**# Spring MVC**

- Spring 자체적으로 제공하는 MVC 프레임워크를 사용하면 Spring이 제공하는 AOP, 트랜잭션

처리, DI 등의 기능을 그대로 사용하면서 MVC 패턴에 기반하여 웹 어플리케이션을 개발할 수

있다.

**# Spring MVC의 구성**

1. DispatcherServlet : 클라이언트의 요청을 전달받아 요청에 맞는 컨트롤러가 리턴한 결과값

을 View에 전달하여 알맞은 응답을 생성

2. HandlerMapping : 클라이언트의 요청 URL을 어떤 컨트롤러가 처리할 지 결정

3. Controller : 클라이언트의 요청을 처리한 뒤, 결과를 DispatcherServlet에게 리턴

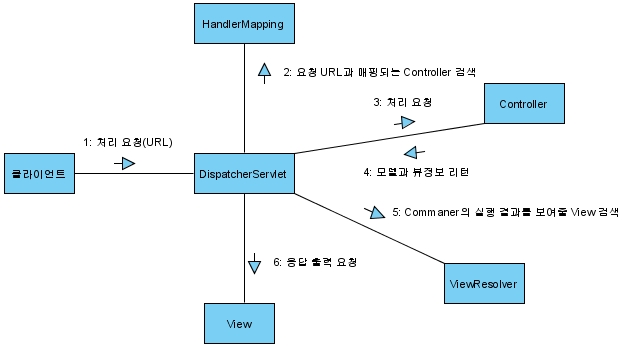
4. ModelAndView : 컨트롤러가 처리한 결과 정보 및 View 선택에 필요한 정보를 담음

5. ViewResolver : 컨트롤러의 처리결과를 생성할 View를 결정

6. View : 컨트롤러의 처리결과 화면을 생성. JSP 또는 Velocity 템플릿파일

등을 View로사용

**# Spring MVC 실행흐름**



1. 클라이언트의 요청이 DispatcherServlet에 전달된다.

2. DispatcherServlet은 HandlerMapping을 사용해 클라이언트의 요청이 전달될 Controller

객체를 구한다.

3. DispatcherServlet은 Controller 객체의 handleRequest() 메소드를 호출하여 클라이언트의

요청을 처리한다.

4. Controller.handleRequest() 메소드는 처리결과 정보를 담은 ModelAndView객체를 리턴

한다.

5. DispatcherServlet은 ViewResolver로부터 처리결과를 보여줄 View를 구한다.

6. View는 클라이언트에 전송할 응답을 생성한다.

-> 직접 개발해야 하는 부분 : 요청을 처리할 Commander클래스, 클라이언트에 응답결과 화

면을 전송할 JSP, Velocity 템플릿 등의 View 코드.

-> 나머지 DispatcherServlet, HandlerMapping, ViewResolver등은 Spring이 제공하는

기본구현체를 사용한다.

**# Spring MVC를 이용한 웹 어플리케이션 개발**

1. 클라이언트의 요청을 받을 DispatcherServlt을 web.xml파일에 설정

2. 요청 URL과 Controller의 매핑 방식 설정

3. Controller의 처리결과를 어떤 View로 보여줄 지의 여부를 결정하는 ViewResolver설정

4. Controller 작성

5. View 영역코드작성

**step 1) DispatcherServlet설정 및 스프링 컨텍스트 설정**

- web.xml에 DispatcherServlet설정 및 공통으로 사용할 어플리케이션 컨텍스트 설정

- <servlet> 태그이용해 DispatcherServlet설정

- <servlet-mapping>태그 이용해 요청 URL 패턴설정

[ web.xml ]

<servlet>

<servlet-name>**mapping name과 같은 이름**</servlet-name>

<servlet-class></servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**servlet name과 같은 이름**</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

**step 2) 컨트롤러 구현 및 설정추가**

1) @Controller 어노테이션을 클래스에 적용

2) @ReuestMapping어노테이션을 이용해서 클라이언트의 요청을 처리할 메서드를 지정

3) ModelAndView는 컨트롤러 처리결과를 보여줄 View, View에서 출력할 Model을 지정

4) DispatcherServlet은 스프링 컨테이너에서 컨트롤러 객체를 검색하기 때문에 스프링 설정

파일(xml)에 컨트롤러를 빈으로 등록 해야 한다.

**# 컨트롤러 클래스**

**@Controller**

public class 컨트롤러 {

**@RequestMapping("경로")**

public ModelAndView 함수명() {

**ModelAndView mv = new ModelAndView();**

mv.setViewName("View이름");

mv.addObject("message", "메시지");

return mv;

}

}

**step 3) 설정파일 View-Resolver 설정추가**

**# 서블릿이름-servlet.xml : bean 등록**

-> web.xml에서 <servlet-name> 태그에 정해줬던 이름을 써야함!

1) 컨트롤러에서 ModelAndViewName() 메서드로 뷰이름 지정

<bean id = "클래스이름" class = "뷰 클래스 경로" />

[ controller ] ModelAndView mav = ModelAndView();

mav.addObject("name", "kosta");

mav.setViewName("**intro**");

-> view 이름 : **intro** > ViewResolver 동작

-> prefix + view 이름 + suffix의 조합

-> 결과 : /WEB-INF/view/intro.jsp

2) 이 뷰 이름과 View구현체를 찾기 위해 ViewResolver사용

ViewResolver : view 설정을 담당하는 클래스

3) prefix프로퍼티 / suffix 프로퍼티 지정

**<property name = "prefix" value = "/WEB-INF/뷰" />**

**<property name = "suffix" value = ".jsp" />**

**step 4) View 코드구현**

------------------------------------------------------------------------------

**# DispatcherServlet설정과 ApplicationContext의 관계**

- web.xml 파일에 한 개 이상의 DispatcherServlet설정을 할 수 있으며, 각각 한 개의

WebApplicationContext를 갖는다.

**step 5) DispatcherServlet설정**

- 기본적으로 /WEB-INF/서블릿이름-servlet.xml 파일로부터 스프링 설정 정보를 읽어옴

- 한 개 이상 설정 파일을 사용할 경우 contextConfigLocation초기화 파라미터에 설정 파일 목

록을 지정함

[ ex ] 위의 코드에 추가

<servlet>

<servlet-name>**mapping name과 같은 이름**</servlet-name>

<servlet-class></servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/beans.xml

/WEB-INF/main.xml

</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>**servlet name과 같은 이름**</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

**step 6) ApplicationContext 설정**

- 한 개 이상의 DispatcherServlet사용할 때 별도의 WebApplicationContext를 생성

- 이 경우 공통 빈을 필요로 할 때 ContextLoaderListener를 사용해 사용될 빈을 설정

**step 7) Controller 구현**

step 1) @Controller 어노테이션, @RequestMapping 어노테이션

- 컨트롤러 클래스에 @Controller 어노테이션 적용

- 클라이언트의 요청을 처리할 메서드에 @RequestMapping어노테이션 적용

- xml 설정파일에 컨트롤러 클래스를 빈으로 등록

- @RequestMapping 어노테이션 : 해당 메서드에 처리할 URI를 값으로 갖는다.

step 2) 컨트롤러 메서드 HTTP 전송방식(method) 한정

- 하나의 요청 URI에 대해 HTTP GET 요청과 POST요청을 한 개의 컨트롤러에서 처리할 경우

- @RequestMapping어노테이션의 method 속성을 이용해서 메서드가 처리할 HTTP 메서드를 제한

**# 컨트롤러 클래스**

[ 방식 1 ]

@Controller

public class 컨트롤러 클래스명 {

@RequestMapping(method = RequestMethod.GET)

public 타입 메소드명 {

return "";

}

@RequestMapping(method = RequestMethod.POST)

public 타입 메소드명 {

return "";

}}

[ 방식 2 ] > 더 많이 씀

@Controler

@RequestMapping("/~~.~~")

public class 컨트롤러클래스 {

@RequestMapping(method=RequestMethod.GET)

public 타입 함수명 {

return "[~~]";

// /WEB-INF/하위폴더/ + [~~] + .jsp > 이 만들어진다 > forward

}

@RequestMapping(method=RequestMethod.POST)

1 public 타입 함수명(NewArticleCommand command)

// parameter로 DTO 객체를 사용한다. parameter 이름이 DTO의 멤버변수이름과 동일

하다면, 자동으로 setter가 된다.

2 public 타입 함수명(@ModelAttribute("Articledata")NewArticleCommand command) {

// @ModelAttribute 어노테이션을 : 자동으로 객체가 생성되고 이름 부여되는 것에 반

대해 직접 객체에 이름을 부여하는 방식

return "[~~]";}

-> 두 메서드가 동일한 URI를 처리하는 경우에 @RequestMapping 어노테이션을 클래스에 적용해

해당 클래스가 처리할 기본 URI를 지정가능!

-> method 속성을 설정하지 않으면 모든 HTTP 전송방식(GET, POST..)을 처리한다.

step 3) HTML Form, Command 객체

- HTML 폼에 입력한 데이터를 자바 빈 객체를 이용해서 전달받음

- HTML 폼의 항목이름과 자바 빈 클래스의 프로퍼티 이름이 일치할 경우엔 폼에 입력한 값을 프

로퍼티 값으로 설정

- @RequestMapping 어노테이션이 적용된 메서드의 파라미터로 자바 빈 타입을 추가

[ JSP ]

<form method = "post">

// form 태그에 action 속성이 없다면?????

// 명시하지 않아도 현재 주소창에 http://localhost:8090/상위폴더/하위폴더/~~.do를 갖고

있다면 위 주소에서 /하위폴더/~~.do가 자동으로 action값이 된다.

<input type = "" name = "" .. />

// input 태그의 name값은 DTO의 멤버변수 이름과 똑같이 쓴다.

</form>

[ Controller ]

@Controller

public class 클래스 {

@RequestMapping(method = Requestmethod.GET)

public 타입 메소드명 {

return "";

}

}

step 4) View에서 커맨드 객체 접근

- View에서 @RequestMapping어노테이션 메서드에서 전달받은 자바 빈 객체의 클래스명을 이용

해 접근가능(첫 글자 소문자!!)

step 5) 커맨드객체로 List 받기

- 폼에서 "프로퍼티명[인덱스].프로퍼티"같이 입력폼의 이름을 구성

[ JSP ]

<input type = "" name = "프로퍼티명[0].프로퍼티" .. />

**# @CookieValue어노테이션 이용 쿠키매핑**

- 쿠키 값을 파라미터로 전달 받을 때 사용

- 해당 쿠키가 존재하지 않으면 500 error 메시지

- 쿠키가 필수가 아닌 경우 파라미터와 같이 required 속성의 값을 false로 지정(기본:true)

값은 null로 저장

- 필수가 아닌 쿠키인 경우엔 defualtValue속성 값으로 기본값 지정

[ 컨트롤러 ]

@Controller

public class 클래스명 {

@RequestMapping(value = "cookie.do", method = ReuestMethod.GET)

public 타입 메소드명(

@CookieValue(value="name", required=false) String name,

@CookieValue(value="age", defaultValue="1") int age) {

return "cookie";

}

}

- @CookieValue의 value 속성은 가져올 쿠키의 이름!

- @CookieValue사용은 request.getCookie()와 같다. 이렇게 쓸 수 있지만 Spring에서 제공하는

@CookieValue를 사용하면 defaultValue옵션을 설정할 수 있다!

**# @RequestHeader어노테이션 이용 헤더 매핑**

- HTTP 요청헤더의 값을 메서드의 파라미터로 전달받을 때 사용

- 해당 헤더가 존재하지 않으면 500 error 메시지

[ 컨트롤러 ]

@Controller

public class 클래스명 {

@RequestMapping(value = "cookie.do", method = ReuestMethod.GET)

public 타입 메소드명(

@RequestHeader(value="Accept-Language", required=false) String languageHeader,

@RequestHeader(value="user-agent", defaultValue="모름") String agentHeader) {

return "header";

}

}

# 17 05 15 수업

1. SpringMVC WebSite생성하기

1.1 Spring Framework가 갖고있는 필요한 jar파일 추가하기

- bean, logging, core, context, web, web-mvc, aop, expression

-

1.2 web.xml 설정하기

- org.springframework.web.servlet.DispathcerServlet

단일 FrontController 설정하기

- url-pattern > \*.htm or \*.do or \*.action ...

1.3 DispatcherServlet read해서 사용할 DI 구성

- [servlet-name]-servlet.xml > 자동으로 read 됨

- 이 xml 안에서 bean 객체 생성 및 injection(constructor-arg / property)

- xml 기반 or 어노테이션 기반(<context:annotation-config>)

1.4 SpringMVC가 갖는 패턴 따라 Model, View, Controller 생성

- Model : DTO(VO or Domain) / DAO(DB 작업)

- View : EL & JSTL

- Controller :

[ 기존 - 코드 ]

public class NoticeController implements Controller {

handleRequest 함수 구현

}

[ 기존 - 문제점 ]

기능 하나 당 각각의 컨트롤러가 필요함

사용할 수 있는 함수가 handleRequest 하나 뿐

[ 해결 ]\*\*\*\*\*\*\*\*\*\*\*\*\*

클래스 하나가 여러 개의 기능을 처리!

함수 단위로 기능을 처리한다

: HandlerMapping - 함수단위로(함수가 mapping자원이 된다) ex)

@Controller

class Board {

@RequestMapping("/list.do")

listBoard(){}

@RequestMapping("/listDetail.do")

BoardInsert(){}

..

}

1.5 실습구성

- package : com.controller

- WEB-INF > view 폴더 생성

- default > WebContent > index.html

# @Controller

- spring MVC의 Controller 클래스 선언을 단순화시켜준다.

- 서블릿을 상속할 필요가 없으며, @Controller로 등록된 클래스 파일에 대한 bean을 자동으로

생성해준다.

- Controller로 사용하고자 하는 클래스에 @Controller 지정해 주면 component-scan으로

자동 등록된다.

- <context:component-scan base-package="com.controller"/>

# Annotation > 카페 870, 877번글 확인

# 아래 예제에 대한 실습 구성

- package : com.controller

- WEB-INF > view 폴더 생성

- default > WebContent > index.html

step 1) web.xml 설정

<servlet>

<servlet-name>***dispatcher***</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</>

</servlet

<servlet-mapping>

<servlet-name>***dispatcher***</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

step 2) dispatcher-servlet.xml 생성 및 설정

- [servlet-name]-servlet.xml 으로 이름을 주면 자동으로 read된다.

- 이 xml 파일에서 bean객체를 생성하고 조립한다.

- xml 기반 혹은 어노테이션 기반을 사용한다.

- 어노테이션 기반을 사용할 경우 <context:annotation-config>를 명시해준다.

step 2-1) @Autowired, @Resource 자동주입을 위한 태그

<context:annotaion-config />

step 2-2) 공통 작업

<bean id = "iternalResourceViewResolver" class = "org....">

**<property name = "prefix">**

<value>/WEB-INF/view/</value>

</property>

**<property name = "suffix">**

<value>.jsp</value>

</property>

</bean>

***만약 <context:component-scan base-package="com.controller" /> 설정을 위에서 해준다면***

***아래의 bean 객체를 만드는 코드는 필요없다.***

<bean id = "helloController" class = "com.controller.HelloController" />

<bean class = "com.controller.NewArticleController" />

<bean class = "com.service.ArticleService" />

***다른 곳에서 참조하지 않는다면 id는 쓰지않아도 된다. 클래스가 컨테이너 안에만 있으면 된다!***

step 3) SpringMVC 패턴따라 Model, View, Controller 생성

- Model : DTO(VO or Domain) / DAO

- View : EL & JSTL

- Controller

기존 실습에서는,

컨트롤러 클래스가 Controller인터페이스를 구현하고 그 안에서 handleRequest함수를

구현하는 방식을 이용했다.

여기서 문제점은,

기능 하나 당 각각의 컨트롤러가 필요하고 컨트롤러 클래스 안에서 사용할 수 있는 함수도

handleRequest() 하나 뿐이다. > 비효율 적임

이에 해결방안은

하나의 컨트롤러 클래스가 여러 개의 기능을 처리하는 방식을 이용하는 것이다.

@HandlerMapping 어노테이션을 사용하면 mapping자원이 함수가 되어 함수단위로 기능을

처리할 수 있다.

**[HelloController.java]**

**@Controller**

public class HelloController {

***@Controller를 갖는 클래스 안에서 함수 단위의 url-mapping***

@RequestMapping("/hello.do")

public ModelAndView hello() {

ModelAndView mv = new ModelAndView();

mv.addObject("greeting", getGreeting());

mv.setViewName("Hello");

***후에 ViewResolver통해서 조합된다.***

return mv;

}

***별도의 일반 함수의 구현이 가능하다***

public String getGreeting() {

..

}

}

**[NewArticleController.java]** : 글쓰기, 글쓰기 완료 기능을 갖는다

- form 태그 내에서 method = post로 명시하는 것 외에는 모두 GET 방식이다.

- 같은 URL에서 GET(form 화면), POST(입력 처리)방식이 모두 이뤄진다.

ex) 회원가입, 글쓰기 등

@Controller

**@RequestMapping("/article/NewArticle.do")**

***위 어노테이션은 클래스에도 걸린다!***

public class NewArticleController {

**@RequestMapping(method = RequestMethod.GET)**

***위에서 한번 걸러진 후 안에서 한번 더 걸러진다***

public String form() {

return "/article/newArticleForm";

***/WEB-INF/view/ + [] + .jsp > forward방식***

}

**@RequestMapping(method = RequestMethod.POST)**

1 public String submit(NewArticleCommand command) {

***파라미터로 DTO를 사용한다. 파라미터 이름이 DTO의 멤버변수 이름과 동일하다면,***

***자동으로 setter가 돌아 DTO가 생성되고 데이터가 insert된다.***

2 public String submit(@ModelAndView("Articledata")NewArticleCommand command) {

***객체에 직접 이름을 주고 싶다면 @ModelAndView 어노테이션을 사용한다.***

***@ModelAndView("객체이름")처럼 사용한다.***

// db에 insert하는 작업이 이뤄져야 함

return "article/newArticleSubmitted";

}

}

step 4) TEST\_3 HTTP전송-List collection 처리

**[dispatcher-servlet.xml]** : OrderController위한 객체 생성

<bean class = "com.controller.OrderController" />

**[OrderItem.java] & [OrderCommand.java]** : DTO / DAO

OrderItem :

private String itemid;

private int number;

private String remark;

OrderCommand :

private List<OrderItem> orderItem;

**[OrderForm.jsp]** : <Form>

<form method = "post">

***form 태그에서 action을 생략하면 현재 url이 submit주소가 된다.***

상품1<br>

상품ID : <input type = "text" name = "**orderItem[0].itemid**><br>

상품수량 : <input type = "text" name = "**orderItem[0].number**><br>

상품 주의사항 : <input type = "text" name = "**orderItem[0].remark**><br>

***input 태그의 name : orderItem[idx] > List memberfield의 이름과 동일해야한다.***

***List의 대상이 되는 memberfield > OrderItem에서 만든 각 변수의 이름과 동일해야 한다.***

상품2<br>

상품ID : <input type = "text" name = "**orderItem[1].itemid**><br>

상품수량 : <input type = "text" name = "**orderItem[1].number**><br>

상품 주의사항 : <input type = "text" name = "**orderItem[1].remark**><br>

<input type = "submit" value = "전송" />

</form>

**[OrderController.java]** : 주문서 클래스

- 시나리오 : 여러 개의 같은 물건을 주문할 수 있다

= 같은 타입을 여러 개 받는다 => Collection!

@Controller

@RequestMapping("/order/order.do")

public class OrderController {

@RequestMapping**(method=RequestMethod.GET)**

public String form() {

return "order/OrderForm";

***WEB-INF/view + [order/OrderForm] + .jsp***

}

@RequetMapping(**method=RequestMethod.POST**)

public String submit(**OrderCommand orderCommad**) {

***내부적으로 <bean id = "orderCommand ..> view에서 id값 read해서 출력한다***

return "order/OrderCommitted";

}

***submit() 코드를 풀면,***

***OrderCommand orderCommand = new OrderCommand();***

***List<OrderItem> list = new ArrayList<OrderItem>();***

***list.add(new OrderItem(1, 10, "A");***

***list.add(new OrderItem(2, 20, "B");***

***list.add(new OrderItem(3, 30, "C");***

***orderCommand.setOrderItem(list);***

***위 작업이 자동화 된다.***

step 5) TEST\_4 HTTP전송-parameter다루기 : **@RequestParam**

**[dispatcher-servlet.xml]** : SearchController위한 객체 생성

<bean class = "com.controller.SearchController" />

**[SearchType.java]** : DTO

**[SearchController.java]**

- parameter 넘어오는 것을 처리

- DTO 생성 후 form 태그의 name 속성값을 DTO memberfield 이름과 같게 한다.

@Controller

public class SearchController {

@RequestMapping("search/internal.do")

public ModelAndView SearchInternal(@RequestParam("query") String query,

@RequestParam("p") int p) {

***String query = request.getParameter("query")***

***int p = request.getParameter("p")***

return new ModelAndView("search/internal");

}

위 코드의 문제점은,

1. 파라미터 값이 없는 경우 -> 기본값을 가지게 한다.

- 옵셥 : required = false

> required를 false로 주면 external.do?p=100으로 들어올 경우 error!

- 옵션 : defaultValue = 1

문제 해결방법 1

@RequestMapping("/search/external.do")

public ModelAndView searchExternal(

@RequestParam(value ="query", defaultValue="title") String query,

@RequestParam(value = "p", required = false, defaultValue="10") int p

) {

return null;

}

문제 해결방법 2 : 위 보다 더 간단한 방법

규칙 : parameter 이름과 함수의 변수명이 동일하면 자동으로 받아온다.

단, default 옵션을 주지 않으므로 넘어오는 값이 없을 때의 오류처리는 안됨!!

넘어오는 parameter들의 값이 꼭!! 있어야 함!!

@RequestMapping("/search"external.do")

public ModelAndView searchExternal(String query, int p) {

return null;

}

step 6) TEST\_5 Spring Cookie 사용하기 : **@CookieValue**

**[dispatcher-servlet.xml]** : CookieController위한 객체 생성

<bean class = "com.controller.CookieController" />

**[CookieController.java]**

- 일반적으로 함수에서 request, response를 사용할 수 있다.

- make(HttpServletResponse res, HttpServletRequest req)

@Controller

public class CookieController {

@RequestMapping("/cookie/make.do")

public String make(HttpServletResponse response) {

**response.addCookie(new Cookie("auth", "1004"));**

return "cookie/CookieMake";

}

@RequestMapping("/cookie/view.do")

public String view(**@CookieValue(value = "auth", defaultValue = "0") String auth**) {

***value = "가져올 쿠키의 이름"***

***defaultValue를 주면 임의로 쿠키를 삭제해도 기본값이 찍힌다.***

return "cookie/CookieView";

}

step 7) TEST\_6 Spring Cookie 사용하기 : **@ModelAttribute(쓰임새정도만)**

**[dispatcher-servlet.xml]** : CodeSearchController위한 객체 생성

<bean class = "com.controller.CodeSearchController" />

**[CodeSearchController.java]**

- @ModelAttribute 사용법

1. public String submit(@ModelAttribute("Articledadata")NewArticleCommand command){}

> @ModelAttribute 생략:NewArticleCommand newArticleCommand = new NewArticleCommand();

>@ModelAttribute("Articledata"):NewArticleCommand Articledata=new ArticleCommand();

2. @ModelAttribute : 함수를 객체화 시킬 수 있다(함수 위에 선언)

@Controller

public class CodeSearchController {

**@ModelAttribute("SearchTypelist")**

***SearchTypeList 객체가 만들어지고 [View]에서 SearchTypeList라는 객체이름으로***

***사용가능하다.***

***>> 공통사용 메뉴구성, 미리 UI구성(코드형 데이터)에 쓸 수 있다!***

public List<SearchType> options = new ArrayList<SearchType>();

options.add(new SearchType(1, "전체"));

options.add(new SearchType(2, "LOL"));

options.add(new SearchType(3, "와우"));

options.add(new SearchType(4, "옵치"));

return options;

}

@ModelAttribute("querylist")

public String[] getQueryList() {

return new String[] {"게임", "star", "city", "world"};

}

step 8) TEST\_7 파일업로드

**[dispatcher-servlet.xml]** : ImageController위한 객체 생성

<bean class = "com.controller.ImageController" />

<bean id = "multipartResolver"

class="com.springframework.web.multipart.commons.CommonsMultipartResolver">

<property name

</bean>

**[ImageController.java]**

- 파일 업로드(CommonsMultipartFile) 전제 조건

1. lib안에 의존객체 추가

- fileupload.jar

- commons.io.jar

2. IOC 컨테이너 안에

CommonsMultipartResolver 객체 bean 생성

>파일 사이즈 , 임시 폴더 정의

3. DTO

file을 담을 수 있는 memberfield 생성

CommonsMultipartFile타입 kkk 생성

(getter , setter ) 구현

-멤버필드이름은 form태그 input의 name 값과 맞출 것

-나중에 파일 이름 처리(컨트롤러에서 처리) 해서 set해줄 멤버필드 하나 추가해서 구현할 것

4. UI > Form > input type="file" name="kkk"

해당 Form 태그 속성 구성 enctype추가 필요(파일 타입 read, write가능하도록)

<form action=*""* method=*"post"* enctype=*"multipart/form-data"*>

5. Controller 에서 DTO 타입(위에서 만든) parameter 사용

submit(Photo photo)