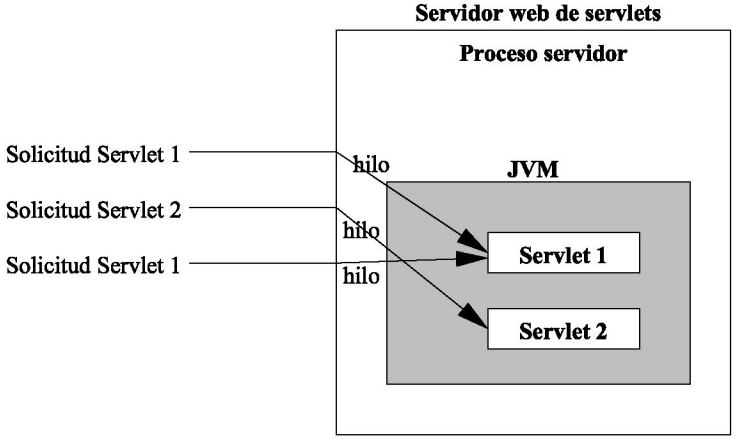
Servlets

Índice

1. Introducción

1.1. Características fundamentales

- Extensiones de funcionalidad en el servidor.
- Se ejecutan en una máquina virtual dentro del proceso del servidor.
- Motor de servlets: *software* capaz de ejecutar servlets.
- Cada petición se ejecuta en un hilo, e invoca un método del servlet (*service()*).
- Además de servidores de HTTP, pueden extender cualquier tipo de servidor (por ejemplo, FTP).



1.2. Ventajas fundamentales

- Portabilidad:
  - Entre plataformas.
  - Entre servidores.
- Potencia:
  - APIs de Java: acceso a red, hilos, manipulación de imágenes, acceso a bases de datos, compresión de datos, internacionalización, RMI, CORBA, serialización de objetos, acceso a directorio, etc.
  - Utilización de código externo, incluído EJBs.

- Eficiencia:
  - Instancia permanententemente cargada en memoria por cada servlet.
  - Ejecución de peticiones mediante invocación de un método.
  - Cada petición se ejecuta en un hilo.
  - Mantiene automáticamente su estado y recursos externos: conexiones a bases de datos, conexiones de red, etc.
- Seguridad:
  - Lenguaje java: máquina virtual, chequeo de tipos, gestión de memoria, excepciones, etc.
  - Gestor de seguridad de Java.
- Elegancia:
  - Código java: modular, orientado a objetos, limpio y simple.
  - API servlets: potente y fácil de utilizar.
- Integración:
  - Integración fuerte entre servlets y servidor: permite colaboración entre ambos.
- Extensibilidad y flexibilidad:
  - API Servlet extensible.
  - Filtros (cadenas de servlets).
  - Integrable con JSP (Java Server Pages).

2. Programación de servlets

2.1. Ciclo de vida de un servlet

2.1.1. Ciclo de vida

1. Instanciación e inicialización del servlet.
2. Gestión de cero o más peticiones (normalmente, cada petición en un hilo).
3. Destrucción del servlet.

2.1.2. Consecuencias del ciclo de vida

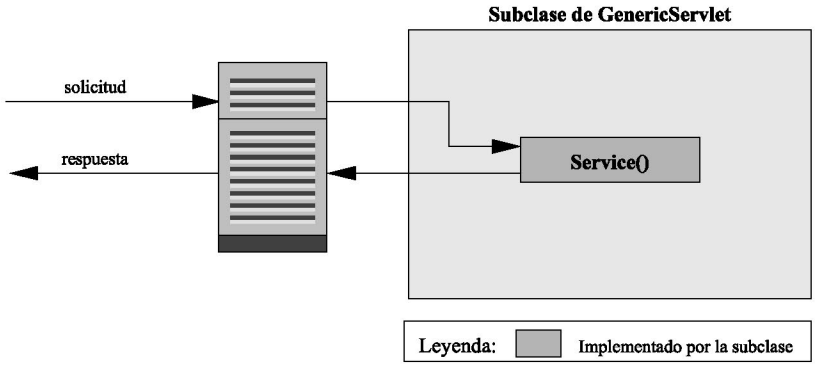
- Una única máquina virtual:
  - Compartición de datos entre servlets.
- Persistencia de instancias:
  - Consumo reducido de memoria.
  - Eliminación del tiempo de instanciación e inicialización.
  - Persistencia de estado, datos y recursos:
    - Atributos del servlet persistentes.
    - Conexiones permanentes a bases de datos.
    - Etc.
  - Persistencia de hilos.
- Necesidad de sincronización:
  - Problemas si se accede a los mismos datos concurrentemente desde distintos hilos (atributos de clase o instancia, base de datos, etc.)
  - Necesidad de utilizar sincronización (por ejemplo, *sinchronized*).
  - *SingleThreadModel*: evita la concurrencia (*pool* de instancias).

2.2. Servlet API

- *javax.servlet*
- *javax.servlet.http*

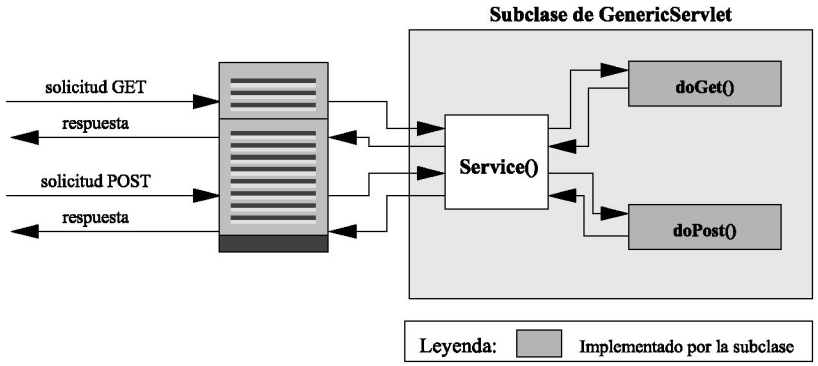
2.3. Interfaz Servlet

- Un servlet implementa la interfaz *javax.servlet.Servlet*, o hereda de implementaciones de la API:
  - *javax.servlet.GenericServlet*
  - *javax.servlet.HttpServlet*
- Métodos importantes de la interfaz:
  - *void service(ServletRequest req, ServletResponse res)*
  - *void init(ServletConfig config)*
  - *void destroy()*



2.4. Servlet HTTP

- Hereda de *javax.servlet.HttpServlet*:
  - Implementa *service()*, que invoca al método correspondiente al método HTTP de la petición:
    - *void doGet(HttpServletRequest req, HttpServletResponse resp)*
    - *void doPost(HttpServletRequest req, HttpServletResponse resp)*
    - *void doHead(HttpServletRequest req, HttpServletResponse resp)*
    - *void do...(HttpServletRequest req, HttpServletResponse resp)*
    - *getLastModified(HttpServletRequest req)*
- No suele redefinir *service()*
- Redefine los métodos *do...()* convenientes:
  - Ejemplo: para procesar peticiones GET redefine *doGet*



2.5. Clases e interfaces útiles

- Interfaz *ServletConfig*
- Interfaz *ServletContext*
- Interfaz *HttpServletRequest*
- Interfaz *HttpServletResponse*
- Interfaz *HttpSession*
- Clase *Cookie*

3. Acceso a información

3.1. Parámetros de inicio

- Interfaz *ServletConfig*
- Método *getInitParameter()*

3.2. Información acerca del servidor

- Método *ServletRequest.getServerName()*
- Método *ServletRequest.getServerPort()*
- Método *ServletContext.getServerInfo()*
- Método *ServletContext.getAttribute(String name)*

3.3. Información acerca del cliente

- Método *ServletRequest.getRemoteAddr()*
- Método *ServletRequest.getRemoteHost()*
- Método *HttpServletRequest.getRemoteUser()*
- Método *HttpServletRequest.getHeader(...)*

3.4. Información de la petición

- Información general:
  - Método *HttpServletRequest.getMethod()*
  - Método *HttpServletRequest.getQueryString()*
  - Método *HttpServletRequest.getHeader(...)*
- Parámetros de la petición:
  - Método *ServletRequest.getParameter(String name)*
  - Método *ServletRequest.getParameterValues(String name)*
  - Método *ServletRequest.getParameterNames()*
  - ¡Actualmente no funciona con *multipart/form-data*!
- Cuerpo de la petición:
  - Método *ServletRequest.getContentType()*
  - Método *ServletRequest.getContentLength()*
  - Método *ServletRequest.getInputStream()*
  - Método *ServletRequest.getReader()*

4. Respuesta

4.1. Tipos de datos de respuesta

- Código de estado.
- Cabeceras.
- Cuerpo.

4.1.1. Código de estado

- *HttpServletResponse.sendError(int sc)*
- *HttpServletResponse.setStatus(int sc)*
- *HttpServletResponse.sendRedirect(String location)*

4.1.2. Cabeceras

- *HttpServletResponse.setHeader(String name, String value)*
- *HttpServletResponse.addCookie(Cookie cookie)*
- *ServletResponse.setContentType(String type)*
- *ServletResponse.setContentLength(int length)*

4.1.3. Cuerpo

- *ServletResponse.getOutputStream()*
- *ServletResponse.getWriter()*

5. Gestión de sesiones

5.1. API de gestión de sesiones

- Interfaz *HttpSession*
- Método *HttpServletRequest.getSession(boolean create)*
- Método *HttpSession.putValue(String name, Object value)*
- Método *HttpSession.getValue(String name)*
- Método *HttpSession.getValueNames()*
- Método *HttpSession.isNew()*
- Método *HttpSession.invalidate()*
- Método *HttpSession.getCreationTime()*
- Método *HttpSession.getLastAccesedTime()*