Desarrollo Basado en Componente y Servicios

Parte I – EJBs

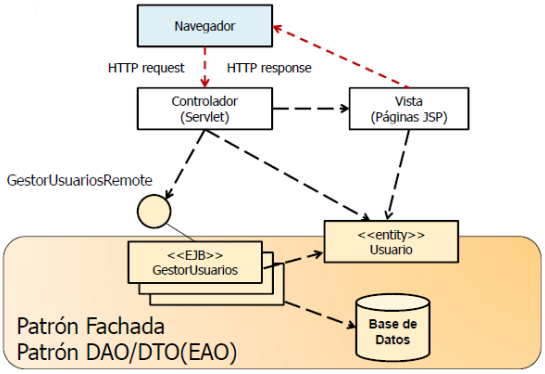
Práctica Session Beans

Ejercicios Básicos Stateless

1. **Hola Mundo. Accediendo desde un cliente JEE (el cliente es una aplicación JEE distinta a la que implementa el Bean, pero ejecutada en el mismo servidor).** Siguiendo lo indicado en: https://netbeans.org/kb/docs/javaee/entappclient.html, modificarlo para realizar el ejemplo EJB "HolaMundo" de la transparencia 7 del tema de introducción.

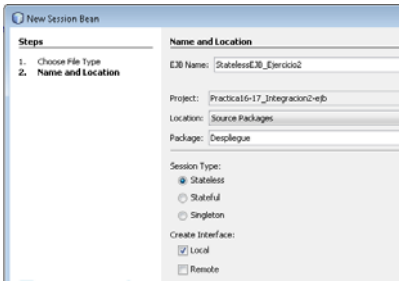
   NOTA: este ejercicio muestra una forma útil de poder realizar pruebas sobre componentes aislados, ya que la manera de llamar a un EJB realizada aquí, se puede extender a cualquier tipo de ellos.

2. **Hola Mundo. Accediendo desde aplicación web (el cliente, ahora, es una aplicación web ejecutada, también, en el mismo servidor).** Vamos a llamar al EJB desde un cliente web. Para la implementación de la capa de presentación vamos a seguir el patrón Modelo-Vista-Controlador (ver figura)



La aplicación web mostrará en su página de inicio un formulario pidiendo un nombre de usuario. Siguiendo el patrón MVC, ese nombre será enviado a un servlet (controlador), que será el que invoque al EJB "HolaMundo" (crear el mismo que en el ejercicio 1) pasándole al método "diHola" el nombre introducido en el formulario. Para este ejercicio se puede:

- crear un proyecto de tipo "Enterprise Application" que incluya los módulos EJB y Aplicación Web. En este caso, al crear el EJB se puede marcar la opción de interface Local (ver figura). Ésta será creará automáticamente por Netbeans y será modificada, también automáticamente, cada vez que añadamos un "Business Method" a nuestro EJB.



- o crear estos dos proyectos de manera independiente, actuando de manera similar al ejercicio anterior, pero aquí con un cliente de tipo web.

Es interesante probar ambos métodos para ver las diferencias.

3. **Devolviendo datos el EJB. Conversor Kilómetros – Millas**. Mediante una página web se pedirá al usuario que introduzca un número; será el número de Kilómetros a convertir. La página web pasará el dato a un servlet (controlador) que llamará a un EJB de sesión sin estado que realizará la conversión de Kilómetros a Millas. El dato se lo pasará el controlador a la página web final que lo mostrará al usuario.

4. **Hola Mundo. Accediendo desde un cliente remoto JSE (el cliente será una aplicación Java normal ejecutada sobre una JVM ajena al servidor).** Vamos a ejecutar un componente de manera remota desde una aplicación JSE. Pasos a realizar:
   a) Añadir a las bibliotecas del proyecto JSE el siguiente fichero del servidor glassfish:
      [rutaGlassfish3]/glassfish/modules/gf-client-module.jar.
   b) Definir, ya en la aplicación JSE, el servidor donde se ejecuta el EJB. Líneas básicas:

```
Properties prop = new Properties();
prop.setProperty("org.omg.CORBA.ORBInitialHost","localhost");
prop.setProperty("org.omg.CORBA.ORBInitialPort","3700");
Context context = new InitialContext(prop);
```

   Los valores dados al objeto de tipo Properties son los de por defecto, por lo que esas líneas pueden ser obviadas, poniendo sólo:

```
Context context = new InitialContext();
```

   c) Referenciar el EJB mediante *lookup*:

BeanInterface bean = (*BeanInterface*) **context.lookup("java:global**/*ModEJB*/*NombreEJB***")**;

   Las partes en negrita son las literales y en cursiva las que deben ser sustituidas por las adecuadas a cada caso. En caso de existir paquetes en nuestra aplicación EJB, no deben ser incluidos en la ruta *lookup*.
   d) Ya podemos invocar a los métodos del EJB.

   Mediante los pasos indicados, crear una aplicación JSE que ejecute el método diHola() del EJB de sesión sin estado creado en el ejercicio 1.

Ejercicios Básicos Stateful

5. **Creando carro con EJB.** Vamos a simular de manera simplificada la implementación de un carro de la compra. Para no complicar la parte web (aquí nuestro interés son los componentes), crear una página web con un formulario que permita introducir dos campos: productos y cantidad. Estos datos serán enviados a un servlet que los almacenará en un EJB de tipo “stateful”. Una vez hecho esto se volverá a la página del formulario, hasta que el usuario pulse el botón ver carro, que entonces tendremos que mostrar el contenido del carro.

   IMPORTANTE: Comprobar que la implementación del carro funciona correctamente abriendo la aplicación al mismo tiempo en dos navegadores, es decir, abriendo dos sesiones. Ir “comprando” simultáneamente diferentes productos. Al mostrar el carro al final tenemos que ver las “compras” de cada sesión, si se han mezclado se ha gestionado mal el EJB.

   IMPORTANTE: recordar que hay que añadir al EJB un método que anotaremos con @Remove que vaciará el carro. Con esto nos aseguramos que el EJB será eliminado cuando ya no le necesitemos. Llamarle tras visualizar el carro.

Ejercicios Básicos Singleton

6. **Gestionando el número de accesos mediante EJB.** Añadir a la aplicación del ejercicio anterior un contador de visitas mediante un EJB de tipo Singleton. NOTA: lo que se propone es lo mostrado como ejemplo en las transparencias de teoría, por lo que se puede reutilizar el código tanto del EJB (transparencia 17, clase “gestionAccesosBean”) como del Servlet (transparencia 23) allí incluidos.

7. **Probando la ejecución programada de tareas.** Añadir a la aplicación creada para el ejercicio anterior (o crear una nueva, daría lo mismo para lo que se quiere probar) un EJB de tipo Singleton que realice una tarea programada cada minuto (se puede poner otro tiempo, si se desea, esta parte no importa). El método asociado a @Schedule simplemente hará lo mismo que se vio en teoría (transparencia 17, clase “logAccesosBean”): mandar un mensaje con System.out con el número de clientes que han accedido a la aplicación web. Dejarlo funcionar varios minutos para ver el resultado, realizando conexiones mientras tanto a la aplicación web.

   Vamos a probar la persistencia del temporizador. Parar el servidor. Volverlo a arrancar y ver como el temporizador sigue activo, es decir, ha sido guardado y una vez arrancado de nuevo el servidor, es recuperado.

   Eliminar el despliegue de la aplicación para que no siga generando mensajes. Se puede usar para ello el IDE o la consola de administración del servidor.

Ejercicios extra

8. **Construyendo una sencilla aplicación JEE: adivinar un número.** Realizar una aplicación JEE que generará números aleatorios entre 0 y 99, que tendrá que adivinar el usuario. Con cada intento del usuario se le mostrará si ha acertado o no la cantidad, y en este caso si el número a adivinar es mayor o menor que el introducido por él. El usuario podrá jugar tantas veces como quiera, es decir, una vez adivinado un número podrá seguir con otro o finalizar el juego. La puntuación será igual al número de intentos, por lo tanto cuanto más baja mejor. Esta puntuación se le mostrará al acabar al juego.

   La aplicación constará de los dos módulos típicos: web y EJB. En el módulo web incluiremos la capa de presentación, incluido el controlador. El módulo EJB se encargará de generar los números aleatorios, comprobar si el número introducido por el usuario es menor/mayo/igual que el generado aleatoriamente y de llevar la puntuación (y cualquier otro dato que se quiera referente al historial del juego); contendrá EJBs de sesión, tanto con estado como sin estado.

   NOTA: estamos practicando EJB, por lo tanto, no se puede usar en este problema la clase HttpSession para almacenar los datos a recordar durante el juego (sesión).

9. **Otra propuesta de juego.** Otra propuesta de aplicación similar a la anterior sería implementar el juego del ahorcado mediante la tecnología EJB.