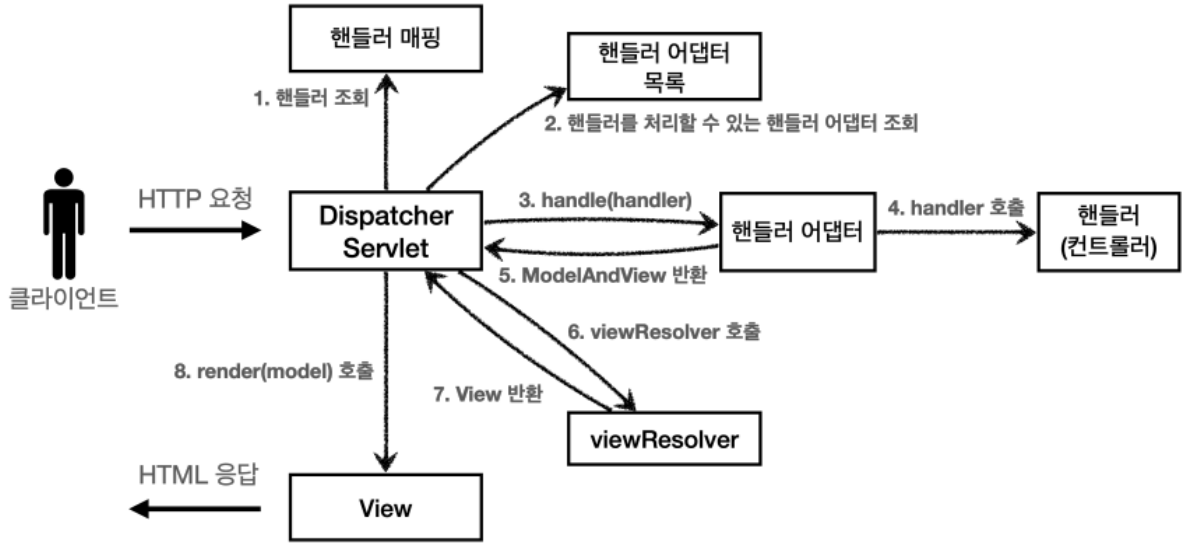
Section5. 스프링 MVC - 구조 이해

스프링 MVC 전체 구조



DispatcherServlet 구조 살펴보기

스프링 MVC의 프론트 컨트롤러가 바로 `DispatcherServlet` 이다.

DispatcherServlet 등록

- `DispatcherServlet` 도 부모 클래스에서 `HttpServlet` 을 상속 받아 사용하고, 서블릿으로 동작한다.
- 스프링 부트는 `DispatcherServlet` 을 서블릿으로 자동으로 등록하면서 모든경로(`urlPatterns="/"`)에 대해서 매핑한다.
  - 더 자세한 경로의 우선순위가 높다.

요청 흐름

- 서블릿이 호출되면 `HttpServlet` 이 제공하는 `service()` 가 호출된다.
- 스프링 MVC는 '`DispatcherServlet`' 의 부모인 '`FrameworkServlet`' 에서 '`service()`' 를 오버라이드 해두었다.

- '`FrameworkServlet.service()`' 를 시작으로 여러 메서드가 호출되면서 '`DispatcherServlet.doDispatch()`' 가 호출된다.

동작 순서

1. **핸들러 조회**: 핸들러 매핑을 통해 요청 URL에 매핑된 핸들러(컨트롤러)를 조회한다.
2. **핸들러 어댑터 조회**: 핸들러를 실행할 수 있는 핸들러 어댑터를 조회한다.
3. **핸들러 어댑터 실행**: 핸들러 어댑터를 실행한다.
4. **핸들러 실행**: 핸들러 어댑터가 실제 핸들러를 실행한다.
5. **ModelAndView 반환**: 핸들러 어댑터는 핸들러가 반환하는 정보를 `ModelAndView` 로 **변환**해서 반환한다.
6. **viewResolver 호출**: 뷰 리졸버를 찾고 실행한다.
7. **View 반환**: 뷰 리졸버는 뷰의 논리 이름을 물리 이름으로 바꾸고, 렌더링 역할을 담당하는 뷰 객체를 반환한다.
8. **뷰 렌더링**: 뷰를 통해서 뷰를 렌더링 한다.

인터페이스 살펴보기

- '`DispatcherServlet`' 코드의 변경 없이, 원하는 기능을 변경하거나 확장할 수 있다는 점이다.
- 이 인터페이스들만 구현해서 `DispatcherServlet` 에 등록하면 여러분만의 컨트롤러를 만들 수도 있다.

> 💡 **※ 주요 인터페이스 목록 ※**
>
> 핸들러 매핑: org.springframework.web.servlet.HandleMapping
> 핸들러 어댑터: org.springframework.web.servlet.HandlerAdapter
> 뷰 리졸버: org.springframework.web.servlet.ViewResolver
> 뷰: org.springframework.web.servlet.View

핸들러 매핑과 핸들러 어댑터

지금은 전혀 사용하지 않지만, 과거에 주로 사용

Controller 인터페이스 실행

💡 Controller 인터페이스는 @Controller 와 전혀 다르다.

```
import org.springframework.web.servlet.mvc.Controller;

@Component("/springmvc/old-controller")
public class OldController implements Controller {
    @Override
    public ModelAndView handleRequest(HttpServletRequest requ
        System.out.println("OldController.handleRequest");
        return null;
    }
}
```

해당 컨트롤러가 호출되려면 2가지가 필요하다.

- **HandlerMapping**
  - 핸들러 매핑에서 이 컨트롤러를 찾을 수 있어야 한다.
- **HandlerAdapter**
  - 핸들러 매핑을 통해 찾은 핸들러를 실행할 수 있는 핸들러 어댑터 필요

개발자가 직접 핸들러 매핑과 핸들러 어댑터를 만드는 일은 거의 없다.

스프링 부트가 자동 등록하는 핸들러 매핑과 핸들러 어댑터

- **HandlerMapping**

```
0 = RequestMappingHandlerMapping : 애노테이션 기반의 컨트롤러인 @Re
1 = BeanNameUrlHandlerMapping : 스프링 빈의 이름으로 핸들러를 찾는다
```

- **HandlerAdapter**

```
0 = RequestMappingHandlerAdapter : 애노테이션 기반의 컨트롤러인 @Re
1 = HttpRequestHandlerAdapter : HttpRequestHandler 처리
2 = SimpleControllerHandlerAdapter : Controller 인터페이스(애노터
```

정리

'**OldController**' 를 실행하면서 사용된 객체는 다음과 같다.

- `HandlerMapping` = `BeanNameUrlHandleMapping`
- `HadnlerAdapter` = `SimpleControllerHandleAdapter`

HttpRequestHandler 인터페이스 실행

```
@Component("/springmvc/request-handler")
public class MyHttpRequestHandler implements HttpRequestHandl
    @Override
    public void handleRequest(HttpServletRequest request, Htt
        System.out.println("MyHttpRequestHandler.handleReques
    }
}
```

정리

'**MyHttpRequestHandler**' 를 실행하면서 사용된 객체

- `HandlerMapping` = `BeanNameUrlHandleMapping`
- `HadnlerAdapter` = `HttpReequestHandlerAdapter`

💡 **※ @RequestMapping ※**

가장 우선순위가 높은 핸들러 매핑과 어댑터는

@RequestMapping의 앞글자를 따서 만든

`RequestMappingHandlerMapping`

`RequestMappingHandlerAdapter`

↑ 스프링에서 주로 사용하는 애노테이션 기반 매핑, 어댑터이다.

뷰 리졸버

기존 `OldController` 에 다음과 같은 `return` 값을 넣어보자.

```
return new ModelAndView("new-form");
```

- Whitelabel Error page 표시, 출력메서드는 실행된다.

  → 컨트롤러는 정상 호출

```
// application.properties 추가
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

뷰 리졸버 - internalResourceViewResolver

스프링은 `internalResourceViewResolver` 라는 view resolver를 자동 등록

- properties 에서 설정한 `prefix`, `suffix` 설정 정보를 사용해서 등록

```
@ServletComponentScan // 서블릿 자동 등록
@SpringBootApplication
public class ServletApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServletApplication.class, args)
    }

    @Bean
    InternalResourceViewResolver internalResourceViewResolver
        return new InternalResourceViewResolver("/WEB-INF/vie
    }
}
```

- 위 코드와 같이 직접 `@Bean` 으로 추가하는 것을 권장하지는 않는다.

스프링 부트가 자동 등록하는 뷰 리졸버

1. **핸들러 어댑터 호출**
   - 핸들러 어댑터를 통해 “new-form” 논리 뷰 이름 획득
2. **ViewResolver 호출**

- new-form 이라는 뷰 이름으로 viewResolver를 순서대로 호출
- `BeanNameViewResolver` 는 new-form 이라는 이름의 스프링 빈으로 등록된 뷰를 찾는데 없는경우
- `InternalResourceViewResolver` 호출

3. **InternalResourceViewResolver**
   - 이 뷰 리졸버는 InternalResourceView 를 반환
4. **뷰 - InternalResourceView**
   - JSP 처럼 **포워드**를 호출해서 처리할 수 있는 경우에 사용
5. **view.render()**
   - `view.render()` 가 호출되고, `InternalResourceView`는 `forward()` 를 사용해서 JSP 실행

> 💡 다른 뷰는 실제 뷰를 렌더링하지만, JSP의 경우 forward() 통해서 해당 JSP 이동해야 렌더링이 된다. 나머지 뷰 템플릿들은 forward() 과정 없이 바로 렌더링 된다.

스프링 MVC - 시작하기

@RequestMapping

가장 우선순위가 높은 매핑과 핸들러 어댑터

- `RequestMappingHandlerMapping`
- `RequestMappingHandlerAdapter`

```
//Component 컴포넌트 스캔을 통해 스프링 빈으로 등록
//RequestMapping 스프링 3.0 이상부터는 오직 controller 만을 인식할 수

@Controller
public class SpringMemberFormControllerV1 {
    @RequestMapping("/springmvc/v1/members/new-form")
    public ModelAndView process(){
        return new ModelAndView("new-form");
```

```
    }
}
```

- **@Controller**
  - 스프링이 자동으로 **스프링 빈으로 등록**한다.
  - 스프링 MVC에서 애노테이션 기반 **컨트롤러로 인식**한다.
- **@RequestMapping**
  - 요청 정보를 매핑한다. 해당 URL이 호출되면 이 메서드가 호출된다.
- **@ModelAndView**
  - 모델과 뷰 정보를 담아서 반환하면 된다.

```
@Controller
public class SpringMemberSaveControllerV1 {
  private MemberRepository memberRepository = MemberRepositor
  @RequestMapping("/springmvc/v1/members/save")
  public ModelAndView process(Map<String, String> paramMap) {
      String username = paramMap.get("username");
      int age = Integer.parseInt(paramMap.get("age"));

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

      ModelAndView mv = new ModelAndView("save-result");
      mv.addObject("member", member);
      return mv;
}
```

- `mv.addObject("member", member)`
  - 이 데이터는 이후 뷰를 렌더링 할 때 사용된다.

스프링 MVC - 컨트롤러 통합

`@RequestMapping`을 잘 보면 클래스 단위가 아니라 **메서드 단위에 적용**된 것을 확인할 수 있다.

클래스 단위로 묶는 경우 (`클래스MappingUrl + 메서드MappingUrl`) 조합 가능

```
@Controller
@RequestMapping("/springmvc/v2/members")
public class SpringMemberControllerV2 {
    private MemberRepository memberRepository = MemberReposit
    @RequestMapping("/new-form")
    public ModelAndView newForm(){
        return new ModelAndView("new-form");
    }

    @RequestMapping("/save")
    public ModelAndView save(HttpServletRequest request, Https
        String username = request.getParameter("username");
        int age = Integer.parseInt(request.getParameter("age"

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

        ModelAndView mv = new ModelAndView("save-result");
        mv.addObject("member", member);
        return mv;
    }

    @RequestMapping
    public ModelAndView members() {
        List<Member> members = memberRepository.findAll();

        ModelAndView mv = new ModelAndView("members");
        mv.addObject("members", members);
        return mv;
    }
}
```

스프링 MVC - 실용적인 방식

```
/**
 * v3
```

```
 * Model 도입
 * ViewName 직접 반환
 * @RequestParam 사용
 * @RequestMapping -> @GetMapping, @PostMapping
 */
@Controller
@RequestMapping("/springmvc/v3/members")
public class SpringMemberControllerV3 {
    private MemberRepository memberRepository = MemberReposit
    @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 members(Model model) {
        List<Member> members = memberRepository.findAll();

        model.addAttribute("members", members);
        return "members";
    }
}
```

Model 파라미터

`save()`, `members()` 를 보면 `Model`을 파라미터로 받는 것을 확인할 수 있다.

ViewName 직접 반환

HTTP 요청 파라미터를 `@RequestParam` 으로 받을 수 있다.

`@RequestMapping` → `@GetMapping`, `@PostMapping`

```
@RequestMapping(value = "/new-form", method = RequestMethod.[
// 이것을 @GetMapping , @PostMapping 으로 더 편리하게 사용할 수 있다
// Get, Post, Put, Delete, Patch 모두 애노테이션이 준비되어 있다.
```