Spring MVC 06

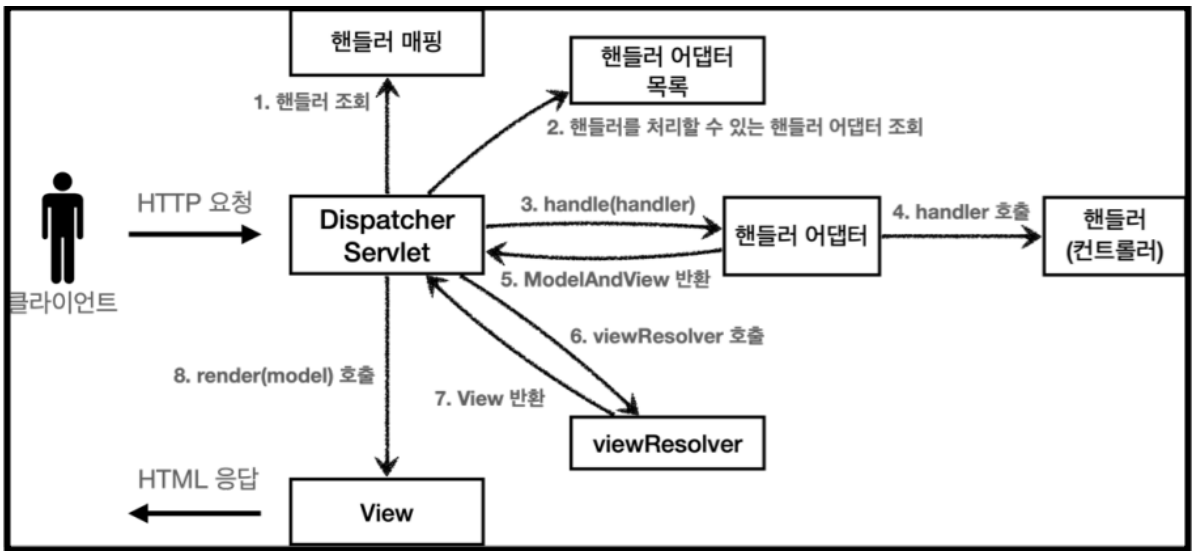
[스프링 **MVC** 구조이해]

스프링 MVC 1편 - 백엔드 웹 개발 핵심 기술

김영한

2022.08.22

스프링 **MVC** 전체 구조



FrontController 패턴을 개선시켜 온 모양과 똑같다**.**

Dispatcher Servlet

- 스프링 MVC도 프론트 컨트롤러 패턴으로 이뤄져있고 스프링 MVC의 프론트 컨트롤러가 DispatcherServlet이다.
- 스프링 부트는 DispatcherServlet을 서블릿으로 자동으로 등록하면서 모든경로(urlPatterns="/")에 대해 매핑한다

요청의 흐름

1. 핸들러 조회
2. 핸들러 어댑터 조회 - 핸들러를 처리할 수 있는 어댑터
3. 핸들러 어댑터 실행

4. 핸들러 어댑터로 해당 핸들러 실행
5. 핸들러 어댑터로 ModelAndView 반환
6. viewResolver 호출 (뷰 리졸버를 찾고 실행)
7. View 반환
8. 뷰 렌더링

모두 인터페이스로 제공 된다.

핸들러 매핑, 핸들러 어댑터, 뷰 리졸버, 뷰 모두 인터페이스로 제공된다.

요약

스프링 MVC는 코드 분량이 많고 구현이 많이 되어 있다. 실제로 나만의 컨트롤러를 만드는 일은 없다. 그래서 이미 웹 어플리케이션을 만들 때 필요로 하는 대부분의 기능이 이미 구현 되어있어서 잘 찾으면 모두 사용할 수 있다. 그럼에도 각각의 기능을 익히는 이유는 결국 문제가 생겼을때 문제가 생긴 부분을 알고 원활하게 해결하기 위해서이다.

핸들러 매핑& 핸들러 어댑터

과거에는 인터페이스 컨트롤러를 사용했다 (현재는 애노테이션을 사용)

스프링은 이미 필요한 핸들러 매핑과 핸들러 어댑터를 구현해 두었다.

HandlerMapping

핸들러 매핑에서 컨트롤러를 찾는다.

스프링 빈의 이름으로 핸들러를 찾을 수 있는 핸들러 매핑이 필요하다.

0 = RequestMappingHandlerMapping : 애노테이션 기반의 컨트롤러인 @RequestMapping에서 사용

1 = BeanNameUrlHandlerMapping : 스프링 빈의 이름으로 핸들러를 찾는다.

HandlerAdapter

핸들러 매핑을 통해서 찾은 핸들러를 구현해둔다.

0 = RequestMappingHandlerAdapter : 애노테이션 기반의 컨트롤러인 @RequestMapping에서 사용

1 = HttpRequestHandlerAdapter : HttpRequestHandler 처리

2 = SimpleControllerHandlerAdapter : Controller 인터페이스(애노테이션X, 과거에 사용) 처리

— 핸들러 매핑, 핸들러 어댑터 모두 위에 번호 순서대로 조회 후 해당하면 실행한다.

핸들러 매핑으로 핸들러 조회 → 핸들러 어댑터 조회 → 디스패처 서블릿이 핸들러 어댑터 실행 → 핸들러를 실행한다.

@RequestMapping이 가장 우선순위가 높은 핸들러 매핑이다**.**

뷰 리졸버

스프링은 InternalResourceViewResolver 라는 뷰 리졸버를 자동 등록해주는데 그때 설정 파일에서 prefix suffix를 지정해 줄 수 있다.

스프링 부트가 자동으로 등록하는 뷰 리졸버

1 = BeanNameViewResolver : 빈 이름으로 뷰를 찾아서 반환한다. (예: 엑셀 파일 생성 기능에 사용)

2 = InternalResourceViewResolver : JSP를 처리할 수 있는 뷰를 반환한다.

...더 많이 존재 .

InternalResourceView 는 랜더를 호출할 수 있고 내부의 .jsp를 실행한다. (forward 실행)

다른 뷰는 실제 뷰를 렌더링을 하지만 **JSP** 는 **forward()** 통해서 해당 **JSP**로 이동해야 랜더링이 된다.

스프링 **MVC** 시작하기

@RequestMapping

- @RequestMappingHandlerMapping
- @RequestMappingHandlerAdapter

가장 우선 순위가 높은 핸들러매핑과 핸들러어댑터

V1 - 컨트롤러 따로 생성

저장하기

```
@Controller 컴포넌트가 속해 있기 때문에 컴포넌트 스캔대상이 되고 스프링 빈에 등록한다.
            애노테이션 기반 컨트롤러로 인식한다.
            ( @Component @RequestMapping) 으로도 사용가능하긴 하다.
public class SpringMemberFormControllerV1 {

   @RequestMapping("/springmvc/v1/members/new-form")
   요청 정보를 매핑한다. 애노테이션 기반이기 때문에 매소드이름은 임의로 가능한다.
   public ModelAndView newForm(){
       return new ModelAndView("new-form");
   }
}
```

목록보기 컨트롤러

```
@Controller
public class SpringMemberListControllerV1 {
```

```
    private MemberRepository memberRepository = MemberRepository.getInstance();

    @RequestMapping("/springmvc/v1/members")
    public ModelAndView list(HttpServletRequest request, HttpServletResponse
response) {
        List<Member> members = memberRepository.findAll();
        ModelAndView mv = new ModelAndView("members");
        mv.addObject("members", members);
        return mv;
    }
}
```

V2 - 컨트롤러 통합

@RequestMapping을 한곳으로 통합

```
@Controller
@RequestMapping("/springmvc/v2/members”)
public class SpringMemberControllerV2 {

    private MemberRepository memberRepository = MemberRepository.getInstance();

    @RequestMapping("/new-form")
    public ModelAndView newForm(){
        . . . 통합하기 전과 동일
    }

    @RequestMapping("/save")
    public ModelAndView save(HttpServletRequest request, HttpServletResponse
response) {
        . . . 통합하기 전과 동일
    }

    @RequestMapping
    public ModelAndView list(HttpServletRequest request, HttpServletResponse
response) {
        . . . 통합하기 전과 동일
    }
}
```

@RequestMapping 클래스 부분과 매소드 부분이 통합된다 **.**

즉, **save** 부분은 **@RequsetMapping("/springmvc/v2/members/save")** 로 매핑되는 것

V3 - 실용적인 컨트롤러

모델엔 뷰를 직접 반환하면 불편하다.

실무에서는 주로 이 방식을 사용하게 된다.

```
@Controller
@RequestMapping("/springmvc/v3/members")
public class SpringMemberControllerV3 {

   private MemberRepository memberRepository = MemberRepository.getInstance();

   @GetMapping("/new-form")
   public String newForm(){
       return "new-form";
   }

   @PostMapping("/save")
   public String save(
           @RequestParam("username") String username,
           @RequestParam("age") int age,
           Model model) {

       Member member = new Member(username, age);
       memberRepository.save(member);

       model.addAttribute("member", member);
       return "save-result";
   }

   @GetMapping
   public String list(Model model) {
       List<Member> members = memberRepository.findAll();
       model.addAttribute("members", members);
       return "members";
   }
}
```

@RequestParam - get, Post 모두 쿼리 파라미터 모양이기 때문에 사용가능