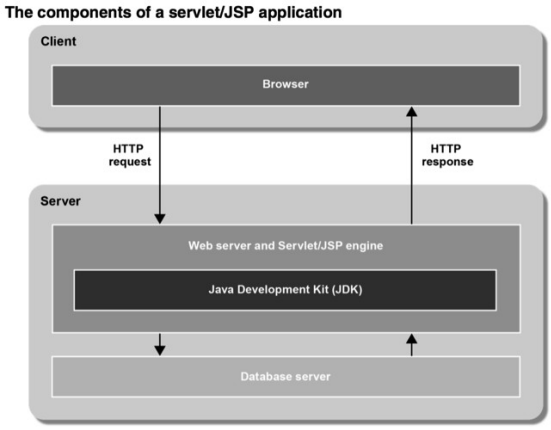
UT02_01: Introducción a servlets y otros conceptos.

Hoy en día, puede encontrar la tecnología Java en redes y dispositivos que comprenden desde Internet y superordenadores científicos hasta portátiles y teléfonos móviles, pasando por simuladores de mercado, juegos y tarjetas de crédito. Proporciona la posibilidad de crear soluciones empresariales seguras, independientes de plataforma, robustas y escalables. Está tecnología está soportada por una comunidad de usuarios que la enriquecen y mejoran abriendo posibilidades en: Desarrollar software en una plataforma y ejecutarlo en prácticamente cualquier otra plataforma.

- Desarrollar aplicaciones para servidores como foros en línea, tiendas, encuestas, formularios HTML…
- Desarrollar eficientes aplicaciones para teléfonos móviles, procesadores remotos, productos de consumo de bajo coste y prácticamente cualquier dispositivo digital.

Definiendo algunos conceptos:

JDK o SDK: Es el ambiente en el cual es posible desarrollar cualquier aplicación Java que incluye; el API de Java, el compilador de Java, así como el JVM (Java Virtual Machine) de la plataforma correspondiente.



JDBC: La tecnología JDBC (Java DataBase Connectivity) es una interface de programación de aplicaciones (API) que permite acceder, desde el lenguaje de programación Java, a virtualmente cualquier fuente de datos tabulados.

JSP (Java Server Pages): Este es un tipo de programa Java que contiene HTML, para ejecutar un JSP se requiere de un servlet engine como Tomcat. JSP separa la interface de usuario de la generación de contenidos, permitiendo cambiar el formato de la página sin alterar el contenido dinámico subyacente.

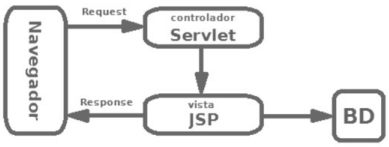
Servlet: Un servlet es una clase (como un applet) Java que se ejecuta en un servidor de aplicaciones. Antes de los servlets, la creación de contenidos dinámicos se hacía a través de CGI. Un servlet generalmente es utilizado para procesar solicitudes de usuarios, para procesos de verificación de usuarios, generación de contenido dinámico. Similar a un JSP, un JSP se convierte eventualmente en un Servlet, la diferencia es que un Servlet solo contiene lenguaje Java desde el inicio, mientras que un JSP contiene Java y HTML.

En resumen:

✓JSP: facilita la presentación de los documentos. Su objetivo es estructurar la interfaz de usuario, de una forma clara y ordenada.

✓Servlet: son programas que se ejecutan en el lado del servidor para generar páginas Web. Provee de una mayor flexibilidad en la programación de la lógica de negocio.

Modelo vista controlador JSP y Servlet:



Ciclo de vida de un servlet.

1. Instalación e inicialización (en la primera petición)

– si no existen instancias del servlet, el contenedor web:

• carga la clase del servlet

• crea una instancia

• inicializa la instancia del servlet llamando a `init.` Simplemente crea o carga datos que serán usados durante la vida del servlet.

```
public void init() throws ServletException {
  // Initialization code...
}
```

2. Manejo de sucesivas peticiones

– el contenedor crea un hilo que llama al método `service` de la instancia

– el método `service` determina lo que ha llegado en la petición (GET, POST, PUT,..) y llama a un método apropiado(doGET, doPUT, ..)

```
public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException {
```

3 Destrucción

– cuando el contenedor decide destruir el servlet, llama a su método `destroy.`

Limpiamos, desconectamos de las bases de datos, escribe la lista de cookies, etc.

```
public void destroy() {
    // Finalization code...
```

Con todo esto conseguimos utilizar

1. una *única máquina virtual* y por tanto compartición de datos entre varias instancias

2. Persistencia (en memoria) de las instancias

– consumo de memoria reducido

– eliminación de los tiempos de inicialización e instanciación

– persistencia (en memoria) del estado, los datos y los recursos: atributos persistentes del servlet y conexiones a bases de datos persistentes, etc

– persistencia (en memoria) de los hilos

API de Servlets

- Paquetes
  - `javax.servlet`
- 7 interfaces
  - `Servlet`
  - `ServletConfig`
  - `ServletContext`
  - `ServletRequest`
  - `ServletResponse`
  - `SingleThreadModel`
  - `RequestDispatcher`
- 3 clases
  - `GenericServlet`
  - `ServletInputStream`
  - `ServletOutputStream`
- 2 clases de excepciones
  - `ServletException`
  - `UnavailableException`

Para más información ver el tutorial de la UC3 (EVAGD).

Tareas de los servlets (1/2)

1. Leer datos enviados por el usuario
   - Típicamente través de un formulario HTML
   - Pero también desde un applet o aplicación cliente
2. Recuperar otra información de usuario embebida en la petición HTTP
   - Capacidades del navegador,
   - cookies,
   - nombre de la máquina del cliente, etc.
3. Generar resultados
   - Cálculo directo de la respuesta,
   - llamando a otro servidor (posiblemente remoto vía RMI o CORBA)
   - accediendo a una base de datos, etc.

Tareas de los servlets (2/2)

4. Formatear los resultados
   - En un documento HTML
5. Asignar los parámetros de la respuesta HTTP
   - Tipo de documento devuelto (HTML)
   - Cookies
   - Parámetros de cache.
6. Enviar el documento al cliente
   - En formato texto (e.g.HTML),
   - Formato binario (e.g. GIF)
   - Comprimido (e.g. gzip)

Antes de seguir con los ejemplos básicos. Nos fijamos de nuevo en nuestras variables de entorno. Ahora vamos a añadir las correspondientes a Tomcat.

C:\apache-tomcat-8.0.28 windows ----> CATALINA_HOME

/usr/local/apache-tomcat-8.0.289 Linux/Unix --->CATALINA_HOME

CLASSPATH:

```
set CATALINA = C:\apache-tomcat-8.0.28
set CLASSPATH = %CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%

setenv CATALINA = /usr/local/apache-tomcat-8.0.28
setenv CLASSPATH $CATALINA/common/lib/servlet-api.jar:$CLASSPATH
```

NOTA: No te olvides de añadir tu directorio de desarrollo de Servlets al classpath

Ejemplos

De momento los ejemplos los hacemos desde linea de comandos. Como siempre tendremos que tener nuestras variables de entorno y permisos en perfectas condiciones. Incluyendo nuestra JAVA_HOME, y classpath.

Ejemplo 1.-

Los servlets son clases Java que dan servicio a las peticiones HTTP e implementan la interfaz javax.servlet.Servlet . Los desarrolladores de aplicaciones Web suelen escribir servlets que extienden javax.servlet.http.HttpServlet, una clase abstracta que implementa la interfaz Servlet y está especialmente diseñado para manejar peticiones HTTP.

A continuación se muestra la estructura de código fuente de ejemplo de un ejemplo de servlet para mostrar Hello World

Compilamos con `javac HelloWorld.java`
En tu classpath debería estar también tu directorio de SERVLETS y tu servlet-api.jar.
Nuestros servlets siempre deberán estar en `<Tomcat-installationdirectory>/webapps/ROOT` y las .class en `<Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes`
Recuerda que si tienes un .class que está en un paquete de tipo com.myorg.MyServlet , entonces la clase del servlet debe localizarse en `WEB-INF/classes/com/myorg/MyServlet/`

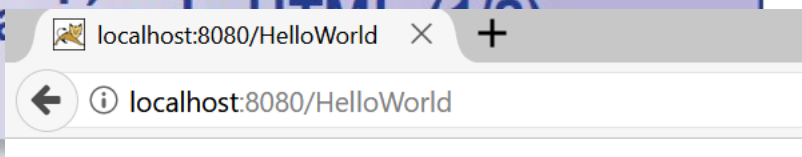
Por ahora vamos simplemente a copiar nuestro HelloWorld.class directamente a (el directorio no existe tienes que crearlo) `<TomcatInstallationdirectory>/webapps/ROOT/WEB-INF/classes`
Ya estamos casi. Ahora nos falta crear nuestro web.xml. Búscalo en `<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/`. Como este directorio está protegido no te olvides de dar permisos a tu usuario para que pueda escribir en todo el directorio Apache. Haz también una copia de tu `web.xml` original. Y coloca el siguiente texto entre <web-app>….y </web-app>

```
<servlet>
   <servlet-name>HelloWorld</servlet-name>
   <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloWorld</servlet-name>
   <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Y listo, activa el Tomcat y conectate a tu página.

Ejemplo 2: Genera

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

public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String docType =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" "
    + "\"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd\" >\n";
      out.println(docType +
                  "<HTML>\n" +
                  "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                  "<BODY>\n" +
                  "<H1>Hello WWW</H1>\n" +
                  "</BODY></HTML>");
  }
}
```



Hello World

Para el ejemplo 3. Mira e intenta entender que hace este código.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

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

   // Set response content type
   response.setContentType("text/html");

   PrintWriter out = response.getWriter();
   String title = "Using GET Method to Read Form Data";
   String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

   out.println(docType +
     "<html>\n" +
       "<head><title>" + title + "</title></head>\n" +
       "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<ul>\n" +
           "  <li><b>First Name</b>: "
           + request.getParameter("first_name") + "\n" +
           "  <li><b>Last Name</b>: "
           + request.getParameter("last_name") + "\n" +
         "</ul>\n" +
       "</body>
     </html>"
   );
  }
}
```

http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI

Compilamos:

```
$ javac HelloForm.java
```

HelloForm.class ------><Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes Y en **web.xml**

```
<servlet>
   <servlet-name>HelloForm</servlet-name>
   <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloForm</servlet-name>
   <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Ahora lo hacemos utilizando html Forms

GET Method Example Using Form

Creamos fichero Hello.html

```
<html>
   <body>
      <form action = "HelloForm" method = "GET">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Hello.htm lo ponemos en el directorio <Tomcat-installationdirectory>/webapps/ROOT. Accedemos a *http://localhost:8080/Hello.htm* y obtenemos algo parecido a lo de antes.

Igual que anteriormente, ahora añadimos en nuestro java, lo necesario para usar el método doPOST

```
 // Method to handle POST method request.
   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
```

Y cambiamos nuestra web:

```
<html>
   <body>
      <form action = "HelloForm" method = "POST">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Comprueba el resultado.