

Capítulo 3



Servlet

Servlets



- Ya conocemos el trabajo del contenedor de servlets
 - Crea el request y response
 - Crea el nuevo thread del servlet
 - Invoca el método service() del servlet
 - Entrega el request y response como argumentos
- Los servlets tienen solo un estado: inicializado
 - Si no está inicializado, entonces está siendo inicializado o destruido
 - Siendo inicializado: ejecutando el método init()
 - Siendo destruido: ejecutando método destroy()
 - De lo contrario, no existe

Servlets



- Secuencia del ciclo de vida
 - El contenedor carga la clase del servlet, luego
 - Instancia el servlet
 - Ejecuta el constructor de la clase
 - No deberíamos proveer este constructor, solo usar el que provee el compilador
 - Ejecución de “init()”
 - Se ejecuta sólo una vez en la vida de un servlet
 - Debe terminar su ejecución antes de que el contenedor pueda invocar a “service()”
 - Ejecución de “service()”
 - Acá el servlet gasta su mayor tiempo de vida
 - Se manejan todos los requerimientos de los clientes (doGet(), doPost(), etc) en threads separados



- Secuencia del ciclo de vida
 - Ejecución de método “destroy()”
 - Se ejecuta sólo una vez, como “init()”
 - Es lo último que se ejecuta antes de que el servlet quede disponible para el garbage collector
- Métodos heredados
 - La interfaz “Servlet” indica que todos los servlets deben tener 5 métodos
 - Service, init, destroy, getServletConfig, getServletInfo
 - La clase “GenericServlet” implementa los métodos básicos
 - Define un servlet independiente del protocolo
 - Provee versiones simples de los métodos del ciclo de vida

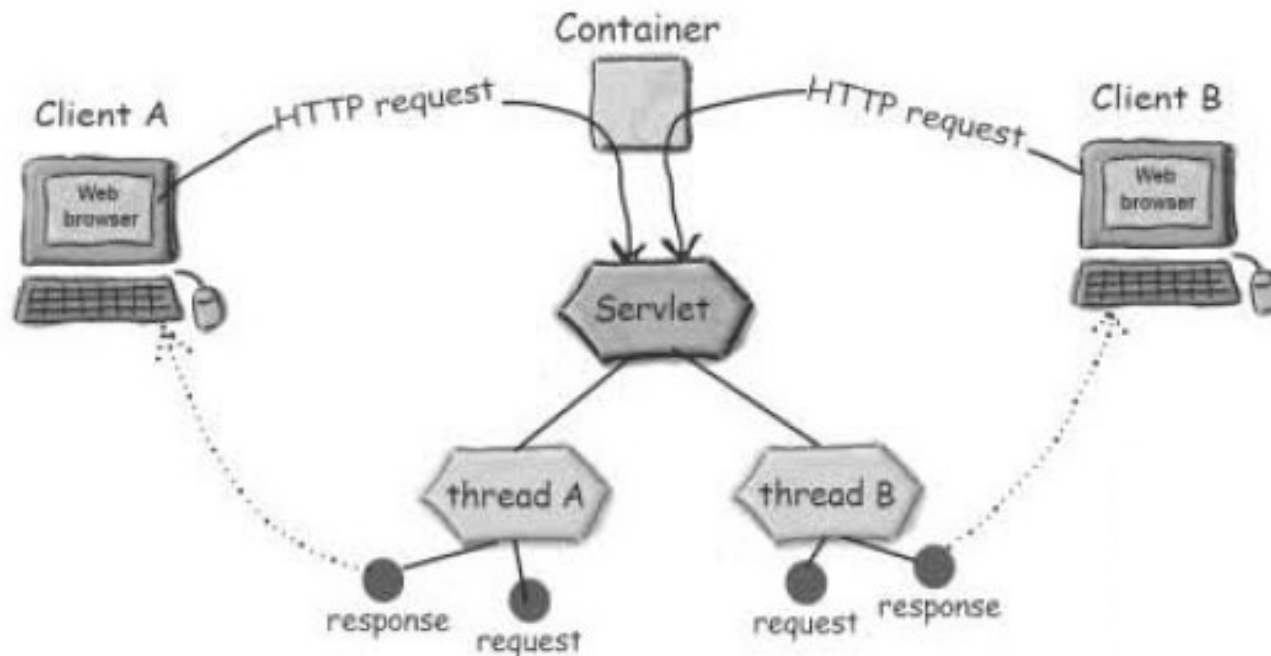


- Métodos heredados
 - La clase HttpServlet implementa el método “service”
 - El método service no recibe cualquier objeto “response” o “request”, sólo los de tipo HTTP
 - Un servlet que extiende HttpServlet, solo sobrescribirá los métodos que le interesen
 - → *Revisar API de servlet*

Servlet



- El contenedor ejecuta múltiples threads para procesar múltiples requerimientos a un servlet
 - Cada requerimiento del cliente genera un nuevo par de objetos “request” y “response”



Servlet



- El Servlet inicia su vida cuando el contenedor lo encuentra
 - Cuando se inicia el contenedor, busca las aplicaciones web implementadas
 - Busca los archivos con clases de servlets
 - Buscar las clases es el primer paso
 - Cargar las clases es el segundo paso
 - Puede suceder cuando el contenedor se inicia o cuando la solicita el primer cliente
 - El contenedor puede dar algún mecanismo para configurar la carga de clases

Servlet



- Un servlet se mueve del estado “no existe” a “inicializado” comenzando con el constructor
 - El constructor solo crea un objeto, no un servlet
 - Cuando un objeto llega a ser un servlet, puede usar la referencia a “ServletContext”
 - Con ServletContext puede obtener información del contenedor
 - El código de inicialización de un servlet debe ir en el método “init()”
 - El constructor es muy prematuro para hacer tareas específicas del servlet
 - En el método init se puede obtener información de configuración de la aplicación o buscar referencias a otras partes de la aplicación



- ServletConfig
 - Un objeto por servlet
 - Se utiliza para pasar información en tiempo de deploy al servlet
 - Ejemplo: Nombre de búsqueda de una base de datos que no se quiere tener codificado en el método init()
 - Se utiliza para acceder a ServletContext
 - Parámetros se configuran en el “Deployment Descriptor”
 - Estos parámetros no cambian mientras el servlet se está ejecutando
 - Para cambiar los valores se debe hacer un “redploy” del servlet



- ServletContext
 - Uno por web-app (debería llamarse AppContext)
 - Se utiliza para acceder a los parámetros de la aplicación
 - Se configura en el deployment descriptor
 - Se puede utilizar para colocar mensajes que otras partes de la aplicación pueden acceder
 - También se utiliza para obtener información del servidor
 - Nombre y versión del contenedor, versión de la API que es soportada



- **HttpServletRequest y HttpServletResponse**
 - Provee métodos relacionados con HTTP
 - Manejo de cookies, headers y sesión
 - Comunicación con el browser (envío de errores)
- **El método indicado en el requerimiento HTTP indica si se ejecuta doGet o doPost**
 - El requerimiento del cliente incluye el método HTTP
 - Existen 8 métodos HTTP, no sólo GET y POST
 - HEAD, TRACE, OPTIONS, PUT, DELETE, CONNECT
 - Todos tienen su método “doXXX()” asociado en HttpServletRequest, menos doConnect (no es parte de HttpServletRequest)
 - Generalmente se usa doGet para requerimientos simples y doPost para procesar formularios



- GET y POST
 - Cuando se usa GET los parámetros se ven en la URL
 - Con POST no se ven en la URL
 - Requerimientos GET se pueden dejar en bookmark
 - Con POST no se puede
 - GET está pensado para obtener datos, sin hacer cambios en el servidor
 - POST está pensado para enviar datos para que sean procesados y cambie algo en el servidor
 - GET es idempotente
 - Al igual que HEAD y PUT, según las especificaciones de HTTP 1.1
 - POST no es idempotente cuando actualiza una BD con los datos recibidos