

- 1주차 ~ 4주차 : //스프링을 위한 선수 지식 : Servlet 요점 정리
- SpringProject1: (5주차) //개발자에 의한 객체 결합 (교재 p39~48)
- SpringProject2: (6주차) //스프링컨테이너에 의한 객체 결합 (XML 방식) - **IoC**- (교재 p49~91)
- SpringProject3: (7주차) //스프링컨테이너에 의한 객체 결합 (어노테이션 방식) - **IoC**- (교재 p93~108)
- SpringProject4_BizStep1: (8주차) //비즈니스 컴포넌트 실습 1 (게시판) -**Business Layer**- (교재 p109~127)
- SpringProject4_BizStep2: (9주차) // 비즈니스 컴포넌트 실습 2 (User추가) -**Business Layer**- (교재 p129~139)
- SpringProject5_AOP_step1: (9주차) // 스프링 AOP 개념 파악 - **AOP** - (교재 p143~170) 9주 과제
- SpringProject5_AOP_step2: (10주차) // AOP 동작 시점 (5개) 실습- **AOP** - (교재 p171~182) 10주 과제
- SpringProject5_AOP_step3: (10주차) // AOP 동작 시점과 JointPoint/바인드변수 실습- **AOP** - (교재 p183~192)
- SpringProject5_AOP_step4: (10주차) // 어노테이션 방식의 AOP 실습- **AOP** - (교재 p193~192)
- SpringProject6_Jdbc_step1: (11주차) // Spring JDBC -스프링 JDBC - (교재 p209~226) 11주 과제
- SpringProject6_Jdbc_step2: (11주차) // 트랜잭션 처리-트랜잭션 - (교재 p227~237)

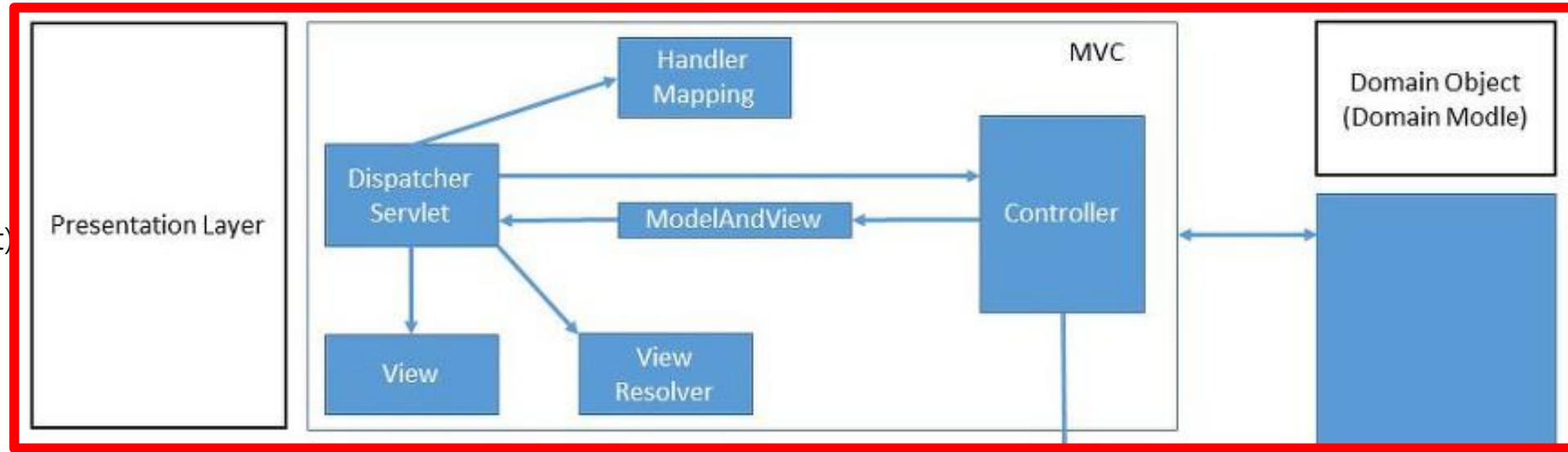


- SpringProject7_MVC_step1 : (12주차) // MVC 패턴 - Model 1 아키텍처 - (교재 p241~261)
- SpringProject7_MVC_step2 : (12주차) // MVC 패턴 - Model 2 아키텍처 - (교재 p263~283)
- SpringProject7_MVC_step3 : (12주차) // MVC 패턴 - MVC 프레임워크 아키텍처 - (교재 p285~311)
- SpringProject7_MVC_step4 : (12주차) // 스프링 MVC - Spring MVC 구조 (xml) - (교재 p313~345)
- SpringProject7_MVC_step5_1 : (14주차) // 스프링 MVC - Spring MVC 구조(어노테이션 1)- (교재 p349~370)
- SpringProject7_MVC_step5_2 : (14주차) // 스프링 MVC - Spring MVC 구조(어노테이션 2)- (교재 p371~394)
- SpringProject7_MVC_step5_3 : (14주차) // 스프링 MVC - 레이어 통합 - (교재 p395~408)
- SpringProject7_MVC_step5_4 : (14주차) // 스프링 MVC - 검색기능 추가 - (교재 p409~418)
- Mybatis 프레임워크 시작하기 개요 - (교재 p467~)



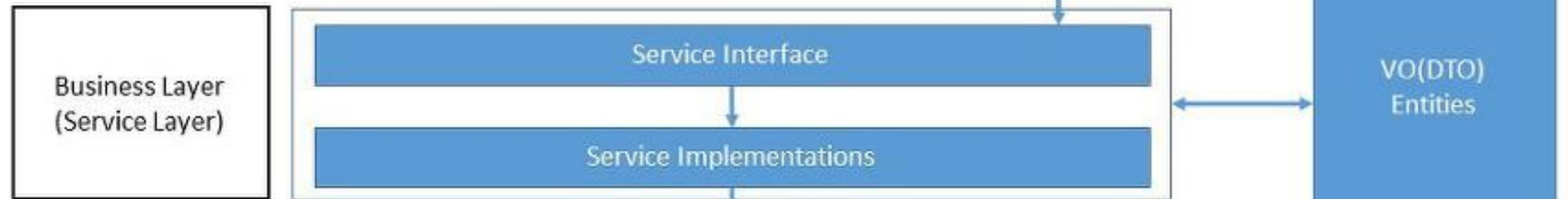
1) Presentation Layer

Spring MVC 객체를 말한다.
프론트 컨트롤러(DispatcherServlet),
컨트롤러, 뷰, 모델이 포함



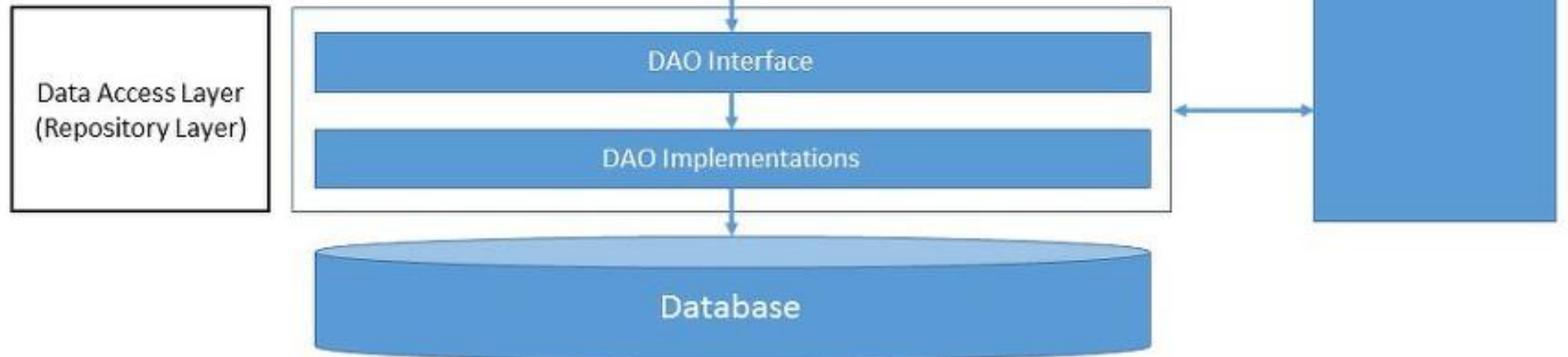
2) Service Layer(Business Layer)

presentation layer에서 요청을 보내면 실제로 비즈니스로직 수행



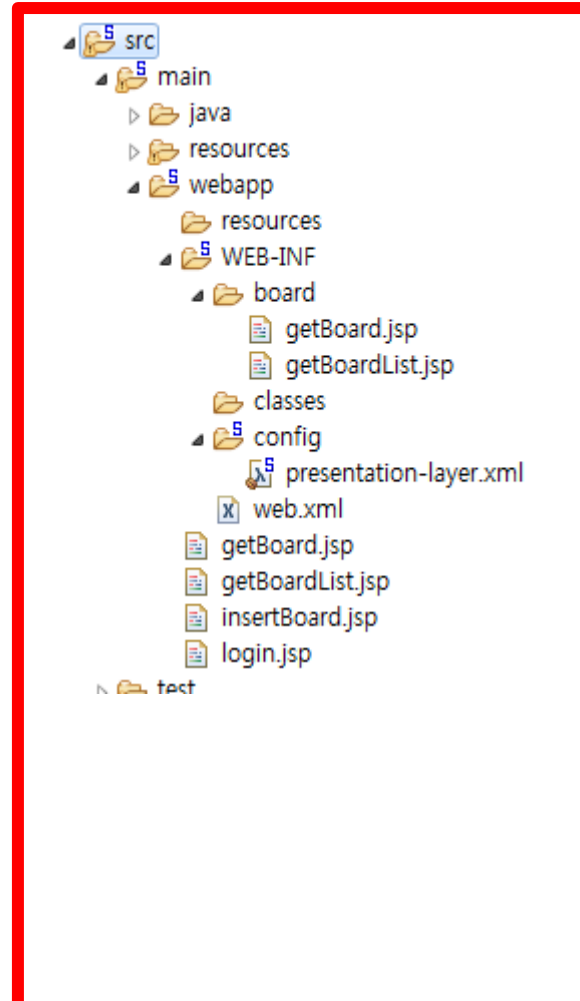
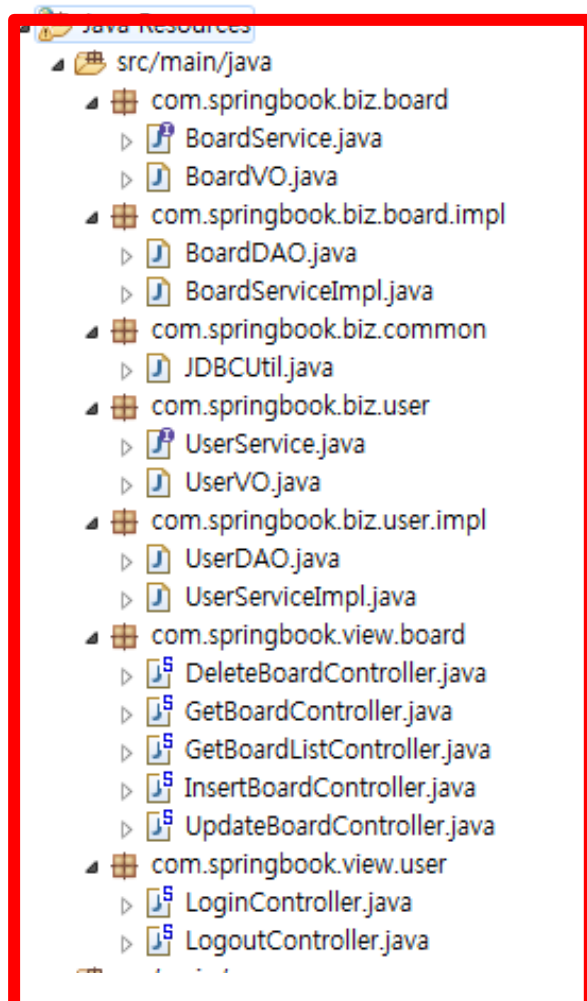
3) Data Access Layer(Repository Layer)

DB에 값을 저장하거나 가져오기 위해 JDBC, Mybatis, JPA 등을 사용해 구현한 DAO



Annotation 설정을 위한 준비 (presentation-layer.xml)

```
<context:component-scan base-package="com.springbook.view"/>
```



• @Controller

- @Controller가 붙은 클래스를 메모리에 생성하고 Controller 객체로 인식하도록 한다.
- Controller를 POJO(Plain Old Java Object) 스타일로 코딩할 수 있다.

@Controller

```
public class InsertBoardController {  
  
}
```

```
<!-- HandlerMapping 등록 -->  
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandle  
    <property name="mappings">  
        <props>  
            <prop key="/login.do">login</prop>  
        </props>  
    </property>  
</bean>
```

xml방식

```
<!-- Controller 등록 -->  
<bean id="login" class="com.springbook.view.user.LoginController"></bean>  
  
<!-- ViewResolver 등록 -->  
<bean id="viewResolver"  
    class="org.springframework.web.servlet.view.InternalResourceView  
    <property name="prefix" value="/WEB-INF/board/"></property>  
    <property name="suffix" value=".jsp"></property>  
</bean>
```

• @RequestMapping

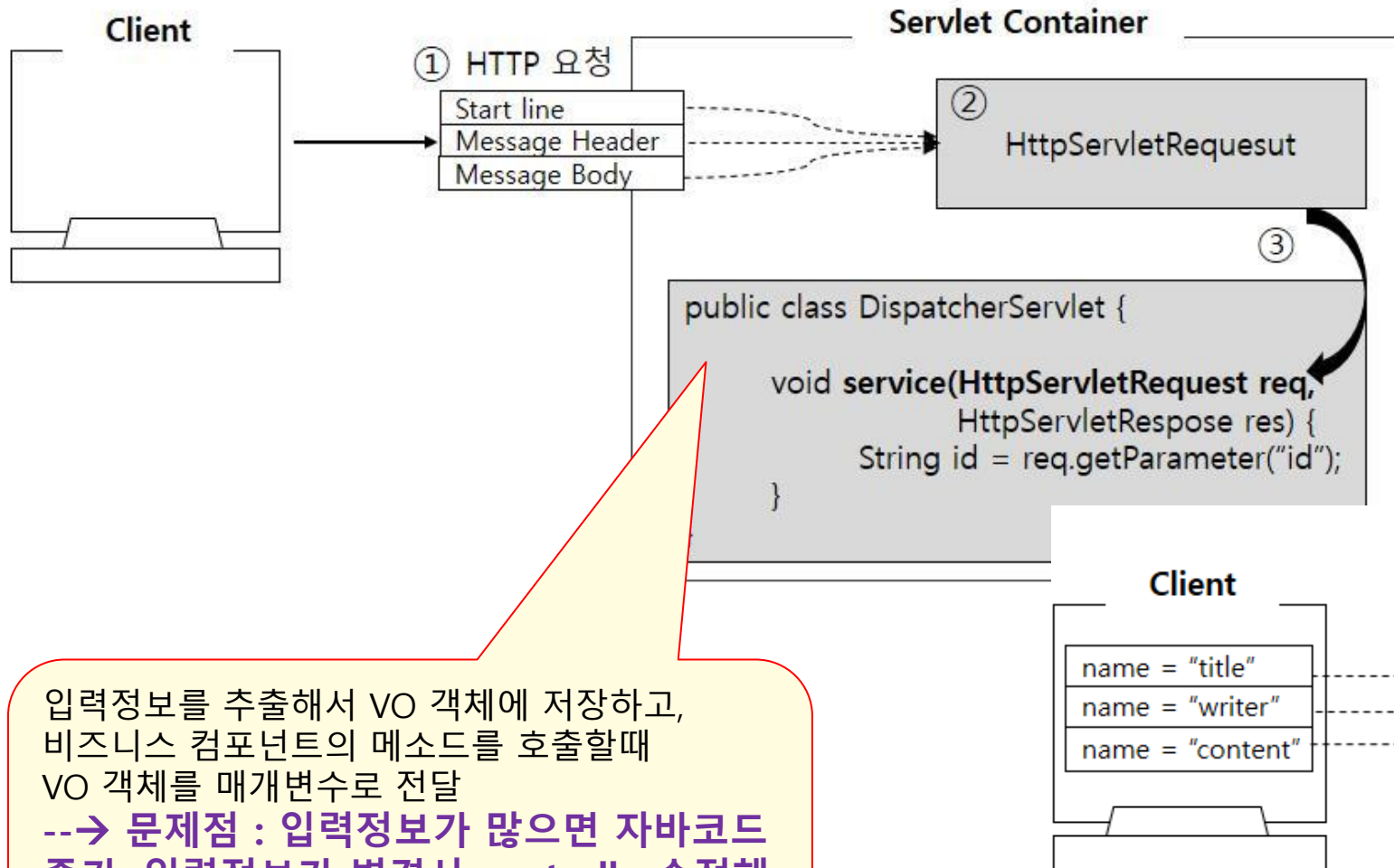
- 클라이언트의 요청 path에 대해 실행될 메소드를 매핑한다.

@Controller

```
public class InsertBoardController {  
    @RequestMapping(value="/insertBoard.do")  
    public void insertBoard(HttpServletRequest request) {  
    }  
}
```



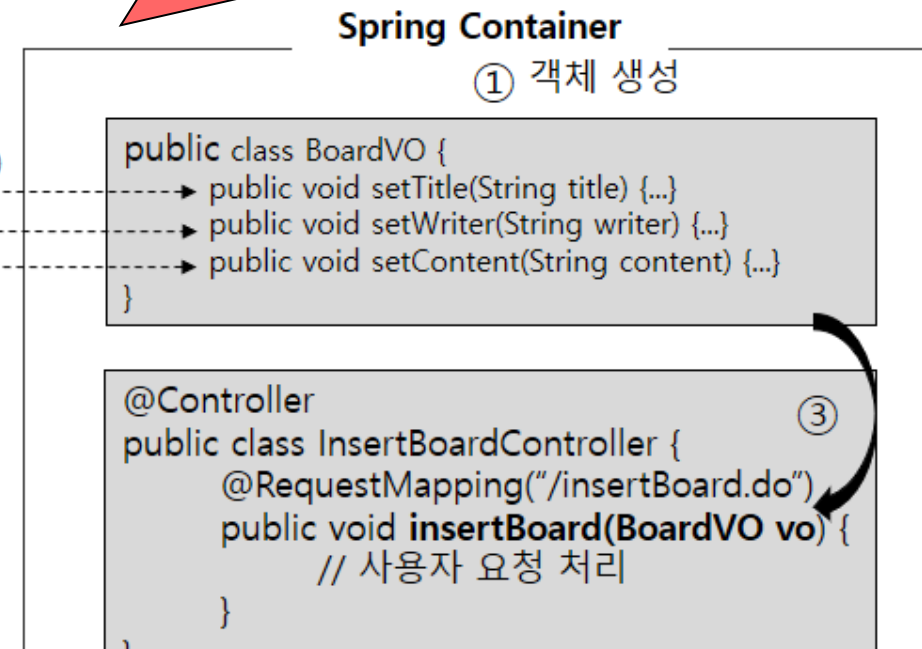
클라이언트의 요청 처리를 위한 입력정보 추출



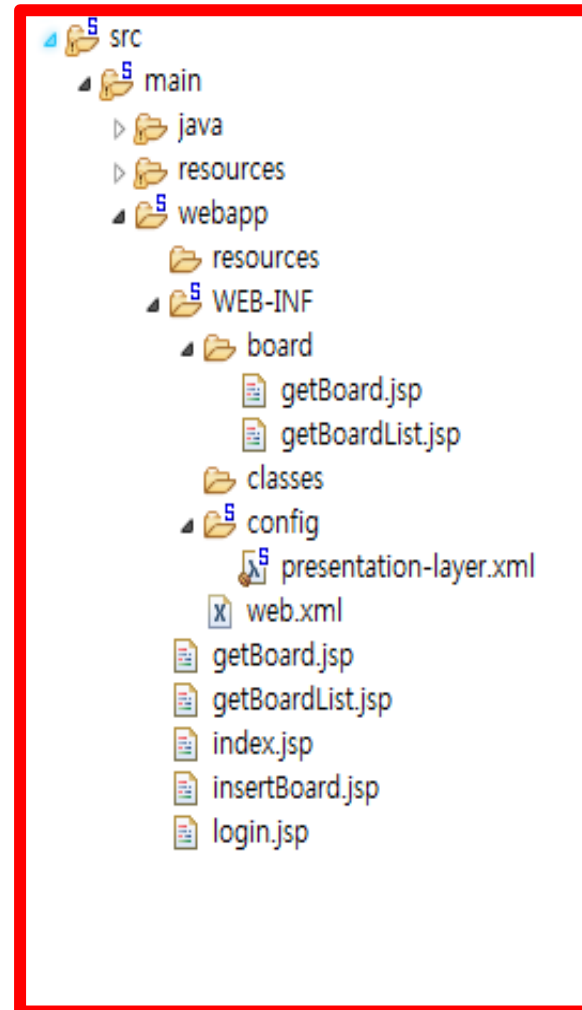
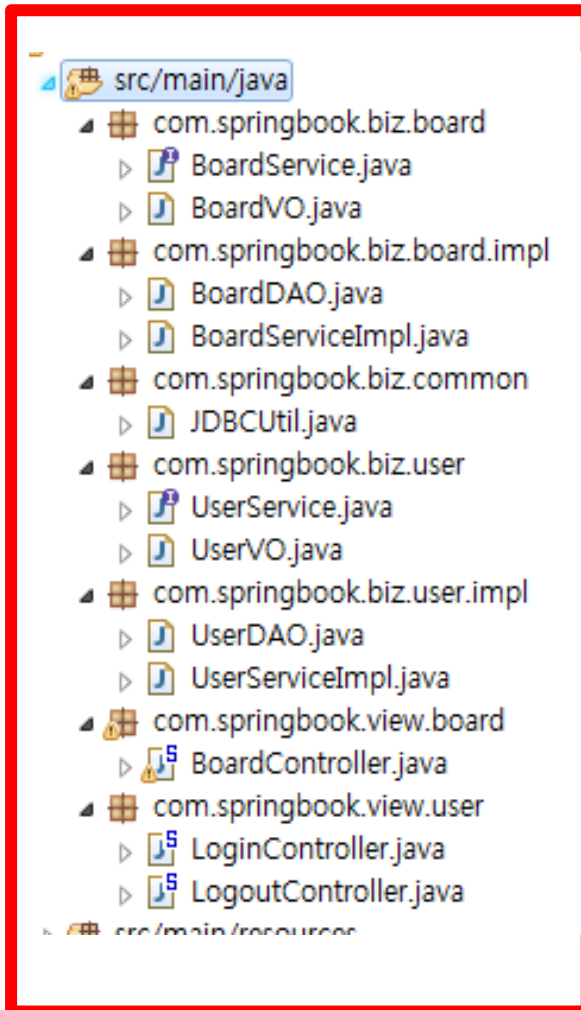
입력정보를 추출해서 VO 객체에 저장하고,
비즈니스 컴포넌트의 메소드를 호출할때
VO 객체를 매개변수로 전달
--> 문제점 : 입력정보가 많으면 자바코드
증가, 입력정보가 변경시 controller수정해
야함

• 커맨드객체란?
폼 요청을 처리하는 컨트롤러는 각 파라미터의 값을
구하기 위해 HttpServletRequest 객체를 이용할 수
있지만
문제점이 있어서 스프링에서는 커맨드객체를
지원함.
요청 파라미터의 이름을 이용한 setter 메소드를
작성한 클래스(VO)를 만들고, 이 클래스의
객체(커맨드 객체)를 메소드의 파라미터 값으로
넣어주면 스프링은 요청 파라미터의 값을 커맨드
객체에 담아준다.
즉, Command객체는 Controller 메소드 매개변수로
받는 VO 객체라고 보면 됨.

Command 객체 사용!!!



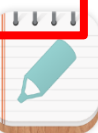
컨트롤러 통합하기



- 요청 방식에 따른 @RequestMapping 사용

```
@Controller
public class LoginController {
    @RequestMapping(value="/login.do", method=RequestMethod.GET)
    public String loginView(UserVO vo) {
        vo.setId("test");
        vo.setPassword("test123");
        return "login.jsp";
    }

    @RequestMapping(value="/login.do", method=RequestMethod.POST)
    public String login(UserVO vo, UserDao userDao) {
        if(userDao.getUser(vo) != null) return "getBoardList.do";
        else return "login.jsp";
    }
}
```



- @Controller 의 메서드 파라미터 타입

파라미터 타입	설명	
HttpServletRequest, HttpServletResponse, HttpSession	서블릿 API	교재 p378
java.util.Locale	현재 요청에 대한 Locale	
InputStream, Reader	요청 콘텐츠에 직접 접근할 때 사용	
OutputStream, Writer	응답 콘텐츠를 생성할 때 사용	
@PathVariable	URI 템플릿 변수에 접근할 때 사용	
@RequestParam 어노테이션 적용파라미터	HTTP요청 파라미터를 매핑	교재 p382
@RequestHeader 어노테이션 적용파라미터	HTTP요청 헤더를 매핑	
@CookieValue 어노테이션 적용파라미터	HTTP쿠키 매핑	
@RequestBody 어노테이션 적용파라미터	HTTP요청의 몸체 내용에 접근할 때 사용	
@ModelAttribute 어노테이션 적용파라미터	HTTP요청 파라미터를 매핑, 뷰에 데이터를 전달	교재 p385
Map, Model, ModelMap	뷰에 전달할 모델 데이터를 설정할 때 사용	
Error, BindingResult	HTTP요청 파라미터를 커맨드 객체에 저장한 결과, 커맨드객체 파라미터 바로 뒤에 위치해야한다.	
SessionStatus	폼처리를 완료 했음을 처리하기 위하여 사용	



- JSP에서 Command 객체 사용하기

```
<table border="1" cellpadding="0" cellspacing="0">
  <tr>
    <td bgcolor="orange">아이디</td>
    <td><input type="text" name="id" value="${userVO.id }"/></td>
  </tr>
  <tr>
    <td bgcolor="orange">비밀번호</td>
    <td><input type="password" name="password" value="${userVO.password }"/></td>
  </tr>
  <tr>
    <td colspan="2"><input type="submit" value="로그인"></td>
  </tr>
</table>
```

- Command객체의 이름을 바꾸고 싶으면....

```
@RequestMapping(value = "/login.do", method = RequestMethod.GET)
public String loginView(@ModelAttribute("user") UserVO vo) {
    System.out.println("로그인 화면으로 이동...");
}
```

• 커맨드 객체의 이름은 특별히 변경하지않으면....
클래스 이름의 첫 글자를 소문자로 한 이름으로
자동 설정!

예) UserVO클래스 → userVO,
BoardVO클래스 → boardVO

```
<table border="1" cellpadding="0" cellspacing="0">
  <tr>
    <td bgcolor="orange">아이디</td>
    <td><input type="text" name="id" value="${user.id }"/></td>
  </tr>
</table>
```

- Servlet API 사용

- Servlet 에서 제공하는 HttpServletRequest, HttpServletResponse, HttpSession, Locale 등 다양한 객체를 매개 변수로 받을 수 있다.

```
@RequestMapping(value="/login.do", method=RequestMethod.POST)
public String login(UserVO vo, UserDao userDao, HttpSession session) {
    UserVO user = userDao.getUser(vo);
    if(user != null) {
        session.setAttribute("userName", user.getName());
        return "getBoardList.do";
    }
    else return "login.jsp";
}
```



- Command객체에는 없는 파라미터를 Controller클래스에서 사용하려면?

→ HTTP 요청 파라미터 정보를 추출하기 위한 @RequestParam 이용

@Controller

public class BoardController {

 @RequestMapping("/getBoardList.do")

 public String getBoardList(

 @RequestParam(value="searchCondition", defaultValue="TITLE", required=false) String condition,

 @RequestParam(value="searchKeyword", defaultValue="", required=false) String keyword) {

 System.out.println("검 째색" + condition);

 System.out.println("검 단색" + keyword);

 return "getBoardList.jsp";

 }

}

value : 화면으로부터 전달될 파라미터 이름

defaultValue : 화면으로부터 전달될 파라미터 정보가 없을 때의 설정할 기본값

required : 파라미터의 생략 여부

<!-- 검색 시작 -->

<form action="getBoardList.do" method="post">

 <table border="1" cellpadding="0" cellspacing="0" width="700">

 <tr>

 <td align="right">

 <select name="searchCondition">

 <c:forEach items="\${conditionMap}" var="option">

 <option value="\${option.value}">\${option.key}

 </c:forEach>

 </select>

 <input name="searchKeyword" type="text" />

 <input type="submit" value="검색" /> </td>

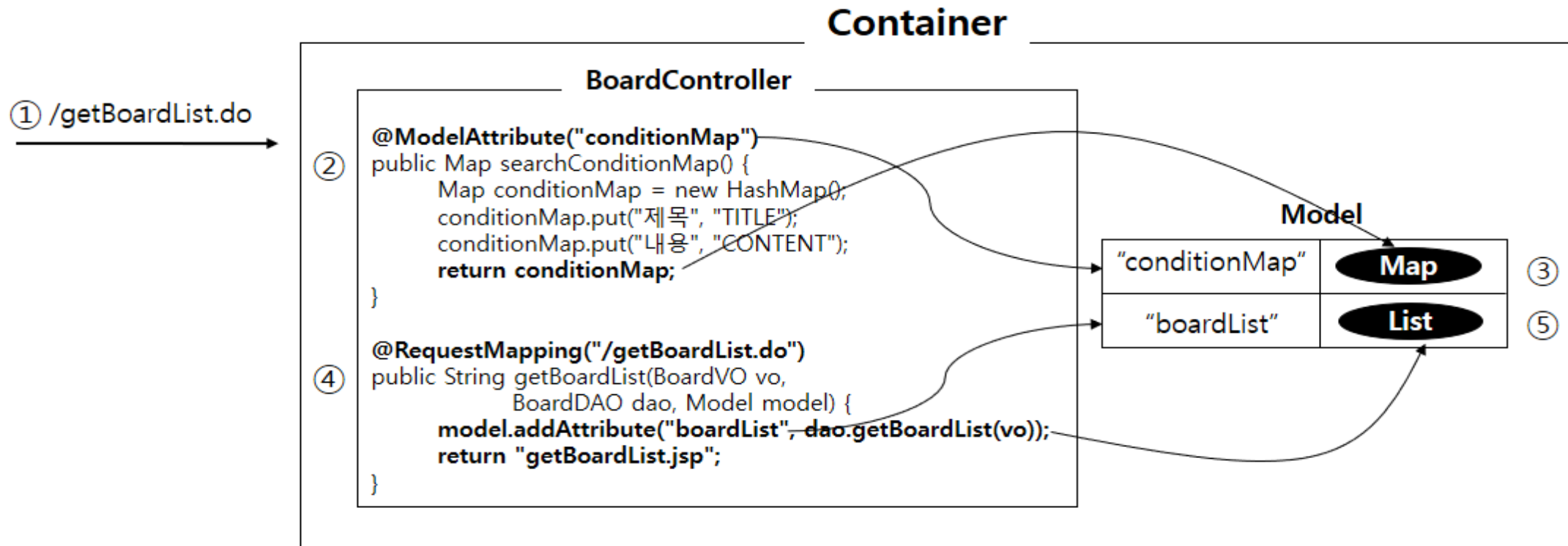
 </tr>

 </table>

 </form>

• @ModelAttribute 사용하기 (1)

- Command객체의 이름을 변경할 목적으로 사용
- View에서 사용할 데이터를 설정하는 용도로도 사용
- @RequestMapping어노테이션이 적용된 메소드보다 먼저 호출
- @ModelAttribute메소드의 실행결과로 리턴된 객체는 자동으로 Model에 저장



- @ModelAttribute 사용하기 (2)

@ModelAttribute("conditionMap")

```
public Map<String, String> searchConditionMap() {  
    Map<String, String> conditionMap = new HashMap<String, String>();  
    conditionMap.put("제목", "TITLE");  
    conditionMap.put("내용", "CONTENT");  
    return conditionMap;  
}  
  
@RequestMapping("/getBoardList.do")  
public String getBoardList(BoardVO vo, BoardDAO boardDAO, Model model) {  
    // Model 정보 저장  
    model.addAttribute("boardList", boardDAO.getBoardList(vo));  
    return "getBoardList.jsp";  
}
```

```
<table border="1" cellpadding="0" cellspacing="0" width="700">  
<tr>  
    <td align="right">  
        <select name="searchCondition">  
            <c:forEach items="${conditionMap}" var="option">  
                <option value="${option.value}">${option.key}</option>  
            </c:forEach>  
        </select>  
        <input name="searchKeyword" type="text"/>  
        <input type="submit" value="검색"/>  
    </td>  
</tr></table>
```

- @SessionAttributes 사용하기 (1)

update board set title=?, content=?, writer=? where seq=? 인 경우.. writer값이 null 이면....

제목	임시 제목3
작성자	홍길동
내용	임시 2.
등록일	2020-05-29
조회수	0

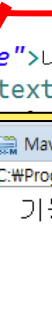
글 수정

글등록글삭제글목록

```
<tr>
  <td bgcolor="orange" width="70">제목</td>
  <td align="left"><input name="title" type="text"
    value="${board.title }" /></td>
</tr>
<tr>
  <td bgcolor="orange">작성자</td>
  <td align="left">${board.writer }</td>
</tr>
<tr>
  <td bgcolor="orange">내용</td>
  <td align="left"><textarea name="content" cols="40">
```

Console Problems Debug Shell Maven Repositories Servers
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_201\bin
==> JDBC로 getBoard() 기능 처리
번호 : 5
제목 : 임시 제목3
작성자 : null
내용 :
등록일 : null
조회수 : 0

4	5 임시 제목3	홍길동	임시 2.	20/05/29	0
---	----------	-----	-------	----------	---



4	5 임시 제목3	(null)	임시 2.
---	----------	--------	-------



- @SessionAttributes 사용하기 (2)

@SessionAttributes("board")

Model에 SessionAttribute로 저장된 이름의 데이터가 있을 경우, 그 데이터를 세션(HttpSession)에도 자동으로 저장하는 설정

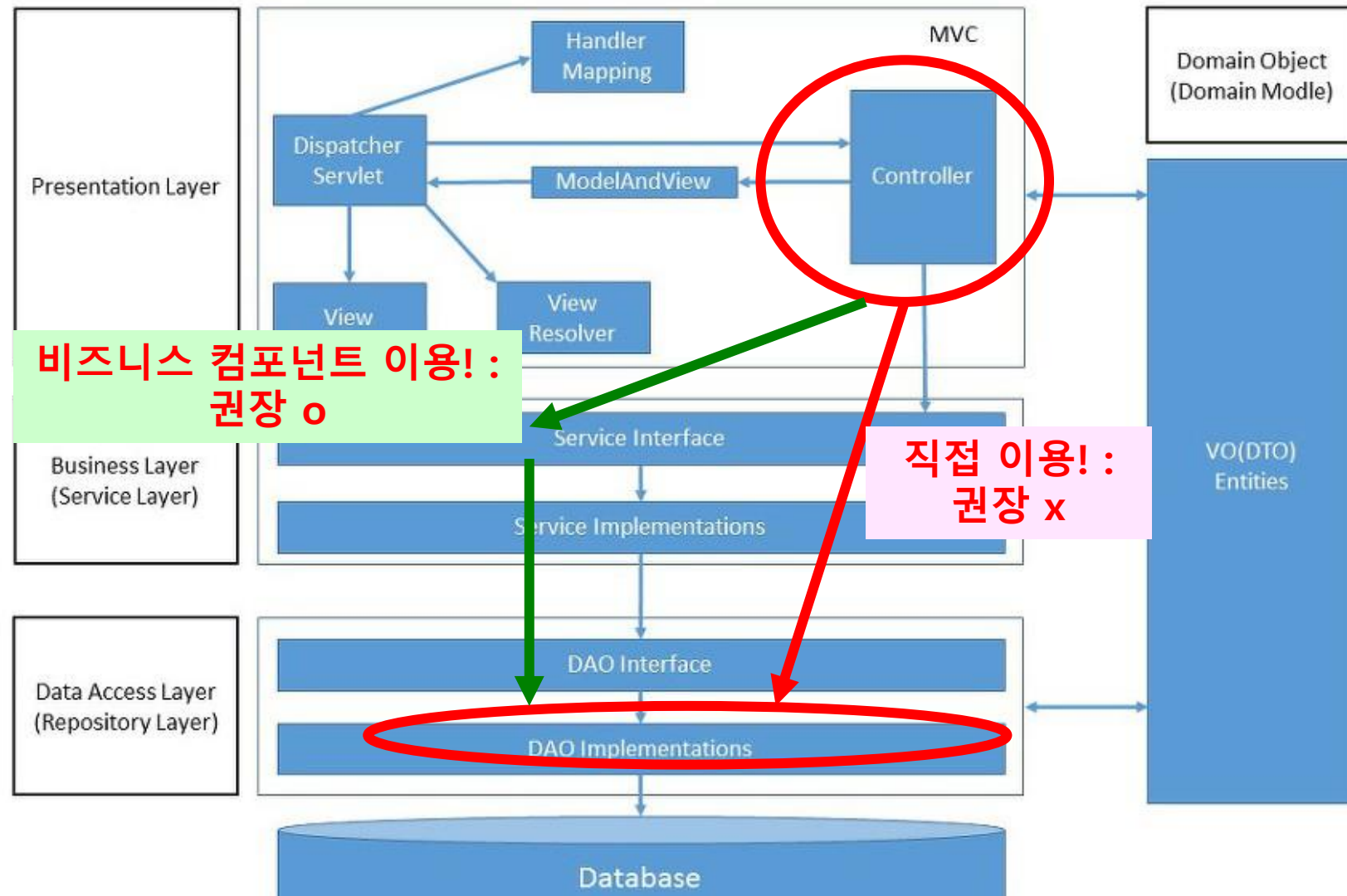
```
@Controller
@SessionAttributes("board") ...
@RequestMapping("/updateBoard.do")
public String updateBoard(@ModelAttribute("board") BoardVO vo) {
    ...
    boardDAO.updateBoard(vo);
    ...
}
```

1. Controller의 updateBoard() 메소드가 호출될 때,
스프링 컨테이너는 우선 @ModelAttribute("board") 설정을 해석하여 세션에 'board'라는 이름으로 저장된 데이터가 있는지 확인
2. 만약 있다면 해당 객체를 세션에서 꺼내서 매개변수로 선언된 vo 변수에 할당
3. 그리고 사용자가 입력한 파라미터 값을 vo 객체에 할당한다.
4. 이때 사용자가 입력한 수정 정보(title, content...) 값만 새롭게 할당되고, 나머지(seq, writer, regDate...)는 세션에 저장된 데이터가 유지



(Presentation-Layer, Business-Layer)

지금까지는 Controller가 DAO객체를 직접 이용 -> 이것은 여러가지 문제점 발생 야기시킴



<문제점>

1. 유지보수時 DAO클래스를 쉽게 교체 못함!
2. AOP 적용 불가능!

그래서,,,

Controller클래스는 비즈니스 컴포넌트의 인터페이스 타입의 멤버변수를 가지고 있어야 하며, 이 변수에 비즈니스 객체를 의존성 주입해야 함! (교재 p401)



< BoardController >

```
public class BoardController {
    @Autowired
    private BoardService boardService;

    @RequestMapping(value = "/insertBoard.do")
    public String insertBoard(BoardVO vo) {
        boardService.insertBoard(vo);
        return "getBoardList.do";
    }
    .....
}
```



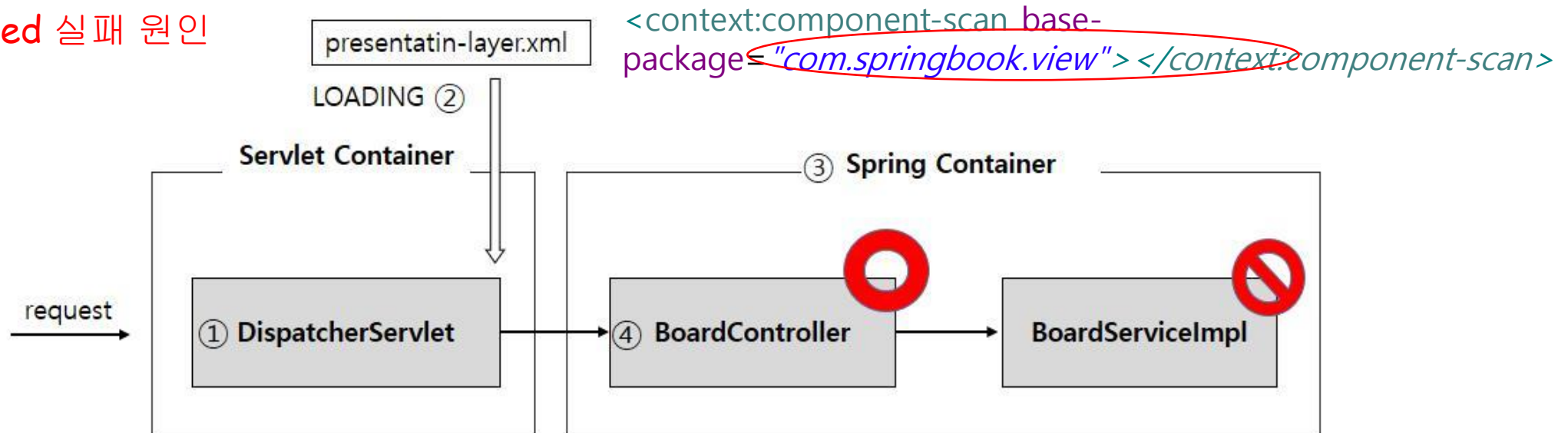
Console Problems Debug Shell Maven Repositories Servers

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (2020. 6. 18. 오전 5:25:38)

AutowiredAnnotationBeanPostProcessor - JSR-330 'javax.inject.Inject' annotation found and supported but no beans were found in context to satisfy the injection point.

WebApplicationContext - Exception encountered during context initialization - cancelling refresh attempt: org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'boardController': Injection of autowired dependencies failed

Autowired 실패 원인



지금까지는..

- 비즈니스 레이어 설정파일 : src/main/resource 폴더의 applicationContext.xml
- 프레젠테이션 레이어 설정파일 : WEB-INF/config폴더의 presentation-layer.xml

presentation-layer.xml을 읽어 Controller 객체가 생성되기 전에 비즈니스 컴포넌트들을 먼저 생성하기 위해 applicationContext.xml을 읽어야 함

: **스프링에서 제공하는 ContextLoaderListener 클래스!**

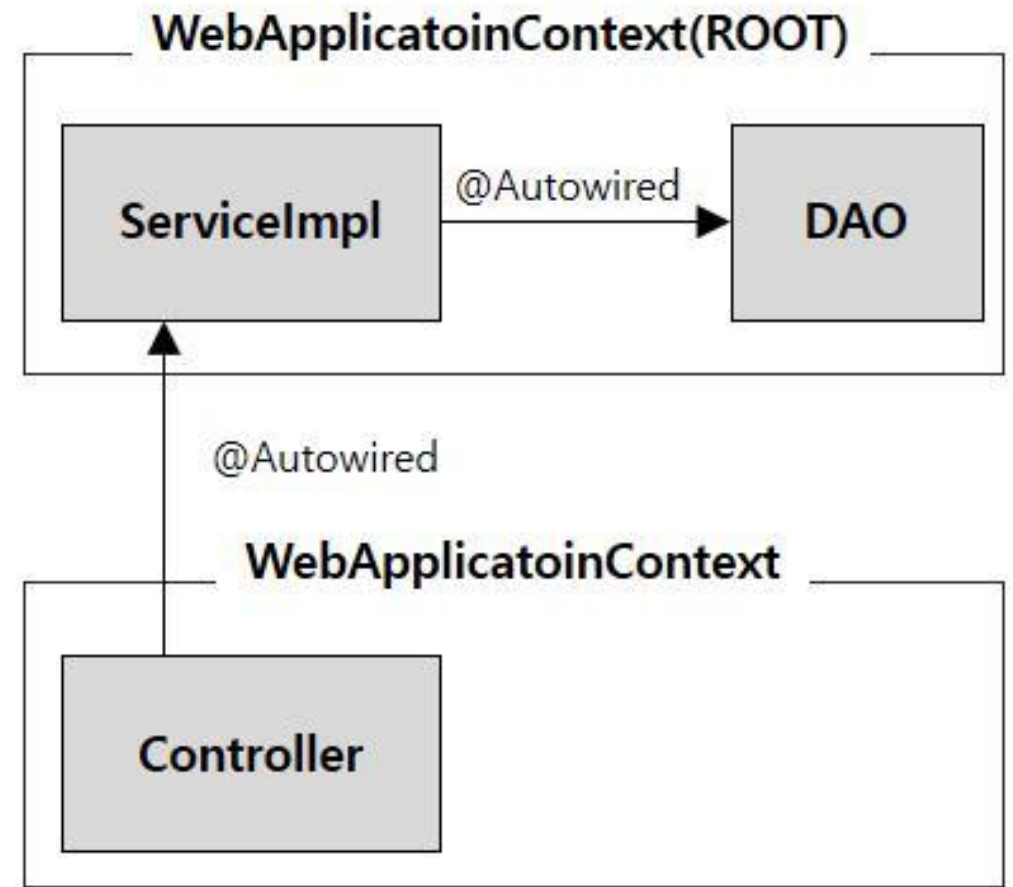
web.xml

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    classpath:applicationContext.xml
  </param-value>
</context-param>
```

스프링 컨테이너의 관계

1. 톰캣 서버 구동
2. web.xml 로딩하여 서블릿 컨테이너 구동
3. 서블릿 컨테이너는 web.xml에 등록된 ContextLoaderListener객체 생성(Pre Loading)
 ➡ application.xml 로딩하여 **"Root 컨테이너"** 구동
 이때, ServiceImpl/DAO객체 생성
4. Client의 "*.do"요청이 들어오는 순간,
 DispatcherServlet객체 생성
5. DispatcherServlet객체가 presentation-layer.xml 로딩
 하여 **두 번째 스프링 컨테이너** 구동
 ➡ Controller 객체 생성



- getBoardList.jsp 확인
- 커맨드 객체 수정 : BoardVo.java
- Controller 수정 : getBoardList() 메소드 수정
- DAO 클래스 수정 :

BoardDAO.java 혹은 BoardDAOSpring 클래스 수정



Mybatis



스프링 JDBC



JDBC

- 전통적인 SQL과 JDBC API 조합만으로는 엔터프라이즈 애플리케이션의 복잡하고 거대한 데이터를 처리하는 깔끔한 코드를 만드는 데는 많은 제약과 문제점 존재
- 자바 계열의 다양한 데이터 액세스 기술
 - JDBC의 사용을 추상화해주는 iBatis / Mybatis , 오픈소스 ORM(Object Relation Mapping)의 대표적인 Hibernate, JPA ,
- Mybatis는 자바 Object와 SQL문 사이의 자동 매핑 기능을 지원하는 ORM 프레임워크
 - 특징 : **한두 줄의 자바 코드로** DB 연동 처리, **SQL명령어를 XML 파일로** 따로 관리

```
public class BoardDAO {  
    private SqlSession mybatis;  
  
    public BoardDAO() {  
        mybatis = SqlSessionFactoryBean.getSqlSessionInstance...  
    }  
  
    public void insertBoard(BoardVO vo) {  
        mybatis.insert("BoardDAO.insertBoard", vo);  
        mybatis.commit();  
    }  
  
    public void updateBoard(BoardVO vo) {  
        mybatis.update("BoardDAO.updateBoard", vo);  
        mybatis.commit();  
    }  
}
```

```
*board-mapping.xml  
1 <?xml version="1.0" encoding="UTF-8"?>  
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
3 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
4  
5 <mapper namespace="BoardDAO">  
6  
7     <insert id="insertBoard">  
8         insert into board(seq, title, writer, content)  
9         values((select nvl(max(seq), 0)+1 from board),#{title},#{writer},#{content})  
10    </insert>  
11  
12    <update id="updateBoard">  
13        update board set title=#{title}, content=#{content} where seq=#{seq}  
14    </update>  
15  
16    <delete id="deleteBoard">
```