Spring Boot Application

스프링 부트 응용

JSP를 이용한 UI 화면 개발
(기타 게시판 기능 개발)

한국기술교육대학교
온라인평생교육원

학습내용

- 글등록, 글상세 기능 구현
- 글수정, 글삭제 기능 구현

학습목표

- JSP를 이용하여 게시글 상세, 등록 기능을 구현할 수 있다.
- JSP를 이용하여 게시글 수정, 삭제 기능을 구현할 수 있다.

글등록, 글상세 기능 구현

1 글등록 기능 구현

⟨1⟩ 등록 화면 이동

- WEB-INF 폴더는 브라우저에서 직접 요청할 수 없기 때문에 ViewResolver를 이용해야 함

1 글등록 기능 구현

⟨1⟩ 등록 화면 이동

- 사용자가 글목록 화면(getBoardList.jsp)에서 [새글 등록] 링크를 클릭했을 때
- 글등록 화면으로 이동하기 위해서는 화면 이동을 처리하는 메소드를 추가해야 함

글등록, 글상세 기능 구현

1 글등록 기능 구현

⟨1⟩ 등록 화면 이동

- 링크를 통해 전달된 요청은 GET 방식이므로 @GetMapping을 사용하여 매핑함

```
@GetMapping("/board/insertBoard")
public String insertBoardView() {
    return "insertBoard";
}
```

1 글등록 기능 구현

⟨2⟩ 글등록 JSP 작성

- 사용자에게 글등록 화면을 제공하기 위해 WEB-INF/board 폴더에 insertBoard.jsp 파일을 작성함

글등록, 글상세 기능 구현

1 글등록 기능 구현

〈2〉 글등록 JSP 작성

- HTML의 〈form〉 태그는 GET과 POST 방식만 지원함
- 사용자 입력 값을 노출하지 않기 위해 method는 POST로 설정함

1 글등록 기능 구현

〈2〉 글등록 JSP 작성

```
<form action="insertBoard" method="post">
<table border="1" cellpadding="0" cellspacing="0">
<tr>
<td bgcolor="orange" width="70">제목</td><td align="left">
<input type="text" name="title"/></td>
</tr>
<tr>
<td bgcolor="orange">작성자</td><td align="left">
<input type="text" name="writer" size="10"/></td>
</tr>
<tr>
<td bgcolor="orange">내용</td><td align="left">
<textarea name="content" cols="40" rows="10"></textarea></td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value=" 새글 등록 "/></td>
</tr>
</table>
</form>
```

글등록, 글상세 기능 구현

1 글등록 기능 구현

⟨3⟩ 비즈니스 컴포넌트 수정

1. 비즈니스 클래스(BoardService)에 글등록 기능의 비즈니스 메소드를 추가함
2. CrudRepository가 제공하는 save 메소드와 매개변수로 받은 Board 엔티티를 이용하여 INSERT를 처리함

```
public void insertBoard(Board board) {
    boardRepositoy.save(board);
}
```

1 글등록 기능 구현

⟨4⟩ 컨트롤러 수정

- 비즈니스 메소드를 호출하는 컨트롤러(BoardController)에 글등록 기능의 insertBoard 메소드를 작성함

글등록, 글상세 기능 구현

1 글등록 기능 구현

④ 컨트롤러 수정

- insertBoard 메소드 위에는 @PostMapping을 설정하여 POST 요청을 매핑함

```
@PostMapping("/board/insertBoard")
public String insertBoard(Board board) {
    boardService.insertBoard(board);
    return "redirect:getBoardList";
}
```

1 글등록 기능 구현

④ 컨트롤러 수정

- 화면을 이동할 때는 Forward 방식과 Redirect 방식을 사용함
- 스프링은 Forward 방식을 기본으로 사용함

글등록, 글상세 기능 구현

1 글등록 기능 구현

⑤ 화면 이동 방식

1 Forward 방식

- Forward 방식은 한 번의 HTTP 요청과 응답으로 사용자의 요청이 처리됨

1 글등록 기능 구현

⑤ 화면 이동 방식

1 Forward 방식

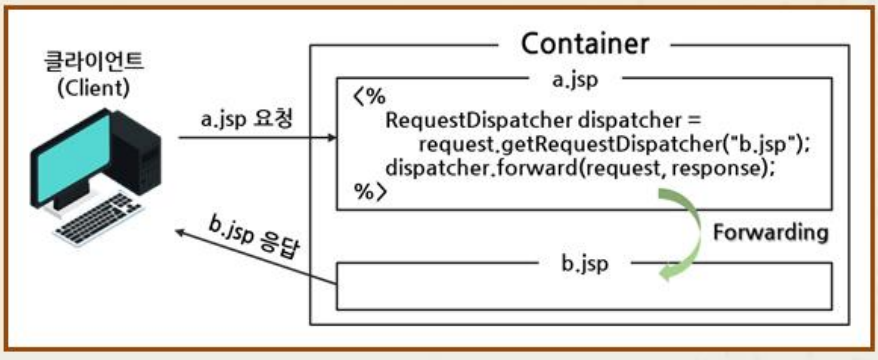
- Forward 방식은 요청 처리 속도가 빠름
- 브라우저의 URL이 요청 시의 URL로 고정되기 때문에 어느 페이지에서 응답이 들어왔는지 확인할 수 없음

글등록, 글상세 기능 구현

1 글등록 기능 구현

⑤ 화면 이동 방식

1 Forward 방식



1 글등록 기능 구현

⑤ 화면 이동 방식

2 Redirect 방식

- 일단 브라우저로 응답을 보냈다가 특정 페이지로 다시 요청하는 방식

글등록, 글상세 기능 구현

1 글등록 기능 구현

⑤ 화면 이동 방식

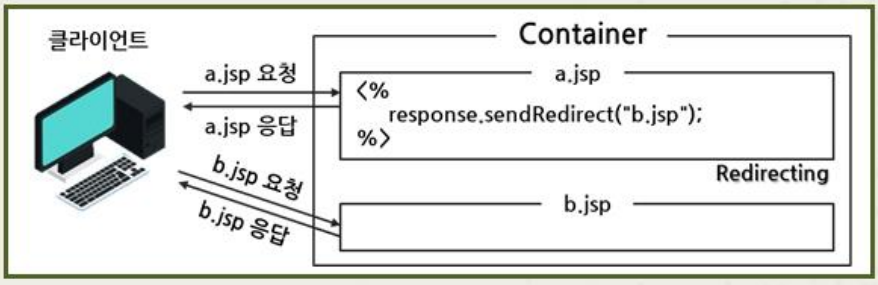
2 Redirect 방식

- Forward 방식에 비해 요청 처리 속도는 느림
- 브라우저의 URL의 변경되기 때문에
  어느 페이지에서 응답이 들어왔는지 확인할 수 있음

1 글등록 기능 구현

⑤ 화면 이동 방식

2 Redirect 방식



글등록, 글상세 기능 구현

2 글상세 기능 구현

① 글상세 링크 추가

- 글목록 화면 제목에 하이퍼 링크를 추가하여 사용자가 클릭한 게시글의 상세 정보를 조회하여 출력함

```
<c:forEach var="board" items="${boardList }">
<tr>
<td>${board.seq }</td>
<td align="left"><a href="getBoard?seq=${board.seq }">${board.title }</a></td>
```

2 글상세 기능 구현

① 글상세 링크 추가

- 사용자가 상세 화면(getBoard)을 요청할 때, 반드시 사용자가 클릭한 게시글의 일련번호(seq)를 파라미터로 전달해야 함

글등록, 글상세 기능 구현

2 글상세 기능 구현

〈1〉 글상세 링크 추가

중요한 것

- 파라미터 정보를 전달할 때 물음표(?) 앞/뒤에는 공백이 있을 수 없음

2 글상세 기능 구현

〈2〉 비즈니스 컴포넌트 수정

- 비즈니스 클래스(BoardService)에 글상세 조회 기능의 비즈니스 메소드(getBoard)를 추가함

글등록, 글상세 기능 구현

2 글상세 기능 구현

〈2〉 비즈니스 컴포넌트 수정

- CrudRepository가 제공하는 findById 메소드를 이용하여 매개변수로 받은 Board 엔티티에서 seq 값을 이용하여 상세조회를 처리함

```
public Board getBoard(Board board) {
    return boardRepositoy.findById(board.getSeq()).get();
}
```

2 글상세 기능 구현

〈3〉 컨트롤러 수정

- 비즈니스 메소드를 호출하는 컨트롤러(BoardController)에 글상세조회 기능의 getBoard 메소드를 구현함

글등록, 글상세 기능 구현

2 글상세 기능 구현

〈3〉 컨트롤러 수정

- 하이퍼링크를 클릭하여 요청했으므로 @GetMapping을 이용하여 GET 방식의 요청을 매핑

```
@GetMapping("/board/getBoard")
public String getBoard(Board board, Model model) {
    model.addAttribute("board", boardService.getBoard(board));
    return "getBoard";
}
```

2 글상세 기능 구현

〈4〉 글상세 JSP 작성

- WEB-INF/board 폴더에 getBoard.jsp 파일을 작성함 (EL을 적용하여 Java 코드를 제거함)

글등록, 글상세 기능 구현

2 글상세 기능 구현

④ 글상세 JSP 작성

```
<table border="1" cellpadding="0" cellspacing="0">
<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" rows="10">${board.content }</textarea></td>
</tr>
<tr>
<td bgcolor="orange">등록일</td>
<td align="left"><fmt:formatDate value="${board.createDate }" pattern="yyyy-MM-dd">
</fmt:formatDate></td>
</tr>
<tr>
<td bgcolor="orange">조회수</td>
<td align="left">${board.cnt }</td>
</tr>
</table>
```

글수정, 글삭제 기능 구현

1 글수정 기능 구현

⟨1⟩ 수정 관련 UI 추가

- 글상세 화면(getBoard.jsp)에 수정을 위한 ⟨form⟩ 태그와 수정 버튼을 추가함

```
<form action="updateBoard" method="post">
<input name="seq" type="hidden" value="${board.seq }"/>
<table border="1" cellpadding="0" cellspacing="0">
<tr>
<td bgcolor="orange" width="70">제목</td>
<td align="left"><input name="title" type="text" value="${board.title }"/></td>
</tr>
    ~ 생략 ~
<tr>
<td bgcolor="orange">조회수</td>
<td align="left">${board.cnt }</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="글 수정"/>
</td>
</tr>
</table>
</form>
```

글수정을 처리하기 위해서는 seq 정보도 파라미터로 전달해야 함

1 글수정 기능 구현

⟨2⟩ 비즈니스 컴포넌트 수정

- 비즈니스 클래스(BoardService)에 글수정 기능의 비즈니스 메소드(updateBoard)를 추가함

글수정, 글삭제 기능 구현

1 글수정 기능 구현

〈2〉 비즈니스 컴포넌트 수정

- CrudRepository의 save 메소드는 등록 기능 뿐만 아니라 수정 기능을 처리할 때도 사용함

1 글수정 기능 구현

〈2〉 비즈니스 컴포넌트 수정

중요한 것

- 수정할 때 수정할 엔티티를 영속 컨텍스트 1차 캐시에 등록하기 위해 상세조회를 먼저 처리해야 한다는 것

```
public void updateBoard(Board board) {
    Board findBoard = boardRepositoy.findById(board.getSeq()).get();
    findBoard.setTitle(board.getTitle());
    findBoard.setContent(board.getContent());
    boardRepositoy.save(findBoard);
}
```

글수정, 글삭제 기능 구현

1 글수정 기능 구현

③ 컨트롤러 수정

- 비즈니스 메소드를 호출하는 컨트롤러(BoardController)에 글수정 기능의 updateBoard 메소드를 구현함

```
@PostMapping("/board/updateBoard")
public String updateBoard(Board board) {
    boardService.updateBoard(board);
    return "forward:getBoardList";
}
```

2 글삭제 기능 구현

① 삭제 관련 UI 추가

- 글상세 화면(getBoard.jsp)에 글등록, 글삭제, 글목록 링크를 추가함

글수정, 글삭제 기능 구현

2 글삭제 기능 구현

〈1〉 삭제 관련 UI 추가

- 글삭제 링크에는 삭제할 게시글의 일련번호(seq) 정보를 파라미터로 전달함

```
<hr>
<a href="insertBoardView">글등록</a>   
<a href="deleteBoard?seq=${board.seq }">글삭제</a>   
<a href="getBoardList">글목록</a>
</center>
</body>
</html>
```

2 글삭제 기능 구현

〈2〉 비즈니스 컴포넌트 수정

- 비즈니스 클래스(BoardService)에 글삭제 기능의 비즈니스 메소드(deleteBoard)를 추가함

글수정, 글삭제 기능 구현

2 글삭제 기능 구현

⟨2⟩ 비즈니스 컴포넌트 수정

- CrudRepository가 제공하는 deleteById 메소드를 이용하여 매개변수로 받은 Board 엔티티를 영속 컨텍스트에서 삭제함

```
public void deleteBoard(Board board) {
    boardRepositoy.deleteById(board.getSeq());
}
```

2 글삭제 기능 구현

⟨3⟩ 컨트롤러 수정

- 비즈니스 메소드를 호출하는 컨트롤러(BoardController)에 글삭제 기능의 deleteBoard 메소드를 구현함

```
@RequestMapping("/board/deleteBoard")
public String deleteBoard(Board board) {
    boardService.deleteBoard(board);
    return "forward:getBoardList";
}
```

글수정, 글삭제 기능 구현

3 다국어 처리

〈1〉 다국어(i18n)란?

- 국제화라고도 하며, 하나의 웹 화면을 다양한 언어로 서비스하는 것을 의미함
- 스프링 부트는 기존에 스프링에서 제공하는 다국어 기능을 좀 더 쉽고 편하게 사용할 수 있도록 지원함

3 다국어 처리

〈2〉 라이브러리 의존성 추가

- 스프링은 기본적으로 프로퍼티(.properties) 파일을 이용하여 언어별 메시지를 관리함
- 프로퍼티 설정은 프로퍼티 경로가 중복되기 때문에 불편함

글수정, 글삭제 기능 구현

3 다국어 처리

〈2〉 라이브러리 의존성 추가

- 메시지를 YAML 파일로 관리하기 위한 라이브러리 의존성을 추가함

```
<dependency>
    <groupId>net.rakugakibox.util</groupId>
    <artifactId>yaml-resource-bundle</artifactId>
    <version>1.2</version>
</dependency>
```

3 다국어 처리

〈3〉 메시지 파일 작성

- JSP 화면에서 사용할 메시지를 YAML 파일로 작성함
- src/main/resources 소스 폴더에 i18n 폴더를 생성하고 지원하려는 언어별로 메시지 파일을 작성함

글수정, 글삭제 기능 구현

3 다국어 처리

〈3〉 메시지 파일 작성

messageSource.yml

```
board:
  insertBoard:
    mainTitle: 새글 등록
    form:
      title: 제목
      writer: 작성자
      content: 내용
      insertBtn: 새글 등록
    locale:
      ko: 한글
      en: 영어
```

messageSource_en.yml

```
board:
  insertBoard:
    mainTitle: New Board
    form:
      title: TITLE
      writer: WRITER
      content: CONTENT
      insertBtn: Insert Board
    locale:
      ko: KOREAN
      en: ENGLISH
```

3 다국어 처리

〈4〉 MessageSource 클래스

- YAML 파일에 등록한 메시지를 애플리케이션에서 사용하려면 메시지 파일들을 읽어서 메모리에 로딩해야 함

글수정, 글삭제 기능 구현

3 다국어 처리

⟨4⟩ MessageSource 클래스

- MessageSource는 메시지 파일을 읽어서 해당 메시지들을 메모리에 로딩하는 객체

3 다국어 처리

⟨4⟩ MessageSource 클래스

- ResourceBundleMessageSource를 상속하여 YAML 파일 형식의 메시지를 로딩하는 MessageSource 클래스를 작성함

```
package com.mycompany.i18n.config;

import java.util.Locale;

class YamlMessageSource extends ResourceBundleMessageSource {
    @Override
    protected ResourceBundle doGetBundle(String basename, Locale locale) throws MissingResourceException {
        return ResourceBundle.getBundle(basename, locale, YamlResourceBundle.Control.INSTANCE);
    }
}
```

글수정, 글삭제 기능 구현

3 다국어 처리

⑤ 다국어 설정 클래스

스프링이 지원하는 항목

- AcceptHeaderLocaleResolver(Default)
- CookieLocaleResolver(Cookie에 저장)
- SessionLocaleResolver(HttpSession에 저장)
- FixedLocaleResolver(로케일 고정)

3 다국어 처리

⑤ 다국어 설정 클래스

- 다국어를 이용하려면 MessageSource와 LocaleResolver 객체를 생성해야 함

글수정, 글삭제 기능 구현

3 다국어 처리

⑤ 다국어 설정 클래스

```
package com.mycompany.i18n.config;

import java.util.Locale;

@Configuration
public class I18nCconfig implements WebMvcConfigurer {
    @Bean
    public MessageSource messageSource() {
        YamlMessageSource messageSource = new YamlMessageSource();
        messageSource.setBasename("i18n/messageSource");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
    }

    @Bean
    public LocaleResolver localeResolver() {
        SessionLocaleResolver localeResolver = new SessionLocaleResolver();
        localeResolver.setDefaultLocale(Locale.KOREA);
        return localeResolver;
    }
}
```

3 다국어 처리

⑥ 메시지 적용

- 스프링에서 제공하는 태그 라이브러리를 이용하여 JSP 파일에 원하는 메시지를 출력함

글수정, 글삭제 기능 구현

3 다국어 처리

⑥ 메시지 적용

- 메시지를 사용하기 위해서 사용하는 태그는 <spring:message>
- code 속성에 YAML 파일에 설정한 메시지 키를 지정함

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<!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>새글등록</title>
</head>
<body>
<center>
<h3><spring:message code="board.insertBoard.mainTitle"/></h3>
<a href="insertBoard?lang=ko"><spring:message code="board.insertBoard.locale.ko"/></a> 
<a href="insertBoard?lang=en"><spring:message code="board.insertBoard.locale.en"/></a>
```

3 다국어 처리

⑦ 로케일 변경

- 인터셉터는 컨트롤러로 전달되는 HTTP 요청과 응답을 가로채는 역할을 제공함
- 일반적으로 인증을 처리하거나 세션을 관리할 때 사용함

글수정, 글삭제 기능 구현

3 다국어 처리

⑦ 로케일 변경

- 사용자의 로케일을 변경하기 위해 환경 설정 클래스를 통해 LocaleChangeInterceptor를 생성하여 등록함

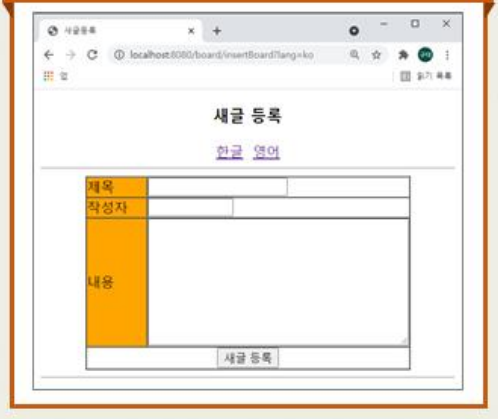
```
@Bean
public LocaleChangeInterceptor localeChangeInterceptor() {
    LocaleChangeInterceptor localeChangeInterceptor = new LocaleChangeInterceptor();
    localeChangeInterceptor.setParamName("lang");
    return localeChangeInterceptor;
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(localeChangeInterceptor());
}
```

3 다국어 처리

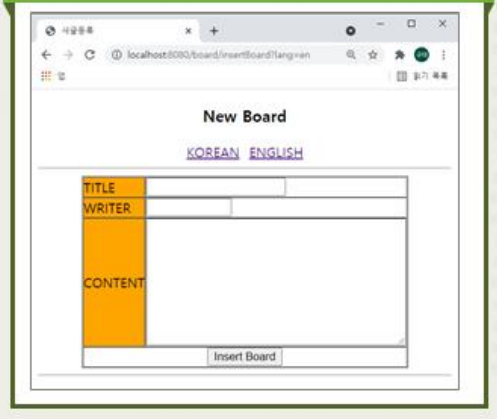
⑦ 로케일 변경

- 실행 결과

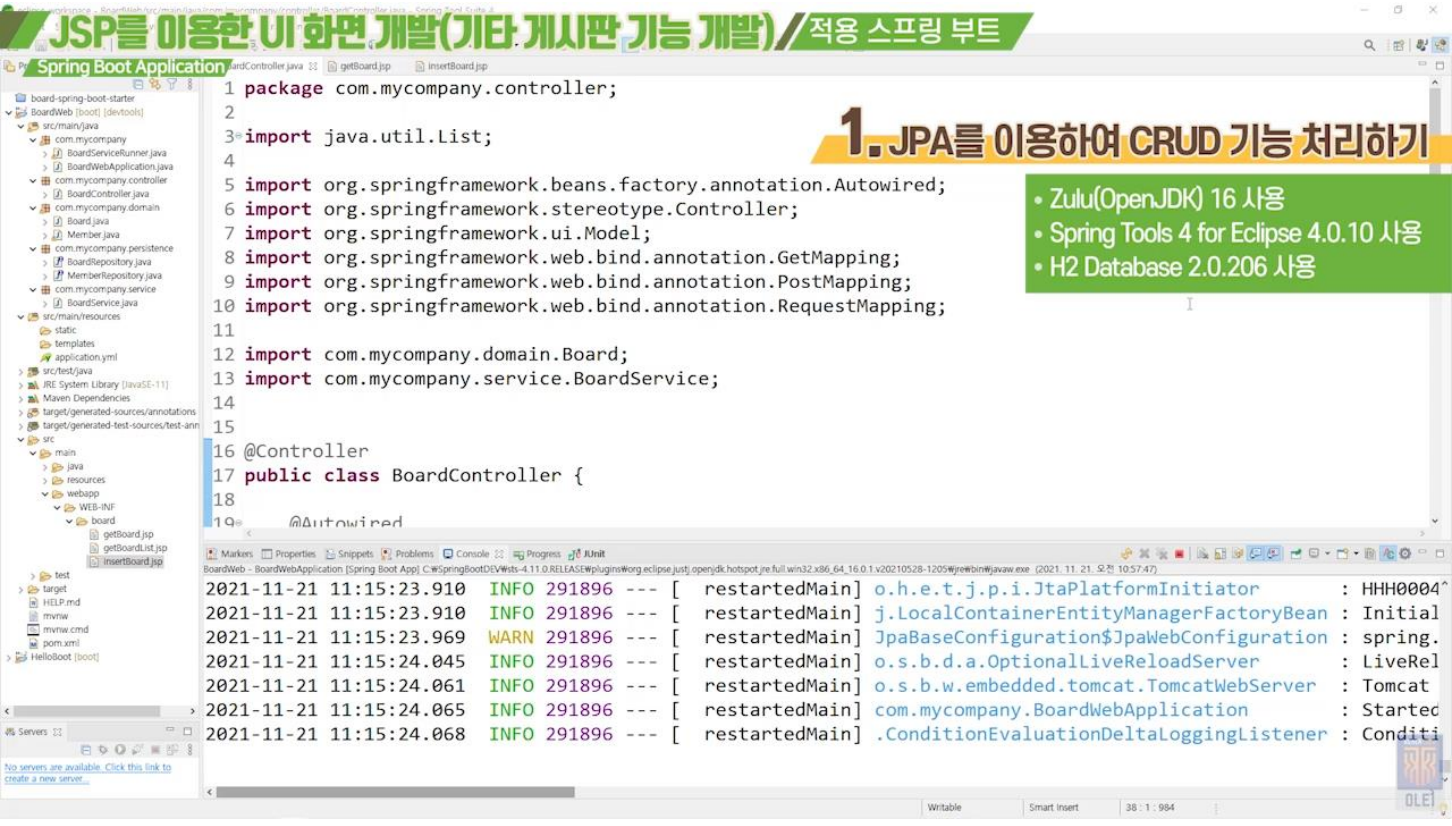
lang=en



lang=ko



적용 스프링 부트



실습단계
게시판 프로그램의 상세보기, 수정, 등록, 삭제 기능
가장 먼저 BoardController 클래스 수정
글등록 JSP 화면으로 이동
똑같은 방식으로 insertBoard 요청이 들어오지만, get 방식의 경우 화면 전환
Post 방식으로 요청이 들어왔을 때 실제 insert 비즈니스 로직 동작
글목록 화면으로 이동
redirect : 두번의 요청과 응답으로 속도는 느리나, URL 주소가 바뀜
forward : 빨리 실행되지만 URL이 최초 요청으로 고정됨
상세조회
상세화면 이동을 위해 getBoardList.jsp 파일 확인
getBoard.jsp 작성
Tip! 화면을 잠깐 멈추고 소스를 타이핑하며 진행할 것
날짜 포맷 출력을 위해 taglib 지시자 추가

적용 스프링 부트

실습단계
상세화면이 수정을 위한 화면이 되기도 함
수정을 위해 seq 파라미터 필요
글등록, 글삭제, 글목록 메뉴 확인 가능
상세화면과 거의 비슷하나 등록화면만 제공함
등록 JSP는 상세 JSP를 커스터마이징 해서 작성
기존 테이블에 데이터가 있을 경우 모두 삭제
localhost:8080/getBoardList 요청