JEUS 시작하기

Tmax Soft

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

차 례

그림 목차

표 목차

매뉴얼에 대하여

매뉴얼의 대상

이 책은 JEUS 를 처음 설치하여, EJB 와 Servlet/JSP 프로그래밍을 하려는 사람들에게 필요한 정보를 담고 있다.

이 책은 자세한 설명보다는 JEUS 의 가장 중점적인 기능들을 중심으로 설명함으로써, 보다 쉽게 JEUS 에 친숙해 지도록 하였다. 따라서 필수적인 사항을 단계적으로 구성하는 자습서의 형식을 띄고 있다.

매뉴얼의 전제조건

Web Application Server 와 J2EE 기술(EJB/Servlet)에 대한 기본적인 개념을 이해하고 있어야 한다

본 매뉴얼을 읽기 전에 JEUS 소개와 JEUS 설치 안내서를 먼저 읽기 바란다. 이 문서는 JEUS 가 이미 올바르게 설치 되어 있음을 가정하고 설명한다.

매뉴얼의 구성

이 책의 각 장은 처음 JEUS 를 사용하여 EJB, Server/JSP 프로그래밍을 하기 위해 필요한 단계들로 구성되며 다음 5 개의 주요 파트로 나누어져 있다

1. 소개: 본 매뉴얼의 목적과 다른 장들과의 차이점 및 이용 방법에 대해 서술한다.

2. **JEUS System** 설정하기: EJB 와 Servlet/JSP 를 사용하기 위해웹 관리자를 이용하여 JEUS 환경 파일을 설정하는 방법과 수동으로 설정하는 방법을 서술한다. 또한 JEUS 를 시작 및 종료 시키는 방법을 서술한다.

3. **EJB 사용하기**: JEUSBuilder 와 웹 관리자를 이용하여 Stateless session bean 과 CMP 2.0 entity bean 을 Packaging, Deploy 하는 방법과 테스트하는 방법에 대해 설명되어 있다.

4. **Servlet/JSP** 사용하기: JEUSBuilder 와 웹 관리자를 이용하여 Web application 을 Packaging, Deploy 하는 법과 테스트하는 방법에 대해 설명되어 있다.

5. **결론:** 이 장은 JEUS 시작하기 문서에 담겨있는 내용을 요약해서 제공하며, 다음의 방향을 제시한다.

관련 매뉴얼

이 문서를 읽기 전에 JEUS 소개와 JEUS 설치 안내서를 먼저 읽는 것이 바람직하다

- JEUS Server 안내서
- JEUS Web Container 안내서
- JEUS EJB 안내서
- JEUS Client Application 안내서

일러두기

표기 예	내용
텍스트	본문, 12 포인트, 바탕체 Times New Roman
텍스트	본문 강조
CTRL+C	Ctrl 과 C 를 동시에 누름
public class myClass { }	Java 코드
<system-config>	XML 문서
참고: / 주의:	참고 사항과 주의할 사항

표기 예	내용
설정 메뉴를 연다	UI 의버튼과 같은 컴포넌트
JEUS_HOME	JEUS 가 실제로 설치된 디렉토리
jeusadmin nodename	콘솔 명령어와 문법
[파라미터]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택사항
…	파라미터 등이 반복 되어서 나옴
?, +, *	보통 XML 문서에 각각 "없거나, 한 번", "한 번 이상","없거나 여러 번"을 나타낸다.
. . .	XML 이나 코드 등의 생략
<<*FileName.ext*>>	코드의 파일명
그림 1.	그림 이름이나 표 이름

OS 에 대해서

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

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

용어설명

본 매뉴얼을 통해 자주 볼 수 있는 용어에 대한 설명이다.

용어	정의
Console	GUI interface 와 반대되는 개념의 command-line interface (terminal window)
Data source	JDBC database connection
DD	deployment descriptor 의 약자
EJB 엔진	EJB 를 위한 infrastructure 을 제공한다
ejbadmin	EJB module 을 관리하기 위한 console 툴 (deploy 와 undeploy 등등)
EJBMain.xml	EJB 엔진 환경 파일
Engine	EJB 와 Servlet 같은 business application 실행을 위한 infrastructure 를 제공한다. Engine 은 J2EE 스펙에 따라 보통 “container” 라 불린다. JEUS 에서 지원하는 Engine 타입: EJB, Servlet, JMS, WS (Web server implementation).
Engine Container	Engine Container 는 JEUS engine 을 관리한다. Engine Container 는 JEUS Manager 와는 다른 별개의 JVM 에서 동작한다.
JEUS Manager	JEUS 를 부팅시킬 때 제일 먼저 실행시켜야 하는 프로그램

용어	정의
jeusadmin	JEUS 와 engine 을 관리하기 위한 console 툴
jeus-client-dd.xml	JEUS client application DD file.
jeus-ejb-dd.xml	JEUS EJB DD file.
JEUSMain.xml	JEUS WAS 환경파일.
jeus-web-dd.xml	JEUS Web DD file.
웹 관리자	JEUS product 에서 제공하는 web administration 툴
Node	각 JEUS node 는 JEUS Manager 에 의해 관리된다. JEUS 의 실제 인스턴스 라고 보면 된다.
Servlet engine	Servlet 과 JSP 를 위한 infrastructure 을 제공한다.
webadmin	Web WAR module 을 관리하기 위한 console 툴
WEBMain.xml	Servlet engine 환경 파일

연락처

Korea
Tmax Soft Co., Ltd
18F Glass Tower, 946-1, Daechi-Dong, Kangnam-Gu, Seoul 135-708
South Korea
Tel: 82-2-6288-2114
Fax: 82-2-6288-2115
Email: info@tmax.co.kr
Web (Korean): http://www.tmax.co.kr

USA
Tmax Soft, Inc.
560 Sylvan Ave, Englewood Cliffs NJ 07632
USA
Tel: 1-201-567-8266
FAX: 1-201-567-7339
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
Tel: 81-3-5210-9270
FAX: 81-3-5210-9277
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: 86-10-6410-6148
Fax: 86-10-6410-6144
E-mail : info@tmaxchina.com.cn
Web (Chinese): http://www.tmaxchina.com.cn

1 소개

본 매뉴얼은 TmaxSoft 의 JEUS 5 Web Application Server 를 사용하여 처음의 프로그램을 작성하는 사용자를 위한 안내서이다.

이 책은 다음과 같이 세 부분으로 나누어져 있다

- **JEUS System 설정하기:** JEUS 의 기본적인 환경 설정 방법과 운영 방법에 대해 설명한다
- **EJB 사용하기:** EJB 모듈의 Packaging 및 Deploy 방법, stateless session bean 과 CMP bean 의 이용 방법에 대해 설명한다.
- **Servlet/JSP 사용하기:** Web application 의 Packaging 방법을 설명하고 예제 Servlet /JSP 페이지를 실행시켜본다

각 부분은 순서에 따라 진행하도록 되어 있다.

JEUS WAS 에 대하여 완벽히 이해하고자 한다면, 이 문서에 담겨있는 내용을 실제로 실행시켜보는 것이 바람직하다. 문서의 내용과 실제 수행되는 화면을 주의 깊게 살펴보기 바란다.

2 JEUS System 설정하기

2.1 소개

이 장은 JEUS 의 환경 설정과 기동 방법에 대한 기본적인 내용을 설명하며, 다음 장에서 J2EE 컴포넌트를 사용하는 방법을 설명한다

이번 장은 다음과 같은 내용으로 구성되어 있다

- 웹 관리자를 이용한 기본 환경설정 방법
- 웹 관리자를 이용하여 JEUS System 컴포넌트를 추가하는 방법
- 웹 관리자를 이용하여 JEUS System 컴포넌트를 설정하는 방법
- 웹 관리자를 이용하여 JEUS System 을 기동하는 방법
- 수동으로 환경 설정하는 방법
- 수동으로 JEUS System 을 기동하는 방법

2.2 웹 관리자를 이용한 환경설정

2.2.1 기본 환경구성

JEUS 웹 관리자는 웹을 통해 JEUS 의 모든 요소를 관리하기 위한 서비스를 제공한다. 사용자는 쉽게 JEUS 에 접근하여 시스템 설정과 모니터링 및 application 을 관리할 수 있다.

다음 순서에 따라 웹 관리자를 실행시킨다.

1. 가장 먼저 JEUS Manager 를 실행시킨다. 방법은 커맨드 창에서 'jeus'라는 실행파일을 실행시킨다. 'jeus' 스크립트는 JEUS_HOME\bin\ 디렉토리에 있으며 시스템 PATH 에 설정되어 있어야 한다.

참고: 웹 관리자를 사용하기 위해서는 JEUSMain.xml 에 <enable-webadmin> 태그의 값이 true 로 설정되어 있어야 한다.

예)

```
C:\jeus\bin>jeus
************************************************************
  - JEUS Home         : C:\jeus
  - JEUS Base Port    : 9736
  - Added Java Option :
  - Java Vendor       : Sun
************************************************************
C:\jdk1.4\bin\java -server  -
Xbootclasspath/p:C:\jeus\lib\system\extension.jar;C:\jeus\lib\sys
tem\classloader.jar –classpath C:\jeus\lib\ system\bootstrap.jar
-Djeus.jvm.version=hotspot -Djeus.home=C:\je us -
Djava.naming.factory.initial=jeus.jndi.JNSContextFactory –Dja
va.naming.factory.url.pkgs=jeus.jndi.jns.url -Djava.library.path=
C:\jeus\lib\system -Djeus.session.version=socket -Djeus.baseport
=9736 -Djava.util.logging.config.file=C:\jeus\bin\logging.propert
ies  jeus.server.JeusBootstrapper
[2005.03.02 11:45:13][0] [johan-10] [MGR-0411] virtual host name
of this manager : johan
[2005.03.02 11:45:15][2] [johan-10] [NET-1011] <SocketDispatcher>
(active-mode) successfully connected to 9736
[2005.03.02 11:45:15][2] [johan-10] [NET-1031] <SocketDispatcher>
```

```
ActiveDispatcher started
[2005.03.02 11:45:15][2] [johan-10] [MGR-0184] Node security
manager started
…
```

2. 웹 브라우저를 사용해서 웹 관리자에 접속하도록 한다

```
http://localhost:9744/webadmin
```

참고: JEUS 의 기본 Base Port 는 9736 이며, 웹 관리자는 Base Port + 8 을 사용한다

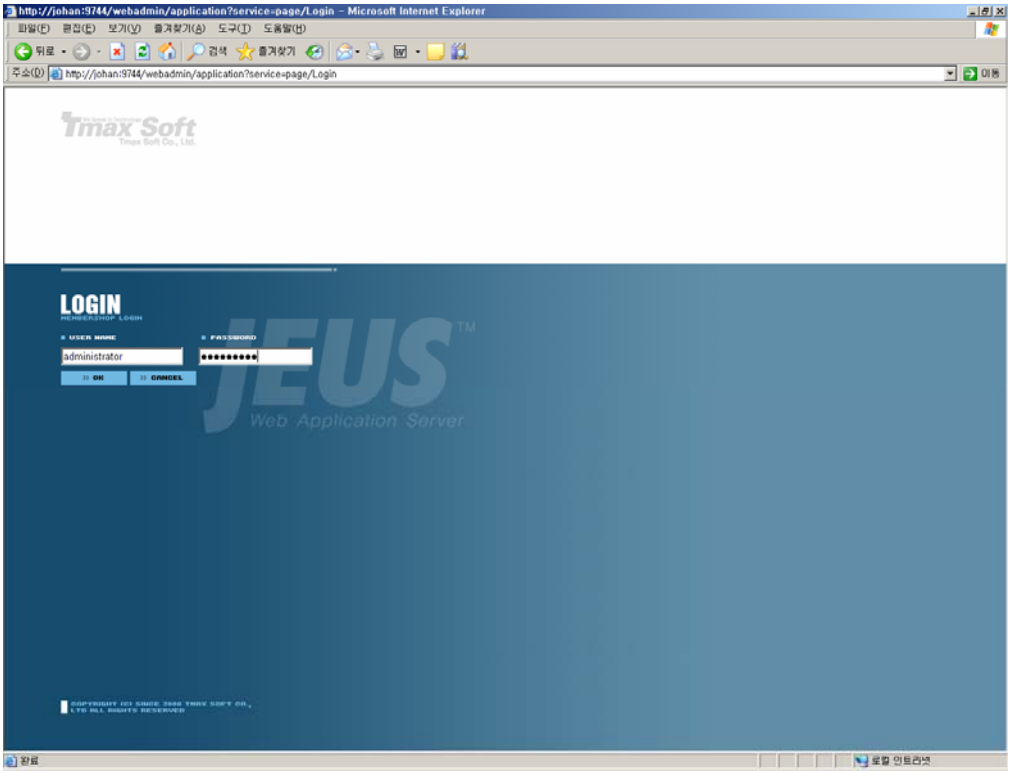

그림 1 로그인 화면

3. JEUS 를 설치할 때 설정해 주었던 JEUS 에 대한 로그인 아이디와 패스워드를 입력한다. 성공적으로 인증되었다면 다음과 같은 웹 관리자 초기화면이 뜰 것이다.

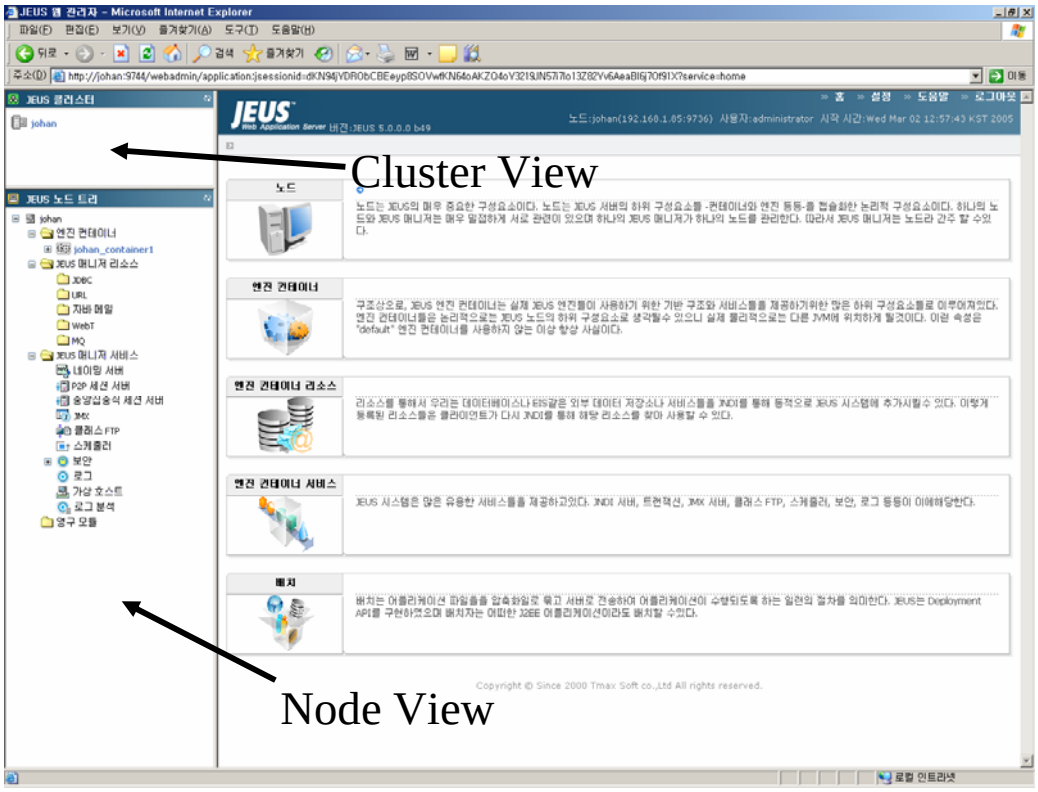

그림 2 초기 화면

아래 그림에서 보는 것과 같이 Node View(JEUS 노드 트리)에 JEUS 를 관리하기 위한 요소들이 노드명을 루트노드로 하여 트리 형태로 구성되어 있다.

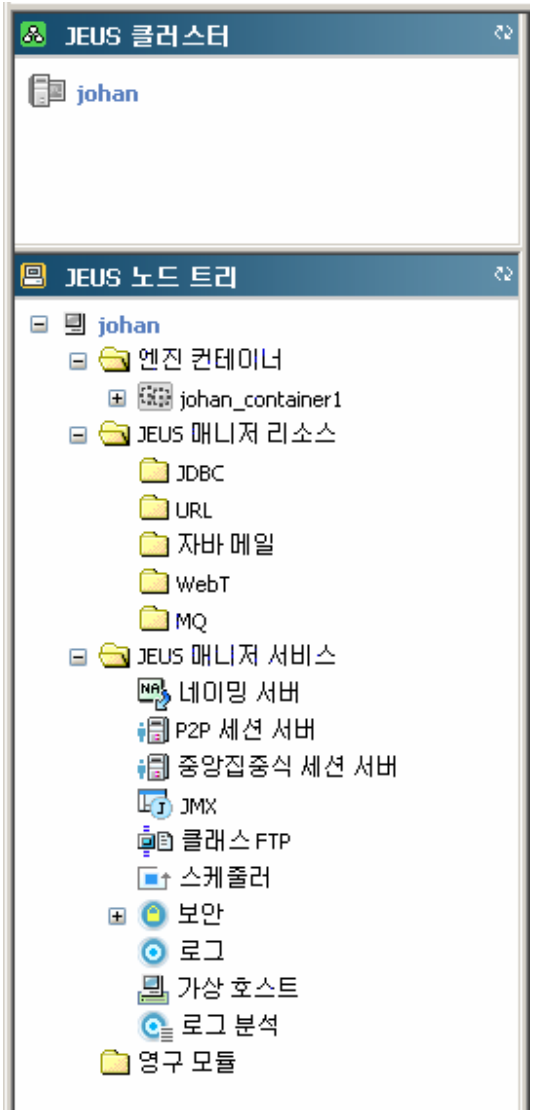

그림 3 Node View

Node View 는 크게 엔진 컨테이너, JEUS 매니저 리소스, JEUS 매니저 서비스, 영구 모듈로 구성되어 있다.

엔진과 관련된 설정변경은 각 엔진에 해당되는 설정 파일에 반영되며, 루트 노드(예: johan)를 선택했을 때 나오는 설정항목과 엔진 컨테이너 설정, JEUS 매니저 리소스 설정, JEUS 매니저 설정, 영구모듈 설정은 JEUS 의 설정파일에 반영된다

JEUS 와 각 엔진을 구동하기 위한 설정파일들은 다음의 위치에 존재한다.

JEUS 메인 %JEUS_HOME%\config\<nodename>\JEUSMain.xml

EJB 엔진 %JEUS_HOME%\config\<nodename>\<nodename>_ejb_<enginename>\EJBMain.xml

Servlet 엔진 %JEUS_HOME%\ config\<nodename>\<nodename>_servlet_<enginename>\WEBMain.xml

JMS 엔진 %JEUS_HOME%\ config\<nodename>\<nodename>_jms_<enginename>\JMSMain.xml

웹 서버 엔진 %JEUS_HOME%\ config\<nodename>\<nodename>_ws_<enginename>\WSMain.xml

2.2.2 JEUS System 컴포넌트 추가

엔진이란 J2EE 스펙을 만족하는 JEUS 엔진이다. JEUS 에는 EJB 엔진, 서블릿 엔진, JMS 엔진, 웹 서버 엔진 (Web Server) 이렇게 네 가지의 엔진 타입이 있다. 이 엔진들은 하나의 엔진 컨테이너 안에 존재한다. 단, 같은 엔진 컨테이너 안에 동일한 타입의 엔진이 두 개 이상 존재할 수는 없다.

JEUS 는 기본적으로 하나이상의 엔진 컨테이너를 필요로 하며, 설치시 하나의 엔진 컨테이너와 EJB 엔진, 서블릿 엔진을 포함한다.

엔진 컨테이너를 추가하는 방법은 기존의 엔진 컨테이너를 복사하는 방법과 새로운 엔진 컨테이너를 생성한 후 엔진을 추가하는 방법이 있다. 여기서는 새로운 엔진 컨테이너를 추가하고, 추가된 엔진 컨테이너에 EJB 엔진과 서블릿 엔진을 추가해 보도록 한다.

1. Node View 에서 루트노드 아래 '엔진 컨테이너'를 클릭한다.

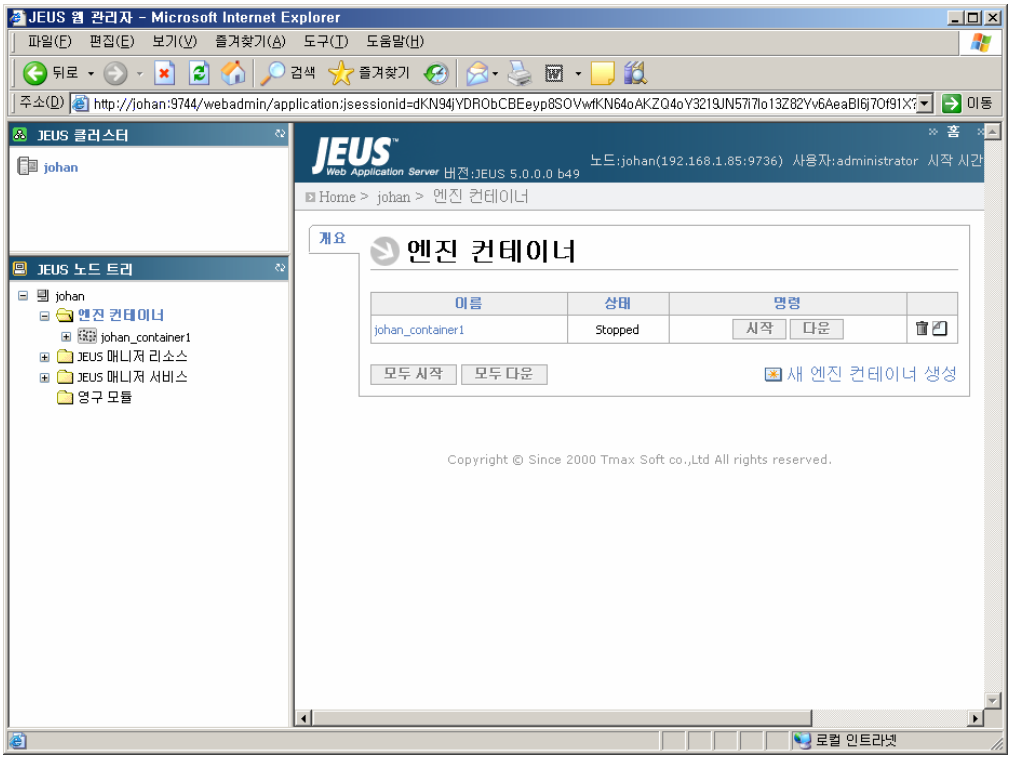

그림 4 엔진 컨테이너 개요

2. Main View 에서 '새 엔진 컨테이너 생성'을 클릭하고 엔진 컨테이너 이름을 입력한다

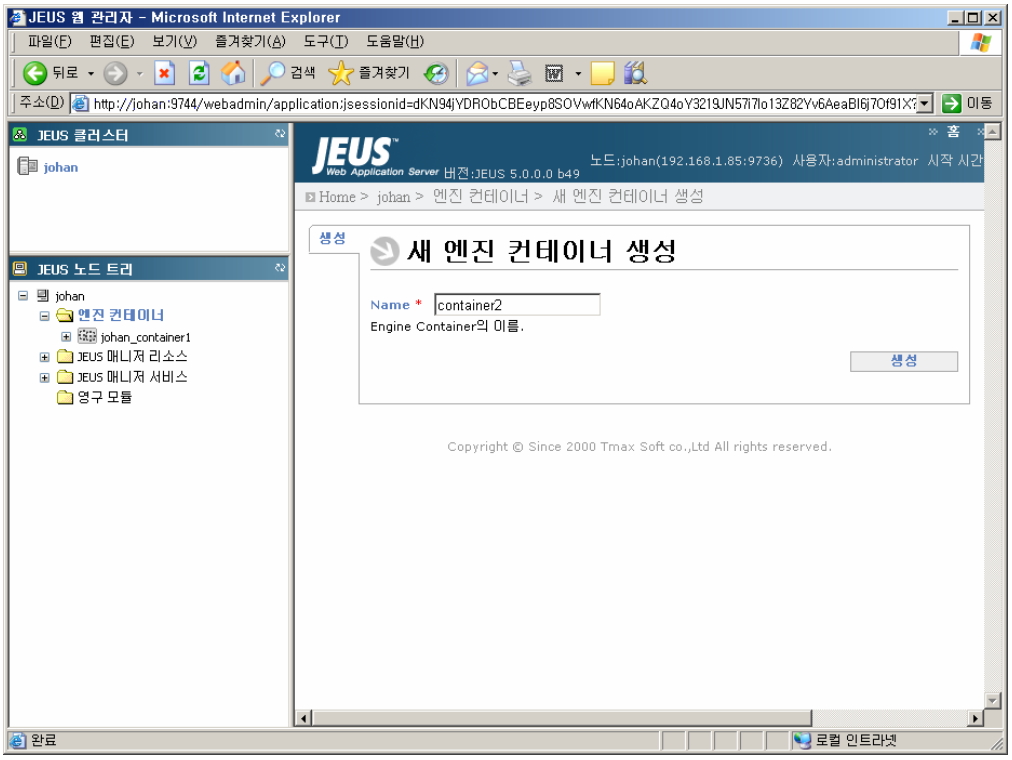

그림 5 엔진 컨테이너 생성

3. 엔진을 추가하라는 팝업 창이 뜨는데 확인버튼을 누른다. 새 엔진 컨테이너 생성이 성공할 경우 아래와 같은 화면이 나온다

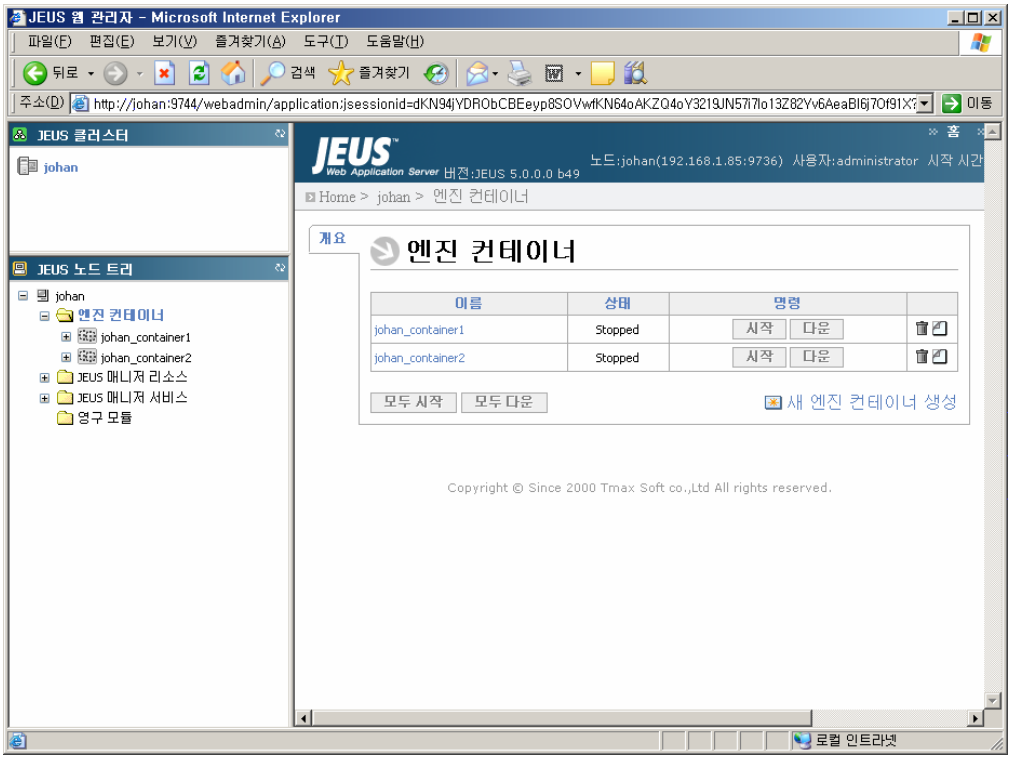

그림6 엔진 컨테이너 생성 후

4. 엔진 컨테이너에 엔진을 추가하기 위해 Node View 에서 생성한 엔진 컨테이너 아래 '엔진'을 클릭한다

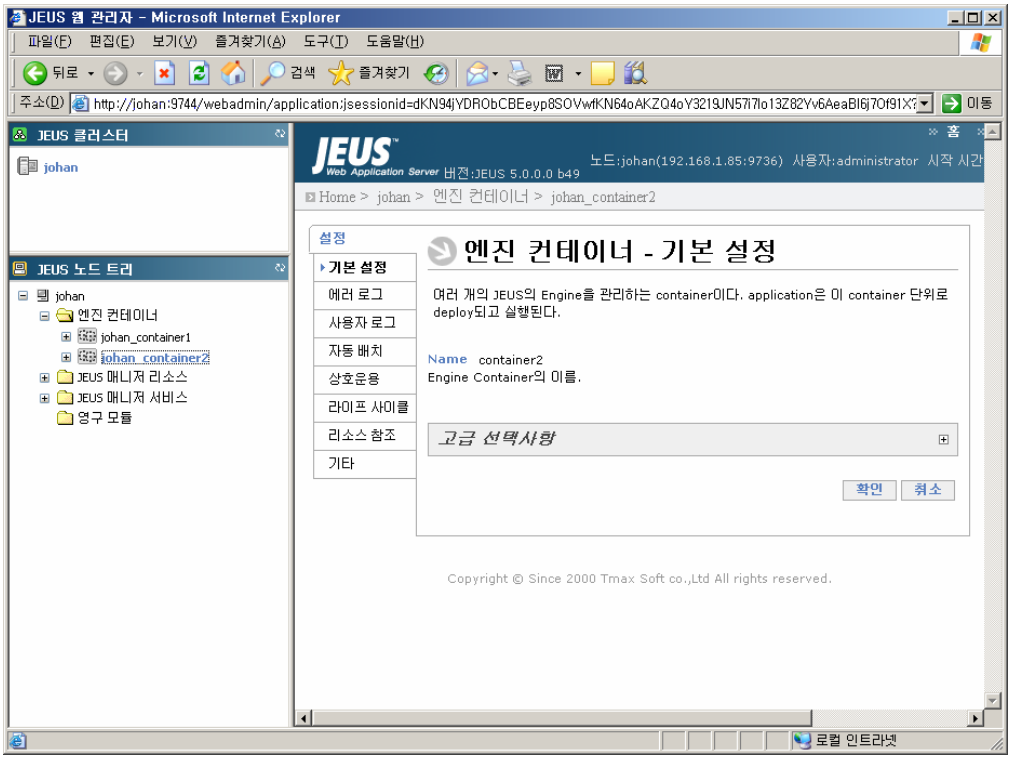

그림 7 엔진 개요

5. '새 EJB 엔진 생성' 을 클릭하고, 엔진 이름을 입력한다

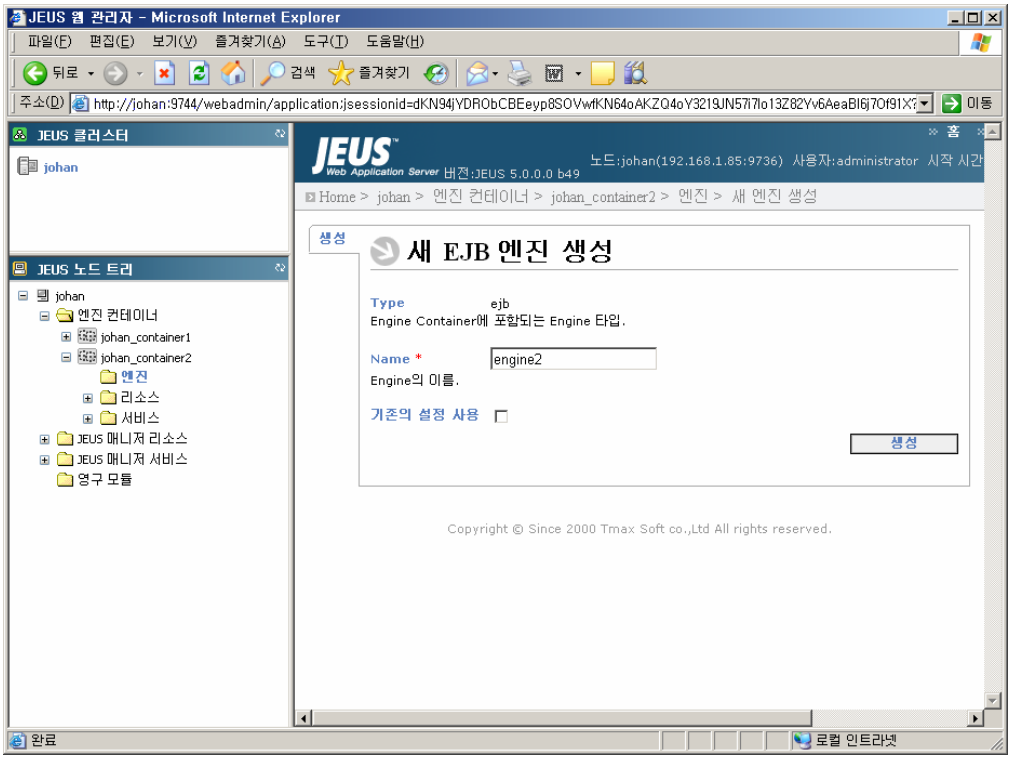

그림 8 엔진 생성

6. 확인 버튼을 클릭하면 새로 생성된 EJB 엔진을 확인할 수 있다

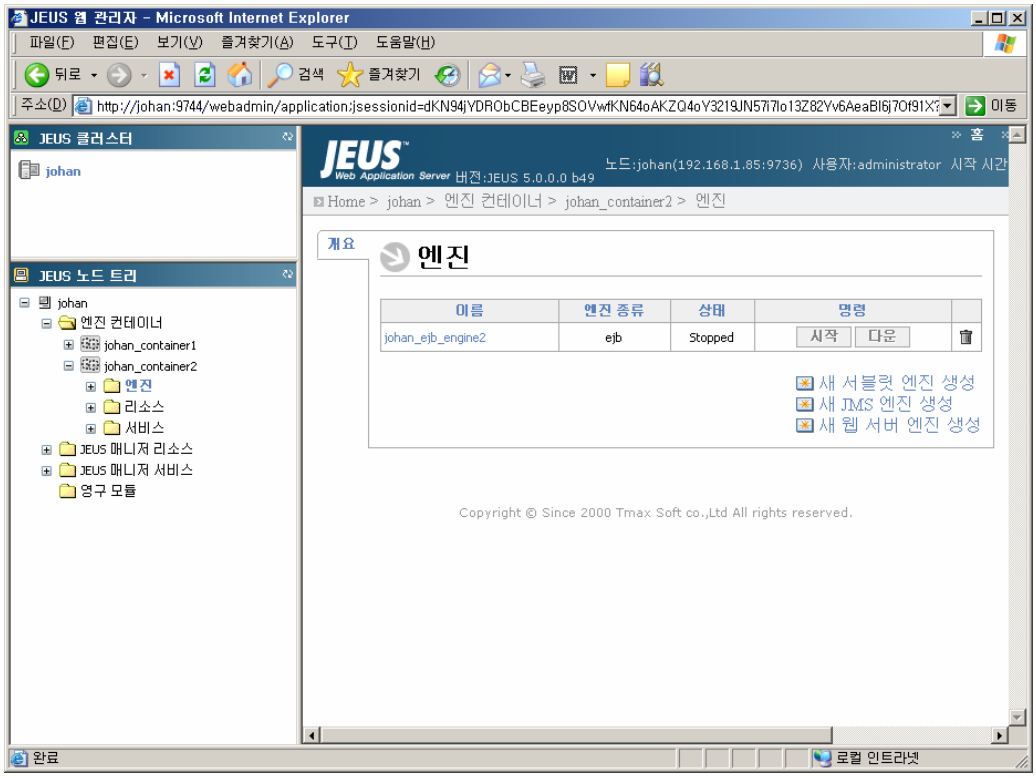

그림9 엔진 생성 후

7. 서블릿 엔진도 같은 방법으로 추가한다.

다음 절에서는 새로 생성한 엔진 컨테이너를 구동하기 위한 환경을 설정 하도록 한다.

2.2.3 JEUS System 컴포넌트 설정

새로 추가된 엔진 컨테이너를 구동하기 위한 설정은 기본 설정으로 충분하며, 서블릿 엔진이 있을 경우 컨텍스트 그룹에 등록된 웹 리스너 포트가 중복되지 않도록 설정해 주어야 한다.

새로 생성된 엔진 컨테이너 아래 '서블릿 엔진->MyGroup->http1' 을 클릭하면 아래와 같은 웹 리스너 설정 화면이 나타난다.

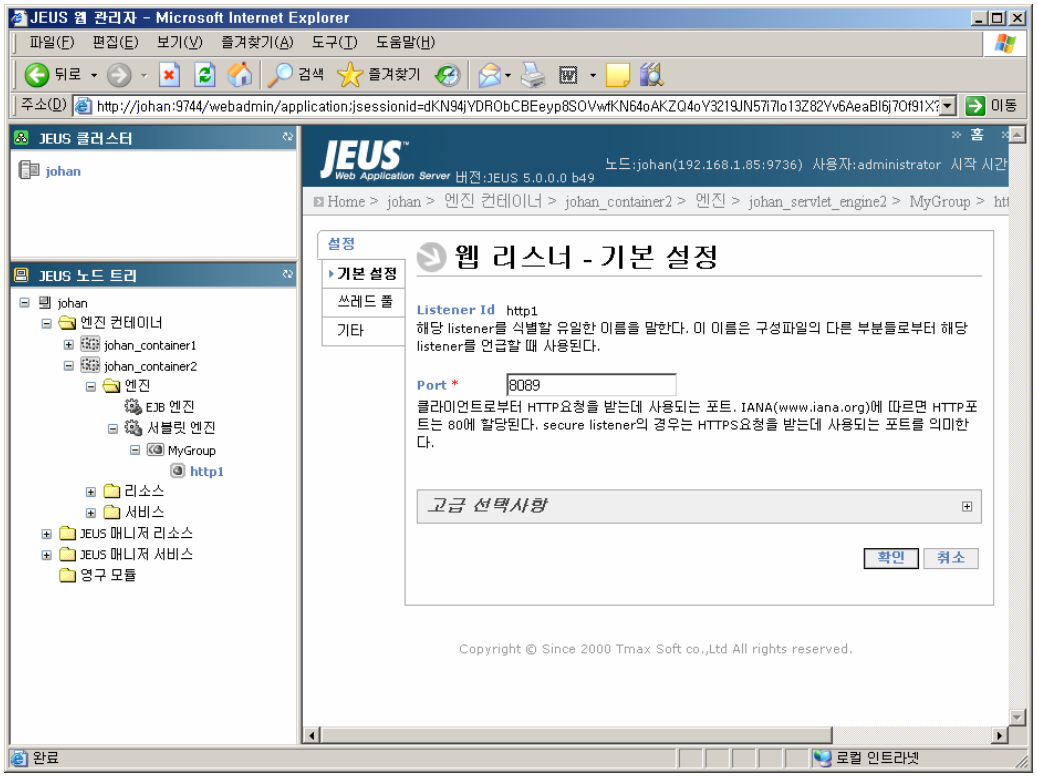

그림 10 웹 리스너 설정

다른 엔진 컨테이너의 Servlet 엔진에서 사용하는 웹 리스너 포트와 중복되지 않도록 설정한다

다음으로 Datasource 를 추가해 보도록 한다

JEUS 에서 Datasource 란 Database 와 EJB 나 Servlet/JSP 프로그램을 연결하기 위해 사용된다.

예제에서는 JEUS 에 기본적으로 포함되어 있는 HsqlDB 를 사용한다.

참고: 만약 HsqlDB 가 실행되어 있지 않다면 실행한다. Windows 의 시작 메뉴에서 TmaxSoft>JEUS5>Example Server>Start HsqlDB 를 선택한다[그림 11].

그림 11 HsqlDB 의 실행

Unix/Linux 에서 또는 Windows 의 명령 프롬프트 창에서는 다음과 같이 실행한다.

```
> starthsqldb
```

참고: HsqlDB 을 JEUS 에서 사용하려면 HsqlDB 의 JDBC 드라이버 파일인 hsqldb.jar 가 %JEUS_HOME%\lib\datasource 에 있어야 한다.

참고: HsqlDB 에 대한 자세한 내용은 다음 사이트를 참고하길 바란다.

http://hsqldb.sourceforge.net/

1. Node View 에서 'JEUS 매니저 리소스->JDBC'를 선택한다

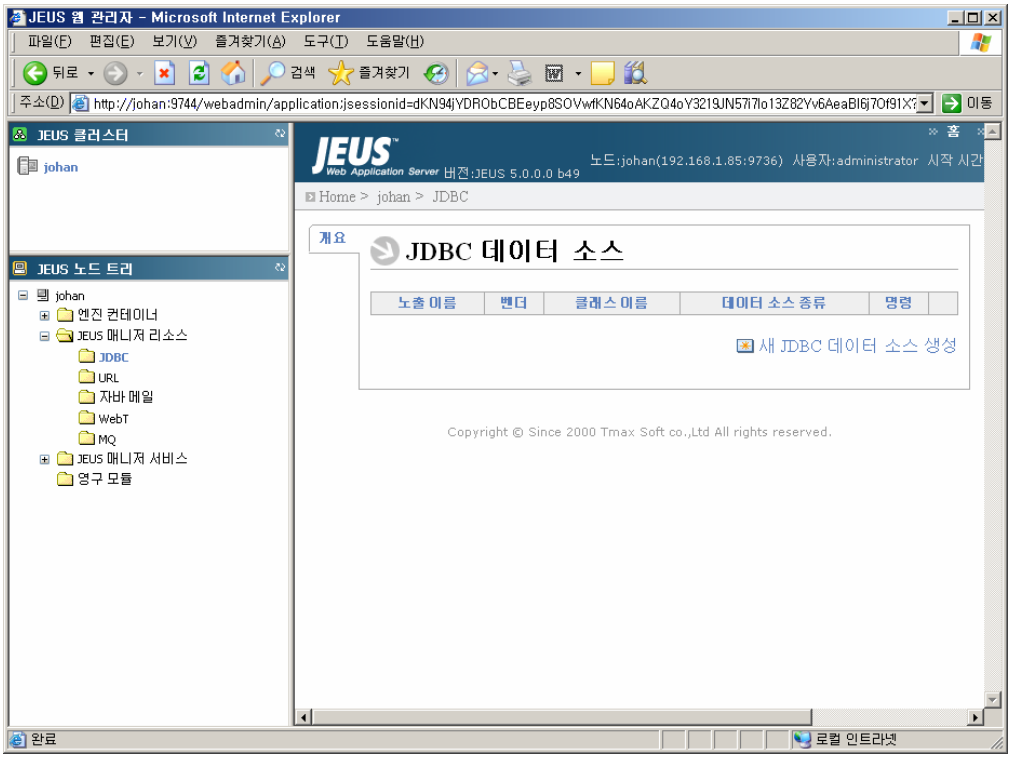

그림 12 JDBC 데이터소스

2. '새 JDBC 데이터 소스 생성'을 클릭하고, 데이터소스 설정 화면에서 'DBMS'는 Hsql 을, '가능한 데이터 소스들'에서는 Hsql Connection PoolDataSource 를 선택한다

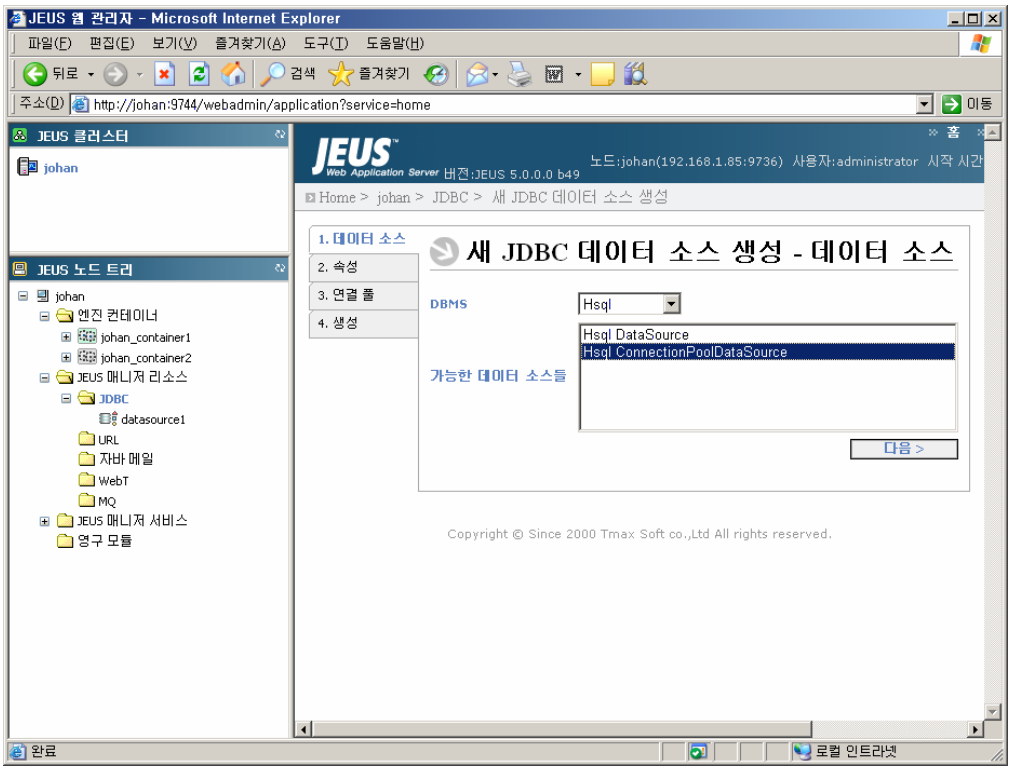

그림 13 데이터소스 선택

3. 다음 화면에서 데이터소스의 속성을 설정한다. 여러 입력 항목들이 기본 값으로 설정되어 나온다. '추가 속성들'에서 User 의 값으로 sa 를 넣어준다. 이외의 항목은 적절한 값을 설정하고 '다음' 버튼을 클릭한다

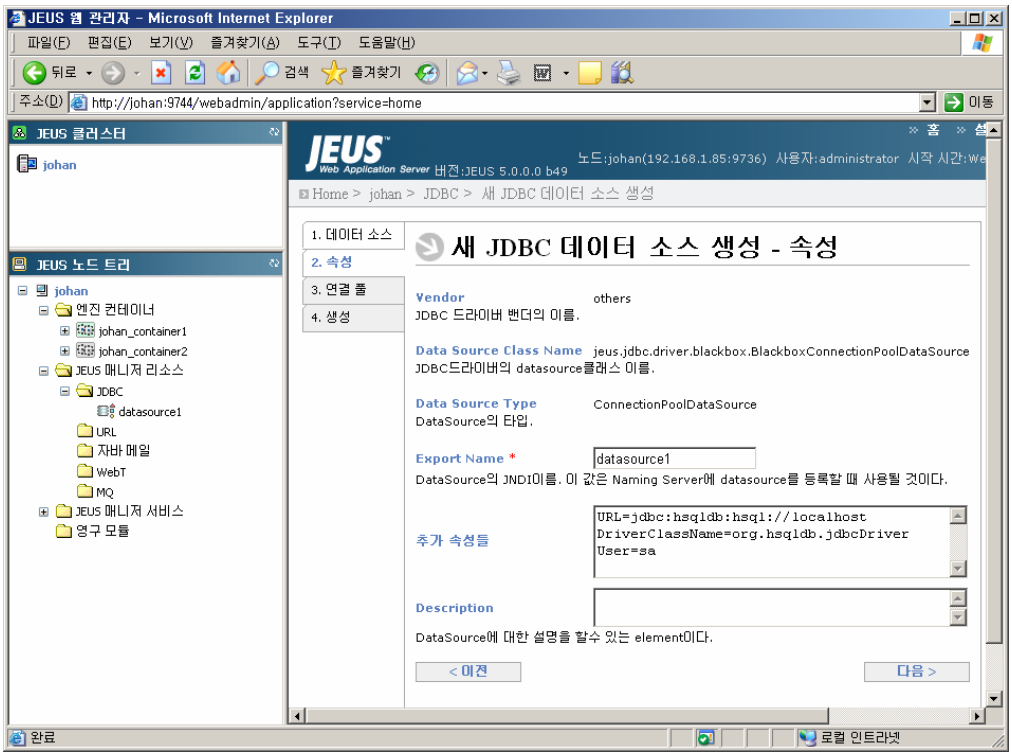

그림 14 데이터소스 속성

4. 쓰레드 풀 설정은 기본설정으로 선택하고, '다음' 버튼을 클릭한다

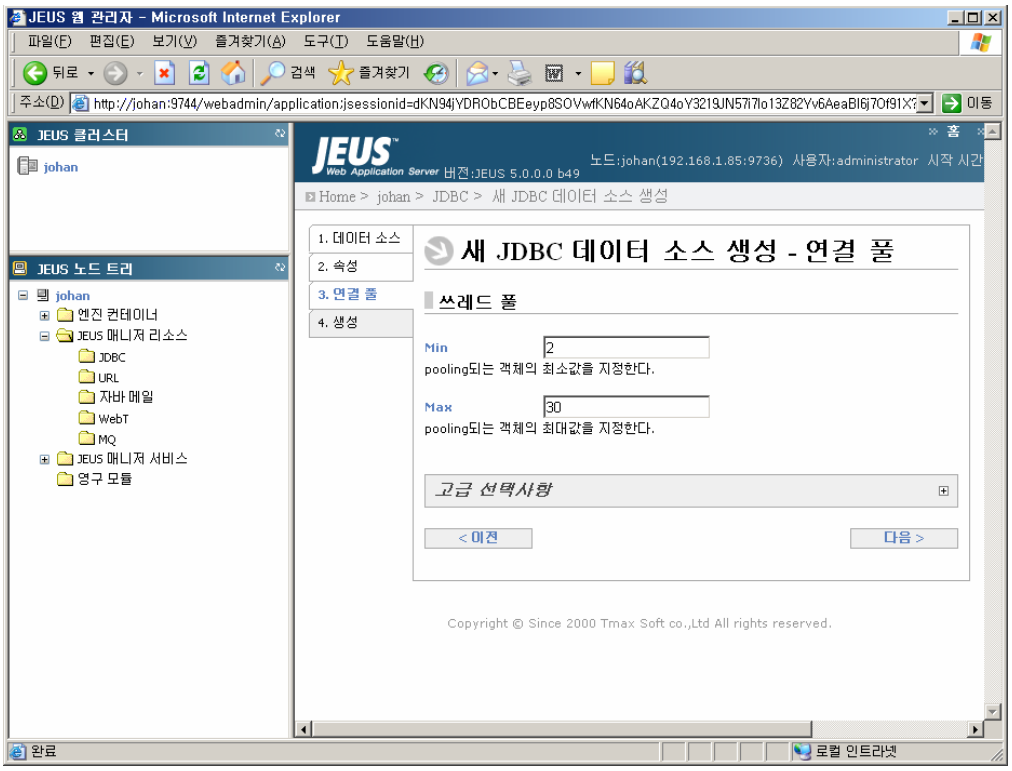

그림 15 데이터소스 연결풀

5. '생성' 버튼을 클릭한다

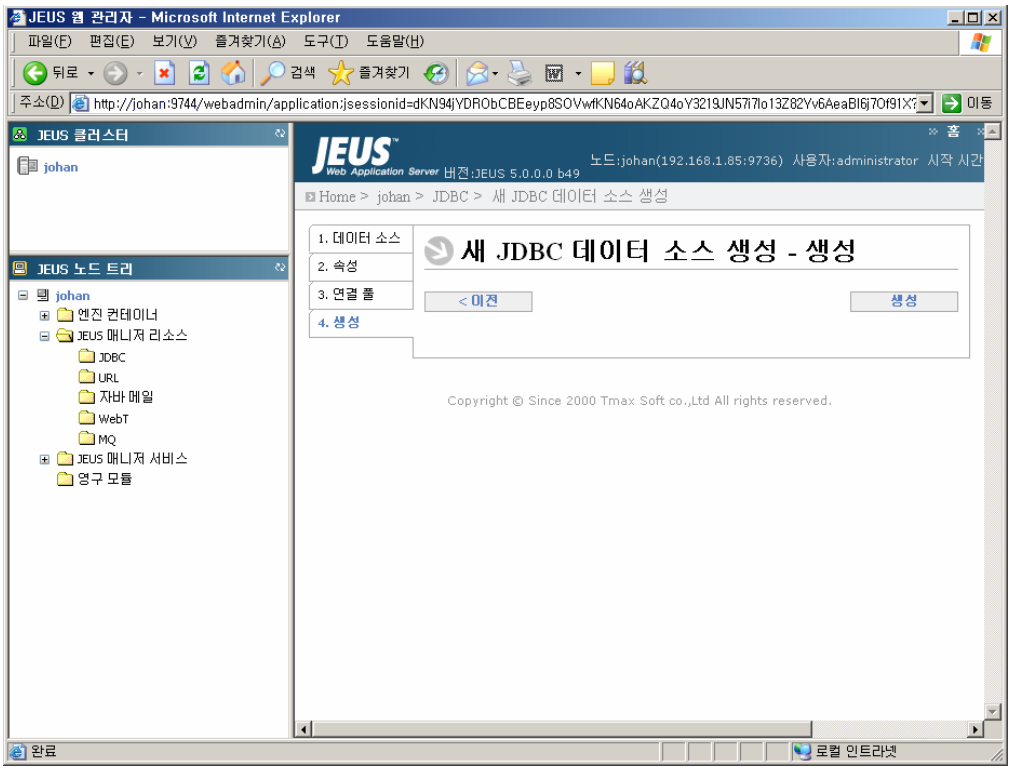

그림 16 데이터소스 생성

6. 데이터소스가 정상적으로 설정되었는지 확인한다

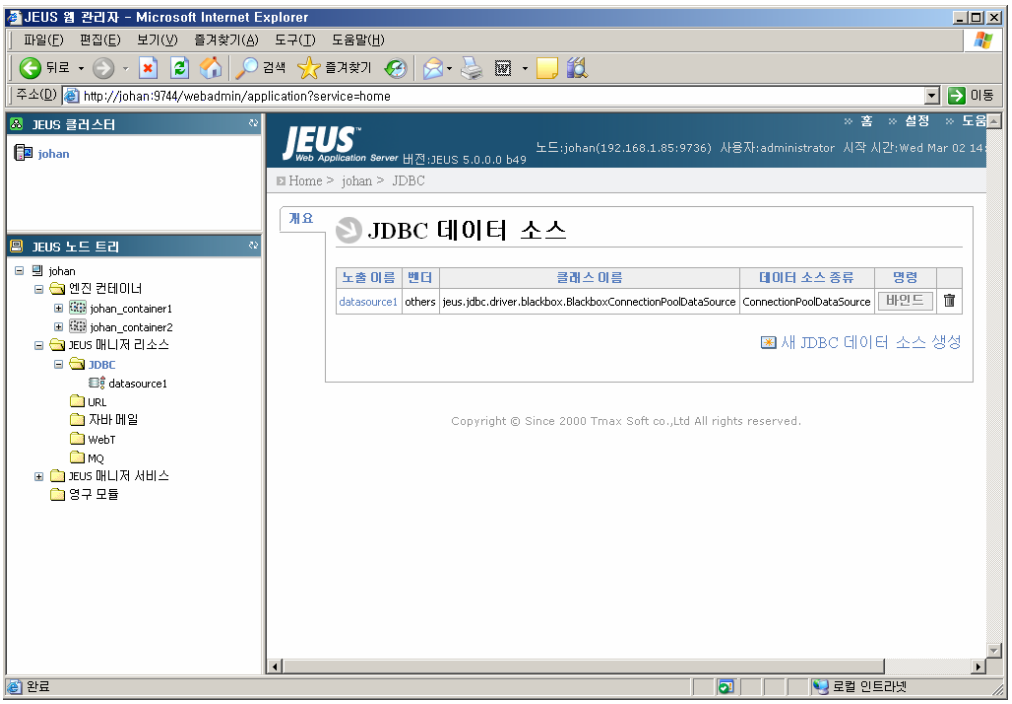

그림 17 데이터소스 생성 후

참고: 데이터소스 설정은 JEUS 설정 파일에 반영된다. Node View 에 'JEUS 매니저 리소스->JDBC' 아래 데이터소스 이름의 트리노드가 생성되며, 바인드할 경우 '루트노드->JNDI 트리'에서 데이터소스를 확인할 수 있다.

커넥션 풀 데이터소스일 경우 설정 후 바로 커넥션 풀이 생성되지 않고, J2EE 모듈에서 처음 데이터소스를 사용할 때, 모듈이 Deploy 된 엔진 켄테이너의 '리소스->JDBC' 아래 나타나게 된다. 데이터소스의 정상적인 동작 여부 또한 커넥션을 처음 사용할 경우 알 수 있다.

2.3 웹 관리자를 이용한 JEUS 기동

웹 관리자를 통해 JEUS 를 기동하고자 할 경우, JEUS Manager 가 실행된 상태여야 한다.

JEUS Manager 가 실행된 상태에서 웹 관리자를 통해 JEUS 를 부트, 다운 시킬 수 있고 JEUS Manager 를 종료시킬 수 있다.

1. Node View 에서 루트노드를 선택한다

2. Main View 에서 제어 탭을 클릭하면, 아래와 같이 부트, 다운, 종료 메뉴를 볼 수 있다. 현재 JEUS 가 부팅되지 않은 상태이므로, 부트 탭을 선택하면 부팅 가능한 노드가 나타난다

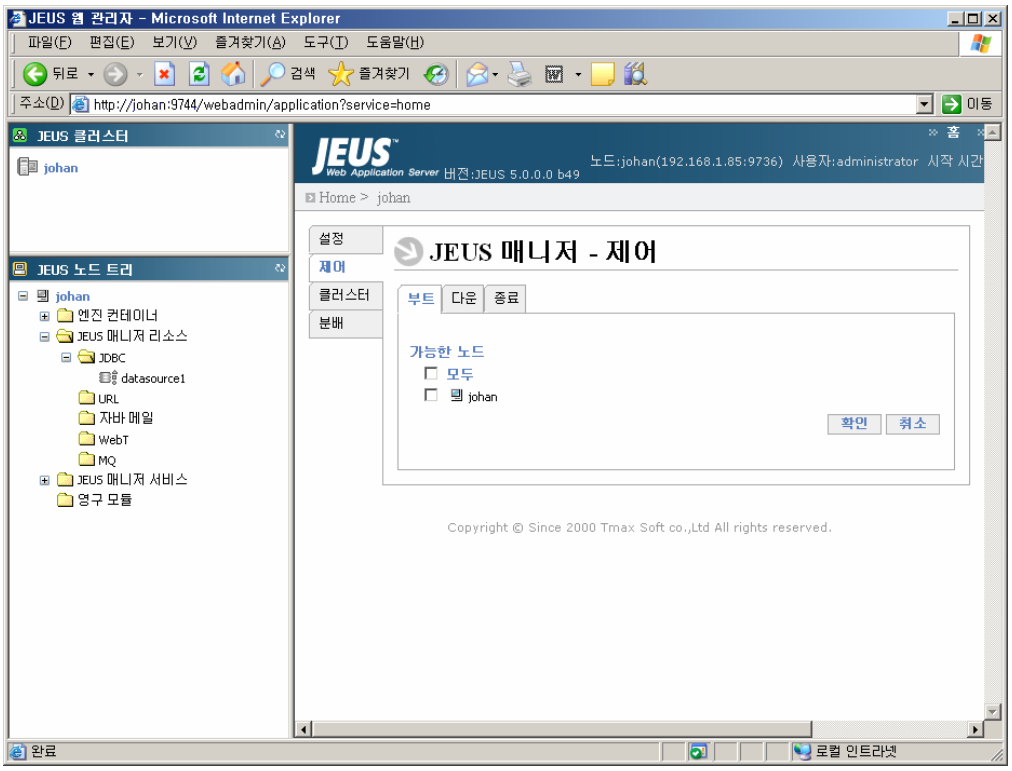

그림 18 JEUS 매니저의 제어 중 부트 화면

3. 앞 절에서 추가한 컨테이너도 부팅하려게 하려면 JEUS 를 다시 실행시켜야 한다. 그러므로 우선 종료 탭을 눌러서 JEUS 를 완전히 종료시키도록 한다.

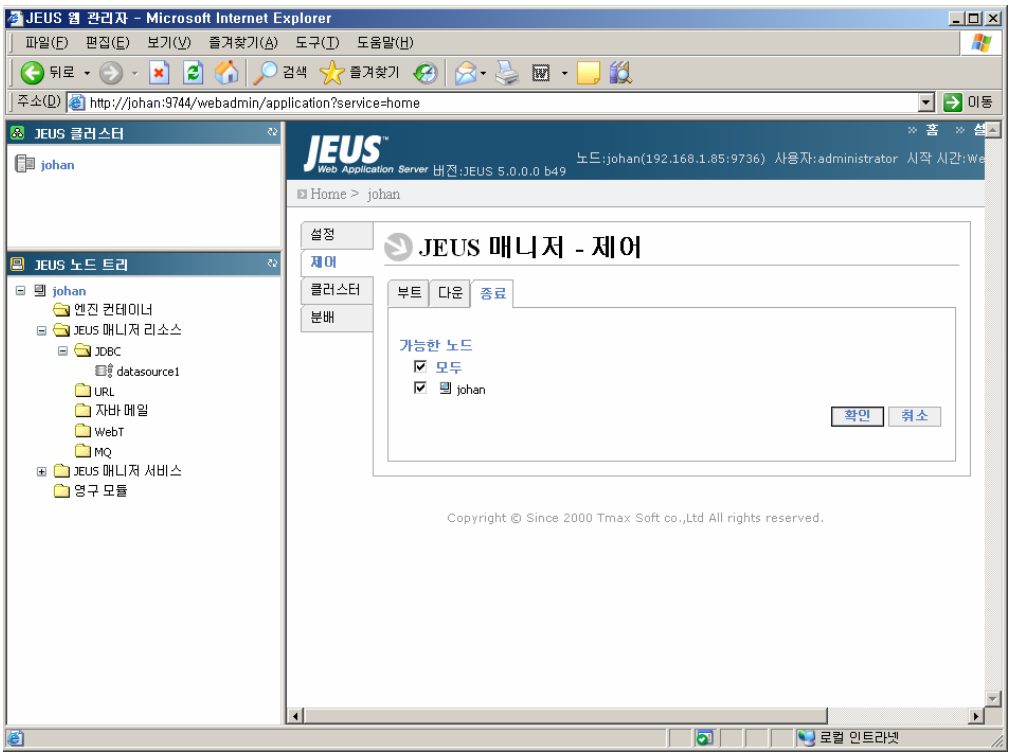

그림 19 JEUS 매니저의 제어 중 종료 화면

4. JEUS 가 완전히 종료된 것을 확인한 다음, JEUS 를 다시 실행시킨다. 그리고 앞서와 같은 방법으로 웹 매니저를 실행시킨다.

5. 노드를 선택하고 확인 버튼을 클릭하면 JEUS 가 부팅된다.

참고: 엔진 컨테이너와 엔진 컨테이너에 포함된 엔진을 개별적으로 부트, 다운할 수 있다. Node View 에서 '엔진 컨테이너'나 '엔진'을 선택하면 [그림 20]과 [그림 21] 같은 메뉴를 볼 수 있다.

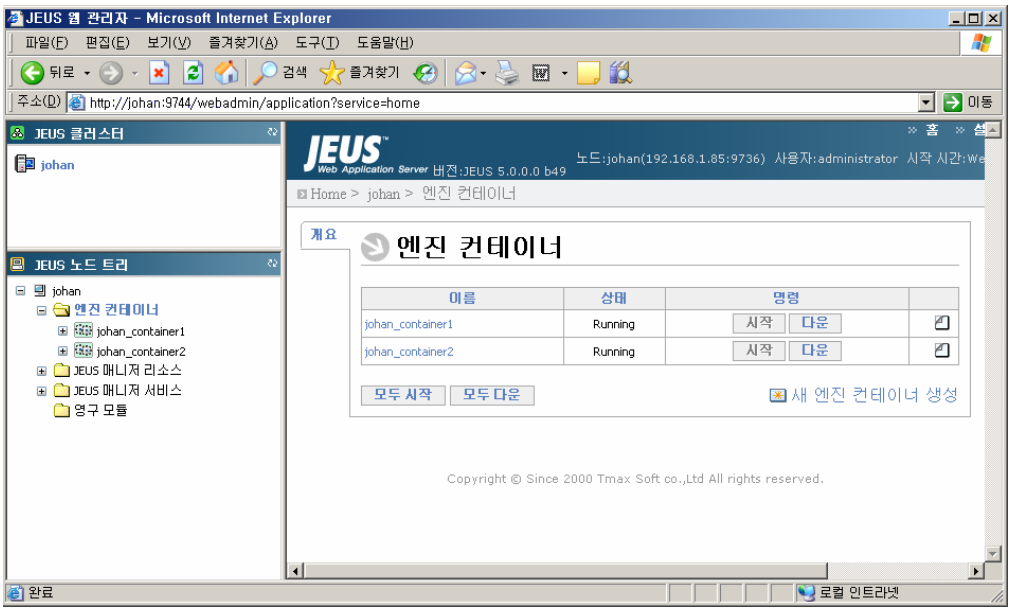

그림 20 엔진 컨테이너 제어

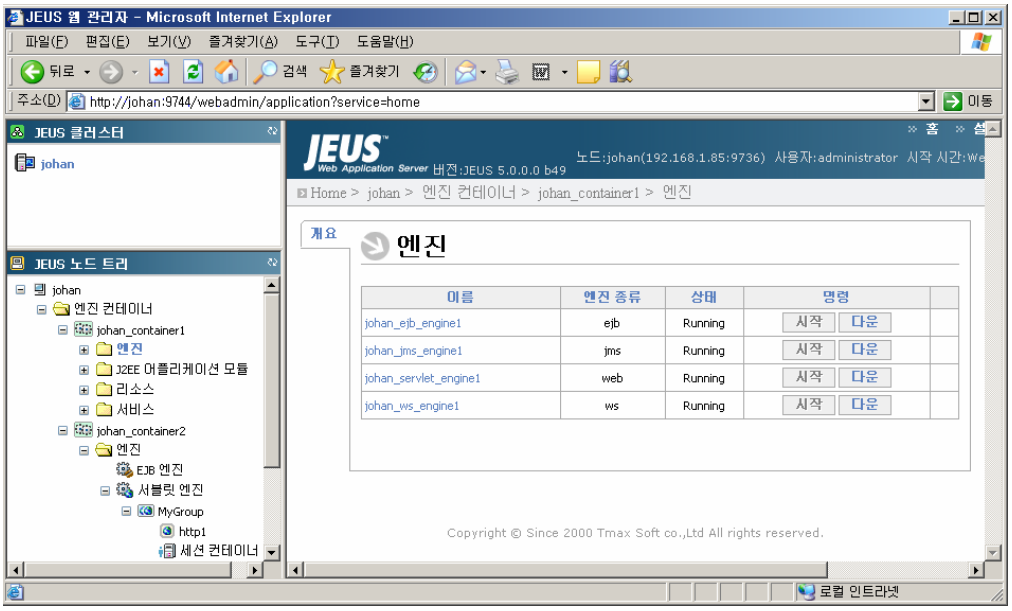

그림 21 엔진 제어

2.4 Text editor 를 이용한 환경설정

웹 관리자가 아닌 text editor 를 이용하여 %JEUS_HOME%\config\<node name>\JEUSMain.xml 파일을 생성하고 설정할 수 있다.

JEUS 가 인스톨시 설정되는 노드 이름은 해당 호스트 머신의 네트워크 ID 명이다. 윈도우상에서는 hostname 이라는 명령어를 통해 유닉스에서는 uname -an 이라는 명령어를 통해 이 이름을 알아낼 수 있다.

만약 JEUS 의 VirtualHost 기능을 사용한다면, vhost.xml 파일을 참고해서 적절한 노드 이름을 선택해서 작업한다.

아래 JEUSMain.xml 예제는 이전 절의 모든 설정값(노드, 엔진 컨테이너, EJB 엔진, Servlet 엔진, database connection)을 적용한 것이다.

<<JEUSMain.xml>>

```
<?xml version="1.0" encoding="utf-8"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <node>
        <name>johan</name>
        <engine-container>
            <name>container1</name>
            <engine-command>
                <type>ws</type>
                <name>engine1</name>
            </engine-command>
            <engine-command>
                <type>jms</type>
                <name>engine1</name>
            </engine-command>
            <engine-command>
                <type>ejb</type>
                <name>engine1</name>
            </engine-command>
            <engine-command>
                <type>servlet</type>
                <name>engine1</name>
            </engine-command>
            <sequential-start>true</sequential-start>
```

```
        </engine-container>
        <engine-container>
            <name>container2</name>
            <engine-command>
                <type>ejb</type>
                <name>engine2</name>
            </engine-command>
            <engine-command>
                <type>servlet</type>
                <name>engine2</name>
            </engine-command>
        </engine-container>
        <class-ftp>true</class-ftp>
        <sequential-start>true</sequential-start>
        <enable-webadmin>true</enable-webadmin>
    </node>
    <resource>
        <data-source>
            <database>
                <vendor>others</vendor>
                <export-name>datasource1</export-name>
                <data-source-class-name>
      jeus.jdbc.driver.blackbox.BlackboxConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                   ConnectionPoolDataSource
                </data-source-type>
                <property>
                    <name>URL</name>
                    <type>java.lang.String</type>
                    <value>jdbc:hsqldb:hsql://localhost</value>
                </property>
                <property>
                    <name>DriverClassName</name>
                    <type>java.lang.String</type>
                    <value>org.hsqldb.jdbcDriver</value>
                </property>
                <property>
                    <name>User</name>
```

```
                    <type>java.lang.String</type>
                    <value>sa</value>
                </property>
            </database>
        </data-source>
    </resource>
</jeus-system>
```

JEUSMain.xml 파일에 EJB 엔진과 Servlet 엔진만 정의하는 것으로는 충분하지 않다. 추가적으로 이들 엔진의 설정 파일이 필요한데, 이 설정은 JEUS 에서 EJB 엔진과 Servlet 엔진을 시작하기 위해 필요하다. EJB 엔진을 위해서는 EJBMain.xml 이, Servlet 엔진을 위해서는 WEBMain.xml 파일이 필요하다

<<EJBMain.xml>>

```
<?xml version="1.0"?>
<ejb-engine xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <resolution>10000</resolution>
</ejb-engine>
```

<<WEBMain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-container xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <context-group>
    <group-name>MyGroup</group-name>
    <group-docbase>webapps</group-docbase>
    <webserver-connection>
      <http-listener>
        <listener-id>http1</listener-id>
        <port>12106</port>
        <output-buffer-size>8192</output-buffer-size>
        <thread-pool>
          <min>10</min>
          <max>20</max>
          <step>1</step>
          <max-idle-time>300000</max-idle-time>
          <max-wait-queue>4</max-wait-queue>
          <max-queue>-1</max-queue>
```

```
        </thread-pool>
        <postdata-read-timeout>30000</postdata-read-timeout>
        <back-log>50</back-log>
        <server-access-control>false</server-access-control>
      </http-listener>
    </webserver-connection>
</context-group>
</web-container>
```

2.5 Command 창에서 JEUS 기동

JEUSMain.xml, EJBMain.xml 과 WEBMain.xml 을 위와 같이 설정했다면 JEUS 를 시작할 준비가 된 것이다. 콘솔 창에서 ‘jeus‘라는 명령어를 통해 JEUS Manager 를 시작한다. 그러면 JEUS 를 시작하고 엔진 컨테이너나 엔진을 시작시킬 수 있는 stand-by 모드로 들어가게 된다.

1. 콘솔 창에 ‘jeus’라고 입력한다

```
C:\jeus\bin>jeus
************************************************************
  - JEUS Home         : C:\jeus
  - JEUS Base Port    : 9736
  - Added Java Option :
  - Java Vendor       : Sun
************************************************************
C:\jdk1.4\bin\java -server  -
Xbootclasspath/p:C:\jeus\lib\system\extension.jar;C:\jeus\lib\sys
tem\classloader.jar -classpath C:\jeus\li
b\system\bootstrap.jar -Djeus.jvm.version=hotspot -
Djeus.home=C:\jeus -
Djava.naming.factory.initial=jeus.jndi.JNSContext
Factory -Djava.naming.factory.url.pkgs=jeus.jndi.jns.url -
Djava.library.path=C:\jeus\lib\system -Djeus.session.version=s
ocket -Djeus.baseport=9736 -
Djava.util.logging.config.file=C:\jeus\bin\logging.properties
jeus.server.JeusBootstrapper
```

```
[2005.03.02 15:55:57][0] [johan-10] [MGR-0411] virtual host name
of this manager : johan
[2005.03.02 15:55:59][2] [johan-10] [NET-1011] <SocketDispatcher>
(active-mode) successfully connected to 9736
[2005.03.02 15:55:59][2] [johan-10] [NET-1031] <SocketDispatcher>
ActiveDispatcher started
[2005.03.02 15:55:59][2] [johan-10] [MGR-0184] Node security
manager started
…
[2005.03.02 15:56:04][2] [johan-10] [MGR-0191] System Engine
booted
[2005.03.02 15:56:04][2] [johan-10] [JMX-0011] create MBean :
JEUS:name=johan,j2eeType=JeusService,JeusManager=johan,jeu
sType=ContainerManagerService,JMXManager=johan
[2005.03.02 15:56:04][2] [johan-10] [JMX-0011] create MBean :
JEUS:name=johan,j2eeType=JeusService,JeusManager=johan,jeu
sType=ContainerMonitorService,JMXManager=johan
[2005.03.02 15:56:04][0] [johan-10] [MGR-0241] JeusServer is
Ready
```

2. "`JeusServer is Ready`" 메시지가 콘솔 창에 나타나면 부팅시킬 수 있는 준비가 된 것이다.

3. JEUS 의 bin 디렉토리에 있는 콘솔 툴인 '`jeusadmin`'을 실행시킨다. "`jeusadmin <node name>`"이라고 실행시키며, 여기서 nodename 은 JEUS node 의 이름을 가리킨다(예제의 경우 "johan"가 된다). JEUS 를 설치할 때 정의 했던 관리자의 ID 와 Password 를 입력한다. 관리자의 ID 는 기본적으로 administrator 로 되어 있다.

```
C:\jeus\bin>jeusadmin johan
Login name>administrator
Password>
JEUS 5.0 Jeus Manager Controller
johan>
```

4. "`johan>`" 라는 jeusadmin prompt 가 나타나면 JEUS 를 제어할 수 있게 된다. 앞서 '`jeus`' 커맨드로 실행시킨 JEUS Manager 에 명령을 전달함으로써 제어한다

5. JEUS 와 설정된 EJB/Servlet 엔진을 실행시키기 위해 'boot'라는 명령어를 입력한다

```
johan>boot
johan_container1
johan_conatiner2
```

6. prompt 가 나타나면 JEUS 가 부팅된 것이다.

7. EJB 와 Servlet 엔진이 정상적으로 부팅되었는지 확인하기 위해 jeusadmin prompt 에서 'allenglist'를 실행시킨다

```
johan>allenglist
johan_servlet_engine1
johan_ejb_engine1
```

8. jeusadmin prompt 에서 'down'명령어를 실행시키면 JEUS 가 종료된다.

```
johan>
Do you really want to down whole JEUS? (y : n):>y
johan down successful
```

9. jeusadmin prompt 에서 'jeusexit'를 치면 JEUS Manager 가 완전히 종료되고 'exit'를 치면 jeusadmin 이 종료된다.

```
johan>jeusexit
johan jeusexit successful
```

10. 각 엔진의 부트, 다운은 starteng, downeng 명령으로 실행할 수 있다.

```
johan>downeng johan_ejb_engine1
johan_ejb_engine1 engine down successful

johan>starteng johan_ejb_engine1
johan_ejb_engine1 engine started successful
```

2.6 결론

이번 장에서는 웹 관리자와 텍스트 기반으로 JEUS 환경 설정하는 방법을 간단히 소개하였다.

웹 관리자를 사용할 수 없는 환경에서 텍스트 기반으로 편집할 경우 JEUS 서버 안내서, EJB 안내서, Web Container 안내서, JMS 안내서의 부록이나 해당 스키마를 참조하면 설정하는데 도움이 될 수 있다.

스키마 파일은 %JEUS_HOME%\config\xsds 아래에서 참조할 수 있다

다음 장에서는 여기서 설정한 EJB, Servlet 엔진과 데이터소스를 기반으로, JEUS 가 클라이언트에게 서비스를 제공하도록 하기 위해 EJB, Servlet/JSP 을 packaging, deploy 하는 방법을 설명할 것이다.

3 EJB 사용하기

3.1 소개

이번 장에서는 다음의 내용에 대해 설명한다

- Stateless Session Bean 의 deploy: Session Bean 을 개발하고 deploy 하는 과정을 다룬다.
- CMP Entity Bean 의 deploy: Entity Bean 을 개발하고 deploy 하는 과정을 다룬다

3.2 Session Bean Deploy

이 장에서는 stateless session bean interface 를 구현하는 간단한 예제코드를 작성하고, 해당 소스의 컴파일과 deploy 과정을 살펴보기로 한다. 예제 코드는 `%JEUS_HOME%\samples\manual_examples\GettingStartedTutorial\EJBTutorial\`에서 찾아 볼 수 있다.

3.2.1 EJB 예제코드

Stateless session bean 은 기본적으로 home interface 와 remote interface 그리고 enterprise bean class 로 구성된다. 아래의 예제코드는 “Hello World”를 출력하는 business method 를 하나만 가진 간단한 `stateless session bean` 예제이다.

Home Interface (HelloHome.java)

<<HelloHome.java>>

```
package hello;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface HelloHome extends EJBHome
{
    Hello create() throws RemoteException, CreateException;
}
```

Remote Interface (Hello.java)

<<Hello.java>>

```
package hello;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Hello extends EJBObject
{
    String sayHello() throws RemoteException;
}
```

Bean Class (HelloEJB.java)

<<HelloEJB.java>>

```
package hello;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.lang.*;

public class HelloEJB implements SessionBean {

    public HelloEJB() {
    }

    public String sayHello() throws RemoteException {
      return "Hello World!";
    }

    public void ejbCreate() {
    }

    public void ejbRemove() throws RemoteException {
    }

    public void setSessionContext(SessionContext sc) {
```

```
    }

    public void ejbActivate() {
    }

    public void ejbPassivate() {
    }
}
```

3.2.2 EJB 예제코드 컴파일

EJB Source 파일들을 컴파일하기 위해서는 EJB 관련 클래스들을 class path 에 추가해주어야 하는데 JEUS 에서 사용하는 클래스 및 J2EE 관련 클래스는 "%JEUS_HOME%\lib\system\jeus.jar"에 묶여서 제공되므로 해당 파일을 class path 에 추가해주면 된다. 만일 클래스 패스에 추가하기 힘들다면 다음처럼 컴파일 시에 직접 –classpath 옵션을 사용해서도 컴파일이 가능하다.

예)

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\EJB
Tutorial\src\bean\hello>javac –classpath C:\jeus\lib\system\jeus.
jar -d ..\..\..\classes *.java
```

컴파일이 정상적으로 수행이 되면 클래스 파일들이 패키지 형태의 하위 디렉토리 아래에 생성이 된다. 위의 경우는 hello 라는 디렉토리가 생성이 되고 그 아래에 클래스 파일이 생성된다. 즉 다음과 같은 경로에 생긴다.

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\EJB
Tutorial\classes
```

3.2.3 JEUS Builder 로 EJB 모듈 패키징 하기

컴파일이 성공하면 class 파일들을 J2EE EJB 모듈로 패키징하는 과정을 거쳐야 한다.

1. 모듈 패키징을 위해 제공되는 JEUS Builder 를 실행시키위해 커맨드 창에서 'jeusbuilder' 명령을 실행시킨다. 아래 화면은 JEUS Builder 를 실행시키면 나오는 첫 화면으로, Workspace View 와 Message View 로 구분된다.

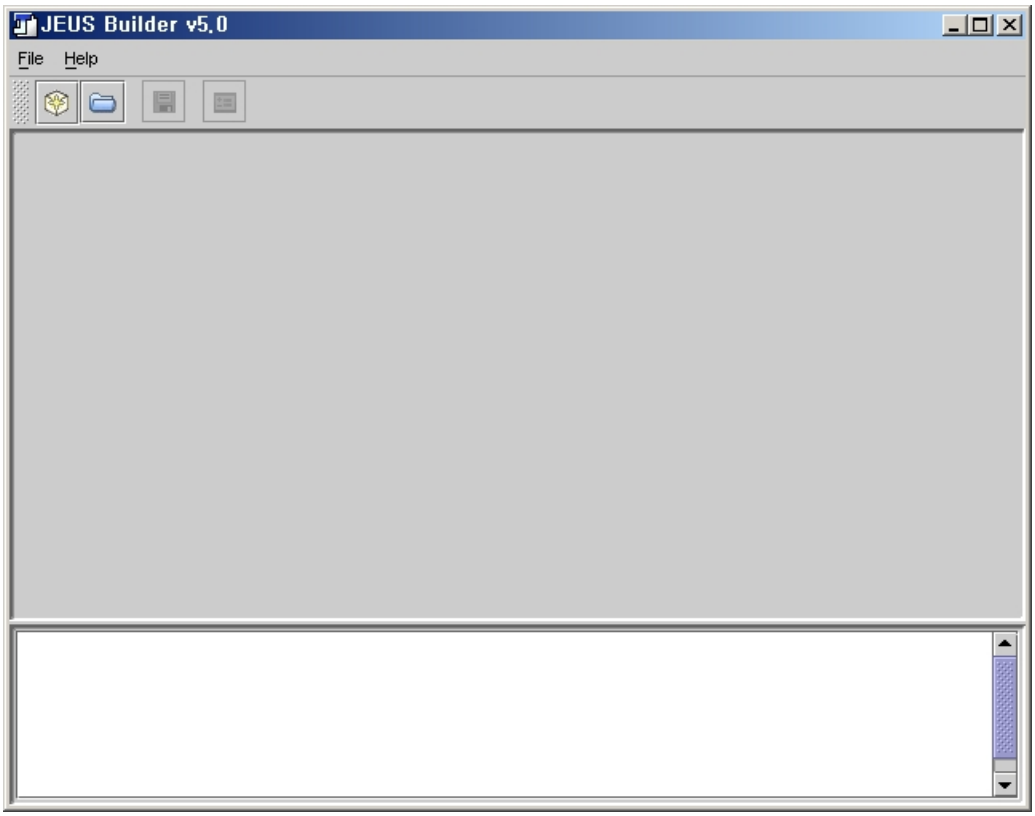

그림 22 JEUS Builder 초기화면

2. 새로운 모듈을 생성하기 위해 File 메뉴에서 New 버튼을 클릭하거나, 아이콘 메뉴에서 New Module 을 선택하면, File Contents 윈도우가 나타난다.

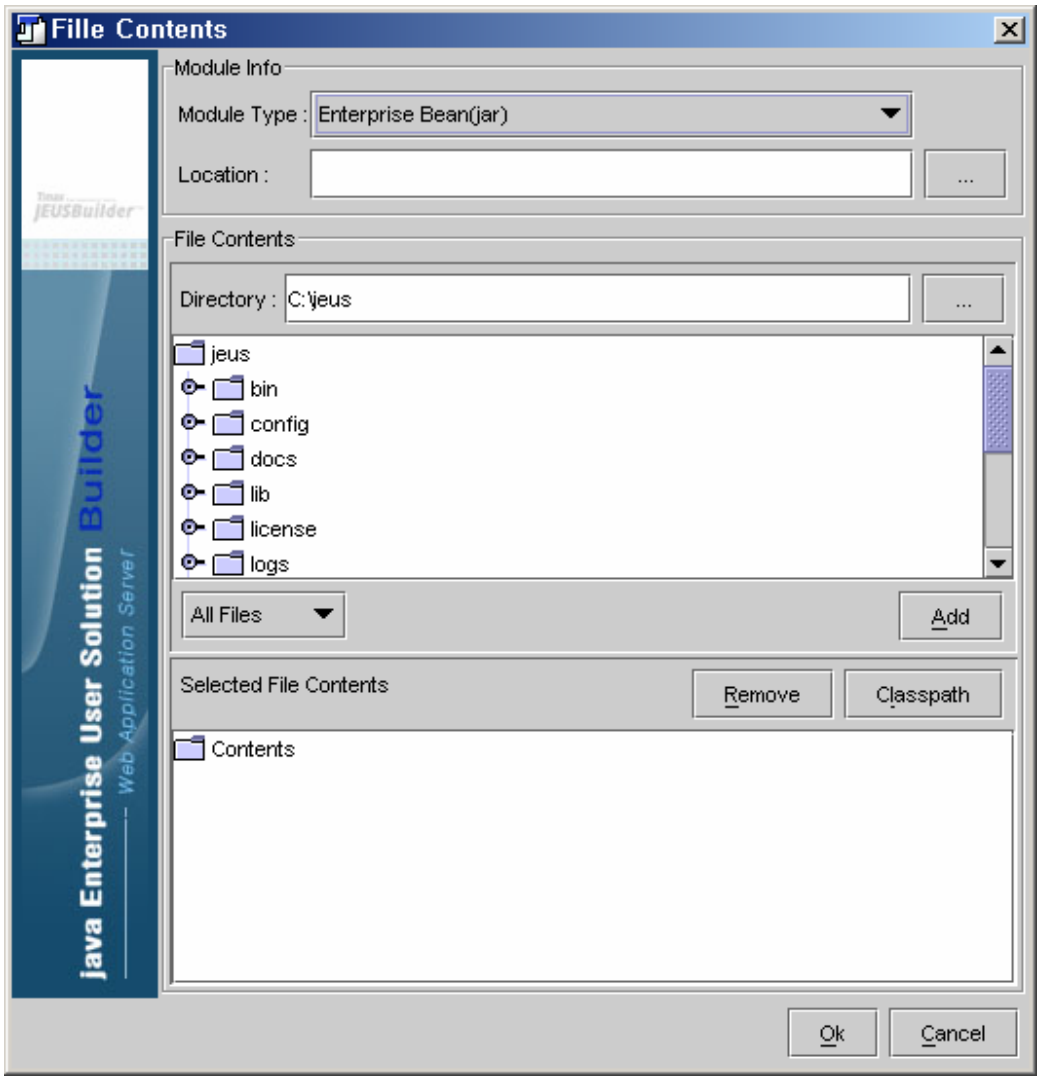

그림 23 Session EJB 모듈 파일 구성

3. File Contents 윈도우에서 아래와 같이 입력하고, 'Ok' 버튼을 클릭한다.

Module Type: EJB 모듈을 패키징할 것이므로, 여기에서 'Enterprise Bean(jar)'을 선택한다.

Location: 패키징할 모듈이 저장될 위치를 입력한다

Directory: 패키징될 파일들이 위치한 디렉토리를 입력하면, 아래쪽 트리에서 파일을 선택할 수 있다. 파일을 선택하고, 'Add' 버튼을 클릭하면 선택된 파일 정보가 'Selected File Contents'에 추가된다.

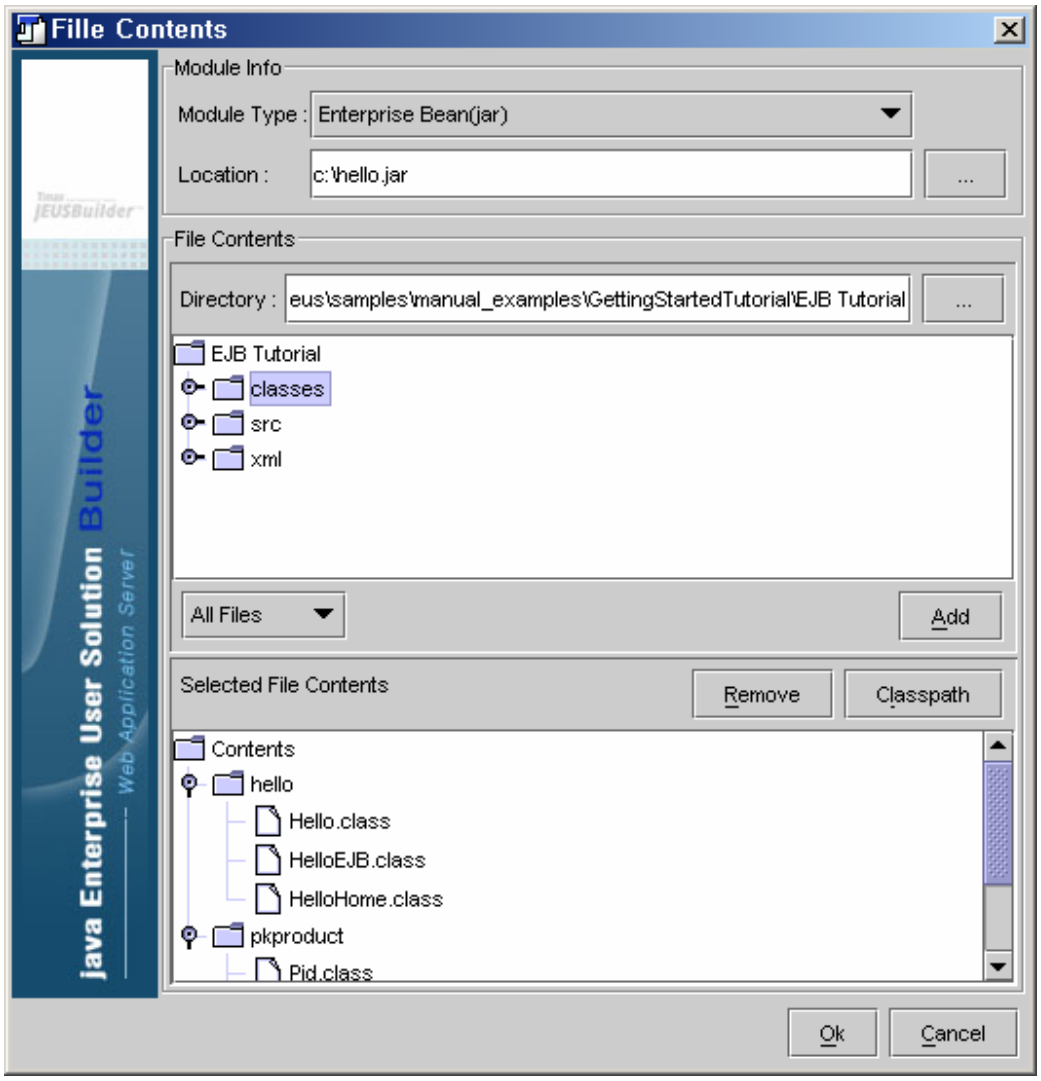

그림 24 Session EJB 모듈 파일선택

4. 새로 생성할 모듈에 대한 파일이 구성되었고, 모듈에 대한 설정을 할 수 있는 화면이 나타난다. 새로 보이는 view 는 Module View 로서, 현재 작업 중인 패키지명과 Deployment Descript 파일(이하 DD 파일)에 설정된 EJB 이름으로 구성된다.

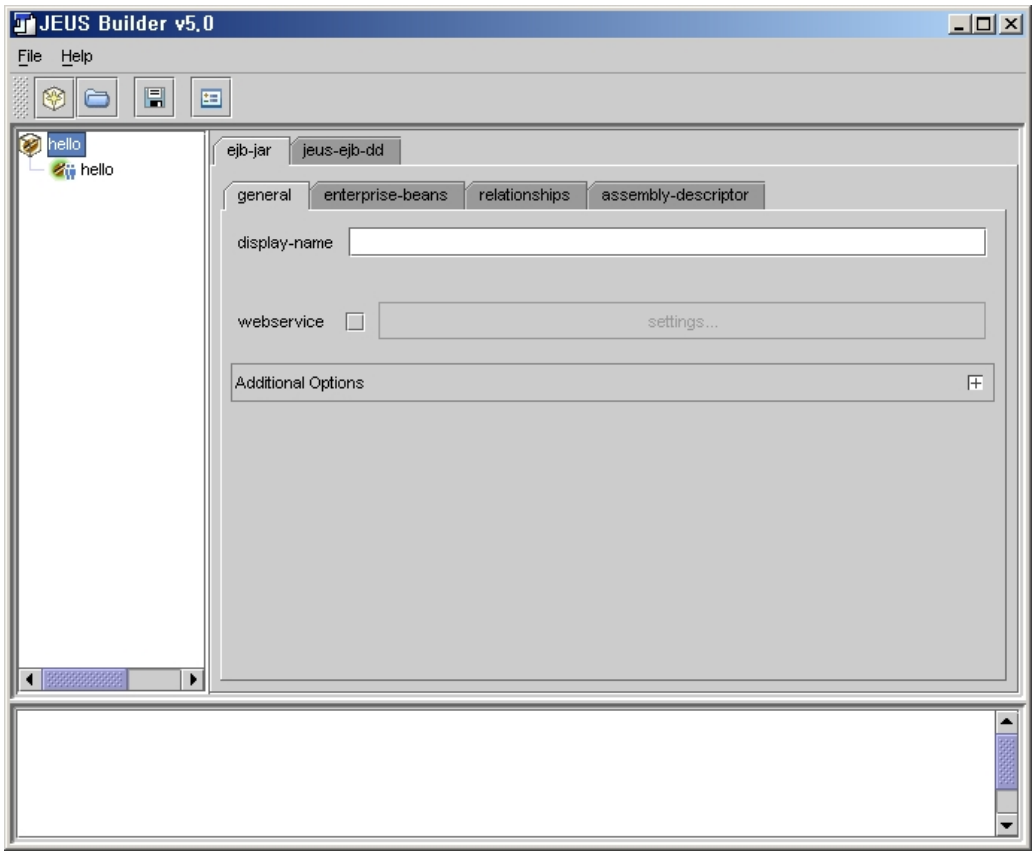

그림 25 Session EJB DD 설정화면

5. 모듈에 포함될 파일에 DD가 없을 경우에도 클래스 파일에서 DD 정보를 추출해서 보여준다. Workspace view 에서 ejb-jar.xml, jeus-ejb-dd.xml 을 수정할 수 있는 두 개의 탭이 존재한다. 패키징할 모듈은 기본적인 정보만 필요하기 때문에 추가적인 입력은 필요 없다.

6. File 메뉴에서 Save 버튼을 클릭하거나, 아이콘 메뉴에서 'Save Module' 버튼을 클릭하면, 지정된 위치에 모듈이 생성된다.

3.2.4 웹 관리자로 EJB 모듈 Deploy 하기

패키징한 EJB 는 웹 관리자와 콘솔 툴로 Deploy 할 수 있다. 이 절에서는 웹 관리자로 Deploy 하는 방법을 알아보도록 하겠다.

1. 커맨드 창에서 `'jeus'` 명령을 실행시키고, 웹 브라우저로 접속한다.

2. Node View 에서 Deploy 하고자 하는 EJB 엔진이 있는 엔진 컨테이너의 'J2EE 어플리케이션 모듈'을 선택한다

3. Main View 에 모듈 Deploy 를 위한 초기 화면이 나타난다. 탭에 보이는 것과 같이 Deploy 할 대상 모듈을 선택하는 화면으로, 3 가지 방법을 제공한다

   모듈: %JEUS_HOME%\webhome\app_home 이나 %JEUS_HOME%\webhome\deploy_home 아래 있는 모듈을 보여주며, 이들 중 하나를 선택해 Deploy 할 수 있다.

   절대 경로: Deploy 할 모듈이 존재하는 위치를 파일명을 포함해 지정한다. 모듈의 위치는 웹 관리자가 실행되는 머신이 아니고 JEUS 가 구동되는 머신의 디렉토리이다.

   파일 업로드: 웹 관리자가 실행되는 머신에서 JEUS 가 구동되는 머신으로 파일을 업로드 할 수 있으며 업로드할 위치는 %JEUS_HOME%\webhome\app_home 이나 %JEUS_HOME%\webhome\deploy_home 중 하나를 선택할 수 있다.

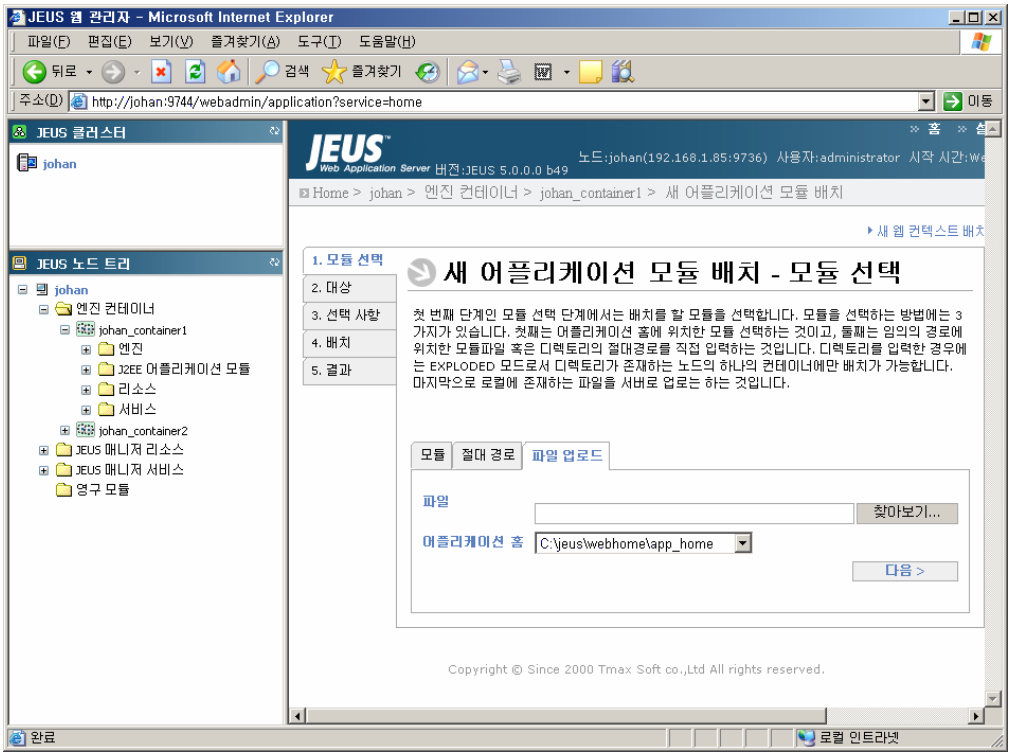

그림 26 Session EJB 모듈 deploy 초기화면

4. 절대경로나 파일 업로드에서 모듈을 선택하고 다음 버튼을 클릭하면, Deploy 할 대상이 엔진 컨테이너 단위로 나타난다. Deploy 할 대상을 선택하고 다음 버튼을 클릭한다

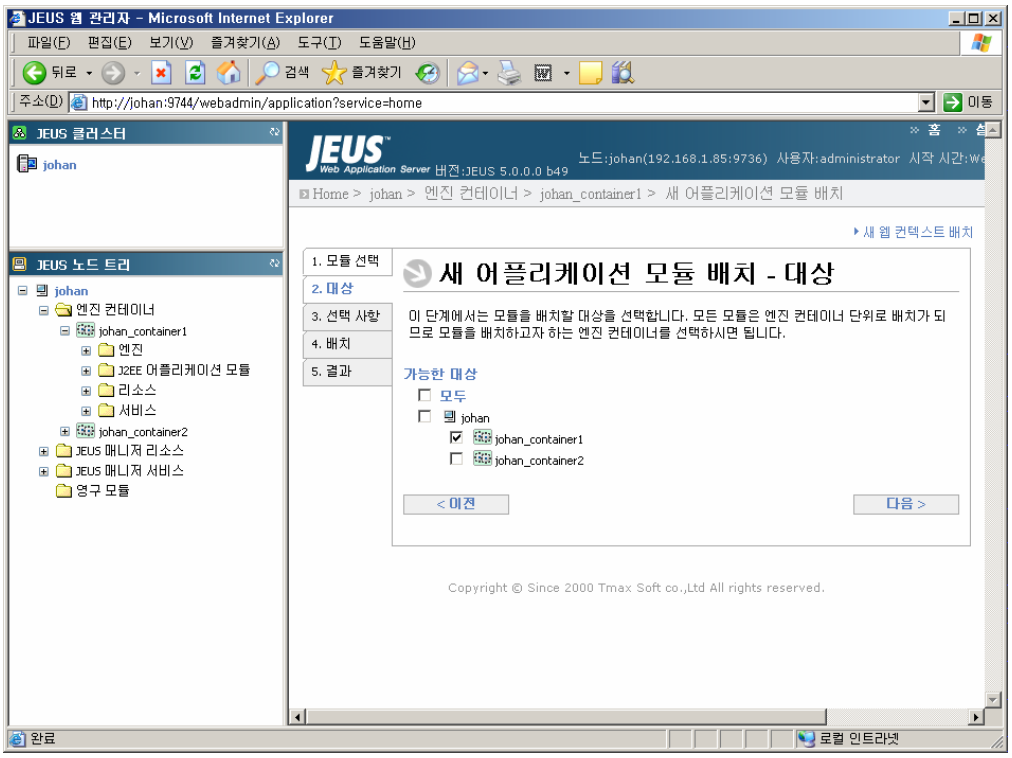

그림 27 Session EJB 모듈 deploy 대상 선택

5. Deploy 시 선택사항을 입력하는 화면으로, Deploy 가 되는 과정에 적용되는 설정과 Deploy 후 적용되는 설정으로 구분할 수 있다. Deploy 되는 과정에서 적용되는 설정은 keep generated, fast deploy, class loading 이고, Deploy 후 적용되는 설정은 security permission, auto deploy check interval 이 있다. 특별한 설정이 필요 없으므로 다음 버튼을 클릭한다.

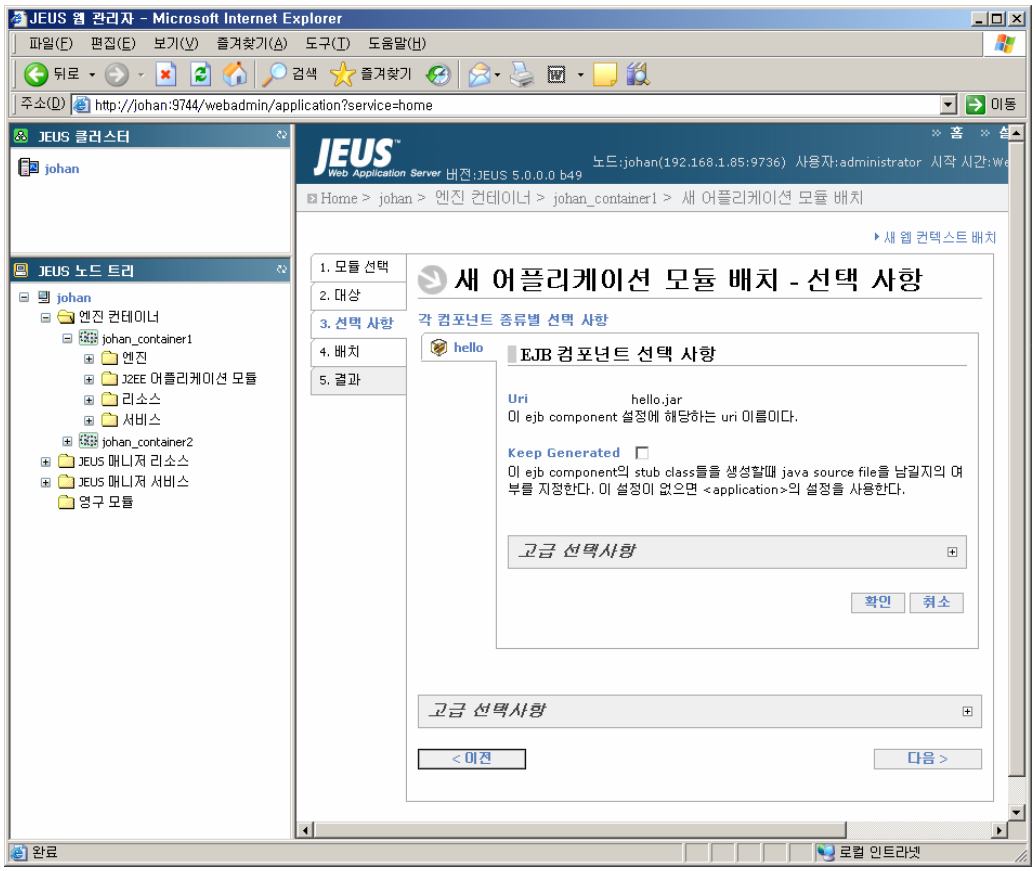

그림 28 Session EJB 모듈 deploy 시 선택사항

6. 배치에 대한 설정화면으로, 두 개 이상의 EJB 엔진에 Deploy 할 경우 2 단계 배치를 설정할 수 있다. 그리고, 다음 부팅할 때부터 자동으로 Deploy 되도록 하는 영구적인 배치를 설정할 수 있다.

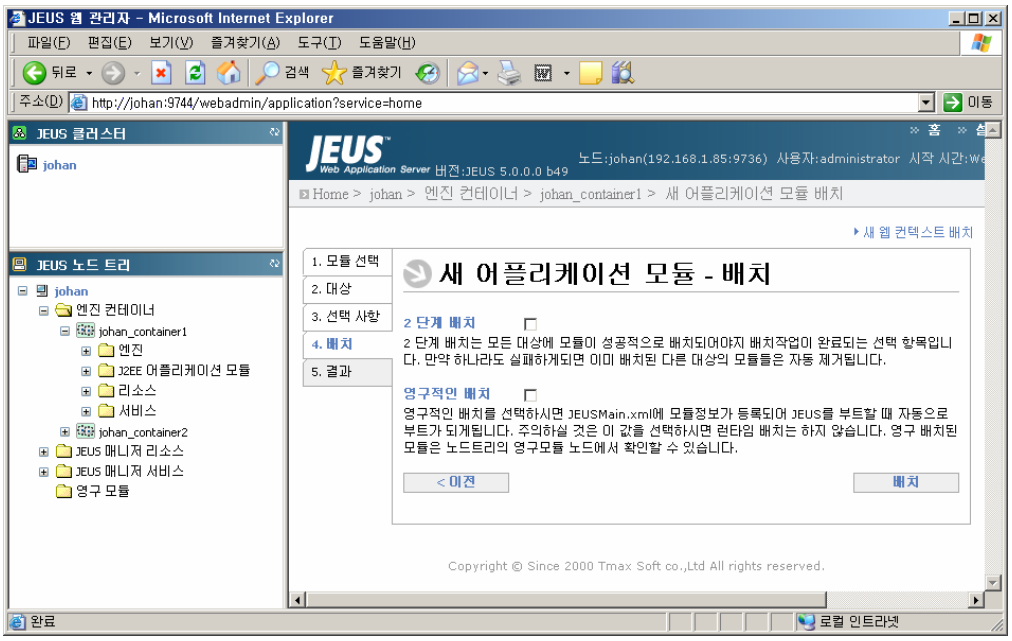

그림 29 Session EJB 모듈 deploy 옵션 선택화면

7. 배치 버튼을 클릭하면 선택된 엔진에 Deploy 를 시도하며, 성공여부 메시지를 출력한다. Deploy 가 성공적으로 완료되었을 경우 'J2EE 어플리케이션 모듈' 아래 나타난다.

3.2.5 EJB 모듈 수동으로 Deploy 하기

이번 절에서는 콘솔 툴을 사용하여 수동으로 Deploy 하는 방법을 설명한다.

1. 우선 ejb-jar.xml 파일과 jeus-ejb-dd.xml 파일을 생성한다(ejb-jar.xml 파일은 J2EE 의 EJB 표준 DD 파일이고, 후자인 jeus-ejb-dd.xml 파일은 JEUS 만의 고유한 DD 파일이다). 파일은 text editor 를 사용하여 작성하거나 `ddinit` 콘솔 툴을 사용하여 작성할 수 있다

   참고: `ddinit` 콘솔 툴을 사용할 경우 '`ddinit <source-archive-file` 또는 `directory>`' 과 같이 입력한다. 디렉토리를 지정할때는 해당 패키지의 시작 디렉토리 이전까지 지정하면된다. 예를 들어 패키지명이 'hello' 이고 c:\temp\hello 디렉토리 아래 클래스 파일이 있는 경우 'c:\temp' 까지만 지정해 주면된다.

   예)

```
c:\ddinit c:\temp
parameters setting successful..
DD Init Started..
```

```
Loading classes
classes are found under basedir = c:\temp
Creating descriptors
found EJBHome = hello.HelloHome
hello.HelloHome
hello.Hello
hello.HelloEJB

Creating jeus descriptors
DD Init finished..
```

생성된 xml 파일의 내용은 다음과 같다

<<ejb-jar.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee">
    <enterprise-beans>
        <session>
            <ejb-name>hello.HelloEJB</ejb-name>
            <home>hello.HelloHome</home>
            <remote>hello.Hello</remote>
            <ejb-class>hello.HelloEJB</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor/>
</ejb-jar>
```

<<jeus-ejb-dd.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info/>
    <beanlist>
        <jeus-bean>
            <ejb-name>hello.HelloEJB</ejb-name>
            <export-name>hello.HelloHome</export-name>
            <export-port>0</export-port>
```

```
            <single-vm-only>false</single-vm-only>
            <local-invoke-optimize>true</local-invoke-optimize>
            <thread-max>100</thread-max>
            <object-management>
                <bean-pool>
                    <pool-min>0</pool-min>
                    <pool-max>100</pool-max>
                </bean-pool>
                <connect-pool>
                    <pool-min>0</pool-min>
                    <pool-max>100</pool-max>
                </connect-pool>
                <capacity>10000</capacity>
                <passivation-timeout>-1</passivation-timeout>
                <disconnect-timeout>-1</disconnect-timeout>
            </object-management>
        </jeus-bean>
    </beanlist>
</jeus-ejb-dd>
```

2. 컴파일된 EJB 파일과 xml 파일을 각각 다음과 같은 디렉토리 아래 위치시킨다.

   EJB 파일: %JEUS_HOME%\webhome\app_home\<모듈명>\

   XML 파일: %JEUS_HOME%\webhome\app_home\<모듈명>\META-INF

   예)

   EJB 파일: C:\jeus\webhome\app_home\hello\

   XML 파일: C:\jeus\webhome\app_home\hello\META-INF

   참고 1: JEUS 에서 EJB 모듈을 Deploy 할 경우, EJB 파일은 적어도 하나 이상의 자바 패키지 레벨을 가져야 한다. 그러므로 모듈에 포함되는 EJB 파일은 클래스파일 단위가 아닌 디렉토리 단위가 된다.

   참고 2: EJB 모듈을 Deploy 하는 방법은 jar 모드와 exploded 모드 2 가지가 제공된다. Deployer 가 Deploy 모드를 따로 선택할 필요 없이 모듈

이 %JEUS_HOME%\webhome\app_home 아래 jar 형태로 존재하느냐 디렉토리 형태로 존재하느냐에 따라 Deploy 방법을 구분된다. 위에 설명한 것은 exploded 모드에 해당된다.

3. 커맨드 창에서 'jeus' 스크립트를 실행하고, 'jeusadmin' 콘솔 툴에서 'boot'명령을 통해 JEUS 를 기동시킨다.

4. 부팅이 정상적으로 되었다면, 커맨드 창에서 'ejbadmin'이라고 입력하면 ejbadmin 콘솔 툴이 실행된다. 아래와 같은 순서로 명령을 입력한다

```
C:\>ejbadmin johan_container1
Login name>administrator
Password>
JEUS 5.0 EJB Engine Controller
johan_container1>deploy hello
using the following application info : applicationType
        path:hello
        deployment-type:COMPONENT
        class-ftp-unit:JAR
        ejb-component

johan_container1>modulelist
name : hello
        type : EJBModule     EngineContainer :
johan_container1     node : johan
johan_container1>
johan_container1>beanlist hello
hello.HelloEJB
```

위의 명령을 살펴보면 'deploy <module name>'을 통해서 EJB module 을 deploy 하고, deploy 된 모듈을 확인하기 위해서는 'modulelist'와, module 내에 포함된 bean 들의 정보를 보기 위해서는 'beanlist <modulename>' 명령어를 이용하는 것을 알 수 있다. 일단 deploy 가 정상적으로 이루어졌다면 'exit' 명령을 통하여 ejbadmin 스크립트를 빠져 나올 수 있다.

이상으로 EJB module 은 지정된 EJB 엔진에 deploy 된 상태로, client 의 요청을 처리하기 위한 준비를 마쳤다.

다음 절에서는 “HelloWorld” Session bean 의 서비스를 이용하는 EJB client 프로그램을 작성하는 방법에 대해서 살펴본다.

3.2.6 EJB Client 예제 코드

이번 절에서는 EJB 엔진에 deploy 된 Session EJB 의 서비스를 이용하는 client 프로그램을 어떻게 작성하고, 컴파일하며, 실행할 것인가에 관해 살펴본다.

EJB client 는 자바 어플리케이션이 될 수도 있으며, Servlet 이나 JSP 혹은 다른 bean 이 될 수도 있다. 아래의 예제에서는 자바 어플리케이션으로 작성된 클라이언트 프로그램을 이용한다.

아래의 HelloClient.java 파일이 이미 우리가 이전 장에서 deploy 한 “hello” Session Bean 을 이용하는 client 코드의 소스이다.

<<HelloClient.java>>

```
package hello;

import java.rmi.server.*;
import java.lang.*;
import java.rmi.*;
import java.net.*;
import javax.ejb.*;
import javax.naming.*;

public class HelloClient {



    HelloHome home = null;
    Hello obj = null;

    private void run() {
      try {
          InitialContext ctx = new InitialContext();

          // basic test
          home = (HelloHome)ctx.lookup("HelloApp");
          System.out.println("lookup done");
          obj = (Hello)home.create();
          System.out.println("create()" + obj);
```

```
            String s = obj.sayHello();
            System.out.println("method()" + s);
        } catch (Exception e) {
              e.printStackTrace();
        }
    }

    public static void main(String args[]) {
        HelloClient hclient = new HelloClient();
        hclient.run();
    }
}
```

3.2.7 EJB Client 예제코드 컴파일

EJB 모듈이 성공적으로 Deploy 될 경우 %JEUS_HOME%\webhome\ <node name>_<container name> 아래 모듈명에 해당하는 디렉토리가 생성된다. 이 디렉토리 아래에는 EJB Remote, Home 클래스를 구현한 클래스와 각각의 스텁/스켈레톤 클래스들이 생성된다.

생성된 스텁/스켈레톤 클래스는 EJB Client 가 컴파일하고 실행하는 과정에서 필요하게 된다. 이 파일들을 client.jar 로 묶어서 클라이언트 어플리케이션을 컴파일하고 실행하는 과정에서 client 가 인식할 수 있도록 classpath 에 추가시켜준다.

예)

```
C:\jeus\webhome\johan_container1\hello\hello>jar cvf
C:\client.jar hello/
```

EJB client 프로그램을 컴파일하기 위해서는 client java file 이 있는 디렉토리로 이동후 아래처럼 실행하면 된다.

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\EJBTutoria
l\src\client\hello>javac -classpath C:\jeus\lib\system\jeus.jar;
c:\client.jar -d ..\..\classes HelloClient.java
```

예제에서 client 코드가 “hello”라는 패키지 구조를 가지므로 실제 컴파일을 하면 “hello”라는 디렉토리 아래에 HelloClient.class 파일이 생성된다.

3.2.8 JEUS Builder 로 EJB Client 모듈 패키징하기

JEUS Builder 를 사용하여 작성된 client application 을 패키징 한다.

1. 커맨드 창에서 jeusbuilder 스크립트를 실행시킨다.

2. 새로운 모듈을 생성하기 위해 File 메뉴에서 New 버튼을 클릭하거나, 아이콘 메뉴에서 New Module 을 선택하면, File Contents 윈도우가 나타난다.

3. File Contents 윈도우에서 Module Type 을 'Application Client (car)'로 선택한다. 모듈이 생성될 위치를 선택하고, 컴파일한 HelloClient.class 파일을 추가시킨다. 'Ok' 버튼을 클릭하면 설정화면으로 넘어간다

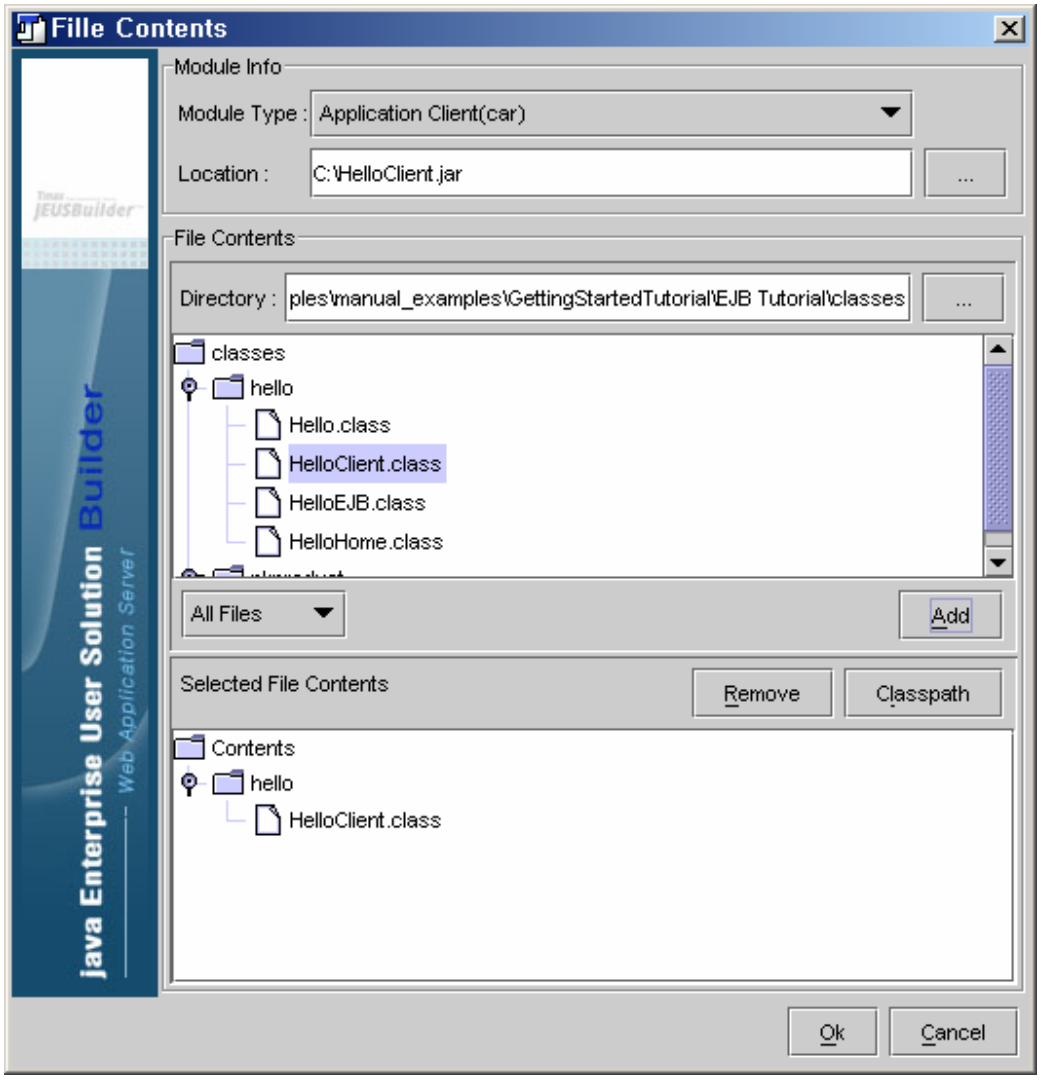

그림 30 Session EJB Client 모듈 파일 선택

4. 먼저 application-client.xml 파일을 설정한다. general 에서 display-name 을 입력한다.

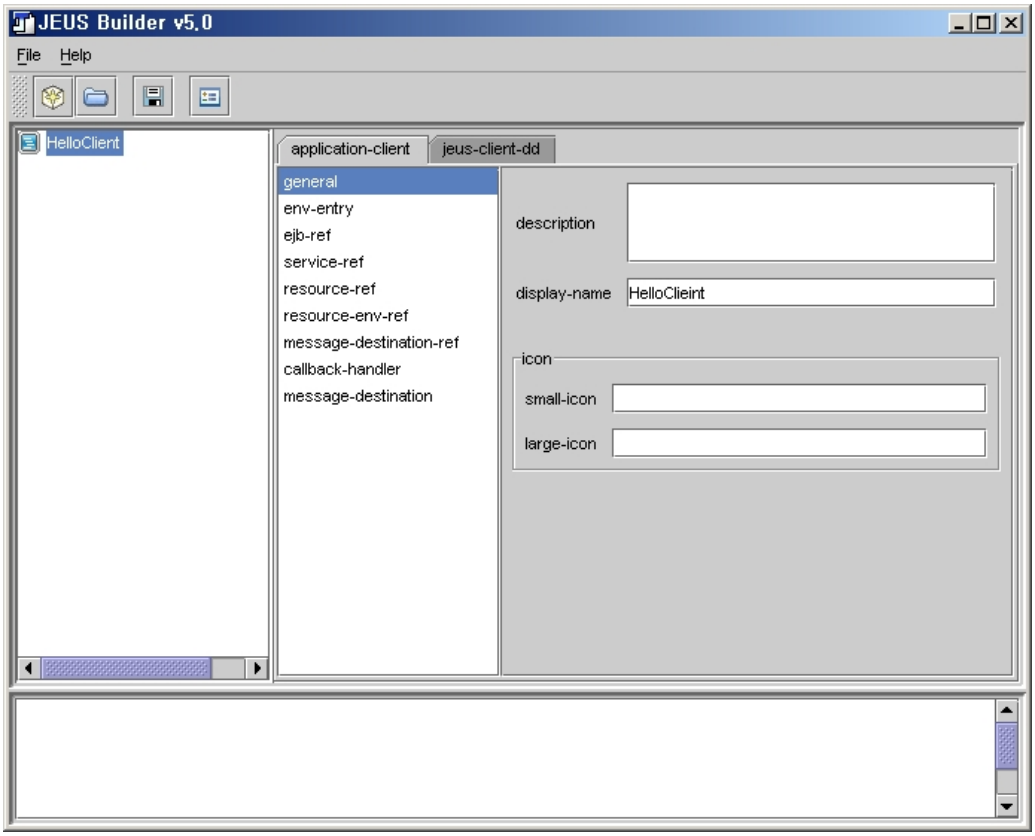

그림 31 Session EJB 모듈 DD 설정화면

ejb-ref-name 을 사용할 경우 ejb-ref 를 선택하고, 다음과 같이 입력한다.

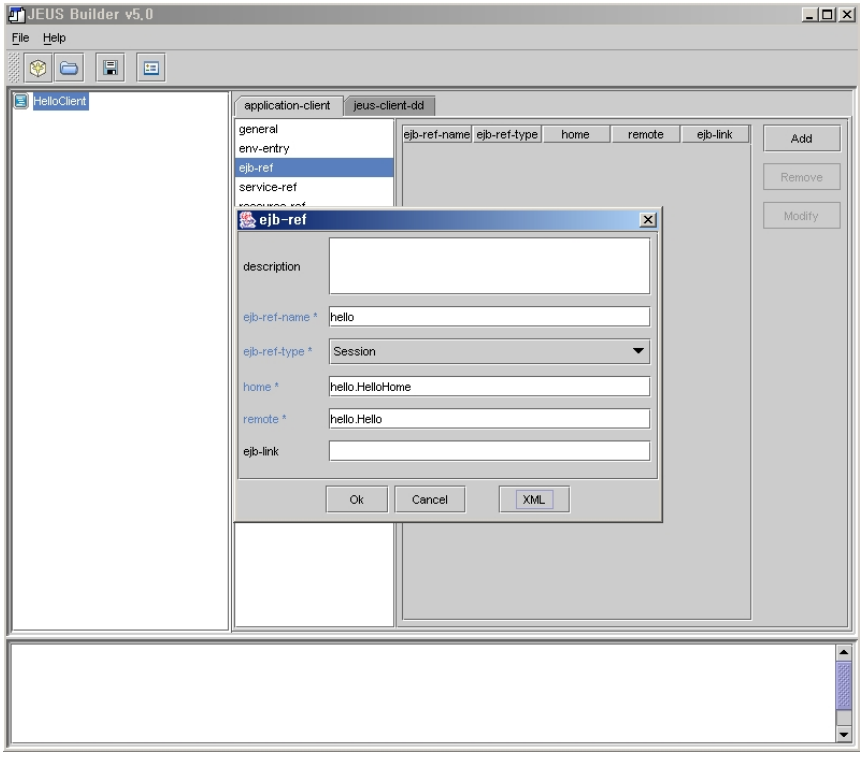

그림 32 Session EJB 모듈 ejb-ref 설정

ejb-ref-name 은 export-name 에 대한 reference name 을 설정할 수 있도록 하며, JNDI lookup 할 때 사용할 수 있다.

5. jeus-cilent-dd.xml 파일을 설정하기 위해, jeus-client-dd 탭을 선택한다. 여기서 필수적으로 입력해야 할 사항은 module-info 에 해당하는 항목들이다.

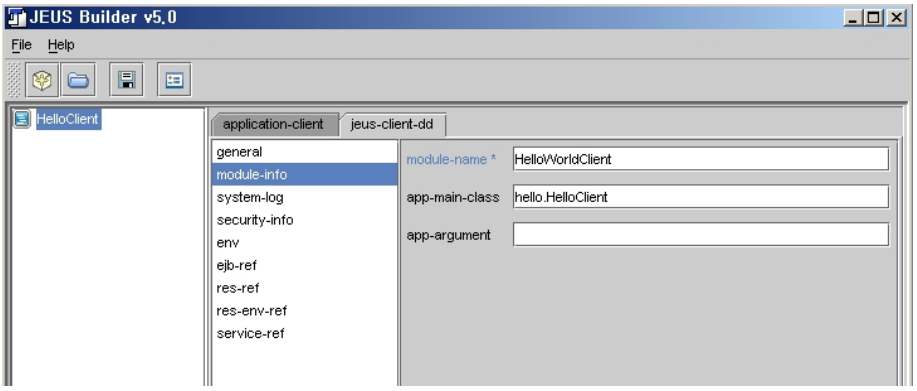

그림 33 Session EJB 모듈 module-info 설정

6. application-client 에서 ejb-ref-name 을 설정했다면, jeus-client-dd 의 ejb-ref 를 설정해야 한다. ejb-ref 를 클릭해보면 application-client 에서 설정한 ref-name 이 추가되어 있을 것이다. Modify 버튼을 클릭해서 export-name 을 입력한다.

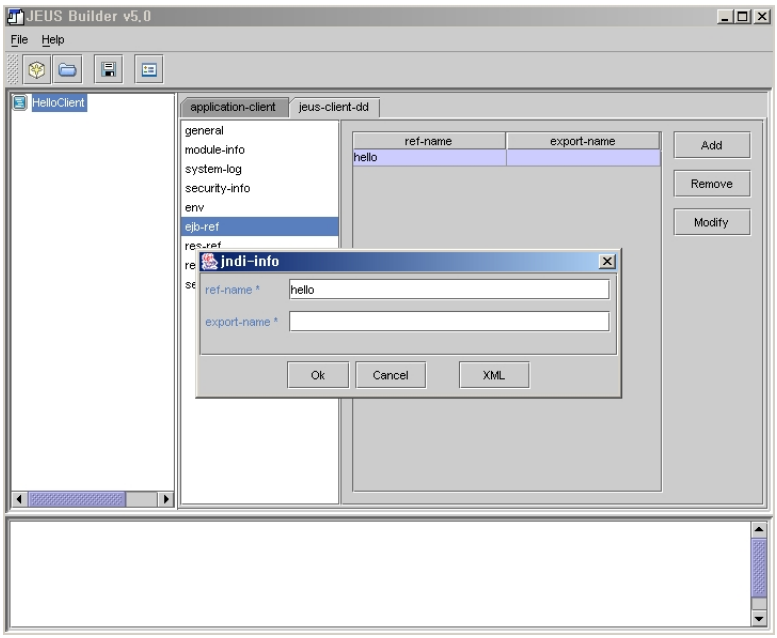

그림 34 Session EJB 모듈 export-name 설정

7. 저장버튼을 클릭하면 지정된 위치에 application client 모듈이 생성된다.

3.2.9 웹 관리자로 EJB Client 모듈 Deploy 하기

패키징이 정상적으로 완료되었을 경우, 아래의 절차에 따라 client application 을 deploy 할 수 있다.

1. 웹 관리자를 실행 시킨다.

2. 엔진 컨테이너 아래 'J2EE 어플리케이션 모듈'을 선택한다.

3. Deploy 할 모듈을 선택하고 다음 버튼을 클릭한다.

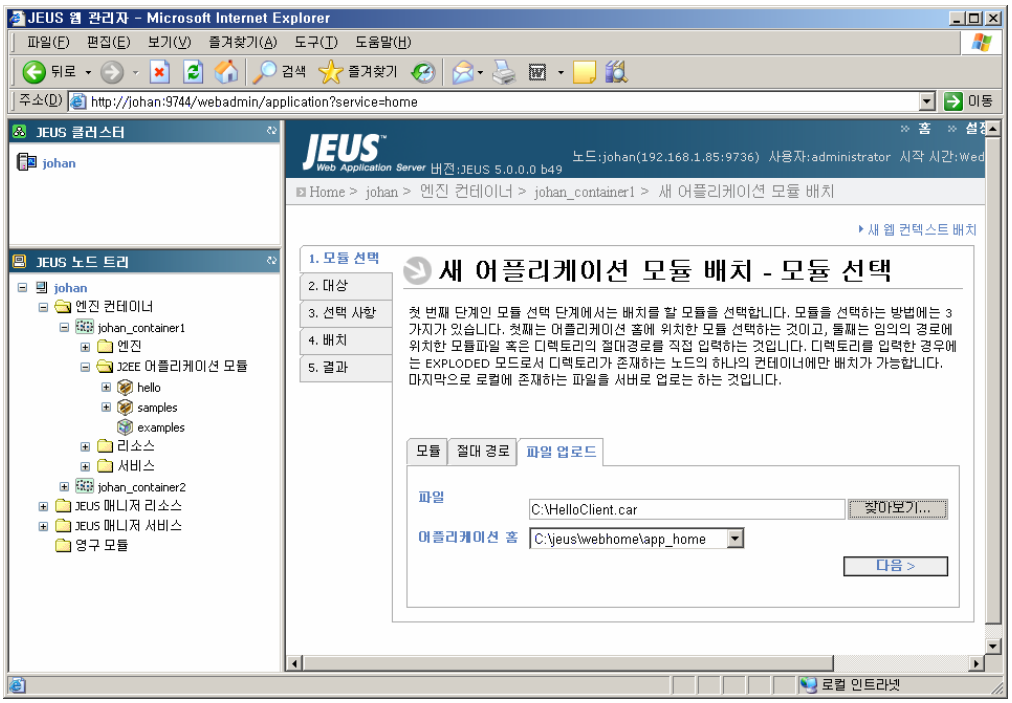

그림 35 Session EJB Client 모듈 deploy 초기화면

4. Deploy 할 컨테이너를 선택하고 다음 버튼을 클릭한다.

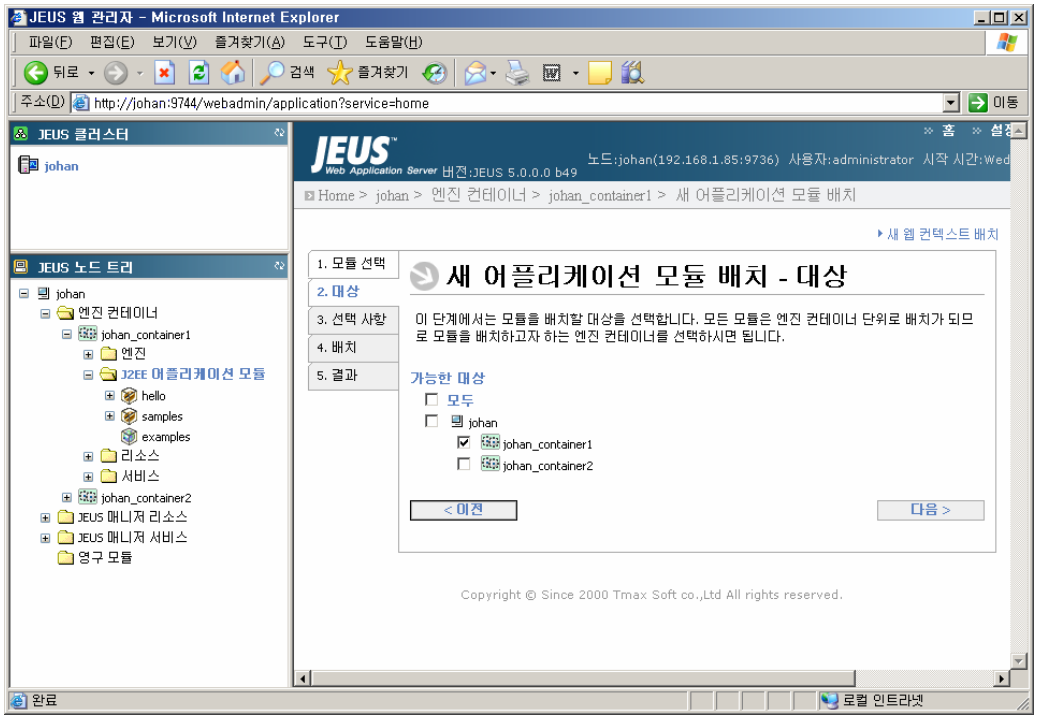

그림 36 Session EJB Client 모듈 deploy 대상 선택

5. 다음 버튼을 클릭한다.

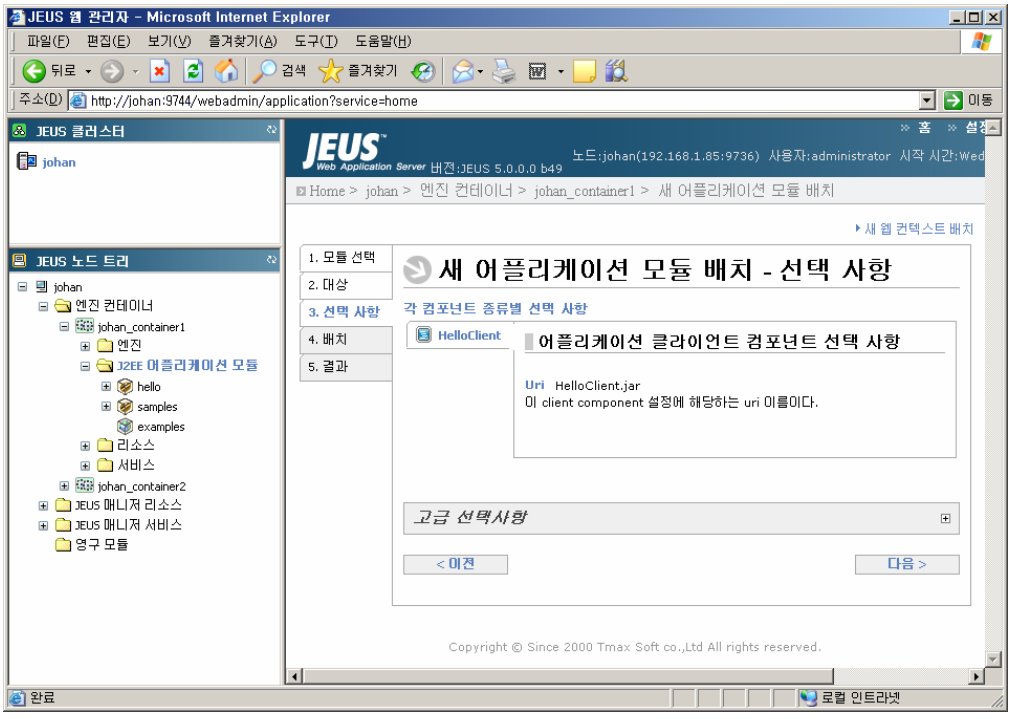

그림 37 Session EJB Client 모듈 deploy 선택사항

6. 배치 옵션을 선택하고 배치 버튼을 클릭하면 Deploy 를 시도한다.

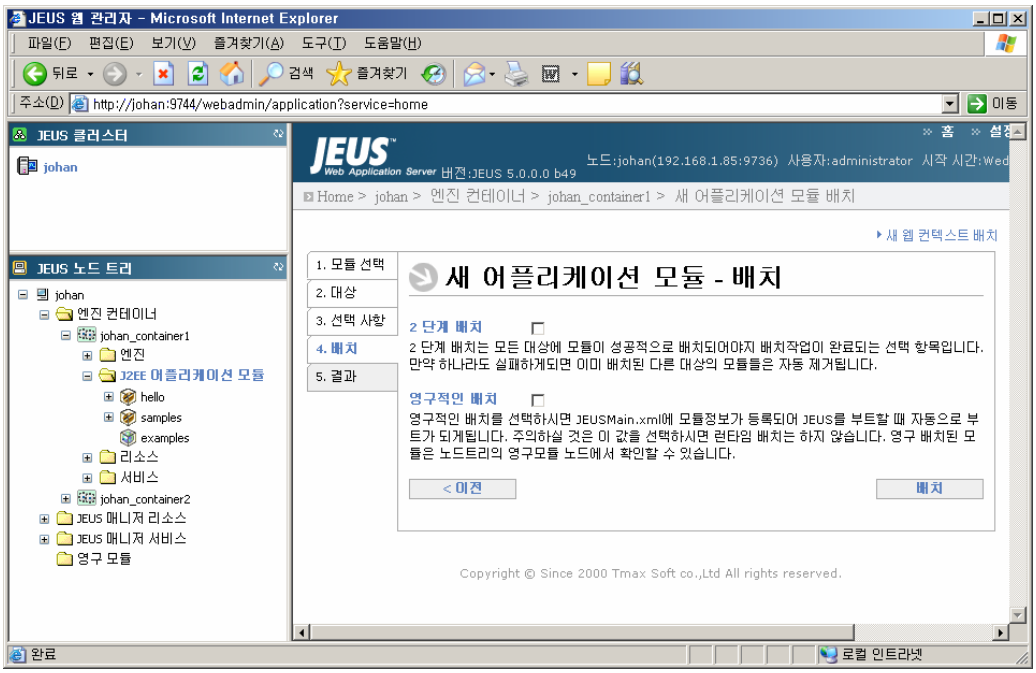

그림 38 Session EJB Client 모듈 deploy 옵션

7. Deploy 가 성공할 경우 'J2EE 어플리케이션 모듈' 아래 HelloClient 가 나타난다.

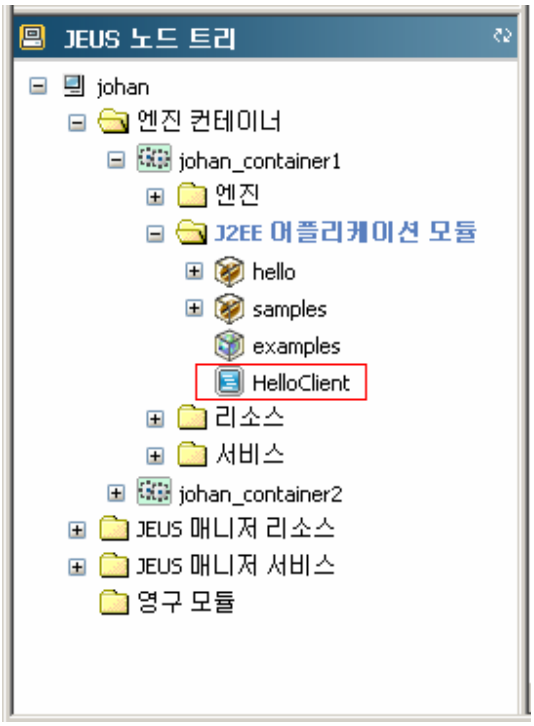

그림 39 Session EJB Client 모듈 deploy 결과

3.2.10 EJB Client 모듈 수동으로 Deploy 하기

만일 위에서 살펴본 바와 같이 웹 관리자를 사용하여 클라이언트 application 을 deploy 할 수 없는 경우에는, 아래처럼 직접 파일을 생성하고 deploy 할 수 있다.

1. 앞 절에 다룬 바와 같이 Client application 을 컴파일한다

2. J2EE application client DD 파일과 JEUS application client DD 파일을 아래처럼 생성한다.

<<application-client.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<application-client xmlns="http://java.sun.com/xml/ns/j2ee"
version="1.4">
    <ejb-ref>
        <ejb-ref-name>hello</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>hello.HelloHome</home>
```

```
        <remote>hello.Hello</remote>
    </ejb-ref>
</application-client>
```

<<jeus-client-dd.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-client-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
        <module-name>HelloWorldClient</module-name>
        <app-main-class>hello.HelloClient</app-main-class>
    </module-info>
    <scheduler>false</scheduler>
    <ejb-ref>
        <jndi-info>
            <ref-name>hello</ref-name>
            <export-name>HelloApp</export-name>
        </jndi-info>
    </ejb-ref>
</jeus-client-dd>
```

3. JAR 유틸리티를 이용하여 application-client.xml 파일과 HelloClient.class 파일을 묶는데, application-client.xml 파일은 META-INF\ 디렉토리 아래에 두어야 하며, HelloClient.class 파일 역시 패키지디렉토리인 “hello” 디렉토리 아래에 두고 압축하여야 한다. JAR utility 에 대한 자세한 설명은 http://java.sun.com 을 참조한다.

4. JAR 파일과 jeus-client-dd.xml 파일을 %JEUS_HOME%\webhome \client_home\<module name> 에 복사한다

3.2.11 EJB Client 모듈 실행 및 결과

runclient.cmd 스크립트는 EJB Client 모듈을 실행하기 위해 제공되며, 일부 내용을 수정함으로써 EJB Client 모듈을 실행시킬 수 있다.

1. EJB 의 Home 과 Remote Interface 의 클래스를, runclient.cmd 의 APP_CLASSPATH 에 `%JEUS_HOME%\webhome\app_home\hello.jar` 를 추가한다.

2. runclient.cmd 의 –client 옵션 다음에 EJB Client 모듈의 절대 경로를 입력한다. jar 모듈로 실행할 경우 모듈 경로 뒤에 –jar 옵션을 추가한다.

예)

```
runclient -client c:\HelloClient.car -jar
[2005.03.02 21:59:07][2] [Client-0028] client container is
started
[2005.03.02 21:59:07][2] [Client-0038] starting client
application [hello.HelloClient:]
lookup done
create()helloObjectImpl1111967647_Stub[UnicastRef [liveRef:
[endpoint:[192.168.1.85:4333](remote),objID:[151f910:1026226
6ab2:-8000, 3]]]]
method()Hello World!
```

3.2.12 결론

이번 절에서는 stateless session bean 과 Session Bean 을 요청하는 EJB client application 을 컴파일하고 패키징하여 deploy 하는 방법을 살펴보았다.

다음 장에서도 이와 유사한 과정이지만 Container Managed Entity Bean 을 사용하는 법에 대해서 알아보기로 한다.

3.3 Entity Bean Deploy

이 절에서는 Container Managed Entity Bean 을 작성하고 컴파일하여 deploy 하는 과정을 설명한다.

이 장에서 사용된 예제코드는 %JEUS_HOME%\samples\manual_examaples \GettingStarted Tutorial\EJBTutorial\src\bean\pkproduct 디렉토리 아래에 있다.

3.3.1 EJB 예제 코드

Entity bean 은 기본적으로 home 인터페이스와 remote 인터페이스, enterprise bean 클래스, 그리고 primary key 클래스로 구성된다. 아래의 java 코드는 CMP2.0 에 맞게 구현된 것이다.

Home Interface (ProductHome.java)

<<*ProductHome.java*>>

```
package pkproduct;
```

```
import java.util.Collection;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface ProductHome extends EJBHome {
    public Product create(Pid productID, double price, String
description) throws RemoteException, CreateException;
    public Product findByPrimaryKey(Pid productID) throws
FinderException, RemoteException;
    public Collection findByDescription(String description)
throws FinderException, RemoteException;
    public Collection findInRange(double low, double high) throws
FinderException, RemoteException;
}
```

Remote Interface (Product.java)

<<Product.java>>

```
package pkproduct;



import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Product extends EJBObject {
      public Pid getProductID() throws RemoteException;
      public double getPrice() throws RemoteException;
      public void setPrice(double price) throws RemoteException;
      public String getDescription() throws RemoteException;
      public void setDescription(String description) throws
RemoteException;
}
```

Enterprise Bean Class (ProductEJB.java)

<<ProductEJB.java>>

```
package pkproduct;

import java.util.*;
import javax.ejb.*;
```

```
public abstract class ProductEJB implements EntityBean {

    private EntityContext context;

      public abstract Pid getProductID();
      public abstract void setProductID(Pid productID);
      public abstract double getPrice();
      public abstract void setPrice(double price);
      public abstract String getDescription();
      public abstract void setDescription(String description);

    public Pid ejbCreate(Pid productID, double price, String
description) throws CreateException {
            if (productID == null) {
                  throw new CreateException("Pid is null.");
            }
            setProductID(productID);
            setPrice(price);
            setDescription(description);

            return null;
      }
    public void ejbPostCreate(Pid productID, double price, String
description) { }

      public void setEntityContext(EntityContext context)  {
            this.context = context;
      }
    public void unsetEntityContext() {
            this.context = null;
      }

public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() { }
    public void ejbLoad() { }
    public void ejbStore() { }
}
```

Primary Key Class (Pid.java)

<<Pid.java>>

```
package pkproduct;

import java.io.*;

public class Pid implements Serializable
{
    public String productID;

    public Pid()
    {
    }

    public Pid(String s)
    {
      productID = s;
    }

    public boolean equals(Object obj)
    {
      if(obj instanceof Pid)
            if( ((Pid)obj).productID.equals(this.productID) )
                  return true;

      return false;
    }

    public int hashCode()
    {
      return productID.hashCode();
    }
}
```

3.3.2 EJB 예제 코드 컴파일

EJB 예제 코드에서 사용하는 클래스들을 classpath 에 추가하고 소스파일을 컴파일 한다.

예)

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\EJB
Tutorial\src\bean\pkproduct>javac -classpath
C:\jeus\lib\system\jeus.jar -d ..\..\..\classes *.java
```

컴파일이 정상적으로 수행되면 '..\..\..\classes\pkproduct'라는 패키지 디렉토리 아래에 class 파일들이 생성된다.

3.3.3 JEUS Builder 로 EJB 모듈 패키징 하기

생성된 클래스 파일들을 패키징 해보도록 한다.

1. 커맨드 창에서 'jeusbuilder' 명령으로 JEUS Builder 를 실행시키고, File 메뉴에서 New 버튼을 클릭하거나, 아이콘 메뉴에서 New Module 을 선택한다.

2. 모듈 타입을 jar 로 설정하고, 모듈이 생성될 위치를 입력한다. 컴파일한 클래스파일을 추가하고 'Ok' 버튼을 클릭한다

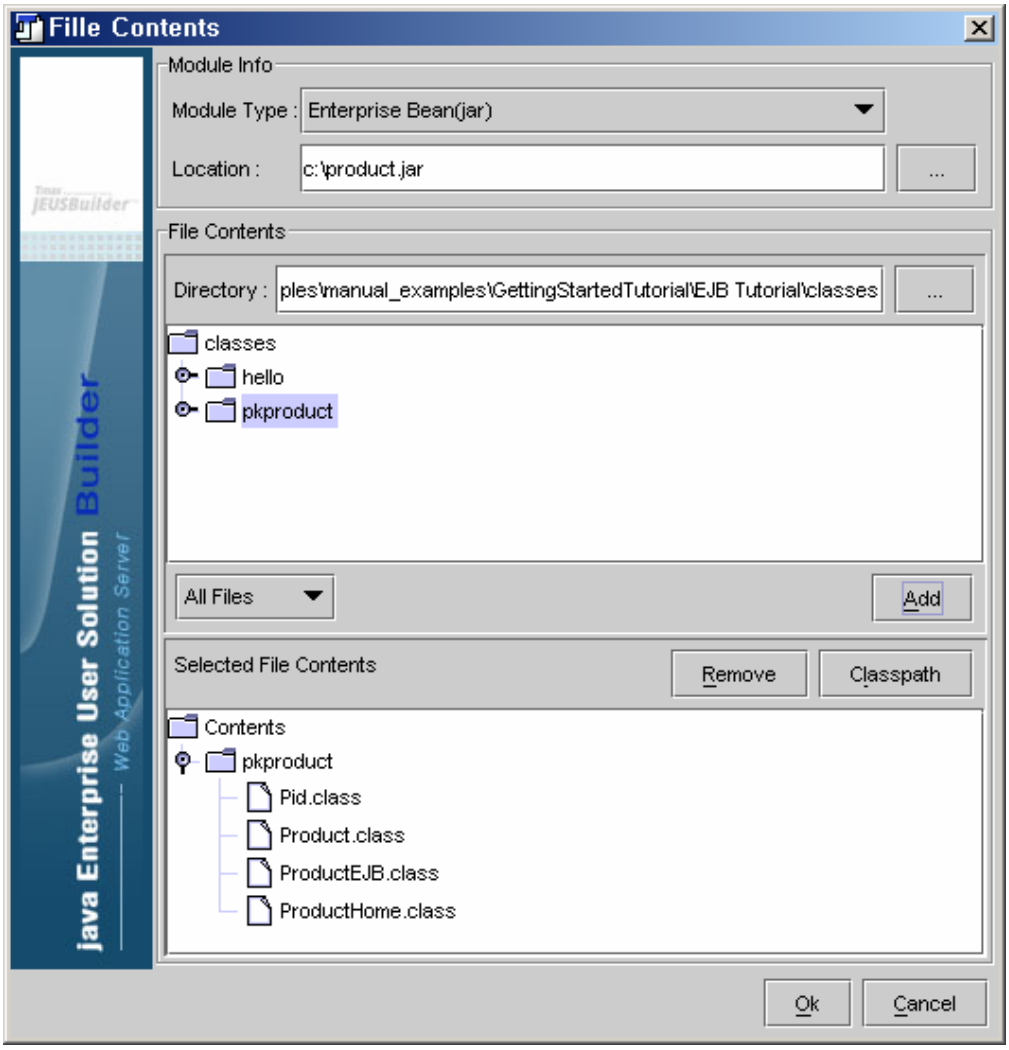

그림 40 Entity EJB 모듈 파일 추가

3. 클래스 파일만 추가할 경우 ejb-jar.xml 파일과 jeus-ejb-dd.xml 파일이 생성된다. 먼저 ejb-jar.xml 파일을 설정해 보도록 한다. general 탭에서 display-name 을 입력한다.

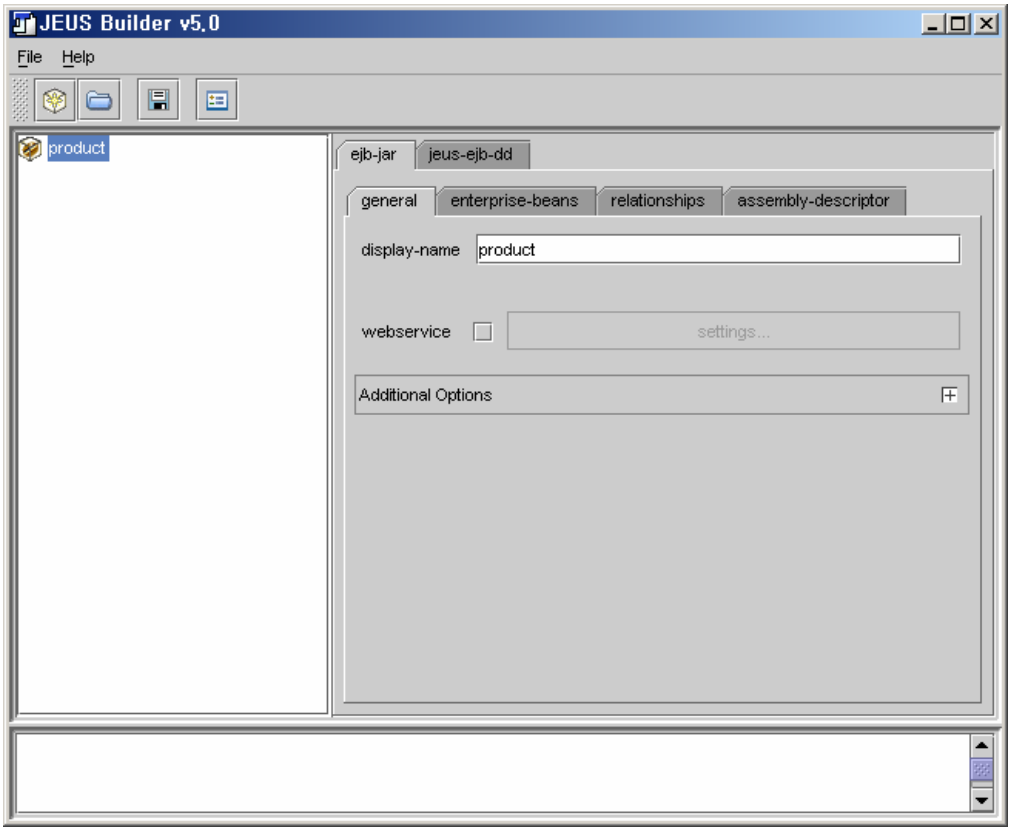

그림 41 Entity EJB 모듈 display-name

4. enterprise-beans 탭에서 엔티티빈을 선택하고 'Modify' 버튼을 클릭한다. 아래 화면은 Module View 에서 DD 파일에 설정된 EJB 이름을 선택하면 나오는 설정화면과 같다. ejb-name 을 'ProdutBean'으로 입력한다

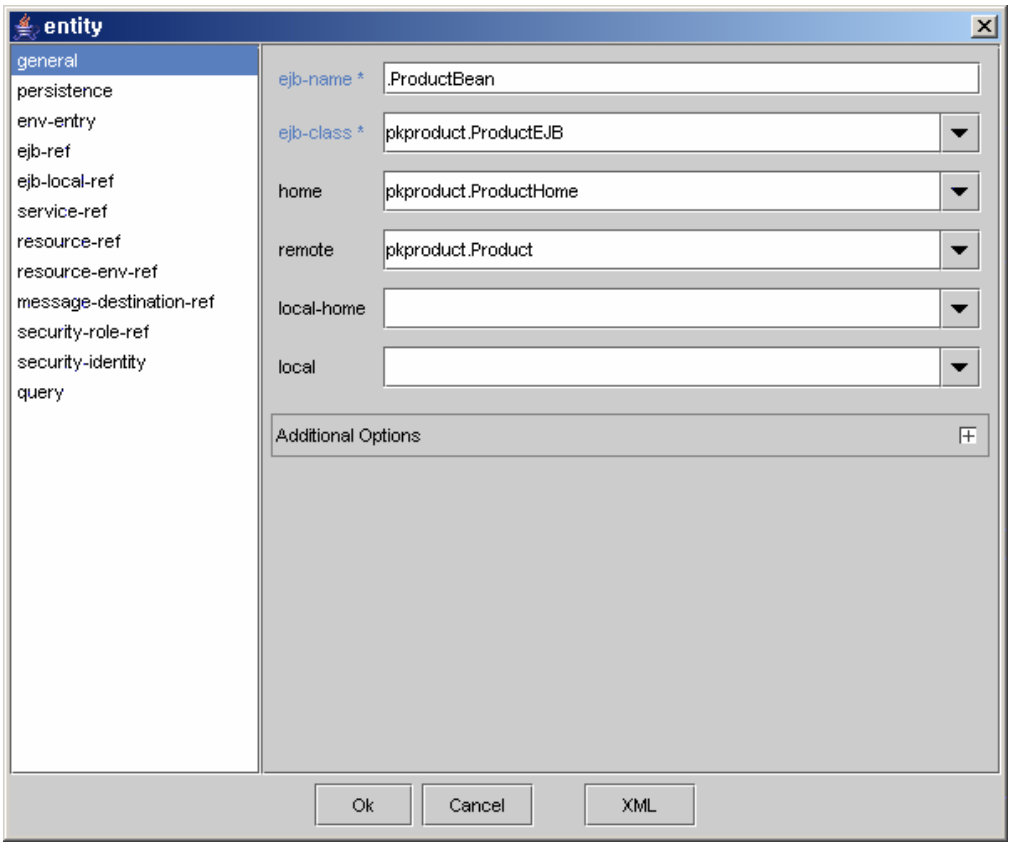

그림 42 Entity EJB 모듈 ejb-name

5. 왼쪽 윈도우에서 ‘persistence’를 선택하면 설정 화면이 나온다. 코드가 CMP 2.0 에 맞게 작성될 경우 기본적인 값들은 자동으로 채워진다.

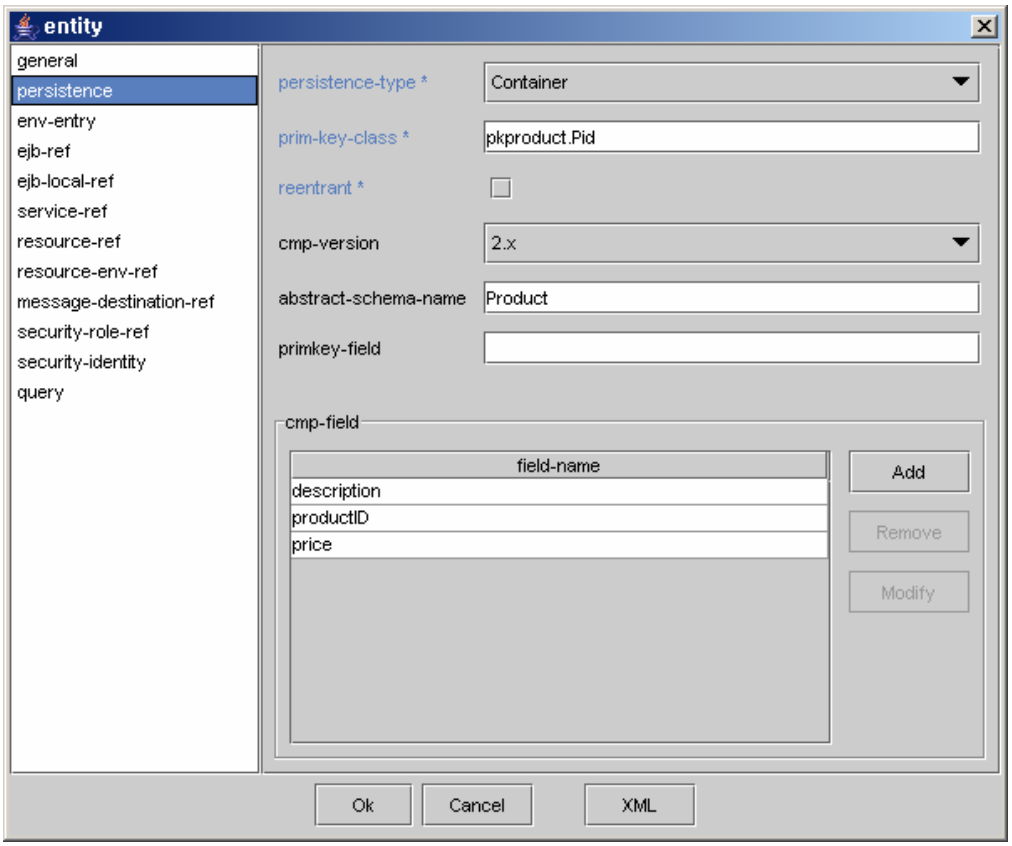

그림 43 Entity EJB 모듈 persistence

6. 다음으로 'query' 를 선택한다. EJB Home 인터페이스에 선언된 finder method 에 대한 EJB QL 을 설정할 수 있는 윈도우가 나타난다. findByPrimaryKey method 를 제외한 나머지 finder method 는 자동으로 설정된다. 각각 finder method 에 대해 result-type-mapping 을 'Remote'로 설정하고, ejb-ql 을 다음과 같이 입력한다.

```
findByDescription: SELECT Object(o) FROM Product o WHERE
o.description = ?1
findInRange: SELECT Object(o) FROM Product o WHERE o.price
between ?1 and ?2
```

참고: ?1, ?2 는 finder method 의 첫번째와 두 번째 parameter 에 해당된다.

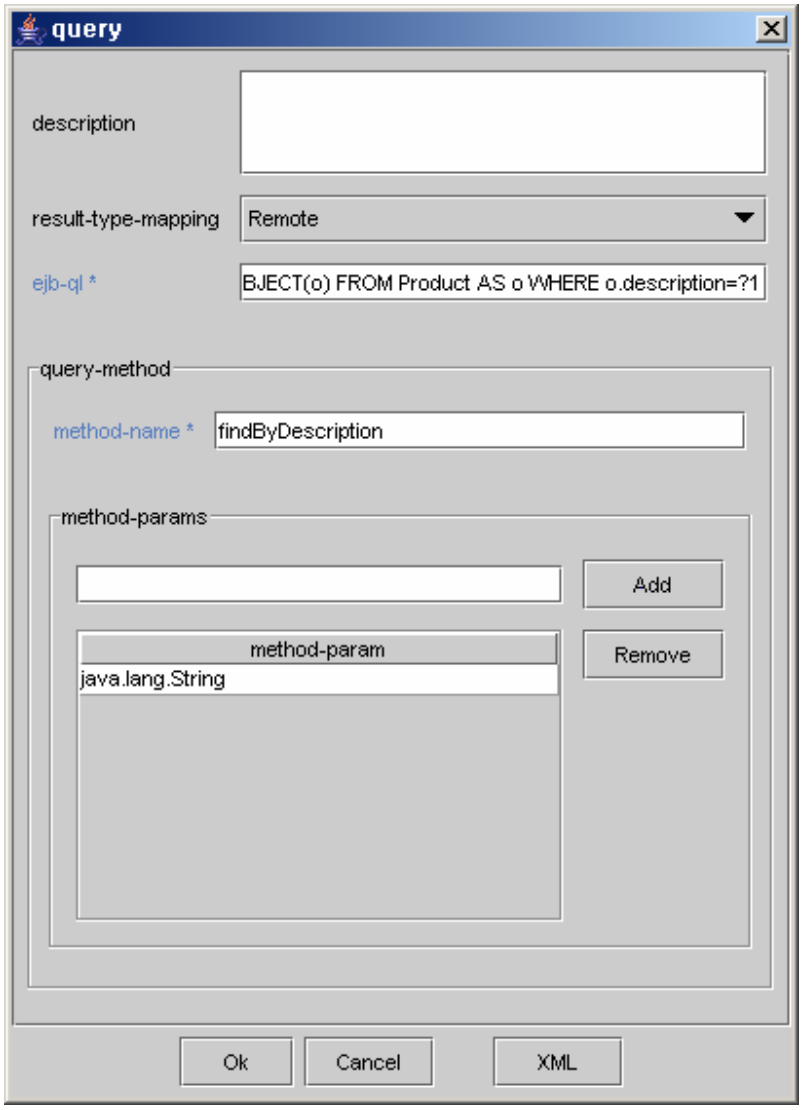

그림 44 Entity EJB 모듈 query

7. 떠 있는 창을 모두 닫고, assembly-descriptor 탭에서 container-transaction 을 선택한다.

8. transaction-attribute 에 Required 를 선택하고, 'Add'버튼을 누른다.

9. method 창이 뜨면 ejb-name 에 `'ProductBean'`을, method-name 에 `'*'`를 넣는다.

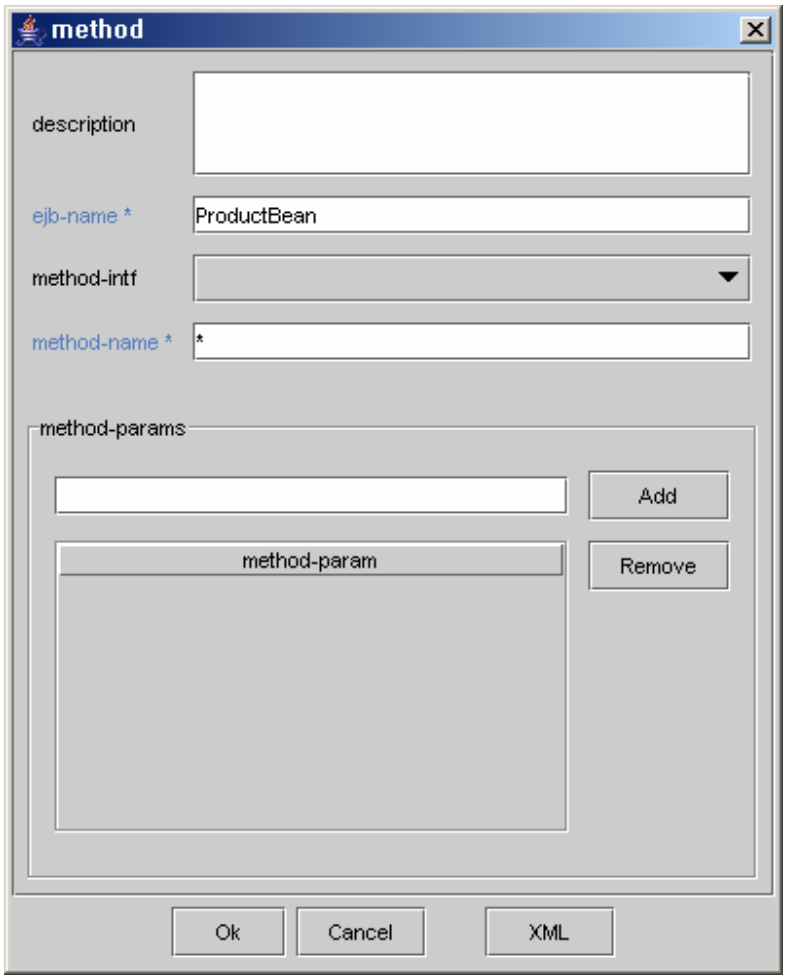

그림 45 Entity EJB 모듈의 method 별 트랜잭션 속성 설정

10. 'Ok'를 눌러 모든 창을 닫고, jeus-ejb-dd.xml 파일을 설정하기 위해 jeus-ejb-dd 탭을 클릭한다. 그리고, 하위에 있는 beanlist 탭을 클릭한다.

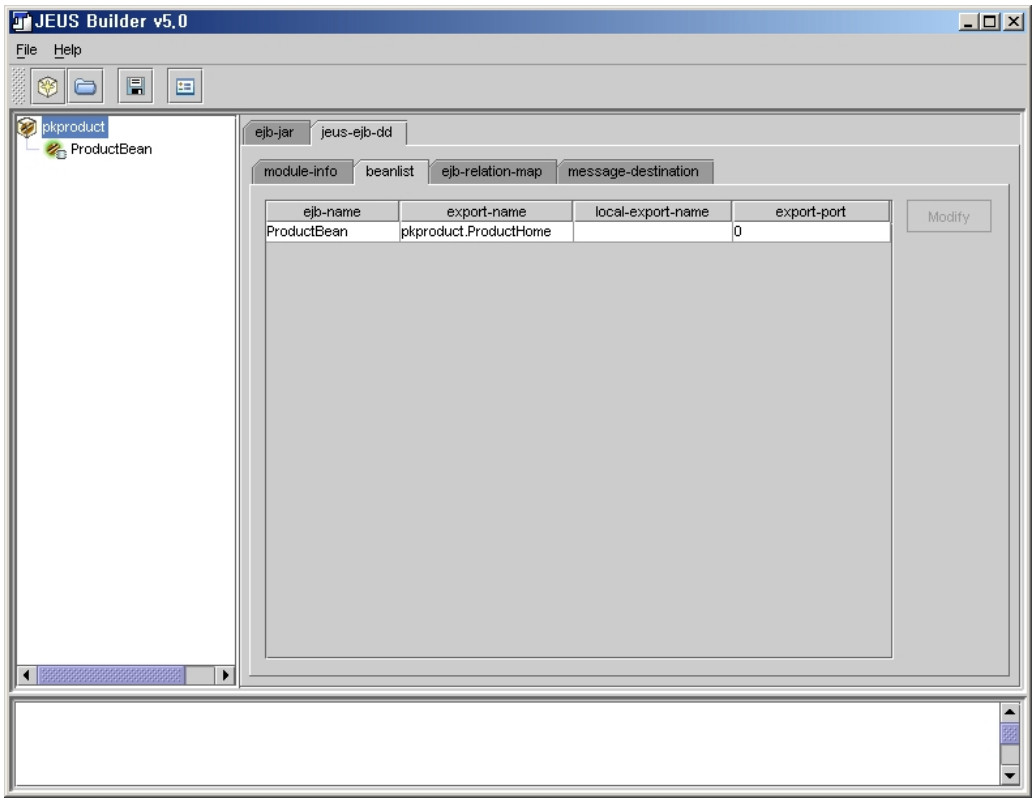

그림 46 Entity EJB 모듈 jeus-ejb-dd 탭 화면

11. beanlist 탭에서 ‘ProductBean’을 선택하고 Modify 버튼을 클릭한다.

12. export-name 에 JNDI 서비스에 등록할 이름으로 ‘ProductBean’이라고 입력한다.

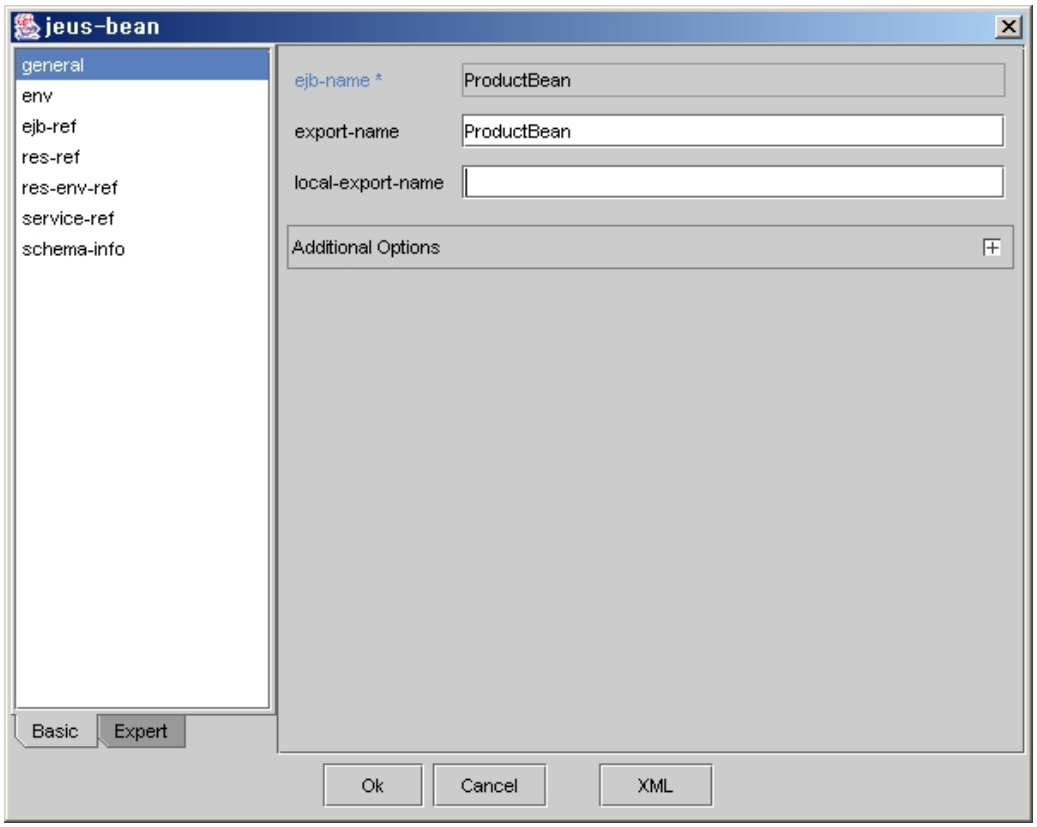

그림 47 Entity EJB 모듈 export-name

13. 좌측 영역에서 schema-info 를 선택한다. 여기서 기본적으로 입력할 부분은 general, creating-table 탭이다.

14. general 탭을 선택하고 table-name 에 product 를 입력한다. deleting-table 을 check 하고, db-vendor 에서는 hsql 을 선택한다. 그리고 data-source-name 으로는 datasource1 을 입력한다.

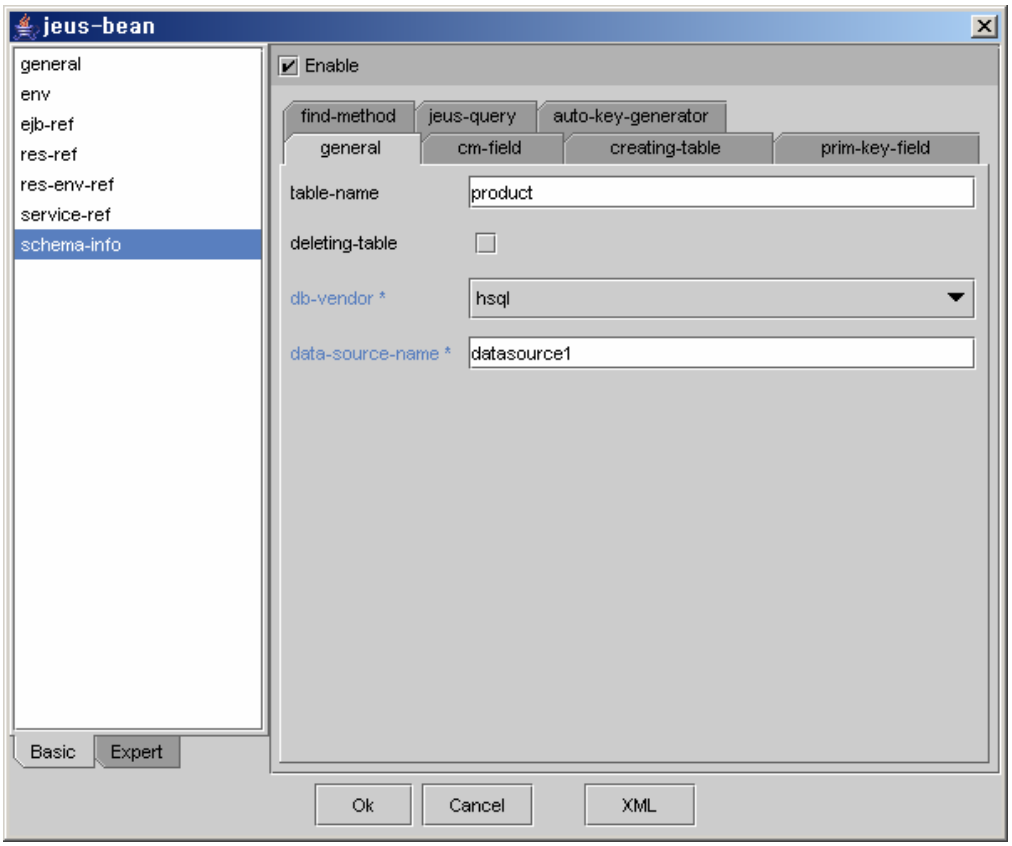

그림 48 Entity EJB 모듈 datasource

15. creating-table 탭을 클릭하고 use-existing-table 을 체크한다. 이값은 Deploy 할 때, 테이블이 없을 경우 생성하고, 존재할 경우 생성하지 않는다.

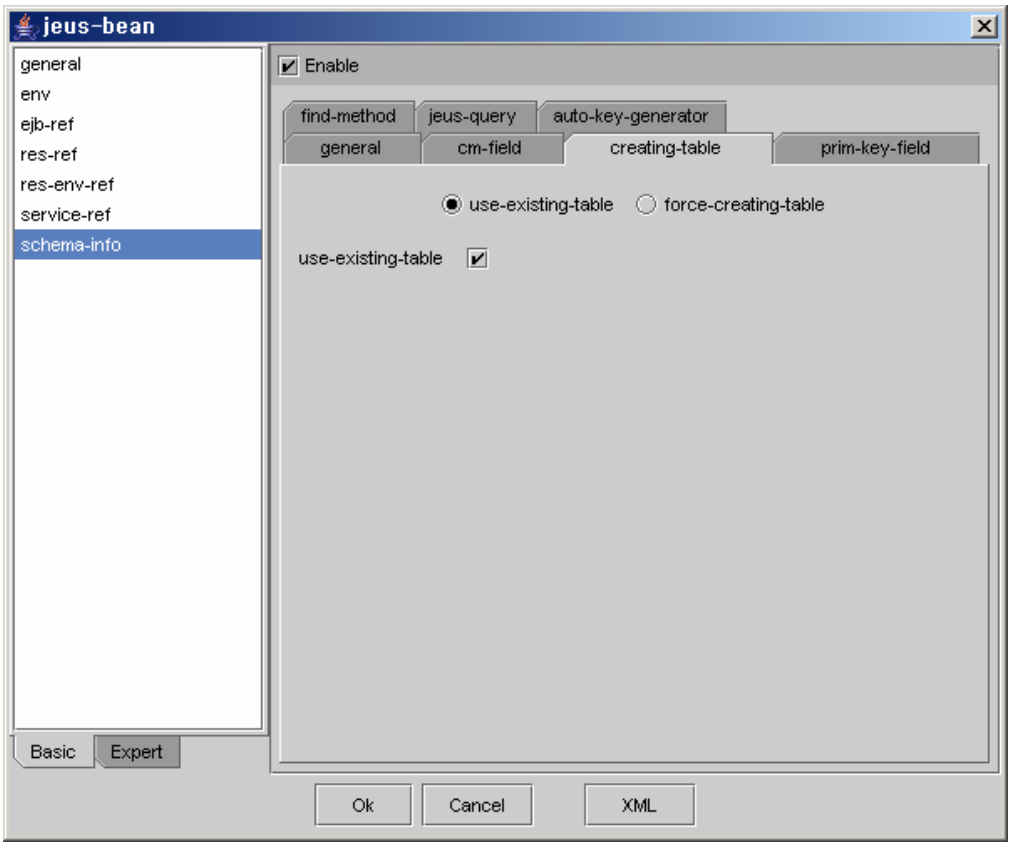

그림 49 Entity EJB 모듈 creating-table

16. jeus-bean 설정 윈도우의 Ok 버튼을 누르면 설정은 끝난다. File 메뉴의 Save 버튼이나 아이콘 메뉴에서 Save Module 아이콘을 클릭하면 지정된 위치에 모듈이 생성된다.

3.3.4 웹 관리자로 EJB 모듈 Deploy 하기

패키징된 모듈을 웹 관리자로 Deploy 해보도록 한다.

1. 웹 매니저로 접속한다.
2. 엔진 컨테이너 아래 'J2EE 어플리케이션 모듈'을 선택한다.
3. Deploy 할 모듈을 선택하고 다음 버튼을 클릭한다.

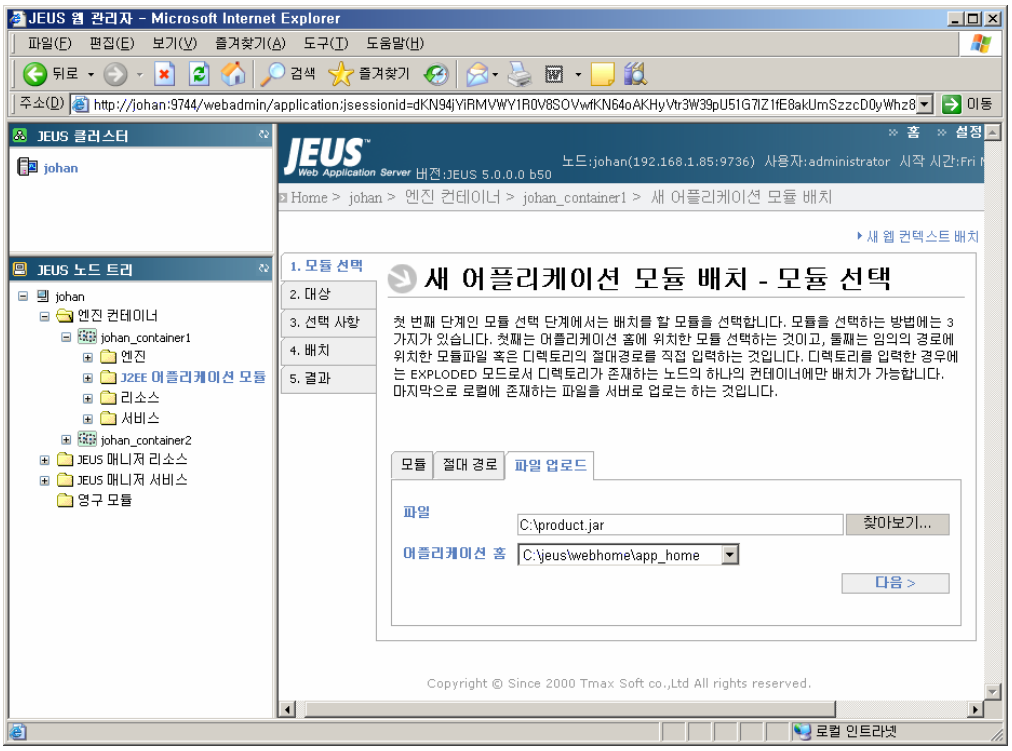

그림 50 Entity EJB 모듈 deploy 초기화면

4. Deploy 할 컨테이너를 선택하고 다음 버튼을 클릭한다.

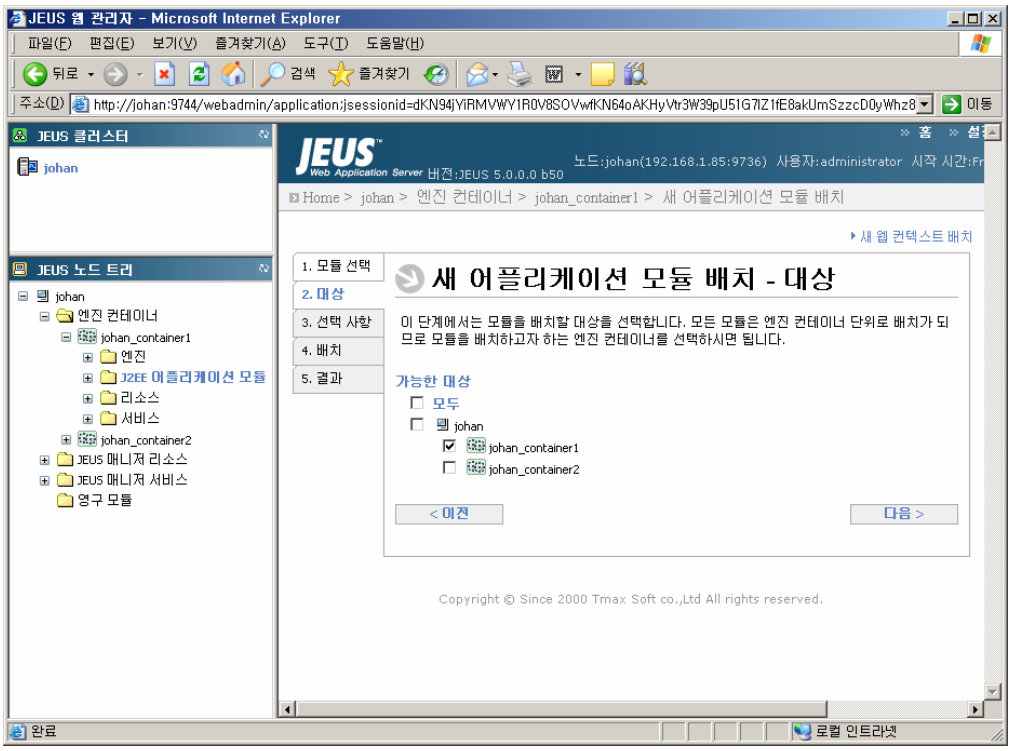

그림 51 Entity EJB 모듈 deploy 대상 선택

5. Deploy 옵션을 선택하고 다음 버튼을 클릭한다.

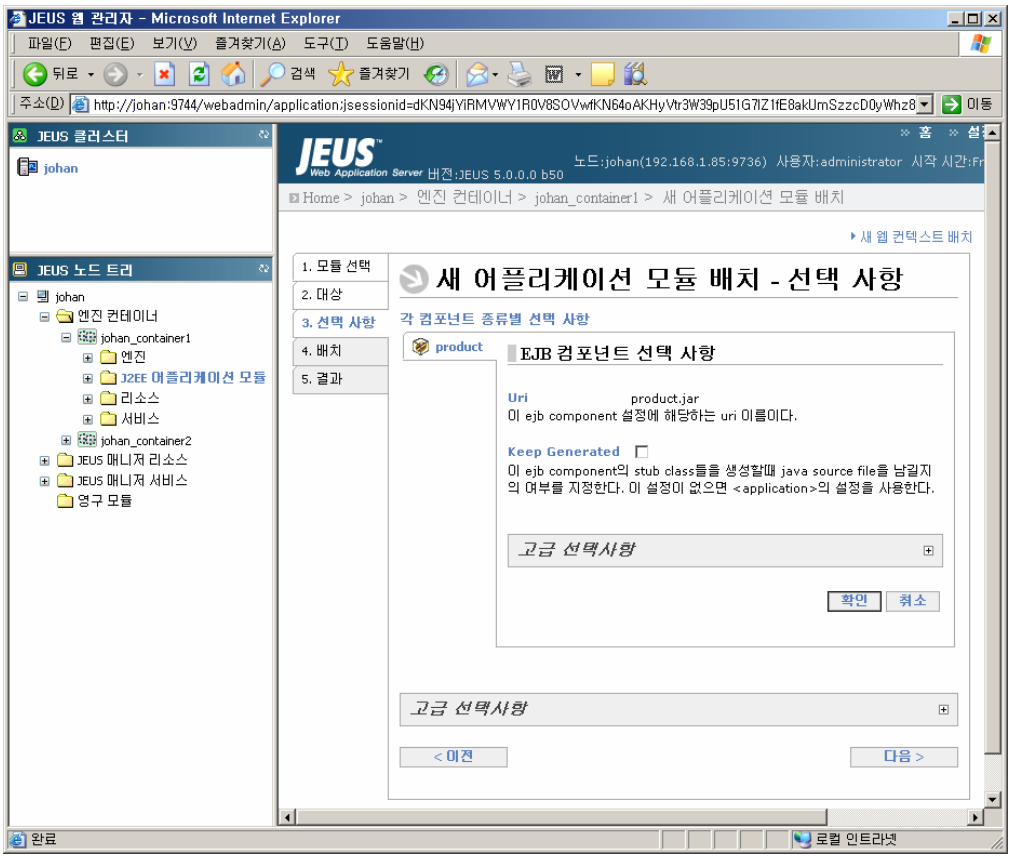

그림 52Entity EJB 모듈 deploy 선택사항

6. 배치 옵션을 선택하고 배치 버튼을 클릭하면 Deploy 를 시도한다

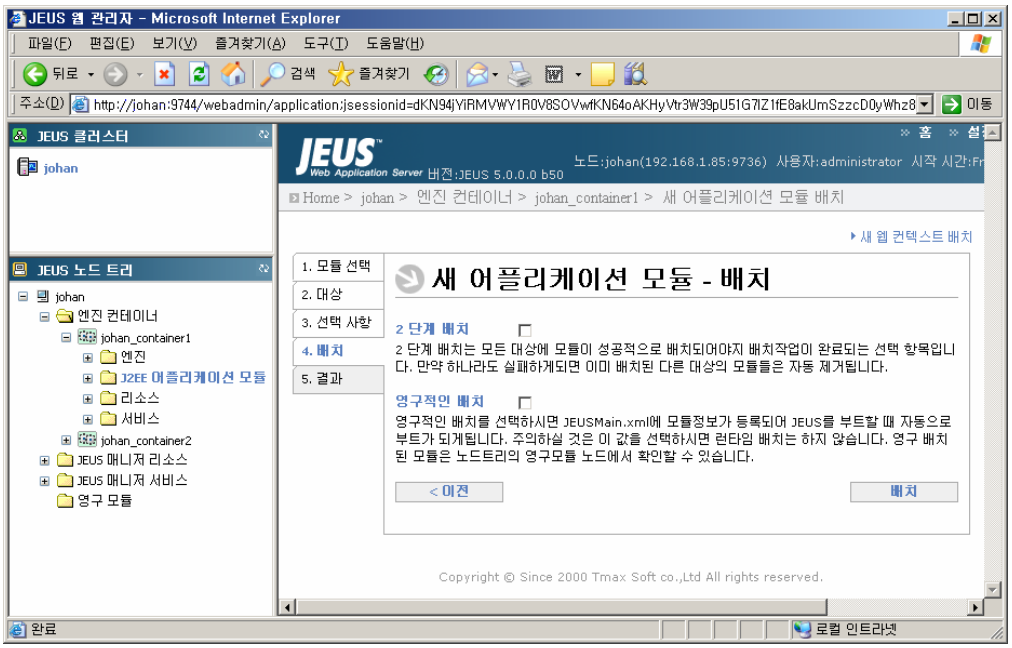

그림 53 Entity EJB 모듈 deploy 배치옵션

7. Deploy 가 성공할 경우 ‘J2EE 어플리케이션 모듈’ 아래 product 라고 나타난다.

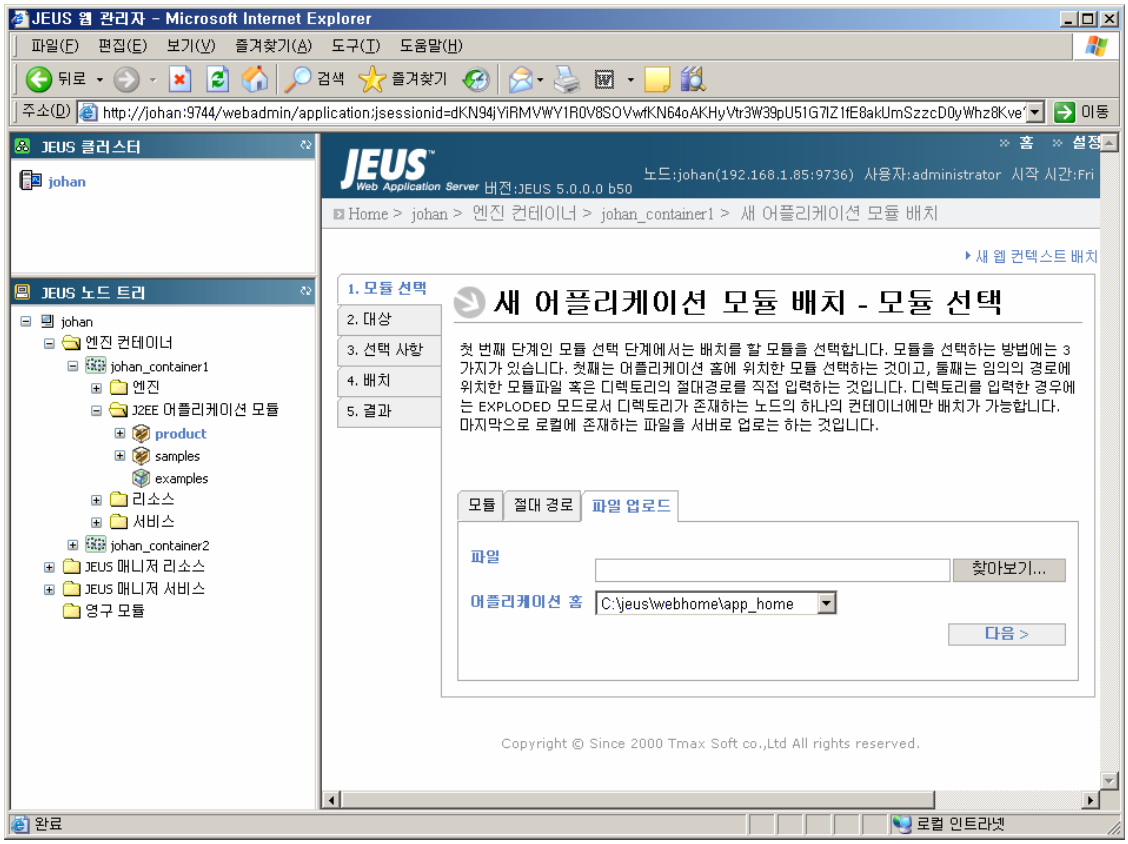

그림 54 deploy 된 Entity EJB

참고: 2.2.3 절에서 데이터소스 설정시 설명했지만, 데이터소스 설정에서 패스워드가 틀렸거나 IP 주소가 정확하지 않을 경우에 Deploy 는 실패하게 된다.

3.3.5 EJB 모듈 수동으로 Deploy 하기

엔티티 빈을 수동으로 Deploy 하는 방법은 세션 빈과 유사하다. 다음과 같은 순서로 Deploy 해보도록 한다.

1. ejb-jar.xml 파일과 jeus-ejb-dd.xml 파일을 생성하기 위해 ddinit 스크립트를 사용한다. 패키지명이 pkproduct 이고, 소스파일이 c:\temp\pkproduct 아래 있는 경우 c:\temp 까지만 입력한다.

   예) C:\jeus\bin\ddinit c:\temp

2. 기본적으로 생성되는 파일은 다음과 같다.

<<ejb-jar.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <enterprise-beans>
    <entity>
      <ejb-name>pkproduct.ProductEJB</ejb-name>
      <home>pkproduct.ProductHome</home>
      <remote>pkproduct.Product</remote>
      <ejb-class>pkproduct.ProductEJB</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>pkproduct.Pid</prim-key-class>
      <reentrant>false</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>Product</abstract-schema-name>
      <cmp-field>
        <field-name>description</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>price</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>productID</field-name>
      </cmp-field>
      <primkey-field>productID</primkey-field>
      <query>
        <query-method>
          <method-name>findByDescription</method-name>
          <method-params>
            <method-param>java.lang.String</method-param>
          </method-params>
        </query-method>
        <ejb-ql>SELECT OBJECT(o) FROM Product AS o</ejb-ql>
      </query>
      <query>
        <query-method>
          <method-name>findInRange</method-name>
          <method-params>
            <method-param>double</method-param>
            <method-param>double</method-param>
          </method-params>
        </query-method>
```

```
        <ejb-ql>SELECT OBJECT(o) FROM Product AS o</ejb-ql>
      </query>
    </entity>
  </enterprise-beans>
  <assembly-descriptor/>
</ejb-jar>
```

<<jeus-ejb-dd.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info/>
  <beanlist>
    <jeus-bean>
      <ejb-name>pkproduct.ProductEJB</ejb-name>
      <export-name>pkproduct.ProductHome</export-name>
      <export-port>0</export-port>
      <single-vm-only>false</single-vm-only>
      <local-invoke-optimize>true</local-invoke-optimize>
      <thread-max>100</thread-max>
      <object-management>
        <bean-pool>
          <pool-min>0</pool-min>
          <pool-max>100</pool-max>
        </bean-pool>
        <connect-pool>
          <pool-min>0</pool-min>
          <pool-max>100</pool-max>
        </connect-pool>
        <capacity>10000</capacity>
        <passivation-timeout>-1</passivation-timeout>
        <disconnect-timeout>-1</disconnect-timeout>
      </object-management>
      <persistence-optimize>
        <engine-type>EXCLUSIVE_ACCESS</engine-type>
      </persistence-optimize>
      <schema-info>
        <table-name>tableName</table-name>
        <cm-field>
          <field>description</field>
```

```
          <column-name>description</column-name>
        </cm-field>
        <cm-field>
          <field>price</field>
          <column-name>price</column-name>
        </cm-field>
        <cm-field>
          <field>productID</field>
          <column-name>productID</column-name>
        </cm-field>
        <db-vendor>oracle</db-vendor>
        <data-source-name>DataSource</data-source-name>
      </schema-info>
      <cm-persistence-optimize>
        <subengine-type>ReadLocking</subengine-type>
        <fetch-size>10</fetch-size>
        <init-caching>false</init-caching>
      </cm-persistence-optimize>
      <enable-instant-ql>false</enable-instant-ql>
    </jeus-bean>
  </beanlist>
</jeus-ejb-dd>
```

3. ejb-jar.xml 파일과 jeus-ejb-dd.xml 파일을 각각 다음과 같이 수정한다. 굵게 표시된 부분이 추가, 변경한 내용이다.

<<ejb-jar.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <enterprise-beans>
    <entity>
      <ejb-name>pkproduct.ProductEJB</ejb-name>
      <home>pkproduct.ProductHome</home>
      <remote>pkproduct.Product</remote>
      <ejb-class>pkproduct.ProductEJB</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>pkproduct.Pid</prim-key-class>
      <reentrant>false</reentrant>
      <cmp-version>2.x</cmp-version>
```

```
      <abstract-schema-name>Product</abstract-schema-name>
      <cmp-field>
        <field-name>description</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>price</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>productID</field-name>
      </cmp-field>
      <primkey-field>productID</primkey-field>
      <query>
        <query-method>
          <method-name>findByDescription</method-name>
          <method-params>
            <method-param>java.lang.String</method-param>
          </method-params>
        </query-method>
        <ejb-ql>SELECT OBJECT(o) FROM Product AS o WHERE
o.description = ?1</ejb-ql>
      </query>
      <query>
        <query-method>
          <method-name>findInRange</method-name>
          <method-params>
            <method-param>double</method-param>
            <method-param>double</method-param>
          </method-params>
        </query-method>
        <ejb-ql>SELECT OBJECT(o) FROM Product AS o WHERE o.price
between ?1 and ?2</ejb-ql>
      </query>
    </entity>
  </enterprise-beans>
  <assembly-descriptor/>
</ejb-jar>
```

<<jeus-ejb-dd.xml>>

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

```
<module-info/>
<beanlist>
  <jeus-bean>
    <ejb-name>pkproduct.ProductEJB</ejb-name>
    <export-name> ProductBean</export-name>
    <export-port>0</export-port>
    <single-vm-only>false</single-vm-only>
    <local-invoke-optimize>true</local-invoke-optimize>
    <thread-max>100</thread-max>
    <object-management>
      <bean-pool>
        <pool-min>0</pool-min>
        <pool-max>100</pool-max>
      </bean-pool>
      <connect-pool>
        <pool-min>0</pool-min>
        <pool-max>100</pool-max>
      </connect-pool>
      <capacity>10000</capacity>
      <passivation-timeout>-1</passivation-timeout>
      <disconnect-timeout>-1</disconnect-timeout>
    </object-management>
    <persistence-optimize>
      <engine-type>EXCLUSIVE_ACCESS</engine-type>
    </persistence-optimize>
    <schema-info>
      <table-name>product</table-name>
      <cm-field>
        <field>description</field>
        <column-name>description</column-name>
      </cm-field>
      <cm-field>
        <field>price</field>
        <column-name>price</column-name>
      </cm-field>
      <cm-field>
        <field>productID</field>
        <column-name>productID</column-name>
      </cm-field>
```

```
        <creating-table>
          <use-existing-table/>
        </creating-table>
        <db-vendor>oracle</db-vendor>
        <data-source-name>datasource1</data-source-name>
      </schema-info>
      <cm-persistence-optimize>
        <subengine-type>ReadLocking</subengine-type>
        <fetch-size>10</fetch-size>
        <init-caching>false</init-caching>
      </cm-persistence-optimize>
      <enable-instant-ql>false</enable-instant-ql>
    </jeus-bean>
  </beanlist>
</jeus-ejb-dd>
```

4. 컴파일된 EJB 파일과 xml 파일을 각각 다음과 같은 디렉토리 아래 위치시킨다.

   EJB 파일 : %JEUS_HOME%\webhome\app_home\<모듈명>\

   XML 파일: %JEUS_HOME%\webhome\app_home\<모듈명>\META-INF

   예)

   EJB 파일 : C:\jeus\webhome\app_home\product\

   XML 파일: C:\jeus\webhome\app_home\product\META-INF

5. 커맨드 창에서 'jeus' 스크립트를 실행하고, 'jeusadmin' 콘솔 툴에서 'boot' 명령을 통해 JEUS 를 기동시킨다.

6. 부팅이 정상적으로 되었다면, 커맨드 창에서 'ejbadmin'이라고 입력하면 ejbadmin 콘솔 툴이 실행된다. 아래와 같은 순서로 명령을 입력한다.

```
C:\>ejbadmin johan_container1
Login name>administrator
Password>
JEUS 5.0 EJB Engine Controller
johan_container1>deploy product
using the following application info : applicationType
```

```
        path:product
        deployment-type:COMPONENT
        class-ftp-unit:JAR
        ejb-component
   johan_container1>modulelist
   name : product
      type : EJBModule   EngineContainer : johan_container1  node
   : johan
   johan_container1>
   johan_container1>beanlist product
   pkproduct.ProductEJB
```

모듈이 정상적으로 Deploy 되었는지 확인한다.

다음 절에서는 예제코드를 작성해 보도록 한다.

3.3.6 EJB Client 예제코드

이 절에서는 Entity EJB 를 사용하는 client 프로그램을 작성하고, 컴파일하여, 실행하는 과정을 다룬다.

EJB 의 클라이언트는 Java application, Servlet, JSP 혹은 다른 EJB 가 될 수도 있다. 여기서는 Java 어플리케이션을 사용하기로 한다.

아래의 예는 'product' entity bean 을 사용하는 ProductClient.java 라는 Java 어플리케이션이다.

<<ProductClient.java>>

```
package pkproduct;

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

public class ProductClient {
 public static void main(String[] args) {
  try {
   if(System.getSecurityManager() == null)
```

```
   System.setSecurityManager(new RMISecurityManager());

   Hashtable env = new Hashtable();
   env.put(Context.INITIAL_CONTEXT_FACTORY,
"jeus.jndi.JEUSContextFactory");
   InitialContext ctx = new InitialContext(env);

   ProductHome home = (ProductHome)ctx.lookup("ProductBean");

   Product duke = home.create("123", "Ceramic Dog", 10.00);
   System.out.println(duke.getDescription() + ": " +
duke.getPrice());
   duke.setPrice(14.00);
   System.out.println(duke.getDescription() + ": " +
duke.getPrice());

   Product duke1 = home.create("456", "Wooden Duck", 13.00);
   Product duke2 = home.create("999", "Ivory Cat", 19.00);
   Product duke3 = home.create("789", "Ivory Cat", 33.00);
   Product duke4 = home.create("876", "Chrome Fish", 22.00);

   Product earl = home.findByPrimaryKey(new Pid("876"));
   System.out.println(earl.getDescription() + ": " +
earl.getPrice());

   Collection c = home.findByDescription("Ivory Cat");
   Iterator i = c.iterator();

   while (i.hasNext()) {
    Product product = (Product)i.next();
    Pid pkey = (Pid)product.getPrimaryKey();
    String description = product.getDescription();
    double price = product.getPrice();
    System.out.println(pkey.productID + " : " + description + " "
+ price);
   }

   c = home.findInRange(10.00, 20.00);
   i = c.iterator();
```

```
   while (i.hasNext()) {
    Product product = (Product)i.next();
    Pid pkey = (Pid)product.getPrimaryKey();
    double price = product.getPrice();
    System.out.println(pkey.productID + " : " + price);
   }

   duke.remove();
   duke1.remove();
   duke2.remove();
   duke3.remove();
   duke4.remove();

   } catch (Exception ex) {
    ex.printStackTrace();
   }
  }
}
```

3.3.7 EJB Client 예제 코드 컴파일

EJB 모듈이 성공적으로 Deploy 될 경우 %JEUS_HOME%\webhome\<node name>_<container name> 아래에, 모듈명에 해당하는 디렉토리가 생성된다. 이 디렉토리 아래에는 EJB Remote, Home 클래스를 구현한 클래스와 각각의 스텁/스켈레톤 클래스들이 생성된다.

생성된 스텁/스켈레톤 클래스는 EJB Client 가 컴파일하고 실행하는 과정에서 필요하게 된다. 이 파일들을 client.jar 로 묶어서 클라이언트 어플리케이션을 컴파일하고 실행하는 과정에서 client 가 인식할 수 있도록 classpath 에 추가시켜준다.

예)

```
C:\jeus\webhome\johan_container1\product\product>jar cvf
c:\client.jar pkproduct
```

EJB client 프로그램을 컴파일 하기 위해서는 client java file 이 있는 디렉토리로 이동 후 아래처럼 실행하면 된다.

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\EJB
Tutorial\src\client>javac -classpath
C:\jeus\lib\system\jeus.jar;c:\client.jar -d ..\..\classes
ProductClient.java
```

예제에서 client 코드가 “pkproduct”라는 패키지 구조를 가지므로 실제 컴파일을 하면 “pkproduct”라는 디렉토리 아래에 ProductClient.class 파일이 생성된다.

3.3.8 EJB Client 실행하기

EJB Client 를 3.2.11 절처럼 ClientContainer 를 사용해서 실행하는 것 이외에, 직접 실행하는 방법이 있다. ClientContainer 를 사용하면 번거롭기 때문에, 실제로 이런 방법이 훨씬 많이 사용된다. 이번 절에서는 이 방법에 대해서 알아본다.

EJB Client 가 실행되기 위해서는 다음과 같은 정보가 필요하다.

1. EJB 관련 클래스

2. EJB 의 Remote 인터페이스와 Home 인터페이스

3. EJB 의 Stub/Skeleton 클래스

4. EJB Client 클래스

5. JNDI 관련 설정(`-Djava.naming.provider.url` 과 `-Djava.naming.factory.initial`)

6. JEUS 의 Base Port 설정

EJB 관련 클래스는 c:\jeus\lib\system\jeus.jar 파일에 모두 포함되어 있고, 인터페이스 클래스는 EJB 의 소스 파일을 컴파일한 디렉토리에 모두 있다. Stub/Skeleton 클래스는 앞 절에서 c:\client.jar 로 묶어 두었다. EJB Client 클래스도 앞 절에서 컴파일했다.

JNDI 관련 설정은 JVM 의 시스템 파라미터로 다음과 같이 넣어준다.

```
-Djava.naming.provider.url=johan
–Djava.naming.factory.initial=jeus.jndi.JEUSContextFactory
```

JEUS 의 Base Port 는 JVM 의 시스템 파라미터인 -Djeus.baseport 에 값을 넣어 주면 된다. 본 예제에서는 기본값인 9736 을 넣어준다.

EJB Client 를 실행하기 위해서는 위 4 가지의 클래스를 java 의 -classpath 옵션에 넣어서 EJB Client 를 실행시켜 주면 된다.

예)

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\EJB
Tutorial\classes>java -Djava.naming.provider.url=johan –Djava.nam
ing.factory.initial=jeus.jndi.JEUSContextFactory –Djeus.baseport=
9736 -classpath c:\jeus\lib\system\jeus.jar;c:\product.jar;c:\cli
ent.jar;. pkproduct.ProductClient
Ceramic Dog: 10.0
Ceramic Dog: 14.0
Chrome Fish: 22.0
789 : Ivory Cat 33.0
999 : Ivory Cat 19.0
123 : 14.0
456 : 13.0
999 : 19.0
```

3.3.9 결론

이 절에서는 CMP 2.0 entity bean 을 작성, 컴파일, 패키징하여 Deploy 한 다음 해당 CMP 2.0 entity bean 을 이용하는 client application 을 작성해보았다.

그리고 Client Container 를 사용하지 않고 EJB Client 를 실행하는 방법에 대해서 알아보았다.

4 Servlet/JSP 사용하기

4.1 소개

이번 장에서는 다음 내용에 대해 설명한다

- Servlet 과 JSP application 의 Deploy
- WAR module(Web application)의 패키징과 deploy 하는 방법에 대해서 설명한다.

4.2 Web Application Deploy

이 절에서는 web application 의 간단한 예제 코드를 작성하고, 해당 소스의 컴파일과 deploy 과정을 살펴보기로 한다.

예제는 `%JEUS_HOME%\samples\manual_examples\GettingStartedTutorial\Servlet JSP Tutorial` 에서 찾아볼 수 있다.

4.2.1 Servlet/JSP 예제코드

아래는 웹 브라우저에 간단히 "Hello World!"라는 메시지를 출력하는 예제 Servlet 이다

<<HelloWorldServlet.java>>

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloWorldServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse
    res) throws IOException, ServletException{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Hello World Sample</TITLE>");
        out.println("</HEAD>");
```

```
        out.println("<BODY>");
        out.println("<CENTER><H1>Hello World!</H1></CENTER>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

다음은 요청을 받았을 때 request 에 대한 몇 가지 정보를 보여주는 “snoop.jsp”라는 샘플 JSP 프로그램이다.

<<snoop.jsp>>

```
<html>

<body bgcolor="white">

<h2> Request Information </h2>
<font size="4">
JSP Request Method: <%= request.getMethod() %>
<br>
Request URI: <%= request.getRequestURI() %>
<br>
Request Protocol: <%= request.getProtocol() %>
<br>
Servlet path: <%= request.getServletPath() %>
<br>
Path info: <%= request.getPathInfo() %>
<br>
Path translated: <%= request.getPathTranslated() %>
<br>
Query string: <%= request.getQueryString() %>
<br>
Content length: <%= request.getContentLength() %>
<br>
Content type: <%= request.getContentType() %>
<br>
Server name: <%= request.getServerName() %>
<br>
```

```
Server port: <%= request.getServerPort() %>
<br>
Remote user: <%= request.getRemoteUser() %>
<br>
Remote address: <%= request.getRemoteAddr() %>
<br>
Remote host: <%= request.getRemoteHost() %>
<br>
Authorization scheme: <%= request.getAuthType() %>
<hr>
The browser you are using is <%= request.getHeader("User-
Agent") %>
<hr>
</font>
</body>
</html>
```

이 예제는 JSP 이므로 사용자가 컴파일 해 줄 필요없이, Servlet engine 이 컴파일을 자동으로 해준다. 다만 “snoop.jsp”를 넣을 jsp 디렉토리가 필요하다.

4.2.2 Servlet 예제코드 컴파일

작성한 Servlet 소스 파일을 컴파일하기 위해서는 Servlet 이 사용하는 class 들과 interface 가 classpath 에 잡혀있어야 한다. 또한 컴파일을 위해서는 기본적인 java 의 classpath 외에 JEUS 의 classpath 역시 classpath 에 추가되어야 한다. 만일 JEUS 가 “C:\jeus”라는 디렉토리에 설치되어 있다면 “C:\jeus\lib\system\jeus.jar” 역시 클래스 패스에 추가되어야 한다.

예)

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\Servlet
JSP Tutorial>javac -classpath C:\jeus\lib\system\jeus.jar -d .
*.java
```

컴파일 후 HelloWorldServlet.class 란 파일이 컴파일을 수행한 디렉토리에 생성된다.

‘helloworld’라는 디렉토리 아래에 ‘WEB-INF’라는 디렉토리를 만든 뒤, WEB-INF 디렉토리 아래에 다시 ‘classes’라는 디렉토리를 만들고 패키지 구조에 따라 HelloWorldServlet.class 를 복사한다(예제에서는 anonymous 패키

지이므로 별도의 디렉토리 없이 WEB-INF\classes 로 복사하기만 하면 된다). 그리고 snoop.jsp 를 'helloworld' 디렉토리에 복사한다.

예)

```
C:\jeus\samples\manual_examples\GettingStartedTutorial\Servlet
JSP Tutorial>copy HelloWorldServlet.class helloworld\WEB-
INF\classes
```

4.2.3 JEUSBuilder 로 WAR 모듈 패키징 하기

지금까지의 소스파일을 작성하고 컴파일에 성공하였다면 다음의 절차에 따라 J2EE WAR 파일을 패키징할 수 있다.

1. 커맨드 창에서 'jeusbuilder' 명령으로 JEUS Builder 를 실행시키고, File 메뉴에서 New 버튼을 클릭하거나, 아이콘 메뉴에서 New Module 을 선택한다.

2. 모듈 타입을 war 로 설정하고, 모듈이 생성될 위치를 입력한다. 컴파일한 클래스파일을 추가하고 'Ok' 버튼을 클릭한다.

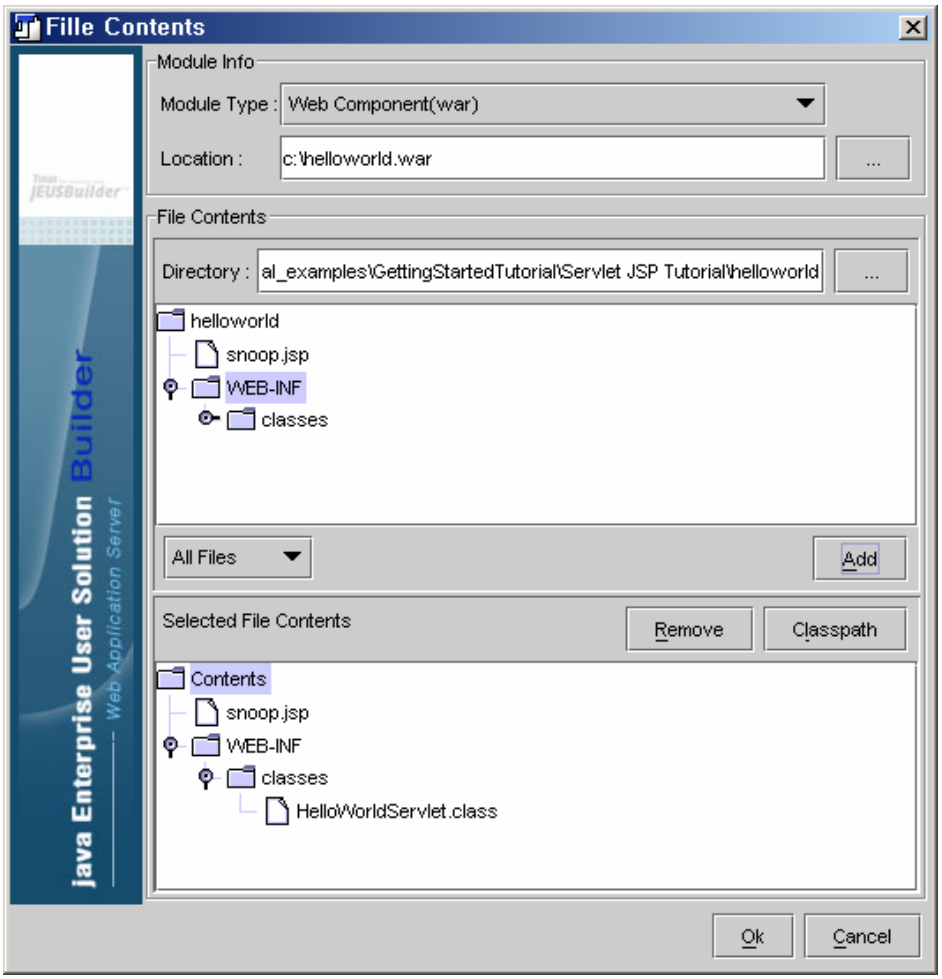

그림 55 WAR 모듈 파일추가

3. Servlet/JSP 파일만 포함시킬 경우 web.xml, jeus-web-dd.xml 파일이 생성된다. 먼저 web.xml 파일을 설정해 보도록 한다. general 에서 display-name 을 입력한다.

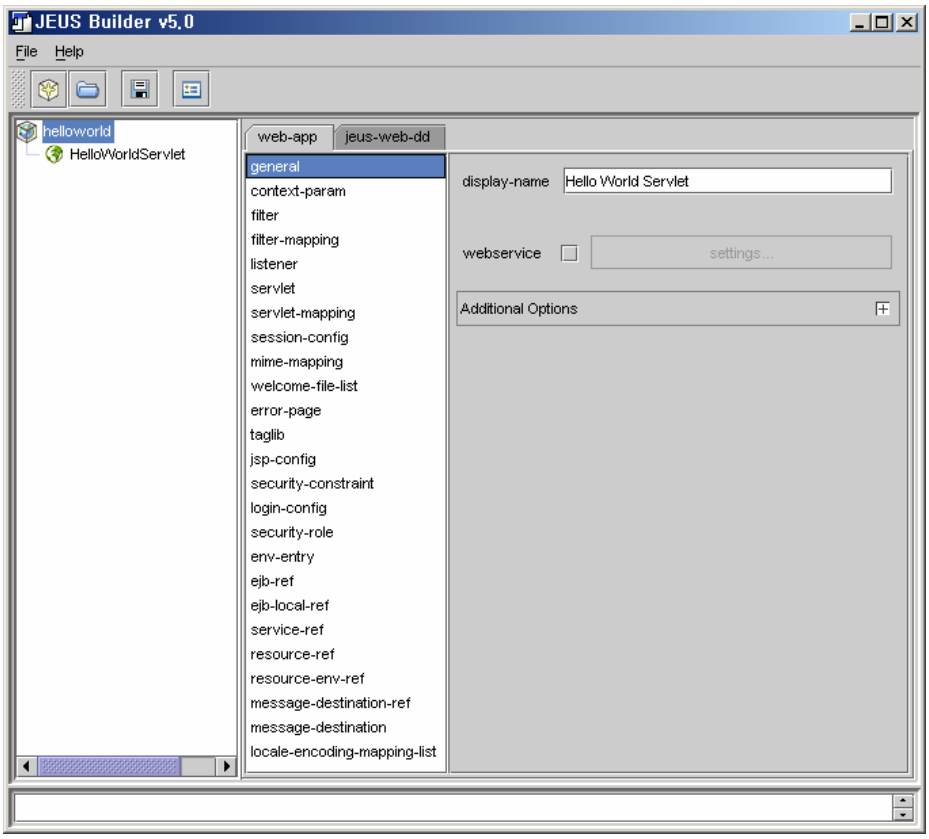

그림 56 WAR 모듈 display-name

4. servlet 을 선택하면 자동으로 생성된 서블릿 정보가 나타난다. Modify 버튼을 누르고 servlet-name 을 'HelloWorld'로 변경한다.

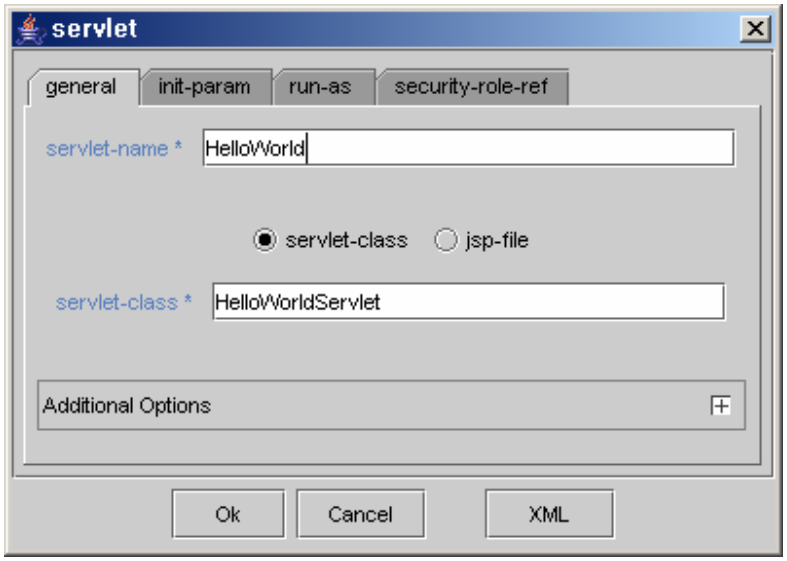

그림 57 WAR 모듈 servlet-name

5. WEB-INF\classes 에 있는 서블릿은 모듈생성시 등록되며, jsp 파일은 추가로 등록해야 한다. Add 버튼을 눌러서 snoop.jsp 를 등록한다.

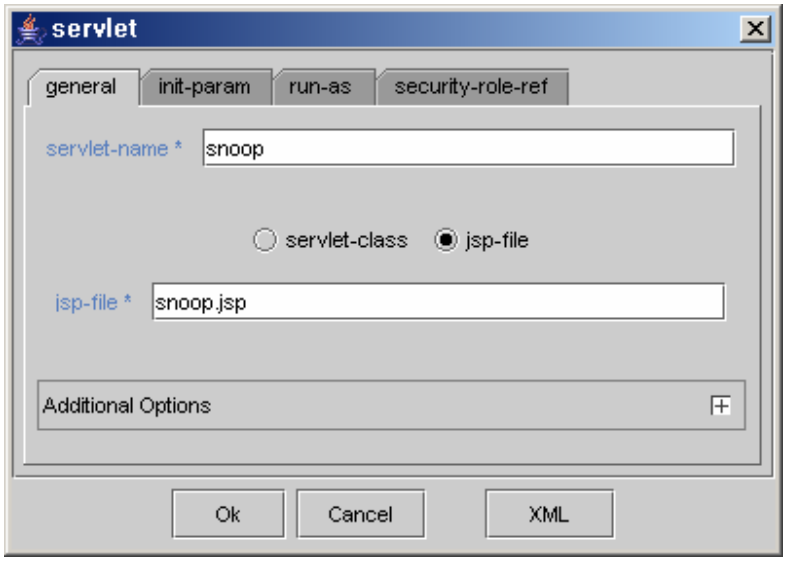

그림 58 JSP 추가

6. 생성된 jeus-web-dd.xml 파일은 기본값이 채워지므로, 필요한 경우에만 값을 입력하도록 한다. docbase 는 Servlet 과 JSP application 그리고 이미지나 자바 클래스와 같은 자원들을 저장하기 위한 기본디렉토리이다. 이 디렉토리는 Context Group 의 디렉토리와도 연관 있다. 그리고 Servlet engine 이 JSP page 를 컴파일 할 수 있도록 enable-jsp 항목 역시 선택해 준다.

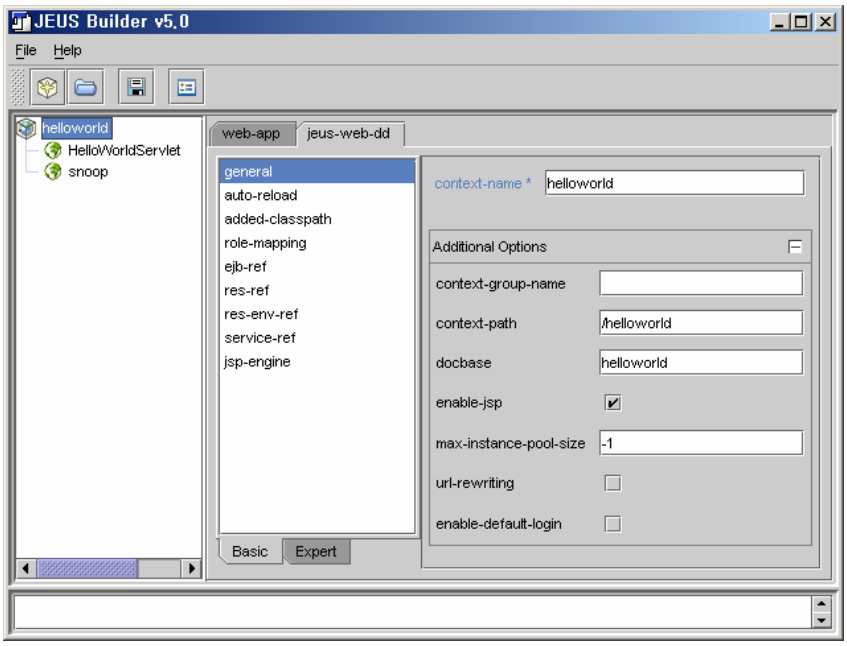

그림 59 WAR 모듈 jeus-web-dd

7. 저장버튼을 클릭하면 지정한 위치에 web application 모듈이 생성된다.

4.2.4 웹 관리자로 WAR 모듈 Deploy 하기

패키징한 WAR 모듈은 웹 관리자와 콘솔 툴로 Deploy 할 수 있다. 웹관리자에서는 EJB 모듈과 같이 WAR 모듈에 대해 archive 모드, exploded 모드를 제공한다. 또한, JEUS 가 기동되는 시스템의 임의의 디렉토리에 web application 이 존재할 경우 exploded 모드로 Deploy 할 수 있는 기능을 제공한다.

이 기능은 Node View 의 엔진 컨테이너 하위에 'J2EE 어플리케이션 모듈'을 선택했을 때, '새 웹 컨텍스트 배치'라는 링크를 클릭함으로써 사용할 수 있다.

자세한 내용은 웹 관리자 안내서를 참조하기 바란다. 여기서는 이전에 사용한 방법과 동일하게 WAR 모듈을 Deploy 해보도록 한다.

1. 웹 브라우저로 접속해서 로그인 한다.

2. Node View 에서 Deploy 하고자 하는 EJB 엔진이 엔진 컨테이너의 'J2EE 어플리케이션 모듈'을 선택한다.

3. Deploy 할 모듈을 선택하고 다음 버튼을 클릭한다.

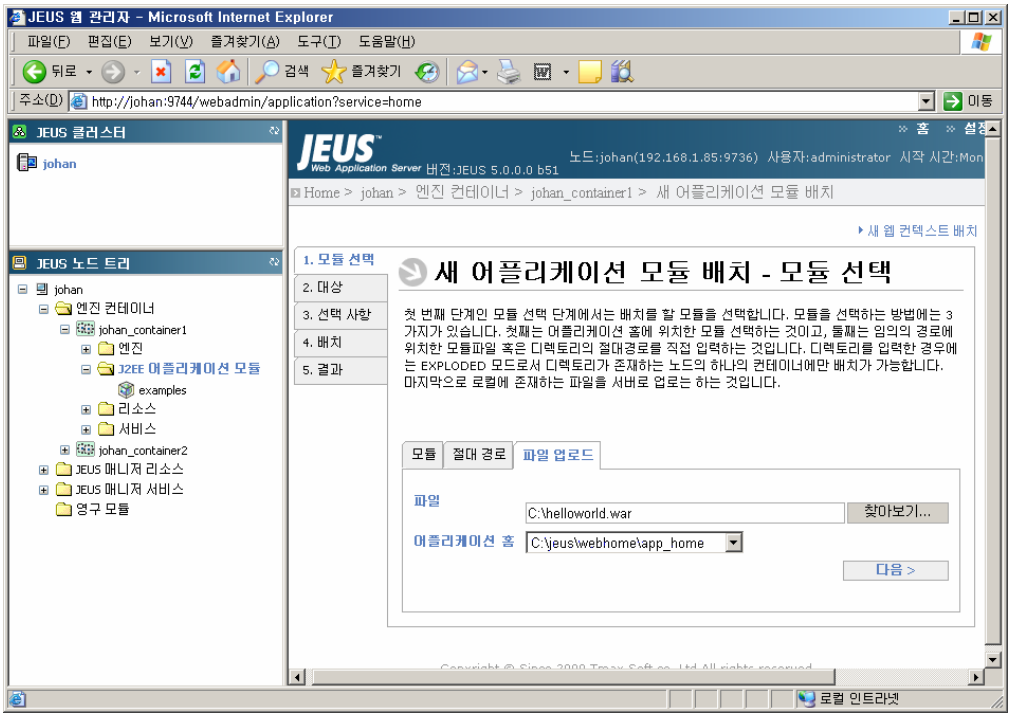

그림 60 WAR 모듈 deploy 초기화면

4. Deploy 할 엔진 컨테이너를 선택하고, 서블릿 엔진의 컨텍스트 그룹을 입력한다.

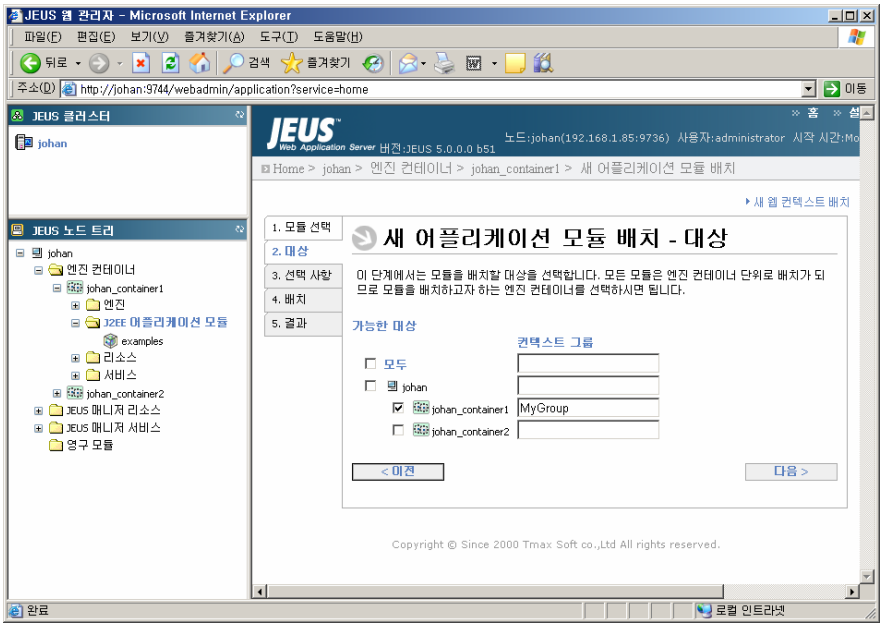

그림 61 WEB 모듈 deploy 대상선택

5. 필요한 선택사항을 입력하고 다음 버튼을 클릭한다.

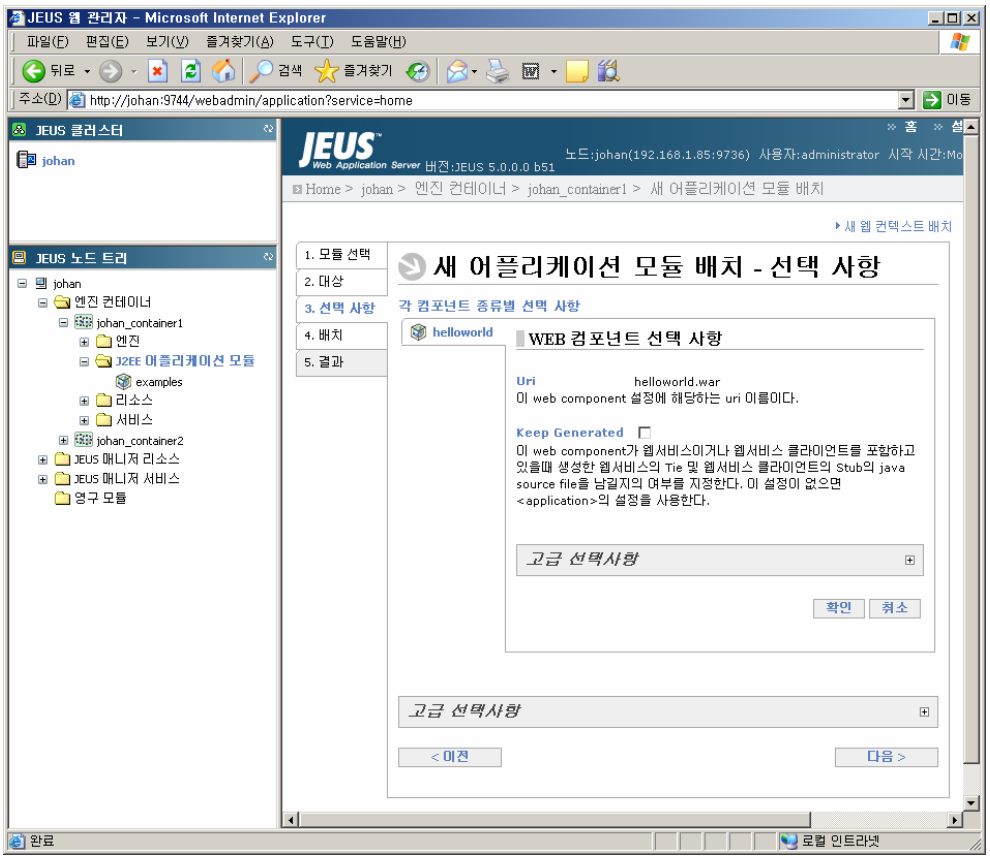

그림 62 WEB 모듈 deploy 선택사항

6. 배치옵션이 필요한 경우 입력하고 다음 버튼을 클릭한다.

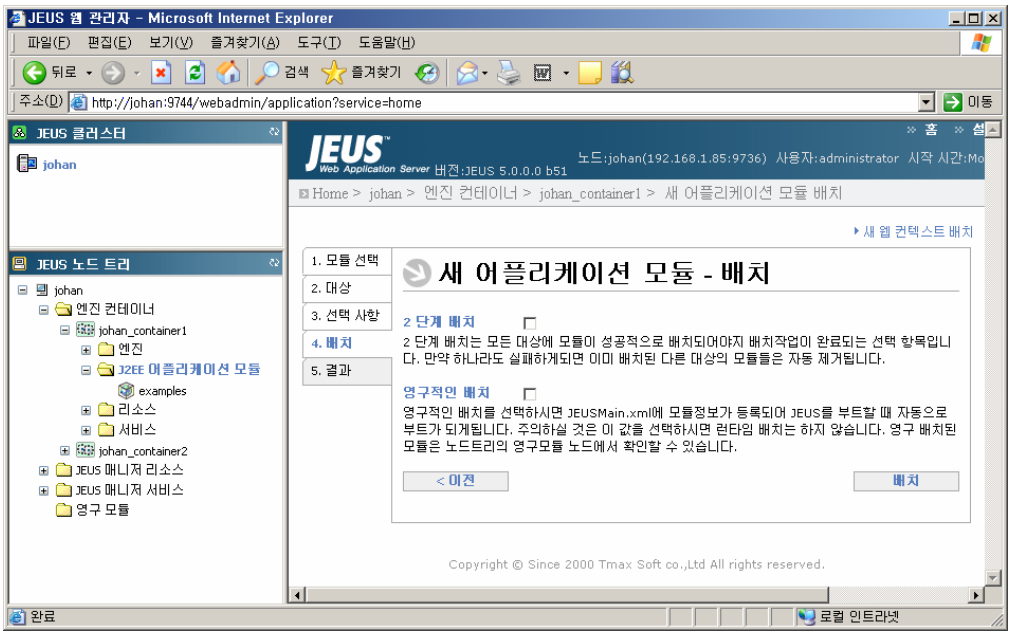

그림 63 WEB 모듈 deploy 옵션

7. Deploy 가 성공적으로 완료되면 Node View 의 'J2EE 어플리케이션 모듈' 아래 helloworld 라고 나타난다.

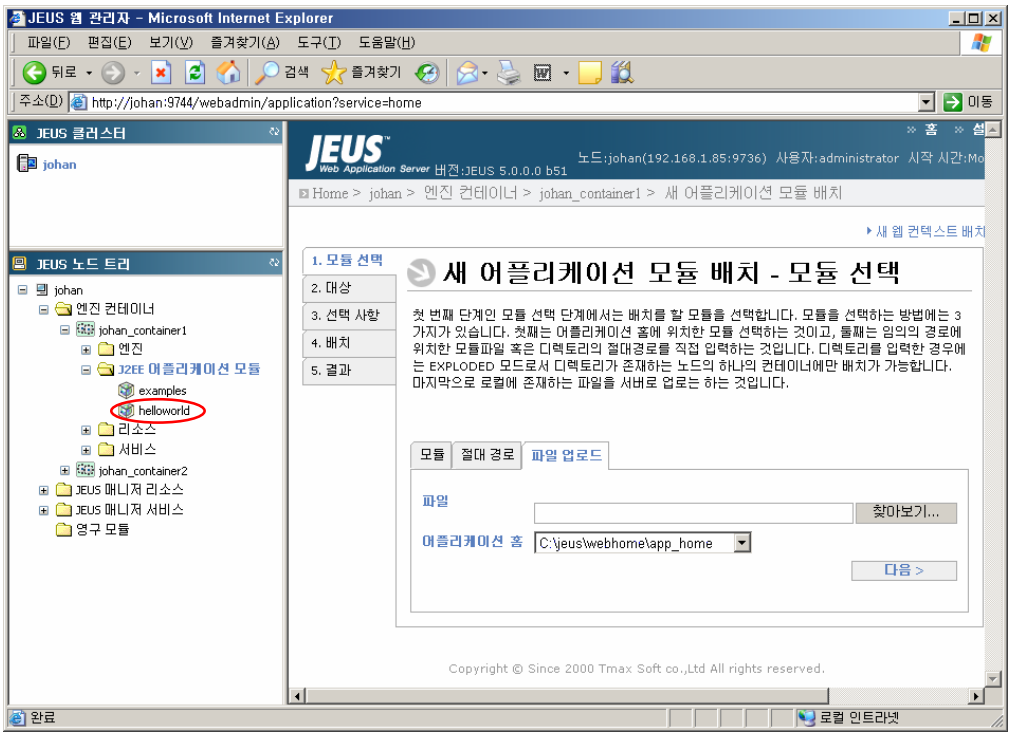

그림 64 WEB 모듈 deploy

4.2.5 WAR 모듈 수동으로 Deploy 하기

이 절에서는 콘솔 툴을 사용하여 수동으로 WEB 모듈을 Deploy 하는 법을 다룬다. 단 Servlet 과 JSP 는 이미 언급한 데로 모두 컴파일된 것으로 가정한다.

1. 텍스트 에디터를 사용하여 아래와 같은 내용으로 web.xml 과 jeus-web-dd.xml 파일을 작성한다.

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

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <context-path>/helloworld</context-path>
</jeus-web-dd>
```

<<*web.xml*>>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>HelloWorldServlet</display-name>
    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>HelloWorldServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>snoop</servlet-name>
        <jsp-file>snoop.jsp</jsp-file>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/HelloWorldServlet/*</url-pattern>
    </servlet-mapping>
</web-app>
```

2. 컴파일된 Servlet, JSP 파일과 xml 파일을 각각 다음과 같은 디렉토리 아래 위치시킨다.

Servlet 파일 :%JEUS_HOME%\webhome\app_home\<모듈명>\WEB-INF\classes

JSP 파일 :%JEUS_HOME%\webhome\app_home\<모듈명>

XML 파일 :%JEUS_HOME%\webhome\app_home\<모듈명>\WEB-INF

예)

Servlet 파일 :C:\jeus\webhome\app_home\helloworld\WEB-INF\classes

JSP 파일 :C:\jeus\webhome\app_home\helloworld

XML 파일 :C:\jeus\webhome\app_home\helloworld\WEB-INF

3. jeusadmin 으로 JEUS 에 접속한다.

예) jeusadmin johan –Uadministrator –Pjeusadmin

4. 접속이 되었다면 아래와 같은 순서로 명령을 입력한다.

johan>**deploy -con johan_container1 helloworld**

using the following application info : applicationType

path:helloworld.war

deployment-type:COMPONENT

class-ftp-unit:JAR

web-component

모듈이 정상적으로 Deploy 되었는지 확인하고, 다음절에서 Deploy 한 WEB 모듈을 실행해 보도록 한다.

4.2.6 WAR 모듈 실행 및 결과

이 절에서는 Deploy 된 Servlet 과 JSP 를 사용하는 법을 살펴본다. 우선 Servlet 부터 사용해보자.

HelloWorld Servlet 을 호출하기 위해서는 브라우저의 주소창에 다음과 같이 입력한다.

```
http://localhost:8088/helloworld/HelloWorld
```

만일 Servlet 엔진이 정상적으로 기동된 상태이고, HelloWorldServlet 이 정상적으로 Deploy 되었다면 다음과 같은 화면이 나타난다.

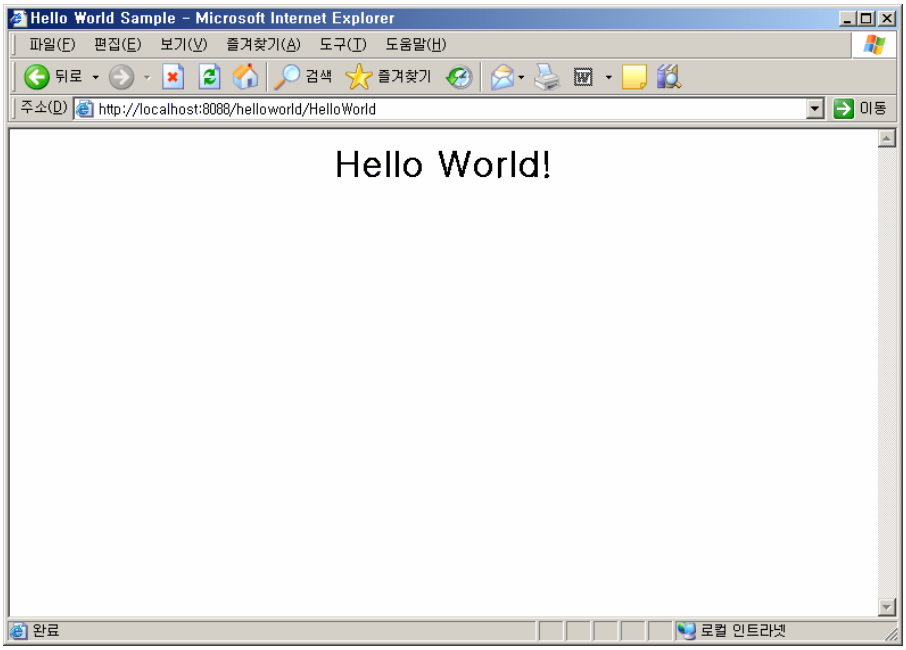

그림 65 WAR 모듈 Servlet 호출

주소 줄에 입력한 URL(`http://localhost:8088/helloworld/HelloWorld`)의 의미는 다음과 같다[표 1].

표 1. Servlet 요청 패스의 항목과 의미

항 목	의 미
http	JEUS 에 접속할 때 사용하는 HTTP Protocol 을 의미
localhost	서비스를 제공하는 서버가 브라우저와 동일한 자신의 주소에 있음을 의미
8088	Context Group 의 HTTP listener 의 포트번호

항 목	의 미
helloworld	Web application 의 context 용 request path. Context group 의 request path 가 아님에 주의해야 한다. Context Group 은 오직 port number 의 구분을 통해서만 요청을 분리할 수 있다. 이 request path 는 WEBMain.xml 의 context-path element 에 설정되며, 이 값을 지정하지 않을 경우 WAR 모듈 파일의 이름과 동일하다.
HelloWorld	Servlet 에 정의된 이름

Snoop.jsp 페이지를 호출하려면 http://localhost:8088/helloworld/snoop 을 주소창을 통해 호출하면 된다(JSP 의 경우 최초 호출 시 Servlet engine 이 컴파일을 자동으로 수행하므로 약간 늦게 실행된다).

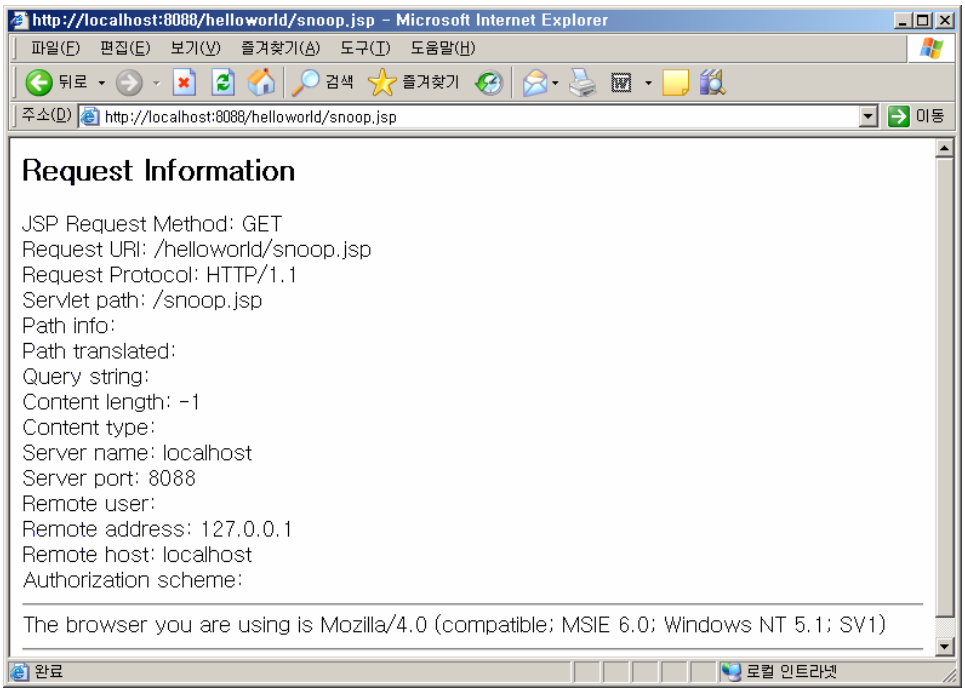

그림 66 WAR 모듈 JSP 호출

JSP 를 호출하는 URL 역시 Servlet 과 동일한 의미로 해석할 수 있다. 특히 JSP 를 호출하기 위해 사용된 “snoop” 역시 Servlet 과 동일한 의미이다.

4.2.7 결론

이상으로 Web application 을 작성하고, 컴파일, 패키징, deploy 한 후 호출하는 법을 살펴 보았다.

4.3 결론

지금까지 Servlet/JSP 사용하기에서는 다음과 같은 내용을 설명했다.

- Servlet 과 JSP 로 이루어진 Web application 을 패키징 하는 방법

이 절에서 다룬 내용에 대해 더 자세한 정보를 알고 싶으면, JEUS Server 안내서, JEUS Web Container 안내서, JEUS WebServer 안내서를 참조한다.

5 결론

지금까지 “JEUS 시작하기”를 통해 다음의 내용을 살펴보았다.

- JEUS 를 설정하고 운영하는 방법
- Session bean 을 패키징하고 deploy 하는 방법
- CMP entity bean 을 패키징하고 deploy 하는 방법
- Servlet/JSP 등의 Web application 을 패키징하고 deploy 하는 방법

만일 위의 내용들을 잘 이해하고 있다면, 이미 JEUS WAS 의 기본을 알고 있다고 할 수 있다.

좀 더 기술적인 내용의 문서를 보려면 JEUS Server 안내서, JEUS EJB 안내서, JEUS Web Container 안내서 등을 참조하기 바란다.

그리고 웹 매니저 툴을 좀 더 효과적으로 사용하길 원한다면 JEUS 웹 매니저 안내서를 읽어보길 권한다.

색인