J2EE: Patrones J2EE

Antonio Espín Herranz

Contenidos

- Introducción.
- Singleton.
- DAO.
- Transfer Object / Value Object.
- Vista Compuesta.
- MVC.

Patrones de Diseño

- ¿Qué son?
  - Soluciones ya probadas para determinados tipos de problemas que se pueden resolver de la misma forma.
- Patrones:
  - Singleton
  - DAO
  - Transfer Object
  - Vista Compuesta
  - MVC

Singleton

- El patrón de diseño **singleton** (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

- Se puede utilizar para tener una única conexión a la Base de datos.

Ejemplo

```
public class Singleton {
private static Singleton INSTANCE = new Singleton();
// El constructor privado no permite que se genere un
    constructor por defecto
// (con mismo modificador de acceso que la definición de la
    clase)
private Singleton() {}
public static Singleton getInstance() { return INSTANCE; }
}
```

Transfer Object

- Define los tipos de datos que se van a mover en una aplicación.
- También se llama **Value Object**.
- Se corresponden con clases Java del tipo bean: atributos, mas métodos set / get.
- Agrupan en una clase las diferentes propiedades de una entidad.
- Cliente, Empleado, Pedido, …

Ejemplo

```
public class Project implements java.io.Serializable {
        public String projectId;
        public String projectName;
        public String managerId;
        public String customerId;
        public Date startDate;
        public Date endDate;
        public boolean started;
        public boolean completed;
        public boolean accepted;
        public Date acceptedDate;
        public String projectDescription;
        public String projectStatus;
        // Transfer Object constructors...  get / set
}
```

DAO

- Data Access Object:
  - Define mediante un interface el conjunto de operaciones a realizar con un determinado objeto.
  - Es la clase que implementa este interfaz quien dice como implementarlas.

Ejemplo

```
// Interface que todos los CustomerDAOs deben
   implementar:
public interface ICustomerDAO {
     public int insertCustomer(...);
     public boolean deleteCustomer(...);
      public Customer findCustomer(...);
     public boolean updateCustomer(...);
     public RowSet selectCustomersRS(...);
     public Collection selectCustomersTO(...); ...
}
public CustomerDAO implements ICustomerDAO
   {}
```

Vista compuesta

- La vista se compone de varios fragmentos de HTML y JSPs, para componer la vista final.

Ejemplo

// Esta página obligatoriamente tiene que ser un JSP.

```
<div id="contenedor">
        <div id="cabecera"><jsp:include page="cabecera.html"  /></div>
        <div id="izquierda"><jsp:include page="menu_izq.html"  /></div>
        <div id="centro"><jsp:include page="listado.jsp"</div>
        <div id="pie"><jsp:include page="pie.html"  /></div>
</div>
```

- Facilita el mantenimiento y la compartición de partes fijas de la Web.
- Por ejemplo, la cabecera, el menú y pie, pueden ser comunes a todas las páginas de la aplicación, antes cualquier modificación sólo se toca en el fragmento concreto y esto hará que se actualice en todas las páginas.

Ejemplo II

- En el pie.html tendríamos el siguiente fragmento:
  - Copyright 2010
- En la cabecera:
  -

Arquitectura **M**odelo **V**ista **C**ontrolador

- Aplicaciones Web de 3 capas.
- Patrón de diseño: MVC.
- El controlador.
- La Vista.
- El Modelo.

Aplicaciones Web de 3 capas

- Capa cliente:
  - Implementada por medio de páginas Web.
  - Captura datos del usuario y envía la información a la capa intermedia.
- Capa intermedia:
  - Es el núcleo, se encarga de recoger las peticiones de los usuarios. Genera y envía la información al usuario.
  - Esta a su vez interactúa con la capa de Datos, recuperando o grabando información.
- Capa de datos:
  - Normalmente representada por una base de datos.

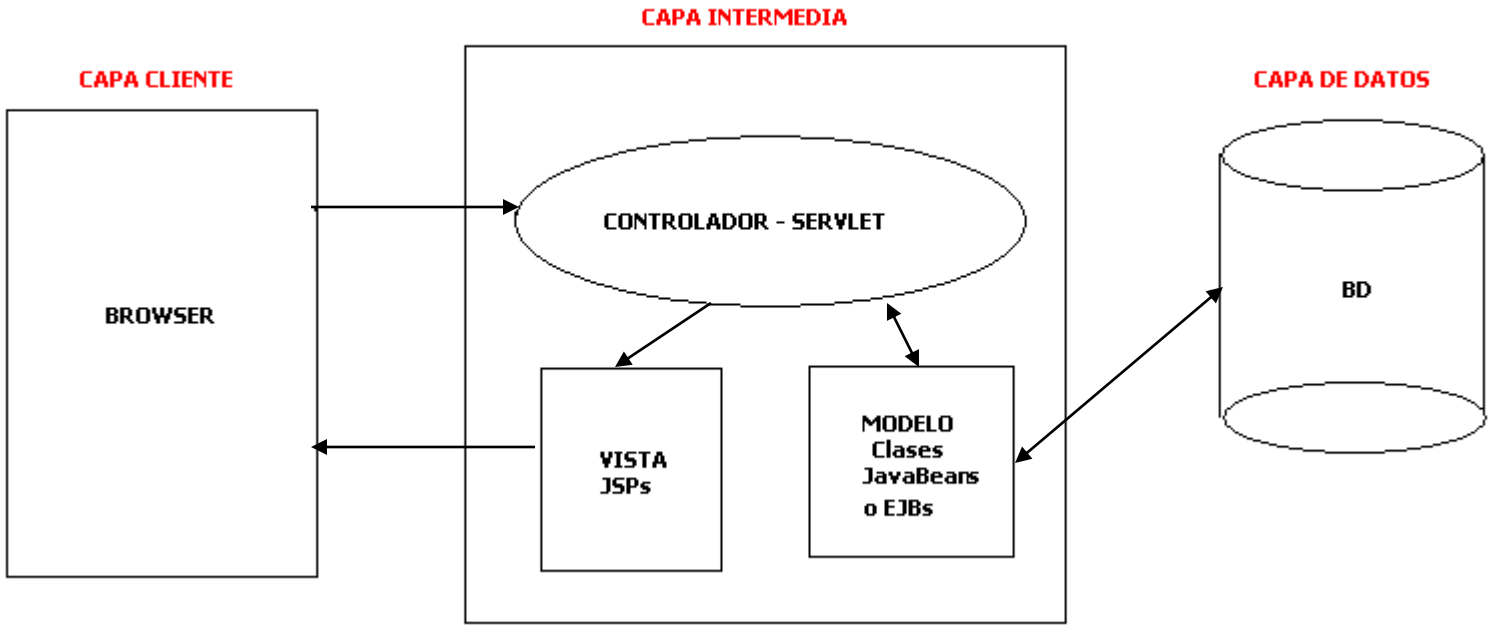
Patrón de diseño: MVC

- Este patrón de diseño se encarga de proporcionarnos como podemos organizar una aplicación Web, de forma que.
  - Las capas sean los mas independiente posible, cuanto mas desligadas entre sí estén mas fácil será el mantenimiento e integración.
  - La organización la centra en la capa intermedia (el núcleo de la aplicación Web).

Los bloques funcionales

- Serían tres: Controlador, Vista y Modelo.
- Cada uno tiene bien definida su responsabilidad.
- Dentro de una aplicación basada en tecnología J2EE.
  - El controlador estará representado por un Servlet.
  - La Vista serán páginas JSP.
  - El modelo se podrá implementar con Clases Ayudante (JavaBeans) o EJBs.

Esquema


CAPA INTERMEDIA
CAPA CLIENTE
CAPA DE DATOS
BROWSER
CONTROLADOR - SERVLET
VISTA
JSPs
MODELO
Clases
JavaBeans
o EJBs
BD

El Controlador

- Es el “cerebro” de la Aplicación.
  - En él se centran todas las peticiones por parte de la capa Cliente.
  - Sabe quien tiene que hacer cada cosa.
  - Se encarga de redirigir el flujo de las peticiones.
  - Si tiene que mostrar datos hará uso de la Vista (las páginas JSP).
  - Si tiene que extraer datos de la BD, llamará a una clase del modelo.

El Controlador

- Si la capa cliente solicita unos determinados datos:
  - El Controlador solicita los datos necesarios al modelo (es el encargado de interactuar con la BD).
  - Una vez que tiene los datos, se los envía a la vista para que se encargue de mostrárselos al Cliente.

¿Por qué un Servlet?

- Es una clase Java, es mas útil para gestionar el flujo de la programación, tiene estructura en el código, y es incómodo para mostrar datos.
- La vista utilizaremos páginas JSP mas parecidas a una página de HTML en la que es mas fácil dar formato a los datos que queremos mostrar.
- Se pueden separar mas los perfiles, programadores pueden desarrollar el Servlet y el modelo.
- Diseñadores se pueden centrar mas en la Vista.

El Controlador

- La centralización del flujo de peticiones en el Servlet, proporciona una serie de ventajas:
  - Desarrollo mas limpio y sencillo.
  - Facilita el posterior mantenimiento de la aplicación, es mas escalable.
  - Facilita la detección de errores.
- Puede darse el caso de que el Servlet se apoye en otros Servlets auxiliares, el Servlet principal recibe las peticiones y las redirige a otros Servlet auxiliares.

La Vista

- Se encarga de generar las respuestas, generalmente serán código xHTML que serán enviadas al Navegador.
- Si la respuesta a generar no es fija y procede de los datos obtenidos del Controlador, está se tendrá que generar de forma dinámica, aquí es donde entran las **páginas JSPs**.
- Si la información a devolver fuera estática se podría implementar directamente con páginas XHTML.

El Modelo

- Representa la lógica de Negocio de nuestra aplicación.
- Incluye el acceso a datos y su manipulación.
- El Modelo está representado por una serie de componentes independientes del Controlador y de la Vista, que permite la reutilización y desacoplamiento de las capas.
- Serán clases ayudante (JavaBeans) o EJBs.

Funcionamiento de la aplicación

- A partir de la petición del cliente, el controlador captura la petición:
  - Como todas las peticiones están centralizadas en el Servlet, la URL (con los parámetros) es la que nos va a determinar la operación a realizar. url?parametro=valor
  - También podemos codificar una serie de valores dentro de la propia URL. En este caso el path info nos determinaría la petición a realizar.
  - La url pattern del Servlet puede ser: *.do → (validar.do, registrar.do, etc.)
  - http://localhost:8080/aplicacion/url_control/info
    - La aplicación desplegada sería: aplicacion.
    - La url pattern del Servlet sería: url_control.
    - La operación podría ser: info

Funcionamiento de la aplicación II

- Procesar la petición:
  - Cuando el controlador determina la operación a realizar procede a ejecutar las acciones pertinentes.
  - Para ello invoca al modelo.
  - Para facilitar el intercambio de datos entre el Controlador y el Modelo se suelen diseñar clases ayundantes (JavaBeans).

- Generar la respuesta:
  - Los resultados devueltos por el Modelo al Controlador se depositan en una variable de petición, sesión o aplicación.