Ejercicios de acceso remoto y pruebas

Índice

1. Acceso remoto con HttpInvoker (1 punto)

Vamos a proporcionar acceso remoto a la aplicación de alquiler de coches de la sesión anterior. Usaremos el HttpInvoker, ya que es razonablemente eficiente y no presentará problemas con *firewalls*.

Configuración de la parte del servidor:

1. Fíjate que en el `web.xml` definimos un nuevo servlet de la clase `DispatcherServlet` al que llamamos `remoting` y lo mapeamos con las URL del tipo `/remoting/*` (no tienes que hacer nada, ya está definido). Aunque ya teníamos otro `DispatcherServlet` definido, se encargaba de la parte MVC y no es recomendable que un solo servlet se encargue de las dos cosas.
2. Modifica el `src/main/webapp/WEB-INF/config/remoting-servlet.xml` para añadir la configuración de la parte del servidor. Adapta la de los apuntes, para dar acceso remoto al interface `ICocheBO`. Cuidado con el atributo "ref": es el nombre del bean CocheBO. Como la anotación @Service en esta clase no lo especifica, el nombre por defecto será el mismo que el de la clase con la inicial en minúscula: cocheBO, como vimos en la primera sesión. Una vez creado, puedes comprobar que está inicializado intentando acceder a su URL con el navegador. Será "remoting/" seguido del "name" que le hayas puesto al bean. En realidad dará un error HTTP 500, ya que al HttpInvoker no se le puede llamar así, pero al menos debería mostrar una excepción de tipo EOFException, y sabremos "que está ahí". Si da otro tipo de excepción o un HTTP 404 es que hay algo mal configurado.

Parte del cliente:

1. Crea un proyecto de tipo "Maven project". Elige el arquetipo que te saldrá por defecto: "Maven-archetype-quickstart". Como groupId pon `es.ua.jtech` y como ArtifactId `ClienteRemotoCoches`.
2. Cambiar el pom.xml generado por Eclipse por el siguiente (tras hacer esto tendrás que ejecutar la opción Maven > Update project para que Eclipse tenga en cuenta los cambios)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>es.ua.jtech</groupId>
    <artifactId>ClienteRemotoCoches</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>ClienteRemotoCoches</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.7</version>
```

```
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>3.2.0.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>3.2.0.RELEASE</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
        </plugins>
        <finalName>ClienteRemotoCoches</finalName>
    </build>
</project>
```

3. Vamos a acceder al BO remotamente, y este nos devolverá objetos `Coche`. Por lo tanto, en el nuevo proyecto necesitarás copiar `es.ua.jtech.spring.negocio.ICocheBO` y `es.ua.jtech.spring.modelo.Coche` del proyecto original. Tendrás que crear los packages correspondientes. Cuidado, en la clase Coche que copies en el nuevo proyecto debes borrar las anotaciones de validación para no introducir dependencias del API JSR303 en el cliente. Atento: no necesitas la clase `CocheBO`, solo el interface.
4. Crea una nueva "Source Folder" (File > New > source folder) dándole como nombre "src/main/resources".
5. Crea un fichero de configuración de beans XML de Spring llamado "cliente.xml" en la carpeta "resources" que acabas de crear (File > New > Spring bean configuration file). Solo necesitarás el espacio de nombres "beans". Pon en él la configuración de la parte del cliente, fijándote en la que aparece en los apuntes y adaptándola a lo que necesitas.
6. En el método `main` de la clase `App` del proyecto, escribe código que obtenga un ICocheBO (necesitarás un ClasspathApplicationContext), llame al método obtener(String matricula) y muestre los datos de uno de los coches por pantalla con `System.out.println`.

2. Pruebas de la capa DAO (0.5 puntos)

Vamos a implementar algunas pruebas para la capa DAO. Usaremos una base de datos embebida (HSQLDB) para acelerar las pruebas

1. Lo primero es **incluir las dependencias** necesarias en el pom.xml: spring-test y hsqldb (JUnit ya está incluida en la plantilla)

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>3.2.0.RELEASE</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
        <version>2.0.0</version>
        <scope>test</scope>
    </dependency>
```

2. Ahora crearemos el **fichero de configuración de Spring para los test de la capa DAO** (créalo dentro de src/test/resources/config y llámalo daos-test.xml). Puedes tomar como modelo el de los apuntes y transparencias. En src/test/resources tienes un script para crear la base de datos e insertar los datos. Modifica el XML de los apuntes ya que éste espera que haya dos (uno para crear tablas y otro para insertar datos). *Disclaimer*: siento el trabajo rutinario, es simplemente para que no te limites a copiar y pegar literalmente el fichero entero en modo "piloto automático".
3. **Crea una clase de prueba** CocheDAOJDBCTest en el paquete es.ua.jtech.spring.datos de src/test/java (cuidado, no la metas en src/main/java). Implementa en ella una prueba del método listar del DAO. Por ejemplo puedes comprobar que hay dos coches en los datos de prueba y que la matrícula del primero de ellos es 1111JKG.

3. Pruebas de la capa BO con y sin objetos mock (1 punto)

1. **Crea una clase de prueba CocheBOTest** en el paquete es.ua.jtech.spring.bo de src/test/java, para probar la integración entre la capa de negocio y datos (es decir, sin usar mock). Implementa en ella alguna prueba que verifique que el listar del BO funciona correctamente.
2. Implementa **pruebas del BO con mocks** de la capa DAO. Tendrás que:
   - Introducir en el pom.xml la dependencia de Mockito:

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <scope>test</scope>
    <version>1.8.5</version>
</dependency>
```

   - Crear el fichero de configuración daos-mock-test.xml en src/test/resources/config. En este fichero se debe crear un mock de ICocheDAO. Puedes tomar como modelo el que crea el IUsuarioDAO en los apuntes y transparencias.
   - Crea una clase de prueba CocheBOMockTest en el paquete es.ua.jtech.spring.bo de src/test/java, para hacer pruebas unitarias del BO usando el mock. Prueba al

menos el método listar(). Tendrás que preparar el mock en el @Before para que le devuelva al BO datos de prueba.

4. Pruebas de la capa web (0.5 puntos)

Como fichero de configuración de testing para la capa web, puedes usar directamente el mismo que se está usando "en producción", el dispatcher-servlet.xml que está en src/main/webapp/WEB-INF/config. Cópialo en src/test/resources.

Implementa pruebas de integración del controller CocheController. Hazla en la clase CocheControllerTest, en el paquete es.ua.jtech.spring.mvc de src/test/java. Prueba al menos el funcionamiento del método listar() del controller. Verifica que existe un atributo en el modelo llamado "listado" y que la vista a la que se salta se llama "listar".