

# J2EE

Java 2 Platform, Enterprise Edition

© JAA 2013

## INSTALACIÓN

© JAA 2013

## Eclipse

- Descargar e instalar JDK:
  - <http://www.oracle.com/technetwork/java/javase/downloads/>
- Descargar y descomprimir Eclipse:
  - <http://www.eclipse.org/downloads/eclipse-packages/>
  - Eclipse IDE for Java EE Developers
- Descargar y descomprimir Apache Tomcat:
  - <http://tomcat.apache.org/>
- Configurar Eclipse:
  - Window → Preferences → Server → Runtimes Environments → Add
- En el workspace:
  - Perspectiva Java EE
  - Solapa inferior Server → Add Server
  - Seleccionar Tomcat → Finish
  - Doble clic en el servidor para configurar
  - Start the server

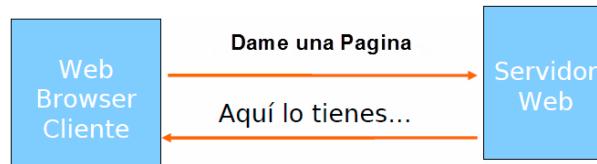
© JAA 2013

# INTRODUCCIÓN Y CONCEPTOS GENERALES

© JAA 2013

## Arquitectura y desarrollo de aplicaciones web

- Al principio de WWW y HTML el contenido de las páginas a las que accedíamos eran estáticas.
- Cada URL mostraba siempre la misma página.

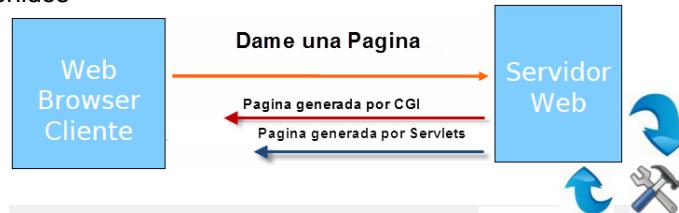


- Para mejorar todo esto, se introdujeron herramientas para poder dar más «dinamismo» a las páginas Webs (Scripts de Servidor)
  - ✓ CGI, FastCGI ( 1<sup>a</sup> generación )
  - ✓ Servlets, ASP ( 2<sup>a</sup> generación )
  - ✓ JSP ( 3<sup>a</sup> generación )

© JAA 2013

## Arquitectura y desarrollo de aplicaciones web

- Con estos nuevos mecanismos, nos permite la creación dinámica de contenidos



- Para complementar todo esto, aparecieron también herramientas en la parte cliente para tratamiento de la información (Script de Cliente)

- ✓ Lenguaje JavaScript
- ✓ Applets Java
- ✓ Formularios,
- ✓ Ajax



© JAA 2013

## Tecnologías Web



### Protocolo HTTP

- HTTP (HyperText Transfer Protocol) es un protocolo de aplicación para transferencia de hipertexto, basado en peticiones de URL
- Está implementado sobre TPC/IP y es el protocolo de comunicación en la Web
- Es un protocolo sin estado (stateless), basado en peticiones y respuestas.
- Define los tipos de peticiones que los clientes pueden enviar, el formato y estructura de las respuestas
  - ✓ HTTP 1.0 (RFC 1945) define GET, POST y HEAD
  - ✓ HTTP 1.1 (RFC 2068) añade OPTIONS, PUT, TRACE y CONNECT
- También define una estructura de meta datos, en forma de cabeceras

© JAA 2013

## Tecnologías Web



### Protocolo HTTP

- URL (Universal Resource Locator) = “apuntador/puntero” a un recurso particular, habitualmente en Internet
- Formato:
  - ✓ protocolo : servidor : puerto / archivo
  - ✓ Al pedir un recurso se indica el protocolo de comunicación, el host en el que se encuentra, el puerto por el que se atiende y el path del recurso dentro del servidor web
- Ejemplos:
  - ✓ <http://java.sun.com/products/servlet/index.html>
  - ✓ <mailto:alguien@servidor.com>
  - ✓ <file:///etc/fstab>

© JAA 2013

## Tecnologías Web



### Protocolo HTTP. Método GET

- Es el tipo de petición más simple y usado.
- Se suele usar para recuperar recursos estáticos del servidor.
- Formato:
  - ✓ `http://servidor/index.html` → GET /index.html HTTP/1.1
- Puede usarse para recuperar recursos dinámicos enviando parámetros al servidor:
  - ✓ - `http://www.servidor.com/contDinamico?param=valor`
- Los parámetros van codificados en la URL

© JAA 2013

## Tecnologías Web



### Protocolo HTTP. Método HEAD

- Se usa para recuperar metadatos de un recurso del servidor: tipo MIME, tamaño, fecha de última modificación...
- El servidor sólo devuelve cabeceras (nunca el cuerpo)
- Formato:
  - ✓ `HEAD /index.html HTTP/1.1`
- Se usa para implementar cachés de navegadores, informar al usuario del tamaño del recurso antes de intentar recuperarlo, etc...

© JAA 2013

## Tecnologías Web



### Protocolo HTTP. Método POST

- Se usa para mandar parámetros al servidor.
- Normalmente sirve para recuperar recursos generados dinámicamente
- Su funcionamiento es igual al de GET, sólo que los parámetros se envían en el cuerpo de la petición, es decir, no son visibles en la URL.
- Formato:  
✓ `http://servidor/index.html → POST /index.html HTTP/1.1`

© JAA 2013

## Tecnologías Web



### Protocolo HTTP. Respuestas

- Cuando el servidor envía una determinada respuesta, ésta es dividida en partes:
  - 1.- Información de la primera línea de respuesta:
    - protocolo código-de-estado descripción  
HTTP/1.1 200 OK
  - 2.- Después el servidor devolverá las cabeceras de la respuesta.
  - 3.- Y finalmente el cuerpo de la respuesta
- Después de enviar el cuerpo de la respuesta el servidor cierra la conexión del cliente.  
✓ Si el cliente quiere mantener abierta la conexión → Connection: KeepAlive.

© JAA 2013

## Tecnologías Web



### Protocolo HTTP. Respuestas y Códigos

- Rangos de respuesta del servidor:
  - ✓ 100-199 Información
  - ✓ 200-299 Petición procesada con éxito
  - ✓ 300-399 Petición redirigida
  - ✓ 400-499 Petición incompleta
  - ✓ 500-599 Errores del servidor

© JAA 2013

## Tecnologías Web

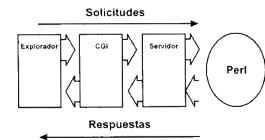


### Protocolo HTTP. Cabeceras

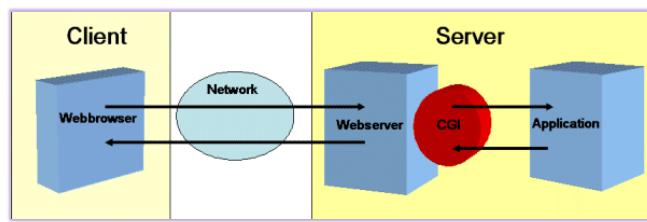
- Son meta datos que se envían tanto en la petición como en la respuesta.
- Hay cuatro tipos:
  - ✓ Generales:
    - Información no relacionada con el cliente, el servidor o el protocolo HTTP.  
( Cache-Control, Connection, Date, Transfer-Encoding )
  - ✓ Petición:
    - Formatos de documento preferidos por el navegador y parámetros para el servidor.  
( User-Agent, Accept, Accept-Charset, Authorization )
  - ✓ Respuesta:
    - Información del servidor.  
( Apache, WebSphere, etc )
  - ✓ Entidad:
    - Información sobre los datos transferidos entre el cliente y el servidor.  
( Content-Encoding, Content-Language, Content-Length, etc )

© JAA 2013

## Tecnologías Web. CGI



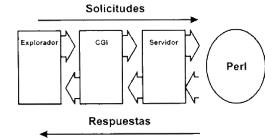
- CGI (Common Gateway Interface) es un script de Servidor que posibilita el envío de información desde el Servidor que no estuviera previamente definida en el Servidor (dinámica).
- Estas páginas son generadas de formas dinámicas por los programas CGI que está ubicados en el servidor.



- Los CGI's pueden estar implementados en múltiples lenguajes:
  - ✓ C/C++, Perl, Python, Java, etc.

© JAA 2013

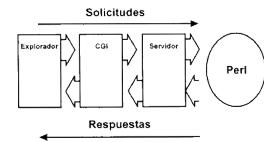
## Tecnologías Web. CGI



- Podemos pasar información a un script CGI es con una cadena de consulta concatenada al final de la URL.
- La cadena de consulta comienza por '?' y los parámetros van separados por '&'.
- Formato de parámetros: 'param=valor'
- Ejemplos:
  - <http://www.ncbi.nlm.nih.gov/blast/blast.cgi?Jform=0>
  - <http://foo/bar?param1=yo&param2=mismo&param3=hola>

© JAA 2013

## Tecnologías Web. CGI

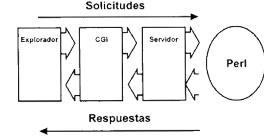


### Inconvenientes

- Bajo Rendimiento
  - ✓ Los programas CGI's se crean con lenguajes interpretados como Unix, Perl, python, etc
- Tiempo de Arranque
  - ✓ Los programas CGI's se cargan cada vez que alguien lo pide y no se mantienen en memoria.
- Mala comunicación entre CGI's
  - ✓ Cada programa CGI puede estar realizado en un lenguaje diferente.
- Baja Seguridad
  - ✓ La mayoría de los lenguajes de creación de los CGI's no aplican seguridad, aunque otros, de forma automática si lo hacen (Perl).
- No universalmente aceptados.

© JAA 2013

## Alternativas al CGI

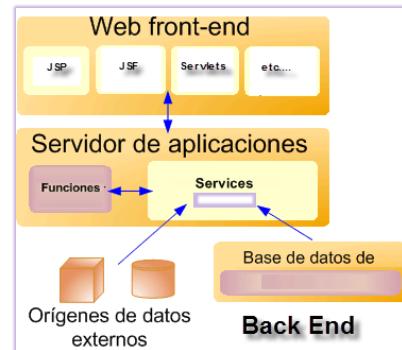


- CGI es la solución más antigua para contenido web dinámico
- Otras alternativas
  - Java Server Pages ([JSP: http://java.sun.com/products/jsp/](http://java.sun.com/products/jsp/))
  - PHP (<http://www.php.net/>)
    - ✓ □ PHP y JSP mezclan HTML y sentencias de un lenguaje en un mismo documento que es parseado por el servidor antes de ser entregado al cliente
  - Java Servlets (<http://java.sun.com/products/servlet/>)
    - ✓ Permite separar la lógica de presentación (HTML) de la de la aplicación
    - ✓ Siguen usando el mismo protocolo para las peticiones (HTTP)
  - Microsoft ASP

© JAA 2013

## Aplicaciones Java EE

- La plataforma Java, Enterprise Edition (Java EE), proporciona una plataforma basada en estándares para el desarrollo de aplicaciones web y empresariales.
- Estas aplicaciones están diseñadas, normalmente, como aplicaciones de varios niveles
  - ✓ Nivel de Front End
    - Entorno Web
  - ✓ Nivel Intermedio
    - Que proporciona la lógica de negocios, transacciones y Seguridad
  - ✓ Nivel de Back End
    - Proporciona conectividad a Sistemas de Almacenamiento (Bases de Datos).



© JAA 2013

## Aplicaciones Java EE

- Estas aplicaciones deben ser sensibles y escalares, para permitir un rápido crecimiento ante la demanda de los usuarios.
- La plataforma Java EE define diferentes componentes, que se despliegan en contenedores que proporcionan soporte de ejecución .
- Componentes de aplicaciones Java EE nunca interactúan directamente con otros componentes de la aplicación Java EE.
- Componentes usan protocolos y métodos del contenedor para interactuar entre sí.



© JAA 2013

## Componentes vs. objetos

- Un componente se caracteriza por:
  - Ser una unidad de despliegue independiente
    - ✓ Encapsula sus características constituyentes respecto a su entorno
      - Las terceras partes no pueden acceder a los detalles de construcción del componente
  - No se implanta de manera parcial
  - Ser una unidad de composición
    - ✓ Con componentes posiblemente desarrollados por otros
    - ✓ Debe ser suficientemente autocontenido
    - ✓ Especificaciones claras de lo que requiere y de lo que proporciona
    - ✓ Interacciona con su entorno a través de interfaces bien definidas
  - No tener estado persistente
    - ✓ Un componente no se distingue de otras copias del mismo
    - ✓ Excepto atributos no funcionales como el número de serie
- Por tanto, en un proceso se puede decir si hay o no un componente, pero no varias instancias del mismo

© JAA 2013

## Componentes vs. objetos

- Un objeto se caracteriza por:
  - Ser una unidad de instancia; tienen una identidad única
    - ✓ No se instancia de manera parcial
    - ✓ La identidad es única y no cambia durante la vida del objeto
  - Tener un estado
    - ✓ Se crea con un estado inicial que evoluciona durante la ejecución
  - Encapsular su estado y comportamiento
    - ✓ Que está definido bien por una clase o por un objeto prototípico

© JAA 2013

## Interfaces

- Puntos de acceso a los componentes
  - Permite a los clientes acceder a los servicios proporcionados por un componente
- Un componente puede tener varias interfaces
  - Una por cada punto de acceso: uso, administración, configuración, ...
  - Pero no conviene tener varias interfaces similares o redundantes
- La especificación de las interfaces es un contrato
  - El respeto de este contrato por cliente y componente asegura el éxito de la interacción

© JAA 2013

## Contenedores

- Un contenedor es un proceso donde se ejecutan los componentes
  - Gestiona los componentes de la aplicación
    - ✓ Ciclo de vida
  - Proporciona acceso a servicios de la plataforma
    - ✓ Seguridad, transacciones, persistencia, conectividad, etc.
- El desarrollador tiene que especificar
  - Los componentes de la aplicación
    - ✓ Servlets
    - ✓ JSPs (JavaServer Pages)
    - ✓ JSFs (JavaServer Faces)
    - ✓ EJBs (Enterprise Java Beans)
- Los descriptores de despliegue (deployment)
  - Ficheros XML que describen los componentes de aplicación

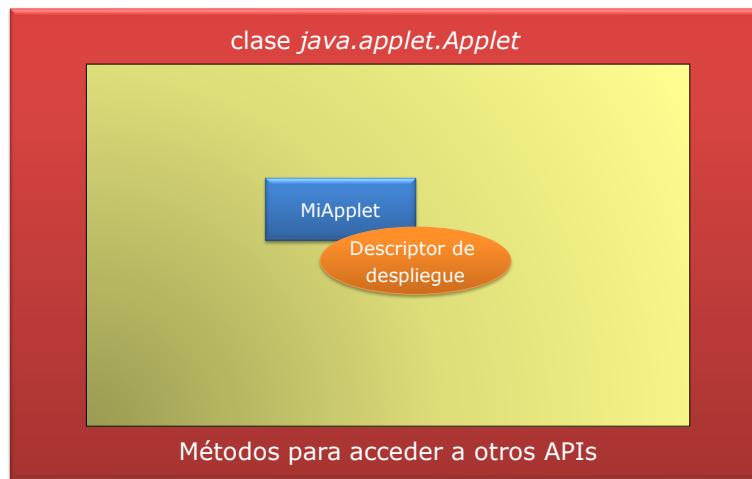
© JAA 2013

## Arquitectura de un contenedor



© JAA42013

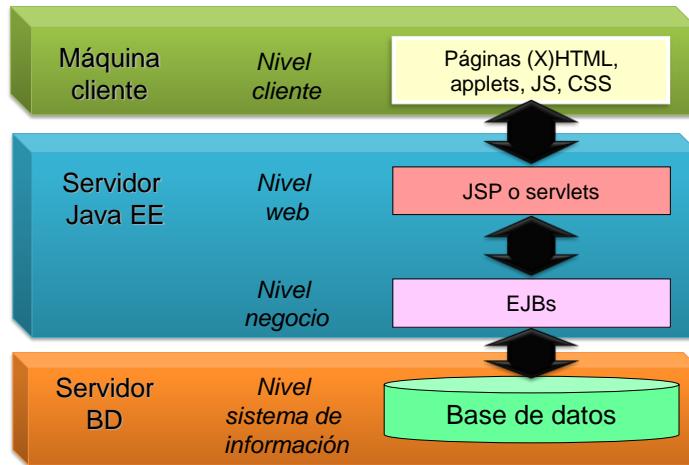
## Contenedor de applets



© JAA42013

## Arquitectura multi-nivel (multi-tier)

- Este modelo propicia aplicaciones web en 4 niveles:



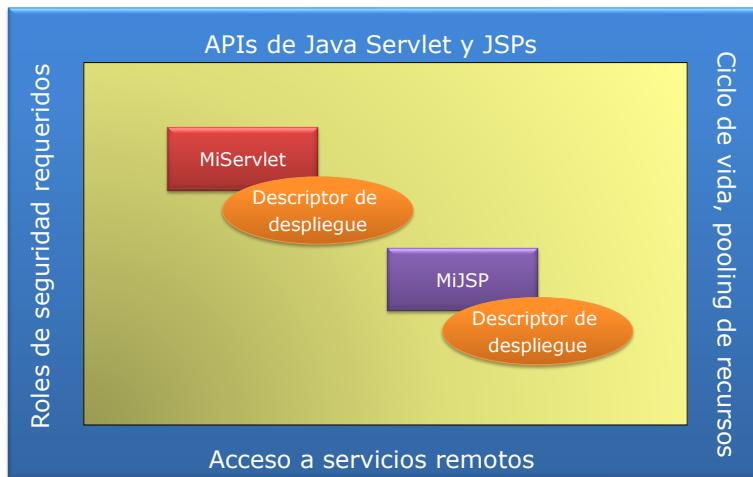
© JAA 2013

## Componentes web en Java EE

- Java Servlets
  - Clases escritas en Java que procesan peticiones HTTP y construyen respuestas
- Java Server Pages (JSP)
  - Documentos basados en texto que contienen dos tipos de texto
    - una plantilla de datos estática que puede expresarse en un formato como (X)HTML o XML
    - elementos JSP que determinan cómo la página construye el contenido dinámico
- Java Server Faces (JSF)
  - Componentes de interfaz de usuario para aplicaciones web
- Los clientes en Java EE son
  - Clients web: navegadores web, páginas web, applets
  - Aplicaciones de cliente

© JAA 2013

## Contenedor Web



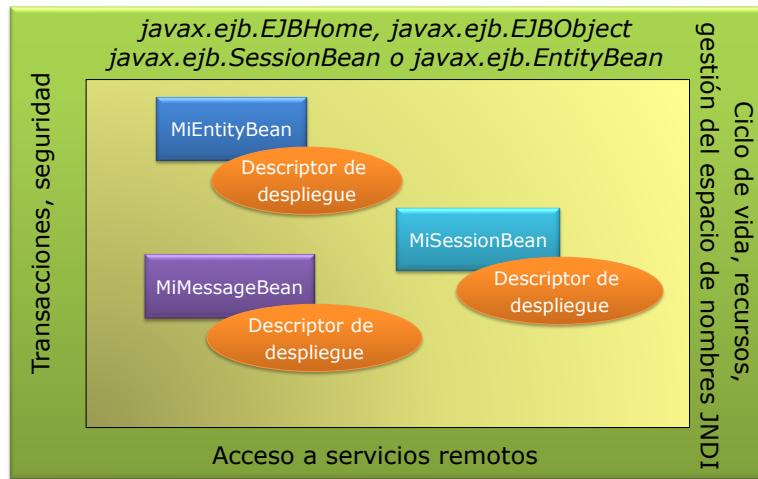
© JAA42013

## Componentes de negocio en Java EE

- Lógica que resuelve las necesidades de un determinado dominio de aplicación
- Enterprise beans (EJBs)
  - Pueden procesar datos recibidos del lado cliente y enviarlos al nivel de sistema de información para su almacenamiento
  - Pueden recuperar datos del sistema de información, procesarlos y enviarlos al cliente
- Tipos de EJBs
  - Bean de sesión
    - ✓ Una conversación con un cliente
  - Bean dirigido por mensajes
    - ✓ Permite que un componente de negocio pueda recibir mensajes asíncronamente, normalmente con el Java Message Service (JMS)

© JAA42013

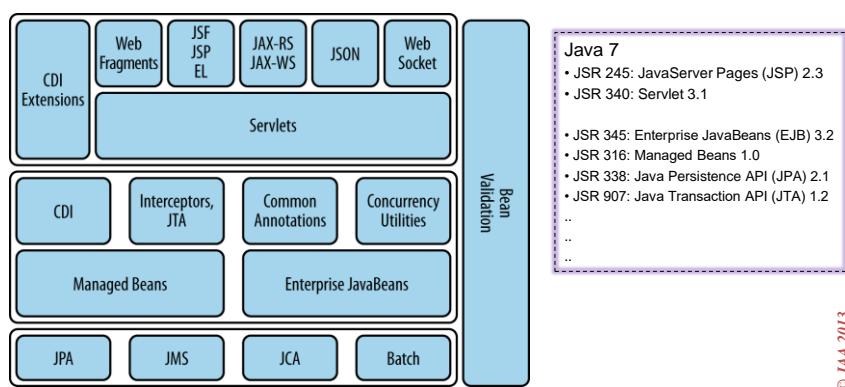
## Contenedor de EJBs



© JAVA 2013

## Aplicaciones Java EE

- Este modelo de contenedores permite una mayor independencia y transparencia a la hora de diseñar aplicaciones JEE.
- Cada componente de la plataforma se define en una especificación separada del resto.



© JAVA 2013

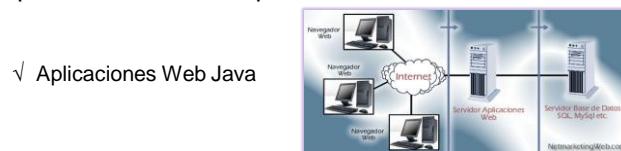
## Servicios Java EE

- Cada contenedor Java EE proporciona servicios a los componentes
  - Java Naming Direct Interface (JNDI)
  - Java Persistence API (JPA)
  - Java Database Connectivity API (JDBC)
  - Java Transaction API (JTA)
  - Java Message Service (JMS)
  - JavaMail
  - Java Beans Active Framework (JAF)
  - Java EE Connector Arquitecture
  - Java Authentication and Authorization Service (JAAS)
  - Java API for XML Procesing (JAXP)
  - SOAP with Attachments API for Java (SAAJ)
  - Servicios Web (JAX-WS)
  - Java API for RESTful Web Services (JAX-RS)

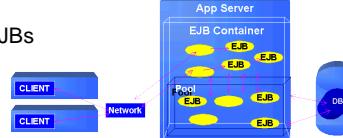
© JAA 2013

## Estructuras y Descriptor de despliegue

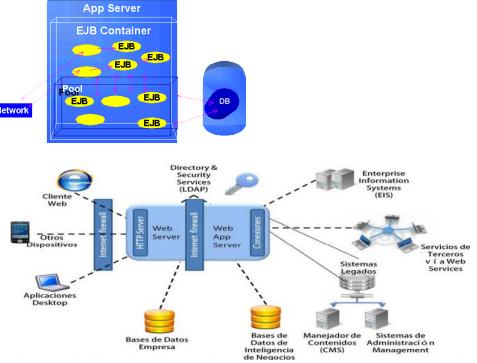
- Las aplicaciones Java se pueden dividir en 3 modelos:



✓ Objetos distribuidos EJBs



✓ Aplicaciones Empresariales



© JAA 2013

## Ensamblado y despliegue de componentes JEE

- Los componentes se instalan en contenedores desde los que pueden utilizar los servicios de la plataforma
- El proceso de ensamblado de los componentes requiere especificar el soporte del servidor J2EE
  - Seguridad: usuarios autorizados
  - Modelo de gestión de transacciones: relaciones entre métodos que constituyen una transacción (tratados como una unidad)
  - Java Naming and Directory Interface (JNDI): acceso a servicios de nombres y directorio
  - Conectividad remota: permite que los clientes invoquen métodos en los EJBs como si estuvieran en la misma máquina virtual

© JAA 2013

## Composición de módulos en aplicaciones

- Una aplicación Java EE se puede entregar como ficheros:
  - Java Archive (JAR)
  - Web Archive (WAR) file
  - Enterprise Archive (EAR)
    - ✓ Módulos Java EE
    - ✓ Descriptor de despliegue (documento XML con extensión .xml)

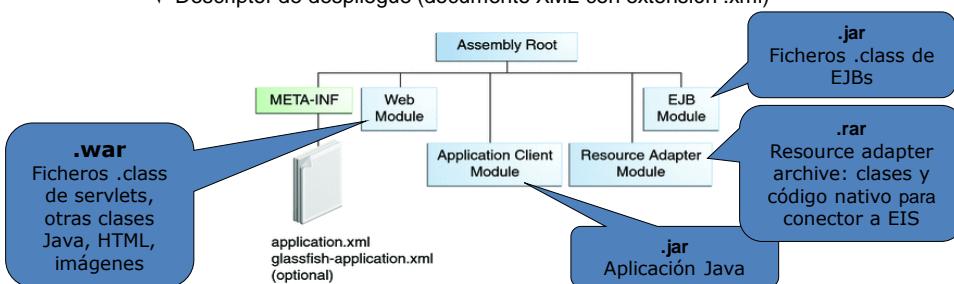
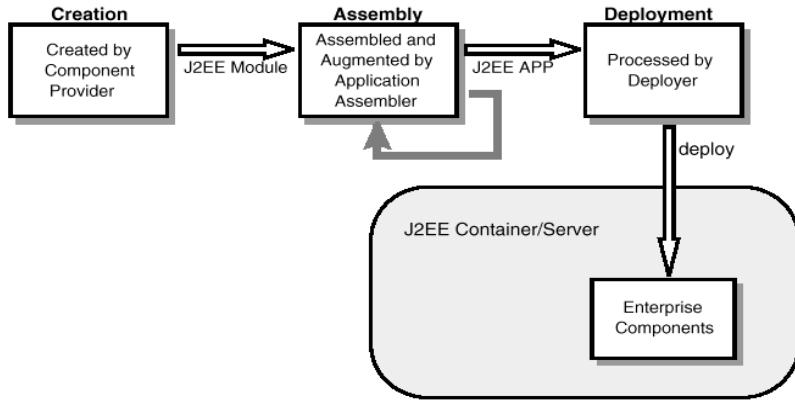


Figura 1-6 EAR File Structure, de *The Java EE 6 Tutorial* (2013).  
<http://docs.oracle.com/javaee/5/tutorial/doc/bnabo.html>

© JAA 2013

## Ciclo de vida de una aplicación Java EE

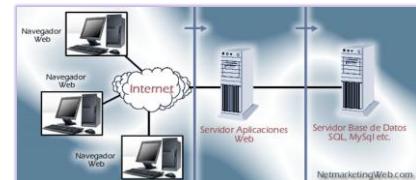


Fuente: Sun Microsystems, Inc., J2EE Connector Architecture Specification

© JAA 2013

## Aplicaciones Web Java

- Una aplicación Web Java es una Colección de recursos tales como
  - Jsp
  - Servlets
  - Ficheros Html
  - Imágenes
  - etc...
- Ubicados en un URI específico.
- Su estructura está compuesta de 2 partes:
  - ✓ Directorio privado WEB-INF
    - Contiene recursos que no son descargable por parte del cliente.
  - ✓ Directorio Público
    - Directorio que contiene los recursos públicos



© JAA 2013

## Aplicaciones Web Java

- Ejemplo: miaplicación\

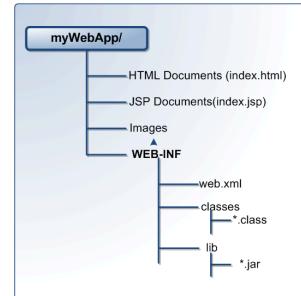
Público

```
Index.html
login.jsp
images\ logo.gif
doc\ tutorial.pdf
```

WEB-INF\

Privado

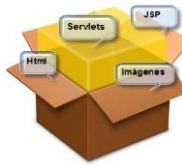
```
web.xml (Deployment Descriptor)
clases\ ServletCompras.class
lib\ cualquierOtraApi.jar
```



- Toda Aplicación Web Java puede ser empaquetada en un fichero WAR (Web Application aRchive)

✓ Permiten empaquetar en una sola unidad aplicaciones web java completas.

- Servlets y JSPs
- Contenido estático
  - Html
  - Imágenes
  - etc.)
- Otros recursos web

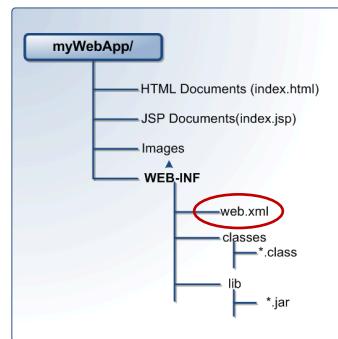


© JAA 2013

## Aplicaciones Web Java

### Características de los WAR

- Son una extensión del archivo JAR
- Se introdujeron en la especificación 2.2 de los servlets.
- Multiplataforma y MultiVendor
- Simplifican el despliegue de aplicaciones web.
  - ✓ Facilidad de instalación
  - ✓ Un solo fichero para cada servidor en un cluster.
- Seguridad
  - ✓ No permite el acceso entre aplicaciones web distintas
- Disponen de un fichero especial denominado «Descriptor de Despliegue» web.xml

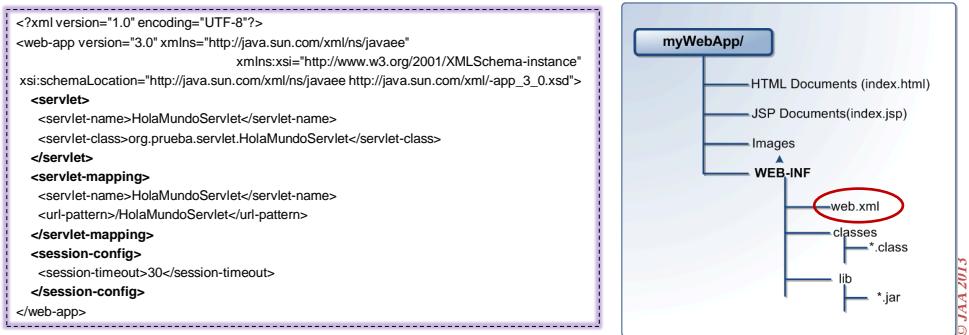


© JAA 2013

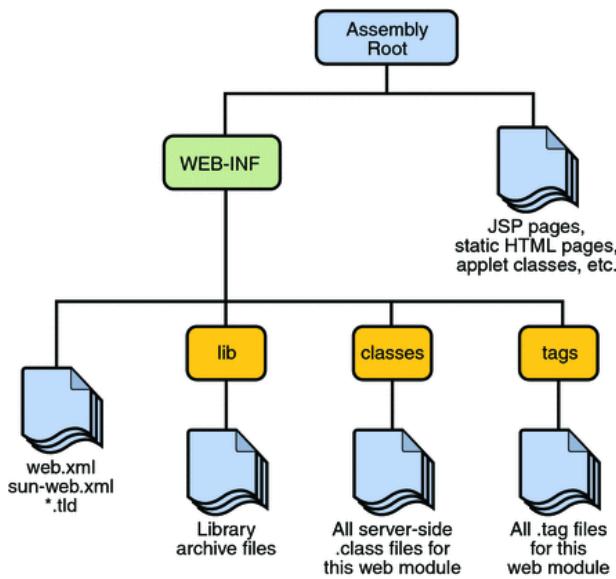
## Aplicaciones Web Java. WEB.XML

### Descriptor de despliegue

- Es un componente de aplicaciones J2EE que describe cómo se debe desplegar (o implantar) una aplicación web.
- Permite a un Servidor de Aplicaciones dirigir la publicación de un módulo o aplicación.
- Debe ser llamado *web.xml*, y debe ser colocado en un subdirectorio llamado *WEB-INF*, directamente debajo de la raíz de la aplicación web.

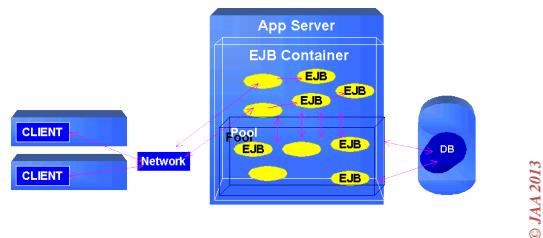


## Composición de módulos en aplicaciones



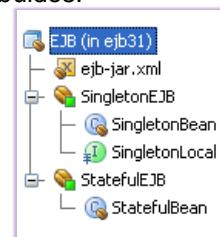
## Objetos distribuidos EJBs

- Los EJB proporcionan un modelo de componentes distribuido del lado del servidor.
- El objetivo de los EJB es dotar al programador de unos elementos que le permita abstraerse de los problemas generales de una aplicación empresarial (conurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí.
- El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.



## Objetos distribuidos EJBs

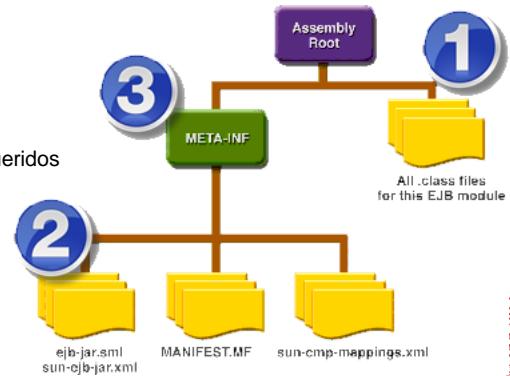
- Los EJBs se agrupan en paquetes para poder ser distribuidos.
- Existen diferentes tipos de EJBs:
  - Sesión
  - Entidad
  - Mensajes - MSB
- Todos los EJB's se distribuyen a través de ficheros EJB-JAR con estructura determinada.
- Estos ficheros facilitan el despliegue de los EJBs en cualquier servidor de aplicaciones compatible J2EE.



© JAA 2013

## Estructura de un EJB-JAR

- /\*.class
  - Bajo este directorio base se encuentran las diversas *clases* que conforman a un EJB
- /META-INF/ejb-jar.xml
  - Descriptor de despliegue
- /META-INF/\*
  - Otros archivos de configuración requeridos por el contenedor de EJBs



## Descriptor EJB-JAR.xml

- El fichero /META-INF/ejb-jar.xml define como debe realizar el despliegue del EJB en el contenedor.
- En él se dan de alta y se declaran
  - ✓ EJBs
  - ✓ Parámetros del contexto
  - ✓ Relaciones
  - ✓ Políticas transaccionales
  - ✓ etc

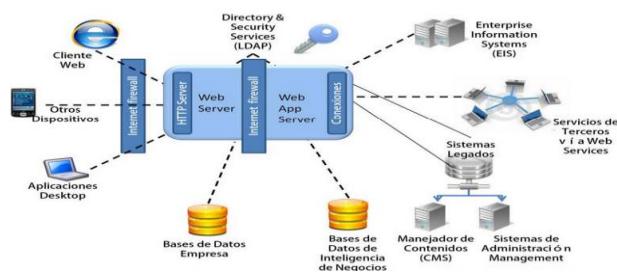
```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
 "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <description><![CDATA[No Description.]]></description>
  <display-name>Generated by XDoclet</display-name>
  <enterprise-beans>
    <session>...</session>
    <entity>...</entity>
    <message-driven>...</message-driven>
  </enterprise-beans>
  ...
</ejb-jar>
  
```

© JAA 2013

## Aplicaciones empresariales J2EE

- Las aplicaciones empresariales es una reunión de las 2 anteriores.
- App J2EE = Aplicación/es web java empaquetadas en WAR
  - + Objetos distribuidos EJB empaquetados en EJB-JAR
- Son distribuidas mediante ficheros EAR (Enterprise Archive)

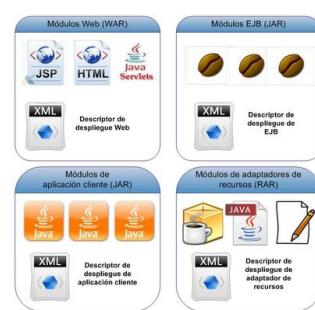


© JAA 2013

## Ficheros EAR

- Los archivos EAR (Enterprise Archive) es un formato utilizado en la arquitectura JEE para desplegar de manera coherente y simultánea varios módulos en un servidor de aplicaciones.
- Contiene los descriptores de despliegue que describen como desplegar los módulos contenidos en el paquete EAR.

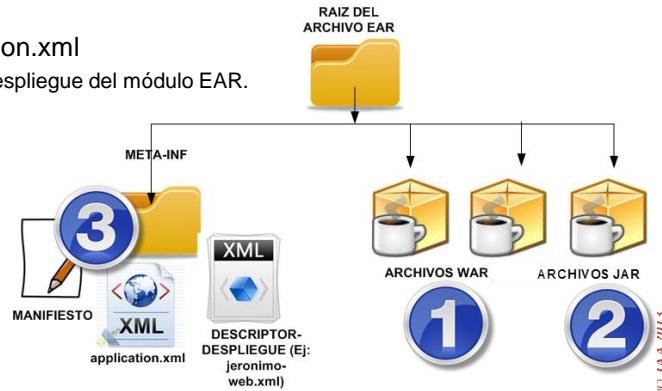
- En su interior, pueden convivir
  - Ficheros WAR
  - Ficheros EJB-JAR
  - Ficheros JAR
  - Ficheros RAR



© JAA 2013

## Estructura de un EAR

- /\*.war
  - ✓ Archivos war.
- /\*.jar
  - ✓ Archivos (ejb) jar.
- /META-INF/application.xml
  - ✓ Descriptor de despliegue del módulo EAR.



## Descriptor Application.xml

- El fichero /META-INF/application.xml enumera los archivos JAR en el EAR y le dice al servidor de Aplicaciones que archivos buscar y dónde.
- En él se dan de alta y se declaran
  - ✓ EJBs
  - ✓ Parámetros del contexto
  - ✓ Relaciones
  - ✓ Políticas transaccionales
  - ✓ etc

```
<?xml version="1.0" ?>
<!DOCTYPE application PUBLIC
"-//Sun Microsystems, Inc. //DTD J2EE Application 1.2//EN"
"http://java.sun.com/j2ee/dtds/application_1_2.dtd">
<application>

<display-name>Curso Piloto.</display-name>
<description>Prácticas del curso de desarrollo web</description>

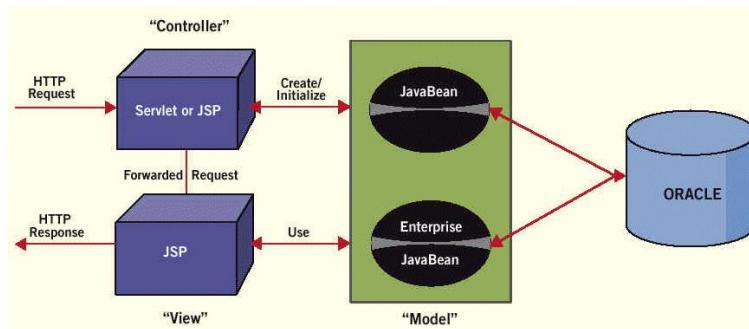
<module>
<web>
<web-uri>appblank.war</web-uri>
<context-root>/appblank</context-root>
</web>
</module>
</application>
```

# INTRODUCCIÓN A SERVLETS Y JSP

© JAA 2013

## Introducción a JSP y Servlets

- Cuando hablamos de Servlets o JSP (Java Server Page), estamos refiriéndonos a Scripts de Servidor.
- Son tecnologías que se ejecutan directamente en el Servidor de Aplicaciones sin intervención del cliente.



© JAA 2013

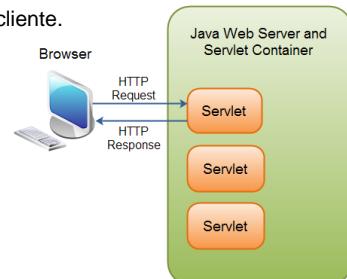
## ¿Qué es un Java Servlet?

- Los servlets son programas desarrollados en Java utilizados para extender la funcionalidad de cualquier servidor de aplicaciones WEB.
- Son habitualmente utilizados para escribir aplicaciones Web y desplegados en un servidor Web (*tomcat, Weblogic, Jboss, etc*)
- Las aplicaciones web creadas, pueden ofrecer un contenido ESTATICO y/o DINAMICO.
- Servlets son especialmente adecuados para la creación de páginas web dinámicas.

© JAA2013

## ¿Qué es un Java Servlet?

- Los Servlets son la alternativa Java a los CGIs.
- Actúan como capa intermedia entre:
  - Petición proveniente de un Navegador Web u otro cliente HTTP
  - Bases de Datos o Aplicaciones en el servidor HTTP
- Son aplicaciones Java especiales, que extienden la funcionalidad del servidor HTTP, dedicadas a:
  - Leer, Extraer, Generar los datos enviados por/para el cliente.
  - Formatear los resultados en un documento HTML.
  - Establecer los parámetros HTTP adecuados
  - Enviar el documento final al cliente



## ¿Qué es un Java Servlet?

- Los objetos servlets cumplen los siguientes requisitos:
  - Utilizan el interface SAPI “Servlet Application Programming Interface” para que todas las peticiones HTTP sean procesadas por esta clase Java (independiente del servidor).
  - Los Servlets son manejados dentro del contener de servlets con una arquitectura simple.
  - El contenedor provee el entorno de ejecución para todos los servlets basados en los anteriores requisitos.
  - Disponibles para la gran mayoría de servidores web.
  - Son independientes de la plataforma y del servidor.

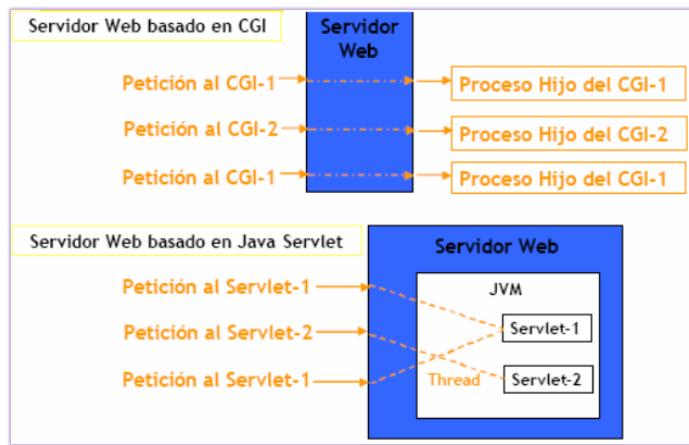
© JAA 2013

## Ventajas de un Java Servlet frente a un CGI?

- **Eficiencia.**
  - Cada petición por parte de un cliente crea un hilo, no un nuevo proceso como ocurría con los CGIs tradicionales.
- **Potencia**
  - Son programados en Java, por lo que se puede emplear todas las clases y herramientas disponibles para esta plataforma.
- **Seguridad**
  - Controlada por la máquina virtual de Java.
  - La mayoría de problemas de seguridad encontrados en los CGIs no aparecen en los Servlets.
- **Portabilidad**
  - Puede ser utilizados sobre cualquier SO. y en la mayoría de servidores Web.

© JAA 2013

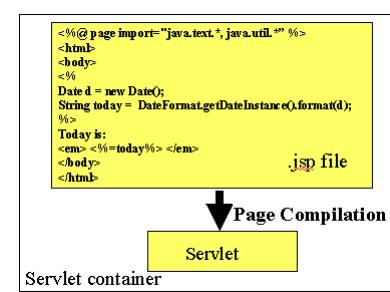
## Ventajas de un Java Servlet frente a un CGI?



© JAA 2013

## ¿Qué es JSP?

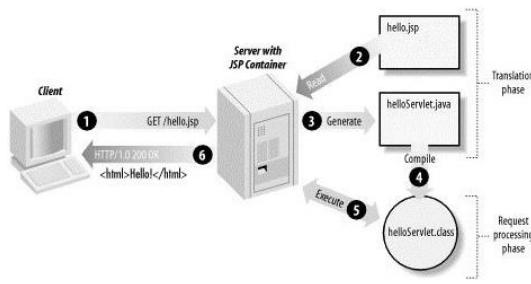
- Los JSP es una alternativa java a la generación de contenidos de forma dinámica en el lado del servidor.
- Un JSP es un componente del lado del servidor Web que se puede utilizar para generar páginas web dinámicas, pero difiere de los Servlets en:
  - Servlets generan código HTML de código Java.
  - JSP incrustar código Java en HTML estático.
- El código Java queda embebido dentro del código HTML de forma similar a PHP o ASP.



© JAA 2013

## ¿Qué es JSP?

- Ahora nos permite realizar una «Separación de Roles»
  - Con un servlet, el programador Java debe generar todo el HTML.
  - Con un JSP:
    - ✓ Separación de código HTML
    - ✓ Utilización de JavaBeans
    - ✓ Utilización de bibliotecas de etiquetas JSP
- JSP es en el fondo una tecnología equivalente a los servlets.
  - Las páginas JSP se traducen en servlets que ejecuta el servidor en cada petición.



© JAA 2013

## ¿Cuáles son las ventajas de JSP?

- Frente a CGI.
  - Seguridad : Entorno de ejecución controlado por la JVM
  - Eficiencia: Cada nueva petición es atendida por un hilo, no por un nuevo proceso
- Frente a PHP
  - Lenguaje más potente para la generación dinámica
    - ✓ Lenguaje de script orientado a objetos (Java)
  - Mejores herramientas de desarrollo y soporte
- Frente a ASP
  - Mejores rendimientos: Código compilado, no interpretado (SDK 1.4 o sup)
  - Independiente de la plataforma: Portable a múltiples servidores y SO.
  - Lenguaje más potente para la generación dinámica (Java)

© JAA 2013

## ¿Cuáles son las ventajas de JSP?

- Frente a Servlets Puro.
  - La lógica de negocio y la presentación están más separados.
  - Simplifican el desarrollo de aplicaciones Web
    - ✓ Más conveniente para crear HTML (no es necesario println).
  - Soporte para reutilizar software a través de JavaBeans y etiquetas adaptadas.
  - Puede utilizarse herramientas estándar (p.e. Homesite)
  - Recompila automáticamente las modificaciones en las páginas jsp
  - No es necesario ubicar las páginas en un directorio especial
    - ✓ /srv/www/tomcat/base/webapps/ROOT/pagina.jsp
  - La URL tampoco es especial.
    - ✓ http://www.uv.es/pagina.jsp

© JAA 2013

## Servlet Hola Mundo

- Crear Web\Dynamic Web Project
- En Java Resources → New Servlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE html><html>");
    out.println("<head><title>Curso</title></head>");
    out.println("<body><h1>Hola mundo</h1></body>");
    out.println("</html>");
}
```

- Run on server

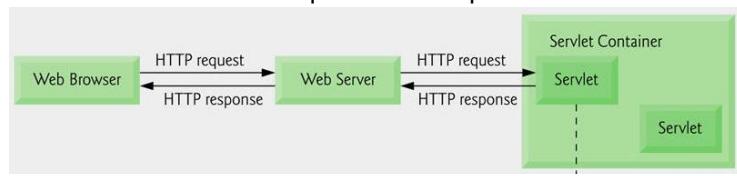
© JAA 2013

# SERVLETS

© JAA 2013

## Introducción a la tecnología Servlet

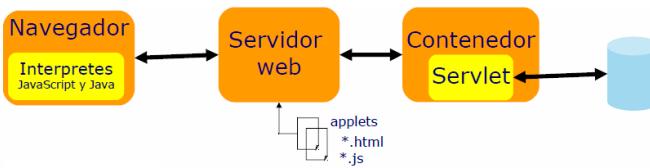
- Servlets se definen como la especificación JSR 340.  
(<http://jcp.org/aboutJava/communityprocess/final/jsr340/>)
- Un servlet es un componente web alojado en un contenedor de servlets que genera contenido dinámico.
- Los clientes web interactúan con un servlet usando un patrón de solicitud / respuesta.
- El contenedor de servlets es responsable del ciclo de vida del servlet, recibe peticiones y envía las respuestas, y realiza cualquier otra codificación / decodificación requerida como parte de eso.



© JAA 2013

## Introducción a la tecnología Servlet

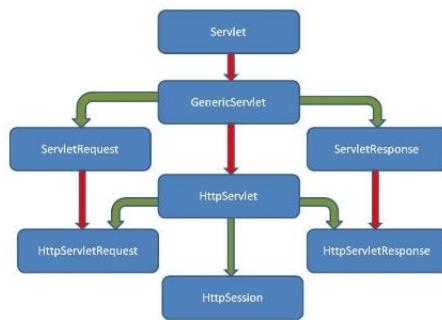
- Tareas del contenedor:
  - ✓ Proporcionar servicios de red para establecimiento de peticiones y respuestas HTTP.
  - ✓ Decodificar y codificar peticiones/repuestas en formato MIME
  - ✓ Configurar los servlets en función de los descriptores de despliegue (web.xml)
  - ✓ Gestionar el ciclo de vida de los servlets
- El contenedor web debe implementar las funciones del servidor web (según especificación Servlet)
- Esto se consigue habitualmente conectando servidor web + contenedor (v.g. apache + tomcat)



© JAA 2013

## Introducción a la tecnología Servlet

- Para que un Servlet pueda ser gestionado dentro de un Contenedor correcto debe de cumplir uno de los 2 siguientes interfaces:
  - `javax.servlet.Servlet`.
  - `javax.servlet.http.HttpServlet o javax.servlet.GenericServlet`
- Estos interfaces proporcionan soporte para:
  - Mantener el ciclo de vida de un Servlet
  - Acceso al contexto del Servlet
  - Utilidades del Servlet
  - Soporte para HTTP



© JAA 2013

## Introducción a la tecnología Servlet



- **GenericServlet**
  - CLASE que permite la implementación de programa no sólo para HTTP sino para otros servicios como FTP, WHOIS, TELNET, etc
- **ServletContext (\*)**
  - INTERFACE que permite a los Servlets obtener información del entorno.
- **ServletConfig(\*)**
  - INTERFACE que permite definir métodos para pasar información al Servlet.
- **ServletRequest(\*)**
  - INTERFACE que permite obtener información sobre las peticiones del cliente.
- **ServletResponde(\*)**
  - INTERFACE que permite enviar su información al cliente.
- **HttpServletRequest → Deriva de ServletRequest**
- **HttpServletResponse → Deriva de ServletResponde**

© JAA 2013

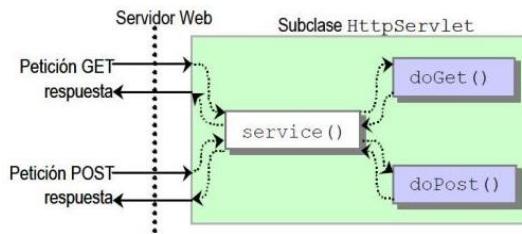
## Servlet. Características

- A simple vista es muy parecido a un Applet.
- No tenemos ningún método *main*, se carga y se ejecuta de forma automática.
- Se carga una vez en memoria (*init*) y se mantiene en ella hasta que es lo destruimos (*destroy*).
- No tienen interface gráfico.
- Al implementar un interface, hay métodos que pueden estar implementados o no.
  - ✓ Service, init, destroy, etc

© JAA 2013

## Servlet. Peticiones de Formulario

- Como vimos en el capítulo anterior, un formulario WEB puede enviar peticiones de 2 modos diferentes:
  - GET: Paso de parámetros en la propia URL de acceso al servicio proporcionado por el formulario.
  - POST Es idéntico al GET pero los parámetros no van en la línea del URL, sino en otra parte del mensaje.
- Todas las peticiones realizadas por los clientes, son tratadas por el método `service()` que posteriormente, las redirige a los métodos `doGet` o `doPost` según corresponda.



© JAA 2013

## Servlet. Peticiones de Formulario

- Habitualmente sólo se implementa uno de los métodos de petición de formulario (`doGet` o `DoPost`) y el no implementado se redirige al otro.

```

protected void doGet( HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println
    ("<html> <head><title>Hola Mundo</title></head>" +
    "<body> <h1>Hola Mundo</h1> </body></html>");
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */

protected void doPost( HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    doGet(request, response);
}

```



© JAA 2013

## Servlet. Peticiones de Formulario

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

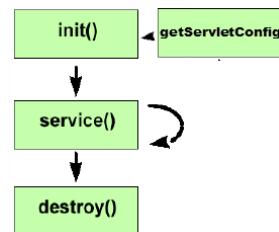
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println
    ("<html> <head><title>Hola Mundo</title></head>" +
    "<body> <h1>Hola Mundo</h1> </body></html>");
}
```

request	Objeto petición realizada por el cliente
response	Objeto respuesta para el envío de información del Servlet al Cliente.
.setContentType	Indicación del tipo de respuesta del Servlet ( Texto / Html )
out	Variable creada como flujo de salida/escritura en el Objeto Respuesta. Es utilizada para introducir el texto en HTML y que aparezca en el navegador

© JAA 2013

## Ciclo de vida de un Servlet

- El ciclo de vida de un Servlet lo controla el Contener en el que se ha desplegado.
- El Contendor utiliza una series de métodos que están implementados por defecto en el Servlet para controlar su ciclo de vida:
  - init()
  - service()
  - destroy()
  - getServletConfig()
- Estos métodos pueden ser «redefinidos» para que nuestro Servlet tenga un tratamiento especial.

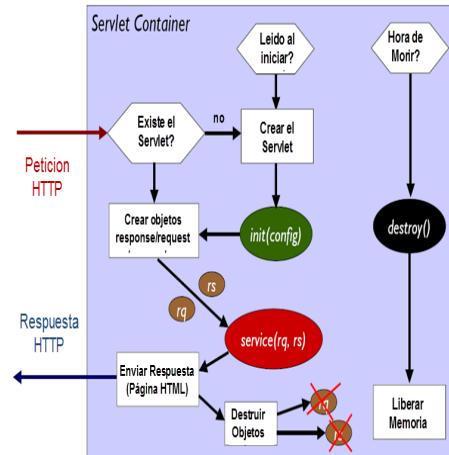


© JAA 2013

## Ciclo de vida de un Servlet

- Cuando una petición es realizada, el contenedor realiza las siguientes operaciones:

1. Comprueba si existe el Servlet en memoria
  - ✓ Si no existe:
    - Carga la clase del Servlet
    - Crea una Instancia del Servlet
    - Inicializa la instancia invocando al método *init()*
2. Se invoca al método *service()*
  - ✓ A este método se les pasan los objetos *request* y *response*, previamente creado
  - ✓ El servlet utiliza estos objetos para tratar la petición y generar la respuesta.
3. Cuando el Servlet ya no es necesario, invoca al método *destroy()* para su destrucción de memoria



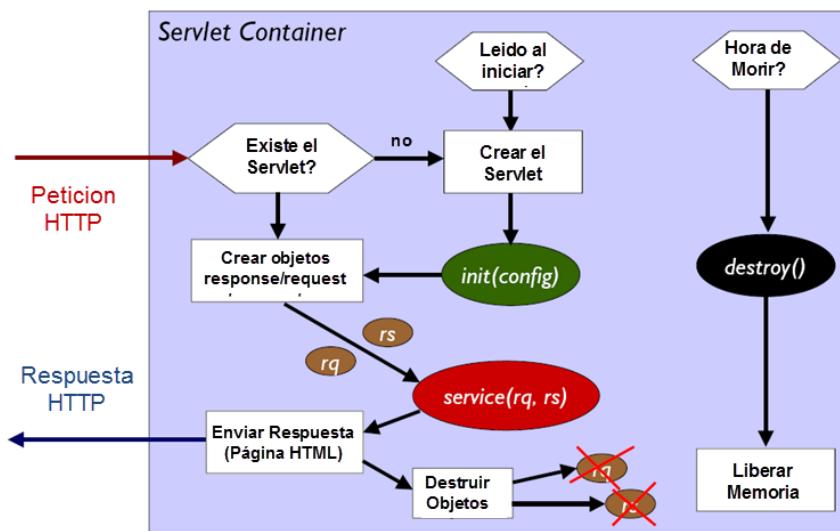
© JAA 2013

## Ciclo de vida de un Servlet

Métodos	Descripción
Init ( ServletConfig Config )	<ul style="list-style-type: none"> <li>- Método utilizado en el arranque de un Servlet sólo la primera vez.</li> <li>- Ideal para inicializar atributos y recursos del Servlets</li> </ul>
Config = getServletConfig()	<ul style="list-style-type: none"> <li>- Devuelve el objeto <i>ServletConfig</i> que contiene los parámetros de arranque e inicialización del Servlet .</li> <li>- Los parámetros de arranque se definen en el descriptor de despliegue</li> </ul>
Service ( ServletRequest req, ServletResponse res )	<ul style="list-style-type: none"> <li>- Este método es lanzado cada vez que hay una petición de servicio del Servlet.</li> <li>- Dependiendo del tipo de petición (GET, POST) se ejecutan los métodos <i>doGet</i> o <i>doPost</i></li> <li>- No se suele sobrescribir este método.</li> <li>- Puede ser invocado de forma concurrente → threads</li> </ul>
destroy()	<ul style="list-style-type: none"> <li>- Sólo se ejecuta una vez, al eliminar el Servlet.</li> <li>- Debe de ser ejecutado en Thread, por la posibilidad de otros Hilos en ejecución</li> </ul>

© JAA 2013

## Ciclo de vida de un Servlet



© JAA 2013

## Ciclo de vida de un Servlet

### NOTAS

- El servlet se instancia al iniciarse el contenedor o al recibir la primera petición (configurable en web.xml).
- Al contrario que otras tecnologías sólo existe una instancia del servlet para manejar todas las peticiones.
- La especificación asegura que el contenedor no llamará a ningún otro método antes de que el método `init()` haya terminado.
- Si hay error durante la gestión de una petición por parte del Contenedor, se lanzará una excepción del tipo `ServletException` o `IOException` (*aunque el contenedor seguirá intentando mandar otras peticiones al servlet*)
- La especificación asegura que:
  - El contenedor no llamará a ningún método del servlet después de `destroy`
  - El contenedor no intentará destruir el servlet antes de que `destroy` haya terminado

© JAA 2013

## Ciclo de vida de un Servlet

### OTROS METODOS

- `getServletInfo()`
  - ✓ Este método devuelve un String con información sobre el servlet.
  - ✓ Usado por las herramientas de gestión del contenedor para mostrar al administrador
- `getServletConfig()`
  - ✓ Es para uso interno del servlet y debe devolver el objeto `ServletConfig` que se recibió en `init()`.

© JAA 2013

## ServletConfig

- Es un interface permite la inicialización del servlet con parámetros.
- Los parámetros se definen en el descriptor de despliegue mediante la etiquete `<init-param>`

```
<servlet>
    <servlet-name>ServletSaludo </servlet-name>
    <servlet-class>ServletSaludo </servlet-class>
    <init-param>
        <param-name>usuario </param-name>
        <param-value>jose </param-value>
    </init-param>
</servlet>
```
- Para utilizarlo deberemos sobrescribir el método `init()`, para pasarle el objeto de configuración del Servlet.

```
public void init ( ServletConfig config ) throws ServletException {
    Enumeration enu=config.getInitParameterNames();
    super.init(config);
    :
}
```

© JAA 2013

## ServletConfig

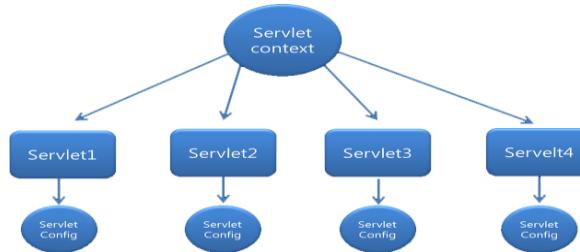
Métodos	Descripción
a.getInitParameter( String )	- Devuelve una cadena que contiene el valor del parámetro de inicialización llamado, o null si el parámetro no existe.
a.getInitParameterNames()	- Devuelve los nombres de los parámetros de inicialización del servlet como una enumeración de objetos String.
a.getServletContext()	- Devuelve una referencia al ServletContext a quien ha realizado la llamada
a.getServletName()	- Devuelve el nombre de esta instancia de servlet.

```
{
:
public void init ( ServletConfig config ) throws ServletException {
    Enumeration enu=config.getInitParameterNames();
    super.init(config);
    for ( ; enu.hasMoreElements(); )
    {
        cadena+=" " +config.getInitParameter ( ( String ) enu.nextElement() );
    }
}
```

© JAA 2013

## ServletContext

- El Interface ServletContext permite manejar propiedades y métodos del contexto de los servlets, que viene a ser el contexto de la aplicación web.
- Todo los servlets dentro de una misma aplicación web tienen el mismo ServletContext.
- Mediante este interface podemos controlar los atributos que se cargan a nivel de toda la aplicación web (*variables de aplicación o globales para la aplicación web*)



© JAA 2013

## ServletContext

- Todos los Servlets de la aplicación podrán ver y compartir estos mismos atributos.
- ServletContext ayudan de forma eficiente a mantener información a nivel de toda la aplicación, de tal manera que todas las páginas JSP y Servlets de tu aplicación las puedan ver y modificar.
- Dispone de diferentes métodos y su utilización es de la siguiente forma, pero nunca en método *init()*

*ServletContext a = getServletContext();*

© JAA 2013

## ServletContext

Métodos	Descripción
a.setAttribute ( String, Objeto )	- Permite guardar el objeto en el Contexto para que otros Servlets lo usen.
a.getAttribute ( String )	- Permite obtener el objeto del Contexto previamente guardado.
a.getServerInfo()	- Devuelve el nombre y la versión del Servidor donde se trasmite la petición
a.getServletNames()	- Devuelve una enumeración de los nombres de los objetos Servlets en el Servidor
<a href="http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html">http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html</a>	

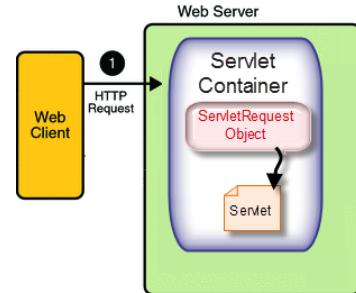
```
{
:
ServletContext a = getServletContext();
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println(
("<html> <head><title>Hola Mundo</title></head>" +
"<body> <h1>Hola Mundo</h1>" + a.getServerInfo() + "<br>" + cadena + "</body></html>");

}
```

© JAA 2013

## Objeto Request.

- Las peticiones realizadas a un Servlet, son encapsuladas mediante la interfaz `javax.servlet.ServletRequest`
- Encapsula información sobre:
  - Parámetros de la petición
  - Atributos
  - Internacionalización
  - El cuerpo de la petición
  - El protocolo de la petición
  - El servidor
  - El cliente
  - Redirección de la petición (Dispatchers)
- El contenedor de servlet crea un objeto `ServletRequest` y lo pasa como argumento al método `service()` al ser invocado.



© JAA 2013

## Objeto Request.

- Dentro de las peticiones y tratamiento de las mismas, deberemos distinguir 2 elementos básicos:
  - Parámetros
    - ✓ Son enviados desde el cliente
    - ✓ Disponen de un formato de pares: clave-valor
    - ✓ Son recibidos como cadena de texto
    - ✓ Disponen de métodos para su tratamiento.
  - Atributos
    - ✓ Se añaden durante la gestión de la petición (dentro del contenedor)
    - ✓ Disponen de un formato de pares: clave-valor
    - ✓ Pueden ser recuperados por cualquier servlet/JSP durante la vida de una petición.
    - ✓ Disponen de métodos para su tratamiento.

© JAA 2013

## Objeto Request. Métodos

Métodos . PARAMETROS	Descripción
getParameter ( " Nombre_parametro ")	- Devuelve una <u>cadena</u> que contiene el valor del parámetro especificado.
getParameterNames()	- Devuelve una <u>enumeración</u> de los nombre de los parámetros pasados en la petición.
getParameterValues()	- Devuelve un <u>array de String</u> , con los valores de los parámetros pasados en la petición
a.getServletNames()	- Devuelve una enumeración de los nombres de los objetos Servlets en el Servidor
<a href="http://docs.oracle.com/javaee/5/api/javax/servlet/ServletRequest.html">http://docs.oracle.com/javaee/5/api/javax/servlet/ServletRequest.html</a>	

© JAA 2013

## Objeto Request. Métodos

Métodos . ATRIBUTOS	Descripción
getAttribute(String name)	- Devuelve el valor (objeto) de un atributo en función de su clave
getAttributeNames()	- Devuelve una <u>enumeración</u> de los nombre de los atributos existentes en la petición.
setAttribute(String name, Object o)	- Añade un atributo a la petición
removeAttribute(String name)	- Elimina un atributo de la petición
<a href="http://docs.oracle.com/javaee/5/api/javax/servlet/ServletRequest.html">http://docs.oracle.com/javaee/5/api/javax/servlet/ServletRequest.html</a>	

© JAA 2013

## Objeto Request. Métodos

Métodos . CABECERAS	Descripción
getHeader( String )	<ul style="list-style-type: none"> <li>- Devuelve el valor de la cabecera indicada, null sino hay esa cabecera</li> </ul> <p>getHeader (" User-Agent " )</p>
getHeaders( String )	<ul style="list-style-type: none"> <li>- Algunas cabeceras pueden devolver mas de un valor.</li> <li>- Devuelve una enumeración con los valores de esta cabecera determinada.</li> </ul>
getHeaderNames()	<ul style="list-style-type: none"> <li>- Devuelve una enumeración con todas las cabeceras presentes.</li> </ul>
getMethod()	<ul style="list-style-type: none"> <li>- Devuelve el método de envío utilizado POP, GET, POST, etc.</li> </ul>
getQueryString()	<ul style="list-style-type: none"> <li>- Devuelve la query String (consulta ) que es usada en la URL</li> </ul>

© JAA 2013

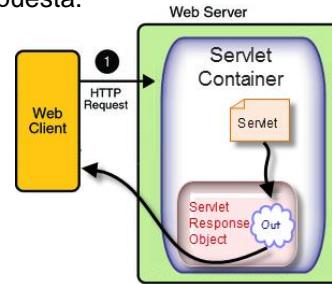
## Objeto Request. Métodos

Métodos . GENERALES	Descripción
getProtocol()	<ul style="list-style-type: none"> <li>- Devuelve el protocolo usado en la petición. &lt;protocolo&gt; / &lt; version ppal &gt; &lt; version sec &gt;</li> </ul>
getRemoteAddr()	<ul style="list-style-type: none"> <li>- Devuelve la dirección IP del Cliente</li> </ul>
getRemoteHost()	<ul style="list-style-type: none"> <li>- Devuelve el nombre completo del Cliente</li> </ul>
getServerName()	<ul style="list-style-type: none"> <li>- Devuelve el nombre del Servidor</li> </ul>
getServerPort()	<ul style="list-style-type: none"> <li>- Devuelve el puerto usado para la petición</li> </ul>
getCharacterEncoding()	<ul style="list-style-type: none"> <li>- Devuelve el juego de caracteres del cuerpo de la petición.</li> </ul>
getContentLength()	<ul style="list-style-type: none"> <li>- Devuelve la longitud total del cuerpo de la petición</li> </ul>
getContentType()	<ul style="list-style-type: none"> <li>- Devuelve el tipo MIME del cuerpo de la petición</li> </ul>

© JAA 2013

## Objeto Response.

- Los servlets generan la respuesta a partir de un objeto que implementa la interfaz `javax.servlet.ServletResponse`.
- Los objetos ServletResponse son instanciados por el contenedor y pasados al servlet.
- ServletResponse encapsula un objeto OutputStream, que es el utilizado para que el Servlet escriba y mande la respuesta.
- La interfaz proporciona métodos para controlar este buffer



© JAA 2013

## Objeto Response.

- El Servlet «monta» la respuesta escribiendo en el objeto OutputStream anterior mediante los métodos que proporciona la interface.
  - Para contenido dinámico: `javax.servlet.ServletOutputStream`
  - Para texto plano: `PrintWriter`

```
{
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Hola Mundo</title>");
  out.println("</head>");
  out.println("<body> <h1>Hola Mundo</h1></body>");
  out.println("</html>");
}
```

© JAA 2013

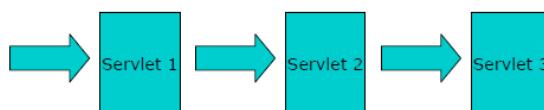
## Objeto Response. Métodos

Métodos . GENERALES	Descripción
res.setContentType( Tipo_respuesta );	- Indica el tipo de respuesta que se va a enviar response.setContentType("text/html");
res.setBufferSize( int )	- Establece el tamaño del buffer (el tamaño por defecto depende de la implementación)
res.getBufferSize()	- Devuelve el tamaño del buffer (o cero si no se está usando buffer)
res.flushBuffer()	- Provoca el volcado de la respuesta (cuerpo y cabeceras).
res.reset()	- Vacía el Stream ( Cabeceras y Cuerpo )
res.isCommitted()	- Devuelve TRUE si el buffer se ha volcado (escrito cabeceras y cuerpo)
res.sendRedirect( "URL" )	- Permite el envío de la petición a una dirección URL determinada.
res.setContentLength( int )	- Indica la cantidad de datos máxima a transmitir
<a href="http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html">http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html</a>	

© JAA 2013

## Redirecciones

- Una petición puede pasar por diferentes servlets y/o JSP's.
- Cada Servlet/JSP es responsable de la gestión de una parte de la petición.
- La respuesta de un Servlet se puede convertir en la Request de otro.

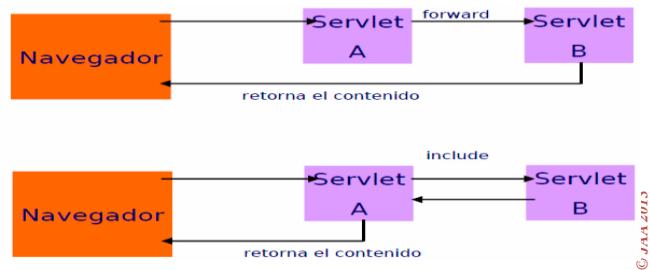


- **Servlet Chaining:**
  - El Servidor Guía la respuesta a través de los diferentes servlets que van enriqueciéndola y al final la redirige al cliente.

© JAA 2013

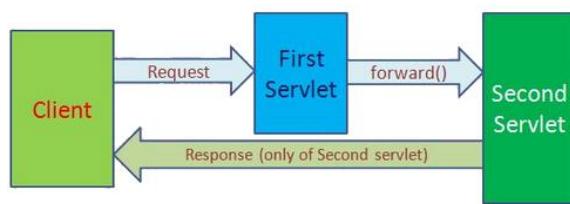
## Redirecciones

- ¿Para qué valen las redirecciones?
  - Evitar repeticiones de tareas
    - ✓ Añadir fechas, preprocesado de etiquetas, añadir elementos habituales, etc
  - Realizar tareas de autenticación
    - ✓ Control de si un usuario está o no autenticado
  - Realizar un diseño modular de la capa de presentación
    - ✓ Cumplimiento de la norma: Un Servlet – una acción.
    - ✓ Reutilización



## Redirecciones

- Las operaciones pueden llevarse a cabo de 2 formas:
  - ✓ Forward
  - ✓ Redirect
- Es importante entender la diferencia entre estos dos casos, en particular con respecto a la recarga del navegador de páginas web.
- Forward
  - ✓ Esta operación es interna del Servlet, por lo que el Navegador no conoce la operación, por lo que su URL original permanece intacta.



## Redirecciones

- Sintaxis. Forward

- ✓ En caso de que sea una pagina JSP o HTML.

```
RequestDispatcher redireccion = null;
redireccion = request.getRequestDispatcher("principal.html");
redireccion.forward (request, response);
```

- ✓ Pero si es otro servlet

```
<servlet-mapping>
<servlet-name>cont_empresas</servlet-name>
```

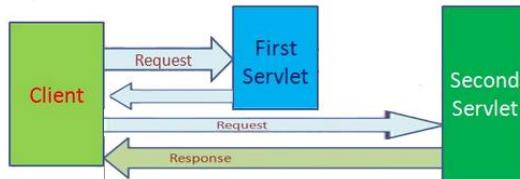
```
RequestDispatcher redireccion = null;
redireccion = request.getRequestDispatcher ("/cont_empresas");
redireccion.forward (request, response);
```

© JAA 2013

## Redirecciones

- Redirect

- ✓ Una redirección es un proceso de dos etapas, donde la aplicación web indica al navegador que buscar una segunda URL, que difiere de la original.
- ✓ Una recarga navegador de la segunda URL no repetirá la solicitud original, sino que más bien buscar la segunda URL.



- Redirect es mas lento que Forward

- ✓ Se requieren dos solicitudes del navegador.
- ✓ Los objetos colocados en el alcance solicitud original no están disponibles para la segunda petición.

© JAA 2013

## Redirecciones

- Sintaxis. Redirect

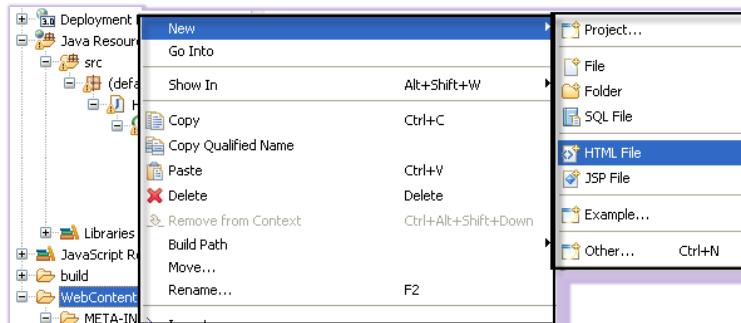
```
public class DemoSiteSelector extends HttpServlet{
    Vector sites = new Vector();
    Random random = new Random();

    public void init(ServletConfig config) throws ServletException{
        super.init(config);
        sites.addElement("http://www.google.com");
        sites.addElement("http://www.yahoo.com");
    }
    public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException{
        res.setContentType("text/plain");
        PrintWriter out= res.getWriter();
        int siteIndex = Math.abs(random.nextInt())%sites.size();
        String site = (String)sites.elementAt(siteIndex);
        res.sendRedirect(site);
    }
}
```

© JAA 2013

## Varios

- Para poder asignar un HTML al proyecto, lo hacemos en WebContent del proyecto



- Para posteriormente llamarlo desde explorador:  
<http://localhost:8081/ServletNombre/fichero.html>

© JAA 2013

## Varios

- Código de formularios

```
<form action="Nombre_Servlet" method="POST" name="form">
<pre>
    Nombre:&nbsp;&nbsp;&nbsp;&nbsp; <input NAME="nombre" size="20">
    Contraseña: <input TYPE="password" NAME="contrasena" size="20">
    <input TYPE="SUBMIT" VALUE="Enviar">
</pre>
</form>
```



© JAA 2013

Servlets.

## COOKIES Y SESIONES

© JAA 2013

## Mantenimiento de estado

### Servidor

- Opciones:
  - Memoria (Session)
  - Disco (BBDD)
- Problemas:
  - Escalabilidad
  - Temporalidad

### Cliente

- Opciones:
  - Cookies
  - <input type="hidden" ...>
- Problemas:
  - Seguridad
  - Sobrecoste
- Nueva opción HTML5
  - Almacenamiento Local
  - Reduce el sobrecoste

© JAA 2013

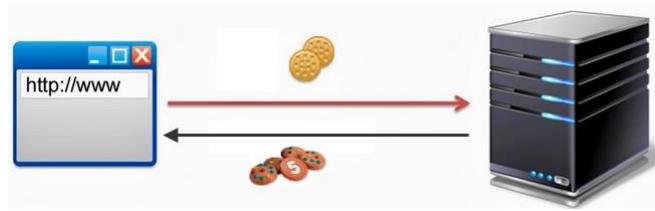
## Introducción

- HTTP es un protocolo stateless (*sin estado*), por lo que cada vez que un cliente pide una página Web, se abre una conexión independiente con el servidor Web, el cual no mantiene *información contextual* del cliente.
- Este tipo de funcionamiento provoca una serie de dificultades que hay que subsanar.
  - ✓ ¿Productos de una cesta de la compra?
  - ✓ ¿Qué cesta de la compra tiene que pagar?
- Varias soluciones típicas
  - Cookies
  - Seguimiento de sesiones (Objetos HttpSession)
  - Reescritura de URLs ( URL Encoding - Codificación de URL's )
  - Campos Ocultos

© JAA 2013

## Cookies. Introducción

- Las cookies es una forma de mantener una "sesion" entre el cliente y el Servidor.
- Permite a un servidor HTTP reconocer clientes previamente identificados.
- Están implementadas como una clase dentro del paquete `javax.servlet.http`.
- Las cookies también se definen como "*un tipo especial de cabeceras*" que se establecen en el servidor y el navegador



© JAA 2013

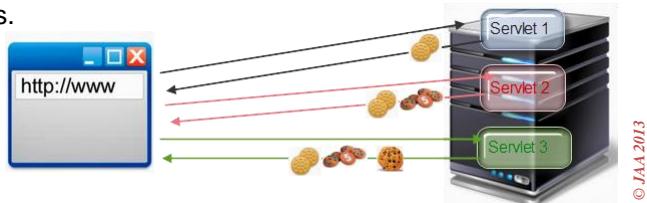
## Cookies. Introducción

- Los servleter envían cookies al cliente añadiendo campos a las cabeceras de respuesta HTTP.
- Los clientes devuelven las cookies automáticamente añadiendo campos a las cabeceras de peticiones HTTP.
- Las cookies están compuestas de 2 elementos : Nombre + Valor
- Además de un nombre y un valor, también se pueden proporcionar atributos opcionales como comentarios (Los navegadores actuales no siempre tratan correctamente a los atributos opcionales).
- Cada cookie tiene un identificador (nombre) y se almacena en un directorio del Disco Duro del cliente como fichero.

© JAA 2013

## Cookies. Introducción

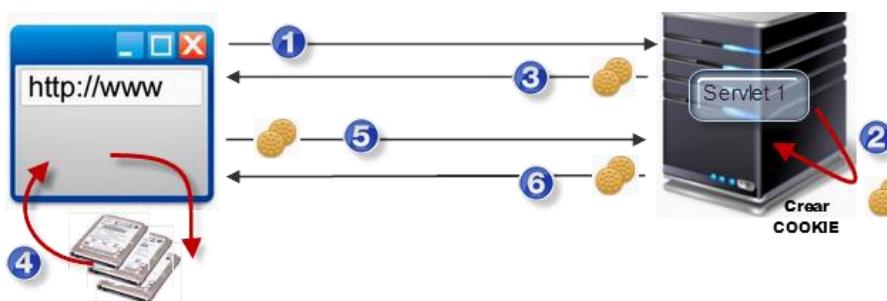
- Un servidor puede proporcionar una o más cookies a un cliente.
- El software del cliente, como un navegador, habitualmente puede soportar hasta 20 cookies / host.
- Las cookies que se almacenan en un cliente de un determinado servidor, sólo pueden ser devueltas a ese mismo servidor.
- Un servidor puede contener múltiples servlets, como las cookies son devueltas al servidor, los servlets que se ejecutan dentro de un servidor comparten las cookies.



© JAA 2013

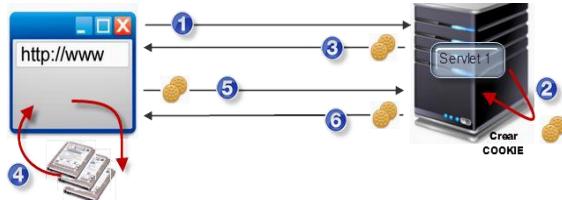
## Cookies. Funcionamiento

- El funcionamiento de las Cookies es muy sencillo.
- Son creadas por el Servlets en la primera llamada recibida desde el cliente y son enviadas a través de las cabecera (HEADER) del mensaje.



© JAA 2013

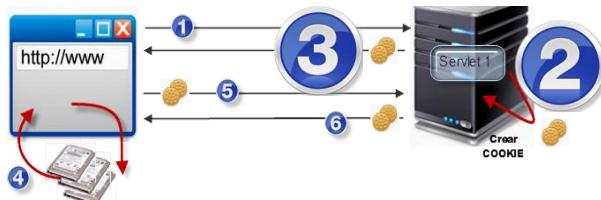
## Cookies. Funcionamiento



1. Petición del Cliente (Navegador) de página (Servlet)
  - Si el Cliente dispone de Cookies anteriores de ese Servidor, y no están caducadas, se las envía en esta primera petición, sino, no envía nada.
2. El Servidor recibe petición y crea cookie/s correspondiente.
3. El Servlet crea la respuesta, añadiendo en ella, la cookie/s creada/s
4. El Cliente recibe la respuesta y la cookie/s y las almacena en Memoria/Disco
5. El cliente realiza una nueva petición al Sevidor, añadiendo las Cookies de dicho servidor

© JAA 2013

## Cookies. Funcionamiento

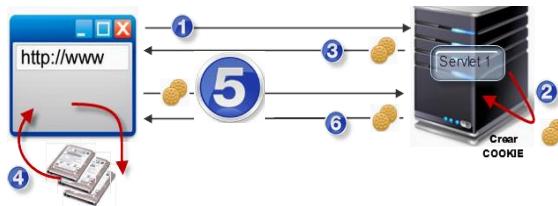


- 2.- El Servidor recibe petición y crea cookie/s correspondiente.
  - La creación de las cookies se realiza mediante la creación de un objeto `Cookie`  
`Cookie MiGalleta = new Cookie(Nombre, Valor);`
  - Podemos modificar también los valores de las cookies mediante métodos como:  
`MiGalleta.setValue(Valor);`

- 3.- El Servlet crea la respuesta, añadiendo en ella, la cookie/s creada/s
  - El Servlet añade la cookie a las cabeceras del mensaje mediante:  
`res.addCookie(MiGalleta);`

© JAA 2013

## Cookies. Funcionamiento



- 5.- El Servidor recibe petición y también recibe las cookie/s correspondiente.
- Las cookies son enviadas al Servidor y disponemos del método `request.getCookies()` para recuperarlas.

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

if ( Cookies_enviadas != null )
{
    Cookie Varcookie ;
    for ( int i = 0; Cookies_enviadas.length(); i ++ ) {
        Varcookie = Cookies[ i ];
        System.out.println ( "Nombre: "+ Varcookie.getName() +
                            "Valor: "+ Varcookie.getValue() );
    }
}
```

© JAA 2013

## Cookies. Gestión y Mantenimiento

- La creación y gestión de Cookies se realiza mediante la clase **javax.servlet.http.Cookie**.
- El constructor de esta clase crea una cookie con un nombre inicial y un valor. *Se puede cambiar el valor posteriormente utilizando el método `setValue`.*
  - Nombre
    - El nombre del cookie debe ser Strings alfanuméricos cualificados como tokens en la especificación RFC 2068.
    - Además, los nombres que empiezan con el carácter dollar ("\$") están reservados.
  - Valor
    - El valor del cookie puede ser cualquier *string*, aunque no está garantizado que los valores *null* funcionen en todos los navegadores.

```
Cookie MiGalleta;
:
MiGalleta = new Cookie("LIBRO", "El Quijote");
:
```

© JAA 2013

## Cookies. Gestión y Mantenimiento

- Las cookies creadas, pueden ser complementadas con atributos de las cookies como:
  - ✓ Comentarios, Duración, Nombre, Valor, etc

Métodos	Descripción
o.getComment() o.setComment ( String )	- Devuelve / Establece el comentario asignado a la Cookie
o.getMaxAge() o.setMaxAge( int )    0    inmediato -1    al cerrar el Navegador	- Devuelve / Establece el número de segundos que la cookie permanece guardada en el disco del cliente.
o.getName() o.setName ( String )	- Devuelve / Establece el nombre la cookie
o.getValue() o.setValue ( String )	- Devuelve / Establece el valor de la cookie
o.getSecure() o.setSecure ( Boolean )	- Devuelve / Establece el parámetro de Seguridad. - Si flag_seguridad es TRUE, la cookie sólo será enviada si la conexión es segura (SSL).

© JAA 2013

## Cookies. Gestión y Mantenimiento

• Ejemplo:

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

```
Cookie auxiliar;
Cookie[] galletas=request.getCookies();
String Informe="";
if (galletas!=null) {
  for (int i=0; i<galletas.length;i++) {
    auxiliar=galletas[i];
    if (auxiliar.getName().equals("JOSE")) {
      MiGalleta=auxiliar;
      existe=true;
      break;
    }
  }
  System.out.println(existe);
  Informe="Bienvenido de nuevo";
}
else
{
  System.out.println("No hay ninguna Cookie");
  Informe="Es la primera vez que te conectas";
  MiGalleta=new Cookie("JOSE","2");
  existe=true;
  response.addCookie(MiGalleta);
}
```

```
response.setContentType("text/html");

String Dir=request.getRemoteAddr();
String RHost=request.getLocalName();

PrintWriter out = response.getWriter();

out.println("<html> head><title>Segundo</title></head>");
out.println("<body><br>");
if (existe) {
  out.println("<h1>" + Informe + "</h1><br>");
}
else {
  out.println("<h1>" + Informe + "</h1><br>");
}
out.println("</body></html>");
```

© JAA 2013

## Seguimiento de Sesiones. Introducción

- El seguimiento de sesión es un mecanismo que los servlets utilizan para mantener el estado sobre los datos de un mismo usuario.
- Se considera un mismo usuario a las peticiones originadas desde el mismo navegador durante un periodo de tiempo.
- Las sesiones son compartidas por todos los servlets de una misma aplicación Web.
- Para utilizar el seguimiento de sesión debemos:
  - Obtener una sesión (un objeto *HttpSession*) para un usuario.
  - Almacenar u obtener datos desde el objeto *HttpSession*.
  - Iniciar la sesión (opcional).

© JAA 2013

## Sesiones. Obtención de Sesiones

- Las sesiones se implementan a través de objetos de la clase *HttpSession*, creados por el contenedor cuando se inicia una sesión para un nuevo usuario.
- Estos objetos:
  - ✓ Se basan en mecanismos de cookies / reescritura de URL's.
  - ✓ Permiten mantener el estado del cliente de forma transparente.
  - ✓ La información de una sesión se mantiene entre las diferentes peticiones del cliente.
- Para extraer la referencia a este objeto desde un servlet utilizamos el método *.getSession()* :
 

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

  - El sistema extrae un ID de sesión, comparándolo con una tabla de objetos *HttpSession* creados anteriormente.
  - Si el resultado es NULL → La sesión es creada.

© JAA 2013

## Sesiones. Obtención de Sesiones

- Las objetos **session**, viven en el servidor y son asociados a los clientes mediante un mecanismo similar a las cookies / reescritura de URL.
- Estos objetos de sesión permite almacenar cualquier dato en formato clave-valor.
- Una vez recuperada el objeto **session** podremos recuperar:
  - ID de Sesión
  - Datos de una aplicación (Objetos) Clave-Valor
- Para mantener la sesión apropiadamente, debemos llamar a `getSession` antes de escribir cualquier respuesta.
  - ✓ Si respondemos utilizando un `PrintWriter`, entonces debemos llamar a `getSession` antes de acceder al `PrintWriter`, no sólo antes de enviar cualquier respuesta.

© JAA 2013

## Sesiones. Obtención de Sesiones

### Ejemplo

```
public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // Obtiene la variable sesión para el usuario.
    HttpSession session = request.getSession(true);
    .....
    out = response.getWriter();
    .....
}
```

© JAA 2013

## Sesiones. Obtención de Sesiones

- Las objetos **session**, viven en el servidor y son asociados a los clientes mediante un mecanismo similar a las cookies / reescritura de URL.

Métodos	Descripción
Object s.getValue( String )	- Devuelve el valor del parámetro almacenado en la sesión. <i>Tipo_Obj nombre = (Tipo_Obj) session.getValue("nom");</i>
s.putValue( String, valor )	- Establece el valor del parámetro.
Object s.getAttribute ( String ) s.setAttribute ( String, valor )	- Devuelve / Establece el valor del atributo almacenado en la sesión.
Enumeration s.getAttributeNames()	- Devuelve todos los nombres de los atributos de la sesión.
String[] s.getValueNames()	- Devuelve todos los nombres de los elementos de la sesión.
s.getId()	- Devuelve el identificador único asociado a la sesión.

© JAA2013

## Sesiones. Obtención de Sesiones

Métodos	Descripción
s.getCreationTime ( )	- Devuelve el valor del momento de la creación de la sesión (expresado en milisegundos transcurridos desde el 1 de enero de 1970).
s.getLastAccessedTime ( )	- Devuelve el valor del último acceso a una página de la sesión (milisegundos transcurridos desde el 1 de enero de 1970).
s.getMaxInactiveInterval ( ) s.setMaxInactiveInterval ( int segundos )	- Devuelve / Establece los segundos que deben transcurrir desde el último acceso para que la sesión sea cerrada.
Enumeration s.getAttributeNames()	- Devuelve todos los nombres de los atributos de la sesión.
s.invalidate()	- Este método invalida la sesión y desenlaza todos los objetos asociados.

© JAA2013

## Sesiones. Obtención de Sesiones

### Ejemplo

```
public class CatalogServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        String mensaje;
        HttpSession sesion = request.getSession(true);
        // Obtenemos la variable sesion, si es la primera vez, objeto vacio → sin atributos.
        // si NO es la primera vez objeto con atributo "fecha"
        if (sesion.getAttribute("fecha")==null) {
            Date f1=new Date();
            sesion.setAttribute("fecha", f1);
            mensaje="Bienvenido, es la primera vez. Son las: "+f1.toString();
        } else {
            Date ff=(Date)sesion.getAttribute("fecha");
            mensaje="Hola Otra vez<br>La primera vez te conectaste a las: "+ff.toString();
            sesion.invalidate(); // Eliminamos el objeto Sesión del Servidor
        }
    }
}
```

© JAA 2013

## Finalización de Sesiones

- Sesiones se convierten en inactivas cuando
  - ✓ La cantidad de tiempo entre los accesos del cliente supera el intervalo especificado por `getMaxInactiveInterval`.  
Si el intervalo es 0 o negativo, la sesión no será cerrada.
  - ✓ Cuando manualmente se provoca. (`session.invalidate()`)
- ¿Qué significa invalidar una sesión?
  - Eliminar el objeto HttpSession y desligar (eliminar) sus valores

© JAA 2013

## Sesiones y Navegadores

- Por defecto, el seguimiento de sesión utiliza cookies para asociar un identificador de sesión con un usuario.
- Para soportar también a los usuarios que acceden al servlet con un navegador que no soporta cookies, se utiliza la reescritura de URL (URL Encoding (rewriting)).
- Cuando se utiliza la reescritura de URL, se llama a los métodos añadiendo un identificador de sesión a todos los enlaces que se devuelven al cliente
- No depende de la configuración del navegador, pero es tedioso y propenso a errores (hay que codificar todas las URLs que se devuelven al cliente)

© JAA 2013

## Sesiones y Navegadores

- El contenedor web proporciona el servicio de generación de identificadores y codificación de URL mediante:  
*HttpServletResponse.encodeUrl (String url)*
- Si se redirecciona al usuario a otra página, el método para asociar el ID de sesión con la URL redireccionada se llama  
*HttpServletResponse.encodeRedirectUrl*
- Los métodos *encodeUrl* y *encodeRedirectUrl* deciden si las URL necesitan ser reescritas, y devolver la URL cambiada o sin cambiar.

© JAA 2013

# DESPLIEGUE

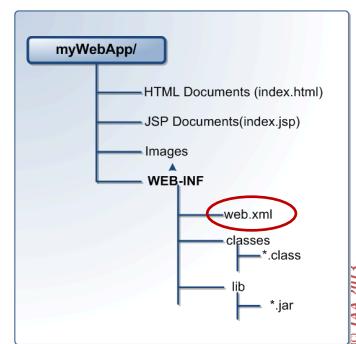
© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

### Descriptor de despliegue

- Es un componente de aplicaciones J2EE que describe cómo se debe desplegar (o implantar) una aplicación web.
- Permite a un Servidor de Aplicaciones dirigir la publicación de un módulo o aplicación.
- Debe ser llamado *web.xml*, y debe ser colocado en un subdirectorio llamado *WEB-INF*, directamente debajo de la raíz de la aplicación web.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/app_3_0.xsd">
    <servlet>
        <servlet-name>HolaMundoServlet</servlet-name>
        <servlet-class>org.prueba.servlet.HolaMundoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HolaMundoServlet</servlet-name>
        <url-pattern>/HolaMundoServlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```



© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

- El contenedor es el encargado de proporcionar los servicios descritos en el Descriptor de Despliegue:
  - ✓ Seguridad, Mapeos, Archivos de inicio, Configuración de la sesión, Filtros, etc...
- Los DD tienen las siguientes características:
  - Modelo declarativo, no programático (XML)
  - Generados por el programador ( o por el IDE empleado )
  - Aplicación es más portátil y flexible
  - Mediante su cambio podemos conseguir que una aplicación se comporte de maneras diferentes en contenedores distintos sin tener que recompilar

© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

- La primera entrada XML es el DTD con las especificaciones aplicadas (2.2, 2.3, 3.0, etc)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
```

- Soporte a herramientas de desarrollo:

Etiquetasopcionales que no afectan al comportamiento de la aplicación

<icon>	Icono que mostrará la herramienta
<display-name>	Nombre que mostrará la herramienta
<description>	Descripción de la aplicación/servlet que mostrará la herramienta

© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

- **Filtros:**

El orden del filtrado es tal cual aparece en el DD.

<filter>	Define un filtro.
<filter-name> y <filter-class>	Indican el nombre y la clase Java que lo implementa.
<filter-mapping>	Define cómo invocar un filtro.
<filter-name>	Debe coincidir con el anterior.
<url-pattern> ó <servlet-name>	Recogen el destino de la invocación.

```
<filter>
    <filter-name>Basic Filter</filter-name>
    <filter-class>BasicFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>Basic Filter</filter-name>
    <servlet-name>Filter Demo</servlet-name>
</filter-mapping>
```

© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

- **Servlets:**

Declaración de servlets

<servlet> que debe contener:	
<servlet-class> o <jsp-file>	Nombre completamente cualificado de la clase que implementa el servlet/JSP
<servlet-name>	Alias para referirse al servlet a lo largo del DD

Otras etiquetas opcionales bajo <servlet>:

<init-param>	Parámetros de inicio; debe contener <param-name> y <param-value>
<load-on-startup>	Fuerza la carga del servlet
<security-role-ref>	Mapeos de roles usados en el servlet con roles de servidor
<servlet-mapping>	Asocia un patrón URL con a un server.
<servlet-name>	Alias del servlet definido con <servlet>
<url-pattern>	es el patrón URL asociado.

© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

- Servlets:

```
< servlet >
    < servlet-name > HolaMundoServlet < /servlet-name >
    < servlet-class > org.prueba.servlet.HolaMundoServlet < /servlet-class >
< /servlet >

< servlet-mapping >
    < servlet-name > HolaMundoServlet < /servlet-name >
    < url-pattern > /HolaMundoServlet < /url-pattern >
< /servlet-mapping >

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

© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

### web.xml - Ejemplo

```
< servlet >
    < servlet-name > ServletBD < /servlet-name >
    < servlet-class > aplicacion.Servlet1 < /servlet-class >
    < init-param >
        < param-name > driver < /param-name >
        < param-value > oracle.jdbc.driver.OracleDriver
        < /param-value >
    < /init-param >
    < load-on-startup > 1 < /load-on-startup >
    < security-role-ref >
        < role-name > admin < /role-name >
        < role-link > root < /role-link >
    < /security-role-ref >
< /servlet >
...
< servlet-mapping >
    < servlet-name > ServletBD < /servlet-name >
    < url-pattern > /bbdd/* < /url-pattern >
< /servlet-mapping >
<!-- Todas las peticiones que comiencen por
/bbdd/ se enviarán a ServletBD -->
```

© JAA 2013

## Ficheros WAR y Descriptores de Ficheros

- El contexto puede tener parámetros de inicio que afectan a la aplicación:  
`<context-param>` Declara un parámetro y debe contener `<param-name>` y `<param-value>`.
- Mapeo de extensiones de archivos con tipos MIME:  
`<mime-mapping>` Debe contener `<extension>` y `<mime-type>`
- Vida máxima de las sesiones:  
`- <session-config>` indicando el número de minutos
- Página de error/excepción:  
`<error-page>` Donde hay que indicar una URL con la página de error, un código de error (404, p.ej.) o una excepción Java (IOException).

© JAA 2013

## Anotaciones (Servlet 3.0)

- `@WebServlet`
  - value: Los patrones de URL del servlet
  - urlPatterns: Los patrones de URL del servlet
  - name: El nombre del servlet
  - displayName: El nombre para mostrar del servlet
  - description: La descripción del servlet
  - smallIcon: El pequeño ícono del servlet
  - largeIcon: El ícono grande del servlet
  - initParams: Los parámetros de inicio del servlet
  - loadOnStartup: El orden del servlet en la carga al inicio

```
@WebServlet({ "/login", "/logout" })
@WebServlet(name="auth-servlet", value = { "/login", "/logout" })
```
- En `javax.servlet.annotation` hay disponibles anotaciones para declarar un filtro de servlet (`@WebFilter`), un WebListener (`@WebListener`), ...

© JAA 2013

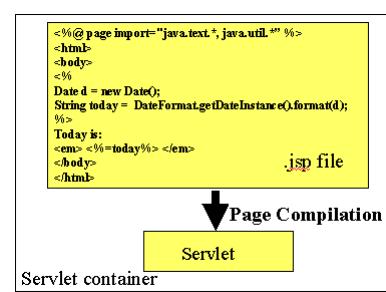
## Java Server Pages

# JSP

© JAA 2013

## Introducción

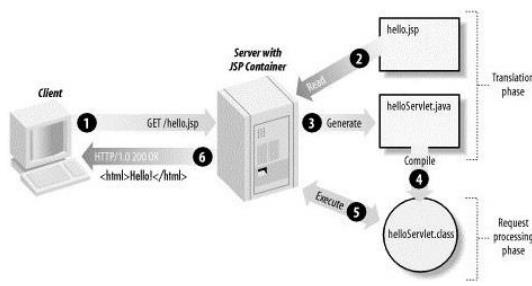
- Los JSP es una alternativa java a la generación de contenidos de forma dinámica en el lado del servidor.
- Un JSP es un componente del lado del servidor Web que se puede utilizar para generar páginas web dinámicas, pero difiere de los Servlets en:
  - Servlets generan código HTML de código Java.
  - JSP incrustar código Java en HTML estático.
- El código Java queda embebido dentro del código HTML de forma similar a PHP o ASP.



© JAA 2013

## Introducción

- Ahora nos permite realizar una «Separación de Roles»
  - Con un servlet, el programador Java debe generar todo el HTML.
  - Con un JSP:
    - ✓ Separación de código HTML
    - ✓ Utilización de JavaBeans
    - ✓ Utilización de bibliotecas de etiquetas JSP
- JSP es en el fondo una tecnología equivalente a los servlets.
  - Las páginas JSP se traducen en servlets que ejecuta el servidor en cada petición.



© JAA 2013

## Introducción

- Frente a CGI.
  - Seguridad : Entorno de ejecución controlado por la JVM
  - Eficiencia: Cada nueva petición es atendida por un hilo, no por un nuevo proceso
- Frente a PHP
  - Lenguaje más potente para la generación dinámica
    - ✓ Lenguaje de script orientado a objetos (Java)
  - Mejores herramientas de desarrollo y soporte
- Frente a ASP
  - Mejores rendimientos: Código compilado, no interpretado (SDK 1.4 o sup)
  - Independiente de la plataforma: Portable a múltiples servidores y SO.
  - Lenguaje más potente para la generación dinámica (Java)

© JAA 2013

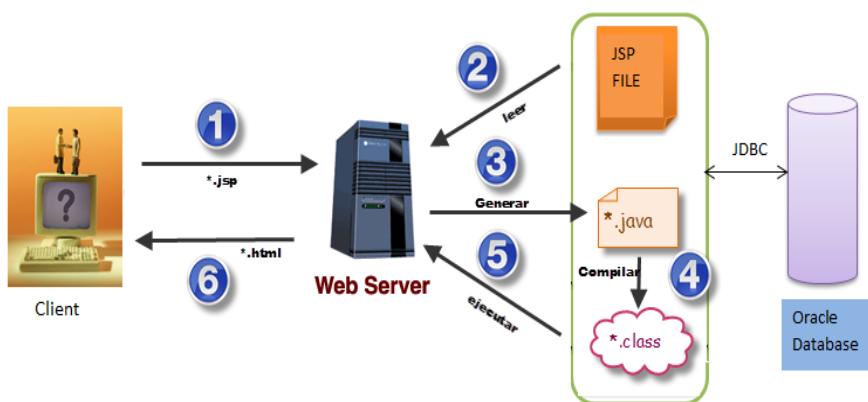
## Introducción

- Frente a Servlets Puro.
  - La lógica de negocio y la presentación están más separados.
  - Simplifican el desarrollo de aplicaciones Web
    - ✓ Más conveniente para crear HTML (no es necesario println).
  - Soporte para reutilizar software a través de JavaBeans y etiquetas adaptadas.
  - Puede utilizarse herramientas estándar (p.e. Homesite)
  - Recompila automáticamente las modificaciones en las páginas jsp
  - No es necesario ubicar las páginas en un directorio especial
    - ✓ /srv/www/tomcat/base/webapps/ROOT/pagina.jsp
  - La URL tampoco es especial.
    - ✓ http://www.uv.es/pagina.jsp

© JAA 2013

## Arquitectura

- Todos los elementos de JSP son procesados por el Servidor, con la siguiente arquitectura de ejecución:



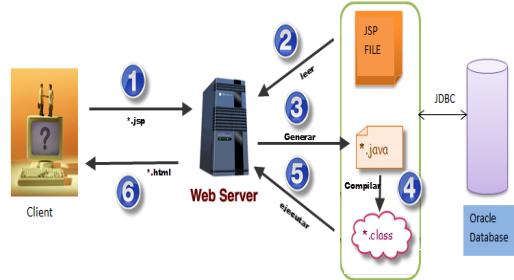
- Cuando el Servidor ejecuta un JSP por primera vez se crea un Servlet asociado que mantendrá hasta que se cierre o se modifique el JSP.

© JAA 2013

# Arquitectura

## Pasos

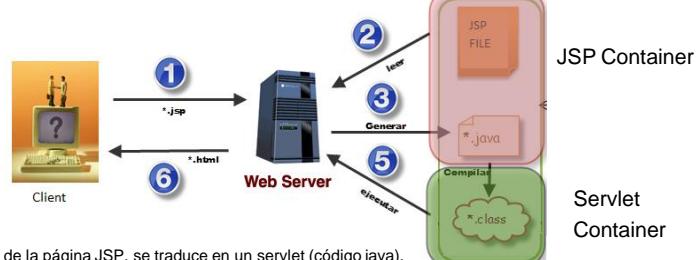
1. El cliente envía la petición de ejecución del fichero JSP.
2. El Servidor localiza el fichero JSP y lo lee.
3. El Servidor genera el correspondiente fichero java (\*.java) basado en Tecnología Servlets.
4. El fichero \*.java es compilado.
5. El Servidor lee y ejecutar el \*.class
6. El Servidor devuelve la ejecución en \*.html



© JAA 2013

# Arquitectura

WEB Container = JSP Container + Servlets Container



### Fase de traducción

- Tras la primera petición de la página JSP, se traduce en un servlet (código java).

### Fase de compilación

- Dicho servlet es compilado para poder servir la petición del cliente.
- Normalmente las fases de traducción y compilación ocurren juntas, y son realizadas por el contenedor automáticamente en la primera petición.

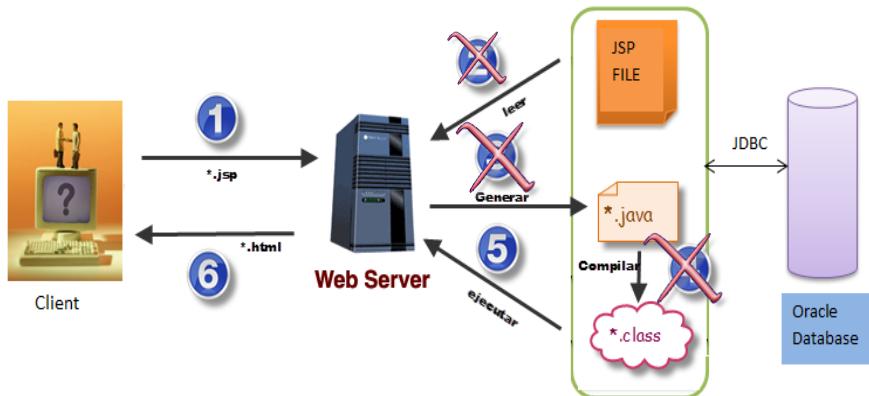
### Fase de ejecución

- Se crea una instancia de dicho servlet, que residirá en memoria de forma permanente mientras el servidor siga en funcionamiento.
- Para las peticiones posteriores se emplea la misma instancia del servlet (no se vuelve a compilar la página).

© JAA 2013

## Arquitectura

- Cuando se recibe una petición a una página JSP que ya ha sido tratada en el servidor (sin cambios), se ejecuta sólo el servlet generado.



© JAA 2013

## Ciclo de ejecución

- Durante la fase de ejecución, el contenedor invoca los siguientes métodos del servlet generado:

**jspInit()**

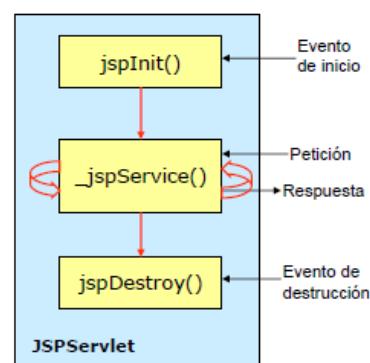
- Permite ejecutar cierto código cuando se produce la primera petición.

**jspService()**

- Se ejecuta en TODAS las peticiones.
- El programador JSP no debe administrarlo directamente.
- La mayoría del código java yHTML se incluye dentro.

**jspDestroy()**

- Permite ejecutar código antes de que finalice el servlet.



NOTA: Todo esto es transparente al programador

© JAA 2013

## Ejemplos

- Ejemplo

```
<HTML>
<HEAD>
    <TITLE>Saludo con Fecha</TITLE>
</HEAD>

<BODY>
    <H1>Hola Mundo</H1>
    <%= new java.util.Date().toString() %>
</BODY>
</HTML>
```

© JAA 2013

Texto Plano

JSP

## Ejemplos

- Ejemplo.jsp:

```
<html>
<body>
<h3>num. aleatorio: <%= Math.random()%> </h3>
</body>
</html>
```

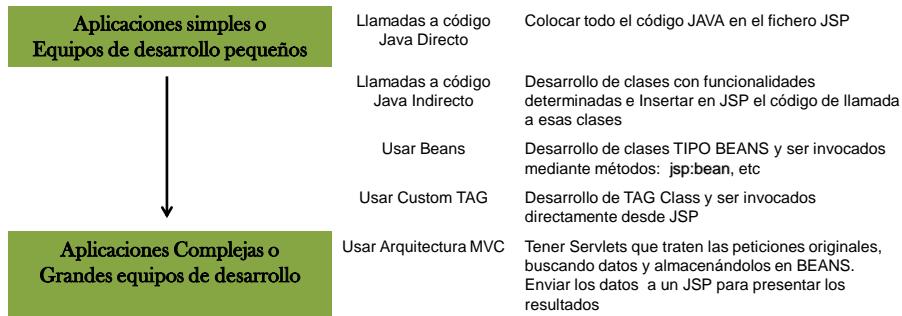
- Servlet Generado:

```
public class Ejemplo_jsp extends HttpJspBase {
    public void _jspService (HttpServletRequest request,
                           HttpServletResponse response) throws ServletException, IOException {
        request.setContentType("text/html");
        JspWriter out = response.getWriter();
        out.print("<html><body> <h3>num. aleatorio:<");
        out.println(Math.random());
        out.println("</h3></body></html>");
    }
}
```

© JAA 2013

## Elementos de JSP

- Hay diferentes maneras de generar contenido dinámico desde JSP.
- Cada uno de estos enfoques tiene un lugar legítimo, el tamaño y la complejidad del proyecto es el factor más importante para decidir qué método es el adecuado.



© JAA 2013

## Elementos de JSP

- Una página JSP está hecho de «*una plantilla de página*», que consiste en:
  - Código HTML,
  - Comentarios JSP y
  - Elementos JSP (Directivas, Secuencias de Comandos (Scriptlets) y Acciones)
- Estos elementos JSP nos permiten insertar código en el servlet que se generará desde la página JSP.
  - Comentarios
  - Directivas
  - Scripting
  - Acciones
- Todos los elementos de JSP comienzan con `<%` y finalizan con `%>`

© JAA 2013

## Elementos de JSP. Comentarios

- Para poder introducir un comentario en una página JSP utilizaremos la siguiente sintaxis:

```
<%-- comentario --%>
```

© JAA 2013

## Elementos de JSP. Directivas

- Las directivas le permiten controlar la estructura general del contenedor JSP (al Servidor JEE) cuyo propósito es proporcionar información para la traducción.
- No depende de la petición ejecutada y no devuelve ningún tipo de respuesta.

```
<%@ directiva Atributo="Valor" %>
<%@ directiva attribute1="value1"
...
attributeN="valueN" %>
```

- Las directivas mas comunes son:
  - INCLUDE                   <%@ include file="....." %>
  - PAGE                        <%@ page Atributo="Valor" %>
  - TAGLIB                     <%@ page Atributo="Valor" %>

© JAA 2013

## Elementos de JSP. Directivas

### INCLUDE

- Se utiliza para introducir ficheros estáticos dentro de una página HTML
- El fichero puede tener cualquier extensión ( \*.html, \*.txt, \*.jsp )
- Si introducimos ficheros HTML, no podemos tener <html> </html>
- Una directiva **include** se debe colocar en el documento en el punto en el que desea que el archivo que se inserte.

<%@ include file="....." %>

© JAA2013

## Elementos de JSP. Directivas

### PAGE

- Permite definir los atributos globales para toda la página (*importación de clases, personalizar la superclase del servlet, establecer el tipo de contenido, etc.*)
- Una directiva de la página se puede colocar en cualquier lugar dentro del documento.

<%@ page Atributo="Valor" %>

Atributos	Descripción
buffer = Tamaño	- Especifica el tamaño del buffer que será utilizado por el objeto OUT
contentType = "text/html"	- Especifica el tipo de archivo que se devuelve
errorPage = "URL"	- Página de envío para cuando el JSP de una excepción.
import = "paquete   clase"	- Importación de un paquete o clase determinada <%@ page import="java.util.Date" %>
Lenguaje = "java"	- Define el lenguaje de programación de los SCRIPTING. Por defecto es JAVA

© JAA2013

## Elementos de JSP. Directivas

### TAGLIB

- Las etiquetas JSP utilizan la sintaxis XML, por lo que se integran limpia y uniformemente a las etiquetas HTML.
- La directiva **taglib**, introducida en JSP 1.1, indica al motor jsp que la página va a utilizar "tag libraries" o librerías de etiquetas .
- Estas librerías contienen etiquetas creadas por el propio programador con sus correspondientes atributos que encapsulan determinada funcionalidad.
- Estas librerías suelen estar añadidas a los proyecto dentro de META-INF

**<%@ taglib uri=" . . ." prefix = " . . ." %>**

URI               permite localizar el fichero descriptor de la librería de extensión tld.

PREFIX           Especifica el identificador que todas las etiquetas de la librería deben incorporar

```
<%@ taglib uri="/META-INF/taglib.tld" prefix="str" %>
```



© JAA 2013

## Elementos de JSP. Directivas

### EJEMPLO

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
    <TITLE>The import Attribute</TITLE>
    <LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>

<BODY>
    <H2>The import Attribute</H2>

    <%-- JSP page directive --%>
    <%@ page contentType="text/html" %>
    <%@ page import="java.util.* , more servlets.*" %>
    <%-- JSP Declaration --%>
    :
    :
    :
```

© JAA 2013

## Elementos de JSP. Scripting

- Son elementos permiten especificar código Java que se convertirá en parte del servlet resultante.
- Están construidos en el lenguaje definido en el atributo  
`<%@ page language..... %>` → Generalmente JAVA
- Tenemos 3 clases de Scripting
  - DECLARACIONES      `<%! ..... %>`  
 ✓ Son declaraciones de variables, métodos, etc que se pueden utilizar en la página JSP.
  - EXPRESIONES      `<%= ..... %>`  
 ✓ Son expresiones UNICAS que se evalúan y se insertan en la pagina resultante.
  - SCRIPTLETS      `<% ..... %>`  
 ✓ Son pequeños códigos JAVA que se ejecutan como un programa normal

© JAA 2013

## Elementos de JSP. Scripting (Declaraciones)

- Una declaración JSP nos permite definir métodos o variables que quedan insertados en el cuerpo principal de la clase servlet, pero fuera del método `_jspService`.
- Estas declaraciones no generan ninguna salida y se utilizan normalmente en combinación con expresiones o scriptlets JSP.
- Los elementos son visibles en toda la página.

### Sintaxis

`<%! Código JAVA %>`

`<%! int contador=0; %>`

```
<H1>Some Heading</H1>
<%!
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>"); }
%>
<%= randomHeading() %>
```

© JAA 2013

## Elementos de JSP. Scripting (Declaraciones)

- Ejemplo:

- El siguiente fragmento JSP imprime el número de veces que la página actual se ha solicitado desde el servidor se inicia
- Un contador de visitas en una sola línea de código!

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

- Múltiples peticiones de los clientes al mismo servlet, utilizan varios subprocessos que llaman a la misma instancia del servlet. (excepto SingleThreadModel)
- En los servlets normales, estas variables son compartidas por múltiples peticiones y no tiene que ser declarado como *static*.

© JAA 2013

## Elementos de JSP. Scripting (Declaraciones)

### XML y sintaxis Especial de Declaración

- A partir de JSP 1.2, los servidores apoyan otra sintaxis para definir declaraciones
- Esta sintaxis está basada en XML y es conveniente no mezclar ambas sintaxis.

```
<%! private int accessCount = 0; %>

equivalencia

<jsp:declaration> private int accessCount = 0; </jsp:declaration>
```

© JAA 2013

## Elementos de JSP. Scripting (Expresiones)

- Una expresión JSP se usa para insertar valores directamente en la salida.
- Se evalúa la expresión, la convierte en una cadena, y se inserta en la página.
- La evaluación de esta expresión, se realiza en tiempo de ejecución (cuando se solicita la página) y así tiene total acceso a la información sobre la solicitud.
- Su sintaxis es:

<%= JAVA EXPRESSION %>

Current time: <%= new java.util.Date() %>

© JAA 2013

## Elementos de JSP. Scripting (Expresiones)

- Para simplificar estas expresiones, se puede utilizar un número de variables predefinidas (o "objetos implícitos").
- Estos objetos implícitos son visibles en toda la página y se utilizan en expresiones para mostrar su contenido.
  - Request: HttpServletRequest
  - Response: HttpServletResponse
  - Session: HttpSession
  - Out: Writer

Your hostname: <%= request.getRemoteHost() %>

<H1>A Random Number</H1>  
<%= Math.random() %>

© JAA 2013

## Elementos de JSP. Scripting (Expresiones)

### XML y sintaxis Especial de Expresiones

- Al igual que la Declaraciones, a partir de JSP 1.2, los servidores apoyan otra sintaxis para definir expresiones
- Esta sintaxis está basada en XML y es conveniente no mezclar ambas sintaxis.

```
<jsp:expression>Java Expression</jsp:expression>
```

```
<%= Math.random() %>
```

equivalencia

```
<jsp:expression> Math.random() </jsp:expression>
```

© JAA 2013

## Elementos de JSP. Scripting (Scriptlets)

- Los Scriptlets se utilizan cuando queremos hacer código complejo en Java para ser introducido en el método `_jspService`.
- Pueden ser sentencias independientes de Java o un conjunto de ellas agrupadas.
- Sintaxis

```
<% Código Java %>
<% out.println( "Hola Mundo" ) %>
<%
for (int i=0; <10;i++ )
    out.println(i);
%>
```

© JAA 2013

## Elementos de JSP. Scripting (Scriptlets)

- Los Scriptlets tienen acceso a las mismas variables definidas automáticamente como lo hacen las expresiones (request, response, session, out, etc.).
- Los Scriptlets pueden realizar una serie de tareas que no se pueden lograr con expresiones solo:
  - ✓ Modificar las cabeceras de respuesta,
  - ✓ Ajustar códigos de estado
  - ✓ Acceder y gestionar Bases de Datos
  - ✓ Creación de Bucles, condicionales, etc.

```
<HTML>
<HEAD> <TITLE>Color Testing</TITLE> </HEAD>
<% String bgColor = request.getParameter("bgColor");
   if (bgColor == null) { bgColor = "WHITE"; }
%>
<BODY BGCOLOR=<%= bgColor %>>
<H2 ALIGN="CENTER">Testing a Background of "<%= bgColor %>"</H2>
</BODY>
</HTML>
```

© JAA 2013

## Elementos de JSP. Scripting (Scriptlets)

### Condicionales

- Otro uso de los scriptlets es proporcionar una salida condicional a un código HTML.
- El código de un scriptlet se inserta dentro del método `_jspService` del servlet resultante, tal y como está escrito de tal manera que se convierte en texto estático que se imprimirá.

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



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

© JAA 2013

## Elementos de JSP. Scripting (Scriptlets)

### XML y sintaxis Especial de Scriptlets

- Al igual que las anteriores, a partir de JSP 1.2, los servidores apoyan otra sintaxis para definir Scriptlets
- Esta sintaxis está basada en XML y es conveniente no mezclar ambas sintaxis.

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

```
<%
for (int i=0; <10;i++ )
    out.println(i);
%>
```

equivalencia

```
<jsp:scriptlet>
for (int i=0; <10;i++ )
    out.println(i);
</jsp:scriptlet>
```

© JAA 2013

## Objetos implícitos

- Dentro de JSP tenemos una serie de objetos ya creados y de uso público.
- Son instancias de clases definidas en Servlets y en las especificaciones JSP (hay 8 objetos implícitos ya creados).
- Son utilizadas para simplificar el código en las expresiones JSP y scriptlets.
- Estos objetos implícitos trabajan en ámbitos(scope) diferentes y no todos están disponibles en todos los ámbitos
  - ✓ Página (page)
  - ✓ Petición (request)
  - ✓ Sesión (session)
  - ✓ Aplicación (application)

© JAA 2013

## Objetos implícitos

Objeto	Clase Java
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
pageContext	javax.servlet.jsp.pageContext
session	javax.servlet.http.Session
application	javax.servlet.ServletContext
out	javax.servlet.jsp.JspWriter
page	java.lang.Object
exception	java.lang.Throwable

© JAA 2013

## Objetos implícitos

### Request.

- Es idéntico al parámetro de los métodos en los Servlet.
- Nos permite tener acceso a los parámetros de la petición, el tipo de solicitud (por ejemplo, GET o POST), y las cabeceras HTTP entrantes (cookies)
- Todos los métodos que utiliza el Objeto Servlet es asignado a este objeto implícito.

```
<% if ( request.getParameter("Nombre") == NULL {  %>
    <BR> Introducir un Nombre
<% } else {  %>
    <BR> Bienvenido <%= request.getParameter("Nombre") %>
<% } %>
```

© JAA 2013

## Objetos implícitos

### Response.

- Es idéntico al parámetro de los métodos en los Servlet.
- Nos permite tener acceso a la respuesta HTTP que se envía de vuelta al cliente.
- Todos los métodos que utiliza el Objeto Servlet es asignado a este objeto implícito.

```
addCookies()  
setContentLength()  
sendRedirect(URL)  
:
```

© JAA 2013

## Objetos implícitos

### out.

- Es un objeto idéntico al usado con `response.getWriter()` en los Servlets para generar la salida de una página.
- Es utilizado para mostrar información en la página HTML
- La variable OUT se utiliza casi exclusivamente en scriptlets.  
 ↘ El scriptlet `<% out.print (expresión) ; %>` hace que el resultado de la expresión que se muestra en el navegador del cliente.
- Métodos:

<code>out.println (String )</code>	Muestra la línea en la salida
<code>out.getBufferize()</code>	Tamaño del buffer en bytes
<code>out.getRemaining()</code>	Número de bytes no usados

© JAA 2013

## Objetos implícitos

### pageContext.

- Se utiliza para acceder a todo el ámbito del JSP y a sus atributos.
- La variable *pageContext* almacena el valor del objeto *PageContext* asociado con la página actual de instancia única por JSP.
- Es utilizado cuando un método o constructor necesita tener acceso a varios objetos relacionadas con la página.
- La clase *PageContext* define varios campos, incluyendo PAGE\_SCOPE, REQUEST\_SCOPE, SESSION\_SCOPE y APPLICATION\_SCOPE, que identifican los cuatro ámbitos posibles .

```
pageContext.removeAttribute("attrName", PAGE_SCOPE);
<%= pageContext.getException().toString() %>
```

© JAA 2013

## Objetos implícitos

### pageContext.

- Métodos

findAttribute ( String )	Busca un atributo con el nombre indicado. (Bean)
getAttribute ( String )	Devuelve el objeto si lo encuentra. (Bean)
getException ()	Devuelve una excepción producida.
getRequest ()	Devuelve el objeto Request
getResponse()	Devuelve el objeto Response

```
<%= pageContext.getException().toString() %>
```

© JAA 2013

## Objetos implícitos

### Application.

- Este objeto representa toda la aplicación ( páginas JSP y páginas de Error), también llamado *ServletContext*, equivale a `this.getServletContext()` dentro del servlet.
- Todas las páginas que componen la aplicación se pueden acceder desde aquí.
- *ServletContext* tiene métodos `setAttribute` y `getAttribute` que le permiten almacenar datos arbitrarios asociados a claves especificadas.
- Almacenamiento de datos
  - ✓ Variables de instancia
    - Las variables de instancia sólo están disponibles para el mismo servlet que almacena los datos.
  - ✓ Almacenamiento en el *ServletContext*:
    - Los datos son compartidos por todos los servleets de la aplicación web,

© JAA 2013

## Objetos implícitos

### Application.

- Métodos

<code>getContext ( URL )</code>	Devuelve el Contexto de una URL específico del Web Container
<code>getMajorVersion ()</code>	Devuelve la versión del Servlet API ( API 2.0 → 2 )
<code>getMinorVersion()</code>	Devuelve la versión del Servlet API ( API 2.0 → 0 )
<code>getRealPath(...)</code>	Devuelve el PATH Absoluto
<code>getServerInfo()</code>	Devuelve el nombre y versión del Servlet Container ( Tomcat / 3.2)

`<%= application.getMajorVersion() %>`

© JAA 2013

## Objetos implícitos

### page.

- Es un sinónimo de *this* y apenas es usada.
- Actualmente es reemplazada por pageContext

### exception

- El objeto *exception* es una instancia de una subclase de Throwable (por ejemplo, java.lang.NullPointerException) y sólo está disponible en las páginas de error.
- Deberemos introducir la directiva <%@ page isErrorPage = "true"%>, para que el objeto implícito esté definido.

✓ `toString()`

✓ `getMessage()`

- Con <%@ page isErrorPage = "error.jsp" %> se redirige a una pagina de error

© JAA2013

## Expression Language (EL)

- El lenguaje de expresión (EL) simplifica la accesibilidad de los datos almacenados en el componente Java Bean y otros objetos como request, session, application, etc.
- El marcador \${expresión} extrae el valor de la expresión.
- Hay muchos objetos implícitos, operadores y palabras de reserva en EL.
- Es la característica agregada a JSP en la versión 2.0.

© JAA2013

JSP

# ACCIONES Y ETIQUETAS

© JAA 2013

## Elementos de JSP. Acciones

- Son bloques de código encapsulado con una funcionalidad determinada.
- Las acciones, como los scriptlets, se procesan cuando se solicita una página, por lo que pueden operar sobre objetos predefinidos.
- Son usadas para trabajar con JavaBean, JDBC (acceso a BBDD), Control de página, etc
- Sintaxis:

```
<jsp:action-name attr1="value1" [attr2="value2" ...]>
```

```
...
```

```
</jsp:action-name>
```

```
<jsp:getProperty>.....
```

```
<jsp:useBeans>.....
```

```
<jsp:forward page=>.....
```

© JAA 2013

## Elementos de JSP. Acciones

- Hay 8 acciones estándar en JSP y 5 sub-acciones (*sólo pueden aparecer en el cuerpo de las anteriores*).
- Acciones
 

forward	include
useBean	setProperty
getProperty	text
element	plugin
- Sub-Acciones
 

param	params
attribute	body
fallback	

© JAA 2013

## JavaBeans

- Los **JavaBeans** son un modelo de componentes creado por Sun Microsystem para la construcción de aplicaciones en Java.
- Los JavaBeans encapsulan uno o varios objetos en un único objeto (Bean), para hacer uso como si fuese un solo objeto.
- Para funcionar como una clase JavaBean, una clase debe obedecer ciertas convenciones sobre nomenclatura de métodos, construcción y comportamiento.
- Las convenciones requeridas son:
  - Debe tener un constructor sin argumentos.
  - Sus propiedades deben ser accesibles mediante métodos **get** y **set** que siguen una convención de nomenclatura estándar.
  - Debe implementar serializable.

© JAA 2013

## Introducción

- En todos los Java Beans se distinguen:
  - Propiedades Atributos que contiene
  - Métodos Métodos get y set para acceder y modificar los atributos.
  - Eventos Permiten comunicar con otros JavaBeans.

```
public class PersonaBean implements java.io.Serializable {
    private String nombre;
    private int edad;

    public PersonaBean() {
        // Constructor sin argumentos
    }

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

    public void setEdad(int e) {
        this.edad = e;
    }

    public String getNombre() { return (this.nombre); }

    public int getEdad() { return (this.edad); }
}
```

Serializable

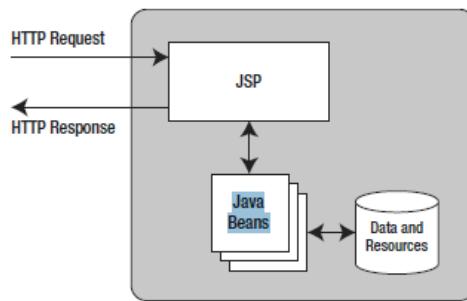
Constructor sin argumentos

Métodos get y set

© JAA 2013

## Acciones Beans

- La acción *jsp:useBean* le permite cargar un bean para ser utilizado en la página JSP.
- Los Beans permiten explotar la reutilización de clases Java y el encapsulamiento de las mismas. (*sé lo que hacen pero no como lo hacen*)



© JAA 2013

## Acciones Beans

- Dentro de JSP podemos distinguir 2 tipos de JavaBeans:
  - Supuestos JavaBeans (Clases simples)
  - Verdaderos JavaBeans (Clases independientes)
- "Supuestos" JavaBeans:
  - Son aquellos que se utilizan como si fueran JavaBeans aunque son objetos independientes de clases conocidas. (No Beans reales).
  - Se definen de la misma forma que los Verdaderos Beans

```
<jsp:useBeans id="Nombre" class=".Paquete.class." />
```

```
<jsp:useBeans id="Fecha" class="java.util.Date" />
<jsp:useBeans id="Operacion" class="java.util.Calendar" />
```

*crear una instancia un objeto de la clase especificada por clase, y se unen a una variable con el nombre especificado por id.*

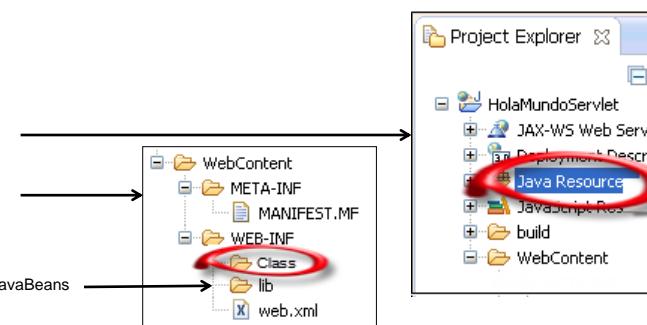
© JAA 2013

## Acciones Beans

- La definición de la clase JavaBean es código Java que debe colocarse en el CLASSPATH del servidor (en los mismos directorios en los servlets), y no en el directorio que contiene el archivo JSP.

- Habitualmente en:

- Java Resources
- /WEB-INF/classes
- /WEB-INF/lib
  - ✓ Ficheros JAR con JavaBeans



© JAA 2013

## Acciones Beans: Instancias

- La creación de una variable Java Beans (instanciación) tiene una serie de especificaciones:
    - Las variables Beans pueden ser asociadas a un ámbito determinado: (page, request, session, application)
    - Los Beans pueden ser compartidos, por lo que deberemos de ser capaces de referenciar los existentes para evitar la construcción de un nuevo objeto.
    - La acción *jsp:useBean* especifica que un nuevo objeto se instancia sólo si no hay uno ya existente con el mismo id y ámbito.
    - El atributo *class* y *beanName* son equivalentes.
- ```
<jsp:useBean id= "NombreBEANS" scope="application" class="eshop.model.DataManager" />
<jsp:useBean id= "NombreBEANS" scope="application" beanName="eshop.model.DataManager" />
```

© JAA 2013

## Acciones Beans: Alcance [scope]

- El alcance del Beans define en que parte de la aplicación Web es visible la variable asignada al objeto Beans.

| Ámbito      | Descripción                                                                                                                                                                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PAGE        | <ul style="list-style-type: none"> <li>- Valor por defecto sino se especifica nada</li> <li>- El Objeto Beans está ligado a una variable local, y está colocado en el objeto <b>PageContext</b> para la duración de la solicitud actual.</li> <li>- El objeto puede ser accedido utilizando <u>getAttribute</u> de <u>pageContext</u></li> </ul> |
| REQUEST     | <ul style="list-style-type: none"> <li>- El Beans es accesible por cualquier otro JSP que haya sido invocado por medio de las acciones <u>forward</u> e <u>include</u>. (ServletRequest)</li> </ul>                                                                                                                                              |
| SESSION     | <ul style="list-style-type: none"> <li>- El bean es almacenado en el objeto <b>HttpSession</b>, por lo que es accesible a toda la sesión generada por la petición.</li> </ul>                                                                                                                                                                    |
| APPLICATION | <ul style="list-style-type: none"> <li>- El bean es almacenado en el objeto <b>ServletContext</b>, por lo que es accesible a toda la aplicación, es decir, por todos los Servlets que la conforman (web.xml).</li> </ul>                                                                                                                         |

© JAA 2013

## Acciones Beans

### useBeans

- Permite declarar una nueva variable de scripting JSP y asociarle un objeto JavaBean.
- JSP utiliza esta variable para acceder a los métodos del objeto sin tener que preocuparse por su ubicación y ejecución.
- Sintaxis

```
<jsp:useBean id= "Nombre" scope=" ..... " class=" ..... " />
<jsp:useBean id= "NombreBEANS" scope="application" class="eshop.model.DataManager"/>
```

- Sólo la Id es obligatorio; Tomcat comprobará si un objeto denominado NombreBEANS existe en pageContext .

- ✓ Si existe, Tomcat creará una variable llamada Nombre del mismo tipo que el objeto, para que pueda acceder al objeto.
- ✓ Si el objeto no existe, Tomcat lanzará un *java.lang.InstantiationException* .

© JAA2013

## Acciones Beans

### setProperty y getProperty

- Son propiedades relacionadas con useBeans.
- Cada propiedad de un JavaBean debe de tener asociada un método get y un método set en su implementación.
- Estas Acciones nos permiten modificar/recuperar el valor de un atributo previamente definido en la variable Beans.
- Sintaxis

```
<jsp:useBean id= "Nombre" scope=" ..... " class=" ..... " />
<jsp:getProperty name= "Nombre" property="nombre_propiedad" />
<jsp:setProperty name= "Nombre" property="nombre_propiedad" value="valor" />
```
- Para mostrar el valor de una propiedad se utiliza getProperty:

```
<jsp:getProperty name="book1" property="title" /> o <%= book1.getTitle() %>
```

© JAA2013

## Acciones Beans

### setProperty y getProperty

- El uso de `setProperty`, es similar, pero disponemos de 2 posibilidades de cambio:
  1. Se mapea la propiedad del Bean y su valor  
`<jsp:setProperty name="book1" property="Title" value="Core Servlets and JavaServer Pages" />`
  2. Se asigna dicho valor a la propiedad  
`<% book1.setTitle("Core Servlets and JavaServer Pages"); %>`
- Cuando trabajamos con el método `jsp:setProperty`, el valor asignado, debe de ser estático o un valor obtenido de forma dinámica del objeto *Request*.  
`<jsp:setProperty name="entry" property="itemID" value='<%= request.getParameter("itemID") %>' />`

© JAA2013

## Acciones Beans

### setProperty

- JSP también permite la asignación de parámetros de entrada (Request) a variables del objeto Beans creado.
- Esta asociación se hace mediante la cláusula *param*, en vez de *valor*

```
...
<jsp:useBean id="entry" class="msajsp.SaleEntry" />
<jsp:setProperty name="entry" property="Nombre" param="NombreEnFormulario"/>
...
```

✓ El anterior código modifica la propiedad NOMBRE del Beans ENTRY con el valor del parámetro de entrada NOMBREENFORMULARIO

- También podemos hacer la misma operación pero con todos los parámetros de Request que tengan idéntico nombre que las propiedades del Beans

```
...
<jsp:useBean id="entry" class="msajsp.SaleEntry" />
<jsp:setProperty name="entry" property="* " />
...
```

© JAA2013

## Acciones Beans

### Ejemplo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Using jsp:setProperty</TITLE>
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
    <TR><TH CLASS="TITLE">Using jsp:setProperty</TH></TR>
    <jsp:useBean id="entry" class="moreservlets.SaleEntry" />
    <jsp:setProperty name="entry" property="itemID"
        value=<%= request.getParameter("itemID") %> />
<%
    int numItemsOrdered = 1;
    try
    { numItemsOrdered =
        Integer.parseInt(request.getParameter("numItems"));
    } catch(NumberFormatException nfe) {}
%>
<jsp:setProperty name="entry" property="numItems"
    value=<%= numItemsOrdered %> />
```

```
<% double discountCode = 1.0;
try {
String discountString =
request.getParameter("discountCode");
discountCode = Double.parseDouble(discountString);
}
catch(NumberFormatException nfe) {}
%>
<jsp:setProperty name="entry" property="discountCode"
value=<%= discountCode %> />
<BR>
<TABLE ALIGN="CENTER" BORDER=1>
    <TR CLASS="COLORED">
        <TH>Item ID</TH>Unit Price</TH>Number Ordered</TH>Total
        Price
        <TR ALIGN="RIGHT">
            <TD><jsp:getProperty name="entry" property="itemID" />
            <TD>$<jsp:getProperty name="entry" property="itemCost" />
            <TD><jsp:getProperty name="entry" property="numItems" />
            <TD>$<jsp:getProperty name="entry" property="totalCost" />
        </TR>
    </TABLE>
</BODY>
</HTML>
```

© JAA 2013

## Otras Acciones

### forward

- Permite abortar la ejecución de la página actual y transferir la solicitud a otra página.
- ```
<jsp:forward page="OtraPagina.jsp">
    <jsp:param name="NuevoParametro" value="Valor"/>
</jsp:forward>
```

### include

- Similar a la anterior, transfiere la solicitud a otra página y cuando esta acaba, devuelve el control a la página que la ha llamado ( hace la inclusión de la página en la principal)

```
<% String dest = "/myJspPages/" + someVar; %>

<jsp:include page= "<%=dest%>" >
    <jsp:param name="Parametro" value="Valor"/>
</jsp:include>
```

<%@include file="..."%>  
Incluye la página sin ningún procesamiento  
<jsp:include page="..." />  
Incluye la página CON procesamiento

© JAA 2013

## Otras Acciones

### text

- Permite utilizar la acción para escribir texto de la plantilla.

```
<jsp:text> TEXTO </jsp:text>
```

- Dentro del texto no puede contener otros elementos, sólo texto.

### element

- Con la Acción element y con las Sub-Acciones (attribute y body), se pueden definir los elementos XML de forma dinámica dentro de una página JSP.
- Con esta acción podríamos crear dinámicamente una página XML en lugar de HTML para, por ejemplo, ser utilizada para intercambiar datos con otros módulos y aplicaciones.

```
<jsp:element name="myElem">
    <jsp:attribute name="myElemAttr">myElemAttr's value</jsp:attribute>
    <jsp:body>myElem's body</jsp:body>
</jsp:element>
```

© JAA 2013

## Otras Acciones

### plugin

- Con la Acción plugin y con las Sub-Acciones (params y fallback ), permiten insertar objetos de forma dinámica dentro de una página JSP. (Applets)
- Sintaxis

```
<jsp:plugin type="applet" code="MyApplet.class" codebase="/tests" height="100" width="100">
    <jsp:params>
        <jsp:param name="line" value="Well said!"/>
    </jsp:params>
    <jsp:fallback>Unable to start plugin</jsp:fallback>
</jsp:plugin>
```

|            |   |
|------------|---|
| params →   | parámetros pasados al objeto                          |
| fallback → | Mensaje de error si no es posible encontrar el objeto |

© JAA 2013

## Debilidades de los scriptlets

- El código Java embebido en scriptlets es desordenado.
- Un desarrollador que no conoce Java no puede modificar el código Java embebido, anulando uno de los mayores beneficios de los JSP: permitir a los diseñadores y personas que escriben la lógica de presentación que actualicen el contenido de la página.
- El código de Java dentro de scriptlets JSP no pueden ser reutilizados por otros JSP, por lo tanto la lógica común termina siendo re-implementado en múltiples páginas.
- La recuperación de objetos fuera del HTTP Request y Session es complicada. Es necesario hacer el Casting de objetos y esto ocasiona que tengamos que importar más Clases en los JSP.

© JAA 2013

## Etiquetas personalizadas

- JSP 1.1 introdujo una nueva función muy valiosa: la capacidad de crear sus propias etiquetas JSP (acciones).
- Podemos definir TAG's propios con sus atributos y usos; podemos agruparlos en colecciones (librerías de etiquetas) y estas librerías pueden ser utilizadas en cualquier JSP.
- Estas TAG's libraries permiten a los desarrolladores:
  - ✓ Utilizar elementos simples para tareas complejas.
  - ✓ La incorporación rápida de nuevas funcionalidades en sus páginas JSP.
  - ✓ Referenciar objetos que se encuentren en los ambientes Request y Session sin conocer el tipo del objeto y sin necesidad de hacer el Casting.
- Las etiquetas personalizadas ofrecen algunas características similares a los Beans (`jsp:useBean`), aunque con diferencias

© JAA 2013

## JSP. Etiquetas personalizadas

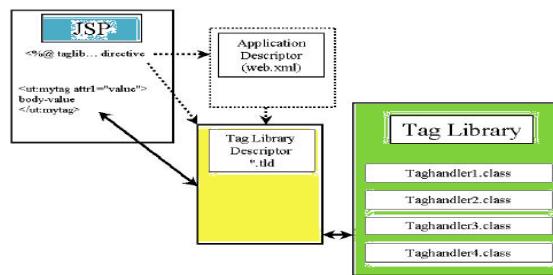
- Diferencias JavaBeans y Tag's Personalizadas.

| Java Beans  | Tag's Personalizado  |
|---|--|
| - Los JavaBeans NO pueden manipular el contenido de JSP.  | - Etiquetas personalizadas SI pueden manipular el contenido de JSP.  |
|   | - Las operaciones complejas se pueden reducir a una forma mucho más sencilla con etiquetas personalizadas que con Beans. |
| - JavaBeans requieren menos trabajo de configuración que las TAG's Personalizadas                       |  |
| - Son definidos en un Servlet/JSP y luego pueden ser usados en otro servlet o una página JSP diferente. | - Definen un comportamiento relativamente autónomo.  |
| - Disponible en todas las versiones 1.X   | - Sólo disponibles a partir de JSP 1.1   |

© JAA 2013

## Tag Library. Componentes

- Para crear etiquetas JSP personalizadas, es necesario definir tres componentes separados:
  - Clase Controladora (\*.java → \*.class)
    - En ella se implementa toda la funcionalidad de la nueva etiqueta.
  - Descriptor de la Librería de Etiquetas (\*.TLD)
    - En él se asigna los nombres de los elementos XML a las implementaciones de etiquetas.
  - Archivo JSP que utiliza la biblioteca de etiquetas. (\*.jsp)



© JAA 2013

## Tag Library. Componentes

### Clase Controladora

- Al definir una nueva etiqueta, la primera tarea es definir una clase Java que le dice al sistema qué hacer cuando ve la etiqueta.
- Esta clase debe implementar la interfaz `javax.servlet.jsp.tagext.Tag` y por lo general, se crea extendiendo de la clase `TagSupport` o clase `BodyTagSupport`.

```
public class ExampleTag extends TagSupport {
    public int doStartTag() {
        try {
            JspWriter out = pageContext.getOut();
            out.print("Etiqueta personalizada");
        } catch(IOException ioe) {
            System.out.println("Error in ExampleTag: " + ioe);
        }
        return(SKIP_BODY);
    }
}
```

Nueva Etiqueta que devuelve «Etiqueta Personalizada»

© JAA 2013

## Tag Library. Componentes

### Clase Controladora

- En la interfaz Tag:
  - ✓ `doStartTag()`: se realizará al inicio de la etiqueta.
  - ✓ `doEndTag()`: se realizará al final de la etiqueta.
- Hay cuatro campos definidos en la interfaz de la etiqueta:
  - ✓ `EVAL_BODY_INCLUDE`: evalúa el contenido del cuerpo de la etiqueta.
  - ✓ `EVAL_PAGE`: evalúa el contenido de la página JSP después de la etiqueta personalizada.
  - ✓ `SKIP_BODY`: omite el contenido del cuerpo de la etiqueta.
  - ✓ `SKIP_PAGE`: omite el contenido de la página JSP después de la etiqueta personalizada.
- En la interfaz IterationTag:
  - ✓ `doAfterBody()`: se realizará después de la evaluación del cuerpo, `EVAL_BODY AGAIN` reevalúa el contenido del cuerpo.

© JAA 2013

# Tag Library. Componentes

## Descriptor TLD

- En este fichero, asociamos la clase implementada anteriormente con el nombre de un etiqueta XML particular.
- Esta tarea se lleva a cabo por medio de este archivo en formato XML.
- Este archivo contiene:
  - ✓ Cabecera de fichero indicando el tipo de elemento a crear.
  - ✓ Uri (Definir de modo único nuestra librería)
  - ✓ Un prefijo de la etiqueta ( *shortname* )
  - ✓ Una breve descripción
  - ✓ Nombre de la etiquete y sus TAG de definición de dicha etiqueta .

© JAA 2013

# Tag Library. Componentes

## Descriptor TLD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>

  <shortname>mia</shortname>

  <info> Etiqueta de prueba </info>
  <uri> http://...../librerias </uri>

  <tag>
    <name>example</name>
    <tagclass>Paquete.ExampleTag</tagclass>
    <bodycontent>empty</bodycontent>
  </tag>
  ...
</taglib>
```

© JAA 2013

## Tag Library. Componentes

### Atributos

- Se pueden definir atributos para cualquier etiqueta personalizada.
- Hay que definir una propiedad en la clase con el nombre del atributo y defina el método de establecimiento.
- Hay que registrar el atributo en el descriptor TLD.
 

```
<attribute>
        <name>repit</name>
        <required>false</required>
      </attribute>
```
- En el archivo contiene:
 

```
<%@ taglib uri="WEB-INF/ejemplos.tld" prefix="m"%>

<m:example repit="3" />
```

© JAA 2013

## Tag Library. Componentes

### Fichero JSP

- Una vez que tengamos la Clase de soporte y el descriptor de la etiqueta, estamos listos para escribir un archivo JSP que haga uso de la etiqueta.
- Antes del primer uso de su etiqueta, es necesario utilizar la directiva **taglib**  
*Directiva para incluir la librería de Tags y nuestra etiqueta*
- Esta Directiva tiene la siguiente forma:
 

```
<% @ Taglib uri = "..." prefix = "..." %>
```
- El uri puede ser relativo o absoluto
  - ✓ Relativo: El fichero TLD debe de estar en el mismo directorio que el JSP donde se utiliza la etiqueta.
  - ✓ Absoluto: Se modifica en el fichero WEB.XML

© JAA 2013

## Tag Library. Componentes

### Fichero JSP

- Habitualmente los ficheros TLD se ubican en un subdirectorio de WEB-INF. (de esta forma los JSP superiores pueden utilizar las etiquetas sin problemas).
- URL Absoluto:
  - ✓ Modificaremos nuestro descriptor de despliegue para que acepte URI absolutos de la siguiente manera:

```

:
<taglib>
  <taglib-uri>
    http://.....librerías
  </taglib-uri>
  <taglib-location>
    /WEB-INF/jsp/msajsp-taglib.tld
  </taglib-location>
</taglib>
:

```

© JAA 2013

## Tag Library. Componentes

### Fichero JSP. Ejemplo

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>

  <HEAD>
    <%@ taglib uri="msajsp-taglib.tld" prefix="mia" %>
    <TITLE>Ejemplo de etiqueta</TITLE>
    <LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
  </HEAD>

  <BODY>
    <H1><mia:example /></H1>
    <mia:example />
  </BODY>

</HTML>

```

© JAA 2013

## Desventajas de las etiquetas

- Las etiquetas pueden agregar mayor sobrecarga en el servidor. Los scriptlets y las librerías de etiquetas son compilados a servlets, los cuales luego son ejecutados por el contenedor.
  - El código Java embebido en los scriptlets es básicamente copiado en el servlet resultante.
  - Las etiquetas generan una mayor cantidad de código en el servlet.
  - En la mayoría de casos esta cantidad no es mensurable pero debe ser considerado.
- Los scriptlets son más potentes que las etiquetas.
  - A pesar que las etiquetas proporcionan conjuntos de librerías reutilizables, no pueden competir con todo lo que el código Java puede hacer en un scriptlets.
  - La librerías de etiquetas disponen de una funcionalidad concreta y están diseñada para evitar o reducir la codificación en el lado de presentación.

© JAA 2013

JSP Standard Tag Library

**JSTL**

© JAA 2013

## JSTL

- JSTL es un conjunto de librerías de etiquetas simples y estándares que encapsulan la funcionalidad principal que es usada comúnmente para escribir páginas JSP.
- Las etiquetas JSTL están organizadas en librerías:
  - core: Comprende las funciones script básicas como loops, condicionales, y entrada/salida.
  - xml: Comprende el procesamiento de xml
  - formatting: Comprende la internacionalización y formato de valores como de moneda y fechas.
  - sql: Comprende el acceso a base de datos.
  - functions: Comprende funciones comunes de manipulación de cadenas
- Las librerías incluyen la mayoría de funcionalidad que será necesaria en una página JSP. Las etiquetas JSTL son muy sencillas de usar por personas que no conocen programación y solo cuentan con conocimientos de etiquetas del estilo HTML.

© JAA 2013

## Instalación y configuración del JSTL

- La librería JSTL es distribuida como un conjunto de archivos JAR que simplemente tenemos que agregar en el classpath del contenedor de servlets.
- Debemos usar un contenedor de servlets compatible con la versión JSP 2.0 para usar el JSTL 1.1.
- Descargar la implementación JSTL de la página de proyecto Jakarta TagLibs [<http://tomcat.apache.org/taglibs/>].
  - Copiar todos los archivos JAR que se encuentran en jakarta-taglibs/standard-1.0/lib al directorio /WEB-INF/lib de la aplicación Web.
- Importar en las páginas JSP las librerías JSTL necesarias mediante las directivas taglib apropiadas:
 

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

© JAA 2013

## jstl/core

<c:out >	Al igual que <% = ...>, pero las expresiones adecuadas.
<c:set >	Establece el resultado de la evaluación de la expresión en un ámbito.
<c:remove >	Elimina una variable del ámbito (o de un ámbito concreto, si se especifica).
<c:catch>	Atrapa cualquier excepción que se produce en su cuerpo y, opcionalmente, la expone.
<c:if>	Condicional simple ejecuta su cuerpo si la condición es verdadera.
<c:choose>	Código condicional múltiple que establece un contexto para las operaciones condicionales que se excluyen mutuamente, marcadas por <c:when> y <c:otherwise>.
<c:when>	Subetiqueta de <c:choose> que incluye su cuerpo si su expresión es verdadera.
<c:otherwise>	Subetiqueta de <c:choose> que sigue a las etiquetas <c:when> y se ejecuta sólo si todas las condiciones anteriores son falsas
<c:import>	Recupera una URL absoluta o relativa y expone su contenido en la página.
<c:forEach >	Recorre una colección.
<c:forTokens>	Recorre los tokens suministrados, separados por delimitadores.
<c:param>	Añade un parámetro a la URL que contiene una etiqueta de "importación".
<c:redirect >	Redirige a una nueva URL.

© JAA 2013

## jstl/core

```

<c:if test="${user.visitCount == 1}">
This is your first visit. Welcome to the site!
</c:if>

<c:choose>
    <c:when test="${count == 0}">No records matched your selection.</c:when>
    <c:otherwise>
        <c:out value="${count}" /> records matched your selection.
    </c:otherwise>
</c:choose>

<table>
    <c:forEach var="customer" items="${customers}">
        <tr><td><c:out value="${customer}" /></td></tr>
    </c:forEach>
</table>

```

© JAA 2013

## jstl/fmt

<fmt:formatNumber>	Para hacer que el valor numérico con precisión o formato específico.
<fmt:parseNumber>	Analiza la representación de cadena de un número, moneda, o porcentaje.
<fmt:formatDate>	Formatea una fecha y / u hora utilizando los estilos y patrones suministrados
<fmt:parseDate>	Analiza la representación de cadena de una fecha y / u hora
<fmt:bundle>	Carga un paquete de recursos para ser utilizados por el cuerpo de la etiqueta.
<fmt:setLocale>	Almacena la configuración regional dada en la variable de configuración local.
<fmt:setBundle>	Carga un paquete de recursos y lo almacena en la variable llamada de ámbito o la variable de configuración paquete.
<fmt:timeZone>	Especifica la zona horaria para la acción de formato de tiempo o de análisis anidados en su cuerpo.
<fmt:setTimeZone>	Almacena la zona horaria dada en la variable de configuración de zona horaria
<fmt:message>	Para mostrar un mensaje de internacionalización.
<fmt:requestEncoding>	Establece la codificación de caracteres de solicitud

© JAA 2013

## jstl/fmt

```

<fmt:message key="athletesRegistered">
    <fmt:param>
        <fmt:formatNumber value="${athletesCount}" />
    </fmt:param>
</fmt:message>

<fmt:formatNumber value="9876543.21" type="currency"/>

<jsp:useBean id="now" class="java.util.Date" />
<fmt:formatDate value="${now}" timeStyle="long" dateStyle="long"/>
<fmt:formatDate value="${now}" pattern="dd.MM.yy"/>

<fmt:parseDate value="4/13/02" var="parsed" />

```

© JAA 2013

## jstl/xml

<x:out>	Igual que <%= ...>, pero para las expresiones XPath.
<x:parse>	Se utiliza para analizar los datos XML especificados ya sea a través de un atributo o en el cuerpo de la etiqueta.
<x:set>	Establece una variable para el valor de una expresión XPath.
<x:if>	Evalúa una expresión XPath, si es cierta se procesa de su cuerpo pero si es falsa, el cuerpo se ignora.
<x:forEach>	Para recorrer los nodos de un documento XML.
<x:choose>	Código condicional múltiple que establece un contexto para las operaciones condicionales que se excluyen mutuamente, marcadas por <x:when> y <x:otherwise>.
<x:when>	Subetiqueta de <x:choose> que incluye su cuerpo si su expresión es verdadera.
<x:otherwise>	Subetiqueta de <x:choose> que sigue a las etiquetas <x:when> y se ejecuta sólo si todas las condiciones anteriores son falsas
<x:transform>	Se aplica una transformación XSL en un documento XML
<x:param>	Utilizada junto con la etiqueta de transformación para establecer un parámetro en la hoja de estilo XSLT

© JAA 2013

## jstl/xml

```

<!-- parse an XML document -->
<c:import url="http://acme.com/customer?id=76567" var="xml"/>
<x:parse xml="${xml}" var="doc"/>
<!-- access XML data via XPath expressions -->
<x:out select="$doc/name"/>
<x:out select="$doc/address"/>
<!-- set a scoped variable -->
<x:set var="custName" scope="request" select="$doc/name"/>

<!-- context set by ancestor tag <x:forEach> -->
<x:forEach select="$doc//customer">
    <x:out select="name"/>
</x:forEach>

```

© JAA 2013

## jstl/sql

<sql:setDataSource>	Crea un origen de datos sencillo apto sólo para la creación de prototipos
<sql:query>	Ejecuta la consulta SQL definida en su cuerpo o mediante el atributo sql.
<sql:update>	Ejecuta la actualización SQL definida en su cuerpo o mediante el atributo sql.
<sql:param>	Establece un parámetro en una sentencia SQL con el valor especificado.
<sql:dateParam>	Establece un parámetro en una instrucción SQL para el valor java.util.Date especificado.
<sql:transaction >	Proporciona elementos de acción de base de datos anidados con una conexión compartida, configura la ejecución todas las declaraciones como una sola transacción.

© JAA 2013

## jstl/sql

```

<sql:setDataSource var="dataSource" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/sakila" user="root" password="root" />
<sql:query var="actores" dataSource="${dataSource}">
    SELECT * FROM actor ORDER BY first_name, last_name
</sql:query>
<table>
    <c:forEach var="row" items="${actores.rows}">
        <tr>
            <td><c:out value="${row.first_name}" /></td>
            <td><c:out value="${row.last_name}" /></td>
        </tr>
    </c:forEach>
</table>

```

© JAA 2013

## jstl/functions

fn:contains()	Comprueba si una cadena de entrada contiene la sub-cadena especificada.
fn:containsIgnoreCase()	Comprueba si una cadena de entrada contiene la sub-cadena especificada de una manera insensible a minúsculas y mayúsculas.
fn:endsWith()	Comprueba si una cadena de entrada termina con el sufijo especificado.
fn:escapeXml()	Escapa los caracteres que podrían ser interpretados como marcado XML.
fn:indexOf()	Devuelve el índice de la primera aparición de una sub-cadena especificada dentro de una cadena.
fn:join()	Une todos los elementos de una matriz en una cadena.
fn:length()	Devuelve el número de elementos en una colección o el número de caracteres de una cadena.

© JAA 2013

## jsp/jstl/functions

fn:replace()	Devuelve una cadena resultante de la sustitución en una cadena de entrada de todas las ocurrencias con una cadena dada.
fn:split()	Divide una cadena en una matriz de sub-cadenas.
fn:startsWith()	Comprueba si una cadena de entrada comienza con el prefijo especificado.
fn:substring()	Devuelve una sub-cadena de una cadena.
fn:substringAfter()	Devuelve una sub-cadena de una cadena después de una sub-cadena específica.
fn:substringBefore()	Devuelve una sub-cadena de una cadena antes de una sub-cadena específica.
fn:toLowerCase()	Convierte todos los caracteres de una cadena a minúsculas.
fn:toUpperCase()	Convierte todos los caracteres de una cadena a mayúsculas.
fn:trim()	Elimina los espacios en blanco de ambos extremos de una cadena.

© JAA 2013

## jsp/jstl/functions

```
<c:if test="${fn:contains(saludo, 'Hola')}">
    <p>Hola mundo<p>
</c:if>

${fn:replace(saluda, "Hello", "Hola")}> ${fn:toUpperCase(nombre)};

<c:set var="str1" value="Welcome-to-JSP-Programming."/>
<c:set var="str2" value="${fn:split(str1, '-')}" />
<c:set var="str3" value="${fn:join(str2, ' ')}" />
```

© JAA 2013

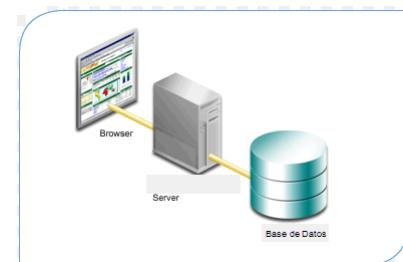
JSP

## JDBC - JSLT SQL

© JAA 2013

## Introducción

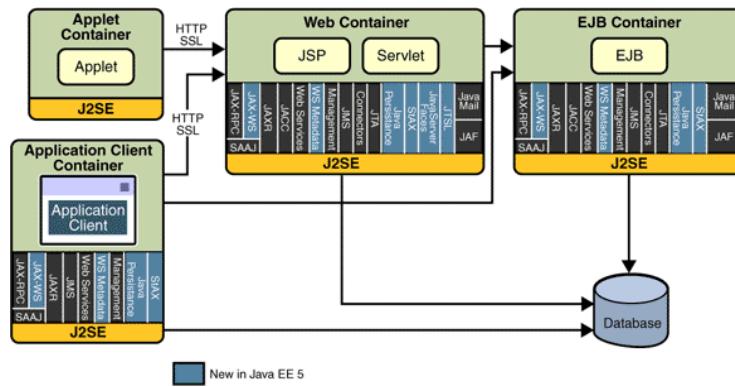
- Casi todas las aplicaciones web's son una interfaz para mostrar datos obtenidos de almacenamientos internos.
- El almacenamiento mas utilizado son las bases de datos (DB).
- De hecho, lo que hace que las páginas web dinámica es precisamente el hecho de que hay una cantidad significativa de datos detrás de ellos.
- Una base de datos se compone de:
  - Unas estructuras de datos
  - Unas formas de acceso (esquemas)
  - Y Datos.
- Hoy en día, la mayoría de los SGBD son relacionales, es decir, sus datos están organizados en tablas.



© JAA42013

## Introducción

- Una estructura habitual de una aplicación Empresarial / J2EE es la siguiente:



© JAA42013

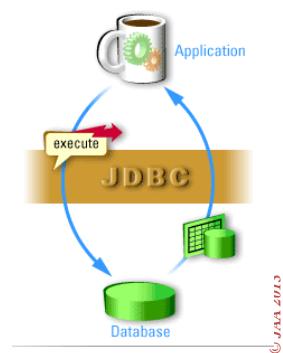
## Introducción

- JDBC proporciona una biblioteca estándar para acceder a bases de datos relacionales.
- Mediante el uso de la API de JDBC, puede acceder a una amplia variedad de bases de datos SQL con exactamente la misma sintaxis de Java.
- La API JDBC estandariza:
  - ✓ La conexión a bases de datos
  - ✓ La sintaxis para el envío de consultas
  - ✓ La confirmación de las transacciones
  - ✓ La estructura de datos que representa el resultado,
- La mayoría de las consultas siguen la sintaxis SQL estándar, y disponemos de múltiples posibilidades: *Cambio de BBDD, Hosts, Puertos, etc*

© JAA2013

## Pasos Generales JDBC

- Para realizar una conexión JDBC debemos realizar los siguiente pasos:
  1. Cargar el controlador JDBC.  
✓ Especificamos el nombre de clase del controlador de base de datos en el método `Class.forName`.
  2. Definir el URL de conexión con BBDD  
✓ Una URL de conexión especifica el host del servidor, el puerto y el nombre de base de datos con la que establecer una conexión.
  3. Establecer la conexión  
✓ Necesitaremos un usuario y password correctos.
  4. Preparar y Ejecutar consulta  
✓ Crear, modificar y enviar el objeto `Statement`, con una consulta específica.
  5. Procesar los resultados  
✓ Mediante el objeto `ResultSet`, se analizan y tratan los resultados obtenidos de la consulta.
  6. Cierre de la conexión



© JAA2013

## Pasos Generales JDBC

### Controladores JDBC

- Los controladores JDBC son el software que sabe cómo hablar con el servidor de base de datos real.
- Para cargar un determinado controlador, deberemos cargar la Clase de Java apropiada para ese Gestor.
 

√ Driver JDBC-ODBC:	sun.jdbc.odbc.JdbcOdbcDriver (incluido en la API de JDBC de la plataforma J2SE)
√ Driver MySQL:	com.mysql.jdbc.Driver (no incluido)
√ Driver DB2:	COM.ibm.db2.jdbc.app.DB2Driver (no incluido)
√ Driver Oracle:	oracle.jdbc.driver.OracleDriver (no incluido)
- Sino están incluidos, deberemos descargar el correspondiente **\*.jar** e insertarlo en el proyecto.

En uno de estos lugares:  
 - Project / propiedades / Java Build Path / Libraries / Add external JAR  
 - Directorio : WEB-INF/lib  
 Si necesitamos los JAR para múltiples aplicaciones → Directorio\_de\_instalación / common / lib

© JAA 2013

## Pasos Generales JDBC

### Controladores JDBC

- Sintaxis

```
try {
    Class.forName ("com.microsoft.MicrosoftDriver");
    Class.forName ("oracle.jdbc.driver.OracleDriver");
    Class.forName ("com.sybase.jdbc.SybDriver");
    Class.forName ("com.mysql.jdbc.Driver");

} catch(ClassNotFoundException cnfe) {
    System.err.println("Error loading driver: " + cnfe);
}
```

WEB.XML  
<init-param>  
<param-name>Driver</param-name>  
<param-value>com.mysql.jdbc.Driver</param-value>  
</init-param>

java.lang.Class.forName(config.getInitParameter("Driver"));

© JAA 2013

## Pasos Generales JDBC

### URL de Conexión

- Una vez que haya cargado el controlador JDBC, debe especificar la ubicación del servidor de base de datos.
- La URL de referencia utiliza:
  - ✓ Protocolo : Host : Puerto : Nombre\_de\_la\_BBDD.
- El formato exacto se define en la documentación que viene con el controlador en particular

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;

String oracleURL = "jdbc:oracle:thin:@" + host + ":" + port + ":" + dbName;
String sybaseURL = "jdbc:sybase:Tds:" + host + ":" + port + ":" + "?SERVICENAME=" + dbName;
String msAccessURL = "jdbc:odbc:" + dbName;
String MySQLURL = "jdbc:mysql://" + host + ":" + port + ":" + dbName;
```

© JAA 2013

## Pasos Generales JDBC

### Creando la Conexión

- Para hacer la conexión, deberemos pasar:
  - ✓ La dirección URL
  - ✓ Nombre de usuario de la Base de Datos
  - ✓ Contraseña del usuario anterior.
- Utilizaremos el método `getConnection` de la clase `DriverManager`,

```
String username = "Usuario01";
String password = "1234";
Connection connection = DriverManager.getConnection(oracleURL, username, password);
```

WEB.XML

```
<init-param>
<param-name>Usuario</param-name>
<param-value>Usuario01</param-value>
<param-name>Clave</param-name>
<param-value>1234</param-value>
</init-param>
```

```
String username= config.getInitParameter("Usuario")
String password= config.getInitParameter("Clave")
Connection connection = DriverManager.getConnection(oracleURL,
username, password);
```

© JAA 2013

## Pasos Generales JDBC

### Preparar y Ejecutar Consulta

- Para preparar y ejecutar una consulta en la Base de Datos necesitamos crear un objeto Statement.
- Sintaxis  
`Statement st = connection.createStatement();`
- La mayoría, pero no todos, los controladores de base de datos permiten tener múltiples objetos *Statement* abiertos la misma conexión.
- Ahora utilizaremos este objeto para insertarle una Consulta SQL y ejecutarla

```
String query = "SELECT col1, col2, col3 FROM Tabla";
ResultSet resultSet = st.executeQuery(query);          SELECT
Int filas = st.executeUpdate(query);                DML
Array[] valores = st.executeBatch(query);            SQL Multiples
```

© JAA 2013

## Pasos Generales JDBC

### Procesar los resultados

- Para procesar los resultados utilizamos el objeto ResultSet, para recorrer todos los resultados devueltos (tipo cursor)
- ResultSet proporciona diversos métodos getXXX que devuelven el resultado en una variedad de diferentes tipos de Java.
  - √ getInt → número entero
  - √ getString → Cadena de texto
  - √ getDate → Fechas
- Ahora utilizaremos este objeto para insertarle una Consulta SQL y ejecutarla

```
while(resultSet.next()) {
    System.out.println ( resultSet.getString(1) + " " + resultSet.getString(2) + " "
                        + resultSet.getString("nombre") + " " + resultSet.getString("apellido") );
}
```

Valor_c1 - 1	Valor_c2 - 1	Valor_nombre - 1	Valor_apellido - 1
Valor_c1 - 2	Valor_c2 - 2	Valor_nombre - 2	Valor_apellido - 2
Valor_c1 - 3	Valor_c2 - 3	Valor_nombre - 3	Valor_apellido - 3

© JAA 2013

## Pasos Generales JDBC

### Cerrar la conexión

- Cuando cerramos la conexión, todos los objetos del tipo *Statement* y *ResultSet* son eliminados.
- Si queremos volver a comunicarnos con la BBDD después de haber realizado el cierre de una conexión, deberemos realizar todo el proceso completo de nuevo.
- Sintaxis  
`connection.close();`

© JAA 2013

## Ejemplo completo JSP-JDBC

```
<%@page language="java" contentType="text/html"%>
<%@page import="java.sql.*"%>
<html><head><title>JDBC test</title></head><body>

<%
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/shop", "root", "");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from books");

%><table border="1"><%
ResultSetMetaData resMetaData = rs.getMetaData();
int nCols = resMetaData.getColumnCount();
%><tr><%
for (int kCol = 1; kCol <= nCols; kCol++) { out.print("<td><b>" + resMetaData.getColumnName(kCol) +
"</b></td>"); }
%></tr><%
while (rs.next()) {
%><tr><%
for (int kCol = 1; kCol <= nCols; kCol++) { out.print("<td>" + rs.getString(kCol) + "</td>"); }
%></tr><%
}
%></table><%
conn.close();
%>
</body></html>
```

© JAA 2013

## JSTL sql:query

- Cuando disponemos de JSTL, disponemos de múltiples TAG's que nos permiten realizar funcionalidades sin desarrollar nuestras propias etiquetas.
- JSTL dispone de la etiqueta <sql:query> para trabajar con BBDD.
- La etiqueta <sql:query> ejecuta una instrucción SELECT de SQL y guarda el resultado en una variable.
- Su utilización es muy sencilla:
  1. Definición del Driver, URL y Conexión
  2. Ejecución de la sentencia SELECT determinada
  3. Tratamiento de los resultados.
- La primera opción se realiza de forma única, el resto de las opciones son reutilizadas.

© JAA 2013

## JSTL sql:query

- <sql:query> dispone de unos atributos para toda su operativa:

Atributos	Descripción
sql	- Comando SQL a ejecutar (debe devolver un conjunto de resultados)
dataSource	- Conexión de base de datos a utilizar (Driver + URL + Conexión )
maxRows	- Número máximo de resultados para almacenar en la variable.
startRow	- Número de la fila en el resultado al que para devolver
var	- Nombre de la variable para representar la base de datos
scope	- Ámbito de aplicación de la variable para exponer el resultado a partir de la base de datos

© JAA 2013

## JSTL sql:query

- Sintaxis

1. Definición del Driver, URL y Conexión

```
<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/TEST"
    user="root"
    password="pass123"/>
```

2. Ejecución de la sentencia SELECT determinada

```
<sql:query dataSource="${snapshot}" var="result">
    SELECT * from Employees;
</sql:query>
```

3. Tratamiento de los resultados.

```
:
<c:forEach var="row" items="${result.rows}"> <tr>
    <td><c:out value="${row.id}" /></td>
    <td><c:out value="${row.first}" /></td>
    <td><c:out value="${row.last}" /></td>
    <td><c:out value="${row.age}" /></td> </tr>
</c:forEach>
:
```

```
<%@ page import="java.io.*;java.util.*;java.sql.*"%>
<%@ page import="javax.servlet.http.*;javax.servlet.* %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
```

© JAA

## JSTL sql:query

```
<%@ page import="java.io.*;java.util.*;java.sql.*"%>
<%@ page import="javax.servlet.http.*;javax.servlet.* %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html> <head>
<title>JSTL sql:query Tag</title> </head>
<body>
<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/TEST"
    user="root" password="pass123"/>

<sql:query dataSource="${snapshot}" var="result">
    SELECT * from Employees;
</sql:query>

<table border="1" width="100%">
    <tr> <th>Emp ID</th> <th>First Name</th> <th>Last Name</th> <th>Age</th> </tr>
    <c:forEach var="row" items="${result.rows}">
        <tr>
            <td><c:out value="${row.id}" /></td>
            <td><c:out value="${row.first}" /></td>
            <td><c:out value="${row.last}" /></td>
            <td><c:out value="${row.age}" /></td>
        </tr>
    </c:forEach>
</table> </body> </html>
```

© JAA 2013

## Frameworks ORM

- La necesidad de mover datos, con facilidad, de la BBDD a un objeto Java y viceversa, numerosos fabricantes han desarrollado Frameworks para tal fin.
- Estos frameworks son denominados Framework ORM (mapeo de objetos a bases de datos relacionales).
- Se trata de una potente capacidad ya que la programación orientada a objetos y bases de datos relacionales siempre han tenido una falta de concordancia.

✓ Castor	<a href="http://castor.exolab.org/">http://castor.exolab.org/</a>
✓ CocoBase	<a href="http://www.cocobase.com/">http://www.cocobase.com/</a>
✓ FrontierSuite	<a href="http://www.objectfrontier.com/">http://www.objectfrontier.com/</a>
✓ Kodo JDO	<a href="http://www.solarmetric.com/">http://www.solarmetric.com/</a>
✓ ObjectRelationalBridge	<a href="http://db.apache.org/ojb/">http://db.apache.org/ojb/</a>
✓ TopLink	<a href="http://otn.oracle.com/products/ias/toplink/">http://otn.oracle.com/products/ias/toplink/</a>
✓ Hibernate	<a href="http://www.hibernate.org">http://www.hibernate.org</a>

© JAA 2013

## INTEGRACIÓN JSP Y SERVLETS

© JAA 2013

## Beneficios MVC

- Los servlets son buenos en el procesamiento de datos:
  - la lectura y comprobación de datos,
  - la comunicación con bases de datos
  - la invocación de la lógica de negocio, etc.
- Las páginas JSP son buenos en la presentación:
  - la construcción de HTML para representar los resultados de las solicitudes.
- En este capítulo se describe cómo combinar servlets y páginas JSP para hacer un mejor uso de los puntos fuertes de cada tecnología

© JAA 2013

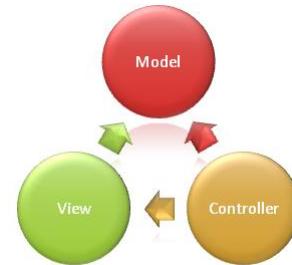
## Beneficios MVC

- El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos, la lógica de negocios y el interface de usuario.
- Para ello MVC propone la construcción de tres componentes distintos que son:
  - El modelo
    - ✓ Es el elemento que trabaja con la información, consultas, actualizaciones, accesos, etc.
    - ✓ Envía a la 'vista' aquella información que en cada momento se le solicita para que sea mostrada.
  - La vista
    - ✓ Presenta la información en un formato adecuado para interactuar (usualmente la interfaz de usuario)
  - El controlador
    - ✓ Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' .
    - ✓ Es un intermediario entre la 'vista' y el 'modelo'

© JAA 2013

## Beneficios MVC

- Aplicando el Model View Controller (MVC) a los componentes que hemos utilizado anteriormente, nos quedaría un enfoque como aparece a continuación:
  - Servlets (Controlador)
    - ✓ Gestión y Control de toda la lógica de la Aplicación.
    - ✓ Redirecciones e Invocaciones
  - JSP (Vista)
    - ✓ Presentación HTML, Font-End
  - EJB (Modelo)
    - ✓ Realiza la lógica de la aplicación, interactúa con otros componentes similares y proporciona independencia y encapsulación.



© JAA 2013

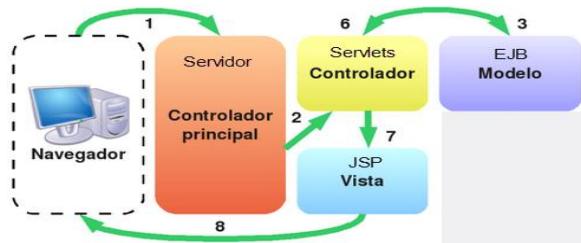
## Beneficios MVC

- ¿Porqué este enfoque?
  - La razón por la que un servlet controla la solicitud original es:
    1. Posibilidad de tratamiento de parámetros de una forma sencilla
    2. La gestión de JavaBeans, requiere una gran cantidad de programación, y es más conveniente hacerlo con un servlet
  - La razón de que la página de destino sea generalmente un documento JSP:
    1. Los documentos JSP simplifican el proceso de creación de contenido HTML.
    2. El acceso a los Jbean para obtención de datos es más simple que desde un Servlet.

© JAA 2013

## Beneficios MVC

- Ejecución de un Model View Controller ( MVC )
  1. La primera llamada se realiza al Servlet (Modelo)
    - ✓ El servlet gestiona la forma de actuar.
    - ✓ Invoca a la lógica de negocio, Acceso a datos y EBJ necesarios para proporcionar la funcionalidad determinada.
  2. Posteriormente, el Servlet decide qué página JSP es conveniente presentar los resultados particulares y reenvía la solicitud allí.
  3. La página JSP(vista) es ejecutada para crear el resultado final de la petición.



© JAA 2013

## Beneficios MVC. Frameworks

- Para implementar un modelo MVC, bastaría con los elementos anteriormente indicados (Servlets, EBJ, JSP).
- En aplicaciones muy complejas, un framework MVC más elaborado es a veces beneficioso.
- Para aplicaciones simples y poco compleja, la implementación MVC desde cero con RequestDispatcher es sencilla y flexible.
- Frameworks:
 

✓ Apple	Cocoa
✓ Apache	Struts
✓ Apache	Spring
✓ Apache	JavaServerFaces
✓ Apache	Spring .NET

Estos frameworks se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor

© JAA 2013

## Uso RequestDispatcher para implementar MVC

- El punto más importante acerca de MVC es la idea de separar la lógica de negocio y la capa de acceso a datos de la capa de presentación.
- Los pasos necesarios para este tipo de implementación son:
  1. Definir los JavaBeans necesarios para trabajar con los datos.
  2. Utilización de Servlets para la gestión de peticiones. Lectura de parámetros de entrada.
  3. Paso de los parámetros a los JavaBeans, ejecución de la lógica de negocios
  4. Almacenamiento de resultados en JavaBeans, Servlets, Sesiones o Aplicaciones.
  5. Remitir la solicitud a las páginas JSP determinadas, utilizando el método *RequestDispatcher* para transferir el control a la página.
  6. Extracción de datos de los JavaBeans, etc, por parte de la página JSP y su devolución.

© JAA2013

## Uso RequestDispatcher para implementar MVC

### NOTAS:

- Los JavaBeans sólo van a ser utilizados para el mantenimiento de las variables privadas y utilización de sus métodos *get / set*.
- Desde la página JSP sólo accederá a los JavaBeans; no crea ni modifica valores en ellos.
- Los Servlets no deben de mostrar ninguna salida al usuario final, sólo utilizarán las técnicas normales de leer la información de la solicitud y generar los datos.
- Los servlets no deben de utilizar las llamadas *response.setContentType*, *response.getWriter* o *out.println*.
- Los servlets deben de almacenar los datos devueltos por la lógica de la aplicación en cualquiera de los 3 elementos accesibles desde las páginas JSP (*HttpServletRequest*, *HttpSession*, o *ServletContext*)
- Al redirigir las peticiones a páginas JSP (*RequestDispatcher*) es conveniente que estas JSP estén el directorio WEB-INF (los clientes no se les permite acceder directamente a los archivos de WEB-INF, pero el servidor está autorizado a transferir el control allí.)

© JAA2013

## Uso RequestDispatcher para implementar MVC

Ejemplo de Redirección:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    String operation = request.getParameter("operation");
    if (operation == null) {
        operation = "unknown";
    }
    String address;

    if (operation.equals("order")) {
        address = "/WEB-INF/Order.jsp";
    }
    else if (operation.equals("cancel")) {
        address = "/WEB-INF/Cancel.jsp";
    } else {
        address = "/WEB-INF/UnknownOperation.jsp";
    }
    RequestDispatcher dispatcher = request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```

© JAA2013

## Redirecting en vez de Forwarding

- Por defecto en MVC estándar, se utiliza el método de RequestDispatcher para transferir el control desde el servlet de la página JSP.
- Sin embargo, cuando se utiliza este reenvío en el ámbito de sesión, a veces es preferible utilizar `response.sendRedirect`.
- Ventajas:
  - El control se transfiere completamente dentro del servidor, por lo que no hay tráfico de red.
  - El usuario no ve la dirección de la página de destino y las páginas JSP se puede colocar en WEB-INF para evitar que el usuario pueda acceder a ellas.
  - Es utilizado, habitualmente, cuando se vuelve a mostrar un formulario HTML incompletas o un resumen del contenido de un carrito de compras.

© JAA2013

## Ejemplo Completo

### JavaBeans

```
public class NumberBean {
private double num = 0;

public NumberBean(double number) {
setNumber(number);
}

public double getNumber() {
return(num);
}

public void setNumber(double number) {
num = number;
}

}
```

### RandomNumberServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/** Servlet that generates a random number, stores it in a bean, and
forwards to JSP page to display it.
*/
public class RandomNumberServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
    NumberBean bean = new NumberBean(Math.random());
    request.setAttribute("randomNum", bean);
    String address = "/WEB-INF/RandomNum.jsp";

RequestDispatcher dispatcher=
request.getRequestDispatcher(address);

dispatcher.forward(request, response);
}
}
```

© JAA 2013

## Ejemplo Completo

### RandomNum.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Random Number</TITLE>
<LINK REL=STYLESHEET HREF="/bank-support/JSP-Styles.css" TYPE="text/css">
</HEAD>

<BODY>
<jsp:useBean id="randomNum" type="package.NumberBean" scope="request" />

<H2>Random Number:

<jsp:getProperty name="randomNum" property="number" />

</H2>

</BODY>
</HTML>
```

© JAA 2013

## Forward en páginas JSP

- El enfoque habitual es:
  - Servlets la primera peticiones y redirecciones a páginas JSP para mostrar una salida.
- Del mismo modo, es muy posible que una página JSP pueda reenviar (forward) las solicitudes en otros lugares.
  - ✓ Por ejemplo, solicita a una página JSP y que reenvíe la solicitud en otro lugar sólo cuando se recibe valores inesperados.
- El envío de peticiones a *servlets* en vez de páginas JSP requiere ningún cambio en absoluto en el uso de la *RequestDispatcher*.
- Desde JSP podemos realizar tal acción mediante el tag *jsp:forward*, más simple y más fácil de usar que la método *RequestDispatcher*.

`<jsp:forward page="Relative URL" />`

© JAA 2013

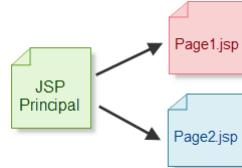
## Forward en páginas JSP

- **Ejemplo:**

El siguiente código envía a la mitad de los clientes a una página JSP y la otra mitad a otra página JSP.

PaginaJSP

```
:
<% String destination;
   if (Math.random() > 0.5) {
      destination = "/examples/page1.jsp";
   } else {
      destination = "/examples/page2.jsp";
   }
%>
<jsp:forward page="<% destination %>" />
:
```



© JAA 2013

## Include en vez de Forward

- El método *forward* se basa en llamar a páginas JSP para generar la salida completa.
- Para poder combinar múltiples salidas, podemos utilizar el método *include* para generar una salida combinada de múltiples páginas JSP.
- Este enfoque es utilizado con servlets para crear portales que permiten a los usuarios especificar cómo se visualizará la página dividida en piezas (gadget).

© JAA 2013

## Include en vez de Forward

### Ejemplo

```
String firstTable, secondTable, thirdTable;

if (someCondition) {
    firstTable = "/WEB-INF/Sports-Scores.jsp";
    secondTable = "/WEB-INF/Stock-Prices.jsp";
    thirdTable = "/WEB-INF/Weather.jsp";
} else if (...) {
{ ... }

RequestDispatcher dispatcher = request.getRequestDispatcher("/WEB-INF/Header.jsp");
dispatcher.include (request, response);

dispatcher = request.getRequestDispatcher(firstTable);
dispatcher.include (request, response);

dispatcher = request.getRequestDispatcher(secondTable);
dispatcher.include (request, response);

dispatcher = request.getRequestDispatcher(thirdTable);
dispatcher.include (request, response);

dispatcher = request.getRequestDispatcher("/WEB-INF/Footer.jsp");
dispatcher.include (request, response);
```

© JAA 2013

REpresentational State Transfer

## SERVICIOS RESTFUL

© JAA 2013

### Objetivos

- Desacoplar el cliente del backend
- Mayor escalabilidad
  - Sin estado en el backend.
- Separación de problemas
- División de especialidades
- API uniforme para todos los clientes
  - Disponer de una interfaz uniforme (basada en URLs)

© JAA 2013

## REST (REpresentational State Transfer)

- Un **estilo de arquitectura** para desarrollar aplicaciones web distribuidas que se basa en el uso del protocolo HTTP e Hypermedia.
- Definido en el 2000 por Roy Fielding, para no reinventar la rueda, se basa en aprovechar lo que ya estaba definido en el HTTP pero que no se utilizaba.
- El HTTP ya define 8 métodos (algunas veces referido como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado:
  - HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT
- El HTTP permite en el encabezado transmitir la información de comportamiento:
  - Accept, Content-type, Response (códigos de estado), Authorization, Cache-control, ...

© JAA 2013

## Uso de la cabecera

- Request: Método /uri?parámetros
  - GET: Recupera el recurso
    - ✓ Todos: Sin parámetros
    - ✓ Uno: Con parámetros
  - POST: Crea un nuevo recurso
  - PUT: Edita el recurso
  - DELETE: Elimina el recurso
- Accept: Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- Content-type: Indica en que formato está codificado el cuerpo, utilizando MIME
- Response: Código de estado con el que el servidor informa del resultado de la petición.

© JAA 2013

## Peticiones

```
Request: GET /users
          Response: 200
          content-type:application/json
Request: GET /users/11
          Response: 200
          content-type:application/json
Request: POST /users
          Response: 201
          content-type:application/json
Request: PUT /users/11
          Response: 200
          content-type:application/json
Request: DELETE /users/11
          Response: 204 no content
```

© JAA 2013

## Control de cache

- Etag: Podemos controlar si el recurso se ha modificado desde la última vez que accedimos con un hash.
- If-None-Match se encarga de indicar que la petición sea efectiva siempre y cuando el eTag sea distinto, If-Match hace lo inverso.
- Last-Modified/If-Modified-Since permiten saber si un recurso se ha modificado en base a una fecha.

© JAA 2013

## Richardson Maturity Model

- # Nivel 1 (Pobre): Se usan URIs para identificar recursos. Reglas:
  - Se debe identificar un recurso  
 /invoices/page/2 → /invoices/?page=2
  - Se construyen con nombres nunca con verbos  
 /getUser/{id} → /users/{id}/  
 /users/{id}/edit/login → users/{id}/access-token
  - Deberían tener una estructura jerárquica  
 /invoices/user/{id} → /user/{id}/invoices
- # Nivel 2 (Medio): Se usa el protocolo HTTP adecuadamente
- # Nivel 3 (Óptimo): Se implementa hypermedia.
- <http://www.crummy.com/writing/speaking/2008-QCon/act3.html>

© JAA 2013

## Hypermedia

- Se basa en la idea de enlazar recursos: propiedades que son enlaces a otros recursos.
- Para que sea útil, el cliente debe saber que en la respuesta hay contenido hypermedia.
- En content-type es clave para esto
  - Un tipo genérico no aporta nada:  
 Content-Type: text/xml
  - Se pueden crear tipos propios  
 Content-Type:application/servicio+xml

© JAA 2013

## JSON Hypertext Application Language

- RFC4627 <http://tools.ietf.org/html/draft-kelly-json-hal-00>
- Content-Type: application/hal+json

```
{
  "_Links": {
    "self": {"href": "/orders/523"},
    "warehouse": {"href": "/warehouse/56"},
    "invoice": {"href": "/invoices/873"}
  },
  "currency": "USD",
  "status": "shipped",
  "total": 10.20
}
```

© JAA 2013

## Basados en Servlet

```
@WebServlet("/ServicioREST")
public class ServicioREST extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
    protected void doPut(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
    protected void doDelete(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
}
```

© JAA 2013

## Jersey JAX-RS 2.0 API

- Anotaciones
  - @Path: indica la ruta relativa a añadir a la URI para acceder a una clase o método
  - @GET, @PUT, @POST, @DELETE, @HEAD hacen referencia al tipo de petición HTTP que satisface un método
  - @Produces especifica el tipo MIME que retorna (plain, html, json, xml, etc.) un método
  - @Consumes especifica el tipo MIME que requiere un método
  - Existen más anotaciones, éstas son sólo las esenciales

```

@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("/saludo");
public String saludar(){ return "Hola"; }

@GET
@Produces(MediaType.TEXT_XML)
@Path("/saludo");
public String saludar(){ return "<?xml version=\"1.0\"?><Saludo>Hola</Saludo>"; }

```

© JAA 2013

## Maven

- <https://search.maven.org>

```

<dependencies>
  <dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <version>2.5.1</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-client</artifactId>
    <version>2.5.1</version>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.jaxrs</groupId>
    <artifactId>jackson-jaxrs-json-provider</artifactId>
    <version>2.7.0</version>
  </dependency>
</dependencies>

```

© JAA 2013

## web.xml

```
<servlet>
  <servlet-name>JAX-RS Servlet</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>servidor.ws</param-value> ← Paquete con los ws
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>JAX-RS Servlet</servlet-name>
  <url-pattern>/ws/*</url-pattern> ← Ruta a interceptar
</servlet-mapping>
```

© JAA 2013

# AUTENTIFICACIÓN Y SEGURIDAD EN APLICACIONES WEB

© JAA 2013

## Tipos de Autentificación

- Delegadas en el cliente y el servidor
  - Basados en estándares de Internet
    - ✓ Autentificación Básica HTTP
    - ✓ Autentificación Digest HTTP
    - ✓ Autentificación de cliente HTTPS
  - Basados en la especificación de servlets
    - ✓ Autentificación basada en formularios
- Autentificación controlada por programa
  - Hay que basarla en conexiones seguras con HTTPS

© JAA 2013

## Autentificación HTTP Básica

- Parte del protocolo HTTP desde sus comienzos.
- Muy simple
- Poco seguro
- Cuando el navegador solicita un recurso (jsp, por ejemplo), el servidor solicita usuario y contraseña, que viajan encriptadas con codificación base 64 bits.
- Se puede reforzar combinándola con protocolo HTTPS, de forma que las peticiones HTTP viajen encriptadas.

© JAA 2013

## Autentificación HTTP digest

- Incorporada en la versión 1.1 de la especificación del protocolo HTTP.
- Algo más complicada internamente que la autentificación básica.
- La codificación va en base al timestamp generado por el servidor en el momento de la petición -> Más difícil de decodificar -> Más seguro.
- No está totalmente difundida ni aceptada por todos los navegadores y servidores.

© JAA 2013

## Autentificación HTTP de cliente

- Es el más seguro de los disponibles hoy en día.
- Se basa en mecanismos de Clave Pública (PKI)
- Requiere una clave generada por un organismo de certificación autorizado.
- Hay un proceso “complicado” de handshake que garantiza la conexión segura.

© JAA 2013

## Autentificación Basada en Formularios

- La única definida en la especificación de servlets.
- Implementada por el contenedor Web.
- No es segura por si sola, y es necesario combinarla con HTTPs (SSL – Secure Socket Layer).
- Permite controlar la apariencia de la pantalla de login.
- La pantalla de login es HTML estándar que sigue un convenio para los nombres de los campos de usuario y contraseña.

© JAA 2013

## Autentificación Por Programa

- Todo lo tenemos que programar
- La aplicación coteja la contraseña contra su propio sistema de control de usuarios
- Hay que basarla en protocolos seguros de comunicación como HTTPs (SSL) para evitar que nadie “escuche” la contraseña interpretando los paquetes HTTP.
- Mas tedioso de programar
- Ventajas: No dependemos de la autentificación del entorno -> Más portable.

© JAA 2013

## Autentificación básica

- Es necesario:
  - Configurar el server.xml de TOMCAT para que solicite autentificación basada en memoria (tomcat\_users.xml).
  - Crear usuarios y perfiles en tomcat\_users.xml
  - Configurar el WEB.xml para restringir el acceso a los recursos a un determinado(s) perfil(es) de usuario.
- El usuario se autentifica contra el servidor mediante la ventana gris
- En el server.xml:  

```
<Realm className="org.apache.catalina.realm.MemoryRealm" />
```
- En Tomcat\_users.xml:  

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="amazin"/>
  <user username="dflanvin" password="dflanvin" roles="amazin"/>
</tomcat-users>
```

© JAA 2013

## Autentificación básica

- En el web.xml:  

```
<!--Autentificación básica. Tener en cuenta que los roles y usuarios
con sus contraseñas están dados de alta en el servidor-->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>arqui-java</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>amazin</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Autentificación de Amazin</realm-name>
</login-config>
<security-role>
  <role-name>amazin</role-name>
</security-role>
<!--Aquí termina la sección de autentificación-->
```

© JAA 2013

## Autentificación Por Programa

1. Crear página Login.jsp
2. Crear el LoginAction en la capa de presentación
3. En el struts-config.xml
  1. Crear el Form bean dinámico LoginForm
  2. Definir forward Global a login.jsp ante un error de autentificación
  3. Redefinir los action mappings
4. Modifico los actions de la aplicación para que comprueben si hay usuarios en sesión. En caso de que no exista, levantamos un evento de error de autentificación.
  - Problema: Estamos repitiendo código en todos los actions y jsps de la aplicación: comprobación de que el usuario está en sesión
  - Soluciones:
    - Redefinir Struts extendiéndolo para que invoque un determinado método ANTES de ejecutar el action
    - Desde la versión 2.3 se servlets, incorporación un filtro HTTP.