Servlet to Spring MVC

Servlet 부터 Spring MVC 까지

Servlet SpringMVC DispatcherServlet FrontController 김영한

hyeonZIP
4 days ago · 16 min read

> *“A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.”*

서블릿은 **request**와 **response model**을 통해 접근하는 애플리케이션 서버의 기능을 확장하는 데 사용되는 자바 프로그래밍 언어 클래스이다.

쉽게 말해, 서블릿은 지정한 URL이 호출되면 HTTP 요청과 응답 정보를 각각 **HttpServletRequest** 객체와 **HttpServletResponse** 객체로 할당하여 요청 정보와 응답 정보를 해체(?),조립(?) 해준다.

HttpServletRequest

HTTP 요청 메세지를 파싱해 HttpServletRequest에 담아준다.

HTTP 요청 메세지를 편리하게 사용하도록 도와주는 객체이다.

```
# START LINE
POST /save HTTP/1.1
# HEADER
Host: localhost:8080 Content-Type: application/x-www-form-urlencoded
```

```
# BODY
username=kim&age=20
```

HTTP 요청 3가지

GET - 쿼리 파라미터

- ex) `http://localhost:8080/request-param?username=hello&age=20`
- Body없이 URL의 **쿼리 파라미터**에 데이터를 담아 요청한다.
- Body가 없기 때문에 `content-type = null` 이다.
- 주로 **필터**, **페이징**, **조회**에 사용된다.

POST - HTML FORM

- ex) `username=hello&age=20`
- Body에 **쿼리 파라미터**를 담아 요청한다.
- Body가 있기 때문에 `content-type = application/x-www-form-urlencoded` 이다.
- 주로 **회원 가입**, **상품 주문**에 사용된다.

HTTP Message Body

- Body에 TEXT 또는 JSON 형식으로 데이터를 담아 요청한다.
- 많이 익숙한 POST, PUT, PATCH가 이곳에 해당한다.

```
@WebServlet(name = "requestBodyJsonServlet", urlPatterns = "/request-body-
string")
public class RequestBodyJsonServlet extends HttpServlet {

    private ObjectMapper objectMapper = new ObjectMapper();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
```

```
    response) throws ServletException, IOException {
            ServletInputStream inputStream = request.getInputStream();
            String messageBody = StreamUtils.copyToString(inputStream,
    StandardCharsets.UTF_8);

            System.out.println("messageBody = " + messageBody);

            HelloData helloData = objectMapper.readValue(messageBody,
    HelloData.class);

            System.out.println("helloData = " + helloData.getUsername());
            System.out.println("helloData = " + helloData.getAge());

            response.getWriter().write("ok");
        }
    }
```

HttpServletResponse

TEXT, HTML 코드를 반환할 수 있지만 주로 JSON 형식을 반환한다.

```
    @WebServlet(name = "responseJsonServlet", value = "/response-json")
    public class ResponseJsonServlet extends HttpServlet {

        private ObjectMapper objectMapper = new ObjectMapper();
        @Override
        protected void service(HttpServletRequest request, HttpServletResponse
    response) throws IOException {
            //Content-Type: application/json
            response.setContentType("application/json");
            response.setCharacterEncoding("utf-8");
```

```
        HelloData helloData = new HelloData();
        helloData.setUsername("kim");
        helloData.setAge(20);

        //{"username":"kim","age":20}
        String result = objectMapper.writeValueAsString(helloData);

        response.getWriter().write(result);
    }
}
```

순수 Servlet의 문제점

```
//회원 정보 저장 요청을 받고 저장한 다음, 성공 화면을 응답한다.
@WebServlet(name = "memberSaveServlet", value = "/servlet/members/save")
public class MemberSaveServlet extends HttpServlet {

    private MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        System.out.println("MemberSaveServlet.service");
        String username = request.getParameter("username");
        int age = Integer.parseInt(request.getParameter("age"));

        Member member = new Member(username,age);

        memberRepository.save(member);

        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        PrintWriter w = response.getWriter();
```

```
            w.write("<html>\n" +
                    "<head>\n" +
                    "    <meta charset=\"UTF-8\">\n" +
                    "</head>\n" +
                    "<body>\n" +
                    "성공\n" +
                    "<ul>\n" +
                    "    <li>id="+member.getId()+"</li>\n" +
                    "    <li>username="+member.getUsername()+"</li>\n" +
                    "    <li>age="+member.getAge()+"</li>\n" +
                    "</ul>\n" +
                    "<a href=\"/index.html\">메인</a>\n" +
                    "</body>\n" +
                    "</html>");
        }
    }
```

현재 코드에서는 BE(Back end) 코드와 FE(Front end) 코드가 한 메서드 안에 존재한다.

따라서 이를 분리하기 위한 **MVC 패턴**이 도입되게 된다.

FE 코드를 위한 **View**

Servlet의 요청과 응답을 위한 **Controller**와

Controller에서 View로 데이터를 넘겨주기 위한 **Model**로 개념을 나눈다.

원시 MVC 패턴의 문제점

```
@WebServlet(name = "mvcMemberSaveServlet", value = "/servlet-mvc/members/save")
public class MvcMemberSaveServlet extends HttpServlet {
```

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

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        String username = request.getParameter("username");
        int age = Integer.parseInt(request.getParameter("age"));

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

        //Model에 데이터를 보관
        request.setAttribute("member",member);

        //해당 위치에
        String viewPath = "/WEB-INF/views/save-result.jsp";
        RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
        dispatcher.forward(request, response);
    }
}
```

먼저 View 로직은 jsp 코드로 `viewPath` 위치에 두고 `dispatcher.forward` 를 통해 Servlet의 HttpServletRequest와 HttpServletResponse를 보내도록 했다.

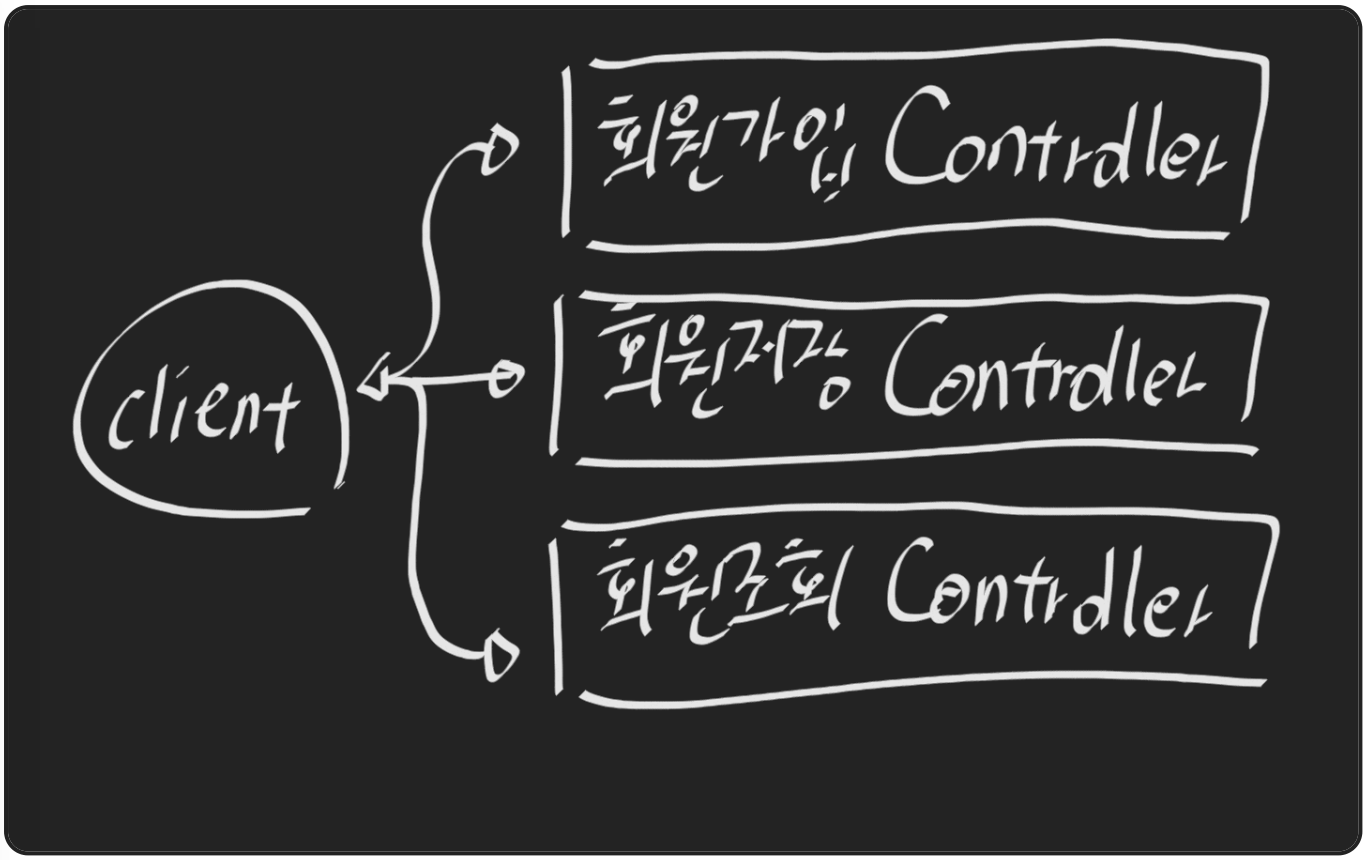
하지만 이 코드에서의 문제점은 하단의 3줄이다.

모든 요청에 대해 해줘야 하는 작업이기 때문에 반복되는 코드가 발생한다.

Java가 제일 싫어하는 것이 **반복하는 작업**이기 때문에 이를 해결하기 위해 Controller와 Client의 사이에 중재자 역할을 하는 것이 등장하고 이를 FrontControllerPattern 이라고 한다.

FrontController

V0 - 초기



초기 형태는 위 그림과 같다. 클라이언트의 요청이 바로 Controller로 가고

Controller에서 Client로 HTML을 보내고 있다.

먼저 FrontController가 들어갈 공간부터 만들어 보자.

V1 - FrontController & View 도입

FrontController

```
@WebServlet(name = "FrontControllerServletV1", urlPatterns = "/front-
controller/v1/*")
public class FrontControllerServletV1 extends HttpServlet {

    private Map<String, ControllerV1> controllerMap = new HashMap<>();

    public FrontControllerServletV1() {
```

```
        controllerMap.put("/front-controller/v1/members/new-form"
                , new MemberFormControllerV1());
        controllerMap.put("/front-controller/v1/members/save"
                , new MemberSaveControllerV1());
        controllerMap.put("/front-controller/v1/members"
                , new MemberListControllerV1());
    }
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String requestURI = request.getRequestURI();

        ControllerV1 controller = controllerMap.get(requestURI);
        if (controller == null){
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            return;
        }

        controller.process(request,response);
    }
}
```

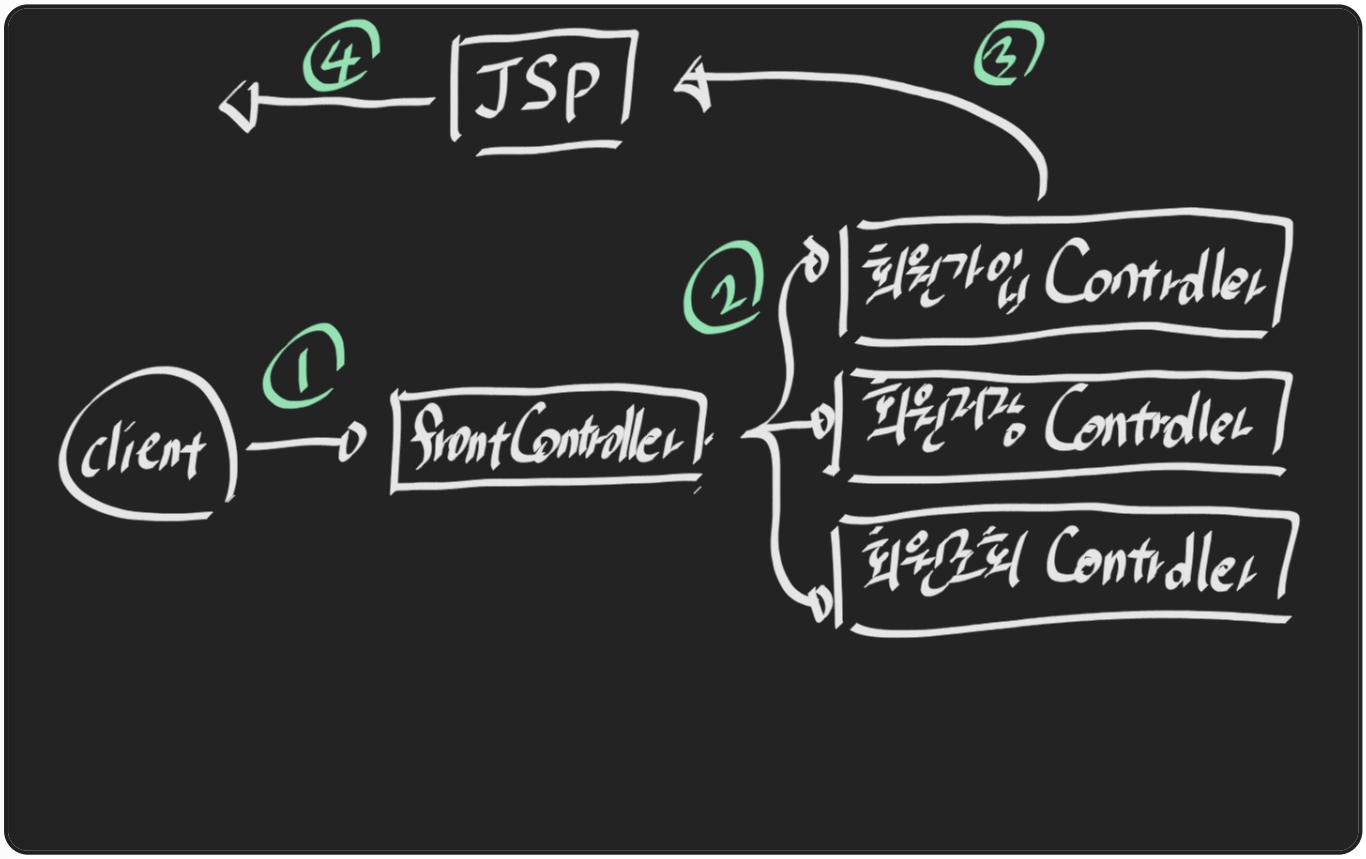
SaveController

```
public class MemberSaveControllerV1 implements ControllerV1 {

    private MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    public void process(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String username = request.getParameter("username");
        int age = Integer.parseInt(request.getParameter("age"));

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

```
            //Memeber라는 data를 request에 담아서 전송
            request.setAttribute("member",member);

            String viewPath = "/WEB-INF/views/save-result.jsp";
            RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
            dispatcher.forward(request, response);
        }
    }
```

`FrontController` 의 생성자 코드를 자세히보면 HashMap으로 각 3개의 Controller의 객체 인스턴스를 생성자 방식으로 받고 있다.

다음으로 RequestDispatcher 부분을 분리한다.



V2 - MyView

FrontController

```
@WebServlet(name = "frontControllerServletV2", urlPatterns = "/front-
controller/v2/*")
public class FrontControllerServletV2 extends HttpServlet {
    private Map<String, ControllerV2> controllerMap = new HashMap<>();

    public FrontControllerServletV2() {
        controllerMap.put("/front-controller/v2/members/new-form"
                , new MemberFormControllerV2());
        controllerMap.put("/front-controller/v2/members/save"
                , new MemberSaveControllerV2());
        controllerMap.put("/front-controller/v2/members"
                , new MemberListControllerV2());
    }

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String requestURI = request.getRequestURI();

        ControllerV2 controller = controllerMap.get(requestURI);
        if (controller == null) {
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            return;
        }

        MyView view = controller.process(request, response);
        view.render(request, response);
    }
}
```

SaveController

```
public class MemberSaveControllerV2 implements ControllerV2 {

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

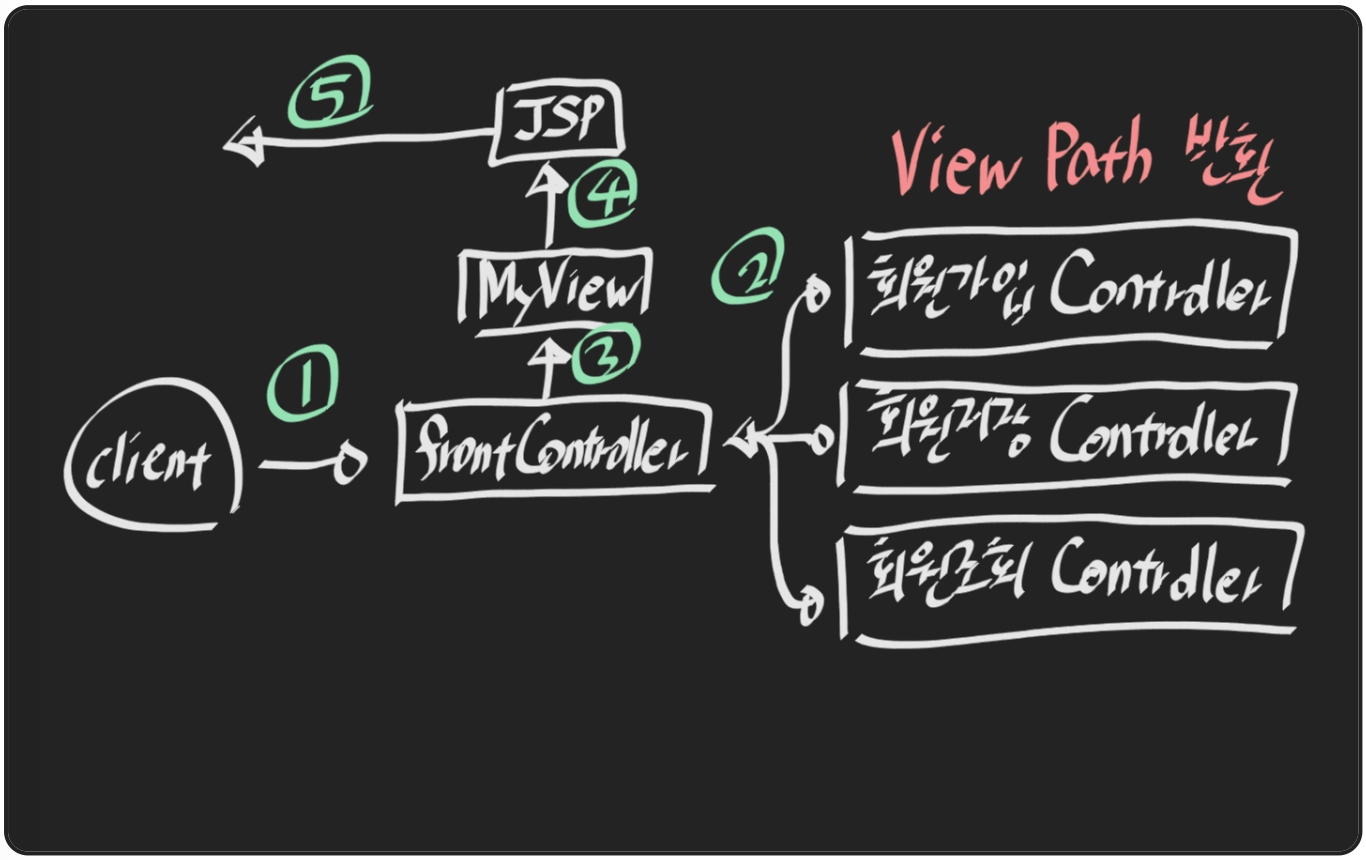
```
    @Override
    public MyView process(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String username = request.getParameter("username");
        int age = Integer.parseInt(request.getParameter("age"));

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

        request.setAttribute("member",member);

        return new MyView("/WEB-INF/views/save-result.jsp");
    }
}
```

MyView

```
public class MyView {
    private String viewPath;

    public MyView(String viewPath) {
        this.viewPath = viewPath;
    }

    public void render(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
        RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
        dispatcher.forward(request, response);
    }
}
```

기존 Controller에서 `RequestDispatcher` 부분을 `MyView` 클래스로 분리하고 `FrontController` 에서 호출하도록 변경하였다.

하지만 `return new MyView("/WEB-INF/views/save-result.jsp");` 이 부분에서 고정적인 부분인 `/WEB-INF/views .... .jsp` 이 부분을 분리 시킴과 동시에 Controller로부터 HttpServeltRequest & Response를 분리 시킨다.



V3 - ModelView & ViewResolver

FrontController

```
@WebServlet(name = "frontControllerServletV3", urlPatterns = "/front-
controller/v3/*")
public class FrontControllerServletV3 extends HttpServlet {
    private Map<String, ControllerV3> controllerMap = new HashMap<>();

    public FrontControllerServletV3() {
        controllerMap.put("/front-controller/v3/members/new-form"
                , new MemberFormControllerV3());
        controllerMap.put("/front-controller/v3/members/save"
                , new MemberSaveControllerV3());
        controllerMap.put("/front-controller/v3/members"
                , new MemberListControllerV3());
```

```
    }

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String requestURI = request.getRequestURI();

        ControllerV3 controller = controllerMap.get(requestURI);
        if (controller == null){
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            return;
        }

        //paramMap
        Map<String, String> paramMap = createParamMap(request);
        ModelView mv = controller.process(paramMap);

        //new-form
        String viewNam = mv.getViewName();
        MyView view = viewResolver(viewNam);

        view.render(mv.getModel(), request,response);
    }

    private static MyView viewResolver(String viewNam) {
        return new MyView("/WEB-INF/views/" + viewNam + ".jsp");
    }

    private static Map<String, String> createParamMap(HttpServletRequest
request) {
        Map<String,String> paramMap = new HashMap<>();
        request.getParameterNames()
                .asIterator()
                .forEachRemaining(paramName ->
                        paramMap.put(paramName, request.getParameter(paramName))
                );
        return paramMap;
```

```
        }
    }
```

SaveController

```
public class MemberSaveControllerV3 implements ControllerV3 {

    private MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    public ModelView 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);

        ModelView mv = new ModelView("save-result");
        mv.getModel().put("member", member);
        return mv;
    }
}
```

ModelView

```
public class ModelView {
    private String viewName;
    private Map<String,Object> model = new HashMap<>();

    public ModelView(String viewName) {
        this.viewName = viewName;
    }

    public String getViewName() {
        return viewName;
    }
```

```
        public Map<String, Object> getModel() {
            return model;
        }

        public void setModel(Map<String, Object> model) {
            this.model = model;
        }
    }
```

앞서 예고했듯 Controller에서 Servlet의 의존성을 제거했다. 대신 `paramMap` 이라는 파라미터가 FrontController로부터 넘어왔다.

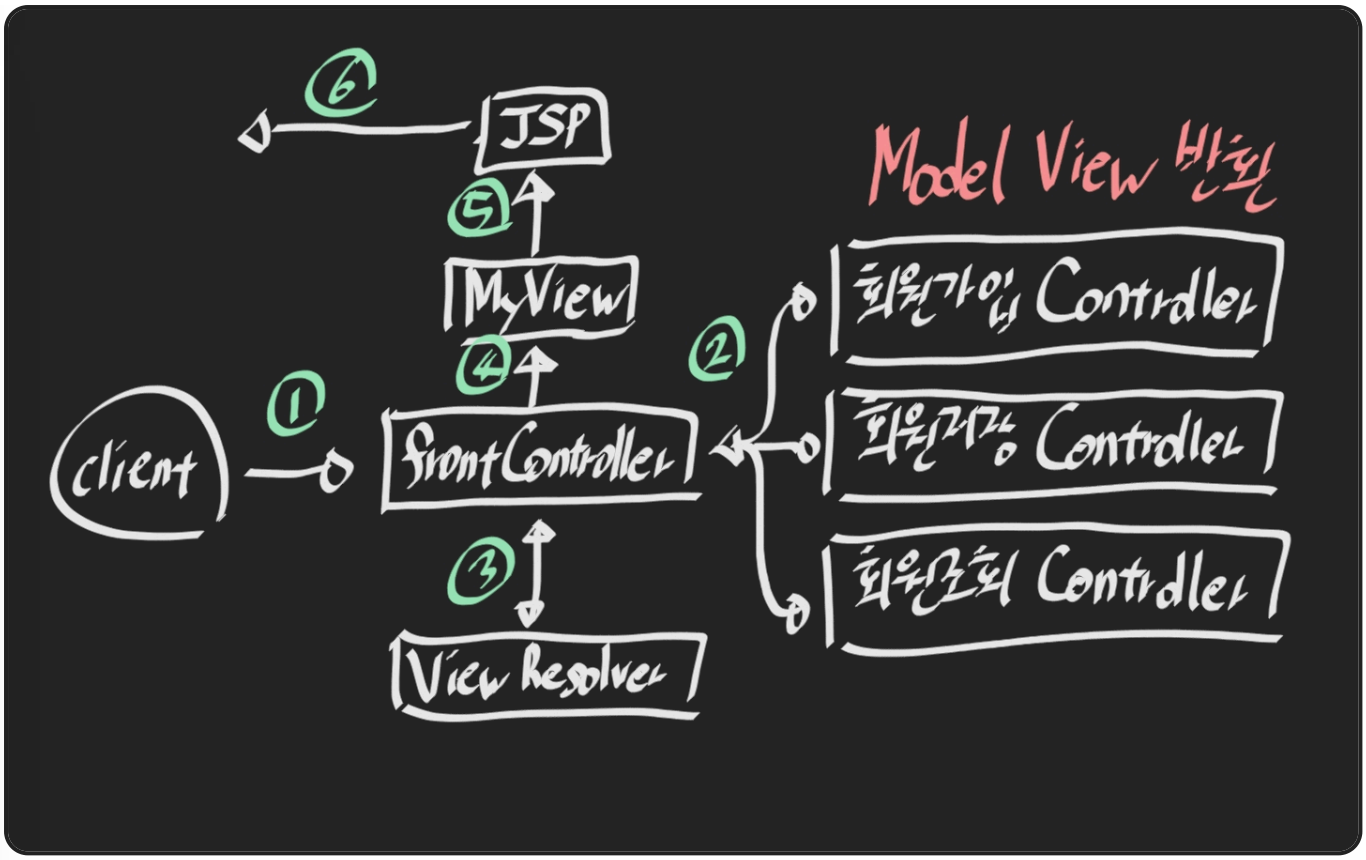
FrontController에서 paramMap은 request로 넘어온 각종 파라미터를 꺼내서 Controller에서 쉽게 사용할 수 있도록 해준다.

다시 Controller로 가서 View로 넘겨주고 싶은 데이터는 ModelView에 담아서 보낸다. ModelView내부에는 HashMap이 존재하는데 Value값이 Object로 어떤 데이터든 다 담을 수 있게 설계되어있다.

그리고 이전에 ViewPath에 중복되는 부분인 WEB-INF 와 .jsp를 FrontController의 ViewResolver라는 메서드를 통해 ViewPath를 완성하고 기존 View 로직을 수행한다.

이를 통해 Controller의 중복되는 코드를 많이 줄임과 동시에 의존성 또한 줄일 수 있었다.

다음에는 사소하지만 조금 더 Controller를 간편하게 해보자.



V4 - model 제공

FrontController

```
@WebServlet(name = "frontControllerServletV4", urlPatterns = "/front-
controller/v4/*")
public class FrontControllerServletV4 extends HttpServlet {
    private Map<String, ControllerV4> controllerMap = new HashMap<>();

    public FrontControllerServletV4() {
        controllerMap.put("/front-controller/v4/members/new-form",
                new MemberFormControllerV4());
        controllerMap.put("/front-controller/v4/members/save",
                new MemberSaveControllerV4());
        controllerMap.put("/front-controller/v4/members",
                new MemberListControllerV4());
    }

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
        String requestURI = request.getRequestURI();

        ControllerV4 controller = controllerMap.get(requestURI);
        if (controller == null){
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            return;
        }

        //paramMap
        Map<String, String> paramMap = createParamMap(request);
        Map<String,Object> model = new HashMap<>();

        String viewName = controller.process(paramMap, model);

        //new-form
        MyView view = viewResolver(viewName);
        view.render(model, request,response);
    }

    private static MyView viewResolver(String viewNam) {
        return new MyView("/WEB-INF/views/" + viewNam + ".jsp");
    }

    private static Map<String, String> createParamMap(HttpServletRequest
request) {
        Map<String,String> paramMap = new HashMap<>();
        request.getParameterNames().asIterator()
                .forEachRemaining(paramName -> paramMap.put(paramName,
request.getParameter(paramName)));
        return paramMap;
    }
}
```

SaveController

```
public class MemberSaveControllerV4 implements ControllerV4 {

    private MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    public String process(Map<String, String> paramMap, Map<String, Object>
model) {
        String username = paramMap.get("username");
        int age = Integer.parseInt(paramMap.get("age"));

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

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

Controller에 model이 추가되었다.

FrontController에서 빈 map을 넘겨주기 때문에 Model에 데이터를 담기위해서는 model에 넣으면 되고 반환 값으로 ViewName만 넘겨주면 된다.

문제점

만약에 V4 Controller에서 V3 Controller로 변경하려고 했을 때를 생각해보자.

V4 FrontController의 생성자 방식으로 Controller의 인스턴스를 생성하고 있는데 Map의 Value값으로 ControllerV4 인터페이스를 받고있다. 이렇게 되면 ControllerV3를 사용할 수가 없게 된다.

이를 해결하려면 Adapter Pattern이라는 것을 적용해야 한다.

V5 - Adapter Pattern (어댑터 패턴)

FrontController

```
@WebServlet(name = "frontControllerServletV5", urlPatterns = "/front-
controller/v5/*")
public class FrontControllerServletV5 extends HttpServlet {

    private final Map<String, Object> handlerMappingMap = new HashMap<>();
    private final List<MyHandlerAdapter> handlerAdapters = new ArrayList<>();

    public FrontControllerServletV5() {
        initHandlerMappingMap();
        initHandlerAdapters();
    }

    private void initHandlerAdapters() {
        handlerAdapters.add(new ControllerV3HandlerAdapter());
        handlerAdapters.add(new ControllerV4HandlerAdapter());
    }

    private void initHandlerMappingMap() {
        //V3 추가
        handlerMappingMap.put("/front-controller/v5/v3/members/new-form"
                , new MemberFormControllerV3());
        handlerMappingMap.put("/front-controller/v5/v3/members/save"
                , new MemberSaveControllerV3());
        handlerMappingMap.put("/front-controller/v5/v3/members"
                , new MemberListControllerV3());
        //V4 추가
        handlerMappingMap.put("/front-controller/v5/v4/members/new-form"
                , new MemberFormControllerV3());
        handlerMappingMap.put("/front-controller/v5/v4/members/save"
                , new MemberSaveControllerV3());
        handlerMappingMap.put("/front-controller/v5/v4/members"
                , new MemberListControllerV3());
    }

    @Override
```

```
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        Object handler = getHandler(request);

        if (handler == null) {
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            return;
        }

        MyHandlerAdapter adapter = getHandlerAdapter(handler);

        ModelView mv = adapter.handle(request, response, handler);

        String viewNam = mv.getViewName();
        MyView view = viewResolver(viewNam);

        view.render(mv.getModel(), request, response);
    }
    //만약 ControllerV3에 관한 인스턴스가 넘어왔을 경우 V3HandlerAdapter로 간다
    private MyHandlerAdapter getHandlerAdapter(Object handler) {
        for (MyHandlerAdapter adapter : handlerAdapters) {
            if (adapter.supports(handler)) {
                return adapter;
            }
        }
        throw new IllegalArgumentException("handler adapter를 찾을 수 없습니다.
handler = " + handler);
    }

    private Object getHandler(HttpServletRequest request) {
        String requestURI = request.getRequestURI();
        return handlerMappingMap.get(requestURI);
    }

    private static MyView viewResolver(String viewNam) {
        return new MyView("/WEB-INF/views/" + viewNam + ".jsp");
```

```
        }
    }
```

ControllerV3HandlerAdapter

```
public class ControllerV3HandlerAdapter implements MyHandlerAdapter {
    @Override
    public boolean supports(Object handler) {
        return (handler instanceof ControllerV3);
    }

    @Override
    public ModelView handle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws HttpServerErrorException, IOException {
        ControllerV3 controller = (ControllerV3) handler;

        Map<String, String> paramMap = createParamMap(request);
        ModelView mv = controller.process(paramMap);

        return mv;
    }

    private static Map<String, String> createParamMap(HttpServletRequest
request) {
        Map<String,String> paramMap = new HashMap<>();
        request.getParameterNames().asIterator()
                .forEachRemaining(paramName -> paramMap.put(paramName,
request.getParameter(paramName)));
        return paramMap;
    }
}
```

뭔가 복잡해 보이지만 단순하다.

먼저 FrontController가 요청을 받는다. 예시로 ControllerV3 요청이 온다고 하자

그러면 `getHandler()` 를 통해 생성할 객체 인스턴스를 가져온다.

`getHandlerAdapter()` 를 통해 사전에 정의된 HandlerAdapter가 있는지 확인한다. 그러면 ControllerV3HandlerAdapter가 adapter에 할당이 된다.

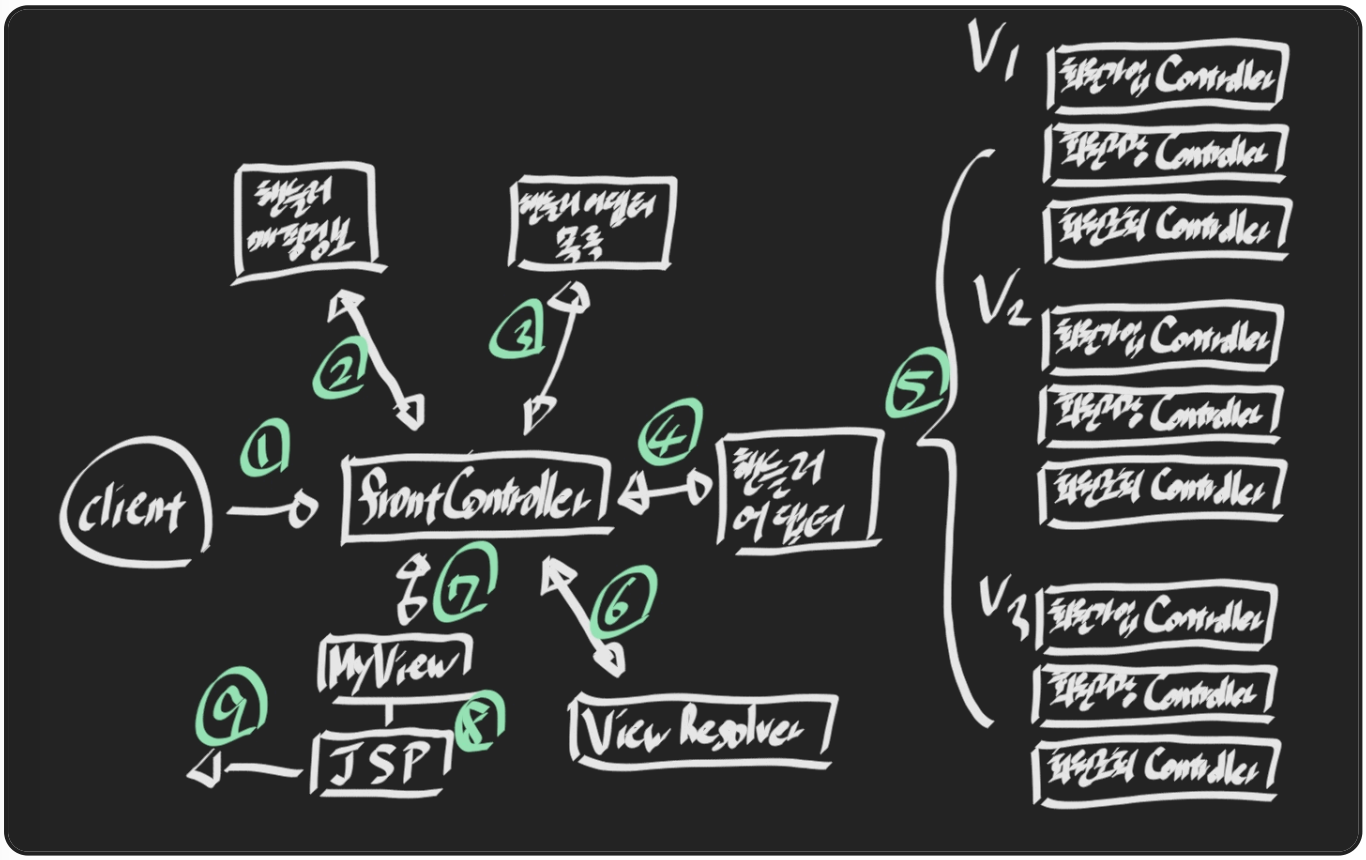
이제 adapter.handle()을 통해 실제 인스턴스와 request와 response가 넘거가고 이 이후부터는 기존 로직과 동일하게 동작한다.

ControllerV4HandlerAdapter는 ControllerV4의 리턴값에 맞춰 파라미터를 생성하여 넘겨준다.

ControllerV4HandlerAdapter

```
public class ControllerV4HandlerAdapter implements MyHandlerAdapter {
    @Override
    public boolean supports(Object handler) {
        return (handler instanceof ControllerV4);
    }

    @Override
    public ModelView handle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws HttpServerErrorException, IOException {

        ControllerV4 controller = (ControllerV4) handler;

        Map<String, String> paramMap = createParamMap(request);
        HashMap<String, Object> model = new HashMap<>();

        String viewName = controller.process(paramMap, model);

        ModelView mv = new ModelView(viewName);
        mv.setModel(model);

        return mv;
    }
```

```
    private static Map<String, String> createParamMap(HttpServletRequest
request) {
        Map<String,String> paramMap = new HashMap<>();
        request.getParameterNames().asIterator()
                .forEachRemaining(paramName -> paramMap.put(paramName,
request.getParameter(paramName)));
        return paramMap;
    }
}
```



Spring MVC

```
@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 members(Model model) {

            List<Member> members = memberRepository.findAll();

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

우리가 가장 잘 아는 형태이며 FrontController는 사실 Spring Framework에 내장되어 있다.

지금 까지 본 코드가 모두 Spring Framework와 대응되는데

FrontController -> DispatcherServlet

HandlerMappingMap -> HandlerMapping

MyHandlerAdapter -> HandlerAdapter

ModelView -> ModelAndView

ViewResolver -> ViewResolver

MyView -> View

DispatcherServlet은 FrontControllerPattern에 따라 구현되었으며 SpringMVC의 핵심이다.

동작순서를 기억하며 이렇게 SpringMVC 패턴의 구조에 대해 알아보았다.

1. 핸들러 조회 : 핸들러 매핑을 통해 요청 URL에 매핑된 핸들러 조회
2. 핸들러 어댑터 조회 : 핸들러를 실행할 수 있는 어댑터 조회
3. 핸들러 어앱터 실행 : 핸들러 어댑터 실행
4. 핸들러 실행 : 실제 컨트롤러 실행
5. ModelAndView 반환 : 컨트롤러가 반환하는 정보를 ModelAndView로 변환해서 반환
6. ViewResolver 호출 : 사용하고있는 View에 맞는 리졸버를 실행
7. View 반환 : 뷰 리졸버는 뷰의 논리 이름("new-form")을 물리 이름("/WEB-INF/views/new-form.jsp")으로 변환한다.
8. View 렌더링 : 뷰를 렌더링 한다.

hyeonZIP
0 팔로워

댓글 0개

댓글을 작성해보세요

댓글 등록