페이스의 사용법에 대해서 설명한다.

B.1. 개요

EJB QL API는 EJB 클라이언트 애플리케이션 개발자가 클라이언트의 코드에 직접 EJB QL 질의를 지정하여 EJB finder 메소드들이 가지는 제약점을 극복할 수 있도록 한다. 이 API는 단 하나의 인터페이스와 하나의 메소드로 구성되어 있다.



API의 사용은 아주 극단적인 상황에서만 사용되어야 한다. 이 API는 표준 EJB Entity finder 메소드보다 고정적이고 다소 비효율적이다.

B.2. EJBInstanceFinder Interface

interface jeus.ejb.bean.objectbase.EJBInstanceFinder

이 인터페이스는 jeus-ejb-dd.xml 파일의 <enable-instant-ql> element가 true로 설정되어 있는 환경에서 CMP 2.0 Entity Bean의 home Interface에 의해 구현된다. 이 인터페이스는 클라이언트 코드 내에 임의의 EJB QL 질의를 직접 삽입할 수 있도록 한다.

```
public abstract interface EJBInstanceFinder extends Remote
```

B.3. EJBInstanceFinder Method

java.util.Collection findWithInstantQL

- 사용법

"ejbQLQuery" 파라미터로 표현된 EJB QL 질의에 해당하는 Bean의 EJB 집합을 리턴한다.

```
java.util.Collection findWithInstantQL (String ejbQLQuery)
```

- 파라미터

파라미터	설명
string ejbQLQuery	유효한 EJB QL 문장으로 "?"가 없는 것이어야 한다. 이 문법은 JEUS에서 정의한 EJB QL 3가지 추가 사항의 대상 중 하나이다.

- 반환값

반환값	설명
java.util.Collection	질의에 답에 해당하는 Bean Interface의 집합이다.

- 예외
 - FinderException
 - RemoteException