JEUS Server 안내서

Document info
Document name: JEUS Server 안내서
Document date: 2005-06-06
Manual release version: 3
Software Version: JEUS 5

차 례

그림 목차

표 목차

매뉴얼에 대해서

매뉴얼의 대상

본 문서는 JEUS 의 전반적인 기술에 대해서 모두 다루었다. 그러므로 JEUS 의 설치부터 세팅, 모니터링 및 유지보수를 하는 관리자나 기타 관계자를 대상으로 한다.

매뉴얼의 전제 조건

본 문서를 읽기 전에 JEUS 소개나 JEUS 설치 안내서를 읽어보길 권한다. 그리고 JEUS 시작하기도 읽어보면 도움이 된다.

그리고, 어떤 장에서는 관련 내용의 기본 지식이 필요하다. 예를 들면, JNDI, JMX, Connector 등을 보기 위해서는 이들 기술에 대한 배경 지식이 있어야 한다. 이런 경우에는 다른 문서를 보거나 스펙을 읽어보길 바란다.

기본적으로 본 문서에서는 J2EE 와 Java 스펙에 대한 것은 설명하지 않고, JEUS 관련 내용만 설명한다.

매뉴얼의 구성

본 매뉴얼은 JEUS 의 메인 설정 파일인 JEUSMain.xml 의 계층 구조를 따라서 구성되었다. 실제 이 구성은 JEUS 의 구조와 비슷하다.

다른 JEUS 매뉴얼처럼 본 매뉴얼도 다양한 방법으로 읽을 수 있다.

1. 일반 책 처럼 처음부터 끝까지 통독할 수 있다.
2. 각 장은 서로 독립적인 내용이므로 필요한 부분만 정독할 수 있다.
3. 핸드북처럼 필요한 부분을 찾아서 볼 수 있다. 그러나 이렇게 하기 전에 먼저 JEUS 에 대해서 충분히 이해하길 권한다.

JEUS Server 안내서는 다음처럼 17 장으로 구분되어 있다.

1. 소개
2. 따라하기
3. JEUS 개요
4. JEUS Manager
5. JNDI Naming Server
6. External Resource
7. DB Connection Pool 과 JDBC
8. Node
9. Session Server
10. 분산 Session Server
11. Engine Container
12. Transaction Manager
13. Engine
14. JEUS Logging
15. JEUS Clustering
16. JEUS 의 J2EE 어플리케이션
17. 마치며

또한, 자세한 정보나 콘솔 툴, XML 설정 파일, API 등을 소개하는 11 개의 부록을 포함하고 있다. 목록은 다음과 같다.

A. jeus Script Reference

B. jeusadmin Console Tool Reference

C. dbpooladmin Console Tool Reference

D. tmadmin Console Tool Reference

E appcompiler 콘솔 툴 레퍼런스

F JEUS Manager Properties

G. JEUS JTS Properties

H. JEUS JNDI Properties

I. 주요 벤더의 JDBC Data Source 구성 예

J. JEUSMain.xml XML Configuration Reference

K. JEUS Server Ant Task Reference

L. standard.property

관련 매뉴얼

본 문서를 읽은 후에는 다음 문서를 읽어보기를 권한다.

- JEUS Web Container 안내서.
- JEUS EJB 안내서.
- JEUS JMS 안내서.
- JEUS Web Server 안내서.
- JEUS Client Application 안내서
- JEUS 웹 관리자 안내서
- JEUS Connector 안내서
- JEUS Security 안내서
- JEUS JMX 안내서
- 문서 중간 중간 언급한 스펙들

JEUS 5 에서 변경된 사항

이 절은 JEUS 버전 4.x 대와 비교해서 JEUS 버전 5 의 가장 큰 변화에 대해서 언급한다. 이 절에서는 JEUS 4.x 대 버전을 이미 사용하고 있는 사용자를 위한 것으로, JEUS 5 에 대한 새로운 정보를 빠르게 찾을 수 있도록 한다. J2EE 1.4 spec 에 의한 변화는 spec 을 참고하기 바란다.

- deployment 방식 : 기존의 4.x대에서는 각 엔진별로 deploy하는 방식이 모두 달랐다. 5에서는 통합되어 단일한 application deployment 방식으로 바뀌었다. 또한 auto deploy와 two phase deploy, application 자동분배 등의 기능이 추가되었다. 이 부분은 16장을 참고하기 바란다.
- 각종 descriptor 가 DTD 기반에서 XML schema 기반으로 바뀌었다.
- JMX MBean 을 통해 보다 많은 JEUS 의 resource 와 application 들을 제어할수 있게 되었다. 이는 JMX MBean javadoc 을 참고하기 바란다.
- JEUS Security 가 보다 강력하고 세밀한 제어가 가능하게 되었다. 이는 JEUS Security 안내서를 참고하기 바란다.
- virtual node 의 개념이 추가되어 하나의 node 에서도 여러 process 의 JEUS Manager 가 Clustering 될수 있게 되었다.
- Port 통합으로 하나의 JEUS Manager 당 하나의 listen port 를 사용하여 통신을 할 수 있게 되었다.
- Non-blocking I/O 의 도입으로 보다 확장성 있는 통신 구조를 가지게 되었다.
- jeusadmin 의 사용법이 변경되었다. 부록 B 를 참고하기 바란다.

위의 변화된 내용 이외에도, 이 매뉴얼은 여러 부분에 걸쳐서 수정이 되었다. 예제에 사용된 그림도 업데이트가 되었다. JEUS 5 의 또 다른 수정 내용과 새로운 기능에 대한 정보는 다른 JEUS 5 매뉴얼의 이 장을 참조하기 바란다.

일러두기

표기 예	내용

표기 예	내용
텍스트	본문, 12 포인트, 바탕체 Times New Roman
텍스트	본문 강조
CTRL+C	CTRL과 동시에 C를 누름
`public class myClass { }`	Java 코드
`<system-config>`	XML 문서
참조: / 주의:	참조 사항과 주의할 사항
JEUS_HOME	JEUS 가 실제로 설치된 디렉토리 예)c:\jeus
jeusadmin nodename	콘솔 명령어와 문법
[파라미터]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨. 예)<node name>은 실제 hostname 으로 변경해서 사용
|	선택 사항. 예) A|B: A 나 B 중 하나
…	파라미터 등이 반복되어서 나옴
?, +, *	보통 XML 문서에 각각 "없거나, 한 번", "한 번 이상", "없거나, 여러 번"을 나타낸다.
. . .	XML 이나 코드 등의 생략

표기 예	내용
<<FileName.ext>>	코드의 파일명
그림 1.	그림 이름이나 표 이름

OS 에 대해서

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

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

용어 설명

다음에 소개되는 용어는 본 문서 전체에 걸쳐서 사용되는 용어이다. 용어가 이해하기 어렵거나 명확하지 않을 때는 아래 정의를 참조하기 바란다.

용어	정의
임시 커넥션	커넥션 풀이 꽉 찼을 때, 만들어져서 사용 후 폐기되는 커넥션
클러스터링	서로간에 연결된 컴포넌트의 그룹. 시스템을 더 안정적이고 효율적으로 만들 때 사용된다.
Base port	다른 포트 번호를 계산하는데 기본이 된다. 그리고 클라이언트가 JEUS Manager 로 접속하기 위해서도 세팅한다. 기본값으로 9736

용어	정의
Container	런타임 어플리케이션 환경을 제공하는 실제 소프트웨어 구조
EIS	기존의 *Enterprise Information Systems.*
Engine	엔터프라이즈 어플리케이션에 서비스를 제공한다. J2EE 스펙에서 Engine 은 *Container* 라고 불린다.
Engine Container	JEUS 에서만 사용하는 개념으로, 여러 Engine 을 관리하는 단위이다. 각 Engine Container 는 자신의 JVM 에서 동작한다. 단 'default' Engine Container 는 JEUS Manager 와 동일한 JVM 에서 동작한다.
Group	JEUS 에서 사용되는 논리적인 개념으로, backup node 와 함께 사용된다. 대부분의 경우 노드와 동일하다.
J2EE	*Java 2 Enterprise Edition* 은 Sun Microsystems 에서 제정한 스펙들을 일컫는다. 이 스펙은 엔터프라이즈 어플리케이션을 Java 플랫폼으로 구현하는 것에 대해 정의하고 있다. JEUS 는 J2EE 1.4 스펙을 구현하고 있다.
JEUS	*Java Enterprise User Solution* 의 약자. JEUS version 5 는 J2EE 1.4 호환 WAS 이다.
JEUS Manager	JEUS Manager 는 JEUS 의 핵심 컴포넌트로 JEUS 노드를 관리하며, 다른 JEUS Manager 와 클러스터링을 구성한다.
JVM	*Java Virtual Machine* 의 약자

용어	정의
Load balancing	클러스터링에서 작업이 한쪽으로 몰리지 않도록 하는 기술
Node	JEUS 에서 사용되는 개념으로, 하나의 머신에서 작동하는 하나의 JEUS 를 뜻한다.
RA	Resource Adapter 의 약자
RM	Resource Manager 의 약자
Session	제한된 시간 동안 하나의 클라이언트에서 실행한 관련 작업 집합
TM	Transaction Manager 의 약자
TO (to)	TimeOut 의 약자
TX (Tx)	Transaction 의 약자
WAS	*Web Application Server* 의 약자. 복잡한 웹 어플리케이션을 실행하고 관리하는 미들웨어.
WebtoB	Tmax Soft 에서 만든 고성능 웹서버

연락처

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

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

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

China
Beijing Silver Tower, RM 1507, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
Tel: 86-10-64106148 Fax: 86-10-64106144
E-mail : info@tmaxchina.com.cn
Web (Chinese): http://www.tmaxchina.com.cn

1 소개

JEUS 는 TmaxSoft㈜가 만든 JEUS Web Application Server 5 의 핵심 컴포넌트이다. JEUS 는 수백만 온라인 사용자가 사용하는, 복잡한 E-Commerce 사이트를 쉽게 구축할 수 있다.

JEUS 5 WAS 는 J2EE 1.4 와 Sun Microsystems 에서 제정한 여러가지 사양을 구현 제공하고 있으며, 효율적이며 뛰어난 신뢰성과 확장성을 제공한다. 본 제품을 사용하면 쉽고, 빠르며, 신뢰성이 뛰어나다는 것을 느끼게 될 것이다.

본 문서는 JEUS 5 서버를 설정하고 운영하기 위한 기본적인 내용을 다룬다. 매뉴얼의 내용은 JEUS 5 서버의 전체적인 내용(3 장)부터 시작하며, 진행해 나갈수록 보다 자세한 내용을 다룬다. 마지막 부록 부분에서는 관련 콘솔 툴과 설정 파일에 대해서 자세하게 설명했다.

몇몇 큰 기능들(EJB 와 Servlet)은 각각의 매뉴얼로 나눠져 있다. 그러나 그 매뉴얼을 보기 전에 본 매뉴얼을 한 번 이상 읽어볼 것을 권한다.

우선 JEUS 를 접해보기 위해서 간략히 사용법을 소개한다

2 따라하기

본 장에서는 JEUS 서버를 시작하고 종료하는 아주 기본적인 내용을 소개한다.

아래 예에서 JEUS 노드명은 “johan”이라고 가정한다. 이 이름이 나오는 부분에 자신의 컴퓨터 이름(정확하게는 hostname)으로 바꿔서 진행 하길 바란다.

1. 우선 JEUS 를 설치하고, OS 의 패스와 환경 변수가 제대로 세팅되었는지 확인한다. 특히 JEUS_HOME/bin 디렉토리가 패스에 잡혀 있는지 확인한다. JEUS 설치에 대한 자세한 내용은 JEUS 설치 안내서를 참조한다.

   참고: 본 매뉴얼에서 “JEUS_HOME”이라고 나오면 JEUS 의 루트 디렉토리를 뜻한다(예: C:\jeus).

2. JEUSMain.xml 이 있는지 확인한다. 이 파일은 JEUS 의 메인 환경 설정 파일이다. 이 파일은 JEUS_HOME\config\<node name> 디렉토리에 위치한다. 여기서 <node name>은 JEUS 가 설치된 머신의 host name(예제에서는 “johan”)이다. 아래는 JEUSMain.xml 의 예제이다.

<<*JEUSMain.xml*>>

```
<?xml version="1.0"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <node>
        <name>johan</name>
        <sequential-start>false</sequential-start>
        <enable-webadmin>true</enable-webadmin>
        <engine-container>
            <name>mycontainer</name>
            <sequential-start>false</sequential-start>
            <engine-command>
                <type>ejb</type>
                <name>Engine1</name>
            </engine-command>
            <engine-command>
                <type>servlet</type>
```

```
                <name>Engine1</name>
            </engine-command>
        </engine-container>
    </node>
</jeus-system>
```

위 예제에서, JEUS(또는 JEUS Manager)는 두 개의 *Engine*(EJB Engine 하나와 Servlet Engine 하나)으로 구성된 *Engine Container* 하나가 구성되어 있다. JEUS 노드의 이름은 hostname 과 일치한다(예, “johan”).

3. JEUS_HOME\bin 디렉토리에 있는 jeus 실행 스크립트를 실행한다. 그러면 JEUS Manager 준비 단계로 실행된다. 이때부터 JEUS Manager 는 boot 같은 명령어를 받아서 시작할 수 있게 된다. 그러나 boot 명령을 실행하기 전까지는 JEUS 가 실행되지 않으며 WAS 서비스도 작동되지 않는다.

4. 다른 터미널 창에서 ‘jeusadmin johan’이라고 입력한다. 여기서 ‘johan’은 반드시 접속하고자 하는 머신의 hostname 으로 대체한다. jeusadmin 은 JEUS Manager 를 통해서 JEUS 를 제어하는 콘솔 툴이다.

5. JEUS 의 관리자의 사용자명과 패스워드를 입력한다. 일반적으로 관리자의 사용자명은 administrator 이고 패스워드는 JEUS 를 설치할 때 입력한 값이다.

6. jeusadmin 의 프롬프트가 뜨면 boot 라고 입력한다.

7. 잠시 후 프롬프트가 다시 뜨면 JEUS 가 제대로 부팅되었고, 다시 명령어를 받을 수 있는 상태가 되었다는 것을 나타낸다.

8. 웹 브라우저를 열어서 주소 창에 http://localhost:9744/webadmin 이라고 입력한다. 만약 위 예제의 JEUSMain.xml 에서 웹 관리자를 사용할 수 있게 해놓았다면, 웹 관리자라는 웹 기반의 관리툴이 시작된다. [그림 1]

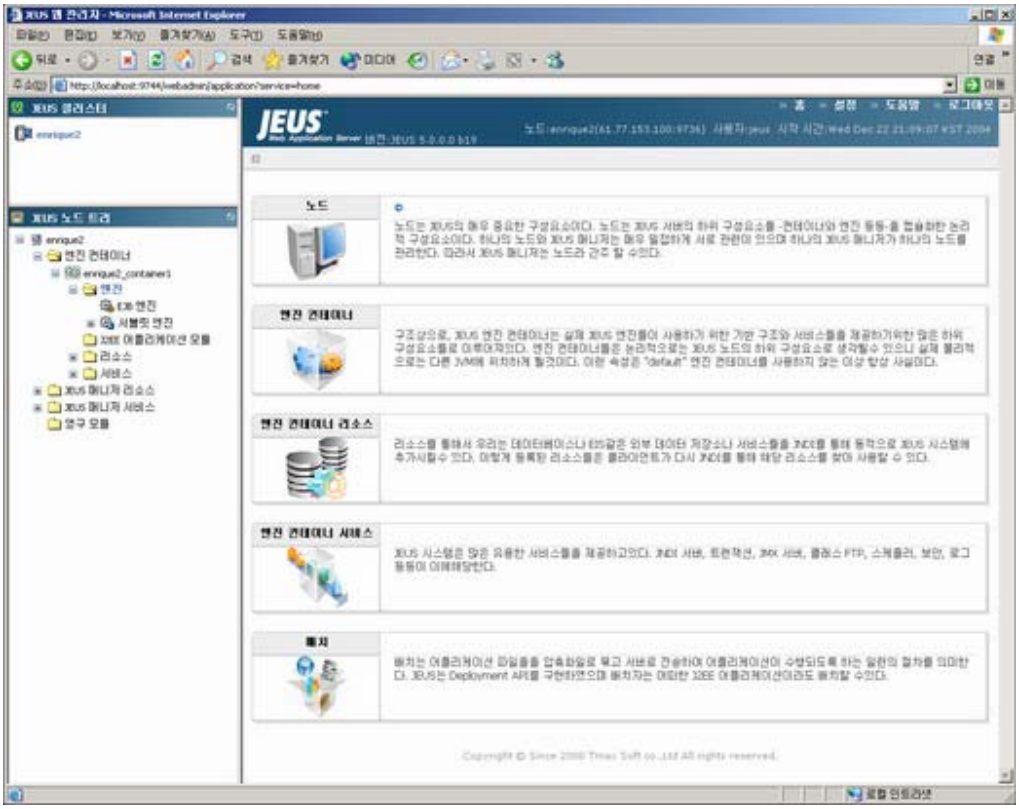

그림 1. 웹 브라우저에서 사용하는 관리툴

9. jeusadmin 콘솔 창으로 다시 가서, 'jeusexit'이라고 명령어를 입력한다. 이렇게 하면 JEUS Manager 는 완전히 종료되게 된다(웹 관리자에서 down 을 하면 JEUS Manager 는 종료되지 않고 대기 모드로 바뀌게 될 뿐이다).

10. 'exit'을 입력해서 jeusadmin 을 빠져 나온다.

3 JEUS 개요

3.1 소개

이번 장에서는 JEUS 의 전반적인 내용과 클러스터링을 구성하기 위한 방법에 대해서 소개한다. 그리고 JEUS 의 디렉토리 구조와 JEUS 에 필요한 환경변수, 그리고 끝으로 관련 XML 설정 파일의 전반적인 내용에 대해서 소개한다.

3.2 JEUS 와 WebInOne

JEUS 는 J2EE 1.4 표준을 준수하는 웹 어플리케이션 서버(WAS)이며, TmaxSoft 의 WebInOne 웹 솔루션의 3 가지 컴포넌트 중 하나이다.

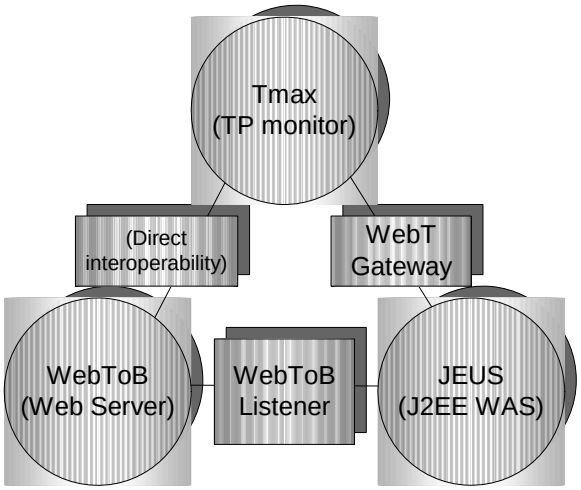

그림 2. WebInOne product suite.

[그림 2] 에서 보듯이 JEUS 는 강력한 성능을 지닌 웹 서버인 WebtoB 와 TP 모니터인 Tmax 와 상호 연동되도록 설계되었다. 뿐만 아니라 JEUS 는 J2EE 1.4 표준을 따르고 있으므로 다른 웹 서버나 DBMS 같은 3rd party 제품들과 완벽한 상호 연동이 가능하다.

3.3 JEUS 시스템의 개념과 역할

[그림 3]는 엔터프라이즈 어플리케이션 솔루션을 제공하기 위해서 JEUS 가 어떻게 다른 웹 서버나 DBMS 등과 통합되는지 보여준다.

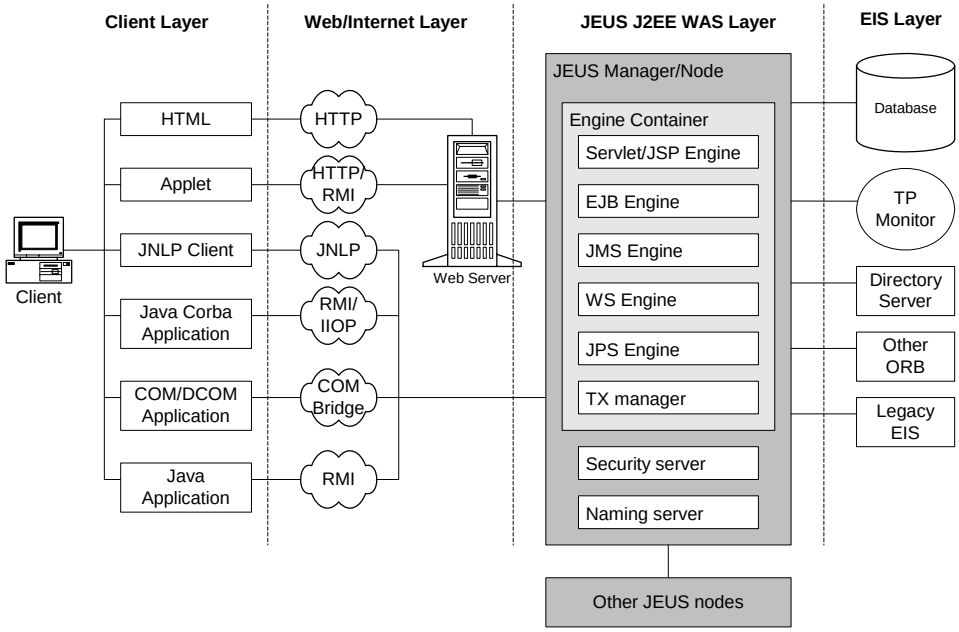

그림 3.JEUS 의 기능과 역할

위 그림에서 나타난 4 가지 레이어는 다음과 같다.

1. **Client Layer** 는 웹 서버나 Java 어플리케이션 또는 Native 어플리케이션으로 구성된다. 최종 사용자는 WAS 의 서비스를 사용하기 위해서 다양한 클라이언트를 사용하며, 이 클라이언트는 다양한 프로토콜 중에 하나를 사용해서 WAS 의 서비스에 접근한다.

2. **Web/Internet layer** 는 클라이언트와 WAS 사이의 중간에서 작동하는 웹 서버나 프로토콜로 정의된다. 이 레이어에서는 정적인 컨텐츠와 부하 분산을 처리한다.

3. **JEUS WAS layer** 는 Java 기반의 미들웨어로 구성되며, Web layer 나 Client layer 로부터 오는 요청을 받아서 처리한다.

4. **EIS layer** 는 비즈니스 데이터나 기존의 legacy 서비스를 나타낸다. WAS 는 JDBC 나 디렉토리 서비스, J2EE Connector 등의 다양한 메커니즘을 통해서 legacy 서비스와 상호 작용한다.

3.4 JEUS System 의 컴포넌트

[그림 4]은 JEUS 시스템의 컴포넌트를 모두 나타낸 구성도이다. 앞 절에서 소개한 내용을 보다 자세하게 알려준다.

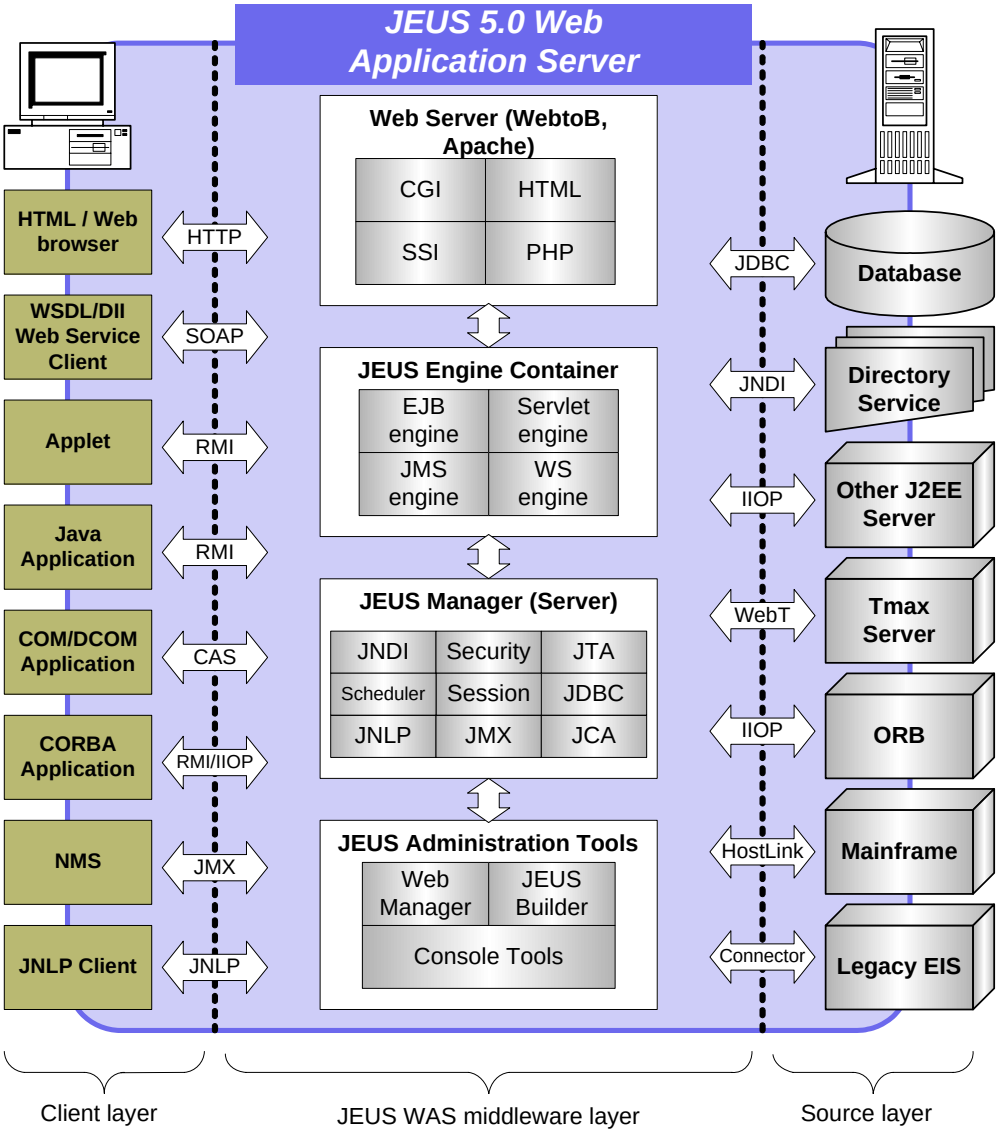

그림 4. JEUS 의 구성

위 그림에서는 다음과 같이 3 가지 layer 로 구성되어 있다.

- **JEUS WAS middleware layer** 는 JEUS Manager 나 Engine Container, Engine 등과 같은 JEUS 의 핵심 부분이다. 이들 내용에 관해서는 차후에 자세히 설명한다. 아래 부분에 있는 것은 JEUS 관리툴이다. 관리

툴은 3 가지로 구성되어 있다. 웹 브라우저에서 사용하는 웹 관리자와 커맨드 라인에서 사용하는 콘솔 툴, 그리고 스윙기반의 GUI 툴인 JEUSBuilder 가 있다. 자세한 내용은 각각의 툴에 대한 안내서를 참조한다.

- **client layer** 는 앞 절에서 소개한 client layer 와 같은 계층으로, JEUS 로 액세스해서 사용하는 계층이다. 이 그룹은 웹 브라우저, Java 어플리케이션, 애플릿, JNLP 어플리케이션, COM/DCOM 어플리케이션, CORBA 어플리케이션 등으로 구성된다.

- **Source layer** 는 기존에 존재하는 디렉토리 서비스나 DB 같은 엔터프라이즈 시스템을 구성한다.

3.5 JEUS 의 서브 컴포넌트

아래 [그림 5]은 JEUS 의 서브 컴포넌트를 나타낸다(3.3 절의 [그림 3] 중에서 JEUS J2EE WAS Layer 부분). 이 그림은 설정 파일인 JEUSMain.xml 에 따라서 논리적으로 기능을 분류한 것이다. 그리고 본 매뉴얼 전체에 걸쳐서, 각 기능의 위치를 나타내기 위해서 계속해서 나온다.

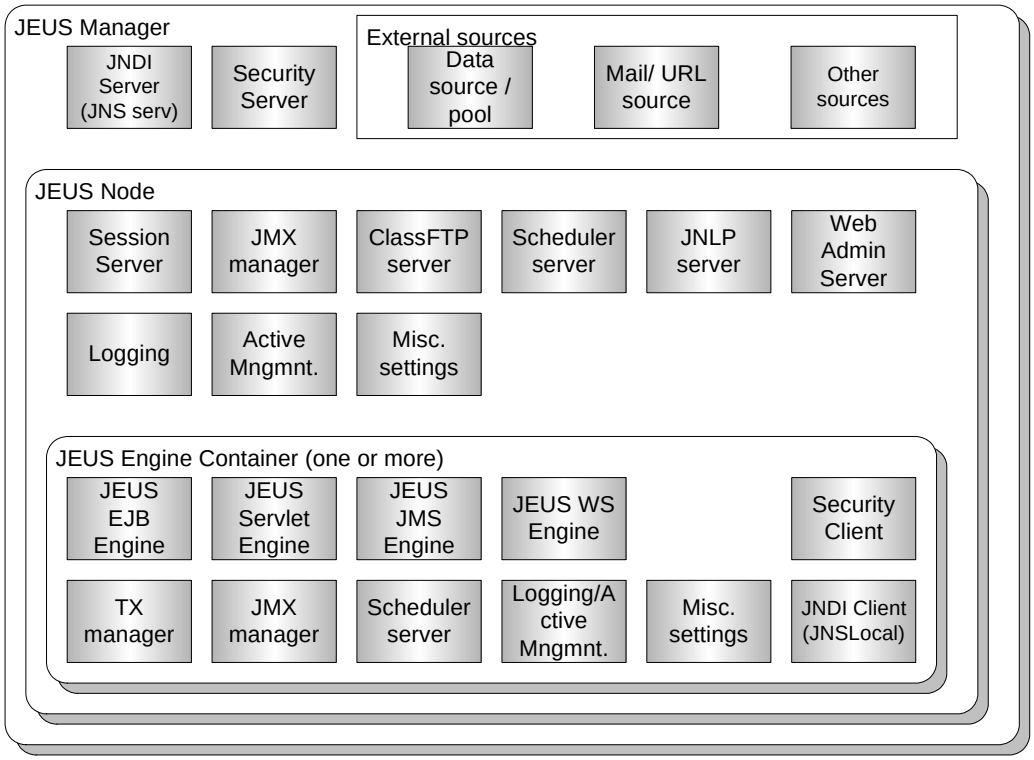

그림 5. JEUS 의 컴포넌트 구성

위 그림에서 중요한 컴포넌트는 다음과 같다.

- **JEUS Manager** 는 전체 서버의 agent 처럼 작동한다. JEUS Manager 는 JNDI Naming Server(5 장) 와 Security Server(JEUS Security 안내서), 그리고 DB Connection 같은 외부 리소스를 등록하고 관리한다. JEUS Manager 는 노드와 연관되어서 작동하므로 Node Manager 라고도 할 수 있다.

- **JEUS 노드**는 가장 중요한 부분으로, 실제 JEUS 자체라고 할 수 있다. JEUS 노드의 메인 컴포넌트는 Engine Container 이다. 하나의 노드에는 여러 개의 Engine Container 가 올라갈 수 있다. 또한 노드는 JEUS Manager 를 통해서 클러스터링으로 동작할 수 있으며, 백업 노드로 설정되어서 Fail-Over 의 기능도 제공한다. 클러스터링과 백업 노드에 관한 자세한 내용은 15 장을 참조한다. Session Server, Scheduler, Logging 같은 기능도 노드의 중요한 컴포넌트들이다.

- **JEUS Group** 은 [그림 5]에는 나타나지 않는데, 하나의 이름으로 여러 개의 Engine Container 를 그룹핑하는 논리적인 개념이다. 그룹핑을 하는데 특별한 세팅이 필요 없으며, 한가지의 경우를 제외하면 노드와

동일하게 사용된다. 이 한가지의 경우란 백업 노드를 설정한 노드가 다운되어서 백업 노드가 작동될 때이다. 이 경우 백업 노드로 작동되는 Engine Container 와 자신의 Engine Container 와는 논리적으로 구분이 되는데, 이 때 그룹이라는 개념이 사용된다. 클러스터링과 백업 노드에 대한 보다 상세한 내용은 15 장에서 다룬다.

- **JEUS Engine Container** 는 여러 개의 JEUS Engine 을 묶는 그룹핑 메커니즘이다. Engine Container 는 각각의 Engine 에게 여러 서비스를 제공한다. 이 서비스에는 JNDI 클라이언트와 보안 클라이언트 등이 있는데, 각각 JNDI Server 와 Security Server 로 접속할 수 있게 하며, 전용 Scheduler 와 트랜잭션 매니저도 제공한다.

  참고: 논리적으로 각 Engine Container 는 [그림 5]에서도 보이듯이 하나의 노드에 속하게 된다. 그러나 실제로는 전용 JVM 위에서 실행되므로 JEUS 노드나 매니저와는 분리되어서 작동한다. 만약 Engine Container 가 “default”로 세팅된다면 JEUS Manager 와 같은 JVM 에서 작동하게 된다. 이에 대한 내용은 11 장에서 다시 논한다.

- **JEUS Engine** 은 J2EE 어플리케이션을 실행시키는 컴포넌트이다. Engine 에는 4 가지 종류가 있다. Web (Servlet)Engine, EJB Engine, JMS Engine, WS(JEUS WebServer) Engine 이 그것이다. 이들 Engine 은 하나의 Engine Container 에 실행되며, 실제로 같은 JVM 위에서 작동된다.

  참고: 위에서 Servlet Engine 이라는 용어가 소개되었다. 그런데, JEUS Web Container 안내서에서는 Web Container 라고 되어 있다. 보통 Engine 과 container 는 서로 통용되는 용어이다. 그러나 “Engine Container”라는 개념은 앞의 두 개념과는 다르다는 것도 주의하기 바란다.

3.6 JEUS 클러스터링

시스템 과부하를 신뢰성있게 처리하는 일반적인 방법인 여러 JEUS 를 연결해서 하나의 클러스터링을 구성하는 것이다. [그림 6]은 JEUS 4 개를 연결한 간단한 클러스터링을 보여준다.

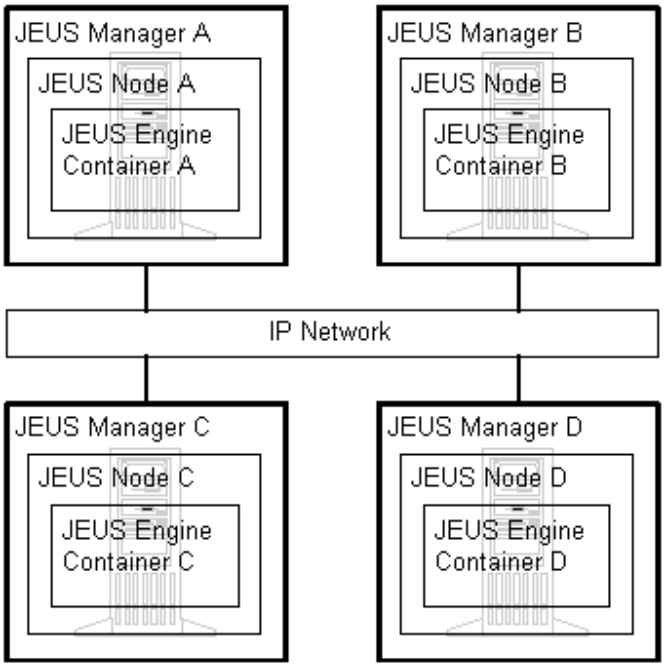

그림 6. 4 개를 연결한 JEUS 클러스터링

[그림 6]에서처럼 각 JEUS 는 고유의 IP 주소를 가지고 있는 전용 머신에서 운영된다.

클러스터링 된 JEUS 들은 계속해서 서로간에 상태를 점검한다. 그래서 노드 하나가 다운되었을 때, 이에 대한 대책으로 백업 노드를 띄운다거나 관리자에게 경고 메일을 보낸다거나 하게 된다.

또한 클러스터링이 되면 각각의 JEUS 에 있는 리소스와 어플리케이션을 같은 클러스터링 환경에 있는 다른 JEUS 에서 액세스가 가능하다. 이는 각 서버의 JNDI Naming Server 가 자신의 클러스터링을 형성하고, 이 JNDI Naming Server 로 리소스와 어플리케이션을 등록시키기 때문이다.

3.7 JEUS 관리 툴

JEUS 에 바로 접속해서 사용하는 툴에는 다음과 같은 것이 있다.

- **웹 관리자**는 HTML 기반의 관리툴이다. JEUS 의 제어, 모니터링, 관리 등의 기능을 제공하며, 별도의 JEUS 웹 관리자 안내서가 제공된다.

- **jeus**(또는 **jeusp**)는 JEUS Manager 를 실행하는 가장 기본적인 툴이다 (부록 A 참조).

- **jeusadmin** 콘솔툴은 command prompt 에서 JEUS 를 컨트롤하는데 사용된다(부록 B 참조).
- **dbpooladmin** 콘솔 툴은 DB Connection Pool 을 모니터링하고 관리한다
- **tmadmin** 은 Engine Container 의 트랜잭션 매니저를 제어한다.

위에서 언급한 것 이외에도 EJB 와 Servlet Engine 을 위한 툴이 존재한다. 자세한 내용은 해당 매뉴얼을 참조한다.

3.8 J2EE 스펙 구현

[표 1]은 Sun Microsystems 에서 제정한 J2EE 의 스펙 중에서 JEUS 5 를 구현한 목록이다.

표 1. JEUS 5 가 지원하는 J2EE 스펙

Specification/Technology	Version supported in JEUS 5
Java 2 Platform, Enterprise Edition Specification	1.4
J2EE Connector Specification	1.5
Enterprise JavaBeans Specification	2.1
Java Server Pages Specification	2.0
Java Servlet Specification	2.4
Java Naming and Directory Interface Specification	1.2.1
Java Message Service Specification	1.1
JDBC Specification	3.0
JavaMail API Specification	1.2

Specification/Technology	Version supported in JEUS 5
Java Transaction API Specification	1.0.1
Java Transaction Service Specification	1.1
JMX Specification	1.2
JNLP Specification	1.0.1
JAAS	1.0.1
SOAP	1.1
WSDL	1.1
JAX-RPC (Full)	1.0
SAAJ (Full)	1.1
J2EE Web Services	1.1
JACC	1.0

위 스펙에 대한 내용을 http://java.sun.com/j2ee/docs.html 에서 구할 수 있다.

중요: 사용하는 JEUS edition 에 따라서 위 표에서 나열된 모든 기능이 구현되지 않을 수 있다. 자세한 것은 각 JEUS edition 의 소개 문서를 참조한다.

3.9 JEUS 의 디렉토리 구조

아래 [그림 7]는 JEUS 를 설치했을 때의 전체 디렉토리 구조이다.

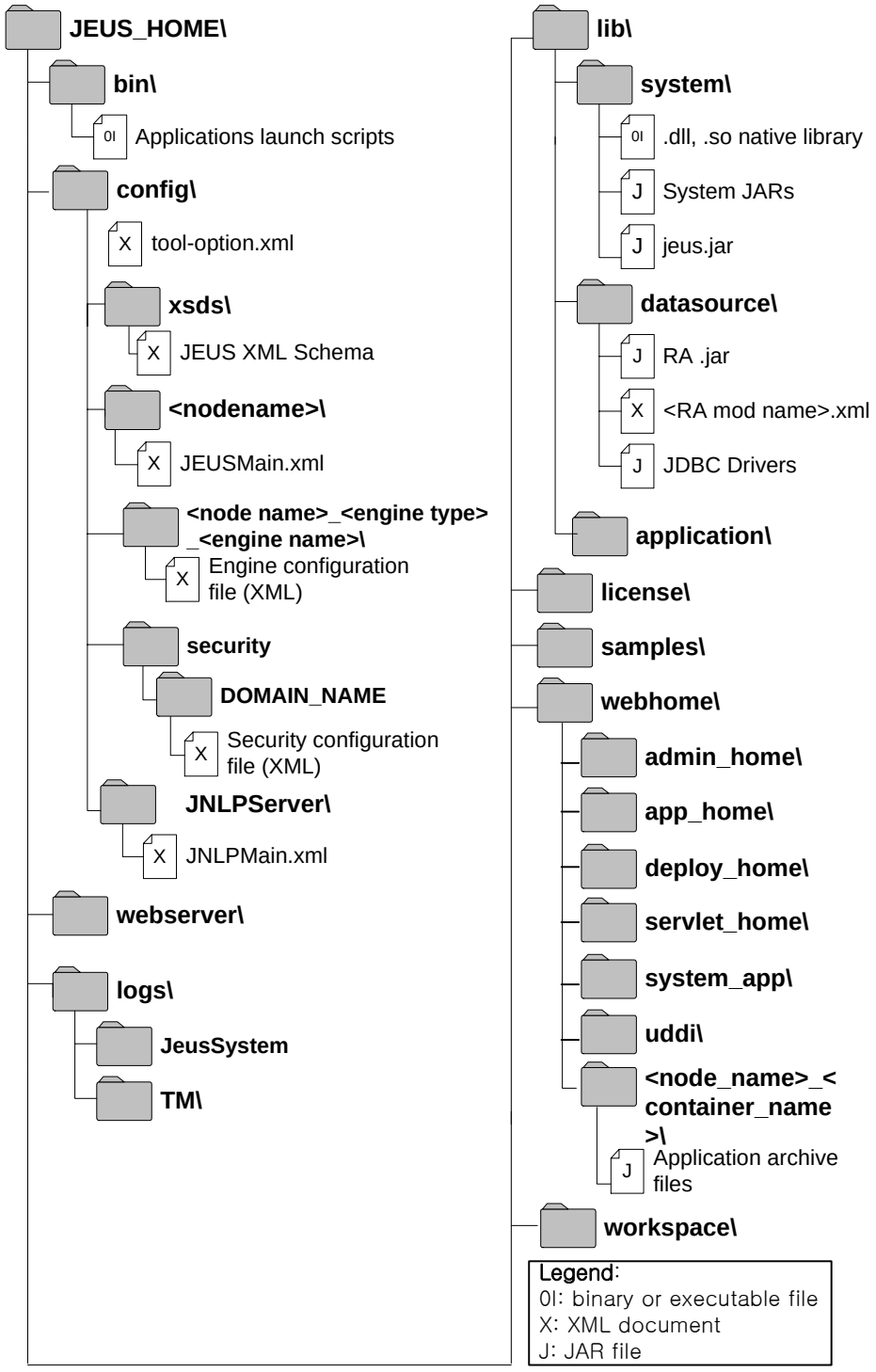

그림 7. JEUS 설치 후 디렉토리 구조

다음은 디렉토리와 파일의 설명이다.

- **JEUS_HOME**은 JEUS 의 최상위 디렉토리이다. 실제 디렉토리 명과 위치는 설치할 때 결정된다. 예) c:\jeus

- **bin**에는 `jeus`, `jeusadmin`, `webadmin`, `ejbadmin`, `securityadmin` 와 같은 실행 파일이나 스크립트가 모여있다.

- **config** 디렉토리는 서버 설정이 저장되는 서브 디렉토리를 가지고 있다.

- **config\xsds** 디렉토리에는 XML 설정 파일을 위한 XML Schema 파일이 위치한다.

- **config\<nodename>**에는 이름이 <nodename>인 설정 파일이 위치한다. <nodename>은 JEUS 가 설치된 서버의 이름(일반적으로 hostname)이다. 만약 현재 JEUS 가 백업 노드로 작동되도록 세팅되었다면, 백업하는 노드의 <nodename> 디렉토리가 존재해야 한다. JEUS Manager 의 설정 파일은 JEUSMain.xml 이며, 이 디렉토리에 존재한다. 예) 노드명이 jeus 일 때 ”c:\jeus\config\jeus”가 된다.

- **config\<nodename>\<nodename>_<enginetype>_<enginename>**에는 Engine 별로 설정 파일을 가지고 있다. <entinetype>은 “ejb”, “jms”, “servlet” 이렇게 셋 중 하나이며, <enginename>은 JEUSMain.xml 에서 설정한 이름이다. 예) “c:\jeus\config\jeus\jeus_servlet_Engine1”일 때, 노드명은 “jeus”, Engine 의 종류는 “servlet”이며, 사용자가 지정하는 Engine 명은 “Engine1”이 된다.

- **config\<nodename>\JNLPServer** 에는 JNLP 서버의 설정 파일인 JNLPMain.xml 파일이 위치한다.

- **config\<nodename>\security 에**는 이 node 에서 사용하는 security configuration 이 들어가 있다. 자세한 것은 Jeus Security 매뉴얼을 참조 하기 바란다.

- **webserver**는 JEUS 가 설치될 때 JEUS WebServer 가 설치되는 디렉토리이다. 자세한 정보는 JEUS Web Server 안내서를 참조한다.

- **lib** 디렉토리는 JEUS 가 부팅될 때, 이 디렉토리의 모든 서브 디렉토리를 스캐닝해서 .jar 파일이나 .zip 파일을 classpath 로 추가한다. 서브 디렉토리 스캐닝 순서는 system → application → datasource 순이다.

- **lib\application** 디렉토리는 사용자 어플리케이션에서 사용하는 클래스 파일이 위치한다. 이 디렉토리에 있는 모든 .zip 파일이나 .jar 파일,

디렉토리 형태의 일반 클래스들은 classpath 로 추가된다. 클래스 파일은 .jar 나 .zip 파일 보다 먼저 읽힌다.

- **lib\system** 디렉토리는 JEUS 가 사용하는 라이브러리를 가지고 있다. 그러므로 사용자는 이 디렉토리를 수정해서는 안 된다. .so 나 .dll 같은 Native Driver(예를 들면, Connector Resource Adapter 에서 필요한 드라이버 등…)는 이 디렉토리에 놓여야 한다. 또 여기에는 JEUS 의 클래스인 jeus.jar 가 있다.
- **lib\datasource**는 JDBC 드라이버의 .jar 파일과 설정 파일이 위치한다.
- **license** 디렉토리에는 JEUS 라이선스 파일이 있다. 이 파일을 JEUS 가 실행되기 위해서 반드시 필요한 파일이다.
- **logs** 디렉토리에는 시스템 로그가 기록된다.
- **logs\TM**에는 트랜잭션 매니저의 로그 정보를 기록한다.
- **logs\JeusSystem\과 그 이하의 directory 에는** node 와 engine container, engine 들이 생성하는 로그를 기록한다.
- **samples** 디렉토리에는 JEUS 의 예제들이 있다.
- **webhome** 디렉토리는 J2EE 어플리케이션(웹 어플리케이션, EJB)이 deploy 되는 홈 디렉토리이다.
- **webhome\admin_home**에는 WebManager 에서 사용되는 Serlver/JSP 파일이 있다.
- **webhome\<node_name>_<container_name>** 디렉토리는**<node_name>_<container_name>에 해당하는 engine container 에서 사용되는 application 들이 존재한다.** 자세한 것은 11 장을 참조하기 바란다.
- **workspace**디렉토리는 JEUS 가 동작하면서 필요한 작업을 하는 임시 디렉토리이다. 그러므로 이 디렉토리를 수정해서는 안 된다.

3.10 JEUS 환경 변수

[표 2]에서 JEUS 에서 사용하는 환경 변수를 정리했다. 이 변수는 필요하다면 수정해서 사용할 수 있다. XML 설정 파일에서는 이들 환경 변수를 사용할 수 없다.

이 환경 변수는 모두 JEUS_HOM\bin\jeus.properties 에서 설정되어 있으며, JEUS_HOME\bin 디렉토리의 모든 스크립트에서 사용된다.

표 2. JEUS 의 환경 변수들

환경변수	내용	예
JEUS_HOME	JEUS 가 설치된 홈 디렉토리. 필수 사항	C:\Jeus
JEUS_BASEPORT	JEUS Manager 로 접속하기 위한 TCP/IP 포트이며, 다른 기능을 위한 포트를 계산하는데 기본이 된다.	9736 (default)
JEUS_WSDIR	JEUS Web Server 의 홈 디렉토리	C:\Jeus\webserver (default)
JDK_HOME	JDK 의 홈 디렉토리	c:\jdk1.4 (must be set)
WORKING_DIR	JEUS 의 임시 디렉토리로 이 환경변수는 clientcontainer 에서만 사용된다.	c:\Jeus\workspace (default)
DEPLOY_HOME	.ear, .jar, .war 파일이 자동으로 Deploy 되기 위해 놓일 디렉토리	c:\Jeus\webhome\deploy_home (default)

환경 변수를 변경하는 방법은 OS 에 따라 다르다. 이에 대해서는 OS 매뉴얼을 참조하기 바란다.

참고: 모든 환경 변수는 설치할 때 디폴트 값으로 정해진다. 대부분의 경우 그 값을 그대로 사용하면 된다.

3.11 JEUS XML 환경 설정 파일

JEUS 는 환경 설정을 위해서 각각 고유의 XML 포맷을 사용하며, 직접 수정하거나 툴을 사용해서 수정할 수 있다.

아래 [표 3]은 JEUS 의 XML 설정파일과 내용, 위치를 정리한 표이다. J2EE 의 표준 디스크립터 파일인 “web.xml”이나 “ejb-jar.xml” 파일도 사용된다. 이들 파일은 해당 J2EE 스펙을 참조한다.

표 3. JEUS 의 XML 설정 파일

파일명 (XML Schema 파일)	위치	목적	참조 매뉴얼
JEUSMain.xml (jeus-main.xsd)	JEUS_HOME\config\<nodename>\	JEUS Manager 와 노드를 관리하는 기본 설정 파일	JEUS Server 안내서 (본 매뉴얼).
WEBMain.xml (web-main.xsd)	JEUS_HOME\config\<nodename>\<Servlet Enginedir>\	Servlet/JSP Engine 설정	JEUS Web Container 안내서
jeus-web-dd.xml (jeus-web-dd.xsd)	Web application archive 의 WEB-INF	JEUS Web application (Servlet app) deployment descriptor.	JEUS Web Container 안내서
EJBMain.xml (ejb-main.xsd)	JEUS_HOME\config\<nodename>\<EJB Enginedir>\	EJB Engine configuration.	JEUS EJB 안내서
jeus-ejb-dd.xml (jeus-ejb-dd.xsd)	EJB application archive 의 META-INF	JEUS EJB module deployment descriptor.	JEUS EJB 안내서

파일명 (XML Schema 파일)	위치	목적	참조 매뉴얼
JNLPMain.xml (jnlp-resource-config.xsd)	JEUS_HOME\config\JNLPServer\	JNLP 설정.	JEUS Client Application 안내서
jeus-client-dd.xml (jeus-client-dd.xsd)	client application archive 의 WEB-INF	Application client deployment descriptor.	JEUS Client Application 안내서
jeus-connector-dd.xml (jeus-connector-dd.xsd)	Resource adaptor archive 의 META-INF	Resource adaptor deployment descriptor.	JEUS Connector Application 안내서
JMSMain.xml (jms-main.xsd)	JEUS_HOME\config\<nodename>\<JMS Enginedir>\	JMS Engine 설정.	JEUS JMS 안내서
policies.xml (**policies**.xsd)	JEUS_HOME\config\<nodename>\security\<domainname>\	JEUS Security policy 설정	JEUS Security 안내서
subjects.xml (**subjects**.xsd)	JEUS_HOME\config\<nodename>\security\<domainname>\	JEUS Security subject 설정	JEUS Security 안내서
vhost.xml (virtual-host.xsd)	JEUS_HOME\config\	JEUS virtual host 설정	JEUS Server 안내서

파일명 (XML Schema 파일)	위치	목적	참조 매뉴얼
jeus-web-dd.xml (jeus-web-dd.xsd)**, jeus-ejb-dd.xml** (jeus-ejb-dd.xsd)**, jeus-client-dd.xml** (jeus-client-dd.xsd)	Webservice client archive 의 META-INF	Webservice client 설정	JEUS Webservice 안내서
jeus-webservices-config.xml (**jeus-webservices-**config.xsd)	Webservice client archive 의 META-INF	Webservice ant task 에서 사용하는 설정파일	JEUS Webservice 안내서

모든 XML schema 파일은 JEUS_HOME\config\xsds\.에 위치한다.

3.12 JEUS XML 파일 헤더와 검증(Validation)

JEUS 에서 사용하는 XML 파일은 JEUS 에서 정의한 XML 헤더로 시작해야만 한다. 각 파일에서 사용하는 헤더는 다음과 같다.

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

“XXX”는 다음 표의 값으로 치환된다.

파일명	**XXX** (root element)
JEUSMain.xml	jeus-system
WEBMain.xml	web-container
jeus-web-dd.xml	jeus-web-dd
EJBMain.xml	ejb-Engine

파일명	**XXX** (root element)
jeus-ejb-dd.xml	jeus-ejb-dd
JNLPMain.xml	jnlp-resource-config
jeus-client-dd.xml	jeus-client-dd
JMSMain.xml	jms-server
jeus-connector-dd.xml	jeus-connector-dd
policies.xml	policies
subjects.xml	subjects
vhost.xml	virtual-hosts

J2EE 스펙에 정의된 XML 파일은 각 스펙에 정의된 헤더를 사용한다.

본 매뉴얼에서는 위 표에 있는 XML 파일이 나올 때는 헤더를 표시하지 않고, 헤더가 포함되어 있는 것으로 가정한다.

중요: 매뉴얼에서 사용된 모든 XML 태그는 XML 설정 파일을 정확하게 나타나기 위해서 순서 대로 작성되어있다. XML 태그 순서에 대한 내용은 부록 J 나 XML Schema 파일을 참조한다. 일반적으로 XML 설정 파일을 수정할 때는 웹 관리자를 사용할 것을 권한다(이 방법에 대한 것은 각 매뉴얼의 XML 설정을 소개하는 부분을 참조한다).

3.13 JEUS Base Port

JEUS 는 서브 컴포넌트와 통신하기 위해서 소켓 포트를 사용한다. 가장 주목해야 할 포트는 JEUS Manager 의 base port 이다. 시스템에서 사용되는 다른 포트들은 이 포트를 기반으로 계산되어 정해진다. 앞서 보았듯이 이 포트는 JEUS_BASEPORT 라는 환경 변수에 세팅된다. 디폴트 값은 “9736”이다. 현재 JMS 를 제외한 나머지 port 들은 JEUS 내부에서 쓰는 번호로 실제 listen port 로 사용하지는 않는다.

3.14 JEUS Virtual Host

JEUS Manager 를 지칭하는 이름은 그 Manager 가 떠있는 hostname 이다. JEUS 4.x 까지는 이 hostname 을 Manager 의 identity 로 사용하였기 때문에 하나의 host 에서 여러 개의 JEUS Manager 를 띄우고 이들을 clustering 으로 묶을 수가 없었다. (clustering 으로 묶지 않는다면 baseport 를 다르게 해서 실행할 수는 있었다.)

JEUS 5 에서는 virtual host 라는 개념을 도입해서 하나의 node 에 여러 개의 JEUS Manager 가 실행되고 이들이 충돌없이 clustering 으로 묶일 수 있도록 하였다. 즉, node name 과 base port 를 한 쌍의 identity 로 사용하고 여기에 임의의 virtual node name 을 지정하고, 이 virtual node name 을 node name 처럼 사용하는 것이다.

이 경우 JEUS 의 config directory 의 node directory 도 virtual node name 을 사용해야 하며 이 node 내의 engine container 나 engine 들도 virtual node name 을 node name 인것처럼 사용한다. 또한 jeus admin 등 node name 을 사용하는 모든 곳에서 virtual node name 을 사용하면 그 virtual node name 에 해당하는 JEUS Manager 를 지칭하게 된다.

Virtual host 지정은 config directory 의 vhost.xml 로 지정한다. 이 파일의 예는 다음과 같다.

<<vhost.xml>>

```
<virtual-hosts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <enable>true</enable>
      <host>
        <name>myNode:9736</name>
        <virtual-name>node1</virtual-name>
      </host>
      <host>
        <name>myNode:9836</name>
        <virtual-name>node2</virtual-name>
      </host>
      <host>
        <name>yourNode:9936</name>
        <virtual-name>node3</virtual-name>
      </host>
      <host>
        <name>yourNode:10036</name>
```

```
        <virtual-name>node4</virtual-name>
</virtual-hosts>
```

각 element 의 설명은 다음과 같다.

- **enable :** virtual host 설정을 적용할지의 여부를 지정한다.
- **host :** 각 virtual host 를 설정한다. 하위 element 의 name 에는 (실제 node name): (base port)가 지정되고 virtual-name 에는 virtual host name 이 지정된다.

지정된 virtual host 로 JEUS manager 를 실행하기 위해서는 –Djeus.baseport option 을 사용해서 virtual host 에 지정된 base port 로 실행하면 된다.

3.15 JEUS 의 Class Loader Hierarchy

3.15.1 소개

JEUS 의 class loader 에는 JEUS 만의 특징을 가지고 있다. 크게 두가지 방식이 존재하는데, 4.x 대의 JAR classloading mode 에 해당하는 Shared classloader 와 JEUS 5 부터 추가된 Isolated classloader 가 있다. 기존의 DIR mode 에 해당하는 classloader 구조는 제거되었다. 하지만 DIR mode 의 장점이었던 directory 의 class file 을 이용한 deploy 는 JEUS 5 에서도 가능하다. 이는 16 장을 참고하기 바란다.

3.15.2 Shared classloader

우선 [그림 8]에서 소개하는 class loader 의 계층 구조를 보자

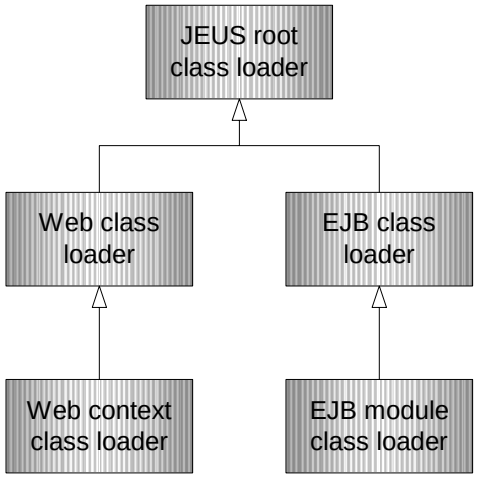

그림 8. Shared class loader 계층 구조.

[그림 8]의 설명은 다음과 같다.

1. **system (or root) class loader** 는 JEUS Manager 레벨에서 사용된다. 이 class loader 는 시스템 라이브러리와 JDBC 드라이버, user class 를 로딩할 때 사용된다.

2. **Web class loader** 는 웹 어플리케이션에서 사용되는 클래스를 로드 할 때 사용된다.

3. **Web context class loader** 는 Web Context 에 해당되는 Servlet 클래스를 로딩한다.

4. **EJB class loader** 는 EJB Engine 자체에서 필요한 클래스와 EJB 클래스를 로딩한다.

5. **EJB module class loader** 는 특정 EJB 모듈의 클래스와 라이브러리를 로딩한다.

위 구조의 shared classloader 에서는 Web classloader 에서 EJB classloader 로 class 를 요청하거나 EJB classloader 중 하나가 class 를 요청할 때에는 모든 EJB classloader 에게 class 를 요청한다. 이렇게 하나의 application 이 다른 application 의 classloader 를 공유하기 때문에 shared classloader 라고 한다. 이렇게 함으로써 4.x 대의 JEUS 에서는 EJB 를 사용하는 application 이 그 EJB 의 interface 나 class 들이 없어도 그 EJB 의 classloader 를 공유함으로써 필요한 class 들을 읽어올수 있다. 하지만 이런 점 때문에 한 application 의 class 가 다른 application 의 class 와 linking 이 이루어지므로 redeploy 를 할 경우에는 연관되어 있는 application 들을 모두 undeploy 한 후 deploy 해야 하는 문제가 발생한다. 그리고 class 들을 공유함으로써 여러가지 문제가 발생할수도 있다.

3.15.3 Isolated classloader

Isolated classloader 의 구조는 다음 그림 [그림 9]에서 소개하는 class loader 의 계층 구조를 보자

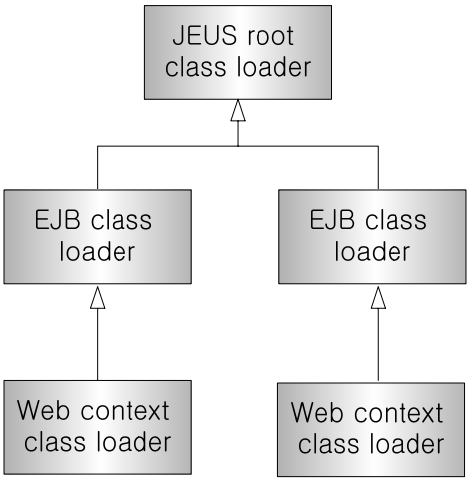

그림 9. Isolated class loader 계층 구조.

이 구조에서는 각 application 마다 application 내의 web module 의 classloader 가 그 application 의 EJB classloader 의 하위에 존재한다. 그리고 한 application 의 classloader 는 다른 application 의 classloader 에게 class 를 요청하지 않는다. J2EE 1.4 spec 은 한 application 이 자신이 사용하는 다른 application 의 interface class 들을 같이 packaging 하도록 규정하고 있고 이를 따른다면 다른 application 의 classloader 에게 interface class 를 요청하지 않아도 자신의 classloader 에서 이 class 들을 읽을수 있으므로 ClassNotFoundException 이 발생하지 않게 된다. 다만 class 를 공유할 수 없으므로 다른 EJB application 의 local interface 는 사용할 수 없게 된다. 이 경우에는 remote interface 만 사용가능하다. Class 를 공유하지 않기 때문에 class 공유로 생길수 있는 문제가 존재하지 않고 한 application 을 redeploy 할 때 다른 application 을 undeploy 할 필요가 없어진다. 이 classloader 구조를 제대로 사용하기 위해서는 연관된 application 끼리 EAR 로 묶어 하나의 application 으로 만드는 작업이 필요하다. 하나의 application 은 하나의 classloader delegation 구조를 가지기 때문에 서로 class 공유가 가능하므로 같은 EAR 내의 EJB 에 대해서는 local interface 사용이 가능하다.

3.16 본 매뉴얼의 자세한 정보

본 매뉴얼의 평면 구조는 매뉴얼 시작 부분의 “매뉴얼에 대해서”라는 장에서 소개했다. 이 구조를 바탕으로 [그림 10]에서는 매뉴얼을 계층 구조로 나타내 보았다. 이 계층 구조는 JEUS 의 구성 및 설정에 대응된다. 본 매뉴얼을 읽는 동안 [그림 10]를 잘 기억하기 바란다.

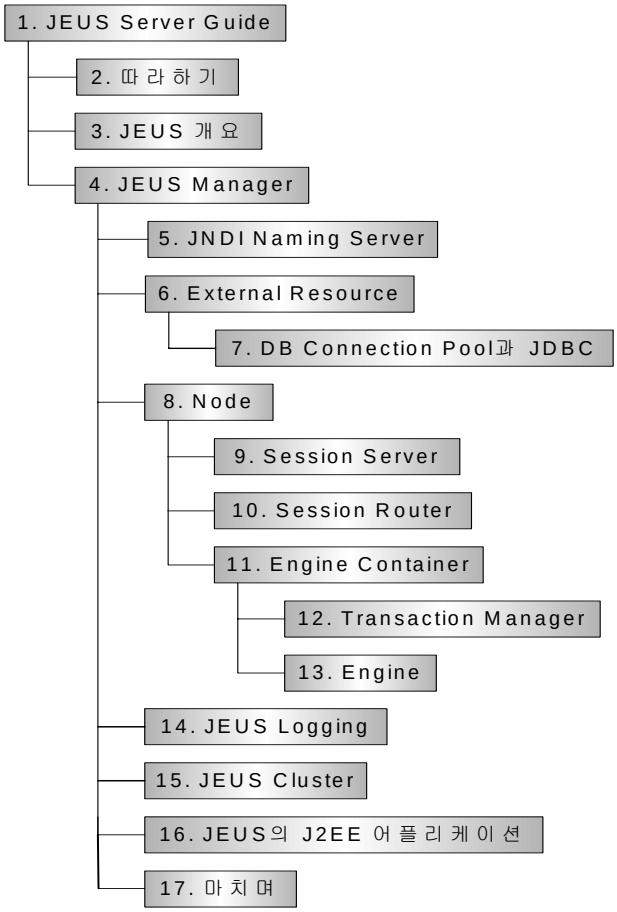

그림 10. 본 매뉴얼의 계층 구조

3.17 결론

이상으로 간단히 JEUS 를 살펴보았다. JEUS 가 엔터프라이즈 시스템에서 어떻게 작동하는지와 메인 컴포넌트들, 그리고 툴과 디렉토리 구조, 다양한 환경 설정 파일에 대해서 보았다.

본 매뉴얼의 나머지 부분에서는 JEUS 의 메인과 서브 컴포넌트들에 대해서 자세히 알아본다. 우선 JEUS Manager 라고 하는 핵심 컴포넌트부터 시작한다.

4 JEUS Manager

4.1 소개

JEUS Manager 는 하나의 JEUS 안에서 최상위 관리계층이다 [그림 11]. 실제 Manager 부분은 외부에서 볼 때에 JEUS 그 자체로 보일 수 있다.

앞으로 JEUS Manager 에 대해서 살펴보기로 하겠다.

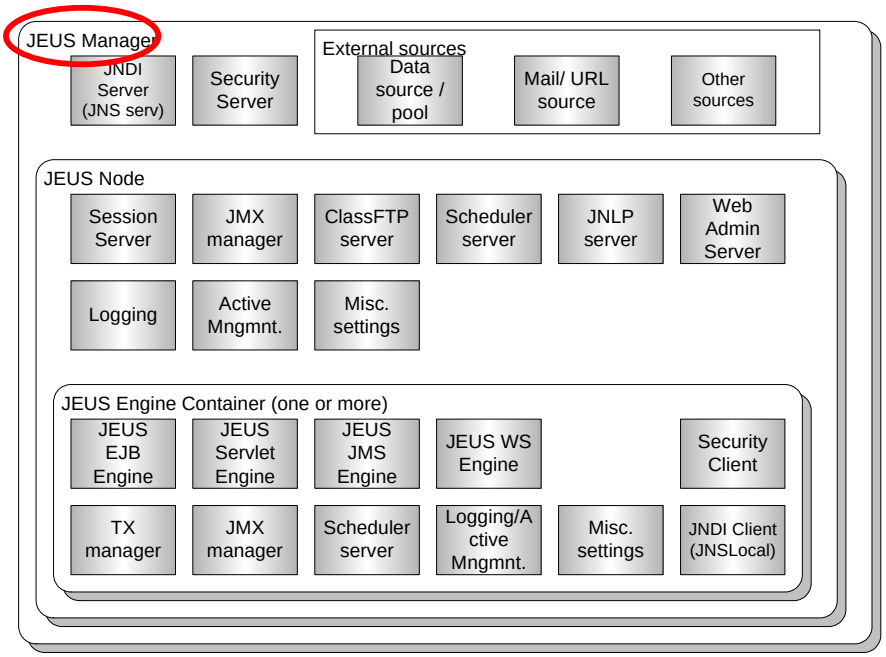

그림 11. JEUS Manager 로 둘러싸인 JEUS server 의 구조.

4.2 JEUS Manager 의 개요

4.2.1 소개

JEUS Manager 는 결국 JEUS WAS 의 하위 시스템과 컴포넌트를 광범위하게 관리하기 위하여 작성된 어플리케이션 이다.

JEUS Manager 에 포함되어 있는 주요 컴포넌트는, Beacon Receiver, Node, Naming Server, Security Manager 와 Resource 가 있다.

JEUS Manager 는 백업과 장애대책을 위해서 자신의 노드 또는 클러스터링에 포함된 다른 JEUS Manager 와 항상 통신할 수 있도록 대기하고 있다.

JEUS 의 컨트롤은 JEUS Manager 를 통해서 이루어 지며, jeusadmin tool (부록 B 참조) 이나 웹 관리자를 이용해서 할 수 있다.

4.2.2 주요 컴포넌트

주요 컴포넌트는 다음과 같다.

- Node
- Naming Server
- Security Server
- Resource

앞으로 나올 4 개 장에서 주요 컴포넌트에 관하여 자세히 알아보도록 하겠다.

4.2.3 Node

JEUS Manager 레벨에서 가장 중요한 설정들을 가지고 있는 부분이 Node 이다. Node 는 JEUS Server 를 구성하는 대부분의 요소들을 포함하는 논리적 구조이며, Engine Container 들과 그 안의 Engine 들을 포함하고 있다.

Node 와 JEUS Manager 는 같은 것은 아니지만 매우 밀접하게 관련되어 있다. 각 JEUS Manager 는 반드시 한 개의 Node 를 관리하고, 각 Node 는 반드시 한 개의 JEUS Manager 에 의해서 관리된다. 따라서 JEUS Manager 와 Node 는 1:1 관계를 가지고 있다.

Node 에 관해서는 8 장에서 더욱 자세하게 다루게 된다.

4.2.4 Naming Server

Naming server 는 JNDI API 를 구현한 것으로서 JEUS 안에 포함되어 있는 다양한 객체들을 Lookup 하여 찾아낼 수 있는 표준화된 방법을 제공하기 위하여 만들어 졌다.

JEUS Manager 들이 클러스터링으로 구성되어도 각각의 JEUS Manager 는 각자 독립적인 Naming Server 를 가지고 있다. 또한, 클러스터에 포함된 각각의 Naming Server 들은 어떤 종류의 객체가 어디에 있더라도 찾아올 수 있도록 구성된다.

JNDI 및 Naming Server 관련 내용은 5 장에서 자세히 다루게 된다.

4.2.5 Security Server

Naming Server 와 마찬가지로 Security Server (또는 Security Manager) 도 각각의 JEUS Manager 에 존재한다. Security Server 는 JEUS 의 보안관련 요청에 대한 응답 및 각종 인증작업을 수행한다.

예를 들어, JEUS Manager 를 제어하기 위한 사용자를 인증하는 것이다.

Security 와 Security server 관련 내용은 JEUS Security 안내서를 참조하기 바란다.

4.2.6 Resource

각 JEUS Manager 에는 여러가지 종류의 자원에 관하여 설정할 수 있다.

- **Data source:** 데이터베이스와의 커넥션과 풀링을 담당한다.
- **Mail source:** JEUS 에서 JEUS 자체 또는 그 밖의 어플리케이션을 이용하여 바깥으로 메일을 보내는 작업에 사용된다.
- **URL source:** 는 URL 자원을 JNDI export name 으로 연결하도록 한다.
- **External source:** IBM MQ 와 Tmax Server 와의 커넥션을 풀링할 수 있도록 해 준다.

자원관리의 주된 기능은 어떤 자원이 존재하고, JEUS 내부에서 모든 어플리케이션이 자원을 사용할 수 있도록 Naming Server 에 등록되어 있다는 사실을 명시적으로 알리는 것이다.

JEUS 는 자원관리를 위해서 3 가지 서비스(트랜잭션 서비스, 보안 서비스, 커넥션 풀링 서비스)를 제공하고 있다.

자원과 관련된 내용은 6 장에서 더욱 자세하게 다루게 되고, datasource 는 7 장에서 설명할 것이다.

4.2.7 JEUS Manager Life Cycle

JEUS Manager 의 주요한 상태는 다음 3 가지 이다.

1. **Not existing**: JEUS Manager 가 실행되지 않은 상태이다.
2. **Started**: JEUS Manager 가 ‘jeus’ 명령에 의해서 실행되어서 boot 명령을 대기한다. 그 외에는 아무런 동작도 하지 않는다.

3. **Booted**: Started 상태의 JEUS Manager 가 일단 boot 명령을 받게 되면 JEUS Manager 가 boot 된다. Boot 가 되면 JEUSMain.xml 파일을 읽어서 사전에 설정된 컴포넌트들(Engine 및 기타)을 구동 및 초기화 한다.

다음의 [그림 12] 는 JEUS Manager 의 상태변화를 간단하게 보여준다.

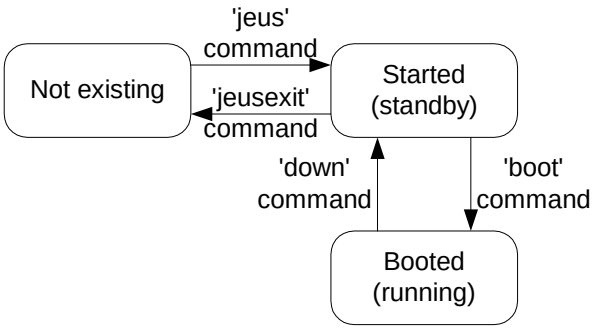

그림 12.JEUS Manager 의 3 가지 상태.

그림에서 'jeus' 명령은 jeus 라는 어플리케이션을 실행하는 명령이다. jeus 명령의 사용방법 및 설정에 관해서는 부록 A 에서 설명한다. 다른 명령어들(boot, down, jeusexit 등)은 부록 B 에서 'jeusadmin' 콘솔 툴과 함께 설명한다. 이 명령어들은 이번 장의 4.4 절에서도 설명하고 있다.

만일, 웹 관리자를 이용해서 JEUS 를 컨트롤 하려고 한다면, 먼저 jeus 명령을 실행하거나 JEUS 를 boot 시켜 놓아야 한다. 더 자세한 사항은 JEUS 웹 관리자 안내서를 참조 하기 바란다..

4.2.8 JEUS Manager Base Port

JEUS Manager 의 BASEPORT 는 jeusadmin 콘솔 툴 또는 웹 관리자와의 연결과 커맨드가 전달되는 기본 포트를 결정하게 된다.

BASEPORT 번호는 Naming Server 또는 security server 와 같이 JEUS Manager 의 주요 하위 컴포넌트를 위한 포트번호를 할당할 때 기본 번호가 된다.

4.2.9 JEUS Manager Clustering

JEUS Manager 는 백업이나 Fail-Over 기능을 위해서 다른 JEUS Manager 들과 쉽게 연결될 수 있도록 하고 있다. 이러한 기능을 *JEUS Manager clustering* 이 라고 한다.

JEUS 의 클러스터링에 관한 내용은 15 장에서 자세하게 다루도록 한다.

4.2.10 결론

지금까지 JEUS 의 핵심 컴포넌트인 JEUS Manager 에 관하여 설명하였다. 그리고 JEUS Manager 의 하위컴포넌트들, life cycle, 클러스터링에 관해서도 배웠다.

이제 JEUS Manager 의 설정방법에 관해서 배우도록 하겠다.

4.3 JEUS Manager 의 설정

4.3.1 소개

JEUS Manager 의 주 설정파일과 JEUS server 의 주 설정파일은 모두 *JEUSMain.xml* 이다. 이 파일은 JEUS 를 설치할 때, JEUS_HOME\config \<nodename> 디렉토리에 생성된다.

위의 경로에 포함되어 있는 "nodename" 이라는 속성은 JEUS Manager 가 동작할 컴퓨터의 실제 이름이다. 또한, 이 이름은 JEUS Manager 에 할당될 JEUS 노드의 이름이기도 하다. 이 내용에 대해서는 8 장에서 자세하게 다루고 있다.

JEUSMain.xml 파일의 최상위 XML 태그는 `<jeus-system>` 이다.

<<*JEUSMain.xml*>>

```
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    ... <!-- JEUS Server/Manager 설정은 여기에-->
</jeus-system>
```

참고: 이곳에서는 JEUSMain.xml 파일에 대하여 개략적인 수준 이상으로 설명하고 있으나 전반적인 자세한 내용은 부록 J 에 실려있는 예제를 참조하도록 한다. 본문의 설명과 부록의 내용이 일치하지 않는 경우에는 부록의 내용을 더 우선적으로 참고하도록 한다. 앞으로의 예에서 root element 의 xmlns=http://www.tmaxsoft.com/xml/ns/jeus 설정은 생략하겠다.

바로 다음 절에서 문서편집기 등을 이용해서 XML 을 직접 편집하는 방법으로 JEUSMain.xml 을 수동으로 설정하는 방법에 관하여 설명하고 있다. 웹 관리자를 이용하여 JEUSMain.xml 을 설정하는 방법은 JEUS 웹 관리자 안내서를 참조 하기 바란다.

4.3.2 JEUSMain.xml 을 통한 JEUS Manager 의 설정

이번 장의 서두에서도 언급했듯이 JEUS Manager 의 설정과 관련해서 앞으로 나올 각종 속성들은 모두 JEUSMain.xml 파일의 <jeus-system> 태그 안쪽에 설정되어야 한다.

- **Node.** 각각의 JEUSMain.xml 파일은 여러 개의 Node 에 관한 설정을 가지고 있을 수 있지만, JEUS Manager 는 그 중 단 한 개의 설정만을 따라서 동작한다. 이런 식으로 동작하는 이유는 15 장에서 설명하도록 한다. 단일 노드에 관한 설정방법은 8 장을 참조하도록 한다.
- **Naming Server**. 5 장 참조.
- **Security Manager**. JEUS Security Manager 참조
- **Resource.** 6 장 참조.

다음의 XML 예제는 JEUSMain.xml 파일의 내용 중 최상위 레벨의 XML 구조이다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    <node>
        ...<!– 노드 설정(8장 참조) -->
    </node>
    <node>
        ...<!–- 노드 설정(8장 참조) -->
    </node>
    <node>
        ...<!–- 노드 설정(8장 참조) -->
    </node>

    <naming-server>
        ...<!–-JNDI Server 설정(5장 참조) -->
    </naming-server>

    <security-manager>
        ...<!–- 보안관련 설정(JEUS Security 안내서 참조) -->
    </security-manager>

    <resource>
```

```
        ...<!– 자원관련설정(6장참조) -->
    </resource>
</jeus-system>
```

위의 예제를 보면, 실제 JEUS Manager 가 사용할 내용은 한 개뿐인데도 여러 개의 노드관련 설정이 들어있는 것을 볼 수 있다. 이중에서 실제 사용될 부분은 현재 컴퓨터의 호스트 이름과 동일한 이름을 가지고 있는 부분의 설정이다. 이점에 대해서는 15 장(클러스터링) 에서 자세하게 설명할 것이다.

위의 예제에는 노드관련 설정 이외에 하나의 JEUSMain.xml 파일 안에 한번씩만 설정할 수 있는 속성으로 Naming Server, Security Manager, Resource 등의 설정도 포함하고 있다. 이 태그들은 선택적인 것으로써, Security Server 나 Naming Server 관련 설정은 생략한다면 자동적으로 기본값으로 사용되도록 되어 있다.

4.3.3 jeus 실행 스크립트에 Java –D 파라미터 추가하기

jeus 명령은 JEUS Manager 를 실행하는 명령으로 알고 있다(물론 그 사이에 JEUSMain.xml 파일을 읽어 들인다). 이미 설정이 완료된 JEUSMain.xml 파일과는 별개로 'jeus' 를 수정하여 런타임에 필요한 파라미터를 제공할 수 있다. 여기에서 사용할 수 있는 명령의 종류와 용도에 관해서는 다음절 (4.4) 과 부록 A 에서 자세히 확인할 수 있다.

옵션 추가는 'jeus' 명령이 JVM 을 실행하여 JEUS Manager 가 동작하도록 하기 위해 작성된 스크립트라인 관계로 특정 위치(JEUS_HOME\bin\) 에서 실행 스크립트(jeus)를 열어서 편집하는 간단한 작업이 된다.

부록 A 에서는 소개한 "–D" 옵션의 파라미터들은 이 스크립트에서 반드시 사용되어야만 한다.(-classpath 다음에 적는다). 이 파라미터를 사용해서 JEUS Manager 와 JEUS 의 실행을 조정할 수 있다.

예제는 –D 파라미터를 이용해서 JEUS base port 또는 JEUS home 경로 등을 수정하는 방법이다.

4.3.4 결론

JEUS Manager 를 설정하는 방법에 관해서 살펴보았다.

다음 절에서는 JEUS Manager 를 어떻게 컨트롤 하는지 살펴보도록 하겠다 (시작, 종료 등).

4.4 JEUS Manager 컨트롤

4.4.1 소개

이번 장에서는 콘솔을 이용하여 JEUS Manager 를 boot, down, start, stop 하는 방법에 관해서 간략하게 살펴볼 것이다.

jeus 와 jeusadmin 콘솔 툴에 관한 자세한 사항은 부록 A 와 B 를, 웹 관리자를 이용한 JEUS Manager 컨트롤은 JEUS 웹 관리자 안내서를 참조하도록 한다.

4.4.2 JEUS Manager 의 시작과 부팅

앞서 살펴본 JEUS Manager 는 'jeus' 실행 스크립트에 의해서 일반적인 형태로 실행된다.

따라서, JEUS Manager 를 실행하여 대기상태가 되도록 하기 위해서는 반드시 JEUS_HOME\bin\ 경로가 시스템 패스에 설정되어 있어야 한다.

```
jeus
```

위 명령이 실행되면 콘솔에 몇 줄의 문자가 찍힌다. 성공적으로 실행되었다면 다른 콘솔에서 jeusadmin 명령을 실행해서 'boot' 명령으로 JEUS Manager 를 boot 시킨다. 다음의 예제는 jeusadmin 툴을 johan 이라는 노드에 접근하도록 실행하는 방법이다.

```
jeusadmin johan
```

사용자 ID 와 비밀번호를 입력한 뒤에 다음의 명령을 입력한다.

```
johan> boot
```

JEUS Manager 창에 booting 관련 정보들이 표시되고, 작업이 성공적으로 끝나면 jeusadmin 창에 다음의 메시지가 출력된다.

```
johan boot done
johan_default
```

jeusadmin 창의 마지막에 출력된 메시지는 "johan_default" 라는 Engine Container 가 동작하기 시작했다는 내용이며, 그것은 JEUS Manager 가 현재 정상적으로 동작하고 있다는 의미이다.

참고: 비록 하나 또는 몇몇 Engine Container 가 boot 되지 않았다고 하더라도 JEUS Manager 는 노드가 정상적으로 boot 되었다고 판단할 것이다.

4.4.3 JEUS Manager 의 down & exit

boot 상태인 JEUS Manager 를 down 시키려고 할 때에는 jeusadmin 에서 다음의 명령을 실행한다.

```
johan> down
```

다음 메시지는 down 명령이 성공적으로 수행되어 JEUS Manager 가 대기상태가 되었을 때 표시되는 메시지다.

```
johan down successful
```

JEUS Manager 를 완전히 종료(not existing)시키기 위해서는 jeusadmin 에서 다음의 명령을 수행한다.

```
johan> jeusexit
```

4.4.4 결론

이번 절에서 jeusadmin 콘솔을 이용하여 JEUS Manager 의 start, boot, down, exit 방법에 관해서 알아보았다.

4.5 결론

이것으로 JEUS Manager 와 관련된 설명을 마무리 한다. 이번 장에서 JEUSMain.xml 에 설정되는 Beacon Receiver, Checker, Node, Naming Server, Security Server 와 자원들에 관하여 확인해 보았다. 중요 컴포넌트에 관해서는 설명할 내용들이 많다. 그 내용들은 이 매뉴얼 전체에 걸쳐서 다룰 것이고, 그 중 다음 장에서는 Naming Server 에 관해서 설명할 것이다.

5 JNDI Naming Server

5.1 소개

Java Naming and Directory Interface™ (JNDI)은 Java 어플리케이션이 네트워크에서 객체를 찾고 가져올 수 있도록 하는 API 이다. 어플리케이션이 알아야 하는 것은, 찾아서 사용하려는 객체의 논리적인 이름 뿐이다. 사용자의 관점에서 볼 때는 이전의 엔터프라이즈 환경보다 손쉽게 다양한 객체를 사용할 수 있다.

JEUS JNDI 는 JNDI 1.2 API 와 호환되며, Sun Microsystems 에서 제안한 표준 JNDI API 를 지원한다. 그리고 엔터프라이즈 환경에 적합하도록 JNDI Service Provider Interface(SPI)도 제공한다. 즉 JNDI SPI 를 구현한 제품은 JEUS JNDI 트리의 객체를 사용할 수 있다는 것을 뜻한다.

본 매뉴얼은 JEUS JNDI 의 기본적인 개념과 용어, 그리고 어떻게 환경을 설정하는지와 어떻게 어플리케이션을 개발하는지에 대해서 설명한다.

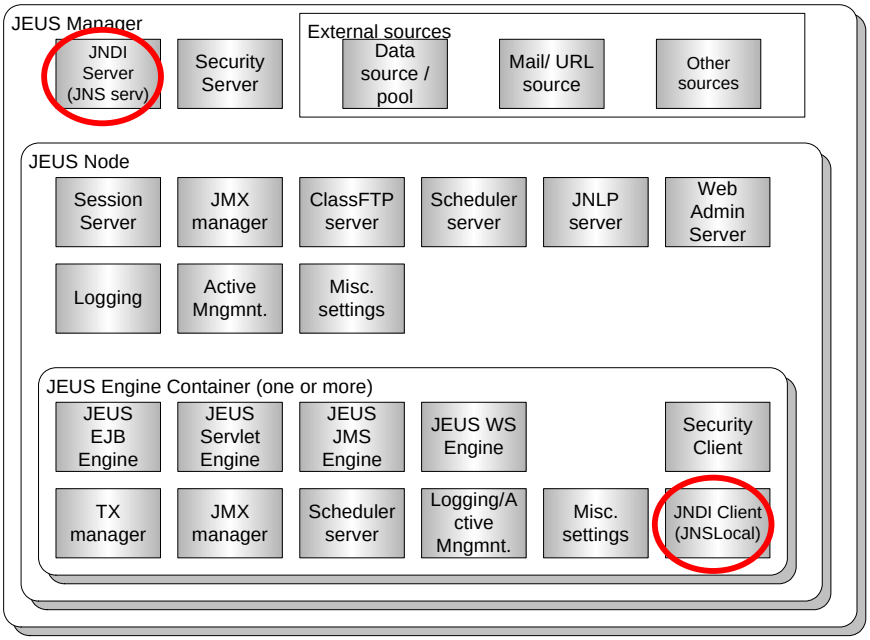

그림 13. JEUS 의 JNDI 컴포넌트

5.2 JEUS JNDI 의 개요

5.2.1 소개

JEUS JNDI 는 객체를 bind 하고 lookup 하는 고유의 아키텍처를 가지고 있다.

우선 JEUS JNDI 의 구조와 기본 개념에 대해서 알아보자.

5.2.2 기본 개념

JEUS 를 시작하면 JEUS JNDI 는 자동적으로 Naming Service 를 시작한다. 이 때, JNDI 트리는 JNS Naming Server(줄여서 JNSServer)와 JNSLocal Naming Server(줄여서 JNSLocal)와 함께 생성된다.

어플리케이션이 실행되는 로컬 JVM 에 JNSLocal 이 놓이게 되고, 어플리케이션이 직접적으로 접속해서 사용한다. JNSLocal 은 JNSServer 와 접속되어 있다. 객체의 bind 와 검색은 JNSLocal 에서 먼저하고, 다음으로 JNSServer 에서 진행된다.

JNDI 트리는 객체를 bind 하거나 lookup 할 때 액세스 되는데, bind 되는 객체의 이름으로 구성되어 있다. 여러 개의 가지로 이뤄진 큰 나무를 생각하면 된다. 이 나무처럼 JNDI 트리는 여러 개의 JNSServer(특히 클러스터링 환경일 때)를 가지고 있으며, 각각의 서버는 JNSLocal 을 가지고 있다. 모든 객체는 이 서버들을 통해서 JNDI 트리로 bind 되고 lookup 된다. JNSLocal 로 bind 하는 요청은 이 트리로 전달되어 bind 되며, 어플리케이션에서 lookup 된다.

JNDI 트리의 객체를 액세스할 때는 InitialContext 를 통해서만 가능하다. 그러므로 어플리케이션에서 JNDI 를 사용하려면 반드시 InitialContext 객체를 생성해야만 한다. 이 객체는 JNDI 트리에 접근해서 객체를 핸들링 할 수 있도록 해준다. 그리고 객체를 bind 하거나 lookup 할 뿐만 아니라, 객체의 목록을 가져올 수 있으며 제거할 수 있는 기능도 가지고 있다.

5.2.3 JEUS JNDI 아키텍처

JNDI 트리는 JNSServer 와 JNSLocal 로 구성되어 있다. 이 아키텍처가 어떻게 작동되는지 자세히 살펴보자

JNSServer 는 JEUS JNDI 아키텍처의 메인으로, JNDI 트리를 생성하고 관리한다. JNSServer 는 그 아래에 JNSLocal 을 둬서 관리한다. [그림 14]은 JNSServer 와 JNSLocal 의 관계를 나타낸다. 전체 JNDI 트리를 액세스하기 위해서는 JNSLocal 이 JNSServer 로 요청을 보내게 된다.

그리고, JNSServer 는 다른 노드의 JNSServer 와 연결되어서 클러스터링 또한 구성하고 있다.

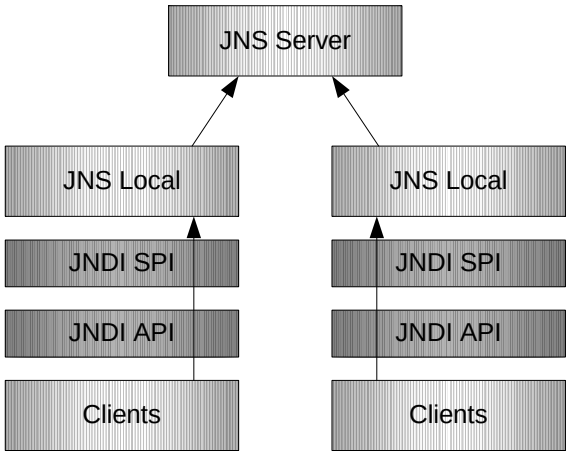

그림 14. JNS server 와 JNSLocal 의 관계

JNSLocal 은 JNS server 와 노드 내에서 상호작용을 하며, 클라이언트의 액세스 요청을 처리해준다. 클라이언트는 JNSServer 로 바로 액세스하는 것이 아니라 JNSLocal 을 통해서 객체를 bind 하고 lookup 한다.

그럼, JNSServer 와 JNSLocal 에 대해서 자세히 알아보자

5.2.4 JNS Naming Server

JNSServer 는 JNDI 트리를 관리하며, JNSLocal 이 JNDI 트리를 액세스할 수 있도록 해주는 독립적인 Naming Server 이다. JDNI 트리를 확장하기 위해서 여러 개의 JNSServer 를 연결할 수 있다. JNSServer 는 다른 노드의 것과 직접적으로 연결할 수 있기 때문이다. JEUS 5 에서는 JEUS Manager 가 시작되면 JNSServer 는 자동적으로 JNSLocal 의 접속을 기다린다.

5.2.5 JNSLocal Naming Server

JNSLocal 의 기본적인 기능은 그 자신이 JNSServer 로 접속하는 것이다. 그래서 클라이언트의 요청을 전송해서 JNSServer 의 결과를 다시 돌려주는 것이다.

각 JVM 에서는 하나의 JNSLocal 이 존재한다. 그래서 lookup 을 처리할 때 하나의 서버만 사용하면 되므로, 엔터프라이즈 환경에서 EJB 와 Servlet 을 사용할 때 효과적이다.

JNSLocal 의 중요 기능을 정리했다.

1. **JNDI Tree 액세스**: JNSLocal 은 JNSServer 에 접속해서, JEUS Manager 가 관리하는 JNDI 트리로 접속하는 방법을 제공한다. bind 되고 lookup 되는 객체는 전체 JNDI 트리에서 공유되거나, 클라이언트의 설정에 따라서 특정 클라이언트만 액세스할 수도 있다.

2. **lookup 된 객체의 캐시**: JNSLocal 은 자주 사용되는 객체를 캐시해서, 클라이언트가 더 빠르게 사용할 수 있도록 한다. JNSLocal 은 JNSServer 와 통신을 하면서 객체를 캐싱(caching)한다.

3. **JNSServer 와의 연결 관리**: JNSLocal 은 클라이언트의 요청을 받아서 JNSServer 로 전달하고, 그 결과를 받아서 돌려준다. 클라이언트가 있는 JVM 에 JNSLocal 이 존재하므로 별다른 I/O 없이 효율적으로 통신할 수 있다.

JNSLocal 에는 JNSServer 와 같은 노드이냐 아니면 원격지에 있느냐에 따라서 두 가지 종류가 있다.

Server-side JNSLocal

Server-side JNSLocal 은 Engine Container 에 있는 Engine 들의 EJB 빈과 Servlet, JSP 가 사용하는 모듈이다. Server-side JNSLocal 을 사용하려면 *java.naming.factory.initial* 프로퍼티의 값을 *jeus.jndi.JNSContextFactory* 로 세팅한다. 그러나 Engine Container 가 부팅될 때 이 값은 기본적으로 세팅되므로 별다르게 고려할 필요는 없다.

Client-side JNSLocal

일반적인 client container 의 클라이언트가 사용하는 것이 client-side JNSLocal 이다. *java.naming.factory.initial* 프로퍼티에 *jeus.jndi.JEUSContextFactory* 를 세팅해서 사용한다. Client-side JNSLocal 은 일정 시간 동안 통신이 없을 때 커넥션을 끊고, 다시 필요할 때 커넥션을 생성하는 등, 리소스를 효율적으로 관리한다.

5.2.6 JNDI Naming Clustering

JNDI 트리는 JNSServer 와 JNSLocal 간의 연결로 구성되어 있다. 이 구조는 정보의 변화가 있을 때 다른 컴포넌트로 전달될 수 있도록 하며, 또한 여러 개의 노드를 지원할 수 있도록 한다. [그림 15]는 클러스터링 환경에서 JNSServer 와 JNSLocal 간의 연결을 보여준다.

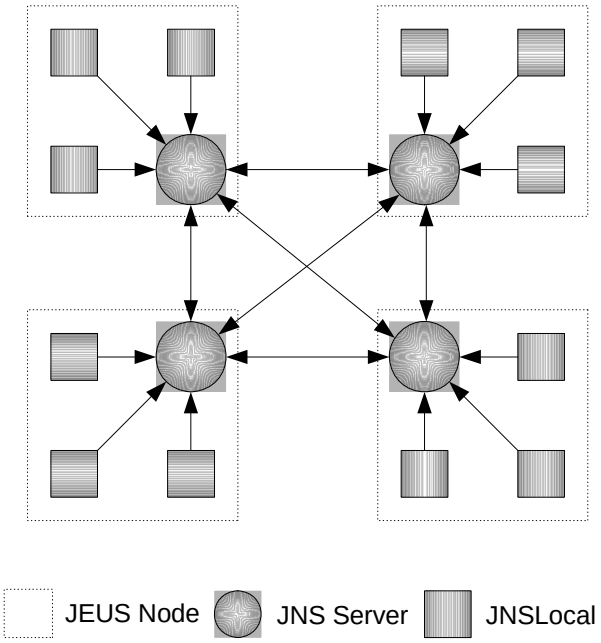

그림 15. 클러스터링 환경에서 JEUS JNDI 아키텍처

각 노드의 JEUS Manager 는 각각의 JNSServer 를 관리할 책임을 가지고 있다. 각 JNSServer 는 JEUS 시스템이 부팅될 때 시작되어서 다른 노드의 JNSServer 와 연결된다. 각 JEUS Engine 은 InitialContext 를 가져옴과 동시에 JNSLocal 이 JNSServer 로 연결된다.

클라이언트가 JNDI 트리의 객체를 lookup 할 때, JNSLocal 로 요청하고, 이어서 해당 노드의 JNSServer 로 요청이 가게 된다. 그리고 이에 대한 객체를 클라이언트가 받게 된다.

5.2.7 확장성

JNDI 트리는 마치 커다란 원형이라고 볼 수 있다. 각 JNSServer 는 다른 서버와 연결되어서 상호작용하기를 기다리고 있다. 기존의 클러스터 안으로 새로운 노드가 들어오면, 새로 들어온 노드의 JNSServer 는 시작할 때 이미 존재하는 다른 JNSServer 로 통보를 보내게 된다. 이때 각 JNSServer 는 자신의 데이터를 새로 들어온 JNSServer 로 전송하게 되며, 이렇게 해서 새로 들어온 JNSServer 에서도 기존에 bind 되어 있는 객체를 lookup 할 수 있게 된다.

이런 확장성 덕택에, 이상 작동으로 리부팅된 JNSServer 는 다른 JNSServer 로부터 JNDI 트리 정보를 받아 정상 상태로 동작할 수 있다.

5.2.8 JEUS 클러스터링 환경에서 원격으로 lookup

보통 JNDI lookup 은 자신이 포함된 JEUS/JNDI 클러스터링 영역 안에서 수행된다. 그러나 어플리케이션이 다른 곳에 있는 JEUS/JNDI 클러스터링을 lookup 해야 할 경우가 발생한다. 이런 특별한 경우를 위해서 특별한 lookup 기능을 제공한다. 이에 대한 내용은 5.5 절을 보길 바란다.

5.2.9 결론

이번 절은 JEUS JNDI 서비스에 대한 간단한 소개를 했다. JNDI 트리는 JNSServer 와 JNSLocal 의 관계를 나타내는 구조이다.

다음 절에서는 어떻게 JNDI 환경을 설정하는지 알아보자

5.3 JEUS JNDI Naming Server 설정

5.3.1 소개

앞서 말했듯이 JEUS Naming Server 는 JNSServer 와 JNSLocal 로 구성되어 있다. 이 둘은 서로 다른 설정을 가진다. JNSServer 에서는 JNSLocal 의 커넥션을 받아들이는 설정과 다른 JNSServer 와 접속하기 위한 설정이 필요하고, JNSLocal 은 JNSServer 와 접속하기 위한 것과 JNDI 트리의 반영을 위한 설정이 필요하다.

이번 절에서는 JNSServer 와 JNSLocal 을 설정하는 것에 대해서 다룬다. 또한 클러스터링 환경을 어떻게 세팅하는지에 대해서도 다룬다.

5.3.2 JNS Naming Server 설정

JNSServer 의 세팅은 JEUSMain.xml 에서 한다. 왜냐하면 JEUS Manager 가 실행될 때 JNSServer 가 실행되기 때문이다. 세팅하는 방법으로는 웹 관리자를 사용하는 것과 직접 XML 파일을 수정하는 것이 있다.

JEUSMain.xml 파일 중에서 관련 내용은 다음과 같다.

- Use-nio : Naming server 사이의 통신에 non blocking I/O 를 사용할지를 설정한다. 대량의 client 를 위해서는 non blocking I/O 가 필수적이다.
- **Export COS naming**: JEUS 가 COS Naming Server(tnameserver)로 작동할 것인지 아닌지를 정한다.
- **Resolution**: JEUS Manager 가 JNSServer 를 체크하는 시간 간격을 정한다.

- **Buffer size**: JNSServer 가 JNSLocal 이나 다른 JNSServer 와 통신하는 데 사용되는 버퍼의 크기를 정한다.
- **Backlog size**: 다른 Naming Server 의 접속을 받아들이는 한계인 back log 의 크기를 정한다.
- **Pooling**: **min**, **max**, **period**: 들어오는 요청을 처리하기 위해서 대기 중인 Worker Thread 를 정한다.

JEUSMain.xml 을 직접 수정할 때, <naming-server>의 <server> 태그는 다음과 같아야 한다.

<<*JEUSMain.xml*>>

```
<jeus-system>
     . . .
    <naming-server>
        <server>
            <use-nio>true</use-nio>
            <export-cos-naming>true</export-cos-naming>
            <resolution>30000</resolution>
            <buffer-size>32000</buffer-size>
            <backlog-size>100</backlog-size>
            <pooling>
                <min>1</min>
                <max>10</max>
                <period>30000</period>
            </pooling>
        </server>
    </naming-server>
</jeus-system>
```

이 파일을 수정한 다음, JEUS 를 재실행해야만 변경된 내용이 적용된다.

5.3.3 JNSLocal Naming Server 설정

JNSLocal 은 놓이는 위치에 따라서 설정하는 법이 다르다.

Server-side JNSLocal Naming Server 설정

Server-side JNSLocal 은 JNSLocal 이 JEUS Manager 와 같이 실행되는 것을 뜻한다. JEUSMain.xml 에 다음과 같은 태그를 사용해서 설정한다.

- **Resolution**: JEUS Manager 가 JNSLocal 의 리소스를 점검하는 시간 간격을 정한다.
- **Buffer size**: JNSLocal 이 JNSServer 와 통신하기 위해 사용되는 버퍼를 정한다.
- **Pooling min, max**: 들어오는 요청을 처리할 Worker Thread 를 정한다.

JEUSMain.xml 을 직접 수정할 때는 <naming-server>의 <local>태그를 사용해서 다음과 같이 한다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
    <naming-server>
        . . .
        <local>
            <resolution>30000</resolution>
            <buffer-size>32000</buffer-size>
            <pooling>
                <min>1</min>
                <max>10</max>
                <period>30000</period>
            </pooling>
        </local>
    </naming-server>
    . . .
</jeus-system>
```

설정 파일을 수정한 후에는 JEUS 를 재시작시켜야 변경 내용이 적용된다.

Client-side JNSLocal Naming Server 설정

Client-side JNSLocal 은 JVM 이 서로 다른 JNSServer 를 액세스한다. 그래서 JNSServer 와 연결되어서 JNDI 트리의 내용을 반영하는 쓰레드가 존재한다. JEUS JNDI 는 쓰레드 풀로 쓰레드를 관리한다. 이 쓰레드 풀은 JEUS 프로퍼티를 사용해서 설정하며, 기본값을 사용해도 무방하다.

Client-side JNSLocal 의 프로퍼티를 설정하려면 JVM 의 시스템 프로퍼티를 사용하거나 InitialContext 의 해시 테이블을 사용해야 한다. Client-side JNSLocal 의 프로퍼티는 다음과 같은 것이 있다.

- JEUSContext.RESOLUTION
- JEUSContext.THREAD_POOLING
- JEUSContext.CONNECT_TIMEOUT
- JEUSContext.CONNECTION_DURATION

이 프로퍼티에 대한 자세한 내용을 보려면 부록 H 를 참조한다.

참고: 만약 EJB 와 Servlet/JSP 같은 서버 측 객체에서 JNDI 를 사용한다면 이런 프로퍼티를 설정할 필요가 없다. 왜냐하면 JEUS Manager 에 의해서 server-side JNSLocal 을 사용해서 bind 나 lookup 되기 때문이다.

5.3.4 결론

지금까지 어떻게 JNSServer 와 JNSLocal 을 설정하는지 보았다. 다음 절에서는 클러스터링 환경에서 bind 옵션을 어떻게 설정할 것인지 본다.

5.4 클러스터링 환경에서 JEUS JNDI 설정

5.4.1 소개

현재 JEUS 클러스터링 환경이 구성되어 있다면 JNDI 를 클러스터링 하기 위해서 별다르게 할 것이 없다. JEUSMain.xml 을 사용해서 클러스터링 하는 것에 대해서는 15 장을 보기 바란다.

만약 JEUS 노드가 클러스터링 된다면, 각 JNSServer 도 자동적으로 클러스터링 환경을 구성한다. JEUS Manager 가 클러스터 내의 다른 JNSServer 로 접속시켜준다. 이런 것은 JEUSMain.xml 에 다른 노드의 정보를 넣어두기만 하면 된다.

JNSLocal 에 대해서 보자. JNSLocal 은 각각의 JNSServer 로 접속하고, 마치 클러스터링 환경이 아닌 것처럼 작동한다. 그러나 JNSLocal 이 client-side 일 경우에는 `Context.PROVIDER_URL` 에 접속하려는 JNSServer 를 세팅 한다. 기본적으로 JNSLocal 은 로컬 JNSServer 로 접속한다.

5.4.2 Binding Option 설정

클러스터에 있는 클라이언트는 JNDI 트리에 bind 되어 있는 어떤 객체든 lookup 할 수 있다. 만약 클라이언트가 JNSLocal 에 객체를 bind 한다면 곧 JNSServer 를 통해서 모든 노드에서 그것을 공유하게 된다. 또한 데이터 삭제나 변경이 발생하면 곧 각 노드의 JNSLocal 에도 반영된다.

JEUS JNDI 에서는 객체의 성격에 따라서 bind 하는 3 가지 옵션을 제공한다. 어떤 객체는 클러스터 전체에 공유되고, 또 어떤 객체는 자신의 노드에서만 존재한다거나, 또 어떤 것은 자신의 JVM 내에서만 보여야 할 경우가 있다.

클라이언트에서는 이 세 가지 옵션을 사용해서 객체를 JNDI 에 bind 시킬 수 있다. 이들 기능을 사용하려면 조금 후에 소개할 프로퍼티를 사용해야 하는데, 기본적으로 세 가지 방법이 있다.

1. `InitialContext` 의 해시 테이블 사용
2. JVM 옵션 '-D' 사용
3. Context 클래스의 addToEnvironment() 메소드 사용

세가지 방법을 사용한 예제를 보자. REPLICATE_BINDINGS 와 CACHE_BINDINGS 옵션을 사용하고, LOCAL_BINDING 을 사용하지 않으려면 다음 코드와 같이 한다.

```
ctx.addToEnvironment(JEUSContext.REPLICATE_BINDINGS, "true");
ctx.addToEnvironment(JEUSContext.CACHE_BINDINGS, "true");
ctx.addToEnvironment(JEUSContext.LOCAL_BINDINGS, "false");
ctx.bind("name", object);
```

이들 프로퍼티가 어떻게 작동하는지 자세히 살펴보자.

REPLICATE_BINDINGS

JNDI 트리 전체로 객체를 사용하려면 JNSLocal 은 객체를 JNSServer 로 전송해야 한다. 그러면 JNSServer 는 연결되어 있는 모든 JNSServer 로 그 객체를 넘기게 된다. REPLICATE_BINDINGS 가 true 이면, 객체는 모든 JNSServer 로 퍼지게 된다. 그러나 false 이면 로컬 노드의 JNSServer 내에서만 유지된다. [그림 16]는 객체가 복제되는 예를 보여준다.

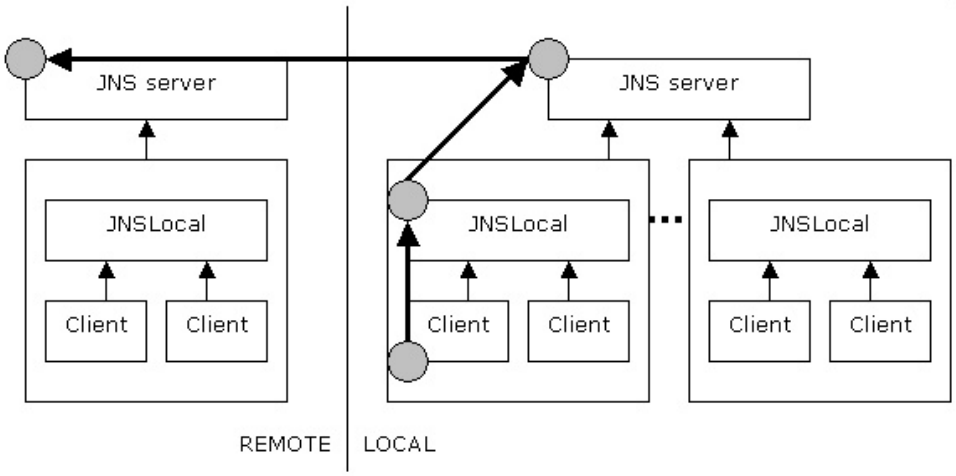

그림 16. Replicate Bind

CACHE_BINDINGS

클라이언트가 처음으로 객체를 lookup 하면, JNSLocal 은 JNSServer 로부터 객체를 lookup 해 온다. CACHE_BINDINGS 가 true 이면 lookup 된 객체는 저장소에 캐싱하게 된다. 이후부터는 JNSLocal 는 JNSServer 로부터 개체를 lookup 해오는 것 대신에 캐시 된 객체를 사용하게 된다. [그림 17]이 캐시 된 객체를 넘겨주는 것을 보여준다.

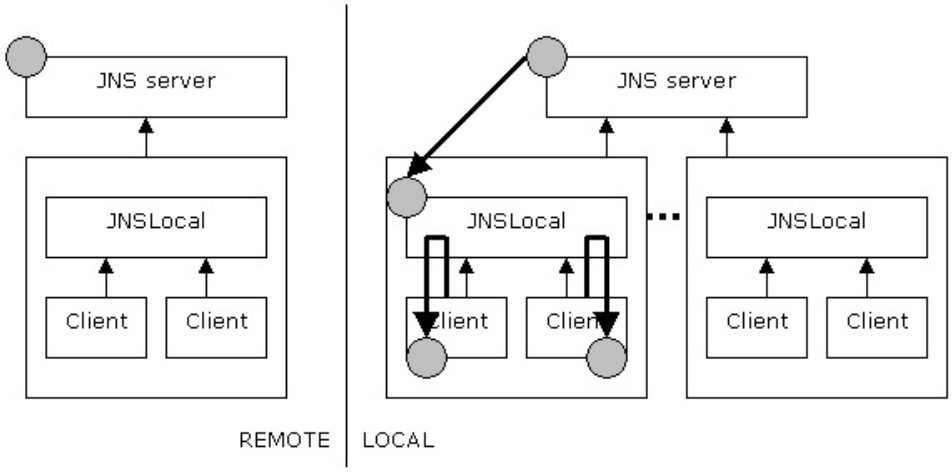

그림 17. Cache Bind

LOCAL_BINDINGS

객체를 JNDI 트리의 모든 곳에서 보이길 원하지 않으면 LOCAL_BINDINGS 를 사용한다. 이 값을 true 로 하면 JNSLocal 에만 bind 되고, 클라이언트의 요청은 JVM 내에서만 처리된다. [그림 18]은 객체가 JNSLocal 에서만 bind 되고 JNSServer 없이 lookup 되는 것을 보여준다.

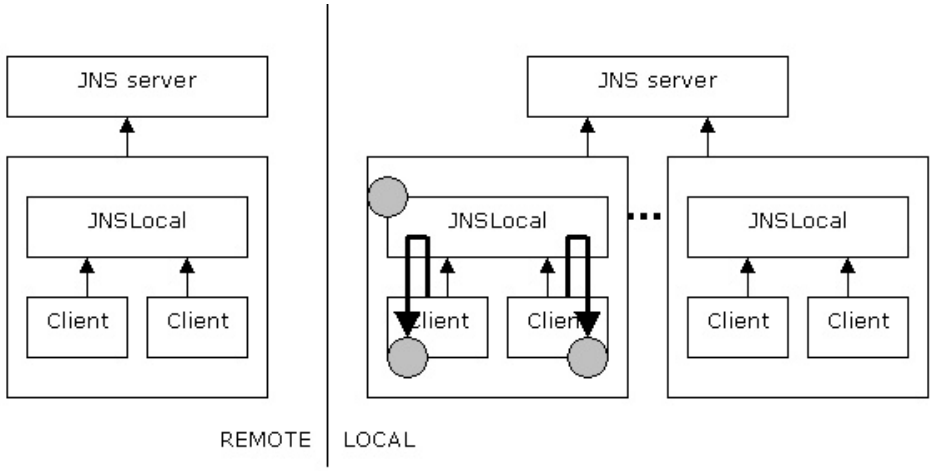

그림 18. Local Bind

5.4.3 결론

지금까지 어떻게 클러스터링 환경에서 JNS 를 세팅 하는지 보았다. 다음 절에서는 JEUS JNDI 를 사용하는 프로그래밍에 대해서 다룬다.

5.5 JEUS JNDI 프로그래밍

5.5.1 소개

Java 클라이언트는 `InitialContext` 를 사용해서 JNDI 트리를 액세스한다. `InitialContext` 에 사용되는 프로퍼티는 `InitialContext` 자체의 프로퍼티와 JEUS 용 프로퍼티가 있다.

Java 클라이언트는 JEUS JNDI 서비스를 사용하기 위해서 다음과 같은 절차를 따른다.

1. JEUS 환경 셋업
2. `InitialContext` 를 위한 프로퍼티 셋업
3. `Context` 를 사용해서 named object lookup
4. named object 을 사용해서 객체의 레퍼런스 취득
5. 커넥션 닫기

아래는 각 단계별 자세한 설명이다.

5.5.2 JEUS 환경 설정

JEUS JNDI 를 사용하려면 JEUS 가 설치되어 있어야 한다. JNSServer 는 JEUS Manager 가 boot 될 때 시작된다.

JEUS_CLIENT 가 JEUS 클라이언트 모듈의 패스라고 한다면, 그 패스를 JNDI 서비스에서 필요한 클래스를 사용할 수 있도록 classpath 에 넣어준다.

```
-classpath %JEUS_CLIENT%;.
```

5.5.3 InitialContext 를 위한 JNDI 환경 프로퍼티 셋업

Java 클라이언트가 JEUS JNDI 서비스를 사용하기 전에 우선 InitialContext 의 환경 프로퍼티를 세팅한다. 이것에는 두 가지 방법이 있다. 하나는 해시 테이블을 만들어서 InitialContext 의 생성자에게 넘기는 것이고, 또 하나는 JVM 의 '-D' 옵션을 사용하는 방법이다. 후자가 전자보다 우선한다.

JEUS JNDI 의 InitialContext 를 생성하기 위해서 세팅해야 하는 프로퍼티는 다음과 같다.

- Context.INITIAL_CONTEXT_FACTORY (required)
- Context.URL_PKG_PREFIXES
- Context.PROVIDER_URL
- Context.SECURITY_PRINCIPAL
- Context.SECURITY_CREDENTIALS

이들 프로퍼티에 대한 자세한 사항은 부록 H 를 참조한다.

이 프로퍼티들을 해시 테이블에 넣어서 InitialContext 를 생성할 때 사용한다. 만약 서버 측 객체 내(EJB 나 Servlet/JSP)에서만 InitialContext 를 사용한다면 별다른 세팅 없이 디폴트로 세팅 된 InitialContext 를 사용하면 된다.

클라이언트 프로그램에서는 다음과 같이 한다.

```
Context ctx = null;
Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY,
"jeus.jndi.JEUSContextFactory");
ht.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
ht.put(Context.PROVIDER_URL, "<hostname>");
ht.put(Context.SECURITY_PRINCIPAL, "<username>");
```

```
ht.put(Context.SECURITY_CREDENTIALS, "<password>");
try {
       ctx = new InitialContext(ht);
       // use the context
} catch (NamingException ne) {
       // fail to get an InitialContext
} finally {
       try{
              ctx.close();
       catch (Exception e) {
              // an error occurred
       }
}
```

클러스터링 환경에서 JNDI 를 사용할 때는, JNDI 트리를 구성하고 JNSLocal 의 내부적인 작동을 효과적으로 관리하기 위해서 `jeus.jndi.JEUSContext` 의 추가적인 환경 프로퍼티를 사용할 수 있다. 자세한 것은 부록 H 를 참조한다.

Hashtable 을 사용해서 Context 생성하기

앞서 설명했듯이 `InitialContext` 의 환경 프로퍼티를 세팅할 때 해시테이블(`java.util.Hashtable`)을 사용할 수 있다. 프로퍼티를 세팅하려면, 해시테이블의 키로 프로퍼티 이름을 넣고, 값으로 프로퍼티의 데이터를 넣은 후에 `InitialContext` 생성자의 파라미터로 넣어준다.

Security Context 생성하기

JEUS Security Domain 에 있는 사용자를 실행 쓰레드에 적용시키기 위해서 몇 가지 보안 관련 환경 프로퍼티를 사용할 수 있다. 사용자명 프로퍼티로는 `java.naming.security.principal` 이 사용되고, 패스워드 프로퍼티로는 `java.naming.security.credentials` 가 사용된다. `InitialContext` 가 생성될 때 Security Context 는 이 프로퍼티로 구성된다.

인증이 성공하면 인증된 사용자의 정보가 실행 쓰레드에 세팅되며, JEUS Security Manager 의 관리하에 있는 리소스를 액세스할 수 있다. 만약 인증에 실패하면 쓰레드는 guest 사용자로 인식된다.

만약 Security Context 가 설정되어 있다고 하더라도 `InitialContext` 가 생성되기 전에 Jeus Security API 를 사용해서 login 을 해 두었다면 Security Context 는 무시된다. 즉, 이미 login 된 subject 를 사용해서 JNDI 통신을 수행한다.

사용자 정보 세팅에 관한 더 자세한 내용과 JEUS Security Service 에 대한 내용은 JEUS Security 안내서를 참조한다.

5.5.4 Context 를 사용한 Named Object 의 lookup

JNDI 트리에 bind 된 객체는 JNDI Context 의 `lookup()` 메소드를 사용해서 찾는다. lookup 하려는 객체의 이름이 "`ejbAppHome1`"라면, 코드는 다음과 같다.

```
try {
       Context ctx = new InitialContext();
       AppHome appHome1 = (AppHome)ctx.lookup("ejbAppHome1");
       // successfully got the object
} catch (NameNotFoundException ex) {
       // no such binding exists
} catch (NamingException e) {
       // an error occurred
}
```

5.5.5 Named Object 를 사용해서 객체의 레퍼런스 가져오기

EJB 클라이언트에서 EJB 홈 객체는 `create()` 메소드를 사용해서 EJB 리모트 객체의 레퍼런스를 가져온다. 이처럼 메소드를 실행해서 사용하려는 객체의 레퍼런스를 가져오고, 그 객체 레퍼런스의 메소드를 실행시킨다.

```
AppBean bean = appHome1.create();
Bean.service();
```

5.5.6 Context 닫기

Context 를 사용한 다음에는 `close()`메소드를 실행해서 Context 를 닫아준다. 다음과 같이 한다.

```
try {
    cx.close();
} catch(Exception e) {
    // an error occurred
}
```

5.5.7 JEUS 클러스터링 환경에서 원격으로 lookup 실행하기

"소개하기"에서 간단히 언급했듯이, 특별한 lookup 구문이 제공된다. 이 구문은 자신이 속한 JEUS/JNDI 클러스터링의 영역을 벗어나 원격지의 JEUS/JNDI 클러스터링에 있는 객체를 lookup 할 때 사용한다. 사용법은 JNDI Context 문자열 안에 다음과 같은 구문을 넣어준다.

```
jh:<remote JEUS host name>:<remote JEUS base port>/<export name>
```

("jh" = JEUS host)

예:

```
try {
Context ctx = new InitialContext();
AppHome appHome1
    = (AppHome)ctx.lookup("jh:dev:9736/ejbAppHome1");
// successfully got the object
} catch (NameNotFoundException ex) {
// no such binding exists
} catch (NamingException e) {
// an error occurred
}
```

5.5.8 결론

지금까지 JEUS JNDI 를 사용하는 프로그래밍을 어떻게 하는지 보았다. 제일 먼저 JNDI 환경을 셋업하고, 그 다음으로 `InitialContext` 를 사용해서 객체를 lookup 한 다음, 객체의 레퍼런스를 가져온다. 마지막으로 `InitialContext` 를 사용한 다음에는 반드시 close 시켜준다.

이외에도 원격 클러스터링 lookup 기능도 알아 보았다

5.6 결론

이것으로 JEUS JNDI 에 대한 설명이 모두 끝났다. JEUS JNDI 를 사용하면 엔터프라이즈 환경에서 다양한 객체를 손쉽게 액세스할 수 있다는 것을 보았다.

JEUS JNDI 서비스는 JEUS 시스템 전반적으로 두루 사용된다. EJB 와 Servlet/JSP, JMS, JDBC 등을 사용할 때마다 만나게 된다.

JEUS JDNI 서비스가 어떻게 동작하는지 예제를 보고 싶으면, JEUS 가 설치된 곳의 samples 디렉토리에 있는 소스를 참조하기 바란다.

6 External Resource

6.1 소개

대개 DB 나 legacy EIS 와 같이 JNDI 에 등록함으로써 동적으로 JEUS 에 추가할 수 있는 외부 데이터 저장소나 서비스들을 resource 라고 한다. resource 는 비즈니스 데이터와 백엔드 서비스로 완벽한 JEUS WAS 솔루션을 형성한다.

현재 JEUS 에서는 네 종류의 resource 들을 지원하고 이들은 모두 JEUS Manager 에서 최상위 레벨에 설정한다. [그림 19]

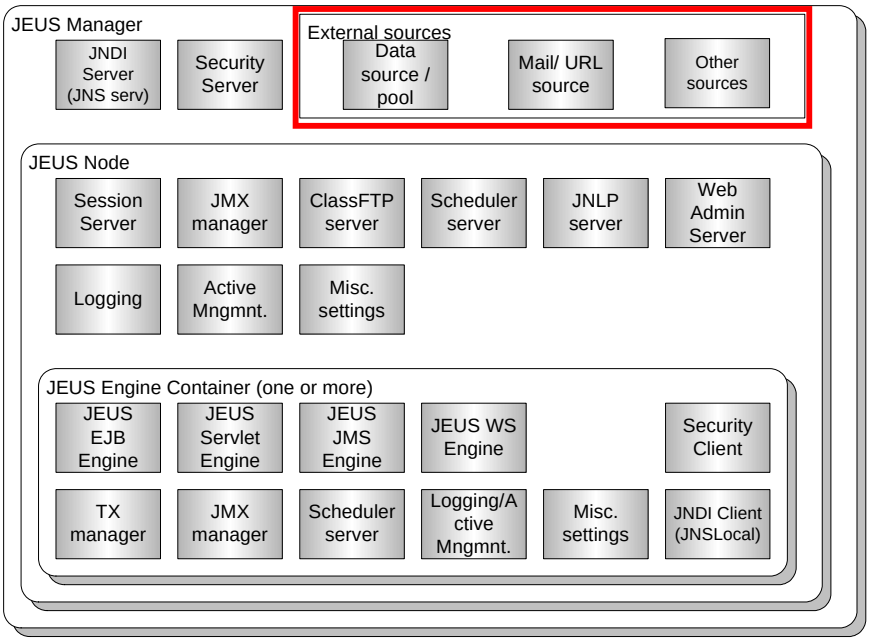

그림 19. JEUS architecture 에서의 resource

이 장에서 “resource”와 “source”라는 단어를 같은 의미로 사용한다.

6.2 Resource 의 개요

6.2.1 소개

다음은 각각의 JEUS Manager 에 설정할 수 있는 네 종류의 resource 이다.

1. Data source
2. Mail source
3. URL source
4. Other source (WebT, IBM MQ 그리고 Sonic MQ)

각각의 내용은 다음과 같다.

6.2.2 Data Source (Database Source)

데이터소스는 JDBC 호환 DB 이며, 데이터를 저장할 때 주로 사용된다. J2EE 에서는 주로 EJB 엔티티 빈의 필드를 저장할 때 사용되지만, JEUS 를 사용하는 어플리케이션에서 직접 사용할 수도 있다.

JDBC Data Source 는 클라이언트에서 직접적으로 접근 할 수 있는데 이런 경우에는 특별히 JEUS 에 설정을 하지 않아도 된다. 그러나 Data Source 를 설정하면 JNDI 를 이용하거나 JDBC connection pooling, 그리고 트랜잭션 관리를 할 수 있으므로 더욱 편리하고 유용하게 사용할 수 있다. Data Source 의 설정에 대해서는 본 매뉴얼 7 장에 설명되어 있다.

JEUS WAS 는 JDBC 3.0 스펙에 있는 DB connection pooling 을 지원한다.

6.2.3 Mail Source

Mail Source 는 SMTP 와 같은 메일 프로토콜을 지원하는 것을 사용하여 클라이언트 어플리케이션으로부터 e-mail 을 보내는데 사용한다. JEUS 에서는 JNDI export name 에 email host 의 정보를 bind 하고, 클라이언트에서 간접적으로 접근하여 host 를 사용하도록 한다.

Mail source 를 사용하면 소스 코드를 수정하지 않고도 클라이언트의 메일 전송을 바꿀 수 있다.

JNDI 를 lookup 하면 javax.mail.Session 타입의 mail source 를 가져온다.

6.2.4 URL Source

URL Source 는 URL 을 JNDI name 에 bind 하는데 사용된다. 이렇게 하면 클라이언트 어플리케이션에서 URL 에 간접적으로 접근할 수 있다.

URL Source 설정에서 URL 을 수정함으로써 클라이언트 소스의 수정 없이 그대로 사용할 수 있다.

JNDI 를 lookup 하면 `java.net.URL` 타입의 URL source 를 가져온다.

6.2.5 Other Source

앞서 소개한 것 이외의 Source 는 JEUS 와 연결 할 수 있는 비정규화된 resource 들을 말한다.

일반적으로 JEUS 에 설정할 수 있는 것으로는 **IBM MQ** 제품, **Tmax server** 그리고 **Sonic MQ** 제품 등 세가지가 있다. IBM MQ 는 IBM 에서 Message Queue 를 구현한 제품이고, Tmax 는 Tmax Soft 사의 TP 모니터이다. 그리고 Sonic MQ 는 Sonic Software 사(www.sonicsoftware.com)에서 messaging 을 구현한 제품이다.

IBM MQ 는 JEUS 의 JMS 와 통합되어있다. 그러므로 J2EE Connector 로 IBM MQ 를 사용하는 것은 피하고, JMS API 를 사용한다. 이는 SONIC MQ 제품도 마찬가지이다.

참고 : 이 resource 들은 JEUS 에 설정하지 않아도 Java API 를 통해서 직접적으로 액세스 할 수 있다. 그러나 JEUS Transaction Manager 에서 이 source 들을 관리하려면 설정을 해야 한다. (12 장 참조)

6.2.6 결론

JEUS 와 통합될 수 있는 resource 는 Data Source, Mail Source, URL Source, Other Source 등 네 가지이다.

다음 절에서는 JEUS 에서 resource 의 설정법을 알아본다.

6.3 Resource 설정

6.3.1 소개

이번 절에서는 Mail Source, URL Source, Other Source 를 JEUSMain.xml 에 설정하는 방법을 설명한다. Data Source 의 설정은 7 장에 기술되어 있다.

하나의 `<resource>` 태그 안에는 4 종류의 source-type 을 설정할 수 있는데, 이는 JEUSMain.xml 파일의 최상위 element 인 `<jeus-system>` 아래에 존재한다.

예제:

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
    <resource>
```

```
        <data-source>
            ...
        </data-source>
        <mail-source>
            ...
        </mail-source>
        <url-source>
            ...
        </url-source>
        <external-source>
            ...<!-- "Other" sources go here -->
        </external-source>
    </resource>
</jeus-system>
```

JEUS Manager 에 source 를 추가한다.

아래는 resource 종류에 따른 설정법이다.

6.3.2 Data Source 설정

이 내용은 7 장에 기술되어 있다.

6.3.3 Mail Source 설정

Mail Source 는 JEUSMain.xml 파일의 <resource> 태그 안에 있는 <mail-source> 에 설정한다. 각각의 SMTP 서버 마다 <mail-source>의 하위 element 인 <mail-entry>에 클라이언트에서 사용하는 JNDI export name 을 설정해야 한다.

<mail-entry> 태그의 설정은 다음과 같다.

- **export name :** SMTP 서비스의 이름이다. 클라이언트는 서비스에 등록되어 있는 이 이름을 간접적으로 사용한다. 이는 클라이언트에서 실질적으로 사용하는 이름으로 클라이언트 디스크립터에서 export name 으로 bind 한 것이다. 이 이름은 java.mail.Session 객체로 bind 된다.

- **properties :** 하나이상 설정할 수 있으며 SMTP 서비스에 접근하고 사용하는 것에 대한 정보를 설정한다. 이는 name 과 value 로 나타내어진다. name 은 JavaMail 1.2 스펙에 정의된 JavaMail 환경 프로퍼티도 사용이 가능하다. 예를 들어, “mail.host”, “mail.from”, “mail.user” and “mail.transport.protocol” 등을 사용할 수 있다.

다음은 Mail Source 에 대한 간단 샘플이다.

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        ...
        <mail-source>
            <mail-entry>
                <export-name>MY_SMTP_HOST</export-name>
                <mail-property>
                    <name>mail.host</name>
                    <value>mail.foo.com</value>
                </mail-property>
                <mail-property>
                    <name>mail.user</name>
                    <value>peter</value>
                </mail-property>
            </mail-entry>
            <mail-entry>
               ...
            </mail-entry>
        </mail-source>
        ...
    </resource>
</jeus-system>
```

Mail Source 의 사용법에 대해서는 6.4.3 절에서 다룬다.

6.3.4 URL Source 설정

URL Source 는 URL 과 JNDI export name 을 mapping 시켜서 간단히 설정할 수 있다. <url-source>의 하위 element 인 <url-entry> 태그를 추가해서 bind 한다.

다음의 두 가지 XML element 는 각 <url-entry> 아래에 정의 된다.

- **export name :** JNDI namespace 에 bind 된 URL Source 객체의 JNDI name 이다. 클라이언트는 URL 을 찾을 때 간접적으로 이 이름을 사용한다. 이는 java.net.URL 객체로 bind 된다.
- **url :** 불려지는 URL 주소이다.

예제:

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        ...
        <url-source>
            <url-entry>
                <export-name>
                    PRIMARY_URL
                </export-name>
                <url>
                    http://www.foo.com
                </url>
            </url-entry>
            <url-entry>
                <export-name>
                    SECONDARY_URL
                </export-name>
                <url>
                    http://www.bar.com
                </url>
            </url-entry>
        </url-source>
        ...
    </resource>
</jeus-system>
```

위의 예제에서 JNDI name 은 PRIMARY_URL 과 SECONDARY_URL 이고 각각에 bind 되는 URL 은 www.foo.com 과 www.bar.com 이다.

URL Source 의 사용법에 대해서는 6.4.4 절에서 다룬다.

6.3.5 External (“other”) Source 설정

External(Other) Source 는 IBM MQ, Tmax TP monitor, 그리고 Sonic MQ 등이 있다. IBM MQ 와 Sonic MQ 를 설정하기 위해서는 각각 IBM MQ 매뉴얼과 Sonic MQ 매뉴얼을 참조하기 바란다. Tmax TP monitor 를 설정하기 위해서는 Tmax 매뉴얼을 참조한다.

다음은 External(Other) Source 에 대한 설정 예제이다.

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        ...
        <external-source>
            <IBMMQ>
                <IBM-QCF>
                    <export-name>IBMMQ</export-name>
                    <qmanager>. . .</qmanager>
                    <description>. . .</description>
                    <transport>. . .</transport>
                    <client-id>. . .</client-id>
                    <port>. . .</port>
                    <host-name>. . .</host-name>
                    <channel>. . .</channel>
                    <ccs-id>. . .</ccs-id>
                    <rec-exit>. . .</rec-exit>
                    <rec-exit-init>. . .</rec-exit-init>
                    <sec-exit>. . .</sec-exit>
                    <sec-exit-init>. . .</sec-exit-init>
                    <send-exit>. . .</send-exit>
                    <send-exit-init>. . .</send-exit-init>
                    <temp-model>. . .</temp-model>
                    <msg-retention>. . .</msg-retention>
                </IBM-QCF>
            </IBMMQ>
            <tmax>
                <export-name>TMAXSERV</export-name>
                <host-name>111.111.111.1</host-name>
                <port>4567</port>
                <backup-host-name>
                    111.111.111.2</backup-host-name>
                <backup-port>4568</backup-port>
                <tmax-connection-pool>
                    <pooling>
                        <min>10</min>
                        <max>20</max>
```

```
                        <step>4</step>
                        <period>3600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>true</enable-wait>
                        <wait-time>20000</wait-time>
                    </wait-free-connection>
                </tmax-connection-pool>
            </tmax>
            <SONICMQ>
                <SONIC-QCF>
                    <export-name>SONICMQ</export-name>
                    <prefetch-count>. . .</prefetch-count>
                    <prefetch-threshold>. . </prefetch-threshold>
                    <broker-host-name>. . .</broker-host-name>
                    <broker-port>. . .</broker-port>
                    <broker-protocol>. . .</broker-protocol>
                    <client-id>. . .</client-id>
                    <connect-id>. . .</connect-id>
                    <connect-url>. . .</connect-url>
                    <default-password>. . .</default-password>
                    <default-user>. . .</default-user>
                    <load-balance>. . .</load-balance>
                    <monitor-interval>. . .</monitor-interval>
                    <persistent-delivery>. </persistent-delivery>
                    <sequential>. . .</sequential>
                </SONIC-QCF>
            </SONICMQ>
        </external-source>
        ...
    </resource>
</jeus-system>
```

6.3.6 결론

이번 절에서는 JEUSMain.xml 에 resource 를 어떻게 설정하는지 간단한 예제를 통해 살펴보았다.

다음 절에서는 클라이언트에서 어떻게 resource 를 사용하는지에 대해 간단하게 설명한다.

6.4 Resource 사용

6.4.1 소개

이 절에서는 위에서 기술했듯이 JEUSMain.xml 에 설정된 Mail Source 와 URL source 를 어떻게 사용하는지 기술한다.

다른 resource 타입들을 사용하기 위해서는 본 매뉴얼의 6 장과 7 장을 참조하기 바란다.

6.4.2 Data Source 사용

7 장을 참조 하기 바란다.

6.4.3 Mail Source 사용

아래는 JEUSMain.xml 에 설정된 Mail Source 를 사용하는 방법이다. 이 예제는 “MY_SMTP_HOST” 라는 이름이 mail entry 에 bind 되어 있어야 한다.

<<*MailSender.java*>>

```
import javax.naming.*;
import java.util.Properties;
public class MailSender {
    public static void main(String[] args) {
        try {
            Context ctx = new InitialContext();
                  javax.mail.Session mailSession =
                (javax.mail.Session) ctx.lookup("MY_SMTP_HOST");
            //Send mail using the JavaMail API...
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

위 예제는 JNDI 환경이 설정되어 있고 JVM classpath 가 잡혀있는 것을 전제로 한다. (5 장 참조)

6.4.4 URL Source 사용

아래는 JEUSMain.xml 에 설정된 URL Source 를 사용하는 방법이다. 아래 예제는 “PRIMARY_URL” 라는 이름이 URL entry 에 bind 되어 있어야 한다.

<<*URLUser.java*>>

```
import javax.naming.*;
import java.util.Properties;
public class URLUser {
    public static void main(String[] args) {
        try {
            Context ctx = new InitialContext();
                  java.net.URL myURL =
                (java.net.URL) ctx.lookup("PRIMARY_URL");
            //Use the URL...
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

위 예제는 기본적으로 JNDI 환경에 등록되어 있어야 하고 JVM classpath 가 잡혀있어야 한다. (5 장 참조)

6.4.5 IBM MQ 와 Tmax TP Monitor Source 사용

IBM MQ 매뉴얼과 Tmax/WebT 매뉴얼을 참조하기 바란다.

6.4.6 결론

이 절에서는 Mail Source 와 URL Source 의 사용법을 살펴봤다. 다른 종류의 external resource 는 다른 장을 참조하기 바란다.

6.5 결론

JEUS external resource 에 대한 전반적인 내용을 살펴봤다. 이러한 resource 는 JEUS WAS 에서 사용하는 back-end 데이터와 서비스를 나타내며, 엔터프라이즈 어플리케이션에 운영된다는 것을 보았다.

다음 장에서는 JDBC Resource 에 대해서 설명한다.

7 DB Connection Pool 과 JDBC

7.1 소개

이번 장에서는 DBMS 을 가지고 작업하는 어플리케이션을 구현할 때 꼭 알아야 할 모든 element 에 대해서 설명한다. 이번 장은 JDBC programming 에 관한 지식을 가지고 있다면, JEUS 의 특화된 element 를 배우는 장이 된다.

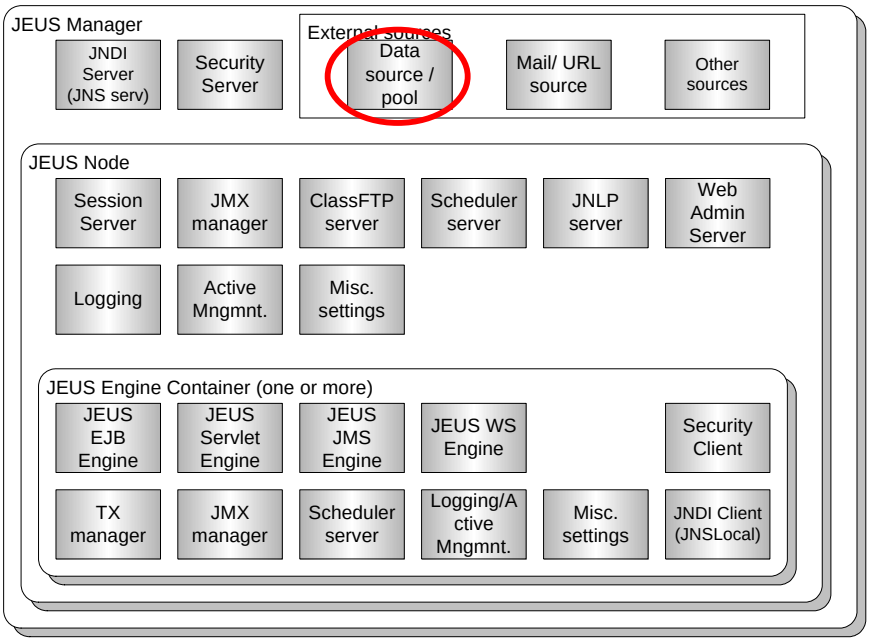

그림 20 Data source 와 JDBC connection pools

7.2 JEUS JDBC 의 개요

7.2.1 소개

JDBC(Java Database Connectivity)는 Java 어플리케이션(Application)이 관계형 DB 에 접속하기 위한 Java 표준 API 이다.

JEUS 는 JDBC 3.0API 를 지원하며, JEUS 가 모든 JDBC 의 기능들을 제공하기 위해서는 확실한 구성이 필수적으로 요구된다. 최적화된 JDBC 기능을 사용하기 위해 본 매뉴얼을 따라 알맞은 구성을 해야 한다.

7.2.2 지원되는 JDBC 드라이버들

JEUS 는 JDBC 인증을 받은 드라이버들을 지원한다. http://industry.java.sun.com/products/jdbc/drives 에서 인증된 드라이버들을 찾을 수 있다. 환경구성은 서로 다르며 이는 각 드라이버 설정을 위해 다른 속성들을 요구하기 때문이다. 보다 자세한 설명은 7.3 절을 읽어보기 바란다.

7.2.3 Web Container 의 Connection Pooling

JEUS Web Container 는 그 자신을 위한 connection pooling 을 제공한다. 자세한 설명은 “Web Container 안내서”를 참조하기 바란다.

7.2.4 JEUS Connection Pooling

Connection Pooling 은 database connection 캐시를 위한 하나의 프레임워크(Framework)이다. Connection pool 이 시작될 때 특정한 수의 물리적 connection 을 만들며 이는 어플리케이션 실행 시간에 connection 생성을 위한 오버헤드(overhead)를 감소시킨다.

7.2.5 Pooling 구조

아래의 [그림 21] 은 JEUS JDBC Connection Pooling 의 전체적인 구조를 보여준다.

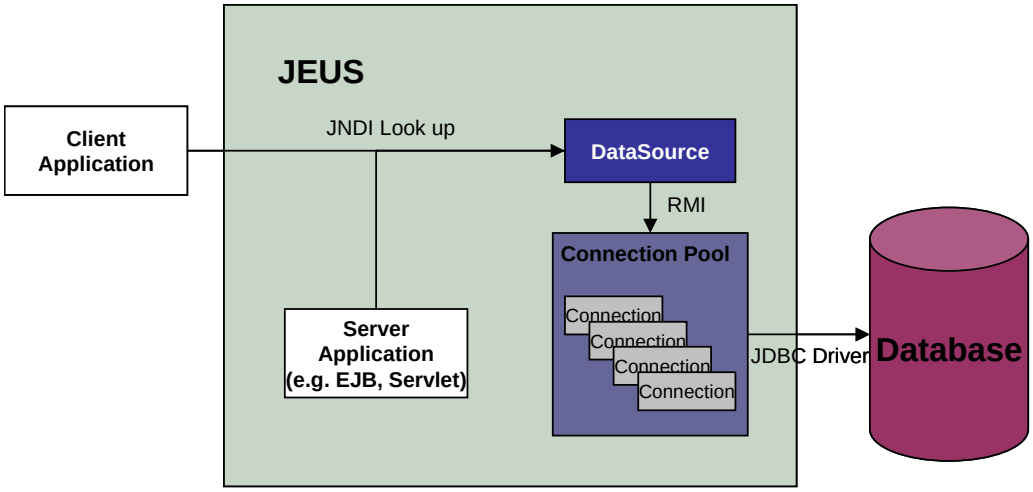

그림 21. JEUS 의 Connection Pooling.

7.2.6 Connection Pool 의 이점

Connection Pool 사용의 이점들은 아래의 두 가지로 요약할 수 있다.

1. **보다 높은 성능**

DB Connection 은 처리과정이 느리다. Connection Pool 안에서의 모든 실제 커넥션들은 사용시에 만들어져 어플리케이션을 위한 준비를 한다. Connection 을 더 이상 사용하지 않을 때에는 그것을 Pool 에 반환시켜서 connection 중단의 오버헤드를 감소시킬 수 있다.

2. 연결 관리

이것은 connection pool 과 함께 동시 connection 들의 수를 제어할 수 있다. 동시 connection 들의 최대 수를 구성함으로써 DB 의 동시 connection 을 제한하는 작업을 효율적으로 할 수 있다.

7.2.7 DataSource

하나의 javax.sql.DataSource 는 어플리케이션과 connection pool 사이의 인터페이스이다. javax.sql.DataSource 객체(object)는 DB Connection 들의 factory 로서 고려되어 질 수 있으며 그리고 이것은 java.sql.DriverManager 이상의 많은 이점을 제공한다.

7.2.8 DataSource 사용의 이점

javax.sql.DataSource 인터페이스 사용의 이점은 아래의 세가지로 요약할 수 있다.

1. **No hard coding**: javax.sql.DataSourc 를 가졌다면 어플리케이션에서 driver 정보의 hard code 가 필요 없을 것이다. 대신에 JEUS 에서는 JDBC 드라이버 구성은 JEUSMain.xml 에 들어가 있다.

2. **Connection Pool**: connection pool 은 성능과 관리면에서 많은 이점을 가져다 준다. `javax.sql.DataSource` 은 하나의 connection pool 의 구성에서 중요 역할을 수행한다.

3. **분산 트랜잭션(Distributed Transaction)**: 분산 트랜잭션을 가진 어플리케이션의 구현을 위해 XADataSource 의 설정이 필수적이다.

7.2.9 DataSource 타입들(DataSource Types)

아래는 DataSource 들의 4 가지 타입들을 간략하게 정리하였다.

- **DataSource**: 사용자들을 위해 Connection 을 반환한다.
- **ConnectionPoolDataSource**: Connection Pool 로부터 사용자들을 위해 Connection 을 반환한다.

- **XADataSource**: XA 연결 Pool 로부터 사용자들을 위해 분산/전역 Transaction 역할을 하는 Connection 을 반환한다.
- **LocalXADataSource:** XA 연결 Pool 로부터 사용자들을 위해 지역 Transaction 역할을 하는 connection 을 반환한다.(일명 지역 XA DataSource).

지금까지 javax.sql.DataSource 를 사용해서 RDB 로부터 connection 을 쉽게 얻어 오는 방법을 보았다. DataSource 에는 다양한 종류가 있으면서 각각 장단점을 가지고 있다. 그러므로 어플리케이션에서 필요한 것이 무엇인지 확인하고, 최적의 성능을 위해 적당한 타입을 사용해야 한다.

7.2.10 Datasource 클러스터링 및 FailOver

어플리케이션 서버 차원에서 데이터베이스의 FailOver 기능을 제공하기 위해서 데이터소스 클러스터링을 사용한다.

데이터소스 클러스터링은 근본적으로는 하나의 JNDI export name 을 가진 데이터소스 인스턴스이다. 이 인스턴스는 DB 호출을 받아서 여러 DB(실제 데이터소스) 중 하나로 전달시켜주는 역할을 한다. 만약 주 DB 가 다운되었을 경우, 클러스터링의 다른 DB 가 선택되어서 어플리케이션의 요청 사항을 처리하게 된다. 어플리케이션에서는 단지 하나의 데이터소스만 보게 되므로, 클러스러링과 FailOver 가 투명하게 제공된다[그림 22].

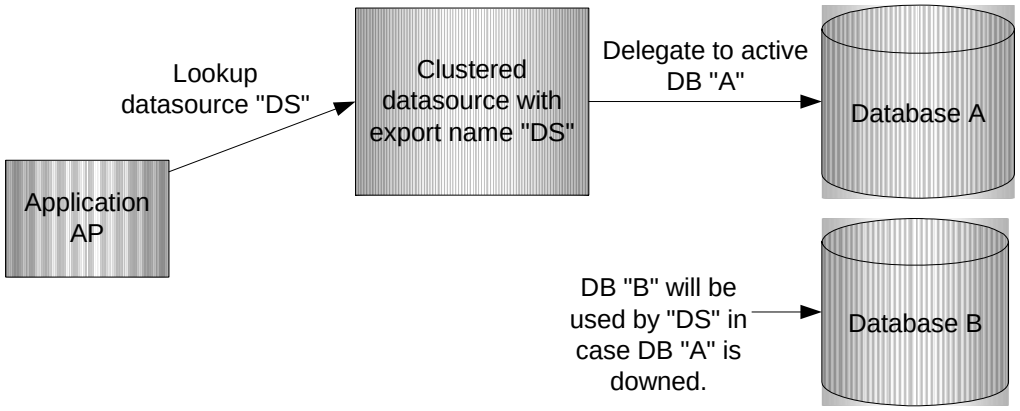

그림 22. 데이터소스 클러스러팅 서비스는 DB 호출을 사용할 수 있는 다른 DB 로 위임시키는 방법으로 DB fail-over 기능을 제공한다.

7.2.11 결론

이번 장에서는 JEUS JDBC 의 개념을 설명했다. 현재 여러분은 JEUS JDBC 의 실제적인 구성을 위한 준비가 되었을 것이다. 구성들의 상세한 설명을 위해 다음 장을 반드시 읽어주기 바란다.

7.3 JDBC DataSource 구성

7.3.1 소개

JEUS 에서 JDBC 드라이버 구성을 시도할 때 JEUS_HOME\lib\datasource\ 디렉토리 안에 JDBC 드라이버 클래스 라이브러리가 있는 지 먼저 확인한 후 다음 절과 같이 JEUSMain.xml 을 구성하면 된다.

7.3.2 JEUSMain.xml 기본구성

JEUSMain.xml 에 DataSource 를 설정 할 수 있다. Javax.sql.DataSource 의 속성들은 각 드라이버별로 다르기 때문에 사용하기 원하는 드라이버의 독점적 특성을 파악하고 그 특성에 맞게 설정을 해야 한다. 각 element 들의 전체 리스트는 부록 J 를 참조하기 바란다. 또한 JDBC 구성 예제들은 부록 I 를 보기 바란다.

아래의 XML element 들은 <resource><data-source><database>XML element 의 하위 element 로서 이용할 수 있다.

- **Vendor**: DB 벤더의 이름 (“oracle”, “jeus_mssql”, “db2”, “sybase”, “inet”, “others”).
- **Export name**: javax.sql.DataSource 객체하의 JNDI export 이름.
- **Data source class name**: JDBC 드라이버의 클래스 이름.
- **Data source type**: “DataSource”, “ConnectionPoolDataSource”, “XADataSource”, “LocalXADataSource”.
- **Database name**: Database 의 이름.
- **Data source name**: 일반적으로 이것은 data source class name 과 동일.
- **Service name**: Oracle DB 의 SID(inet JDBC 드라이버 사용시).
- **Description**: DataSource 를 위해 설명하는 텍스트.

- **Network protocol**: Database Connection 을 할 때 사용되는 프로토콜 (예를 들면 Sybase 의 "Tds").
- **Password**: 사용자의 암호.
- **Encryption**: 만약 "true"로 설정되었다면 password 는 Base64 로 암호화되어 있을 것이다. 그렇지않다면 test 기반의 password 일 것이다.
- **Port number**: Database 리스너의 포트번호.
- **Server name**: Database 가 운용중인 서버의 이름이나 IP 주소.
- **User**: 사용자 이름.
- **Driver type**: Oracle JDBC 드라이버의 타입(예를 들면 thin,oci).

예제 (Oracle):

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                <vendor>oracle</vendor>
                <export-name>datasource1</export-name>
                <data-source-class-name>
                    oracle.jdbc.pool.OracleDataSource
                </data-source-class-name>
                <data-source-type>
                    DataSource
                </data-source-type>
                <database-name>MYDB</database-name>
                <data-source-name>
                    oracle.jdbc.pool.OracleDataSource
                </data-source-name>
                <service-name>oradb1</service-name>
                <description>
                    This is the DataSource setting for all
                    transactions involved with online shopping
                    application
```

```
                </description>
                <network-protocol/> <!--Not used with Oracle-->
                <password>tiger</password>
                <encryption>false</encryption>
                <port-number>1521</port-number>
                <server-name>192.168.1.2</server-name>
                <user>scott</user>
                <driver-type>thin</driver-type>
                ...<!-- See later sub-section -->
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

참고: <database>element 들은 <data-source>element 아래에 추가된다.

7.3.3 주요 벤더들이 사용하는 속성들

다음 표 4 는 JEUSMain.xml 안의 DataSource 에서, 각 벤더 별 기본 속성을 사용하는 정리한 것이다. 각 속성은 각 드라이버 벤더를 위해서 Required, Optional, Not Supported 로 표시했다.

표 4. JDBC DataSource XML 구성 element(JEUSMain.xml 에서).

	Oracle Type 2	Oracle Type 4	i-net	Sybase	DB2
vendor	O	O	O	O	O
export-name	O	O	O	O	O
data-source-class-name	O	O	O	O	O
data-source-type	O	O	O	O	O
database-name	O	O	O	O	O
data-source-name	O	□	□	□	X
service-name	X	X	O	X	X
description	□	□	□	□	□
network-protocol	□	□	□	O	□
password	O	O	O	O	O
encryption	□	□	□	□	□
port-number	O	O	O	O	O
server-name	O	O	O	O	O
user	O	O	O	O	O
driver-type	O	X	X	X	X

	Oracle Type 2	Oracle Type 4	i-net	Sybase	DB2

Legend: ○ ***Required*** □ ***Optional*** **X** ***Not supported***

7.3.4 DB Connection Pool 의 구성

<database>element 에 구성된 각 DataSource 는 향상된 성능을 위하여 Database Connection 캐시를 사용하는 DB 연결 Pool 에 속하는 설정을 포함할 것이다.

DB Connection Pool 은 JEUSMain.xml 의 <database>element 아래 <connection-pool>을 사용하여서 구성된다.

- Connection Pooling 옵션들:
  - **Min**: 캐시를 위한 DB connection 들의 초기값.
  - **Max**: 캐시를 위한 DB connection 들의 최대값.
  - **Step**: 동시에 connection 을 증가하여 pool 에 추가시킬 값.
  - **Period**: 시간주기는 idle connection 을 결정하는데 사용된다. 시스템은 connection 들의 상태를 체크하고 만약 connection 들이 정해진 시간 동안 활동하지 않는다면 그것들은 connection 들의 최소값까지 pool 에서 제거된다.
- **wait free connection** 설정은 pool 안에 있는 모든 connection 들이 사용되었을 때 동작된다.
  - **enable wait** 설정: 이 태그는 pool 안에 이용 가능한 connection 이 없거나 pool 안에 connection 들이 이미 최대값이 되어질 때 DB Connection 요청을 처리하는 방법을 결정한다. 만약 true 라면 시스템은 이용 가능한 connection 을 얻기 위해 대기한다. 만약 false 라면 시스템은 사용자 요청이 올 때 새로운 connection 을 만들고 사용이 끝난 이후에 pool 에 반납하지는 않는다.
  - **Wait time**: 이 태그는 <enable-wait>가 true 일 때만 유효하다. 이것은 사용자가 connection 을 위해 대기하는 시간을 나타낸다. 만약 어떠한 connection 도 사용자가 이 시간 동안 대기해서 이용할 수 없을 때는 시스템은 사용자에게 Exception 을 던져준다.

- **Operation timeout**: DB operation 의 time out 을 결정하는데 사용되는 시간 주기를 정의한다.

- **Delegation datasource**: 이 설정은 현재의 XA DataSource 의 전역/지역 Transaction 을 NonXA-DataSource 로 넘길 때 사용한다.

- **Max use count**: 설정된 숫자만큼 connection 을 사용하게 되면 해당 connection 을 버리고 새로운 connection 을 맺게 된다. 기본값은 '0'이며 이는 connection 들을 교체하지 않겠다는 의미이다.

- **Delegation DBA**: DBA 의 id 와 password 를 세팅해서 DBA Connection 을 만들 수 있다. 이 같은 강력한 DBA Connection 들은 DB 연결들에 문제가 있는 경우에 JEUS 가 사용한다. 예를 들어 만약 JEUS 가 con.close()를 호출함으로써 DB Connection 를 강제적으로 close 를 시도할 때, 그러나 이 call 이 Exception 을 발생한다면 JEUS 는 아마도 low-level 'kill'신호로 강제적으로 close 시키려 할 것이다. 이 같은 kill 신호들은 특별한 전용의 DBA Connection 을 요청한다. 이 element 는 DBA DataSource Connection 의 export name 으로 구체화 된다. 이 DBA DataSource 는 kill 신호들이 보내 졌을 경우에 사용될 것이다.이 element 는 오직 sybase 와 oracle 에서만 적용이 가능하다.

- **DBA timeout**: DBA 의 "kill" 명령어가 실행될 시간을 지정할 수 있다. <dba-timeout>태그를 사용하며, millisecond 단위로 설정한다. timeout 의 기본값은 무제한이다('-1').

- **Check Query**: 커넥션에 문제가 있는지를 어플리케이션 서버에 보고할 때 사용된다. check query 가 지정되면, 어플리케이션 서버가 활성화된 커넥션을 점검할 때 이 query 를 전송하게 된다. 이 check query 는 Oracle, Sybase, MSSQL Server 에서는 속도 저하를 발생시키므로 사용하지 않도록 한다.

- **Statement Caching Size**: PreparedStatement 객체를 얼마나 캐시할 지 지정한다. 기본값으로는 이 기능을 사용하지 않는다. Oracle 9i 이후 버전에서는 사용하지 않도록 한다. 다른 DBMS 에서는 시스템 리소스의 낭비를 줄임으로써 성능 향상을 가져 올 수 있다.

예제:

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
```

```
    <resource>
        <data-source>
            <database>
                ...<!-- See earlier sub-section -->
                <connection-pool>
                    <pooling>
                        <min>2</min>
                        <max>20</max>
                        <step>1</step>
                        <period>500000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>true</enable-wait>
                        <wait-time>60000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                    <delegation-datasource>
                        datasource2
                    </delegation-datasource>
                    <max-use-count>30</max-use-count>
                    <delegation-dba>MYDBA</delegation-dba>
                   <dba-timeout>60000</dba-timeout>
                    <check-query>SELECT * FROM chk</check-query>
                    <stmt-caching-size>10</stmt-caching-size>
                </connection-pool>
            </database>
            ...
        </data-source>
    </resource>
</jeus-system>
```

7.3.5 Custom DB Property 추가

앞서 설명한 기본 설정으로 부족할 때 <database>구성에 custom property 몇 개를 추가하여 property 를 가진 Database 드라이버를 사용할 수 있다.

참고: property 들은 DB 벤더를 "others"로 설정할 때만 사용된다.

각각의 새로운 property 을 추가를 위해 세가지 항목을 정의함으로써 사용할 수 있다.

- property 의 **name**.
- property 의 **type**: String class 의 전체적인 유효한 class 이름 또는 우선 wrapper class (예: “java.lang.Integer”).
- property 에 문자열을 줌으로써 **value** 를 부여할 수 있다.

예제:

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                ... <!-- See earlier sub-section -->
                <property>
                    <name>PortNumber</name>
                    <type>java.lang.Integer</type>
                    <value>1099</value>
                </property>
            </database>
            ...
        </data-source>
    </resource>
</jeus-system>
```

<property>element 는 위의 예처럼 추가된다.

위에서 설명된 DataSource 를 적절히 구성해 주었다면, JEUS 를 재시작 해주어야 DataSource 를 클라이언트 어플리케이션에서 이용 가능하다. (Servlet, EJB 등).

참고: 클라이언트에서 Connection Pool 에 최초로 접근할 때 실제로 connection 이 생성된다.

7.3.6 데이터소스 클러스터링 설정(DS Cluster)

앞서 보았듯이, 어플리케이션 서버 차원에서 여러 DB 간의 fail-over 기능을 제공하기 위해서 데이터소스 클러스터링 서비스를 설정할 수 있다. 이 설정은 <resource><data-source><cluster-ds> 태그를 사용해서 설정한다. 이 태그 내의 설정은 다음과 같다.

- export name: 데이터소스 클러스터링을 위한 논리적인 JNDI 이름. 이 어플리케이션에서 lookup 할 때 사용된다.
- is-pre-conn: Boolean 값으로, true 이면 클러스터링에 참여된 모든 DB 의 커넥션을 유지하며 사용한다. 이렇게 하면 성능이 향상된다. 반대로 값이 false 이며, 활성화된 DB(주 DB) 에서만 커넥션이 살아서 사용되고, 활성화된 DB 가 다운되어 백업 DB 가 동작할 때만 새로운 커넥션이 생성된다. 이렇게 하면 시스템 리소스를 절약할 수 있다.
- backup data source: 데이터소스 클러스러팅에 포함되는 데이터소스의 export name 을 ','로 구분하면서 적어준다.

리스트의 첫번째가 주 데이터소스가 되어 사용되다가 다운되면 다음 DB 가 선택된다.

예

<<*JEUSMain.xml*>>

```
<jeus-system>
    . . .
    <resource>
        <data-source>
            <database>
                <export-name>A</export-name>
                . . .
            </database>
            <database>
                <export-name>B</export-name>
                . . .
            </database>
            . . .
            <cluster-ds>
                <export-name>DS</export-name>
                <is-pre-conn>true</is-pre-conn>
```

```
                <backup-data-source>A,B</backup-data-source>
            </cluster-ds>
        </data-source>
    </resource>
</jeus-system>
```

7.3.7 결론

Connection Pool 을 구성할 때는 JDBC 드라이버를 위한 정보가 필요하다. JEUS 에는 주요 벤더들의 JDBC 드라이버를 위한 프로퍼티를 가지고 있다. 그러나 이외에도 수많은 벤더가 있기 때문에, 이 모두를 만족시키는 프로퍼티를 만들 수가 없다. 이런 경우에 <property>태그를 사용하면, 별 문제 없이 설정할 수 있다.

7.4 DBConnectionPool 의 제어

7.4.1 소개

사용자는 DB Connection Pool 을 실시간으로 제어하기 위해서 콘솔 툴인 ‘dbpooladmin’이나 웹 관리자를 사용할 수 있다. 웹 관리자에 대한 설명은 JEUS 웹 관리자 안내서를 참조 하기 바란다.

7.4.2 dbpooladmin 를 사용한 DB Connection Pool 의 제어

JEUS 는 콘솔 툴인 dbpooladmin 을 제공한다. 그것은 JEUS_HOME\bin\아래에서 위치해 있다.

사용 예제:

```
C:\> dbpooladmin johan_mycontainer
```

이것은 dbpooladmin 를 시작하는 것이며 DB Pool 이 JEUS 의 노드인 'johan'과 'mycontainer'이라고 명명된 Engine Container 에 연결하는 것이다. 다음과정을 진행하기 위해서는 유효한 사용자명과 패스워드의 입력이 필요하며 잠시 후에 프롬프트가 나타날 것이다.

```
johan_mycontainer>disable datasource1
```

위의 명령어는 'datasource1'이라 명명된 DB Pool 을 불능상태로 만들 때 사용된다.

```
johan_mycontainer>info
```

아래와 유사한 정보를 볼 수 있는 명령어이다.

```
=========================================================
id     name       min  max  current  idle  waiting   working
=========================================================
1 datasource1     2    10   2      2     false     false
2 jdbc/DB1        6    10   6      6     true      true
3 localxa         6    10   6      6     true      true
=========================================================
```

위의 "working"필드는 pool 이 비활성 상태임을 나타낸다. pool 을 활성화시키기 위해서 아래의 명령어를 적용한다.

```
johan_mycontainer>enable datasource1
```

'help'명령어를 사용하여 다른 명령어들의 정보를 확인할 수 있다. 부록 C 를 보면 DB Connection Pool 의 제어를 위한 이 툴에 대한 더 많은 정보를 볼 수 있을 것이다.

info 명령의 옵션에 대한 설명은 부록 C 를 참조하기 바란다.

7.4.3 결론

이런 명령어들은 자주 사용되지 않지만, 필요할 때에는 요긴하게 사용할 수 있다. 만약 JEUS 가 트랜잭션을 많이 사용한다면 dbpooladmin 의 모든 명령어들을 필수적으로 기억해야 한다.

다음 장에서는 구성된 DataSource 를 어떻게 모니터링 해야 하는지 간단히 설명한다.

7.5 DB Connection Pool 의 모니터링

7.5.1 소개

이번 장에서는 어떻게 DataSource(connection pool)를 dbpooladmin 을 사용하여 모니터링할 것인가를 간략하게 설명할 것이다. 웹 관리자에 대한 설명은 JEUS 웹 관리자 안내서를 참조 하기 바란다.

7.5.2 dbpooladmin 을 사용한 모니터링

dbpooladmin 에서 하나의 Engine Container 에 구성된 DB Pool 을 모니터링 하기 위해서 오직 하나의 명령어인 'info'를 사용할 수 있다. dbpooladmin 프롬프트에서 이 명령어의 타입은 아래의 리스트와 유사한 정보를 제공할 것이다:

```
=========================================================
```

```
id     name       min  max  current  idle  waiting   working
=========================================================
1 datasource1     2    10   2      2     false     false
2 jdbc/DB1        6    10   6      6     true      true
3 localxa         6    10   6      6     true      true
=========================================================
```

이 출력창에 대한 각 필드의 설명이다:

- **name** : DB Pool 의 export name.
- **min** : Pool 안에서 유지되는 DB 연결들의 최소크기.
- **max** : Pool 안에서 유지되는 DB 연결들의 최대크기.
- **current** : Pool 안에서 DB 연결들의 현재 총 수 (active + idle).
- **idle** : idle 상태의 DB Connection 들의 현재 수.
- **waiting**: 어느 클라이언트든지 wait-queue 에 들어 있는지를 나타낸다.
- **working**: 만약 DB Pool 이 활성화 상태이면 "true"이고 비활성화이면 "false"이다.

info 명령의 옵션에 관한 설명은 부록 C 를 참조하기 바란다.

7.5.3 결론

JDBC Connection Pool 의 모니터링에 관한 설명은 끝났다. 다시 한번 강조하는데 JDBC Connection Pool 은 처음 요청이 왔을 때 비로소 생긴다.

다음 장과 마지막 장에서는 클라이언트 코드에서 어떻게 JDBC Connection Pool 의 사용 할 것인가를 기술할 것이다.

7.6 JEUS JDBC 프로그래밍

7.6.1 소개

이전 장에서 구성한 Connection Pool 을 사용하여 어떻게 코드를 작성하는 지를 여기서 배울 것이다. JDBC 어플리케이션 프로그래밍 가이드로서 이번 장을 활용하기 바란다.

7.6.2 간단한 JDBC 프로그래밍

따르는 코드는 JEUSMain.xml 에 구성된 DataSource 를 사용하여 어떻게 connection 을 얻는 지를 보여주고 있다.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("ds1");
Connection con = ds.getConnection();
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from test");
...
...
con.close();
```

주의: Connection 을 사용한 후에는 반드시 connection 을 닫아야 한다.

7.6.3 Transaction 프로그래밍 규칙

JEUS 는 transaction 프로그래밍을 위한 표준 단계를 정의 한다. UserTransaction 를 사용하는 모든 어플리케이션은 표준단계들을 따라야 한다.

1. transaction 을 시작한다.
2. DB connection 을 얻어 온다.
3. transaction 의 나머지를 코딩한다.

주의: 만약 여러분이 다른 transaction 처리를 원한다면 반드시 새로운 connection 을 다시 얻어야 한다.

7.6.4 결론

이번 장에서는 Connection Pool 을 사용하는 프로그래밍을 어떻게 할 것인가를 기술했다. 잘못된 프로그래밍은 어플리케이션을 느리게 하거나 시스템 failure 을 야기할 수 있다. 만약 본 매뉴얼의 권장 사항을 따르려 한다면, 다시 유심히 다시 한 번 보기 바란다.

7.7 DB Connection Pool 튜닝

추가적인 성능향상을 위해서는 DB Connection Pool/JDBC DataSource 를 설정할 때 아래의 사항을 숙지해야 한다:

- 적절한 <vendor>이름을 선택한다. 이렇게하면 Connection Pool 을 최적화할 수 있다.
- 알맞은 <data-source-type>를 선택한다.
- 일반적인 pool 튜닝을 적용한다: 보다 높은 min, max ,step 값은 요청처리면에서 볼 때, 전체성능이 향상되지만 잠재적인 비용면으로 보면 시스템 자원을 낭비하게 된다.
- 연결 pool 구성의 <period>태그는 미약하나마 보다 나은 성능을 위해서 보다 길게 설정해야 한다.

7.8 결론

JEUS JDBC 와 JEUS JDBC Connection Pool 에 관한 설명은 여기서 끝을 맺는다. 클라이언트 코드 안에서 JDBC 연결의 성능향상을 위해서 Connection Pool 을 사용함을 보았다.

다음 장에서는 JEUS 의 핵심 중 하나인 JEUS 노드에 대해서 살펴본다.

8 Node

8.1 소개

JEUS 노드 는 JEUS Manager 와 밀접하게 관련된 JEUS Server 의 기본 구성요소이다 [그림 23].

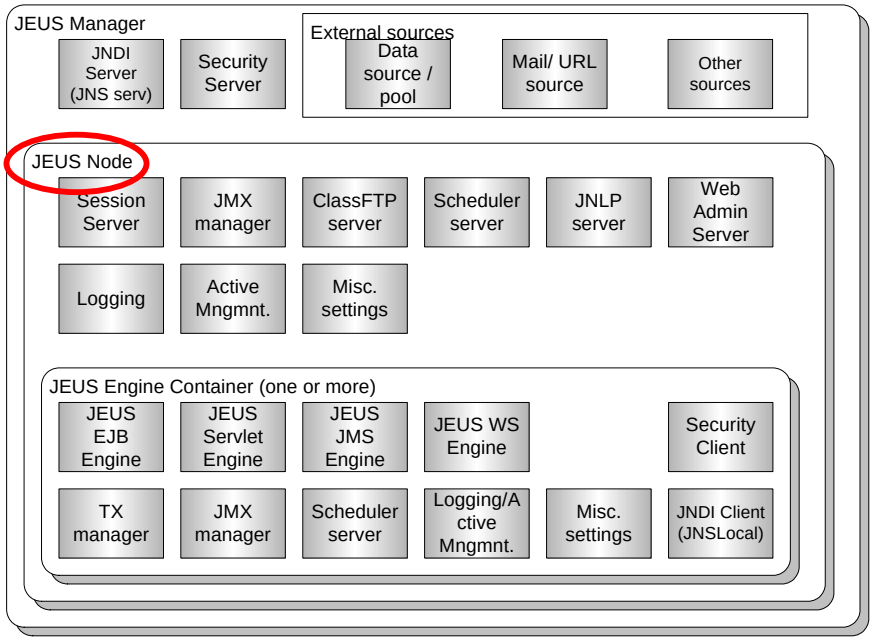

그림 23. 노드 구성요소가 표시된 JEUS 구조

8.2 JEUS 노드 개요

8.2.1 소개

구조상으로 JEUS 노드는 JEUS Manager 와 Engine Container 들 사이에 존재하는 중간 레벨의 구성 요소이다. 각 JEUS Manager 는 하나의 노드와 관련되어 있으며 하나의 노드만을 관리한다.

Session Server 나 JMX 와 Scheduler 등 많은 주요 기능들이 노드레벨에서 설정된다.

이 절에서는 구조적 관점에서 JEUS 노드의 기본적 특징들에 대해서 살펴보도록 할 것이다.

8.2.2 JEUS 노드의 주요 구성요소들

JEUS 노드의 주요 구성요소들은 다음과 같다.

- Listener
- Engine Containers
- Session Server
- 분산 Session Server
- JMX Manager
- Class FTP Server
- Scheduler Server
- JNLP Server
- Web Admin Server
- Logging Features
- Active Management(email notification)

이 후 절들에서 각 구성요소들에 대해서 간략하게 설명할 것이다.

8.2.3 Listener

3.13 절에서 JEUS Base Port 에 대해서 언급했었다. JEUS Base Port 는 JEUS 가 서브 컴포넌트나 클러스터링 환경의 다른 JEUS 와 통신하기 위한 소켓 포트이다.

Listener 는 JEUS Base Port 를 통해 통신하기 위해 필요한 여러가지 설정의 집합이다. 대부분의 경우 별도로 이 설정을 하지 않아도 무방할 것이다.

Listener 의 하위 설정으로는 다음과 같은 것이 있다.

- backlog : JEUS system listener port 인 JEUS Base Port 에 대한 backlog 값을 지정한다.

- thread-pool : Jeus system listener port (JEUS Base Port)에 요청되는 socket connection 처리를 위한 thread pool 을 설정한다.

ssl : Jeus Security system 에 관련된 SSL 속성을 지정한다. 이 element 를 설정하면 Jeus Security system 을 사용하는 모든 socket connection 에 SSL 이 적용된다.

8.2.4 Engine Container 들

JEUS 노드의 핵심 구성요소는 Engine Container 이다. 이미 여러 번 언급했듯이 Engine Container 는 J2EE 어플리케이션들(EJB, Servlet, JMS)을 실행할 책임을 가지는 JEUS Engine 들을 관리한다.

Engine Container 는 그 내부에서 동작하고 있는 Engine 들에게 중요한 서비스들을 제공한다. 이 서비스들에는 트랜잭션이나 네이밍과 보안등이 있다.

비록 하나의 노드에 몇 개의 Container 라도 추가 될 수 있지만, 적어도 하나의 Engine Container 가 각 JEUS 노드에 설정되어야 한다.

Engine Container 에 대한 보다 자세한 사항은 이 매뉴얼의 11 장을 보기 바란다.

주의: Engine Container 의 이름으로는 영문자와 수자의 조합만 가능하다. 단 영문자부터 시작해야 한다.

8.2.5 Session Server 와 분산 Session Server

분산 Session Server 는 Session Server 와 유사한 기능으로 Session Server 를 대체할 수 있다. Session Server 는 JEUS Manager 를 중앙 세션 서버로 두고 클러스터링에 참여하는 모든 Servlet Engine 들이 클라이언트가 되어 세션 유지를 하는 구조이다. 분산 Session Server 는 이와 같은 중앙 집중형 서버가 존재하지 않고 각 Servlet Engine 들이 동등한 입장에서 서로 세션을 교환하여 세션을 유지하는 완전 분산식 세션 클러스터링 기법이다. 완전 분산식인 분산 Session Server 방식이 Session Server 방식보다 확장성면에서 유리하므로 대규모 클러스터링 환경에서는 분산 Session Server 방식을 사용하고 Session Server 방식은 중,소규모 클러스터링 환경에서 사용할 것을 권장한다.

분산 Session Server 설정은 JEUSMain.xml 에 하지만 실제 이 설정을 이용하는 것은 JEUS Web Container 이다. 분산 Session Server 방식으로 세션 클러스터링을 하기 위해서 WEBMain.xml 에 설정해야 할 것은 따로 없다.

분산 Session Server 를 설정하는 자세한 방법은 10 장을 참조하기 바란다.

EJB 클러스터에서 Session Server 를 어떻게 사용하는지에 대한 정보는 JEUS EJB 안내서를 보기 바란다.

8.2.6 JMX Manager

Java Management Extensions (JMX)은 관리와 모니터링을 위한 일반적이고 개방적인 기술로서 관리와 혹은 모니터링이 필요한 어디서든지 배치가 가능하다. 설계상 이 새로운 표준은 기존의 시스템을 붙이거나 새로운 관리기능을 구현하고 솔루션을 모니터링하는데 적합하다.

JMX 는 JEUS 의 관리와 monitoring 을 위한 기반 기술이다. JEUS 에서의 JMX 는 JEUS JMX 안내서를 참조하기 바란다.

8.2.7 Class FTP Server

Class FTP Server 는 EJB 스텁 클래스를 FTP 를 통해서 전송하기 위해 사용한다.

EJB Engine 과 떨어진 다른 JVM 에서 동작하고 있는 EJB 클라이언트 어플리케이션이 어떤 EJB 의 비즈니스 메소드를 호출하려고 시도하려고 할 때, EJB 의 스텁 클래스들이 클라이언트의 클래스 패스에 존재해야 한다. 만약 Class FTP Server 가 사용된다면 스텁 클래스들이 클라이언트로 자동으로 전송될 수 있다.

수작업을 통해 스텁 클래스들을 적당한 위치에 놓을 수도 있다.

참고: 다소 혼란스럽지만 파일을 전송하는 프로토콜은 FTP 프로토콜이 아니라 HTTP 프로토콜을 사용하고 있다

8.2.8 Scheduler Server

Scheduler Server 는 미리 지정한 시간에 특정 작업들이 실행되도록 하는데 사용된다.

Scheduler Server 는 Engine Container 레벨이나 노드 레벨에서 설정이 가능하다.

스케줄러에 관한 보다 자세한 정보는 JEUS Scheduler 안내서를 참조 하기 바란다.

8.2.9 JNLP Server

JNLP Server 는 인트라넷 Java 클라이언트 어플리케이션이 원격의 서버 (JNLP Server 가 있는 곳)로부터 클래스 파일들을 가져와 로드 할 수 있도록 한다.

보다 자세한 정보는 JNLP 스펙서를 참조하기 바란다(Java Web Start 와 JNLP 정보는 다음 사이트를 보기 바란다.

http://java.sun.com/products/javawebstart

또한 이 기능에 관련한 보다 자세한 정보는 JEUS Client Application 안내서를 보기 바란다.

8.2.10 Web Admin Server

Web Admin Server 은 JSP/HTTP 기반의 관리 어플리케이션이다. 이 웹 어플리케이션에 접속 하기 위해서는 Web Admin Server 가 노드에 설정되어 동작하고 있어야 한다.

이 웹 어플리케이션은 JEUS 노드 레벨에서 설정할 수 있다.

8.2.11 Logging

JEUS 시스템의 각 노드는 실행하는 동안 발생하는 에러나 이벤트들을 기록한다. JEUS 의 logging system 은 J2SE 1.4 이상 버전의 logging API 를 지원한다. 따라서 개발자는 logging API 를 통해 JEUS 의 logger 설정을 변경할수 있다. Logging 에 대한 자세한 설명은 14 장을 참고하기 바란다.

8.2.12 Active Management (email notification)

Active Management 는 노드에 심각한 장애가 발생할 때 e-mail 을 통해 통지를 받을 수 있음을 뜻한다. 이것은 시스템에 어떤 문제가 발생할 때 관리자가 가능한 빨리 시스템을 복구할 수 있도록 도와 준다.

8.2.13 노드의 이름

JEUS 시스템(클러스터)의 각 노드는 유일한 이름을 가진다. 이 이름은 시스템 관리자에 의해 설정된다.

기본적으로 노드의 이름은 노드와 노드 관리자가 동작하고 있는 물리적인 호스트의 네트워크 식별 이름과 동일한 이름이다. 예를 들어 만약 어떤 물리적 머신의 네트워크 이름이 “johan”이라면,

그곳에서 동작하는 JEUS 노드의 이름 또한 “johan”이어야 한다. 런타임시에 이 이름은 `java.net.InetAddress. getLocalHost().getHostName()`을 통해서 얻을 수 있다.

단, Virtual Host 기능을 사용하면 노드의 이름을 임의로 지정할 수 있다.

주의: 노드의 이름은 영문자와 수자의 조합으로만 이뤄져야하되, 영문자로 시작하는 문자열이어야 한다.

8.2.14 노드와 관련된 디렉토리의 구조

[그림 24]은 JEUS 노드의 디렉토리를 보여준다. 이 예에서는 노드의 이름이 “moon”이다. 이 그림에서 볼 수 있듯이 JEUS Main 설정 파일은 이 노드 디렉토리 밑에 위치해 있다.

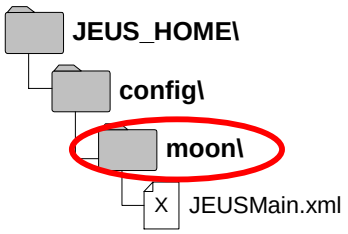

그림 24. 노드와 관련된 디렉토리구조.

8.2.15 노드 클러스터링

노드 클러스터는 JEUS Manager 클러스터와 동일한 표현이다. 보다 자세한 정보는 15 장을 보기 바란다.

8.2.16 결론

노드는 많은 하부의 구성요소들 -Engine Container, Session Server, JMX Manager, Class FTP Server, Scheduler Server-로 이루어져 있다.

다음 절에서는 이러한 주요 구성요소들이 JEUS 의 노드 레벨에서 각각 어떻게 설정되는지 살펴 볼 것이다.

8.3 JEUS 노드 설정

8.3.1 소개

JEUS 노드를 JEUS 시스템에 추가 하기 위해서 <node> XML 요소를 JEUSMain.xml 에 추가해야 한다. 이 요소는 최상위의 <jeus-system>요소 바로 밑에 위치한다. 적어도 하나의 노드가 각 JEUSMain.xml 에 설정되어야만 한다.

예제:

<<JEUSMain.xml>>

```
<jeus-system>
```

```
    ...
    <node>
        ...
    </node>
    ...
</jeus-system>
```

노드 클러스터를 설정하기 바란다면 8.3.5 에서 설명하는 특별한 설정 구조를 적용할 필요가 있다.

8.3.2 기본 노드 설정

아래의 네가지 기본 설정은 각 <node>에 적용된다.

- **node name** 은 노드가 동작할 물리적 장치의 이름과 동일해야 한다. 이 이름은 유닉스에서 “uname –n” 명령을 수행하거나 윈도우 환경에서 컴퓨터의 네트워크 이름을 확인하여 얻을 수 있다.(보다 자세한 정보는 8.2.13 를 보기 바란다). 만약 이 명령들을 통해 “johan.tmax.co.kr”이라는 이름을 얻게 된다면 “johan”이라는 이름을 얻기 위해 첫 번째 ‘.’문자 이후의 것은 무시해야 한다.

- **backup node** 이름은 현재 노드에 심각한 장애가 발생할 경우 노드를 대신해서 동작해야 하는 노드의 이름이다.

- **sequential start** 설정은 Engine Container 들이 순차적으로 시작될지 여부를 결정하는 설정이다. 만약 false 로 설정될 경우 각 container 가 동시에 시작 된다.

- **security switch**: 만약 이 값이 “false”로 설정되면 모든 Engine 에서의 인증과 권한부여가 불가능하게 된다(이것은 성능은 높이지만 어플리케이션에서 보안이 필요하지 않는 경우에만 사용되어야 한다).

예제:

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
    <node>
        <name>johan</name>
        <backup-node>star</backup-node>
        <security-switch>true</security-switch>
        <sequential-start>true</sequential-start>
```

```
        ...
    </node>
    ...
</jeus-system>
```

위의 XML 설정 부분에서 노드의 머신 이름은 "johan"이고 백업 노드의 이름은 "star" 이며 Engine Container 가 순차적으로 시작된다.

8.3.3 노드 하부 구성요소들의 설정

[표 4]는 JEUSMain.xml 의 XML 설정 요소들에 따라, 노드의 Active Management 와 로깅을 제외한 모든 하부 구성요소들을 나열하고 있다. 이 구성요소들 중 몇몇은 복잡하고 별도의 처리를 요구하기도 한다. 이런 구성요소를 위한 추가적인 장이 포함되어있다.

표 4. XML 에서의 설정과 참고할 장이 표시된 노드의 하위 구성요소들

구성 요소	XML 요소 이름	장
Engine Container	<engine-container>	11 장.
Session Server	<session-server>	9 장.
JMX 관리자	<jmx-manager>	JMX 안내서 참조
Class FTP Server	<class-ftp>	현재 장
Scheduler Server	<scheduler>	JEUS Scheduler 안내서 참조
JNLP Server	<enable-jnlp>	JEUS Client Application 안내서
Web Admin Server	<enable-webadmin>	현재 장.

[표 4]에서 처음 세 구성 요소들은 복잡한 설정이 필요하다. 그러나 마지막 네 개의 구성요소들은 구성요소들이 가능한지 불가능한지를 설정하기 위해 간단히 "true/false" 값만을 설정하면 된다.

XML 예제:

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <node>
        <name>johan</name>
        <backup-node>star</backup-node>
        <sequential-start>true</sequential-start>
        <engine-container>
            ...
        </engine-container>
        <class-ftp>true</class-ftp>
        <scheduler>true</scheduler>
        <enable-jnlp>true</enable-jnlp>
        <enable-webadmin>true</enable-webadmin>
        . . .
        <session-server>
            . . .
        </session-server>
        ...
        <jmx-manager>
            ...
        </jmx-manager>
        ...
    </node>
    ...
</jeus-system>
```

위의 예제에서, 마지막 굵게 표시된 4 개 요소들은 Class FTP Server, Schedule r Server, JNLP Server, Web Admin Server 를 가능하도록 각각 설정한 것이다.

8.3.4 Active Management

Active Management 는 <node>의 <system-logging>요소 아래에 <smtp-handler >를 설정함으로써 이 handler 에 설정된 주소로 email 을 받을 수 있도록 설정 한다. <system-logging>과 <smtp-handler>의 설정은 14 장 을 참조하기 바란 다.

8.3.5 노드 클러스터링 설정

여러 노드로 구성된 JEUS 클러스터링을 설정하기 위해서는 클러스터에 속 한 각각의 노드들에 대해 <node> 요소가 있어야 한다. 클러스터에 속한 모든 노드들은 JEUSMain.xml 파일에 같은 설정을 가지고 있어야 한다.

이것에 관한 보다 자세한 사항은 15 장을 참조 하기 바란다.

8.3.6 결론

지금까지 JEUS 노드의 기본 설정과 각 하위 구성요소들의 설정을 찾는 방법에 대해 알아보았다. 또한 마지막으로 Logging 설정과 Active Management 그리고 노드 클러스터 문제들에 대해 고찰해보았다.

다음 절에서는 JEUS 노드를 제어하기 위해 필요한 명령에 대해 간략히 설명한다.

8.4 Node 제어

8.4.1 소개

보통 JEUS 노드의 제어는 jeusadmin 콘솔 툴이나 웹 관리자를 이용한다. 웹 관리자에 대한 설명은 JEUS 웹 관리자 안내서를 참조 하기 바란다.

참고: 이 절에서 논의되는 내용은 부분적으로 4 장 JEUS Manager 에서 논의된 내용과 중복된다. 그 이유는 JEUS Manager 와 JEUS 노드를 항상 명확히 분리 할 수 없기 때문이다.

8.4.2 콘솔을 이용한 Node 제어

만약 jeusadmin 콘솔 툴을 이용한다면 단순히 “boot” 명령을 사용하여 노드와 하위 구성요소들을 시작하고 “down” 명령을 이용하여 정지시킬 수 있다. “boot –d” 명령은 노드를 동적으로 시작하고 현재 활성화된 클러스터에 노드를 추가할 수 있다.

이 명령들에 대한 보다 자세한 정보와 어떻게 jeusadmin 이 수행되는지 배우기 위해서 부록 B 를 참조 하기 바란다.

.

8.5 Node 모니터링

8.5.1 개요

이 절에서는 JEUS 노드를 어떻게 그리고 무엇을 모니터 해야 하는지에 대해 알아볼 것이다.

8.5.2 콘솔을 이용한 JEUS 노드의 모니터링

먼저 jeusadmin 에서 “nodelist” 명령을 사용하여 현재 JEUS 클러스터에 활성중인 노드들의 목록을 얻는 것으로 시작한다. 그런 후에 모니터링 하고자 하는 노드를 선택하여 jeusadmin 을 통해 연결한다. 그리고 “pidlist”같은 “<xyz>list” 명령을 특정 노드의 상태와 활성중인 하위 구성요소의 정보를 얻기 위해 사용한다.

이 명령들에 대한 보다 자세한 정보와 어떻게 jeusadmin 이 수행되는지 배우기 위해서 부록 B 를 참조 하기 바란다.

.

8.5.3 결론

이제 까지 jeusadmin 을 통해 JEUS 노드를 모니터링하는 것에 대해 보았다.

다음 절에서는 JEUS 노드를 튜닝하는 것에 대해 살펴볼 것이다.

8.6 노드 튜닝

노드를 설정할 때 최고의 성능을 보장하기 위해 다음을 고려해봐야 한다.

- Sequential Start 를 사용하지 말아야 한다. 때에 따라서는 Sequential Start 가 필요하기도 하지만, 보통은 노드의 부트시간이 길어지기 때문이다.

위와는 별도로 하위 구성요소들에 대한 튜닝도 확인해야 한다. 보다 자세한 설명은 각 구성요소에 대한 장을 참조하기 바란다.

8.7 결론

이번 장에서는 JEUS 노드에 대한 개념을 알아 보았다. JEUS 노드는 JEUS Manager 와 어느 정도 동등하다는 것을 보았으며 서버 자체라고도 표현할 수 있을 것이다.

JEUS 노드에서 다른 것과 비교해 주목할 만한 하위 구성요소에는 Engine Container, Session Server 그리고 JMX Manager 가 있다.

다음 장에서는 Session Server 에 대해서 논의할 것이다.

9 Session Server

9.1 소개

Session Server 는 클라이언트의 세션 데이터를 관리하거나 백업하는데 사용된다. 그 중에서도 특히 여러 웹서버나 Servlet Engine 으로 클러스터링된 환경을 관리할 때 유용하다.

이번 장에서는 Session Server 의 기본적인 내용을 다룬다. Session Server 는 노드 레벨에서 운영된다 [그림 25]. Session Server 의 모든 사용법을 알기 위해서는 JEUS Web Container 안내서를 참조하기 바란다.

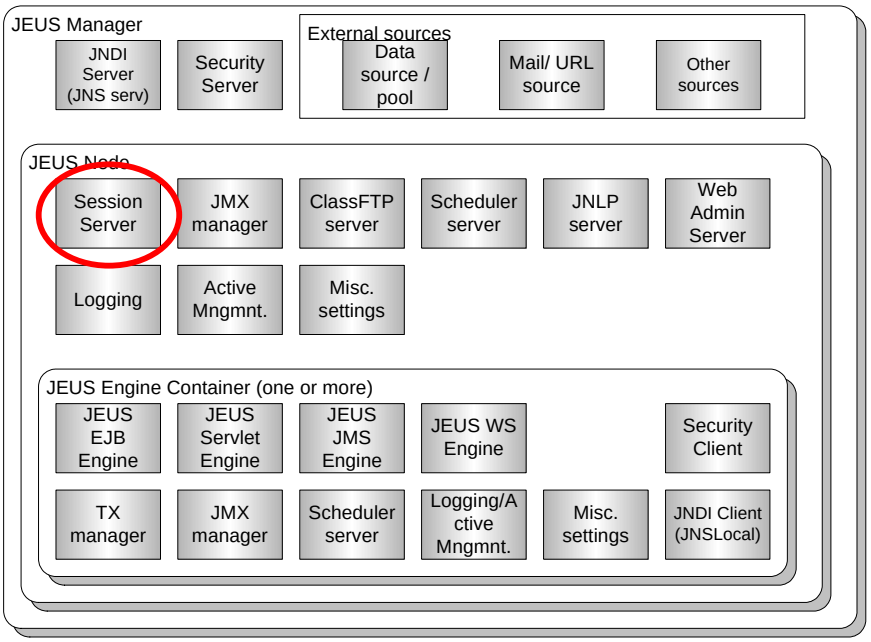

그림 25. JEUS 구조에서의 Session Server

9.2 Session Server 의 개요

9.2.1 소개

이 절에서는 Session Server 의 기본적인 내용을 다룬다.

간단히 Session Server 는 JEUS 에서 세션 객체를 저장하는 곳이라 생각하면 된다. 세션 객체는 Servlet API 의 HTTPSession 객체를 나타낸다. HTTPSession 같은 객체는, 상태 유지가 안 되는 프로토콜인 HTTP 요청을 임시 데이터와 매핑시킨다. 그래서 이 임시 데이터를 통해서 클라이언트를 구별하게 된다. 이렇게 함으로써 WAS 는 작업 중인 클라이언트를 다른 클라이언트와 구분할 수 있으며, 이전에 작업한 내용을 기억할 수 있다. 이런 “session tracking”은 다이나믹한 웹 사이트에서 가장 중요한 부분이다.

그러나 문제는 HTTPSession 객체는, 그 세션이 처음 생성된 Servlet Engine 에만 저장된다는 것이다. 이는 여러 Servlet Engine 들로 클러스터링 환경을 구성했을 경우, 세션을 처음 생성한 Servlet Engine 에게 해당 클라이언트의 모든 요청을 보내야 한다는 의미이다. 그렇지만 항상 특정 Servlet Engine 으로 요청이 간다고는 볼 수 없다. 예를 들어서 웹 서버가 세션 라우팅을 지원하지 않거나, Servlet Engine 이 내부 에러로 인해 갑자기 다운될 경우 세션 객체를 잃어버리게 되기 때문이다.

Session Server 는 세션 데이터들을 한 곳에 모아서 관리한다. 즉, 어떤 클라이언트를 위한 세션 객체나 특정 Servlet Engine 에서 생성된 것도 모두 Session Server 를 이용하는 것이다. 이는 두 가지의 이득이 있다.

첫째는 Session Server 를 사용함으로써 더 이상 웹 서버의 세션 라우팅이 필요 없게 된다.

둘째는 Session Server 를 사용하고 백업 Session Server 를 사용하여 세션 데이터를 백업함으로써 시스템이 더욱 안정화 된다는 것이다.

이로써 Servlet Engine 이 다운이 되더라도 안전하게 세션을 유지할 수 있다.

좀 더 자세한 사용방법이나 Servlet Engine 에서 Session Server 를 사용하는 방법을 익히려면 JEUS Web Container 안내서 9 장의 Session Server 를 참조하기 바란다.

EJB 클러스터링에서도 Session Server 를 사용하는데 이는 JEUS EJB 안내서를 참조하기 바란다.

이 번 장에서는 Session Server 의 기본적인 구조와 Servlet Engine 에 서비스하기 위해 어떻게 세팅하지 설명한다.

9.2.2 Session Server 의 구조

[그림 26]은 하나의 Session Server 에 대한 구조이다. 여러 개의 서브 컴포넌트로 구성된 것을 확인할 수 있다.

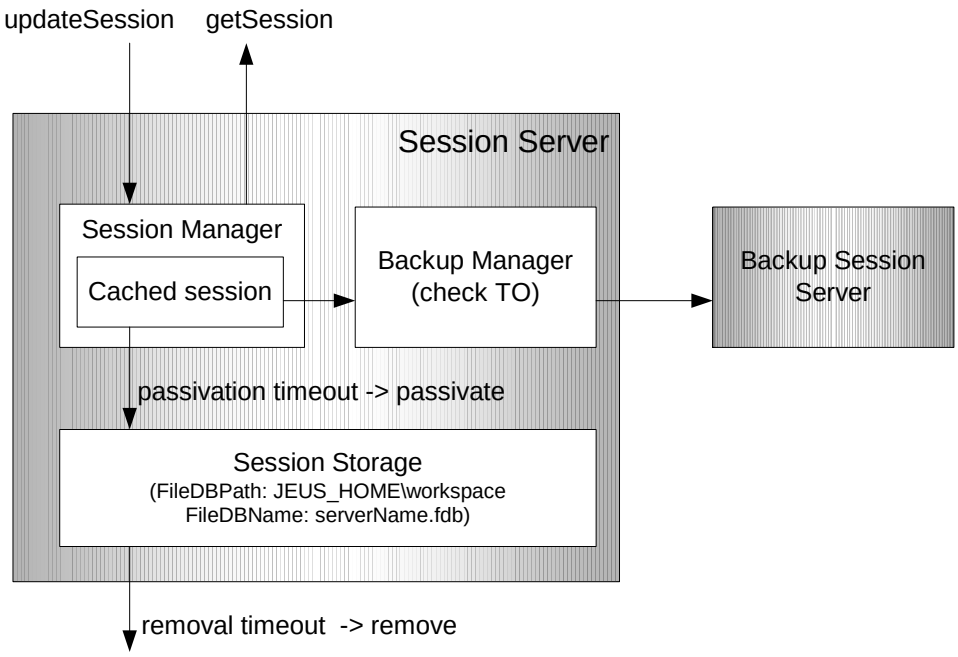

그림 26. Session Server 의 구조

Session Server 의 서브 컴포넌트는 다음과 같다.

- **Session manager**: 각 Session Server 에는 하나 이상의 세션 매니저가 존재한다. 이는 세션 데이터를 저장하거나 요청에 의해 그 값을 가져오는 역할을 한다. Web Container 는 이 세션 매니저에 연결을 맺는다. 일반적으로 각 Session Server 는 하나의 세션 매니저를 가진다.

- **Session cache memory**: 활성화 되어 있거나 한번 사용된 세션 객체는 빠른 사용을 위해서 이곳에 저장된다.

- **Backup manager**: 이는 세션 데이터를 백업 Session Server 에 백업할 때 사용된다. 백업 체크는 환경파일에 설정된 “check-to”의 시간 단위로 이루어 진다(“to”는 TimeOut 을 의미한다).

- **Backup session server**: 일반적인 Session Server 와 같으나 세션 데이터의 백업용으로 설정한 Session Server 이다.

- **Session (permanent) storage**: 이는 세션 데이터의 사용이 줄어들면 저장하는데 사용된다. 스토리지는 공간으로 하드 드라이브를 사용한다. 세션 스토리지에 있는 세션 객체는 활성화 되지 않은 것으로 간주된다. 활성화된 스토리지는 관리자가 지정한 이름이나 지정하지 않았을 경우에는 기본적으로 <Session Server 이름>.fdb 라는 이름으로 파일

을 가진다. 세션 데이터는 “passivationTO”이 경과되면 Session cache 로부터 비활성화 되어 세션 스토리지로 담기고 “removalTO” 이 경과되면 영원히 삭제된다.

9.2.3 클라이언트와 Session Server 의 관계

Web Container 는 Session Server 와 연결된다(JEUS Web Container 안내서 참조). 한 번 연결할 때, Web Container 는 세션 객체가 새로 생성되거나 수정이 되면 Session Server 에 세션 객체를 저장하거나 업데이트 한다. 그리고 클라이언트의 요청이 들어올 때 마다 저장소로부터 세션 객체를 가지고 온다.

Stateful EJB 의 기능도 이와 유사하다.

클라이언트 요청에 관하여 “operation-to”이라는 타임 아웃 값이 있다. 이 값은 Session Server 와 Web Container 사이의 통신 타임 아웃 값이다. 이 시간이 지나면 클라이언트 단으로 에러를 던진다.

9.2.4 두 가지 통신 방식:RMI 와 Socket

세션 매니저와 Web Container 사이의 통신 방식으로 RMI 와 Socket 의 두 가지를 제공한다. RMI 방식은 firewall 이 사용되는 환경에서 문제가 있을 수 있으므로 SOCKET 을 이용한 방식을 사용하는 것을 추천한다.

다음은 소켓 통신 방식에서 알아두어야 할 중요한 것들이다.

- Session Server 의 SOCKET 포트를 신경 쓸 필요가 없다. 이는 JEUS 에서 자동적으로 JEUS_BASEPORT+9 로 해서 설정한다.
- 하지만 아래와 같이 몇몇 SOCKET 관련된 파라미터 값들은 설정해 주어야 한다.
- SOCKET 방식을 설정해야 한다. “-Djeus.session.version=SOCKET” 라는 옵션을 JVM 이 동작하는 JEUS Manager 와 Engine Container 에 추가해야 한다. JEUS Manager 는 JEUS_HOME\bin\jeus 라는 실행 스크립트에서 설정해야 하고, Engine Container 는 JEUSMain.xml 의 <command-option>에 추가해야 한다. 일반적으로 이 옵션을 설정하지 않아도 되는데, 설정하지 않았을 경우 Session Server 는 기본적으로 SOCKET 방식을 사용한다.
- RMI 방식을 사용하려면 “–Djeus.session.version=RMI” 옵션을 위와 같은 방식으로 설정해야 한다.

9.2.5 Session Manager Clustering

Session Server 를 가지고 있는 JEUS Manager 들 간에 클러스터링을 하면 Session Server 간에도 클러스터링이 형성된다. 이렇게 되면 세션 데이터를 여러 서버에서 같이 사용할 수 있게 된다.

JEUS Manager 의 클러스터링에 대해서는 15 장을 참조하기 바란다.

9.2.6 Multi Session Mode

Web Container 가 여러 개의 Session Manager 와 load balancing 을 하기 위해서는 “multi session mode”를 사용해야 한다. 기본적으로 Web Container 는 모든 세션 요청에 대해 하나의 Session Manager 를 사용한다. 그러나 이 방식을 사용하면 여러 개의 Session Manager 를 사용할 수 있다.

이 방식은 PRIMARY 그룹과 BACKUP 그룹으로 Session Manager 가 그룹화 되어있으며 Web Container 가 Session Manager 를 사용하려면 sessionID 에서 특정값을 산출하여 이 값을 기준으로 여러 개의 Session Manager 중에 하나의 Session Manager 를 선택한다. 만약 PRIMARY Session Manager 에 장애가 발생하면 BACKUP 그룹 중에 하나가 대신 사용된다.

Multi Session 은 많은 부하가 걸리는 환경에서 사용한다.

9.2.7 결론

JEUS Session Server 는 기본적으로 Session Server, Session Manager, Backup Manager, Backup Session Server, Session Cache Memory, Session Storage 등 여섯 가지 파트로 되어있다.

Session Server 는 JEUS Web Container 와 연결하여 운영되며, Web Container 에 있는 클라이언트의 세션을 라우팅하거나 백업하는 기능을 제공한다.

다음 절에서 JEUS 에 Session Server 를 설정하는 방법을 다룬다.

9.3 Session Server 의 설정

9.3.1 소개

이번 절에서는 Session Server 의 설정에 대해서 다루는데 이는 상당히 간단하다.

JEUS Web Container 에서 Session Server 를 사용하기 위한 설정 방법은 JEUS Web Container 안내서를 참조하기 바란다.

Session Server 를 사용하여 Stateful EJB 클러스터링을 하는 설정 방법은 JEUS EJB 안내서를 참조하기 바란다.

9.3.2 Session Server 설정

Session Server 는 JEUSMain.xml 파일의 node 절 안에 하나만 존재해야 하며 이는 여러 개의 Session Manager 를 가질 수 있다.

다음과 같이 설정한다.

- **resolution :** 세션 데이터의 passivation/removal 을 체크하는 주기를 결정한다.
- **backlog size:** 서버소켓의 backlog 값을 결정한다. 자세한 사항은 Java API 를 참조하기 바란다. 이 설정은 소켓 방식일 경우에만 사용된다.
- 하나 이상의 Session Manager (다음절에서 설명한다)

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <node>
        ...
        <session-server>
            <resolution>30000</resolution>
            <backlog-size>10</backlog-size>
            <session-manager>
                ...<!-- Next sub-section -->
            </session-manager>
        </session-server>
        ...
    </node>
    ...
</jeus-system>
```

9.3.3 Session Manager 의 설정

Session Manager 는 JEUSMain.xml 의 <node>절 아래에 있는 <session-server>의 하위 태그인 <session-manager>태그에 설정한다.

다음과 같이 설정한다.

- **name :** 세션 매니저의 이름으로 WEBMain.xml 에서 Session Manager 를 찾을 때 사용한다. 이는 Session Manager 를 사용하는 각 Web Container 에서 유일해야 한다. 자세한 사항은 JEUS Web Container 안내서를 참조하기 바란다.

- **multi session :** `multi session mode` 일 때 사용하는 속성으로 `primary` 와 `backup` 둘 중에 하나를 선택할 수 있다. `multi session mode` 를 사용하지 않을 경우 이 태그는 설정하지 않는다.

- **passivation to :** 세션 객체가 세션 스토리지로 옮겨지기 전에 얼마나 오랫동안 캐시 메모리에 남아있을 것인지를 결정한다.

- **removal to :** 세션 객체가 영구히 삭제되기 전에 얼마나 오랫동안 세션 스토리지에 보관될 것인지를 결정한다.

- **operation timeout ("to")** : Session Server 와 Web Container 사이의 통신 타임아웃 값을 결정한다.

- **check timeout ("to") :** 세션 데이터의 백업이 필요할 경우 백업 주기를 결정한다.

- **file DB path** 와 **file DB name :** 세션 데이터를 디스크에 저장할 경로와 이름을 결정한다. 설정하지 않으면 기본적으로 JEUS_HOME\work space\" 경로에 <session server name>.fdb 라는 파일로 사용된다.

- **packing rate :** 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 되는데, 현재 session 객체 갯수 대비 file I/O 횟수가 이 값으로 지정된 ratio 를 넘어서면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

- **min hole :** fdb 파일의 검색 속도를 향상시키기 위해 fdb 파일을 재정리하는 Threshold 값이다. min hole 값에서 지정한 숫자와 같거나 그 이상이 되면 fdb 파일을 액세스 하게되며 더 빠른 액세스를 위하여 fdb 파일을 다시 정리한다.

- **backup name :** Backup Session Server 로 사용될 Session Manager 의 이름을 말한다. 이 이름 역시 WEBMain.xml 에서 Session Manager 를 찾을 때 사용한다. 그러나, multi session mode 의 경우 이 태그는 필요치 않다. 자세한 사항은 JEUS Web Container 안내서를 참조하기 바란다.

- **backup trigger :** 주 Session Server 에서 현재 백업이 이루어지지 않은 세션 객체의 수가 이 값보다 크면 백업하게 된다.

좀 더 자세한 사항은 부록 J 를 참조하기 바란다.

다음은 Session Manager 의 이름이 “session1”이고 백업 Session Manager 의 이름이 “session2” 인 경우의 예제이다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
    <node>
        ...
        <session-server>
            <resolution>30000</resolution>
            <backlog-size>10</backlog-size>
            <session-manager>
                <name>session1</name>
                <passivation-to>1800000</passivation-to>
                <removal-to>3600000</removal-to>
                <file-db-path>c:\sessiondata</file-db-path>
                <file-db-name>sessiondata.fdb</file-db-name>
                <min-hole>2000</min-hole>
                <packing-rate>0.8</packing-rate>
                <check-to>20000</check-to>
                <backup-name>session2</backup-name>
                <backup-trigger>500</backup-trigger>
                <operation-to>10000</operation-to>
            </session-manager>
        </session-server>
        ...
    </node>
    ...
</jeus-system>
```

9.3.4 Multi Session Mode 설정

Multi Session Mod 는 9.3.3 의 설정을 기반으로 한다. 단 <session-manager> <multi-session>의 태그를 추가하여야 하며, <session-manager> <backup-name>의 태그를 추가할 필요가 없다. 왜냐면 <multi-session>의 정의로

primary, backup 을 이미 그룹화하여 관리하므로 특별히 backup-name 을 설정할 필요가 없기 때문이다.

다음은 Session Manager 의 이름이 “session1”이고 백업 Session Manager 의 이름이 “session2” 인 경우의 기본적인 예제이다. session1 의 경우는 node1 에 설정되었고, session2 의 경우는 node2 에 설정되었으며, node1 과 node2 는 이미 노드 클러스터링이 설정되었다고 가정을 한다.

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <node>
        <name>node1</name>
        ...
        <session-server>
            <resolution>30000</resolution>
            <backlog-size>10</backlog-size>
            <session-manager>
                <name>session1</name>
                <multi-session>primary</multi-session>
                <passivation-to>1800000</passivation-to>
                <removal-to>3600000</removal-to>
                <check-to>20000</check-to>
                <operation-to>10000</operation-to>
            </session-manager>
        </session-server>
        ...
    </node>
    <node>
        <name>node2</name>
        ...
        <session-server>
            <resolution>30000</resolution>
            <backlog-size>10</backlog-size>
            <session-manager>
                <name>session2</name>
                <multi-session>backup</multi-session>
                <passivation-to>1800000</passivation-to>
                <removal-to>3600000</removal-to>
                <check-to>20000</check-to>
```

```
                <operation-to>10000</operation-to>
            </session-manager>
        </session-server>
        ...
    </node>
    ...
</jeus-system>
```

9.3.5 Web Container 의 설정

Session Server 를 사용하기 위해서는 하나 이상의 Web Container 가 존재해야 한다(multi session mode 포함). 이의 설정 방법은 JEUS Web Container 안내서를 참조하기 바란다.

EJB 에서 Session manager 를 사용할 때의 설정 방법은 JEUS EJB 안내서를 참조하기 바란다.

9.3.6 Communication Mode 의 설정

다음과 같이 두 가지 통신 방식을 사용할 수 있다.

- RMI : Session manager 와 Engine Container 사이에 RMI 로 통신한다.
- SOCKET : Session manager 와 Engine Container 사이에 SOCKET 으로 통신한다. 이것이 기본 방식이다.

RMI 방식의 설정은 “–Djeus.session.version=RMI” 옵션을 jeus 실행 스크립트 (부록 A 참조)와 JEUSMain.xml 파일의 <engine-container>의 하위 태그인 <command-option> 태그에 설정한다.

SCOKET 방식 설정인 “–Djeus.session.version=SOCKET” 옵션을 위의 방식과 동일하게 설정한다.

9.3.7 웹 관리자를 사용한 Session Server 와 Session manager 의 설정

JEUS 웹 관리자 안내서 25 장 세션 추적을 참조 하기 바란다.

9.3.8 결론

지금까지 Session Server 와 Session manager 에 대한 설정법 및 통신 방식에 따른 설정법을 알아봤다.

다음 절에서는 Session Server 의 성능을 향상시키기 위한 튜닝 방법에 대해서 살펴보겠다.

9.4 Session Server 의 튜닝

9.4.1 Session Server 튜닝

최고의 성능을 위해 Session Server 를 설정할 때 다음을 주의하자.

- resolution 을 증가시키면 메모리는 좀 더 소비하지만 성능을 증가 시킬 수 있다. (위의 9.3 절을 참조하기 바란다)
- Session Server 의 통신 방법을 “SOCKET”으로 한다. 이에 대한 설명은 9.3.6 절에 기술되어 있다.

9.4.2 Session Manager 튜닝

Session Server 의 하위요소인 Session Manager 의 성능을 향상 시키기 위해서는 다음과 같이 설정한다.

- 성능을 향상 시키기 위해서는 passivation timeout 값을 좀 더 크게 설정한다. 그러나 이렇게 하면 시스템 메모리를 많이 소모하게 한다.
- check timeout 과 backup trigger 에 설정값을 크게 설정한다. 그러면 backup 에 대한 호출이 적어지면서 시스템이 빨라진다. 이 방식은 백업 기능을 훨씬 적게 사용한다.
- 많은 부하가 걸리는 사이트에서는 multi session mode 를 사용한다.

9.5 결론

이상으로 JEUS server 의 관점에서의 Session Server 에 대한 설명을 마친다. 더 많은 정보를 원하면 JEUS Web Container 안내서를 참조하기 바란다.

다음 장에서는 분산 Session Server 에 대해서 다룬다.

10 분산 Session Server

10.1 소개

앞장에서 우리는 Session Server 에 대해서 살펴보았다. 이 장에서는 Session Server 와 같은 기능을 제공하는 또 다른 세션 클러스터링 방식인 분산 Session Server 를 소개한다. 분산 Session Server 는 Session Server 와 같은 세션 클러스터링 기능을 제공하면서 확장성을 개선한 방식이다. 따라서, 대규모 클러스터링 환경에서 Session Server 에 비해 더 나은 성능을 발휘할 수 있다.

10.2 분산 Session Server 의 개요

10.2.1 소개

이 절에서는 분산 Session Server 의 기본적인 내용을 다룬다.

앞서 설명한 바와 같이 분산 Session Server 가 제공하는 기본 기능은 Session Server 와 동일하다. 즉 여러 Servlet Engine 들로 구성된 클러스터링 환경에서 동일한 클라이언트에서 요청된 일련의 요청들이 특정 Servlet Engine 에서 처리되지 않더라도 세션을 계속 유지해 주는 기능을 제공한다.

기능적인 측면은 Session Server 와 동일하나 동작 방식이 Session Server 와 달리 완전 분산식이어서 Session Server 에 비해 확장성이 더 용이하다. 분산 Session Server 가 가지는 특징을 정리하면 다음과 같다.

- 여러 개의 Servlet Engine 으로 구성된 클러스터링 환경에서 지속적인 세션 유지가 가능하게 한다.
- 바로 이전의 요청을 처리하던 Servlet Engine 이 다운되더라도 다른 Servlet Engine 들이 이후 요청을 처리할 때 세션이 끊기지 않도록 해준다.
- 분산식 프로토콜을 사용하기 때문에 클러스터링 규모가 커지더라도 확장성이 용이하다.

10.2.2 분산 Session Server 의 구조

분산 Session Server 는 세션 객체를 서비스하는 서버가 각 Servlet Engine(Web Container)에 분산되어 있는 분산식 구조이다. [그림 27] 에 분산 Session Server 방식을 사용하여 4 개의 Web Container 를 세션 클러스터링하는 구조를 나타내었다. 이처럼 분산 Session Server 방식은 클러스터링에 참여하는 모든 Web Container 내에 독립적인 세션 서버가 존재하고 이들 세션 서버들이 peer-to-peer 로 다른 Web Container 의 세션 서버와 통신하여 지속적인 세션 서비스를 제공한다. 그림에 있는 화살표는 세션 서버간 소켓 연결을 의미하는데 보통의 경우 연결이 필요없으며 세션 유지를 위해 다른 Web Container 와 통신할 필요가 있을 경우 연결을 맺고 유지한다.

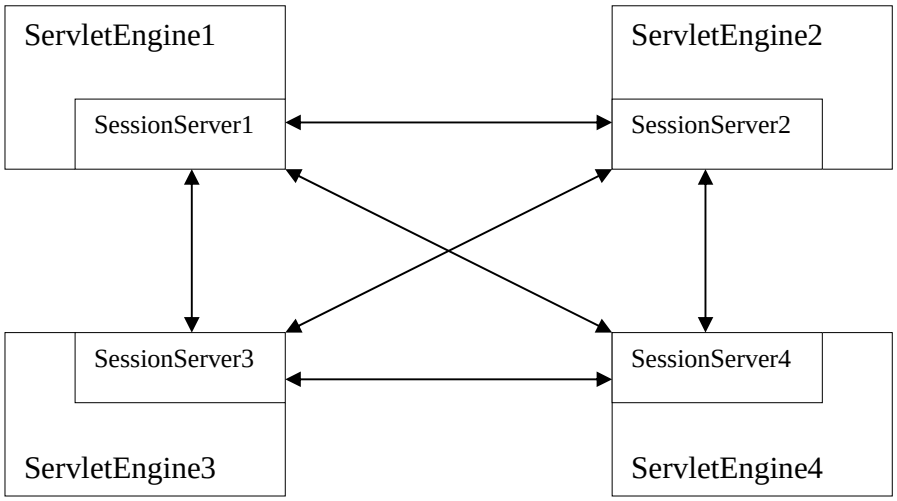

그림 27. 분산 Session Server 를 이용한 세션 클러스터링 구조

[그림 28] 은 Web Container 의 서브 컴포넌트로 동작하는 분산 Session Server 의 내부 구조이다.

- Connection Manager : 다른 Web Container 의 분산 Session Server 와 통신을 담당한다.
  - 가져올 세션 객체가 다른 Web Container 에 있을 경우 getRemoteSession 을 통해 다른 Web Container 로부터 가져온다.
  - 다른 Web Container 는 migLocalSession 을 통해 Session Manager 가 관리하는 세션 객체를 얻을 수 있다.

- 다른 Web Container 는 migBackupSession 을 통해 Backup Manager 가 관리하는 백업 세션 객체를 얻을 수 있다.

- Session Manager : 로컬에서 생성된 세션 객체를 관리한다.
  - Local Web Container 는 getLocalSession 을 통해 Session Manager 로부터 사용할 세션 객체를 얻어 오고, 수정된 세션 객체는 updateLocalSession 을 통해 Session Manager 에 전달 된다.
  - 메모리 상에 caching 된 세션(Cached Session)은 일정 주기(passivation timeout)동안 사용하지 않으면 passivateLocalSession 을 통해 file 에 저장되고 메모리상에서 지워진다.
  - 수정된 세션은 remote backup 이 설정된 경우 Backup Manager 로 전달되어 백업 서버로 지정된 다른 Web Container 의 분산 Session Server 로 in-memory 백업된다(backupToRemote).

- Backup Manager : 로컬 Web Container 를 백업으로 지정한 다른 Web Container 가 주기적으로 전송하는 백업 세션 객체를 관리한다. 이 백업 세션 객체는 원본을 가지고 있는 Web Container 에 장애가 발생한 경우 대신하여 세션을 제공한다.
  - 자신을 백업으로 지정한 remote 분산 Session Server 가 전송하는 백업 세션을 받아 저장/관리한다(backupSession).
  - 메모리 상에 caching 된 세션(Cached Session)은 일정 주기(passivation timeout)동안 사용하지 않으면 passivateBackupSession 을 통해 file 에 저장되고 메모리상에서 지워진다.
  - backupToRemote : 로컬에서 수정된 세션 객체는 조건이 만족되면 지정된 remote 분산 Session Server 로 백업된다. 여기서 조건은 두가지가 있다. 첫째는 check-to 설정으로 설정된 시간주기로 백업 프로세스가 기동된다. 두번째는 backup-trigger 설정으로 설정된 개수만큼 수정된 세션객체가 발생하면 백업 프로세스가 기동된다.

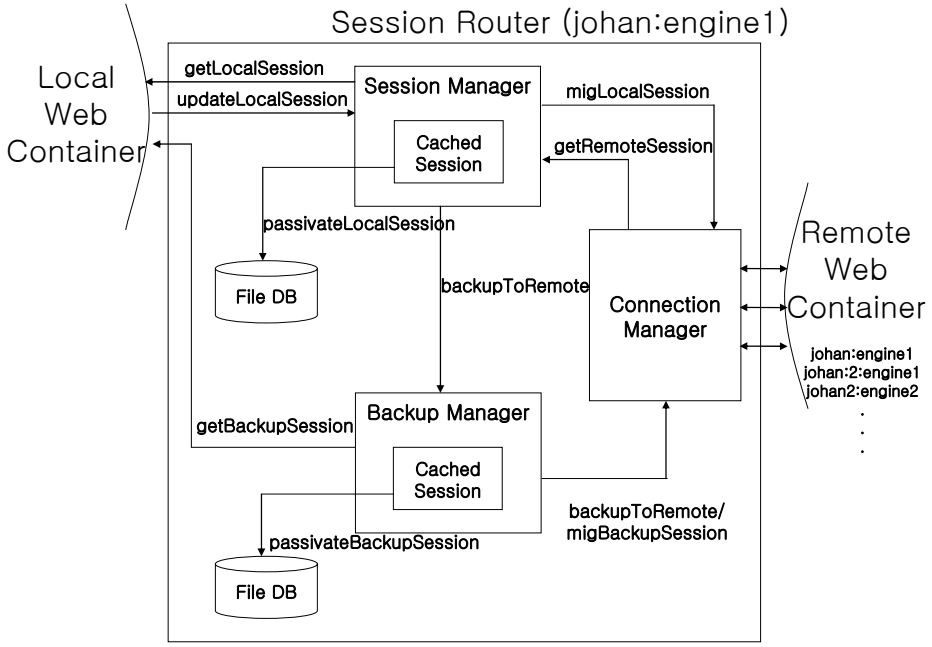

그림 28. 분산 Session Server 내부 구조

10.2.3 분산 Session Server 의 동작 방식

지금부터는 분산 Session Server 의 동작 방식을 설명하겠다.

앞서 설명하였듯이 분산 Session Server 방식은 세션 라우팅을 기본으로 한다. 먼저, 분산 Session Server 방식이 사용하는 Session Key 의 구조를 설명하겠다. 분산 Session Server 방식이 사용하는 Session Key 는 아래와 같은 형식을 갖는다.

```
<SessionID>.<primary>[&<backup>] ==

<SessionID>.<primary_engine_name>:<primary_host_name>:<primary_host_port>[&< backup
 _engine_name>:<backup_host_name>:< backup _host_port>]

예 1) XXX.engine1:johan:9736

예 2) XXX.engine1:johan:9736&engine1:johan2:9736
```

그림 29 분산 Session Server 에서 사용하는 Session Key 형식

예 2 는 backup 분산 Session Server 가 지정된 경우이고 예 1 은 그렇지 않은 경우이다. “XXX”는 <SessionID>를 상징적으로 나타낸 것이다. 실제로 <SessionID>는 이보다 훨씬 긴 random string 형태이다.

이처럼 Session Key 에는 어떤 Web Container 가 해당 세션 객체를 서비스 할 수 있는지에 대한 정보가 들어있다. 세션 라우팅을 지원하는 웹 서버를 앞단에 배치한다면 정상적인 경우 세션 객체가 존재하는 Web Container 로 요청이 routing 될 것이다. 따라서, 이러한 경우의 동작 방식은 세션 라우팅에 의한 세션 클러스터링과 동일하다(Web Container 안내서 8 장 참조).

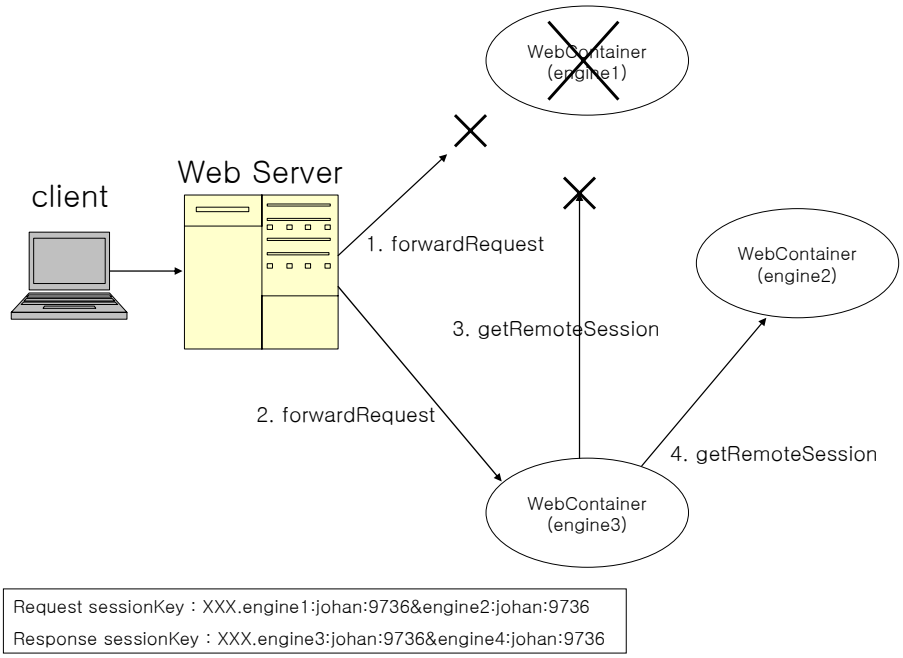

그림 30. 분산 Session Server 에 의한 fail-over 구조

웹 서버가 세션 라우팅을 지원할 수 없는 상황을 살펴보자. 웹 서버가 세션 라우팅을 지원하더라도 웹 서버에 Session Routing ID 에 해당하는 Web Container 로의 연결이 존재하지 않거나 해당 Web Container 에 장애가 발생하여 요청을 전달할 수 없는 경우가 여기에 해당한다. 이러한 경우 웹 서버는 보통 임의로 선택된 다른 Web Container 로 요청을 전달한다. [그림 30]에 이러한 상황에서 분산 Session Server 의 동작 방식을 나타내었다.

각 Web Container 는 세션 라우팅에 사용하는 Session Routing ID 를 하나씩 부여받는다. 이 ID 는 웹 서버에 Web Container 가 접속할 때 Web Container

를 구분하는 구분자로 사용된다. Session Routing ID 는 설정에 의해 자동 생성된다. [그림 30]의 3 개의 Web Container 는 다음과 같은 Session Routing ID 가 할당되었다고 가정한다.

1. engine1 : engine1:johan:9736&engine2:johan:9736
2. engine2 : engine2:johan:9736&engine3:johan:9736
3. engine3 : engine3:johan:9736&engine4:johan:9736

이제 [그림 30] 의 상황을 설명해 보겠다.

a. Request sessionKey 값의 Session Routing ID 는 engine1 Web Container 의 것이다. 이에 따라 Web Server 는 클라이언트의 요청을 engine1 Web Container 로 전달하는 것을 시도한다. 현재 engine1 에 장애가 발생하였으므로 이 시도는 실패한다.

b. Web Server 는 나머지 2 개의 Web Container 중 하나를 임의로 선택하여 클라이언트 요청을 전달하는 것을 시도한다. 선택된 Web Container 는 engine3 이며 이 요청 전달은 성공하였다.

c. Web Server 로부터 요청을 전달 받은 engine3 는 Session Routing ID 를 분석한다. 분석 결과 이 요청을 처리할 세션 객체는 engine1, engine2 에 있으며 engine1 이 primary 분산 Session Server 이고 engine2 가 backup 분산 Session Server 임을 알게 된다. 먼저 primary 분산 Session Server 가 engine1 에 접속하여 세션 객체를 가져오려고 시도한다. engine1 에 장애가 발생하였으므로 이 시도는 실패한다.

d. backup 분산 Session Server 인 engine2 로 다시 시도한다. engine2 는 백업한 세션 객체를 engine3 로 넘기고 자신의 저장 공간에는 해당 세션 객체를 지운다. 성공적으로 세션 객체를 가져온 engine3 는 이후 클라이언트의 요청을 처리하여 응답 메시지를 클라이언트에게 보낸다. 이 때 새로운 session key 를 작성하여 클라언트에게 보낸다. 이렇게 함으로써 클라이언트의 session key 가 변경되고 이 후 요청이 engine3 로 오게끔 한다. 새로운 session key 를 작성할 때는 Session Routing ID 부분만 자신의 것으로 치환한다.

10.3 분산 Session Server 설정

10.3.1 소개

분산 Session Server 를 설정하는 방법은 다음과 같이 JEUSMain.xml 의 <node > 태그 밑에 <session-router>를 추가하면 된다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    . . .
    <node>
        . . .
        <session-router-config>
           . . .
        </ session-router-config>
        . . .
    </node>
    . . .
</jeus-system>
```

10.3.2 분산 Session Server Config 설정

<session-router-config> 태그는 JEUSMain.xml 파일의 node 절 안에 하나만 존재해야 한다. 여기에는 다음과 같은 설정 항목이 있다.

- **connection timeout :** WebContainer 에 존재하는 session server 간 socket connection 을 생성할 때 적용되는 timeout 값이다.
- **read timeout** : WebContainer 에 존재하는 session server 간 socket connection 을 생성할 때 적용되는 timeout 값이다.
- **backup trigger** 는 local 세션 서버에서 세션 객체의 update 가 어느 정도 발생하였을 때 backup 세션 서버로 update 된 세션 객체들을 backup 할 지를 설정한다. 이 설정에 지정된 횟수 만큼 local 세션 서버에 update 가 발생하면 backup 을 수행한다.
- **check to** 는 backup 과정을 수행할 지를 체크하는 주기를 설정한다. 이 설정에 지정된 시간마다 update 된 세션 객체가 있는 지를 조사하고 update 된 세션 객체가 존재하면 backup 을 수행한다.

- **check level :** 사용된 session 을 remote web container 또는 local file db 에 백업하기 전에 백업할 필요가 있는지를 체크하는 것이 필요하다. 이 설정은 백업의 필요성을 체크하는 기준을 정한다. set 으로 설정하면 세션 객체에 setter 를 호출한 경우에만 백업을 하고, get 으로 설정하면 세션 객체에 getter 를 호출한 경우에도 백업을 한다. all 로 설정하면 무조건 백업을 한다.

- **default file db :** update 된 local 세션 객체를 backup 하는 방법으로는 backup 세션 서버에 backup 하는 방법외에 local 파일 시스템 상에 backup 하는 방법도 있다. 이 설정은 이와 같이 local 파일 시스템 상에 update 된 세션 객체를 backup 하는 방법을 제공한다. 실제 file backup 은 WebContainer 별로 수행되나 이 설정은 분산식 session clustering 에 참여하는모든 WebContainer(session-router)들에 default 로 동일하게 적용된다. 단, session-router 하위 element 로 "file-db"가 설정될 경우 이 설정은 무시되고 session-router 의 설정이 적용된다. 자세한 설정은 10.3.3 에서 설명한다.

- **분산 Session Server** 분산식 session clustering 에 참여할 WebContainer 를 지정하고 해당 session server 에 대한 각종 속성을 설정한다. 반드시 하나 이상이 존재해야 하며, 자세한 내용은 10.3.3 을 참조한다.

다음은 JEUSMain.xml 에 분산 Session Server config 를 설정의 일례이다.

<<JEUSMain.xml>>

```
<jeus-system>
    . . .
    <node>
        . . .
        <session-router-config>
            <connect-timeout>60000</connect-timeout>
            <read-timeout>60000</read-timeout>
            <backup-trigger>1</backup-trigger>
            <check-to>1000</check-to>
            <check-level>set</check-level>
            <default-file-db>
                <startup-clear-to>86400000</startup-clear-to>
                <passivation-to>-1</passivation-to>
                <min-hole>1000</min-hole>
                <packing-rate>0.5</packing-rate>
```

```
            </default-file-db>
            <session-router>
            . . . <!-- Next sub-section -->
            </session-router>
        </session-router-config>
        . . .
    </node>
    . . .
</jeus-system>
```

10.3.3 분산 Session Server 설정

session clustering 에 참여할 session server 에 대해 설정한다. 다음과 같은 세부 설정 항목이 있다.

- **servlet engine name :** session clustring 에 참여할 WebContainer 의 엔진 이름을 지정한다.

- **file db** : 세션 객체를 파일 시스템에 백업할 때 사용하는 설정으로 기본적인 개념은 10.3.2 에서 설명되었고, 여기에서는 하위 태그들의 의미를 살펴 본다.

  WebContainer 를 기동할 때 지정된 file 에 저장된 session 객체들이 복구되는데, 만약 현재 시간과 file 의 last modified time 의 시간차가 **startup-clear-to** 설정에 지정된 값보다 크면 복구를 시도하지 않고 file 의 내용을 모두 clear 한다. **path** 는 backup session 을 저장할 file 이름을 절대 경로로 지정한다. 기본값은 $(JEUS_HOME)/sessiondb/<node_name>_<servlet_engine_name>_1.fdb 이다. memory 에 존재하는 session 객체를 **passivation-to** 에 설정된 시간 이상 사용하지 않으면 삭제하고 대신 file-db 에 저장된 객체를 사용한다. 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 된다. **min-hole** 에 설정된 횟수 만큼 file I/O 가 발생하거나, 현재 session 객체 갯수 대비 file I/O 횟수가 **packing-rate** 에 지정된 ratio 를 넘어서면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

- **backup 분산 Session Server** 는 해당 router 의 백업으로 사용될 session-router 를 **node-name** 과 **servlet-engine-name** 으로 설정하는 태그이고, router 당 하나만 존재할 수 있다.

다음은 JEUSMain.xml 에 두 개의 session-router 를 설정한 예이다.

<<JEUSMain.xml>>

```
<jeus-system>
    . . .
    <node>
        . . .
        <session-router-config>
        . . .
        <session-router>
            <servlet-engine-name>engine1</servlet-engine-name>
            <file-db>
                <startup-clear-to>86400000</startup-clear-to>
                <passivation-to>-1</passivation-to>
                <min-hole>1000</min-hole>
                <packing-rate>0.5</packing-rate>
            </file-db>
            <backup-session-router>
                <node-name>tmaxh4</node-name>
                <servlet-engine-name>eng2</servlet-engine-name>
            </backup-session-router>
        </session-router>
        <session-router>
            <servlet-engine-name>engine2</servlet-engine-name>
            <file-db>
                <startup-clear-to>86400000</startup-clear-to>
                <passivation-to>-1</passivation-to>
                <min-hole>1000</min-hole>
                <packing-rate>0.5</packing-rate>
            </file-db>
            <backup-session-router>
                <node-name>tmaxh4</node-name>
                <servlet-engine-name>eng1</servlet-engine-name>
            </backup-session-router>
            </session-router>
        </session-router-config>
        . . .
    </node>
    . . .
```

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

10.4 결론

분산 Session Server 방식은 대규모 클러스터링 환경에 적합한 세션 클러스터링 방식이다. 소규모 클러스터링 환경에서는 Session Server 방식을 대규모 클러스터링 환경에서는 분산 Session Server 방식을 사용할 것을 권장한다.

다음 장에서는 Engine Container 에 대해서 다룬다.

11 Engine Container

11.1 소개

이번 장에서는 JEUS 노드의 Engine Container 구성요소에 대해 좀 더 깊이 알아본다 [그림 30]. 하나의 Engine Container 는 하나 이상의 JEUS Engine 들과 네이밍, 보안, 트랜잭션 관련 서비스들을 서로 묶는데 사용되는 논리적인 구성요소이다.

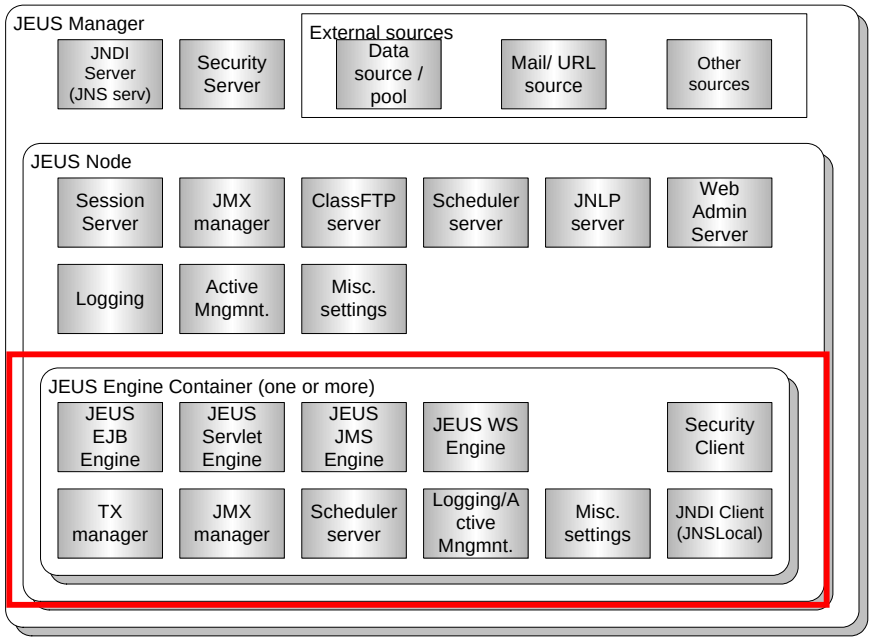

그림 31. Engine Container 가 표시된 JEUS 구조

11.2 Engine Container 의 개요

11.2.1 소개

구조상으로 JEUS Engine Container 는 JEUS Engine 들이 사용하게 될 기반 구조와 서비스들을 제공하는 많은 하위 구성요소들로 구성된다

Engine Container 들이 논리적으로는 JEUS 노드의 하위 구성요소로서 여겨질 수 있지만, 물리적으로는 JEUS 노드/manager 가 실행되는 JVM 과는 별도

의 JVM 에 존재한다. 앞으로 설명하게 될 “default” Engine Container 를 사용하지 않는 한 항상 이렇다.

11.2.2 주요 하위 구성요소들

아래의 주요 구성요소들은 각각의 Engine Container 에 설정된다.

- Engine
- JNDI 클라이언트
- Security 클라이언트
- 트랜잭션 매니저
- JMX 매니저
- Scheduler Server
- 시스템 로깅
- 유저 로깅
- e-mail 통보
- Invocation Manager.

이 후 절들에서 각각의 구성 요소들에 대해 간략히 알아볼 것이다.

11.2.3 Engine

Engine 들은 JEUS Server 의 핵심 구성 요소이다. 이것들은 J2EE 어플리케이션 을 실행하고 런타임 환경을 제공한다.

각 Engine Container 에 설정할 수 있는 종류에는 4 가지가 있다.

1. EJB Engine
2. Servlet Engine
3. JMS Engine
4. WS Engine (JEUS WebServer)

참고: 각 Engine 종류는 최대 하나만이 각 Engine Container 에 설정될 수 있다. 그러므로 같은 종류의 Engine 을 2 개 이상, 하나의 Engine Container 에 추가할 수 없다.

이 Engine 구성요소들은 13 장에서 설명한다.

11.2.4 JNDI 클라이언트

JEUS 시스템에서 JNDI Naming Service 에 관한 자세한 정보는 5 장을 참고하기 바란다.

11.2.5 Security 클라이언트

JEUS 시스템에서 보안 서비스에 관한 자세한 정보는 JEUS Security 안내서를 참조 하기 바란다.

11.2.6 트랜잭션 매니저

트랜잭션 매니저는 12 장에서 설명한다.

11.2.7 JMX 매니저

JMX 매니저는 JEUS JMX 안내서 참조하기 바란다.

11.2.8 Scheduler Server

Scheduler Server 는 JEUS Scheduler 안내서를 참조 하기 바란다.

11.2.9 시스템 로깅

JEUS 시스템에서의 각 Engine Container 는 노드 처럼 실행 시간 동안 발생하는 모든 사건들을 기록한다. 자세한 설명은 14 장을 참고하기 바란다.

11.2.10 User Logging

각 Engine Container 는 각자의 User logging 을 처리한다. EJB 와 같은 엔터프라이즈 어플리케이션에서 프로그래머가 정의한 코드가 생성하는 한 어떠한 종류의 로그 메시지라도 User logging 을 사용하여 저장할 수 있다.

어플리케이션 코드에서 그러한 메시지들을 생성하는데 사용되는 클래스(API)는 `jeus.util.UserLog` 이다. 하지만 J2SE 1.4 이상에서 JEUS 를 사용한다면 logging API 를 사용해서 Application 에서 필요한 logging 을 남기는 것이 좋다.

11.2.11 e-mail 통보

만약 Engine Container 레벨에서 user log 에 <smtp-handler>가 설정되면 생성될 메시지들이 지정된 수신자로 e-mail 을 통해 전송된다. 또한 User log 메시지들은 위에서 설명한 데로 log file 에 저장된다.

11.2.12 Invocation Manager

Method invocation manager 는 Engine Container 에서 Servlet/JSP methods, state less EJB methods 그리고 MDB methods 와 같은 Stateless 메소드를 호출 하는 동안 사용하는 외부 자원(external resource)을 추적하고 보고한다.

이 구성요소를 위해 3 가지 모드 중 하나를 선택할 수 있다.

- NoAction: 동작을 하지 않는다.
- Warning: 이 선택사항이 선택되면, 만약 한 자원이 Stateless 메소드 호출 동안 사용되었지만 반환할 때 닫지 않게 될 경우 이벤트가 container log 에 warning 메시지로 기록될 것이다.
- AutoClose: 이 선택사항이 선택되면, 만약 한 자원이 Stateless 메소드 호출 동안 사용되었으나 반환할 때 닫히지 않는다면 자원이 자동적으로 닫힌다.

설정하는 법을 설명할 때 어떻게 모드를 선택하는지 자세히 설명한다.

11.2.13 Database Mapping

각 엔진 컨테이너별로 deploy 된 어플리케이션을 위해서 각각 할당된 DB 가 필요할 경우가 있다. 이 기능은 어플리케이션에서 사용되는 데이터소스의 이름을 실제 데이터소스로 링크하는데 사용된다. 이 기능은 각 엔진 컨테이너가 각각 다른 DB 를 사용하기 위해서 필요하다.

예를 들어보자. 어플리케이션인 “APP”는 “A”라는 export-name 을 사용해서 데이터소스를 lookup 한다. 그런데 어떤 이유로 인해서 각 엔진 컨테이너는 전용의 DB 하나씩을 사용해야 한다고 가정하자. 그리고 동일한 어플리케이션인 “APP” 가 deploy 된 “C1”, “C2” 두 개의 엔진 컨테이너가 있다. “C1”에서 호출하는 데이터소스 lookup 은 실제 “B”라는 데이터소스를 lookup 해야 하고, “C2”에서는 “C”라는 데이터소스를 lookup 해야 한다.

이 기능을 사용하면 논리적인 데이터소스인 “A”(실제 “APP”가 사용)를 lookup 하면, 컨테이너 “C1”을 위한 “B”로 매핑된다[그림 32].

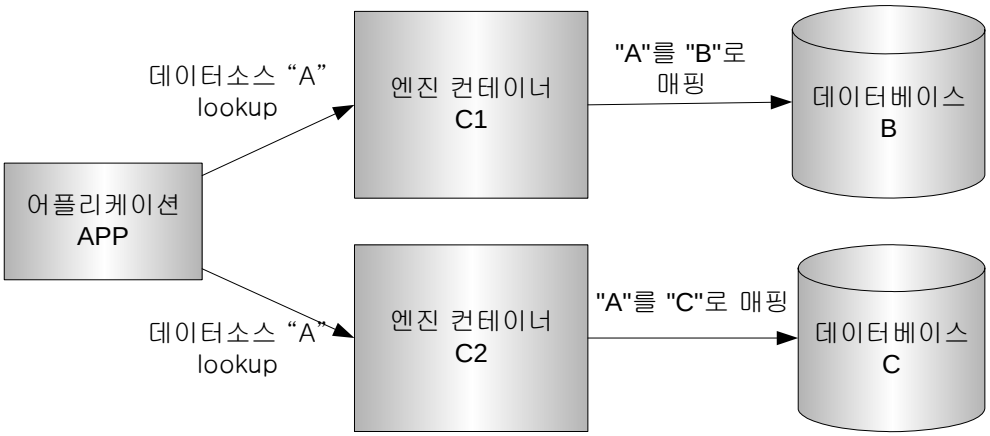

그림 32. 엔진 컨테이너 범위의 데이터베이스 매핑

11.2.14 Default Engine Container

보통 모든 Engine Container 는 JEUS Manager 가 실행되는 JVM 과는 다른 JVM 에서 실행된다. 이 것은 Engine Container 나 혹은 그것에 포함된 Engine 들에 문제가 발생할 경우 JEUS Manager 에 아무런 피해를 주지 않음을 암시한다.

그러나 명시적으로 Engine Container 의 이름을 “default”로 정함으로써, JEUS Manager 와 같은 JVM 에서 Engine Container 를 실행 시킬 수 있다. 이런 설정은 EJB 나 Servlet 디버깅을 위해 사용된다. JEUS Manager JVM 을 디버그 모드로 시작하고 표준 JPDA 를 사용하여 붙임으로써 JEUS Manager JVM 에 존재하는 어떤 구성요소라도 디버깅이 가능하다. 이것이 “default” 선택사항이 존재하는 이유이다.

보다 자세한 것은 부록 A 를 참고하기 바란다.

11.2.15 Engine Container 디렉토리 구조

Engine Container 는 논리적 구성요소이기 때문에 물리적인 디렉토리 구조가 존재하지 않는다.

11.2.16 결론

Engine Container 의 주요 특징들에는 JEUS Engines, Transaction manager, security client 그리고 naming client 등이 있다. 위에서 보았듯이 대부분의 개념은 이 매뉴얼에서 설명되었다.

또한 디버깅 시 유용한 “default” Engine Container 에 대해 간략히 설명하였다.

다음 절은 Engine Container 의 설정에 초점을 맞추어 설명한다.

11.3 Engine Container 설정

11.3.1 소개

Engine Container 를 설정하는 것은 간단하다. 단순히 <engine-container> 요소만 JEUSMain.xml 의 <node>요소 밑에 추가하면 된다.

<<*JEUSMain.xml*>>

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

11.3.2 Engine Container 의 기본 설정

Engine Container 의 기본 설정은 아래와 같다.

- **name**: 임의적으로 Engine contianer 의 이름을 선택할 수 있다.

  참고: 만약 이름이 “default”라고 설정되면 Engine Container 와 모든 하위 구성요소들은 JEUS Manager 와 같은 JVM 에서 동작하게 된다. 이것은 여러분들이 EJB 나 Servlet 기반의 어플리케이션의 디버깅을 위해 JPDA 디버거를 JEUS Server 에 붙일 때 필요하다. JEUS 시스템의 디버깅에 관한 보다 자세한 정보는 부록 A 를 참고하기 바란다.

- **command option**: Engine Container 를 실행하기 위해 개별적인 JVM 에 추가할 파라미터들을 선언하는데 사용된다. 여기서 지정 가능한 JEUS 파라미터들의 목록이 부록 F 에 있다. 표준 JVM 파라미터들도 설정 가능하다.

- **sequential start** : Boolean 값으로 만약 “true”라고 설정하면 , 이 container 에 존재 하는 모든 Engine 들이 순차적으로 시작된다. 예를 들어 WS Engine 은 반드시 WS Engine 이 사용하는 Servlet Engine 전에 시작되어야 한다. 따라서 JEUSMain.xml 에서 Servlet Engine 전에 WS Engi

ne 을 정의 하고 sequential start 값을 “true”라 선택하면 WS Engine 이 먼저 부팅된다.

- **user class path**: Engine Container 가 사용하는 JVM 에 사용자 정의 클래스 패스를 추가한다.

아래는 Engine Container 의 설정의 예제이다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
    <node>
        ...
        <engine-container>
            <name>mycontainer</name>
            <command-option>-Xms64m -Xmx128m</command-option>
            <sequential-start>true</sequential-start>
            <user-class-path>
                c:\mylib\classes;c:\mylib\lib\mylib.jar
            </user-class-path>
            ...
        </engine-container>
        ...
    </node>
    ...
</jeus-system>
```

11.3.3 Engines, Transaction Manager, JMX Manager 그리고 Scheduler Server 의 설정

각 항목을 설정하기 위해서는 각각 해당하는 장을 참조하기 바란다. 아래 표에서 이 구성요소들과 설정을 위해 필요한 XML 요소와 자세한 정보를 위해 참고할 장을 표시하였다[표 5].

표 5. Engine Container 의 하위 구성요소와 XML 요소 태그.

구성 요소	XML 요소 이름	장
Engines	<engine-command>	13 장

구성 요소	XML 요소 이름	장
Transaction Manager	<tm-config>	12 장
JMX Manager	<jmx-manager>	JEUS JMX 안내서 참조
Scheduler Server	<scheduler>	JEUS Secheduler 안내서 참조

Scheduler Server 의 설정은 매우 간단하다. Engine Container 에서 Scheduler Server 를 실행하기 위해서 단순히 Boolean 값으로 “true”만 설정하면 된다.

기본설정을 포함한 XML 부분이 아래에 있다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    ...
    <node>
        ...
        <engine-container>
            <name>mycontainer</name>
            <command-option>-Xms64m -Xmx128m</command-option>
            <sequential-start>true</sequential-start>
            <user-class-path>
                c:\mylib\classes;c:\mylib\lib\mylib.jar
            </user-class-path>
            <security-switch>true</security-switch>
            <engine-command>
                ...
            </engine-command>
            <tm-config>
                ...
            </tm-config>
            <jmx-manager>
                ...
            </jmx-manager>
            <scheduler>
                true
```

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

11.3.4 Logging 설정

Engine Container 의 로그 설정은 JEUSMain.xml 에서 <engine-container> XML 요소의 바로 밑에 <system-logging>에 설정한다. 또한 User logging 의 설정은 <user-logging>에 설정한다. 자세한 설정 방법은 14 장을 참고하기 바란다.

11.3.5 Invocation Manager 설정

앞서 언급했듯이 Method invocation manager 는 Engine Container 에서 Servlet/ JSP methods, stateless EJB methods 그리고 MDB methods 와 같은 Stateless 메소드를 호출 하는 동안 사용하는 외부 자원(external resource)을 추적하고 보고한다.

Method invocaiton manager 는 JEUSMain.xml 의 각 Engine Container 요소 밑에 하나의 <invocation-manager-action> 으로 설정된다.

- NoAction: 기능을 사용하지 않는다.
- Warning: 이 선택사항이 선택되면, 만약 한 자원이 무상태 메소드 호출 동안 사용되었지만 반환할 때 닫지 않게 될 경우 이벤트가 container log 에 warning 메시지로 기록된다.
- AutoClose: 이 선택사항이 선택되면, 만약 한 자원이 무상태 메소드 호출 동안 사용되었으나 반환할 때 닫히지 않는다면 자원이 자동적으로 닫힌다.

예제:

<<*JEUSMain.xml*>>

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

```
            ...
            <invocation-manager-action>
                AutoClose
            </invocation-manager-action>
            ...
        </engine-container>
        ...
    </node>
    ...
</jeus-system>
```

위의 XML 부분은 무상태 메소드 호출 후 반환된 열려진 자원을 Invocation manager 가 자동으로 닫도록 한다.

11.3.6 Database Mapping 설정

엔진 컨테이너 단위로 데이터베이스 매핑을 하려면 JEUSMain.xml 의 `<engine-container><res-ref><jndi-info>` 태그를 사용해서 설정한다. 이 태그에는 다음을 설정한다

- **ref-name:** 태그는 데이터소스를 위한 레퍼런스 이름을 선언한다. 이 이름이 어플리케이션 코드에서 lookup 할 때 사용하는 이름이다.
- export-name: 태그에는 실제 작업할 데이터소스의 export-name 을 지정한다. 실제 datasource/database 의 JNDI 이름이다.

위 태그에서 정의된 매핑은, 해당 엔진 컨테이너에서만 적용된다.

<<JEUSMain.xml>>

```
<jeus-system>
    <node>
        . . .
        <engine-container>
            . . .
            <res-ref>
                <jndi-info>
                    <ref-name>A</ref-name>
                    <export-name>B</export-name>
                </jndi-info>
            </res-ref>
        </engine-container>
        . . .
```

```
    </node>
    . . .
</jeus-system>
```

11.3.7 결론

지금까지 JEUS 에서 Engine Container 를 어떻게 설정하는지 알아보았다. 대부분의 설정은 이 매뉴얼의 다른 장에서 설명된다.

다음은 JEUS Engine Container 를 어떻게 제어하는지 알아 보겠다.

11.4Engine Container 의 제어

11.4.1 개요

Engine Container 는 jeusadmin 나 웹 관리자로 각각 제어가 가능하다. 웹 관리자를 이용한 제어는 JEUS 웹 관리자 안내서를 참조 하기 바란다.

11.4.2 Jeusadmin 콘솔 툴을 이용한 Engine Container 제어

jeusadmin 에서 비활성화된 Engine Container 를 시작하기 위해서 “startcon”을 사용하고, 활성화된 Engine Container 를 정지 시키기 위해 “downcon” 명령을 사용할 수 있다.

11.5Engine Container 모니터링

11.5.1 개요

Engine Container 는 jeusadmin 나 웹 관리자로 각각 모니터링이 가능하다. 웹 관리자를 이용한 제어는 JEUS 웹 관리자 안내서를 참조 하기 바란다.

11.5.2 Jeusadmin 을 이용한 모니터링

jeusadmin 툴에서 “conlist” 명령을 수행함으로써 활성화된 Engine Container 목록을 볼 수 있으며 “englist” 명령을 통해 특정 Engine Container 에 속한 Engine 들의 모든 이름을 볼 수 있다(부록 B 참고).

11.6 Engine Container 튜닝

만약 높은 성능이 필요하다면 다음의 Engine Container 설정에 대해 생각해 보야 한다.

- Sequential Start 를 사용하지 말아야 한다. 때에 따라서는 Sequential Start 가 필요하기도 하지만 보통은 Engine Container 의 부트시간이 길어지기 때문이다.
- 만약 Engine Container 안의 Engine 에서 동작하는 어플리케이션이 보안 인증과 권한 검사를 필요로 하지 않는다면 보안기능을 사용하지 않도록 설정한다.
- 만약 invocation manager 가 필요 없다면 비활성화로 설정한다.

Engine Container 튜닝은 8 장에서 설명한 노드 튜닝과 많은 부분이 같다. 왜냐하면 Engine Container 와 노드는 설정을 할 때 많은 유사한 점이 있기 때문이다.

11.7 결론

JEUS Engine Container 에 관한 설명을 마쳤다.

Engine Container 가 JEUS Engine 의 시작과 동작을 관리한다. Engine Container 는 보통 JEUS Manager 와는 분리된 JVM 에서 실행된다. 이 규칙의 예외는 디버깅 목적으로 “default” Engine Container 를 사용할 때이다.

또한 구성요소를 어떻게 제어하고 모니터링하며 튜닝 하는지에 대해 간단히 알아 보았다.

다음 장에서는 JEUS Transaction manager 에 대한 간략히 설명한다.

12 Transaction Manager

12.1 소개

엔터프라이즈 시스템에서 트랜잭션 서비스는 대용량의 클라이언트 요청을 안전하고 효율적으로 처리하는데 기본적이고 중요한 서비스이다. 그리고 인터넷 상의 서비스는 점점 복잡해지고 있으며, 트랜잭션 서버에 대한 기능적인 요구도 점점 증가하는 추세이다.

트랜잭션에 관련되는 작업은 여러 개의 리소스를 사용하게 된다. 하나의 리소스를 사용하는 작업은 LocalTransaction 으로 처리할 수 있는 반면에, 여러 개의 리소스를 사용하는 작업은 트랜잭션 매니저에 의해서 하나의 Global Transaction 으로 관리되게 된다.

JEUS 의 트랜잭션 매니저는 Java Transaction Service(JTS)와 완벽한 호환성을 제공하며, Java Transaction API(JTA)를 완전 지원한다. J2EE 어플리케이션 서버의 어플리케이션에서 트랜잭션의 시작과 끝을 지정하거나, 아니면 Web Container 나 EJB Container 에 트랜잭션 지정을 맡길 수 있다.

JEUS 트랜잭션 매니저는 AP 가 EJB Container 나 Web Container, Client Container 에서 트랜잭션 작업을 할 수 있도록 지원한다. container 는 데이터베이스나 메시지 서버 같은 리소스 매니저(RM)를 JDBC, JMS 를 통해서 연결을 제공한다. 어플리케이션이 리소스 매니저를 사용해서 작업을 하면, 트랜잭션 매니저가 이를 알아채고 Global Transaction 에서 운영되도록 관리해서 COMMIT 이나 ROLLBACK 이 되도록 한다.

본 매뉴얼의 구성은 다음과 같다. 절 12.2 는 JEUS 트랜잭션 매니저의 기능에 대해서 설명하고, 그 설정 사항은 절 12.3 에서 설명한다. LocalTransaction 과 client managed transaction, container managed transaction 은 절 12.4 에서 설명한다. 트랜잭션의 무결성을 보장하는 fail-over 메커니즘은 절 12.6 에서 설명한다.

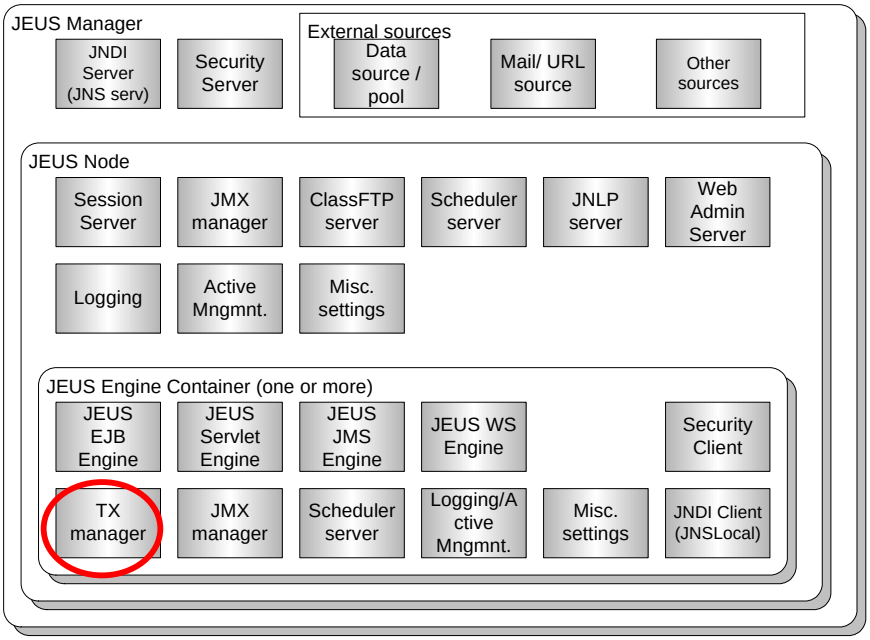

그림 33. JEUS Transaction Manager

12.2 JEUS Transaction Manager 의 개요

12.2.1 소개

JEUS 에서, 어플리케이션은 Web Container 나 EJB Container, Client Container 를 통해서 다양한 리소스매니저로 접속할 수 있다. 어플리케이션에서는 작업의 시작과 끝을 정의해서 LocalTransaction 이나 Global Transaction 을 사용할 수 있다. Global Transaction 의 경우에는 JEUS 트랜잭션 매니저가 트랜잭션의 시작과 끝 사이에 사용된 리소스 매니저를 추적해서 작업을 조율하게 된다.

본 절에서는 우선 어플리케이션이 선택할 수 있는 트랜잭션에 대해서 이야기한다. Global Transaction 을 선택했을 때 JEUS 트랜잭션 매니저는 트랜잭션 관련 실시간 정보를 관리한다. 두 개의 트랜잭션 타입에 대해서는 다음 절에서 다룬다. 트랜잭션 작업이란 트랜잭션이 가능한 리소스 매니저에서 작업하는 것을 말하는데, 이들 트랜잭션 리소스 매니저와 JEUS 간의 상호 작용은 마지막 절에서 소개한다.

12.2.2 트랜잭션 어플리케이션

LocalTransaction

리소스 매니저를 사용하는 트랜잭션 작업들을 하나의 LocalTransaction 으로 묶을 수 있다. 어플리케이션은 리소스 매니저의 드라이버를 사용해서 LocalTransaction 을 시작하거나 종료한다.

기본적으로 JEUS 트랜잭션 매니저는 LocalTransaction 처리에는 관여하지 않는다. 어플리케이션이 작업을 Global Transaction 으로 처리할 수 있지만, LocalTransaction 이 훨씬 효율적이므로 LocalTransaction 을 사용하길 권장한다.

Global Transaction

두 개 이상의 리소스 매니저를 사용하는 일련의 트랜잭션 작업은 하나의 Global Transaction 으로 묶어서 처리할 수 있다. 어플리케이션은 JTA 를 사용해서 Global Transaction 을 시작하며, JEUS 트랜잭션 매니저는 트랜잭션의 범위를 알아채서, 여러 리소스 매니저로 분산된 작업이 적절하게 작동되어 정확한 결과가 나오도록 조정한다. 단, JEUS 트랜잭션 매니저가 작업을 관리하기 위해서는, 반드시 JEUS 커넥션 매니저의 커넥션을 사용해서 트랜잭션 작업을 해야 한다.

트랜잭션 commit 시에 JEUS 트랜잭션은 어플리케이션의 실행 흐름을 추적해서 1 Phase Commit 을 할지 2 Phase Commit 을 할지, 아니면 3 Phase Commit 을 할지를 결정한다. 이 결정은 트랜잭션 내에 사용된 리소스 매니저의 개수와 타입에 따라서 결정된다. JEUS 트랜잭션은 중복된 트랜잭션을 지원하지 않으며, 두 개의 트랜잭션이 섞여서 진행하는 것은 허락하지 않는다.

Bean Managed Transaction

Bean managed transaction 에서는 어플리케이션이 *javax.transaction.User Transaction* 클래스를 사용해서 트랜잭션의 시작과 종료 범위를 지정한다. 트랜잭션을 commit 할 것인지 rollback 할 것인지는 전적으로 어플리케이션에 달려있다. 그러나 commit 을 했음에도 불구하고, 리소스 매니저가 트랜잭션을 처리할 수 없을 때 JEUS 트랜잭션 매니저가 rollback 시킬 수도 있다.

Container Managed Transaction

Container managed transaction 에서는 EJB Container 가 메소드에 설정된 트랜잭션 속성에 맞춰서, 트랜잭션의 시작과 종료, 중지와 재시작을 관리한다. EJB 배치 시에 각 메소드 별로 트랜잭션 속성 NotSupported, Required, Supports, RequiresNew, Mandatory, Never 를 세팅한다.

12.2.3 JEUS Transaction Manager

JEUS 는 두 가지 트랜잭션 매니저를 제공한다. 서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저가 그것이다. 서버 트랜잭션 매니저는 Global Transaction 을 관리하기 위한 모든 기능을 제공한다. 반면에 클라이언트 트랜잭션 매니저는 서버 트랜잭션의 proxy 역할만을 한다.

Server Transaction Manager

서버 트랜잭션은 Java Transaction Service 를 완전 구현했으며, Web Container 와 EJB Container, Client Container 에서 동작한다. Global Transaction 이 시작된 Container 의 서버 트랜잭션 매니저는 Global Transaction 과 관련된 모든 정보를 수집해서, 트랜잭션의 Root Coordinator 로서 작업을 진행한다.

만약 Global Transaction 이 하나의 container 를 벗어나 다른 container 와 트랜잭션 작업을 한다면, 새로운 container 의 서버 트랜잭션 매니저는 Root Coordinator 의 proxy 로 작동하게 되며, Root Coordinator 와 같은 역할을 자신의 container 에서 하게 된다. 이를 Sub Coordinator 라고 한다.

Engine Container 의 Server Transaction Manager

Engine Container 위에 서버 트랜잭션이 동작하면서, Engine Container 의 어플리케이션이나 EJB Container 가 시작한 Global Transaction 관련 작업을 관리한다. Global Transaction 이 Engine Container 를 벗어나 다른 Container 와 작업을 할 때, 다른 트랜잭션의 트랜잭션 매니저로 트랜잭션 정보를 전송(propagation)하게 되고, 이를 바탕으로 트랜잭션의 작업을 추적하게 된다. 이와 같이 트랜잭션 매니저 간에 메시지와 정보를 교환함으로써 두 개 이상의 Engine Container 에서 작동하는 트랜잭션을 일관성 있게 관리하게 된다.

Client Container 의 Server Transaction Manager

J2EE 스펙에서는 Client Container 에서의 트랜잭션 지원을 정의하고 있지 않다. 그러나, Global Transaction 으로 관리되는 작업을 어플리케이션에서는 많이 하게 된다. Client Container 에서의 서버 트랜잭션 매니저는 JEUS 에서 제공하는 모든 트랜잭션 리소스 매니저를 사용해서 작업할 수 있다. 또한 클라이언트의 실패를 복구할 수 있는 기능을 제공하므로 Global Transaction 의 무결성을 보장한다.

Client Transaction Manager

클라이언트 트랜잭션 매니저는 서버 트랜잭션 매니저의 proxy 로 작동한다. Global Transaction 의 신호를 가장 먼저 받은 서버 트랜잭션 매니저가 Root Coordinator 의 역할을 하게 되며, 해당 트랜잭션에 관한 정보를 모두 모아서 처리한다. commit 시점에 클라이언트 트랜잭션 매니저는 commit 신호를 Root Coordinator 에 전송하고, 복잡한 내용을 알 필요 없이 commit 의 결과만을 받게 된다.

12.2.4 Resource Manager

JEUS 에서 어플리케이션은 다양한 리소스 매니저와 작업을 할 수 있다. 커넥션 매니저는 리소스 매니저와의 연결을 관리한다. JEUS 에서는 4 가지 타입의 커넥션 매니저를 제공한다. JDBC connection manager 와 JMS connection manager, WebT connection manager , 그리고 Connector manager 가 그것이다. Global Transaction 을 위해서 커넥션 매니저는 트랜잭션과 관련되는 정보를 트랜잭션 매니저에 보고한다.

JDBC RM

Java Database Connectivity (JDBC)는 Java 언어로 개발할 때 사용하는, 데이터베이스와 어플케이션간의 연결을 정의한 것이다. 어플리케이션은 j*avax.sql.DataSource* 클래스를 사용해서 JDBC 커넥션을 Naming Server 로부터 lookup 해서 사용한다. 자세한 것은 5 장과 7 장을 참조한다.

- **DataSource 와 PooledConnection DataSource**: 이들 데이터소스를 사용해서는 Global Transaction 을 사용할 수 없다. 만약 동일한 리소스 매니저를 사용한다면 LocalTransaction 으로 작업을 관리할 수 있다.

- **XA DataSource**: 이 데이터소스를 사용하면 작업을 Global Transaction 으로 묶어서 처리할 수 있다. 어플리케이션이 커넥션을 요청할 때, 그 커넥션은 트랜잭션 매니저에 등록이 되어서, Global Transaction 에 참여하게 된다. 그리고 해당 리소스 매니저와 제어 메시지를 통신하기 위한 채널을 제공한다.

- **LocalXA DataSource**: 이 데이터소스는 Global Transaction 의 기능을 제공하는 PooledConnection DataSource 이다. 또, XA DataSource 를 통해서 다른 작업과 연계될 수 있지만, LocalXA DataSource 는 하나의 Global Transaction 내에서 최대 한 개만 사용될 수 있다. 만약 한 개 이상의 LocalXA DataSource 가 사용되면 Global Transaction 의 무결성을 보장할 수 없기 때문에, JEUS 트랜잭션 매니저가 Exception 을 발생시킨다.

JMS RM

Java Message Service (JMS)는 Queue 와 Topic 그리고 거기에 저장된 메시지에 액세스하기 위한 API 을 정의한 스펙이다. 자세한 내용은 JEUS JMS 가이드나 JMS 스펙을 참조하기 바란다(http://java.sun.com/j2ee/docs).

- **JEUS JMS Server**: JEUS JMS Server 는 트랜잭션 매니저와 커넥션이 통합된 모듈을 가지고 있다. 그래서 메시지를 보내거나 받는 작업은 모두 LocalTransaction 이나 Global Transaction 으로 처리될 수 있다. Global Transaction 으로 작업을 하기 위해서는, 어플리케이션이 JEUS JMS 설정 파일에 있는, *javax.jms.XAConnectionFactory* 을 구현한 Connection Factory 를 사용해야만 한다. JEUS JMS 에 대한 내용은 JEUS JMS 가이드와 본 매뉴얼의 13 장을 참조한다.

- **IBM MQ**: JEUS 의 트랜잭션 서비스는 IBM MQ 와의 통합 서비스를 제공한다. JEUS 설정 파일에서 IBM MQ 에 있는 QUEUE 와 TOPIC 을 JEUS JNDI 서버로 등록할 수 있다. 또한 *javax.jms.XAConnectionFactory* 를 구현한 IBM MQ 객체를 통해서 Global Transaction 작업을 할 수도 있다. IBM MQ 에 대한 자세한 내용은 IBM MQ 을 참조한다.

Tmax (WebT) RM

Tmax 는 Tmax Soft 에서 개발된 TP-monitor 이다. JEUS 에서는 Tmax 의 트랜잭션 매니저와 투명한 양방향 서비스를 위해서 WebT 라는 브리지를 제공한다. WebT 는 양방향 메소드 호출을 지원한다. 그래서 Tmax 위에서 일반적인 EJB client 가 작동될 수 있으며, JEUS 에 있는 EJB 빈을 동일한 방법으로 호출할 수 있다. 자세한 것은 JEUS WebT 가이드를 참조한다.

Java Connector Architecture (JCA) RM

JCA 는 J2EE 스펙 중의 하나로서, J2EE 호환 플랫폼과 다양한 EIS(Enterprise Information System) 과의 통합 메커니즘을 제공한다. XA 트랜잭션을 지원하는 커넥터를 사용해서, 어플리케이션이 다양한 트랜잭션 작업을 하나의 Global Transaction 으로 다룰 수 있다. 자세한 내용은 JEUS JCA 안내서를 참조한다.

12.2.5 결론

이번 절에서는 JEUS 에서 트랜잭션의 개념에 대해서 소개했다.

우선 LocalTransaction 과 Global Transaction, bean-managed transaction 과 container-managed transaction 에 대해서 보았다. 그 다음으로 트랜잭션 매니저의 두 가지 타입인 서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저에 대해서 이야기했다. 끝으로 트랜잭션이 가능한 리소스 매니저인 JDBC 와 JMS, WebT 그리고 J2EE 커넥터에 대해서 간략하게 보았다.

다음 절에는 JEUS 에서 서버 트랜잭션 매니저를 어떻게 설정하는지를 소개한다.

12.3 Server Transaction Manager 설정

12.3.1 소개

JEUS 는 두 가지 종류의 트랜잭션 매니저를 제공한다. 바로 서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저이다. 서버 트랜잭션 매니저는 Java Transaction API(JTA)를 JEUS 에서 완전 구현한 것이다. 서버 트랜잭션 매니저는 Web Container 나 EJB Container 또는 Client Container 에서 작동한다.

클라이언트 트랜잭션 매니저는 Client Container 의 어플리케이션이 Global Transaction 을 시작하기 위해서 만들어졌다. 이 기능은 단지 서버 트랜잭션 매니저로 트랜잭션의 시작과 끝을 나타내는 메시지만 전달 한다. 이 메시지를 가장 먼저 받은 트랜잭션 매니저가 Root Coordinator 로 작동하게 된다. 즉, Global Transaction 에서, 클라이언트가 가장 먼저 액세스하는 EJB 빈의 JVM 에서 작동하는 서버 트랜잭션 매니저가 Root Coordinator 가 된다. 해당 Global Transaction 의 모든 정보는 그것을 관리하는 Root Coordinator 가 모아

서 commit 할 때 사용된다. 클라이언트 트랜잭션 매니저의 JVM 에서 사용된 트랜잭션 작업은 Global Transaction 과는 연관성이 없다.

서버 트랜잭션 매니저의 설정은 JEUSMain.xml 의 각 <engine-container> 태그에 있는 <tm-config> 태그에서 한다. 다음 예를 참고한다.

<<JEUSMain.xml>>

```
<jeus-system>
    <node>
        ...
        <engine-container>
            ...
            <tm-config>
                ...
            </tm-config>
            ...
        </engine-container>
        ...
    </node>
    ...
</jeus-system>
```

이 내용을 어떻게 설정하는지 보기 전에, 클라이언트용 트랜잭션 매니저를 어떻게 선택할지부터 봐야 한다.

12.3.2 트랜잭션 매니저 타입 설정

Web Container 와 EJB Container 에서는 서버 트랜잭션 매니저만 사용되는 반면에, Client Container 에서는 서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저 둘 다 사용된다. Client Container 에서 클라이언트 트랜잭션 매니저를 사용하려면 다음을 클라이언트 어플리케이션의 스크립트에 추가한다.

```
-Djeus.tm.version=client
```

12.3.3 트랜잭션 매니저의 TCP 포트 설정

서버 트랜잭션 매니저는 다른 트랜잭션 매니저와 통신하기 위해서 TCP/IP 포트를 사용한다. Web Container 나 EJB Container 의 서버 트랜잭션 매니저에는 자동적으로 포트가 하나 할당된다. 클라이언트의 서버 트랜잭션 매니저에서는 스스로 적당한 포트를 하나 찾아서 사용한다. 포트를 직접 할당하려면 클라이언트 어플리케이션의 실행 스크립트에 다음을 추가한다.

```
-Djeus.tm.port=<port number>
```

12.3.4 Worker Thread Pool 설정

JEUS 트랜잭션 매니저에서는 다른 트랜잭션 매니저간의 내부 통신을 처리하거나 Global Transaction 의 commit 처리를 위해서 여러 개의 Worker Thread 를 사용한다. Worker Thread Pool 은 다음과 같은 프로퍼티로 설정된다. 다음 프로퍼티들은 JEUSMain.xml 의 <engine-container> 태그 내부의 <tm-config> <pooing> 태그에서 설정한다.

- **Min**: 풀에 생성되는 Worker Thread 의 수. Client Container 에서 이 값을 설정하려면, `-Djeus.tm.tmMin=<number of threads>` 를 클라이언트 실행 스크립트에 넣어준다. Web Container 나 EJB Container 에서 설정하려면 <pool>태그 안의 <min> 태그에서 설정한다.

- **Max**: 풀에서 생성되는 쓰레드의 최대 개수. Client Container 에서 설정하려면 `-Djeus.tm.tmMax=<number of threads>`를 클라이언트 실행 스크립트에 넣어준다. Web Container 나 EJB Container 에서 설정하려면 <pool>태그 안의 <max> 태그에서 설정한다.

- **Resizing Period**: Worker Thread 를 없애기 위해서 풀을 검사하는 시간 간격. Client Container 에서 설정하려면 `-Djeus.tm.resizing Period=<number of threads>`를 클라이언트 실행 스크립트에 넣어준다. Web Container 나 EJB Container 에서 설정하려면 <pool>태그 안의 <period> 태그에서 설정한다.

예:

<<*JEUSMain.xml*>>

```
<jeus-system>
    <node>
        ...
        <engine-container>
            ...
            <tm-config>
                <pooling>
                    <min>10</min>
                    <max>50</max>
                    <period>1800000</period>
                </pooling>
                ...
            </tm-config>
```

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

12.3.5 타임 아웃 설정

JEUS 트랜잭션 매니저는 예외 상황 처리를 위해서 다양한 타임 아웃 메커니즘을 사용한다. 타임 아웃 메커니즘의 값을 조정해서 어플리케이션 시스템에 가장 적합하도록 트랜잭션 매니저를 튜닝 할 수 있다.

- **Resolution**: 타임 아웃 메커니즘의 기본 시간 간격. 이 값을 작게 설정하면 타임 아웃이 세밀하게 작동한다. 값이 너무 크면 타임 아웃이 너무 느슨하게 작동한다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `-Djeus.tm.tmResolution=<time in milliseconds>`를 추가한다. Web Container 나 EJB Container 에 설정하려면 <resolution> 태그에 값을 설정한다.

- **Active Timeout**: Global Transaction 이 시작되면 이 시간 안에 commit 이 실행되어야 한다. 그렇지 않으면 트랜잭션 매니저가 rollback 시켜버린다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `-Djeus.tm.activeto=<time in milliseconds>`추가한다. Web Container 나 EJB Container 에 설정하려면 <active-timeout> 태그에 값을 설정한다.

- **Prepare Timeout**: Root Coordinator 는 이 시간 내에 Sub Coordinator 와 리소스 매니저로부터 'prepare' 신호를 받아야 한다. 만약 받지를 못하면 Root Coordinator 는 Global Transaction 을 rollback 시킨다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `-Djeus.tm.prepareto=<time in milliseconds>`를 추가한다. Web Container 나 EJB Container 에 설정하려면 <prepare-timeout> 태그에 값을 설정한다.

- **Prepared Timeout**: Sub Coordinator 는 자신의 Root Coordinator 로부터 이 시간 안에 commit 을 해야 할지, rollback 을 해야 할지를 나타내는 global decision 을 받아야 한다. 만약 이 시간 내에 받질 못하면, Root Coordinator 로 다시 'prepare'에 대한 응답 메시지를 보낸다. 그래도 여전히 시간 내에 global decision 이 오지 않는다면, <heuristic-rollback> 의 값이 true 일 때 Global Transaction 을 rollback 시켜버린다. <heuristi

c-rollback>이 false 이면, Root Coordinator 로 메시지를 보내고 global decision 을 기다리기를 계속 한다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `- Djeus.tm.preparedto=<time in milliseconds>`를 추가한다. Web Container 나 EJB Container 에 설정하려면 <prepared-timeout> 태그에 값을 설정한다.

- **Commit Timeout**: Root Coordinator 는 이 시간 내에 Sub Coordinator 와 리소스 매니저로부터 'commit'이나 'rollback' 에 대한 결과를 받아야 한다. 만약 결과가 오지 않으면, Root Coordinator 는 Global Transaction 을 'Uncompleted List'에 기록해서, 트랜잭션이 완전히 끝나지 않았음을 남겨둔다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `- Djeus.tm.committo =<time in milliseconds>`를 추가한다. Web Container 나 EJB Container 에 설정하려면 <commit-timeout> 태그에 값을 설정한다.

- **Recovery Timeout**: 이 값은 트랜잭션 복구 시에 사용된다. 트랜잭션 매니저는 트랜잭션 복구를 위해서 복구될 트랜잭션 정보를 가져오려고 한다. 만약 다른 트랜잭션 매니저에서 이 시간 내에 복구 정보를 알려주지 않으면, <heuristic-rollback>이 true 일 때, 해당 트랜잭션을 rollback 시킨다. 값이 false 이면 트랜잭션 복구를 시스템 관리자에게 남겨두고 더 이상 진행하지 않는다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `-Djeus.tm.recoveryto=<time in milliseconds>`를 추가한다. Web Container 나 EJB Container 에 설정하려면 <recovery-timeout> 태그에 값을 설정한다.

- **Uncompleted Timeout**: 트랜잭션 매니저는 전체 트랜잭션 처리를 완료하기 위해, 실패한 Global Transaction 의 목록을 보관한다. 완료되지 못한 Global Transaction 의 정보는 복구 처리시에 사용되므로, 이 타임 아웃 시간까지 보관된다. 그러므로 이 시간이 너무 짧으면 복구 정보가 빨리 지워지게 되고, 트랜잭션 매니저가 해당 Global Transaction 의 무결성을 복구할 수 없게 된다. 그 결과 Global Transaction 복구를 위해서, 시스템 관리자가 많은 작업을 직접 처리해야만 한다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `- Djeus.tm.uncompletedto=<time in milliseconds>`를 추가한다. Web Container 나 EJB Container 에 설정하려면 `<uncompleted-timeout>` 태그에 값을 설정한다.

아래는 타임 아웃 설정에 대한 예이다.

<<JEUSMain.xml>>

```
<jeus-system>
    <node>
        ...
        <engine-container>
            ...
            <tm-config>
                <pooling>
                    <min>10</min>
                    <max>50</max>
                    <period>1800000</period>
                </pooling>
                <active-timeout>300000</active-timeout>
                <prepare-timeout>300000</prepare-timeout>
                <prepared-timeout>300000</prepared-timeout>
                <commit-timeout>300000</commit-timeout>
                <recovery-timeout>100000</recovery-timeout>
                <uncompleted-timeout>
                    90000000
                </uncompleted-timeout>
                ...
                <resolution>120000</resolution>
            </tm-config>
            ...
        </engine-container>
        ...
    </node>
    ...
</jeus-system>
```

12.3.6 로깅 설정

JEUS 트랜잭션 매니저는 Global Transaction 관련 로그를 남긴다. 이 로그 정보는 Global Transaction 복구에 꼭 필요한 정보로, 없으면 복구 작업을 진행할 수 없다. 시스템 관리자가 직접 복구할 경우에는 로깅 할 필요가 없다. 아니면 성능 향상을 위해서 로깅을 막는 경우도 있다. Client Container 에서 이 기능을 설정하려면 다음을 클라이언트 어플리케이션 스크립트에 추가해준다.

-Djeus.tm.noLogging=true

Web Container 나 EJB Container 에서 설정하려면 JEUSMain.xml 의 <command-option> 태그에서 위와 동일하게 설정해준다.

12.3.7 트랜잭션 매니저 용량 설정

이 값을 사용해서 JEUS 트랜잭션 매니저는 내부 구조를 최적화시킨다. 트랜잭션 매니저가 동시에 처리하는 Global Transaction 의 개수를 고려해서 값을 정한다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션 스크립트에 다음을 추가한다.

-Djeus.tm.capacity=<number of concurrent transactions>

Web Container 나 EJB Container 에서 설정하려면 JEUSMain.xml 의 <tm-config><capacity> 태그에서 값을 설정한다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    <node>
        ...
        <engine-container>
            ...
            <tm-config>
                ...
                <capacity>20000</capacity>
                ...
            </tm-config>
            ...
        </engine-container>
        ...
    </node>
    ...
</jeus-system>
```

12.3.8 트랜잭션 리소스 관리 정책 설정

JEUS 트랜잭션 매니저는 Global Transaction 과 관련 있는 정보를 기록하거나 commit 하기 위해서 시스템 리소스를 사용한다. 이 때 사용한 시스템 리소스를 놓아주기 위해서 두 가지 메커니즘, 'eager'와 'lazy' 메커니즘을 JEUS 에서 제공한다. 'eager' 메커니즘은 C 의 메모리 관리 방법과 유사하다. Global Transaction 에서 사용된 시스템 리소스는 트랜잭션이 완료될 때 모두 놓아주는 방법이다. 'lazy' 메커니즘은 Java 의 메모리 관리 방법과 유사하다. Global

Transaction 과 관련된 시스템 리소스는 active timeout 후에 놓아진다. 이 방법은 트랜잭션 매니저의 성능에 지대한 영향을 미친다.

eager 메커니즘은 Global Transaction 과 관련된 모든 정보를 commit 전에 열심히 모아야 한다. 그러므로 Root Coordinator 와 Sub Coordinator 간에 추가적인 통신이 발생하게 되고, 성능의 저하를 가져온다.

lazy 메커니즘의 경우, active timeout 이 발생할 때까지 시스템 리소스의 해제가 미뤄진다. 이 메커니즘에서는 Sub Coordinator 가 Root Coordinator 에게 Global Transaction 관련 정보 제공을 줄일 수 있으므로, 상호 통신 발생 수를 줄이게 된다. 보고해야 할 특별한 정보가 없다면, Sub Coordinator 는 Root Coordinator 와 통신을 하지 않고, 사용 중인 시스템 리소스는 나중에 해제하게 된다. JVM 자체가 Garbage Collection 이라는 lazy 메모리 관리를 하고 있기 때문에, 전혀 문제가 되지 않는다. 그러나 시스템 리소스가 빨리 해제되지 않기 때문에, active timeout 이 너무 클 경우에는 메모리 문제를 일으킬 수 있다.

어플리케이션 시스템의 메모리가 작을 경우에만 eager 메커니즘을 추천한다.

Client Container 에서 lazy 메커니즘을 사용하려면 클라이언트 어플리케이션 스크립트에 다음을 추가한다.

-Djeus.tm.forcedReg=false

Web Container 나 EJB Container 에서 설정하려면 JEUSMain.xml 의 <command-option> 태그에서 위와 동일하게 설정해 준다.

12.3.9 어플리케이션 실행과 병렬로 처리되는 트랜잭션 매니저 연결 설정

Global Transaction 이 메소드 호출 시에 자신의 container 를 벗어나 다른 container 로 간다면 다른 container 에서 Sub Coordinator 가 생성되고, 이 Sub Coordinator 는 트랜잭션이 시작된 container 의 Root Coordinator 로 자신을 등록한다. 메소드 호출이 끝날 때, Sub Coordinator 등록이 완료되었는지 체크한다. 등록 작업이 진행 중이면, 끝날 때까지 기다리게 된다.

그러나 Global Transaction 이 끝나기 전에 Sub Coordinator 가 등록되기 때문에, 어떤 때는 엄격하게 체크할 필요가 없다. 예를 들면, 어플리케이션이 다른 JVM 의 메소드를 몇 번 호출한다면, 처음 메소드 호출부터 등록이 되었는지 엄격하게 검사할 필요가 없다. 다음 번에 호출되는 메소드가 충분히 시간이 걸린다면, 등록 체크를 병렬로 처리할 수 있다. 그렇지만 메소드가 실행되는 시간이, 병렬로 등록 체크를 하는 것보다 충분하게 시간이 걸리는지 확인해야 한다. 가장 확실한 방법은 병렬 처리를 하지 않고, 메소드 호출 시에 등

록을 하는 것이다. 이 옵션은 트랜잭션 전문가를 위한 옵션이므로 주의를 바란다.

Web Container 와 EJB Container 에서 병렬로 등록 처리를 하려면, JEUSMain.xml 의 <command-option> 태그에 다음을 추가한다.

-Djeus.tm.checkReg=false

Client Container 에서는 Sub Coordinator 가 생성되지 않으므로, 이 기능은 의미가 없다.

12.3.10 Heuristic Rollback 설정

가끔 JEUS 트랜잭션 매니저에서 Global Transaction 을 commit 할 것인지 아닌지 결정을 못할 경우가 있다. 이런 경우 트랜잭션 매니저는 Global Transaction 의 무결성을 최대화하려고 노력한다. 그래서 적절한 global decision 이 넘어올 때까지 기다릴 경우도 있다. 그러나 무작정 기다리는 것은 시스템 행(hang)을 초래할 수도 있다. 왜냐하면 Global Transaction 이 리소스 매니저에 락을 걸고 있고, 결국 이 때문에 다른 작업 요청이 락에 걸려 진행하지 못하게 된다.

이 프로퍼티는 두 가지의 경우에 의미를 가지고 있다. 하나는 Sub Coordinator 가 Root Coordinator 로부터 global decision 을 받지 못했을 경우이고, 다른 하나는 트랜잭션 매니저가 트랜잭션 복구를 하려는데 uncompleted Global Transaction 에서 정보를 얻어 올 수 없을 경우이다.

Global Transaction 을 기다릴 것인지 아니면 rollback 처리해서 다른 Global Transaction 을 처리하게 할 것인지는 시스템 매니저가 결정할 사항이다. rollback 을 하도록 설정하려면 클라이언트 어플리케이션의 스크립트에 다음을 추가한다.

```
-Djeus.tm.heuristicRollback=true
```

Web Container 나 EJB Container 에서 설정하려면 <heuristic-rollback> 태그의 값을 true 로 설정 한다.

<<JEUSMain.xml>>

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

```
            <tm-config>
                ...
                <heuristic-rollback>true</heuristic-rollback>
                ...
            </tm-config>
            ...
        </engine-container>
        ...
    </node>
    ...
</jeus-system>
```

12.3.11 결론

이 절에서는 JEUS 의 트랜잭션 매니저를 어떻게 설정하는지 다뤘다. 그리고 JEUSMain.xml 의 각 Engine Container 에서 서버 트랜잭션 매니저를 설정하는 것도 배웠다.

이 파일에서 가장 중요한 설정인 Worker Thread Pool, 타임아웃, 트랜잭션 매니저의 용량과 Heuristic Rollback 도 보았다. 또한 –D 옵션을 통해서 설정할 수 있는 다른 설정 사항들도 보았다.

다음 절은 클라이언트 트랜잭션 매니저를 설정하는 방법에 대해서 설명한다.

12.4 클라이언트 트랜잭션 매니저 설정

12.4.1 소개

클라이언트 트랜잭션 매니저는 Client Container 에서 클라이언트 어플리케이션이 Global Transaction 을 시작하고 종료하기 위해서 만들어졌다. 그러나 기능은 단지 서버 트랜잭션 매니저로 트랜잭션의 시작과 종료 신호를 보내는 것 뿐이다. 그 메시지를 제일 먼저 받은 서버 트랜잭션 매니저가 그 트랜잭션의 Root Coordinator 로 작동한다.

12.4.2 트랜잭션 매니저 타입 설정

클라이언트 트랜잭션 매니저는 Client Container 에서만 사용된다. 사용하려면 다음을 클라이언트 어플리케이션 스크립트에 추가한다.

```
-Djeus.tm.version=client
```

12.4.3 트랜잭션 매니저의 TCP 포트 설정

클라이언트 트랜잭션 매니저는 다른 트랜잭션 매니저와 통신하기 위해서 TCP/IP 포트를 사용한다. 기본적으로 클라이언트 트랜잭션 매니저는 적절한 포트를 스스로 찾아서 사용하나, 직접 설정하려면 다음을 클라이언트 어플리케이션 스크립트에 추가한다.

```
-Djeus.tm.port=<port number>
```

12.4.4 Worker Thread Pool 설정

JEUS 트랜잭션 매니저는 다른 트랜잭션 매니저나 Global Transaction 처리를 위해서 내부적으로 Worker Thread Pool 을 운영한다. 이 쓰레드 풀은 다음과 같은 프로퍼티를 사용해서 설정된다.

- **Min**: 풀 안에 기본적으로 만들어질 쓰레드 수. `-Djeus.tm.tmMin=<number of threads>` 를 클라이언트 어플리케이션 스크립트에 추가한다.
- **Max**: 풀에서 만들어질 수 있는 최대 쓰레드 수. `-Djeus.tm.tmMax=<number of threads>`를 클라이언트 어플리케이션 스크립트에 추가한다.
- **Resizing Period**: 풀에 idle 상태인 Work Thread 를 점검해서 없애는 시간 간격. `-Djeus.tm.resizingPeriod=<time in milliseconds>`를 클라이언트 어플리케이션 스크립트에 추가한다.

12.4.5 타임 아웃 설정

JEUS 트랜잭션 매니저는 예외 상황 처리를 위해서 다양한 타임 아웃 메커니즘을 사용한다. 타임 아웃 메커니즘의 값을 조정해서 어플리케이션 시스템에 가장 적합하도록 트랜잭션 매니저를 튜닝 할 수 있다.

- **Resolution**: 타임 아웃 메커니즘의 기본 시간 간격. 이 값을 작게 설정하면 타임 아웃이 세밀하게 작동한다. 값이 너무 크면 타임 아웃이 너무 느슨하게 작동한다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 `-Djeus.tm.tmResolution=<time in milliseconds>`를 추가한다.
- **Active Timeout**: Global Transaction 이 시작되면 이 시간 안에 commit 이 실행되어야 한다. 그렇지 않으면 트랜잭션 매니저가 rollback 시켜버린다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리

케이션의 스크립트에 -Djeus.tm.activeto=<time in milliseconds>추가한다.

- **Commit Timeout**: Root Coordinator 는 이 시간 내에 Sub Coordinator 와 리소스 매니저로부터 ‘commit’이나 ‘rollback’ 에 대한 결과를 받아야 한다. 만약 결과가 오지 않으면, Root Coordinator 는 Global Transaction 을 ‘Uncompleted List’에 기록해서, 트랜잭션이 완전히 끝나지 않았음을 남겨둔다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션의 스크립트에 - Djeus.tm.committo =<time in milliseconds>를 추가한다.

12.4.6 트랜잭션 매니저 용량 설정

이 값을 사용해서 JEUS 트랜잭션 매니저는 내부 구조를 최적화한다. 트랜잭션 매니저가 동시에 처리하는 Global Transaction 의 개수를 고려해서 값을 정한다. Client Container 에서 이 값을 설정하려면 클라이언트 어플리케이션 스크립트에 다음을 추가한다.

-Djeus.tm.capacity=<number of concurrent transactions>

12.4.7 결론

이번 절에서는 클라이언트 트랜잭션 매니저를 어떻게 세팅 하는지 설명했다.

다음 절에서는 트랜잭션 매니저를 사용하기 위한 어플리케이션 작성에 대해서 다룬다.

12.5 트랜잭션 어플리케이션 작성

12.5.1 소개

어플리케이션은 트랜잭션 작업을 하나의 트랜잭션으로 관리할 수 있다. 작업이 하나의 리소스 매니저를 통해서 처리된다면 LocalTransaction 을 형성한다. 그렇지 않다면 트랜잭션을 관리하기 위해서 Global Transaction 을 사용할 수밖에 없다.

트랜잭션 프로그래밍 패턴에는 4 가지가 있다.

12.5.2 LocalTransaction

LocalTransaction 은 하나의 리소스 매니저에 대한 트랜잭션 작업을 관리하는데 효과적인 방법이다. 가볍고 빠르기 때문에 가능하면 LocalTransaction 을

사용한다. LocalTransaction 은 JEUS 트랜잭션 매니저와는 아무 관계가 없으며, 모든 타입의 container 에서 사용할 수 있다.

다음은 LocalTransaction 을 사용하는 간단한 예제이다. 비록 Java 어플리케이션이지만 코드 일부분은 Servlet 이나 EJB 등과 같은 다른 J2EE 프로그래밍에서도 사용할 수 있다.

<<*Client.java*>>

```
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;
import javax.transaction.UserTransaction;
import java.sql.*;
import javax.sql.*;

public class Client {
    private static Connection con;

    private static Connection getConnection() throws
                              NamingException, SQLException {
              // get a JDBC connection
    }

    private static String insertCustomer(String id, String name,
             String phone) throws NamingException, SQLException {
        // insert a product entity given by the arguments to DB
    }

    private static void deleteCustomer(String id) throws
                              NamingException, SQLException {
        // delete a product entity given by the arguments from DB

    }

    public static void main(String[] args) {
        try {
            // get a JDBC connection
            con = getConnection();

            // set the autocommit attribute as false
```

```
            con.setAutoCommit(false);

            // insert customers
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting customer id="+i+"c
                                                ... from Client");
                customers[i] = insertCustomer(i+"c",
                          "Hong Kil Dong "+i, "000-123-1234-"+i);
            }
            System.out.println("completed inserting
customers!!");
            con.commit();

            // delete customers
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("deleting customer
                         id="+customers[i]+" ... from Client");
                deleteCustomer(customers[i]);
            }
            System.out.println("completed deleting customers!!");
            con.commit();


        } catch (NamingException ne) {
             System.out.println("Naming Exception caught : " +
                                ne.getMessage());
        } catch (SQLException sqle) {
             System.out.println("SQL Exception caught : " +
                                sqle.getMessage());
        } catch(Exception e) {
            try {
                con.rollback();
            } catch (Exception ee) {
                System.out.println("Transaction Rollback error :
                                    " + ee.getMessage());
            }
            System.out.println("Error caught : " +
                               e.getMessage());
            e.printStackTrace();
```

```
        } finally {
            try {
                if (con!=null) con.close();
            } catch (SQLException se) {}
        }
    }
}
```

12.5.3 Client Managed Transaction

Client Container 의 어플리케이션에서 Global Transaction 을 사용할 수 있다. 이 기능은 J2EE 스펙의 기능은 아니지만, 인트라넷 클라이언트가 여러 서버에 있는 서비스를 사용하는 트랜잭션 집약적인 작업을 할 때 유용하다.

다음은 간단한 예제이다.

클라이언트

Global Transaction 을 시작하기 전에, 'java:comp/UserTransaction'을 lookup 해서 javax.transaction.UserTransaction 의 인스턴스를 가져온다. 이 인스턴스의 begin()을 호출해서 Global Transaction 을 시작한 다음에, EJB 빈을 여러 번 호출한다. EJB 빈의 트랜잭션 작업을 commit 하려면 UserTransaction 인스턴스의 commit() 메소드를 실행해서 트랜잭션이 끝났음을 알린다.

정상적으로 commit 이 되었다면 메소드는 조용히 끝나게 된다. 장애가 발생해서 Global Transaction 이 commit 되지 못한다면 메소드에서 Exception 을 던지게 된다. javax.transaction.RollbackException 은 JEUS 트랜잭션 매니저가 Global Transaction 이 롤백 되었다는 것을 보장하는 Exception 이다. 이외에 다른 Exception 은 JEUS 트랜잭션 매니저가 예측할 수 없는 Exception 이 발생해서 Global Transaction 을 롤백 하려고 한다는 것을 나타낸다. 그러나 Global Transaction 에 포함된 모든 트랜잭션 작업들이 완전히 롤백 된다는 것은 아니다. 이럴 때는 catch 문 안에서 다시 한 번 직접 UserTransaction 의 rollback() 메소드를 호출해 Global Transaction 을 롤백 시켜준다. 그러지만 대부분 JEUS 트랜잭션 매니저가 롤백 처리 해버리기 때문에 별 효과가 없을 수 있다.

<<*Client.java*>>

```
package umt;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
```

```
import java.util.*;
import javax.transaction.UserTransaction;

public class Client {
    public static void main(String[] args) {
        UserTransaction tx = null;
        try {
            InitialContext initial = new InitialContext();
            Object objref = initial.lookup("productmanager");
            ProductManagerHome home = (ProductManagerHome)
                              PortableRemoteObject.narrow(objref,
                                       ProductManagerHome.class);
            ProductManager productManager = home.create();
            System.out.println("");
            System.out.println("<< Testing ProductManager EJBBean
                             Using User Managed Transaction >>");
            System.out.println("");
            int number = 10;
            String products[] = new String[number];
            tx = (UserTransaction)initial.lookup
                        ("java:comp/UserTransaction");
            tx.begin();
            // insert products
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting product id="+i+"b
                                                          ...");
                products[i] = productManager.insertProduct(i+"b",
                                              "ball pen", i*10);
            }
            for (int i=number/2 ; i<number ; i++) {
                System.out.println("inserting product id="+i+"f
                                                          ...");
                products[i] = productManager.insertProduct(i+"f",
                                "fountain pen", (i-number/3)*50);
            }
            System.out.println("completed inserting products!!");
            // delete products
            for (int i=0 ; i<number ; i++) {
                System.out.println("deleting product
```

```
                                    id="+products[i]+" ...");
                productManager.deleteProduct(products[i]);
            }
            System.out.println("completed deleting products!!");
            tx.commit();
            } catch (javax.transaction.SystemException se) {
                System.out.println("Transaction System Error
                                    caught : " + se.getMessage());
            } catch (javax.transaction.RollbackException re) {
                System.out.println("Transaction Rollback Error
                                    caught : " + re.getMessage());
            } catch(Exception e) {
                try {
                    tx.rollback();
                } catch (Exception ee) {
                    System.out.println("Transaction Rollback
                                    error : " + ee.getMessage());
                }
                System.out.println("Error caught : " +
                                     e.getMessage());
                e.printStackTrace();
            }
        }
    }
}
```

EJB

아래의 EJB 빈은 트랜잭션 작업을 하는 메소드를 가지고 있다. 클라이언트에서 Global Transaction 을 관리하기 위해서는 EJB 의 트랜잭션 타입이 container-managed 이고 트랜잭션 속성이 Required 나 Supports, Mandatory 셋 중 하나여야 한다. 트랜잭션 속성에 대한 자세한 내용은 EJB 스펙을 참조한다.

<<ProductManagerEJB.java>>

```
package umt;

import javax.ejb.*;

public class ProductManagerEJB implements SessionBean {
```

```
    public String insertProduct(String id, String name, double
                                price) throws EJBException {
        // insert a product entity given by the arguments to DB
    }

    public void deleteProduct(String id) throws EJBException {
        // delete a product entity indicated by the argument from
        // DB
    }

    // EJB callback methods
}
```

<<ejb-jar.xml>>

```
<ejb-jar>
    <enterprise-beans>
        <session>
             <ejb-name>productmanager</ejb-name>
             <home>umt.ProductManagerHome</home>
             <remote>umt.ProductManager</remote>
             <ejb-class>umt.ProductManagerEJB</ejb-class>
             <session-type>Stateless</session-type>
             <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>productmanager</ejb-name>
                <method-name>*</method-name>
                 <method-params/>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

12.5.4 Bean Managed Transaction

Web Container 와 EJB Container 에서 어플리케이션은 JTA 를 사용해서 Global Transaction 경계를 정할 수 있다. 어플리케이션이 Global Transaction 을 세밀하게 조절할 필요가 있을 때 유용한다. 실행 순서에 따라 클라이언트의 요청을 처리할 수 있기 때문에, 어플리케이션은 자신의 판단에 따라서 commit 과 rollback 을 할 수 있다.

다음 예제는 EJB Container 에서 어플리케이션이 Global Transaction 을 실행하는 것을 보여준다.

클라이언트

BMT(Bean Managed Transaction)에서는 EJB 빈이 Global Transaction 영역을 처리한다. 그러므로 모든 클라이언트는 단지 트랜잭션 작업을 하는 메소드를 호출하기만 하면 된다.

<<*Client.java*>>

```
package bmt;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class Client {
    public static void main(String[] args) {
        try {
            ProductManager productManager;
            // Getting a reference to an instance of
            // ProductManager EJB bean.

            productManager.transactionTest();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

EJB

EJB 빈은 Global Transaction 을 시작하기 위해서 javax.transaction.User Transaction 를 사용한다. javax.ejb.EJBContext(본 예제에서는 EJB 가 세션 빈이므로

javax.ejb.SessionContext)를 사용해서 UserTransaction 의 인스턴스를 얻어온다. Global Transaction 은 메소드의 실행으로 begin 되고 commit 된다. EJB 가 BMT 로 작동하려면 ejb-jar.xml 의 <transaction-type>태그에서 'Bean'이라고 입력한다.

<<ProductManagerEJB.java>>

```
package bmt;

import javax.ejb.*;
import javax.naming.*;
import java.sql.*;
import java.util.*;
import javax.transaction.UserTransaction;

public class ProductManagerEJB implements SessionBean {
    SessionContext ejbContext;

    public void setSessionContext(SessionContext ctx) {
        this.ejbContext = ctx;
    }

    public void transactionTest() throws EJBException {
        UserTransaction tx = null;
        try {
            int number = 20;
            String products[] = new String[number];
            tx = ejbContext.getUserTransaction();
            tx.begin();
            // insert products
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting product id="+i+"b
                                                        ...");
                products[i] = insertProduct(i+"b", "ball pen",
                                                        i*10);
            }
            for (int i=number/2 ; i<number ; i++) {
                System.out.println("inserting product id="+i+"f
                                                        ...");
                products[i] = insertProduct(i+"f", "fountain
                                        pen", (i-number/3)*50);
```

```
            }
            System.out.println("completed inserting products!!");
            // delete products
            for (int i=0 ; i<number ; i++) {
                System.out.println("deleting product
                                    id="+products[i]+" ...");
                deleteProduct(products[i]);
            }
            System.out.println("completed deleting products!!");
            tx.commit();
        } catch (javax.transaction.SystemException se) {
            throw new EJBException("Transaction System Error
                                    caught : " + se.getMessage());
        } catch (javax.transaction.RollbackException re) {
            throw new EJBException("Transaction Rollback Error
                                    caught : " + re.getMessage());
        } catch(Exception e) {
            try {
                tx.rollback();
            } catch (Exception ee) {
                throw new EJBException("Transaction Rollback
                                    error : " + ee.getMessage());
            }
            throw new EJBException("Error caught : " +
                                     e.getMessage());
        }
    }

    private String insertProduct(String id, String name, double
                    price) throws NamingException, SQLException {
        // insert a product entity given by the arguments to DB
    }

    private void deleteProduct(String id) throws NamingException,
                                                  SQLException {
        // delete a product entity indicated by the argument from
        // DB
    }
```

```
    // some EJB callback methods
}
```

<<ejb-jar.xml>>

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>productmanager</ejb-name>
            <home>bmt.ProductManagerHome</home>
            <remote>bmt.ProductManager</remote>
            <ejb-class>bmt.ProductManagerEJB</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Bean</transaction-type>
        </session>
    </enterprise-beans>
</ejb-jar>
```

12.5.5 Container Managed Transaction

J2EE 스펙에 의하면 어플리케이션은 Global Transaction 경계(demarcation) 지정을 container 에 위임할 수 있다. Web Container 나 EJB Container 는 메소드와 관련된 Global Transaction 을 관리한다. 어플리케이션은 설정 파일에 메소드 별로 트랜잭션 속성을 정할 수 있다. 이 방법이 Global Transaction 을 처리하는 가장 쉬운 방법이다.

다음은 EJB Container 가 관리하는 Global Transaction 의 예를 보여준다.

클라이언트

CMT(Container Managed Transaction)에서 서버 측 EJB Container 가 EJB 의 Global Transaction 작업을 관리하는 것을 보여준다.

<<Client.java>>

```
package bmt;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class Client {
```

```
    public static void main(String[] args) {
        try {
            ProductManager productManager;
            // getting a reference to an instance of
            // ProductManager EJB bean.
            productManager.transactionTest();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

EJB

CMT 의 장점은 EJB 빈 개발자가 비즈니스 로직 코드에서 더 이상 트랜잭션 관련 코드를 작성하지 않아도 되도록 한다는 것이다. EJB 빈의 메소드에 적절한 트랜잭션 속성만 지정하면 Global Transaction 관련 작업은 모두 EJB Container 에 위임된다. 아래 예제에서 transactionTest()는 트랜잭션 관련 코드가 아무 것도 없다. EJB 빈이 CMT 로 작동하도록 EJB Container 에 알려주려면, 해당 EJB 빈의 ejb-jar.xml 파일에 있는 <transaction-type>태그에서 'Container'라고 써준다.

<<ProductManagerEJB.java>>

```
package cmt;

import javax.ejb.*;
import javax.naming.*;
import java.sql.*;
import java.util.*;

public class ProductManagerEJB implements SessionBean {
    public void transactionTest() throws EJBException {
            try {
            int number = 20;
            String products[] = new String[number];
            // insert products
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting product id="+i+"b
                                                          ...");
                products[i] = insertProduct(i+"b", "ball pen",
```

```
                                                              i*10);
            }
            for (int i=number/2 ; i<number ; i++) {
                System.out.println("inserting product id="+i+"f
                                                              ...");
                products[i] = insertProduct(i+"f", "fountain
                                          pen", (i-number/3)*50);
            }
            System.out.println("completed inserting products!!");
            // delete products
            for (int i=0 ; i<number ; i++) {
                System.out.println("deleting product
                                      id="+products[i]+" ...");
                deleteProduct(products[i]);
            }
            System.out.println("completed deleting products!!");
        } catch(Exception e) {
            throw new EJBException("Error caught : " +
                                                  e.getMessage());
        }
    }

    private String insertProduct(String id, String name, double
                    price) throws NamingException, SQLException {
        // insert a product entity given by the arguments to DB
    }

    private void deleteProduct(String id) throws NamingException,
                                                   SQLException {
        // delete a product entity indicated by the argument from
        // DB
    }

    // some EJB callback methods
}
```

<<ejb-jar.xml>>

```
<ejb-jar>
    <enterprise-beans>
```

```
        <session>
            <ejb-name>productmanager</ejb-name>
            <home>cmt.ProductManagerHome</home>
            <remote>cmt.ProductManager</remote>
            <ejb-class>cmt.ProductManagerEJB</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>productmanager</ejb-name>
                    <method-name>*</method-name>
                    <method-params/>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

12.5.6 결론

지금까지 JEUS 에서 트랜잭션 프로그래밍 예제를 여러 가지 보았다. LocalTransaction 과 client managed transactions, bean managed transactions 그리고 container managed transactions 에 대해서 보았다.

12.6 트랜잭션 매니저의 트랜잭션 복구

12.6.1 소개

시스템 장애 시에 Global Transaction 의 무결성을 보장하는 것은 아주 중요한 문제다. 장애가 발생하는 위치로는 두 곳이 있다. 하나는 어플리케이션을 관리하는 container 이고, 또 하나는 DB 같은 리소스 매니저와 메시징 서버 또는 container 와 다양한 방법으로 연결된 EIS 등이 있다. JEUS 트랜잭션 매니저는 리소스 매니저와 관련된 로그와 Global Transaction 진행 상황을 계속 남기고 있다. 시스템 장애 발생 시에 이 자료를 바탕으로 복구를 진행하게 된다. 복구 절차는 container 에 의해서 자동 실행되거나 시스템 매니저가 tmadmin 콘솔 툴을 이용해서 조정할 수 있다.

12.6.2 복구 로그 파일

로그에는 두 가지가 있다. 하나는 Global Transaction 의 진행 상황을 기록한 것이고, 다른 것은 Global Transaction 과 관련된 리소스 매니저에 대한 정보를 기록한 것이다. Web Container 와 EJB Container 에서 log 파일을 JEUS_HOME/logs/TM/<ip port> 디렉터리에 만든다. Client Container 에서는 <user working directory>/logs/TM/<ip port>에서 만들어진다. 대량의 Global Transaction 이 처리된 후에 로그는, 정보를 패키징한 다음 다른 로그 파일로 기록되어서 재정비된다. 로그 파일의 형식은 JEUS 트랜잭션 매니저에서 읽히는 형태이므로 시스템 매니저는 읽을 수 없다. container 가 정상적으로 내려가면 로그 파일은 제거된다.

12.6.3 container 장애

container 에 장애가 발생하면, Uncompleted Global Transaction 은 container 가 리부팅될 때 자동 복구된다. 그래서 시스템 매니저가 할 일은 없다. JEUS 매니저가 container 를 자동으로 리부팅 시켜주므로, container 장애는 자동 복구되는 것을 보장한다.

12.6.4 리소스 매니저 장애

리소스 매니저에서 장애가 발생했다면, 시스템 매니저는 tmadmin 콘솔 툴을 사용해서 복구해야 한다. 왜냐하면 JEUS 트랜잭션 매니저는 리소스 매니저가 다시 살아났는지 어떤지 체크하기에 무리가 있기 때문이다. 리소스 매니저의 문제를 고친 후에 tmadmin 툴의 *resync* 명령어로 복구를 진행한다. tmadmin 의 자세한 사용법은 본 문서의 부록 D 를 참조한다.

12.6.5 결론

이번 절에서는 트랜잭션의 복구에 대해서 간략하게 보았다. 복구 로그 파일과 Engine Container 의 에러, 리소스 매니저의 에러에 대해서 이야기했다.

12.7 결론

JEUS 에서의 트랜잭션 매니저와 트랜잭션 프로그래밍에 대해서 배웠다.

서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저를 설정하는 것을 배웠으며, JEUS 에서 트랜잭션 어플리케이션을 개발하는 것도 배웠다. 마지막으로 JEUS 트랜잭션 매니저에서 어떻게 트랜잭션 복구를 하는지를 보았다.

커넥터와 JDBC 에 대해서 대한 자세한 내용을 각 장을 참고하기 바란다.

다음 장에서는 container 라고도 하는 JEUS Engine 에 대해서 다룬다.

13 Engine

13.1 소개

Engine 은 JEUS 서버의 핵심 구성 요소 중 하나이다. 각 Engine 은 Engine Container 의 내부에서 실행[그림 34]되고 EJB 나 Servlet 같은 비즈니스 어플리케이션을 실행할 책임을 가지고 있다.

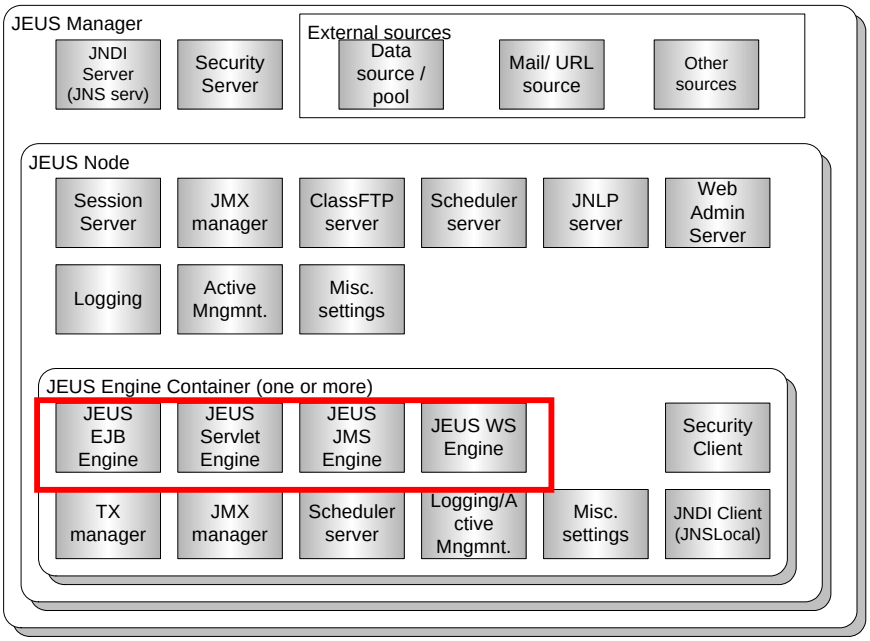

그림 34.Engine 이 표시된 JEUS 구조

13.2 JEUS Engine 개요

13.2.1 소개

이미 알고 있다시피 Engine 은 JEUS 핵심 구성 요소 중 하나이다. Engine 들은 Engine Container 내부에서 동작하며 container 에서 제공하는 트랜잭션, 네이밍, 보안 서비스 같은 것들을 이용한다.

Engine 들은 웹 어플리케이션을 구성하는 J2EE 비즈니스 컴포넌트를 실행할 책임을 가지고 있다.

참고: “Engine”과 “container”는 서로 교환 가능한 용어들이다. (J2EE 스펙에서는 후자를 더 일반적으로 사용한다. 반면 전자는 JEUS 만의 개념이다.).

13.2.2 4 가지 Engine 종류

JEUS 시스템에는 4 가지 종류의 Engine 이 있다.

1. EJB Engine.
2. Servlet (Web) Engine.
3. JMS Engine.
4. WS (Web Server) Engine.

다음 절들에서는 이 들 Engine 들에 대해서 간략히 알아볼 것이다.

13.2.3 EJB Engine

EJB Engine 은 Enterprise JavaBeans 를 실행할 책임을 가지고 있다. EJB 는 비즈니스 로직을 추상화하고 캡슐화하기 위한 표준화된 프레임워크이다. EJB 는 3 가지 중 하나가 될 수 있다.

1. **Entity beans** : 데이터 개체를 나타낸다.(예를 들어 데이터베이스의 row)
2. **Session beans**: 사용자의 작업이나 상호동작을 나타낸다.
3. **Message-driven beans**: EJB 2.0 스펙부터 소개된 특별한 EJB 이다. 이것은 JMS(Java Messsage Service) API 를 이용하며 EJB 와 다른 비즈니스 어플리케이션과 비동기, 메시지 기반의 통신을 가능하게 한다.

JEUS 서버의 EJB Engine 은 Sun Microsystems 의 EJB 2.1 스펙을 준수하는 이 3 가지 종류의 EJB 를 완전히 지원한다.

EJB 에 관한 보다 자세한 정보는 JEUS EJB 안내서를 참조하기 바란다.

13.2.4 Servlet Engine (Web Engine)

Servlet Engine(“Web Engine”혹은 “Web container”라고 불린다.)은 사용자 요청에 기반하여 동적인 웹 컨텐츠를 생성할 책임을 가지고 있다. 동적인 컨텐츠는 Java Servlet 이나 JSP 로 생성된다. Engine Container 가 지원하는 트랜잭

션이나 네이밍, 보안 서비스를 이용하여 이러한 Java 컴포넌트들을 실행하는 것이 바로 Servlet Engine 이다.

보통 HTTP 를 통해서 들어오는 클라이언트와 통신을 하기 위해서 웹 서버는 Servlet Engine 앞 단에 위치하여야 하며 Servlet Engine 에 연결되어야 한다.

JEUS 의 Servlet Engine 은 J2EE 1.4 과 Servlet 2.4 그리고 JSP 2.0 스펙을 완전히 지원한다.

좀 더 많은 정보를 위해서 JEUS Web Container 안내서를 보기 바란다.

13.2.5 JMS Engine

Java Message Service (JMS)는 중요한 비즈니스 데이터와 이벤트를 비동기적으로 주고받는 비즈니스 어플리케이션을 작성하기 쉽게 한다. Java Message Server 는 광범위한 엔터프라이즈 메시징 제품을 쉽고 효과적으로 지원하기 위한 일반적인 엔터프라이즈 메시징 API 를 정의하고 있다. 이것은 Message Queue 나 Publish/Subscribe 형태의 메시지 타입 모두를 지원한다.

JEUS 서버는 JMS Engine 포함하여 JMS1.1 을 완전하게 구현하여 제공하고 있다.

JMS 와 JEUS JMS Engine 에 관한 보다 자세한 내용은 JEUS JMS 안내서를 참조하기 바란다.

13.2.6 WS Engine (Web Server)

WS (Web Server) Engine 은 JEUS 의 앞단에 위치하며, HTTP 클라이언트들과 JEUS 시스템을 연동시키기 위해서 Servlet Engine 과 연결되어 있다.

WS Engine 을 설정할 때 사용되는 웹서버는 JEUS Web Server 이다. JEUS Web Server 는 WebtoB Web Server(standard version)의 기능인 클러스터링 같은 일부기능이 제외된 웹 서버이다.

간단히 말해 WS Engine 은 JEUS Web Server 를 시작 또는 정지 시키기 위해 사용되는 단순한 에이전트(혹은 proxy)이다.

참고: 하나의 노드에는 WS Engine 하나만 설정이 가능하다.

참고: JEUS Web Server 를 사용할 때 Servlet Engine 과 이것들을 연결하기 위해 Servlet Engine 보다 먼저 시작되어야 한다. 보다 자세한 것은 이 장의 13.3 절의 설정에 관한 절에서 설명한다.

JEUS Web Server 에 관한 보다 자세한 정보는 JEUS Web Server 안내서를 참고하기 바란다.

Servlet Engine 에 관한 보다 자세한 사항은 JEUS Web Container 안내서를 참조하기 바란다.

WS Engine 을 사용하기 위한 일부 기본적인 설정 정보는 이 장의 나중에 설명한다.

13.2.7 Engine 디렉토리 구조

우리는 이전에 JEUS Engine 에 포함된 기본적인 디렉토리 구조에 대해서 알아보았다. 그러나 명확하게 하기 위해 좀 더 자세한 예제를 보여준다[그림 35].

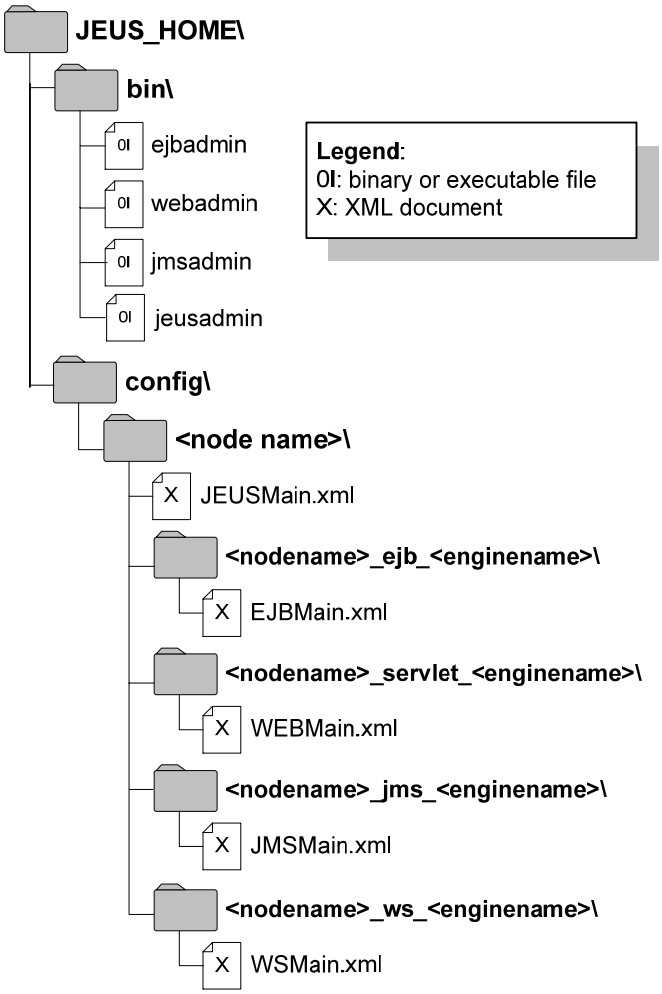

그림 35. JEUS Engine 들의 디렉토리 구조

위의 그림에서 볼 수 있듯이 각 Engine 은 Engine 홈 디렉토리 밑에 각각의 설정파일을 가지고 있다(config\<node name>\ 디렉토리 밑에). 또한 각 Engine 은 JEUSMain.xml 에 설정이 되어 있어야 한다.

각 Engine 은 또한 bin\ 디렉토리 밑에 각각의 관리 콘솔 툴을 가지고 있다.

WS Engine 의 설정은 다른 Engine 들처럼 해당 디렉토리에서 가져오는 것이 아니라 JEUS Web Server 가 설치된 디렉토리의 ws_engine.m 파일을 대신 사용한다는 것을 유념하기 바란다.

13.2.8 결론

JEUS 의 4 가지의 Engine(EJB Engine, Servlet Engine, JMS Engine 그리고 WS Engine) 종류에 대해 간략히 알아보았다. 이러한 Engine 들은 서로 다른 J2EE 비즈니스 컴포넌트를 실행할 책임을 가지고 있다.

이러한 Engine 들은 JEUS 노드의 Engine Container 밑에 설정되어야 한다는 것을 언급했다.

다음은 실제로 어떻게 이 4 종류의 Engine 을 설정하는지 보여준다.

13.3 Engine 설정

13.3.1 소개

JEUS 서버에서 4 가지 종류의 Engine 을 설정하는 것은 매우 쉽다. 단순히 몇 개의 XML 요소들을 JEUSMain.xml 에 추가 하면 된다.

그러나 Engine 이 동작하기 위해서는 각 Engine 에 필요한 설정파일을 생성해야만 한다(EJBMain.xml, WEBMain.xml, JMSMain.xml 그리고 ws_engine.m). 이런 파일들을 어떻게 설정하는지는 각각의 매뉴얼를 참고 하기 바란다.

JEUS 서버의 Engine Container 내에서의 설정(JEUSMain.xml)은 단순히 노드가 Engine 설정 파일 위치로 가서 해당 Engine 을 부트하도록 지시하는 것이다. 만약 해당 Engine 설정 파일을 찾지 못한다면 에러가 발생한다.

13.3.2 4 가지 Engine 종류의 설정

각 Engine 은 JEUSMain.xml 에서 `<engine-container>`요소 밑의 `<engine-command>`요소에 설정이 된다.

아래 설정은 이 요소 밑에 존재하는 것들이다.

- **Engine name** : 임의적으로 선택할 수 있다. 이 이름은 Engine 의 설정 파일을 찾을 수 있는 Engine 의 홈 디렉토리의 엔진 이름 부분과 일치해야 한다. 예를 들어서 만약 Engine 홈 디렉토리가 JEUS_HOME \config\johan\johan_servlet_Engine1\ 이라면 Servlet Engine 의 이름은 " Engine1" 이다.

- **Engine type**: 이전에 설명한 "ejb", "servlet", "jms", "ws" 중에 하나이다.

- **system-logging** : 이것은 Engine 의 logging 설정을 가지고 있다. 자세한 설정 방법은 14 장을 참조하기 바란다.

모든 Engine 종류가 포함된 XML 예제 일부가 아래에 있다. 적어도 하나의 <engine-command>요소가 각<engine-container>에 있어야 함을 유념하기 바란다.

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <node>
        ...
        <engine-container>
            ...
            <engine-command>
                <type>ejb</type>
                <name>Engine1</name>
            </engine-command>
            <engine-command>
                <type>servlet</type>
                <name>Engine1</name>
            </engine-command>
            <engine-command>
                <type>jms</type>
                <name>Engine1</name>
            </engine-command>
            <engine-command>
                <type>ws</type>
                <name>Engine1</name>
            </engine-command>
```

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

13.3.3 JEUS Web Server (WS Engine) 설정

WS Engine 은 ”JEUS Web Sever”라 불리는 내장 WebtoB Web Server 로서 Ser vletEngine 앞 단에서 사용될 수 있다.

WS Engine 이 JEUS Engine Container 에서 동작하기 위해 설정되었을 때 앞서 설명하였듯이 JEUS Web Server 설정 파일 또한 반드시 설정해야 한다. 설정 파일의 이름은 “ws_engine.m”이 되어야 하고 JEUS Web Server 의 홈 디렉토리“JEUS_HOME\webserver” (JEUS Web Server 안내서 참조)에 위치해야 한다.

<<*ws_engine.m*>>

```
*DOMAIN
jeuservice
*NODE
johan WEBTOBDIR="c:\jeus\webserver",
            SHMKEY = 54000,
            DOCROOT="c:\jeus\webserver\docs",
            PORT = "8080",
            LOGGING = "log1",
            ERRORLOG = "log2",
            JSVPORT = 9900,
            HTH = 1
*SVRGROUP
htmlg       NODENAME = johan, SvrType = HTML
cgig        NODENAME = johan, SVRTYPE = CGI
ssig        NODENAME = johan, SVRTYPE = SSI
jsvg        NODENAME = johan, SVRTYPE = JSV
*SERVER
html        SVGNAME = htmlg, MinProc = 1, MaxProc = 2
cgi         SVGNAME = cgig, MinProc = 1, MaxProc = 2
ssi         SVGNAME = ssig, MinProc = 1, MaxProc =2
MyGroup     SVGNAME = jsvg, MinProc = 25, MaxProc = 30
```

```
*URI
uri1        Uri = "/cgi-bin/", Svrtype = CGI
uri2        Uri = "/examples/",Svrtype = JSV, Svrname = MyGroup
*ALIAS
alias1      URI = "/cgi-bin/", RealPath = "c:\jeus\webserver\cgi-
bin\"
*LOGGING
log1        Format = "DEFAULT", FileName =
"c:\jeus\webserver\log\access.log", Option = "sync"
log2        Format = "ERROR", FileName =
"c:\jeus\webserver\log\error.log", Option = "sync"
*EXT
htm         MimeType = "text/html", SvrType = HTML
```

위의 예제에서 첫 번째 굵은 줄은 Web Engine 이 연결해야 하는 포트의 설정이다. 실제 연결을 얻기 위한 책임을 가진 Web Engine 구성요소를 *WebtoB listener* 라 부른다. 같은 포트 번호가 WEBMain.xml 에 설정되어야 한다.

두 번째 굵은 줄은 Servlet Server Group("jsvg")를 생성하여 JEUS Web Server 안에서 Servlet 모듈을 지원하도록 하는 것이다.

세 번째 굵은 줄은 새로운 HTH 프로세스("server")를 생성하고 등록 아이디로서 "MyGroup" 문자열을 사용하여 WebtoB listener 로부터 오는 요청을 처리하도록 설정하는 것이다. 이 HTH 는 "jsvg" 서버 그룹과 매핑된다.

마지막 굵은 줄은 URL 경로가 어떻게 "MyGroup"과 매핑되는지 보여준다. 예제에서 이것은 "/examples" 문자열을 포함하는 모든 HTTP 요청들은 "MyGroup" 컨텍스트 그룹을 포함하고 있는 Web Engine 으로 리다이렉트 할 것이다.

Web Engine 과 Web Server 연결에 관해 좀 더 많은 정보는 JEUS Web Container 안내서와 JEUS Web Server 안내서를 참조하기 바란다.

13.3.4 결론

지금까지 JEUSMain.xml 에서 기본적인 "Engine command"의 추가와 설정에 관해 알아보았다.

다음 절에서는 JEUS Engine 들을 제어하는 것에 대해서 알아본다.

13.4 Engine 제어

13.4.1 소개

jeusadmin 툴과 웹 관리자를 이용하여 각각 Engine 들을 제어하는 두 개의 명령들이 있다. 웹 관리자를 이용하는 방법은 JEUS 웹 관리자 안내서를 참조하고 여기서는 jeusadmin 툴만을 다룬다.

13.4.2 jeusadmin 콘솔 툴을 이용한 Engine 제어

jeusadmin 에서 “starteng”과 “downeng” 명령은 하나의 Engine 을 시작하거나 정지 시킬 때 사용한다. 좀 더 자세한 정보는 부록 B 를 참고 하기 바란다.

참고: Engine 들은 Engine Container 가 시작될 때 자동으로 시작된다. Engine Container 는 또한 JEUS 노드가 jeusadmin 에서 “boot”명령을 통해서 부트될 때 자동으로 시작된다. 또한 노드나 Engine Container 를 다운 시킬 때도 같이 적용이 된다.

참고: JEUS 4.x 까지는 EAR 을 deploy 하더라도 EAR 내의 EJB 모듈과 WEB 모듈의 deploy 가 별개로 이루어졌지만 JEUS 5 에서는 하나의 단위로 deploy 가 이루어진다. 따라서 EAR 이 deploy 되었다면 EJB 엔진만 down 시킬수 없다. 이럴 경우에는 engine container down 을 시도해야 한다.

13.5Engine 모니터링

13.5.1 jeusadmin 콘솔 툴을 이용한 Engine 모니터링

jeusadmin 콘솔 툴을 이용하여 Engine 을 모니터 하기 위해 “allenglist” 명령만을 사용할 수 있다. 이것은 현재 노드에서 활성화된 Engine 목록을 출력한다.

13.6결론

여기서는 JEUS Engine 들에 관한 결론을 내리고자 한다.

이미 보았듯이 4 개의 Engine 종류(Servlet, EJB, JMS and WS)가 JEUSMain.xml 의 Engine Container 레벨의 밑에서 설정이 된다. 그러나 각 Engine 은 Engine 만의 설정이 또한 필요하다. 설정 파일에 관한 좀 더 자세한 설명은 해당 메뉴얼을 참조하기 바란다.

다음 장에서는 JEUS 의 Logging 에 관해 다룬다.

14 JEUS Logging

14.1 소개

JEUS 는 JEUS 에서 발생하는 여러가지 상황들을 logging 을 통해 알려준다. JEUS logging 은 J2SE 1.4 이상의 버전에서 logging API 를 기반으로 동작한다. 따라서 설정하는 방식도 logging API 의 logger, handler, formatter 구조를 그대로 반영하고 있으며 개발자가 logging API 를 이용하여 JEUS 의 logger 를 원하는 대로 사용할 수 있다. 따라서 이번 장을 이해하기 위해서는 logging API 에 대한 이해가 필요하다. 이번 장은 JEUS logging 에 대한 개념과 설정방법, customization 방법 등을 설명한다.

14.2 JEUS logging 개요

JEUS 의 logger 는 "jeus" logger 를 기준으로 만들어져 있다. (여기서 "jeus"는 logger 의 이름을 뜻한다.) JEUS manager 와 engine container 들은 모두 "jeus" logger 를 기본적으로 생성한다. 하위에 생기는 여러가지 모듈들은 jeus.ejb 등과 같이 jeus 하위의 logger 를 생성해서 사용한다. 이런 여러가지 logger 중에서 JEUS 설정 파일에서 설정 가능한 logger 들은 다음과 같다.

- **jeus** : JEUS system 에서 사용하는 최상위의 logger 이다. JEUSMain.xml 의 <node> 항목의 <system-logging> 항목에서는 JEUS manager JVM 의 jeus logger 를 설정하고 <engine-container> 항목의 <system-logging>은 해당 engine-container JVM 의 jeus logger 를 설정한다.

- **jeus.systemuser :** engine container 의 user logger 에 해당한다. 따라서 JEUSMain.xml 의 <engine-container> 항목의 <user-logging>에 의해 설정된다.

- **jeus.ejb :** EJB 엔진에서 사용하는 logger 로 해당 EJB 엔진의 <engine-command>의 <system-logging>에 의해 설정된다.

위의 logger 중 jeus 를 제외한 다른 logger 들은 설정하지 않으면 상위 logger 의 handler 를 사용해서 log message 를 출력한다. 또한 jeus logger 는 설정되

지 않으면 console handler 를 사용한다. 따라서 JEUSMain.xml 에 아무런 logger 설정이 되어 있지 않다면 JEUS manager 와 모든 engine container 의 log message 가 JEUS manager 의 console 창으로 보이게 된다. jeus logger 는 기본적으로 상위 logger 의 handler 를 사용하지 않는다.

이 logger 들을 JEUSMain.xml 에 설정한 예는 다음과 같다.

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <node>
        ...
        <engine-container>
            ...
            <engine-command>
                <type>ejb</type>
                <name>Engine1</name>
                <system-logging>
                    <level>FINE</level>
                    <use-parent-handlers>
                        true
                    </use-parent-handlers>
                    <handler>
                        <file-handler>
                            <name>fileHandler</name>
                            <level>FINEST</level>
                            <valid-hour>1</valid-hour>
                        </file-handler>
                    </handler>
                </system-logging>
            </engine-command>
            <engine-command>
                <type>servlet</type>
                <name>Engine1</name>
            </engine-command>
            . . .
            <system-logging>
                <level>INFO</level>
                <handler>
                    <file-handler>
```

```
                    <name>fileHandler</name>
                    <level>FINEST</level>
                    <valid-hour>1</valid-hour>
                </file-handler>
            </handler>
            </system-logging>
    </engine-container>
    . . .
    <user-logging>
        <level>FINE</level>
        <use-parent-handlers>
              true
        </use-parent-handlers>
        <handler>
            <smtp-handler>
                <name>smtpHandler</name>
                <level>SEVERE</level>
                <smtp-host-address>
                    mail.com
                </smtp-host-address>
                <from-address>
                    jeus@mail.com
                </from-address>
                <to-address>
                    admin@mail.com
                </to-address>
                <send-for-all-messages>
                    false
                </send-for-all-messages>
            </smtp-handler>
        </handler>
    </user-logging>

    <system-logging>
        <level>FINE</level>
        <handler>
            <console-handler>
                <name>consoleHandler</name>
                <level>INFO</level>
```

```
                </console-handler>
                <file-handler>
                    <name>fileHandler</name>
                    <level>FINE</level>
                    <valid-hour>1</valid-hour>
                </file-handler>
            </handler>
        </system-logging>
    </node>
    . . .
</jeus-system>
```

14.3JEUS logging 사용

14.3.1 <system-logging> 설정

각 <system-logging> 항목의 하위 항목들은 다음과 같다.

- **<level>** : logger 의 level 을 설정한다. 이 level 이하의 message 만 logger 를 통해 출력될 수 있다. 이 level 의 값으로는 logging API 의 level 인 SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST 가 올수 있다. 기본 설정은 INFO 이다.

- **<use-parent-handlers> :** logger 가 자신의 handler 이외에 상위 logger 의 handler 를 사용할 것인지의 여부를 지정한다. 기본값은 false 이다.

- **<filter-class> :** logger 가 log message 를 handler 에게 보내기 전에 수행하는 filtering 에 사용할 class 를 지정한다. 여기에 지정된 class 는 lib/application 디렉토리의 jar 파일 내에 포함되어야 한다.

- **<handler> :** logger 가 사용할 handler 를 지정한다. 이 항목이 설정되어 있지 않다면 console handler 가 기본적으로 사용된다.

<handler> 에는 다음과 같은 하위 항목들이 있다.

- **<console-handler> :** 화면으로 log message 를 출력하는 handler 이다. 이 handler 는 다음과 같은 기본적인 handler 설정만 가지고 있다.

    - **name :** 이 handler 가 tool 에서 보여질 때 사용할 이름을 지정한다. 만약 지정되어 있지 않으면 class name 과 hash code 로 이름이 대체된다.

- **level :** 이 handler 가 출력할 message 의 level 을 지정한다. 즉, logger 를 통과한 log message 가 이 logger 가 사용하는 각각의 handler 에게 전달되는데 이 handler 의 level 에 부합하는 log message 만 이 handler 에 의해 출력된다. 기본 값은 FINEST 로 logger 를 통과하는 모든 log message 가 handler 에 의해 출력되도록 되어 있다.

- **encoding :** 이 handler 가 출력하는 문자열의 encoding 을 지정한다. 기본은 system encoding 으로 설정되어 있다.

- **filter-class :** 이 handler 가 log message 를 출력하기 전에 수행할 filtering 에 이용되는 class 이다. Logger 의 filter-class 와 마찬가지로 lib/application 에 이 클래스를 포함한 jar 파일이 존재해야 한다.

- **<file-handler> :** 파일로 log message 를 출력하는 handler 이다. 이 handler 는 <console-handler>의 설정 이외에 다음과 같은 설정을 가지고 있다.

  - **file-name :** 이 handler 가 출력할 file 의 이름을 지정한다. 절대경로로 되어 있다면 그 경로로 file 이 생기고 상대경로라면 각 logger 의 기본 경로를 기준으로 한 상대경로로 인식한다. 이 설정을 하지 않으면 각 logger 별로 지정된 path 로 file 을 생성해서 log message 를 출력한다.

  - **valid-day 와 valid-hour :** 이 handler 가 출력할 file 을 시간마다 따로 생성할 경우에 사용한다. 둘 중 하나만 사용할 수 있는데, valid-day 는 날짜별로, valid-hour 는 시간별로 file 을 변경한다. Valid-hour 는 24 의 약수이거나 (ex. 3, 6) 24 로 나눈 나머지가 약수(ex. 27, 30)의 이름을 지정한다. File 이름의 형식은 valid-day 의 경우 file 끝에 _YYYYMMDD 가 붙거나 valid-hour 의 경우 _YYYYMMDD_HH 가 붙는다. 이때 HH 는 file 로그의 시작 시간이다.

  - **buffer-size :** file 로 출력할 때 사용할 buffer 의 크기를 지정한다. buffer 가 클수록 logging 의 성능은 좋아지지만 예상치 못한 상황으로 JEUS 가 종료될 때에는 그 buffer 크기만큼 log 가 손실된다. 기본값은 20KB 이다.

  - **append :** file 로 출력할 때 이미 file 이 존재하면 덮어쓸지 file 끝에 추가할지를 결정한다. 기본값은 true 이다.

- **<smtp-handler> :** log message 를 email 로 전송하는 handler 이다. 하나의 log message 가 하나의 email 로 전송된다. 이 handler 는 <console-handler>의 설정 이외에 다음과 같은 설정을 가지고 있다.
    - **smtp-host-address :** email 을 보낼 host 의 주소를 지정한다.
    - **from-address :** email 을 보내는 사람의 주소를 지정한다.
    - **to-address :** email 을 받는 사람의 주소를 지정한다.
    - **cc-address :** email 을 참조하는 사람의 주소를 지정한다.
    - **bcc-address :** email 을 숨은 참조하는 사람의 주소를 지정한다.
    - **send-for-all-messages :** 모든 message 를 smtp-handler 로 보낼지를 결정한다. 만약 false 이면 JEUS system 에서 email 로 전송하기로 결정되어 있는 message 만 이 handler 를 사용해서 보내진다. 현재 이 설정은 jeus.systemuser logger 에만 유효하다.
- **<socket-handler> :** log message 를 socket 으로 전송하는 handler 이다. 이 handler 는 <console-handler>의 설정 이외에 다음과 같은 설정을 가지고 있다.
    - **address :** 이 handler 가 접속할 machine 의 IP 주소를 지정한다.
    - **port :** 이 handler 가 접속할 machine 의 port 를 지정한다.
- **<user-handler> :** user 가 만든 handler class 를 지정하는 항목이다. 이 handler 는 <console-handler>의 설정 이외에 다음과 같은 설정을 가지고 있다.
    - **handler-class :** user 가 만든 handler 의 class 를 지정한다. 이 class 는 lib/application 디렉토리의 jar 파일에 포함되어 있어야 한다. 또한 이 class 는 logging API 의 java.util.logging.Handler 를 상속받고 jeus.util.logging.JeusHandler 를 구현해야 한다.
    - **handler-property :** jeus.util.logging.JeusHandler 의 setProperty() 에 사용되는 Map 객체에 들어갈 property 를 지정한다.
    - **formatter-class :** 이 handler 가 사용할 formatter class 를 지정한다. 이 class 도 lib/application 디렉토리의 jar 파일에 포함되어 있어야 한다. 또한 jeus.util.logging.JeusFormatter interface 를 구

현해야 한다. 기본값은 JEUS 에서 사용하는 jeus.util.logging.SimpleFormatter 이다.

- **formatter-property :** jeus.util.logging.JeusFormatter 의 setProperty()에 사용되는 Map 객체에 들어갈 property 를 지정한다.

14.3.2 기본 Logger file 위치

File name 을 따로 지정하지 않고 File handler 를 사용할 경우 각 jeus logger 의 log message 는 정해진 위치에 file 을 생성한다. 다음은 각 logger 의 기본 위치 이다.

- **jeus** : JEUS manager 에서 남겨지는 log 는 logs/JeusSystem 디렉토리에 JeusServer.log 라는 파일로 남겨진다. 각 engine container 의 log 는 logs/JeusSystem/<nodename>_<containername> 디렉토리에<nodename>_<containername>.log 라는 file 로 남겨진다.

- **jeus.systemuser :** 각 engine container 의 user log 는 logs/JeusSystem/<nodename>_<containername> 디렉토리에 UserLog.log 라는 file 로 남겨진다.

- **jeus.ejb :** EJB 엔진의 logger 는 실행되는 engine container 이름의 logs/JeusSystem/<nodename>_<containername>/ejb 디렉토리 아래에 error.log 라는 file 로 남겨진다.

- **jeus.servlet :** Servlet 엔진의 logger 는 실행되는 engine container 이름의 logs/JeusSystem/<nodename>_<containername>/servlet 디렉토리 아래에 남겨진다.

- **jeus.jms :** Servlet 엔진의 logger 는 실행되는 engine container 이름의 logs/JeusSystem/<nodename>_<containername>/jms 디렉토리 아래에 남겨진다.

14.3.3 User logger

JEUS 의 각 engine container 마다 제공되는 user logger 는 개발자가 별도의 logger 를 사용할 필요없이 JEUS 에서 제공하는 logger 를 사용할 수 있도록 한다. 4.x 대의 jeus.util.UserLog API 를 사용할수도 있고 J2SE logging API 의 java.util.logging.Logger API 를 사용해서 user logger 를 사용할 수 있다.

14.4결론

지금까지 JEUS logging 시스템의 개념과 설정방법에 대해 알아 보았다. J2SE logging API 를 기반으로 구현되었기 때문에 개발자는 이 API 를 통해 JEUS logging 시스템을 원하는대로 customization 할수 있게 되었다.

15 JEUS Clustering

15.1 소개

잠재적인 시스템 과부하와 장애 대책을 위해서 보통 JEUS 여러 대를 클러스터링으로 묶어서 운영한다. 일반적으로 Node Clutering 이라고도 말한다.

이번 장은 JEUS 클러스터링에 관해 설명하고 몇 가지 예제도 제공한다.

15.2 JEUS 클러스터링 개요

15.2.1 소개

이번 절은 클러스터링 설정을 보다 쉽게 이해하기 위해서, 우선 JEUS 클러스터링의 아키텍처에 대해 알아본다.

JEUS 클러스터링의 주요 컴포넌트로는 노드, JEUSMain.xml 설정 파일 그리고 백업 노드가 있다:

15.2.2 기본 개념

JEUS 클러스터는 JEUSManager 또는 JEUS 노드의 Map 형태 구성이다. 즉, 각각의 node 는 다른 node 들과 connection 을 형성하고 있다.

여기서 JEUS Manager 와 JEUS 노드는 같은 것으로 가정한다. 최소한 클러스터링에 관한 내용일 때는 서로 통용해서 문제 될 것이 없다.

클러스터링 상의 각 노드는 다른 노드가 down 과 up 되는지를 감지해서 클러스터링을 다시 구성한다. 그래서 각 노드는 기본적으로 30 초마다 클러스터링 메시지를 확인한다. 이 값은 System property(부록 F)로 설정 가능하다. 자세한 내용은 하위 절의 15.2.7 에서 설명한다.

15.2.3 클러스터링에서 JEUSMain.xml 기능

클러스터링에 속하는 노드의 모든 정보는 각 노드의 JEUSMain.xml 파일에 담겨 있다. 그래서 각 노드에 있는 JEUSMain.xml 을 사용해서 클러스터링 안에 있는 모든 노드를 구별할 수 있다. [그림 36] 을 보기바란다:

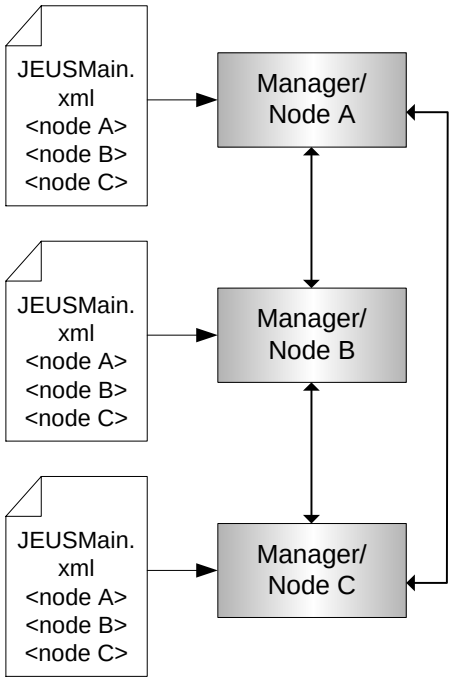

그림 36. 3 개의 노드와 설정파일 (JEUSMain.xml)을 포함한 JEUS 클러스터링.

그림에서처럼, 세 노드 A, B, C 의 JEUSMain.xml 파일 3 개는 거의 유사하다. 설정에서 <node> 태그가 100% 동일할 필요는 없으나, 동일하게 사용하는 것을 권장한다. 실제 중요한 것은 cluster 내의 노드가 모두 JEUSMain.xml 에 들어 있어야 한다는 것이다.

그리고, 타 노드의 정보는 노드의 이름 정도만 포함시켜도 무방하다. 예를 들면, A 노드의 JEUSMain.xml 에 B 노드의 모든 정보를 포함시킬 필요는 없고 단지 B 노드의 <node><name> 태그까지의 정보만 포함시키면 가능하다.

그러면, 각 노드는 자신이 사용할 노드가 셋 중 어떤 것인지 어떻게 알아내는가? 바로 hostname(표준 Java API 를 사용해서)을 얻고 JEUSMain.xml 에서 설정된 노드 명을 비교함으로써 결정된다. 노드명이 hostname 과 일치하는 노드의 설정을 사용하게 된다.

예로, 노드가 호스트 “A”에서 동작을 시작할 경우, 노드 “A” 의 설정을 사용한다. 다른 2 개 노드의 설정 정보는 클러스터링을 형성하는 다른 노드에서 사용한다.

15.2.4 JEUS 클러스터링을 위한 필수 설정 사항

JEUS 클러스터링은 서로 다른 노드 간의 클러스러링이므로 네트웍 관련 설정이 정확하게 이뤄져야 한다. 그리고, 하나의 노드는 다른 노드에 대해서 정확하게 알아야 하므로 이에 대한 설정이 필요하다.

그러므로 다음과 같은 3 가지 사항을 반드시 설정한다.

1. 각 머신의 hosts 파일에, 클러스터링되는 머신의 hostname 과 IP 주소를 정확히 매핑 정확해야 한다. Unix 의 경우 etc/hosts 파일에서 hostname 과 IP 주소를 매핑하며, Windows 의 경우는 %SystemRoot%\system32\drivers\etc\hosts 파일에서 매핑한다.

2. Virtual Node 를 사용한다면, 모든 머신이 Virtual Node 를 사용하도록 설정하며, 각 머신의 vhost.xml 에 모든 머신의 정보를 입력한다. 만약, Virtual Node 를 사용하지 않는다면, 모든 머신이 Virutal Node 사용하지 않도록 설정하며, 각 머신의 JEUS_BASEPORT 를 동일하게 설정한다.

3. JEUS 의 관리자 ID 와 Password 를 모든 노드에 동일하게 적용한다.

15.2.5 JEUS 클러스터링의 3 가지 규칙

클러스터링을 다룰 때, 클러스터링을 형성하는, 3 가지 기본 규칙을 반드시 알아야 한다.

1. 각 JEUS Manager 는 설정파일(JEUSMain.xml)상에서 자신의 노드 이외의 노드와 연결을 시도한다.

2. 만약 노드 하나가 “dead” 로 확인되면 (JEUS Manager 간의 연결이 되지 않았거나 끊어졌을 때) 강제로 클러스터링에서 제외된다. 그래서 “dead” 노드는 클러스터가 형성될 때 무시하고 넘어가게 된다. 자세한 내용은 다음절을 참고한다.

3. JEUS Manager 가 이전에 죽은 노드와 제외된 노드가 다시 살았는지를 확인하고, 살았다면 클러스터링에 포함한다.

15.2.6 클러스터링을 형성한 노드에 장애가 발생 할 경우

JEUS 클러스터링 상태에서 장애가 발생 경우를 생각해보자. 장애는 OS 레벨 에러나 네트워크 단절 등, 다양한 양상으로 발생될 수 있다.

[그림 37]에서 노드 B 는 에러로 장애가 발생한다:

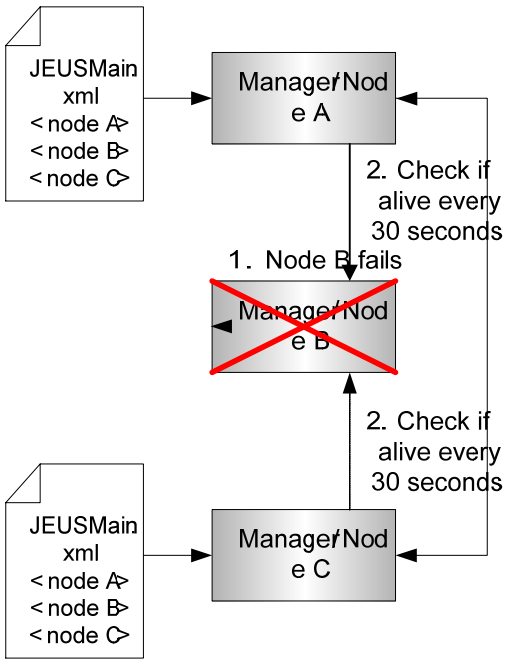

그림 37. 어떤 원인으로 노드/Manager B 장애. 대신에 노드 C 가 노드 A 와 연결.

[그림 37]에서 노드 B 가 장애가 발생했을 때, 노드 A 와 C 는 그것을 감지하고 노드 B 가 다시 살아나는지 여부를 계속 확인한다(이것을 alive-check 라고 한다). 이 alive-check 는 30 초마다 수행된다.

만약 노드 B 가 살아나면, 노드 A 와 C 는 alive-check 를 통하여 감지해 낸다. 그리고 클러스터링은 이전의 형태를 복구해서 [그림 36]로 되돌아간다.

이런 식으로 각 node 는 자신의 위치에서 다른 node 들의 이상 여부를 check 해서 clustering 정보를 유지한다. (이 정보는 jeusadmin 의 'nodelist' 명령에서 사용한다).

15.2.7 백업 노드의 사용

노드가 다운되었을 때, 클러스터링 내의 노드들은 이 상황을 감지한다. 만약 다운된 노드가 자신이 백업해야할 노드라면 다운 된 노드의 설정 데이터를 읽고 (클러스터링에서 JEUSMain.xml 은 다른 노드의 정보를 모두 가지고 있으므로) 장애가 발생한 Engine Container 의 백업 인스턴스를 부팅한다. 이것이 JEUS 에서 Fail-Over 처리를 하는 방법이다.

백업 노드도 자신의 Engine Container 설정을 가지고 있으므로, 백업을 위한 Engine Container 와 구별 할 방법이 있어야 한다. 이것은 'Engine Group'이라는 개념을 사용함으로써 해결된다. 백업 Engine Container 들은 백업 노드 내

에 자신만의 Namespace 에 놓이고, 원본 Engine Container 은 다른 Namespace 에 놓이므로 서로 혼동되지 않는다. 이 내용은 간단히 3 장에서 언급되었다.

참고: 원본 노드가 복구된 뒤에 백업노드를 다운하려면 jeusadmin 콘솔 툴의 "downenggr" 명령을 사용한다.

15.2.8 클러스터링에 새로운 노드의 동적 추가

가끔 JEUS 노드를 동적 추가해야 할 경우가 있다. 이것은 별개로 설정된 JEUS 가 동작중인 JEUS 클러스터링에 추가되는 것을 의미한다. 일반적인 방법은 앞서 설명했듯이, 각 노드에 클러스터링을 구성하는 것이다. 이것을 정적 클러스터링 구성이라고 한다.

동적인 노드 추가는 시스템의 확장을 위해서 필요하지만, 클러스터링을 완전히 리부팅시킬 수 없는 경우가 있다.

노드를 동적으로 추가하려면 단순히 보통 때처럼 구성하고 그 다음에 "`jeus -d`" 를 사용하여 부팅한다 (더 자세한 내용은 부록 A 참조). [그림 38]은 이러한 작업을 보여준다. 그림 (2 번째 지점) 에서 보듯, 새 노드에서 "`jeus -d`"를 실행하면, 클러스터링의 다른 노드에서 새 노드의 존재와 설정에 대해서 알지 못하더라도 "join" 시도가 진행된다.

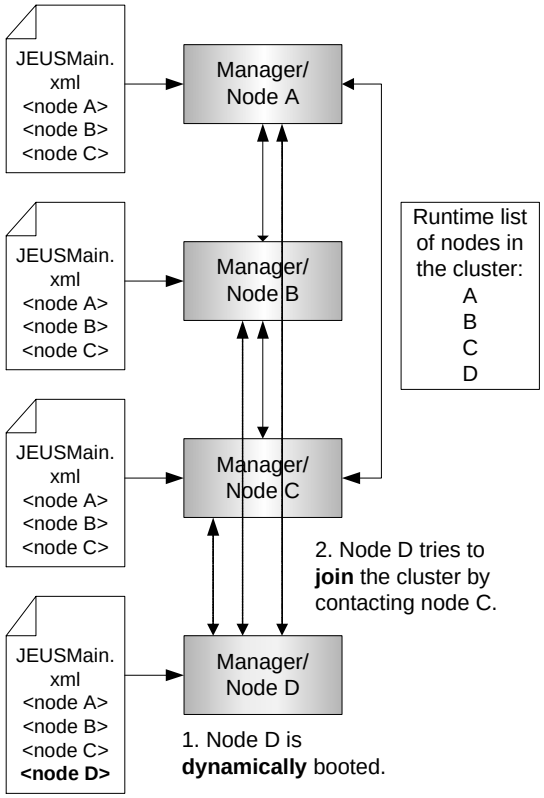

그림 38. 새로운 노드에 대해서 모르는 JEUS 클러스터에 새로운 JEUS 노드(D)의 동적 추가

그 결과 D 는 A, B, C 노드에게 접속을 시도해서 A, B, C 에게 자신의 존재를 알린다. 결국 A, B, C, D 는 모두 서로를 check 하면서 클러스터링을 이룬다.

15.2.9 클러스터링의 다른 타입

이번 장에서는 주로 JEUS Manager 의 클러스터링에 관해 다루고 있다. 그렇지만 이외에도 다른 종류의 클러스터 다수가 있다.

- **JNDI Naming Clustering**: JNS 서버로 형성된다. 자세한 내용은 5 장을 참조한다.
- **Security Server Clustering**: JSM(JEUS security Managers)으로 구성된다. 자세한 내용은 JEUS Security 안내서를 참조한다.
- **Web Server Clustering**: 웹 서버와 Servlet Engine 으로 구성된다. 시스템의 HTTP 처리의 성능을 증가하기 위해서 사용한다. 자세한 내용은 JEUS Web Server 안내서와 JEUS Web Container 안내서 를 참조한다.
- **Session Clustering**: Servlet 의 세션 데이터를 분산 처리하기 위해서 사용된다. Session Clustering 에는 중앙집중식 세션 서버와 멀티 세션 서

버, 분산 세션 서버의 방식이 있다. 자세한 내용은 본 매뉴얼 9 장 Session Server 와 10 장 분산 Session Server 를 참조한다.

- **EJB Clustering**: EJB Engine 과 빈의 백업 및 부하 분산을 위해서 사용한다. 자세한 내용은 JEUS EJB 안내서를 참조한다.

- **Destination Clustering**: JMS 의 분산된 시스템 채널을 가상적인 하나의 채널로 구성하기 위해서 여러 개의 Destination Clustering 을 지원한다. 자세한 내용은 JEUS JMS 안내서를 참조한다.

JNDI Naming Clustering 과 Security Clustering 은 JEUS Mangers 가 클러스터링을 형성할 때 자동으로 형성된다. 그러므로 별다른 설정은 필요없다. 하지만 Web Server 와 Session, EJB, Desination Clustering 은 특별한 설정을 해 주어야 한다.

15.2.10 결론

지금까지 JEUS 클러스터링의 아키텍처에 대해서 보았다. JEUS 클러스터링에서는 Managers, 노드, 백업 노드, alive-check 과 JEUSMain.xml 설정 파일로 구성된다.

클러스터링 형성을 정적으로 구성하는 것과 동적으로 구성하는 것도 보았다.

다음 절은 클러스터가 실제로 어떻게 JEUSMain.xml 파일에 설정되는지 알아 본다.

15.3JEUS 클러스터링의 설정

이번 절에서는 앞서 소개한 정적 클러스터링이 JEUSMain.xml 에서 어떻게 설정되는지 설명한다. 여기에는 노드 클러스터링에 관해서만 설명한다. 다른 형태의 클러스터링(예를 들면 Web 또는 EJB)에 관해서는 해당 매뉴얼을 참고하기 바란다.

앞 절에서 3 개의 JEUS 가 어떻게 Map 형태의 클러스터를 형성하는가에 대해서 알아보았다. 클러스터링을 만들기 위해서는 클러스터링에 참여하는 각 노드의 JEUSMain.xml 을 수정하여야 한다. 각 JEUSMain.xml 파일에는 반드시 클러스터링에 참여하는 노드들의 `<node>` 설정이 되어야 한다.

예)

<<*JEUSMain.xml*>>

```
<jeus-system>
    <node>
        <name>
            A
        </name>
        <backup-node>
            B
        </backup-node>
        ...   <!—다른 노드의 설정부분, 8장 참조 -->
    </node>
    <node>
        <name>
            B
        </name>
        <backup-node>
            C
        </backup-node>
        ...   <!—다른 노드의 설정부분, 8장 참조 -->

    </node>
    <node>
        <name>
            C
        </name>
        <backup-node>
            A
        </backup-node>
        ...   <!—다른 노드의 설정부분, 8장 참조 -->
    </node>
    ... <!—다른 Manager 설정부분, 4장 참조 -->
</jeus-system>
```

위의 예에서 3 개의 노드 A, B, C 는 클러스터링의 각 JEUSMain.xml 파일에서 설정되어 있다. 그리고 각각의 백업노드 설정도 포함시켰다. 이것은 하나의 노드가 다운되더라도 백업 노드가 응답할 수 있도록 하기 위한 설정이다.

위의 클러스터링 예에 동적으로 노드를 추가하는 작업은 위의 JEUSMain.xml 을 복사해서 네 번째 <node> 설정을 추가한다. 그리고, 새로

참여할 노드에서 jeusadmin 콘솔 툴을 이용해 boot –d 명령을 실행한다. 그러면 현재 동작중인 클러스터에 동적으로 새로운 노드가 추가된다.

15.4 클러스터링 컨트롤

클러스터를 시작하기 위해서는 먼저 'jeus' 명령으로 각 JEUS 들을 시작한다. 그 다음 jeusadmin 콘솔 툴 또는 웹 관리자를 열어서 각각의 JEUS Manager 에 boot 명령을 내린다. 부트 순서는 별 상관이 없다. 만일 동적으로 노드를 추가할 경우에는 boot –d 명령을 사용한다.

클러스터링을 종료하려면, 클러스터링 상의 각 노드에 down 명령을 실행시켜서 shutdown 시킨다. 완전히 클러스터링을 종료하려면 "jeusexit" 명령을 사용한다.

jeusadmin 콘솔 툴에 관한 자세한 설명은 부록 B 를, 웹 관리자에 관한 자세한 설명은 JEUS 웹 관리자 안내서를 참조 하도록 한다.

15.5 결론

이번 장에서는 JEUS 노드 클러스터링를 구성하기 위해서 반드시 필요한 내용에 관해서 살펴보았다.

비즈니스 어플리케이션이 시스템 장애 시에도 정상 작동하도록 보장하기 위해서, 백업 노드로 클러스터링이 사용되는 것을 보았다. 그러나 일반적으로 클러스터링은 시스템의 성능 향상을 위해서 사용된다.

JEUSMain.xml 에서의 클러스터링 설정은 이것으로 마친다.

16 JEUS 의 J2EE 어플리케이션

16.1 소개

J2EE 어플리케이션은 .ear(Enterprise Application aRchive) 형태의 파일로 배포된다. 이 파일에는 EJB 모듈(EJB .jar 파일), Web 모듈(.war 파일), Connector Resource Adapter(.rar 파일)과 기타 필요한 Java 클래스를 포함하고 있다. 또한 하나의 모듈 archive file 만을 가진 standalone application 도 J2EE application 의 한 종류이다. 이번 장에서는 JEUS 5 에서 이러한 J2EE 어플리케이션들을 만들어 deploy 하는 방법에 대해서 설명한다.

또한 이 장에서는 J2EE application 을 효율적으로 수행하기 위해 알아야 할 deploy 된 후 classloader 의 구조에 대해서도 설명한다.

EJB 모듈과 Web 모듈에 대한 내용은 JEUS EJB 안내서와 Web Container 안내서를 참고하길 바란다. Connector Resource Adapter 에 대해서는 JEUS Connector 안내서를 읽어보길 바란다.

16.2 J2EE 어플리케이션의 개관

16.2.1 소개

여기서는 .ear 파일의 일반적인 내용을 알아보고, JEUS 디렉토리의 어디에 놓아둬야 하는지 알아본다.

16.2.2 .ear 파일의 내용

.ear 파일은 일반적인 jar archive 형식의 파일로, 어플리케이션 DD 파일과 JEUS application DD 파일, 여러 개의 EJB 모듈, Web 모듈, Connector .rar 파일, Client application 으로 구성된다. [그림 39]은 간단한 예를 보여준다.

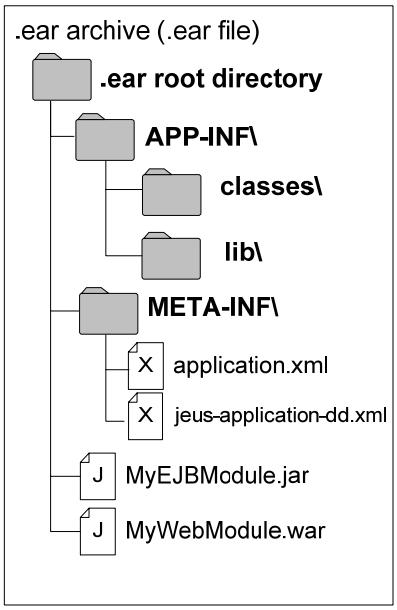

그림 39. Schematic of a sample .ear archive.

APP-INF 는 WEB 모듈의 WEB-INF 의 classes 와 lib 디렉토리와 같은 개념의 디렉토리를 가지고 있다. classes 의 class 들과 lib 의 jar 파일들은 이 application 에서 사용할 수 있는 class 들을 제공한다. 이때 classloading 설정이 SHARED 라고 되어 있다면 이 class 들은 다른 application 과 공유된다. Classloader 에 대한 설명은 아래를 참고하기 바란다.

META-INF\application.xml 파일은 J2EE 어플리케이션의 표준 DD 파일로, 어플리케이션 범위의 설정 사항을 포함하고 있다. 아래는 이 파일의 예제이다. 이 파일에 대해서는 J2EE 1.4 spec 을 참고하기 바란다.

<<application.xml>>

```
<application>
    <description>MyApp</description>
    <display-name>My application</display-name>
    <module>
        <web>
            <web-uri>MyWebModule.war</web-uri>
            <context-root>mycontext</context-root>
        </web>
     </module>
     <module>
         <ejb>MyEJBModule.jar</ejb>
     </module>
```

```
    <security-role>
        <role-name>Administrator</role-name>
    </security-role>
</application>
```

16.2.3 jeus-application-dd.xml

각 모듈마다 J2EE 표준 descriptor 이외에 JEUS 를 위한 descriptor 를 따로 가지고 있는것처럼 application descriptor 에 대해서도 이에 대응하는 jeus-application-dd.xml 이 존재한다. 이 파일은 EAR 의 META-INF 에 위치한다. 또한 standalone application 을 위해서도 존재할 수 있는데, WEB 모듈에 대해서는 WEB 모듈 archive 내의 WEB-INF 에 존재하고 다른 모듈에 대해서는 META-INF 내에 존재한다.

이 파일은 application 에 대한 정교한 설정을 하기 위해 사용한다. 만약 이 파일이 EAR 이나 standalone application 에 포함되어 있지 않으면 기본 설정으로 deploy 가 된다.

jeus-application-dd.xml 은 jeus-main.xsd 의 <application> 항목으로 정의되어 있다. jeus-application-dd.xml 의 예는 다음과 같다.

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

```
<application>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <path>myApplication.ear</path>
  <deployment-type>EAR</deployment-type>
  <ejb-component>
      <uri>ejb.jar</uri>
      <fast-deploy>true</fast-deploy>
  </ejb-component>
  <web-component>
      <uri>web.war</uri>
  </web-component>
</application>
```

<application> 항목 아래의 각 설정중 기본적인 설정을 나열하면 다음과 같다.

- **deployment-type** : 이 application 의 type 을 지정한다. 이 type 에 따라 EAR application 인지 standalone application 인지가 결정된다. 이 두가지 기준의 조합으로 다음의 2 가지 type 이 존재한다.

  - **EAR** : ear application 을 deploy 할 때 사용한다.
  - **COMPONENT** : standalone application 을 deploy 할 때 사용한다.

- **path** : application 의 path 를 설정한다. 이 값은 상대경로일 경우 deploy 가 될 engine container 의 webhome application directory 에서의 상대경로가 되고 절대경로일 경우 절대 경로 그대로 사용된다. Application 의 archive file path 가 오거나 그 archive 가 풀려진 directory path 가 올 수 있다.

<application> 하위에는 application 에 포함된 각 모듈들의 설정이 들어간다. 이는 <client-component>, <ejb-component>, <connector-component>, <web-component> 등으로 설정된다. EAR application 을 deploy 할 때 각 module 에 대한 설정을 할 필요가 없다면 이 element 들을 설정하지 않아도 된다. 하지만 standalone application 의 경우에는 그 application module 에 해당하는 element 가 항상 존재해야 한다. 이 항목의 공통 설정은 다음과 같다.

- **uri** : archive file 형태로 deploy 될 경우에는 해당 module 의 archive file 이름, directory 형태로 deploy 될 경우에는 해당 module 의 directory 이름이 지정된다. Standalone application 의 경우에는 uri 가 지정될 필요가 없다.

위에서 설명한 기본설정 이외에 보다 전문적인 설정은 다음과 같다.

- **auto-deploy** : 이 application 에 대해 archive file 이나 J2EE 표준 descriptor 가 변경되었을 경우 자동적으로 redeploy 를 실행한다. 여기에 대해서는 아래의 16.4.4 절에서 설명한다.

- **classloading** : application 을 deploy 할 때 사용할 classloading 방식을 설정한다. SHARED 인 경우에는 다른 application 의 EJB, Connector classloader 를 공유하고 ISOLATED 인 경우에는 공유하지 않는다. 기본설정은 -Djeus.classloading 으로 설정할수 있는데 이것도 설정되어 있지 않다면 SHARED 이다. Classloader 에 대해서는 아래에서 다시 설명한다.

- **class-ftp-unit** : application 에 포함된 EJB 모듈의 class 를 class ftp 를 통해서 client 에게 전달할 때의 단위로 CLASS 와 JAR 두가지 단위가 있다. 기본값은 JAR 이다.

- **security-domain :** application 에서 사용할 JEUS security domain 이다. 자세한 설명은 JEUS Security 안내서를 참고하기 바란다.

- **role-permission :** application scope 에서 사용할 principal-role mapping 을 지정한다. 자세한 설명은 JEUS Security 안내서를 참고하기 바란다.

- **java-security-permission :** J2SE security manager 를 사용할 경우 이 application 에 할당할 permission 들을 지정한다. 자세한 설명은 JEUS Security 안내서를 참고하기 바란다.

<application> 하위 태그의 설정에 대한 설명은 부록 J 을 참고하기 바란다.

16.2.4 J2EE 어플리케이션 작성과 Deploy

JEUS 에서 J2EE 어플리케이션을 작성하는 것에는 두 가지 방법이 있다.

1. J2EE DD 파일과 JEUS DD 파일, .jar, .war, .rar, .ear 파일을 'jar' 유틸리티를 사용해서 직접 작성하는 방법

2. 웹 관리자를 사용해서 작성하는 방법

이 두 가지 방법에 대해서는 마지막 절에서 설명한다.

JEUS 에 J2EE 어플리케이션을 deploy 하는방법에는 세 가지가 있다.

1. JEUSMain.xml 에 태그를 추가해서 JEUS 가 boot 할 때 자동으로 deploy 를 하는 방법(Boot deploy)

2. jeusadmin 에서 실행 중인 JEUS 로 deploy 하는 방법

3. 웹 관리자를 사용해서 실행 중인 JEUS 로 deploy 하는 방법

위 방법에 대해서는 마지막 절에서 설명한다.

16.2.5 EAR 파일 관련 디렉토리 구조

아래 [그림 40]은 ear 을 deploy 할 때 사용되는 디렉토리 구조를 보여준다.

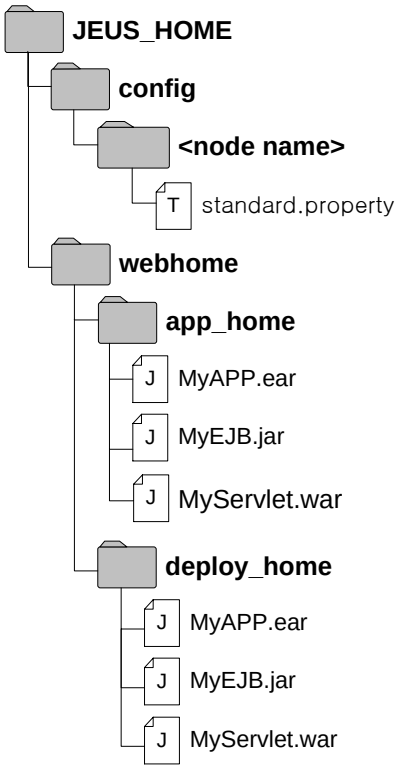

그림 40. JEUS EAR 디렉토리

- **JEUS_HOME**: JEUS 가 설치된 디렉토리(예, c:\jeus).
- **standard.property**: Deployment Descriptor 파일이 없는 J2EE 어플리케이션을 auto deploy 시킬 때 사용된다.
- **webhome**: 모든 JEUS 어플리케이션과 모듈을 위한 루트 디렉토리.
- **app_home**: application archive 나 directory 를 가지고 있는 directory 이다. 이 directory 는 JEUSMain.xml 의 설정으로 변경할 수 있다.
- **deploy_home:** 이 디렉토리로 J2EE 어플리케이션을 복사하면 자동으로 deploy 시켜준다. 이 디렉토리는 EAR_HOME 환경 변수에 저장되어서, 시스템 프로퍼티 –Djeus.deployhome 으로 설정된다.

16.2.6 결론

J2EE 어플리케이션과 ear 파일의 내용과 목적에 대해서 알아보았다. 그리고 ear 파일을 작성하고 deploy 하는 방법과 ear 디렉토리에 대해서도 간단히 알아보았다.

다음 절에서는 J2EE 어플리케이션을 deploy 하기 위한 JEUS 설정에 대해서 설명한다. 그 다음 절에서는 J2EE 어플리케이션을 어떻게 작성하는지 알아보며, 그 다음에는 JEUS 에서 실제 어떻게 deploy 하는지 알아본다.

16.3J2EE 어플리케이션 작성

16.3.1 소개

이번 절에서는 J2EE 어플리케이션 파일을 작성하는 방법을 배우게 된다. 다음 절에서는 만들어진 .ear 파일을 deploy 해 보도록 한다.

.ear 파일을 작성하기 전에, 우선 포함될 모듈을 작성해야 한다. .ear 파일에 들어가는 모듈은 EJB 모듈인 .jar 파일과 Web 모듈인 .war, Connector Resource Adapter 인 .rar 파일 등이 있다.

16.3.2 예제

이번 절에는 다음과 같은 상황을 가정해서 설명한다.

1. MyApp.ear 이라는 ear 파일을 작성한다.
2. 이 파일은 calc.jar 라는 EJB 모듈을 가지고 있다.
3. cookie.war 라는 Web 모듈도 가지고 있다.
4. MyApp.ear 파일은 “johan”이라는 노드에 deploy 되며, 엔진 컨테이너의 이름은 “johan_mycontainer”이다. 이 엔진 컨테이너는 하나의 EJB engine 과 Servlet engine 을 가지고 있다.

16.3.3 수작업으로 .ear 파일 작성하기

수작업으로 .ear 파일을 작성하려면 다음과 같이 한다.

1. .ear 파일에 포함될 .jar, .war, .rar 파일을 작성한다.
2. .jar, .war, .rar 파일과 같은 디렉토리에서 “META-INF’ 디렉토리를 만든다.
3. application.xml 파일을 생성해서 META-INF 디렉토리에 복사한다 (.ear 파일에 들어가 있는 모듈을 포함시킨다). 다음은 application.xml 의 예이다.

<<*application.xml*>>

```
<?xml version="1.0"?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
version="1.4" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/application_1_4.xsd">
<display-name>MyApp</display-name>
    <module>
        <ejb>calc.jar</ejb>
    </module>
    <module>
        <web>
            <web-uri>cookie.war</web-uri>
            <context-root>/cookie.war</context-root>
        </web>
    </module>
</application>
```

4. 다음과 같이 해서 .ear 파일을 생성한다.

```
jar cf MyApp.ear calc.jar cookie.war META-INF
```

위 명령을 실행하면 MyApp.ear 파일이 생성된다.

주의: META-INF 디렉토리가 대문자인 것에 주의하기 바란다. 만약 소문자로 작성하면 문제가 발생한다.

JEUS Builder GUI 툴을 사용해서 .ear 파일을 작성하는 방법은 JEUS Builder 안내서를 참조 하기 바란다.

16.3.4 결론

이상으로 J2EE 어플리케이션 작성이 완료되었다. J2EE 어플리케이션 패키징에 대한 더 자세한 정보는 J2EE 1.4 스펙이나 관련 서적을 참조하길 바란다.

다음 절에서는 작성한 .ear 파일을 JEUS 로 deploy 하는 방법에 대해서 알아본다.

16.4 J2EE 어플리케이션 Deploy

16.4.1 소개

JEUS 5 에 J2EE 어플리케이션을 deploy 하는 방법에는 4 가지가 있다.

1. JEUSMain.xml 에 .ear 을 등록해서 JEUS 부팅시에 deploy 되도록 한다.
2. jeusadmin 콘솔 툴에서 deploy 한다.
3. 웹 관리자에서 deploy 한다.
4. Auto Deploy 가 되는 DEPLOY_HOME 디렉토리에 복사한다.

웹 관리자를 이용하는 방법은 JEUS 웹 관리자 안내서를 참조하고, 나머지 방법에 대해서 하나씩 알아본다.

16.4.2 JEUSMain.xml 을 사용한 J2EE 어플리케이션 deploy

JEUSMain.xml 의 `<application>` 태그를 사용해서 부팅시에 deploy 될 .ear 을 설정한다.

중요: EJB 와 Web 모듈의 JEUS DD 파일은 deploy 되기 전에 각각 .jar 와 .war 파일에 포함되어야 한다. .ear 파일은 필요한 모든 JEUS DD 파일을 완전히 갖추고 있어야 한다.

`<application>` 항목은 jeus-application-dd.xml 의 <application> 항목과 동일하다. 따라서 EAR 에 포함된 jeus-application-dd.xml 의 <application> 항목을 그대로 사용할 수 있다. 다만 부팅시에 deploy 되는 과정에서 추가적으로 필요한 필수 정보가 있으므로 <application> 항목 아래에 다음과 같은 항목이 더 필요하다.

- **deployment-target** : 이 application 이 deploy 되는 target 을 설정한다. 하위 항목으로는 다음과 같이 있다. 이 항목들은 여러 번 나올 수 있으며 이들의 합집합이 application 이 deploy 될 target 이 된다.
  - **target :** 이 application 이 deploy 될 target 을 설정한다.
  - **all-targets :** 모든 engine-container 에 이 application 을 deploy 한다. 실제로 하나의 JEUSMain.xml 은 하나의 node 에 의해서만 사용되므로 node-name 으로 설정한 것과 같은 효과를 가진다.

<target>의 하위 element 로는 다음과 같은 element 가 있다.

- **node-name or engine-container-name :** 이 application 이 deploy 될 node name 이나 <node-name>_<container-name>을 설정한다. node name 을 사용할 경우 해당 node 의 모든 container 가 대상이 된다.

- **context-group :** 이 application 이 WEB module 을 가지고 있거나 standalone WEB module 인 경우에 deploy 될 context group 의 이름을 지정한다. 만약 container 별로 다른 context group 에 지정하고 싶다면 <target> 을 여러 번 사용해서 각각 지정해 줘야 한다.

만약 이 항목이 비어 있다면 target 이 설정되어 있지 않으므로 어떤 engine container 에도 deploy 되지 않는다. <deployment-target> 항목은 <application> 이외에 <application> 항목 아래의 각 component 의 항목인 <ejb-component> , <web-component>, <client-component> 등의 아래에도 나올 수 있다. 이 경우는 한 EAR 의 각 모듈들을 각각 다른 target 에 deploy 하기 위해 사용하는 것이다. 이 설정이 되어 있다면 이 설정을 포함하고 있는 모듈에 대해서는 <application> 아래의 target 설정이 무시된다. <connector-component> 는 application 의 다른 모듈에서 사용하는 resource 의 특징이 있으므로 이 application 이 deploy 되는 곳이라면 항상 deploy 되어야 하므로 별도의<deployment-target>을 가지고 있지 않다.

공통 설정을 가진 application 들은 반복해서 설정하는 대신에 <applications> 항목으로 묶어둘 수 있다. 이 경우 <application> 항목은 <applications> 항목의 하위에 있게 된다. <applications>의 설정은 다음과 같다.

- **auto-deploy** : <application>의 <auto-deploy> 하위 항목과 같은 설정으로 <applications> 내의 모든 application 에 대해 auto-deploy 를 설정한다.

- **deployment-target** : <application>의 <deployment-target> 하위 항목과 같은 설정으로 <applications> 내의 모든 application 의 deployment target 을 설정한다. 이 경우 각 application 의 deployment target 은 <applications>과 <application>의 target 설정의 합집합이다.

이 외에도 여러가지 설정이 있지만 이는 4.x 와의 호환성을 위해 존재한다. 자세한 설명은 부록 J 을 참고하기 바란다.

다음은 완전한 형태의 XML 예제이다.

<<JEUSMain.xml>>

```
<jeus-system>
```

```
    . . .
  <applications>
        <application>
            <name>myApp</name>
            <path>myApp.ear</path>
            <deployment-target>
                <target>
                 <node-name>johan</node-name>
                </target>
            </deployment-target>
            <deployment-type>EAR</deployment-type>
            <client-component>
                <uri>client.jar</uri>
            </client-component>
            <ejb-component>
                <uri>ejb.jar</uri>
            </ejb-component>
        </application>
    </applications>
</jeus-system>
```

위와 같이 JEUSMain.xml 에 정의하면, JEUS 가 부트될 때 .ear 파일이 자동으로 deploy 된다.

.ear 파일을 undeploy 하려면 JEUS 를 다운시키거나, jeusadmin 툴을 사용해서 undeploy 할 수 있다.

주의: 다시 한 번 강조하지만, EJB 와 Web 모듈을 위한 JEUS DD 파일은 .jar 파일과 .war 파일에 포함되어야 한다.

16.4.3 jeusadmin 툴을 이용한 J2EE 어플리케이션 Runtime Deploy

앞에서 만든 MyApp.ear 파일이 정상적으로 만들어졌다면, jeusadmin 콘솔 툴을 사용해서 JEUS 가 실행 중일 때 이 파일을 deploy 시킬 수 있다.

1. MyApp.ear 파일을 deploy 하고자하는 Engine Container 의 APP_HOME directory 에 복사한다. 기본 값은 JEUS_HOME\webhome\ app_home 이다.

2. JEUS 를 기동시킨다.

   ```
   >jeus
   ```

3. jeusadmin 이 실행되어 있지 않으면 실행시켜준다.

```
>jeusadmin johan
```

4. 정확한 username 과 password 를 입력한다.

5. JEUS 가 부트되어 있지 않다면, 'boot' 명령을 입력한다.

```
johan>boot
```

6. JEUS 가 부팅이 완료될 때까지 기다린다.

7. 프롬프트에서 다음과 같이 명령을 입력한다. 이 경우 MyJ2eeApp.ear 이 container 의 APP_HOME directory 에 존재해야 한다.

```
johan>deploy -con johan_mycontainer MyJ2eeApp
```

8. MyJ2eeApp 어플리케이션은 johan_mycontainer 에 deploy 된다.

9. 아래와 같이 deploy 여부를 확인한다.

```
johan>applist
name : MyJ2eeApp
        type : J2EEApplication
        EngineContainer : johan_mycontainer
        node : johan
```

10. 어플리케이션을 undeploy 하려면 다음과 같이 한다.

```
johan>undeploy MyJ2eeApp
```

jeusadmin 의 deploy 명령에 대한 자세한 내용은 부록 B 를 살펴보기를 바란다.

16.4.4 Auto Deploy 를 이용한 J2EE 어플리케이션 deploy

JEUS 5 에서는 J2EE application 을 deploy 할 때 별도의 deploy 명령없이 J2EE application archive file 을 지정된 directory 에 복사함으로써 deploy 할 수 있다. 또한 그 archive file 이 변경되면 자동으로 redeploy 가 수행되고 제거되면 undeploy 가 된다. 또한 다른 방식으로 deploy 했더라도 복사된 archive file 이 변경되거나 J2EE 표준 DD 파일의 파일 생성 시간이 변경되면 redeploy 가 되게 설정할 수 있다. 역시 이 경우에도 archive 가 제거되면 undeploy 가 된다.

이런 auto deploy 기능은 container 의 설정과 각 application 의 설정 두가지가 있다. 먼저 container 의 설정은 JEUSMain.xml 의 <engine-container> 안의 <auto-deploy> 태그로 설정한다. 만약 이 설정이 되어 있지 않다면 container 의 auto-deploy 는 실행되지 않는다.

중요: EJB 와 Web 모듈의 JEUS DD 파일은 deploy 되기 전에 각각 .jar 와 .war 파일에 포함되어야 한다. .ear 파일은 필요한 모든 JEUS DD 파일을 완전히 갖추고 있어야 한다.

<auto-deploy> 태그에는 다음과 같은 하위 태그를 설정한다.

- **auto-deploy-path**: Auto Deploy 가 동작할 디렉토리의 풀 패스를 지정한다. 지정하지 않으면 JEUS_HOME\webhome\deploy_home 을 사용한다. 예) c:\Jeus\webhome\deploy_home
- **auto-deploy-check-interval**: 파일의 변경 여부를 확인할 주기를 mi llisecond 단위로 지정한다. 예) 10000

지정된 directory 에는 오직 .ear 이나 .jar, .war, .rar 형태의 archive file 만 둘수 있다. archive 를 풀어둔 directory 형식은 적용되지 않는다.

Deploy 하고자 하는 .ear, .jar, .war 파일을 <auto-deploy> 태그에서 지정한 디렉토리로 복사하면 자동으로 Deploy 된다. Undeploy 하려면 간단히 파일을 삭제해 주는 것으로 가능한다.

다음은 이 기능을 설정한 JEUSMain.xml 의 예이다.

<<JEUSMain.xml>>

```
<jeus-system>
   . . .
   <engine-container>
      . . .
      <auto-deploy>
          <auto-deploy-path>
              c:\Jeus\webhome\deploy_home
          </auto-deploy-path>
          <auto-deploy-check-interval>
              2000
          </auto-deploy-check-interval>
       </auto-deploy>
</jeus-system>
```

application 별로 auto-deploy 를 설정하는 것은 이 application 이 deploy 된 후에 auto-deploy 를 적용하고 싶을 때 사용한다. 가령 auto-deploy 를 설정한 application 을 jeusadmin 을 통해 deploy 를 하고 난 다음에 application archive file 이 변경되거나 제거되는 경우 redeploy 나 undeploy 를 수행하고 싶을 경우 사용한다. 또한 이 경우에는 archive 를 풀어둔 directory 형태의 deployment type 에도 적용된다. 이 경우에는 해당 application 의 J2EE 표준 DD 파일이 변경되거나 제거된 경우 redeploy 나 undeploy 가 실행된다.

이 설정은 jeus-application-dd.xml 이나 JEUSMain.xml 의 <application>의 <auto-deploy> 항목으로 설정된다. 이 설정은 다음과 같다.

- **auto-deploy-check-interval**: 파일의 변경 여부를 확인할 주기를 millisecond 단위로 지정한다. 예) 10000

다음은 이 기능을 설정한 jeus-application-dd.xml 의 예이다.

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

```
<application>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <name>myApplication</name>
  <path>myApplication.ear</path>
  <deployment-type>EAR</deployment-type>
  <auto-deploy>
      <auto-deploy-check-interval>
            2000
      </auto-deploy-check-interval>
  </auto-deploy>

  <ejb-component>
      <uri>ejb.jar</uri>
      <fast-deploy>true</fast-deploy>
  </ejb-component>
  <web-component>
      <uri>web.war</uri>
  </web-component>
</application>
```

16.4.5 DD 파일이 없는 J2EE 어플리케이션의 Deploy

일반적으로 J2EE 어플리케이션을 deploy 하려면, jeus-ejb-dd.xml 이나 jeus-web-dd.xml 이 필요하다. 그러나 이런 JEUS DD 파일이 없을 때에도 deploy 가 가능하다.

우선 DD 파일이 없으므로 이를 위한 기본 설정이 필요한데, 이 설정을 위한 파일로 JEUS_HOME/config/<node name>/standard.property 가 있다. 이 파일의 속성들은 EJB 안내서의 DDInit 유틸리티 설명을 참고하기 바란다. DD 파일이 없는 .ear, .ejb, .war 파일은 이 파일의 설정을 바탕으로해서 deploy 되게 된다. 이 파일의 위치는 –Djeus.deploy.default 값으로 변경할 수 있다. 이 파라미터를 JEUSMain.xml 의 <command-option>에 넣어준다.

<<*JEUSMain.xml*>>

```
<jeus-system>
   . . .
   <engine-container>
       <name>mycontainer</name>
       <command-option>
         -Djeus.deploy.default=c:\Jeus\config
       </command-option>
     . . .
</jeus-system>
```

standard.property 의 파일은 jeus-ejb-dd.xml 과 jeus-web-dd.xml 파일의 항목을 프로퍼티 파일 형식으로 만든 것이다. 내용은 부록 L 를 참조 바란다. 다음은 standard.property 의 간단한 예제이다. 설정하지 않은 항목은 기본값이 적용된다.

<<*standard.property*>>

```
vendor=oracle
datasource-name=datasource1
creating-table=true
deleting-table=true
local-optimize=true
logging-level=debug
```

중요: JNDI export name 은 EJB Home 인터페이스의 이름으로 등록된다. 즉, sample.book.BookHome 이라는 Home 인터페이스를 사용하면 export name 도 sample.book.BookHome 이 된다.

주의: DD 파일 없이 standard.property 를 사용해서 deploy 하는 것은 간단한 EJB 를 기본 설정으로 deploy 하는 것이므로, 실제 작업에는 무리가 있다.

16.4.6 결론

지금까지 JEUS 에서 J2EE 어플리케이션을 deploy 시키는 방법을 알아보았다. JEUSMain.xml 에 .ear 파일을 설정해서 Boot deploy 를 할 수 있고, jeusadmin 이나 웹 관리자를 사용해서 실시간으로 Deploy 시킬 수 있다. 그리고, Auto Deploy 기능을 사용하면 파일을 복사하는 것으로 손쉽게 Deploy 할 수 있다.

16.5 결론

JEUS 5 에서 .ear 파일을 작성하고 Deploy 시키는 것에 대해서 알아보았다.

더 자세한 정보, 특히 application.xml 에 대한 정보를 원하면 J2EE 1.4 스펙을 보길 바란다.

17 마치며

지금까지 JEUS 의 가장 큰 기능들을 모두 보았다. 아래는 지금껏 보아온 것을 간단하게 정리한 것이다.

- JEUS Manager: 각 JEUS 노드에서 최상위의 관리자
- Naming Server: JEUS 가 구현한 JNDI Server
- Security Server: JEUS Engine 과 비즈니스 어플리케이션을 사용하기 위해서 시스템 차원에서 강제로 인증과 인가를 처리한다.
- Eternal Source: JDBC Data Source, URL Source, Mail Source, Tmax Connectivity, IBM MQ 와 SonicMQ Connectivity 그리고 J2EE Connector 를 제공한다.
- JEUS 노드
- Session Server: Servlet 과 EJB 의 세션 데이터를 저장하고 백업한다.
- JMX Manager.
- Engine Container: 별도의 JVM 에서 실행되며, JEUS Engine 을 탑재하고 있다.
- Transaction Manager.
- JEUS Engine: 4 가지 타입을 가지고 있다(Web, EJB, JMS, WS).
- JEUS 클러스터링: 고성능과 Fail-Over 기능을 제공하기 위해서 사용된다.
- J2EE 어플리케이션을 작성하고 Deploy 하는 기능을 제공한다.

본 매뉴얼의 처음부터 읽어서 여기까지 읽은 것을 축하드린다. 이것으로 JEUS 관리자나 Deployer 로서의 기술을 충분히 습득했을 것이다.

본 매뉴얼은 제한된 분량이지만, 많은 내용을 다루고 있다. 그러나 본문이 끝나면 보다 자세하고 정확한 정보를 지닌 부록이 있으므로, 이 부분도 완전히 습득하길 바란다.

그리고 여기에서 자세히 언급하지 못한 부분은 다른 매뉴얼을 보길 바란다. 참고할 매뉴얼로는 다음이 있다.

- JEUS Web Container 안내서
- JEUS EJB 안내서
- JEUS JMS 안내서
- JEUS Web Server 안내서 (WS Engine)
- JEUS Client Application 안내서
- JEUS 웹 관리자 안내서

A. jeus Script Reference

A.1 소개

본 부록에서는 JEUS 를 실행하는데 필요한 정보를 다룬다. 이 실행 명령어는 JEUS_HOME\bin\ 디렉토리에 있다.

A.2 목적

jeus 실행 명령은 JEUS Manager 를 기동 시키는데 사용된다. JEUS Manager 는 JEUS 에서 가장 중요한 컴포넌트이다. 따라서 JEUS 를 띄우기 전에 이 명령을 가장 먼저 실행해야 한다. JEUS Manager 는 실제로 JEUS 를 띄우고, 관리자의 명령을 서버로 전달하는 agent 역할을 한다.

“jeusadmin” 이나 웹 관리자 같은 관리툴을 사용하기 위해서는 JEUS Manager 를 기동시켜야 한다.

JEUS Manager 는 하나의 JEUS node 와 통신하고 하나의 JEUS Manager 는 각 host machine 에서 동작한다. JEUS Manager 는 클러스터링을 형성한 다른 JEUS Manager 와 연결된다.

JEUS Manager 는 JEUS_HOME\config\<nodename>\ 디렉토리 아래에 있는 JEUSMain.xml 파일을 읽어 들인다. 여기서 “nodename”은 JEUS Manager 가 관리하는 node 의 이름으로 머신의 이름이거나, vhost.xml 에서 정의한 가상 노드명이다.

A.3 사용법

JEUS Manager 를 실행하려면, Unix 나 Linux “jeus”를, Windows 에서는 “jeus.cmd”를 실행한다.

“jeus” 의 사용법은 다음과 같다.

```
jeus [–U<username> -P<password>] [-d]
```

간단하게는 명령 프롬프트에 “jeus” 라고 입력한다. “jeus” 스크립트를 수정해서 JVM 파라미터를 변경할 수 있다. 다음 절에서는 JEUS Manager 를 디버그 모드로 운영할 수 있게 “jeus” 스크립트를 설정하는 법을 다룬다.

“jeus” 스크립트의 옵션은 다음과 같다.

`-U<username>`

JEUS Manager 를 제어할 관리자의 이름.

`-P<password>`

관리자의 패스워드.

`-d`

클러스터링에 dynamic 하게 추가되는 Dynamic boot 를 하고자 할 때 사용한다.

위 옵션에서 `-U` 와 `-P` 는 항상 같이 사용되는 옵션으로, JEUS 를 One Step Boot 를 시킬 때 사용한다.

만약 jeus.properties 파일(Windows 에서는 jeus.properties.cmd)의 USERNAME 과 PASSWORD 환경 변수를 세팅하면, “jeus” 스크립트를 실행시키는 것만으로도 One Step Boot 가 진행된다.

A.4 디버그 모드에서 JEUS Manager 실행

가끔 디버그 모드로 JEUS Manager 를 실행시키는 경우가 있다. 디버그 모드로 실행할 경우, JPDA 지원 디버거를 사용하여 Servlet 과 EJB 어플리케이션을 디버그 할 수 있다.

JEUS 를 디버그 모드로 시작하기 위해서는 다음의 두 가지를 설정해야 한다.

1. Engine Container 의 이름은 “default”이어야 한다. 이는 JEUS Manager 와 같은 JVM 에서 동작하기 위해서다. Engine Container 에 대한 정보는 11 장을 참조한다.

2. “jeus” 스크립트 파일을 수정해야 하는데, 스크립트의 “java” 명령 바로 뒤에 다음 라인을 추가한다.

```
-classic -Xdebug -Xnoagent -Xrunjdwp:transport=dt_socket,adderss=
8888,suspend=n,server=y
```

위의 문자열(한 줄에 쓰여져야 한다.)은 “8888”의 디버그 포트(디버그 포트는 JPDA 디버거가 JVM 에 접근할 때 사용된다.) 를 사용하여 JEUS Manager 를 디버그 모드에서 동작되도록 한다.

Tip: “jeus” 스크립트 파일은 “jeus_debug”와 같이 복사본을 수정해서 사용하고 원본은 수정하지 않도록 한다.

수정된 스크립트로 JEUS Manager 를 기동시킨다. 그런 다음 JPDA 를 지원하는 IDE (Integrated Development Environment)를 사용하여 EJB 와 Servlet 어플리케이션을 디버그한다.

A.5 JEUS Manager 의 JVM Java 옵션

부록 F 에서는 JVM 에서 사용하는 “-D” 옵션에 대하여 기술한다. 이 옵션 중 중요한 옵션은 jeus.properties(또는 jeus.properties.cmd) 파일에서 환경 변수 세팅으로 사용할 수 있지만, JEUS 튜닝을 하거나 JEUSMain.xml 에는 없는 기능을 설정하기 위해 “jeus” 스크립트에 적용할 수 있다.

EJB 와 Servlet 에서 “-D” 옵션의 사용에 대해서는 각각 JEUS EJB 안내서와 JEUS Web Container 안내서 매뉴얼을 참고한다.

부록에서 주어진 Java 환경변수는 “-D”뒤에 추가하고 사용될 값은 “=” 문자 뒤에 붙여서 사용한다. 예:

```
-Djeus.baseport=9999
```

위의 예제는 ‘jeus’ 스크립트에서 “-classpath”절 보다는 뒤에 적고 클래스 이름 보다는 앞에 적는다. JEUS Manager 의 base port 는 9999 로 설정된다. 기본적으로 ‘jeus’ 스크립트에서 JVM 속성은 JEUS_HOME 이 “jeus.home”으로 할당되는 것처럼 시스템 환경 변수로 할당된다. 시스템 환경 변수에 대해서는 3 장을 참조한다.

참고: 환경 변수는 Engine Container 를 띄우는 개별 JVM 에 추가된다. 이러한 JVM 파라미터들은 JEUSMain.xml 파일의 Engine Container 환경에 설정되어야 한다. 자세한 내용은 부록 F 를 참고하기 바란다.

B. jeusadmin Console Tool Reference

B.1 소개

본 부록에서는 “jeusadmin” 콘솔 툴에 대한 모든 내용을 설명한다. 이 파일은 JEUS_HOME\bin 디렉토리에 있다.

B.2 목적

jeudadmin 은 JEUS 를 직접 관리하는데 사용하는 콘솔 툴이다. 이 툴을 사용하면 JEUS 의 start/stop, Engine 의 start/stop, Engine Container 의 start/stop 같은 기본적인 관리작업을 할 수 있다. 또한 Application 의 deploy/undeploy 와 logger level 변경 작업, JMX MBean 의 list 를 보는 등 여러가지 주요 작업을 jeusadmin 에서 할 수 있다.

각각의 Engine 에 대한 자세한 작업은 ejbadmin 이나 webadmin 같이 전용 툴을 사용한다. 이들 툴에 대한 내용은 해당 매뉴얼을 참조한다.

B.3 사용법

jeusadmin 의 사용법은 다음과 같다.

```
jeusadmin ([-verbose] [-jmx-connector connector_JNDI_name]  nodename [-U<username> -P<password>] [command])
          | -version |  -licensedue | -licenseinfo
```

파라미터:

`nodename`

nodename 을 가진 JEUS 로 jeusadmin 을 사용해서 접속한다. 그러면 사용자명과 패스워드를 입력하라는 프롬프트가 뜬다.

실행할 때 –verbose 를 사용했다면 콘솔창에 자세한 정보가 출력된다.

또한 –jmx-connector 옵션을 사용한다면 JEUS 의 기본적인 JMX Connector 대신에 옵션에서 주어진 JNDI name 으로 export 된 reference 를 사용하여 JMX Connector 를 만들어 쓴다.

`-U<username> -P<password>`

인증 프롬프트를 거치지 않고 바로 로그인할 때 사용한다. 아래의 “command” 파라미터를 사용할 때는 반드시 사용해야만 한다.

`command`

jeusadmin 의 명령어를 한 번 실행한다. 스크립트 같은 곳에서 사용할 때 유용하다. 기능은 대화형 command 에서 실행할때와 같은데 다만 jeusexit 는 down 이 되어 있지 않으면 down 명령을 시도하고 jeusexit 를 수행한다.

`-version`

JEUS 의 버전을 출력한다. 예) JEUS 5

`-licensedue`

라이센스의 남은 기간을 출력한다. 남은 기간이 0 이 되면 JEUS 는 더 이상 동작하지 않으므로, TmaxSoft 로부터 새로운 라이센스를 받아야 한다.

`-licenseinfo`

현재 인스톨되어진 라이센스에 대한 정보를 보여준다.

다음 절에서 jeusadmin 이 실행되었을 때 사용하는 명령어에 대해서 설명한다.

B.4 Manager 명령어

`Jeusexit [-all]`

현재 접속한 JEUS 의 JEUS Manager 를 종료한다. all 이 주어지면 clustering 상의 모든 JEUS Manager 를 종료시킨다.

B.5 Node & Group 명령어

`boot [-all]`

JEUSMain.xml 에서 설정 정보를 읽어와 JEUS 를 부팅한다. 노드를 부팅하면 모든 Engine Container 가 시작된다.

-all 옵션은 JEUS 클러스터내에 모든 JEUS 서버들을 부트하는 것을 의미한다.

`down [-e] [-i] [-all]`

JEUS 를 종료한다.

"-e" 옵션은 서버를 다운 시킨후 'jeusexit' 명령을 직접 실행 시키는 것을 의미한다.

"-i" 옵션은 JEUS 를 다운하기 전에 "yes/no" 확인을 무시하고 진행함을 의미한다.

"-all" 옵션은 JEUS 클러스터내에 있는 모든 서버들을 다운 시키는 것을 의미한다.

`nodelist [-a]`

JEUS 시스템에서 활성화 중인 노드의 목록을 출력한다.

"-a" 옵션은 jeusadmin 이 접속중인 node 의 nodelist 에 나온 목록에 대해 nodelist 명령을 반복적으로 실행해서 clustering 의 각 노드의 관점에서의 nodelist 를 출력한다.

`Pidlist [-a]`

현재 jeusadmin 이 접속된 node 의 Manager 프로세스와 각 Engine Container 의 프로세스 ID 목록을 출력한다.

"-a" 옵션은 jeusadmin 이 접속중인 node 의 nodelist 에 나온 목록에 대해 pidlist 명령을 반복적으로 실행해서 clustering 의 각 노드에 대해 pidlist 를 실행한다.

B.6 Engine Container 명령어

`conlist node_name`

입력한 그룹에서 활성화된 Engine Container 의 목록을 출력한다.

예제)

```
johan> conlist johan
```

`startcon [|container_name |all]`

입력한 Engine Container 을 실행시킨다. all 이라고 입력하면 jeusadmin 이 접속한 node 의 모든 engine container 를 실행시킨다.

예제)

```
johan> startcon johan_mycontainer
```

`downcon [|[-node node_name] |container_name |all]`

입력한 Engine Container 를 down 시킨다.

Container name 대신에 all 을 사용하면 jeusadmin 이 접속된 node 의 backup engine container 를 포함한 모든 engine container 들이 down 된다.

-node option 으로 container_name 대신 node_name 을 준 경우에는 해당 node 내의 모든 engine container 에 대해 down 을 시도한다. 이때 주의할 점은 여기서의 node 는 jeusadmin 이 접속한 node 에서 backup 하고 있는 node 를 의미한다는 것이다. 즉, downcon –node backup_node_name 을 실행하면 jeusadmin 이 접속한 node 의 해당 backup node 의 container 들을 down 시킨다.

예제)

```
johan> downcon johan_mycontainer
```

`englist container_name`

입력한 Engine Container 에서 활성화된 모든 Engine 목록을 출력한다.

예제)

```
johan> englist johan_mycontainer
```

`downbackup`

jeusadmin 이 접속해 있는 node 에서 backup 되어있는 node 들을 down 시킨다. 더불어 backup 시에 실행된 service 들도 모두 down 된다.

B.7 Engine 명령어

`allenglist`

현재 group(노드)의 활성화된 모든 Engine 목록을 출력한다.

`starteng Engine_name`

입력한 Engine 을 시작한다.

예제)

```
johan> starteng johan_servlet_Engine1
```

`downeng Engine_name`

입력한 Engine 을 down 시킨다.

예제)

```
johan> downeng johan_servlet_Engine1
```

B.8 애플리케이션 Deploy 명령

```
deploy [-con container_name] [-node node_name] [-f] [-per] [-ex]
            [-isolated] [-absolute-path archive_path] [-auto]
 [-localAppDD] [-writeAppDD]
 [-class-ftp-unit [|JAR |CLASS]] modulename
```

지정된 container 나 node 내의 모든 container 에 modulename 의 J2EE application 을 deploy 한다. 이 명령이 실행되려면 그 모듈의 archive file 이나 directory 가 해당 container 의 APP_HOME directory 에 존재해야 한다. (-absolute-path 옵션을 사용할 경우는 제외) node 의 모든 container 에 deploy 하기 위해서는 각각의 container 의 APP_HOME directory 에 modulename 에 해당하는 archive 나 directory 가 존재해야 한다. mo

dulename 이 그 archive file 의 확장자를 포함한 file name 이거나 direct ory 이름이 된다. modulename 이 file name 인 경우에 APP_HOME 에서 file 을 찾지 못하면 확장자를 땐 file name 과 같은 derectory 를 찾아서 deploy 한다. 또한 modulename 이 directiry 인 경우에 APP_HOME 에서 directory 를 찾지 못하면 그 module name 에 .ear, .jar, .war, .rar 과 같은 확장자가 있는 file 순서로 찾아서 deploy 한다.

-con 옵션은 deploy 할 container 를 지정한다. 이 옵션이 사용되지 않으면 node 의 모든 container 에 대해 deploy 를 수행한다.

-node 옵션은 jeusadmin 이 접속되어 있는 node 이외에 다른 node 에 deploy 할 때 사용한다.

-f 옵션은 J2EE application 내의 EJB 모듈에 대해 “fast deploy”를 하도록 명령한다. 이 때는 home 과 remote stub 과 skeleton 클래스들이 재생성되지 않는다. 처음 Deploy 를 하거나 모듈의 클래스 파일들이 변경되었을 때에는 이 옵션을 사용해서는 안 된다. 그러나, appcompiler 툴을 사용하여 필요한 클래스들을 생성했을 때에는 사용할 수 있다

-per 옵션은 모듈이 JEUSMain.xml 의 <application> 태그로 추가되서, 엔진이 부팅할 때 그 모듈이 로딩되도록 한다.

-ex 옵션은 application 이 directory 형태로 존재할 때에 사용한다.

-isolated 옵션은 deploy 하는 J2EE application 의 classloader 를 다른 application 들과 분리시킬 때 사용한다.

-absolute-path 옵션은 deploy 를 할 archive file 이나 directory 가 존재하는 절대경로를 설정한다. 따라서 container 의 application directory 에 application 이 존재하지 않아도 이 옵션을 사용해서 deploy 할 수 있다. 주의할 것은 여기의 path 는 jeusadmin 이 실행되는 node 의 path 가 아니라 deploy 를 수행하는 node 에서의 path 라는 것이다.

-auto 옵션은 deploy 하는 module 이 auto deploy 옵션을 사용하도록 한다. auto deploy 에 대한 자세한 설명은 16.4.4 를 참고하기 바란다.

-localAppDD 옵션은 deploy 를 할 때 application archive 에 존재하는 jeus-application-dd.xml 을 사용하게 하는 옵션이다. jeus-application-dd.xml 에 대한 자세한 설명은 16.2.3 를 참고하기 바란다.

-writeAppDD 옵션은 deploy 할 때 사용하는 jeus-application-dd.xml 를 container 의 해당 application archive 내에 복사해 두도록 하는 옵션이다. 이 옵션을 사용하고 나면 deploy 할 때 사용했던 –absolute-path 나 –auto 옵션등을 다음 deploy 할때에 –localAppDD 를 통해서 그대로 사용할 수 있게 된다.

-class-ftp-unit 은 deploy 된 application 내의 EJB 모듈의 stub class 를 client 에서 class-ftp 를 통해 가져올 때 JAR file 단위로 보낼지 class file 단위로 보낼지를 결정하는 옵션이다. 기본값은 JAR file 단위이다.

예)

```
johan> deploy –f mymodule
```

위의 명령은 “mymodule”이라는모듈을 johan node 의 모든 container 에 deploy 하면서 “fast deploy”를 하도록 설정하고 있다.

`reload modulename`

지정한 모듈을 다시 로딩한다. 이 옵션은 “deploy” 명령과 같은 동작을 하지만, 현재 동작중인 application 을 Undeploy 하고 Deploy 하도록 한다. 다양한 option 을 사용하고 싶다면 undeploy 명령과 deploy 명령을 이용하도록 한다.

`stop [-con containername] [-a] [-mod moduletype] modulename`

지정한 모듈을 멈추고, 다른 컴포넌트들은 이 application 에 접근할 수 없게 된다.

-con 옵션은 stop 시킬 모듈이 deploy 되어 있는 container 를 가리킨다. 이 옵션이 사용되지 않으면 node 내의 모든 container 에 적용된다.

-a 옵션은 modulename 이 사용되지 않을 경우에 쓰이는 옵션으로 모든 application 을 대상으로 stop 을 시킬 때 사용한다.

-mod 옵션은 stop 시킬 모듈의 type 을 지정할 때 사용한다. J2EEApplication, EJBModule, WEBModule, ResourceAdaptorModule, AppClientModule 등의 J2EE management spec 에 지정되어 있는 J2EEDeployedObject 의 하위 type 이 올 수 있다.

```
start [-con containername] [-a] [-mod moduletype] modulename
```

지정한 모듈을 다시 활성화하고, application 이 다시 서비스 가능한 상태로 되돌린다. 옵션은 stop 의 것과 같다.

```
undeploy [-con containername] [-a] [-mod moduletype] [-forced]
    [-per] modulename
```

지정한 모듈을 EJB 엔진에서 Undeploy 한다. 이 명령어가 수행될 때 어떤 파일들도 제거되지않는다.

-per 옵션이 사용되면 JEUSMain.xml 의 해당하는 <application> 태그가 제거된다. 나머지 옵션은 stop 의 것과 같다.

-forced 는 system application 이라도 undeploy 하고 싶을 때 사용한다. System application 은 JEUS 의 운영에 필요한 application 이므로 대부분의 경우 절대로 undeploy 되면 안되므로 주의해서 사용해야 한다.

```
applist [-con containername] [-a] [-mod moduletype] modulename
```

해당 container 나 node 내의 모든 container 에 대해 deploy 되어 있는 application 의 list 를 얻어온다. 옵션은 stop 의 것과 같다.

B.9 Logging 명령

```
loglevel [-con container_name] [-node node_name] [-f]
    [-set-handler] loggername [handlername] [loglevel]
```

지정된 container 나 node 의 지정된 logger 에 대한 level 을 보여주거나 새로운 level 로 지정한다.

-con 옵션은 지정된 container 의 logger 에 대해 실행할 때 사용한다. 만약 이 옵션이 사용되지 않으면 jeusadmin 의 기본 node 의 logger 에 대해 실행된다.

-node 는 jeusadmin 의 기본 node 이외의 다른 node 의 logger 에 대해 실행할 때 사용한다.

-f 는 하위의 logger 에 대해서 모두 loglevel 을 적용할 때 사용한다. 만약 사용하지 않은 경우 이전에 하위 logger 에 따로 level 을 설정한게 있으면 이를 overriding 하지 않는다. 참고로 이 옵션을 사용하더라도 JEUSMain.xml 에서 설정 가능한 하위 logger 의 level 은 변경되지 않는다.

-set-handler option 은 logger 의 level 을 변경할 때 이 logger 가 사용하는 handler 들의 level 을 logger 의 변경된 level 의 message 가 출력될 수 있도록 자동으로 조절하는 option 이다. 이 option 을 사용하면 이 logger 가 사용하는 handler 와 상위 logger 의 handler 의 level 이 변경될 수 있으므로 주의있게 사용해야 한다.

handlername 을 지정할 경우 이 handler 의 level 을 변경하게 된다.

loglevel 을 지정하지 않으면 현재의 설정을 보여주고 지정되어 있다면 지정된 level 로 변경한다.

B.10 기타 명령어

`help`

명령어에 대한 도움말을 출력한다.

`ftp sourcefile ftproot:/destinationfile [-u]`

파일을 다른 위치로 옮길 때 사용한다. “ftproot”는 다음 중 하나이다.

1. jeus:/ JEUS_HOME

-u 옵션은 압축된 파일을 우선 압축을 푼 후에 전송한다.

예제)

```
johan>ftp c:\test\hello.jar jeus:/hello.jar
[JeusCommander] c:\test\hello.jar -> jeus:/hello.jar ftp
      successful
```

`rftp ftproot:/sourcefile destinationfile [-u]`

앞의 명령어와 동일하다. 그러나 소스측에 “ftproot”를 사용한다.

`meminfo [|container_name]`

jeusadmin 이 접속해 있는 manager 프로세스의 memory 정보를 출력한다. container name 이 지정되면 해당 container 프로세스의 memory 정보를 출력한다.

`dump [|container_name |-a]`

jeusadmin 이 접속해 있는 manager 프로세스의 thread dump 를 출력한다. 이때 dump 는 Manager 프로세스의 console 창에 출력된다. container name 이 지정되면 해당 container 프로세스의 thread dump 가 출력된다. –a 가 주어지면 Manager 프로세스와 모든 Engine Container 의 thread dump 가 출력된다.

`mbeanlist [query_name]`

clustering 상의 JMX MBean 의 list 를 출력한다. Query name 이 주어지지 않으면 모든 MBean 이 출력된다. Query name 의 각 key 에 대해서는 JEUS JMX 참고서나 JMX API Javadoc 을 참고하기 바란다.

`exit`

jeusadmin 을 종료한다.

C. dbpooladmin Console Tool Reference

C.1 소개

본 부록에서는 JEUS_HOME\bin 디렉토리 위치에 있는 'dbpooladmin' 콘솔 어플리케이션의 모든 사용법과 특징등을 기술할 것이다.

C.2 목적

dbpooladmin 툴은 JEUS 안에 구성된 DB Connection Pool 을 제어하는데 사용된다. 일반적으로 이 pool 들은 JEUSMain.xml 의 <database>element 안에 구성된다. 그러나 pool 들은 Web container 내부적으로 추가되어질 수 있을 것이다.

DB pool 이 "global" 수준으로 구성되어졌다면 직접적으로 JEUSMain.xml 의 <jeus-system>element 아래에 있을 것이며 실제로 생성되어지거나 각 Engine Container 의 내부적으로 실행 되어질 것이다.

C.3 사용법

'dbpooladmin' 툴의 사용문법은 아래와 같다.

```
dbpooladmin [ | -verbose] [engine container name] [ |-Uuser_name
 -Ppassword]
```

파라미터:

`-verbose`

자세한 디버깅 정보를 출력한다.

`-Engine_container_name`

지정된 Engine Container (e.g. “johan_mycontainer”)에 연결한다.

예제:

```
C:\>dbpooladmin johan_mycontainer
```

프롬프트가 시작되었을 때 username 과 password 를 입력한다. 아래의 dbpooladmin 프롬프트가 나타날것이다.

```
johan_mycontainer>
```

다음 절에서는 현재 상태에서 사용할 수 있는 명령어 리스트를 볼 수 있을 것이다.

C.4 명령어들

`info [-k <times>] [-i <interval>]`

지정된 datasource 에 관한 정보를 보여준다. -k <횟수>옵션은 info 명령어를 지정된 횟수만큼 반복적으로 수행한다. -i <주기>옵션은 info 명령어가 수행될 주기를 결정한다. <주기>는 초단위로 표현되며 -k 와 -i 옵션을 사용함으로서 connection pool 를 지속적인 모니터링이 가능하다.

```
jeus_container1>info
ex)jeus_container1>info –k 3 –i 3
============================================================
id     name       min  max  current  idle  waiting  working
============================================================
1 datasource1     2    10   2      2     false     false
2 jdbc/DB1        6    10   6      6     true      true
3 localxa         6    10   6      6     true      true
============================================================
```

`enable <data_source_name>`

지정한 datasource 를 '활성화'상태로 만든다. datasource 의 이름은 JEUSMain.xml 에 설정한 export name 과 동일하다.

```
jeus_container1>enable DataSource Name
ex) jeus_container1>enable dataSource1
```

`disable <data_source_name>`

지정한 datasource 를 '비활성화'상태로 만든다.

```
jeus_container1>disable DataSource Name
ex) jeus_container1>disable dataSource1
```

`shrink <data_source_name>`

Idle Connection 들을 제거함으로서 conneciton pool 크기를 줄인다.

```
jeus_container1>shrink DataSource Name
ex) jeus_container1>shrink dataSource1
```

`resync <data_source_name>`

XA transaction 이 DB 의 장애로 인해 중단되었을 때 재시작한다.

```
jeus_container1>resync DataSource Name
ex) jeus_container1>resync dataSource1
```

주의: 만약 DB 가 XA transactions 중에 중단되었다면, DB 가 살아나자마자 이 명령어를 실행시켜준다.

`reconfig <data_source_name>`

지정된 datasource 의 사용자명, 패스워드, pool min, pool max, pool step 을 재구성한다. 그러나 JEUSMain.xml 의 구성은 변경되지 않는다. 이 변경은 단지 JEUS Manager 가 실행하는 동안 지속된다.

예제:

```
jeus_container1>reconfig datasource1
type properties
reconfig>user scott
reconfig>password tiger
reconfig>.
jeus_container1>
```

위의 명령어는 DB username 을 "scott", password 를 "tiger"로 변경시킨다. "."을 사용하여서 reconfig session 을 종결시킨다. 이 명령어을 수행하여 원하는 결과를 얻기 위해서는 위의 예제에서 bold 체 부분을 확실히 해야한다.

`help`

dbpooladmin 에서 이용 가능한 명령어를 보여준다.

```
jeus_container1>help
```

```
============================================================
dbpooladmin command
 - info : gather informations for connection pools in the engine
container
    ex> info : monitoring all connection pools
    ex> info -k count -i interval : monitoring all connection
pools for the number of specified count for each interval.
 - enable [connection pool name] : enable the connection pool
    ex> enable mydbpool -> enable mydbpool
 - disable [connection pool name] : disable the connection pool
    ex> disable mydbpool -> disable mydbpool
 - shrink [connection pool name] : shrink the size of the
connection pool by closing idle connections
    ex> shrink mydbpool -> shrink mydbpool
 - resync [connection pool name] : resync global transaction
statue for the connection pool
    ex> resync mydbpool -> recover the status of the global
transaction done with mydbpool

 - reconfig [connection pool name] : reconfigurate configuration
parameters
    ex> reconfig mydbpool -> reconfig mydbpool
============================================================
```

D. tmadmin Console Tool Reference

D.1 소개

시스템 매니저는 이 콘솔 툴을 사용해서 리소스 매니저를 복구(Recovery)한다. 이 툴로 복구할 수 있는 리소스 매니저로는 DB, JMS 와 Java Connector Architecture(JCA)를 사용해서 연결된 리소스 매니저가 있다.

D.2 사용법

사용법은 다음과 같다.

```
tmadmin [–verbose] Engine_container_name [node_name]
```

`-verbose`

옵션 기능으로, 복구를 진행할 때 자세한 정보를 출력한다.

`Engine_container_name`

복구할 리소스 매니저에 연결된 Connection Pool Manager 가 올라가 있는 Engine Container 이름을 입력한다. 필수 입력 사항이다. 복구할 리소스 매니저의 개수가 여러 개라면, 각각 툴을 실행시켜서 복구한다.

`node_name`

옵션 기능으로, 접속하려면 Engine Container 가 backup Engine Container 로 작동하는 경우에 입력한다. 입력하는 노드명은 backup Engine Container 가 작동하는 노드의 이름이다.

D.3 명령어

tmadmin 의 명령어로는 트랜잭션 복구를 진행하도록 하는 명령어, 하나 뿐이다. 명령어 형식은 다음과 같다.

```
resync <type of transactional resource manager> <connection pool manager name>
```

파라미터:

`<type of transactional resource manager>`

복구하고자하는 리소스 매니저의 종류를 지정한다. 이 값으로는 “sql”, “jms”, “connector”가 있다. 각각 DB, JMS, JCA 를 나타낸다.

`<connection pool manager name>`

복구할 리소스 매니저와 접속되어 있는 Connection Pool Manager 의 export-name 을 입력한다.

E appcompiler 콘솔 툴 레퍼런스

E.1 소개

이 문서에서는 Application 의 EJB stub class 나 webservice 의 endpoint class 등을 생성하는 application compiler 에 대해 설명한다.

appcompiler 는 하나의 file 을 생성한 후 컴파일하는 each 모드와 모든 파일을 생성한 후 한번에 컴파일하는 batch 모드가 있다. batch 모드는 each 모드에 비해 속도는 빠르지만 컴파일 시 에러가 나면 원인이 되는 파일을 찾기 힘들다는 단점이 있다. default 는 each 모드이므로 batch 모드를 원할 경우 jeus.ejb.compiler.mode 를 설정하면 된다.

JEUS_HOME\bin\ 디렉토리에서 appcompiler 스크립트를 찾을 수 있다.

E.2 목적

이 툴은 pre-deployment 후에 EJB 모듈(또는 개개의 bean)에 필요한 RMI stub 과 skeleton 클래스, 또는 webservice endpoint class 들을 미리 생성하기 위해 선택적으로 사용할 수 있다.

appcompiler 툴은 부팅때마다 또는 runtime-deployment 가 실행될 때 자동으로 실행된다. 이는 상당한 시간이 소모되는 작업이기 때문에, 엔진 부팅시에 많은 양의 bean 들이 Deploy 될 경우에는 더 유용하다.

EJB 엔진의 부팅시간을 줄이려고 하면(또는 수동 runtime-deployment 를 빠르게 진행하려면) EJB 엔진을 시작하기 전에(또는 수동으로 runtime-deployment 를 하기전에) 이 툴을 실행하는 것이 효과적이다. 이 경우에 EJB 엔진 설정파일에 ‘fast deploy”설정을 하여야 한다.(또는 ejbadmin tool 의 ‘deploy’ 명령어에 “-f”옵션을 추가한다) fast deploy 옵션을 “true”로 설정하는 것으로 appcompiler 가 자동으로 수행되는 것을 막을 수 있다.

E.3 호출

appcompiler 는 JEUS_HOME\bin\ 에 위치한 일반적인 스크립트 파일이다.

appcompiler 툴을 실행하는데에는 다음과 같은 문법이 사용된다.

```
appcompiler [-verbose] [<ear_or_jar_file_path>| <directory_path>]
```

이 파라미터들은 다음과 같은 해석을 가진다.

`ear_or_jar_file_path`

appcompiler 를 실행할 archive file path. 만약 ear file 이면 ear file 내의 모든 EJB 모듈에 대해서 appcompiler 를 실행한다.

`directory_path`

appcompiler 를 실행할 directory path. 만약 directory 의 내용이 EAR 이면 이 directory 내의 모든 EJB 모듈에 대해서 appcompiler 를 실행한다.

예)

```
C:\> appcompiler countermod.jar
```

위의 명령어는 “countermod.jar” 모듈의 home, remote, client 클래스를 생성해서 jar file 에 포함시킨다.

F JEUS Manager Properties

F.1 소개

아래[표 6]은 JEUS Manager 가 적용가능한 유용하고 일반적인 “-D” 옵션을 나타낸다. 이 셋팅은 JEUS Manager 에 의해 관리된 모든 Engine Container 에 복사된다.

이 Java 환경 변수는 “-D” 로 전환한 후 덧붙여지고 사용 값은 “=” 문자 뒤에 덧붙여진다. 다음은 예이다:

```
-Djeus.baseport=9999
```

상기 샘플이 ‘jeus’ 스크립트에서 “-classpath” 소절 뒤에 덧붙여지나 클래스 명 앞이면 JEUS Manager 기본 포트는 9999 로 설정된다.

기본적으로, ‘jeus’ script 에서 JVM 속성은 시스템 환경변수의 값으로 할당된다 (JEUS_HOME 이 “jeus.home” 으로 할당되는 것 처럼). 시스템 환경 변수는 3 장에서 자세히 설명한다.

참고: 환경 변수는 독립 Engine Container 에 실행되는 개별 JVM 에 추가된다. 이런 JVM 파라미터는 JEUSMain.xml 파일의 Engine Container 에 지정되어야 한다. 자세한 내용은 11 장을 참고한다.

이렇게 많은 –D 옵션을 정의하지만 이 옵션은 전용 문서나 장에서 설명하게 된다. 아래의 리스트된 속성은 JEUS Manager 의 동작에 직접적으로 속하게 된다.

부록 G, H 에서는 JTS 와 JNDI 의 추가 속성에 대하여 정리되어 있다. JEUS EJB 안내서 와 JEUS Web Container 안내서에 EJB 와 Servlet 에 대한 추가적인 “-D” 속성을 다루고 있다.

F.2 환경 변수 참조

표 6. JEUS Manager Properties.

JEUS Manager Properties	의미	Default 값(또는 예)
jeus.home	JEUS 의 Home directory 를 가리키는 property 이다.	예) c:\jeus
jeus.server.classpath	JEUS 의 각각의 Root Classloader 에 적용할 User classpath 를 지정한다.	예)
jeus.baseport	JEUS 에서 사용하는 base port	9736
jeus.logging.initmsg	log message 를 처음에 모두 initialize 할지를 결정한다.	false
jeus.server.checktmout	JEUS 를 관리하는 RMI Connection 이나 JMX Connector 에 대해 적용할 timeout	60 * 1000 (ms)
jeus.config.home	JEUS 의 config home 을 지정한다	JEUS_HOME/config
jeus.log.home	JEUS 의 log home 을 지정한다	JEUS_HOME /logs
jeus.earhome	JEUS 의 ear home 을 지정한다. 이 설정은 JEUS 4.x 와의 호환을 위한 설정이다.	JEUS_HOME /webhome/ear_home
JEUS Manager Properties	의미	**Default 값(또는 예)**

jeus.deployhome	JEUS 의 deploy home 을 지정한다.	JEUS_HOME /webhome/deploy_home
jeus.servlethome	JEUS 의 servlet home 을 지정한다.	JEUS_HOME /webhome/servlet_home
jeus.clienthome	JEUS 의 client home 을 지정한다. 이 설정은 JEUS 4.x 와의 호환을 위해 설정한다.	JEUS_HOME /webhome/client_home
jeus.apphome	JEUS 의 application home 을 지정한다	JEUS_HOME /webhome/app_home
jeus.logging.propertyfile	JEUS 의 logger 들에 대해 Java Logging API 에서 제공하는 logging property file 을 적용하고 싶을때 사용한다.	JEUS_CONFIG_HOME/jeuslogging.properties
jeus.vhost.file	virtual host descriptor (vhost.xml)의 위치를 지정한다.	JEUS_CONFIG_HOME/vhost.xml
jeus.jndi.cluster.readtimeout	Clustering 환경의 Remote JNSServer 들간에 사용되는 Socket 에 대한 Read Timeout value	30 * 1000 (ms)
jeus.jndi.cluster.connecttimeout	Clustering 환경의 Remote JNSServer 에 접속시도시 상대방의 Socket 에 접속하기 위해서 허용되는 적용되는 Timeout value	10 (sec)
JEUS Manager Properties	**의미**	**Default 값(또는 예)**

jeus.jndi.cluster.checkcount	Clustering 환경의 Remote JNSServer 간의 최초 Data 교환시 read timeout 등의 이유로 인하여 문제점 발생시 retry 할 count value	2
jeus.jndi.cluster.url	JNDI 의 clustering 에 적용되는 JEUS Manager 를 ip:port,ip:port 형태로 나열한다.	
jeus.jndi.local.tmout	JNDI 에서 request 를 보낸 후 응답이 올때까지 기다리는 시간	20 * 1000(ms)
jeus.deploy.default	deploy 할 때 descriptor 가 없는 경우 사용하는 ddinit.properties 의 위치를 설정한다.	jeus/config/node_name/ddinit.properties
jeus.ssl.keystore	SSL 에서 사용하는 keystore 를 지정한다.	jeus/config/node_name/keystore
jeus.ssl.keypass	SSL 에서 사용하는 keypass 를 지정한다.	jeuskeypass
jeus.ssl.truststore	SSL 에서 사용하는 truststore 를 지정한다.	jeus/config/node_name/truststore
jeus.ssl.trustpass	SSL 에서 사용하는 trustpass 를 지정한다.	jeustrustpass

G. JEUS JTS Properties

G.1 소개

JEUS 트랜잭션 매니저를 컨트롤하는 프로퍼티는 JVM 의 –D 옵션으로 세팅된다. 클라이언트 어플리케이션에서는 스크립트에 직접 옵션을 추가해 줘야 한다.

Engine Container 의 트랜잭션 매니저를 설정하려면 JEUSMain.xml 의 <tm-config> 를 사용하거나 <command-option>을 사용한다. 클라이언트의 트랜잭션 매니저를 설정하려면 JVM 옵션만이 제공된다. 그러나 JVM 옵션은 Engine Container 나 클라이언트나 비슷하게 사용한다.

G.2 프로퍼티 레퍼런스

표 7. JTS 환경 변수

프로퍼티	Description	값
jeus.tm.version	사용할 JEUS 트랜잭션 매니저의 타입.	‘server’ ‘client’ (default)
jeus.tm.port	JEUS 트랜잭션 매니저가 사용할 TCP/IP 포트 번호	
jeus.tm.tmMin	부팅될 때 생성될 Worker Thread Pool 의 개수	Default: 2
jeus.tm.tmMax	Work Thread 가 생성될 수 있는 최대 개수	Default: 30

프로퍼티	Description	값
jeus.tm.resizingPeriod	더 이상 사용하지 않는 쓰레드를 줄이기 위한 시간 간격. millisecond	Default: 3600000
jeus.tm.tmResolution	타임 아웃 메커니즘의 기본 시간 간격 **참고**: 만약 이 값이 타임 아웃값 보다 크면, 타임 아웃이 제대로 작동하지 않는다. 그러므로 이 값은 타임 아웃 메커니즘이 정상적으로 작동할 수 있을 정도로 작게 설정한다.	Default: 60000
jeus.tm.activeto	Global Transaction 이 시작되면, 이 프로퍼티의 시간 안 에 commit 되어야 한다. 그렇지 않으면 강제 rollback 되어 버린다.	Default: 600000
jeus.tm.prepareto	Root Coordinator 는 이 시간 내에 Sub Coordinator 와 리소스 매니저로부터 'prepare' 신호를 받아야 한다. 만약 받지를 못하면 Root Coordinator 는 Global Transaction 을 rollback 시킨다.	Default: 120000

프로퍼티	Description	값
jeus.tm.preparedto	Sub Coordinator 는 자신의 Root Coordinator 로부터 이 시간 안에 commit 을 해야 할지, rollback 을 해야 할지를 나타내는 global decision 을 받아야 한다. 만약 이 시간 내에 받질 못하면, Root Coordinator 로 다시 'prepare'에 대한 응답 메시지를 보낸다. 그래도 여전히 시간 내에 global decision 이 오지 않는다면, <heuristic-rollback>의 값이 true 일 때 Global Transaction 을 rollback 시켜버린다. <heuristic-rollback>이 false 이면, Root Coordinator 로 메시지를 보내고 global decision 을 기다리기를 계속 한다.	Default: 60000
jeus.tm.committo	Root Coordinator 는 이 시간 내에 Sub Coordinator 와 리소스 매니저로부터 'commit'이나 'rollback' 에 대한 결과를 받아야 한다. 만약 결과가 오지 않으면, Root Coordinator 는 Global Transaction 을 'Uncompleted List' 에 기록해서, 트랜잭션이 완전히 끝나지 않았음을 남겨둔다.	Default: 240000
jeus.tm.recoveryto	이 값은 트랜잭션 복구 시에 사용된다. 트랜잭션 매니저는 트랜잭션 복구를 위해서 복구될 트랜잭션 정보를 가져오려고 한다. 만약 다른 트랜잭션 매니저에서 이 시간 내에 복구 정보를 알려주지 않으면, <heuristic-rollback>이 true 일 때, 해당 트랜잭션을 rollback 시킨다. 값이 false 이면 트랜잭션 복구를 시스템 관리자에게 남겨두고 더 이상 진행하지 않는다.	Default: 120000

프로퍼티	Description	값
jeus.tm.uncompletedto	트랜잭션 매니저는 전체 트랜잭션 처리를 완료하기 위해, 실패한 Global Transaction 의 목록을 보관한다. 완료되지 못한 Global Transaction 의 정보는 복구 처리시에 사용되므로, 이 타임 아웃 시간까지 보관된다. 그러므로 이 시간이 너무 짧으면 복구 정보가 빨리 지워지게 되고, 트랜잭션 매니저가 해당 Global Transaction 의 무결성을 복구할 수 없게 된다. 그 결과 Global Transaction 복구를 위해서, 시스템 관리자가 많은 작업을 직접 처리해야만 한다.	Default: 86400000
jeus.tm.noLogging	JEUS 트랜잭션 매니저가 복구(Recovery)를 위한 로깅을 할지 안할지 지정한다.	true (no logs) false (logging, default)
jeus.tm.capacity	이 값을 사용해서 JEUS 트랜잭션 매니저는 내부 구조를 최적화시킨다. **참고**: 트랜잭션 매니저가 동시에 처리하는 Global Transaction 의 개수를 고려해서 값을 정한다.	Default: 10000
jeus.tm.forcedReg	트랜잭션 매니저가 'eager' 메커니즘을 쓸지, 'lazy' 메커니즘을 쓸지 지정한다.	true ('eager', default) false ('lazy')

프로퍼티	Description	값
jeus.tm.checkReg	Sub Coordinator 가 Root Coordinator 에 등록되는 것은 병렬로 할지 결정한다.	true (default) false (parallel)
jeus.tm.heuristicRollback	어떤 이유로 Global Transaction 이 완료되지 못하는 경우, 이 값이 true 이면 트랜잭션 매니저가 rollback 시켜 버리고, false 라면 시스템 관리자가 처리하도록 Uncompleted List 에 남겨둔다.	true false (default)

H. JEUS JNDI Properties

H.1 소개

JEUS JNDI 에서 사용되는 프로퍼티에 대해서 소개한다.

H.2 프로퍼티 레퍼런스

표 8. JNDI 프로퍼티와 JVM-옵션

프로퍼티 (코드 상에서)	**설명**	**JVM 옵션**
`INITIAL_CONTEXT_FACTORY` (필 수)	이 프로퍼티에 `jeus.jndi.JEUSContextFactory` 를 넣어서 JEUS 의 InitialContext 를 생성한다.	`-Djava.naming.factory.initial=jeus.jndi.JEUSContextFactory`
`URL_PKG_PREFIXES`	JEUS 의 InitialContext 에서 URL scheme 을 사용해서 lookup 하기 위해 `jeus.jndi.jns.url` 를 세팅한다.	`-Djava.naming.factory.url.pkgs=jeus.jndi.jns.url`
`PROVIDER_URL`	JNSServer 의 IP 주소를 세팅한다. Default Value 은 127.0.0.1 이다.	`-Djava.naming.provider.url=<host_address>`

프로퍼티 (코드 상에서)	설명	JVM 옵션
SECURITY_ PRINCIPAL	JEUS Naming Server 에서 인증받기 위한 사용자명을 입력한다. 사용자명이 세팅되지 않으면 "guest"로 처리된다. 사용자명은 JEUS Security Domain 에서 정의되어 있어야 한다.	-Djava.naming.security.principal=<*username*>
SECURITY_ CREDENTIALS	JEUS Naming Server 에서 인증 받기 위한 패스워드를 입력한다.	-Djava.naming.security.credentials=<*password*>
RESOLUTION	JNSLocal 이 리소스를 관리하기 위해서 사용하는 시간 간격을 millisecond 단위로 넣는다. Default Value 은 30000(30초)	N/A
THREAD_POOLING	쓰레드 풀을 설정한다. 형식은 다음과 같다. "MinSize:MaxSize:Step:ResizingPreoid" Default Value 은 2:10:2:60000	N/A

프로퍼티 (코드 상에서)	설명	JVM 옵션
CONNECT_TIMEOUT	JNSLocal 이 JNSServer 로 접속하는데 사용하는 타임아웃 시간을 지정한다. 이 시간 동안 접속을 시도하다가 시간이 초과되면 Exception 이 발생한다. Default Value 60(60 초)	N/A
CONNECTION_ DURATION	JNSLocal 이 JNSServer 로 접속한 커넥션을 얼마나 유지할지 정한다. 여기에 지정한 시간동안 커넥션을 사용하지 않으면 자동으로 커넥션이 종료된다. Default Value 은 –1 로 계속 유지된다.	N/A
REPLICATE_ BINDINGS	bind 하는 전체 JNDI 트리 전체에서 공유할 것인지 결정한다. Default Value 은 true 로, 객체를 공유한다.	N/A
CACHE_BINDINGS	lookup 한 객체를 JNSLocal 에서 캐싱한다. Default Value 은 true.	N/A
LOCAL_BINDINGS	객체를 자신의 JVM 에 있는 JNSLocal 에만 bind 한다. Default Value 은 false	N/A

I. 주요 벤더의 JDBC Data Source 구성 예

I.1 소개

이번 부록에서는 따르는 주요 DB 밴더들과 JDBC 드라이버를 위한 data source 구성을 예제로서 제공한다.

- Oracle Thin, type 4.
- Oracle OCI, type 2.
- DB2.
- Sybase jConnect.
- MSSQL, type 1.
- MSSQL, type 4.
- Informix, type 4.

이 XML 샘플들은 JEUSMain.xml 안에 위치한다. 각 예제에서 중요한 값들은 bold 체로 표현할 것이다.

몇몇 Database 의 Connection Pool Datasource, Local XA Datasource, XA Datasource 의 구성 예를 보여준다.

I.2 Oracle Thin (Type 4) 구성 예제

I.2.1 Oracle Thin Connection Pool Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
```

```
            <database>
                < vendor>oracle</ vendor>
                <export-name>datasource1</export-name>
                <data-source-class-name>
                  oracle.jdbc.pool.OracleConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <database-name>orcl</database-name>
                <data-source-name>
                  oracle.jdbc.pool.OracleConnectionPoolDataSource
                </data-source-name>
                <description>Customer DB</description>
                <password>tiger</password>
                <encryption>false</encryption>
                <port-number>1521</port-number>
                <server-name>192.168.1.33</server-name>
                <user>scott</user>
                <driver-type>thin</driver-type>
                <connection-pool>
                    <pooling>
                        <min>2</min>
                        <max>4</max>
                        <step>1</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>true</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

I.2.2 Oracle Thin XA Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                < vendor>oracle</ vendor>
                <export-name>xaoralce</export-name>
                <data-source-class-name>
                    oracle.jdbc.xa.client.OracleXADataSource
                </data-source-class-name>
                <database-name>ora9</database-name>
                <data-source-name>
                    oracle.jdbc.xa.client.OracleXADataSource
                </data-source-name>
                <port-number>1521</port-number>
                <server-name>192.168.1.49</server-name>
                <user>scott</user>
                <driver-type>thin</driver-type>
                <password>tiger</password>
                <data-source-type>XADataSource</data-source-type>
                <connection-pool>
                    <pooling>
                        <min>2</min>
                        <max>10</max>
                        <step>2</step>
                        <period>1800000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>false</enable-wait>
                        <wait-time>100000</wait-time>
                    </wait-free-connection>
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
```

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

I.3 Oracle OCI (Type 2) 구성 예제

I.3.1 Oracle OCI Connection Pool Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                < vendor>oracle</ vendor>
                <export-name>test_or</export-name>
                <data-source-class-name>
                  oracle.jdbc.pool.OracleConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <database-name>orcl</database-name>
                <data-source-name>
                  oracle.jdbc.pool.OracleConnectionPoolDataSource
                </data-source-name>
                <description>Customer DB</description>
                <password>tiger</password>
                <encryption>false</encryption>
                <port-number>1521</port-number>
                <server-name>pollux5</server-name>
                <user>scott</user>
                <driver-type>oci8</driver-type>
                <connection-pool>
                    <pooling>
                        <min>6</min>
                        <max>10</max>
                        <step>2</step>
                        <period>30000</period>
                    </pooling>
                    <wait-free-connection>
```

```
                        <enable-wait>true</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

I.4 DB2 구성 예제

I.4.1 DB2 Local XA Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                <vendor>db2</vendor>
                <export-name>db2local</export-name>
                <data-source-class-name>
                    COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    LocalXADataSource
                </data-source-type>
                <database-name>SAMPLE</database-name>
                <description>Customer DB</description>
                <password>jeus2013</password>
                <encryption>false</encryption>
                <port-number>0</port-number>
                <server-name/>
                <user>db2inst1</user>
                <connection-pool>
                    <pooling>
```

```
                        <min>2</min>
                        <max>4</max>
                        <step>1</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>true</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

I.4.2 DB2 XA Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                <vendor>db2</vendor>
                <export-name>xadb2</export-name>
                <data-source-class-name>
                    COM.ibm.db2.jdbc.DB2XADataSource
                </data-source-class-name>
                <data-source-type>XADataSource</data-source-type>
                <database-name>sample</database-name>
                <description>Test DB</description>
                <password>db2inst1</password>
                <encryption>false</encryption>
                <port-number>6788</port-number>
                <server-name>192.168.1.11</server-name>
                <user>db2inst1</user>
                <connection-pool>
```

```
                    <pooling>
                        <min>2</min>
                        <max>4</max>
                        <step>1</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>false</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

I.5 Sybase jConnect 구성 예제

I.5.1 Sybase Connection Pool Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                <vendor>sybase</vendor>
                <export-name>sybasedatasource</export-name>
                <data-source-class-name>
                com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <database-name>testdb</database-name>
                <data-source-name>
```

```
                com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource
                </data-source-name>
                <description>Test DB</description>
                <network-protocol>Tds</network-protocol>
                <password>sybase</password>
                <encryption>false</encryption>
                <port-number>4100</port-number>
                <server-name>192.168.1.10</server-name>
                <user>sa</user>
                <connection-pool>
                    <pooling>
                        <min>10</min>
                        <max>50</max>
                        <step>5</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>false</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

I.5.2 Sybase XA Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                <vendor>sybase</vendor>
                <export-name>XAsybase</export-name>
                <data-source-class-name>
```

```
                    com.sybase.jdbc2.jdbc.SybXADataSource
                </data-source-class-name>
                <data-source-type>
                    XADataSource
                </data-source-type>
                <database-name>testdb</database-name>
                <data-source-name>
                    com.sybase.jdbc2.jdbc.SybXADataSource
                </data-source-name>
                <description>Customer DB</description>
                <network-protocol>Tds</network-protocol>
                <password>sybase</password>
                <encryption>false</encryption>
                <port-number>4100</port-number>
                <server-name>serverIP</server-name>
                <user>sa</user>
                <connection-pool>
                    <pooling>
                        <min>10</min>
                        <max>50</max>
                        <step>5</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>true</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

I.6 MSSQL (Type 1) 구성 예제

I.6.1 MSSQL Type 1 Connection Pool Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                <vendor>jeus_mssql</vendor>
                <export-name>mssqldatasource</export-name>
                <data-source-class-name>
                    jeus.jdbc.driver.mssql.MSSQLConnectionPool
DataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <database-name/>
                <data-source-name/>
                <description/>
                <network-protocol/>
                <password>password</password>
                <encryption>false</encryption>
                <port-number/>
                <server-name>ssa</server-name> <!--  ODBC DNS -->
                <user>sa</user>
                <connection-pool>
                    <pooling>
                        <min>10</min>
                        <max>20</max>
                        <step>2</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>true</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
```

```
                </connection-pool>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

참고: ODBC 설정은 JDBC 보다 먼저 설정되어 있어야 한다.

I.7 MSSQL (Type 4) 구성 예제

I.7.1 MSSQL Type 4 Connection Pool Datasource

<<JEUSMain.xml>>

```
<jeus-system>
    ...
    <resource>
        <data-source>
            <database>
                <vendor>others</vendor>
                <export-name>datasource1</export-name>
                <data-source-class-name>
                    jeus.jdbc.driver.blackbox.BlackboxConnection
PoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <connection-pool>
                    <pooling>
                        <min>4</min>
                        <max>4</max>
                        <step>1</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>true</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
```

```
                    <operation-to>30000</operation-to>
                </connection-pool>
                <property>
                    <name>URL</name>
                    <type>java.lang.String</type>
                    <value>
                    jdbc:microsoft:sqlserver://192.168.1.209:1433
                    </value>
                </property>
                <property>
                    <name>DriverClassName</name>
                    <type>java.lang.String</type>
                    <value>
                    com.microsoft.jdbc.sqlserver.SQLServerDriver
                    </value>
                </property>
                <property>
                    <name>User</name>
                    <type>java.lang.String</type>
                    <value>sa</value>
                </property>
                <property>
                    <name>Password</name>
                    <type>java.lang.String</type>
                    <value>jeus</value>
                </property>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

I.8 Informix (Type 4) 구성 예제

I.8.1 Informix Connection Pool Datasource

<<JEUSMain.xml>>

```
<jeus-system>
```

```
    ...
    <resource>
        <data-source>
            <database>
                <vendor>others</vendor>
                <export-name>informixdatasource</export-name>
                <data-source-class-name>
                    jeus.jdbc.driver.blackbox.BlackboxConnection
PoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <connection-pool>
                    <pooling>
                        <min>4</min>
                        <max>8</max>
                        <step>1</step>
                        <period>600000</period>
                    </pooling>
                    <wait-free-connection>
                        <enable-wait>false</enable-wait>
                        <wait-time>10000</wait-time>
                    </wait-free-connection>
                    <operation-to>30000</operation-to>
                </connection-pool>
                <property>
                    <name>URL</name>
                    <type>java.lang.String</type>
                    <value>
                  jdbc:informixsqli://192.168.1.10:1526/kangto94:
                  informixserver=ids921
                    </value>
                </property>
                <property>
                    <name>DriverClassName</name>
                    <type>java.lang.String</type>
                    <value>com.informix.jdbc.IfxDriver</value>
                </property>
```

```
                <property>
                    <name>User</name>
                    <type>java.lang.String</type>
                    <value>informix</value>
                </property>
                <property>
                    <name>Password</name>
                    <type>java.lang.String</type>
                    <value>hope123</value>
                </property>
            </database>
            ...
        </data-source>
        ...
    </resource>
</jeus-system>
```

J. JEUSMain.xml XML Configuration Reference

J.1 소개

본 부록의 레퍼런스는 JEUS 의 메인 설정 파일인 JEUSMain.xml 의 모든 태그에 대해서 설명하고 있다. 이 파일의 xsd 파일은 “JEUS_HOME\config\xsds” 디렉토리의 “jeus-main.xsd” 파일이다.

본 레퍼런스는 3 부분으로 나눠져있다.

1. **XML Schema/XML 트리**: XML 설정 파일의 모든 태그 리스트를 정리했다. 각 노드의 형식은 다음과 같다.
    a. 태그 레퍼런스로 빨리 찾아보기위해서 각 태그마다 인덱스 번호(예: (11))를 붙여놓았다. 태그 레퍼런스에서는 이 번호 순서로 설명한다.
    b. XML Schema 에서 정의한 XML 태그명을 <**tag name**> 형식으로 표시한다.
    c. XML Schema 에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element
    d. 몇몇 태그에는 “P” 문자를 붙여놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.
2. **태그 레퍼런스**: 트리에 있는 각 XML 태그를 설명한다.
    a. **Description**: 태그에 대한 간단한 설명.
    b. **Value Description**: 입력하는 값과 타입.
    c. **Value Type**: 값의 데이터 타입. 예) String

d. **Default Value**: 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값.

f. **Example**: 해당 XML 태그에 대한 예.

g. **Performance Recommendation**: 성능 향상을 위해서 추천하는 값.

h. **Child Elements**: 자신의 태그 안에 사용하는 태그.

3. **Example XML 파일**: "JEUSMain.xml"에 대한 완전한 예제.

J.2 XML Schema/XML 트리

```
(1) <jeus-system>
   (2) <node>+
      (3) <name>
      (4) <listener>?
         (5) <backlog>? P
         (6) <ssl>?
            (7) <port>?
         (8) <thread-pool>?
            (9) <min>? P
            (10) <max>? P
            (11) <period>? P
      (12) <backup-node>?
      (13) <engine-container>*
         (14) <name>
         (15) <command-option>?
         (16) <engine-command>*
            (17) <type>
            (18) <name>
            (19) <system-logging>?
               (20) <level>? P
               (21) <use-parent-handlers>? P
               (22) <filter-class>?
               (23) <handler>?
                  (24) <console-handler>
```

```
      (25) <name>
      (26) <level>? P
      (27) <encoding>?
      (28) <filter-class>?
   (29) <file-handler>
      (30) <name>
      (31) <level>? P
      (32) <encoding>?
      (33) <filter-class>?
      (34) <file-name>?
      (35) <valid-day>
      (36) <valid-hour>
      (37) <buffer-size>? P
      (38) <append>? P
   (39) <smtp-handler>
      (40) <name>
      (41) <level>? P
      (42) <encoding>?
      (43) <filter-class>?
      (44) <smtp-host-address>
      (45) <from-address>
      (46) <to-address>
      (47) <cc-address>?
      (48) <bcc-address>?
      (49) <send-for-all-messages>? P
   (50) <socket-handler>
      (51) <name>
      (52) <level>? P
      (53) <encoding>?
      (54) <filter-class>?
      (55) <address>
      (56) <port>
   (57) <user-handler>
      (58) <handler-class>
      (59) <name>
      (60) <level>? P
      (61) <encoding>?
      (62) <filter-class>?
      (63) <handler-property>?
```

```
                  (64) <property>*
                     (65) <key>
                     (66) <value>
               (67) <formatter-class>? P
               (68) <formatter-property>?
                  (69) <property>*
                     (70) <key>
                     (71) <value>
(72) <enable-interop>?
   (73) <use-OTS> P
(74) <sequential-start>? P
(75) <user-class-path>?
(76) <tm-config>?
   (77) <use-nio>? P
   (78) <pooling>?
      (79) <min>? P
      (80) <max>? P
      (81) <period>? P
   (82) <active-timeout>? P
   (83) <prepare-timeout>? P
   (84) <prepared-timeout>? P
   (85) <commit-timeout>? P
   (86) <recovery-timeout>? P
   (87) <uncompleted-timeout>? P
   (88) <capacity>? P
   (89) <heuristic-rollback>? P
   (90) <resolution>? P
(91) <scheduler>?
   (92) <default-lookup-name>?
   (93) <thread-pool>?
      (94) <min>? P
      (95) <max>? P
      (96) <period>? P
   (97) <job-list>?
      (98) <job>*
         (99) <class-name>
         (100) <name>?
         (101) <description>?
         (102) <begin-time>?
```

```
            (103) <end-time>?
            (104) <interval>?
               (105) <millisecond>
               (106) <minutely>
               (107) <hourly>
               (108) <daily>
            (109) <count>? P
(110) <user-logging>?
   (111) <level>? P
   (112) <use-parent-handlers>? P
   (113) <filter-class>?
   (114) <handler>?
      (115) <console-handler>
         (116) <name>
         (117) <level>? P
         (118) <encoding>?
         (119) <filter-class>?
      (120) <file-handler>
         (121) <name>
         (122) <level>? P
         (123) <encoding>?
         (124) <filter-class>?
         (125) <file-name>?
         (126) <valid-day>
         (127) <valid-hour>
         (128) <buffer-size>? P
         (129) <append>? P
      (130) <smtp-handler>
         (131) <name>
         (132) <level>? P
         (133) <encoding>?
         (134) <filter-class>?
         (135) <smtp-host-address>
         (136) <from-address>
         (137) <to-address>
         (138) <cc-address>?
         (139) <bcc-address>?
         (140) <send-for-all-messages>? P
      (141) <socket-handler>
```

```
            (142) <name>
            (143) <level>? P
            (144) <encoding>?
            (145) <filter-class>?
            (146) <address>
            (147) <port>
         (148) <user-handler>
            (149) <handler-class>
            (150) <name>
            (151) <level>? P
            (152) <encoding>?
            (153) <filter-class>?
            (154) <handler-property>?
               (155) <property>*
                  (156) <key>
                  (157) <value>
            (158) <formatter-class>? P
            (159) <formatter-property>?
               (160) <property>*
                  (161) <key>
                  (162) <value>
(163) <system-logging>?
   (164) <level>? P
   (165) <use-parent-handlers>? P
   (166) <filter-class>?
   (167) <handler>?
      (168) <console-handler>
         (169) <name>
         (170) <level>? P
         (171) <encoding>?
         (172) <filter-class>?
      (173) <file-handler>
         (174) <name>
         (175) <level>? P
         (176) <encoding>?
         (177) <filter-class>?
         (178) <file-name>?
         (179) <valid-day>
         (180) <valid-hour>
```

```
            (181) <buffer-size>? P
            (182) <append>? P
         (183) <smtp-handler>
            (184) <name>
            (185) <level>? P
            (186) <encoding>?
            (187) <filter-class>?
            (188) <smtp-host-address>
            (189) <from-address>
            (190) <to-address>
            (191) <cc-address>?
            (192) <bcc-address>?
            (193) <send-for-all-messages>? P
         (194) <socket-handler>
            (195) <name>
            (196) <level>? P
            (197) <encoding>?
            (198) <filter-class>?
            (199) <address>
            (200) <port>
         (201) <user-handler>
            (202) <handler-class>
            (203) <name>
            (204) <level>? P
            (205) <encoding>?
            (206) <filter-class>?
            (207) <handler-property>?
               (208) <property>*
                  (209) <key>
                  (210) <value>
            (211) <formatter-class>? P
            (212) <formatter-property>?
               (213) <property>*
                  (214) <key>
                  (215) <value>
(216) <invocation-manager-action>?
(217) <jmx-manager>?
   (218) <jmx-connector>?
      (219) <jmxmp-connector>?
```

```
            (220) <jmxmp-connector-port>? P
         (221) <rmi-connector>?
            (222) <rmi-connector-port>?
            (223) <export-name>?
            (224) <ref-export-name>?
      (225) <html-adaptor-port>?
      (226) <snmp-adaptor>?
         (227) <snmp-adaptor-port>
         (228) <snmp-version>? P
         (229) <snmp-max-packet-size>? P
         (230) <snmp-security>? P
         (231) <trap-demon>*
            (232) <ip-address>
            (233) <port>
         (234) <pooling>?
            (235) <min>? P
            (236) <max>? P
            (237) <period>? P
      (238) <mlet-url>*
   (239) <auto-deploy>?
      (240) <enable-auto-deploy>? P
      (241) <auto-deploy-path>?
      (242) <auto-deploy-check-interval>? P
   (243) <use-MEJB>? P
   (244) <lifecycle-invocation>*
      (245) <class-name>
      (246) <invocation>+
         (247) <invocation-method>
            (248) <method-name>
            (249) <method-params>?
               (250) <method-param>*
         (251) <invocation-argument>*
         (252) <invocation-type>
   (253) <application-path>*
   (254) <res-ref>?
      (255) <jndi-info>+
         (256) <ref-name>
         (257) <export-name>
(258) <class-ftp>?
```

```
(259) <security-switch>? P
(260) <sequential-start>?
(261) <scheduler>?
   (262) <default-lookup-name>?
   (263) <thread-pool>?
      (264) <min>? P
      (265) <max>? P
      (266) <period>? P
   (267) <job-list>?
      (268) <job>*
         (269) <class-name>
         (270) <name>?
         (271) <description>?
         (272) <begin-time>?
         (273) <end-time>?
         (274) <interval>?
            (275) <millisecond>
            (276) <minutely>
            (277) <hourly>
            (278) <daily>
         (279) <count>? P
(280) <enable-jnlp>?
(281) <enable-webadmin>?
(282) <system-logging>?
   (283) <level>? P
   (284) <use-parent-handlers>? P
   (285) <filter-class>?
   (286) <handler>?
      (287) <console-handler>
         (288) <name>
         (289) <level>? P
         (290) <encoding>?
         (291) <filter-class>?
      (292) <file-handler>
         (293) <name>
         (294) <level>? P
         (295) <encoding>?
         (296) <filter-class>?
         (297) <file-name>?
```

```
            (298) <valid-day>
            (299) <valid-hour>
            (300) <buffer-size>? P
            (301) <append>? P
         (302) <smtp-handler>
            (303) <name>
            (304) <level>? P
            (305) <encoding>?
            (306) <filter-class>?
            (307) <smtp-host-address>
            (308) <from-address>
            (309) <to-address>
            (310) <cc-address>?
            (311) <bcc-address>?
            (312) <send-for-all-messages>? P
         (313) <socket-handler>
            (314) <name>
            (315) <level>? P
            (316) <encoding>?
            (317) <filter-class>?
            (318) <address>
            (319) <port>
         (320) <user-handler>
            (321) <handler-class>
            (322) <name>
            (323) <level>? P
            (324) <encoding>?
            (325) <filter-class>?
            (326) <handler-property>?
               (327) <property>*
                  (328) <key>
                  (329) <value>
            (330) <formatter-class>? P
            (331) <formatter-property>?
               (332) <property>*
                  (333) <key>
                  (334) <value>
   (335) <session-server>?
      (336) <resolution>? P
```

```
   (337) <backlog-size>? P
   (338) <session-manager>*
      (339) <name>
      (340) <multi-session>?
      (341) <passivation-to>?
      (342) <removal-to>?
      (343) <file-db-path>?
      (344) <file-db-name>?
      (345) <min-hole>? P
      (346) <packing-rate>? P
      (347) <check-to>? P
      (348) <backup-name>?
      (349) <backup-trigger>? P
      (350) <operation-to>? P
(351) <session-router-config>?
   (352) <connect-timeout>? P
   (353) <read-timeout>? P
   (354) <backup-trigger>? P
   (355) <check-to>? P
   (356) <check-level>? P
   (357) <default-file-db>?
      (358) <startup-clear-to>? P
      (359) <path>?
      (360) <passivation-to>? P
      (361) <min-hole>? P
      (362) <packing-rate>? P
   (363) <session-router>+
      (364) <servlet-engine-name>
      (365) <file-db>?
         (366) <startup-clear-to>? P
         (367) <path>?
         (368) <passivation-to>? P
         (369) <min-hole>? P
         (370) <packing-rate>? P
      (371) <backup-session-router>?
         (372) <node-name>
         (373) <servlet-engine-name>
(374) <jmx-manager>?
   (375) <jmx-connector>?
```

```
            (376) <jmxmp-connector>?
               (377) <jmxmp-connector-port>? P
            (378) <rmi-connector>?
               (379) <rmi-connector-port>?
               (380) <export-name>?
               (381) <ref-export-name>?
         (382) <html-adaptor-port>?
         (383) <snmp-adaptor>?
            (384) <snmp-adaptor-port>
            (385) <snmp-version>? P
            (386) <snmp-max-packet-size>? P
            (387) <snmp-security>? P
            (388) <trap-demon>*
               (389) <ip-address>
               (390) <port>
            (391) <pooling>?
               (392) <min>? P
               (393) <max>? P
               (394) <period>? P
         (395) <mlet-url>*
(396) <naming-server>?
   (397) <server>?
      (398) <use-nio>? P
      (399) <export-cos-naming>? P
      (400) <backlog-size>? P
      (401) <resolution>? P
      (402) <buffer-size>? P
      (403) <pooling>?
         (404) <min>? P
         (405) <max>? P
         (406) <period>? P
   (407) <local>?
      (408) <resolution>? P
      (409) <buffer-size>? P
      (410) <pooling>?
         (411) <min>? P
         (412) <max>? P
         (413) <period>? P
(414) <security-manager>?
```

```
   (415) <domain-name>*
(416) <resource>?
   (417) <data-source>?
      (418) <database>*
         (419) <vendor>
         (420) <export-name>
         (421) <data-source-class-name>
         (422) <data-source-type>
         (423) <database-name>?
         (424) <data-source-name>?
         (425) <service-name>?
         (426) <description>?
         (427) <network-protocol>?
         (428) <encryption>? P
         (429) <port-number>?
         (430) <server-name>?
         (431) <user>?
         (432) <password>?
         (433) <driver-type>?
         (434) <property>*
            (435) <name>
            (436) <type>
            (437) <value>
         (438) <connection-pool>?
            (439) <pooling>?
               (440) <min>? P
               (441) <max>? P
               (442) <step>? P
               (443) <period>? P
            (444) <wait-free-connection>?
               (445) <enable-wait>? P
               (446) <wait-time>? P
            (447) <operation-to>? P
            (448) <delegation-datasource>?
            (449) <max-use-count>? P
            (450) <delegation-dba>?
            (451) <dba-timeout>? P
            (452) <check-query>?
            (453) <stmt-caching-size>? P
```

```
            (454) <stmt-fetch-size>? P
      (455) <cluster-ds>*
         (456) <export-name>
         (457) <is-pre-conn>? P
         (458) <backup-data-source>+
   (459) <mail-source>?
      (460) <mail-entry>*
         (461) <export-name>
         (462) <mail-property>*
            (463) <name>
            (464) <value>
   (465) <url-source>?
      (466) <url-entry>*
         (467) <export-name>
         (468) <url>
   (469) <external-source>?
      (470) <jms-source>*
         (471) <vendor>
         (472) <factory-class-name>
         (473) <resource-type>
         (474) <export-name>
         (475) <queue>?
         (476) <queueManager>?
         (477) <topic>?
         (478) <property>*
            (479) <name>
            (480) <type>
            (481) <value>
      (482) <tmax>*
         (483) <export-name>
         (484) <webt-datasource-type>? P
         (485) <webt-logging>?
            (486) <file-name>? P
            (487) <level>? P
            (488) <valid-day>? P
            (489) <buffer-size>? P
         (490) <host-name>
         (491) <port>
         (492) <backup-host-name>?
```

```
(493) <backup-port>?
(494) <fdb-file>?
(495) <default-charset>?
(496) <enable-pipe>? P
(497) <enable-extended-header>? P
(498) <inbuf-size>? P
(499) <outbuf-size>? P
(500) <max-idle-timeout>? P
(501) <service-timeout>?
(502) <transaction-timeout>?
(503) <transaction-block-timeout>?
(504) <security>?
   (505) <user-name>?
   (506) <user-password>?
   (507) <domain-name>?
   (508) <domain-password>?
(509) <tmax-connection-pool>?
   (510) <pooling>?
      (511) <min>? P
      (512) <max>? P
      (513) <step>? P
      (514) <period>? P
   (515) <wait-free-connection>?
      (516) <enable-wait>? P
      (517) <wait-time>? P
   (518) <tmax-property>*
      (519) <name>
      (520) <value>
(521) <tmax-cluster>?
   (522) <export-name>
   (523) <enable-load-balance>? P
   (524) <is-pre-conn>? P
   (525) <tmax-delegation-source>*
(526) <jaxr-source>?
   (527) <jaxr-entry>*
      (528) <export-name>
      (529) <connection-factory-class-name>?
      (530) <query-manager-URL>?
      (531) <lifeCycle-manager-URL>?
```

```
            (532) <authentication-method>?
            (533) <jaxr-property>*
               (534) <name>
               (535) <value>
(536) <applications>*
   (537) <absolute-path>?
   (538) <absolute-ejb-jar>?
   (539) <absolute-jeus-ejb-dd>?
   (540) <auto-deploy>?
      (541) <auto-deploy-check-interval>? P
   (542) <deployment-target>?
      (543) <all-targets>
         (544) <context-group-name>?
      (545) <target>
         (546) <node-name>
         (547) <engine-container-name>
         (548) <context-group-name>?
   (549) <application>+
      (550) <path>?
      (551) <deployment-target>?
         (552) <all-targets>
            (553) <context-group-name>?
         (554) <target>
            (555) <node-name>
            (556) <engine-container-name>
            (557) <context-group-name>?
      (558) <deployment-type>
      (559) <auto-deploy>?
         (560) <auto-deploy-check-interval>? P
      (561) <classloading>?
      (562) <class-ftp-unit>? P
      (563) <security-domain-name>?
      (564) <role-permission>*
         (565) <principal>+
         (566) <role>
         (567) <actions>?
         (568) <classname>?
         (569) <excluded>?
         (570) <unchecked>?
```

```
(571) <java-security-permission>?
   (572) <description>?
   (573) <security-permission-spec>
(574) <keep-generated>?
(575) <fast-deploy>?
(576) <client-component>*
   (577) <uri>?
   (578) <deployment-target>?
      (579) <all-targets>
         (580) <context-group-name>?
      (581) <target>
         (582) <node-name>
         (583) <engine-container-name>
         (584) <context-group-name>?
(585) <connector-component>*
   (586) <uri>?
(587) <ejb-component>*
   (588) <uri>?
   (589) <deployment-target>?
      (590) <all-targets>
         (591) <context-group-name>?
      (592) <target>
         (593) <node-name>
         (594) <engine-container-name>
         (595) <context-group-name>?
   (596) <client-view-path>?
   (597) <keep-generated>?
   (598) <ejb-jar>?
   (599) <jeus-ejb-dd>?
   (600) <java-security-permission>?
      (601) <description>?
      (602) <security-permission-spec>
(603) <web-component>*
   (604) <uri>?
   (605) <deployment-target>?
      (606) <all-targets>
         (607) <context-group-name>?
      (608) <target>
         (609) <node-name>
```

```
                (610) <engine-container-name>
                (611) <context-group-name>?
            (612) <keep-generated>?
            (613) <jeus-web-dd>?
            (614) <java-security-permission>?
               (615) <description>?
               (616) <security-permission-spec>
(617) <application>*
   (618) <path>?
   (619) <deployment-target>?
      (620) <all-targets>
         (621) <context-group-name>?
      (622) <target>
         (623) <node-name>
         (624) <engine-container-name>
         (625) <context-group-name>?
   (626) <deployment-type>
   (627) <auto-deploy>?
      (628) <auto-deploy-check-interval>? P
   (629) <classloading>?
   (630) <class-ftp-unit>? P
   (631) <security-domain-name>?
   (632) <role-permission>*
      (633) <principal>+
      (634) <role>
      (635) <actions>?
      (636) <classname>?
      (637) <excluded>?
      (638) <unchecked>?
   (639) <java-security-permission>?
      (640) <description>?
      (641) <security-permission-spec>
   (642) <keep-generated>?
   (643) <fast-deploy>?
   (644) <client-component>*
      (645) <uri>?
      (646) <deployment-target>?
         (647) <all-targets>
            (648) <context-group-name>?
```

```
      (649) <target>
         (650) <node-name>
         (651) <engine-container-name>
         (652) <context-group-name>?
(653) <connector-component>*
   (654) <uri>?
(655) <ejb-component>*
   (656) <uri>?
   (657) <deployment-target>?
      (658) <all-targets>
         (659) <context-group-name>?
      (660) <target>
         (661) <node-name>
         (662) <engine-container-name>
         (663) <context-group-name>?
   (664) <client-view-path>?
   (665) <keep-generated>?
   (666) <ejb-jar>?
   (667) <jeus-ejb-dd>?
   (668) <java-security-permission>?
      (669) <description>?
      (670) <security-permission-spec>
(671) <web-component>*
   (672) <uri>?
   (673) <deployment-target>?
      (674) <all-targets>
         (675) <context-group-name>?
      (676) <target>
         (677) <node-name>
         (678) <engine-container-name>
         (679) <context-group-name>?
   (680) <keep-generated>?
   (681) <jeus-web-dd>?
   (682) <java-security-permission>?
      (683) <description>?
      (684) <security-permission-spec>
```

J.3 Element Reference

(1) **<jeus-system>**

Description JEUS 설정 파일의 최상위 element. JEUS 시스템에 속하는 모든 노드의 설정을 이 element 아래에 기록한다.

Child Elements
(2)node+
(396)naming-server?
(414)security-manager?
(416)resource?
(535)applications*
(616)application*

(2) <jeus-system> **<node>**

Description JEUS 'node'는 기본적으로 JEUS 가 실행 중인 서버의 머신 이름이다. JEUS 클러스터링 환경에서는 각 JEUS 노드마다 하나의 <node> element 를 가진다.각 서버마다 JEUSMain.xml 을 가지고 있으며, 이 파일을 통해서 다른 서버의 정보를 얻는다.

Child Elements
(3)name
(4)listener?
(12)backup-node?
(13)engine-container*
(258)class-ftp?
(259)security-switch?
(260)sequential-start?
(261)scheduler?
(280)enable-jnlp?
(281)enable-webadmin?
(282)system-logging?
(335)session-server?
(351)session-router-config?
(374)jmx-manager?

(3) <jeus-system> <node> **<name>**

Description 노드의 이름.

Value Description 실제 JEUS 가 동작하는 머신의 이름. 유닉스에서는 "uname -n"으로 알아낼 수 있으며, Windows 에서는 [시스템 등록 정보]-[네트워크 식별]에서 나오는 컴퓨터 이름이다.

Value Type token

(4) <jeus-system> <node> **<listener>**

Description 이 노드의 JEUS 시스템이 사용하는 socket listener 및 socket connection request 처리에 필요한 각종 속성을 지정한다.

Child Elements
(5)backlog?
(6)ssl?
(8)thread-pool?

(5) <jeus-system> <node> <listener> **<backlog>**

Description $JEUS_BASEPORT 로 지정한 JEUS system listener port 에 대한 backlog 값을 지정한다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Default Value 128

(6) <jeus-system> <node> <listener> **<ssl>**

Description Jeus Security system 에 관련된 SSL 속성을 지정한다. 이 element 를 설정하면 Jeus Security system 을 사용하는 모든 socket connection 에 SSL 이 적용된다.

Child Elements (7)port?

(7) <jeus-system> <node> <listener> <ssl> **<port>**

Description Jeus Security system 이 SSL connection 에 사용할 listen port 를 지정한다.

Value Description 기본값은 $JEUS_BASEPORT + 1 이다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

(8) <jeus-system> <node> <listener> **<thread-pool>**

Description Jeus system listener port ($JEUS_BASEPORT) 에 요청되는 socket

connection 처리를 위한 thread pool 을 설정한다.

Child Elements

(9)min?
(10)max?
(11)period?

(9) <jeus-system> <node> <listener> <thread-pool> **<min>**

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

(10) <jeus-system> <node> <listener> <thread-pool> **<max>**

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

(11) <jeus-system> <node> <listener> <thread-pool> **<period>**

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

(12) <jeus-system> <node> **<backup-node>**

Description 이 node 의 JEUS 가 이상 작동으로 down 될 때 backup node 로 작동될 서버를 지정한다. 이 backup node 에서 이 node 의 여러가지 service 를 backup 해준다.

Value Description backup node 로 작동할 노드의 이름.

Value Type token

(13) <jeus-system> <node> **<engine-container>**

Description 여러 개의 JEUS 의 Engine 을 관리하는 container 이다. application 은 이 container 단위로 deploy 되고 실행된다.

Child Elements
(14)name
(15)command-option?
(16)engine-command*
(72)enable-interop?
(74)sequential-start?
(75)user-class-path?
(76)tm-config?
(91)scheduler?
(110)user-logging?
(163)system-logging?
(216)invocation-manager-action?
(217)jmx-manager?
(239)auto-deploy?
(243)use-MEJB?
(244)lifecycle-invocation*
(253)application-path*
(254)res-ref?

(14) <jeus-system> <node> <engine-container> **<name>**

Description Engine Container 의 이름.

Value Description 적절한 이름을 입력한다. 여기서 입력한 이름과 노드의 이름을 조합해서 실제 Engine Container 이름이 생성된다.예)
“johan_mycontainer”에서 “mycontainer”가 여기서 입력한 이름이다.

Value Type token

Defined Value default
이 이름이 사용되면, 이 Engine Container 가 JEUSManager 와 동일한 JVM 에서 동작하게 된다. 다른 이름에 대해서는 새로운 JVM process 로 Engine Container 가 운영된다.

(15) <jeus-system> <node> <engine-container> **<command-option>**

Description Engine Container 의 JVM 에 들어가는 JVM 옵션을 적어준다.

Value Type token

Example

```
<command-option>-Xms64m -Xmx128m</command-option>
```

(16) <jeus-system> <node> <engine-container> **<engine-command>**

Description

Engine 은 J2EE 어플리케이션이 작동하기 위한 환경을 제공한다. J2EE 스펙에서의 Container 에 대응하는 기능이다.모든 Engine 은 Engine Container 가 부팅될 때 실행되며, Engine Container 하나에서는 각 Engine 타입 당 하나씩 지원한다. Engine 타입으로는 servlet, ejb, jms, ws 가 있다. 각각 EJB Engine, Servlet Engine, JMS Engine, WebServer Engine 을 나타낸다.

Child Elements

(17)type
(18)name
(19)system-logging?

(17) <jeus-system> <node> <engine-container> <engine-command> **<type>**

Description

Engine Container 에 포함되는 Engine 타입.

Value Type

engine-typeType

Defined Value

ejb
EJB engine

servlet
Servlet/Web engine

jms
JMS engine

ws
JEUS Web Server engine (JEUS 의 기본 Web server).

(18) <jeus-system> <node> <engine-container> <engine-command> **<name>**

Description

Engine 의 이름.

Value Description

적절한 이름을 입력한다. 이 이름은 <node_name>_<engine_type><engine_name> 형태로 변경되어서 사용된다.예) “johan_servlet_Engine1”은 “Engine1”이라는 EJB Engine 의 이름이다.

Value Type token

(19) <jeus-system> <node> <engine-container> <engine-command> **<system-logging>**

Description 이 Engine 의 error log 를 기록하는 logger 에 대한 설정이다.

Child Elements
(20)level?
(21)use-parent-handlers?
(22)filter-class?
(23)handler?

(20) <jeus-system> <node> <engine-container> <engine-command> <system-logging> **<level>**

Description logging 의 level 을 설정한다. 각 level 의 의미는 J2SE 의 logging API 의 Level class 를 참고하기 바란다.

Value Type loggingLevelType

Default Value INFO

Defined Value
FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(21) <jeus-system> <node> <engine-container> <engine-command> <system-logging> **<use-parent-handlers>**

Description parent logger 의 handler 들을 이 logger 에서도 사용할지를 결정한다.

Value Type boolean

Default Value false

(22) <jeus-system> <node> <engine-container> <engine-command> <system-logging> **<filter-class>**

Description logger 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-class>com.tmax.logging.filter.MyFilter</filter-class>
```

(23) <jeus-system> <node> <engine-container> <engine-command> <system-logging> **<handler>**

Description logger 에서 사용할 handler 를 설정한다.

Child Elements (24)console-handler

(29)file-handler
(39)smtp-handler
(50)socket-handler
(57)user-handler

```
(24) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <console-handler>
```

Description logging 을 화면에 남기고자 하는 경우에 사용하는 handler 이다.

Child Elements
(25)name
(26)level?
(27)encoding?
(28)filter-class?

```
(25) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <console-handler> <name>
```

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

```
(26) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <console-handler> <level>
```

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value
FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(27) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <console-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(28) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <console-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을

설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(29) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> **<file-handler>**

Description logging 을 file 로 출력하고자 하는 경우에 사용하는 handler 이다.

Child Elements
(30)name
(31)level?
(32)encoding?
(33)filter-class?
(34)file-name?
(35)valid-day
(36)valid-hour
(37)buffer-size?
(38)append?

(30) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <file-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(31) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <file-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value FATAL

SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(32) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <file-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(33) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <file-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(34) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <file-handler> <file-name>
```

Description 이 handler 가 사용할 file name 을 설정한다. 만약 user 가 이 설정을 하지 않으면 각 logger 의 default file name 이 사용된다. 각각의 default file name 은 JEUS Server 메뉴얼을 참고하기 바란다.

Value Type token

Example
```
<file-name>C:\logs\mylog.log</file-name>
```

```
(35) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <file-handler> <valid-day>
```

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 날짜 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜가 자동으로 붙게 된다.

Value Description day

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Example
```
<valid-day>1</valid-day>
```

```
(36) <jeus-system> <node> <engine-container> <engine-command> <system-
```

```
logging> <handler> <file-handler> <valid-hour>
```

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 시간 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜와 시간이 자동으로 붙게 된다.

Value Description 시간을 나타내며 24 의 약수 + n*24 (n 은 0 이상의 정수)의 값을 가져야 한다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Example

```
<valid-hour>3</valid-hour>
```

```
(37) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <file-handler> <buffer-size>
```

Description 이 handler 가 file 에 출력할때 사용하는 buffer 의 크기를 지정한다.

Value Description byte 단위이다. [Performance Recommandation]: 이 값이 클수록 file 에 출력되는 message 는 지연되어 출력되지만 logging 성능은 좋아진다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 1024

```
(38) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <file-handler> <append>
```

Description 이 handler 가 사용하는 file 이 이미 존재하는 경우 file 뒤에 덧붙여 쓸지를 결정한다. false 로 설정되어 있다면 기존의 file 은 제거된다.

Value Type boolean

Default Value true

```
(39) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler>
```

Description logging 을 email 로 보내고자 하는 경우에 사용하는 handler 이다.

[Performance Recommandation]: logging message 하나가 하나의 email 로 전송되므로 적절한 filter 없이 사용하는 것은 엄청난 양의 email 을 발생시켜 아주 위험하므로 주의를 요한다.

Child Elements

(40)name
(41)level?
(42)encoding?
(43)filter-class?
(44)smtp-host-address
(45)from-address
(46)to-address
(47)cc-address?
(48)bcc-address?
(49)send-for-all-messages?

(40) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<name>**

Description

handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type

token

Example

```
<name>handler1</name>
```

(41) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<level>**

Description

이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type

loggingLevelType

Default Value

FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(42) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <smtp-handler> <encoding>
```

Description 이 handler 가 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(43) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <smtp-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-class>com.tmax.logging.filter.MyFilter</filter-class>
```

(44) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<smtp-host-address>**

Description email 을 보낼 smtp server 의 주소를 지정한다.

Value Type token

(45) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<from-address>**

Description email 을 보내는 사람의 address 를 지정한다.

Value Type token

(46) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<to-address>**

Description email 을 받는 사람의 address 를 지정한다.

Value Type token

(47) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<cc-address>**

Description email 을 참조로 받는 사람의 address 를 지정한다.

Value Type token

(48) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<bcc-address>**

Description email 을 숨은 참조로 받는 사람의 address 를 지정한다.

Value Type token

(49) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <smtp-handler> **<send-for-all-messages>**

Description 이 handler 가 등록한 logger 의 log() method 를 통해 들어온 message 들이 이 handler 로 들어왔을때 이를 email 로 보낼 대상으로 여길지를 설정한다. 만약 false 로 설정되어 있으면 logger 의 특별한 send() method 로 호출된 message 들만 email 로 전송된다. 즉, 처음부터 email 로 보낼 의도로 지정된 message 들만 email 로 전송된다.

Value Type boolean

Default Value false

(50) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> **<socket-handler>**

Description logging 을 지정된 socket 으로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나당 Socket 으로 전송이 되므로 적절한 filter 없이 사용하는 것은 성능 저하를 가져온다.

Child Elements
(51)name
(52)level?
(53)encoding?
(54)filter-class?
(55)address
(56)port

(51) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <socket-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(52) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <socket-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(53) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <socket-handler> **<encoding>**

Description 이 handler 가 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(54) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <socket-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(55) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <socket-handler> **<address>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 IP address 를 설정한다.

Value Type token

(56) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <socket-handler> **<port>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 port 를 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(57) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> **<user-handler>**

Description User 가 J2SE logging API 에 따라 만든 handler 를 사용할 경우의 설정이다.

Child Elements
```
(58)handler-class
(59)name
```

(60)level?
(61)encoding?
(62)filter-class?
(63)handler-property?
(67)formatter-class?
(68)formatter-property?

```
(58) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <user-handler> <handler-class>
```

Description user 가 만든 handler 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Handler 를 상속받고 jeus.util.logging.JeusHandler 를 구현해야 한다.

Value Type token

Example
```
<handler-
class>com.tmax.logging.handler.MyHandler</handler-
class>
```

```
(59) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <user-handler> <name>
```

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

```
(60) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <user-handler> <level>
```

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(61) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <user-handler> **<encoding>**

Description 이 handler 가 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(62) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <user-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(63) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <user-handler> **<handler-property>**

Description handler 가 생성될 때 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusHandler.setProperty() method 를 통해 handler 로 전달된다.

Child Elements (64)property*

(64) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <user-handler> <handler-property> **<property>**

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements
(65)key
(66)value

(65) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <user-handler> <handler-property> <property> **<key>**

Description property 의 key 값이다.

Value Type token

(66) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <user-handler> <handler-property> <property> **<value>**

Description property 의 value 값이다.

Value Type token

(67) <jeus-system> <node> <engine-container> <engine-command> <system-logging> <handler> <user-handler> **<formatter-class>**

Description 이 handler 가 사용할 formatter 의 fully qualified class name 을 설정한다.이

클래스는 java.util.logging.Formatter 를 상속받고
jeus.util.logging.JeusFormatter 를 구현해야 한다.

Value Type token

Default Value jeus.util.logging.SimpleFormatter

Example
```
<formatter-
class>com.tmax.logging.handler.MyHandler</formatter-
class>
```

```
(68) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <user-handler> <formatter-property>
```

Description handler 가 생성될 때 만들어진 formatter 에게 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusFormatter.setProperty() method 를 통해 formatter 로 전달된다.

Child Elements (69)property*

```
(69) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <user-handler> <formatter-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements (70)key
(71)value

```
(70) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <user-handler> <formatter-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

```
(71) <jeus-system> <node> <engine-container> <engine-command> <system-
logging> <handler> <user-handler> <formatter-property> <property>
<value>
```

Description property 의 value 값이다.

Value Type token

```
(72) <jeus-system> <node> <engine-container> <enable-interop>
```

Description 이 설정이 존재한다면 이 엔진 컨테이너는 IIOP 프로토콜을

사용하는 모든 ORB 와 상호동작 할 수 있다. 이 경우 두 개의 특별한 리스너 인터페이스가 엔진 컨테이너에 Deploy 된다(CSI 리스너와 OTS 리스너). 이 인터페이스는 보안(예 : "principal")과 트랜잭션(예: GTID)을 포함하는 IIOP 메시지 헤더를 인식하고 조정한다. 만약 이 기능을 활성화 하려면 엔진 컨테이너의 JVM 에 반드시 보안 파라미터가 명시 되어야만 한다.

Child Elements (73)use-OTS

(73) <jeus-system> <node> <engine-container> <enable-interop> **<use-OTS>**

Description OTS 를 지원하는 ORB 로 interoperability 를 사용할지를 결정한다. false 라면 OTS 를 지원하지 않는 방식으로 동작한다.

Value Type boolean

Default Value false

(74) <jeus-system> <node> <engine-container> **<sequential-start>**

Description Engine 을 하나씩 부팅할지를 정한다. 기본적으로 모든 Engine 은 동시에 부팅되기 때문에 문제가 발생할 수 있다. 부팅 순서는 JEUSMain.xml 에서 지정된 순서에 따라서 부팅된다.

Value Type boolean

Default Value false

Performance Recommendation 이 기능이 사용되면 Engine Container 의 실행시간은 길어지지만, 좀 더 안정적으로 부팅된다.

(75) <jeus-system> <node> <engine-container> **<user-class-path>**

Description Engine Container 를 실행하는 JVM 의 classpath 에 패스를 추가한다. 이 설정은 default Engine Container 에는 적용되지 않는다.

Value Description “;”로 구분되는 classpath(유닉스의 경우 “:”

Value Type token

Example

```
<user-class-path>c:\mylib\classes;c:\mylib\lib\mylib.jar</user-class-path>
```

(76) <jeus-system> <node> <engine-container> **<tm-config>**

Description 트랜잭션 매니저(TM)는 Global Transaction 을 시작하고 종료한다. 트랜잭션을 종료할 때 TM 은 RM(리소스 매니저)과 통신을 하면서 commit 인지 rollback 인지 결정한다. 이렇게 함으로써 TM 은 Global Transaction 의 원자성을 보장하게 된다.그러나 실제 상황에서는 많은 예외적인 상황이 발생하는데, 이에 대한 대응책으로 다양한 타임아웃 메커니즘을 제공한다. 하위 element 에서는 TM 의 타임아웃 등을 설정한다.

Child Elements

(77)use-nio?
(78)pooling?
(82)active-timeout?
(83)prepare-timeout?
(84)prepared-timeout?
(85)commit-timeout?
(86)recovery-timeout?
(87)uncompleted-timeout?
(88)capacity?
(89)heuristic-rollback?
(90)resolution?

(77) <jeus-system> <node> <engine-container> <tm-config> **<use-nio>**

Description TM 사이의 통신을 Nonblocking I/O 를 이용할지의 여부를 결정한다.

Value Type boolean

Default Value true

Performance Recommendation 많은 수의 Engine Container 및 client 를 사용할 경우에는 Nonblocking I/O 가 더 좋은 효율을 보인다. transaction 에 참여하는 JVM process 의 수가 적다면 Blocking I/O 가 더 효율적일수 있다.

(78) <jeus-system> <node> <engine-container> <tm-config> **<pooling>**

Description TM pool 은 트랜잭션을 처리하는 쓰레드로 구성되어 있다. 아래 element 는 이 쓰레드를 관리하는 pool 을 설정한다.

Child Elements

(79)min?
(80)max?
(81)period?

(79) <jeus-system> <node> <engine-container> <tm-config> <pooling> **<min>**

Description	pooling 되는 객체의 최소값을 지정한다.
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	2

```
(80) <jeus-system> <node> <engine-container> <tm-config> <pooling>
<max>
```

Description	pooling 되는 객체의 최대값을 지정한다.
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	30

```
(81) <jeus-system> <node> <engine-container> <tm-config> <pooling>
<period>
```

Description	pooling 되는 객체를 정리하는 시간을 지정한다.
Value Type	long
Default Value	3600000
Performance Recommendation	이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(82) <jeus-system> <node> <engine-container> <tm-config> <active-
timeout>
```

Description	글로벌 트랜잭션이 시작되면 이 시간 안에 commit 이 실행되어야 한다. 그렇지 않으면 트랜잭션 매니저가 rollback 시켜버린다.
Value Description	milliseconds 단위의 시간 간격.
Value Type	nonNegativeLongType
Value Type Description	0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.
Default Value	600000

```
(83) <jeus-system> <node> <engine-container> <tm-config> <prepare-
```

timeout>

Description transaction 이 commit 될때 Root Coordinator 는 이 시간 내에 Sub Coordinator 와 리소스 매니저로부터 'prepare' 신호를 받아야 한다. 만약 받지를 못하면 Root Coordinator 는 글로벌 트랜잭션을 rollback 시킨다.

Value Description milliseconds 단위의 시간 간격.

Value Type long

Default Value 120000

(84) <jeus-system> <node> <engine-container> <tm-config> **<prepared-timeout>**

Description transaction 이 commit 되어 Root Coordinator 로부터 prepare message 를 받으면 Sub Coordinator 는 prepare 에 대한 응답을 Root Coordinator 로 보내고 global decision 을 기다린다. Sub Coordinator 는 자신의 Root Coordinator 로부터 여기에 설정된 시간 안에 global decision 을 받아야 한다. 만약 이 시간 내에 받질 못하면, Root Coordinator 로 다시 'prepare'에 대한 응답 메시지를 보낸다. 그래도 여전히 시간 내에 global decision 이 오지 않는다면, <heuristic-rollback>의 값이 true 일 때 Global Transaction 을 rollback 시켜버린다. <heuristic-rollback>이 false 이면, Root Coordinator 로 메시지를 보내고 global decision 을 기다리기를 계속 한다.

Value Description milliseconds 단위의 시간 간격.

Value Type long

Default Value 60000

(85) <jeus-system> <node> <engine-container> <tm-config> **<commit-timeout>**

Description Root Coordinator 는 Sub Coordinator 와 리소스 매니저에거 commit message 를 보낸 후 이 시간 이내에 'commit'이나 'rollback' 에 대한 결과를 받아야 한다. 만약 결과가 오지 않으면, Root Coordinator 는 글로벌 트랜잭션을 'Uncompleted List'에 기록해서, 트랜잭션이 완전히 끝나지 않았음을 남겨둔다.

Value Description milliseconds 단위의 시간 간격.

Value Type long

Default Value 240000

(86) <jeus-system> <node> <engine-container> <tm-config> **<recovery-timeout>**

Description 이 값은 트랜잭션 복구 시에 사용된다. 트랜잭션 매니저는 트랜잭션 복구를 위해서 복구될 트랜잭션 정보를 가져오려고 한다. 만약 다른 트랜잭션 매니저에서 이 시간 내에 복구 정보를 알려주지 않으면, <heuristic-rollback>이 true 일 때, 해당 트랜잭션을 rollback 시킨다. 값이 false 이면 트랜잭션 복구를 시스템 관리자에게 남겨두고 더 이상 진행하지 않는다.

Value Description milliseconds 단위의 시간 간격.

Value Type long

Default Value 120000

(87) <jeus-system> <node> <engine-container> <tm-config> **<uncompleted-timeout>**

Description 트랜잭션 매니저는 전체 트랜잭션 처리를 완료하기 위해, 실패한 글로벌 트랜잭션의 목록을 보관한다. 완료되지 못한 글로벌 트랜잭션의 정보는 복구 처리시에 사용되므로, 이 타임 아웃 시간까지 보관된다. 그러므로 이 시간이 너무 짧으면 복구 정보가 빨리 지워지게 되고, 트랜잭션 매니저가 해당 글로벌 트랜잭션의 무결성을 복구할 수 없게 된다. 그 결과 글로벌 트랜잭션 복구를 위해서, 시스템 관리자가 많은 작업을 직접 처리해야만 한다.

Value Description milliseconds 단위의 시간 간격.

Value Type long

Default Value 86400000

(88) <jeus-system> <node> <engine-container> <tm-config> **<capacity>**

Description 이 값을 사용해서 JEUS 트랜잭션 매니저는 내부 구조를 최적화시킨다. 트랜잭션 매니저가 동시에 처리하는 글로벌 트랜잭션의 개수를 고려해서 값을 정한다.

Value Type int

Default Value 10000

Performance Recommendation 이 값은 트랜잭션 매니저가 처리하는 동시 트랜잭션의 수에 가깝게 세팅해야 한다.

(89) <jeus-system> <node> <engine-container> <tm-config> **<heuristic-rollback>**

Description 어떤 이유로 Global Transaction 이 완료되지 못하는 경우, 이 값이 true 이면 트랜잭션 매니저가 rollback 시켜 버리고, false 라면 시스템 관리자가 처리하도록 Uncompleted List 에 남겨둔다.

Value Type boolean

Default Value false

(90) <jeus-system> <node> <engine-container> <tm-config> **<resolution>**

Description 타임 아웃 메커니즘의 기본 시간 간격. 이 값을 작게 설정하면 타임 아웃이 세밀하게 작동한다. 값이 너무 크면 타임 아웃이 너무 느슨하게 작동한다

Value Type long

Default Value 60000

Performance Recommendation 만약 이 값이 타임 아웃값보다 크면, 타임 아웃이 제대로 작동하지 않는다. 그러므로 이 값은 타임 아웃 메커니즘이 정상적으로 작동할 수 있을 정도로 작게 설정한다.

(91) <jeus-system> <node> <engine-container> **<scheduler>**

Description Scheduler Server 를 부팅할지 결정한다. 이 element 를 설정하면 scheduler 를 사용하게 된다.

Child Elements
(92)default-lookup-name?
(93)thread-pool?
(97)job-list?

(92) <jeus-system> <node> <engine-container> <scheduler> **<default-lookup-name>**

Description jeus_service/Scheduler 로 lookup 할때 return 할 Scheduler 객체가

존재하는 node 나 container 의 이름을 지정한다. 지정이 되어 있지 않으면 lookup 하는 곳에 존재하는 Scheduler 객체가 사용된다.

Value Type token

(93) <jeus-system> <node> <engine-container> <scheduler> **<thread-pool>**

Description scheduler 에서 multi-thread 로 job 을 실행할때 사용하는 thread pool 을 설정한다.

Child Elements
(94)min?
(95)max?
(96)period?

(94) <jeus-system> <node> <engine-container> <scheduler> <thread-pool> **<min>**

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

(95) <jeus-system> <node> <engine-container> <scheduler> <thread-pool> **<max>**

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

(96) <jeus-system> <node> <engine-container> <scheduler> <thread-pool> **<period>**

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로

지정한다.

```
(97) <jeus-system> <node> <engine-container> <scheduler> <job-list>
```

Description scheduler 에 등록할 job list 을 지정한다.

Child Elements (98)job*

```
(98) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job>
```

Description scheduler 에 등록할 하나의 job 을 지정한다.

Child Elements
(99)class-name
(100)name?
(101)description?
(102)begin-time?
(103)end-time?
(104)interval?
(109)count?

```
(99) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <class-name>
```

Description job 을 수행하는 class 의 fully qualified name 이다.

Value Type token

```
(100) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <name>
```

Description 이 job 의 이름을 지정한다.

Value Type token

```
(101) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <description>
```

Description 이 job 의 설명을 적을 수 있다.

Value Type string

```
(102) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <begin-time>
```

Description 이 job 의 시작 시간을 지정한다.

Value Type dateTime

```
(103) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <end-time>
```

Description 이 job 의 종료 시간을 지정한다.

Value Type dateTime

```
(104) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval>
```

Description 이 job 이 수행되는 주기를 지정한다.

Child Elements
(105)millisecond
(106)minutely
(107)hourly
(108)daily

```
(105) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <millisecond>
```

Description 주기를 millisecond 단위로 지정한다.

Value Type long

```
(106) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <minutely>
```

Description 주기를 분 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

```
(107) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <hourly>
```

Description 주기를 시간 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

```
(108) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <daily>
```

Description 주기를 날짜 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(109) <jeus-system> <node> <engine-container> <scheduler> <job-list> <job> **<count>**

Description 이 job 이 수행되는 횟수를 지정한다.

Value Type long

Default Value -1

Defined Value -1
수행되는 횟수를 제한하지 않는다.

(110) <jeus-system> <node> <engine-container> **<user-logging>**

Description 이 element 는 jeus.util.UserLogger 클래스를 사용해서 생성한 로그를 어떻게 처리할 것인지 세팅한다.

Child Elements
(111)level?
(112)use-parent-handlers?
(113)filter-class?
(114)handler?

(111) <jeus-system> <node> <engine-container> <user-logging> **<level>**

Description logging 의 level 을 설정한다. 각 level 의 의미는 J2SE 의 logging API 의 Level class 를 참고하기 바란다.

Value Type loggingLevelType

Default Value INFO

Defined Value FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(112) <jeus-system> <node> <engine-container> <user-logging> **<use-parent-handlers>**

Description	parent logger 의 handler 들을 이 logger 에서도 사용할지를 결정한다.
Value Type	boolean
Default Value	false

(113) <jeus-system> <node> <engine-container> <user-logging> **<filter-**

class>

Description logger 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(114) <jeus-system> <node> <engine-container> <user-logging> **<handler>**

Description logger 에서 사용할 handler 를 설정한다.

Child Elements
(115)console-handler
(120)file-handler
(130)smtp-handler
(141)socket-handler
(148)user-handler

(115) <jeus-system> <node> <engine-container> <user-logging> <handler> **<console-handler>**

Description logging 을 화면에 남기고자 하는 경우에 사용하는 handler 이다.

Child Elements
(116)name
(117)level?
(118)encoding?
(119)filter-class?

(116) <jeus-system> <node> <engine-container> <user-logging> <handler> <console-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(117) <jeus-system> <node> <engine-container> <user-logging> <handler> <console-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(118) <jeus-system> <node> <engine-container> <user-logging> <handler> <console-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(119) <jeus-system> <node> <engine-container> <user-logging> <handler> <console-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(120) <jeus-system> <node> <engine-container> <user-logging> <handler> **<file-handler>**

Description logging 을 file 로 출력하고자 하는 경우에 사용하는 handler 이다.

Child Elements
(121)name
(122)level?
(123)encoding?
(124)filter-class?
(125)file-name?
(126)valid-day
(127)valid-hour
(128)buffer-size?
(129)append?

(121) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example

```
<name>handler1</name>
```

(122) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(123) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(124) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-class>com.tmax.logging.filter.MyFilter</filter-class>
```

(125) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<file-name>**

Description 이 handler 가 사용할 file name 을 설정한다. 만약 user 가 이 설정을 하지 않으면 각 logger 의 default file name 이 사용된다. 각각의 default file name 은 JEUS Server 메뉴얼을 참고하기 바란다.

Value Type token

Example
```
<file-name>C:\logs\mylog.log</file-name>
```

(126) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<valid-day>**

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 날짜 단위로

file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜가 자동으로 붙게 된다.

Value Description day

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Example
```
<valid-day>1</valid-day>
```

(127) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<valid-hour>**

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 시간 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜와 시간이 자동으로 붙게 된다.

Value Description 시간을 나타내며 24 의 약수 + n*24 (n 은 0 이상의 정수)의 값을 가져야 한다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Example
```
<valid-hour>3</valid-hour>
```

(128) <jeus-system> <node> <engine-container> <user-logging> <handler> <file-handler> **<buffer-size>**

Description 이 handler 가 file 에 출력할때 사용하는 buffer 의 크기를 지정한다.

Value Description byte 단위이다. [Performance Recommandation]: 이 값이 클수록 file 에 출력되는 message 는 지연되어 출력되지만 logging 성능은 좋아진다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 1024

(129) <jeus-system> <node> <engine-container> <user-logging> <handler>

<file-handler> **<append>**

Description 이 handler 가 사용하는 file 이 이미 존재하는 경우 file 뒤에 덧붙여 쓸지를 결정한다. false 로 설정되어 있다면 기존의 file 은 제거된다.

Value Type boolean

Default Value true

(130) <jeus-system> <node> <engine-container> <user-logging> <handler> **<smtp-handler>**

Description logging 을 email 로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나가 하나의 email 로 전송되므로 적절한 filter 없이 사용하는 것은 엄청난 양의 email 을 발생시켜 아주 위험하므로 주의를 요한다.

Child Elements

(131) name
(132) level?
(133) encoding?
(134) filter-class?
(135) smtp-host-address
(136) from-address
(137) to-address
(138) cc-address?
(139) bcc-address?
(140) send-for-all-messages?

(131) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example

```
<name>handler1</name>
```

(132) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(133) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(134) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(135) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<smtp-host-address>**

Description email 을 보낼 smtp server 의 주소를 지정한다.

Value Type token

(136) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<from-address>**

Description email 을 보내는 사람의 address 를 지정한다.

Value Type token

(137) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<to-address>**

Description email 을 받는 사람의 address 를 지정한다.

Value Type token

(138) <jeus-system> <node> <engine-container> <user-logging> <handler> <smtp-handler> **<cc-address>**

Description email 을 참조로 받는 사람의 address 를 지정한다.

Value Type token

```
(139) <jeus-system> <node> <engine-container> <user-logging> <handler>
<smtp-handler> <bcc-address>
```

Description email 을 숨은 참조로 받는 사람의 address 를 지정한다.

Value Type token

```
(140) <jeus-system> <node> <engine-container> <user-logging> <handler>
<smtp-handler> <send-for-all-messages>
```

Description 이 handler 가 등록한 logger 의 log() method 를 통해 들어온 message 들이 이 handler 로 들어왔을때 이를 email 로 보낼 대상으로 여길지를 설정한다. 만약 false 로 설정되어 있으면 logger 의 특별한 send() method 로 호출된 message 들만 email 로 전송된다. 즉, 처음부터 email 로 보낼 의도로 지정된 message 들만 email 로 전송된다.

Value Type boolean

Default Value false

```
(141) <jeus-system> <node> <engine-container> <user-logging> <handler>
<socket-handler>
```

Description logging 을 지정된 socket 으로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나당 Socket 으로 전송이 되므로 적절한 filter 없이 사용하는 것은 성능 저하를 가져온다.

Child Elements

```
(142)name
(143)level?
(144)encoding?
(145)filter-class?
(146)address
(147)port
```

```
(142) <jeus-system> <node> <engine-container> <user-logging> <handler>
<socket-handler> <name>
```

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(143) <jeus-system> <node> <engine-container> <user-logging> <handler> <socket-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(144) <jeus-system> <node> <engine-container> <user-logging> <handler> <socket-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(145) <jeus-system> <node> <engine-container> <user-logging> <handler> <socket-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(146) <jeus-system> <node> <engine-container> <user-logging> <handler> <socket-handler> **<address>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 IP address 를 설정한다.

Value Type token

(147) <jeus-system> <node> <engine-container> <user-logging> <handler> <socket-handler> **<port>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 port 를 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(148) <jeus-system> <node> <engine-container> <user-logging> <handler> **<user-handler>**

Description User 가 J2SE logging API 에 따라 만든 handler 를 사용할 경우의 설정이다.

Child Elements
(149)handler-class
(150)name
(151)level?
(152)encoding?
(153)filter-class?
(154)handler-property?
(158)formatter-class?
(159)formatter-property?

(149) <jeus-system> <node> <engine-container> <user-logging> <handler> <user-handler> **<handler-class>**

Description user 가 만든 handler 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Handler 를 상속받고 jeus.util.logging.JeusHandler 를 구현해야 한다.

Value Type token

Example
```
<handler-
class>com.tmax.logging.handler.MyHandler</handler-
class>
```

(150) <jeus-system> <node> <engine-container> <user-logging> <handler> <user-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(151) <jeus-system> <node> <engine-container> <user-logging> <handler> <user-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이

이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(152) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(153) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(154) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <handler-property>
```

Description handler 가 생성될 때 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusHandler.setProperty() method 를 통해 handler 로 전달된다.

Child Elements (155)property*

```
(155) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <handler-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements

```
(156)key
(157)value
```

```
(156) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <handler-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

```
(157) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <handler-property> <property> <value>
```

Description property 의 value 값이다.

Value Type token

```
(158) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <formatter-class>
```

Description 이 handler 가 사용할 formatter 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Formatter 를 상속받고 jeus.util.logging.JeusFormatter 를 구현해야 한다.

Value Type token

Default Value jeus.util.logging.SimpleFormatter

Example

```
<formatter-
class>com.tmax.logging.handler.MyHandler</formatter-
class>
```

```
(159) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <formatter-property>
```

Description handler 가 생성될 때 만들어진 formatter 에게 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusFormatter.setProperty() method 를 통해 formatter 로 전달된다.

Child Elements (160)property*

```
(160) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <formatter-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements

```
(161)key
(162)value
```

```
(161) <jeus-system> <node> <engine-container> <user-logging> <handler>
<user-handler> <formatter-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

(162) <jeus-system> <node> <engine-container> <user-logging> <handler> <user-handler> <formatter-property> <property> **<value>**

Description property 의 value 값이다.

Value Type token

(163) <jeus-system> <node> <engine-container> **<system-logging>**

Description 이 Engine Container 에서 사용할 logger 를 설정한다. 이 logger 에는 Engine Container 에서 발생하는 error message 들이 출력된다.

Child Elements
(164)level?
(165)use-parent-handlers?
(166)filter-class?
(167)handler?

(164) <jeus-system> <node> <engine-container> <system-logging> **<level>**

Description logging 의 level 을 설정한다. 각 level 의 의미는 J2SE 의 logging API 의 Level class 를 참고하기 바란다.

Value Type loggingLevelType

Default Value INFO

Defined Value
FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(165) <jeus-system> <node> <engine-container> <system-logging> **<use-parent-handlers>**

Description	parent logger 의 handler 들을 이 logger 에서도 사용할지를 결정한다.
Value Type	boolean
Default Value	false

(166) <jeus-system> <node> <engine-container> <system-logging> **<filter-class>**

Description	logger 에 지정할 filer class 의 fully qualified class name 을 설정한다.
Value Type	token
Example	`<filter-class>com.tmax.logging.filter.MyFilter</filter-class>`

```
(167) <jeus-system> <node> <engine-container> <system-logging>
<handler>
```

Description logger 에서 사용할 handler 를 설정한다.

Child Elements
(168)console-handler
(173)file-handler
(183)smtp-handler
(194)socket-handler
(201)user-handler

```
(168) <jeus-system> <node> <engine-container> <system-logging>
<handler> <console-handler>
```

Description logging 을 화면에 남기고자 하는 경우에 사용하는 handler 이다.

Child Elements
(169)name
(170)level?
(171)encoding?
(172)filter-class?

```
(169) <jeus-system> <node> <engine-container> <system-logging>
<handler> <console-handler> <name>
```

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

```
(170) <jeus-system> <node> <engine-container> <system-logging>
<handler> <console-handler> <level>
```

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(171) <jeus-system> <node> <engine-container> <system-logging>
<handler> <console-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(172) <jeus-system> <node> <engine-container> <system-logging>
<handler> <console-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(173) <jeus-system> <node> <engine-container> <system-logging>
<handler> <file-handler>
```

Description logging 을 file 로 출력하고자 하는 경우에 사용하는 handler 이다.

Child Elements
```
(174)name
(175)level?
(176)encoding?
(177)filter-class?
(178)file-name?
(179)valid-day
(180)valid-hour
(181)buffer-size?
(182)append?
```

```
(174) <jeus-system> <node> <engine-container> <system-logging>
<handler> <file-handler> <name>
```

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

```
(175) <jeus-system> <node> <engine-container> <system-logging>
<handler> <file-handler> <level>
```

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(176) <jeus-system> <node> <engine-container> <system-logging>
<handler> <file-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(177) <jeus-system> <node> <engine-container> <system-logging>
<handler> <file-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(178) <jeus-system> <node> <engine-container> <system-logging>
<handler> <file-handler> <file-name>
```

Description 이 handler 가 사용할 file name 을 설정한다. 만약 user 가 이 설정을 하지 않으면 각 logger 의 default file name 이 사용된다. 각각의 default file name 은 JEUS Server 메뉴얼을 참고하기 바란다.

Value Type token

Example
```
<file-name>C:\logs\mylog.log</file-name>
```

```
(179) <jeus-system> <node> <engine-container> <system-logging>
<handler> <file-handler> <valid-day>
```

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 날짜 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜가 자동으로 붙게 된다.

Value Description day

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지

않은게 된다.즉, off 된다.

Example `<valid-day>1</valid-day>`

(180) <jeus-system> <node> <engine-container> <system-logging> <handler> <file-handler> **<valid-hour>**

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 시간 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜와 시간이 자동으로 붙게 된다.

Value Description 시간을 나타내며 24 의 약수 + n*24 (n 은 0 이상의 정수)의 값을 가져야 한다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Example `<valid-hour>3</valid-hour>`

(181) <jeus-system> <node> <engine-container> <system-logging> <handler> <file-handler> **<buffer-size>**

Description 이 handler 가 file 에 출력할때 사용하는 buffer 의 크기를 지정한다.

Value Description byte 단위이다. [Performance Recommandation]: 이 값이 클수록 file 에 출력되는 message 는 지연되어 출력되지만 logging 성능은 좋아진다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 1024

(182) <jeus-system> <node> <engine-container> <system-logging> <handler> <file-handler> **<append>**

Description 이 handler 가 사용하는 file 이 이미 존재하는 경우 file 뒤에 덧붙여 쓸지를 결정한다. false 로 설정되어 있다면 기존의 file 은 제거된다.

Value Type boolean

Default Value true

```
(183) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler>
```

Description logging 을 email 로 보내고자 하는 경우에 사용하는 handler 이다.
[Performance Recommandation]: logging message 하나가 하나의 email 로 전송되므로 적절한 filter 없이 사용하는 것은 엄청난 양의 email 을 발생시켜 아주 위험하므로 주의를 요한다.

Child Elements
```
(184)name
(185)level?
(186)encoding?
(187)filter-class?
(188)smtp-host-address
(189)from-address
(190)to-address
(191)cc-address?
(192)bcc-address?
(193)send-for-all-messages?
```

```
(184) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <name>
```

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

```
(185) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <level>
```

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(186) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(187) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(188) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <smtp-host-address>
```

Description email 을 보낼 smtp server 의 주소를 지정한다.

Value Type token

```
(189) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <from-address>
```

Description email 을 보내는 사람의 address 를 지정한다.

Value Type token

```
(190) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <to-address>
```

Description email 을 받는 사람의 address 를 지정한다.

Value Type token

```
(191) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <cc-address>
```

Description email 을 참조로 받는 사람의 address 를 지정한다.

Value Type token

```
(192) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <bcc-address>
```

Description email 을 숨은 참조로 받는 사람의 address 를 지정한다.

Value Type token

```
(193) <jeus-system> <node> <engine-container> <system-logging>
<handler> <smtp-handler> <send-for-all-messages>
```

Description 이 handler 가 등록한 logger 의 log() method 를 통해 들어온 message 들이 이 handler 로 들어왔을때 이를 email 로 보낼 대상으로 여길지를 설정한다. 만약 false 로 설정되어 있으면 logger 의 특별한 send() method 로 호출된 message 들만 email 로 전송된다. 즉, 처음부터 email 로 보낼 의도로 지정된 message 들만 email 로 전송된다.

Value Type boolean

Default Value false

```
(194) <jeus-system> <node> <engine-container> <system-logging>
<handler> <socket-handler>
```

Description logging 을 지정된 socket 으로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나당 Socket 으로 전송이 되므로 적절한 filter 없이 사용하는 것은 성능 저하를 가져온다.

Child Elements

```
(195)name
(196)level?
(197)encoding?
(198)filter-class?
(199)address
(200)port
```

```
(195) <jeus-system> <node> <engine-container> <system-logging>
<handler> <socket-handler> <name>
```

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example

```
<name>handler1</name>
```

```
(196) <jeus-system> <node> <engine-container> <system-logging>
<handler> <socket-handler> <level>
```

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이

이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(197) <jeus-system> <node> <engine-container> <system-logging>
<handler> <socket-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(198) <jeus-system> <node> <engine-container> <system-logging>
<handler> <socket-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(199) <jeus-system> <node> <engine-container> <system-logging>
<handler> <socket-handler> <address>
```

Description 이 handler 가 생성될때 message 들을 보낼 곳의 IP address 를 설정한다.

Value Type token

```
(200) <jeus-system> <node> <engine-container> <system-logging>
<handler> <socket-handler> <port>
```

Description 이 handler 가 생성될때 message 들을 보낼 곳의 port 를 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

```
(201) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler>
```

Description User 가 J2SE logging API 에 따라 만든 handler 를 사용할 경우의 설정이다.

Child Elements

(202)handler-class
(203)name
(204)level?
(205)encoding?
(206)filter-class?
(207)handler-property?
(211)formatter-class?
(212)formatter-property?

(202) <jeus-system> <node> <engine-container> <system-logging> <handler> <user-handler> **<handler-class>**

Description

user 가 만든 handler 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Handler 를 상속받고 jeus.util.logging.JeusHandler 를 구현해야 한다.

Value Type

token

Example

```
<handler-
class>com.tmax.logging.handler.MyHandler</handler-
class>
```

(203) <jeus-system> <node> <engine-container> <system-logging> <handler> <user-handler> **<name>**

Description

handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type

token

Example

```
<name>handler1</name>
```

(204) <jeus-system> <node> <engine-container> <system-logging> <handler> <user-handler> **<level>**

Description

이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type

loggingLevelType

Default Value

FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로

compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(205) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(206) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(207) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <handler-property>
```

Description handler 가 생성될 때 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusHandler.setProperty() method 를 통해 handler 로 전달된다.

Child Elements (208)property*

```
(208) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <handler-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements

```
(209)key
(210)value
```

```
(209) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <handler-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

```
(210) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <handler-property> <property> <value>
```

Description property 의 value 값이다.

Value Type token

```
(211) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <formatter-class>
```

Description 이 handler 가 사용할 formatter 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Formatter 를 상속받고 jeus.util.logging.JeusFormatter 를 구현해야 한다.

Value Type token

Default Value jeus.util.logging.SimpleFormatter

Example
```
<formatter-
class>com.tmax.logging.handler.MyHandler</formatter-
class>
```

```
(212) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <formatter-property>
```

Description handler 가 생성될 때 만들어진 formatter 에게 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusFormatter.setProperty() method 를 통해 formatter 로 전달된다.

Child Elements
```
(213)property*
```

```
(213) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <formatter-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements
```
(214)key
(215)value
```

```
(214) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <formatter-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

```
(215) <jeus-system> <node> <engine-container> <system-logging>
<handler> <user-handler> <formatter-property> <property> <value>
```

Description property 의 value 값이다.

Value Type token

(216) <jeus-system> <node> <engine-container> **<invocation-manager-action>**

Description invocation manager 는 Engine Container 내의 Stateless 메소드(Servlet/JSP, Stateless EJB 와 MDB)에서 사용한 리소스를 추적해서 보고한다.

Value Type invocation-manager-actionType

Defined Value

NoAction
아무 작업을 하지 않는다.

Warning
리소스를 사용했지만 메소드 종료시에 close 하지 않았을 때, Container 로그에 이에 대한 로그를 기록한다.

AutoClose
리소스를 사용했지만 메소드 종료시에 close 하지 않았을 때, 사용된 리소스를 자동적으로 close 시켜준다.

Performance Recommendation "NoAction"이 성능면에서 가장 좋다. 다른 옵션은 어플리케이션 코드에 버그가 많아서 신뢰성이 떨어질 때 사용한다.

(217) <jeus-system> <node> <engine-container> **<jmx-manager>**

Description JMX Manager element 는 이 Engine Container 의 JMX 관련 모든 설정을 담고 있다.

Child Elements
(218)jmx-connector?
(225)html-adaptor-port?
(226)snmp-adaptor?
(238)mlet-url*

(218) <jeus-system> <node> <engine-container> <jmx-manager> **<jmx-connector>**

Description 다른 process 에서 이 Engine Container 의 JMX 를 access 할 때 사용하는 JMX Connector 를 설정한다. 기본적으로는 JEUSMP Connector 를 사용한다.

Child Elements
(219)jmxmp-connector?
(221)rmi-connector?

```
(219) <jeus-system> <node> <engine-container> <jmx-manager> <jmx-
connector> <jmxmp-connector>
```

Description JMX Connector 로 JMXMP Connector 를 사용한다.

Child Elements (220)jmxmp-connector-port?

```
(220) <jeus-system> <node> <engine-container> <jmx-manager> <jmx-
connector> <jmxmp-connector> <jmxmp-connector-port>
```

Description 다른 process 에서 이 Engine Container 의 JMX 를 access 할 때 사용하는 JEUSMP Connector 의 listen port 를 지정한다. 만약 이 값이 0 이거나 지정하지 않으면 JEUSMP Connector 가 사용하는 listen port 를 따로 만들지 않고 jeus 의 공통 port 를 사용한다. 만약 JEUS 의 JMX RemoteAPI 를 사용하지 않고 다른 Runtime 에서 JMXMP protocol 로 접근하고자 한다면 이를 0 이 아닌 다른 값으로 지정해야 한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 0

Defined Value 0
jeus 의 공통 port 를 사용한다.

```
(221) <jeus-system> <node> <engine-container> <jmx-manager> <jmx-
connector> <rmi-connector>
```

Description JMX Connector 로 RMI Connector 를 사용한다. 만약 jmxmp-connector 와 같이 설정되어 있는 경우에는 JEUS system 내부적으로는 jmxmp-connector 를 기본적으로 사용하게 된다. 또한 이 경우에는 rmi-connector 의 ref-export-name 이 별도로 설정되어 있어야 한다. 이 이름이 JEUS 에서 기본적으로 사용하는 이름과 같거나 설정이 되어있지 않다면 exception 이 발생한다.

Child Elements (222)rmi-connector-port?
(223)export-name?
(224)ref-export-name?

```
(222) <jeus-system> <node> <engine-container> <jmx-manager> <jmx-
connector> <rmi-connector> <rmi-connector-port>
```

Description 다른 process 에서 이 Engine Container 의 JMX 를 access 할 때 사용하는 RMI Connector 의 port 를 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Defined Value 0
random 하게 port 를 지정한다.

```
(223) <jeus-system> <node> <engine-container> <jmx-manager> <jmx-
connector> <rmi-connector> <export-name>
```

Description JEUS 의 기본 export name 대신에 다른 export name 을 사용할때 설정한다. 이 export name 은 JMXServiceURL 의 URL path 에 들어가게 된다.

Value Type token

```
(224) <jeus-system> <node> <engine-container> <jmx-manager> <jmx-
connector> <rmi-connector> <ref-export-name>
```

Description 이 connector 를 얻을수 있는 jndi name 을 JEUS 의 기본 jndi name 이 아닌 다른 name 으로 지정하고자 할 때 사용한다. 이 export name 으로 lookup 하면 JMXConnector 객체를 얻을수 있다.

Value Type token

```
(225) <jeus-system> <node> <engine-container> <jmx-manager> <html-
adaptor-port>
```

Description JMX 의 adapter 중 하나인 HTML adapter 의 port 를 지정한다. 여기에 지정된 값으로 Web Browser 가 접속하게 된다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

```
(226) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-
adaptor>
```

Description JMX 의 adapter 중 하나인 SNMP adapter 를 설정한다.

Child Elements

(227)snmp-adaptor-port
(228)snmp-version?
(229)snmp-max-packet-size?
(230)snmp-security?
(231)trap-demon*
(234)pooling?

(227) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> **<snmp-adaptor-port>**

Description SNMP 어댑터의 리스너 포트

Value Type snmp-adaptor-portType

(228) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> **<snmp-version>**

Description SNMP 버전을 지정하며 1, 2 또는 3 을 지정할 수 있다

Value Type snmp-versionType

Default Value 3

Defined Value 1

2

3

(229) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> **<snmp-max-packet-size>**

Description SNMP 패킷에 대한 최대값을 설정하며 최소 256 바이트부터 설정 할 수 있다.

Value Type snmp-max-packet-sizeType

Default Value 4096

(230) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> **<snmp-security>**

Description 보안을 적용시킬 것 인지를 설정한다. 보안은 SNMP 버전 3 에서만 지정이 가능 하다.

Value Type boolean

Default Value false

(231) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> **<trap-demon>**

Description 장애 상황 발생시 TRAP 메세지를 보낼 서버를 설정한다. 여러 개 설정이 가능하며 설정된 모든 ip, address 로 메세지를 보낸다.

Child Elements
(232)ip-address
(233)port

(232) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> <trap-demon> **<ip-address>**

Description Demon 의 IP address

Value Description a valid IP address

Value Type token

Example
```
<host-name>111.111.111.1</host-name>
```

(233) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> <trap-demon> **<port>**

Description Demon 의 port

Value Description a port number

Value Type int

Example
```
<port>8888</port>
```

(234) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> **<pooling>**

Description SNMP Server 에서 요청을 처리하는 쓰레드로 구성되어 있다. 아래 element 는 이 쓰레드를 관리하는 pool 을 설정한다.

Child Elements
(235)min?
(236)max?
(237)period?

(235) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-adaptor> <pooling> **<min>**

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

```
(236) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-
adaptor> <pooling> <max>
```

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

```
(237) <jeus-system> <node> <engine-container> <jmx-manager> <snmp-
adaptor> <pooling> <period>
```

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(238) <jeus-system> <node> <engine-container> <jmx-manager> <mlet-url>
```

Description 이 Engine Container 의 MBeanServer 에 등록할 MLet 의 URL 을 지정한다.

Value Type token

```
(239) <jeus-system> <node> <engine-container> <auto-deploy>
```

Description 이 Engine Container 의 auto-deploy 에 대한 설정이다.

Child Elements

```
(240)enable-auto-deploy?
(241)auto-deploy-path?
(242)auto-deploy-check-interval?
```

(240) <jeus-system> <node> <engine-container> <auto-deploy> **<enable-auto-deploy>**

Description auto-deploy 를 이 engine container 에 대해 설정할지를 지정한다.

Value Type boolean

Default Value true

(241) <jeus-system> <node> <engine-container> <auto-deploy> **<auto-deploy-path>**

Description Engine Container 가 auto-deploy 로 인식할 directory path 를 설정한다. 설정하지 않으면 jeus.deployhome System property 의 값이나 webhome/deploy_home 이 사용된다.

Value Type token

(242) <jeus-system> <node> <engine-container> <auto-deploy> **<auto-deploy-check-interval>**

Description application 이 변경되었는지 check 하는 주기를 설정할 수 있다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 10000

Performance Recommendation 너무 자주 check 하게 되면 성능저하가 생길 수 있으므로 필요한 간격만큼만 설정하도록 한다.

(243) <jeus-system> <node> <engine-container> **<use-MEJB>**

Description J2EE Management Spec 에서 제시하는 MEJB 를 사용할 것인지를 설정한다. 사용하지 않는다면 MEJB 를 deploy 하지 않는다.

Value Type boolean

Default Value false

(244) <jeus-system> <node> <engine-container> **<lifecycle-invocation>**

Description Engine Container 의 각종 lifecycle event 에 호출할 method 를 지정할 수 있다.

Child Elements (245)class-name
(246)invocation+

```
(245) <jeus-system> <node> <engine-container> <lifecycle-invocation>
<class-name>
```

Description lifecycle event 의 callback method 가 존재하는 fully qualified class name 을 지정한다.

Value Type token

Example com.tmax.event.EngineContainerHandler

```
(246) <jeus-system> <node> <engine-container> <lifecycle-invocation>
<invocation>
```

Description 이 class 내의 invocation 세부 정보를 설정한다.

Child Elements (247)invocation-method
(251)invocation-argument*
(252)invocation-type

```
(247) <jeus-system> <node> <engine-container> <lifecycle-invocation>
<invocation> <invocation-method>
```

Description 이 invocation 에 사용될 method 를 지정한다.

Child Elements (248)method-name
(249)method-params?

```
(248) <jeus-system> <node> <engine-container> <lifecycle-invocation>
<invocation> <invocation-method> <method-name>
```

Description method 의 이름을 지정한다.

Value Type token

Example foo

```
(249) <jeus-system> <node> <engine-container> <lifecycle-invocation>
<invocation> <invocation-method> <method-params>
```

Description method 의 parameter 들을 순서대로 지정한다.

Child Elements (250)method-param*

```
(250) <jeus-system> <node> <engine-container> <lifecycle-invocation>
```

```
<invocation> <invocation-method> <method-params> <method-param>
```

Description method 의 parameter 의 fully qualified class name 을 지정한다.

Value Type token

Example `java.lang.String`

```
(251) <jeus-system> <node> <engine-container> <lifecycle-invocation>
<invocation> <invocation-argument>
```

Description method 를 호출할 때 사용하는 argument 를 지정한다.

Value Type token

```
(252) <jeus-system> <node> <engine-container> <lifecycle-invocation>
<invocation> <invocation-type>
```

Description 이 method 가 호출되는 시점을 지정한다.

Value Type invocation-typeType

Defined Value

BOOT
Engine Container 가 시작되고 engine 들이 띄워지기 전의 시점이다.

BEFORE_DEPLOY
Engine Container 가 시작되고 이 Engine Container 에게 지정된 application 이 deploy 되기 전의 시점이다.

AFTER_DEPLOY
Engine Container 가 시작되고 이 Engine Container 에게 지정된 application 이 deploy 된 후의 시점이다.

BEFORE_UNDEPLOY
Engine Container 가 down 명령을 받았을 때 이 Engine Container 에서 운영중인 application 들을 undeploy 하기 전의 시점이다.

AFTER_UNDEPLOY
Engine Container 가 down 명령을 받았고 이 Engine Container 에서 운영중인 application 들을 undeploy 한 후의 시점이다.

```
(253) <jeus-system> <node> <engine-container> <application-path>
```

Description application archive file 들을 넣을 디렉토리를 지정한다. 상대경로인 경우에는 JEUS_HOME path 에서의 상대경로이다. application 은 이 element 의 순서대로 검색된다. [Default Value]: jeus.apphome system property 로 설정한 path 거나 webhome/app_home

Value Type token

(254) <jeus-system> <node> <engine-container> **<res-ref>**

Description 이 Engine Container 의 JNDI 에 등록할 resource reference 들이다.

Child Elements (255)jndi-info+

(255) <jeus-system> <node> <engine-container> <res-ref> **<jndi-info>**

Description 이 Engine Container 의 JNDI 에 등록할 각 resource reference 의 export name 과 reference name 을 지정한다.

Child Elements (256)ref-name
(257)export-name

(256) <jeus-system> <node> <engine-container> <res-ref> <jndi-info> **<ref-name>**

Description 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수 있다.

Value Description 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

Value Type token

Example <ref-name>ejb/AccountEJB</ref-name>

(257) <jeus-system> <node> <engine-container> <res-ref> <jndi-info> **<export-name>**

Description JEUS DD 에 정의된 실제 JNDI 이름.

Value Type token

Example <export-name>ACCEJB</export-name>

(258) <jeus-system> <node> **<class-ftp>**

Description class FTP 는 EJB 스텁을 클라이언트로 FTP 를 사용해서 전송한다.

class FTP 를 사용하지 않으면 직접 해당 파일을 복사해줘야 한다.

Value Type boolean

(259) <jeus-system> <node> **<security-switch>**

Value Type boolean

Default Value true

(260) <jeus-system> <node> **<sequential-start>**

Description 'true'이면 Engine Container 가 순서대로 부팅된다. 기본적으로 Engine Container 는 동시에 부팅되도록 되어있는데, 어떤 경우에는 문제가 발생할 수 있다.

Value Type boolean

(261) <jeus-system> <node> **<scheduler>**

Description JEUS Manager 에서 사용할 scheduler 의 설정이다.

Child Elements
(262)default-lookup-name?
(263)thread-pool?
(267)job-list?

(262) <jeus-system> <node> <scheduler> **<default-lookup-name>**

Description jeus_service/Scheduler 로 lookup 할때 return 할 Scheduler 객체가 존재하는 node 나 container 의 이름을 지정한다. 지정이 되어 있지 않으면 lookup 하는 곳에 존재하는 Scheduler 객체가 사용된다.

Value Type token

(263) <jeus-system> <node> <scheduler> **<thread-pool>**

Description scheduler 에서 multi-thread 로 job 을 실행할때 사용하는 thread pool 을 설정한다.

Child Elements
(264)min?
(265)max?
(266)period?

(264) <jeus-system> <node> <scheduler> <thread-pool> **<min>**

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

(265) <jeus-system> <node> <scheduler> <thread-pool> **<max>**

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

(266) <jeus-system> <node> <scheduler> <thread-pool> **<period>**

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

(267) <jeus-system> <node> <scheduler> **<job-list>**

Description scheduler 에 등록할 job list 을 지정한다.

Child Elements (268)job*

(268) <jeus-system> <node> <scheduler> <job-list> **<job>**

Description scheduler 에 등록할 하나의 job 을 지정한다.

Child Elements
(269)class-name
(270)name?
(271)description?
(272)begin-time?
(273)end-time?
(274)interval?
(279)count?

(269) <jeus-system> <node> <scheduler> <job-list> <job> **<class-name>**

Description job 을 수행하는 class 의 fully qualified name 이다.

Value Type token

(270) <jeus-system> <node> <scheduler> <job-list> <job> **<name>**

Description 이 job 의 이름을 지정한다.

Value Type token

(271) <jeus-system> <node> <scheduler> <job-list> <job> **<description>**

Description 이 job 의 설명을 적을 수 있다.

Value Type string

(272) <jeus-system> <node> <scheduler> <job-list> <job> **<begin-time>**

Description 이 job 의 시작 시간을 지정한다.

Value Type dateTime

(273) <jeus-system> <node> <scheduler> <job-list> <job> **<end-time>**

Description 이 job 의 종료 시간을 지정한다.

Value Type dateTime

(274) <jeus-system> <node> <scheduler> <job-list> <job> **<interval>**

Description 이 job 이 수행되는 주기를 지정한다.

Child Elements
(275)millisecond
(276)minutely
(277)hourly
(278)daily

(275) <jeus-system> <node> <scheduler> <job-list> <job> <interval> **<millisecond>**

Description 주기를 millisecond 단위로 지정한다.

Value Type long

(276) <jeus-system> <node> <scheduler> <job-list> <job> <interval> **<minutely>**

Description 주기를 분 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(277) <jeus-system> <node> <scheduler> <job-list> <job> <interval> **<hourly>**

Description 주기를 시간 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(278) <jeus-system> <node> <scheduler> <job-list> <job> <interval> **<daily>**

Description 주기를 날짜 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(279) <jeus-system> <node> <scheduler> <job-list> <job> **<count>**

Description 이 job 이 수행되는 횟수를 지정한다.

Value Type long

Default Value -1

Defined Value -1
수행되는 횟수를 제한하지 않는다.

(280) <jeus-system> <node> **<enable-jnlp>**

Description JNLP Server 를 사용할지 지정한다.

Value Type boolean

(281) <jeus-system> <node> **<enable-webadmin>**

Description WebAdmin 을 사용할지 지정한다. WebAdmin 은 JEUS 를 관리하는 웹 기반의 툴이다.

Value Type boolean

(282) <jeus-system> <node> **<system-logging>**

Description JEUS Manager 에서 사용할 logger 의 설정이다.

Child Elements
(283)level?
(284)use-parent-handlers?
(285)filter-class?
(286)handler?

(283) <jeus-system> <node> <system-logging> **<level>**

Description logging 의 level 을 설정한다. 각 level 의 의미는 J2SE 의 logging API 의 Level class 를 참고하기 바란다.

Value Type loggingLevelType

Default Value INFO

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(284) <jeus-system> <node> <system-logging> **<use-parent-handlers>**

Description parent logger 의 handler 들을 이 logger 에서도 사용할지를 결정한다.

Value Type boolean

Default Value false

(285) <jeus-system> <node> <system-logging> **<filter-class>**

Description logger 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(286) <jeus-system> <node> <system-logging> **<handler>**

Description logger 에서 사용할 handler 를 설정한다.

Child Elements

(287)console-handler
(292)file-handler
(302)smtp-handler
(313)socket-handler
(320)user-handler

(287) <jeus-system> <node> <system-logging> <handler> **<console-handler>**

Description logging 을 화면에 남기고자 하는 경우에 사용하는 handler 이다.

Child Elements

(288)name
(289)level?
(290)encoding?
(291)filter-class?

(288) <jeus-system> <node> <system-logging> <handler> <console-handler> **<name>**

Description

handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type

token

Example

```
<name>handler1</name>
```

(289) <jeus-system> <node> <system-logging> <handler> <console-handler> **<level>**

Description

이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type

loggingLevelType

Default Value

FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(290) <jeus-system> <node> <system-logging> <handler> <console-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(291) <jeus-system> <node> <system-logging> <handler> <console-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-class>com.tmax.logging.filter.MyFilter</filter-class>
```

(292) <jeus-system> <node> <system-logging> <handler> **<file-handler>**

Description logging 을 file 로 출력하고자 하는 경우에 사용하는 handler 이다.

Child Elements
(293)name
(294)level?
(295)encoding?
(296)filter-class?
(297)file-name?
(298)valid-day
(299)valid-hour
(300)buffer-size?
(301)append?

(293) <jeus-system> <node> <system-logging> <handler> <file-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(294) <jeus-system> <node> <system-logging> <handler> <file-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value
FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로

compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(295) <jeus-system> <node> <system-logging> <handler> <file-handler>
<encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(296) <jeus-system> <node> <system-logging> <handler> <file-handler>
<filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(297) <jeus-system> <node> <system-logging> <handler> <file-handler> **<file-name>**

Description 이 handler 가 사용할 file name 을 설정한다. 만약 user 가 이 설정을 하지 않으면 각 logger 의 default file name 이 사용된다. 각각의 default file name 은 JEUS Server 메뉴얼을 참고하기 바란다.

Value Type token

Example
```
<file-name>C:\logs\mylog.log</file-name>
```

(298) <jeus-system> <node> <system-logging> <handler> <file-handler> **<valid-day>**

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 날짜 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜가 자동으로 붙게 된다.

Value Description day

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Example
```
<valid-day>1</valid-day>
```

(299) <jeus-system> <node> <system-logging> <handler> <file-handler> **<valid-hour>**

Description 이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 시간 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜와 시간이 자동으로 붙게 된다.

Value Description 시간을 나타내며 24 의 약수 + n*24 (n 은 0 이상의 정수)의 값을 가져야 한다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Example
```
<valid-hour>3</valid-hour>
```

(300) <jeus-system> <node> <system-logging> <handler> <file-handler> **<buffer-size>**

Description 이 handler 가 file 에 출력할때 사용하는 buffer 의 크기를 지정한다.

Value Description byte 단위이다. [Performance Recommandation]: 이 값이 클수록 file 에 출력되는 message 는 지연되어 출력되지만 logging 성능은 좋아진다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 1024

(301) <jeus-system> <node> <system-logging> <handler> <file-handler> **<append>**

Description 이 handler 가 사용하는 file 이 이미 존재하는 경우 file 뒤에 덧붙여 쓸지를 결정한다. false 로 설정되어 있다면 기존의 file 은 제거된다.

Value Type boolean

Default Value true

(302) <jeus-system> <node> <system-logging> <handler> **<smtp-handler>**

Description logging 을 email 로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나가 하나의 email 로 전송되므로 적절한 filter 없이 사용하는 것은 엄청난 양의 email 을 발생시켜 아주 위험하므로 주의를 요한다.

Child Elements
```
(303)name
(304)level?
(305)encoding?
(306)filter-class?
(307)smtp-host-address
(308)from-address
(309)to-address
```

(310)cc-address?
(311)bcc-address?
(312)send-for-all-messages?

(303) <jeus-system> <node> <system-logging> <handler> <smtp-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example `<name>handler1</name>`

(304) <jeus-system> <node> <system-logging> <handler> <smtp-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(305) <jeus-system> <node> <system-logging> <handler> <smtp-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(306) <jeus-system> <node> <system-logging> <handler> <smtp-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-class>com.tmax.logging.filter.MyFilter</filter-class>
```

(307) <jeus-system> <node> <system-logging> <handler> <smtp-handler> **<smtp-host-address>**

Description email 을 보낼 smtp server 의 주소를 지정한다.

Value Type token

```
(308) <jeus-system> <node> <system-logging> <handler> <smtp-handler>
<from-address>
```

Description email 을 보내는 사람의 address 를 지정한다.

Value Type token

```
(309) <jeus-system> <node> <system-logging> <handler> <smtp-handler>
<to-address>
```

Description email 을 받는 사람의 address 를 지정한다.

Value Type token

```
(310) <jeus-system> <node> <system-logging> <handler> <smtp-handler>
<cc-address>
```

Description email 을 참조로 받는 사람의 address 를 지정한다.

Value Type token

```
(311) <jeus-system> <node> <system-logging> <handler> <smtp-handler>
<bcc-address>
```

Description email 을 숨은 참조로 받는 사람의 address 를 지정한다.

Value Type token

```
(312) <jeus-system> <node> <system-logging> <handler> <smtp-handler>
<send-for-all-messages>
```

Description 이 handler 가 등록한 logger 의 log() method 를 통해 들어온 message 들이 이 handler 로 들어왔을때 이를 email 로 보낼 대상으로 여길지를 설정한다. 만약 false 로 설정되어 있으면 logger 의 특별한 send() method 로 호출된 message 들만 email 로 전송된다. 즉, 처음부터 email 로 보낼 의도로 지정된 message 들만 email 로 전송된다.

Value Type boolean

Default Value false

(313) <jeus-system> <node> <system-logging> <handler> **<socket-handler>**

Description logging 을 지정된 socket 으로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나당 Socket 으로 전송이 되므로 적절한 filter 없이 사용하는 것은 성능 저하를 가져온다.

Child Elements
(314)name
(315)level?
(316)encoding?
(317)filter-class?
(318)address
(319)port

(314) <jeus-system> <node> <system-logging> <handler> <socket-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(315) <jeus-system> <node> <system-logging> <handler> <socket-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value
FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(316) <jeus-system> <node> <system-logging> <handler> <socket-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(317) <jeus-system> <node> <system-logging> <handler> <socket-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을

설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(318) <jeus-system> <node> <system-logging> <handler> <socket-handler> **<address>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 IP address 를 설정한다.

Value Type token

(319) <jeus-system> <node> <system-logging> <handler> <socket-handler> **<port>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 port 를 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(320) <jeus-system> <node> <system-logging> <handler> **<user-handler>**

Description User 가 J2SE logging API 에 따라 만든 handler 를 사용할 경우의 설정이다.

Child Elements
```
(321)handler-class
(322)name
(323)level?
(324)encoding?
(325)filter-class?
(326)handler-property?
(330)formatter-class?
(331)formatter-property?
```

(321) <jeus-system> <node> <system-logging> <handler> <user-handler> **<handler-class>**

Description user 가 만든 handler 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Handler 를 상속받고 jeus.util.logging.JeusHandler 를 구현해야 한다.

Value Type token

Example
```
<handler-
class>com.tmax.logging.handler.MyHandler</handler-
class>
```

(322) <jeus-system> <node> <system-logging> <handler> <user-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(323) <jeus-system> <node> <system-logging> <handler> <user-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(324) <jeus-system> <node> <system-logging> <handler> <user-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(325) <jeus-system> <node> <system-logging> <handler> <user-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(326) <jeus-system> <node> <system-logging> <handler> <user-handler> **<handler-property>**

Description handler 가 생성될 때 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusHandler.setProperty() method 를 통해 handler 로 전달된다.

Child Elements (327)property*

```
(327) <jeus-system> <node> <system-logging> <handler> <user-handler>
<handler-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements (328)key
(329)value

```
(328) <jeus-system> <node> <system-logging> <handler> <user-handler>
<handler-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

```
(329) <jeus-system> <node> <system-logging> <handler> <user-handler>
<handler-property> <property> <value>
```

Description property 의 value 값이다.

Value Type token

```
(330) <jeus-system> <node> <system-logging> <handler> <user-handler>
<formatter-class>
```

Description 이 handler 가 사용할 formatter 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Formatter 를 상속받고 jeus.util.logging.JeusFormatter 를 구현해야 한다.

Value Type token

Default Value jeus.util.logging.SimpleFormatter

Example

```
<formatter-
class>com.tmax.logging.handler.MyHandler</formatter-
class>
```

```
(331) <jeus-system> <node> <system-logging> <handler> <user-handler>
<formatter-property>
```

Description handler 가 생성될 때 만들어진 formatter 에게 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusFormatter.setProperty() method 를 통해 formatter 로 전달된다.

Child Elements (332)property*

(332) <jeus-system> <node> <system-logging> <handler> <user-handler> <formatter-property> **<property>**

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements (333)key
(334)value

(333) <jeus-system> <node> <system-logging> <handler> <user-handler> <formatter-property> <property> **<key>**

Description property 의 key 값이다.

Value Type token

(334) <jeus-system> <node> <system-logging> <handler> <user-handler> <formatter-property> <property> **<value>**

Description property 의 value 값이다.

Value Type token

(335) <jeus-system> <node> **<session-server>**

Description 중앙 집중식 session clustering 을 제공하는 server 에 관한 설정이다. Servlet/JSP 에서 사용하는 HttpSession 클러스터링 및 EJB 세션 클러스터링을 서비스할 수 있다.

Child Elements (336)resolution?
(337)backlog-size?
(338)session-manager*

(336) <jeus-system> <node> <session-server> **<resolution>**

Description session removal, session passivation 등의 operation 을 수행하는 thread 의 활동 주기를 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 60000

(337) <jeus-system> <node> <session-server> **<backlog-size>**

Description 서버 socket 의 backlog 값을 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 50

(338) <jeus-system> <node> <session-server> **<session-manager>**

Description session 을 보관하고 WebContainer/EJB 에 session 객체를 서비스하는 관리자를 설정한다.

Child Elements
(339)name
(340)multi-session?
(341)passivation-to?
(342)removal-to?
(343)file-db-path?
(344)file-db-name?
(345)min-hole?
(346)packing-rate?
(347)check-to?
(348)backup-name?
(349)backup-trigger?
(350)operation-to?

(339) <jeus-system> <node> <session-server> <session-manager> **<name>**

Description session 의 이름을 설정한다. jeus clustering system 상에서 unique 한 값을 지정해야한다.

Value Type token

(340) <jeus-system> <node> <session-server> <session-manager> **<multi-session>**

Description 이 설정은 두개 이상의 session-server 가 연계하여 중앙 집중식 session clustering 을 수행하게 하는 설정이다. clustering 규모가 커서 하나의 session-server 로 session clustering 을 수행하기에 부적절한 환경에서 이 설정을 이용한다.

Value Type multi-sessionType

Defined Value primary
primary 로 지정된 session-manager 끼리 session 객체를 분산하여 관리한다.

backup
backup 으로 지정된 session-manager 들은 primary session-manager 가 정상 동작하지 않을 경우 backup 으로서 서비스한다.

(341) <jeus-system> <node> <session-server> <session-manager> **<passivation-to>**

Description memory 에 존재하는 session 객체를 일정시간 사용하지 않으면 삭제하고 대신 file-db 에 저장된 객체를 사용하게 하는 설정이다.

Value Description -1 이면 passivation 을 수행하지 않는다. 0 이상의 시간을 설정하면 지정된 시간 이상 사용하지 않는 memory 상의 session 객체는 passivation 된다. 단위는 msec 이다. 기본값은 -1 이다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

(342) <jeus-system> <node> <session-server> <session-manager> **<removal-to>**

Description file-db 에 저장된 session 객체의 보존 기간을 지정한다.

Value Description -1 이면 file-db 에서 제거하지 않는다. 0 이상의 시간을 설정한 경우에는, 생성시간으로 부터 지정 시간이 지난 session 객체는 제거한다. 단위는 msec 이다. 기본값은 -1 이다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

(343) <jeus-system> <node> <session-server> <session-manager> **<file-db-path>**

Description file-db 에 경로를 지정한다.

Value Description 절대 경로로 나타내야 한다. 기본 값은 $JEUS_HOME/workspace 이다.

Value Type token

(344) <jeus-system> <node> <session-server> <session-manager> **<file-db-name>**

Description file-db 이름을 지정한다.

Value Description 기본 값은 session-manager 의 name 설정을 이용한다.

Value Type token

(345) <jeus-system> <node> <session-server> <session-manager> **<min-hole>**

Description 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 된다. 이 설정에 지정된 횟수 만큼 file I/O 가 발생하면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

Value Type int

Default Value 1000

(346) <jeus-system> <node> <session-server> <session-manager> **<packing-rate>**

Description 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 된다. 현재 session 객체 갯수 대비 file I/O 횟수가 지정된 ratio 를 넘어서면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

Value Type fractionType

Value Type Description 0 과 1 사이의 float 형으로 비율을 나타낸다.

Default Value 5.0E-1

(347) <jeus-system> <node> <session-server> <session-manager> **<check-to>**

Description 얼마만큼의 시간 간격으로 backup 과정을 수행할 지를 결정한다. 이 설정에 지정된 시간 주기로 update 된 session 객체가 있는지를 조사하고 update 된 session 객체가 존재하면 backup 을 수행한다.

Value Type int

Default Value 30000

(348) <jeus-system> <node> <session-server> <session-manager> **<backup-name>**

Description 이 session-manager 의 backup 으로 사용할 session-manager 의 name 을 지정한다.

Value Type token

(349) <jeus-system> <node> <session-server> <session-manager> **<backup-trigger>**

Description local session manager 에서 session 객체의 update 가 어느 정도 발생하였을 때 backup session manager 로 update 된 session 객체들을 backup 할지를 결정한다. 이 설정에 지정된 횟수 만큼 local session manager 에 session object update 가 발생하면 backup 을 수행한다.

Value Type int

Default Value 1000

(350) <jeus-system> <node> <session-server> <session-manager> **<operation-to>**

Description session-manager 와 WebContainer 간 read operation 에 적용될 timeout 을 지정한다.(RMI 를 사용할 경우에만 의미가 있다.)

Value Type int

Default Value 30000

(351) <jeus-system> <node> **<session-router-config>**

Description 분산식 session clustering 을 제공하는 server 에 관한 설정이다. Servlet/JSP 에서 사용하는 HttpSession 클러스터링을 서비스한다.

Child Elements

(352)connect-timeout?
(353)read-timeout?
(354)backup-trigger?
(355)check-to?
(356)check-level?
(357)default-file-db?
(363)session-router+

(352) <jeus-system> <node> <session-router-config> **<connect-timeout>**

Description WebContainer 에 존재하는 session server 간 socket connection 을 생성할 때 적용되는 timeout 값이다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 60000

(353) <jeus-system> <node> <session-router-config> **<read-timeout>**

Description WebContainer 에 존재하는 session server 간 통신시에 적용되는 read timeout 값이다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 60000

(354) <jeus-system> <node> <session-router-config> **<backup-trigger>**

Description local session server 에서 session 객체의 update 가 어느 정도 발생하였을 때 backup session server 로 update 된 session 객체들을 backup 할지를 결정한다. 이 설정에 지정된 횟수 만큼 local session server 에 session object update 가 발생하면 backup 을 수행한다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 1000

(355) <jeus-system> <node> <session-router-config> **<check-to>**

Description 얼마만큼의 시간 간격으로 backup 과정을 수행할 지를 결정한다. 이 설정에 지정된 시간 주기로 update 된 session 객체가 있는지를 조사하고 update 된 session 객체가 존재하면 backup 을 수행한다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 30000

(356) <jeus-system> <node> <session-router-config> **<check-level>**

Description 사용된 session 을 remote web container 또는 local file db 에 백업하기 전에 백업할 필요가 있는지를 체크하는 것이 필요하다. 이 설정은 백업의 필요성을 체크하는 기준을 정한다. 기본적으로 사용된 세션이 invalidate 되었을 경우 설정한 기준에 관계없이 백업한다.

Value Type check-levelType

Default Value set

Defined Value set
해당 session 의 setAttribute/putValue/removeAttribute/removeValue 함수 호출이 일어난 경우에만 update 된 것으로 간주하여 해당 session 객체를 backup 한다.

get
해당 session 의 setAttribute/putValue/removeAttribute/removeValue/getAttribute/getValue 함수 호출이 일어난 경우에만 update 된 것으로 간주하여 해당 session 객체를 backup 한다.

all
조건없이 사용된 세션은 모두 백업한다. 해당 session 객체가 HttpServletRequest.getSession() API 로 호출될 경우 update 된 것으로 간주하여 해당 session 객체를 backup 한다.

(357) <jeus-system> <node> <session-router-config> **<default-file-db>**

Description update 된 local session 객체를 backup 하는 방법으로는 backup session server 에 backup 하는 방법외에 local file system 상에 backup 하는 방법도 있다. 이 설정은 이와 같이 local file system 상에 update 된 session 객체를 backup 하는 방법을 제공한다. 실제 file backup 은 WebContainer 별로 수행된다. 이 설정은 분산식 session clustering 에 참여하는 모든 WebContainer(session-router)들에 동일하게 적용된다. 단, session-router 하위 element 로 "file-db"가 설정될 경우 이 설정(default-file-db)은 무시되고 "file-db"설정이 적용된다.

Child Elements (358)startup-clear-to?
(359)path?

(360)passivation-to?
(361)min-hole?
(362)packing-rate?

(358) <jeus-system> <node> <session-router-config> <default-file-db> **<startup-clear-to>**

Description WebContainer 를 기동할 때 지정된 file 에 저장된 session 객체들이 복구된다. 만약 현재 시간과 file 의 last modified time 의 시간차가 이 설정에 지정된 값보다 크면 복구를 시도하지 않고 file 의 내용을 모두 clear 한다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Default Value 86400000

(359) <jeus-system> <node> <session-router-config> <default-file-db> **<path>**

Description backup session 을 저장할 file 이름을 지정한다.(절대 경로) 기본값은 $(JEUS_HOME)/sessiondb/<servlet_engine_name>_1.fdb 이다.

Value Type token

(360) <jeus-system> <node> <session-router-config> <default-file-db> **<passivation-to>**

Description memory 에 존재하는 session 객체를 일정시간 사용하지 않으면 삭제하고 대신 file-db 에 저장된 객체를 사용하게 하는 설정이다.

Value Description -1 이면 passivation 을 수행하지 않는다. 0 이상의 시간을 설정하면 지정된 시간 이상 사용하지 않는 memory 상의 session 객체는 passivation 된다. 단위는 msec 이다. 기본값은 -1 이다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Default Value -1

(361) <jeus-system> <node> <session-router-config> <default-file-db>

<min-hole>

Description 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 된다. 이 설정에 지정된 횟수 만큼 file I/O 가 발생하면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

Value Type int

Default Value 1000

(362) <jeus-system> <node> <session-router-config> <default-file-db> **<packing-rate>**

Description 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 된다. 현재 session 객체 갯수 대비 file I/O 횟수가 지정된 ratio 를 넘어서면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

Value Type fractionType

Value Type Description 0 과 1 사이의 float 형으로 비율을 나타낸다.

Default Value 5.0E-1

(363) <jeus-system> <node> <session-router-config> **<session-router>**

Description 분산식 session clustering 에 참여할 WebContainer 를 지정하는데 사용하는 설정이다. 이 외에도 session clustering 에 참여하는 WebContaienr 에 기동될 session server 에 대한 각종 속성을 설정한다.

Child Elements
(364)servlet-engine-name
(365)file-db?
(371)backup-session-router?

(364) <jeus-system> <node> <session-router-config> <session-router> **<servlet-engine-name>**

Description 분산식 session clustering 에 참여할 WebContainer 의 엔진 이름을 지정한다. [Exmaple]: engine1 or myNode_servlet_engine1

Value Type token

(365) <jeus-system> <node> <session-router-config> <session-router> **<file-db>**

Description "default-file-db" 설정과 동일한 역할을 하는 설정이다. 단, 이 설정은

해당 WebContainer(session-router)에만 적용된다. "default-file-db" 설정보다 높은 우선 순위를 갖는다.

Child Elements

(366)startup-clear-to?
(367)path?
(368)passivation-to?
(369)min-hole?
(370)packing-rate?

(366) <jeus-system> <node> <session-router-config> <session-router> <file-db> **<startup-clear-to>**

Description WebContainer 를 기동할 때 지정된 file 에 저장된 session 객체들이 복구된다. 만약 현재 시간과 file 의 last modified time 의 시간차가 이 설정에 지정된 값보다 크면 복구를 시도하지 않고 file 의 내용을 모두 clear 한다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Default Value 86400000

(367) <jeus-system> <node> <session-router-config> <session-router> <file-db> **<path>**

Description backup session 을 저장할 file 이름을 지정한다.(절대 경로) 기본값은 $(JEUS_HOME)/sessiondb/<servlet_engine_name>_1.fdb 이다.

Value Type token

(368) <jeus-system> <node> <session-router-config> <session-router> <file-db> **<passivation-to>**

Description memory 에 존재하는 session 객체를 일정시간 사용하지 않으면 삭제하고 대신 file-db 에 저장된 객체를 사용하게 하는 설정이다.

Value Description -1 이면 passivation 을 수행하지 않는다. 0 이상의 시간을 설정하면 지정된 시간 이상 사용하지 않는 memory 상의 session 객체는 passivation 된다. 단위는 msec 이다. 기본값은 -1 이다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Default Value -1

(369) <jeus-system> <node> <session-router-config> <session-router> <file-db> **<min-hole>**

Description 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 된다. 이 설정에 지정된 횟수 만큼 file I/O 가 발생하면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

Value Type int

Default Value 1000

(370) <jeus-system> <node> <session-router-config> <session-router> <file-db> **<packing-rate>**

Description 일정 시간 file-db 를 운용하면 file 의 크기가 필요이상 커지게 된다. 현재 session 객체 갯수 대비 file I/O 횟수가 지정된 ratio 를 넘어서면 file packing 을 수행하여 필요이상 file 크기가 늘어나는 것을 막는다.

Value Type fractionType

Value Type Description 0 과 1 사이의 float 형으로 비율을 나타낸다.

Default Value 5.0E-1

(371) <jeus-system> <node> <session-router-config> <session-router> **<backup-session-router>**

Description 이 session-router 의 in-memory 백업으로 사용할 session-router 를 지정한다.

Child Elements (372)node-name
(373)servlet-engine-name

(372) <jeus-system> <node> <session-router-config> <session-router> <backup-session-router> **<node-name>**

Description backup 으로 사용할 session-router 가 존재하는 노드의 노드 이름을 설정한다.

Value Type token

(373) <jeus-system> <node> <session-router-config> <session-router> <backup-session-router> **<servlet-engine-name>**

Description backup 으로 사용할 session-router "servlet-engine-name"을 설정한다.

Value Type token

(374) <jeus-system> <node> **<jmx-manager>**

Description JEUS Manager JVM 에서 사용하는 JMX 에 대한 설정이다.

Child Elements
(375)jmx-connector?
(382)html-adaptor-port?
(383)snmp-adaptor?
(395)mlet-url*

(375) <jeus-system> <node> <jmx-manager> **<jmx-connector>**

Description 다른 process 에서 이 Engine Container 의 JMX 를 access 할 때 사용하는 JMX Connector 를 설정한다. 기본적으로는 JEUSMP Connector 를 사용한다.

Child Elements
(376)jmxmp-connector?
(378)rmi-connector?

(376) <jeus-system> <node> <jmx-manager> <jmx-connector> **<jmxmp-connector>**

Description JMX Connector 로 JMXMP Connector 를 사용한다.

Child Elements (377)jmxmp-connector-port?

(377) <jeus-system> <node> <jmx-manager> <jmx-connector> <jmxmp-connector> **<jmxmp-connector-port>**

Description 다른 process 에서 이 Engine Container 의 JMX 를 access 할 때 사용하는 JEUSMP Connector 의 listen port 를 지정한다. 만약 이 값이 0 이거나 지정하지 않으면 JEUSMP Connector 가 사용하는 listen port 를 따로 만들지 않고 jeus 의 공통 port 를 사용한다. 만약 JEUS 의 JMX RemoteAPI 를 사용하지 않고 다른 Runtime 에서 JMXMP protocol 로 접근하고자 한다면 이를 0 이 아닌 다른 값으로 지정해야 한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 0

Defined Value 0
jeus 의 공통 port 를 사용한다.

```
(378) <jeus-system> <node> <jmx-manager> <jmx-connector> <rmi-connector>
```

Description JMX Connector 로 RMI Connector 를 사용한다. 만약 jmxmp-connector 와 같이 설정되어 있는 경우에는 JEUS system 내부적으로는 jmxmp-connector 를 기본적으로 사용하게 된다. 또한 이 경우에는 rmi-connector 의 ref-export-name 이 별도로 설정되어 있어야 한다. 이 이름이 JEUS 에서 기본적으로 사용하는 이름과 같거나 설정이 되어있지 않다면 exception 이 발생한다.

Child Elements
(379)rmi-connector-port?
(380)export-name?
(381)ref-export-name?

```
(379) <jeus-system> <node> <jmx-manager> <jmx-connector> <rmi-connector> <rmi-connector-port>
```

Description 다른 process 에서 이 Engine Container 의 JMX 를 access 할 때 사용하는 RMI Connector 의 port 를 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Defined Value 0
random 하게 port 를 지정한다.

```
(380) <jeus-system> <node> <jmx-manager> <jmx-connector> <rmi-connector> <export-name>
```

Description JEUS 의 기본 export name 대신에 다른 export name 을 사용할때 설정한다. 이 export name 은 JMXServiceURL 의 URL path 에 들어가게 된다.

Value Type token

(381) <jeus-system> <node> <jmx-manager> <jmx-connector> <rmi-connector> **<ref-export-name>**

Description 이 connector 를 얻을수 있는 jndi name 을 JEUS 의 기본 jndi name 이 아닌 다른 name 으로 지정하고자 할 때 사용한다. 이 export name 으로 lookup 하면 JMXConnector 객체를 얻을수 있다.

Value Type token

(382) <jeus-system> <node> <jmx-manager> **<html-adaptor-port>**

Description JMX 의 adapter 중 하나인 HTML adapter 의 port 를 지정한다. 여기에 지정된 값으로 Web Browser 가 접속하게 된다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

(383) <jeus-system> <node> <jmx-manager> **<snmp-adaptor>**

Description JMX 의 adapter 중 하나인 SNMP adapter 를 설정한다.

Child Elements
(384) snmp-adaptor-port
(385) snmp-version?
(386) snmp-max-packet-size?
(387) snmp-security?
(388) trap-demon*
(391) pooling?

(384) <jeus-system> <node> <jmx-manager> <snmp-adaptor> **<snmp-adaptor-port>**

Description SNMP 어댑터의 리스너 포트

Value Type snmp-adaptor-portType

(385) <jeus-system> <node> <jmx-manager> <snmp-adaptor> **<snmp-version>**

Description SNMP 버전을 지정하며 1, 2 또는 3 을 지정할 수 있다

Value Type snmp-versionType

Default Value 3

Defined Value 1

2

3

(386) <jeus-system> <node> <jmx-manager> <snmp-adaptor> **<snmp-max-packet-size>**

Description SNMP 패킷에 대한 최대값을 설정하며 최소 256 바이트부터 설정 할 수 있다.

Value Type snmp-max-packet-sizeType

Default Value 4096

(387) <jeus-system> <node> <jmx-manager> <snmp-adaptor> **<snmp-security>**

Description 보안을 적용시킬 것 인지를 설정한다. 보안은 SNMP 버전 3 에서만 지정이 가능 하다.

Value Type boolean

Default Value false

(388) <jeus-system> <node> <jmx-manager> <snmp-adaptor> **<trap-demon>**

Description 장애 상황 발생시 TRAP 메세지를 보낼 서버를 설정한다. 여러 개 설정이 가능하며 설정된 모든 ip, address 로 메세지를 보낸다.

Child Elements (389)ip-address
(390)port

(389) <jeus-system> <node> <jmx-manager> <snmp-adaptor> <trap-demon> **<ip-address>**

Description Demon 의 IP address

Value Description a valid IP address

Value Type token

Example <host-name>111.111.111.1</host-name>

(390) <jeus-system> <node> <jmx-manager> <snmp-adaptor> <trap-demon> **<port>**

Description Demon 의 port

Value Description a port number

Value Type int

Example
```
<port>8888</port>
```

(391) <jeus-system> <node> <jmx-manager> <snmp-adaptor> **<pooling>**

Description SNMP Server 에서 요청을 처리하는 쓰레드로 구성되어 있다. 아래 element 는 이 쓰레드를 관리하는 pool 을 설정한다.

Child Elements
```
(392)min?
(393)max?
(394)period?
```

(392) <jeus-system> <node> <jmx-manager> <snmp-adaptor> <pooling> **<min>**

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

(393) <jeus-system> <node> <jmx-manager> <snmp-adaptor> <pooling> **<max>**

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

(394) <jeus-system> <node> <jmx-manager> <snmp-adaptor> <pooling> **<period>**

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로

지정한다.

(395) <jeus-system> <node> <jmx-manager> **<mlet-url>**

Description 이 Engine Container 의 MBeanServer 에 등록할 MLet 의 URL 을 지정한다.

Value Type token

(396) <jeus-system> **<naming-server>**

Description Naming-Server element 는 JEUS Naming Server 의 정보를 포함한다.

Child Elements
(397)server?
(407)local?

(397) <jeus-system> <naming-server> **<server>**

Description server element 는 JNSServer 가 다른 JNSServer 와 그것의 JNSLocal 과 연결을 관리하기 위해 사용하는 리소스들을 정한다.

Child Elements
(398)use-nio?
(399)export-cos-naming?
(400)backlog-size?
(401)resolution?
(402)buffer-size?
(403)pooling?

(398) <jeus-system> <naming-server> <server> **<use-nio>**

Description JNSServer 가 Nonblocking I/O 를 사용해서 통신을 할지의 여부를 지정한다.

Value Type boolean

Default Value true

Performance Recommendation 많은 수의 Engine Container 및 client 를 사용할 경우에는 Nonblocking I/O 가 더 좋은 효율을 보인다. JNDI 를 사용하는 client 의 수가 적다면 Blocking I/O 가 더 효율적일수 있다.

(399) <jeus-system> <naming-server> <server> **<export-cos-naming>**

Description Export-cos-naming element 는 JEUS 가 COS Naming Server(tnameserver)로 작동할 것인지 아닌지를 정한다. 만약 true 이면,tnameserver 는 %JEUS_BASEPORT%+4 포트를 가지고

실행을 시작할 것이다.

Value Type boolean

Default Value false

(400) <jeus-system> <naming-server> <server> **<backlog-size>**

Description backlog-size element 는 다른 Naming Server 의 접속을 받아들이는 한계인 back log 의 크기를 정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 50

(401) <jeus-system> <naming-server> <server> **<resolution>**

Description resolution element 는 JEUS Manager 가 JNS 서버의 리소스를 점검하는 시간 간격을 정한다.

Value Description millisecond 단위이다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 60000

(402) <jeus-system> <naming-server> <server> **<buffer-size>**

Description buffer-size 는 JNSServer 가 JNSLocal 이나 다른 JNSServer 와 통신하는데 사용되는 버퍼의 크기를 정한다.

Value Description Bytes 단위이다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 20480

(403) <jeus-system> <naming-server> <server> **<pooling>**

Description pooling element 는 load-balance 를 유지하기 위해서 JNSServer 의 thread pool 구성을 정한다.

Child Elements

(404)min?
(405)max?
(406)period?

```
(404) <jeus-system> <naming-server> <server> <pooling> <min>
```

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

```
(405) <jeus-system> <naming-server> <server> <pooling> <max>
```

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

```
(406) <jeus-system> <naming-server> <server> <pooling> <period>
```

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(407) <jeus-system> <naming-server> <local>
```

Description local element 는 JNSLocal 이 JNSServer 와 JNDI 트리의 내용들을 얻기 위해서 사용하는 리소스를 정한다.

Child Elements

(408)resolution?
(409)buffer-size?
(410)pooling?

```
(408) <jeus-system> <naming-server> <local> <resolution>
```

Description resolution element 는 JEUS Manager 가 JNS 서버의 리소스를 점검하는 시간 간격을 정한다.

Value Description millisecond 단위이다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 60000

(409) <jeus-system> <naming-server> <local> **<buffer-size>**

Description buffer-size 는 JNSServer 가 JNSLocal 이나 다른 JNSServer 와 통신하는데 사용되는 버퍼의 크기를 정한다.

Value Description Bytes 단위이다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 20480

(410) <jeus-system> <naming-server> <local> **<pooling>**

Description pooling element 는 load-balance 를 유지하기 위해서 JNSServer 의 thread pool 구성을 정한다.

Child Elements
(411)min?
(412)max?
(413)period?

(411) <jeus-system> <naming-server> <local> <pooling> **<min>**

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

(412) <jeus-system> <naming-server> <local> <pooling> **<max>**

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

(413) <jeus-system> <naming-server> <local> <pooling> **<period>**

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

(414) <jeus-system> **<security-manager>**

Description security-manager elment 는 JEUS security 정보 관리에 대한 설명이다.

Child Elements (415)domain-name*

(415) <jeus-system> <security-manager> **<domain-name>**

Description 이 JEUS Manager 에서 사용할 security domain 들을 설정한다. 자세한 것은 JEUS Security 메뉴얼을 참고하기 바란다.

Value Type token

(416) <jeus-system> **<resource>**

Description 여기서는 JEUS 와 연동되는 외부 리소스의 정보를 담고 있다. 여기서 사용되는 리소스는 DB, TP monitor 가 있다.

Child Elements
(417)data-source?
(459)mail-source?
(465)url-source?
(469)external-source?
(525)jaxr-source?

(417) <jeus-system> <resource> **<data-source>**

Description 어플리케이션에서 사용할 수 있는 datasource 를 정의 한다.

Child Elements
(418)database*
(455)cluster-ds*

(418) <jeus-system> <resource> <data-source> **<database>**

Description

DataSource 를 구성하기 위해 JDBC 드라이버에 필수 적인 특성들을 담고 있다.

Child Elements

(419)vendor
(420)export-name
(421)data-source-class-name
(422)data-source-type
(423)database-name?
(424)data-source-name?
(425)service-name?
(426)description?
(427)network-protocol?
(428)encryption?
(429)port-number?
(430)server-name?
(431)user?
(432)password?
(433)driver-type?
(434)property*
(438)connection-pool?

(419) <jeus-system> <resource> <data-source> <database> **<vendor>**

Description

JDBC 드라이버 밴더의 이름.

Value Type

vendorType

Defined Value

oracle
Oracle JDBC driver

sybase
Sybase JDBC driver

mssql
JEUS MS SQL Server driver

db2
DB2 JDBC driver

others

기타 JDBC driver

(420) <jeus-system> <resource> <data-source> <database> **<export-name>**

Description DataSource 의 JNDI 이름. 이 값은 Naming Server 에 datasource 를 등록할 때 사용될 것이다.

Value Type token

(421) <jeus-system> <resource> <data-source> <database> **<data-source-class-name>**

Description JDBC 드라이버의 datasource 클래스 이름.

Value Type token

Example
```
<data-source-class-name>oracle.jdbc.pool.OracleConnectionPoolDataSource</data-source-class-name>
```

(422) <jeus-system> <resource> <data-source> <database> **<data-source-type>**

Description DataSource 의 타입.

Value Type data-source-typeType

Defined Value

DataSource
Connection 을 반환하는 기본적인 DataSource.

ConnectionPoolDataSource
Connection Pool 로부터 Connection 을 반환한다.

LocalXADataSource
XA 연결 Pool 로부터 Local Transaction 역할을 하는 connection 을 반환한다

XADataSource
XA 연결 Pool 로부터 분산/전역 Transaction 역할을 하는 Connection 을 반환한다.

(423) <jeus-system> <resource> <data-source> <database> **<database-name>**

Description Database 의 이름. Oracle 은 database 의 SID.

Value Type token

(424) <jeus-system> <resource> <data-source> <database> **<data-source-name>**

Description DataSource 의 이름. 드라이버 밴더에 의존적이며 일반적으로 DataSourceClassName 값과 동일하다.

Value Type token

(425) <jeus-system> <resource> <data-source> <database> **<service-name>**

Description 단지 i-net JDBC 드라이버에서만 해당하는 설정으로 Oracle DB 의 SID 를 지정한다.

Value Type token

(426) <jeus-system> <resource> <data-source> <database> **<description>**

Description DataSource 에 대한 설명을 할수 있는 element 이다.

Value Type token

Example <description>이 DataSource 는 Oracle XA 를 사용한다.</description>

(427) <jeus-system> <resource> <data-source> <database> **<network-protocol>**

Description Database 와 연결에 사용되는 프로토콜을 나타낸다.

Value Description Sybase 의 경우 "Tds"이고 JDBC 밴더에 의존적이다.

Value Type token

(428) <jeus-system> <resource> <data-source> <database> **<encryption>**

Description 이 설정이 true 로 되어 있다면 <password>에 지정된 값은 실제 password 를 Base64 로 encoding 한 값이라고 간주한다.

Value Type boolean

Default Value false

(429) <jeus-system> <resource> <data-source> <database> **<port-number>**

Description Database listener 의 포트번호.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

(430) <jeus-system> <resource> <data-source> <database> **<server-name>**

Description Database 가 실행되는 곳의 서버이름.

Value Type token

(431) <jeus-system> <resource> <data-source> <database> **<user>**

Description DB 사용자 ID 로 transaction 처리등을 위해서는 충분한 system 특권을 가지고 있어야 한다.

Value Type token

(432) <jeus-system> <resource> <data-source> <database> **<password>**

Description DB 사용자의 password 이다. <encryption> 설정에 따라 Base64 로 인코딩된 값이 들어올 수 있다.

Value Type token

(433) <jeus-system> <resource> <data-source> <database> **<driver-type>**

Description JDBC 드라이버의 타입으로 Oracle 드라이버에만 해당되는 설정이다.

Value Type token

Example

```
<driver-type>thin</driver-type>
```

(434) <jeus-system> <resource> <data-source> <database> **<property>**

Description vendor 가 other 인 경우 DataSource 설정에 필요한 property 들을 기술한다.

Child Elements

(435)name
(436)type
(437)value

(435) <jeus-system> <resource> <data-source> <database> <property> **<name>**

Description 프로퍼티의 이름.

Value Type token

(436) <jeus-system> <resource> <data-source> <database> <property> **<type>**

Description 프로퍼티의 값의 타입.

Value Description primitive 타입이나 java.lang.String, Primitive Wrapper Class 등이 올 수 있다.

Value Type token

Example
```
<type>java.lang.Integer</type>
```

(437) <jeus-system> <resource> <data-source> <database> <property> **<value>**

Description property 의 값.

Value Type token

(438) <jeus-system> <resource> <data-source> <database> **<connection-pool>**

Description Connection Pool 를 위해 필요한 정보를 담고 있다.

Child Elements
(439)pooling?
(444)wait-free-connection?
(447)operation-to?
(448)delegation-datasource?
(449)max-use-count?
(450)delegation-dba?
(451)dba-timeout?
(452)check-query?
(453)stmt-caching-size?
(454)stmt-fetch-size?

(439) <jeus-system> <resource> <data-source> <database> <connection-pool> **<pooling>**

Description DB Connection Pooling 에 관한 정보를 담고 있다.

Child Elements
(440)min?
(441)max?
(442)step?
(443)period?

```
(440) <jeus-system> <resource> <data-source> <database> <connection-
pool> <pooling> <min>
```

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

```
(441) <jeus-system> <resource> <data-source> <database> <connection-
pool> <pooling> <max>
```

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

```
(442) <jeus-system> <resource> <data-source> <database> <connection-
pool> <pooling> <step>
```

Description pooling 되는 객체가 증가될때의 증가량을 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 4

```
(443) <jeus-system> <resource> <data-source> <database> <connection-
pool> <pooling> <period>
```

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(444) <jeus-system> <resource> <data-source> <database> <connection-
```

```
pool> <wait-free-connection>
```

Description Pool 안에 있는 모든 connection 들이 점유되어 있을 때 연결요청을 핸들링하는 메소드를 정의한다.

Child Elements
```
(445)enable-wait?
(446)wait-time?
```

```
(445) <jeus-system> <resource> <data-source> <database> <connection-
pool> <wait-free-connection> <enable-wait>
```

Description 이 태그는 pool 안에 이용 가능한 connection 이 없거나 pool 안에 connection 들이 이미 최대값이 되어질 때 connection 을 요청 처리하는 방법을 결정한다. 만약 true 라면 시스템은 이용 가능한 connection 을 얻기위해 대기한다. 만약 false 라면, 시스템은 사용자 요청이 올 때 새로운 connection 을 만들고 사용이 끝난 이후에 pool 에 반납하지 않는다.

Value Type boolean

Default Value false

```
(446) <jeus-system> <resource> <data-source> <database> <connection-
pool> <wait-free-connection> <wait-time>
```

Description 이 태그는 <enable-wait>가 true 일때만 유효하다. 이것은 사용자가 connection 을 위해 대기하는 시간을 나타낸다. 만약 어떠한 connection 도 사용자가 이 시간동안 대기하여도 이용할 수 없을 때는 시스템은 사용자에게 exception 을 던져준다.

Value Description millisecond 단위이다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 10000

```
(447) <jeus-system> <resource> <data-source> <database> <connection-
pool> <operation-to>
```

Description DB operation 의 time out 을 결정하는데 사용되는 시간주기를 정의한다.

Value Description millisecond 단위이다.

Value Type int

Default Value 30000

(448) <jeus-system> <resource> <data-source> <database> <connection-pool> **<delegation-datasource>**

Description 이 설정은 현재의 XA DataSource 의 global/local transaction 을 NonXA-DataSource 로 넘길때 사용한다. 이 경우 여기에 지정된 datasource 의 connection 을 사용한다.

Value Description 대리할 datasource 의 export name.

Value Type token

(449) <jeus-system> <resource> <data-source> <database> <connection-pool> **<max-use-count>**

Description 하나의 connetion 은 이 element 회수만큼 사용되고 교체된다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 0

Defined Value 0
무한대를 의미한다. 즉, 한번 얻어진 Connection 은 계속 사용된다.

(450) <jeus-system> <resource> <data-source> <database> <connection-pool> **<delegation-dba>**

Description "DBA" 권한을 가진 특별한 <database> 를 지정한다. 이 connection 은 DBMS 관리자의 connection 으로, 특권을 가진 사용자 ID 와 password 로 생성된다.이 connection 은 DB connection 에 문제가 발생했을 때 JEUS 가 사용한다. 예를 들면, JEUS 가 DB connection 을 종료하려고 하지만 Exception 이 발생해서 실패하는 경우를 보자. 이 경우 JEUS 는 low-level 의 "kill" 시그널을 보내어 강제로 close 하려고 한다. 이 때 이 "kill" 시그널을 보내기 위해서는 DBA connection 이 필요하다.이 element 에는 DBA connection 을 설정한 DataSource 의 export-name 을 넣어준다. 그러면 이 DataSource 를 사용해서 "kill" 시그널을 보내게 된다. 이 element 는 Oracle 과 Sybase 에서만 적용된다.

Value Description DBA database connection 구성의 JNDI export name

Value Type token

```
(451) <jeus-system> <resource> <data-source> <database> <connection-
pool> <dba-timeout>
```

Description DBA 모드는 Oracle, Sysbase 의 DB 에서 사용된다. DBA 모드는 "kill" 시그널을 보낼 수 있는 권한을 말하며, DB 상에 문제(Deadlock 같은)가 발생했을 때 어플리케이션 서버가 사용하게 된다. 이 element 는 "kill" 시그널을 보내기 전, getConnection()부터 Connection Pool 로의 반환까지 동안의 시간을 지정한다. 이 시간이 초과되면 "kill" 시그널이 강제로 그 Connection 을 중단시켜버린다. 이 옵션은 Oracle 과 Sybase 에서만 사용된다.

Value Description millisecond 단위이다.

Value Type off-intType

Value Type Description 기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.

Default Value -1

```
(452) <jeus-system> <resource> <data-source> <database> <connection-
pool> <check-query>
```

Description 어떤 DB 는 getConnection()에서 새로운 DB Connection 을 요청할 때 문제가 있어도 제대로 보고하지를 않는다. 이런 DB 를 위해서 JEUS 에서는 "check query"를 보내어서 Connection 의 상태를 알아온다. check query 는 간단한 SQL 로서, 제대로 된 결과를 가져와야 한다. 이 query 는 JEUS 에서 Connection 의 상태를 알아야 할 때 DB 로 전송된다. Oracle, Sybase, MS SQL Server 와 같이 setAutoCommit()을 통해서 에러를 보고할 수 있다면, 이 기능을 사용하지 않아야 한다.

Value Description DB 로 전송될 간단한 SQL 문장

Value Type token

Example

```
<check-query>SELECT check FROM
checktable;</check-query>
```

Performance Recommendation	Oracle, Sybase, MS SQL 에서는 성능 저하가 발생하므로 사용하지 않는다.

(453) <jeus-system> <resource> <data-source> <database> <connection-pool> **<stmt-caching-size>**

Description	어플리케이션에서 prepareStatement()를 호출할 때 마다 SQL 문장이 전처리되고, PreparedStatement 가 DB 로 전송된다. 매 호출마다 PreparedStatement 가 계속 재생성되는 것을 피하기 위해서, JEUS 는 PreparedStatement 를 내부적으로 캐시할 수 있다. 이 element 는 PreparedStatement 를 캐시할 개수를 지정한다. 가장 많이 사용되는 PreparedStatement 들만 캐시에서 유지된다.
Value Type	off-intType
Value Type Description	기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.
Default Value	-1
Performance Recommendation	이 element 는 성능을 위해서 꼭 사용한다. 어플리케이션에서 Connection.prepareStatement()를 사용한다면 10 이상의 값을 설정한다. 그러면 성능 향상이 있다.

(454) <jeus-system> <resource> <data-source> <database> <connection-pool> **<stmt-fetch-size>**

Description	Connection 에서 만드는 statement 의 fetch size 를 설정한다.
Value Type	off-intType
Value Type Description	기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.
Default Value	-1

(455) <jeus-system> <resource> <data-source> **<cluster-ds>**

Description	어플리케이션 서버 차원에서 데이터베이스의 FailOver 기능을 제공하기 위해서 데이터소스 클러스터링을 사용한다. 데이터소스 클러스터링은 근본적으로는 하나의 JNDI export name 을 가진 데이터소스 인스턴스이다. 이 인스턴스는 DB 호출을 받아서 여러 DB(실제 데이터소스) 중 하나로 전달시켜주는 역할을 한다. 만약 주

DB 가 다운되었을 경우, 클러스터링의 다른 DB 가 선택되어서 어플리케이션의 요청 사항을 처리하게 된다.

Child Elements

```
(456)export-name
(457)is-pre-conn?
(458)backup-data-source+
```

```
(456) <jeus-system> <resource> <data-source> <cluster-ds> <export-name>
```

Description 데이터소스 클러스터링의 export name

Value Type token

```
(457) <jeus-system> <resource> <data-source> <cluster-ds> <is-pre-conn>
```

Description 이 기능이 사용되면, 데이터소스 클러스터링은 클러스터링 상의 모든 DB 에서 Connection 을 열게된다. 이것은 성능에는 좋지만 리소스의 절약면에서는 좋지 못하다. 만약 true 로 설정하면 총 생성된 Connection 의 개수가 <connection-pool> <pooling> <min>값과 동일하게 된다.

Value Type boolean

Default Value false

Performance Recommendation 성능 향상을 위해서는 “true”를, 리소스의 절약을 위해서는 “false”를 설정한다.

```
(458) <jeus-system> <resource> <data-source> <cluster-ds> <backup-data-source>
```

Description 이 클러스터링에 참여할 DB 의 export name 을 ‘,’로 분리해서 적는다. 리스트의 첫번째 DB 가 주 DB 로 동작한다.

Value Type token

Example

```
<backup-data-source>datasource1, datasource2,
datasource3</backup-data-source>
```

```
(459) <jeus-system> <resource> <mail-source>
```

Description Mail Source 는 클라이언트 어플리케이션에서 메일을 보낼 때 사용되어질 여러 SMTP 호스트를 설정한다.

Child Elements

```
(460)mail-entry*
```

(460) <jeus-system> <resource> <mail-source> **<mail-entry>**

Description 각 mail entry 는 JNDI naming lookup 을 통해서 이용하는 e-mail host 를 말한다. 이 기능에 대한 더 많은 정보는 JavaMail 1.2 스펙을 참조하기 바란다.

Child Elements (461)export-name
(462)mail-property*

(461) <jeus-system> <resource> <mail-source> <mail-entry> **<export-name>**

Description 클라이언트는 서비스에 등록되어 있는 이름을 간접적으로 사용한다. 이는 클라이언트 디스크립터에서 export name 으로 bind 한 것이다. 이 이름은 java.mail.Session 객체로 bind 된다.

Value Type token

(462) <jeus-system> <resource> <mail-source> <mail-entry> **<mail-property>**

Description 이는 mail host 를 지정하거나 접근하는데 사용된다.

Child Elements (463)name
(464)value

(463) <jeus-system> <resource> <mail-source> <mail-entry> <mail-property> **<name>**

Description mail property 의 이름. property 이름은 JavaMail 1.2 스펙을 따라야 한다

Value Type token

Defined Value mail.user
서버에서 인식하는 사용자 명.

mail.host
메일 서버의 IP 주소.

mail.from
보내는 측의 e-mail.

mail.transport.protocol

사용할 메일 프로토콜.

(464) <jeus-system> <resource> <mail-source> <mail-entry> <mail-property> **<value>**

Description mail property 의 값.

Value Type token

(465) <jeus-system> <resource> **<url-source>**

Description URL Source 는 클라이언트가 표준 JNDI lookup 방식으로 URL resource 에 접근 가능하도록 하기 위해 Naming Server 의 JNDI name 에 URL 주소를 bind 하는데 사용된다.

Child Elements (466)url-entry*

(466) <jeus-system> <resource> <url-source> **<url-entry>**

Description 각 entry 는 실제 URL 주소와 JNDI name 을 매핑한다. 이는 URL 이 Naming Server 에 bind 되는것을 말한다.

Child Elements (467)export-name
(468)url

(467) <jeus-system> <resource> <url-source> <url-entry> **<export-name>**

Description JNDI name 은 Naming Server 에 URL 을 bind 할 때 사용된다.

Value Type token

Example <export-name>MYURL</export-name>

(468) <jeus-system> <resource> <url-source> <url-entry> **<url>**

Description URL 은 bind 된 JNDI Server 의 JNDI name 에 매핑된다.

Value Type token

Example <url>http://www.foo.com</url>

(469) <jeus-system> <resource> **<external-source>**

Description 이 element 는 IBM MQ 나 Tmax 에 연결할 때 사용된다.

Child Elements (470)jms-source*
(482)tmax*

(470) <jeus-system> <resource> <external-source> **<jms-source>**

Description JEUS Transation Manager 와 IBM MQ 나 Sonic MQ 등 messasing source 제품 사이의 상호 작용을 하기위해서는 아래 element 들을 설정해야 한다. 이 절의 설정에 대한 자세한 정보는 해당 제품 메뉴얼을 참조하기 바란다.

Child Elements
(471)vendor
(472)factory-class-name
(473)resource-type
(474)export-name
(475)queue?
(476)queueManager?
(477)topic?
(478)property*

(471) <jeus-system> <resource> <external-source> <jms-source> **<vendor>**

Description jms source 드라이버 밴더의 이름.

Value Type jmsVendorType

Defined Value
ibmmq
ibm mq driver

sonicmq
sonic mq driver

others
기타 jms source driver

(472) <jeus-system> <resource> <external-source> <jms-source> **<factory-class-name>**

Description jms source 드라이버의 factory 클래스 이름.

Value Type token

(473) <jeus-system> <resource> <external-source> <jms-source> **<resource-type>**

Description jms source type

Value Type typeResourceType

Defined Value

QCF
IBM MQ 나 Sonic MQ 매뉴얼을 참조하기 바란다.

TCF
IBM MQ 또는 Sonic MQ 매뉴얼을 참조하기 바란다.

Q
IBM MQ 또는 Sonic MQ 매뉴얼을 참조하기 바란다.

T
IBM MQ 또는 Sonic MQ 매뉴얼을 참조하기 바란다.

XAQCF
IBM MQ 또는 Sonic MQ 매뉴얼을 참조하기 바란다.

XATCF
IBM MQ 또는 Sonic MQ 매뉴얼을 참조하기 바란다.

LOCALXAQCF
IBM MQ 또는 Sonic MQ 매뉴얼을 참조하기 바란다.

LOCALXATCF
IBM MQ 또는 Sonic MQ 매뉴얼을 참조하기 바란다.

(474) <jeus-system> <resource> <external-source> <jms-source> **<export-name>**

Description JNDI 에 등록되어 서비스 되는 이름이다.

Value Type token

(475) <jeus-system> <resource> <external-source> <jms-source> **<queue>**

Description resource-type 이 Q 일때만 사용된다.자세한 것은 IBM MQ 나 Sonic MQ 매뉴얼을 참조하기 바란다.

Value Type token

(476) <jeus-system> <resource> <external-source> <jms-source> **<queueManager>**

Description T type 을 제외한 ibmmq 를 사용할때만 사용된다. 자세한 것은 IBM MQ 매뉴얼을 참조하기 바란다.

Value Type token

(477) <jeus-system> <resource> <external-source> <jms-source> **<topic>**

Description resource-type 이 T 일때만 사용된다.자세한 것은 IBM MQ 나 Sonic MQ 매뉴얼을 참조하기 바란다.

Value Type token

(478) <jeus-system> <resource> <external-source> <jms-source> **<property>**

Description jms source 설정에 필요한 property 들을 기술한다.

Child Elements (479)name
(480)type
(481)value

(479) <jeus-system> <resource> <external-source> <jms-source> <property> **<name>**

Description 프로퍼티의 이름.

Value Type token

(480) <jeus-system> <resource> <external-source> <jms-source> <property> **<type>**

Description 프로퍼티의 값의 타입.

Value Description primitive 타입이나 java.lang.String, Primitive Wrapper Class 등이 올 수 있다.

Value Type token

Example <type>java.lang.Integer</type>

(481) <jeus-system> <resource> <external-source> <jms-source> <property> **<value>**

Description property 의 값.

Value Type token

(482) <jeus-system> <resource> <external-source> **<tmax>**

Description 이 element 는 JEUS JNDI namespace 에 있는 Tmax TP monitor 서버를 export 하는데 사용된다. Tmax 제품의 사용법은 Tmax 매뉴얼을 참조하기 바란다.

Child Elements
(483)export-name
(484)webt-datasource-type?
(485)webt-logging?
(490)host-name
(491)port
(492)backup-host-name?
(493)backup-port?
(494)fdb-file?
(495)default-charset?
(496)enable-pipe?
(497)enable-extended-header?
(498)inbuf-size?
(499)outbuf-size?
(500)max-idle-timeout?
(501)service-timeout?
(502)transaction-timeout?
(503)transaction-block-timeout?
(504)security?
(509)tmax-connection-pool?
(518)monitoring?
(522)tmax-property*

(483) <jeus-system> <resource> <external-source> <tmax> **<export-name>**

Description JNDI 에 등록되어 서비스 되는 이름이다.

Value Description a JNDI export name

Value Type token

Example <export-name>webtResource</export-name>

(484) <jeus-system> <resource> <external-source> <tmax> **<webt-datasource-type>**

Description WebtDataSource 의 type 을 정의한다.

Value Type webt-datasource-poolType

Default Value WebtConnectionPoolDataSource

Defined Value WebtConnectionPoolDataSource
일반 ConnectionPool Service 를 지원하는 WebtDataSource 타입이다.

WebtXADataSource
XA Service 를 지원하는 WebtDataSource 타입이다.

Example
```
<webt-datasource-
type>WebtConnectionPoolDataSource</webt-
datasource-type>
```

(485) <jeus-system> <resource> <external-source> <tmax> **<webt-logging>**

Description Webt 모듈에서 남겨지는 error 로그 설정이다.

Example
```
<webt-logging> ... </webt-logging>
```

Child Elements
```
(486)file-name?
(487)level?
(488)valid-day?
(489)buffer-size?
```

(486) <jeus-system> <resource> <external-source> <tmax> <webt-logging> **<file-name>**

Description 생성될 WebT log file 경로와 이름을 정의한다. OS 가 Windows 인 경우에는 파일 구분자를 '\'가 아닌 '\\'로 해 주어야 한다.

Value Type token

Default Value webt.log

Example
```
<file-name>/home/jeus/log/webt.log</file-name>
```

(487) <jeus-system> <resource> <external-source> <tmax> <webt-logging> **<level>**

Description WebT log level 다음의 none|info|debug 값들 중 하나를 정의한다.

Value Description none/info/debug

Value Type token

Default Value none

Defined Value	none 로그 메시지를 남기지 않는다. [info] 기타 로그 메시지 이외의 Non-critical messages 추가로 볼 수 있다. [debug] debugging 을 하기위한 모든 로그 메시지 정보를 살펴볼 수 있다
Example	`<level>debug</level>`

(488) <jeus-system> <resource> <external-source> <tmax> <webt-logging> **<valid-day>**

Description	지정된 로그파일에 기록된 내용에 날짜의 제한 값을 설정한다. 만약 1 이상이라는 값으로 설정되었을 경우 하루 단위로 로그 파일이 대체되어 해당 시간에 맞는 log message 를 기록한다.
Value Description	a number of days
Value Type	off-intType
Value Type Description	기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.
Default Value	-1
Defined Value	-1 만약 -1 값으로 설정되었을 경우 지정된 하나의 로그 파일에 WebT log message 를 모두 남긴다
Example	`<valid-day>2</valid-day>`

(489) <jeus-system> <resource> <external-source> <tmax> <webt-logging> **<buffer-size>**

Description	Log file 에 WebT log message 를 남길 때 이 buffer size 만큼 메모리에서 임시로 저장하였다가 한꺼번에 기록한다.
Value Description	bytes
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	0

Defined Value 0
0 값으로 지정하면 log message 에 대한 buffering 을 하지 않는다

Example
```
<buffer-size>1024</buffer-size>
```

(490) <jeus-system> <resource> <external-source> <tmax> **<host-name>**

Description Tmax server 의 이름이나 IP 주소를 말한다.

Value Description a valid IP address

Value Type token

Example
```
<host-name>111.111.111.1</host-name>
```

(491) <jeus-system> <resource> <external-source> <tmax> **<port>**

Description Tmax server 환경 파일에 설정된 "PORT" 값과 동일한 값을 설정해 주어야 한다.

Value Description a port number

Value Type int

Example
```
<port>8888</port>
```

(492) <jeus-system> <resource> <external-source> <tmax> **<backup-host-name>**

Description Main Tmax Server 의 failover 상황에 직면하였을 때 서비스에 무리가 없도록 설정할 Backup Tmax Server 의 IP address 값을 설정한다.

Value Description a valid IP address

Value Type token

Example
```
<backup-addresss>111.111.111.2</backup-address>
```

(493) <jeus-system> <resource> <external-source> <tmax> **<backup-port>**

Description Backup Tmax Server 의 환경 파일에 "PORT" 값 동일한 값을 설정해 주어야 한다.

Value Description a port number

Value Type int

Example `<backup-port>8889</backup-port>`

(494) <jeus-system> <resource> <external-source> <tmax> **<fdb-file>**

Description tmax fdl file 의 위치를 설정한다

Value Description a tmax fdl filename

Value Type token

Example `<fdl-file>/home/tmax/fdl/tmax.fdl</fdl-file>`

(495) <jeus-system> <resource> <external-source> <tmax> **<default-charset>**

Description Application 에 적용될 default character set 을 설정한다. 기본적으로 WebtSystem.setDefaultCharset(java.lang.String charset)수행과 동일하다.

Value Description a encoding name

Value Type token

Example `<default-charset>euc-kr</default-charset>`

(496) <jeus-system> <resource> <external-source> <tmax> **<enable-pipe>**

Description WebT 와 Tmax server 가 local machine 에 같이 존재할 경우 pipe 통신 여부를 설정하여 performance 를 꾀할 수 있다.

Value Description pipe 통신 enable/disable 여부

Value Type boolean

Default Value false

Example `<enable-pipe>true</enable-pipe>`

(497) <jeus-system> <resource> <external-source> <tmax> **<enable-extended-header>**

Description Tmax Header 확장된 3.11.x 이후 버전과의 서비스가 성공적으로 이루어 지도록 설정하는 option 값.

Value Description tmax header 확장 사이즈 여부

Value Type boolean

Default Value false

Defined Value true
Tmax header size 가 112byte 로 확장된 tmax engine 과의 서비스.

false
Tmax header size 가 기본 96byte 인 tmax engine 과의 서비스.

Example
```
<enable-extended-header>true</enable-extended-header>
```

(498) <jeus-system> <resource> <external-source> <tmax> **<inbuf-size>**

Description Tmax 으로부터 데이터 수신 시 한번에 읽어 들일 buffer size 단위를 설정할 수 있다.

Value Description bytes

Value Type int

Default Value 4096

Example
```
<inbuf-size>8192</inbuf-size>
```

(499) <jeus-system> <resource> <external-source> <tmax> **<outbuf-size>**

Description Tmax 으로부터 한번에 데이터 송신할 buffer size 단위를 설정한다

Value Description bytes

Value Type int

Default Value 4096

Example
```
<outbuf-size>8192</outbuf>
```

(500) <jeus-system> <resource> <external-source> <tmax> **<max-idle-timeout>**

Description connection pool 에서 여기서 설정된 maxIdleTime 동안 사용되지 않은 idle 상태의 connection 이 있을 경우 해당 connection 객체를 connection pool 에서 remove 한다. 최소한의 iniCapacity 값은 유지하도록 한다

Value Description millisecond

Value Type int

Default Value 60000

Example
```
<max-idle-timeout>30000</max-idle-timeout>
```

(501) <jeus-system> <resource> <external-source> <tmax> **<service-timeout>**

Description Tmax Server 로 service 요청하여 여기서 설정된 시간 내에 서비스가 이루어지지 않으면 해당 서비스 요청을 중지한다.

Value Description millisecond

Value Type int

Example
```
<max-idle-timeout>60000</max-idle-timeout>
```

(502) <jeus-system> <resource> <external-source> <tmax> **<transaction-timeout>**

Description Tmax Server 로 Transaction service 를 요청한 경우 여기서 설정된 시간 내에 서비스가 이루어지지 않으면 요청된 서비스를 중지한다

Value Description millisecond.

Value Type int

Example
```
<transaction-timeout>60000</transaction-timeout>
```

(503) <jeus-system> <resource> <external-source> <tmax> **<transaction-block-timeout>**

Description Tmax Server 로 Transaction commit service 를 요청한 경우 여기서 설정된 시간 내에 commit 서비스 정상적인 종료가 이루어지지 않으면 요청된 서비스를 중지한다.

Value Description millisecond.

Value Type int

Example
```
<transaction-block-timeout>60000</transaction-block-timeout>
```

(504) <jeus-system> <resource> <external-source> <tmax> **<security>**

Description 이 element 는 Tmax server 와 통신시 security 권한을 정의한다.

Example
```
<securityType> ... </securityType>
```

Child Elements

(505)user-name?
(506)user-password?
(507)domain-name?
(508)domain-password?

(505) <jeus-system> <resource> <external-source> <tmax> <security> **<user-name>**

Description Tmax server 로의 연결시 Tmax server 에 등록된 user 에 대한 인증을 위해 user name 을 설정한다

Value Description a user name

Value Type token

Example

```
<user-name>admin</user-name>
```

(506) <jeus-system> <resource> <external-source> <tmax> <security> **<user-password>**

Description Tmax server 로의 연결시 Tmax server 에 등록된 user 에 대한 인증을 위해 user password 을 설정한다.

Value Description a password string

Value Type token

Example

```
<user-password>tmax</user-password>
```

(507) <jeus-system> <resource> <external-source> <tmax> <security> **<domain-name>**

Description Tmax server 로의 연결시 Tmax server 에 등록된 domain 에 대한 인증을 위해 domain name 을 설정한다.

Value Description a domain name

Value Type token

Example

```
<domain-name>tmaxadm</domain-name>
```

(508) <jeus-system> <resource> <external-source> <tmax> <security> **<domain-password>**

Description Tmax server 로의 연결시 Tmax server 에 등록된 domain 에 대한 인증을 위해 domain password 을 설정한다

Value Description a domain password

Value Type token

Example `<domain-password>tmaxadm</domain-password>`

(509) <jeus-system> <resource> <external-source> <tmax> **<tmax-connection-pool>**

Description 이 element 는 Tmax server 와 연결 풀을 설정한다.

Example `<tmax-connection-pool> ... </tmax-connection-pool>`

Child Elements
(510)pooling?
(515)wait-free-connection?

(510) <jeus-system> <resource> <external-source> <tmax> <tmax-connection-pool> **<pooling>**

Description 풀 설정.

Child Elements
(511)min?
(512)max?
(513)step?
(514)period?

(511) <jeus-system> <resource> <external-source> <tmax> <tmax-connection-pool> <pooling> **<min>**

Description pooling 되는 객체의 최소값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 2

(512) <jeus-system> <resource> <external-source> <tmax> <tmax-connection-pool> <pooling> **<max>**

Description pooling 되는 객체의 최대값을 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 30

```
(513) <jeus-system> <resource> <external-source> <tmax> <tmax-
connection-pool> <pooling> <step>
```

Description pooling 되는 객체가 증가될때의 증가량을 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

Default Value 4

```
(514) <jeus-system> <resource> <external-source> <tmax> <tmax-
connection-pool> <pooling> <period>
```

Description pooling 되는 객체를 정리하는 시간을 지정한다.

Value Type long

Default Value 3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(515) <jeus-system> <resource> <external-source> <tmax> <tmax-
connection-pool> <wait-free-connection>
```

Description 이 element 는 connection 이 사용중 인 경우 다른 요청이 들어왔을 때 connection pool 을 다루는 방법을 설정한다.

Child Elements
```
(516)enable-wait?
(517)wait-time?
```

```
(516) <jeus-system> <resource> <external-source> <tmax> <tmax-
connection-pool> <wait-free-connection> <enable-wait>
```

Description 이 태그는 pool 안에 이용 가능한 connection 이 없거나 pool 안에 connection 들이 이미 최대값이 되어질 때 connection 을 요청 처리하는 방법을 결정한다. 만약 true 라면 시스템은 이용 가능한 connection 을 얻기위해 대기한다. 만약 false 라면, 시스템은 사용자 요청이 올 때 새로운 connection 을 만들고 사용이 끝난 이후에 pool 에 반납하지 않는다.

Value Type boolean

Default Value false

(517) <jeus-system> <resource> <external-source> <tmax> <tmax-connection-pool> <wait-free-connection> **<wait-time>**

Description 이 태그는 <enable-wait>가 true 일때만 유효하다. 이것은 사용자가 connection 을 위해 대기하는 시간을 나타낸다. 만약 어떠한 connection 도 사용자가 이 시간동안 대기하여도 이용할 수 없을 때는 시스템은 사용자에게 exception 을 던져준다.

Value Description millisecond 단위이다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 10000

(518) <jeus-system> <resource> <external-source> <tmax> **<tmax-property>**

Description 이 element 는 일반적은 속성 이외의 사용자가 Tmax 와의 통신 시 추가로 정의하고자 하는 속성 값을 정의한다.

Example
```
<tmax-property> ... </tmax-property>
```

Child Elements
```
(523)name
(524)value
```

(519) <jeus-system> <resource> <external-source> <tmax> <tmax-property> **<name>**

Description WebT Service 에서 사용자 속성을 추가하고자 할때의 Key 값.

Value Type token

Example
```
<name>eventHandler</name>
```

(520) <jeus-system> <resource> <external-source> <tmax> <tmax-property> **<value>**

Description WebT Service 에서 사용자 속성을 추가하고자 할때의 Value 값.

Value Type token

Example
```
<value>GenericServlet</value>
```

(521) <jeus-system> <resource> <external-source> <tmax-source> **<tmax-**

cluster>

Description 어플리케이션 서버 차원에서 데이터베이스의 FailOver 기능을 제공하기 위해서 데이터소스 클러스터링을 사용한다. 데이터소스 클러스터링은 근본적으로는 하나의 JNDI export name 을 가진 데이터소스 인스턴스이다

Child Elements
(522)export-name
(523)enable-load-balance?
(524)is-pre-conn?
(525)tmax-delegation-source*

(522) <jeus-system> <resource> <external-source> <tmax-source> <tmax-cluster> **<export-name>**

Description 데이터소스 클러스터링의 export name

Value Type token

(523) <jeus-system> <resource> <external-source> <tmax-source> <tmax-cluster> **<enable-load-balance>**

Description load balacne 기능의 사용 여부를 결정한다. true 이면 load balance 기능이 구동되고 false 이면 단순히 failover 기능이 구동된다.

Value Type boolean

Default Value true

(524) <jeus-system> <resource> <external-source> <tmax-source> <tmax-cluster> **<is-pre-conn>**

Description 이 기능이 사용되면, 데이터소스 클러스터링은 클러스터링 상의 모든 TMAX 에서 Connection 을 열게된다. 이것은 성능에는 좋지만 리소스의 절약면에서는 좋지 못하다. 만약 true 로 설정하면 총 생성된 Connection 의 개수가 <connection-pool> <pooling> <min>값과 동일하게 된다.

Value Type boolean

Default Value false

Performance Recommendation 성능 향상을 위해서는 “true”를, 리소스의 절약을 위해서는 “false”를 설정한다.

(525) <jeus-system> <resource> <external-source> <tmax-source> <tmax-cluster> **<tmax-delegation-source>**

Description 이 클러스터링에 참여할 TMAX 의 export name 을 정의한다.

Value Type token

Example
```
<tmax-delegation-source>webtdatasource1</tmax-delegation-source> <tmax-delegation-source>webtdatasource2</tmax-delegation-source>
```

(526) <jeus-system> <resource> **<jaxr-source>**

Description JAXR 어플리케이션에서 사용할 수 있는 xml-registry source 를 정의한다.

Child Elements (526)jaxr-entry*

(527) <jeus-system> <resource> <jaxr-source> **<jaxr-entry>**

Description JAXR Connection 를 구성하기 위한 ConnectionFactory 의 구성값을 담고 있다.

Child Elements
(527)export-name
(528)connection-factory-class-name?
(529)query-manager-URL?
(530)lifeCycle-manager-URL?
(531)authentication-method?
(532)jaxr-property*

(528) <jeus-system> <resource> <jaxr-source> <jaxr-entry> **<export-name>**

Description JAXR ConnectionFactory 의 JNDI 이름. 이 값은 Naming Server 에 ConnectionFactory 를 등록할 때 사용될 것이다.

Value Type token

(529) <jeus-system> <resource> <jaxr-source> <jaxr-entry> **<connection-factory-class-name>**

Description JAXR ConnectionFactory 클래스 이름.

Value Type token

Example
```
<connection-factory-class-name>jeus.webservices.registry.ConnectionFactoryImpl</connection-
```

factory-class-name>

(530) <jeus-system> <resource> <jaxr-source> <jaxr-entry> **<query-manager-URL>**

Description Target registry provider 의 query manager service 를 위한 URL. UDDI Registry 의 Inquiry URL.

Value Type token

(531) <jeus-system> <resource> <jaxr-source> <jaxr-entry> **<lifeCycle-manager-URL>**

Description Target registry provider 의 life cycle manager service 를 위한 URL. UDDI Registry 의 Publishing URL. 기술되지 않으면 queryManagerURL 과 같다.

Value Type token

(532) <jeus-system> <resource> <jaxr-source> <jaxr-entry> **<authentication-method>**

Description Registry Provider 의 인증을 얻기 위하여 사용하는 인증 방법.

Value Type token

Example
```
<authenticationMethod>UDDI_GET_AUTHTOKEN</authenticationMethod>
```

(533) <jeus-system> <resource> <jaxr-source> <jaxr-entry> **<jaxr-property>**

Description ConnectionFactory 를 구성하기 위한 Configuration properties 를 설정하기 위하여 사용된다.

Child Elements
(533)name
(534)value

(534) <jeus-system> <resource> <jaxr-source> <jaxr-entry> <jaxr-property> **<name>**

Description jaxr configuration property 의 이름. property 이름은 JAXR 1.0 스펙을 따라야 한다. [Standard Properties]: [javax.xml.registry.uddi.maxRows]: UDDI Provider 에서 find operation 에 대하여 return 하는 열의 최대값. [javax.xml.registry.postalAddressScheme]: 이 Connection 에 대하여 기본 postal address scheme 으로 사용되는 ClassificationScheme 의 id.

Value Type token

```
(535) <jeus-system> <resource> <jaxr-source> <jaxr-entry> <jaxr-
property> <value>
```

Description jaxr property 의 값.

Value Type token

```
(536) <jeus-system> <applications>
```

Description JEUS 가 실행될 때 deploy 가 되는 application 들을 모은 것이다. <application>과 다른 점은 여러 개의 application 에 공통적으로 적용되는 설정을 한 단위로 묶어서 적용할 수 있다는 것이다.

Child Elements

```
(536)absolute-path?
(537)absolute-ejb-jar?
(538)absolute-jeus-ejb-dd?
(539)auto-deploy?
(541)deployment-target?
(548)application+
```

```
(537) <jeus-system> <applications> <absolute-path>
```

Description 하위 application 들이 존재하는 절대경로를 지정할 수 있다.

Value Type token

```
(538) <jeus-system> <applications> <absolute-ejb-jar>
```

Description 하위 standalone EJB module 의 ejb-jar.xml 을 찾을 수 있는 절대 경로를 지정할 수 있다. 이는 4.x 의 EJB module 과의 호환성을 위한 것이다. 자세한 것은 JEUS Server 메뉴얼을 참고하기 바란다.

Value Type token

```
(539) <jeus-system> <applications> <absolute-jeus-ejb-dd>
```

Description 하위 standalone EJB module 의 jeus-ejb-dd.xml 을 찾을 수 있는 절대 경로를 지정할 수 있다. 이는 4.x 의 EJB module 과의 호환성을 위한 것이다. 자세한 것은 JEUS Manager 메뉴얼을 참고하기 바란다.

Value Type token

```
(540) <jeus-system> <applications> <auto-deploy>
```

Description 하위 application 들에게 공통적으로 auto-deploy 기능을 적용하기 위해 사용한다.

Child Elements (540)auto-deploy-check-interval?

(541) <jeus-system> <applications> <auto-deploy> **<auto-deploy-check-interval>**

Description application 이 변경되었는지 check 하는 주기를 설정할 수 있다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 10000

Performance Recommendation 너무 자주 check 하게 되면 성능저하가 생길 수 있으므로 필요한 간격만큼만 설정하도록 한다.

(542) <jeus-system> <applications> **<deployment-target>**

Description 하위 application 들의 공통 target 을 지정할 수 있다.

Child Elements (542)all-targets
(544)target

(543) <jeus-system> <applications> <deployment-target> **<all-targets>**

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements (543)context-group-name?

(544) <jeus-system> <applications> <deployment-target> <all-targets> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

(545) <jeus-system> <applications> <deployment-target> **<target>**

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements

(545)node-name
(546)engine-container-name
(547)context-group-name?

(546) <jeus-system> <applications> <deployment-target> <target> **<node-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example

```
<node-name>MyNode</node-name>
```

(547) <jeus-system> <applications> <deployment-target> <target> **<engine-container-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example

```
<engine-container-name>MyNode_container1</engine-container-name>
```

(548) <jeus-system> <applications> <deployment-target> <target> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(549) <jeus-system> <applications> **<application>**

Description <applications>의 설정을 적용받는 application 들을 지정한다.

Child Elements

(549)path?
(550)deployment-target?
(557)deployment-type
(558)auto-deploy?
(560)classloading?
(561)class-ftp-unit?

(562)security-domain-name?
(563)role-permission*
(570)java-security-permission?
(573)keep-generated?
(574)fast-deploy?
(575)client-component*
(584)connector-component*
(586)ejb-component*
(602)web-component*

(550) <jeus-system> <applications> <application> **<path>**

Description 이 application 의 path 를 설정한다. EAR, COMPONENT 공통적으로 archive file 일 경우에는 file path, directory 일 경우에는 directory path 가 사용된다. 절대 경로가 아니라 file name 이나 directory name 일 경우에는 APP_HOME 에 있다고 간주된다.

Value Type token

(551) <jeus-system> <applications> <application> **<deployment-target>**

Description 이 application 이 deploy 될 target 을 지정한다.

Child Elements
(551)all-targets
(553)target

(552) <jeus-system> <applications> <application> <deployment-target> **<all-targets>**

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements (552)context-group-name?

(553) <jeus-system> <applications> <application> <deployment-target> <all-targets> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

(554) <jeus-system> <applications> <application> <deployment-target>

<target>

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements
(554)node-name
(555)engine-container-name
(556)context-group-name?

(555) <jeus-system> <applications> <application> <deployment-target> <target> **<node-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example <node-name>MyNode</node-name>

(556) <jeus-system> <applications> <application> <deployment-target> <target> **<engine-container-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example <engine-container-name>MyNode_container1</engine-container-name>

(557) <jeus-system> <applications> <application> <deployment-target> <target> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(558) <jeus-system> <applications> <application> **<deployment-type>**

Description 이 application 이 deploy 되는 type 을 지정한다.

Value Type deployment-typeType

Defined Value EAR

EAR 형태의 archive file type 이다.

COMPONENT
standalone application (.jar, .war, .rar) 형태의 archive file type 이다.

(559) <jeus-system> <applications> <application> **<auto-deploy>**

Description 이 application 에 대해 auto-deploy 기능이 적용되도록 한다.

Child Elements (559)auto-deploy-check-interval?

(560) <jeus-system> <applications> <application> <auto-deploy> **<auto-deploy-check-interval>**

Description application 이 변경되었는지 check 하는 주기를 설정할 수 있다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 10000

Performance Recommendation 너무 자주 check 하게 되면 성능저하가 생길 수 있으므로 필요한 간격만큼만 설정하도록 한다.

(561) <jeus-system> <applications> <application> **<classloading>**

Description 이 application 이 사용할 classloading 방식을 선택한다. 지정하지 않으면 jeus.classloading system property 에 설정되어 있는 값을 사용한다.

Value Type classloadingType

Defined Value

ISOLATED
이 application 의 classloader 는 다른 application 의 classloader 와 분리되어서 서로의 class 들을 사용할수 없게 된다. J2EE spec 에 따른 application packaging 을 했을 때 사용할 수 있다.

SHARED
이 application 의 classloader 는 다른 application 의 classloader 와 같이 쓰여서 서로의 class 를 순서에 따라 공유할 수 있다. JEUS 4.x 이전 환경에서 개발한 application 의 경우에는 이 설정을 이용하여야 한다.

(562) <jeus-system> <applications> <application> **<class-ftp-unit>**

Description 이 application 에 포함된 EJB module 의 class 를 remote 로 전송할 때 JAR file 자체로 전송할지 한 class 씩 전송할지를 설정한다.

Value Type class-ftp-unitType

Default Value JAR

Defined Value JAR
이 application 내의 class 를 remote 로 보낼 때 JAR 파일 단위로 보낸다.

CLASS
이 application 내의 class 를 remote 로 보낼 때 class 파일 단위로 보낸다. JEUS 4.x 에서 DIR mode 로 EJB 를 개발했을때를 위해 사용하는 설정이다.

(563) <jeus-system> <applications> <application> **<security-domain-name>**

Description 이 application 에게 적용할 security domain 을 설정한다. 지정하지 않으면 SYSTEM_DOMAIN 을 사용하게 된다.

Value Type token

(564) <jeus-system> <applications> <application> **<role-permission>**

Description 이 application 의 모든 module 에게 적용할 principal - role mapping 을 설정할때 사용한다.

Child Elements
(564)principal+
(565)role
(566)actions?
(567)classname?
(568)excluded?
(569)unchecked?

(565) <jeus-system> <applications> <application> <role-permission> **<principal>**

Description role 에 해당하는 user principal 을 지정한다.

Value Description security 의 subjects.xml 에서 지정되어 있는 principal 이름

Value Type token

(566) <jeus-system> <applications> <application> <role-permission> **<role>**

Description principal 들에게 부여할 role 이름을 지정한다.

Value Type token

(567) <jeus-system> <applications> <application> <role-permission> **<actions>**

Description 이 role permission 객체에 대한 action 을 정의한다. default 로 사용되는 role permission 은 정해진 action 이 없다.

Value Type token

(568) <jeus-system> <applications> <application> <role-permission> **<classname>**

Description 사용할 role permission class name 을 지정한다. 지정하지 않으면 JEUS 에서 기본적으로 제공하는 class 가 사용된다.

Value Type token

(569) <jeus-system> <applications> <application> <role-permission> **<excluded>**

Description role 을 사용하지 못하도록 만든다.

Example

```
<excluded/>
```

(570) <jeus-system> <applications> <application> <role-permission> **<unchecked>**

Description 아무런 체크없이 role 을 사용가능하도록 만든다.

Example

```
<unchecked/>
```

(571) <jeus-system> <applications> <application> **<java-security-permission>**

Description JEUS 가 J2SE security 를 사용할 때 이 application 에게 허용할 J2SE permission 을 지정할 수 있다.

Child Elements

```
(571)description?
(572)security-permission-spec
```

(572) <jeus-system> <applications> <application> <java-security-permission> **<description>**

Description 이 security permission 설정에 대한 설명을 적을 수 있다.

Value Type token

(573) <jeus-system> <applications> <application> <java-security-permission> **<security-permission-spec>**

Description security permission 을 기술한다. 형식은 Java policy file 을 따른다.

Value Type token

Example

```
grant { permission java.lang.RuntimePermission
"foo"; }
```

(574) <jeus-system> <applications> <application> **<keep-generated>**

Description 이 applicaton 내의 module 들에 keep-generated 를 적용한다. 즉, 이 appliation 이 deploy 과정에서 generated 되어야 하는 클래스를 미리 포함하고 있다고 가정한다. 만약 설정이 되어 있지 않으면 jeus.application.keepgenerated system property 에 지정된 값이 사용된다.

Value Type boolean

(575) <jeus-system> <applications> <application> **<fast-deploy>**

Description 이 application 내의 EJB module, Web application 의 webservice module 에 대해 fast deploy 를 적용한다. 즉, 이 application 이 deploy 과정에서 generated 되어야 하는 클래스를 미리 포함하고 있다고 가정한다. 만약 설정이 되어 있지 않다면 engine 의 기본 설정을 따른다. 이 설정은 jeus.application.fastdeploy system property 를 설정한다.

Value Type boolean

(576) <jeus-system> <applications> <application> **<client-component>**

Description 이 application 내의 client-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements

```
(576)uri?
(577)deployment-target?
```

(577) <jeus-system> <applications> <application> <client-component> **<uri>**

Description 이 client component 설정에 해당하는 uri 이름이다.

Value Description EAR 이나 COMPONENT type 인 경우에는 .jar 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type token

Example `<uri>client.jar</uri>`

(578) <jeus-system> <applications> <application> <client-component> **<deployment-target>**

Description 이 client component 를 deploy 하고자 하는 target 을 설정할 수 있다.

Child Elements (578)all-targets
(580)target

(579) <jeus-system> <applications> <application> <client-component> <deployment-target> **<all-targets>**

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements (579)context-group-name?

(580) <jeus-system> <applications> <application> <client-component> <deployment-target> <all-targets> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

(581) <jeus-system> <applications> <application> <client-component> <deployment-target> **<target>**

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements (581)node-name
(582)engine-container-name
(583)context-group-name?

(582) <jeus-system> <applications> <application> <client-component> <deployment-target> <target> **<node-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example `<node-name>MyNode</node-name>`

(583) <jeus-system> <applications> <application> <client-component> <deployment-target> <target> **<engine-container-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example `<engine-container-name>MyNode_container1</engine-container-name>`

(584) <jeus-system> <applications> <application> <client-component> <deployment-target> <target> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(585) <jeus-system> <applications> <application> **<connector-component>**

Description 이 application 내의 connector-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements (585)uri?

(586) <jeus-system> <applications> <application> <connector-component> **<uri>**

Description 이 connector component 설정에 해당하는 uri 이름이다.

Value Description EAR 이나 COMPONENT type 인 경우에는 .rar 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type token

Example
```
<uri>connector.rar</uri>
```

```
(587) <jeus-system> <applications> <application> <ejb-component>
```

Description 이 application 내의 ejb-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements
```
(587)uri?
(588)deployment-target?
(595)client-view-path?
(596)keep-generated?
(597)ejb-jar?
(598)jeus-ejb-dd?
(599)java-security-permission?
```

```
(588) <jeus-system> <applications> <application> <ejb-component> <uri>
```

Description 이 ejb component 설정에 해당하는 uri 이름이다.

Value Description EAR 이나 COMPONENT type 인 경우에는 .jar 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type token

Example
```
<uri>ejb.jar</uri>
```

```
(589) <jeus-system> <applications> <application> <ejb-component>
<deployment-target>
```

Description 이 ejb component 를 deploy 할 target 을 지정한다.

Child Elements
```
(589)all-targets
(591)target
```

```
(590) <jeus-system> <applications> <application> <ejb-component>
<deployment-target> <all-targets>
```

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements
```
(590)context-group-name?
```

```
(591) <jeus-system> <applications> <application> <ejb-component>
<deployment-target> <all-targets> <context-group-name>
```

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group

이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

```
(592) <jeus-system> <applications> <application> <ejb-component>
<deployment-target> <target>
```

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements
```
(592)node-name
(593)engine-container-name
(594)context-group-name?
```

```
(593) <jeus-system> <applications> <application> <ejb-component>
<deployment-target> <target> <node-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example
```
<node-name>MyNode</node-name>
```

```
(594) <jeus-system> <applications> <application> <ejb-component>
<deployment-target> <target> <engine-container-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example
```
<engine-container-name>MyNode_container1</engine-
container-name>
```

```
(595) <jeus-system> <applications> <application> <ejb-component>
<deployment-target> <target> <context-group-name>
```

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(596) <jeus-system> <applications> <application> <ejb-component> **<client-view-path>**

Description 이 ejb component 의 stub class 들이 존재할 directory 를 설정한다.

Value Type token

(597) <jeus-system> <applications> <application> <ejb-component> **<keep-generated>**

Description 이 ejb component 의 stub class 들을 생성할때 java source file 을 남길지의 여부를 지정한다. 이 설정이 없으면 <application>의 설정을 사용한다.

Value Type boolean

(598) <jeus-system> <applications> <application> <ejb-component> **<ejb-jar>**

Description 이 ejb component 의 ejb-jar.xml file 의 path 를 설정할 수 있다. 이는 JEUS 4.x 의 packaging 을 지원하기 위해 사용한다.

Value Type token

Example `d:\jeus\config\node_ejb_engine1\ejb-jar_modulename.xml`

(599) <jeus-system> <applications> <application> <ejb-component> **<jeus-ejb-dd>**

Description 이 ejb component 의 jeus-ejb-dd.xml file 의 path 를 설정할 수 있다. 이는 JEUS 4.x 의 packaging 을 지원하기 위해 사용한다.

Value Type token

Example `d:\jeus\config\node_ejb_engine1\jeus-ejb-dd_modulename.xml`

(600) <jeus-system> <applications> <application> <ejb-component> **<java-security-permission>**

Description 이 ejb component 에게 부여할 J2SE security permission 을 설정할 수 있다.

Child Elements

(600)description?
(601)security-permission-spec

(601) <jeus-system> <applications> <application> <ejb-component> <java-security-permission> **<description>**

Description

이 security permission 설정에 대한 설명을 적을 수 있다.

Value Type

token

(602) <jeus-system> <applications> <application> <ejb-component> <java-security-permission> **<security-permission-spec>**

Description

security permission 을 기술한다. 형식은 Java policy file 을 따른다.

Value Type

token

Example

```
grant { permission java.lang.RuntimePermission
"foo"; }
```

(603) <jeus-system> <applications> <application> **<web-component>**

Description

이 application 내의 web-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements

(603)uri?
(604)deployment-target?
(611)keep-generated?
(612)jeus-web-dd?
(613)java-security-permission?

(604) <jeus-system> <applications> <application> <web-component> **<uri>**

Description

이 web component 설정에 해당하는 uri 이름이다.

Value Description

EAR 이나 COMPONENT type 인 경우에는 .war 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type

token

Example

```
<uri>web.war</uri>
```

(605) <jeus-system> <applications> <application> <web-component> **<deployment-target>**

Description

이 web component 를 deploy 할 target 을 지정한다.

Child Elements (605)all-targets
(607)target

```
(606) <jeus-system> <applications> <application> <web-component>
<deployment-target> <all-targets>
```

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements (606)context-group-name?

```
(607) <jeus-system> <applications> <application> <web-component>
<deployment-target> <all-targets> <context-group-name>
```

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

```
(608) <jeus-system> <applications> <application> <web-component>
<deployment-target> <target>
```

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements (608)node-name
(609)engine-container-name
(610)context-group-name?

```
(609) <jeus-system> <applications> <application> <web-component>
<deployment-target> <target> <node-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example <node-name>MyNode</node-name>

```
(610) <jeus-system> <applications> <application> <web-component>
<deployment-target> <target> <engine-container-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example `<engine-container-name>MyNode_container1</engine-container-name>`

(611) `<jeus-system> <applications> <application> <web-component> <deployment-target> <target>` **`<context-group-name>`**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(612) `<jeus-system> <applications> <application> <web-component>` **`<keep-generated>`**

Description 이 web component 가 웹서비스이거나 웹서비스 클라이언트를 포함하고 있을때 생성한 웹서비스의 Tie 및 웹서비스 클라이언트의 Stub 의 java source file 을 남길지의 여부를 지정한다. 이 설정이 없으면 <application>의 설정을 사용한다.

Value Type boolean

(613) `<jeus-system> <applications> <application> <web-component>` **`<jeus-web-dd>`**

Description 이 web component 의 jeus-web-dd.xml file 의 path 를 설정할 수 있다. 이는 JEUS 4.x 의 packaging 을 지원하기 위해 사용한다.

Value Type token

Example `d:\jeus\config\node_servlet_engine1\jeus-web-dd_modulename.xml`

(614) `<jeus-system> <applications> <application> <web-component>` **`<java-security-permission>`**

Description 이 web component 에게 부여할 J2SE security permission 을 설정할 수 있다.

Child Elements `(614)description?`
`(615)security-permission-spec`

(615) <jeus-system> <applications> <application> <web-component> <java-security-permission> **<description>**

Description 이 security permission 설정에 대한 설명을 적을 수 있다.

Value Type token

(616) <jeus-system> <applications> <application> <web-component> <java-security-permission> **<security-permission-spec>**

Description security permission 을 기술한다. 형식은 Java policy file 을 따른다.

Value Type token

Example
```
grant { permission java.lang.RuntimePermission
"foo"; }
```

(617) <jeus-system> **<application>**

Description JEUS 가 실행될 때 deploy 가 되는 application 을 지정한다.

Child Elements
(617)path?
(618)deployment-target?
(625)deployment-type
(626)auto-deploy?
(628)classloading?
(629)class-ftp-unit?
(630)security-domain-name?
(631)role-permission*
(638)java-security-permission?
(641)keep-generated?
(642)fast-deploy?
(643)client-component*
(652)connector-component*
(654)ejb-component*
(670)web-component*

(618) <jeus-system> <application> **<path>**

Description 이 application 의 path 를 설정한다. EAR, COMPONENT 공통적으로 archive file 일 경우에는 file path, directory 일 경우에는 directory path 가 사용된다. 절대 경로가 아니라 file name 이나 directory name 일 경우에는 APP_HOME 에 있다고 간주된다.

Value Type token

(619) <jeus-system> <application> **<deployment-target>**

Description 이 application 이 deploy 될 target 을 지정한다.

Child Elements (619)all-targets
(621)target

(620) <jeus-system> <application> <deployment-target> **<all-targets>**

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements (620)context-group-name?

(621) <jeus-system> <application> <deployment-target> <all-targets> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

(622) <jeus-system> <application> <deployment-target> **<target>**

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements (622)node-name
(623)engine-container-name
(624)context-group-name?

(623) <jeus-system> <application> <deployment-target> <target> **<node-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example <node-name>MyNode</node-name>

(624) <jeus-system> <application> <deployment-target> <target> **<engine-container-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example
```
<engine-container-name>MyNode_container1</engine-container-name>
```

(625) <jeus-system> <application> <deployment-target> <target> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(626) <jeus-system> <application> **<deployment-type>**

Description 이 application 이 deploy 되는 type 을 지정한다.

Value Type deployment-typeType

Defined Value
EAR
EAR 형태의 archive file type 이다.

COMPONENT
standalone application (.jar, .war, .rar) 형태의 archive file type 이다.

(627) <jeus-system> <application> **<auto-deploy>**

Description 이 application 에 대해 auto-deploy 기능이 적용되도록 한다.

Child Elements (627)auto-deploy-check-interval?

(628) <jeus-system> <application> <auto-deploy> **<auto-deploy-check-interval>**

Description application 이 변경되었는지 check 하는 주기를 설정할 수 있다.

Value Type nonNegativeLongType

Value Type Description 0 이상의 long type 이다. 즉, long 범위에서 0 이상의 값들을 포함한다.

Default Value 10000

Performance 너무 자주 check 하게 되면 성능저하가 생길 수 있으므로 필요한

Recommendation 간격만큼만 설정하도록 한다.

(629) <jeus-system> <application> **<classloading>**

Description 이 application 이 사용할 classloading 방식을 선택한다. 지정하지 않으면 jeus.classloading system property 에 설정되어 있는 값을 사용한다.

Value Type classloadingType

Defined Value ISOLATED
이 application 의 classloader 는 다른 application 의 classloader 와 분리되어서 서로의 class 들을 사용할수 없게 된다. J2EE spec 에 따른 application packaging 을 했을 때 사용할 수 있다.

SHARED
이 application 의 classloader 는 다른 application 의 classloader 와 같이 쓰여서 서로의 class 를 순서에 따라 공유할 수 있다. JEUS 4.x 이전 환경에서 개발한 application 의 경우에는 이 설정을 이용하여야 한다.

(630) <jeus-system> <application> **<class-ftp-unit>**

Description 이 application 에 포함된 EJB module 의 class 를 remote 로 전송할 때 JAR file 자체로 전송할지 한 class 씩 전송할지를 설정한다.

Value Type class-ftp-unitType

Default Value JAR

Defined Value JAR
이 application 내의 class 를 remote 로 보낼 때 JAR 파일 단위로 보낸다.

CLASS
이 application 내의 class 를 remote 로 보낼 때 class 파일 단위로 보낸다. JEUS 4.x 에서 DIR mode 로 EJB 를 개발했을때를 위해 사용하는 설정이다.

(631) <jeus-system> <application> **<security-domain-name>**

Description 이 application 에게 적용할 security domain 을 설정한다. 지정하지

않으면 SYSTEM_DOMAIN 을 사용하게 된다.

Value Type token

(632) <jeus-system> <application> **<role-permission>**

Description 이 application 의 모든 module 에게 적용할 principal - role mapping 을 설정할때 사용한다.

Child Elements
(632)principal+
(633)role
(634)actions?
(635)classname?
(636)excluded?
(637)unchecked?

(633) <jeus-system> <application> <role-permission> **<principal>**

Description role 에 해당하는 user principal 을 지정한다.

Value Description security 의 subjects.xml 에서 지정되어 있는 principal 이름

Value Type token

(634) <jeus-system> <application> <role-permission> **<role>**

Description principal 들에게 부여할 role 이름을 지정한다.

Value Type token

(635) <jeus-system> <application> <role-permission> **<actions>**

Description 이 role permission 객체에 대한 action 을 정의한다. default 로 사용되는 role permission 은 정해진 action 이 없다.

Value Type token

(636) <jeus-system> <application> <role-permission> **<classname>**

Description 사용할 role permission class name 을 지정한다. 지정하지 않으면 JEUS 에서 기본적으로 제공하는 class 가 사용된다.

Value Type token

(637) <jeus-system> <application> <role-permission> **<excluded>**

Description role 을 사용하지 못하도록 만든다.

Example `<excluded/>`

(638) <jeus-system> <application> <role-permission> **<unchecked>**

Description 아무런 체크없이 role 을 사용가능하도록 만든다.

Example `<unchecked/>`

(639) <jeus-system> <application> **<java-security-permission>**

Description JEUS 가 J2SE security 를 사용할 때 이 application 에게 허용할 J2SE permission 을 지정할 수 있다.

Child Elements (639)description?
(640)security-permission-spec

(640) <jeus-system> <application> <java-security-permission> **<description>**

Description 이 security permission 설정에 대한 설명을 적을 수 있다.

Value Type token

(641) <jeus-system> <application> <java-security-permission> **<security-permission-spec>**

Description security permission 을 기술한다. 형식은 Java policy file 을 따른다.

Value Type token

Example
```
grant { permission java.lang.RuntimePermission
"foo"; }
```

(642) <jeus-system> <application> **<keep-generated>**

Description 이 applicaton 내의 module 들에 keep-generated 를 적용한다. 즉, 이 appliation 이 deploy 과정에서 generated 되어야 하는 클래스를 미리 포함하고 있다고 가정한다. 만약 설정이 되어 있지 않으면 jeus.application.keepgenerated system property 에 지정된 값이 사용된다.

Value Type boolean

(643) <jeus-system> <application> **<fast-deploy>**

Description 이 application 내의 EJB module, Web application 의 webservice

module 에 대해 fast deploy 를 적용한다. 즉, 이 application 이 deploy 과정에서 generated 되어야 하는 클래스를 미리 포함하고 있다고 가정한다. 만약 설정이 되어 있지 않다면 engine 의 기본 설정을 따른다. 이 설정은 jeus.application.fastdeploy system property 를 설정한다.

Value Type boolean

(644) <jeus-system> <application> **<client-component>**

Description 이 application 내의 client-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements
(644)uri?
(645)deployment-target?

(645) <jeus-system> <application> <client-component> **<uri>**

Description 이 client component 설정에 해당하는 uri 이름이다.

Value Description EAR 이나 COMPONENT type 인 경우에는 .jar 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type token

Example
```
<uri>client.jar</uri>
```

(646) <jeus-system> <application> <client-component> **<deployment-target>**

Description 이 client component 를 deploy 하고자 하는 target 을 설정할 수 있다.

Child Elements
(646)all-targets
(648)target

(647) <jeus-system> <application> <client-component> <deployment-target> **<all-targets>**

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements
(647)context-group-name?

(648) <jeus-system> <application> <client-component> <deployment-target> <all-targets> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group

이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

```
(649) <jeus-system> <application> <client-component> <deployment-
target> <target>
```

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements
```
(649)node-name
(650)engine-container-name
(651)context-group-name?
```

```
(650) <jeus-system> <application> <client-component> <deployment-
target> <target> <node-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example
```
<node-name>MyNode</node-name>
```

```
(651) <jeus-system> <application> <client-component> <deployment-
target> <target> <engine-container-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example
```
<engine-container-name>MyNode_container1</engine-
container-name>
```

```
(652) <jeus-system> <application> <client-component> <deployment-
target> <target> <context-group-name>
```

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(653) <jeus-system> <application> **<connector-component>**

Description 이 application 내의 connector-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements (653)uri?

(654) <jeus-system> <application> <connector-component> **<uri>**

Description 이 connector component 설정에 해당하는 uri 이름이다.

Value Description EAR 이나 COMPONENT type 인 경우에는 .rar 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type token

Example
```
<uri>connector.rar</uri>
```

(655) <jeus-system> <application> **<ejb-component>**

Description 이 application 내의 ejb-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements
```
(655)uri?
(656)deployment-target?
(663)client-view-path?
(664)keep-generated?
(665)ejb-jar?
(666)jeus-ejb-dd?
(667)java-security-permission?
```

(656) <jeus-system> <application> <ejb-component> **<uri>**

Description 이 ejb component 설정에 해당하는 uri 이름이다.

Value Description EAR 이나 COMPONENT type 인 경우에는 .jar 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type token

Example
```
<uri>ejb.jar</uri>
```

(657) <jeus-system> <application> <ejb-component> **<deployment-target>**

Description 이 ejb component 를 deploy 할 target 을 지정한다.

Child Elements (657)all-targets
(659)target

```
(658) <jeus-system> <application> <ejb-component> <deployment-target>
<all-targets>
```

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements (658)context-group-name?

```
(659) <jeus-system> <application> <ejb-component> <deployment-target>
<all-targets> <context-group-name>
```

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

```
(660) <jeus-system> <application> <ejb-component> <deployment-target>
<target>
```

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements (660)node-name
(661)engine-container-name
(662)context-group-name?

```
(661) <jeus-system> <application> <ejb-component> <deployment-target>
<target> <node-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example <node-name>MyNode</node-name>

```
(662) <jeus-system> <application> <ejb-component> <deployment-target>
<target> <engine-container-name>
```

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example `<engine-container-name>MyNode_container1</engine-container-name>`

(663) <jeus-system> <application> <ejb-component> <deployment-target> <target> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(664) <jeus-system> <application> <ejb-component> **<client-view-path>**

Description 이 ejb component 의 stub class 들이 존재할 directory 를 설정한다.

Value Type token

(665) <jeus-system> <application> <ejb-component> **<keep-generated>**

Description 이 ejb component 의 stub class 들을 생성할때 java source file 을 남길지의 여부를 지정한다. 이 설정이 없으면 <application>의 설정을 사용한다.

Value Type boolean

(666) <jeus-system> <application> <ejb-component> **<ejb-jar>**

Description 이 ejb component 의 ejb-jar.xml file 의 path 를 설정할 수 있다. 이는 JEUS 4.x 의 packaging 을 지원하기 위해 사용한다.

Value Type token

Example `d:\jeus\config\node_ejb_engine1\ejb-jar_modulename.xml`

(667) <jeus-system> <application> <ejb-component> **<jeus-ejb-dd>**

Description 이 ejb component 의 jeus-ejb-dd.xml file 의 path 를 설정할 수 있다. 이는 JEUS 4.x 의 packaging 을 지원하기 위해 사용한다.

Value Type token

Example `d:\jeus\config\node_ejb_engine1\jeus-ejb-dd_modulename.xml`

`(668) <jeus-system> <application> <ejb-component>` **`<java-security-permission>`**

Description 이 ejb component 에게 부여할 J2SE security permission 을 설정할 수 있다.

Child Elements
`(668)description?`
`(669)security-permission-spec`

`(669) <jeus-system> <application> <ejb-component> <java-security-permission>` **`<description>`**

Description 이 security permission 설정에 대한 설명을 적을 수 있다.

Value Type token

`(670) <jeus-system> <application> <ejb-component> <java-security-permission>` **`<security-permission-spec>`**

Description security permission 을 기술한다. 형식은 Java policy file 을 따른다.

Value Type token

Example
```
grant { permission java.lang.RuntimePermission
"foo"; }
```

`(671) <jeus-system> <application>` **`<web-component>`**

Description 이 application 내의 web-component 에 대한 특별한 설정을 하고 싶을때 사용한다.

Child Elements
`(671)uri?`
`(672)deployment-target?`
`(679)keep-generated?`
`(680)jeus-web-dd?`
`(681)java-security-permission?`

`(672) <jeus-system> <application> <web-component>` **`<uri>`**

Description 이 web component 설정에 해당하는 uri 이름이다.

Value Description EAR 이나 COMPONENT type 인 경우에는 .war 로 끝나고 EXPLODED 형태인 경우 해당 directory 이름이 온다.

Value Type token

Example `<uri>web.war</uri>`

`(673) <jeus-system> <application> <web-component> <deployment-target>`

Description 이 web component 를 deploy 할 target 을 지정한다.

Child Elements `(673)all-targets`
`(675)target`

`(674) <jeus-system> <application> <web-component> <deployment-target> <all-targets>`

Description 이 설정을 읽는 모든 engine container 가 target 에 해당된다.

Child Elements `(674)context-group-name?`

`(675) <jeus-system> <application> <web-component> <deployment-target> <all-targets> <context-group-name>`

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description <all-target>에 대해 사용되므로 application 을 deploy 하는 모든 Servlet Engine 이 이 context group 을 가지고 있어야 한다.

Value Type token

`(676) <jeus-system> <application> <web-component> <deployment-target> <target>`

Description 이 application 을 deploy 할 target 을 설정한다.

Child Elements `(676)node-name`
`(677)engine-container-name`
`(678)context-group-name?`

`(677) <jeus-system> <application> <web-component> <deployment-target> <target> <node-name>`

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 node 내의 engine container 들로 지정한다.

Value Type token

Example `<node-name>MyNode</node-name>`

(678) <jeus-system> <application> <web-component> <deployment-target> <target> **<engine-container-name>**

Description 이 application 을 deploy 할 target 을 여기에 지정된 이름의 engine container 로 지정한다.

Value Type token

Example
```
<engine-container-name>MyNode_container1</engine-container-name>
```

(679) <jeus-system> <application> <web-component> <deployment-target> <target> **<context-group-name>**

Description 이 application 내의 web module 이 deploy 될 때 사용될 context group 이름을 지정한다.

Value Description 이 element 를 가진 target 의 servlet engine 은 모두 이 이름의 context group 을 가지고 있어야 한다.

Value Type token

(680) <jeus-system> <application> <web-component> **<keep-generated>**

Description 이 web component 가 웹서비스이거나 웹서비스 클라이언트를 포함하고 있을때 생성한 웹서비스의 Tie 및 웹서비스 클라이언트의 Stub 의 java source file 을 남길지의 여부를 지정한다. 이 설정이 없으면 <application>의 설정을 사용한다.

Value Type boolean

(681) <jeus-system> <application> <web-component> **<jeus-web-dd>**

Description 이 web component 의 jeus-web-dd.xml file 의 path 를 설정할 수 있다. 이는 JEUS 4.x 의 packaging 을 지원하기 위해 사용한다.

Value Type token

Example
```
d:\jeus\config\node_servlet_engine1\jeus-web-dd_modulename.xml
```

(682) <jeus-system> <application> <web-component> **<java-security-permission>**

Description 이 web component 에게 부여할 J2SE security permission 을 설정할 수 있다.

Child Elements (682)description?
(683)security-permission-spec

(683) <jeus-system> <application> <web-component> <java-security-permission> **<description>**

Description 이 security permission 설정에 대한 설명을 적을 수 있다.

Value Type token

(684) <jeus-system> <application> <web-component> <java-security-permission> **<security-permission-spec>**

Description security permission 을 기술한다. 형식은 Java policy file 을 따른다.

Value Type token

Example
```
grant { permission java.lang.RuntimePermission
"foo"; }
```

J.4 JEUSMain.xml 예제

<<JEUSMain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus" >
  <node>
    <name>johan</name>
    <listener>
      <backlog>128</backlog>
      <ssl>
        <port>9727</port>
      </ssl>
      <thread-pool>
        <min>2</min>
        <max>30</max>
        <period>3600000</period>
      </thread-pool>
    </listener>
    <backup-node>star</backup-node>
    <engine-container>
```

```
      <name>mycontainer</name>
      <command-option>>-Xms64m -Xmx128m</command-option>
      <engine-command>
        <type>ejb</type>
        <name>engine1</name>
        <system-logging>
          <level>INFO</level>
          <use-parent-handlers>false</use-parent-handlers>
          <filter-class>com.tmax.logging.filter.MyFilter</filter-
class>
          <handler>
            <console-handler>
              <name>consoleHandler</name>
              <level>FINEST</level>
              <encoding>EUC-KR</encoding>
              <filter-
class>com.tmax.logging.filter.MyFilte2</filter-class>
            </console-handler>
          </handler>
        </system-logging>
      </engine-command>
      <enable-interop>
        <use-OTS>false</use-OTS>
      </enable-interop>
      <sequential-start>false</sequential-start>
      <user-class-
path>c:\mylib\classes;c:\mylib\lib\mylib.jar</user-class-path>
      <tm-config>
        <use-nio>true</use-nio>
        <pooling>
          <min>2</min>
          <max>30</max>
          <period>3600000</period>
        </pooling>
        <active-timeout>600000</active-timeout>
        <prepare-timeout>120000</prepare-timeout>
        <prepared-timeout>60000</prepared-timeout>
        <commit-timeout>240000</commit-timeout>
        <recovery-timeout>120000</recovery-timeout>
```

```
        <uncompleted-timeout>86400000</uncompleted-timeout>
        <capacity>10000</capacity>
        <heuristic-rollback>false</heuristic-rollback>
        <resolution>60000</resolution>
      </tm-config>
      <scheduler>
        <default-lookup-name>test_scheduler</default-lookup-name>
        <thread-pool>
          <min>2</min>
          <max>30</max>
          <period>3600000</period>
        </thread-pool>
        <job-list>
          <job>
            <class-name>ClientSchedule</class-name>
            <name>test client schedule</name>
            <description>test</description>
            <begin-time>2005-02-01T00:00:00.001</begin-time>
            <end-time>2005-02-01T00:01:00.001</end-time>
            <interval>
              <millisecond>5000</millisecond>
            </interval>
            <count>5</count>
          </job>
        </job-list>
      </scheduler>
      <user-logging>
        <level>INFO</level>
        <use-parent-handlers>false</use-parent-handlers>
        <filter-class>com.tmax.logging.filter.MyFilter3</filter-
class>
        <handler>
          <console-handler>
            <name>consoleHandler1</name>
            <level>FINEST</level>
            <encoding>EUC-KR</encoding>
            <filter-
class>com.tmax.logging.filter.MyFilter4</filter-class>
          </console-handler>
```

```
        </handler>
      </user-logging>
      <system-logging>
        <level>INFO</level>
        <use-parent-handlers>false</use-parent-handlers>
        <filter-class>com.tmax.logging.filter.MyFilter5</filter-
class>
        <handler>
          <console-handler>
            <name>consolHandler2</name>
            <level>FINEST</level>
            <encoding>EUC-KR</encoding>
            <filter-
class>com.tmax.logging.filter.MyFilter6</filter-class>
          </console-handler>
        </handler>
      </system-logging>
      <invocation-manager-action>NoAction</invocation-manager-
action>
      <jmx-manager>
        <jmx-connector>
          <jmxmp-connector>
            <jmxmp-connector-port>5001</jmxmp-connector-port>
          </jmxmp-connector>
        </jmx-connector>
        <html-adaptor-port>7070</html-adaptor-port>
        <snmp-adaptor>
          <snmp-adaptor-port>9090</snmp-adaptor-port>
          <snmp-version>3</snmp-version>
          <snmp-max-packet-size>4096</snmp-max-packet-size>
          <snmp-security>true</snmp-security>
          <trap-demon>
            <ip-address>111.111.111.1</ip-address>
            <port>11</port>
          </trap-demon>
          <pooling>
            <min>2</min>
            <max>30</max>
            <period>3600000</period>
```

```
          </pooling>
        </snmp-adaptor>
      </jmx-manager>
      <auto-deploy>
        <enable-auto-deploy>true</enable-auto-deploy>
        <auto-deploy-path>c:\Jeus\webhome\deploy_home</auto-
deploy-path>
        <auto-deploy-check-interval>2000</auto-deploy-check-
interval>
      </auto-deploy>
      <use-MEJB>false</use-MEJB>
      <lifecycle-invocation>
        <class-name>com.tmax.ProcessClass</class-name>
        <invocation>
          <invocation-method>
            <method-name>processMethod</method-name>
            <method-params>
              <method-param>int</method-param>
            </method-params>
          </invocation-method>
          <invocation-argument>1</invocation-argument>
          <invocation-type>BOOT</invocation-type>
        </invocation>
      </lifecycle-invocation>
      <application-path>webhome/app_home</application-path>
      <res-ref>
        <jndi-info>
          <ref-name>A</ref-name>
          <export-name>B</export-name>
        </jndi-info>
      </res-ref>
    </engine-container>
    <class-ftp>true</class-ftp>
    <security-switch>true</security-switch>
    <sequential-start>true</sequential-start>
    <enable-jnlp>false</enable-jnlp>
    <enable-webadmin>false</enable-webadmin>
    <system-logging>
      <level>INFO</level>
```

```
    <handler>
      <file-handler>
     <name>fileHandler</name>
     <level>FINE</level>
     <valid-hour>1</valid-hour>
   </file-handler>
    </handler>
  </system-logging>
  <session-server>
    <resolution>60000</resolution>
    <backlog-size>50</backlog-size>
    <session-manager>
      <name>session1</name>
      <multi-session>primary</multi-session>
      <passivation-to>1800000</passivation-to>
      <removal-to>3600000</removal-to>
      <file-db-path>c:\sessiondata</file-db-path>
      <file-db-name>sessiondata.fdb</file-db-name>
      <min-hole>1000</min-hole>
      <packing-rate>0.5</packing-rate>
      <check-to>30000</check-to>
      <backup-name>session2</backup-name>
      <backup-trigger>1000</backup-trigger>
      <operation-to>30000</operation-to>
    </session-manager>
  </session-server>
  <jmx-manager>
      <jmx-connector>
        <jmxmp-connector>
          <jmxmp-connector-port>5001</jmxmp-connector-port>
        </jmxmp-connector>
      </jmx-connector>
      <html-adaptor-port>7070</html-adaptor-port>
      <snmp-adaptor>
        <snmp-adaptor-port>9090</snmp-adaptor-port>
        <snmp-version>3</snmp-version>
        <snmp-max-packet-size>4096</snmp-max-packet-size>
        <snmp-security>true</snmp-security>
        <trap-demon>
```

```
            <ip-address>111.111.111.1</ip-address>
            <port>11</port>
          </trap-demon>
          <pooling>
            <min>2</min>
            <max>30</max>
            <period>3600000</period>
          </pooling>
        </snmp-adaptor>
      </jmx-manager>
</node>
<naming-server>
  <server>
    <use-nio>true</use-nio>
    <export-cos-naming>false</export-cos-naming>
    <backlog-size>50</backlog-size>
    <resolution>60000</resolution>
    <buffer-size>20480</buffer-size>
    <pooling>
      <min>2</min>
      <max>30</max>
      <period>3600000</period>
    </pooling>
  </server>
  <local>
    <resolution>60000</resolution>
    <buffer-size>20480</buffer-size>
    <pooling>
      <min>2</min>
      <max>30</max>
      <period>3600000</period>
    </pooling>
  </local>
</naming-server>
<security-manager>
  <domain-name>SYSTEM_DOMAIN</domain-name>
</security-manager>
<resource>
  <data-source>
```

```
      <database>
        <vendor>oracle</vendor>
        <export-name>datasource1</export-name>
        <data-source-class-
name>oracle.jdbc.pool.OracleConnectionPoolDataSource</data-
source-class-name>
        <data-source-type>ConnectionPoolDataSource</data-source-
type>
        <database-name>MYDB</database-name>
        <data-source-
name>oracle.jdbc.pool.OracleDataSource</data-source-name>
        <service-name>oradb1</service-name>
        <description>이 DataSource 는 Oracle
ConnectionPoolDataSource 를 사용한다.</description>
        <encryption>true</encryption>
        <port-number>1521</port-number>
        <server-name>192.168.1.2</server-name>
        <user>scott</user>
        <password>tiger</password>
        <driver-type>thin</driver-type>
        <property>
       <name>PortNumber</name>
       <type>java.lang.Integer</type>
       <value>1521</value>
    </property>
        <connection-pool>
          <pooling>
            <min>2</min>
            <max>30</max>
            <step>4</step>
            <period>3600000</period>
          </pooling>
          <wait-free-connection>
            <enable-wait>false</enable-wait>
            <wait-time>10000</wait-time>
          </wait-free-connection>
          <operation-to>30000</operation-to>
          <delegation-datasource>delegate1</delegation-
datasource>
```

```
          <max-use-count>3</max-use-count>
          <dba-timeout>30000</dba-timeout>
          <check-query>SELECT check FROM checktable;</check-
query>
          <stmt-caching-size>10</stmt-caching-size>
          <stmt-fetch-size>10</stmt-fetch-size>
        </connection-pool>
      </database>
      <cluster-ds>
        <export-name>DSCluster</export-name>
        <is-pre-conn>true</is-pre-conn>
        <backup-data-source>datasource2</backup-data-source>
      </cluster-ds>
    </data-source>
    <mail-source>
      <mail-entry>
        <export-name>HOST</export-name>
        <mail-property>
           <name>mail.host</name>
       <value>mail.foo.com</value>
        </mail-property>
      </mail-entry>
    </mail-source>
    <url-source>
      <url-entry>
        <export-name>MYURL</export-name>
        <url>http://www.foo.com</url>
      </url-entry>
    </url-source>
    <external-source>
      <jms-source>
    <vendor>ibmmq</vendor>
    <factory-class-name>com.ibm.mq.jms.MQQueue</factory-class-
name>
    <resource-type>Q</resource-type>
    <export-name>myMQ</export-name>
    <queue>queue</queue>
    <queueManager>qmgr</queueManager>
    <property>
```

```
      <name>ccsid</name>
      <type>java.lang.String</type>
      <value>myqid</value>
    </property>
    <property>
      <name>persistence</name>
      <type>java.lang.String</type>
      <value>APP</value>
    </property>
    <property>
      <name>targ-client</name>
      <type>java.lang.String</type>
      <value>MQ</value>
    </property>
    <property>
      <name>encoding</name>
      <type>java.lang.String</type>
      <value>euc-kr</value>
    </property>
  </jms-source>
    <tmax>
    <export-name>tmax1</export-name>
    <webt-datasource-type>WebtConnectionPoolDataSource</webt-
datasource-type> <!-- WebtXADataSource -->
    <host-name>111.111.111.1</host-name>
    <port>8888</port>
    <backup-host-name>111.111.111.2</backup-host-name>
    <backup-port>8889</backup-port>
    <fdl-file>c:/tmax/fdl/tmax.fdl</fdl-file>
    <default-charset>euc-kr</default-charset>
    <enable-pipe>false</enable-pipe>
    <enable-extended-header>false</enable-extended-header>
    <inbuf-size>4096</inbuf-size>
    <outbuf-size>4096</outbuf-size>
    <max-idle-timeout>60000</max-idle-timeout>
    <service-timeout>120000</service-timeout>
    <transaction-timeout>100000</transaction-timeout>
    <transaction-block-timeout>30000</transaction-block-timeout>
    <security>
```

```
      <user-name>user</user-name>
      <user-password>1234</user-password>
      <domain-name>tmax</domain-name>
      <domain-password>4321</domain-name>
    </security>
    <tmax-connection-pool>
      <pooling>
        <min>10</min>
        <max>20</max>
        <step>2</step>
        <period>2</period>
      </pooling>
      <wait-free-connection>
        <enable-wait>true</enable-wait>
        <wait-time>60000</wait-time>
      </wait-free-connection>
    </tmax-connection-pool>
    <webt-logging>
      <level>debug</level>
    </webt-logging>
    <monitoring>
      <enable-check-alive>true</enable-check-alive>
      <enable-check-idle>true</enable-check-idle>
      <check-pool-interval>3000</check-pool-interval>
    </monitoring>
    <tmax-property>
      <name>eventSvcType</name>
      <value>all</value>
    </tmax-property>
    <tmax-property>
      <name>eventHandler</name>
      <value>GenericServlet</value>
    </tmax-property>
    </tmax>
    </external-source>
    <jaxr-source>
      <jaxr-entry>
        <export-name>JAXRConnection</export-name>
```

```
        <connection-factory-class-
name>jeus.webservies.ConnectionFactoryImpl</connection-factory-
class-name>
        <query-manager-
URL>http://localhost:8088/uddi/inquiry</query-manager-URL>
        <lifeCycle-manager-
URL>http://localhost:8088/uddi/publish</lifeCycle-manager-URL>
        <authentication-
method>UDDI_GET_AUTHTOKEN</authentication-method>
        <jaxr-property>
          <name>javax.xml.registry.uddi.maxRows</name>
          <value>10</value>
        </jaxr-property>
      </jaxr-entry>
    </jaxr-source>
  </resource>
  <applications>
    <absolute-path>c:\myapps\MyJ2eeApp.ear</absolute-path>
    <auto-deploy>
      <auto-deploy-check-interval>10000</auto-deploy-check-
interval>
    </auto-deploy>
    <deployment-target>
      <all-targets>
        <context-group-name>token</context-group-name>
      </all-targets>
    </deployment-target>
    <application>
      <path>token</path>
      <deployment-target>
        <all-targets/>
      </deployment-target>
      <deployment-type>EAR</deployment-type>
      <auto-deploy>
        <auto-deploy-check-interval>10000</auto-deploy-check-
interval>
      </auto-deploy>
      <classloading>ISOLATED</classloading>
      <class-ftp-unit>JAR</class-ftp-unit>
```

```
      <java-security-permission>
        <description>token</description>
        <security-permission-spec>token</security-permission-
spec>
      </java-security-permission>
      <keep-generated>true</keep-generated>
      <fast-deploy>false</fast-deploy>
      <ejb-component>
        <uri>myejbs/ejbModuleA.jar</uri>
        <deployment-target>
          <all-targets/>
        </deployment-target>
        <client-view-path>c:\myapps</client-view-path>
        <keep-generated>true</keep-generated>
      </ejb-component>
    </application>
  </applications>
</jeus-system>
```

K. JEUS Server Ant Task Reference

K.1 소개

어플리케이션을 개발하면서 특정 작업을 자동하기 위해, Ant 의 task 를 제공한다. Task 에는 다음과 같은 것이 있다.

- jeusstart
- boot
- down

Ant 에 대한 설정법과 사용법은 http://ant.apache.org 를 참조하기 바란다. 그리고 JEUS EJB 에 대한 Ant Task 는 JEUS EJB 안내서를 참조한다.

K.2 jeusstart

Ant Task 인 'jeusstart' 의 목적은 Ant 스크립트에서 JEUS Manager 를 실행시키는 것이다. 'jeustart' Task 의 속성은 다음 표에서 설명했다.

표 9. jeusstart Ant task 의 속성

속성	설명	필수
`classpath`	JEUS 를 실행할 때 필요한 classpapth 로 JEUS_HOME\lib\system\bootstrap.jar 를 세팅한다.	Yes
`libraryPath`	JEUS 에서 사용하는 Library 디렉토리로 JEUS_HOME\lib\system 를 세팅한다.	Yes
`jeusHome`	JEUS 의 홈 디렉토리	Yes

속성	설명	필수
appHome	Application 의 홈 디렉토리	Yes
servletHome	Servlet 의 홈 디렉토리.	Yes
jeusBaseport	JEUS 의 base port	No default 9736
tmCheckReg	-Djeus.tm.checkReg 의 값	Yes
classLoadingMode	Classloader 의 모드로 'SHARED'과 'ISOLATED' 중 하나를 입력한다.	No default SHARED
xml	JEUS 설정 파일로 'true'이면 .xml 파일을 사용하고 'false'이면 .conf 파일을 사용한다.	No default true

예제)

<<*Ant build script*>>

```
<jeusstart classpath="${jeus.bootclasspath}"
      libraryPath="${jeus.libpath}"
            jeusHome="${jeus.jeushome}"
            appHome="${jeus.apphome}"
            servletHome="${jeus.servlethome}"
            jeusBaseport="${jeus.baseport}"
            tmCheckReg="${jeus.tmcheckreg}"
            classLoadingMode="${jeus.classloadingmode}"
            xml="${jeus.xml}" />
```

K.3 boot

"boot" Ant Task 는 JEUS 를 부팅한다. 이 Task 의 속성은 다음 표에서 설명한다.

표 10. 'boot' Ant task 속성

속성	설명	필수
node	JEUS 의 노드명	Yes
dynamic	'true'이면 dynamic 모드로 JEUS 를 부팅한다.	No default false
isXML	'true'이면 EJB 설정을 .xml 파일에서 읽어오고, 'false'이면 .conf 파일에서 읽어온다.	No default true
username	부팅할 수 있는 권한을 가진 사용자명	Yes
password	위 사용자의 패스워드	Yes

예제)

<<Ant build script>>

```
<boot node="${jeus.nodename}" dynamic="${jeus.boot.dynamic}"
isXML="${jeus.xml}" username="${jeus.username}"
password="${jeus.password}"/>
```

K.4 down

JEUS 를 다운시킨다.

Task 의 속성은 다음 표를 본다.

표 11. 'down' Ant task 속성

속성	설명	필수
node	JEUS 의 노드명	Yes

속성	설명	필수
jeusexit	'true'이면 'jeusexit' 명령이 자동으로 실행된다.	No default false
username	JEUS 를 다운시킬 수 있는 권한을 가진 사용자명	Yes
password	위 사용자의 패스워드	Yes

예제)

<<*Ant build script*>>

```
<down node="${jeus.nodename}" jeusexit="${jeus.down.jeusexit}"
username="${jeus.username}" password="${jeus.password}"/>
```

L. standard.property

L.1 소개

Jeus-ejb-dd.xml, jeus-web-dd.xml 없이 J2EE 어플리케이션을 deploy 할 때, 기본 설정으로 사용되는 항목 값을 정의한다. 이 항목은 jeus-ejb-dd.xml 과 jeus-web-dd.xml 의 항목을 하나로 정리한 것이다. 각 항목의 자세한 내용은 JEUS EJB 안내서나 WebContainer 안내서를 참고한다.

기본적으로 JEUS_HOME\config\<node name>에 위치한다.

L.2 프로퍼티 레퍼런스

프로퍼티	설명	기본값
vendor	DB 벤더명	N/A
datasource-name	CMP 에서 사용할 데이터소스	N/A
creating-table	CMP 의 테이블 생성 여부	false
deleting-table	CMP 의 테이블 삭제 여부	false
engine-type	Entity Bean 의 엔진 타입	EXCLUSIVE_ACCESS
subengine-type	CMP 의 서브 엔진타입	ReadLocking
fetch-size	ResultSet 으로 한 번에 가져올 수 있는 레코드의 수	10
enable-instant-ql	JEUS 의 Instance QL 사용 여부	false

프로퍼티	설명	기본값
local-optimize	Local Invoke Optimize 사용 여부	true
logging-level	로그 레벨	fatal
export-port	RMI Listener Port	0
export-iiop	COS Naming 사용 여부	false
single-vm-only	해당 컨테이너의 JNDI 에만 객체를 bind 한다.	false
thread-pool-min	최소 tread 개수	20
thread-pool-max	최대 thread 개수	100
thread-pool-step	thread 증가 개수	20
thread-pool-resize	thread 개수 조정 시간 간격	3600000
bean-pool-min	EJB bean 의 최소 개수	0
bean-pool-max	EJB bean 의 최대 개수	100
bean-pool-resize	EJB bean 개수 조정 시간 간격	3600000
capacity	EJB bean 인스턴스의 capacity	10000
passivation-timeout	EJB 가 Passivate 되는 시간	-1
disconnect-timeout	EJB 의 인스턴스가 완전히 사라지는 시간	-1
connect-pool-min	EJB 커넥션 풀의 최소 개수	0

프로퍼티	설명	기본값
connect-pool-max	EJB 커넥션 풀의 최대 개수	100
connect-pool-resize	EJB 커넥션 개수의 조정 시간 간격	3600000
jms-connection	JMS connection 의 export name	N/A
mail-connection	Mail 리소스의 export name	N/A
url-connection	URL 리소스의 export name	N/A
init-caching	Entity Bean 을 미리 초기화할 지 여부	false

색 인

D

E

F

G

L

M

N

O

P

R

S

T

U

V

W

X