Enterprise Java Beans 2부

중앙대학교 컴퓨터공학과
자바 동호회
JSTORM
http://www.jstorm.pe.kr

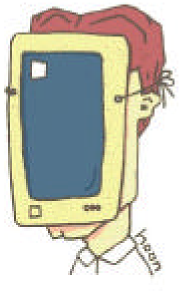
Document Information

Document title:	Enterprise Java Beans 2부
Document file name:	EJB2_JSTORM.doc
Revision number:	<0.9>
Issued by:	<박지훈> pjh@jstorm.pe.kr <남궁윤> yoonng@hanmail.net <김종윤> foxkjy@chollian.net <전부현> kanaloa@orgio.net 정리 : <윤준호> junoyoon@orgio.net
Issue Date:	<2000/9/7 >
Status:	final

Content Information

Audience	EJB 프로그래밍을 학습하는 중급 JAVA 프로그래머
Abstract	간단한 EJB 프로그램을 따라하기 식으로 짜봄으로 감을 잡고 EJB 아키텍쳐에 대한 이해와 EJB 구성요소의 역할과 기능, 서비스등을 알아본다.
Reference	1) Enterprise Java Bean, Second Edition (Oreill'y) 2) Mastering EJB (Wiley) 3) http://www.itplus.co.kr 자료 4) http://www.bea.com의 자료 5) http://www.javasoft.com의 자료 6) http://www.javaworld.com의 자료 7) Sun사의 EJB 스펙 1.1 PDF파일 8) J2EE SDK의 example9 9) J2EE blueprint 및 java pet store example
Benchmark information	

Document Approvals

고문	Signature 	date
지위	Signature	date

Revision History

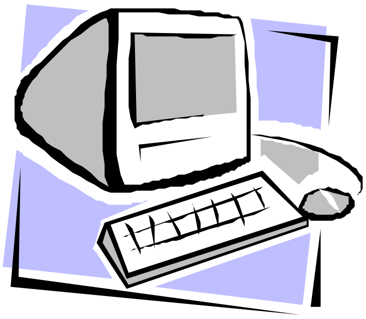
Revision	Date	Author	Description of change

Table of Contents

2 부

Enterprise Java Beans 시작

2부에서는 간단한 EJB 프로그램을 따라하기 식으로 코딩해 봄으로 감을 잡고 EJB 아키텍쳐에 대한 이해와 EJB 구성요소의 역할과 기능을 알아본다.



Chapter 4

EJB 프로그래밍 맛보기

간단한 엔터프라이즈 빈 개발절차

- 비즈니스 로직을 리모트 인터페이스(*.java)안에 메쏘드로 선언한다.
- 리모트 인터페이스안에 선언된 비즈니스 로직을 클래스로 구현한 EJB 빈 클래스(*.java)를 정의한다.
- EJB 객체를 생성하고 삭제해주는 홈 인터페이스(*.java)를 선언한다.
- “가”,”나”,”다”의 소스코드를 컴파일한다.(*.class)
- Deployment Descriptor를 작성한다.(*.xml) (선택)
- “라”,”마”의 결과물들을 ejb-jar파일로 초기 패키지 화한다. (*.jar)
- 컨테이너의 종류에 따라 적절히 Deploy한다.
- 만약 “마.” 의 작업을 하지 않았다면 여기서 해야 한다.
- GUI 또는 Command Line등의 2가지 방법이 존재한다.
- EJB 컴포넌트를 호출하는 클라이언트 코드를 작성하여 테스트한다. (*.java, *.class)

Simple Session Bean 개발

“Hello, World”를 출력하는 간단한 Stateless Session EJB 빈을 개발해보자

리모트 인터페이스 정의

```
import javax.ejb.*;
import java.rmi.RemoteException;
import java.rmi.Remote;

/**
```

```
* This is the HelloBean remote interface.
*
* This interface is what clients operate on when
* they interact with EJB objects. The container
* vendor will implement this interface; the
* implemented object is the EJB object, which
* delegates invocations to the actual bean.
*/

public interface Hello extends EJBObject {
        /**
        * The one method - hello - returns a greeting to the client.
        */
        public String hello() throws java.rmi.RemoteException;
}
```

<소스> Hello.java

EJB 빈 클래스 정의

```
import javax.ejb.*;

/**
* Demonstration stateless session bean.
*/
public class HelloBean implements SessionBean {
    SessionContext ctx;
        // EJB-required methods
        public void ejbCreate() {
                System.out.println("ejbCreate()");
        }
        public void ejbRemove() {
                System.out.println("ejbRemove()");
        }
        public void ejbActivate() {
                System.out.println("ejbActivate()");
        }
        public void ejbPassivate() {
                System.out.println("ejbPassivate()");
        }
        public void setSessionContext(SessionContext ctx) {
                System.out.println("setSessionContext()");
        }
        public void setSessionContext(SessionContext ctx) {
                this.ctx = ctx;
        }
        //
        // Business methods
        //
        public String hello() {
        System.out.println("hello()");
                return "Hello, World!";
        }
}
```

<소스> HelloBean.java

홈 인터페이스 정의

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

/**
* This is the home interface for HelloBean. This interface
* is implemented by the EJB Server's glue-code tools - the
* implemented object is called the Home Object and serves
* as a factory for EJB Objects.
*
* One create() method is in this Home Interface, which
* corresponds to the ejbCreate() method in HelloBean.
*/

public interface HelloHome extends EJBHome {
    /*
    * This method creates the EJB Object.
    * @return The newly created EJB Object.
    */
    public Hello create() throws RemoteException, CreateException;
}
```

<소스> HelloHome.java

Deployment Descriptor 작성

- EJB 1.1에서는 XML파일로 작성한다.(EJB1.0에서는 객체 직렬화(Object Serialization)로 구현함.)
- 내용

Deployment Descriptor Setting	값
Bean Home Name	HelloHome
Enterprise bean class name	HelloBean
Home Interface class name	HelloHome
Remote Interface class name	Hello
Stateful or stateless	STATELESS_SESSION
Transaction 지원	컨테이너가 담당

- Deployment Descriptor XML파일

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>

<ejb-jar>
    <enterprise-beans>
      <session>
     <ejb-name>Hello</ejb-name>
     <home>HelloHome</home>
     <remote>Hello</remote>
     <ejb-class>HelloBean</ejb-class>
     <session-type>Stateless</session-type>
     <transaction-type>Container</transaction-type>
      </session>
    </enterprise-beans>
    <assembly-descriptor></assembly-descriptor>
</ejb-jar>
```

<소스> ejb-jar.xml

- 컴파일하고 ejb-jar파일로 패키지화 한다.
  - J2EE가 설치되었다고 가정한다.(J2EE_HOME환경변수 설정)

- 만약 EJB컨테이너가 설치되어있다면 EJB컨테이너가 제공하는 EJB 클래스 라이브러리를 사용할 수 있다.

- 현재 디렉토리 밑에 META-INF디렉토리를 생성하고 그안에 Deployment Descriptor인 XML파일을 집어 넣는다.

- 최종 패키지화

```
c:\> javac -classpath %CLASSPATH%;%J2EE_HOME%\lib\j2ee.jar;. *.java
c:\> copy ejb-jar.xml META-INF\ejb-jar.xml
c:\> jar cvf ejbBean.jar *.class META-INF\ejb-jar.xml
```

Deployment

컨테이너 종류마다 EJB를 Deploy하는 방법이 다르다.

Deploy란 ejb-jar파일을 컨테이너안에 설치하여 엔터프라이즈 빈이 비즈니스 로직을 수행하도록 셋업하는 것이다.

Deploy할 때 컨테이너 내부에서 발생되는 일들

✍✍ ejb-jar파일이 제대로 구성되어 있는 지 검증한다.

✍✍ 리모트 인터페이스를 구현한 객체와 홈 인터페이스를 구현한 객체를 자동으로 생성한다.

✍✍ 엔터프라이즈 빈들끼리의 메시지패싱 프로토콜인 RMI 스텁과 스켈레턴을 자동으로 생성한다.

"Hello, World " EJB를 웹로직 5.1에서 GUI를 통해 Deploy하는 과정

EJB1.1에서부터는 XML 파일을 이용하여 디스크립터를 작성한다. 웹로직에서는 세가지 디스크립터 XML 파일이 존재한다. 아래는 각각의 파일에 대한 설명이다.

✍✍ ejb-jar.xml

이 파일은 EJB역할 중에서 빈 제공자와 빈 취합자(Assembler)에 의해서 작성되는 부분으로 빈 그 자체에 대한 내용을 담고 있다. 즉, home, remote interface의 이름,JNDI lookup 이름등등 빈 그 자체에 대한 내용과 트랜잭션 처리방법이나 보안정책등과 같은 내용을 담고 있다.

✍✍ weblogic-ejb-jar.xml

이 파일은 웹로직에 의존적인 속성에 관련된 부분을 정의하는 파일로서 캐싱, 클러스터링, 최적화 관련된 내용이 들어간다.

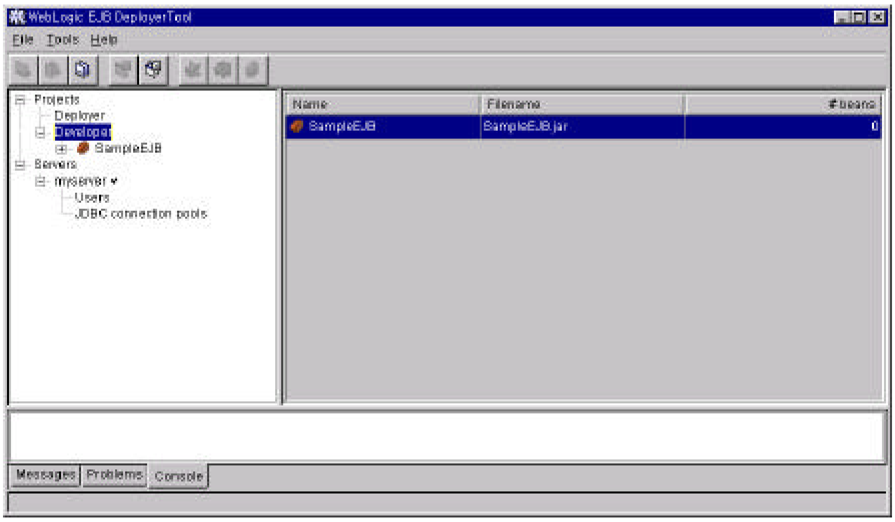
✍✍ weblogic-cmp-rdbms-jar.xml

이 파일은 RDBMS의 persistence를 표현하는 엔티티빈에 대한 정보를 가지고 있다. 어떠한 커넥션 풀을 사용할 것이며, container-managed로 관리되는 빈 클래스의 필드들이 실제 데이터베이스의 어떠한 컬럼과 매칭이 되는지에 대한 정보등등이 들어간다. 이 파일은 세션빈인 경우는 이 파일이 필요가 없다.

개발자나 배치자는 EJB Deployer를 이용하여 빈의 속성에 대한 부분을 GUI방식을 통해 정의하게 된다. 이 툴을 이용하는 경우에는 각각의 속성이 어느 파일에 위치해야 하는지에 대해서는 알 필요가 없다. 이 툴을 이용한 배치과정에서 자동으로 생성되며 해당 파일이 필요한 경우는 이 툴상에서 자동적으로 생성한다. 하지만, 이 툴을 이용하지 않고 직접 이러한 파일들을 수작업을 통해서 작성한다면 각각의 xml 파일에 대한 태그들이나 DTD를 알고 있어야 할 것이다. 그러면, 이 툴을 이용하여 첫번째 빈인 세션빈의 배치 작업을 해보자.

✍✍ JAR 파일이름 정의

EJB1.1 스펙에 의하여 빈과 관련된 모든 인터페이스, 클래스, XML파일, 컨테이너 클래스등등은 하나의 JAR파일로 묶여야 된다. 따라서, 새로운 빈을 작성하기 위해서는 먼저 jar 파일 이름을 정의해야 하고 정의를 하게 되면 Projects 폴더에 새로운 프로젝트가 추가되게 된다. 먼저, File메뉴에서 New를 선택하면 다이알로그가 나타난다. JAR파일을 생성할 디렉토리와 파일이름을 정의한 후에 finish버튼을 누르면 다음 <그림>과 같이 될 것이다. 여기서는 SampleEJB.jar로 정의하였다.

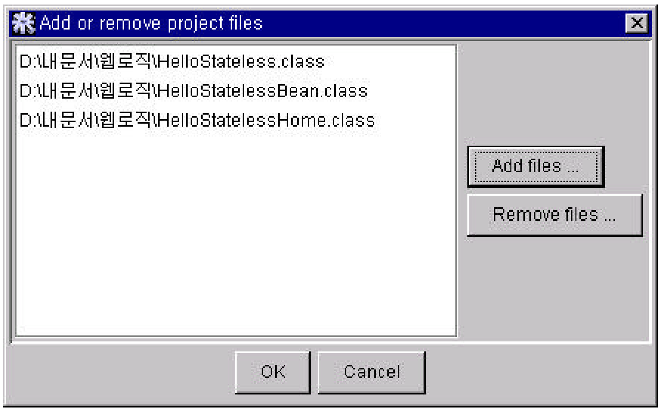


Name	Filename	#beans
SampleEJB	SampleEJB.jar	0

<그림> jar파일 정의

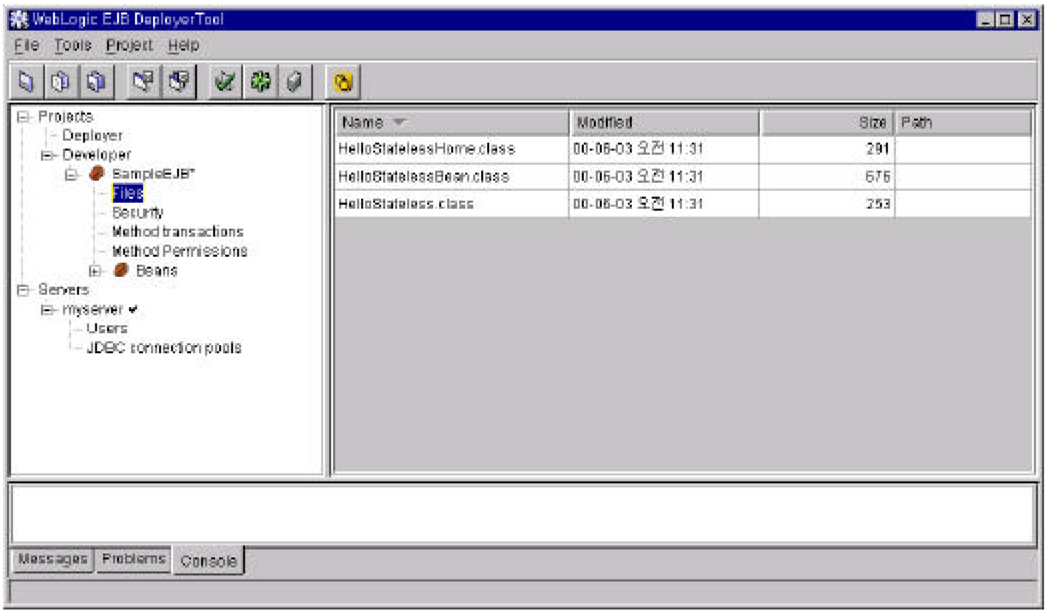
✍✍ "Hello, World" EJB만들기에서 작성된 3개의 클래스를 JAR파일 내에 추가한다
.

홈 인터페이스, 리모트 인터페이스, 빈 클래스등은 jar 파일에 추가되어야 할 클래스들이기 때문에 이것들을 프로젝트상에 추가해야 한다. 위에서 정의한 SampleEJB를 선택하면 메뉴중에서 Project라는 메뉴가 나타나며 Edit File set.. 메뉴 아이템을 선택하면 다이알로그가 나타난다. Add files 버튼을 눌러 1번에서 작성한 3개의 클래스를 추가하면 다음 페이지의 첫번째 <그림>과 같이 된다. OK 버튼을 누르게되면 다음 페이지의 두번째 <그림>과 같이 된다.



D:\내문서\웹로직\HelloStateless.class
D:\내문서\웹로직\HelloStatelessBean.class
D:\내문서\웹로직\HelloStatelessHome.class

<그림> 엔터프라이즈 빈 관련 클래스 첨가



Name	Modified	Size	Path
HelloStatelessHome.class	00-06-03 오전 11:31	291	
HelloStatelessBean.class	00-06-03 오전 11:31	676	
HelloStateless.class	00-06-03 오전 11:31	253	

<그림> 첨가한 결과

주의사항

만약에, 위의 '나'의 작업을 하는 동안에 에러가 발생하는 경우는 작성된 클래스에 문제가 있는 경우일 수가 있다. 이런 경우는 홈 인터페이스나 리모트 인터페이스에서 정의한 메쏘드와 빈 클래스에서 구현한 메쏘드가 다르거나 없는 경우가 대부분이다. 따라서 소스를 수정하도 다시 컴파일 한 후에 이 세개의 클래스를 다시 읽어들어야 한다. Projects -> Edit File Set.. 메뉴 아이템을 선택하면 전에 추가하려고 하였던 3개의 파일이 다시 보이게 되는데 여기서 그냥 확인버튼을 누르게 되면 새로 컴파일된 클래스를 로드하는 것이 아니라 전에 에러가 발생하였던 클래스를 그대로 가지게 된다. 따라서 클래스를 컴파일 한 후에 다시 로드하기 위해서는 이 3개의 클래스들을 위의 <그림>에서 remove files 버튼을 눌러서 모두 제거한 후에 확인 버튼을 누르고 다시 메뉴를 선택하여 파일을 추가해야 새로운 클래스를 읽어들이게 된다.

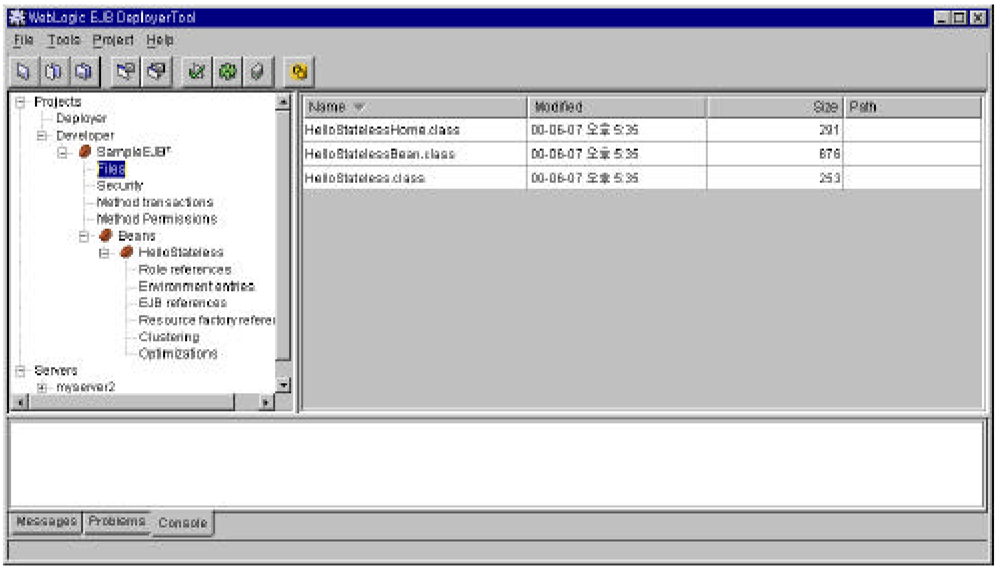
✍✍ **빈에 관련된 속성을 정의한다.**

빈에 관련된 속성은 다음 페이지의 첫번째 <그림>에서 볼 수 있듯이 보안, 트랜잭션, 메쏘드 권한에 대한 부분과 각각의 빈에 대한 환경속성, 참조하는 다른 빈에 대한 정보 및 클러스터링, 최적화 등등의 내용을 담고 있다. Deployment descriptor파일에 정의할 수 있는 모든 속성에 대한 설명은 주제를 벗어나는 것이기 때문에 모든 내용을 여기서 다루지는 않을 것이다. 자세한 내용은 웹로직의 Developer Guide의 EJB관련 부분을 참조하거나 EJB1.1 스펙을 참조하기 바란다. 이 HelloStateless 빈은 데이터베이스처리가 없기 때문에 트랜잭션과 무관하지만 이 툴을 이용하여 어떻게 속성을 정의하는가를 보여주고자 하는 것이 목적이므로 트랜잭션에 대한 속성을 추가하여 보겠다.

트랜잭션 속성을 추가하기 위해서는 Method transactions부분을 선택한 후에 오른쪽 창에 Add버튼을 누르면 새로운 하나의 트랜잭션 속성이 추가가 된다. 하나의 트랜잭션 속성을 추가하면 tx-0이라는 하위 폴더가 생성이 되는데 다음 페이지의 두번째 <그림>과 같이 remote에 트랜잭션 속성은 Required로 정의하도록 하자. 이 뜻은 리모트 인터페이스에 존재하는 모든 메쏘드에 대해서 트랜잭션 속성

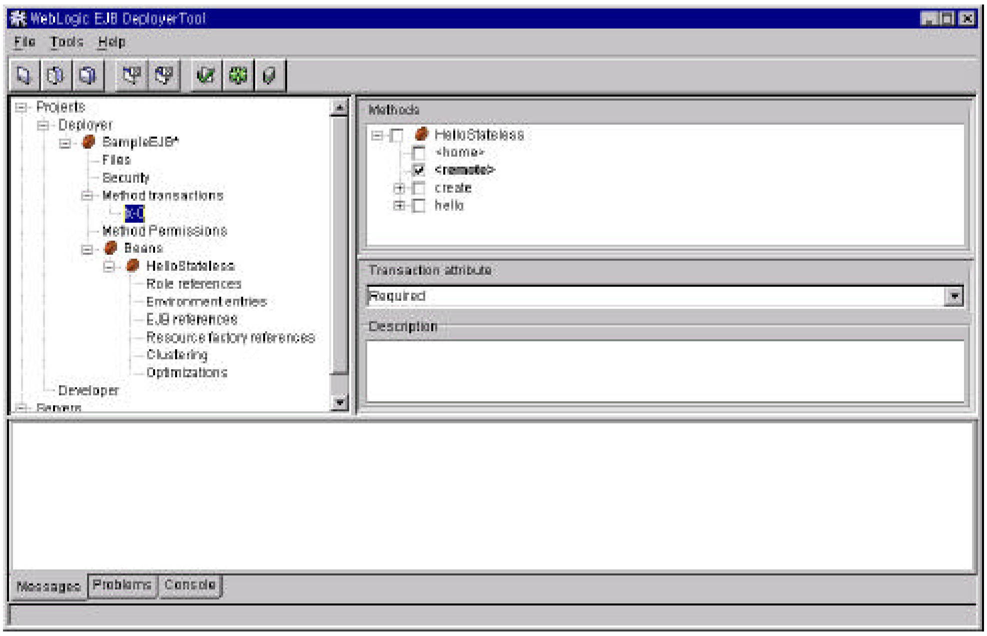
을 Required로 설정한다는 뜻이다.

그리고, JNDI 이름을 등록해야 한다. HelloStateless 폴더를 선택하면 Home JNDI name 부분에 JNDI 이름을 등록한다. 간단하게 이 정도의 속성만을 설정하도록 하자. 이 외의 다른 부분도 GUI 기반으로 속성을 설정하는 부분이기 때문에 직관적으로 쉽게 작성할 수 있을 것이다.



Name	Modified	Size	Path
HelloStatelessHome.class	00-06-07 오후 5:35	291	
HelloStatelessBean.class	00-06-07 오후 5:35	676	
HelloStateless.class	00-06-07 오후 5:35	253	

<그림> EJB 속성들



Methods
□ HelloStateless
□ <home>
☑ <remote>
□ create
□ hello

Transaction attribute: Required

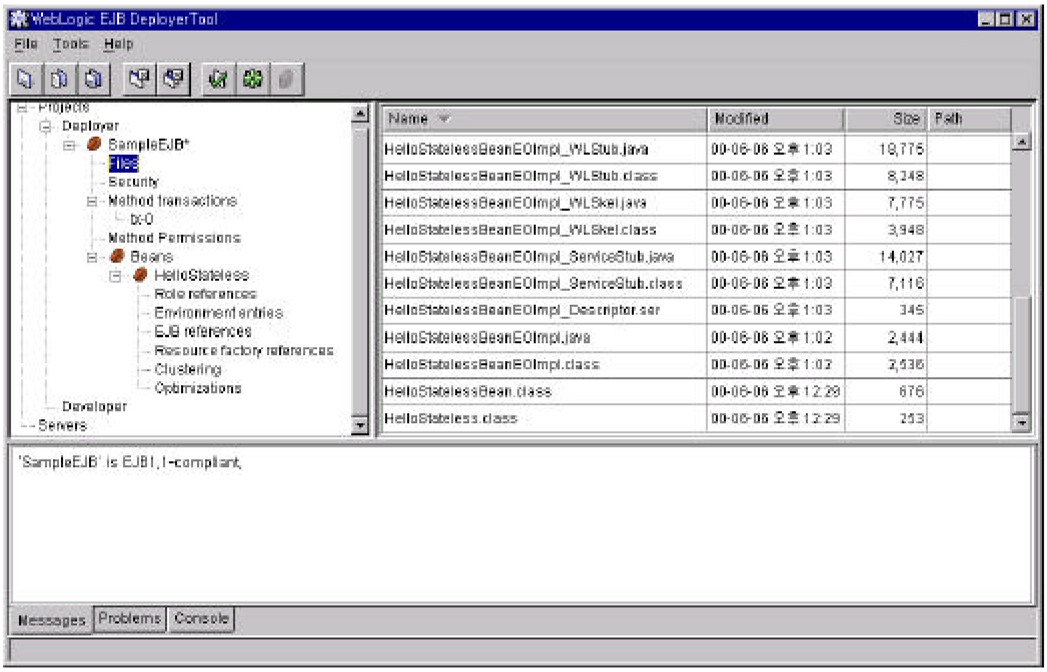
Description

<그림> 트랜잭션 속성 정의

✍✍ 컨테이너 클래스를 생성한다.

이렇게 하여 빈의 속성을 모두 정의하고 난 후에는 Tools ->Generate Container 메뉴아이템을 통해서 컨테이너 클래스를 작성한다. 컨테이너 클래스 작성이 완료

된 후에 File->Save를 한 후에 Files 폴더를 선택하면 다음 <그림>과 같이 새로 작성된 클래스들이 추가되게 된다.



Name	Modified	Size	Path
HelloStatelessBeanEOImpl_WLStub.java	00-06-08 오후 1:03	19,775	
HelloStatelessBeanEOImpl_WLStub.class	00-06-06 오후 1:03	8,348	
HelloStatelessBeanEOImpl_WLSkel.java	00-06-08 오후 1:03	7,775	
HelloStatelessBeanEOImpl_WLSkel.class	00-06-06 오후 1:03	3,948	
HelloStatelessBeanEOImpl_ServiceStub.java	00-06-08 오후 1:03	14,027	
HelloStatelessBeanEOImpl_ServiceStub.class	00-06-08 오후 1:03	7,116	
HelloStatelessBeanEOImpl_Descriptor.ser	00-06-06 오후 1:03	345	
HelloStatelessBeanEOImpl.java	00-06-08 오후 1:02	2,444	
HelloStatelessBeanEOImpl.class	00-06-06 오후 1:02	2,536	
HelloStatelessBean.class	00-06-08 오후 12:29	676	
HelloStateless.class	00-06-06 오후 12:29	253	

'SampleEJB' is EJB1.1-compliant.

<그림> 컨테이너에 의해 생성된 클래스들

✍✍ 웹로직에 등록한다.

이렇게 작성된 후에는 바로 웹로직에 해당 빈을 등록할 수 있다. Tools->Deploy를 선택하면 자동으로 웹로직 서버에 등록된다. 직접 웹로직에 등록하고자 한다면 weblogic.properties에 아래와 같이 등록하면 된다.

```
weblogic.ejb.deploy=d:\내문서\웹로직\SampleEJB.jar
```

주의할 점은 Tools->Deploy를 통해서 웹로직에 등록하는 경우는 웹로직 서버를 운영중에 바로 등록/변경할 수 있지만 weblogic.properties파일에 추가되는 것은 아니다. 따라서 웹로직 서버를 다시 기동하는 경우에는 빈에 대한 정보를 weblogic.properties파일에서 읽어들이므로 전에 등록된 빈을 잃어버리게 된다. 따라서 새로운 빈을 deploy하는 경우에는 weblogic.properties파일도 같이 추가해 주어야 한다.

웹로직 5.1에서 GUI 사용하지 않고 Deploy하는 과정

이 방식은 X윈도우가 지원되지 않는 유닉스와 같은 환경이나 개발시에 여러 개의 빈을 배치작업으로 수행하는 경우에 유용한 방법이다. 실제로 개발시에는 위와 같은 deploytool을 사용하기보다는 배치 스크립트를 작성하여 직접 커맨드 상에서 작업하는 방식을 많이 사용할 것이다.

✍✍ EJB1.1방식으로 EJB를 배치하는 방법은 EJB1.0 에서처럼 DDCreator를 수행할 필요도 없으며 xml 파일과 빈관련 클래스만 묶은 후에 아래와 같은 커맨드만

수행하면 빈을 deployment하는 과정이 끝나게 된다.

- ✍✍ XML 파일은 jar 파일안에 path로 META-INF라는 하위 디렉토리에 존재해야 한다. 그렇지 않으면 이 XML 파일을 찾지 못한다. 예로, 아래와 같이 클래스와 xml 파일을 jar로 묶은 후에 weblogic.ejbc 유틸리티를 수행시키면 target.jar 파일이 생성된다.
- ✍✍ 만약에 클래스가 package가 정의되어 있는 경우는 묶고자 하는 클래스들이 package와 같은 형식의 디렉토리 구조로 jar안에 묶여야 한다.
- ✍✍ Command Line 명령어 실행

$>mkdir META-INF (컴파일된 코드의 하위디렉토리에 META-INF를 만든다.)

$>mv *.xml META-INF (Deployment Descriptor인 XML파일을 옮긴다.)

$>jar cvf source.jar *.class META-INF (클래스파일과 XML파일을 초기 패키지화 한다.)

$>java weblogic.ejbc source.jar target.jar (웹로직안에 있는 자바유틸리티를 실행시켜 웹로직용 EJB패키지를 만들어 낸다.)

"Hello, World" EJB빈을 호출하는 클라이언트 코드 및 실행

클라이언트는 자바애플릿, 자바애플리케이션, 서블릿, JSP 등 자바가 실행될 수 있는 어떤 환경도 상관없다.
다음은 자바 어플리케이션 형태로 코드를 작성하였다.

- ✍✍ 클라이언트 코드

```
import javax.ejb.*;
import javax.naming.*;
import java.rmi.*;
import java.util.Properties;

/**
* This class is an example of client code that invokes
* methods on a simple stateless session bean.
*/

public class HelloClient {
   public static void main(String[] args) {
      try {
      /*
      * Get System properties for JNDI initialization
      */
      Properties props = System.getProperties();
```

```
        /*
        * Form an initial context
        */
        Context ctx = new InitialContext(props);

        /*
        * Get a reference to the home object
        * (the factory for EJB objects)
        */

        HelloHome home = (HelloHome) ctx.lookup("HelloHome");

        /*
        * Use the factory to create the EJB Object
        */
        Hello hello = home.create();

        /*
        * Call the hello() method, and print it
        */
        System.out.println(hello.hello());

        /*
        * Done with EJB Object, so remove it
        */
        hello.remove();
        } catch (Exception e) {
        e.printStackTrace();
        }
    }
}
```

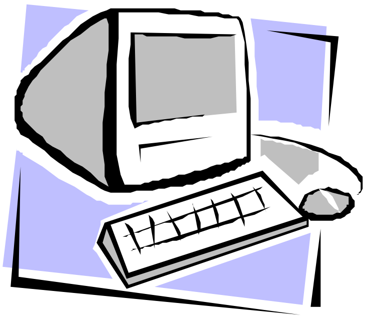
<소스> HelloClient.java

✍✍ **실행방법**

a. 먼저 EJB서버를 실행시킨다. (웹로직으로 가정)

b. 실행

java -Djava.naming.factory.initial=weblogic.jndi.TengahInitialContextFactory -Djava.naming.provider.url= t3://서버이름:7001 HelloClient



Chapter 5

EJB Architecture

개요

EJB Architecture의 목적

EJB 아키텍쳐는 다음과 같은 목적을 가지고 있다.

- EJB 아키텍쳐는 자바 언어를 사용해서 분산 객체지향 비즈니스 애플리케이션을 제작하기 위한 표준 컴포넌트 아키텍쳐이다. EJB 아키텍쳐는 다른 업체의 제품으로 개발된 컴포넌트를 조립함으로써 분산 애플리케이션을 제작할 수 있도록 한다.
- EJB 아키텍쳐는 애플리케이션 제작을 쉽게 해준다. 애플리케이션 개발자는 트랜잭션, 상태관리, 멀티 쓰레드, 컨넥션 풀링과 같은 저수준(low-level) API에 대한 세부사항을 알 필요가 없어진다.
- EJB 애플리케이션은 자바 언어의 'Write Once, Run Anywhere' 철학을 따른다. 따라서 한번 개발된 엔터프라이즈 빈은 재컴파일이나 소스코드 수정없이 배치될 수 있다.
- EJB 아키텍쳐는 개발, 배치, 그리고 엔터프라이즈 애플리케이션의 라이프 사이클에서 런타임시의 측면 등을 기술한다.
- EJB 아키텍쳐는 여러 업체들이 제작한 툴들이 공통적으로 컴포넌트를 개발하고 배치할 수 있는 약속을 정의한다.
- EJB 아키텍쳐는 이미 존재하는 서버 플랫폼들과 호환가능하다. 업체들은 자신의 서버들이 EJB를 지원하도록 확장할 수 있을 것이다.
- EJB 아키텍쳐는 자바 API와 호환가능하다.
- EJB 아키텍쳐는 엔터프라이즈 빈과 자바로 개발되지 않은 애플리케이션 사이의 상호운용성을 제공한다.
- EJB 아키텍쳐는 CORBA 프로토콜과 호환가능하다.

EJB Architecture Roles

EJB 아키텍쳐는 애플리케이션 개발과 배치의 라이프 사이클에 있어서 6개의 구별되는 역할을 정의하고 있다. EJB 아키텍쳐는 각 EJB 역할분담자의 생산물이 다른 EJB 아키텍쳐 역할분담자의 생산물과 호환가능하도록 보장하는 계약을 지정하고 있다. EJB 스펙은 엔터프라이즈 빈의 개발과 배치를 지원하는 계약에 초점을 맞추고 있다.

- 빈 제공자 (bean provider)

  빈 제공자는 업체들이 구입할 수 있고 복잡한 비즈니스 문제를 해결할 수 있는 재사용 가능한 비즈니스 컴포넌트를 제공한다. 빈은 완전한 애플리케이션이 아니며, 완전한 솔루션으로 조립될 수 있는 배치 가능한 컴포넌트이다.

- 컨테이너 제공자 (container provider)

  컨테이너 제공자는 EJB 애플리케이션이 실행되기 위한 저수준(low -level)의 런타임 실행 환경을 제공한다.

- 서버 제공자 (server provider)

  서버 제공자는 컴포넌트를 적재, 관리, 배치할 수 있는 애플리케이션 서버 로직을 제공한다. 현재 EJB 컨테이너 제공자와 EJB 서버 제공자 사이의 구분은 없다. EJB 컨테이너/서버 상품의 예로는 BEA사의 WebLogic, Sun사의 NetDynamics, IBM사의 WebSphere, Oracle사의 Oracle 8i, Persistence Software사의 PowerTier, 그리고 Inprise사의 Inprise 등이 있다.

- 애플리케이션 조립자 (application assembler)

  애플리케이션 조립자는 종합적인 애플리케이션 설계자(architect)로 세부적인 배치(deployment)를 담당할 것이다. 애플리케이션 조립자는 여러가지 컴포넌트들이 어떻게 상호작용하며 애플리케이션이 이들 컴포넌트를 어떻게 사용할 것인지에 대한 이해가 요구된다. 또한 애플리케이션 조립자는 필요에 따라 소수의 컴포넌트들을 개발하게 될 수도 있다.

- 배치자 (deployer)

  배치자는 애플리케이션 조립자에 의해 선택된 컴포넌트들을 가져와 배치 전문기술을 적용하여 하나 또는 그 이상의 애플리케이션 서버에 컴포넌트들을 설치한다.

- 시스템 관리자 (system administrator)

  시스템 관리자는 배치된 시스템을 관리하는 역할을 담당한다.

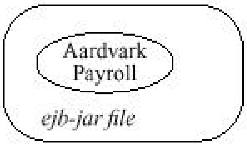
Scenario

다음의 예에서 애플리케이션의 개발, 조립, 배치에 대한 시나오를 살펴봄으로써 EJB 아키텍쳐의 목적과 EJB 아키텍쳐의 역할분담에 대한 이해를 높이도록 한다.

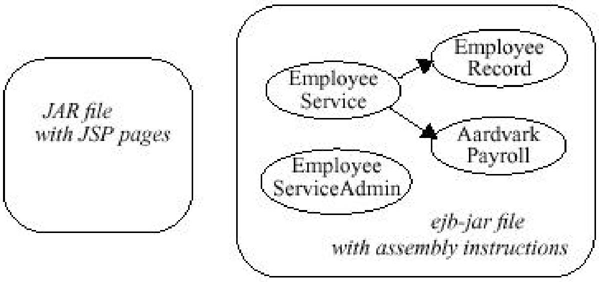
- Aardvark사는 애플리케이션 통합(integration)을 전문으로 하는 회사이다. Aardvark는 자바 기술을 적용한 애플리케이션이 ERP 시스템의 지불 모듈에 접근가능하도록 해주는 일반적인 지불 접근 컴포넌트인 AardvarkPayroll 엔터프라이즈 빈을 개발하였다. Aardvark사는 AardvarkPayroll 엔터프라이즈 빈을 표준 ejb-jar 파일로 패키징하고 이를 주문 엔터프라이즈 빈으로 애플리케이션 제작자들에게 판매하였다. EJB 아키텍쳐의 용어로 Aardvark사는 Aardvark Payroll 빈의 빈 제공자이다.

- Wombat사는 웹 애플리케이션 개발사이다. Wombat사는 직원 셀프서비스 애플리케이션을 개발하였다. 이 애플리케이션은 기업의 직원들이 직원 기록정보에 접근하고 수정하는 것을 가능하게 해준다. 애플리케이션은 EmployeeService, EmployeeServiceAdmin, 그리고 EmployeeRecord 엔터프라이즈 빈으로 구성되어 있다. EmployeeRecord 빈은 컨테이너 관리 엔티티 빈으로, 배치시 기업의 인사관리 애플리케이션과 통합된다. EJB 아키텍쳐의 용어로 Wombat사는 EmployeeService, EmployeeServiceAdmin, 그리고 EmployeeRecord 엔터프라이즈 빈의 빈 제공자 이다.
- Wombat사는 직원 기록에 대한 접근을 제공하는 것과 더불어 직원의 기업의 지불과 수당 계획 시스템에 대한 접근을 제공하려 한다. 지불에 대한 접근을 제공하기 위해 Wombat사는 AardvarkPayroll 엔터프라이즈 빈을 Aardvark사로부터 라이센스하여 애플리케이션에 포함시켰다. 수당 계획 시스템에대한 접근을 위한 일반적인 엔터프라이즈 빈이 존재하지 않기 때문에 Wombat사는 배치시에 알맞은 수당 계획 엔터프라이즈 빈을 개발하기로 결정하였다. 수당 계획 빈은 필요한 애플리케이션 통합 로직을 구현할것이며 수당 계획 빈은 각 Wombat사의 고객들에 맞추어 개발될 것이다.
- 완전한 솔루션을 제공하기 위해서 Wombat사는 직원 셀프서비스 애플리케이션의 비 EJB 컴포넌트를 개발한다. 주로 JSP로 구현될 이 컴포넌트는 엔터프라이즈 빈을 구동시키고 클라이언트에게 HTML 프레젠테이션(presentation)을 제공할 것이다. Wombat사의 고객인 수많은 기업들에대한 판매를 위해서 JSP 페이지와 엔터프라이즈 빈 모두 배치시 알맞게 제작 될 것이다.
- Wombat사의 애플리케이션은 JAR 파일들의 집합으로 패키징될 것이다. 하나의 ejb-jar 파일은 Wombat사에서 개발된 엔터프라이즈 빈들과 Aardvark사에서 개발된 AardvarkPayroll 엔터프라이즈 빈이 포함될 것이다. 그리고 다른 JAR 파일들은JSP 컴포넌트 같은 비 EJB 애플리케이션 컴포넌트를 포함한다. ejb-jar 파일은 어떻게 엔터프라이즈 빈들이 구성될 것인지를 기술하는 애플리케이션 조립 명령어(instruction)를 포함한다. EJB 아키텍쳐의 용어로 Wombat사는 애플리케이션 조립자의 역할을 수행한다.
- Acme사는 서버 소프트웨어 업체이다. Acme사는 EJB 서버와 컨테이너를 개발하였다. EJB 아키텍쳐의 용어로 Acme 사는 EJB 컨테이너 제공자와 EJB 서버 제공자의 역할을 수행한다.
- ABC사는 직원들이 직원 기록과 지불 정보, 그리고 수당 계획 정보를 웹을 통해 접근하고 수정할 수 있도록 하기를 원한다. 정보는 ABC사의 ERP 시스템에 저장되어 있다. ABC사는 Wombat사로부터 직원 셀프서비스 애플리케이션을 구입하였다. 애플리케이션을 운영하기 위해서 ABC사는 Acme사로부터 EJB 컨테이너와 서버를 구입하였다. ABC사의 IT 부서는 Wombat사의 컨설팅 서비스의 도움으로 Wombat 셀프서비스 애플리케이션을 배치하였다. EJB 아키텍쳐의 용어로 ABC사의 IT 부서와 Wombat사의 컨설팅 서비스는 배치자의 역할을 수행하였다. 또한 ABC사의 IT 부서는 Wombat사의 애플리케이션에 ABC사의 수당 계획 애플리케이션에 대한 접근을 제공하는 ABCPensionPlan 엔터프라이즈 빈을 개발하였다.
- ABC사의 IT 직원들은 Acme사의 제품을 설정하고 ABC사의 네트워크 인프라

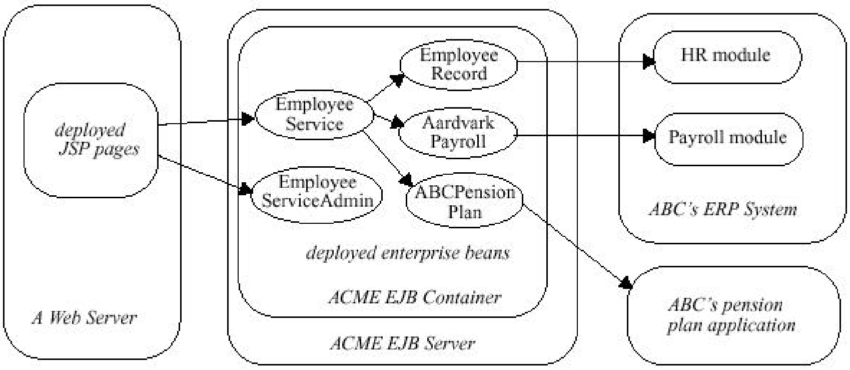
스트럭처(infrastructure)에 통합하는 책임을 지니고 있다. 또한 IT 직원들은 실행시 Wombat사의 애플리케이션의 상태를 모니터하며, 애플리케이션에서 발생한 에러도 처리해야 한다. EJB 아키텍쳐의 용어로 ABC사의 IT 직원들은 시스템 관리자로서의 역할을 수행한다.



✍✍ Aardvark사의 제품은 엔터프라이즈빈의 ejb-jar 파일이다.



✍✍ Wombat사의 제품은 여러 개의 엔터프라이즈 빈들이 애플리케이션으로 조립된 ejb-jar 파일이다. Wombat사의 제품에는 비EJB 컴포넌트도 포함되어 있다.



✍✍ Wombat사의 애플리케이션은 ABC사의 ACME사의 EJB 컨테이너에 배치되었다.

EJB (Enterprise JavaBeans) 컴포넌트

✍✍ EJB는 분산다중계층(distributed multi-tier)환경에서 배치될 수 있는 서버측 컴포넌트를 말한다. EJB 아키텍쳐는 컴포넌트 기반 분산 컴퓨팅을 위한 아키텍쳐이다. 엔터프라이즈 빈은 분산 트랜잭션기반 엔터프라이즈 애플리케이션의 컴포넌트라 할수 있다.

EJB 컴포넌트의 특징

엔터프라이즈 빈은 다음과 같은 특징을 가진다.

- 엔터프라이즈 빈은 기업의 데이터를 처리하는 비즈니스 로직을 포함한다.
- 엔터프라이즈 빈의 인스턴스는 컨테이너에 의해 런타임시에 생성되고 관리된다.
- 엔터프라이즈 빈은 환경 엔트리를 편집함으로써 커스터마이즈 될 수 있다.
- 트랜잭션이나 보안 속성과 같은 여러 서비스 정보들은 엔터프라이즈 빈 클래스로부터 분리된다. 이는 애플리케이션 조립이나 배치 시 툴에 의해서 서비스 정보가 관리될 수 있도록 해준다.
- 클라이언트 접근은 엔터프라이즈 빈이 배치된 컨테이너에 의해 이루어진다
- 만약 엔터프라이즈 빈이 EJB 스펙에 정의된 서비스만을 사용한다면 엔터프라이즈 빈은 어떤 EJB 컨테이너에도 배치될 수 있다. 특수화된 컨테이너들은 EJB 스펙에 정의된 것 이상의 부가적인 서비스를 제공한다. 특수한 서비스를 사용하는 엔터프라이즈 빈은 해당 서비스를 제공하는 컨테이너에서만 배치될 수 있다.
- 엔터프라이즈 빈은 재컴파일이나 소스코드 수정없이 조립된 애플리케이션에 포함될 수 있다.
- 빈 제공자는 엔터프라이즈 빈의 클라이언트 뷰(view)를 정의한다. 빈 개발자는 직접 클라이언트 뷰를 정의할 수 있으며 애플리케이션 개발 툴에 의해 자동적으로 생성되기도 한다. 클라이언트 뷰는 빈이 배치된 컨테이너나 서버의 영향을 받지 않는다. 따라서 빈과 클라이언트가 수정이나 재컴파일 없이 다양한 실행 환경에서 배치될 수 있다.

EJB 컴포넌트의 구성요소

- 엔터프라이즈 자바 빈즈 (Enterprise Java Beans)

  EJB 1.0과 1.1에서는 엔티티 빈(entity beans)와 세션 빈(session beans) 이라는 서로 다른 종류의 엔터프라이즈 빈즈를 정의한다.

  - 엔티티 빈(entity beans) : 엔티티 빈은 지속성있는 데이터를 나타내는 컴포넌트를 말한다.

  - 세션 빈(session beans) : 세션 빈은 비즈니스 프로세스를 위한 로직을 포함하고 있는 재사용 가능한 컴포넌트를 말한다.

  - 배치 디스크립터(Deployment Descripter) 와 JAR 파일

    배치 디스크립터 : 런타임 시에 EJB 서비스들이 각각의 빈 클래스에 어떻게 적용되는지에 대한 정보를 표현한다.

    EJB-JAR 파일 : EJB-JAR 파일은 엔터프라이즈 빈들을 묶기위한 표준 포맷이다.

EJB 컴포넌트 구성

Entity Beans

✍✍ 엔티티 빈(entity beans)이란

엔티티 빈은 지속성 있는 데이터를 표현하는 EJB 컴포넌트를 말한다. 예를 들어 엔티티 빈은 고객, 상품, 은행계좌, 주문 등을 모델링 할 수 있다. 이와 같이 엔티티 빈은 현실 세계의 객체를 나타내며, 이런 객체들은 일반적으로 데이터베이스에 저장되는 레코드에 해당된다.

엔티티 빈의 구성

엔티티 빈은 다음과 같은 인터페이스와 클래스들로 구성된다.

✍✍ 빈 클래스

엔티티 빈 클래스는 지속성 있는 데이터를 모델링 한 자바 클래스를 말한다. 일반적으로 엔티티 빈 클래스는 데이터베이스의 테이블의 구조에 맵핑된다. 이러한 경우 엔티티 빈 인스턴스는 해당 테이블의 하나의 레코드에 맵핑될 것이다. 엔티티 빈 클래스는 엔티티 빈의 리모트 인터페이스나 홈 인터페이스를 구현하지 않으며, 단지 리모트 인터페이스에 정의된 메소드에 대응하는 메소드를 가질 뿐이다.

엔티티 빈 클래스는 javax.ejb.EntityBean을 상속받는다.

✍✍ 리모트 인터페이스

클라이언트가 호출할 수 있도록 엔티티 빈의 인터페이스를 제공한다. 리모트 인터페이스에서는 외부 세계에 제공되는 빈의 비즈니스 메소드를 정의한다.

리모트 인터페이스는 javax.ejb.EJBObject를 상속받으며, 리모트 인터페이스를 구현한 객체를 EJB 객체라 한다.

✍✍ 홈 인터페이스

엔티티 빈의 홈 인터페이스는 엔티티 빈의 EJB 객체를 생성, 검색, 소멸하는 등의 라이프 사이클 메소드를 정의한다.

홈 인터페이스는 javax.ejb.EJBHome을 상속 받으며, 이러한 홈 인터페이스를 구현한 객체를 EJB 홈이라 한다.

✍✍ 프라이머리 키 클래스

프라이머리 키 클래스는 엔티티 빈의 유일한 식별자 역할을 수행한다. 프라이머리 키는 모든 엔티티 빈을 구별 가능하도록 해준다. 프라이머리 키 클래스는 하나 또는 그 이상의 속성으로 표현가능하며, 반드시 직렬화(Serializable)를 구현해야 한다. EJB 1.1에서는 반드시 프라이머리 키 클래스를 정의할 필요는 없다.

엔티티 빈의 정의

✍✍ EntityBean interface

엔티티 빈을 만들기 위해서는 반드시 javax.ejb.EntityBean 인터페이스를 구현해야 한다. 엔티티 빈 인터페이스는 컨테이너가 여러 가지 동작 시에 발생하는 이벤

트를 빈 인스턴스에게 알리기 위해 사용하는 콜백 메소드들을 가지고 있다.

```
public interface javax.ejb.EntityBean  implements javax.ejb.EnterpriseBean {
    public abstract void setEntityContext (javax.ejb.EntityContext);
    public abstract void unsetEntityContext();
    public abstract void ejbRemove();
    public abstract void ejbActivate();
    public abstract void ejbPassivate();
    public abstract void ejbLoad();
    public abstract void ejbStore();
}
```

✍✍ ejbCreate() 메소드

ejbCreate() 메소드는 빈을 초기화하며 데이터베이스의 데이터를 생성하는 역할을 수행한다. ejbCreate() 메소드는 클라이언트가 여러가지 방법으로 엔티티 빈을 생성할 수 있도록 해준다.

ejbCreate() 메소드는 다음과 같은 특징을 지닌다.

- 엔티티 빈 클래스에 ejbCreate() 메소드가 반드시 작성되어야 할 필요는 없다. 개발자는 엔티티 빈을 통해서 데이터베이스의 레코드를 생성할 수 있지만 또한 데이터베이스에 대한 직접적인 접근을 통해 레코드를 생성할 수 있다. 따라서 ejbCreate() 메소드의 작성 여부는 개발자의 선택이다.

- ejbCreate() 메소드의 파라미터는 다양할 수 있다. 개발자는 여러 개의 ejbCreate() 메소드를 정의할 수 있으며 이를 통해 다양한 방법으로 엔티티 빈 데이터를 데이터베이스에 생성하는 것이 가능하다.

- ejbCreate() 메소드는 홈 인터페이스에도 반드시 정의되어야 한다. 클라이언트는 빈 객체에 직접적으로 접근할 수 없음에 유의하여야 한다. EJB 객체는 홈 객체를 통해 생성되므로 각각의 ejbCreate() 메소드는 홈 인터페이스에 create()라는 이름으로 정의되어야 한다.

- AccountBean에 다음과 같이 ejbCreate() 메소드가 정의되었다면

```
public AccountPK ejbCreate(String accountID, String owner) ...
```

홈 인터페이스에도 create() 메소드를 정의해야 한다. (‘ ejb ’ 가 없음에 유의할 것)

```
public Account create(String accountID, String owner) throws ...
```

✍✍ ejbFind() 메소드

엔티티 빈 클래스의 찾기(finder) 메소드는 이미 존재하는 엔티티 빈을 찾는데 사용된다. 따라서 찾기 메소드는 새로운 데이터베이스 데이터를 생성하지 않으며,

단지 존재하는 엔티티 빈 데이터를 로드(load)할 뿐이다. 개발자는 다음의 예와 같이 여러 개의 찾기 메소드를 정의할 수 있으며 이들 메소드들은 모두 다른 작업을 수행할 것이다.

```
/**
* 프라이머리 키를 통해 은행 계정을 검색한다.
*/
public AccountPK ejbFindByPrimaryKey(AccountPK key)
/**
* 입력값 이상의 잔액을 가진 모든 은행 계정을 검색한다.
* 프라이머리 키의 Enumeration을 반환한다.
*/
public Enumeration ejbFindBigAccounts(int minimum)
/**
* 가장 최근의 주문을 검색한다.
*/
public OrderPK ejbFindMostRecentOrder()
```

찾기 메소드들은 다음과 같은 특징을 갖는다.

- 모든 찾기 메소드들은 ‘ ejbFind ’ 로 시작해야 한다.

- ‘ ejbFindByPrimaryKey ’ 라는 이름의 찾기 메소드가 반드시 정의되어야 한다. 이 메소드는 프라이머리 키를 통해 유일한 엔티티 빈 인스턴스를 찾는다.

- 다른 이름과 다른 파라미터를 가지는 여러 개의 찾기 메소드를 정의할 수 있다.

- 찾기 메소드는 검색된 엔티티 빈의 프라이머리 키를 반환한다. 만약 하나 이상의 엔티티 빈을 검색할 경우 Enumeration을 반환한다.

- 찾기 메소드는 홈 인터페이스에도 반드시 정의되어야 한다. ejbCreate() 메소드에서 설명했던 바와 같이 클라이언트는 빈 클래스에 직접적으로 접근할 수 없다. 따라서 찾기 메소드는 홈 인터페이스에도 정의되어야 한다.

- AccountBean에 다음의 찾기 메소드가 정의되었다면

```
public Enumeration ejbFindAllProducts() ...
```

홈 인터페이스에도 찾기 메소드를 정의해야 한다. (‘ ejb ’ 가 없음에 유의할 것)

```
public Enumeration findAllProducts() ...
```

- ejbRemove() 메소드

  데이터베이스의 엔티티 빈 데이터를 삭제하기 위해서 클라이언트는 EJB 객체나 홈 객체의 remove() 메소드를 호출한다. 이때 컨테이너에 의해 빈의 ejbRemove() 메소드가 호출된다. ejbRemove() 메소드는 메모리상의 엔티티 빈 인스턴스를 삭제하지 않으며 단지 데이터베이스 데이터를 삭제할 뿐이다.

- EntityContext

  모든 엔터프라이즈 빈들은 빈의 환경을 구별하는 컨텍스트 객체를 가지고 있다. 이러한 컨텍스트 객체들은 EJB 컨테이너가 설정하는 환경 정보를 포함하고 있다. 빈은 트랜잭션이나 보안 등에 대한 정보를 검색하기 위해서 컨텍스트에 접근한다.

  EntityContext는 빈 인스턴스가 클라이언트에스 사용 가능해지기 전인 라이프 사이클의 초기에 빈 인스턴스에 할당된다. EntityContext는 빈의 인스턴스 필드 내에 존재해야 하고 인스턴스가 존재하는 동안 유지되어야 한다.

```
public interface javax.ejb.EJBContext {
    public abstract javax.ejb.EJBHome getEJBHome();
    public abstract java.util.Properties getEnvironment();
    public abstract java.security.Identity getCallerIdentity();
    public abstract boolean isCallerInRole(java.security.Identity);
    public abstract javax.jts.UserTransaction getUserTransaction();
    public abstract void setRollbackOnly();
    public abstract boolean getRollbackOnly();
}
```

getEJBHome()	엔터프라이즈 빈 클래스의 홈 객체에 대한 레퍼런스를 반환한다.
getEnvironment()	빈과 함께 디플로이된 환경 프로퍼티들의 리스트를 반환한다. 이 메소드는 환경 프로퍼티들에 접근해야 할 필요가 있을때 유용하게 사용될 수 있다.
getCallerIdentify()	빈 인스턴스의 EJB 객체에서 메소드를 호출하고 있는 클라이언트의 식별자를 반환한다.
isCallerRole()	빈의 클라이언트가 인증된 클라이언트인지, 그리고 허가된 작업을 수행하는지 테스트 한다.
setRollbackOnly()	현재의 트랜잭션을 롤백으로 표시한다.
getRollbackOnly()	현재의 트랜잭션이 롤백으로 표시되어 있는지를 테스트 한다.
getUserTransaction()	트랜잭션 구분 인터페이스인 javax.transaction.User-Transaction을 반환한다.

```
public interface javax.ejb.EntityContext
implements javax.ejb.EJBContext {
    public abstract javax.ejb.EJBObject getEJBObject();
```

```
        public abstract java.lang.Object getPrimaryKey();
    }
```

getEJBObject()	빈 인스턴스의 EJB 객체에 대한 리모트 레퍼런스를 반환한다. 이것은 클라이언트 애플리케이션 혹은 다른 빈에 의해서 사용될 수 있는 같은 종류의 레퍼런스이다. 이 메소드의 목적은 빈 인스턴스가 자체 내의 메소드를 수행할 필요가 있을 경우, 이에 대한 레퍼런스를 제공하는 것이다. 루프백(loopback)은 빈이 다른 빈상의 메소드를 호출할 경우에 발생하는데 빈 자신을 하나의 매개변수로 넘겨주는 것을 말한다.
getPrimaryKey()	빈 인스턴스가 현재 할당된 프라이머리 키의 복제를 얻을 수 있게 한다. ejbLoad()와 ejbStore() 메소드의 외부에서 혹은 EJBContext 내 getEJBHome() 메소드에서와 같은 곳에서 이 메소드를 사용하는 경우는 드물다. 그러나 EntityContext는 필요하다면 그런 일상적이지 않은 환경에서 프라이머리 키를 사용할 수 있도록 한다.

EntityBean 클래스의 setEntityContext() 메소드는 빈 인스턴스가 인스턴스 풀에 생기기 전에 호출된다. 인스턴스 풀에는 엔티티 빈과 세션 빈의 인스턴스가 사용되어지기 위해서 보관된다. 인스턴스 풀에 있는 엔티티 빈 인스턴스 들은 데이터베이스 내의 데이터와는 관계가 없다. 그리고 그 상태는 유일하지 않다. 클라이언트에서 특정 엔티티에 대한 요청이 있을 경우에, 풀에서 해당 인스턴스가 선택되고, 데이터베이스로부터 데이터가 채워진 후에 클라이언트의 요청을 처리하도록 지정된다. setEntityContext() 메소드는 인스턴스 필드 컨텍스트 내에서 빈에 할당된 EntityContext를 저장한다. 빈 인스턴스가 한 EJB 객체에서 다른 EJB 객체로 옮겨감에 따라 EntityContext를 참조함으로써 얻어질 수 있는 정보는 인스턴스가 할당된 EJB 객체를 반영하여 변한다. 이것은 EntityContext가 정적인 (static) 클래스 정의가 아니라, 인터페이스이기 때문에 가능하다. 이는 컨테이너는 EntityContext를 자신이 관리하는 완전한 클래스로 구현할 수 있다는 것을 의미한다. 빈 인스턴스가 하나의 EJB 객체에서 다른 객체로 바뀜에 따라, EntityContext를 통해서 얻어질 수 있는 몇가지 정보도 바뀔 것이다.

빈 인스턴스가 인스턴스 풀로부터 완전히 제거되고, 빈 인스턴스가 가비지 컬렉션되기 전인 빈 인스턴스의 라이프 사이클에서 그 빈 인스턴스의 Entity Context가 더 이상 컨테이너에 의해서 구현되지 않음을 나타내는 unsetEntityContext() 메소드가 호출된다.

✍✍ ejbLoad() / ejbStore() 메소드

데이터베이스의 데이터가 메모리상의 엔티티 빈 인스턴스로 로드(load)되어 사용될때 엔티티 빈의 데이터와 데이터베이스의 데이터는 반드시 일치해야 한다는 사실을 유의해야 한다. 이는 엔티티 빈 인스턴스의 데이터가 수정된다면 데이터베이스의 데이터 역시 자동으로 수정되어야 함을 뜻한다. 따라서 이때 동기화(synchoronization) 과정이 필요하다.

따라서 데이터베이스와 엔티티 빈 인스턴스간에 데이터를 상호 교환하는 메커니즘이 필요하다. 이러한 메커니즘은 엔티티 빈에서 다음의 두 메소드를 통해 구현

된다.

ejbLoad() 메소드는 지속성있는 저장장치(주로 데이터베이스)로부터 메모리상의 엔티티 빈으로 데이터를 읽어온다. 반면 ejbStore() 메소드는 엔티티 빈 인스턴스의 현재 데이터를 데이터베이스에 저장하는 역할을 수행한다.

✍✍ ejbActivate() / ejbPassivate()

EJB 서버는 필요에 따라 데이터베이스의 데이터를 표현하는 엔티티 빈을 생성하고 관리할 책임이 있다. 클라이언트의 요청에 따라 EJB 서버는 빈을 생성하고 파괴할 것이다. 그러나 클라이언트의 요청마다 빈을 생성하고 파괴하는 것은 매우 경제적이지 못하며 이때 인스턴스 풀링의 개념이 사용된다. 인스턴스 풀링을 사용함으로써 빈의 인스턴스 수를 줄여주고, 따라서 클라이언트에 서비스를 해주는데 소요되는 자원도 줄어들게 된다. 자주 인스턴스를 생성시키고 소멸시키는 것보다 인스턴스 풀링을 사용하여 빈의 인스턴스를 재사용 하는 것이 훨씬 경제적이다.

빈이 인스턴스 풀을 사용함에 있어서 필요한 리소스를 확보하거나 해제하는 과정이 필요하게 된다. 예를 들어 엔티티 빈이 소켓 컨넥션과 같은 리소스가 필요하다고 할 때, 이 리소스는 빈이 인스턴스 풀에서는 필요가 없으며 따라서 빈의 인스턴스가 인스턴스 풀에 들어갈때는 해제 과정이 필요하다. 엔티티 빈에서는 이러한 작업을 위해서 두개의 콜백 메소드를 정의한다.

ejbActivate() 메소드는 빈의 인스턴스가 인스턴스 풀로부터 꺼내질 때 컨테이너에 의해 호출되는 콜백 메소드이다. 이러한 과정을 활성화라 하며 빈의 ejbActivate() 메소드에서는 필요한 자원을 확보할 것이다. 반면 ejbPassivate() 메소드는 빈의 인스턴스가 인스턴스 풀로 들어갈 때 컨테이너에 의해 호출되는 콜백 메소드이다. 이러한 과정을 비활성화라 하며 빈의 ejbPassivate() 메소드는 필요없게 된 자원들을 해제하는 작업을 수행할 것이다.

엔티티 빈의 종류

엔티티 빈에는 두가지 종류가 있으며 퍼시스턴스(persistence)**를 관리하는 방법에 따라 구분된다**.

✍✍ **컨테이너 관리**(container-managed) **엔티티 빈**

컨테이너 관리 엔티티 빈은 그 퍼시스턴스가 EJB 컨테이너에 의해서 자동으로 관리된다. 컨테이너는 빈 인스턴스의 필드가 어떻게 데이터베이스에 대응되어야 하는지를 알고, 데이터베이스 내의 엔티이와 관련된 데이터를 추가, 변경, 삭제하는 것을 자동으로 관리한다.

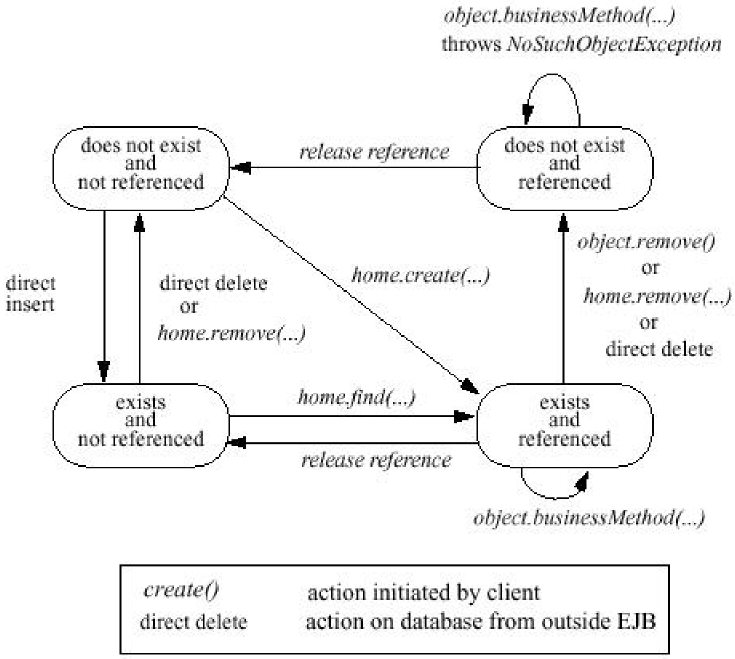
✍✍ **빈 관리**(bean-managed) **엔티티 빈**

빈 관리 퍼시스턴스를 사용하는 엔티티 빈은 이런 모든 작업을 개발자가 명시적으로 한다. 빈 개발자는 데이터베이스를 다루는 코드를 반드시 사용해야 한다. EJB 컨테이너는 빈 인스턴스에게 언제 데이터베이스의 데이터를 추가, 변경, 삭제를 하는 것이 안전한지를 알려준다. 그러나 다른 어떤 도움도 주지 않는다. 빈 인스턴스는 모든 퍼시스턴스 작업을 스스로 한다.

엔티티 빈의 라이프사이클

✍✍ 엔티티 객체의 라이프 사이클

다음의 그림은 클라이언트 측면에서 바라본 엔티티 객체의 라이프 사이클이다. 엔티티 객체는 생성될 때까지 존재하지 않는다. 클라이언트는 엔티티 빈의 홈 인터페이스를 이용해 엔티티 객체를 생성한다. 엔티티 빈의 홈 인터페이스는 컨테이너에 의해 클래스로 구현된다. 클라이언트가 엔티티 객체를 생성할 때, 클라이언트는 새로 생성된 엔티티 객체에 대한 레퍼런스를 확보한다.



<그림> 엔티티 객체의 라이프 사이클

클라이언트는 존재하는 엔티티 객체의 리모트 레퍼런스를 다음과 같은 방법으로 얻을 수 있다.

- 메소드 호출에서 파라미터로 레퍼런스를 받는다.
- 엔티티빈의 홈 인터페이스의 찾기 메소드를 이용해 엔티티 객체를 찾는다.
- 엔티티 객체의 핸들(handle)로부터 레퍼런스를 얻는다.

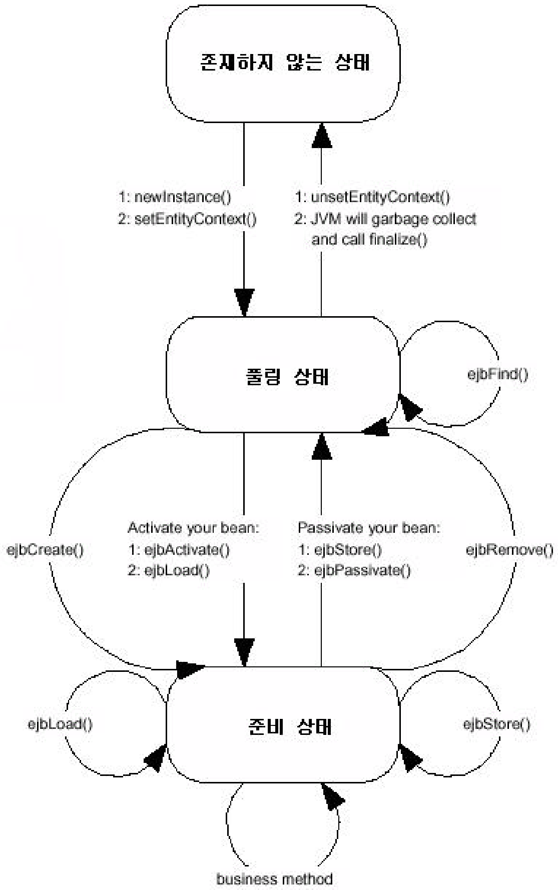
엔티티 객체의 리모트 인터페이스에 대한 레퍼런스를 가지고 있는 클라이언트는 다음과 같은 작업을 할 수 있다.

- 리모트 인터페이스를 통해서 엔티티 객체의 비즈니스 메소드를 호출한다.
- 엔터프라이즈 빈의 홈 인터페이스에 대한 레퍼런스를 얻는다.
- 원격 메소드 호출의 파라미터나 반환값으로 레퍼런스를 넘긴다.
- 엔티티 객체의 프라이머리 키를 얻는다.
- 엔티티 객체의 핸들을 얻는다.
- 엔티티 객체를 제거한다.

존재하지 않는 엔티티 빈에 대한 모든 레퍼런스들은 무효하다. 존재하지 않는 엔티티 빈을 사용하기 위한 시도는 java.rmi.NoSuchObjectException이 발생하는 결과를 낳게 될것이다.

✍✍ 엔티티 빈의 라이프 사이클

엔티티 빈을 잘 개발하기 위해서는 어떻게 컨테이너가 엔티티 빈을 관리하는지를 이해하는 것이 중요하다. EJB 스펙은 초기화된 시점에서 시작해서 가비지 컬렉션 될 때까지 엔티티 빈의 라이프에서 발생하는 모든 주된 이벤트에 대해서 정의하고 있다. 이것을 라이프 사이클이라 한다. 그리고 빈 개발자와 EJB 업체에게 일관성 있는 프로토콜에 따라서 빈과 EJB 서버를 개발할 때에 필요한 모든 정보를 제공한다. 라이프 사이클을 이해하기 위해서, 몇 가지의 라이프 사이클 이벤트를 통해서 엔티티 인스턴스를 추적해 보고, 컨테이너가 이런 이벤트 동안에 어떻게 엔티티 빈과 상호 동작을 하는지를 기술할 것이다. 그림은 엔티티 인스턴스의 라이프 사이클을 나타낸다.



<그림> 엔티티 빈의 라이프 사이클

이제 그림을 따라가며 엔티티 빈의 라이프 사이클에 대해서 알아보자.

- 존재하지 않는 상태

엔티티 빈이 아직 인스턴스화 되지 않은 상태를 말한다.

- 인스턴스 생성

새로운 인스턴스를 생성하기 위해서 컨테이너는 빈 클래스의 newInstance() 메소드를 호출한다. newInstance() 메소드는 엔티티 빈의 기본 생성자를 호출하고 새로운 인스턴스를 메모리로 가져온다. 그런다음 컨테이너가 setEntityContext(EntityContext ctx) 메소드를 호출함으로써 엔티티 컨텍스트를 엔티티 빈에 할당한다. 이 과정은 컨테이너가 풀에서 사용 가능한 엔티티 빈의 인스턴스를 늘리기를 원할 때 일어난다는 사실을 명심해야 한다.

- 풀링 상태

'b' 의 과정 이후 생성된 엔티티 빈은 다른 엔티티 빈들과 함께 인스턴스 풀에 존재하게 된다. 인스턴스 풀에서 빈 인스턴스는 클라이언트 요청을 처리할 수 있는 후보자로, 컨테이너가 사용할 수 있다. 따라서 빈 인스턴스는 어떤 데이터베이스 데이터도 가지고 있지 않으며, 빈에 필요한 리소스도 가지고 있지 않다.

- 준비 상태

빈 인스턴스가 준비 상태에 있을 경우에는, 클라이언트 요청을 받아들일 수 있다. 컨테이너가 빈 인스턴스를 EJB 객체에 할당할 때에, 빈 인스턴스는 준비 상태로 옮겨진다. 이것은 새로운 엔티티 빈이 생성될 때나, 컨테이너가 엔티티를 활성화할 때라는 두가지 상황에서 일어난다.

- 생성을 통한 풀링 상테에서 준비 상태로의 전이

클라이언트가 EJB 홈상의 create() 메소드를 호출할 때에, EJB 컨테이너가 리모트 레퍼런스(EJB 객체)를 클라이언트로 반환할 수 있기 전에 반드시 몇 가지 동작이 이루어져야 한다. 우선 EJB 객체는 반드시 EJB 서버상에서 생성되어야 한다. 일단 EJB 객체가 생성되면, 빈 인스턴스는 인스턴스 풀에서 얻어지고 EJB 객체에 할당된다. 다음으로 클라이언트에서 호출되는 create() 메소드는 빈 인스턴스상의 대응되는 ejbCreate() 메소드에 전달된다. 그 ejbCreate() 메소드가 완료된 후에 프라이머리 키가 생성된다. 컨테이너 관리 퍼시스턴스에서 컨테이너는 자동으로 키를 인스턴스화하고 값을 채운다. 빈 관리 퍼시스턴스에서는 빈이 프라이머리 키를 인스턴스화하고 값을 채우며 ejbCreate() 메소드에서 이를 반환한다. 키는 EJB 객체에 포함되어 식별자(identifier)를 제공한다. 일단 EJB 객체가 식별자를 갖게 되면, 빈 인스턴스의 EntityContext는 프라이머리 키, 리모트 인터페이스, 클라이언트 정보, 보안, 그리고 트랜잭션 관련 정보를 포함한 EJB 객체에 연관된 정보에 접근할 수 있다.

ejbCreate() 메소드가 완료되면, 빈 인스턴스상의 ejbPostCreate() 메소드가 호출된다. 이 시점에서 빈 인스턴스는 그 자신을 클라이언트에서 사용 가능하도록 만들기 전에 필요한 일들을 수행할 수 있다.

마지막으로, ejbPostCreate() 메소드가 성공적으로 수행이 된 후에, 홈은 리모트 레퍼런스(EJB 객체 스텁)를 클라이언트로 반환할 수 있다. 빈 인스턴스와 EJB 객체는 이제 클라이언트로부터 들어오는 메소드 요청을 처리할

준비가 된것이다. 이것이 빈 인스턴스가 풀링 상태에서 준비 상태로 옮겨갈 수 있는 한가지 방법이다.

- 찾기 메소드를 통한 풀링 상태에서 준비 상태로의 전이

찾기 메소드가 호출될 때에, 각 빈은 인스턴스를 풀링 상태에서 준비 상태로의 전이로 인식될 것이다. 빈을 찾게 되면, 그것은 EJB 객체에 할당되고 리모트 레퍼런스는 클라이언트로 반환된다. 발견된 빈은 비활성화된 빈과 같은 통신 프로토콜을 따른다. 그것은 클라이언트가 비즈니스 메소드를 호출할 때에 활성화 된다. 발견된 빈은 비활성화된 빈과 같이 다루어질 수 있고, 앞 섹션에서 기술된 바와 같이 활성화를 통해 준비상태로 옮겨질 것이다.

- 활성화를 통한 풀링 상태에서 준비 상태로의 전이

컨테이너가 빈 인스턴스를 EJB 객체에 다시 할당하려 할때 빈 인스턴스는 활성화될 것이다. 컨테이너는 빈의 ejbActivate() 메소드를 호출하여 필요한 리소스를 확보하도록 한다. 그리고나서 컨테이너는 빈의 ejbLoad() 메소드를 호출하여 데이터베이스 데이터를 빈으로 로드(load)한다.

- 비활성화를 통한 풀링 상태에서 준비 상태로의 전이

빈은 비활성화를 통해 준비 상태에서 풀링 상태로 옮겨질 수 있다. 이것은 빈 인스턴스가 사용되지 않을 때에, EJB 객체와의 연관성을 끊는 과정이다. 빈 인스턴스가 EJB 객체에 할당된 후에, EJB 컨테이너는 인스턴스가 메소드를 수행하는 중이 아니라면 언제나 그 인스턴스를 비활성화할 수 있다. 비활성화의 과정으로서 빈 인턴스상에서 ejbPassivate() 메소드가 호출되어 빈의 리소스를 반환하는 등의 작업을 수행하며, 이어서 ejbStore() 메소드를 통해 빈의 상태를 데이터베이스에 저장한다.

- 제거를 통한 준비 상태에서 풀링 상태로의 전이

빈 인스턴스가 준비 상태에서 풀링 상태로 전이될 수 있는 또 다른 방법은 홈 객체에서 remove() 메소드를 호출하는 것이다. 그러면 컨테이너는 빈 인스턴스의 ejbRemove() 메소드를 호출한다. 엔티티 빈에서 제거 메소드를 호출하는 것은 엔티티 빈의 데이터가 데이터베이스로부터 제거되는 것을 의미하며, 물론 엔티티 빈 인스턴스는 클라이언트의 EJB 객체와의 연관관계는 사라질 것이다.

- 라이프 사이클의 종료

빈 인스턴스의 라이프 사이클은 컨테이너가 풀에서 이를 제거하고, 가비지 컬렉션되는 것을 허용할 때에 끝나게 된다. 이때 해당 빈 인스턴스의 unsetEntityContext() 메소드가 컨테이너에 의해 호출된다. 이 콜백 메소드는 빈이 풀에서 유지하는 모든 자원을 해제하고, EntityContext에 대한 참조를 없애게 된다. 일단 빈의 unsetEntityContext() 메소드가 호출되고 풀에서 제거되면, 가비지 컬렉션 될것이다. 빈의 finalize() 메소드는 다른 자바 객체에서처럼 가비지 컬렉션되기 전에 호출된다. 빈의 라이프 사이클 종료는 컨테이너가 풀에서 인스턴스의 수를 줄이기로 결정했을 경우와 EJB 서버가 셧다운 될 경우 등에 발생할 수 있다.

Session Beans

✍✍ **세션 빈(session beans)이란**

세션 빈은 비즈니스 프로세스를 위한 로직을 포함하고 있는 재사용 가능한 컴포넌트이다. 세션 빈은 서로 다른 빈 간의 상호 작용, 즉 워크플로우(workflow)를 표현하거나 특정 과제를 구현하는 데 유용하게 쓰인다. 엔티티 빈과는 달리 세션 빈은 데이터베이스 내의 공유된 데이터를 표현하는 것이 아니라 공유된 데이터에 액세스하는 역할을 한다. 이것은 데이터를 읽고, 변경하고, 추가하는 데 세션 빈을 사용할 수 있다는 것을 의미한다.

세션 빈의 구성

세션 빈 역시 엔티티 빈과 같은 구성을 가지지만 프라이머리 키 클래스가 존재하지 않는다. 따라서 세션 빈은 빈 클래스, 홈 인터페이스, 리모트 인터페이스로 구성된다.

세션 빈의 정의

✍✍ **세션 빈 인터페이스**

세션 빈을 작성하기 위해서는 javax.ejb.SessionBean 인터페이스를 구현해야 한다. 인터페이스에는 개발자가 구현해야할 메소드들이 정의되어 있다. EJB 컨테이너는 이러한 메소드를 통해서 빈에게 중요한 이벤트를 알려준다. 이들 메소드는 EJB 객체를 통해서만 접근이 가능하므로 클라이언트에 의한 직접적인 호출은 불가능하다.

```
public interface javax.ejb.SessionBean
            extends javax.ejb.EnterpriseBean {
    public abstract void setSessionContext(SessionContext ctx)
                    throws java.rmi.RemoteException;
    public abstract void ejbPassivate()
                    throws java.rmi.RemoteException;
    public abstract void ejbActivate()
                    throws java.rmi.RemoteException;
    public abstract void ejbRemove()
                    throws java.rmi.RemoteException;
}
```

✍✍ setSessionContext(SessionContext ctx) **메소드**

컨테이너는 빈을 세션 컨텍스트와 연관짓기 위해서 이 메소드를 호출한다. 세션 컨텍스트는 컨테이너와 빈이 상호작용을 하기위한 게이트웨이(gateway) 역학을 수행한다. 빈은 컨테이너로부터 트랜잭션 상태나 보안 상태등의 정보를 알아내기 위하여 세션 컨텍스트를 사용한다.

다음은 빈에서 나중에 사용하기 위해 세션 컨텍스트를 멤버 변수에 저장하는 간

단한 예이다.

```
import javax.ejb.*;
public class MyBean implements SessionBean {
    private SessionContext ctx;
    public void setSessionContext(SessionContext ctx) {
    this.ctx = ctx;
  }
...
}
```

✍✍ ejbCreate() 메소드

ejbCreate() 메소드는 세션 빈을 초기화한다. 서로 다른 파라미터를 갖는 여러 개의 ejbCreate() 메소드가 정의될 수 있다. 이를 통해 클라이언트는 여러가지 방법으로 세션 빈을 초기화 할 수 있다. 세션 빈의 초기화를 위해 적어도 하나 이상의 ejbCreate() 메소드가 정의되어야 한다는 사실에 주의해야 한다.

ejbCreate() 메소드는 컨테이너에 의해 호출되는 콜백 메소드이다. 클라이언트는 직접적으로 빈에 접근하여 ejbCreate() 메소드를 호출할 수 없다. 따라서 ejbCreate() 메소드는 다음의 예처럼 홈 인터페이스에도 반드시 정의되어야 한다.

✍✍ public void ejbCreate(int i) throws ...

홈인터페이스에도 create() 메소드를 정의해야 한다.(‘ ejb ’ 가 없음에 유의)

public void create(int i) throws ...

✍✍ ejbPassivate() 메소드

너무 많은 빈이 인스턴스화 되어있을 경우, EJB 컨테이너는 이들 중 일부를 비활성화 시킬 수 있다. 비활성화시 빈은 데이터베이스나 파일과 같은 임시저장장소에 저장된다. 컨테이너는 빈이 가지고 있는 자원들을 해제할 수 있다. 이와 같이 컨테이너가 빈을 비활성화시킨 직후 컨테이너는 ejbPassivate() 메소드를 호출한다.

✍✍ ejbActivate() 메소드

클라이언트가 비활성화된 빈을 다시 사용할 필요가 있을 때 비활성화의 반대 과정이 자동적으로 이루어진다. 컨테이너는 빈을 다시 메모리로 가져옴으로써 활성화 시킨다. 빈이 활성화된 직후 컨테이너는 빈의 ejbActivate() 메소드를 호출한다.

✍✍ ejbRemove() 메소드

세션 빈 인스턴스가 제거될 때 컨테이너는 빈의 ejbRemove() 메소드를 호출한다. ejbRemove() 메소드는 빈이 이제 곧 소멸될것임을 빈에게 알려주는 역할을 수행

한다. ejbRemove() 메소드는 모든 빈에 반드시 구현되어야 한다. 또한 ejbRemove() 메소드는 파라미터를 가지지 않으므로 각 빈마다 하나만 존재한다.

✍ Session context

세션 컨텍스트는 엔티티 컨텍스트와 마찬가지로 EJBContext 인터페이스를 구현하며, 따라서 EJBContext 인터페이스에 정의된 모든 메소드에 접근할 수 있다. 세션 컨텍스트에 정의된 메소드는 getEJBObject() 메소드 뿐이다.

```
public interface javax.ejb.SessionContext
                    extends javax.ejb.EJBContext {
    public javax.ejb.EJBObject getEJBObject();
}
```

getEJBObject()	빈 인스턴스의 EJB 객체에 대한 리모트 레퍼런스를 반환한다. 엔티티 빈의 경우와 마찬가지로 빈이 다른 빈을 호출하며 빈 자신을 파라미터로 넘겨줄 필요가 있을 때 사용한다.

세션 빈의 종류

세션 빈은 두 가지의 기본 형태, 즉 무상태(stateless) 세션 빈과 상태유지(stateful) 세션 빈으로 나누어진다.

✍ 무상태(stateless) 세션 빈

무상태 세션 빈은 메소드로 대표되는 관련된 서비스들의 집합으로, 빈은 한 번의 메소드 호출이 일어난 후, 다음 번 호출까지 상태를 유지하지 않는다. 무상태 세션빈에 메소드를 호출하면, 빈은 이전에 어떤 요청이 있었는지 아니면 이후에 어떤 요청이 있을지를 상관하지 않고, 호출된 메소드를 실행한 다음 결과를 반환한다. 세션 빈은 특정 파라미터를 받아 요청된 사항을 수행한 다음 결과를 반환하는 일련의 프로시져나 배치 프로그램으로 생각하면 된다. 무상태 세션 빈은 소프트웨어 서비스처럼 일반적인 목적에 사용되거나 재사용 가능한 성격을 갖는다.

무상태 세션 빈은 상대적으로 개발하기가 쉽고, 매우 효율적이다. 무상태 세션 빈은 퍼시스턴스를 갖지도 않고, 특정 클라이언트에 할당되는 것이 아니기 때문에 서버의 자원도 거의 요구하지 않는다. 하나의 클라이언트에 할당되지 않기 때문에 많은 EJB 객체들이 빈의 인스턴스가 몇 개만 있어도 충분히 이용할 수 있다.

✍ 상태유지(stateful) 세션 빈

상태유지 세션 빈은 클라이언트 애플리케이션의 연장이다. 클라이언트 대신 작업을 수행하고, 그 클라이언트와 관련된 상태를 유지한다. 상태유지 세션 빈과 클라이언트간의 지속적인 대화를 나타낸다고 하여 이 상태를 ' 대화 상태 (conversational state) ' 라고 한다. 상태유지 세션 빈에 호출된 메소드들은 이 대화 상태로부터 데이터를 읽고, 쓸 수 있는데 이 대화 상태는 해당 빈의 모든 메소

드들에 공유된다. 상태유지 세션 빈은 주로 하나의 시나리오에 특화되는 경향이 있다. 즉 상태유지 세션 빈은 2계층 시스템의 클라이언트 애플리케이션에서 주로 찾아볼 수 있던 로직을 주로 나타낸다.

상태유지 세션 빈은 엔티티 빈과 무상태 세션 빈의 중간적인 성격을 갖는다. 상태유지 세션 빈은 빈 인스턴스가 살아있는 동안 오직 하나의 클라이언트에 할당되며, 상태유지 세션 빈은 클라이언트의 대리인 역할을 한다. EJB 객체들 간에 스와핑이 일어나지도 안혹, 엔티티 빈이나 무상태 세션 빈처럼 인스턴스 풀에 들어가지도 않는다. 상태유지 세션빈의 인스턴스가 만들어지고 EJB 객체에 할당되면, 그것의 라이프 사이클 전체가 할당된 EJB 객체에만 서비스를 제공하게 된다.

세션 빈의 라이프 사이클

✍✍ 세션 객체의 라이프 사이클

다음의 그림은 클라이언트 측면에서 바라본 세션 객체의 라이프 사이클이다. 세션 객체는 생성될때까지 존재하지 않는다. 클라이언트가 세션 객체를 생성할 때 클라이언트는 새로 생성된 세션 객체의 리모트 인터페이스를 가지게 된다. 세션 객체에 대한 레퍼런스를 가지고 있는 클라이언트는 다음의 작업을 수행할 수 있다.

- 세션 객체의 리모트 인터페이스에 정의된 비즈니스 메소드를 호출한다.

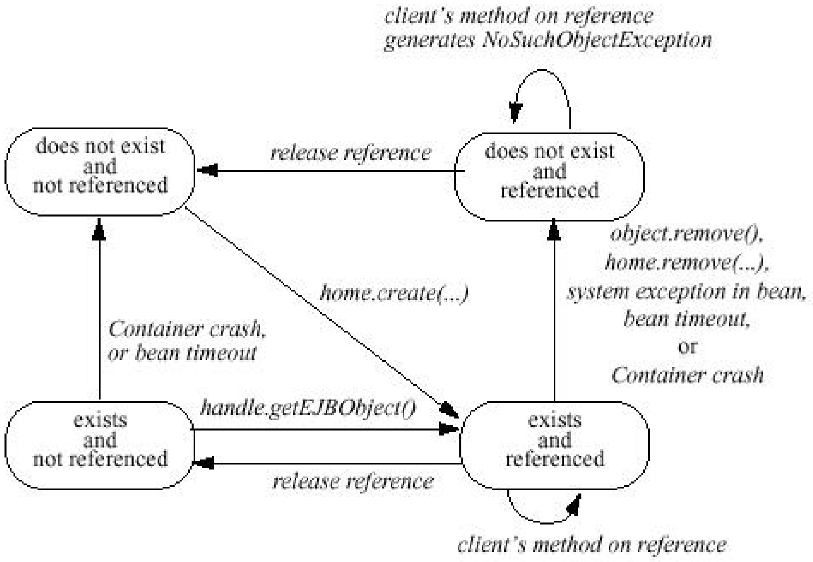
- 세션 객체의 홈 인터페이스에 대한 레퍼런스를 얻는다.

- 세션 객체의 핸들을 얻는다.

- 클라이언트의 범위(scope)안에서 파라미터나 반환값으로 레퍼런스를 넘긴다.

- 세션 객체를 제거한다. 또한 세션 객체의 라이프타임이 만료되었을 때 컨테이너는 세션객체를 자동으로 제거할 수도 있다.

존재하지 않는 세션객체에 대한 레퍼런스는 무효하다. 존재하지 않는 세션 빈을 사용하기 위한 시도는 java.rmi.NoSuchObjectException이 발생하는 결과를 낳게 될것이다.

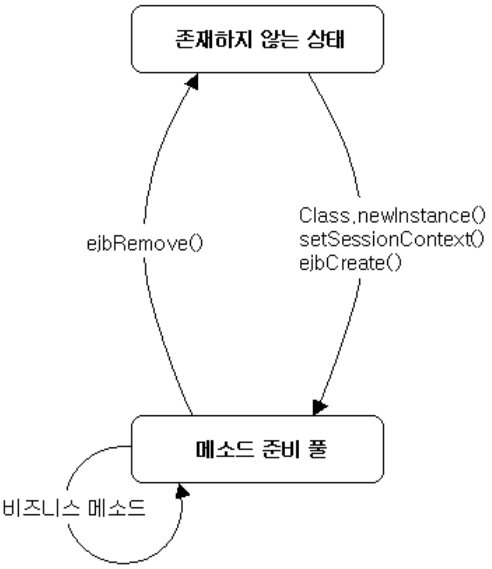


✍✍ **무상태 세션빈의 라이프 사이클**

- 무상태 세션 빈의 라이프 사이클은 컨테이너가 새로운 인스턴스를 생성하기 위해서 세션 빈 클래스의 newInstance() 메소드를 호출할 때 시작된다. 다음으로 컨테이너는 인스턴스의 setSessionContext() 메소드를 호출한 후 ejbCreate() 메소드를 호출한다. 컨테이너는 언제라도 세션 빈을 생성할 수 있으며, 이는 클라이언트의 create() 메소드 호출과는 상관이 없다.

- 세션 빈 인스턴스는 클라이언트의 요청에 의해 비즈니스 메소드를 실행할 수 있다.

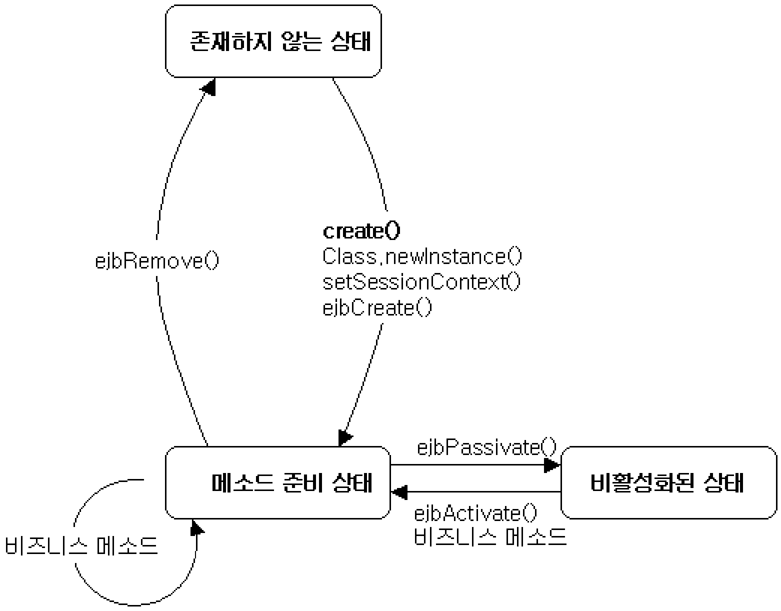
- 컨테이너가 메소드 준비 풀의 인스턴스의 숫자를 줄이고 싶은 경우처럼 더 이상 인스턴스를 유지할 필요가 없을 때, 컨테이너는 ejbRemove() 메소드를 호출한다. 이를 통해 무상태 세션 빈의 라이프 사이클은 종료된다.



<그림 : 무상태 세션 빈의 라이프 사이클>

✍✍ 상태유지 세션 빈의 라이프 사이클

상태 유지 세션 빈과 다른 빈 타입들의 가장 큰 차이점은 상태유지 세션 빈이 인스턴스 풀을 사용하지 않는다는 점이다. 상태유지 세션 빈의 전체 라이프 사이클은 하나의 클라이언트에 대하여 전적으로 할당되므로, 스와핑이나 인스턴스 풀을 사용하지 않는다. 상태유지 세션 빈은 인스턴스 풀에 들어가는 대신 자원을 절약하기 위해 메모리로부터 제거된다. 활동을 하지 않는 기간 동안에는 EJB 객체가 클라이언트에 연결되어 있더라도 빈 인스턴스가 레퍼런스를 잃고 가비지 컬렉션된다. 이는 상태유지 세션 빈이 메모리 절약을 위해 강제 퇴출되기 전에 비활성화되어야 하며, EJB 객체가 다시 활동을 시작하면 다시 상태를 회복하여 활성화되어야 함을 의미한다.



<그림 : 상태유지 세션 빈의 라이프 사이클>

- 클라이언트가 상태유지 세션 빈의 EJB 홈의 create() 메소드를 호출하면 빈의 라이프 사이클이 시작된다. 컨테이너가 create() 메소드를 받게 되면, 컨테이너는 빈 클래스에 대하여 newInstance() 메소드를 호출하여 빈의 새로운 인스턴스를 생성하게 된다. 다음으로 컨테이너는 setSessionContext() 메소드와 ejbCreate( ) 메소드를 호출하고 EJB 객체의 리모트 레퍼런스를 클라이언트에게 돌려준다. 이제 인스턴스는 메소드 준비 상태로 클라이언트가 빈의 리모트 레퍼런스에 호출하는 비즈니스 메소드를 서비스 할 수 있는 상태가 된다.

- 메소드 준비 상태에 있는 동안 빈 인스턴스는 클라이언트로부터 메소드 호출을 받을 수 있게 된다.

- 컨테이너에 의해 어떤 빈 인스턴스가 비활성화 될것인지(메모리로부터 쫓겨날 것인지) 결정된다. 이때 컨테이너는 비활성화될 빈 인스턴스의 ejbPassivate() 메소드를 호출한다. 이 작업이 완료되면 컨테이너는 빈의 상태를 2차 저장장소에 저장한다.

- 빈이 비활성화된 상태에서 클라이언트가 EJB 객체에 어떤 요청을 하면, 컨테이너는 빈 인스턴스를 활성화시킨다. 이때 컨테이너는 2차 저장장소로부터 빈 인스턴스의 상태를 읽어오고 ejbActivate() 메소드를 호출한다.

- 클라이언트가 세션빈을 제거하기 위해서 EJB 홈이나 EJB 객체의 remove() 메소드를 호출하면 컨테이너는 빈 인스턴스의 ejbRemove() 메소드를 호출한다. 이때 상태유지 세션빈의 라이프 사이클은 종료된다.

배치 디스크립터와 EJB-JAR 파일

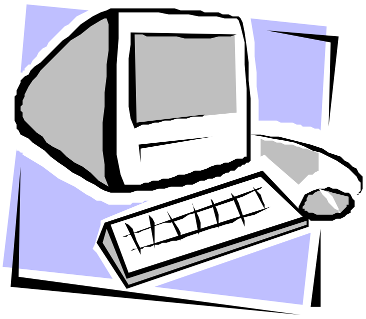
배치 디스크립터 (Deploy Descripter)

엔티티 빈 및 세션 빈의 인터페이스나 클래스에는 런타임 시에 빈이 어떻게 관리되는지에 대한 많은 정보들이 없다. 보안, 트랜잭션, 네이밍, 그리고 기타 분산 객체 시스템에서 제공되는 기본적인 서비스들은 EJB 서버에 의해 자동으로 관리된다. 따라서 EJB 서버는 런타임 시에 어떻게 이런 기본적인 서비스들이 각각의 빈 클래스에 적용되는지에 대한 정보를 가지고 있어야 한다. 이를 위해서 사용되는 것이 배치 디스크립터(Deployment Descriptor)이다. 배치 디스크립터에 대한 자세한 사항은 12장에서 자세히 설명한다.

EJB-JAR 파일

EJB-JAR 파일은 엔터프라이즈 빈들을 묶기위한 표준 포맷이다. EJB-JAR 파일 포맷은 (빈 제공자의 산출물인) 조립되지 않은 엔터프라이즈 빈들을 묶는데 사용되거나 (애플리케이션 조립자의 산출물인) 조립된 애플리케이션을 묶는데 사용된다
.

빈 제공자에 의해 만들어진 EJB-JAR 파일은 애플리케이션 조립 명령을 포함하지 않은 하나 또는 여러개의 엔터프라이즈 빈을 포함한다. 하지만 애플리케이션 조립자에 의해 만들어진 EJB-JAR 파일은 하나 또는 여러개의 엔터프라이즈 빈과 함께 엔터프라이즈 빈들이 어떻게 하나의 애플리케이션 배치 유닛으로 결합될 것인가를 기술한 애플리케이션 조립 정보를 포함한다.



Chapter 6

EJB 주요 서비스

사용자가 많은 대규모의 비즈니스 시스템은 수천에서 수백만 개까지의 객체를 동시에 사용하게 되는데, 이들 객체 간의 상호 작용이 증가하게 되면 동시성 제어나 트랜잭션 문제로 시스템 응답속도가 저하되고, 사용자들은 불만을 갖게된다. EJB 서버는 객체간의 상호작용을 동기화해주고, 자원을 공유할 수 있게 함으로써 수행력을 높여준다. EJB는 런타임 시에 수많은 빈의 관리를 쉽게 하기 위해 두 가지의 메커니즘, 즉 인스턴스 풀링과 활성화라는 두가지 메커니즘을 제공한다.

Instance Pooling

인스턴스 풀링(Instance Pooling)의 개념

콘도에서 휴가를 보내는 것을 생각해 보자. 모든 사람들이 자기 자신의 콘도에서 휴가를 보내기를 원하겠지만, 대부분의 사람들에게 그것은 불가능하다. 이 문제의 해결책은 바로 시간을 공유(time-share)하는 것이다. 당신이 이번 주에 콘도를 사용한다면, 내가 다음주, 그리고 다른 누군가가 그 다음주에 콘도를 사용하는 것이다. 하나의 콘도를 우리 모두가 공유하는 것이다. 이를 통해 싼 가격에 누구나 콘도를 이용할 수 있다.

이와 같이 인스턴스 풀링은 유한한 풀에 인스턴스를 넣거나 꺼내오는 과정을 통해 비싼(expensive) 자원을 관리하는 제어구조이다. 인스턴스 풀링의 가장 효과적이고도 일반적인 예로 데이터베이스 컨넥션이 있다. 데이터베이스 컨넥션은 데이터베이스에 연결하고 연결을 끊는데 필요한 시간의 측면에서 매우 비싼 자원이다. 인스턴스 풀링을 통해 컨넥션을 사용 후 버리고 다시 생성하는 것이 아니라 재사용 (reuse)함으로써 시스템의 자원을 절약할 수 있다. 클라이언트가 요청할 때 컨넥션이 이미 생성돼 있음으로 해서 인스턴스 풀링을 통한 컨넥션의 재사용은 성능면에서 경이적인 향상을 가져온다.

EJB에서의 인스턴스 풀링(Instance Pooling)

인스턴스 풀링은 EJB에서 새로 등장하는 개념은 아니다. 일반적인 풀링 기술로는 데이터베이스 컨넥션을 여러 객체들이 공유할 수 있도록 하는 데이터베이스 컨넥션 풀링 기술이 있다. EJB 서버는 서버측 컴포넌트에 이러한 인스턴스 풀링 기술을 적용함으로써 컴포넌트의 인스턴스 수를 줄여주고, 클라이언트에 대한 서비스에 필요한 자원도 줄게 된다.

EJB 클라이언트는 EJB 객체로 구현된 리모트 인터페이스와 상호 작용한다. 클라이언트 애플리케이션은 실제 빈에 직접적으로 액세스 하지 않는다. 대신에 그들은 빈 인스턴스를 둘러싸고 있는 EJB 객체와 상호작용한다. 인스턴스 풀링은 빈에 간접적으로 접속하도록 함으로써 수행력을 높여준다. 다시 말해 클라이언트는 빈에 직접적으로 접속하지 않기 때문에, 각각의 클라이언트에 대응하는 빈을 클라이언트 숫자만큼 유지할 필요가 전혀 없는 것이다. 서버는 훨씬 더 적은 수의 빈만 유지하면서 필요할 때마다 데이터를 넣었다, 꺼내왔다 하면 된다. 언뜻 생각하면 자원을 많이 소모시킬 것 같지만 잘만 구현하면 특정 시점에 소요되는 자원의 양을 현격히 줄여주는 역할을 한다.

인스턴스 풀링의 사용

엔티티 빈과 무상태 세션 빈은 인스턴트 풀링을 사용한다. 엔티티 빈과 무상태 세션 빈의 인스턴스 풀링에는 약간의 차이가 있으며 각각의 빈이 인스턴스 풀링을 어떻게 사용하는지 알아보겠다.

✍✍ 엔티티 빈의 인스턴스 풀링

인스턴스 풀링이 어떻게 작동되는지를 이해하기 위해 엔티티 빈의 라이프 사이클을 살펴보기로 한다. EJB는 엔티티빈의 라이프 사이클을 인스턴스 풀에 대한 관계로 정의한다. 엔티티 빈은 항상 아래 세 가지 중의 하나의 상태로 존재한다.

- 무상태

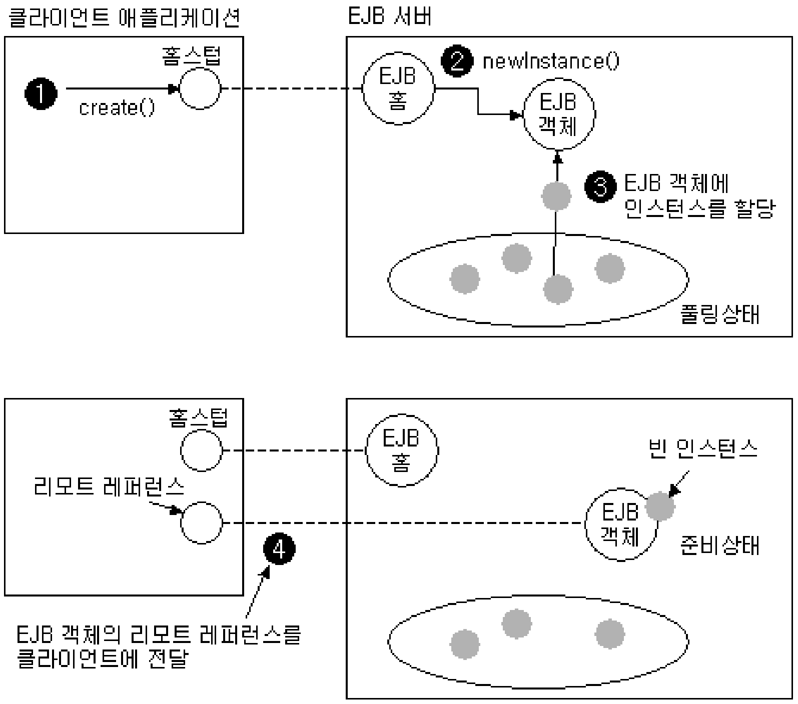
  빈 인스턴스가 아직 생성되지 않은 상태를 뜻한다.

- 풀링 상태

  컨테이너에 의해 인스턴스가 생성되었으나, 아직 EJB 객체와 연관지어지지 않은 상태이다.

- 준비상태

  빈 인스턴스가 EJB 객체와 연관을 맺고, 비즈니스 메소드 호출에 대응할 준비가 되어있는 상태이다.



<그림 : 인스턴스 풀로부터 준비상태로의 전이>

빈 인스턴스가 생성되어 풀에 들어가게 되면, 곧바로 컨테이너에 의해 제공되는 javax.ejb.EJBContext에 대한 레퍼런스가 주어진다. EJBContext는 빈이 EJB 환경과 통신하는데 사용할 수 있는 인터페이스를 제공한다.

클라이언트가 빈에 대한 리모트 인터페이스를 얻기 위해서 EJB 홈을 사용할 때, 컨테이너는 이에 대응하여 EJB 객체를 생성한다. 일단 EJB 객체가 생성되면, EJB 객체는 인스턴스 풀에서 생성되는 빈 인스턴스를 할당받는다. 그리고 EJB 객체에 할당되었을 때 빈 인스턴스는 공식적으로 준비 상태가 된다. 준비 상태가 되면 빈은 클라이언트로부터의 요청과 컨테이너의 처리 결과를 받을 수 있게 된다.

빈 인스턴스가 준비 상태가 되면, EntityContext는 새로운 정보를 얻게 된다. EntityContext는 해당 빈에게 현재 사용하고 있는 클라이언트에 대한 정보를 제공하게 된다. 또한 인스턴스가 자신의 EJB 홈과 EJB 객체에 접속할 수 있도록 해주는데, 이는 빈이 자신에 대한 레퍼런스를 다른 인스턴스들에게 전해줄 필요가 있거나 자신의 클래스 빈을 생성하거나 배치, 제거할 필요가 있을 때 유용하다.

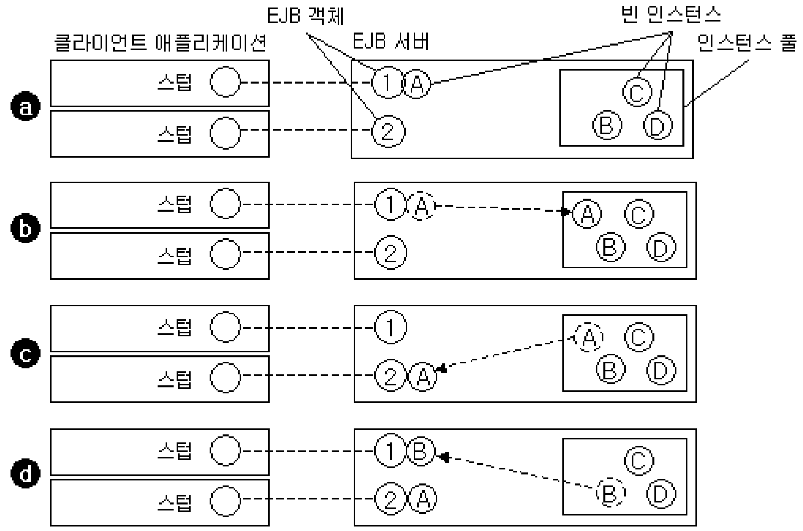
클라이언트가 빈의 리모트 레퍼런스를 마치게 되면, 리모트 레퍼런스가 사라지거나 빈의 제거(remove) 메소드 중의 하나가 호출된다. 빈이 제거되거나 사라지게 되면 빈의 인스턴스는 EJB 객체와 분리되어 인스턴스 풀로 돌아간다.

무상태 세션 빈의 인스턴스 스와핑

무상태 세션 빈은 인스턴스 풀링 기능을 왕성하게 사용한다. 무상태 세션 빈은 메

소드 호출 간에 어떤 상태도 유지하지 않는다. 무상태 세션 빈에 대하여 일어나는 모든 메소드 호출은 각기 독립적으로 일어나 어떤 인스턴스 변수도 참조하거나 이용하지 않고 자신의 역할을 수행한다. 이는 컨테이너가 클라이언트에게 호출하는 메소드들 간에 빈 인스턴스를 스와핑해주므로, 어떤 무상태 세션 인스턴스도 자신에게 맞는 타입의 EJB 객체의 요청에 서비스를 제공할 수 있다는 것을 의미한다.

다음의 그림은 메소드 호출 간의 인스턴스 스와핑을 보여주고 있다.



<그림 : 무상태 세션 빈의 인스턴스 스와핑>

ⓐ 인스턴스 A는 EJB 객체 1이 요청한 비즈니스 메소드 호출 서비스를 제공하고 있다.

ⓑ 일단 인스턴스 A가 요청된 서비스를 수행하고 나면, 인스턴스 풀로 되돌아간다.

ⓒ EJB객체 2에 대하여 비즈니스 메소드 호출이 일어나면, 인스턴스 A는 다시 EJB 객체 2에 대한 서비스를 제공한다.

ⓓ 만일 인스턴스 A가 EJB 객체 2에 대한 서비스를 제공하는 동안에 클라이언트로부터 EJB 객체 1에 대한 메소드 호출이 일어나면, 이는 인스턴스 B가 제공하게 된다.

이와 같은 스와핑 전략을 사용하면, 얼마 안되는 무상태 세션 빈으로도 수백 클라이언트에 대한 서비스를 제공할 수 있다. 이것이 가능한 것은 메소드가 수행되는데 소요되는 시간이 하나의 메소드가 호출된 후 다음 메소드가 호출될 때까지의 비활성화 시간들 보다 짧기 때문이다. 빈 인스턴스의 라이프 사이클 중에서 EJB 객체에 대한 서비스를 수행하고 있지 않는 기간은 비생산적이며, 인스턴스 풀링은 이러한 비생산적인 기간을 최소화시켜주는 역할을 하게 된다. 하나의 빈 인스턴스가 EJB 객체에 대한 서비스를 마치면 바로 그것을 필요로 하는 다른 EJB 객체에 서비스를 할 수 있게 되는 것이다. 이는 보다 적은 무상태 세션 인스턴스로 훨씬 많은 서비스를 수행할 수 있고, 결국 더 적은 자원으로 더 높은 성능을 낼 수 있음을 의미한다.

Activation Mechanism

무상태 세션 빈과는 달리 상태유지 세션 빈은 메소드 호출 간에 상태를 유지한다. 이는 상태유지 세션 빈이 클라이언트와의 지속적인 대화를 유지한다는 의미에서 '대화 상태(conversational state)'라고도 한다. 빈이 클라이언트에 서비스를 수행하는 동안에 이 대화 상태는 무결하게 유지될 필요가 있다. 상태유지 세션 빈은 무상태 세션 빈이나 엔티티 빈과 같은 방식으로 인스턴스 풀링이 이루어지지 않는다. 대신 상태유지 세션 빈에서는 자원을 유지하기 위해서 활성화(activation) 메커니즘이 사용된다.

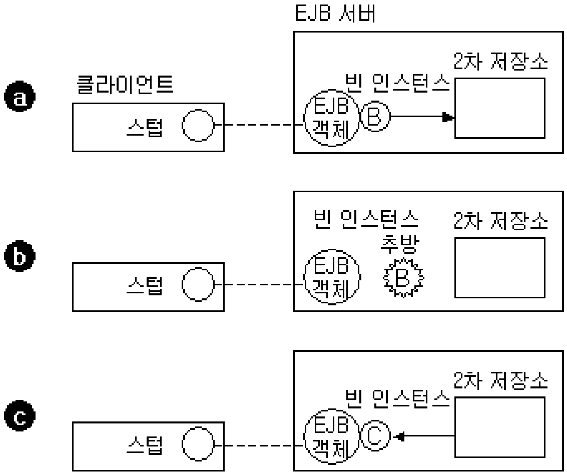
상태유지 세션 빈의 활성화 메커니즘

자원과의 어떤 작용을 할 필요가 있을 때, EJB 서버는 상태유지 세션 빈을 메모리로부터 꺼내 온다. 이러한 방법은 시스템이 유지해야 하는 인스턴스 수를 줄여준다. 빈을 활성화 시키고 대화 상태를 유지하기 위해서는 빈의 상태를 2차 저장소에 직렬화시켜 EJB 객체에 대응하도록 유지시킨다. 클라이언트가 EJB 객체의 메소드를 호출하면 새로운 상태유지 인스턴스가 생성되어 비활성화된 2차 저장소에 저장된다.

비활성화(passivate)는 상태유지 빈 인스턴스를 EJB 객체로부터 분리시키고, 그 상태를 저장하는 것이다. 비활성화를 위해서는 빈 인스턴스의 상태가 EJB 객체에 대응하여 체계적으로 저장될 필요가 있다. 일단 빈이 비활성화되면, 빈 인스턴스를 EJB 객체로부터 제거하고, 메모리에서 없애는 것이 안전하다. 클라이언트는 이러한 비활성화 과정을 전혀 알아채지 못한다. 클라이언트가 EJB 객체에 의해 구현된 리모트 인터페이스를 사용하므로, 빈 인스턴스와 직접적으로 통신하지 않는다는 점을 유의하자. 결과적으로 클라이언트와 EJB 객체의 연결은 빈이 비활성화된 후에도 유지될 수 있는 것이다.

빈을 활성화하는 것은 상태유지 세션 빈 인스턴스의 EJB 객체에 대한 상태를 회복하는 것이다. 비활성화된 EJB 객체에 대하여 메소드가 호출되면, 컨테이너는 자동적으로 새로운 인스턴스를 생성하고 비활성화 기간 동안 저장되어 있던 데이터와 똑같이 필드를 채운다. 그러고 나면 EJB 객체는 정상적으로 빈 메소드를 호출한다.

다음의 그림은 상태유지 빈의 활성화와 비활성화를 보여준다.



<그림 : 활성화 프로세스>

ⓐ 빈은 비활성화되고 있다. 인스턴스 B의 상태가 읽혀지고 서비스하고 있던 EJB 객체에 대응하여 저장된다.

ⓑ 빈은 비활성화되어 그 상태가 저장된다. 여기서 EJB 객체는 빈 인스턴스와 어떤 관계도 맺지 않게 된다.

ⓒ 빈은 다시 활성화된다. EJB 객체와 연결된 새로운 인스턴스 C가 생성되어 상태를 복원하는 과정을 거치게 된다 인스턴스 C는 EJB 객체에 대응하여 저장되어 있던 바로 그 상태를 유지하게 된다.

활성화 과정은 상태 관리 콜백 메소드에 의해 지원된다. 특히, ejbActivate() 메소드와 ejbPassivate() 메소드는 각기 상태유지 빈 인스턴스가 활성화될 것인지 비활성화될 것인지를 알려준다. ejbActivate() 메소드는 빈 인스턴스가 성공적으로 활성화된 다음에 바로 호출되며 ejbPassivate() 메소드는 빈 인스턴스가 비활성화되기 바로 전에 호출된다. 따라서 비활성화 이전에 어떤 조작 또는 격리가 필요하거나 활성화 이후에 재유지가 필요한 자원과의 관계를 빈 인스턴스가 유지해야 할 때 이들 두 메소드는 특히 유용하다.

기본 서비스

분산 애플리케이션을 작성할 때 이용할 수 있는 많은 부가 서비스들이 있다. CORBA의 주축이 되고 있는 OMG에서는 '서비스(services)'라고 불리는 13 가지의 부가서비스를 정의하였다. 여기서는 효과적인 EJB 서버를 만들기 위해 꼭 필요한 여섯 가지의 서비스, 즉 '기본 서비스(primary services)'라고 불리는 것들에 대하여 간단히 살펴본다.

동시성 제어

세션 빈은 동시접속을 지원하지 않는다. 반면 엔티티 빈은 데이터베이스의 데이터를 나타내므로, 공유와 동시접속이 필요하다. 따라서 엔티티 빈은 공유 컴포넌트이다. 엔티티 빈에 대한 동시접근이 가능해지기 위해서는 많은 클라이언트들이 동시에 빈에 접근할 수 있되 공유된 빈이 나타내고 있는 데이터는 보호될 필요가 있다.

EJB는 기본적으로 빈 인스턴스에 대한 동시접근을 막고 있다. 다시 말하면 여러 개의 클라이언트가 한 개의 EJB 객체와 연결할 수는 있지만, 한번에 하나의 클라이언트 쓰레드만이 빈 인스턴스에 연결할 수 있다. EJB는 이와 같은 방법으로 동시 접근과 관련하여 발생할 수 있는 위험성을 해결한다.

트랜잭션

트랜잭션은 함께 수행되어야 할 태스크들의 집합이다. 트랜잭션은 원자성(atomic)을 갖는다. 즉 하나의 트랜잭션이 성공하기 위해서는 해당 트랜잭션 내의 모든 태스크가 완료 되어야 한다. EJB 서버가 트랜잭션을 자동적으로 관리하므로, 빈 개발자는 트랜잭션과 관련된 빈을 관리하기 위해 별도의 API를 사용할 필요가 없다. 분산 배치할 때, 트랜잭션 속성을 주기만 하면 EJB 서버는 런타임 시에 빈을 어떻게 관리해야 할지를 자동적으로 안다. EJB는 빈이 트랜잭션을 명확하게 관리할 수 있도록 해주는 메커니즘을 제공한다.

퍼시스턴스

세션 빈과는 달리 엔티티 빈은 퍼시스턴스를 갖는다. 엔티티 빈의 상태는 데이터베이스 내에 영구적으로 저장되는 것이다. 엔티티 빈이 퍼시스턴스를 갖기 때문에 그들의 행위(behavior)와 데이터는 언제든지 접근이 가능하다.

빈의 상태가 퍼시스턴스 서비스로 자동 관리되므로 컨테이너는 엔티티 빈의 인스턴스 필드와 데이터베이스 내의 데이터를 일치시켜 줄 의무를 갖는다. 이 같은 자동 퍼시스턴스는 '컨테이너 관리 퍼시스턴스(container-managed persistence)'라 불린다. 이와 달리 빈이 자신의 상태를 스스로 관리하도록 만들어질 때는 '빈 관리 퍼시스턴스(bean-managed persistence)'라 부르며, 이는 주로 기존 시스템을 다룰 때 주로 사용된다. EJB 업체들이 퍼시스턴스를 구현하기 위해서 사용하는 가장 일반적인 메커니즘은 '객체-관계형 퍼시스턴스(object-to-relation persistence)'와 '객체 데이터베이스 퍼시스턴스(object database persistence)'이다.

분산객체

EJB는 자바 RMI를 이용하여 자신의 분산 인터페이스를 정의한다. 리모트 인터페이스와 홈 인터페이스의 수퍼타입인 javax.ejb.EJBObject와 javax.ejb.EJBHome 은 둘다 java.rmi.Remote를 상속받고 있다. 이들은 Remote 인터페이스의 서브타입으로서 Remote 인터페이스를 위해 자바 RMI 스펙을 따르도록 되어있다.

네이밍

분산 객체 서비스는 네이밍 서비스를 이용하고 있다. 모든 네이밍 서비스는 어떻게 구현되었건 간에 근본적으로 같은 일, 즉 클라이언트에 분산 객체의 위치를 알려 주는 메커니즘을 제공한다. 이를 위해 네이밍 서비스는 객체 바인딩(object binding)과 룩업(lookup) API를 제공해야 한다. 객체 바인딩은 분산 객체와 일반 언어의 네임 또는 식별자를 결합한 것이라 볼 수 있다. 이는 수백 개의 분산 객체를 관리해야 하는 환경에서 필수적인 요소이다. 룩업 API는 클라이언트에 네이밍 시스템과의 인터페이스를 제공한다. 간단히 말하면, 룩업 API는 클라이언트가 분산 서비스에 연결하여 특정 객체의 리모트 레퍼런스를 요청할 수 있도록 해준다. EJB는 자바 클라이언트들이 자바 네이밍 및 데렉토리 인터페이스(JNDI, Java Naming and Directory Interface)를 룩업 API로 사용할 수 있도록 한다.

보안

EJB 서버들은 세가지 정도의 보안 방법, 즉 '인증(authentication)', '접근 제어(access control)', '보안 통신(secure communication)'등을 지원한다.

✍✍ 인증

인증은 사용자의 정체를 확인하는 것을 뜻하며 인증 중에서 가장 흔한 것은 사용자 이름과 패스워드를 요구하는 간단한 로그인 스크린이다.

✍✍ 접근 제어

접근 제어는 특정 사용자가 시스템 내에서 할 수 있는 일과 할 수 없는 일을 규정하는 보안 정책이다.

✍✍ 보안 통신

클라이언트와 서버간의 통신 채널은 물리적인 차단이나 통신 암호화로 보안을 유지할 수 있다. 암호화 통신을 하게 되면 메시지가 암호화되기 때문에 인증을 받지 못한 사람은 읽을 수도, 데이터를 변경할 수도 없다. 이 방법에서는 대개 클라이언트와 서버 간에 암호화 키를 주고 받는다. 메시지 수신자는 키를 이용하여 메시지를 복호화하여 읽게 된다