Servlets, JSP y EJB

Índice

Servlets ____ 2

¿Qué es un Servlet? ____ 2

Ciclo de vida ____ 2

Características ____ 3

Ejemplo de Servlet: ____ 3

JSP (Java Server pages) ____ 5

¿Qué es JSP? ____ 5

Características ____ 5

Ejemplo JSP ____ 6

EJB (Enterprise JavaBeans) ____ 7

¿Qué es EJB? ____ 7

Tipos de EJB ____ 8

Funcionamiento de un EJB ____ 9

Conclusión ____ 11

Bibliografía ____ 12

Adrián Vázquez Agost

Servlets

¿Qué es un Servlet?

Un Servlet es un sustituto del antiguo CGI, estos están escritos en C o Perl y los servlets en Java, un servlet es principalmente, una clase en Java utilizada para mejorar las capacidades de un servidor. Se utilizan para responder peticiones del servidor, pero también son integrados en aplicaciones web en forma de Applets de Java para que se ejecuten en los servidores en vez de en los navegadores web.

El principal uso de los Servlets es para generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

Ciclo de vida

El ciclo de vida de un servlet consta de tres fases.

1. Inicializar el servlet:

   Cuando el servidor carga el servlet, se ejecuta el método “init” del servlet, esto es requerido para poder manejar las peticiones de los clientes y antes de que el servlet sea destruido. Solo se llama una vez al “init” y no se llamara de nuevo a menos que se recargue el servlet. Para recargar un servlet primero el servidor tiene que llamar al método “destroy”

2. Interactuar con los clientes:

   Después de inicializarse, el servlet responde a las peticiones de los clientes. Estas peticiones son procesadas por el propio servlet, por eso, las variables compartidas pueden dar problemas.

3. Destruir el servlet:

   Un servlet es ejecutado hasta que el servidor lo destruye, bien por cierre de servidor o a petición del administrador del sistema. El servidor usa el método “destroy” para destruir un servlet. Este método solo se ejecuta una vez, y puede ser llamado cuando hay respuestas en proceso, por lo que hay que tener la atención de esperarlas.

Características

- Son independientes del servidor utilizado y su sistema operativo.
- Los Servlets pueden llamar a otros Servlets.
- Obtienen fácilmente información.
- Permiten la utilización de cookies y sesiones.
- Permiten generación dinámica de código HTML.

Ejemplo de Servlet:

Este es un ejemplo de Servlet en código Java.

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

public class HolaMundo extends HttpServlet {

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

  public void service(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException  {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<html>");
    out.println("<body>");
    out.println("<h1>Hola Mundo</h1>");
    out.println("</body>");
    out.println("</html>");
  }
}
```

Aquí vemos el método init, que se tiene que implementar siempre para inicializarlo.

Después del método “init” se redefine el método service, este tiene dos parámetros, el primero (HttpServletRequest) representa la petición del cliente, el segundo método (HttpServletResponse) representa la respuesta del servidor (del servlet, más concretamente).

Con este ejemplo no se usa el parámetro HttpServletRequest porque no necesitamos ninguna información del cliente, de la clase HttpServletResponse se usan dos métodos: setContextType(string str) para establecer el tipo de respuesta, en este caso para indicar que se trata de una página web, por eso se usa “text/html”. El segundo método es: PrinterWriter getWriter(void) con este obtenemos una clase PrinterWriter donde se ira escribiendo los datos que el cliente reciba.

JSP (Java Server pages)

¿Qué es JSP?

JSP (JavaServer Pages) es una tecnología hibrida que al igual que PHP permite incorporar scripts para añadir código Java directamente a las páginas .jsp, pero también permiten implementar un conjunto de etiquetas que interaccionan con los objetos Java del servidor, sin necesidad que aparezca código fuente en la página. Para desplegar y correr JSP se necesita un servidor web compatible con contenedores servlet.

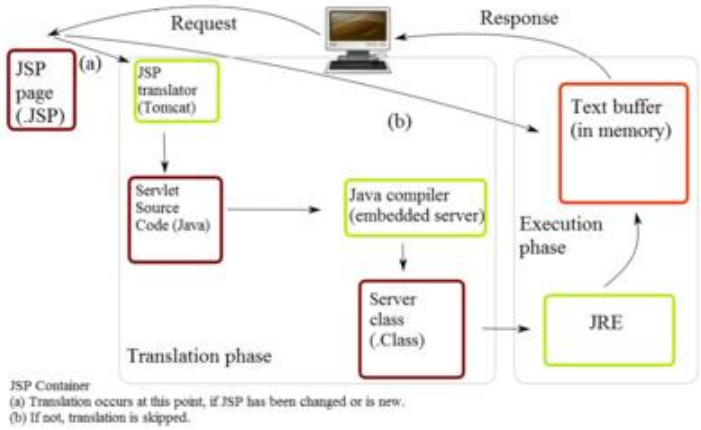
En realidad los JSPs son una forma alternativa de crear servlets ya que el código JSP se traduce a código servlet que se ejecuta produciendo como salida el código HTML que compone la página web de respuesta.

El uso de JSP es para crear páginas web dinámicas en servidor, usando lenguaje Java (igual que los servlets), en ese sentido son similares a otros lenguajes como PHP, ASP o los CGIs.

Características

- Mejor rendimiento al usar procesos ligeros (hilos Java).
- Manejo de múltiples peticiones.
- Facilidad para compartir recursos entre peticiones.
- Reutilizable
- Separación de código de presentación con el de implementación.
- División de trabajo

Ciclo de vida JSP:



Ejemplo JSP

Ejemplo de JSP.

```
package jsp_servlet;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import com.foo.bar; //importado como resultado de <%@ page
import="com.foo.bar" %>
import ...
class _myservlet implements javax.servlet.Servlet,
javax.servlet.jsp.HttpJspPage {
    //insertado como
    //resultado de <%! int serverInstanceVariable = 1;%>
    int serverInstanceVariable = 1;
    ...
    public void _jspService( javax.servlet.http.HttpServletRequest
request,
                             javax.servlet.http.HttpServletResponse
response )
      throws javax.servlet.ServletException,
             java.io.IOException
    {
        javax.servlet.ServletConfig config = ...;//obtener la
configuración del servlet
        Object page = this;
        PageContext pageContext = ...;//obtener el contexto de la
página para esta petición
        javax.servlet.jsp.JspWriter out = pageContext.getOut();
        HttpSession session = request.getSession( true );
        ...
```

EJB (Enterprise JavaBeans)

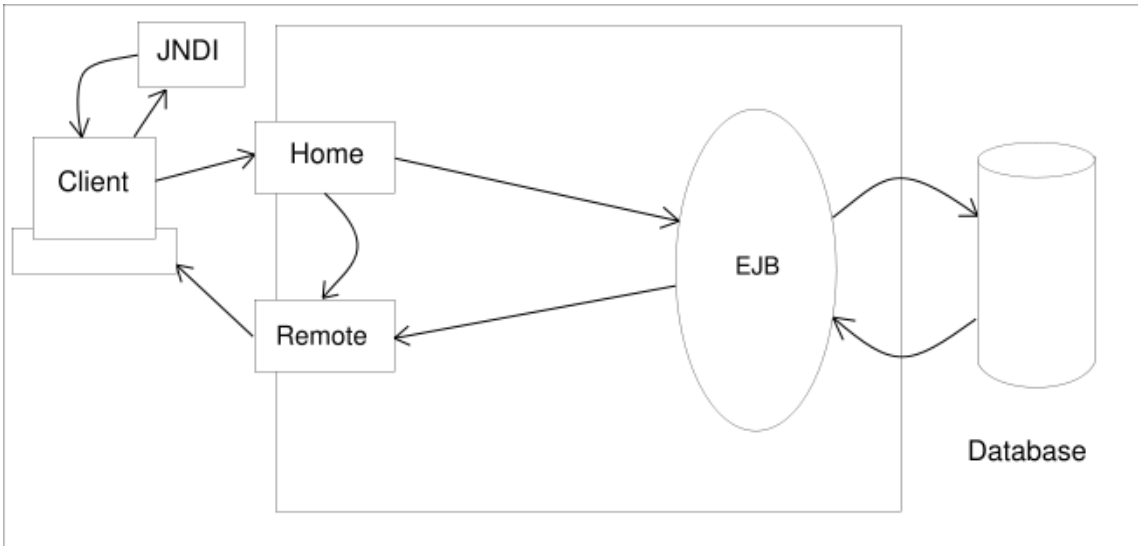
¿Qué es EJB?

EJB (Enterprise JavaBeans) es un componente de Java Enterprise que se usa principalmente para negocios, simplifica bastante la construcción de soluciones empresariales.

Su objetivo es otorgar al programador un modelo que permita abstraerse de los problemas generales de una aplicación empresarial, para centrarse en el desarrollo de la lógica del negocio, el hecho de estar basado en componentes permite que sean flexibles y reutilizables.

Para usar EJB se necesita un contenedor EJB/J2EE. Al usar EJB, al ser una API da acceso a todos los servicios EJB (manejo de transacciones, seguridad, persistencia…etc).

Aquí aparece la arquitectura básica de EJB:



Un ejemplo seria: existe una aplicación de bolsa y el bean proporciona implementación de un Broker, la interfaz de negocio de Broker, se compone de varios métodos, como “compra” o “vende”, esto provoca las siguientes llamadas:

Cliente: "Necesito realizar una petición de compra al bean Broker."
EJBObject: "Espera un momento, necesito comprobar tus permisos."
Contenedor EJB: "Sí, el cliente tiene permisos suficientes para llamar al método compra."
Contenedor EJB: "Necesito un bean Broker para realizar una operación de compra. Y no olvidéis comenzar la transacción en el momento de instanciaros."
Pool de beans: "A ver... ¿a quién de nosotros le toca esta vez?".
Contenedor EJB: "Ya tengo un bean Broker. Pásale la petición del cliente."

(Así sería el funcionamiento del proceso)

Tipos de EJB

- EJB de Entidad (Entity EJB):
  Objetivo: encapsular objetos del lado del servidor que almacena los datos.
  Característica: Persistencia.
    Dos tipos de persistencia:
      Gestionada por contenedor (CMP): se encarga de almacenar y recuperar datos del objeto de la entidad mediante mapeo o vinculación de las columnas de una tabla de la base de datos con atributo “objeto”.

      Gestionada por Bean (BMP): se encarga, mediante una base de datos u otro mecanismo, de almacenar y recuperar los datos a los que se refiere, responsabilidad del programador de implementar los mecanismos de persistencia

- EJB de Sesión (Session EJB):
  Objetivo: gestionan el flujo de la información en el servidor.
  Característica: sirven como fachada a los clientes de los servicios proporcionados por otros componentes del servidor.
    Dos tipos:
      Con estado (stateful): las variables de instancia almacenan datos específicos obtenidos en la conexión del cliente. Cada bean de sesión con estado, almacena el estado conversacional del cliente que interactúa con Bean. Este estado se modifica conforme el cliente realiza llamadas a los métodos de negocio de Bean y no se guarda cuando el cliente termina la sesión.
      Sin estado (stateless): son objetos distribuidos que carecen de estado asociado, permitiendo, que se acceda concurrentemente.

- EJB dirigidos por mensajes (Message-driven EJB):
  Objetivo: Se subscriben a un tema (topic) o a una cola (queue) y se activan al recibir un mensaje dirigido a dicho tema o cola. No requieren instanciación por parte del cliente.
  Característica: funcionamiento asíncrono.

Funcionamiento de un EJB

Para programar un bean se siguen estos pasos:

1. Escribe y compila la clase bean que contiene a todos los métodos de negocio.
2. Escribe y compila las dos interfaces del bean: home y componente.
3. Crea un descriptor XML del despliegue en el que se describa qué es el bean y cómo debe manejarse. Este fichero debe llamarse ejb-jar.xml.
4. Pon la clase bean, los interfaces y el descriptor XML del despliegue en un fichero EJB JAR . Podría haber más de un bean el mismo fichero EJB JAR, pero nunca habrá más de un descriptor de despliegue.
5. Despliega el bean en el servidor usando las herramientas proporcionadas por el servidor de aplicaciones.

Esto sería un ejemplo de clase EJB:

Primero se tendría que saber qué tipo de EJB implementar (estos tres tipos se definen con tres interfaces distintas: SessionBean, EntityBean y MessageBean).

En nuestro caso, vamos a definir un bean de sesión sin estado, por lo que la clase SaludoBean implementará la interfaz SessionBean.

```
package especialista;
import javax.ejb.*;

public class SaludoBean implements SessionBean {
   private String[] saludos = {"Hola", "Que tal?", "Como estas?",
   "Cuanto tiempo sin verte!", "Que te cuentas?", "Que hay de nuevo?"};

   // Los cuatro metodos siguientes son los de la interfaz
   // SessionBean
   public void ejbActivate() {
      System.out.println("ejb activate");
   }

   public void ejbPassivate() {
      System.out.println("ejb pasivate");
   }

   public void ejbRemove() {
      System.out.println("ejb remove");
   }

   public void setSessionContext(SessionContext cntx) {
      System.out.println("set session context");
   }

   // El siguiente es el metodo de negocio, al que
   // van a poder llamar los clientes del bean
   public String saluda() {
      System.out.println("estoy en saluda");
      int random = (int) (Math.random() * saludos.length);
      return saludos[random];
   }

   // Por ultimo, el metodo ejbCreate que no es de
   // la interfaz sessionBean sino que corresponde al
   // metodo de creacion de beans de la interfaz Home del EJB
   public void ejbCreate() {
      System.out.println("ejb create");
   }
}
```

Aparte de la clase EJB, hay que definir las interfaces que usare el cliente para comunicarse con el bean:

```
package especialista;

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

public interface Saludo extends EJBObject {
   public String saluda() throws RemoteException;
}
```

En esta interfaz se definen los métodos de negocio del bean que el cliente podrá llamar. Estos son métodos remotos.

Esta otra interfaz se usa para obtener una referencia a la interfaz de arriba.

```
package especialista;

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

public interface SaludoHome extends EJBHome {
   public Saludo create() throws CreateException, RemoteException;
}
```

El método create() se corresponde con el método ejbCreate() definido en la clase SaludoBean, y debe devolver el tipo Saludo de la interfaz componente.

Esto es todo lo que hay que escribir en Java, una clase y dos interfaces.

Falta un archivo XML que es el descriptor de despliegue, que detalla todo lo que el servidor necesita saber para gestionar el bean. El nombre del fichero siempre es: ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!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>
  <display-name>Ejb1</display-name>
  <enterprise-beans>

    <session>
      <display-name>SaludoBean</display-name>
      <ejb-name>SaludoBean</ejb-name>
      <home>especialista.SaludoHome</home>
      <remote>especialista.Saludo</remote>
      <ejb-class>especialista.SaludoBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>

  </enterprise-beans>
</ejb-jar>
```

Simplemente le decimos cuales son las clases bean y las interfaces home y componente.

Una vez hecho esto, solo hay que desplegar el bean.

Conclusión

Para realizar aplicaciones web de empresas es mejor utilizar Enterprise JavaBeans (EJB) puesto que está diseñado para eso y simplifica bastante a la hora de hacer las aplicaciones al tener en cuenta solo la parte lógica del negocio.

Para las aplicaciones corrientes, se pueden usar tanto Servlets como JSP, aunque JSP se traduce a código Servlet y para hacer aplicaciones en JSP se necesita un servidor Servlet. Es más potente el código Servlet para hacer aplicaciones web.

Bibliografía

Wikipedia (Información de Servlets mayormente, JSP y EJB)

Ejemplo de servlet

JSP

EJB