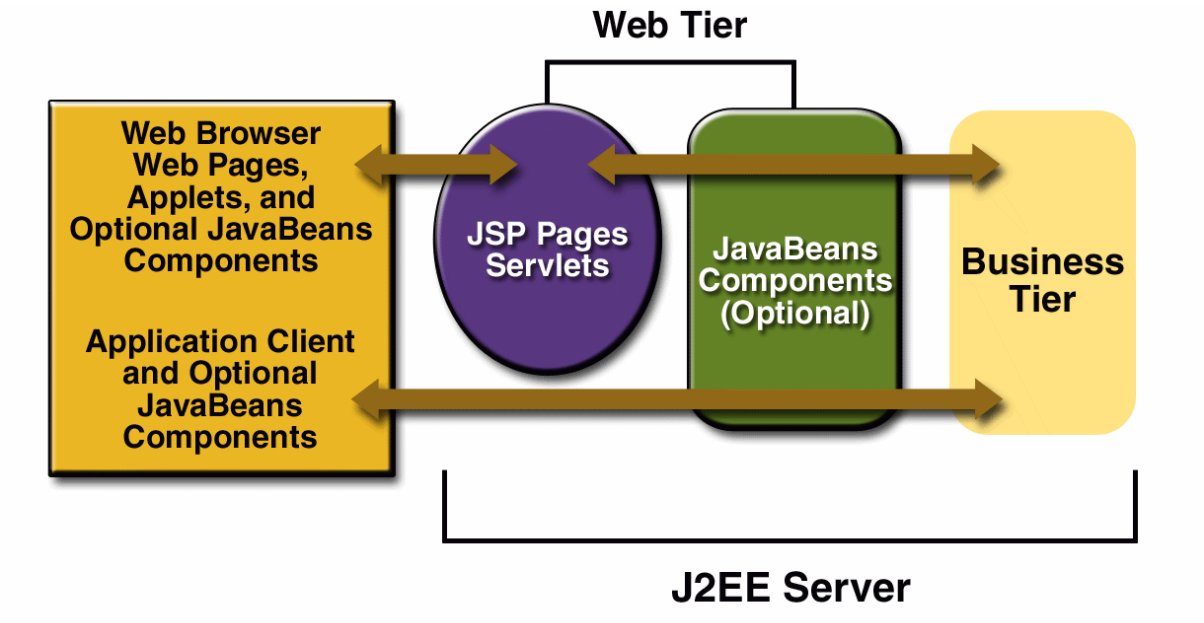
2. Aplicaciones Web con JavaEE. Servlets y JSP.

1. Modelo de aplicaciones web. Componentes. El marco de referencia JavaEE.

Las aplicaciones web son programas que se ejecutan en uno o varios servidores web e interactúan con los usuarios a través de páginas web (HTML). De esta forma, los clientes sólo tienen que disponer de un navegador web y una conexión de red para comunicarse con el servidor. En dicho navegador se visualiza el interfaz de usuario, formado por una o más páginas web creadas dinámicamente en el servidor en respuesta a las peticiones del cliente y al estado de la propia aplicación web. De esta forma se simplifica el mantenimiento de la aplicación, ya que los cambios y ampliaciones sólo deben ser aplicadas en el servidor.

La Edición Empresarial de Java (JavaEE) define los componentes que intervienen en dichas aplicaciones según el siguiente esquema:



El cliente se conecta a la aplicación normalmente mediante un navegador, aunque también puede hacerlo mediante una aplicación de escritorio con conexión al servidor (cuadro amarillo).

En el servidor se encuentra la aplicación, que se compondrá principalmente de la capa web (Web Tier) encargada de recibir las peticiones del cliente y generar las páginas de respuesta, y de una capa de negocio (Business Tier) donde se encontrará la lógica de la aplicación propiamente dicha (y el modelo de datos, normalmente implementado mediante conexiones a sistemas de bases de datos).

Los componentes que integran la capa web, son los Servlets y las páginas JSP. La

comunicación entre la capa web y la capa de negocio se implementa generalmente mediante componentes JavaBeans, que mantienen el estado de la aplicación y permiten consultar y modificar dicho estado (y todos los datos en general).

Los servlets son clases Java especializadas (que heredan de la clase Servlet) que implementan un servicio. Para ello implementan un método de servicio que recibe la petición y genera la respuesta a dicha petición.

Las páginas JSP realizan la misma función que los servlets, pero tienen la estructura de una página web en la que se insertan determinadas etiquetas especiales o elementos JSP que generan la parte dinámica de la página. De esta forma el diseño estético de la página es mucho más cercano al diseño de páginas HTML, y puede ser realizado por diseñadores web sin conocimientos avanzados de programación.

2. Servidores JavaEE.

Los elementos que componen la aplicación web (servlets, páginas JSP, JavaBeans...) se ejecutan en el entorno de un servidor que recibe las peticiones, las preprocesa, y llama a los métodos adecuados de dichos componentes, para que éstos generen dinámicamente la respuesta. Posteriormente, el propio servidor JavaEE envía la respuesta al cliente. El servidor JavaEE se encarga de generar el entorno de ejecución, inicializar los componentes, gestionar la concurrencia, etc. Para ello, el propio servidor dispone de un entorno JDK.

A éstos servidores se les llama también contenedores, ya que contendrán en unos directorios específicos los componentes de la aplicación o las aplicaciones desplegadas en ellos.

Existen varios servidores y todos ellos deben seguir la especificación JavaEE. Los más conocidos son Apache Tomcat, GlassFish (originalmente de Sun Microsystems), Oracle WebLogic, etc.

Para instalar o desplegar una aplicación web, es necesario instalar los distintos componentes de la aplicación y las bibliotecas necesarias en los directorios adecuados del contenedor JavaEE, así como definir un fichero de despliegue (en formato XML normalmente), que entre otras cosas, define ciertas propiedades de dichos componentes y establece las URL's que se usarán para acceder a ellos.

3. Tecnología Java Servlets. Métodos HTTP. Formularios HTML.

Los **servlets** son clases Java especiales que ofrecen un servicio generando una respuesta a partir de la petición del usuario.

En general, los servlets pueden dar servicio a cualquier protocolo de red, aunque los más usados son los servlets HTTP, que heredan de la clase **HTTPServlet** y dan servicio web. Esto es, reciben una petición web, la procesan y generan una respuesta web (normalmente en formato HTML, aunque a veces se pueden usar otros formatos, como XML).

Cuando se realiza el despliegue de la aplicación, a cada servlet se le asigna un patrón de URL. Cuando el usuario realiza una petición a dicha URL, el servidor analiza la petición y crea un objeto **HTTPServletRequest** que contiene los elementos de dicha petición así como información acerca del cliente. También crea un objeto **HTTPServletResponse** y lo inicializa para que éste contenga la respuesta. Una vez hecho esto, llama al método de servicio del servlet, pasándole dichos objetos como parámetros. En dicho método se realizará el procesamiento de la petición y se generará la respuesta, que se añadirá al objeto HTTPServletResponse. Una vez ejecutado dicho método, el contenedor JavaEE enviará la respuesta al cliente y continuará a la espera de nuevas peticiones.

El protocolo HTTP define varios métodos de petición (varias formas en las que el cliente realiza la petición al servidor), de los cuales, los más importantes son el método *GET* y el método *POST*. El primero se usa para pedirle al servidor un recurso (una página web, por ejemplo). El segundo se usa también para solicitarle un recurso al servidor, pero en ésta ocasión, se le envían datos en la petición (parámetros de la petición).

Los métodos más importantes de la clase **HTTPServlet** (que pueden ser sobreescritos para darles una funcionalidad especifica) son:

- *void init()* : Para realizar inicializaciones. Se ejecuta cuando el contenedor carga el servlet por primera vez.

- *void destroy()* : Para liberar recursos. Se ejecuta justo antes de destruir el servlet.

- *String getServletInfo()* : Debe devolver una cadena con la descripción del servlet.

- *void doGet(HTTPServletRequest request, HTTPServletResponse response)* : Es el método de servicio para atender peticiones de tipo GET.

- *void doPost(HTTPServletRequest request, HTTPServletResponse response)* : Método de servicio para peticiones de tipo Post.

Además podemos usar el método *getServletContext()* para obtener un objeto de tipo **ServletContext**, que contiene información sobre el contexto de ejecución del servlet (la aplicación web), y que nos permite guardar y recuperar atributos entre invocaciones y compartir dichos atributos con otros servlets, así como lanzar peticiones de un servlet a otro o acceder a ficheros o recursos estáticos de la aplicación web,

Los métodos más importantes de **HTTPServletRequest** son:

- *String getParameter(String name)* : Devuelve el valor del parámetro de la petición indicado como parámetro (o null si no existe el parámetro).
- *Cookie[] getCookies()* : Para obtener un array con las cookies incluídas en la petición. Ver la referencia de la API para la clase **Cookie**.

- *HTTPSession getSession(boolean create)* : Devuelve la sesión HTTP actual, o crea una sesión nueva si no está definida y el parámetro create es true.

Los métodos más importantes de **HTTPServletResponse** son:

- *Writer getWriter()* : Devuelve un objeto de tipo Writer que nos permite escribir el resultado de la ejecución del servlet en el objeto de respuesta.

- *void setContentType(String type)* : Establece el tipo de contenido, normalmente “text/html”.

- *void addCookie(Cookie cookie)* : Añade una cookie a la respuesta.

Para obtener más información sobre las clases mencionadas se recomienda consultar la referencia de la API estándar.

A continuación aparece el código fuente de un ejemplo típico de un servlet sencillo:

```
public class ServletHolaMundo extends HttpServlet {

   /**
    * Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
    * @param request servlet request
    * @param response servlet response
    * @throws ServletException if a servlet-specific error occurs
    * @throws IOException if an I/O error occurs
    */
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet ServletHolaMundo</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("<h1>Hola, Mundo!</h1>");
         if (request.getParameter("nombre")!=null)
            out.println("<h2> nombre = " + request.getParameter("nombre") +" </h2>");
         out.println("</body>");
         out.println("</html>");
      } finally {
         out.close();
      }
   }


   /**
    * Returns a short description of the servlet.
    */
   @Override
   public String getServletInfo() {
      return "Mi primer servlet";
   }

}
```

Como hemos visto, el servlet puede tomar parámetros de la petición. Ésta es la forma

estándar de leer datos de entrada del usuario. Para que éste pueda introducir dichos datos en una página web se usan los formularios HTML.

Un formulario HTML es una porción de la página web que puede contener varios elementos tales como campos de texto, áreas de edición, campos ocultos, listas desplegables, casillas de verificación, botones de envío, etc.

Al formulario HTML se le asocia una acción, que será la URL de la petición que se realizará cuando el usuario pulse el botón de enviar. A cada elemento de entrada se le asocia un nombre, que coincidirá con el nombre del parámetro generado en la petición para contener el valor.

A continuación se muestra un fragmento de una página web donde se aprecia un formulario HTML:

```
<form action="ServletConsultaEmpleados" method="POST">
Departamento: <input type= "text" name ="dto" size="20">
<input type="submit" name="boton" value="Enviar"><br>
</form>
```

En éste fragmento se define un formulario HTML que contiene un campo de texto llamado “dto” y un botón con la etiqueta “enviar”. Cuando el usuario pulse dicho botón, se realizará una petición al recurso indicado por la acción asociada al formulario, es decir a ServletConsultaEmpleados. En esa petición se incluirá el parámetro “dto” con el valor introducido por el usuario en el campo de texto y el parámetro “boton” con el valor “Enviar”. De ésta forma el servlet podrá procesar los datos introducidos por el usuario y generar una respuesta en función de ellos.

4. Páginas JSP. Componentes.

En el servlet de ejemplo del apartado anterior se puede apreciar claramente cierta redundancia en las llamadas out.println(...) para generar líneas de código HTML en la respuesta. Si además, la página a generar tiene un tamaño considerable, el código del servlet se puede hacer muy largo y difícil de mantener. Hay que tener en cuenta también los caracteres de escape cada vez que aparecen comillas en el código HTML, que también complican dicho código.

El problema anterior se soluciona con la tecnología JSP (*Java Server Pages*, o páginas de servidor Java).

En dichas páginas escribimos directamente el código HTML que se generará, es decir, en principio serán similares a las páginas HTML, con la particularidad de que podemos insertar determinados elementos JSP para generar contenido dinámico sobre la página.

Una página **JSP**, consistirá por tanto en un fichero de texto, con formato HTML y con extensión .jsp, que se instalará en el directorio adecuado del contenedor JavaEE y que será traducido automáticamente por el propio contenedor generando el código fuente de un

servlet, que será compilado también automáticamente y desplegado para dar servicio.

De ésta forma se pueden utilizar herramientas de diseño web para generar la parte estática de las páginas (la parte visual), y luego insertar los elementos JSP para darle funcionalidad.

Los componentes de una página JSP, además del código HTML son:

- **Directivas:** Entre las que se incluyen las directivas **page**, **include** y **taglib**. Su sintaxis es **<%@ nombre-de-directiva [atributo="valor" atributo="valor" …] %>**

- **Elementos de secuencias de comandos**, que incluyen:
  - **Expresiones:** se substituyen por el valor devuelto al evaluar la expresión Java contenida. Su sintaxis es : **<%=expresion%>**.
  - **Scriptlets:** Un scriptlet es un conjunto de sentencias java que se ejecutará al generar la página. Las sentencias java contenidas en un scriptlet, se copiarán directamente al código fuente del servlet generado. Su sintaxis es **<% instrucciones java %>**.
  - **Declaraciones:** Las declaraciones también contienen instrucciones java dedicada a realizar declaraciones de variables y métodos. Éstas líneas también se copiarán al código fuente del servlet generado, pero fuera del método de servicio. Su sintaxis es **<%! declaraciones java %>**.

- **Acciones:** Son elementos JSP de alto nivel que crean, utilizan o modifican otros objetos. Su sintaxis genérica es **<nombre-de-etiqueta [atr="valor"...]> … </nombre-de-etiqueta>**. Algunas de las acciones JSP más importantes son:
  - **<jsp:useBean>** para usar un JavaBean.
  - **<jsp:setProperty>** para establecer una propiedad de un JavaBean.
  - **<jsp:getProperty>** para consultar una propiedad de un JavaBean.
  - **<jsp:include>** para fusionar otro recurso a la salida de nuestra página.
  - **<jsp:forward>** para reenviar la petición a página JSP o a un servlet.

- **Comentarios:** Tienen la sintaxis **<%-- comentario --%>**

Existen también unos **objetos implícitos** que podemos usar en nuestro código Java dentro de la página JSP y que son:

- **request**: el objeto de petición.
- **response**: el objeto de respuesta.
- **pageContext**: el contexto de la página
- **session**: la sesión HTTP.
- **application**: el contexto del servlet o contexto de aplicación.
- **out**: el flujo de salida empleado para generar la respuesta.
- **config**: el objeto servletConfig.
- **page**: una referencia a la propia página JSP.
- **exception**: Una excepción sin capturar que invoca a una página de error.