Métodos de ServletRequest	Comentarios
public abstract int **getContentLength()**	Devuelve el tamaño de la petición del cliente o -1 si es desconocido.
public abstract String **getContentType()**	Devuelve el tipo de contenido **MIME** de la petición o **null** si éste es desconocido.
public abstract String **getProtocol()**	Devuelve el protocolo y la versión de la petición como un ***String*** en la forma <protocolo>/<versión mayor>.<versión menor>
public abstract String **getScheme()**	Devuelve el tipo de esquema de la **URL** de la petición: http, https, ftp...
public abstract String **getServerName()**	Devuelve el nombre del host del servidor que recibió la petición..
public abstract int **getServerPort()**	Devuelve el número del puerto en el que fue recibida la petición.
public abstract String **getRemoteAddr()**	Devuelve la dirección IP del ordenador que realizó la petición.
public abstract String **getRemoteHost()**	Devuelve el nombre completo del ordenador que realizó la petición.
public abstract ServletInputStream **getInputStream()** throws IOException	Devuelve un ***InputStream*** para leer los datos binarios que vienen dentro del cuerpo de la petición.
public abstract String **getParameter(String)**	Devuelve un ***String*** que contiene el valor del parámetro especificado, o null si dicho parámetro no existe. Sólo debe emplearse cuando se está seguro de que el parámetro tiene un único valor.
public abstract String[] **getParameterValues(String)**	Devuelve los valores del parámetro especificado en forma de un array de ***Strings***, o null si el parámetro no existe. Útil cuando un parámetro puede tener más de un valor.
public abstract Enumeration **getParameterNames()**	Devuelve una enumeración en forma de ***String*** de los parámetros encapsulados en la petición. No devuelve nada si el ***InputStream*** está vacío.
public abstract BufferedReader **getReader()** throws IOException	Devuelve un ***BufferedReader*** que permite leer el texto contenido en el cuerpo de la petición.
public abstract String **getCharacterEncoding()**	Devuelve el tipo de codificación de los caracteres empleados en la petición.

Tabla 1. Métodos de la interface ***ServletRequest***.

6.1.2 El método service() en la clase GenericServlet

Este método es el núcleo fundamental del ***servlet.*** Recuérdese que es ***abstract*** en ***GenencServlet()*** por lo que si el ***servlet*** deriva de esta clase deberá ser definido por el programador. Cada petición por parte del cliente se traduce en una llamada al método ***service()*** del ***servlet.*** El método ***service()*** lee la petición y debe producir una respuesta en base a los dos argumentos que recibe:

- Un objeto de la interface ***ServletRequest*** con datos enviados por el cliente. Estos incluyen pares de parámetros clave/valor y un ***InputStream.*** Hay diversos métodos que proporcionan información acerca del cliente y de la petición efectuada, entre otros los mostrados en la Tabla 1.
- Un objeto de la interface ***ServietResponse,*** que encapsula la respuesta del ***servlet*** al cliente. En el proceso de preparación de la respuesta, es necesario llamar al método ***setContentType()*** para establecer el tipo de contenido **MIME** de la respuesta. La Tabla 2 indica los métodos de la interfase ***ServietResponse.***

Puede observarse en la Tabla 1 que hay dos formas de recibir la información de un forrnulario HTML en un ***servlet.*** La primera de ellas consiste en obtener los valores de los parámetros (mé todos ***getParameterNames()*** y ***getParameterValues()*** y la segunda en recibir la información mediante un ***InutStream*** o un ***Reader*** y hacer por uno mismo su decodificación.

El cometido del método ***service()*** es simple: genera una respuesta por cada petición recibida de un cliente. Es importante tener en cuenta que puede haber múltiples respuestas que están siendo procesadas al mismo tiempo, pues los ***servlets*** son ***multithread.*** Esto hace que haya que ser especialmente cuidadoso con los ***threads,*** para evitar por ejemplo que haya dos objetos de un ***servlet*** escribiendo simultáneamente en un mismo campo de una base de datos.

Es aconsejable que el programador derive las clases de sus servlets de ***HttpServlet).*** El motivo es simple: la clase ***HttpServlet*** define ***service()*** de una forma muy adecuada, llamando a otros métodos ***(doPot(). doGetO.*** etc.) que son los que tiene que redefinir el programador.

Métodos de ServletResponse	Comentarios
ServletOutput Stream getOutputStream()	Permite obtener un ServletOutputStream para enviar datos binarios
PrintWriter getWriter()	Permite obtener un PrintWriter para enviar caracteres
setContentType(String)	Establece el tipo MIME de la salida
setContentLength(int)	Establece el tamaño de la respuesta

Tabla 2. Métodos de la interface ***ServletResponse***.

6.1.3 El método destroy() en la clase GenericServlet: forma de terminar ordenadamente

Una buena implementación de este método debe permitir que el ***servlet*** concluya sus tareas de forma ordenada. De esta forma, es posible liberar recursos (ficheros abiertos, conexiones con bases de datos, etc.) de una forma limpia y segura. Cuando esto no es necesario, no redefinir el método ***destroyO.***

Puede suceder que al llamar al método ***destroy()*** haya peticiones de servicio que estén todavía siendo ejecutadas por el método ***service(),*** lo que podría provocar un fallo general del sistema. Por este motivo, es conveniente escribir el método ***destroy()*** de forma que se retrase la liberación de recursos hasta que no hayan satisfecho todas las llamadas al método ***service().*** A continuación se presenta una forma de lograr una correcta descarga del ***servlet:***

- Primero, es preciso saber si existe alguna llamada al método ***service()*** pendiente de ejecución, para lo cual se debe llevar un ***contador*** con las llamadas activas a dicho método.
- Segundo, en general es poco recomendable redefinir ***service()*** (en caso de tratarse de un ***service()*** que derive de ***HttpServlet).*** Sin embargo, en este caso sí resulta conveniente su redefinición, para poder saber cuándo ha sido llamado. El método redefinido deberá llamar al método ***service()*** de su super-clase para mantener íntegra la funcionalidad del ***servlet.***
- Tercero, los métodos de actualización del ***contador*** deben estar ***sincronizados,*** para evitar que dicho valor sea accedido simultáneamente por dos o más ***threads,*** *(ya hemos visto esto).*
- *Finalmente,* no basta con que el ***servlet*** espere a que todos los métodos ***service()*** hayan terminado. Es preciso indicarle a dicho método que el servidor se dispone a apagarse. De otra forma, el ***servlet*** podría quedar esperando indefinidamente a que los métodos ***service()*** acabaran. Esto puede lograrse utilizando una variable ***boolean*** que establezca esta condición.

Todas las consideraciones anteriores se han contemplado en el siguiente código:

```
public class ServletSeguro extends HttpServlet {

    private int contador=O;
    private boolean apagandose=false;

    protected synchronized void entrandoEnService() {contador++;}

    protected synchronized void saliendoDeService() {contador--;}

    protected synchronized int nurneroDeServicios() {return contador;}

    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
                                ServietException, IOException {

        entrandoEnService();
```

```
        try {
            super.service(req, resp);
        }finally {saliendoDeService();}

    } // service()

    protected void setApagandose(boolean flag){apagandose=flag;}

    protected boolean estaApagandose() {return apagandose;}

    public void destroy(){

        //  Comprobar que hay servicios en ejecución y en caso afirmativo
        //  ordernarles que paren la ejecución
        if (numeroDeServicios ()>o)
            setApagandose (true>;

        //  Mientras haya servicios en ejecución, esperar
        while(numServices()>O) {
            try
                Thread.sleep(intervalo);
            { catch<InterruptedException e){;} // fin del catch
        } // while
    }  //destroy()

    ...
    //  Servicio
    public void doPost(...){

        //  Comprobación de que el servidor no se está apagando
        for  <i=O; ((i<numeroDeCosasAHacer)&& !estaApagandose());i++)
            try{
              ...
                // codificación ...
            } match(Exception e){;}
        } //  for
    } // doPost()
} // ServletSeguro
```

6.2 EL CONTEXTO DEL SERVLET (SERVLET CONTEXT)

Un ***servlet*** vive y muere dentro de los límites del proceso del servidor. Por este motivo, puede ser interesante en un determinado momento obtener información acerca del entorno en el que se está ejecutando el ***servlet.*** Esta información incluye la disponible en el momento de inicialización del ***servlet,*** la referente al propio servidor o la información contextual específica que puede contener cada petición de servicio.

6.2.1 Información durante la inicialización del servlet

Esta información es suministrada al ***servlet*** mediante el argumento ***ServletConfig*** del método init(). Cada ***servidor HTTP*** tiene su propia forma de pasar información al ***servlet***. Para acceder a dicha información:

```
String valorParametro;
public void init(servletContig config){
valorParametro = config. getInitParmeter (nombreparametro);
```

Como puede observarse, se ha empleado el método ***getInitParameter()*** de la interface ***Servletconfig*** (implementada por ***GenericServlet)*** para obtener el valor del parámetro. Asimismo puede obtenerse una ***enumeración*** de todos los nombres de parámetros mediante el métodc ***getInitParameterNames()*** de la misma interfase.

6.2.2 Información contextual acerca del servidor

La información acerca del servidor está disponible en todo momento a través de un objeto de la interfase ***ServletContext.*** Un ***servlet*** puede obtener dicho objeto mediante el método ***getServletContex()*** aplicable a un objeto ***ServletConfig.***

La interface ***ServletContext*** define los métodos descritos en la Tabla 3:

Métodos de ServletContext	Comentarios
public abstract Object **getAttribute(String)**	Devuelve información acerca de determinados atributos del tipo clave/valor del servidor. Es propio de cada servidor.
public abstract Enumeration **getAttributeNames()**	Devuelve una enumeración con los nombre de atributos disponibles en el servidor. **Sólo disponible en la versión *2.1***
public abstract String **getMimeType(String)**	Devuelve el tipo **MIME** de un determinado fichero.
public abstract String **getRealPath(String)**	Traduce una ruta de acceso virtual a la ruta relativa al lugar donde se encuentra el directorio raíz de páginas ***HTML***
public abstract String **getServerInfo()**	Devuelve el nombre y la versión del servicio de red en el que está siendo ejecutado el ***servlet***.
public abstract Servlet **getServlet(String)** throws ServletException	Devuelve un objeto ***servlet*** con el nombre dado. **Deprecado en la versión *2.1***, pues es un potencial foco de errores.
public abstract Enumeration **getServletNames()**	Devuelve un enumeración con los ***servlets*** disponibles en el servidor. **Deprecado en la versión *2.1***.
public abstract void **log(String)**	Escribe información en un fichero de *log*. El nombre del mismo y su formato son propios de cada servidor.

Tabla 3. Métodos de la interface ***ServletContext***.

6.3 TRATAMIENTO DE EXCEPCIONES

El ***Servlet API*** incluye dos clases de ***excepciones:***

1. La excepción ***javax.servlet.ServletException*** puede ser empleada cuando ocurre un fallo general en el ***servlet.*** Esto hace saber al servidor que hay un problema.

2. La excepción ***javax.servlet.UnavailableException*** indica que un ***servlet*** no se encuentra disponible. Los ***servlets*** pueden notificar esta excepción en cualquier momento.

6.4 CLASE HTTPSERVLET: SOPORTE ESPECÍFICO PARA EL PROTOCOLO HTTP

Los ***servlets*** que utilizan el protocolo ***HTTP*** son los más comunes. Por este motivo, ***Sun*** ha incluido un package específico para estos ***servlets*** en su ***JSDK: javax.servlet.http.*** Antes de estudiar dicho ***package*** en profundidad, hagamos una pequeña referencia al protocolo ***HTTP.***

HTTP son las siglas de ***HyperText Transfer Protocol,*** que es un protocolo mediante el cual los browser y los servidores puedan comunicarse entre sí, mediante la utilización de una serie de ***métodos: GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT*** y ***OPTIONS.*** Para la mayoría de las aplicaciones, bastará con conocer los tres primeros.

6.4.1 Método GET: codificación de URLs

El método ***HTTP GET*** solicita ***información*** a un ***servidor web.*** Esta información puede ser un fichero, el resultado de un programa ejecutado en el servidor (como un ***servlet,*** un ***programa CGi*** ...), etc.