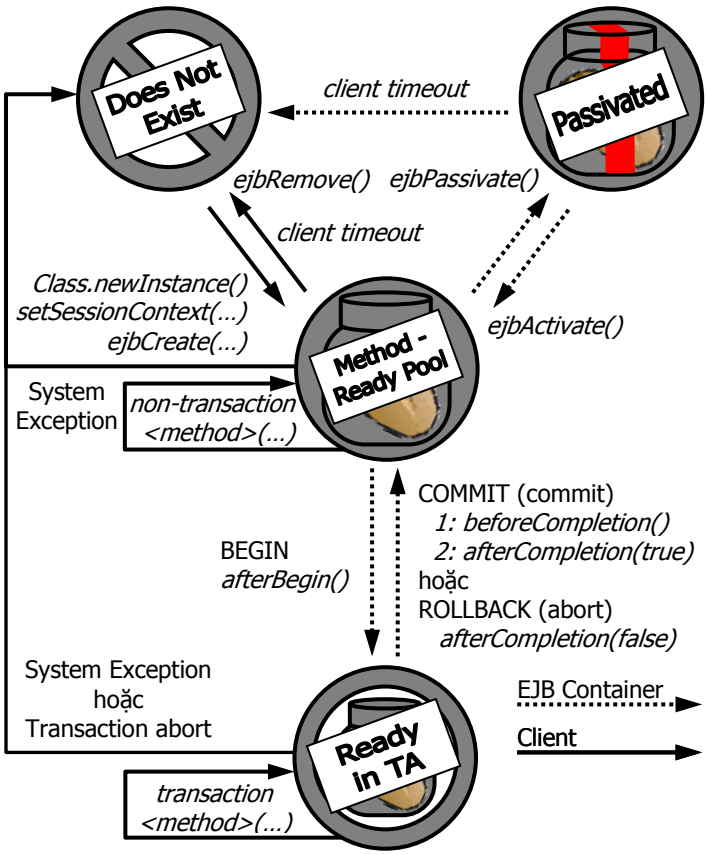
Stateful Session Bean

I. Khái niệm

1. Stateful Session Bean
- Một phiên giao dịch giữa client và bean còn gọi là một conversation. Một Stateful Session Bean có khả năng lưu lại trạng thái của conversation trong các biến thực thể của lớp bean xuyên qua nhiều request của một conversation. Một ví dụ quen thuộc là shopping cart dùng để lưu trữ các món hàng đã chọn xuyên qua hàng loạt request chọn hàng (hoặc loại bỏ hàng đã chọn) của một session mua hàng. Như vậy, Stateful Session Bean chứa trong nó business logic và trạng thái conversation của client.
- Stateful Session Bean được thiết kế để phục vụ cho **một**[1] client trong đời sống của nó thông qua một EJB(Local) Object.
- Stateful Session Bean không có khả năng tự lưu giữ (persistent) giống như entity bean. Để lưu giữ được trạng thái của conversation như trên, Stateful Session Bean có 3 trạng thái:
    - Does Not Exist: không tồn tại trong bộ nhớ.
    - Method-Ready Pool: sẵn sàng phục vụ các yêu cầu từ client được ủy nhiệm (delegate) bởi EJB(Local) Object. Nếu phương thức yêu cầu tham gia một transaction, thực thể bean sẽ được chuyển đến trạng thái Ready in TA.
    - Passivated: thụ động, lưu trữ trạng thái của conversation, sẽ được kích hoạt (activate) khi cần.
- Chọn cách cài đặt Stateful Session Bean khi:
    - Client triệu gọi phương thức nhiều lần, một session có nhiều request.
    - Cần lưu trữ thông tin client xuyên qua nhiều request đó.
- Trạng thái conversation cần lưu trữ, sẽ được lưu trong biến thực thể của EJB, có thể là các đối tượng do đặc tả EJB quy định:
    - Đối tượng serializable;
    - Tham chiếu đến NULL;
    - Tham chiếu đến một remote interface;
    - Tham chiếu đến một home interface;
    - Tham chiếu đến một local interface;
    - Tham chiếu đến một local home interface;
    - Tham chiếu đến SessionContext;
    - Tham chiếu đến JNDI ENC;
    - Tham chiếu đến UserTransaction;
    - Tham chiếu đến một resource factory.

2. Vòng đời của Stateful Session Bean

[1] Đây là mô hình khái niệm, một số container cung cấp khả năng instance pooling và instance swapping



- Khi phương thức **`create(...)`** của Home(Local) interface được gọi, container triệu gọi phương thức **`newInstance()`** của lớp bean để tạo một thực thể bean mới. Tiếp theo, container triệu gọi phương thức **`setSessionContext()`** để liên kết bean với một SessionContext dùng suốt vòng đời của nó, lúc này bean sẽ được gán cho một EJB(Local) Object. Cuối cùng, container triệu gọi phương thức **`ejbCreate(...)`** tương ứng với **`create(...)`** của Home(Local) interface. Thực thể bean bây giờ ở trạng thái Method-Ready Pool.
- Thực thể bean trong trạng thái Method-Ready Pool sẵn sàng đáp ứng các lời gọi bussiness method từ client. Tùy theo việc mô tả các thuộc tính transaction trong deployment descriptor và transaction context liên kết với client triệu gọi, một business method được triệu gọi hoặc trong một transaction context hoặc không (non-transaction):
  - Một phương thức non-transaction được thực hiện bởi thực thể bean trong trạng thái Method-Ready Pool.
  - Khi phương thức triệu gọi có tham gia một transaction, container sẽ gọi phương thức **`afterBegin()`** trước phương thức triệu gọi để bắt đầu transaction. Thực thể bean bây giờ ở trạng thái Ready in TA, nghĩa là đang liên kết với một transaction và sẽ duy trì liên kết này cho đến khi transaction hoàn tất.
    Phương thức tham gia transaction được thực hiện bởi thực thể bean trong trạng thái Ready in TA.
    Khi một transaction hoàn tất, container gọi **`afterCompletion()`** trên thực thể, chỉ định trạng thái hoàn tất (commit hoặc rollback). Thực thể bean được chuyển về trạng thái Method Ready Pool. Nếu rollback xuất hiện, thực thể bean sẽ khôi phục lại các trạng thái conversation như thời điểm bắt đầu transaction.
- Để tiết kiệm tài nguyên khi không phục vụ client, tùy theo chính sách lưu trữ, container có thể chuyển bean trạng thái Passivated bằng phương thức **`ejbPassivate()`**. Trạng thái của conversation sẽ được lưu, cách lưu trữ tùy đối tượng. Sau đó thực thể bean bị đẩy khỏi bộ nhớ một cách đơn giản trong lúc EJB(Local) Object vẫn còn giữ liên hệ với client. Một thực thể bean không thể chuyển sang trạng thái Passivated khi đang tham gia trong một transaction.
- Khi thực thể bean đang trong trạng thái Passivated, container có thể chuyển thực thể bean sang trạng thái Does Not Exist sau khi quá hạn session timeout do deployer chỉ định.
- Khi có yêu cầu của client đối với một bean đang ở trạng thái Passivated, container sẽ kích hoạt thực thể bean bằng cách khôi phục trạng thái conversation trong các field từ nơi lưu trữ rồi gọi **`ejbActivate()`**. Thực thể bean được chuyển trở lại trạng thái Method Ready Pool.
- Khi client triệu gọi phương thức **`remove()`** trên Home(Local) hoặc EJB(Local), container sẽ triệu gọi **`ejbRemove()`** trên thực thể bean. Điều này đưa bean vào trạng thái Does Not Exist, tách rời khỏi EJB(Local) Object. Triệu gọi từ client sẽ ném ra **`java.rmi.NoSuchObjectException`** hoặc **`javax.ejb.NoSuchObjectLocalException`**. Phương thức ejbRemove() không thể gọi khi thực thể đang tham gia trong một transaction, nếu thử gọi sẽ ném ra

javax.ejb.RemoveException. Chú ý rằng container cũng có thể triệu gọi phương thức **ejbRemove()** trên thực thể bean không có client gọi (client timeout).
- Trong các trường hợp sau, thực thể bean sẽ chuyển trực tiếp sang trạng thái Does Not Exist, không gọi phương thức **ejbRemove()**:

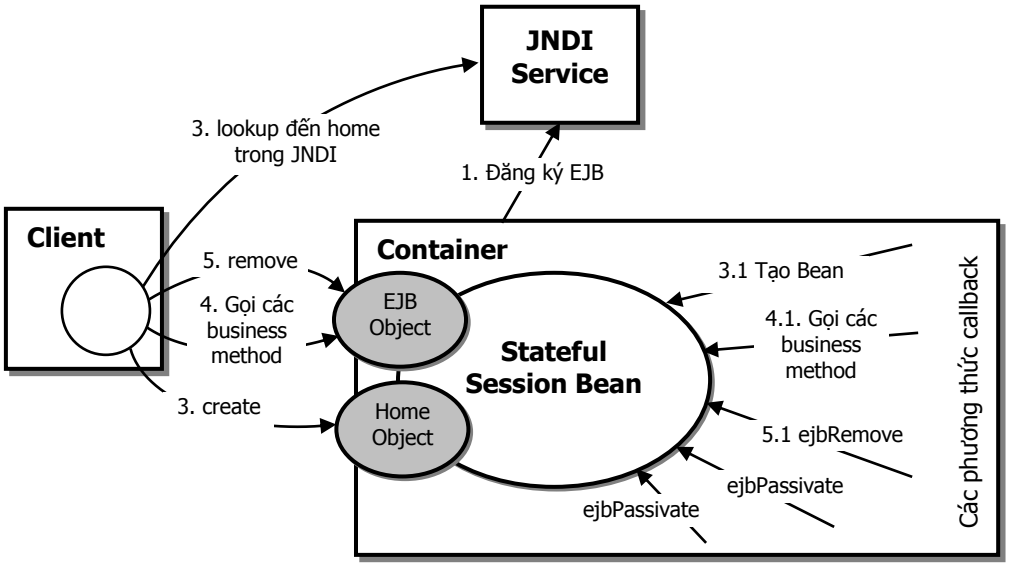
- EJB Container không hoạt động (crash).
- Exception hệ thống ném từ phương thức của thực thể đến container: **java.lang.RuntimeException** (hoặc subclass), **javax.ejb.EJBException**, **java.rmi.RemoteException**, lỗi khi kết nối cơ sở dữ liệu, các exception JNDI.
- Transaction rơi vào trạng thái abort.

3. Các phương thức callback
- Phương thức **ejbCreate(...)** khởi gán dữ liệu thành viên với trạng thái conversation cần lưu trữ. Sau khi được tạo, bean có thể ở:

- Trạng thái Method-Ready, thực hiện các business method phục vụ cho client.
- Không cần thiết, tùy chính sách lưu trữ container chuyển bean sang trạng thái Passivated.
- Chuyển sang trạng thái Does Not Exist do lỗi container hoặc do người dùng chấm dứt session.

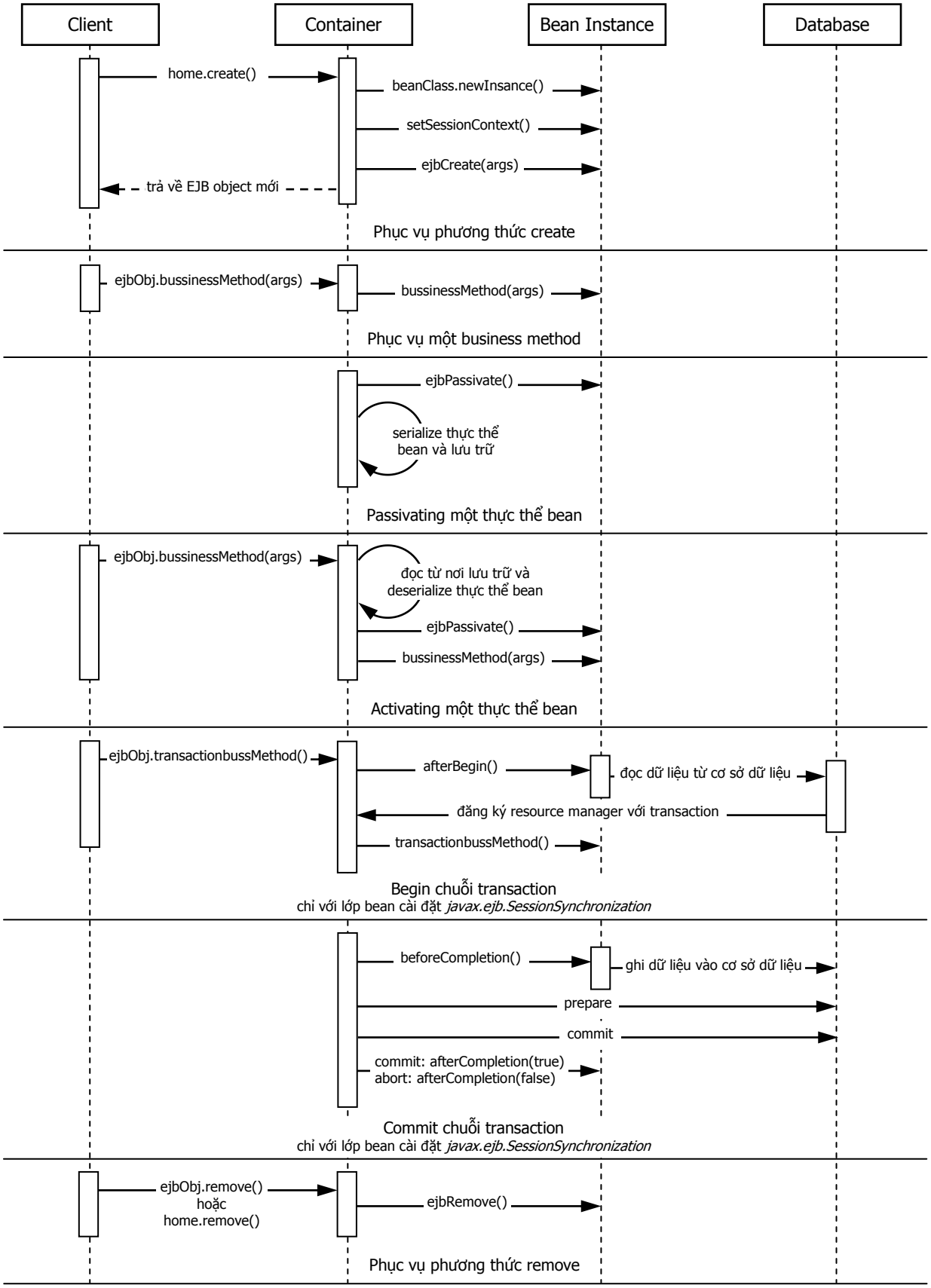
Phương thức **ejbCreate(...)** được nạp chồng cho phép khởi tạo thực thể bean bằng nhiều cách, phải có ít nhất một phương thức **ejbCreate(...)**.
- Phương thức **ejbPassivate()** sẽ được container gọi *trước* khi chuyển bean vào trạng thái Passivated, lưu trữ trạng thái của conversation. Phương thức này chuyển thực thể bean đến trạng thái thích hợp cho quá trình passivation, các tài nguyên liên kết với bean sẽ được giải phóng. Tài nguyên có thể là: socket, kết nối JDBC, ... tham chiếu và nguồn Container chuyển trạng thái của bean thành Passivated theo nguyên tắc **LRU** (Least Recently Used – bean tồn tại lâu nhất sẽ được chọn để thụ động hóa).
- Phương thức **ejbActivate()** chuyển bean từ trạng thái Passivated ngược trở lại trạng thái Method-Ready Pool. Container gọi **ejbActivate()** sau khi trạng thái conversation lưu trữ được đọc lại và bean được khôi phục. Phương thức này thường dùng khôi phục lại các tài nguyên đã bị giải phóng trong quá trình passivation. Các tài nguyên này do không serialize được nên cần tạo lại, ví dụ các kết nối socket hay cơ sở dữ liệu. Khi kích hoạt bean, container dùng nguyên tắc **JIT** (Just-In-Time – tức thời).
- Phương thức **ejbRemove()** được container gọi khi client triệu gọi phương thức **remove()** trên home interface hoặc remote interface. Phương thức này giải phóng tài nguyên, loại bỏ thực thể bean.
- Phương thức **setSessionContext(SessionContext ctx)** được container gọi để liên kết một session context với một bean chỉ định ngay từ đầu vòng đời của bean đó. Lớp bean cũng dùng phương thức này để lưu trữ các biến thực thể để truy xuất lại sau này nếu cần.

4. Hoạt động của Stateful Session Bean

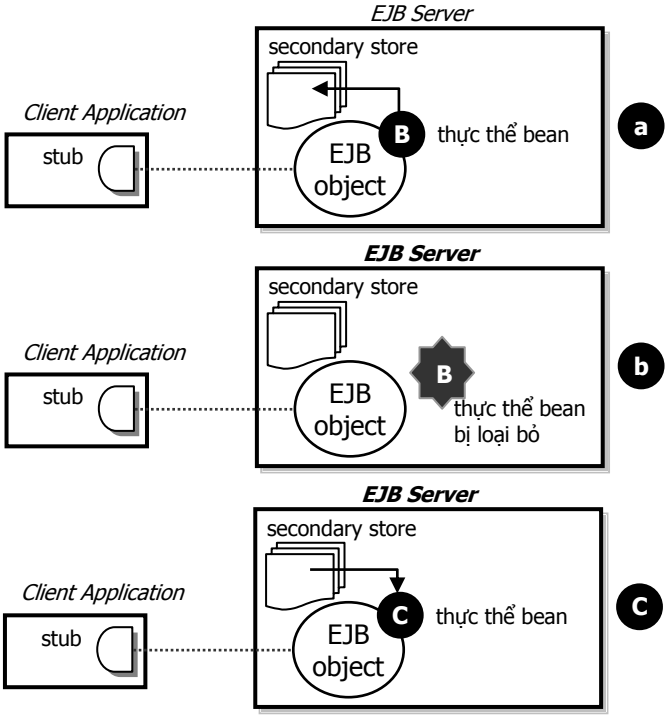


1- Container đăng ký tất cả bean đã triển khai với JNDI với tên JNDI chỉ định trong deployment descriptor.
2- Client tìm thấy Home(Local) interface của bean bằng cách **lookup()** thông qua JNDI.

3- Client dùng phương thức **`create()`** của Home(Local) interface để tạo ra EJB(Local) Object của bean. Phương thức này sẽ triệu gọi **`ejbCreate()`** trên bean.
4- Client gọi các business method thông qua EJB Object.
5- Để loại bỏ bean, client gọi phương thức **`remove()`** của Home(Local) interface, container sẽ gọi phương thức **`ejbRemove()`** trên bean.

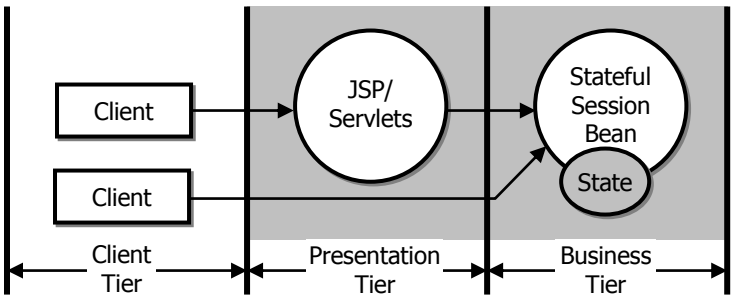


5. Cơ chế activation và passivation
- Không giống như Stateless Session Bean, Entity Bean và Message-Driven Bean, Stateful Session Bean không tham gia vào instance pool. Thay vào đó nó dùng cơ chế passivation và activation để tiết kiệm tài nguyên. Khi container cần tiết kiệm tài nguyên, nó có thể loại bean ra khỏi bộ nhớ, lúc này trạng thái của bean sẽ được lưu trữ bằng cách serialize xuống thiết bị lưu trữ. Khi client triệu gọi phương thức của bean, một thực thể bean mới lại được tạo ra, chứa trạng thái conversation do bean trước lưu trữ.
- Passivation (thụ động hóa) tách bean ra khỏi EJB Object liên kết với nó và lưu trạng thái chứa trong bean (trạng thái kết liên hệ với EJB Object và trạng thái conversation), sau đó loại bean ra khỏi bộ nhớ. Client không biết điều này và có thể vẫn liên lạc với EJB Object trong khi bean đã thụ động hóa.
- Activation (kích hoạth hóa) hồi phục lại thực thể bean liên kết với EJB Object. Lúc này container sẽ tự động tạo một thực thể mới và thiết lập các field theo dữ liệu được lưu trữ khi passivation. Sau đó EJB Object có thể ủy nhiệm các triệu gọi phương thức từ client đến bean như bình thường.
- Hình bên trình bày quá trình passivation và activation một Sateful Session Bean. (a) bean B được thụ động hóa, trạng thái của bean B được lưu trữ; (b) bean B tách ra khỏi EJB Object, loại khỏi bộ nhớ; (c) bean được kích hoạt, một thực thể mới, bean C được tạo ra với thông tin lưu trữ từ bean B và liên kết với EJB Object từ các thông tin nhận được.



II. Thiết kế

A. Tạo Stateful Session Bean



Thay vì lưu trữ trạng thái giao dịch (conversational state) trong **`HttpSession`** (JSP/Servlet) ta lưu trữ trạng thái giao dịch trong Stateful Session Bean

1. Các lớp hỗ trợ (helper)
- Lớp EJB có thể ném ra một số exception riêng. Cần bổ sung các lớp exception này vào gói .JAR chứa EJB.

```
package myejb.session.helper;
```

```
public class BookException extends Exception {
  public BookException() { }
  public BookException( String msg ) {
    super( msg );
  }
}
```

Ngoài ra cũng cần một số lớp tiện ích (helper class):

```
package myejb.session.helper;

public class IdVerifier {
  public IdVerifier() { }
  public boolean validate( String id ) {
    boolean result = true;
    for ( int i = 0; i < id.length(); ++i ) {
      if ( Character.isDigit( id.charAt( i ) ) == false )
        result = false;
    }
    return result;
  }
}
```

2. Lớp EJB

- Stateful Session Bean dùng các field dữ liệu riêng của lớp bean để lưu giữ trạng thái conversation. Các business method của nó thường thay đổi trực tiếp các field này. Vì vậy các business method của lớp bean thường là các getter/setter cho các field dữ liệu riêng của lớp bean.
- Do Stateful Session Bean dùng các field của lớp bean để lưu giữ trạng thái conversation nên thường cần một hoặc vài phương thức nạp chồng **`ejbCreate( … )`** có đối số để khởi tạo cho các field này.
- Một số resource không thể serialize được như: kết nối socket, kết nối cơ sở dữ liệu, … cần được giải phóng trong **`ejbPassivate()`** khi bean chuyển sang trạng thái Passivated, và cần được khôi phục trong **`ejbActivate()`** khi bean được kích hoạt trở lại. Các resource này được khai báo là **`transient`** (tài nguyên tạm thời).

Hãy quan sát việc sử dụng các resource này trong các phương thức callback **`ejbCreate()`** (tạo resource, lookup từ ENC), **`ejbPassivate()`** (giải phóng tạm thời resource), **`ejbActivate()`** (tạo lại resource) và **`ejbRemove()`** (giải phóng luôn resource). Resource này được dùng trong phương thức helper **`getListBooks()`**.

```
package myejb.session;

import myejb.session.helper.BookException;
import myejb.session.helper.IdVerifier;

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

public class CartBean implements SessionBean {
  private String customerName;       // state ví dụ cho việc nạp chồng ejbCreate()
  private String customerId;         // không minh họa cho việc lưu trữ state
  private Vector contents;           // minh họa cho việc lưu trữ state

  public static final String dbRef = "java:/Books";
  private SessionContext ctx = null;
  private transient DataSource dataSource = null;

  // Business methods, tác động vào các field của Bean
  public String getCustomerName() {
    return customerName;
  }

  public void addBook( String title ) {
```

```
    contents.addElement( title );
  }

  public void removeBook( String title ) throws BookException {
    boolean result = contents.removeElement( title );
    if ( result == false ) {
      throw new BookException( title + " not in cart." );
    }
  }

  public Vector getContents() {
    return contents;
  }

  // Cũng là business method (dùng resource)
  public Collection getListBooks() throws EJBException {
    final String query = "SELECT Name FROM Books";
    Vector list = new Vector();
    Connection con = null;
    Statement st = null;
    ResultSet rs = null;
    try {
      con = this.dataSource.getConnection();                    // dùng resource
      st = con.createStatement();
      rs = st.executeQuery( query );
      while ( rs.next() ) list.addElement( rs.getString( 1 ) );
    } catch ( SQLException ex ) {
      ex.printStackTrace();
      throw new EJBException( "db-error: " + ex.getMessage() );
    } finally {
      try { rs.close();  } catch ( Exception ex ) {}
      try { st.close();  } catch ( Exception ex ) {}
      try { con.close(); } catch ( Exception ex ) {}
    }
    return list;
  }

  // Callback methods
  public void ejbCreate( String person ) throws CreateException {
    if ( person == null ) {
      throw new CreateException( "Null person not allowed." );
    } else {
      customerName = person;
    }
    customerId = "0";
    contents = new Vector();
  }

  public void ejbCreate( String person, String id ) throws CreateException {
    if ( person == null ) throw new CreateException( "Null person not allowed." );
    else customerName = person;
    IdVerifier idChecker = new IdVerifier();
    if ( idChecker.validate( id ) ) customerId = id;
    else throw new CreateException( "Invalid id: " + id );
    contents = new Vector();
    // tạo resource
    try {
      Context c = new InitialContext();
      this.dataSource = ( DataSource )c.lookup( dbRef );     // khởi tạo resource
    } catch ( NamingException ex ) {
      String msg = "Cannot get Resource-Reference: " + ex.getMessage();
      throw new CreateException( msg );
```

```
    }
  }

  public void ejbRemove() {
    this.dataSource = null;                                    // giải phóng resource
  }

  public void ejbActivate() {
    try {
      Context c = new InitialContext();
      this.dataSource = ( DataSource )c.lookup( dbRef );       // khởi tạo lại resource
    } catch ( NamingException ex ) {
      String msg = "Cannot get Resource-Reference: " + ex.getMessage();
      throw new EJBException( msg );
    }
  }

  public void ejbPassivate() {
    this.dataSource = null;                                    // giải phóng resource
  }

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

3. Các giao diện

- Ví dụ minh họa này ta chỉ cần tạo giao diện truy xuất cục bộ từ lớp web đến EJB.

+ Component Local interface (EJB Object Local interface)

```
package myejb.session;

import myejb.session.helper.BookException;
import java.util.Vector;
import javax.ejb.EJBLocalObject;

public interface CartLocal extends EJBLocalObject {
  // business method (local invocation)
  public String getCustomerName();
  public void addBook( String title );
  public void removeBook( String title ) throws BookException;
  public Vector getContents();
  public java.util.Collection getListBooks();
}
```

+ Home Local interface

```
package myejb.session;

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

public interface CartLocalHome extends EJBLocalHome {
  CartLocal create( String person ) throws CreateException;
  CartLocal create( String person, String id ) throws CreateException;
}
```

4. Deployment Descriptor (DD)

- Các DD sau cần cho module EJB của ứng dụng J2EE, gồm:

+ DD cho EJB Container **ejb-jar.xml**. Chú ý **<session-type>** là **Stateful**.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>Cart</ejb-name>
      <local-home>myejb.session.CartLocalHome</local-home>
      <local>myejb.session.CartLocal</local>
      <ejb-class>myejb.session.CartBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
      <resource-ref>
        <res-ref-name>Books</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
        <res-sharing-scope>Shareable</res-sharing-scope>
      </resource-ref>
    </session>
  </enterprise-beans>
</ejb-jar>
```

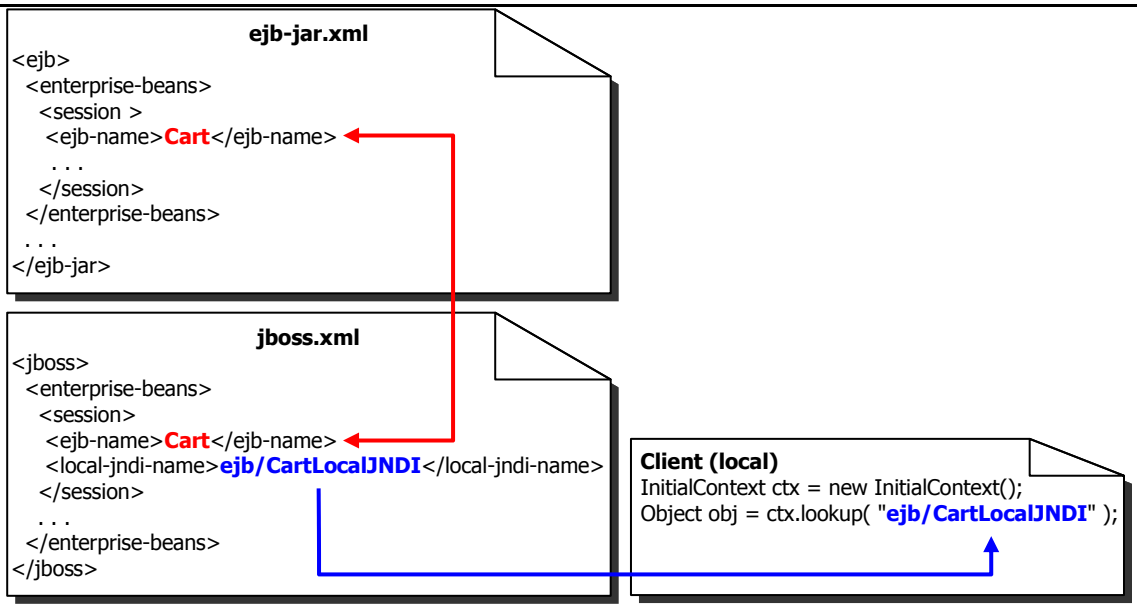
ENC

+ DD cho môi trường tác vụ (vendor-specific) cụ thể là JBoss: **`jboss.xml`** (do deployer đưa vào).

Trong DD này ta thấy tên gọi "nội bộ" trong ứng dụng của bean là **`Cart`**, trong khi tên gọi cho truy xuất cục bộ là **`ejb/CartLocalJNDI`**, đây chính là tên JNDI dùng khi lookup bean.

Có thể thêm vào element **`<container-configurations>`** để thấy khả năng của lưu trữ cache, điều kiện điều khiển quá trình activation và passivation của bean.

```
<?xml version="1.0"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"
 "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>Cart</ejb-name>
      <local-jndi-name>ejb/CartLocalJNDI</local-jndi-name>
      <resource-ref>
        <res-ref-name>Books</res-ref-name>
        <jndi-name>java:/Books</jndi-name>
      </resource-ref>
    </session>
  </enterprise-beans>
</jboss>
```



B. Tạo client truy xuất Staful Session Bean

1. Client là Web tier (truy xuất cục bộ)
- EJB được tạo trên là một shopping cart (giỏ hàng), sử dụng phổ biến trong các ứng dụng bán hàng online. Trang JSP front-end sẽ liệt kê các mặt hàng (sách) để người dùng lựa chọn. Người dùng có thể thêm hàng vào giỏ hàng, xem nội dung giỏ hàng, loại bỏ hàng khỏi giỏ hàng. Các thao tác này đều được Stateful Session bean thực hiện dưới sự triệu gọi cục bộ từ servlet.
a) Trang JSP front-end **index.jsp**:
- Trang JSP front-end lấy danh mục sách từ cơ sở dữ liệu thông qua session bean.

```
<%@ page import="java.util.*, javax.naming.*,
                 myejb.session.CartLocal, myejb.session.CartLocalHome"%>
<%@ page contentType="text/html;charset=windows-1252" %>
<%
Collection c = null;
try {
  Context ctx = new InitialContext( );
  CartLocalHome home = ( CartLocalHome )ctx.lookup( "ejb/CartLocalJNDI" );
  CartLocal cart = home.create( "Bill Gates", "1234" );
  c = cart.getListBooks();
  cart.remove();
  ctx.close();
} catch ( Exception e ) {
  out.println( e.getMessage() );
}
%>
<html>
<head><title>Shopping Cart Demo</title>
<script language="javascript">
function doAction( t ) {
  if ( t.value == "Add to Cart" )
    document.frm.action = 'CartServlet?mode=add';
  if ( t.value == "View Cart" )
    document.frm.action = 'CartServlet?mode=view';
  document.frm.submit();
}
</script>
</head>
<body>
<h1>Aptech Book's Store</h1>
<form method="post" name="frm">
 <table border="0" cellpadding="0" cellspacing="0" width="550">
  <tr>
   <td width="180"><font face="Verdana" size="2">Please choose your book:</font></td>
   <td width="370">
   <%
    if ( c == null || c.isEmpty() ) out.println( "Book list not found" );
    else {
      Iterator iter = c.iterator();
      out.println( "<select name='lstBook' size='1'>" );
      while ( iter.hasNext() ) {
       String b = ( String )iter.next();
       out.println( "<option>"+ b + "</option>" );
      }
      out.println( "</select>" );
    }
   %>
   </td>
  </tr>
  <tr>
   <td> </td>
   <td>
    <input value="Add to Cart" onclick="doAction( this )" type="button">
```

```
    <input value="View Cart" onclick="doAction( this )" type="button">
   </td>
  </tr>
  </table>
</form>
</body>
</html>
```

b) Servlet (controler)

- Servlet sẽ triệu gọi các business method của session bean để thực hiện các thao tác chèn (**`addBook()`**), loại bỏ (**`removeBook()`**) và xem nội dung của giỏ hàng (**`getContents()`**). Trong servlet, EJB được đặt vào biến session **`cart`**.

```
import java.io.PrintWriter;
import javax.naming.*;
import java.util.*;
import javax.rmi.PortableRemoteObject;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import myejb.session.*;

public class CartServlet extends HttpServlet {
  private CartLocalHome home;

  public void init( ServletConfig config ) throws ServletException {
    super.init( config );
  }

  private void callBean( HttpServletRequest req, HttpServletResponse resp )
     throws ServletException, IOException {
    resp.setContentType( "text/html; charset=windows-1252" );
    PrintWriter out = resp.getWriter();
    out.println( "<html><head><title>Cart</title></head><body>" );
    try {
      HttpSession session = req.getSession( true );
      CartLocal shoppingCart = ( CartLocal )session.getAttribute( "cart" );
      // Not cart? -> create Cart (Stateful Session Bean)
      if ( shoppingCart == null ) {
        Context ctx = new InitialContext();
        home = ( CartLocalHome )ctx.lookup( "ejb/CartLocalJNDI" );
        shoppingCart = home.create( "Bill Gates", "1234" );
      }

      String action = req.getParameter( "mode" );
      String bookTitle = req.getParameter( "lstBook" );
      // 'Add' action  -> EJBObject.addBook()
      if ( action.equals( "add" ) ) {
```

lookup EJB cục bộ, dùng EJB như shopping cart

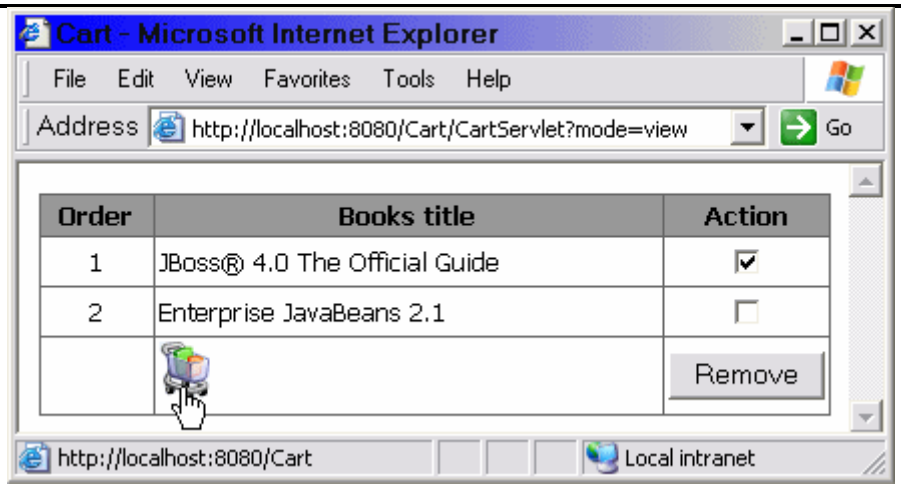
gọi **addBook()** của EJB, chèn sách vào cart

```
        shoppingCart.addBook( bookTitle );
        RequestDispatcher rqc = getServletContext().getRequestDispatcher( "/" );
        rqc.forward( req, resp );
      }
      // 'View' action -> EJBObject.getContents()
      if ( action.equals( "view" ) ) {
        int count = 0;
        String booktitle;
        out.println( "<font face='Tahoma' size='2'>Your cart contents:</font> <br>" );
        Vector bookList = shoppingCart.getContents();
        Enumeration item = bookList.elements();
        out.println( "<form action='CartServlet?mode=remove' method='post'>" );
        out.println( "<table border='1' cellpadding='2' cellspacing='0' width='395' " );
        out.println( "bordercolor='#666666' style='border-collapse: collapse'><tr>" );
        out.println( "<td bgcolor='#999999' width='53' align='center'>" );
        out.println( "<b><font face='Tahoma' size='2'>Order</font></b></td>" );
        out.println( "<td bgcolor='#999999' width='257' align='center'><b>" );
        out.println( "<font face='Tahoma' size='2'>Books title</font></b></td>" );
        out.println( "<td bgcolor='#999999' align='center'>" );
        out.println( "<b><font face='Tahoma' size='2'>Action</font></b></td>" );
        out.println( "</tr>" );

        while ( item.hasMoreElements() ) {
          ++count;
          booktitle = ( String )item.nextElement();
          out.println( "<tr>" );
          out.println( "<td align='center'><font face='Tahoma' size='2'>"
                       + count + "</font></td>" );
          out.println( "<td><font face='Tahoma' size='2'>" + booktitle + "</font></td>");
          out.println( "<td align='center' valign='middle'>" );
          out.println( "<input type='checkbox' name='rmv' value='" + booktitle + "'>" );
          out.println( "</td>" );
          out.println( "</tr>" );
        }

        out.println( "<tr>" );
        out.println( "<td></td><td>" );
        out.println( "<a href='/Cart'><img src='cart.gif' border='0' /></a>" );
        out.println( "</td><td><input type='submit' value='Remove'></td>" );
        out.println( "</tr>" );
        out.println( "</table></form>" );
      }
      // 'Remove' action -> EJBObject.removeBook()
      if ( action.equals( "remove" )  ) {
        String title[] = req.getParameterValues( "rmv" );
        if ( title == null ) title = new String[0];
        for ( int i = 0; i < title.length; i++ )
          shoppingCart.removeBook( title[i] );
        RequestDispatcher rqc =
           getServletContext().getRequestDispatcher( "/CartServlet?mode=view" );
        rqc.forward( req, resp );
      }
      // EJB -> session object
      session.setAttribute( "cart", shoppingCart );
    } catch ( Exception e ) {
      e.printStackTrace( out );
    } finally {
      out.println( "</body></html>" );
      out.close();
    }
  }
```

gọi **getContents()** của EJB, xem nội dung cart được EJB lưu giữ

gọi **remove()** của EJB, loại bỏ sách khỏi cart

```
  protected void doGet( HttpServletRequest req, HttpServletResponse resp )
     throws ServletException, IOException {
    callBean( req, resp );
  }

  protected void doPost( HttpServletRequest req, HttpServletResponse resp )
     throws ServletException, IOException {
    callBean( req, resp );
  }
}
```



c) DD cho Module Web

- Chuẩn bị **web.xml**, dành cho gói ứng dụng Web **.WAR**.

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Cart</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>CartServlet</servlet-name>
    <servlet-class>CartServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>CartServlet</servlet-name>
    <url-pattern>/CartServlet</url-pattern>
  </servlet-mapping>

  <ejb-local-ref>
    <ejb-ref-name>Cart</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>myejb.session.CartLocalHome</local-home>
    <local>myejb.session.CartLocal</local>
    <ejb-link>cart.jar#Cart</ejb-link>
  </ejb-local-ref>
</web-app>
```

- Chuẩn bị **jboss-web.xml**. là DD của môi trường tác vụ Jboss (do deployer đưa vào).

```
<?xml version="1.0"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"
 "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss-web>
```

```
  <ejb-local-ref>
    <ejb-ref-name>Cart</ejb-ref-name>
    <local-jndi-name>ejb/CartLocalJNDI</local-jndi-name>
  </ejb-local-ref>
</jboss-web>
```

2. Lắp ráp ứng dụng

- Chuẩn bị DD **application.xml** cho việc lắp ráp ứng dụng J2EE, tạo thành gói **.EAR**.

```
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN"
 "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>Cart</display-name>
  <module>
    <ejb>cart.jar</ejb>
  </module>

  <module>
    <web>
      <web-uri>cart.war</web-uri>
      <context-root>Cart</context-root>
    </web>
  </module>
</application>
```

Context khi gọi ứng dụng Web

- Chuẩn bị nguồn dữ liệu ODBC **Books**, đăng ký trong DSN. Tạo file **sqlserver-ds.xml** để đăng ký với JBoss bằng cách chuyển tập tin này vào **%JBOSS_HOME%/server/default/deploy/**.

Books

	Column Name	Data Type	Length	Allow Nulls
🔑	id	int	4	
	name	varchar	100	✓
	price	float	8	✓

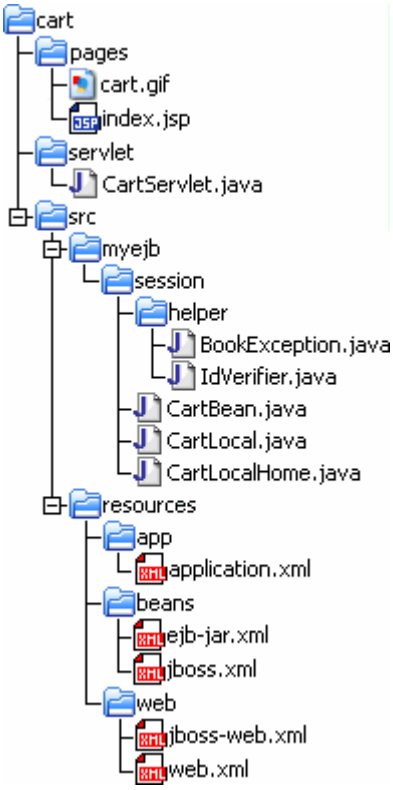
```
<?xml version="1.0"?>
<!DOCTYPE datasources
    PUBLIC "-//JBoss//DTD JBOSS JCA Config 1.5//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-ds_1_5.dtd"
<datasources>
  <local-tx-datasource>
    <jndi-name>Books</jndi-name>
    <connection-url>jdbc:odbc:Books</connection-url>
    <driver-class>sun.jdbc.odbc.JdbcOdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

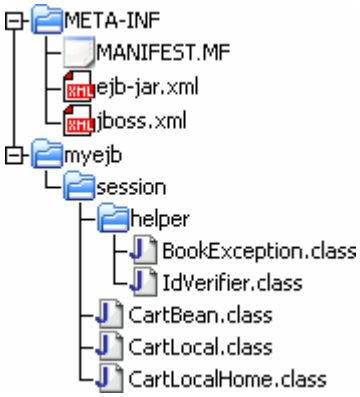
Cầu nối JDBC-ODBC

III. Triển khai trên JBoss 4.x

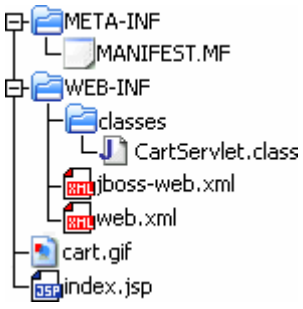
1. Cấu trúc thư mục và đóng gói

- Cấu trúc thư mục lưu trữ thường tổ chức như sau để tiện quản lý và build ứng dụng. Bên phải là các gói sau khi build, cần kiểm tra nội dung các gói để chắc rằng đã thực hiện đúng.





cart.jar



cart.war

cart.ear

3. Tạo (build) và triển khai nhanh bằng Ant

```
<?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="cart" />

  <!-- Build classpath -->
  <path id="classpath">
    <fileset dir="${jboss.home}/client" includes="**/*.jar" />
    <pathelement location="${jboss.home}/server/default/lib/javax.servlet.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
============
-->
```

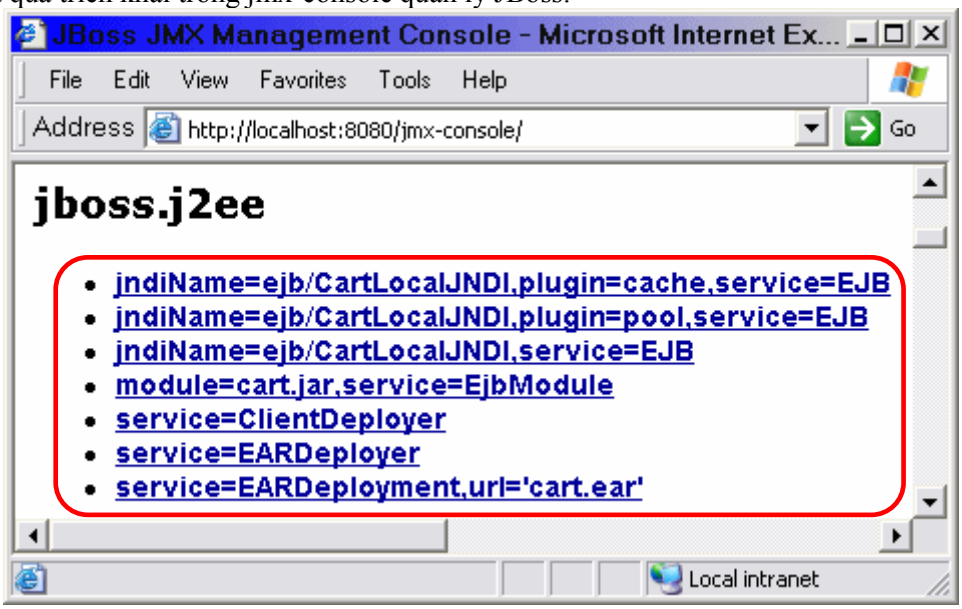
```
  <target name="compile" depends="prepare">
    <javac srcdir="${src.dir}"
           destdir="${build.classes.dir}"
           debug="on"
           deprecation="on"
           optimize="off"
           includes="**">
      <classpath refid="classpath" />
    </javac>
  </target>

  <target name="ejbjar" depends="compile">
    <jar jarfile="${build.dir}/${app.name}.jar">
      <metainf dir="${src.resources}/beans" includes="*.xml" />
      <fileset dir="${build.classes.dir}" includes="myejb/session/**/*.class" />
    </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="${src.resources}/web/web.xml">
      <webinf dir="${src.resources}/web" includes="jboss-web.xml" />
      <classes dir="${build.dir}/servlet" />
      <fileset dir="${basedir}/pages" />
    </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" />
    <copy file="${build.dir}/${app.name}.ear"
          todir="${jboss.home}/server/default/deploy" />
    <copy file="sqlserver-ds.xml" todir="${jboss.home}/server/default/deploy" />
  </target>

<!--
2. Cleans up
============
-->
  <target name="clean">
    <delete dir="${build.dir}" />
    <delete file="${jboss.home}/server/default/deploy/${app.name}.ear" />
    <delete file="${jboss.home}/server/default/deploy/sqlserver-ds.xml" />
  </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 **`cart.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.
- 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://localhost:8080/Cart`**

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