



UNIVERSIDAD DE CÓRDOBA

PROGRAMACIÓN WEB - JSP-I

J2EE/JSP, JDBC,
DAO, DTO, MVC

Dr. José Raúl Romero Salguero
jrromero@uco.es



JSP & Servlet

Contenidos

1. Instalación
2. Elementos de JSP
3. Objetos implícitos en JSP
4. Acceso a datos desde Java
5. Patrón Modelo-vista-controlador (MVC)
6. Recomendaciones de diseño

1.

Instalación

Primeros pasos



Servidor de aplicaciones Apache Tomcat

Servidor Apache Tomcat

- Implementación de **código abierto** del **servidor de aplicaciones** para tecnologías *Java Servlet*, *JavaServer Pages* (JSP), *Java Expression Language* y *Java WebSockets*
- Distribuido bajo **licencia Apache v2**, y mantenido por la comunidad JCP (*Java Community Process*)

<https://jcp.org/en/introduction/overview>

- En un entorno industrial – no académico (aprendizaje) – **es conveniente y recomendable tener siempre actualizado el servidor de aplicaciones** a su última versión: mejora de eficiencia, nuevos protocolos, prevención de ataques y riesgos de seguridad, etc.

<http://tomcat.apache.org/>

Servidor Apache Tomcat

Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	JASPIC Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.27	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.47	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.53 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x	7.0.96	6 and later (7 and later for WebSocket)
2.5	2.1	2.1	N/A	N/A	6.0.x (archived)	6.0.53 (archived)	5 and later
2.4	2.0	N/A	N/A	N/A	5.5.x (archived)	5.5.36 (archived)	1.4 and later
2.3	1.2	N/A	N/A	N/A	4.1.x (archived)	4.1.40 (archived)	1.3 and later
2.2	1.1	N/A	N/A	N/A	3.3.x (archived)	3.3.2 (archived)	1.1 and later

Versión UCO

Directorios de Apache Tomcat

- `/bin` – Ejecutables y scripts para lanzar el servidor y servicios
- `/common` – Clases y ficheros JAR (*Java Archive*) comunes a todo el servidor (librerías globales), si fuera necesario
- `/lib` – Ficheros JAR necesarios para la compilación e interpretación de servlets y JSPs
- `/logs` – Ficheros de trazas y seguimiento del servidor, errores, conexiones, etc.
- `/conf` – Ficheros de configuración del servidor
- `/webapps` – Aplicaciones web de Tomcat y de usuario:
 - Una aplicación por directorio o archivo WAR (*Web Application Archive*)
 - Se pueden crear páginas en el ROOT
- `/work` – Ficheros compilados de aplicaciones web

Directorios de WEBAPPS/

- `/nombreApp` – Raíz de la aplicación “nombreApp”
 - ❑ Se accederá a través de la URL: <http://localhost:8080/nombreApp>
 - ❑ Contiene ficheros JSP, html, css, js, imágenes, etc. → **debe estructurarse adecuadamente** en subdirectorios
 - ❑ Por defecto, lee el archivo **index.jsp**
- `/WEB-INF/web.xml` – Fichero de configuración de la aplicación web
 - ❑ Puede guardar **parámetros de configuración específicos** del despliegue de la aplicación (p.ej. ruta de fichero de propiedades)
- `/WEB-INF/classes` – Contiene las clases Java de la aplicación (lógica de negocio, acceso a datos, modelos (denominados *beans* en J2EE), etc.
 - ❑ Las clases **deben estar estructuradas en paquetes**
- `/WEB-INF/lib` – Ficheros JAR específicos de la aplicación web

Directorios de WEBAPPS/ - Recomendaciones 1

- Además del `/WEB-INF`, se recomienda estructurar adecuadamente el directorio de la aplicación web para alojar todos los ficheros html, js, css, imágenes y datos de forma directa y estructurada.
- Algunas **recomendaciones** de nomenclatura:
 - ❑ **Directorio raíz** de la aplicación – aloja la página de inicio (index.jsp) y otras páginas (no *includes* de la plantilla) triviales como acerca.html, etc.
 - ❑ `/css` – Hoja(s) de estilos
 - ❑ `/img` – Imágenes, fondos, figuras
 - ❑ `/js` – *Scripts* de Javascript
 - ❑ `/include` – Ficheros JSP correspondientes a plantillas parciales que se incluirán en 2 o más páginas (p.ej. control de acceso, cabecera, etc.) y plantillas auxiliares comunes (p.ej. página de error 404)

Directorios de WEBAPPS/ -

Recomendaciones 2

- ❑ `/mvc` – Inicio de la estructura de controladores y vistas del MVC
- ❑ `/mvc/control` – Estructura de subdirectorios para controladores
 - ❑ Se deben **organizar por activos**: p.ej. el subdirectorio `/cv` puede tener controladores referidos a añadir elementos en el CV, buscar en el CV, modificar CV, etc.
- ❑ `/mvc/view` – Estructura de subdirectorios para vistas
 - ❑ Debe ser la **misma estructura de subdirectorios que `/mvc/control`**, donde cada subdirectorio en `/view` se corresponde directamente con el del controlador referenciado
- Los **controladores** deberán nombrarse con la funcionalidad que realizan de la forma: `addItemCvController`, `userLoginController`, etc.
- Las **vistas** deben tener un nombre igual al del controlador pero especificando su acción. Por ejemplo: `userLoginViewFail`, `userLoginViewSuccess`

Ficheros de configuración

- `server.xml` – contiene la definición estructural del servidor, como nombre del host, servicios, conectores, directorios base, etc.
- `tomcat-users.xml` – define los roles de usuario de Tomcat, los nombres de usuario y sus password
 - ❑ No confundir con los usuarios de las aplicaciones, ya que estos roles y usuarios se refieren únicamente a los referidos a la administración del servidor de aplicaciones Apache Tomcat
- `web.xml` – contiene los valores por defecto a utilizar por todas las aplicaciones web cargadas en la instancia de Tomcat (p.ej. la página a cargar por defecto, páginas de error, etc.)

UCO - Apache *stand-alone*

- El servidor de aplicaciones **Tomcat 8.5** está instalado en la UCO en el siguiente directorio base:
`/opt/apache-tomcat-8.5.24`
- Para **arrancar el servidor**, debe ejecutarse el siguiente comando:
`/opt/apache-tomcat-8.5.24/bin/startup.sh`
- **La primera vez que se ejecute**, creará en el directorio *home* del usuario una **estructura de subdirectorios de Tomcat** en la ruta:
`/home/i82rosaj/.tomcat/`
- En esa estructura, se encuentran `/conf`, `/logs` y `/webapps`

UCO - Apache *standalone*

- Para desplegar una aplicación, se debe crear un subdirectorio en `~/tomcat/webapps/` conforme a la estructura requerida
- Se puede comprobar que el servidor está activo si se accede a la dirección web <http://localhost:8080/>
- Cualquier aplicación desplegada en webapps (p.ej. "miapp") será accesible desde la URL <http://localhost:8080/miapp>
- Para echar abajo el servidor de aplicaciones, se deberá ejecutar:
`/opt/apache-tomcat-8.5.24/bin/shutdown.sh`
 - ❑ En ocasiones, puede ser necesario reiniciarlo al recompilar la aplicación para que se regeneren los *servlets* por parte de Catalina

UCO - Integración en IDE Eclipse

- La UCO ofrece la versión de **Eclipse Luna integrada con Apache Tomcat 7.0** para poder desarrollar y ejecutar las aplicaciones web desde el propio IDE
- Obsérvese que **las instalaciones de Tomcat *stand-alone* y Tomcat+Eclipse son diferentes**, por lo no se usarán simultáneamente, ni guardan relación en cuanto a directorios y configuración de uno respecto al otro
 - ❑ El uso del IDE nos permite ayuda contextual y uso del color para facilitar la programación
- **IMPORTANTE:** Aunque se implemente desde un IDE como Eclipse, **deberá ser posible ejecutar las prácticas desde un Apache Tomcat 8.5 *standalone***, por lo que deberán hacerse las comprobaciones pertinentes a tal efecto

UCO - Integración en IDE Eclipse

- En Eclipse, abrir el menú "File" > "New" > "Other"
 - Elegir la opción "Server" situado en la carpeta "Server"
 - Crear un nuevo servidor de la siguiente forma:
 - ❑ En la carpeta "Apache" elegir "Tomcat v7.0 Server"
 - ❑ Introducir los campos siguientes:
 - Server's host name: localhost
 - Server name: Tomcat v7.0 Server
 - Server runtime environment: Apache Tomcat V7
 - ❑ Tras pulsar "Next" introducir los siguientes parámetros en "Tomcat Server" (después pulsar "Finish"):
 - Name: Tomcat v7.0 Server
 - Tomcat installation dorectory: /opt/apache-tomcat-7.0.41
 - JRE: Workbench defaut JRE

UCO - Integración en IDE Eclipse

- Con ello, ya está vinculado Eclipse a Tomcat v7.0
- Para **crear un nuevo proyecto**, abrir el menú "File" > "New" > "Other" y elegir "Dynamic Web Project" en la carpeta "Web"
 - Indicando el nombre y las opciones por defecto, se creará un proyecto con la estructura genérica de J2EE
 - **Nota:** Se debe permitir abrir (o abrirla el usuario) la perspectiva J2EE
- **Probar** que todo funciona con el siguiente ejemplo:
 - Sobre el proyecto, seleccionar "New" > "JSP file" (no marcar la casilla correspondiente a la plantilla de JSP – sin plantilla)
 - Escribir el siguiente código de prueba:

```
Son las <%= new java.util.Date() %><br/>  
Bienvenido mundo!!
```


UCO - Integración en IDE Eclipse

- Con ello, ya está creado el proyecto en Eclipse para Tomcat v7.0
- Para **ejecutar el proyecto**, sobre la carpeta del proyecto, seleccionar "Run as" > "Run on Server"
 - Deberá aparecer el servidor anteriormente configurado
 - Tras pasar al siguiente paso, se elige el servidor desplegado y "Finish"
- Se arrancará Tomcat y se abre una ventana en Eclipse para la ejecución de la aplicación. En el navegador, se podrá **probar con la URL**:
<http://localhost:8080/nombreApp>
- Se puede **exportar el proyecto** a un fichero WAR para poder –por ejemplo– traspasarlo al servidor *stand-alone*, donde copiaremos el WAR en el directorio /webapps y Tomcat lo desplegará automáticamente

Ciclo de vida de un JSP

- Los JSP no son ejecutados directamente por el servidor de aplicaciones, sino que son primero convertidos a un Servlet, que recibe las peticiones `HttpRequest` y genera las respuestas `HttpResponse`
- El *engine* (motor) de Tomcat encargado de traducir y generar un *servlet* equivalente al JSP se denomina **Catalina**
- Los pasos del **ciclo de vida de un JSP** son:
 1. Traducción del código JSP al fuente Java de un *servlet*
 2. Compilación del fuente del *servlet* a *bytecode*
 3. Carga de la clase `HttpServlet`
 4. Creación de la instancia del *servlet*
 5. Inicialización del *servlet* mediante invocación a `jspInit()`
 6. Procesamiento de la petición invocando a `_jspService()`
 7. Destrucción de la instancia invocando a `jspDestroy()`

```
<html>
  <head>
    <title>Hello world!</title>
  </head>
  <body>
    <% String nombre = "Mr. Romero"; %>
    Welcome <%= nombre %>
  </body>
</html>
```

helloworld.jsp

Ciclo de vida de un JSP

Se traduce a...

Es habitual que los controladores se desarrollen directamente como clases *servlet*

```
public class helloworld_jsp extends HttpServlet
{
    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.write("<html>[...]<body>");
        String nombre = "Mr. Romero";
        out.write("Welcome ");
        out.print(nombre);
        out.write("</body></html>");
    }
}
```

helloworld_jsp.java

2.

Elementos de JSP

Codificando las páginas de servidor



Código Java

Expresiones

- Una **expresión en JSP** se marca con `<%= ... %>` y equivale al contenido de una invocación a `system.out.println(expr)`

```
<%= "su nombre es " + usu.nombre %>
```


```
<%= Math.PI * 2 %>
```

```
<%= new java.util.Date() %>
```


- Escribe el resultado de la evaluación de la expresión *expr* en el documento

Scriptlets

- Un **scriptlet en JSP** se enmarca con `<% ... %>` y contiene una porción de longitud variable de Código en Java
- Puede invocar a librerías Java y código propio del proyecto (en `WEB-INF/`)



```
<table>
<% for (int i=0;i<10;i++) { %>
<tr><td> <%= i %> </td></tr>
<% } %>
</table>
```



```
<%
out.println("<table>");
for (int i=0;i<10;i++)
    out.println("<tr><td>"+i+"</td></tr>");
out.println("</table>");
%>
```

- Puede escribirse en el documento utilizando el objeto de flujo de salida **out**
 - ❑ Es recomendable **evitar su uso** en favor de expresiones
- Se recuerda la necesidad de **separar código Java y salida al navegador** en la mayor medida posible

Declaraciones

- Cualquier declaración de variables se puede realizar bien dentro de un *scriptlet*, como parte de más código, cuando se traten de declaraciones locales al propio *template*
- Se puede hacer un *bloque de declaración de JSP*, que se enmarca en los símbolos `<% ! ... %>`
 - ❑ Estas variables **son globales** y, por tanto, accesibles desde todos los JSP de la aplicación
 - ❑ Puesto que el servidor transforma la página JSP en un *servlet* (en el caso de Tomcat, lo hace el motor Catalina), y éste es usado por múltiples peticiones, las variables declaradas **conservan su valor entre sucesivas llamadas** o ejecuciones

Declaraciones

- ❑ La declaración **se ejecuta únicamente la primera vez** que se invoca al *servlet* equivalente al JSP
- El orden respecto a las expresiones es indiferente, no obstante se recomienda –por limpieza de código– **situar las declaraciones al inicio del JSP** para tenerlas siempre localizadas

```
<%! tipo1 variable1 = valor1; [tipo2 variable2 = valor2; ...] %>
```

Código HTML

```
<%! float precio = 0.23; int productId = 234; %>
```

Código HTML

Comentarios

- Un **comentario en JSP** se marca con `<%-- ... --%>` y equivale al comentario HTML con `<!-- ... -->`
- Dentro de un *scriptlet*, al tratarse de código Java, el comentario es igual que en Java

```
// Comentario de una línea
/*
    Comentarios de
    varias líneas
*/
```



Directivas

Directiva @page

- La directiva **@page** ofrece meta-información al contenedor de JSP acerca de la página (p.ej. lenguaje de *scripting*, página de error si procede, requisitos de almacenamiento, etc.)

```
<%@ page atributo="valor" %>
```

- Permite ofrecer hasta 13 atributos de información específicos. Entre los más utilizados destacan:
 - ❑ **Language**. Se refiere al lenguaje de programación utilizado en la página (*language="java"*)
 - ❑ **Import**. Indica a Catalina que importe otras clases e interfaces de Java durante la generación del código del *servlet*. Es la equivalente al *import* de Java y es, posiblemente, la directiva/atributo más utilizada

```
<%@page import="java.io.*, otroPaquete.otraClase"%>
```

Directiva @page

- ❑ **contentType**. Especifica el esquema de codificación de caracteres ("text/html; charset=ISO-8859-1" por defecto)
- ❑ **info**. Cadena de descripción del *servlet*, que podría ser accedida por el método `getServletInfo()`
- ❑ **session**. Indica si la página tendrá acceso a la sesión definida ("true", por defecto) o si no es necesario crear la sesión para esta página ("false")

```
<%@ page session="true/false"%>
```

- ❑ **autoFlush**. Indica si el *servlet* realizará de forma automática el flujo de salida ("true", por defecto) o si deberá hacerse de forma manual ("false"). En el último caso, el buffer de salida podría llenarse

Directiva @page

- ❑ **buffer**. Especifica el tamaño del buffer de salida (por defecto, "8KB") que utilizará el contenedor antes de redireccionar su contenido al objeto *response* (representa *HTTPResponse*)

```
<%@ page buffer="16KB"%>
```

- ❑ **isErrorPage**. Indica si el JSP va a gestionar errores de otras páginas (JSP) y se mostrará, cuando sea requerido, como página de error
 - ❑ Si el valor es "true", permitirá utilizar el objeto *exception*, que contiene una referencia a la excepción lanzada, en el fichero JSP
 - ❑ Si el valor es "false" (por defecto), no se podrá utilizar el objeto de excepción

Directiva @page

- ❑ **errorPage**. Establece el JSP que recibirá cualquier excepción lanzada por el JSP que declara la directiva
 - ❑ Cuando se produzca un error, se lanzará la excepción y se redireccionará a la página de error (definida con **isErrorPage**)

```
<%@page isErrorPage="true">
<html>
  <body>
    ¡Ha ocurrido un error!
    [...]
  </body>
</html>
```

error.jsp

```
<%@page errorPage="error.jsp">
[...]
```

cualquierJSP.jsp

Directiva @include

- La directiva `@include` copia el contenido literal del fichero incluido en el JSP

```
<%@ include file="url relativa" %>
```

- La **copia byte a byte** se realiza en el momento de compilación por Catalina
 - ❑ Si la **página incluida cambia después de la compilación**, los cambios no se verán reflejados → es **necesario re-compilar** la página que la contiene
- Se puede incluir otro JSP, HTML estático, elementos de *script*, directivas, acciones o cualquier fichero de texto convencional
 - ❑ Precaución con la inclusión de etiquetas o *scripts* duplicados
- Utilizado para la **inclusión de cabeceras, pies de página, barras laterales, etc.**



Acciones

Acciones de JSP

- Las **acciones JSP** permiten controlar el comportamiento del motor *servlet*
 - ❑ Se construyen conforme a la sintaxis de XML
 - ❑ Permiten insertar un fichero de forma dinámica, reutilizar componentes, redirigir a otra página, etc.
- Las acciones **se reevalúan cada vez que la página es accedida** – al contrario que las directivas
- Existen **11 tipos de acciones** (se estudian aquí las más relevantes) y siempre tienen la forma:

```
<jsp:action_name attribute="value" />
```

Acción `jsp:include`

- La acción `jsp:include` incluye la salida de otro JSP después de haber sido ejecutada
 - ❑ Nótese que `@include` incluye byte a byte el contenido, no la salida
 - ❑ La inclusión se realiza durante la el procesamiento del *HTTPRequest*
- El JSP incluido tiene acceso a los parámetros enviados a la principal, y también puede recibir parámetros enviados expresamente a la página

```
<jsp:include page="fichero.jsp" />
```

```
<jsp:param name="nombre" value="valor"/>
```

```
<html>
  <head>
    <jsp:include page="cabecera.jsp"/>
  </head>
  <body>
    <jsp:include page="cuerpo.jsp">
      <jsp:param name="prodId" value="320"/>
    </jsp:include>
  </body>
</html>
```

Acción `jsp:forward`

- La acción `jsp:forward` realiza una transferencia de control al JSP indicado, descartando el actual contenido del buffer de salida
- La página redireccionada tiene acceso a los parámetros de control del JSP invocante. Además, este puede declarar otros parámetros con la subetiqueta `jsp:param`

```
<jsp:forward page="fichero.jsp" />
```

```
<jsp:forward page="siguiente.jsp">  
  <jsp:param name="prodId" value="320"/>  
</jsp:forward>
```

- Los parámetros se reciben con el método `getParameter()` del objeto `request`

Acción `jsp:useBean`

- Un **componente JavaBeans** es una clase de Java, normalmente concebida para ser alojada en la sesión, y que tiene las siguientes características:
 - ❑ La clase **debe implementar la interfaz `java.io.Serializable`**
 - ❑ Su **constructor no tiene argumentos**
 - ❑ Ninguna de sus variables de instancia es pública, siendo sus propiedades **únicamente accesibles mediante métodos `get` y `set`**
- A este tipo de clases se les suele denominar **`xxBean`** (*`xxBean.java`*)
 - ❑ Un conocido ejemplo de bean es **`CustomerBean`**, que contiene la información del usuario registrado/logado en el sistema

```
<jsp:useBean id="name-path" class|type|beanName="bean-path"  
            scope="page | request | session | application" />
```

Acción `jsp:useBean`

- ❑ `id`. Indica el nombre del objeto *bean* para el JSP. El resto de código Java podrá referirse al *bean* utilizando este nombre de objeto
- ❑ `scope`. Especifica dónde se almacena el *bean*:
 - ❑ `"page"` indica que el ámbito es únicamente el de la página actual (por defecto)
 - ❑ `"request"` indica que el *bean* puede ser utilizado a partir de cualquier página JSP que procese la misma solicitud
 - ❑ `"session"` indica que el *bean* estará guardado en la sesión, tanto si se procesa la solicitud (*request*) como si no
 - ❑ `"application"` indica que el *bean* será accesible para cualquier aplicación del servidor

Acción `jsp:useBean`

- ❑ **class**. Nombre cualificado de la clase (esto es, incluyendo ruta de paquetes completa) que será instanciada para crear el objeto *bean*
- ❑ **type**. Indica el tipo de dato/clase del objeto *bean* en caso de que este ya exista en el ámbito – se diferencia con *class* en que *class* siempre instancia un objeto de la clase **xxBean**

```
<jsp:useBean id="customer"  
             class="es.uco.pw.display.beans.CustomerBean"  
             scope="session" />
```

- No conviene abusar del uso de *beans* (clases de la capa de interfaz de usuario), ya que ocupan memoria en el ámbito en que se han definido
- Una vez no se utilicen más, los *beans* **deben ser eliminados** de memoria por el **garbage collector**. Para ello, asigne el *bean* a **null**

Acción

jsp:useBean

Este *bean* podría tener otros tipos de datos más completos (p.ej. *arrays*) o métodos de chequeo (p.ej. `checkPassword()`)

```
package es.uco.pw.display.javabean;

import java.io.Serializable;

public class CustomerBean implements java.io.Serializable {

    private String idUser = "";
    private String idRol = "";
    private String sLogin = "";

    public CustomerBean () {      }

    public void setIdUser(String idUser) {
        this.idUser = idUser;    }

    public String getIdUser() {    return idUser;      }

    public void setIdRol(String idRol) {
        this.idRol = idRol;      }

    public String getIdRol() {    return idRol;      }

    public void setLogin (String login){
        this.sLogin = login;    }

    public String getLogin() {    return sLogin;      }

}
```


Acción `jsp:getProperty`

- La acción `jsp:getProperty` permite recuperar e imprimir el valor de una propiedad de una instancia de un *JavaBean*

```
<jsp:getProperty name="bean-name" property="property-name" />
```

```
<jsp:useBean id="persona" class="PersonBean" scope="request" />
<html>
  <head>
    <title>Información de tu cliente</title>
  </head>
  <body>
    La edad de <jsp:getProperty name="persona" property="nombre" /> es
    <jsp:getProperty name="persona" property="edad" />
  </body>
</html>
```

Acción `jsp:setProperty`

- La acción `jsp:setProperty` permite establecer el valor de una propiedad o a todas las propiedades simultáneamente (`property="*"`)

```
<jsp:setProperty name="bean-name" property="*|property-name" value="value"/>
```

```
<jsp:useBean id="persona" class="PersonBean" scope="request" />
<jsp:setProperty name="persona" property="edad" value="25" />
<html>
  <head>
    <title>Información de tu cliente</title>
  </head>
  <body>
    La edad es <jsp:getProperty name="persona" property="edad" />
  </body>
</html>
```

3.

Objetos implícitos

Objetos implícitos en JSP

- Los **objetos implícitos** son instancias creadas por el servidor de aplicaciones y pueden ser **accedidos como objetos** para facilitar el acceso a información de la aplicación y realizar acciones sobre la misma
 - Los objetos implícitos **se crean durante la fase de traducción** del JSP a *servlet*
 - Pueden ser directamente utilizados en los *scriptlets*
- Hay **9 tipos de objetos implícitos** disponibles: `out`, `request`, `response`, `config`, `application`, `sesión`, `pageContext`, `page`, `exception`

<https://docs.oracle.com/javaee/7/api/overview-summary.html>

Objetos implícitos en JSP

Objeto	Nombre cualificado de la clase
out	<code>javax.servlet.jsp.JspWriter</code>
request	<code>javax.servlet.http.HttpServletRequest</code>
response	<code>javax.servlet.http.HttpServletResponse</code>
config	<code>javax.servlet.ServletConfig</code>
application	<code>javax.servlet.ServletContext</code>
session	<code>javax.servlet.http.HttpSession</code>
pageContext	<code>javax.servlet.jsp.PageContext</code>
page	<code>Object</code>
exception	<code>Throwable</code>

Objeto implícito out

- El objeto `out` permite escribir datos en el buffer y enviarlos al flujo de salida del *servlet* para la respuesta al cliente
- Es equivalente al siguiente código (que sería necesario en caso de implementar un *servlet*):

```
[...]  
PrintWriter out=response.getWriter();  
[...]
```

Código servlet

```
1.<body>  
2.<%  
3.  int num1=10;  
4.  int num2=20;  
5.  out.println("num1 is " +num1);  
6.  out.println("num2 is "+num2);  
7.%>  
8.</body>
```

Objeto implícito `request`

- El objeto `request` es creado por el servidor para cada solicitud HTTP
- Se utiliza para obtener la información de la solicitud como parámetros de la llamada, información del *header*, nombre del servidor, etc.
- Se accede a los parámetros de la solicitud mediante el método `String getParameter(String)`, aunque este objeto implementa otros muchos métodos de interés, tanto para acceso a parámetros como información sobre el puerto, path de la URL, servidor, protocolo, si es protocolo seguro o no, etc.

<https://docs.oracle.com/javaee/7/api/javax/servlet/ServletRequest.html>

Objeto implícito request

```
<body>
  <form action="controlador.jsp">
    <input type="text" name="nombreUsuario">
    <input type="submit" value="submit">
  </form>
</body>
```

midocumento.html

```
[...]
<body>
<% // Inicio scriptlet
String nombre = request.getParameter("nombreUsuario");
out.println("Bienvenido <b>" + nombre + "</b>!!");
%>
</body>
```

controlador.jsp

Objeto implícito `response`

- Un objeto `response` es creado por el servidor para cada solicitud HTTP, representando la respuesta que puede dars al cliente
- El objeto permite especificar el tipo de contenido de la respuesta (*content type*), añadir cookies y redirigir a una página de respuesta
 - ❑ Permite `añadir o manipular` la cabecera de la salida, enviar errores, etc.

```
<%  
response.sendRedirect("http://www.jrromero.net");  
%>
```

Objeto implícito `config`

- El objeto `config` permite acceder a los parámetros de inicialización del *servlet* y a su contexto
 - ❑ Es creado para cada JSP/servlet
- **Muy utilizado** para obtener los parámetros de inicialización almacenados en el fichero `web.xml` para una determinada funcionalidad
 - **Deben diferenciarse las propiedades** de inicialización de la aplicación (almacenadas en `web.xml`) de las propiedades específicas de la aplicación (guardadas en ficheros de propiedades):

https://commons.apache.org/proper/commons-configuration/userguide/howto_properties.html

Objeto implícito config

```
[...]
<servlet>
  <servlet-name>importar</servlet-name>
  <jsp-file>/importarCV.jsp</jsp-file>

  <init-param>
    <param-name>DBdriver</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
  </init-param>
</servlet>
[...]
```

web.xml

controlador.jsp

```
<%
String driver = config.getInitParameter("DBdriver");
// Aquí se podría inicializar ...
out.print("driver: " + driver);
%>
```

Objeto implícito `application`

- Un objeto `application` por aplicación es creado por el servidor de aplicaciones cuando se despliega la aplicación
- Es un objeto de tipo `ServletContext`, que permite acceder al contexto y parametrización de la aplicación desplegada en el servidor
 - ❑ Estos parámetros pueden ser utilizados por todos los JSP
 - ❑ Toma los parámetros utilizados por todos los JSP, frente a `config`, que toma los parámetros de un *servlet* específico
- **Muy utilizado** para obtener los parámetros de inicialización almacenados en el fichero `web.xml` para toda la aplicación, como *drivers* de la base de datos, o rutas de ficheros generales

Objeto implícito `application`

```
[...]
<servlet>
  <servlet-name>importar</servlet-name>
  <jsp-file>/importarCV.jsp</jsp-file>
</servlet>
[...]
<context-param>
  <param-name>dname</param-name>
  <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
[...]
```

web.xml

controlador.jsp

```
<%
String driver = application.getInitParameter("DBdriver");
// Aquí se podría inicializar ...
out.print("driver: " + driver);
%>
```

Objeto implícito `session`

- El objeto `session` es utilizado para **consultar** (*get*), **asignar** (*set*) o **eliminar atributos** y valores del ámbito de sesión
- También permite obtener **informar de la propia sesión de navegación**
- **Recomendaciones** a tener en cuenta:
 - Tener **precaución con el uso de memoria de sesión**, evitando almacenar objetos demasiado pesados
 - **Vigilar el uso de estructuras dinámicas** que puedan saturar los recursos del cliente
 - **No es una buena práctica** almacenar directamente atributos de la lógica de negocio (p.ej. el login/rol del usuario) → se recomienda **utilizar JavaBeans** en su lugar para ello (independientemente de que estos beans se guarden en el ámbito de la sesión)

Objeto implícito session

Nota: este código es sólo un ejemplo y no sigue la estructura propia del MVC, ni emplea JavaBeans

```
<form action="loginController.jsp">
  <input type="text" name="username">
  <input type="submit" value="login"><br/>
</form>
```

1

index.jsp

```
<% // Se recoge info del usuario desde formulario
String usuario=request.getParameter("username");
// Se podrían establecer comprobaciones antes del logado
[...]
// Se guarda el nombre de usuario activo en sesión
session.setAttribute("login",usuario);
%>
// Se continúa navegando, si bien podría ser una c
<a href="miperfil.jsp">Continúa navegando...</a>
```

2

loginController.jsp

```
[...]
Está conectado como <%= session.getAttribute("login") %>
[...]
```

miperfil.jsp

3

Objeto implícito `pageContext`

- El objeto `pageContext` se utiliza para establecer, consultar o eliminar atributos de un ámbito: `page` (por defecto), `request`, `session`, `application`
- Habitualmente se accede al `pageContext` a través de las acciones equivalentes
 - ❑ No se utilizará en la práctica pero es interesante conocer la posibilidad de acceder al contexto directamente desde código Java

```
<% // Obtiene el nombre del usuario desde el request (p.ej. formulario)
String nombre=request.getParameter("usuario");
out.print("Hola "+nombre);
// Cambia el nombre del usuario logado al nombre leído como parámetro para
toda la sesión
pageContext.setAttribute("customer_name", nombre, PageContext.SESSION_SCOPE); %>
```


Objeto implícito `page`

- El objeto `page` mantiene una referencia de tipo `Object` al propio objeto *servlet* siendo ejecutado desde el JSP

```
Object page=this;
```

- Para ser utilizado, al tratarse de un `Object`, el objeto debe ser dotado de tipo a `HttpServlet` de la forma:

```
<% (HttpServlet)page.log("message"); %>
```

- Este código es equivalente a:

```
<% this.log("message"); %>
```

Objeto implícito `exception`

- El objeto `exception` representa a la excepción que maneja el JSP y, por tanto, permite acceder a la información del error que ha provocado el lanzamiento de la excepción
- Sólo puede ser **utilizado en páginas de error**

```
<%@ page isErrorPage="true" %>

<h3>Lamentamos el error!</h3>

Ha ocurrido el siguiente error:<br/>
<%= exception %>
```

error.jsp

```
// La página de error puede definirse
globalmente en el fichero web.xml
<%@ page errorPage="error.jsp" %>
<%
String p1=request.getParameter("num1");
String p2=request.getParameter("num2");

int a=Integer.parseInt(p1);
int b=Integer.parseInt(p2);
int c=a/b;
%>
El resultado es: <%= i %>
```

Si error



Programación Web

Presentación de la asignatura__ **Curso 2020/21**