Spring + DBunit

What is dbunit?

When developing integration test, we like the db in the same state every time the test are run, so the result of the test can be predictable. DBUnit is the framework to store the data from xml files. The collection of data is called dataset. The structure like below:

<?xml version='1.0' encoding='UTF-8'?>

<dataset>

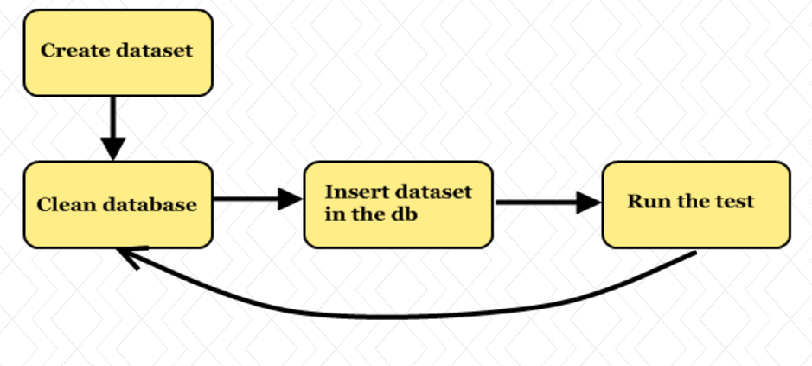
<PackagingUnit id="0" description="Pallet" />

<PackagingUnit id="1" description="Master Case” />

<PackagingUnit id="2" description="Reel” />

</dataset>

It’s quite straightforward, inside every dataset component there’s an element that represents a row in the db table. The name of the element indicates the name of the table, the attributes represent the name of the columns and the value of the attributes represent the data of that column. Inside a file there can be only a dataset, but inside a dataset there can be multiple tables, so there is all the data that can be needed for a test. Each test class would better have its own dataset files. Once defined what a dataset we can see the workflow of a test using DBUnit:



Spring just simplified the DBUnit usage. Spring DBUnit provides integration between the Spring testing framework and the popular DBUnit project. It allows you to setup and teardown database tables using simple annotations as well as checking expected table contents once a test completes.

DBUnit is wired with Spring test framework by using a Spring TestExecutionListener. To have spring process dbunit annotations you must

1. Configure your test to use the DbUnitTestExceutionListener class.

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration

@TestExecutionListeners({ DependencyInjectionTestExecutionListener.class,

DirtiesContextTestExecutionListener.class,//optional

TransactionalTestExecutionListener.class, //optional

DbUnitTestExecutionListener.class })

1. Configure db connection for DBUnit

In order to access the database, Spring DBUnit requires a bean to be registered in your test context. By default a bean named dbUnitDatabaseConnection or dataSource can be used (see the Advanced Configuration section below if you need to use another name). The bean can reference either a IDatabaseConnection or more typically a standard Java DataSource. Here is a typical XML configuration for accessing an in-memory hypersonic database:

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="org.hsqldb.jdbcDriver" />

<property name="url" value="jdbc:hsqldb:mem:paging" />

<property name="username" value="sa" />

<property name="password" value="" />

</bean>

Once we have configured DbUnitTestExecutionListener and provided the bean to access the database we can use the DBUnit annotation like @DatabaseSetup @DatabaseTearDown etc.

DBUnit annotations

@DatabaseSetup annotation indicates how db tables should be setup before test methods are run. The annotation can be on method or class level. When applied at the class level the setup occurs before EACH method in the test. The annotation value references a file that contains the table DataSet used when resetting the database.

@DatabaseTearDown has the similar feature as @DatabaseSetup

@ExpectedDatabase can be used to verify the contents of database once a test has completed eg.

@ExpectedDatabase(“expectedData.xml”) //the dataset file used to verify results. The @ExpectedDatabase annotation supports two different modes: 1. DatabaseAssertionMode.DEFAULT (completed compare of the expected and actual dataset) 2.DatabaseAssertionMode.NON\_STRICT (ignore tables and columns which are not specified in the expected dataset but exist in the actual datasets. It is useful during integration tests when we care about only the ‘interesting’ data)/

@Transaction

When we like to test transaction in the test, we must wire TransactionalTestExecutionListener with spring framework.

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration

@Transactional

@TestExecutionListeners({ DependencyInjectionTestExecutionListener.class,

DirtiesContextTestExecutionListener.class,

TransactionDbUnitTestExecutionListener.class })

Once transaction is wired, Transactions start before @DatabaseSetup and end after @DatabaseTearDown and @ExpectedDatabase.

================================================

Advanced DBUnit configuration (OPTIONAL)

Reference: <https://springtestdbunit.github.io/spring-test-dbunit/>

The @DbUnitConfiguration annotation can be used if you need to configure advanced options for DBUnit.

The databaseConnection attribute allows you to specify a specific bean name from the Spring Context that contains the database connection. When not specified the names or can be used. The bean must be either an IDatabaseConnection or aDataSource.

The dataSetLoader or dataSetLoaderBean attribute allows you to specify a custom loader that will be used when reading datasets (see below). If no specific loader is specified a dbUnitDataSetLoader bean will be used from theApplicationContext (or if no such bean exists, the FlatXmlDataSetLoader will be used).

The databaseOperationLookup attribute allows you to specify a custom lookup strategy for DBUnit database operations (see below).

**Working with multiple connections**

It is possible to configure Spring Test DBUnit to work with multiple connections within the same test. First declare multipleDataSource or IDatabaseConnection beans in your application context. For example, here is XML configuration for two in-memory databases:

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="org.hsqldb.jdbcDriver" />

<property name="url" value="jdbc:hsqldb:mem:paging" />

<property name="username" value="sa" />

<property name="password" value="" />

</bean>

<bean id="customerDataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="org.hsqldb.jdbcDriver" />

<property name="url" value="jdbc:hsqldb:mem:customers" />

<property name="username" value="sa" />

<property name="password" value="" />

</bean>

You then need to use the @DbUnitConfiguration on your test to link to the connections:

@DbUnitConfiguration(databaseConnection={"dataSource", "customerDataSource"})

The @DatabaseSetup, @DatabaseTearDown and @ExpectedDatabase annotations all have a connection attribute which can be used if you need to target a specific connection. If you don’t specify a connection the first databaseConnectionfrom @DbUnitConfiguration will be used (“dataSource” in the example above).

Spring Test DBUnit Annotations are repeatable so if you are using Java 8+ you can use several with the same test. For example:

@Test

@DatabaseSetup(value = "insert.xml")

@DatabaseSetup(connection="customerDataSource", value="insert-custs.xml")

public void testInsert() throws Exception {

// Inserts "insert.xml" into dataSource and "insert-custs.xml" into customerDataSource

// ...

}

If you are using an earlier version of Java you will need to use one of the wrapper annotations:

@Test

@DatabaseSetups({

@DatabaseSetup(value = "insert.xml")

@DatabaseSetup(connection="customerDataSource", value="insert-custs.xml")

})

public void testInsert() throws Exception {

// Inserts "insert.xml" into dataSource and "insert-custs.xml" into customerDataSource

// ...

}

**Custom IDatabaseConnections**

In some situations you may need to create an IDatabaseConnection with a specific DBUnit configuration. Unfortunately, the standard DBUnit DatabaseConfig class cannot be easily using with Spring. In order to overcome this limitation, theDatabaseConfigBean provides an alternative method to configure a connection; with standard getter/setter access provided for all configuration options. The DatabaseDataSourceConnectionFactoryBean accepts a configuration property and should be used to construct the final connection. Here is a typical example:

<bean id="dbUnitDatabaseConfig" class="com.github.springtestdbunit.bean.DatabaseConfigBean">

<property name="skipOracleRecyclebinTables" value="true"/>

</bean>

<bean id="dbUnitDatabaseConnection" class="com.github.springtestdbunit.bean.DatabaseDataSourceConnectionFactoryBean">

<property name="databaseConfig" ref="dbUnitDatabaseConfig"/>

</bean>

NOTE: In most circumstances the username and password properties should not be set on theDatabaseDataSourceConnectionFactoryBean. These properties will cause DBUnit to start a new transaction and may cause unexpected behavior.

**Writing a DataSet Loader**

By default DBUnit datasets are loaded from flat XML files. If you need to load data from another source you will need to write your own DataSet loader and configure your tests to use it. Custom loaders must implement the DataSetLoader interface and provide an implementation of the loadDataSet method. The AbstractDataSetLoader is also available and provides a convenient base class for most loaders.

Here is an example loader that reads data from a CSV formatted file.

public class CsvDataSetLoader extends AbstractDataSetLoader {

protected IDataSet createDataSet(Resource resource) throws Exception {

return new CsvURLDataSet(resource.getURL());

}

}

See above for details of how to configure a test class to use the loader.

**Custom DBUnit Database Operations**

In some situations you may need to use custom DBUnit DatabaseOperation classes. For example, DBUnit includesorg.dbunit.ext.mssql.InsertIdentityOperation for use with Microsoft SQL Server. The DatabaseOperationLookupinterface can be used to create your own lookup strategy if you need support custom operations. AMicrosoftSqlDatabaseOperationLookup class is provided to support the aforementioned MSSQL operations.

See above for details of how to configure a test class to use the custom lookup.