7주차 Spring MVC(1)

1. MVC 패턴이란?

a. 하나의 애플리케이션, 프로젝트를 구성할 때 그 구성요소를 Model, View, Controller의 역할로 구분한 디자인 패턴

b. 위와 같이 역할 구분없이 기능을 구현하는 코드를 짜는 경우, html코드와 java코드가 서로 섞여 있게 되서 코드를 파악하기 어려울 뿐만 아니라 나중에 코드 수정이 필요할 때 Business Logic과 Control 코드가 서로에 영향을 끼치는 코드가 있어서 코드 수정을 위해서 전체 코드를 다 수정해야 할 수도 있는 문제를 위의 패턴을 적용해서 해결이 가능하다.

c. **Model**

i. 애플리케이션의 정보, 데이터 또는 데이터와 정보들의 가공을 책임지는 컴포넌트
ii. Controller가 호출할 때, 요청에 맞는 역할을 수행
iii. 데이터를 추출하거나 저장, 삭제, 업데이트, 변환 등의 작업 수행
iv. 상태의 변화가 있을 때 Controller와 View에 통보해 후속 조치 명령을 받을 수 있게 함
v. DTO, DAO

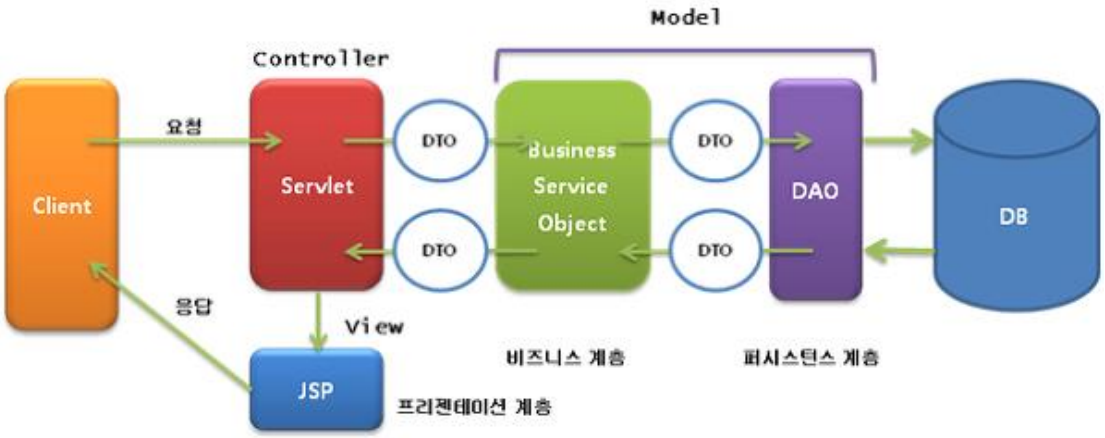
d. **View**

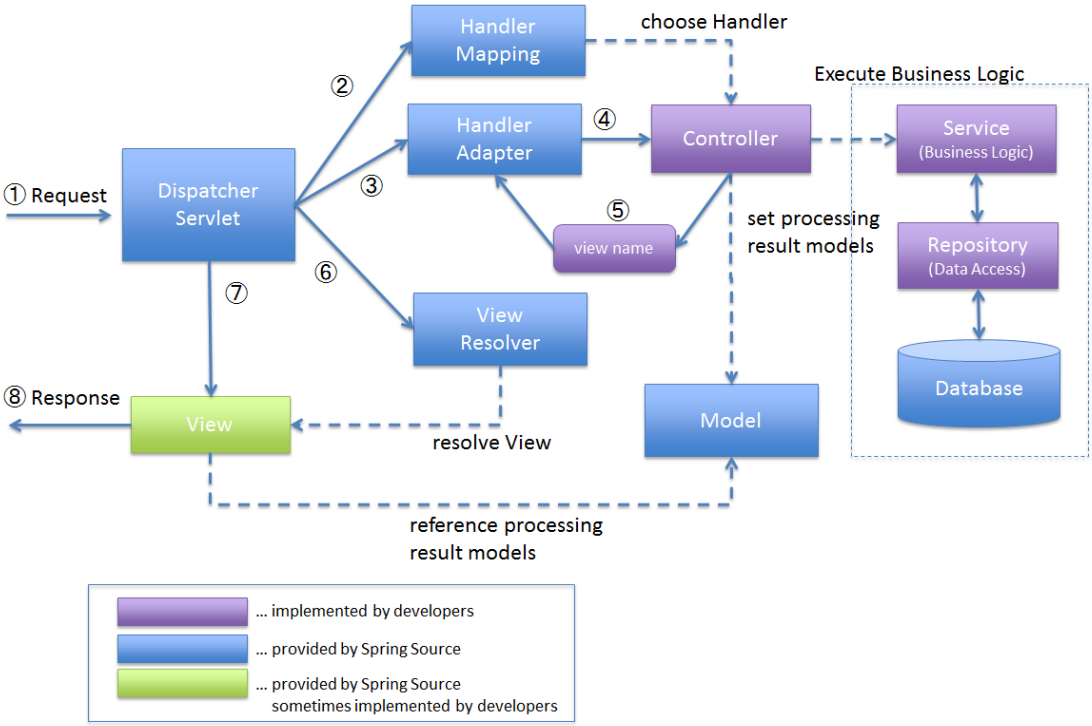
i. 사용자 인터페이스 요소, 데이터 기반으로 사용자들이 볼 수 있는 화면
ii. Controller로부터 받은 Model의 결과값을 가지고 사용자에게 출력할 화면을 만듬
iii. jsp 등

e. **Controller**

i. 데이터와 사용자인터페이스 요소들을 잇는 다리역할
ii. Client의 요청을 받았을 때, 그 요청에 대해 실제 업무를 수행하는 Model 컴포넌트를 호출
iii. 클라이언트가 보낸 데이터가 있다면, Model에 전달하기 쉽게 데이터를 가공 후 전달
iv. Model이 업무를 마치면 그 결과를 View에게 전달
v. DispatcherServlet에 의해 호출되어 사용자의 Request를 전달받고, 해당 요청의 비즈니스 처리를 담당하는 서비스 객체를 Spring으로부터 DI받아서, 그 서비스 객체에 처리를 위임하고, 처리 결과와 결과 화면에 대한 정보를 DispatcherServlet에게 반환
vi. Servlet : 웹 기반의 요청에 대한 동적인 처리가 가능한 하나의 클래스
vii. url mapping(requestMapping)

f. **Spring MVC**





1) DispatcherServlet이 request를 받는다.
2) DispatcherServlet은 적절한 Controller를 선택하는 작업을 HandlerMapping에 전달한다. HandlerMapping은 수신된 request URL에 매핑된 Controller를 선택하고 Controller를 DispatcherServlet에 return한다.
3) DispatcherServlet은 Controller의 비즈니스 로직을 HandlerAdapter로 전달한다.
4) HandlerAdapter는 Controller의 비즈니스 로직 프로세스를 호출한다.
5) Controller는 비즈니스 로직을 실행하고 처리 결과를 Model에 설정하고 View 이름을 HandlerAdapter에 return한다.
6) DispatcherServlet은 View이름에 해당하는 View를 해결하는 작업을 ViewResolver로 전달한다. ViewResolver는 View이름에 매핑된 View를 반환한다.

7) DispatcherServlet은 렌더링 프로세스를 return 된 View로 전달한다.
8) View는 모델 데이터를 렌더링하고 응답을 반환한다.

2. 컨트롤러와 서비스의 차이(url mapping, 비즈니스 로직)

a. Controller

i. DispatcherServlet에 의해 호출되어 사용자의 Request를 전달받고, 해당 요청의 비즈니스 처리를 담당하는 서비스 객체를 Spring으로부터 DI받아서, 그 서비스 객체에 처리를 위임하고, 처리 결과와 결과 화면에 대한 정보(ModelAndView)를 DispatcherServlet에게 반환

ii. url mapping

1. DispatcherServlet에게 전달받은 Request의 URI 정보와 mapping된 메서드를 호출해서 서비스 객체의 메서드를 통해 비즈니스 로직이 처리되게 한다.

2. mapping 방법
   a. dispatcherServlet-context.xml(어노테이션 X)

SampleController.java

```
public class SampleController implements Controller{
  public ModelAndView handleRequest(HttpServletRequest request,
                        HttpServletResponse response) throws Exception{

        SampleVO vo  = new SampleVO();
        String id = request.getParameter("seq");

        vo.setId(seq);

     SampleDAO dao = new SampleDAO();
     SampleVO sample = dao.getSample(vo);

     ModelAndView mv = new ModelAndView();
     mv.addObject("sample",sample); // 보낼 model
     mv.setViewName("sample.jsp"); // 띄울 view

        return mv;
  }
}
```

dispatcherServlet-context.xml

```
<!-- HandlerMapping 등록-->
```

```
<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
        <prop key="/sample.do">sample</prop>
        </props>
</property>
</bean>
```

b. @RequestMapping

SampleController.java

```
@Controller
public class SampleController {
  @RequestMapping("/sample.do")
  public Spring handleRequest(SampleVO vo,SampleDAO dao,Model model){
      // Command 객체에 의해서 Setter Injection 됨

      model.addAttribute("sample", dao.getSample(vo));

          return "sample.jsp";
     // Spring Container는 자동적으로 문자열 반환값과 일치하는 View를 찾고
     // model도 같이 전송해준다.
  }
```

b. Service

i. Service는 단일 데이터의 CRUD를 처리하는 DAO와 다르게 트랜잭션(한 작업) 단위로 여러 DAO를 호출하여 사용자의 요구에 맞게 가공한다.

+ 트랜잭션 : 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위
+ Ex) 다른 사람의 계좌로 송금할 때
    + (트랜잭션) 다른 사람에게 계좌로 송금한다.
    + (계좌 송금하는 과정 중 일부 : CRUD)
        + 보내고자 하는 대상의 계좌를 입력한다.
        + 보낼 금액을 입력한다.

ii. Controller에 의해 호출되어 실제 비즈니스 로직과 트랜잭션을 처리하고, DB CRUD를 담당하는 DAO 객체를 Spring으로부터 주입받아서, DAO DB CRUD 처리를 위임하고, 처리 결과를 Controller에게 반환

iii.

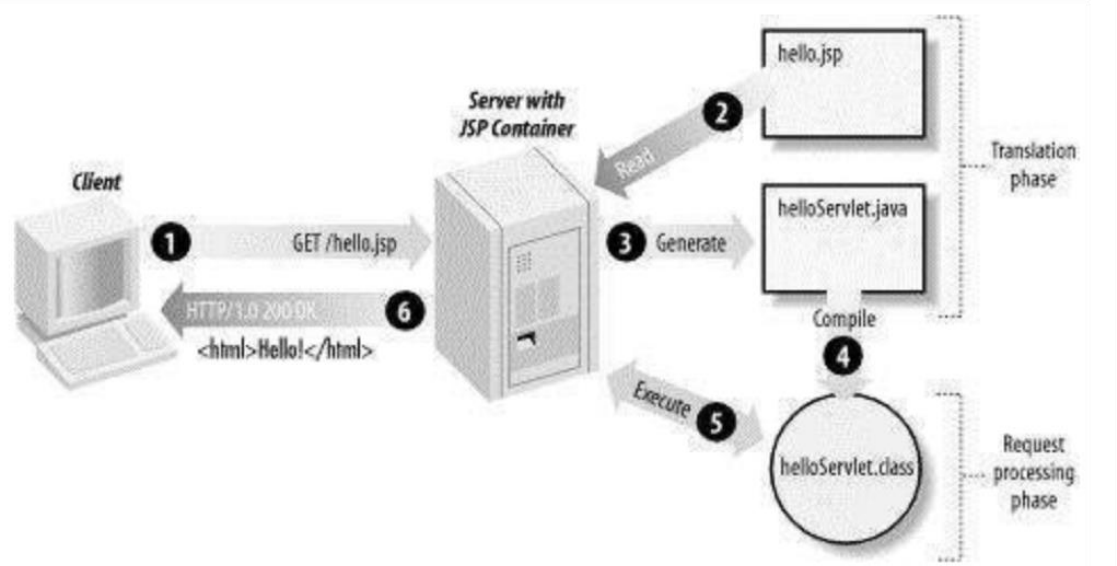
3. JSP란? Form 데이터 주고받는 방법

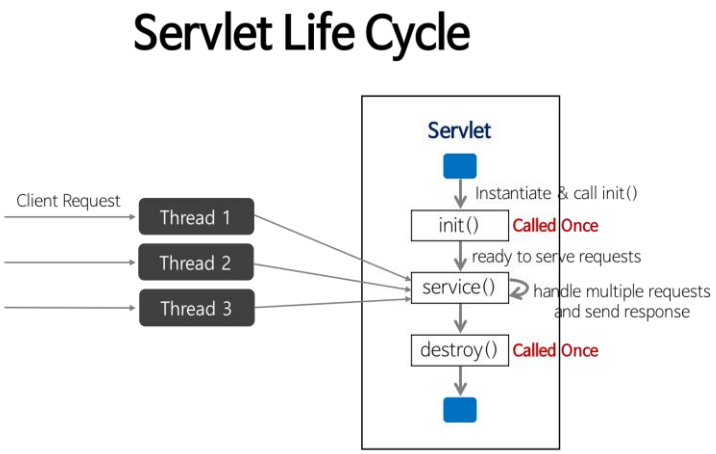
a. JSP란?

i. Java 언어를 기반으로 하는 Server Side 스크립트 언어
ii. HTML 코드에 Java 코드를 넣어 동적인 웹 페이지를 생성하는 웹 어플리케이션 도구
   - JSP를 통해 정적인 HTML과 동적으로 생성된 contents(HTTP request parameter)를 혼합하여 사용한다.
   - 사용자가 입력한 contents에 따라 동적으로 웹 페이지를 생성함
iii. HTML 코드 안에 Java 코드

```
<td><%= board.getTitle()%></td>
```

b. JSP의 내부 동작 과정





i. JSP가 실행되면 WAS는 내부적으로 JSP파일을 Java Servlet.java으로 변환한다.

ii. WAS는 변환한 Servlet을 동작하여 필요한 기능을 수행한다.

1. WAS는 사용자 요청에 맞는 적절한 Servlet 파일을 컴파일(.class 파일 생성)한다.
2. .class 파일을 메모리에 올려 Servlet 객체를 만든다.
3. 메모리에 로드될 때 Servlet 객체를 초기화하는 init() 메서드가 실행된다.
4. WAS는 Request가 올 때마다 thread를 생성하여 처리한다.
5. 각 thread는 Servlet의 단일 객체에 대한 service() 메서드를 실행한다.
6. service() 메서드는 요청에 맞는 적절한 메서드(doGet, doPost 등)를 호출한다.

iii. 수행 완료 후 생성된 데이터를 웹 페이지와 함께 클라이언트로 응답한다.

c. JSP특징

i. predefined values(=Implicit Object)

1. 미리 정의된 객체로, WAS가 제공하는 객체를 의미한다.
   a. request : HttpServletRequest Object
   b. response : HttpServletResponse Object
   c. session : HttpSession Object
   d. out : PrintWriter Object
   e. application : ServletContext Object

ii. custom tags

1. 사용자 정의 태그(custom tags)를 사용
2. JSTL(JSP Standard Tag Library) 사용

iii. expression language

1. JSP에서 데이터를 표현할 때 사용하는 언어
2. EL은 변수를 선언하기 위한 언어가 아니기 때문에 변수나 객체를 선언해서 사용하지 않고 기존의 데이터를 가져와서 사용한다.
3. ${objectName}, ${objectName.property}

iv. 수정 사항이 있을 때마다 컴파일한 후 재배포를 해야하는 Servlet과 다르게 JSP는 수정된 경우 재배포할 필요없이 Tomcat(WAS)이 알아서 처리해준다.

d. JSP문법

i. JSP Expression

- <%= expression %>
- JSP Expression element는 String으로 변환되어 Servlet의 출력에 삽입됨 (php - echo)

```
<%= application.getServerInfo() %>
```

ii. JSP Scriptlet

- <% code fragment %>
- 임의의 Java 코드를 삽입할 수 있다.
- 해당 태그에서 메서드가 아닌 변수만 선언할 수 있다.

```
<%
        BoardVO vo = new BoardVO();
        BoardDAO boardDAO = new BoardDAO();
        List<BoardVO> boardList = boardDAO.getBoardList(vo);
%>
```

iii. JSP Declaration

- <!% declaration %>
- 해당 태그를 사용하면 Servlet 클래스에 삽입되는 메서드나 필드를 정의할 수 있다.
- JSP Scriptlet Tag와 달리 메서드와 변수 모두 선언 가능

```
<%! private int accessCount = 0; %>
```

iv. JSP Comment

- 주석
- <%-- comment --%>

v. JSP Directive

- <%@ directive %>
- 지시어로 JSP 페이지의 전체 구조에 영향을 줌
- 전체 구조에 대한 지시를 WAS의 Container에게 내림
- page, include, taglib
- 지시어
    - page
        - <%@ page attribute = "value" %>
        - import, contentType
        - Container에 명령을 제공
    - include
        - <%@ include file = "relative_url"%>
        - include 지시어는 변환 단계에서 다른 외부 파일의 내용을 현재 JSP에 병합하도록 Container에 지시

```
<%@ include file = "header.jsp"%>
<center>
 <p> 글 목록 </p>
</center>
<%@ include file = "footer.jsp"%>
```

+ 고정되는 내용이 자주 반복되면 따로 jsp 파일을 작성해서 include 지시어로 병합하면 가독성을 높일 수 있다.

    - taglib
        - <%@ taglib uri="uri" prefix="prefixOfTag"%>
        - JSP API를 사용하면 HTML 또는 XML 태그처럼 보이는 사용자 정의 태그(custom tags)를 정의할 수 있다.

```
<%@ taglib uri=http://java.sum.com/jsp/jstl/core
prefix="c" %>
```

```
<c:out value="hello world"></c:out>
```

vi. **JSP Action**

- JSP Action XML 구문 안의 구조들을 사용하여 WAS의 동작을 제어

1. **<jsp:forward>**
   - 다른 리소스로 요청을 전달하는데 사용

```
<jsp:forward page="/display.jsp"/>
```

2. **<jsp:include>**
   - 현재 JSP에 다른 리소스를 포함시키는데 사용

```
<jsp:include page="/jsp/common/uppermen.jsp"
flush="true"/>
```

3. **<jsp:useBean>**
   - 해당하는 JavaBean이 이미 존재하는지 확인하고 객체가 없으면 지정된 객체를 생성한다

```
<jsp:useBean id="user" class="beans.user"/>
```

4. **<jsp:setProperty>**
   - Bean의 속성을 설정

```
<jsp:setProperty property="name" name="user"/>
```

5. **<jsp:getProperty>**
   - 주어진 속성값을 가져오는데 사용되며 이를 문자열로 변환하고 동적인 웹 페이지를 생성하는데 해당 속성값을 사용할 수 있다.

```
<jsp:getProperty property="email" name="user"/>
```

+ name이 user인 객체의 email 필드를 읽어온다

e. Form 데이터를 주고 받는 방법

i. HTML과 마찬가지로 form tag와 그 속성인 action, method를 이용해서 데이터를 주고 받는다.

1. action : 수신할 곳
2. method : 전송방법

```
<form action= "/request" method= "post">
        <p><input name= "id" type= "text"></p>
</form>
```

ii. “/request”와 매핑된 Servlet 클래스에서 디폴트 객체인 request와 메서드 getParameter(“inputName”)을 통해 받아온 request 내의 데이터를 이용할 수 있다.

```
@RequestMapping("/request")
public String request(HttpServletRequest request){
        String id = request.getParameter("id");
        return id;
}
```

+ Ajax를 사용해서 데이터를 전달하는 방법도 있다.

4. 어노테이션? XML?

a. XML

i. Extensible Markup Language
ii. 데이터에 대한 정보(메타 데이터)를 표시하기 위해 사용(=설정파일)
iii. 사용자 정의 태그(custom tag)가 가능
iv. Spring에서의 XML
    1. applicationContext.xml : bean 관련 설정
    2. dispatcher-servlet.xml : 내부 웹 관련 처리 작업 설정
    3. web.xml : tomcat 구동 관련 설정
    4. pom.xml : Maven build, dependency, ProjectInfo 에 대한 설정
v. applicationContext.xml
    1. DI를 java코드와 별개인 파일에서 할 수 있기 때문에 java 코드를 일일이 확인하지 않아도 XML 파일만 봐도 DI가 어떻게 설정되어있는지 확인 가능하다.
    2. 하지만, 대규모 프로젝트에서 모든 DI를 XML에만 작성하면 XML 코드의 길이가 매우 길어져서 오히려 유지보수가 어려워질 수 있기 때문에 변경될 여지가 있는 객체만 XML로 설정하고 변경되지 않고 고정으로 있을 객체는 Annotation으로 설정하는 것이 좋다.
    3. DI, AOP 설정 가능

b. 어노테이션

i. 주석 기능과 더불어 선언적 프로그래밍 모델(해당 데이터가 무엇을 할지 선언)을 지원하는 기술
ii. 컴파일, 런타임 시에 반영됨
iii. 설계 시 확정되는 부분은 Annotation 기반 설정으로 개발의 생산성을 향상 시킬 수 있음

iv. Spring Annotation 종류

1. 컴포넌트 계열 Annotation
   해당 Annotation이 붙은 클래스는 Bean 객체가 된다. 해당 클래스들은 컴포넌트 스캐닝의 대상이 되고, 스프링 Container에 의해서 관리될 것임을 표시한다.

```
<context:component-scan base-package="com.spring.biz.board"/>
```

   a. @Controller
      - Servlet 클래스에 주로 사용
      - 프레젠테이션 계층
      - 해당 객체를 MVC 아키텍처에서 컨트롤러 객체로 인식하게 해줌

   b. @Service
      - *ServiceImpl 클래스에 주로 사용
      - 비즈니스 계층

   c. @Repository
      - *DAO 클래스에 주로 사용
      - 영속성 계층
      - DB 연동 과정에서 발생하는 예외를 변환해주는 기능 존재

```
@Service
public class MemberServiceImpl{}
```

   위 클래스는 별도의 이름이 명시되어 있지 않으므로 소문자로 시작하는 memberServiceImpl bean 객체가 생성된다.

2. DI Annotation

   a. @Autowired
      - 의존대상 객체를 메모리에서 타입으로 검색해서 주입
      - 동일한 타입의 객체가 없거나 여러 개일 경우 Exception 발생

   b. @Resource(name="bean id")
      - 의존 대상 객체를 이름으로 주입
      - (=@Qualifier)
      - @Autowired의 단점 보완 가능

- 동일한 타입의 객체가 여러 개 있어도 하나의 객체를 지정해서 주입할 수 있음

c. @Inject
   - 의존 대상 객체를 타입으로 검색해 주입
   - @Named("bean id")
     - @Inject와 @Named를 함께 명시하면 이름으로 주입

5. Spring MVC 프로젝트 생성 및 실습

a. 링크 : https://github.com/HyunSeokJung-student/schoolpoint