Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova

Centrul de Excelență în Informatică și Tehnologii Informaționale

SPECIALITATEA

«Programare și analiza produselor de program»

Индивидуальная работа

По предмету: Tehn. moderne de progr.

Тема: Spring Data JPA

Преподаватель:

Cerbu O.

Подготовил:

Ambrosii Maxim

Группа: P – 1642R

Chişinău, 2019

**Spring Data JPA**

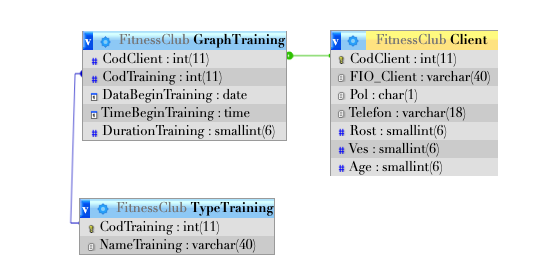
Миссией Spring DATA является предоставление единой модели программирования с использованием Spring для доступа к данным, сохраняя при этом специальные черты базового хранилища.

Фреймворк позволяет облегчить использование технологий доступа к данным, реляционных и не реляционных баз данных, облачных баз данных. Spring DATA — базовый проект, который включает множество других подпроектов, которые работают с конкретными базами данных.

**Проект Spring Data JPA**

1. Для начала, создаем Maven project, для добавления нужных Dependencies в файл pom.xml, а так-же все будет производиться на примере базы данных Fitness Club.

**Fitness Clud База Данных**



**File pom.xml**

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

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>SpringDataJPA</groupId>

<artifactId>SpringDataJPA</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.15</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.4.2.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa -->

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-jpa</artifactId>

<version>2.0.13.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>5.4.2.Final</version>

</dependency>

</dependencies>

</project>

1. Следующим шагом будет создание в папке resources файла applicationContext.xml для того, чтобы начать работу с самими framework Spring Data JPA

**File applicationContext.xml**

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

<!-- beans нужен для подключения нужных элементов Spring Framework с сервера (Конечно же для запуска понадобится интернет)-->

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:security="http://www.springframework.org/schema/security"

xmlns:tx="http://www.springframework.org/schema/tx" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:jpa="http://www.springframework.org/schema/data/jpa"

xsi:schemaLocation="

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa-1.0.xsd

http://www.springframework.org/schema/security ">

<!-- Directory to scan for repository classes -->

<jpa:repositories base-package="repository"/>

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName">

<!-- Здесь мы указываем какой драйвер для подключения к базе данных мы будем использовать -->

<value>com.mysql.jdbc.Driver</value>

</property>

<property name="url">

<!-- Здесь мы указываем полную ссылку к базе данных -->

<value>jdbc:mysql://192.168.64.2/FitnessClub?serverTimezone=UTC</value>

</property>

<property name="username">

<!-- Для того чтобы работать с базой данных мы так же предоставляем доступ программе к ней, поэтому указываем логин, в нашем случае это логин от Phpmyadmin, у меня это MaxDB, у вас скорее всего это будет root -->

<value>MaxDB</value>

</property>

<property name="password">

<!-- Так же пароль, у вас это будет просто пустое место -->

<value>localhostdb</value>

</property>

</bean>

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="persistenceUnitName" value="jpaData"/>

<property name="jpaVendorAdapter">

<bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>

</property>

<property name="jpaProperties">

<!-- Здесь мы начинаем перечислять все наши настройки -->

<props>

<!-- Подключаем hibernate dialect чтоб программа понимала, что связь будет через hibernate -->

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<!-- Так как мы работаем с Sql базой данных, нам соответственно нужно это показать, что и делает данная функция -->

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.format\_sql">false</prop>

<!-- Следующая строчка гораздо важней, для того чтобы мы работали уже с созданной базой данных мы указываем update, то есть после каждого выполнения мы будем только обновлять нашу базу данных, с Spring Data JPA Framework мы можем создать базу данных с нуля, а для этого указываем Create-->

<prop key="hibernate.hbm2ddl.auto">update</prop>

</props>

</property>

</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory"/>

</bean>

</beans>

P.S Поподробнее о каждой строчке в комментариях кода

1. Для того чтобы сам Framework знал, с какой базой данных мы работаем, мы создаем для каждой таблицы отдельный класс, конечно же так как у нас JPA, мы будем работать с таблицами как с Объектами данных классов, чтобы было удобнее создаем отдельный Package для всех классов с описанием таблиц, назовем его entity, так как все классы идентичны, я прокомментирую один из них, остальные по стандарту.

**File GraphTraining.java**

package entity;

import java.sql.Date; // Подключаем для работы с Датой в Sql

import java.sql.Time; // Подключаем для работы с Временем в Sql

import javax.persistence.\*; // Так как у нас JPA, подключаем persistence в наш класс

@Table(name = "GraphTraining") // Здесь мы указываем аннотацию, что данный класс будет таблицей, а так же указываем название таблицы

public class GraphTraining{

@ManyToMany // Данная анотация указывает на связь Многие ко Многим с другой таблицей Client

private Long CodClient; // Так мы обьявляем атрибуты private Тип данных атрибута название

@ManyToMany // Данная анотация указывает на связь Многие ко Многим с другой таблицей TypeTraining

private Long CodTraining;

@Column(name = "DataBeginTraining") // Данная анотация указывает на атрибут, то есть каждая колонка, это отдельный атрибут таблицы, а также указываем имя атрибута

private Date DataBeginTraining;

@Column(name = "TimeBeginTraining")

private Time TimeBeginTraining;

@Column(name = "DurationTraining")

private Long DurationTraining;

public GraphTraining() { // Здесь мы создаем конструктор по умолчанию

}

public Long getCodClient() {

return CodClient;

} // Создаем getters and setters, для того чтобы мы могли получать и передавать значения из таблиц

public void setCodClient(Long codClient) {

CodClient = codClient;

}

public Long getCodTraining() {

return CodTraining;

}

public void setCodTraining(Long codTraining) {

CodTraining = codTraining;

}

public Date getDataBeginTraining() {

return DataBeginTraining;

}

public void setDataBeginTraining(Date dataBeginTraining) {

DataBeginTraining = dataBeginTraining;

}

public Time getTimeBeginTraining() {

return TimeBeginTraining;

}

public void setTimeBeginTraining(Time timeBeginTraining) {

TimeBeginTraining = timeBeginTraining;

}

public Long getDurationTraining() {

return DurationTraining;

}

public void setDurationTraining(Long durationTraining) {

DurationTraining = durationTraining;

}

@Override // Здесь мы переопределяем метод ToString, для удобного вывода на экран всех данных

public String toString() {

return "GraphTraining [CodClient=" + CodClient + ", CodTraining=" + CodTraining + ", DataBeginTraining="

+ DataBeginTraining + ", TimeBeginTraining=" + TimeBeginTraining + ", DurationTraining="

+ DurationTraining + "]";

}

}

**File Client.java**

package entity;

import javax.persistence.\*;

@Entity // Данная аннотация указывает на то, что в таблице есть первичный ключ

@Table(name = "Client")

public class Client{

@Id // Здесь мы указываем что данный атрибут таблицы является первичным ключем

@GeneratedValue(strategy = GenerationType.IDENTITY) // Для того чтобы не заполнять ID, мы делаем авто генерацию ID

private Long CodClient;

@Column(name = "FIO\_Client")

private String FIO\_Client;

@Column(name = "Pol")

private String Pol;

@Column(name = "Telefon")

private String Telefon;

@Column(name = "Rost")

private Long Rost;

@Column(name = "Ves")

private Long Ves;

@Column(name = "Age")

private Long Age;

public Client() {

}

public Long getCodClient() {

return CodClient;

}

public void setCodClient(Long codClient) {

CodClient = codClient;

}

public String getFIO\_Client() {

return FIO\_Client;

}

public void setFIO\_Client(String fIO\_Client) {

FIO\_Client = fIO\_Client;

}

public String getPol() {

return Pol;

}

public void setPol(String pol) {

Pol = pol;

}

public String getTelefon() {

return Telefon;

}

public void setTelefon(String telefon) {

Telefon = telefon;

}

public Long getRost() {

return Rost;

}

public void setRost(Long rost) {

Rost = rost;

}

public Long getVes() {

return Ves;

}

public void setVes(Long ves) {

Ves = ves;

}

public Long getAge() {

return Age;

}

public void setAge(Long age) {

Age = age;

}

@Override

public String toString() {

return "Client [CodClient=" + CodClient + ", FIO\_Client=" + FIO\_Client + ", Pol=" + Pol + ", Telefon=" + Telefon

+ ", Rost=" + Rost + ", Ves=" + Ves + ", Age=" + Age + "]";

}

}

**File TypeTraining.java**

package entity;

import javax.persistence.\*;

@Entity

@Table(name = "TypeTraining")

public class TypeTraining{

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long CodTraining;

@Column(name = "NameTraining")

private String NameTraining;

public TypeTraining() {

}

public Long getCodTraining() {

return CodTraining;

}

public void setCodTraining(Long codTraining) {

CodTraining = codTraining;

}

public String getNameTraining() {

return NameTraining;

}

public void setNameTraining(String nameTraining) {

NameTraining = nameTraining;

}

@Override

public String toString() {

return "TypeTraining [CodTraining=" + CodTraining + ", NameTraining=" + NameTraining + "]";

}

}

1. Теперь мы создаем отдельный Package для интерфейсов, Интерфейсы нужны нам, чтобы установить связь между таблицами и обьявить их как repositories.

**File ClientReposotory.java**

package repository;  
  
import entity.Client;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface ClientRepository extends JpaRepository<Client, Long>{  
  
}

**File GraphTrainigRepository.java**

package repository;  
  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface GraphTrainingRepository{  
  
}

**File TypeTrainingRepository.java**

package repository;  
  
import entity.TypeTraining;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface TypeTrainingRepository extends JpaRepository <TypeTraining, Long>{  
  
}

На этом мы можем закончить, выполнение всех манипуляций для работы с Spring Data JPA, ну а собственно для того чтобы работать со всем этим, создаем класс Domain и попробуем поработать с базой данных.

**File Domain.java**

package domain;

import java.sql.SQLException;

import org.springframework.context.support.ClassPathXmlApplicationContext; // Добавляем библиотеку в класс

import entity.Client; // Добавляем нужные созданные ранее вспомогательные классы

import entity.TypeTraining;

import repository.ClientRepository;

import repository.TypeTrainingRepository;

public class Domain {

public static void main(String[] args) throws SQLException{

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml"); // Создаем связь с нашим файлом, для того чтобы программа взяла все нужные данные для подключения к базе данных

TypeTrainingRepository typeTrainingRepos = context.getBean(TypeTrainingRepository.class); // Создаем обьект класса Type Training c соответствующим подключением к файлам bean

ClientRepository clientRepos = context.getBean(ClientRepository.class); // Создаем обьект класса ClientRepository c соответствующим подключением к файлам bean, чтобы обьявить все из интерфейса

TypeTraining type\_tr = new TypeTraining(); // Создаем обьект класса typeTraining

type\_tr.setNameTraining("Test Spring"); // Добавляем новую тренировку в таблицу Type Training

Client client = new Client(); // Создаем обьект класса Client

client.setFIO\_Client("Test Test Test"); // Добавляем информацию о новом клиенте в таблицу базы данных

client.setPol("M");

client.setTelefon("078123456");

client.setRost(165L);

client.setVes(65L);

client.setAge(25L);

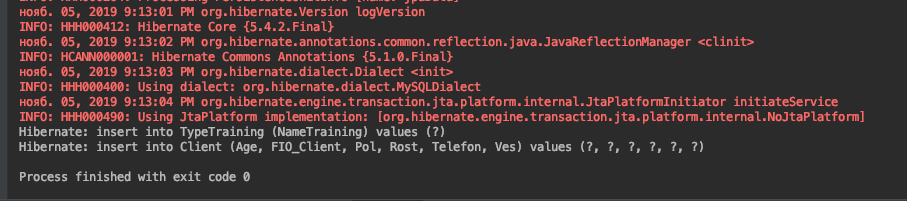
typeTrainingRepos.save(type\_tr); //Сохраняем все изменения

clientRepos.save(client); // И отправляем их в базу данных

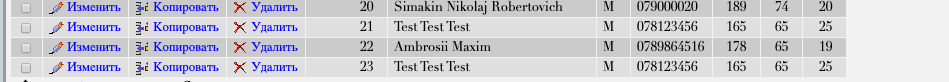
}

}

Давайте посмотрим что у нас вывело в консоль(не стоит пугаться красного цвета, здесь он не всегда сигнализирует о плохом):



Как мы видим вместо соответствующих знаков вопроса, подставляются введенные данные, что существенно облегляет нам работу с базой данных. Давайте зайдем в базу данных и проверим, добавился ли такой клиент



Здесь мы видим, да действительно все сработало успешно, и наш клиент Test был успешно добавлен в базу данных.

Вывод: Spring Data — дополнительный удобный механизм для взаимодействия с сущностями базы данных, организации их в репозитории, извлечение данных, изменение, в каких то случаях для этого будет достаточно объявить интерфейс и метод в нем, без имплементации. Что в разы упрощает работу программисту.