Spring MVC 03

[서블릿, JSP, MVC]

스프링 MVC 1편 - 백엔드 웹 개발 핵심 기술

김영한

2022.08.20

서블릿으로 회원 관리 구현 - 비효율

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

response.setContentType("text/html");
response.setCharacterEncoding("utf-8");
PrintWriter w = response.getWriter();
w.write("<html>");
w.write("<head>");
w.write(" <meta charset=\"UTF-8\">");
w.write(" <title>Title</title>");
w.write("</head>");
w.write("<body>");
w.write("<a href=\"/index.html\">메인</a>");
w.write("<table>");
w.write(" <thead>");
w.write(" <th>id</th>");
w.write(" <th>username</th>");
w.write(" <th>age</th>");
w.write(" </thead>");
w.write(" <tbody>");
for (Member member : members) {
   w.write(" <tr>");
   w.write(" <td>" + member.getId() + "</td>");
   w.write(" <td>" + member.getUsername() + "</td>");
   w.write(" <td>" + member.getAge() + "</td>");
   w.write(" </tr>");
}
w.write(" </tbody>");
w.write("</table>");
w.write("</body>");
w.write("</html>");
```

자바 코드안에서 **HTML**을 구현해야 되기 때문에 엄청나게 불편하다 그래서 탬플릿 엔진이 나왔다.

JSP로 동일작업

우선 build.gradle 에 JSP 라이브러리 추가해준다.

```
implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'
implementation 'javax.servlet:jstl'
```

회원 관리 구현 **- JSP** 목록보기

```
<%@ page import="java.util.List" %>
<%@ page import="hello.servlet.domain.member.MemberRepository" %>
<%@ page import="hello.servlet.domain.member.Member" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%  ----------------------------------비즈니스 로직
    MemberRepository memberRepository = MemberRepository.getInstance();
    List<Member> members = memberRepository.findAll();
%>  ----------------------------------비즈니스 로직
    ----------------------------------뷰 화면 구성
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<a href="/index.html">메인</a>
<table>
   <thead>
        <th>id</th>
        <th>username</th>
        <th>age</th>
   </thead>
   <tbody>
   <%
        for (Member member : members) {
            out.write(" <tr>");
            out.write(" <td>" + member.getId() + "</td>");
            out.write(" <td>" + member.getUsername() + "</td>");
            out.write(" <td>" + member.getAge() + "</td>");
            out.write(" </tr>");
        }
   %>
```

```
   </tbody>
</table>
</body>
</html>
    ----------------------------------뷰 화면 구성
```

위의 서블릿 작성과 반대로 HTML 형식안에 JAVA 코드가 들어있다. → 템플릿 엔진

JSP는 자바 코드를 그대로 다 사용할 수 있다. 액션 태그를 사용해 여러 기능들을 사용할 수 있다.

반복문 등을 사용할때에도 편리하다 . (JSP 파일도 서블릿 파일로 변환되어 실행 된다.)

문제는 비즈니스 로직이 화면과 함께 **JSP**파일안에 같이 들어있다**.**

→ 이러한 문제를 해결하고자 **MVC** 패턴이 등장

그러면 서블릿 (비즈니스 로직) + JSP (뷰) 로 구현 해보자.

MVC 패턴

- 역할의 가중, 변화 사이클이 다름, 기능 특화의 필요

MVC 패턴 **(Model View Controller)**

모델 : 뷰에 출력할 데이터를 담아둔다.

뷰 : 모델에 담겨있는 데이터를 화면에 구현해준다.

컨트롤러 : 비즈니스 로직을 수행한다. (호출한다)

```
@WebServlet(name = "mvcMemberFormServlet", urlPatterns =
"/servlet-mvc/members/new-form")
public class MvcMemberFormServlet extends HttpServlet {
   @Override
   protected void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
       String viewPath = "/WEB-INF/views/new-form.jsp";
       →WEB-INF폴더 파일은 WAS에서 클라이언트가 경로로 바로 접근하는 것을 막아준다.
       RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
```

```
        dispatcher.forward(request, response);
        → 서버 내부에서 다시 호출이 발생한다. 서버안에서 메소드 호출하듯이.
    }
}
```

redirect vs forward

리다이렉트는 실제 클라이언트(웹 브라우저)에 응답이 나갔다가, 클라이언트가 redirect 경로로 다시 요청한다. 따라서 클라이언트가 인지할 수 있고, URL 경로도 실제로 변경된다. 반면에 포워드는 서버 내부에서 일어나는 호출이기 때문에 클라이언트가 전혀 인지하지 못한다.

즉 리다이렉트는 두번 요청하는 것 . (재 요청)

MVC 패턴으로 회원목록보기

서블릿 - Controller

```
@WebServlet(name = "mvcMemberListServlet", urlPatterns = "/servlet-mvc/members")
public class MvcMemberListServlet extends HttpServlet {

    MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        List<Member> members = memberRepository.findAll();
        request.setAttribute("members", members);
        String viewPath = "/WEB-INF/views/members.jsp";
        RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
        dispatcher.forward(request, response);
    }
}
```

members.jsp - View

```
<%@ page import="java.util.List" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>
<head>
```

```
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<a href="/index.html">메인</a>
<table>
   <thead>
        <th>id</th>
        <th>username</th>
        <th>age</th>
   </thead>
   <tbody> JSTL태그 라이브러리 같은 라이브러리는 각각 템플릿이 가지고 있다.
   <c:forEach var="item" items="${members}">
       <tr>
           <td>${item.id}</td>
           <td>${item.username}</td>
           <td>${item.age}</td>
       </tr>
   </c:forEach>
   </tbody>
</table>
</body>
</html>
```

MVC 패턴의 한계점

- 컨트롤러가 중복이 많고 필요하지 않은 코드들도 많다.

포워드 중복

- View로 이동하는 코드가 항상 중복 호출되어야 한다.

viewPath 중복

- prefix : /WEB-INF/views/ , suffix : .jsp 가 중복이 된다.

사용하지 않는 코드

- request, response를 사용되지 않아도 호출해야 한다.

공통 처리가 어렵다.

- 기능이 복잡해 질 수록 공통처리 부분이 필요할 텐데 반복적인 호출도 문제이고 호출하지 않아도 문제가 된다고 생긴다.

정리

공통된 처리가 필요성을 느껴서 수문장 역할이 나오게 된다. 결과적으로 수문장 역할을 하는 컨트롤러를 두고 수문장역할을 하며 공통된 부분을 처리하게 되는 패턴이 나오게 된다.

프론트 컨트롤러**(Front Controller)** 패턴 이 나오게된 배경이된다.