
Aplicaciones web con Java

Raúl Caballero Ortega (rcortega@di.uc3m.es)
Álvaro Sánchez-Mariscal (mariscal@di.uc3m.es)

*Grupo de usuarios de
linux*

Univ. Carlos III de Madrid



3 de abril de 2003

Estructura básica de un Servlet

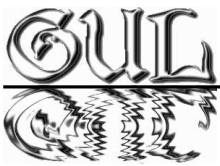
Para construir un Servlet deberemos:

1. Importar:

- *java.io.PrintWriter*
- *javax.servlet.**
- *javax.servlet.http.**

2. Extender *HttpServlet*

3. Implementar *doGet*, *doPost* o *doXxx*



Estructura básica de un Servlet (Ejemplo)

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

public class PrimerServlet extends HttpServlet{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException{
        PrintWriter out = response.getWriter();
        out.println("Hola Mundo");
    }
}
```

Generación de código HTML (I)

Para devolver código HTML deberemos seguir unos pasos:

1. Indicar qué vamos a devolver

```
response.setContentType("text/html");
```

2. Añadir las cabeceras que queramos

```
response.setHeader(String nombreCabecera,  
                    String valorCabecera);
```

3. Obtener el flujo de salida

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

4. Enviar la página

```
out.println("<html>");  
...  
out.println("</html>");
```

Generación de código HTML (y II)

Detalles a tener en cuenta al generar código HTML:

- Las cabeceras y el estado de respuesta deben indicarse antes del primer `out.println(...)`;
- “Conviene” que el documento de respuesta esté bien formado (aconsejable utilizar validadores o herramientas similares).

Ciclo de vida de un Servlet

1. **Creación:**

El servidor crea una única instancia del servlet y llama al método *init* del mismo.

2. **Uso:**

Cuando recibe una petición, el servidor arranca un nuevo hilo partiendo de la instancia ya creada y sobre ese hilo llama al método *service*.

3. **Destrucción:**

Cuando el servidor quiere eliminar la instancia del servlet llama al método *destroy* del mismo.

Método *init*

- Se ejecuta una sola vez al crear la instancia del Servlet.
- Según la configuración el Servlet se instanciará
 - al arrancar el servidor
 - al recibir la primera petición dirigida a ese servlet
- Es útil para establecer parámetros iniciales (conexiones JDBC, p.ej.).

Método *init* y los parámetros iniciales (I)

- Si no vamos a necesitar parámetros iniciales

```
public void init() throws ServletException{ ... }
```

- Si el Servlet necesita parámetros iniciales

```
public void init(ServletConfig config) throws ServletException{
    super.init(config);
    String param = config.getParameter("nombreDelParametro1");
    ...
}
```

NOTA: la llamada al *init* de la clase padre es obligatoria.

Método *init* y los parámetros iniciales (II)

- Parámetros iniciales en fichero de conf. del servidor.
- El formato de dicho fichero varía de servidor en servidor.
- Si hay muchos parámetros, lo mejor es
 1. Meter los parámetros en un fichero
 2. Poner como parámetro inicial el nombre del fichero

Ejemplo de fichero web.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
    <servlet>
        <servlet-name>MiServlet</servlet-name>
        <servlet-class>MiServlet</servlet-class>
        <init-param>
            <param-name>numero</param-name>
            <param-value>5</param-value>
        </init-param>
    </servlet>
</web-app>
```

Método *service*

Declaración:

```
public void service(HttpServletRequest request,  
                    HttpServletResponse response)  
    throws IOException, ServletException;
```

Cuando un servlet recibe una nueva petición

1. Crea un nuevo hilo
2. Ejecuta el método *service* de ese hilo
 - No conviene sobrescribir este método
 - Sobrescribir mejor el *doXxx* correspondiente

Método *doXxx*

Declaración:

```
public void doXxx(HttpServletRequest request,  
                  HttpServletResponse response)  
    throws IOException, ServletException;
```

Para invocar a uno de estos métodos

- El *service* analiza el tipo de petición
 - Si la petición es de tipo GET invocará a *doGet*
 - Si es de tipo POST invocará a *doPost*
 - Y así con los demás métodos (PUT, DELETE,...)

Método destroy

- Es invocado cuando se va a destruir la instancia
- Útil para resetear variables o guardar datos
- Conveniente cerrar conexiones JDBC que hubiera abiertas
- Tener en cuenta que no siempre se puede ejecutar

Compartición de variables

- Una sola instancia del Servlet, lo que implica:
 - Variables de instancia compartidas
 - Posible condición de carrera
- Posibles soluciones
 - Sincronizar el acceso concurrente a las variables
 - Implementar el interfaz *SingleThreadModel*

Depuración de un Servlet

A la hora de depurar nuestro código hay algunos buenos consejos:

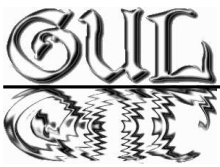
- Observar/validar el código generado
- Devolver código de error avisando (método *sendError* de `HttpServletResponse`)
- Arrancar el servidor desde una consola
- Escribir en logs (método *log* de `httpServlet`)
- Observar los datos de petición y de respuesta
- Si es necesario, reiniciar el servidor

Manejo de formularios: método *getParameter*

- Recibe un String indicando el nombre del parámetro
- Devuelve
 - Si el parámetro existe y tiene valor, devuelve el primero
 - Si el existe pero no tiene valor, devuelve una cadena vacía
 - Si no existe devuelve *null*
- Útil si sabes el nombre del parámetro
- Sólo devuelve un valor del parámetro

Manejo de formularios: método *getParameterValues*

- Recibe un String con el nombre del parámetro
- Devuelve
 - Un array de Strings con todos los valores del parámetro
 - Un array con un solo elemento (si solo tiene un valor)
 - *Null* si el parámetro no existe
- Devuelve todos los valores del parámetro
- Necesitas saber el nombre del parámetro



Manejo de formularios: método *getParameterNames*

- No recibe parámetros
- Devuelve un *Enumeration* con los nombres de los parámetros
- Los parámetros no vienen en ningún orden particular

Manejo de formularios: encabezados (I)

Los encabezados nos llegan entre la línea de petición y el cuerpo del documento

Hay métodos para recoger los más importantes:

- *getCookies* devuelve un array de Cookies (obj)
- *getAuthType* y *getRemoteUser* devuelven las dos partes del encabezado *Authorization*
- *getContentLength* devuelve la cabecera *Content-length* traducida a int
- *getContentType* devuelve el contenido de *Content-Type*

Manejo de formularios: encabezados (y II)

Para recoger cualquier otra cabecera hay métodos genéricos

- *getHeader* recibe un *String* con el nombre y devuelve otro con el valor
- *getDateHeader* como el anterior, pero devuelve un obj. *Date*
- *getIntHeader* como el anterior, pero devuelve un *int*
- *getHeaderNames* devuelve un *Enumeration* de *Objects* con los nombres de las cabeceras
- *getHeaders* devuelve un *Enumeration* con todos los valores de la cabecera pasada como parámetro

Manejo de formularios: línea de petición

Para conocer detalles sobre la petición recibida

- *getMethod* devuelve el método (GET, POST, PUT,...)
- *getRequestUri* devuelve el path al recurso solicitado
- *getProtocol* devuelve el protocolo (HTTP/1.0, HTTP/1.1, ...)

Generando respuestas

Una respuesta HTTP típica consta de

1. Línea de estado
2. Encabezados de la respuesta
3. Línea en blanco
4. Documento de respuesta

Ejemplo

```
HTTP/1.1 200 OK
```

```
Content-Type: text/plain
```

```
Hola a todos
```

El código de estado

- Es un número de tres cifras (XYZ)
- Cada una tiene su significado
- Significados de la primera (X)
 - 1 información que requiere una acción del usuario
 - 2 ok
 - 3 error temporal
 - 4 error en el cliente
 - 5 error en el servidor

Estableciendo el código de estado

- Se hace con el método *setStatus* pasándole un *int*
- Debe hacerse antes de imprimir el documento de respuesta
- Se pueden utilizar las constantes de *HttpServletResponse*
Ejemplo: `setStatus(response.SC_NOT_FOUND)` ;

Para estados especiales también hay:

- *sendError* recibe un *int* y un *String* (código y mensaje)
- *sendRedirect* recibe la URL (*String*) y devuelve un 302 con la URL

Estableciendo los encabezados de respuesta

- Para las cabeceras más frecuentes hay método específicos:
 - *setContentType* para el tipo MIME que vamos a devolver
 - *setContentLength* para indicar la longitud de la respuesta
 - *addCookie* para **añadir** cookies
- Para las demás tenemos:
 - *setHeader* recibe dos *Strings* nombre y valor de la cabecera
 - *setDateHeader* recibe el nombre (*String*) y fecha (*long*) en milisegundos
 - *setIntHeader* recibe el nombre y el valor en forma de *int*

Introducción a JSP

- *Java Server Pages.*
- Tecnología para desarrollar páginas web con contenido dinámico.
- Un fichero JSP puede contener etiquetas HTML normales, y elementos especiales para generar el contenido dinámico.
- Supone una evolución frente a la tecnología CGI, y los Servlets.
- Permite una separación en n capas de la arquitectura de la aplicación web.
- Se integra perfectamente con todas las API's empresariales de Java: JDBC, RMI (y CORBA), JNDI, EJB, JMS, JTA, ...

Ventajas respecto a otras tecnologías

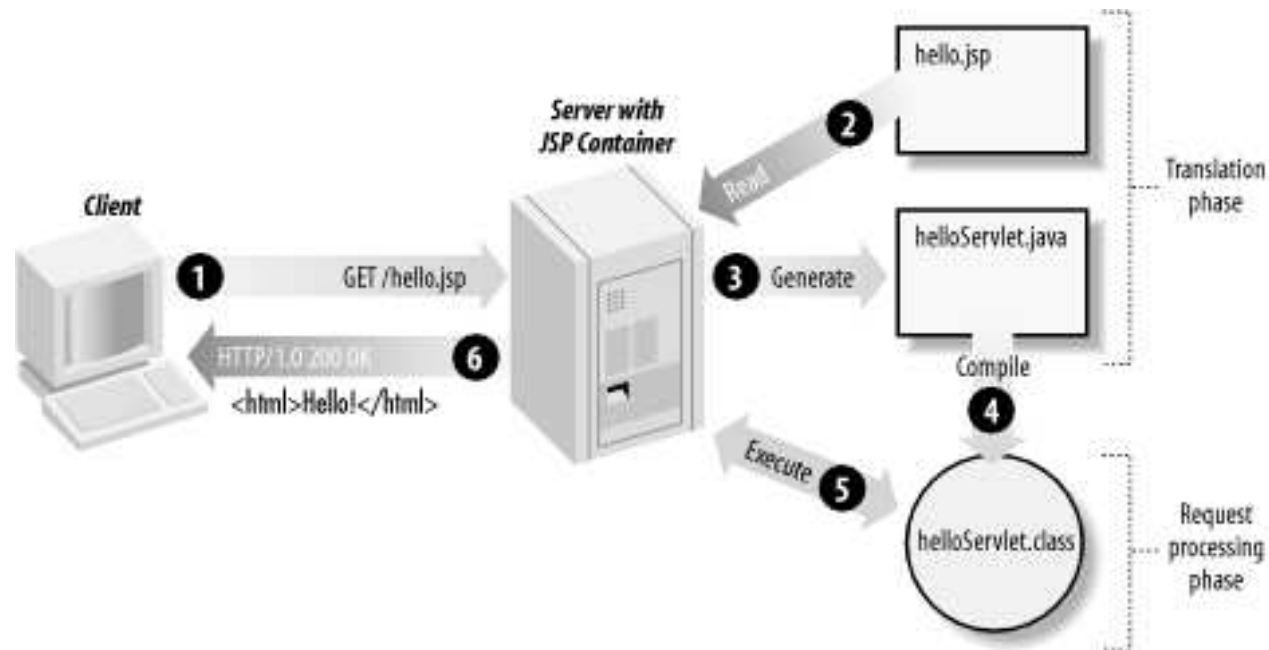
JSP tiene lo mejor de otras tecnologías como ASP, PHP, ColdFusion, etc:

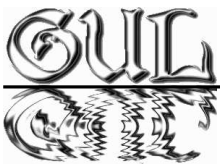
- JSP soporta tanto *scripting* como etiquetas para generar contenido dinámico. Permite desarrollar librerías de etiquetas personalizadas.
- Las páginas JSP son precompiladas para mejorar la eficiencia.
- Pueden usarse conjuntamente con Servlets para manejar la lógica de negocio.

Además, hay dos ventajas que únicamente tiene JSP:

- JSP es una especificación, no un producto. Esto se traduce en que se pueden crear distintas implementaciones y elegir la ofrezca más calidad y rendimiento.
- JSP es una parte integral de J2EE, la plataforma para aplicaciones empresariales más extendida.

Procesado de una página JSP





holamundo.jsp

```
<%@ page language="java" contentType="text/html" %>

<html>
  <head>
    <title>Hola, mundo!!</title>
  </head>
  <body>
    <h1>Hola, mundo!</h1>
    Hoy es <%= new java.util.Date() %>.
  </body>
</html>
```

Estructura de una página JSP

```

<%@ page language="java" contentType="text/html" %>
<html>
  <body bgcolor="white">

    <jsp:useBean
      id="userInfo"
      class="com.ora.jsp.beans.userInfo.UserInfoBean">
      <jsp:setProperty name="userInfo" property="*" />
    </jsp:useBean>

    The following information was saved:
    <ul>
      <li>User Name:

      <jsp:getProperty name="userInfo"
        property="userName" />

      <li>Email Address:

      <jsp:getProperty name="userInfo"
        property="emailAddr" />

    </ul>
  </body>
</html>

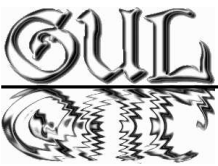
```

Diagram illustrating the structure of a JSP page, showing the sequence of elements and template text:

- JSP element: `<%@ page language="java" contentType="text/html" %>`
- template text: `<html>`
- template text: `<body bgcolor="white">`
- JSP element: `<jsp:useBean id="userInfo" class="com.ora.jsp.beans.userInfo.UserInfoBean">`
- JSP element: `<jsp:setProperty name="userInfo" property="*" />`
- JSP element: `</jsp:useBean>`
- template text: `The following information was saved:`
- template text: `<ul>`
- template text: `<li>User Name:`
- JSP element: `<jsp:getProperty name="userInfo" property="userName" />`
- template text: `<li>Email Address:`
- JSP element: `<jsp:getProperty name="userInfo" property="emailAddr" />`
- template text: `</ul>`
- template text: `</body>`
- template text: `</html>`

JavaBeans

- Son clases java normales y corrientes ...
- ... pero que están escritas siguiendo unas convenciones:
 - Constructor sin argumentos.
 - Implementa la interfaz `java.io.Serializable`.
 - Las propiedades o atributos se modifican mediante métodos *accesor*



PersonBean.java

```
public class PersonBean implements java.io.Serializable {
    private String fullName;
    private boolean teacher;
    public void setFullName(String fullName) {
        this.fullName = fullName;
    }
    public String getFullName(){
        return fullName;
    }
    public void setTeacher(boolean teacher) {
        this.teacher = teacher;
    }
    public boolean isTeacher() {
        return teacher;
    }
}
```

Generando contenido dinámico

- Usando JavaBeans:

```
<jsp:useBean id="user" class="es.uc3m.beans.PersonBean">
```

```
...
```

```
Nombre: <jsp:getProperty name="user" property="fullName" />
```

- Usando expresiones:

```
PI vale <%= Math.PI %>
```

- Usando *scriptlets*:

```
<% for (int i=0; i<3; i++) { %>
```

```
    Hola!!<br>
```

```
<% } %>
```

- Comentarios:

```
<!-- Comentario. No se incluye en la respuesta al cliente --%>
```

Objetos implícitos

- Son objetos que se encuentran definidos por defecto en todas las páginas JSP.
- **page**
 - `javax.servlet.jsp.HttpJspPage`.
 - Instancia del servlet de la página.
- **config**
 - `javax.servlet.ServletConfig`.
 - Datos de configuración del servlet.
- **request**
 - `javax.servlet.http.HttpServletRequest`.
 - Datos de la petición, incluyendo los parámetros.

■ response

- `javax.servlet.http.HttpServletResponse`.
- Datos de la respuesta.

■ out

- `javax.servlet.jsp.JspWriter`.
- Flujo de salida para el contenido de la página.

■ session

- `javax.servlet.http.HttpSession`.
- Datos específicos de la sesión de un usuario.

- **application**

- `javax.servlet.ServletContext`.
- Datos compartidos por todas las páginas de una aplicación.

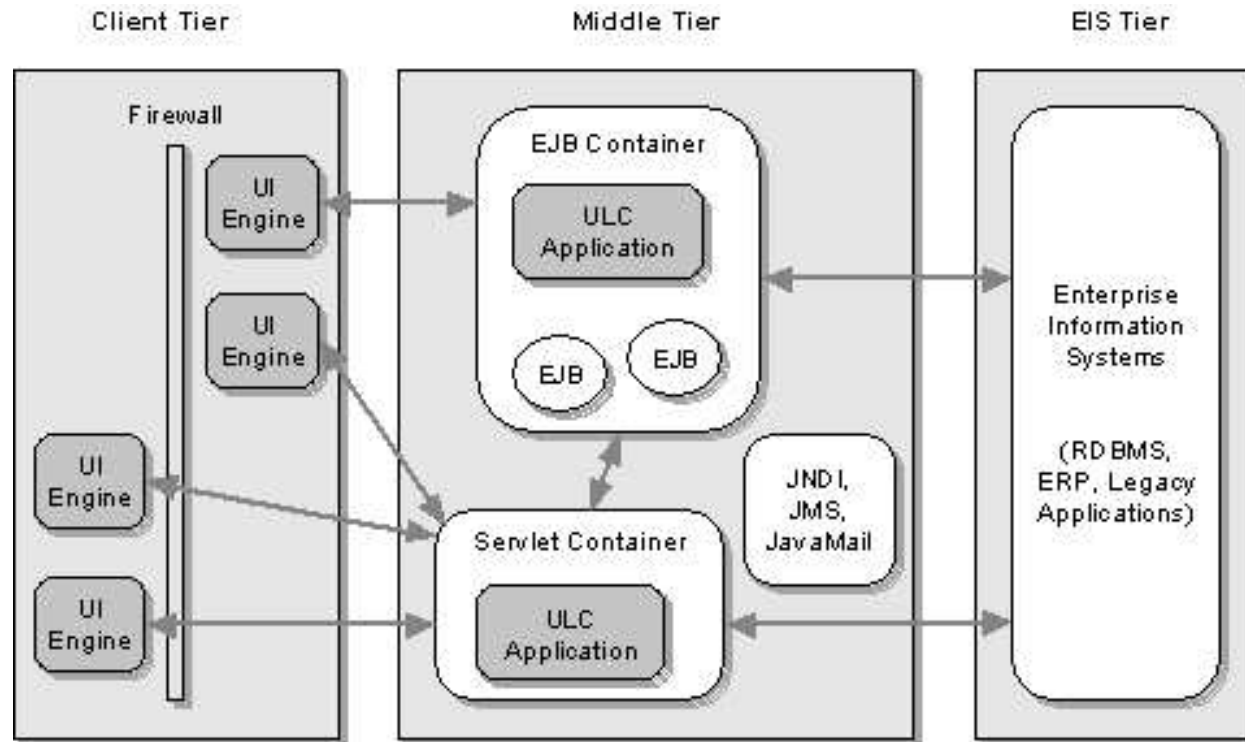
- **pageContext**

- `javax.servlet.jsp.PageContext`.
- Datos de contexto para la ejecución de la página.

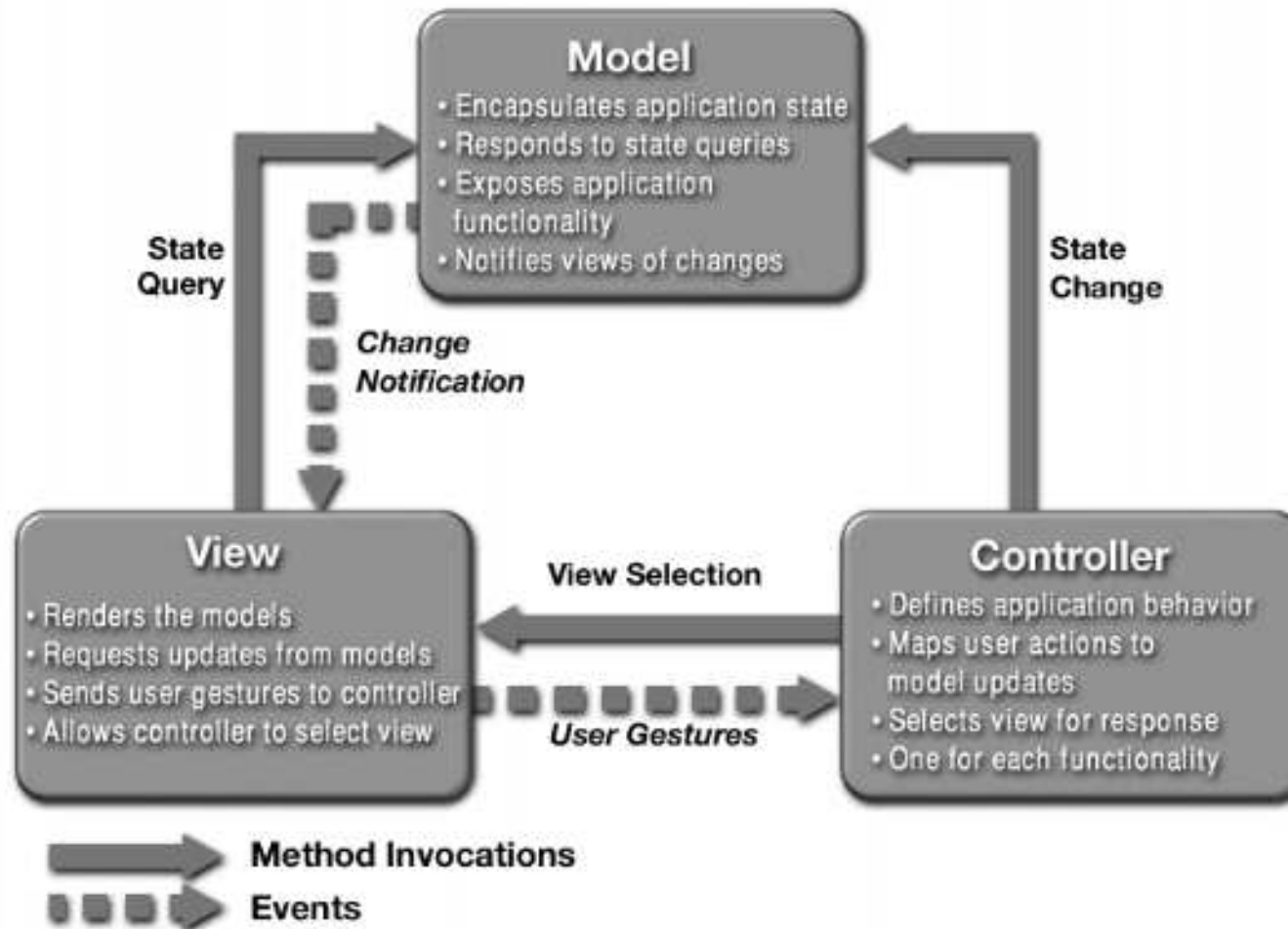
- **exception**

- `java.lang.Throwable`.
- Errores o excepciones no capturadas.

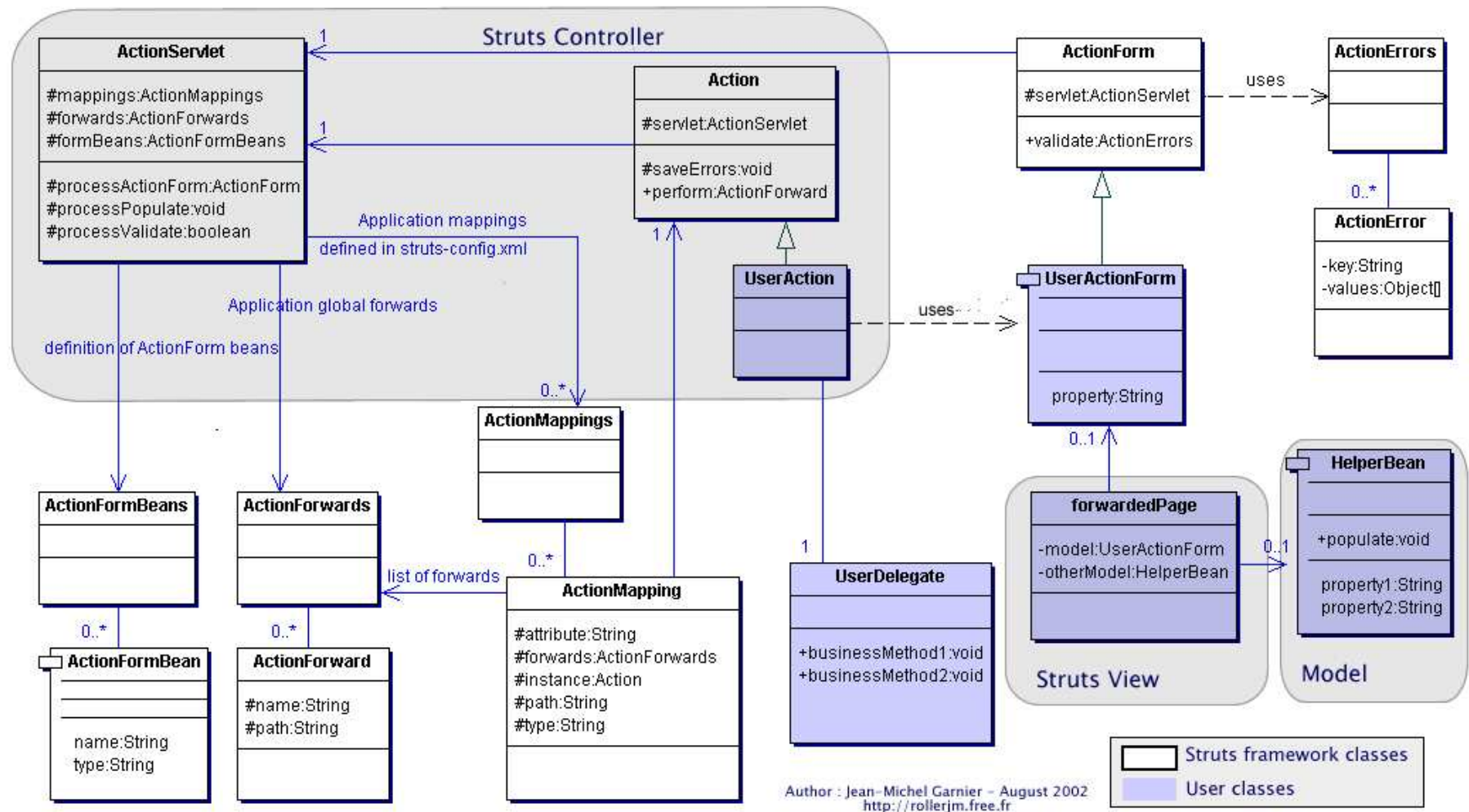
Java 2 Enterprise Edition Model

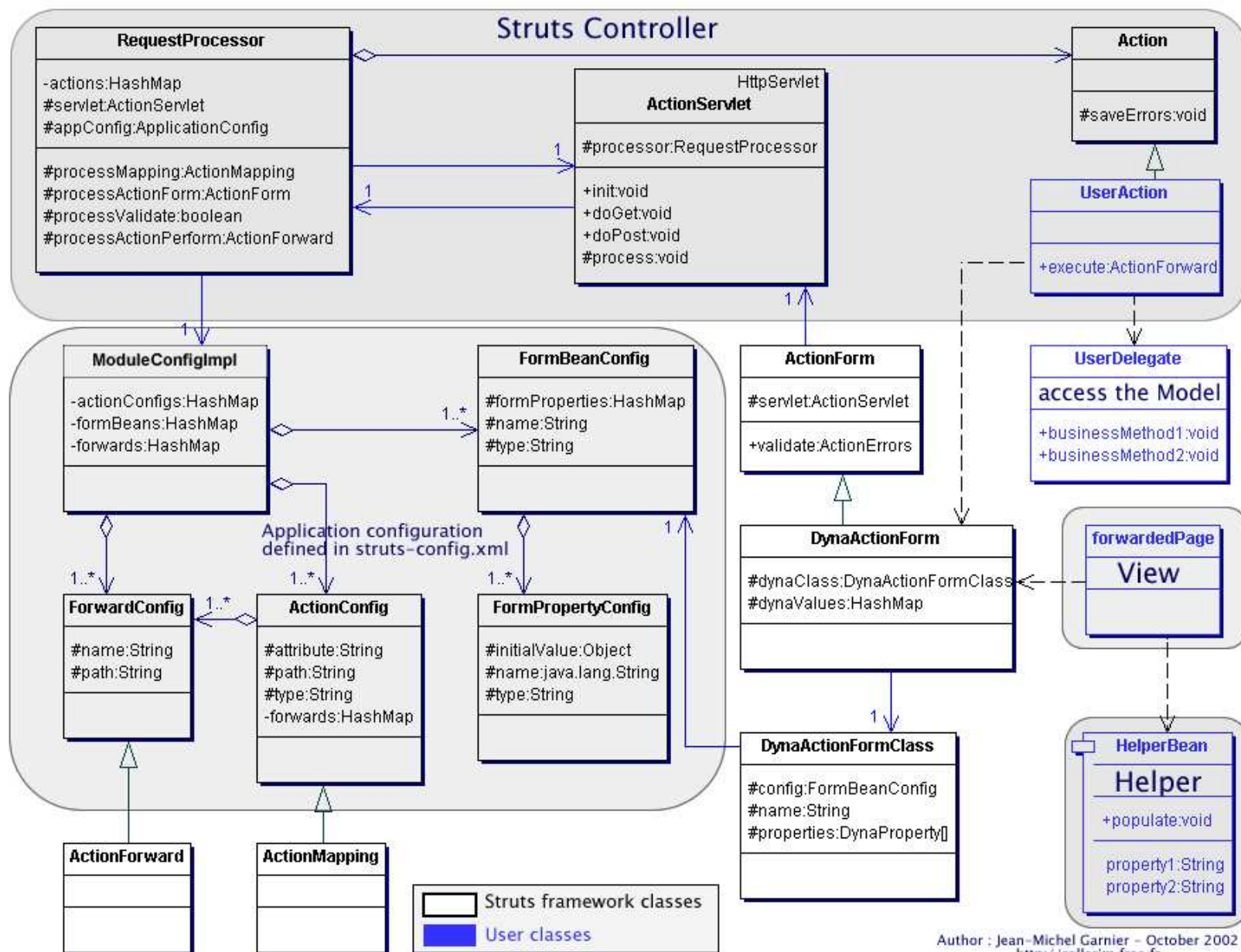


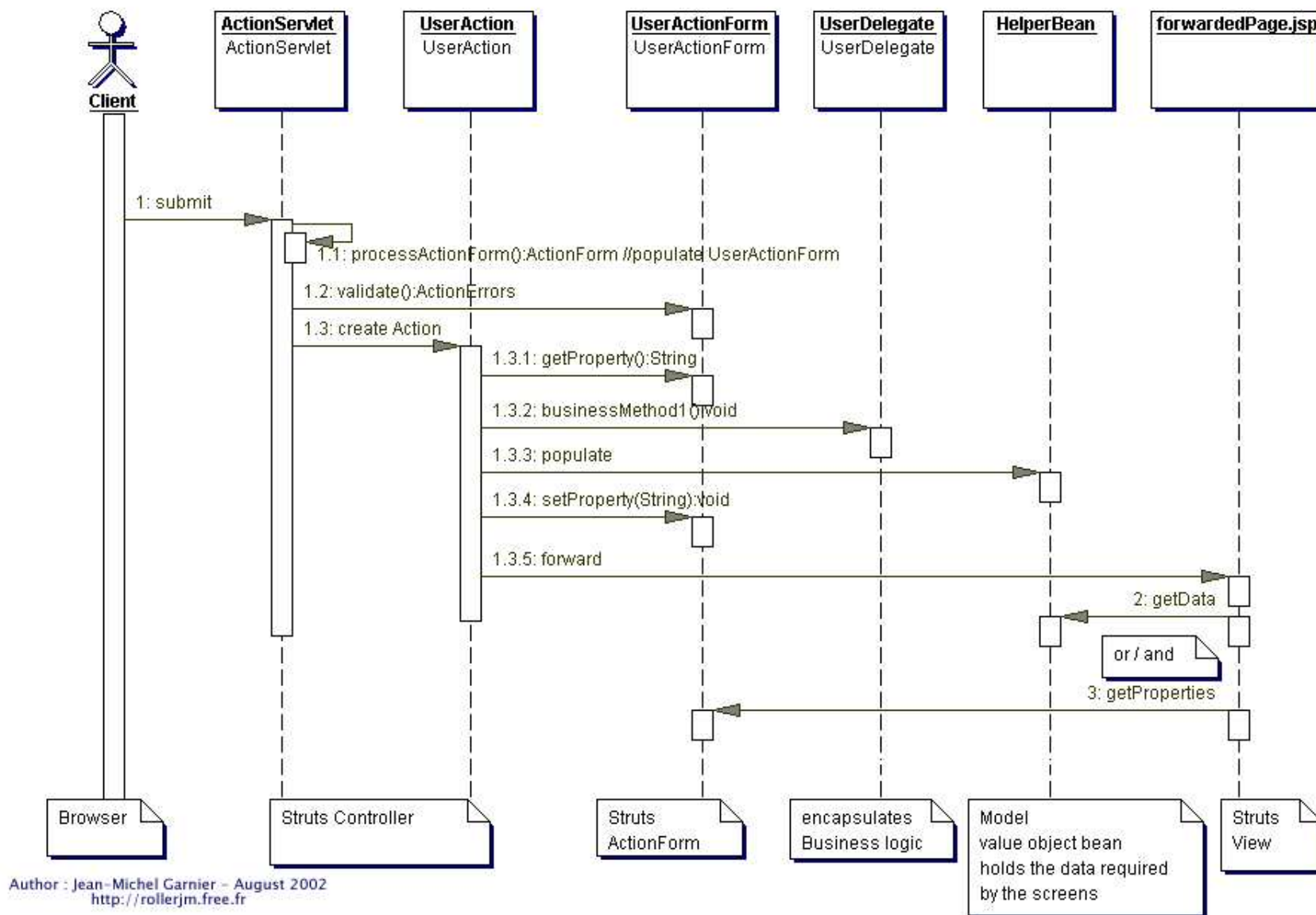
MVC: Modelo-Vista-Controlador



Struts, una implementación del patrón MVC









Dudas y preguntas

¿...?