

스프링 프레임워크 ver.2

생명주기

- 빈(Bean) 객체의 생성주기는 스프링 컨테이너의 생명주기와 같이한다.

빈 객체 생성 & 소멸 시 실행되는 **Method**

1. 방법 (xml에서 선언)

```
<bean id="bookRegisterService"
      class="com.brms.book.service.BookRegisterService"
      init-method="initMethod" destroy-method="destroyMethod"/>
```

```
public void initMethod() {
    System.out.println(" bookRegisterService 생성시 실행");
}

public void destroyMethod() {
    System.out.println(" bookRegisterService 소멸시 실행");

}
```

2. 방법 (Interface 사용)

```
public class BookDao implements InitializingBean, DisposableBean {

    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("빈(Bean) 객체 생성 단계");
    }

    @Override
    public void destroy() throws Exception {
        System.out.println("빈(Bean) 객체 소멸 단계");
    }
}
```

어노테이션을 이용해 스프링 설정

.java 파일을 이용한 스프링 설정

GenericXmlApplicationContext -> AnnotationConfigApplicationContext 사용

```
AnnotationConfigApplicationContext ctx =  
    new AnnotationConfigApplicationContext(MemberConfigJinho.class);
```

```
@Configuration  
public class MemberConfigJinho {  
  
    //<bean id="studentDao" class="ems.member.dao.StudentDao" ></bean>  
    @Bean  
    public StudentDao studentDao() {  
        return new StudentDao();  
    }  
    /*  
     * <bean id="registerService"  
     * class="ems.member.service.StudentRegisterService">  
     * <constructor-arg ref="studentDao" ></constructor-arg>*/  
    @Bean  
    public StudentRegisterService registerService() {  
        return new StudentRegisterService(studentDao());  
    }  
    .  
    .  
}
```

.xml 파일 대신 자바 코드로 변경해서 사용.

여러개의 Config.java 를 import 할때

```
@Import({MemberConfig2.class, MemberConfig3.class})
```

import 어노테이션을 사용해서 import해서 사용

웹 프로그래밍 설계 모델

MVC 모델1

이용자(클라이언트)(브라우저) 가 서버에 Request 를 하게 된다.

서버는 Request를 처리하게 된다. DB요청등을 수행하고 다시 정보처리를 수행하고

서버가 Response를 진행하게 된다.

모든 업무를 하나의 controller단에서 구성

장점 : 개발이 빠르다.

단점 : 유지보수가 힘들다.

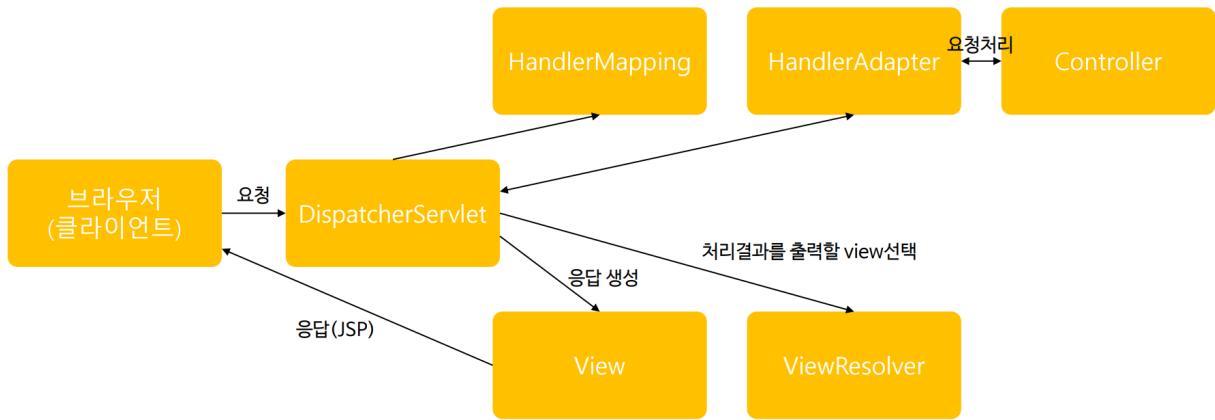
MVC 모델2

Controller / Service / DAO / View(jsp) / Model /DB 로 구성

장점 : 유지보수가 좋다.

Front-Controller 패턴(MVC Model2)

req -(요청 ,URL) -> DispatcherServlet -> 요청-> [HandlerMapping] 찾음 -> DispatcherServlet -> HandlerAdapter (DispatcherServlet이 전달한 컨트롤러를 찾음) -> Controller로 이동 -> DBMS등 이용 -> HandlerAdapter -> DispatcherServlet (돌아감) -> ViewResolver (jsp확장자 페이지를 만들어주는 것)-> DispatcherServlet -> view 경로로 이동 -> (html, jsp등) -> view -> DispatcherServlet -> Response 마무리



정리

1. 사용자의 Request는 Front-Controller인 DispatcherServlet을 통해 처리된다.
2. HandlerMapping은 Request의 처리를 담당하는 컨트롤러를 찾기 위해서 존재한다. 여러 객체중 @RequestMapping 어노테이션이 적용된 것을 기준으로 판단하며, 적절한 컨트롤러가 찾아졌다면 HandlerAdapter를 이용해서 해당 컨트롤러를 동작시킨다.
3. Controller는 Request를 처리하는 비즈니스 로직을 작성하며, View에 전달해야 하는 데이터를 RequestScope에 담아서 전달한다. 응답 페이지에 대한 경로 처리는 ViewResolver를 이용하게 된다.
4. ViewResolver는 Controller가 리턴한 결과 값을 전체 경로로 완성시켜준다.
5. View는 실제 응답을 보내야하는 데이터를 HTML, JSP 등을 이용해서 생성하는 역할을 한다.
6. 만들어진 응답 페이지를 DispatcherServlet을 통해 전송한다.

HandlerMapping

사용자가 요청에 부합하는 컨트롤러 검색

HandlerAdapter

사용자의 요청에 부합하는 컨트롤러의 메소드 실행 요청

참조 블로그

[스프링\(Spring\) MVC 동작 구조 및 스프링 컨테이너](#)

Web.xml - DispatcherServlet 설정

```
<servlet>
    <servlet-name>appServlet</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-c
lass>
    <init-param>
        <param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

web.xml 한글 필터

```
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filte
r-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

servlet-context.xml

- <annotation-driven /> 으로 어노테이션 사용 명시
 - @Controller 사용 가능
- ViewResolver 설정

```
<beans:bean  
class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
    <beans:property name="prefix" value="/WEB-INF/views/" />  
    <beans:property name="suffix" value=".jsp" />  
</beans:bean>
```

Service & Dao 객체 구현

방법1. 스프링 설정 파일에서 생성

```
<beans:bean id="service"  
class="com.bs.lec17.member.service.MemberService"></beans:bean>
```

방법2. 어노테이션 사용

```
@Service  
public class MemberService implements IMemberService {
```

후에 의존 자동주입을 통해 Controller에서 사용한다.

```
@Autowired  
MemberService service;
```

Component 계층

```
@Component  
    └─ @Controller  
    └─ @Service  
    └─ @Repository
```

구현되어 있으므로 Component를 써서 사용해도 무관하지만 정확하게 명시하는 것이 좋다.

```
ex Service)
```

```
===== Service 생성
@Repository("memService")
public class MemberService implements IMemberService
===== Controller 자동주입
@Resource(name="memService")
MemberService service;
```

특정 이름을 지정 할 수도 있다.

```
ex Dao)
```

```
===== Dao 생성
@Repository
public class MemberDao implements IMemberDao
===== Service 자동주입
@.Autowired
MemberDao dao;
```

Controller

HomeController 분석

```
@Controller
public class HomeController {
```

컨트롤러 어노테이션

```
@RequestMapping(value = "/", method = RequestMethod.GET)
```

value 해당 위치에 따른 맵핑 , method 속성

```
public String home(Locale locale, Model model) {
```

locale -> 시간

model : data 전달자 -request도 Model 객체속에 있다.

```
model.addAttribute("serverTime", formattedDate );
```

request.setAttribute방식.

```
return "home";
```

이것을 servlet-context.xml (viewresolver)-> /WEB-INF/views/home.jsp 로 반환

SampleController 생성

```
@Controller  
@RequestMapping("/ex/*")  
public class SampleController {  
  
    @RequestMapping(value = "/basic", method = {RequestMethod.GET,  
RequestMethod.POST})  
    public void basic() {}  
}
```

/ 루트 경로 아래 ex 아래 모든 풀더 맵핑 => 클래스에도 사용 가능하다.

@RequestMapping(value = "/basic", method = {RequestMethod.GET,
RequestMethod.POST})

/ex/basic =>이고 GET, POST 메소드일 때 도달.

공통된 부분을 클래스 위에 `RequestMapping`으로 선언

Controller

Spring-Controller02 실습

command 객체 사용

```
<form action="/lec17P002/memJoin" method="post">  
    ID : <input type="text" name="memId" ><br />  
    PW : <input type="password" name="memPw" ><br />  
    MAIL : <input type="text" name="memMail" ><br />  
    PHONE : <input type="text" name="memPhone1" size="5" > -  
            <input type="text" name="memPhone2" size="5" > -  
            <input type="text" name="memPhone3" size="5" ><br />  
    <input type="submit" value="Join" >  
    <input type="reset" value="Cancel" >  
</form>
```

Join form에서 `RequestMapping("/memJoin")` 인 아래에 Controller에 전송.

```
@RequestMapping(value="요청값", method=RequestMethod.POST or .GET);
```

기존방식

```
public String memJoin( Model model, HttpServletRequest request) {  
    String memId = request.getParameter("memId");  
    String memPw = request.getParameter("memPw");  
    String memMail = request.getParameter("memMail");  
    String memPhone1 = request.getParameter("memPhone1");  
    String memPhone2 = request.getParameter("memPhone2");  
    String memPhone3 = request.getParameter("memPhone3");  
    . . .
```

command 객체 사용 (주로 사용)

```
public String memJoin(Member member) {  
    service.memberRegister(member.getMemId(), member.getMemPw(),  
    member.getMemMail(), . . .
```

parameter를 받는 방식이 Spring에서는 Command 객체를 사용

이때 Parameter로 설정한 객체에 Getter, Setter 가 전부 선언되어 있어야 한다.

jsp에서

```
<h1> memJoinOk </h1>  
ID : ${member.memId}<br />  
PW : ${member.memPw}<br />  
Mail : ${member.memMail} <br />  
Phone : ${member.memPhone} <br />
```

이런식으로 member 객체명의 앞을 소문자로 변경된 Command 객체로 사용 가능

```
@ModelAttribute("memberVO") Member member  
이런식으로 메소드의 파라미터를 받게 되면 jsp 상에서  
ID : ${memberVO.memId}<br />이런식으로 사용 가능.
```

[Lombok 라이브러리]

Lombok(롬복)을 이용하면 JAVA 개발 시 getter/setter, toString 생성자(), hash 등을 @DATA 등의 어노테이션을 통해서 자동으로 생성해준다.

다만 주의사항 등도 숙지할 필요가 있다.

[Lombok 사용상 주의점\(Pitfall\)](#) -블로그

```
@RequestMapping(value="/memLogin", method=RequestMethod.POST)
public String memLogin(Model model,
    @RequestParam("memId") String memId,
    @RequestParam("memPw") String memPw) {
```

=> `@RequestParam ("요청한 파라미터 이름") String memId`
= `String memId = request.getParameter("memId")` 같은 의미

`@RequestParam(value="요청한 파라미터 이름", required="필수 여부(boolean)")`
- 필수여부 : 잘 사용하지 않음

@ModelAttribute

```
@ModelAttribute("serverTime")
public String getServerTime(Locale locale) {

    Date date = new Date();
    DateFormat dateFormat =
DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, locale);

    return dateFormat.format(date);
}

@RequestMapping(value = "/memJoin", method = RequestMethod.POST)
public String memJoin(Member member) {

    service.memberRegister(member);

    return "memJoinOk";
}
```

ModelAttribute를 Method에 적용하게 되면 어느 부분이 실행되더라도 같이 호출된다.

```
<P> The time on the server is ${serverTime}. </P>
```

바로 사용 가능

Model & ModelAndView

ModelAndView 를 사용하면 addObject로 바인딩

`setViewName("view이름")`으로 메소드에 ModelAndView 객체로 전달

의문점

1. Interface 로 Service , Dao 를 구성하고 Impl 클래스를 만들어서 사용하는 이유

JinhoService / JinhoServiceImpl - class

2. Service - Mapper / Service - Dao - Mapper

3. 협업 프로그램 ? PL > 개발자 > PL JIRA<->Git MQ CI/CD

API 이거 API 고민

/로그인 회원가입 / 게시판 - 게시판 / 전국 캠핑장 자료 - api

내위치 api 위도 경도

전국 데이터베이스로 가까운곳 거리

지도 api 찍고

위치에 따라 날씨 뿌려주고

api 월케 좋아해?

-로그인 회원가입 =>

-git / db / 서버(aws) 서버는 마련했고 (700원) -> docker -> 컨테이너 서버 배포

= PL PM

PM

PMO

PL1 PL2 PL3

개발1,2 개발3,4

DBA 백엔드 프론트엔드

프로젝트 구성의 정석

전체적인 구조도(모듈 분해) - 테이블 구조도 - 화면정의서(설계서) UI가 섞인 기능 설계가 나와 -
개발자가 개발