Tutorial: Cliente para servicio web REST con REST Hooks usando JAX-RS (Jersey) y Maven

El título del post parece un poco críptico, así que primero lo analizaremos con calma.

Un servicio web se puede resumir como el conjunto de tecnologías que permiten intercambiar información a través de la web entre aplicaciones.

REST (Representational State Transfer) es un estilo de arquitectura software, orientado a crear servicios web escalables. A grandes rasgos establece una serie de restricciones y permite el acceso y modificación de recursos a través de URLs mediante los verbos HTTP (GET, POST, PUT y DELETE). Para el envío de datos en el cuerpo HTTP se suele usar JSON o XML.

Los REST Hooks surgen para solventar una carencia de las arquitecturas REST. Un servicio REST permite que un cliente acceda a los recursos que el servidor ofrece, como hemos dicho, mediante HTTP. Para crear, modificar o eliminar recursos, no ofrece ninguna complicación. El problema viene cuando queremos, como cliente, saber si se ha producido un cambio en un recurso. Ante este caso, la única solución es consultar el servicio REST periódicamente para obtener dicho recurso y comprobar si ha cambiado. Esto crea una sobrecarga enorme sobre el servidor, y tiene la limitación de que siempre va a haber un retardo entre que se produce el cambio y el cliente lo detecta. Los REST Hooks permiten es una suscripción al servicio REST. El cliente hace un POST al servicio REST, suscribiéndose a un evento concreto. Para ello envía en el POST una URL en la que permanece a la escucha. El servidor, cuando se produce el evento, comprueba las suscripciones y envía un mensaje a las URLs asociadas.

Finalmente, Jersey es una implementación de la especificación JAX-RS, que permite la creación de servicios REST Java mediante el uso de anotaciones. De Jersey también utilizaremos la capacidad de mapear JSON a POJO y viceversa, cosa que hace apoyándose en Jackson, una librería Java para el procesado de JSON.

Este tutorial cuenta de tres partes. En esta primera parte veremos como implementar el cliente, y en una segunda parte afrontaremos la parte servidor. Finalmente veremos como implementar el cliente de forma standalone, evitando la necesidad de usar un servidor de aplicaciones.

Nuestro servicio REST tendrá un único punto de acceso, mediante el que suscribirse a los eventos. Periódicamente, enviará un mensaje a los clientes que se hayan suscrito. Remarcar que no contemplaremos ningún tipo de persistencia de datos, generalmente necesaria para ofrecer un servicio web funcional.

Lo primero es crear un proyecto con Maven para aplicación web con NetBeans. Completamos los

datos de nombre de aplicación, y cuando nos solicite servidor elegimos TomCat.

Dentro del paquete principal, creamos una clase a la que llamaremos TestRestHooks.

Añadimos un nuevo paquete, Models, donde definiremos nuestro modelo de datos y creamos la clase EventDTO:
[crayon-55c1b23b2669f549745652/]
Ahora en la clase TestRestHooks, usaremos anotaciones y añadiremos una función que definirá como recibiremos un evento:
[crayon-55c1b23b266ac662478568/]
Vamos a analizar las anotaciones:

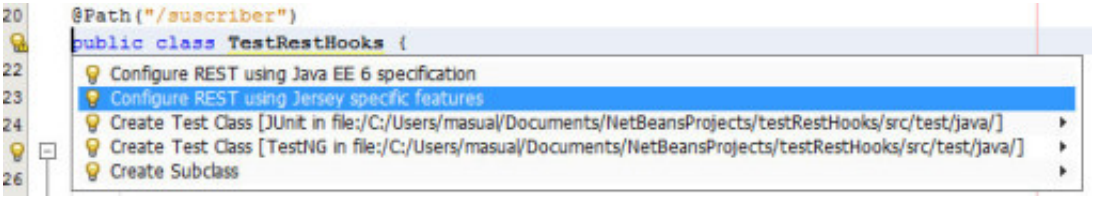
@Path("/suscriber") indica la ruta. Dentro de nuestro cliente REST llegarán a la clase TestRestHooks las peticiones que vayan a una URL "http://ip:puerto/projectname/servletname/**suscriber**" donde servletname será el nombre del servlet que correrá el servicio.

@POST indica que la función recibirá peticiones HTTP POST y @Path("event") indica que la URL a resolver por la función es "http://ip:puerto/servletname/suscriber/**event**". Finalmente @Consumes("application/json") sierve para indicar que dicha petición lleva asociado un objeto JSON. A continuación vemos la función, que recibe un EventDTO. La conversión de JSON a nuestro DTO la hace Jersey (veremos luego como configurarlo). La función únicamente imprime el nombre del evento y devuelve HTTP 200 OK si no se ha producido ningún fallo.

Debemos de resolver una serie de dependecias, ayudados por Maven. En el menú de la izquierda, donde se muestra el árbol del proyecto, hacemos clic con el botón derecho sobre "Dependencies" y "Add Dependency":

- json-20090211
- jersey-server-1.19
- jersey-bundle-1.9.1
- jersey-json 1.19
- asm-3.3.1

En este paso veremos un triangulo amarillo al lado de la primera línea. Si ponemos el cursor encima vemos que el mensaje nos dice "REST is not configured". Haciendo clic se nos muestran diferentes opciones. Seleccionamos "Configure REST using Jersey specific features".



En el árbol del proyecto vamos a "web pages", "WEB-INF". Dentro de esta carpeta creamos un archivo al que llamamos "web.xml", que contendrá la información de configuración de nuestro servicio web para poder ser lanzado por el servidor de aplicaciones (en nuestro caso TomCat). Añadimos lo siguiente:
[crayon-55c1b23b266b5072887642/]

Las líneas
[crayon-55c1b23b266bc802307092/]

son las que habilitan el mapeo a POJO, lo que permite que el JSON recibido se transforme en una entidad EventDTO de forma transparente a nosotros.

Para lanzar el proyecto sobre TomCat, en mi caso tuve que desmarcar la opción "Use IDE Proxy Settings de Tomcat". Se encuentra en el menú "Tools", "Servers", en la pestaña de Platform.

Fuentes:

REST

REST Hooks

Jersey

POJO