cf. https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-의존성-주입

**[Spring\_Boot] Controller & Service & Repository/ 의존성 주입**

[**최현석**](https://velog.io/@chlgustjr/posts)·2022년 11월 22일

[🧩 MemberController](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9-membercontroller)

[🧩@Autowired](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9autowired)

[🧩 @Autowired를 사용할 수 있는 위치](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9-autowired%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%A0-%EC%88%98-%EC%9E%88%EB%8A%94-%EC%9C%84%EC%B9%98)

[🧩 제어 역전(IoC, Inversion of Control )](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9-%EC%A0%9C%EC%96%B4-%EC%97%AD%EC%A0%84ioc-inversion-of-control-)

[🧩 스프링 컨테이너란?](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9-%EC%8A%A4%ED%94%84%EB%A7%81-%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EB%9E%80)

[🧩 POJO란?](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9--pojo%EB%9E%80)

[🧩 의존성 주입(DI : Dependency Injection)](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85di--dependency-injection)

[🧩 회원저장 DB 실습](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#%F0%9F%A7%A9-%ED%9A%8C%EC%9B%90%EC%A0%80%EC%9E%A5-db-%EC%8B%A4%EC%8A%B5)

[💊 Member.java](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-memberjava)

[💊 MemberRepository](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-memberrepository)

[💊 JdbcMemberRepository](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-jdbcmemberrepository)

[💊 MemberService](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-memberservice)

[💊 MemberController](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-membercontroller)

[💊 HomeController](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-homecontroller)

[💊 home.html](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-homehtml)

[💊 createMemberForm.html](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-creatememberformhtml)

[💊 MemberFormDTO](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-memberformdto)

[💊 memberList](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-memberlist)

[💊 DB](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-db)

[💊 build gradle](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-build-gradle)

[💊 application.properties](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-applicationproperties)

[💊 결과](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85#-%EA%B2%B0%EA%B3%BC)

[**Spring\_Boot**](https://velog.io/@chlgustjr/series/Spring)

목록 보기 [Spring\_Boot](https://velog.io/@chlgustjr/series/Spring)

1. [[Spring\_Boot] 설치 및 환결설정](https://velog.io/@chlgustjr/SpringBoot-%EC%84%A4%EC%B9%98-%EB%B0%8F-%ED%99%98%EA%B2%B0%EC%84%A4%EC%A0%95)
2. [[Spring\_Boot] Tymeleaf 세팅](https://velog.io/@chlgustjr/SpringBoot-Tymeleaf-%EC%84%B8%ED%8C%85)
3. [[Spring\_Boot]의 기초](https://velog.io/@chlgustjr/Spring%EC%9D%98-%EA%B8%B0%EC%B4%88)
4. [**[Spring\_Boot] Controller & Service & Repository/ 의존성 주입**](https://velog.io/@chlgustjr/SpringBoot-Controller-Service-Repository-%EC%9D%98%EC%A1%B4%EC%84%B1-%EC%A3%BC%EC%9E%85)
5. [[Spring\_Boot] SOLID원칙, JPA](https://velog.io/@chlgustjr/SpringBoot-SOLID%EC%9B%90%EC%B9%99-JPA)
6. [[Spring\_Boot] lombok 라이브러리 설치](https://velog.io/@chlgustjr/SpringBoot-lombok-%EB%9D%BC%EC%9D%B4%EB%B8%8C%EB%9F%AC%EB%A6%AC-%EC%84%A4%EC%B9%98)
7. [[Spring\_Boot] Thymeleaf, 기본문법](https://velog.io/@chlgustjr/SpringBoot-RequestParam-ResponseParam-Thymeleaf-%EC%82%AC%EC%9A%A9)
8. [[Spring\_Boot] RequestParam & ResponseParam/ ResponseBody/ ModelAndView](https://velog.io/@chlgustjr/SpringBoot-RequestParam-ResponseParam)
9. [[Spring\_Boot] 상품 목록,등록,수정,상세페이지 / PathVariable/ ModelAttribute/ RedirectAttributes/ RequiredArgsConstructor](https://velog.io/@chlgustjr/SpringBoot-%EC%83%81%ED%92%88-%EB%AA%A9%EB%A1%9D%EB%93%B1%EB%A1%9D%EC%88%98%EC%A0%95%EC%83%81%EC%84%B8%ED%8E%98%EC%9D%B4%EC%A7%80)
10. [[Spring\_Boot] enum(열거 타입)](https://velog.io/@chlgustjr/SpringBoot-enum%EC%97%B4%EA%B1%B0-%ED%83%80%EC%9E%85)
11. [[Spring\_Boot] 페이지( 판매 여부/ 등록 지역/ 상품 종류/ 배송 방식)버튼](https://velog.io/@chlgustjr/SpringBoot-%ED%8E%98%EC%9D%B4%EC%A7%80-%ED%8C%90%EB%A7%A4-%EC%97%AC%EB%B6%80-%EB%93%B1%EB%A1%9D-%EC%A7%80%EC%97%AD-%EC%83%81%ED%92%88-%EC%A2%85%EB%A5%98-%EB%B0%B0%EC%86%A1-%EB%B0%A9%EC%8B%9D%EB%B2%84%ED%8A%BC)
12. [[Spring\_Boot] 유효성 검사, 오류 메세지](https://velog.io/@chlgustjr/SpringBoot-%EC%9C%A0%ED%9A%A8%EC%84%B1-%EA%B2%80%EC%82%AC-%EC%98%A4%EB%A5%98-%EB%A9%94%EC%84%B8%EC%A7%80)
13. [[Spring\_Boot] 회원가입 로그인/ PostConstruct/ Component/ 메세지](https://velog.io/@chlgustjr/SpringBoot-%ED%9A%8C%EC%9B%90%EA%B0%80%EC%9E%85-%EB%A1%9C%EA%B7%B8%EC%9D%B8)
14. [[Spring\_Boot] 로그인 - 쿠키, 세션 & SessionConst](https://velog.io/@chlgustjr/SpringBoot-%EB%A1%9C%EA%B7%B8%EC%9D%B8-%EC%BF%A0%ED%82%A4-%EC%84%B8%EC%85%98)
15. [[Spring\_Boot] 필터, 인터셉터](https://velog.io/@chlgustjr/SpringBoot-%ED%95%84%ED%84%B0-%EC%9D%B8%ED%84%B0%EC%85%89%ED%84%B0)
16. [[Spring\_Boot] JPA](https://velog.io/@chlgustjr/SpringBoot-JPA)
17. [[Spring\_Boot] JPA 설정](https://velog.io/@chlgustjr/SpringBoot-JPA-%EC%84%A4%EC%A0%95)
18. [[Spring\_Boot] JPA Entity , JPQL](https://velog.io/@chlgustjr/SpringBoot-JPA-%EB%8F%99%EC%9E%91-%EB%B0%A9%EC%8B%9D-%EC%A4%80%EB%B9%84-%EA%B3%BC%EC%A0%95)
19. [[Spring\_Boot] JPA 영속성 - 내부 동작](https://velog.io/@chlgustjr/SpringBoot-JPA-%EC%98%81%EC%86%8D%EC%84%B1-%EB%82%B4%EB%B6%80-%EB%8F%99%EC%9E%91)
20. [[Spring\_Boot] JPA - 엔티티 매핑](https://velog.io/@chlgustjr/SpringBoot-JPA-%EC%97%94%ED%8B%B0%ED%8B%B0-%EB%A7%A4%ED%95%91)
21. [[Spring\_Boot] JPA - 연관관계 매핑(단방향),예제](https://velog.io/@chlgustjr/SpringBoot-JPA-%EC%97%B0%EA%B4%80%EA%B4%80%EA%B3%84-%EB%A7%A4%ED%95%91)
22. [[Spring\_Boot] JPA - 연관관계 매핑(양방향), 연관관계의 주인](https://velog.io/@chlgustjr/SpringBoot-JPA-%EC%97%B0%EA%B4%80%EA%B4%80%EA%B3%84-%EB%A7%A4%ED%95%91%EC%96%91%EB%B0%A9%ED%96%A5)
23. [[Spring\_Boot] JPA - 연관관계 매핑(단,양방향),예제](https://velog.io/@chlgustjr/SpringBoot-JPA-%EC%97%B0%EA%B4%80%EA%B4%80%EA%B3%84-%EB%A7%A4%ED%95%91%EC%96%91%EB%B0%A9%ED%96%A5%EC%98%88%EC%A0%9C)
24. [[Spring\_Boot] JPQL - 반환타입 & 결과 조회 API & 파라미터 바인딩](https://velog.io/@chlgustjr/SpringBoot-JPQL-%EB%B0%98%ED%99%98%ED%83%80%EC%9E%85-%EA%B2%B0%EA%B3%BC-%EC%A1%B0%ED%9A%8C-API-%ED%8C%8C%EB%9D%BC%EB%AF%B8%ED%84%B0-%EB%B0%94%EC%9D%B8%EB%94%A9)
25. [[Spring Boot] 임베디드 타입](https://velog.io/@chlgustjr/Spring-Boot-%EC%9E%84%EB%B2%A0%EB%94%94%EB%93%9C-%ED%83%80%EC%9E%85)
26. [[Spring\_Boot] JPQL 정리](https://velog.io/@chlgustjr/SpringBoot-JPQL-%EC%A0%95%EB%A6%AC)
27. [[Spring\_Boot] 프로젝트 설정, JPA SHOP - Member & Validation Check](https://velog.io/@chlgustjr/SpringBoot-JPA-SHOP-Member)
28. [[Spring\_Boot] JPA SHOP - Item & 변경감지와 머지(merge)](https://velog.io/@chlgustjr/SpringBoot-JPA-SHOP-Member-0xx5h78t)
29. [[Spring\_Boot] JPA SHOP - Order( 예외처리 만들기 & 가변인자 & CasecadeType & queryDSL)](https://velog.io/@chlgustjr/SpringBoot-JPA-SHOP-Order)
30. [[Spring\_Boot] queryDSL 활용(1)](https://velog.io/@chlgustjr/SpringBoot-queryDSL-%ED%99%9C%EC%9A%A9)
31. [[Spring\_Boot] queryDSL 활용(2)](https://velog.io/@chlgustjr/SpringBoot-queryDSL-%ED%99%9C%EC%9A%A92)

**🧩 MemberController**

* MemberController가 생성될 때, 생성자를 호출해준다.
* 즉, Service까지 생성해서 자동으로 호출해준다.
* **@Autowired**를 선언해주면 **MemberController가** 생성하면 스프링이 **memberService와**  
  연결을 해준다.
* 기존은 테스트를 통해서만 service의 오류 발생을 확인할 수 있지만,  
  서버 기동시에 연결 실패시 에러가 발생한다.
* 생성자에 **@Autowired**가 있으면 스프링이 연관된 객체를 **스프링 컨테이너에서 찾아서  
  넣어준다**. 이렇게 **의존 관계를 외부에서 넣어주는 것을 DI**(Dependency Injection),  
  **의존성 주입**이라고 한다.
* 이전엔 개발자가 직접 주입했고, 여기서는 **@Autowired에 의해 스프링이 주입**해준다.

**🧩@Autowired**

* @Autowired란, 스프링 DI(Dependency Injection)에서 사용되는 어노테이션입니다.
* 스프링에서 빈 인스턴스가 생성된 이후 @Autowired를 설정한 메서드가 자동으로 호출되고, 인스턴스가 자동으로 주입됩니다.

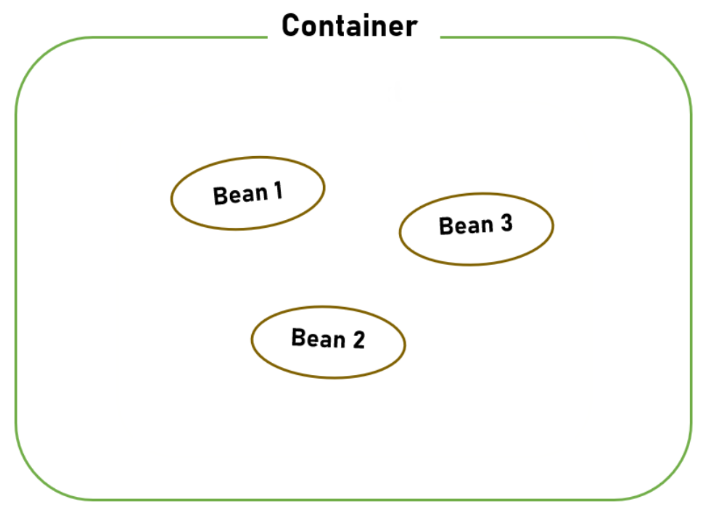
**🧩 @Autowired를 사용할 수 있는 위치**

* 생성자 (스프링 4.3부터는 생략 가능)
* Setter
* 필드

**🧩 제어 역전(IoC, Inversion of Control )**

* 개발자가 프레임워크의 기능을 호출하는 형태가 아니라 **프레임워크가 개발자의  
  코드를 호출**하기 때문에, 개발자는 전체를 직접 구현하지 않고 자신의 코드를  
  부분적으로 "**끼워넣기**"하는 형태로 구현할 수 있다.
* 프레임워크가 객체의 생성, 소멸과 같은 라이프 사이클을 관리하며 스프링으로부터  
  필요한 객체를 얻어올 수 있다.
* **객체의 의존성을 역전**시켜 객체 간의 결합도를 줄이고 유연한 코드를 작성할 수  
  있게 하여 가독성 및 코드 중복, 유지 보수를 편하게 할 수 있게 한다.

**🧩 스프링 컨테이너란?**

* 스프링 컨테이너는 자바 객체의 생명 주기를 관리하며, **생성된 자바 객체들에게  
  추가적인 기능을 제공**하는 역할을 합니다. 여기서 말하는 자바 객체를 **스프링에서는  
  빈(Bean)**이라고 부릅니다.
* 개발자는 객체를 생성하고 소멸할 수 있는데, 스프링 컨테이너가 이 역할을 대신해 준다.  
  즉, **제어의 흐름을 외부에서 관리**하는 것. 또한, **객체들 간의 의존관계**를 **스프링 컨테이너**가  
  런타임 과정에서 **알아서 만들어 준다**.
* 스프링은 실행시 객체들을 담고있는 Container가 있다.
* 스프링은 스프링 컨테이너에 스프링 **빈(객체)을 등록**할 때, 기본으로 **싱글톤으로 등록**  
  

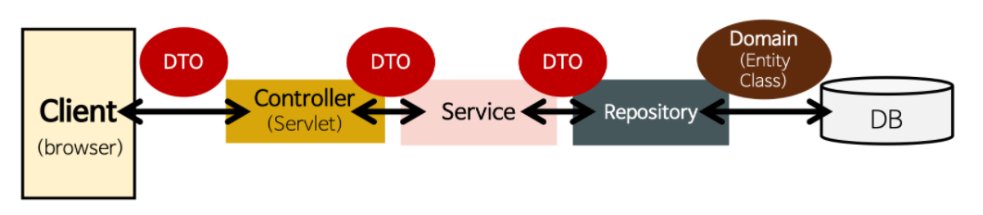
**🧩 POJO란?**

* Plain Old Java Object, 단순한 자바 오브젝트
* POJO란, 객체지향적인 원리에 충실하면서 환경과 기술에 종속되지 않고  
  필요에 따라 재활용될 수 있는 방식으로 설계된 오브젝트를 말한다.
* 그러한 POJO에 애플리케이션의 핵심 로직과 기능을 담아 설계하고 개발하는  
  방법을 POJO프로그래밍이라고 할 수 있다.

**🧩 의존성 주입(DI : Dependency Injection)**

* 객체간의 의존성이 존재할 경우 개발자가 직접 객체를 생성하거나 제어하는 것이 아니라,  
  **제어반전에 의하여 특정 객체에 필요한 다른 객체를 프레임워크가 자동으로 연결**시켜주는  
  것을 말한다.
* 개발자는 자신에게 필요한 객체를 직접 할당하지 않고, **인터페이스**를 통해 선언한  
  객체에 스프링 프레임워크에 의해 주입받아 사용할 수 있기 때문에 비지니스 로직에만  
  집중할 수 있다.
* 개발자는 객체를 선언만 할 뿐, 할당은 프레임워크에서 자동으로 이루어진다.

**스프링 프레임워크에서 정형화된 패턴**

* Component스캔을 통해 스프링의 관리대상이됨  
  

**🧩 회원저장 DB 실습**

* Controller -> Service호출 -> Repository호출
* Controller -> Repository호출 가능
* 스프링 프레임워크에서 관리하는 대상임을 Annotation(@)으로 명시

**💊 Member.java**

public class Member {

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**💊 MemberRepository**

* 인터페이스이므로 추상 메서드로 정의만 한다.

public interface MemberRepository {

// 회원 저장

Member save(Member member);

// 전체 찾기

List<Member> findAll();

}

**💊 JdbcMemberRepository**

DB가 아닌 메모리에 저장

@Repository

public class JdbcMemberRepository implements MemberRepository {

//주입

private final DataSource dataSource;

// 생성자가 하나일땐 생략 가능

//@Autowired

public JdbcMemberRepository(DataSource dataSource) {

this.dataSource = dataSource;

}

@Override

public Member save(Member member) {

String sql = "INSERT INTO MEMBER VALUES (member\_seq.nextval, ?)";

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

conn = dataSource.getConnection();

String generatedColums[] = { "ID" };

pstmt = conn.prepareStatement(sql, generatedColums);

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

pstmt.executeUpdate();

rs = pstmt.getGeneratedKeys();

if (rs.next()) {

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

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

rs.close();

pstmt.close();

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

return member;

}

@Override

public List<Member> findAll() {

String sql = "select \* from member";

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

List<Member> members = null;

try {

conn = dataSource.getConnection();

pstmt = conn.prepareStatement(sql);

rs = pstmt.executeQuery();

members = new ArrayList<Member>();

while (rs.next()) {

Member member = new Member();

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

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

members.add(member);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

rs.close();

pstmt.close();

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

return members;

}

}

**💊 MemberService**

@Service

public class MemberService {

// Service에서 join메서드와 findMembers를 호출해서 회원가입과 전체회원조회

// MemberRepository memberRepository = new MemoryMemberRepository();

private final MemberRepository memberRepository;

@Autowired

public MemberService(MemberRepository memberRepository) {

this.memberRepository = memberRepository;

}

// 회원가입

// 파라미터로 member받아옴

public int join(Member member) {

//save메서드 호출 member를 넘겨준다

memberRepository.save(member);

//member의 Id만 넘겨준다

return member.getId();

}

// 전체회원 조회

public List<Member> findMembers(){

// memberRepository에 findAll메서드로 받은 것을 전체리턴

return memberRepository.findAll();

}

}

**💊 MemberController**

* Controller가 Service에 의존한다라고 표현
* Service는 Repository를 의존
* Service는 여러 Controller에서 가져다 쓸 수 있기 떄문에 Spring Container에 등록을 해야한다.

@Controller

public class MemberController {

//스프링스럽게 사용하는것 X

//MemberService mService = new MemberService();

// 스프링스럽게 작업하기

// 중간에 memberservice가 다른영역으로 교체될 일이 없기 때문에 private final로 잠군다

// service는 Spring Container에 하나만 생성 및 등록해서 같이 공유해서 쓸 수 있다.

private final MemberService memberService;

@Autowired

public MemberController(MemberService memberService) {

this.memberService = memberService;

}

/\* 1.

\* 필드 주입(Field Injection)

\* - final 키워드를 사용할 수 없어, 순환 참조가 발생할 수 있다. 권장X

\*/

//@Autowired private final MemberService memberService;

/\* 2.

\* Setter Injection(수정자 주입)

\* - public으로 노출이 되기 때문에 다른곳에서 주입 가능하다.

\*/

// private MemberService memberService;

// @Autowired

// public void setMember(MemberService memberService) {

// this.memberService = memberService;

// }

// 회원가입 폼태그

@GetMapping(value = "/members/new")

public String createForm() {

// createMemberForm.html 생성

return "members/createMemberForm";

}

// createMemberForm에서 post방식으로 전달함

@PostMapping(value = "/members/new")

public String create(MemberFormDTO form){

Member member = new Member();

// 폼데이터로 받아온 setName세팅

// id값은 seq로 repository에 자동증가 세팅해놨음

member.setName(form.getName());

// 의존성주입으로 만든 memberService에서 join에서 member넘겨준다

memberService.join(member);

// 홈 화면으로 돌린다.

// 회원가입 완료후 입력했던 데이터는 지워야함 redirect

return "redirect:/";

//return "forward:/";

}

// 회원목록 확인

@GetMapping("/members")

public String list(Model model) {

List<Member> members = memberService.findMembers();

model.addAttribute("members",members);

//memberList.html 생성

return "members/memberList";

}

}

**💊 HomeController**

@Controller

public class HomeController {

// 기본 localhost:9090으로 들어오면 여기를 호출

@GetMapping("/")

public String home() {

return "home";

}

}

**💊 home.html**

<html xmlns:th="http://www.thymeleaf.org"><head>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

</head>

<body>

<div>

<h1>Hello Spring</h1>

<p>회원기능</p>

<p>

<a href="/members/new">회원가입</a>

<a href="/members">회원목록</a>

</p>

</div>

</body>

</html>

**💊 createMemberForm.html**

<body>

<form action="/members/new" method="post">

<label for="name">이름</label>

<input type="text" id="name" name="name" placeholder="이름을 입력하세요.">

<button type="submit">등록</button>

</form>

</body>

**💊 MemberFormDTO**

public class MemberFormDTO {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**💊 memberList**

<html xmlns:th="http://www.thymeleaf.org"><head>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

</head>

<body>

<div>

<table border="1">

<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>

</body>

</html>

**💊 DB**

CREATE TABLE member(

id number(10) PRIMARY KEY,

name varchar2(600)

);

CREATE SEQUENCE member\_seq

START WITH 1

INCREMENT BY 1;

INSERT INTO MEMBER VALUES (member\_seq.nextval, 'spring');

SELECT \* FROM MEMBER;

**💊 build gradle**

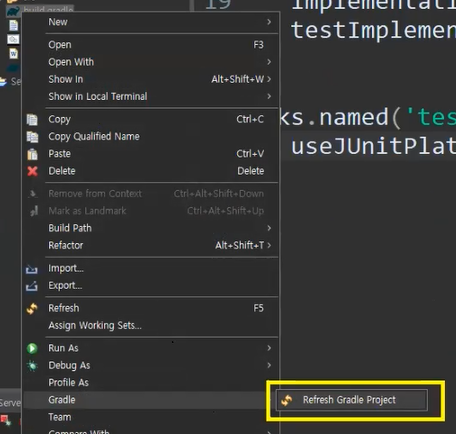
dependencies {

// 추가

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

implementation group: 'com.oracle.database.jdbc', name: 'ojdbc6', version: '11.2.0.4'

}

* 추가 후  
  

**💊 application.properties**

#port

server.port=9090

#thymleaf cashe

spring.thymeleaf.cache=false

#encoding

server.servlet.encoding.charset=UTF-8

server.servlet.encoding.force=true

server.servlet.encoding.enabled=true

#dbms

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:XE

spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

spring.datasource.username=myspring

spring.datasource.password=myspring