Introducción a los servlets

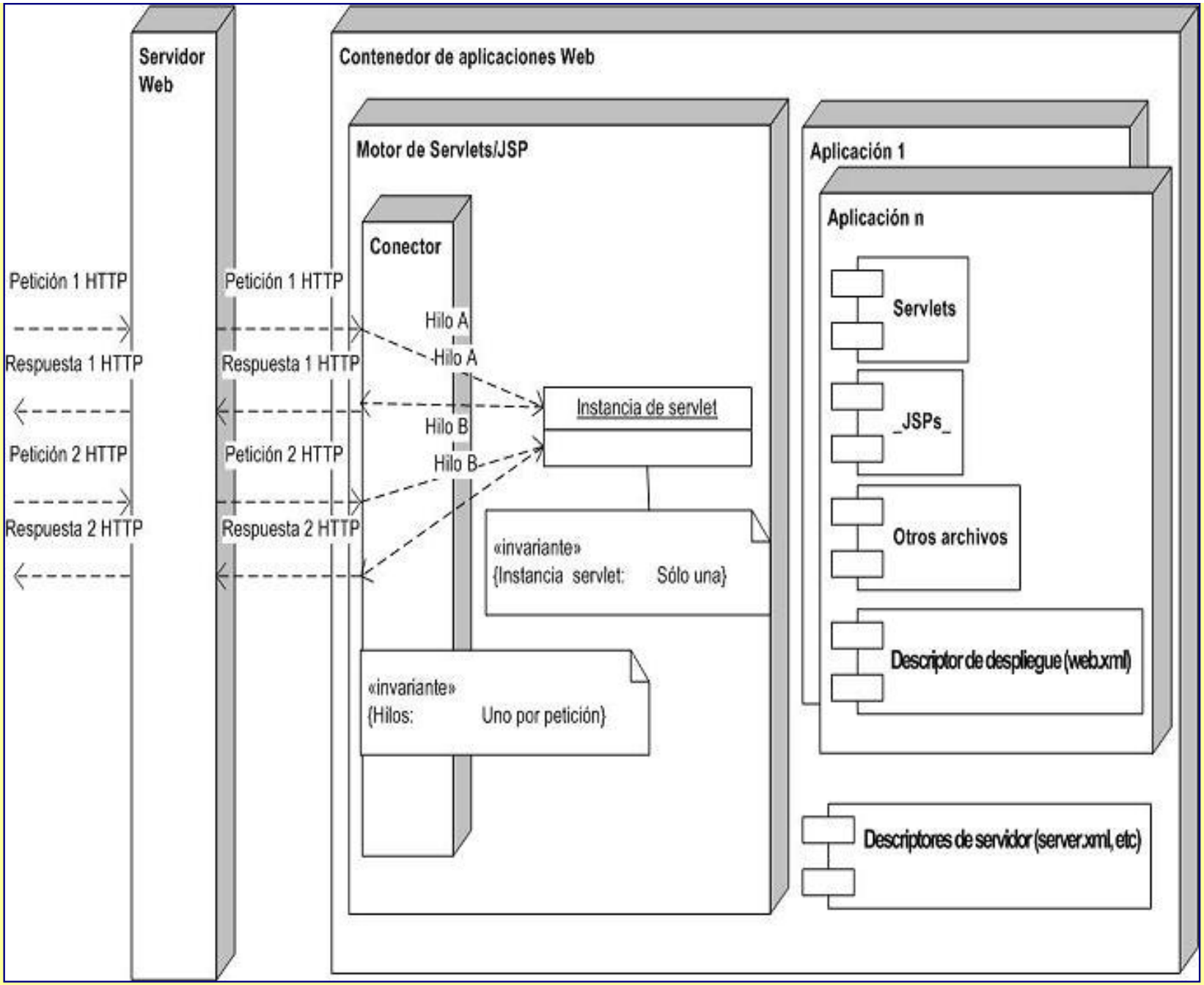
(Febrero de 2005)

Introducción

En breve: un servlet es un programa ejecutado en el servidor (a diferencia de los applets que se ejecutan en el cliente). Es un mecanismo para implantar aplicaciones en el lado del servidor. Reciben peticiones y mandan resultados en HTTP, siendo el formato más común de salida una página HTML o un archivo XML (pero puede ser cualquier tipo MIME, una imagen, un objeto serializado, etc).

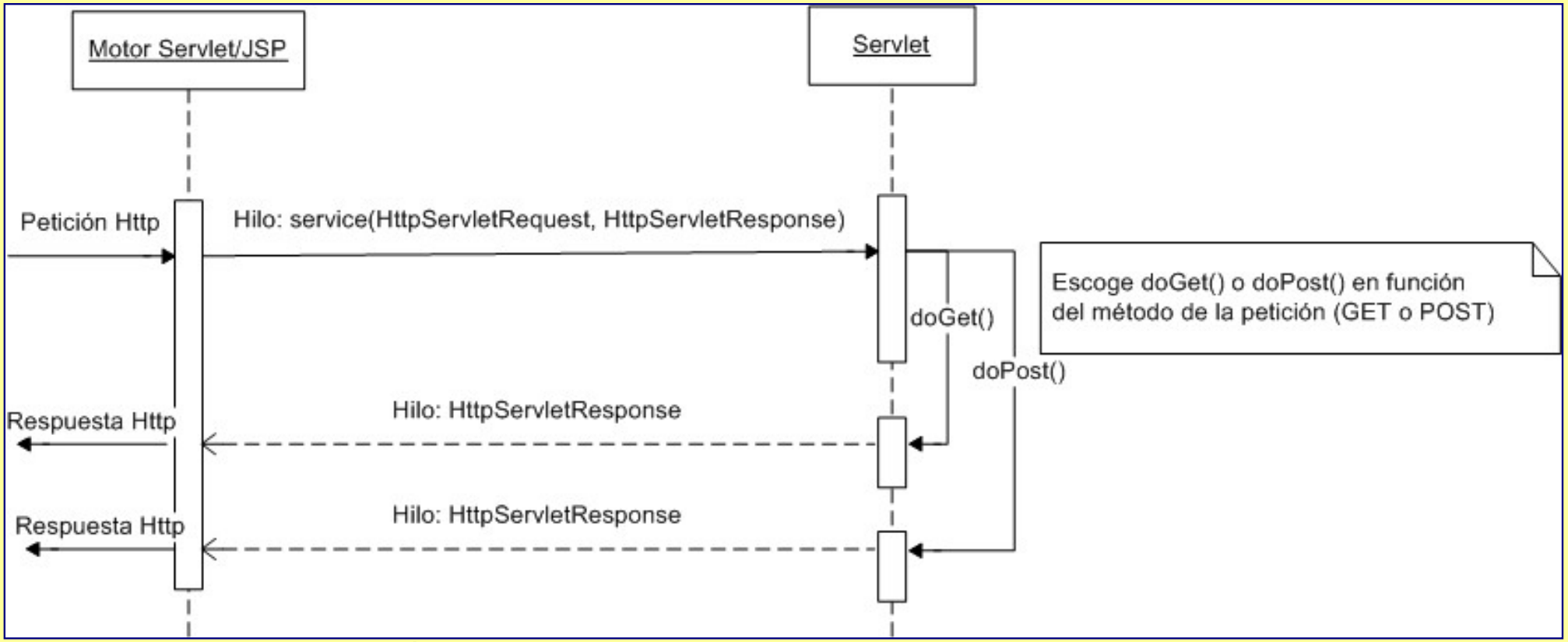
Antes de aprender sobre servlets conviene entender lo que es un servidor de aplicaciones: Capítulo de Arquitectura JEE - Servidor de aplicaciones.

Un primer ejemplo de servlet: Capítulo de Arquitectura JEE - Servlets.



El **proceso de ejecución de un servlet**:

1. El proceso comienza con la **petición HTTP** que llega en primer lugar al servidor web (ejemplo Apache).
2. La petición **se traslada al Contenedor de aplicaciones**, en concreto a su motor del servicio Servlet/JSP (con su propia JVM).
3. El motor encapsula la información de la petición en un objeto del tipo **HttpServletRequest**, además encapsula en un objeto **HttpServletResponse** el flujo de respuesta.
4. El motor crea **por cada petición un hilo**, sobre el que se invoca a la función **service()** del servlet. En función del método de la petición (POST o GET ...), service() llamará al método correspondiente del servlet: **doPost(), doGet() ...**, pasándoles los objetos de HttpServletRequest y HttpServletResponse. En realidad hay más métodos HTTP (trace, etc.), pero GET y POST son los más habituales.



5. Cada clase del tipo servlet, tiene **una única instancia**, sobre la que corren los diferentes hilos (peticiones).

Cada petición **genera un hilo independiente**. Pero sólo la primera petición genera **una instancia de la clase servlet**. Podremos tener N peticiones al servlet, por tanto N hilos, pero únicamente **una instancia** de la clase (un objeto).

El esquema es muy parecido al que existe en el CGI (Common Gateway Interface). Ventajas de los servlets:

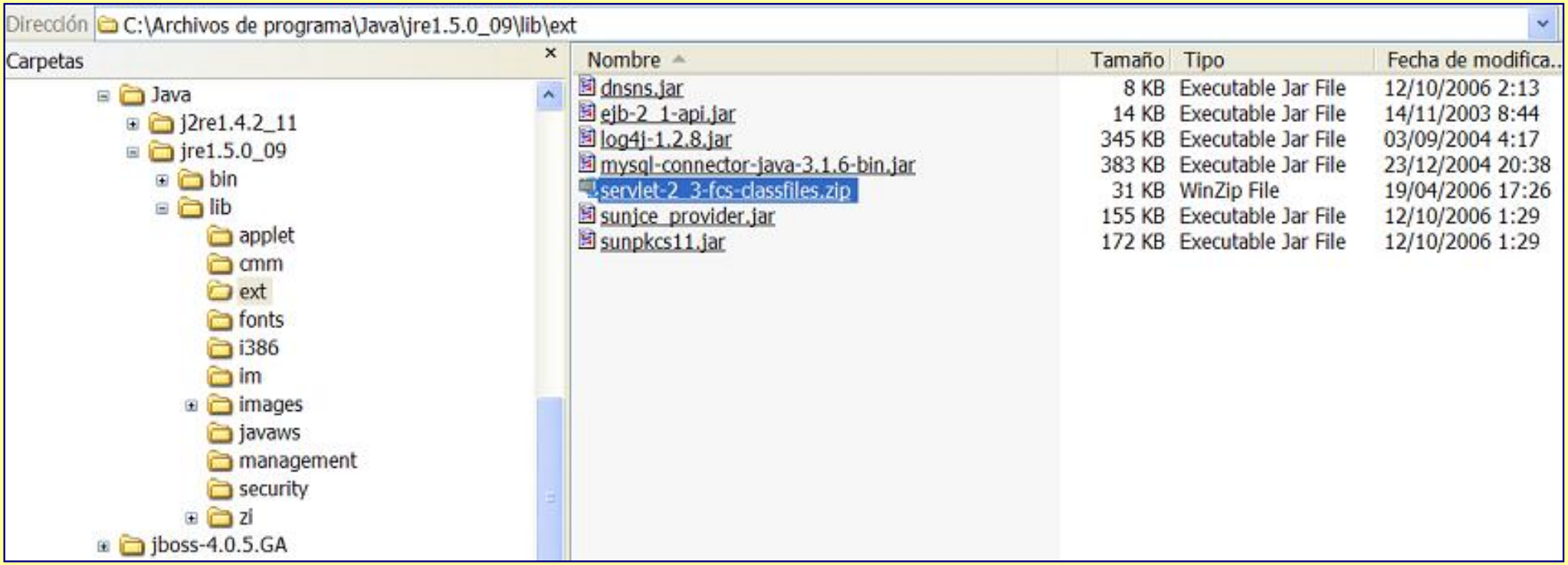
- Multiplataforma (es Java)
- Cada petición genera un hilo dentro de un proceso. Frente a la arquitectura CGI, donde cada petición genera un proceso. Existe el interfaz SingleThreadModel para indicar que se traten las peticiones de forma secuencial (un hilo).

- Pueden programarse de tal forma que se ejecuten en lo que en Java se conoce como "sand box", es decir, una zona de memoria con restricciones de seguridad para no acceder a determinados espacios de archivos del servidor o determinadas aplicaciones.

¿Qué necesitamos?

Para la instalación de un servidor de aplicaciones como Tomcat: Capítulo de Herramientas - Tomcat.

El API para programar Servlets no forma parte del núcleo JDK (Standard Edition) y hay que descargarlo (a menos que use la Enterprise Edition o que haya sido instalado por el entorno de programación que este utilizando). Viene suministrado también por Tomcat en el directorio common/lib. En el siguiente ejemplo lo hemos descargado de (SUN) y lo hemos puesto **en el directorio ext del JRE, con la ventaja de que lo reconoce de forma automática el IDE Eclipse**:



A la hora de programar los paquetes necesarios son:

```
javax.servlet
javax.servlet.http
```

El primer servlet

Los servlets HTTP derivan de la clase **javax.servlet.http.HttpServlet**. Esta clase deriva de la clase javax.servlet.GenericServlet, que a su vez implementa javax.servlet.Servlet. La clase HttpServlet sobreescribe el método service() de forma que puede manejar diferentes tipos de solicitudes HTTP: DELETE, GET, OPTIONS, POST, PUT, y TRACE. Por cada uno de estos tipos de solicitud, la clase HttpServlet proporciona su correspondiente método doXXX().

Aunque podemos sobreescribir (redefinir) el método service() en nuestra clase servlet, raramente hay alguna necesidad de hacerlo. Más frecuentemente querremos redefinir métodos doXXX() individuales. Para la mayoría de las aplicaciones querremos **redefinir los métodos doPost() y doGet()**, ya que ellos manejan normalmente los datos enviados por un href o un formulario (FORM).

Enlace a una página sobre la creación de Servlets con JBuilder 9 Enterprise

Un primer ejemplo de servlet: Capítulo de Arquitectura JEE - Servlets.

En el siguiente ejemplo tenemos un servlet que nos saluda y cuenta las invocaciones a doGet():

```
package docen_servlet01;

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

public class SaludoContador extends HttpServlet {

        int contador = 0;       // Contador de veces que se invoca doGet()

        public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                contador++;
                out.println("<html>");
                out.println("<head><title>Bienvenido al servlet contador</title></head>");
                out.println("<body bgcolor=\"#FFFF9D\"><FONT color=\"#000080\" FACE=\"Arial,Helvetica,Times\"
SIZE=2>"+
```

```
                                        "<CENTER><H3>Servlets</H3></CENTER>");
                Date d = new Date();
                out.println("<HR><p>¡Hola Mundo!. Fecha y hora: " + d.toString() +  "</p>");
                out.println("<p>Número de invocaciones a doGet(): " + contador + "</p>");
                out.println("</body></font></html>");

    }
}
```

El formulario puede ser:

```
<form action="../../servlet/saludo_contador" method="get">
<p><input type="submit" name="Submit" value="Pulse Enviar para llamar al servlet que saluda y cuenta"></p>
</form>
```

Ejemplo "real" del servlet:

Pulse Enviar para llamar al servlet que saluda y cuenta

En este ejemplo la aplicación se llama **public_html**, de la que cuelgan los directorios:

- **java**. Contiene:
  - **servlets**. Subdirectorio para páginas HTML. Aqui pondremos nuestro formulario.
- **WEB-INF**. Que contiene **web.xml** (donde indicamos nuestros servlets) y:
  - **classes**:
    - **docen_servlet01**. Subdirectorio que corresponde al paquete del servlet. Por tanto, aqui se encuentra **SaludoContador.class**

La invocación al servlet se hace desde el atributo action del form. Incluye una **dirección relativa** ("../../servlet/saludo_contador"). La expresión "../.." es tanto como decir "public_html", es decir, el contexto raíz de nuestra aplicación. Por ello, sería lo mismo decirle en **forma absoluta http://localhost:puerto/public_html/servlet/saludo_contador**.

Estructura de una aplicación

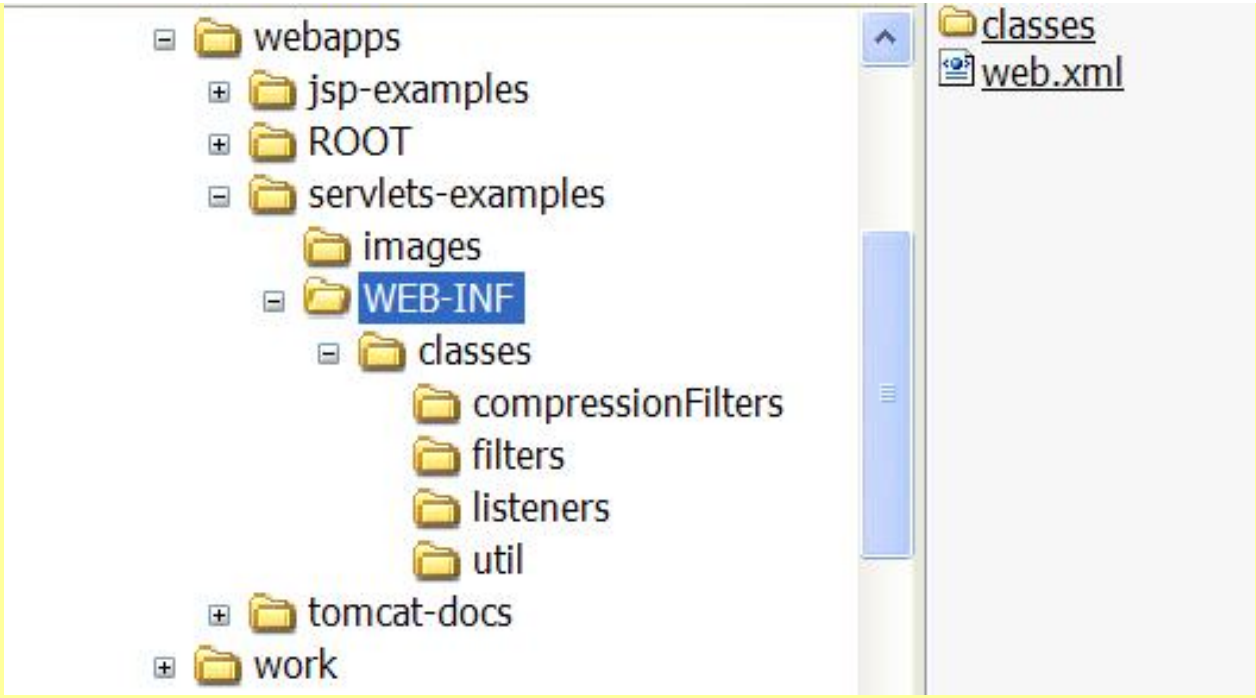
Una aplicación al menos se compone de:

- Uno o varios servlets/JSP
- Un descriptor de despliegue

Cada aplicación tiene que tener una estructura de directorios predeterminada. Empieza por un **directorio raíz** del contexto de aplicación, por ejemplo **public_html**, cuya URL sería http://www.host.com/public_html. Suele ser habitual tener el archivo index.jsp o index.html en este directorio (pero no obligatorio). Los subdirectorios que parten de este directorio raíz serán:

- Directorios para vistas (JSP, HTML, imágenes). Estos directorios no son imprescindibles y pueden llamarse como quiera.
- **WEB-INF**. Este directorio es imprescindible y debe llamarse exactamente como aparece aquí. Contiene el archivo **web.xml**, es decir, el descriptor de despliegue de la aplicación, principalmente indica los servlets que utilizará la aplicación. Subdirectorios:
  - **classes**. Este subdirectorio es imprescindible si usamos archivos .class. Debe llamarse exactamente como aparece aquí. Los subdirectorios de "classes" serán los correspondientes a los paquetes de nuestras clases Java, reproduciendo la estructura de paquetes y subpaquetes.
  - **lib**. Necesario si se quiere incluir librerías, normalmente archivos jar. Debe llamarse exactamente como aparece aquí.

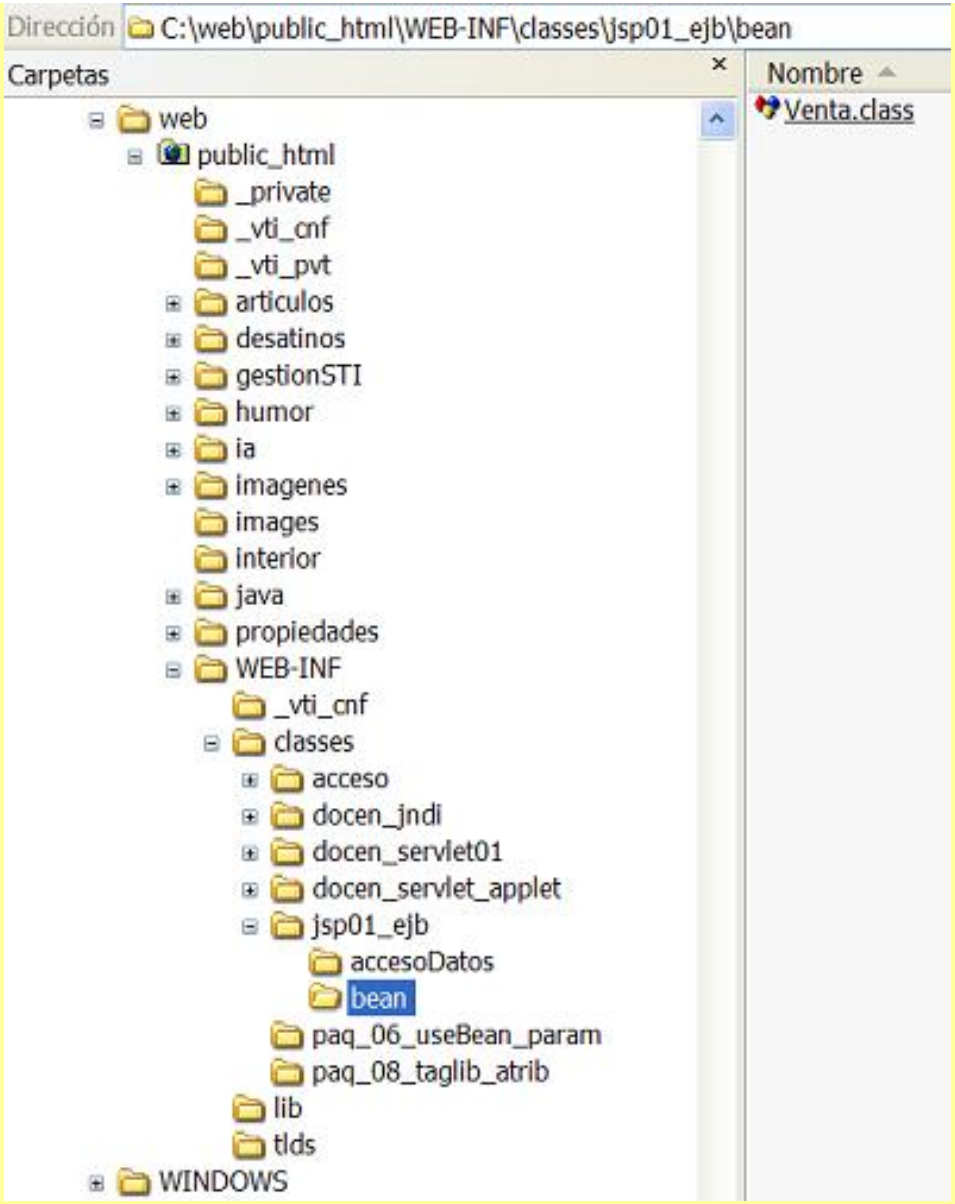
En **WEB-INF** se encuentra el archivo **web.xml**, donde se indican los Servlets contenidos en la aplicación. Hablaremos de este archivo cuando tengamos que referirnos a la instalación de los servlets. Por cierto, para hablar de forma estricta se dice "despliegue" (deployment) y no "instalación". Otro aspecto interesante, además del archivo **web.xml**, es que los servlets están físicamente en **WEB-INF/classes** (si son .class) o en **WEB-INF/lib** (si son .jar). Ver como **ejemplo** examples/WEB-INF/classes, que viene por defecto en nuestra versión de Tomcat; donde no hay librerias y por tanto no hay directorio lib. Existe un directorio WEB-INF, ya que resulta imprescindible como receptáculo de web.xml y del directorio classes:



En realidad de WEB-INF **puede colgar cualquier subdirectorio** (hay quien pone el código fuente en un subdirectorio src).

Notas dignas de tener en cuenta:

- **WEB-INF y todo lo que cuelga de él es privado**, es oculto al usuario. Intente en una instalación estándar de Tomcat acceder a web.xml, verá que no puede.
- **No caer en la falacia de que lo único que hay que poner en classes o lib son los servlets**. Una clase cualquiera, que no sea un servlet puede colgar de classes (respetando la estructura de paquetes y subpaquetes). En suma, en los directorios de WEB-INF tendremos cualquier clase que utilice nuestra aplicación, sea servlet o no.
- La estructura de directorios de nuestra aplicación no tiene que colgar necesariamente del directorio wepapps, que viene por defecto en Tomcat. En el siguiente ejemplo la aplicación public_html está en un directorio independiente de los directorios de instalación de nuestro Tomcat. De hecho, **es aconsejable por razones de mantenimiento que los directorios de nuestras aplicaciones estén fuera del espacio de directorios de Tomcat**.



Despliegue de servlets: el archivo web.xml

Ya tenemos los archivos necesarios: .class y .html. Ahora de lo que se trata es de desplegar la aplicación, es decir, instalarlo en nuestro servidor de aplicaciones.

En el siguiente caso vamos a seguir el ejemplo de SaludoContador.class, que pertenece a la aplicación **cuyo directorio "físico" es c:/web/public_html**

y **su URL es http://localhost:puerto/public_html**.

Ahora necesitamos señalar al servidor de aplicaciones el nombre y el lugar de nuestro servlet. Para ello existe un archivo denominado "**descriptor de despliegue**", que tiene como nombre **web.xml** y está en el directorio **WEB-INF**. Tomcat viene con archivos web.xml para sus aplicaciones preinstaladas. En nuestro caso vamos a crear web.xml dentro de c:/web/public_html/WEB-INF:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>public_html: proactiva-calidad LOCAL</display-name>
  <servlet>
    <servlet-name>saludo_contador</servlet-name>
    <servlet-class>docen_servlet01.SaludoContador</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>saludo_contador</servlet-name>
    <url-pattern>/servlet/saludo_contador</url-pattern>
  </servlet-mapping>
</web-app>
```

Explicación de la estructura de web.xml:

- La etiqueta servlet-name indica el nombre de la clase.
- La etiqueta servlet-class indica el paquete.clase donde se encuentra (o lo que es lo mismo el directorio.archivo_de_clase)
- La etiqueta url-pattern indica la url a la que se debe llamar para invocar al servlet. Recordar que antes hemos escrito el FORM de HTML, donde **la llamada al servlet era "../../servlet/saludo_contador". url-pattern indica la forma en la que se debe invocar al servlet**.

Interesa que se entienda **la secuencia resumida de acciones que van desde la invocación (por ejemplo en un formulario) hasta la ejecución del servlet**:

1. Previamente hay que tener en cuenta que el servidor ha cargado en memoria el contenido de server.xml y de web.xml.
2. Desde el formulario (en HTML, JSP, Servlet, etc.) se realiza la **llamada "/public_html/servlet/saludo_contador"**.
3. El contenedor Servlet/JSP **reconoce el contexto de aplicación** al que se dirige la invocación, por tanto sabe el descriptor de despliegue (web.xml) al que debe consultar.

4. El contenedor **busca dentro de la aplicación /public_html la <url-pattern> (en este caso es "/servlet/saludo_contador")**. Puesto que coincide la url de la llamada con la url almacenada en la aplicación (en web.xml), el contenedor **sabe que el alias (<servlet-name>) del servlet es "saludo_contador"**.

5. El contenedor ya sabe el alias o nombre del servlet, a continuación **busca dicho nombre dentro de las etiquetas <servlet> (subetiequeta <servlet-name>), cuando la encuentra busca la subetiqueta <servlet-class>, donde se define el lugar físico donde se encuentra la clase a la que debe invocar**.

Una moraleja de esta secuencia es que **el servidor separa la dirección lógica (URL) de la localización física del archivo .class**. En la llamada al servlet lo primero es encontrar la URL y, una vez encontrada ésta, se localiza al servlet "físicamente". Esto permite cambiar la URL de llamada sin recompilar la clase, simplemente cambiando web.xml.

Despliegue de servlets: el archivo server.xml

La ruta física de nuestra aplicación (c:/web/public_html) no corresponde con la URL. **La URL de la aplicación es una dirección "virtual" o lógica dentro de nuestro servidor de aplicaciones y es independiente del espacio físico donde guardamos los archivos de la aplicación**

En el archivo **server.xml** es donde configuramos las aplicaciones de nuestro servidor, entre otras cosas definiremos el directorio de los archivos y **la URI de nuestra aplicación, en nuestro caso es /public_html**.

Para una explicación de cómo definir una aplicación en server.xml ver el capítulo dedicado a Herramientas - Tomcat

El archivo .class del servlet se colocan en un subdirectorio de **WEB-INF** que se llama **classes/docen_servlet01**, ya que éste es el paquete de la clase. Con lo cual tendremos la clase en **c:/web/public_html/WEB-INF/classes/docen_servlet01**.

Aplicando todo lo indicado escribimos en server.xml:

```
                        ....
                        <Context path="/public_html"
                                    reloadable="true"
                                    docBase="C:\DOC\Java_eclipse\public_html" />
                    </Host>
                </Engine>
        </Service>
</Server>
```

Un consejo de prudencia: antes de tocar server.xml haga una copia de respaldo del original.

Todo lo dicho corresponde a nuestra versión de Tomcat, pero puede cambiar en otras y también depende de si tenemos configurados servidores virtuales. Nuestra aplicación ("context" en la jerga Tomcat) la instalaremos **dentro del host por defecto**. Para ello, tenemos que buscar en el final de server.xml, **justo antes de </host>**.

Una buena noticia: esta modificación del archivo server.xml la suelen realizar la mayor parte de IDEs. Por ejemplo en **Eclipse** o JBuilder se modifica el archivo introduciendo como nombre de la aplicación el nombre del proyecto. En **Eclipse** es necesario indicar a la hora de crear un proyecto que el tipo de proyecto es un "Tomcat project".

Un aspecto que a menudo se olvida es que el despliegue del servlet exige (en general, depende de la configuración del servidor) **una recarga de la aplicación**. En algunas instalaciondes del servidor de aplicaciones, la recarga puede producirse de forma automática cuando el servlet y el web.xml son modificados (depende del correcto funcionamiento del atributo reloadable=true). Sin embargo en muchos casos se exige una recarga manual. Si en la invocación al servlet aparece el siguiente código de error:

> ***The requested resource (/Nombre_del_Servlet) is not available.***

La solución (presuponiendo que se ha editado web.xml y server.xml correctamente) está en la mayoría de los casos en entrar en el Manager del servidor de aplicaciones y **recargar (reload) la aplicación** correspondiente. Para acceder al manager en una instalación estándar de Tomcat: http://localhost:8080/manager/html. Si esto último no funciona y empieza a desesperarse reinicie el servicio Tomcat desde el sistema operativo.

Por ultimo, para comprobar que todo ha ido bien se puede probar a hacer desde el navegador: http://localhost:8080/public_html/servlet/saludo_contador. En cuento esto funcione podemos probarr con nuestro formulario. Si el formulario no invoca al servlet, probablemente el error sea que no se ha introducido correctamente la ruta del servlet en el atributo **action** del formulario o que no se usa el método HTTP correcto (post, get, etc.).

Un enlace que explica en mayor detalle el despliegue de servlets y los archivos WAR.

Existen dos métodos de despliegue:

- Por medio de archivos WAR (Web Archive) no es más que un fichero comprimido (al igual que un JAR) que contiene todos los archivos necesarios para la aplicación web. Este fichero lo puedes generar tu mismo, pero es más fácil si dejas que lo cree alguna de las herramientas que existen para ello (están en el J2EE)
- Instalación de una aplicación editando web.xml y server.xml. Este capítulo sigue este método.

Volver al índice