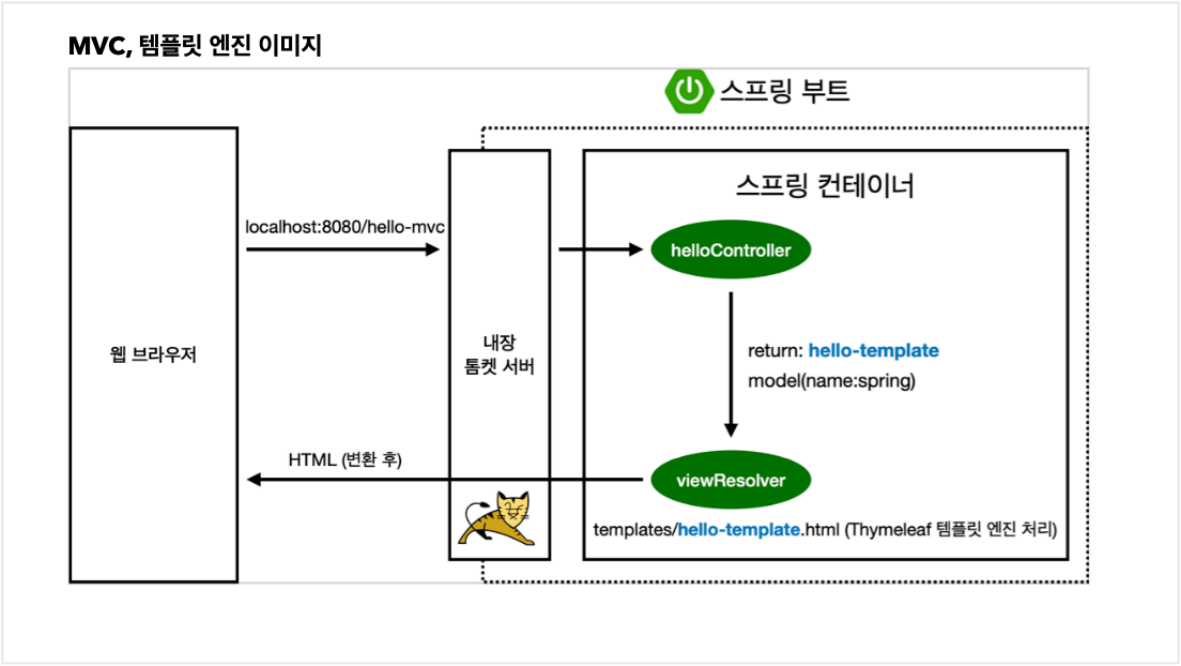
SpringBoot 2일차/웹MVC까지

스프링 빈과 의존관계

컴포넌트 스캔과 자동 의존관계 설정

이제 화면을 붙인다. 그러려면 컨트롤러랑 뷰템플릿이 필요함. 회원가입하고 회원가입 결과를 html로 뿌려준다.
멤버컨트롤러를 만들고 그가 멤버서버를 통해서 회원가입하고 데이터를 조회할수있어야함. -> 맴버컨트롤러가 멤버서버에 **의존함**
@Controller를 써놓으면 스프링 컨테이너가 컨트롤러 어노테이션이 있으면 멤버컨트롤러 객체를 생성해서 스프링이 컨테이너에 넣어놓고 관리함.



컨트롤러가 있으면 스프링이 알아서 관리함.
스프링이 관리하기 시작하면 다 스프링컨테이너에 등록하고 스프링에서 받아서 써야함.
membercontroller에서 new해서 만들어서 쓰게되면 다른데서도 가져다쓸수있는데 별 기능이 없다. 여러개 생성할 필요가없고 하나 생성해서 공용으로 쓰면됨
=> 스프링컨테이너한테 등록하고 쓴다. 하나만 등록되고 외의 부가적인 효과를 볼수있으므로

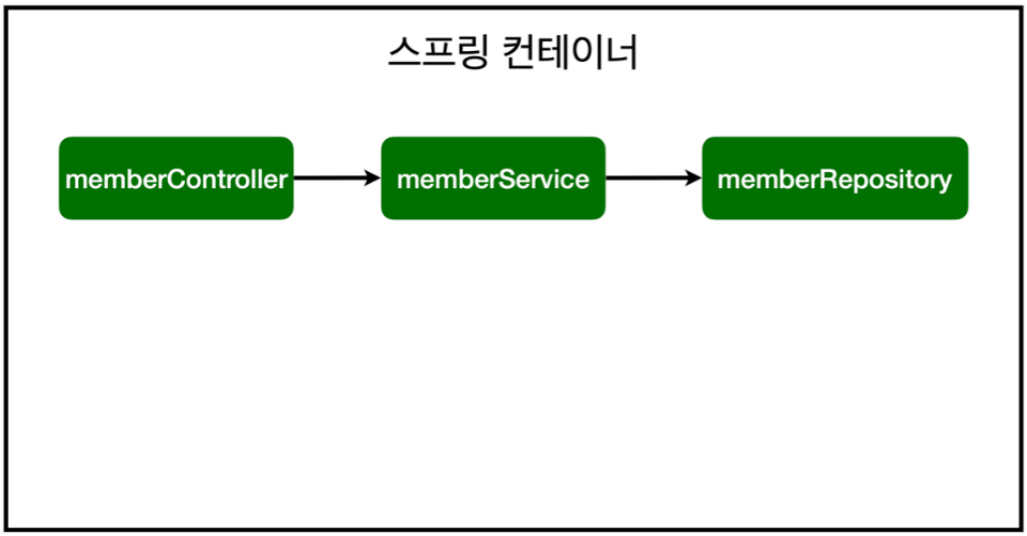
@Autowired : **멤버컨트롤러는** 스프링컨테이너가 뜰때 호출되고 이게 표시되어있으면 스프링 컨테이너의 **멤버서비스랑** 연결시켜줌
=> 생성자에서 오토와이어드 쓰면 멤버컨트롤러 생성될때 스프링 빈에 등록되어있는 멤버서비스 객체를 연결해줌 => Dependency Injection(DI) 의존관계 주입. 스프링이 밖에서 넣어줌

그냥 오토와이어드만 하면 멤버컨트롤러는 안됨. 왜냐면 MemberService 객체가 그냥 순수한 자바 클래스이므로 => memberservice에 @service추가하면 스프링컨테이너에 추가해줌

repository(memberRepository-interface이므로 말고 구현체로.)는 @Repository 추가

컨트롤러로 외부요청을 받고 서비스에서 비즈니스 로직 만들고 레포지토리에 데이터 저장하

기!!!!!



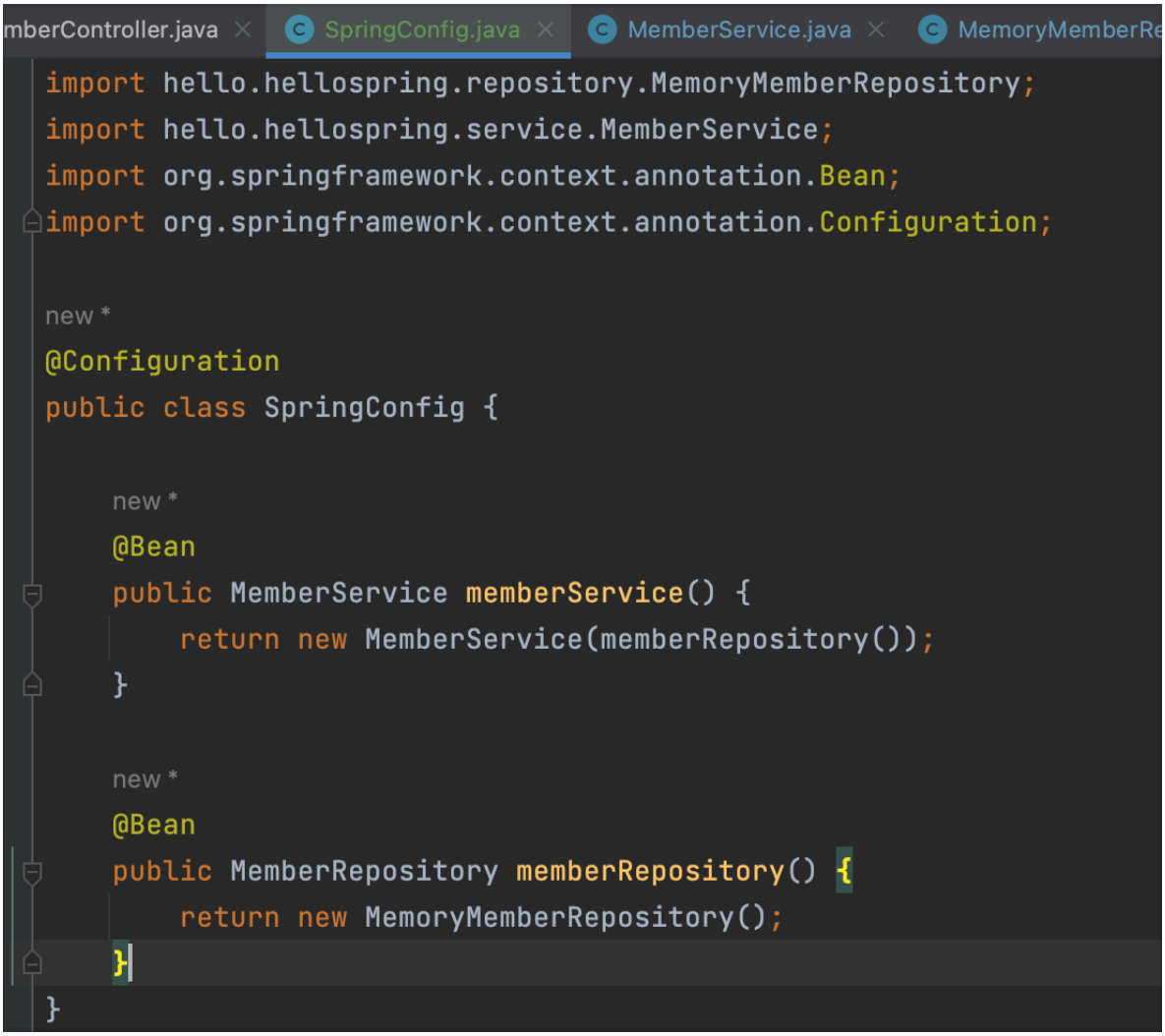
DI(의존성주입)
컨트롤러랑 서비스랑 연결시켜줘야함. 이때 오토와이어드 -> 멤버컨트롤러가 생성이 될때 스프링 빈에 있는 멤버서비스 객체를 가져다가 딱 넣어줌. => 의존관계 주입
멤버서비스는 멤버리포지토리가 필요함.
멤버 서비스를 생성할때 @Service네? 하면서 등록할 때 @Autowired를 보고 레포지토리가 필요하네 하고 memberRepository를 호출함
지금같은경우에 리포지토리의 구현체는 메모리멤버리포지토리. 그래서 이걸 서비스에 딱 주입!

정리 : 스프링 빈을 등록하는데는 두가지 방법
- 컴포넌트 스캔과 자동 의존관계 설정 : @service, @Autowired, @Repository - 얘네한텐 다 @Component가 내장됨. => 컴포넌트 스캔!
  - 오토 와이어드는 위 그림에서 선 역할을 한다 -> 컨트롤러가 서비스를 쓸수있게, 서비스가 리포지토리를 쓸수있게.,.
  - @component 어노테이션이 있으면 스프링빈으로 자동 등록.
    - 아무데나 @component가 붙어도 되려나? : 아니다. 어디서부터 되냐면
      - 우리는 지금 HelloSpringApplication을 실행시키는데 이 파일의 패키지는 hello.hellospring이다. 이 하위애들은 자동으로 스프링빈에 등록되지만 이 패키지에 속하지 않는 애들은 등록이 안된다
      - @SpringBootApplication 어노테이션 까보면 @ComponentScan이 있다 -> 이게있으면 다 찾아올수있음!
  - 스프링 빈은 무조건 **싱글톤**으로 등록함! 유일하게 하나만 등록해서 **공유**
- 자바 코드로 직접 스프링 빈 등록하기

자바 코드로 직접 스프링 빈 등록하기

hello.hellospring 폴더에 SpringConfig 파일 만들고 클래스 작성하기
@Configuration을 써놓으면 알아서 설정이구나 하고 시작함

@Bean을 써놓으면 밑에있는걸 알아서 빈에 추가



멤버서비스랑 멤버리포지토리를 스프링빈에 등록하고 등록되어있는 멤버리포지토리를 멤버서비스에 넣어줌
=> 두개 다 넣어놓고 나중에 서비스에 리포지토리를 넣어줌. 이것도 의존성 주입?
이렇게 되면 스프링 컨테이너 연결 그림이 딱 완성됨.
@Controller는 어쩔수없이 그냥 컴포넌트스캔으로 해야함.
오토와이어로 연결해주긴해야함

DI

```java
public class MemberService {

    // setter를 통해 들어오기
//    private MemberRepository memberRepository;
//
//    @Autowired
//    public void setMemberRepository(MemberRepository memberRepository) {
//        this.memberRepository = memberRepository;
//    }

//     생성자를 통해 들어오기
    5 usages
    private final MemberRepository memberRepository;
    2 usages  bongsh0112
    public MemberService(MemberRepository memberRepository) { //직접 new로 생성하는게 아니라 외부에서 넣어주도록
        this.memberRepository = memberRepository;
    }
```

- **생성자를 통해 들어오는 방법 : 조립 시점에 딱 조립해놓고 나중에 변경 못하게 막아버리기**
- 필드를 통해 하는 법 : Autowired로 필드에서 바로 해줌. 별로안좋음 -> 바꿀수있는방법 이없어서..
  - @Autowired private MemberService memberService; 와 같은 형식...
- 세터 주입 : 생성은 생성대로 되고 나중에 세터가 호출돼서 멤버서비스에 딱 들어옴
  - 세터 주입 시 멤버 리포지토리 멤버변수에서 final 빠지는 이유 : 처음 초기화가 아니라 세터 초기화이기 때문에..
  - 커맨드 n 눌러서 세터 하고 오토와이어드 -> 누군가가 멤버컨트롤러를 호출했을 때 퍼블릭으로 열려있어야함 근데 중간에 바꿀필요없는데 근데 퍼블릭이라 중간에 바뀔수도 있음
  - 이때 누군가가 memberService.~~ 하면 아무데나 노출됨

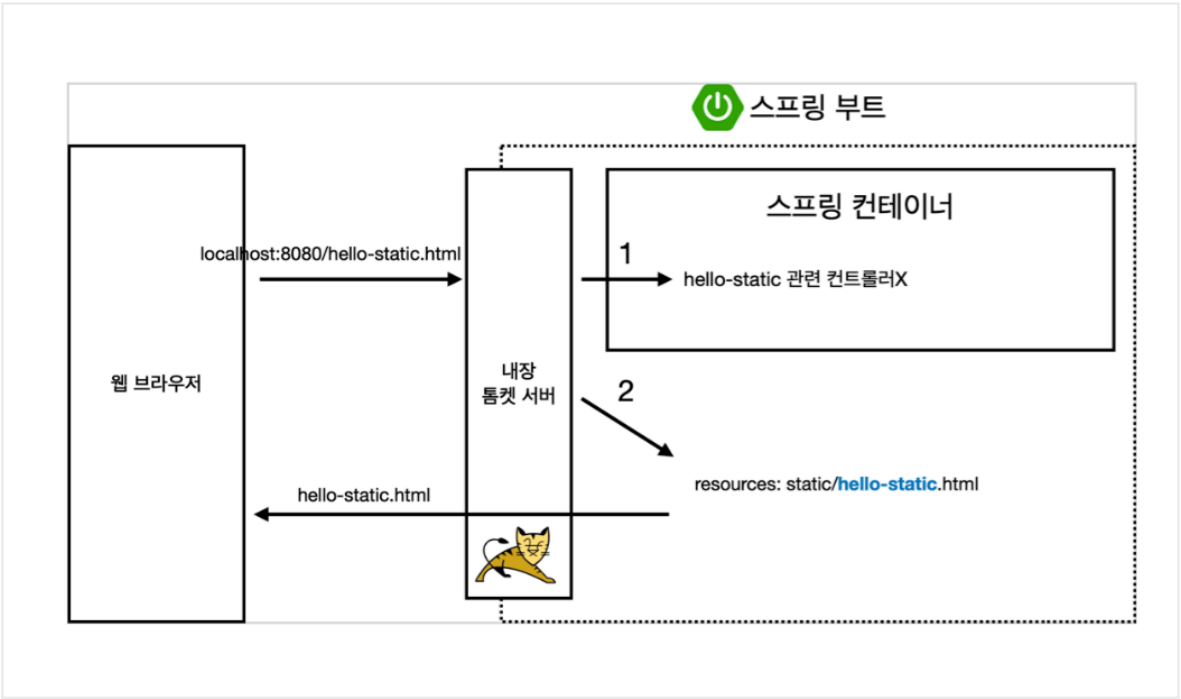
실무에서 비정형화된거는 거의 스프링빈을 직접 써줘야됨 -> 상황에 따라 구현 클래스를 변경 -> 지금은 메모리레파지토리를 쓰지만 나중에 디비를 갖고오면 다른거 손 안대고 한줄만 바꾸면 되게 하기위해
정형화된 코드는 그냥 컴포넌트 스캔

실행중에 동적으로 변하는 경우 : 서버가 떠있는데 중간중간에 막 바꿔치기됨 -> 그럴일이있으면 컨피그 파일 바꿔서 서버를 다시 올림

스프링빈이 관리하고 있는 것에 대해서만 Autowired가 작동한다. 스프링 컨테이너에 있는 애들(만약 스프링컨피그로 등록한다면 컨피그에 @Bean으로 있는애들)만 오토와이어가 동작함

회원 관리 예제를 웹 MVC로 개발

홈 화면 추가



homecontroller에서 home.html에 연결되도록 만들어놓고
templates에서 home.html을 만들면 static에 있는 애들보다 먼저 나온다. 그 이유가 위 그림임
(요청이 들어올 때 관련 컨트롤러를 먼저 찾기 때문!! 1, 2번의 차례를 보자)

<input type="text" id="name" name="name" placeholder="이름을 입력하세요">
members/new에서의 위 input 태그의 name은 키. 입력값은 밸류 -> http의 post

회원등록하는 컨트롤러 만들기

html의 폼에서 등록 버튼을 누르면
@PostMapping("/members/new") -> membercontroller에 삽입되는 코드. 아마 name을 post해주는거라 postmapping인듯
public String create(MemberForm form) {}

먼저 회원가입을 들어가면 members/new에 들어가는데 url에 직접 이걸 쳐서 들어가면 get 방식
겟매핑이 리턴하는건 templates에서 찾는다했으니까 viewResolver를 통해서 선택이 되고
thymeleaf 엔진이 members/createMemberForm을 렌더링해준다.

이름을 치고 (name 키값의 value) 등록을 누르면 form 태그의 action url로 Post된다. -> postmapping으로 옴.

포스트매핑은 보통 폼같은거에 데이터를 넣어서 전달할때 씀
겟매핑은 조회할때주로씀

```java
@GetMapping("/members/new")
public String createForm() {
    return "members/createMemberForm";
}

new *
@PostMapping("/members/new")
public String create(MemberForm form) {
    Member member = new Member();
    member.setName(form.getName());

    memberService.join(member);

    return "redirect:/"; //회원가입이 끝났으니 홈화면으로 돌아감
}
```

여기서 같은 url이라도 post 방식으로 가면 postmapping이 선택되고 get방식으로가면 getmapping이 선택됨
post 방식으로 넘어가려면 회원가입 버튼을 누르면 post방식으로 넘어가진다
그러면 멤버컨트롤러의 포스트매핑으로 온다

여기서 재밌는건 MemberForm form인데

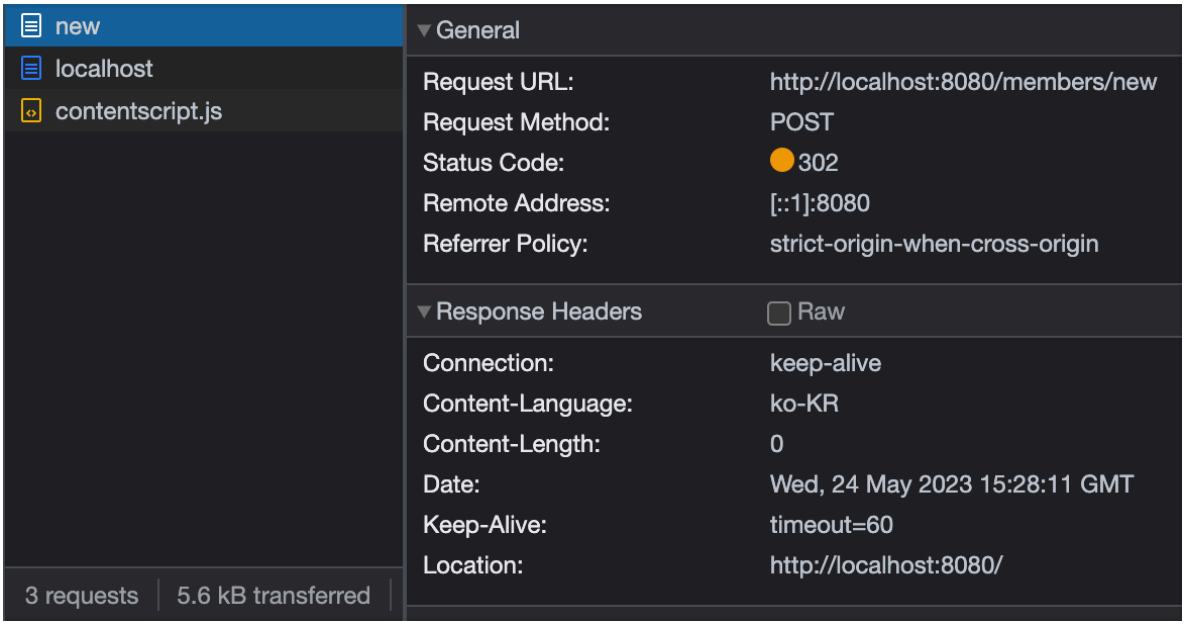
```java
package hello.hellospring.controller;

1 usage  new *
public class MemberForm {
    2 usages
    private String name; // createMemberForm.html의 name이랑 맞아떨어지는 부분

    1 usage  new *
    public String getName() {
        return name;
    }

    no usages  new *
    public void setName(String name) {
        this.name = name;
    }
}
```

여기에 name이라는 키로 받은 value가 setName(name)으로 변수에 값을 넣어주고
위에 포스트매핑에서는 form.getName()으로 꺼내올수있다



버튼누르면 나오는 요청. 개발자도구의 네트워크탭에서 볼수잇음!

```
<!DOCTYPE HTML>
<html>
<body>
<div class="container">
    <div>
        <table>
            <thead>
            <tr>
                <th>#</th>
                <th>이름</th> </tr>
            </thead>
            <tbody>
            <tr>
                <td>1</td>
                <td>spring1</td>
            </tr>
            <tr>
                <td>2</td>
                <td>spring2</td>
            </tr>
            </tbody>
        </table>
    </div>
</div> <!-- /container -->
</body>
</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>
```

th: 라고 되어있는 부분이 타임리프 문법임. forEach랑 비슷함
데이터는 현재까지는 메모리에 있기 때문에 서버 껐다키면 없어짐.