Aula 21 - Criando CRUDs com Hibernate

Docupedia Export

Author:Silio Leonardo (SO/OPM-TS21-BR)
Date:25-Oct-2024 14:41

Table of Contents

- Criando um projeto com Hibernate usando Maven
- Usando SQL Server
- Problemas que podemos encontrar
- Usando o Hibernate

1 Criando um projeto com Hibernate usando Maven

Vamos criar um projeto que conecta com o banco de dados usando Hibernate. Essa técnologia irá gerenciar nosso banco de dados criando tabelas para nós. Usando JavaFX já instalado, Hibernate com SqlServer e JUnit, você pode configurar sua aplicação com o seguinte pom.xml:

pom.xml

```xml
<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>org.openjfx</groupId>
    <artifactId>sample</artifactId>
    <version>1.0.0</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>22</maven.compiler.source>
        <maven.compiler.target>22</maven.compiler.target>
    </properties>

    <dependencyManagement>

        <dependencies>
            <dependency>
                <groupId>org.junit</groupId>
                <artifactId>junit-bom</artifactId>
                <version>5.11.2</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>

    </dependencyManagement>

    <dependencies>

        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
```

```
32              <scope>test</scope>
33          </dependency>
34
35          <dependency>
36              <groupId>org.openjfx</groupId>
37              <artifactId>javafx-controls</artifactId>
38              <version>22.0.1</version>
39          </dependency>
40
41          <dependency>
42              <groupId>org.openjfx</groupId>
43              <artifactId>javafx-fxml</artifactId>
44              <version>22.0.1</version>
45          </dependency>
46
47          <dependency>
48              <groupId>jakarta.persistence</groupId>
49              <artifactId>jakarta.persistence-api</artifactId>
50              <version>3.1.0</version>
51          </dependency>
52
53          <dependency>
54              <groupId>org.hibernate.orm</groupId>
55              <artifactId>hibernate-core</artifactId>
56              <version>6.2.0.Final</version>
57          </dependency>
58
59          <dependency>
60              <groupId>com.microsoft.sqlserver</groupId>
61              <artifactId>mssql-jdbc</artifactId>
62              <version>12.8.1.jre11</version>
63          </dependency>
64
65          <dependency>
66              <groupId>com.h2database</groupId>
67              <artifactId>h2</artifactId>
68              <version>2.3.232</version>
69          </dependency>
70
```

```
    </dependencies>

    <build>

        <plugins>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.10.1</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.8</version>
                <configuration>
                    <mainClass>com.desktopapp.App</mainClass>
                </configuration>
            </plugin>

            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.5.0</version>
            </plugin>

        </plugins>

    </build>

</project>
```

Além disso há um arquivo próprio de configuração do Hibernate que fica na pasta resources. Observe com cuidado suas configurações:

src/main/resources/META-INF/persistence.xml

```xml
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
             version="2.2">
    <persistence-unit name="my-persistence-unit">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <class>com.desktopapp.model.User</class>
        <properties>
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:test;DB_CLOSE_DELAY=-1"/>
            <property name="javax.persistence.jdbc.user" value="sa"/>
            <property name="javax.persistence.jdbc.password" value=""/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

Dentro do seu projeto na pasta 'desktopapp', onde está a Main.java, criaremos também uma pasta model para colocar nossas classes que representarão nossas tabelas. Observe a criação de uma tabela de usuários:

src/main/java/com/desktopapp/model/UserData,java

```java
package com.desktopapp.model;

import jakarta.persistence.Id;
import jakarta.persistence.Table;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;

@Entity
@Table(name = "tbUser")
```

```java
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }

    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    private String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

}
```

Note que usamos várias notações especiais para indicar como o Hibernate deve criar a tabela e se comunicar com o banco de dados. Aqui estamos usando Im Memory Database, isso significa que os dados são salvos na memória o que é bom para testes. Abaixo uma classe de configuração para facilitar nosso uso do Hibernate.

src/main/java/com/desktopapp/Context,java

```java

package com.desktopapp;

import java.util.*;
```

```
4
5  import jakarta.persistence.EntityManager;
6  import jakarta.persistence.EntityManagerFactory;
7  import jakarta.persistence.Persistence;
8
9  public class Context {
10     private EntityManagerFactory emf;
11     private EntityManager em;
12
13     public Context() {
14         emf = Persistence.createEntityManagerFactory("my-persistence-unit");
15     }
16
17     public void begin() {
18         em = emf.createEntityManager();
19         try {
20             em.getTransaction().begin();
21         } catch (Exception e) {
22             if (em.getTransaction().isActive()) {
23                 em.getTransaction().rollback();
24             }
25             e.printStackTrace();
26             em = null;
27         }
28     }
29
30     public <T> List<T> find(Class<T> entytyClass, String query, Object... values)
31     {
32         EntityManager em = emf.createEntityManager();
33         List<T> users = null;
34         try {
35             var queryObj = em.createQuery(query, entytyClass);
36             for (Integer i = 0; i < values.length; i++) {
37                 queryObj = queryObj.setParameter("arg" + i.toString(), values[i]);
38             }
39             users = queryObj.getResultList();
40         } finally {
41             em.close();
42         }
```

```
43          return users;
44
45      }
46
47      public <T> T find(Class<T> entityClass, Object primaryKey) {
48          EntityManager em = emf.createEntityManager();
49          T user = null;
50          try {
51              user = em.find(entityClass, primaryKey);
52          } finally {
53              em.close();
54          }
55          return user;
56      }
57
58      public void save(Object object) {
59          if (em == null) {
60              System.out.println("connection is null.");
61              return;
62          }
63
64          try {
65              em.persist(object);
66          } catch (Exception e) {
67              if (em.getTransaction().isActive()) {
68                  em.getTransaction().rollback();
69              }
70              e.printStackTrace();
71              em = null;
72          }
73      }
74
75      public void commit() {
76          if (em == null) {
77              System.out.println("connection is null.");
78              return;
79          }
80
81          try {
```

```
82              em.getTransaction().commit();
83          } catch (Exception e) {
84              if (em.getTransaction().isActive()) {
85                  em.getTransaction().rollback();
86              }
87              e.printStackTrace();
88          } finally {
89              em.close();
90              em = null;
91          }
92      }
93  }
```

2 Usando SQL Server

1. Baixe o JDBC na última versão: https://learn.microsoft.com/pt-br/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver16
2. Adicione a seguinte dependência no pom.xml.
    a. **pom.xml**

```xml
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <version>12.8.1.jre11</version>
</dependency>
```

3. Ubicación en el sistema: Copia el archivo `sqljdbc_auth.dll` en una de las siguientes ubicaciones:
    a. En la carpeta `bin` de tu instalación de JDK (por ejemplo, `C:\Program Files\Java\jdk-xx\bin`).
4. Ajuste o persistence.xml da seguinte forma:
    a. **persistence.xml**

```xml
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
             version="2.2">
    <persistence-unit name="my-persistence-unit">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <class>com.desktopapp.USer</class>
        <properties>
            <property name="javax.persistence.jdbc.driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"
/>
            <property name="javax.persistence.jdbc.url" value="jdbc:sqlserver://
localhost:1433;databaseName=JavaFXHibernate;integratedSecurity=true;trustServerCertificate=true"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.SQLServerDialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
```

```
        </properties>
    </persistence-unit>
</persistence>
```

3 Problemas que podemos encontrar

Caso você não tenha o JDBC (super comum) você pode buscar sua última versão aqui: https://learn.microsoft.com/pt-br/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver16.
Caso você encontre o erro "This driver is not configured for integrated authentication", você pode, dentro do JDBC procurar o arquivo na pasta auth/x64 (a depender da sua arquitetura) e copiá-lo nas pastas **bin** e **lib** do Java Runtime Enviroment. Para achar essa pasta mais facilmente, procure na busca do windows por Java, clique com botão direito na resposta e então em "Abrir no Local do Arquivo", você entrará na pasta bin do JRE.
Muitas vezes erros estão relacionados ao **SQL Server Browser** que precisa estar rodando na aplicação. Procurando o aplicativo **SQL Server Configuration Manager** você pode iniciá-lo. Contudo, as vezes é necessário abrir o app **Serviços** do windows e procurar o **SQL Server Browser** ou o **SQL Server Agent** e mudar sua forma de inicialização de Desabilitado para Automático ou Manual. Muitas vezes é necessário abrir esses softwares como administrador para ter acesso as opções.
Por fim, você pode ter problema com portas TCP. Também no **SQL Server Configuration Manager** em configuração de rede do sql server > Protocolos do ***server > TCP/IP deve estar habilitado, além disso, abrindo a opção em Endereços IP todos os Ips devem estar ativos e habilitados.

4 Usando o Hibernate

Observe o uso do Hibernate e seu estilo de queries para fazer um Login funcional:

src/main/java/com/desktopapp/Authentification.java

```
1  package com.desktopapp;
2
3  import java.net.URL;
4
5  import com.desktopapp.model.User;
6
7  import javafx.event.ActionEvent;
8  import javafx.fxml.FXML;
9  import javafx.fxml.FXMLLoader;
10 import javafx.scene.Scene;
11 import javafx.scene.Parent;
12 import javafx.scene.control.Alert;
13 import javafx.scene.control.Alert.AlertType;
14 import javafx.scene.control.Button;
15 import javafx.scene.control.CheckBox;
16 import javafx.scene.control.TextField;
17 import javafx.scene.control.ButtonType;
18 import javafx.scene.control.PasswordField;
19 import javafx.stage.Stage;
20
21 public class LoginSceneController {
22
23     public static Scene CreateScene() throws Exception {
24         URL sceneUrl = LoginSceneController.class
25                 .getResource("login-scene.fxml");
26         Parent root = FXMLLoader.load(sceneUrl);
27         Scene scene = new Scene(root);
28         return scene;
29     }
30
31     @FXML
32     protected Button btLogin;
```

```
33
34      @FXML
35      protected TextField tfLogin;
36
37      @FXML
38      protected PasswordField pfPass;
39
40      @FXML
41      protected CheckBox cbPass;
42
43      @FXML
44      protected void submit(ActionEvent e) throws Exception {
45
46          Context ctx = new Context();
47          var users = ctx.find(User.class,
48              "SELECT u FROM User u WHERE u.name = :arg0",
49              tfLogin.getText()
50          );
51
52          if (users.size() == 0) {
53              Alert alert = new Alert(
54                  AlertType.ERROR,
55                  "Usuário não está cadastrado!",
56                  ButtonType.OK
57              );
58              alert.showAndWait();
59              return;
60          }
61          var user = users.get(0);
62
63          if (!pfPass.getText().equals(user.getPassword())) {
64              Alert alert = new Alert(
65                  AlertType.ERROR,
66                  "Senha incorreta!",
67                  ButtonType.OK
68              );
69              alert.showAndWait();
70              return;
71          }
```

```
72
73          var crrStage = (Stage)btLogin
74              .getScene().getWindow();
75          crrStage.close();
76
77          var stage = new Stage();
78          var scene = MainSceneController.CreateScene();
79          stage.setScene(scene);
80          stage.show();
81      }
82
83  }
```

Uma coisa que devemos tomar cuidado com o Sql Injection. O que acontece se o usuário digitado na tela de login for " or 1=1? Observe a query construída: **from UserData u where u.username = " or 1=1**. Note que você conseguirá acessar um usuário mesmo sem saber seu username. Por isso vamos usar parâmetros para evitar esse tipo de ataque. No nosso caso o ataque parece infantil já que é necessário saber a senha, contudo, se verificarmos a senha e login ao mesmo tempo, SQL Injection poderia invadir a aplicação por alguma conta qualquer (normalmente a conta mais antiga, possivelmente um ADM). Para resolver isso basta usar a seguinte implementação:

src/main/java/com/desktopapp/Authentification.java

```
1           Query query = session.createQuery(
2               "from UserData u where u.username = :user"
3           );
4           query.setParameter("user", user);
5           List<UserData> users = query.list();
```