03

Preparando Spring com JPA

Transcrição

Nossa aplicação ainda não salva os produtos no banco de dados. Para essa tarefa usaremos a **JPA** (*Java Persistence API*) e o **Hibernate**, que ainda não estão configurados em nosso projeto. Vamos configura-los agora.

No `pom.xml` vamos declarar algumas novas dependências. Entre elas estão a **JPA**, o **Hibernate**, O **SpringORM** e o **Driver MySQL**. No final do `pom.xml` antes do fechamento da tag `<dependencies>` cole as seguintes dependencias:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.0.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.0.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>1.0.0.Final</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>4.1.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.15</version>
</dependency>
```

Salve o `pom.xml` e aguarde um momento, pois o Maven irá baixar e deixar disponivel em nosso projeto as bibliotecas que acabamos de adicionar como dependências.

Com as dependências configuradas, podemos começar a criar a lógica responsável por salvar os produtos. O primeiro passo é definir que o produto é uma entidade. Fazemos isso marcando a classe `Produto` com a anotação `@Entity` .

```
import javax.persistence.Entity;

@Entity
public class Produto {
    [...]
}
```

Atenção: O importe deve ser do pacote: `javax.persistence.Entity` .

Com este passo a classe `Produto` já representa uma entidade em nosso sistema. Fazer isso é apenas o primeiro passo, ainda não temos a lógica responsável por efetivamente salvar o produto no banco de dados.

Vamos então criar uma classe de acesso a dados responsável por manipular os dados dos produtos. Criaremos então a classe ProdutoDAO (DAO: Data Access Object ou Objeto de Acesso a Dados). Inicialmente, criamos nesta classe o método `gravar` que receberá um objeto `produto` e o salvará no banco de dados. Está classe deve ficar no pacote: `br.com.casadocodigo.loja.daos`

```
package br.com.casadocodigo.loja.daos;
import br.com.casadocodigo.loja.models.Produto;

public class ProdutoDAO {

    public void gravar(Produto produto){

    }
}
```

Para que o `ProdutoDAO` realize a **persistencia** ou seja, para que ele salve o **produto** no banco de dados. É necessário que ele tenha um gerenciador de entidades, um `EntityManager` . Este `EntityManager` é fornecido pelo `Spring` . Assim podemos usar o `EntityManager` para persistir os produto no banco de dados. No código teremos algo como:

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import br.com.casadocodigo.loja.models.Produto;

public class ProdutoDAO {

    @PersistenceContext
    private EntityManager manager;

    public void gravar(Produto produto){
        manager.persist(produto);
    }

}
```

Temos quase tudo pronto neste ponto. Precisamos fazer com que agora, quando o **Controller** receba o `produto` , ele use o `ProdutoDAO` para salvar o produto no banco de dados. Estas modificações serão feitas no `ProdutosController` .

```
@Controller
public class ProdutosController {

    @Autowired
    private ProdutoDAO produtoDao;

    @RequestMapping("/produtos")
    public String gravar(Produto produto){
        System.out.println(produto);
```

```
        produtoDao.gravar(produto);
        return "/produtos/ok";
    }
    [...]
}
```

A anotação `@Autowired` serve para que nós não nos preocupemos em criar manualmente o `ProdutoDAO` no **Controller**. O **Spring** fará isso automaticamente. Mas para isso, o **Spring** precisa "conhecer" o `ProdutoDAO`. Em outras palavras dizemos que devemos definir que o `ProdutoDAO` será gerenciado pelo **Spring**. Para isso devemos marcar o `ProdutoDAO` com a anotação `@Repository`.

```
@Repository
public class ProdutoDAO {

    @PersistenceContext
    private EntityManager manager;

    public void gravar(Produto produto){
        manager.persist(produto);
    }

}
```

Se tentarmos inicializar o projeto neste momento, teremos dois problemas. O primeiro deles será que, apesar de termos aparentemente configurado todo o necessário para persistir os produtos no banco de dados, **Spring** não conseguirá gerenciar nossas classes, nem mesmo encontrá-las.

Esta configuração está presente em nossa classe `AppWebConfiguration`, na qual configuramos para o **Spring** encontrar nossos *controllers*. Nós vamos configurar para que encontre nossos **daos** também. A anotação `@ComponentScan` deve ficar assim:

```
ComponentScan(basePackageClasses={HomeController.class, ProdutoDAO.class})
```

Note que em momento algum estamos fornecendo para o **Spring** qual é o banco, o usuário ou a senha do banco de dados. Faremos essas configurações em uma nova classe. Crie uma classe no pacote `br.com.casadocodigo.conf` chamada `JPAConfiguration`.

Nesta nova classe, criaremos o método que será gerenciado pelo **Spring** e criará o `EntityManager` usado em nosso **DAO**. Ela também terá as configurações de banco de dados e algumas outras propriedades importantes. Vejamos o código:

```
package br.com.casadocodigo.loja.conf;

import java.util.Properties;

import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaVendorAdapter;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

public class JPAConfiguration {
```

```
    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
        LocalContainerEntityManagerFactoryBean factoryBean = new LocalContainerEntityManagerFact
        JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();

        factoryBean.setJpaVendorAdapter(vendorAdapter);

        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setUsername("root");
        dataSource.setPassword("");
        dataSource.setUrl("jdbc:mysql://localhost:3306/casadocodigo");
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");

        factoryBean.setDataSource(dataSource);

        Properties props = new Properties();
        props.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");
        props.setProperty("hibernate.show_sql", "true");
        props.setProperty("hibernate.hbm2ddl.auto", "update");

        factoryBean.setJpaProperties(props);

        factoryBean.setPackagesToScan("br.com.casadocodigo.loja.models");

        return factoryBean;
    }
}
```

Nesta classe estamos criando um único método, que será usado pelo **Spring** para gerar o **EntityManager**. Este precisa de um **adapter** e estamos passando um que o **Hibernate** disponibiliza.

Criamos também um **DataSource** que contém as configurações de banco de dados. Criamos um objeto do tipo **Properties** para podermos setar algumas configurações, como por exemplo o dialeto usado para a comunicação com o banco de dados. Setamos também onde o **EntityManager** encontrará nossos **Models**.

Feito isso retornamos nossas configurações para o **Spring** poder utiliza-las.