**회원 관리 예제 - 스프링 빈과 의존관계**

**📒 컴포넌트 스캔과 자동 의존관계 설정**

**멤버 컨트롤러**가 **멤버 서비스**를 통해 회원가입하고, 데이터를 조회할 수 있어야 한다.  
이러한 것을 **멤버 컨트롤러**가 **멤버 서비스**를 의존하는 **의존관계**가 있다고 한다.

**🖋️ 멤버 컨트롤러 생성**

**의존관계**를 만들기 위해, 우선 **멤버 컨트롤러**를 생성해야 한다.

src/main/java/spring.study1/controller/MemberController

package spring.study1.controller;

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

import org.springframework.stereotype.Controller;

import spring.study1.service.MemberService;

@Controller // 스프링 컨테이너가 뜰 때 MemberController를 생성하고 관리 해줌

public class MemberController {

private final MemberService memberService;

// MemberController를 생성할 때 호출

@Autowired // spring이 memberService를 스프링 컨테이너에 있는 memberService를 가져다 연결시켜줌(의존관계 주입)

public MemberController(MemberService memberService) {

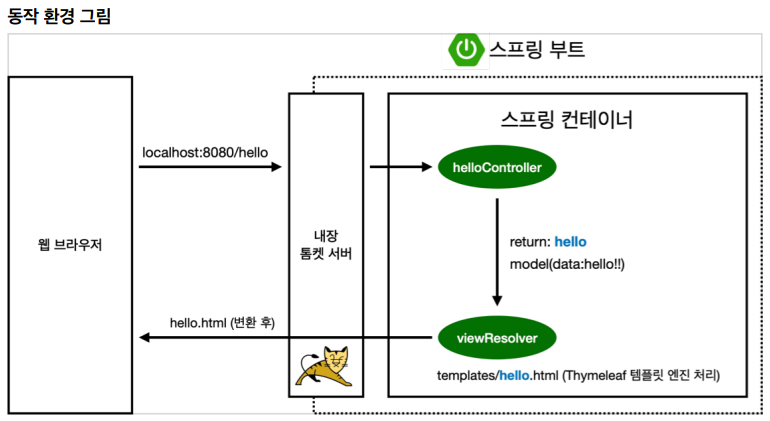
this.memberService = memberService;

}

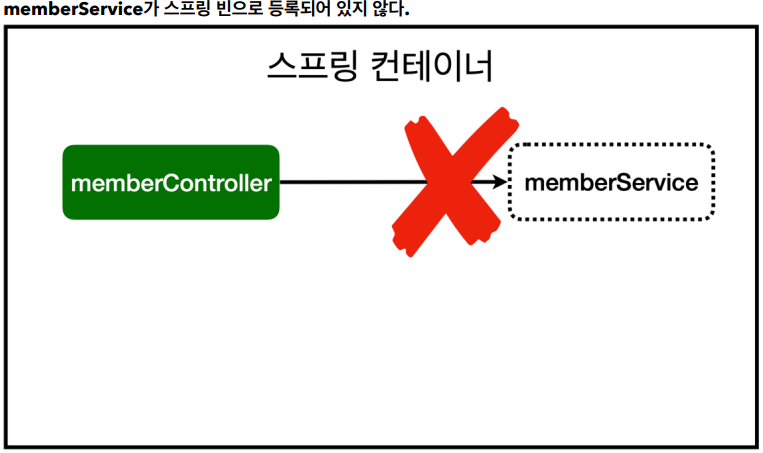
}

Alt + Ins 단축키로 생성자를 만들어 위와 같은 코드를 작성한다.

**@Controller**를 통해 컨트롤러가 **스프링 빈**에 등록이 된다. 여기서 **스프링 빈**은 쉽게 말하면 아래 그림에 있는 초록색 콩 모양이다.



그러나 이러한 상태에서 코드를 바로 실행하면 오류가 발생한다.  
왜냐하면 **멤버 컨트롤러**가 **멤버 서비스**를 통해 회원가입하고, 데이터를 조회를 해야돼서 **@Autowired**를 통해 스프링 컨테이너에 있는 **MemberService**를 가져와야 하는데, 스프링 컨테이너에는 **MemberService**가 없기 때문이다.  
그래서 의존관계가 정상적으로 작동하기 위해서 **MemberService**도 **스프링 빈**에 등록를 해줘야 한다.



**🖋️ 스프링 컨테이너에 스프링 빈 연결**

src/main/java/spring.study1/service/MemberService

package spring.study1.service;

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

import org.springframework.stereotype.Service;

import spring.study1.domain.Member;

import spring.study1.repository.MemberRepository;

import java.util.List;

import java.util.Optional;

// spring이 스프링 컨테이너에 MemberService를 등록해줌

@Service

public class MemberService {

// 저장소

private final MemberRepository memberRepository;

// spring이 MemberRepository를 스프링 컨테이너에 있는 MemberRepository를 가져다 연결시켜줌

@Autowired

public MemberService(MemberRepository memberRepository) {

this.memberRepository = memberRepository;

}

...

}

**@Service**을 통해 **MemberService**를 **스프링 컨테이너**에 넣어 **스프링 빈**에 등록하였다.  
그러면 **MemberService**와 **MemberController**는 서로 연결이 되는 **의존관계가** 된다.

그리고 **MemberService**가 **MemberRepository**를 통해 save, findById 등 메서드를 사용할 수 있어야 하기 때문에 **@Autowired** 코드를 추가해 준다.  
그러면 **@Autowired**를 통해 **스프링 컨테이너**에 있는 **MemberRepository**를 가져다 연결시켜주지만, **스프링 컨테이너**에 **MemberRepository**가 없어 이도 **MemberService**처럼 **스프링 빈**에 등록해줘야 한다.

src/main/java/spring.study1/repository/MemoryMemberRepository

package spring.study1.repository;

import org.springframework.stereotype.Repository;

import spring.study1.domain.Member;

import java.util.\*;

// spring이 스프링 컨테이너에 MemoryMemberRepository를 등록해줌

@Repository

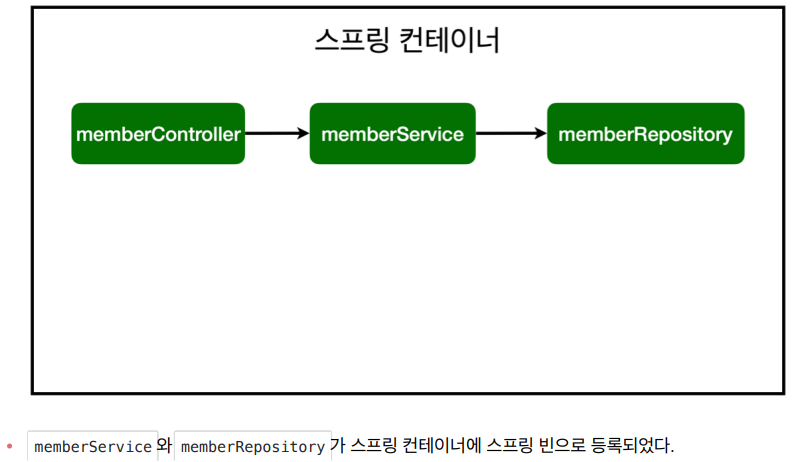
public class MemoryMemberRepository implements MemberRepository{

...

}

**@Repository**를 통해 **MemoryMemberRepository**를 **스프링 빈**으로 등록시켰다.

그러면 아래 그림과 같이 **스프링 컨테이너**에 **스프링 빈**으로 등록이 된 것을 확인할 수 있고 **의존관계**도 확인할 수 있다.

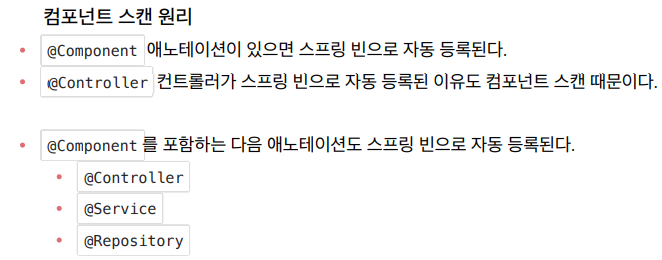


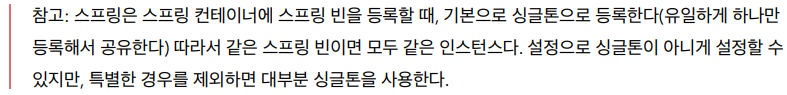
**🖋️ 스프링 빈 정리**

**스프링 빈**을 등록하는 방법에는 2가지가 있다.

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

우리가 위에서 사용한 **@Controller**, **@Service**, **@Repository**가 바로 **컴포넌트 스캔과 자동 의존관계 설정** 방법이었다.





**📒 자바 코드로 직접 스프링 빈 등록하기**

**🖋️ 스프링 빈 등록 전 사전 설정**

**스프링 빈**을 등록하는 방법 2가지 중에 하나는 위에서 직접 사용해 봤고, 나머지 다른 방법인 **자바 코드로 직접 스프링 빈 등록하기** 를 직접 사용할 것이다.

**자바 코드로 직접 스프링 빈 등록하기** 방법을 실행하기 위해,  
회원 서비스와 회원 리포지토리의 **@Service**, **@Repository**, **@Autowired** 애노테이션을 제거하고 진행해야 한다.

이때, 회원 컨트롤러는 그대로 둔다.

src/main/java/spring.study1/service/MemberService

package spring.study1.service;

import spring.study1.domain.Member;

import spring.study1.repository.MemberRepository;

import java.util.List;

import java.util.Optional;

public class MemberService {

// 저장소

private final MemberRepository memberRepository;

public MemberService(MemberRepository memberRepository) {

this.memberRepository = memberRepository;

}

...

}

src/main/java/spring.study1/repository/MemoryMemberRepository

package spring.study1.repository;

import spring.study1.domain.Member;

import java.util.\*;

public class MemoryMemberRepository implements MemberRepository{

...

}

회원 서비스와 회원 리포지토리의 **@Service**, **@Repository**, **@Autowired** 를 제거하고 프로젝트를 실행하면 오류가 발생한다.

**🖋️ JAVA 코드로 스프링 빈 연결**

**자바 코드로 직접 스프링 빈 등록하기** 위해

java/spring/study1 에 **SpringConfig** 라는 새로운 클래스 파일을 생성하여 다음과 같이 코드를 작성한다.

package spring.study1;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import spring.study1.repository.MemberRepository;

import spring.study1.repository.MemoryMemberRepository;

import spring.study1.service.MemberService;

@Configuration

public class SpringConfig {

// Spring bean에 MemberService 등록

@Bean

public MemberService MemberService() {

// Spring bean에 등록되어 있는 MemberRepository를 MemberService에 넣어줌

return new MemberService(memberRepository());

}

// Spring bean에 MemberRepository 등록

@Bean

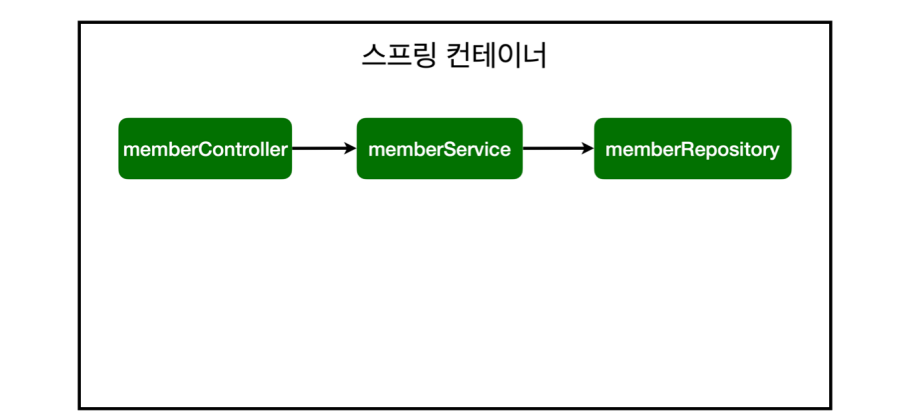
public MemberRepository memberRepository() {

return new MemoryMemberRepository();

}

}

그러면 아래 그림과 같이 **스프링 컨테이너**에 **스프링 빈**으로 등록이 되고, 서로 **의존 관계**를 가지게 된다.



실무에서는 주로 정형화된 컨트롤러, 서비스, 리포지토리 같은 코드는 **컴포넌트 스캔**을 사용한다. 그러나 정형화 되지 않거나, 상황에 따라 구현 클래스를 변경해야 하면 설정을 통해 **스프링 빈**으로 등록한다.

그래서 아직 DB가 정해져 있지 않는 프로젝트라서 **스프링 빈**으로 등록하는 것을 추천한다.

**📖 회원 관리 예제 - 웹 MVC 개발**

**📒 회원 웹 기능 - 홈 화면 추가**

**🖋️ 홈 화면 컨트롤러, HTML**

홈 화면의 **컨트롤러**를 우선 만들어 준다.

src/main/java/spring.study1/controller/HomeController

package spring.study1.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class HomeController {

@GetMapping("/") // localhost::8080

public String home() {

return "home"; // home.html 호출

}

}

<http://localhost:8080/> 페이지에 들어가면 해당 메서드를 호출하게 되고,  
**return "home";** 이 home.html 파일을 호출해주기 때문에 이에 대한 파일도 새롭게 추가해야 한다.

src/main/resources/template/home.html

<!DOCTYPE HTML>

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

<body>

<div class="container">

<div>

<h1>Hello Spring</h1>

<p>회원 기능</p>

<p>

<a href="/members/new">회원 가입</a>

<a href="/members">회원 목록</a>

</p>

</div>

</div> <!-- /container -->

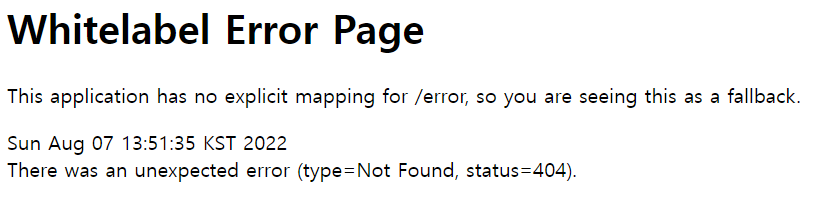
</body>

</html>

홈 화면을 만들고 실행을 해서 <http://localhost:8080/> 에 들어가면 아래와 같은 화면이 뜬다.



그리고 하이퍼링크 처리가 되어 있는 **회원 가입**과 **회원 목록** 페이지에 들어가면, 아직 해당 html 파일과 컨트롤러가 만들어 있지 않아서 **Error Page**가 뜬다.



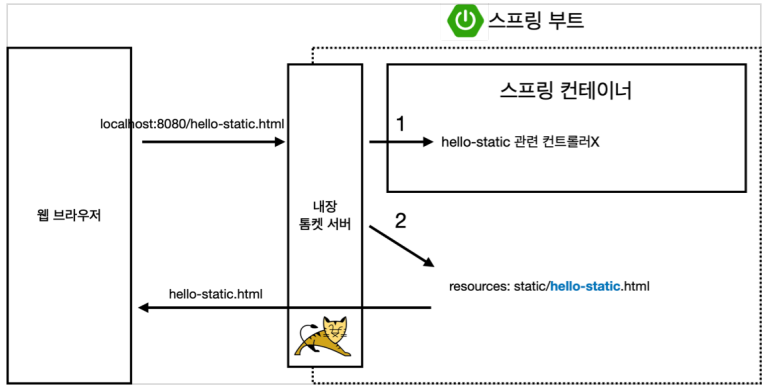
**🖋️ 컨트롤러 설명**

src/main/resources/static/index.html

그 전에 **정적 html** 파일인 index.html을 만들었고, <http://localhost:8080/> 에 들어가면 해당 html이 떴었지만, 지금은

src/main/resources/template/home.html 에 있는 **html**이 화면에 뜬다.

왜냐하면 **내장 톰켓 서버**에서 해당 페이지에 대한 **컨트롤러**를 찾고, 만약 그 **컨트롤러**가 없으면 **정적 컨텐츠**를 찾게 되는 **컨트롤러가 정적 파일보다 우선순위가 높다** 라는 이유 때문이다.  
그래서 내장 톰켓 서버는 HomeController라는 **컨트롤러**를 찾고, 해당 **컨트롤러**에서 **return** 해준 **home.html**이 화면에 뜨게 된다.



**📒 회원 웹 기능 - 등록**

**🖋️ 회원 등록 폼 컨트롤러, HTML**

회원 등록 폼 **컨트롤러**를 추가해 준다.

src/main/java/spring.study1/controller/MemberController

package spring.study1.controller;

...

@Controller // 스프링 컨테이너가 뜰 때 MemberController를 생성하고 관리 해줌

public class MemberController {

private final MemberService memberService;

// MemberController를 생성할 때 호출

@Autowired // spring이 memberService를 스프링 컨테이너에 있는 memberService를 가져다 연결시켜줌(의존관계 주입)

public MemberController(MemberService memberService) {

this.memberService = memberService;

}

@GetMapping("/members/new") // localhost::8080/members/new

public String createForm() {

return "members/createMemberForm";

}

}

**localhost:8080/members/new** 페이지에 들어가면 해당 메서드를 호출하게 되고, **"members/createMemberForm"** 를 **return** 해주기 때문에 **members** 폴더에 **createMemberForm.html** 파일도 새롭게 추가해야 한다.

src/main/resources/template/members/createMemberForm.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>

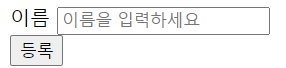
</form>

</div> <!-- /container -->

</body>

</html>

**회원 등록 폼 화면**을 만들고 실행을 해서 <http://localhost:8080/members/new> 에 들어가면 아래와 같은 화면이 뜬다.



그러나 이름을 입력하고 등록을 하면 Error Page가 뜬다.

이는 html의 **form태그** 안에 **input태그**에서 **POST 방식**으로 입력을 받으면 서버로 전달이 되기 때문에, **input**으로 입력받은 내용을 받기 위한 **회원 등록 컨트롤러**를 만들어야 한다.

**🖋️ 회원 등록 컨트롤러**

회원 등록 화면에서 **input**을 통해 입력받은 데이터를 전달 받을 **폼 객체**를 만들어야 한다.

src/main/java/spring.study1/controller/MemberForm

package spring.study1.controller;

public class MemberForm {

// createMemberForm의 input의 name 속성 값인 name과 같아야 함

private String name;

// getter

public String getName() {

return name;

}

// setter

public void setName(String name) {

this.name = name;

}

}

그리고 입력받은 데이터를 **컨트롤러**에서 회원을 **실제 등록**하는 기능을 만들어야 한다.

src/main/java/spring.study1/controller/MemberController

package spring.study1.controller;

...

@Controller // 스프링 컨테이너가 뜰 때 MemberController를 생성하고 관리 해줌

public class MemberController {

private final MemberService memberService;

...

@PostMapping("/members/new") // Post 방식으로 localhost::8080/members/new

public String create(MemberForm form) {

Member member = new Member(); // member 객체 생성

member.setName(form.getName()); // form에서 입력받은 이름을 member 객체 이름으로 넣음

memberService.join(member); // member 객체로 join(회원가입)

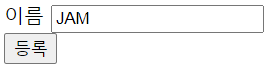
return "redirect:/"; // 바로 "localhost::8080/" 화면으로 이동

}

}

회원 등록 화면에서 입력받은 회원 이름을 등록하는 기능을 만들었다.

<http://localhost:8080/members/new> 에 들어가서 이름을 입력하고 등록 버튼을 누르면 **return "redirect:/";** 로 인해 home 화면으로 돌아가게 된다.





**회원을 등록**하는 기능까지 만들었고, 이제 이 등록한 회원을 **조회**하기 위한 **컨트롤러**와 **html** 을 만들어야 한다.

**🖋️ 회원 등록 컨트롤러 설명**

우리가 일반적으로 웹 페이지 주소를 입력해 들어가는 방식을 **GET 방식**이라고 한다.

회원 등록 **Form**에서 **input 속성** 중 **name**은 입력받아서 제출한 값의 **key값**이 되어, 값을 입력 받고 등록 버튼을 누르면 **"/members/new"** 로 **POST 방식**으로 넘어가게 된다.

<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>

그럼 **POST 방식**으로 넘거가게 되면 아래 **@PostMapping("/members/new")** 메서드로 호출이 된다.

@PostMapping("/members/new") // Post 방식으로 localhost::8080/members/new

public String create(MemberForm form) {

Member member = new Member(); // member 객체 생성

member.setName(form.getName()); // form에서 입력받은 이름을 member 객체 이름으로 넣음

memberService.join(member); // member 객체로 join(회원가입)

return "redirect:/"; // 바로 "localhost::8080/" 화면으로 이동

}

**📒 회원 웹 기능 - 조회**

**🖋️ 회원 조회 화면 추가**

입력 받은 input 값을 직접 화면으로 볼 수 있는 **회원 조회 화면**을 만들기 위해 우선 **컨트롤러**를 만들어 준다.

src/main/java/spring.study1/controller/MemberController

package spring.study1.controller;

...

@Controller // 스프링 컨테이너가 뜰 때 MemberController를 생성하고 관리 해줌

public class MemberController {

private final MemberService memberService;

...

@GetMapping("/members") // localhost::8080/members

public String list(Model model) {

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

model.addAttribute("members",members); // 회원 List를 model에 넣음

return "members/memberList";

}

}

**컨트롤러**에서 **return "members/memberList";** 를 했기 때문에 members 폴더에 **memberList.html** 이라는 파일을 만들어 **회원 조회 html**을 만들어 준다.

src/main/resources/template/members/memberList.html

<!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>

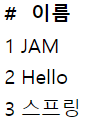
</div> <!-- /container -->

</body>

</html>

**회원 조회**하는 페이지를 다 만들었고 <http://localhost:8080/members> 에 들어가면 아래와 같은 화면이 뜬다.  


그래서 회원 가입 페이지에서 입력하고 등록을 눌러, **회원 조회** 페이지에 들어오면 아래와 같은 화면이 뜬다.



**🖋️ 회원 조회 설명**

html 파일에서 **${members}**는 controller에서 넘겨준 **모든 member들의 List**인 **members** 라는 **model**이다.

<tr th:each="member : ${members}">

<td th:text="${member.id}"></td>

<td th:text="${member.name}"></td>

</tr>

**th:each**는 **members**의 객체의 수만큼 loop를 도는 **반복문**의 역할을 하며, **members** 객체를 하나씩 꺼내 **member**에 넣고, member의 id와 name이 각각 **${member.id}** 와 **${member.name}** 에 치환이 된다.

그래서 개발자 모드에 들어가면 다음과 같이 값이 렌더링 된 것을 확인할 수 있다.



그러나 회원 등록한 정보들은 **memory**에 저장되어 있기 때문에 서버를 껐다가 키면, 등록된 정보들이 사라지게 된다.  
그래서 이를 해결하기 위해 데이터를 저장하기 위한 **데이터베이스(DB)** 를 다음 게시물에서 만들 것이다.