H2 DataBase, Spring을 이용한 Web 게시판

진명국

목차

spring, mybatis, h2 database

spring

- **Context.xml에서 compnent-scan으로 해당 패키지를 스캔**
- 해당 클래스의 **Bean객체를 자동으로 생성해 주고 관리**
- Component의 속성에는 Controller, Service, DAO
- AutoWired는 해당 변수의 타입을 체크하고 객체가 메모리에 존재하면 변수에 주입

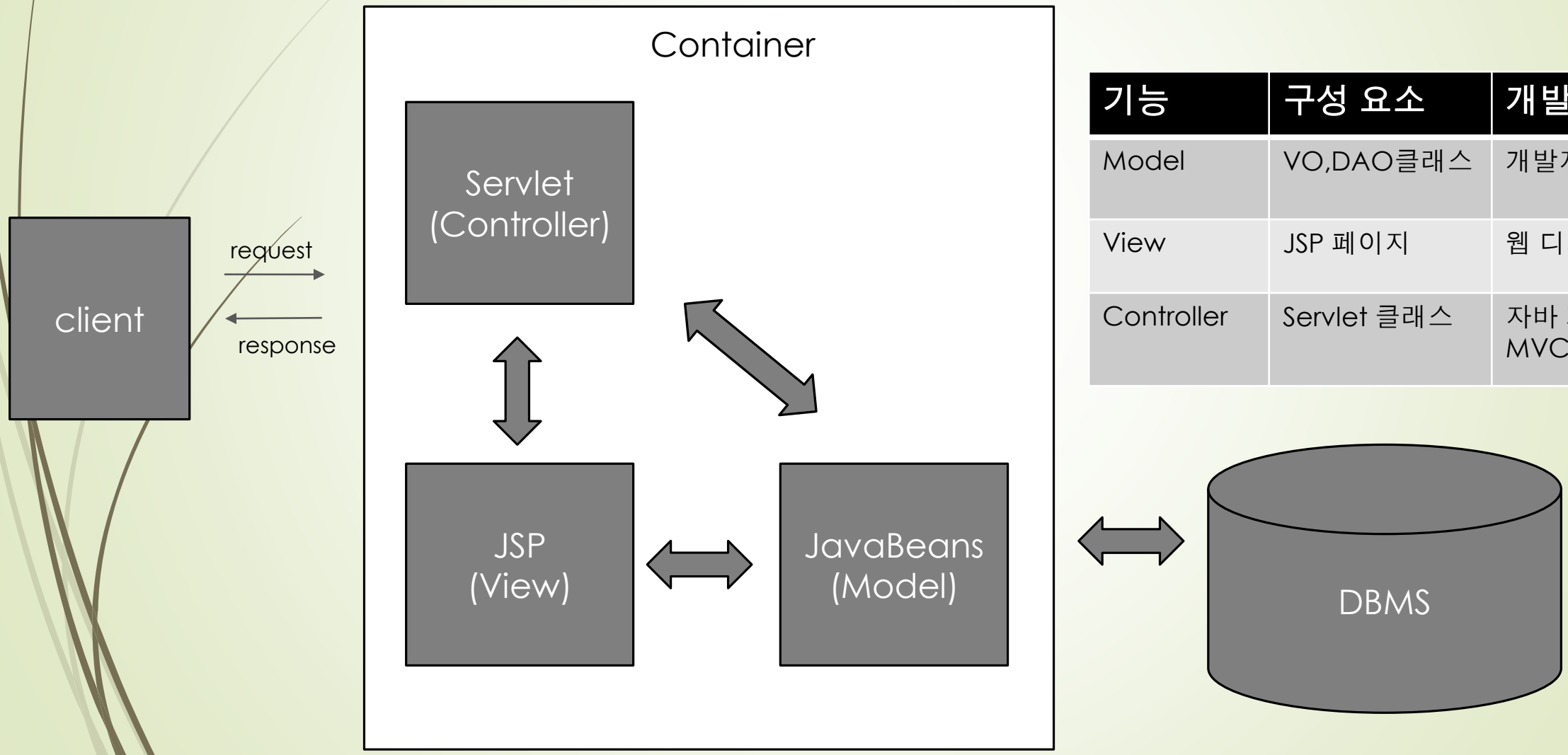
mybatis

- mybatis는 XML파일에 저장된 SQL 명령어를 대신 실행하고 실행결과를 VO같은 자바 객체에 **자동으로 매핑해주는 역할**
- <mapper>는 namespace 속성을 가집니다. 네임스페이스가 지정된 Mapper의 SQL을 참조

H2 DATABASE

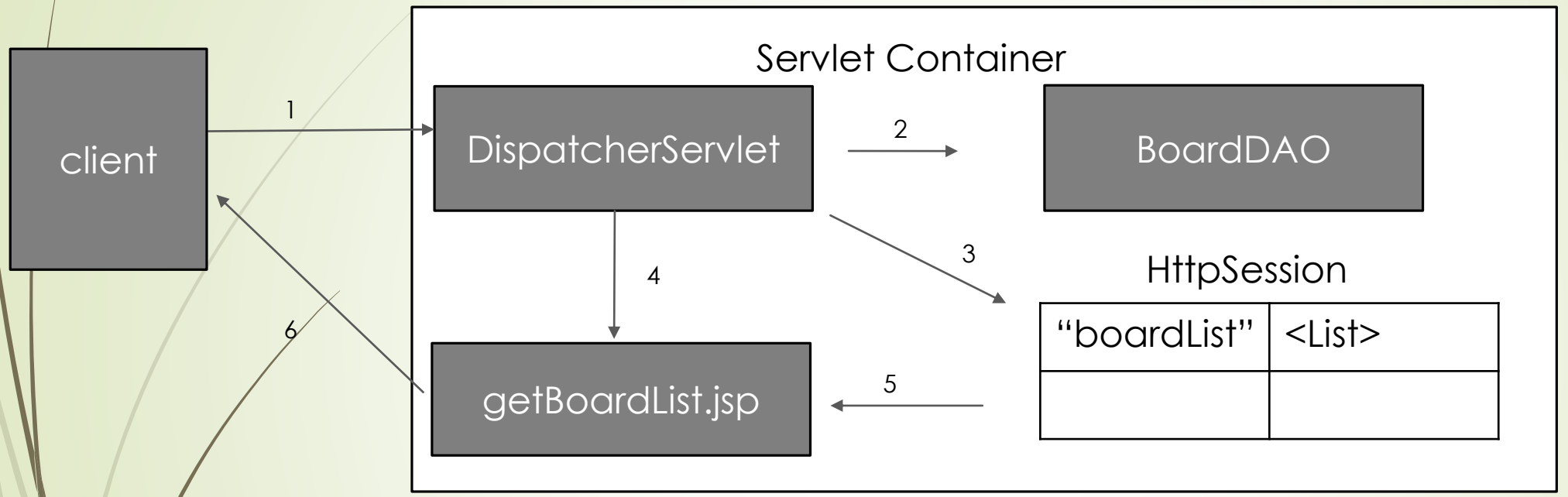
- **2MB정도의 적은 용량으로 설치** 가능
- 브라우저 기반 콘솔 프로그램
- 대규모 프로젝트에서는 **안정성과 성능이 부족하다는 단점**이 있습니다.

MVC1 패턴



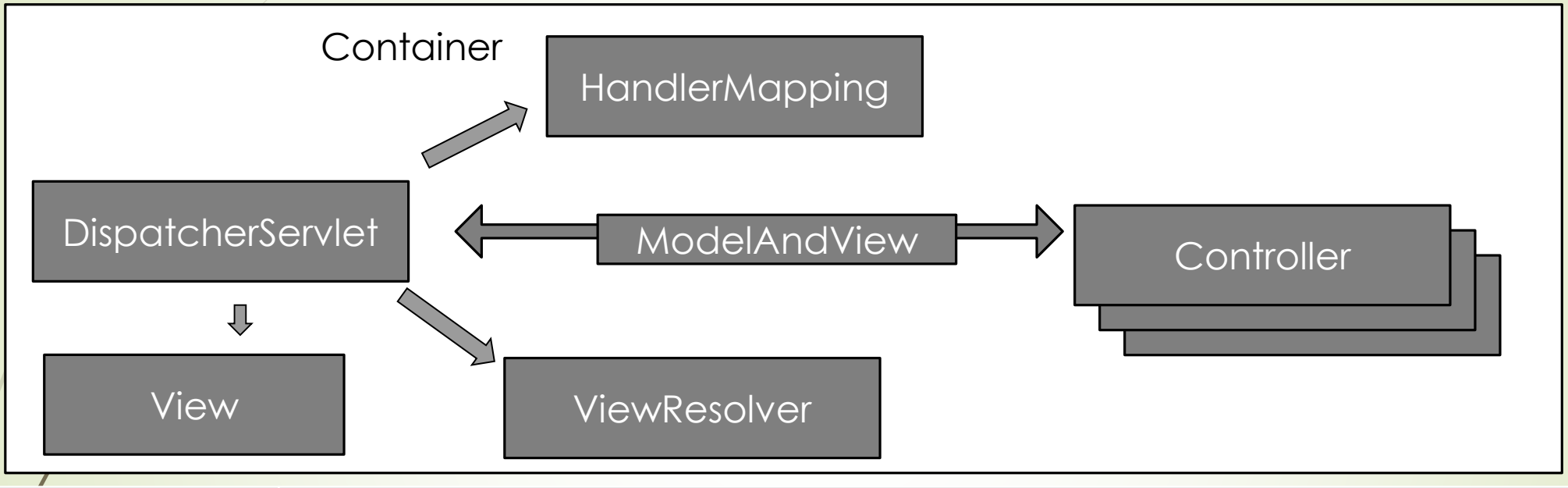
기능	구성 요소	개발 주체
Model	VO,DAO클래스	개발자
View	JSP 페이지	웹 디자이너
Controller	Servlet 클래스	자바 개발자 또는 MVC 프레임 워크

MVC1 패턴



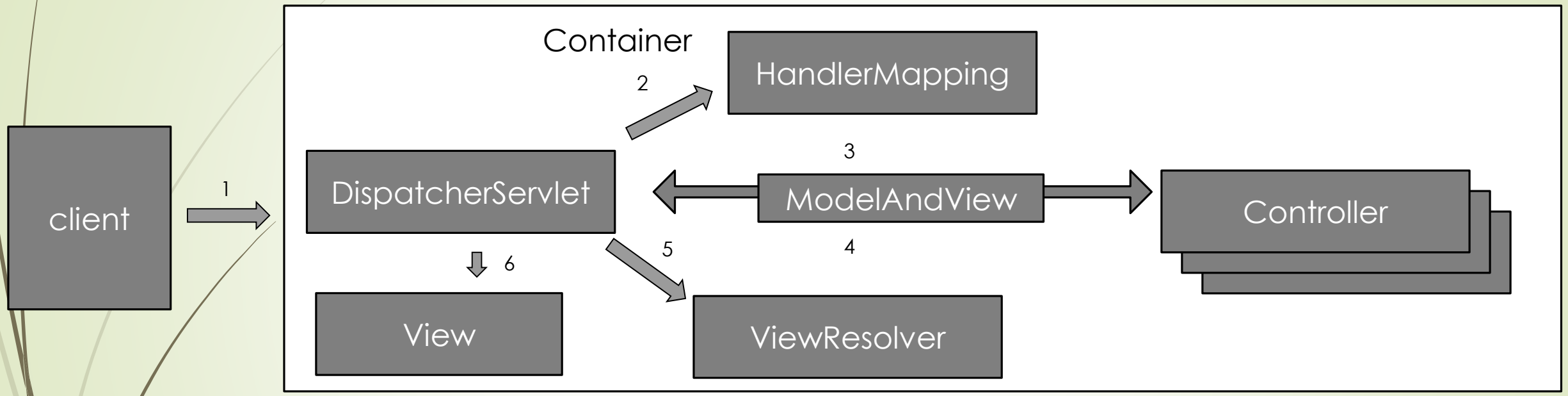
1. DispatcherServlet이 클라이언트의 /getBoardList.do를 요청받으면
2. **DispatcherServlet은 BoardDAO객체를 이용하여 글 목록을 검색**
3. 검색된 **글 목록을 세션에 등록**하고
4. getBoardList.jsp 화면을 요청하면
5. **getBoardList.jsp 는 세션에 저장된 글 록을 꺼내어 목록 화면을 구성**
6. 마지막으로 이 응답 화면이 **브라우저에 전송**

MVC2 패턴



클래스	기능
DispatcherServlet	유일한 서블릿 클래스로서 **모든 클래스의 요청을 가장 먼저 처리하는 Front Controller**
HandlerMapping	**클라이언트의 요청 Controller 객체를 검색**, 검색된 Controller를 실행 Map타입의 컬렉션을 멤버변수로 가지고 있으면서 모든 Controller의 객체를 등록 관리
Controller	실질적인 **클라이언트의 요청 처리**
ViewResolver	Controller가 리턴한 View 이름에prepix, suffix를 결합하여 **최종 실행 될 View의 경로와 파일명을 완성**

MVC2 패턴



1. 클라이언트가 로그인 버튼을 클릭하여 **".do"요청을 전송하면 DispatcherServlet**이 요청을 받음
2. DispatcherServlet은 **HandlerMapping** 객체를 통해 로그인 요청을 처리할 **Controller**를 **검색**하고
3. **DispatcherServlet**은 검색된 **Contoller**를 실행하여 클라이언트의 요청을 처리
4. **Controller**는 비지니스 로직의 수행 결과로 얻어낸 **Model** 정보와 **Model** 을 보여줄 **View** 정보를 **ModelAndView** 객체에 저장하여 리턴
5. DispatcherServlet은 **ModelAndView**로부터 **View** 정보를 추출하고, **ViewResolver**를 이용하여 응답으로 사용할 **View**를 얻음
6. DispatcherServlet은 **ViewRrsolver**를 통해 찾아낸 **View**를 실행하여 응답을 전송

DispatcherServlet

DispatcherServlet

```
<!-- Processes application requests -->
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/config/presentation-layer.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

- "/*.do"**요청이 있을 때 DispatcherServlet**는 객체를 생성
- DispatcherServlet 클래스에 재정의 된 **init()**메서드가 자동으로 실행되어 **XmlWebApplicationContext**라는 스프링 컨테이너가 구동되고 스프링 설정 파일인 **action – servlet.xml**을 로딩하여 XmlWebApplicationContext를 생성하고 스프링 컨테이너가 구동되는 것
- 스프링 설정파일 **action-servlet.xml**에 **DispatcherServlet**이 사용할 **Handler, Controller, ViewResolver** 클래스를 **<bean>** 등록하면 스프링 컨테이너가 해당 객체들을 생성

DispatcherServlet

DispatcherServlet

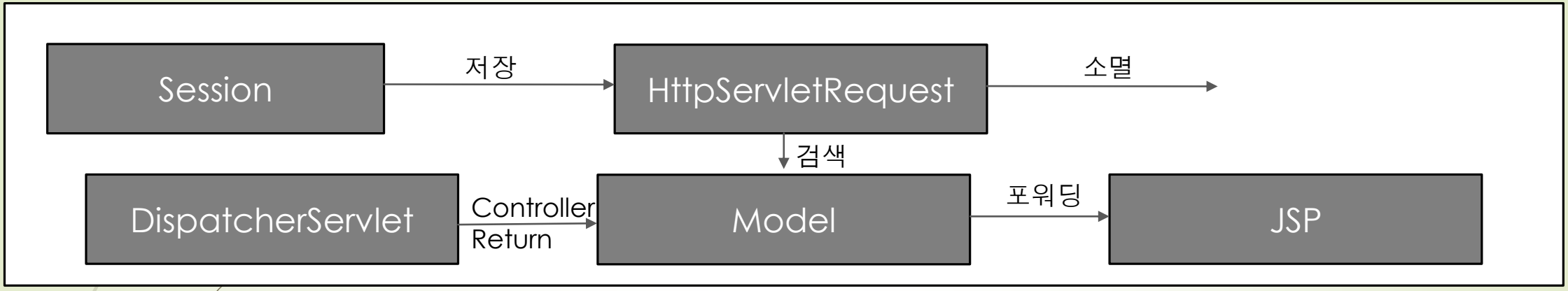
```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/config/presentation-layer.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
```

```
<filter>
    <filter-name>characterEncoding</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>EUC-KR</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>characterEncoding</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
```

- **contextConfigLocation**이라는 파라미터로 설정한 정보를 추출하여 스프링 컨테이너를 구동할 때 사용
- 스프링에서는 인코딩 처리를 위해 **CharacterEncodingFilter** 클래스를 제공
  DispatcherServlet은 직접 만든 클래스가 아니여서 인코딩 설정 필요
  url 패턴을 *.do로 처리하여 모든 클라이언트의 do요청에 일괄 적용

HttpServletRequest



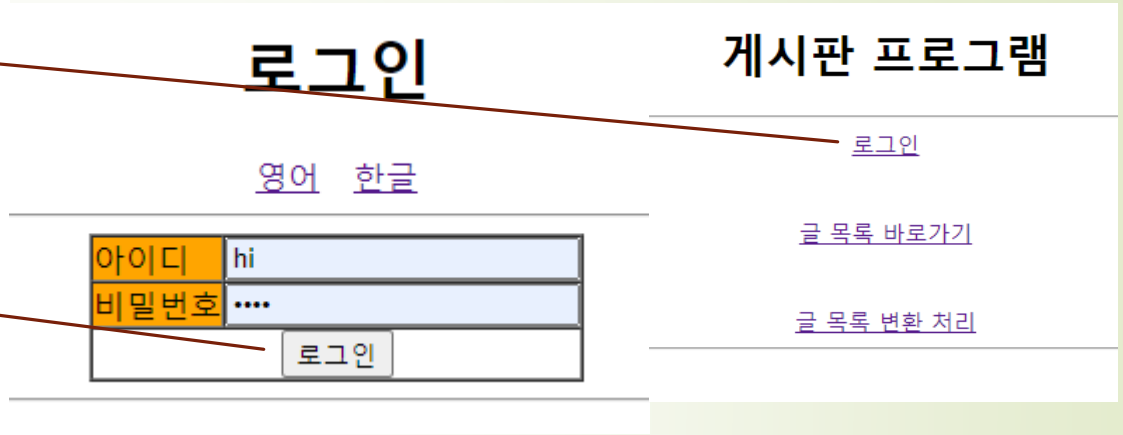
- **세션은 클라이언트 브라우저 하나당 하나씩 서버 메모리에 생성되어 클라이언트의 상태정보를 유지하기 위해서 사용**
- 세션에 많은 정보가 저장될수록 서버에 부담
- **검색 결과는 세션이 아닌 HttpServletRequest 객체에 저장**
- HttpServletRequest은 클라이언트의 요청으로 생성됐다가 **응답 메세지가 클라이언트로 전송되면 자동으로 삭제되는 일회성 객체**
- **DispatcherServlet은 Controller가 리턴한 ModelAndView 객체에서 Model 정보를 추출한 다음 HttpServlet 객체에 검색 결과에 해당하는 Model 정보를 저장하여 JSP로 포워딩**

method = requestMethod.GET/POST

```java
@Controller
public class LoginController {


    @RequestMapping(value = "/login.do", method = RequestMethod.GET)
    public String loginView(@ModelAttribute("user") UserVO vo) {
        System.out.println("로그인 화면으로 이동...");
//      vo.setId("test");
//      vo.setPassword("test123");
        return "login.jsp";

    }
    @RequestMapping(value = "/login.do", method = RequestMethod.POST)
    public String login(UserVO vo, UserDAO userDAO, HttpSession session) {
        if (vo.getId() == null || vo.getId().equals("")) {
            throw new IllegalArgumentException("아이디는 반드시 입력해야 합니다.");
        }
        UserVO user = userDAO.getUser(vo);
        if (user != null) {
            session.setAttribute("userName", user.getName());
            return "getBoardList.do";
```



- 클라이언트의 요청방식 **(GET/POST)에 따라 수행될 메서드를 다르게 설정**
- 클라이언트가 GET방식으로 입력 폼을 요청하면 입력화면을 보여주고,
  입력 화면에서 submit 버튼을 클릭하여 POST 방식으로 요청하면
  클라이언트의 요청을 적절히 처리하고자 할 때 이 방법을 사용
- **login()과 loginView() 메서드에 /login.do로 요청에 실행되도록 설정**
- 요청이 **GET방식이면 스프링컨테이너는 loginView() 를 실행**하고 **POST 방식이면**
  login()메서드를 실행하여 로그인 인증 작업을 처리

loginView(@ModelAttribute("user"))

LoginController.java

```
public String loginView(@ModelAttribute("user") UserVO vo) {
    System.out.println("로그인 화면으로 이동...");
```

login.jsp

```
<td bgcolor="orange"><spring:message code="message.user.login.id"/></td>
<td><input type="text" name="id" value="${user.id }" /></td>
```

- **SpringContainer**가 생성하는 **Command** 객체의 이름은 클래스 이름의 첫 글자를 소문자로 변경한 이름이 자동으로 설정
- 따라서 login.jsp화면에서 **UserVO** 객체의 변수에 접근 할 때 "**${userVO.변수명}**"을 사용
- Command 객체의 **이름을 변경하기위해 @ModelAttribute**를 사용

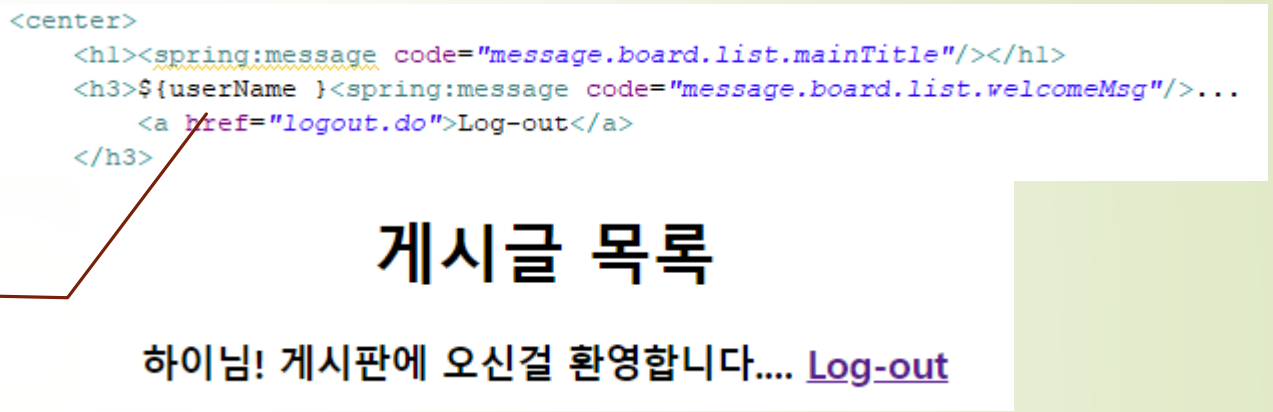
HttpSession

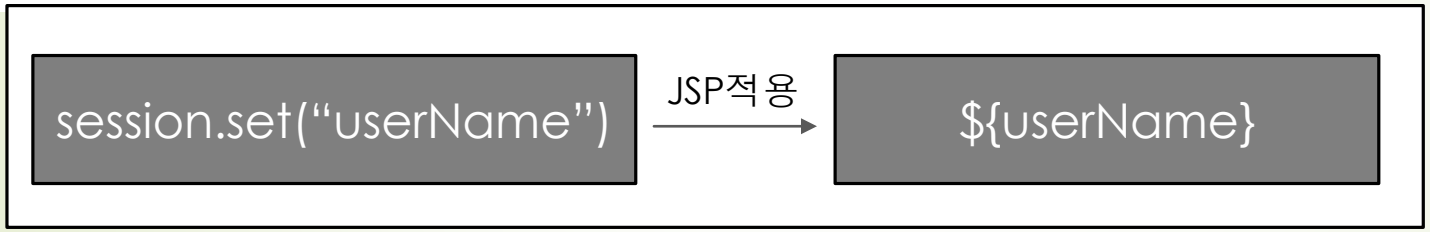
UserController.java

```
@RequestMapping(value = "/login.do", method = RequestMethod.POST)
public String login(UserVO vo, UserDAO userDAO, HttpSession session) {
    if (vo.getId() == null || vo.getId().equals("")) {
        throw new IllegalArgumentException("아이디는 반드시 입력해야 합니다.");
    }
    UserVO user = userDAO.getUser(vo);
    if (user != null) {
        session.setAttribute("userName", user.getName());
        return "getBoardList.do";
    } else
        return "login.jsp";
}
```

getBoardlist.jsp

```
<center>
    <h1><spring:message code="message.board.list.mainTitle"/></h1>
    <h3>${userName }<spring:message code="message.board.list.welcomeMsg"/>...
        <a href="logout.do">Log-out</a>
    </h3>
```





- **HttpSession**의 주된 메서드는 **getAttribute(String name)**, getId(), invalidate(), **removeAttribute(String name), setAttribute(String name, Object value)**
- HttpSession 객체를 매개변수로 받아서 **로그인 성공 시에 사용자 이름을 세션에 저장**
- 로근인에 성공할 때 사용자의 이름을 세션에 저장하기 위해 LoginController 소스를 **session.setAttribute("userName", user.getName());을 "getBoardlist.do"로 반환**

@ModelAttribute

BoardController.java
searchConditionMap()

```
// 검색 조건 목록 설정
@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;
}
```

- @ModelAttribute는 본 PPT 12페이지의 Controller메서드의 매개변수로 선언된 ModelAttribute 또는 **View(JSP)에서 사용할 데이터를 설정하는 용도로 사용** 가능
- ModelAttribute가 설정된 메서드는 **@RequestMapping 어노테이션이 적용된 메서드보다 먼저 호출**
- ModelAttribute 메서드 실행 결과로 **리턴된 객체는 자동으로 Model에 저장**
- ModelAttribute 메서드의 실행 결과를 **리턴된 객체를 View페이지에서 사용**

@ModelAttribute

BoardController.java
getBoardList

```java
// 검색 조건 목록 설정
@ModelAttribute("conditionMap")
public Map<String, String> searchConditionMap()
// 글 목록 검색
@RequestMapping("/getBoardList.do")
public String getBoardList(BoardVO vo, Model model) {
    // Null Check
    if (vo.getSearchCondition() == null)
        vo.setSearchCondition("TITLE");
    if (vo.getSearchKeyword() == null)
        vo.setSearchKeyword("");
    // Model 정보 저장
    model.addAttribute("boardList", boardService.getBoardList(vo));
    return "getBoardList.jsp"; // View 이름 리턴
}
```

BoardVO.java

```java
public String getSearchKeyword() {
    return searchKeyword;
}
public void setSearchKeyword(String searchKeyword) {
    this.searchKeyword = searchKeyword;
}
```

- searchConditionMap() 메서드 위에 @ModelAttribute(conditionMap)가 선언되었으므로 **getBoardList() 메서드가 실행되기 전에 먼저 실행**
- **searchConditionMap()메서드는 다양한 검색 조건이 저장된 conditionMap을 리턴하는데 이 리턴 결과를 다음에 실행된 getBoardList() 메서드가 리턴한 JSP에서사용**

@ModelAttribute

BoardController

```
// 검색 조건 목록 설정
@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;
}

// 글 목록 검색
@RequestMapping("/getBoardList.do")
public String getBoardList(BoardVO vo, Model model) {
    // Null Check
    if (vo.getSearchCondition() == null)
        vo.setSearchCondition("TITLE");
    if (vo.getSearchKeyword() == null)
        vo.setSearchKeyword("");
    // Model 정보 저장
    model.addAttribute("boardList", boardService.getBoardList(vo));
    return "getBoardList.jsp"; // View 이름 리턴
}
```

/getBoardlist.do

1

2

3

4

5

model

"conditionMap"	**Map**
"boardList"	List

1.클라이언트가 "/getBoardList.do" 요청을 전송
2.**@ModelAttribute** 가 설정된 **serachConditionMap()**메서드가 먼저 실행
3.@ModelAttribute 로 지정한 이름으로 **searchConditionMap()**메서드가 리턴한 값을 **Model**객체에 저장
4.클라이언트가 호출한 **getBoardList()** 메서드가 실행
5.boardlist라는 이름으로 검색 결과를 **Model**에 저장하면 최종적으로 **Model**에는 두개의 컬렉션이 저장

@ModelAttribute

```
<!-- 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="<spring:message code="messag
            </td>
        </tr>
    </table>
</form>
<!-- 검색 종료 -->
```

- **searchConditionMap()** 메서드 위에 **ModelAttribute**가 선언되었으므로 **getBoardList()** 메서드가 실행되기 전에 먼저 실행
- searchConditionMap()메서드는 다양한 검색 조건이 저장된 **conditionMap**을 리턴하는데 이 리턴 결과를 다음에 실행된 **getBoardList()** 메서드가 리턴한 **JSP**에서 사용

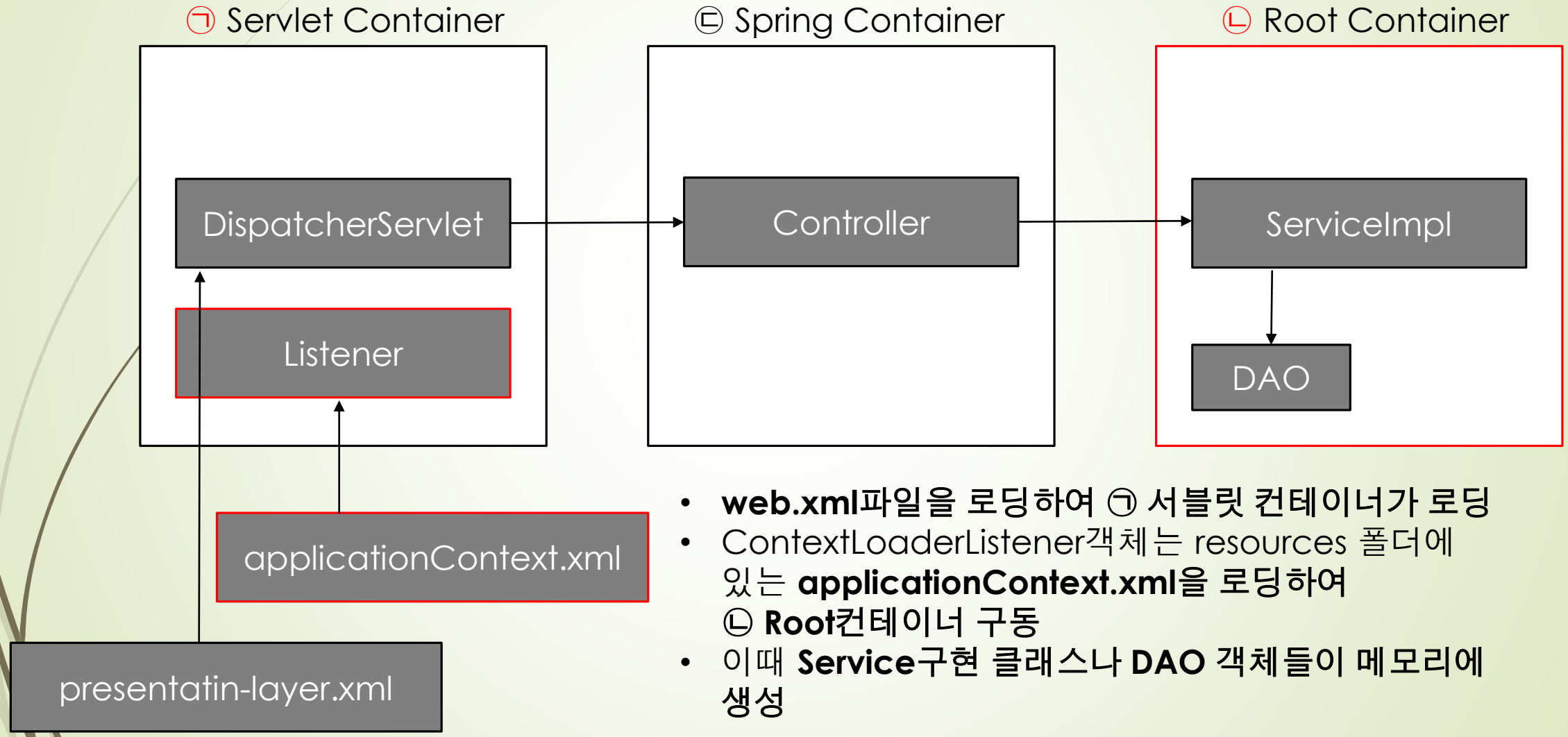
ContextLoaderListener

```
<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

```
INFO: Initializing Spring root WebApplicationContext
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization started
INFO : org.springframework.web.context.support.XmlWebApplicationContext - Refreshing Root WebApplicationCont
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from cla
INFO : org.springframework.context.support.PropertySourcesPlaceholderConfigurer - Loading properties file fr
INFO : org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor - JSR-330 'javax.in
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization completed
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/dataTransform.do]}" or
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/getBoardList.do]}" ont
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/insertBoard.do]}" onto
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/updateBoard.do]}" onto
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/deleteBoard.do]}" onto
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/getBoard.do]}" onto pu
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/login.do],methods=[POS
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/login.do],methods=[GET
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "{[/logout.do]}" onto publ
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Looking for @ControllerAdvice: We
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Looking for @ControllerAdvice: We
INFO : org.springframework.web.servlet.DispatcherServlet - FrameworkServlet 'action': initialization completed in 1075 ms
```

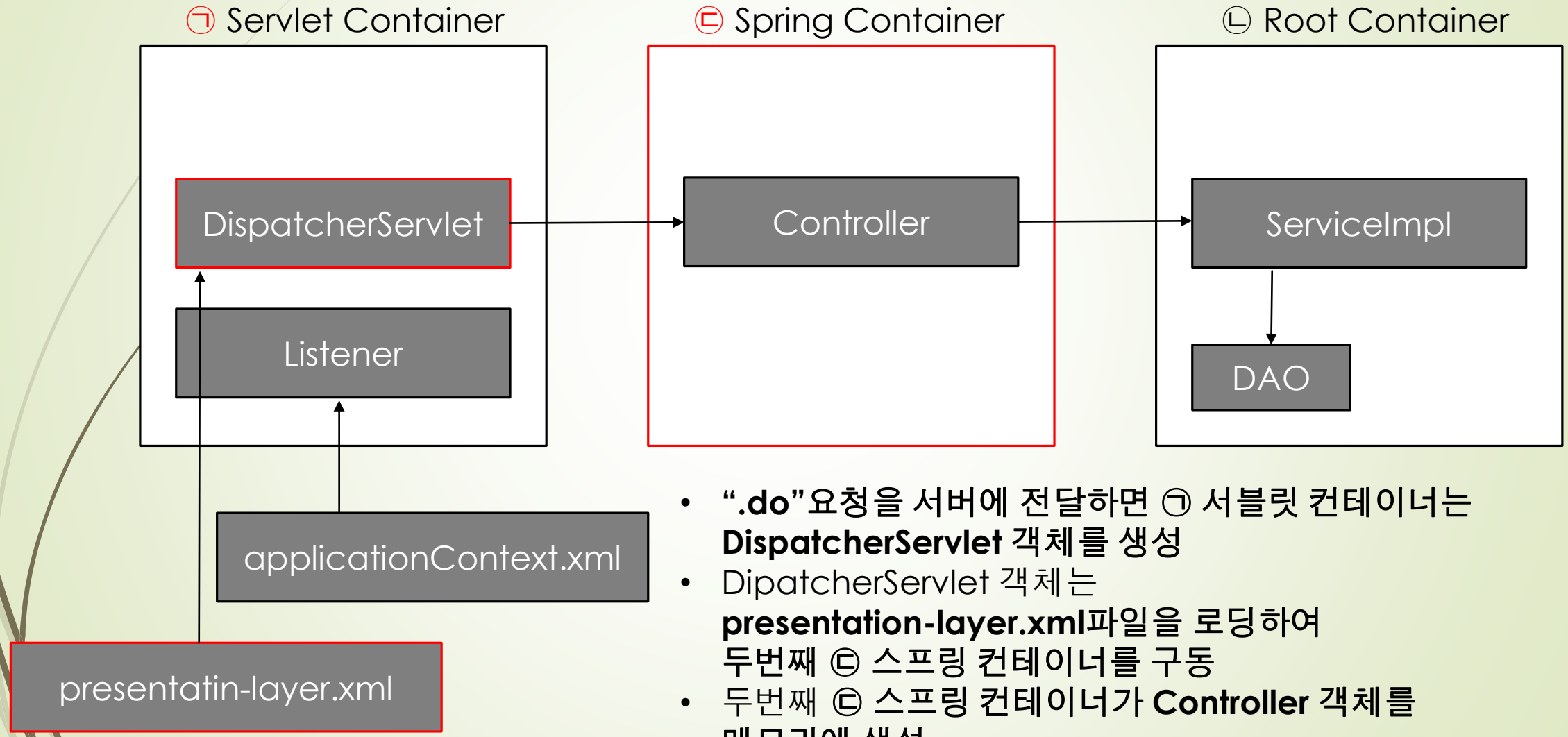
- **ContextLoaderListener**는 **Servlet**이나 **Filter** 클래스와 마찬가지로 **web.xml**파일에 등록
- **<listener-class>** 태그를 이용하여 스프링에서 제공하는 ContextLoaderListener 클래스를 등록
- 컨테이너가 구동될 때 **Pre-Loading**되는 객체

ContextLoaderListener



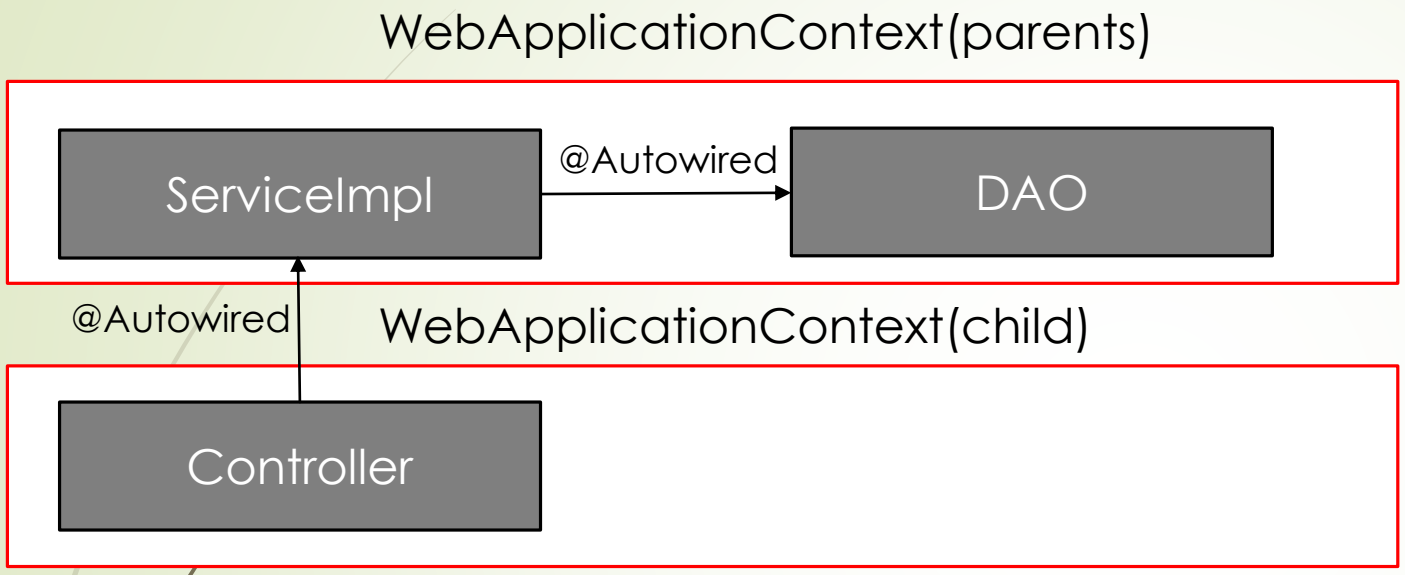
- **web.xml**파일을 로딩하여 ㉠ 서블릿 컨테이너가 로딩
- ContextLoaderListener객체는 resources 폴더에 있는 **applicationContext.xml**을 로딩하여 ㉢ **Root**컨테이너 구동
- 이때 **Service**구현 클래스나 **DAO** 객체들이 메모리에 생성

ContextLoaderListener



- **".do"요청을 서버에 전달하면 ㉠ 서블릿 컨테이너는 DispatcherServlet 객체를 생성**
- DipatcherServlet 객체는 **presentation-layer.xml파일을 로딩하여 두번째 ㉡ 스프링 컨테이너를 구동**
- 두번째 ㉡ 스프링 컨테이너가 **Controller 객체를 메모리에 생성**

DI - AutoWired



- **ContextLoaderlistener**와 **DispatcherServlet**이 각각 **XmlWebApplicationContext**를 생성
- 이때 두 스프링 컨테이너의 다른 역할과 기능
- **ContextloaderListener**가 **Root** 컨테이너를 생성하고 **DispatcherServlet**이 생성한 컨테이너는 **Root**컨테이너가 생성한 객체를 이용하는 자식 컨테이너를 생성
- 부모 컨테이너가 생성한 비지니스 객체를 자식 컨테이너가 생성한 **Controller**에서 참조하여 사용

JSON

BoardController.java

```java
@RequestMapping("/dataTransform.do")
@ResponseBody
public BoardListVO dataTransform(BoardVO vo) {
    vo.setSearchCondition("TITLE");
    vo.setSearchKeyword("");
    List<BoardVO> boardList = boardService.getBoardList(vo);
    BoardListVO boardListVO = new BoardListVO();
    boardListVO.setBoardList(boardList);
    return boardListVO;
}
```

pom.xml – Jackson

```xml
<!-- Jackson2 -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.7.2</version>
</dependency>
```

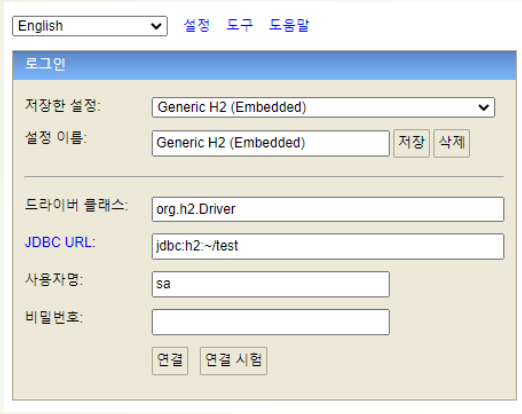
index.jsp

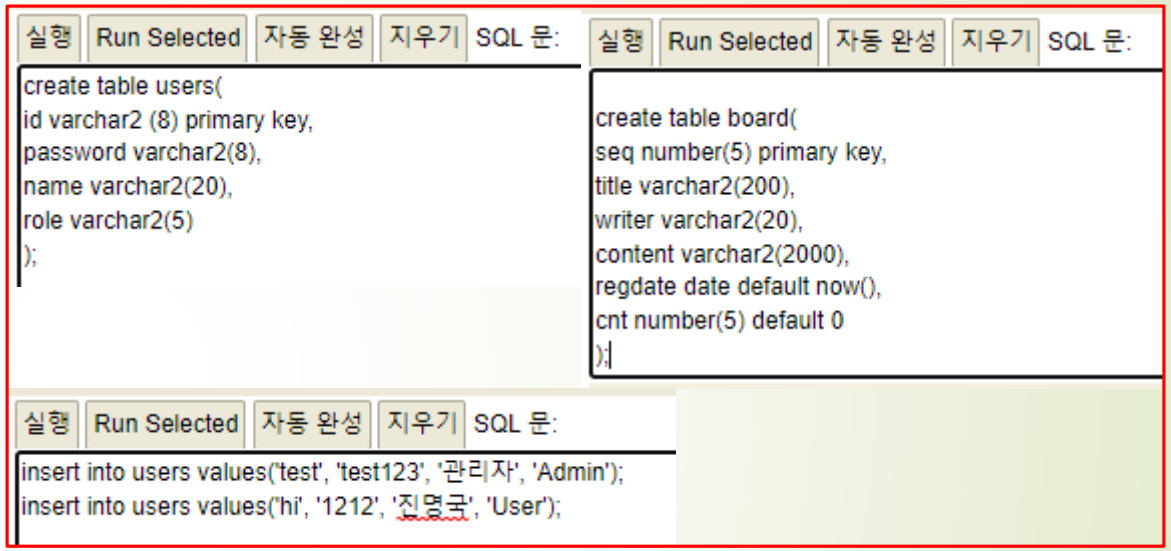
```
<a href="dataTransform.do">글 목록 변환 처리</a><br>
```


```
{"boardList":[{"seq":2,"title":"가을이 옵니다","writer":"진명국","content":"주황, 초록, 빨강 아름다운 가을이₩r₩n왔습니다.","regDate":"2022-11-09","cnt":0,"searchCondition":null,"searchKeyword":null,"uploadFile":null},{"seq":1,"title":"가입인사","writer":"관리자","content":"잘 부탁드립니다","regDate":"2022-11-03","cnt":0,"searchCondition":null,"searchKeyword":null,"uploadFile":null}]}
```


- 시스템이 복잡해지면서 다른 시스템과 정보를 주고받을 일이 발생
- 이때 교환 포맷으로 JSON을 사용
- **JSON은 BoardVO가 가진 각 변수와 변수에 저장된 값이 “키:값”으로 표현**

H2 DataBase





- **table users, board 생성**
- insert into users values('hi', '1212', '진명국', 'User');
- 새로운 사용자를 등록

H2 DataBase

%MAVEN_HOME%₩bin

```
C:\Users\jinmk>MVN
[INFO] Scanning for projects...
[INFO] ------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------
[INFO] Total time:  0.071 s
[INFO] Finished at: 2022-11-11T19:35:56+09:00
[INFO] ------------------------------------------
[ERROR] No goals have been specified for this bu
lugin-prefix>:<goal> or <plugin-group-id>:<plugi
: validate, initialize, generate-sources, proces
es, generate-test-sources, process-test-sources
test-classes, test, prepare-package, package, pr
tall, deploy, pre-clean, clean, post-clean, pre-
[ERROR]
[ERROR] To see the full stack trace of the erro
[ERROR] Re-run Maven using the -X switch to enal
[ERROR]
[ERROR] For more information about the errors a
[ERROR] [Help 1] http://cwiki.apache.org/conflu
```

	Version	Vulnerabilities	Repository
2.1.x	2.1.214		Central
	2.1.212		Central
	2.1.210		Central

```
<!-- H2 데이터베이스 -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.210</version>
</dependency>
```

h2-2.1.210.jar

database.properties

```
1 jdbc.driver=org.h2.Driver
2 jdbc.url=jdbc:h2:tcp://localhost/~/test
3 jdbc.username=sa
4 jdbc.password=
```

- 환경 변수 설정에 maven을 등록
- maven repository에서 **H2 DataBase** 버전에 맞는 **<dependency>**태그를 **pom.xml**에 등록
- **database.properties**를 만들고 **driver, url, username, password**를 등록

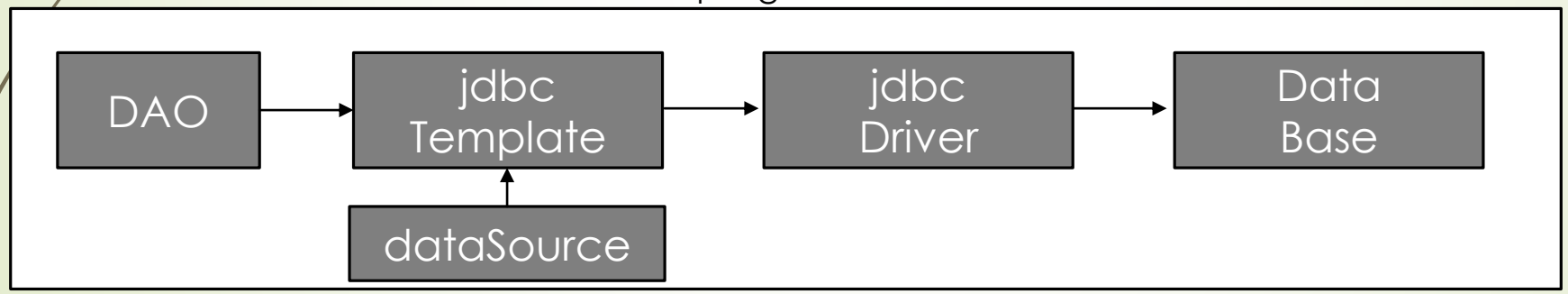
H2 DataBase - template


```
<!-- DataSource 설정 -->
<context:property-placeholder location="classpath:config/database.properties" />

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="${jdbc.driver}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>

<!-- Spring JDBC 설정 -->
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource" />
</bean>
```

Spring Jdbc



- **applicationContext.xml**에서 생성된 **RootContainer**를 실행할 때 위치를 **location="database.properties"**로 설정하고 **DAO**와 연결할 때 **<bean>** 의 **<property>**속성으로 정보를 불러들여 **H2**와 연동
- **Template** 메서드 패턴은 알고리즘을 캡슐화해서 재사용하는 패턴으로 반복되는 **DB** 연동 로직으로 **JdbcTemplate** 클래스가 제공하고 **Spring**이 실행하는 **DAO**와 같은 역할

mybatis – spring 연동


```
<!-- Mybatis -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis</artifactI
<version>3.3.1</version>
</dependency>
```

```
<!-- Mybatis Spring -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis-spring</artifactId>
<version>1.2.4</version>
</dependency>
```

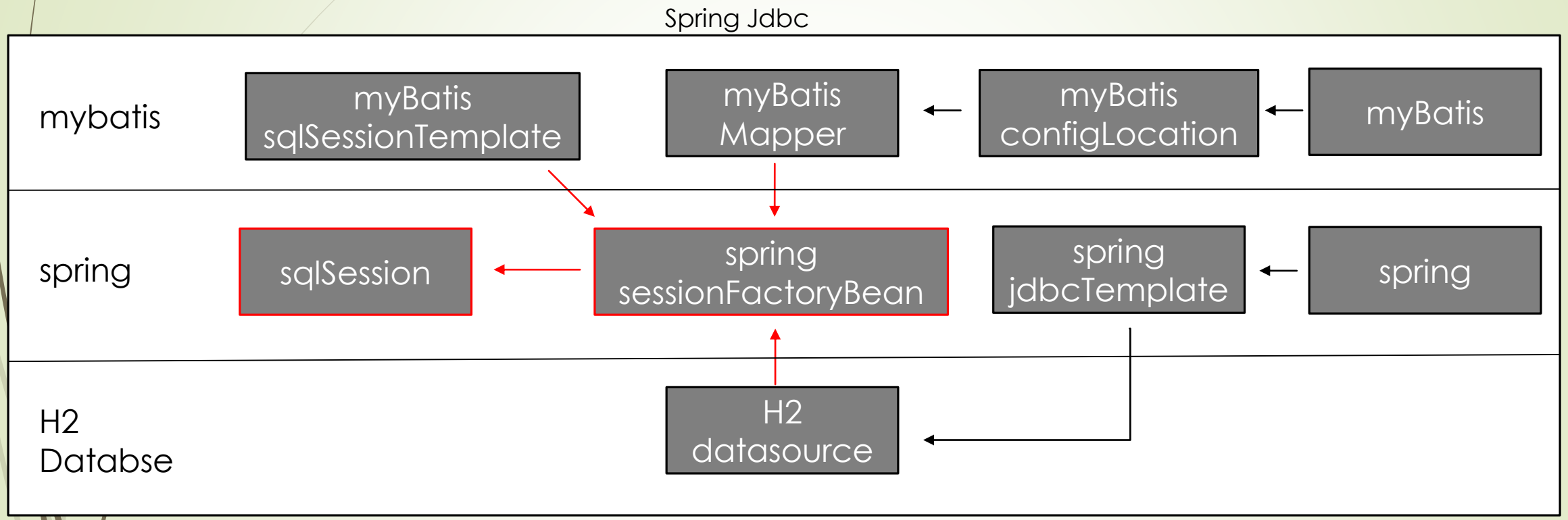
- mybatisFrameWork 는 원래 **Apache에서 Ibatis라는 이름의 프레임워크로 탄생**
- 2010년에 **Ibatis가 Apache에서 탈퇴하여** Google로 넘어가면서 이름이 **Mybatis로 변경**
- JDBC기반의 코드를 mybatis는 **XML파일에 저장된 SQL 명령어를 대신 실행하고 실행결과를 VO같은 자바 객체에 자동으로 매핑해주는 역할**
- 스프링에서는 Ibatis와 연동하기위해 API를 제공하지만 mybatis의 API는 미제공
- 하지만 **mybatis에서는 스프링 프레임워크와 연동에 필요한 API를 제공하여 mybatis에서 제공하는 클래스를 이용하여 연동 가능**

spring – sqlSession


```
<!-- Spring과 Mybatis 연동 설정 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="configLocation" value="classpath:sql-map-config.xml"/>
</bean>
<bean class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSession"></constructor-arg>
</bean>
```

- 스프링과 mybatis를 연동하려면 우선 **스프링 설정 파일에 SqlSessionFactoryBeen 클래스를 Bean등록**
- **SqlSessionFactoryBean 객체로부터 DB연동 구현에 사용할 SqlSession 객체를 생성**
- **SqlSessionFactoryBean 객체가 SqlSession 객체를 생성하려면 반드시 DataSource와 SQLMapper 정보가 필요**
- 앞에 등록된 DataSource를 Setter 인젝션으로 참조하고,
  SQL Mapper가 등록된 sql-map-config.xml파일도 Setter 인젠셕으로 설정
- **<bean> 등록된 SqlSessionFactoryBean이 SqlSession 객체를 생성**
- **configLocation은 Aliases설정과 해당 mapper.xml 경로를 잡아주는 설정**

spring– sqlSession



- **mybatis, spring, H2**를 연동하고 핵심인 **sqlSession** 생성에 필요한 요소들을 나름 정리해보았습니다.
- **sqlSessionTemplate**은 **SqlSession**이 현재의 스프링 트랜잭션에서 사용될 수 있도록 보장하고 필요한 시점에서 세션을 닫고, 커밋하거나 롤백하는 것을 포함한 세션의 생명주기를 관리
- **mybatisMapper.xml, H2 dataSource, myBatisSqlsessionTemplate**를 **sessionFactoryBean**클래스가 최종적으로 **sqlSession**을 생성

Mapper XML

board-mapping.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="BoardDAO">
```

mapper	<mapper namespace="BoardDAO"> <delete id="deleteBoard"> delete board where seq=#{seq}</></></>

DAO클래스	public void deleteBoard(BoardVO vo){
	mybatis.delete("BoardDAO.deleteBoard", vo)
	}

- SQL Mapper XML, 이하 **Mapper**는 **<mapper>**를 루트 엘리먼트로 가지는 **XML**파일
- **<mapper>**는 **namespace** 속성을 가짐
- **namespace**를 이용하여 더 쉽게 유일한 **SQL** 아이디를 생성
- 네임스페이스가 지정된 Mapper의 SQL을 **DAO**클래스에서 참조할 때 **namespace**와 **SQL**의 아이디를 결합하여 참조

typeAliaes

```
6     <!-- Alias 설정 -->
7⊖    <typeAliases>
8         <typeAlias alias="board" type="com.springbook.biz.board.BoardVO" />
9     </typeAliases>
```

```
<select id="getBoard" resultType="board">
    <![CDATA[
    SELECT *
    FROM BOARD
    WHERE SEQ = #{seq}
    ]]>
</select>
```

parameterType="com.springbook.biz.board.BoardVO" → parameterType="board"

- <typeAliases>엘리먼트는 <typeAlias>를 여러개 가질 수 있으며,
  특정 클래스의 별칭<Alias>을 선언
- Alias는 SQL명령어들이 저장되는 SqlMapper에서 사용 가능
- **Sql Mapping 파일의 크기를 줄여주기도하고 설정을 간단히 처리**

<select> 엘리먼트

```
<insert id="insertBoard" parameterType="board">
    <![CDATA[
    INSERT INTO BOARD(SEQ, TITLE, WRITER, CONTENT)
    VALUES((SELECT NVL(MAX(SEQ), 0) + 1 FROM BOARD),
    #{title}, #{writer}, #{content})
    ]]>
</insert>
```

```
<select id="getBoard" resultType="board">
    <![CDATA[
    SELECT *
    FROM BOARD
    WHERE SEQ = #{seq}
    ]]>
</select>
```

- **parameterType과 resultType 속성을 사용**
- parameterType은 mapper 파일에 등록된 SQL 실행에 필요한 데이터를 외부로부터 받음
- **일반적으로 기본형이나 VO형태의 클래스를 지정**
- **parameterType으로 지정된 클래스에는 사용자가 입력한 값들을 저장할 여러 변수가 있고, 변수들을 이용하여 SQL 구문에 사용자 입력값들을 설정하는데, 이때 #{변수명} 표현을 사용**

<select> 엘리먼트

```
<insert id="insertBoard" parameterType="board">
    <![CDATA[
    INSERT INTO BOARD(SEQ, TITLE, WRITER, CONTENT)
    VALUES((SELECT NVL(MAX(SEQ), 0) + 1 FROM BOARD),
    #{title}, #{writer}, #{content})
    ]]>
</insert>
```

```
<select id="getBoard" resultType="board">
    <![CDATA[
    SELECT *
    FROM BOARD
    WHERE SEQ = #{seq}
    ]]>
</select>
```

- <resultType> 속성은 검색 관련 SQL 구문이 실행되면 ResultSet이 리턴
- **Resultet에 저장된 검색 결과를 어떤 자바 객체에 매핑할지 지정해야하는데, 이때 사용 하는 것이 <resultType>**
- **<resultType>속성값으로 board를 사용했다면 SELECT 실행 결과를 BoardVO 객체에 매핑하여 리턴하라는 의미**
- resultType속성은 당연히 **쿼리 명령어가 등록되는 <select>엘리먼트에서만 사용할 수 있으여, 절대 생략할 수 없는 속성**
- **resultType대신 resultMap속성 사용 가능**

<resultMap>

```xml
<mapper namespace="BoardDAO">

    <resultMap id="boardResult" type="board">
        <id property="seq" column="SEQ" />
        <result property="title" column="TITLE" />
        <result property="writer" column="WRITER" />
        <result property="content" column="CONTENT" />
        <result property="regDate" column="REGDATE" />
        <result property="cnt" column="CNT" />
    </resultMap>
```

```xml
<select id="getBoardList" resultMap="boardResult">
    SELECT *
    FROM BOARD
    WHERE 1 = 1
    <if test="searchCondition == 'TITLE'">
        AND TITLE LIKE '%'||#{searchKeyword}||'%'
    </if>
    <if test="searchCondition == 'CONTENT'">
        AND CONTENT LIKE '%'||#{searchKeyword}||'%'
    </if>
    ORDER BY SEQ DESC
</select>
```

- 검색 결과를 특정 자바 객체에 매핑하여 리턴하기 위해서 parameterType 속성을 사용하지만 검색 **결과를 parameterType 속성으로 매핑할 수 없는 몇가지**가 있는데 예를들어, 검색 쿼리가 **단순 테이블 조회가 아닌 JOIN구문을 포함할 때는 검색 결과를 정확하게 하나의 자바 객체로 매핑 불가능**
- 검색된 테이블의 칼럼 이름과 매핑에 사용될 자바 객체의 변수의 이름이 다를 때 **검색 결과가 정확하게 자바 객체로 매핑되지 않기때문에 <resultMap>을 사용하여 처리**
- **이처럼 boardResult라는 아이디로 <resultMap>을 설정하고 설정은 PK에 해당하는 SEQ칼럼만 <id> 엘리먼트를 사용했고 나머지는 <result> 엘리먼트를 이용하여 검색 결과로 얻어낸 칼럼의 값과 BoardVO 객체의 변수를 매핑**
- 이렇게 설정된 resultMAp을 getBoardList로 등록된 쿼리에서 resultMap 속성으로 참조

<![CDATA[ ]]>

```
<select id="getBoard" resultType="board">
    <![CDATA[
    SELECT *
    FROM BOARD
    WHERE SEQ = #{seq}
    ]]>
</select>
```

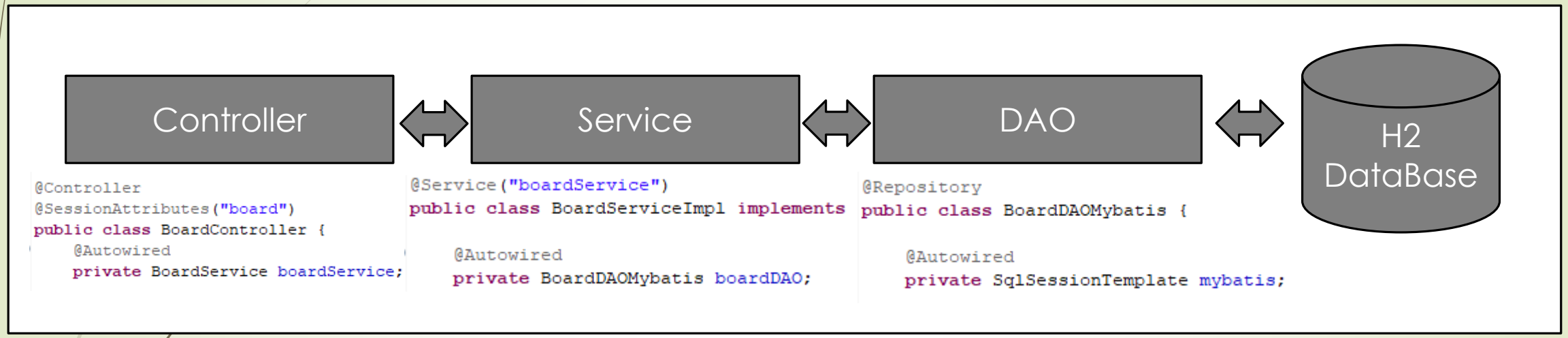
- SQL 구문내에 '<'기호를 사용하면 에러가 발생
- **이는 XML파서가 XML 파일을 처리할 때 '<' 기호를 '작다' 라는 의미의 연산자가 아닌 또 다른 태그의 시작으로 처리하기 때문**
- 하지만 <![CDATA[ ]]의 CDATA Section으로 SQL 구문을 감싸주어 SQL문을 정상 작동
- **CDATA안에 작성된 데이터는 XML파서가 처리하지 않고 데이터 베이스에 그대로 전달**

Dynamic SQL

```
<select id="getBoardList" resultMap="boardResult">
    SELECT *
    FROM BOARD
    WHERE 1 = 1
    <if test="searchCondition == 'TITLE'">
        AND TITLE LIKE '%'||#{searchKeyword}||'%'
    </if>
    <if test="searchCondition == 'CONTENT'">
        AND CONTENT LIKE '%'||#{searchKeyword}||'%'
    </if>
    ORDER BY SEQ DESC
</select>
```

- **mybatis**에서는 **Dynamic SQL**을 지원하여 조건에 따라 다양한 쿼리 데이터베이스에 전송
- 구문에서는 **if**라는 동적 요소를 사용하여 조건에 따라 분기를 처리

비지니스 로직



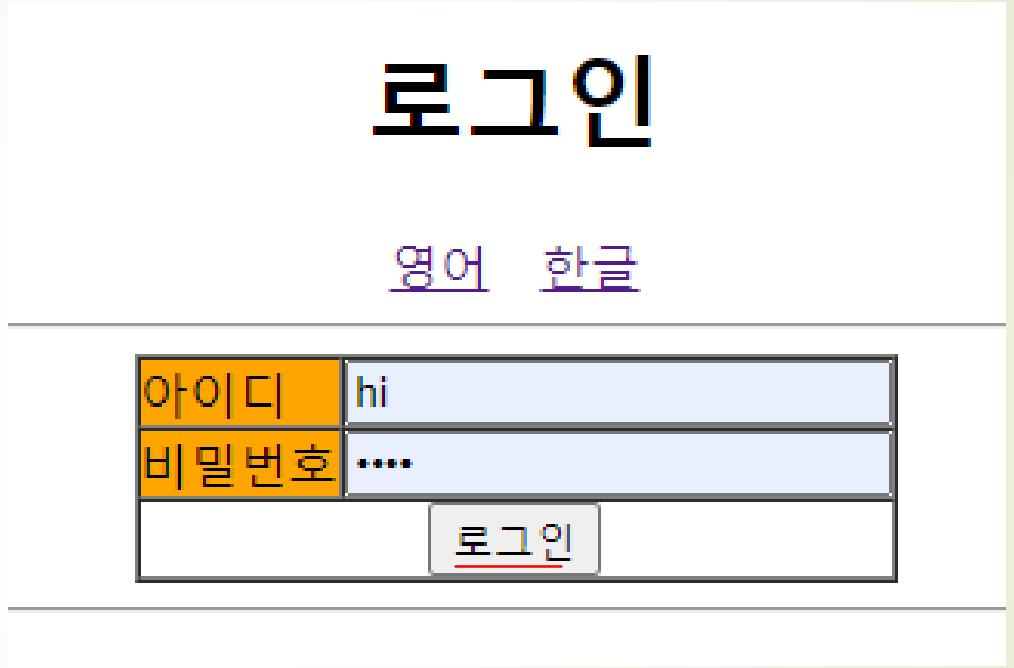
BoardController.java

```
// 글 목록 검색
@RequestMapping("/getBoardList.do")
public String getBoardList(BoardVO vo, Model model) {
    // Null Check
    if (vo.getSearchCondition() == null)
        vo.setSearchCondition("TITLE");
    if (vo.getSearchKeyword() == null)
        vo.setSearchKeyword("");
    // Model 정보 저장
    model.addAttribute("boardList", boardService.getBoardList(vo));
    return "getBoardList.jsp"; // View 이름 리턴
}
```

- **DispatcherServlet**객체가 요청 받은 **Controller**의 **/*.do**를 찾아 비지니스 로직에 의해 **Model**정보를 **DB**에서 쿼리문으로 검색
- **BoardVO**타입의 정보를 **Model**에 저장
- 요청된 값을 **JSP**로 전송

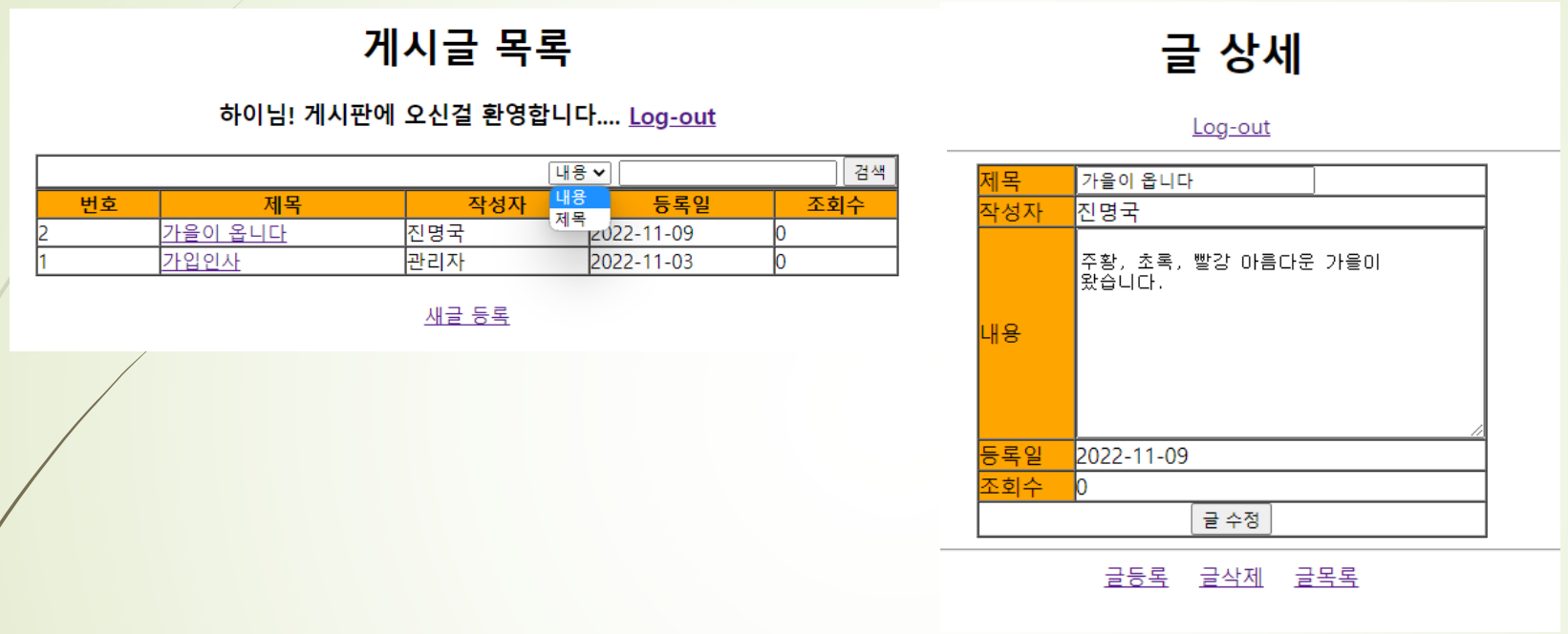
웹 게시판의 형태

http://localhost:8080/BoardWebFinal/



- 웹 게시판은 로그인, 글 목록 바로가기가 있고 로그인을 누르면 **login.jsp**를 보여주고 로그인 버튼을 누르면 **login()**함수를 호출
- **DB**의 정보를 세션에 머무르게 하여 해당 페이지로 이동

웹 게시판의 형태



- 새글 등록, 글 수정, 글 삭제, 글 목록 등을 선택하면 각자의 함수를 부르고 로직처리되어 해당 **Session**에 값을 담고 **JSP**페이지로 이동

- 검색 창에 제목 또는 내용을 선택하는 변수 **VO.searchKeyword**를 만들어 **'TITLE'**과 **'CONTENT'**를 선택하여 해당 글을 검색하는 기능 등이 있습니다.

---이상입니다---

마치며

지금까지 “**H2 DataBase, Spring을 이용한 Web 게시판**”을 봐주신 분들께 감사드리며

PPT를 마치겠습니다.

감사합니다.

진명국