UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA

“Servlets”

Integrantes

Paola Rojas Contreras
Juan Villarroel Leiva
José Pérez Villalobos

Profesor
Agustín González V.

Fecha
22 de Noviembre de 2004

Resumen: Descripción de la Tecnología Java Servlet

Los **servlets** son objetos que corren dentro del contexto de un servidor de aplicaciones (ej: Tomcat) y extienden su funcionalidad. La palabra *servlet* deriva de otra anterior, *applet*, que se refería a pequeños programas escritos en Java que se ejecutan en el contexto de un navegador web. Por contraposición, un *servlet* es un programa que se ejecuta en un servidor web.

El uso más común de los *servlets* es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

Los *servlets* forman parte de J2EE (Java 2 Enterprise Edition), que es una ampliación de J2SE (Java 2 Standard Edition).

Un *servlet* es un objeto Java que implementa la interfaz javax.servlet.Servlet ó hereda alguna de las clases más convenientes para un protocolo específico (ej: javax.servlet.HttpServlet). Entre el servidor de aplicaciones (ó contenedor web) y el *servlet* existe un contrato que determina cómo han de interactuar. La especificación de éste se encuentra en los JSR (Java Specification Requests) del JCP (Java Community Process).

Introducción

La manera más tradicional de invocar programas desde un servidor web es el mecanismo denominado *Common Gateway Interface* (CGI), que surgió desde los inicios del desarrollo del HTML. Desde el browser del cliente, y en forma de un formulario HTML, se envían datos hacia el programa CGI en el servidor. La URL ingresada determina qué programa CGI se ejecutará (puede ser un script o un archivo ejecutable). Los datos son procesados y se genera una respuesta, normalmente una página HTML. Esta respuesta es devuelta al servidor web, que empaqueta esta página web y la envía en forma de una respuesta HTTP, la que es finalmente recibida en el browser del cliente.

Sun Microsystems desarrolló los servlets como un avance respecto a la tecnología tradicional de CGI . Un servlet Java es un programa que, al igual que su similar en CGI, corre en un servidor web. El tipo de tareas que uno y otro pueden realizar son similares. Los servlets también procesan peticiones HTTP, ejecutan acciones (consultas a bases de datos, o conexiones a un host remoto, etc.) y generan respuestas en forma dinámica, por ejemplo una página web. No obstante, las arquitecturas de ejecución de una y otra tecnología son distintas.

A diferencia de los programas CGI, los servlets se ejecutan dentro de un contenedor web. Éste es en la práctica una máquina virtual Java que entrega una implementación de la API de Java para servlets (paquete javax.servlet). De esta manera, un servlet es una instancia de un HTTPServlet (podrían emplearse también los denominados GenericServlet), administrada por el contenedor web para responder ante peticiones HTTP, mediante los métodos doPost() o doGet().

El objetivo de este trabajo es dar un marco teórico para conocer, comprender y utilizar los servlets. Además se entregan dos aplicaciones para mostrar los aspectos prácticos del tema.

LA API Java Servlet

Un servlet es una componente de Java que se puede incluir en un servidor web para proporcionar distintos servicios de encargo. Estos servicios pueden incluir:

- Nuevas características.
- Actualizaciones a los contenidos de páginas.
- Actualizaciones a la presentación de páginas.
- Interacción con protocolos estándares (tales como ftp).
- Interacción con protocolos nuevos.

Los servlets han sido diseñados para trabajar dentro de un modelo de requerimientos y respuestas. En este tipo de modelo, un cliente envía una petición a un servidor y el servidor responde enviando de vuelta un mensaje. Las peticiones pueden venir en la forma de HTTP URL, FTP URL, o cualquier otro tipo de protocolo.

La petición y la respuesta correspondiente reflejan el estado del cliente y del servidor en el momento de la petición. Normalmente, el estado de la conexión cliente/servidor no se puede mantener a través de diversos pares de requerimientos/respuestas. Sin embargo, la información de la sesión puede ser mantenida con los servlets, lo que será descrito más adelante.

La API Java Servlet incluye varias interfaces de Java y define completamente el acoplamiento entre un servidor web y los servlets. La API Servlet se define como una extensión al JDK estándar. Las extensiones del JDK se empaquetan bajo `javax` -- la raíz del árbol de la biblioteca de la extensión de Java. La API Java Servlet contiene los paquetes siguientes:

- Paquete `javax.servlet`
- Paquete `javax.servlet.http`

Los servlets son un poderoso agregado al ambiente de Java. Son rápidos, seguros, confiables, y 100% puro Java. Debido a que los servlets se incluyen en un servidor existente, reemplazan mucho código y tecnología existentes. El servidor dirige las conexiones de red, la negociación del protocolo, el tipo de carga, y más; todo este trabajo no necesita ser replicado por cada petición, como sucedía con CGI. Y, debido que los servlets están situados en como un proceso intermedio, se utilizan para agregar valor y flexibilidad a un sistema.

Rol estructural de los servlets.

Debido a su potencia y flexibilidad, los servlets pueden desempeñar un papel significativo en una arquitectura del sistema. Pueden realizar el procesamiento asignado a una capa intermedia, actúan como proxy para un cliente, e incluso aumentan las características de la capa intermedia agregando soporte de nuevos protocolos u otras características. Una capa intermedia actúa como un servidor de aplicación en los sistemas de tipo cliente/servidor de tres capas, colocándose entre un cliente ligero como un browser y una fuente de datos.

Procesos de la capa intermedia o de aplicación

En muchos sistemas una capa intermedia o de aplicación sirve como enlace entre los clientes y los servicios finales. Al utilizar una capa intermedia o de aplicación, muchos procesos pueden ser excluidos tanto de clientes (haciéndolos más ligeros y más rápidos) y de servidores (permitiendo que se concentren en su misión).

Una ventaja del procesamiento de capa intermedia es la simplicidad en la administración de la conexión. Un conjunto de servlets podría manejar conexiones con centenares de clientes, incluso miles, mientras recicla una conjunto de costosas conexiones a servidores de bases de datos.

Otras funciones de la capa intermedia incluyen:

- Ejecución de reglas de navegación.
- Administración de transacciones.

- Mapeo de clientes a un sistema de servidores con redundancia (garantizando la alta disponibilidad del conjunto).
- Soporte de diferente tipos de clientes: aquellos que soportan sólo HTML, ó HTML y Java, etc.

Servidores proxy.

Cuando se utilizan los servlets para soportar applets, pueden actuar como sus propios proxy. Esto puede ser importante porque la seguridad de Java permite que los applet hagan solamente conexiones al servidor desde el cual fueron descargados. Si un applet necesita conectarse con un servidor de la base de datos situado en otra máquina, un servlet puede hacer esta conexión a nombre del applet.

Soporte de protocolos.

La API Servlet proporciona una relación estrecha entre servidor y los servlets. Esto permite que los servlets agreguen soporte de nuevos protocolos a un servidor. Esencialmente, cualquier protocolo que siga un modelo de requerimiento y respuesta se puede atender a través de un servlet. Esto podría incluir: SMTP, POP, FTP, etc.

Debido a que HTTP es uno de los protocolos más comunes, y porque el HTML puede proporcionar una atractiva presentación de la información, los servlets contribuyen probablemente en la construcción de la mayoría a los sistemas basados en HTTP.

Soporte de HTML.

El lenguaje HTML proporciona una atractiva presentación de la información debido a su flexibilidad y la amplia gama de contenidos que puede soportar. Los servlets pueden desempeñar un papel importante en la generación de contenidos HTML. De hecho, el soporte de servlet para HTML es tan común, que el paquete `javax.servlet.http` soporta protocolo HTTP

y generación HTML. Los sitios complejos de la web necesitan a menudo proporcionar páginas HTML que se adapten a cada visitante, o aún para cada clic ó enlace. Los servlets pueden ser escritos para procesar páginas HTML y modificarlas para requisitos particulares mientras que se envían a un cliente. Esto puede ser tan simple como las substituciones en línea o ser tan complejo como la compilación de una descripción basada en gramática de una página y la correspondiente generación del HTML configurado.

Generación en línea de HTML.

Algunos servidores web, tales como el servidor Web de Java (JWS), permiten que las etiquetas del servlet sean encajadas directamente en archivos del HTML (también es el caso de apache Tomcat). Cuando el servidor encuentra tal etiqueta, llama al servlet mientras que está enviando el archivo HTML al cliente. Esto permite que un servlet inserte su contribución directamente en el stream saliente del HTML.

Server-Side Includes

Otro ejemplo de procesamiento en línea es conocido como server-sides includes (SSI). Con SSI, una página HTML puede contener comandos especiales que se procesan cada vez que la página se solicita. Un servidor web requiere generalmente los archivos HTML que incorporan SSI para utilizar una extensión única, tal como shtml.

Los servlets son una buena manera de agregar procesamiento SSI a un servidor web. Con más y más servidores web que soporten servlets, sería posible estandarizar el uso de servlets para SSI y utilizarlo en diversos servidores web.

Substitución de scripts CGI.

Un servlet HTTP es un reemplazo directo para los scripts CGI. Los servlets HTTP son accesados por el usuario al escribir una URL en un browser o como producto de una acción de un formulario HTML. Por ejemplo, si un

usuario incorpora el siguiente URL en un campo de dirección del browser, el browser solicita un servlet para enviar una página HTML con el tiempo actual: http://localhost/servlet/DateTimeServlet. El DateTimeServlet responde a esta petición enviando una página HTML al browser. Nótese que los servlets no están restringidos a generar páginas Web; pueden realizar cualquier otra función, tal como almacenar y traer la información de la base de datos, o abrir un socket a otra máquina.

Instalación de Servlets.

Los servlets no se ejecutan en el mismo sentido que los applets y aplicaciones. Proporcionan una funcionalidad que extiende el servidor web. Para probar un servlet, se requieren dos pasos:

1. Instalar los servlets en un servidor web.
2. Solicitar un servicio de servlet's vía una petición del cliente

Hay muchos servidores web que soportan servlets. Está más allá del alcance de este documento cubrir las diversas maneras de instalar servlets en cada tipo servidor. En este trabajo se utilizó el ambiente de desarrollo NetBeans, que incorpora el servidor Apache Tomcat.

Servlets temporales v/s. Servlets permanentes.

Los servlets pueden ser iniciados y detenidos por cada requerimiento de un cliente, o pueden ser iniciados en el mismo instante que el servidor web y permanecer activos mientras lo esté el servidor web. Los servlets temporales son cargados cuando son necesarios, y ofrecen una buena manera de ahorrar recursos del servidor.

Los servlets permanentes son cargados con el servidor web y se mantienen mientras lo haga el servidor. Son instalados como extensiones permanentes del servidor cuando el costo de hacerlos partir es muy alto (tales como establecer una conexión con un DBMS), cuando ofrecen una

funcionalidad del servidor permanente (tal como un servicio RMI) , o cuando deben responder los más rápido posible a los requerimientos de los clientes.

No es necesario ningún código especial para dejar un servlet temporal o permanente. Esto es función de la configuración del servidor.

Servlet API

La API Java Servlet define el interfaz entre los servlets y los servidores. Esta API se empaqueta como extensión estándar al JDK bajo `javax`:

- Paquete `javax.servlet`
- Paquete `javax.servlet.http`

La API proporciona soporte en cuatro categorías:

- Administración del ciclo de vida de un servlet.
- Acceso al contexto del servlet.
- Clases utilitarias.
- Clases de soporte específico para HTTP.

Ciclo de vida de un Servlet

Los servlets corren en la plataforma del servidor web como parte del mismo proceso que el mismo servidor web. El servidor web es responsable de inicializar, de invocar, y de destruir cada instancia de un servlet. Un servidor web se comunica con un servlet a través de un interfaz simple, `javax.servlet.Servlet`. Este interfaz se compone de tres métodos principales:

- `init()`
- `service()`
- `destroy()`

y dos métodos secundarios:

- `getServletConfig()`
- `getServletInfo()`

Es posible distinguir una similitud entre esta interfaz y la interfaz de los applets. Esto es por diseño. Los servlets son a los servidores web lo que los applets son a los browsers. Un applet corre en un browser, realizando acciones solicitadas a través de una interfaz específica. Un servlet hace lo mismo, corriendo en un servidor web.

El método `init()`

Cuando un servlet se carga, primero se invoca su método `init()`. Esto permite al servlet realizar cualquier proceso de inicialización, por ejemplo abrir archivos, ó establecer conexiones a sus servidores. Si un servlet se ha estado instalado permanentemente en un servidor, carga cuando el servidor comienza a funcionar. Si no, el servidor activa un servlet cuando recibe el primer requerimiento de un cliente de un servicio proporcionado por dicho servlet.

El método del `init()` garantiza terminar antes de que se haga cualquier otra llamada al servlet - tal como una llamada al método del `service()`. Observe que el método `init()` será llamado solamente una vez; no será llamado de nuevo a menos que el servlet haya sido descargado y después recargado por el servidor.

El método del `init()` toma un argumento, una referencia un objeto de la clase `ServletConfig`, el cual provee los argumentos de inicialización para el servlet. Este objeto tiene un método denominado `getServletContext()` que devuelve un objeto de la clase `ServletContext` que contiene la información sobre el ambiente del servlet.

El método `service()`

El método `service()` es el corazón del servlet. Cada petición de un cliente da lugar a una llamada al método `service()` del servlet. El método `service()` lee la petición y produce el mensaje de respuesta a partir de sus dos parámetros:

- Un objeto de la clase `ServletRequest` con datos del cliente. Los datos consisten en pares nombre/valor de parámetros y de un `InputStream`. Varios métodos devuelven información de los parámetros del cliente. El `InputStream` enviado por el cliente se puede obtener a través del método `getInputStream()`. Este método devuelve un `ServletInputStream`, que se puede utilizar para conseguir datos adicionales del cliente. Si interesa realizar un procesamiento a nivel de caracteres de datos en vez de bytes de datos, se puede obtener un objeto `BufferedReader` en lugar del stream de entrada con el método `getReader()`.

- Un `ServletResponse` representa la contestación del servlet al cliente. Al preparar una respuesta, el método `setContentType()` se invoca primero para fijar el tipo MIME de la respuesta. Después, el método `getOutputStream()` o el método `getWriter()` se pueden utilizar para obtener un `ServletOutputStream` o un `PrintWriter`, respectivamente, para enviar datos de regreso al cliente.

Como se puede ver, hay dos maneras para que un cliente envíe la información a un servlet. La primera consiste en enviar valores de parámetros y la segunda forma es enviar la información vía un `InputStream` (o `Reader`). Los valores de los parámetros se pueden encajar en un URL (método GET definido por HTTP).

El trabajo del método `service()` es conceptualmente simple -- crea una respuesta para cada petición del cliente enviada a él a través del servidor

web. Sin embargo, es importante destacar que pueden haber múltiples peticiones de servicio que son procesadas inmediatamente. Si el método `service()` requiere cualquier recurso externo, tales como archivos, bases de datos, o un ciertos datos externos, se debe asegurar de que el acceso del recurso sea seguro para el hilo en el cual corre el servlet.

El método `destroy()`

El método `destroy()` se llama para permitir que el servlet limpie o desocupe cualquier recurso (tal como archivos o conexiones abiertas a la base de datos) antes de que se descargue el servlet. Si no se requiere ninguna operación de limpieza, este puede ser un método vacío. El servidor espera para llamar el método `destroy()` hasta que, ó todas las llamadas de servicio son completadas, o cierta cantidad de tiempo ha pasado. Esto significa que el método `destroy()` puede ser llamado mientras que algunos métodos `service()` todavía están funcionando. Es importante que se escriba el método `destroy()` considerando este hecho para evitar de cerrar cualquier recurso necesario hasta que todas las llamadas al método `service()` hayan terminado.

Código de ejemplo báico de un Servlet

El siguiente código implementa un servlet simple que vuelve una página HTML estática a un browser. Este ejemplo implementa completamente la interfaz `Servlet`.

```
import java.io.*;
import javax.servlet.*;
public class SampleServlet implements Servlet {
  private ServletConfig config;

  public void init (ServletConfig config)
    throws ServletException {
    this.config = config;
  }
```

```
  public void destroy() {} // este método no hace nada

  public ServletConfig getServletConfig() {
    return config;
  }

  public String getServletInfo() {
    return "Mi primer servlet";
  }

  public void service (ServletRequest req,
    ServletResponse res
  ) throws ServletException, IOException  {
    res.setContentType( "text/html" );
    PrintWriter out = res.getWriter();
    out.println( "<html>" );
    out.println( "<head>" );
    out.println( "<title>Mi primer servlet</title>" );
    out.println( "</head>" );
    out.println( "<body>" );
    out.println( "<h1>Este es mi primer servlet... </h1>"
);
    out.println( "</body>" );
    out.println( "</html>" );
    out.close();
  }
}
```

Contexto de los Servlets.

Un servlet se ejecuta dentro del contexto del proceso del servidor web ó del servidor de aplicación. Para entender el ambiente de operación del servlet, un servlet puede obtener información de su entorno en diferentes momentos.

Clases Utilitarias

Hay varias utilidades proporcionadas en la API Servlet . La primera es la interfaz `javax.servlet.SingleThreadModel` que facilita escribir servlets simples. Si un servlet implementa esta interfaz, el servidor sabe que debe nunca llamar el método `service()` del servlet mientras esté procesando una

petición. Es decir, el servidor procesa todas las peticiones del servicio dentro de un único hilo.

Mientras que esto hace más fácil escribir un servlet, disminuye el rendimiento que puede alcanzar.

Dos clases de excepciones se incluyen en la API Servlet. La excepción `javax.servlet.ServletException` puede ser utilizada cuando hay una falta general en el servlet. Esto notifica al servidor que hay un problema. La excepción `javax.servlet.UnavailableException` indica que un servlet es inasequible. Los servlets puede lanzar esta excepción en cualquier momento. Hay dos tipos de indisponibilidad:

- Permanente. El servlet no puede funcionar hasta que un administrador toma una cierta acción. En este estado, un servlet debe escribir una entrada del registro con un informe del problema y resoluciones posibles
- Temporal. El servlet encontró problema temporal (potencial), tal como un disco lleno, un servidor fallado, un etc. El problema puede corregirse con tiempo o puede requerir la intervención del operador.

Soporte para HTTP

Servlets que utilicen el protocolo del HTTP son muy comunes. No debe ser una sorpresa que exista ayuda específica para los desarrolladores de este tipo de servlets. El soporte para protocolo HTTP se proporciona en el paquete `javax.servlet.http`. Antes de detallar este paquete, se repasa brevemente el protocolo HTTP.

HTTP es el acrónimo de protocolo de transferencia de hipertexto. Define un protocolo usado por los browsers y los servidores web para comunicarse entre ellos. El protocolo define un conjunto peticiones basadas en mensajes de texto métodos HTTP. (nota: La especificación del HTTP llama este conjunto de mensajes métodos HTTP; no se confunda este término con los métodos de Java. Se puede pensar los métodos HTTP como mensajes que solicitan cierto tipo de respuesta). Los métodos del HTTP incluyen GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT, OPTIONS.

En este breve repaso se miran solamente tres de estos métodos: GET, HEAD, y POST.

***El Método HTTP** GET*

El método HTTP GET la requiere información desde el servidor web. Esta información podría ser un archivo, salida de un dispositivo en el servidor, o salida de un programa (tal como un servlet o escritura de un script CGI). Un ejemplo de método HTTP GET corresponde al requerimiento de la página principal de www.mageland.com:

```
GET / HTTP/1.1
 Connection: Keep-Alive
 User-Agent: Mozilla/4.0 (
  compatible;
  MSIE 4.01;
  Windows NT)
 Host: www.magelang.com
 Accept: image/gif, image/x-xbitmap,
  image/jpeg, image/pjpeg
```

En la mayoría de los servidores web, los servlets son accesados vía URLs que comienzan con /servlet/. El siguiente método HTTP GET está solicitando el servlet MyServlet en el servidor www.magelang.com :

```
GET /servlet/MyServlet?name=Scott&
   company=MageLang%20Institute HTTP/1.1
 Connection: Keep-Alive
 User-Agent: Mozilla/4.0 (
  compatible;
  MSIE 4.01;
  Windows NT)
 Host: www.magelang.com
 Accept: image/gif, image/x-xbitmap,
  image/jpeg, image/pjpeg
```

El URL en este método GET invoca el servlet llamado MyServlet y le pasa dos parámetros, name y compay. Cada parámetro es un par de

nombre/valor que sigue el formato nombre=valor. Los parámetros son especificados a continuación del nombre del servlet con un signo de interrogación ('?'), y donde cada parámetro se separa por un signo "&" ('&').

Observe el uso de %20 en el valor del parámetro company. Un espacio señalaría el fin del URL en la línea de la petición del GET, así que debe ser coficado, o substituido por %20. Los desarrolladores de servlets no necesitan preocuparse de esta codificación pues es decodificada automáticamente por la clase `HttpServletRequest`.

El método HTTP GET tiene una limitación importante. La mayoría de los servidores web limita cuánto datos se pueden pasar como parte del nombre del URL (generalmente algunos cientos bytes.) Si se deben pasar más datos entre el cliente y el servidor, el método del POST del HTTP se debe utilizar en lugar de GET.

Cuando un servidor contesta a una petición HTTP GET, envía un mensaje de respuesta HTTP devuelta. El encabezado de una respuesta HTTP se ve de la siguiente forma:

```
HTTP/1.1 200 Document follows
  Date: Tue, 14 Apr 1997 09:25:19 PST
  Server: JWS/1.1
  Last-modified: Mon, 17 Jun 1996 21:53:08 GMT
  Content-type: text/html
  Content-length: 4435

  <4435 bytes worth of data -- the document body>
```

***El Método** HEAD*

El método HEAD del HTTP es muy similar al método GET. La petición se ve exactamente igual que la petición GET (a menos que la palabra HEAD se utiliza en vez de GET), pero el servidor web solamente devuelve la información del encabezado.

HEAD se utiliza a menudo para comprobar lo siguiente:

- La última fecha de modificación de un documento en el servidor para los propósitos de almacenaje en el cache.
- El tamaño de un documento antes de descargar (el browser puede presentar la información del progreso).
- El tipo del servidor, permitiendo que el cliente modifique los pedidos para requisitos particulares ese servidor.
- El tipo del documento solicitado, así que el cliente pueden estar seguro que lo soporta.

El Método POST

Una petición POST HTTP permite que un cliente envíe datos al servidor. Estos datos se pueden utilizar para varios propósitos, por ejemplo:

- Fijar la información a de un newsgroup.
- Agregar entradas a un libro de visitas de un determinado sitio.
- Pasar más información que lo que una petición GET permite.

El método GET pasa todas sus argumentos como parte del URL. Muchos servidores web tienen un límite de cuánto datos pueden aceptar como parte del URL. El método del POST pasa todos sus parámetros como un flujo de datos de entrada (`InputStream`), quitando este límite.

Una petición típica del POST puede ser como sigue:

```
POST /servlet/MyServlet HTTP/1.1
  User-Agent: Mozilla/4.0 (
   compatible;
   MSIE 4.01;
   Windows NT)
  Host: www.magelang.com
  Accept: image/gif, image/x-xbitmap,
   image/jpeg, image/pjpeg, */
  Content-type: application/x-www-form-urlencoded
  Content-length: 39

  name=Scott&company=MageLang%20Institute
```

Clases de soporte para HTTP

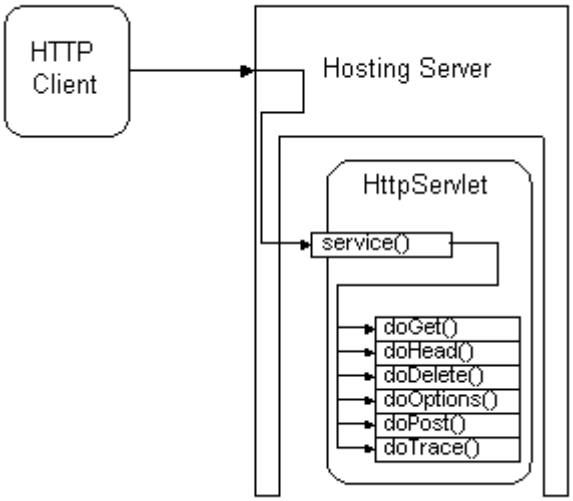
Ahora que se ha reseñado brevemente el protocolo HTTP, es posible considerar cómo el paquete `javax.servlet.http` ayuda a escribir servlets con soporte para HTTP. La clase abstracta `javax.servlet.http.HttpServlet` proporciona una implementación de la interfaz `javax.servlet.Servlet` e incluye mucha funcionalidad provechosa por defecto. La manera más fácil de escribir un servlet HTTP es extender la clase `HttpServlet` y agregar su propio procesamiento.

La clase `HttpServlet` proporciona una implementación del método del `service()` que envía los mensajes HTTP a uno de varios métodos especiales. Estos métodos son:

- `doGet()`
- `doHead()`
- `doDelete()`
- `doOptions()`
- `doPost()`
- `doTrace()`

corresponden directamente con los métodos del protocolo HTTP.

Según lo mostrado en el diagrama siguiente, el método `service()` interpreta cada método HTTP y se determina si es un HTTP GET, HHTP POST, HTTP HEAD, u otro método del protocolo HTTP:



Utilización de las clases de soporte de HTTP

Cuando se usan las clases de soporte de HTTP, se crea generalmente un nuevo servlet que extiende la clase `HttpServlet` y sobrescribir el método `doGet()` o el método `doPost()`, o posiblemente ambos. Otros métodos se pueden eliminar para conseguir un control más fino.

Los métodos de procesamiento del HTTP se pasan a través de dos parámetros: un objeto de la clase `HttpServletRequest` y un objeto de la clase `HttpServletResponse`. La clase `HttpServletRequest` tiene varios métodos convenientes que ayudan a analizar la petición.

Un método `doGet()` del servlet debe:

- Leer los datos de la petición, tales como parámetros de la entrada.
- Establecer el encabezamiento de la respuesta (longitud, tipo, y codificación).
- Escriba los datos de la respuesta.

Es importante observar que se espera que la manipulación del método GET sea segura e idempotente (es decir que funcione de la misma forma para una ó múltiples peticiones).

El método doPost() de un servlet debe ser sobrescrito cuando se necesita procesar formularios HTML ó manipular grandes cantidades de datos que serán enviadas por el cliente.

Conclusiones

La API Java Servlet es una extesión estandar. Esto quiere decir que hay una definición explícita de las interfaces de los servlets, pero que no es parte del Java Developmente Kit (JDK) 1.1 ó de la plataforma Java 2. Las clases de servlets están incluídas en la Java Servlet Development Kit (JSDK) versión 2.0 de la Sun (http://java.sun.com/products/servlet/index.jsp). La siguiente tabla resume los dos principales paquetes que implementan las clases de servlets:

javax.servlet: Soporte General de Servlets	
`Servlet`	Una interfaz que define la comunicación entre el servidor web y un servlet. Esta interfaz define los métodos `init()`, `service()`, y `destroy()` (y algunos otros).
`ServletConfig`	Una interfaz que describe los parámetros de configuración para un servlet. Estos parámetros son pasados al servlet cuando el servidor web invoca el método `init()`. Nótese que el servlet debería grabar la referencia al objeto `ServletConfig`, y definir un método `getServletConfig()` para retornarla cuando sea necesario. Esta interfaz define como obtener los parámetros de inicialización para el servlet y el contexto bajo el cual se ejecuta el servlet.
`ServletContext`	Una interfaz que describe como un servlet puede obtener información acerca del servidor en el cual se está ejecutando. Esta información puede ser obtenida a través del método `getServletContext()` del objeto `ServletConfig`.
`ServletRequest`	Una interfaz que describe como obtener información acerca de una petición de un cliente.
`ServletResponse`	Una interfaz que describe como devolver información al cliente.
`GenericServlet`	Una implementación base de un servlet. Se encarga de guardar la referencia al objeto `ServletConfig` y provee varios metodos que delegan su funcionalidad al objeto `ServletConfig`. También provee una implementación básica para los métodos `init()` y `destroy()`.
`ServletInputStream`	Una subclase de un `InputStream` usada para leer la parte de datos contenida en el requerimiento de un cliente. Se agrega un método `readLine()` por conveniencia.
`ServletOutputStream`	Un objeto `OutputStream` el cual responde hacia el cliente cuando es accesado.
`ServletException`	Debe ser lanzada cuando se encuentra un problema en el servlet.
`UnavailableException`	Debe ser lanzada cuando el servlet no está disponible por alguna razón.

javax.servlet.http: Soporte para HTTP Servlets	
HttpServletRequest	Una subclase de `ServletRequest` que define varios métodos que analizan los encabezados de un requerimiento HTTP.
HttpServletResponse	Una subclase de `ServletResponse` que provee acceso e interpretación de los códigos de estado del protocolo HTTP e información del encabezado.
HttpServlet	Una subclase de `GenericServlet` que provee una separación automática de los requerimientos HTTP por el tipo de método. Por ejemplo, un requerimiento HTTP `GET` será procesado por el método `service()` y pasado al método `doGet()`.
HttpUtils	Una clase que provee asistencia para el análisis de requerimientos HTTP `GET` y `POST`.

Anexos

Ejemplos de Servlets

A continuación se muestra el código fuente de dos ejemplos que fueron mostrados en la presentación correspondiente. Ambos ejemplos se desarrollaron utilizando el IDE NetBeans.

Ejemplo 1.

```
/*
 * FormBasedHello.java
 * Created on 15 de noviembre de 2004, 11:49
 */
package ejemplosCurso;

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

/**
 * @author  PROJAS
 * @version
 */
public class FormBasedHello extends HttpServlet {
    private static final String DEFAULT_NAME = "World";

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    public void destroy() {
    }
```

```
<code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    // process request <=> generate response
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        // Determine the specified name (or use default)

        String name = request.getParameter("name");

        if ( (name == null) || (name.length() == 0) ) {
            name = DEFAULT_NAME;
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Hello Servlet</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY BGCOLOR='white'>");
        out.println("<B>Hello, " + name + "</B>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }

    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    public String getServletInfo() {
        return "Short description";
    }
}
```

Ejemplo 2

```
/*
 * LanguageServlet2.java
 *
 * Created on 15 de noviembre de 2004, 19:32
 */

package ejemplosCurso;

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

/**
 *
 * @author  PROJAS
 * @version
 */
public class LanguageServlet2 extends HttpServlet {

    /** Initializes the servlet.
     */
    String langArray [] = {"Hello buddy!",
                           "alles klar?",
                           "c'est superbe!",
                           "Hola nenes!"};
    int n = 0;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);

    }

    /** Destroys the servlet.
     */
    public void destroy() {

    }

    /** Processes requests for both HTTP <code>GET</code> and
<code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String lenguaje = request.getParameter("radiobutton");

        /*if (lenguaje.compareTo("ingles") == 0) n = 0;
        if (lenguaje.compareTo("aleman") == 0) n = 1;
        if (lenguaje.compareTo("frances") == 0) n = 2;
```

```
        if (lenguaje.compareTo("castellano") == 0) n = 3;*/
        if (lenguaje.equals("ingles")) n = 0;
        if (lenguaje.equals("aleman")) n = 1;
        if (lenguaje.equals("frances")) n = 2;
        if (lenguaje.equals("castellano")) n = 3;


        /* TODO output your page here
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet</title>");
        out.println("</head>");
        out.println("<body>");

        out.println("</body>");
        out.println("</html>");
         */
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Language Servlet</title>");
        out.println("</head>");
        out.println("<body bgcolor='white'>");
        out.println("Your message is: <BR>");
        out.println("<B>" + langArray[n] + "</B>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }

}
```

Referencias

- Design & Development of simple Java[tm] servlet applications. https://learningcenter-sai.sun.com/gui/ito/generic_index.jsp
- The Apache Jakarta Project http://jakarta.apache.org
- Package javax.servlet http://java.sun.com/products/servlet/2.1/api/Package-javax.servlet.html
- The Java Servlet API http://java.sun.com/developer/onlineTraining/Servlets/Fundamentals/servlets.html
- The NetBeans IDE http://www.netbeans.org