# model attribute 이름

<JSP>

<form:form method="post" modelAttribute="userRegistrationModel">

<Controller>

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

public String register(Model model, UserRegistrationModel userModel)

{

model.addAttribute("departments", departmentService.findAll());

      return "user/register";

}

@RequestMapping(value="register", method=RequestMethod.POST)

public String register(@Valid UserRegistrationModel userModel, BindingResult bindingResult, Model model)

{

        if (bindingResult.hasErrors()) {

        model.addAttribute("departments", departmentService.findAll());

           return"user/register";

   }

      userService.save(userModel);

   return "redirect:registerSuccess";

   }

(가)

@RequestMapping(....)

public String method1(Pagination pagiation) {

}

model attribute name = pagination

액션 메소드 파라미터가 객체일 경우에

그 객체는 자동으로 model에 add attribute 된다.

이때 그 이름(name)은 클래스 이름에서 첫글자만 소문자.

액션 메소드의 파라미터가 객체가 아니고, String 이거나 int 인 경우에는 반드시 이름을 지정해 주어야 함.

객체일 경우에는 @RequestPAram을 붙여도 되고, 생략해도 된다.

대부분의 경우에 생략함

예)model.addAttribute("pagination", pagination);

(나)

@RequestMapping(....)

public String method1(Pagination pagiation1) {

}

model attribute name = pagination

예)model.addAttribute("pagination",pagination1);

(다)

@RequestMapping(....)

public String method1(Pagination1 pagiation) {

}

model attribute name = pagination1

예)model.addAttribute("pagination1",pagination);

(라)

@RequestMapping(....)

Public String method1(@RequestParam("Pagination2") Pagination pagiation1) {

}

@RequestParam("model attribute의 이름")

model attribute name = Pagination2

예) model.addAttribute("Pagination2", pagination1);

Pagination : 클래스 이름

Pagination1 : Pagination 타입의 변수

@RequestParam : 객체 넘길 수 있음. 우리는 그동안 생략해왔던 것임.

@RequestParam("Pagination2") :

Spring web mvc엔진에 model에 자동으로 채워줄 때 이름을 객체의 앞만 소문자로 바꿔서 넣어준다. 하지만 종종 그 이름을 바꾸고싶을때가 있다. 그럴 때 @RequestParam("바꿔주고싶은 이름")으로 바꿔준다.

이것은 model의 이름이다.

# spring core(핵심) 기능

## Dependency Injection (DI) 기능

사용할 객체를 소스코드에서 직접 생성하지 않는다.

직접 생성의 예) StudentService studentService = new StudentService();

스프링 엔진이 생성하여 넣어 준다(inject).

예) @Autowired StudentService studentService;

이 기능이 좋은(핵심인) 이유.

프로젝트가 커지면, 수정할(유지보수할)일이 점점 많아진다.

처음에는 StudentService 클래스를 구현해서 잘 사용했는데,

요구사항이 변해서, StudentService 클래스를 상속해서,

StudentService2 클래스를 새로 구현하였다.

예)

class StudentService2 extends StudentService {

@Oveerride

public method1(...) { ... }

}

직접 생성하는 코드의 경우, 클래스 이름을 수정해야 한다.

StudentService studentService = new StudentService2();

이렇게 수정하는 것이, 프로젝트가 커지면 문제가 된다.

@Autowired StudentService studentService;

이 경우에는 수정하지 않아도 된다.

단 현재 프로젝트에 StudentService2 클래스, StudentService3 클래스 등이 있다면

이들이 모두 StudentSerivce의 자식 클래스라면,

어떤 클래스 객체를 생성하여 대입할지(autowired 할지) 설정해 주어야 한다.

설정 방법1) xml 설정 파일 (spring 3)

설정 방법2) java config 클래스 구현.

## AOP 기능 Aspect Oriented Programming

예를 들어,

메소드 앞 뒤로 log 메시지를 찍는 코드를 추가하고 싶은데,

메소드 앞 뒤로 begin transaction, commit 코드를 추가하고 싶은데,

소스 코드 수정 없이 추가할 수 잇는가?

설정만 바꿔서, 메소드 앞뒤에 소스코드를 자동으로 넣어주는 기능. AOP

예

@Transactional(...)

메소드에 위 어노테이션을 추가하면

그 메소드 앞에 begin tran, 끝에 commit이 자동 추가된다.