Servlets
Conceptos avanzados

Alcances: **`Request, Session, Application.`** Atributos

Respuestas de los Servlets

Redireccionamiento del requerimiento: **`sendRedirect`()**

Delegación de requerimiento y respuesta: **`forward`() - `include`()**

Atributos y Alcances

¿Qué es un atributo y dónde pueden guardarse?

Un atributo es un objeto java guardado dentro de algunos de los siguientes objetos (conocidos como alcances) de la API de servlets: **`ServletContext, HttpSession`** o **`HttpServletRequest.`**

alcance aplicación	alcance sesión	alcance request
Los objetos ligados a este contexto pueden ser usados por todos los **servlets/JSP** de la aplicación y duran mientras la aplicación web esté ejecutando.	Los objetos ligados a la sesión de un usuario pueden ser usados por todos los **servlets/JSP** que accedan a esa sesión y permanecen ahí mientras dure la sesión.	Los objetos ligados al request pueden ser usados por todos los **servlets/JSP** que dispongan de ese request y permanecen en él, mientras dure el request. Los atributos NO son parámetros.

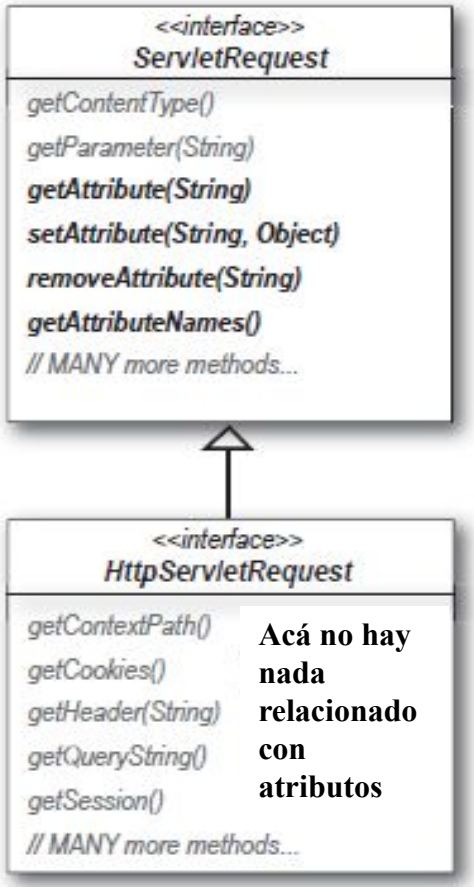
Se pueden usar los siguientes métodos para ligar, recuperar y eliminar atributos de cualquier alcance.

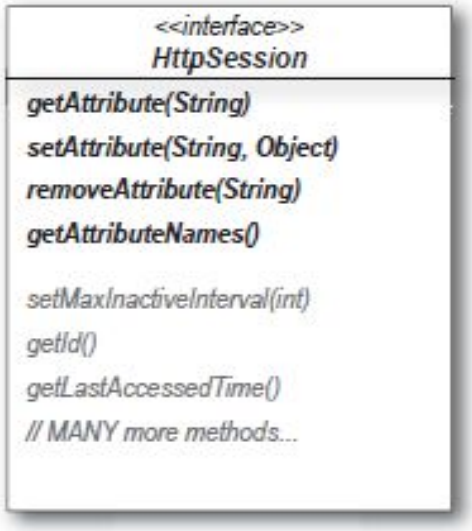
```
void setAttribute(String nombre, Object attr)
Object getAttribute(String nombre)
void removeAttribute(String nombre)
Enumeration getAttributeNames()
```

Notar que el tipo de dato de retorno del método getAttribute es Object y comúnmente se necesita usar casting para recuperar el tipo original.

Atributos y Alcances

Los atributos pueden ser ligados a uno de los siguientes alcances: **requerimiento**, **sesión** o **aplicación**. Los métodos de la API para ligar atributos son exactamente los mismos.





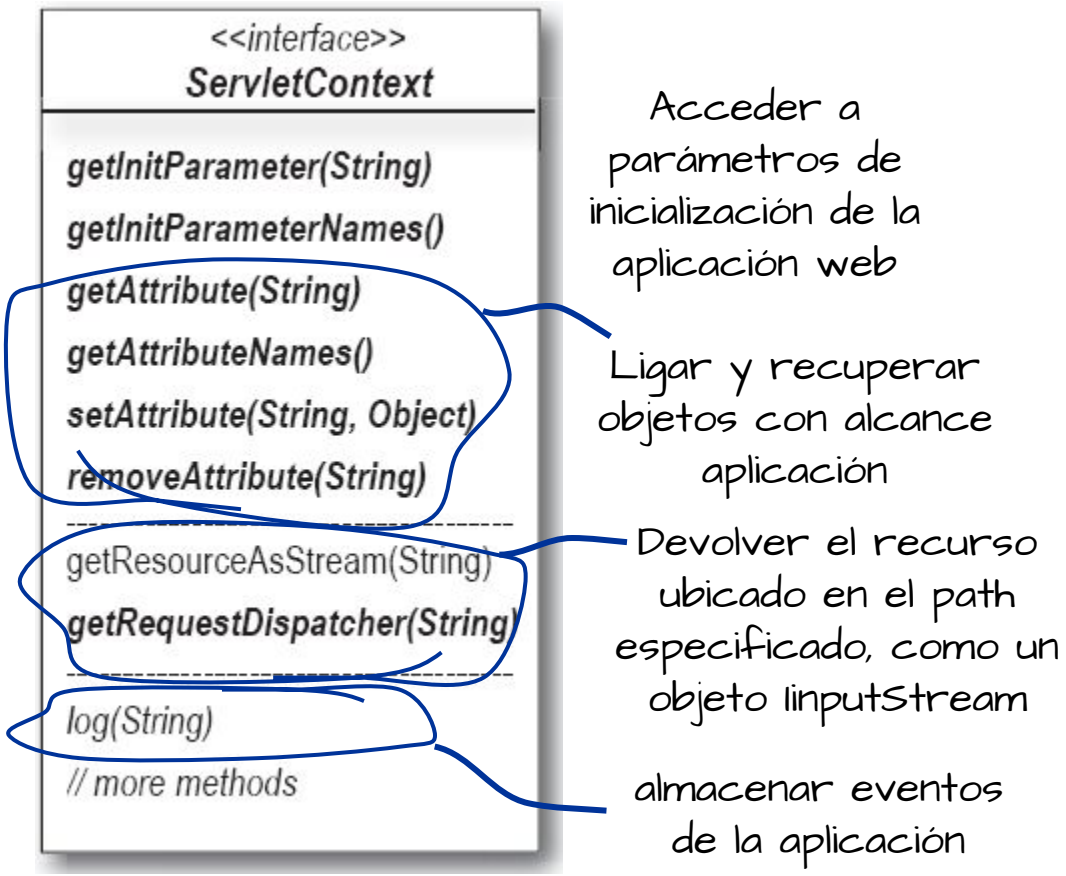
Veremos más adelante

Object getAttribute(String name)
void setAttribute(String name, Object value)
void removeAttribute(String name)
Enumeration getAttributeNames()

ServletContext

¿Qué es?

La interface **`ServletContext`** define una vista de la aplicación web para los **`servlets`**. A través del objeto **`ServletContext`** se pueden lograr varias funcionalidades observadas en el diagrama de la clase.



Cada Contenedor Web provee una implementación específica de la interfaz **`ServletContext`**.

Existe un único objeto **`ServletContext`** por aplicación web ejecutándose en el Contenedor Web.

Los servlets disponen del método **`getServletContext`**() que retorna una referencia al objeto **`ServletContext`**. El alcance del objeto **`ServletContext`** es toda la aplicación.

Para hacer logging de aplicaciones web se recomienda usar una API que permita desde cualquier clase acceder y usar el logging. Ejemplo: **Log4j**, http://jakarta.apache.org/log4j o la API estándar para logging del JDK (java.util.logging).

ServletContext

Parámetros de Inicialización de la Aplicación Web

De manera similar a los Servlets, las aplicaciones también pueden tener parámetros de inicialización

web.xml

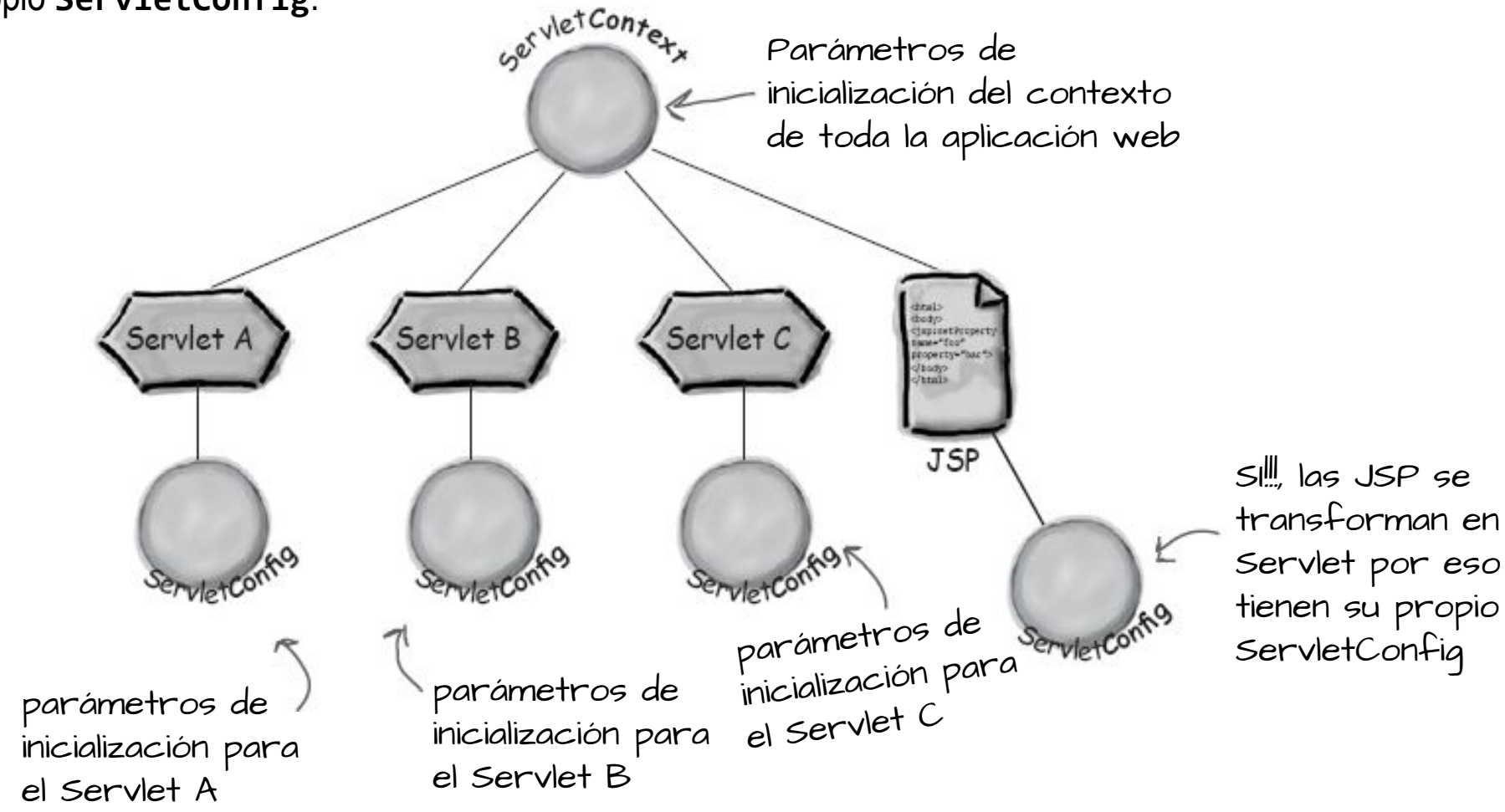
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns=http://java.sun.com/xml/ns/javaee  . . . id="WebApp_ID" version="2.5">
  <context-param>
     <param-name>email</param-name>
     <param-value>admin@info.unlp.edu.ar</param-value>
  </context-param>
. . .
</web-app>
```

Los parámetros de inicialización de la aplicación web pueden ser accedidos desde todas las componentes web.
Por ejemplo desde un Servlet

```
package com.servlets;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PaginaError extends HttpServlet {
 public void doGet(HttpServletRequest req,HttpServletResponse resp)..{
     resp.setContentType(“text/html”);
     PrintWriter out=resp.getWriter();
     out.println(“<html><head>”);
     out.println(“<title> Ocurrió un Error </title>”);
     out.println(“</head><body>”);
     ServletContext sc = this.getServletContext();
     String mail = sc.getInitParameter(“email”);
     out.println(“<h1> Error inesperado </h1>”);
     out.println(“Por favor, reportar a: ”+ mail);
     out.println(“</body></html>”);
  }
}
```

Parámetros de inicialización
de Servlets y del Contexto

Hay un único **`ServletContext`** para toda la aplicación web y todas las componentes lo comparten. El contenedor crea el objeto **`ServletContext`** cuando se carga la aplicación. Cada servlet en la aplicación tiene su propio **`ServletConfig`**.



Los parámetros de inicialización

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/j2ee/dts/web-app_2_3.dtd">
<web-app>

  <context-param>
    <param-name>email</param-name>
    <param-value>admin@info.unlp.edu.ar</param-value>
  </context-param>

  . . .


<servlet>
 <servlet-name>ServletFecha</servlet-name>
 <servlet-class>misServlets30.ServletFecha</servlet-class>
   <init-param>
     <param-name>dia</param-name>
     <param-value>Hoy es:</param-value>
   </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>ServletFecha</servlet-name>
    <url-pattern>/ServletFecha</url-pattern>
</servlet-mapping>
. . .
</web-app>
```

web.xml

del Contexto o de la Aplicación

```
getServletContext().getInitParameter("email")
```

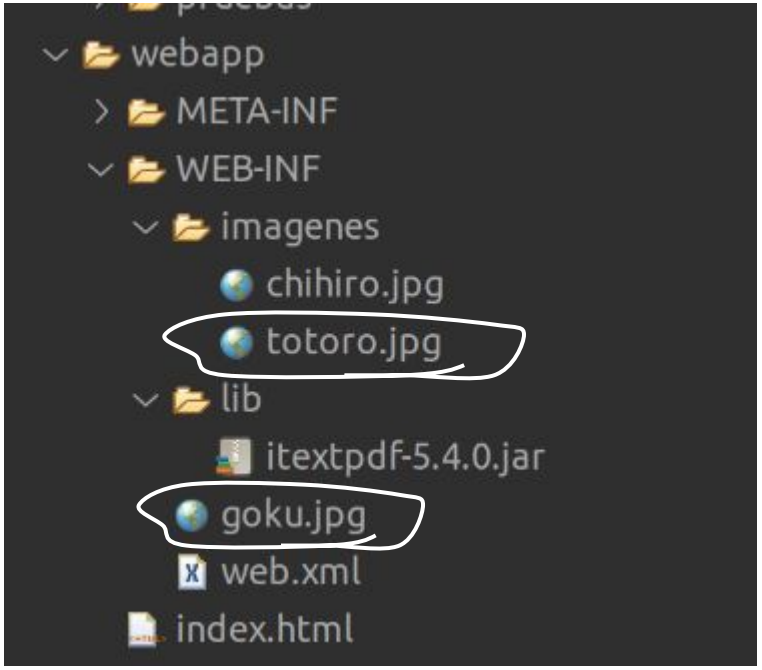
del Servlet ServletFecha

```
getServletConfig().getInitParameter("dia")
```

ServletContext

Recursos web estáticos

El objeto `ServletContext` también provee acceso a la jerarquía de documentos estáticos que forman parte de la aplicación Web, como por ejemplo: archivos `TXT`, `GIF`, `JPEG`.



webapp o context root de Eclipse se traduce al nombre de la aplicación en los contenedores Web. **Por ejemplo** `cupones`

Nombre de la aplicación web

`http://www.cinemacity.com/cupones`

Paths virtuales de las imágenes:

```
/WEB-INF/goku.jpg
/WEB-INF/imagenes/totoro.jpg
/index.html
```

Todos los recursos de una aplicación web son abstraídos en directorios virtuales a partir de la raíz de la aplicación web. Un directorio virtual comienza con "/" y continúa con un path virtual a directorios y recursos.

ServletContext
Recursos web estáticos

La forma correcta para leer recursos de una aplicación web es usando los métodos **`getResource()`** o **`getResourceAsStream()`**. Estos métodos garantizan que el servlet siempre tendrá acceso al recurso deseado. Aún cuando el *deploy* de la aplicación web se realice en múltiples servidores o en un archivo WAR.

El objeto **`ServletContext`** provee acceso a la jerarquía de documentos estáticos a través de los siguientes métodos:

- **`URL getResource(String path)`**

  Devuelve la URL al recurso que coincide con el **path** dado como parámetro. El **path** debe comenzar con "/" y es relativo a la raíz de la aplicación web.

```
URL unaUrl = getServletContext().getResource("/WEB-INF/goku.jpg");
```

- **`InputStream getResourceAsStream(String path)`**

  Devuelve el recurso ubicado en el **path** especificado como parámetro, como un objeto **InputStream**. Los datos en el **InputStream** pueden ser de cualquier tipo y longitud. El path debe empezar con "/" y es relativo a la raíz de la aplicación web.

```
InputStream is=this.getServletContext().getResourceAsStream("/WEB-INF/salas.txt")
```

ServletContext

Recursos web estáticos – Archivos de texto

Utilizando el método **getResourceAsStream()** podríamos leer un archivo de texto ubicado por ejemplo en el directorio **WEB-INF** de la aplicación, de la siguiente manera:

```
@WebServlet("/ServletLeeProperties")
public class ServletLeeProperties extends HttpServlet {

 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws . . .{

  String filename = "/WEB-INF/datos.properties";
  ServletContext context = this.getServletContext();

  // Obtenemos un InputStream usando ServletContext.getResourceAsStream()
  InputStream is = context.getResourceAsStream(filename);
  if (is != null) {
      InputStreamReader isr = new InputStreamReader(is);
      BufferedReader reader = new BufferedReader(isr);
      String text;
      // Leemos por linea y escribimos en la salida
      while ((text = reader.readLine()) != null) {
                // Acá se obtiene la  linea de texto en la variable text
               // Por ejemplo -> jdbc.driver="com.mysql.jdbc.Driver"    }
  }
 }
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws . . . {
           doGet(request, response);
 }
}
```

datos.properties

```
jdbc.driver= "com.mysql.jdbc.Driver"
jdbc.user= "admin"
jdbc.password = "ttps@admin2017"
jdbc.url="jdbc:mysql://localhost:3306/myBase"
```

ServletContext

Recursos web estáticos – Archivos de Propiedad

Un archivo de propiedades está formado por líneas de texto de la forma **`clave=valor`**. La extensión de estos archivos es **`properties`**.
La clase **`ResourceBundle`**, del paquete **`java.util`**, facilita la lectura de archivos de propiedades. Esta clase tiene un método **`static getBundle(String name)`** que obtiene un objeto **`ResourceBundle`** usando el nombre del archivo.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
                                                    ServletException, IOException {
     PrintWriter out = response.getWriter();
     ResourceBundle rb = ResourceBundle.getBundle("recursos.datos");
     Enumeration<String> claves = rb.getKeys();
     while (claves.hasMoreElements()) {
          String clave = (String) claves.nextElement();
          out.println(clave+" = "+rb.getString(clave));
     }
}          // manualmente -> out.println(rb.getString("jdbc.driver");
```

en este caso, el archivo datos.properties debe ubicarse en la carpeta de fuentes java y carpeta recursos

Estos archivos de texto son comúnmente usados para guardar datos de configuración de una aplicación. A continuación se muestra el contenido de un archivo llamado **`datos.properties`** y ubicado en **`src/recursos`**

`datos.properties`

```
jdbc.driver= "com.mysql.jdbc.Driver"
jdbc.user= "admin"
jdbc.password = "ttps@admin2017"
jdbc.url="jdbc:mysql://localhost:3306/myBase"
```

ServletContext

Listeners de Contexto

¿Cómo haríamos para correr algún código antes de que un Servlet -o JSP- pueda responder a un requerimiento? ¿Podemos saber cuando la aplicación web arranca?

Podemos crear una clase separada (ni Servlet, ni JSP, ni Filtro), que pueda escuchar por 2 eventos del ciclo de vida de la aplicación: **creación** y **destrucción** de la aplicación web. Para lograr esto, la clase debe implementar la interface **ServletContextListener** => Listeners de Contexto.

```
package misServlet;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class InicializaSalas implements ServletContextListener {

  public void contextDestroyed(ServletContextEvent sce) {  }

  public void contextInitialized(ServletContextEvent sce) {
      // Se leen parametros de inicializacion de la aplicación
      String peli1 = sce.getServletContext().getInitParameter("sala1");
      String peli2 = sce.getServletContext().getInitParameter("sala2");
      String peli3 = sce.getServletContext().getInitParameter("sala3");
      // Se guardan en el contexto, las peliculas en Cartelera
      ServletContext contexto = sce.getServletContext();
      contexto.setAttribute("sala1", peli1);
      contexto.setAttribute("sala2", peli2);
      contexto.setAttribute("sala3", peli3);
  }
}
```

Se ligan los atributos al contexto de la aplicación. Ahora cualquier componente puede recuperarlos usando por ejemplo:

`getServletContext().getAttribute("sala1")`

ServletContext

Listener de contexto – configuración en el WEB.XML

Los **servlet listeners** deben configurarse en el archivo **`web.xml`** y de esta manera el Contenedor Web se enterará de su existencia. Para publicarlos se utiliza el tag **`<listener>`**.

`web.xml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/j2ee/dts/web-app_2_3.dtd">
<web-app>
 <context-param>
    <param-name>sala1</param-name>
    <param-value>Parasitos</param-value>
  </context-param>
  <context-param>
    <param-name>sala2</param-name>
    <param-value>Joker</param-value>
 </context-param>
 <context-param>
    <param-name>sala3</param-name>
    <param-value>El irlandes</param-value>
 </context-param>
 <listener-class>mislisteners.InicializaSalas</listener-class>
  . . .

</web-app>
```

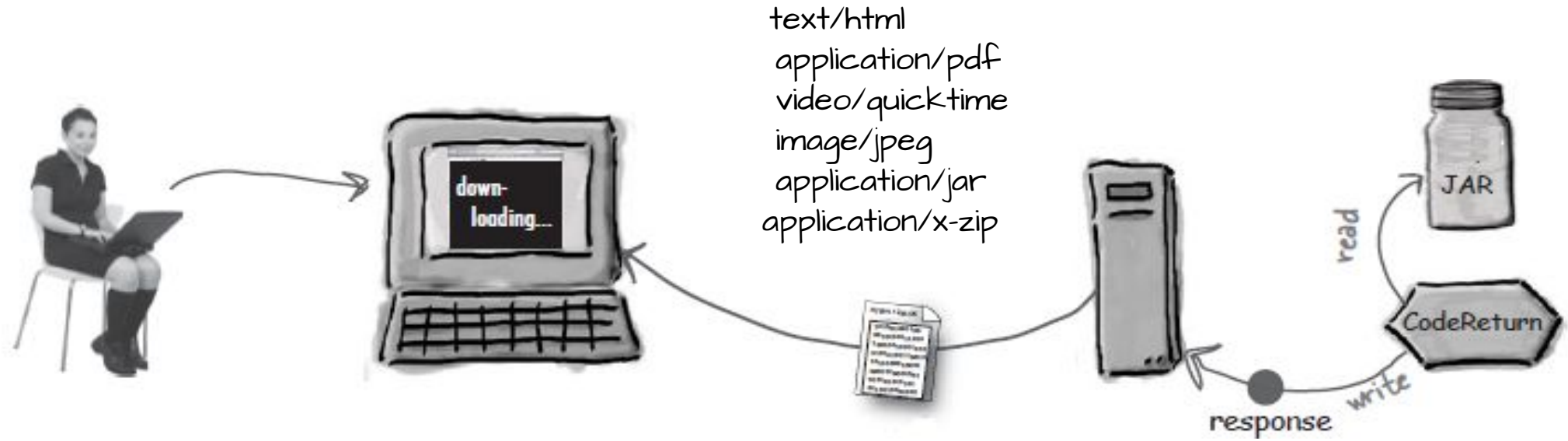
Es posible declarar múltiples listeners.
El orden en que son declarados determina el orden en el que son invocados por el Contenedor Web.

```
package misListeners;

@WebListener
public class InicializaSalas implements ServletContextListener {

  public void contextDestroyed(ServletContextEvent sce)  {
   . .
  }
  public void contextInitialized(ServletContextEvent sce)  {
   . . .
  }
}
```

Si usamos anotaciones evitamos el tag `<listener>` en el archivo web.xml

Servlets
Tipos de respuestas

Los tipos más comunes de MIME son:



Tenemos un JAR con código fuente java que queremos enviárselo a nuestros clientes usando un Servlet.

¿Qué tipo MIME usamos?

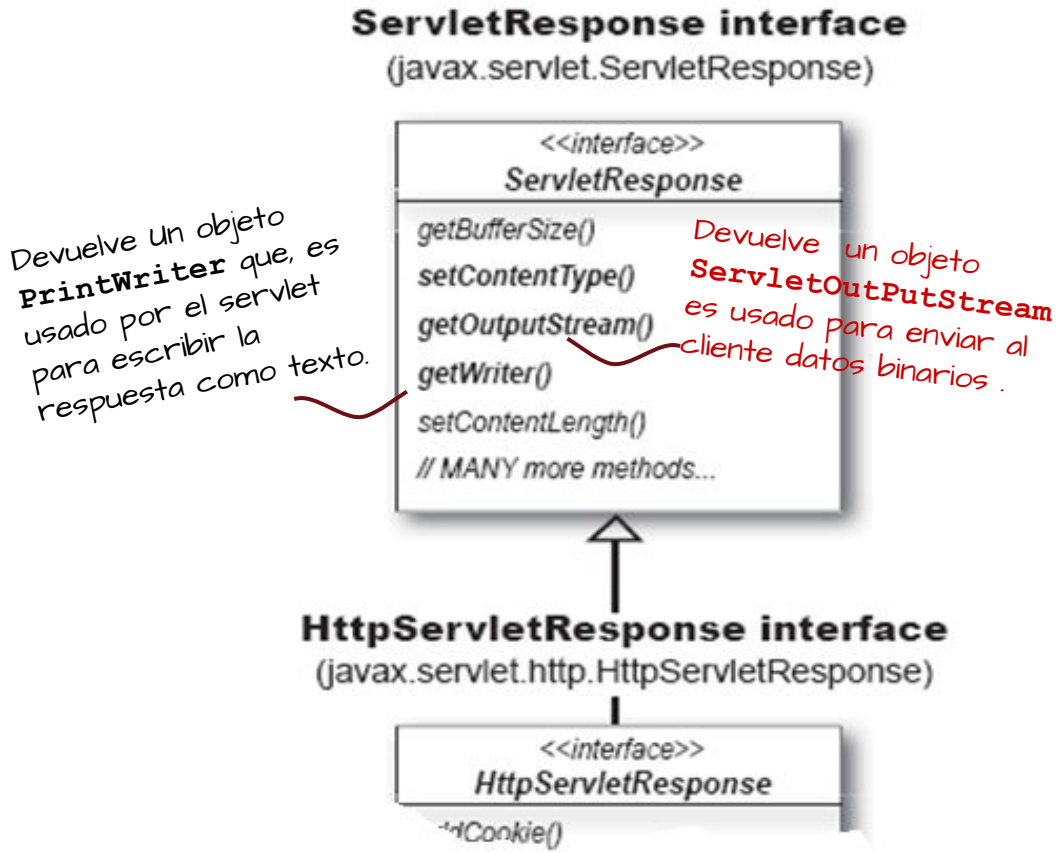
¿Qué objeto usamos para escribir la respuesta?

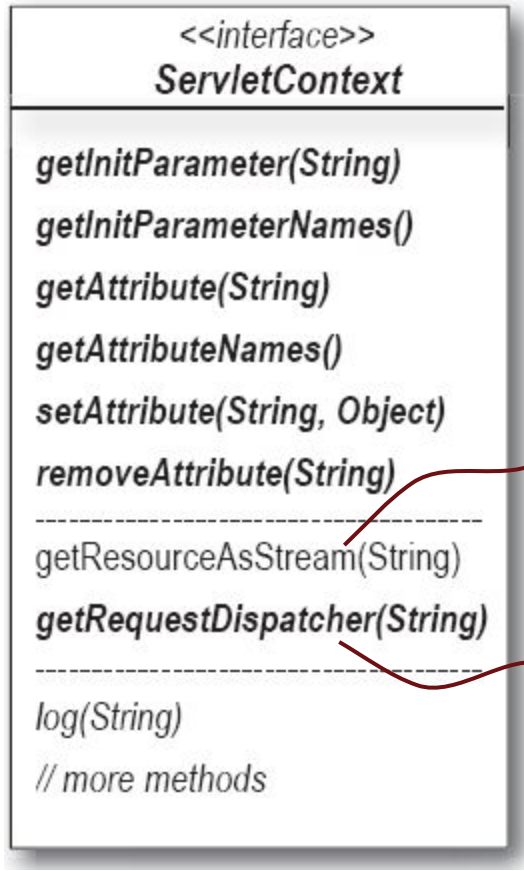
Las interfaces de programación

HttpServletResponse y ServletContext

Vimos que el objeto **HttpServletResponse** tiene métodos que retornan objetos donde se puede escribir el contenido que se enviará en la respuesta.

También existe la interface **ServletContext** que permite obtener referencias a URLs de recursos web.





Cada Contenedor Web provee una implementación específica de esta interface. La funcionalidad es idéntica, no depende de la implementación, está establecida en la interface.

Servlets

Tipos de respuestas – Retornando una imagen

Para enviar datos binarios desde un servlet se debe usar el objeto **`OutputStream`** retornado por el método **`getOutputStream()`**. En este caso el archivo está en memoria.

```
// imports
@WebServlet("/ServletImage")
public class ServletImage extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) thr

  OutputStream outputStream = response.getOutputStream();
  BufferedImage image = new BufferedImage(500, 300, BufferedImage.TYPE_INT_BGR);
  Graphics2D graphics = image.createGraphics();
  graphics.setBackground(Color.WHITE);
  graphics.clearRect(0, 0, 500,300);
  graphics.setFont(new Font("TimesRoman", Font.BOLD, 14));
  graphics.setColor(Color.DARK_GRAY);
  graphics.drawString("   Comienza tu día con una sonrisa y verás,", 30, 30);
  graphics.drawString("lo divertido que es andar por ahí desentonando", 30, 45);
  graphics.drawString("             con todo el mundo!!!", 30, 60);

  BufferedImage img = ImageIO.read(this.getServletContext().getResourceAsStream("/WEB-INF/mafalda.png"));
  graphics.drawImage(img, 40, 80, null, null);

  javax.imageio.ImageIO.write(image, "png", outputStream);
  outputStream.close();

} . . .

}
```

Transferir el control

`sendRedirect()` del objeto HttpServletResponse

Algunas veces el servlet puede redireccionar el requerimiento a otro recurso del mismo contenedor web o a una URL de otro dominio.

El `sendRedirect()` hace trabajar al navegador

¿Cómo lo hace?

El servlet invoca al método **`sendRedirect(String url)`** sobre la respuesta. La respuesta HTTP lleva el **código 302** que indica "El recurso que está buscando el cliente fue temporariamente movido". El navegador obtiene la respuesta, ve el código de estado "302" genera un nuevo requerimiento usando la URL que recibió como parámetro (el usuario puede observar en la barra del navegador que la URL cambia) y finalmente muestra una página al usuario que no fue la que el originalmente pidió.

```
public void doPost (HttpServletRequest request, HttpServletResponse response) {
   String person = request.getParameter("name");
   if(person==null) {
        response.sendRedirect("/app/DatosMal.html");
        return;
   }
     . . .
}
```

La url que se quiere que el navegador use para crear el requerimiento. Es la que verá el cliente

Se puede usar una url relativa o absoluta: "http://www..."

Transferir el control

`forward()` del objeto `ServletContext`

Usar el método **`forward()`** del objeto **`RequestDispatacher`** es otro mecanismo para transferir el control.

A diferencia del redireccionamiento de la respuesta, este mecanismo no requiere de ninguna acción por parte del cliente ni del envío de información extra entre el cliente y el servidor.

El **proceso íntegro de delegación del requerimiento se realiza del lado del servidor**. Además, este mecanismo permite pasar el requerimiento a otro servlet para que continúe el procesamiento y responda al cliente.

Instancia un RequestDispatcher para un servlet

```
public class ForwardServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response) {
    . . .
  RequestDispatcher dispatcher  = getServletContext().getRequestDispatcher("/mostrar");
  if (dispatcher != null) {
     request.setAttribute("hora",new Date());
     dispatcher.forward(request, response);
  }
  . . .
}}
```

Con el `forward()` trabaja más el servidor

Transferir el control

include() del objeto SevletContext

El objeto **RequestDispatcher** también cuenta con el método **include()** que se utiliza de manera similar que el **forward()** y que incluye contenido del lado del servidor en la respuesta que se está generando. El servlet que funciona como receptor del método **include()** – en el ejemplo los servlets **header** y **footer-** tienen acceso al requerimiento y a la respuesta original.

```
public class IncludeServlet extends HttpServlet {
 public void doGet(HttpServletRequest req, HttpServletResponse res) {
      . . .
 RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/header");
 if (dispatcher != null)

     dispatcher.include(request, response);

 . . .
 dispatcher = getServletContext().getRequestDispatcher("/footer");
 if (dispatcher != null)

     dispatcher.include(request, response);

}
. . .
}
```

Acá se podrían setear atributos en el request antes de delegar