하나씩 쉽게 따라 해보는 IBM WebSphere Application Server(WAS) v7 – 25

이정운 (juwlee@kr.ibm.com)

하나씩 쉽게 따라 해보는 IBM WAS v7 스물 다섯 번째 이야기를 시작합니다. 스물 다섯 번째 이야기는 3부 강의의 네번째 로서 WebSphere 에서 JNDI 에 대해서 이야기하는 시간을 가져보도록 하겠습니다.

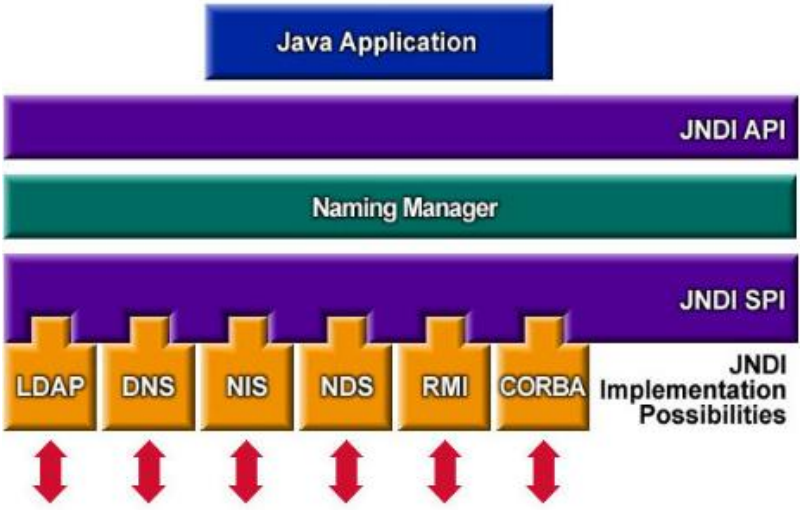
그럼 먼저 JNDI 에 대해서 알고 계시나요? JNDI 는 Java Naming and Directory Interface 의 약자로서 Java 2 SDK, v1.3 과 이후 버전에 포함되며 Java 프로그래밍 언어에서 사용하기 위해 작성된 어플리케이션에 대한 네이밍과 디렉토리 기능을 제공하는 API 입니다.

JNDI(Java Naming and Directory Interface)는 디렉터리 서비스에서 제공하는 데이터 및 객체를 발견(discover)하고 참고(lookup)하기 위한 자바 API다 참고: 한국어 위키피디아

위키피디아에도 간단한 설명만이 있어서 이해가 잘 안가실수도 있는데 네이밍/디렉토리 서비스는 사용자가 원하는 리소스/서비스를 등록하고 찾기 위한 모든 것을 말한다고 생각하시면 됩니다.(쉽게는 전화번호부에 이름을 등록하고 해당 이름을 찾는 서비스랑 매핑해서 이해하시면 이해가 빠르실 것 입니다.^^&;) 자원을 등록하기 위한 이름을 어떻게 할것인가와 이것을 어떻게 찾을 것인가에 대한 문제의 해결을 제시해주는 API 가 바로 JNDI 입니다. 어떤 특정 디렉토리 서비스 구현에도 독립적으로 정의되어져 있기 때문에 다양한 디렉토리 방식에서 공통된 방법으로 접근 가능한 장점을 가지고 있으며 보통 EJB를 사용하거나 Data source 나 Connection factory 등의 객체, 즉 리소스를 찾을 경우에 주로 사용됩니다.

```
Context ctx = new InitialContext();
Object o = ctx.lookup("java:comp/env/ejb/SessionBeanCall");
```

EJB 를 사용할 때, 많이 보던 위와 같은 소스가 바로 JNDI 를 이용해서 해당 객체를 찾는 좋은 예제 입니다. 그럼 이제 강좌를 진행해 볼까요.^^&;



참조: JNDI tutorial

Part 1. JNDI 이해하기 in WebSphere

JNDI 에 대한 강좌를 진행하지만 JNDI 그 자체에 대해서 자세하게 다루지는 않을 예정입니다. 시중에 관련자료도 많고 책도 많으므로 궁금하신 분이나 좀 더 깊게 알고 싶으신 분들은 해당 자료를 참고하시면 충분하다고 생각됩니다. 저는 원래 제가 진행하던 하나씩 쉽게 따라 해보는 IBM WAS v7 강좌의 범위에 맞게 WebSphere 에서 JNDI 에 대한 부분을 좀 더 자세히 이야기 하도록 하겠습니다.

JNDI 가 많이 사용되는 EJB 객체를 기준으로 설명하면, EJB 빈이 전개(deploy)될 때, EJB 컨테이너(여기서는 WAS)는 빈의 JNDI Name 과 함께 해당 빈의 Business Interface 객체를 네이밍 서비스에 등록합니다.(EJB 2.1 까지는 Home 객체였는데, 이게 EJB 3.0 이 되면서 없어지고 Business Interface 만 존재하게 변경되었습니다.) 이를 다시 말하면, WAS 에 해당 어플리케이션을 설치하게 되면, 전개되면서 사용되어 질 EJB 객체들이 네이밍 서비스에 자동으로 등록되어 집니다. (전개 시에 이 작업이 자동으로 진행되며 별도의 작업이 필요하지 않습니다.) 이 작업이 완료되면 EJB 서비스를 사용할 클라이언트는 빈의 JNDI Name 을 통해 해당 빈의 Interface 객체를 얻어올 수 있습니다. (사실 정확히는, EJB 컨테이너가 네이밍 서비스에 등록하는 것은 빈의 Interface 스텁(Interface 를 구현한 RMI 스텁) 객체이고 클라이언트가 JNDI Name 을 이용하여 얻어오는 것도 Interface 의 스텁 객체입니다만 이 부분은 이번 강좌에서는 잊고 진행하도록 하겠습니다.)

이해하기에는 조금 어려운 설명인가요? 그럼 네이밍 서비스에 자동으로 등록되어진다는 말부터 먼저 조금 더 자세하게 예시를 보면서 이야기를 해보도록 하겠습니다.

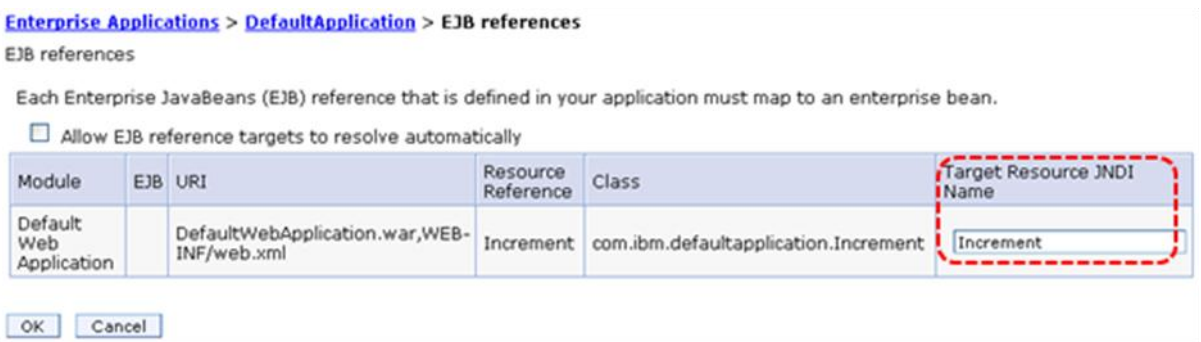
어플리케이션의 전개 시에 명시적으로 바인딩(binding)을 지정해주지 않는다면 interface 는 EJB 컨테이너가 제공하는 기본 이름을 사용해서 바인딩 됩니다. 이때 사용되는 기본 이름은 두 가지가 있는데 short 버전과 long 버전입니다. Short 이름은 java package 이름과 interface 의 class 이름으로 구성되며 long 이름은 short 이름에 component ID 가 접두사로 붙게 되어 있습니다.(여기서 component ID 란 어플리케이션 이름, 모듈 이름 같은 것을 말합니다.)

예를 들어 TEST01EJBWebEAR 이 TEST01EJB.jar 라는 EJB Jar 를 가지고 있으며 그 jar 안에 EJBBankTestBean 이라는 session bean 이 구현되어 있고 local interface 로 itso.bank.service.EJBBankTestService, remote interface 로 itso.bank.service.EJBBankTestRemote 를 구현하고 있다면 자동 생성된 short 이름과 long 이름은 다음과 같습니다.

- ejblocal:itso.bank.service.EJBBankTestService
- ejblocal:TEST01EJBWebEAR/TEST01EJB.jar/EJBBankTestBean#itso.bank.service.EJBBankTestService
- itso.bank.service.EJBBankTestRemote
- ejb/TEST01EJBWebEAR/TEST01EJB.jar/EJBBankTestBean#itso.bank.service.EJBBankTestRemote

local 이름은 ejblocal 이라고 불리어지는 JVM-local namespace 에 바인딩 되며 remote 이름은 namespace 의 루트 에 ejb/ 접두사를 활용하여 global namespace 에 바인딩 됩니다. (이때, 기본 바인딩에 대한 정보들은 EJB Jar 모듈의 META-INF 디렉토리의 ibm-ejb-jar-bnd.xml 라는 설명 파일(deploy description)에 오버라이드 됩니다.)

팁1) 아무것도 지정하지 않아 기본으로 어플리케이션이 전개되면서 자동 바인딩된 EJB 참조는 원하신다면, WAS 관리콘솔의 Enterprise application > 해당 어플리케이션 이름 > EJB references 메뉴에서 직접 변경 가능합니다.



결국은 위와 같은 방식으로 namespace 에 바인딩 되므로 바인딩 된 short 이름이나 long 이름으로 해당 객체를 쉽게 찾을(lookup) 수 있습니다. 그런데 여기서 한가지 더 생각해야 할 것이 있는데 EJB 3.0 에서 POJO 방식의 개발을 위해서 주로 사용되며 EJB 객체를 찾을 수 있는 @EJB (EJB 어노테이션) 입니다.

```
@EJB
    private HelloBeanLocal helloBean;
또는,
@EJB(name="ejb/testBean", beanName="TestBean", beanInterface=TestBean.class)
```

EJB 어노테이션의 경우는 EJB 3.0 부터 사용가능하며 lookup 코드를 사용하지 않고 자동으로 해당 EJB 를 찾을 수 있도록 해주며, POJO 방식으로 EJB 를 개발할 수 있게 해줄 수 있는 장점을 가지고 있습니다.

실제로 EJB 어노테이션이 수행되는 경우를 좀 더 자세히 들여다 보면, EJB 컨테이너가 EJB 참조를 위한 어노테이션을 만나게 되면 자동으로 참조되는 EJB 를 찾도록 되어 있습니다. 이를 AutoLink 알고리즘이라고 하는데 처음에는 바인딩 파일에 명시적으로 주어진 이름을 이용해서 찾으려고 시도합니다. 이를 실패할 경우에는 interface 를 구현한 참조하는 EJB 모듈 안에서 찾기를 시작합니다. 여기에도 없다면 어플리케이션 안에 정의된 다른 모듈로 범위를 확장해서 찾기를 시작합니다. 이때, 해당 interface 를 구현한 EJB 를 하나 찾았다면 해당 EJB 가 참조 타겟으로 사용되어집니다.

결국, AutoLink 의 범위는 EJB 참조가 나타난 어플리케이션에 제한적입니다. 만약 타겟 EJB 가 참

조가 나타난 클라이언트가 아닌 어플리케이션에 거주되고 있거나, 클라이언트 쪽이 아니라 어플리케이션 서버에 전개되어 있다면 AutoLink 는 작동되지 않습니다. 다시 말하면, AutoLink 는 자기가 볼 수 있는 범위 안에 있는 경우에만 EJB 참조를 찾을 수 있습니다. 그러므로, AutoLink 가 찾지 못하는 경우에는 클라이언트 바인딩 파일에 명시적으로 타겟 바인딩이 정의 되어져야 합니다. (EJB 모듈이라면 ibm-ejb-jar.bnd.xml 파일이며, Web 모듈이라면 ibm-web-bnd.xmi 파일입니다.)

결국, EJB 어노테이션을 사용하든, 그렇지 않든 간에 사용하고자 하는 객체를 네이밍 서비스에 등록하고 이를 호출하고자 하는 클라이언트가 특정 조건에 의해서 해당 객체를 찾아서 사용할 수 있는 서비스를 제공하는 것이 바로 JNDI 입니다.

Part 2. JNDI 이해하기2 in WebSphere

이전 파트에서 JNDI 에 대한 간단한 설명을 했는데 이제는 WebSphere 의 JNDI 에 대해 좀 더 자세히 살펴보도록 하겠습니다. 이미 설명한 것처럼 EJB 어플리케이션의 Interface 스텁 객체를 EJB 컨테이너가 네이밍 서비스로 등록하고 사용자는 이를 호출(Lookup) 하여 원하는 EJB 어플리케이션을 실행하게 됩니다. 하단의 그림은 WebSphere 의 경우 네이밍 서비스를 제공하기 위한 Namespace 위치와 JNDI client 가 lookup 할 수 있는 방식을 쉽게 설명하고 있습니다.

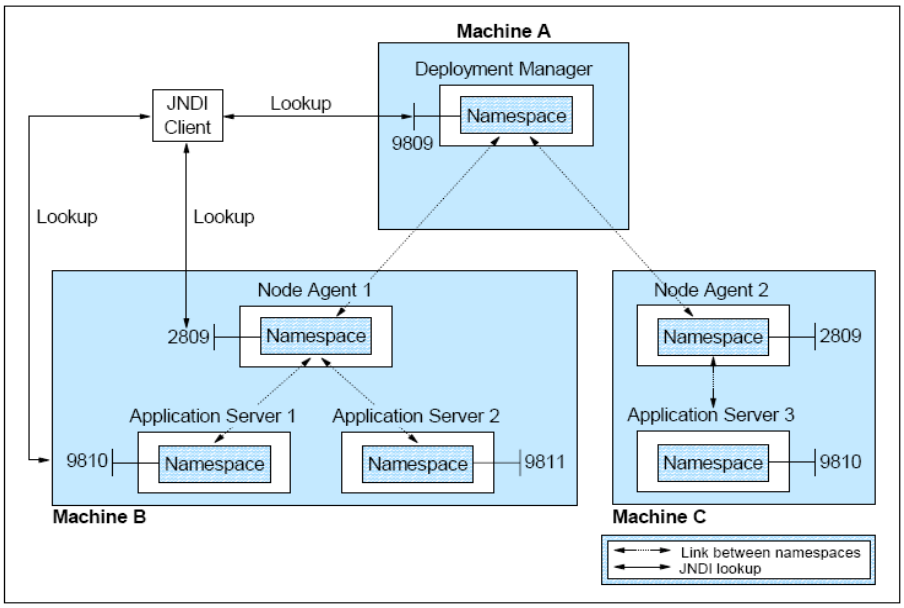
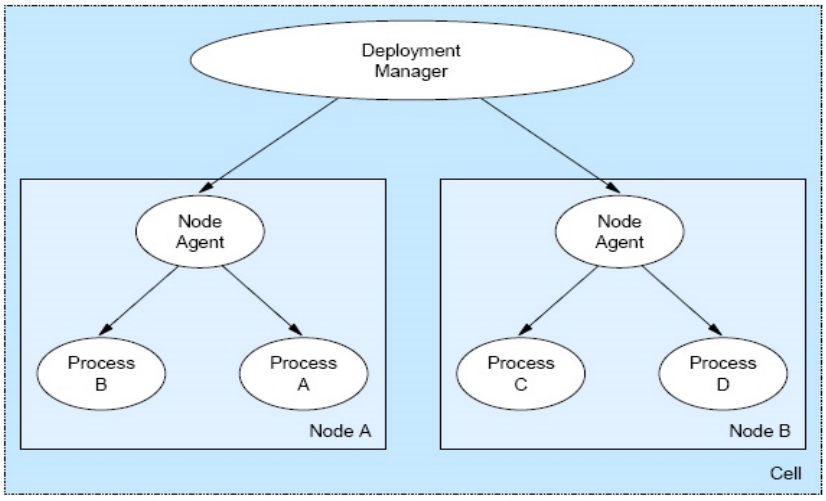


Figure 11-1 Naming topology

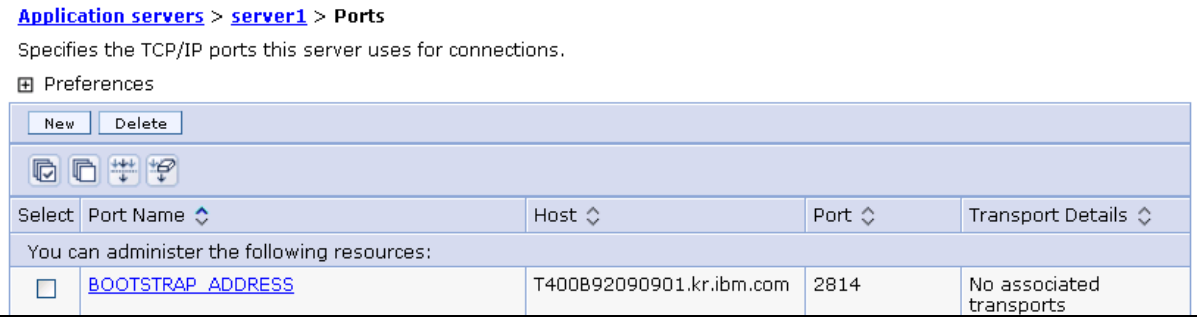
참조: IBM WAS v7 redbook

무언가가 뒤통수를 딱! 하고 치시는게 느껴지시나요? 분명히 namespace 는 이야기 했지만 이런 얘기는 없었는데, namespace 가 하나가 아니고 너무 많이 그림에 나타나 있죠. 그것은 바로 IBM WAS ND 의 토폴로지 구성 때문입니다. 이전 강좌에서 이미 언급 드렸지만 IBM WAS ND 는 관리를 위하여 Dmgr 과 Node agent 등이 사용되는 하단과 같은 토폴로지로 보통 구성되어 있습니다.



이러한 토폴로지 구성과 마찬가지로 관리에 필요한 WAS 노드와 서버들도 모두 namespace 를 가지고 있습니다. Namespace 그림을 잘 살펴보시면 아시겠지만 각 namespace 들은 토폴로지 구조와 마찬가지로 계층구조를 가지고 있으며 링크가 연결되어 있습니다. 즉, Application server 1 의 namespace 에 등록된 EJB 참조를 Dmgr 의 namespace 에 접속해서 찾을 수 있다는 이야기입니다. 그리고 첨부 그림에 나타난 namespace 에 접속 가능한 port 번호는 BOOTSTRAP_ADDRESS 포트입니다. 해당 port 로 접속하여 namespace 에 등록되어 있는 이름을 검색할 수 있습니다.

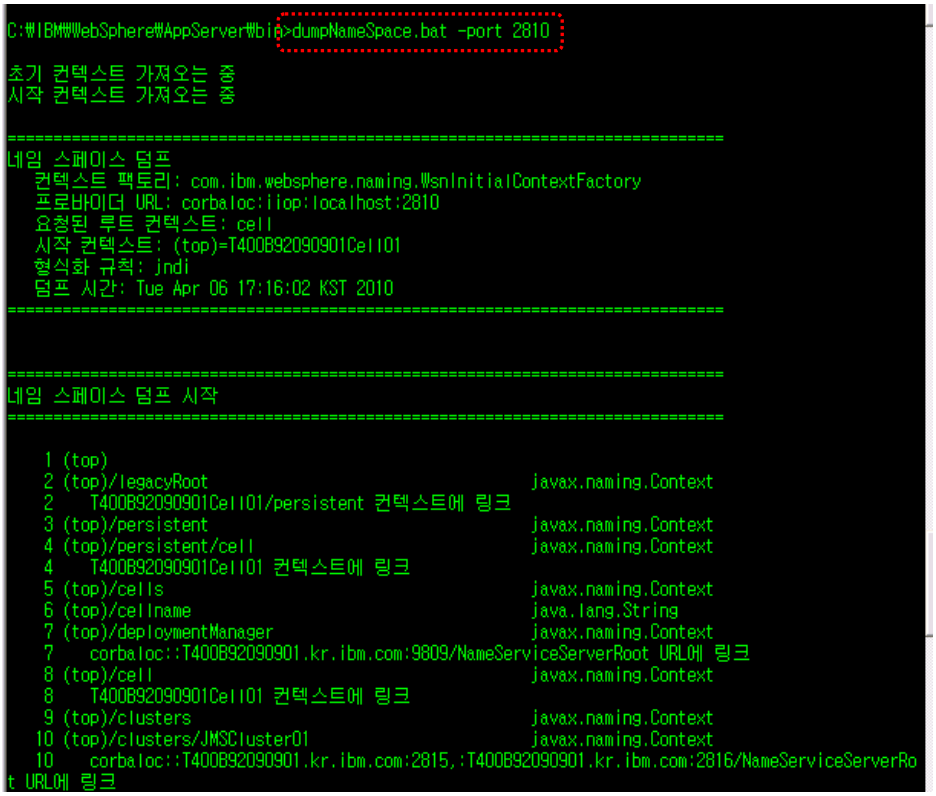
팁2) BOOTSTRAP_ADDRESS 포트는 관리콘솔에서 Application servers > 해당서버 > ports 메뉴에서 하단과 같이 확인할 수 있습니다.

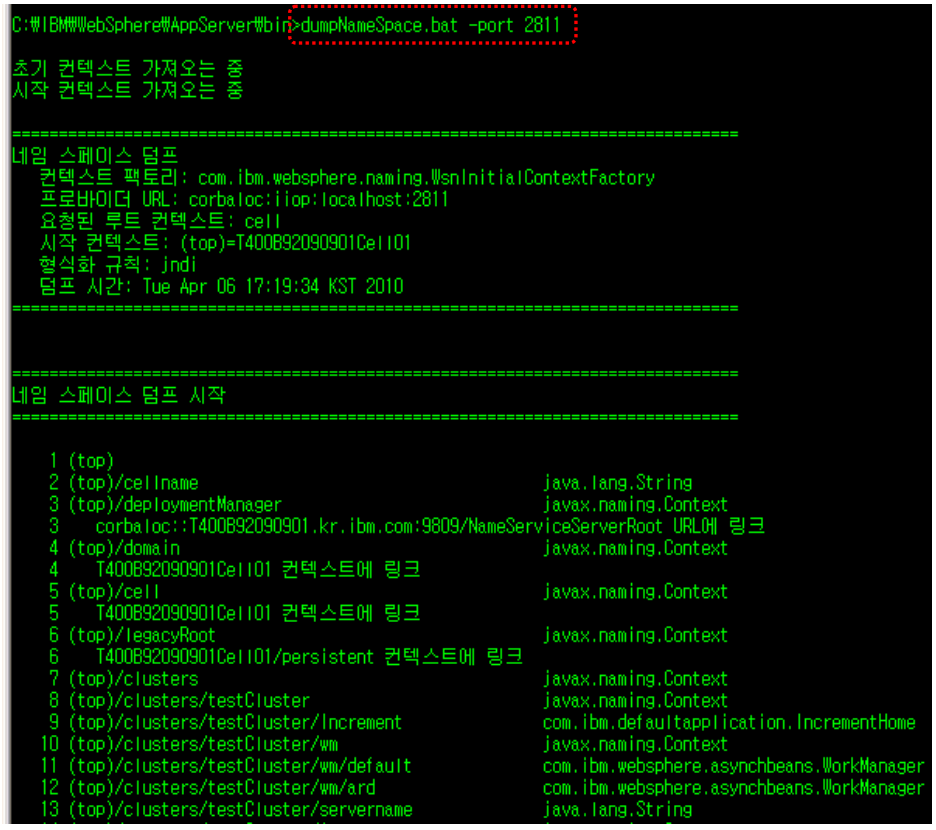


그럼 실제로 해당 Port 에 접속하여 namespace 에 등록된 이름들을 한번 검색해 보도록 하겠습니다. IBM WAS 가 설치된 폴더의 bin 폴더에는 dumpNameSpace.bat/sh 라는 실행파일이 있습니다. 해당 실행파일에 port 를 명시해 주고 실행하면 namespace 에 등록된 모든 이름을 하단과 같이 출력해 줍니다. (여기서는 우선 Dmgr 의 namespace 를 출력해본 것 입니다.)

```
C:₩IBM₩WebSphere₩AppServer₩bin>dumpNameSpace.bat -port 9809

초기 컨텍스트 가져오는 중
시작 컨텍스트 가져오는 중

================================================================================
네임 스페이스 덤프
   컨텍스트 팩토리: com.ibm.websphere.naming.WsnInitialContextFactory
   프로바이더 URL: corbaloc:iiop:localhost:9809
   요청된 루트 컨텍스트: cell
   시작 컨텍스트: (top)=T400B92090901Cell01
   형식화 규칙: jndi
   덤프 시간: Tue Apr 06 17:14:21 KST 2010
================================================================================



================================================================================
네임 스페이스 덤프 시작
================================================================================

    1 (top)
    2 (top)/persistent                                  javax.naming.Context
    3 (top)/persistent/cell                             javax.naming.Context
    3    T400B92090901Cell01 컨텍스트에 링크
    4 (top)/cellname                                    java.lang.String
    5 (top)/nodes                                       javax.naming.Context
    6 (top)/nodes/T400B92090901Node02                   javax.naming.Context
    7 (top)/nodes/T400B92090901Node02/node              javax.naming.Context
    7    T400B92090901Cell01/nodes/T400B92090901Node02 컨텍스트에 링크
    8 (top)/nodes/T400B92090901Node02/persistent        javax.naming.Context
    8    corbaname::T400B92090901.kr.ibm.com:2812/NameServiceNodeRoot#persistent URL에 링크
    9 (top)/nodes/T400B92090901Node02/cell              javax.naming.Context
    9    T400B92090901Cell01 컨텍스트에 링크
   10 (top)/nodes/T400B92090901Node02/nodename          java.lang.String
   11 (top)/nodes/T400B92090901Node02/domain            javax.naming.Context
   11    T400B92090901Cell01 컨텍스트에 링크
```

위와 마찬가지로 port 정보만을 Node agent 의 BOOTSTRAP_ADDRESSS 포트나 보고자 하는 WAS Server 로 변경하면 하단처럼 각각의 namespace 가 가지고 있는 이름들의 출력 결과를 볼 수 있습니다.





팁3) dumpNameSpace.bat 실행파일을 통해서 namespace 에 대한 dump 를 보는 것은 EJB JNDI name 에 대한 호출이 실패했을 때 문제 해결을 위해 접근하기 가장 좋은 시작 포인트중의 하나입니다. 실제로 해당 EJB JNDI name 이 네이밍 서비스에 등록되어 있는지 확인하는 것 입니다.

Namespace 의 dump 결과를 확인해 보시면서 이전에 이야기 드린 short 이름과 long 이름이 아니라 다른 방식으로 이름이 지정된 것을 확인하실 수 있습니다. 그것은 WebSphere 가 가지고 있는 JNDI name 바인딩을 위한 3가지 옵션중의 compound name 형식으로 보여지고 있기 때문입니다.

1) Simple name
   - Simple name 방식은 같은 서버에서 lookup 하거나 lookup 할 대상을 포함하고 있는 서버의 네임 스페이스로 직접 연결되었을 경우 사용 가능합니다.
   예) ejb/webbank/Account
2) Compound Name
   - 주로 같은 서버에서 어플리케이션이 돌지 않을 경우 사용되지만 같은 서버에서 어플리케이션이 돌고 있을 경우에도 범용적으로 사용 가능 합니다. (권장방안)
   예) cell/nodes/node1/servers/server1/ejb/webbank/Account
3) Corbaname
   - 항상 사용 가능하나 deployment 시점의 올바른 경로를 알고 있어야 합니다.
   예) corbaname::myhost1:9812/NameServiceServerRoot#ejb/webbank/Account

위의 리스트에서 볼 수 있는 것처럼 EJB(<ejb-ref>) 와 리소스(<resource-ref>) 객체 이름을 namespace 에 바인딩하기 위하여 상단의 3가지 옵션을 제공하며 가급적 권장하는 방안은 Compound Name 을 사용하는 것 입니다. Compound Name 을 좀 더 살펴보면, 아주 직관적으로 구성된 것을 확인할 수 있습니다. WAS 가 가지고 있는 토폴로지 구조와 동일하게 cell 부터 시작해서 nodes, 그리고 해당 node, 다음으로 servers 에 해당 server 로 단계가 진행되며 그 뒤에 해당 엔티티(EJB 나 리소스) 명을 사용하게 되어 있습니다. (따라서, 익숙해지면 dump 를 보지 않고도 바로 작성할 수 있습니다.)

그러나, Compound Name 은 직관적으로 EJB 객체의 위치를 알 수 있고 권장되는 방안이긴 하나, 사용 하다 보면 너무 길다라는 부담이 있을 수 있습니다. 이런 경우에 사용할 수 있는 것이 JNDI 참조를 사용하는 방식입니다. EJB lookup 을 위해 JNDI 네임을 쓰는 경우 JNDI 네임은 환경에 따라 유동적으로 변동될 수 있습니다. 이 경우 참조를 사전에 만들어 둔 후 java:comp/env 라는 접두어를 사용해서 참조네임으로 원하시는 EJB 를 찾을 수 있습니다. 이 방식의 경우 찾아야 할 EJB 가 거주하는 컨테이너뿐만 아니라 Nodeagent 나 DMGR 의 BOOTSTRAP_ADDRESSS 포트에 연결해서 지정된 이름으로 찾는 것이 가능합니다.

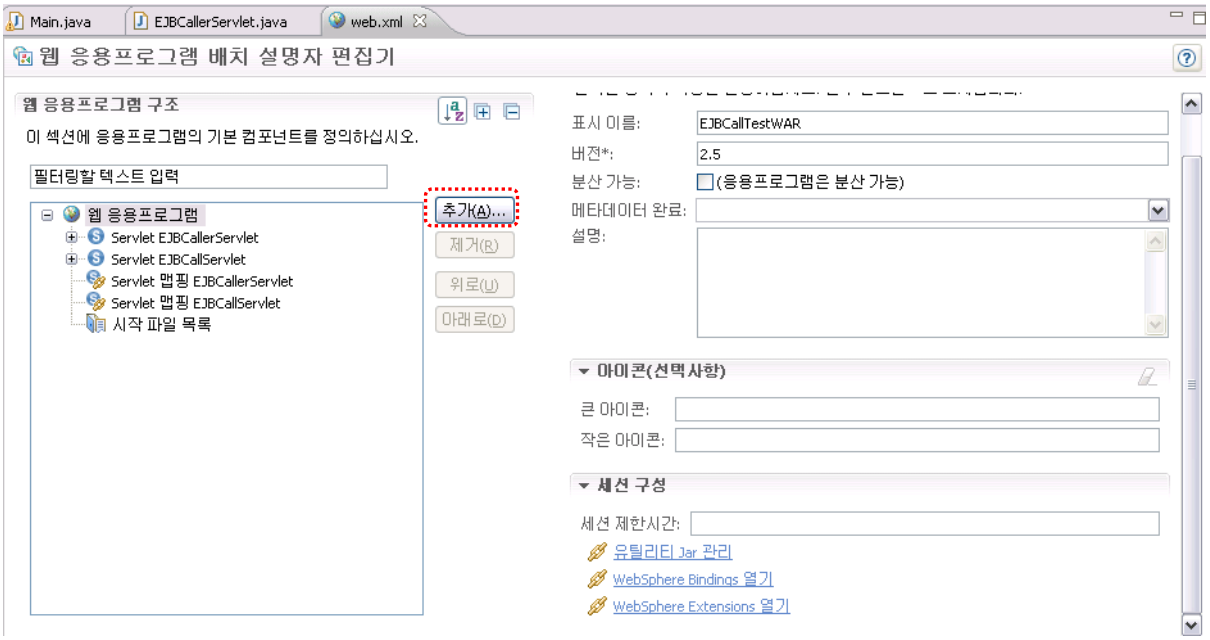
하단의 소스에서 사용되는 방식의 그 예입니다.

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:localhost:2815");

Context ctx;
Object o=null;
try {
    ctx = new InitialContext(env);
    o = ctx.lookup("java:comp/env/ejb/SessionBeanCall");
} catch (NamingException e) {
    e.printStackTrace();
}
```

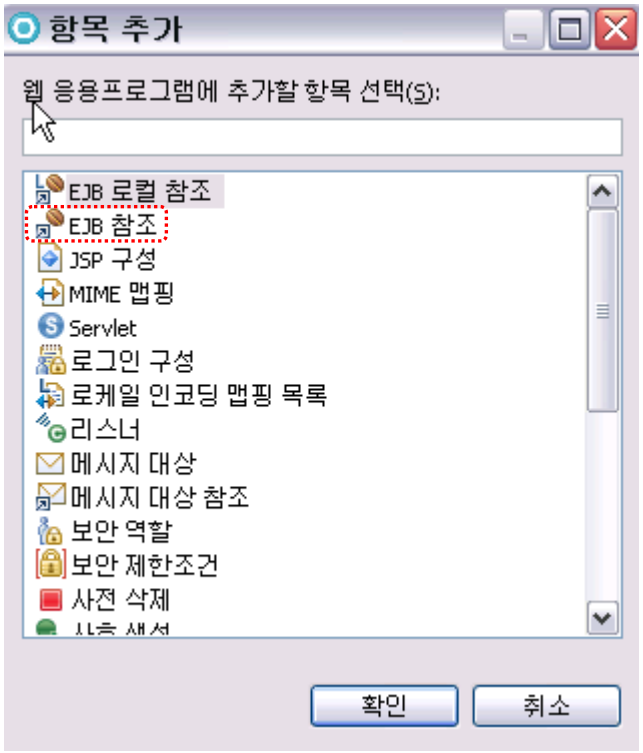
여기서 하나 주의할 점이 있는데, 찾아야 할 EJB 의 컨테이너가 아니라 실제로 lookup 이 실행되는 위치에서 java:comp/env 를 위한 참조를 사전에 정의해야 합니다. 예를 들어 WAR 안의 jsp 나 servlet 에서 호출할 경우에는 web.xml 파일에 <ejb-ref>를 만드셔야 하며 EJB 안에서 lookup 을 하는 경우에는 ejb-jar.xml 에서 참조를 정의해야 합니다.

```
<ejb-ref id="EjbRef_1243398390840">
    <description>
    </description>
    <ejb-ref-name>ejb/SessionBeanCall</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.juwlee.ejbTest.SessionBeanCallHome</home>
    <remote>com.ibm.juwlee.ejbTest.SessionBeanCall</remote>
    <ejb-link>JNDITestEJB.jar#SessionBeanCall</ejb-link>
</ejb-ref>
```

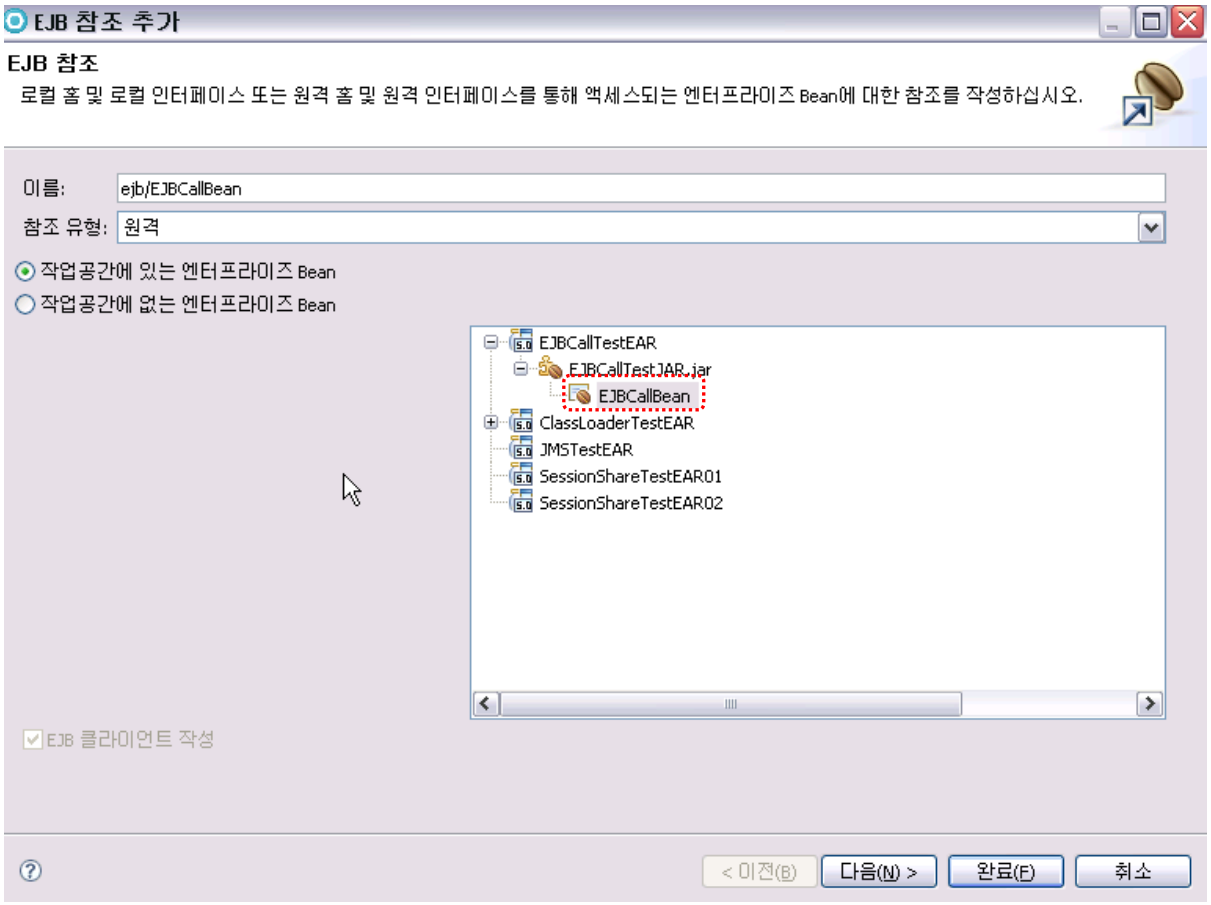
이를 다시 한번 RAD 툴을 이용해서 EJB 참조를 만드는 샘플을 보여드리도록 하겠습니다. RAD 에서 Web 모듈에 JNDI 참조를 건다고 가정하고 Web.xml 을 하단처럼 열고 추가버튼을 클릭합니다.



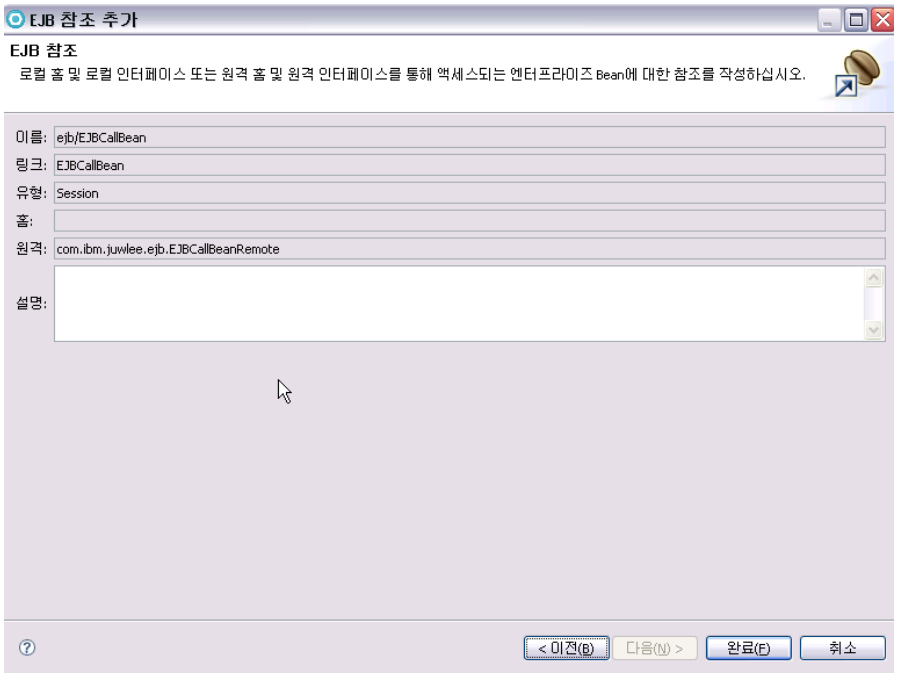
항목 추가 화면이 나오면 EJB 참조를 선택하고 확인을 클릭합니다.



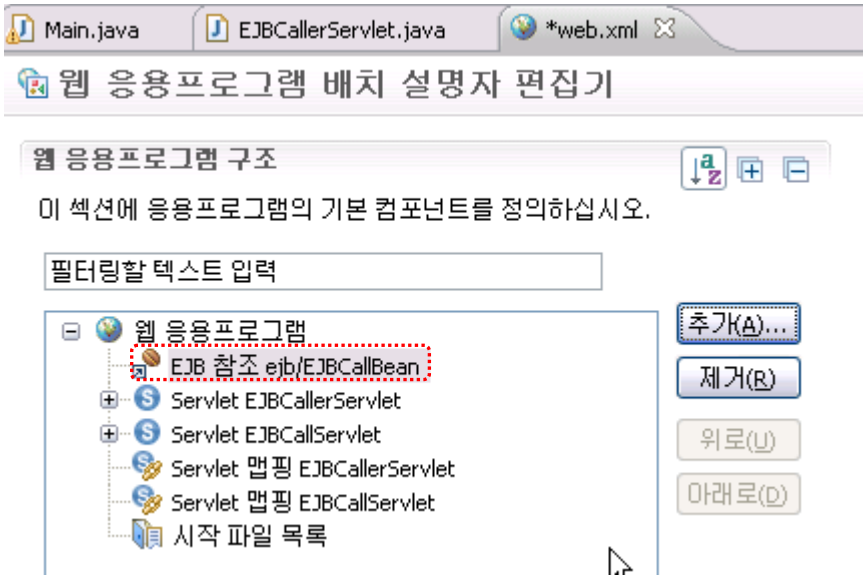
EJB 참조 추가 마법사 화면이 나오면 EJB 참조를 걸고자 하는 EJB 빈을 선택하고 lookup 할 때 사용할 이름을 지정하고 참조 유형도 선택한 후 다음을 클릭합니다.



이름과 실제 호출되어질 Interface 등을 확인하고 완료를 클릭합니다.



그러면 하단에서처럼 정상적으로 EJB 참조가 추가된 것을 확인할 수 있습니다.



Web.xml 의 소스보기를 해보시면 이미 말씀드린데로 하단의 그림에서 보는 것처럼 <ejb-ref> 항목이 추가된 것을 확인하실 수 있습니다.

```
<ejb-ref>
    <description></description>
    <ejb-ref-name>ejb/EJBCallBean</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home></home>
    <remote>com.ibm.juwlee.ejb.EJBCallBeanRemote</remote>
    <ejb-link>EJBCallBean</ejb-link>
</ejb-ref>
```

Part 3. JNDI 의 INS 이해하기

다음으로 JNDI 를 이해하기 위해서는 INS(Interoperable Naming Service) 를 이해하실 수 있어야 합니다. J2EE 1.4 이후 부터는 Interoperable Naming Service(INS)를 통해서 EJB 상호운영성을 지원하기 위하여 CosNaming 서비스를 지원합니다. 이를 위해서 WebSphere 에서는 CORBA URL 방식으로의 접근을 제공합니다. 다시 말해서 CORBA URL 방식을 통해서 WAS의 각 BOOTSTRAP_ADDRESSS port 로 접속할 수 있습니다. (예: WAS, Node agent, DMGR)

CORBA URL 방식은 보통 두 가지 방식으로 분류됩니다. 하나는 corbaloc 방식이고 다른 하나는 corbaname 방식입니다.

Corbaloc 방식은 corbaloc:<protocol>:<addresslist>/<key> 으로 기술됩니다. 여기서 protocol 은 통신을 위해 사용되는 protocol 로 iiop 방식만을 지원합니다. Addresslist 는 host 이름과 port 이름으로 구성할 수 있으며 ',' 를 이용해서 하나가 아닌 여러 주소를 지정할 수 있습니다. (이렇게 여러 주소를 넣는 경우는 해당 서버의 장애를 대비할 수 있으므로 기업 환경에서 많이 사용됩니다.) 마지막으로 key 는 접근 루트를 지정하는 것으로 생략 가능합니다. (key 부분을 다루기에는 너무 복잡해지므로 관심 있는 분은 CORBA 관련 서적을 참고하시기 바라겠습니다.)

예제 corbaloc::myhost corbaloc::myhost1:9333,myhost2:9333/NameServiceServerRoot

Corbaname 방식은 corbaname:<protocol>:<addresslist>/<key>#<INS string-formatted-name> 으로 기술됩니다. 여기서 protocol 은 corbaloc 과 마찬가지로 iiop 이며 다른 것도 비슷합니다. 다만 #<INS string-formatted-name> 부분만이 조금 다른데 여기에 fully qualified path 로 특정 루트 컨텍스트 하위의 엔트리를 직접 지정할 수 있습니다.

예제 corbaname::myhost:9333#cell/nodes/node1/servers/server1/someEJB corbaname::myhost:9333/NameServiceServerRoot /someEJB

CORBA URL 을 기술하는 방식의 패턴만을 이해하면 그렇게 어렵지 않으므로 바로 다음으로 넘어가도록 하겠습니다. 지금까지는 CORBA URL 을 살펴 봤는데 이제는 INS 에서 중요한 포인트중의 하나인 Initial context 에 대해서 살펴보도록 하겠습니다.

JNDI 에서 모든 네이밍과 디렉토리 오퍼레이션은 context 에 연관되어서 수행되어 집니다. 하지만, 절대 루트가 존재하지 않기 때문에 JNDI 는 네이밍과 디렉토리 오퍼레이션을 위한 시작 점을 제공해주는 initial context 를 정의해야 합니다.(찾기의 시작점) WebSphere 에서는 namespace 를 위한 initial context 는 호스트, BOOTSTRAP_ADDRESSS 포트와 연관되며 이러한 연결된 값은 initial context 를 소유한 네임 서버의 주소로서 보여질 수 있습니다. Initial context 를 받아오기 위해서는 initial context 의 네임서버를 위한 호스트와 BOOTSTRAP_ADDRESSS 포트를 알아야만

합니다.

이에 대한 간단한 예제를 살펴보면 하단과 같습니다.

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;


public class testJNDICall {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming.WsnInitialContextFactory");
        env.put(Context.PROVIDER_URL,"corbaloc:iiop:myhost.mytest.com:2809");

        Context initialContext = new InitialContext(env);

    }

}
```

결국, Initial context 의 경우는 EJB 나 리소스를 찾기 위한 검색을 처음으로 시작할 시작점 루트로 생각하시면 됩니다. 즉, 여기서부터 찾겠다라는 의미입니다.

Part 4. JNDI 를 이용한 Lookup 샘플

다음으로 JNDI 를 이용한 lookup 샘플을 몇 개 보여드리도록 하오니 실제 어플리케이션 개발시에 참고하시기 바라겠습니다.

1) Simple name 을 이용한 찾기 예제

- Simple name 을 사용하기 위해서는 lookup 을 할 EJB 가 거주하는 WAS 컨테이너로 직접 접속을 해야 하며 Nodeagent 나 DMGR 로 접속해서 simple name 으로 lookup 을 할 경우에는 java.lang.NullPointerException 이 발생합니다.

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:localhost:2815");

Context ctx;
Object o=null;
try {
    ctx = new InitialContext(env);
    o = ctx.lookup("ejb/com/ibm/juwlee/ejbTest/SessionBeanCallHome");
} catch (NamingException e) {
    e.printStackTrace();
}
```

2) Compound name 을 이용한 찾기 예제

- Compound name 을 사용하는 경우에는 WAS, Nodeagent, DMGR 중 어디든 bootstrap 포트로 접속하여 EJB를 lookup 할 수 있습니다.

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:localhost:2815");

Context ctx;
Object o=null;
try {
    ctx = new InitialContext(env);
    o = ctx.lookup("cell/clusters/DC_testCluster02/ejb/com/ibm/juwlee/ejbTest/SessionBeanCallHome");
} catch (NamingException e) {
    e.printStackTrace();
}
```

3) JNDI 참조 방식을 이용한 찾기 예제

- JNDI 참조 방식을 이용한 찾기 예제이며 JSP 로 구현하였습니다.

```
<%
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming.WsnInitialContextFactory");
        env.put(Context.PROVIDER_URL, "corbaloc:iiop:localhost:2815");

        Context ctx;
        Object o=null;
        try {
            ctx = new InitialContext(env);
            o = ctx.lookup("java:comp/env/ejb/SessionBeanCall");
        } catch (NamingException e) {
            e.printStackTrace();
        }
        SessionBeanCallHome home=(SessionBeanCallHome)PortableRemoteObject.narrow(o,SessionBeanCallHome.class);
        SessionBeanCall sbc = null;

        try {
            System.out.println("create start");
            sbc = home.create();
        } catch (CreateException e) {
            e.printStackTrace();
        }
        System.out.println("Number : " + sbc.getNum());
        sbc.increaseNum();
        System.out.println("Finish");
%>
```

지금까지 WebSphere 에서 JNDI 에 대한 부분을 강좌로 다루었습니다. 많이 어렵지는 않지만 JNDI 네이밍 부분에는 제공하는 벤더별로 약간의 자율성이 있기 때문에 WebSphere 에서 EJB 호출이나 리소스 참조를 사용하실 경우에는 잘 알아두고 숙지하셔야 합니다. 다시 말씀 드리지만, 이 강좌도 가급적 꼭 숙지하시길 바라며, 이번 강좌도 여기서 마무리 하도록 하겠습니다. 이만 ~~~~~휘리릭... ^^&;

참고 1) IBM WebSphere Application Server v7.0 InfoCenter
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/welcome_nd.html

참고 2) IBM WebSphere Application Server v7.0 InfoCenter
- Naming and directory
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.express.doc/info/exp/ae/welc6tech_nam.html?resultof=%22%6a%6e%64%69%22%20