Creación y ejecución de servlets

1. Escribiendo un servlet básico

Podemos encontrar introducciones al desarrollo de servlets en múltiples libros de Java, y abundantes manuales o tutoriales en Internet. Por ejemplo:

- http://flanagan.ugr.es/docencia/2005-2006/2/servlets/servlet.html
- http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html

La descripción completa de la API Servlet (clases, métodos…) se puede consultar en múltiples sitios de Internet (buscar simplemente "*javadoc servlet api*").

Los paquetes principales que contienen la API de servlets son:

- *jakarta.servlets*
- *jakarta.servlets.http*

Al llamar al servidor TOMCAT sin ningún contexto (*http://localhost:puerto* siendo puerto *'7000+X'*) aparece una pantalla de presentación con un enlace en la parte central de la pantalla (*examples*) hacia un conjunto de ejemplos de servlets, donde se puede ver tanto su ejecución como su código fuente. A veces, al código le falta alguna línea poco importante, para simplificar. Se recomienda ver el código fuente completo de cada uno en:

apache-tomcat/webapps/examples/WEB-INF/classes (apache-tomcat es el directorio base del TOMCAT)

El siguiente es el código de un servlet (*HelloWorld.java*) que se supone que recibe un parámetro llamado *user* (*user=nombre*) y que simplemente devuelve una página HTML con el texto "Hello *nombre*".

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

public class HelloWorld extends HttpServlet {

   public void doGet(HttpServletRequest req, HttpServletResponse res)
   {
     try {
        this.log("se imprime en tomcat/logs/localhost.fechadehoy.log");
        System.out.println("se imprime en tomcat/logs/catalina.out");

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

        String user = req.getParameter("user");

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello " + user + "World!</h1>");
        out.println("</body>");
        out.println("</html>");
     }
     catch (Exception ex) {System.out.println("Algo fue mal: "+ex.toString());}
   }
}
```

Como vemos, la clase que implementa el servlet (*HelloWorld*) extiende la clase básica *HttpServlet*, e implementa un único método (*doGet*) que será ejecutado cuando el servlet reciba una solicitud con el método GET.

El método *doGet* recibe dos parámetros:

- *req* (de tipo *HttpServletRequest*): a través del cual se accederá a toda la información de la solicitud.
- *res* (de tipo *HttpServletResponse*): a través del cual se configura y escribe la respuesta.

2. Cómo compilar y ejecutar servlets

La siguiente explicación supone que el TOMCAT ha sido instalado y configurado de acuerdo al documento "*TOMCAT.pdf*".

El primer paso es crear el directorio *WEB-INF/classes* dentro del directorio *public_html/webapps* de la cuenta del alumno. En ese directorio (y sus hijos) es donde deben residir las clases de los *servlets*. **Tanto el directorio WEB-INF como sus hijos no son accesibles para pedir un recurso directamente, como una página web.**

Para compilar un *servlet* se puede emplear la JVM instalada por defecto. Para ello, es necesario indicarle al compilador dónde residen las clases de la API *servlet*, que se encuentra en:

~sintX/apache-tomcat/lib/servlet-api.jar

Por ejemplo, para compilar el fichero "*HelloWorld.java*", emplearíamos:

javac -classpath ~sintX/apache-tomcat/lib/servlet-api.jar HelloWorld.java

Es importante compilar el servlet en un ordenador del laboratorio para asegurar que la JVM conoce la versión de los ficheros *class* (de lo contrario, si traemos al laboratorio ficheros *class* compilados en otro ordenador con una versión más reciente, la JVM del laboratorio no los reconocerá y dará un error).

Tras compilar un *servlet*, para instalarlo e invocar su ejecución se debe:

1. Colocar la clase en el directorio *WEB-INF/classes*, que creamos anteriormente.
2. Colocar en el directorio *WEB-INF* un fichero *web.xml* que describa los servlets de vuestro contexto. Habrá un único *servlet* por cada práctica. En la sección 3 se presenta un ejemplo.

3. Fichero web.xml

El fichero *web.xml* situado en el directorio *WEB-INF* describe los servlets que existen en un determinado contexto. Por ejemplo, dado el siguiente contenido (fichero disponible en MOOVI, no copiar de este PDF, ya que el resultado puede contener caracteres extraños que provoquen errores):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app version="2.4">
   <display-name>ServletSimple </display-name>
   <description> Servlet muy simple </description>
   <servlet>
      <servlet-name>MiServlet</servlet-name>
      <servlet-class> HelloWorld </servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>MiServlet</servlet-name>
      <url-pattern> /HP </url-pattern>
   </servlet-mapping>
</web-app>
```

En el elemento <*servlet*> se declara un único *servlet*, llamado *MiServlet*, implementado por la clase *HelloWorld.class* (que se supone está en el directorio *WEB-INF/classes*).

En el elemento <*servlet-mapping*> se declara que la llamada "*/HP"* será asociada al *servlet* llamado *MiServlet* (y, por tanto, según lo descrito en el párrafo anterior, provocará la ejecución de la clase *HelloWorld.class*).

La combinación de ambas cosas implica que la llamada *http://localhost:puerto/sintX/HP* (siendo *puerto* igual *7000+X*) debe provocar la ejecución del servlet implementado por la clase *HelloWorld.class*).

Después de modificar el fichero *web.xml* hay que reiniciar el TOMCAT para que aplique los cambios. Si el fichero es erróneo, es posible que el TOMCAT no arranque. Al arrancar o parar el TOMCAT, es bueno asegurarse de que la operación tiene éxito (con el comando "*ps auxw | grep tomcat*"). Si el TOMCAT no arrancase, el fichero *catalina.out* informa del posible error.

4. Depuración de un servlet

El mecanismo más sencillo para depurar un servlet es imprimir mensajes informativos con la traza de ejecución del mismo (con *this.log* o *System.out.println*, como se describe en el ejemplo de la sección 1).

También podemos usar opcionalmente el paquete **Log4j**, desarrollado por Apache para gestionar el registro de mensajes de una aplicación y actualmente el mecanismo más utilizado para esa tarea.

A continuación, se describe una implementación mínima y sencilla de esta API para almacenar mensajes en un fichero. El primer paso será descargar los paquetes de la API, e importar en el servlet los siguientes:

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

A continuación, en el servlet se declara e inicializa una variable que representa un proceso que escribe en el log:

```
Logger logger = LogManager.getLogger(MiServlet.class);     // suponiendo que MiServlet es mi servlet
```

A partir de ese momento, siempre que queramos podemos escribir mensajes en el fichero de *log* mediante la siguiente instrucción:

```
logger.info("Mensaje de depuración");
```

¿Cómo se indica en qué fichero se deben escribir los mensajes?

Para ello hay crear un fichero de configuración que será leído por el **Log4j** al inicializarse. Este fichero se llama *log4j2.xml* y se almacenará en el directorio *WEB-INF/classes*. Su contenido podría ser el siguiente (fichero disponible en MOOVI, no copiar de este PDF):

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Appenders>
    <File name="SINT"  fileName="sintX.log" append="true">
      <PatternLayout pattern="%d -&gt; %m%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Root level="debug">
      <AppenderRef ref="SINT"/>
    </Root>
  </Loggers>
</Configuration>
```

En este fichero se crea un *logger* llamado SINT, que añade mensajes al fichero *sintX.log*. Este fichero se creará en el directorio desde donde se lance el TOMCAT.

Lo descrito anteriormente es un uso mínimo del mecanismo proporcionado por **Log4j**, que puede gestionar mensajes de distinta jerarquía, volcados a distintos medios (consola, fichero, sockets…), etc.

Para compilar el servlet tras haberle añadido las instrucciones mencionadas, es necesario indicar en el *classpath* dónde están las clases utilizadas. Ahora, nuestro *classpath* debe ser (suponiendo que hemos descargado la versión 2.11.1):

```
-classpath  ~sintX/apache-tomcat/lib/servlet-api.jar:~sintX/ruta donde esté el Log4j/log4j-api-2.11.1.jar
```

Además, para que el TOMCAT ejecute correctamente las instrucciones mencionadas, es necesario proporcionarle los *jars* del *Log4j*. Concretamente, si nos limitamos a emplear lo descrito en esta sección, bastará con que copiemos al directorio *WEB-INF/lib* los siguientes *jars* (disponibles en MOOVI):

```
log4j-api-2.11.1.jar
log4j-core-2.11.1.jar
```

Dado que tenemos el fichero *log4j-api-2.11.1.jar* en el directorio *WEB-INF/lib*, esa puede ser la ruta que añadamos al *classpath* para compilar el servlet.

Creación y ejecución de servlets

1. Escribiendo un servlet básico

Podemos encontrar introducciones al desarrollo de servlets en múltiples libros de Java, y abundantes manuales o tutoriales en Internet. Por ejemplo:

- http://flanagan.ugr.es/docencia/2005-2006/2/servlets/servlet.html
- http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html

La descripción completa de la API Servlet (clases, métodos…) se puede consultar en múltiples sitios de Internet (buscar simplemente "*javadoc servlet api*").

Los paquetes principales que contienen la API de servlets son:

- *jakarta.servlets*
- *jakarta.servlets.http*

Al llamar al servidor TOMCAT sin ningún contexto (*http://localhost:puerto* siendo puerto *'7000+X'*) aparece una pantalla de presentación con un enlace en la parte central de la pantalla (*examples*) hacia un conjunto de ejemplos de servlets, donde se puede ver tanto su ejecución como su código fuente. A veces, al código le falta alguna línea poco importante, para simplificar. Se recomienda ver el código fuente completo de cada uno en:

apache-tomcat/webapps/examples/WEB-INF/classes (apache-tomcat es el directorio base del TOMCAT)

El siguiente es el código de un servlet (*HelloWorld.java*) que se supone que recibe un parámetro llamado *user* (*user=nombre*) y que simplemente devuelve una página HTML con el texto "Hello *nombre*".

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

public class HelloWorld extends HttpServlet {

   public void doGet(HttpServletRequest req, HttpServletResponse res)
   {
     try {
        this.log("se imprime en tomcat/logs/localhost.fechadehoy.log");
        System.out.println("se imprime en tomcat/logs/catalina.out");

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

        String user = req.getParameter("user");

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello " + user + "World!</h1>");
        out.println("</body>");
        out.println("</html>");
     }
     catch (Exception ex) {System.out.println("Algo fue mal: "+ex.toString());}
   }
}
```

Como vemos, la clase que implementa el servlet (*HelloWorld*) extiende la clase básica *HttpServlet*, e implementa un único método (*doGet*) que será ejecutado cuando el servlet reciba una solicitud con el método GET.

El método *doGet* recibe dos parámetros:

- *req* (de tipo *HttpServletRequest*): a través del cual se accederá a toda la información de la solicitud.
- *res* (de tipo *HttpServletResponse*): a través del cual se configura y escribe la respuesta.

Creación y ejecución de servlets

1. Escribiendo un servlet básico

Podemos encontrar introducciones al desarrollo de servlets en múltiples libros de Java, y abundantes manuales o tutoriales en Internet. Por ejemplo:

- http://flanagan.ugr.es/docencia/2005-2006/2/servlets/servlet.html
- http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html

La descripción completa de la API Servlet (clases, métodos…) se puede consultar en múltiples sitios de Internet (buscar simplemente "*javadoc servlet api*").

Los paquetes principales que contienen la API de servlets son:

- *jakarta.servlets*
- *jakarta.servlets.http*

Al llamar al servidor TOMCAT sin ningún contexto (*http://localhost:puerto* siendo puerto *'7000+X'*) aparece una pantalla de presentación con un enlace en la parte central de la pantalla (*examples*) hacia un conjunto de ejemplos de servlets, donde se puede ver tanto su ejecución como su código fuente. A veces, al código le falta alguna línea poco importante, para simplificar. Se recomienda ver el código fuente completo de cada uno en:

apache-tomcat/webapps/examples/WEB-INF/classes (apache-tomcat es el directorio base del TOMCAT)

El siguiente es el código de un servlet (*HelloWorld.java*) que se supone que recibe un parámetro llamado *user* (*user=nombre*) y que simplemente devuelve una página HTML con el texto "Hello *nombre*".

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

public class HelloWorld extends HttpServlet {

   public void doGet(HttpServletRequest req, HttpServletResponse res)
   {
     try {
        this.log("se imprime en tomcat/logs/localhost.fechadehoy.log");
        System.out.println("se imprime en tomcat/logs/catalina.out");

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

        String user = req.getParameter("user");

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello " + user + "World!</h1>");
        out.println("</body>");
        out.println("</html>");
     }
     catch (Exception ex) {System.out.println("Algo fue mal: "+ex.toString());}
   }
}
```

Como vemos, la clase que implementa el servlet (*HelloWorld*) extiende la clase básica *HttpServlet*, e implementa un único método (*doGet*) que será ejecutado cuando el servlet reciba una solicitud con el método GET.

El método *doGet* recibe dos parámetros:

- *req* (de tipo *HttpServletRequest*): a través del cual se accederá a toda la información de la solicitud.
- *res* (de tipo *HttpServletResponse*): a través del cual se configura y escribe la respuesta.

Creación y ejecución de servlets

1. Escribiendo un servlet básico

Podemos encontrar introducciones al desarrollo de servlets en múltiples libros de Java, y abundantes manuales o tutoriales en Internet. Por ejemplo:

- http://flanagan.ugr.es/docencia/2005-2006/2/servlets/servlet.html
- http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html

La descripción completa de la API Servlet (clases, métodos…) se puede consultar en múltiples sitios de Internet (buscar simplemente "*javadoc servlet api*").

Los paquetes principales que contienen la API de servlets son:

- *jakarta.servlets*
- *jakarta.servlets.http*

Al llamar al servidor TOMCAT sin ningún contexto (*http://localhost:puerto* siendo puerto *'7000+X'*) aparece una pantalla de presentación con un enlace en la parte central de la pantalla (*examples*) hacia un conjunto de ejemplos de servlets, donde se puede ver tanto su ejecución como su código fuente. A veces, al código le falta alguna línea poco importante, para simplificar. Se recomienda ver el código fuente completo de cada uno en:

apache-tomcat/webapps/examples/WEB-INF/classes (apache-tomcat es el directorio base del TOMCAT)

El siguiente es el código de un servlet (*HelloWorld.java*) que se supone que recibe un parámetro llamado *user* (*user=nombre*) y que simplemente devuelve una página HTML con el texto "Hello *nombre*".

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

public class HelloWorld extends HttpServlet {

   public void doGet(HttpServletRequest req, HttpServletResponse res)
   {
     try {
        this.log("se imprime en tomcat/logs/localhost.fechadehoy.log");
        System.out.println("se imprime en tomcat/logs/catalina.out");

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

        String user = req.getParameter("user");

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello " + user + "World!</h1>");
        out.println("</body>");
        out.println("</html>");
     }
     catch (Exception ex) {System.out.println("Algo fue mal: "+ex.toString());}
   }
}
```

Como vemos, la clase que implementa el servlet (*HelloWorld*) extiende la clase básica *HttpServlet*, e implementa un único método (*doGet*) que será ejecutado cuando el servlet reciba una solicitud con el método GET.

El método *doGet* recibe dos parámetros:

- *req* (de tipo *HttpServletRequest*): a través del cual se accederá a toda la información de la solicitud.
- *res* (de tipo *HttpServletResponse*): a través del cual se configura y escribe la respuesta.