Aplicaciones JEE: Descriptores de despliegue

En un servidor de aplicaciones web, por ejemplo Glashfish o JBoss, podemos encontrarnos una colección de recursos tales como:

- JSP's
- Servlets
- Ficheros HTML
- Imágenes
- etc...

Todos ellos ubicados en un URI específico, muy bien definido y estructurado.
Recordemos que JBoss es un servidor de aplicaciones JavaEE de código abierto implementado en Java puro, que puede ser utilizado en cualquier sistema operativo para el que esté disponible la máquina virtual de Java.

Aplicaciones Web JEE

Una aplicación web está organizada en una estructura jerárquica de directorios:

- Un directorio privado **WEB-INF**: contiene los recursos que no son descargables para el cliente
- Un **directorio público** que contiene los recursos públicos, por ejemplo:

```
\miaplicación\
\index.html
login.jsp
\images\logo.gif
\doc\tutorial.pdf
\WEB-INF\
      web.xml (Deployment Descriptor)
      \classes\ServletCompras.class
      \lib\cualquierOtraAPI.jar
```

Además, una aplicación web puede ser empaquetada en un fichero **WAR**.

Archivos WAR

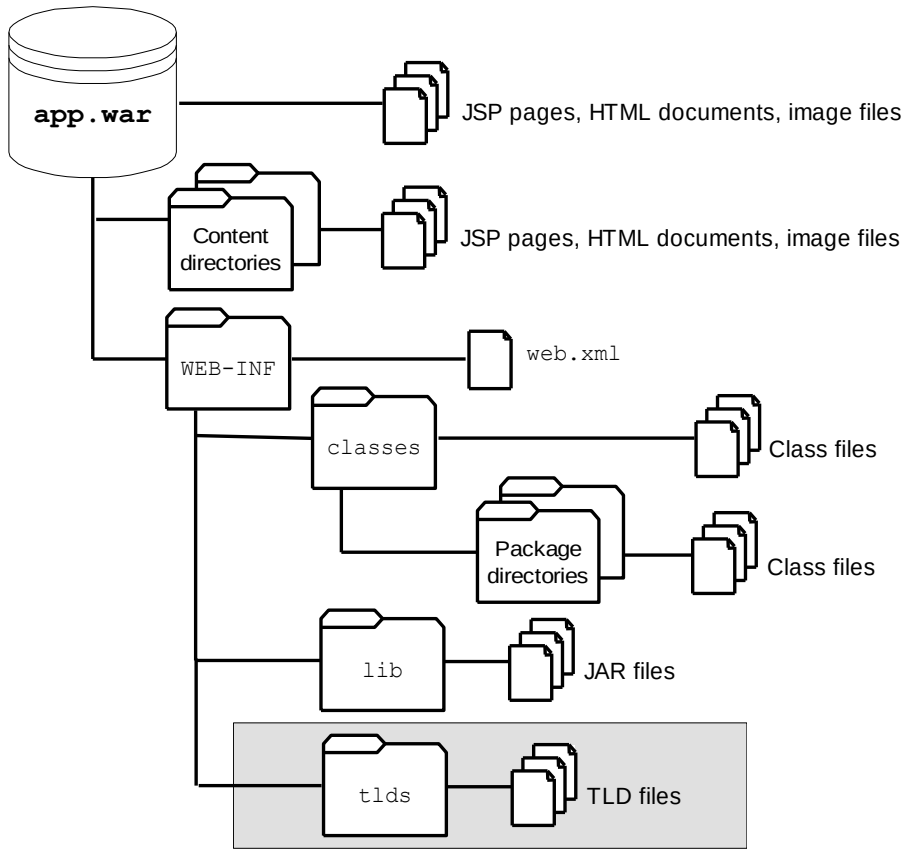
Su nombre completo es: **W**eb **A**pplication **R**esource o a veces se le puede llamar **W**eb **AR**chive, y permite empaquetar en una sola unidad aplicaciones web java completas:

- Servlets y JSPs
- Contenido estático
    - HTML
    - Imágenes
    - etc…
- Otros recursos web

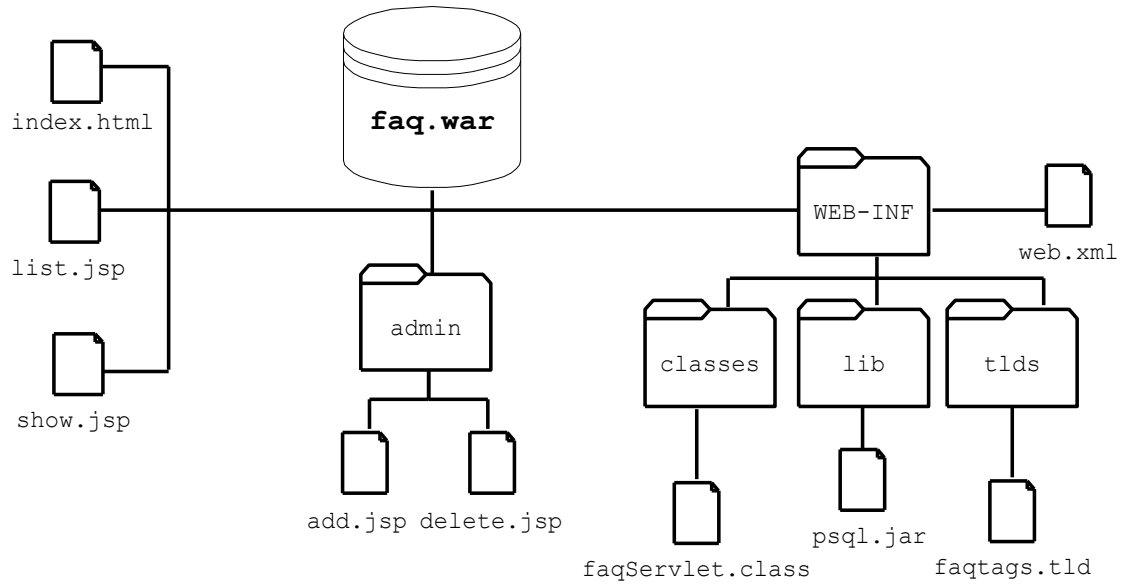
Sus principales ventajas son:

- Simplifican el despliegue de aplicaciones web.
    - Facilidad de instalación
    - Un solo fichero para cada servidor en un cluster.
- Seguridad
    - No permite el acceso entre aplicaciones web distintas

Su estructura es la que sigue:



Un ejemplo práctico podría ser:



Un archivo WAR puede ser firmado digitalmente de la misma manera que un archivo JAR, con el fin de permitir que otros puedan determinar la procedencia del código fuente. Usar archivos `*.war` es práctico cuando queremos desplegar aplicaciones en una maquina remota.

El directorio `/WEB-INF/` del archivo WAR contiene el descriptor **web.xml** que define la estructura de la aplicación web. Si la aplicación web sólo está sirviendo archivos JSP, el archivo web.xml no es estrictamente necesario. Si la aplicación web utiliza **servlets**, a continuación, el contenedor servlet utiliza web.xml para comprobar a qué servlet se dirigirá una solicitud de URL. Web.xml también se utiliza para definir las variables que pueden ser referenciadas dentro de los servlets y se utiliza para definir las dependencias que el modulo espera configurar. Un ejemplo es una dependencia de una sesión de correo utilizado para enviar un email, el contenedor de servlets es responsable de proveer este servicio. Ventajas de los archivos WAR:

- Fácil desarrollo , testeo y despliegue
- La versión de la aplicación desplegada se identifica fácilmente
- Todos los contenedores de JavaEE soportan archivos WAR

Una desventaja es usarlos en entornos muy dinámicos, dónde los cambios menores no se pueden hacer en tiempo de ejecución. Cualquier cambio requiere crear de nuevo todo el archivo WAR.

web.xml: Descriptor de despliegue

- /WEB-INF/web.xml
- Documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
 <!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>HelloServlet</servlet-name>
         <servlet-class>mypackage.HelloServlet</servlet-class>
     </servlet>
     <servlet-mapping>
         <servlet-name>HelloServlet</servlet-name>
         <url-pattern>/HelloServlet</url-pattern>
     </servlet-mapping>
     <resource-ref>
         <description>
            Resource reference to a factory for javax.mail.Session
            instances that may be used for sending electronic mail
            messages, preconfigured to connect to the appropriate SMTP
            server.
         </description>
         <res-ref-name>mail/Session</res-ref-name>
         <res-type>javax.mail.Session</res-type>
         <res-auth>Container</res-auth>
     </resource-ref>
 </web-app>
```

- En él se dan de alta
    - Servlets
    - Parámetros del contexto
    - TLDs
    - Filtros
    - Etc.

En resumen, **web.xml** será el descriptor de despliegue de los módulos **WAR**. Estos módulos están empaquetados en un archivo JAR utilizado para distribuir una colección de Java Server Pages (JSP), servlets, clases Java, archivos XML, librerías de tags y páginas web estáticas que juntos constituyen una aplicación web.

Los descriptores son un elemento necesario antes de que pueda invocarse cualquier servlet. Describen las clases, los recursos y la configuración de la aplicación, así como la forma en que el servidor web los utiliza para suministrar las solicitudes web.

Descriptores de implementación, Servlets y rutas URL

A continuación, se muestra un ejemplo sencillo de `web.xml` que asigna todas las rutas de URL (/*) a la clase de servlet `mysite.server.ComingSoonServlet`:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    <servlet>
        <servlet-name>comingsoon</servlet-name>
        <servlet-class>mysite.server.ComingSoonServlet</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>comingsoon</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Es habitual que a los descriptores de despliegue también se les llame descriptores de implementación.

`web.xml` define las asignaciones entre las rutas de URL y los servlets que controlan las solicitudes con estas rutas. El servidor web utiliza esta configuración para identificar el servlet que controla una solicitud concreta y, a continuación, invoca el método de clase que concuerda con el método de solicitud (por ejemplo, el método `doGet()` de las solicitudes de tipo GET HTTP).

Para asignar una URL a un servlet, declara el servlet con el elemento `<servlet>` y, a continuación, define una asignación desde una ruta de URL hasta una declaración de servlet con el elemento `<servlet-mapping>`.

El elemento `<servlet>` declara el servlet, incluidos el nombre que utilizan otros elementos del archivo para hacer referencia al mismo, la clase que se va a utilizar para el servlet y los parámetros de inicialización. Puedes declarar varios servlets mediante la misma clase utilizando para ello diferentes parámetros de inicialización. El nombre de cada servlet debe ser único en todo el descriptor de implementación.

El elemento `<servlet-mapping>` especifica un patrón de URL y el nombre de un servlet declarado que se va a utilizar para las solicitudes cuyas URL coinciden con el patrón. El patrón de URL puede utilizar un asterisco (`*`) al principio o al final del patrón para indicar

cero o varios caracteres. (El estándar no admite comodines en mitad de una cadena y no permite varios comodines en un patrón.) El patrón coincide con la ruta completa de la URL, que comienza con la barra inclinada (`/`) seguida a continuación del nombre del dominio.

Una opción a menudo necesaria es el tiempo de espera de sesión. Diferentes contenedores de servlet utilizan valores predeterminados para diferentes tiempos de espera, como 30 minutos para Apache Tomcat. Puedes configurar el tiempo de espera con:

```
<session-config>
     <session-timeout>30</session-timeout>
 </session-config>
```

Después de que el tiempo de espera haya caducado, el método `close()` de la clase Application será llamado. Debe ser implementado si se desea controlar situaciones de tiempo de espera.