



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

**FACULTAD DE ESTUDIOS SUPERIORES
CAMPUS ARAGÓN**

**Desarrollo de aplicaciones web
utilizando el framework Struts**

T E S I S

**QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN**

P R E S E N T A:

Muñoz Gutiérrez Juan

**ASESOR DE TESIS:
M. en C. Jesús Hernández Cabrera**

MÉXICO, 2008.





Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Esta tesis esta dedicada a mi papá, a mi mamá y hermanas

Contenido

Introducción	6
Objetivos	7

1 INTRODUCCIÓN A J2EE Y A LOS SERVICIOS WEB

1.1 ¿Qué es J2EE?	8
1.2 Arquitectura J2EE	9
1.3 Características J2EE	13
1.4 Contenedores de la plataforma J2EE	14
1.5 Tecnologías y Servicios Empleados J2EE	16
1.5.1 Tecnologías J2EE	16
1.5.2 Servicios proporcionados por J2EE	18
1.6 Introducción a los servicios J2EE	21
1.7 ¿Qué es y que hace un Servidor Web?	22
1.7.1 Servidor web	22
1.7.2 ¿Qué hace un servidor Web?	22
1.8 Métodos de petición (HTTP post y HTTP get)	23
1.8.1 ¿Qué es HTTP?	23
1.8.2 Especificación http	24
1.8.3 El envío de la petición http	25
1.8.4 Métodos de petición	26
1.8.5 Common Gateway Interface (CGI)	27

2 MANIPULACIÓN DE BASE DE DATOS CON JDBC

2.1 Introducción	29
2.2 ¿Qué es JDBC?	30
2.3 Tipos de controladores JDBC	31
2.4 Visión general del proceso de JDBC	32
2.4.1 Cargar controlador JDBC	33
2.4.2 Conectar con el DBMS	33
2.4.3 Crear y ejecutar una instrucción SQL	35
2.4.4 Procesar los datos que devuelve el DBMS	36
2.4.5 Terminar la conexión del DBMS	37
2.5 Objetos Statement	38
2.5.1 El objeto Statement	38
2.5.2 El objeto PreparedStatement	41
2.5.3 El objeto CallableStatement	42
2.6 Metadatos	44

3 SERVLETS Y JAVA SERVER PAGES (JSP)

3.1 Servlets	46
3.1.1 ¿Qué son los Servlets?	46
3.1.2 Ventajas de usar Servlets	48
3.1.3 Estructura básica de un Servlet	49
3.1.4 Pasos para crear un Servlet	50
3.1.5 Descriptor de despliegue (DD)	54
3.1.6 Como el contenedor toma una petición	56
3.1.7 Ciclo de vida de los Servlets	58
3.1.8 Configuración de parámetros	60
3.1.9 Atributos	65
3.1.10 Administración de Sesiones	67
3.2 Java Server Page (JSP)	74
3.2.1 ¿Que son las JSP?	74
3.2.2 Ventajas de usar JSP	74
3.2.3 Ciclo de vida de las JSP	75
3.2.4 Componentes de una JSP	78
3.2.5 Objetos implícitos	82
3.2.6 Java Standard Tag Library (JSTL)	83
3.2.7 Etiquetas Personalizadas (Custom Tags)	88

4 ARQUITECTURA DEL FRAMEWORK STRUTS

4.1 ¿Qué son las aplicaciones Frameworks?	89
4.2 ¿Qué son los Struts?	89
4.3 Componentes de Struts	90
4.3.1 Hyperlinks	90
4.3.2 HTML Forms	91
4.3.3 ActionForm	92
4.3.4 Custom actions	92
4.3.5 Actionmapping	93
4.3.6 Actionservlet	93
4.4 Modelo Vista Controlador (MVC)	94
4.5 Struts Arquitectura	98
4.6 Flujo de control Struts	100
4.7 Configuración de componentes de Struts	101
4.7.1 El archivo web.xml	102
4.7.2 El Struts configuration	103
4.8 Struts Aplicación	107

5 DESARROLLO DE UN CASO PRÁCTICO

5.1 Descripción del problema	116
5.2 Modelo de casos de uso	117
5.3 Diagrama delimitación del sistema	123
5.4 Diagrama casos de uso	124
5.5 Diagramas de secuencia	125
5.6 Script SQL base de datos SVONLINE	130
5.7 Modelo Entidad Relación	132
5.8 Pantallas	133
CONCLUSIÓN	146
BIBLIOGRAFÍA	147
REFERENCIAS	148

INTRODUCCIÓN

Las aplicaciones web han evolucionado constantemente al paso del tiempo, los desarrolladores pueden escoger entre varios lenguajes de programación para diseñar aplicaciones web como son PHP, ASP.NET y JAVA que destacan entre los más usados en cuanto a tecnologías de desarrollo web.

Al hablar de aplicaciones web, JAVA es una buena alternativa para el desarrollo de este tipo de aplicaciones ya que las puede hacer robustas, portables y escalables, además de que no se requiere pagar licencia para desarrollar en Java.

La arquitectura J2EE, tiene cuatro capas las cuales son: capa cliente, capa web, capa de negocios y capa de datos, aunque la aplicación multicapa J2EE está generalmente considerada en tres capas de aplicación porque estas están distribuidas en tres diferentes localidades: máquina cliente, la máquina del servidor J2EE y la máquina del servidor de la base de datos.

La capa cliente o capa de presentación o aplicación está compuesta por los programas que interactúan con el usuario, estos programas tal como un navegador piden datos al usuario y convierten su respuesta en peticiones que son reenviados a un componente que procesa la petición y devuelve el resultado al programa cliente.

En la máquina del servidor J2EE es donde se encuentran la capa web y la capa de negocios. Los componentes de la capa web utilizan HTTP para recibir peticiones y enviar respuestas a clientes, luego un componente de la capa web recibe una petición de datos del cliente, procesa la información controlada por una clase java, este recibe los datos del DBMS en la capa de Enterprise Information System (Capa de datos).

En la capa de Enterprise Information System (EIS), conecta a las aplicaciones J2EE con recursos y sistemas heredados que se encuentran en la red de la empresa. En el EIS es donde la aplicación J2EE se conecta en forma directa o indirecta con distintas tecnologías, incluyendo DBMS y mainframes que forman parte de los sistemas de misión crítica que mantienen funcionando a la empresa.

Para desarrollar aplicaciones web los desarrolladores han buscado diferentes caminos para reducir tiempos, esfuerzos y calidad de las aplicaciones web el resultado han sido el uso de patrones de diseño y utilización de frameworks que facilitan algunas tareas en el desarrollo de aplicaciones web.

Un Framework Web, por tanto, podemos definirlo como un conjunto de componentes (por ejemplo clases en java, descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de aplicaciones Web.

En nuestro estudio nos enfocaremos en estas capas tomando como punto importante el modelo MVC el cual es un patrón de diseño que nos ayuda a separar la lógica de negocios (modelo), la vista y el controlador y descubriremos las ventajas de utilizar el framework Struts que está basado en dicho modelo.

OBJETIVOS

Objetivo general:

Desarrollo de aplicaciones web utilizando el framework Struts usando Apache-Tomcat como servidor y Postgresql como base de datos.

Objetivo particular:

- Conocer la arquitectura J2EE para el desarrollo de aplicaciones web.
- Aprender acerca del patrón de diseño MVC.
- Conocer que es un framework.
- Aprender acerca del framework Struts desde su funcionamiento hasta su aplicación.
- Desarrollo e implementación de una aplicación web utilizando el framework Struts.

INTRODUCCIÓN A J2EE Y A LOS SERVICIOS DE INTERNET

1.1 ¿Qué es J2EE?

Java 2 Enterprise Edition (J2EE) es una plataforma basada en java la cual nos ofrece un ambiente multicapa para el despliegue, desarrollo y ejecución de aplicaciones empresariales. Se le denomina plataforma porque proporciona especificaciones técnicas que describen el lenguaje pero además provee herramientas para implementar productos de software (aplicaciones) basadas en dichas especificaciones. Una parte importante en el desarrollo de J2EE es el entorno de colaboración fomentado por Sun Microsystem ya que los proveedores y técnicos se reúnen en Java Community Program (JCP) para crear e implementar tecnologías basadas en Java, todo esto con el fin de desarrollar aplicaciones más robustas que trabajen del lado del servidor, los paquetes inicialmente introducidos por el J2SDK han estado mejorando para la construcción de aplicaciones a gran escala como los servidores web todo esto condujo a la creación de una edición empresarial conocida como J2EE.

J2EE ha sido diseñada para aplicaciones distribuidas que son construidas con base a componentes(unidades funcionales de software), las cuales interaccionan entre si para formar parte de una aplicación J2EE. Un componente de esta plataforma debe formar parte de una aplicación y ser desplegado en un contenedor.

Para comprender como esta conformada la plataforma J2EE es indispensable analizar su arquitectura, tecnologías y servicios empleados para desarrollar y soportar aplicaciones en J2EE.

1.2 Arquitectura J2EE

J2EE esta basado en una arquitectura multicapa, para comprender que queremos decir con multicapa podemos definir una capa como un conjunto de tecnologías que proporcionan varios (uno o más) servicios a sus clientes.

Entonces podemos decir que una arquitectura multicapa es aquella que esta formada por varias capas.

Muchas empresas o consultarías de informática utilizaban un modelo de cliente-servidor de dos capas, en el cual los programas de escritorio conocidos como clientes solicitaban información a través de la red corporativa a servidores que ejecutaban software que respondían a las peticiones de los clientes, sin embargo esta arquitectura de dos capas depende mucho de mantener actualizado el software de los clientes, esto es difícil de mantener en una gran empresa, además de que presenta una fuerza de trabajo consistente de empleados en campo y otros usuarios remotos. Hubo que abandonar la arquitectura cliente-servidor de dos capas y construir en su lugar una arquitectura multicapa nueva. El equipo de desarrollo de Java Sun Microsystem, en conjunto con el programa comunitario de Java (JCP), desarrolló el lenguaje de Java para utilizarlo para construir software para esta nueva arquitectura multicapa, esta se compone por lo siguiente:

- Clientes
- Recursos
- Componentes
- Contenedores

Cliente. Un cliente es un programa que solicita un servicio de un componente.

Recurso. Es cualquier cosa que no necesita el componente para proporcionar el servicio.

Un servicio es cualquier recurso que recibe y atiende una petición de un cliente y dicho recurso podría a su vez tener que hacer peticiones a otros recursos para atender la petición de un cliente.

Componente. Cada componente es un servicio que se construye y mantiene en forma independiente de otros servicios, es decir, es parte de una capa que consiste en un conjunto de clases que realiza una función para proporcionar un servicio.

Contenedor. Es el software que gestiona al componente.

Las empresas de informática, o consultoras utilizan la arquitectura multicapa debido a que es la forma más efectiva de construir aplicaciones flexibles, escalables y que respondan a las expectativas de los usuarios.

Se conoce comúnmente como contrato a la relación entre un componente y un contenedor cuyos términos se rigen por una interfaz de programación de aplicaciones

(API)¹, esta define las reglas que debe seguir un componente y los servicios que recibe el componente del contenedor.

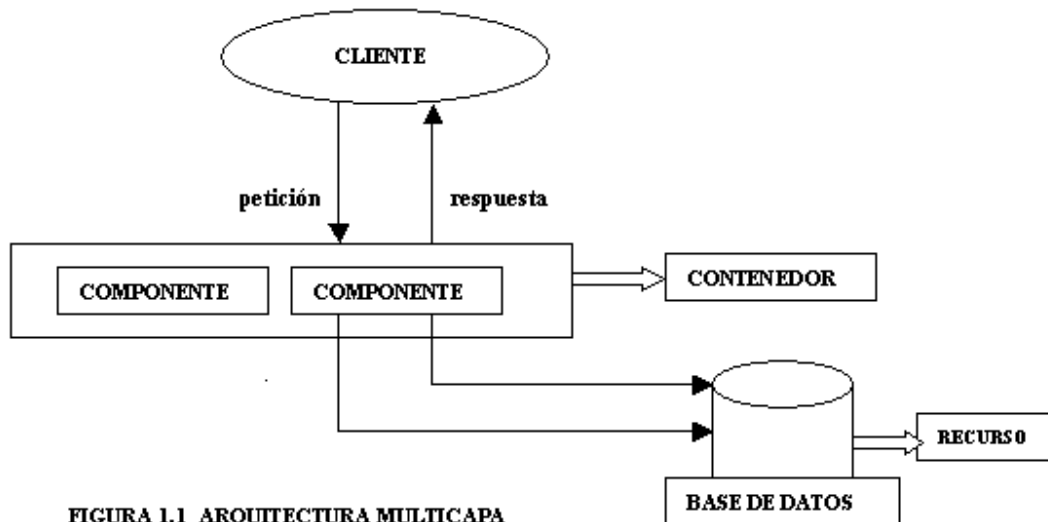


FIGURA 1.1 ARQUITECTURA MULTICAPA

En la figura 1.1 se ilustra como el cliente hace la petición y espera la respuesta, el contenedor se encarga de la gestión de recursos, la seguridad, los hilos y otros servicios a nivel sistema para los componentes asociados con él. Los componentes son responsables de implementar la lógica de negocio. Esto significa que los programadores se pueden concentrar en codificar las reglas de negocio en los componentes sin tener que preocuparse sobre los servicios de sistema de bajo nivel, ni gestionar los servicios de red.

La arquitectura J2EE consta de 4 capas:

- Cliente(capa de presentación o aplicación), componentes que se ejecutan en la maquina del cliente.
- Capa Web. Componentes que corren en el servidor J2EE(Servlets y JSP).
- La capa de Enterprise Java Beans(capa de negocio), componente que corre en el servidor J2EE.
- EIS (Enterprise Information System), software que corre en el servidor EIS.

Cada una de estas capas se enfoca en proporcionar a una aplicación un tipo especifico de funcionalidad.

¹ <http://java.sun.com/j2se/1.5.0/docs/api/>

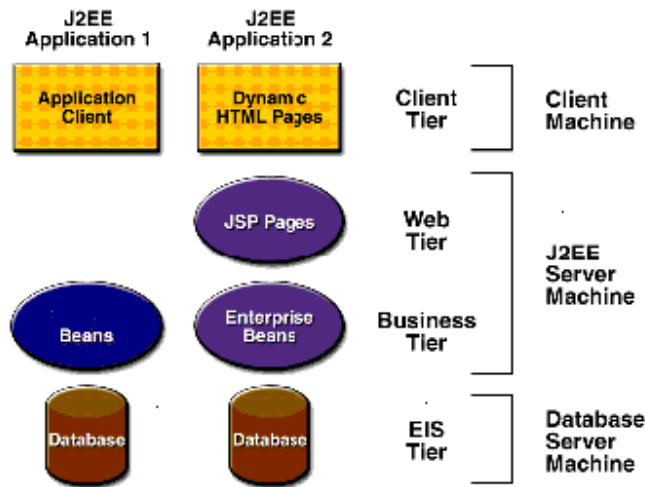


FIGURA 1.2 CAPAS DE APLICACION J2EE

Aunque una aplicación J2EE puede consistir en tres o cuatro capas como se muestra en la figura 1.2. La aplicación multicapa J2EE está generalmente considerada en tres capas de aplicación porque estas están distribuidas en tres diferentes localidades: máquina cliente, la máquina del servidor J2EE y la máquina del servidor de la base de datos.

La capa cliente o capa de presentación o aplicación está compuesta por los programas que interactúan con el usuario, estos programas piden datos al usuario y convierten su respuesta en peticiones que son reenviadas a un componente que procesa la petición y devuelve el resultado al programa cliente, es importante señalar que el componente puede operar en cualquier capa aunque la mayoría de las peticiones del cliente van dirigidas a componentes de la capa web. Una aplicación cliente corre en la máquina del cliente. El programa cliente también traduce la respuesta del servidor en texto y pantallas que se presentan al usuario.

Los componentes de la capa web utilizan HTTP para recibir peticiones y enviar respuestas a clientes que pueden estar en cualquier capa, esta capa proporciona a la aplicación J2EE las API's necesarias para interactuar con www, luego un componente de la capa web recibe una petición de datos del cliente y lo pasa a la capa Enterprise Java Bean este recibe los datos del DBMS en la capa de Enterprise Information System.

La capa de Enterprise Java Beans (EJB) o capa de negocios, contiene la lógica de negocios de las aplicaciones J2EE. Aquí se encuentran uno o más Enterprise JavaBeans, estos contienen reglas de negocios los cuales mandan a los clientes en forma indirecta. Esta capa es la pieza clave de toda aplicación J2EE, ya que los EJB que se encuentran en este nivel permiten múltiples instancias de una aplicación y tener acceso a la lógica y datos del negocio en forma concurrente de manera que no afecte a su rendimiento.

Introducción a J2EE

Aunque un EJB puede obtener acceso a componentes de cualquier capa, por lo general un EJB llama a componentes y recursos como el DBMS en la capa de sistemas de información empresarial (EIS).

En la capa de Enterprise Information System (EIS), conecta a las aplicaciones J2EE con recursos y sistemas heredados que se encuentran en la red de la empresa. En el EIS es donde la aplicación J2EE se conecta en forma directa o indirecta con distintas tecnologías, incluyendo DBMS y mainframes que forman parte de los sistemas de misión crítica que mantienen funcionando a la empresa.

NOTA: Cabe señalar que no nos enfocaremos en los EJB, solo utilizaremos beans Java que también simplemente les podemos llamar clases java.

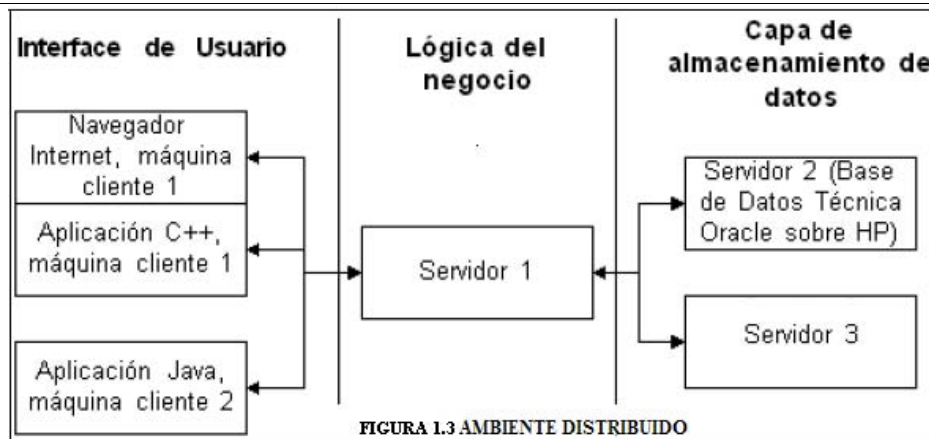
1.3 Características de la Arquitectura J2EE

La arquitectura J2EE esta enfocadas principalmente por las siguientes características:

- Arquitectura multicapa
- Ambiente distribuido
- Portabilidad
- Interoperabilidad
- Escalabilidad
- Alta disponibilidad y desempeño
- Simplicidad

En cuanto a la **arquitectura multicapa** es un conjunto de software que ofrece un específico servicio o funcionalidad. La arquitectura multicapa nos permite que los componentes de software puedan estar distribuidos en diferentes máquinas lo cual facilita la división de responsabilidad del desarrollo, despliegue y ejecución, además de que hay mayor seguridad y facilita la escalabilidad. Dependiendo del diseño de la aplicación un cliente puede acceder a la capa EJB o solo a la capa web y un elemento de la capa web puede acceder directamente a la base de datos o solo a través de la capa EJB tomando en cuenta que esto depende del diseño de la aplicación, esto nos deja varias opciones para nuestras aplicaciones tomando en cuenta la complejidad y tamaño de la aplicación. Comúnmente se le llama capa de presentación a la capa web ya que aquí es donde se manejan las entradas y resultados al cliente. En cuanto a la capa EJB también se le llama capa de negocios porque es aquí donde se implementa la lógica de negocios, finalmente la capa de datos es llamada EIS (Enterprise Information System).

El **ambiente distribuido** permite correr en diferentes partes los componentes de un sistema para que este desempeñe una tarea común. La arquitectura multicapa soporta un ambiente distribuido para las aplicaciones, es decir, mientras que los componentes que pertenecen a diferentes capas pueden correr en la misma máquina también pueden correr en distintas máquinas, las aplicaciones distribuidas suelen tener una estructura en la que se pueden distinguir esencialmente tres capas: **La interfaz de usuario** (paginas HTML), **la lógica de negocios**, que encapsula todo lo referente a como trabajar con clientes, cuentas, proveedores, etc. Y **la capa de almacenamiento de datos**, posiblemente implementadas en otras máquinas, típicamente mediante uno o más gestores de base de datos como se puede observar en la figura 1.3.



La **portabilidad** se refiere a la capacidad que tiene la aplicación en correr en distintas plataformas, es decir distintos sistemas operativos. La propuesta de J2EE de "unificar" el desarrollo de aplicaciones que pueden utilizarse en la empresa resulta viable y con posibilidades de amplia aceptación, sobre todo con el incentivo principal de Java: **Write Once, Run Anywhere(WODA)**- Escrito una vez ejecuta donde sea, es decir, nuestras aplicaciones pueden correr en cualquier sistema operativo. Entonces podemos hablar que es independiente del sistema operativo y también independiente de vendedor, esto debido a que J2EE esta basado en la plataforma Java y tiene la habilidad de que sus aplicaciones pueden ser ejecutadas y desplegadas en diferentes implementaciones de la plataforma J2EE por diferentes vendedores. Esto es una gran ventaja ya que no necesitamos modificar nuestro código para que podamos correr nuestra aplicación en cualquier sistema operativo.

La **interoperabilidad** de J2EE con plataformas diferentes a Java es gracias al soporte de los protocolos estándares de comunicación de Internet como son TCP/IP, HTTP, HTTPS y XML. Esto se refiere a la habilidad que tienen los componentes de software escritos en un lenguaje y ejecutadas en un ambiente J2EE puedan comunicarse con software escrito en otro lenguaje y corriendo en diferentes plataformas.

La **escalabilidad** se refiere a la capacidad que tiene para no degradar su confiabilidad y desempeño, es decir, si hay un gran incremento en el numero de clientes solo se necesita hacer pocas modificaciones o adiciones al sistema para satisfacer la demanda.

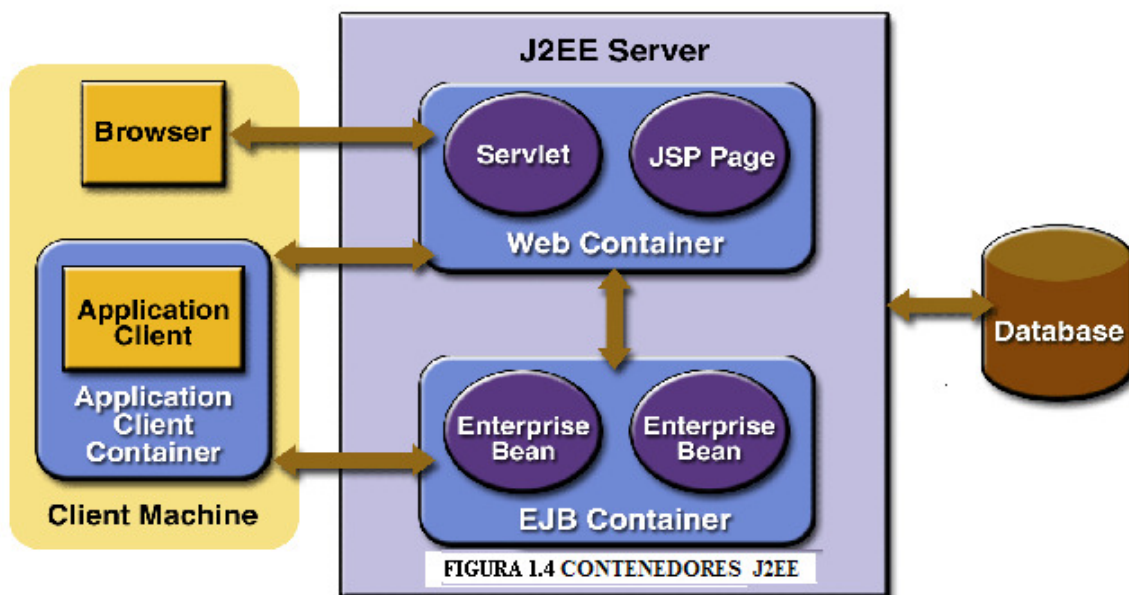
La **alta disponibilidad y alto desempeño** pueden ser implementadas a través de clusters y balanceo en la carga, los servidores pueden añadirse dinámicamente para satisfacer la demanda por lo tanto la alta disponibilidad y alto desempeño en este contexto son atributos del software ya que J2EE es un sistema de software.

Por último nos referimos a **simpleza** debido a que el desarrollador solo tiene que codificar la lógica de negocios, gracias a que las tareas de implementación a nivel de infraestructura de servicios como soporte a las transacciones, sockets, etc. Son echas por el servidor J2EE.

1.4 Contenedores de la plataforma J2EE

Los contenedores proporcionan el soporte para los componentes de la aplicación, interponer un contenedor entre el componente de la aplicación y el servicio J2EE permite a los contenedores inyectar transparentemente servicios definidos por los componentes tal como el manejo de transacciones y chequeo de seguridad. Antes de que un componente pueda ser ejecutado primero tiene que ser ensamblado dentro de una aplicación J2EE y después desplegado dentro del contenedor, es decir, un componente de una aplicación J2EE puede ser ejecutado solo por un contenedor. Un contenedor J2EE mantiene solo un tipo de componentes de aplicación J2EE, proveyendo a los componentes el ciclo de ejecución que ellos requieren desde su inicio hasta su fin. Algunas funciones que desempeñan los contenedores son las siguientes:

- El contenedor proporciona la API necesaria que ocupan los componentes.
- El contenedor ofrece servicios para los componentes tales como manejo de transacciones, seguridad, conectividad remota, etc.
- Proporciona métodos para los componentes de la aplicación para que interactúen uno con otro para que así los contenedores sean los mediadores de la comunicación.



A continuación se describe lo mostrado en la figura 1.4:

Contenedor Enterprise JavaBeans (EJB)

Administra la ejecución de Enterprise Java Beans para aplicaciones J2EE. Los Enterprise Beans y este contenedor corren en el servidor J2EE.

NOTA: En la presente tesis no nos enfocaremos en este contenedor EJB, ya que solo utilizaremos el contenedor web para llegar a nuestro objetivo.

Contenedor Web

Controlan o administran la ejecución de paginas JSP, Servlets y clases Java (beans) para una aplicación J2EE. Los componentes web y este contenedor corren en el servidor J2EE.

Contenedor de Aplicaciones Cliente

Controlan o administran la ejecución de componentes de aplicaciones del cliente. Las aplicaciones cliente y este contenedor corren en el cliente.

Con todo esto los desarrolladores de aplicaciones J2EE se pueden enfocar solo en la presentación y lógica de negocio, implementando el contenedor toda la infraestructura necesaria como la conectividad remota.

1.5 Tecnologías y servicios empleadas para desarrollar y soportar aplicaciones J2EE

J2EE nos ofrece soporte de principio a fin en aplicaciones desde la capa cliente hasta la capa empresarial. Un servidor J2EE simplifica el desarrollo, despliegue y ejecución de una aplicación. Los servidores de aplicaciones reciben el mismo estándar definido por la especificación J2EE. Esto se basa en un principio llamado "**WODA**", es decir, una aplicación J2EE puede ser ejecutada y desplegada por un servidor J2EE de cualquier vendedor ya que como se mencionó reciben el mismo estándar como los servidores de aplicaciones Websphere de IBM, Weblogic de BEA y Jboss.

1.5.1 Tecnologías J2EE

Es necesario utilizar una variedad de tecnologías para desarrollar aplicaciones empresariales las cuales funcionan en distintas capas estas tecnologías son:

- Tecnología cliente
- Tecnologías web
- Tecnología lógica de negocios
- Tecnología para interactuar con la base de datos

Tecnologías cliente

Es un componente de software que se encuentra en la capa cliente hay dos tipos de clientes: de aplicación y web.

El cliente de aplicación se puede comunicar directamente con un componente de software en la capa EJB o con un componente en la capa web. Un componente de aplicación puede tener una interfaz gráfica de usuario (GUI) o una interfaz de línea de comandos, el cliente de aplicación son desarrolladas en lenguaje Java.

Un cliente web es un componente de software en la capa cliente que se comunica con los componentes en la capa web usando http. Puede ser un browser o un programa Java que abre una conexión.

Tecnologías web (Tecnologías de presentación)

Estas tecnologías son usadas para desarrollar componentes que trabajan en la capa web, estos componentes son los que reciben la petición cliente y después se comunican con la capa EJB o pueden directamente interactuar con la base de datos. Estos al recibir la petición se encargan de dar una respuesta y la envían al cliente web, estos componentes de software se desarrollan utilizando la tecnología Servlet y JSP.

Un Servlet es una clase de Java que se encuentra en la capa web que tienen la función de dar respuesta a las peticiones echas por un cliente web a través del protocolo HTTP. El contenedor de Servlets se asocia al Servlet con un URL.

Los JSP's son un documento de script y este es traducido a un Servlet por el contenedor de JSP y Servlets (servidor web).

La lógica de negocio que utilizan las JSP y Servlets se encuentran en uno o más Enterprise JavaBeans. El código es el mismo tanto para las JSP como para los Servlets, aunque el formato del código es distinto. Las JSP utilizan etiquetas propias para llamar a un EJB, mientras que los Servlets pueden llamar a un EJB directamente.

Tecnologías de lógica de negocios

Como ya se menciona J2EE utiliza tecnología de objetos distribuidos para permitir a los desarrolladores Java construir aplicaciones portables, escalables y eficientes que cumplan con la disponibilidad 24x7 que se espera de un sistema empresarial. La capa de Enterprise JavaBeans contiene al servidor de Enterprise JavaBeans, que es el servidor de objetos que almacena y gestiona los EJB.

En estas tecnologías se desarrollan componentes de software en la capa EJB o comúnmente conocida como capa de negocios, estos componentes de la capa de negocios son desarrollados con la tecnología llamada Enterprise Java Beans (EJB's), estos implementan la lógica de negocios de la aplicación.

Tecnologías para interactuar con base de datos

La capa EIS representa la conexión de la arquitectura J2EE. Estos pueden incluir una gran variedad de recursos, como pueden ser los sistemas heredados, los DBMS y los sistemas de terceros a los que los componentes de la infraestructura J2EE pueden tener acceso.

En esta tecnología los componentes de software en la capa web o en la capa EJB usan la API para base de datos en dado caso de que ellos quieran comunicarse y manipular los datos directamente con la base de datos. Un controlador JDBC implementa la interfaz para una base de datos específica. Al separar la API de los controladores específicos, los desarrolladores pueden cambiar la base de datos subyacente sin necesidad de modificar el código de Java para tener acceso a la base de datos. La mayoría de los sistemas de administración de base de datos incluyen controladores

JDBC. La plataforma J2EE requiere el uso de la API JDBC para comunicarse con las bases de datos.

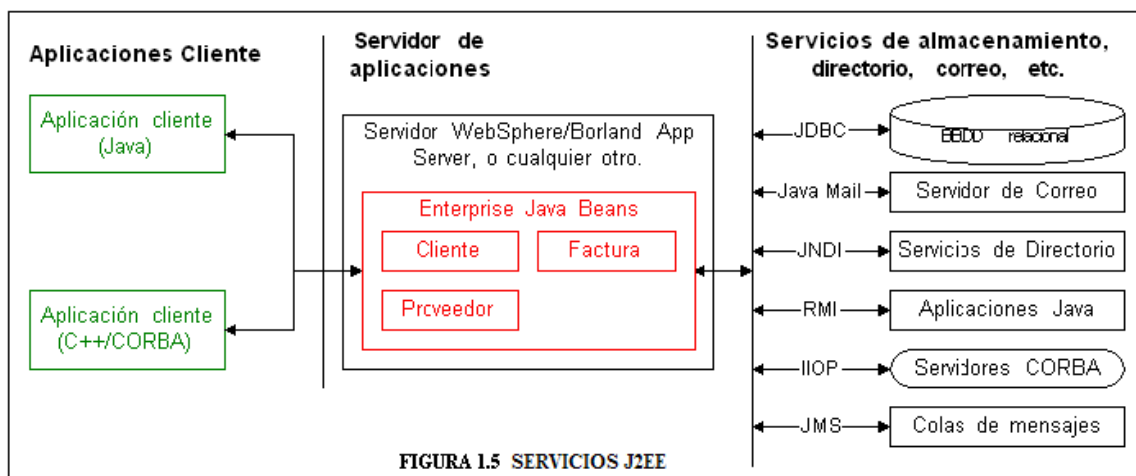
Los componentes que proporcionan a la aplicación J2EE el acceso a base de datos interactúan con los DBMS comerciales a través de una combinación de objetos de Java que se definen en la especificación de la conexión Java con base de datos (Java DataBase Connection o JDBC) y mediante el lenguaje estructurado de consultas (SQL). Los objetos Java construyen un enlace de comunicación con un DBMS, mientras que el SQL es el lenguaje que utiliza para construir el lenguaje (consulta) que se envía al DBMS para solicitar, actualizar, eliminar o manipular los datos que contiene el DBMS.

1.5.2 Servicios proporcionados por J2EE

Para que los componentes de distintas capas puedan estar ejecutándose en distintas máquinas (estar distribuidas) o se puedan comunicar con componentes de distintas capas, la especificación J2EE requiere un conjunto de servicios estándar que un desarrollador o vendedor de productos J2EE debe implementar, estos son usados para soportar las aplicaciones J2EE. Estos servicios son:

- Servicio de nombres (Naming Server)
- Servicio para transacciones
- Servicio de mensajes
- Servicio de correo
- Servicio de acceso remoto
- Servicio de procesamiento XML
- Servicio de seguridad
- Servicio de acceso a información JDBC

La figura 1.5 muestra como interactúan estos servicios con las otras capas:



Naming Server

Una aplicación J2EE es una colección de objetos que se localiza ya sea en la máquina local o máquinas remotas a las que se tiene acceso a través de la red. Para utilizar los objetos los programadores hacen referencia a sus nombres dentro de la aplicación. Como ya mencionamos los componentes y servicios pueden estar en distintas máquinas implementando un ambiente distribuido que la plataforma J2EE ofrece, se necesita un mecanismo de búsqueda que se utiliza para localizar componentes y servicios. Java Naming and Directory Interface (JNDI) proporciona este mecanismo. Por ejemplo, un cliente usaría JNDI para encontrar un servicio en otra máquina, de forma que los programadores puedan buscar objetos desde dentro de sus propios programas Java, sin necesidad de seguir el rastreo de la ubicación de los objetos.

Servicios para Transacciones

Una transacción puede involucrar varios componentes y el equipo de desarrollo de Java necesitaba alguna forma en que los componentes pudieran gestionar sus propias transacciones. El equipo creó la API Java para transacciones (JTA) para permitir a los programadores incluir código para gestionar las transacciones dentro de los componentes.

Al hablar de transacciones nos referimos a accesos concurrentes a la base de datos en la cual la parte más importante es mantener la integridad de los datos durante estos accesos concurrentes, por ello el manejo de transacciones para los componentes de aplicación es una tarea importante. JTA (API Java Transaction) nos ayuda a mantener la integridad de los datos durante estos accesos concurrentes a la base de datos.

Servicio de mensaje

En nuestras aplicaciones empresariales se necesitan intercambiar notificaciones de ciertos eventos a otras aplicaciones, es decir, intercambiar información entre aplicaciones, para facilitar el envío de mensajes en ambos modelos: punto a punto (point to point) y publicación por suscripción (publish-subscribe) se utiliza Java Message Service (JMS).

publish-subscribe. El mensaje enviado por el transmisor puede ser recibido por múltiples receptores llamados suscriptores.

point to point. Un mensaje es enviado solo por un transmisor y es recibido solo por un receptor.

Servicio de correo

Este servicio de correo es implementado por Java Mail API, ya que el servicio de correo en ambientes empresariales es muy necesario para mantenerse comunicados e informados en el ambiente empresarial, es decir, se necesita una forma eficiente para que los clientes y los sitios de comercio electrónico puedan intercambiar información, las confirmaciones de pedido y la API Java Mail implementa toda la funcionalidad requerida.

Servicio de procesamiento de XML

Muchas empresas y algunas industrias han adoptado XML como una forma de almacenar, manipular e intercambiar información textual contenida en documentos.

El servicio de procesamiento de lenguaje de marcado extendido (XML) es una parte integral de aplicaciones J2EE. La API de Java para XML (JAXP) es usada para escribir e interpretar archivos XML. JAXP provee soporte para herramientas de XML tales como la API simple para XML (SAX), Modelo de Objeto de Documento (DOM), y XML Stylesheet Language para Transformations (XSLT).

Servicio de Seguridad

Una aplicación J2EE es segura gracias a las características que forman parte de las clases fundamentales de Java, de la JVM y el lenguaje de programación Java.

Para controlar, autenticar y autorizar el acceso de un usuario en una aplicación empresarial utilizamos el servicio de seguridad Java Authentication Authorization Service (JAAS) provee este tipo de seguridad para ofrecer un mecanismo para autenticación de usuarios o un grupo de usuarios que quieren acceder a las aplicación aplicaciones empresariales.

Servicio de acceso remoto

En ambientes en donde las aplicaciones y componentes residen en distintas máquinas o comúnmente llamados ambientes distribuidos, un componente en una máquina puede necesitar invocar un método de un componente en otra máquina, lo cual es conocido como invocación remota. El Standard RMI (Remote Method Invocation) oculta los detalles de red, proveyendo a los objetos que invocan métodos remotos transparencia en la invocación. La API de Java proporciona soporte para RMI la cual nos facilita el uso de invocaciones remotas de métodos simplificándonos el uso de detalles específicos de red.

Un objeto Java se ejecuta dentro de la Máquina Virtual Java (JVM), una aplicación J2EE también se ejecuta dentro de una JVM. Sin embargo, los objetos que utilizan una aplicación J2EE no tienen que ejecutarse en la misma JVM que la aplicación J2EE. Esto se debe a que una aplicación J2EE y sus componentes pueden llamar objetos ubicados en una JVM distinta mediante el sistema de Invocación remota de métodos de Java (Remote Method Invocation o RMI). RMI se utiliza para la comunicación remota entre aplicaciones y componentes Java. Ambas aplicaciones o componentes tienen que estar escritos en el lenguaje Java.

RMI se utiliza para conectar un cliente con un servidor. Un cliente es una aplicación o componente que requiere los servicios de un objeto distinto para atender una petición. Un servidor crea un objeto y lo pone a disposición de los clientes. Un cliente contacta con el servidor para obtener una referencia y llamar al objeto mediante RMI.

JDBC

Muchos programas y componentes Java tienen que obtener acceso a información contenida en una base de datos. El equipo de desarrollo de Java creó la API JDBC, la cual permite a un programa conectarse e interactuar con prácticamente cualquier DBMS comercial.

1.6 Introducción a los servicios de Internet

Desde la aparición del servicio de World Wide Web (WWW) en el año de 1989, el uso de las computadoras para el intercambio de información tuvo un crecimiento masivo. El World Wide Web fue creado en 1989 en el Laboratorio Europeo de Física de Partículas, que se encuentra en Génova, Suiza, el cual tenía la capacidad de usar hipervínculos, los cuales permiten hacer referencias a otros documentos, desde un documento actual.

La infraestructura interna del World Wide Web está construida sobre un conjunto de reglas llamado Protocolo de transferencia de hipertexto (Hypertext Transfer Protocol: HTTP) y un lenguaje de marcación de hipertexto (Hypertext Markup Language: HTML). HTTP utiliza direcciones de Internet en un formato especial llamado localizador uniforme de recursos (Uniform Resource Locator: URL). Los URL son parecidos a lo siguiente:

Protocolo:// Nombre de dominio:puerto/directorios/recurso

Protocolo: también llamado esquema URL, especifica que protocolo es utilizado para acceder al documento.

Nombre de computadora o dominio: Especifica el nombre de la computadora (usualmente un nombre de dominio o una dirección IP) donde el contenido está alojado.

Puerto: Especifica el puerto en el cual está corriendo el servidor.

Directorios: Secuencia de directorios separados por barras ("/") que define la ruta a seguir para llegar al documento.

Archivo o recurso: El nombre del archivo donde el recurso se encuentra ubicado.

Tim Berners-Lee fue quien desarrolló HTTP como un protocolo que funciona para distribuir documentos de una máquina a otra y diseñó el primer navegador. En 1993, la llegada del navegador Mosaic provocó la explosión del uso comercial de la Web. En un plazo de cinco años ya había más de 650,000 servidores Web en todo el mundo.

1.7 ¿Qué es y que hace un Servidor Web?

1.7.1 Servidor web

Las aplicaciones que se encargan de enviar páginas Web utilizando el protocolo HTTP, reciben el nombre de **servidores Web**.

Un servidor Web actúa poniéndose a la escucha esperando peticiones en un número de puerto concreto y muy conocido, normalmente el número 80, aunque se puede utilizar cualquier otro puerto disponible. Si un servidor Web escucha en un puerto diferente, las URL que van dirigidas a este servidor deben incluir dos puntos (:) y el número de puerto inmediatamente después del nombre del servidor.

Por ejemplo,

```
http://www.mipagina.com/documento.html
```

señala a un documento HTML que será atendido por un servidor Web que opera en el servidor host `www.mipagina.com` del puerto predeterminado 80. Pero si el servidor opera en el puerto 8080, la URL será:

```
http://www.mipagina.com:8080/documento.html
```

1.7.2 ¿Qué hace un servidor Web?

Entre sus funciones principales están:

- Aceptar la peticiones enviadas por clientes (navegadores web por ejemplo) solicitando recursos como páginas web, imágenes.
- Transmitir datos como son las imágenes y las páginas HTML.

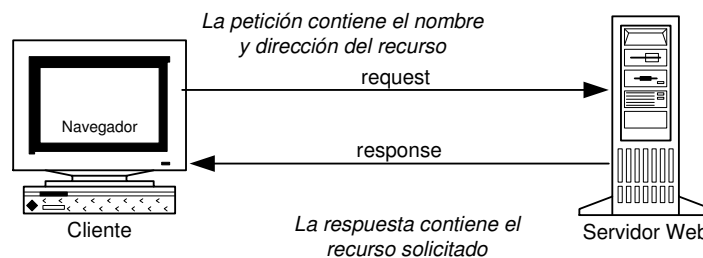


FIGURA 1.6 PETICIÓN / RESPUESTA A UN SERVIDOR WEB

En la figura 1.6 se muestra como el navegador(cliente) hace una petición la cual contiene el nombre y dirección del recurso, esta va al servidor el cual procesa la petición y envía una respuesta al cliente.

1.8 Métodos de petición (HTTP POST Y HTTP GET)

1.8.1 ¿Qué es HTTP?

Mientras el Lenguaje de marcado de hipertexto (HTML) es el lenguaje empleado para describir el interior de los documentos Web, el Protocolo de transferencia de hipertexto (HTTP) es el lenguaje empleado para describir cómo se envían estos documentos por Internet.

La estructura de una conversación HTTP es una simple secuencia de petición/respuesta; un navegador hace una petición y un servidor responde como se muestra en la figura 1.7:

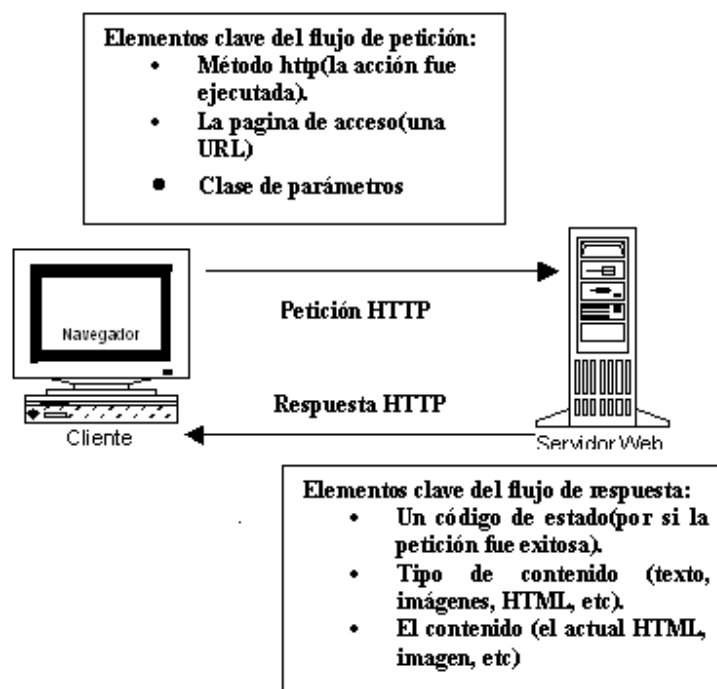


FIGURA 1.7 ESTRUCTURA DE UNA PETICIÓN HTTP

HTTP proporciona las normas para que los navegadores hagan peticiones y los servidores entreguen respuestas. Este conjunto de normas, o protocolo, incluye la manera de:

- Solicitar un documento por su nombre
- Ponerse de acuerdo en el formato de los datos

- Determinar quién es el usuario
- Decidir cómo manejar recursos obsoletos
- Indicar los resultados de la petición

Por lo tanto, podemos decir que HTTP consiste en un conjunto de comandos para realizar las peticiones de documentos desde el cliente, incluyendo los parámetros que se les pasa al servidor, los comandos están escritos como líneas de texto ASCII ordinario. Cuando se utiliza un navegador Web, no se tiene acceso directo a los comandos http. Sin embargo, al escribir una dirección URL o al hacer clic en un hipertexto el navegador convierte su acción en comandos http que piden el documento al servidor que se está especificando en la URL. El servidor Web encuentra el documento y lo envía al navegador, el cual lo muestra al usuario.

1.8.2 Especificación HTTP

Las normas establecidas para Internet se especifican normalmente en un documento RFC (Request For Comments) editado por la Internet Engineering Task Force (IETF). Los RFC están ampliamente aceptados por la comunidad de desarrollo e investigación en Internet. Estos documentos están escritos en un lenguaje formal, además de estar numerados y con la característica de que nunca cambian una vez publicados.

Algunos de los principales RFC dedicados a http:

RFC 1945	Una descripción de http versión 1.0
RFC 2068	La descripción inicial de la versión 1.1
RFC 2616	Una versión actualizada de la especificación 1.1

La especificación de HTTP lo describe como un protocolo de petición/respuesta sin estado cuya operación básica es la siguiente:

Una aplicación cliente, por ejemplo un navegador Web, abre un socket al puerto HTTP del servidor Web (el puerto predeterminado es el 80).

A través de la conexión el cliente escribe una línea de petición de texto ASCII, seguida de ninguna, una o varias cabeceras http, una línea en blanco y cualquier dato que acompañe a la petición.

El servidor Web analiza la petición y localiza el recurso especificado.

El servidor envía una copia del recurso al socket, donde es leído por el cliente.

El servidor cierra la conexión.

Un punto importante de este protocolo es que carece de estado. Esto significa que al manejar una petición el servidor Web no recuerda nada sobre las peticiones previas procedentes del mismo cliente. El protocolo es simplemente una petición y una respuesta. Esto supone un problema al programar aplicaciones, ya que se necesita guardar información del cliente en el servidor, y de los datos que se han ido generando o que se generan durante el flujo del programa. Este problema se soluciona mediante el uso de sesiones o la utilización de campos ocultos.

1.8.3 El envío de la petición HTTP

Una vez hecha la conexión, el navegador Web escribe un comando http para solicitar el documento. Una petición puede constar de hasta cuatro partes:

La primera parte es la línea de petición. Consta de tres categorías, separadas por espacios: el método de la petición, el URI (identificador uniforme de recursos) de la petición y la versión http.

Después de la línea de petición llegan las cabeceras de petición. Se trata de pares clave/valor, separados por dos puntos (:) de modo que cada par ocupa una línea. Tras la última cabecera de petición se envía una línea en blanco que consiste únicamente en un retorno de carro. Esto informa al servidor de que ya no quedan más cabeceras. Incluso cuando no hay ninguna cabecera, se debe enviar esta línea en blanco para que el servidor no busque cabeceras.

Las cabeceras de petición proporcionan al servidor información adicional sobre la identidad y las capacidades del cliente. Las cabeceras de petición más comunes podrían ser:

User-Agent	La marca y la versión del cliente (o navegador Web)
Accept	Una lista de tipos de contenido que el cliente reconoce
Content-Length	El número de bytes de datos anexados a la petición

Aceptación de la petición por parte del servidor

Cuando un cliente se conecta al puerto de escucha del servidor Web, el servidor acepta la conexión y gestiona la petición. La mayoría de las veces lo hace iniciando un subproceso de manera que pueda continuar atendiendo nuevas peticiones. Cuando se gestiona una petición implica dos cosas diferentes dependiendo del URI. Si el URI representa un documento estático, el servidor abre el fichero del documento y se prepara para enviar su contenido al cliente. Si el URI es un nombre de programa, como una secuencia de comandos CGI, un servlet o una página JSP y el servidor está configurado para manejar una petición de ese tipo, entonces el servidor se prepara para invocar al programa o al proceso.

Sea cual sea la manera en que el servidor procesa la respuesta, el resultado que se obtiene es el mismo: una respuesta HTTP. La respuesta HTTP, similar a la petición, consta de un máximo de cuatro partes: una línea de estado; ninguna, una o varias cabeceras de respuesta; una línea en blanco que señala el final de las cabeceras; y los datos que conforman la petición.

La línea de estado tiene hasta tres categorías:

La **versión HTTP**. Del mismo modo que el cliente indica la versión más avanzada que es capaz de entender, también el servidor indica sus capacidades.

El **código de respuesta**. Es un código numérico de tres dígitos que indica si la petición se ha realizado correctamente y, en caso de fallo, la razón por la que éste se ha producido.

Una **descripción opcional de la respuesta**, que es una explicación del código de respuesta comprensible para el usuario.

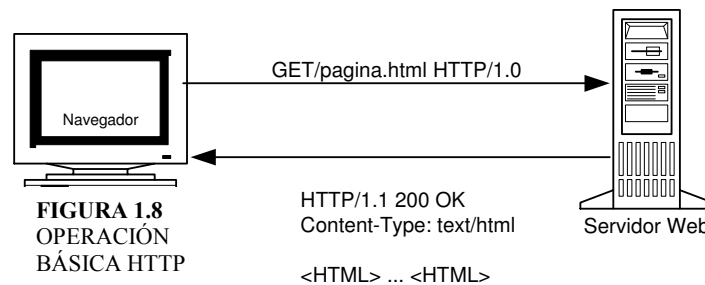
Para poder solicitar un recurso lo hacemos a través de alguno de los métodos de petición que nos proporciona HTTP.

1.8.4 Métodos de petición

A continuación se listan los métodos de petición:

Método	Descripción
GET	Una petición simple para recuperar el recurso identificado en el URI.
HEAD	Lo mismo que GET, excepto en que el servidor no devuelve el recurso solicitado. El servidor sólo devuelve la línea de estado y las cabeceras.
POST	Una petición dirigida al servidor para que acepte datos que se enviarán al flujo de salida del cliente.
PUT	Una petición dirigida al servidor para que guarde los datos de la petición como los nuevos contenidos del URI especificado.
DELETE	Una petición al servidor para que elimine el nombre del recurso del URI.
OPTIONS	Una petición de información sobre los métodos de petición que soporta el servidor.
TRACE	Una petición dirigida al servidor Web para que nos devuelva la petición HTTP y sus cabeceras.
CONNECT	Un método documentado pero que no se utiliza normalmente, reservado para su uso con un proxy de tunelización.

Ejemplo de operación básica de HTTP:



Los dos métodos de petición más usados son **GET** y **POST**. GET sirve para solicitar recursos principalmente, y POST sirve para enviar datos que modifican la base de datos, subir archivos, etc.

1.8.5 Common Gateway Interface (CGI)

Con el desarrollo del servidor Web NCSA HTTPd llegó una nueva especificación denominada Interfaz de pasarela común (Common Gateway Interface).

El servidor Web invoca a un programa CGI como respuesta a ciertos tipos de peticiones, generalmente peticiones de documentos de un directorio concreto o nombres de ficheros con una extensión específica, como por ejemplo .cgi. Los parámetros de la petición se pasan como pares clave/valor y las cabeceras de respuesta como variables de entorno. El programa lee estos parámetros y las cabeceras, realiza la tarea de la aplicación con la que está trabajando (normalmente obtiene acceso a una base de datos para hacerlo), y entonces genera una respuesta HTTP. La respuesta se envía al navegador Web solicitante como si fuera un documento estático corriente.

CGI es muy útil, pero tiene un grave inconveniente. Normalmente genera un nuevo proceso para cada petición HTTP. Esto no supone un problema cuando el tráfico es escaso, pero provoca sobrecarga cuando el nivel de tráfico aumenta.

Con la aparición en 1997 de la API Servlet de Java, que fue seguida rápidamente por la API JSP (Java Server Pages, Páginas Java en servidor), se produjo una mejora significativa. Pronto le siguió al año siguiente la tecnología EJB (Enterprise Java Beans) para soporte de transacciones en bases de datos distribuidas con una escalabilidad y robustez que se necesitan para el comercio electrónico. Las tecnologías anteriores llevan todo el potencial de Java al servidor Web, con conectividad a base de datos, acceso a trabajo en red, operaciones de subprocesos múltiples y, sobre todo, un modelo de proceso diferente.

Los servlets y las páginas JSP operan desde un solo ejemplar o instancia que permanece en la memoria y utiliza múltiples subprocesos para responder a distintas peticiones de forma simultánea.

Las especificaciones servlet, JSP y EJB forman parte del entorno Java 2 en su edición para empresas (J2EE, Java 2 Enterprise Edition).

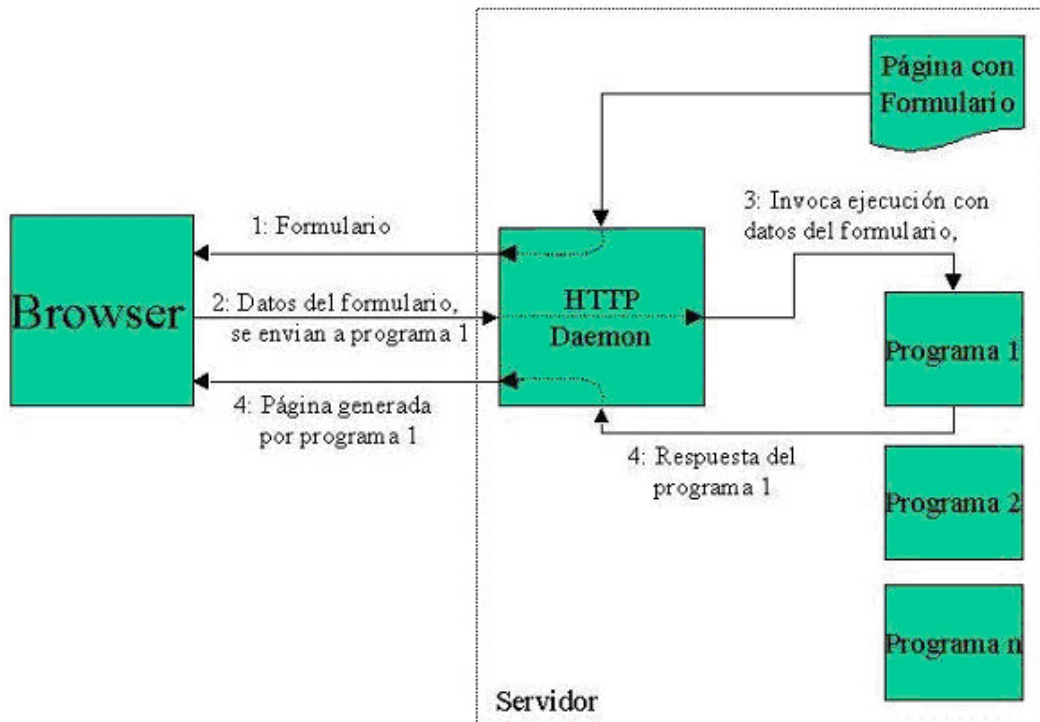


FIGURA 1.9 ESQUEMA EJECUCIÓN CGI

Cada vez que se envían los datos de un formulario se crea un nuevo proceso en el servidor para ejecutar el programa en cuestión, quien genera una respuesta que se envía a través del servidor HTTP al navegador, luego de lo cual se elimina el proceso creado. Esta creación de procesos implica una carga importante para el servidor, por lo cual esta modalidad de generación de contenido dinámico no es muy escalable.

Entre las acciones más comunes de los programas CGI está el acceso a Bases de Datos, ya que de esta forma se pueden acceder a los datos desde su localización estándar y ser presentados y eventualmente modificados a través de páginas WEB.

La comunicación desde el navegador al servidor con los datos del formulario se puede hacer en dos modalidades, a través de variables de entorno (GET) o a través de la línea de comando (POST). Específicamente, los datos van en la variable QUERY_STRING en los envíos con GET y en la entrada estándar en los envíos con POST.

MANIPULACIÓN DE BASE DE DATOS CON JDBC

2.1 Introducción

Como ya se menciona en el capítulo anterior para construir una aplicación J2EE se ensamblan componentes, los cuales se comparten con muchas aplicaciones, cada componente proporciona a la aplicación J2EE un servicio, una de los cuales es el acceso a una o mas base de datos.

Una base de datos es una colección organizada de datos. Existen diversas estrategias para organizar datos y facilitar el acceso y la manipulación. Un sistema de administración de base de datos (database management system o DBMS) proporciona los mecanismos para almacenar y organizar datos en una manera consistente con el formato de la base de datos. Los DBMS permiten el acceso y almacenamiento de datos sin necesidad de preocuparse por su representación interna.

Los sistemas de base de datos mas populares son las base de datos relacionales. El lenguaje estructurado de consultas SQL(Structured Query Language) es el lenguaje estándar internacional que se utiliza casi universalmente con las base de datos relacionales para realizar consultas, es decir, para solicitar información que satisfaga ciertos criterios y para manipular datos. Algunos sistemas de administración de base de datos relacionales (RDBMS) mas populares son Microsoft SQL Server, Oracle, Sybase, DB2, Informix y Mysql.

El desafío que presentaba Sun Microsystems a fines de la década de los noventa era desarrollar alguna forma de que los desarrolladores de Java pudieran crear código de alto nivel para que pudiera tener acceso a todos los DBMS comerciales. Uno de los obstáculos era precisamente la barrera del lenguaje. Cada DBMS utilizaba su propia forma de bajo nivel de interactuar con los programas para que estos puedan tener acceso a los datos almacenados en sus bases de datos. Esto significaba que se tendría que rescribir el código para comunicarse con una base de datos Mysql si se necesitara tener acceso a una base de datos en Oracle.

La solución para este problema fue la creación de los controladores del JDK y la API JDBC, estos se crearon por necesidad. El controlador JDBC que desarrollo Sun Microsystem no era de hecho un controlador, en vez de ello era una especificación que indicaba a detalle la funcionalidad de un controlador JDBC. Se animo a los fabricantes de DBMS y a terceros a crear controladores JDBC que se ajustaran a las especificaciones de Sun Microsystem. Aquellas empresas que construyeran controladores JDBC para sus productos podrían aprovechar el creciente mercado de aplicaciones Java.

2.2 ¿Qué es JDBC?

JDBC (Java DataBase Connection). La especificación indica que un controlador JDBC es un traductor que convierte los mensajes propietarios de bajo nivel del DBMS a mensajes de bajo nivel comprensibles a la API JDBC y viceversa.

El significado de esto es que los programadores Java pueden utilizar objetos Java de alto nivel definidos en la API JDBC para escribir código que interactúe con el DBMS. Los objetos Java convierten el código en mensajes de bajo nivel que se ajusta a la especificación del controlador JDBC y los envía a éste último. El controlador JDBC traduce los mensajes en mensajes de bajo nivel que el DBMS comprende y procesa.

Los programadores Java se comunican con las bases de datos y manipulan sus datos utilizando API JDBC. Un controlador de JDBC implementa la interfaz para una base de datos específica. Al separar la API de los controladores específicos, los desarrolladores pueden cambiar la base de datos subyacentes sin necesidad de modificar el código de Java para tener acceso a la base de datos como se ilustra en la figura 2.1:

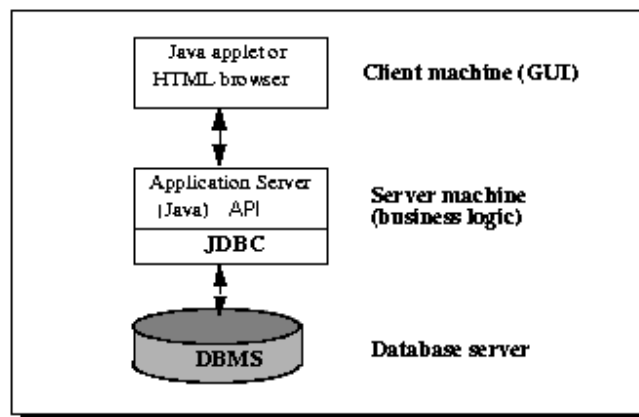


FIGURA 2.1 COMUNICACIÓN CON LA DB UTILIZANDO JDBC

Los controladores JDBC que generan los fabricantes deben de poder:

- Abrir una conexión entre el DBMS y el componente JDBC
- Traducir equivalentes de bajo nivel de instrucciones SQL que envía el componente J2EE en lenguaje que el DBMS pueda procesar.
- Devolver al controlador JDBC datos que se ajustan a la especificación JDBC.
- Devolver al controlador JDBC información, como puede ser mensaje de error, que se ajuste a la especificación JDBC.
- Proporcionar rutinas de gestión de transacciones que ajusten a la especificación JDBC.
- Cerrar conexión entre el DBMS y el componente J2EE.

El controlador JDBC hace que los componentes J2EE sean independientes de la base de datos, lo cual se ajusta a la filosofía Java de independencia de plataforma. Actualmente existen controladores JDBC para casi todos los DBMS comerciales, estos se pueden encontrar en la página oficial de Sun o en los sitios web del fabricante DBMS.

2.3 Tipos de controladores JDBC

JDBC soporta cuatro categorías de controladores, es decir, la especificación del controlador JDBC clasifica los controladores en cuatro grupos:

- Controlador puente JDBC a ODBC (tipo 1).
- Controlador nativo de API desarrollado parcialmente en Java (tipo 2).
- Controlador JDBC-NET desarrollado puramente en Java (tipo 3).
- Controlador de protocolo nativo desarrollado puramente en Java (tipo 4).

Cada grupo se conoce como tipo de controlador JDBC y resuelve una necesidad específica al comunicarse con distintos DBMS. A continuación se describe cada uno de ellos:

Controlador tipo 1 puente JDBC a ODBC

Microsoft fue la primera empresa en encontrar una forma de crear un programa de base de datos independiente del DBMS, con la creación de la conexión abierta a base de datos (Open Database Connection u ODBC). ODBC y JDBC tienen especificaciones similares en cuanto a controladores y a API. El controlador puente JDBC a ODBC conecta los programas de Java con orígenes de datos de Microsoft ODBC. El controlador JDBC a ODBC se utiliza para traducir llamadas a DBMS entre la especificación JDBC y la especificación ODBC. Este controlador recibe el mensaje de un componente J2EE que utiliza la especificación J2EE, y traduce dichos mensajes al formato de ODBC el cual a su vez se traduce al formato de mensaje que comprende el DBMS. El kit de desarrollo de software para Java 2 de Sun incluye el controlador puente JDBC a ODBC(`sun.jdbc.odbc.JdbcOdbcDriver`). este controlador por lo general requiere que el controlador ODBC este instalado en el equipo cliente, y normalmente es necesario que se configure el origen de datos ODBC. El controlador puente se introdujo principalmente para fines de desarrollo, por lo cual no debe utilizarse en aplicaciones de producción, es decir, no debe utilizarse en aplicaciones de misión crítica, ya que el paso adicional de traducción puede tener un fuerte impacto en el rendimiento.

Controlador tipo 2 Controlador Java/Código nativo.

Los controladores nativos de API desarrollados parcialmente en Java permiten a los programadores de JDBC utilizar APIs para base de datos específicas (generalmente escritas en C o C++), las cuales permiten a los programadores clientes utilizar base de datos mediante la interfaz nativa de Java (Java Native Interface), es decir, este controlador utiliza clases Java para generar código específico de la plataforma, código que solo comprende un DBMS determinado. Este tipo de controlador traduce el código JDBC en código de una base de datos específica. la desventaja obvia de utilizar un controlador tipo 2 es la pérdida de algo de portabilidad en el código. Las clases de la API Java/Código nativo probablemente no funcionarían con el DBMS de otro fabricante. Los controladores de tipo 2 se introdujeron por razones similares al controlador puente ODBC tipo1.

Controlador tipo 3 JDBC/Net

Los controladores JDBC-Net desarrollados puramente en Java toman las peticiones de JDBC y las traducen en un protocolo de red que no es para una base de datos específica. Estas peticiones se envían a un servidor, el cual traduce las peticiones de la base de datos en un protocolo específico para esa base de datos. El controlador tipo 3 es el controlador más utilizado, ya que convierte las consultas SQL en instrucciones en formato JDBC. Dichas instrucciones se traducen después al formato que requiere el DBMS.

Controlador tipo 4 JDBC/protocolo nativo

Los controladores de protocolo nativo desarrollados puramente en Java convierten las peticiones de JDBC en protocolos de red para base de datos específicas, de manera que los programas de Java puedan conectarse directamente con una base de datos. Este controlador es similar al tipo 3, salvo que las consultas SQL se traducen directamente al formato que requiere el DBMS. No es necesario convertir las consultas SQL al formato JDBC. Esta es la forma más rápida de enviar consultas SQL al DBMS.

Paquetes de JDBC

La API JDBC se encuentra en dos paquetes. El primer paquete se llama `java.sql`, y contiene los objetos Java fundamentales de la API JDBC. Estos incluyen los objetos que proporcionan el soporte básico para conectar con el DBMS e interactuar con los datos almacenados en él. El paquete `java.sql` forma parte de J2SE.

El otro paquete que incluye la API JDBC es el paquete `javax.sql` y forma parte de J2EE. El paquete `javax.sql` incluye los objetos que interactúan con la Interfaz de nombres y directorios para Java (Java Naming and Directory Interface o JNDI) y otros que gestionan pools de conexiones, entre otras características avanzadas de JDBC.

2.4 Visión general del proceso de JDBC

Aunque cada componente J2EE es distinto, los componentes J2EE utilizan un proceso similar para interactuar con un DBMS. Este proceso se divide en cinco partes:

- Cargar el controlador JDBC
- Conectar con el DBMS
- Crear y ejecutar una instrucción SQL
- Procesar los datos que devuelve el DBMS
- Terminar la conexión con el DBMS

Con esto nos daremos un panorama general de cómo funciona el proceso de JDBC.

2.4.1 Cargar controlador JDBC

Para que el componente JDBC se pueda conectar al DBMS es necesario cargar el controlador JDBC, para hacer esto se utiliza el método *Class.forName()*. Para explicar como utilizar este método supongamos que queremos desarrollar un componente J2EE que interactúe con Microsoft Access. Debemos escribir una rutina que cargue el controlador del puente JDBC/ODBC, el cual se llama *sun.jdbc.odbc.JdbcOdbcDriver*. Para cargar el controlador se llama al método *Class.forName()* y se le pasa un String con el nombre del controlador:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Si se desea utilizar otro DBMS como por ejemplo MySql tenemos que pasarle el nombre del controlador para MySql

```
Class.forName("com.mysql.jdbc.Driver");
```

La finalidad de cargar y registrar el controlador JDBC es incorporarlo a la Máquina Virtual de Java (JVM). Una vez cargado el controlador JDBC, este se registra en forma automática con el DriverManager con lo que se encuentra disponible a la JVM y por lo tanto los componentes J2EE lo pueden utilizar.

El método *Class.forName()* lanza una excepción *ClassNotFoundException* si ocurre un error al cargar el controlador JDBC. Para atrapar los errores es necesario utilizar un bloque `catch {}` cada vez que se carga un controlador.

2.4.2 Conectar con el DBMS

Una vez ya cargado el controlador, el componente J2EE se debe conectar con el DBMS mediante el método *DriverManager.getConnection()*. La clase *java.sql.DriverManager* (gestor de controladores) es la clase mas alta dentro de la jerarquía de JDBC y es la responsable de gestionar la información de los controladores.

El método *DriverManager.getConnection()* recibe el url de la base de datos y el identificador de usuario (login) y la contraseña de acceso, estos dos últimos solo se utilizan si es que el DBMS los requiere. El url es un objeto String que contiene el nombre del controlador y el nombre de la base de datos a la que el componente quiere tener acceso.

DriverManager.getConnection() devuelve un objeto que implementa la interfaz *Connection* (conexión), el cual se utiliza a lo largo del proceso para hacer referencia a la base de datos. La clase *java.sql.Connection* es otro miembro del paquete *java.sql*, el cual gestiona la comunicación entre la base de datos, el controlador JDBC y el componente J2EE.

La interfaz *java.sql.Connection* es la que envía instrucciones al DBMS para su procesamiento.

```
Private Connection con;
    try{

        // Cargamos el controlador.
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        con = DriverManager.getConnection("jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=C:/Proyectos/cursos/java/JDBC/agenda.mdb");
    }
}
```

En este ejemplo se muestra como conectarnos a una base de datos de Access llamada agenda.mdb, obsérvese que no se requiere el nombre de usuario y contraseña en el siguiente ejemplo se muestra como conectarse a una base de datos de MySql utilizando el nombre de usuario y contraseña.

```
String url = "jdbc:mysql://localhost:3306/Productos";
String usr = "root";
String pwd = "antar99";
private Connection con = null;

    try {
        // Carga del controlador
        Class.forName("com.mysql.jdbc.Driver");

        con = DriverManager.getConnection(url, usr, pwd);
    }
}
```

La conexión se establece mediante uno de los tres métodos de `getConnection()` que contiene la clase `DriverManager`. El método `getConnection` solicita al DBMS el acceso a la base de datos. Es responsabilidad del DBMS otorgar o negar el acceso. Si se permite el acceso, el método `getConnection()` devuelve un objeto `Connection`. De lo contrario el método `getConnection()` lanza una `SQLException`.

En ocasiones un DBMS puede requerir información adicional al identificador de usuario y clave de acceso para otorgar acceso a la base de datos. Esta información adicional se conoce como propiedades y se debe asociar a un objeto de tipo `Properties`, el cual se pasa a la base de datos como parámetro del método `getConnection()`.

Por lo general las propiedades necesarias para tener acceso a una base de datos se almacenan en un archivo de texto cuyo contenido define el fabricante del DBMS. El componente J2EE utiliza un objeto de tipo `FileInputStream` para abrir el archivo y luego utiliza el método `load()` del objeto `Properties` para copiar las propiedades a un objeto `Properties`. A continuación se muestra un segmento de código para ejemplificar lo anterior:

```
Connection db;
Properties props = new Propierties ();
Try{
    FileInputStream propFileStream = new FileInputStream ("DBProps.txt");
    Props.load(propFileStream);
}
catch(IOException err){
    System.out.println("Error al cargar las propiedades");
}
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    con = DriverManager.getConnection(url, props);
    }
catch (ClassNotFoundException error){
    System.out.println("No se puede cargar el puente JDBC a ODBC");
}

```

2.4.3 Crear y ejecutar una instrucción SQL

Una vez ya esta cargado el controlador y establecida una conexión con el DBMS lo que necesitamos es interactuar con la base de datos para recuperar datos específicos de la base de datos, necesitamos enviar una consulta SQL al DBMS para su procesamiento. Una consulta SQL consiste de una serie de instrucciones SQL que indican al DBMS que debe hacer algo, como puede ser devolver filas de datos al componente J2EE.

El método *Connection.createStatement()* se utiliza para crear un objeto de tipo *Statement* (instrucción). El objeto *Statement* se utiliza para ejecutar una consulta y devolver un objeto *ResultSet* (conjunto d resultados), el cual contiene la respuesta del DBMS, la cual consiste de una o mas filas de información solicitada por el componente J2EE.

Generalmente asignamos la consulta a un objeto de tipo String la cual se pasa al método *executeQuery()* (ejecutar consulta) del objeto *Statement*, como se muestra en el siguiente segmento de código. Una vez que se recibe el *ResultSet* del DBMS, se llama al método *close()* para terminar el uso de la instrucción.

```
Connection conexion;  
Statement instruccion;  
ResultSet conjuntoResultados;  
  
try  
{  
  
    instruccion = conexion.createStatement();  
  
    String consulta = ("select * from empleados");  
    conjuntoResultados = instruccion.executeQuery(consulta);  
    instrucción.close();  
  
}
```

En este segmento de código se muestra como recuperar todas las filas y columnas de la tabla empleados.

2.4.4 Procesar los datos que devuelve el DBMS

Una vez que ya fue procesada la consulta el objeto `java.sql.ResultSet` contiene los resultados recibidos del DBMS, este objeto se compone de métodos que se utilizan para interactuar con los datos que devuelve el DBMS al componente J2EE.

En el siguiente segmento de código se muestra como recuperar los datos del `ResultSet`:

```
ResultSet results;  
String nombre, apellido, printrow;  
boolean records = results.next();  
  
if(!records){  
    System.out.println("No hay datos");  
    Return;  
}  
else{  
    do{  
        nombre = results.getString ("nombre");  
        apellido = results.getString ("apellido");  
        printrow = nombre + " " + apellido;  
        System.out.println(printrow);  
    }while(results.next() );  
}
```

Acceso y manipulación de base de datos con JDBC

Como se puede observar en el anterior segmento de código es un ejemplo abreviado que incluye una rutina común para extraer los datos que devuelve el DBMS. Este fragmento de código solicita los nombres y apellidos de los empleados que se contienen en una tabla. El resultado que devuelve el DBMS ya se ha asignado al objeto `ResultSet` llamado `results`. La primera vez que se llama al método `next()` (siguiente) del objeto `ResultSet`, el puntero `ResultSet` se coloca en la primera fila del `ResultSet` y el método devuelve un valor lógico que si es falso, indica que el `ResultSet` no contiene filas. La instrucción `if` atrapa la condición y muestra un mensaje de "No hay datos".

Si el método `next()` devuelve un valor verdadero significa que el `ResultSet` contiene al menos una fila, lo cual hace que el código entre al ciclo `do while`.

El método `getString()` del objeto `ResultSet` se utiliza para copiar el valor de una columna determinada de una fila actual del `ResultSet` a un objeto `String`. El método `getString` recibe como parámetro el nombre de la columna del `ResultSet` cuyo resultado se desea copiar y devuelve el valor de la columna indicada.

También se podría pasar al método `getString` el número de columna en lugar de su nombre. Sin embargo esto solo se debe hacer si las columnas se han nombrado en forma explícita en la instrucción `select`. De lo contrario no hay forma de estar seguro del orden en que las columnas aparecen en el `ResultSet`, sobre todo porque es posible que la tabla halla sido reorganizada después de su creación y que por lo tanto las columnas estén en un orden distinto.

La primera columna del `ResultSet` contiene el nombre del cliente y la segunda contiene su apellido, ambas se concatenan y se muestran en un mensaje, este proceso continua hasta que el método `next()`, el cual se llama como parámetro condicional de la instrucción `while`, devuelve un valor falso, lo cual significa que el puntero se encuentra al final del `ResultSet`.

2.4.5 Terminar la conexión del DBMS

Para terminar la conexión del DBMS se utiliza el método `close()` del objeto `Connection`, una vez que el componente J2EE ha terminado de utilizar el DBMS. El método `close()` manda una excepción si ocurre algún problema al desconectar del DBMS. Aunque al cerrar la conexión a la base de datos cierra automáticamente al **ResultSet**, es mejor cerrar al **ResultSet** en forma explícita antes de cerrar la conexión.

Conexión.close ();

2.5 Objetos Statement

Una vez abierta la conexión a la base de datos, el componente J2EE crea y envía una consulta, para tener acceso a los datos que contiene. La consulta se escribe con lenguaje SQL .

Para ejecutar la consulta se utilizan 3 tipos de objetos Statements disponibles:

- **Objeto Statement.** Ejecuta una consulta en forma inmediata.
- **PreparedStatement.** Se utiliza para ejecutar una consulta precompilada con anterioridad.
- **CallableStatement.** Se utiliza para ejecutar procedimientos almacenados.

2.5.1 El objeto Statement

El objeto Statement se utiliza para ejecutar una consulta en forma inmediata, es decir, que no necesita compilarse con anterioridad. El objeto Statement contiene el método **executeQuery()**, el cual recibe como parámetros una consulta, este transmite una consulta al DBMS para procesarla.

El método **executeQuery()** devuelve un objeto de tipo ResultSet, el cual contiene filas, columnas y metadatos que representan los datos que solicito la consulta.

El método **execute()** se utiliza si es posible que se devuelvan múltiples resultados, este devuelve true si se ha devuelto alguna fila y false en caso contrario. Un método también utilizado con frecuencia es **executeUpdate()**, el cual se utiliza para ejecutar consultas que contienen instrucciones INSERT, UPDATE Y DELETE, este método devuelve un entero que indica el numero de filas afectadas por la consulta.

```
public class ProcesarConsulta {  
  
String url = "jdbc:mysql://localhost:3306/Productos";  
String usr = "root";  
String pwd = "antar99";  
Statement dataRequest;  
ResultSet result;  
Connection db;  
  
try{  
    // Cargamos el controlador.  
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
    db =(url, usr, pwd);  
    }  
  
catch(ClassNotFoundException error)  
{  
    System.out.println("No se puede cargar el puente JDBC/ODBC " + error);  
    System.exit(1);  
}  
catch(SQLException error)
```

```
{
    System.out.println("No se puede conectar con la base de datos" + error);
    System.exit(2);
}
try{
    String query = "SELECT * FROM Clientes";
    dataRequest = db.createStatement();
    results = DataRequest.executeQuery (query);
    //Aqui va el codigo que interactua con el ResultSet
    dataRequest.close();
}
catch(SQLException error)
{
    System.out.println("Error SQL" + error);
    System.exit(3);
}
db.close()

} //fin de la clase
```

En el código anterior se puede observar que declaramos un objeto de tipo Statement llamado dataRequest y un objeto de tipo ResultSet llamada results, en el segundo bloque try {} se asigna la consulta al objeto query, de tipo String. La consulta solicita al DBMS todas las filas de las tablas clientes de la base de datos llamada Productos.

Se llama al objeto **createStatement()** del objeto Connection para obtener un objeto Statement. Se llama al objeto **executeQuery()** del objeto Statement y se le pasa como parámetro a la consulta, para que devuelva un objeto ResultSet que contiene los datos que devuelve el DBMS. Finalmente se llama al método **close()** para cerrarlo.

El método **close()** cierra todas las instancias del objeto ResultSet que ha devuelto el Statement, el no llamarlo puede hacer que los recursos utilizados por el objeto Statement no puedan ser utilizados por otra aplicación J2EE sino hasta que la rutina de recolección de basura se ejecute automáticamente. Las instrucciones Java para manipular el ResultSet se colocan entre el método **executeQuery()** y el método **close()**.

El método **executeQuery()** lanza una SQLException si ocurre algún error durante el procesamiento de la consulta.

A continuación se muestra como utilizar el método **executeUpdate()** del objeto Statement. Este código es casi idéntico que el que utilizamos para executeQuery sin embargo la consulta actualiza el valor de la base de datos en vez de recuperarlo.

```
public class ProcesarConsulta {

    String url = "jdbc:mysql://localhost:3306/Productos";
    String usr = "root";
```



```

String pwd = "antar99";
Statement dataRequest;
Int rowUpdate;
Connection db;

try{
    // Cargamos el controlador.
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    db =(url, usr, pwd);
}

catch(ClassNotFoundException error)
{
    System.out.println("No se puede cargar el puente JDBC/ODBC " + error);
    System.exit(1);
}

catch(SQLException error)
{
    System.out.println("No se puede conectar con la base de datos" + error);
    System.exit(2);
}

try{
    String query = "UPDATE Clientes SET PAID = 'Y' WHERE BALANCE = '0' ";
    dataRequest = db.createStatement();
    rowUpdate = DataRequest.executeUpdate (query);
    dataRequest.close();
}
catch(SQLException error)
{
    System.out.println("Error SQL" + error);
    System.exit(3);
}

db.close()

} //fin de la clase

```

en el código anterior el objeto ResultSet se ha cambiado por la declaración de una variable int llamada rowUpdate, también la consulta es distinta, la instrucción SQL UPDATE indica al DBMS que actualice la tabla cliente de la base de datos Producto. El valor de la columna PAID se cambia por <<Y>> si el valor de la columna BALANCE es igual con cero.

Finalmente se utiliza el método **executeUpdate()**, en vez del método executeQuery y se le pasa la consulta. El DBMS devuelve al método **executeUpdate()** el numero de filas actualizadas, el cual se asigna a las variables rowUpdate. El componente J2EE puede utilizar este numero para diferentes fines, por ejemplo enviar una notificación de confirmación a la aplicación J2EE que solicita el acceso a la base de datos.

2.5.2 El objeto PreparedStatement

Antes de procesar una consulta SQL, el DBMS tiene que compilarla. La compilación ocurre al llamar a cualquiera de los métodos de ejecución del objeto Statement. La compilación de una consulta implica un gasto aceptable si dicha consulta se llama solo una vez.

Es posible de compilar una consulta SQL antes de ejecutarla mediante el uso del objeto PreparedStatement. La consulta es similar a las que se han estado mostrando, sin embargo se utiliza un signo de interrogación (?) como marcador del lugar donde se agregaron los valores una vez echa la consulta. Este valor puede variar cada vez que se ejecuta la consulta.

El uso de las PreparedStatement se recomienda cuando una consulta se va a ejecutar repetidamente, por ejemplo cuando se insertan varios registros en una base de datos y esto es de gran utilidad ya que podemos evitar que se inserten valores incorrectos a la base de datos.

La ventaja que tiene el uso de PreparedStatement es que mejora la seguridad en la base de datos haciendo a los sistemas inmunes a los ataques de inyección de SQL.

A continuación mostramos como utilizarlo, este es casi similar que los ejemplos anteriores.

```
public class ProcesarConsulta {  
  
String url = "jdbc:mysql://localhost:3306/Productos";  
String usr = "root";  
String pwd = "antar99";  
ResultSet result;  
Connection db;  
  
try{  
    // Cargamos el controlador.  
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
db =(url, usr, pwd);  
    }  
  
catch(ClassNotFoundException error)  
{  
    System.out.println("No se puede cargar el puente JDBC/ODBC " + error);  
    System.exit(1);  
}  
  
catch(SQLException error)  
{  
    System.out.println("No se puede conectar con la base de datos" + error);  
    System.exit(2);  
}  
  
}
```

```
try{
String query = "SELECT * FROM Clientes WHERE NumCliente = ? ";
PreparedStatement pstatement = db.prepareStatement(query);
pstatement.setString(1, "123");
results = pstatement.executeQuery ();
//Aquí va el código que interactúa con el ResultSet
pstatement.close();
}
catch(SQLException error)
{
    System.out.println("Error SQL" + error);
    System.exit(3);
}
db.close()

} //fin de la clase
```

En el código anterior se puede observar que en la consulta se puso un signo de interrogación (?) como valor de número de cliente, este es una marca que indica donde irá el número de cliente que será agregado más adelante en el código a la consulta ya compilada.

Para crear el objeto PreparedStatement se llama al objeto preparedStatement del objeto Connection. La consulta se pasa al método **prepareStatement()** y se compila en ese momento.

El método **setString()** se utiliza para reemplazar el signo de interrogación con el valor del método **setString()** recibe como parámetro, este recibe 2 parámetros. El primero es un entero que indica la posición del signo de interrogación, mientras que el segundo parámetro es el valor que reemplazará el signo de interrogación.

Es importante señalar que el objeto PreparedStatement tiene varios métodos setxxx(), cada uno de los cuales especifica el valor que recibe como parámetro.

Otra de las ventajas de utilizar el objeto PreparedStatement es que la consulta se compila una sola vez y se llama a los métodos setxxx() según va siendo necesario para modificar valores específicos de la consulta sin tener que volver a compilarla.

2.5.3 El objeto CallableStatement

El objeto CallableStatement se utiliza para llamar a un procesamiento almacenado desde un componente J2EE. Un procedimiento almacenado es un bloque de código el cual se identifica con un nombre único. El tipo y estilo de código depende del proveedor del DBMS y puede estar escrito en PL/SQL, Transact-SQL, C u otro lenguaje de programación. Para ejecutar el procesamiento almacenado se debe invocar su nombre.

El objeto CallableStatement utiliza tres tipos de parámetros al llamar los procedimientos almacenados: IN, OUT e INOUT.

Acceso y manipulación de base de datos con JDBC

Un parámetro de tipo IN contiene datos que se deben pasar al procedimiento almacenado y cuyo valor se asigna mediante los métodos `setxxx()` en forma idéntica al objeto `PreparedStatement` del que se habla en la sección anterior.

Un parámetro de tipo OUT contiene un valor que devuelve al procedimiento almacenado, si es que devuelve alguno. Es necesario registrar los parámetros OUT mediante el método `registerOutParameter()` antes de llamar al método `execute()`. Para recuperar los valores de los parámetros OUT se utilizan los métodos `getxxx()`.

Un parámetro de tipo INOUT es un parámetro que se utiliza tanto en pasar información al procedimiento almacenado como para recuperar información del mismo, para lo cual se utilizan las técnicas IN y OUT ya mencionadas.

```
public class ProcesarConsulta {

    String url = "jdbc:mysql://localhost:3306/Productos";
    String usr = "root";
    String pwd = "antar99";
    String numUltimoPedido;
    Connection db;

    try{
        // Cargamos el controlador.
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        db =(url, usr, pwd);
        }

    catch(ClassNotFoundException error)
    {
        System.out.println("No se puede cargar el puente JDBC/ODBC " + error);
        System.exit(1);
    }
    catch(SQLException error)
    {
        System.out.println("No se puede conectar con la base de datos" + error);
        System.exit(2);
    }

    try{
        String query = "{CALL NumUltimoPedido (?)} ";
        CallableStatement cstatement = db.prepareCall(query);
        cstatement.registerOutParameter(1, Types.VARCHAR);
        cstatement.execute();
        numUltimoPedido = cstatement.getString(1);
        cstatement.close();
    }
    catch(SQLException error)
    {
```

```
        System.out.println("Error SQL" + error);
        System.exit(3);
    }
    db.close()
} //fin de la clase
```

2.6 Metadatos

Los metadatos son datos sobre los datos. Un componente J2EE puede obtener acceso a los metadatos de la base de datos a través de la interfaz DatabaseMetaData. La interfaz DatabaseMetaData se utiliza para obtener información sobre las bases de datos, tablas columnas e índices, además de otra información sobre el DBMS.

Para obtener los metadatos de la base de datos un componente J2EE llama al método `getMetaData()` del objeto `Connection`. El método `getMetaData` devuelve un objeto `DatabaseMetaData`, el cual contiene información sobre la base de datos y sus componentes.

Una vez obtenido el objeto `DatabaseMetaData` se pueden llamar los distintos métodos que contiene para recuperar metadatos específicos. A continuación se mencionan algunos de los métodos mas utilizados del objeto `DatabaseMetadatas`:

- **`getDatabaseProductName()`**. Devuelve el nombre del producto DBMS.
- **`getUserName()`**. Devuelve el identificador del usuario.
- **`getURL()`**. Devuelve el URL de la base de datos.
- **`getSchemas()`**. Devuelve los nombres de todos los esquemas disponibles en la base de datos.
- **`getPrimaryKeys()`**. Devuelve las claves primarias.
- **`getProcedures()`**. Devuelve los nombres de los procedimientos almacenados.
- **`getTables()`**. Devuelve el nombre de las tablas que contiene la base de datos.

Metadatos ResultSet

Se pueden extraer dos tipos de metadatos del DBMS: los metadatos que describen la base de datos y metadatos que describen un `ResultSet`. Los metadatos `ResultSet` se obtienen mediante el método `getMetaData()` del objeto `ResultSet`. Este devuelve un objeto de tipo `ResultSetMetaData`, como se muestra a continuación.

```
ResultSetMetaData rm = Result.getMetadatas ();
```

Una vez recuperados los métodos del `ResultSet`, el componente J2EE puede llamar los métodos del objeto `ResultSetMetaData` para obtener información sobre los datos que contiene el `ResultSet`. Algunos de los métodos mas comunes son los siguientes:

- **`getCololumnCount()`**. Devuelve el número de columnas que contiene el `ResultSet`.
- **`getColumnName(int numero)`**. Devuelve el nombre de la columna que indica el numero de columna.

Acceso y manipulación de base de datos con JDBC

- ***getColumnType(int numero)***. Devuelve los tipos de dato de la columna que indica el numero de columna. Los valores de los tipos están definidos como constantes de la clase `java.sql.Types`.

SERVLETS Y JAVA SERVER PAGE (JSP)

3.1 Servlets

3.1.1 ¿Qué son los Servlets?

En el primer capítulo se menciona acerca de los CGI's, bueno pues los Servlets son la respuesta de la tecnología Java a la programación CGI. Son programas que se ejecutan en un servidor web y construyen paginas web, es decir, son programas Java que amplían la funcionabilidad de un Servidor web mediante la generación dinámica de páginas web. El motor de Servlets o mejor conocido como contenedor de Servlets es un entorno de ejecución en el cual se administra la carga y descarga del Servlet y trabaja con el Servidor web para dirigir peticiones a los Servlets y enviar las respuestas a los clientes.

Los Servlets no tienen un método main(). Ellos están bajo el control de otras aplicaciones Java llamado contenedor (Tom-Cat es un ejemplo de un contenedor), el contenedor es el que toma la petición y respuesta HTTP del Servlet y a los métodos Servlets (como doPost() o doGet()).

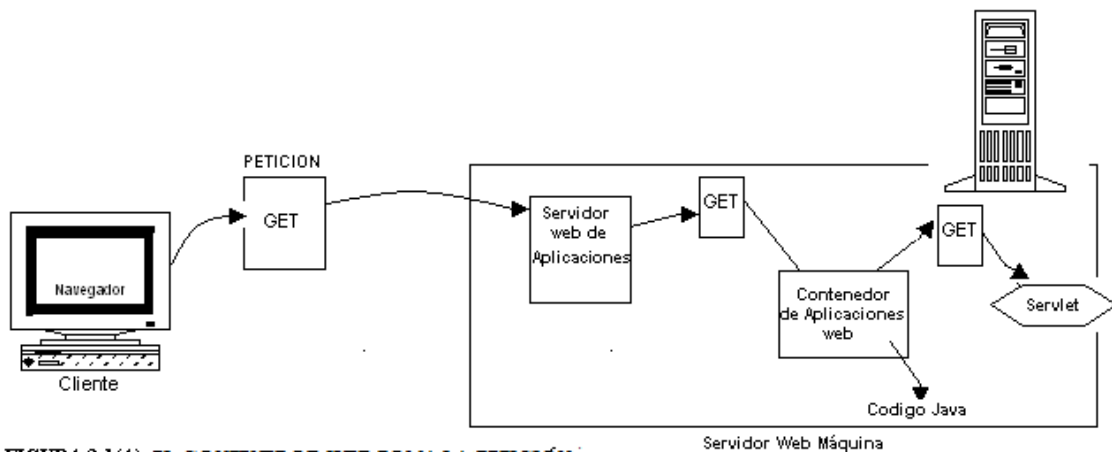


FIGURA 3.1(A) EL CONTENEDOR WEB TOMA LA PETICIÓN

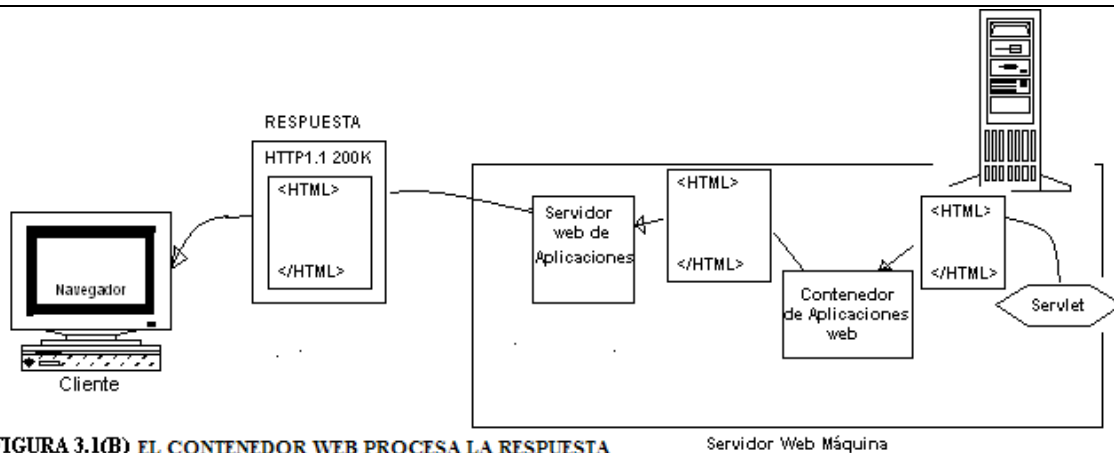


FIGURA 3.1(B) EL CONTENEDOR WEB PROCESA LA RESPUESTA

Cabe señalar que en un Servlet incrustamos código HTML, esto es algo tedioso para los programadores, en respuesta a dar solución a esto salieron las JSP's (hablaremos de esto con mas detalle en el capítulo JSP).

Aunque la tecnología de Servlets reemplaza a los CGI, ambos proporcionan la misma funcionalidad básica, es decir, que un cliente envía datos explícitos e implícitos a un programa servidor en forma de una petición. El programa procesa dicha petición y devuelve otro conjunto de datos explícitos e implícitos.

Los datos explícitos se refieren a la información que se recibe del cliente, que el usuario por lo general introduce en la interfaz de usuario o bien que la interfaz de usuario mismo genera. Por ejemplo, el identificador de usuario y la clave de acceso son datos explícitos que puede enviar un navegador, un applet o una aplicación cliente. Los datos implícitos se componen de información que forman parte del protocolo de transferencia de hipertexto (HTTP) que genera el cliente, es decir, el navegador en vez del usuario. Dicha información incluye datos sobre la petición como pueden ser un esquema de cookies u otro tipo de medios.

Claro ahora la pregunta será ¿Por qué razón utilizar Servlets?, ¿por qué usar paginas dinámicas?, ¿qué ventajas obtengo?.

RAZONES

- Las páginas web están basadas en datos enviados por el usuario.
- Los datos cambian frecuentemente.
- Las paginas web que usan información desde base de datos corporativas u otras fuentes.

3.1.2 Ventajas de usar Servlets:

Los Servlets Java son eficientes, fáciles de usar, poderosos, portables y más baratos que otras tecnologías.

EFICIENTES

Con los Servlets la JVM permanece arrancada y cada petición es manejada por un hilo Java de peso ligero no un pesado proceso del sistema operativo (cada petición corre por un hilo por separado).

PORTABLE

Los Servlets están escritos en Java y siguen una API bien especificada. Los Servlets están soportados directamente mediante un plug-in en la mayoría de los servidores web, es decir, un Servlet maneja varias peticiones creando un hilo por cada petición.

BARATO

Una vez que tengamos un servidor web, no importa el coste del Servidor, añadirle soporte Servlet (si no viene soporte preconfigurado para soportarlos) es gratuito y muy barato. Además de que no se requiere licencia para desarrollar en Servlets.

Cada vez que se realiza una petición de un programa CGI se inicia un nuevo proceso lo cual toma mucho tiempo pero es aceptable siempre que el tiempo de ejecución del programa CGI sea mayor que el tiempo que tarda en iniciar un nuevo proceso, por ejemplo si se realizan 100 peticiones simultaneas que debe atender un programa CGI, será necesario cargar el programa CGI en memoria 100 veces, es decir, que habrá 100 copias del programa en memoria. Cada instancia implica iniciar un nuevo proceso, lo cual indica que se iniciaran 100 nuevos procesos, lo cual es insuficiente, además un programa CGI no es persistente. Una vez que termina se pierden todos los datos que utilizo el proceso y otros programas CGI no pueden utilizarlo.

La tecnología Servlet evita las insuficiencias. En primer lugar solo se carga una copia de un Servlet en la JVM sin importar el número de peticiones que tenga que atender. Con cada petición se inicia un hilo en vez de un nuevo proceso, lo cual reduce el uso de memoria en el Servidor y reduce el tiempo de respuesta. Cada copia de un Servlet se ejecuta en un solo hilo. Un Servlet tiene persistencia, lo cual quiere decir que el Servlet sigue vivo una vez terminada la petición.

3.1.3 Estructura básica de un Servlet

Los Servlets también pueden manejar peticiones GET y POST muy fácilmente, que son generados cuando alguien crea un formulario HTML que especifica METED = "POST" o "GET". A continuación se muestra un Servlet básico:

```
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();
        //usamos out para enviar contenido al navegador

        out.println("Hola Mundo");
    }
}
```

Observando el código para que sea un Servlet tiene que extender de `HttpServlet` y sobrescribir el método `doGet()` o `doPost()` (o ambos). Claro este depende de los datos que se estén enviando de un formulario HTML mediante GET o POST. Estos métodos toman dos argumentos:

- `HttpServletRequest`
- `HttpServletResponse`

El objeto `HttpServletRequest` el cual contiene la información que proviene del cliente, es decir, tiene métodos que nos permiten encontrar información entrante como los datos de un FORMULARIO, cabeceras de petición http, etc.

El objeto `HttpServletResponse` que utiliza el Servlet para enviar información al cliente, es decir, tiene métodos que nos permiten especificar líneas de respuesta (200, 400, etc), cabeceras de respuesta (Content-Type, set-cookies, etc) y también nos permite obtener un `PrintWriter` usado para enviar la salida al cliente (Para Servlets sencillos la mayoría de los esfuerzos se gastan en sentencias `println` que generan la pagina deseada). Ambos métodos (`HttpServletRequest` y `HttpServletResponse`) lanzan un `ServletException` y un `IOException`.

También podemos observar que tenemos que importar las clases de los paquetes `java.io` (para `PrintWriter`, etc), `javax.servlet`(para `HttpServlet`, etc) y `javax.servlet.http` (para `HttpServletRequest` y `HttpServletResponse`).

3.1.4 Pasos para crear un Servlet

Los pasos necesarios para crear un Servlet son:

- Crear nuestra estructura de directorios.
- Escribir una clase Java que herede de HttpServlet.
- Configurar el Descriptor de Despliegue(hablaremos del DD mas adelante)..
- Ejecutar nuestro contenedor de Servlets.
- Invocar al Servlet a través del url pattern que declaremos.

Ambiente de despliegue

Desplegar una aplicación web envuelve reglas específicas del Contenedor de servlets y requerimientos de la especificación para JSP y Servlets. A continuación en la figura 3.2 se muestra una estructura de directorios para desplegar una aplicación Web usando Tomcat.

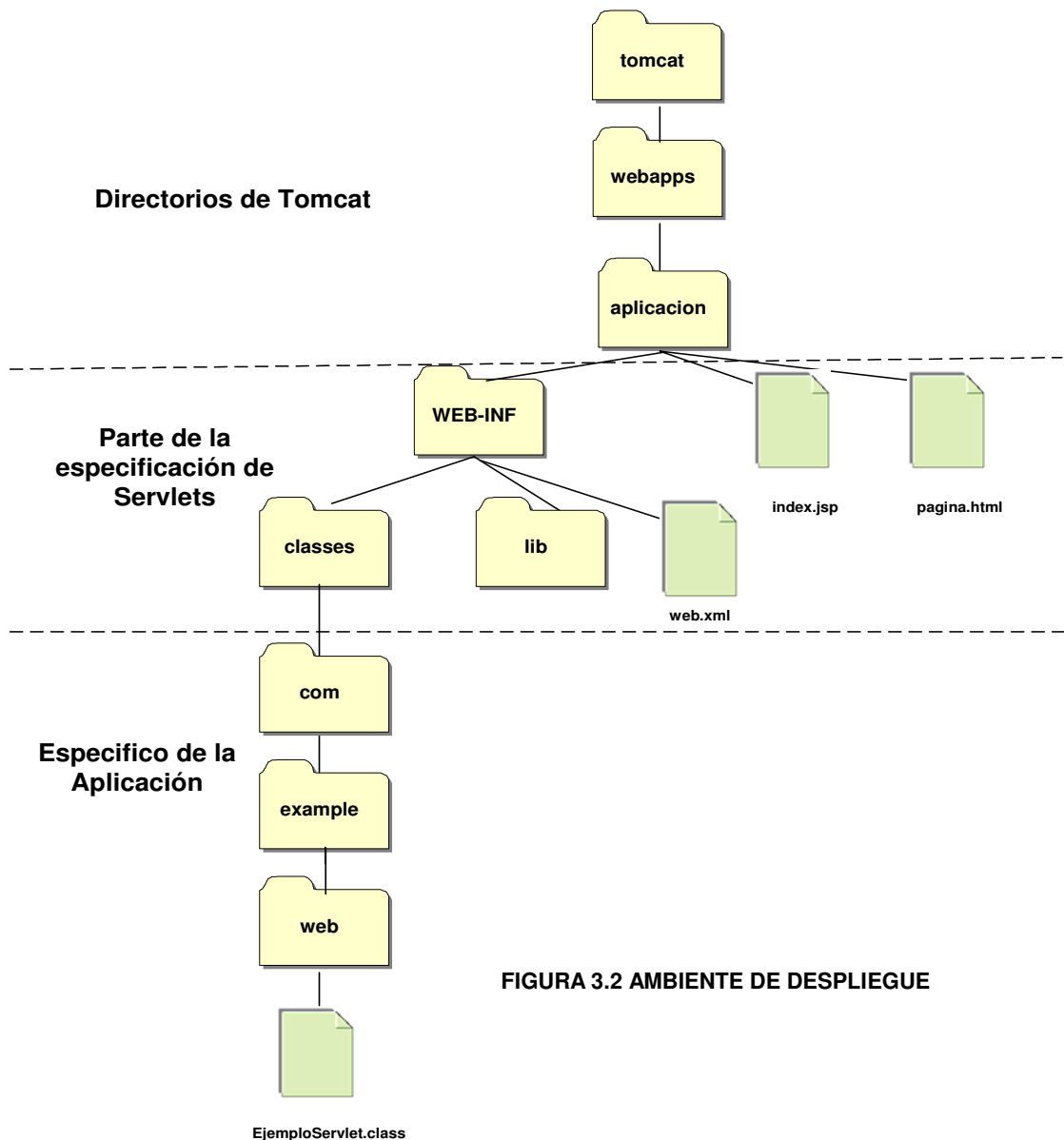


FIGURA 3.2 AMBIENTE DE DESPLIEGUE

Deployment Descriptor (web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
  app_2_4.xsd">

<servlet>
  <servlet-name>Chapter1 Servlet</servlet-name>
  <servlet-class>Ch1Servlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>Chapter1 Servlet</servlet-name>
  <url-pattern>/Serv1</url-pattern>
</servlet-mapping>

</web-app>
```

Servlet

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

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

    out.println("<html><head><title>Ejemplo Servlet</title></head>");
    out.println("<body>");

    String serverName = request.getServerName();

    out.println("<h1>El Nombre del servidor es " + serverName + "</h1>");
    out.println("</body>");
  }
}
```

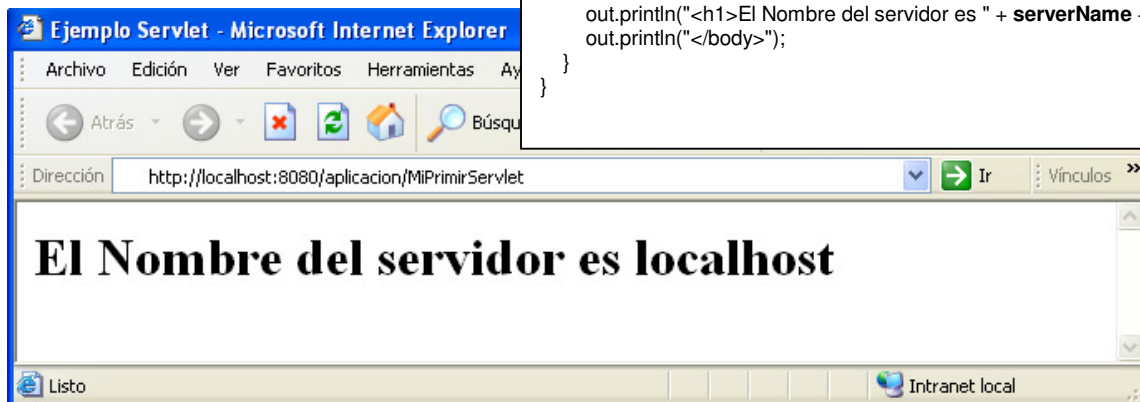


FIGURA 3.3 EJEMPLO DE UN SERVLET

En la figura 3.3 se muestra un ejemplo básico de un servlet,, se puede observar como se declaran los servlets en el descriptor de despliegue y el código del servlet que manda como respuesta el nombre del servidor en el navegador.

COMO EJECUTAR EL SERVLET

Antes que nada para administrar bien hay que construir una estructura de directorios, también llamada ambiente de desarrollo como se muestra en la figura 3.4:

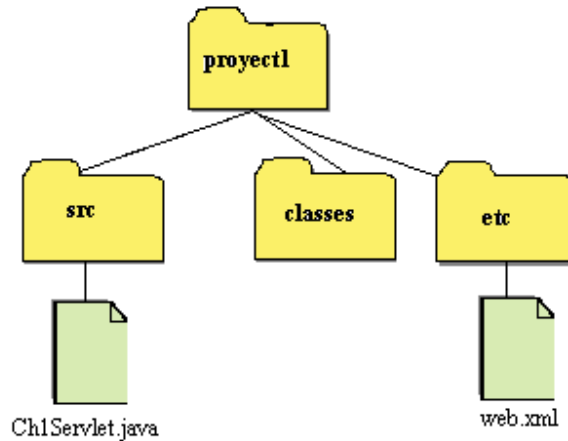


FIGURA 3.4 AMBIENTE DE DESARROLLO

- 1) En el directorio src colocamos nuestro Servlet llamado Ch1Servlet.java.
- 2) Creamos un Descriptor de Despliegue (DD) llamado web.xml y lo colocamos en el directorio etc.
- 3) Construir los directorios que se muestran en la figura 3.5 estos estarán bajo el directorio de TomCat (también llamado ambiente de despliegue). (Para todos los ejemplos utilizaremos el tomcat, aunque hay otros servidores que se pueden usar pero este cumple con todo lo que necesitamos).

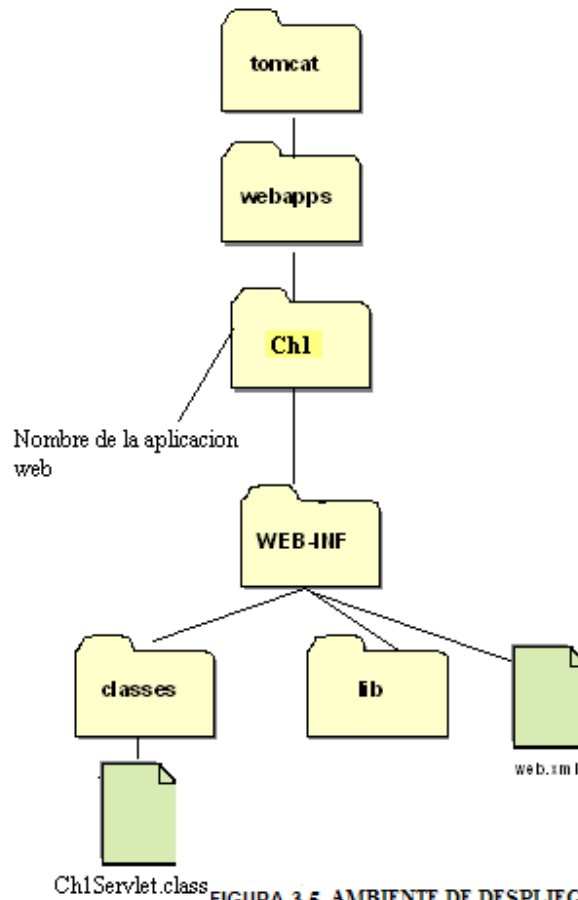


FIGURA 3.5 AMBIENTE DE DESPLIEGUE

4) Del directorio project1 compila el Servlet.

Para Unix → % javac -classpath /tu path/tomcat/common/lib/Servlet-api.jar -d classes src/Ch1Servlet.java

Para Windows → C: project1> javac -classpath \tu path\tomcat\common\lib\Servlet-api.jar -d classes src\Ch1Servlet.java

5) Copia el Ch1Servlet.class en WEB-INF/classes y copia el web.xml a WEB-INF.

6) Del directorio de tomcat, inicia tomcat bin/startup.sh

7) Abre el navegador y teclea en la barra de direcciones la siguiente url:

<http://localhost:8080/ch1/Serv1> → el webapp es llamado ch1 y el Servlet es llamado Serv1.

3.1.5 Descriptor de despliegue (DD)

Un Servlet puede tener 3 nombres

- La ruta del nombre del archivo, es decir, como `classes/Nombre de mi Servlet.class` (una ruta del actual archivo `.class`).
- Un nombre secreto interno (Secret internal name). Este puede tener el mismo nombre de la clase o nombre del archivo, la ruta relativa pero también puedes poner algún otro nombre completamente diferente.
- El nombre publico URL. El nombre que el cliente conoce.

Pero claro la pregunta es porque nosotros no usamos solo el nombre real para no confundir el nombre del archivo.

Mapear el nombre de tus Servlets provee a tu aplicación flexibilidad y seguridad, es decir, cuando necesitamos organizar nuestra aplicación y posiblemente mover algunas cosas dentro de diferentes estructuras de directorios una solución es mapeando el nombre en lugar de modificar el código en el archivo real y el nombre de la ruta. Entonces tenemos la flexibilidad de mover cosas sin tener la pesadilla del mantenimiento cambiando el código que se refiere al archivo Servlet.

Además un punto importante es la seguridad, nosotros no quisiéramos que un cliente sepa exactamente como están las cosas en la estructura de nuestro Servidor, es decir, que el cliente intente navegar en el directorio de nuestros Servlets no es seguro es mejor tener nuestras páginas fuera porque si el usuario final puede ver la ruta real, estos pueden curiosear dentro de su navegador y tratar de acceder a nuestro directorio.

USANDO NUESTRO DESCRIPTOR DE DESPLIEGUE (DD) PARA MAPEAR URL'S A SERVLETS

Cuando tu despliegas un Servlet dentro del contenedor web tu creas un simple documento XML llamado descriptor de despliegue DD (ver Figura 3.6) que dice al contenedor como correr tus Servlets y JSP's. Aunque se usa el DD para mas que solo mapear nombres.

Usaremos 2 elementos XML para mapear URL's a Servlets –Uno para mapear el nombre publico que conoce el cliente y otro para mapear tu propio nombre interno.

Los dos elementos DD para mapear URL:

1)<servlet> → mapea el nombre interno, es decir, el totalmente calificado nombre de la clase.

2)<servlet-mapping> → mapea el nombre interno de el nombre publico de tu URL.

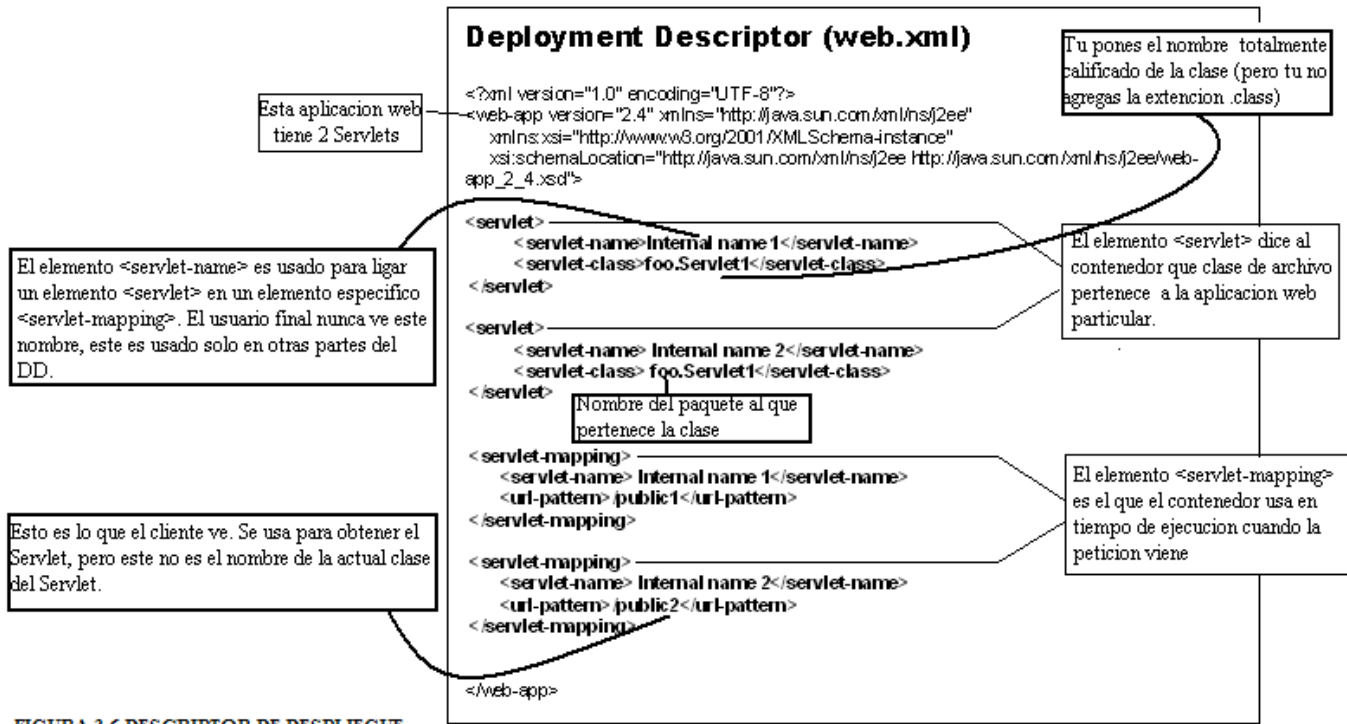


FIGURA 3.6 DESCRIPTOR DE DESPLIEGUE.

Pero esto no es todo lo que puedes hacer con tu DD, además de mapear URL's del Servlet actual, tu puedes usar tu descriptor de despliegue para otros aspectos de tu aplicación web incluyendo roles de seguridad, páginas de error, tags libraries, información de configuración inicial y se puede declarar que se estén accediendo únicamente Enterprise Java Beans.

El DD provee un mecanismo de declarativas para tu aplicación web sin tener que tocar el código. Mas adelante se describirá como podemos declarar parámetros de inicio en nuestro DD, lo cual nos permite liberarnos de modificar código.

3.1.6 Como el contenedor toma una petición

El los roles generales del contenedor en la vida de los Servlets, este crea los objetos petición y respuesta, crea un nuevo hilo para el Servlets y llama al método service() de los Servlets. Pasando las referencias petición y respuesta como argumentos. A continuación se ejemplifica lo mencionado en la figura 3.7(A) y 3.7(B):

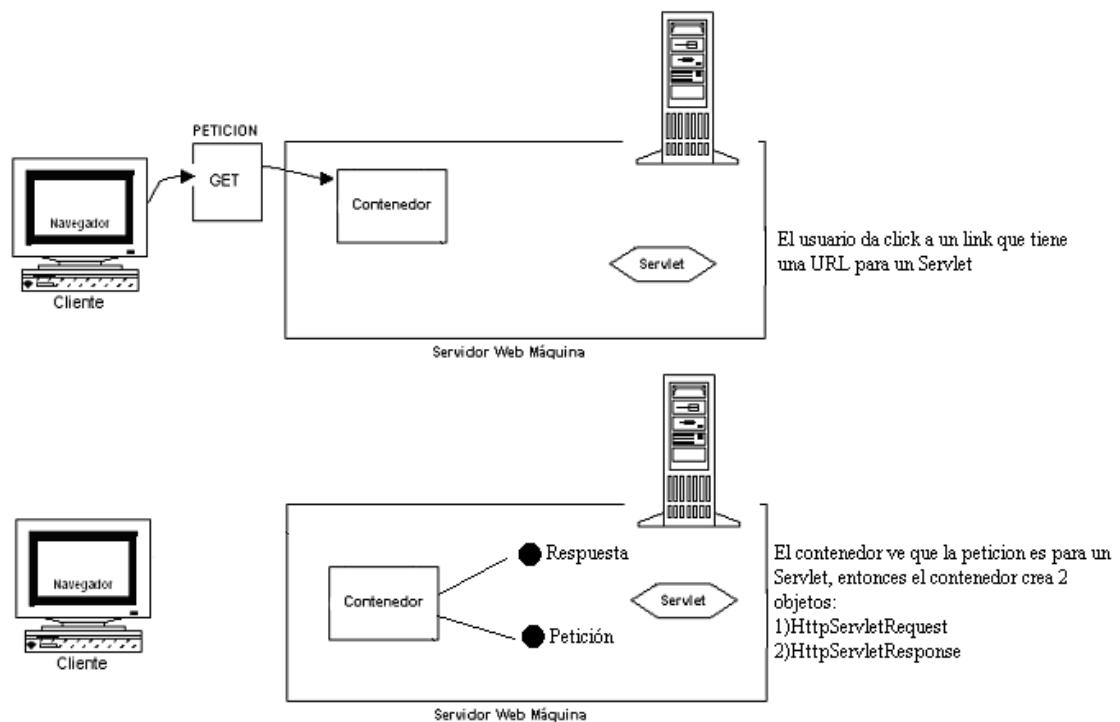


FIGURA 3.7(A) COMO EL CONTENEDOR TOMA UNA PETICIÓN / RESPUESTA PARTE A

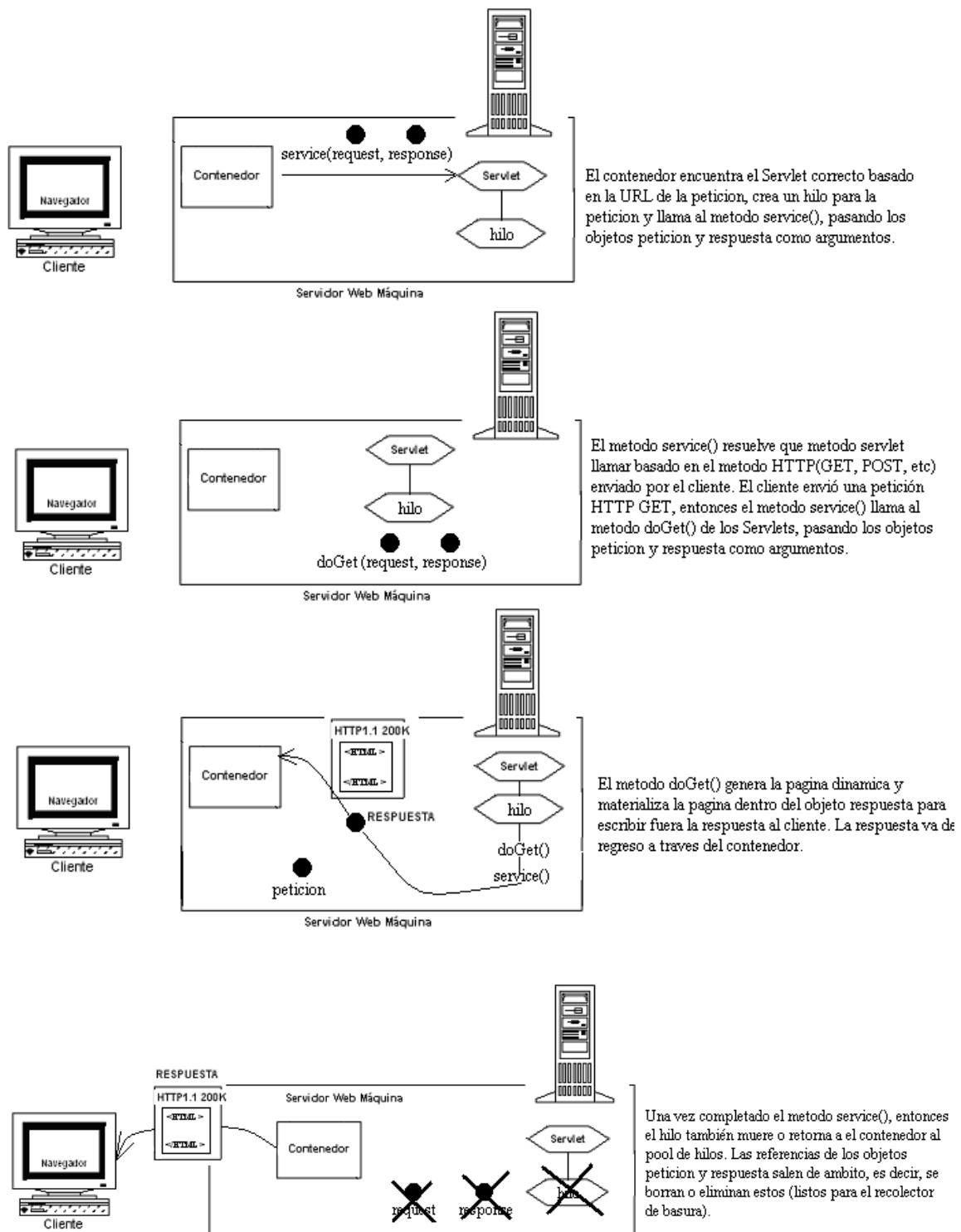


FIGURA 3.7(B) COMO EL CONTENEDOR TOMA UNA PETICIÓN / RESPUESTA PARTE B

3.1.7 Ciclo de vida de los Servlets

Algunas preguntas saldrán a relucir como:

- ¿Cuándo es cargada la clase Servlet?
- ¿Cuándo el Servlet hace que corra su constructor?
- ¿Cuánto tiempo vive el objeto Servlet?
- ¿Cuándo tu podrás inicializar los recursos del Servlet?
- ¿Cuándo deberás limpiar estos recursos?

El ciclo de vida de los Servlets es simple: Hay solo un estado principal - inicializado(initialized).

Si el Servlet no es inicializado, entonces este será inicializado (corriendo su constructor o método `init()`), será destruido (corriendo su método `destroy()`), o simplemente este no existe, la figura 3.8 ilustra lo mencionado.

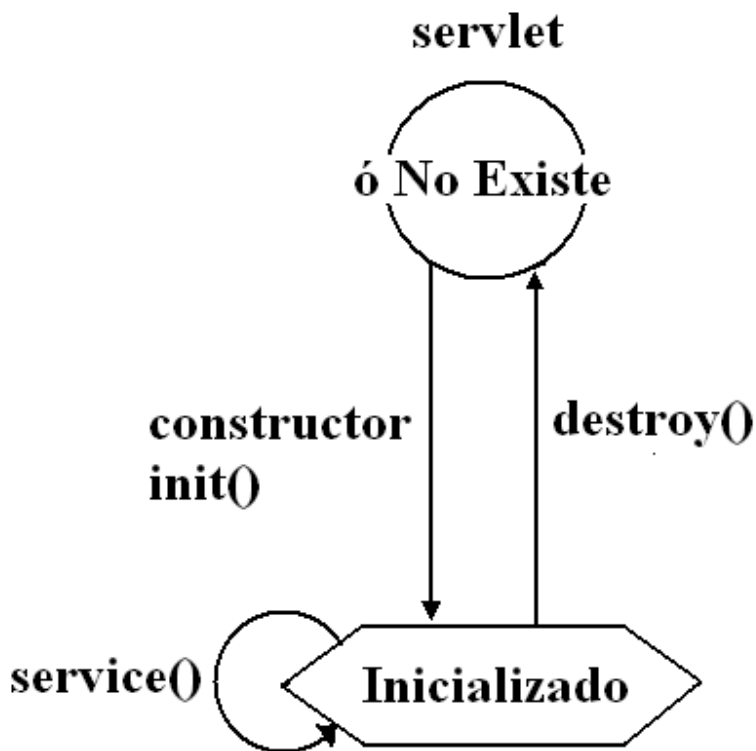


FIGURA 3.8 DIAGRAMA CICLO DE VIDA DE UN SERVLET

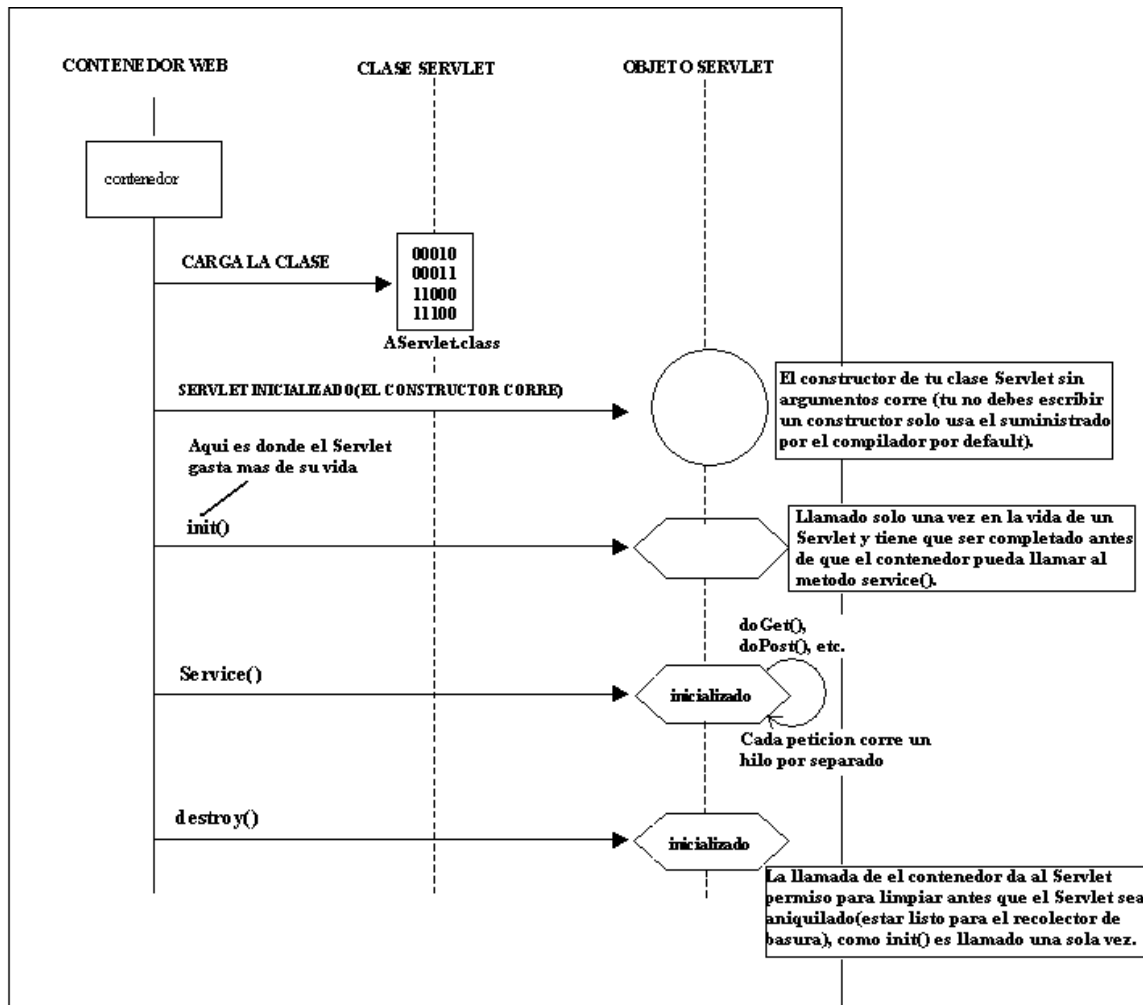


FIGURA 3.9 DIAGRAMA DE SECUENCIA CICLO DE VIDA DE UN SERVLET

Como observamos en la figura 3.9 un Servlet tiene por lo menos 4 métodos que se llaman durante su ciclo de vida. Estos son el método `init()`, `service()`, el método de servicio y `destroy()`.

El método `init()` se llama de forma automática al crear el Servlet. Se puede sobre escribir el método `init()` para incluir instrucciones que se ejecutan una sola vez durante la vida de un Servlet y no en cada petición que recibe, también se utiliza para inicializar variables y objetos que se utilizan en todo el Servlet. El método `init()` no recibe parámetros, devuelve `void` y lanza una excepción `ServletException`. En el mundo real el método `init()` se utiliza para leer parámetros de inicialización, como pueden ser parámetros de la base de datos o rendimiento.

El método `service()` se llama cada vez que el servidor web recibe una petición para el Servlet. La petición hace que se inicialice un nuevo hilo ó que se utilice un hilo libre del pool de hilos. El método `service()` puede recibir varias llamadas simultaneas. El método

service() examina el tipo de petición http y llama al método adecuado, por ejemplo doGet() ó doPost().

El resultado de las llamadas simultaneas al método service() es que hay varios hilos concurrentes en los métodos doGet() , doPost u otros métodos de petición. Esto significa que es necesario sincronizar los accesos de los datos compartidos para evitar problemas de concurrencia.

El método destroy() es el último método que se llama justo antes de destruir el Servlet, por ejemplo al eliminar de memoria la instancia del Servlet se puede sobrescribir el método destroy() de forma que libere recursos como por ejemplo cerrar la conexión a una base de datos.

NOTA: El método destroy() no se llama cuando el servidor termina en forma abrupta, por ejemplo un fallo de sistema.

3.1.8 Configuración de parámetros

Imagina un escenario en el que quieres que tu aplicación muestre tu Email en tu página pero este cambia constantemente y no queremos estar recompilando nuestro Servlet para cada modificación, esto se puede hacer con ayuda de nuestro DD sin necesidad de tocar el código.

Entonces en ciertas ocasiones es necesario configurar ciertos datos que necesita un Servlet por ejemplo rutas de acceso a base de datos, passwords, etc. Para este propósito se utiliza la etiqueta **<init-param>**.

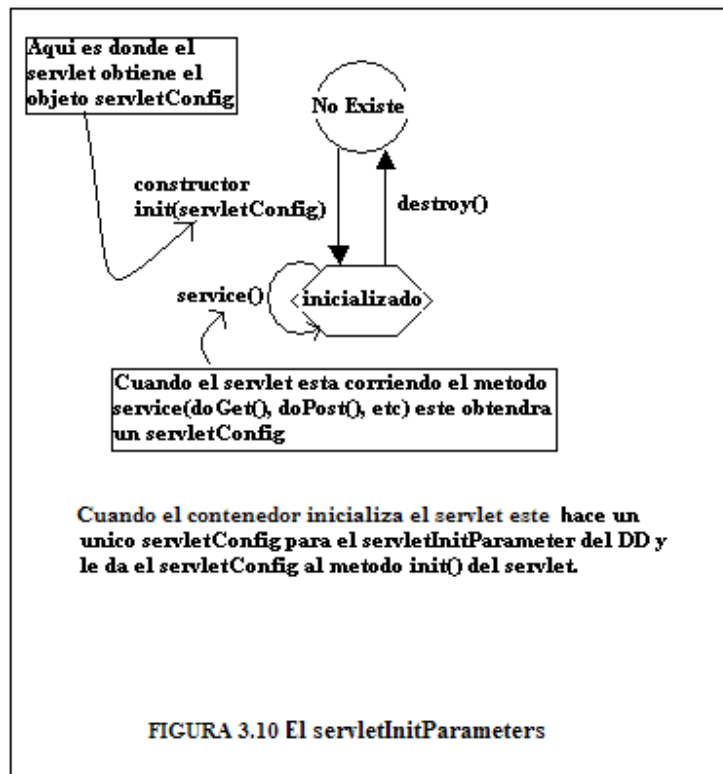
En el DD (web.xml) se pone:

```
<servlet>
  <servlet-name>Uso de parametros</servlet-name>
  <servlet-class> EjemploParametros</servlet-class>
  <init-param>
    <param-name>email</param-name>
    <param-value>aplicacion@hotmail.com</param-value>
  </init-param>
</servlet>
```

En el servlet se recupera el parámetro de la siguiente manera:

```
out.println( getServletConfig( ).getInitParameter("email") );
```

Los parámetros del servlet solo pueden ser recuperados por el servlet que esta siendo mapeado dentro de la etiqueta **<servlet>**.



El servletInitParameters es leído solo una vez cuando el contenedor inicializa el servlet, es decir, cuando el contenedor hace un Servlet este lee el DD y crea los pares nombre / valor para el servletConfig como se ilustra en la figura 3.10. El contenedor nunca lee el initParameter otra vez. Una vez que los parámetros están en el servletConfig, ellos no serán leídos de nuevo hasta que ó al menos que tu redespliegues el Servlet.

Bueno pero nosotros podríamos necesitar esto para toda nuestra aplicación , en lugar de usar esto solo para un servlet.

Configuración de parámetros globales

Cuando requerimos que ciertos datos sean visibles para toda la aplicación web, declaramos parámetros globales. Context Init Parameters trabaja solo como un servletInitParameter, excepto que contextParameters son accesibles para la aplicación web entera.

En el DD (web.xml) se pone:

```
<servlet>
  <servlet-name>Uso de parametros</servlet-name>
  <servlet-class> EjemploParametros</servlet-class>
  <init-param>
    <param-name>email</param-name>
    <param-value>aplicacion@hotmail.com</param-value>
  </init-param>
</servlet>
```

```
<context-param>
  <param-name>global</param-name>
  <param-value>Parametro global</param-value>
</context-param>
```

En el servlet se recupera el parámetro de la siguiente manera:

```
out.println( getServletContext().getInitParameter("global"));
```

otra forma seria:

```
ServletContext context = getServletContext();
Out.println(context.getInitParametr("global"));
```

IMPORTANTE: El <cotext-param> esta para la aplicación entera, entonces este no necesita estar dentro de un elemento <servlet> individual.

DIFERENCIAS ENTRE INITPARAMETER Y CONTEXTINITPARAMETER

- InitParameter esta dentro de un elemento <servlet> por cada servlet especifico y contextParameter se encuentra dentro de el elemento <web-app> pero no dentro del elemento Servlet especifico.

Context init parameters

Deployment Descriptor

```
<web-app ...>
  <context-param>
    <param-name>foo</param-name>
    <param-value>bar</param-value>
  </context-param>
</web-app>
```

Servlet Code

```
getServletContext().getInitParameter("foo");
```

Servlet init parameters

Deployment Descriptor

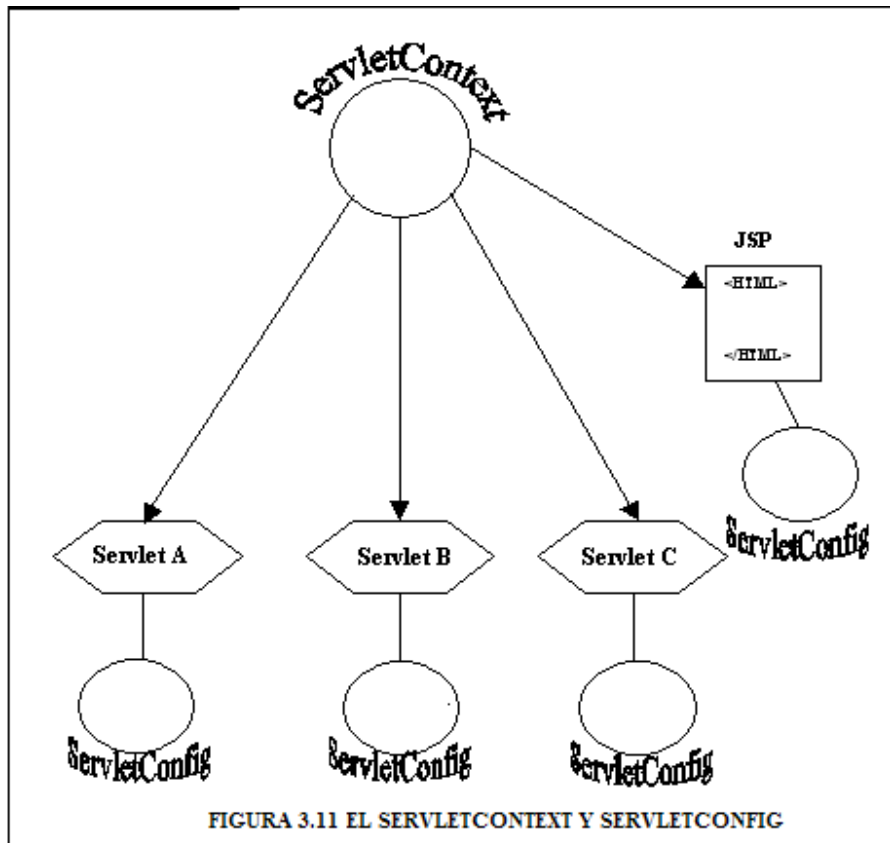
```
<web-app ...>
  <servlet>
    <servlet-name>BeerParamTest</servlet-name>
    <servlet-class>com.TestInitParams</servlet-class>
    <init-param>
      <param-name>foo</param-name>
      <param-value>bar</param-name>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>MyServlet.do</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlet Code

```
getServletConfig().getInitParameter("foo");
```

Servlets y Java Server Pages(JSP)

Hay solo un servletContext para una aplicación web entera, y todas las partes de la aplicación web comparten esta, pero cada servlet en la aplicación tiene su propio servletConfig. El contenedor hace un servletContext cuando una aplicación web es desplegada y hace el context accesible para todos los Servlets y JSP en la aplicación web como se puede observar en la figura 3.11.



ServletContext y ServletConfig

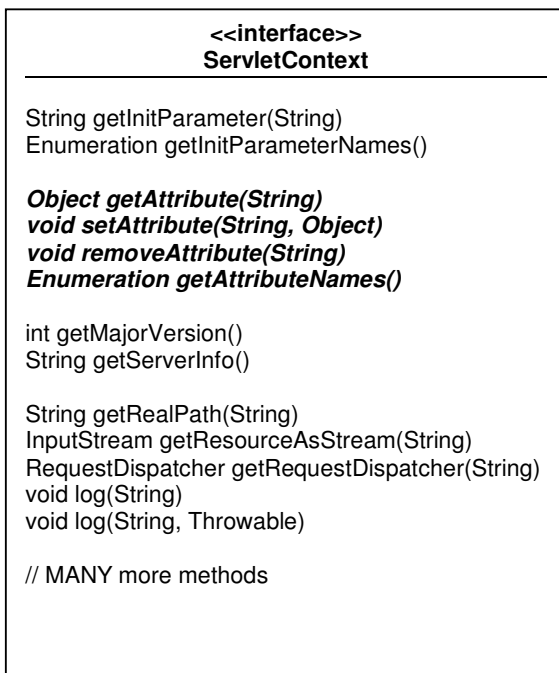
<<interface>> javax.servlet.ServletContext
Object getAttribute(String name) Enumeration getAttributeNames() ServletContext getContext(String uripath) String getInitParameter(String name) Enumeration getInitParameterNames() int getMajorVersion() String getMimeType(String file) int getMinorVersion() RequestDispatcher getNameDispatcher(String name) String getRealPath(String path) RequestDispatcher getRequestDispatcher(String path) URL getResource(String path) InputStream getResourceAsStream(String path) Set getResourcePaths(String path) String getServerInfo() String getServletContextName() void log(String message) void log(String message, Throwable throwable) void removeAttribute(String name) void setAttribute(String name, Object object)

<<interface>> javax.servlet.ServletConfig
String getInitParameter(String name) Enumeration getInitParameterNames() ServletContext getServletContext() String getServletName()

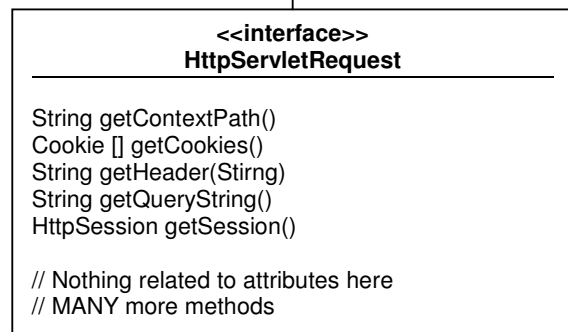
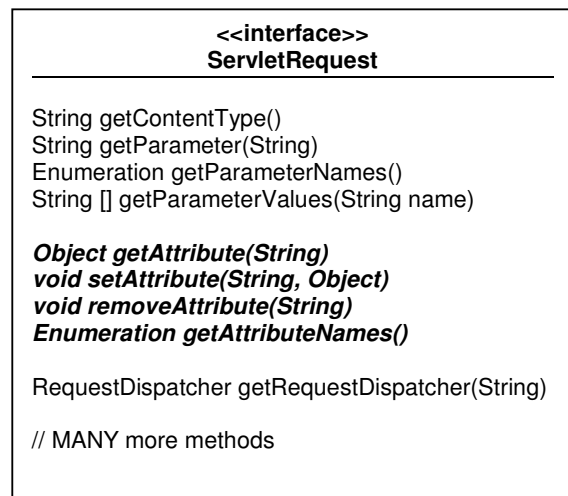
3.1.9 Atributos

Un atributo es un objeto establecido dentro de uno de los siguientes Objetos de la API – ServletContext, HttpServletRequest (o ServletRequest), HttpSession. Se puede pensar en un atributo como un simple par nombre/valor, donde el nombre es un String y el valor es un Object.

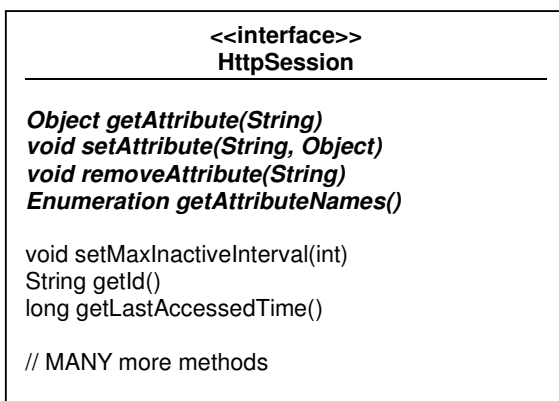
Context



Request



Session



Métodos relacionados con los atributos

Object `getAttribute(String name)`

Con este método recuperamos el atributo que guardamos, para lo cual le pasamos el nombre del atributo y nos regresa un Object si lo encuentra, o null si no encontró el atributo.

void `setAttribute(String name, Object value)`

Este método nos sirve para guardar un atributo, para lo cual debemos indicar el nombre del atributo, y pasarle el objeto que vamos a guardar.

void `removeAttribute(String name)`

Este método se utiliza cuando queremos remover un atributo.

Enumeration `getAttributeNames()`

Con este método obtenemos todos los nombre que están guardados en un determinado objeto de la API, ya sea **ServletContext**, **HttpServletRequest** ó **HttpSession**.

Ejemplo:

Servlet (com.example.web.EjemploAtributos)

```
package com.example.web;

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

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

        out.println("<html><head><title>Ejemplo Servlet</title></head>");
        out.println("<body>");

        // Guardamos un atributo
        request.setAttribute("nombre", "Michel");

        // Recuperamos el atributo
        out.println("<h1>El Nombre es " + request.getAttribute("Michel") + "</h1>");
        out.println("</body>");
    }
}
```

3.1.10 Administración de Sesiones

Inicialmente el Protocolo de transferencia de hipertexto (HTTP) fue diseñado para distribuir documentos e imágenes por la World Wide Web (WWW). Como tal protocolo, usa un modelo de comunicaciones bastante simple. Un cliente hace una petición de un documento, el servidor responde con el documento o con un código de error y la transacción termina. El servidor no conserva nada de la petición. Cuando un cliente hace una nueva petición otra vez, el servidor no tiene forma de distinguirla de la de cualquier otro cliente. Por esta razón se dice que HTTP es un protocolo sin estado. En ocasiones necesitamos que nuestra aplicación web pueda guardar la traza de todas las cosas que el cliente ha dicho durante la conversación, no solo la respuesta en la actual petición.

Seguimiento de sesiones

Como el servidor Web no recuerda los clientes de una petición a otra, la única manera de mantener una sesión es que los clientes guarden una traza de está. Esto se puede conseguir de dos formas básicamente:

- Hacer que el cliente recuerde todos los datos relacionados con la sesión y enviárselos de vuelta al servidor si es necesario.
- Hacer que el servidor conserve todos los datos, asignarles un identificador y hacer que el cliente recuerde el identificador.

La primera técnica es sencilla de implementar y no requiere ninguna capacidad especial por parte del servidor. Sin embargo, esta técnica puede implicar tener que transmitir grandes cantidades de datos de un lado a otro, lo que puede afectar el rendimiento.

La segunda técnica ofrece mayor funcionalidad. Una vez que un servidor haya iniciado una sesión y el cliente la haya aceptado, el servidor puede construir objetos activos complejos y mantener grandes cantidades de datos, siendo sólo necesaria una clave para distinguir entre sesiones.

Así pues, ¿cómo podemos conseguir que el cliente recuerde los datos y los devuelva al servidor Web? Normalmente se utilizan cuatro técnicas:

- Los campos ocultos
- La reescritura de URL
- Las cookies
- La API de sesión HTTP.

Campos ocultos

Los formularios HTML soportan elementos de entrada (input) de tipo HIDDEN. Estos campos ocultos se pasan junto con otros parámetros del formulario en la petición HTTP enviada al servidor Web, pero no se visualizan en la página Web. Sirven sólo para incluir literales o valores constantes en las peticiones. En principio, los campos ocultos se pueden utilizar en páginas Web de HTML ordinario, pero para el seguimiento de sesiones se deben utilizar páginas Web generadas dinámicamente, creadas por procesos del servidor tales como CGI, servlets o JSP.

`<INPUT TYPE="hidden" NAME="campo" VALUE="valor del campo" >`

Los campos ocultos van muy bien con aplicaciones interactivas bidireccionales que no requieren un gran almacenamiento de datos ni la inicialización de un gran número de objetos.

Reescritura de URL

Una dirección URL puede tener parámetros añadidos que se envían junto con la petición al servidor Web. Estos parámetros son pares nombre/valor y tienen la sintaxis siguiente:

`http://servidor/EjemploServlet?name1=value1&name2=value"&...`

Cuando el Servlet recibe la petición, puede leer los valores con

```
String value1 = request.getParameter("name1");
String value2 = request.getParameter("name2");
```

Las páginas Web generadas dinámicamente pueden aprovechar esta utilidad para almacenar datos de sesión en direcciones URL que se escriben a la página como hiperenlaces. Esto permite al cliente poder recordarle al servidor todos los valores necesarios para que éste ponga la aplicación en el estado apropiado.

Un ejemplo sería un contador que indique el número de veces que un usuario ha accedido a una página durante la sesión en curso.

Se garantiza que esta técnica funciona en todos los entornos de navegador y en todas las configuraciones de seguridad, pero ésta es su única ventaja. Esta técnica tiende a reducir el rendimiento si se almacenan grandes cantidades de datos. Las direcciones URL pueden llegar a ser muy extensas y posiblemente exceder el tamaño aceptado por el servidor Web. Además, las direcciones URL no son seguras, ya que son visibles en la ventana de direcciones del navegador y en los registros del servidor Web.

Cookies

La técnica usada con más frecuencia para el almacenamiento persistente de los datos del cliente es el uso de cookies HTTP. Aunque fueron originalmente designadas para soporte de ayuda a las sesiones de estado, estas se pueden usar para otras cosas. Una cookie no es nada más que una pequeña pieza de datos (un par de nombre/valor) intercambiando entre el cliente y el servidor, o mejor dicho, una cookie es un pequeño elemento de datos con nombre que el servidor le pasa a un cliente con una cabecera Set-Cookie como parte de la respuesta HTTP. Se espera que el cliente almacene la cookie y la devuelva al servidor por medio de una cabecera Cookie en las peticiones posteriores que se realicen al mismo servidor.

Por default una cookie vive solo mientras haya una sesión, una vez que el cliente quita su navegador, el cookie desaparece. Así es como el "JSESSIONID" cookie trabaja. Pero nosotros podemos decirle a una cookie que este viva después de que el navegador se cierre.

Servlets y Java Server Pages(JSP)

Junto con el nombre y el valor, la cookie puede contener:

- Una fecha de caducidad, después de la cual no se espera ya que el cliente conserve la cookie. Si no se especifica ninguna fecha, la cookie expira en el momento en que termina la sesión con el navegador.
- Un nombre de dominio, como por ejemplo servername.com, que delimita el subconjunto de direcciones URL para las que la cookie es válida. Si no se especifica, la cookie se devuelve con todas las peticiones al servidor Web que la creó.
- Un nombre de ruta que delimita aún más el subconjunto de direcciones URL.
- Un atributo secure, que si está presente indica que la cookie sólo debe ser devuelta si la conexión usa un canal seguro, como por ejemplo SSL.

RESPONSE

```
HTTP/1.1 200 OK
Set-Cookie:username=Juan
Content-Type=text/html
Date:Wed,19 Nov 2005 03:34:43 GMT
Server:Apache-Coyote/1.1
Connection:close
<html>
...
```

REQUEST

```
POST /select/BeerTest.do HTTP/1.1
Host:www.wickedlysmart.com
User-Agent:Mozilla/5.0
Cookie:username=Juan
Accept:text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language:en-us,en;q=0.5
Accept-Encoding:gzip,deflate
```

API relacionada con las COOKIES

<<interface>> javax.servlet.http.HttpServletRequest

```
String getContextPath()
Cookie [] getCookies()
String getHeader(String)
String getQueryString()
HttpSession getSession()
//MANY more methods
```

<<interface>> javax.servlet.http.HttpServletResponse

```
void addCookie(Cookie cookie)
void addHeader(String name, String value)
String encodeRedirectURL(String URL)
void sendError(int sc, String msg)
void setStatus(int sc)
//MANY more methods
```

javax.servlet.http.Cookie Cookie(String name, String value)

```
String getComment()
String getDomain()
int getMaxAge()
String getName()
String getPath()
boolean getSecure()
String getValue()
int getVersion()
void setComment(String purpose)
void setDomain(String pattern)
void setMaxAge(int expirySeconds)
void setPath(String uri)
void setSecure(boolean flag)
void setValue(String newValue)
void setVersion(int v)
```

El tiempo de vida de una cookie es por default -1, que indica que la cookie persistirá hasta que el browser sea cerrado.

Creando una nueva Cookie

```
Cookie cookie = new Cookie("username", "juan");
```

El constructor Cookie toma un par de nombre/valor String.

Estableciendo cuanto tiempo una cookie vivirá en el cliente

```
cookie.setMaxAge( 30 * 60 );
```

setMaxAge esta definido en segundos. Este codigo dice que vivira en el cliente por 30* 60 segundos (30 minutos)

Enviando una cookie al cliente

```
response.addCookie(cookie);
```

Objteniendo la cookie(s) de la petición del cliente

```
Cookie[] cookies = request.getCookies();
```

API de sesión

Existe una última forma de mantener una conversación entre un cliente y el servidor, que es utilizando la interfaz HttpSession.

El servidor es el que se encargar de crear un identificador para cada cliente automáticamente. Se crea un sesión por medio de una llamada al método getSession() de HttpServletRequest, y persiste hasta que termine su tiempo o sea invalidada por un servlet que participa en la sesión.

Un objeto HttpSession puede sostener el estado conversacional a través de múltiples peticiones de el mismo cliente. En otras palabras, este persiste para la sesión entera con un cliente especifico. Nosotros podemos utilizar esto para almacenar algunas cosas que nosotros queremos de regreso del cliente en toda la petición que el cliente hizo durante la sesión.

COMO TRABAJAN LAS SESIONES:

En la figura 3.12(A) y 3.12(B) se ilustra la manera en que trabajan las sesiones.

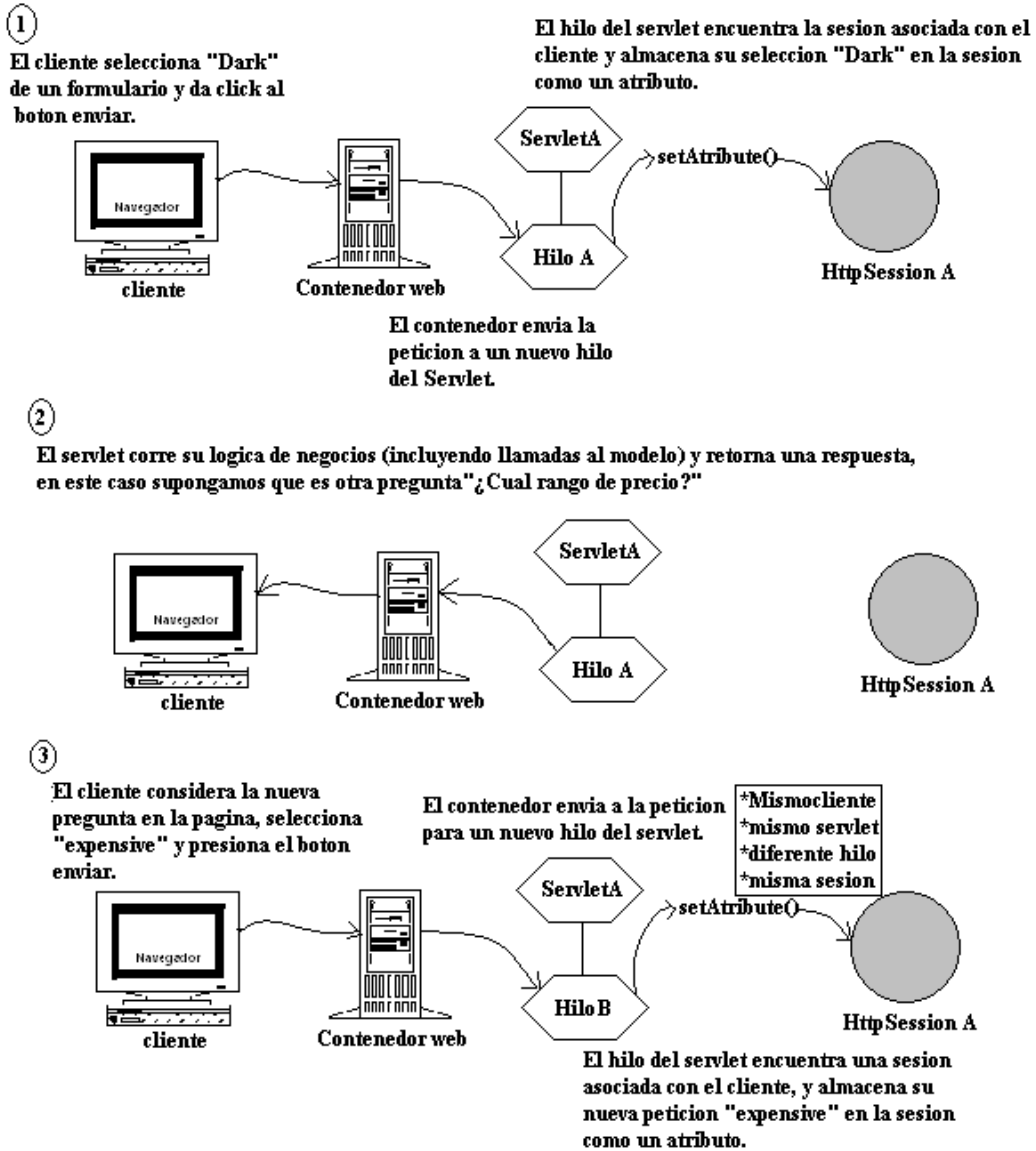
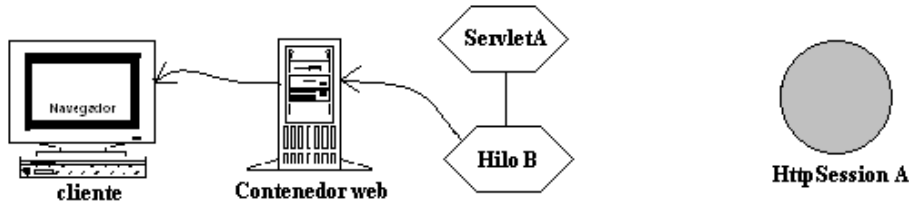


FIGURA 3.12(A) COMO TRABAJAN LAS SESIONES PARTE A

④

El servlet corre la logica de negocios (incluyendo la llamada al modelo) y retorna una respuesta en este caso otra pregunta.



Mientras tanto, imagina que otro cliente visita el mismo sitio web:

⑤

La sesion del primer cliente esta activa, pero mientras tanto el otro cliente selecciona "Yellow" y presiona el boton enviar

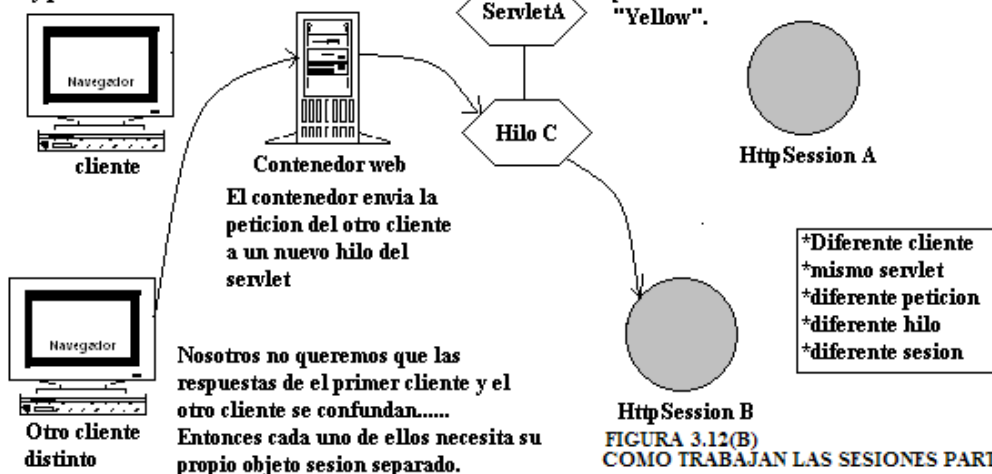


FIGURA 3.12(B)
COMO TRABAJAN LAS SESIONES PARTE B

Creación de sesiones

Un servlet indica que quiere utilizar una sesión invocando a los métodos `getSession()` o `getSession(boolean create)`, como se muestra a continuación:

```
HttpSession session = request.getSession( true );
```

El método `getSession()` sin parámetros es un método de utilidad que simplemente llama a `getSession(true)`. El parámetro `create` indica si el motor de servlets debe crear una nueva sesión si no la hay todavía. Si el parámetro es `false`, el servlet sólo puede actuar sobre la sesión existente. En ambos casos, se examina la petición a ver si contiene un identificador de sesión (session ID) válido. Si es así, el contenedor de servlet devuelve una referencia al objeto sesión, que se puede utilizar a partir de entonces para guardar y recuperar atributos de sesión.

Destrucción de sesiones

Una vez creada, una sesión normalmente persiste hasta que se suspende (cuando termina el tiempo de espera) o se cierra. El tiempo de espera (Timeout) se refiere a la máxima cantidad de tiempo entre peticiones en que la sesión se mantendrá válida. Éste es un punto importante porque el servidor sólo tiene dos formas de saber si un cliente ha terminado de trabajar con una sesión: siendo explícitamente informado o esperando un período de tiempo determinado.

El intervalo de tiempo de espera por defecto se puede establecer en el descriptor de despliegue **web.xml**:

```
<web-app ...>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
</web-app>
```

El intervalo se especifica como cantidad de minutos, 30 por defecto. El valor introducido aquí se aplica a todas las sesiones de la aplicación a menos que éstas lo revoquen de forma individual.

Algunas aplicaciones que usan pocos recursos como las conexiones a base de datos pueden determinar reducir el tiempo de espera. Para ello utilizan el método **setMaxInactiveInterval(int time)** y seleccionan un período de tiempo más corto:

```
session.setMaxInactiveInterval( 180 );
```

El argumento proporcionado a **setMaxInactiveInterval()** es una cantidad de segundos. Si se especifica un valor negativo, la sesión terminará hasta que el navegador se cierre.

En algunos casos se puede proporcionar un final definitivo para la sesión. Para ello se puede usar el método **invalidate()**:

```
session.invalidate( );
```

Interfaz HttpSession

<pre><<interface>> HttpSession Object getAttribute(String name) Enumeration getAttributeNames() long getCreationTime() String getId() long getLastAccessedTime() int getMaxInactiveInterval() ServletContext getServletContext() void invalidate() boolean isNew() void removeAttribute(String name) void setAttribute(String name, Object value) void setMaxInactiveInterval(int interval)</pre>
--

3.2 Java Server Page (JSP)

3.2.1 ¿Qué son las JSP?

Un página de servidor java (JSP, acrónimo del inglés Java Server Pages) es un programa del servidor cuyo diseño y funcionalidad son similares a los servlets. Un cliente llama a una JSP para que le proporcione un servicio web cuya naturaleza depende de la aplicación J2EE. En otras palabras, una JSP es una plantilla para una página Web que emplea código Java para generar un documento HTML dinámicamente. Las páginas JSP se ejecutan en un contenedor de servlets, que las traduce a servlets Java equivalentes, es decir, una JSP llega a ser un servlet, pero un servlet que tu no creas, el contenedor busca tu JSP traduce esto dentro de código Java y compila este dentro de una clase Java servlet.

Por esta razón los servlets y las páginas JSP están íntimamente relacionados. Lo que se puede hacer con una tecnología es, en gran medida, también posible con la otra; aunque cada una tiene capacidades propias. Como son servlets, las páginas JSP tienen todas las ventajas de los servlets:

- Tienen un mayor rendimiento y capacidad de adaptación que las secuencias de comandos CGI porque se conservan en la memoria y admiten múltiples subprocesos.
- No es necesaria una configuración especial por parte del cliente.
- Incorporan soporte para sesiones HTTP, lo que hace posible la programación de aplicaciones.
- Tienen, pleno acceso a la tecnología Java –capacidad de reconocimiento de trabajo en red, subprocesos y conectividad a bases de datos.

Sin embargo, la diferencia entre una JSP y un servlet radica en la forma en que se escribe la JSP. Un servlet se escribe en el lenguaje Java y las respuestas se codifican como objetos String que se envían al método `println()`. El objeto String debe tener formato HTML, XML o cualquier formato que requiera el cliente. A diferencia de esto una JSP se escribe en HTML, XML o en el formato del cliente, entremezclados con elementos de código, directivas y acciones escritas en el lenguaje Java y con sintaxis de JSP.

3.2.2 Ventajas de usar JSP

Crear una JSP es más sencillo que crear un servlet debido a que la JSP esta escrita en HTML, en lugar de lenguaje Java. Esto significa que una JSP no esta llena de los métodos `println()` que se encuentran en un servlet, sin embargo cabe señalar que una JSP proporciona fundamentalmente las mismas características que un servlet, ya que una JSP se convierte en un Servlet la primera vez que el cliente la solicita.

Contra Active Server Pages (ASP).

ASP es una tecnología similar de Microsoft. Las ventajas de JSP están duplicadas. Primero, la parte dinámica está escrita en Java, no en Visual Basic, otro lenguaje específico de MS, por eso es mucho más poderosa y fácil de usar. Segundo, es portable a otros sistemas operativos y servidores Web.

Contra los Servlets.

Como ya se menciona JSP no nos da nada que no pudiéramos en principio hacer con un servlet. Pero es mucho más conveniente escribir (y modificar!) HTML normal que tener que hacer un billón de sentencias println que generen HTML. Además, separando el formato del contenido podemos poner diferentes personas en diferentes tareas: nuestros expertos en diseño de páginas Web pueden construir el HTML, dejando espacio para que los programadores de servlets inserten el contenido dinámico.

Contra JavaScript.

JavaScript puede generar HTML dinámicamente en el cliente. Esta es una capacidad útil, pero sólo maneja situaciones donde la información dinámica está basada en el entorno del cliente. Con la excepción de las cookies, el http y el envío de formularios no están disponibles con JavaScript. Y, como se ejecuta en el cliente, JavaScript no puede acceder a los recursos en el lado del servidor, como bases de datos, catálogos, información de precios, etc.

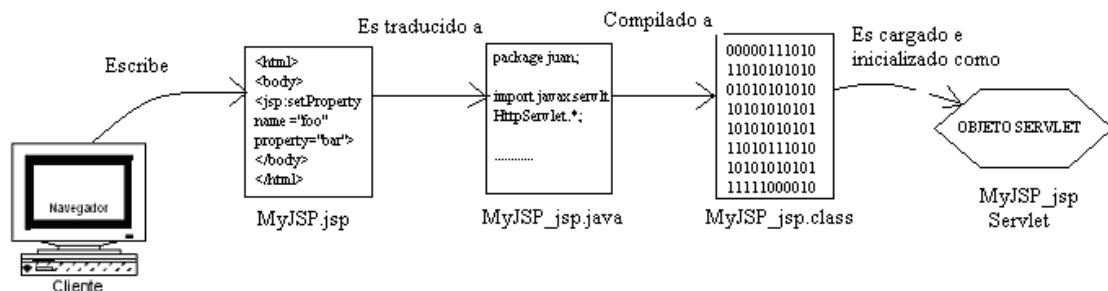


FIGURA 3.13 TRADUCCIÓN DE UNA JSP

La figura 3.13 muestra como una JSP eventualmente llega a ser un servlet corriendo en tu aplicación web. El contenedor toma lo que tu has escrito en tu JSP, traduciendo este dentro de una clase Java servlet archivo (.java), entonces compila este dentro de una clase Java servlet. Después de que este solo es un servlet sigue el camino normal como si tu hubieras compilado una clase java. En otras palabras, el contenedor carga la clase servlet, instancia e inicializa éste, hace un hilo separado por cada petición y llama al método service() de Servlets.

3.2.3 Ciclo de vida de las JSP

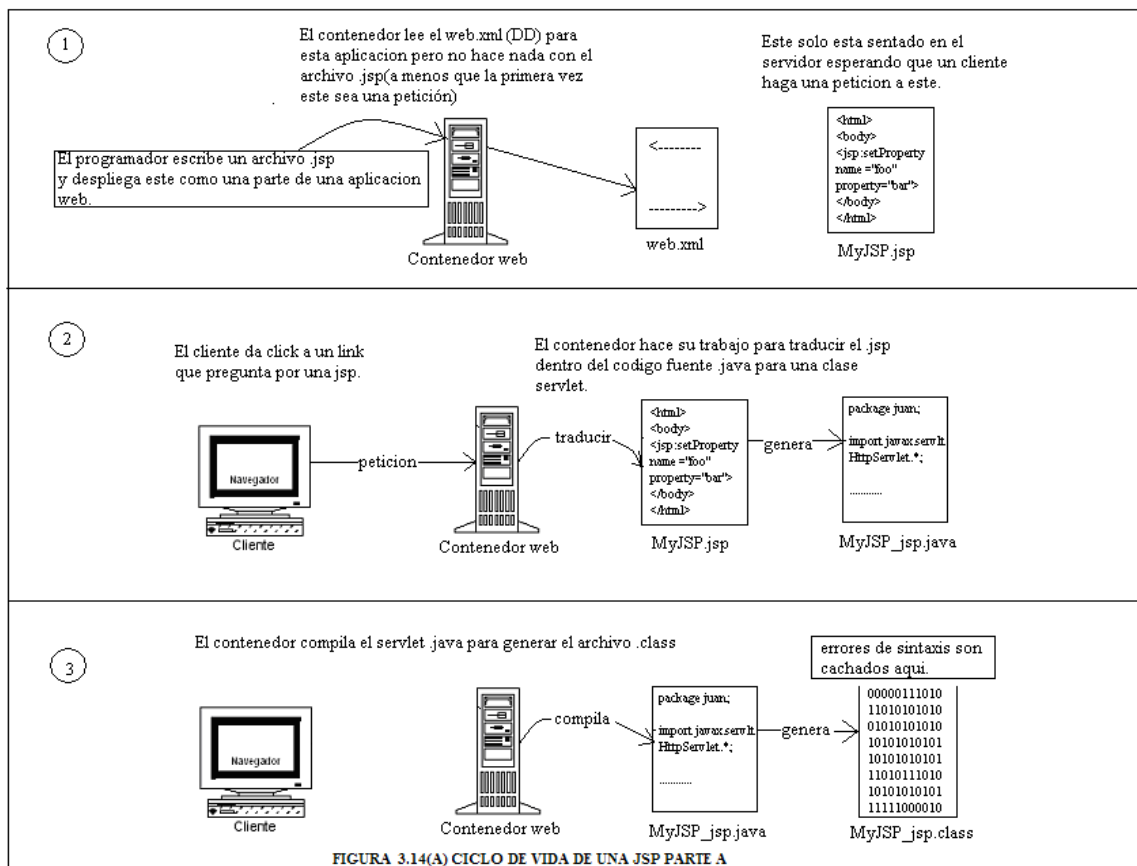
La página JSP pasa por tres etapas en la evolución de su código:

- **Código fuente JSP** Este código es el que realmente escribe el desarrollador. Se encuentra en un archivo de texto con extensión .jsp y consiste en una mezcla de código de plantilla HTML, instrucciones en lenguaje Java, directivas JSP y acciones que describen cómo generar una página Web para dar servicio a una petición concreta.

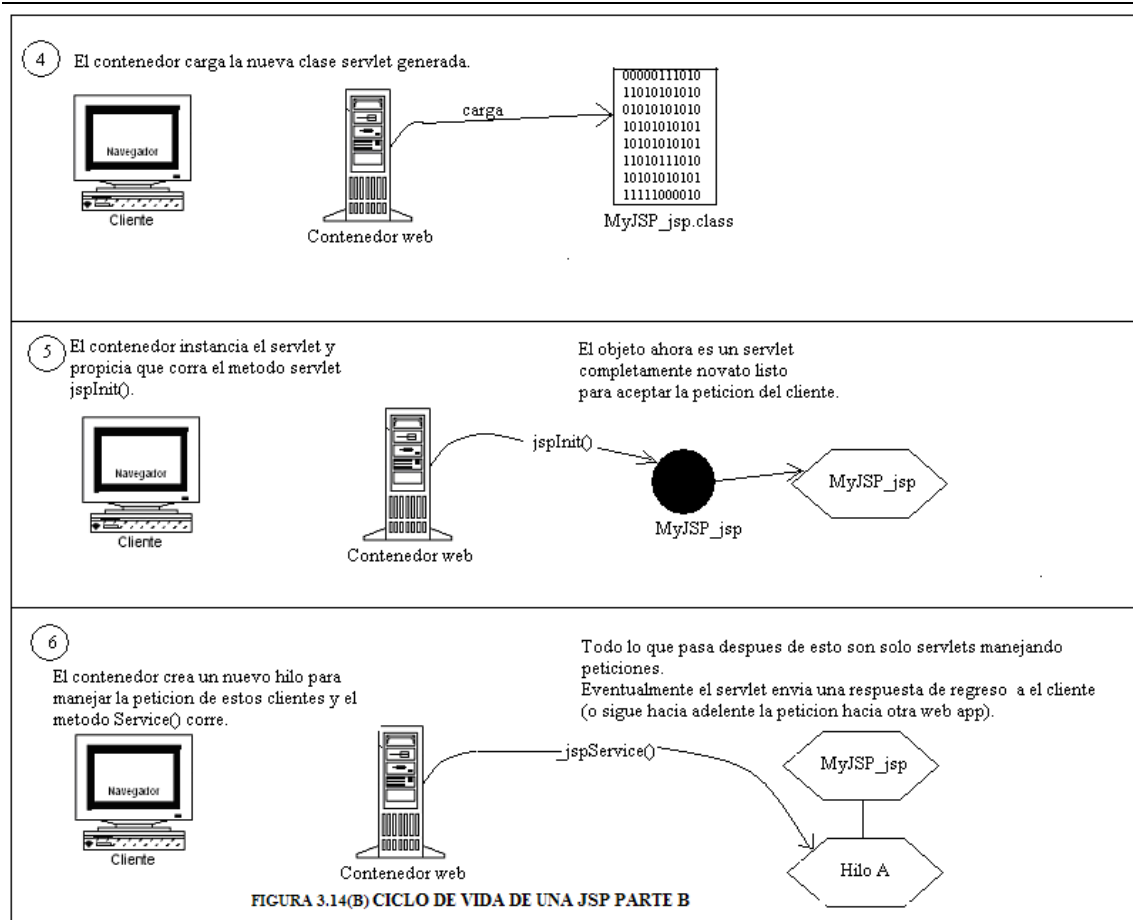
Servlets y Java Server Pages(JSP)

- **Código fuente Java** El contenedor de Servlets traduce el código fuente JSP al código fuente de un servlet Java equivalente. Este código fuente se guarda en un área de trabajo y suele ser útil en el proceso de depuración de errores.
- **Clase Java compilada** Como cualquier otra clase Java, el código del servlet generado se compila en código de bytes en un archivo .class, preparado para ser cargado y ejecutado.

En la figura 3.14(A) y 3.14(B) se ilustra el ciclo de vida de una JSP.



Servlets y Java Server Pages(JSP)



Hay tres métodos que se llaman de forma automática al llamar a una JSP y cuando la JSP termina en forma normal: `jspInit()`, `jspDestroy()` y `service()`. Estos métodos se pueden sobrescribir. Es común sobrescribir los métodos `jspInit()` y `jspDestroy()` en una JSP para realizar operaciones al crear y al destruir la JSP.

El método `jspInit()` es idéntico al método `init()` de un servlet. Este método se llama antes de enviar la primera petición al servlet y se utiliza para inicializar objetos y variables que se utilizan a lo largo de la vida de la JSP.

El método `jspDestroy` es idéntico al método `destroy()` de un servlet. Se llama en forma automática cuando la JSP termina en forma normal. No se llama si la JSP termina en forma abrupta, por ejemplo, si el servidor se cae. El método `destroy()` se utiliza para limpiar y liberar los recursos que utilizó la JSP durante su ejecución, por ejemplo, para desconectar de una base de datos.

El método `service()` se llama en forma automática y envía el resultado de la JSP mediante http.

3.2.4 Componentes de una página JSP

Un archivo .jsp puede contener elementos JSP, datos de plantilla fijos o cualquier combinación de ambos. Los elementos JSP son instrucciones dadas al contenedor de JSP sobre el código a generar y sobre cómo debe operar. Estos elementos tienen etiquetas específicas de comienzo y fin que los identifican ante el compilador de JSP. Los datos de plantilla (normalmente HTML) son todo aquello que el contenedor de JSP no reconoce. Pasan a través del contenedor sin sufrir modificaciones, así que el HTML finalmente generado contiene los datos de plantilla tal y como estaban codificados en el archivo .jsp.

Hay tres tipos de elementos JSP principalmente:

- Directivas
- Elementos de secuencias de comandos (scripts) que incluyen expresiones, scriptlets y declaraciones.
- Acciones

Directivas

Las directivas son instrucciones dirigidas al contenedor de JSP que describen el código que se debe generar. Tienen la forma genérica siguiente:

```
<%@ nombre-de-directiva [ atributo="valor" atributo="valor"] %>
```

Las directivas vienen en tres tipos: page, include y taglib.

La directiva page nos permite definir uno o más de los siguientes atributos sensibles a las mayúsculas:

import="package.class" o **import="package.class1,...,package.classN"**.

Esto nos permite especificar los paquetes que deberían ser importados. Por ejemplo:

```
<%@ page import="java.util.*" %>
```

El atributo **import** es el único que puede aparecer múltiples veces.

contentType="MIME-Type" o

contentType="MIME-Type; charset=Character-Set"

Esto especifica el tipo MIME de la salida. El valor por defecto es text/html. Por ejemplo, la directiva:

```
<%@ page contentType="text/plain" %>
```

tiene el mismo valor que el scriptlet

```
<% response.setContentType("text/plain"); %>
```

isThreadSafe="true|false".

Un valor de true (por defecto) indica un procesamiento del servlet normal, donde múltiples peticiones pueden procesarse simultáneamente con un sólo ejemplar del servlet, bajo la suposición que del autor sincroniza las variables de ejemplar. Un valor

de **false** indica que el servlet debería implementar `SingleThreadModel`, con peticiones enviadas serialmente o con peticiones simultáneas siendo entregadas por ejemplares separados del servlet.

session="true|false".

Un valor de **true** (por defecto) indica que la variable predefinida `session` (del tipo **HttpSession**) debería unirse a la sesión existente si existe una, si no existe se debería crear una nueva sesión para unirla. Un valor de **false** indica que no se usarán sesiones, y los intentos de acceder a la variable `session` resultarán en errores en el momento en que la página JSP sea traducida a un servlet.

buffer="sizekb|none".

Esto especifica el tamaño del buffer para el **JspWriter out**. El valor por defecto es específico del servidor, debería ser de al menos **8kb**.

autoflush="true|false".

Un valor de **true** (por defecto) indica que el buffer debería descargarse cuando esté lleno. Un valor de **false**, raramente utilizado, indica que se debe lanzar una excepción cuando el buffer se sobrecarga. Un valor de **false** es ilegal cuando usamos **buffer="none"**.

extends="package.class".

Esto indica la superclase del servlet que se va a generar. Debemos usarla con extrema precaución, ya que el servidor podría utilizar una superclase personalizada.

info="message".

Define un string que puede usarse para ser recuperado mediante el método **getServletInfo**.

errorPage="url".

Especifica una página JSP que se debería procesar si se lanzará cualquier `Throwable` pero no fuera capturado en la página actual.

isErrorPage="true|false".

Indica si la página actual actúa o no como página de error de otra página JSP. El valor por defecto es **false**.

language="java".

En algunos momentos, esto está pensado para especificar el lenguaje a utilizar. Por ahora, no debemos preocuparnos por él ya que `java` es tanto el valor por defecto como la única opción legal.

La sintaxis XML para definir directivas es:

```
<jsp:directive.TipoDirectiva atributo=valor />
```

Por ejemplo, el equivalente XML de:

```
<%@ page import="java.util.*" %>
```

es:

```
<jsp:directive.page import="java.util.*" />
```


Expresiones

JSP proporciona un medio sencillo para obtener acceso al valor de una variable Java u otra expresión y unir ese valor con el HTML de la página. La estructura sintáctica es:

```
<%= exp >
```

donde exp es cualquier expresión Java válida. La expresión puede tomar cualquier valor como un dato, mientras pueda convertirse en una cadena. Esta conversión se hace normalmente generando una instrucción `out.print()`. Por ejemplo el código JSP:

```
La hora actual es <%= new java.util.Date() %>
```

puede generar el código de servlet:

```
out.write("La hora actual es ");  
out.print( new java.util.Date() );
```

Scriptlets

Un scriptlet es un conjunto de una o más sentencias o instrucciones (statements) en lenguaje Java concebidas para su uso en el procesamiento de una petición HTTP. La sintaxis de un scriptlet es:

```
<% instrucción; %>
```

El contenedor JSP simplemente incluye el contenido del scriptlet textualmente en el cuerpo del método `_jspService()`. Una página Java puede contener tantos scriptlets como se deseen. Si hay múltiples scriptlets, cada uno de ellos se anexa al método `_jspService()` en el orden en el que están codificados. Por lo tanto, un scriptlet puede dejar una llave abierta y cerrarla en otro scriptlet.

Los Scriptlets tienen acceso a las mismas variables predefinidas que las expresiones. Por eso, por ejemplo, si queremos que la salida aparezca en la página resultante, tenemos que usar la variable **out**:

```
<%  
String queryData = request.getQueryString();  
out.println("Attached GET data: " + queryData);  
%>
```

Observa que el código dentro de un scriptlet se insertará exactamente como está escrito, y cualquier HTML estático (plantilla de texto) anterior o posterior al scriptlet se convierte en sentencias `print`. Esto significa que los scriptlets no necesitan completar las sentencias Java, y los bloques abiertos pueden afectar al HTML estático fuera de los scriptlets.

Por ejemplo, el siguiente fragmento JSP, contiene una mezcla de texto y scriplets:

```
<% if (Math.random() < 0.5) { %>
Have a <B>nice</B> day!
<% } else { %>
Have a <B>lousy</B> day!<% } %>
```

que se convertirá en algo como esto:

```
if (Math.random() < 0.5) {
out.println("Have a <B>nice</B> day!");
} else {
out.println("Have a <B>lousy</B> day!");
}
```

Si queremos usar los caracteres "%>" dentro de un scriptlet, debemos poner "%\>". Finalmente, observa que el equivalente XML de **<% Código %>** es

```
<jsp:scriptlet>
Código
</jsp:scriptlet>
```

Declaraciones

Al igual que los scriptlets, las declaraciones contienen instrucciones (statements) en lenguaje Java, pero con una gran diferencia: el código del scriptlet se convierte en parte del método `_jspService()`, mientras el código de la declaración se incorpora en el fichero fuente generado fuera del método `_jspService()`. La sintaxis de una declaración es:

```
<%! Instrucción; [ instrucción;] %>
```

Las secciones de la declaración se pueden usar para declarar clases o variables de instancia, métodos o clases internas.

Como las declaraciones no generan ninguna salida, normalmente se usan en conjunción con expresiones JSP o scriptlets. Por ejemplo, aquí tenemos un fragmento de JSP que imprime el número de veces que se ha solicitado la página actual desde que el servidor se arrancó (o la clase del servlet se modificó o se recargó):

```
<%! private int accessCount = 0; %>
Accesses to page since server reboot:
<%= ++accessCount %>
```

Acciones estándar

Las acciones son elementos JSP de alto nivel que crean, modifican o utilizan otros objetos. A diferencia de las directivas y elementos de secuencias de comandos, las acciones están codificadas usando solamente sintaxis XML.

Servlets y Java Server Pages(JSP)

`<nombre-de-etiqueta [atr="valor" atr="valor" ..]> ... </nombre-de-etiqueta>`

o si la acción no tiene cuerpo, una forma abreviada:

`<nombre-de-etiqueta [atr="valor" atr="valor" ..] />`

Las acciones JSP usan construcciones de sintaxis XML para controlar el comportamiento del motor de Servlets.

Podemos insertar un fichero dinámicamente, reutilizar componentes JavaBeans, reenviar al usuario a otra página, o generar HTML para el plug-in Java. Las acciones disponibles incluyen:

jsp:include - Incluye un fichero en el momento de petición de esta página.

jsp:useBean - Encuentra o ejemplariza un JavaBean.

jsp:setProperty - Selecciona la propiedad de un JavaBean.

jsp:getProperty - Inserta la propiedad de un JavaBean en la salida.

jsp:forward - Reenvía al peticionario a una nueva página.

jsp:plugin - Genera código específico del navegador que crea una etiqueta OBJECT o EMBED para el plug-in Java.

Recuerda que, como en XML, los nombre de elementos y atributos son sensibles a las mayúsculas.

3.2.5 Objetos implícitos

Aunque los scriptlets, las expresiones y los datos de la plantilla HTML están incorporados dentro del método `_jspService()`, el contenedor de JSP escribe el esqueleto del propio método, inicializando el contexto de página y una serie de variables de utilidad. Estas variables están disponibles implícitamente dentro de los scriptlets y en las expresiones (pero no en las declaraciones). Se puede obtener acceso a ellas, como a cualquier otra variable, pero no es preciso declararlas primero.

API	Implicit Object
JspWriter	out
HttpServletRequest	request
HttpServletResponse	response
HttpSession	session
ServletContext	application
ServletConfig	config
JspException	exception
PageContext	pageContext
Object	page

3.2.6 Java Standard Tag Library (JSTL)

JSTL no es más que un conjunto de librerías de etiquetas simples y estándares que encapsulan la funcionalidad principal que es usada comúnmente para escribir páginas JSP.

Sabemos que no es muy conveniente utilizar scriptless en una jsp, pero cuando desarrollamos JSP, a veces es necesario realizar tareas como iteración a través de un arreglo, comparación de condiciones o remover variables. Para estas tareas, se encuentran una serie de etiquetas o tags que realizan estas tareas y que se encuentran en todos los contenedores de servlets.

El problema de usar scriptless en una jsp es básicamente porque el código java embebido en una jsp es desordenado y difícil de depurar, el código java que existe dentro de una jsp no puede ser reutilizado por otra clase java

¿Cuales son las desventajas de los JSTL?

Los JSTL pueden agregar mas sobrecarga al servidor, debido a que las etiquetas JSTL generan mas código en el servlet traducido, pero en la mayoría de los casos esta cantidad no es mensurable pero debe ser considerado.

Los scriptless son mas potentes que los JSTL .A pesar que las etiquetas JSTL proporciona un potente conjunto de librerías reutilizables, no puede hacer todo lo que el código Java puede hacer. La librería JSTL está diseñada para facilitar la codificación en el lado de presentación que es típicamente encontrado en la capa de Vista si hablamos de la arquitectura Modelo-Vista-Controlador.

Core Library de JSTL

Propósito General	Condicional	URL	Iteración
<code><c:out></code>	<code><c:if></code>	<code><c:import></code>	<code><c:forEach></code>
<code><c:set></code>	<code><c:choose></code>	<code><c:url></code>	<code><c:forTokens></code>
<code><c:remove></code>	<code><c:when></code>	<code><c:redirect></code>	
<code><c:catch></code>	<code><c:otherwise></code>	<code><c:param></code>	

Ejemplo:

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    pageContext.setAttribute("movies", new String[]{
        "Matrix Revolutions", "Kill Bill",
        "Amelie", "Casablanca"},
        PageContext.REQUEST_SCOPE);
%>
<html>
  <body>
    <table>
      <c:forEach var="movie" items="${movies}" varStatus="contador">
        <tr><td>${contador.count}</td><td>${movie}</td></tr>
      </c:forEach>
    </table>
  </body>
</html>
```

Propósito General

<c:out>

```
<c:out value="{variable}" escapeXml="true" default="hola"/>
```

<c:set>

Set viene en dos sabores: **var** y **target**. La versión var es para establecer atributos, la versión target es para establecer propiedades de beans o valores de Maps. Cada uno de los dos sabores viene en dos variaciones: con o sin cuerpo. El <c:set> body es solo otro lugar donde poner el valor.

Estableciendo un atributo con <c:set> (var)

SIN body

```
<c:set var="variable_var" value="{cadena}" scope="page" />
```

CON a body

```
<c:set var="variable_var" scope="page">
```

Valores establecido en el body

```
</c:set>
```

Note: Si el valor establecido evalúa a null, la variable será removida.

Usando <c:set> con beans y Maps (target)

SIN body

```
<c:set target="{person}" property="dogName" value="Clover"
scope="page" />
```

CON body

```
<c:set target="{person}" property="dogName" scope="application" >
```

Valores establecido en el body

```
</c:set>
```

Nota: "target" debe evaluar a un OBJECT. Si la expresión en "target" no es un Map o un bean, el Contenedor lanzara una excepción, también el Contenedor lo hará sino encuentra la propiedad.

<c:remove>

```
<c:remove var="userStatus" scope="request" />
```

<c:catch>

```
<c:catch var="myException">
```

Inside the catch { 10 % 0 }

```
</c:catch>
```

```
{myException.message}
```

Condicional

<c:if>

```
<c:if test="{var == null}" var="mylf" scope="request">
  Verdadero {mylf}
</c:if>
```

<c:choose>, <c:when> and <c:otherwise>

```
<c:choose>
  <c:when test="{ userPref == 'performance'}">
    Performance
  </c:when>
  <c:when test="{userPref == 'safety'}">
    Security
  </c:when>
  <c:otherwise>
    Performance and Security
  </c:otherwise>
</c:choose>
```

URL related

<c:import>

```
<c:import url="http://www.misitio.com" var="contenido" scope="session"/>

<c:import url="header.jsp">
  <c:param name="subTitle" value="We use SOAP" />
</c:import>
```

<c:url>

```
<c:url value="/input.jsp" var="url_jsessionid" scope="application" />
```

Codificar parámetros

```
<c:url value="/input.jsp" var="url_jsessionid" scope="application">
  <c:param name="firstName" value="{name}" />
  <c:param name="lastName" value="{last}" />
</c:url>
```

<c:redirect>

Redirecciona a un nuevo URL

```
<c:redirect url="http://www.misitio.com" />
```

Iteración

<c:forEach>

```
<c:forEach var="movie"
  items="{movieList}"
  varStatus="contador"
  begin="1"
  end="3" step="2">
  Contador : ${contador.count} movie : ${movie}<br>
</c:forEach>
```

<c:forTokens>

```
<c:forTokens var="movie"
  items="{movieSentence}"
  delims=" "
  varStatus="contador"
  begin="0"
  end="5">
  Contador : ${contador.count} movie : ${movie}<br>
</c:forTokens>
```


3.2.7 Etiquetas Personalizadas (Custom Tags)

Las etiquetas personalizadas son extensiones similares a XML de la sintaxis y la semántica de las páginas JSP, que están respaldadas por manejadores de etiquetas creados por el usuario. Las colecciones de etiquetas se organizan en bibliotecas de etiquetas que se pueden empaquetar como archivos JAR, permitiendo así que su funcionalidad sea fácilmente distribuida e instalada en cualquier motor de servlets.

Ejemplo:

JSP that uses the tag

```
<%@taglib prefix="mine" uri="ejemploTaglib" %>

<mine:ejemTag nombre="Juan">
    Body que se ejecuta<br>
</mine:ejemTag>
```

TLD file

```
<?xml version="1.0" encoding="UTF-8">
<taglib version="2.0" ..>
  <tlib-version>1.0</tlib-version>
  <short-name>mitaglib</short-name>
  <uri>ejemploTaglib</uri>
  <tag>
    <description>Ejemplo de una taglib</description>
    <name>ejemTag</name>
    <tag-class>com.taglib.EjemploTaglib</tag-class>
    <body-content>scriptless</body-content>

    <attribute>
      <name>nombre</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

Handler class

```
package com.taglib;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

public class EjemploTaglib extends SimpleTagSupport{
    private String nombre;
    public void setNombre(String nombre){
        this.nombre = nombre;
    }
    public String getNombre(){
        return nombre;
    }
    public void doTag() throws JspException, IOException {
        getJspContext().getOut().print("Nombre = " + nombre);
        getJspBody().invoke(null);
    }
}
```

ARQUITECTURA DEL FRAMEWORK STRUTS

4.1 ¿Qué son las aplicaciones frameworks?

Un framework es una aplicación reusable, semi-completa que puede ser especializada para producir aplicaciones personalizadas. En general, con el término Framework, nos estamos refiriendo a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un Framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

Los objetivos principales que persigue un Framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Un Framework Web, por tanto, podemos definirlo como un conjunto de componentes (por ejemplo clases en java y descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web.

Un framework provee a los desarrolladores para establecer un esqueleto de componentes que tienen las siguientes características:

- Ellos tienen que saber trabajar bien en otras aplicaciones.
- Ellos están listos para usarse con los siguientes proyectos.
- Ellos pueden también ser usados por otros equipos en la organización.

4.2 ¿Qué son los Struts?

Los Struts es un tipo de framework, que nos proporciona las siguientes características:

- Una aplicación framework Java Modelo 2.
- Basado en servlets, jsp y XML
- Open source parte de Apache Jakarta project
- Free software (incluso para uso comercial)
- Nos ayuda a construir aplicaciones web mas rápido y mas fácil.

La pieza centro de Struts esta en el estilo controlador MVC. El controlador Struts acorta el hueco que hay entre Modelo y vista. El modelo también incluye otras piezas que necesitan los desarrolladores para escribir aplicaciones web escalables. Struts es una colección de "sostenes invisibles" que ayudan a los desarrolladores regresar la materia prima como base de datos y paginas web dentro de una coherente aplicación.

Arquitectura del framework Struts

El controlador Struts es un juego de componentes programables que permiten a los desarrolladores definir exactamente como su aplicación interactúa con el usuario. Estos componentes ocultan lo sucio, detalles incómodos de implementación detrás de nombres lógicos. Los desarrolladores pueden programar estos detalles una vez, entonces regresan para pensar en términos de cómo el programa lo hace mejor de lo que ya esta hecho.

Los usuarios interactúan con las aplicaciones web por hiperlinks y formularios html. Los hiperlinks conducen a paginas para desplegar datos y otros elementos como imágenes y texto. Los formularios generalmente envían datos para la aplicación vía algunas acciones de costumbre custom action.

Como se muestra en la figura 4.1 Struts provee componentes que los programadores pueden usar para definir los hyperlinks, forms y custom action que las aplicaciones web usan para interactuar con el usuario.

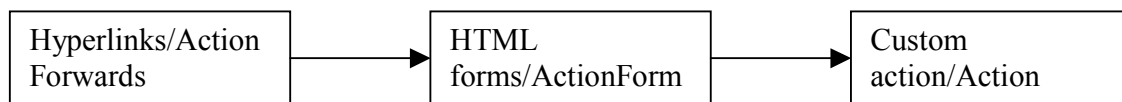


FIGURA 4.1 Struts components

NOTA: Los componentes Struts son configurados vía XML. En la practica los elementos de configuración son una parte integral de los frameworks Struts.

4.3 Componentes de Struts

4.3.1 Hyperlinks

Para los desarrolladores de aplicaciones, un hyperlink es una ruta de algunos recursos en la aplicación. Este puede ser una página web o un custom action. Esto puede incluir también parámetros especiales. En Struts los desarrolladores pueden definir un hyperlink con un ActionForward. Estos objetos tienen un nombre lógico y una propiedad path. Esto permite a los desarrolladores establecer el path y entonces referirse al ActionForward por nombre.

Los ActionForward son usualmente definidos en un archivo de configuración XML que Struts lee cuando la aplicación web carga. Struts usa las definiciones del XML para crear la configuración de Struts, el cual incluye una lista de ActionForwards. Los elementos XML que podrían crear un ActionForward para un hyperlink de bienvenida pueden verse como este:

```
<forward  
  name="welcome"  
  path="/pages/index.jsp"/>
```

Estos elementos podrían crear un ActionForward JavaBean con esta propiedad nombre establecida en welcome y la propiedad path establecida como /pages/index.jsp.

Las páginas jsp y otros componentes pueden entonces enviarse a la página welcome forward. El Framework Struts se vera en el ActionForward welcome bean y recuperara

el path completo para el hyperlink. Esto permite a los desarrolladores cambiar el destino de un link sin cambiar todos los componentes que envían a este link. En muchas aplicaciones web destacan como esto es código difícil dentro de una jsp y código Java, haciendo cambios difíciles y son propensas a errores. En una aplicación Struts estos detalles pueden ser cambiados en cualquier parte de la aplicación sin tocar una sola página o clase Java.

4.3.2 HTML Forms

Los protocolos web, http y HTML, proveen un mecanismo para enviar datos de un formulario pero recibir los datos de salida es un ejercicio para los desarrolladores. El Struts framework provee una clase `ActionForm`, que es diseñada para manejar input de un formulario HTML, validar el input y redespigar el form al usuario para corrección (cuando sea necesario) adelante con un correspondiente prompt o mensaje.

Los `ActionForms` son solo `JavaBeans` con una pareja de métodos standard para administrar el ciclo de validación y revisión. Struts automáticamente hace juego con las propiedades `JavaBeans` con atributos de controles HTML. Los desarrolladores definen la clase `ActionForm`, Struts hace el resto.

Esta clase automáticamente habitara el campo `username` de un formulario con un elemento del formulario HTML con el mismo nombre, como se muestra aquí:

```
Public final class LogonForm extends ActionForm{
    Private String username = null;

    Public String getUsername(){
        Return (this.username);
    }

    public void setUsername(String username)
    {
        this.username = username;
    }

}
```

Otras propiedades podrían ser agregadas para cada campo en el formulario. Esto permite a otros componentes obtener lo que ellos necesitan de un standard `JavaBean`, entonces no todos tienen que separar por una petición http.

4.3.3 ActionForm

Las clases ActionForm son creadas usando clases Java normales. La configuración de Struts envía a las clases ActionForm establecido por el descriptor: los elementos <form-beans> y <form-bean>. El elemento <form-bean> son descriptores que el framework usa para identificar e instanciar los objetos ActionForm, como se muestra aquí:

```
<form-bean  
    name="articleForm">  
    type="org.apache.artimus.struts.Form"/>
```

El Struts configuration lista los beans ActionForm para usarlos y dar a las clases Action-Form un nombre lógico para usar con la aplicación.

4.3.4 Custom actions

Un formulario HTML usa un parámetro action para decirle al navegador donde enviar los datos del formulario. El framework Struts suministra una correspondiente clase Action para recibir cada dato. El framework automáticamente crea, habita, valida y finalmente pasa la propiedad ActionForm en el objeto Action. El Action puede entonces obtener el dato que este necesita directamente del bean ActionForm. Aquí esta un ejemplo:

```
Public final class LogonAction extends Action {  
    Public ActionForward perform(ActionMapping mapping,  
    ActionForm form,  
    HttpServletRequest request,  
    HttpServletResponse response) throws IOException, ServletException{  
        MyForm myForm = (MyForm) form;  
        //.....  
        return mapping.findForward("continue");  
    }  
}
```

Un Action concluye por regresar un objeto ActionForward para el controlador. Esto permite al Action escoger una definición usando un nombre lógico, como continue o cancel, en lugar de rutas de sistema.

Para asegurar la extensibilidad, el controlador también pasa el objeto petición y respuesta en curso. En la practica un Action puede hacer cualquier cosa que un Java Servlet puede hacer.

Los objetos ActionForward, ActionForm y Action, la capa del controlador Struts proveen algunos otros componentes especializados incluyendo ActionMapping y ActionServlet. Struts también da soporte de localizar tu aplicación de la capa del controlador.

4.3.5 Actionmapping

En una aplicación web, todos los recursos tienen que ser referenciados por un Identificador de Recursos Uniformes (URI). Estos incluyen HTML, paginas, paginas jsp, y algunos custom action. Para dar al custom action un URI, o path, el Framework Struts provee un objeto ActionMapping. Como el ActionForward y ActionForm, los mappings son usualmente definidos en el archivo de configuración XML.

<action-mapping>

```
<action  
  path="/logonSubmit"  
  type="app.LogonAction"  
  name="logonForm"  
  scope="request"  
  validate="true"  
  input="/pages/logon.jsp"/>
```

</action-mapping>

esto también permite al mismo objeto Action para ser usado por diferentes mappings. por ejemplo un mapping puede requerir validación, otro puede que no.

4.3.6 Actionservlet

El Struts ActionServlet trabaja silenciosamente detrás de escena, encuadrando los otros componentes juntos. Aunque esto puede ser una subclase muchos desarrolladores Struts tratan el ActionServlet como una caja negra: ellos configuran este y sale esto solo.

4.4 Modelo Vista Controlador (MVC)

Comenzaremos describiendo este modelo con un escenario para que sea mas comprensible el uso del MVC. Imagina que vas a desarrollar una aplicación web, tu sabes Java y has leído acerca de los Servlets entonces harás un diseño rápido y comenzarás con el código.

Entonces comenzarás a construir un montón de Servlets uno por cada página, es decir, considerar tener un Servlet por cada responsabilidad, cada Servlet tendrá la lógica de negocios que este necesita para modificar o leer la base de datos y pintar el HTML del flujo de respuesta al cliente. Por ejemplo:

```
//imports statements

public class DatingServlet extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
    IOException
    {
        //la lógica de negocios ira aquí dependiendo de que hará supuestamente este
        Servlet(escribir a una base de datos, hacer una consulta o leer una consulta, etc)

        PrintWriter out = response.getWriter();

        //composición de la pagina dinámica
        out.println("cosas realmente feas van aqui");

    }

}
```

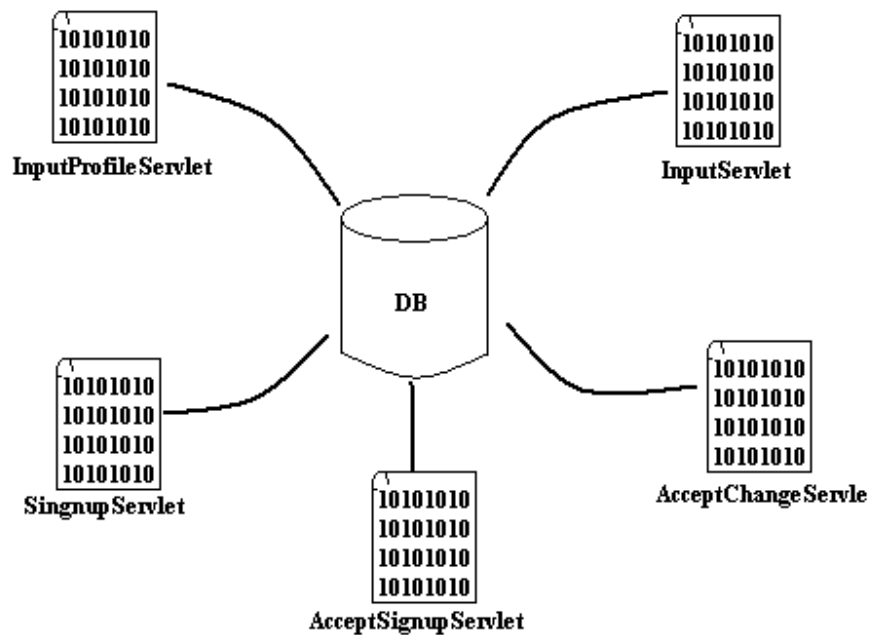


FIGURA 4.2 DISEÑO DE UNA APLICACIÓN WEB UTILIZANDO SOLO SERVLETS

Tu podrías creer que este es un gran diseño orientado a objetos, todos los Servlets tienen exactamente un trabajo como se puede observar en la figura 4.2, toda la lógica de negocios y la pagina HTML de respuesta al cliente esta dentro del código Servlet, pero obtendrás cosas desagradables entonces se te ocurre utilizar JSP's.

El utilizar los `out.println()` para la salida de respuesta es realmente desagradable y decides tener para cada Servlet la lógica de negocios que necesitas (consultas a la base de datos, insertar y actualizar datos en la base de datos, registros, etc). Entonces la petición avanzara a la JSP como un HTML (la vista), para la respuesta (Figura 4.3). Entonces dices esto ya separa la lógica de negocios de la presentación (vista). Por ejemplo:

//imports statements

public class DatingServlet extends HttpServlet {

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

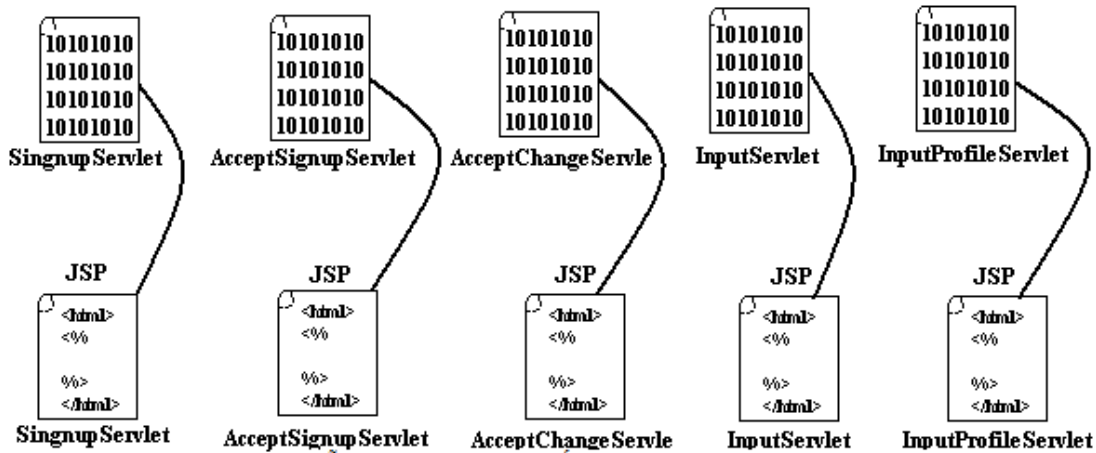
{

//la logica de negocios ira aqui dependiendo de que hara supuestamente este Servlet (escribir a una base de datos, hacer una consulta o leer una consulta, etc)

//mandara la petición a una pagina JSP especifica en lugar de pintar HTML en el flujo de salida.

}

}



Pero entonces una amigo te dice: Realmente estas usando MVC? Claro tu responderás por supuesto que si, pero que hay si la aplicación requiere también el uso de una interfaz grafica GUI Swing, tu amigo dirá bueno esto no es un problema porque tu estas seguro de que utilizaste MVC.

Tu responderás que solo separaste la presentación de la lógica de negocios, esto es un buen comienzo pero toda la lógica de negocios estará dentro en los Servlets.

La esencia de MVC es que tu separas la lógica de negocios de la presentación, pero tu pones algunas cosas entre estos, entonces la lógica de negocios puede estar en una clase Java con código totalmente reusable y no es necesario que tenga que saber algunas cosas de la vista. Entonces sabemos que no es recomendable mezclar la lógica de negocios en un Servlet, esto quiere decir, que no se podrá usar la lógica de negocios para una aplicación GUI swing o alguna otra interfaz, el patrón de diseño MVC repara esto:

El Modelo Vista Controlador toma la lógica de negocios fuera del Servlet y pone este en un "MODELO" una clase Java reusable. El modelo es una combinación de datos de negocio(como el estado de una tarjeta de compras) y los métodos(reglas) que operan en estos datos (Figura 4.4 y 4.5).

Arquitectura del framework Struts

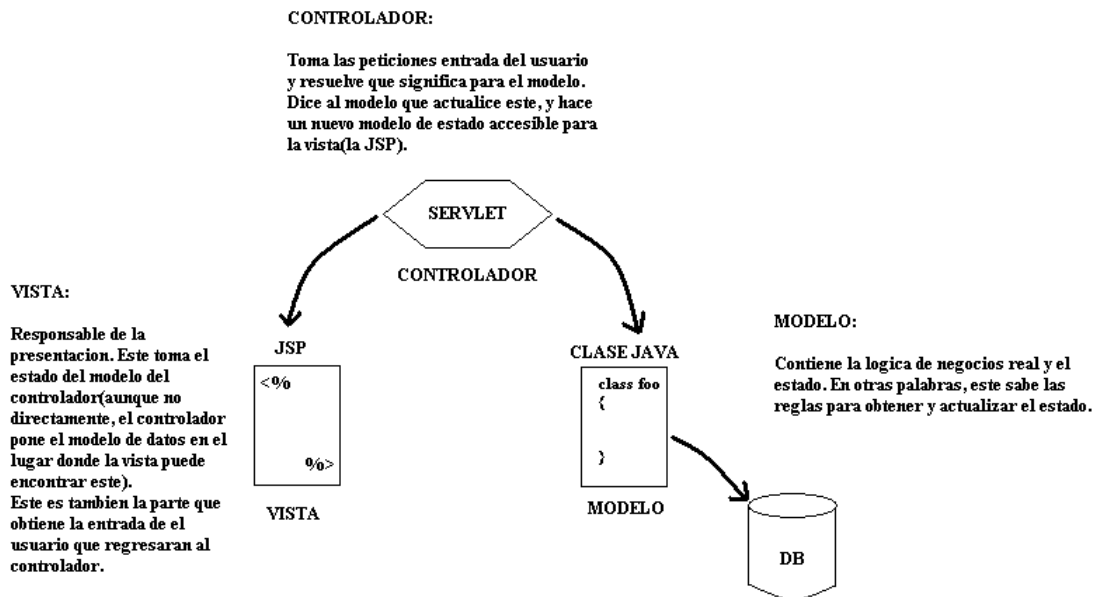
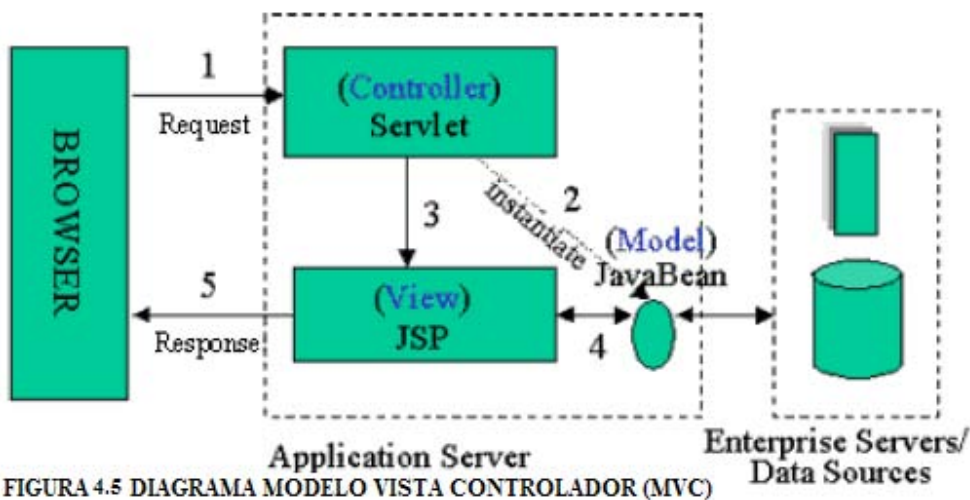


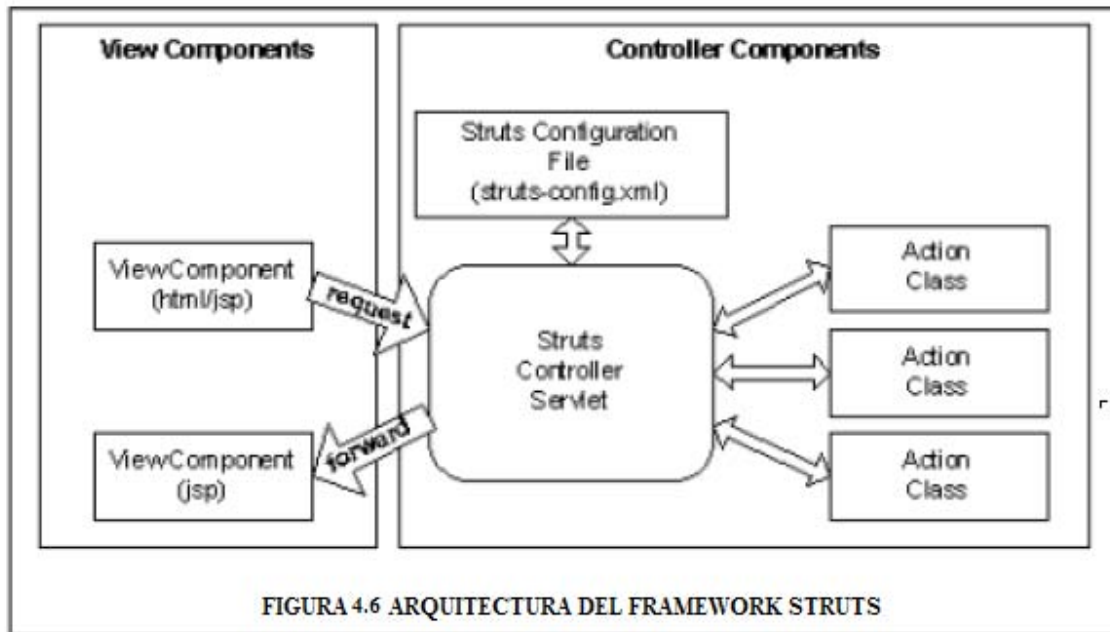
FIGURA 4.4 MODELO VISTA CONTROLADOR (MV)



Esta arquitectura consiste a grandes rasgos en la utilización de Servlets para procesar las peticiones (controladores) y paginas JSP para mostrar la interfaz de usuario (vistas), implementando la parte del modelo mediante JavaBeans.

4.5 Struts Arquitectura

Struts ha sido desarrollado en Java mediante Servlets y esta basado en el modelo 2, el cual es una variante del patrón MVC (Figura 4.6).



Struts ofrece un sistema de tuberías que permite la comunicación mediante el modelo que contiene los datos y las vistas que ofrecen estos datos a los usuarios y reciben sus ordenes.

Struts provee un controlador Servlet que puede ser usado para manejar el flujo entre paginas JSP y otro dispositivo de capa de presentación. Struts implementa el patrón MVC/capas con el uso de ActionForwards y ActionMappings para guardar el flujo de control de decisiones de salida de la capa de presentación. Las JSP pueden referirse a destinos lógicos. Los componentes del controlador proveen el actual URI en tiempo de ejecución.

En la siguiente tabla describe las responsabilidades de los componentes MVC en Struts:

Clases:	Descripción
ActionForward	Detalle de usuario o selección de vista
ActionForm	Los datos para un cambio de estado
ActionMapping	Los estados de cambio de eventos
ActionServlet	La parte del controlador que recibe detalles de usuarios y estados de cambio y cuestiones de vista de selección.

Arquitectura del framework Struts

Action clases	La parte del controlador que interactua con el modelo para ejecutar un cambio de estado o consultas y consejos del ActionServlet de la siguiente vista para seleccionar.
---------------	--

Complementando Struts usa un numero de archivos de configuración y vistas de ayuda como puente para el hueco entre el Controlador y el Modelo, que a continuación se describen en la siguiente tabla:

File	Propósito
ApplicationResources.properties	Almacena mensajes y etiquetas para que tu aplicación pueda ser internacionalizada.
Struts-config.xml	Almacena la configuración por default para los objetos del controlador que incluyen los detalles, los cambios de estado y consultas de estado soportados por el modelo.

Para exponer los datos en la configuración de Struts para la Vista, el framework provee un numero de ayudantes en el formulario para etiquetas JSP, que a continuación se muestran:

Tag Library Descriptor	Proposito
struts-html.tld	Estenciones de etiquetas JSP para formularios HTML.
struts-beans.tld	Estenciones de etiquetas JSP para el manejo de JavaBeans.
Struts-logic.tld	Estenciones de etiquetas JSP para probar el valor de las propiedades.

Ahora pongamos todo junto en los componentes de Struts por capas.

Capa de Vista	Capa Controlador	Capa Modelo
Estenciones de etiquetas JSP	ActionForwards ActionForm clases ActionMapping ActionServlet Action clases ActionErrors MessageResource	GenericDataSource

4.6 Flujo de Control Struts

Un cliente pide una ruta que hace juego con el patrón Action URI .

El contenedor pasa la petición al ActionServlet.

Si esta es una aplicación modular, el actionservlet selecciona al modulo apropiado.

El ActionServlet busca el mapping para el path.

Si el mapping especifica un form bean, el ActionServlet vee si hay uno listo o crea uno.

Si el form bean esta en juego, el ActionServlet resetea y puebla este de la petición http.

Si el mapping tiene la propiedad validate como true, este llama a validate en el form bean.

Si este falla, el servlet continua a la ruta o path especificado por la propiedad input y este flujo de control finaliza.

Si el mapping especifica un tipo Action, este es usado si este esta listo o instanceado.

La ejecución del Action o la ejecución del método es llamado y pasado el form bean instanceado.

El Action puede alojar el form bean, llama al objeto de negocios, y lo hace independientemente si no es necesario.

El Action retorna un ActionForward para el ActionServlet.

En el ActionForward esta el otro Action URI.

En la figura 4.7 se muestra el diagrama de secuencia de l flujo de control de Struts.

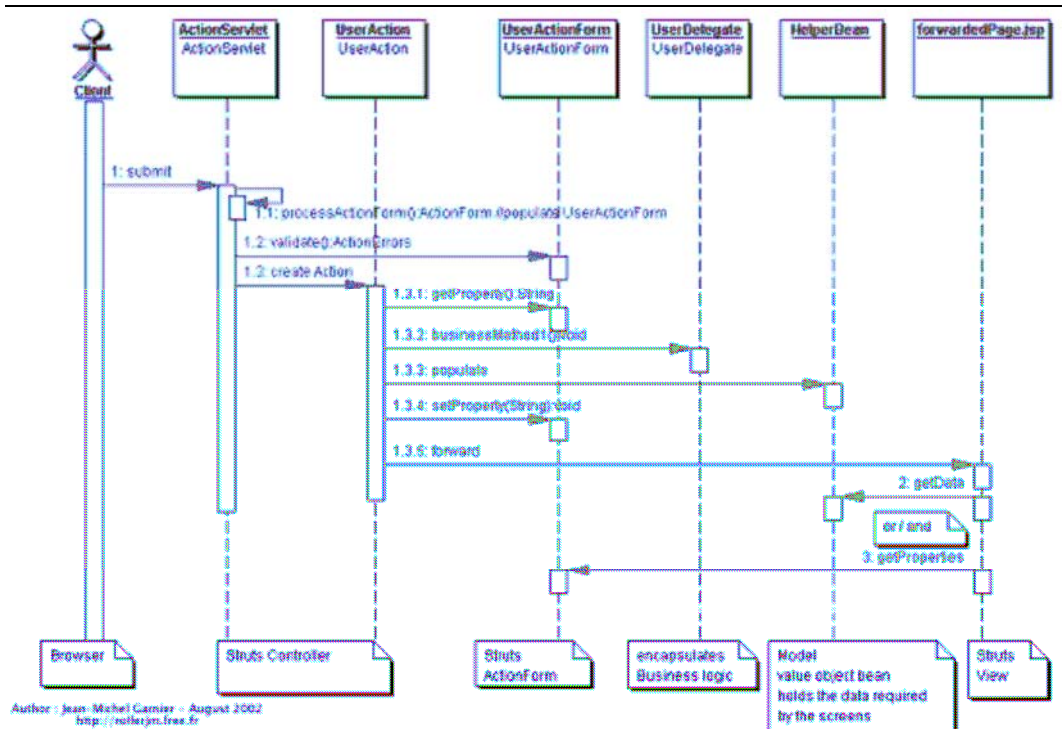


FIGURA 4.7 FLUJO DE CONTROL STRUTS

4.7 Configuración de componentes de Struts

Ahora que ya se menciona como funciona el Framework Struts es hora de mencionar como configurar nuestros componentes de Struts para iniciar lo primero que tenemos que hacer es descargar Struts este lo podemos obtener del sitio <http://struts.apache.org/>, cabe mencionar que algunos IDE's ya lo traen incluido.

Una vez descargado el Framework Struts para poder utilizarlo en nuestro ambiente de trabajo en nuestra aplicación web necesitamos colocar algunos archivos .jar dentro de nuestra carpeta lib, los cuales vienen incluidos cuando descargamos Struts. Estos los podemos encontrar dentro de la carpeta lib del zip descomprimido. Los .jar necesarios para trabajar con Struts son los siguientes:

commons-beanutils.jar
commons-digester.jar
standard.jar
struts.jar

También tenemos que tener 3 XMLs y un archivo de propiedades:

Tenemos que crear, o cambiar, algunos archivos de configuración para que nuestra aplicación con Struts pueda ser ejecutada y corra:

Web.xml el descriptor de despliegue requerido para la especificación de Java Servlets. El contenedor servlets/JSPs usa este archivo para cargar y configurar tu aplicación.

Arquitectura del framework Struts

Struts-config.xml El descriptor de despliegue del Framework. Este es usado para cargar y configurar varios componentes usados en el Framework Struts.

Build.xml Un archivo usado por la herramienta de construcción Jakarta ant para compilar y desplegar tu aplicación. El uso de ant no es un requerimiento, pero este es popularmente escogido por los desarrolladores.

Application.properties Este archivo provee recursos de mensajes para tu aplicación Struts. Como el built.xml este no es un requerimiento estricto pero es usado por muchos en aplicaciones con Struts.

A continuación veremos a detalle como configurar los principales elementos para poder correr una aplicación con Struts:

4.7.1 El archivo web.xml

El propósito y formato del descriptor de despliegue de tu aplicación web esta cubierta por la especificación SUN Servlet. Básicamente es usado para decirle al contenedor de Servlets como configurar los servlets y otros objetos de alto nivel que tu aplicación necesita.

El Framework Struts incluye dos componentes que necesita para configurarse pensando en el descriptor de despliegue de tu aplicación: el ActionServlet, y opcionalmente los tag libraries.

```
<!-- 1 -->
<?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>
<!-- 2 -->
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
<init-param>
<param-name>application</param-name>
<param-value>Application</param-value>
</init-param>
<init-param>
<param-name>config</param-name>
<param-value>/WEB-INF/conf/struts-config.xml</param-value>

<load-on-startup>2</load-on-startup>
</servlet>
<!-- 3 -->
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
<!-- 4 -->
<taglib>
<taglib-uri>/tags/struts-bean</taglib-uri>
<taglib-location>/WEB-INF/lib/struts-bean.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/tags/struts-html</taglib-uri>
<taglib-location>/WEB-INF/lib/struts-html.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/tags/struts-logic</taglib-uri>
<taglib-location>/WEB-INF/lib/struts-logic.tld</taglib-location>
```

```
</taglib>  
</web-app>
```

1.-Identifica un descriptor de despliegue de una aplicación web.-Estas dos primeras líneas identifican el archivo como un descriptor de despliegue de una aplicación web.

2.-Configurar el ActionServlet.-este bloque le dice al contenedor que cargue el ActionServlet bajo el nombre de action. Dos parámetros son pasados para el ActionServlet: application y config.

Application: El nombre de un bonche de recursos de la aplicación. Para referirnos a un archivo llamado application.properties en un paquete llamado resources, usar recurso.aplicación, es decir, en este caso recurso puede ser un subdirectorio sobre clases (o paquete en un archivo jar).

Config: Es donde colocamos la ruta context-relative(contexto-relativo) para el recurso XML que contiene nuestra información de configuración, el valor por default es /WEB-INF/struts-config.xml.

Al final de este bloque <load-on-startup>, da al action servlet un suplemento con el contenedor. Estableciendo este en 2 permite a otros servlets cargar primero si es necesario.

Solo un ActionServlet o subclase de ActionServlet puede ser cargado por una aplicación. El ActionServlet es diseñado para compartir recursos con otros componentes en la aplicación.

3.-Identificar Struts request.- Este bloque dice al contenedor como continuar con alguna petición de archivos que hacen juego con el patrón *.do para el action servlet. Este es el ActionServlet que nosotros configuramos en el bloque 2. las peticiones para los archivos que no hacen juego con este patrón no son manejados por Struts.

4.-Configurar Tag Libraries.- Aquí es donde se configuran los tag Libraries que nuestra aplicación usara. Los tres core Struts taglibs (bean, html, y logic).

4.7.2 El Struts configuration

El archivo de configuración de Struts (struts-config.xml) es usado para cargar algunos componentes críticos del framework. Juntos todos estos objetos hacen la configuración de Struts. El Struts configuration y Struts ActionServlet trabajan juntos para crear la capa de control de la aplicación.

El Struts configuration sabe que fields (campos) están en tus forms(formularios).este sabe donde tu JSP puede ser encontrado. Este sabe acerca de todos los actions que tu aplicación ejecuta y exactamente que recursos de cada uno de estos necesitan tus actions.

Este puede parecer demasiada información para coleccionar en un lugar. Y esto es. Pero para guardar toda esta implementación de detalles juntos, muchos desarrolladores encuentran que en su aplicación son mucho mas fáciles para crear y mantener.

Todos los componentes en Struts configuration están en objetos Java. El objeto ActionForm sabe acerca de los campos y forms. El objeto ActionForward sabe donde

encontrar tus JSPs. El objeto ActionMapping sabe que forms y forwards son usados con cada comando que tu aplicación entiende.

Una simple aplicación podría crear toda esta información de objetos en un método de inicialización y entonces establecer cada objeto con el valor por default necesario. Por ejemplo:

```
ActionForwards globals = new ActionForwards();
ActionForward logoff = new ActionForward();
logoff.setName("logoff");
logoff.setPath("/Logoff.do");
globals.addForward (logoff);
ActionForward logon = new ActionForward();
logoff.setName("logon");
logoff.setPath("/Logon.do");
globals.addForward (logon);
```

Pero en la practica los métodos de inicialización rápidamente suelen requerir mantenimiento lo cual es un problema para solventar esto.

Pon estas características juntas y tu realmente no necesitas clases Java. Tu solo necesitas un documento que describa como instanciar una clase Java como un objeto totalmente funcionando.

Por supuesto, los Frameworks como Struts no son los únicos que tienen este problema. El contenedor de servlets necesita hacer las mismas cosas por la misma razón. Los desarrolladores tienen que decirle al contenedor que servlets y otros objetos son necesarios para su aplicación. En vez de tener que escribir una clase Java y enchufarlos dentro del contenedor, los desarrolladores SUN escogieron un documento XML en reemplazo de esto. El contenedor lee el documento XML y usa este para instanciar y configurar cualquier servlet necesario en una aplicación.

El archivo struts-configuration es en Struts como el descriptor de despliegue que esta en tu contenedor. El controlador de Struts lee el archivo de configuración y usa este para crear y configurar cualquier objeto que el framework necesita.

A continuación se muestra el esqueleto de struts-config.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<struts-config>

  <!-- 1.-Form Beans Configuration -->
  <form-beans>
    <form-bean name="datosForm" type="controller.DatosForm" />
  </form-beans>

  <!-- 2.-Action Mappings Configuration -->
  <action-mappings>
    <action path="/datos"
      type="controller.DatosAction"
      name="datosForm"
      scope="request" />
  </action-mappings>
</struts-config>
```

```

        validate="true"
        input="/formularios/datosForm.jsp">
        <forward name="resumen" path="/formularios/datosCuentaForm.jsp" />
    </action>
</action-mappings>

<!-- 3.-Global Forwards Configuration -->
<global-forwards>
    <forward name="datosForm" path="/formularios/datosForm.jsp" />
    <forward name="inicio" path="/formularios/index.jsp" />
</global-forwards>

<!-- 4.-Message Resources Configuration -->
<message-resources parameter="recursos.ApplicationResources" />

</struts-config>

```

1.-Form Beans Configuration.- En este tag es donde le especificamos el nombre de nuestra ActionForm que maneja los datos del formulario en el atributo type especificamos el nombre del ActionForm que estaremos utilizando y en el atributo name le ponemos un alias a nuestro form ya que este lo utilizaremos para mapearlo a nuestro action-mapping.

2.-Action Mappings Configuration.- Aquí es donde vamos a mapear nuestros componentes para que Struts pueda hacer bien su trabajo como controlador el atributo path nos especifica la ruta en la cual vamos a dirigir el formulario, es decir, es el nombre el cual va a ser identificado por el action del formulario. El atributo type nos especifica el Action el cual va a ser utilizado para manejar los datos del form y buscar el jsp correcto para continuar con el flujo de la información. En el atributo name vamos a colocar el alias de nuestro form y es utilizado para especificar el ActionForm bean que estará interactuando con el Action. El atributo scope nos especifica que ámbito vamos a utilizar ya sea request o session, el atributo validate nos especifica si vamos a utilizar validaciones para los campos del formulario se debe de poner el valor true para que esta opción sea habilitada. El atributo input es donde especificamos el nombre del formulario el cual enviara los datos y por ultimo en el tag <forward> es donde especificamos a donde sera enviados los datos del formulario según lo especificado en nuestro bean Action.

3.-Global Forwards.- En este tag es donde hacemos uso de los URI, es decir, cuando utilizamos un link en un formulario le especificamos el path a donde queremos que la pagina nos envíe, este path es identificado en el struts-config en el tag <forward>, el atributo name nos especifica el alias de el path a donde el link nos enviara.

Arquitectura del framework Struts

4.-Message Resources.- Aquí es donde se especifica donde utilizaremos los parametros de mensajes de error, o mensajes de información, en el atributo parameter le especificamos el nombre de nuestro archivo que tendrá esos recursos que regularmente es nombrado *ApplicationResources.properties*.

A continuación se describirá como realizar una pequeña aplicación con Struts, este ejemplo, es realizado con la ayuda del IDE NetBeans.

DESPLEGANDO UNA APLICACIÓN WEB CON STRUTS

Para construir una aplicación web con Struts los desarrolladores definirán los hiperlinks que ellos necesitan en ActionForwards, el HTML form que ellos necesitan en ActionForms y que custom actions del lado de servidor que ellos necesitan en Action classes.

Los desarrolladores quien necesitan acceso a EJB o base de datos JDBC pueden hacerse por un camino de objetos Action. En este camino la pagina de presentación no necesita interactuar con la capa del modelo.

El objeto Action de Struts, colectara cualquier dato que la vista puede necesitar y entonces adelantara este a la pagina de presentación. Struts provee JSP tags library para usar con paginas JSP que simplifican escribir formularios HTML y acceder a otros datos que un Action puede adelantar como se muestra en la figura 4.8:

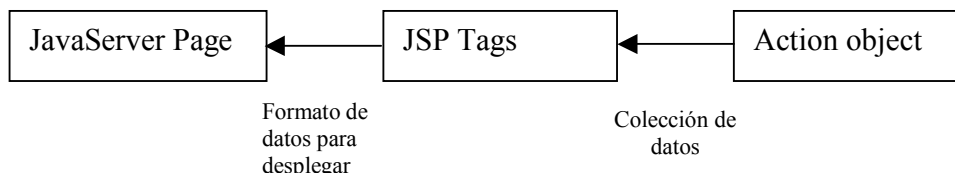


FIGURA 4.8 DESPLEGANDO UNA APLICACIÓN CON STRUTS

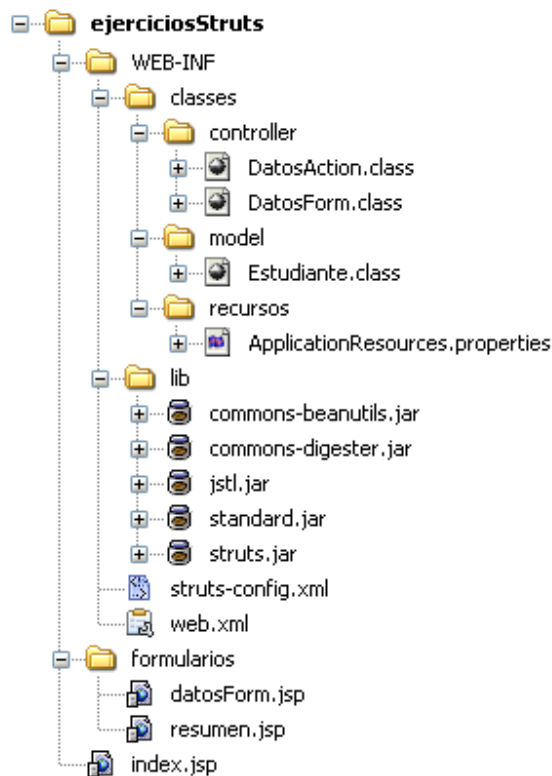
4.8 Struts Aplicación

La aplicación consta de las siguientes partes:

Arquitectura del framework Struts

File	Description
index.jsp	Archivo de entrada a nuestra aplicación
datosForm.jsp	Formulario de nuestra aplicación
resumen.jsp	Reporte de datos capturados
DatosAction.java	Clase que se encarga de procesar peticiones, y es una clase controlador.
DatosForm.java	Bean que es llenado con los datos del formulario. Es parte de la vista.
Estudiante.java	Clase del modelo que conceptualiza un estudiante.
ApplicationResources.properties	Archivo en el cual se definen los nombres de las etiquetas.
struts-config.xml	Archivo de configuración de struts, para esta aplicación.
web.xml	Descriptor de despliegue de la aplicación.

Nuestra aplicación web tendrá la siguiente apariencia:

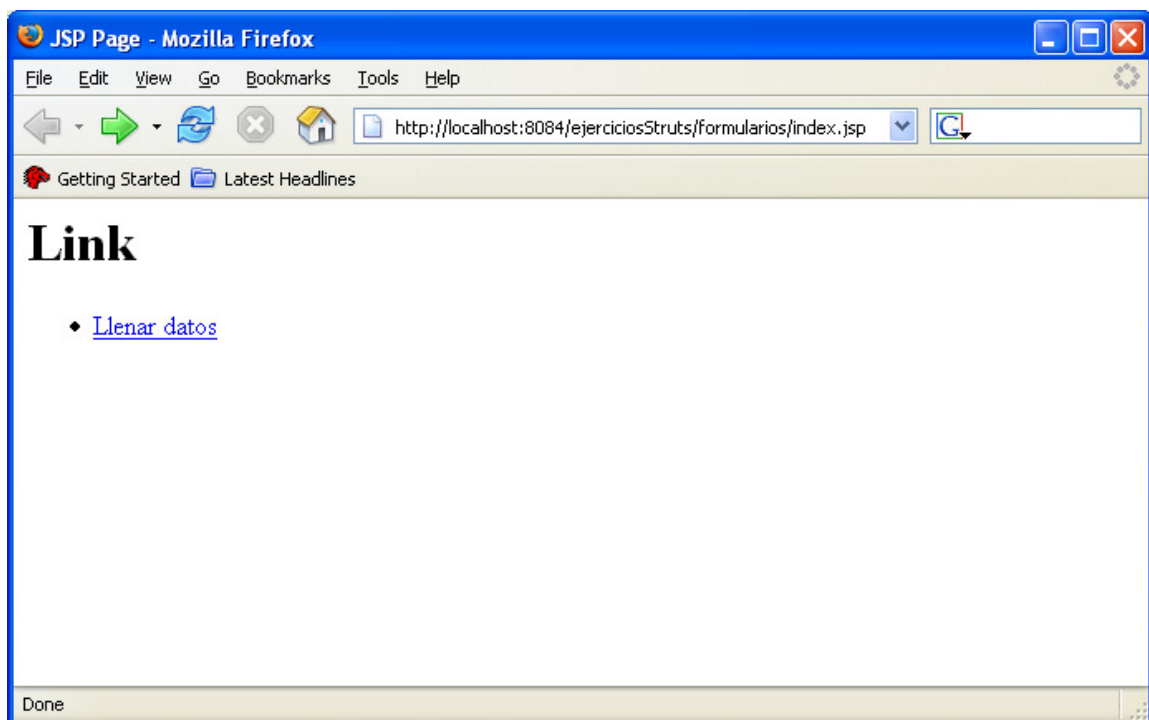


La aplicación que utilizan struts, es necesario que incluyan los siguientes archivos jar: commons-beanutils.jar, commons-digester.jar, standard.jar, struts.jar, los cuales vienen incluidos cuando descargamos struts.

Arquitectura del framework Struts

index.jsp

```
<%@taglib prefix="html" uri="http://jakarta.apache.org/struts/tags-html" %>
<html>
  <head><title>Bienvenidos</title></head>
  <body>
    <h1>Link</h1>
    <ul>
      <li><html:link forward="datosForm">Llenar datos</html:link></li>
    </ul>
  </body>
</html>
```

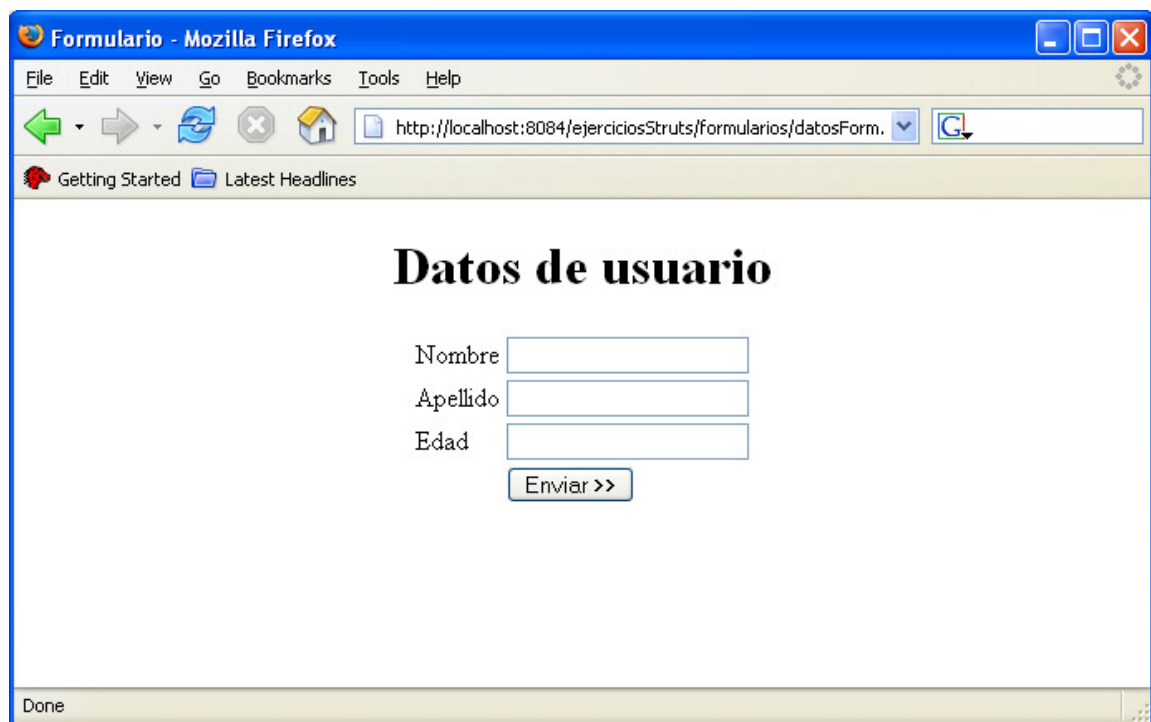


Arquitectura del framework Struts

El formulario de datos se presenta a continuación, además de que observamos el uso de etiquetas personalizadas para escribir los campos de texto, y el botón de submit.

datosForm.jsp

```
<%@taglib prefix="html" uri="http://jakarta.apache.org/struts/tags-html"%>
<html>
  <head><title>Formulario</title></head>
  <body>
    <center>
      <html:errors />
      <h1>Datos de usuario</h1>
      <html:form action="/datos" >
        <table>
          <tr><td>Nombre</td><td><html:text property="nombre" /></td></tr>
          <tr><td>Apellido</td><td><html:text property="apellido" /></td></tr>
          <tr><td>Edad</td><td><html:text property="edad" /></td></tr>
          <tr><td>&nbsp;</td><td><html:submit value="Enviar >>" /></td></tr>
        </table>
      </html:form>
    </center>
  </body>
</html>
```

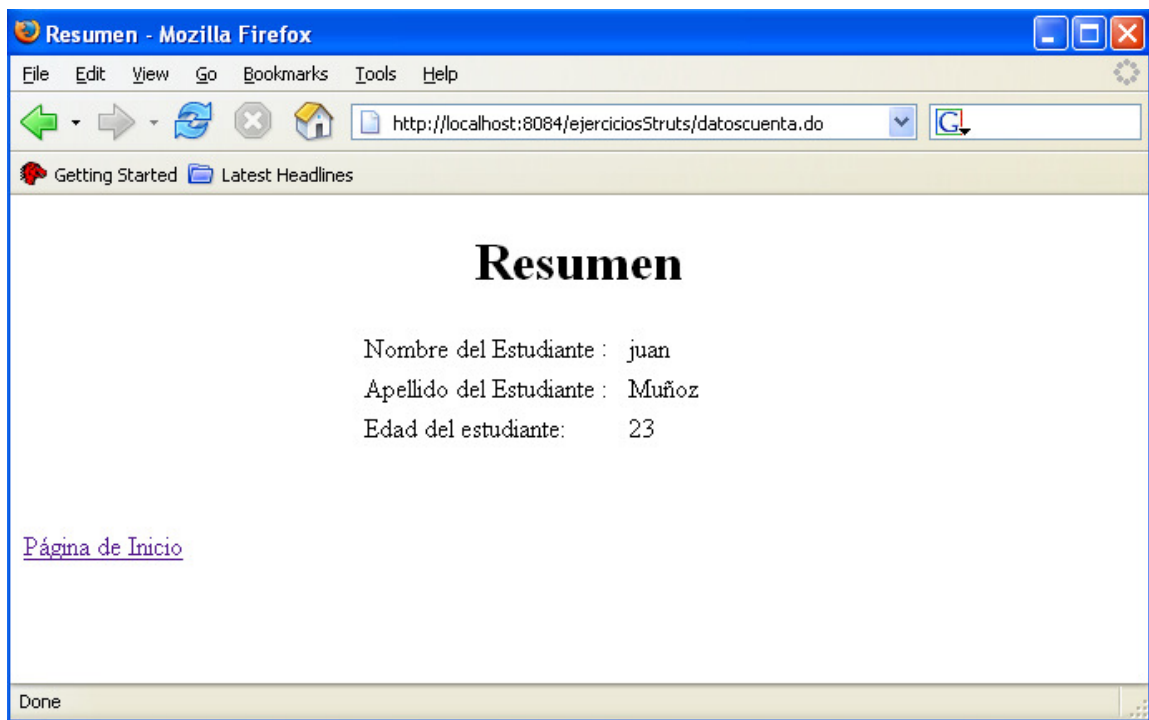


resumen.jsp

```
<%@taglib prefix="bean" uri="http://jakarta.apache.org/struts/tags-bean" %>
```

Arquitectura del framework Struts

```
<%@taglib prefix="html" uri="http://jakarta.apache.org/struts/tags-html" %>
<html>
  <head><title>Resumen</title> </head>
  <body>
    <center>
      <h1>Resumen</h1>
      <table>
        <tr>
          <td><bean:message key="label.resumen.nombre" /></td>
          <td>${sessionScope.estudiante.nombre}</td>
        </tr>
        <tr>
          <td><bean:message key="label.resumen.apellido" /></td>
          <td>${sessionScope.estudiante.apellido}</td>
        </tr>
        <tr>
          <td><bean:message key="label.resumen.edad" /></td>
          <td>${sessionScope.estudiante.edad}</td>
        </tr>
      </table>
      <center>
        <html:link forward="inicio" >Página de Inicio</html:link>
      </center>
    </body>
  </html>
```



Los datos que son enviados del formulario a nuestra aplicación, son capturados por un bean que hereda de la clase ActionForm. Este bean debe tener métodos set y get por cada campo que se desea capturar del formulario.

DatosForm.java

```
package controller;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionMapping;
import javax.servlet.http.HttpServletRequest;

public class DatosForm extends ActionForm{
    private String nombre;
    private String apellido;
    private String edad;

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
    public String getEdad() {
        return edad;
    }
    public void setEdad(String edad) {
        this.edad = edad;
    }
    public void reset(ActionMapping mapping, HttpServletRequest request){
        this.edad = null;
        this.nombre = null;
        this.apellido = null;
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request){
        ActionErrors ae = new ActionErrors();
        boolean nombreEntered = false;
        if( nombre != null && nombre.length() > 0)
            nombreEntered = true;
        else{
            ae.add("nombre", new ActionError("error.datosForm.nombre"));
        }
        return ae;
    }
}
```

La clase DatosAction hereda de Action, y es encargada de invocar a las clases de modelo de nuestra aplicación.

DatosAction.java


```
package controller;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.Action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import model.Estudiante;

public class DatosAction extends Action{

    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response ) throws Exception
    {
        DatosForm datosForm = (DatosForm)form;
        Estudiante estudiante = new Estudiante( datosForm.getNombre(),
            datosForm.getApellido(),
            datosForm.getEdad());
        request.getSession().setAttribute("estudiante", estudiante);

        return mapping.findForward("resumen");
    }
}
```

Estudiante.java

```
package model;

public class Estudiante {
    private String nombre;
    private String apellido;
    private String edad;

    public Estudiante();
    public Estudiante(String nombre, String apellido, String edad){
        this.setNombre(nombre);
        this.setApellido(apellido);
        this.setEdad(edad);
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
    public String getEdad() {
        return edad;
    }
    public void setEdad(String edad) {
        this.edad = edad;
    }
}
```

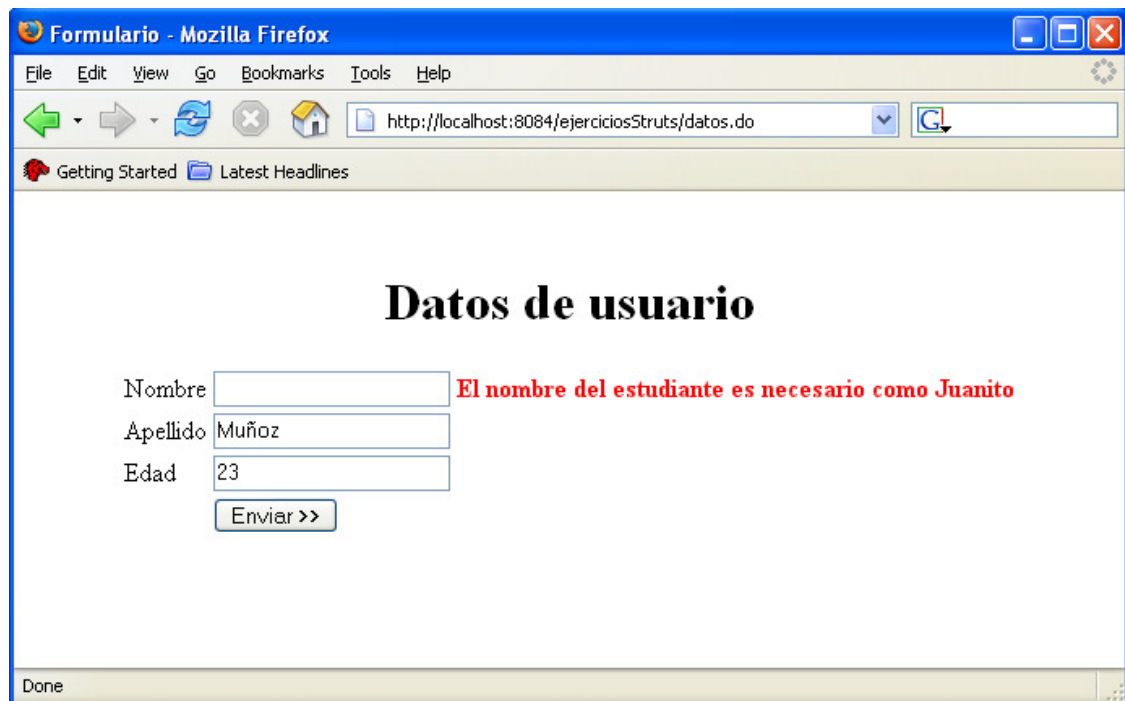
Arquitectura del framework Struts

En el archivo `ApplicationResources.properties` es en donde se dan valores a las etiquetas de los formularios o de cualquier otra parte, de modo que si se quiere cambiar el nombre de una etiqueta solo tenga que modificarse en este archivo.

ApplicationResources.properties

```
# Error Messages
error.datosForm.nombre=<font color="red"><b>El nombre es necesario como Juanito</b></font>
```

```
#Label Resumen
label.resumen.nombre=Nombre del Estudiante :
label.resumen.apellido=Apellido del Estudiante :
label.resumen.edad=Edad del estudiante :
```



El archivo `struts-config.xml` sirve para configurar el struts framework, es aquí donde declaramos nuestras clase `ActionForm`, `Action` y nuestro archivo `properties`. En este archivo es el que contiene las rutas relativas de nuestra aplicación, de modo que en los `Action` no tengamos que codificar direcciones de recursos relativas o absolutas.

struts-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>

    <!-- Form Beans Configuration -->
    <form-beans>
        <form-bean name="datosForm" type="controller.DatosForm" />
    </form-beans>

    <!-- Action Mappings Configuration -->
    <action-mappings>
        <action path="/datos"
            type="controller.DatosAction"
            name="datosForm"
            scope="request"
            validate="true"
            input="/formularios/datosForm.jsp">
            <forward name="resumen" path="/formularios/resumen.jsp" />
        </action>
    </action-mappings>

    <!-- Global Forwards Configuration -->
    <global-forwards>
        <forward name="datosForm" path="/formularios/datosForm.jsp" />
        <forward name="inicio" path="/index.jsp" />
    </global-forwards>

    <!-- Message Resources Configuration -->
    <message-resources parameter="recursos.ApplicationResources" />

</struts-config>
```

En el archivo `web.xml` se configura cual va ser el único servlet que se va a levantar, que es el `ActionServlet`.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <servlet>
    <servlet-name>Controller</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Controller</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

</web-app>
```

DESARROLLO DE UN CASO PRÁCTICO

5.1 Descripción del problema

Sistema de ventas online (cds, dvds y artículos de artistas y grupos musicales)

El objetivo de desarrollar esta aplicación web es entender el funcionamiento del framework Struts y aprovechar sus características por lo que se consideró desarrollar algo que regularmente se maneja en la vida real como son las transacciones de ventas de artículos por Internet para así observar las ventajas que el framework Struts nos da en el desarrollo como son la separación de los componentes en capas (MVC), manejo de validaciones mediante archivos de configuración, la reducción de tiempos de desarrollo y la facilitación de mantenimiento de nuestra aplicación apoyándonos en archivos de configuración del framework Struts.

El sistema de ventas online es un sistema que permite al cliente(usuario) hacer consultas y compras de discos, dvds y artículos musicales en línea sin la necesidad de ir a las tiendas de discos. Se desea que el sistema de ventas online sea accesible para todo público en Internet.

El sistema presenta en su pantalla principal un mensaje de bienvenida describiendo los servicios ofrecidos, junto con la opción de registrarte por primera vez o si ya estas registrado con solo introducir tu login y el password podrás realizar compras.

Una vez validado el usuario puede tener acceso a las siguientes actividades:

Hacer consultas específicas para CDs, DVDs o artículos de grupos y artistas favoritos. En la parte del menú se puede seleccionar el artículo que se desea comprar para hacer búsquedas específicas dependiendo el género musical o el artículo deseado como playeras ó gorras.

Colocar varios artículos ha comprar en el carrito virtual y seleccionar la cantidad de artículos deseados.

Una vez ya seleccionados todos los artículos se puede continuar con el proceso de compra introduciendo los datos de tarjeta de crédito valida.

Los artículos serán posteriormente enviados al cliente tomando los datos que se obtuvieron en el registro.

Es necesario estar previamente registrado con datos correctos de dirección para que al momento de hacer compras los artículos lleguen a la dirección que el cliente registro, así como introducir datos de tarjeta de crédito valida para poder realizar la transacción.

5.2 Modelo de casos de uso

ACTORES

Usuario

Actor:	Usuario
Casos de Uso:	Validar usuario, Registrar usuario, Registrar tarjeta, Agregar artículo al carrito, Consultar información, Ofrecer servicio, Registrar compra.
Tipo:	Primario
Descripción:	Es el actor principal y representa a cualquier persona que desee utilizar el sistema de ventas online.

Base de Datos registros

Actor:	Base de Datos de registros
Casos de Uso:	Validar usuario, Registrar usuario, Registrar tarjeta, Registrar compra.
Tipo:	Secundario
Descripción:	Es un actor secundario y representa a la base de datos donde se guarda toda la información que requiere el sistema.

CASOS DE USO

Validar Usuario

Caso de Uso:	Validar Usuario
Actores:	Usuario, base de datos registros
Tipo:	Inclusión
Propósito:	Validar a un usuario ya registrado para que pueda hacer uso del sistema de ventas online.
Resumen:	Este caso de uso es iniciado por el usuario, valida al usuario mediante un login y un password al ser validado con su respectivo registro de usuario para así poder utilizar el sistema y poder realizar compras.
Precondiciones:	Si el usuario aun no se ha registrado, se requerirá ejecutar el caso de uso <i>Registrar usuario</i> subflujo Crear registro usuario.
Flujo Principal:	Se presenta al usuario la pantalla principal (P-1) el usuario puede seleccionar entre las siguientes opciones: "Registrarse", "Login" y "cerrar sesión". Si la actividad seleccionada es Registrar por primera vez, se ejecuta el caso de uso " Registrar usuario " subflujo Crear registro usuario (S-1).

Desarrollo de un caso práctico

	<p>Si la actividad seleccionada es "Login" se valida el registro de usuario mediante un login y un password insertados por el usuario colocado en la parte del menú (M-1).</p> <p>Una vez validado el usuario (E-1) se continua con el caso de uso Ofrecer servicio, si se selecciono cerrar sesión se saldrá del sistema.</p>
Subflujo:	Ninguno.
Excepciones:	E-1 no hubo validación. El login/password no se valido correctamente. Se solicita al usuario validarse nuevamente.

Ofrecer Servicio

Caso de uso:	Ofrecer servicio
Actores:	Usuario
Tipo:	Inclusión
Propósito:	Ofrecer los diversos servicios a un usuario ya registrado para el uso del sistema de ventas Online.
Resumen:	Este caso de uso es iniciado por el usuario. Tiene opciones para poder utilizar los servicios del sistema.
Predicciones:	Se requiere la validación correcta del usuario.
Flujo principal:	<p>Se presenta al usuario la pantalla de servicios (P-2). El usuario puede seleccionar entre las siguientes actividades: "Consultar información", "registrar compras", "Obtener registro" y cerrar sesión.</p> <p>Si la actividad seleccionada es consultar información, se continua con el caso de uso "Consultar información", subflujo <i>consultar</i> (S-1, S-2, S-3).</p> <p>Si la actividad seleccionada fue "registrar compra" se continua con el caso de uso Registrar compra.</p> <p>Si la actividad seleccionada es obtener registro, se continua con el caso de uso Registrar usuario, subflujo Crear registro usuario (S-1).</p> <p>Si la actividad seleccionada es cerrar sesión se saldrá del sesión.</p>
Subflujo:	Ninguno.
Excepciones:	Ninguno.

Desarrollo de un caso práctico

Registrar Usuario	
Caso de uso:	Registrar usuario
Actores:	Usuario, base de datos registros.
Tipo:	Básico.
Propósito:	Permitir al usuario registrarse en el sistema para su uso posterior.
Resumen:	Este caso de uso es iniciado por el usuario, ofrece funcionalidad para crear el registro de usuario con el sistema.
Precondiciones:	Todos los subflujos, con excepción de crear registro usuario (S-1), requiere ejecutar inicialmente el caso de uso <i>Validar usuario</i> .
Flujo principal	Se ejecuta el caso de uso <i>Validar usuario</i> . Dependiendo de las opciones seleccionadas por el usuario se continuara con los diversos subflujos de este caso de uso.
Subflujos:	<p>S-1 Crear registro usuario Se presenta al usuario la pantalla crear registro usuario (P-3). Esta pantalla contiene información del registro que debe ser llenado por el usuario, lo cual incluye nombre, apellido paterno, apellido materno, sexo, fecha de nacimiento, código postal, dirección, ciudad, estado, teléfono, E-MAIL.</p> <p>S-2 Crear login/password Se presenta al usuario una pantalla para generar el login y password (P-4), esta pantalla contiene información de registro que debe ser llenada por el usuario para ingresar el login y password y una entrada adicional de repetir password para asegurarse de su corrección. El login y el password serán usados por el sistema para validar al usuario.</p> <p>El usuario puede seleccionar la opción registrar para generar un nuevo registro de usuario.(E1-E2-E3-E4-E5).</p> <p>Si no se ha presionado registrar la información será perdida.</p> <p>Si la actividad seleccionada es "cerrar sesión" se saldrá de la sesión.</p>
Excepciones:	<p>E-1 información incompleta: falta llenar información en el registro de usuario. Se vuelve a solicitar al usuario que complete el registro.</p> <p>E-2 Registro ya existe: si ya existe un registro bajo ese mismo login se solicitara al usuario que cambie o que salga del caso de uso.</p> <p>E-4 login incorrecto: se solicita al usuario que ingrese un login valido.</p>

Desarrollo de un caso práctico

	E-5 la contraseña no se valido correctamente.
--	---

Registrar Tarjeta

Caso de uso:	Registrar tarjeta
Actores:	Usuario, base de datos registro.
Tipo:	Extensión.
Propósito:	Permitir a un usuario registrar una tarjeta de crédito con el sistema para poder pagar los artículos seleccionados.
Resumen:	El caso de uso es iniciado por el usuario. Ofrece funcionalidad para crear el registro de tarjeta usuario para poder pagar los artículos directamente con el sistema.
Precondiciones:	El usuario ya debe haberse registrado mediante la activación del caso de uso <i>Registrar usuario</i> .
Flujo principal:	Se validan los datos de la tarjeta y se registra la tarjeta dependiendo del usuario que este registrado.
Subflujos:	Si el usuario selecciona registrar se mostrara una pagina con las condiciones de envío, Si la actividad seleccionada es cerrar sesión se saldrá de la sesión y se perderá la información.
Excepciones:	E-1 información incompleta: Falta llenar información indispensable para completar registro de tarjeta. Se vuelve a pedir al usuario que complete el registro de tarjeta.

Consultar información

Caso de uso:	Consultar información.
Actores:	Usuario, base de datos registros.
Tipo:	Básico
Propósito:	Permitir a un usuario consultar información con el sistema de ventas.
Resumen.	Este caso de uso es iniciado por el usuario. Ofrece funcionalidad para consultar información especifica de CDs o DVDs por genero musical y artículos específicos de grupos musicales.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso <i>Validar usuario</i> .
Flujo principal:	Se ejecuta el caso de uso validar usuario. Dependiendo de las opciones seleccionadas por el usuario se continuará con los diversos subflujos de este caso de uso.
Subflujos:	S-1 Consultar CD`s Se desplegara la pantalla consultas de cd (P-5), el usuario puede seleccionar entre las siguientes categorías: Blues, clásica, pop y rock. S-2 Consultar DVDs

Desarrollo de un caso práctico

	<p>Se desplegara la pantalla consultas de dvd (P-6) el usuario puede seleccionar entre las siguientes categorías: Blues, clásica, pop y rock.</p> <p>S-3 Consultar Artículos</p> <p>Se desplegara la pantalla consultas de artículos (P-7), el usuario puede seleccionar entre las siguientes categorías: Playeras ó gorras.</p> <p>S-4 Selección de artículo ha comprar.</p> <p>Se desplegara la pantalla (P-8) con las características del artículo seleccionada para posteriormente seleccionarse para agregar al carrito de compras y finalmente ser comprado</p>
Excepciones	Ninguna.

Agregar artículo al carrito

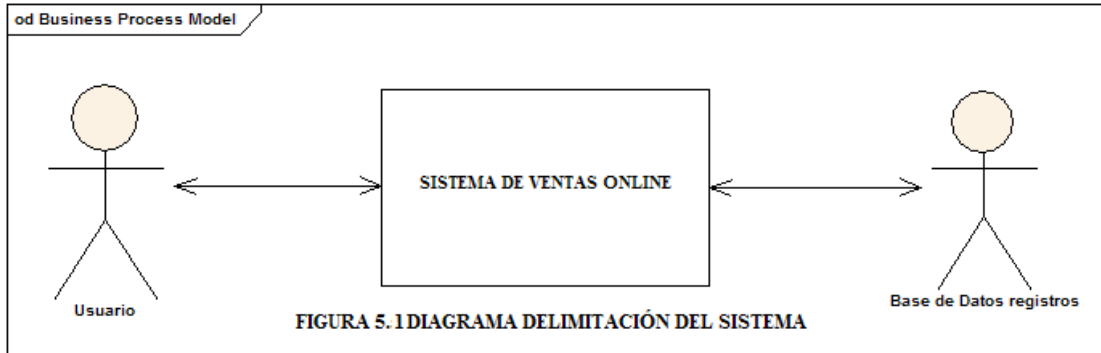
Caso uso:	Agregar artículo al carrito
Actores:	Usuario, Base de datos registros
Tipo:	Básico
Propósito:	Permitir al usuario agregar artículos para posteriormente ser comprados.
Resumen:	Este caso de uso es iniciado por el usuario. Ofrece funcionalidad para agregar la cantidad deseada de varios artículos (P-9) para posteriormente ser comprados.
Precondiciones:	Se debe haber ejecutado posteriormente el caso de uso Validar Usuario, Consultar información subflujo Selección de artículo ha comprar (S-4).
Flujo principal:	Se ejecuta el caso de uso validar usuario. Dependiendo de las opciones seleccionadas por el usuario se continuará con los diversos subflujos de este caso de uso.
Subflujos	<p>S-1 Carrito compras</p> <p>Se desplegara la pantalla de carrito de compras (P-8), el usuario puede seleccionar la cantidad (E-1) que desea de el artículo seleccionado para ser agregado al carrito de compras.</p> <p>S-2 Artículos agregados</p> <p>Se desplegara la pantalla de información de artículos agregados al carrito (P-9), donde el usuario podra observar la cantidad de artículos en su carrito de compras con la información del precio total que lleva para efectuar su compra.</p>
Excepciones:	<p>E-1 Seleccionar un valor correcto.</p> <p>Se tiene que seleccionar al menos una cantidad valida de compra del artículo seleccionado.</p>

Registrar Compra

Caso de uso:	Registrar compra
Actores:	Usuario, Base de datos registros
Tipo:	Extensión
Propósito:	Permitir al usuario finalizar compras.
Resumen:	Este caso de uso es iniciado por el usuario. Ofrece funcionalidad para finalizar compra.
Precondiciones:	Se debe haber ejecutado posteriormente el caso de uso <i>Validar usuario, Agregar artículo al carrito, Registrar Tarjeta..</i>
Flujo principal:	Se ejecuta el caso de uso validar usuario. Dependiendo de las opciones seleccionadas por el usuario , se continuara con el subflujo de este caso de uso.
Subflujos:	S-1 Despliegue pagina de registro exitoso. Se desplegara una pagina de registro exitoso y condiciones de envió.
Excepciones:	Ninguna.

Las pantallas del sistemas serán presentadas en la ultima sección.

5.3 Diagrama delimitación del sistema



En la figura 5.1 se muestra un panorama general de la aplicación lo podemos de ver de una manera muy simple la aplicación requiere que el usuario mande peticiones a la aplicación, esta atenderá las peticiones y si así lo requiere se comunicara con la base de datos y procesara los resultados para que sean presentados al usuario.

Como se puede observar solo contamos con 2 actores principales el usuario y la base de datos de registros, los cuales estarán interactuando directamente con la aplicación y se llevaran a cabo los casos de uso ya descritos anteriormente.

5.4 Diagrama casos de uso

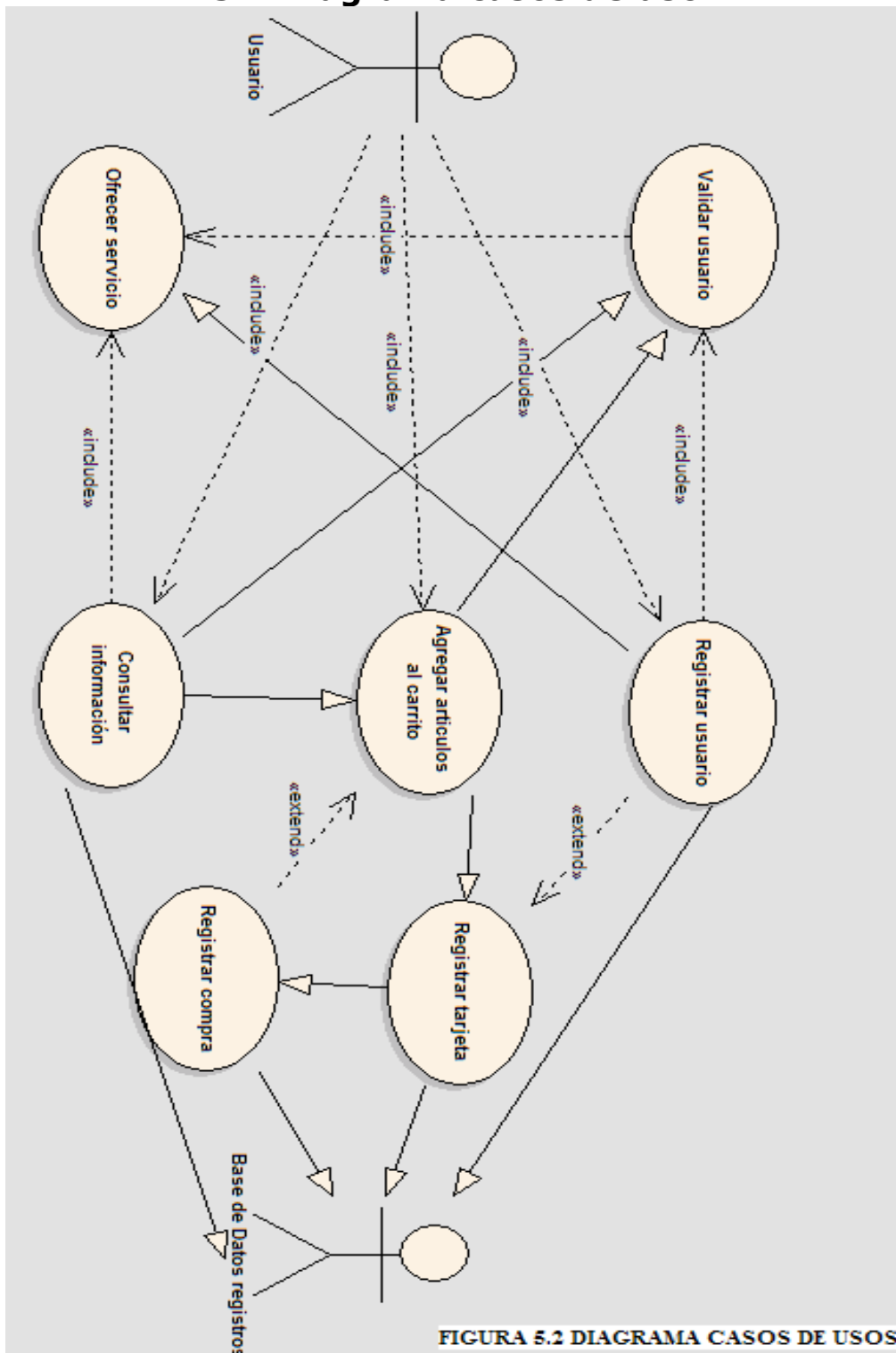


FIGURA 5.2 DIAGRAMA CASOS DE USOS

En la figura 5.2 se ilustra el diagrama de nuestros casos de uso anteriormente descritos y como los actores (usuario y base de datos registros) interactúan con estos casos de uso.

5.5 Diagramas de secuencia

En los siguientes diagramas de secuencia se muestra cada secuencia que sigue la aplicación según el caso de uso.

Validar Login

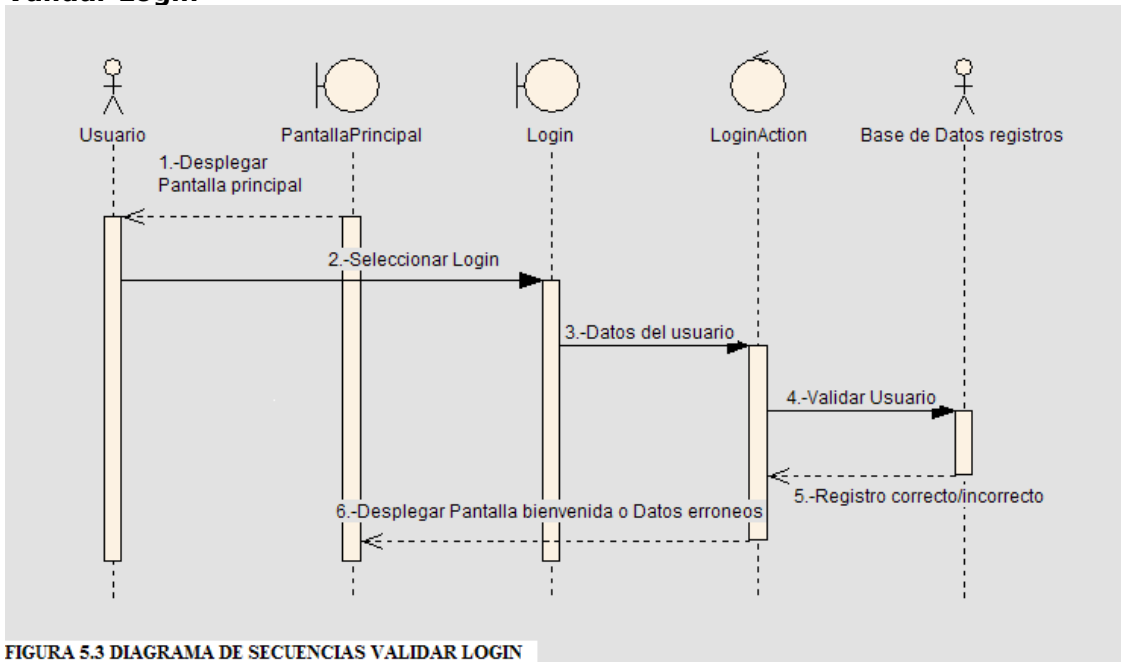


FIGURA 5.3 DIAGRAMA DE SECUENCIAS VALIDAR LOGIN

En la figura 5.3 el diagrama de validar login, muestra que en un principio la aplicación muestra la pantalla principal (1.-Desplegar pantalla principal), la cual es la que el usuario observara en primera instancia, el usuario al seleccionar login (2.-Seleccionar login) se desplegara el formulario donde se pide el login y la contraseña donde será necesario ingresar los datos que serán enviados (3.-Datos del usuario) a el LoginAction (4.-Validar usuario) que es el encargado de verificar en la base de datos si es un usuario valido o enviar si los datos son erróneos (5.-Registro correcto / incorrecto) para así desplegar la pantalla de bienvenida si los datos son correctos(6.-Desplegar pantalla de bienvenida).

Registrar usuario

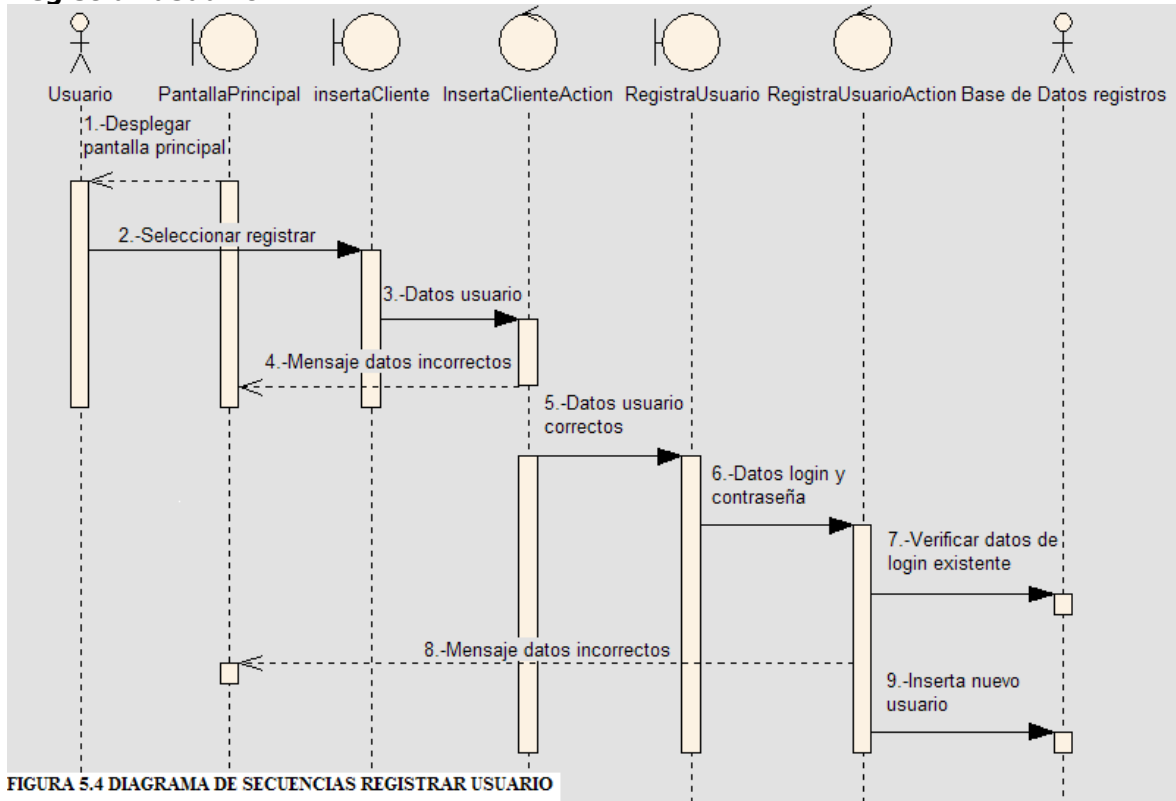


FIGURA 5.4 DIAGRAMA DE SECUENCIAS REGISTRAR USUARIO

La figura 5.4 el diagrama de registrar usuario nos muestra que cuando el usuario observa la pantalla principal (1.-Desplegar pantalla principal) al ser seleccionada la opción de registrar (2.-Seleccionar registrar) es mostrada la pantalla inserta cliente que contiene el formulario de datos personales requeridos para llevar a cabo el registro estos datos son enviados a (3.-Datos usuario) InsertaClienteAction que es el encargado de verificar si los datos son erróneos se mandaran los mensajes de datos erróneos (4.-Mensaje datos incorrectos) o si son correctos (5.-Datos usuario correctos) este despliega la página de registrarUsuario que es la que contiene el formulario para la creación de login y contraseña, una vez que el usuario lleno los datos estos son enviados (6.-Datos login y contraseña) a RegistraUsuarioAction que es el encargado de verificar (7.-Verificar datos login existente) en la base de datos si el login ya existe y mostrar los mensajes de error (8.-Mensaje de datos incorrecto) y si no es así ingresar el registro del nuevo usuario a la base de datos (9.-Inserta nuevo usuario).

Consultar información

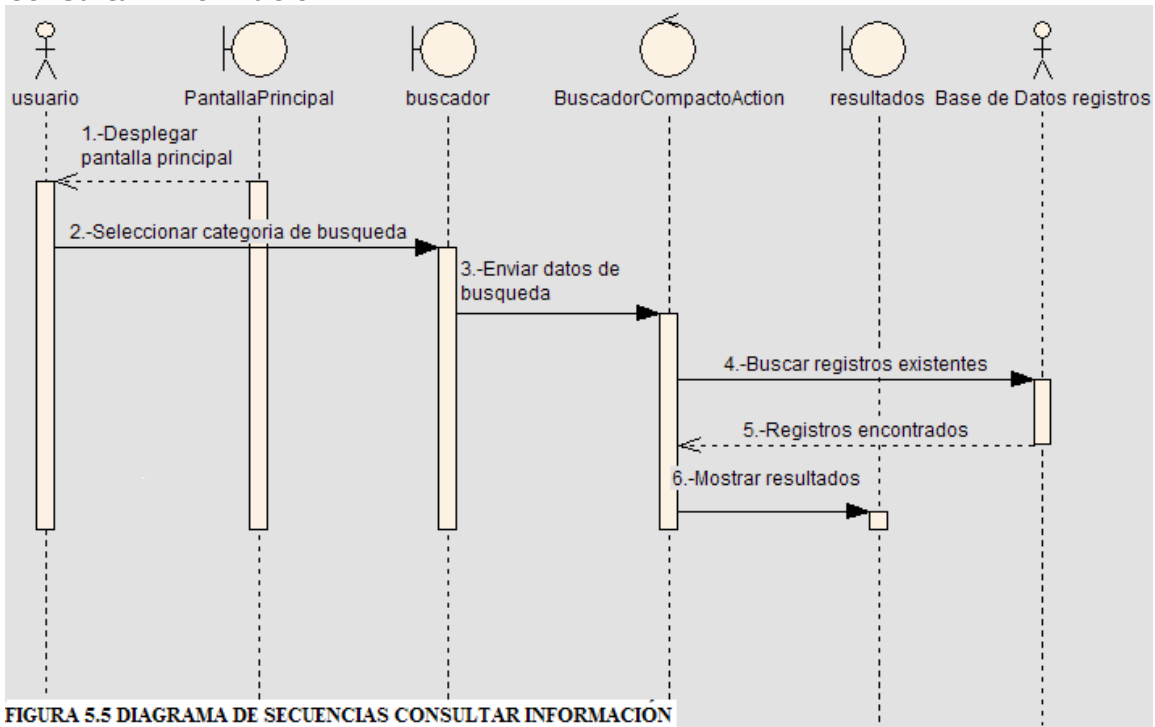


FIGURA 5.5 DIAGRAMA DE SECUENCIAS CONSULTAR INFORMACION

La figura 5.5 el diagrama de consultar información nos muestra que cuando el usuario observa la pantalla principal (1.-Desplegar pantalla principal) al ser seleccionada la opción de búsqueda (2.-Seleccionar categoría de búsqueda) es desplegada la pantalla de buscador la cual será encargada de enviar los datos de la búsqueda (3.-Enviar datos de búsqueda) a BuscadorCompactoAction que es el encargado de buscar si hay registros existentes (4.-Buscar registros existentes) de la base de datos y una vez encontrados (5.-Registros encontrados) mostrar los resultados en la pantalla de resultados (6.-Mostrar resultados)

Agregar artículos al carrito

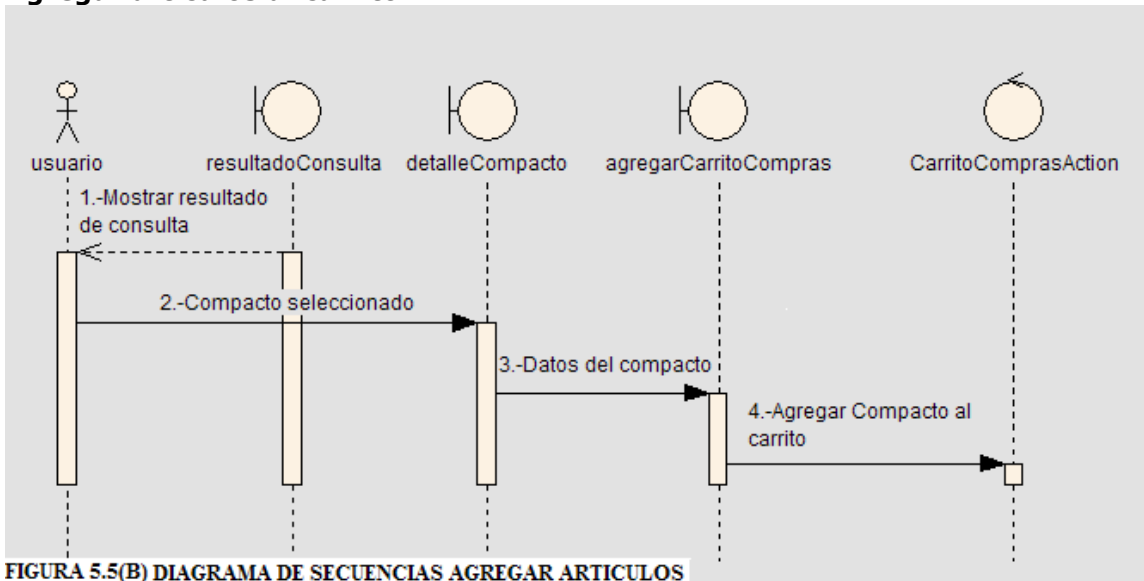


FIGURA 5.5(B) DIAGRAMA DE SECUENCIAS AGREGAR ARTICULOS

La figura 5.5(B) el diagrama de agregar artículos al carrito nos muestra que cuando el usuario una vez hecha una búsqueda este observa los resultados encontrados (1.-Mostrar resultados de consulta) y selecciona el compacto (2.-Compacto seleccionado) que desea agregar al carrito para ser comprado, los datos del compacto son desplegados en la pantalla detalleCompacto donde se muestra la cantidad y la portada del disco a detalle y también contiene el botón de agregar al carrito que nos envía los datos del compacto (3.-Datos del compacto) a la pantalla de agregarCarritoCompras que es donde se muestran los datos de los artículos almacenados en el carrito así como el total a pagar y la cantidad de artículos deseados y finalmente aquí se podrá agregar el artículo seleccionado encargándose de almacenar (4.-Agregar compacto al carrito) estos artículos el CarritoComprasAction.

Registrar compra

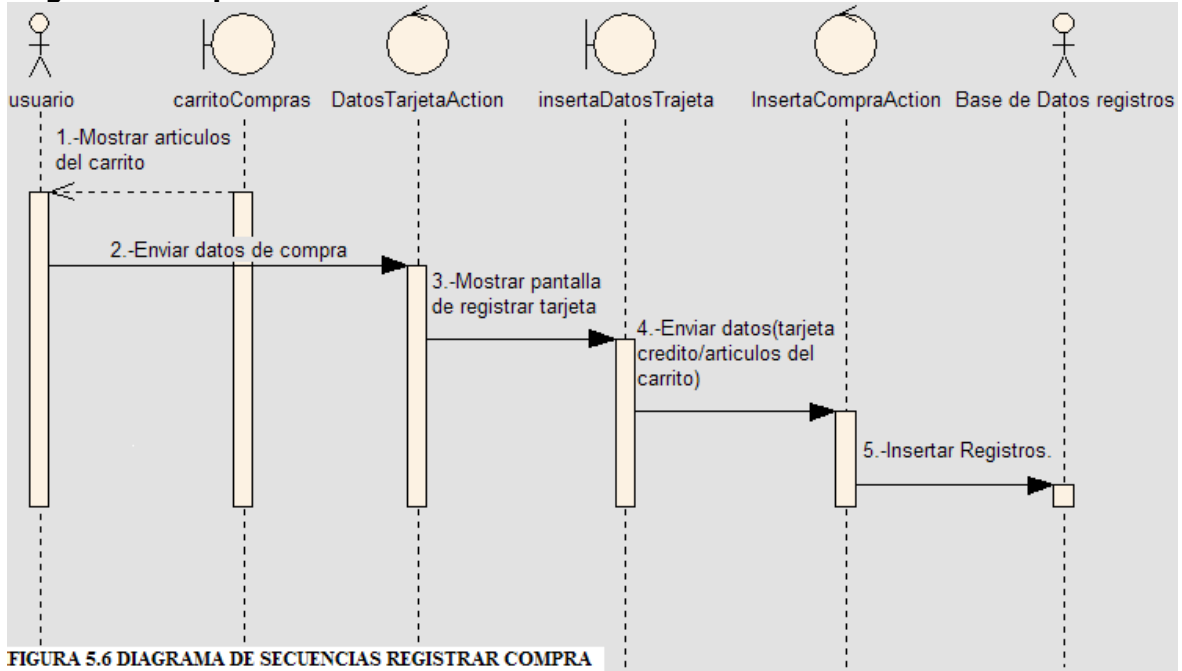


FIGURA 5.6 DIAGRAMA DE SECUENCIAS REGISTRAR COMPRA

La figura 5.6 el diagrama de registrar compra nos muestra que cuando el usuario esta en la pantalla de carritoCompras la cual contiene los artículos que el usuario ha agregado al carrito de compras (1.-Mostrar artículos del carrito) , el usuario elige realizar la compra (2.-Enviar datos de compra) y los datos de la compra son enviados a DatosTarjetaAction el cual se encargara de mostrar la pantalla que contiene el formulario de registro de tarjeta de crédito (3.-Mostrar pantalla de registrar tarjeta) necesario para realizar la compra una vez llenado el formulario de datos de tarjeta de crédito, los datos son enviados (4.-Enviar datos) a InsertaCompraAction que es el encargado de ingresar los datos de la compra a la base de datos (5.-Insertar registro) y mostrar la pagina de registro exitoso.

5.6 Script SQL base de datos svonline.

A continuación se muestra el script SQL que se utilizó para generar la base de datos de la aplicación:

```
CREATE TABLE genero (id_genero serial not null , nombre varchar(40), PRIMARY KEY (id_genero))
```

```
CREATE TABLE estado (id_estado serial not null, nombre varchar(60), PRIMARY KEY (id_estado))
```

```
CREATE TABLE cliente (id_cliente serial not null, nombre varchar(60), apellido_paterno varchar(60), apellido_materno varchar(60), id_genero serial , fecha_nacimiento date, cp varchar(10), telefono numeric, mail varchar(60), direccion varchar(60), ciudad varchar(60), id_estado serial, PRIMARY KEY (id_cliente), constraint FK_GENERO foreign key (id_genero) references genero, constraint FK_ESTADO foreign key (id_estado) references estado)
```

```
CREATE TABLE rol (id_role numeric not null, rol varchar(60), PRIMARY KEY (id_role))
```

```
CREATE TABLE usuario (login varchar(40) not null, password varchar(30), id_cliente serial, id_role serial, PRIMARY KEY (login), constraint FK_CLIENTE foreign key (id_cliente) references cliente, constraint FK_ROL foreign key (id_role) references rol)
```

```
CREATE TABLE categoria (id_categoria serial not null, nombre varchar(60), PRIMARY KEY (id_categoria))
```

```
CREATE TABLE artista (id_artista serial not null, nombre varchar(60), grupo varchar(60), id_producto serial, PRIMARY KEY (id_artista), constraint FK_PRODUCTO FOREIGN KEY (id_producto) references producto)
```

```
CREATE TABLE tipo (id_tipo serial not null, nombre varchar(40), PRIMARY KEY (id_tipo))
```

```
CREATE TABLE producto (id_producto serial not null, nombre varchar(60), descripcion varchar(120), existencias numeric, imagen varchar(40), precio_unitario numeric, id_categoria serial, id_tipo serial, id_artista serial, PRIMARY KEY (id_producto), constraint FK_CATEGORIA FOREIGN KEY (id_categoria) references categoria, constraint FK_TIPO FOREIGN KEY (id_tipo) references tipo, constraint FK_ARTISTA FOREIGN KEY (id_artista) references artista)
```

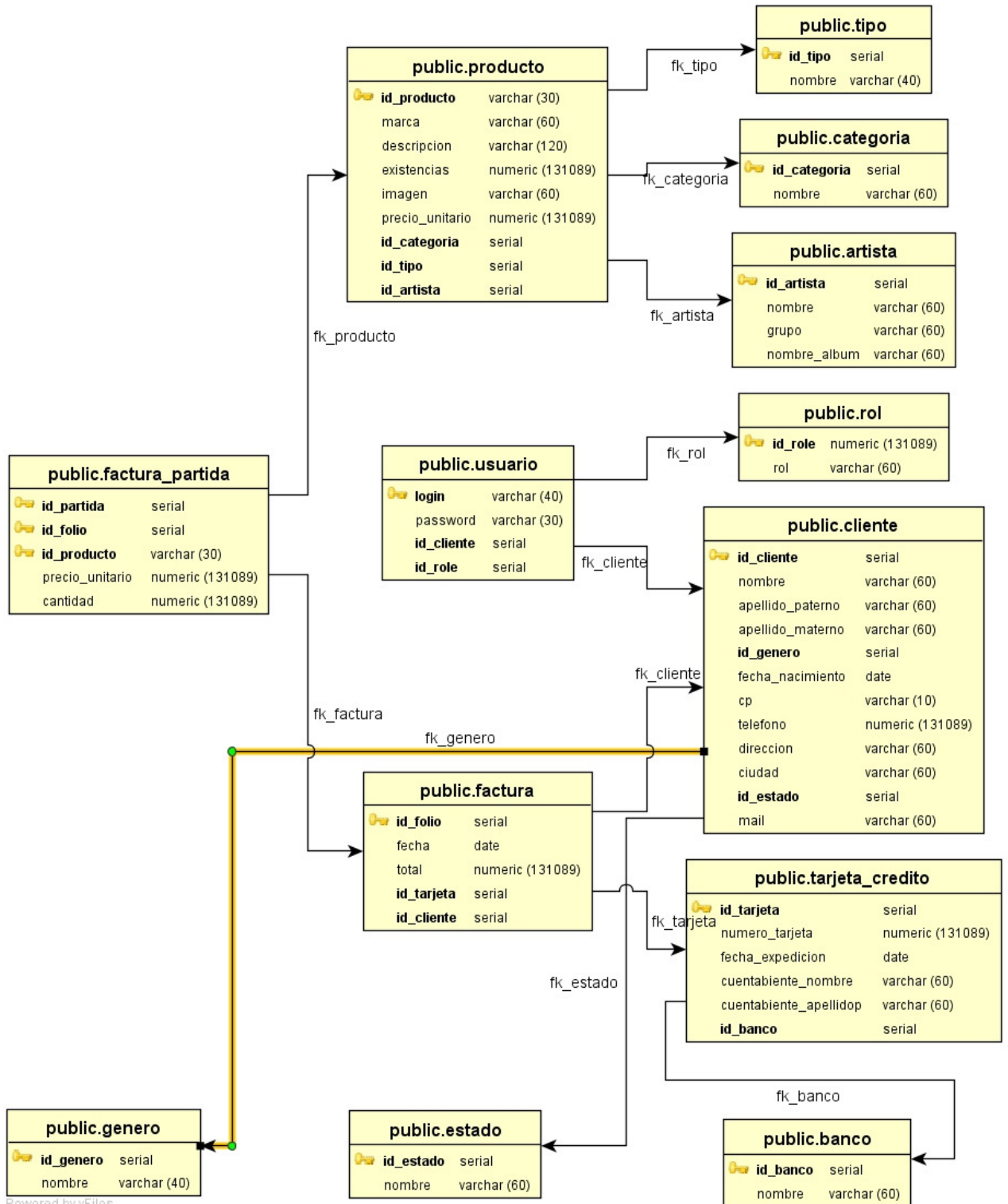
CREATE TABLE banco (*id_banco serial not null, nombre varchar(60), PRIMARY KEY (id_banco)*)

CREATE TABLE tarjeta_credito (*id_tarjeta serial not null, numero_tarjeta numeric, fecha_expedicion date, cuentabiente_nombre varchar(60), cuentabiente_apellido varchar(60), id_banco serial, PRIMARY KEY (id_tarjeta), constraint FK_BANCO FOREIGN KEY (id_banco) references banco*)

CREATE TABLE factura (*id_folio serial not null, fecha date, total numeric, id_tarjeta serial, id_cliente serial, PRIMARY KEY (id_folio), constraint FK_TARJETA FOREIGN KEY (id_tarjeta) references tarjeta_credito, constraint FK_CLIENTE FOREIGN KEY (id_cliente) references cliente*)

CREATE TABLE factura_partida (*id_partida serial, id_folio serial, id_producto varchar(30), precio_unitario numeric, cantidad numeric, PRIMARY KEY (id_partida, id_folio, id_producto), constraint FK_FACTURA FOREIGN KEY (id_folio) references factura, constraint FK_PRODUCTO FOREIGN KEY (id_producto) references producto*)

5.7 Modelo Entidad Relación



Powered by yFiles

5.8 PANTALLAS

index.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Desarrollo de Aplicaciones Web utilizando el framework Struts</title>
    <link rel="stylesheet" type="text/css" href="styles/site.css" />
  </head>

  <body>

    <div id="title">
      DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS
    </div>

    <div id="userbar">
      <span class="user">User:</span>
      <span class="data"><%if (request.getSession().getAttribute("autenticado") == null) %>Sin sesion<%> else
      %>${sessionScope.autenticado.login}<%>%></span>
      <span class="role">Role :</span>
      <span class="data"><%if (request.getSession().getAttribute("autenticado") == null) %> <%> else
      %>usuario<%>%></span>
    </div>

    <div id="navigation">
      <ul>
        <li>
          <a href="#">CDs</a>
          <ul>
            <li><li><html:link forward="blues">Blues</html:link></li></li>
            <li><html:link forward="clasica">Clasica</html:link></li>
            <li><html:link forward="pop">Pop</html:link></li>
            <li><html:link forward="rock">Rock</html:link></li>
          </ul>
        </li>
        <li>
          <a href="#">DVDs</a>
          <ul>
            <li><li><html:link forward="bluesdvd">Blues</html:link></li></li>
            <li><html:link forward="clasicadvd">Clasica</html:link></li>
            <li><html:link forward="popdvd">Pop</html:link></li>
            <li><html:link forward="rockdvd">Rock</html:link></li>
          </ul>
        </li>
        <li>
          <a href="#">Articulos</a>
          <ul>
            <li><li><html:link forward="playera">Playeras</html:link></li></li>
            <li><html:link forward="gorras">Gorras</html:link></li>
          </ul>
        </li>
        <li><html:link forward="loginLink">Login</html:link></li>
        <li><html:link forward="newcliente">Registrate</html:link></li>
      </ul>
    </div>

    <div id="main">
      <center><h1><i>SVONLINE MUSICA A TU ALCANCE</i></h1></center>

```

Desarrollo de un caso práctico

```
<br/>
<center>
  
  
  
  
  
</center>
<br/><br/>
<center><html:link forward="loginLink" styleClass="button-link">Login</html:link>
<html:link forward="newcliente" styleClass="button-link">Registrate</html:link></center>

</div>

<div id="pie">
  All the trademarks is Copyright @
</div>

</body>
</html>
```

Como se puede observar se hizo uso de hojas de estilo para la parte del menu y todo el marco de presentación de la aplicación este se importa con la siguiente línea de código `<link rel="stylesheet" type="text/css" href="styles/site.css" />` y también se utilizan algunos scripts:

```
<span class="user">User:</span>
<span class="data"><%if (request.getSession().getAttribute("autenticado") == null) {%>Sin sesion<%> else
{%>${sessionScope.autenticado.login}<%}%></span>
```

se verifica si el usuario ya está autenticado y si es así se muestra el nombre del usuario.

Pantalla principal

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

User: Sin sesion Role :

- CDs
- DVDs
- Articulos
- Login
- Registrate

SVONLINE MUSICA A TU ALCANCE



Login Registrate

P1

Es la primer pantalla que el usuario observara al visitar la aplicación.

Pantalla ofreser servicio

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

Cerrar Sesion User: metamorfis Role : usuario

- CDs
 - Blues
 - Clasica
 - Pop
 - Rock
- DVDs
- Articulos
- Registrate

BIENVENIDO A SVONLINE AHORA PODRAS BUS



YA PUEDES REALIZAR COMPRAS



CDs, DVDs y Articulos

P2

Una vez que la aplicación verifico que el login y contraseña son correctos sera presentada esta pantalla al usuario, y podra utilizar todos os servicios que ofrece la aplicación.

insertaCliente.jsp

```
<div id="main">
    <html:form action="/insertacliente">
        <center><h1>AGREGAR CLIENTE</h1></center>
        <table align="center" width="70%" border="0">
            <tr>
                <td>Nombre:</td>
                <td><html:text property="nombre"/></td>
                <td><html:errors property="nombre"/></td>
            </tr>
            <tr>
                <td>Apellido Paterno:</td>
                <td><html:text property="apellidoPaterno"/></td>
                <td><html:errors property="apellido"/></td>
            </tr>
            <tr>
                <td>Apellido Materno:</td>
                <td><html:text property="apellidoMaterno"/></td>
            </tr>
            <tr>
                <td>Sexo:</td>
                <td><html:select property="sexo">
                    <c:forEach var="sexo" items="{genero}">
                        <html:option value="{sexo.nombre}"/>
                    </c:forEach>
                </html:select>
            </td>
            </tr>
            <tr>
                <td>Fecha de Nacimiento:</td>
                <td><html:text property="fechaNacimiento"/></td>
                <td><html:errors property="fechaNacimiento"/></td>
            </tr>
            <tr>
                <td>Codigo Postal:</td>
                <td><html:text property="codigoPostal"/></td>
            </tr>
            <tr>
                <td>Dirección:</td>
                <td><html:text property="direccion"/></td>
            </tr>
            <tr>
                <td>Ciudad:</td>
                <td><html:text property="ciudad"/></td>
            </tr>
        </table>
    </form>
</div>
```

```
</tr>

<tr>
  <td>Estado:</td>
  <td><html:text property="estado"/><br></td>
</tr>

<tr>
  <td>Telefono:</td>
  <td><html:text property="telefono"/><br></td>
  <td><html:errors property="telefono"/></td>
</tr>

<tr>
  <td>E-mail:</td>
  <td><html:text property="mail"/><br></td>
  <td><html:errors property="mail"/></td>
</tr>

<tr><td colspan="2" align="center"><html:submit value="continuar"/></td></tr>

</table>
</html:form>

</div>
```

El código anterior es el encargado de presentar el formulario de registro de nuevo usuario en el cual hacemos uso de etiquetas de Struts para generar el formulario

```
<tr>
  <td>Nombre:</td>
  <td><html:text property="nombre"/></td>
  <td><html:errors property="nombre"/></td>
</tr>
```

podemos observar que utilizamos la etiqueta `html:text` para generar el campo de texto y la etiqueta `html:errors` para el uso de las validaciones de la pantalla registro nuevo usuario, si los datos son correctos se manda llamar a la clase Action respectiva para que valide el flujo que debe seguir la página.

Pantalla registro nuevo usuario

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

User: metamorfis Role : usuario

- CDs
- DVDs
- Artículos
- Regístrate

AGREGAR CLIENTE

Nombre: ♦ Es necesario ingresar un nombre

Apellido Paterno: ♦ Es necesario ingresar el apellido

Apellido Materno:

Sexo:

Fecha de Nacimiento: ♦ El formato es :dd/mm/yy

Código Postal:

Dirección:

Ciudad:

Estado:

Telefono: ♦ Deben de ser caractere numericos

E-mail:

P3

Esta pantalla será presentada una vez que el usuario elija la opción regístrate, para crear una nueva cuenta y poder hacer uso de la aplicación y si los datos son correctos se presentara la pantalla de creación de login y password.

Pantalla creación login y password.

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

User: metamorfis Role : usuari

- CDs
- DVDs
- Artículos
- Regístrate

FIN DEL REGISTRO

Login: ♦ Ya existe un usuario con este login

Password:

Confirma password: ♦ No coinciden los passwords

P4

Desarrollo de un caso práctico

Esta pantalla es el ultimo paso para el registro de un nuevo usuario, una vez que se ingresen datos de login y password correctos se terminara el registro y se podra ingresar el nuevo login y contraseña para hacer uso de la aplicación.

ValidLogin.java

```
public class ValidLogin {
    private Connection conn = null;

    /** Creates a new instance of ValidLogin */
    public ValidLogin() {
    }

    public boolean isUserValid(String user, String passwd){
        String consulta = "SELECT login FROM usuario WHERE login = ? AND password = ?";

        // verificar si el conn fue proporcionado o no
        boolean isSupplied = conn != null;

        //hacemos la comparacion si me dieron la conexion se utilizara el constructor
        //que recibe un argumento y si es el caso que no creamos una conexion
        Connection cone = isSupplied ? conn : Conexion.getConexion() ;
        PreparedStatement pstmt = null;

        //por cuestiones de seguridad la declaramos como false
        boolean isValid = false;

        try{
            System.out.println("Inicio " + consulta + " " + user + " " + passwd);
            //usamos prepareStatement para precompilar consulta
            pstmt = cone.prepareStatement(consulta);
            pstmt.setString(1, user);
            pstmt.setString(2, passwd);
            ResultSet rs = pstmt.executeQuery();
            isValid = rs.next();
            System.out.println("Fin " + consulta);
        }catch(SQLException e){
            e.printStackTrace();
        }catch(Exception e){
            e.printStackTrace();
        }finally{//ojo esto es muy importante ya que si la consulta la creamos nosotros
            //debemos de serciorarnos de cerrarla o bien si me dieron la consulta
            //debemos de dejarla abierta por si la llegasemos a utilizar posteriormente.
            if(!isSupplied){
                Conexion.close(cone);
                System.out.println("Cerrando conexion creada en el metodo");
            }
        }
        return isValid;//retornamos el valor booleano true o false segun sea el caso
    }

    public boolean isLoginValid(String user){
        String consulta = "SELECT login FROM usuario WHERE login = ? ";

        // verificar si el conn fue proporcionado o no
        boolean isSupplied = conn != null;

        //hacemos la comparacion si me dieron la conexion se utilizara el constructor
        //que recibe un argumento y si es el caso que no creamos una conexion
        Connection cone = isSupplied ? conn : Conexion.getConexion() ;
        PreparedStatement pstmt = null;
```

Desarrollo de un caso práctico

```
//por cuestiones de seguridad la declaramos como false
boolean isValid = false;

try{
    System.out.println("Inicio " + consulta + " " + user);
    //usamos preparedStatement para precompilar consulta
    pstmt = cone.prepareStatement(consulta);
    pstmt.setString(1, user);
    ResultSet rs = pstmt.executeQuery();
    isValid = rs.next();
    System.out.println("Fin " + consulta);
}catch(SQLException e){
    e.printStackTrace();
}catch(Exception e){
    e.printStackTrace();
}finally{//ojo esto es muy importante ya que si la consulta la creamos nosotros
    //debemos de serciorarnos de cerrarla o bien si me dieron la consulta
    //debemos de dejarla abierta por si la llegasemos a utilizar posteriormente.
    if(!isSupplied){
        Conexion.close(cone);
        System.out.println("Cerrando conexion creada en el metodo");
    }
}
return isValid;//retornamos el valor booleano true o false segun sea el caso
}

public static void main(String arg[]){
    ValidLogin validar = new ValidLogin();
    boolean valor = validar.isUserValid("meta@meta.com", "antar99");
    System.out.println(valor);
}
}
```

Pantalla de login y contraseña

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

User: metamorfis Role : usuari

- CDs
- DVDs
- Artículos
- Regístrate

LOGIN

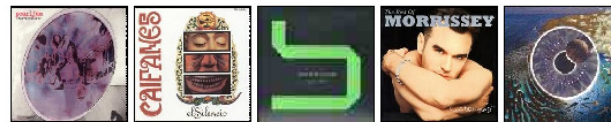
Usuario:

Contraseña:

• USUARIO O CONTRASEÑA NO VALIDOS

M-1

SVONLINE MUSICA A TU ALCANCE



Desarrollo de un caso práctico

En esta pantalla el usuario podrá ingresar sus datos para ser validado y si el login y contraseña son correctos el usuario podrá acceder a todos los servicios de la aplicación.

Pantallas de búsqueda

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

[Cerrar Sesión](#) *User: metamorfis Role : usuario*

[CDs](#)
[DVDs](#)
[Artículos](#)
[Regístrate](#)

Busador compactos Blues

Ingresar tu búsqueda:

En estas pantallas se podrán realizar las búsquedas dependiendo el tipo de producto y su categoría.


Pantalla de resultados

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

[Cerrar Sesión](#) *User: metamorfis Role : usuario*

[CDs](#)
[DVDs](#)
[Artículos](#)
[Regístrate](#)

RESULTADOS

ID	Categoría	Artista	Album	Precio	Imagen
caif01	pop	saul hernandez	el nervio del volcan	\$123	

Desarrollo de un caso práctico

En la pantalla de resultados se desplegarán todos los artículos existentes según la palabra de búsqueda, en esta pantalla se podrá seleccionar el artículo a comprar.

Pantalla artículo seleccionado

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

[Cerrar Sesión](#) User: metamorfis Role : usuario

[CDs](#)
[DVDs](#)
[Artículos](#)
[Registrate](#)

saul hernandez



ID	Precio
caif01	\$123

[COMPRAR](#)

En la pantalla de artículo seleccionado se podrá ver el detalle del artículo que se quiere comprar, antes de que se realice la compra este se enviara al carrito de compras.

CarritoComprasAction.java

```
package com.mx.svonline.controlador;

import com.mx.svonline.modelo.Buscador;
import com.mx.svonline.modelo.beans.Compacto;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForward;
/**
 *
 * @author Ing Juan M G
 * @version
 */
public class CarritoComprasAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        //Se crea un objeto de tipo CarritoComprasForm para obtener los datos
        //del artículo seleccionado.
        CarritoComprasForm carritoForm = (CarritoComprasForm)form;
```

```
//Se crea un nuevo objeto de tipo Buscador que contiene la logica para  
//buscar el producto por su id  
Buscador buscador = new Buscador();  
  
//Se obtiene el id del producto y la cantdad  
String idProducto = carritoForm.getIdProducto();  
Integer cantidad = carritoForm.getCantidad();  
  
//Se hace la busqueda del producto  
Compacto compactoBean = buscador.producto(idProducto);  
compactoBean.setCantidad(cantidad);  
  
  
//Se agrega el producto al carrito de compras  
if (idProducto != null) {  
    Set<Compacto> compacto = (Set) request.getSession().getAttribute("compactoCarrito");  
    if(compacto != null){  
        compacto.add(compactoBean);  
    }else{  
        compacto = new HashSet();  
        compacto.add(compactoBean);  
    }  
  
    int total = 0;  
    for (Compacto com : compacto) {  
        com.setPrecio(com.getPrecio() * com.getCantidad());  
        total += com.getPrecio();  
    }  
  
    request.setAttribute("total", total);  
    request.getSession().setAttribute("total", total);  
    request.getSession().setAttribute("compactoCarrito", compacto);  
}  
  
//Se envan los datos a la pagina mapeada con el nombre success  
return mapping.findForward(SUCCESS);  
  
}
```

Pantalla agregar al carrito de compras

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

Cerrar Sesión *User: metamorfis Role : usuario*

CDs

DVDs

Artículos

Registrate

AGREGAR AL CARRITO DE COMPRAS

ID	Cantante/Grupo	Album	Cantidad	Precio
caif01	saul hernandez	el nervio del volcan	2 <input style="width: 50px;" type="text"/>	\$123

En la pantalla agregar al carrito de compras el usuario una vez seleccionado el artículo que quiere comprar tiene la opción de poner cuantos artículos desea comprar y almacenarlos en el carrito.

Pantalla carrito de compras

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

Cerrar Sesión *User: metamorfis Role : usuario*

CDs

DVDs

Artículos

Registrate

CARRITO DE COMPRAS

ARTICULOS ALMACENADOS

ID	Cantante/Grupo	Album	Cantidad	Precio
CADVD01	saul hernandez	ultimo conciento nervio del volcan	2	\$1086
caif01	saul hernandez	el nervio del volcan	1	\$123
TOTAL				\$1209

En la pantalla carrito de compras se muestran los artículos almacenados en el carrito con la descripción de el precio por artículo y el total de la compra, una vez que el usuario ya no quiera agregar mas artículos podrá seleccionar realizar compra para que finalmente sean llenados los datos de tarjeta de crédito y finalizar la compra.

Pantalla datos de tarjeta de crédito.

DESARROLLO DE APLICACIONES WEB UTILIZANDO EL FRAMEWORK STRUTS

[Cerrar Sesión](#) *User: metamorfis Role : usuario*

CDs	<p>TU COMPRA SE REALIZARA CON LOS SIGUIENTES DATOS</p> <p>Datos del cliente: Nombre: JUAN Apellido paterno: munoz Apellido materno: gutierrez Telefono: 58246846</p> <p>Tu compra se enviara a la siguiente direccion: Direccion: avenue Codigo Postal: 123213</p> <p>DATOS DE LA TARJETA DE CREDITO</p> <p>Banco: <input type="text" value="BANAMEX"/></p> <p>Número de Tarjeta: <input type="text" value="23455343"/></p> <p>Fecha de Expedicion: <input type="text" value="04/03/2008"/></p> <p>Nombre Cuentabiente: <input type="text" value="juan"/></p> <p>Apellido Paterno Cuentabiente: <input type="text" value="munoz"/></p> <p style="text-align: center;"><input type="button" value="enviar"/></p>
DVDs	
Articulos	
Registrate	

En la pantalla de datos de tarjeta de crédito se presentan los datos del usuario que realizara la compra (estos datos son los datos de la sesión del usuario) y un formulario para ingresar los datos de la tarjeta de crédito, esta pantalla es el ultimo paso requerido para finalizar la compra.

CONCLUSIÓN

Las aplicaciones web han estado evolucionando constantemente y una muestra de ello es el uso constante de estas hoy en día, ya que se han vuelto indispensables para la mayoría de las empresas privadas y de gobierno, esto implicó generar una solución para reducir tiempos, esfuerzos y calidad en el desarrollo de aplicaciones web el resultado ha sido el uso de patrones de diseño y utilización de frameworks, esto se logro debido a que los desarrolladores identificaron que había tareas que eran comunes en todas las aplicaciones web y el resultado fue el desarrollo de frameworks que simplificaran estas tareas.

Se aprendió como el uso del patrón de diseño MVC es una buena alternativa para el desarrollo de aplicaciones web, es por ello que el framework Struts esta basado en dicho modelo.

El uso de frameworks ha sido una parte importante para el desarrollo rápido y confiable de aplicaciones web, el éxito de los frameworks es tal que ya existen muchos mas en el mercado para poder hacer uso de estos según las necesidades de la aplicación un ejemplo de ello son los frameworks Struts, hibernate, Spring, Java Server Faces y Tapestry con el fin de obtener los beneficios que el desarrollador desea de cada una de ellos.

Hemos aprendido como el framework Struts nos simplifica varias tareas de desarrollo como son las validaciones de formularios, login, y un excelente uso del patrón de diseño MVC que nos permite separar la lógica de negocios, el controlador y la vista mediante sus archivos de configuración lo cual nos evita generar código y por lo tanto reducir tiempos de desarrollo.

En la practica el uso de frameworks nos ayuda a mejorar la administración de la aplicación con ayuda de archivos de configuración sin necesidad de cambiar líneas de código ya que en el momento de hacer un refactor los frameworks nos facilitan el mantenimiento, además de que tienen soporte para validación de formularios, es decir, nos ayudan a reducir el tiempo requerido para el desarrollo.

El uso de frameworks en el futuro va por buen camino ya que además de que hay frameworks especializados dependiendo las necesidades de desarrollo la mayoría de estos son compatibles y se puede hacer uso de uno o mas frameworks en el desarrollo de una aplicación web por ejemplo Struts puede ser usado en conjunto con Hibernate que es un framework que se utiliza para la persistencia de datos (Mapeo de objetos relacionales) que tiene como función ligar un objeto a una base de datos relacional y dar mejor soporte al acceso y recuperación de datos y también Spring que es un framework que hace posible configurar los objetos mediante un archivo XML. Spring también provee de toda la infraestructura necesaria para proveer un framework que facilita implementar la persistencia, por lo cual es bueno aprender utilizar frameworks ya que nos facilitan el desarrollo y con Struts aprendimos el beneficio que nos da en cuanto al desarrollo de aplicaciones web.

Bibliografía

- Ted Usted, Cedric Dumoulin, George Franciscus, David Winterfeldt, **Struts in Action, Building web applications with the leading Java framework**, editorial Manning, Primera Edición, 2003.
- Bill Siggelkow, **Jakarta Struts Cook Book**, editorial O'RELLY, Primera Edición, 2005.
- James Holmes, **The Complete Reference Struts**, Editorial Mc Graw Hill, Segunda Edición, 2007.
- Bryan Basham, Katy Sierra, Bert Bates, **Head First Servlets & JSP, Passing the Sun Certified Web Component Developer Exam**, editorial O'RELLY, Primera Edición, 2004.
- Jim Keogh, J2EE, **Manual de referencia**, Editorial Mc Graw Hill, Primera Edición, 2004.
- Harvey M. Deitel, Paul J. Deitel, **Cómo Programar en Java**, Editorial Prentice Hall, Quinta Edición, 2004.
- Deepak Alur, John Crupi, Dan Malks, **Core J2EE Patterns, Best Practices and Design Strategies**, editorial Prentice Hall, Segunda Edición, 2003.
- Lynn Beighley, **Head First SQL**, editorial O'RELLY, Primera Edición, 2007.
- Ken Arnold, James Gosling, David Holmes, **The Java Programming Language**, editorial Addison Wesley, Cuarta Edición, 2004.
- Ramez A. Elmasri, Shamkant B. Navathe, **Fundamentos de Sistemas de Bases de Datos**, Editorial Addison Wesley, Tercera Edición, 2004.
- Vivek Chopra, Amit Bakore, Jon Eaves, Ben Galbraith, Sing Li, Chanoch Wiggers, **Profesional Apache Tomcat 5**, Editorial Wrox, Primera Edición, 2004.

Referencias

- <http://struts.apache.org/>
Página oficial del framework Struts.
- <http://java.sun.com>
Página oficial de Java Sun Microsystems.