SISTEMAS DISTRIBUIDOS BASADOS EN LA WORLD WIDE WEB

Sistemas informáticos I

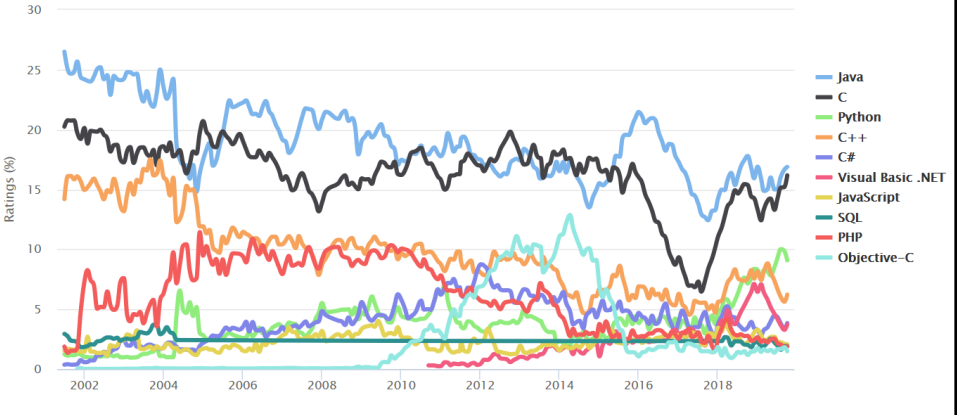
2.6 APLICACIONES WEB ALTERNATIVAS DEL LADO DEL SERVIDOR

Sistemas distribuidos basados en la World Wide Web

CAMBIO EN LAS NECESIDADES

- Desde el origen de la WWW la tecnología se ha ido adaptando a las nuevas necesidades:
  - Ya no quiero servir documentos que leer de manera no secuencial
  - Quiero ejecutar aplicaciones
- En temas anteriores nos hemos focalizado en PHP, Python y JavaScript como tecnologías para ejecutar funcionalidad en el servidor, pero existen otras muchas alternativas

COMPARATIVA ENTRE LENGUAJES



JAVA EE: *JAVA PLATFORM, ENTERPRISE EDITION*

- Extensión de la *Java SE* con especificaciones y APIs para:
  - Gestión de transacciones
  - Acceso a base de datos
  - Desarrollo de aplicaciones web
  - Etc
- Objetivo en el contexto WWW: ejecutar componentes Java en el servidor
- Filosofía: lenguaje único sobre múltiples plataformas (cf. .Net)
- Interfaz con los elementos de comunicación establecidos en el protocolo HTTP y los formularios HTML
- Especificaciones web:
  - Programas en el servidor: *Servlets*
  - Páginas dinámicas: *JavaServer Pages*, JSP
  - *Enterprise Java Beans*, EJB
  - *Java Server Faces*, *JSF*
  - Distintas APIs para web services
  - …

https://www.oracle.com/java/technologies/java-ee-glance.html

SERVLETS

- Especificación de más bajo nivel
- Extienden la funcionalidad de los servidores web con una arquitectura basada en componentes "ejecutables"
- Aplicación Java que se ejecuta en el servidor gestionando y procesando peticiones HTTP
  - Se ejecutan totalmente en el servidor bajo petición de un cliente (vs. applets)
  - Se invocan a través de una URL que identifica el programa
  - Reemplaza a los programas de interfaz CGI. Sintaxis más sencilla
  - Necesitan un servidor JEE específico: *Tomcat*, *JBOSS*, *GlassFish*…
- Su implementación se incluyó en *Java EE 5 SDK*: *javax.servlet.**
- Todo servlet debe implementar la interfaz *javax.servlet.Servlet*
  - *javax.servletGenericServlet*
  - *javax.servlet.http.HttpServlet*

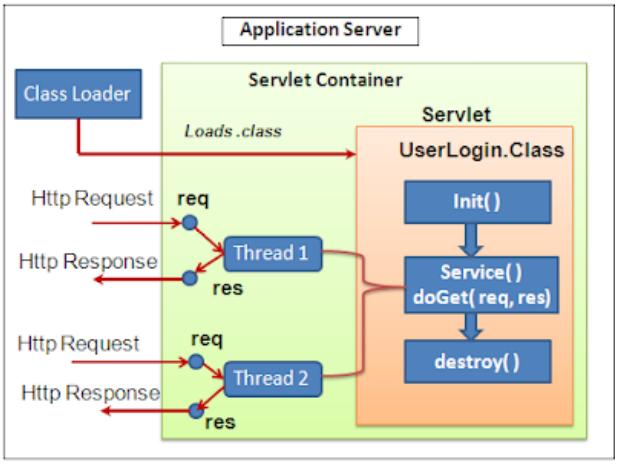
https://www.oracle.com/technetwork/java/index-jsp-135475.html

VENTAJAS DE LOS SERVLETS

- Portabilidad y flexibilidad:
  - La conexión del servlet se basa en una API independiente de la plataforma definida en el estándar de Java
  - El código Java altamente transportable
  - Permite aprovechar objetos reutilizables (EJBs)
- Seguridad:
  - Los servlets se ejecutan bajo un único proceso (*servlet engine*)
  - Permite acceso protegido integrado en un entorno de autenticación única
- Rendimiento:
  - Entorno de ejecución propio
  - Se ejecutan y permanecen en memoria
  - Se pueden precargar o cargar bajo demanda
  - Mantienen sesiones entre peticiones HTTP
  - Son multitarea
  - Escalables entre multiprocesadores y sistemas heterogéneos

EPS

CICLO DE VIDA DE UN SERVLET



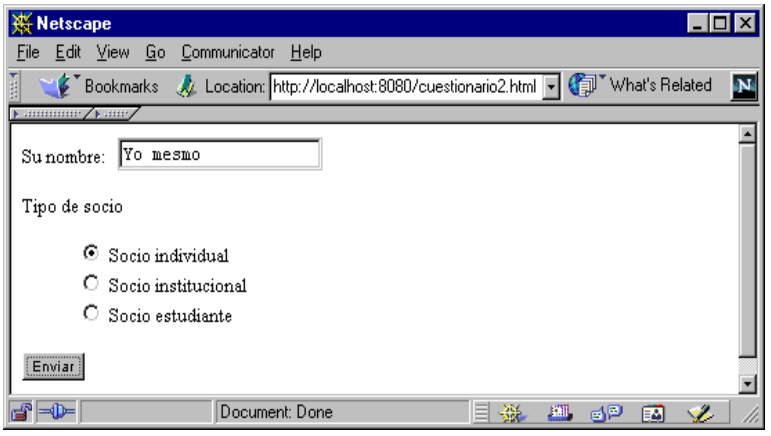
https://docs.oracle.com/javaee/7/tutorial/servlets002.htm

EPS

EJEMPLO: LA PARTE CLIENTE

```
<form action="http://localhost:8080/servlet/Ejemplo" method="get">
Su nombre:   <input type="field" name="nombre"> <br><br>
Tipo de socio
<blockquote>
  <input type="radio" name="tipo" value="individual">
  Socio individual <br>

  <input type="radio" name="tipo" value="institucional">
  Socio institucional <br>

  <input type="radio" name="tipo" value="estudiante">
  Socio estudiante <br>
</blockquote>

<input type="hidden" name="escondido" value="xxx">

<input type="submit" value="Enviar">
</form>
```

EJEMPLO: LA PARTE CLIENTE

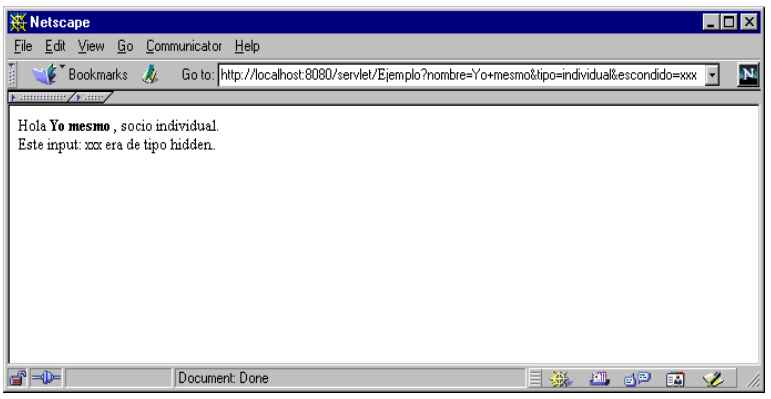


EJEMPLO: LA PARTE SERVIDORA

```
public class Ejemplo extends HttpServlet {

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

    String nombre = request.getParameter("nombre");
    String tipo = request.getParameter("tipo");
    String escondido = request.getParameter("escondido");
    PrintWriter out = response.getWriter();
    out.println(
      "<html><body>"
      + "Hola <b>" + nombre + "</b>, socio " + tipo + ".<br>"
      + "Este input: " + escondido + " era de tipo hidden."
      + "</body></html>"
    );
    out.close ();
  }
}
```

DEPRECATED

EJEMPLO: LA PARTE SERVIDORA



IMPLEMENTACIÓN DE SESIONES

- Interfaz *javax.servlet.http.HttpSession*
  - Id de sesión
  - Fecha de creación
  - Fecha de último acceso
  - Mecanismos para invalidar la sesión
- Aunque las implementaciones de JEE lo hacen automáticamente, no hay "magia" que modifique el protocolo HTTP (protocolo sin estado)
  - Cookies
  - Reescritura de URL
  - Campos ocultos

  Con las limitaciones y ventajas de cada uno de ellos

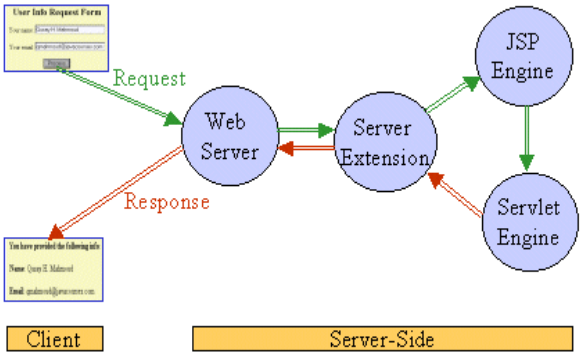
JSP: *JAVASERVER PAGES*

- Tecnología *Java* de *server side scripting*
- Generación dinámica de contenidos
- Un archivo JSP contiene:
  - Código HTML nativo (*template data*)
  - Elementos JSP: directivas y acciones
  - Código funcional Java
    - Se pueden utilizar variables implícitas: `request`, `response`…
  - Mecanismos de extensión de etiquetas: *JSP Standard Tag Library*, JSTL
- Necesita un servidor web que soporte JSP
- Se solicitan igual que un documento HTML desde un navegador

https://www.oracle.com/technetwork/java/index-jsp-138231.html

CICLO DE INVOCACIÓN DE JSP

- Se ejecutan en un *servlet engine* específico (*jsp engine*):
  - Convertidas en un servlet dinámicamente la primera vez que se ejecutan
  - Se crea un objeto que implementa la interfaz *HttpJspPage*



EPS

ELEMENTOS JSP: DIRECTIVAS

- Directivas: Mensajes para la *jsp engine*
- Posibles ejemplos:
  - `<%@ page {atributo="valor"}* %>`
    Define propiedades generales de la página: lenguaje, Classpath, herencia, imports...
  - `<%@ include file="nombre_archivo" %>`
    Orden de incluir un archivo en la página actual, en tiempo de compilación
  - `<%@ taglib uri="URI_archivo_etiquetas" %>`
    Referencia a un archivo con declaración de etiquetas de usuario.

EPS

ELEMENTOS JSP: ACCIONES

- Acciones: Comandos a realizar en tiempo de ejecución
- Posibles ejemplos:
  - `<jsp:include page="URLrelativa" />`
  - `<jsp:forward page="URLrelativa" />`
  - `<jsp:useBean id="nombre" scope="page|request|session|application" class="nombreCompletoClase" />`
    Define para su uso en la página un bean con el nombre "nombre"
  - `<jsp:setProperty name="nombre" property="propName" value="valor"/>`
    Asigna valor a una propiedad de un bean
  - `<jsp:getProperty name="nombre" property="propName" />`
    Recupera una propiedad de un bean

CÓDIGO FUNCIONAL

- Declaraciones de variables y métodos
  `<%! declaracion %>`
  Métodos especiales : `jspInit()` y `jspDestroy()`
- *Scriptlets*: Fragmentos de código Java
  `<% fragmento_de_código %>`
  El código introducido se copia al cuerpo del método del servlet compilado
- Expresiones:
  `<%= expresión %>`
  En tiempo de ejecución:
  - Se evalúa la expresión
  - El resultado se convierte a String
  - Se incluye en la página resultado

EJEMPLO

```
<html><body>
Hola <b> <%= request.getParameter("nombre") %> </b>,
socio <%= request.getParameter("tipo") %>. <br>
Este input: <%= request.getParameter("escondido") %>  era de tipo hidden.

<br>
<% String tipo = request.getParameter("tipo");
   if(tipo.equals("individual")) { %>
    Te toca pagar 30 euros
<% } else if(tipo.equals("estudiante")) { %>
    Te toca pagar 15 euros
<% } else { %>
    Te toca pagar 300 euros
<% } %>
</body></html>
```

BIBLIOTECAS DE ETIQUETAS

- El código se vuelve farragoso y surgen bibliotecas de etiquetas para separar código funcional del HTML
  - Específicas
  - Estándar: *JSP Standard Tag Library*, JSTL
- Las bibliotecas englobadas en JSTL son:
  - **core**: iteraciones, condicionales, manipulación de URL y otras funciones generales.
  - **xml**: para la manipulación de XML y para XML-Transformation.
  - **sql**: para gestionar conexiones a bases de datos.
  - **fmt**: para la internacionalización y formateo de las cadenas de caracteres como cifras

https://www.oracle.com/technetwork/java/index-jsp-135995.html

BIBLIOTECAS DE ETIQUETAS

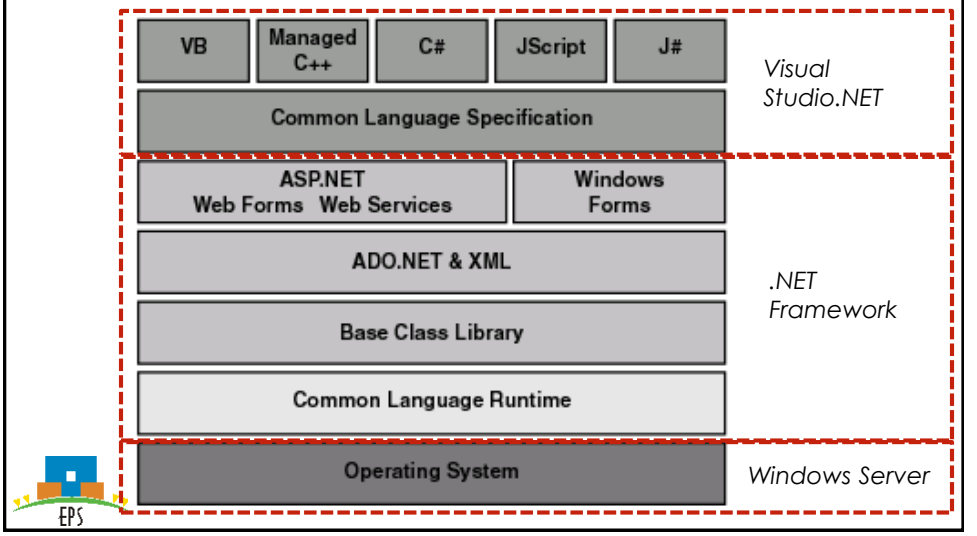
```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html><body>
Hola <b> <c:out value="${param.nombre}"/> </b>,
socio <c:out value="${param.tipo}"/>. <br>
Este input: <c:out value="${param.escondido}"/>  era de tipo hidden.

<br>
<c:choose>
   <c:when test = "${param.tipo == 'individual'}">
      Te toca pagar 30 euros
   </c:when>
   <c:when test = "${param.tipo == 'estudiante'}">
      Te toca pagar 15 euros
   </c:when>
   <c:otherwise>
      Te toca pagar 300 euros
   </c:otherwise>
</c:choose>
</body></html>
```

https://www.oracle.com/technetwork/java/index-jsp-135995.html

ARQUITECTURA .NET

- Arquitectura de Microsoft para aplicaciones Web
- Consta de tres componentes principales:
  - Un entorno de aplicaciones independiente del lenguaje y optimizado para el entorno distribuido: **.NET Framework**
  - Un entorno de desarrollo para la programación de aplicaciones: **Visual Studio .NET**
  - Un sistema operativo que soporta entornos distribuidos y el framework .NET: **Windows Server**
- Filosofía: plataforma única compartida por múltiples lenguajes
  - Compilados a Microsoft Intermediate Language (MSIL)

COMPONENTES ARQUITECTURA .NET



.NET FRAMEWORK

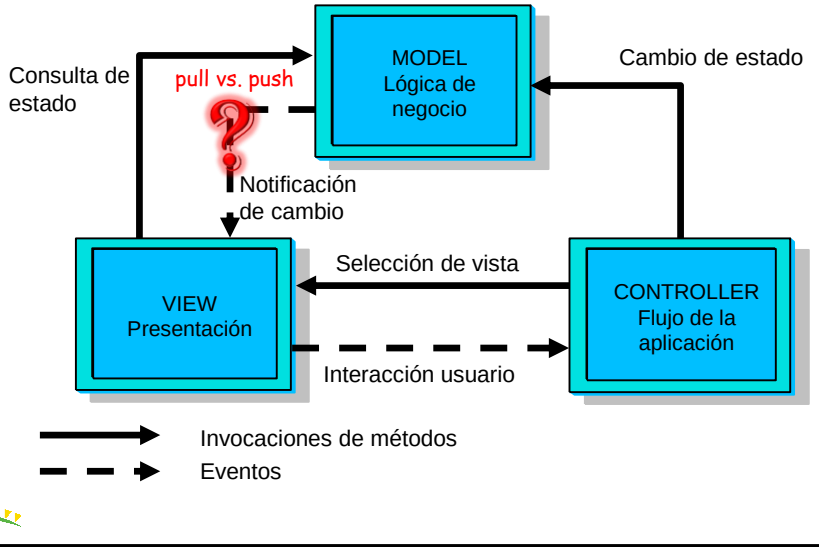
- *Common Language Runtime*, CLR: Entorno de ejecución de los programas MSIL
  - Compilación Just In Time (JIT) al entorno de ejecución
- *Base Class Library*: Biblioteca básica de funciones estándar, como las de gestión de E/S, seguridad, comunicaciones, tareas, texto…
- *ADO.NET*: Clases para acceder a datos a través de ODBC, OLE DB, Oracle o SQL Server
- *XML*: Clases para manipulación de documentos XML
- *ASP.NET*: Clases para el soporte de construcción de páginas Web
  - Soporta formularios Web y Web Services
- *Windows Forms*: Clases para soportar la construcción de clientes Windows

- .NET Framework no integra componentes para realizar lógica de negocio similares a EJBs. Se deben realizar como servicios de Windows o como componentes COM+

MODELO DE ARQUITECTURA DE APLICACIONES WEB

- Algunos de los requerimientos de las aplicaciones Web actuales son difíciles de resolver:
  - Soporte de múltiples canales de presentación: clientes HTML, Java, WAP, portales de voz...
  - La funcionalidad a presentar al cliente depende del perfil del mismo
    - Puede necesitarse personalización
  - Los ciclos de desarrollo y puesta en producción son cortos
  - Hay gran cantidad de aplicaciones heredadas (legacy) ya desarrolladas
- Un modelo de arquitectura de aplicaciones que intenta resolver estos problemas es el Modelo-Vista-Controlador (MVC)
  - Recomendado para desarrollo en la JEE

MVC

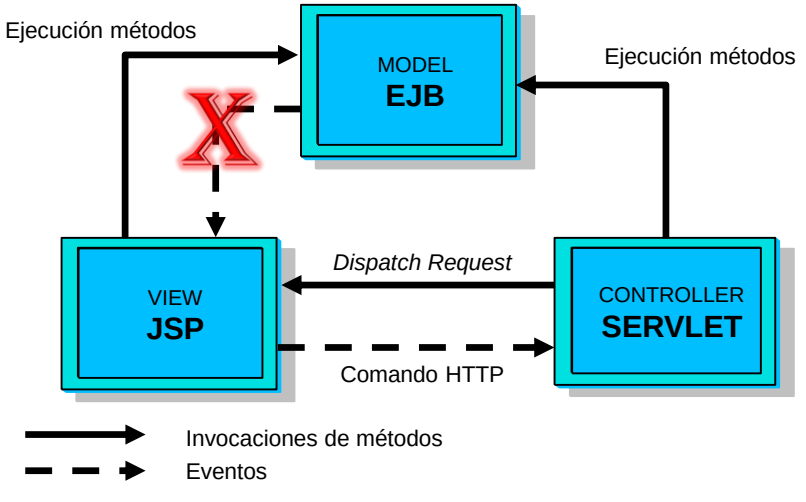


MVC

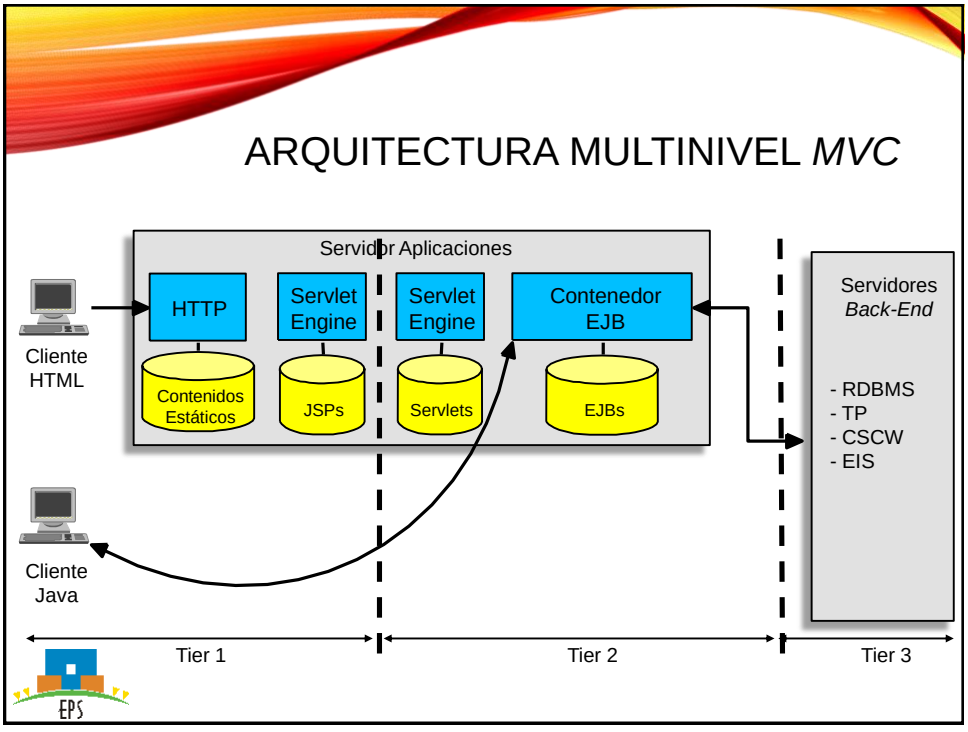
- Modelo:
  - Representa los datos de la aplicación y las funciones de negocio
  - Los encapsula para hacer transparente su manejo al resto de la aplicación
  - Permite tener la **funcionalidad** de la aplicación independiente, y en cualquier tipo de soporte, aunque sean sistemas heredados
- Vista:
  - Realiza la presentación de los datos del modelo
  - Accede al modelo y adapta los datos presentados al cliente final
  - Permite múltiples vistas dependiendo del medio de salida de los datos, sin necesidad de cambiar la lógica de la aplicación ni la navegación de la misma
- Controlador:
  - Recibe las interacciones del usuario y las traslada en acciones sobre el modelo
  - Decide la salida que es necesario presentar al usuario solicitándolo a la Vista
  - Permite realizar la navegación que se necesite para el usuario sin necesidad de alterar la vista ni el modelo
-

EPS

MVC EN JEE



EPS


ARQUITECTURA MULTINIVEL *MVC*
Servidor Aplicaciones
HTTP
Servlet Engine
Servlet Engine
Contenedor EJB
Contenidos Estáticos
JSPs
Servlets
EJBs
Cliente HTML
Cliente Java
Servidores *Back-End*
- RDBMS
- TP
- CSCW
- EIS
Tier 1
Tier 2
Tier 3
EPS