Spring MVC 04

[프론트 컨트롤러 패턴 - 1]

스프링 MVC 1편 - 백엔드 웹 개발 핵심 기술

김영한

2022.08.21

MVC 프레임워크 만들기

프론트 컨트롤러 패턴

공통의 관심사를 하나로 모아주는 서블릿을 도입해보자.

특징

- 프론트 컨트롤러 서블릿 하나로 클라이언트 요청을 받음
- 프론트 컨트롤러가 요청에 맞는 컨트롤러를 찾아서 호출
- 입구를 하나로 공통 처리 가능
- 나머지 컨트롤러는 서블릿을 사용하지 않아도 된다.

스프링 웹 **MVC**의 **DispatcherServlet**이 **FrontController** 패턴으로 구현되어 있다**.**

프론트 컨트롤러 도입

version -1

기존 MVC 패턴을 최대한 유지하며 프론트 컨트롤러를 도입한다.

```
public interface ControllerV1 {
   void process(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException;
}
```

컨트롤러를 구현할 인터페이스를 설계한다.

```
public class MemberFormControllerV1 implements ControllerV1 {
   @Override
   public void process(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
       String viewPath = "/WEB-INF/views/new-form.jsp";
       RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
       dispatcher.forward(request, response);
   }
}
```

기존 servlet 코드의 service와 같은 형태의 process를 각각 구현체에 구현해준다.

FrontController 서블릿

```
@WebServlet(name="frontControllerServletV1", urlPatterns =
"/front-controller/v1/*") urlPattern 이 front-controller/v1/*로 시작할경우 매핑
public class FrontControllerServletV1 extends HttpServlet {

   private Map<String, ControllerV1> controllerMap = new HashMap<>();
   각각의 url 을 매핑해줄 Map
   public FrontControllerServletV1() { 생성자가 호출될 때 각 컨트롤러를 올려준다.
       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 서블릿이 실행될때 매핑만 찾아서 컨트롤러와 연결해주는것.
   protected void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
       String requestURI = request.getRequestURI();

       ControllerV1 controllerV1 = controllerMap.get(requestURI);

       if (controllerV1 == null){
           response.setStatus(HttpServletResponse.SC_NOT_FOUND);
           return;
       }

       controllerV1.process(request, response);
```

```
    }
}
```

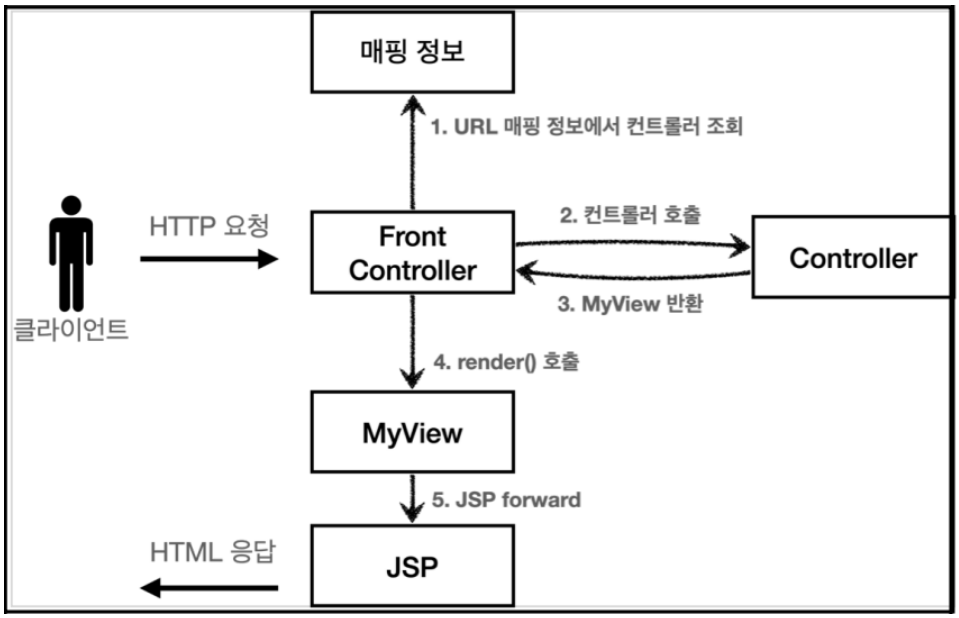
클라이언트가 요청한 정보를 프론트 컨트롤러가 받고 → 프론트 컨트롤러가 각 컨트롤러 인터페이스 구현체를 연결해 준다.

아키텍처의 구조를 개선할 때는 구조적인것을 최대한 개선한 후에 세밀한 부분을 조정해야 한다.

그렇기에 기존 소스를 그대로 동작시키면서 구조를 변경시킨것.

View 이동 분리

구조 . MyView 도입




```
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);
    }
}
```

중복되는 이동을 제거해준다.

Model 추가

서블릿 종속성 제거

컨트롤러입장에서 사용하지 않는 request, response 를 request객체를 Model 객체를 만들어 반환 한다. 그래서 컨트롤러에서 서블릿 기술을 전혀 사용하지 않게 한다.

중복되는 뷰 경로를 제거

논리 이름만으로 경로(물리 이름)를 생성하게 변경 - 만약 view 폴더 변경을 주더라도 프론트 컨트롤러만 고치면 되기 때문에 좋은 설계 모습이 된다.

- ViewResolever 도입

ModelView 간단히 설계

```
@Getter @Setter
public class ModelView {

    private String viewName;
    private Map<String, Object> model = new HashMap<>();

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

model : request 객체속에 있는 파라미터를 전부 넣어줄것.

컨트롤러 인터페이스 와 구현체 **- ModelView** 반환

```
public interface ControllerV3 {
   ModelView process(Map<String, String> paramMap);
}
```

```
public class MemberListControllerV3 implements ControllerV3 {

   private MemberRepository memberRepository = MemberRepository.getInstance();

   @Override
   public ModelView process(Map<String, String> paramMap) {
       List<Member> members = memberRepository.findAll();
       ModelView mv = new ModelView("members");
       mv.getModel().put("members", members);
       return mv;
   }
}
```

FrontController V3

- ModelView 객체를 파라미터로하는 **view.render** 메소드추가

```
@WebServlet(name="frontControllerServletV3", urlPatterns =
"/front-controller/v3/*")
public class FrontControllerServletV3 extends HttpServlet {

    . . . servletV2 와 동일 코드

       Map<String, String> paramMap = createParamMap(request);
       ModelView mv = controllerV3.process(paramMap);
       String viewName = mv.getViewName(); // 논리 이름 밖에 못 얻는다.
       MyView view = viewResolver(viewName);
       view.render(mv.getModel(), request, response);
   }

   //논리 이름 --> 물리 이름
   private MyView viewResolver(String viewName) {
       return new MyView("/WEB-INF/views/" + viewName + ".jsp");
   }
```

```
    private Map<String, String> createParamMap(HttpServletRequest request) {
        //paramMap에 request에 있는 파라미터를 전부 PUT
        Map<String, String> paramMap = new HashMap<>();
        request.getParameterNames().asIterator()
                .forEachRemaining(paramName -> paramMap.put(paramName,
request.getParameter(paramName)));
        return paramMap;
    }
}
```

MyView 에 메소드 추가

```
public void render(Map<String, Object> model, HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    modelToRequestAttribute(model, request);
    RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
    dispatcher.forward(request, response);
}

private void modelToRequestAttribute(Map<String, Object> model, HttpServletRequest
request) {
    model.forEach((key, value)-> request.setAttribute(key,value));
}
```

정리

- ModelView 객체를 추가 하여 Servlet과 Controller의 종속관계를 없애주고 항상 전달하던 request, response를 사용하지 않게됨.
- ModelView 객체속 맴버로 논리 이름을 넣어주고 그것으로 물리 경로를 생성하는 viewResolver를 도입한다.