



Universidade do Minho
Departamento de Informática

App Servers e JEE

António Nestor Ribeiro
anr@di.uminho.pt

JEE

- Plataforma para desenvolvimento de aplicações orientadas à Web
 - Enterprise Java Beans
 - Servlets/Containers de Servlets
 - Java Server Pages
- Os produtos EE são servidores aplicacionais que providenciam uma implementação das tecnologias associadas (EJB, servlets, jsp, etc.)
- Providenciam três mecanismos básicos:
 - Naming (JNDI) para referenciar componentes
 - Descritores de anotações para código e para informação de deployment
 - Modelo de empacotamento de aplicações (WAR, EAR, etc.) mais sofisticado que o providenciado pelo JAR

Servlets

- Componente server-side para gerar conteúdos HTML dinamicamente.
- São uma classe Java que especializam uma classe base existente

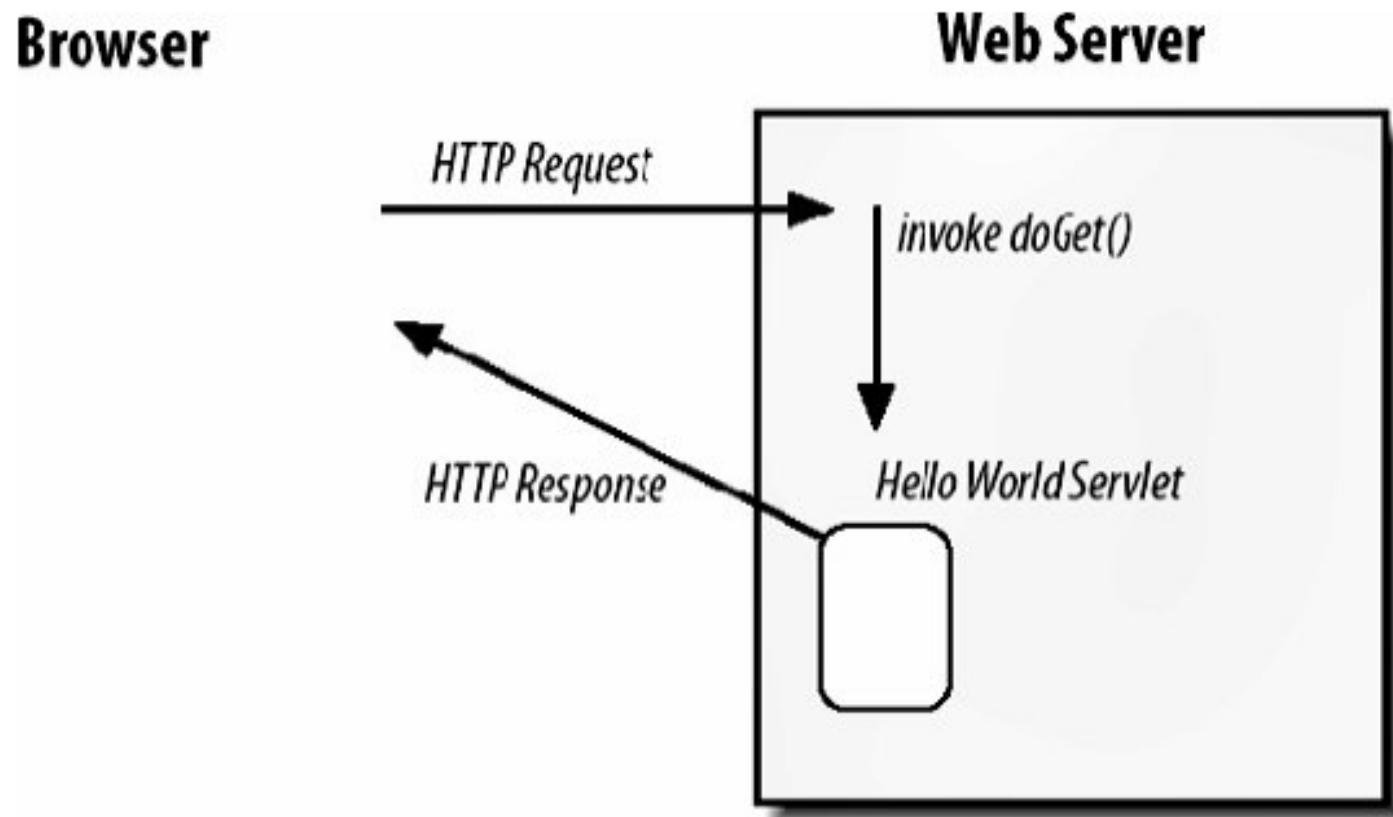
```
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet
{
    protected void doGet(HttpServletRequest
req,
HttpServletResponse
response)
    throws ServletException,java.io.IOException {
    try {
        ServletOutputStream writer = response.getWriter( );
        writer.println("<HTML><BODY>");
        writer.println("<h1>Hello World!!</h1>");
        writer.println("</BODY></HTML>");
    } catch(Exception e) {
        // handle exception
    }
    ...
}
```

JSP (Java Server Pages)

- Especialização das servlets de forma a simplificar o mecanismo de criação de conteúdos Web dinâmicos
- As páginas JSP são traduzidas e compiladas para servlets Java
 - Que são executadas no contexto de um container de servlets
- JSTL (JSP Standard Tag Library) permite que se utilizem tags para indicar em que ponto é que devem ser introduzidos os valores calculados e que se pretendem apresentar.

Servlets – ciclo de vida



Web Layer: alternativas possíveis

- A servlet faz output do código da página em HTML, isto é, o controller (do pattern MVC) gera a forma de apresentação final
- Utilização de JSP e JSTL. Combina informação de *layout* com código embedido que efectua a lógica da apresentação.
- Utilização de CSS. Permite a separação efectiva dos detalhes de apresentação e *rendering* ao separar a informação de lógica da informação relativa à interface.

Alternativa 1 – escrever HTML

```
<html>
<body>
  <table border="1">
    <tr>
      <th bgcolor="cccccc" align="left">Make</th>
      <th bgcolor="cccccc" align="left">Model</th>
      <th bgcolor="cccccc" align="right">Model Year</th>
    </tr>

    <tr>
      <td align="left">Toyota</td>
      <td align="left">Camry</td>
      <td align="right">2005</td>
    </tr>

    <tr>
      <td align="left">Toyota</td>
      <td align="left">Corolla</td>
      <td align="right">1999</td>
    </tr>

    <tr>
      <td align="left">Ford</td>
      <td align="left">Explorer</td>
      <td align="right">2005</td>
    </tr>
  </table>
</body>
</html>
```

Make	Model	Model Year
Toyota	Camry	2005
Toyota	Corolla	1999
Ford	Explorer	2005

Alternativa 2 – JSP e JSTL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    // DON'T FREAK OUT!!! This scriptlet code will go away once
    // we have a model and controller in place...

    String[ ][ ] carList = {
        {"Toyota", "Camry", "2005"},
        {"Toyota", "Corolla", "1999"},
        {"Ford", "Explorer", "2005"}
    };

    pageContext.setAttribute("carList", carList);
%>

<html>
<body>
    <table border="1">
        <tr>
            <th bgcolor="cccccc" align="left">Make</th>
            <th bgcolor="cccccc" align="left">Model</th>
            <th bgcolor="cccccc" align="right">Model Year</th>
        </tr>

        <c:forEach items="${carList}" var="car">
            <tr>
                <td align="left">${car[0]}</td>
                <td align="left">${car[1]}</td>
                <td align="right">${car[2]}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

Alternativa 3 – JSP e CSS

JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    // DON'T FREAK OUT!!! This scriptlet code will go away once
    // we have a model and controller in place...

    String[][] carList = {
        {"Toyota", "Camry", "2005"},
        {"Toyota", "Corolla", "1999"},
        {"Ford", "Explorer", "2005"}
    };

    pageContext.setAttribute("carList", carList);
%>

<html>
<head>
    <link rel="stylesheet" type="text/css" href="default.css">
</head>

<body>
    <table>
        <tr>
            <th>Make</th>
            <th>Model</th>
            <th class="model-year">Model Year</th>
        </tr>

        <c:forEach items='${carList}' var='car'>
            <tr>
                <td>${car[0]}</td>
                <td>${car[1]}</td>
                <td class="model-year">${car[2]}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

CSS

```
table
{
    border-style: solid;
    border-color: #aaa;
    border-width: 1px;
}

th
{
    color: #000;
    background-color: #ccc;
    border-style: solid;
    border-color: #aaa;
    border-width: 1px;
    font-weight: bold;
    text-align: left;
}

td
{
    color: #000;
    background-color: #fff;
    border-style: solid;
    border-color: #aaa;
    border-width: 1px;
    text-align: left;
}

.model-year
{
    text-align: right;
}
```

Um controlador

- Servlet que faz de controlador (do padrão MVC)
- Implementação dos métodos doGet e doPost

```
package com.jbossatwork;

import java.io.IOException;
import java.util.List;
import java.util.ArrayList;
import javax.servlet.*;
import javax.servlet.http.*;

public class ControllerServlet extends HttpServlet
{
    private static final String ACTION_KEY = "action";
    private static final String VIEW_CAR_LIST_ACTION = "viewCarList";
    private static final String ERROR_KEY = "errorMessage";
    private static final String ERROR_PAGE="/error.jsp";

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        processRequest(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        processRequest(request, response);
    }

    protected void processRequest(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String actionPerformed = request.getParameter(ACTION_KEY);
        String destinationPage = ERROR_PAGE;

        // perform action
        if(VIEW_CAR_LIST_ACTION.equals(actionName))
        {
            List carList = new ArrayList( );
            carList.add(new CarBean("Toyota", "Camry", "2005"));
            carList.add(new CarBean("Toyota", "Corolla", "1999"));
            carList.add(new CarBean("Ford", "Explorer", "2005"));
            request.setAttribute("carList", carList);

            destinationPage = "/carList.jsp";
        }
        else
        {
            String errorMessage = "[" + actionPerformed + "] is not a valid action.";
            request.setAttribute(ERROR_KEY, errorMessage);
        }

        // Redirect to destination page.
        RequestDispatcher dispatcher =
            getServletContext( ).getRequestDispatcher(destinationPage);
        dispatcher.forward(request, response);
    }
}
```

Especificação da View (JSP)

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
    <link rel="stylesheet" type="text/css" href="default.css">
</head>

<body>
    <table>
        <tr>
            <th>Make</th>
            <th>Model</th>
            <th class="model-year">Model Year</th>
        </tr>

        <c:forEach items='${carList}' var='car'>
            <tr>
                <td>${car.make}</td>
                <td>${car.model}</td>
                <td class="model-year">${car.modelYear}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

Configuração dos recursos

- Ficheiro **web.xml** que lista os recursos disponíveis de uma aplicação

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app version="2.4"
    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 web-app_2_4.xsd">

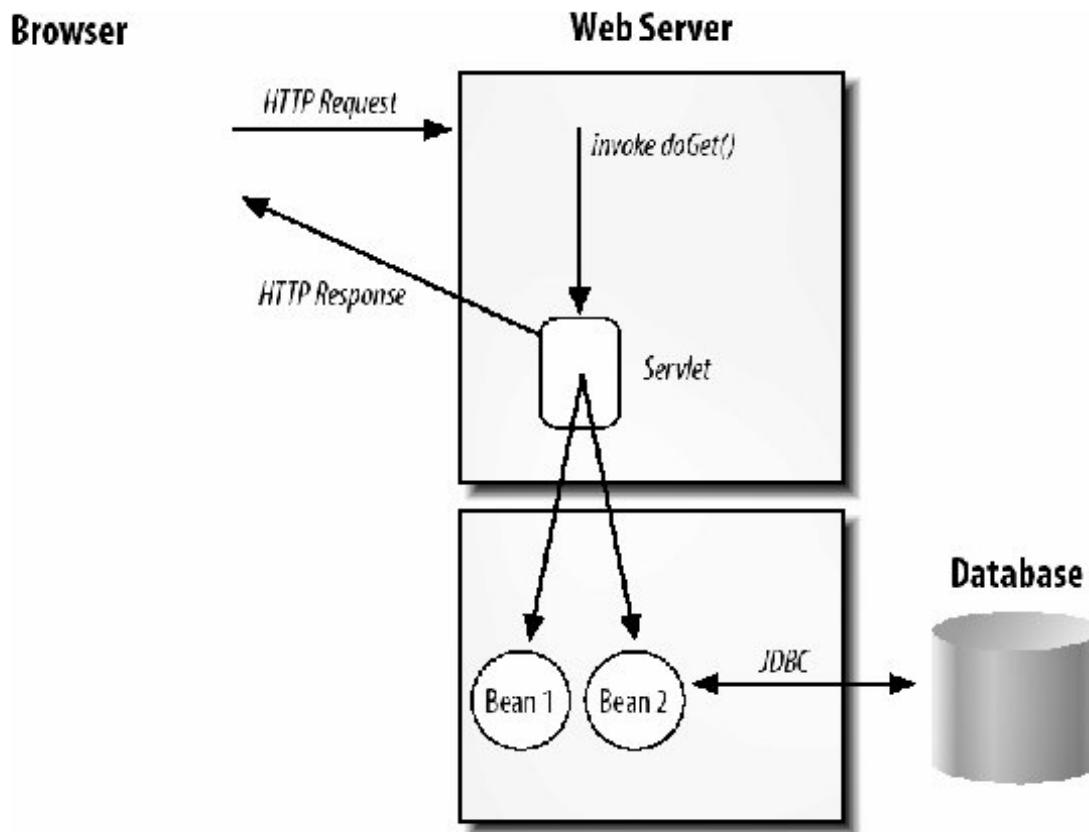
    <!-- servlet definition -->
    <servlet>
        <servlet-name>Controller</servlet-name>
        <servlet-class>com.jbossatwork.ControllerServlet</servlet-class>
    </servlet>

    <!-- servlet mapping -->
    <servlet-mapping>
        <servlet-name>Controller</servlet-name>
        <url-pattern>/controller/*</url-pattern>
    </servlet-mapping>

    <!-- The Welcome File List -->
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

Web e Camada de negócio

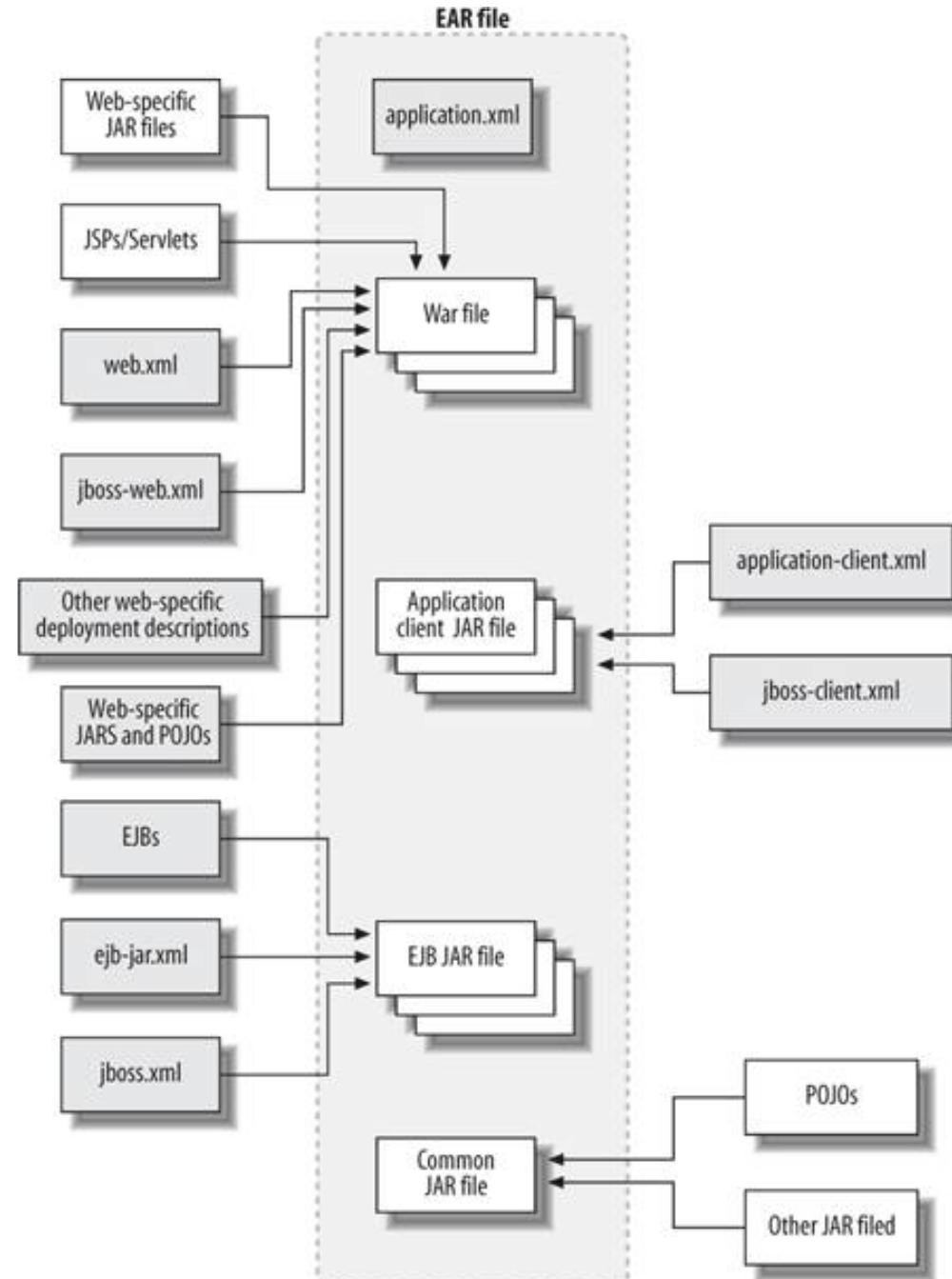


EAR Files

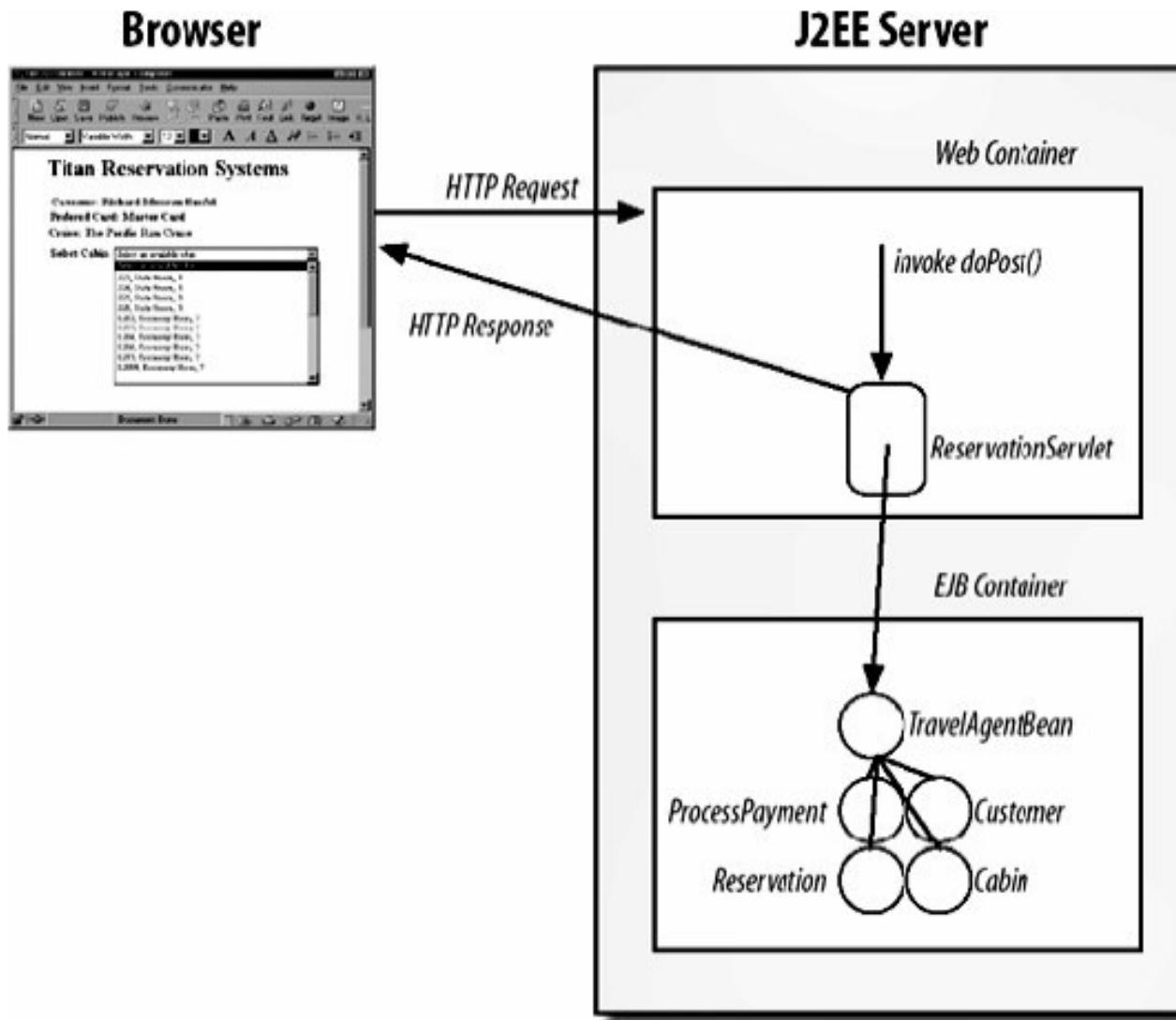
- Um EAR (enterprise archive file) é um ficheiro que permite empacotamento de:
 - Enterprise Java Beans JAR Files
 - Web Components JAR Files (war files)
 - Outros componentes JAVA (jar files)
- A esse empacotamento dá-se o nome de Java EE application, isto é, um componente que de forma autónoma representa uma aplicação multi-tier.

EAR

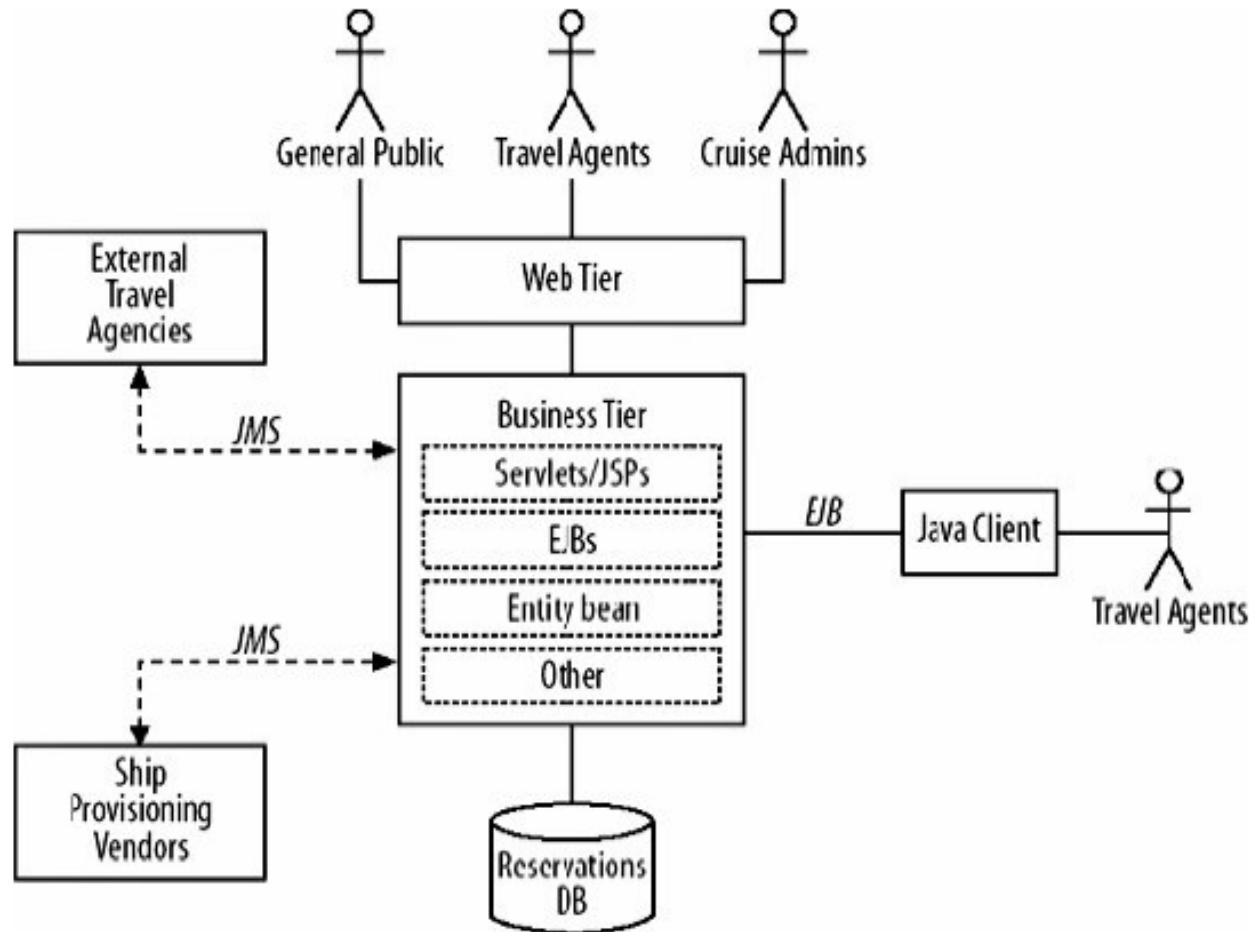
- Estrutura de uma EAR file



Uma aplicação multi-tier



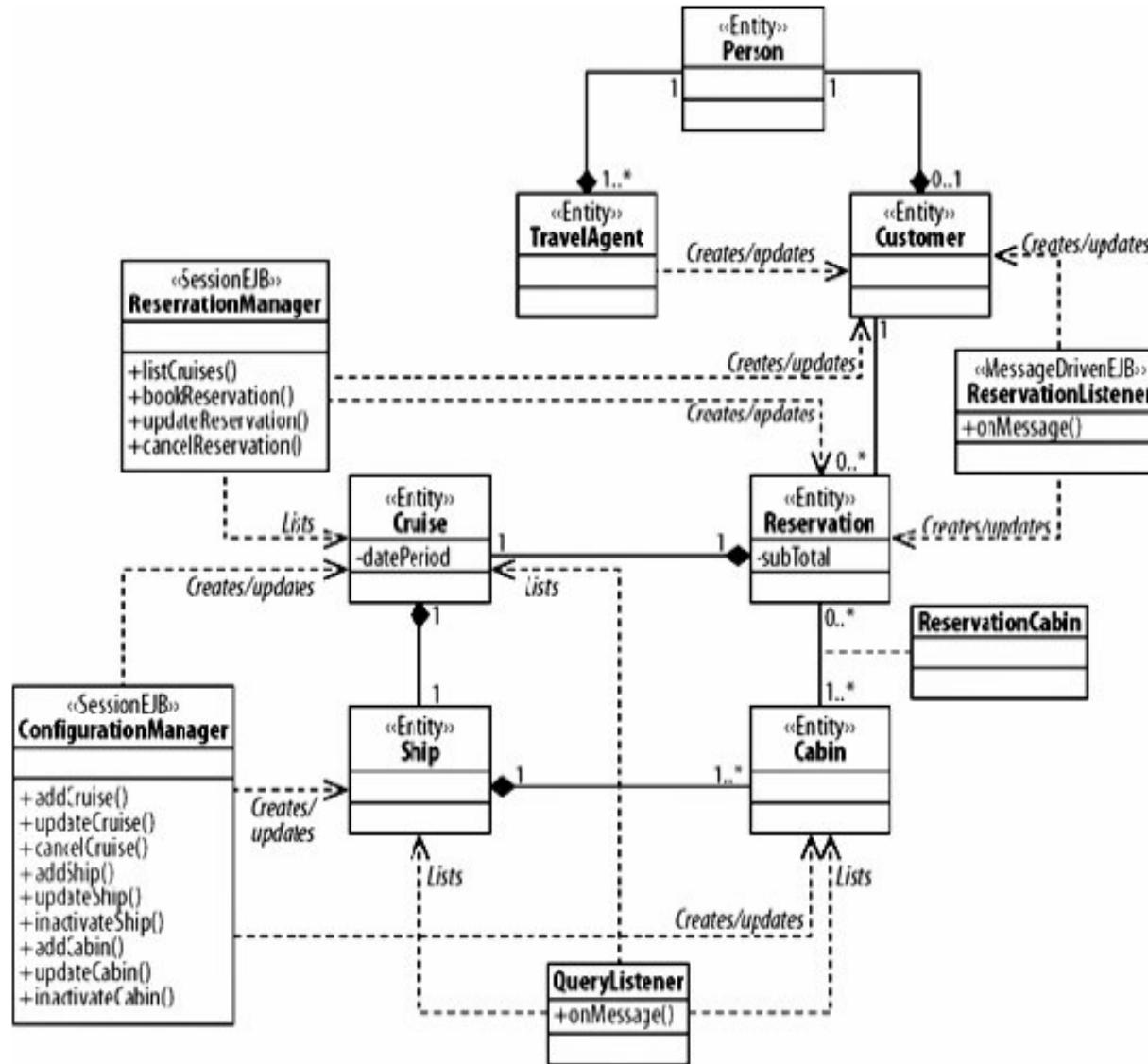
Aplicação multi-tier: componentes



Enterprise Java Beans

- Existem três tipos de EJBs
 - Session Beans: gerem processos ou tarefas, invocando outros EJBs e non-EJBs. As actividades dentro de um Session Bean são síncronas
 - Entity Beans: representam entidades guardadas em suporte persistente. São as entidades relevantes do modelo de domínio e sobre as quais existem operações do tipo CRUD.
 - Message Driven Beans: gerem processos ou tarefas, mas são invocadas assincronamente via JMS ou outro sistema de mensagens.

Do modelo à implementação



Session Beans

- Dois tipos de session beans:
 - Stateful Session Beans:
 - Entidade que mantém um relacionamento (diálogo) continuado com um cliente. Exemplo: ShoppingCart
 - A aplicação não tem de passar toda a informação porque ela reside no stateful bean.
 - O estado interno é mantido e é criada uma instância para cada cliente que invoca o construtor do Bean.
 - Stateless Session Beans:
 - Não é mantido o estado e todos os parâmetros devem ser fornecidos explicitamente.
 - São mais escaláveis, por serem menos “pesados”.

Em resumo... por agora!

- Aplicações multi-tier:
 - Presentation Layer
 - Colecção de componentes que implementam a UI. Utilizam-se servlets, JSPs, CSS, etc.
 - Business Layer
 - EJBs e Managed Beans
 - Persistence Layer
 - Serialized Java beans, ORM, Hibernate, etc.