Curso de desarrollo web en entorno de servidor

Pedro R. Benito da Rocha - Julio de 2012

Para contactar con el autor: ceronuevenueve@gmail.com

Índice de contenido

Prefacio

Este texto se ha escrito pensando en un curso acelerado de programación en el lado del servidor usando el lenguaje Java como referencia, pero al mismo tiempo teniendo en cuenta otros conocimientos que debe poseer el desarrollador de este tipo de aplicaciones.

El tema 1 ofrece una visión general del entorno de trabajo en el que se ejecutan las aplicaciones del servidor. Este tema también es adecuado para personas con otros perfiles que necesitan una visión global de este tipo de desarrollos.

El tema 2 continúa con la formación teórica ahondando en la generación de contenido dinámico como eje fundamental de las aplicaciones web de servidor. Las prácticas que se realizarán en este tema servirán para afianzar los contenidos materializando algunos de los conceptos vistos. Los alumnos aprenderán el significado del concepto de lógica de negocio y su situación dentro del esquema general.

El tema 3 trata de la capa de acceso a datos, de las alternativas y de su uso eficiente.

El tema 4 habla de los servicios web y de las tecnologías asociadas a esta arquitectura tan extendida en la actualidad y se intentará demostrar los beneficios que tiene respecto a su complejidad inicial.

El tema 5 realiza una introducción al procesado del lado del cliente como complemento a las tecnologías de servidor, haciendo mención especial a la tecnología AJAX para la creación de aplicaciones mas eficientes.

El tema 6 habla de la reutilización del código. Tan importante como conocer el lenguaje de programación es conocer los recursos complementarios que ofrece y cómo generar recursos propios que puedan volver a utilizarse en otros proyecto ahorrando costes.

En el tema 6 se implementa una pequeña aplicación en la que se utilizan diversos conocimientos adquiridos durante todo el curso, de manera que se tenga una visión global mediante un producto acabado.

Espero que el lector disfrute leyendo este texto y que obtenga de el no solo unos conocimientos básicos, si no también la chispa que le incite a saber mas sobre el tema e investigar por su cuenta. Al final del texto se han puesto unas referencias que pueden servir como punto de partida en este interesante viaje por el conocimiento. Animo al lector a que se embarque en tan maravillosa y gratificante aventura.

Tema 1

Introducción. Selección de arquitecturas y herramientas de programación.

Objetivos:

- Conocer las diferentes herramientas de programación y arquitecturas usadas en el desarrollo de aplicaciones web en entorno de servidor.
- Identificar los distintos tipos de arquitecturas y tecnologías usadas en aplicaciones distribuidas y aplicaciones web.
- Introducción a los contenedores de servlets.

Introducción

En la actualidad se ha consolidado el uso de aplicaciones web en todo tipo de entornos, puesto que sus ventajas son evidentes a la hora de acceder a la información desde cualquier lugar y desde cualquier dispositivo.

La tendencia cada vez mayor hacia este tipo de aplicaciones en las que los interfaces de usuario son páginas web, y el procesamiento se realiza en servidores hace que dominar los elementos de este tipo de arquitecturas sea importante para las personas que se encuentran implicadas en el desarrollo de aplicaciones modernas.

En este tema se verá una imagen general de los entornos disponibles actualmente así como una descripción de las tecnologías que hacen posible el funcionamiento de aplicaciones en entorno del servidor.

Modelos de programación y tipos de aplicaciones en entornos cliente-servidor, web y web 2.0

Los modelos de programación, así como los tipos de aplicaciones han evolucionado desde los primeros días de la informática hasta hoy. Los entornos han ido pasando a lo largo del tiempo por sucesivas fases que a continuación se indican de forma simplificada:

- Era “mainframe”. Los primeros ordenadores eran muy caros, ocupaban mucho espacio y utilizaban muchos recursos. Por lo tanto eran caros de usar y mantener y había que optimizar su uso. En esta era existía un único ordenador al que se accedía por diversos medios, pero era el terminal “tonto” (sin cpu ni memoria) el método que hizo que se popularizase su uso en las empresas. Todo el procesamiento se realizaba exclusivamente en el servidor usando aplicaciones monolíticas, y raramente se comunicaba con otros mainframes.
- Era “PC”. Con el desarrollo de los ordenadores personales la informática se descentraliza y el procesamiento de la información se realiza en el equipo cliente. El bajo coste de los equipos y el incremento de la potencia de los mismos hacen que las empresas pequeñas y medianas comiencen a informatizarse masivamente. Se pasa de un procesamiento en un único equipo a procesamiento

en el propio terminal, que ahora tiene capacidad de proceso y almacenamiento, utilizando diversas aplicaciones instaladas en el equipo.

- Trabajo en red. Las redes locales hacen que los equipos PC se interconecten entre sí creando grupos de trabajo en red, donde cada equipo comparte información con el resto. Las aplicaciones siguen estando instaladas en los clientes y los servidores se limitan a ofrecer recursos de almacenamiento e impresión.
- Internet. Con la llegada de Internet al público general comienzan a aparecer pequeñas aplicaciones que se ejecutan en servidores web para ofrecer información personalizada a los clientes. Los equipos con los que se accede a la red siguen ejecutando aplicaciones localmente y usan la red principalmente para acceder e intercambiar información. Aparecen las aplicaciones con el modelo cliente-servidor.
- Web 2.0. La llamada web 2.0, donde los contenidos dinámicos y personalizados son la principal novedad, hace que el uso de aplicaciones en el servidor crezca exponencialmente. Este tipo de aplicaciones son cada vez mas complejas y deben dar servicio a un número de usuarios cada vez mayor. Comienzan a desarrollarse aplicaciones modeladas por capas donde servidores especializados realizan funciones concretas.
- Era “post-PC”. El uso de dispositivos móviles como tabletas y smartphones que usan intensivamente las aplicaciones de la red hacen que el desarrollo de las tecnologías usadas en el servidor tengan que dar nuevas respuestas a nuevos problemas. Los lenguajes de programación deben ofrecer soluciones nuevas a estos problemas.

Como se ha visto las aplicaciones han pasado por varias etapas. Primero aplicaciones monolíticas basadas en mainframe, luego aplicaciones que se ejecutan en el cliente pero que intercambian información por la red y mas tarde aplicaciones que se ejecutan en servidores remotos y que son accedidas por clientes ligeros y navegadores web, evitando en lo posible el procesamiento de la información en el cliente.

Por lo tanto se pueden clasificar las aplicaciones en:

- Monolíticas: se ejecutan en un entorno de servidor y son accedidas mediante terminales. Las aplicaciones se encargan de todas las tareas. Son caras de mantener y presentan dificultades de escalabilidad.
- De cliente: se ejecutan exclusivamente en el equipo cliente. El equipo en el que se ejecutan no tiene porqué estar conectadas a la red. Son aplicaciones monousuario.
- Cliente-servidor: son aplicaciones instaladas en un cliente que se comunican con un servidor para intercambiar información. Tanto el cliente como el servidor procesan información, ya que es el cliente el encargado de interactuar con el usuario y de presentar la información. Las aplicaciones web y web 2.0 son casos particulares de esta arquitectura.
- Modelo de *n*-capas: las aplicaciones se modelan en capas, cada una de las cuales tiene una función específica. Un tipo muy usado es la arquitectura en 3 capas que se verá mas adelante.

- Aplicaciones distribuidas: las aplicaciones están formadas por componentes que se pueden ejecutar en distintos servidores. Usan tecnologías de integración y localización de servicios para la interconexión de componentes. Los lenguajes de programación usados para este tipo de aplicaciones proporcionan herramientas y mecanismos de abstracción para facilitar el desarrollo de aplicaciones (API) y facilitan la reutilización de código. Este tipo de aplicaciones suelen ser desarrolladas por equipos multidisciplinares donde los roles de cada integrante están bien definidos.

Introducción a los lenguajes de programación en entorno servidor

Como se ha visto anteriormente los entornos de trabajo han ido evolucionando a través del tiempo. Desde el nacimiento de las primeras aplicaciones web los lenguajes de programación y las herramientas disponibles han ido adaptándose a las nuevas necesidades de escalabilidad, seguridad e integración de nuevas tecnologías.

Atendiendo a las tecnologías empleadas en el desarrollo de aplicaciones en entorno de servidor podríamos obtener el siguiente listado:

- CGI (Common Gateway Interface): es la tecnología mas antigua para el desarrollo de aplicaciones del lado del servidor. Consiste en ejecutar un programa que procesa la información que le proporciona el servidor web, y que devuelve una respuesta que será enviada al navegador a través del propio servidor. Esta tecnología es poco eficiente puesto que requiere ejecutar un programa por cada petición de cliente, y además no contempla aspectos tan importantes como la seguridad o la escalabilidad, tarea que se deja en manos del servidor web.

  Un programa CGI puede estar escrito en cualquier lenguaje que pueda resultar en un programa ejecutable por el servidor, como por ejemplo C, perl, shell script, etc. Por lo tanto depende fuertemente de la plataforma en la que se va a ejecutar.

- Lenguajes de scripting integrados con el servidor web. Son lenguajes que se integran en las páginas web alojadas en el servidor y que son interpretadas por el mismo. El servidor web se apoya en un módulo que es capaz de ejecutar las órdenes embebidas en las páginas web y producir un resultado dinámico.

  Algunos de los lenguajes que usan esta tecnología son:

  - PHP: es un lenguaje interpretado que se integra con el servidor web como un módulo o como un programa CGI. Es un lenguaje muy usado y posee infinidad de módulos que amplían sus funcionalidades. Su sintaxis está heredada del lenguaje C y de los shell script. También tiene la posibilidad de usar programación orientada a objetos, y posee potentes funciones para integrarlo con otros productos y comunicarse en red.
  - ASP: es también un lenguaje interpretado embebido en páginas web propietario de Microsoft y que solamente funciona en plataformas Windows.

    Se basa en Visual Basic Script pero con bastantes añadidos.

La versión mas reciente se denomina ASP.Net y se basa en la tecnología de componentes .net de Microsoft, que le dota de mejores capacidades y de una estructura de componentes distribuida.

- JSP (Java Server Pages): es una tecnología que usa Java como lenguaje de programación. También usa código embebido en HTML pero se apoya en la tecnología servlet de Java EE como entorno de ejecución.

- Servidores de aplicaciones empresariales. Son máquinas que ejecutan aplicaciones o parte de ellas y que trabajan en conjunto con otros servidores para ofrecer una aplicación completa. Ofrecen servicios web o bien servicios de componentes que interactúan con otros servidores. Se integran en entornos de tecnologías de *n*-capas.
  - Servlets: usa la tecnología Java EE para ejecutar aplicaciones del lado del servidor. Proporciona ayudas al programador en forma de APIs de programación que ofrecen distintos servicios de bajo nivel para que el desarrollo de aplicaciones sea mas rápido y eficiente, al reutilizar el código ya escrito. Se basa en componentes distribuidos que formas una máquina virtual donde se ejecuta la aplicación completa.

Los lenguajes de programación mas extendidos son (en orden alfabético) ASP.net, Java y PHP.

A continuación se muestran varios ejemplos del programa “Hola mundo” en distintos lenguajes.

```
#include <stdio.h>

int main()
{
     printf(“Hola mundo”);

     return 0;
}
```

Programa “Hola mundo” para un CGI escrito en lenguaje C.

Este programa habría que compilarlo y generar un ejecutable. Este CGI se podría haber escrito en cualquier otro lenguaje, como shell script o perl. Los CGI reciben los datos mediante variables de entorno y entrada estándar (stdin) y ofrecen los resultados mediante salida estándar (stdout).

```
<%@ Page Language="VB" %>

<script runat="server">
Sub Page_Load(Sender As Object, E As EventArgs)
     HelloWorld.Text = "Hola mundo."
End Sub
</script>

<html>
```

```
<head>
<title>Hola mundo (en ASP.NET)</title>
</head>
<body>
<asp:label id="HelloWorld" runat="server" />
</body>
</html>
```

Programa "Hola Mundo" escrito en ASP.net.

El fichero que lo contiene tendrá extensión aspx (por ejemplo `hola.aspx`).

```
<html>
<head>
     <title>Hola mundo (en PHP)</title>
</head>
<body>
<?php echo "Hola mundo"; ?>
</body>
</html>
```

Programa "Hola mundo" escrito en PHP.

El fichero que contiene este código tendrá extensión php, por ejemplo `hola.php`.

```
<html>
<head>
     <title>Hola mundo (en JSP)</title>
<%!
String message = "Hola mundo";
%>
</head>
<body>
<%= message%>
</body>
</html>
```

Programa "Hola mundo" escrito en Java usando JSP.

El fichero que contiene este programa tendrá extensión jsp, por ejemplo `hola.jsp`, y necesitará formar parte de una aplicación web. En los servidores de aplicaciones Java a esta acción se le llama "desplegar la aplicación". Los ficheros jsp se compilan "al vuelo" cada vez que cambian.

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

public class HelloWorld extends HttpServlet {

   public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {
```

```
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<html>");
        out.println("<head><title>Hola mundo (servlet)</title></head>");
        out.println("<body>");
        out.println("Hola mundo");
        out.println("</body></html>");
    } // doGet

} // HelloWorld
```

Programa "Hola mundo" escrito como un servlet en Java.

Este programa se deberá guardar en un fichero con extensión .java, como por ejemplo `hola.java`. Luego se compila y se despliega dentro de una aplicación.

Introducción a los lenguajes de marcas

Los lenguajes de marcas son importantes en el desarrollo de aplicaciones web, ya que hay muchos procesos en los que se utilizan este tipo de lenguajes.

Existen diferentes tipos de lenguajes de marcas dependiendo de la función a la que están orientados:

- Presentación: son lenguajes que se orientan a formatear el contenido para presentarlo de determinada forma.
- Procedimientos: está orientado a la presentación y edición del contenido del documento.
- Descriptivo: son lenguajes que se utilizan para describir la información contenida en el documento.

Los principales lenguajes de marcas usados en las tecnologías web son descriptivos, ya que estructuran la información e indican el tipo de la misma.

- SGML: es un lenguaje orientado a la organización y etiquetado de documentos. Es la base para otros lenguajes de marcas.
- HTML: es una extensión de SGML. Sirve para describir la estructura y el contenido formal del texto, así como incluir en el documento otros elementos como por ejemplo imágenes y scripts. Se utiliza fundamentalmente en páginas web.
- XML: es una extensión de SGML. Se creó por el W3 para estructurar documentos grandes. Se pretende que sea (y es) un estándar para intercambiar información de forma estructurada entre varias plataformas.
- XHTML: es una implementación del lenguaje HTML expresado como XML válido.

El uso de los distintos lenguajes de marcas se irá viendo a través del resto de temas, ya que se emplean en diferentes lugares con diferentes usos.

Aplicaciones distribuidas basadas en componentes

Las aplicaciones distribuidas son aquellas cuyas partes se ejecutan en máquinas separadas pero conectadas por red. Algunos ejemplos de modelos distribuidos son: cliente-servidor, arquitectura de tres capas y arquitecturas multinivel.

En las aplicaciones distribuidas cada componente se ejecuta en un lugar determinado dentro de las distintas capas o niveles de la arquitectura.

Un ejemplo de este tipo de aplicaciones es la tecnología Java EE (Java Enterprise Edition).

El modelo de aplicación Java EE consiste en el uso del lenguaje Java en una máquina virtual Java como base, y un entorno controlado llamada “middle tier” que tiene acceso a todos los servicios empresariales. El middle tier suele ejecutarse en una máquina dedicada.

Por lo tanto es un modelo multinivel en el que la lógica de negocio y la presentación corren a cargo del desarrollador y los servicios estándar del sistema a cago de la plataforma Java EE.

Para las aplicaciones empresariales usa una arquitectura multinivel distribuida donde la lógica de la aplicación se divide en varios componentes según su función y estos componentes residen en distintas máquinas dependiendo del nivel al que pertenezca el componente de la aplicación.

Las distintas capas son:

- Client-tier: componentes que se ejecutan en la máquina cliente.
- Web-tier: componentes que se ejecutan en el servidor Java EE.
- Business-tier: componentes que se ejecutan en el servidor Java EE.
- EIS-tier (Enterprise Information Systems): software que se ejecuta en el servidor EIS (normalmente un servidor de bases de datos).

Las aplicaciones Java EE pueden ser aplicaciones de 3 o 4 capas, aunque por el número de actores implicados se suele decir que es una arquitectura de tres capas.

Las aplicaciones Java EE están formadas por **componentes**. Los componentes son piezas de software autocontenidas y funcionales en si mismas que se integran con la aplicación y se comunican con otros componentes.

Se pueden clasificar en:

- Aplicaciones cliente y applets, que corren en el cliente.
- Servlets, componentes JSP y JavaServer Faces que corren del lado del servidor.
- Componentes EJB (Enterprise Java Beans) que corren del lado del servidor.

Los **clientes** pueden ser:

- Aplicaciones cliente: suelen ser aplicaciones gráficas o en línea de comandos que se comunican con la capa de negocio directamente o a través de conexiones HTTP.
- Clientes Web: tienen dos partes, páginas web dinámicas que contienen varios elementos que se generan en el web-tier y un navegador web que muestra las páginas generadas. Las tareas pesadas las ejecuta un servidor Java EE.
- Applets: son pequeños programas escritos en Java e inclrustados en páginas web. Estos programas se ejecutan en una máquina Java instalada en el navegador.

El método preferido es usar componentes web, puesto que se desliga el diseño de las páginas con la programación, pudiéndose hacer por equipos diferentes o multidisciplinares.

Los **componentes web** pueden ser servlets o páginas creadas con tecnologías JSP o JavaServer Faces.

- Servlets: son clases java que se ejecutan en el servidor, procesan peticiones y generan respuestas de forma dinámica.
- Páginas JSP: son ficheros de texto que se ejecutan como servlets pero cuyo contenido mezcla elementos estáticos con elementos interpretados.
- JavaServer Faces: es una tecnología que se apoya en servlets y JSP para proporcionar un conjunto de herramientas para crear interfaces de usuario para aplicaciones web.

Los **componentes de negocio** son los que implementan la lógica de la aplicación. Resuelven los problemas para los que se diseñó la aplicación. Pueden ejecutarse en la capa web o en la capa de negocio. Son los componentes que se comunican con el almacenamiento de datos.

> En Java existen las EJB (Enterprise Java Beans) que es una API de Java EE que indica como los servidores de aplicaciones proveen objetos del lado del servidor llamados EJBs. Es un modelo distribuido del lado del servidor para abstraer al programador de varios aspectos generales.

Contenedores de servlets

Un contenedor de servlets es un componente de un servidor web que interactúa con los servlets. Implementa el componente web de la arquitectura Java EE.

Sus funciones principales son las de manejar el ciclo de vida de un servlet, mapearlo a una URL y especificar un entorno de trabajo para el servlet.

Además se ocupa de cuestiones como: seguridad, concurrencia, transacciones, implementación (deployment), etc.

Un servlet es una clase de Java que extiende las capacidades del servidor que aloja la aplicación usando un modelo petición-respuesta. Se podría pensar como un applet que corre del lado del servidor.

Es conforme a la API Java Servlet, no está ligado a ningún protocolo en concreto, pero comúnmente se usa junto con HTTP.

Se usan en los servidores web para generar contenido dinámico.

> Uno de los contenedores de servlets mas completos y populares es Apache Tomcat.

Apache Tomcat es un servidor HTTP 1.1 además de un contenedor de servlets, por lo que puede hacer las funciones de frontal + contenedor de servlets o contenedor de servlets autónomo.

Mantenimiento del estado usando protocolos “stateless”

Las aplicaciones web utilizan el protocolo HTTP para comunicar el cliente con el servidor, y también como método de comunicación entre distintos componentes.

Este protocolo tiene una característica importante: no mantiene el estado entre peticiones, es un protocolo “stateless”. Esto quiere decir que trata cada petición de forma independiente de las anteriores.

Los protocolos sin estado tienen ventajas e inconvenientes: son muy simples pero a veces es necesario incluir mas información en cada petición y ésta ha de ser procesada.

Esta característica es una gran desventaja cuando se necesita mantener información entre peticiones, por ejemplo para saber el usuario que está accediendo a la web o en qué punto de un procedimiento se encuentra.

Para paliar esta desventaja existen varios métodos que ayudan a mantener cierta información entre distintas peticiones:

- Cookies: consiste en almacenar en el cliente información necesaria para relacionarlo con peticiones anteriores, como por ejemplo un identificador de usuario u otra información relevante. Este método tiene la desventaja de que el usuario puede desactivar el uso de cookies y por lo tanto no ser válido.
- Tokens: consiste en generar un identificador único y agregarlo a la URL, de forma que al leer la URL el servidor puede ofrecer información que relacione la petición actual con peticiones anteriores.

En ambos casos se puede combinar con información almacenada en el servidor, de forma que los datos que se desean conservar entre peticiones se mantenga en el servidor y se use la información del cliente para localizar dichos datos.

Introducción a las sesiones de usuario en entornos distribuidos

Una característica importante de los lenguajes de programación del lado del servidor es el uso de sesiones de usuario. Se entiende por sesión de usuario la información mantenida por el servidor usada en las peticiones que vienen de un mismo navegador en un intervalo de tiempo determinado.

Con este método se permite una cierta forma de persistencia de la información entre peticiones de forma que se mantiene el estado de la aplicación aunque se use un protocolo sin estado.

Cuando un usuario accede a una aplicación web ésta puede buscar información proporcionada por el cliente para establecer una sesión. Los servlets y los programas en PHP, por poner dos ejemplos, pueden establecer un identificador de sesión a partir de una cookie almacenada en el cliente o a través de la URL que se solicita el servidor. Otros lenguajes utilizan otros métodos como por ejemplo el paso de parámetros en las peticiones GET y POST para crear manualmente una sesión de usuario.

Normalmente las sesiones de usuario manejadas por los lenguajes de programación del lado del servidor son mas fáciles de usar y suelen ser mas seguras que las sesiones creadas y gestionadas por el propio programador.

Las sesiones gestionadas por el servidor permiten almacenar información en el servidor mientras dure la sesión. La forma de almacenar la información de la sesión depende de la plataforma y del lenguaje de servidor usado. De todos modos si se quiere almacenar cierta información manualmente siempre se puede usar el identificador de sesión como índice de búsqueda en un almacenamiento controlado por el programador.

En Java EE las peticiones HTTP que se usan en los servlets pueden tener asociada una sesión manejada por el servidor. Esta sesión se establece a través de una cookie del navegador, y si ello no es posible entonces se hace a través de la reescritura de la URL.

En el siguiente ejemplo se obtiene un identificador de sesión en un servlet. Se han resaltado las partes de código mas interesantes:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * @author pedro
 */
public class ejemploSesion extends HttpServlet {
  protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
       PrintWriter out = response.getWriter();
        HttpSession sesion = request.getSession(true);
        try {
```

```
            out.println("Identificador de sesi&oacute;n: " + sesion.getId());
        } finally {
            out.close();
        }
    }
     /* . . . resto del código del servlet . . */
}
```

Servlet que muestra el identificador de sesión asignado por el servidor.

Existen casos en los que por la propia arquitectura de la aplicación o bien por el tamaño de los datos que maneja el servidor no deben ser guardados como parte de la sesión, por ejemplo cuando se suben ficheros, ya que la información de sesión en Java se guarda como un objeto en el servidor, y ocupa memoria, haciendo que los requisitos de la aplicación se disparen y degeneren en una aplicación poco eficiente. Para estos casos se suele usar algún tipo de almacenamiento secundario, como una base de datos o un sistema de ficheros en red, que permite almacenar información relacionada con la sesión de usuario pero que no maneja el servidor. En estos casos hay que establecer métodos que limpien el almacenamiento secundario cuando la información de sesión ya no es necesaria.

En los entornos distribuidos en los que existe mas de un servidor para ejecutar una aplicación como por ejemplo grupos de servidores y entornos de alta escalabilidad, el mantenimiento de la sesión puede ser problemático. En estos entornos se pueden usar varias estrategias para mantener la sesión cuando se usan varios servidores:

- Afinidad: cada sesión de usuario se asigna a un servidor, de forma que siempre es el mismo servidor el que atiende a una sesión y por lo tanto el problema queda solucionado.
- Replicación: la sesión se replica entre todos o parte de los servidores de forma que cuando llega una petición de usuario se recupera la sesión en cualquiera de los servidores.

Es muy común usar varios servidores que trabajan en paralelo para atender servicios de alta demanda o que necesitan gran disponibilidad, por lo que mantener la sesión de usuario envarios servidores es una necesidad muy importante.

Se procura que los métodos de mantenimiento de sesión sean transparentes para las aplicaciones, aunque la implementación en cada tipo de servidor sea distinta.

Por supuesto siempre se puede hacer de forma manual si el programador así lo desea.

En el tema 2 se verá con mas profundidad el manejo de sesiones en Java.

Prácticas

Instalación de un contenedor de servlets: Tomcat.

Esta práctica ilustra cómo instalar un servidor Tomcat en un ordenador con Windows.

IMPORTANTE: Es necesario tener instalado Java 6 o superior.

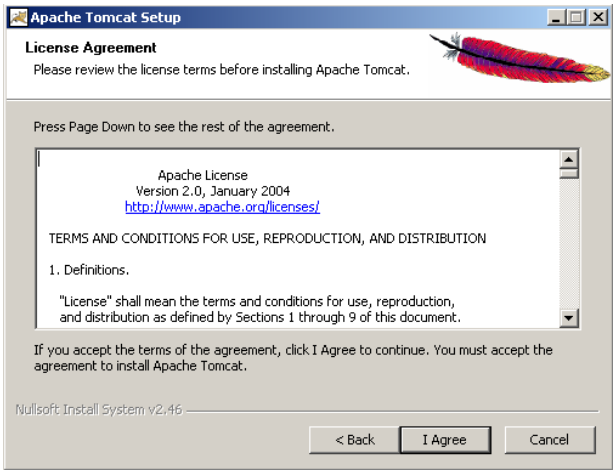
1 Descargar Apache Tomcat desde http://tomcat.apache.org/

El fichero a descargar dependerá de la versión de Windows en la que se vaya a instalar (32 o 64 bits). E recomendable descargar el "Service Installer" si se quiere que corra como un servicio de Windows.

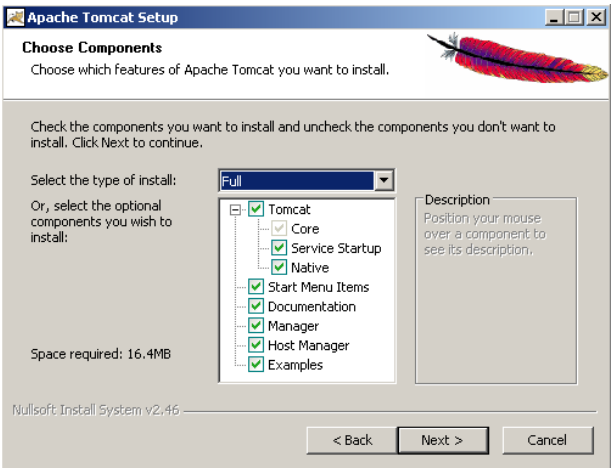
2 Ejecutar el instalador.

3 Pulsar "Next".

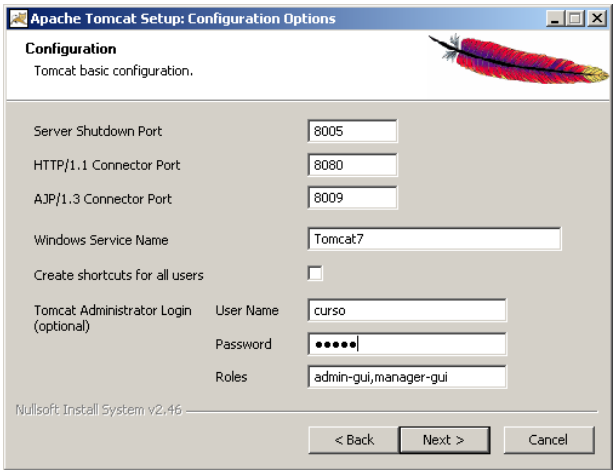
4 Aceptar el contrato de licencia. Pulsar "I Agree".



5 Seleccionar la instalación completa (Full) y pulsar "Next".



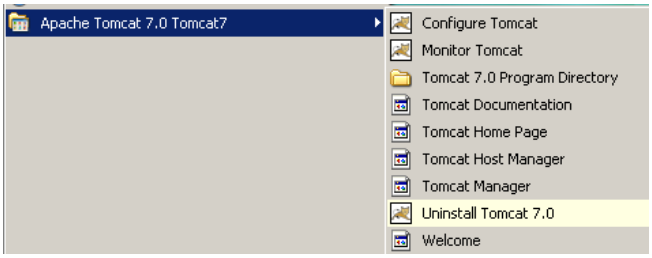
6 Rellenar los datos como se indica.



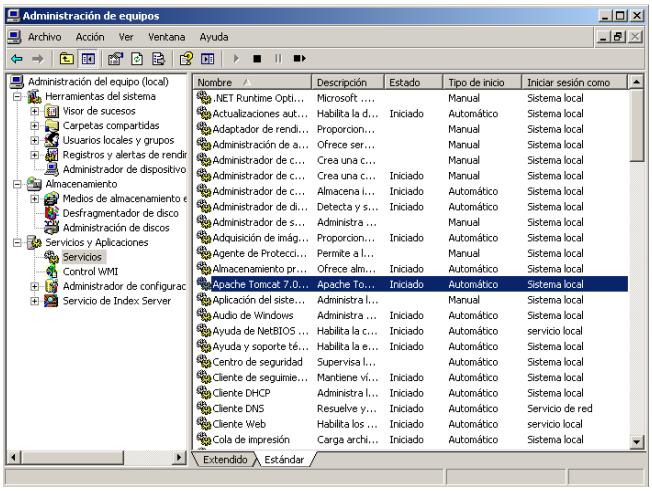
7 Aceptar el resto de parámetros por defecto en las siguientes pantallas.

8 Finalizar la instalación.

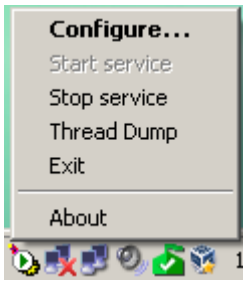
Una vez terminada la instalación de Apache Tomcat:



Se ha creado un grupo de programas en el menú inicio tal y como muestra la figura.



Para arrancar y parar Tomcat se puede usar el gestor de servicios de Windows.



Tomcat se puede gestionar desde un icono en la barra de tareas.

Para comprobar que se ha instalado correctamente se puede acceder a la dirección http://localhost:8080/ y comprobar que no da error.

Instalación de un entorno WAMP: Windows + Apache + PHP + MySQL

Se instalará el servidor Ampps descargado desde http://www.ampps.com/download.

Se descarga el paquete u se siguen las instrucciones en pantalla.

Es importante instalar el servidor MySQL puesto que se utilizará en ejercicios posteriores.

Tema 2

Generación dinámica de páginas web.

Objetivos:

- Conocer las distintas formas en las que se genera el contenido web dinámico.
- Conocer los métodos y tecnologías usadas actualmente en la programación del lado del servidor.

Métodos de separación de la presentación y la lógica de negocio

Con la llegada de las aplicaciones web empresariales y la popularización del uso de Internet, las aplicaciones han tenido que ir evolucionando para dar respuesta rápida a un elevado número de usuarios. También se han tenido que mejorar las medidas de seguridad y de interoperabilidad, ya que en muchas ocasiones las aplicaciones web deben acceder a servicios ya implementados sean propios o de terceros.

Una de las principales evoluciones ha consistido en separar las aplicaciones en varias partes, de forma que cada una de ellas se especializa en un cometido. Este tipo de arquitecturas presenta importantes ventajas, como por ejemplo:

- Mejoras en el tiempo empleado en desarrollar las aplicaciones. Se pueden destinar recursos distintos para las distintas partes, pudiendo realizar el desarrollo de cada componente en paralelo con el resto.
- Mayor escalabilidad de las aplicaciones. Los entornos de las aplicaciones pueden crecer de forma mas rápida y ordenada al separar cada una de las partes en servidores físicos distintos. Por ejemplo, se pueden emplear granjas de servidores web para la parte estática y frontales, y clústeres de bases de datos para el almacenamiento y recuperación de la información.
- Seguridad mejorada. El acceso de cada componente puede ser auditado y autorizado, de forma que no se pueda acceder a información a la que no se tenga permiso.
- Uso de distintas tecnologías. Al usar estándares para la comunicación e intercambio de la información se puede emplear en cada parte las tecnologías mas adecuadas en cada momento.
- Mantenimiento simplificado. Los cambios en una de las partes no tienen por qué afectar al resto, demás se pueden localizar los errores con mayor exactitud así como facilitar los cambios que de deban realizar en las aplicaciones.

Existen varios métodos para la separación de la presentación y la lógica de negocio. Los mas importantes son:

- Cliente-servidor: el cliente interactúa con el usuario y el servidor procesa la información. Es uno de los métodos mas antiguos, y que ha dado lugar a nuevas arquitecturas.

- Arquitecturas multicapa: consiste en dividir la aplicación en varios niveles o capas que realizan tareas concretas. La información fluye entre las capas contiguas en una organización vertical.
- Arquitectura Modelo – Vista – Controlador (MVC): consiste en separar la aplicación en tres partes, el modelo (datos y reglas de negocio), la vista (representación de los datos) y el controlador (proceso de los datos de entrada). La diferencia fundamental con la arquitectura de tres capas reside en la forma en que se comunican los tres elementos.

Tecnologías POO y multicapa

Para lograr la separación de la presentación de la lógica de negocio se utilizan tecnologías basadas en la programación orientada a objetos y multicapa. La combinación de estas tecnologías produce aplicaciones cuyas ventajas ya se han visto en el apartado anterior.

Dentro de las tecnologías utilizadas destacan las basadas en Java y en ASP.net.

Java, como lenguaje orientado a objetos, proporciona clases que implementan tecnologías usadas en distintas capas. Las tecnologías mas importantes son:

- JavaServer Faces: trabaja en la capa web del servidor proporcionando métodos para separar la representación y el comportamiento de las aplicaciones web. Simplifica la creación de interfaces de usuario.
- Java Servlet technology: extiende la capacidad de un servidor web mediante clases usando un modelo de petición-respuesta. Trabaja en el middle-tier.
- JavaServer Pages: simplifica y acelera la creación de contenido web dinámico embebiendo código Java directamente en los documentos HTML o XML.
- Tecnologías de servicios web (web services): JAX-RS, JAX-WS, JABX, JAX-RPC, XML messagging, etc.
- Enterprise Beans: son controles del servidor que ayudan al programador proporcionando servicios de bajo nivel que ayudan al desarrollo de aplicaciones distribuidas.

Java proporciona prototipos y APIs para facilitar el uso de estas tecnologías.

ASP.net, a través de su herramienta de desarrollo Visual Studio, proporciona controles para implementar las distintas capas basándose en tecnologías propietarias de Microsoft y en estándares. Un ejemplo de estas tecnologías son los controles de servidor, como por ejemplo:

- Controles de servidor HTML.
- Controles de servidor web.
- Controles web básicos.
- Controles de validación.

- Controles de usuario.
- Controles personalizados.

Controles de servidor

En esta sección se discutirá acerca de los controles de servidor en Java, usando las tecnologías JavaServer Faces, Servlet y Enterprise Beans.

JavaServer Faces es un framework para construir interfaces de usuario en aplicaciones web. Sus componentes principales son:

- Framework para componentes de interfaz gráfica de usuario.
- Un modelo flexible para renderizar los componentes en diferentes lenguajes de marcas incluyendo HTML.
- Un kit de renderizado para generar HTML 4.01.

Los componentes de la interfaz gráfica de usuario (GUI) soportan las siguientes características:

- Validación de entrada.
- Manejo de eventos.
- Conversión de datos entre objetos del modelo y componentes.
- Configuración de la navegación por la página.

En la versión 6 se han añadido, entre otras cosas:

- Facelets, que reemplazan a la tecnología JSP usando ficheros XHTML.
- Soporte AJAX.

Para ello utiliza APIs estándar de Java y ficheros de configuración XML.

Java Servlet permite procesar las peticiones del cliente y generar una respuesta. Los servlets son programas escritos en Java que pueden mantener un estado respecto al cliente. Los servlets son componentes de servidor que responden a peticiones de clientes, normalmente por HTTP.

En este manual se verán principalmente servlets que funcionan con el protocolo HTTP. La estructura de un servlet es la de una clase Java que extiende la clase abstracta HttpServlet.

Los servlets tienen un ciclo de vida que controla el servidor. Este ciclo de vida comienza con la carga de la clase y la instanciación del servlet. Una vez instanciado se llama al método init() y el servlet queda listo para atender peticiones. Cada petición que se realiza provoca una llamada al método service().

El ciclo de vida del servlet termina por alguna de las siguientes circunstancias:

- Termina la ejecución del servidor.

- Se reemplaza el servlet por una nueva versión.
- Si el servidor llama al método destroy(). El servidor decide que el servlet debe finalizar su ejecución puesto que ya no se usa, por ejemplo.

La estructura de un servlet se puede ver en el siguiente ejemplo:

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

public class EjmeploSencillo extends HttpServlet {

   public void doGet(HttpServletRequest peticion, HttpServletResponse resuesta)
                        throws ServletException, IOException {

         respuesta.setContentType("text/html");
         PrintWriter out = respuesta.getWriter();

         out.println("<html>");
         out.println("<head><title>Hola mundo</title></head>");
         out.println("<body>");
         out.println("Hola mundo");
         out.println("</body></html>");
   }
}
```

Ejemplo de servlet sencillo.

Los siguientes métodos son algunos de los mas usados en los servlets:

- `doGet()`: se ejecuta en peticiones de tipo GET.
- `doPost()`: se ejecuta en peticiones de tipo POST.
- `doPut()`: se ejecyta en peticiones de tipo PUT.
- `service()`: recibe la petición HTTP y la redirige a alguna de las anteriores.

Los métodos anteriores reciben un par de objetos como parámetros:

- HttpServletRequest: un objeto que almacena la petición. Se puede preguntar al objeto sobre parámetros, sesiones y en general sobre todo lo referente a las peticiones.
- HttpServletResponse: es un objeto que permite manipular la respuesta que el servidor ofrecerá al cliente. La respuesta puede contener códigos de error, cookies, redirecciones a otras páginas, se pueden insertar cabeceras, etc.

Un servlet, como cualquier programa Java, puede crear objetos y utilizar otras clases, teniendo en cuenta las limitaciones del servidor (por ejemplo no se pueden crear aplicaciones que lean del teclado o que usen elementos de la librería swing).

Los **Enterprise Beans** requieren un servidor de aplicaciones para funcionar. Son componentes del servidor que encapsula la lógica de la aplicación, proporcionando servicios a otras partes de la aplicación.

Los enterprise beans simplifican el desarrollo de grandes aplicaciones distribuidas, descargando al programador de tareas de bajo nivel. También permiten crear clientes mas ligeros y separar la presentación de la lógica. Otra de las ventajas de los EB es la reutilización, ya que un mismo componente se puede usar varias veces en una aplicación y/o en distintas aplicaciones que corren en un mismo servidor de aplicaciones. Se deben usar EB cuando la aplicación necesite crecer y distribuirse entre varios servidores, cuando se necesite integridad de los datos (los EB soportan transacciones) o cuando se quiera acceder desde varios clientes distintos.

Existen dos tipos de EB:

- De sesión: ejecutan una tarea para un cliente. Opcionalmente pueden implementar un servicio.
- Dirigidos por mensajes: actúa como un oyente para un tipo particular de mensaje. Permite procesar mensajes de forma asíncrona.

Mantenimiento del estado

En Java EE la sesión se representa mediante un objeto HttpSession. La API proporciona varios mecanismos para implementar las sesiones.

Las sesiones se mantienen con cookies o reescribiendo la URL. Si un cliente no soporta cookies o tiene el soporte de cookies desactivado se utiliza la reescritura de URL.

Las sesiones tienen un tiempo de espera (timeout) de forma que si no se accede a la información de una sesión en un tiempo determinado la sesión caduca y se destruye. También se puede terminar explícitamente una sesión.

Los componentes web pueden enviar y recibir información de sesión siempre que manejen una petición que pertenece a una misma sesión y estén en el mismo contexto web.

En el tema 1 se vio una breve introducción al manejo de sesiones en Java, con un ejemplo que ilustraba cómo obtener el identificador de sesión asociado a un cliente.

En la documentación de Java EE se detallan los métodos que pueden usarse en un objeto HttpSession: http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html

A partir de la especificación Servlet 2.2 de Java los métodos para el manejo de sesiones que se aconseja usar se basan en atributos. Los mas interesantes son:

- `getAttributeNames()`: obtiene una enumeración de los nombres de los atributos almacenados en la sesión.
- `getAttribute(String atributo)`: devuelve el objeto asignado a una sesión bajo el nombre de atributo dado, o `null` si no existe.
- `setAttribute(String nombre, Object atributo)`: asigna un objeto a una sesión bajo el nombre de atributo dado.

- `removeAttribute(String nombre)`: elimina la asignación del objeto a la sesión.
- `invalidate()`: elimina la sesión y todas las asignaciones de objetos a la misma.

El siguiente ejemplo ilustra el uso de las sesiones en Java EE:

```
public class contador extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        HttpSession sesion = request.getSession(true);
        Integer contador = 0;
        try {
           contador = (Integer) sesion.getAttribute("contador");
           if (contador == null) {
               contador = 0;
           }
           contador ++;
           sesion.setAttribute("contador", contador);
        } finally {
            out.println("Ha visitado esta pagina " + contador + " veces.");
            out.close();
        }
    }
```

Implementación de un contador usando sesiones.

Java EE proporciona otros métodos para el control de las sesiones, incluyendo los "Session Listener", que son clases que pueden "escuchar" y actuar cuando se crean y se destruyen sesiones.

Los servidores modernos, ya sean contenedores de servlets o servidores de aplicaciones, implementan métodos para mantener la sesión independientemente del servidor que atienda la petición de un cliente en entornos donde se utilizan varios servidores trabajando en paralelo. Un ejemplo de toolkit usado para replicar las sesiones entre servidores es Jgroups (http://www.jgroups.org/) que implementa un sistema de comunicación confiable entre servidores usando multicast.

Apache Tomcat puede usar métodos de afinidad implementada en servidores frontales o bien usar técnicas de replicación para el mantenimiento de la sesión en un cluster de servidores.

Métodos de generación del contenido web

Un servidor web debe enviar un contenido como respuesta a las peticiones del cliente. Este contenido puede ser estático o dinámico. El contenido dinámico se genera en el servidor y puede ser diferente en función de la petición que efectúa el cliente.

Existen diferentes métodos de generación de contenido web según el lenguaje y la tecnología empleados.

Atendiendo a la forma de obtener el código HTML se pueden citar los siguientes métodos:

- Código embebido. En un fichero de texto con código HTML se insertan marcas e instrucciones que son interpretadas por el servidor. El resultado es la suma del código HTML y el resultado de la interpretación de las marcas y códigos embebidos en el fichero.

  Ejemplos típicos de código embebido son ASP, PHP y JSP.

- Generación de código HTML. La petición del cliente hace que se ejecute un programa en el servidor, y este programa genera código HTML como salida.

  Ejemplos de código generado son los programas CGI y los servlets, aunque en general cualquier método de ejecución del lado del servidor es capaz de generar código HTML.

- Uso de plantillas. Los programas que se ejecutan en el servidor usan plantillas en HTML para generar la salida que se envía al cliente. Las plantillas pueden ser ficheros con código HTML, XML u otro lenguaje de marcas, información extraída de una base de datos o una combinación de ambos.

- Frameworks y librerías. El programador puede hacer uso de frameworks o librerías para generar el código. Es una particularización de la generación de código pero en este caso el programador no genera la salida directamente.

  Ejemplos de framework y librerías son JavaServer Faces y controles web de ASP.net.

Atendiendo a la integración con el servidor se pueden citar:

- Ejecución en el servidor. El servidor web ejecuta directa o indirectamente un programa para generar el contenido dinámico.

  Ejemplos: CGI.

- Integración con el servidor. El servidor posee uno o varios módulos para procesar las peticiones que generan contenido dinámico.

  Ejemplos: PHP, ASP.

- Generación por el propio servidor. El servidor crea un entorno en el que se procesan las peticiones dinámicas.

  Ejemplos: servidores de aplicaciones y contenedores de servlets.

- Proxy. El servidor actúa como intermediario y reenvía la solicitud a otros servidores que se encargan de generar el contenido dinámico.

  Ejemplos: workers AJP.

- Aplicaciones distribuidas. Un servidor de aplicaciones hace uso de servicios que residen en otras máquinas para generar el contenido dinámico.

  Ejemplos: aplicaciones Java EE, servicios web, arquitecturas de n-capas y MVC.

Formularios de respuesta

La mayoría de las aplicaciones web utiliza formularios para recoger información y posteriormente procesarla. Desde pantallas de inicio de sesión a formularios de registro en un sitio web el uso de formularios cumple una de las principales funciones de interacción con el usuario.

Puesto que los formularios con principalmente usados por personas es importante que éstas reciban información acerca del resultado de la introducción de datos, bien con un mensaje indicando que los datos han sido recibidos con éxito por el servidor o bien informar acerca de qué datos se consideran erróneos para que el usuario pueda corregirlos.

Los formularios de respuesta bien hechos son capaces de procesar la información en dos fases, una del lado del cliente usando JavaScript y otra del lado del servidor. También se pueden usar técnicas como AJAX para crear formularios inteligentes que den respuesta rápida al usuario o que se adapten para ofrecer alternativas según las respuestas que va introduciendo el usuario.

Los formularios se implementan como peticiones GET o POST al servidor, aunque en los formularios inteligentes parte de las peticiones que realiza el cliente pueden ser GET.

Los formularios simples siguen esta secuencia:

1. Acceso al formulario.
2. El usuario introduce los datos en el formulario.
3. Los datos se envían al servidor (GET o POST).
4. El servidor procesa los datos.
5. Si los datos no son correctos vuelve al paso 1.

   Si los datos son correctos redirige a la página de respuesta.

En el servidor los datos del formulario se traducen a variables que pueden ser evaluadas por el programador.

Cuando se diseña un formulario se deben tener en mente algunas recomendaciones:

- Limitar el tamaño de los campos del formulario. Si se sabe que un campo no debe tener un contenido que exceda de un número determinado de caracteres se debe limitar a fin de evitar ataques por desbordamiento de buffer.
- Procesar los valores de los campos y eliminar los caracteres que puedan provocar ma funcionamiento de la aplicación. Ataques de este tipo son los llamados “SQL injection”, donde el atacante inserta sentencias SQL que alteran el funcionamiento de la aplicación.

> Supongamos una aplicación de consulta de calificaciones donde un alumno inserta su identificador de usuario y obtiene sus calificaciones. Un usuario malintencionado podría escribir en el campo de texto algo similar a “`usuario; DROP DATABASE calificaciones;`”, de modo que al ejecutar la sentencia SQL también se ejecuta un “drop” de la base de datos, dejando sin servicio la aplicación y derivando en una pérdida de datos.

- Enviar siempre una respuesta para cada uno de los casos posibles, si asumir que un caso en concreto nunca va a ocurrir. Esto evita situaciones en las que la aplicación queda en un estado desconocido o que el usuario de la aplicación no puede continuar utilizándola normalmente.
- Usar campos ocultos solo cuando sea necesario. Los campos ocultos ayudan e enviar información que no se quiere que el usuario conozca, pero son también una fuente de problemas de seguridad fácilmente explotable por un atacante. Es recomendable usar datos de sesión en vez de formularios con campos ocultos.

En un servlet los datos del formulario se pueden procesar en el método `doGet()` o `doPost()`. Los datos de los campos se obtienen mediante el método getParameter(). Si un parámetro tiene mas de un valor se puede recuperar un array con todos los valores usando el método `getParameterValues()`.

Usar formularios en conjunción con variables de sesión permiten realizar aplicaciones con formularios enlazados, como por ejemplo asistentes.

A la hora de generar un formulario hay que tener en cuenta que los formularios estáticos no sirven de mucha ayuda cuando se desean notificar datos dinámicos, por ejemplo cuando un usuario ha introducido mal un campo y se desea presentar de nuevo el formulario con los datos que el usuario ha introducido anteriormente para que no tenga que volver a rellenarlos. Esto se puede conseguir mediante páginas HTML con código embebido (por ejemplo JSP) o bien con servlets que generen el formulario, bien directamente o bien mediante plantillas.

En el siguiente ejemplo se muestra un servlet que crea y procesa un formulario sencillo.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        String nombre = (String) request.getParameter("nombre");
        // Indica si se debe mostrar el formulario.
        Boolean mostrarFormulario = true;
        // Indica si los datos del formulario son correctos.
        Boolean procesoCorrecto = true;

        // Ver si los datos del formulario son correctos.
        if(nombre != null) {
            if(nombre.length() > 0) {
                // Se ha escrito un nombre en el campo del formulario.
                mostrarFormulario = false;
                out.println("Hola " + nombre + "!");
            } else {
                procesoCorrecto = false;
            }
        }
        if(mostrarFormulario == true) {
```

```
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Diga su nombre</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<form action=\"formulario1\" method=\"POST\">");
            out.println("Introduzca su nombre: <input type=\"text\" maxsize=\"30\"
name=\"nombre\">");
            if(procesoCorrecto == false) {
                out.println("<font color=\"red\">Debe introducir un nombre.</font>");
            }
            out.println("<input type=\"submit\">");
            out.println("</form>");
            out.println("</body>");
            out.println("</html>");
        }
    } finally {
        out.close();
    }
}
```

Servlet que muestra un formulario sencillo, lo procesa y responde en función de la entrada.

Prácticas

Creación y despliegue de una aplicación Java sencilla que presente un mensaje en HTML usando varios métodos

Para esta práctica se usará el entorno de desarrollo integrado Netbeans. Este entorno tiene soporte para el lenguaje Java y se puede extender su funcionalidad mediante plugins. También tiene soporte para los servidores de aplicaciones y contenedores de servlets mas usuales, por lo que es uno de los entornos mas indicados para desarrollar aplicaciones Web Java.

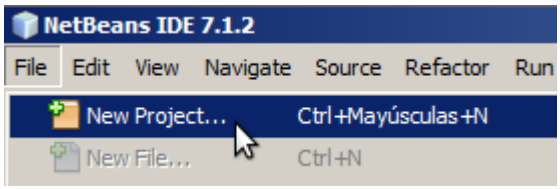
Posee varias ventajas como la ayuda en la escritura de programas proporcionando generación de código, validación del programa y otras ayudas que hacen de este entorno una valiosa herramienta para el programador.

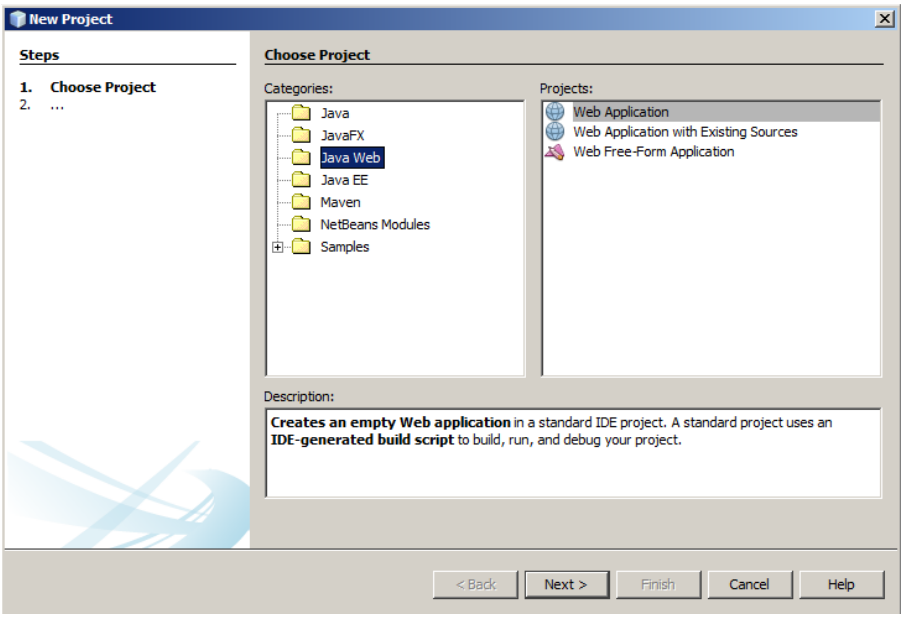
La práctica consiste en crear una página JSP y un servlet que muestren un sencillo mensaje. Estos programas deberán desplegarse en un servidor Tomcat previamente instalado.

Pasos:

1 – Abrir el entorno de desarrollo integrado Netbeans.

2 – Crear un nuevo proyecto.

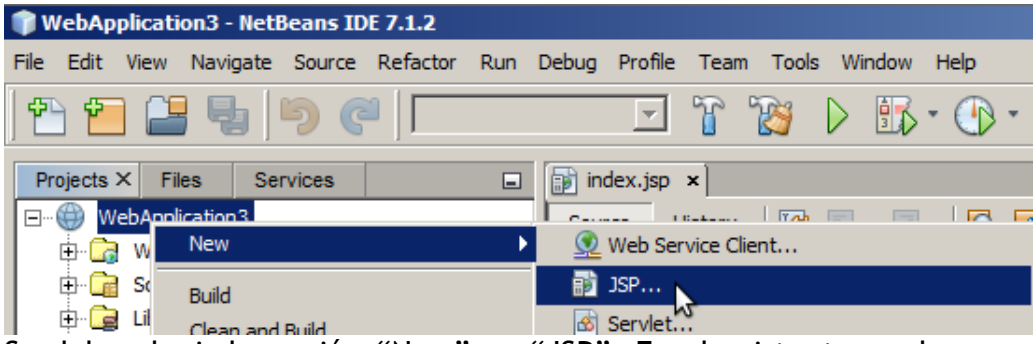




Seguir el asistente hasta el final aceptando las opciones por defecto.

3 – Añadir una página JSP.

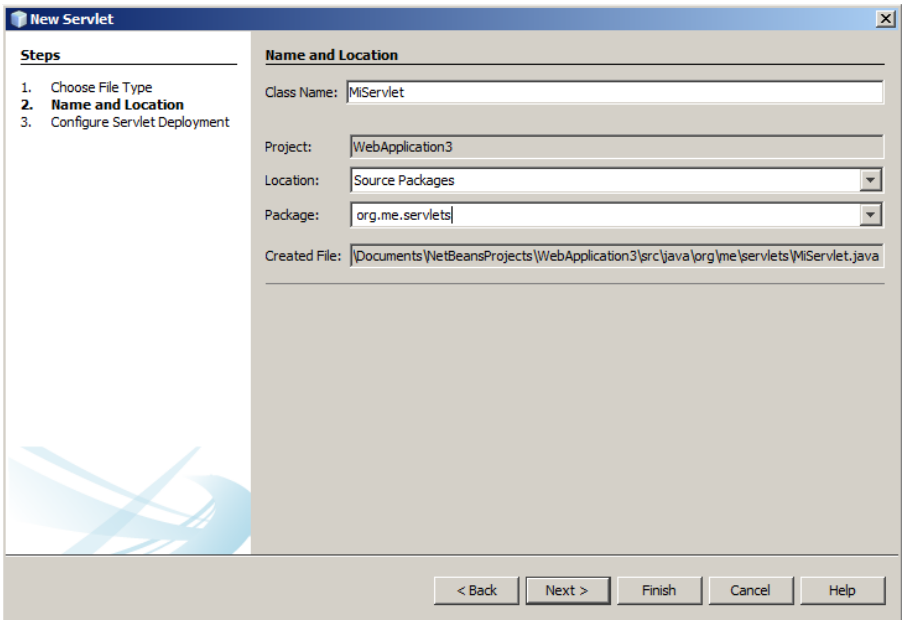
El editor crea una página JSP llamada index.jsp, pero se pueden añadir mas desde el menú contextual del panel de proyectos:



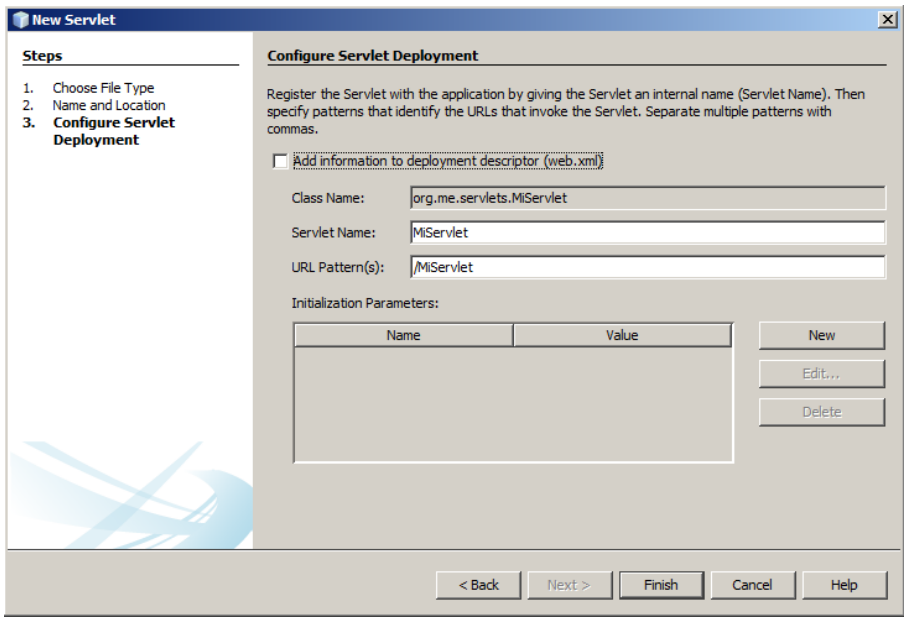
Se debe elegir la opción “New” → “JSP”. En el asistente se da un nombre a la página JSP y se pulsa en el botón “Finish”.

4 – Añadir un servlet.

Para añadir un servlet se procede como en el paso anterior pero se selecciona la opción “New” → “Servlet”. Se abre un asistente en el que hay que dar un nombre al servlet y asignarla a un paquete. Se puede crear un paquete nuevo o usar uno ya existente.



En el siguiente paso se le asigna una URL dentro de la aplicación y opcionalmente los parámetros de inicialización.



El editor genera el fichero con el contenido del servlet.

Tema 3

Técnicas de acceso a datos.

Objetivos:

- Conocer los métodos de acceso a servidores de datos en diferentes sistemas.
- Manejar datos del servidor usando SQL.
- Conocer otros orígenes de datos.

Introducción a las conexiones a servidores SQL

Las aplicaciones del lado del servidor necesitan acceder a datos, ya sea de forma temporal durante el ciclo de vida de la aplicación, o bien de forma permanente.

El almacenamiento temporal ya se ha visto al tratar las sesiones de usuario. Los datos de las aplicaciones se guardan en la memoria del servidor y se destruyen cuando la aplicación termina.

El almacenamiento permanente permite que los datos recogidos o procesados por la aplicación estén disponibles para un procesado posterior. El almacenamiento permanente también permite acceder a datos generados con anterioridad por la aplicaciones o por otras aplicaciones.

El almacenamiento permanente las extendido son los servidores de bases de datos, y entre ellos los servidores de bases de datos relacionales que utilizan SQL como lenguaje de consulta.

El uso de servidores SQL requiere seguir ciertos pasos:

1. **Establecer una conexión**. Normalmente se usa una cadena de conexión particular para cada tipo de servidor. Una vez establecida la conexión con el servidores ya se puede comenzar a usar.

   El establecimiento de la conexión suele requerir conocer la dirección del servidor, el usuario y contraseña de acceso, el nombre de la base de datos que se desea utilizar, y en ciertas ocasiones el protocolo de comunicaciones.

2. **Crear una consulta**. Los servidores SQL trabajan mediante consultas SQL. Con una consulta se pueden insertar, modificar, eliminar y obtener datos almacenados en una o varias y tablas del servidor.

   Una vez creada la consulta se envía al servidor.

3. **Procesar la respuesta**. La respuesta varía en función de la consulta realizada. Una consulta puede devolver un resultado de tipo correcto/incorrecto (por ejemplo si se está insertando un dato) o bien una serie de datos organizados en una tabla de resultados.

   Los pasos 2 y 3 se pueden repetir tantas veces como sea necesario.

4. **Cerrar la conexión**. Una vez finalizado el uso de la conexión se procede a su cierre. En ciertos lenguajes se puede producir de forma automática al terminar la

ejecución del programa. En algunos lenguajes se pueden usar conexiones persistentes, de forma que no haya que abrir y cerrar las conexiones cada vez que se carga una página. En Java se pueden usar los llamados “pooles de conexión” para acelerar y mejorar el uso de este tipo de conexiones.

Establecimiento y uso de conexiones SQL en varios lenguajes de programación

Cada lenguaje de programación del lado del servidor tiene sus propios métodos de conexión a un servidor de bases de datos SQL.

Java

Java permite el uso de este tipo de servidores mediante un driver que conecta las API de acceso a servidores de bases de datos con distintos tipos de servidores reales. La tecnología usada por Java para acceder a servidores de bases de datos se llama JDBC (Java Database Connectivity).

La tecnología JDBC es la forma en que los programas Java acceden a los servidores de bases de datos. Esta API no es exclusiva del servidor y forma parte de la familia de tecnologías Java SE.

La arquitectura JDBC está formada por la API JDBC y uno o varios drivers JDBC.

Los drivers JDBC pueden ser de cuatro tipos:

- Tipo 4: es un driver Java puro que conecta directamente con la base de datos usando los protocolos de red propios del servidor de bases de datos.
- Tipo 3: es un driver Java puro que conecta con una capa de middleware del servidor el cual a su vez traduce las consultas al protocolo del servidor de bases de datos. Utiliza por lo tanto una capa mas que el driver tipo 3.
- Tipo 2: es un driver que habla con una librería cliente que se conecta con el servidor de bases de datos. Requiere software extra instalado en el servidor. Este software no está escrito en Java y depende de la plataforma del servidor.
- Tipo 1: habla directamente con un driver ODBC local que realiza la conexión con el servidor de bases de datos.

Se suele preferir el uso de drivers tipo 2 y tipo 4.

En la página http://devapp.sun.com/product/jdbc/drivers se pueden buscar drivers JDBC para bases de datos.

El API para usar JDBC reside en los paquetes `java.sql` y `javax.sql`.

En Java se pueden usar dos clases: `DriverManager` o `DataSource`.

La forma mas sencilla de usar JDBC es a través de `DriverManager`. Por defecto el objeto creado será de tipo “auto-commit”, que ejecuta los comandos una vez enviados a la conexión.

Al crear la conexión se debe especificar una cadena de conexión que incluye el nombre del driver.

Las cadenas de conexión dependen del driver a usar, ya que parte de esa cadena es enviada al driver para que la interprete y realice la conexión. En general se puede decir que una cadena de conexión genérica puede ser: "`jdbc:`*`driver`*`:`*`parametros`*"

```
Connection con = DriverManager.getConnection("jdbc:myDriver:myDatabase", username,
                        password);
```

Ejemplo de conexión SQL con JDBC usando DriverManager.

Ejemplo: cadena de conexión para el driver oficial de MySQL:

"jdbc:mysql://localhost/mysql?user=*usuario*&password=*clave*"

Una vez creada la conexión se usan los métodos descritos en el interface Connection para trabajar con la conexión a la base de datos.

```
Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

    while (rs.next()) {
        int x = rs.getInt("a");
        String s = rs.getString("b");
        float f = rs.getFloat("c");
    }
```

Ejemplo de consulta a una base de datos a través de una conexión JDBC.

Es recomendable cerrar la conexión una vez que ya no se vaya a utilizar mas. El método a usar para cerrar la conexión es `close()`.

En algunas implementaciones de Java no se carga correctamente el driver, por lo que hay que forzar su carga desde la propia aplicación. Antes de instanciar la conexión se usa esta orden que resuelve el problema:

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

Por otro lado, el método recomendado es usar DataSource. Es mas complejo de configurar pero ofrece ventajas respecto al otro método, como el uso de pooles de conexiones y transacciones distribuídas. En en tornos con muchas conexiones es mejor usar pooles de conexiones que se reutilizan y ahorran recursos al no tener que abrir y cerrar conexiones continuamente.

Los DataSource se implementan en el driver y pueden estar hechos de tres maneras:

- Implementación básica: producen objetos "Connection" estándar que no soportan pooles ni transacciones distribuídas.
- Soporte de pooles: los objetos "Connection" producidos soportan pooles de conexiones, es decir, conexiones que se pueden reutilizar.
- Soporte de transacciones distribuidas: los objetos "Connection" producidos pueden usar transacciones que afectan a mas de un servidor de bases de datos.

El uso de DataSource implica configurar ciertos parámetros que dependen del servidor, por lo que la forma de configurar este tipo de conexiones depende del servidor utilizado. En Tomcat hay que definir las conexiones y posteriormente acceder a ellas a través de JNDI.

De forma genérica se puede realizar la configuración durante el despliegue de la aplicación. Los pasos serían. Respecto a las conexiones realizadas con DriverManager solamente cambia la forma en la que se realiza la conexión:

1. Crear una instancia de la clase DataSource y ajustar sus propiedades.

```
DataSourceDelDriver ds = new com.dbaccess.BasicDataSource();
ds.setServerName("nombreservidor");
ds.setDatabaseName("BASE_DATOS");
ds.setDescription("Descripcion de la base de datos");
```

2. Registrar el objeto DataSource en un servicio de nombres usando la API de JNDI.

```
Context ctx = new InitialContext();
ctx.bind("jdbc/baseDeDatos", ds);
```

3. Usar el objeto.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/baseDeDatos");
Connection con = ds.getConnection("aaa","bbb");
```

Cuando se recupera información de una base de datos relacional los datos se suelen devolver como una tabla o bien como un conjunto de filas. En Java, cuando se usa JDBC se devuelven datos de tipo ResultSet.

Un objeto de clase ResultSet es una tabla de datos resultado de una consulta a una base de datos. Los objetos capaces de realizar una consulta a una base de datos son de tipo Statement en cualquiera de sus variantes: PreparedStatement, CallableStatement y RowSet.

El acceso a los datos del resultset se realiza mediante un cursor, que es un puntero a una fila de la tabla de resultados. No hay que confundirlo con un cursos de base de datos.

Inicialmente el cursor se encuentra en la primera fila de resultados, y a través del método next se mueve a la siguiente file. Si el método devuelve false entonces se ha llegado a la última fila. Habitualmente se usa un bucle while para recorrer los resultados.

Los ResultSet pueden ser de varios tipos:

- De solo avance (TYPE_FORWARD_ONLY): el cursor puede avanzar por los resultados, pero no puede retroceder. Es el tipo por defecto.
- De desplazamiento sin actualización (TYPE_SCROLL_INSENSITIVE): el cursor puede avanzar y retroceder por los resultados. Los resultados no cambian una vez que se ha efectuado la consulta.

- De desplazamiento con actualización (TYPE_SCROLL_SENSITIVE): el cursor puede avanzar y retroceder por los resultados, que se actualizan de forma dinámica con los cambios que se produzcan en los datos originales.

También pueden ser:

- De solo lectura (CONCUR_READ_ONLY): los resultados no pueden ser actualizados a través del interfaz ResultSet. Es el tipo por defecto.
- Actualizables (CONCUR_UPDATABLE): los resultados si pueden ser actualizados a través de l interfaz ResultSet.

El tipo de ResultSet se indica como parámetro en el método `createStatement()`.

Los métodos para desplazarse por un ResultSet son:

- `absolute(int fila)`: desplaza el cursor a la fila indicada.
- `afterLast()`: desplaza el cursor después de la última fila.
- `beforeFirst()`: desplaza el cursor al inicio de los resultados, justo antes de la primera fila.
- `first()`: desplaza el cursor a la primera fila.
- `last()`: desplaza el cursor a la última fila.
- `next()`: desplaza el cursor a la siguiente fila. Devuelve false si no hay mas filas.
- `previous()`: desplaza el cursor a la fila anterior.
- `relative(int filas)`: desplaza el cursor el numero de filas indicado desde la fila actual. El número puede ser positivo (avanza) o negativo (retrocede).

Una vez realizada la consulta y recuperados los datos el acceso a la fila actual se puede hacer bien a través de un índice numérico con la posición del campo o a través de una alias o nombre de campo. El primer número de columna es el 1 y se leen de izquierda a derecha.

Según el tipo de dato almacenado en la columna se usa un método distinto para obtener el dato. Los métodos están sobrecargados para poder usar el número de columna (int) o el alias o nombre de campo (String). De forma general el nombre del método para recuperar un tipo de dato concreto se llama `get`*`NombreTipo`*, por ejemplo *`getBoolean`* o *`getString`*. No solo existen métodos para los tipos básicos de Java, también los hay para otros tipos de datos usados en bases de datos relacionales, como Blob, Time, Date, etc.

```
ResultSet rs = s.executeQuery("SHOW DATABASES”);
if(rs != null) {
    while(rs.next()) {
        out.println(rs.getString("Database") + "<br />");
    }
    rs.close();
}
```

Ejemplo de uso de un ResultSet.

Se puede encontrar mas información y la lista completa de métodos en la documentación oficial de Java:

http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html

Si se desea obtener información sobre el propio ResultSet se puede utilizar un objeto de tipo ResultSetMetadata mediante el método `getMetaData()`. Esta clase permite saber entre otras cosas los nombres de los campos con el método `getColumName()`, el número de campos con `getColumCount()` y otro tipo de información como por ejemplo si el ResultSet es de solo lectura con el método `isReadOnly()`.

En Java existe una evolución mas potente de ResultSet llamada RowSet, que añade funcionalidades de Java Beans (uso de propiedades y notificaciones) y son siempre editables y desplazables, al contrario que algunos ResultSets que dependen del driver.

Puesto que RowSet desciende de ResultSet los métodos con los que se maneja son los mismos.

PHP

PHP suele vienen con drivers compilados que utilizan librerías de cliente para acceder al servidor o bien conexiones ODBC. Por lo tanto se podrían comparar con los drivers tipo 2 y tipo 1 de JDBC.

En PHP hay varias formas de acceder a un servidor SQL:

- Usando ODBC.
- Usando funciones propias de cada servidor (módulos específicos).
- Usando PDO.

El método recomendado en la actualidad es PDO, que es una capa de abstracción para distintos tipos de driver de acceso a bases de datos, y por lo tanto es una mejora de cara al mantenimiento del código de la aplicación. El driver a usar en una conexión viene definido por la cadena de conexión.

A continuación se muestra un ejemplo:

```
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Ejemplo de uso de una conexión PDO en PHP.

Los resultados se obtienen fila a fila o en una tabla completa. Las columnas de la tabla pueden ser accedidas por su índice numérico o bien como un array asociativo usando el nombre del campo.

Técnicas de representación de datos en clientes web

Los clientes web pueden representar la información de muy diversas maneras. De hecho, una de las características de la llamada web 2.0 consiste en mejorar la visualización de la información, hacerla mas atractiva y útil.

Combinando diversos medios y tecnologías se puede lograr que la visualización y manejo de la información sea cómoda y agradable para el usuario.

Visualización de listas y tablas de datos

Para mostrar tablas y listas de resultados se usan estructuras similares en HTML. Si se agregan tecnologías como JavaScript se puede hacer que esos resultados sean dinámicos.

Uno de los métodos mas simples y eficientes para mostrar información proveniente de una base de datos relacional es el uso de tablas, puesto que los resultado ya vienen en un formato similar y tan solo hay que ir creando las filas y rellenando los campos según se procesan los datos de entrada.

Las listas estáticas es otro método bastante usado cuando los datos son susceptibles de tal visualización, como por ejemplo los resultados de una búsqueda o el listado de ítems sin descripción. El uso de listas anidadas también resulta útil cuando se desean mostrar datos con algún tipo de jerarquía.

En el caso de tener los datos tabulados, como por ejemplo como resultado de una consulta a la base de datos, lo que se hace es preparar una estructura (lista, tabla, etc.) y recorrer mediante un bucle todas las filas de datos a la vez que se generan las correspondientes estructuras HTML.

```
ResultSet rs = s.executeQuery("SELECT * FROM libros");
if(rs != null) {
    out.println("<table>");
    out.println("<tr><th>Autor</th><th>Titulo</th></tr>");
    while(rs.next()) {
        out.println("<tr>");
        out.println(<td>rs.getString("autor") + "</td>");
        out.println(<td>rs.getString("titulo") + "</td>");
        out.println("</tr>");
    }
    rs.close();
    out.println("</table>");
}
```

Ejemplo de creación de una tabla a partir de un ResultSet.

En el ejemplo se ve como se genera una tabla en HTML a partir de un ResultSet. Primero se escribe la cabecera de la tabla y luego se generan las distintas filas mediante un blucle. El contenido de las celdas se integra con el código HTML.

Cuando la cantidad de datos a mostrar es muy elevada se pueden utilizar técnicas de paginación de datos para mostrar los resultados por partes. La paginación se puede hacer mediante programación, eligiendo la parte del ResultSet que se desea mostrar, o bien paginar los resultados usando el servidor mediante sentencias SQL.

En el primer caso los datos se guardan como parte de la sesión y se muestran los que el usuario de la aplicación vaya solicitando. En el segundo caso se piden al servidor solamente aquellos datos que vayan a ser visualizados. La consulta SQL depende del motor de bases de datos usado. En MySQL se usa la cláusula LIMIT:

```
SELECT * FROM libros LIMIT 11,10
```

Consulta que muestra los resultados del 11 al 20.

Visualización de registros

Cuando lo que se desea visualizar son registros independientes, se puede organizar la información de varias maneras, las mas usadas consisten en organizar los campos dentro de una tabla, usar una plantilla o bien, si posteriormente se desea modificar algún dato, usar formularios.

La diferencia entre la visualización de registros y la visualización de tablas consiste en que no hay filas que recorrer, ya que se muestra una única fila.

Mecanismos de edición de datos usando clientes web

Para la edición de datos en un cliente web se utilizan formularios. Los formularios se rellenan usando datos obtenidos de una fuente de datos, y se muestran en un formulario. La ventaja del formulario consiste en que hay datos que se pueden ocultar usando campos ocultos y se puede evitar la edición de los campos clave o de los campos que no se desea que se modifiquen.

Una vez mostrados los campos en el formulario el usuario puede modificarlo y enviar el formulario. Al recibir el formulario se modifican los datos mediante los métodos que se citan en el apartado siguiente.

En la actualidad se usan combinaciones de formularios y librerías AJAX para que lo se tengan que recargar los formularios enteros. Junto con el uso de ResultSets dinámicos se desarrollan aplicaciones que realizan modificaciones rápidas y sencillas para el usuario.

Implementación de altas, bajas y modificaciones.

En las herramientas de gestión existen tres tipos de acciones que modifican los datos guardados, y son altas, bajas y modificaciones.

En SQL se corresponden con las siguientes sentencias:

- Altas: crear un nuevo registro. Se corresponde con la sentencia INSERT de SQL.
- Bajas: eliminar un registro. Se corresponde con la sentencia DELETE de SQL.
- Modificaciones: cambiar los datos de un registro. Se corresponde con la sentencia UPDATE de SQL.

El ResultSet de Java proporciona métodos que ayudan a realizar estos cambios de una forma mas sencilla e independiente del motor de bases de datos usado. Primero hay que crear el ResultSet con las opciones apropiadas:

```
stmt = con.createStatement();
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
     ResultSet.CONCUR_UPDATABLE);
```

Creación de un resultset que permite modificar datos.

No todos los drivers soportan insertar o modificar filas en un ResultSet. Un intento de añadir filas o de modificar su contenido en un driver que no soporta esta característica resulta en una excepción del tipo `SQLFeatureNotSupportedException`.

Actualizar datos

Los métodos que proporciona el ResultSet para modificar los datos de la fila actual son similares a los que se utilizan para obtener las columnas, solamente que en vez de empezar por el prefigo "get" comienzan por el prefijo "update", como por ejemplo `updateFloat` o `updateString`.

Una vez que se han actualizado los datos se deben volcar con el método `updateRow()`.

Agregar nuevos datos

Para agregar una fila se una un buffer en el que se anotan los datos que se quieren insertar y luego se consolidan. Los pasos a seguir son los siguientes:

- Mover el cursor a la fila de inserción. La fila de inserción es una fila especial que funciona como buffer para añadir nuevas filas. Esto se hace mediante el método `moveToInsertRow()`.
- Actualizar la fila de inserción con los datos nuevos. Se usan los mismos métodos que en la actualización de datos.
- Llamar al métod `insertRow()`. Este método inserta la fila con los datos en su lugar.
- Opcionalmente llamar al método `moveToCurrentRow()` para posicionar el cursor en la fila en la cual se encontraba antes de insertar la nueva fila. También se puede desplazar el cursor a otra fila a elección del programador.

Eliminar datos

Para eliminar la fila actual se utiliza el método `deleteRow()`. Este método no se puede usar si el cursor está posicionado en la fila de inserción.

Uso de transacciones

Las transacciones se utilizan para asegurar que una serie de modificaciones de datos sigue el esquema del "todo o nada", es decir, si se necesita realizar una serie de operaciones que pueden afectar a la consistencia de los datos si no se ejecutan todas, se

puede indicar al motor de bases de datos que en caso de que una operación no se ejecute "deshaga" las operaciones anteriores. Esto crea el efecto de que varias operaciones se ejecuten como una sola unidad.

Para poder usar transacciones primero hay que deshabilitar el auto-commit, que trata cada sentencia como una transacción individual. Esto se hace mediante el método `setAutoCommit()` de los objetos de tipo conexión.

```
// con es un objeto que representa una conexión activa.
con.setAutoCommit(false);
```

Deshabilitar el auto-commit de una conexión.

Una ves deshabilitado este parámetro habrá que hacer un commit explícito para que las sentencias tengan efecto. Todas las sentencias ejecutadas entre dos commit se tratarán como una sola unidad.

```
// Consolidar las sentencias anteriores.
con.commit();
```

Uso de commit.

En el caso de que se quisiera abortar una transacción habría que hacer un "rollback" de la transacción para anular las sentencias anteriores. Para hacer un "rollback" se usa el método `rollback()` asociado a la conexión usada.

```
// Abortar la transacción actual.
con.rollback();
```

Uso de rollback para echar atrás una transacción.

Si se produce una excepción durante una transacción la forma mas segura de proceder consiste en hacer un rollback abortando la transacción y comenzar de nuevo.

Una vez que no se desean utilizar mas transacciones es deseable volver a habilitar el auto-commit, ya que usa menos bloqueos de la base de datos o conflictos con otros usuarios.

Los bloqueos se usan en la base de datos para evitar conflictos durante las transacciones al obtener lecturas incorrectas de valores que no están totalmente actualizados. Los bloqueos deben mantenerse no mas tiempo del necesario.

Existen varios tipos de bloqueos que actúan ofreciendo diferentes niveles de aislamiento. No todos los gestores de bases de datos soportan todos los tipos de bloqueos.

Existe la posibilidad de crear puntos de restauración (savepoints) dentro de una transacción de forma que se puede indicar a rollback que no aborte toda la transacción, si no que retroceda hasta un punto de restauración concreto.

```
// Se crea un objeto de clase Savepoint.
Savepoint pto1 = con.setSavepoint();
//...
// Volver al punto de restauración.
con.rollback(pto1);
```

Ejemplo de uso de un Savepoint.

Otros orígenes de datos

Java permite el uso de otros orígenes de datos que no usan JDBC. Java puede leer de ficheros del sistema de archivos del servidor, procesar ficheros XML y utilizar bases de datos no-SQL mediante clases desarrolladas por terceros.

En este capítulo se verán los ficheros y los ficheros XML como otros orígenes de datos posibles.

Ficheros

Java usa streams (flujos) para realizar operaciones básicas de entrada y salida. Los ficheros son un tipo de flujo particular. Los streams pueden ser con o sin buffers. Los streams con buffer reducen la cantidad de operaciones de e/s, y por lo tanto mas recomendables para la mayoría de los casos.

Los streams pueden ser de bytes (8 bit) o de caracteres. He aquí las clases:

- `BufferedInputStream`: stream de entrada con buffer para leer bytes.
- `BufferedOutputstream`: stream de salida con buffer para escribir bytes.
- `BufferedReader`: stream de entrada con buffer para leer caracteres.
- `BufferedWriter`: stream de salida con buffer para escribir caracteres.

La plataforma Java soporta tres streams estándar para las aplicaciones de consola: System.in (entrada), System.out (salida) y System.err (salida de errores).

Cuando se programa del lado del servidor estos streams deben ser reemplazados por clases especializadas que lean las peticiones y escriban los resultados, pero si se pueden usar streams que accedan al sistema de archivos del servidor.

```
inputStream = new BufferedReader(new FileReader("fichero_entrada.txt"));
outputStream = new BufferedWriter(new FileWriter("fichero_salida.txt"));
```

Ejemplos de uso de un stream de entrada y otro de salida usando ficheros de texto como origen y destino respectivamente.

Los métodos mas importantes para utilizar ficheros de texto son:

- `read()`: lee un solo carácter.
- `readLine()`: lee una línea de texto. Elimina los caracteres de fin de línea.
- `write()`: escribe un carácter o una cadena de caracteres, dependiendo del tipo de parámetro.
- `newLine()`: escribe el separador de una nueva línea en el fichero.
- `flush()`: vuelca las escrituras pendientes en el stream.

Una vez finalizado el uso del fichero se cierra con el método `close()`.

Ficheros XML

El principal uso de los ficheros XML es el de transportar datos usando un formato potente, conocido y extensible. Los fiicheros XML se pueden usar como orígenes de datos para tareas concretas, como por ejemplo traspaso de datos, ficheros de configuración, etc.

En Java se usa la API JAXP para tratar ficheros XML. Esta API se verá mas adelante, y por lo tanto en este capítulo no se muestra su uso.

Bases de datos no-SQL

Existen otras bases de datos que no siguen el modelo relacional, y cuyo uso está pensado para fines concretos. Algunos ejemplos de bases de datos no SQL son Apache CouchDB que se orienta a la web, MongoDB que se orienta a documentos, Apache Cassandra que es una base de datos distribuída de tipo clave-valor o BigTable de Google entre otras.

Java permite el acceso a varias de estas bases de datos mediante drivers y librerías de terceros, y su forma de uso depende de la base de datos en concreto.

Prácticas

Creación de una base de datos SQL

En esta práctica se creará una base de datos que será utilizada en el resto del manual para la realización de otros ejercicios. La base de datos se llamará “biblioteca” y constará de tres tablas, una para almacenar datos de usuario, otra para almacenar información sobre libros y otra para alamacenar los préstamos de los libros a los usuarios.

Existen múltiples formas de crear la base de datos, a continuación se muestra la forma de realizar la creación desde el cliente de línea de comandos de MySQL.¡, pero se puede usar cualquier otra herramienta de las existentes

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 5.5.16 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database biblioteca;
Query OK, 1 row affected (0.00 sec)

mysql> grant all privileges on biblioteca.* to curso@localhost;
Query OK, 0 rows affected (0.06 sec)

mysql> quit
Bye
```

Secuencia de creación de la base de datos. En negrita los comandos introducidos.

Una vez creada la base de datos y asignados los permisos se procede a la creación de las tablas. La secuencia de comandos SQL es la siguiente:

```
CREATE TABLE `libros` (
  `idlibro` varchar(12) COLLATE latin1_spanish_ci NOT NULL,
  `titulo` varchar(45) COLLATE latin1_spanish_ci DEFAULT NULL,
  `autor` varchar(45) COLLATE latin1_spanish_ci DEFAULT NULL,
  PRIMARY KEY (`idlibro`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_spanish_ci COMMENT='Libros
registrados.';

CREATE TABLE `usuarios` (
  `id` varchar(16) COLLATE latin1_spanish_ci NOT NULL,
  `nombre` varchar(45) COLLATE latin1_spanish_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_spanish_ci COMMENT='Usuarios de
la biblioteca.';

CREATE TABLE `prestamos` (
  `idlibro` varchar(12) COLLATE latin1_spanish_ci NOT NULL,
```

```
  `id` varchar(16) COLLATE latin1_spanish_ci NOT NULL,
  `fechaprestamo` date NOT NULL,
  `fechadevolucion` date DEFAULT NULL,
  PRIMARY KEY (`idlibro`,`id`),
  KEY `idlibro` (`idlibro`),
  KEY `id` (`id`),
  CONSTRAINT `idlibro` FOREIGN KEY (`idlibro`) REFERENCES `libros` (`idlibro`) ON
DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `id` FOREIGN KEY (`id`) REFERENCES `usuarios` (`id`) ON DELETE NO ACTION
ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_spanish_ci;
```

Visualización de información contenida en una base de datos

En este ejercicio se creará un servlet que acceda a la base de datos creada en el ejercicio anterior y muestre el contenido de las tablas. Se asume que está instalado el conector JDBC del motor de bases de datos.

Los pasos a seguir son:

1. Crear un servlet tal y como se ha visto en el ejercicio anterior.
2. Crear un método para consultar la base de datos.
3. Crear un método para visualizar los datos.
4. Programar la lógica del servlet.

El servlet debe tomar como parámetro el nombre de la tabla a mostrar. También se puede crear una página JSP en la que el usuario introduzca el nombre de la tabla a visualizar.

```
/*
 * Servlet que consulta una serie de tablas y devuelve los resultados
 * formateados en HTML.
 */
package org.curso.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.sql.*;

/**
 *
 * @author pedro
 */
@WebServlet(name = "vertabla", urlPatterns = {"/vertabla"})
public class vertabla extends HttpServlet {
```

```
    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet vertabla</title>");
            out.println("</head>");
            out.println("<body>");
            out.println(muestraTablaHTML(request.getParameter("nombretabla")));
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

    /*
     * Devuelve un String con la tabla formateada en HTML.
     *
     * @param String tabla Nombre de la tabla a visualizar.
     * @return String Codigo HTML con el resultado.
     */
    private String muestraTablaHTML(String tabla)
    {
        String resultado = "<table border=1>\n";
        String[] listaCampos = null;
        boolean c = true;

        //...
        try {
            ResultSet rs = consultaBD(tabla);

            if(tabla.equals("usuarios")) {
                 resultado += tablaHTMLusuarios(rs);
            } else if(tabla.equals("libros")) {
                 resultado += tablaHTMLlibros(rs);
            } else if(tabla.equals("prestamos")) {
                 resultado += tablaHTMLprestamos(rs);
            };

            resultado += "</table>";

        } catch (Exception e) {
            // Tratar la excepcion.
```

```
            e.printStackTrace();
        }

        return resultado;
    }

    /*
     * Realiza una consulta a la base de datos para la tabla dada y devuelve
     * los resultados.
     *
     * @param String tabla Nombre de la tabla.
     * @return String Tabla formateada en HTML.
     * @throws SQLException.
     */
    private ResultSet consultaBD(String tabla) throws SQLException
    {
        ResultSet rs = null;
        Statement s = null;
        Connection conexion = null;

        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();

             conexion =
DriverManager.getConnection("jdbc:mysql://localhost/biblioteca?user=curso&password=");
             s = conexion.createStatement();
             rs = s.executeQuery("SELECT * FROM " + tabla);

        } catch (Exception e) {
            // Tratar las excepciones.
            //...
            e.printStackTrace();
        }

        return rs;
    }

    /*
     * Devuelve el contenido de la tabla 'libros' formateado a partir
     * de un ResulSet.
     *
     * @param ResultSet rs Datos de los usuarios.
     * @return String Datos formateados en HTML.
     * @throws SQLException.
     */
    private String tablaHTMLlibros(ResultSet rs) throws SQLException
    {
        String resultado = "<th>Identificador de
libro</th><th>T&iacute;tulo</th><th>Autor</th>\n";

        while(rs.next()) {
            resultado += "<tr>";
            resultado += "<td>" + rs.getString("idlibro") + "</td>";
            resultado += "<td>" + rs.getString("titulo") + "</td>";
            resultado += "<td>" + rs.getString("autor") + "</td>";
            resultado += "</tr>";
        }
```

```
        return resultado;

    }
    /*
     * Devuelve el contenido de la tabla 'usuarios' formateado a partir
     * de un ResulSet.
     *
     * @param ResultSet rs Datos de los usuarios.
     * @return String Datos formateados en HTML.
     * @throws SQLException.
     */
    private String tablaHTMLusuarios(ResultSet rs) throws SQLException
    {
        String resultado = "<th>Identificador de usuario</th><th>Nombre</th>\n";

        while(rs.next()) {
            resultado += "<tr>";
            resultado += "<td>" + rs.getString("id") + "</td>";
            resultado += "<td>" + rs.getString("nombre") + "</td>";
            resultado += "</tr>";
        }

        return resultado;

    }

    /*
     * Devuelve el contenido de la tabla 'prestamos' formateado a partir
     * de un ResulSet.
     *
     * @param ResultSet rs Datos de los usuarios.
     * @return String Datos formateados en HTML.
     * @throws SQLException.
     */
    private String tablaHTMLprestamos(ResultSet rs) throws SQLException
    {
        String resultado = "<th>Identificador de libro</th><th>Identificador de
usuario</th><th>Fecha pr&eacute;stamo</th><th>Fecha devoluci&oacute;n</th>\n";

        while(rs.next()) {
            resultado += "<tr>";
            resultado += "<td>" + rs.getString("idlibro") + "</td>";
            resultado += "<td>" + rs.getString("id") + "</td>";
            resultado += "<td>" + rs.getString("fechaprestamo") + "</td>";
            resultado += "<td>" + rs.getString("fechadevolucion") + "</td>";
            resultado += "</tr>";
        }

        return resultado;

    }

    /**
     * Handles the HTTP
     * <code>GET</code> method.
     *
```

```
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP
     * <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

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

Código fuente de un servlet que muestra los contenidos de una tabla de la base de datos según su nombre.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Visualizador de tablas</title>
    </head>
    <body>
        <form name="frmvertabla" method=""POST" action="vertabla">
              Nombre de la tabla <input type="text" name="nombretabla"> <input
type="submit" value="ver">
        </form>
    </body>
</html>
```

Página JSP que llama al servlet anterior con el parámetro adecuado.

Tema 4

Programación de servicios web.

Objetivos:

- Conocer las arquitecturas de programación orientadas a servicios, así como los mecanismos y protocolos asociados.
- Conocer la implementación y el uso de servicios web.

Introducción a la arquitectura de programación orientadas a servicios (SOA)

SOA se refiere a un tipo de arquitectura basada en servicios usando estándares de Internet y protocolos abiertos. Se fundamenta en los principios de reusabilidad y bajo acoplamiento, de forma que modificar un elemento de la arquitectura no tenga un gran impacto en el resto y también busca facilitar la integración e interacción entre servicios propios y de terceros.

SOA define como integrar diversas aplicaciones dispares que usan distintas plataformas, mas que un API, SOA define el interfaz en términos de protocolos y funcionalidades. Se trata de utilizar los servicios independientemente de su implementación.

Una arquitectura SOA refleja la estructura de la empresa.

En una arquitectura SOA pueden intervenir varios elementos:

- Servicio: programa software que proporciona una funcionalidad de negocio.
- Broker: es un elemento que ayuda a localizar servicios.
- Locator: es un servicio que actúa como un registro de los servicios disponibles y ayuda en la localización de un servicio concreto.
- Proveedor (provider): es la entidad que proporciona el servicio.
- Consumidor (requester): es la entidad que demanda el servicio.

SOA es una aproximación al diseño de aplicaciones complejas basada en la identificación y definición de los servicios que debe ofrecer y la interacción entre dichos servicios.

Las interfaces son muy importantes, ya que la idea base es desarrollar el sistema a partir de estas interfaces, por lo que su definición ha de ser muy rigurosa.

Los servicios web (web services o abreviado WS) pueden usarse para implementar una arquitectura SOA. Estos servicios web pueden ser desarrollos específicos o envolturas de sistemas heredados para que puedan usar los servicios de red.

La definición de servicio web según el W3C es: “Un Servicio Web es una aplicación software identificada por un URI, cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML. Los Servicios Web hacen posible la interacción

entre “agentes” de software utilizando XML intercambiados mediante protocolos de Internet.”

Por lo tanto las bases de un servicio web son: interoperabilidad, uso de estándares abiertos y acoplamiento mínimo.

Las organizaciones encargadas de definir la arquitectura y estándares para los servicios web son OASIS y el W3C.

En Java la implementación de servicios web se puede hacer con JAX-WS, JAX-RS o con otras librerías de terceros como AXIS2 (http://axis.apache.org/axis2/java/core/). Oracle utiliza por defecto JAX-WS y JAX-RS en vez de AXIS2.

JAX-WS se utiliza para servicios web “grandes”, que usan SOAP para el intercambio de mensajes y WSDL para la definición de los interfaces, donde la integración requiere el uso de conceptos avanzados, y JAX-WS se hace para entornos mas pequeños donde la integración sobre la web es mas fácil se utiliza JAX-RS.

Estándares usados en arquitecturas SOA

La arquitectura SOA se basa en estándares. Al ser una arquitectura distribuida y heterogénea necesita estándares para la integración de todos sus componentes. A continuación se describen los mas importantes.

HTML

El lenguaje HTML ya se ha visto anteriormente. Es un lenguaje de marcas que usan los clientes web para presentar la información.

XML

XML es un lenguaje de marcas pensado fundamentalmente para el intercambio de información.

SOA hace un uso intensivo de XML para describir los servicios y los contenedores de los datos que usar y ofrece dicho servicio, así como en el paso de mensajes entre componentes.

En Java existen varias API específicas para el procesado de XML. Una de las mas completas u flexibles es JAXP. Permite elegir entre varios parsers y es modular y ampliable mediante plugins.

Se puede elegir entre varias APIs: SAX, DOM y StAX entre otras. SAX está orientada a eventos y es muy potente, pero mas difícil de usar, en cambio DOM es mas sencillo pero consume mas recursos, al almacenar todo el documento en memoria puesto que requiere conocer la estructura completa del fichero XML. La API StAX es mas simple de usar que SAX y consume menos recursos que DOM, por lo que se puede decir que es un punto intermedio entre ambas API. Se define en el paquete `javax.xml.stream`. Es una API mas moderna y se basa en un parser de flujos bidireccional de tipo “pull” que se utiliza mediante eventos.

Todas estas API se enmarcan en las librerías JAXP, que es la parte de Java que se encarga del procesado XML.

JSON

Es un formato de texto ligero para el intercambio de datos basado en un subconjunto del lenguaje JavaScript. Es entendible directamente por humanos y fácilmente interpretable y generable por máquinas, lo cual lo posiciona como un sustituto de XML en algunos casos. Su estructura lo hace especialmente fácil de interpretar para programadores familiarizados con la sintaxis de C, que comparten muchos otros programas como C++, Perl, Python o Java.

Java no soporta directamente JSON, pero existen librerías de libre uso para soportarlo. Hay mas información disponible en http://json.org/java/.

SOAP

SOAP (Simple Object Access Protocol) es un protocolo de intercambio de información en XML en red que se usa principalmente en servicios web usando HTTP como transporte. Deriva del protocolo XML-RPC, y actualmente se encuentra auspiciado por el W3C. La versión actual es la 1.2.

Este protocolo forma parte de la pila de protocolos que usan los servicios web. Los servicios web usan mensajes SOAP para realizar consultas y devolver resultados.

WSDL

WSDL (Web Services Description Language) es una información escrita en XML que proporciona información acerca de un servicio web, permitiendo conocer la interfaz pública del servicio web. La versión actual es la 2.0.

La información que proporciona WSDL acerca de un servicio web permite conocer de forma abstracta la interfaz del servicio web, es decir, la forma de comunicarse con el servicio web, los requisitos del protocolo, el formato de los datos, etc. La descripción del servicio web permite al cliente conocer los detalles para comunicarse con el mismo.

REST

REST (REpresentational State Transfer) es un modelo arquitectura de software distribuida que se aplica a servicios web y mas simple que SOAP y WSDL. Es adecuado para escenarios básicos donde la integración sea sencilla, en contraposición de los escenarios “grandes”. A las implementaciones que siguen los principios de REST se las denomina “*RESTful*”.

Esta arquitectura se centra en al concepto de recursos, cuyo estado reside en el servidor. Los clientes acceden al recurso mediante representaciones del mismo de alguna de las siguientes formas:

- Transferidas mediante HTTP.

- Mediante formatos de representación (HTML, XML, JSON, etc.)
- Mediante tipos MIME especificados en las cabeceras.

Los recursos se identifican mediante URI, y se basa el uso de este tipo de servicios web en un conjunto de operaciones sobre estor URI con una semántica predefinida.

En Java se usa la API JAX-RS para la implementación de servicios web con REST.

UDDI

UDDI (Universal Description, Discovery and Integration) es un servicio de registro basado en XML que ayuda a la localización de servicios web. El servicio recibe mensajes SOAP y devuelve información sobre el WSDL del servicio.

En 2006 Microsoft, IBM y SAP dejaron de dar soporte al registro en Internet, y actualmente se usa de forma residual dentro de algunas empresas.

Protocolos de intercambio de información en XML

Los mensajes SOAP constan de tres partes: sobre (envelope), cabecera (header) y cuerpo (body).

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:hello xmlns:ns2="http://curso.me.org/">
            <name>mundo</name>
        </ns2:hello>
    </S:Body>
</S:Envelope>
```

Ejemplo de petición SOAP.

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:helloResponse xmlns:ns2="http://curso.me.org/">
            <return>Hola mundo !</return>
        </ns2:helloResponse>
    </S:Body>
</S:Envelope>
```

Ejemplo de respuesta SOAP.

(SOAP, XML-RPC.

Estructura de los mensajes.)

Consejo: si se utiliza Glassfish como servidor se pude probar el servicio directamente sin escribir un cliente, basta con acceder a la url del servicio y añadir "`?Tester`". En la respuesta al test se pueden ver los mensajes SOAP en XML.

Descripción del servicio: WSDL

La descripción del servicio es muy importante, ya que permite a los clientes saber cómo comunicarse con el servicio web. La descripción del servicio se organiza como un conjunto de terminaciones de red (endpoints) o puertos (ports) dependiendo de la versión.

La información ofrecida por WSDL se escribe en XML y tiene, entre otros, los siguientes elementos:

- `types`: describe los datos, bien con un esquema referenciado o bien incrustado.
- `message`: contiene información acerca de una operación, normalmente hay un message por operación. En WSDL 2.0 no se utilizan al usar esquemas XML para definir los cuerpos de las entradas y salidas.
- `portType`: define un servicio web, las operaciones que se pueden realizar y los mensajes para esas operaciones. En WSDL 2.0 se llaman `interfaces`.
- `operation`: define las acciones SOAP y la forma en que se codifican.
- `binding`: especifica el interfaz y define parámetros y operaciones de SOAP.
- `service`: contiene el conjunto de funciones que se muestran a los protocolos web.
- `port`: define el punto de conexión o la dirección del servicio web. Normalmente es una url. En WSDL 2.0 se llaman `endpoint`.

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://curso.me.org/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://curso.me.org/" name="srv1">
<types>
     <xsd:schema>
     <xsd:import namespace="http://curso.me.org/"
           schemaLocation="http://localhost:8080/ws1/srv1?xsd=1"/>
     </xsd:schema>
</types>
<message name="hello">
     <part name="parameters" element="tns:hello"/>
</message>
<message name="helloResponse">
     <part name="parameters" element="tns:helloResponse"/>
```

```
</message>
<portType name="srv1">
	<operation name="hello">
		<input wsam:Action="http://curso.me.org/srv1/helloRequest"
			message="tns:hello"/>
		<output wsam:Action="http://curso.me.org/srv1/helloResponse"
			message="tns:helloResponse"/>
	</operation>
</portType>
<binding name="srv1PortBinding" type="tns:srv1">
	<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
	<operation name="hello">
		<soap:operation soapAction=""/>
		<input>
			<soap:body use="literal"/>
		</input>
		<output>
			<soap:body use="literal"/>
		</output>
	</operation>
</binding>
<service name="srv1">
	<port name="srv1Port" binding="tns:srv1PortBinding">
		<soap:address location="http://localhost:8080/ws1/srv1"/>
	</port>
</service>
</definitions>
```

Ejemplo de fichero WSDL.

> **Consejo**: se puede obtener el fichero WSDL de un servicio web añadiendo "`?wsdl`" a su url.

Interfaz del servicio web a partir de su descripción

El fichero WSDL detalla perfectamente las características del servicio web. Es a partir de esta descripción que los clientes sabrán cómo han de conectar con el servicio web, cuales son sus parámetros y cuales las respuestas. Esta descripción define la interfaz de comunicación en un sistema distribuido, por lo que se asume que los clientes que pueden acceder al servicio son heterogéneos pero con una capa de middleware que homogeneiza el comportamiento de todos los elementos. De ahí la importancia de la descripción del servicio a la hora de definir su interfaz.

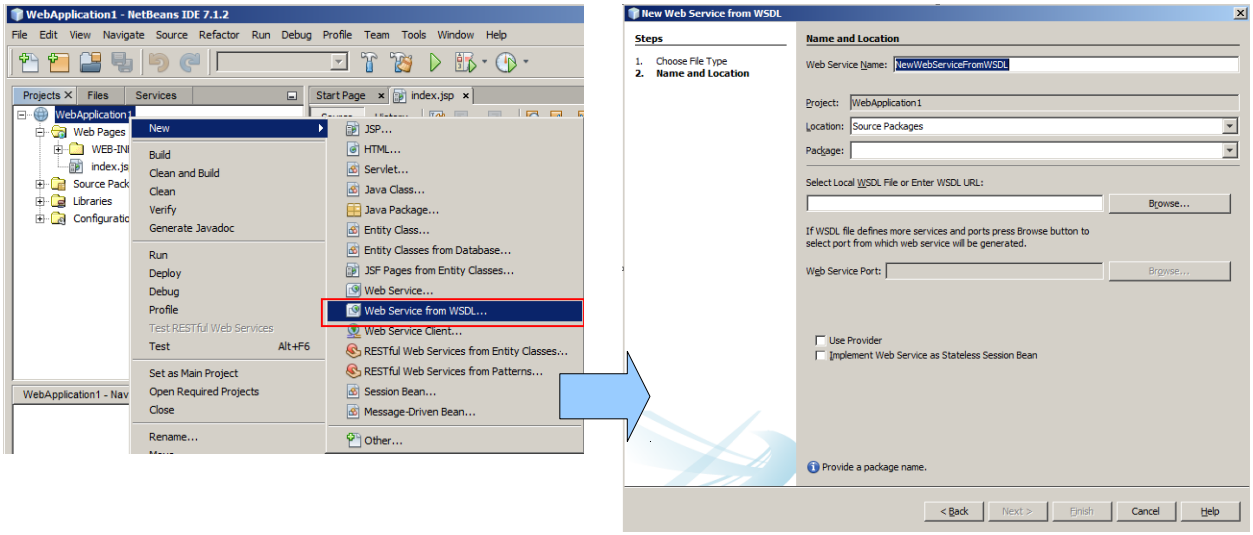
Tan importante es la descripción del servicio que hay herramientas de desarrollo que permiten crear la base de un nuevo servicio web a partir de un fichero WSDL.

En los desarrollos de aplicaciones SOA normalmente se crean primero las descripciones del servicio y posteriormente se codifican los programas. Esto es así porque es importante conocer la estructura de los servicios a priori, ya que hay que verificar que cumplen con las expectativas y requisitos del cliente, y porque además varios equipos de desarrollo pueden trabajar en paralelo. Diseñar la interfaz del servicio a partir de su descripción hace que se dependa menos de las herramientas de generación de ficheros

WSDL a partir de código, que pueden hacer referencia tipos de datos o procedimientos propios de la plataforma en la que se desarrolló el código.

La interfaz de un servicio web la componen los datos que se intercambian y los servicios que se publican. Ambos tipos de información están contenidos en la descripción del servicio, y se pueden obtener fácilmente con tan solo mirar las secciones types o las secciones operation.

Un ejemplo de uso de la descripción de la interfaz a partir de la descripción lo tenemos en la herramienta Netbeans con la opción de crear un nuevo servicio web a partir de un fichero WSDL.



Creación de un servicio web a partir de un fichero WSDL.

Este tipo de herramientas facilitan la creación de prototipos y de los “esqueletos” de las aplicaciones cliente, como se verá mas adelante en la práctica.

Uso de servicios web mediante un cliente

Cuando un programador debe implementar un servicio web conocer la descripción del servicio web a partir del fichero WSDL le permite crear parte de la estructura necesaria para desarrollar la parte de la aplicación que se comunica con el servicio web.

Con la sección “types” se puede saber el tipo de dato que se se envía o recibe. Esto tiene la ventaja de ser independiente del lenguaje utilizado tanto en el desarrollo del servicio web como de los clientes, debiendo traducir los tipos genéricos a tipos de datos del lenguaje de programación usado y viceversa.

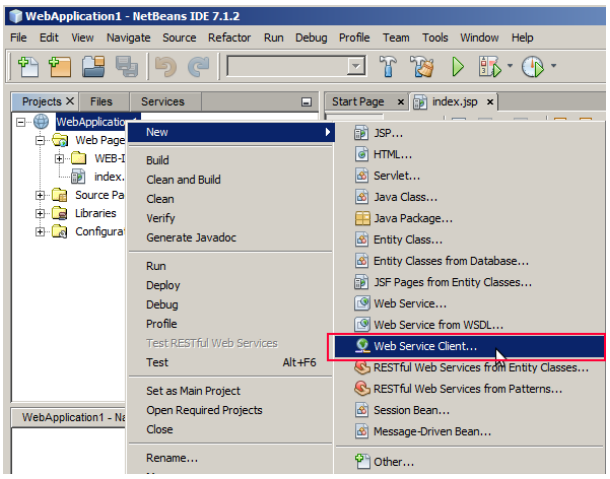
Las secciones “operation” se podrían ver como los métodos que soporta el servicio web.

Por lo tanto, uniendo todas las secciones de un fichero WSDL se pueden obtener los prototipos de las funciones que puede implementar un cliente del servicio web.

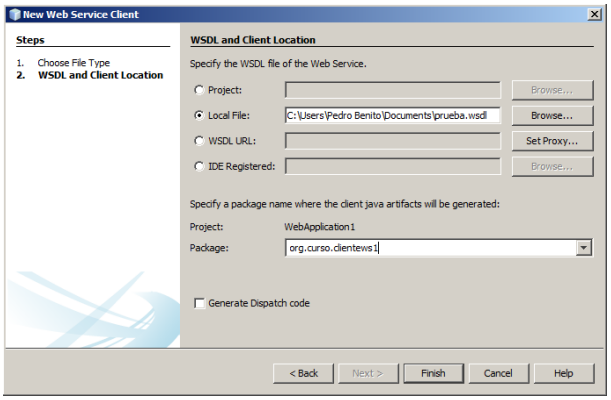
Desde el entorno de desarrollo Netbeans se puede crear un cliente de servicio web a partir de un fichero WSDL siguiendo los siguientes pasos:

1 Abrir un proyecto o crear uno nuevo.

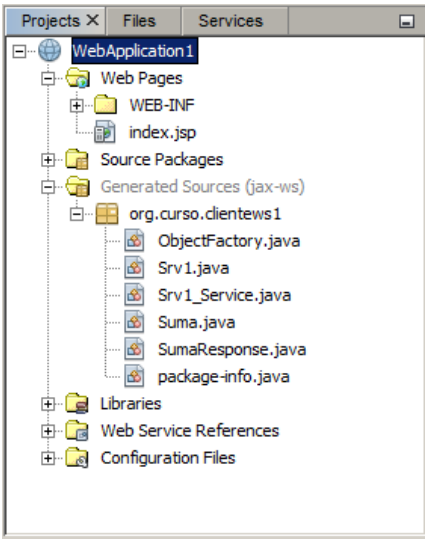
2 En la aplicación añadir un nuevo cliente de servicios web.



3 Elegir el origen del fichero WSDL.



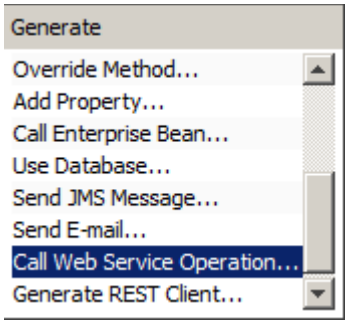
4 Se crean las clases necesarias de forma automática.



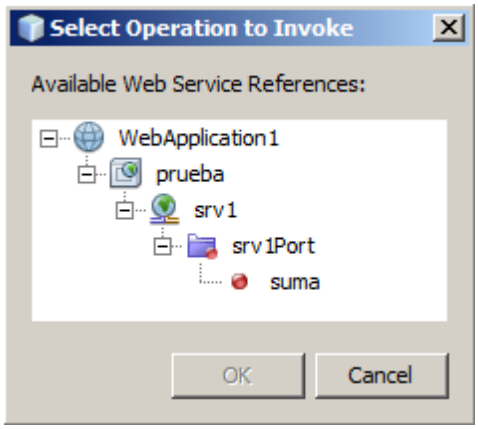
5 Ahora los servicios web pueden ser usados por la aplicación, por ejemplo desde un sevlet o desde una página JSP.

5.1 Para usar los servicios desde un servlet se crea un servlet nuevo o se usa uno ya existente.

En la ventana código se selecciona la opción “Insert code” del menú contextual. Entre las operaciones que aparecen se debe seleccionar “Call Web Service operation”.



En la nueva ventana que aparece se selecciona la operación deseada.



El editor inserta el código automáticamente.

```
private Integer suma(int op1, int op2) {
    org.curso.clientews1.Srv1 port = service.getSrv1Port();
    return port.suma(op1, op2);
}
```

Código generado por Netneans para llamar a un servicio web desde un servlet.

Ahora el servicio web puede ser llamado invocando el nuevo método.

5.2 Para usar los servicios web desde una página JSP basta con desplegar la rama “Web Services Reference” del panel de proyectos y arrastrar la función deseada al lugar de la página JSP deseado. El editor se encarga de generar el código base para que luego pueda ser ajustado.

```
<%
    try {
     org.curso.clientews1.Srv1_Service service = new
org.curso.clientews1.Srv1_Service();
     org.curso.clientews1.Srv1 port = service.getSrv1Port();
      // TODO initialize WS operation arguments here
     int op1 = 0;
     int op2 = 0;
     // TODO process result here
     java.lang.Integer result = port.suma(op1, op2);
     out.println("Result = "+result);
    } catch (Exception ex) {
     // TODO handle custom exceptions here
    }
    %>
```

Código generado por Netbeans para usar un servicio web desde una página JSP.

Ejemplo de uso de servicios web: Google Maps

Google maps se puede integrar en otras aplicaciones mediante servicios web. En la dirección https://developers.google.com/maps/documentation/webservices/?hl=es se encuentra mas información al respecto.

Prácticas

Creación y uso de un servicio web simple que se comunique con una base de datos

En esta práctica se creará un servicio web que consulte una base de datos y que devuelva los resultados a un cliente. Como base de datos se usará la base de datos de biblioteca creada anteriormente. El servicio web consistirá en devolver, a partir de un identificador de libro, si está o no prestado.

Para probar el servicio se utilizarán los siguientes elementos:

- Una página JSP con un formulario para que el usuario introduzca el código del libro.
- Un servlet que actúe como cliente del servicio web y muestre la respuesta.
- Un servicio web que consulte la base de datos e informe de si el título está prestado o no.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Consulta de pr&eacute;stamo</title>
    </head>
    <body>
        <form name="consultaprestamo" action="consultaprestamo" method="POST">
            Introduzca el c&oacute;digo del libro <input type="text"
name="codigolibro"> <input type="submit" value="Comprobar">
        </form>
    </body>
</html>
```

Página JSP con el formulario de entrada de datos.

```
/*
 * Servlet que consulta un servicio web para saber si el libro que se indica
 * en el parametro esta o no prestado.
 */
package org.curso.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author pedro
 */
@WebServlet(name = "ConsultaPrestamo", urlPatterns = {"/consultaprestamo"})
public class ConsultaPrestamo extends HttpServlet {
```

```
    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            /*
             * TODO output your page here. You may use following sample code.
             */
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ConsultaPrestamo</title>");
            out.println("</head>");
            out.println("<body>");
            if(consultaPrestado(request.getParameter("codigolibro"))) {
                out.println("El libro est&aacute; prestado.");
            } else {
                out.println("El libro est&aacute; disponible.");
            }
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

//...

    private static boolean consultaPrestado(java.lang.String codigoLibro) {
        org.curso.wscli.WsConsulta_Service service = new
org.curso.wscli.WsConsulta_Service();
        org.curso.wscli.WsConsulta port = service.getWsConsultaPort();
        return port.consultaPrestado(codigoLibro);
    }
}
```

Servlet que actúa como cliente del servicio web.

```
package org.curso.ws;

import java.sql.*;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
```

```
 *
 * @author pedro
 */
@WebService(serviceName = "wsConsulta")
public class wsConsulta {

    /**
     * Comprueba si un libro esta prestado.
     *
     * @return boolean true si esta prestado.
     */
    @WebMethod(operationName = "consultaPrestado")
    public boolean consultaPrestado(@WebParam(name = "codigoLibro") String idLibro) {
        boolean resultado = false;

        try {
            ResultSet rs =  consultaBDibroPrestado(idLibro);

            if(rs.next() != false) {
                resultado = true;
            }

        } catch (SQLException ex) {
            ex.printStackTrace();
        }

        return resultado;
    }

    private ResultSet consultaBDibroPrestado(String idlibro) throws SQLException
    {
        ResultSet rs = null;
        Statement s = null;
        Connection conexion = null;

        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();

             conexion =
DriverManager.getConnection("jdbc:mysql://localhost/biblioteca?user=curso&password=");
             s = conexion.createStatement();
             rs = s.executeQuery("SELECT * FROM prestamos WHERE idlibro='" + idlibro
+"' AND fechadevolucion is NULL");

        } catch (Exception e) {
            // Tratar las excepciones.
            //...
            e.printStackTrace();
        }

        return rs;
    }

}
```

Servicio web que consulta la base de datos.

Creación de un servicio web a partir de un fichero WSDL

El ejercicio consiste en, dado un fichero WSDL, crear un nuevo servicio web e identificar cómo los distintos parámetros del fichero se convierten en métodos y variables Java. Para ello se usará el entorno Netbeans que generará el código a partir del fichero.

Fichero WSDL a partir del cual se generará un servicio web:

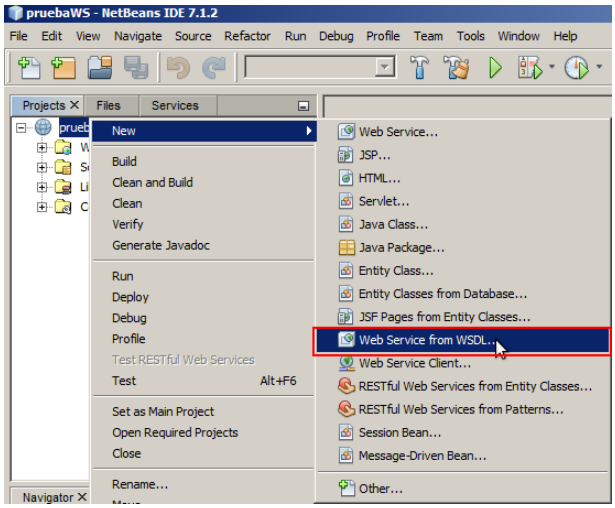
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://srv1.curso.org/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://srv1.curso.org/" name="srv1">
  <types>
    <xs:schema xmlns:tns="http://srv1.curso.org/"
          xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
          targetNamespace="http://srv1.curso.org/">
      <xs:element name="suma" type="tns:suma"/>
      <xs:element name="sumaResponse" type="tns:sumaResponse"/>
      <xs:complexType name="suma">
        <xs:sequence>
          <xs:element name="op1" type="xs:int"/>
          <xs:element name="op2" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="sumaResponse">
        <xs:sequence>
          <xs:element name="return" type="xs:int" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </types>
  <message name="suma">
    <part name="parameters" element="tns:suma"/>
  </message>
  <message name="sumaResponse">
    <part name="parameters" element="tns:sumaResponse"/>
  </message>
  <portType name="srv1">
    <operation name="suma">
      <input wsam:Action="http://srv1.curso.org/srv1/sumaRequest" message="tns:suma"/>
      <output wsam:Action="http://srv1.curso.org/srv1/sumaResponse"
         message="tns:sumaResponse"/>
    </operation>
  </portType>
  <binding name="srv1PortBinding" type="tns:srv1">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="suma">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
```

```
    </operation>
  </binding>
  <service name="srv1">
    <port name="srv1Port" binding="tns:srv1PortBinding">
      <soap:address location="http://localhost:8080/app1/srv1"/>
    </port>
  </service>
</definitions>
```

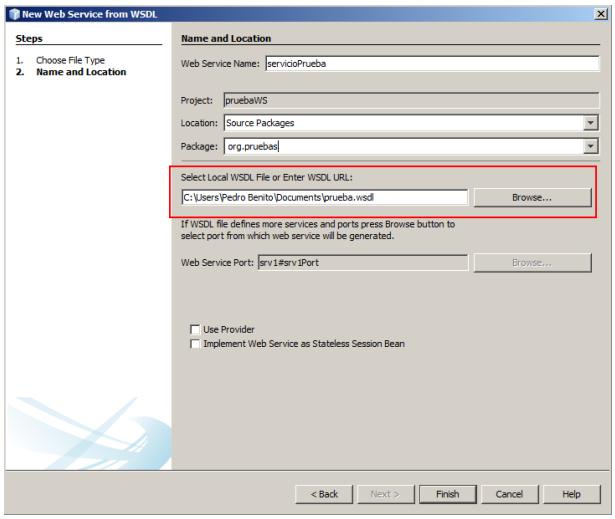
Los pasos a seguir ya han sido descritos con anterioridad:

1 Crear un nuevo proyecto tipo “Aplicación Web Java”.

2 Abrir el menú contextual del proyecto y seleccionar “Nuevo → Servicio web a partir de WSDL”.



3 Abrir el archivo WSDL.



4 El editor genera el código para el servicio web.

```
package org.pruebas;

import javax.jws.WebService;

/**
 *
 * @author Pedro Benito
 */
@WebService(serviceName = "srv1", portName = "srv1Port", endpointInterface =
"org.curso.srv1.Srv1", targetNamespace = "http://srv1.curso.org/", wsdlLocation =
"WEB-INF/wsdl/servicioPrueba/prueba.wsdl")
public class servicioPrueba {

    public java.lang.Integer suma(int op1, int op2) {
        //TODO implement this method
        throw new UnsupportedOperationException("Not implemented yet.");
    }

}
```

5 Ya se puede realizar la implementación del método o métodos. Hay que acordarse de devolver un valor, puesto que por defecto lanza una excepción.

Tema 5

Generación dinámica de páginas web interactivas.

Objetivos:

- Conocer las técnicas de generación de páginas web interactivas.
- Conocer las diferencias entre la generación de contenido en el servidor y en el cliente.
- Usar las tecnologías actuales para la generación de contenido de forma dinámica, así como la modificación de la estructura de una página web.

Introducción al procesamiento del lado del cliente

Como ya se ha mencionado, en las arquitecturas distribuidas el usuario interactúa con el cliente y éste se apoya en el servidor para realizar ciertas tareas. Según el tipo de cliente se puede hablar de clientes ligeros, pesados e híbridos.

Los clientes ligeros (thin client) actúan como meros intermediarios entre el usuario y el servidor, suelen necesitar pocos recursos hardware y no implementan ninguna parte de la lógica de la aplicación. Un ejemplo de cliente ligero son los navegadores web.

Los clientes pesados (fat client) implementan gran parte de la lógica de la aplicación, procesan los datos antes de enviarlos al servidor y tienen mayores requisitos de hardware,.

Los clientes híbridos suelen utilizarse en escenarios en los que la lógica de la aplicación está repartida entre el cliente y el servidor. Un ejemplo de cliente híbrido es un cliente que accede a una base de datos y ejecuta procedimientos almacenados.

A veces el procesamiento del lado del cliente es deseable, e incluso necesario. Los clientes de aplicaciones enriquecidas web 2.0 necesitan que el navegador realice ciertas operaciones que no se limitan al mero renderizado de los datos ya procesados por el servidor.

Cada tipo de cliente tiene sus ventajas y sus inconvenientes:

	Ventajas	Inconvenientes
Cliente ligero	Necesita pocos recursos. Multiplataforma (web). Implementación sencilla.	Mayor complejidad del servidor.
Cliente pesado	Menor complejidad del servidor.	Mayor uso de recursos del cliente. Implementación mas compleja.

Los clientes pueden realizar operaciones relacionadas con la lógica de negocio pero también necesitan realizar operaciones relacionadas con le interacción del usuario.

Cuanto mas completa sea la interacción con el usuario mayor complejidad tendrá el cliente.

Los desarrollos web utilizan cada vez mas tecnologías de cliente para proporcionar al usuario la ergonomía y la experiencia de uso que se demanda. Esto provoca que los sencillos cliente web de hace unos años ahora se han dotado de gran cantidad de tecnologías que requieren mayores recursos hardware. Aún así se pueden considerar clientes ligeros ya que la parte mas importante del procesamiento se realiza del lado del servidor.

Tecnologías relacionadas con la generación dinámica de páginas web interactivas

En las aplicaciones web la mayor ventaja consiste en la universalidad del acceso, ya que al basarse en estándares y residir la lógica de negocio en las aplicaciones de servidor los clientes solamente tienen que implementar las tecnologías de renderizado y de acceso al servidor.

Las páginas web dinámicas pueden ser generadas por el servidor, el cliente o ambos. El servidor puede descargar al cliente de la parte de generación del contenido de muy diversas maneras, no solo al procesar los datos si no también al utilizar controles que ayudan a la presentación de los datos, como por ejemplo JavaServer Faces o los controles de servidor utilizados en ASP.net para crear interfaces de usuario.

En cambio hay otras tecnologías que se implementan íntegramente en el lado del cliente web, como por ejemplo JavaScript y CSS.

JavaScript es un lenguaje de programación que está implementado en la mayoría de navegadores, y que permite crear páginas web dinámicas. Es orientado a objetos, interpretado y su sintaxis es similar a la de C. Se puede implementar en el cliente pero también en el servidor.

Los programas hechos en JavaScript se integran en la página web incrustando el código dentro de los tags <*script*> o bien reverenciándolo mediante una URL.

JavaScript permite realizar muchos tipos de tareas en los navegadores: validar entradas, ocultar o mostrar información, realizar efectos visuales, usar el DOM para manipular el documento, e incluso comunicarse asíncronamente con el servidor usando AJAX.

En la actualizad el uso de JavaScript está muy extendido y lo soportan la práctica totalidad de los navegadores. El uso de JavaScript en páginas web interactivas es tan importante que los desarrolladores de navegadores invierten gran cantidad de tiempo en mejorar el motor de interpretación de este lenguaje, anunciando las mejoras en este aspecto en cada nueva versión.

Para evitar pagar derechos Microsoft ha realizado su propia versión de JavaScript llamada Jscript.

La versión oficial de JavaScript se denomina ECMAScript y está estandarizado con la norma ISO 16262.

```
/* Mostrar u ocultar un elemento. */
function MuestraOculta(id)
{
    var elemento;

    elemento = document.getElementById(id);

    if(elemento.style.visibility == 'visible') {
        elemento.style.visibility = 'hidden';
    } else {
        elemento.style.visibility = 'visible';
    }
}
```

Ejemplo de código JavaScript.

CSS es un lenguaje para describir el aspecto de un documento. Los navegadores usan CSS para poder separar el contenido de la presentación en las páginas web. Permite que muchas páginas puedan tener un mismo formato, y reduce el uso de tablas.

Las hojas de estilo pueden incrustarse en el documento mediante las etiquetas <*style*> o bien cargar se usando una URL.

```
body {
    font-family: Verdana,Arial,Helvetica,sans-serif;
    font-size: 12px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    color: black;
    text-decoration: none;
}

a {
    color: #0000ff;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}

.titulo {
    font-size: 18px;
    font-weight: bold;
}

.titulo2 {
    font-size: 14px;
    font-weight: bold;
}
```

Ejemplo de un fichero de hoja de estilo CSS.

Mezclando varias de estas tecnologías surgen otras nuevas, como por ejemplo AJAX. Esta tecnología utiliza JavaScript y XML para realizar llamadas asíncronas al servidor y dotar a la página web de mayor interacción y dinamismo. Se verá con mayor detalle mas adelante.

Existen librerías que facilitan el uso de estas tecnologías, como por ejemplo la librerías jQuery (http://jquery.com/) que simplifica el uso de JavaScript en los navegadores a través de un framework.

jQuery está diseñado para realizar funciones como seleccionar elementos del DOM, crear animaciones, manejar eventos y crear aplicaciones que usen AJAX. Existe un subproyecto llamado jQuery-ui que ayuda a desarrollar aplicaciones con interfaces gráficos mas potentes y uniformes abstrayendo al programador de las funciones de bajo nivel.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery demo</title>
</head>
<body>
  <a href="http://jquery.com/">jQuery</a>
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js"></script>
  <script>
    $(document).ready(function(){
      $("a").click(function(event){
        alert("As you can see, the link no longer took you to jquery.com");
        event.preventDefault();
      });
    });
  </script>
</body>
</html>
```

Ejemplo de uso de jQuery en una página web (Fuente: jquery.com).

Verificación de la información en el cliente

Una de las funciones mas interesantes del procesado del lado del servidor es la de realizar una primera verificación de los datos que ha introducido el usuario antes de enviarlos al servidor.

Usando JavaScript se pueden controlar ciertos aspectos relativos a los datos introducidos por el usuario cuando éste pulsa el botón “Enviar” o cuando abandona un control, por ejemplo para saber si el campo está vacío o si el dato tiene un formato concreto.

Estas verificaciones no sustituyen a las que realice el servidor, si no que las complementa al no tener que acceder a la red para realizar algunas comprobaciones sobre los datos introducidos.

Se utiliza la gestión de eventos de JavaScript para ejecutar las acciones de verificación sobre los datos introducidos por el usuario.

Los eventos de JavaScript se pueden asociar a distintos elementos de una página web especificándolos en su tag. Son eventos que se producen en el cliente (navegador) como

resultado de la interacción del usuario mediante el ratón o el teclado o bajo otras circunstancias como por ejemplo la carga de la página o el envío de un formulario.

Los eventos JavaScript son:

- onClick: hacer clic con el ratón sobre el elemento.
- onDblClick: hacer doble clic sobre el elemento.
- onMouseDown: al pulsar el botón.
- onMouseUp: cuando se deja de pulsar el botón.
- onContextMenu: al pulsar el botón derecho del ratón.
- onMouseOver: cuando el puntero del ratón pasa sobre el elemento.
- onMouseOut: cuando el ratón deja de estar sobre el elemento.
- onKeyPress: cuando se pulsa y se suelta una tecla.
- onKeyDown: cuando se pulsa una tecla.
- onKeyUp: cuando se deja de pulsar una tecla.
- onLoad: se produce cuando la página se ha cargado completamente.
- onUnload: cuando se carga una nueva página o se vuelve a recargar la actual.
- onAbort: cuando se cancela la carga de una página.
- onSubmit: se produce al enviar un formulario.
- onReset: cuando se resetea el formulario.
- onChange: cuando cambia el contenido de un campo del formulario.
- onFocus: cuando el campo del formulario recibe el foco.
- onBlur: cuando el campo del formulario pierde el foco.
- onSelect: cuando se selecciona contenido de un campo.
- onResize: cuando se redimensiona una ventana.
- onScroll: cuando se desplaza el contenido de una ventana.

Obtención remota de información. AJAX

Hasta hace un tiempo si se deseaba actualizar el contenido de una página web ésta debía ser cargada de nuevo en su totalidad. El uso de JavaScript y XML han hecho posible que se pueden realizar llamadas asíncronas al servidor para actualizar solamente una parte de la página web mediante llamadas al servidor. Esta agrupación de tecnologías se denomina AJAX (**A**synchronous **J**avaScript **A**nd **X**ML).

El uso de AJAX tiene considerables ventajas:

- Ahorro de ancho de banda.

- Mayor rapidez en la respuesta, al cargar solamente las modificaciones necesarias y no toda la página web.
- Mejor experiencia de usuario, al aproximarse a la forma de actuar de una aplicación de escritorio.

Aunque también tiene algunos inconvenientes:

- El desarrollo de aplicaciones con AJAX es mas complejo.
- Las aplicaciones con AJAX no quedan grabadas en el historial del navegador, por lo que no siempre funciona el botón “volver atrás”. Esto se resuelve en los navegadores que soportan HTML5.
- Los motores de búsqueda no interpretan JavaScript.
- No funcionan sin conexión (offline).
- Hacen un uso mas intensivo del servidor.

AJAX se basa en usar JavaScript para llamar a un objero de tipo XMLHttpRequest que realiza la petición, y posteriormente se vuelve a usar JavaScript para modificar la página a través del DOM.

El objeto ActiveX XMLHTTP fue creado por Microsoft para su navegador Internet Explorer 5, y posteriormente fue adoptado por otros navegadores como el objeto XMLHttpRequest en JavaScript. Aunque la versión original del control ActiveX de Microsoft aún se implementa, desde la versión 7 de su navegador también se puede utilizar el objeto XMLHttpRequest.

Los métodos que soporta el objeto XMLHttpRequest son:

- `open`: abre una conexión a la URL indicada usando GET, POST, etc. de forma síncrona o asíncrona. Se puede indicar usuario y contrasela. La URL puede ser http o https.
- `send`: envía el contenido indicado como parámetro.
- `abort`: detiene una petición en curso.
- `setRequestHeader`: indica un valor para una de las cabeceras de la petición.
- `getResponseHeader`: devuelve el valor de la cabecera indicada.
- `getAllResponseHeaders`: devuelve todas las cabeceras de la respuesta.

Propiedades del objeto XMLHttpRequest:

- `onreadystatechange`: contiene el nombre de la función que se ejecutará cada vez que cambie el estado de la petición.
- `readyState`: estado de la conexión de 0 (nada) a 4 (completa).
- `responseText`: contenido de la respuesta.
- `responseXML`: contenido de la respuesta en XML.
- `status`: código de estado que devuelve el servidor. 200 significa correcto.

- `statusText`: mensaje que el servidor ha enviado en la respuesta.

AJAX se implementa en una página web como un script que responde a eventos. En este ejemplo se puede ver una capa y un botón. Se asocia un evento al botón.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Prueba de AJAX</title>
        <script></script>
    </head>
    <body>
        <div id="capa1">Sin datos.</div>
        <button type="button" onclick="cargaDatos()">Cargar los datos</button>
    </body>
</html>
```

Cuando se pulse el botón se ejecutará la función cargaDatos() de JavaScript que aún no existe.

Ahora hay que escribir un script para implementar la función que cargará los datos y los insertará en la capa identificada como “capa1”.

```
<script type="text/javascript">
    var peticion;

    // Funcion que realiza la peticion de datos.
    function cargaDatos() {
        peticion=new XMLHttpRequest();
        peticion.onreadystatechange=muestraDatos;
        peticion.open("GET","http://localhost:8080/testajax/meteo",true);
        peticion.send();
    }

    // Funcion que muestra los datos recibidos.
    function muestraDatos()
    {
        // Acciones a realizar cuando los datos esten listos.
        if (peticion.readyState==4 && peticion.status==200)
        {
            document.getElementById("capa1").innerHTML=peticion.responseText;
        }
    }
</script>
```

Código que realiza la petición XMLHttpRequest.

En este ejemplo se accede a la URL http://localhost:8080/testajax/meteo en la que se ha puesto un servlet que responde a la petición con un texto.

Como se ha realizado una petición asincrona (parámetro “true” del método open) hay que definir una función que se ejecutará cuando los datos estén listos.

Para crear un objeto que funcione en navegadores que soporten solo el objeto ActiveX y el objeto JavaScript se puede usar este código:

```
var xmlhttp;
if (window.XMLHttpRequest)
 {// code for IE7+, Firefox, Chrome, Opera, Safari
 xmlhttp=new XMLHttpRequest();
 }
else
 {// code for IE6, IE5
 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
```

Fuente: http://www.w3schools.com

Este ejemplo tan sencillo puede resultar tedioso cuando hay que realizar muchas peticiones con AJAX. Los frameworks ayudan a que la programación sea mas sencilla.

Modificación de la estructura de la página web

La estructura de una página web se representa mediante el DOM (Document Object Model). Es una tecnología pensada inicialmente para manipular estructuras en XML, por lo que es también válida para manejar XHTML.

El DOM crea a partir de la página web una estructura en forma de árbol que se genera en los navegadores modernos una vez que la página se ha cargado completamente, transformado la página HTML en XHTML. Por lo tanto los programadores web pueden manipular la estructura de la página web a través de DOM.

Una vez transformado el documento en XHTML se crea un árbol cuyos nodos son los elementos del documento. La jerarquía de los nodos indica cómo están relacionados. Por ejemplo, el cuerpo y la cabecera del documento (tags <body> y <head>) descienden del nodo “Documento XHTML”. Los elementos presentes en la cabecera descenderán de la cabecera, y los elementos presentes en el cuerpo estarán representados por nodos que descienden del nodo “cuerpo del documento”. Ya sí sucesivamente hasta completar todo el documento.

Cada una de las etiquetas del XHTML se transforma en dos nodos, uno de tipo elemento y otro de tipo texto. La primera define el elemento, y la segunda el contenido del elemento. Esta estructura también soporta las etiquetas anidadas.

Existen doce tipos de nodos, pero en un documento XHTML suelen aparecer éstos:

- `Document`. Aparece siempre, ya que es el nodo raíz del que parten todos.
- `Element`. Representa un elemento XHTML. Puede contener atributos y de el pueden descender otros nodos.
- `Attr`. Representa cada uno de los atributos de una etiqueta XHTML (atributo=valor).
- `Text`. Texto que se encuentra contenido en una etiqueta (por ejemplo texto).
- `Comment`. Comentarios del documento.

Los demás tipos de nodos son: `DocumentType`, `CDataSection`, `DocumentFragment`, `Entity`, `EntityReference`, `ProcessingInstruction` y `Notation`.

Una vez que el árbol está creado se puede manipular mediante las funciones del DOM. Los elementos del árbol pueden ser creados, modificados y eliminados.

Existen dos formas de acceder a los elementos del árbol, una forma directa y otra a través de la estructura jerárquica. La forma mas rápida es la directa. Las funciones para acceder a los nodos de forma directa son:

- `getElementById()`: obtene una referencia al nodo a través del id que le ha proporcionado el programador, por ejemplo * *. Cada elemento de la página web debe tener un id único para que este sistema funcione. Es la función mas usada.
- `getElementsByName()`: obtiene una lista de todos los elementos de la página cuyo nombre se indique como parámetro. El nombre se especifica en las etiquetas con el atributo "name".
- `getElementsByTagName()`: obtiene una colección de todos los elementos de la página que tienen una etiqueta determinada.

En las dos últimas funciones el valor devuelto es un array de elementos DOM.

Una vez que se ha accedido a un nodo se pueden **modificar sus propiedades y atributos**. Estas propiedades coinciden con las propiedades de los estilos CSS. Como los atributos se transforman de forma automática en propiedades de los nodos basta con poner el nombre del atributo detrás del nodo para acceder a ellos, por ejemplo:

```
var midiv=getElementById("capa1");
midiv.visible=true;
```

Existe una excepción a esta regla, en JavaScript la palabra "class" está reservada, por lo que se usa en su lugar "className".

Las otras operaciones que se pueden hacer sobre un nodo son las de agregar nodos y eliminar nodos.

Para **agregar un nodo** de tipo elemento hay que seguir cuatro pasos:

1. Crear un nuevo nodo de tipo Element.
2. Crear un nodo tipo Text con el contenido del elemento.
3. Añadir el nodo Text al nodo Element como un hijo.
4. Añadir el nodo Element como hijo del nodo donde se desea insertar el elemnto.

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");
// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);
// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

Ejemplo de creación de un nodo. Fuente: www.librosweb.es.

Para **eliminar un nodo** se usa el método removeChild().

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);
...
<p id="provisional">...</p>
```

Ejemplo de eliminación de un nodo. Fuente: www.librosweb.es.

El uso de esta tecnología permite crear páginas web dinámicas donde aparecen nuevos elementos o se eliminan en función de la lógica de negocio del programa, la cual se implementa en el servidor. Es una de las tecnologías que se utilizan en AJAX para dotar al documento de una apariencia de aplicación de escritorio.

Prácticas

Creación de un formulario con verificación de datos del lado del cliente

En esta práctica se pretende ver el uso de JavaScript como medio de verificación del lado del cliente. El ejercicio consiste en crear un formulario donde el usuario introducirá un DNI y el cliente (navegador) deberá indicar si es válido o no.

Solución:

En negrita se remarca el código JavaScript para diferenciarlo del resto de la página web.

```
<html>
    <head>
        <title>Validar formulario</title>
        <!-- Estilos empleados en la página. -->
        <style type="text/css">
            body table {
                font-family: verdana,arial,helvetica;
                font-size: 12px;
            }

            .titulotabla {
                text-align: center;
                font-size: 16px;
                font-weight: bold;
                text-decoration: underline;
            }

            #capa1 {
                margin: 100;
                background-color: #DADADA;
            }
        </style>
        <!-- Script para validar el formulario. -->
        <script>
            /*
             Función para validar un DNI con letra.
            */
            function ValidarDNI(dni)
            {
                // Según la Wikipedia: http://es.wikipedia.org/wiki/N
%C3%BAmero_de_identificaci%C3%B3n_fiscal
                // en la parte en la que habla del DNI.
                var letrasDNI = "TRWAGMYFPDXBNJZSQVHLCKE";

                // Comprobar que no se ha pasado un texto vacio.
                if(dni.length == 0)
                {
                    resultado = false;
                }
                else
                {
                    // comprobar la longitud (>= 7+1 y <= 8+1)
                    if((dni.length < 8 ) || (dni.length > 9))
                    {
                        // Longitud erronea.
```

```
                    resultado = false;
                }
                else
                {
                    // Obtener el ultimo digito
                    letra = dni[dni.length - 1];

                    // Obtener el resto de digitos menos el ultimo.
                    numero = parseInt(dni);

                    // Comprobar la ultima letra teniendo en cuenta que se
                    // puede introducir el NIF en minúsculas, para poder
                    // comparar bien se pasa a mayúsculas.
                    if(letra.toUpperCase() == letrasDNI.charAt(numero % 23))
                    {
                        // Letra correcta.
                        resultado = true;
                    }
                    else
                    {
                        resultado = false;
                    }
                }
            }

            return resultado;
        }

        /*
         Función que valida un formulario con una fecha y un dni.
         Los parámetros son los nombres de los campos, cuyos objetos se
         recuperarán con getElementById().
        */
        function ValidaFormulario(cdni)
        {
            objdni = document.getElementById(cdni);
            dni = objdni.value;
            // Comprobar el campo dni, y poner los colores del campo dni de
            // acorde a esta circunstancia.
            if(ValidarDNI(dni) == true)
            {
                // DNI correcto.
                objdni.style.backgroundColor = '#80FF80';
            }
            else
            {
                objdni.style.backgroundColor = '#FF8080';
                alert("El DNI introducido no es correcto.");
            }

            // Evitar que se envíe el formulario.
            return false;
        }
    </script>
</head>
<body>
   <!-- Formulario con un campo fecha y otro DNI que deben ser validados. -->
```

```
       <div name="capa1" id="capa1">
        <form name="formu1" id="formu1">
            <table align="center">
                <tr>
                    <td colspan="2" class="titulotabla">Datos del formulario</td>
                </tr>
                <tr>
                    <td>DNI con letra:</td>
                    <td><input type="text" name="campodni" id="campodni"></td>
                </tr>
                <tr>
                    <td colspan="2" align="center"><input type="submit"
value="Validar" onclick="return ValidaFormulario('campodni');"></td>
                </tr>
            </table>
        </form>
       </div>
    </body>
</html>
```

Formulario que valida la entrada correcta de un DNI.

Creación de una página web que mantiene la información actualizada mediante AJAX

Esta práctica consiste en tener un formulario en el que figuran los datos de un usuario (identificador, nombre y libros que ha pedido prestados), los cuales se actualizan de forma automática al introducir un nuevo identificador de usuario sin tener que recargar la página.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Informaci&oacute;n sobre el usuario</title>
        <script language="javascript">
            var peticion;

            // Funcion que realiza la peticion de datos.
            function cargaDatos(funcion,control)
            {
                var cnt=document.getElementById(control).value;
                peticion=new XMLHttpRequest();
                peticion.onreadystatechange=funcion;
                peticion.open("GET","http://localhost:8080/biblioteca3/infousuario?
idusuario="+cnt,true);
                peticion.send();
            }

            function datosCargados()
            {
                if (peticion.readyState==4 && peticion.status==200)
                {
```

```
                    return true;
                } else {
                    return false;
                }
            }

            function infoUsuario()
            {
                if(datosCargados()) {
                    document.getElementById("info1").innerHTML=peticion.responseText;
                }
            }
        </script>
    </head>
    <body>
        <form>
            Introduzca el codigo de usuario <input id="idusuario" type="text"
name="idusuario" onblur="cargaDatos(infoUsuario,'idusuario')"> <input type="button"
value="Buscar" onclick="cargaDatos(infoUsuario,'idusuario')">
        </form>
        Nombre del usuario: <span id="info1"></span>
    </body>
</html>
```

Página JSP con el formulario que se actualiza mediante AJAX.

```
/*
 * Servlet que devuelve el nombre completo de un usuario a partir de su codigo.
 */
package org.curso.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.sql.*;

/**
 *
 * @author pedro
 */
@WebServlet(name = "infousuario", urlPatterns = {"/infousuario"})
public class infousuario extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
```

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {

            out.println(getNombreUsuario(request.getParameter("idusuario")));

        } finally {
            out.close();
        }
    }

    private String getNombreUsuario(String usuario)
    {
        ResultSet rs = null;
        Statement s = null;
        Connection conexion = null;
        String resultado = ":-P";

        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();

             conexion =
DriverManager.getConnection("jdbc:mysql://localhost/biblioteca?user=curso&password=");
             s = conexion.createStatement();
             rs = s.executeQuery("SELECT * FROM usuarios WHERE id='" + usuario + "'");

             if(rs.next() != false) {
                 resultado = rs.getString("nombre");
             }
        } catch (Exception e) {
            // Tratar las excepciones.
            //...
            e.printStackTrace();
        }

        return resultado;

    }
    //...
}
```

Servlet que proporciona la información.

En esta aplicación basta con pasar de un control a otro con el tabulador o bien pulsando en el botón “Buscar” para que se lance la petición al servlet y se actualice la información de la página.

Tema 6

Desarrollo de aplicaciones web híbridas.

Objetivos:

- Aprender a reutilizar código.
- Conocer las APIs de programación disponibles.
- Conocer el uso de repositorios.

Introducción

Las aplicaciones web híbridas, en inglés *mashup* (remezcla), son aplicaciones web que toman el contenido de diversas fuentes y lo integran para crear una nueva aplicación.

Las *mashup* utilizan contenido de terceros a través de API (interfaces de programación) públicas y fuentes de datos. Las *mashup* también utilizan web services para obtener contenidos o servicios.

En una *mashup* intervienen tres elementos:

- El proveedor de contenidos. Pueden ser varios los proveedores de contenidos de los que se toman datos o servicios. Pueden ser accedidos mediante API, servicios web, etc.
- Sitio web *mashup*. Proporciona un nuevo servicio tomando información de terceras partes e integrándola en una nueva aplicación.
- Cliente. Suele ser un navegador que acede a la aplicación web. El clinte puede integrar la aplicación a través de medios locales como JavaScript.

Existen tres tipos de *mashup*: de negocio o empresa, consumidores y de datos.

- Empresariales: son aplicaciones que combinan recursos internos de la empresa con recursos externos.
- Consumidores: combinan diferentes tipos de datos púbicos y los presentan en una única aplicación.
- De datos: combinan datos similares de diversas fuentes en una única representación.

Aplicaciones típicas son los agregadores de noticias, aplicaciones que trabajan sobre mapas de proveedores (por ejemplo Google Maps) o buscadores de servicios como viajes o seguros.

Métodos de reutilización de código e información

Las aplicaciones web híbridas toman tanto código (programas) como información (datos) de un proveedor.

La **reutilización de código** consiste en usar código de terceras partes, normalmente a través de API públicas, o bien volver a utilizar código propio ya escrito de forma que no haya que escribir código nuevo para realizar una función ya implementada.

Las distribuciones de Java (SE, EE, etc.) poseen muchas API para utilizar código escrito por los desarrolladores de Java y software de terceros. Por ejemplo, JDBC es un API de Java que permite usar servicios de terceros (gestores de bases de datos). También hay API de terceros que se pueden usar de forma directa mediante clases Java o bien a través de servicios web, etc.

Cuando se desarrolla una aplicación Java las clases se guardan en paquetes y ficheros que posteriormente pueden ser reutilizados en el mismo o en distintos proyectos. La reutilización de código está muy presente en Java. En este tipo de reutilización de código también se engloban las Java Beans y los EJB (controles de servidor) que pueden ser reutilizados en el mismo o en otros proyectos.

Los servicios web también proporcionan una forma de reutilización de código, ya que implementan funcionalidades que ofrece el programa sin que haya que reescribir ninguna línea de código.

La **reutilización de la información** puede ocurrir de varias formas, ya sea información propia almacenada en una base de datos o en otro tipo de soporte permanente o información obtenida de otras fuentes mediante servicios web, RSS o tecnologías similares.

Por lo tanto algunas de tecnologías empleadas en la reutilización de la información pueden ser:

- JDBC: consulta de bases de datos relacionales (ya visto).
- XML: para el traspaso de la información.
- RSS y ATOM: sistemas de redifusión de la información basados en XML.
- Servicios web: reutilizan la información de otros conponentes.

Interfaces de programación API disponibles

En este manual se ha hablado muchas veces de las API. Las API (Application Programming Interface) es un interfaz mediante el cual los componentes de software pueden comunicarse entre ellos. Cuando se utiliza una API lo que realmente se está haciendo es utilizar un método conocido por ambas partes para el uso de código e intercambio de información mediante métodos y atributos. Las API pueden incluir especificaciones para rutinas (funciones y procedimientos), estructuras de datos y objetos. Las API se utilizan en el desarrollo de software, por lo cual tienen su aplicación en el código fuente.

Las API se refieren a interfaces, por lo cual la implementación que realice la otra parte es, en principio, desconocida y funciona como una “caja negra”.

En los lenguajes de programación orientada a objetos las API a menudo se refieren a definiciones de clases con un conjunto de métodos públicos.

Las API se suelen referir a librerías y frameworks, que describen su forma de trabajar. Una misma API puede estar implementada por mas de un fabricante, por lo que el interfaz es idéntico pero la implementación distinta.

En Java existen tres tipos de API:

- Oficial del núcleo (core) de cualquiera de las ediciones de la plataforma Java, contenidas en el JRE y en el JDK.

  Algunas API para la plataforma Java SE:

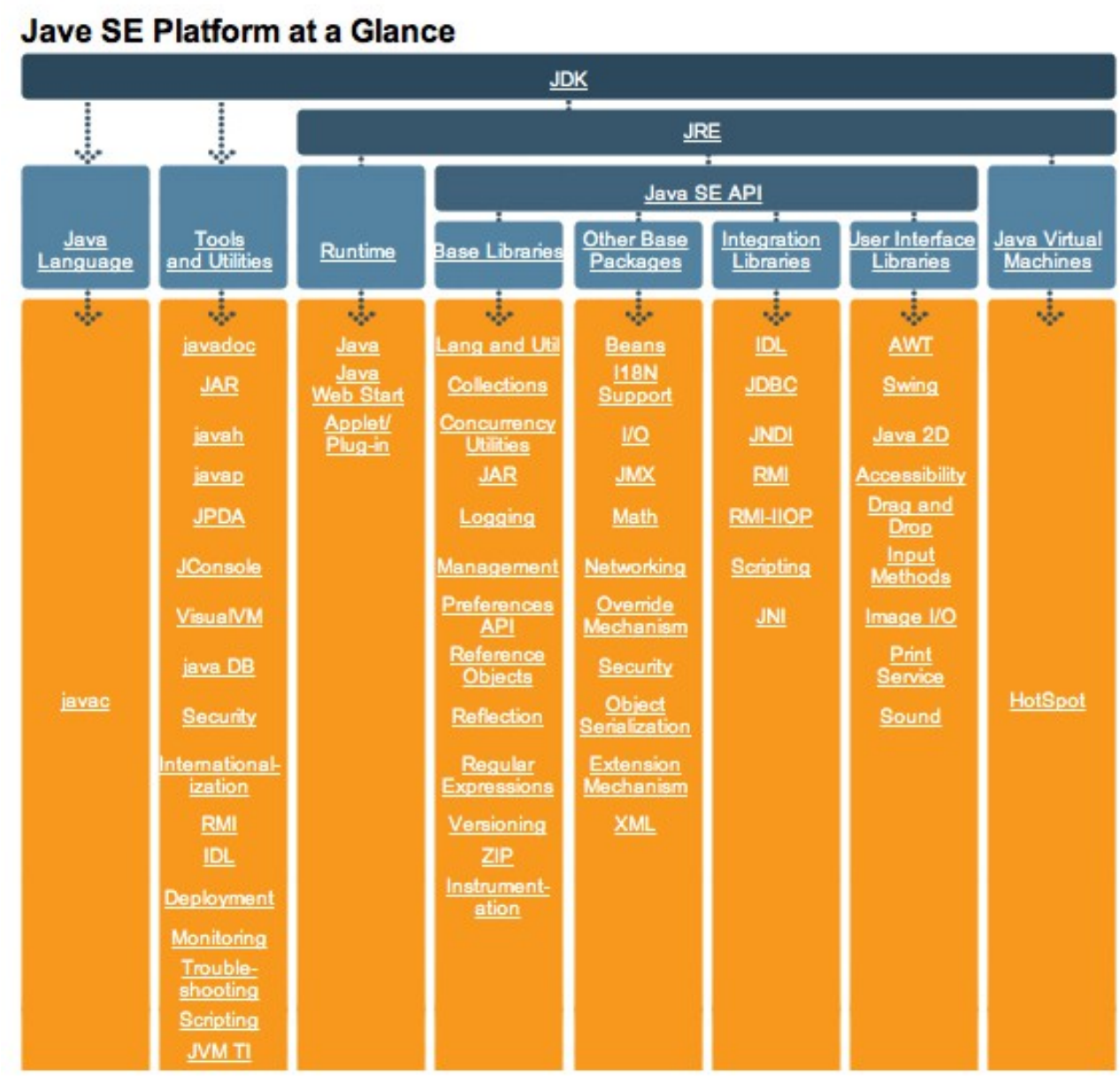
  - Java Data Object (JDO).
  - Java Help.
  - Java Media Framework (JMF).
  - Java Naming and Directory Interface (JNDI).
  - Java 3D (J3D).
  - Java OpenGL (JOGL)
  - Java Mail

  Para la plataforma Java EE:

  - Java Message Services (JMS).
  - JavaServer Faces (JSF).
- Oficial opcional, que se descargan por separado. Siguen la JSR (Java Specification Request). Con el tiempo algunas de estas tecnologías de distribuyen con el núcleo.

  La lista de API opcionales se encuentra en http://www.jcp.org/en/jsr/stage?listBy=final, algunas de estas API son:

  - Java API for XML Processing (JAXP).
  - Java Enterprise Beans (EJB).
  - Java Servlet y Java Server Pages (JSP).
  - Java Database Conectivity (JDBC).
  - Streaming API for XML (StAX).
  - Java API por XML Web Services (JAX-WS).
- No oficiales. Son API de terceras partes que no están relacionadas con los JSR.



Tecnologías implementadas en Java SE. Fuente: Oracle.

Uso de repositorios de información

Dos de las tecnologías empleadas para recuperar y agregar información en un sitio web son los “feeds” RSS y ATOM. Ambas tecnologías están basadas en el formato XML para el intercambio de información. Puesto que son tecnologías muy parecidas se estudiará mas en profundidad solo una de ellas: RSS 2.0

RSS 2.0

Los feed RSS 2.0, desde el punto de vista del cliente, es un flujo de información en XML que se obtiene mediante el protocolo HTTP.

Un servlet que recupera la información RSS en bruto puede ser así:

```
package org.curso.servlets;

import java.io.*;
import java.io.PrintWriter;
import java.net.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author pedro
 */
public class LectorRSS extends HttpServlet {

     protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
             throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            URL conexion = new URL(request.getParameter("rssurl"));
            BufferedReader entrada = new BufferedReader(new
InputStreamReader(conexion.openStream()));

            String linea;
            while((linea = entrada.readLine()) != null) {
                out.println(linea);
            }
        } finally {
            out.close();
        }
    }
    // ...
}
```

Servlet que obtiene un feed RSS y lo muestra sin procesar.

Los feed RSS se organizan en canales. Los canales tiene elementos obligatorios y opcionales.

Elementos obligatorios	
`title`	Título del canal. Puede contener espacios.
`link`	URL del sitio web al que corresponde el canal.
`description`	Frase o texto corto que describe el canal.
Algunos elementos opcionales	
`language`	El lenguaje en el que está escrito el canal. Ejemplo: `es-es`
`copyright`	Información de copyright del contenido del canal.

managingEditor	Dirección de correo electrónico del responsable de edición del canal.
webMaster	Correo electrónico del responsable técnico del canal.
pubDate	Fecha de publicación según el formato RFC 822. El año puede tener dos o cuatro caracteres. Ejemplo: `Sat, 07 Sep 2002 00:00:01 GMT`
lastBuildDate	Hora a la que cambió por última vez el canal.
category	Categoría(s) a la(s) que pertenece el canal.
generator	Programa usado para generar el canal.
ttl	Indica los minutos que puede estar un canal en caché.
image	Imagen asociada al canal. Este elemento tiene sub-elementos.

Los canales contienen elementos (ítems) cada uno de los cuales representa una información (puede ser una noticia, un evento, etc.). Cada información es un resumen de una historia que reside en una URL o que está totalmente contenida en el ítem. Todos los elementos del ítem a excepción de `link`, `description` y `title` son opcionales:

title	Título del ítem.
link	URL del ítem.
author	Direccón de correo electrónico del autor.
description	Resumen del ítem.
category	Categoría o categorías del ítem.
comments	URL con comentarios relativos al ítem.
enclosure	Describe un objeto multimedia adjunto al ítem.
guid	Identificador único para este ítem.
pubDate	Fecha de publicación del ítem.
source	Canal RSS del que proviene el ítem.

```
<rss version="2.0">
<channel>
```

```
      <title>Canal curso</title>
      <link>http://localhost:8080/rss/canalcurso</link>
      <description>Canal de ejemplo para el curso</description>
      <language>es-es</language>
      <pubDate>Wed, 04 Jul 2012 19:30:00 GMT</pubDate>
      <lastBuildDate>Wed, 04 Jul 2012 19:30:00 GMT</lastBuildDate>
      <generator>HandMade Tech 2.2</generator>
      <managingEditor>me@curso.org</managingEditor>
      <webMaster>megain@curso.org</webMaster>
      <item>
            <title>Prueba 1</title>
      <link>
            http://localhost:8080/noticia1
      </link>
      <description>
            Esto es una <u>prueba</u>.
      </description>
      </item>
</channel>
</rss>
```

Ejemplo de canal RSS 2.0 en XML.

El canal puede ser enviado en bruto para que lo interprete el cliente o bien puede ser interpretado por la aplicación y mostrado de forma integrada.

Creación de repositorios de información a medida

Se pueden crear varios tipos de repositorios de información. Si se desean crear fuentes RSS y/o ATOM se pueden usar servlets y servicios web para generar feeds de estos tipos, también se pueden reenviar feeds y agregarlos sirviéndolos como un solo feed.

Este tipo de controles debe enviar la información en formato RSS o ATOM (XML).

Otra posibilidad es crear un servicio web que proporcione información según nuestras necesidades.

Hasta ahora se habían creado servicios web sencillos que enviaban datos simples (enteros, valores booleanos, cadenas de caracteres, etc.). Los repositorios de información devuelven información mas compleja, y por lo tanto es necesario crear **servicios web que devuelvan tipos de datos específicos** distintos a los tipos de datos estándar. Para devolver información compleja primero hay que crear los tipos de datos necesarios para almacenar y transportar la información.

> El editor Netbeans se encarga de la parte mas tediosa, sin que el programador tenga que modificar manualmente ningún fichero XML.

Primero hay que definir una clase para almacenar el nuevo tipo de dato. Esta clase deberá seguir las especificaciones de las Java Beans:

- Debe tener un constructor sin argumentos.

- Debe ser serializable. Esto es imprescindible para que pueda crearse el esquema en el fichero WSDL.
- Las propiedades deben ser accesibles mediante métodos get y set con la convención de nomenclatura estándar. (Por ejemplo si el atributo es “altura” tendrá dos métodos: `public void setAltura(int altura)` y `public int getAltura()`).

Una vez definida la clase las operaciones del servicio web pueden devolver datos del nuevo tipo. Si se necesita devolver mas de un dato se devolverá un array del tipo de dato elegido, el cliente Java verá este array como una lista (List) de un tipo de dato creado a partir del esquema del fichero WDSL, y no del tipo creado en el primer paso.

Ejemplo de Java Bean sencillo:

```
/*
 * Java Bean.
 */
package org.curso.tipos;

/**
 *
 * @author pedro
 */
public class UsuarioBean implements java.io.Serializable
{

    private String id;
    private String nombre;

    public UsuarioBean()
    {
        // Constructor sin argumentos. Necesario para ser un Bean.
    }

    public void setId(String id)
    {
        this.id = id;
    }

    public void setNombre(String nombre)
    {
        this.nombre = nombre;
    }

    public String getId()
    {
        return (id);
    }

    public String getNombre()
    {
        return (nombre);
    }
}
```

Tipo de dato sencillo que almacena dos propiedades.

Un servicio web para probar este tipo de dato:

```
package org.curso.ws;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import org.curso.tipos.UsuarioBean;

/**
 *
 * @author pedro
 */
@WebService(serviceName = "WsPruebaCompleja")
public class WsPruebaCompleja {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "pruebacompleja")
    public UsuarioBean pruebacompleja(@WebParam(name = "id") String id) {
        UsuarioBean u = new UsuarioBean();

        u.setId(id);
        u.setNombre("Asdf");

        return u;
    }

    /**
     * Web service operation
     */
    @WebMethod(operationName = "pruebacomplejaarray")
    public org.curso.tipos.UsuarioBean[] pruebacomplejaarray() {
        UsuarioBean u = new UsuarioBean();
        UsuarioBean v = new UsuarioBean();

        u.setId("AA");
        u.setNombre("ABCD");

        v.setId("XX");
        v.setNombre("WXYZ");


        UsuarioBean[] a = new UsuarioBean[2];

        a[0] = u;
        a[1] = v;

        return a;
    }
}
```

Servicio web que ofrece datos de prueba.

Este servicio web genera una respuesta SOAP así:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:pruebacomplejaarrayResponse xmlns:ns2="http://ws.curso.org/">
            <return>
                <id>AA</id>
                <nombre>ABCD</nombre>
            </return>
            <return>
                <id>XX</id>
                <nombre>WXYZ</nombre>
            </return>
        </ns2:pruebacomplejaarrayResponse>
    </S:Body>
</S:Envelope>
```

Respuesta SOAP con múltiples apartados “response”.

Para usar el servicio web desde un servlet:

```
//...

private UsuarioBean pruebacompleja(java.lang.String id) {
        org.curso.wscli.WsPruebaCompleja port = service.getWsPruebaComplejaPort();
        return port.pruebacompleja(id);
    }


    private java.util.List<org.curso.wscli.UsuarioBean> pruebacomplejaarray() {
        org.curso.wscli.WsPruebaCompleja port = service.getWsPruebaComplejaPort();
        return port.pruebacomplejaarray();
    }

//...
```

Extracto del servlet que actúa como cliente del servicio web.

Prácticas

Creación de un sencillo gestor de biblioteca

En esta práctica se pretende aplicar las tecnologías mencionadas en el manual resultando en un sencillo gestor de biblioteca en el que los usuarios registrados pueden pedir prestados libros, llevando el sistema un control sobre los usuarios, los libros y los préstamos.

Este proyecto se puede implementar de muchas formas, pero puesto que debe ser un compendio del uso de las tecnologías estudiadas se impondrán algunos requisitos extra:

- El cliente debe ser rico (web 2.0) y emplear Ajax y JavaScript ahí donde sea posible. Se empleará validación en los formularios por parte del cliente.
- El servidor se implementará con servlets y páginas JSP.
- Los datos se obtendrán mediante servicios web.
- Se incluirá un canal RSS con información sobre los libros prestados.

La aplicación tendrá un menú principal y tres secciones bien diferenciadas: gestión de usuarios, gestión de libros y gestión de préstamos.

La información se almacenará en una base de datos, la cual ya se ha creado en ejercicios anteriores.

Referencias

Existen publicados muchos documentos sobre el desarrollo de aplicaciones web en entono de servidor. A continuación se muestran algunas referencias que pueden ser útiles para su estudio y análisis o como material de consulta puntual.

[1] Documentación oficial de Java EE: http://docs.oracle.com/javaee/

[2] Documentación oficial de Java SE: http://docs.oracle.com/javase/

[3] Documentación de ASP.net:

- Controles de Servidor: http://support.microsoft.com/kb/306459/es
- Programación web con ASP.net: http://msdn.microsoft.com/es-es/asp.net/centrum-asp-net.aspx
- Información general de ASP.net: http://msdn.microsoft.com/es-es/library/4w3ex9c2.aspx

[4] Documentación de PHP: http://es.php.net

[5] Apache Tomcat: http://tomcat.apache.org/

[6] Netbeans: http://netbeans.org/

[7] Wikipedia:

- http://es.wikipedia.org/
- http://en.wikipedia.org/