Configuração do Spring Data JPA

Site: Moodle institucional da UTFPR
Curso: CETEJ34 - Sistemas De Banco De Dados - JAVA_XIX (2020_01)
Livro: Configuração do Spring Data JPA

Impresso por: Alexandre Tiago Ximenes
Data: sábado, 22 ago 2020, 06:48

Sumário

1. Configuração do Spring Data JPA

O Spring Data JPA é um recurso que precisa ser configurado junto ao Spring framework para que possa ser usado em um projeto. Essa con figuração não é complicada, mas necessária, porque a partir dela são adicionadas as informações de conexão com o banco de dados, qual provedor vai ser usado, entre algumas mais que veremos a frente.

Outra parte importante são as bibliotecas que o Spring Data precisa para trabalhar. Existem algumas formas diferentes de adicionar estas bibliotecas em um projeto Java. Eu particula mente prefiro usar um geren iador de dependências como o Maven e, será desta forma que vou apresentá-las no decorrer deste curso.

A primeira dependência ou biblioteca, que será apresentada é a do próprio Spring Data JPA. Neste momento a versão utilizada será **2.1.15.RELEASE,** conforme descrito a seguir:

```
<!-- Spring Data JPA -->
<dependency>
   <groupId>org.springframework.data</groupId>
   <artifactId>spring-data-jpa</artifactId>
   <version>2.1.5.RELEASE</version>
</dependency>
```

A dependência acima vai fornecer todas as classes e interfaces que são necessárias para lidar com o Spring Data JPA. Porém, é preciso definir qual provedor será utilizado.

Neste ponto, *você* pode escolher qualquer framework ORM que siga a especificação JPA. Particularmente, eu gosto muito de trabalhar com o Hibernate, então as dependênciasnecessárias são as seguintes:

```
<!-- Hibernate Core -->
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>5.4.10.Final</version>
</dependency>
```

```
<!-- Hibernate JPA -->
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-entitymanager</artifactId>
   <version>5.4.10.Final</version>
</dependency>
```

Temos aqui o **hi bernate-core** que vai entregar todos os recursos do Hibernate Framework ORM. Já o **hibernate-entitymanager** é responsável por fornecer os recursos de integração entre o Hibernate e a especificação JPA.

Com estas três dependências já é possível colocar o SpringData JPA para funcionar em um projeto. Faltaria aqui, pelo menos mais duas dependências, o driver de conexão com o banco de dados escolhido para seu projeto e um pool de conexões, caso necessite de um. Então, como exemplo, serão incluídas as dependências do MySQL Connector *(driver)* e do BoneCP*(conneciot npool)*.

```
<!-- MySQL Doatabase -->
<dependency>
   <groupId>Mysql</groupId>
   <artifactId>Mysql-connector-java</artifactId>
   <version>8.0.19</version>
</dependency>
```

```
<!-- BoneCP -->
<dependency>
   <groupId>com.jolbox</groupId>
   <artifactId>bonecp</artifactId>
   <version>0.8.0.RELEASE</version>
</dependency>
```

1.1. Configuração Programática

No Spring Framework é possível, desde a versão 3.0, configurar recursos via código Java e anotações. Este modelo de configuração pode ser chamado de programático, enquanto a forma mais antiga de configuração, anterior a versão 3.0, era realizada especificamente por meio de arquivos do tipo XML.

Embora o uso de XML ainda seja possível nas atuais versões do Spring, a abordagem explorada neste curso será totalmente por meio de configuração programática.

Desta forma, uma classe que contém a configuração programática do Spring Data JPA precisa mais ou menosde três métodos para estar concluída.

Além disso, no topo da classe é necessário adicionar anotações responsáveis por habilitar alguns recursos do Spring Framework e do Spring Data JPA. Então, observe a **Listagem 2.1** que contém a assinatura da classe de configuração e algumas anotações.

LISTAGEM 2.1: CLASSE OE CONFIGURAÇÃO SPRING DATA CONFIG

```
package com.curso.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableJpaRepositories("com.curso.repository")
@EnableTransactionManagement
public class SpringDataConfig{

// métodos de configurações serão adicionados aqui.
}
```

Analisando o código da classe **SpringDataConfig,** note que existem três anotações sobre a assinatura desta classe. Cada uma destas anotações tem umobjetivo bem específico que é:

- **@Configuration** - é u ma anotaçãodo Spring Framework que marca a classe como sendo uma classe de configuração. Este tipo de classe será uma das primeiras a ser inicializada pelo Spring para liberar os recursos que estão configurados nela;
- **@EnableJpaRepositories** - esta anotação tem dois objetivos. O primeiro é habilitar, ou dizer ao Spring Framework, que os recursos do Spring Data JPA serão usados em tal projeto . O segundo é informar em qual pacote da aplicação estarão as interfaces que vão representar os repositórios do SpringData JPA, ou seja, a camada de persistência de seu projeto;
- **@EnableTransactionManagement** - a função desta anotação é habilitar no Spring o controle transacional com o banco de dados. Deste modo, o Spring é quem será o responsável em lidar com as transações;
- **@Bean** - embora esta anotação ainda não tenha si do apresentada no código, ela será muito importante. Sua função é transformar um método, presente em uma classe, em um *bean* gerenciado pelo Spring Framework. Isto é importante para o processo de injeção de dependências e inversão de controle do framework.

A partir de agora, serão apresentados, individualmente, os métodos que fazem parte da classe SpringDataConfig. O primeiro recurso que deve ser configurado é o DataSource, o qual fornece a fonte de dados para o Spring **(Listagem 2.2).**

LISTAGEM 2.2: MÉTODO DATASOURCE()

```
@Bean
public DataSource dataSource() {

   BoneCPDataSource ds = new BoneCPDataSource();

   ds.setUser("root");
   ds.setPassword("root");
   ds.setJdbcUrl("jdbc:mysql://localhost/curso-spring-db");
   ds.setDriverClass("com.mysql.jdbc.Driver");
   return ds;
}
```

Veja que método **dataSource()** está configurado com as informações básicas de uma conexão com um banco de dados. Estas informações são especificamente o usuário e senha de acesso, mais a URL e a classe do driver de conexão. Observe que no topo da assinatura do método foi incluída a anotação **@Bean**, para torná-lo um objeto gerenciado pelo Spring.

Para criar o recurso do DataSource foi usada a classe **com.jolbox.bonecp.BoneCPDataSource,** a qual é proveniente do driver de conexão BoneCP, referente a uma das dependências do Maven que foi apresentada anteriormente.

Por fim, o método tem um objeto javax.sql.DataSource retornado. Este objeto será importante para criar o recurso da JPA.

Seguindo em frente, o próximo mé todo · **Listagem 2.3** • é o responsável por criar a fábrica de gerenciamento de entidades via JPA . Esse será o recurso que vai adicionar o Hibernate como sendo o framework ORM utilizado.

LISTAGEM 2.3: MÉTODO ENTITYMANAGERFACTORY()

```
@Bean
public EntityManagerFactory entityManagerFactory() {

   LocalContainerEntityManagerFactoryBean factory =
      new LocalContainerEntityManagerFactoryBean();

   HibernateJpaVendorAdapter vendorAdapter =
      new HibernateJpaVendorAdapter();
   vendorAdapter.setGenerateDdl(false);
   vendorAdapter.setShowSql(true);

   factory.setDataSource(dataSource());
   factory.setJpaVendorAdapter(vendorAdapter);
   factory.setPackagesToScan("com.curso.entity");
   factory.afterPropertiesSet();

   return factory.getObject();
}
```

Analisando o método **entityManagerFactory() ,** veja que há uma instancia de **LocalContainerEntityManagerFactoryBean.** Esta classe faz par te do Spring Framework e, pode ser encontrada no pacote **org.springframework.orm.jpa.**

A partir de seus métodos são configurados os recursos necessários par o Spring Data lidar com o JPA. O primeiro deles é a instrução para vincular o DataSource com a EntityManagerFactory, neste casoo **set DataSource() .**

Logo após, é informado o provedor que vai ser aplicado a este recurso via método **setJpaVendorAdapter() .** Nesse caso, a instancia da classe **HibernateJpaVendorAdapter** realiza este trabalho, a qual faz parte do pacote **org.springframework.orm.jpa.vendor.** Esta instancia fornece o acesso aos métodos **setGenerateDdl()** e **setShowSql().**

Respectivamente, o primeiro foi configurado como false, o que significa que o esquema do banco de dados não será gerado pelo Hibernate. Altere para true se existir essa necessidade.

Já o segundo método, é o responsável por escrever no log as instruções SQL geradas pelo Hibernate. Caso seja false, essas instruções são omitidas do log.

Já o método **setPackagesToScan()**, recebe como parâmetro o pacote da aplicação que contém os mapeamentos do tipo objeto-relacional. Essa informação é essencial para o Hibernate junto a JPA saber onde se encontram os mapeamentos entre as classes de entidades e as tabelas do banco de dados. Caso não seja adicionada ou o local esteja errado, uma exceção será lançada.

Ainda há o método **afterPropertiesSet() ,** acessado a par tir da variável **factory** a ser analisado. Este método é necessário para que o **EntityManagerFactory** só seja criado após todas as configurações terem sido carregadas, caso contrário, uma exceção será lançada.

E por fim, o objeto **factory** retorna um objeto do tipo **javax.persistence.EntityManagerFactory** via método **getObject().**

Caso outras propriedades, vinculadas ao Hibernate, precisem ser incluídas à configuração por qualquer motivo adicional, use os métodos **setJpaProperties()** ou **setJpaPropertyMap()** para esta finalidade.

O último método a ser analisado é o **transactionManager()** presente na **Listagem 2.4.** Nele serão configurados dois recursos a partir da instancia da classe **JpaTransactionManager**, encontrada no pacote **org.springframework.orm.jpa**.

LISTAGEM 2.4: MÉTODO TRANSACTIONM ANAGER()

```
@Bean
public PlatformTransactionManager transactioMnanager() {

   JpaTransactionManager manager = new JpaTransactionManager();
   manager.setEntityManagerFactory(entityManagerFactory());
   manager.setJpaDialect(new HibernateJpaDialect());

return manager;
}
```

Se você está lembrado, no topo da classe **SpringDataConfig** foi adicionada uma anotação que passa para o Spring o controle transacional das operações com o banco de dados.

Agora, é necessário informar o que será gerenciado, ou seja, o objeto que vai lidar com as interações no banco de dados. Este objeto é o retorno do recurso criado no método **entityManagerFactory()**, o qual deve ser adicionado com o parâmetro de **setEntityManagerFactory()** da variável **manager.**

Já o **setJpaDialect() ,** acessível via **manager,** indica ao controle transacional qual o dialeto do banco de dados que ele está lidando. A instancia da classe **HibernateJpaDialect** identifica automaticamente este dialeto a partir do banco de dados configurado.

Com os três métodos apresentados, mais as anotações no topo da classe, a configuração programática do Spring Data JPA está concluída.