Day 02(2019-01-09)

1. Servlet과 JSP

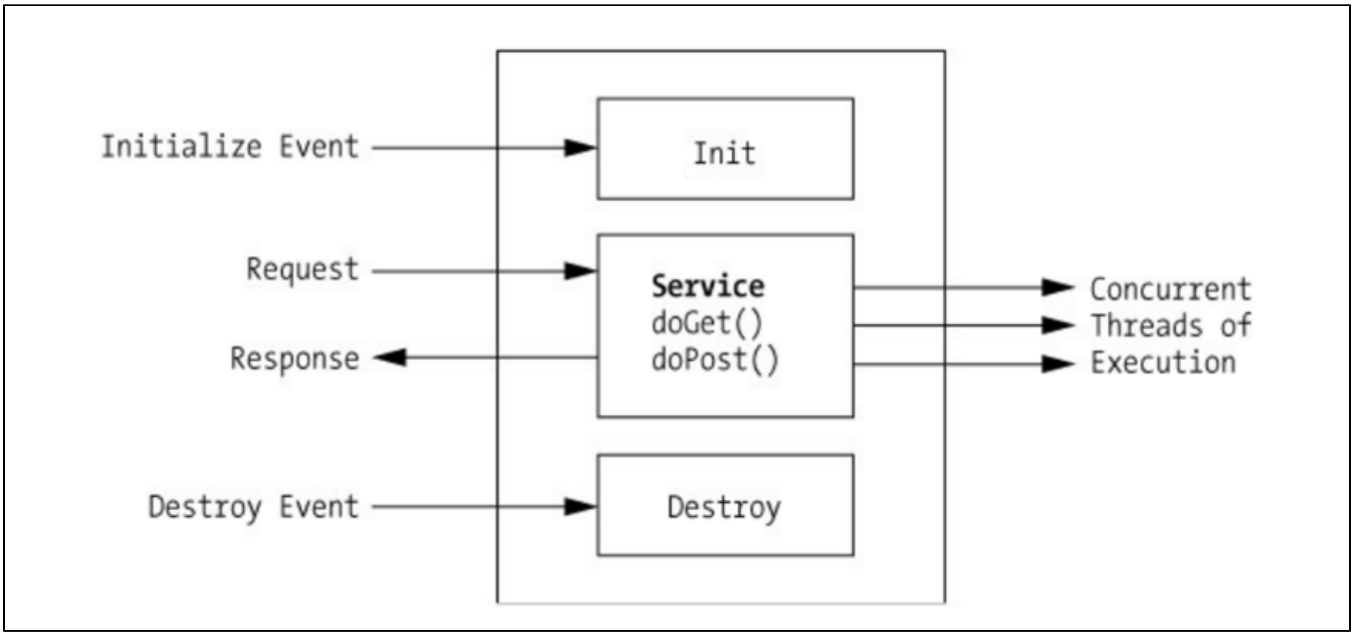
(1) Servlet(Server + Applet)이란?

WAS위에서 동작하는 Java 프로그램으로 Web application을 구성한다
이름 그대로 Servet에서 구동되는 Applet임

1) Servlet의 동작

- Client가 URL을 통해 HTTP Request를 해당 서버의 전송
- Servlet Container는 HttpServletRequest, HttpServletResponse 객체를 생성
- 요청한 URL 정보를 참고하여 매핑된 서블릿을 탐색
- HTTP Method에 따라 Servlet의 doGet(), doPost() 메소드 등을 호출
- 동적인 페이지가 생성되어 HttpServletResponse에 담긴 후 응답
- 응답 완료 후 HttpServletRequest, HttpServletResponse 객체 소멸

2) Servlet 생명주기



- Init() : Servlet이 메모리에 Load 될 때(Servlet이 처음으로 요청 될 때) 실행 됨
- service(): HTTP Method 타입에 따라 맞는 doGet(), doPost()를 호출함
- destroy() : 메모리에 Unload 되기 전 실행 되는 Callback

※ 꼭 알아두어야 할 것

초기화 된 Servlet은 Client의 요청이 있을 때 마다 Thread를 생성하여 병렬적으로 Service를 수행함

즉, Servlet 객체는 여러개 생성되지 않는다

```
public class HelloServlet extends HttpServlet {
        public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException {
                request.setCharacterEncoding("UTF-8");
                response.setCharacterEncoding("UTF-8");


                String userId = req.getParameter("userId") != null ? (String)req.getParameter("userId") : null;
                //
                //->    validation
                //...
                //
                //-> DB
                Connection dbConnection = null;
                PreparedStatement preparedStatement = null;

                String selectSQL = "SELECT USER_ID, USERNAME FROM DBUSER WHERE USER_ID = ?";

                PrintWriter out = res.getWriter();
                try {
                        dbConnection = getDBConnection();
                        preparedStatement = dbConnection.prepareStatement(selectSQL);
                        preparedStatement.setString(1, userId);

                        // execute select SQL stetement
                        ResultSet rs = preparedStatement.executeQuery();

                        res.setContentType("text/html;charset=UTF-8");


                        out.println("<HTML>");
                        out.println("<BODY>");
                        while (rs.next()) {
                                String username = rs.getString("USERNAME");


                                //
                                out.println("<h1>Hello World!!"+userId+"["+username+"]</h1>");
                        }


                        out.println("</BODY>");
                        out.println("</HTML>");
                        out.close();

                } catch (SQLException e) {

                        out.println("<HTML>");
                        out.println("<BODY>");

                        while (rs.next()) {
                                out.println("<h1>ERROR!</h1>");
                        }

                        out.println("</BODY>");
                        out.println("</HTML>");

                } finally {
                        if (preparedStatement != null) {
                                try(
                                        preparedStatement.close();
                                } catch(Exception e){}
                        }

                        if (dbConnection != null) {
```

```
                                try{
                                        dbConnection.close();
                                } catch(Exception e){}
                        }
                }
        }
}
```

- Java코드안에 HTML코드가 내장되어있음
  → 화면 로직을 작성하기 매우 힘들며 약간의 변경에도 화면이 쉽게 깨지는 문제점 존재

- Boilerplate code가 산재
  - 요청으로부터 파라미터 가져오는 부분 (NPE가 가장 많이 남)
  - 인코딩 처리
  - DB 접근시 try - catch 구조
    → Exception이 발생해도 추가 처리가 불가능함에도 의무적으로 try-catch로감싸야 함

- 비즈니스 로직이 화면로직과 매우가까이 위치함
  → 디자인 개선????, 퍼블리셔가 클래스 이름을 바꿔????, html 구조좀 바꿔 주세요????

- 잊지말자 XML
  → web.xml에 servlet mapping을 잊지 말 것

(2) JSP (Java Server Page)란?

Java 소스 내에 HTML이 포함된 Servlet이 가지는 단점을 보완하고자 나타남

→ 내부적으로 JSP가 컴파일 되면 Servlet이 생성 됨

※ 누가 컴파일하나? 언제 컴파일 되나?

누가? **WAS가**, 언제? **최초 JSP가 쓰일 때** (Servlet은 Class가 Load 될 때, JSP는 Runtime에)

```
<%@page import="java.util.Calendar" %>
<%@ page contentType="text/html; charset=UTF-8"%>
<%
        String str=String.format("%tF",Calendar.getInstance());
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
</head>
<body>
         <%=str%><br/>
</body>
</html>
```

```
<%-- JSP 1.2(old) --%>
<jsp:useBean id="customer" type="beans.Customer" scope="request">
</jsp:useBean>

<ul>
  <li>Name: <jsp:getProperty name="customer" property="name" /></li>
  <li>Email: <jsp:getProperty name="customer" property="email" /></li>
  <li>ID: <jsp:getProperty name="customer" property="id" /></li>
</ul>




<%-- JSP 2.0(Preferred)  --%>
<ul>
  <li>Name: ${customer.name}</li>
  <li>Email: ${customer.email}</li>
  <li>ID: ${customer.id}</li>
</ul>




<%-- JSTL--%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<title>Count to 10 Example (using JSTL)</title>
</head>
<body>
   <c:forEach var="i" begin="1" end="10" step="1">
      <c:out value="${i}"/>
      <br/>
   </c:forEach>
</body>
</html>
```

- HTML코드 안에 Java 코드가 내장되어있음
  → 프리젠테이션 로직을 처리하기 수월, 유지보수성 증가

- JSTL을 통해 개발 생산성 향상

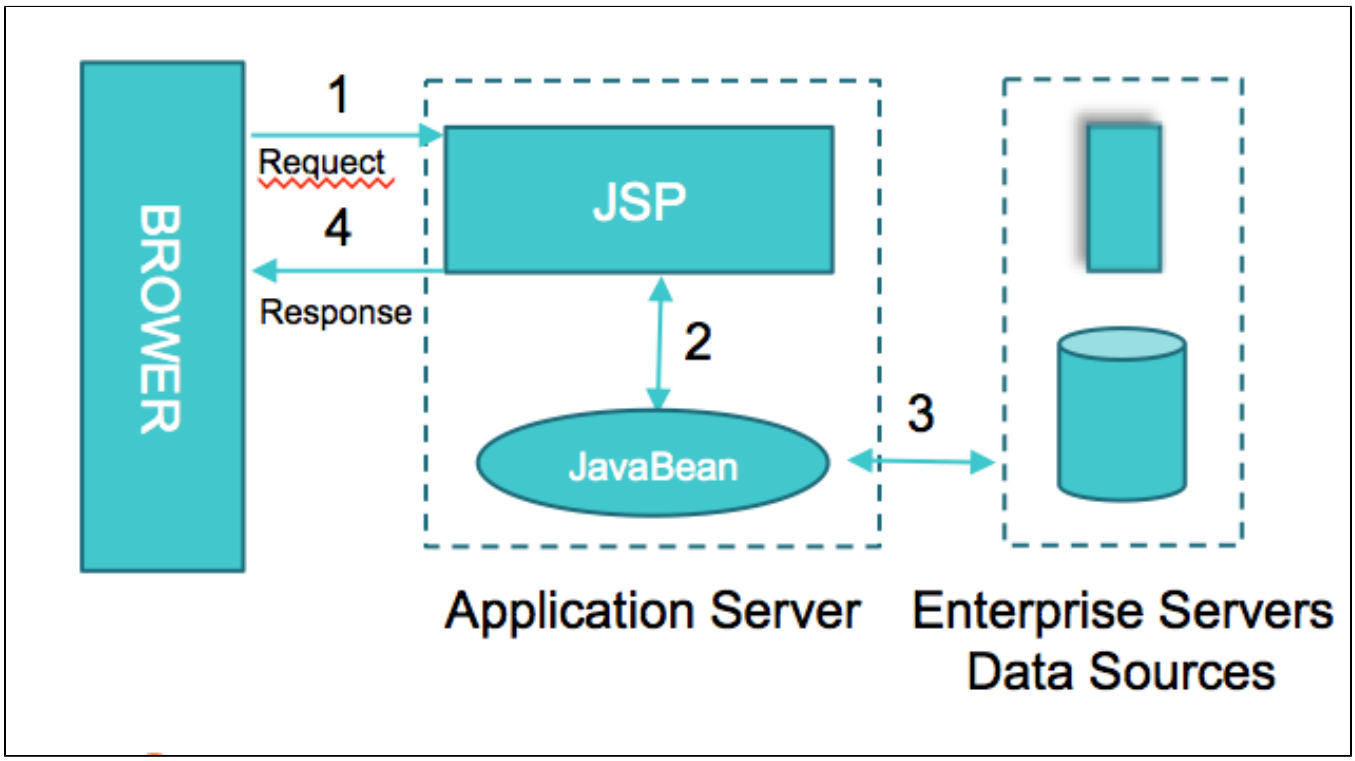
- 여전히 비즈니스 로직과 프리젠테이션 로직이 가까울 수 있는 가능성이 높음
  → 스크립트릿의 사용을 자제하고 EL, JSTL의 사용을 장려하여 비즈니스 로직이 섞이지 않게 하자

2. MVC 패턴

(1) MVC(Model View Controller) 개념

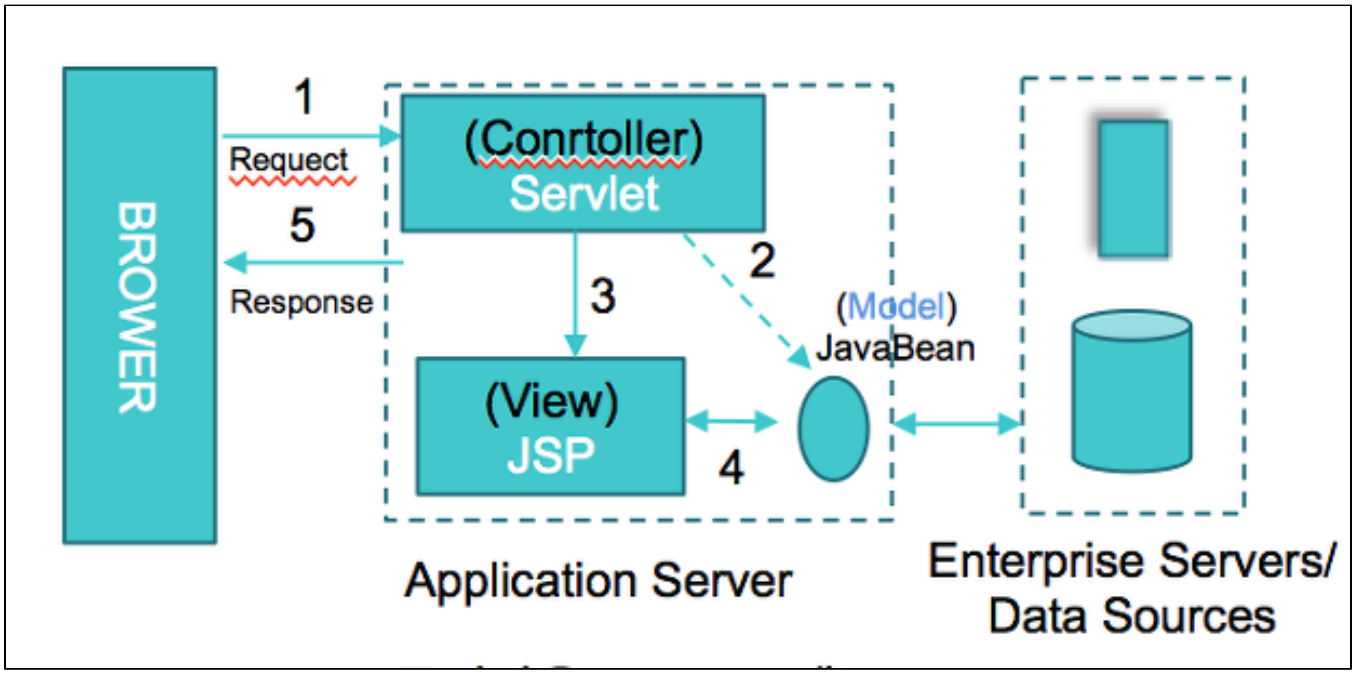
- Model: 어떠한 동작을 수행하는 코드로 객체의 상태와 연관있음
- View: 사용자 인터페이스 요소, Client에게 보여지는 영역, Model로부터 값을 가져와 사용자에게 보여줌
- Controller : Model과 View의 사이에서 상호작용을 도우며 흐름을 제어

(2) JSP & Servlet model 1



- JSP만 사용함
- JSP를 통해 비즈니스로직과 프리젠테이션 로직을 모두 처리 함
- 개발속도가 빠르고 쉽기 때문에 진입장벽이 낮다
- Application의 크기가 커질 수록 유지보수가 어려워짐

(3) JSP & Servlet model 2 (MVC 패턴이 적용 됨)



- JSP와 Servlet을 같이 사용

- Model은 DAO, DTO가 Controller는 Servlet이, View는 JSP가
- 구조가 복잡하여 진입장벽이 높음
- Applicatiion의 크기가 커져도 역할이 분담되어 있어 유지보수가 용이함

1) → 4)로 갈수록 간단한 app 또는 작은 규모의 개발팀 -> 복잡한 app 또는 큰 규모의 개발팀(Enterprise)

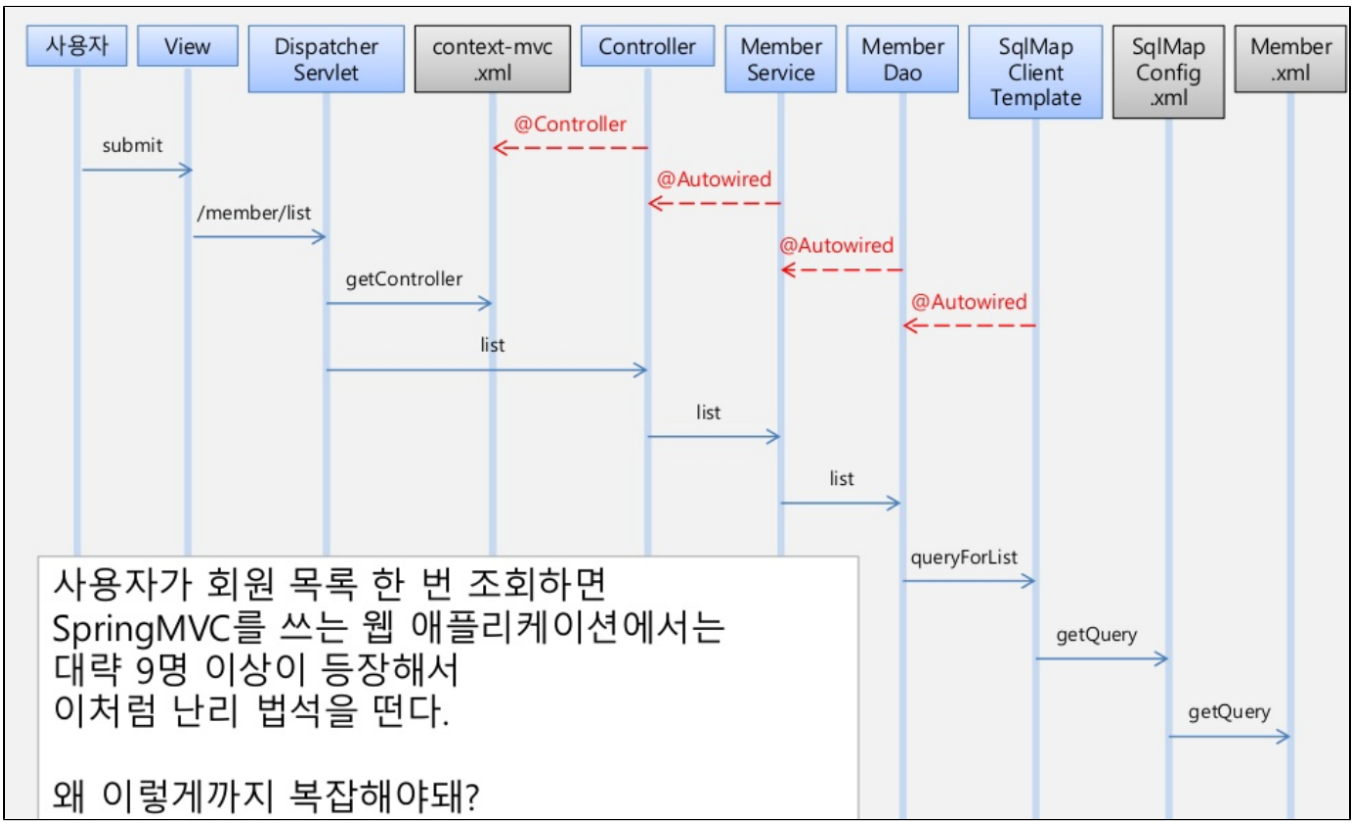
1) Call Java code directly
   Java 코드를 직접 호출
   모든 Java 코드를 JSP 페이지에 넣는다.
   아주 적은 양의 코드에만 적합한 전략

2) Call Java code indirectly
   Java 코드를 간접적으로 호출
   별도의 utility class(Java Class)를 작성한다.
   utility class를 호출하는 데 필요한 Java 코드만 JSP 페이지에 넣는다.

3) Use beans
   beans로 구조화된 별도의 utility class(Java Class)를 작성한다.
   jsp:useBean, jsp:getProperty, jsp:setProperty를 사용하여 utility class를 호출한다.

4) Use the MVC architecture
   MVC 아키텍처를 사용
   Servlet(Controller)이 요청에 응답하고 적절한 데이터를 검색하여 결과를 beans(Model)에 저장한다.
   이 결과를 JSP 페이지(View)로 전달하여 결과를 표시한다.
   즉, JSP 페이지는 bean을 사용한다

3. Spring MVC



(1) DispatcherServlet

- Spring MVC의 핵심
- 사용자 요청 검증 및 요청을 우리가 원하는 형태의 DTO로 가공, 매핑된 URI에 맞는 handler 호출, 결과 화면을 그려낼 View resolver 탐색 및 처리, Locale 처리 등... 모든 것을 다함

(2) MemberController

- DispatcherServlet에 의해 호출되며 가공된 사용자의 요청을 받고 해당 요청에 맞는 비즈니스 로직을 담은 Service 객체를 주입 받아 처리를 위임 및 결과 반환

(3) MemberService

- Controller에 의해 호출되며 비즈니스로직이 존재하는 곳
- DB 접근을 담당하는 DAO 객체를 주입받아 DAO CRUD 작업을 위임하고 결과를 반환

(4) MemberDao

- Service에 의해 호출되어 쿼리를 담당하는 SqlMapClientTemplate 객체를 주입받아 SqlMapClientTemplate 객체에 쿼리 수행을 위임하고 결과를 반환

(5) Member.xml

- 실제 쿼리가 저장된 곳