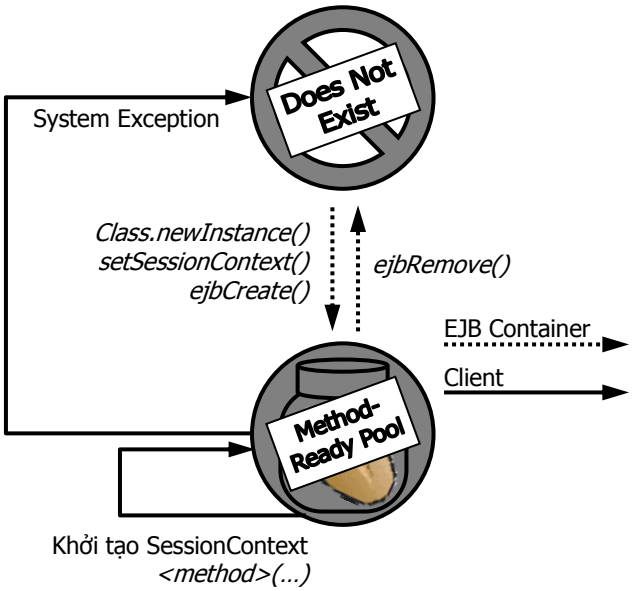
Bài 3: Stateless Session Bean

I. Khái niệm

1. Stateless Session Bean

- Trạng thái giao dịch liên kết với một client, gọi là trạng thái đặc trưng cho client (client-specific state). Trạng thái này gọi là conversation state, thực tế còn bao gồm cả các kết nối socket, kết nối database, tham chiếu đến bean khác...
- Stateless Session Bean không lưu giữ trạng thái converation Container phân biệt Stateless Session Bean với Stateful Session Bean nhờ những mô tả trong tập tin deployment descriptor **`ejb-jar.xml`**.
- Stateless Session Bean được thiết kế để phục vụ cho **nhiều** client, với những session chỉ có một request.
- Stateless Session Bean chỉ có 2 trạng thái:
    - Does Not Exist: không có thực thể bean tồn tại trong bộ nhớ.
    - Method-Ready Pool: thực thể bean có mặt trong pool, sẵn sàng phục vụ các triệu gọi business method từ client.
- Stateless Session Bean yêu cầu ít resource nên còn gọi là *lightweight bean*.



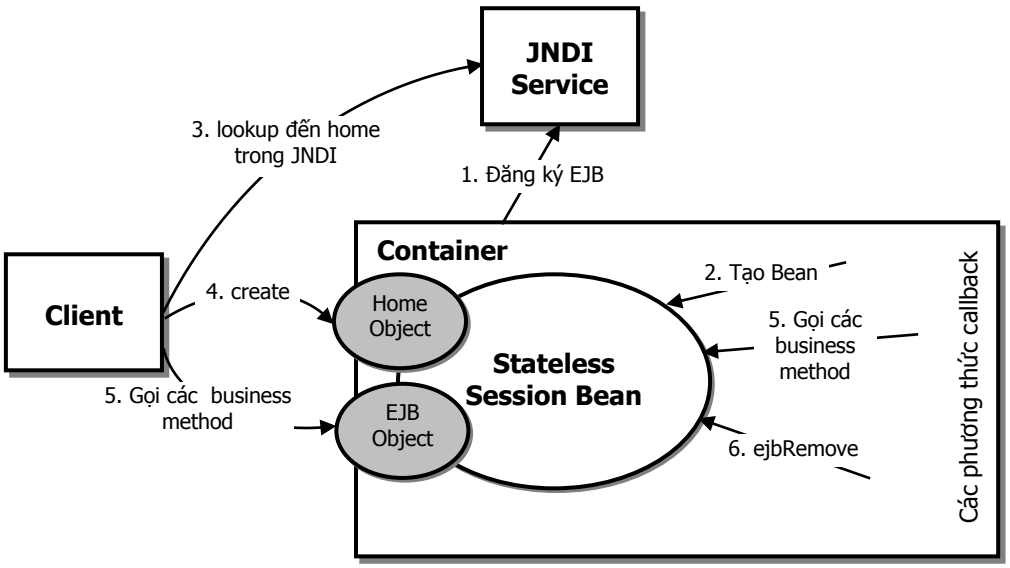
2. Vòng đời của Stateless Session Bean

- EJB Container quản lý các thực thể bean. Khi cần thực thể, thực thể bean được khởi tạo bằng cách dùng phương thức **`Class.newInstance()`** trên lớp bean. Sau đó phương thức **`setSessionContext( sc )`** của thực thể bean được triệu gọi để thực thể bean nhận tham chiếu đến đối tượng SessionContext. Thực thể bean cần SessionContext để truy xuất thông tin môi trường: identity và role của client, transaction context,… Cuối cùng phương thức **`ejbCreate()`** không đối số của bean được EJB Container triệu gọi (chỉ một lần trong vòng đời của bean). Lúc này bean đã ở trạng thái sẵn sàng thực hiện các yêu cầu của client do EJB(Local) Object ủy nhiệm (delegate) đến, gọi là trạng thái Method-Ready Pool.
- Do không cần lưu trữ trạng thái giao dịch, Stateless Session Bean không quan tâm đến việc kích hoạt (activation) bean, container không bao giờ gọi đến các phương thức callback **`ejbActivate()`** và **`ejbPassivate()`** của nó. Ra khỏi trạng thái Method-Ready Pool, bean sẽ rơi vào trạng thái Does Not Exits nghĩa là container không cần nó nữa. Khả năng lưu trữ của pool là cơ sở để container thu giảm số bean ở trạng thái Method-Ready Pool bằng phương thức **`ejbRemove()`**, giải phóng tài nguyên và hủy bean. Xem phần Instance swapping phía dưới để hiểu rõ cơ chế.

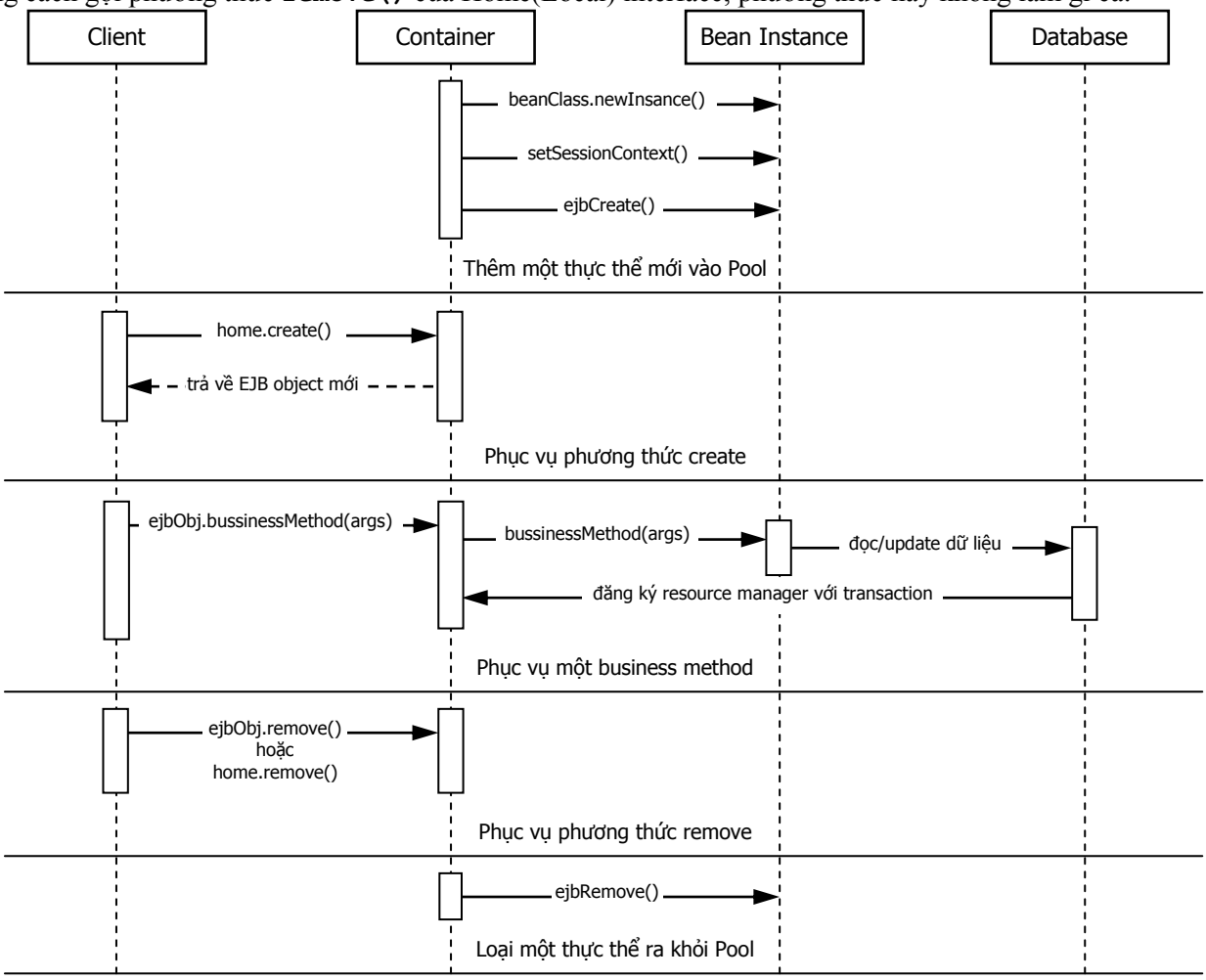
3. Các phương thức callback

- Phương thức **`void ejbCreate()`** chỉ có duy nhất một và không tham số vì không cần khởi tạo trạng thái conversation. Session bean phải có ít nhất một phương thức **`ejbCreate()`**. Thường rỗng do client không triệu gọi đến.
- Phương thức **`void ejbRemove()`** thường rỗng do client không triệu gọi đến.
- Phương thức **`void ejbPassivate()`** rỗng vì container không bao giờ thụ động hóa thực thể bean.
- Phương thức **`void ejbActivate()`** rỗng vì container không bao giờ kích hoạt thực thể bean.
- Phương thức **`void setSessionContext( SessionContext ctx )`** thường rỗng vì bean không có biến thực thể (biến lớp riêng) nên không cần lưu các tham chiếu đến các biến đó.

4. Hoạt động của Stateless Session Bean



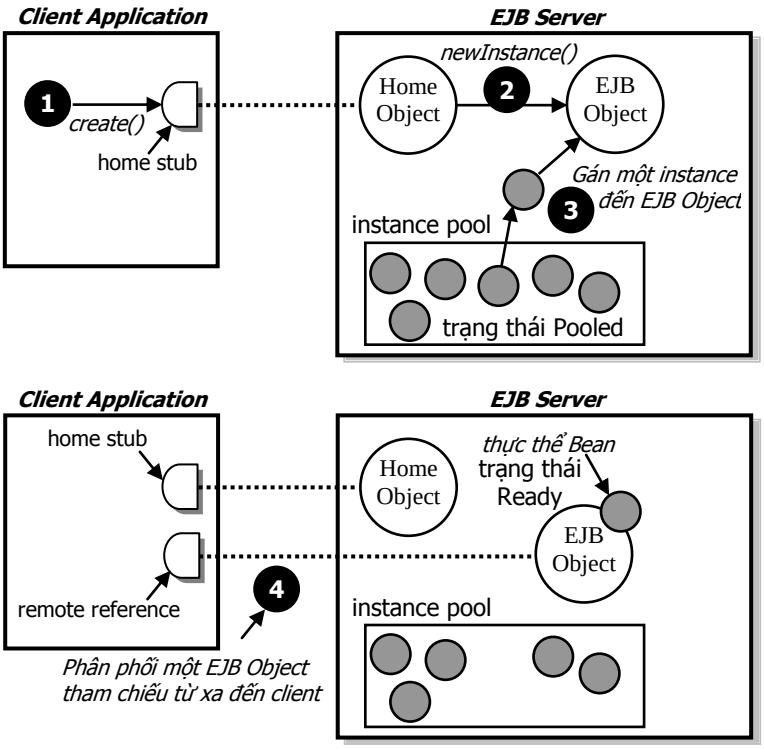
1- Container đăng ký tất cả bean đã triển khai với JNDI với tên JNDI chỉ định trong deployment descriptor.
2- Container chịu trách nhiệm quyết định tạo thực thể bean tùy theo chính sách lưu (caching). Container lần lượt gọi các phương thức **`Class.newInstance()`**, **`setSessionContext()`** và **`ejbCreate()`**.
3- Client tìm thấy Home(Local) interface của bean bằng cách **`lookup()`** thông qua JNDI.
4- Client dùng phương thức **`create()`** của Home(Local) interface để tạo ra EJB(Local) Object của bean. Chú ý phương thức này *không* triệu gọi **`ejbCreate()`** trên bean.
5- Client gọi các business method thông qua EJB(Local) Object, lời gọi này sẽ được container ủy nhiệm đến bean.
6- Container chịu trách nhiệm quyết định loại bỏ thực thể bean bằng cách gọi **`ejbRemove()`**. Không loại bỏ thực thể bean bằng cách gọi phương thức **`remove()`** của Home(Local) interface, phương thức này không làm gì cả.



5. Quản lý thực thể với Stateless Session Bean

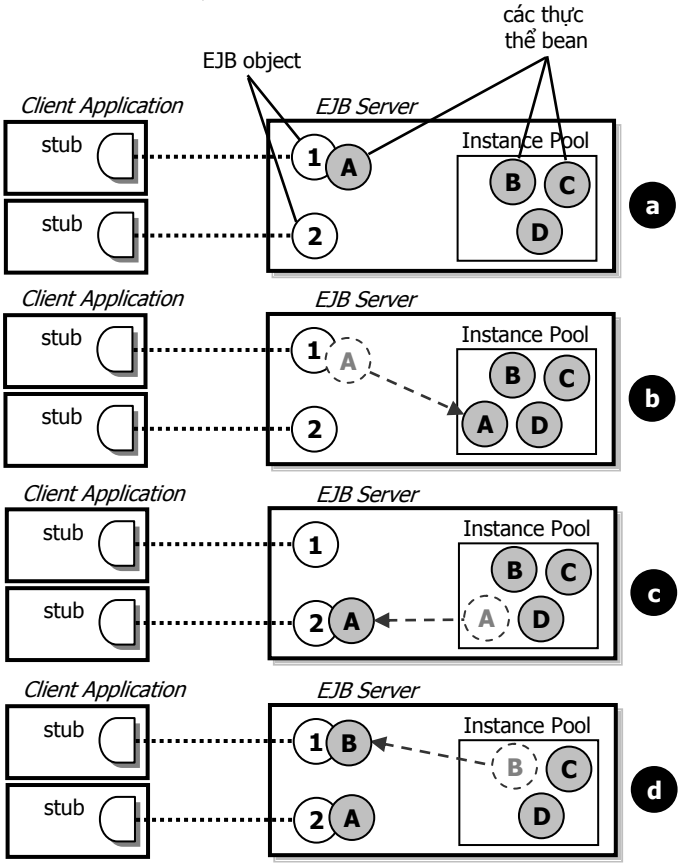
a) Instance Pool

- Các thực thể bean sẽ được tạo khi container hoạt động. Instance pooling là cơ chế để thu giảm số thực thể bean cần có mặt cho việc phục vụ các yêu cầu của client, giảm chi phí tài nguyên khi tạo và hủy bean bằng cách dùng lại các thực thể có trong pool. Như vậy thu giảm việc sử dụng tài nguyên.
- Các instance pool đều cố gắng quản lý các tập hợp bean sao cho có thể truy xuất nhanh các bean trong thời gian chạy. Để thiết lập instance pool, container tạo một số thực thể của lớp bean và giữ chúng cho đến khi cần đến. Khi client yêu cầu một business method, thực thể bean từ pool được gán đến EJB(Local) Object liên kết với client. Khi một EJB(Local) Object không cần đến thực thể, bean được trả về cho instance pool.
- Một EJB server duy trì nhiều instance pool cho mỗi kiểu pool được triển khai. Mọi thực thể trong instance pool là tương đương với nhau. Thực thể bean sẽ được chọn tùy ý để gán đến EJB(Local) Object cần đến nó. Số thực thể trong bean dao động do hoạt động vào/ra của bean.
- Sau khi thực thể bean được đặt trong pool, gọi là trạng thái Pooled, nó lấy tham chiếu đến một **`javax.ejb.EJBContext`**. **`EJBContext`** cung cấp một giao diện để bean có thể dùng liên lạc với môi trường EJB.



- Khi bean liên kết với một EJB(Local) Object, nó ở trạng thái Ready. Lúc này **`EJBContext`** có một ý nghĩa mới, nó cung cấp thông tin về client dùng bean, cần khi bean truyền tham chiếu đến chính bean hoặc các bean khác.

b) Instance swapping (hoán chuyển thực thể bean)

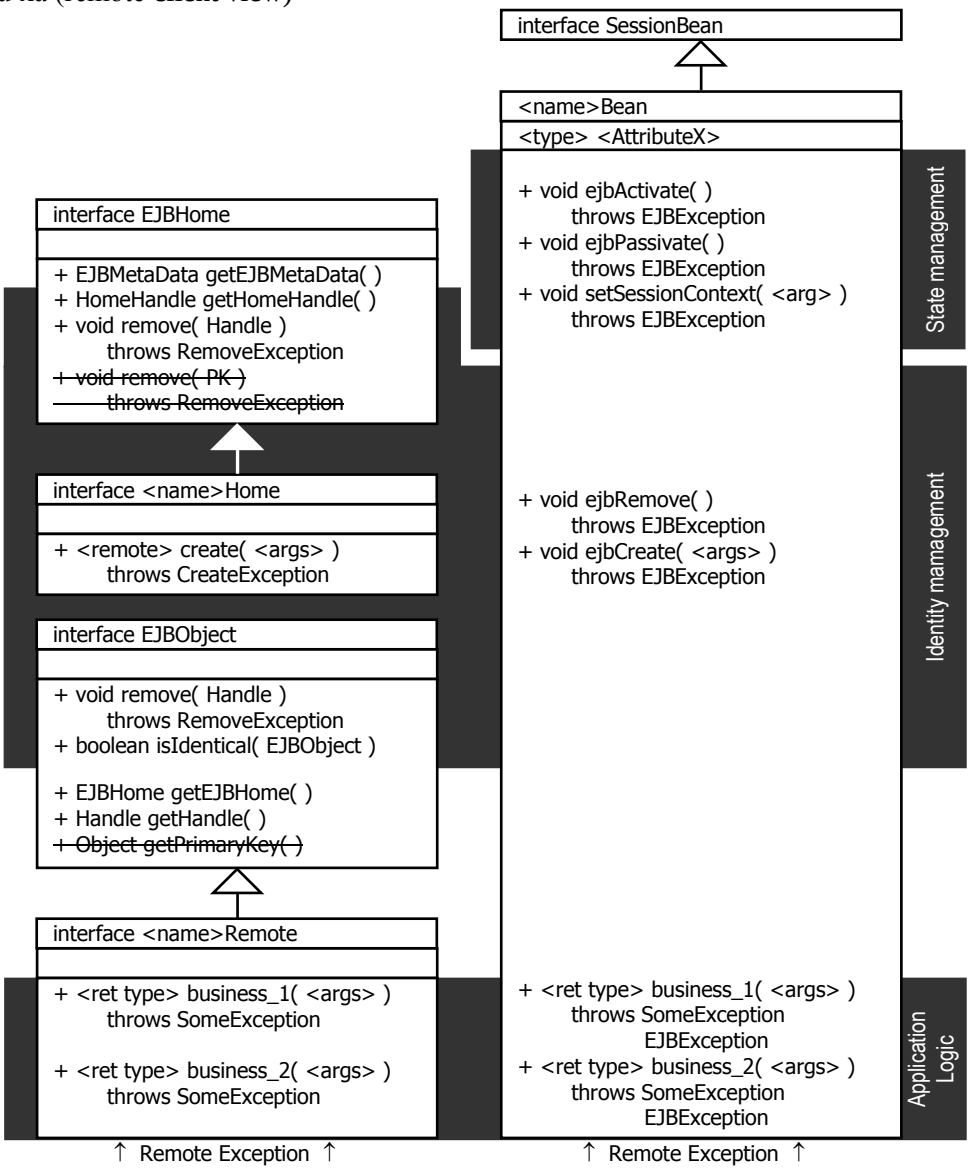


- Sau khi phục vụ yêu cầu của client, bean tách rời EJB Object và quay trở về instance pool. Nếu client yêu cầu đến EJB Object mà không có thực thể bean liên kết với nó, container sẽ tìm bean trong pool và lại gán cho EJB Object. Thao tác này gọi là instance swapping.
- Stateless Session Bean không cần lưu trữ trạng thái conversation trong nó nên mọi triệu gọi phương thức từ nó tiến hành độc lập. Điều này có nghĩa bất kỳ Stateless Session Bean nào cũng có thể phục vụ yêu cầu cho bất kỳ EJB Object nào có kiểu ủy nhiệm hợp lệ đến nó. Container có thể hoán chuyển các thực thể bean vào hoặc ra giữa các triệu gọi phương thức.
- Hình trên mô tả cơ chế hoán chuyển thực thể khi triệu gọi phương thức của Stateless Session Bean: Bean A phục vụ một triệu gọi business method được ủy nhiệm từ EJB Object 1 (a). Khi bean A thực hiện xong yêu cầu nó chuyển ngược trở về instance pool (b). Khi client triệu gọi phương thức trên EJB Object 2, bean A lại liên kết với EJB Object 2 để phục vụ yêu cầu này (c). Khi bean A đang phục vụ, nếu có một triệu gọi trên EJB Object 1, bean B sẽ phục vụ yêu cầu này.
- Dùng chiến lược hoán chuyển trên, vài bean có thể phục vụ cho hàng trăm client. Khi một thực thể bean chấm dứt phục vụ, nó lập tức trở về trạng thái sẵn sàng trong instance pool cho một EJB Object bất kỳ cần ủy nhiệm đến nó.

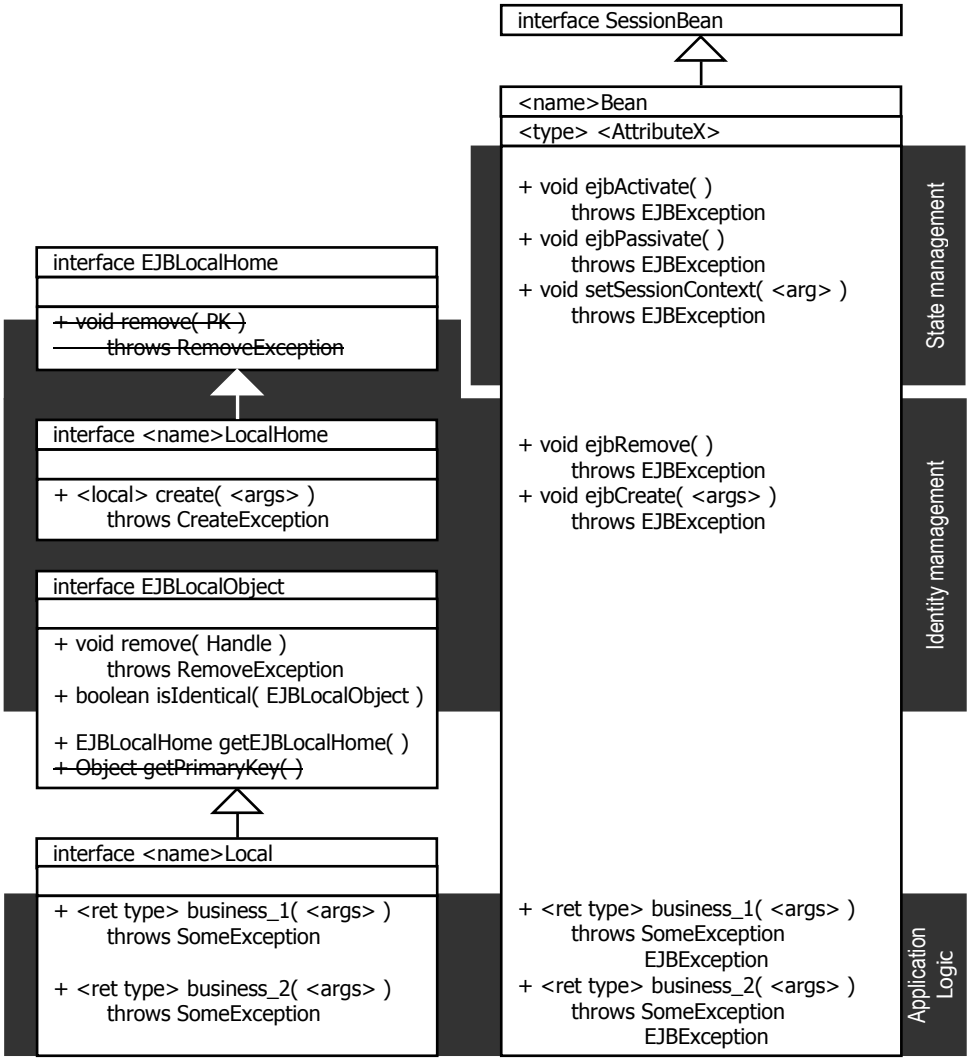
6. Lớp và giao diện
- Hàm dựng (constructor) nói chung không dùng khi tạo lớp bean, nếu có phải là hàm dựng mặc định (không đối số) và **`public`**. Trong lớp bean không có các phương thức **`finalize`**.

+ **`void ejbActivate()`**

a) Client truy xuất từ xa (remote client view)



b) Client truy xuất cục bộ (local)



II. Thiết kế và triển khai

A. Tạo Stateless Session Bean

- Cần có gói **`javax.ejb`** trong **`CLASSPATH`**: nghĩa là cần có **`ejb-2_1-api.jar`**, hoặc **`j2ee.jar`** (nếu có cài đặt j2eesdk), hoặc **`jboss-j2ee.jar`** (**`%JBOSS_HOME%\client`**) trong **`CLASSPATH`**. Cần chú ý đến phiên bản của EJB đang triển khai.

1. Lớp EJB

- Lớp EJB cài đặt:
  - Các phương thức quan trọng nhất của EJB là các business method (phương thức nghiệp vụ). EJB được thiết kế nhằm mục tiêu giúp người lập trình chỉ quan tâm đến việc thực hiện các business method. Các business method này *cùng tên* với phương thức khai báo trong EJB interface để có thể delegate từ đó được, nhưng không cần ném **`RemoteException`** vì được delegate, mà không triệu gọi từ xa. Các phương thức này không được bắt đầu bằng **`ejb`**, không được **`static`** hoặc **`final`** và phải **`public`**.
  - Các phương thức bắt buộc cài đặt, dành cho container dùng quản lý EJB, gọi là các phương thức callback. Các phương thức này thừa kế từ giao diện **`javax.ejb.SessionBean`**.

Lớp EJB không chứa các tác vụ cấp hệ thống (persistence, security, transaction, …) nên dễ triển khai hơn RMI nhiều.

- Lớp EJB cần phải:
  - import các giao diện: **`javax.ejb.SessionBean`**, **`javax.ejb.SessionContext`**.
  - thừa kế giao diện **`javax.ejb.SessionBean`**.
  - Nếu có constructor phải là constructor mặc định (không đối số) và không có phương thức **`finalize`**.

```
package myejb.session;

import javax.ejb.*;
```

```
import javax.naming.*;

public class ConvertBean implements SessionBean {
  private SessionContext ctx;
  private Context environment;

  // Callback methods
  public void setSessionContext( SessionContext sc ) {
    this.ctx = sc;
  }

  public void ejbCreate() throws CreateException {
    try {
      InitialContext ic = new InitialContext();
      environment = ( Context )ic.lookup( "java:comp/env" );
    } catch ( NamingException ne ) {
      throw new CreateException( "Could not look up context" );
    }
  }

  public void ejbRemove() { }
  public void ejbActivate() { }
  public void ejbPassivate() { }

  // Business methods
  public double dollarToVND( double dollars ) {
    Double rate = new Double( 0 );
    try {
      rate = ( Double )environment.lookup( "ExchangeRate" );
    } catch ( NamingException ne ) {
      System.out.println( "Invalid Exchange Rate" );
    }
    return dollars * rate.doubleValue();
  }
}
```

tham chiếu đến ENC
java:comp/env

tham chiếu đến thuộc tính môi trường
java:comp/env/ExchangeRate

- Passivation và Activation chỉ thực hiện trên Stateful Session Bean, không thực hiện trên Stateless Session Bean nhưng có trong giao diện **javax.ejb.SessionBean**.

Business logic của bean trên là phương thức **dollarToVND()** chuyển đổi tiền từ USD sang VND. Tỷ giá hối đoái (exchange rate) được lưu và tham chiếu như một thuộc tính ENC trong tập tin DD **ejb-jar.xml**.

2. Các giao diện

- Tùy kịch bản sử dụng, một trong hai hoặc cả hai loại giao diện sau sẽ được triển khai. Có thể dùng XDoclet[1] để sinh các giao diện này một cách tự động.

a) Giao diện truy xuất từ xa (outside view, remote view)

- Cặp giao diện truy xuất từ xa dùng khi có triệu gọi các business method của EJB từ xa, ví dụ từ ứng dụng client độc lập, từ Web tier thuộc Application server khác. Gồm: Remote interface và Remote Home interface.
- Remote interface khai báo tất cả các business method của EJB mà client sẽ *triệu gọi từ xa*, vì vậy có ném **RemoteException** khi lời triệu gọi thất bại. Các business method khai báo ở đây sẽ được cài đặt trong lớp EJB và sẽ được Remote interface ủy nhiệm đến.

Remote interface là một giao diện "bộc lộ" các business method của EJB liên kết với nó cho phía client triệu gọi.

Remote interface cần phải:

- import các giao diện **javax.ejb.EJBObject** và **java.rmi.RemoteException**.
- Remote interface phải **public** (để có thể triệu gọi từ xa) và thừa kế giao diện **javax.ejb.EJBObject**.
- Các business method được định nghĩa ở đây có khả năng ném ra **java.rmi.RemoteException** khi gặp sự cố (mạng, server, …)

```
package myejb.session;
```

[1] Để giảm mức độ phức tạp của tài liệu, XDoclet được giới thiệu riêng trong phần phụ lục. Ngoài ra, phiên bản dùng XDoclet cho mỗi ví dụ có kèm theo CD.

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

public interface Convert extends EJBObject {
  public double dollarToVND( double dollars ) throws RemoteException;
}
```

- Home interface định nghĩa các phương thức quản lý vòng đời của bean, hoạt động như một factory[2]. Home interface được tạo ra cùng cặp với giao diện triệu gọi tương ứng. Với Remote interface, ta có Remote Home interface.

Giao diện này cần phải:

- import các giao diện: **`java.rmi.RemoteException`**, **`javax.ejb.CreateException`** và **`javax.ejb.EJBHome`**.
- Remote Home interface phải thừa kế giao diện **`javax.ejb.EJBHome`**.
- Có phương thức **`create()`** ném ra các exception: **`RemoteException`** và **`CreateException`** (cùng với các exception của người dùng nếu có) và trả về một EJB Remote Object có kiểu *Remote interface*.

```
package myejb.session;

import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface ConvertHome extends EJBHome {
  Convert create() throws java.rmi.RemoteException,CreateException;
}
```

Phương thức **`create()`** của Stateless Session Bean không nhận bất kỳ một đối số nào vì không cần lưu trữ trạng thái conversation giữa client và server.

b) Giao diện truy xuất cục bộ (inside view, local view)

- Giao diện truy xuất cục bộ tương tự giao diện truy xuất từ xa nhưng dùng khi triệu gọi EJB cục bộ từ một EJB khác hoặc từ JSP, servlet thuộc Web tier trong cùng ứng dụng J2EE. Giao diện truy xuất cục bộ không dựa trên RMI và thường được dùng trong một JVM. Gồm: Local interface và Local Home interface.

- Local interface cần phải:

- import giao diện **`javax.ejb.EJBLocalObject`**.
- Local interface phải **`public`** và thừa kế giao diện **`javax.ejb.EJBLocalObject`**.
- Các business method được định nghĩa ở đây không ném ra **`java.rmi.RemoteException`** khi gặp sự cố, vì được triệu gọi cục bộ.

```
package myejb.session;

import javax.ejb.EJBLocalObject;

public interface ConvertLocal extends EJBLocalObject {
  public double dollarToVND( double dollars );
}
```

- Local Home interface tương tự Remote Home interface, nhưng dùng chung cặp với Local interface, phương thức **`create()`** của nó được gọi cục bộ nên *không* ném ra RemoteException.

Local Home interface cần phải:

- import các giao diện: **`javax.ejb.CreateException`** và **`javax.ejb.EJBLocalHome`**.
- Local Home interface phải thừa kế giao diện **`javax.ejb.EJBLocalHome`**.
- Có phương thức **`create()`** ném ra exception **`CreateException`** cùng với các exception của người dùng nếu có và trả về một EJB Local Object có kiểu *Local interface*.

```
package myejb.session;

import javax.ejb.CreateException;
import javax.ejb.EJBLocalHome;

public interface ConvertLocalHome extends EJBLocalHome {
```

[2] Tham khảo mô hình factory object trong RMI.

```
  ConvertLocal create() throws CreateException;
}
```

3. Deployment Descriptor (DD)

- Chuẩn đóng gói các component của J2EE quy định EJB được đóng vào gói **.JAR**, kèm theo tập tin mô tả triển khai (Deployment Descriptor) **ejb-jar.xml**. Ta dùng Ant để thực hiện việc đóng gói.
- JBoss cung cấp một XML Editor là **ejx.jar** (chỉ dùng cho EJB1.1) nhưng từ sau phiên bản 2.1 không thấy gói này nữa. Có thể dùng tiện ích soạn thảo bất kỳ để tạo tập tin XML trên. Nếu dùng XDoclet, các tập tin mô tả cũng được sinh ra tự động như các giao diện.
- Trong DD, người phát triển bean cần cung cấp các thông tin mà EJB container không thể nhận được bằng cách "nội soi" (introspection) các lớp.

```
<?xml version="1.0"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
  <enterprise-beans>
    <session>
      <ejb-name>Convert</ejb-name>
      <home>myejb.session.ConvertHome</home>
      <remote>myejb.session.Convert</remote>
      <local-home>myejb.session.ConvertLocalHome</local-home>
      <local>myejb.session.ConvertLocal</local>
      <ejb-class>myejb.session.ConvertBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>

      <env-entry>                                          ENC
        <env-entry-name>ExchangeRate</env-entry-name>
        <env-entry-type>java.lang.Double</env-entry-type>
        <env-entry-value>15740.0</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>

  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>Convert</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Supports</trans-attribute>
    </container-transaction>

    <security-role>
      <description>Users</description>
      <role-name>users</role-name>
    </security-role>
  </assembly-descriptor>
</ejb-jar>
```

Với ví dụ trên, chỉ cần element **<enterprise-beans>**. Các element khác chỉ minh họa thêm phần transaction và security, sẽ thảo luận sau.

- Mỗi component của ứng dụng J2EE có một JNDI ENC riêng (Enterprise Naming Context), cung cấp nơi đăng ký các đối tượng phân tán, thường dùng khi truy xuất cục bộ. Trong đó, **java:comp** là đặc tả component (component – specific) riêng của JNDI ENC. Một số thông tin được khai báo trong các DD chuẩn và được truy xuất thông qua JNDI:
  - Các mục nhập (thuộc tính) môi trường (environment entry), khai báo trong các element **<env-entry>**. Xem ví dụ khai báo thuộc tính ENC **ExchangeRate** trong DD **ejb-jar.xml** trên.
  - Các tham chiếu đến EJB, khai báo trong các element **<ejb-ref>** và **<ejb-local-ref>**.

- Các tham chiếu đến nguồn kết nối dữ liệu (resource manager connection factory), khai báo trong các element **<resource-ref>.**
- Các tham chiếu tài nguyên môi trường (resource environment), khai báo trong các element **<resource-env-ref>.**

B. Tạo client truy xuất Stateless Session Bean

1. Client là ứng dụng độc lập
- Sau khi triển khai, Home Object của EJB được đăng ký trong cây JNDI. Để truy xuất bean ta lần lượt thực hiện:

a) Định vị Home Object
- Client dùng tên đăng ký JNDI để định vị Home Object.
  - Thiết lập thuộc tính môi trường (Initial Context Factory, vị trí server cung cấp dịch vụ Naming, …) cho JVM của client. Các cách thiết lập đã được giới thiệu trong chương 1. Trong ví dụ minh họa này, ta tạo tập tin **jndi.properties** có nội dung:

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=venus:1099
```

venus là hostname (cũng có thể dùng địa chỉ IP, địa chỉ vu hồi **localhost** – **127.0.0.1**, …) của máy chủ chạy JBoss. Port 1099 là mặc định cho JNDI.

  - Tạo JNDI naming context như một giao diện giữa client và JNDI.

```
Context ctx = new InitialContext();
```

b) Tìm (lookup) đối tượng thông qua JNDI
- Sau khi tạo JNDI context, phương thức **lookup()** của đối tượng lớp **javax.naming.InitialContext** được sử dụng để định vị đối tượng có tên JNDI chỉ định. Phương thức này trả về một đối tượng (kiểu **Object** chung).

```
Object obj = ctx.lookup( "Convert" );
```

c) Thu hẹp (narrow) tham chiếu
- Đối tượng trả về bởi phương thức **lookup()** trong trường hợp client truy xuất từ xa có thể là đối tượng CORBA hoặc Java. Để tổng quát hơn, nên thu hẹp tham chiếu cho phép ép kiểu một cách tường minh đối tượng nhận được thành kiểu home interface. Điều này được thực hiện bởi phương thức **PortableRemoteObject.narrow()**, có hai tham số: đối tượng do **lookup()** trả về và tên tập tin class của home interface. Lớp **javax.rmi.PortableRemoteObject** được dùng thay cho lớp **java.rmi.server.UnicastRemoteObject** trong RMI để bảo đảm tính tương thích với các giao thức khác **JRMP**, ví dụ **RMI/IIOP.**

```
ConvertHome home = ( ConvertHome )PortableRemoteObject.narrow( obj, ConvertHome.class );
```

- Trong trường hợp client truy xuất cục bộ, không cần đến phương thức này.

d) Sinh ra một thực thể EJB
- EJB sẽ triệu gọi phương thức **create()** của đối tượng remote home để trả về EJB Object (đối tượng remote interface).

```
Convert convert = home.create();
```

e) Triệu gọi các business method
- EJB Interface định nghĩa các business method thực hiện trong lớp EJB, client sẽ triệu gọi các phương thức này thông qua tham chiếu đến EJB Object do phương thức **create()** trả về. Khi EJB Object nhận lời triệu gọi, nó ủy nhiệm (delegate) cho bean bên trong liên quan thực hiện.

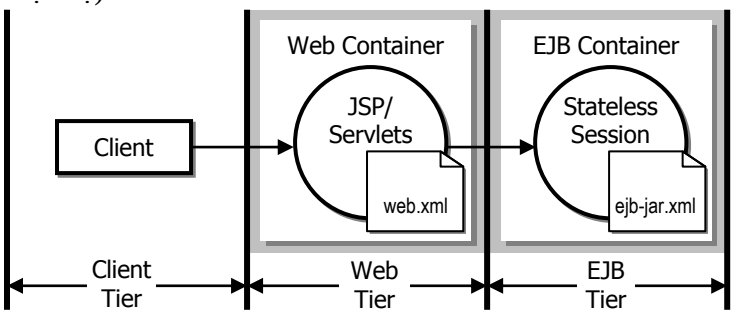
```
vnd = convert.dollarToVND( amt );
```

f) Client truy xuất từ xa
- Chương trình client sau có giao diện Swing, các thao tác chủ yếu được thực hiện trong phương thức **actionperformed()** của lớp **ButtonListener.** Chú ý tên JNDI của bean khi truy xuất từ xa.

```
package clients;
```

```
import myejb.session.Convert;
import myejb.session.ConvertHome;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

public class ConvertClient extends JFrame {
  public static String value;
  public static double vnd;
  public static double amt;

  Container c;
  JTextField t = new JTextField( 10 );
  JButton b = new JButton( "Convert" );
  JLabel result = new JLabel();

  public ConvertClient( String s ) {
    super( s );
    c = getContentPane();
    c.setLayout( new GridLayout( 2, 2, 2, 2 ) );
    c.add( new JLabel( "Enter the amount in USD:" ) );
    c.add( t );
    c.add( b );
    c.add( result );
    value = t.getText();
    b.addActionListener( new ButtonListener() );
    setSize( 400, 95 );
    setVisible( true );
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

class ButtonListener implements ActionListener {
  public void actionPerformed( ActionEvent ev ) {
    value = t.getText();
    amt = Double.parseDouble( value );
    try {
      Context ctx = new InitialContext();
      Object obj = ctx.lookup( "Convert" );
      ConvertHome home = (ConvertHome)PortableRemoteObject.narrow(obj,ConvertHome.class);
      Convert convert = home.create();
      vnd = convert.dollarToVND( amt );
      convert.remove();
      if ( vnd >= 1E6 )
        result.setText( String.valueOf( vnd/1000000 ) + " million(s) VND" );
      else
        result.setText( String.valueOf( vnd ) + " VND" );

      ctx.close();
    } catch ( Exception e ) {
      e.printStackTrace();
    }
  }
}

  public static void main( String[] args ) {
    new ConvertClient( "Convert Tool" );
  }
}
```

2. Client là Web tier (truy xuất cục bộ)



- Servlet **`ConvertSevlet.java`** là client truy xuất EJB cục bộ, gồm hai phần:
  - **`doGet()`**: dùng để hiển thị form ban đầu vì lúc đầu gọi trang bằng GET; hoặc dùng để hiển thị form phía trên kết quả, do **`doPost()`** gọi **`doGet()`** trước tiên.
  - **`doPost()`**: dùng để hiển thị kết quả vì lúc này gọi trang bằng POST thông qua form. Kết quả có được từ việc triệu gọi cục bộ EJB.

- Khi triệu gọi EJB thông qua giao diện truy xuất cục bộ, không cần thiết lập các thuộc tính môi trường liên quan JNDI vì client đã lấy thông tin môi trường từ Container, ta chỉ sử dụng một thực thể **`InitialContext`** mặc định để lookup Local Home Object từ ENC hoặc tên JNDI. Cũng không cần thu hẹp tham chiếu nhận được bằng phương thức **`narrow()`** của lớp **`PortableRemoteObject`**.

```
import myejb.session.*;
import javax.naming.*;
import java.io.PrintWriter;
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.ejb.CreateException;

public class ConvertServlet extends javax.servlet.http.HttpServlet {
  private ConvertLocalHome home = null;

  public void init( ServletConfig conf ) throws ServletException {
    super.init( conf );
    try {
      Context ctx = new InitialContext();
      if ( home == null )
        home = ( ConvertLocalHome )ctx.lookup( "java:comp/env/Convert" );
    } catch ( NamingException e ) {
      e.printStackTrace ();
    }
  }

  public void doGet( HttpServletRequest req, HttpServletResponse res )
     throws ServletException ,IOException {
    res.setContentType( "text/html" );
    PrintWriter out = ( PrintWriter )res.getWriter();
    out.println( "<html><head><title>Convert</title>" );
    out.println( "<style>input.std { width:200; }" );
    out.println( "div.frame { font-family:verdana; font-size: 9pt;}" );
    out.println( "</style></head><body><div class=\"frame\">");
    out.println( "<form action=\"ConvertServlet\" method=\"post\">" );
    out.print( "Enter the amount in USD: " );
    out.println( "<input type=\"text\" name=\"amt\" class=\"std\" /><br />" );
    out.println( "<input type=\"submit\" value=\"Convert\" /></form></div>" );
    out.println( "</body></html>" );
  }

  public void doPost( HttpServletRequest req, HttpServletResponse res )
     throws ServletException, IOException {
```

```
    double vnd = 0.0;
    doGet(req,res);
    PrintWriter out = ( PrintWriter )res.getWriter();
    try {
      ConvertLocal convert = home.create();
      vnd = convert.dollarToVND( Double.parseDouble( req.getParameter( "amt" ) ) );
      convert.remove();
    } catch ( Exception ex ) {
      out.println( "<BR>Error: " + ex.toString() );
    }
    String result = "";
    if ( vnd >= 1E6 )
      result = String.valueOf( vnd/1000000 ) + " million(s) VND";
    else
      result = String.valueOf( vnd ) + " VND";
    out.println( "<html><head><style>" );
    out.println( "p { font-family:verdana;font-size:9pt; }</style></head>" );
    out.println( "<body><p>Convert result:<br />" );
    out.println( req.getParameter( "amt" ) + "USD = <b>" + result + ".</b></p>" );
    out.println( "</body></html>" );
  }
}
```

- Module Web cần DD **web.xml** cho gói **.WAR** của ứng dụng Web. Nếu dùng XDoclet, DD này cũng được sinh ra một cách tự động.

```
<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <display-name>Convert</display-name>
  <welcome-file-list>
    <welcome-file>ConvertServlet</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>ConvertServlet</servlet-name>
    <servlet-class>ConvertServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ConvertServlet</servlet-name>
    <url-pattern>/ConvertServlet</url-pattern>
  </servlet-mapping>

  <ejb-local-ref>                                          ENC
    <ejb-ref-name>Convert</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>myejb.session.ConvertLocalHome</local-home>
    <local>myejb.session.ConvertLocal</local>
    <ejb-link>convert.jar#Convert</ejb-link>
  </ejb-local-ref>
</web-app>
```

- Tham chiếu cục bộ đến EJB được khai báo bằng cách dùng element **<ejb-local-ref>** trong DD **web.xml** trên. Element này chứa các element lồng:

- Element tùy chọn **<description>** chứa mô tả.
- Element **<ejb-ref-name>** chứa tên của tham chiếu liên quan đến context **java:comp/env.**
- Element **<ejb-ref-type>** chỉ định kiểu của EJB. Phải là **Entity** hoặc **Session.**
- Cặp element **<local>** và **<local-home>** chứa tên đầy đủ cặp giao diện truy xuất cục bộ của EJB cần tham chiếu.
- Element tùy chọn **<ejb-link>** kết nối tham chiếu đến một EJB khác trong cùng gói EJB hoặc trong cùng ứng dụng J2EE. Trị của **<ejb-link>** là **<ejb-name>** của bean được tham chiếu. Nếu có nhiều EJB cùng **<ejb-**

name> trị này sẽ là đường dẫn đến gói EJB (**.jar**), **<ejb-name>** của bean cần tham chiếu gắn phía sau tên gói EJB, tách ra bởi dấu **#**. Như vậy trong DD trên trị của **<ejb-link>** là **Convert** hoặc cụ thể hơn **convert.jar#Convert** đều được. Element **<ejb-link>** phải được chỉ định trong JBoss để so trùng tham chiếu cục bộ đến bean tương ứng.

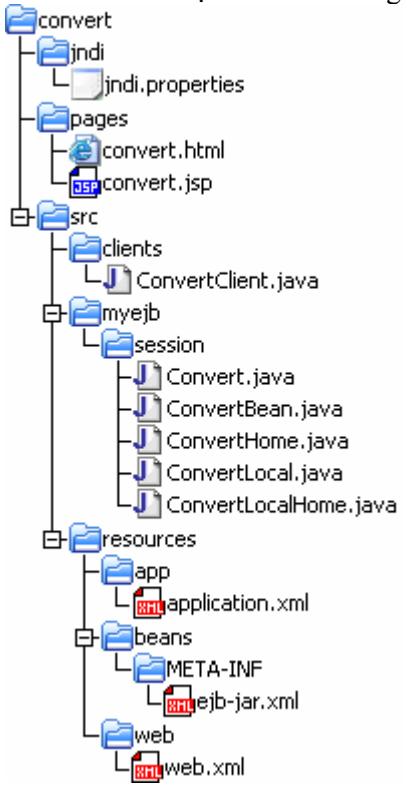
- Ứng dụng J2EE cũng cần DD **application.xml** cho việc lắp ráp ứng dụng J2EE, tạo thành gói **.EAR**.

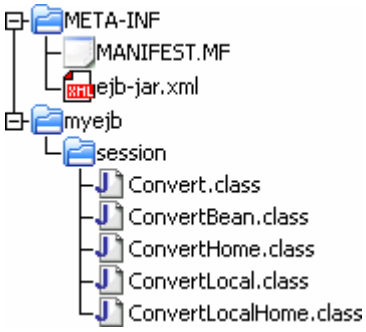
```
<?xml version="1.0"?>
<application xmlns="http://java.sun.com/xml/ns/j2ee" version="1.4"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com /xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/application_1_4.xsd">
  <display-name>Convert</display-name>
  <module>
    <web>
      <web-uri>convert.war</web-uri>
      <context-root>Convert</context-root>
    </web>
  </module>

  <module>
    <ejb>convert.jar</ejb>
  </module>
</application>
```

C. Triển khai trên JBoss 4.x

1. Cấu trúc thư mục và đóng gói

- Cấu trúc thư mục lưu trữ thường tổ chức như sau để tiện quản lý và build ứng dụng.





convert.jar

convert.war

convert.ear

2. Tạo **build.xml** để triển khai nhanh bằng Ant

- Chuẩn bị tập tin **build.xml** có nội dung như sau:

```
<?xml version="1.0"?>
<project name="JBoss" default="assemble" basedir=".">
  <property environment="env" />
  <property name="src.dir" value="${basedir}/src" />
```

```
  <property name="websrc.dir" value="${basedir}/servlet" />
  <property name="src.resources" value="${basedir}/src/resources" />
  <property name="jboss.home" value="${env.JBOSS_HOME}" />
  <property name="build.dir" value="${basedir}/build" />
  <property name="build.classes.dir" value="${build.dir}/classes" />
  <property name="app.name" value="convert" />

  <!-- Build classpath -->
  <path id="classpath">
    <fileset dir="${jboss.home}/client">
      <include name="**/*.jar" />
    </fileset>
    <pathelement location="${basedir}/lib/j2ee.jar" />
    <pathelement location="${build.classes.dir}" />
    <pathelement location="${basedir}/jndi" />
  </path>

<!--
0. Prepares the build directory
============
-->
  <target name="prepare">
    <delete dir="${build.dir}" />
    <mkdir dir="${build.dir}" />
    <mkdir dir="${build.dir}/servlet" />
    <mkdir dir="${build.classes.dir}" />
  </target>

<!--
1. Compiles, bundles, assembles, deploys
============
-->
  <target name="compile" depends="prepare">
    <javac srcdir="${src.dir}"
           destdir="${build.classes.dir}"
           debug="on"
           deprecation="on"
           optimize="off"
           includes="**">
           <classpath refid="classpath" />
    </javac>
  </target>

  <target name="ejbjar" depends="compile">
    <jar jarfile="${build.dir}/${app.name}.jar">
      <fileset dir="${build.classes.dir}">
        <include name="myejb/session/*.class" />
      </fileset>
      <fileset dir="${src.resources}/beans">
          <include name="**/*.xml" />
      </fileset>
     </jar>
   </target>

  <target name="webwar">
    <javac srcdir="${websrc.dir}"
           destdir="${build.dir}/servlet"
           debug="on"
           deprecation="on"
           optimize="off"
           includes="**">
      <classpath refid="classpath" />
```

```
    </javac>
    <war warfile="${build.dir}/${app.name}.war" webxml="${src.resources}/web/web.xml">
      <fileset dir="${build.dir}/servlet" />
    </war>
  </target>

  <target name="assemble" depends="ejbjar,webwar">
    <ear earfile="${build.dir}/${app.name}.ear"
         appxml="${src.resources}/app/application.xml">
      <fileset dir="${build.dir}" includes="*.jar,*.war" />
    </ear>
    <delete file="${build.dir}/${app.name}.jar" />
    <delete file="${build.dir}/${app.name}.war" />
    <move file="${build.dir}/${app.name}.ear"
          todir="${jboss.home}/server/default/deploy" />
  </target>

<!--
2. Runs the tests
============
-->
 <target name="run.client">
    <java classname="clients.ConvertClient" fork="yes" dir=".">
      <classpath refid="classpath" />
    </java>
  </target>
<!--

3. Cleans up
============
-->
  <target name="clean">
    <delete dir="${build.dir}" />
    <delete file="${jboss.home}/server/default/deploy/${app.name}.ear" />
  </target>
</project>
```

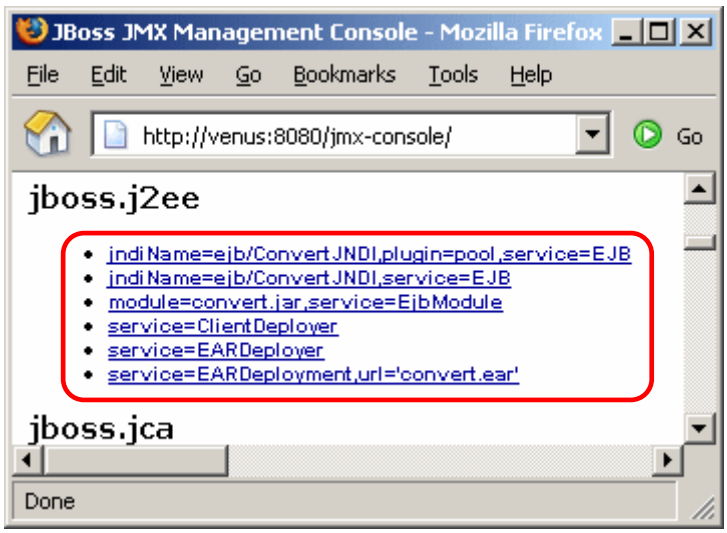
3. Triển khai và chạy ứng dụng

a) Chạy JBoss server

- Chạy **%JBOSS_HOME%\bin\run.bat** trong một console.

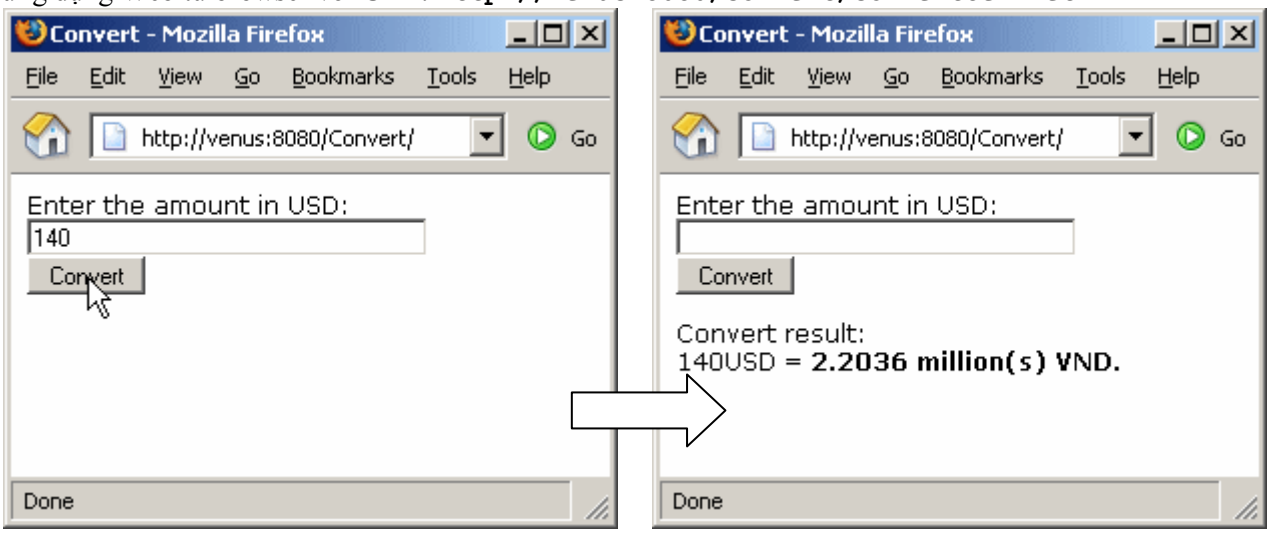
b) Chạy Ant

- Dùng Ant để tự động biên dịch, đóng gói, lắp ráp ứng dụng và triển khai nhanh: **ant**
- Ngay sau khi gói ứng dụng J2EE **convert.ear** được tạo trong thư mục **build** và được sao chép một cách tự động vào thư mục **%JBOSS_HOME%\server\default\deploy**, lập tức thấy chi tiết triển khai gói này trong console chạy JBoss server. Đây là khả năng hot deployment của JBoss.
- Cũng có thể theo dõi kết quả triển khai trong jmx-console quản lý JBoss:



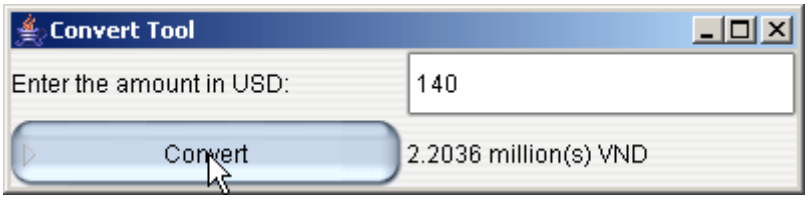
c) Chạy ứng dụng Web truy xuất cục bộ

- Chạy ứng dụng Web từ browser với URL: **`http://venus:8080/Convert/ConvertServlet`**



d) Chạy ứng dụng độc lập truy xuất từ xa

- Dùng Ant: **`ant run.client`**



e) Thu dọn

- Thu dọn kết quả bằng Ant: **`ant clean`**