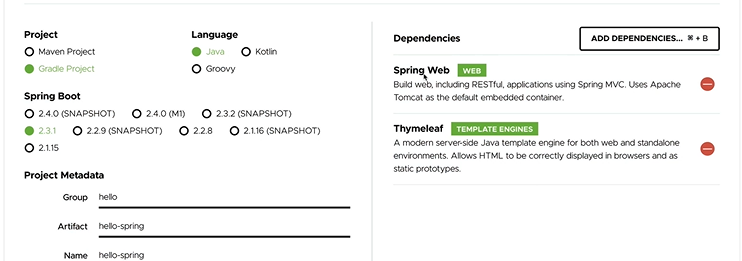
//1-5

Spring를 하기 위한 준비물 (Java 11, intellij / eclipse)

프로젝트 생성

요즘은 스프링부트를 통하여 프로젝트를 만드는 것이 더 쉽기 때문에 start.spring.io를 통해서 프로젝트를 생성한다. 다음과 같이 설정한다.



생성하면 zip파일이 나오는데 압축을 풀고 intellij에서 import를 한다.

Import -> Existing Gradle Project -> 생성 폴더 -> 생성된 프로젝트를 확인한다.

//2-1

프로젝트를 실행시키면 console에 정보가 나오며 chrome 주소에 localhost:8080을 입력하게 되면 에러페이지가 나온다면 성공적으로 한 것 이다.

Resources -> static -> index.html생성 후 내용을 입력하면 홈페이지에 hello가 출력된다.

<!DOCTYPE HTML>  
<html>  
<head>  
 <title>Hello</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<body>  
Hello  
<a href="/hello">hello</a>  
</body>  
</html>

Main -> java -> hellospring -> package(controller)생성 -> javaclass(HelloController) 생성 -> 입력

@Controller  
public class HelloController {  
  
 @GetMapping("hello")  
 public String hello(Model model) {  
 model.addAttribute("data", "hello!!");  
 return "hello";  
 }  
}

Resources -> templates -> hello.html 생성 -> 입력

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Hello</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<body>  
<p th:text="'안녕하세요. ' + ${data}" >안녕하세요. 손님</p>  
</body>  
</html>

정적 컨텐츠 -> 파일을 그대로 웹 브라우저에 전달해주는 것

MVC와 템플릿엔진 -> 서버에서 변형 후 html을 바꿔서 내려주는 방식

API->객체반환 / json이라는 구조포맷을 통하여 클라이언트에 전달하는 방식, 서버끼리 통신할 때

스프링 부트 정적 컨텐츠

Resources -> static -> hello-static.html (생성) / <http://localhost:8080/hello-static.html>를 입력하면 바로 확인이 가능하다.

<!DOCTYPE HTML>  
<html>  
<head>  
 <title>static content</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<body>  
정적 컨텐츠 입니다.  
</body>  
</html>

MVC : Model, View, Controller

Controller 추가

@GetMapping("hello-mvc")  
public String helloMvc(@RequestParam("name") String name, Model model) {  
 model.addAttribute("name", name);  
 return "hello-template";  
}

View

Resources -> template -> hello-template.html 생성

<html xmlns:th="http://www.thymeleaf.org">  
<body>  
<p th:text="'hello ' + ${name}">hello! empty</p>  
</body>  
</html>

// hello! Empty 는 없어도 되는 부분이다

view-source:/ 경로를 입력하면 서버 연결 없이 확인 했을 때 hello! Empty가 출력된다.

실행 -> http://localhost:8080/hello-mvc?name=(spring, 아무거나)

API

Controller 추가

@GetMapping("hello-string")  
@ResponseBody // http의 body부에 데이터를 직접 넣어주겠다.  
public String helloString(@RequestParam("name") String name){  
 return "hello " + name; //"hello spring"  
}

실행 : <http://localhost:8080/hello-string?name=spring> / 페이지 소스를 보면 Html태그가 없음

Controller 추가

@GetMapping("hello-api")  
@ResponseBody  
public Hello helloApi(@RequestParam("name") String name) {  
 Hello hello = new Hello();  
 hello.setName(name);  
 return hello;  
}  
  
 static class Hello {  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
 }  
}

실행 -> <http://localhost:8080/hello-api?name=spring> / 출력은 비슷하지만 json방식으로 출력됨

Json = key, value로 이루어진 단순한 구조

회원 도메인과 저장소 만들기

회원 객체

package hellospring.hellospring.domain;  
  
public class Member {  
  
 private Long id;  
 private String name;  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

회원 저장소 인터페이스

package hellospring.hellospring.repository;  
  
public interface MemberRepository {  
 Member save(Member member);  
 Optional<Member> findById(Long id);  
 Optional<Member> findByName(String name);  
 List<Member> findAll();  
}

회원 저장소 메모리 구현체

package hellospring.hellospring.repository;

public class MemoryMemberRepository implements MemberRepository{  
  
 private static Map<Long, Member> *store* = new HashMap<>();  
 private static long *sequence* = 8L;  
  
 @Override  
 public Member save(Member member) {  
 member.setId(++*sequence*);  
 *store*.put(member.getId(), member);  
 return member;  
 }  
  
 @Override  
 public Optional<Member> findById(Long id) {  
 return Optional.*ofNullable*(*store*.get(id));  
 }  
  
 @Override  
 public Optional<Member> findByName(String name) {  
 return *store*.values().stream()  
 .filter(member -> member.getName().equals(name))  
 .findAny();  
 }  
  
 @Override  
 public List<Member> findAll() {  
 return new ArrayList<>(*store*.values());  
 }  
  
 public void clearStore() {  
 *store*.clear();  
 }  
}

이것들을 검증하기 위해서 test를 작성하는 것이다.

Test -> repository(package)생성 -> MemoryMemberRepositoryTest(java class)생성

전체 or 부분으로 실행하면 확인이 가능하다

public class MemroyMemberRepositoryTest {  
  
 MemoryMemberRepository repository = new MemoryMemberRepository();  
  
 @AfterEach  
 public void afterEach() {  
 repository.clearStore();  
 }//실행 후 repository초기화  
 @Test  
 public void save(){  
 Member member = new Member();  
 member.setName("spring");  
  
 repository.save(member);  
  
 Member result = repository.findById(member.getId()).get();  
 //Assertions.assertEquals(member, result);  
 *assertThat*(member).isEqualTo(result);  
 }  
  
 @Test  
 public void fineByName(){  
 Member member1 = new Member();  
 member1.setName("spring1");  
 repository.save(member1);  
  
 Member member2 = new Member();  
 member2.setName("spring2");  
 repository.save(member2);  
  
 Member result = repository.findByName("spring1").get();  
  
 *assertThat*(result).isEqualTo(member1);  
 }  
 @Test  
 public void findAll() {  
 //given  
 Member member1 = new Member();  
 member1.setName("spring1");  
 repository.save(member1);  
 Member member2 = new Member();  
 member2.setName("spring2");  
 repository.save(member2);  
 //when  
 List<Member> result = repository.findAll();  
 //then  
 *assertThat*(result.size()).isEqualTo(2);  
 }  
}

회원 서비스 개발

package hellospring.hellospring.service;  
  
import hellospring.hellospring.domain.Member;  
import hellospring.hellospring.repository.MemberRepository;  
import hellospring.hellospring.repository.MemoryMemberRepository;  
  
import java.util.\*;  
  
public class MemberService {  
 private final MemberRepository memberRepository = new MemoryMemberRepository();  
  
 //회원가입  
 public long join(Member member) {  
 //같은 이름이 있는 중복 회원x  
 validateDuplicateMember(member);//중복회원검증  
 memberRepository.save(member);  
 return member.getId();  
 }  
  
 private void validateDuplicateMember(Member member) {  
 memberRepository.findByName(member.getName())  
 .ifPresent(m -> {  
 throw new IllegalStateException("이미 존재하는 회원입니다.");  
 });  
 }  
 //전체 회원 조회  
 public List<Member> fineMembers(){  
 return memberRepository.findAll();  
 }  
  
 public Optional<Member> findOne(Long memberId){  
 return memberRepository.findById((memberId));  
 }  
}

회원 서비스 테스트

기존 회원 서비스 코드 수정

private final MemberRepository memberRepository;  
public MemberService(MemberRepository memberRepository) {  
 this.memberRepository = memberRepository;  
}

Ctrl + Shift + T를 통하여 쉽게 Test파일을 생성할 수 있다.

package hellospring.hellospring.service;  
  
import hellospring.hellospring.domain.Member;  
import hellospring.hellospring.repository.MemoryMemberRepository;  
import org.assertj.core.api.Assertions;  
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.util.Optional;  
  
import static org.assertj.core.api.Assertions.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class MemberServiceTest {  
  
 MemberService memberService;  
 MemoryMemberRepository memoryMemberRepository;  
  
 @BeforeEach  
 public void beforeEach() {  
 memoryMemberRepository = new MemoryMemberRepository();  
 memberService = new MemberService(memoryMemberRepository);  
 }  
  
 @AfterEach  
 public void afterEach() {  
 memoryMemberRepository.clearStore();  
 }  
  
 @Test  
 void 회원가입() {  
 //given  
 Member member = new Member();  
 member.setName("spring");  
  
 //when  
 Long saveId = memberService.join(member);  
 //than  
 Member findMember = memberService.findOne(saveId).get();  
 *assertThat*(member.getName()).isEqualTo(findMember.getName());  
 }  
  
 @Test  
 public void 중복\_회원\_예외() {  
 //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));  
  
 *assertThat*(e.getMessage()).isEqualTo("이미 존재하는 회원입니다.");  
  
// try {  
// memberService.join(member2);  
// fail();  
// } catch (IllegalStateException e) {  
// assertThat(e.getMessage()).isEqualTo("이미 존재하는 회원입니다.");  
// }  
 //than  
  
 }  
}

//2-2

스프링 빈과 의존관계

컴포넌트 스캔과 자동 의존관계 설정

회원 컨트롤러가 회원서비스와 회원저장소를 사용할 수 있게 의존관계를 만든다.

package hellospring.hellospring.controller;  
  
import hellospring.hellospring.service.MemberService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
  
@Controller  
public class MemberController {  
  
 private final MemberService memberService;  
  
 @Autowired  
 public MemberController(MemberService memberService) {  
 this.memberService = memberService;  
 }  
}

생성자에 @Autiwired가 있으면 스프링이 연관된 객체를 스프링 컨테이너에서 찾아서 넣어준다.

스프링 빈을 등록하는 2가지 방법

컴포넌트 스캔과 자동 의존관계 설정, 자바 코드로 직접 스프링 빈 등록하기

컴포넌트 스캔

@Repository  
public class MemoryMemberRepository implements MemberRepository{

@Service  
public class MemberService {  
  
 private final MemberRepository memberRepository;  
  
 @Autowired  
 public MemberService(MemberRepository memberRepository) {  
 this.memberRepository = memberRepository;  
 }

자바 코드로 직접 스프링 빈 등록하기

package hellospring.hellospring;  
  
import hellospring.hellospring.repository.MemberRepository;  
import hellospring.hellospring.repository.MemoryMemberRepository;  
import hellospring.hellospring.service.MemberService;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class SpringConfig {  
  
 @Bean  
 public MemberService memberService() {  
 return new MemberService(memberRepository());  
 }  
  
 @Bean  
 public MemberRepository memberRepository() {  
 return new MemoryMemberRepository();  
 }  
}

회원 관리 예제 / web MVC 개발

1. 홈 화면 추가(컨트롤러&회원 관리용 홈)
2. Controller

package hellospring.hellospring.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
  
@Controller  
public class HomeController {  
  
 @GetMapping("/")  
 public String home() {  
 return "home";  
 }  
}

1. HTML

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<body>  
<div class="container">  
 <form action="/members/new" method="post">  
 <div class="form-group">  
 <label for="name">이름</label>  
 <input type="text" id="name" name="name" placeholder="이름을  
입력하세요">  
 </div>  
 <button type="submit">등록</button>  
 </form>  
</div> <!-- /container -->  
</body>  
</html>

회원 등록 폼 개발

@Controller  
public class MemberController {  
  
 private final MemberService memberService;  
  
 @Autowired  
 public MemberController(MemberService memberService) {  
 this.memberService = memberService;  
 }  
  
 @GetMapping("/members/new")  
 public String createForm() {  
 return "members/createMemberForm";  
 }  
  
 @PostMapping("/members/new")  
 public String create(MemberForm form) {  
 Member member = new Member();  
 member.setName(form.getName());  
  
 memberService.join(member);  
  
 return "redirect:/";  
 }  
}

회원 등록 폼 HTML

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<body>  
<div class="container">  
 <form action="/members/new" method="post">  
 <div class="form-group">  
 <label for="name">이름</label>  
 <input type="text" id="name" name="name" placeholder="이름을  
입력하세요">  
 </div>  
 <button type="submit">등록</button>  
 </form>  
</div> <!-- /container -->  
</body>  
</html>

웹 등록 화면에서 데이터를 전달 받을 폼 객체

package hellospring.hellospring.controller;  
  
public class MemberForm {  
  
 private String name;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

컨트롤러에서 회원을 실제 등록하는 기능

@PostMapping("/members/new")  
public String create(MemberForm form) {  
 Member member = new Member();  
 member.setName(form.getName());  
  
 memberService.join(member);  
  
 return "redirect:/";  
}

Get방식은 조회일 때 자주 사용하며, Post는 데이터 등록할 때 사용.

회원 웹 기능 – 조회

Controller에 추가

@GetMapping("/members")  
public String list(Model model) {  
 List<Member> members = memberService.fineMembers();  
 model.addAttribute("members", members);  
 return "members/membersList";  
}

membersList 폼을 생성 후 입력

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<body>  
<div class="container">  
 <div>  
 <table>  
 <thead>  
 <tr>  
 <th>#</th>  
 <th>이름</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="member : ${members}">  
 <td th:text="${member.id}"></td>  
 <td th:text="${member.name}"></td>  
 </tr>  
 </tbody>  
 </table>  
 </div>  
</div> <!-- /container -->  
</body>  
</html>

H2 데이터베이스 설치

h2database.com에서 다운로드 및 설치

h2Console를 실행 후

jdbc:h2:~/test (최초 한번) 이후 jdbc:h2:tcp://localhost/~/test 이렇게 접속

테이블 생성

drop table if exists member CASCADE;

create table member (

id bigint generated by default as identity, (= 값을 세팅하지않고 입력하면 db가 자동으로 채워줌)

name varchar(255),

primary key (id)

);

순수 JDBC

Build.fradle에 추가

implementation 'org.springframework.boot:spring-boot-starter-jdbc'  
runtimeOnly 'com.h2database:h2'

파일 생성 후 복붙

package hellospring.hellospring.repository;  
  
import hellospring.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 통합 테스트

//Transactional을 통해서 테스트가 끝나면 롤백을 해줌 -> 끝나면 db에 남지않음

@SpringBootTest  
@Transactional  
class MemberServiceIntegrationTest {  
  
 @Autowired MemberService memberService;  
 @Autowired MemberRepository memoryMemberRepository;  
  
 @Test  
 public void 회원가입() throws Exception {  
 //Given  
 Member member = new Member();  
 member.setName("spring");  
 //When  
 Long saveId = memberService.join(member);  
 //Then  
 Member findMember = memoryMemberRepository.findById(saveId).get();  
 *assertEquals*(member.getName(), findMember.getName());  
 }

//2-3

스프링 JdbcTemplate

순수 Jdbc와 동일한 환경설정을 하면 된다.

Jdbc Api에서 본 반복 코드를 대부분 제거해준다.

package hellospring.hellospring.repository;  
  
public class JdbcTemplateMemberRepository implements MemberRepository {  
  
 private final JdbcTemplate jdbcTemplate;  
  
 @Autowired // 생성자가 한개일때 빈으로 등록되면 생략가능  
 public JdbcTemplateMemberRepository(DataSource dataSource) {  
 jdbcTemplate = new JdbcTemplate(dataSource);  
 }  
  
 @Override  
 public Member save(Member member) {  
 SimpleJdbcInsert jdbcInsert = new SimpleJdbcInsert(jdbcTemplate);  
 jdbcInsert.withTableName("member").usingGeneratedKeyColumns("id");  
 Map<String, Object> parameters = new HashMap<>();  
 parameters.put("name", member.getName());  
 Number key = jdbcInsert.executeAndReturnKey(new  
 MapSqlParameterSource(parameters));  
 member.setId(key.longValue());  
 return member;  
 }  
  
 @Override  
 public Optional<Member> findById(Long id) {  
 List<Member> result = jdbcTemplate.query("select \* from member where id = ?", memberRowMapper(), id);  
 return result.stream().findAny();  
 }

@Override  
 public Optional<Member> findByName(String name) {  
 List<Member> result = jdbcTemplate.query("select \* from member where name = ?", memberRowMapper(), name);  
 return result.stream().findAny();  
 }  
  
 @Override  
 public List<Member> findAll() {  
 return jdbcTemplate.query("select \* form member", memberRowMapper());  
 }  
  
 private RowMapper<Member> memberRowMapper() {  
 return (rs, rowNum) -> {  
 Member member = new Member();  
 member.setId(rs.getLong("id"));  
 member.setName(rs.getString("name"));  
 return member;  
 };  
 }  
}

JPA

SQL query도 JPA가 자동으로 처리함

라이브러리 추가

// implementation 'org.springframework.boot:spring-boot-starter-jdbc'  
 implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

스프링부트에 JPA 설정 추가

spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=none

JPA 엔티티 매핑

@Entity  
public class Member {  
  
 @Id @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
// @Column(name = "username")  
 private String name;

JPA 회원 리포지토리

package hellospring.hellospring.repository;  
  
public class JpaMemberRepository implements MemberRepository {  
  
 private final EntityManager entityManager;  
 //jpa는 em으로 동작함  
 public JpaMemberRepository(EntityManager entityManager) {  
 this.entityManager = entityManager;  
 }  
  
 @Override  
 public Member save(Member member) {  
 entityManager.persist(member);  
 return member;  
 }  
  
 @Override  
 public Optional<Member> findById(Long id) {  
 Member member = entityManager.find(Member.class, id);  
 return Optional.*ofNullable*(member);  
 }  
  
 @Override  
 public Optional<Member> findByName(String name) {  
 List<Member> result = entityManager.createQuery("select m from Member m where m.name = :name", Member.class)  
 .setParameter("name", name)  
 .getResultList();  
 return result.stream().findAny();  
 }  
  
 @Override  
 public List<Member> findAll() {  
 return entityManager.createQuery("select m from Member m", Member.class) //테이블x 객체를 대상으로query를 날림  
 .getResultList();  
 }  
}

서비스 계층에 트랜잭션 추가

@Transactional //데이터를 저장하거나 변경할때는 필수  
public class MemberService {

스프링 설정 변경

@Configuration  
public class SpringConfig {  
  
 private EntityManager entityManager;  
 @Autowired  
 public SpringConfig(EntityManager entityManager) {  
 this.entityManager = entityManager;  
 }  
  
 @Bean  
 public MemberService memberService() {  
 return new MemberService(memberRepository());  
 }  
 @Bean  
 public MemberRepository memberRepository() {  
 return new JpaMemberRepository(entityManager);  
 }  
}

스프링 데이터 JPA

Interface로 생성

package hellospring.hellospring.repository;  
  
public interface SpringDataJpaMemberRepository extends JpaRepository<Member, Long>, MemberRepository {  
  
 @Override  
 Optional<Member> findByName(String Name);  
}

스프링 설정 변경

package hellospring.hellospring;

@Configuration  
public class SpringConfig {  
  
 private final MemberRepository memberRepository;  
 @Autowired  
 public SpringConfig(MemberRepository memberRepository) {  
 this.memberRepository = memberRepository;  
 }  
  
 @Bean  
 public MemberService memberService() {  
 return new MemberService(memberRepository);  
 }  
}

AOP(Aspect oriented Programming)

long start = System.*currentTimeMillis*();  
try{  
 validateDuplicateMember(member);//중복회원검증  
 memberRepository.save(member);  
 return member.getId();  
} finally {  
 long finish = System.*currentTimeMillis*();  
 long timeMs = finish - start;  
 System.*out*.println("Join = " + timeMs + "ms");  
}

시간 측정 방법 : 일일이 다 코딩해야해서 귀찮음

package hellospring.hellospring.aop;  
  
@Aspect  
@Component  
public class TimeTraceAop {  
  
 @Around("execution (\* hellospring.hellospring..\*(..))")//적용위치  
 public Object excute(ProceedingJoinPoint joinPoint) throws Throwable {  
 long start = System.*currentTimeMillis*();  
 System.*out*.println("START : " + joinPoint.toLongString());  
 try{  
 return joinPoint.proceed();  
 }finally {  
 long finish = System.*currentTimeMillis*();  
 long timeMs = finish - start;  
 System.*out*.println("END : " + joinPoint.toLongString() + " " + timeMs + "ms");  
 }  
  
 }  
}

@Component 대신 가능

@Bean  
public TimeTraceAop timeTraceAop() {  
 return new TimeTraceAop();  
}

//2-4

Eclipse에서 실습하기

준비물 java 11, Eclipse

예제와 동일하게 start.spring.io에서 프로젝트 생성

eclipse에서 import 후 HelloController 생성, static-> hello.html 생성 및 코딩

이후 예제 순으로 따라하며 큰 에러는 아니지만 시간을 낭비함.

주의)

Templates를 실행하기 위해서 STS툴을 설치하였고(설치하여서 되는건지 애매함) 서버를 사용 후 종료하는 것이 필요함

Import를 하는 과정에서 eclipse에서 알려주는 것으로 하다 보면 잘못된 게 있을 수 있음

//2-5

Eclipse로 다시 하는 중

JdbcMemberRepository를 작성중 DataSourceUtils -> 에러 발생

원인으로 jdbc파일이 없다고 판단.

java.lang.Object

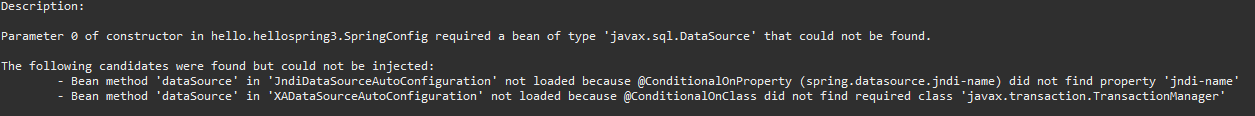
extended by**org.springframework.jdbc.datasource.DataSourceUtils**

<https://jar-download.com/artifacts/org.springframework/spring-jdbc> 다운로드

스프링 jdbc를 다운받고 lib라는 빈 폴더를 생성후 jdbc파일을 넣고

Add JARs…을 클릭 후 jdbc를 추가한다.

에러



해결하기 위해서 트랜잭션을 다운받은 후 적용 에러…

