

# Spring Boot 게시판

# 1. Spring Boot 게시판 CRUD 구현

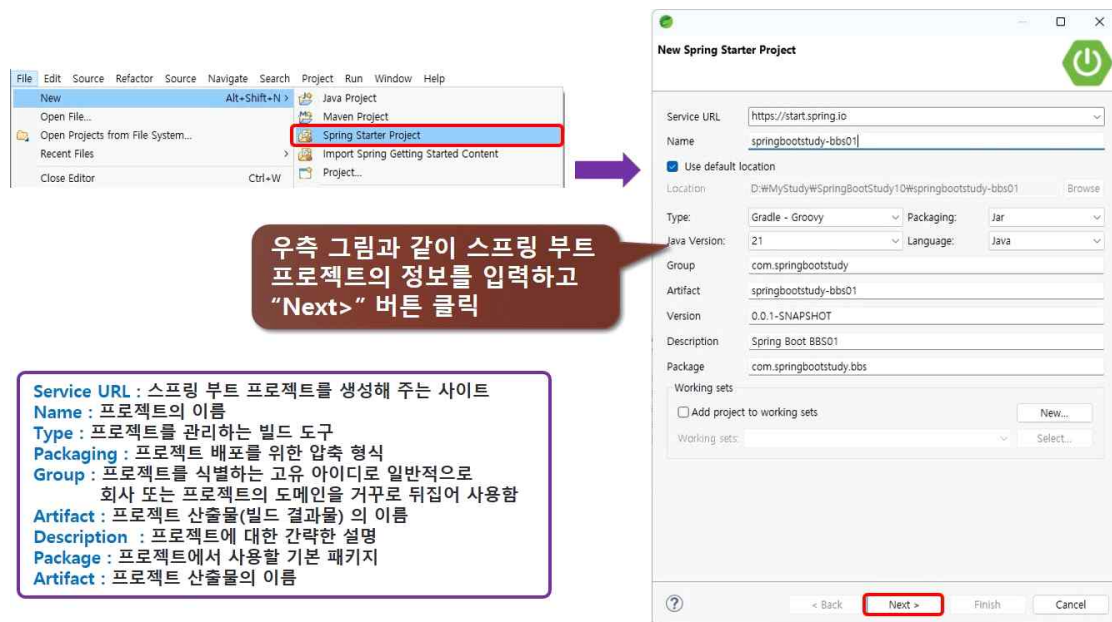
앞에서 스프링 부트를 이용해 웹 애플리케이션을 만드는 기본적인 방법에 대해서 알아보았다. 이번에는 스프링 부트에서 MyBatis를 활용해 게시판을 구현하는 방법에 대해서 알아보자.

## 1) 프로젝트 생성 및 환경설정

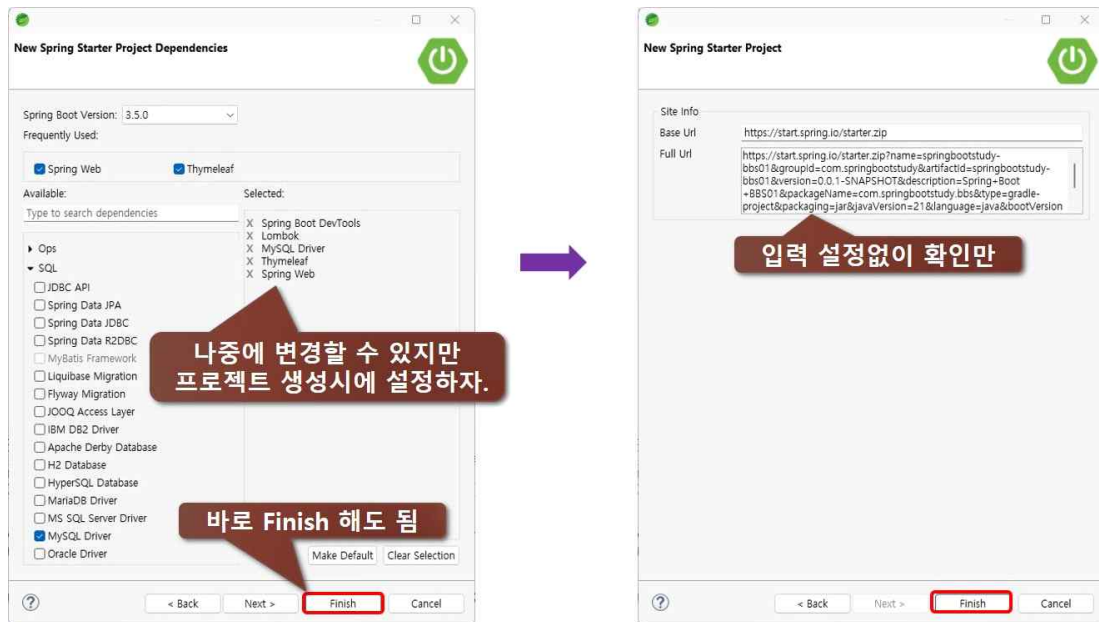
### 1-1) 프로젝트 생성

- 실습용 프로젝트 : springbootclass-bbs01
- 완성 프로젝트 : springbootstudy-bbs01

먼저 STS에서 다음 그림과 같이 Spring Starter Project 메뉴를 선택하고 스프링 부트 프로젝트를 만들자. 프로젝트의 이름과 프로젝트 관련 정보를 아래 그림과 같이 입력하자.



프로젝트의 의존성을 설정하는 대화상자에서 아래 그림에서 좌측 그림과 같이 설정하고 다음으로 이동하자.



## 1-2) 의존 라이브러리 설정

일반적으로 Spring Starter Project 메뉴를 사용해 프로젝트에 필요한 기본적인 의존성을 선택해 프로젝트를 생성했다면 build.gradle파일에 추가적으로 설정할 필요는 없다. 그런데 이번에 프로젝트를 생성할 때 무슨 이유인지는 모르겠지만 아래 그림과 같이 mybatis에 대한 의존 설정을 선택할 수 없었기 때문에 build.gradle에 mybatis에 대한 의존 설정을 아래와 같이 추가하고 “Refresh Gradle Project” 메뉴로 Gradle 설정을 적용하자.

### build.gradle

... 중 략 ...

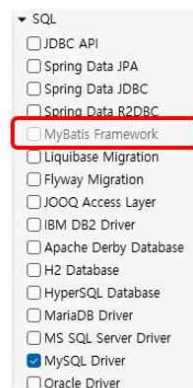
```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'

    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.mysql:mysql-connector-j'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'

    implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.4'
    testImplementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter-test:3.0.4'
}
```

... 중 략 ...



## 1-3) 프로젝트 환경 설정

프로젝트 환경 설정 파일인 application.properties 파일에 데이터베이스 등의 설정을 다음과 같이

추가하자.

- **src/main/resources/application.properties**

spring.application.name=springbootstudy-bbs01

#Database 접속 설정

spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/springboot?useSSL=false&useUnicode=true&serverTimezone=Asia/Seoul&allowPublicKeyRetrieval=true

spring.datasource.username=root

spring.datasource.password=12345678

#MyBatis

mybatis.type-aliases-package=com.springbootstudy.bbs.domain

mybatis.mapper-locations=classpath:mappers/\*\*/\*.xml

mybatis.configuration.map-underscore-to-camel-case=true

#정적 파일 변경 실시간 반영

spring.devtools.remote.restart.enabled=false

spring.devtools.livereload.enabled=true

#Thymeleaf 뷰 설정 - prefix와 suffix는 아래 경로가 기본 값임

#spring.thymeleaf.prefix=classpath:/templates/

#spring.thymeleaf.suffix=.html

#spring.thymeleaf.view-names=views/\*

# thymeleaf에 대한 캐시를 남기지 않음, cache=false 설정(운영시는 true)

spring.thymeleaf.cache=false

#Thymeleaf 사용 활성화

#spring.thymeleaf.enabled=true

#렌더링 전에 템플릿 존재여부 확인 옵션

#spring.thymeleaf.check-template=false

#template 위치 존재여부 확인 옵션 - 없으면 오류 발생

#spring.thymeleaf.check-template-location=true

## 1-5) 메시지 자원 등록

게시판에서 사용하는 각 화면의 타이틀을 스프링 메시지 자원으로 등록하여 사용할 것이다.

프로젝트의 src/main/resources/ 폴더에 messages.properties 파일을 새로 만들고 다음과 같이 타이틀을 작성하자.

- **src/main/resources/messages.properties**

# 게시판 타이틀을 스프링 메시지 자원으로 등록

# 게시글 리스트 타이틀

bbs.list.title=게시글 리스트

```
# 게시물 상세보기 타이틀
bbs.detail.title=게시글 상세보기
# 게시물 쓰기 폼 타이틀
bbs.write.title=게시글 쓰기
# 게시물 수정 폼 타이틀
bbs.update.title=게시글 수정하기
```

## 1-6) DB TABLE 생성 및 데이터 입력

먼저 수업 시간에 제공한 springbootstudy-bbs01 프로젝트의 “src/main/resources/SQL/” 폴더의 springbbs01.sql 파일을 이용해 MySQL에 테이블을 생성하고 데이터를 추가하자.

```
## DATABASE 생성 및 선택
DROP DATABASE IF EXISTS springboot;
CREATE DATABASE IF NOT EXISTS springboot;
use springboot;

-- 게시물 번호, 제목, 이메일, 내용, 글쓴이, 날짜, 조회수, 비밀번호, 파일정보,
-- no, title, email, content, writer, reg_date, read_count, pass, file
DROP TABLE IF EXISTS springbbs;
CREATE TABLE IF NOT EXISTS springbbs(
  no INTEGER AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(50) NOT NULL,
  writer VARCHAR(20) NOT NULL,
  content VARCHAR(1000) NOT NULL,
  reg_date TIMESTAMP NOT NULL,
  read_count INTEGER(5) NOT NULL,
  pass VARCHAR(20) NOT NULL,
  file1 VARCHAR(100)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

## 2) Spring Boot 게시판 CRUD 구현

Spring Boot 프로젝트에서 Spring MVC와 MyBatis를 활용해 게시판의 기본적인 기능인 게시글 리스트 보기, 상세보기, 쓰기, 수정, 삭제하기를 구현할 것이다. 게시판을 구현할 때 게시글 리스트 보기를 먼저 구현하고 상세보기, 글쓰기, 수정하기, 삭제하기 순으로 구현할 것이다.

게시판의 기본적인 CRUD(Create, Read, Update, Delete) 기능이 완료되면 게시판 페이징 기능과 검색 기능을 추가하고 파일 업로드, 다운로드를 구현할 계획이다.

### 2-1) 게시글 리스트 보기

게시글 리스트는 DB 테이블에 있는 모든 게시글을 읽어와 화면에 출력할 수 있도록 구현할 것이다. 게시판의 CRUD(Create, Read, Update, Delete) 기능이 완료되면 이후에 게시판 페이징 기능을 추가할 것이다.

#### ▶ 게시글 DTO 클래스

프로젝트에 com.springbootstudy.bbs.domain 패키지를 새로 만들어서 게시글 하나의 정보를 관리하며 springbbs 테이블의 컬럼과 1:1로 맵핑되는 Board 클래스를 다음 코드를 참고해 작성하자.

##### - com.springbootstudy.bbs.domain.Board

```
/* DTO(Data Transfer Object) 객체는 요청과 응답 사이에서 계층 간에 값을
 * 전달하는 목적으로 사용하는 객체이다. 요청이 들어올 때와 응답으로 나갈 때
 * 각각 용도에 맞게 정의하여 사용해야 하지만 우리는 게시판 관련 요청과 응답
 * 모두에서 Board 클래스 하나를 정의해 사용할 것이다.
 */
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Board {
    private int no;
    private String title;
    private String writer;
    private String content;
    private Timestamp regDate;
    private int readCount;
    private String pass;
    private String file1;
}
```

#### ▶ Persistence 계층 구현

Persistence 계층은 데이터베이스에 접근하는 영역으로 우리는 MyBatis를 활용하여 매퍼 인터페이

스와 매퍼 XML을 사용할 것이다.

## ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

먼저 프로젝트의 src/main/resources/ 폴더 아래에 mappers 폴더를 새로 생성하고 수업 시간에 제공한 springbootstudy-bbs01 프로젝트에서 Mapper-Template.xml 파일을 복사해 가져와서 파일 이름을 BoardMapper.xml로 변경한 후 다음을 참고해 DB 테이블에서 게시글 리스트를 읽어오는 매퍼링 구문을 작성하자.

### - src/main/resources/mappers/BoardMapper.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">
<!--
    마이바티스 참고 사이트
    - http://blog.mybatis.org/
    - http://www.mybatis.org/mybatis-3/ko/index.html

    @Mapper 애노테이션이 적용된 인터페이스와 매퍼 XML파일은
    namespace 속성으로 연결되기 때문에 매퍼 XML의 namespace를
    지정할 때 @Mapper 애노테이션이 적용된 매퍼 인터페이스의
    완전한 클래스 이름(패키지를 포함한 이름)을 지정하면 된다.
-->
<mapper namespace="com.springbootstudy.bbs.mapper.BoardMapper" >
```

```
<!--
    게시글 리스트를 가져오는 매퍼링 구문
```

테이블의 컬럼명은 일반적으로 언더스코어 표기법("\_")을 사용하는 경우가 많고 자바 클래스의 인스턴스 멤버는 카멜표기법을 사용한다.  
application.properties 파일에 MyBatis 설정을 할 때 다음과 같이 설정 했기 때문에 별도 설정 없이 컬럼 데이터를 잘 읽어올 수 있다.

```
mybatis.configuration.map-underscore-to-camel-case=true
```

만약 위와 같이 설정할 수 없고 테이블의 컬럼명과 도메인 클래스의 프로퍼티 이름이 다른 경우 아래와 같이 SELECT 쿼리에 별칭을 사용해 도메인 클래스의 프로퍼티 이름과 동일하게 맞춰야 한다. 그렇지 않으면 오류는 발생하지 않지만 해당 컬럼의 데이터를 읽어 올 수 없다. 이외에 ResultMap을 사용해 컬럼명과 다른 이름을 가진 도메인 클래스의 프로퍼티를 연결할 수도 있다.

```
SELECT read_count AS readCount FROM springbbs
```

resultType에 Board를 지정하고 BoardMapper 인터페이스에서 메서드를 정의할 때 반환 타입을 List<Board>로 지정하면 MyBatis가 쿼리를 실행한 결과 집합에서

한 행의 데이터를 Board 객체로 만들어 List에 담아서 반환되도록 처리해 준다.

```
-->
<select id="boardList" resultType="Board">
    SELECT
        *
    FROM springbbs
    ORDER BY no DESC
</select>
</mapper>
```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

프로젝트에 com.springbootstudy.bbs.mapper 패키지를 새로 만들고 다음 코드를 참고해 MyBatis를 활용해 게시판 관련 DB 작업을 전담하는 BoardMapper 인터페이스를 작성하자.

### - com.springbootstudy.bbs.mapper.BoardMapper

```
/* 이전에는 DAO(Data Access Object) 클래스에 @Repository 애노테이션을
 * 적용하여 해당 클래스가 DB 작업을 하는 클래스임을 명시하고
 * MyBatis의 Mapper 인터페이스나 XML 매퍼를 통해서 DB와 통신하였다.
 * 이때 DAO 클래스에서 XML 매퍼 파일에 정의한 namespace라는 속성과
 * 매퍼 파일 안에 작성한 SQL 쿼리(매퍼링 구문)의 id를 조합해 SQL 쿼리를
 * 호출했었다. 하지만 요즘에는 아래와 같이 자바 인터페이스에 @Mapper
 * 애노테이션을 적용하고 이 인터페이스의 메서드에 @Select, @Insert,
 * @Update, @Delete 등의 애노테이션을 지정해 쿼리를 직접 매퍼링할 수도 있다.
 * 또한 별도의 XML 매퍼 파일을 만들고 그 안에 SQL 쿼리를 정의하여 이 매퍼링
 * 구문을 호출할 수도 있다. 이때 한 가지 주의할 점은 @Mapper 애노테이션을
 * 적용한 인터페이스와 XML 매퍼 파일은 namespace라는 속성으로 연결되기
 * 때문에 XML 매퍼 파일의 namespace를 정의할 때 매퍼 인터페이스의 완전한
 * 클래스 이름과 동일한 namespace를 사용해야 한다는 것이다.
 *
 * @Mapper는 MyBatis 3.0부터 지원하는 애노테이션으로 이 애노테이션이 붙은
 * 인터페이스는 별도의 구현 클래스를 작성하지 않아도 MyBatis 매퍼로 인식해
 * 스프링 Bean으로 등록되며 Service 클래스에서 주입 받아 사용할 수 있다.
 */
@Mapper
public interface BoardMapper {

    // 게시글 리스트를 DB 테이블에서 읽어와 반환하는 메소드
    public List<Board> boardList();
}
```

## ▶ Business 계층 구현

프로젝트에 com.springbootstudy.bbs.service 패키지를 새로 만들고 다음 코드를 참고해 게시판 관련 비즈니스 로직을 담당하는 BoardService 클래스를 작성하자.



#### - com.springbootstudy.bbs.service.BoardService

```
// BoardService 클래스가 서비스 계층의 스프링 빈(Beans)임을 정의
@Service
@Slf4j

public class BoardService {

    // DB 작업에 필요한 BoardMapper 객체를 의존성 주입 설정
    @Autowired
    private BoardMapper boardMapper;

    // 전체 게시글을 읽어와 반환하는 메서드
    public List<Board> boardList() {
        log.info("BoardService: boardList()");
        return boardMapper.boardList();
    }
}
```

### ▶ Controller 클래스

프로젝트에 com.springbootstudy.bbs.controller 패키지를 새로 만들고 다음 코드를 참고해 게시판 관련 HTTP 요청을 처리하는 BoardController 클래스를 작성하자.

#### - com.springbootstudy.bbs.controller.BoardController

```
/* Spring MVC Controller 클래스임을 정의
 * @Controller 애노테이션이 적용된 클래스의 메서드는 기본적으로 뷰의 이름을 반환한다.
 */
@Controller
@Slf4j

public class BoardController {

    @Autowired
    private BoardService boardService;

    /* 게시글 리스트 요청을 처리하는 메서드
     * "/", "/boardList" 로 들어오는 HTTP GET 요청을 처리하는 메서드
     *
     * 스프링은 컨트롤러에서 모델에 데이터를 담을 수 있는 다양한 방법을 제공하며
     * 아래와 같이 파라미터에 Model을 지정하는 방식이 많이 사용된다.
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터에 Model
     * 을 지정하면 스프링이 이 메서드를 호출하면서 Model 타입의 객체를 넘겨준다.
     * 우리는 Model을 받아 이 객체에 결과 데이터를 담기만 하면 뷰에서 사용할 수 있다.
     */
    @GetMapping({"/", "/boardList"})
    public String boardList(Model model) {
```

```

model.addAttribute("bList", boardService.boardList());

/* application.properties 파일에서 Thymeleaf 설정을 별도로 지정하지
 * 않았기 때문에 다음과 같이 기본 설정이 적용되어 templates/ 폴더를
 * 기준으로 뷰를 찾아서 포워드 되고 제어가 뷰 페이지로 이동한다.
 *
 * spring.thymeleaf.prefix=classpath:/templates/
 * spring.thymeleaf.suffix=.html
 */
return "main";
}
}

```

## ▶ View 페이지

수업 시간에 제공한 springbootstudy-bbs01 프로젝트의 src/main/resources/templates/ 폴더에 main\_template.html 파일이 있는데 이 파일은 아래 main.html 파일의 header 부분과 footer 부분이 작성되어 있고 게시글 리스트를 출력하는 부분은 작성되어 있지 않은 미완성 파일이다. 이 파일을 활용하면 header와 footer를 작성하는 수고 없이 Thymeleaf를 사용해 게시글 리스트를 출력하는 실습을 바로 할 수 있다.

### - src/main/resources/templates/main.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>BoardService</title>
    <link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
    <style>
    </style>
  </head>
  <body>
    <div class="container">
      <!-- header -->
      <div class="row border-bottom border-primary">
        <div class="col-4">
          <p></p>
        </div>
        <div class="col-8">
          <div class="row">
            <div class="col">
              <ul class="nav justify-content-end">
                <li class="nav-item">
                  <a class="nav-link" href="#">로그인</a>

```

```

        </li>
        <li class="nav-item">
            <a class="nav-link" href="boardList">게시글 리스트</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">회원가입</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">주문/배송조회</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">고객센터</a>
        </li>
    </ul>
</div>
</div>
<div class="row">
    <div class="col text-end">로그인시 인사말 출력</div>
</div>
</div>
<!-- content -->
<div class="row my-5" id="global-content">
    <div class="col-10 offset-1">
        <div class="row text-center">
            <div class="col">
                <h2 class="fs-3 fw-bold">게시글 리스트</h2>
            </div>
        </div>
    </div>
    <form name="searchForm" id="searchForm" action="boardList"
        class="row justify-content-center my-3">
        <div class="col-auto">
            <select name="type" class="form-select">
                <option value="title">제목</option>
                <option value="writer">작성자</option>
                <option value="content">내용</option>
            </select>
        </div>
        <div class="col-4">
            <input type="text" name="keyword" class="form-control"/>
        </div>
        <div class="col-auto">
            <input type="submit" value="검 색" class="btn btn-primary"/>
        </div>
    </form>

```

```

<div class="row">
  <div class="col text-end">
    <a class="btn btn-outline-success">글쓰기</a>
  </div>
</div>
<div class="row my-3">
  <div class="col">
    <table class="table">
      <thead>
        <tr class="table-dark">
          <th>NO</th>
          <th>제목</th>
          <th>작성자</th>
          <th>작성일</th>
          <th>조회수</th>
        </tr>
      </thead>
      <tbody>
        <th:block th:if="{not #lists.isEmpty(bList)}">
          <tr th:block th:each="board, status: ${bList}">
            <td>[[ ${board.no} ]]</td>
            <td>[[ ${board.title} ]]</td>
            <td th:text="{ ${ board.writer} }"></td>
            <td>[[ ${ #dates.format(board.regDate, 'yyyy-MM-dd')} ]]</td>
            <td th:text="{ ${ board.readCount} }"></td>
          </tr>
        </th:block>
        <th:block th:unless="{not #lists.isEmpty(bList)}">
          <tr>
            <td colspan="5">게시글이 존재하지 않음</td>
          </tr>
        </th:block>
      </tbody>
    </table>
  </div>
</div>
</div>
<!-- footer -->
<div class="row border-top border-primary my-5">
  <div class="col text-center py-3">
    <p>고객상담 전화주문:1234-5678 사업자등록번호 :111-11-123456
    대표이사: 홍길동 통신판매업 서울 제 000000호<br/>
    개인정보관리책임자:임격정 분쟁조정기관표시 : 소비자보호원, 전자거래분쟁중재위원회
  </div>
</div>

```

```

        Copyright (c) 2016 Spring2U Corp. All right Reserved.
    </p>
</div>
</div>
</div>
<script src="bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

여기까지 소스코드를 작성했다면 게시글 리스트가 잘 동작하는지 프로젝트를 시작하고 테스트를 해 보자.

## 2-2) 뷰 페이지 모듈화

앞에서 작성한 뷰 페이지를 보면 메뉴와 푸터 부분은 여러 페이지에 표시되기 때문에 이 부분은 매번 중복되는데 이렇게 중복되는 코드를 줄이고 효율적으로 뷰 페이지를 관리하기 위해서 중복되는 코드를 header, content, footer 부분으로 나눠서 각각의 파일로 모듈화 할 것이다.

페이지 모듈화는 하나의 페이지를 표현하는 템플릿을 여러 개로 나눠서 템플릿 조각 파일로 저장하고 실행될 때 다시 하나로 동작하도록 만들면 되는데 Thymeleaf에서 하나의 페이지를 템플릿 조각으로 나눠서 저장하기 위해서 fragment와 layout을 사용한다. 그리고 여러 개로 나눠진 fragment와 layout 파일을 하나로 묶어 주는 메인 레이아웃 파일을 만들어서 이들을 하나로 동작할 수 있도록 설정하면 된다. 다음은 페이지 모듈화에 적용할 폴더와 용도를 정리한 표이다.

폴더	용도
templates/fragments	header와 footer 등의 fragment를 저장
templates/layouts	여러 fragment와 layout을 하나로 묶어 주는 레이아웃 파일 저장
templates/views	요청마다 변경되는 contents 페이지 저장

### ▶ Thymeleaf 레이아웃 의존설정

- build.gradle에 추가하고 Refresh Gradle Project 실행  
implementation 'nz.net.ultraq.thymeleaf:thymeleaf-layout-dialect'

### ▶ 템플릿 헤더 페이지

“src/main/resources/templates” 폴더에 fragments 폴더를 새로 만들고 header.html 파일을 생성해 앞에서 작성한 main.html 파일에서 header 부분만 가져와 다음과 같이 작성하자.

- src/main/resources/templates/fragments/header.html
- ```

<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">

```

```

<!--/* header */-->
<!--/* header라는 프래그먼트를 정의 */-->
<div th:fragment="header" class="row border-bottom border-primary">
  <div class="col-4">
    <p></p>
  </div>
  <div class="col-8">
    <div class="row">
      <div class="col">
        <ul class="nav justify-content-end">
          <li class="nav-item">
            <a class="nav-link" href="#">로그인</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="boardList">게시글 리스트</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">회원가입</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">주문/배송조회</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">고객센터</a>
          </li>
        </ul>
      </div>
    </div>
    <div class="row">
      <div class="col text-end">로그인시 인사말 출력</div>
    </div>
  </div>
</div>
</html>

```

## ▶ 템플릿 푸터 페이지

fragments 폴더에 footer.html 파일을 새로 만들고 앞에서 작성한 main.html 파일에서 footer 부분만 가져와 다음과 같이 작성하자.

- src/main/resources/templates/fragments/footer.html

```

<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
  <!--/* footer */-->
  <!--/* footer라는 프래그먼트를 정의 */-->

```

```

<div th:fragment="footer" class="row border-top border-primary my-5">
  <div class="col text-center py-3">
    <p>고객상담 전화주문:1234-5678 사업자등록번호 :111-11-123456
    대표이사: 홍길동 통신판매업 서울 제 000000호<br/>
    개인정보관리책임자:임격정 분쟁조정기관표시 : 소비자보호원, 전자거래분쟁중재위원회
  </p>
  Copyright (c) 2016 Spring2U Corp. All right Reserved.
</div>
</div>
</html>

```

## ▶ 게시글 리스트 보기 View

“src/main/resources/templates” 폴더에 views 폴더를 새로 만들고 boardList.html 파일을 생성해 앞에서 작성한 main.html 파일에서 content 부분만 가져와 다음과 같이 작성하자.

### - src/main/resources/templates/views/boardList.html

```

<!DOCTYPE html>
<!--/*
  Thymeleaf와 Thymeleaf Layout을 사용하기 위한 NameSpace를 정의한다.
  layout:decorate 옵션은 아래의 <th:block layout:fragment="content">
  부분을 어떤 레이아웃에 적용할지 설정하는 것으로 경로 지정은
  templates 폴더를 기준으로 지정하면 된다.
*/-->
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layouts/main_layout}">
  <!--/* content */-->
  <!--/*
    block을 사용해도 되고 div에 직접 layout:fragment를 적용해도 된다.
    main_layout.html에서 지정한 layout:fragment의 이름과 같아야 한다.
  */-->
  <th:block layout:fragment="content">
    <div class="row my-5" id="global-content">
      <div class="col-10 offset-1">
        <div class="row text-center">
          <div class="col">
            <h2 class="fs-3 fw-bold" th:text="#{bbs.list.title}>게시글 리스트</h2>
          </div>
        </div>
        <form name="searchForm" id="searchForm" action="boardList"
              class="row justify-content-center my-3">
          <div class="col-auto">
            <select name="type" class="form-select">

```

```

        <option value="title">제목</option>
        <option value="writer">작성자</option>
        <option value="content">내용</option>
    </select>
</div>
<div class="col-4">
    <input type="text" name="keyword" class="form-control"/>
</div>
<div class="col-auto">
    <input type="submit" value="검 색" class="btn btn-primary"/>
</div>
</form>
<div class="row">
    <div class="col text-end">
        <a class="btn btn-outline-success">글쓰기</a>
    </div>
</div>
<div class="row my-3">
    <div class="col">
        <table class="table">
            <thead>
                <tr class="table-dark">
                    <th>NO</th>
                    <th>제목</th>
                    <th>작성자</th>
                    <th>작성일</th>
                    <th>조회수</th>
                </tr>
            </thead>
            <tbody>
                <th:block th:if="{not #lists.isEmpty(bList)}">
                    <tr th:block th:each="board, status: ${bList}">
                        <td>[[ ${board.no} ]]</td>
                        <td>[[ ${board.title} ]]</td>
                        <td th:text="{ board.writer }"></td>
                        <td>[[ ${ #dates.format(board.regDate, 'yyyy-MM-dd') }]]</td>
                        <td th:text="{ board.readCount }"></td>
                    </tr>
                </th:block>
                <th:block th:unless="{not #lists.isEmpty(bList)}">
                    <tr>
                        <td colspan="5">게시글이 존재하지 않음</td>
                    </tr>
                </th:block>
            </tbody>
        </table>
    </div>

```



```

        </table>
    </div>
</div>
</div>
</div>
</th:block>
</html>

```

## ▶ 메인 템플릿 페이지

“src/main/resources/templates” 폴더에 layouts 폴더를 새로 만들고 main\_layout.html 파일을 생성해 앞에서 작성한 header.html, footer.html, boardList.html 페이지가 하나로 동작할 수 있도록 다음과 같이 작성하자.

### - src/main/resources/templates/layouts/main\_layout.html

```

<!DOCTYPE html>
<!--/* Thymeleaf와 Thymeleaf Layout을 사용하기 위한 Namespace 정의 */-->
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>BoardService</title>
        <link th:href="@{bootstrap/bootstrap.min.css}" rel="stylesheet" >
        <style>
        </style>
    </head>
    <body>
        <div class="container">
            <!--/* header */-->
            <!--/*
                ~{} 표현식은 화면의 조각인 프래그먼트를 지정하는 표현식으로
                th:replace 속성에 지정했기 때문에 div 요소의 내용을 대체한다.
                프래그먼트는 ~{경로/파일명::프래그먼트 이름} 형식으로 지정한다.
                경로는 templates 폴더를 기준으로 상대 경로 방식으로 지정하면 된다.
            */-->
            <div th:replace="~{fragments/header::header}"></div>
            <!--/* content : 요청 마다 변경되는 부분 */-->
            <div layout:fragment="content"></div>
            <!--/* footer */-->
            <div th:replace="~{fragments/footer::footer}"></div>
        </div>
        <script th:src="@{bootstrap/bootstrap.bundle.min.js}"></script>
    </body>
</html>

```

## ▶ Controller 클래스 수정

HTTP 요청에 따라가 변경되는 content 페이지가 “templates/views” 폴더로 변경되었기 때문에 컨트롤러에서 뷰의 이름을 반환하는 부분도 다음과 같이 수정되어야 한다.

### - com.springbbsstudy.bbs.controller.BoardController

```
@GetMapping("/{", "/boardList"})
public String boardList(Model model) {
    model.addAttribute("bList", boardService.boardList());

    /* application.properties 파일에서 Thymeleaf 설정을 별도로 지정하지
    * 않았기 때문에 다음과 같이 기본 설정이 적용되어 templates/ 폴더를
    * 기준으로 뷰를 찾아서 포워드 되고 제어가 뷰 페이지로 이동한다.
    *
    * spring.thymeleaf.prefix=classpath:/templates/
    * spring.thymeleaf.suffix=.html
    */
    // 페이지 모듈화로 content 페이지가 "/templates/views" 폴더로 이동함
    return "views/boardList";
}
```

## 2-4) 게시물 상세보기

앞에서 스프링 부트와 MyBatis를 활용해 웹 애플리케이션을 구현하기 위한 설정과 Mapper 인터페이스를 이용해 게시판 테이블로부터 게시물 리스트를 읽어와 출력하는 과정에 대해서 알아보았다. 지금부터는 앞에서 설정한 정보를 그대로 사용하면서 구현하는 기능에 맞춰서 Mapper, Service, Controller, View 페이지를 구현해 추가하면 된다.

## ▶ 게시물 리스트에 게시물 상세보기 링크 추가

게시물 리스트 화면에서 게시물 상세보기로 연결되는 링크를 다음과 같이 추가하자.

### - src/main/resources/templates/views/boardList.html에 링크 추가

... 중 략 ...

```
<tr th:block th:each="board, status: ${bList}">
    <td>[[ ${board.no} ]]</td>
    <td><a th:href="@{boardDetail(no=${board.no})}">[[ ${board.title} ]]</a></td>
    <td th:text="${ board.writer }"></td>
    <td>[[ ${ #dates.format(board.regDate, 'yyyy-MM-dd') } ]]</td>
    <td th:text="${ board.readCount }"></td>
```

</tr>

... 중 략 ...

## ▶ Persistence 계층 구현

MyBatis를 활용하여 게시글 정보를 DB 테이블에서 읽어오는 매퍼 인터페이스와 매퍼 XML을 다음과 같이 작성하자.

### ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음은 참고해 BoardMapper.xml 파일에 DB 테이블에서 no에 해당하는 게시글을 읽어오는 매퍼 구문을 추가하자.

#### - src/main/resources/mappers/BoardMapper.xml

```
<!--
```

```
no에 해당하는 게시글 하나를 가져오는 매퍼 구문
```

```
application.properties 파일에 MyBatis 설정을 할 때 다음과 같이 설정 했기  
때문에 resultType에 패키지를 제외하고 클래스 이름을 지정할 수 있다.
```

```
mybatis.type-aliases-package=com.springbootstudy.bbs.domain
```

```
parameterType은 기본형(int)이므로 생략할 수 있으며 BoardMapper 인터페이스에서  
전달된 파라미터는 #{ } 로 감싸서 SQL 쿼리에서 사용할 수 있다. 이 매퍼 구문을  
호출하는 getBoard(int no) 메서드의 파라미터 no를 매퍼 구문의 SQL 쿼리에서  
사용하려면 다음과 같이 #{no}로 사용하면 된다.
```

```
-->
```

```
<select id="getBoard" resultType="Board">
```

```
SELECT
```

```
    *
```

```
FROM springbbs
```

```
WHERE no = #{no}
```

```
</select>
```

### ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 no에 해당하는 게시글을 DB 테이블에서 읽어와 반환하는 메서드를 다음과 같이 작성하자.

#### - com.springbootstudy.bbs.mapper.BoardMapper

```
// DB 테이블에서 no에 해당하는 게시글을 읽어와 Board 객체로 반환하는 메서드
```

```
public Board getBoard(int no);
```

## ▶ Business 계층 구현

BoardService 클래스에 게시글 상세보기 기능을 수행하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.service.BoardService

```
// no에 해당하는 게시글을 읽어와 반환하는 메서드
public Board getBoard(int no) {
    log.info("BoardService: getBoard(int no)");
    return boardMapper.getBoard(no);
}
```

## ▶ Controller 클래스

BoardController 클래스에 게시글 상세보기 요청을 처리하는 메서드를 추가하자.

### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 상세보기 요청 처리 메서드
 * "/boardDetail"로 들어오는 HTTP GET 요청을 처리하는 메서드
 *
 * 스프링은 클라이언트로부터 넘어 오는 요청 파라미터를 받을 수 있는 여러 가지
 * 방법을 제공하고 있다. 아래와 같이 HTTP 요청을 처리하는 메서드의 파라미터
 * 앞에 @RequestParam("요청 파라미터 이름") 애노테이션을 사용해 요청
 * 파라미터의 이름을 지정하면 된다.
 *
 * @GetMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
 * @RequestParam 애노테이션을 사용해 요청 파라미터 이름을 지정하면
 * 이 애노테이션이 앞에 붙은 매개변수에 요청 파라미터 값을 바인딩 시켜준다.
 *
 * @RequestParam 애노테이션에 사용할 수 있는 속성은 아래와 같다.
 * value : HTTP 요청 파라미터의 이름을 지정한다.
 * required : 요청 파라미터가 필수인지 설정하는 속성으로 기본값은 true 이다.
 * 이 값이 true인 상태에서 요청 파라미터의 값이 존재하지 않으면
 * 스프링은 Exception을 발생시킨다.
 *
 * defaultValue : 요청 파라미터가 없을 경우 사용할 기본 값을 문자열로 지정한다.
 *
 * @RequestParam(value="no" required=false defaultValue="1")
 *
 * @RequestParam 애노테이션은 요청 파라미터 값을 읽어와 Controller 메서드의
 * 파라미터 타입에 맞게 변환해 준다. 만약 요청 파라미터를 Controller 메서드의
 * 파라미터 타입으로 변환할 수 없는 경우 스프링은 400 에러를 발생시킨다.
 */
@GetMapping("/boardDetail")
public String getBoard(Model model, @RequestParam("no") int no) {
```

```

    model.addAttribute("board", boardService.getBoard(no));
    return "views/boardDetail";
}

```

## ▶ 게시글 상세보기 View

- src/main/resources/templates/views/boardDetail.html

```

<!DOCTYPE html>
<!--/*
    Thymeleaf와 Thymeleaf Layout을 사용하기 위한 NameSpace 정의
    layout:decorate 옵션은 아래의 <th:block layout:fragment="content"> 부분을 어떤
    레이아웃에 적용할지 설정하는 부분으로 경로 지정은 templates를 기준으로 지정하면 됨
*/-->
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layouts/main_layout}">
<!--/* content */-->
<!--/*
    block을 사용해도 되고 div에 직접 layout:fragment를 적용해도 됨
    main_layout.html에서 지정한 layout:fragment의 이름과 같아야 함
*/-->
<th:block layout:fragment="content">
    <div class="row my-5" id="global-content">
        <div class="col-10 offset-1">
            <form name="checkForm" id="checkForm">
                <input type="hidden" name="no" id="no" th:value="${board.no}"/>
                <input type="hidden" name="pass" id="rPass" />
            </form>
            <div class="row text-center">
                <div class="col">

                    <h2 class="fs-3 fw-bold" th:text="#{bbs.detail.title}>게시글 상세보기</h2>
                </div>
            </div>
            <div class="row my-3">
                <div class="col">
                    <table class="table table-bordered" >
                        <tbody>
                            <tr>
                                <th class="table-secondary">제 목</th>
                                <td colspan="3" th:text="${board.title}></td>
                            </tr>
                            <tr>
                                <th>글쓴이</th>
                                <td th:text="${board.writer}></td>

```



## 2-5) 게시글 쓰기

게시글 쓰기는 게시글을 작성할 수 있는 폼을 보여주는 요청과 게시글 쓰기 폼에서 사용자가 작성한 게시글을 저장하는 요청으로 나누어 처리해야 한다. 게시글 쓰기 폼을 보여주는 요청은 단순히 폼을 화면에 보여주지만 하면 되므로 Business 계층과 Persistence 계층의 클래스는 구현할 필요가 없다.

### 2-5-1) 게시글 쓰기 폼 요청 처리

게시글 쓰기 폼 요청은 폼만 보여주면 되므로 컨트롤러 클래스에 다음과 같이 뷰만 반환하는 메서드를 추가하자.

#### ▶ 게시글 리스트에 게시글 쓰기 폼 링크 추가

게시글 리스트 화면에서 게시글 쓰기 폼으로 연결되는 링크를 다음과 같이 추가하자.

- src/main/resources/templates/views/boardList.html에 링크 추가

... 중 략 ...

```
<div class="row">
  <div class="col text-end">
    <a th:href="@{addBoard}" class="btn btn-outline-success">글쓰기</a>
  </div>
</div>
```

... 중 략 ...

#### ▶ Controller 클래스

- com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 쓰기 폼 요청 처리 메서드
 * "/addBoard"로 들어오는 HTTP GET 요청을 처리하는 메서드
 */
@GetMapping("/addBoard")
public String addBoard() {
    // 게시글 쓰기 폼은 모델이 필요 없기 때문에 뷰만 반환
    return "views/writeForm";
}
```

## ▶ WebMvc 환경설정 클래스에 뷰 전용 컨트롤러 설정

다음과 같이 WebMvcConfigurer 인터페이스를 상속받아서 스프링 MVC 환경설정 클래스를 만들고 이 클래스에 모델이 필요 없는 요청에 대해서 뷰 전용 컨트롤러를 설정할 수 있다.

- com.springbootstudy.bbs.configurations.WebConfig

```
/* @Configuration 애노테이션은 스프링 환경설정과 스프링 빈(Spring Bean)을
 * 등록하기 위한 애노테이션 이다. 이 애노테이션은 스프링 Bean을 등록할 때
 * 싱글톤(Singleton)으로 생성해주며 스프링 DI 컨테이너가 Bean을 관리해 준다.
 * WebMvcConfigurer는 스프링 MVC 환경 설정을 위해서 제공되는 인터페이스 이다.
 */
@Configuration
public class WebConfig implements WebMvcConfigurer{

    /* 어떤 요청에 대해서 요청을 처리한 결과 데이터인 모델은 필요 없고 화면만
     * 보여주는 경우 아래와 같이 addViewControllers() 메서드를 오버라이드하여
     * 뷰 전용 컨트롤러를 설정할 수 있다.
     */
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        /* 아래는 /writeForm과 /writeBoard 요청에 대한 뷰를 지정한 것이다.
         * 여기서 지정한 뷰의 이름과 prefix, suffix를 조합하여 실제 뷰의
         * 물리적인 이름이 결정된다.
         */
        registry.addViewController("/writeForm").setViewName("views/writeForm");
        registry.addViewController("/writeBoard").setViewName("views/writeForm");
    }
}
```

## ▶ 게시글 쓰기 폼 View

- src/main/resources/templates/views/writeForm.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layouts/main_layout}">
<!--/* content */-->
<th:block layout:fragment="content">
    <div class="row my-5" id="global-content">
        <div class="col">
            <div class="row text-center">
                <div class="col">
                    <h2 class="fs-3 fw-bold" th:text="#{bbs.write.title}">게시글 쓰기</h2>
                </div>
            </div>
            <form name="writeForm" action="addBoard" id="writeForm">
```



```

class="row g-3 border-primary" method="post">
  <div class="col-4 offset-md-2">
    <label for="writer" class="form-label">글쓴이</label>
    <input type="text" class="form-control" name="writer" id="writer"
      placeholder="작성자를 입력해 주세요">
  </div>
  <div class="col-4 ">
    <label for="pass" class="form-label">비밀번호</label>
    <input type="password" class="form-control" name="pass" id="pass" >
  </div>
  <div class="col-8 offset-md-2">
    <label for="title" class="form-label">제 목</label>
    <input type="text" class="form-control" name="title" id="title" >
  </div>
  <div class="col-8 offset-md-2">
    <label for="content" class="form-label">내 용</label>
    <textarea class="form-control" name="content" id="content"
rows="10"></textarea>
  </div>
  <div class="col-8 offset-md-2 text-center mt-5">
    <input type="submit" value="등록하기" class="btn btn-primary"/>
    &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
      onclick="location.href='boardList'" class="btn btn-primary"/>
  </div>
</form>
</div>
</div>
</th:block>
</html>

```

## ▶ JavaScript

resources/static 폴더 하위에 자바스크립트를 위한 js 폴더를 새로 만들고 그 안에 formcheck.js 파일을 생성해 게시글 쓰기 폼의 유효성 검사를 수행하는 다음과 같은 코드를 추가하자.

- src/main/resources/static/js/formcheck.js

```

(function() {
  // 게시글 쓰기 폼 유효성 검사
  $("#writeForm").on("submit", function() {
    if($("#writer").val().length <= 0) {
      alert("작성자가 입력되지 않았습니다.\n작성자를 입력해주세요");
      $("#writer").focus();
      return false;
    }
    if($("#title").val().length <= 0) {

```

```

        alert("제목이 입력되지 않았습니다.\n제목을 입력해주세요");
        $("#title").focus();
        return false;
    }
    if($("#pass").val().length <= 0) {
        alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
        $("#pass").focus();
        return false;
    }
    if($("#content").val().length <= 0) {
        alert("내용이 입력되지 않았습니다.\n내용을 입력해주세요");
        $("#content").focus();
        return false;
    }
});
});

```

## ▶ 메인 레이아웃 파일에 자바스크립트 참조 추가

- src/main/resources/templates/layouts/main\_layout.html

```

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>BoardService</title>
    <link th:href="@{bootstrap/bootstrap.min.css}" rel="stylesheet" >
    <script th:src="@{js/jquery-3.7.1.min.js}"></script>
    <script th:src="@{js/formcheck.js}"></script>
</head>

```

## 2-5-2) 게시글 쓰기 요청 처리

게시글 쓰기 폼으로부터 들어오는 요청을 처리하는 방법에 대해서 알아보자. 사용자가 게시글 쓰기 폼에서 작성한 게시글을 서버로 전송하면 서버에서는 게시글 정보를 DB에 저장한 후 게시글 리스트 페이지로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다.

## ▶ Persistence 계층 구현

MyBatis를 활용하여 게시글 정보를 DB 테이블에 추가하는 매퍼 인터페이스와 매퍼 XML을 다음과 같이 작성하자.

## ▣ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음을 참고해 BoardMapper.xml 파일에 DB 테이블에 게시글을 추가하는 매퍼 구문을 작성하자.

- src/main/resources/mappers/BoardMapper.xml

<!--

게시글을 테이블에 추가하는 맵핑 구문

아래는 BoardMapper 인터페이스의 insertBoard(Board board) 메서드에서 호출하는 맵핑 구문으로 parameterType을 Board 타입으로 지정하면 된다. 아래와 같이 parameterType에 도메인 클래스 타입을 지정하는 경우 VALUES()에 지정하는 값은 getter 메서드를 지정하는 것이 아니라 클래스의 프로퍼티만 (인스턴스 변수)를 #{ }로 감싸서 지정하면 MyBatis가 알아서 처리해 준다. 이 맵핑 구문이 실행되고 반환되는 값은 테이블에 추가된 행의 개수를 정수로 반환하는데 반환 값이 필요없다면 resultType은 생략할 수 있다.

테이블에 하나의 레코드를 INSERT 할때 자동으로 증가되는 컬럼이나 Sequence를 사용하는 컬럼의 값을 읽어 와야 할 때도 있다.

보통 자동 증가되는 컬럼의 값은 데이터가 INSERT 된 후에 읽어오고 Sequence일 경우 INSERT 이전에 값을 읽어 와야 한다.

이렇게 INSERT 작업을 하면서 생성된 키의 값을 읽어 와야 할 경우

아래와 같이 useGeneratedKeys="true"를 지정하고 자동 생성된

키의 값을 설정할 자바 모델 객체의 프로퍼티 이름을 keyProperty에

지정하면 Board 객체의 no 프로퍼티에 값을 설정해 준다.

-->

```
<insert id="insertBoard" parameterType="Board"
```

```
  useGeneratedKeys="true" keyProperty="no">
```

```
  INSERT INTO springbbs(title, writer, content,
```

```
    reg_date, read_count, pass, file1)
```

```
    VALUES(#{title}, #{writer}, #{content},
```

```
      SYSDATE(), #{readCount}, #{pass}, #{file1})
```

```
</insert>
```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 DB 테이블에 게시글을 추가하는 메서드를 다음과 같이 작성하자.

- com.springbootstudy.bbs.mapper.BoardMapper

```
//게시글을 Board 객체로 받아서 DB 테이블에 추가하는 메서드
```

```
public void insertBoard(Board board);
```

## ▶ Business 계층 구현

BoardService 클래스에 게시글 쓰기 기능을 수행하는 다음과 같은 메서드를 추가하자.

- com.springbootstudy.bbs.service.BoardService

```
// 게시글 정보를 추가하는 메서드
```

```
public void addBoard(Board board) {
```

```

    log.info("BoardService: addBoard(Board board)");
    boardMapper.insertBoard(board);
}

```

## ▶ Controller 클래스

BoardController 클래스에 게시글 쓰기 폼에서 들어오는 HTTP POST 요청을 처리하는 메서드를 추가하자. 이번에 게시글 쓰기 요청을 처리하는 방법은 폼으로부터 전달된 요청 파라미터를 스프링이 커맨드 객체를 이용해 한 번에 편리하게 받을 수 있도록 지원하는 기능을 이용해 요청 파라미터를 처리하는 방법을 설명하고 있다. 스프링을 사용하기 전에는 HttpServletRequest 객체를 이용해 요청 파라미터를 하나씩 일일이 받아서 Board 객체에 저장했지만 스프링이 지원하는 기능을 사용하면 보다 편리하게 요청 파라미터를 DTO 객체로 받을 수 있다.

### - com.springbootstudy.bbs.controller.BoardController

```

/* 게시글 쓰기 폼에서 들어오는 게시글 쓰기 요청을 처리하는 메서드
 * "/addBoard"로 들어오는 HTTP POST 요청을 처리하는 메서드
 *
 * @PostMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
 * 게시글 쓰기 폼에서 전달되는 요청 파라미터를 받을 Board 객체를
 * 지정했다.
 *
 * 스프링은 폼으로부터 전달된 파라미터를 객체로 처리 할 수 있는 아래와 같은
 * 방법을 제공하고 있다. 아래와 같이 요청 파라미터를 전달받을 때 사용하는
 * 객체를 커맨드 객체라고 부르며 이 커맨드 객체는 자바빈 규약에 따라 프로퍼티에
 * 대한 setter를 제공하도록 작성해야 한다. 그리고 파라미터 이름이 커맨드 객체의
 * 프로퍼티와 동일하도록 폼 컨트롤의 name 속성을 지정해야 한다.
 *
 * HTTP 요청을 맵핑하는 애노테이션(@PostMapping, @GetMapping 등)이
 * 적용된 컨트롤러의 메서드에 커맨드 객체를 파라미터로 지정하면 커맨드 객체의
 * 프로퍼티와 동일한 이름을 가진 요청 파라미터의 데이터를 스프링이 자동으로
 * 연결해 준다. 이때 스프링은 자바빈 규약에 따라 적절한 setter 메서드를 사용해
 * 값을 설정한다.
 *
 * 커맨드 객체의 프로퍼티와 일치하는 파라미터 이름이 없다면 기본 값으로 설정된다.
 * 또한 프로퍼티의 데이터 형에 맞게 적절히 형 변환 해 준다. 형 변환을 할 수 없는
 * 경우 스프링은 400 에러를 발생 시킨다. 예를 들면 프로퍼티가 정수형 일 때 매칭
 * 되는 값이 정수로 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 */
@PostMapping("/addBoard")
public String addBoard(Board board) {
    log.info("title : ", board.getTitle());
    boardService.addBoard(board);

    /* 게시글 쓰기가 완료되면 게시글 리스트로 리다이렉트 시킨다.
     * 클라이언트 요청을 처리한 후 리다이렉트 해야 할 경우 아래와 같이 redirect:

```

- \* 접두어를 붙여 뷰 이름을 반환하면 된다. 뷰 이름에 redirect 접두어가 붙으면
- \* HttpServletResponse를 사용해서 지정한 경로로 Redirect 된다.
- \* redirect 접두어 뒤에 경로를 지정할 때 "/"로 시작하면 ContextRoot를
- \* 기준으로 절대 경로 방식으로 Redirect 된다. "/"로 시작하지 않으면 현재
- \* 경로를 기준으로 상대 경로로 Redirect 된다. 또한 다른 사이트로 Redirect
- \* 되기를 원한다면 "redirect:http://사이트 주소"를 지정하면 된다.

```

**/
return "redirect:boardList";
}

```

## 2-6) 게시글 수정

게시글 수정하기는 게시글을 수정할 수 있는 폼을 먼저 보여주는 요청과 게시글 수정 폼에서 사용자가 수정한 게시글을 저장하는 요청으로 나누어 처리해야 한다. 게시글 수정 폼을 보여주는 요청은 이전에 작성한 게시글을 폼에 출력하여 보여줘야 하므로 게시글 쓰기와는 다르게 DB에 저장된 게시글 내용을 읽어와 폼에 출력하는 처리가 필요하다. 그리고 게시글 수정 기능은 게시글 상세보기에서 수정하기 버튼을 클릭해 수정 폼으로 이동하는 방식으로 구현할 것이다. 이때 사용자가 입력한 비밀번호와 기존 비밀번호를 비교해 일치해야 게시글 수정 폼으로 이동할 수 있도록 구현할 것이다. 그래서 사용자가 입력한 비밀번호와 기존 비밀번호가 일치하는지 체크하는 기능도 필요하다.

### 2-6-1) 게시글 수정 폼 요청처리

게시글 수정 폼에 출력해야 할 데이터는 앞에서 구현한 게시글 상세보기의 getBoard(int no) 메서드를 그대로 사용하면 된다. 그러므로 Mapper 설정 파일과 BoardMapper 인터페이스에 추가할 기능은 게시글 상세보기에서 게시글 수정 폼 요청을 할 때 비밀번호를 체크하는 기능만 추가하면 된다.

#### ▶ JavaScript

먼저 게시글 상세보기 페이지에서 수정하기 버튼이 클릭되면 게시글 수정 폼을 요청하는 자바스크립트 이벤트 처리 코드를 formcheck.js 파일의 \$(function() { } 코드 안쪽에 작성하자.

#### - src/main/resources/static/js/formcheck.js 에 추가

```

/* 게시글 상세보기에서 게시글 수정 폼 요청 처리
 * 아래와 같이 hidden 폼을 통해 post 방식으로 요청할 수 있다.
 */
$("#detailUpdate").on("click", function() {

    let pass = $("#pass").val();
    if(pass.length <= 0) {
        alert("게시글을 수정하려면 비밀번호를 입력해주세요");
        return false;
    }
}

```

```

    $("#rPass").val(pass);
    $("#checkForm").attr("action", "updateForm");
    $("#checkForm").attr("method", "post");
    $("#checkForm").submit();
});

```

## ▶ Persistence 계층 구현

MyBatis를 활용하여 no에 해당하는 게시글의 비밀번호를 읽어오는 매퍼 인터페이스와 매퍼 XML을 다음과 같이 작성하자.

### ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음은 참고해 BoardMapper.xml 파일에 DB 테이블에서 no에 해당하는 게시글의 비밀번호를 읽어오는 매퍼 구문을 추가하자.

#### - src/main/resources/mappers/BoardMapper.xml

```

<!--
    게시판 테이블에서 no에 해당하는 게시글의 비밀번호를 가져오는 매퍼 구문

    아래는 BoardMapper 인터페이스의 isPassCheck(int no) 메서드에서
    호출하는 매퍼 구문으로 파라미터 no는 기본형이므로 parameterType은 생략할 수
    있으며 SQL 쿼리에서 이 파라미터를 사용하려면 #{no}와 같이 지정하면 된다.

    반환 값이 자바 기본형 또는 String이면 resultType은 생략할 수 있다.
-->
<select id="isPassCheck">
    SELECT
        pass
    FROM springbbs
    WHERE no = #{no}
</select>

```

### ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 no에 해당하는 게시글의 비밀번호를 읽어와 반환하는 메서드를 다음과 같이 작성하자.

#### - com.springbootstudy.bbs.mapper.BoardMapper

```

// no에 해당하는 비밀번호를 DB 테이블에서 읽어와 반환하는 메서드
public String isPassCheck(int no);

```

## ▶ Business 계층 구현

게시글 수정 폼에 출력해야 할 데이터는 앞에서 구현한 게시글 상세보기의 getBoard(int no) 메서드를 그대로 사용하면 되고 게시글 상세보기에서 게시글 수정 폼 요청을 할 때 비밀번호를 체크하는 메서드만 추가하면 된다.

#### - com.springbootstudy.bbs.service.BoardService

```
/* 게시글 수정과 삭제 할 때 비밀번호가 맞는지 체크하는 메서드
 *
 * - 게시글의 비밀번호가 맞으면 : true를 반환
 * - 게시글의 비밀번호가 맞지 않으면 : false를 반환
 */
public boolean isPassCheck(int no, String pass) {
    log.info("BoardService: isPassCheck(int no, String pass)");
    boolean result = false;

    // BoardDao를 이용해 DB에서 no에 해당하는 비밀번호를 읽어온다.
    String dbPass = boardMapper.isPassCheck(no);

    if(dbPass.equals(pass)) {
        result = true;
    }

    // 비밀번호가 맞으면 true, 맞지 않으면 false가 반환된다.
    return result;
}
```

### ▶ Controller 클래스

BoardController 클래스에 게시글 수정 폼 요청을 처리하는 메서드를 추가하자.

#### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 수정 폼 요청을 처리하는 메서드
 * "/updateForm"으로 들어오는 HTTP POST 요청을 처리하는 메서드
 *
 * @GetMapping 애노테이션이 적용된 Controller 메서드의 파라미터에 PrintWriter와
 * HttpServletResponse를 지정했고 요청 파라미터를 받을 no와 pass도 지정했다.
 * 이렇게 Controller 메서드의 파라미터에 필요한 객체나 요청 파라미터 이름과 동일한
 * 이름의 파라미터를 지정하면 스프링이 자동으로 연결해 준다. 만약 요청 파라미터와
 * 메서드의 파라미터 이름이 다른 경우 Controller 메서드의 파라미터 앞에
 * @RequestParam("요청 파라미터 이름")을 사용해 요청 파라미터의 이름을 지정하면
 * 스프링이 데이터 형에 맞게 적절히 형 변환까지 해 준다. 형 변환을 할 수 없는 경우
 * 스프링은 400 에러를 발생 시킨다. 예를 들면 Controller 메서드의 파라미터가 정수형
 * 일 때 요청 파라미터의 값이 정수형으로 형 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 */
@PostMapping("/updateForm")
public String updateBoard(Model model,
```

```

    HttpServletResponse response, PrintWriter out,
    @RequestParam("no") int no, @RequestParam("pass") String pass) {

    // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
    boolean isPassCheck = boardService.isPassCheck(no, pass);
    if(! isPassCheck) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println(" alert('비밀번호가 맞지 않습니다.');");
        out.println(" history.back();");
        out.println("</script>");

        // null을 반환하면 위에서 스트림에 출력한 자바스크립트 코드가 응답된다.
        return null;
    }

    // 비밀번호가 맞으면 no에 해당하는 게시글 정보를 모델에 담아 수정 폼으로 이동
    Board board = boardService.getBoard(no);
    model.addAttribute("board", board);
    return "views/updateForm";
}

```

## ▶ 게시글 수정 폼 View

- src/main/resources/templates/views/updateForm.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layouts/main_layout}">
<!--/* content */-->
<th:block layout:fragment="content">
    <div class="row my-5" id="global-content">
        <div class="col-10 offset-1">
            <div class="row text-center">
                <div class="col">
                    <h2 class="fs-3 fw-bold" th:text="#{bbs.update.title}">게시글 수정하기</h2>
                </div>
            </div>
        </div>
        <form name="updateForm" action="/update" id="updateForm"
              class="row g-3 border-primary" method="post">
            <input type="hidden" name="no" th:value="{board.no}">
            <div class="col-4 offset-md-2">
                <label for="writer" class="form-label">글쓴이</label>
                <input type="text" class="form-control" name="writer" id="writer"
                      th:value="{board.writer}" readonly>
            </div>
        </form>
    </div>
</th:block>

```



```

</div>
<div class="col-4 ">
  <label for="pass" class="form-label">비밀번호</label>
  <input type="password" class="form-control" name="pass" id="pass">
</div>
<div class="col-8 offset-md-2">
  <label for="title" class="form-label">제 목</label>
  <input type="text" class="form-control" name="title"
    id="title" th:value="${board.title}">
</div>
<div class="col-8 offset-md-2">
  <label for="content" class="form-label">내 용</label>
  <textarea class="form-control" name="content" id="content"
    rows="10">[[${board.content}]]</textarea>
</div>
<div class="col-8 offset-md-2 text-center mt-5">
<input type="submit" value="수정하기" class="btn btn-primary"/>
  &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
    onclick="location.href='boardList'" class="btn btn-primary"/>
</div>
</form>
</div>
</div>
</th:block>
</html>

```

여기까지 구현하고 프로젝트를 실행해서 게시글 상세보기에서 게시글 수정 폼을 요청할 때 비밀번호를 틀리게 요청하면 다음과 같이 경고 창이 뜨는데 한글이 깨져서 표시되는 것을 볼 수 있다.



이는 스프링 부트의 HTTP 요청과 응답의 기본 인코딩은 “ISO-8859-1”로 한글과 같은 문자는 처리할 수 없기 때문에 한글이 깨지게 된다. 이 문제는 application.properties 파일에 다음과 같이 인코딩 설정을 추가하면 해결할 수 있다.

- src/main/resources/application.properties 에 추가

# 한글 처리

server.servlet.encoding.charset=UTF-8

server.servlet.encoding.force=true

## ▶ JavaScript

게시글 수정 폼에 대한 유효성 검사 자바스크립트 코드를 formcheck.js 파일의 `$(function() { }` 코드 안쪽에 작성한다.

- src/main/resources/static/js/formcheck.js 에 추가

```
// 게시글 수정 폼 유효성 검사
$("#updateForm").on("submit", function() {
    if($("#writer").val().length <= 0) {
        alert("작성자가 입력되지 않았습니다.\n작성자를 입력해주세요");
        $("#writer").focus();
        return false;
    }
    if($("#title").val().length <= 0) {
        alert("제목이 입력되지 않았습니다.\n제목을 입력해주세요");
        $("#title").focus();
        return false;
    }
    if($("#pass").val().length <= 0) {
        alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
        $("#pass").focus();
        return false;
    }
    if($("#content").val().length <= 0) {
        alert("내용이 입력되지 않았습니다.\n내용을 입력해주세요");
        $("#content").focus();
        return false;
    }
});
```

## 2-6-2) 게시글 수정 요청처리

사용자가 게시글 수정 폼에서 수정한 게시글을 서버에서 받아 수정하는 요청은 먼저 사용자가 입력한 비밀번호와 기존 게시글의 비밀번호가 일치하는지 검사한 후 비밀번호가 일치할 때만 게시글이 수정되도록 구현할 것이다. 비밀번호를 체크하는 기능은 앞에서 구현한 `isPassCheck()` 메서드를 그대로 사용할 것이다. 그리고 게시글 수정 요청은 수정된 게시글을 DB에서 수정한 후 게시글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다.

## ▶ Persistence 계층 구현

MyBatis를 활용하여 게시글 정보를 DB 테이블에서 수정하는 매퍼 인터페이스와 매퍼 XML을 다음과 같이 작성하자.

## ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음은 참고해 BoardMapper.xml 파일에 no에 해당하는 게시글을 DB 테이블에서 수정하는 매퍼 구문을 추가하자.

### - src/main/resources/mappers/BoardMapper.xml

```
<!--
    게시판 테이블에서 no에 해당하는 게시글을 수정하는 매퍼 구문

    아래는 DAO 클래스의 updateBoard(Board board) 메서드에서
    사용하는 매퍼 구문으로 parameterType을 Board 타입으로 지정했다.
    parameterType에 모델 클래스 타입을 지정하는 경우 VALUES()에
    지정하는 값은 getter 메서드를 지정하는 것이 아니라 클래스의 프로퍼티
    (인스턴스 변수)를 #{ }로 감싸서 지정하면 MyBatis가 알아서 처리해 준다.
-->
<update id="updateBoard" parameterType="Board">
    UPDATE springbbs
    SET title = #{title}, content = #{content},
        reg_date = SYSDATE()
    WHERE no = #{no}
</update>
```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 no에 해당하는 게시글을 DB 테이블에서 수정하는 메서드를 다음과 같이 작성하자.

### - com.springbootstudy.bbs.mapper.BoardMapper

```
// 수정된 게시글을 Board 객체로 받아서 DB 테이블에서 수정하는 메서드
public void updateBoard(Board board);
```

## ▶ Business 계층 구현

BoardService 클래스에 BoardMapper를 이용해 DB 테이블에서 게시글 정보를 수정하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.service.BoardService

```
// 게시글을 수정하는 메서드
public void updateBoard(Board board) {
    log.info("BoardService: updateBoard(Board board)");
    boardMapper.updateBoard(board);
}
```

## ▶ Controller 클래스

BoardController 클래스에 게시글 수정 폼에서 들어오는 게시글 수정 요청을 처리하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 수정 폼에서 들어오는 게시글 수정 요청을 처리하는 메서드
 * "/update"로 들어오는 HTTP POST 요청을 처리하는 메서드
 *
 * @PostMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
 * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
 * Board 객체를 지정했다.
 *
 * 스프링은 폼으로부터 전달된 파라미터를 객체로 처리 할 수 있는 아래와 같은
 * 방법을 제공하고 있다. 아래와 같이 요청 파라미터를 전달받을 때 사용하는
 * 객체를 커맨드 객체라고 부르며 이 커맨드 객체는 자바빈 규약에 따라 프로퍼티에
 * 대한 setter를 제공하도록 작성해야 한다. 그리고 파라미터 이름이 커맨드 객체의
 * 프로퍼티와 동일하도록 폼 컨트롤의 name 속성을 지정해야 한다.
 *
 * HTTP 요청을 맵핑하는 애노테이션(@PostMapping, @GetMapping 등)이
 * 적용된 컨트롤러의 메서드에 커맨드 객체를 파라미터로 지정하면 커맨드 객체의
 * 프로퍼티와 동일한 이름을 가진 요청 파라미터의 데이터를 스프링이 자동으로
 * 연결해 준다. 이때 스프링은 자바빈 규약에 따라 적절한 setter 메서드를 사용해
 * 값을 설정한다.
 *
 * 커맨드 객체의 프로퍼티와 일치하는 파라미터 이름이 없다면 기본 값으로 설정된다.
 * 또한 프로퍼티의 데이터 형에 맞게 적절히 형 변환 해 준다. 형 변환을 할 수 없는
 * 경우 스프링은 400 에러를 발생 시킨다. 예를 들면 프로퍼티가 정수형 일 때 매칭
 * 되는 값이 정수로 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 */
@PostMapping("/update")
public String updateBoard(Board board,
    HttpServletResponse response, PrintWriter out) {

    // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
    boolean isPassCheck = boardService.isPassCheck(board.getNo(), board.getPass());
    if(! isPassCheck) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println(" alert('비밀번호가 맞지 않습니다.');"");
        out.println(" history.back();"");
        out.println("</script>");

        // null을 반환하면 위에서 스트림에 출력한 자바스크립트 코드가 응답된다.
        return null;
    }
}
```

```

/* 비밀번호가 맞으면 DB 테이블에서 no에 해당하는
 * 게시글 정보를 수정하고 게시글 리스트로 리다이렉트
 */
boardService.updateBoard(board);
return "redirect:boardList";
}

```

## 2-7) 게시글 삭제하기

게시글 삭제 기능은 게시글 상세보기에서 삭제하기 버튼을 클릭하면 먼저 비밀번호를 검사해 비밀번호가 일치해야 게시글을 삭제할 수 있도록 구현할 것이다. 그리고 게시글 삭제 요청은 게시글을 DB에서 삭제한 후 게시글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다.

### ▶ JavaScript

게시글 상세보기 페이지에서 삭제하기 버튼이 클릭되면 게시글 삭제를 요청하는 자바스크립트 이벤트 처리 코드를 formcheck.js 파일의 \$(function() { } 코드 안쪽에 작성한다.

#### - src/main/resources/static/js/formcheck.js 에 추가

```

/* 게시글 상세보기에서 게시글 삭제 요청 처리
 * 아래와 같이 hidden 폼을 통해 post 방식으로 처리 할 수 있다.
 */
$("#detailDelete").on("click", function() {

    let pass = $("#pass").val();
    if(pass.length <= 0) {
        alert("게시글을 삭제하려면 비밀번호를 입력해주세요");
        return false;
    }

    $("#rPass").val(pass);
    $("#checkForm").attr("action", "delete");
    $("#checkForm").attr("method", "post");
    $("#checkForm").submit();
});

```

### ▶ Persistence 계층 구현

MyBatis를 활용하여 게시글 정보를 DB 테이블에서 삭제하는 매퍼 인터페이스와 매퍼 XML을 다음과 같이 작성하자.

## ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음에 참고해 BoardMapper.xml 파일에 DB 테이블에서 no에 해당하는 게시글을 삭제하는 매퍼 구문을 추가하자.

### - src/main/resources/mappers/BoardMapper.xml

```
<!--
    게시판 테이블에서 no에 해당하는 게시글을 삭제하는 매퍼 구문

    아래는 BoardMapper 인터페이스의 deleteBoard(int no) 메서드에서
    사용하는 매퍼 구문으로 parameterType이 int 형이므로 생략했다.
-->
<delete id="deleteBoard">
    DELETE FROM springbbs
    WHERE no = #{no}
</delete>
```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

매퍼 인터페이스에 no에 해당하는 게시글을 DB 테이블에서 삭제하는 메서드를 다음과 같이 작성하자.

### - com.springbootstudy.bbs.mapper.BoardMapper

```
//no에 해당 하는 게시글을 DB 테이블에서 삭제하는 메서드
public void deleteBoard(int no);
```

## ▶ Business 계층 구현

BoardService 클래스에 사용자가 삭제를 요청한 게시글 번호를 받아 그 번호에 해당하는 게시글을 BoardMapper를 이용해 DB 테이블에서 삭제하는 메서드를 추가하자.

### - com.springbootstudy.bbs.service.BoardService

```
// no에 해당하는 게시글을 삭제하는 메서드
public void deleteBoard(int no) {
    log.info("BoardService: deleteBoard(int no)");
    boardMapper.deleteBoard(no);
}
```

## ▶ Controller 클래스

BoardController 클래스에 게시글 삭제 요청을 처리하는 메서드를 추가한다.

### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 삭제 요청을 처리 메서드
 * "/delete"로 들어오는 HTTP POST 요청을 처리하는 메서드
```

```

    /**/
    @PostMapping("/delete")
    public String deleteBoard(
        HttpServletResponse response, PrintWriter out,
        @RequestParam("no") int no, @RequestParam("pass") String pass) {

        // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
        boolean isPassCheck = boardService.isPassCheck(no, pass);
        if(! isPassCheck) {
            response.setContentType("text/html; charset=utf-8");
            out.println("<script>");
            out.println(" alert('비밀번호가 맞지 않습니다.');");
            out.println(" history.back();");
            out.println("</script>");

            return null;
        }

        /* 비밀번호가 맞으면 DB 테이블에서 no에 해당하는
        * 게시글을 삭제하고 게시글 리스트로 리다이렉트
        */
        boardService.deleteBoard(no);
        return "redirect:boardList";
    }

```

## 2. 게시글 페이징 기능 구현

앞에서 구현한 게시판은 MyBatis를 이용해 CRUD 기능만 제공하는 단순한 형태의 게시판이다. 이 게시판은 게시글 리스트에서 페이징 처리가 되어 있지 않아서 게시글의 수가 늘어날수록 한 페이지에 길게 늘어지기 때문에 깔끔하지 못한 느낌이 든다. 그래서 게시글 리스트 페이지에서 한 페이지에 게시글을 몇 개씩 출력할지를 정해서 그 개수만큼씩 한 페이지에 출력되도록 페이징 처리 기능을 구현할 것이다.

### 1) 프로젝트 생성 및 환경설정

이번 예제는 앞에서 구현한 springbootstudy-bbs01 프로젝트에 새로운 기능을 추가하는 것으로 이 프로젝트를 import하여 프로젝트 이름만 springbootclass-bbs02로 변경하고 페이징 처리에 필요한 부분만 수정할 것이므로 프로젝트를 새로 생성할 필요는 없다. 프로젝트 생성에 대한 부분은 앞의 예제에서 설명한 주석을 참고하길 바란다.

이번 예제는 앞에서 구현한 게시판에 페이징 기능을 추가하는 것이므로 설정 파일과 소스가 거의 동일하다. 그러므로 꼭 필요한 부분과 새로 추가되거나 수정되는 부분만 교안에 작성할 것이다.

#### 1-1) 프로젝트 생성

- 실습용 프로젝트 : springbootclass-bbs02
- 완성 프로젝트 : springbootstudy-bbs02

#### 1-2) 의존 라이브러리 설정

이번에 구현할 페이징 처리 기능은 이번 프로젝트에서 사용한 라이브러리 그대로 사용하면 되기 때문에 build.gradle 파일에 별도의 추가적인 설정은 필요 없다.



## 2) 게시글 페이징 기능 구현

### 2-1) 게시글 리스트 보기

이번에 구현할 게시글 리스트는 앞에서 구현한 게시글 리스트에 페이징 처리를 구현하는 것이므로 Domain 클래스는 앞에서 사용한 Board 클래스를 그대로 사용하기 때문에 교안에는 생략했다.

#### ▶ Persistence 계층 구현

페이징 기능은 한 페이지에 출력할 게시글의 개수를 정해서 정해진 개수만큼 한 페이지에 일정하게 출력하는 기능으로 우리는 한 페이지에 10개씩 출력할 것이다. 그러므로 게시글 리스트 요청이 들어오며 현재 페이지가 몇 번째 페이지인지 판단해서 해당 페이지에 출력할 게시글 10개를 DB 테이블에서 읽어와 출력해야 한다. 또한 페이징 기능을 구현하려면 전체 페이지 수를 알아야 하는데 이는 전체 게시글의 수를 가지고 계산하면 되기 때문에 DB 테이블에 저장된 전체 게시글의 수를 읽어오는 기능도 구현해야 한다.

#### ▣ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음을 참고해 BoardMapper.xml 파일에 한 페이지에 출력할 게시글 리스트를 테이블로부터 읽어오는 boardList 매퍼 구문을 다음과 같이 수정하고 페이징 처리를 위해 전체 게시글의 수를 카운트해서 가져오는 getBoardCount 매퍼 구문을 추가하자.

##### - src/main/resources/mappers/BoardMapper.xml

<!--

한 페이지 출력할 게시글 리스트를 가져오는 매퍼 구문

테이블의 컬럼명은 일반적으로 언더스코어 표기법("\_")을 사용하는 경우가 많고 자바 클래스의 인스턴스 멤버는 카멜표기법을 사용한다.

application.properties 파일에 MyBatis 설정을 할 때 다음과 같이 설정 했기 때문에 별도 설정 없이 컬럼 데이터를 잘 읽어올 수 있다.

mybatis.configuration.map-underscore-to-camel-case=true

만약 위와 같이 설정할 수 없고 테이블의 컬럼명과 도메인 클래스의 프로퍼티 이름이 다른 경우 아래와 같이 SELECT 쿼리에 별칭을 사용해 도메인 클래스의 프로퍼티 이름과 동일하게 맞춰야 한다. 그렇지 않으면 오류는 발생하지 않지만 해당 컬럼의 데이터를 읽어 올 수 없다. 이외에 ResultMap을 사용해 컬럼명과 다른 이름을 가진 도메인 클래스의 프로퍼티를 연결할 수도 있다.

SELECT read\_count AS readCount FROM springbbs

resultType에 Board를 지정하고 BoardMapper 인터페이스에서 메서드를 정의할 때 반환 타입을 List<Board>로 지정하면 MyBatis가 쿼리를 실행한 결과 집합에서 한 행의 데이터를 Board 객체로 만들어 List에 담아서 반환되도록 처리해 준다.

Oracle에서는 페이징 처리를 위해 의사컬럼인 ROWNUM을 사용했지만 MySQL은 검색된 데이터에서 특정 행 번호부터 지정한 개수만큼 행을 읽어오는 LIMIT 명령을 제공하고 있다. LIMIT의 첫 번째 매개변수에 가져올 데이터의 시작 행을 지정하고 두 번째 매개변수에 가져올 데이터의 개수를 지정하면 된다.

BoardMapper 인터페이스에서 현재 페이지에 해당하는 게시글 리스트를 조회할 아래 맵핑 구문을 호출할 때 @Param("파라미터 이름") 애노테이션을 사용해 파라미터 이름을 "startRow"와 "num"으로 설정하여 맵핑 구문을 호출하면 Map 객체에 담겨서 이 맵핑 구문으로 전달된다. 그러므로 parameterType="map"으로 설정하고 SQL 쿼리에서 사용할 때는 아래 LIMIT 명령 다음에 파라미터를 지정한 것 처럼 Map의 키로 지정한 #{startRow}와 #{num}을 지정하면 된다.

-->

```
<select id="boardList" parameterType="map" resultType="Board">
```

```
SELECT
```

```
    no,
```

```
    title,
```

```
    writer,
```

```
    content,
```

```
    reg_date AS regDate,
```

```
    read_count AS readCount,
```

```
    pass,
```

```
    file1
```

```
FROM springbbs
```

```
ORDER BY no DESC
```

```
LIMIT #{startRow}, #{num}
```

```
</select>
```

```
<!--
```

```
    전체 게시글 수를 반환하는 맵핑 구문
```

SQL 쿼리 결과가 int형이므로 resultType은 생략했다.

-->

```
<select id="getBoardCount">
```

```
SELECT
```

```
    COUNT(no)
```

```
FROM springbbs
```

```
</select>
```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 한 페이지에 출력할 게시글 리스트를 DB 테이블에서 읽어와 반환하는 boardList() 메서드를 수정하고 페이징 처리를 위해 전체 게시글의 수를 카운트해서 가져와 반환하는 getBoardCount() 메서드를 추가하자.

- com.springbootstudy.bbs.mapper.BoardMapper

```

/* 한 페이지에 해당하는 게시글 리스트를 DB 테이블에서 읽어와 반환하는 메서드
 *
 * 매퍼 XML의 맵핑 구문을 호출하면서 전달해야할 파라미터가 여러 개일 때
 * 다음과 같이 @Param("파라미터 이름") 애노테이션을 사용해 맵핑 구문에서
 * 사용할 파라미터 이름을 지정하면 파라미터 이름을 키로 지정해 Map 객체에
 * 저장되고 맵핑 구문으로 전달된다.
 */
public List<Board> boardList(
    @Param("startRow") int startRow, @Param("num") int num);

// DB 테이블에 등록된 전체 게시글 수를 읽어와 반환하는 메서드
public int getBoardCount();

```

## ▶ Business 계층 구현

BoardService 클래스에 다음 코드를 참고해 한 페이지에 출력할 게시글의 수와 페이지 이동을 위해서 필요한 한 페이지에 표시할 페이지 링크(페이지 그룹)를 저장할 상수를 정의하자. 그리고 한 페이지에 출력할 게시글 리스트를 BoardMapper를 통해서 DB 테이블에서 읽어와 반환하는 boardList 메서드를 다음과 같이 수정하자.

- com.springbootstudy.bbs.service.BoardService

... 중 략 ...

```

// 한 페이지에 출력할 게시글의 수를 상수로 선언
private static final int PAGE_SIZE = 10;

/* 한 페이지에 출력할 페이지 그룹의 수를 상수로 선언
 * [이전] 1 2 3 4 5 6 7 8 9 10 [다음]
 */
private static final int PAGE_GROUP = 10;

// 전체 게시글을 읽어와 반환하는 메서드
public Map<String, Object> boardList(int pageNum) {
    log.info("BoardService: boardList(int pageNum)");

    // 요청 파라미터의 pageNum을 현재 페이지로 설정
    int currentPage = pageNum;

    /* 현재 페이지에 해당하는 게시글 리스트의 첫 번째 행의 값을 계산
     *
     * MySQL에서 검색된 게시글 리스트의 row에 대한 index는 0부터 시작한다.
     * 현재 페이지가 1일 경우 startRow는 0, 2페이지일 경우 startRow는 10이 된다.
     *
     * 예를 들어 3페이지에 해당하는 게시글 리스트를 가져 온다면 한 페이지에

```

```

* 출력할 게시글 리스트의 수가 10개로 지정되어 있으므로 startRow는 20이 된다.
* 즉 아래의 공식에 의해 startRow(20) = (3 - 1) * 10;
* 1페이지 startRow = 0, 2 페이지 startRow = 10이 된다.
**/
int startRow = (currentPage - 1) * PAGE_SIZE;

// BoardMapper를 이용해 전체 게시글 수를 가져온다.
int listCount = boardMapper.getBoardCount();

// 현재 페이지에 해당하는 게시글 리스트를 BoardMapper를 이용해 DB에서 읽어온다.
List<Board> boardList = boardMapper.boardList(startRow, PAGE_SIZE);

/* 페이지 그룹 이동 처리를 위해 전체 페이지 수를 계산 한다.
* [이전] 11 12 13... 또는 ... 8 9 10 [다음]과 같은 페이지 이동 처리
*
* 전체 페이지 = 전체 게시글 수 / 한 페이지에 표시할 게시글 수가 되는데
* 이 계산식에서 나머지가 존재하면 전체 페이지 수는 전체 페이지 + 1이 된다.
**/
int pageCount =
    listCount / PAGE_SIZE + (listCount % PAGE_SIZE == 0 ? 0 : 1);

/* 페이지 그룹 처리를 위해 페이지 그룹별 시작 페이지와 마지막 페이지를 계산
*
* 첫 번째 페이지 그룹에서 페이지 리스트는 1 ~ 10이 되므로 currentPage가
* 1 ~ 10 사이에 있으면 startPage는 1이 되고 11 ~ 20 사이면 11이 된다.
* 페이지 그룹 별 시작 페이지 : 1, 11, 21, 31...
*
* 정수형 연산의 특징을 이용해 startPage를 아래와 같이 구할 수 있다.
* 아래 연산식으로 계산된 결과를 보면 현재 그룹의 마지막 페이지일 경우
* startPage가 다음 그룹의 시작 페이지가 나오게 되므로 삼항 연산자를
* 사용해 현재 페이지가 속한 그룹의 startPage가 되도록 조정 하였다.
* 즉 currentPage가 10일 경우 다음 페이지 그룹의 시작 페이지가 되므로
* 삼항 연산자를 사용하여 PAGE_GROUP으로 나눈 나머지가 0이면
* PAGE_GROUP을 차감하여 현재 페이지 그룹의 시작 페이지가 되도록 하였다.
**/
int startPage = (currentPage / PAGE_GROUP) * PAGE_GROUP + 1
    - (currentPage % PAGE_GROUP == 0 ? PAGE_GROUP : 0);

// 현재 페이지 그룹의 마지막 페이지 : 10, 20, 30...
int endPage = startPage + PAGE_GROUP - 1;

/* 위의 식에서 endPage를 구하게 되면 endPage는 항상 PAGE_GROUP의
* 크기만큼 증가(10, 20, 30 ...) 되므로 맨 마지막 페이지 그룹의 endPage가
* 정확하지 못할 경우가 발생하게 된다. 다시 말해 전체 페이지가 53페이지라고
* 가정하면 위의 식에서 계산된 endPage는 60 페이지가 되지만 실제로

```

```

    * 60페이지는 존재하지 않는 페이지이므로 문제가 발생하게 된다.
    * 그래서 맨 마지막 페이지에 대한 보정이 필요하여 아래와 같이 endPage와
    * pageCount를 비교하여 현재 페이지 그룹에서 endPage가 pageCount 보다
    * 크다면 pageCount를 endPage로 지정 하였다. 즉 현재 페이지 그룹이
    * 마지막 페이지 그룹이면 endPage는 전체 페이지 수가 되도록 지정한 것이다.
    **/
    if(endPage > pageCount) {
        endPage = pageCount;
    }

    /* View 페이지에서 필요한 데이터를 Map에 저장한다.
    * 현재 페이지, 전체 페이지 수, 페이지 그룹의 시작 페이지와 마지막 페이지
    * 게시글 리스트의 수, 한 페이지에 출력할 게시글 리스트의 데이터를 Map에
    * 저장해 컨트롤러로 전달한다.
    **/
    Map<String, Object> modelMap = new HashMap<String, Object>();

    modelMap.put("bList", boardList);
    modelMap.put("pageCount", pageCount);
    modelMap.put("startPage", startPage);
    modelMap.put("endPage", endPage);
    modelMap.put("currentPage", currentPage);
    modelMap.put("listCount", listCount);
    modelMap.put("pageGroup", PAGE_GROUP);

    return modelMap;
}

... 중 략 ...

}

```

## ▶ Controller 클래스

BoardController 클래스에 게시글 리스트 요청을 처리하는 메서드를 아래와 같이 수정하자.

- com.springbootstudy.bbs.controller.BoardController

... 중 략 ...

```

/* 게시글 리스트 요청을 처리하는 메서드
* "/", "/boardList" 로 들어오는 HTTP GET 요청을 처리하는 메서드
*
* 스프링은 컨트롤러에서 모델에 데이터를 담을 수 있는 다양한 방법을 제공하며
* 아래와 같이 파라미터에 Model을 지정하는 방식이 많이 사용된다.

```

```

* @RequestMapping 애노테이션이 적용된 메서드의 파라미터에 Model
* 을 지정하면 스프링이 이 메서드를 호출하면서 Model 타입의 객체를 넘겨준다.
* 우리는 Model을 받아 이 객체에 결과 데이터를 담기만 하면 뷰에서 사용할 수 있다.
*
* @RequestParam 애노테이션을 이용해 pageNum이라는 요청 파라미터를 받도록 하였다.
* 아래에서 pageNum이라는 요청 파라미터가 없을 경우 required=false를 지정해 필수
* 조건을 주지 않았고 기본 값을 defaultValue="1"로 지정해 메서드의 파라미터인
* pageNum으로 받을 수 있도록 하였다. defaultValue="1"이 메서드의 파라미터인
* pageNum에 바인딩될 때 스프링이 int 형으로 형 변환하여 바인딩 시켜준다.
**/
@GetMapping("/{", "/boardList"})
public String boardList(Model model,
    @RequestParam(value="pageNum", required=false,
    defaultValue="1") int pageNum) {

    // Service 클래스를 이용해 게시글 리스트를 가져온다.
    Map<String, Object> modelMap = boardService.boardList(pageNum);

    /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
    * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
    */
    model.addAllAttributes(modelMap);

    /* application.properties 파일에서 Thymeleaf 설정을 별도로 지정하지
    * 않았기 때문에 다음과 같이 기본 설정이 적용되어 templates/ 폴더를
    * 기준으로 뷰를 찾아서 포워드 되고 제어가 뷰 페이지로 이동한다.
    *
    * spring.thymeleaf.prefix=classpath:/templates/
    * spring.thymeleaf.suffix=.html
    */
    // 페이지 모듈화로 content 페이지가 "/templates/views" 폴더로 이동함
    return "views/boardList";
}

```

... 중 략 ...

## ▶ 게시글 리스트 View

게시글 리스트를 보여주는 페이지에서 게시글 상세보기로 넘어가는 링크에 페이징 처리와 관련해서 pageNum이라는 요청 파라미터가 서버로 전송될 수 있도록 링크를 수정해야 한다. 왜냐하면 사용자가 3페이지 있는 게시글의 상세정보를 클릭해 상세보기 페이지로 이동한 후 다시 게시글 리스트로 돌아올 때 이전에 있었던 3페이지로 되돌아올 수 있는 서비스를 제공하기 위해서 게시글의 현재 페이지를 서버에 알려줘야 서버에서 처리할 수 있기 때문이다.

- src/main/resources/templates/views/boardList.html

.. 중 략 ..

```
<tbody>
  <th:block th:if="{not #lists.isEmpty(bList)}">
    <tr th:block th:each="board, status: ${bList}">
      <td>[[ ${board.no} ]]</td>
      <td><a
        th:href="@{boardDetail(no=${board.no},
pageNum=${currentPage})}"
        class="text-decoration-none      link-text-dark">[[      ${board.title}
]]</a></td>

      <td th:text="{ board.writer }"></td>
      <td>[[${ #dates.format(board.regDate, 'yyyy-MM-dd') }]]</td>
      <td th:text="{ board.readCount }"></td>
    </tr>
  </th:block>
  <th:block th:unless="{not #lists.isEmpty(bList)}">
    <tr>
      <td colspan="5">게시글이 존재하지 않음</td>
    </tr>
  </th:block>
</tbody>
</table>
</div>
</div>
<div class="row">
  <div class="col">
    <nav aria-label="Page navigation">
      <ul class="pagination justify-content-center">
        <!--/*
        현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
        이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작
        페이지에서 1을 빼면 이전 페이지 그룹의 endPage로 이동할 수 있다.
        */-->
        <li class="page-item" th:if="{ startPage > pageGroup }">
          <a class="page-link" th:href="@{|?pageNum=${startPage - 1}|">Pre</a>
        </li>
        <!--/*
        현재 페이지 그룹의 startPage부터 endPage까지 반복하면서 페이지
        번호를 출력하고 링크를 설정한다. 현재 페이지는 링크에서 제외 시킨다.
        */-->
        <li th:each="num : ${ #numbers.sequence(startPage, endPage) }"
          th:classappend="{num == currentPage} ? 'active'"
          class="page-item" aria-current="page">
```

```

        <th:block th:if="${num != currentPage}">
            <a                th:text="${num}"                th:href="@{?pageNum=${num}}"
class="page-link"></a>
        </th:block>
        <!--/* 현재 페이지는 링크에서 제외 시킨다.*/-->
        <th:block th:unless="${num != currentPage}">
            <span th:text="${num}" class="page-link"></span>
        </th:block>
    </li>
    <!--/*
현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
pageGroup을 더하여 링크를 설정하면 다음 페이지 그룹의 startPage로
이동할 수 있다.
*/-->
        <li class="page-item" th:if="${endPage < pageCount}">
            <a class="page-link" th:href="@{?pageNum=${startPage +
pageGroup}}">Next</a>
        </li>
    </ul>
</nav>
</div>
</div>

```

... 중 략 ...

## 2-2) 게시물 상세보기

게시글 리스트의 페이징 기능이 추가되면 페이징 처리와 관련해 몇 가지 더 추가적인 작업이 더 필요하다. 예를 들면 게시물 상세보기에서 게시물 리스트로 이동할 경우 사용자가 바로 이전에 있었던 게시물 리스트 페이지로 이동시켜야 사용자 편의성을 제공할 수 있을 것이다. 또한 게시물 수정 폼에서 게시물 리스트로 이동할 경우 바로 이전에 있었던 게시물 리스트 페이지로 이동시켜야 할 필요도 있다. 그리고 게시물 수정하기가 완료되었을 때 이전에 있었던 게시물 리스트 페이지로 리다이렉트 시켜야 할 필요도 있으며 게시글이 삭제되었을 때도 이전에 있었던 게시물 리스트 페이지로 리다이렉트 시켜야 할 필요도 있다. 이를 위해 요청을 받는 Controller에서 pageNum을 받아 모델에 담고 각각의 뷰 페이지에서 게시물 리스트로 이동하는 링크에 pageNum에 해당하는 요청 파라미터를 추가해야 한다.

이전의 프로젝트에서 게시판을 구현할 때 게시물 상세보기에서 처리해야 하는 게시물 읽은 횟수를 증가하는 기능을 구현하지 않았다. 이번 프로젝트에서는 게시물 상세보기 요청에서 게시물 읽은 횟수를 증가하는 기능도 추가할 것이다. 이 기능을 추가하기 위해 몇 가지 추가적인 작업이 필요하다. BoardService 클래스의 getBoard(int no)는 게시물 상세보기와 게시물 수정 폼 요청에서 사용되므로 이 메서드를 조금 수정해서 Controller에서 BoardService 클래스의 getBoard(int no) 메서드를 호출할 때 게시물 상세보기인지 아니면 게시물 수정 폼 요청인지를 구분해서 호출해야 할 필요가



있으므로 이 부분도 조금 수정이 필요하다.

## ▶ Persistence 계층 구현

기존에 게시글 하나의 정보를 읽어와 반환하는 getBoard() 메서드는 기존 메서드를 그대로 사용하면 되고 게시글 읽은 횟수를 증가시키는 기능만 추가로 구현하면 된다.

### ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음은 참고해 BoardMapper.xml 파일에 게시판 테이블에서 no에 해당하는 게시글 읽은 횟수를 증가시키는 매퍼 구문을 추가하자

- src/main/resources/mappers/BoardMapper.xml

```
<!--
    게시판 테이블에서 no에 해당하는 게시글 읽은 횟수를 증가시키는 매퍼 구문
-->
<update id="incrementReadCount">
    UPDATE springbbs
        SET read_count = read_count + 1
        WHERE no = #{no}
</update>
```

### ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 위에서 작성한 매퍼 구문을 이용해 게시판 테이블에서 no에 해당하는 게시글의 읽은 횟수를 증가시키는 incrementReadCount(int no) 메서드를 다음과 같이 추가하자.

- com.springbootstudy.bbs.mapper.BoardMapper

```
// no에 해당하는 게시글의 읽은 횟수를 DB 테이블에서 증가시키는 메서드
public void incrementReadCount(int no);
```

## ▶ Business 계층 구현

BoardService 클래스에 게시글 하나의 정보를 반환하는 getBoard(int no) 메서드를 상황에 따라서 게시글 읽은 횟수를 증가시키거나 또는 증가시키지 않고 no에 해당하는 게시글 정보를 반환하도록 수정해야 한다. 다시 말해서 게시글 상세보기 요청에서 사용하는 경우에는 게시글 읽은 횟수를 하나 증가시키고 그 외에는 게시글 읽은 횟수를 증가시키지 않고 no에 해당하는 게시글 정보를 반환하도록 작성하면 된다. 다음 코드를 참고해서 기존의 getBoard(int no) 메서드를 수정하자.

- com.springbootstudy.bbs.service.BoardService

```
/* no에 해당하는 게시글을 읽어와 반환하는 메서드
 *
 * isCount == true : 게시 상세보기 요청, false : 그 외 요청임
```

```

    **/
    public Board getBoard(int no, boolean isCount) {
        log.info("BoardService: getBoard(int no, boolean isCount)");

        // 게시글 상세보기 요청만 게시글 읽은 횟수를 증가 시킨다
        if(isCount) {
            boardMapper.incrementReadCount(no);
        }
        return boardMapper.getBoard(no);
    }
}

```

## ▶ Controller 클래스

BoardController 클래스에서 게시글 상세보기 HTTP 요청을 처리하는 메서드를 페이징 기능이 추가될 수 있도록 다음과 같이 수정하자.

### - com.springbootstudy.bbs.controller.BoardController

```

/* 게시글 상세보기 요청 처리 메서드
 * "/boardDetail"로 들어오는 HTTP GET 요청을 처리하는 메서드
 *
 * 페이징 기능 연동을 위해서 @RequestParam 애노테이션을 이용해 pageNum이라는
 * 요청 파라미터를 받도록 하였다. 아래에서 pageNum이라는 요청 파라미터가 없을
 * 경우 게시글 리스트의 첫 페이지로 보내기 위해서 defaultValue="1"로 지정해
 * 메서드의 파라미터인 pageNum으로 받을 수 있도록 하였다
 */
@GetMapping("/boardDetail")
public String getBoard(Model model, @RequestParam("no") int no,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum) {

    /* 게시글 상세보기는 게시글 조회에 해당하므로 no에 해당하는 게시글 정보를
     * 읽어오면서 두 번째 인수에 true를 지정해 게시글 읽은 횟수를 1 증가시킨다.
     */
    Board board = boardService.getBoard(no, true);

    // no에 해당하는 게시글 정보와 pageNum을 모델에 저장한다.
    model.addAttribute("board", board);
    model.addAttribute("pageNum", pageNum);

    return "views/boardDetail";
}

```

## ▶ 게시글 상세보기 View

게시글 상세보기에서 게시글 수정 폼으로 넘어가는 기능과 게시글 삭제 기능으로 연결되는 기능을 제공하고 있기 때문에 각각 연결되는 요청에 pageNum을 요청 파라미터로 보내야 한다. 왜냐하면

게시글 수정 폼에서 게시글 리스트로 넘어갈 때 사용자가 이전에 있었던 페이지로 이동할 수 있도록 해야 하고 게시글이 수정되거나 삭제되었을 때도 이전에 있었던 페이지로 이동할 수 있는 기능을 제공해야 한다. 그러므로 게시글 수정 폼 요청과 게시글 삭제 요청에 사용되는 checkForm 안에 pageNum이 요청 파라미터로 넘어갈 수 있도록 아래와 같이 히든 폼 컨트롤을 추가하고 목록보기 버튼에도 pageNum이 요청 파라미터로 넘어갈 수 있도록 링크를 수정하자.

- src/main/resources/templates/views/boardDetail.html

... 중 략 ...

```
<form name="checkForm" id="checkForm">
  <input type="hidden" name="no" id="no" th:value="${board.no}"/>
  <input type="hidden" name="pass" id="rPass" />
  <input type="hidden" name="pageNum" th:value="${pageNum}"/>
</form>
```

... 중 략 ...

```
<div class="row my-3">
  <div class="col text-center">
    <input class="btn btn-warning" type="button" id="detailUpdate" value="수정하기" />

    &nbsp;&nbsp;&nbsp;<input class="btn btn-danger" type="button" id="detailDelete"
value="삭제하기" />

    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
th:onclick="@{|location.href='boardList?pageNum=${pageNum}'}|"/>
  </div>
</div>
```

... 중 략 ...

## 2-3) 게시글 쓰기

이번 페이지징 처리에서는 게시글 쓰기 폼에서 게시글 리스트로 이동할 때 게시글 리스트의 첫 페이지로 이동하도록 할 것이다. 또한 사용자가 게시글 쓰기 폼에서 작성한 게시글을 저장하는 요청은 게시글을 DB에 저장한 후 게시글 리스트로 리다이렉트 될 때 사용자가 이전에 있었던 페이지로 보내는 것 보다는 새로운 게시글이 등록된 것을 확인할 수 있는 게시글 리스트의 첫 페이지로 리다이렉트 하는 것이 더 좋을 것 같다. 일반적으로 게시판에서 게시글 리스트는 항상 최신 글을 맨 위에 보여주도록 설계되기 때문에 사용자가 자신이 작성한 신규 게시글을 확인하려면 첫 페이지로 이동해야 하므로 앞에서와 마찬가지로 게시글 쓰기가 완료되면 게시글 리스트의 첫 페이지로 리다이렉트 시키면 된다.

페이지징 기능이 추가되더라도 게시글 쓰기는 앞에서 구현한 Persistence 계층, Business 계층과 Controller의 코드는 변경되지 않기 때문에 교안에서 생략했다.

## 2-4) 게시글 수정 폼 요청

이번 예제는 앞에서 구현한 게시글 수정 폼 요청 기능에 페이징 기능을 연동하기 위한 것이므로 게시글 수정 폼에서 게시글 리스트로 돌아갈 때 사용자가 이전에 있었던 페이지로 이동시키기 위해서 pageNum을 Controller 클래스에서 받아 모델에 담아 게시글 수정 폼에서 사용할 수 있도록 구현해야 하며 게시글 수정 폼에서 게시글을 수정한 후 수정 요청을 할 때도 pageNum을 요청 파라미터에 추가해야 한다.

게시글 수정 폼 요청은 앞에서 구현한 Persistence 계층과 Business 계층의 기능을 그대로 사용하면 되기 때문에 교안에서 생략되었고 Controller 클래스와 뷰 페이지만 수정하면 된다.

### ▶ Controller 클래스

BoardController 클래스의 게시글 수정 폼 요청을 처리하는 메서드를 페이징 기능 연동을 위해서 다음과 같이 수정하자.

#### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 수정 폼 요청을 처리하는 메서드
 * "/updateForm"으로 들어오는 HTTP POST 요청을 처리하는 메서드
 *
 * 페이징 기능 연동을 위해서 @RequestParam 애노테이션을 이용해 pageNum이라는
 * 요청 파라미터를 받도록 하였다. 아래에서 pageNum이라는 요청 파라미터가 없을
 * 경우 게시글 리스트의 첫 페이지로 보내기 위해서 defaultValue="1"로 지정해
 * 메서드의 파라미터인 pageNum으로 받을 수 있도록 하였다
 */
@PostMapping("/updateForm")
public String updateBoard(Model model,
    HttpServletResponse response, PrintWriter out,
    @RequestParam("no") int no, @RequestParam("pass") String pass,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum) {

    // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
    boolean isPassCheck = boardService.isPassCheck(no, pass);
    if(! isPassCheck) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println("alert('비밀번호가 맞지 않습니다.');
```

```

/* 수정 폼 요청은 게시글 조회가 아니므로 비밀번호가 맞으면 no에 해당하는
 * 게시글 정보를 읽어오면서 두 번째 인수로 false를 지정해 게시글 읽은 횟수를
 * 증가시키지 않는다.
 */
Board board = boardService.getBoard(no, false);

// no에 해당하는 게시글 정보와 pageNum을 모델에 저장한다.
model.addAttribute("board", board);
model.addAttribute("pageNum", pageNum);
return "views/updateForm";
}

```

## ▶ 게시글 수정 폼 View

게시글 수정 폼 페이지에서도 이 페이지로 들어올 때 받은 페이지 정보를 게시글 수정 요청을 하면서 서버로 보내줘야 서버에서 게시글 수정이 완료된 후 사용자가 이전에 있었던 게시글 리스트 페이지로 연결되도록 서비스 할 수 있다. 그러므로 게시글 수정 폼 안에 pageNum이 요청 파라미터로 넘어갈 수 있도록 아래와 같이 히든 폼 컨트롤을 추가하고 목록보기 버튼에도 pageNum이 요청 파라미터로 넘어갈 수 있도록 링크를 수정하자.

- src/main/resources/templates/views/updateForm.html

... 중 략 ...

```

<form name="updateForm" action="/update" id="updateForm"
class="row g-3 border-primary method="post">
    <input type="hidden" name="no" th:value="${board.no}">
    <input type="hidden" name="pageNum" th:value="${pageNum}"

```

... 중 략 ...

```

<div class="col-8 offset-md-2 text-center mt-5">
    <input type="submit" value="수정하기" class="btn btn-primary"/>
    &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
        th:onclick="@{|location.href='boardList?pageNum=${pageNum}'}|"
        class="btn btn-primary"/>
</div>

```

... 중 략 ...

## 2-5) 게시글 수정 요청

이번 예제는 앞에서 구현한 게시글 수정 요청 기능에 페이징 기능을 연동하기 위한 것이므로 게시

글 수정 요청을 처리한 후 리다이렉트할 때 pageNum을 파라미터로 추가해야 사용자가 이전에 있던 게시글 리스트 페이지로 이동시킬 수 있다.

게시글 수정 요청 또한 앞에서 구현한 Persistence 계층과 Business 계층의 기능이 모두 동일하므로 교안에서 생략되었고 Controller 클래스만 수정하면 된다.

## ▶ Controller 클래스

### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 수정 폼에서 들어오는 게시글 수정 요청을 처리하는 메서드
 * "/update"로 들어오는 HTTP POST 요청을 처리하는 메서드
 *
 * 페이징 기능 연동을 위해서 @RequestParam 애노테이션을 이용해 pageNum이라는
 * 요청 파라미터를 받도록 하였다. 아래에서 pageNum이라는 요청 파라미터가 없을
 * 경우 게시글 리스트의 첫 페이지로 보내기 위해서 defaultValue="1"로 지정해
 * 메서드의 파라미터인 pageNum으로 받을 수 있도록 하였다. 또한 리다이렉트할 때
 * pageNum을 파라미터로 보내기 위해서 RedirectAttributes 객체를 파라미터로 지정했다.
 */
@PostMapping("/update")
public String updateBoard(Board board, RedirectAttributes reAttrs,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum,
    HttpServletResponse response, PrintWriter out) {

    // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
    boolean isPassCheck = boardService.isPassCheck(board.getNo(), board.getPass());
    if(! isPassCheck) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println("alert('비밀번호가 맞지 않습니다.');
```

```

*
* Redirect 되는 경우 주소 끝에 파라미터를 지정해 GET방식의 파라미터로
* 전송할 수 있지만 스프링프레임워크가 지원하는 RedirectAttributes객체를
* 이용하면 한 번만 사용할 임시 데이터와 지속적으로 사용할 파라미터를
* 구분해 지정할 수 있다.
*
* 아래와 같이 RedirectAttributes의 addAttribute() 메서드를 사용해
* 지속적으로 사용할 파라미터를 지정하면 자동으로 주소 뒤에 파라미터로
* 추가되며 addFlashAttribute() 메서드를 사용해 파라미터로 지정하면
* 한 번만 사용할 수 있고 이후에는 주소 뒤에 파라미터로 추가되지 않는다.
* addAttribute() 메서드를 사용해 파라미터로 지정한 데이터는 페이지를
* 새로 고침해도 계속해서 주소 뒤에 파라미터로 남아있지만 addFlashAttribute()
* 메서드를 사용해 지정한 파라미터는 사라지기 때문에 1회성으로 필요한
* 데이터를 addFlashAttribute() 메서드를 사용해 지정하면 편리하다.
*
* 파라미터에 한글이 포함되는 경우 URLEncoder를 java.net 패키지의
* URLEncoder 클래스를 이용해 코드로 인코딩 처리를 해야 하지만
* application.properties에 인코딩 관련 설정이 되어 있기 때문에 별도로
* 처리할 필요가 없다.
**/
reAttrs.addAttribute("pageNum", pageNum);
reAttrs.addFlashAttribute("test1", "1회성 파라미터");
return "redirect:boardList";
}

```

## 2-6) 게시물 삭제 요청

이번 예제는 앞에서 구현한 게시물 삭제 기능에 페이징 기능을 연동하는 것이므로 게시물 삭제 요청을 처리한 후 리다이렉트할 때 pageNum을 파라미터로 추가해야 사용자가 이전에 있었던 게시물 리스트 페이지로 이동시킬 수 있다.

게시글 삭제 요청 또한 앞에서 구현한 Persistence 계층과 Business 계층의 기능이 모두 동일하므로 교안에서 생략되었고 Controller 클래스만 수정하면 된다.

### ▶ Controller 클래스

- com.springbootstudy.bbs.controller.BoardController

```

/* 게시물 삭제 요청을 처리 메서드
* "/delete"로 들어오는 HTTP POST 요청을 처리하는 메서드
**/
@PostMapping("/delete")
public String deleteBoard(RedirectAttributes reAttrs,
    HttpServletResponse response, PrintWriter out,
    @RequestParam("no") int no, @RequestParam("pass") String pass,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum) {

```

```

// 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
boolean isPassCheck = boardService.isPassCheck(no, pass);
if(! isPassCheck) {
    response.setContentType("text/html; charset=utf-8");
    out.println("<script>");
    out.println("alert('비밀번호가 맞지 않습니다.');"");
    out.println("history.back();"");
    out.println("</script>");

    return null;
}

// 비밀번호가 맞으면 DB 테이블에서 no에 해당하는 게시글을 삭제한다.
boardService.deleteBoard(no);

// RedirectAttributes를 이용해 리다이렉트 할 때 필요한 파라미터를 지정
reAttrs.addAttribute("pageNum", pageNum);

/* 게시글 삭제가 완료되면 게시글 리스트로 리다이렉트 시킨다.
 * 클라이언트 요청을 처리한 후 리다이렉트 해야 할 경우 아래와 같이 redirect:
 * 접두어를 붙여 뷰 이름을 반환하면 된다. 뷰 이름에 redirect 접두어가 붙으면
 * HttpServletResponse를 사용해서 지정한 경로로 Redirect 된다.
 */
return "redirect:boardList";
}

```



### 3. 게시물 검색 기능 구현

앞에서 게시물 리스트에 페이징 기능을 추가하는 방법에 대해 알아보았다.

이번에는 게시물 리스트에서 사용자가 특정 검색어에 해당하는 게시물을 검색하는 기능을 추가할 것이다. 게시판에 검색 기능을 구현하기 전에 이 기능을 제공하려면 애플리케이션이 어떻게 동작해야 하는지 먼저 생각해 보자.

게시물 검색 기능이 포함된 게시판에서 게시물 리스트는 검색이 아닌 경우와 특정 검색어로 검색한 경우에 따라서 게시물 리스트를 출력하는 부분이 다르기 때문에 서버에서 요청을 처리하는 부분도 각각에 맞게 처리해야 한다. 그래서 현재 요청이 게시물 리스트만 보여 달라는 것인지 아니면 검색 결과에 대한 리스트를 보여 달라는 것인지 먼저 판단하고 단순 게시물 리스트 요청인지 아니면 검색 리스트 요청인지에 따라서 다르게 처리하는 코드를 작성해야 한다.

#### 1) 프로젝트 생성 및 환경설정

이번 예제도 springbootstudy-bbs02 프로젝트를 복사해 springbootclass-bbs03 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 검색 기능을 추가할 것이다. 그러므로 프로젝트 생성과 설정 그리고 의존 라이브러리 설정 등에 대한 자세한 내용은 앞의 프로젝트에서 설명한 주석을 참고하길 바란다.

이번 예제도 앞에서 구현한 게시판에서 검색 기능을 추가하는 것이므로 설정파일과 소스 코드가 동일한 부분이 많으므로 꼭 필요한 부분과 새로 추가되거나 수정되는 부분만 교안에 작성할 것이다.

##### 1-1) 프로젝트 생성

- 실습용 프로젝트 : springbootclass-bbs03
- 완성 프로젝트 : springbootstudy-bbs03

##### 1-2) 의존 라이브러리 설정

이번에 구현할 검색 기능은 추가로 필요한 라이브러리가 없으므로 build.gradle 파일에 별도의 추가 설정이 필요 없다.

## 2) 게시글 검색 기능 구현

### 2-1) 게시글 리스트와 검색 리스트 요청

게시판에 검색 기능이 추가되면 게시글 리스트의 페이징 처리와 검색 리스트의 페이징 처리를 구분해서 처리해야 한다. 왜냐하면 게시글 리스트는 페이징 처리와 관련해서 pageNum이라는 파라미터만 하나만 있으면 되지만 검색 리스트는 pageNum과 검색 타입(제목, 작성자, 내용)과 검색어 정보가 같이 있어야 검색어에 해당하는 검색 리스트를 구성하고 그에 맞는 페이징 처리를 할 수 있기 때문이다.

게시글 리스트는 앞에서와 마찬가지로 전체 게시글의 수를 이용해 페이지 수를 계산하고 페이징 처리에 필요한 데이터를 만들면 되겠지만 검색 리스트는 사용자가 제목, 작성자, 내용 중 하나를 선택하고 검색어를 입력해 검색하는 경우에 따라서 사용자가 입력한 검색어가 포함된 게시글의 수를 기준으로 페이징 처리에 필요한 데이터를 만들어야 한다. 위와 같이 동작하도록 구현하기 위해서 게시글 리스트와 검색 리스트 처리를 별도의 메서드로 분리해서 처리할 수도 있지만 우리는 하나의 메서드에서 게시글 리스트 요청인지, 검색 리스트 요청인지를 구분해서 각각에 맞게 구현할 것이다. 이렇게 하나의 메서드에서 두 가지 상황에 따라서 처리하기 위해서는 각 상황에 맞게 동적으로 변화되는 동적 쿼리를 작성해야 한다. 우리는 MyBatis를 사용하므로 BoardMapper.xml 매퍼 파일에서 동적 쿼리를 작성하는 방법에 대해서 알아볼 것이다.

이번 프로젝트는 게시글 페이징 기능에 게시글 검색 리스트 기능을 추가로 구현하는 것이므로 DTO 클래스인 Board 클래스는 앞의 프로젝트와 동일하므로 교안에는 생략했다.

#### ▶ Persistence 계층 구현

일반 게시글 리스트와 마찬가지로 검색 리스트도 페이징 기능은 한 페이지에 출력할 게시글의 개수를 정해서 정해진 개수만큼 일정하게 출력하는 기능으로 우리는 한 페이지에 10개씩 출력할 것이다.

#### ▣ 게시판 관련 SQL을 분리한 매퍼 XML 파일

다음은 참고해 BoardMapper.xml 파일에 게시글 리스트와 검색 리스트에 따라서 한 페이지에 출력할 게시글 리스트를 게시판 테이블로부터 읽어오는 boardList 매퍼 구문을 수정하고 각각의 상황에 따라서 페이징 처리를 위해 게시판 테이블로부터 게시글의 수를 가져오는 getBoardCount 매퍼 구문도 수정하자. 이 두 매퍼 구문에서 사용하는 SQL 쿼리는 상황에 따라서 일반 게시글 리스트 또는 검색 리스트에 해당하는 결과가 조회되도록 동적쿼리로 작성해야 한다.

#### - src/main/resources/mappers/BoardMapper.xml

<!--

한 페이지에 해당하는 게시글 리스트, 검색리스트를 가져오는 매퍼 구문

BoardMapper 인터페이스에서 아래 매퍼 구문을 호출할 때 @Param("파라미터 이름") 애노테이션을 사용해 파라미터 이름을 "startRow", "num", "type", "keyword" 등으로 설정하여 매퍼 구문을 호출하면 MyBatis가 Map 객체에 담아서 이 매퍼 구문으로 전달해 준다. 그러므로 parameterType="map"으로 설정하고 SQL 쿼리에서 데이터로 지정할 때는 Map의 키를 #{startRow}, #{num}, #{keyword}와 같이 지정하면 된다. 그리고 MyBatis의 태그 안에서 사용할 경우 아래 if문과 같이 변수 이름을 그대로 사용하면 된다.

게시글 리스트와 검색 리스트 요청에 따라서 각각을 처리하기 위해 동적으로 변화되는 SQL 쿼리를 작성해야 한다. 검색 리스트도 제목, 작성자, 내용을 기준으로 검색어가 포함된 검색 리스트만 구성하기 위해서 동적으로 변화되는 SQL이 필요하므로 각 상황에 맞게 조건절이 동적으로 생성되도록 작성했다.

동적 SQL 참고 :

<http://www.mybatis.org/mybatis-3/ko/dynamic-sql.html>

-->

```
<select id="boardList" parameterType="map" resultType="Board">
```

```
SELECT
```

```
    no,
```

```
    title,
```

```
    writer,
```

```
    content,
```

```
    reg_date AS regDate,
```

```
    read_count AS readCount,
```

```
    pass,
```

```
    file1
```

```
FROM springbbs
```

```
<!--
```

WHERE 절을 동적으로 생성하는 요소

where 요소는 하위 요소(조건)에서 생성한 내용이 있으면 WHERE 절을 추가하고 그렇지 않으면 무시한다. 또한 WHERE 다음에 바로 AND나 OR가 나타나면 그 또한 무시하여 AND나 OR를 지워준다.

아래는 where 요소의 하위 요소인 if 요소가 true가 되면 SQL문에 WHERE 절을 추가해 주기 때문에 WHERE를 생략하고 다음 문장부터 추가하였다.

SQL 파라미터로 사용할 데이터가 여러 개라서 BoardMapper 인터페이스에서 @Param("파라미터 이름") 애노테이션을 사용해 Map 객체로 전달되도록 하였다. 이렇게 전달된 파라미터를 조건절에서 사용할 때는 #{ }를 사용하지 않고 파라미터 이름만 지정해야 하며 문자열을 사용할 때는 쌍 따옴표("")나 홑 따옴표('')로 감싸줘야 한다.

만약 WHERE 절에 수치 데이터의 대소 비교를 하는 부등호가 들어가는 경우라면 이 부등호는 XML에서 태그로 사용되는 문자이기 때문에 문제가 발생한다.

이런 경우에는 아래와 같이 CDATA Section으로 묶어 주면 된다.

CDATA는 Character DATA라는 뜻으로 CDATA Section 안에 있는 데이터는 해석(Parsing)하지 말고 문자 데이터 그대로 처리하라는 의미이다. 이렇게 SQL 쿼리문에 XML 태그와 같은 문자를 사용해야 할 경우 아래와 같이 CDATA Section 안에 기술되도록 작성해야 한다.

```
<![CDATA[
```

```
    price <= #{price}
```

```
]]>
```

```
-->
<where>
  <if test="type == 'title'">
    title LIKE CONCAT('%', #{keyword}, '%')
  </if>
  <if test="type == 'writer'">
    writer LIKE CONCAT('%', #{keyword}, '%')
  </if>
  <if test="type == 'content'">
    content LIKE CONCAT('%', #{keyword}, '%')
  </if>
</where>
ORDER BY no DESC
LIMIT #{startRow}, #{num}
</select>
```

<!--

전체 게시글 수와 검색 리스트에 대한 게시글 수를 반환하는 맵핑 구문

게시글 리스트와 검색 리스트 요청에 따라서 페이징 처리에 사용하는 게시글 수를 반환해야 하기 때문에 아래 상황에 맞게 동적으로 변화되는 SQL 쿼리를 작성해야 한다. 게시글 리스트 요청일 때는 전체 게시글의 수를 반환되도록 하고 검색 리스트일 때는 제목, 작성자, 내용을 기준으로 검색어가 포함된 게시글 수가 반환될 수 있도록 구현해야 한다.

게시글 리스트 요청 : type == null, keyword == null  
 게시글 검색 요청 : type == title 일 때 제목으로 검색,  
                   type == writer 일 때 글쓴이로 검색,  
                   type == content 일 때 게시글 내용으로 검색

동적 SQL 참고 :

<http://www.mybatis.org/mybatis-3/ko/dynamic-sql.html>

BoardMapper 인터페이스에서 이 맵핑 구문을 호출하면서 @Param("파라미터 이름") 애노테이션을 사용해 파라미터 이름을 "type", "keyword" 등으로 설정해 호출하면 MyBatis가 파라미터 이름을 키로하여 Map 객체로 만들어 넘겨준다.

아래에서 parameterType="map"을 생략해도 map의 key 값으로 지정한 이름과 #{}에 지정한 이름이 같은 파라미터에 데이터가 바인딩 된다.

SQL 쿼리 결과가 int형이므로 resultType은 생략이 가능하다.

```
-->
<select id="getBoardCount" parameterType="map" resultType="int">
  SELECT
    COUNT(no)
```

FROM springbbs

<!--

WHERE 절을 동적으로 생성하는 요소

where 요소는 하위 요소(조건)에서 생성한 내용이 있으면 WHERE 절을 추가하고 그렇지 않으면 무시한다. 또한 WHERE 다음에 바로 AND나 OR가 나타나면 그 또한 무시하여 AND나 OR를 지워준다.

아래는 where 요소의 하위 요소인 if 요소가 true가 되면 SQL문에 WHERE 절을 추가해 주기 때문에 WHERE를 생략하고 다음 문장부터 추가하였다.

조건절에서 DAO로 부터 받은 파라미터를 지정할 때는 #{ }를 사용하지 않고 파라미터 이름만 지정해야 하며 문자열을 사용할 때는 쌍 따옴표("")나 홑 따옴표(')로 감싸줘야 한다.

SQL 파라미터로 사용할 데이터가 여러 개라 BoardDao의 getBoardCount() 메서드에서 이 맵핑 구문을 호출할 때 HashMap에 담아 전달하였다. 맵핑 구문에서 HashMap의 데이터를 조건절 이나 SQL 파라미터로 지정할 때는 HashMap에 저장할 때 사용한 키의 이름을 지정하면 된다.

만약 WHERE 절에 수치 데이터의 대소 비교를 하는 부등호가 들어가는 경우라면 이 부등호는 XML에서 태그로 사용되는 문자이기 때문에 문제가 발생한다.

이럴 경우에는 아래와 같이 CDATA Section으로 묶어 주면 된다.

CDATA는 Character DATA라는 뜻으로 CDATA Section 안에 있는 데이터는 해석(Parsing)하지 말고 문자 데이터 그대로 처리하라는 의미이다. 이렇게 SQL 쿼리문에 XML 태그와 같은 문자를 사용해야 할 경우 아래와 같이 CDATA Section 안에 기술되도록 작성해야 한다.

```
<![CDATA[
    price <= #{price}
]]>
-->
<where>
    <if test="type == 'title'">
        title LIKE CONCAT('%', #{keyword}, '%')
    </if>
    <if test="type == 'writer'">
        writer LIKE CONCAT('%', #{keyword}, '%')
    </if>
    <if test="type == 'content'">
        content LIKE CONCAT('%', #{keyword}, '%')
    </if>
</where>
</select>
```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 한 페이지에 출력할 게시글 리스트 또는 검색 리스트를 게시판 테이블에서 읽어와 반환하는 boardList() 메서드와 페이징 처리를 위해 전체 게시글의 수 또는 검색 게시글의 수를 카운트해서 가져와 반환하는 getBoardCount() 메서드를 수정하자.

#### - com.springbootstudy.bbs.mapper.BoardMapper

```
/* 한 페이지에 출력할 게시글 리스트 또는 검색 리스트를 게시판 테이블에서
 * 읽어와 반환하는 메서드
 *
 * 매퍼 XML의 맵핑 구문을 호출하면서 전달해야 할 파라미터가 여러 개일 때
 * 다음과 같이 @Param("파라미터 이름") 애노테이션을 사용해 맵핑 구문에서
 * 사용할 파라미터 이름을 지정하면 파라미터 이름을 키로 지정해 Map 객체에
 * 저장되고 맵핑 구문으로 전달된다.
 */
public List<Board> boardList(
    @Param("startRow") int startRow, @Param("num") int num,
    @Param("type") String type, @Param("keyword") String keyword);

// DB 테이블에서 전체 게시글 수 또는 검색 게시글 수를 읽어와 반환하는 메서드
public int getBoardCount(
    @Param("type") String type, @Param("keyword") String keyword);
```

### ▶ Business 계층 구현

BoardService 클래스에서 한 페이지에 출력할 게시글 리스트 또는 검색 리스트와 페이징 처리를 위해서 필요한 데이터를 Map 객체로 반환하는 boardList 메서드를 다음과 같이 수정하자.

#### - com.springbootstudy.bbs.service.BoardService

```
/* 한 페이지에 출력할 게시글 리스트 또는 검색 리스트와
 * 페이징 처리에 필요한 데이터를 Map 객체로 반환하는 메서드
 */
public Map<String, Object> boardList(int pageNum, String type, String keyword) {
    log.info("boardList(int pageNum, String type, String keyword)");

    // 요청 파라미터의 pageNum을 현재 페이지로 설정
    int currentPage = pageNum;

    /* 현재 페이지에 해당하는 게시글 리스트의 첫 번째 행의 값을 계산
     *
     * MySQL에서 검색된 게시글 리스트의 row에 대한 index는 0부터 시작한다.
     * 현재 페이지가 1일 경우 startRow는 0, 2페이지일 경우 startRow는 10이 된다.
     *
     * 예를 들어 3페이지에 해당하는 게시글 리스트를 가져 온다면 한 페이지에
     * 출력할 게시글 리스트의 수가 10개로 지정되어 있으므로 startRow는 20이 된다.
     * 즉 아래의 공식에 의해 startRow(20) = (3 - 1) * 10;
     * 1페이지 startRow = 0, 2 페이지 startRow = 10이 된다.
     */
}
```

```

    **/
    int startRow = (currentPage - 1) * PAGE_SIZE;
    log.info("startRow : " + startRow);

    /* 요청 파라미터에서 type 또는 keyword가 비어 있으면 일반 게시글 리스트를
     * 요청한 것으로 간주하여 false 값을 갖게 한다. Controller에서 type 또는
     * keyword의 요청 파라미터가 없으면 기본값을 "null"로 지정했기 때문에
     * 아래와 같이 체크했다.
     */
    boolean searchOption = (type.equals("null")
        || keyword.equals("null")) ? false : true;

    /* 맵핑 구문 안에서 동적쿼리를 사용해 type이 없으면 전체 게시글의
     * 수를 반환하고, type이 존재하면 제목이나 내용 또는 작성자를 기준으로
     * 검색어가 포함된 게시글 수를 반환한다.
     */
    int listCount = boardMapper.getBoardCount(type, keyword);

    /* 현재 페이지에 해당하는 게시글 리스트를 BoardMapper를 이용해 DB에서 읽어온다.
     *
     */
    List<Board> boardList = boardMapper.boardList(startRow, PAGE_SIZE, type, keyword);

    /* 페이지 그룹 이동 처리를 위해 전체 페이지 수를 계산 한다.
     * [이전] 11 12 13... 또는 ... 8 9 10 [다음]과 같은 페이지 이동 처리
     *
     * 전체 페이지 = 전체 게시글 수 / 한 페이지에 표시할 게시글 수가 되는데
     * 이 계산식에서 나머지가 존재하면 전체 페이지 수는 전체 페이지 + 1이 된다.
     */
    int pageCount =
        listCount / PAGE_SIZE + (listCount % PAGE_SIZE == 0 ? 0 : 1);

    /* 페이지 그룹 처리를 위해 페이지 그룹별 시작 페이지와 마지막 페이지를 계산
     *
     * 첫 번째 페이지 그룹에서 페이지 리스트는 1 ~ 10이 되므로 currentPage가
     * 1 ~ 10 사이에 있으면 startPage는 1이 되고 11 ~ 20 사이면 11이 된다.
     * 페이지 그룹 별 시작 페이지 : 1, 11, 21, 31...
     *
     * 정수형 연산의 특징을 이용해 startPage를 아래와 같이 구할 수 있다.
     * 아래 연산식으로 계산된 결과를 보면 현재 그룹의 마지막 페이지일 경우
     * startPage가 다음 그룹의 시작 페이지가 나오게 되므로 삼항 연산자를
     * 사용해 현재 페이지가 속한 그룹의 startPage가 되도록 조정 하였다.
     * 즉 currentPage가 10일 경우 다음 페이지 그룹의 시작 페이지가 되므로
     * 삼항 연산자를 사용하여 PAGE_GROUP으로 나눈 나머지가 0이면

```

```

    * PAGE_GROUP을 차감하여 현재 페이지 그룹의 시작 페이지가 되도록 하였다.
    **/
    int startPage = (currentPage / PAGE_GROUP) * PAGE_GROUP + 1
        - (currentPage % PAGE_GROUP == 0 ? PAGE_GROUP : 0);

    // 현재 페이지 그룹의 마지막 페이지 : 10, 20, 30...
    int endPage = startPage + PAGE_GROUP - 1;

    /* 위의 식에서 endPage를 구하게 되면 endPage는 항상 PAGE_GROUP의
    * 크기만큼 증가(10, 20, 30 ...) 되므로 맨 마지막 페이지 그룹의 endPage가
    * 정확하지 못할 경우가 발생하게 된다. 다시 말해 전체 페이지가 53페이지라고
    * 가정하면 위의 식에서 계산된 endPage는 60 페이지가 되지만 실제로
    * 60페이지는 존재하지 않는 페이지이므로 문제가 발생하게 된다.
    * 그래서 맨 마지막 페이지에 대한 보정이 필요하여 아래와 같이 endPage와
    * pageCount를 비교하여 현재 페이지 그룹에서 endPage가 pageCount 보다
    * 크다면 pageCount를 endPage로 지정 하였다. 즉 현재 페이지 그룹이
    * 마지막 페이지 그룹이면 endPage는 전체 페이지 수가 되도록 지정한 것이다.
    **/
    if(endPage > pageCount) {
        endPage = pageCount;
    }

    /* View 페이지에서 필요한 데이터를 Map에 저장한다.
    * 현재 페이지, 전체 페이지 수, 페이지 그룹의 시작 페이지와 마지막 페이지
    * 게시글 리스트의 수, 한 페이지에 출력할 게시글 리스트의 데이터를 Map에
    * 저장해 컨트롤러로 전달한다.
    **/
    Map<String, Object> modelMap = new HashMap<String, Object>();

    modelMap.put("bList", boardList);
    modelMap.put("pageCount", pageCount);
    modelMap.put("startPage", startPage);
    modelMap.put("endPage", endPage);
    modelMap.put("currentPage", currentPage);
    modelMap.put("listCount", listCount);
    modelMap.put("pageGroup", PAGE_GROUP);
    modelMap.put("searchOption", searchOption);

    // 검색 요청이면 type과 keyword를 모델에 저장한다.
    if(searchOption) {
        modelMap.put("type", type);
        modelMap.put("keyword", keyword);
    }

    return modelMap;

```



```
}
```

## ▶ Controller 클래스

BoardController 클래스에서 게시글 리스트 요청을 처리하는 메서드를 검색 리스트 요청과 일반 게시글 리스트 모두를 처리할 수 있도록 다음과 같이 수정하자.

### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 리스트 요청을 처리하는 메서드
 * "/", "/boardList" 로 들어오는 HTTP GET 요청을 처리하는 메서드
 *
 * @RequestParam 애노테이션을 이용해 pageNum이라는 요청 파라미터를 받도록 하였다.
 * 아래에서 pageNum이라는 요청 파라미터가 없을 경우 required=false를 지정해 필수
 * 조건을 주지 않았고 기본 값을 defaultValue="1"로 지정해 메서드의 파라미터인
 * pageNum으로 받을 수 있도록 하였다. defaultValue="1"이 메서드의 파라미터인
 * pageNum에 바인딩될 때 스프링이 int 형으로 형 변환하여 바인딩 시켜준다.또한 검색
 * 타입과 검색어를 받기 위해 type과 keyword를 메서드의 파라미터로 지정하고 요청
 * 파라미터가 없을 경우를 대비해 required=false를 지정해 필수 조건을 주지 않았고
 * 기본 값을 defaultValue="null"로 지정해 type과 keyword로 받을 수 있도록 하였다.
 */
@GetMapping({"/", "/boardList"})
public String boardList(Model model,
    @RequestParam(value="pageNum", required=false,
        defaultValue="1") int pageNum,
    @RequestParam(value="type", required=false,
        defaultValue="null") String type,
    @RequestParam(value="keyword", required=false,
        defaultValue="null") String keyword) {

    // Service 클래스를 이용해 게시글 리스트를 가져온다.
    Map<String, Object> modelMap = boardService.boardList(pageNum, type, keyword);

    /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
     * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
     */
    model.addAllAttributes(modelMap);

    return "views/boardList";
}
```

## ▶ 게시글 리스트 View

게시글 리스트를 보여주는 페이지에서 일반 게시글 리스트와 검색 리스트를 같이 처리하기 때문에 게시글 상세보기로 넘어갈 때 일반 게시글 리스트라면 현재 페이지 정보인 pageNum을 요청 파라미터로 보내야 하고 검색 리스트를 보여주는 페이지라면 검색 리스트의 현재 페이지 정보인

pageNum과 검색어는 무엇이고 어떤 타입(제목, 작성자, 글 내용)으로 검색된 리스트인지를 알려주기 위해서 keyword와 type이라는 파라미터를 같이 보내줘야 한다. 왜냐하면 사용자가 일반 검색 리스트에서 게시글 상세페이지로 이동한 후 다시 게시글 리스트로 돌아올 경우 이전에 있었던 게시글 리스트의 페이지로 돌아올 수 있어야 하고 검색 리스트라면 이전에 있었던 검색 리스트로 돌아올 수 있도록 서비스하기 위해서 필요하다.

- src/main/resources/templates/views/boardList.html

... 중 략 ...

```
<form name="searchForm" id="searchForm" action="boardList"
      class="row justify-content-center my-3">
  <div class="col-auto">
    <select name="type" class="form-select">
      <option value="title">제목</option>
      <option value="writer">작성자</option>
      <option value="content">내용</option>
    </select>
  </div>
  <div class="col-4">
    <input type="text" name="keyword" class="form-control"/>
  </div>
  <div class="col-auto">
    <input type="submit" value="검 색" class="btn btn-primary"/>
  </div>
</form>
<!--/* 검색 요청일 경우 표시 */-->
<th:block th:if="${searchOption}">
  <div class="row my-3">
    <div class="col text-center">
      "[[${ keyword }]]" 검색 결과
    </div>
  </div>
  <div class="row my-3">
    <div class="col-6">
      <a th:href="@{boardList}" class="btn btn-outline-success">리스트</a>
    </div>
    <div class="col-6 text-end">
      <a th:href="@{addBoard}" class="btn btn-outline-success">글쓰기</a>
    </div>
  </div>
</th:block>
<div class="row my-3" th:if="${not searchOption}">
  <div class="col text-end">
    <a th:href="@{addBoard}" class="btn btn-outline-success">글쓰기</a>
```

```

</div>
</div>
<div class="row my-3">
  <div class="col">
    <table class="table">
      <thead>
        <tr class="table-dark">
          <th>NO</th>
          <th>제목</th>
          <th>작성자</th>
          <th>작성일</th>
          <th>조회수</th>
        </tr>
      </thead>
      <tbody>
        <!--/*
          검색 요청이면서 검색된 리스트가 존재할 경우
          게시글 상세보기로 링크를 적용할 때 type과 keyword
          파라미터를 적용해 링크를 설정한다.
        */-->
        <th:block th:if="${searchOption and not #lists.isEmpty(bList)}">
          <tr th:block th:each="board, status: ${bList}">
            <td>[[ ${board.no} ]]</td>
            <td><a
                                  th:href="@{boardDetail(no=${board.no},
pageNum=${currentPage},
                                  type=${type}, keyword=${keyword})}"
                                  class="text-decoration-none link-text-dark">[[ ${board.title}
]]</a></td>
            <td th:text="${ board.writer }"></td>
            <td>[[ ${ #dates.format(board.regDate, 'yyyy-MM-dd') } ]]</td>
            <td th:text="${ board.readCount }"></td>
          </tr>
        </th:block>
        <!--/*
          검색 요청이면서 검색된 리스트가 존재하지 않을 경우
        */-->
        <th:block th:if="${searchOption and #lists.isEmpty(bList)}">
          <tr>
            <td colspan="5">[[ ${keyword} ]]가 포함된 게시글이 존재하지 않음</td>
          </tr>
        </th:block>
        <!--/*
          일반 게시글 리스트 요청이면서 게시글 리스트가 존재할 경우
          게시글 상세보기로 링크를 적용할 때 type과 keyword
          파라미터는 제외하고 링크를 설정한다.

```

```

* /-->
<th:block th:if="${not searchOption and not #lists.isEmpty(bList)}">
    <tr th:block th:each="board, status: ${bList}">
        <td>[[ ${board.no} ]]</td>
        <td><a
                                th:href="@{boardDetail(no=${board.no},
pageNum=${currentPage})}"
                                class="text-decoration-none    link-text-dark">[[      ${board.title}
]]</a></td>

        <td th:text="${ board.writer }"></td>
        <td>[[ ${ #dates.format(board.regDate, 'yyyy-MM-dd') } ]]</td>
        <td th:text="${ board.readCount }"></td>
    </tr>
</th:block>
<!--/*
    일반 게시글 리스트 요청이면서 게시글 리스트가 존재하지 않을 경우
* /-->
<th:block th:if="${not searchOption and #lists.isEmpty(bList)}">
    <tr>
        <td colspan="5">게시글이 존재하지 않음</td>
    </tr>
</th:block>
</tbody>
</table>
</div>
</div>
<div class="row">
    <div class="col">
        <!--/*
            검색 요청 이면서 검색된 게시글이 존재할 경우 페이지네이션
            게시글 상세보기로 링크를 적용할 때 type과 keyword
            파라미터를 적용해 링크를 설정한다.
        * /-->
        <nav th:if="${searchOption and not #lists.isEmpty(bList)}" aria-label="Page
navigation">
            <ul class="pagination justify-content-center">
                <!--/*
                    현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
                    이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작
                    페이지에서 1을 빼면 이전 페이지 그룹의 endPage로 이동할 수 있다.
                * /-->
                <li class="page-item" th:if="${ startPage > pageGroup }">
                    <a class="page-link" th:href="@{/?pageNum=${startPage - 1}
                        &type=${type}&keyword=${keyword}}">Pre</a>
                </li>
                <!--/*

```

현재 페이지 그룹의 startPage부터 endPage까지 반복하면서 페이지 번호를 출력하고 링크를 설정한다. 현재 페이지는 링크에서 제외 시킨다.

```
*/-->
<li th:each="num : ${ #numbers.sequence(startPage, endPage) }"
    th:classappend="${num == currentPage} ? 'active'"
    class="page-item" aria-current="page">
    <th:block th:if="${num != currentPage}">
        <a th:text="${num}" class="page-link" th:href="@{|?pageNum=${num}
            &type=${type}&keyword=${keyword}}|" ></a>
    </th:block>
    <!--/* 현재 페이지는 링크에서 제외 시킨다.*/-->
    <th:block th:unless="${num != currentPage}">
        <span th:text="${num}" class="page-link"></span>
    </th:block>
</li>
<!--/*
```

현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에 pageGroup을 더하여 링크를 설정하면 다음 페이지 그룹의 startPage로 이동할 수 있다.

```
*/-->
<li class="page-item" th:if="${endPage < pageCount}">
    <a class="page-link" th:href="@{|?pageNum=${startPage + pageGroup}
        &type=${type}&keyword=${keyword}}|">Next</a>
</li>
</ul>
</nav>
<!--/*
```

일반 게시글 요청 이면서 게시글이 존재할 경우 페이지네이션  
게시글 상세보기로 링크를 적용할 때 type과 keyword  
파라미터는 제외하고 링크를 설정한다.

```
*/-->
<nav th:if="${not searchOption and not #lists.isEmpty(bList)}" aria-label="Page
navigation">
```

```
<ul class="pagination justify-content-center">
    <!--/*
    현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
    이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작
    페이지에서 1을 빼면 이전 페이지 그룹의 endPage로 이동할 수 있다.
    */-->
    <li class="page-item" th:if="${ startPage > pageGroup }">
        <a class="page-link" th:href="@{|?pageNum=${startPage - 1}}|">Pre</a>
    </li>
    <!--/*
```

현재 페이지 그룹의 startPage부터 endPage까지 반복하면서 페이지

```

번호를 출력하고 링크를 설정한다. 현재 페이지는 링크에서 제외 시킨다.
*/-->
<li th:each="num : ${ #numbers.sequence(startPage, endPage) }"
    th:classappend="${num == currentPage} ? 'active'"
    class="page-item" aria-current="page">
    <th:block th:if="${num != currentPage}">
        <a
            th:text="${num}"
            th:href="@{?pageNum=${num}}}"
class="page-link"></a>
    </th:block>
    <!--/* 현재 페이지는 링크에서 제외 시킨다.*/-->
    <th:block th:unless="${num != currentPage}">
        <span th:text="${num}" class="page-link"></span>
    </th:block>
</li>
<!--/*
현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
pageGroup을 더하여 링크를 설정하면 다음 페이지 그룹의 startPage로
이동할 수 있다.
*/-->
<li class="page-item" th:if="${endPage < pageCount}">
    <a class="page-link" th:href="@{?pageNum=${startPage +
pageGroup}}">Next</a>
</li>
</ul>
</nav>
</div>
</div>

```

... 중 략 ...

## 2-2) 게시물 상세보기 요청

게시글 검색 기능이 추가되면 일반 게시물 리스트와 검색 리스트의 페이징 처리와 관련해 몇 가지 더 추가적인 작업이 더 필요하다. 예를 들면 게시물 상세보기에서 게시물 리스트로 이동할 경우 사용자가 바로 이전에 있었던 게시물 리스트와 검색 리스트로 이동할 수 있도록 구분해서 처리해야 한다. 또한 게시물 수정 폼에서 게시물 리스트나 검색 리스트로 이동할 경우 바로 이전에 있었던 게시물 리스트나 검색 리스트로 구분해서 이동시켜야 사용자 편의성을 제공할 수 있다. 그리고 게시물 수정하기가 완료되었을 때도 이전에 있었던 게시물 리스트나 검색 리스트로 구분해서 리다이렉트 시켜야 할 필요도 있으며 게시물이 삭제되었을 때도 수정과 마찬가지로 이전에 있었던 게시물 리스트나 검색 리스트로 리다이렉트 시켜야 할 필요도 있다. 이를 위해 요청을 받는 Controller에서 pageNum과 type 그리고 keyword를 받아 모델에 담고 각각의 뷰 페이지에서 게시물 리스트나 검색 리스트로 이동하는 링크에 pageNum과 type 그리고 keyword를 요청 파라미터에 추가해 줘야

한다.

게시글 상세보기는 앞에서 구현한 Persistence 계층과 Business 계층의 기능을 그대로 사용하면 되기 때문에 교안에서 생략되었고 Controller 클래스와 뷰 페이지만 수정하면 된다.

## ▶ Controller 클래스

BoardController 클래스에서 게시글 상세보기 HTTP 요청을 처리하는 메서드를 검색 기능과 페이지 기능이 추가될 수 있도록 다음과 같이 수정하자.

### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 상세보기 요청 처리 메서드
 * "/boardDetail"로 들어오는 HTTP GET 요청을 처리하는 메서드
 */
@GetMapping("/boardDetail")
public String getBoard(Model model, @RequestParam("no") int no,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum,
    @RequestParam(value="type", defaultValue="null") String type,
    @RequestParam(value="keyword", defaultValue="null") String keyword) {

    /* 요청 파라미터에서 type 또는 keyword가 비어 있으면 일반 게시글 리스트를
     * 요청한 것으로 간주하여 false 값을 갖게 한다. Controller에서 type 또는
     * keyword의 요청 파라미터가 없으면 기본값을 "null"로 지정했기 때문에
     * 아래와 같이 체크했다.
     */
    boolean searchOption = (type.equals("null")
        || keyword.equals("null")) ? false : true;

    /* 게시글 상세보기는 게시글 조회에 해당하므로 no에 해당하는 게시글 정보를
     * 읽어오면서 두 번째 인수에 true를 지정해 게시글 읽은 횟수를 1 증가시킨다.
     */
    Board board = boardService.getBoard(no, true);

    // no에 해당하는 게시글 정보와 pageNum, searchOption을 모델에 저장한다.
    model.addAttribute("board", board);
    model.addAttribute("pageNum", pageNum);
    model.addAttribute("searchOption", searchOption);

    // 검색 요청이면 type과 keyword를 모델에 저장한다.
    if(searchOption) {
        model.addAttribute("type", type);
        model.addAttribute("keyword", keyword);
    }

    return "views/boardDetail";
}
```

## ▶ 게시글 상세보기 View

게시글 상세보기에서 게시글 수정 폼으로 넘어가는 기능과 게시글 삭제 기능으로 연결되는 기능을 제공하고 있기 때문에 일반 게시글 리스트 요청인 경우에는 각각 연결되는 요청에 pageNum만 요청 파라미터로 보내면 되지만 검색 요청인 경우에는 pageNum과 더불어 검색에 필요한 type과 keyword가 요청 파라미터로 같이 전송될 수 있도록 해야 한다. 왜냐하면 게시글 수정 폼에서 게시글 리스트로 넘어갈 때 사용자가 이전에 있었던 일반 게시글 리스트 또는 검색 리스트 페이지로 이동할 수 있도록 해야 하고 게시글이 수정되거나 삭제되었을 때도 이전에 있었던 페이지로 이동할 수 있도록 서비스를 제공하기 위해서 필요하다.

- src/main/resources/templates/views/boardDetail.html

... 중 략 ...

```
<form name="checkForm" id="checkForm">
  <input type="hidden" name="no" id="no" th:value="{board.no}"/>
  <input type="hidden" name="pass" id="rPass" />
  <input type="hidden" name="pageNum" th:value="{pageNum}"/>
<!--/*
  검색 리스트에서 들어온 요청일 경우 다시 keyword에 해당하는
  검색 리스트로 돌려보내기 위해서 아래의 파라미터가 필요하다.
*/-->
<th:block th:if="{searchOption}>
  <input type="hidden" name="type" th:value="{ type }" />
  <input type="hidden" name="keyword" th:value="{ keyword }" />
</th:block>
</form>
```

... 중 략 ...

```
<div class="row my-3">
  <div class="col text-center">
    <input class="btn btn-warning" type="button" id="detailUpdate" value="수정하기" />
    &nbsp;&nbsp;&nbsp;<input class="btn btn-danger" type="button" id="detailDelete"
value="삭제하기" />
  <!--/*
    일반 게시글 리스트에서 온 요청이면 일반 게시글 리스트로 돌려 보낸다.
  */-->
  <th:block th:unless="{searchOption}>
    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
      th:onclick="@{|location.href='boardList?pageNum={pageNum}'}"/>
  </th:block>
<!--/*
```



```

        검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
*/-->
<th:block th:if="{searchOption}">
    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
th:onclick="@{location.href='boardList?pageNum={pageNum}&type={type}&keyword={key
word}'}"/>
    </th:block>
</div>
</div>

... 중 략 ...

```

## 2-3) 게시물 쓰기

게시글 쓰기는 검색이나 페이징과 상관없이 게시물 쓰기 폼에서 게시물 리스트로 이동할 때 게시물 리스트의 첫 페이지로 이동하도록 할 것이다. 또한 사용자가 게시물 쓰기 폼에서 작성한 게시글을 저장하는 요청은 게시글을 DB에 저장한 후 게시물 리스트로 리다이렉트 될 때 사용자가 있었던 페이지로 보내는 것이 아니라 새로운 게시글이 등록된 것을 확인할 수 있는 첫 페이지로 리다이렉트 하면 된다. 일반적으로 게시판에서 게시물 리스트는 항상 최신 글을 맨 위에 보여주도록 설계되기 때문에 사용자가 자신이 작성한 신규 게시글을 확인하려면 첫 페이지로 이동해야 하므로 앞에서와 마찬가지로 게시물 쓰기가 완료되면 게시물 리스트의 첫 페이지로 리다이렉트 시키면 된다. 게시물 쓰기는 검색 또는 페이징 기능이 추가되더라도 게시물 쓰기는 앞에서 구현한 Persistence 계층, Business 계층과 Controller의 코드는 변경되지 않기 때문에 교안에서 생략했다.

## 2-4) 게시물 수정 폼 요청

이번 예제는 앞에서 구현한 페이징 기능에서 게시물 검색 기능을 추가하는 것이므로 게시물 리스트와 검색 리스트를 구분해서 처리해야 한다. 게시물 수정 폼에서 게시물 리스트와 검색 리스트로 돌아갈 때 사용자가 이전에 있었던 각각의 페이지로 이동시키기 위해서 pageNum과 type 그리고 keyword를 Controller 클래스에서 받아 모델에 담아 게시물 수정 폼에서 사용할 수 있도록 구현해야 하며 게시물 수정 폼에서 게시글을 수정한 후 수정 요청을 할 때도 pageNum과 type 그리고 keyword를 요청 파라미터에 추가해야 한다.

게시글 수정 폼 요청은 앞에서 구현한 Persistence 계층과 Business 계층의 기능을 그대로 사용하면 되기 때문에 교안에서 생략되었고 Controller 클래스와 뷰 페이지만 수정하면 된다.

### ▶ Controller 클래스

BoardController 클래스의 게시물 수정 폼 요청을 처리하는 메서드를 검색 기능을 구현하기 위해서 다음과 같이 수정하자.

- **com.springbootstudy.bbs.controller.BoardController**

```
/* 게시글 수정 폼 요청을 처리하는 메서드
 * "/updateForm"으로 들어오는 HTTP POST 요청을 처리하는 메서드
 */
@PostMapping("/updateForm")
public String updateBoard(Model model,
    HttpServletResponse response, PrintWriter out,
    @RequestParam("no") int no, @RequestParam("pass") String pass,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum,
    @RequestParam(value="type", defaultValue="null") String type,
    @RequestParam(value="keyword", defaultValue="null") String keyword) {

    // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
    boolean isPassCheck = boardService.isPassCheck(no, pass);
    if(! isPassCheck) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println(" alert('비밀번호가 맞지 않습니다.');");
        out.println(" history.back();");
        out.println("</script>");

        return null;
    }

    // 게시글 수정 폼 요청은 읽은 횟수를 증가시키지 않는다.
    Board board = boardService.getBoard(no, false);

    // 현재 요청이 검색 요청인지 여부를 판단하는 searchOption 설정
    boolean searchOption = (type.equals("null")
        || keyword.equals("null")) ? false : true;

    // no에 해당하는 게시글 정보와 pageNum, searchOption을 모델에 저장한다.
    model.addAttribute("board", board);
    model.addAttribute("pageNum", pageNum);
    model.addAttribute("searchOption", searchOption);

    // 검색 요청이면 type과 keyword를 모델에 저장한다.
    if(searchOption) {
        model.addAttribute("type", type);
        model.addAttribute("keyword", keyword);
    }
    return "views/updateForm";
}
```

## ▶ 게시물 수정 폼 View

게시글 수정 폼 페이지에서도 이 페이지로 들어올 때 받은 검색과 관련된 정보와 페이지 정보를 게시물 수정 요청을 하면서 서버로 보내줘야 서버에서 게시물 수정이 완료된 후 사용자가 이전에 있었던 게시물 리스트 페이지로 연결되도록 서비스 할 수 있다.

- src/main/resources/templates/views/updateForm.html

... 중 략 ...

```
<form name="updateForm" action="/update" id="updateForm"
class="row g-3 border-primary" method="post">
  <input type="hidden" name="no" th:value="${board.no}">
  <input type="hidden" name="pageNum" th:value="${pageNum}">
  <!--/*
    검색 리스트에서 들어온 요청일 경우 다시 keyword에 해당하는
    검색 리스트로 돌려보내기 위해서 아래의 파라미터가 필요하다.
  */-->
  <th:block th:if="${searchOption}">
    <input type="hidden" name="type" th:value="${ type }" />
    <input type="hidden" name="keyword" th:value="${ keyword }" />
  </th:block>
```

... 중 략 ...

```
<div class="col-8 offset-md-2 text-center mt-5">
  <input type="submit" value="수정하기" class="btn btn-primary"/>
  <!--/*
    일반 게시물 리스트에서 온 요청이면 일반 게시물 리스트로 돌려보낸다.
  */-->
  <th:block th:unless="${searchOption}">
    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
      th:onclick="@{|location.href='boardList?pageNum=${pageNum}'}"/>
  </th:block>
  <!--/*
    검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
  */-->
  <th:block th:if="${searchOption}">
    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
th:onclick="@{|location.href='boardList?pageNum=${pageNum}&type=${type}&keyword=${key
word}'}"/>
  </th:block>
</div>
</form>
```

... 중 략 ...

## 2-5) 게시글 수정 요청

이번 예제는 앞에서 구현한 페이징 기능에 게시글 검색 기능을 추가하는 것이므로 게시글 수정 요청을 처리한 후 게시글 리스트와 검색 리스트로 구분해 리다이렉트할 때 사용자가 이전에 있었던 각각의 페이지로 리다이렉트 될 수 있도록 pageNum과 type 그리고 keyword를 Controller 클래스에서 받아 요청 파라미터에 추가해야 한다. 그래야 사용자가 이전에 있었던 게시글 리스트나 검색 리스트로 이동시킬 수 있다.

게시글 수정 요청 또한 앞에서 구현한 Persistence 계층과 Business 계층의 기능이 모두 동일하므로 교안에서 생략되었고 Controller 클래스만 수정하면 된다.

### ▶ Controller 클래스

게시글 수정 폼에서 들어오는 게시글 수정 요청을 처리하는 메서드를 다음과 같이 수정하고 이 메서드 안에서 리다이렉트할 때 RedirectAttributes 객체를 이용해 pageNum과 type 그리고 keyword를 파라미터로 추가하는 부분도 다음과 같이 수정하자.

#### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 수정 폼에서 들어오는 게시글 수정 요청을 처리하는 메서드
 * "/update"로 들어오는 HTTP POST 요청을 처리하는 메서드
 */
@PostMapping("/update")
public String updateBoard(Board board, RedirectAttributes reAttrs,
    HttpServletResponse response, PrintWriter out,
    @RequestParam("no") int no, @RequestParam("pass") String pass,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum,
    @RequestParam(value="type", defaultValue="null") String type,
    @RequestParam(value="keyword", defaultValue="null") String keyword) {

    // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
    boolean isPassCheck = boardService.isPassCheck(board.getNo(), board.getPass());
    if(! isPassCheck) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println("alert('비밀번호가 맞지 않습니다.');"");
        out.println("history.back();"");
        out.println("</script>");

        return null;
    }

    // 비밀번호가 맞으면 DB 테이블에서 no에 해당하는 게시글 정보를 수정한다.
    boardService.updateBoard(board);
```

```

// 현재 요청이 검색 요청인지 여부를 판단하는 searchOption 설정
boolean searchOption = (type.equals("null")
    || keyword.equals("null")) ? false : true;

// RedirectAttributs의 addAttribute() 메서드를 사용해 파라미터 설정
reAttrs.addAttribute("searchOption", searchOption);
reAttrs.addAttribute("pageNum", pageNum);

// 검색 요청이면 type과 keyword를 모델에 저장한다.
if(searchOption) {

    /* Redirect 되는 경우 주소 끝에 파라미터를 지정해 GET방식의 파라미터로
     * 전송할 수 있지만 스프링프레임워크가 지원하는 RedirectAttributs객체를
     * 이용하면 한 번만 사용할 임시 데이터와 지속적으로 사용할 파라미터를 구분해
     * 지정할 수 있다.
     *
     * 리다이렉트 될 때 필요한 파라미터를 스프링이 제공하는 RedirectAttributs의
     * addAttribute() 메서드를 사용해 파라미터를 지정하면 자동으로 주소 뒤에
     * 요청 파라미터로 추가되며 파라미터에 한글이 포함되는 경우 URLEncoding을
     * java.net 패키지의 URLEncoder 클래스를 이용해 인코딩을 해줘야 하지만
     * application.properties에 인코딩 관련 설정이 되어 있기 때문에 별도로
     * 처리할 필요가 없다.
     *
     * 다음은 검색 리스트로 Redirect 하면서 같이 보내야할 keyword와 type을
     * RedirectAttributs를 이용해 파라미터로 설정하고 있다.
     */
    reAttrs.addAttribute("type", type);
    reAttrs.addAttribute("keyword", keyword);
}

/* 게시글 수정이 완료되면 게시글 리스트로 리다이렉트 시킨다.
 * 클라이언트 요청을 처리한 후 리다이렉트 해야 할 경우 아래와 같이 redirect:
 * 접두어를 붙여 뷰 이름을 반환하면 된다. 뷰 이름에 redirect 접두어가 붙으면
 * HttpServletResponse를 사용해서 지정한 경로로 Redirect 된다.
 */
return "redirect:boardList";
}

```

## 2-6) 게시글 삭제 요청

이번 예제는 앞에서 구현한 페이징 기능에 게시글 검색 기능을 추가하는 것이므로 게시글 삭제 요청을 처리한 후 게시글 리스트와 검색 리스트로 구분해 리다이렉트할 때 사용자가 이전에 있었던 각각의 페이지로 리다이렉트 될 수 있도록 pageNum과 type 그리고 keyword를 Controller 클래스

에서 받아 요청 파라미터에 추가해야 한다. 그래야 사용자가 이전에 있었던 게시글 리스트나 검색 리스트로 이동시킬 수 있다.

게시글 삭제 요청 또한 앞에서 구현한 Persistence 계층과 Business 계층의 기능이 모두 동일하므로 교안에서 생략되었고 Controller 클래스만 수정하면 된다.

## ▶ Controller 클래스

BoardController에서 게시글 삭제 요청을 완료하고 게시글 리스트로 리다이렉트하는 코드를 다음과 같이 수정하자.

### - com.springbootstudy.bbs.controller.BoardController

```
/* 게시글 삭제 요청을 처리 메서드
 * "/delete"로 들어오는 HTTP POST 요청을 처리하는 메서드
 */
@PostMapping("/delete")
public String deleteBoard(RedirectAttributes reAttrs,
    HttpServletResponse response, PrintWriter out,
    @RequestParam("no") int no, @RequestParam("pass") String pass,
    @RequestParam(value="pageNum", defaultValue="1") int pageNum,
    @RequestParam(value="type", defaultValue="null") String type,
    @RequestParam(value="keyword", defaultValue="null") String keyword) {

    // 사용자가 입력한 비밀번호가 틀리면 자바스크립트로 응답
    boolean isPassCheck = boardService.isPassCheck(no, pass);
    if(! isPassCheck) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println(" alert('비밀번호가 맞지 않습니다.');"");
        out.println(" history.back();"");
        out.println("</script>");

        return null;
    }

    // 비밀번호가 맞으면 DB 테이블에서 no에 해당하는 게시글을 삭제한다.
    boardService.deleteBoard(no);

    // 현재 요청이 검색 요청인지 여부를 판단하는 searchOption 설정
    boolean searchOption = (type.equals("null")
        || keyword.equals("null")) ? false : true;

    // RedirectAttributes를 이용해 리다이렉트 할 때 필요한 파라미터 설정
    reAttrs.addAttribute("pageNum", pageNum);
    reAttrs.addAttribute("searchOption", searchOption);
}
```

```
// 검색 요청이면 type과 keyword를 모델에 저장한다.  
if(searchOption) {  
  
    /* 다음은 검색 리스트로 Redirect 하면서 같이 보내야할 keyword와 type을  
     * RedirectAttributs를 이용해 파라미터로 지정하고 있다.  
     **/  
    reAttrs.addAttribute("type", type);  
    reAttrs.addAttribute("keyword", keyword);  
}  
  
// 게시글 삭제가 완료되면 게시글 리스트로 리다이렉트 시킨다.  
return "redirect:boardList";  
}
```

## 4. 회원 로그인/로그아웃과 파일업로드 구현

앞에서 게시판을 구현하면서 기본적인 CRUD 기능과 페이징 처리 그리고 게시물 검색 기능을 구현하는 방법에 대해 알아보았다.

이번에는 회원 관련 서비스를 위해서 새로운 Controller 클래스를 추가하고 회원 로그인 기능과 로그아웃 기능을 구현해 보자. 이번 예제는 회원 로그인을 유지하기 위해서 HttpSession을 사용하는 방법에 대해 알아볼 것이다. 그리고 실무에서는 주민번호나 비밀번호와 같은 민감한 회원 정보는 암호화하여 저장하는 것이 일반적이므로 회원 로그인 요청을 처리할 때 스프링 시큐리티에서 제공하는 PasswordEncoder를 사용하여 로그인을 처리하는 방법에 대해서도 알아볼 것이다.

회원이가입에 대한 기능은 다음 챕터에서 Spring MVC가 지원하는 Interceptor를 알아보면서 추가로 구현할 것이며 이번 챕터에서는 회원가입 없이 로그인과 로그아웃 기능을 구현하는 방법에 대해 알아볼 것이며 이와 더불어서 우리의 게시판에 자료실 기능을 지원하기 위해서 파일 업로드와 다운로드 기능을 구현하는 방법에 대해서도 같이 알아볼 것이다.

### 1) 프로젝트 생성 및 환경설정

이번 예제는 앞에서 게시판 검색 기능까지 구현되어 있는 springbootstudy-bbs03 프로젝트를 복사해 springbootclass-bbs04 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 로그인과 로그아웃 기능을 추가할 것이다. 그리고 이어서 파일 업로드와 다운로드 기능도 추가할 것이다.

이번 예제도 앞에서 구현한 게시판에 회원 로그인/로그아웃과 파일 업로드/다운로드 기능을 추가하는 것이므로 설정 파일과 소스 코드가 많은 부분 동일하기 때문에 새로 추가 또는 수정되는 코드만 교안에 작성할 것이다.

#### 1-1) 프로젝트 생성

- 실습용 프로젝트 : springbootclass-bbs04
- 완성 프로젝트 : springbootstudy-bbs04

#### 1-2) 테이블 생성 및 데이터 입력

이번 프로젝트는 앞에서 사용한 게시판 관련 DTO 클래스인 Board 클래스와 springbbs 테이블은 수정 없이 그대로 사용하고 회원 관련 DTO 클래스인 Member 클래스를 새로 만들고 member 테이블을 새롭게 생성해 회원 정보를 추가하여 사용할 것이다. 아래 member.sql 파일을 살펴보면 비밀번호 부분이 암호화된 것을 볼 수 있는데 이 암호화된 데이터는 스프링 시큐리티에서 제공하는 BCryptPasswordEncoder 클래스를 이용해 암호화 인코딩을 한 후 저장한 데이터 이다.

- src/main/resources/SQL/member.sql

## DATABASE 생성 및 선택

CREATE DATABASE IF NOT EXISTS springboot;



```
use springboot;
```

```
# 테이블 생성
```

```
DROP TABLE IF EXISTS member;
```

```
CREATE TABLE IF NOT EXISTS member(
```

```
  id VARCHAR(20) PRIMARY KEY,
```

```
  name VARCHAR(10) NOT NULL,
```

```
  pass VARCHAR(100) NOT NULL,
```

```
  email VARCHAR(30) NOT NULL,
```

```
  mobile VARCHAR(13) NOT NULL,
```

```
  zipcode VARCHAR(5) NOT NULL,
```

```
  address1 VARCHAR(80) NOT NULL,
```

```
  address2 VARCHAR(60) NOT NULL,
```

```
  phone VARCHAR(13),
```

```
  email_get VARCHAR(1),
```

```
  reg_date TIMESTAMP NOT NULL
```

```
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
# 회원 정보 추가
```

```
INSERT INTO member VALUES('midas', '홍길동',
```

```
  '$2a$10$aWYm2BGI/OiMuemBeF4Y8.7WZeVKAoudv/VzgQx697IYlZgQxr/pe',
```

```
  'midastop@naver.com', '010-1234-5678', '14409',
```

```
  '경기 부천시 오정구 수주로 18 (고강동, 동문미도아파트)', '미도아파트 101동 111호',
```

```
  '032-1234-5678', '1', '2022-06-06 12:10:30');
```

```
INSERT INTO member VALUES('admin', '이순신',
```

```
  '$2a$10$b3t8sn6QZGHYARx3OS5KUuPxzWZdY5yHPRxlSdAgByQ7v0BICLzrO',
```

```
  'midastop1@naver.com', '010-4321-8765', '08787',
```

```
  '서울시 관악구 남부순환로 1820 (봉천동, 에그엘로우)', '15층',
```

```
  '02-5678-4325', '0', '2022-05-11 11:20:50');
```

```
INSERT INTO member VALUES('spring', '강감찬',
```

```
  '$2a$10$.g6l.wyIFO1.j4u4gvVtKOnG9ACBUT1GRIDwLMZcjBxZPrCAURLaG',
```

```
  'midas@daum.net', '010-5687-5678', '06043',
```

```
  '서울 강남구 강남대로146길 28 (논현동, 논현아파트)', '논현신동아파밀리에아파트 111동 1234호
```

```
,
```

```
  '02-5326-5678', '1', '2022-06-05 12:10:30');
```

```
commit;
```

```
SELECT * FROM member;
```

### 1-3) 의존 라이브러리 설정

우리는 이번 프로젝트에서 회원 로그인/로그아웃과 파일 업로드 기능을 구현할 것이다.

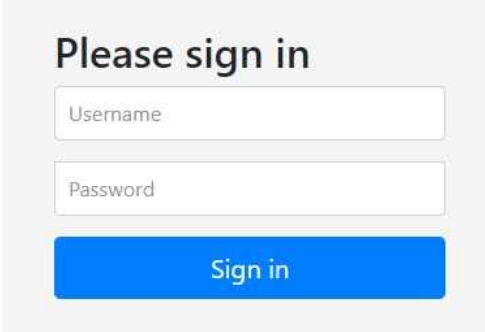
로그인 기능을 구현할 때 회원이 로그인 폼에서 입력한 회원 아이디와 비밀번호를 DB에 저장되어 있는 아이디와 비밀번호가 동일한지를 비교해서 로그인을 처리하는 것이 일반적이다. 그런데 비밀번호

호와 같은 민감한 정보는 DB에 저장할 때 쉽게 노출되지 않도록 암호화 과정을 거쳐서 저장하기 때문에 사용자가 입력한 데이터와 암호화된 데이터를 직접 비교할 수 없으므로 암호화된 데이터를 일반 데이터와 비교할 수 있는 라이브러리가 필요하다. 스프링 프레임워크를 사용할 때는 일반적으로 스프링 시큐리티에서 제공하는 PasswordEncoder를 사용한다.

우리가 구현할 로그인 처리에 필요한 PasswordEncoder는 스프링 시큐리티에서 제공하는 인터페이스이므로 스프링 시큐리티에 대한 의존설정이 필요하다. 그리고 파일 업로드를 처리할 때는 주로 MultipartFile을 사용하기 때문에 이 인터페이스가 포함되어 있는 spring-web을 의존설정 하면 되므로 build.gradle 파일에서 dependencies 부분에 다음과 같이 의존설정을 추가하면 된다. 아래 설정에서 spring-boot-starter-web은 이미 프로젝트를 생성할 때 선택해서 의존설정을 했기 때문에 spring-boot-starter-security에 대해서만 의존설정을 추가하면 된다.

```
implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.springframework.boot:spring-boot-starter-security'
```

build.gradle 파일에 의존설정을 완료하고 Refresh Gradle Project 명령을 실행해 설정을 적용한 후 프로젝트를 실행하여 브라우저로 접속해 보면 다음과 같이 로그인 화면이 나타나는 것을 볼 수 있다.



The image shows a simple login interface. At the top, it says "Please sign in". Below this, there are two text input fields. The first is labeled "Username" and the second is labeled "Password". Below these fields is a prominent blue button with the text "Sign in" in white.

이는 스프링 시큐리티를 프로젝트에 적용하면 서버로 들어오는 모든 요청을 기본적으로 스프링 시큐리티의 제어를 받아 접근하도록 URL 필터가 설정되어 있어서 사용자 인증 절차를 거쳐야 사이트에 접속할 수 있다.

우리 프로젝트에서 스프링 시큐리티 의존설정을 완료하고 프로젝트를 다시 실행한 후 서버 콘솔 창을 살펴보면 다음과 같은 비밀번호가 로그로 출력된 것을 볼 수 있을 것이다.

**Using generated security password: cfc8006c-548f-47bd-8eac-594f1bfc7c0a**

이 비밀번호는 스프링 시큐리티가 사이트 접속을 위해서 사용할 사용자 비밀번호를 임의로 생성해서 출력해 주는 로그이며 로그인 폼에 사용자 이름 : user, 비밀번호 : 로그에 출력된 password를 입력하면 로그인 할 수 있다.

## 1-4) 스프링 시큐리티 환경설정

스프링 시큐리티를 프로젝트에 적용하면 첫 페이지부터 인증을 위한 로그인 폼으로 이동하게 되는데 이렇게 되면 기본적으로 익명 사용자에게 제공되는 웹 페이지도 모두 로그인이 필요하게 된다. 그래서 스프링 시큐리티 설정을 통해서 사이트에 모든 URL에 대해서 접근할 수 있도록 열어두고 필요한 부분만 인증을 거쳐서 접근할 수 있도록 설정할 것이다. 이를 위해서 스프링 시큐리티 설정을 위한 SecurityConfig 클래스를 새로 만들고 다음과 같이 작성하자.

### ▶ 스프링 시큐리티 환경설정 클래스

- `com.springbootstudy.bbs.configurations.SecurityConfig`

```
/* @Configuration은 스프링 환경설정 클래스를 지정하는 애노테이션으로
 * 이 애노테이션이 붙은 클래스는 스프링 DI 컨테이너가 초기화 될 때 빈으로 등록된다.
 */
@Configuration
// 요청 URL이 스프링 시큐리티의 제어를 받도록 지정하는 애노테이션
@EnableWebSecurity
public class SecurityConfig {

    /* @Configuration이 적용된 클래스의 메서드에 @Bean 애노테이션을 지정하면
     * 빈을 생성하는 메소드가 되며 이 메서드 안에서 반환하는 객체는 스프링 DI
     * 컨테이너에 의해서 스프링 Bean으로 관리된다. 그리고 이 객체를 주입 받기를
     * 원하는 @Autowired, @Inject 등과 같은 애노테이션이 적용된 곳으로 주입된다.
     */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /* @EnableWebSecurity 애너테이션을 사용하면 내부적으로
     * SpringSecurityFilterChain이 동작하여 URL 필터가 자동으로 적용된다.
     */
    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(
            /* 스프링 시큐리티를 적용하면 모든 요청 URL에서 인증을 시도하여
             * 로그인 창이 나타난다. 그래서 다음과 같이 별도로 인증하지 않도록
             * 설정하면 사이트의 모든 페이지에 접근할 수 있다.
             */
            /**/
            authorizeHttpRequests ->
                authorizeHttpRequests.requestMatchers(
                    new AntPathRequestMatcher("/**"))
                .permitAll()
        );
    }
}
```

```

/* 스프링 시큐리티가 CSRF 처리시 아래 URL은 예외로 처리하도록 설정
 * CSRF(cross site request forgery)는 웹 사이트 취약점 공격을 방지하기
 * 위한 기술로 서버 상태를 변화시킬 수 있는 POST, PUT, PATCH, DELETE
 * 등의 요청을 보호한다.
 */
.csrf(csrf -> csrf.ignoringRequestMatchers(
    new AntPathRequestMatcher("/h2-console/**")))

/* 스프링 시큐리티를 사용하면 CSRF protection(보호)이 기본 설정되기
 * 때문에 초기 개발 중에는 일시적으로 CSRF 적용을 해제하기도 한다.
 *
 * CSRF protection을 적용하였을 때, 서버의 상태를 변화시킬 수 있는
 * POST 등의 요청을 서버로 보낼때는 CSRF 토큰이 포함되어야 요청을
 * 받아들이게 된다. 만약 CSRF 토큰이 포함되지 않았으면 403(Forbidden)
 * 오류가 발생한다. HTTP 403 상태 코드는 서버에 요청은 전달되었지만
 * 서버에서 허용하지 않는 자원을 요청했기 때문에 거부하는 것으로
 * 권한이 없는 자원에 접근할 때 해당 요청이 거절되었다는 의미이다.
 *
 * 아래에서 CSRF를 disable 시키지 않으며 게시글 쓰기 등의 POST 요청에서
 * CSRF 토큰이 포함되어야 요청을 받아들이기 때문에 게시글 쓰기 폼에는
 * 다음과 같이 CSRF 토큰이 서버로 전달되도록 숨김 폼 컨트롤을 적용해야 한다.
 *
 * <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
 */
.csrf(csrf -> csrf.disable())

// 스프링 시큐리티의 기본 로그인을 사용하지 않거나 로그인 페이지 지정하기
// .formLogin(formLogin -> formLogin
//     .disable()
//     .loginPage("/login")
//     .defaultSuccessUrl("/boardList"));

// 스프링 시큐리티 로그아웃 기능 설정
.logout((logout) -> logout
    //     .logoutUrl("/logout") // 기본 URL은 POST 방식의 /logout
    .logoutSuccessUrl("/loginForm") // 로그아웃 성공 리다이렉트 페이지
    .invalidateHttpSession(true)); // 기존 세션 삭제 여부

return http.build();
}
}

```

## 2) 회원 로그인/로그아웃 기능 구현

우리의 프로젝트에서 Spring MVC와 MyBatis를 활용해 회원 로그인/로그아웃 기능을 구현해 보자. 회원 관련 기능에 대해서 매퍼 인터페이스와 XML 매퍼 파일 그리고 서비스 클래스를 분리하여 별도로 작성하고 회원 관련 요청을 담당하는 MemberController를 새로 추가할 것이다.

### 2-1) 회원 로그인 구현

회원 로그인 기능은 회원 아이디와 비밀번호를 입력할 수 있는 로그인 폼을 보여주는 요청과 로그인 폼에서 회원이 입력한 정보를 가지고 로그인을 요청하는 처리로 나뉘어서 처리해야 한다.

스프링 시큐리티를 사용해 로그인 처리를 설정할 수도 있지만 우리는 컨트롤러에서 직접 URL을 맵핑하여 로그인을 처리할 것이다.

이번에 회원 로그인을 구현할 때 사용할 로그인 폼은 2가지 유형을 사용해 로그인을 처리하는 방법에 대해서 알아볼 것이다. 로그인 폼 2가지 유형 중 하나는 별도의 로그인 폼 페이지를 표시하여 사용자로부터 회원 아이디와 비밀번호를 입력받을 수 있도록 구현할 것이고 또 다른 하나는 로그인 요청이 발생한 현재 페이지에서 팝업 형식으로 모달 로그인 폼을 표시하여 회원 아이디와 비밀번호를 입력받을 수 있도록 로그인 폼을 구현할 것이다.

이번에 구현하는 회원 로그인 기능에서 로그인 폼을 2가지를 사용한다고 해서 로그인을 처리하는 로직을 2가지로 작성해야 하는 것은 아니다. 단지 로그인 폼만 2가지 형식으로 만들고 로그인을 처리하는 로직은 하나만 구현하면 된다. 다시 말해 Controller 클래스, Service 클래스, 매퍼 인터페이스에서 로그인 기능을 처리하는 메서드는 하나만 작성하면 된다는 말이다.

#### 2-1-1) 회원 로그인 폼 요청 처리

회원 로그인 폼 요청은 단순히 로그인 폼만 화면에 보여주면 되므로 Business 계층과 Persistence 계층의 클래스는 구현할 필요가 없다. 컨트롤러 클래스에 로그인 폼 요청을 처리하는 메서드를 만들어도 되지만 지금과 같이 화면에 출력할 모델은 없고 단순히 폼만 보여주는 경우에는 뷰 전용컨트롤러를 사용하는 것이 더 효율적일 수도 있다.

#### ▶ WebConfig 클래스에 뷰 전용 컨트롤러 설정

앞에서 작성한 WebConfig 클래스의 addViewControllers() 메서드에 다음과 같이 로그인 폼을 화면에 보일 수 있도록 뷰 전용 컨트롤러를 설정하자.

- com.springbootstudy.bbs.configurations.WebConfig 클래스 수정

... 중 략 ...

@Override

```
public void addViewControllers(ViewControllerRegistry registry) {  
    registry.addViewController("/writeForm").setViewName("views/writeForm");  
    registry.addViewController("/writeBoard").setViewName("views/writeForm");  
    // 로그인 폼 뷰 전용 컨트롤러 설정 추가
```

```
registry.addViewController("/loginForm").setViewName("member/loginForm");
}
```

## ▶ 로그인 폼 View

src/main/resources/templates 아래에 회원 관련 뷰 템플릿 페이지를 저장할 member 폴더를 새로 추가하고 loginForm.html을 새로 생성해 다음과 같이 회원 로그인 폼을 작성하자.

### - src/main/resources/templates/member/loginForm.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layouts/main_layout}">
<!-- content -->
<div layout:fragment="content" class="row my-5" id="global-content">
<th:block>
    <link th:href="@{css/member.css}" rel="stylesheet" />
</th:block>
    <div class="col">
        <form class="my-5" id="loginForm" action="login" method="post">
            <h2 class="fw-bold">Member Login</h2>
            <fieldset>
                <legend>Member Login</legend>
                <div id="login">
                    <p>
                        <label for="userId" class="labelStyle">아이디</label>
                        <input type="text" id="userId" name="userId" />
                    </p>
                    <p>
                        <label for="userPass" class="labelStyle">비밀번호</label>
                        <input type="password" id="userPass" name="pass" />
                    </p>
                </div>
                <input type="submit" value="로그인" id="btnLogin" />
                <div id="searchBtn">
                    <p>
                        <span>비밀번호 찾기</span> | <span>아이디 찾기</span> | <span>회원가입</span>
                    </p>
                </div>
            </fieldset>
        </form>
    </div>
</th:block>
```

```
</th:block>
</div>
</html>
```

## ▶ 회원 관련 뷰 페이지 스타일시트(CSS)

회원 관련 뷰 페이지에서 사용되는 스타일을 작성할 member.css 파일을 아래 경로에 새롭게 만들고 회원 로그인 폼의 스타일을 작성하자.

- src/main/resources/static/css/member.css

```
@CHARSET "UTF-8";
/* 로그인 폼 */
#loginForm h2, #loginForm p,
#loginForm input, #loginForm label {
    margin: 0px;
    padding: 0px;
    font-family: "맑은 고딕",돋움;
    font-size: 12px;
}
#loginForm h2 {
    font-size: 40px;
    letter-spacing: -1px;
    color: #C8C8C8;
    text-align: center;
}
#loginForm legend {
    display: none;
}
#loginForm fieldset {
    width: 430px;
    margin: 10px auto;
    border: 5px solid #efefef;
    padding: 50px;
    border-radius: 15px;
}
#loginForm #login {
    float: left;
}
#loginForm label.labelStyle {
    width: 60px;
    display: block;
    float: left;
    font-weight: bold;
}
#loginForm #userId, #loginForm #userPass {
```

```

width: 150px;
border: 1px solid #999;
margin-bottom: 5px;
padding: 2px;
}
#loginForm #btnLogin {
display: block;
background-color: #FF6633;
border-radius: 5px;
border-style: none;
color: #fff;
width: 80px;
height: 57px;
position: relative;
float: left;
left: 10px;
font-size: 13px;
font-weight: bold;
cursor: pointer;
}
#loginForm #searchBtn {
clear: both;
}
#searchBtn > p {
position: relative;
top: 20px;
text-align: center;
}
#searchBtn > p > span {
padding: 15px 10px;
}

```

## ▶ 메인 레이아웃 페이지 모달 로그인 폼 추가

main\_layout.html 파일의 <head> 태그 안쪽에 회원 관련 자바스크립트 코드를 작성할 member.js 파일의 참조를 다음과 같이 추가하자. 그리고 main\_layout.html 문서의 아래쪽에 모달 로그인 폼을 다음과 같이 추가하자.

- src/main/resources/templates/layouts/main\_layout.html

... 중 략 ...

```

<script th:src="@{js/formcheck.js}"></script>
<script th:src="@{js/member.js}"></script>

```



... 중 략 ...

```
<script th:src="@{bootstrap/bootstrap.bundle.min.js}"></script>
<!-- 모달 로그인 폼 추가 -->
<div class="modal fade" id="loginModal" tabindex="-1" aria-labelledby="loginModalLabel"
aria-hidden="true"
data-bs-backdrop="static" data-bs-keyboard="false">
<div class="modal-dialog">
<div class="modal-content">
<div class="modal-header bg-primary bg-gradient text-white">
<h1 class="modal-title fs-5 fw-bold" id="modalLabel">회원 로그인</h1>
<button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
</div>
<form id="modalLoginForm" action="login" method="post">
<div class="modal-body">
<div class="mb-3">
<label for="modalUserId" class="col-form-label fw-bold">아이디 :
</label>
<input type="text" class="form-control" id="modalUserId"
name="userId">
</div>
<div class="mb-3">
<label for="modalUserPass" class="col-form-label fw-bold">비밀번호 :
</label>
<input type="password" class="form-control" id="modalUserPass"
name="pass">
</div>
</div>
<div class="modal-footer">
<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">취소
</button>
<button type="submit" class="btn btn-primary">로그인</button>
</div>
</form>
</div>
</div>
</div>
</body>
</html>
```

## ▶ header 페이지 링크 수정

header.html 파일의 메뉴 부분에 로그인/로그아웃과 관련된 링크를 다음과 같이 수정하자.  
로그아웃 처리는 MemberController에서 GET 방식 요청을 처리하여 로그아웃을 처리하는 방법과

스프링 시큐리티 설정을 통해서 로그아웃을 처리하는 방법에 대해서 알아보기 위해서 아래 로그아웃 링크를 각각 다르게 지정했다.

- src/main/resources/templates/fragments/header.html

```
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
  <!--/* header */-->
  <!--/* header라는 프래그먼트를 정의 */-->
  <div th:fragment="header" class="row border-bottom border-primary">
    <div class="col-4">
      <p></p>
    </div>
    <div class="col-8">
      <div class="row">
        <div class="col">
          <ul class="nav justify-content-end">
            <li class="nav-item">
              <!--/* session에 isLogin 값을 비교해 로그인 상태인지 체크 */-->
              <th:block th:unless="${session.isLogin}">
                <a class="nav-link" th:href="@{loginForm}">로그인-폼</a>
              </th:block>
              <th:block th:if="${session.isLogin}">
                <!--/* MemberController를 통해서 로그아웃 처리 */-->
                <a class="nav-link" th:href="@{memberLogout}">로그아웃</a>
              </th:block>
            </li>
            <li class="nav-item">
              <th:block th:unless="${session.isLogin}">
                <a th:href="@{#}" class="nav-link"
                  th:attrappend="data-bs-toggle='modal', data-bs-target='#loginModal'">
                  로그아웃-모달
                </a>
              </th:block>
              <th:block th:if="${session.isLogin}">
                <!--/* 스프링 시큐리티 설정을 통해서 로그아웃 처리 */-->
                <a class="nav-link" th:href="@{logout}">로그아웃</a>
              </th:block>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="boardList">게시 글 리스트</a>
            </li>
            <li class="nav-item">
              <th:block th:unless="${session.isLogin}">
                <a class="nav-link" href="joinForm">회원가입</a>
              </th:block>
            </li>
          </ul>
        </div>
      </div>
    </div>
  </div>
```

```

        <th:block th:if="${session.isLogin}">
            <a class="nav-link" href="memberUpdateForm">정보수정</a>
        </th:block>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#">주문/배송조회</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#">고객센터</a>
    </li>
</ul>
</div>
</div>
<div class="row">
    <div class="col text-end pe-5 text-primary">
        <div th:if="${session.isLogin}">
            안녕하세요 [[ ${session.member != null} ? ${session.member.id} ]]님
        </div>
    </div>
</div>
</div>
</div>
</html>

```

## ▶ 회원 관련 자바스크립트 파일

회원 관련 자바스크립트 코드를 작성할 member.js 파일을 아래 경로에 새로 만들고 로그인 폼과 모달 로그인 폼이 submit 될 때 아이디와 비밀번호가 제대로 입력되었는지 유효성 검사를 수행하는 자바스크립트 코드를 작성하자.

### - src/main/resources/static/js/member.js

```

// DOM이 준비되면 실행될 콜백 함수
$(function() {

    // 회원 로그인 폼이 submit 될 때 폼 유효성 검사를 위한 이벤트 처리
    $("#loginForm").submit(function() {
        var id = $("#userId").val();
        var pass = $("#userPass").val();

        if(id.length <= 0) {
            alert("아이디가 입력되지 않았습니다.\n아이디를 입력해주세요");
            $("#userId").focus();
            return false;
        }
        if(pass.length <= 0) {

```

```

        alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
        $("#userPass").focus();
        return false;
    }
});

// 모달 로그인 폼이 submit 될 때 폼 유효성 검사를 위한 이벤트 처리
$("#modalLoginForm").submit(function() {
    var id = $("#modalUserId").val();
    var pass = $("#modalUserPass").val();

    if(id.length <= 0) {
        alert("아이디가 입력되지 않았습니다.\n아이디를 입력해주세요");
        $("#modalUserId").focus();
        return false;
    }
    if(pass.length <= 0) {
        alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
        $("#modalUserPass").focus();
        return false;
    }
});
});

```

## 2-1-2) 회원 로그인 요청 처리

회원 로그인 폼에서 들어오는 로그인 요청을 처리하는 방법에 대해서 알아보자.

회원이 로그인 폼에서 회원ID와 비밀번호를 입력하여 서버로 전송하면 DB에 저장된 회원 정보와 비교하여 다른 경우에는 알림 창을 띄워 알려주고 다시 로그인 페이지로 이동하도록 구현할 것이다. 그리고 DB 정보와 동일할 경우에는 로그인 처리를 완료하고 게시글 리스트로 리다이렉트 하도록 구현할 것이다.

### ▶ 회원 DTO 클래스

프로젝트의 com.springbootstudy.bbs.domain 패키지에 회원 한 명의 정보를 관리하며 member 테이블과 1:1로 매핑되는 Member 클래스를 다음 코드를 참고해 작성하자.

#### - com.springbootstudy.bbs.domain.Member

```

/* 한 명의 회원 정보를 저장하는 DTO(Data Transfer Object) 클래스
 * 회원 정보를 저장하고 있는 테이블의 컬럼과 1:1 매핑되는 클래스
 */
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor

```

```
public class Member {
    private String id, name, pass, email, mobile;
    private String zipcode, address1, address2, phone;
    private boolean emailGet;
    private Timestamp regDate;
}
```

## ▶ Persistence 계층 구현

Persistence 계층은 데이터베이스에 접근하는 영역으로 우리는 MyBatis를 활용하여 회원 관련 매퍼 인터페이스와 매퍼 XML을 사용할 것이다.

### ■ 회원 관련 SQL을 분리한 매퍼 XML 파일

먼저 프로젝트의 src/main/resources/mappers 폴더에 Mapper-Template.xml 파일을 복사해서 파일 이름을 MemberMapper.xml로 변경한 후 다음을 참고해 member 테이블에서 회원 id에 해당하는 회원 정보를 읽어오는 매퍼링 구문을 작성하자.

#### - src/main/resources/mappers/MemberMapper.xml

```
<mapper namespace="com.springbootstudy.bbs.mapper.MemberMapper" >
    <!--
        회원 id에 해당하는 회원 정보를 반환하는 매퍼링 구문
    -->
    <select id="getMember" resultType="Member">
        SELECT * FROM member WHERE id = #{id}
    </select>
</mapper>
```

### ■ 회원 관련 DB작업을 수행하는 매퍼 인터페이스

프로젝트의 com.springbootstudy.bbs.mapper 패키지에 MyBatis를 활용해 회원 관련 DB 작업을 전담하는 MemberMapper 인터페이스를 새롭게 만들고 회원 id에 해당하는 회원 정보를 읽어와 반환하는 메서드를 추가하자.

#### - com.springbootstudy.bbs.mapper.MemberMapper

```
/* @Mapper는 MyBatis 3.0부터 지원하는 애노테이션으로 이 애노테이션이 붙은
 * 인터페이스는 별도의 구현 클래스를 작성하지 않아도 MyBatis 매퍼로 인식해
 * 스프링 Bean으로 등록되며 Service 클래스에서 주입 받아 사용할 수 있다.
 *
 * @Mapper 애노테이션을 적용한 인터페이스와 XML 매퍼 파일은 namespace라는
 * 속성으로 연결되기 때문에 XML 매퍼 파일의 namespace를 정의할 때 매퍼
 * 인터페이스의 완전한 클래스 이름과 동일한 namespace를 사용해야 한다.
 */
@Mapper
public interface MemberMapper {
```

```
// 회원 ID에 해당하는 회원 정보를 member 테이블에서 읽어와 반환하는 메서드
public Member getMember(String id);
}
```

## ▶ Business 계층 구현

프로젝트의 com.springbootstudy.bbs.service 패키지에 회원 관련 Business 로직 처리를 전담할 MemberService 클래스를 새로 생성하고 로그인을 처리하는 메서드와 회원 ID에 해당하는 회원 정보를 읽어와 반환하는 메서드를 작성하자.

### - com.springbootstudy.bbs.service.MemberService

```
//MemberService 클래스가 서비스 계층의 스프링 빈(Been)임을 정의
@Service
@Slf4j
public class MemberService {

    // 회원 관련 DB 작업에 필요한 MemberMapper 객체를 의존성 주입하도록 설정
    @Autowired
    private MemberMapper memberMapper;

    /* 회원 로그인 처리에 필요한 PasswordEncoder 객체를 의존성 주입하도록 설정
     *
     * 회원 비밀번호 암호화 처리를 위해서 스프링 시큐리티의 PasswordEncoder를 사용
     * 회원 로그인 요청시 DB 테이블에 암호화 되어 저장된 비밀번호와 사용자가
     * 입력한 일반 문자열 비밀번호를 비교하는데 사용되고 회원가입과 회원 정보 수정에서
     * 사용자가 입력한 비밀번호를 암호화 인코딩하여 저장하는데도 사용된다.
     * com.springbootstudy.bbs.configurations.SecurityConfig 클래스에서 스프링 빈을
     * 생성해 반환하는 @Bean을 적용한 메서드를 정의했기 때문에 여기로 주입된다.
     */
    @Autowired
    private PasswordEncoder passwordEncoder;

    // 회원 로그인 요청을 처리하고 결과를 반환하는 메서드
    public int login(String id, String pass) {
        int result = -1;
        Member m = memberMapper.getMember(id);

        // id가 존재하지 않으면 : -1
        if(m == null) {
            return result;
        }

        /* 로그인 성공 : 1
         * PasswordEncoder 객체의 matches 메소드를 이용해 암호가 맞는지 확인
         */
    }
}
```

```

    * matches() 메소드의 첫 번 인수로 인코딩이 안된 문자열, 두 번째 인수로 인코딩된
    * 문자열을 지정하면 두 문자열의 원본 데이터가 같을 경우 true를 반환해 준다.
    **/
    if(passwordEncoder.matches(pass, m.getPass())) {
        result = 1;

    } else { // 비밀번호가 틀리면 : 0
        result = 0;
    }

    return result;
}

// 회원 ID에 해당하는 회원 정보를 읽어와 반환하는 메서드
public Member getMember(String id) {
    return memberMapper.getMember(id);
}
}

```

## ▶ Controller 클래스

프로젝트의 com.springbootstudy.bbs.controller 패키지에 회원 관련 HTTP 요청 처리를 전담할 MemberController를 새로 만들고 다음 코드를 참고해 회원 로그인 요청을 처리하는 메서드를 추가하자.

### - com.springbootstudy.bbs.controller.MemberController

```

// 스프링 MVC의 컨트롤러 클래스임을 정의
@Controller

/* 스프링은 데이터를 세션 영역에 저장할 수 있도록 @SessionAttributes("모델이름")
 * 애노테이션을 제공하고 있다. 클래스 레벨에 @SessionAttributes("모델이름")와
 * 같이 애노테이션과 모델 이름을 지정하고 아래 login() 메서드와 같이 그 컨트롤러
 * 메서드에서 @SessionAttributes에 지정한 모델이름과 동일한 이름으로 모델에
 * 객체를 추가하면 이 객체를 세션 영역에 저장해 준다.
 */
@SessionAttributes("member")
public class MemberController {

    // 회원 관련 Business 로직을 담당하는 객체를 의존성 주입하도록 설정
    @Autowired
    private MemberService memberService;

    /* "/login"으로 들어오는 POST 방식의 요청을 처리하는 메서드
     *
     * 요청을 처리한 결과를 뷰에 전달하기 위해 사용하는 것이 Model 객체이다.
    */
}

```

- \* 컨트롤러는 요청을 처리한 결과 데이터를 모델에 담아 뷰로 전달하고 뷰는
- \* 모델로부터 데이터를 읽어와 클라이언트로 보낼 결과 페이지를 만들게 된다.
- \*
- \* 스프링은 컨트롤러에서 모델에 데이터를 담을 수 있는 다양한 방법을 제공하는데
- \* 아래와 같이 파라미터에 Model을 지정하는 방식이 많이 사용된다.
- \* @RequestMapping, @PostMapping, @GetMapping 애노테이션이 적용된 메서드의
- \* 파라미터에 Model을 지정하면 스프링이 이 메서드를 호출하면서 Model 타입의
- \* 객체를 넘겨준다.
- \* 우리는 Model을 받아 이 객체에 결과 데이터를 담기만 하면 뷰로 전달된다.
- \*
- \* @RequestMapping, @PostMapping, @GetMapping 애노테이션이 적용된 메서드의
- \* 파라미터에 @RequestParam 애노테이션을 사용해 파라미터 이름을 지정하면
- \* 이 애노테이션이 앞에 붙은 매개변수에 파라미터 값을 바인딩 시켜준다.
- \*
- \* @RequestParam 애노테이션에 사용할 수 있는 속성은 아래와 같다.
- \* value : HTTP 요청 파라미터의 이름을 지정한다.
- \* required : 요청 파라미터가 필수인지 설정하는 속성으로 기본값은 false 이다.
- \* 이 값이 true인 상태에서 요청 파라미터의 값이 존재하지 않으면
- \* 스프링은 Exception을 발생시킨다.
- \* defaultValue : 요청 파라미터가 없을 경우 사용할 기본 값을 문자열로 지정한다.
- \*
- \* @RequestParam(value="id" required="false" defaultValue="")
- \*
- \* @RequestParam 애노테이션은 요청 파라미터 값을 읽어와 메서드의
- \* 파라미터 타입에 맞게 변환해 준다.
- \* 스프링은 요청 파라미터의 값으로 변환할 수 없는 경우 400 에러를 발생시킨다.
- \*\*/

```
@PostMapping("/login")
```

```
public String login(Model model, @RequestParam("userId") String id,
    @RequestParam("pass") String pass,
    HttpSession session, HttpServletResponse response)
    throws ServletException, IOException {
```

```
// MemberService 클래스를 사용해 로그인 성공여부 확인
```

```
int result = memberService.login(id, pass);
```

```
if(result == -1) { // 회원 아이디가 존재하지 않으면
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<script>");
    out.println(" alert('존재하지 않는 아이디 입니다.');"");
    out.println(" history.back();"");
    out.println("</script>");
```

```
/* 컨트롤러에서 null을 반환하거나 메서드의 반환 타입이 void일 경우
```



```

    * Writer나 OutputStream을 이용해 응답 결과를 직접 작성할 수 있다.
    * DispatcherServlet을 경유해 리소스 자원에 접근하는 경우에
    * 자바스크립트의 history.back()은 약간의 문제를 일으킬 수 있다.
    * history 객체를 이용하는 경우 서버로 요청을 보내는 것이 아니라
    * 브라우저의 접속 이력에서 이전 페이지로 이동되기 때문에 발생한다.
    **/
    return null;

} else if(result == 0) { // 비밀번호가 틀리면
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<script>");
    out.println("alert('비밀번호가 다릅니다.');

```

## 2-2) 로그아웃 구현

로그아웃 구현은 별도의 뷰 페이지도 필요 없으며 현재 세션을 HTTP 요청을 처리하는 컨트롤러에서 기존 세션을 종료하고 다시 시작하면 되므로 의외로 간단하다. 하지만 프로젝트에서 스프링 시큐리티를 사용하면 “/logout” URL에 대해서 스프링 시큐리티 설정이 우선 적용되어 POST 방식의 “/logout”이 기본 URL로 매핑 된다. 그래서 우리는 로그아웃 URI를 “/memberLogout”으로 처리할 것이다.

### ▶ Controller 클래스

MemberController 클래스에 다음 코드를 참고해 로그아웃 요청을 처리하는 메서드를 추가하자.

#### - com.springbootstudy.bbs.controller.MemberController

```
/* "/membeLogout"으로 들어오는 GET 방식 요청 처리 메서드
 * 스프링 시큐리티를 사용하면 스프링 시큐리티 설정이 우선 적용되어 POST 방식의
 * "/logout"이 기본 URL로 매핑된다. GET 방식으로 "/logout" 요청을 보내면 405
 * (Method Not Allowed)가 발생한다. 그래서 아래와 같이 매핑 URL을 적용했다.
 *
 * 이 컨트롤러 매핑은 직접 로그아웃을 처리하는 방법에 대해서 설명하고 있지만
 * com.springbootstudy.bbs.configurations.SecuurityConfig 클래스에는 스프링
 * 시큐리티를 적용해 로그인과 로그아웃을 적용하는 설정이 주석으로 설명되어 있다.
 */
@GetMapping("/memberLogout")
public String logout(HttpSession session) {
    log.info("MemberController.logout(HttpSession session)");
    // 현재 세션을 종료하고 새로운 세션을 시작한다.
    session.invalidate();

    /* 클라이언트 요청을 처리한 후 리다이렉트 해야할 경우 아래와 같이 redirect:
     * 접두어를 붙여 뷰 이름을 반환하면 된다. 뷰 이름에 redirect 접두어가 붙으면
     * HttpServletResponse를 사용해서 지정한 경로로 Redirect 된다.
     * redirect 접두어 뒤에 경로를 지정할 때 "/"로 시작하면 ContextRoot를
     * 기준으로 절대 경로 방식으로 Redirect 된다. "/"로 시작하지 않으면 현재
     * 경로를 기준으로 상대 경로로 Redirect 된다. 또한 다른 사이트로 Redirect
     * 되기를 원한다면 redirect:http://사이트 주소를 지정한다.
     *
     * 로그아웃 되면 로그인 폼으로 리다이렉트 된다.
     */
    return "redirect:/loginForm";
}
```

### 3) 파일 업로드/다운로드 기능 구현

게시글 쓰기에서 파일 업로드를 구현하고 게시글 상세보기에서 파일 다운로드를 구현해 보자.

파일 업로드 기능이 추가되면 게시글 수정하기에서도 기존에 업로드된 파일을 수정하는 기능도 제공해야 하지만 이 부분은 게시글 쓰기의 파일 업로드 부분을 참고해 직접 구현해 보자.

#### 3-1) 파일 업로드

스프링은 MultipartResolver를 사용해 Multipart 데이터에 접근할 수 있는 여러 가지 방법을 제공하고 있으며 우리는 MultipartFile 인터페이스를 이용해 파일을 업로드하는 방법에 대해서 알아볼 것이다. 파일을 업로드하는 위치는 서버의 특정 폴더를 지정해 업로드 할 수도 있지만 우리는 프로젝트 안에 위치한 static 폴더 하위에 files/ 폴더에 업로드할 것이다.

##### ▶ WebConfig 클래스에 리소스 핸들러 등록

먼저 스프링 MVC 환경 설정을 위해서 작성한 WebConfig 클래스에 addResourceHandlers() 메서드를 다음과 같이 오버라이드하여 리소스 핸들러를 등록하자.

##### - com.springbootstudy.bbs.configurations.WebConfig

```
/* addResourceHandlers() 메서드는 리소스 등록 및 핸들러를 관리하는 ResourceHandler
 * Registry 객체를 통해서 리소스 URL과 이 URL에 매칭되는 리소스의 위치를 등록할 수 있다.
 */
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {

    /* 기존에 정적 리소스 핸들러의 설정은 그대로 유지하며 새로운 리소스 핸들러 추가
     * /resources/** 로 요청되는 리소스 요청 설정
     */
    registry.addResourceHandler("/resources/files/**")
        // file: 프로토콜을 사용하면 업로드한 이미지가 바로 보인다.
        .addResourceLocations("file:./src/main/resources/static/files/")
        .setCachePeriod(1); // 캐쉬 지속시간(초)
}
```

##### ▶ application.properties 파일에 Multipart 관련 설정

프로젝트 환경 설정 파일인 application.properties 파일에 Multipart 관련 설정을 다음과 같이 추가하자.

##### - src/main/resources/application.properties

```
# 파일 업로드 설정 - 최대 크기 기본이 1MB이며 초과하면 오류 발생
# 파일 한 개 당 최대 크기
spring.servlet.multipart.max-file-size=10MB
```

```
# 요청당 최대 파일 크기 - 10MB 10개까지
spring.servlet.multipart.max-request-size=100MB
```

## ▶ 게시글 쓰기 폼 View

게시글 쓰기 폼은 앞에서와 동일한 writeForm.jsp 페이지에서 파일 업로드를 처리하기 위해서 form 태그에 아래와 같이 enctype="multipart/form-data"을 추가하여 폼이 전송될 때 파일도 같이 전송될 수 있도록 인코딩 타입을 지정하고 파일 선택 상자를 폼에 추가하자.

- src/main/resources/templates/views/writeForm.html

```
<form name="writeForm" action="addBoard" id="writeForm"
      class="row g-3 border-primary" method="post" enctype="multipart/form-data">

    ... 중 략 ...

    <div class="col-8 offset-md-2">
        <label for="content" class="form-label">내용</label>
        <textarea class="form-control" name="content" id="content"
rows="10"></textarea>
    </div>
    <div class="col-8 offset-md-2">
        <label for="addFile" class="form-label">파일</label>
        <input type="file" class="form-control" name="addFile" id="addFile">
    </div>

    ... 중 략 ...

</form>
```

## ▶ Persistence, Business 계층 구현

Persistence와 Business 계층의 클래스는 앞에서 사용한 클래스 그대로 사용하면 되기 때문에 수정 또는 추가되는 코드는 없다.

## ▶ Controller 클래스

BoardController 클래스에 업로드 되는 파일을 저장할 폴더 위치를 DEFAULT\_PATH 라는 상수로 정의하고 MultipartFile 인터페이스를 이용해 파일 업로드를 처리할 HTTP Post 요청을 받는 addBoard() 메서드를 다음과 같이 수정하자.

- com.springbootstudy.bbs.controller.BoardController  
@Controller

@Slf4j

**public class** BoardController {

// 업로드한 파일을 저장할 폴더 위치를 상수로 선언하고 있다.

**private static final** String **DEFAULT\_PATH** = "src/main/resources/static/files/";

... 중 략 ...

/\* 게시글 쓰기 폼에서 들어오는 게시글 쓰기 요청을 처리하는 메서드

\* "/addBoard"로 들어오는 HTTP POST 요청을 처리하는 메서드

\*\*/

@PostMapping("/addBoard")

**public** String addBoard(Board board,

@RequestParam(value="addFile", required=false) MultipartFile multipartFile)

**throws** IOException {

System.out.println("originName : " + multipartFile.getOriginalFilename());

System.out.println("name : " + multipartFile.getName());

// 업로드된 파일이 있으면

**if** (multipartFile != **null** && !multipartFile.isEmpty()) {

// File 클래스는 파일과 디렉터리를 다루기 위한 클래스

File parent = **new** File(**DEFAULT\_PATH**);

// 파일 업로드 위치에 폴더가 존재하지 않으면 폴더 생성

**if** (!parent.isDirectory() && !parent.exists()) {

parent.mkdirs();

}

UUID uid = UUID.randomUUID();

String extension = StringUtils.getFilenameExtension(  
 multipartFile.getOriginalFilename());

String saveName = uid.toString() + "." + extension;

File file = **new** File(parent.getAbsolutePath(), saveName);

// File 객체를 이용해 파일이 저장될 절대 경로 출력

log.info("file abs path : " + file.getAbsolutePath());

log.info("file path : " + file.getPath());

// 업로드 되는 파일을 static/files 폴더에 복사한다.

multipartFile.transferTo(file);

// 업로드된 파일 이름을 게시글의 첨부 파일로 설정한다.

board.setFile1(saveName);



```

<td colspan="3">
    <th:block th:if="{ not #strings.isEmpty(board.file1) }">
        <a th:href="@{|fileDownload?fileName={board.file1}|">다운로드</a>
    </th:block>
    <th:block th:unless="{ not #strings.isEmpty(board.file1) }">
        첨부파일 없음
    </th:block>
</td>
</tr>

```

... 중 략 ...

## ▶ 게시글 다운로드 Controller 클래스

게시글 상세보기에서 파일 다운로드 요청이 들어오면 클라이언트가 파일을 다운로드 받을 수 있도록 Stream 객체를 이용해 클라이언트에 파일을 전송하는 download() 메서드를 BoardController에 추가한다.

### - com.springbootstudy.bbs.controller.BoardController

// 게시글 상세보기에서 들어오는 파일 다운로드 요청을 처리하는 메서드

@GetMapping("/fileDownload")

```

public void download(HttpServletRequest request,
    HttpServletResponse response) throws Exception {

```

```

    String fileName = request.getParameter("fileName");

```

```

    log.info("fileName : " + fileName);

```

```

    File parent = new File(DEFAULT_PATH);

```

```

    File file = new File(parent.getAbsolutePath(), fileName);

```

```

    log.info("file.getName() : " + file.getName());

```

// 응답 데이터에 파일 다운로드 관련 콘텐츠 타입 설정이 필요하다.

```

response.setContentType("application/download; charset=UTF-8");

```

```

response.setContentLength((int) file.length());

```

// 한글 파일명을 클라이언트로 바로 내려보내기 때문에 URLEncoder가 필요하다.

```

fileName = URLEncoder.encode(file.getName(), "UTF-8");

```

```

log.info("다운로드 fileName : " + fileName);

```

// 전송되는 파일 이름을 한글 그대(원본 파일 이름 그대로)로 보내주기 위한 설정이다.

```

response.setHeader("Content-Disposition",
    "attachment; filename=\"" + fileName + "\"");

```

// 파일로 전송되어야 하므로 전송되는 데이터 인코딩은 바이너리로 설정해야 한다.

```

response.setHeader("Content-Transfer-Encoding", "binary");

// 파일을 클라이언트로 보내기 위해 응답 스트림을 구한다.
OutputStream out = response.getOutputStream();
FileInputStream fis = null;

/* FileInputStream을 통해 클라이언트로 보낼 파일을 읽어
 * 스프링이 제공하는 FileCopyUtils 클래스를 통해서
 * 데이터를 읽고 응답 스트림을 통해 클라이언트로 출력한다.
 */
fis = new FileInputStream(file);

// 스프링이 제공하는 FileCopyUtils를 이용해 응답 스트림에 파일을 복사한다.
FileCopyUtils.copy(fis, out);

if(fis != null) {
    fis.close();
}

/* 파일 데이터를 클라이언트로 출력한다.
 * 버퍼에 남아 있는 모든 데이터를 클라이언드로 출력한다.
 */
out.flush();
}

```



## 5. 회원 전용 게시판 구현

앞에서 구현한 웹 애플리케이션은 회원가입 없이 로그인과 로그아웃 기능만 제공하고 있다. 실제로 회원가입 없이 로그인과 로그아웃 기능만을 제공하는 웹 사이트는 아마도 존재하지 않을 것이다. 그래서 이번에는 회원가입과 회원 정보수정 기능을 구현할 것이다. 또한 로그인을 하지 않으면 게시글 리스트만 볼 수 있고 검색을 하거나 게시글 쓰기, 게시글 상세보기 등의 기능을 이용할 수 없도록 회원 전용 게시판을 구현할 것이다. 이런 기능을 구현하기 위해서는 로그인 서비스가 필요한 부분에서 회원이 로그인 상태인지를 일일이 체크하는 코드가 필요하다. 그러므로 회원 전용서비스의 각 요청을 처리하는 여러 컨트롤러의 메서드마다 세션을 체크해 로그인 상태인지 아닌지를 확인해야하므로 동일한 코드의 중복이 많이 발생할 수 있다. 이렇게 여러 컨트롤러에 걸쳐서 공통으로 적용해야 할 기능을 구현할 때 스프링프레임워크가 제공하는 HandlerInterceptor를 애플리케이션에 적용하면 로그인 체크와 같이 클라이언트가 요청할 때 마다 여러 곳에서 걸쳐서 중복되는 코드를 줄일 수 있다. 실무에서도 여러 요청 경로 또는 여러 컨트롤러에 걸쳐서 공통으로 적용해야 할 기능을 구현할 때 HandlerInterceptor를 많이 활용하는데 우리도 이 HandlerInterceptor를 활용해 회원 전용서비스를 구현할 것이다. 그리고 회원가입 시에 주소를 선택하고 우편번호를 입력하는 기능은 다음 (daum.net)에서 제공하는 우편번호 API를 사용할 것이다.

### 1) 프로젝트 생성 및 환경설정

이번 예제는 앞에서 구현한 springbootstudy-bbs04 프로젝트를 복사해 springbootclass-bbs05 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 회원가입, 회원 정보수정과 Interceptor를 활용한 로그인 체크 기능을 구현할 것이다.

이번 예제도 앞에서 구현한 게시판에 회원제 게시판 기능을 추가하는 것이므로 설정 파일과 소스 코드가 많은 부분 동일하기 때문에 새로 추가 또는 수정되는 코드만 교안에 작성할 것이다.

#### 1-1) 프로젝트 생성

- 실습용 프로젝트 : springbootclass-bbs05
- 완성 프로젝트 : springbootstudy-bbs05

#### 1-2) 테이블 생성 및 데이터 입력

이번 예제는 앞에서 사용한 springbbs, member 테이블을 수정 없이 그대로 사용하기 때문에 추가로 테이블을 생성할 필요는 없다.

#### 1-3) 의존 라이브러리 설정

회원가입이나 회원 정보 수정에서 회원이 입력한 비밀번호를 암호화하여 저장하는 기능이 필요하다. 그래서 스프링 시큐리티에서 제공하는 PasswordEncoder를 사용해 비밀번호 암호화를 진행하기 위해서 spring-boot-starter-security에 대한 의존설정이 필요하지만 이전 프로젝트에서 회원 로그인

을 처리할 때 사용하기 위해서 이미 build.gradle 파일에 의존설정을 했기 때문에 추가로 설정할 필요는 없다.

## 1-4) 스프링 시큐리티 환경설정

스프링 시큐리티를 프로젝트에 적용하면 스프링 시큐리티 설정을 위한 SecurityConfig 클래스를 새로 만들고 스프링 시큐리티에 대한 환경설정을 해야한다. 하지만 우리는 이미 앞의 프로젝트에서 해당 클래스를 만들고 설정을 완료했기 때문에 그 설정을 그대로 사용할 것이다.

## 2) 회원가입 구현

회원가입 기능은 회원 정보를 입력할 수 있는 폼을 먼저 보여주고 사용자가 정보를 입력하고 폼을 전송해 회원정보를 등록할 수 있도록 구현해야 한다. 그러므로 회원가입 폼을 보여주는 요청 처리와 회원가입 폼에서 입력한 정보를 가지고 회원등록을 요청하는 처리로 나눠서 구현해야 한다.

### 2-1) 회원가입 폼 요청 처리

회원가입 폼 요청은 단순히 회원 정보를 입력할 수 있는 폼만 화면에 보여주면 되므로 Business 계층과 Persistence 계층의 클래스는 필요 없다. 컨트롤러 클래스에서 회원가입 폼 요청을 처리하는 메서드를 만들어도 되지만 지금과 같이 화면에 출력할 모델은 없고 단순히 폼만 보여주는 경우에는 뷰 전용컨트롤러를 사용하는 것이 더 효율적이다.

#### ▶ WebConfig 클래스에 뷰 전용 컨트롤러 설정

앞에서 작성한 WebConfig 클래스의 addViewControllers() 메서드에 다음과 같이 회원가입 폼을 화면에 보일 수 있도록 뷰 전용 컨트롤러를 설정하자.

- com.springbootstudy.bbs.configurations.WebConfig 클래스 수정

```
@Override
public void addViewControllers(ViewControllerRegistry registry) {

    ... 중 략 ...

    // 회원가입 폼 뷰 전용 컨트롤러 설정 추가
    registry.addViewController("/joinForm").setViewName("member/memberJoinForm");
}
```

#### ▶ 회원가입 폼 View

회원 관련 뷰 템플릿을 작성하는 아래 폴더에 memberJoinForm.html을 새로 추가하고 다음과 같이 회원가입 폼을 작성하자.

- src/main/resources/templates/member/memberJoinForm.html

```
<!DOCTYPE html>
<!--/*
    Thymeleaf와 Thymeleaf Layout을 사용하기 위한 NameSpace를 정의한다.
    layout:decorate 옵션은 아래의 <th:block layout:fragment="content">
    부분을 어떤 레이아웃에 적용할지 설정하는 것으로 경로 지정은
    templates 폴더를 기준으로 지정하면 된다.
*/-->
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
```

```

    layout:decorate="~{layouts/main_layout}">
<!--/*
    block을 사용해도 되고 div에 직접 layout:fragment를 적용해도 된다.
    main_layout.html에서 지정한 layout:fragment의 이름과 같아야 한다.
*/-->
<th:block layout:fragment="content">
<!--/*
    새로운 5자리 우편번호로 회원 주소를 입력 받기 위해 daum.net에서
    제공하는 우편번호 찾기 API를 사용하였다.
    참고 사이트 : http://postcode.map.daum.net/guide
*/-->
<script src=
"https://t1.daumcdn.net/mapjsapi/bundle/postcode/prod/postcode.v2.js"></script>
<div class="row my-5" id="global-content">
    <div class="col">
        <div class="row my-3 text-center">
            <div class="col">
                <h2 class="fs-3 fw-bold">회원 정보 입력</h2>
            </div>
        </div>
        <form action="joinResult" name="joinForm" method="post" id="joinForm">
            <!--/*
                회원 아이디 중복 검사를 했는지의 정보를 hidden 필드로 저장
            */-->
            <input type="hidden" name="isIdCheck" id="isIdCheck" value="false"/>
            <div class="row mt-5 mb-3">
                <div class="col-8 offset-2">
                    <label for="name" class="form-label">* 이 름 : </label>
                    <input type="text" class="form-control" name="name" id="name">
                </div>
            </div>
            <div class="row my-3">
                <div class="col-8 offset-2">
                    <label for="userId" class="form-label">* 아이디 : </label>
                    <div class="row">
                        <div class="col-6">
                            <input type="text" class="form-control" name="id" id="id">
                        </div>
                        <div class="col-4">
                            <input type="button" class="btn btn-warning" id="btnOverlapId" value="중
복확인">
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```

<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="pass1" class="form-label">* 비밀번호 : </label>
    <input type="password" class="form-control" name="pass1" id="pass1">

  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="pass2" class="form-label">* 비밀번호 확인 : </label>
    <input type="password" class="form-control" name="pass2" id="pass2">
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="zipcode" class="form-label">* 우편번호 : </label>
    <div class="row">
      <div class="col-4">
        <input type="text" class="form-control" name="zipcode" id="zipcode"
maxlength="5" readonly>
      </div>
      <div class="col-4">
        <input type="button" class="btn btn-warning" id="btnZipcode" value="우편
번호 찾기">
      </div>
    </div>
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="address1" class="form-label">* 자택주소 : </label>
    <input type="text" class="form-control" name="address1" id="address1"
readonly>
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="address2" class="form-label">상세주소 : </label>
    <input type="text" class="form-control" name="address2" id="address2">
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="emailId" class="form-label">* 이메일 : </label>
    <div class="row">

```

```

        <div class="col-md-4">
            <input type="text" class="form-control" name="emailId" id="emailId">
        </div> @
        <div class="col-md-4">
            <input type="text" class="form-control" name="emailDomain"
id="emailDomain">
        </div>
        <div class="col-md-3">
            <select class="form-select" name="selectDomain" id="selectDomain">
                <option>직접입력</option>
                <option>네이버</option>
                <option>다음</option>
                <option>한메일</option>
                <option>구글</option>
            </select>
        </div>
    </div>
</div>
<div class="row my-3">
    <div class="col-8 offset-2">
        <label for="mobile2" class="form-label">* 휴 대 폰 : </label>
        <div class="row">
            <div class="col-md-3">
                <select class="form-select" name="mobile1" id="mobile1">
                    <option>010</option>
                    <option>011</option>
                    <option>016</option>
                    <option>017</option>
                    <option>018</option>
                    <option>019</option>
                </select>
            </div>-
            <div class="col-md-4">
                <input type="text" class="form-control" name="mobile2" id="mobile2"
maxlength="4">
            </div>-
            <div class="col-md-4">
                <input type="text" class="form-control" name="mobile3" id="mobile3"
maxlength="4">
            </div>
        </div>
    </div>
</div>
<div class="row my-3">

```

```

<div class="col-8 offset-2">
  <label class="form-label">메일 수신여부 : </label>
  <div class="row">
    <div class="col-md-3">
      <div class="form-check">
        <input type="radio" class="form-check-input" name="emailGet"
id="emailOk" value="true">
          <label class="form-check-label" for="emailOk">수신함</label>
        </div>
      </div>
      <div class="col-md-3">
        <div class="form-check">
          <input type="radio" class="form-check-input" name="emailGet"
id="emailNo" value="false">
            <label class="form-check-label" for="emailNo">수신않함</label>
          </div>
        </div>
      </div>
    </div>
  </div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="phone2" class="form-label">자택전화 : </label>
    <div class="row">
      <div class="col-md-3">
        <select class="form-select" name="phone1" id="phone1">
          <option>02</option>
          <option>031</option>
          <option>032</option>
          <option>033</option>
          <option>041</option>
          <option>042</option>
          <option>043</option>
          <option>044</option>
          <option>051</option>
          <option>052</option>
          <option>053</option>
          <option>054</option>
          <option>055</option>
          <option>061</option>
          <option>062</option>
          <option>063</option>
          <option>064</option>
        </select>
      </div> -
    </div>
  </div>

```

```

        <div class="col-md-4">
            <input type="text" class="form-control" name="phone2" id="phone2"
maxlength="4">
        </div> -
        <div class="col-md-4">
            <input type="text" class="form-control" name="phone3" id="phone3"
maxlength="4">
        </div>
    </div>
</div>
<div class="row mb-3 mt-5">
    <div class="col-8 offset-2">
        <input type="submit" value="가입하기" class="btn btn-primary">
    </div>
</div>
</form>
</div>
</div>
</th:block>
</html>

```

## 2-2) 회원 아이디 중복 확인 요청 처리

회원가입 폼에서 회원 아이디가 중복되는지 체크하는 기능을 추가할 것이다. 위에서 작성한 회원가입 폼에서 “중복확인” 버튼이 클릭되면 회원 아이디가 입력되었는지 체크한 후 아이디가 입력되었으면 새 창을 띄워서 아이디 중복 확인 요청을 서버로 보내서 회원 아이디가 중복되는지를 확인하는 처리를 구현해 보자.

### ▶ 회원 관련 자바스크립트 파일

member.js 파일에 회원가입 폼에서 회원 아이디 입력란에는 영문자와 숫자만 입력되도록 처리하기 위해서 키보드의 키가 눌릴 때마다 이를 확인하는 이벤트 핸들러와 “중복확인” 버튼이 클릭되면 새 창을 띄워서 회원 아이디 중복 확인을 요청하는 버튼 이벤트 핸들러를 다음과 같이 작성하자.

#### - src/main/resources/static/js/member.js

```
// DOM이 준비되면 실행될 콜백 함수
```

```
$(function() {
```

```
    ... 중 략 ...
```

```
/* 회원가입 폼, 회원정보 수정 폼에서 폼 컨트롤에서 키보드 입력을
```

```
* 체크해 유효한 값을 입력 받을 수 있도록 keyup 이벤트를 처리 했다.
```



```

**/
$("#id").on("keyup", function() {
    // 아래와 같이 정규표현식을 이용해 영문 대소문자, 숫자만 입력되었는지 체크할 수 있다.
    var regExp = /^[A-Za-z0-9]/gi;
    if(regExp.test($(this).val())) {
        alert("영문 대소문자, 숫자만 입력할 수 있습니다.");
        $(this).val($(this).val().replace(regExp, ""));
    }
});

```

/\* 바로 위에서 키보드의 키가 눌러질 때 사용자가 아이디 입력란에 영문 대소문자  
 \* 또는 숫자만 입력하였는지를 체크하는 이벤트 처리 코드를 작성하였다. 이런 이벤트  
 \* 처리는 아이디 입력 체크에만 사용할 수 있는 것이 아니라 비밀번호 입력이나 이메일  
 \* 입력에서도 유효한 데이터가 입력되었는지 체크해야 된다. 이렇게 여러 문서 객체에  
 \* 동일한 이벤트 처리를 해야 하므로 각각 이벤트 처리를 구현하게 되면 동일한 코드가  
 \* 계속해서 중복되는 현상이 발생하게 된다. 이럴 경우에는 동일한 코드를 하나의  
 \* 함수로 만들어 놓고 여러 곳에서 사용하면 코드의 중복을 막고 코드의 재사용성을  
 \* 높일 수 있다. 아래는 동일한 코드에 대한 별도의 함수를 이 자바스크립트 파일의  
 \* 아래 부분에 작성해 놓고 이 함수를 이벤트 핸들러를 등록해 사용하는 방식이다.

```

**/
$("#pass1").on("keyup", inputCharReplace);
$("#pass2").on("keyup", inputCharReplace);
$("#emailId").on("keyup", inputCharReplace);
$("#emailDomain").on("keyup", inputEmailDomainReplace);

```

/\* 회원가입 폼에서 아이디 중복확인 버튼이 클릭되면  
 \* 아이디 중복을 확인할 수 있는 새 창을 띄워주는 함수

```

**/
$("#btnOverlapId").on("click", function() {
    var id = $("#id").val();
    url="overlapIdCheck?id=" + id;

    if(id.length == 0) {
        alert("아이디를 입력해주세요");
        return false;
    }

    if(id.length < 5) {
        alert("아이디는 5자 이상 입력해주세요.");
        return false;
    }

    window.open(url, "idCheck", "toolbar=no, scrollbars=no, resizable=no, "
        + "status=no, memubar=no, width=500, height=330");
});

```

```
});

// 아래 코드는 $(function() {}); 밖에 작성합니다.
/* 회원 아이디, 비밀번호, 비밀번호 확인, 이메일 아이디 폼 컨트롤에
 * 사용자가 입력한 값이 영문 대소문자, 숫자 만 입력되도록 수정하는 함수
 */
function inputCharReplace() {
    // 아래와 같이 정규표현식을 이용해 영문 대소문자, 숫자만 입력되었는지 체크할 수 있다.
    var regExp = /^[A-Za-z0-9]/gi;
    if(regExp.test($(this).val())) {
        alert("영문 대소문자, 숫자만 입력할 수 있습니다.");
        $(this).val($(this).val().replace(regExp, ""));
    }
}

/* 이메일 도메인 입력 폼 컨트롤에 사용자가 입력한 값이
 * 영문 대소문자, 숫자, 점(.)만 입력되도록 수정하는 함수
 */
function inputEmailDomainReplace() {
    var regExp = /^[a-z0-9\.\.]/gi;
    if(regExp.test($(this).val())) {
        alert("이메일 도메인은 영문 소문자, 숫자, 점(.)만 입력할 수 있습니다.");
        $(this).val($(this).val().replace(regExp, ""));
    }
}
}
```

## ▶ Controller 클래스

앞에서 회원가입 폼에서 “중복확인” 버튼이 클릭되면 자바스크립트를 이용해 새 창을 띄워서 회원 아이디 중복 확인 요청을 서버로 보내는 코드를 작성했다. 이 요청을 컨트롤러에서 받아서 뷰 페이지만 화면에 띄워주는 코드를 작성하여 실행해 본 후에 본격적으로 회원 아이디 중복 확인 기능을 구현해 보자. MemberController 클래스에 회원 아이디 중복 확인 요청을 처리하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.controller.MemberController에 추가

```
@GetMapping("/overlapIdCheck")
public String overlapIdCheck(Model model, @RequestParam("id") String id) {

    // model에 회원 ID를 저장한다.
    model.addAttribute("id", id);

    // 뷰 페이지만 먼저 작성해 새 창으로 잘 표시되는지 확인해 보자.
    return "member/overlapIdCheck.html";
}
```

## ▶ 회원 아이디 중복 확인 View

회원 아이디 중복 확인 요청의 결과를 새 창으로 출력할 뷰 페이지를 다음과 같이 작성하고 실행해 보자.

- src/main/resources/templates/member/overlapIdCheck.html

```
<!--/*
```

회원 가입시 아이디 중복검사 요청에 대한 처리 결과를 출력할 View 페이지  
이 페이지는 새창으로 실행되고 중복 아이디 체크를 할 수 있는 폼을 제공한다.

회원 가입 폼에서 아이디 중복확인 버튼을 클릭하면 이 페이지만 새 창으로  
보여야 하므로 레이아웃 템플릿은 사용하지 않고 단독으로 실행될 수 있도록 했다.

```
*/-->
```

```
<!DOCTYPE html>
```

```
<html xmlns:th="http://www.thymeleaf.org">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<link th:href="@{bootstrap/bootstrap.min.css}" rel="stylesheet" >
```

```
<script th:src="@{js/jquery-3.7.1.min.js}"></script>
```

```
<script th:src="@{js/member.js}"></script>
```

```
<title>중복 아이디 체크</title>
```

```
</head>
```

```
<body>
```

```
<div class="row my-5" id="global-content">
```

```
<div class="col">
```

```
<div class="row text-center">
```

```
<div class="col">
```

```
<h2 class="fs-3 fw-bold">사용할 수 없는 아이디</h2>
```

```
</div>
```

```
</div>
```

```
<div class="row my-3 text-center">
```

```
<div class="col">
```

```
입력하신 [{id}]는 이미 사용 중인 아이디 입니다.
```

```
</div>
```

```
</div>
```

```
<div class="row my-3">
```

```
<div class="col text-center">
```

```
다른 아이디를 선택해 주세요
```

```
</div>
```

```
</div>
```

```
<form action="overlapIdCheck" name="idCheckForm"
```

```
method="post" id="idCheckForm" class="row mt-5">
```

```
<div class="col-10 offset-1">
```

```
<div class="input-group">
```

```
<span class="input-group-text">* 아이디 : </span>
```

```

        <input type="text" class="form-control" name="id" id="checkId">
        <input type="submit" class="btn btn-primary" value="중복확인">
    </div>
</div>
</form>
</div>
</div>
</body>
</html>

```

프로젝트를 실행하고 회원가입 메뉴로 회원가입 폼을 화면에 띄운 후에 회원 ID를 입력하고 “중복 확인” 버튼을 클릭하면 다음 그림과 같이 새 창으로 실행되는 것을 볼 수 있을 것이다.

이제 여기에서 회원이 입력한 아이디를 member 테이블에 있는 회원 정보와 비교해서 중복 아이디가 존재하는지 확인하는 기능을 추가해 보자.



## ▶ Persistence 계층 구현

Persistence 계층은 앞에서 회원 로그인 기능을 구현할 때 사용한 `getMember()` 메서드와 맵핑 구문을 그대로 사용하면 되므로 회원 아이디 중복 확인 기능에서는 별도로 추가할 코드가 없다.

## ▶ Business 계층 구현

`MemberService` 클래스에 회원 아이디 중복을 체크하는 메서드를 다음과 같이 추가하자.

### - `com.springbootstudy.bbs.service.MemberService`

```

// 회원 가입시 아이디 중복을 체크하는 메서드
public boolean overlapIdCheck(String id) {
    Member member = memberMapper.getMember(id);
    log.info("overlapIdCheck - member : " + member);
    if(member == null) {

```

```

        return false;
    }
    return true;
}

```

## ▶ Controller 클래스

앞에서 작성한 MemberController 클래스에서 앞에서 작성한 회원 아이디 중복 확인 요청을 처리하는 overlapIdCheck() 메서드를 다음과 같이 수정하자.

### - com.springbootstudy.bbs.controller.MemberController에서 메서드 수정

```

/* 회원가입 폼에서 들어오는 아이디 중복 체크 요청을 처리하는 메서드
 * 새 창으로 요청할 때는 GET 방식 요청을 보내지만 아이디가 이미 사용중이라면
 * 새 창에서 폼을 통해 POST 방식 요청을 보내기 때문에 두 가지 방식 모두를 이
 * 메서드에서 처리하기 위해서 @RequestMapping 애노테이션을 사용하였다.
 * @RequestMapping에 method 속성을 지정하지 않으면 "/overlapIdCheck"로
 * 들어오는 GET과 POST 방식 요청을 모두 이 메서드가 받을 수 있다.
 */
@RequestMapping("/overlapIdCheck")
//@GetMapping("/overlapIdCheck")
public String overlapIdCheck(Model model, @RequestParam("id") String id) {

    // 회원 아이디 중복 여부를 받아 온다.
    boolean overlap = memberService.overlapIdCheck(id);

    // model에 회원 ID와 회원 ID 중복 여부를 저장한다.
    model.addAttribute("id", id);
    model.addAttribute("overlap", overlap);

    return "member/overlapIdCheck.html";
}

```

## ▶ 회원 아이디 중복 확인 View

앞에서 작성한 회원 아이디 중복 확인 요청의 결과를 새 창으로 출력할 뷰 페이지를 다음과 같이 수정하자.

### - src/main/resources/templates/member/overlapIdCheck.html 파일 수정

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <link th:href="@{bootstrap/bootstrap.min.css}" rel="stylesheet" >
    <script th:src="@{js/jquery-3.7.1.min.js}"></script>
    <script th:src="@{js/member.js}"></script>

```

```

<title>중복 아이디 체크</title>
</head>
<body>
  <div class="row my-5" id="global-content">
    <div class="col" th:if="{overlap}">
      <div class="row text-center">
        <div class="col">
          <h2 class="fs-3 fw-bold">사용할 수 없는 아이디</h2>
        </div>
      </div>
      <div class="row my-3 text-center">
        <div class="col">
          입력하신 [{id}]는 이미 사용 중인 아이디 입니다.
        </div>
      </div>
      <div class="row my-3">
        <div class="col text-center">
          다른 아이디를 선택해 주세요
        </div>
      </div>
      <form action="overlapIdCheck" name="idCheckForm"
        method="post" id="idCheckForm" class="row mt-5">
        <div class="col-10 offset-1">
          <div class="input-group">
            <span class="input-group-text">* 아이디 : </span>
            <input type="text" class="form-control" name="id" id="checkId">
            <input type="submit" class="btn btn-primary" value="중복확인">
          </div>
        </div>
      </form>
    </div>
    <div class="col" th:unless="{overlap}">
      <div class="row text-center">
        <div class="col">
          <h2 class="fs-3 fw-bold">사용할 수 있는 아이디</h2>
        </div>
      </div>
      <div class="row my-3 text-center">
        <div class="col">
          입력하신 [{id}]는 사용할 수 있는 아이디 입니다.
        </div>
      </div>
      <div class="row mt-5">
        <div class="col text-center">
          <input type="button" th:value="{id}을(를) 아이디로 사용하기"

```

```

        id="btnIdCheckClose" th:data-id-value="${id}" class="btn btn-primary"/>
    </div>
</div>
</div>
</div>
</body>
</html>

```

## ▶ 회원 관련 자바스크립트 파일

member.js 파일에 회원 아이디 중복 확인 결과를 표시하는 overlapIdCheck.html 파일에서 사용하는 폼 유효성 검사 코드와 “아이디 사용 버튼”에 대한 이벤트 처리 코드 그리고 회원가입 폼에서 사용하는 이벤트 처리 코드, 유효성 검사 코드와 다음 우편번호 API를 사용해 우편번호 찾기 기능을 수행하는 자바스크립트 코드를 다음과 같이 작성하자.

### - src/main/resources/static/js/member.js

// DOM이 준비되면 실행될 콜백 함수

```
$(function() {
```

... 중 략 ...

/\* 새 창으로 띄운 아이디 찾기 폼에서

\* 아이디 중복확인 버튼이 클릭되면 유효성 검사를 하는 함수

\*/

```
$("#idCheckForm").on("submit", function() {
```

```
    var id = $("#checkId").val();
```

```
    if(id.length == 0) {
```

```
        alert("아이디를 입력해주세요");
```

```
        return false;
```

```
    }
```

```
    if(id.length < 5) {
```

```
        alert("아이디는 5자 이상 입력해주세요.");
```

```
        return false;
```

```
    }
```

```
});
```

/\* 새 창으로 띄운 아이디 중복확인 창에서 "아이디 사용 버튼"이 클릭되면

\* 새 창을 닫고 입력된 아이디를 부모창의 회원가입 폼에 입력해 주는 함수

\*/

```
$("#btnIdCheckClose").on("click", function() {
```

```
    var id = $(this).attr("data-id-value");
```

```
    opener.document.joinForm.id.value = id;
```

```
    opener.document.joinForm.isIdCheck.value = true;
```

```

    window.close();
});

/* 회원 가입 폼과 회원정보 수정 폼에서 "우편번호 검색" 버튼의 클릭 이벤트 처리
 * findZipcode() 함수는 다음 우편번호 API를 사용해 우편번호를 검색하는 함수로
 * 두 페이지에서 사용되어 중복된 코드가 발생하므로 아래에 별도의 함수로 정의하였다.
 */
$("#btnZipcode").click(findZipcode);

// 이메일 입력 셀렉트 박스에서 선택된 도메인을 설정하는 함수
$("#selectDomain").on("change", function() {
    var str = $(this).val();

    if(str == "직접입력") {
        $("#emailDomain").val("");
        $("#emailDomain").prop("readonly", false);
    } else if(str == "네이버"){
        $("#emailDomain").val("naver.com");
        $("#emailDomain").prop("readonly", true);

    } else if(str == "다음") {
        $("#emailDomain").val("daum.net");
        $("#emailDomain").prop("readonly", true);

    } else if(str == "한메일"){
        $("#emailDomain").val("hanmail.net");
        $("#emailDomain").prop("readonly", true);

    } else if(str == "구글") {
        $("#emailDomain").val("gmail.com");
        $("#emailDomain").prop("readonly", true);
    }
});

// 회원 가입 폼이 서브밋 될 때 이벤트 처리 - 폼 유효성 검사
$("#joinForm").on("submit", function() {

    /* 회원 가입 폼에서 서브밋 이벤트가 발생하면 true를 인수로 지정한다.
     * 회원 가입 폼과 회원 정보 수정 폼 유효성 검사를 하는 기능도 중복 되는
     * 코드가 많으므로 joinFormCheck()라는 별도의 함수를 만들어서 사용하였다.
     * joinFormChcek() 함수에서 폼 유효성 검사를 통과하지 못하면
     * false가 반환되기 때문에 그대로 반환하면 폼이 서브밋 되지 않는다.
     */
    return joinFormCheck(true);
});

```



```
});  
});
```

```
/* 회원 가입 폼과 회원정보 수정 폼의 유효성 검사를 하는 함수  
 * 두 페이지에서 처리하는 코드가 중복되어 하나의 함수로 정의하였다.  
 **/
```

```
function joinFormCheck(isJoinForm) {  
    var name = $("#name").val();  
    var id = $("#id").val();  
    var pass1 = $("#pass1").val();  
    var pass2 = $("#pass2").val();  
    var zipcode = $("#zipcode").val();  
    var address1 = $("#address1").val();  
    var emailId = $("#emailId").val();  
    var emailDomain = $("#emailDomain").val();  
    var mobile2 = $("#mobile2").val();  
    var mobile3 = $("#mobile2").val();  
    var isIdCheck = $("#isIdCheck").val();  
  
    if(name.length == 0) {  
        alert("이름이 입력되지 않았습니다.\n이름을 입력해주세요");  
        return false;  
    }  
    if(id.length == 0) {  
        alert("아이디가 입력되지 않았습니다.\n아이디를 입력해주세요");  
        return false;  
    }  
    if(isJoinForm && isIdCheck == 'false') {  
        alert("아이디 중복 체크를 하지 않았습니다.\n아이디 중복 체크를 해주세요");  
        return false;  
    }  
    if(pass1.length == 0) {  
        alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");  
        return false;  
    }  
  
    if(pass2.length == 0) {  
        alert("비밀번호 확인이 입력되지 않았습니다.\n비밀번호 확인을 입력해주세요");  
        return false;  
    }  
    if(pass1 != pass2) {  
        alert("비밀번호와 비밀번호 확인이 일치하지 않습니다.");  
        return false;  
    }  
    if(zipcode.length == 0) {
```

```

    alert("우편번호가 입력되지 않았습니다.\n우편번호를 입력해주세요");
    return false;
}
if(address1.length == 0) {
    alert("주소가 입력되지 않았습니다.\n주소를 입력해주세요");
    return false;
}
if(emailId.length == 0) {
    alert("이메일 아이디가 입력되지 않았습니다.\n이메일 아이디를 입력해주세요");
    return false;
}
if(emailDomain.length == 0) {
    alert("이메일 도메인이 입력되지 않았습니다.\n이메일 도메인을 입력해주세요");
    return false;
}
if(mobile2.length == 0 || mobile3.length == 0) {
    alert("휴대폰 번호가 입력되지 않았습니다.\n휴대폰 번호를 입력해주세요");
    return false;
}
}

```

/\* 우편번호 찾기 - daum 우편번호 찾기 API 이용

\* 회원 가입 폼과 회원정보 수정 폼에서 "우편번호 검색" 버튼이 클릭되면 호출되는 함수

\*

\* 새로운 5자리 우편번호로 회원 주소를 입력 받기 위해 daum.net에서

\* 제공하는 우편번호 찾기 API를 사용하였다.

\* 참고 사이트 : <http://postcode.map.daum.net/guide>

\*\*/

```
function findZipcode() {
```

```
    new daum.Postcode({
```

```
        oncomplete: function(data) {
```

```
            // 우편번호 검색 결과 항목을 클릭했을때 실행할 코드를 여기에 작성한다.
```

```
            // 각 주소의 노출 규칙에 따라 주소를 조합한다.
```

```
            // 내려오는 변수가 값이 없는 경우엔 공백('')값을 가지므로, 이를 참고하여 분기한다.
```

```
            var addr = ''; // 주소 변수
```

```
            var extraAddr = ''; // 참고 항목 변수
```

```
            //사용자가 선택한 주소 타입과 상관없이 모두 도로명 주소로 처리
```

```
            addr = data.roadAddress;
```

```
            // 법정동명이 있을 경우 추가한다. (법정리는 제외)
```

```
            // 법정동의 경우 마지막 문자가 "동/로/가"로 끝난다.
```

```
            if(data.bname !== '' && /[동|로|가]$/g.test(data.bname)){
```

```
                extraAddr += data.bname;
```

```
            }
```

```

// 건물명이 있고, 공동주택일 경우 추가한다.
if(data.buildingName != '' && data.apartment === 'Y'){
    extraAddr += (extraAddr != '' ?
        ', ' + data.buildingName : data.buildingName);
}

// 표시할 참고 항목이 있을 경우, 괄호까지 추가한 최종 문자열을 만든다.
if(extraAddr != ''){
    extraAddr = ' (' + extraAddr + '>';
}

// 조합된 참고 항목을 상세주소에 추가한다.
addr += extraAddr;

// 우편번호와 주소 정보를 해당 입력상자에 출력한다.
$("#zipcode").val(data.zonecode);
$("#address1").val(addr);

// 커서를 상세주소 입력상자로 이동한다.
$("#address2").focus();
}
}).open();
}

```

## 2-3) 회원가입 요청 처리

회원가입 폼에서 회원 정보를 입력하고 “가입하기” 버튼을 클릭하여 폼에 입력된 정보를 서버로 전송하면 서버에서는 회원 정보를 DB에 저장한 후 회원 로그인 폼으로 리다이렉트 하도록 구현할 것이다.

### ▶ Persistence 계층 구현

MyBatis를 활용하여 회원 정보를 member 테이블에 저장하는 매퍼 인터페이스와 매퍼 XML을 다음과 같이 작성하자.

#### ▣ 회원 관련 SQL을 분리한 매퍼 XML 파일

다음 코드를 참고해 MemberMapper.xml 파일에 member 테이블에 회원 정보를 저장하는 매퍼 구문을 작성하자.

- src/main/resources/mappers/MemberMapper.xml

```
<!--
```

회원 정보를 추가하는 맵핑 구문

-->

```
<insert id="addMember" parameterType="Member">
  INSERT INTO member
  VALUES(#{id}, #{name}, #{pass}, #{email},
    #{mobile}, #{zipcode}, #{address1}, #{address2},
    #{phone}, #{emailGet}, SYSDATE())
</insert>
```

## ■ 회원 관련 DB작업을 수행하는 매퍼 인터페이스

MemberMapper 인터페이스에 회원 정보를 member 테이블에 저장하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.mapper.MemberMapper

```
// 회원 정보를 회원 테이블에 저장하는 메서드
public void addMember(Member member);
```

## ▶ Business 계층 구현

MemberService 클래스에 MemberMapper를 사용해 회원 정보를 DB에 저장하는 메서드를 다음과 같이 작성하자. MemberService에서는 회원 정보를 저장할 때 PasswordEncoder를 이용해 비밀번호를 암호화하는 작업을 수행한 후 회원 정보를 저장해야 한다.

### - com.springbootstudy.bbs.service.MemberService

```
// 회원 정보를 MemberMapper를 이용해 회원 테이블에 저장하는 메서드
public void addMember(Member member) {

    // BCryptPasswordEncoder 객체를 이용해 비밀번호를 암호화한 후 저장
    member.setPass(passwordEncoder.encode(member.getPass()));

    /* 아래는 문자열 "1234"를 BCryptPasswordEncoder 객체를 사용해
     * 암호화한 것으로 동일한 문자열을 암호화 하더라도 그 결과는 모두 다를 수 있다.
     */
    // $2a$10$aWYm2BGI/0iMuemBeF4Y8.7WZeVKAoudv/VzgQx697IYlZgQxr/pe
    // $2a$10$b3t8sn6QZGHYarX3OS5KUuPxzWZdY5yHPRxlSdAgByQ7v0BICLzrO
    // $2a$10$.g6l.wyIF01.j4u4gvVtKONG9ACBUT1GRlDwlMZcjBxZPrCAURLaG
    // $2a$10$137iJWozST9.2El11vjQOmSk9rus.5cawhTiPuQagCzVNTZcoFDa
    // $2a$10$qtXKvlgk.URhyqgx93KYFuw4e8Rh3lhIkR0CTZhd.HazLal7d3rqK
    log.info(member.getPass());
    memberMapper.addMember(member);
}
```

## ▶ Controller 클래스

MemberController에 회원가입 폼에서 들어오는 회원가입 요청을 처리하는 메서드를 추가하자.

- **com.springbootstudy.bbs.controller.MemberController**

```
// 회원가입 폼에서 들어오는 회원가입 요청을 처리하는 메서드
@PostMapping("/joinResult")
public String joinResult(Model model, Member member,
    @RequestParam("pass1") String pass1,
    @RequestParam("emailId") String emailId,
    @RequestParam("emailDomain") String emailDomain,
    @RequestParam("mobile1") String mobile1,
    @RequestParam("mobile2") String mobile2,
    @RequestParam("mobile3") String mobile3,
    @RequestParam("phone1") String phone1,
    @RequestParam("phone2") String phone2,
    @RequestParam("phone3") String phone3,
    @RequestParam(value="emailGet", required=false,
        defaultValue="false")boolean emailGet) {

    member.setPass(pass1);
    member.setEmail(emailId + "@" + emailDomain);
    member.setMobile(mobile1 + "-" + mobile2 + "-" + mobile3);

    if(phone2.equals("") || phone3.equals("")) {
        member.setPhone("");
    } else {
        member.setPhone(phone1 + "-" + phone2 + "-" + phone3);
    }
    member.setEmailGet(Boolean.valueOf(emailGet));

    // MemberService를 통해서 회원 가입 폼에서 들어온 데이터를 DB에 저장한다.
    memberService.addMember(member);
    log.info("joinResult : " + member.getName());

    // 로그인 폼으로 리다이렉트 시킨다.
    return "redirect:loginForm";
}
```

### 3) 회원 정보 수정 구현

회원 정보 수정 기능은 회원 가입과는 다르게 기존의 회원 정보를 폼에 보여주고 회원이 자신의 정보를 확인하여 수정할 내용을 수정한 후 수정하기 버튼을 클릭하면 수정된 정보를 받아서 DB에서 수정되도록 구현해야 한다. 그러므로 회원 정보 수정 폼을 보여주는 요청 처리와 회원 정보 수정 폼에서 들어오는 회원 정보 수정 요청 처리를 나눠서 구현해야 한다. 그리고 회원 정보 수정 폼에서 비밀번호 수정은 기존의 비밀번호를 입력하여 “비밀번호 확인”을 먼저 한 후에 비밀번호를 수정할 수 있도록 구현할 것이다. 이때 비밀번호 확인 기능은 Ajax를 활용해 구현할 것이다.

#### 3-1) 회원 정보 수정 폼 요청 처리

회원 정보 수정 폼에 출력해야 할 데이터는 회원 로그인을 구현할 때 세션 영역에 회원 정보를 저장했기 때문에 MemberMapper.xml 파일과 MemberMapper 인터페이스에 새로 추가할 기능은 없고 MemberController 클래스에 회원 정보 수정 폼 요청을 처리하는 메서드만 작성하고 회원 정보 수정 폼에서 세션에 저장된 회원 정보를 출력하면 된다.

##### ▶ Controller 클래스

MemberController 클래스에 회원 정보 수정 폼 요청을 처리하는 메서드를 다음과 같이 추가하자.

##### - com.springbootstudy.bbs.controller.MemberController에 추가

```
// 회원 정보 수정 폼 요청을 처리하는 메서드
@GetMapping("/memberUpdateForm")
public String updateForm(Model model, HttpSession session) {

    /* 로그인 처리를 할 때 세션 영역에 회원 정보를 저장했기 때문에 뷰의 정보만 반환한다.
     * 이렇게 별도의 처리가 필요 없을 경우 뷰 전용 컨트롤러를 사용하는 것도 좋다.
     */
    return "member/memberUpdateForm";
}
```

##### ▶ 회원 정보 수정 폼 View

##### - src/main/resources/templates/member/memberUpdateForm.html

```
<!DOCTYPE html>
<!--/*
    Thymeleaf와 Thymeleaf Layout을 사용하기 위한 NameSpace를 정의한다.
    layout:decorate 옵션은 아래의 <th:block layout:fragment="content">
    부분을 어떤 레이아웃에 적용할지 설정하는 것으로 경로 지정은
    templates 폴더를 기준으로 지정하면 된다.
*/-->
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
```

```

    layout:decorate=~{layouts/main_layout}">
<!--/*
    block을 사용해도 되고 div에 직접 layout:fragment를 적용해도 된다.
    main_layout.html에서 지정한 layout:fragment의 이름과 같아야 한다.
*/-->
<th:block layout:fragment="content">
<!--
    새로운 5자리 우편번호로 회원 주소를 입력 받기 위해 daum.net에서
    제공하는 우편번호 찾기 API를 사용하였다.
    참고 사이트 : http://postcode.map.daum.net/guide
-->
<script src=
"https://t1.daumcdn.net/mapjsapi/bundle/postcode/prod/postcode.v2.js"></script>
<div class="row my-5" id="global-content">
    <div class="col">
        <div class="row my-3 text-center">
            <div class="col">
                <h2 class="fs-3 fw-bold">회원 정보 수정</h2>
            </div>
        </div>
        <form action="memberUpdateResult" name="memberUpdateForm"
            method="post" id="memberUpdateForm">
            <div class="row mt-5 mb-3">
                <div class="col-8 offset-2">
                    <label for="name" class="form-label">* 이름 : </label>
                    <input type="text" class="form-control" name="name" id="name"
                        th:value="${session.member.name}" readonly>
                </div>
            </div>
            <div class="row my-3">
                <div class="col-8 offset-2">
                    <label for="userId" class="form-label">* 아이디 : </label>
                    <div class="row">
                        <div class="col">
                            <input type="text" class="form-control" name="id" id="id"
                                th:value="${session.member.id}" readonly>
                        </div>
                    </div>
                </div>
            </div>
            <div class="row my-3">
                <div class="col-8 offset-2">
                    <label for="oldPass" class="form-label">* 기존 비밀번호 : </label>
                    <div class="row">
                        <div class="col-6">

```

```

        <input type="password" class="form-control" name="oldPass"
            id="oldPass">
    </div>
    <div class="col-4">
        <input type="button" class="btn btn-warning" id="btnPassCheck"
            value="비밀번호 확인">
    </div>
</div>
</div>
<div class="row my-3">
    <div class="col-8 offset-2">
        <label for="pass1" class="form-label">* 새 비밀번호 : </label>
        <input type="password" class="form-control" name="pass1" id="pass1">

    </div>
</div>
<div class="row my-3">
    <div class="col-8 offset-2">
        <label for="pass2" class="form-label">* 새 비밀번호 확인 : </label>
        <input type="password" class="form-control" name="pass2" id="pass2">
    </div>
</div>
<div class="row my-3">
    <div class="col-8 offset-2">
        <label for="zipcode" class="form-label">* 우편번호 : </label>
        <div class="row">
            <div class="col-4">
                <input type="text" class="form-control" name="zipcode" id="zipcode"
                    maxlength="5" readonly th:value="{session.member.zipcode}">
            </div>
            <div class="col-4">
                <input type="button" class="btn btn-warning" id="btnZipcode"
                    value="우편번호 찾기">
            </div>
        </div>
    </div>
</div>
<div class="row my-3">
    <div class="col-8 offset-2">
        <label for="address1" class="form-label">* 자택주소 : </label>
        <input type="text" class="form-control" name="address1" id="address1"
            readonly th:value="{session.member.address1}">
    </div>
</div>

```



```

<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="address2" class="form-label">상세주소 : </label>
    <input type="text" class="form-control" name="address2" id="address2"
      th:value="${session.member.address2}">
    </div>
  </div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="emailId" class="form-label">* 이 메 일 : </label>
    <div class="row">
      <div class="col-md-4">
        <input type="text" class="form-control" name="emailId" id="emailId"
          th:value="${session.member.email.split('@')[0]}">
      </div> @
      <div class="col-md-4">
        <input type="text" class="form-control" name="emailDomain"
          id="emailDomain" th:value="${session.member.email.split('@')[1]}">
      </div>
      <div class="col-md-3">
        <select class="form-select" name="selectDomain" id="selectDomain">
          <option>직접입력</option>
          <option>네이버</option>
          <option>다음</option>
          <option>한메일</option>
          <option>구글</option>
        </select>
      </div>
    </div>
  </div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="mobile2" class="form-label">* 휴 대 폰 : </label>
    <div class="row">
      <div class="col-md-3">
        <select class="form-select" name="mobile1" id="mobile1">
          <option th:selected="${session.member.mobile.split('-')[0] == '010'}">
            010</option>
          <option th:selected="${session.member.mobile.split('-')[0] == '011'}">
            011</option>
          <option th:selected="${session.member.mobile.split('-')[0] == '016'}">
            016</option>
          <option th:selected="${session.member.mobile.split('-')[0] == '017'}">
            017</option>
        </select>
      </div>
    </div>
  </div>

```

```

        <option th:selected="${session.member.mobile.split('-')[0] == '018'}">
            018</option>
        <option th:selected="${session.member.mobile.split('-')[0] == '019'}">
            019</option>
    </select>
</div>-
<div class="col-md-4">
    <input type="text" class="form-control" name="mobile2" id="mobile2"
        maxlength="4" th:value="${session.member.mobile.split('-')[1]}">
</div>-
<div class="col-md-4">
    <input type="text" class="form-control" name="mobile3" id="mobile3"
        maxlength="4" th:value="${session.member.mobile.split('-')[2]}">
</div>
</div>
</div>
</div>
<div class="row my-3">
    <div class="col-8 offset-2">
        <label class="form-label">메일 수신여부 : </label>
        <div class="row">
            <div class="col-md-3">
                <div class="form-check">
                    <input type="radio" class="form-check-input" name="emailGet"
                        id="emailOk" value="true" th:checked="${session.member.emailGet}">
                    <label class="form-check-label" for="emailOk">수신함</label>
                </div>
            </div>
            <div class="col-md-3">
                <div class="form-check">
                    <input type="radio" class="form-check-input" name="emailGet"
                        id="emailNo" value="false" th:checked="${!session.member.emailGet}">
                    <label class="form-check-label" for="emailNo">수신않함</label>
                </div>
            </div>
        </div>
    </div>
    <div class="row my-3">
        <div class="col-8 offset-2">
            <label for="phone2" class="form-label">자택전화 : </label>
            <div class="row">
                <div class="col-md-3">
                    <select class="form-select" name="phone1" id="phone1">
                        <option th:selected="${! #strings.isEmpty(session.member.phone)}

```

```

? session.member.phone.split('-')[0] == '02' : false}">
02</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '031' : false}">
031</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '032' : false}">
032</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '033' : false}">
033</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '041' : false}">
041</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '042' : false}">
042</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '043' : false}">
043</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '044' : false}">
044</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '051' : false}">
051</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '052' : false}">
052</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '053' : false}">
053</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '054' : false}">
054</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '055' : false}">
055</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '061' : false}">
061</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
? session.member.phone.split('-')[0] == '062' : false}">
062</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)

```

```

        ? session.member.phone.split('-')[0] == '063' : false}">
063</option>
<option th:selected="${! #strings.isEmpty(session.member.phone)
        ? session.member.phone.split('-')[0] == '064' : false}">
064</option>
</select>
</div> -
<div class="col-md-4">
    <input type="text" class="form-control" name="phone2" id="phone2"
        maxlength="4"
        th:value="${#strings.length(session.member.phone) > 0
        ? session.member.phone.split('-')[1] : ''}">
</div> -
<div class="col-md-4">
    <input type="text" class="form-control" name="phone3" id="phone3"
        maxlength="4"
        th:value="${#strings.length(session.member.phone) > 0
        ? session.member.phone.split('-')[2] : ''}">
</div>
</div>
</div>
</div>
<div class="row mb-3 mt-5">
    <div class="col-8 offset-2">
        <input type="submit" value="수정하기" class="btn btn-primary">
    </div>
</div>
</form>
</div>
</div>
</th:block>
</html>

```

### 3-2) 회원 비밀번호 확인 요청 처리

회원 정보 수정에서 비밀번호 변경을 위해서 기존 비밀번호를 확인하는 기능을 추가할 것이다. 위에서 작성한 회원 정보 수정 폼에서 “비밀번호 변경” 버튼이 클릭되면 비밀번호가 입력되었는지 체크한 후 비밀번호가 입력되었으면 자바스크립트를 이용해 Ajax 요청을 서버에 보내서 비밀번호가 맞는지 확인하는 기능을 구현해 보자.

#### ▶ 회원 관련 자바스크립트 파일

member.js 파일에 회원 정보 수정 폼에서 “비밀번호 확인” 버튼이 클릭될 때 이벤트 처리 코드와 회원 정보 수정 폼 유효성 검사 코드를 다음과 같이 작성하고 테스트해보자.

- src/main/resources/static/js/member.js

// DOM이 준비되면 실행될 콜백 함수

\$(function() {

... 중 략 ...

/\* 회원정보 수정 폼에서 "비밀번호 확인" 버튼이 클릭될 때 이벤트 처리

\* 회원정보 수정 폼에서 기존 비밀번호가 맞는지를 Ajax 통신을 통해 확인한다.

\*/

\$("#btnPassCheck").click(function() {

var oldId = \$("#id").val();

var oldPass = \$("#oldPass").val();

if(\$.trim(oldPass).length == 0) {

alert("기존 비밀번호가 입력되지 않았습니다.\n기존 비밀번호를 입력해주세요");

return false;

}

var data = "id=" + oldId + "&pass="+oldPass;

console.log("data : " + data);

\$.ajax({

"url": "passCheck.ajax",

"type": "get",

"data": data,

"dataType": "json",

"success": function(resData) {

if(resData.result) {

alert("비밀번호가 확인되었습니다.\n비밀번호를 수정해주세요");

\$("#btnPassCheck").prop("disabled", true);

\$("#oldPass").prop("readonly", true);

\$("#pass1").focus();

} else {

alert("비밀번호가 틀립니다.\n비밀번호를 다시 확인해주세요");

\$("#oldPass").val("").focus();

}

},

"error": function(xhr, status) {

console.log("error : " + status);

}

});

});

// 회원정보 수정 폼에서 수정하기 버튼이 클릭되면 유효성 검사를 하는 함수

```

$("#memberUpdateForm").on("submit", function() {

    /* 회원정보 수정 폼에서 "비밀번호 확인" 버튼이 disabled 상태가 아니면
    * 기존 비밀번호를 확인하지 않았기 때문에 확인하라는 메시지를 띄운다.
    */
    if(! $("#btnPassCheck").prop("disabled")) {
        alert("기존 비밀번호를 확인해야 비밀번호를 수정할 수 있습니다.\n"
            + "기존 비밀번호를 입력하고 비밀번호 확인 버튼을 클릭해 주세요");
        return false;
    }

    /* 회원정보 수정 폼에서 서브밋 이벤트가 발생하면 false를 인수로 지정한다.
    * joinFormChcek() 함수에서 폼 유효성 검사를 통과하지 못하면
    * false가 반환되기 때문에 그대로 반환하면 폼이 서브밋 되지 않는다.
    */
    return joinFormCheck(false);
});
});

```

## ▶ Persistence 계층 구현

MyBatis를 활용하여 회원 ID에 해당하는 비밀번호를 DB 테이블에서 읽어오는 매퍼 XML과 매퍼 인터페이스를 다음과 같이 작성하자.

### ■ 회원 관련 SQL을 분리한 매퍼 XML 파일

다음은 참고해 MemberMapper.xml 파일에 회원 ID에 해당하는 비밀번호를 DB 테이블에서 읽어오는 매퍼 구문을 추가하자.

#### - src/main/resources/mappers/MemberMapper.xml

```

<!--
    회원 테이블에서 id에 해당하는 비밀번호를 가져오는 매퍼 구문
-->
<select id="memberPassCheck" resultType="String">
    SELECT
        pass
    FROM member
    WHERE id = #{id}
</select>

```

### ■ 회원 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 회원 ID에 해당하는 비밀번호를 DB 테이블에서 읽어와 반환하는 메서드를 다음과 같이 작성하자

- **com.springbootstudy.bbs.mapper.MemberMapper**

```
// 회원 정보 수정 시에 기존 비밀번호가 맞는지 체크하는 메서드
public String memberPassCheck(String id);
```

▶ **Business 계층 구현**

MemberService 클래스에 PasswordEncoder를 사용해 회원 정보 수정 폼에서 입력한 비밀번호와 DB에 저장된 암호화된 비밀번호가 일치하는지 확인하여 그 결과를 반환하는 메서드를 다음과 같이 추가하자.

- **com.springbootstudy.bbs.service.MemberService**

```
// 회원 정보 수정 시에 기존 비밀번호가 맞는지 체크하는 메서드
public boolean memberPassCheck(String id, String pass) {

    String dbPass = memberMapper.memberPassCheck(id);
    boolean result = false;

    /* 비밀번호가 맞으면 true를 반환하도록 작성한다.
     * BCryptPasswordEncoder 객체의 matches 메소드를 이용해 암호가 맞는지 확인
     * matches() 메소드의 첫 번째 인수로 인코딩이 안된 문자열, 두 번째 인수로 인코딩된
     * 문자열을 지정하면 두 문자열의 원본 데이터가 같을 경우 true를 반환해 준다.
     */
    if(passwordEncoder.matches(pass, dbPass)) {
        result = true;
    }
    return result;
}
```

▶ **Ajax 요청처리 Controller 클래스**

프로젝트에 com.springbootstudy.bbs.ajax 패키지를 추가하고 이 패키지에 Ajax 요청을 처리하는 AjaxProcessController 클래스 새롭게 추가하자. 그리고 회원 정보 수정 폼에서 회원의 기존 비밀번호가 맞는지 확인하는 Ajax 요청이 들어오면 이를 처리하는 memberPassCheck() 메서드를 다음과 같이 작성하자.

- **com.springbootstudy.bbs.ajax.AjaxProcessController**

```
/* RestController 클래스임을 정의
 * @RestController 애노테이션은 @Controller에 @ResponseBody가
 * 추가된 것과 동일하다. RestController의 주용도는 JSON으로 응답하는 것이다.
 */
@RestController
public class AjaxProcessController {

    @Autowired
    private MemberService memberService;
```

```

/* 비밀번호 확인 요청 처리 메서드
 * @RestController 애노테이션이 클래스에 적용되었기 때문에
 * 이 메서드에서 반환하는 값은 JSON으로 직렬화되어 응답 본문에 포함된다.
 */
@GetMapping("/passCheck.ajax")
public Map<String, Boolean> memberPassCheck(
    @RequestParam("id") String id, @RequestParam("pass") String pass) {

    /* MemberService를 사용해 요청 파라미터로 받은 id와 pass를 입력해
     * 비밀번호가 일치하는지 여부를 받아와 Map 객체에 담아서 반환하면
     * 이 객체를 JSON 형식으로 변환해 응답 본문에 추가해 준다.
     */
    boolean result = memberService.memberPassCheck(id, pass);
    Map<String, Boolean> map = new HashMap<String, Boolean>();
    map.put("result", result);

    /* MappingJackson2HttpMessageConverter에 의해서
     * Map 객체가 아래와 같이 json 형식으로 변환된다.
     *
     * {"result": true} 또는 {"result": false}
     */
    return map;
}
}

```

### 3-3) 회원 정보 수정 요청 처리

회원 정보 수정 폼에서 회원 정보를 수정하고 “수정하기” 버튼을 클릭하여 수정된 회원 정보를 서버로 전송하면 서버에서는 회원 정보를 DB에서 수정한 후 게시글 리스트로 리다이렉트 하도록 구현할 것이다.

#### ▶ Persistence 계층 구현

회원이 수정한 정보를 MyBatis를 활용하여 member 테이블에서 수정하는 매퍼 인터페이스와 매퍼 XML을 다음과 같이 작성하자.

#### ▣ 회원 관련 SQL을 분리한 매퍼 XML 파일

다음 코드를 참고해 MemberMapper.xml 파일에 member 테이블에서 회원 정보를 수정하는 매퍼 구문을 작성하자.

- src/main/resources/mappers/MemberMapper.xml



```

<!--
회원 정보를 수정하는 맵핑 구문
-->
<update id="updateMember" parameterType="Member">
    UPDATE member
    SET pass=#{pass}, email=#{email}, mobile=#{mobile},
        zipcode=#{zipcode}, address1=#{address1}, address2=#{address2},
        phone=#{phone}, email_get=#{emailGet}, reg_date=SYSDATE()
    WHERE id=#{id}
</update>

```

## ▣ 회원 관련 DB작업을 수행하는 매퍼 인터페이스

MemberMapper 인터페이스에 수정된 회원 정보를 받아서 member 테이블에서 수정하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.mapper.MemberMapper

```

// 회원 정보를 회원 테이블에서 수정하는 메서드
public void updateMember(Member member);

```

## ▶ Business 계층 구현

MemberService 클래스에 MemberMapper를 사용해 회원 정보를 member 테이블에서 수정하는 메서드를 다음과 같이 작성하자. MemberService에서는 회원 정보를 수정할 때 새로 수정된 비밀번호를 PasswordEncoder를 이용해 암호화하는 작업을 수행한 후 회원 정보를 저장해야 한다.

### - com.springbootstudy.bbs.service.MemberService

```

// 회원 정보를 MemberMapper를 이용해 회원 테이블에서 수정하는 메서드
public void updateMember(Member member) {

    // BCryptPasswordEncoder 객체를 이용해 비밀번호를 암호화한 후 저장
    member.setPass(passwordEncoder.encode(member.getPass()));
    log.info(member.getPass());

    memberMapper.updateMember(member);
}

```

## ▶ Controller 클래스

MemberController에 회원 정보 수정 폼에서 들어오는 회원 정보 수정 요청을 처리하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.controller.MemberController

```

// 회원 수정 폼에서 들어오는 요청을 처리하는 메서드

```

```

@PostMapping("/memberUpdateResult")
public String memberUpdateInfo(Model model, Member member,
    @RequestParam("pass1") String pass1,
    @RequestParam("emailId") String emailId,
    @RequestParam("emailDomain") String emailDomain,
    @RequestParam("mobile1") String mobile1,
    @RequestParam("mobile2") String mobile2,
    @RequestParam("mobile3") String mobile3,
    @RequestParam("phone1") String phone1,
    @RequestParam("phone2") String phone2,
    @RequestParam("phone3") String phone3,
    @RequestParam(value="emailGet", required=false,
        defaultValue="false")boolean emailGet) {

    member.setPass(pass1);
    member.setEmail(emailId + "@" + emailDomain);
    member.setMobile(mobile1 + "-" + mobile2 + "-" + mobile3);

    if(phone2.equals("") || phone3.equals("")) {
        member.setPhone("");
    } else {
        member.setPhone(phone1 + "-" + phone2 + "-" + phone3);
    }
    member.setEmailGet(Boolean.valueOf(emailGet));

    // MemberService를 통해서 회원 수정 폼에서 들어온 데이터를 DB에서 수정한다.
    memberService.updateMember(member);
    log.info("memberUpdateResult : " + member.getId());

    /* 클래스 레벨에 @SessionAttributes({"member"})
    * 애노테이션을 지정하고 컨트롤러의 메서드에서 아래와 같이 동일한
    * 이름으로 모델에 추가하면 스프링이 세션 영역에 데이터를 저장해 준다.
    */
    model.addAttribute("member", member);

    // 게시글 리스트로 리다이렉트 시킨다.
    return "redirect:boardList";
}

```

## 4) Interceptor를 이용한 로그인 체크 기능 구현

이번 프로젝트에는 앞에서 언급한 것처럼 로그인을 하지 않으면 게시글 리스트만 볼 수 있고 검색을 하거나 게시글 상세보기 같은 기능은 이용할 수 없도록 구현할 것이다. 웹 애플리케이션을 이렇게 동작시키기 위해서는 요청이 들어올 때 컨트롤러 메서드마다 매번 세션을 체크해 로그인 상태인지를 확인해야 한다. 이렇게 로그인 상태를 매번 체크해야한다면 로그인을 체크하는 공통 기능이 여러 곳에서 중복적으로 사용되기 때문에 코드의 중복이 많이 발생한다. 이를 스프링프레임워크가 제공하는 HandlerInterceptor를 사용하면 중복 코드를 줄이고 개발자는 핵심로직에 더 집중할 수 있어 매우 효율적이다.

HandlerInterceptor는 요청 경로마다 접근 제어를 다르게 처리하거나 특정 URL에 접근할 때 공통적으로 처리해야 할 것이 있을 때 주로 사용한다. 로그인 체크와 같이 클라이언트가 요청할 때 매번 체크해야 하는 경우에 HandlerInterceptor를 이용하면 코드의 중복을 최소화 할 수 있고 접근 URL에 따라서 접근 제어를 할 수 있다. 이와 같이 여러 요청 경로나 다수의 컨트롤러에서 공통적으로 처리해야 할 기능이 필요하다면 실무에서는 HandlerInterceptor를 많이 활용한다.

HandlerInterceptor를 사용하면 다음과 같은 시점에 필요한 기능을 적용할 수 있다.

1. 컨트롤러 실행 전
2. 컨트롤러 실행 후 - 아직 뷰를 생성하지 않은 상태
3. 뷰가 생성되고 클라이언트로 전송된 후

HandlerInterceptor 인터페이스에 정의된 추상 메서드는 모두 3개이며 HandlerInterceptor를 구현하고 동작시키기 위해서는 Spring MVC 환경설정 클래스에 Bean으로 등록하고 해당 인터셉터가 동작할 경로 패턴(PathPattern)을 설정해야 한다.

### ▶ HandlerInterceptor 클래스

프로젝트에 `com.springbootstudy.bbs.interceptor` 패키지를 추가하고 이 패키지에 로그인 여부를 체크할 `LoginCheckInterceptor` 클래스 새로 생성한 후 다음 코드를 참고해 HandlerInterceptor 인터페이스를 상속하여 구현하자.

#### - `com.springbootstudy.bbs.interceptor.LoginCheckInterceptor`

```
/* HandlerInterceptor는 요청 경로마다 접근 제어를 다르게 하거나 특정 URL에 접근할 때
 * 공통적으로 처리해야 할 것이 있을 때 주로 사용한다. 앞의 프로젝트에서는 회원이 로그인
 * 상태인지 컨트롤러 메서드마다 세션을 체크해 로그인 상태를 체크 했기 때문에 동일한
 * 코드의 중복을 피할 수 없었다. 하지만 스프링프레임워크가 제공하는 HandlerInterceptor를
 * 구현해 애플리케이션에 적용하면 중복 코드를 줄일 수 있다. 이처럼 여러 요청 경로 또는
 * 여러 컨트롤러에서 공통으로 적용해야 할 기능을 구현할 때 HandlerInterceptor를
 * 사용하면 아주 유용하다.
 *
 * HandlerInterceptor를 사용하면 다음과 같은 시점에 공통 기능을 적용할 수 있다.
 *
 * 1. 컨트롤러 실행 전
 * 2. 컨트롤러 실행 후 - 아직 뷰가 생성되지 않았다.
 * 3. 뷰가 생성되고 클라이언트로 전송된 후
```

```

*
* HandlerInterceptor 인터페이스에 정의된 추상 메서드는 모두 3개이며
* HandlerInterceptor를 구현하고 동작시키기 위해서는 Spring MVC 환경설정
* 클래스에 Bean으로 등록하고 해당 인터셉터가 동작할 경로 패턴(PathPattern)을
* 설정해야 한다.
**/
// 접속자가 로그인 상태인지 체크하는 인터셉터
@Slf4j
public class LoginCheckInterceptor implements HandlerInterceptor {

    /* preHandle() 메서드는 클라이언트의 요청이 들어오고 컨트롤러가 실행되기
    * 전에 공통으로 적용할 기능을 구현할 때 사용한다.
    * 예를 들면 특정 요청에 대해 로그인이 되어 있지 않으면 컨트롤러를 실행하지
    * 않거나 컨트롤러를 실행하기 전에 그 컨트롤러에서 필요한 정보를 생성해
    * 넘겨 줄 필요가 있을 때 preHandler() 메서드를 이용해 구현하면 된다.
    * 이 메서드가 false를 반환하면 다음으로 연결된 HandlerInterceptor
    * 또는 컨트롤러 자체를 실행하지 않게 할 수 있다.
    **/
    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        log.info("#####LoginCheckInterceptor - preHandle()#####");
        // 현재 세션에 저장된 loginMsg 속성을 삭제
        HttpSession session = request.getSession();
        session.removeAttribute("loginMsg");

        // 세션에 isLogin란 이름의 속성이 없으면 로그인 상태가 아님
        if(request.getSession().getAttribute("isLogin") == null) {
            // 로그인 상태가 아니라면 로그인 폼으로 리다이렉트 시킨다.
            response.sendRedirect("loginForm");
            session.setAttribute("loginMsg", "로그인이 필요한 서비스");
            return false;
        }
        return true;
    }

    /* postHandle() 메서드는 클라이언트 요청이 들어오고 컨트롤러가 정상적으로
    * 실행된 이후에 공통적으로 적용할 추가 기능이 있을 때 주로 사용한다.
    * 만약 컨트롤러 실행 중에 예외가 발생하게 되면 postHandle() 메서드는
    * 호출되지 않는다.
    **/
    @Override
    public void postHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler,
        ModelAndView modelAndView) throws Exception {

```

```

log.info("#####LoginCheckInterceptor - postHandle()#####");
/* 수정 폼에서 수정 요청을 보내면서 비밀번호가 틀리면 자바스크립트로
 * history.back()을 사용하는데 POST 요청에서 Redirect 시키지 않을 경우
 * 브라우저에서 "양식 다시 제출 확인 - ERR_CACHE_MISS" 페이지가 뜰 수
 * 있다. 이런 경우 응답 데이터에 노캐쉬 설정을 하면 해결할 수 있다.
 */
response.setHeader("Cache-Control", "no-cache");
}

/* afterCompletion() 메서드는 클라이언트의 요청을 처리하고 뷰를 생성해
 * 클라이언트로 전송한 후에 호출된다. 클라이언트 실행 중에 예외가 발생하게 되면
 * 이 메서드 4번째 파라미터로 예외 정보를 받을 수 있다. 예외가 발생하지 않으면
 * 4번째 파라미터는 null을 받게 된다.
 */
@Override
public void afterCompletion(HttpServletRequest request,
    HttpServletResponse response, Object handler, Exception ex)
    throws Exception {
log.info("#####LoginCheckInterceptor - afterCompletion()#####");
}
}

```

## ▶ WebConfig 클래스에 인터셉터 등록과 URI 패턴 설정

앞에서 작성한 Spring MVC 환경설정 클래스인 WebConfig 클래스에 addInterceptors() 메서드를 오버라이드해 LoginCheckInterceptor 클래스를 다음과 같이 인터셉터로 등록하고 이 인터셉터를 적용한 경로 패턴을 설정하자.

### - com.springbootstudy.bbs.configurations.WebConfig 클래스에 메서드 오버라이드

```

/* LoginCheckInterceptor 클래스가 동작하기 위해서는 이 클래스를 Interceptor로
 * 등록해야 한다. 아래와 같이 addInterceptors() 메서드를 이용해 Interceptor로
 * 등록하면 스프링 빈으로 등록되고 addPathPatterns() 메서드에 지정한 URI로
 * 요청이 들어오고 나갈 때 LoginCheckInterceptor에 구현된 코드가 실행된다.
 */
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new LoginCheckInterceptor())
        // Interceptor를 적용할 경로 패턴을 설정한다.
        .addPathPatterns("/boardDetail", "/add*", "/write*",
            "/update*", "/memberUpdate*");
    // Interceptor를 적용하지 않을 경로 패턴을 설정한다.
    //.excludePathPatterns("/boardList");
}

```

## ▶ 로그인 폼 View

LoginCheckInterceptor에서 리다이렉트 되었을 때 “로그인이 필요한 서비스”라는 문구를 로그인 폼 위쪽에 출력하기 위해서 loginForm.html 파일에서 form 태그 바로 위쪽에 th:block 태그를 다음과 같이 추가하자.

- src/main/resources/templates/member/loginForm.html 수정

... 중 략 ...

```
<th:block th:unless="${#strings.isEmpty(session.loginMsg)}">
    <h4 class="fw-bold my-3 text-center text-primary" th:text="${session.loginMsg}">
        로그인에 필요한 서비스
    </h4>
</th:block>
<form class="my-5" id="loginForm" action="login" method="post">
    <h2 class="fw-bold">Member Login</h2>
```

... 중 략 ...

## 6. 추천/댓글 및 댓글 구현하기

이번 예제는 맘에 드는 게시글을 추천하거나 자료실 게시판이나 좋은 자료를 올려서 고맙다는 의사를 업로더에게 표시할 수 있는 댓글 기능을 추가해 볼 것이다.

대부분의 추천/댓글 기능은 게시글 상세보기에서 이루어지는데 사용자가 추천이나 댓글을 클릭했을 때 게시글 상세보기 페이지 전체를 새롭게 갱신하는 것 보다 추천이나 댓글이 표시되는 부분만 갱신하는 것이 네트워크 트래픽도 줄이고 페이지 이동 없이 서비스 할 수 있다는 장점이 있다. 그래서 이번 추천/댓글과 댓글 기능을 추가할 때 비동기통신 방식인 Ajax(Asynchronous Javascript And XML) 기술을 이용해 구현할 것이다.

Ajax는 비동기식 자바스크립트와 XML(Asynchronous Javascript And XML)의 약자로 HTML만으로 어려운 작업을 자바스크립트를 사용해 구현하고 사용자와 웹페이지가 상호 작용을 할 수 있도록 도와주는 기술이다. Ajax는 플래시나 ActiveX와 같이 별도의 프로그램을 설치할 필요가 없으며 화면에 보이는 전체 웹페이지를 다시 로딩 할 필요 없이 화면에서 필요한 부분만 다시 갱신하면 되므로 가볍고 속도 또한 빠르다.

먼저 추천/댓글 기능을 구현하고 뒤이어 댓글 기능을 추가로 구현하는 순서로 진행할 것이다.

### 1) 프로젝트 생성 및 환경설정

이번 예제는 앞에서 사용한 member 테이블은 수정 없이 그대로 사용할 것이며 댓글 기능이 추가됨에 따라서 springbbs 테이블을 수정하고 댓글을 저장할 reply 테이블을 새롭게 생성할 것이다.

#### 1-1) 프로젝트 생성

이번 예제도 springstudy-bbs05 프로젝트를 복사해 springclass-bbs06 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 게시글 상세보기 페이지에서 추천/댓글 기능과 댓글 기능을 Ajax로 구현할 것이다.

- 실습용 프로젝트 : springbootclass-bbs06
- 완성 프로젝트 : springbootstudy-bbs06

#### 1-2) 테이블 생성 및 데이터 입력

먼저 springbootstudy-bbs06 프로젝트의 src/main/resources/SQL/springbbs2.sql을 참고해 데이터베이스에서 springbbs 테이블을 수정하고 reply 테이블을 새로 생성하자.

#### 1-3) 의존 라이브러리 설정

이번 프로젝트에서 추천/댓글과 댓글 기능은 스프링이 지원하는 Ajax 요청 처리 방식을 사용할 것이다. Ajax 요청 처리는 요청을 처리한 결과 데이터인 자바 객체를 JSON으로 변환하여 응답하는데

이를 위해서 내부적으로 여러 객체가 사용되지만 spring-web을 의존설정 하면 필요한 라이브러리 설정은 완료된다. 우리는 이미 프로젝트를 생성할 때 spring-boot-starter-web을 의존설정을 했기 때문에 추가로 설정할 필요는 없다.



## 2) 게시글 상세보기 페이지 댓글 리스트 출력

추천/댓글 기능을 Ajax로 구현하기 위해서 앞에서 구현한 게시글 상세보기 화면을 수정할 것이다. 이 게시글 상세보기 화면에서 추천/댓글 기능과 댓글 기능을 제공할 것이므로 추천/댓글 버튼과 댓글 리스트를 출력하는 부분을 먼저 구현하고 추천/댓글 기능을 Ajax로 구현하게 될 것이다. 게시글 상세보기 화면을 구성할 때 게시글의 추천/댓글 수도 출력해야 하므로 기존에 사용하던 springbbs 테이블에 추천, 댓글 수를 저장할 수 있는 컬럼이 추가되었다. 그러므로 Board 클래스에 추천/댓글을 저장하는 인스턴스 변수를 추가해야 된다. 또한 게시글의 댓글을 저장할 테이블인 reply 테이블이 추가되었기 때문에 댓글 하나를 저장할 수 있는 DTO 클래스인 replay 클래스도 추가해야 된다. 아래를 참고해 Board 클래스를 수정하고 Reply 클래스를 추가하자.

### ▶ 게시글 DTO 클래스

게시글 하나의 정보를 관리하며 springbbs 테이블의 컬럼과 1:1로 맵핑되는 Board 클래스는 이번 프로젝트에서 게시글의 추천과 댓글 수를 저장하는 컬럼이 추가되었기 때문에 이에 맞게 추천과 댓글 수를 저장할 인스턴스 변수를 추가한다.

- `com.springbootstudy.bbs.domain.Board`

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

```
public class Board {  
    private int no;  
    private String title;  
    private String writer;  
    private String content;  
    private Timestamp regDate;  
    private int readCount;  
    private String pass;  
    private String file1;  
    private int recommend;  
    private int thank;  
}
```

### ▶ 댓글 DTO 클래스

`com.springbootstudy.bbs.domain` 패키지에 댓글 하나의 정보를 관리하며 reply 테이블의 컬럼과 1:1로 맵핑되는 Reply 클래스를 새로 생성하고 다음과 같이 작성하자.

- `com.springbootstudy.bbs.domain.Reply`

/\* 하나의 댓글 정보를 저장하는 댓글 DTO(Data Transfer Object) 클래스

\* 댓글 정보를 저장하고 있는 reply 테이블의 컬럼과 1:1 맵핑되는 클래스

\*/

@Getter

```

@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Reply {
    private int no;
    private int bbsNo;
    private String replyContent;
    private String replyWriter;
    private Timestamp regDate;
}

```

## ▶ Persistence 계층 구현

게시글 상세보기 기능은 앞에서 구현한 기능을 그대로 사용하면 되는데 추천과 땡큐 컬럼이 추가되었으니 매퍼 XML에서 추가된 컬럼이 조회될 수 있도록 2개의 컬럼을 추가하면 된다. 그리고 이번 에 추가되는 댓글 리스트를 DB 테이블에서 읽어오는 기능을 새로 추가하면 된다.

## ▣ 게시판 관련 SQL을 분리한 매퍼 XML 파일

BoardMapper.xml 파일에 no에 해당하는 게시글 정보를 읽어오는 맵핑 구문에서 추천과 땡큐 컬럼이 조회될 수 있도록 thank, recommend 두 개의 컬럼을 추가하고 게시글 번호에 해당하는 댓글 리스트를 읽어오는 맵핑 구문을 새롭게 추가하자.

### - src/main/resources/mappers/BoardMapper.xml

```

<!--
    게시글 번호에 해당하는 댓글 리스트를 가져오는 맵핑 구문
    테이블의 컬럼명은 일반적으로 언더스코어 표기법("_")을 사용하는 경우가
    많고 클래스의 인스턴스 멤버는 카멜표기법을 사용한다.
    테이블의 컬럼명과 도메인 클래스의 프로퍼티 이름이 다른 경우 아래와 같이
    SELECT 쿼리에 별칭을 사용해 도메인 클래스의 프로퍼티 이름과 동일하게
    맞춰서 조회할 수도 있다.
-->
<select id="replyList" resultType="Reply">
    SELECT
        no,
        bbs_no AS bbsNo,
        reply_content AS replyContent,
        reply_writer AS replyWriter,
        reg_date AS regDate
    FROM reply
    WHERE bbs_no = #{no}
    ORDER BY no DESC
</select>

```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 게시글 상세보기 페이지에 출력한 게시글 번호에 해당하는 댓글 리스트를 DB 테이블에서 읽어와 반환하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.mapper.BoardMapper

```
// 게시글 번호에 해당하는 댓글 리스트를 읽어와 반환하는 메서드
public List<Reply> replyList(int no);
```

## ▶ Business 계층 구현

BoardService 클래스에 게시글 번호에 해당하는 댓글 리스트를 반환하는 메서드를 아래와 같이 추가하자.

### - com.springbootstudy.bbs.service.BoardService

```
// 게시글 번호에 해당하는 댓글 리스트를 반환하는 메서드
public List<Reply> replyList(int no) {
    return boardMapper.replyList(no);
}
```

## ▶ Controller 클래스

BoardController 클래스의 게시글 상세보기 요청을 처리하는 getBoard() 메서드에서 댓글 리스트를 읽어와 모델에 저장하는 코드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.controller.BoardController에서 수정

... 중 략 ...

```
// no에 해당하는 게시글 정보와 pageNum, searchOption을 모델에 저장한다.
model.addAttribute("board", board);
model.addAttribute("pageNum", pageNum);
model.addAttribute("searchOption", searchOption);

// 현재 게시글 번호에 해당하는 댓글 리스트를 가져와 모델에 저장한다.
List<Reply> replyList = boardService.replyList(no);
model.addAttribute("replyList", replyList);
```

... 중 략 ...

## ▶ 메인 레이아웃 페이지에 부트스트랩 아이콘 링크 추가

게시글 상세보기에서 댓글 리스트를 출력할 때 부트스트랩이 제공하는 아이콘을 사용할 것이다. 아래 사이트로 접속하면 부트스트랩의 아이콘 설명 페이지로 연결되는데 이 페이지에서 맨 아래로 스

크롤해 내려가면 부트스트랩 아이콘 설치와 사용에 대한 설명을 볼 수 있다. 이 설명을 참고해서 main\_layout.html 파일에 부트스트랩 아이콘 사용을 위한 css 외부 링크를 추가하자.

<https://icons.getbootstrap.kr/>

- src/main/resources/templates/layouts/main\_layout.html

... 중 략 ...

```
<title>BoardService</title>
<link th:href="@{bootstrap/bootstrap.min.css}" rel="stylesheet" >
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/
bootstrap-icons.min.css">
```

... 중 략 ...

## ▶ 게시글 상세보기 View

게시글 상세보기는 댓글 관련 부분을 제외하고는 앞의 프로젝트에서 사용한 게시글 상세보기 소스와 동일하다. 아래 코드를 참고해 댓글 관련 자바스크립트 파일인 “reply.js” 파일을 참조하는 코드와 게시글 상세보기 아래쪽에 댓글 리스트를 출력하는 부분을 작성하자.

- src/main/resources/templates/views/boardDetail.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layouts/main_layout}">
<th:block layout:fragment="content">
  <!--/* th:block 안쪽에 추가해야 함 */-->
  <script src="js/reply.js"></script>
  <div class="row my-5" id="global-content">
```

... 중 략 ...

```
    <!--/*
      검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
    */-->
    <th:block th:if="${searchOption}">
      &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"

th:onclick="@{location.href='boardList?pageNum=${pageNum}&type=${type}&keyword=${key
word}'}"/>
    </th:block>
  </div>
```

```

</div>
<!--
아래부터 추천/댓글과 댓글 기능을 처리하는 부분으로
기존의 게시 글 상세보기 뷰 페이지의 아래쪽에 다음의 코드를 추가하면 된다.
-->
<!-- 추천/댓글 영역 -->
<div class="row my-5">
  <div class="col border p-3">
    <div id="recommend" class="text-end">
      <span id="commend" class="btnCommend text-primary" style="cursor:
pointer;">
        &nbsp;&nbsp;추천
        <span class="recommend" th:text="|({board.recommend})|"></span>
      </span> |
      <span id="thank" class="btnCommend text-primary" style="cursor: pointer;">
        &nbsp;&nbsp;댓글
        <span class="recommend" th:text="|({board.thank})|"></span>
      </span> |
      <span id="replyWrite" class="text-primary" style="cursor: pointer;">
        <i class="bi bi-file-earmark-text-fill" style="color: cornflowerblue;"></i> 댓글
        </span>
      </div>
    </div>
  </div>
</div>
<!-- 댓글 헤더 영역 -->
<div class="row" id="replyTitle">
  <div class="col p-2 text-center bg-dark text-white">
    <h3 class="fs-4">이 글에 대한 댓글 리스트</h3>
  </div>
</div>
<!-- 댓글 리스트 영역 -->
<!-- 댓글이 존재하는 경우 -->
<div th:if="{not #lists.isEmpty(replyList)}" class="row mb-3">
  <div class="col" id="replyList">
    <div th:each="reply : {replyList}" class="replyRow row border border-top-0">
      <div class="col">
        <div class="row bg-light p-2">
          <div class="col-4">
            <span th:text="{reply.replyWriter}">댓글 작성자</span>
          </div>
          <div class="col-8 text-end">
            <span class="me-3">
              [{ { #dates.format(reply.regDate, 'yyyy-MM-dd HH:mm:ss') }}]
            </span>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

```

                <button class="modifyReply btn btn-outline-success btn-sm"
th:data-no="{reply.no}">
                    <i class="bi bi-journal-text">수정</i>
                </button>
                <button class="deleteReply btn btn-outline-warning btn-sm"
th:data-no="{reply.no}">
                    <i class="bi bi-trash">삭제</i>
                </button>
                <button class="btn btn-outline-danger btn-sm"
th:onclick="reportReply('{reply.no}')">
                    <i class="bi bi-telephone-outbound">신고</i>
                </button>
            </div>
        </div>
        <div class="row">
            <div class="col p-3">
                <pre th:text="{reply.replyContent}">댓글 내용</pre>
            </div>
        </div>
    </div>
</div>
<!-- 댓글이 존재하지 않는 경우 -->
<div th:unless="{not #lists.isEmpty(replyList)}" class="row mb-3">
    <div class="col" id="replyList">
        <div class="replyRow row border border-top-0">
            <div class="col text-center p-5">
                이 게시 글에 대한 댓글이 존재하지 않습니다.
            </div>
        </div>
    </div>
</div>
<!-- 댓글 쓰기 폼 -->
<div class="row my-3 d-none" id="replyForm">
    <div class="col">
        <form name="replyWriteForm" id="replyWriteForm">
            <input type="hidden" name="bbsNo" th:value="{board.no}"/>
            <input type="hidden" name="replyWriter" th:value="{member.id}"/>
            <div class="row bg-light my-3 p-3 border">
                <div class="col">
                    <div class="row">
                        <div class="col text-center">
                            <span>악의적인 댓글은 예고 없이 삭제될 수 있으며 글쓰기 제한과 아이디
삭제 처리됩니다.</span>

```



### 3) 게시물 상세보기 페이지 추천/댓글 구현

앞에서 게시물 상세보기 페이지에 추천/댓글 기능을 적용할 수 있는 버튼과 댓글 리스트를 출력하였다. 앞에서 구성한 게시물 상세보기 페이지에서 추천 버튼과 댓글 버튼이 클릭되면 추천 수와 댓글 수를 증가하는 기능을 Ajax를 통해서 구현할 것이다.

추천/댓글 기능은 사용자가 추천 또는 댓글 버튼을 클릭하면 Ajax를 통해서 서버로 요청되면 이때 서버에서는 해당 요청을 처리한 후에 HTML 페이지로 응답하는 것이 아니라 결과 데이터를 JSON 형식으로 응답하고 클라이언트 단에서는 현재 페이지 새로고침 하지 않고 응답 받은 데이터를 화면에 적용하는 방식으로 구현해야 한다. 이렇게 구현하기 위해서는 어떤 요청을 받으면 해당 요청을 서버에서 처리하고 화면에 출력할 데이터를 JSON으로 응답해야 한다.

#### ▶ 추천/댓글 관련 자바스크립트 파일

추천/댓글과 댓글 관련 자바스크립트 코드를 작성할 reply.js 파일을 아래 경로에 새로 만들고 추천/댓글 버튼이 클릭되면 서버로 Ajax 요청을 보내는 코드를 다음과 같이 작성하자.

- src/main/resources/static/js/reply.js

// DOM(Document Object Model)이 준비되면

```
$(function() {
```

```
    // 추천/댓글 Ajax
```

```
    $(".btnCommend").click(function() {
```

```
        var com = $(this).attr("id");
```

```
        console.log("com : " + com);
```

```
        $.ajax({
```

```
            url: "recommend.ajax",
```

```
            // type을 지정하지 않으면 기본은 get 방식 요청이다.
```

```
            type: "post",
```

```
            // 파라미터로 보낼 데이터를 객체 리터럴로 작성
```

```
            data : { recommend: com, no : $("#no").val(),
```

```
            /* 응답 데이터를 json 형식으로 받기 위해서 dataType을 json으로
```

```
            * 지정했다. 서버에서 {"recommend": "10", "thank": "10"}와
```

```
            * 같이 json 문자열로 응답했기 때문에 아래에서 Ajax 통신이 성공하면
```

```
            * 실행될 콜백 함수의 첫 번째 인수로 지정한 data에는 자바스크립트
```

```
            * 객체가 넘어오기 때문에 닷(.) 연산자를 이용해 접근할 수 있다.
```

```
            **/
```

```
            dataType: "json",
```

```
            success: function(data) {
```

```
                /* 추천/댓글이 반영된 것을 사용자에게 알리고
```

```
                * 응답으로 받은 갱신된 추천하기 데이터를 화면에 표시한다.
```



```

    **/
    var msg = com == 'commend' ? "추천이" : "땡큐가";
    alert(msg + " 반영 되었습니다.");
    $("#commend > .recommend").text(" (" + data.recommend + ")");
    $("#thank > .recommend").text(" (" + data.thank + ")");
},
error: function(xhr, status, error) {
    alert("error : " + xhr.statusText + ", " + status + ", " + error);
}
});
});
});

```

## ▶ Persistence 계층 구현

추천 수와 땡큐 수를 DB 테이블에서 증가하는 기능과 새로 반영된 추천/땡큐 수를 읽어오는 기능을 새로 추가하면 된다.

### ▣ 게시판 관련 SQL을 분리한 매퍼 XML 파일

BoardMapper.xml 파일에 게시글 번호에 해당하는 추천/땡큐 수를 DB 테이블에서 업데이트 하는 맵핑 구문과 새롭게 반영된 추천/땡큐 정보를 DB 테이블에서 가져오는 맵핑 구문을 다음과 같이 추가하자.

#### - src/main/resources/mappers/BoardMapper.xml

```
<!--
```

추천/땡큐 정보를 업데이트하는 맵핑 구문

SET 절에 <if> 태그를 사용해 추천 또는 땡큐 요청인지를 판단해 추천일 때는 추천 정보를 업데이트하고 땡큐 일 때는 땡큐 정보를 업데이트 할 수 있도록 동적 SQL로 맵핑 구문을 작성했다.

SQL 파라미터로 사용할 데이터가 여러 개라서 BoardMapper 인터페이스에서 @Param("파라미터 이름") 애노테이션을 사용해 Map 객체로 전달되도록 하였다. 이렇게 전달된 파라미터를 조건절에서 사용할 때는 #{}를 사용하지 않고 파라미터 이름만 지정해야 하며 문자열을 사용할 때는 쌍 따옴표("")나 홑 따옴표('')로 감싸줘야 한다.

Map에 저장한 데이터가 int와 같은 자바 기본 형과 String 데이터이므로 parameterType을 생략할 수 있다.

```
-->
```

```
<update id="updateRecommend">
```

```
UPDATE springbbs
```

```
SET
```

```
<if test="recommend == 'commend'">
```

```

        recommend=recommend + 1
    </if>
    <if test="recommend == 'thank'">
        thank = thank + 1
    </if>
    WHERE no = #{no}
</update>

```

```

<!--

```

추천/땡큐 정보를 가져오는 맵핑 구문

추천/땡큐 두 가지 정보를 반환할 경우에는 객체 타입으로 반환하는 것이 일반적이지만 새로운 클래스를 만들지 않고 Board 클래스에 이 두 정보를 저장할 수 있는 프로퍼티가 있어서 Board 객체로 반환되도록 했다.

```

-->

```

```

<select id="getRecommend" resultType="Board">
    SELECT recommend, thank
    FROM springbbs
    WHERE no = #{no}
</select>

```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 게시글 번호에 해당하는 추천/땡큐 수를 DB 테이블에서 업데이트 하는 메서드와 새롭게 반영된 추천/땡큐 정보를 DB 테이블에서 가져오는 메서드를 아래와 같이 추가 하자.

### - com.springbootstudy.bbs.mapper.BoardMapper

```

// 게시글 번호에 해당하는 추천/땡큐를 업데이트 하는 메서드
public void updateRecommend(
    @Param("no") int no, @Param("recommend") String recommend);

// 게시글 번호에 해당하는 추천/땡큐 정보를 읽어와 반환하는 메서드
public Board getRecommend(int no);

```

## ▶ Business 계층 구현

게시글 번호에 해당하는 추천/땡큐 수를 업데이트 하고 새롭게 반영된 추천/땡큐 정보를 가져와 반환하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.service.BoardService

```

// 추천/땡큐 정보를 업데이트하고 갱신된 추천/땡큐를 가져오는 메서드
public Map<String, Integer> recommend(int no, String recommend) {

    boardMapper.updateRecommend(no, recommend);

```

```

Board board = boardMapper.getRecommend(no);

Map<String, Integer> map = new HashMap<String, Integer>();
map.put("recommend", board.getRecommend());
map.put("thank", board.getThank());
return map;
}

```

## ▶ Ajax 요청을 처리하는 Controller 클래스

com.springbootstudy.bbs.ajax 패키지에 BoardAjaxController 클래스를 새롭게 생성하고 추천/댓글에 대한 Ajax 요청을 처리하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.ajax.BoardAjaxController

```

/* RestController 클래스임을 정의
 * @RestController 애노테이션은 @Controller에 @ResponseBody가
 * 추가된 것과 동일하다. RestController의 주용도는 JSON으로 응답하는 것이다.
 */
@RestController
public class BoardAjaxController {

    // 의존객체 주입 설정
    @Autowired
    private BoardService boardService;

    /* 추천/댓글에 대한 Ajax 요청을 처리하는 메서드
     *
     * @RestController 애노테이션이 클래스에 적용되었기 때문에
     * 이 메서드에서 반환하는 값은 JSON으로 직렬화되어 응답 본문에 포함된다.
     */
    @PostMapping("/recommend.ajax")
    public Map<String, Integer> recommend(@RequestParam("no") int no,
        @RequestParam("recommend") String recommend) {

        /* RestController 클래스에서 @RequestMapping, @GetMapping,
         * @PostMapping 등과 같은 요청 매핑이 적용된 메서드의 반환 타입이
         * String인 경우 HttpMessageConverter를 이용해 String 객체를 직렬화
         * 하여 반환하고 반환 타입이 위와 같이 Map이거나 자바 객체인 경우
         * MappingJackson2HttpMessageConverter를 사용해 JSON으로 변환한다.
         *
         * Service 클래스에서 맵에 저장할 때 아래와 같이 저장하였다.
         *
         * map.put("recommend", board.getRecommend());
         * map.put("thank", board.getThank());
         */
    }
}

```

```
* 이 데이터는 다음과 같이 JSON 형식으로 변환되어 응답된다.  
*  
* { "recommend": 15, "thank": 26 }  
**/  
return boardService.recommend(no, recommend);  
}  
}
```

## 4) 게시물 상세보기 페이지 댓글 기능 구현

앞에서 게시물 상세보기 페이지에서 댓글 리스트를 출력하는 기능을 추가하였다. 이번에는 게시물 상세보기 페이지에서 댓글을 쓰고 수정하고 삭제하는 기능을 구현해 보자.

### 4-1) 댓글 쓰기

#### ▶ 댓글 관련 자바스크립트 파일

reply.js 파일에 “댓글쓰기” 버튼이 클릭되었을 때와 댓글 쓰기 폼이 Submit 될 때 처리하는 자바스크립트 이벤트 처리 코드를 DOM이 준비되면 실행될 콜백 함수 안에 다음과 같이 작성하자.

#### - src/main/resources/static/js/reply.js

```
// 댓글 쓰기가 클릭되었을 때 이벤트 처리
$("#replyWrite").on("click", function() {

    // 화면에 보이는 상태인지 체크
    console.log($("#replyForm").css("display"));
    console.log($("#replyForm").is(":visible"));

    // 댓글 쓰기 폼이 화면에 보이는 상태라면
    if($("#replyForm").is(":visible")) {

        /* 댓글 쓰기 폼이 현재 보이는 상태이고 댓글 쓰기 위치가 아닌
        * 댓글 수정에 있으면 댓글 쓰기 폼을 슬라이드 업 하고 댓글 쓰기
        * 위치로 이동시켜 0.3초 후에 슬라이드 다운을 한다.
        */
        var $prev = $("#replyTitle").prev();
        if(! $prev.is("#replyForm")) {
            $("#replyForm").slideUp(300);
        }
        setTimeout(function(){
            $("#replyForm").insertBefore("#replyTitle").slideDown(300);
        }, 300);
    } else { // 댓글 쓰기 폼이 보이지 않는 상태라면
        $("#replyForm").removeClass("d-none")
            .css("display", "none").insertBefore("#replyTitle").slideDown(300);
    }

    /* 댓글 쓰기 폼과 댓글 수정 폼을 같이 사용하기 때문에 아래와 같이 id를
    * 동적으로 댓글 쓰기 폼으로 변경하고 댓글 수정 버튼이 클릭될 때 추가한
    * data-no라는 속성을 삭제 한다.
    */
}
```

```

$("#replyForm").find("form")
    .attr("id", "replyWriteForm").removeAttr("data-no");
$("#replyContent").val("");
$("#replyWriteButton").val("댓글쓰기");

});

/* 댓글 쓰기 폼이 submit 될 때
 * 최초 한 번은 완성된 html 문서가 화면에 출력되지만 댓글 쓰기를 한 번
 * 이상하게 되면 ajax 통신을 통해 받은 데이터를 이전 화면과 동일하게
 * 출력하기 위해 그때 그때 동적으로 요소를 생성하기 때문에 delegate 방식의
 * 이벤트 처리가 필요하다. 댓글 쓰기, 수정 또는 삭제하기를 한 후에
 * 결과 데이터를 화면에 출력하면서 자바스크립트를 이용해 html 요소를
 * 동적으로 생성하기 때문에 이벤트 처리가 제대로 동작하지 않을 수 있다.
 * 이럴 때는 아래와 같이 delegate 방식의 이벤트 처리가 필요하다.
 */
$(document).on("submit", "#replyWriteForm", function(e) {

    if($("#replyContent").val().length < 5) {
        alert("댓글은 5자 이상 입력해야 합니다.");
        return false;
    }

    var params = $(this).serialize();
    console.log(params);

    $.ajax({
        "url": "replyWrite.ajax",
        "data": params,
        "type": "post",
        "dataType": "json",
        "success": function(resData) {
            console.log(resData);

            // 반복문을 통해서 - html 형식으로 작성
            $("#replyList").empty();
            $.each(resData, function(i, v) {

                // v.regData == 1672300816000
                var date = new Date(v.regDate);
                var strDate = date.getFullYear() + "-" + ((date.getMonth() + 1 < 10)
                    ? "0" + (date.getMonth() + 1) : (date.getMonth() + 1)) + "-"
                    + (date.getDate() < 10 ? "0" + date.getDate() : date.getDate()) + " "
                    + (date.getHours() < 10 ? "0" + date.getHours() : date.getHours()) + ":"
                    + (date.getMinutes() < 10 ? "0" + date.getMinutes() : date.getMinutes()) + ":"

```

```

    + (date.getSeconds() < 10 ? "0" + date.getSeconds() : date.getSeconds());

    var result =
      '<div class="row border border-top-0 replyRow">'
      + '<div class="col">'
      + ' <div class="row bg-light p-2">'
      + '   <div class="col-4">'
      + '     <span>' + v.replyWriter + '</span>'
      + '   </div>'
      + '   <div class="col-8 text-end">'
      + '     <span class="me-3">' + strDate + "</span>"
      + '     <button class="modifyReply btn btn-outline-success btn-sm"
data-no="' + v.no + '">'
      + '       <i class="bi bi-journal-text">수정</i>'
      + '     </button>'
      + '     <button class="deleteReply btn btn-outline-warning btn-sm"
data-no="' + v.no + '">'
      + '       <i class="bi bi-trash">삭제</i>'
      + '     </button>'
      + '     <button class="btn btn-outline-danger btn-sm"
onclick="reportReply(\'' + v.no + '\')">'
      + '       <i class="bi bi-telephone-outbound">신고</i>'
      + '     </button>'
      + '   </div>'
      + ' </div>'
      + ' <div class="row">'
      + '   <div class="col p-3">'
      + '     <pre>' + v.replyContent + '</pre>'
      + '   </div>'
      + ' </div>'
      + '</div>'
      + '</div>'

    $("#replyList").append(result);
    $("#replyList").removeClass("text-center");
    $("#replyList").removeClass("p-5");

  }); // end $.each()

  // 댓글 쓰기가 완료되면 폼을 숨긴다.
  $("#replyForm").slideUp(300)
    .add("#replyContent").val("");
},
"error": function(xhr, status) {
  console.log("error : " + status);
}

```

```

    }
});

// 폼이 submit 되는 것을 취소한다.
return false;
});

```

## ▶ Persistence 계층 구현

Persistence 계층은 댓글 정보를 파라미터로 받아서 DB 테이블에 추가하는 기능을 추가하면 된다.

### ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

BoardMapper.xml 파일에 댓글을 reply 테이블에 추가하는 매퍼 구문을 다음과 같이 작성하자.

#### - src/main/resources/mappers/BoardMapper.xml

```

<!--
    댓글을 추가하는 매퍼 구문
-->
<insert id="addReply" parameterType="Reply">
    INSERT INTO reply(bbs_no, reply_content, reply_writer, reg_date)
    VALUES(#{bbsNo}, #{replyContent}, #{replyWriter}, SYSDATE())
</insert>

```

### ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에서 reply 테이블에 댓글 정보를 추가하는 메서드를 다음과 같이 작성하자.

#### - com.springbootstudy.bbs.mapper.BoardMapper

```

// 게시글 번호에 해당하는 댓글을 DB에 등록하는 메서드
public void addReply(Reply reply);

```

## ▶ Business 계층 구현

BoardService 클래스에 BoardMapper를 사용해 댓글을 DB에 저장하는 메서드를 다음과 같이 추가하자.

#### - com.springbootstudy.bbs.service.BoardService

```

// 현재 게시글에 해당하는 댓글을 등록하는 메서드
public void addReply(Reply reply) {
    boardMapper.addReply(reply);
}

```



## ▶ Ajax 요청을 처리하는 Controller 클래스

BoardAjaxController 클래스에 댓글 쓰기 Ajax 요청을 처리하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.ajax.BoardAjaxController

```
// 댓글 쓰기 Ajax 요청을 처리하는 메서드
@PostMapping("/replyWrite.ajax")
public List<Reply> addReply(Reply reply) {

    // 새로운 댓글을 등록한다.
    boardService.addReply(reply);

    /* 댓글 쓰기가 완료되면 새롭게 추가된 댓글을 포함해서 게시글
     * 상세보기에 다시 출력해야 하므로 갱신된 댓글 리스트를 가져와 반환한다.
     *
     * 아래는 게시글 번호에 해당하는 댓글이 List<Reply>로 반환되기
     * 때문에 스프링은 MappingJackson2HttpMessageConverter를
     * 사용해 다음과 같이 객체 배열 형태의 JSON 형식으로 변환되어 응답된다.
     *
     * [
     *   {bbsNo: 100, no: 27, regDate: 1516541138000,
     *     replyContent: "저도 동감이에요..", replyWriter: "midas"},
     *   {bbsNo: 100, no: 20, ... },
     *   ...
     *   {bbsNo: 100, no: 1, regDate: 1462682672000,
     *     replyContent: "항상 감사합니다...", replyWriter: "midas"}
     * ]
     */
    return boardService.replyList(reply.getBbsNo());
}
```

## 4-2) 댓글 수정하기

### ▶ 댓글 관련 자바스크립트 파일

댓글 리스트에서 “수정” 버튼이 클릭되었을 때와 댓글 수정 폼이 Submit 될 때 처리하는 자바스크립트 이벤트 처리 코드를 DOM이 준비되면 실행될 콜백 함수 안에 다음과 같이 작성하자.

### - src/main/resources/static/js/reply.js

```
/* 댓글 수정 버튼이 클릭되면
 * 댓글을 수정한 후에 동적으로 요소를 생성하기 때문에 delegate 방식으로 이벤트를
 * 등록해야 한다. 만약 $(".modifyReply").click(function() {}); 형식으로 이벤트를
 * 등록했다면 새로운 댓글을 등록하거나, 수정 또는 삭제한 후에는 클릭 이벤트가
```

```

* 제대로 동작되지 않을 수 있기 때문에 delegate 방식의 이벤트 처리가 필요하다.
**/
$(document).on("click", ".modifyReply", function() {

    // 화면에 보이는 상태인지 체크
    console.log($("#replyForm").css("display"));
    console.log($("#replyForm").is(":visible"));

    // 수정 버튼이 클릭된 최상의 부모를 구한다.
    console.log($(this).parents(".replyRow"));
    var $replyRow = $(this).parents(".replyRow");

    // 댓글 쓰기 폼이 화면에 보이는 상태라면
    if($("#replyForm").is(":visible")) {

        /* 댓글 쓰기 폼이 현재 보이는 상태이고 현재 클릭된 수정 버튼의
        * 부모 요소의 다음 요소가 아니라면 댓글 쓰기 폼을 슬라이드 업 하고
        * 댓글 수정 위치로 이동시켜 0.3초 후에 슬라이드 다운을 한다.
        */
        var $next = $replyRow.next();
        if(! $next.is("#replyForm")) {
            $("#replyForm").slideUp(300);
        }
        setTimeout(function(){
            $("#replyForm").insertAfter($replyRow).slideDown(300);
        }, 300);

    } else { // 댓글 쓰기 폼이 화면에 보이지 않는 상태라면
        $("#replyForm").removeClass("d-none")
            .css("display", "none").insertAfter($replyRow).slideDown(300);
    }

    /* 댓글 쓰기 폼과 댓글 수정 폼을 같이 사용하기 때문에 아래와 같이 동적으로
    * id를 replyUpdateForm으로 변경하고 댓글 수정 버튼에서 data-no라는
    * 속성의 값을 읽어와 댓글 수정 폼의 data-no라는 속성을 추가한다.
    */
    $("#replyForm").find("form")
        .attr({id: "replyUpdateForm", "data-no": $(this).attr("data-no") });
    $("#replyWriteButton").val("댓글수정");

    // 현재 클릭된 수정 버튼이 있는 댓글을 읽어와 수정 폼의 댓글 입력란에 출력한다.
    var reply = $(this).parent().parent().next().find("pre").text();
    $("#replyContent").val($.trim(reply));

});

```

```

// 댓글 수정 폼이 submit 될 때
$(document).on("submit", "#replyUpdateForm", function() {
//$("#replyUpdateForm").on("submit", function() {

    if($("#replyContent").val().length <= 5) {
        alert("댓글은 5자 이상 입력해야 합니다.");
        // Ajax 요청을 취소한다.
        return false;
    }

    /* 아래에서 $("#replyTable").empty(); 가 호출되면 댓글 쓰기
    * 폼도 같이 문서에서 삭제되기 때문에 폼을 원래 위치로 이동시킨다.
    */
    $("#global-content > div").append($("#replyForm").slideUp(300));

    /* replyNo는 최초 폼이 출력될 때 설정되지 않았다.
    * 댓글 쓰기 폼과 댓글 수정 폼을 하나로 처리하기 때문에 댓글 번호는
    * 동적으로 구하여 요청 파라미터에 추가해 줘야 한다. 댓글 수정시
    * 댓글 번호를 서버로 전송해야 댓글 번호에 해당하는 댓글을 수정할 수 있다.
    */
    var params = $(this).serialize() + "&no=" + $(this).attr("data-no");
    console.log(params);

    $.ajax({
        url: "replyUpdate.ajax",
        type: "patch",
        data: params,
        dataType: "json",
        success: function(resData, status, xhr) {

            console.log(resData);

            // 반복문을 통해서 - html 형식으로 작성
            $("#replyList").empty();
            $.each(resData, function(i, v) {

                // v.regData == 1672300816000
                var date = new Date(v.regDate);
                var strDate = date.getFullYear() + "-" + ((date.getMonth() + 1 < 10)
                    ? "0" + (date.getMonth() + 1) : (date.getMonth() + 1)) + "-"
                    + (date.getDate() < 10 ? "0" + date.getDate() : date.getDate()) + " "
                    + (date.getHours() < 10 ? "0" + date.getHours() : date.getHours()) + ":"
                    + (date.getMinutes() < 10 ? "0" + date.getMinutes() : date.getMinutes()) + ":"

```

```

        + (date.getSeconds() < 10 ? "0" + date.getSeconds() : date.getSeconds());

    var result =
        '<div class="row border border-top-0 replyRow">'
        + '<div class="col">'
        + ' <div class="row bg-light p-2">'
        + ' <div class="col-4">'
        + ' <span>' + v.replyWriter + '</span>'
        + ' </div>'
        + ' <div class="col-8 text-end">'
        + ' <span class="me-3">' + strDate + "</span>"
        + ' <button class="modifyReply btn btn-outline-success btn-sm"
data-no="' + v.no + '">'
        + ' <i class="bi bi-journal-text">수정</i>'
        + ' </button>'
        + ' <button class="deleteReply btn btn-outline-warning btn-sm"
data-no="' + v.no + '">'
        + ' <i class="bi bi-trash">삭제</i>'
        + ' </button>'
        + ' <button class="btn btn-outline-danger btn-sm"
onclick="reportReply(\'' + v.no + '\')">'
        + ' <i class="bi bi-telephone-outbound">신고</i>'
        + ' </button>'
        + ' </div>'
        + ' </div>'
        + ' <div class="row">'
        + ' <div class="col p-3">'
        + ' <pre>' + v.replyContent + '</pre>'
        + ' </div>'
        + ' </div>'
        + ' </div>'
        + '</div>'

    $("#replyList").append(result);

}); // end $.each()

// 댓글 수정하기가 완료되면 폼에 작성된 댓글 내용을 지운다.
$("#replyContent").val("");

},
error: function(xhr, status, error) {
    alert("ajax 실패 : " + status + " - " + xhr.status);
}

```

```
});

// 폼이 submit 되는 것을 취소한다.
return false;
});
```

## ▶ Persistence 계층 구현

Persistence 계층은 수정된 댓글 정보를 파라미터로 받아서 DB 테이블에서 수정하는 기능을 추가하면 된다.

### ■ 게시판 관련 SQL을 분리한 매퍼 XML 파일

BoardMapper.xml 파일에 댓글을 reply 테이블에서 수정하는 매퍼 구문을 다음과 같이 작성하자.

#### - src/main/resources/mappers/BoardMapper.xml

```
<!--
    댓글을 수정하는 매퍼 구문
-->
<update id="updateReply" parameterType="Reply">
    UPDATE reply
        SET reply_content = #{replyContent},
            reg_date = SYSDATE()
        WHERE no = #{no}
</update>
```

### ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에서 reply 테이블에서 댓글 정보를 수정하는 메서드를 다음과 같이 작성하자.

#### - com.springbootstudy.bbs.mapper.BoardMapper

```
// DB에서 댓글 번호에 해당하는 댓글을 수정하는 메서드
public void updateReply(Reply reply);
```

## ▶ Business 계층 구현

BoardService 클래스에 BoardMapper를 사용해 댓글을 DB에서 수정하는 메서드를 다음과 같이 추가하자.

#### - com.springbootstudy.bbs.service.BoardService

```
// 댓글 번호에 해당하는 댓글을 수정하는 메서드
public void updateReply(Reply reply) {
    boardMapper.updateReply(reply);
}
```

```
}
```

## ▶ Ajax 요청을 처리하는 Controller 클래스

BoardAjaxController 클래스에 댓글 수정 Ajax 요청을 처리하는 메서드를 다음과 같이 추가하자.

### - com.springbootstudy.bbs.ajax.BoardAjaxController

```
// 댓글 수정 Ajax 요청을 처리하는 메서드
@PatchMapping("/replyUpdate.ajax")
public List<Reply> updateReply(Reply reply) {

    // 수정된 댓글 정보를 받아서 댓글 번호에 해당하는 댓글을 수정한다.
    boardService.updateReply(reply);

    // 새롭게 갱신된 댓글 리스트를 가져와 반환한다.
    return boardService.replyList(reply.getBbsNo());
}
```

## 4-3) 댓글 삭제하기

### ▶ 댓글 관련 자바스크립트 파일

댓글 리스트에서 “삭제” 버튼이 클릭되었을 때 처리하는 자바스크립트 이벤트 처리 코드를 DOM이 준비되면 실행될 콜백 함수 안에 다음과 같이 작성하자.

### - src/main/resources/static/js/reply.js

```
/* 댓글 삭제 버튼이 클릭되면
 * 댓글을 삭제한 후에 동적으로 요소를 생성하기 때문에 delegate 방식으로 이벤트를
 * 처리를 해야 한다. 만약 $(".deleteReply").click(function() {}); 와 같이 이벤트를
 * 등록했다면 댓글을 삭제한 후에는 클릭 이벤트가 제대로 동작되지 않을 수 있기 때문에
 * 아래 코드와 같이 delegate 방식의 이벤트 처리가 필요하다.
 */
$(document).on("click", ".deleteReply", function() {

    /* 아래에서 $("#replyTable").empty(); 가 호출되면 댓글 쓰기
     * 폼도 같이 문서에서 삭제되기 때문에 폼을 원래 위치로 이동시킨다.
     */
    $("#global-content > div").append($("#replyForm").slideUp(300));
    $("#replyContent").val("");

    var no = $(this).attr("data-no");
    var writer = $(this).parent().prev().find("span").text();
    var bbsNo = $("#replyForm input[name=bbsNo]").val();
```

```

var params = "no=" + no + "&bbsNo=" + bbsNo;
console.log(params);

var result = confirm(writer + "님이 작성한 " + no + "번 댓글을 삭제하시겠습니까?");

if(result) {
    $.ajax({
        url: "replyDelete.ajax",
        type: "delete",
        data: params,
        dataType: "json",
        success: function(resData, status, xhr) {
            console.log(resData);
            $("#replyList").empty();

            // 반복문을 통해서 - html 형식으로 작성
            $.each(resData, function(i, v) {

                // v.regData == 1672300816000
                var date = new Date(v.regDate);
                var strDate = date.getFullYear() + "-" + ((date.getMonth() + 1 < 10)
                    ? "0" + (date.getMonth() + 1) : (date.getMonth() + 1)) + "-"
                    + (date.getDate() < 10 ? "0" + date.getDate() : date.getDate()) + " "
                    + (date.getHours() < 10 ? "0" + date.getHours() : date.getHours()) + ":"
                    + (date.getMinutes() < 10 ? "0" + date.getMinutes() : date.getMinutes()) +
                    ":"
                    + (date.getSeconds() < 10 ? "0" + date.getSeconds() : date.getSeconds());

                var result =
                    '<div class="row border border-top-0 replyRow">'
                    + '<div class="col">'
                    + ' <div class="row bg-light p-2">'
                    + ' <div class="col-4">'
                    + ' <span>' + v.replyWriter + '</span>'
                    + ' </div>'
                    + ' <div class="col-8 text-end">'
                    + ' <span class="me-3">' + strDate + '</span>'
                    + ' <button class="modifyReply btn btn-outline-success btn-sm"
data-no="' + v.no + '">'
                    + ' <i class="bi bi-journal-text">수정</i>'
                    + ' </button>'
                    + ' <button class="deleteReply btn btn-outline-warning btn-sm"
data-no="' + v.no + '">'
                    + ' <i class="bi bi-trash">삭제</i>'
                    + ' </button>'

```

```

        + '      <button      class="btn      btn-outline-danger      btn-sm"
onclick="reportReply(\' + v.no + '\')">'
        + '      <i class="bi bi-telephone-outbound">신고</i>'
        + '      </button>'
        + '    </div>'
        + '  </div>'
        + ' <div class="row">'
        + '   <div class="col p-3">'
        + '     <pre>' + v.replyContent + '</pre>'
        + '   </div>'
        + ' </div>'
        + '</div>'
+ '</div>'

$("#replyList").append(result);

}); // end $.each()
},
error: function(xhr, status, error) {
    alert("ajax 실패 : " + status + " - " + xhr.status);
}
});
}
// 앵커 태그에 의해 페이지가 이동되는 것을 취소한다.
return false;
});

/* 아래는 신고하기 버튼을 임시로 연결한 함수
 * DOM(Document Object Model)이 준비되면 호출되는 콜백 함수 밖에 작성
 */
function reportReply(elemId) {
    var result = confirm("이 댓글을 신고하시겠습니까?");
    if(result == true) {
        alert("report - " + result);
    }
}
}

```

## ▶ Persistence 계층 구현

Persistence 계층은 댓글 번호를 파라미터로 받아서 DB 테이블에서 댓글을 삭제하는 기능을 추가하면 된다.

## ▣ 게시판 관련 SQL을 분리한 매퍼 XML 파일

BoardMapper.xml 파일에 댓글을 reply 테이블에서 삭제하는 맵핑 구문을 다음과 같이 작성하자.



- src/main/resources/mappers/BoardMapper.xml

```
<!--
    댓글을 삭제하는 맵핑 구문
-->
<delete id="deleteReply">
    DELETE FROM reply
    WHERE no = #{no}
</delete>
```

## ■ 게시판 관련 DB작업을 수행하는 매퍼 인터페이스

BoardMapper 인터페이스에 reply 테이블에서 댓글 정보를 삭제하는 메서드를 다음과 같이 작성하자.

- com.springbootstudy.bbs.mapper.MemberMapper

```
// DB에서 댓글 번호에 해당하는 댓글을 삭제하는 메서드
public void deleteReply(int no);
```

## ▶ Business 계층 구현

BoardService 클래스에 BoardMapper를 사용해 DB에서 댓글을 삭제하는 메서드를 다음과 같이 추가하자.

- com.springbootstudy.bbs.service.BoardService

```
// 댓글 번호에 해당하는 댓글을 삭제하는 메서드
public void deleteReply(int no) {
    boardDao.deleteReply(no);
}
```

## ▶ Ajax 요청을 처리하는 Controller 클래스

BoardAjaxController 클래스에 댓글 삭제 Ajax 요청을 처리하는 메서드를 다음과 같이 추가하자.

- com.springbootstudy.bbs.ajax.BoardAjaxController

```
// 댓글 삭제 Ajax 요청을 처리하는 메서드
@DeleteMapping("/replyDelete.ajax")
public List<Reply> deleteReply(@RequestParam("no") int no,
    @RequestParam("bbsNo") int bbsNo) {

    // 댓글 번호에 해당하는 댓글을 삭제한다.
    boardService.deleteReply(no);
```

```
// 새롭게 갱신된 댓글 리스트를 가져와 반환한다.  
return boardService.replyList(bbsNo);  
}
```