**스프링 DB 접근 기술**

스프링 데이터 엑세스

* H2 데이터베이스 설치
* 순수 Jdbc
* 스프링 통합 테스트
* 스프링 JdbcTemplate
* JPA
* 스프링 데이터 JPA

순서  
1. 순수 JDBC 만을 이용해서 DB Connection, CRUD  
2. JDBC Template 의 라이브러리를 이용해서 더 간편하게  
3. JPA를 이용해서 더 간편하게  
4. 스프링 데이터 JPA를 이용해서 더 간편하게

H2 데이터베이스는 개발이나 테스트 용도로 가볍고 편리한 DB, 웹화면도 제공하기 때문에 설치

테이블 생성하기  
테이블 관리를 위해 프로젝트 루트에 sql/ddl.sql 파일을 생성

drop table if exists member CASCADE;

create table member

(

id bigint generated by default as identity,

name varchar(255),

primary key (id)

);

H2 데이터베이스에 접근해서 member 테이블 생성

**순수 Jdbc**  
환경설정  
build.gradle 파일에 jdbc, h2 데이터베이스 관련 라이브러리 추가

implementation 'org.springframework.boot:spring-boot-starter-jdbc'

runtimeOnly 'com.h2database:h2'

스프링 부트 데이터베이스 연결 설정 추가

spring.datasource.url=jdbc:h2:tcp://localhost/~/test

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

**주의,** 스프링부트 2.4 부터는 spring.datasource.username=sa를 꼭 추가해주어야 한다. 그렇지 않으면 오류가 발생한다.

**Jdbc 회원 리포지토리**

package hello.hellospring.repository;

import hello.hellospring.domain.Member;

import org.springframework.jdbc.datasource.DataSourceUtils;

import javax.sql.DataSource;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

public class JdbcMemberRepository implements MemberRepository {

private final DataSource dataSource;

public JdbcMemberRepository(DataSource dataSource) {

this.dataSource = dataSource;

}

@Override

public Member save(Member member) {

String sql = "insert into member(name) values(?)";

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

conn = getConnection();

pstmt = conn.prepareStatement(sql,

Statement.RETURN\_GENERATED\_KEYS);

pstmt.setString(1, member.getName());

pstmt.executeUpdate();

rs = pstmt.getGeneratedKeys();

if (rs.next()) {

member.setId(rs.getLong(1));

} else {

throw new SQLException("id 조회 실패");

}

return member;

} catch (Exception e) {

throw new IllegalStateException(e);

} finally {

close(conn, pstmt, rs);

}

}

@Override

public Optional<Member> findById(Long id) {

String sql = "select \* from member where id = ?";

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

conn = getConnection();

pstmt = conn.prepareStatement(sql);

pstmt.setLong(1, id);

rs = pstmt.executeQuery();

if(rs.next()) {

Member member = new Member();

member.setId(rs.getLong("id"));

member.setName(rs.getString("name"));

return Optional.of(member);

} else {

return Optional.empty();

}

} catch (Exception e) {

throw new IllegalStateException(e);

} finally {

close(conn, pstmt, rs);

} }

@Override

public List<Member> findAll() {

String sql = "select \* from member";

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

conn = getConnection();

pstmt = conn.prepareStatement(sql);

rs = pstmt.executeQuery();

List<Member> members = new ArrayList<>();

while(rs.next()) {

Member member = new Member();

member.setId(rs.getLong("id"));

member.setName(rs.getString("name"));

members.add(member);

}

return members;

} catch (Exception e) {

throw new IllegalStateException(e);

} finally {

close(conn, pstmt, rs);

}

}

@Override

public Optional<Member> findByName(String name) {

String sql = "select \* from member where name = ?";

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

conn = getConnection();

pstmt = conn.prepareStatement(sql);

pstmt.setString(1, name);

rs = pstmt.executeQuery();

if(rs.next()) {

Member member = new Member();

member.setId(rs.getLong("id"));

member.setName(rs.getString("name"));

return Optional.of(member);

}

return Optional.empty();

} catch (Exception e) {

throw new IllegalStateException(e);

} finally {

close(conn, pstmt, rs);

}

}

private Connection getConnection() {

return DataSourceUtils.getConnection(dataSource);

}

private void close(Connection conn, PreparedStatement pstmt, ResultSet rs)

{

try {

if (rs != null) {

rs.close();

}

} catch (SQLException e) {

e.printStackTrace();

} try {

if (pstmt != null) {

pstmt.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

try {

if (conn != null) {

close(conn);

}

} catch (SQLException e) {

e.printStackTrace();

} }

private void close(Connection conn) throws SQLException {

DataSourceUtils.releaseConnection(conn, dataSource);

} }

스프링 설정변경

package hello.hellospring;

import hello.hellospring.repository.JdbcMemberRepository;

import hello.hellospring.repository.JdbcTemplateMemberRepository;

import hello.hellospring.repository.MemberRepository;

import hello.hellospring.repository.MemoryMemberRepository;

import hello.hellospring.service.MemberService;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;

@Configuration

public class SpringConfig {

private final DataSource dataSource;

public SpringConfig(DataSource dataSource) {

this.dataSource = dataSource;

}

@Bean

public MemberService memberService() {

return new MemberService(memberRepository());

}

@Bean

public MemberRepository memberRepository() {

// return new MemoryMemberRepository();

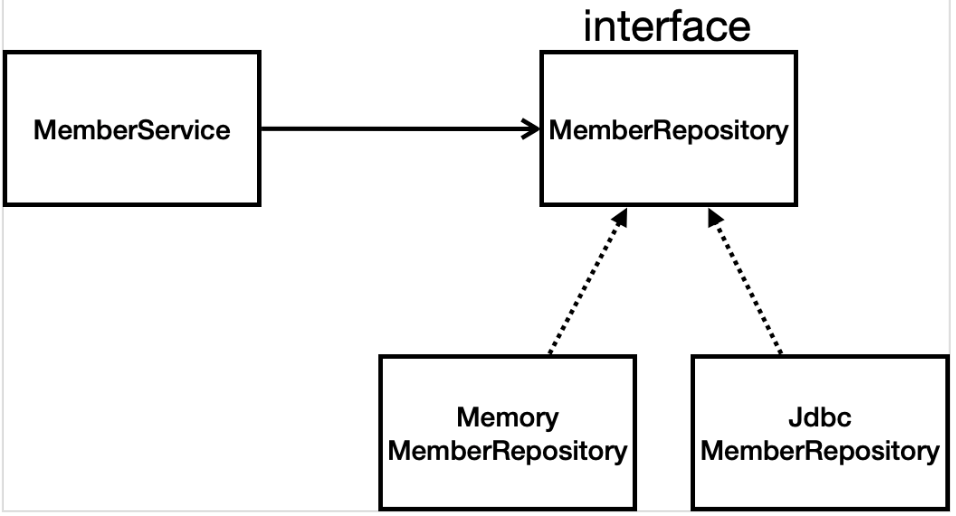
return new JdbcMemberRepository(dataSource);

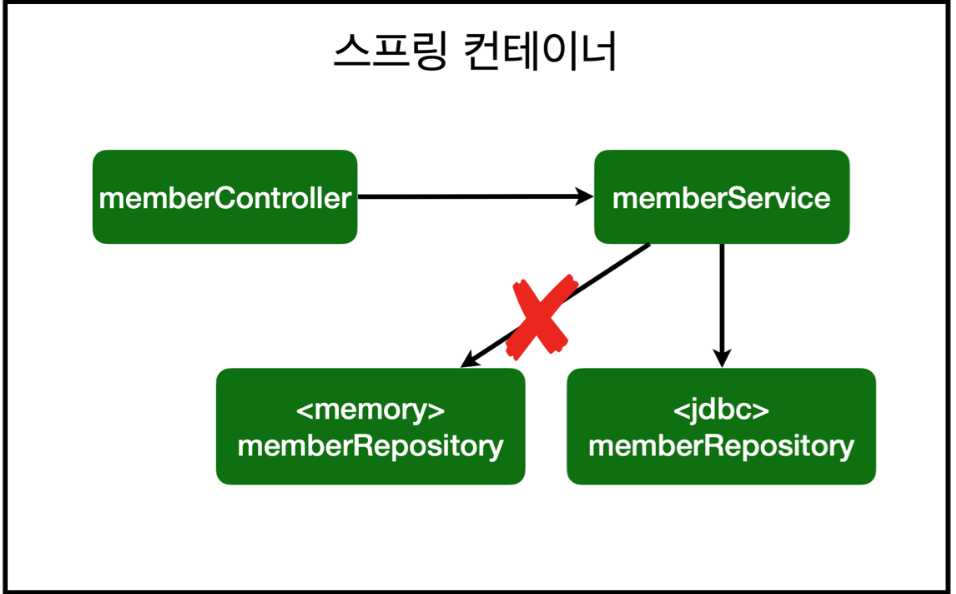
}

}

* DataSource는 데이터베이스 커넥션을 획들할 떄 사용하는 객체다. 스프링 부트는 데이터베이스 커넥션 정보를 바탕으로 DataSource를 생성하고 스프링 빈으로 만들어 둔다. 그래서 DI를 받을 수 있다.

즉, 메모리DB를 이용한 구현체를 주석하고, 순수 JDBC를 이용한 H2DB를 사용하기 위한 변화

구현 클래스 추가  


스프링 설정  


* 개방-폐쇄 원칙(OCP, Open-Closed Principle)  
  - 확장에는 열려있고, 변경에는 닫혀있다.
* 스프링의 DI를 사용하면 기존 코드를 전혀 손대지 않고, 설정만으로 구현클래스를 변경할 수 있다.

**스프링 통합 테스트**  
스프링 컨테이너와 DB까지 연결한 통합 테스트

회원 서비스 스프링 통합 테스트

package hello.hellospring.service;

import hello.hellospring.domain.Member;

import hello.hellospring.repository.MemberRepository;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.transaction.annotation.Transactional;

import static org.assertj.core.api.Assertions.assertThat;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.junit.jupiter.api.Assertions.assertThrows;

@SpringBootTest

@Transactional

class MemberServiceIntegrationTest {

@Autowired MemberService memberService;

@Autowired MemberRepository memberRepository;

@Test

public void 회원가입() throws Exception {

//Given

Member member = new Member();

member.setName("hello");

//When

Long saveId = memberService.join(member);

//Then

Member findMember = memberRepository.findById(saveId).get();

assertEquals(member.getName(), findMember.getName());

}

@Test

public void 중복\_회원\_예외() throws Exception {

//Given

Member member1 = new Member();

member1.setName("spring");

Member member2 = new Member();

member2.setName("spring");

//When

memberService.join(member1);

IllegalStateException e = assertThrows(IllegalStateException.class,

() -> memberService.join(member2));//예외가 발생해야 한다. assertThat(e.getMessage()).isEqualTo("이미 존재하는 회원입니다.");

} }

* @SpringBootTest: 스프링 컨테이너와 테스트를 함께 실행한다.
* @Transactional: 테스트 케이스에 이 에노테이션이 있으면, 테스트 시작 전에 트랜잭션을 시작하고, 테스트 완료 후에 항상 롤백한다. 이렇게 하면 DB에 데이터가 남지 않으므로 다음 테스트에 영향을 주지 않는다.