Informe de Parameterized Tests

1. Motivación

Teniendo el proyecto prácticamente terminado, a falta de corregir ciertos matices, y al haber planteado una discusión sobre los problemas que causa la plantilla dada para realizar los test y una posible solución, se planteó en la revisión del día 22/03/2017 realizar un segundo A+ para este proyecto. Este A+ consiste en la inclusión de test parametrizados usando la herramienta parameterized del paquete JUnit.

2. Objetivo

Nuestro objetivo es la reutilización de código en los test, de forma que una vez planteado el caso de test basado en un caso de uso en concreto, podamos modificar ciertos parámetros para realizar nuevos casos de test sobre la misma plantilla. Además, queremos que cada uno de estos test se realice por separado, de forma transaccional y, a ser posible, en diferentes test JUnit para poder identificarlos más rápidamente.

3. Inicialización del contexto de Spring

Utilizaremos Parameterized de JUnit (org.junit.runners.Parameterized) para realizar las llamadas a nuestro @Test con distintos parámetros, en vez de hacerlas a una plantilla mediante un for contenido en un @Test, con lo que conseguiremos que todos los test se hagan de forma transaccional.

Para poder usar esto, necesitamos cargar el contexto de Parameterized, al igual que lo hacíamos en Spring. Esto causa un problema en su implementación: no podemos usar @RunWith para cargar dos contextos diferentes. Hemos encontrados dos formas diferentes para arreglarlo, de las cuales una solo esta disponible a partir de la versión 4.2.0 RELEASE de Spring-test. La forma que es compatible con nuestra versión (3.2.4.RELEASE) no proporciona, desgraciadamente, manejo de transacciones, por lo que los objetos creados correctamente quedan persistidos.

3.1. Clase TestContextManager (v. 3.2.4.RELEASE)

Para poder inicializar tanto el contexto de Spring como el de Parameterized, lo que hemos realizado es la inicialización con @RunWith de Parameterized, y la

inicialización de Spring manualmente en cada @Test. Para hacerlo hemos agregado un nuevo atributo privado a la clase AbstractTest, llamado TestContextManager, clase del paquete org.Springframework.test.context.TestContextManager de Spring. Como dice en su documentación, esta clase es el punto de entrada del framework Spring TestContext. En el método Before de la clase AbstractTest le damos valor al atributo testContextManager, simplemente llamando al constructor de su clase y pasándole la clase del test como parámetro. Una vez que este atributo sabe a que clase hace referencia el test, solo tenemos que usar su método prepareTestInstance. Este método es el que se encarga de cargar todo el contexto de Spring, por ejemplo inyectando las dependencias (con etiquetas @Autowired). Si se usara, por algún motivo, algún artefacto del framework Spring en el método Before, debería ser después de este par de líneas que hemos comentado, para que el contexto esté cargado.

Como comentábamos, esto no nos permite el uso de la etiqueta @Transactional (aún poniendo la etiqueta, no funcionará), dejando persistidos en base de datos los objetos creados.

3.2. Clases SpringClassRule y SpringMethodRule (v. 4.2.0.RELEASE)

A partir de esta versión de Spring, el framework da soporte mediante etiquetas de JUnit @ClassRule y @Rule para cargar su contexto a nivel de clase de forma alternativa. Lo hace precisamente, como dice en la documentación de las clases necesarias, para dar soporte a otros frameworks que necesitan el uso de runners, de forma que estos puedan usar la etiqueta @RunWith de JUnit. Al cargarse el contexto a nivel de clase, y no de test como hacíamos en el apartado anterior, sí que se permite el uso de la etiqueta @Transactional, por lo que optamos por dejar implementada esta opción.

Para hacerlo basta con añadir a la clase AbstractTest un par de atributos públicos y finales inicializados con constructores vacíos: uno estático de la clase SpringClassRule y otro de la clase SpringMethodRule, con etiquetas de JUnit @ClassRule y @Rule respectivamente.

También tenemos que actualizar nuestra versión de Spring. Si solo actualizamos el artefacto Spring-test no funcionará por un problema de compatibilidad, ya que las clases mencionadas se apoyan en otras que en nuestra versión del artefacto Spring-context no estaban aún implementadas. Por tanto tenemos que actualizar nuestro Spring-context también a la versión 4.2.0.RELEASE.

4. Implementación de test con Parameterized

Una vez hecho lo anterior, lo que queda por hacer es muy parecido a la implementación propuesta en clase. La estructura que se seguía en la plantilla se desplegará dentro del @Test, ya que en una misma clase no vamos a poder probar más de un caso de test parametrizado. Los parámetros que queramos pasarle a nuestro test lo definiremos como atributos de la clase, pero con la etiqueta @Parameter. Esta etiqueta cuenta con un parámetro "value" que sirve para definir que posición va a ocupar el atributo que estamos definiendo en el array que se le pasará al test. Si no lo definimos toma por defecto "value = 0".

Una vez hemos definido todos los atributos, debemos definir un método que vaya inyectando los valores de cada atributo en cada test que estamos usando como parámetros. Para esto hay que definir un método con la etiqueta @Parameters. Este método debe ser público y estático, según dice su documentación, pero no dice nada acerca de que debe devolver. Intuimos que debe ser un iterable que contenga un array de objetos, siendo el array de igual tamaño que el número de atributos definidos con @Parameter. En todos los ejemplos que nos hemos encontrado se usa un Collection<Object[]>. Nosotros hemos implementado este

método para que devuelva también este Collection. En este método se definen todos los casos de test a probar, cambiando los valores de los parámetros. Para ellos simplemente añadimos distintos arrays a la colección que se va a devolver, respetando el orden que hemos impuesto por los value de los @Parameter. El tamaño de la colección devuelta será el número de test que se probarán.

La etiqueta @Parameters cuenta además con un parámetro opcional llamado "name", que sirve para definir un identificador a cada test concreto que se está lanzando. Podríamos pasar un primer atributo, por ejemplo, con el nombre del test concreto que estamos probando y usar "name=0" para que se mostrara este String que estamos usando. En nuestro caso lo hemos dejado con su valor por defecto, que es index, un parámetro especial que usa el framework para ir numerando de 0 hasta (n-1) todos los test realizados, siendo n el tamaño del Collection comentado anteriormente.

5. Otras consideraciones

En los test que definimos en el proyecto base, sin usar esta herramienta de JUnit, incluíamos varios casos de uso en una misma clase de test. Esto ya no es posible con la nueva implementación. Cada caso de uso parametrizado debe ir en una clase distinta, ya que hace uso de los atributos de la clase para ir inyectando los parámetros a probar. No podemos definir unos cuantos atributos para que se inyecten en un @Test y otros para que se inyecten en un segundo @Test, y aun si lo intentaramos (definiendo, por ejemplo, a partir de cierto número del array para el segundo caso de uso), no podemos definir dos Collections de parametros para probar, tendríamos que definir en un mismo array los parámetros para un test y para el otro, teniendo mezclados los test. En definitiva, sería una mala práctica y la solución es tan fácil como la que hemos comentado, definir una clase de test para cada caso de test parametrizado.

No se puede ejecutar un test por separado en esta versión de eclipse. Para ejecutar o debugear un test en concreto tendríamos el mismo problema que teníamos en la implementación sin Parameterized: tendríamos que ejecutarlos todos e ir saltándolos hasta llegar al que nos interesa. Esto se ha corregido en la versión Mars M4 release de eclipse, siendo un problema del IDE y no de la librería JUnit.

6. Bibliografía

https://github.com/junit-team/junit4/wiki/parameterized-tests

http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/test/context/TestContextManager.html

http://stackoverflow.com/a/28561473

https://raymondhlee.wordpress.com/2016/02/13/writing-parameterized-tests-with-spring-and-junit-4/

https://www.javadoc.io/doc/org.springframework/spring-test/4.2.0.RELEASE

http://stackoverflow.com/questions/23789271/java-lang-nosuchmethoderror-org-springframework-core-annotation-annotationutils