Spring 게시판

1. Spring MVC 게시판 CRUD 구현

우리는 앞에서 SpringLegacyProject로 SpringMVC 프로젝트를 생성하고 Spring MVC가 지원하는 기능을 이용해 웹 애플리케이션을 구현하는 방법에 대해 알아보았다.
이번에는 SpringLegacyProject로 SpringMVC 프로젝트를 만들고 CRUD 기능을 제공하는 게시판을 구현해 보면서 Spring MVC가 지원하는 다양한 애노테이션을 사용해 컨트롤러를 구현하는 방법에 대해 알아볼 것이다. 이외에도 DB를 보다 쉽고 간편하게 연동할 수 있도록 지원하는 퍼시스턴스(Persistence) 프레임워크인 MyBatis를 사용해 DB와 연동하는 게시판을 구현할 것이다. 퍼시스턴스 프레임워크란 기존의 전통적인 JDBC 프로그래밍 방식을 대체하기 위한 프레임워크로 DB와 연동되는 시스템을 보다 간편하고 빠르게 개발하고 안정적으로 구동할 수 있도록 도와주는 프레임워크를 말한다. 퍼시스턴스 프레임워크에는 SQL Mapper와 ORM(Object Relational Mapping) 이라는 두 가지 종류가 있다.
SQL Mapper는 SQL 쿼리와 자바 객체(Object)를 매핑해주는 프레임워크이며 대표적인 SQL Mapper 프레임워크에는 MyBatis와 Spring 프로젝트에서 제공하는 JDBC Template이 있다.
ORM은 DB 데이터와 객체 지향 프로그래밍 언어 간의 호환되지 않는 데이터를 변환하는 프로그래밍 기법을 말하며 자바 진영의 ORM은 JPA(Java Persistence API, 자바 ORM 기술에 대한 API 표준 명세로 ORM을 사용하기 위한 인터페이스의 묶음)를 기준으로 구현되어 있으며 DB 데이터를 자바 객체로 맵핑하여 객체 간의 관계를 바탕으로 SQL을 자동으로 생성해 주는 프레임워크를 말한다. 대표적인 ORM 프레임워크에는 Hibernate, EclipseLink 등이 있다.
우리가 사용할 MyBatis는 시중의 여러 도서에서 ORM 프레임워크로 부르고 있는 경우도 있지만 MyBatis는 ORM 프레임워크라기 보다는 JDBC를 기반으로 하는 SQL Mapping 프레임워크 이다. MyBatis는 Hibernate와 같이 DB 데이터와 자바 객체 간의 관계를 바탕으로 SQL을 자동으로 생성해 주는 것이 아니기 때문에 개발자가 직접 SQL을 작성해줘야 SQL과 자바 객체를 매핑할 수 있다.
MyBatis는 예전에 iBatis라는 이름으로 아파치 프로젝트에 편입되어 발전해 오다가 2010년에 MyBatis3.x가 공개되면서 MyBatis라는 이름으로 변경되고 아파치 프로젝트에서 분리되어 Google Code의 프로젝트로 편입 되었다.
MyBatis의 가장 큰 특징은 SQL 쿼리를 프로그램 소스에서 분리해 별도의 Mapper XML 파일이나 Mapper 인터페이스를 작성해 관리한다는 점이다.
국내에서 기업용 애플리케이션을 구축할 때 스프링프레임워크와 MyBatis를 연동한 웹 애플리케이션 구현 기법이 많이 사용된다. 또한 전자정부프레임워크도 스프링프레임워크와 MyBatis 기반으로 설계되어 있다.

1) 프로젝트 생성 및 환경설정

1-1) 프로젝트 생성

- **실습용 프로젝트 : springclass-bbs01**
- **완성 프로젝트 : springstudy-bbs01**

1-2) 의존 라이브러리와 BuildPath 설정

Spring Legacy Project 메뉴를 이용해 Spring MVC Project를 생성하게 되면 자바 버전은 1.6, 스프링프레임워크 버전 3.1.1, Dynamic Web Module 2.5가 기본 설정되어 있다.

우리는 스프링프레임워크 버전 4.2.4 버전을 사용할 것이므로 Maven을 통해 버전을 변경해야 한다. 그리고 자바 1.8 버전과 Dynamic Web Module 3.1을 사용할 것이므로 Configure Build Path를 통해 Java Build Path의 자바 버전과 Java Compiler 버전을 1.8로 설정하고 Server Runtime 설정을 변경해야 한다. 그리고 프로젝트를 생성할 때 패키지의 3번째 단계에 지정한 bbs가 ContextRoot와 Artifact Id로 사용되기 때문에 이를 변경해야 할 필요도 있을 것이다. 이에 대한 자세한 내용은 springstudy-bbs01의 web.xml의 주석을 참고하길 바란다.

1-3) DB TABLE 생성 및 데이터 입력

먼저 springstudy-bbs01 프로젝트의 src/main/resources/SQL/springbbs.sql 파일을 이용해 MySQL에 테이블을 생성하고 데이터를 추가하자.

- 데이터베이스 이름 : spring

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

-- 게시글 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(10) NOT NULL,
  content VARCHAR(1000) NOT NULL,
  reg_date TIMESTAMP NOT NULL,
  read_count INTEGER(5) NOT NULL,
  pass VARCHAR(10) NOT NULL,
  file1 VARCHAR(100)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

1-4) 웹 애플리케이션 배포 서술자(Deployment Descriptor)

- src/main/webapp/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
  <!--
    * Spring MVC 게시판 01 - springstudy-bbs01
```

- Spring MVC와 MyBatis를 활용한 웹 애플리케이션 구현
- Annotation 기법을 이용한 Controller 맵핑하기,
- commons-dbcp2를 이용해 DB 접속 정보를 properties로 저장하고
  DBCP2 BasicDataSource를 Bean으로 설정하여 DBCP 적용
- MyBatis를 활용한 게시판 CRUD - 글쓰기, 리스트 보기, 상세보기, 수정, 삭제
-->
<!--
SpringSTS에서 SpringLegacyProject로 SpringMVCProject 만들기

이 예제는 SpringSTS에서 Spring Legacy Project로 Spring MVC Project를
생성하고 설정하는 방법과 Spring MVC와 MyBatis를 활용해 웹 애플리케이션을
구현하는 방법을 설명하기 위한 예제이다.

1. 먼저 File -> New 메뉴에서 Spring Legacy Project 선택하거나 Package Explorer의
   빈 공간에 마우스 오른쪽을 클릭해 나타나는 컨텍스트 메뉴의 New 메뉴에서 Spring
   Legacy Project를 선택하여 나타나는 New Spring Legacy Project 대화상자에서
   Spring MVC Project를 선택하고 프로젝트 이름을 springstudy-bbs01로 지정한 후
   "Next" 버튼을 클릭한다. 이어서 나타나는 대화상자에서 top-level package를
   com.springstudy.bbs로 입력하고 "Finish" 버튼을 클릭해 프로젝트를 생성한다.

2. 스프링프레임워크 버전 및 의존 라이브러리 버전, Artifact Id 등을 변경한다.
   Spring MVC Project를 생성하면 스프링프레임워크 버전 3.1.1이 기본 설정되어 있다.
   우리는 스프링프레임워크 버전 4.2.4를 사용할 것이므로 Maven을 통해 버전을 수정해야
   한다. 또한 테스트, 로깅. AOP, MyBatis와 Spring MVC 연동을 위해 MyBatis에서
   제공하는 라이브러리 의존성도 추가해야 한다.

   pom.xml을 더블클릭해서 선택하고 Overview 탭에서 Properties 부분을 아래와 같이
   변경해 준다. 만약 스프링프레임워크 버전을 변경한다면 org.springframework-version
   부분을 더블 클릭해 Value의 값을 "4.2.4.RELEASE"로 지정한 후 OK 버튼을 클릭하면 된다.
   나머지 의존 라이브러리도 아래를 참고해 property를 설정하면 된다.

```
<properties>
    <java-version>1.8</java-version>
    <org.springframework-version>4.2.4.RELEASE</org.springframework-version>
    <log4j-version>1.2.17</log4j-version>
    <jstl-version>1.2</jstl-version>
    <junit-version>4.7</junit-version>
    <mysql-version>8.0.31</mysql-version>
    <mybatis-version>3.4.5</mybatis-version>
    <mybatis-spring-version>1.3.1</mybatis-spring-version>
</properties>
```

   그리고 이 Overview 탭에서 Artifact Id를 springstudy-bbs01로 수정하자.

3. 메이븐을 통해 스프링 MVC 관련 라이브러리 의존성을 설정한다.
pom.xml의 Dependencies 탭을 선택하고 "Add" 버튼을 클릭해 아래 모듈을
의존 설정하면 메이븐이 의존 관계에 있는 모듈을 자동으로 등록해서 Spring MVC와
MyBatis를 연동하기 위한 기본적인 라이브러리 의존 설정을 할 수 있다.

- spring-context 모듈
- spring-webmvc 모듈
- spring-jdbc 모듈
- mybatis 모듈
- mybatis-spring 모듈
- commons-dbcp2 모듈
- mysql-connector-java 모듈
- 기타 필요한 라이브러리(jstl, log4j, slf4j, junit, aspectjrt 등등)

참고로 mybatis 모듈은 MyBatis3 프레임워크를 애플리케이션에서 사용하기 위해 지원하는 클래스 라이브러리 이고 mybatis-spring 모듈은 Spring MVC와 MyBatis 연동을 지원하기 위한 클래스 라이브러리 이다.
우리는 MyBatis-3.4.5 버전과 mybatis-spring-1.3.1 버전을 사용할 것이다.

*** MyBatis3 프레임워크와 스프링프레임워크를 연동하는 방법과 매퍼 작성 방법에 대한 자세한 내용은 아래의 파일을 참고하기 바란다.**

- Spring Bean 설정
src/main/webapp/WEB-INF/spring/root-context.xml

- MyBatis 코드로 변환하기 위한 Mapper 설정파일
src/main/resources/repository.mappers/*Mapper.xml 매퍼설정 파일

4. Java Build Path와 Compiler 등을 변경한다.
Spring MVC Project를 생성하면 자바 버전 1.6, Dynamic Web Module 2.5가
기본 설정되어 있다. 우리는 자바 1.8 이상과 Dynamic Web Module 3.1을
사용할 것이므로 Configure Build Path를 통해 Java Build Path의 자바 버전과
Java Compiler 버전을 1.8로 설정하고 Server Runtime 설정을 변경해야 한다.
그리고 프로젝트를 생성할 때 top-level package의 3번째 단계에 지정한 bbs가
자동으로 ContextRoot와 Artifact Id로 설정되기 때문에 bbs가 아닌 프로젝트
이름을 ContextRoot와 Artifact Id로 사용하려면 추가적인 설정이 필요하다.

* Configure Build Path에서 자바 버전과 Dynamic Web Module 변경하기
새로 생성한 프로젝트 springstudy-bbs01에 마우스 오른쪽 버튼을 클릭해
나타나는 컨텍스트 메뉴에서 "Build Path" -> "Configure Build Path"를
선택하여 나타나는 Properties for springstudy-bbs01 대화상자에서
다음과 같이 설정한다.

- Java Build Path 설정
Java Build Path 선택 -> JRE System Library 선택 -> Edit 버튼을 클릭

Edit Libray 대화상자에서 Workspace default JRE를 선택하고 Finish 클릭
한 후 Apply 버튼을 클릭한다.

- Java Compiler 설정
Java Compiler 선택 -> Enalbe project specific settings 선택 해제
우측의 Confiqure Workspace Settings 선택하여 나타나는 대화상자에서
Java Build 설정에 지정한 버전을 선택하고 Apply 버튼을 클릭하면
다시 이전 대화상자로 돌아오는데 여기서도 Apply 버튼 클릭

- Dynamic Web Module 설정
Project Facets 선택 -> 우측의 Dynamic Web Module 3.1 선택 -> Java도
Java Build Path 설정에서 지정한 버전과 동일한 버전을 선택한다.
그리고 우측의 Runtimes 탭을 선택하여 Apache Tomcat v8.5를 선택하고
Apply 버튼을 클릭한다.

- Dynamic Web Module을 3.1으로 변경 했다면 web.xml의 <web-app> 태그에서
version과 XML 스키마 정의 파일(xsd)의 버전도 아래와 같이 3.1로 변경하자.

version="3.1"
http://java.sun.com/xml/ns/javaee/web-app_3_1.xsd

- Context root 변경
Web Project Settings 선택 -> 우측의 Context root에 springstudy-bbs01를
입력하고 Apply 버튼을 클릭한다.

5. web.xml에 스프링이 제공하는 프런트 컨트롤러인 DispatcherServlet을 서블릿으로 등록하고 요청 처리를 위한 서블릿 매핑을 설정한다. 기본적으로 Spring MVC Project를 생성하면 Spring MVC에 필요한 bean 설정 파일이 서블릿 초기화 파라미터로 설정되어 있다. 또한 Spring MVC 외에 필요한 bean 설정은 별도의 파일로 분리되어 설정할 수 있도록 웹 애플리케이션 초기화 파라미터와 리스너가 설정되어 있다. 이외에도 추가로 스프링이 제공하는 Character Encoding Filter 등을 설정할 수 있다.

6. Spring Web MVC 설정 파일을 작성한다.
Spring MVC Project를 생성하게 되면 기본적으로 두 개의 스프링 Bean 설정 파일이 생성된다. 하나는 DispatcherServlet이 읽어서 DI 컨테이너를 생성하고 SpringWebMVC에 필요한 Bean을 설정하는데 사용하는 SpringWebMVC 설정용 XML 파일이고 또 다른 하나는 ContextLoaderListener가 읽어서 DI 컨테이너를 생성하고 추가적으로 필요한 Bean을 설정하는데 사용하는 Bean 설정용 XML 파일이다.

대규모 애플리케이션 개발에서는 스프링 설정을 각각의 성격에 맞게 여러 개를 작성하여 관리하는 경우가 대부분이다. 보통은 SpringWebMVC 관련 설정과 이외의 설정을 각각의 용도에 맞게 분할하여 작성한다. 분할한 설정 파일을 스프링이 인식할 수 있도록 지정하는 방법은 아래의 Listener와 웹 애플리케이션 초기화 파라미터, 서블릿 초기화 파라미터를 참고하기 바란다.

- SpringWebMVC 관련 설정(annotation 적용, viewResolver 등등)
- 추가적인 Bean 설정(DBCP, Mybatis, 기타 애플리케이션에서 필요한 Bean 등등)

7. 실제 요청을 처리할 Controller 클래스를 구현한다.
   이때 Controller 클래스와 연동해서 사용되는 Service 계층 클래스, DataAccess 계층 클래스는 Interface 규칙을 적용해 구현하고 View 페이지를 같이 구현한다.

DataAccess 계층인 DAO에서 MyBatis가 지원하는 SqlSessionTemplate을 사용해 DB 작업을 하게 되는데 DAO를 작성하기에 앞서 MyBatis 설정 파일을 작성하거나 SQL을 분리한 Mapper를 작성해야 하는데 자세한 설명은 /WEB-INF/spring/root-context.xml 파일의 주석에서 마이바티스와 스프링을 연동하는 방법을 참고하기 바란다.

-->

<!-- Listener와 웹 애플리케이션 초기화 파라미터 설정 -->
<!--
ContextLoaderListener는 ServletContextListener를 구현하고
ContextLoader 클래스를 상속한 클래스로 특정 이벤트에 의해서 실행된다.
ServletContextListener 구현체는 ServletContext 인스턴스가 생성될 때
즉 톰캣 서버가 웹 애플리케이션을 로드 할 때 이벤트가 발생하여 실행된다.

ContextLoaderListener는 SpringWebMVC 설정 외에 추가적으로 필요한
Bean 설정 정보를 DispatcherServlet이 실행되기 전에 애플리케이션 초기화
파라미터에서 읽어 Spring DI 컨테이너를 생성하고 필요한 Bean을 생성하는
기능을 제공한다. DispatcherServlet이 실행되면서 SpringWebMVC 설정을
읽어서 Spring DI 컨테이너를 생성하고 SpringWebMVC에 필요한 Bean을
생성할 때 필요한 객체를 미리 생성해야 할 필요가 있거나 스프링 설정 파일을
분리하여 관리하기 위해서 사용하며 ContextLoaderListener도 서블릿과
마찬가지로 web.xml에 등록해야 해당 이벤트가 발생할 때 실행될 수 있다.
Listener는 특정 이벤트에 의해서 실행되기 때문에 리스너 매핑이 필요 없다.

ContextLoaderListener가 실행될 때 읽어야 할 스프링 Bean 설정 파일을
웹 애플리케이션 초기화 파라미터를 통해 지정할 수 있다.
만약 ContextLoaderListener가 읽어야 할 스프링 Bean 설정 파일이 하나가
아니라 여러 개인 경우 아래와 같이 콤마(","), 공백(" "), 탭("\t"), 줄 바꿈("\n"),
세미콜론(";") 등을 사용해 각각의 설정 파일을 구분하여 지정할 수 있으며 이때
웹 애플리케이션 초기화 파라미터 이름은 반드시 contextConfigLocation으로
지정해야 한다.

<context-param>
    <param-name>contextConfigLocation</param-name>
 <param-value>
    /WEB-INF/applicationConfig-context.xml
    classpath:resources/resources-context.xml
    classpath:resources/dataSouce.xml

```
    </param-value>
  </context-param>

  ContextLoaderListener는 contextConfigLocation이라는 웹 애플리케이션
  초기화 파라미터가 없을 경우 WEB-INF/applicationContext.xml 파일을 찾는다.

  아래는 SpringToolSuite에서  SpringMVCProject를 만들면 기본적으로
  생성해 주는 애플리케이션 초기화 파라미터 설정과 ContextLoaderListener를
  등록하는 설정을 그대로 사용한 것이다.

  이렇게 설정하면 ContextLoaderListener는 톰캣 서버가 웹 애플리케이션을 로드 할
  때 실행되어 웹 애플리케이션 초기화 파라미터의 contextConfigLocation에 지정한
  /WEB-INF/spring/root-context.xml 파일을 읽어서 스프링 DI 컨테이너인
  WebApplicationContext를 생성 한다.
-->
  <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>
  <listener>
      <listener-class>
          org.springframework.web.context.ContextLoaderListener
      </listener-class>
  </listener>

<!--
  DispatcherServlet은 스프링 MVC의 핵심으로 스프링 MVC의 프론트 컨트롤러
  역할을 담당한다. 다른 서블릿과 마찬가지로 DispatcherServlet도 서블릿으로
  동작하기 위해서는 웹 애플리케이션의 배포서술자인 web.xml에 등록해야 한다.
  스프링 MVC에서는 클라이언트로부터 들어오는 모든 요청을 DispatcherServlet이
  받아 각 요청에 대응하는 각각의 처리는 개발자가 구현한 컨트롤러를 통해 처리한다.

  DispatcherServlet이 처음 실행될 때 SpringWebMVC에 필요한 여러 가지
  Bean을 생성할 수 있도록 SpringWebMVC 설정 파일을 자신의 서블릿 초기화
  파라미터에서 읽어와 Spring DI 컨테이너를 생성하는 기능도 제공한다.
  DispatcherServlet이 실행될 때 읽어야 할 SpringWebMVC 설정 파일을
  서블릿 초기화 파라미터를 통해 지정할 수 있다.
  만약 DispatcherServlet이 읽어야할 SpringWebMVC 설정 파일이 하나가
  아니라 여러 개인 경우 아래와 같이 콤마(","), 공백(" "), 탭("\t"), 줄 바꿈("\n"),
  세미콜론(";") 등을 사용해 각각의 설정 파일을 구분하여 지정할 수 있으며 이때
  서블릿 초기화 파라미터 이름은 반드시 contextConfigLocation으로 지정해야
  한다.

  <servlet>
      <servlet-name>appServlet</servlet-name>
```

```
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                /WEB-INF/mainServlet/spring-context.xml
                classpath:resources/spring-context02.xml
                file:d:\spring\spring-context03.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    DispatcherServlet은 contextConfigLocation이라는 서블릿 초기화
    파라미터가 없을 경우 자신의 서블릿 이름인 appServlet에 -servlet.xml을
    추가한 파일을 WEB-INF 디렉터리에서 찾는다. 다시 말해 별도의 설정이 없는
    경우 DispatcherServlet은 WEB-INF/appServlet-servlet.xml 파일을
    읽어서 Spring DI 컨테이너를 생성하게 된다.

    아래는 SpringProject에서  SpringMVCProject로 프로젝트를 생성하면
    Spring STS가 기본적으로 생성해 주는 DispatcherServlet의 서블릿 설정이다.

    아래는 SpringToolSuite에서  SpringMVCProject를 만들면 기본적으로
    생성해 주는 DispatcherServlet의 서블릿 설정을 그대로 사용한 것이다.

    이렇게 설정하면 DispatcherServlet은 자신이 톰캣 서버에 의해 실행 될 때
    자신의 서블릿 초기화 파라미터의 contextConfigLocation에 지정한
    /WEB-INF/spring/appServlet/servlet-context.xml 파일을 읽어서
    스프링 DI 컨테이너인 WebApplicationContext를 생성 한다.
-->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring/appServlet/servlet-context.xml
        </param-value>
    </init-param>
<!--
    서블릿 클래는 최초 요청이 들어올 때 톰갯 서버에 의해 서블릿 클래스가 로딩되고
    인스턴스화 된다. 그리고 서블릿 초기화 작업이 이루어지고 서블릿 컨테이너에 담겨
    서블릿으로 동작하게 된다. 이렇게 서블릿 클래스는 최초 요청이 들어올 때
```

```
    클래스 로딩 -> 인스턴스화 -> 서블릿 초기화  작업을 거치므로 맨 처음 실행될 때
    보통의 실행보다 많은 시간이 걸리게 되는데 이런 문제를 해결하기 위해
    <load-on-startup>을 설정하여 톰캣이 실행되면서 서블릿을 초기화 하도록
    설정할 수 있다. <load-on-startup> 에 지정하는 값이 0 보다 크면 톰갯이
    실행되면서  서블릿을 초기화 하게 되는데 이 값이 0보다 크고 가장 작은 정수 값을
    가진 서블릿이 우선 순위가 제일 높다. 다시 말해 <load-on-startup>에 지정된
    값이 1, 2, 3의 값을 가진 서블릿이 있다면 가장 작은 1의 값을 지정한 서블릿이
    제일 먼저 초기화 된다. 같은 값을 가진 서블릿이 존재 한다면 먼저 정의된 서블릿이
    먼저 초기화 된다.
  -->
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!--
    아래는 ContextRoot 들어오는 .jsp 요청을 제외한 모든 요청을
    DispatcherServlet가 받을 수 있도록 url-pattern을 지정한 것이다.
  -->
  <servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <!-- 스프링 MVC가 제공하는 인코딩용 필터 정의 -->
  <!--
    CharacterEncodingFilter 클래스를 사용하면 폼 입력으로
    넘어오는 요청 파라미터 데이터를 지정한 문자 셋으로 처리해 준다.

    get 방식의 요청은 톰캣 서버의 servlet.xml에 지정한 문자 셋으로 처리되고
    post 방식의 요청은 별도로 문자 셋 처리 코드를 작성하지 않아도 된다.
  -->
   <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>
        org.springframework.web.filter.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
    <!--
        기존에 문자 셋이 설정되어 있다 하더라도 request, response에
        강제로 위에서 지정한 문자 셋으로 인코딩을 설정하라고 지정하는 셋팅
        즉 getCharacterEncoding()을 호출해 null이 아니라 하더라도
        request와 response에 utf-8 문자 셋을 강제로 설정한다.
    -->
```

```
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
  </filter>
  <!--
    Servlet과 마찬가지로 Filter도 <filter-mappign> 태그를 사용해
    필터를 매핑하며 <url-pattern> 태그에 지정한 패턴에 따라서 실행된다.
  -->
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

 <!--
    아래의 리소스가 src/main/webapp/META-INF 폴더의 context.xml 파일에
    정의되어 있다면 아래는 반드시 정의되어야 하는 것은 아니다. 다만 웹 어플리테이션을
    위해 JNDI를 사용하는 리소스 참조를 web.xml에 정의하는 것을 권장하고 있다.
    http://kenu.github.io/tomcat70/docs/jndi-resources-howto.html 참고
  -->
  <resource-ref>
    <description>DBCP Connection 정의</description>
    <res-ref-name>jdbc/springDBPool</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

1-5) Spring MVC Bean 정의

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <!--
```

```
    Annotation 기법을 이용한 SpringMVC 설정 파일

    이 설정 파일은 Spring MVC 관련 설정 파일로 MVC 위주의 설정이 많기 때문에
    xmlns를 지정할 때 mvc에는 접두어를 사용하지 않았다. 그래서 mvc 네임스페이스는
    접두어를 사용하지 않은 태그를 사용해야 한다. 나머지 네임스페이스는 접두어를
    사용해 xmlns지정했기 때문에 접두어를 붙여 태그를 사용해야 한다.

    <annotation-driven /> 태그는 Spring MVC에서 필요한 Annotation 기반
    모든 기능을 사용할 수 있는 설정으로 이 태그는 JSR-303 검증 지원, 메시지 변환,
    필드 포매팅 지원 등을 포함한 강력하고 다양한 기능을 제공한다.

    <context:component-scan>에 의해 컨트롤러가 스프링 DI 컨테이너에
    등록되고 @Controller Annotation이 지정된 클래스의 객체가 컨트롤러라는
    것을 판단하고 Spring MVC의 다양한 Annotation을 사용하기 위한 설정으로
    Spring MVC의 Annotation을 사용할 때는 필수라고 생각하면 되겠다.

    <annotation-driven /> 태그는 아래 두 클래스를 스프링 빈으로 등록하여
    @Controller 애노테이션이 지정된 클래스를 컨트롤러로 사용할 수 있도록 해준다.

    * org.springframework.servlet.mvc.method.annotation 패키지의
        - RequestMappingHandlerMapping
        - RequestMappingHandlerAdapter

    또한 이 태그는 JSON이나 XML에 대한 요청과 응답 처리를 위한 모듈이나 파라미터
    데이터 바이딩 처리를 위한 ConversionService 등을 스프링 빈으로 등록 해준다.
-->
<annotation-driven />

<!--
    DispatcherServlet을 경유해 정적 리소스에 접근하기 위한 설정으로
    정적 리소스란 Html, 이미지, CSS, JavaScript 파일 등을 의미한다.
    아래 설정은 ContextRoot/resources/ 로 들어오는 정적 리소스 요청에
    대해 ContextRoot/resources/ 디렉터리를 매핑한 것이다. 이 설정은
    DispatcherServlet의 url-pattern을 "/"로 지정하여 DispatcherServlet이
    정적 컨텐츠를 포함한 모든 요청을 처리할 수 있도록 설정해야 제대로 동작한다.
-->
<resources mapping="/resources/**" location="/resources/" />

<!--
    <context:component-scan /> 태그는 base-package에 지정한 패키지를
    기준으로 스프링이 컴포넌트를 스캔하여 자동으로 bean을 생성해 주는 설정으로
    <context:annotation-config /> 태그가 수행하는 모든 것을 수행하며 여기에
    더해 스프링이 자동으로 bean을 발견하여 선언하고 생성해 주는 설정이다.

    다시 말해 Spring bean 설정 파일에 <bean> 태그를 사용해 클래스를 등록하지
```

```
    않아도 스프링프레임워크가 아래 Annotation이 적용된 클래스를 찾아 annotation
    기반 와이어링(스프링 DI 컨테이너에 등록하고 클래스 간의 관계를 맺어 주는)을
    해주는 설정이다.

    컨트롤러 클래스(@Controller), DAO 클래스(@Repository),
    Service(@Service) 클래스가  스프링에 의해 스프링 DI 컨테이너에 등록된다.
-->
<context:component-scan base-package="com.springstudy.bbs" />

<!--
    ## 1. HandlerMapping 정의하기 ##
    HandlerMapping은 클라이언트가 보낸 웹 요청 URL과 그 요청을 처리할
    컨트롤러를 매핑해 주는 클래스로 최초로 요청을 받은 DispatcherServlet이
    그 요청을 어느 컨트롤러로 보내야 할지 HandlerMapping에게 의뢰해 요청을
    처리할 컨트롤러를 선택하게 된다. HandlerMapping은 요청 URL을 참고해
    요청을 처리할 컨트롤러를 결정하여 DispatcherServlet에게 전달하게 된다.
    HandlerMapping을 지정하지 않으면 BeanNameUrlHandlerMappging과
    DefaultAnnotationHandlerMapping이 기본 적용된다.

    하지만 스프링 설정 파일에 <annotation-driven /> 태그를 설정하게 되면
    @Controller 애노테이션이 지정된 클래스를 컨트롤러로 사용할 수 있도록 해주는
    RequestMappingHandlerMapping과 RequestMappingHandlerAdapter를
    스프링 빈으로 등록하게 되므로 이 예제는 이 두 클래스를 우선 사용되게 된다.
-->
<!--
    ## 2. Controller 정의하기 ##
    <annotation-driven/>과 <context:component-scan />을 적용했기
    때문에 Spring MVC Annotation 기반의 Bean을 선언하고 검색하게 되므로
    우리가 구현하는 Controller 클래스에 @Controller 애노테이션을 적용하면
    Spring DI 컨테이너가 Bean을 스캔하여 Controller로 등록해 준다.
-->
<!--
    ## 3. ViewResolver 정의하기 ##
    컨트롤러로 부터 요청에 대한 처리 결과를 전달 받은 DispatcherServlet은
    뷰 리졸버에게 논리적인 뷰 이름을 실제 뷰 구현체(JSP등)에 매핑하도록 요청하고
    뷰 리졸버는 요청 결과를 표시할 실제 뷰 객체를 DispatcherServlet에게
    전달한다. DispatcherServlet은 뷰 리졸버로 부터 전달 받은 뷰 객체에
    컨트롤러로 부터 받은 모델 정보를 렌더링 하고 그 결과를 클라이언트에 응답한다.

    뷰 리졸버는 컨트롤러로 부터 전달 받은 논리적인 뷰 이름을 가지고 실제 응답
    결과를 생성할 뷰 객체를 구할 때 사용되며 실제 데이터가 출력될 뷰를 선택하는
    역할을 한다. 만약 컨트롤러로 부터 반환된 논리적인 뷰의 이름이 main이라면
    뷰 리졸버는 prefix와 suffix를 조합해 다음과 같은 물리적인 뷰의 이름을 만든다.
    뷰가 결정이 되면 DispatcherServlet에서 이 뷰로 포워딩 된다.
```

```
    /WEB-INF/index.jsp?body=views/main.jsp

    뷰 리졸버 구현체는 몇 가지가 있으며 뷰 리졸버를 별도로 지정하지 않으면
    InternalResourceViewResolver가 기본 적용된다.
    뷰 리졸버의 viewClass 속성에 뷰 리졸버가 생성할 뷰 객체를 지정할 수 있는데
    이를 생략하면 InternalResourceView가 기본 적용된다.
    InternalResourceView는 JSP, HTML과 같이 웹 애플리케이션의 내부 자원을
    사용해 응답 결과를 만드는 객체로 속성에 모델을 지정하고 RequestDispatcher를
    이용해 지정된 뷰로 포워딩 된다.

    아래에서 viewClass 속성에 지정한 JstlView는 JSTL을 사용한 자원용 뷰 이다.
    InternalResourceView의 자식 클래스로 스프링 메시지 자원 파일을 JSTL의
    포맷 태그에서 사용할 수 있도록 해 준다. 뷰에서 JSTL을 사용한다면 별도로
    viewClass를 지정하지 않아도 ViewResolver는 JstlView를 반환한다.
-->
<beans:bean id="viewResolver" class=
    "org.springframework.web.servlet.view.InternalResourceViewResolver"
    p:viewClass="org.springframework.web.servlet.view.JstlView"
    p:prefix="/WEB-INF/index.jsp?body=views/"
    p:suffix=".jsp"/>

<!--
    클라이언트의 요청이 특별한 처리 없이 단순히 뷰만 보여줘야 할 경우에
    아래와 같이 뷰 전용 컨트롤러를 설정하여 뷰 페이지를 지정할 수 있다.

    <view-controller /> 태그는 뷰 전용 컨트롤러를 설정하는 태그로
    어떤 요청이 특별한 비즈니스 로직 처리 없이 단순히 뷰만 보여줘야 할 필요가
    있을 때 유용하게 사용할 수 있는 설정이다. 아래는 /writeForm, /boardWrite로
    들어오는 요청에 대한 뷰를 writeForm으로 지정한 예로 이 설정 파일에 bean으로
    등록한 ViewResolver의 prefix와 suffix가 적용된 뷰를 보여주는 설정이다.
    다시 말해 view-name 속성에 지정하는 뷰의 이름은 ViewResolver에서
    prefix, suffix에 지정한 정보를 제외한 나머지를 지정하면 된다.

    <view-controller /> 태그의 path 속성에 지정한 경로와 동일한
    요청 URI가 컨트롤러의 @RequestMapping에 POST 방식만 지정되어
    있다면 HTTP Status 405 - Request method 'GET' not supported
    예외가 발생한다.

    여러 개의 요청 URI에 같은 뷰를 적용하려면 아래와 같이 여러 번 지정하면 된다.
-->
<view-controller path="/writeForm" view-name="writeForm" />
<view-controller path="/boardWrite" view-name="writeForm" />

<!--
    Redirect View 설정
```

```
        특정 URI에 대한 클라이언트의 요청을 다른 페이지로 Redirect 시켜야할 경우에
        아래와 같이 리다이렉트 전용 컨트롤러를 설정할 수 있다.

        아래는 ContextRoot("/"), ContextRoot/index, ContextRoot/default로
        들어오는 요청을 게시 글 리스트로 리다이렉트 시키는 예 이다.
    -->
    <view-controller path="/" view-name="redirect:/boardList" />
    <redirect-view-controller path="/index" redirect-url="/boardList"/>
    <redirect-view-controller path="/default" redirect-url="/boardList"/>

    <!--
        아래는 ContextRoot/write로 들어오는 요청을 위에서 view-controller에
        설정한 /writeForm 으로 리다이렉트 하는 예 이다.
    -->
    <redirect-view-controller path="/write" redirect-url="/writeForm"/>
</beans:beans>
```

1-6) Spring Bean 정의

- src/main/webapp/WEB-INF/spring/root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:c="http://www.springframework.org/schema/c"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.2.xsd">
    <!--
        SpringWebMVC 설정 외에 추가적으로 필요한 DBCP 설정, MyBatis 설정
        메시지 자원 설정, 추가적인 스프링 Bean을 등록하는 설정파일

        DBCP 정보를 properties 파일로부터 읽어서 DataSource를 구하는 설정
        파일을 import 태그를 사용해 현재 Bean 설정 파일에 포함시킬 수 있다.
    -->
    <import resource="classpath:datasource/dbcpdatasource.xml" />

    <!--
        JNDI 방식의 DBCP를 참조해 DataSource를 구하는 설정 파일을
        import 태그를 사용해 현재 Bean 설정 파일에 포함시킬 수 있다.

        JNDI를 이용한 DBCP DataSource 설정과 관련된 자세한 내용은
```

```
    dacpdatasource_jndi.xml 파일의 주석을 참고하기 바란다.
-->
<!--
    <import resource="classpath:datasource/dbcpdatasource_jndi.xml" />
-->

<!--
    마이바티스와 스프링을 연동하는 방법
    1. MyBatis3 프레임워크를 사용하기 위한 라이브러리 의존 설정
        mybatis-3.4.5.jar

    2. MyBatis3 프레임워크와 스프링프레임워크 연동을 위한 라이브러리 의존 설정
        mybatis-spring-1.3.1.jar

    3. MyBatis 설정 파일 작성(생략 가능)
      스프링과 MyBatis를 연동하는 애플리케이션을 구현할 때는 MyBatis 설정 파일은
      필수사항이 아니며 우리도 별도 설정이 필요하지 않기 때문에 이 설정 파일을 사용하지
      않지만 MyBatis 설정 파일 작성에 대한 자세한 사항은 아래 파일을 참고하기 바란다.

        src/main/resources/mybatis-config_참고.xml

    4. 스프링 Bean 설정 파일인 root-context.xml 파일에
       SqlSessionFactory 생성을 위한 SqlSessionFactoryBean을 Spring Bean으로 정의

    5. 스프링 Bean 설정 파일인 root-context.xml 파일에
       DAO에서 의존하는 SqlSessionTemplate을 Spring Bean으로 정의

    6. 스프링 Bean 설정 파일인 root-context.xml 파일에
        스프링이 지원하는 TransactionManager를 Spring Bean으로 정의
        트랜잭션을 처리하지 않는 경우에는 TransactionManager는 생략 가능하다.

    7. 마이바티스를 이용한 DAO 구현
      - SqlSession 구현체인 SqlSessionTemplate을 이용한 구현
      - 매퍼(Mapper Interface) 동적생성을 이용한 구현

    8. SQL 문장을 분리한 Mapper 작성(Mapper XML 또는 Mapper Interface)

-->

<!--
    1. MyBatis3 프레임워크를 사용하기 위한 라이브러리 의존 설정

    MyBatis3 프레임워크를 사용하기 위해서는 mybatis 모듈을 아래와 같이
    pom.xml에 라이브러리 의존 설정을 해야 한다.
```

```
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.5</version>
    </dependency>
-->

<!--
    2. MyBatis3 프레임워크와 스프링프레임워크 연동을 위한 라이브러리 의존 설정

    스프링과 마이바티스를 연동하기 위해서는 마이바티스에서 지원하는 스프링
    연동 모듈을 아래와 같이 pom.xml에 라이브러리 의존 설정을 해야 한다.

    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.3.1</version>
    </dependency>

    마이바티스와 스프링프레임워크를 연동하기 위해서는 DataSource와
    스프링이 지원하는 TransactionManager 등이 스프링 빈으로 정의되어
    있어야 한다.
-->

<!--
    3. MyBatis 설정 파일 작성(mybatis-config.xml, 생략가능)

    MyBatis는 SQL Mapping 프레임워크로 별도의 설정 파일을 가질 수 있다.
    스프링프레임워크와 MyBatis를 연동할 경우 MyBatis 설정이 필수적으로 필요한
    것은 아니지만 MyBatis에서 추가적으로 필요한 부분을 지정할 수 있다.
    이 MyBatis 설정 파일에는 DB의 접속 주소(별도 설정 가능)나 매핑(Mapper)
    파일의 경로, 도메인 객체의 별칭, TransactionManager, DBCP 등의 정보를
    설정할 수 있다. 일반적으로 "src/main/resources" 폴더에 mybatis-config.xml
    파일로 작성하면 된다.

    참고로 우리는 mybatis-config.xml 파일은 사용하지 않지만 MyBatis 설정 파일에
    대한 자세한 사항은 아래 파일의 주석을 참고하기 바란다.

    src/main/resources/mybatis-config_참고.xml
-->

<!--
    4. 스프링 Bean 설정 파일인 root-context.xml 파일에
       SqlSessionFactory 생성을 위한 SqlSessionFactoryBean을 Spring Bean으로 정의
```

```
        SqlSessionFactory 객체는 MyBatis와 스프링프레임워크 연동에서 핵심적인
        객체로 MyBatis의 전반적인 정보를 가지고 있는 객체이다.
        이 객체는 DB Connection을 생성하고 관리하며 SQL 실행에 대한 모든 것을
        처리하는 객체로 SqlSessionFactoryBean을 통해 SqlSessionFactory
        객체를 한 번만 생성해 사용한다.
    -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--
            DBCP를 사용하기 위한 DataSource를 참조 한다.
        -->
        <property name="dataSource" ref="dataSource" />

        <!--
            추가로 MyBatis 설정이 필요하다면 설정 파일의 위치를 지정할 수 있다.
            이번 예제는 스프링에서 제공하는 DataSource와 TransactionManager를
            사용하기 때문에 MyBatis 설정인 mybatis-config.xml은 사용하지 않는다.

            스프링과 MyBatis를 연동할 경우 MyBatis 설정 파일인
            mybatis-config.xml은 필수가 아니기 때문에 생략할 수  있다.
            우리의 예제도 Mybatis 설정 파일은 사용하지 않는다. 이 설정 파일은
            스프링을 사용하지 않을 때 필수적으로 사용하는 MyBatis 설정 파일이므로
            mybatis-config_참고.xml의 주석은 참고삼아 꼭 한 번 읽어보기 바란다.
        -->

        <!--
            <property name="configLocation" value="classpath:mybatis-config.xml" />
        -->

        <!--
            MyBatis가 동작하면 Mapper를 인식해야 정상적인 DB 작업을 할 수 있다.
            아래와 같이 SqlSessionFactory의 mapperLocations 프로퍼티로
            지정할 수 있다. 아래와 같이 지정하면 "repository/mappers/"를 포함한
            하부 폴더의 "*Mapper.xml"로 끝나는 파일을 Mapper로 인식하게 된다.
        -->
        <property name="mapperLocations"
            value="classpath:repository/mappers/**/*Mapper.xml" />

        <!--
            Mapper 설정이 여러 곳에 나누어져 있을 때 아래와 같이 list로 지정할 수 있다.
        -->
        <!--
        <property name="mapperLocations">
            <list>
                <value>classpath:repository/mappers/BoardMapper.xml</value>
                <value>classpath:repository/mappers/MemberMapper.xml</value>
```

```
        </list>
    </property>
    -->

    <!--
        typeAliasesPackage 속성은 Mapper 설정에서 타입 별칭을 사용할
        클래스가 위치한 패키지를 지정하는데 사용한다. 이 속성에 지정한
        패키지를 자동으로 검색해 클래스 이름으로 타입 별칭을 사용할 수 있도록
        해 준다. 아래와 같이 typeAliasesPackage를 지정하면 value에 지정한
        패키지를 포함한 하부 패키지의 모든 클래스를 완전한 클래스 이름이 아닌
        클래스 이름만 지정하여 사용할 수 있도록 해 준다. 만약 이를 설정하지 않으면
        Mapper 설정에서 타입을 지정할 때 패키지를 포함한 완전한 클래스 이름으로
        지정해야 한다.
    -->
    <property name="typeAliasesPackage" value="com.springstudy.bbs.domain" />

    <!--
        typeAliases 속성을 사용해 Mapper 설정에서 완전한 클래스 이름 대신
        별칭을 사용할 클래스 목록을 지정할 수도 있다.
        typeAliases에 지정한 클래스에 @Alias("Board") 애노테이션을 적용하면
        Mapper 설정에서 Board 라는 별칭으로 타입을 지정할 수 있다.
    -->
    <!--
    <property name="typeAliases">
        <list>
            <value>com.springstudy.bbs.domain.Board</value>
            <value>com.springstudy.bbs.domain.Member</value>
        </list>
    </property>
    -->
</bean>

<!--
    5. 스프링 Bean 설정 파일인 root-context.xml 파일에
       DAO에서 의존하는 SqlSessionTemplate을 Spring Bean으로 정의

    mybatis-spring 모듈은 MyBatis의 SqlSession 기능과 스프링 DB 지원 기능을
    연동해 주는 SqlSessionTemplate 클래스를 제공한다. SqlSessionTemplate은
    SqlSession을 구현해 스프링과 연동하는 기능을 구현했기 때문에 우리가 만드는
    DAO에서 SqlSessionTemplate 객체를 사용해 SqlSession에 정의된 메서드를
    사용할 수 있다.

    SqlSession과 SqlSessionTemplate은 같은 역할을 담당하고 있지만 트랜잭션
    처리에서 다른 부분이 있다. SqlSession은 commit(), rollback() 메서드를
    명시적으로 호출해 트랜잭션을 처리 하지만 SqlSessionTemplate은 스프링이
```

```
    트랜잭션을 처리할 수 있도록 구현되어 있기 때문에 별도로 commit(), rollback()
    메서드를 호출할 필요가 없다.

    SqlSessionTemplate은 SqlSessionFactoryBean을 사용해
    DB 접속해 작업하기 때문에 아래와 같이 생성자로 주입 받도록 설정하면 된다.
-->
<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate"
    c:sqlSessionFactory-ref="sqlSessionFactory" />

<!--
    6. 스프링 Bean 설정 파일인 root-context.xml 파일에
        스프링이 지원하는 TransactionManager를 Spring Bean으로 정의(생략 가능)

    마이바티스는 JDBC 기반이기 때문에 DataSourceTransactionManager를
    이용해 다음과 같은 방식의 트랜잭션을 처리할 수 있다.

    tx 네임스페이스를 이용한 트랜잭션 처리
    @Transactional 애노테이션을 이용한 트랜잭션 처리

    ## 트랜잭션 매니저 정의하기 ##
    스프링은 트랜잭션 처리를 위해 PlatformTransactionManager 인터페이스를
    통해 추상화 하고 각각의 DB 연동 기술에 따라서 PlatformTransactionManager
    구현 클래스를 아래와 같이 제공하고 있다.

    JDBC 기반 : DataSourceTransactionManager
    하이버네이트 : HibernateTransactionManager
    JPA : JpaTransactionManager
    JTA : JtaTransactionManager

    아래는 JDBC 기반 DB 연동기술을 사용하는 스프링 DB 지원 템플릿 클래스
    (JdbcTemplate 등)를 사용하거나 MyBatis를 사용할 경우 트랜잭션 매니저 설정이다.

    DBCP를 사용하기 위한 DataSource를 setter 주입 받도록 설정하면 된다.
-->
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
    p:dataSource-ref="dataSource" />

<!--
    7. 마이바티스를 이용한 DAO 구현
      - SqlSession 구현체 SqlSessionTemplate을 이용한 구현
      - 매퍼(Mapper Interface를) 동적생성을 이용한 구현

    우리는 SqlSessionTemplate을 이용해 DAO를 구현할 것이다. 그러므로 이 설정
    파일에 SqlSessionTemplate을 Spring Bean으로 정의하고 스프링으로 부터
```

```
        주입 받아 사용할 수 있도록 DAO에서 생성자 또는 세터를 추가하면 된다.

        스프링과 MyBatis를 연동해 DAO를 구현할 경우 DAO에서 SqlSessionTemplate을
        사용해 DB 작업을 하게 된다. 이 SqlSessionTemplate은 DAO에서 SQL 쿼리를
        분리한 MyBatisMapper에서 매핑 구문을 호출해 SQL을 실행하는 역할을 담당하는
        객체이며 스프링과 MyBatis를 연동할 때 SqlSessionTemplate은 DB에 실제
        SQL 쿼리를 발행하거나 트랜잭션을 관리하는 핵심 객체 이다.
    -->

    <!--
        8. SQL을 분리한 Mapper 작성(Mapper XML 또는 Mapper Interface)

          일반적으로 Mapper 설정 파일은 "src/main/resources" 폴더에 작성하거나
          이 폴더에 새로운 폴더를 추가해 작성하면 되며 Mapper Interface는
          "src/main/java"에 패키지를 추가해 작성하면 된다.

          우리는 src/main/resources 폴더에 Mapper XML을 작성해 사용할 것이다.
    -->

    <!--
        ## Service 정의하기 ##
        ## Dao 정의하기 ##
        servlet-context.xml 파일에 <context:component-scan />을 적용했기
        때문에 Spring MVC Annotation 기반의 Bean을 선언하고 검색하게 되므로
        별도로 Service 클래스와 Dao 클래스를 이 Bean 설정 파일에 정의하지 않고
        서비스 클래스에는 @Service 애노테이션과 Dao 클래스에는 @Repository
        애노테이션을 적용시키면 Spring DI 컨테이너가 Bean을 스캔해 등록한다.
    -->
</beans>
```

1-7) DB 접속정보 설정

- src/main/resources/datasource/datasource.properties

```
## ########################################
## MySQL DB와 DBCP를 사용하기 위한 설정
##
db.driverClassName=com.mysql.cj.jdbc.Driver
db.url=jdbc:mysql://localhost:3306/spring?useSSL=false&allowPublicKeyRetrieval=true&useU
nicode=true&characterEncoding=utf8
db.username=root
db.password=12345678
## DBCP가 시작될 때 생성할 커넥션 수
db.initialSize=5
## DBCP에서 동시에 제공할 최대 커넥션 수 commons 2.0
```

```
db.maxTotal=10
## 휴먼 상태에서 유지할 최대 커넥션 수
db.maxIdle=3
```

1-8) DataSource Bean 설정

- src/main/resources/datasource/datasource.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.2.xsd">

    <!-- 프로퍼티 대치 변수 설정자(Property Placeholder Configurer)를 설정한다. -->
    <context:property-placeholder
        location="classpath:datasource/dbcpdatasource.properties" />

    <!--
        Apache Commons의 DBCP의 BasicDataSource 타입의 Bean을 선언하고
        위에서 프로퍼티 대치 변수 설정자로 지정하여 properties 파일로 부터 읽어온
        데이터를 대치 변수를 사용해 dataSource의 각 프로퍼티에 지정하고 있다.
        이 프로젝트에서는 commons-dbcp2-2.6.0 사용하였다.

        Group Id : org.apache.commons
        Artifact Id : commons-dbcp2
        Version : 2.6.0
     -->
    <bean id="dataSource" destroy-method="close"
        class="org.apache.commons.dbcp2.BasicDataSource" >
        <property name="driverClassName" value="${db.driverClassName}" />
        <property name="url" value="${db.url}" />
        <property name="username" value="${db.username}" />
        <property name="password" value="${db.password}" />
        <property name="initialSize" value="${db.initialSize}" />
        <property name="maxTotal" value="${db.maxTotal}" />
        <property name="maxIdle" value="${db.maxIdle}" />
    </bean>
</beans>
```

2) Spring MVC와 MyBatis를 활용한 게시판 CRUD 구현

Spring MVC와 MyBatis를 활용해 게시판의 기본적인 기능인 게시 글 리스트 보기, 상세보기, 쓰기, 수정, 삭제하기를 구현할 것이다. 일반적으로 Spring MVC를 활용해 웹 애플리케이션을 구현할 때 Interface 규칙을 사용하므로 우리도 Service와 DAO를 Interface 규칙을 적용해 구현할 것이다. 그리고 게시판을 구현할 때 게시 글 리스트 보기를 먼저 구현하고 상세보기, 글쓰기, 수정하기, 삭제하기 순으로 구현할 것이다.
기본적인 게시판 CRUD 기능이 완료되면 게시판 페이징 처리와 검색 기능을 추가하고 파일 업로드, 다운로드를 구현할 계획이다.

2-1) 게시 글 리스트 보기 요청처리

▶ Domain 클래스

- com.springstudy.bbs.domain.Board

```
public class Board {
    // no, title, writer, content, reg_date, read_count, pass, file1
    private int no;
    private String title;
    private String writer;
    private String content;
    private Timestamp regDate;
    private int readCount;
    private String pass;
    private String file1;

    public Board() { }
    public Board(int no, String title, String writer, String content,
            Timestamp regDate, int readCount, String pass, String file1) {
        this.no = no;
        this.title = title;
        this.writer = writer;
        this.content = content;
        this.regDate = regDate;
        this.readCount = readCount;
        this.pass = pass;
        this.file1 = file1;
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
```

```
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getWriter() {
        return writer;
    }
    public void setWriter(String writer) {
        this.writer = writer;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public Timestamp getRegDate() {
        return regDate;
    }
    public void setRegDate(Timestamp regDate) {
        this.regDate = regDate;
    }
    public int getReadCount() {
        return readCount;
    }
    public void setReadCount(int readCount) {
        this.readCount = readCount;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
    public String getFile1() {
        return file1;
    }
    public void setFile1(String file1) {
        this.file1 = file1;
    }
}
```

▶ 게시 글 CRUD 관련 SQL을 분리한 Mapper

src/main/resources/repository/mappers/에 있는 Mapper-Template를 복사해 아래 맵퍼 파일을 만들어 작성하자.

- src/main/resources/repository/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">

<!--
    맵퍼의 namespace 속성은 맵핑 구문을 그룹핑 하는 역할을 한다.
    여러 맵퍼에서 맵핑 구문의 id 속성의 값이 중복되더라도 namespace와
    맵핑 구문의 id 속성에 지정한 값을 합쳐서 호출하기 때문에 맵핑 구문이
    중복되지 않게 분류하여 관리할 수 있다.
    테이블 이름을 바탕으로 namespace를 지정하고 맵핑 구문의 id 속성의
    값은 SQL 문의 맥락에 따라서 명명하는 것이 일반적 이다.

    parameterType 속성에는 주로 SQL 문의 조건에 사용할 파라미터의
    데이터 타입을 지정하는 속성으로 자바 원시 타입(기본 타입, String)과 자바빈,
    Map과 같은 타입을 지정할 수 있다. resultType 속성도 parameterType
    속성에서 지정한 타입을 많이 사용한다. parameterType과 resultType에
    지정할 데이터 타입이 자바 원시 타입이면 생략가능하다.

    http://www.mybatis.org/mybatis-3/ko/index.html 참고
-->
<mapper namespace="com.springstudy.bbs.mapper.BoardMapper" >

    <!--
        게시 글 리스트를 가져오는 맵핑 구문

        테이블의 컬럼명은 일반적으로 언더스코어 표기법("_")을 사용하는 경우가
        많고 클래스의 인스턴스 맴버는 카멜표기법을 사용한다.
        테이블의 컬럼명과 모델 클래스의 프로퍼티 이름이 다른 경우 아래와 같이
        SELECT 쿼리에 별칭을 사용해 모델 클래스의 프로퍼티 이름과 동일하게
        맞춰야 한다. 그렇지 않으면 오류는 발생하지 않지만 데이터를 읽어 올
        수 없다.

        SELECT read_count AS readCount FROM springbbs

        resultType에 Board를 지정하고 DAO 클래스에서 SqlSession의
        selectList() 메서드를 호출하면 List<Board> 객체로 받을 수 있다.

        아래는 테이블에 언더스코어 표기법으로 작성된 컬럼이 있기 때문에 이 컬럼의
        값은 제대로 읽어오지 못한다. 그래서 이 맵퍼 파일 맨 아래에 resultMap을
        boardResultMap으로 선언하고 아래 select 태그에 resultMap을
```

```
        boardResultMap으로 지정하면 모든 컬럼의 값을 읽어 올 수 있다.
    -->
    <select id="boardList" resultType="Board" resultMap="boardResultMap">
        SELECT
            *
        FROM springbbs
        ORDER BY no DESC
    </select>

    <!--
        Board 클래스의 프로퍼티와 테이블의 컬럼을 맵핑하는 ResultMap

        테이블에 언터스코어 표기법으로 작성된 컬럼이 존재하면 도메인 객체의
        카멜케이스된 프로퍼티를 통해 컬럼의 값을 제대로 읽어오지 못한다.
        아래와 같이 resultMap을 정의해 테이블의 컬럼과 도메인 객체의
        프로퍼티를 설정하면 제대로 읽어 올 수 있다.

        resultMap 태그에 type 속성은 쿼리 결과를 매핑할 자바 도메인 객체를
        지정하고 그 하부에 <id> 태그를 사용해 기본키 컬럼을 지정한다.
        나머지 컬럼은 <result> 태그를 사용해 지정하면 도메인 객체로 맵핑할 수 있다.
    -->
    <resultMap id="boardResultMap" type="Board">
        <id property="no" column="no" />
        <result property="title" column="title" />
        <result property="writer" column="writer" />
        <result property="content" column="content" />
        <result property="regDate" column="reg_date" />
        <result property="readCount" column="read_count" />
        <result property="pass" column="pass" />
        <result property="file1" column="file1" />
    </resultMap>
</mapper>
```

▶ DAO(Data Access Object) 계층 구현

프로젝트에 com.springstudy.bbs.dao 패키지를 만들고 BoardDao 인터페이스를 정의한다.

- com.springstudy.bbs.dao.BoardDao

```
public interface BoardDao {

    /* 한 페이지에 보여 질 게시 글 리스트 요청 시 호출되는 메소드
     * 현재 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환 하는 메소드
     **/
    public abstract List<Board> boardList();
```

```
    /* 게시 글 상세보기 요청 시 호출되는 메서드
     * no에 해당하는 게시 글 을 DB에서 읽어와 Board 객체로 반환 하는 메서드
     **/
    public abstract Board getBoard(int no);

    /* 게시 글쓰기 요청 시 호출되는 메서드
     * 게시 글쓰기 요청 시 게시 글 내용을 Board 객체로 받아 DB에 추가하는 메서드
     **/
    public abstract void insertBoard(Board board);

    /* 게시 글 수정과 삭제 할 때 비밀번호 체크에서 호출되는 메서드
     * 게시 글 수정, 삭제 시 no에 해당하는 비밀번호를 DB에서 읽어와 반환하는 메서드
     **/
    public String isPassCheck(int no, String pass);

    /* 게시 글 수정 요청 시 호출되는 메서드
     * 게시 글 수정 요청 시 수정된 내용을 Board 객체로 받아 DB에 수정하는 메서드
     **/
    public abstract void updateBoard(Board board);

    /* 게시 글 삭제 요청 시 호출되는 메서드
     * no에 해당 하는 게시 글을 DB에서 삭제하는 메서드
     **/
    public abstract void deleteBoard(int no);
}
```

▶ BoardDao 인터페이스 구현 클래스

com.springstudy.bbs.dao 패키지에 BorardDao를 구현하는 BoardDaoImpl 클래스를 만들고 게시 글 리스트를 MyBatis의 SessionTemplate을 이용해 DB로부터 읽어와 반환하는 메서드를 구현한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
// 이 클래스가 데이터 액세스(데이터 저장소) 계층의 컴포넌트(Bean) 임을 선언한다.
@Repository
public class BoardDaoImpl implements BoardDao {

    /* src/main/resources/repository/mappers/BoardMapper.xml에
     * 정의한 Mapper namespace를 상수로 정의
     **/
    private final String NAME_SPACE = "com.springstudy.bbs.mapper.BoardMapper";

    /* 이 예제는 MyBatis가 제공하는 SqlSessionTemplate 객체를 사용하기
     * 때문에 스프링으로부터 DI 받을 수 있도록 생성자나 setter를 준비해야 한다.
     *
     * mybatis-spring 모듈은 MyBatis의 SqlSession 기능과 스프링 DB 지원 기능을
```

```
 * 연동해 주는 SqlSessionTemplate 클래스를 제공한다. SqlSessionTemplate은
 * SqlSession을 구현해 스프링 연동 부분을 구현하였기 때문에 우리가 만드는 DAO에서
 * SqlSessionTemplate 객체를 사용해 SqlSession에 정의된 메서드를 사용할 수 있다.
 *
 * SqlSession과 SqlSessionTemplate는 같은 역할을 담당하고 있지만 트랜잭션
 * 처리에서 다른 부분이 있다. SqlSession은 commit(), rollback() 메서드를
 * 명시적으로 호출해 트랜잭션을 처리 하지만 SqlSessionTemplate은 스프링이
 * 트랜잭션을 처리할 수 있도록 구현되어 있기 때문에 별도로 commit(), rollback()
 * 메서드를 호출할 필요가 없다.
 **/
private SqlSessionTemplate sqlSession;

/* setter 주입 방식은 스프링이 기본 생성자를 통해 이 클래스의 인스턴스를
 * 생성한 후 setter 주입 방식으로 SqlSessionTemplate 타입의 객체를
 * 주입하기 때문에 기본 생성자가 존재해야 하지만 이 클래스에 다른 생성자가
 * 존재하지 않으므로 컴파일러에 의해 기본 생성자가 만들어 진다.
 **/
@Autowired
public void setSqlSession(SqlSessionTemplate sqlSession) {
    this.sqlSession = sqlSession;
}

/* 한 페이지에 보여 질 게시 글 리스트 요청 시 호출되는 메소드
 * 현재 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환 하는 메소드
 **/
@Override
public List<Board> boardList() {

    /* BoardMapper.xml에서 맵핑 구문을 작성하고 아래와 같이 SqlSession
     * 객체의 메서드를 호출하면서 맵핑 설정을 지정하게 되면 이 메서드 안에서
     * PreparedStatement 객체를 생성하고 PreparedStatement 객체에
     * 필요한 파라미터가 설정된다.
     *
     * SqlSessionTemplate 객체의 select(), selectOne(), selectList()
     * 메서드를 호출하면 PreparedStatement 객체의 executeQuery() 메서드를
     * 실행하고 쿼리를 발행한 결과인 ResultSet 객체에서 데이터를 읽어와 모델
     * 클래스인 Board 객체를 생성하고 이 객체에 값을 설정하게 된다.
     *
     * 아래와 같이 SqlSessionTemplate의 메서드가 호출되면
     * repository/mappers/BoardMapper.xml 맵퍼 파일에서
     * mapper 태그의 namespace 속성에 지정한
     * com.springstudy.bbs.mapper.BoardMapper인 맵퍼가 선택되고
     * 그 하부에 <select> 태그의 id 속성에 지정한 boardList인 맵핑 구문이
     * 선택된다. 그리고 MyBatis 내부에서 JDBC 코드로 변환되어 실행된다.
     *
```

```
         * 매핑 구문에 resultType 속성에 Board를 지정했기 때문에 요청한
         * 페이지에 해당하는 게시 글 리스트가 담긴 List<Board> 객체가
         * 반환된다. Board 테이블에 게시 글 정보가 하나도 없으면 null이 반환 된다.
         *
         * 만약 SQL 파라미터를 지정해야 한다면 두 번째 인수에 필요한 파라미터를
         * 지정하면 되는데 파라미터가 여러 개일 경우 Map 객체에 담아 두 번째
         * 인수로 지정하면 된다.
         **/
        return sqlSession.selectList(NAME_SPACE + ".boardList");
    }
}
```

▶ Service 계층 구현

프로젝트에 com.springstudy.bbs.service 패키지를 만들고 BoardService 인터페이스를 정의한다.

- com.springstudy.bbs.service.BoardService

```
public interface BoardService {

    /* BoardDao를 이용해 게시판 테이블에서
     * 현재 페이지에 해당하는 게시 글 리스트를 읽어와 반환 하는 메소드
     **/
    public abstract List<Board> boardList();

    /* BoardDao를 이용해 게시판 테이블에서
     * no에 해당하는 게시 글 을 읽어와 반환하는 메서드
     **/
    public abstract Board getBoard(int no);

    // BoardDao를 이용해 새로운 게시 글을 추가하는 메서드
    public abstract void insertBoard(Board board);

    /* 게시 글 수정과 삭제 할 때 비밀번호가 맞는지 체크하는 메서드
     *
     * - 게시 글의 비밀번호가 맞으면 : true를 반환
     * - 게시 글의 비밀번호가 맞지 않으면 : false를 반환
     **/
    public boolean isPassCheck(int no, String pass);

    // BoardDao를 이용해 게시 글을 수정하는 메서드
    public abstract void updateBoard(Board board);

    // BoardDao를 이용해 no에 해당하는 게시 글을 삭제하는 메서드
    public abstract void deleteBoard(int no);
}
```

▶ BoardService 인터페이스 구현 클래스

com.springstudy.bbs.service 패키지에 BorardService를 구현하는 BoardServiceImpl 클래스를 만들고 BoardDao를 이용해 게시 글 리스트를 반환하는 메서드를 구현한다.

- com.springstudy.bbs.service.BoardServiceImpl

```
// 이 클래스가 서비스 계층(비즈니스 로직)의 컴포넌트(Bean) 임을 선언하고 있다.
@Service
public class BoardServiceImpl implements BoardService {

    /* 인스턴스 필드에 @Autowired annotation을 사용하면 접근지정자가
     * private이고 setter 메서드가 없다 하더라도 문제없이 주입 된다.
     * 하지만 우리는 항상 setter 메서드를 준비하는 습관을 들일 수 있도록 하자.
     *
     * setter 주입 방식은 스프링이 기본 생성자를 통해 이 클래스의 인스턴스를
     * 생성한 후 setter 주입 방식으로 BoardDao 타입의 객체를 주입하기 때문에
     * 기본 생성자가 존재해야 하지만 이 클래스에 다른 생성자가 존재하지 않으므로
     * 컴파일러에 의해 기본 생성자가 만들어 진다.
     **/
    @Autowired
    private BoardDao boardDao;

    public void setBoardDao(BoardDao boardDao) {
        this.boardDao = boardDao;
    }

    /* BoardDao를 이용해 게시판 테이블에서
     * 현재 페이지에 해당하는 게시 글 리스트를 읽어와 반환 하는 메소드
     **/
    @Override
    public List<Board> boardList() {
        return boardDao.boardList();
    }
}
```

▶ Controller 클래스

프로젝트에 com.springstudy.bbs.controller 패키지를 만들고 게시판 관련 요청을 처리하는 BoardController 클래스를 구현한다.

- com.springstudy.bbs.controller.BoardController

```
//스프링 MVC의 컨트롤러임을 선언하고 있다.
@Controller
```

```java
public class BoardController {

    /*  인스턴스 필드에 @Autowired annotation을 사용하면 접근지정자가
     * private이고 setter 메서드가 없다 하더라도 문제없이 주입 된다.
     * 하지만 우리는 항상 setter 메서드를 준비하는 습관을 들일 수 있도록 하자.
     *
     * setter 주입 방식은 스프링이 기본 생성자를 통해 이 클래스의 인스턴스를
     * 생성한 후 setter 주입 방식으로 BoardService 타입의 객체를 주입하기 때문에
     * 기본 생성자가 존재해야 하지만 이 클래스에 다른 생성자가 존재하지 않으므로
     * 컴파일러에 의해 기본 생성자가 만들어 진다.
     **/
    @Autowired
    private BoardService boardService;

    public void setBoardService(BoardService boardService) {
        this.boardService = boardService;
    }

    /* @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입
     *
     * 1. Controller 메서드의 파라미터 타입으로 지정할 수 있는 객체와 애노테이션
     * @RequestMapping 애노테이션이 적용된 컨트롤러 메서드의 파라미터에
     * 아래와 같은 객체와 애노테이션을 사용할 수 있도록 지원하고 있다.
     *
     * - HttpServletRequest, HttpServletResponse
     *   요청/응답을 처리하기 위한 서블릿 API
     *
     * - HttpSession
     *   HTTP 세션을 위한 서블릿 API
     *
     * - org.springframework.ui.Model, ModelMap, java.util.Map
     *   뷰에 모델 데이터를 전달하기 위한 모델 객체
     *
     * - 커맨드 객체(VO, DTO)
     *   요청 데이터를 저장할 객체
     *
     * - Errors, BindingResult
     *   검증 결과를 저장할 객체로 커맨드 객체 바로 뒤에 위치 시켜야 한다.
     *
     * - @RequestParam
     *   HTTP 요청 파라미터의 값을 메서드의 파라미터로 매핑하기 위한 애노테이션
     *
     * - @RequestHeader
     *   HTTP 요청 헤더의 값을 파라미터로 받기 위한 애노테이션
     *
```

```
 * - @RequestCookie
 *   Cookie 데이터를 파라미터로 받기 위한 애노테이션
 *
 * - @RequestVariable
 *   RESTful API 방식의 파라미터를 받기 위한 경로 변수 설정 애노테이션
 *
 * - @RequestBody
 *   요청 몸체의 데이터를 자바 객체로 변환하기 위한 애노테이션
 *   String이나 JSON으로 넘어오는 요청 몸체의 데이터를 자바 객체로
 *   변환하기 위한 사용하는 애노테이션 이다.
 *
 * - Writer, OutputStream
 *   응답 데이터를 직접 작성할 때 메서드의 파라미터로 지정해 사용한다.
 *
 * 2. Controller 메서드의 반환 타입으로 지정할 수 있는 객체와 애노테이션
 * - String
 *   뷰 이름을 반환할 때 메서드의 반환 타입으로 지정
 *
 * - void
 *   컨트롤러의 메서드에서 직접 응답 데이터를 작성할 경우 지정
 *
 * - ModelAndView
 *   모델과 뷰 정보를 함께 반환해야 할 경우 지정
 *   이전의 컨트롤는 스프링이 지원한는 Controller 인터페이스를
 *   구현해야 했는데 이때 많이 사용하던 반환 타입이다.
 *
 * - 자바 객체
 *   메서드에 @ResponseBody가 적용된 경우나 메서드에서 반환되는
 *   객체를 JSON 또는 XML과 같은 양식으로 응답을 변환 할 경우에 사용한다.
 **/

/* 게시 글 리스트 보기 요청을 처리하는 메서드
 *
 * @RequestMapping은 클래스 레벨과 메서드 레벨에 지정할 수 있다.
 * @RequestMapping의 ()에 처리할 요청 URI만 지정할 때는 value 속성을
 * 생략하고 처리할 요청 URI를 String 또는 String 배열을 지정할 수 있지만
 * 다른 속성을 같이 지정할 경우 value 속성에 처리할 요청 URI를 지정해야 한다.
 * 또한 method 속성을 지정해 컨트롤러가 처리할 HTTP 요청 방식을
 * 지정할 수 있는데 아래는 "/boardList", "/list"로 들어오는 GET 방식의
 * 요청을 이 메서드가 처리할 수 있도록 설정한 것이다.
 * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
 *
 * 요청을 처리한 결과를 뷰에 전달하기 위해 사용하는 것이 Model 객체이다.
 * 컨트롤러는 요청을 처리한 결과 데이터를 모델에 담아 뷰로 전달하고 뷰는
 * 모델로 부터 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 애노테이션이 적용된 메서드의 파라미터에 Model
     * 을 지정하면 스프링이 이 메서드를 호출하면서 Model 타입의 객체를 넘겨준다.
     * 우리는 Model을 받아 이 객체에 결과 데이터를 담기만 하면 뷰로 전달된다.
     **/
    @RequestMapping(value= {"/boardList", "/list"}, method=RequestMethod.GET)
    public String boardList(Model model) {


        // Service 클래스를 이용해 게시 글 리스트를 가져온다.
        List<Board> bList = boardService.boardList();

        /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
         * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
         **/
        model.addAttribute("bList", bList);


        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "boardList";
    }
}
```

▶ 웹 템플릿 View

스프링 프레임워크를 활용해 서버프로그램 구현을 학습하는 것이므로 웹 템플릿에 사용되는 index.jsp, header.jsp, footer.jsp 파일과 bootstrap, css, 자바스크립트 파일 등은 완성 프로젝트인 springstudy-bbs01 프로젝트에서 복사해서 실습용 프로젝트로 가져오자.

완성 프로젝트	실습 프로젝트
/WEB-INF/index.jsp 파일	/WEB-INF/ 폴더에 복사
/WEB-INF/template 폴더의 파일	/WEB-INF/template 폴더에 복사
/webapp/resources 폴더의 파일	/webapp/resources 폴더에 복사

* 메인 템플릿 페이지

- src/main/webapp/WEB-INF/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Spring MVC 게시판</title>
    <!--
        BoardController에서 클래스 레벨에 @RequestMapping 애노테이션을 사용해
        별도로 경로 매핑을 하지 않고 boardList(Model model) 메서드에만
        @RequestMapping({"/boardList", "/list"}) 애노테이션으로 요청 매핑을 설정했다.
        그리고 WEB-INF/spring/appServlet/servlet-context.xml에서 정적 리소스와
        관련된 url 맵핑을 아래와 같이 설정했기 때문에
        <mvc:resources mapping="/resources/**" location="/resources/" />
        css의 위치를 "resources/css/index.css"와 같이 지정해야 한다.

        브라우저 주소 표시줄에 http://localhost:8080/springstudy-bbs01/list
        또는 http://localhost:8080/springstudy-bbs01/boardList 등으로
        표시되므로 css 디렉터리는 ContextRoot/resources/css에 위치하기 때문에 현재
        위치를 기준으로 상대 참조 방식으로 "resources/css/index.css"를 지정해야 한다.
    -->
    <title>스프링 게시판</title>
    <link href="resources/bootstrap/bootstrap.min.css" rel="stylesheet" >
    <style>
    </style>
    <script src="resources/js/jquery-3.2.1.min.js"></script>
    <script src="resources/js/formcheck.js"></script>
</head>
<body>
    <div class="container">
        <%@ include file="template/header.jsp" %>
        <jsp:include page="${ param.body }" />
        <%@ include file="template/footer.jsp" %>
    </div>
    <script src="resources/bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>
```

*** 템플릿 헤더 페이지**

- src/main/webapp/WEB-INF/template/header.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header  -->
```

```
<div class="row border-bottom border-primary" id="global-header">
    <div class="col-4">
        <p><img src="https://via.placeholder.com/200x100"></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">게시 글 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>
```

*** 템플릿 푸터 페이지**

- src/main/webapp/WEB-INF/template/footer.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- footer -->
<div class="row border-top border-primary my-5" id="global-footer">
    <div class="col text-center py-3">
        <p>고객상담 전화주문:1234-5678 사업자등록번호 :111-11-123456
    대표이사: 홍길동  통신판매업 서울 제 000000호<br/>
    개인정보관리책임자:임꺽정  분쟁조정기관표시 : 소비자보호원, 전자거래분쟁중재위원회<br/>

    Copyright (c) 2016 Spring2U Corp. All right Reserved.
```

```
        </p>
    </div>
</div>
```

▶ 게시 글 리스트 보기 View

- src/main/webapp/WEB-INF/views/boardList.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!-- 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">게시 글 리스트</h2>
            </div>
        </div>
        <form name="searchForm" id="searchForm" action="#"
            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 href="writeForm" class="btn btn-outline-success">글쓰기</a>
            </div>
        </div>
        <div class="row my-3">
            <div class="col">
                <table class="table table-hover">
                    <thead>
                        <tr class="table-dark">
```

```
                        <th>NO</th>
                        <th>제목</th>
                        <th>작성자</th>
                        <th>작성일</th>
                        <th>조회수</th>
                    </tr>
                </thead>
                <tbody class="text-secondary">
                    <!-- 게시 글이 있는 경우 -->
                <c:if test="${ not empty bList }">
                    <c:forEach var="b" items="${bList}">
                    <tr>
                        <td>${ b.no }</td>
                        <td><a href="boardDetail?no=${b.no}"
class="text-decoration-none link-secondary">${ b.title }</a></td>
                        <td>${ b.writer }</td>
                        <td>${ b.regDate }</td>
                        <td>${ b.readCount }</td>
                    </tr>
                    </c:forEach>
                </c:if>
                <!-- 게시 글이 없는 경우 -->
                <c:if test="${ empty bList }">
                    <tr>
                        <td colspan="5" class="text-center">
                            게시 글이 존재하지 않습니다.
                        </td>
                    </tr>
                </c:if>
                </tbody>
            </table>
        </div>
    </div>
</div>
```

2-2) 게시 글 상세보기 요청처리

앞에서 Spring MVC와 MyBatis를 활용해 웹 애플리케이션을 구현하기 위한 설정과 MyBatis Mapper와 DAO를 이용해 게시판 테이블로부터 게시 글 리스트를 읽어와 출력하는 과정에 대해서 알아보았다. 지금부터는 앞에서 설정한 Spring MVC 설정을 그대로 사용하면서 DAO, Service, View 페이지를 구현해 추가하기만 하면 된다.

▶ 게시 글 관련 SQL을 분리한 Mapper 에 추가

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

```
<!--
    no에 해당하는 게시 글 하나를 가져오는 맵핑 구문

    resultMap 속성을 사용해 아래에서 resultMap 태그로 정의한 id를 지정하면 SQL
    쿼리에 컬럼명의 별칭을 사용하지 않고 자바 모델 객체로 변환할 수 있다.

    root-context.xml에서 SqlSessionFactoryBean를 Bean으로 정의할 때
    typeAliasesPackage를 com.springstudy.bbs.domain으로 지정했기
    때문에 resultType에 클래스 이름만 지정했다.

    DAO에서 no에 해당하는 게시 글을 조회할 때 기본형인 no를 selectOne()
    메서드의 두 번째 인수로 지정했기 때문에 parameterType은 생략할 수 있다.

    SQL문의 조건에 사용할 파라미터는 아래와 같이 #{} 로 감싸서 지정하면 된다.
-->
<select id="getBoard" resultType="Board" resultMap="boardResultMap">
    SELECT
        *
    FROM springbbs
    WHERE no = #{no}
</select>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 no에 해당하는 게시 글을 DB로부터 읽어와 Board 객체로 반환하는 아래 메서드를 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
/* 게시 글 상세보기 요청 시 호출되는 메서드
 * no에 해당하는 게시 글 을 DB에서 읽어와 Board 객체로 반환 하는 메서드
 **/
@Override
public Board getBoard(int no) {

    // getBoard 맵핑 구문을 호출하면서 게시 글 번호인 no를 파라미터로 지정했다.
    return sqlSession.selectOne(NAME_SPACE + ".getBoard", no);
}
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardDao를 이용해 no에 해당하는 게시 글 상세 내용을 반환하는 아래 메서드를 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl

```
/* BoardDao를 이용해 게시판 테이블에서
 * no에 해당하는 게시 글 을 읽어와 반환하는 메서드
 **/
@Override
public Board getBoard(int no) {
    return boardDao.getBoard(no);
}
```

▶ Controller 클래스

BoardController 클래스에 게시 글 상세보기 요청을 처리하는 아래 메서드를 추가한다.

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 상세보기 요청을 처리하는 메서드
 *
 * 아래 @RequestMapping에서 method를 생략했기 때문에 "/boardDetail"로
 * 들어오는 GET 방식과 POST 방식의 요청 모두를 처리할 수 있다.
 *
 * 스프링은 클라이언트로부터 넘어 오는 요청 파라미터를 받을 수 있는 여러 가지
 * 방법을 제공하고 있다. 아래와 같이 Controller 메서드에 요청 파라미터 이름과
 * 동일한 이름의 메서드 파라미터를 지정하면 스프링으로부터 요청 파라미터를 넘겨
 * 받을 수 있다. 만약 요청 파라미터와 메서드의 파라미터 이름이 다른 경우 아래와
 * 같이 메서드의 파라미터 앞에 @RequestParam("요청 파라미터 이름")을
 * 사용해 요청 파라미터의 이름을 지정하면 된다.
 *
 * @RequestMapping 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.
 *
 * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터
```

```
     * 이름과 요청 파라미터의 이름이 같은 경우 @RequestParam 애노테이션을
     * 지정하지 않아도 스프링으로부터 요청 파라미터를 받을 수 있다.
     **/
    @RequestMapping("/boardDetail")
    public String boardDetail(Model model, int no) {

        // Service 클래스를 이용해 no에 해당하는 게시 글을 가져온다.
        Board board = boardService.getBoard(no);

        /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
         * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
         **/
        model.addAttribute("board", board);

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "boardDetail";
    }
```

▶ 게시 글 상세보기 View

- src/main/webapp/WEB-INF/views/boardDetail.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!-- content -->
<div class="row my-5" id="global-content">
    <div class="col">
        <form name="checkForm" id="checkForm">
            <input type="hidden" name="no" id="no" 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">게시 글 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">${ board.title }</td>
                        </tr>
                        <tr>
                            <th>글쓴이</th>
                            <td>${ board.writer }</td>
                            <th>작성일</th>
                            <td><fmt:formatDate value="${ board.regDate }"
                                pattern="yyyy-MM-dd HH:mm:ss" /></td>
                        </tr>
                        <tr>
                            <th>비밀번호</th>
                            <td>
                                <div class="col-sm-8">
                                    <input class="form-control" type="password"
name="pass" id="pass">
                                </div>
                            </td>
                            <th>조회수</th>
                            <td>${ board.readCount }</td>
                        </tr>
                        <tr>
                            <th>파    일</th>
                            <td colspan="3">
                            <c:if test="${ empty board.file1 }">
                                첨부파일 없음
                            </c:if>
                            <c:if test="${ not empty board.file1 }">
                                <a href="upload/${ board.file1 }">파일 다운로드</a>
                            </c:if>
                            </td>
                        </tr>
                        <tr>
                            <td colspan="4">
                                <pre>${ board.content }</pre>
                            </td>
                        </tr>
                    </tbody>
                </table>
            </div>
        </div>
        <div class="row my-3">
            <div class="col text-center">
                <input class="btn btn-warning" type="button" id="detailUpdate" value="수정
```

```
하기"/>
                  <input class="btn btn-danger"  type="button" id="detailDelete"
value="삭제하기" />
                  <input class="btn btn-primary" type="button" value="목록보기"
                        onclick="location.href='boardList'"/>
            </div>
        </div>
    </div>
</div>
```

2-3) 게시 글쓰기 폼 요청처리

게시 글쓰기는 게시 글을 작성할 수 있는 폼을 보여주는 요청과 게시 글쓰기 폼에서 사용자가 작성한 게시 글을 저장하는 요청으로 나누어 처리해야 한다. 게시 글쓰기 폼을 보여주는 요청은 단순히 폼을 화면에 보여주기만 하면 되므로 Service 계층과 DAO 계층의 클래스는 구현할 필요가 없다. 이렇게 클라이언트의 요청이 특별한 처리 없이 단순히 뷰만 보여줘야 하는 경우 Spring MVC 설정 파일인 servlet-context.xml에 뷰 전용 컨트롤러를 정의해 뷰 페이지를 지정하면 Controller에서도 이 요청을 처리하는 메서드를 별도로 구현하지 않아도 된다.
앞에서 servlet-context.xml에 게시 글쓰기 요청에 대한 뷰 전용 컨트롤러를 아래와 같이 정의하였기 때문에 writeForm과 boardWrite로 들어오는 GET 방식 요청에 대한 별도의 처리는 필요 없고 아래에서 view-name에 지정한 뷰 페이지만 작성하면 된다. 이때 뷰 페이지의 위치는 Spring MVC 설정 파일인 servlet-context.xml에 ViewResolver 클래스를 정의할 때 prefix에 지정한 위치(WEB-INF/views)를 참고하고 확장자는 ViewResolver의 suffix에 지정한 .jsp를 사용해 파일 명을 지정하면 된다. 그러므로 뷰 페이지의 위치와 파일 명은 /WEB-INF/views/ 폴더에 writeForm.jsp로 작성하면 된다.

```
<view-controller path="/writeForm" view-name="writeForm" />
<view-controller path="/boardWrite" view-name="writeForm" />
```

▶ 게시 글쓰기 폼 View

- src/main/webapp/WEB-INF/views/writeForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- 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">게시 글쓰기</h2>
            </div>
        </div>
        <form name="writeForm" action="writeProcess" 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"/>
                      <input type="button" value="목록보기"
                        onclick="location.href='boardList'" class="btn btn-primary"/>
            </div>
        </form>
    </div>
</div>
```

▶ JavaScript

게시 글쓰기 폼의 유효성 검사 자바스크립트를 아래의 formcheck.js 파일을 새롭게 만들어 작성한다.

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

```
$(function() {
    // 게시 글쓰기 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;
        }
    });
});
```

2-4) 게시 글쓰기 요청처리

사용자가 게시 글쓰기 폼에서 작성한 게시 글을 저장하는 요청은 게시 글을 DB에 저장한 후 게시 글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다.

▶ 게시 글 관련 SQL을 분리한 Mapper 에 추가

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

```
    <!--
        게시 글을 테이블에 추가하는 맵핑 구문

        아래는 DAO 클래스의 insertBoard(Board board) 메서드에서
        사용하는 맵핑 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.

        SqlSessionTemplate의 insert() 메서드의 반환 타입이 int 이므로
        resultType은 생략 가능하다.

        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)

            VALUES(#{title}, #{writer}, #{content},
                SYSDATE(), #{readCount}, #{pass})
    </insert>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 사용자가 입력한 게시 글을 Board 객체로 넘겨받아 DB에 저장하는 아래 메서드를 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
    /* 게시 글쓰기 요청 시 호출되는 메서드
     * 게시 글쓰기 요청 시 게시 글 내용을 Board 객체로 받아 DB에 추가하는 메서드
     **/
    @Override
    public void insertBoard(Board board) {

        // insertBoard 맵핑 구문을 호출하면서 Board 객체를 파라미터로 지정했다.
        sqlSession.insert(NAME_SPACE + ".insertBoard", board);
    }
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardDao를 이용해 게시 글을 저장하는 아래 메서드를 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl

```
    // BoardDao를 이용해 새로운 게시 글을 추가하는 메서드
    @Override
    public void insertBoard(Board board) {
        boardDao.insertBoard(board);
    }
```

▶ Controller 클래스

BoardController 클래스에 게시 글쓰기 요청을 처리하는 메서드를 추가한다.
이번에 게시 글쓰기 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 VO 객체에 저장했지만 스프링이 지원하는 기능을 사용하면 보다 편리하게 요청 파라미터를 VO(스프링에서는 주로 도메인 객체로 부른다.) 객체로 만들 수 있다.

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글쓰기 폼에서 들어오는 게시 글쓰기 요청을 처리하는 메서드
 *
 * @RequestMapping의 ()에 value="/writeProcess", method=RequestMethod.POST를
 * 지정해 "/writeProcess"로 들어오는 POST 방식의 요청을 처리하는 메서드를
 * 지정한 것이다.
 *
 * 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.
 *
 * @RequestMapping 애노테이션이 적용된 컨트롤러 메서드에 커맨드 객체를
 * 파라미터로 지정하면 커맨드 객체의 프로퍼티와 동일한 이름을 가진 요청
 * 파라미터의 데이터를 스프링이 자동으로 설정해 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 정수형으로 형 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명은 setBoardService() 메서드 바로 아래에 있는 주석을 참고하고
 * @RequestParam 애노테이션을 사용해 요청 파라미터를 읽어오는 방법은
 * boardDetail() 메서드의 주석을 참고하기 바란다.
 **/
@RequestMapping(value="/writeProcess", method=RequestMethod.POST)
public String insertBoard(Board board) {

    /* BoardService 클래스를 이용해
     * 폼에서 넘어온 게시 글 정보를 게시 글 테이블에 추가한다.
     **/
    boardService.insertBoard(board);

    /* 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";
```

```
}
```

2-5) 게시 글 수정 폼 요청처리

게시 글 수정하기는 게시 글을 수정할 수 있는 폼을 보여주는 요청과 게시 글 수정 폼에서 사용자가 수정한 게시 글을 저장하는 요청으로 나누어 처리해야 한다. 게시 글 수정 폼을 보여주는 요청은 이전에 작성한 게시 글을 폼에 출력하여 보여줘야 하므로 게시 글쓰기와는 다르게 DB에 저장된 게시 글 내용을 읽어와 폼에 출력하는 처리가 필요하다. 그리고 게시 글 수정 기능은 게시 글 상세보기에서 수정하기 버튼을 클릭해 수정 폼으로 이동하는 방식으로 구현할 것이다. 이 때 이전의 비밀번호를 검사해 비밀번호가 일치해야 게시 글 수정 폼으로 이동할 수 있도록 구현할 것이므로 비밀번호를 체크하는 기능도 필요하다.
게시 글 수정 폼에 출력해야 할 데이터는 앞에서 구현한 게시 글 상세보기의 getBoard(int no) 메서드를 그대로 사용하면 된다. 그러므로 Mapper 설정 파일과 BoardDaoImpl 클래스에 추가할 기능은 게시 글 상세보기에서 게시 글 수정 폼 요청을 할 때 비밀번호를 체크하는 기능만 추가하면 된다.

▶ JavaScript

게시 글 상세보기 페이지에서 수정하기 버튼이 클릭되면 게시 글 수정 폼을 요청하는 자바스크립트 이벤트 처리 코드를 formcheck.js 파일의 $(function() { } 코드 안쪽에 작성한다.

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

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

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

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

▶ 게시 글 관련 SQL을 분리한 Mapper 에 추가

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

```
<!--
```

```
        게시판 테이블에서 no에 해당하는 게시 글의 비밀번호를 가져오는 맵핑 구문

        아래는 DAO 클래스의 isPassCheck(int no, String pass) 메서드에서
        사용하는 맵핑 구문으로 DAO에서 게시 글 번호인 no에 해당하는 게시 글의
        비밀번호를 조회할 때 selectOne() 메서드의 두 번째 인수로 기본형인 no를
        지정했기 때문에 parameterType은 생략할 수 있다.

        SQL문의 조건에 사용할 파라미터는 아래와 같이 #{} 로 감싸서 지정하면 된다.
    -->
    <select id="isPassCheck" resultType="String">
        SELECT
            pass
        FROM springbbs
        WHERE no = #{no}
    </select>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 게시 글 상세보기에서 게시 글 수정 폼 요청을 할 때 비밀번호를 체크하는 메서드를 다음과 같이 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
/* 게시 글 수정과 삭제 할 때 비밀번호 체크에서 호출되는 메서드
 * 게시 글 수정, 삭제 시 no에 해당하는 비밀번호를 DB에서 읽어와 반환하는 메서드
 **/
public String isPassCheck(int no, String pass) {

    // isPassCheck 맵핑 구문을 호출하면서 게시 글 번호인 no를 파라미터로 지정했다.
    return sqlSession.selectOne(NAME_SPACE + ".isPassCheck", no);
}
```

▶ BoardService 인터페이스 구현 클래스

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

- com.springstudy.bbs.service.BoardServiceImpl

```
/* 게시 글 수정과 삭제 할 때 비밀번호가 맞는지 체크하는 메서드
 *
 * - 게시 글의 비밀번호가 맞으면 : true를 반환
 * - 게시 글의 비밀번호가 맞지 않으면 : false를 반환
 **/
public boolean isPassCheck(int no, String pass) {
```

```
        boolean result = false;

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

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

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

▶ Controller 클래스

BoardController 클래스에 게시 글 수정 폼 요청을 처리하는 메서드를 추가한다.

- com.springstudy.bbs.controller.BoardController

```
    /* 게시 글 수정 폼 요청을 처리하는 메서드
     *
     * 아래는 "/update"로 들어오는 요청을 처리하는 메서드를 지정한 것이다.
     * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
     *
     * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
     * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
     * no와 pass도 지정했다. 이렇게 Controller 메서드의 파라미터에 필요한
     * 객체나 요청 파라미터 이름과 동일한 이름의 파라미터를 지정하면 스프링이 자동으로
     * 설정해 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.
     *
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
     * 대한 설명은 setBoardService() 메서드 바로 아래에 있는 주석을 참고하고
     * @RequestParam 애노테이션을 사용해 요청 파라미터를 읽어오는 방법은
     * boardDetail() 메서드의 주석을 참고하기 바란다.
     **/
    @RequestMapping(value="/update")
    public String updateBoard(Model model, HttpServletResponse response,
            PrintWriter out, int no, String pass) {

        // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
        boolean result = boardService.isPassCheck(no, pass);
```

```
        // 비밀번호가 맞지 않으면
        if(! result) {

            /* 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.
             **/
            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;
        }

        /* Service 클래스를 이용해 게시 글 수정 폼에 출력할
         * no에 해당하는 게시 글을 가져온다.
         **/
        Board board = boardService.getBoard(no);

        /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
         * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
         **/
        model.addAttribute("board", board);

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "updateForm";
    }
```

▶ 게시 글 수정 폼 View

- src/main/webapp/WEB-INF/views/updateForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- 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">게시 글 수정하기</h2>
            </div>
        </div>
        <form name="updateForm" action="updateProcess" id="updateForm"
            class="row g-3 border-primary" method="post">
            <input type="hidden" name="no" value="${board.no}">
            <div class="col-4 offset-md-2">
                <label for="writer" class="form-label">글쓴이</label>
                <input type="text" class="form-control" name="writer"  id="writer"
                    placeholder="작성자를 입력해 주세요" value="${ board.writer }">
            </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" 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"/>
                      <input type="button" value="목록보기"
                        onclick="location.href='boardList'" class="btn btn-primary"/>
            </div>
        </form>
    </div>
</div>
```

▶ JavaScript

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

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

```
    // 게시 글 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) 게시 글 수정 요청처리

사용자가 게시 글 수정 폼에서 수정한 게시 글을 수정하는 요청은 먼저 기존의 게시 글 비밀번호와 일치하는지 검사한 후 비밀번호가 일치할 때만 게시 글이 수정되도록 구현할 것이다. 그리고 게시 글 수정 요청은 수정된 게시 글을 DB에서 수정한 후 게시 글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다.

▶ 게시 글 관련 SQL을 분리한 Mapper 에 추가

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

```
<!--
    게시판 테이블에서 no에 해당하는 게시 글을 수정하는 맵핑 구문

    아래는 DAO 클래스의 updateBoard(Board board) 메서드에서
    사용하는 맵핑 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.

    SqlSessionTemplate의 update() 메서드의 반환 타입이 int 이므로
    resultType은 생략 가능하다.
```

```
-->
<update id="updateBoard" parameterType="Board">
    UPDATE springbbs
        SET title = #{title}, content = #{content},
            reg_date = SYSDATE()
    WHERE no = #{no}
</update>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 사용자가 수정한 게시 글을 Board 객체로 받아 DB에서 게시 글을 수정하는 메서드를 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
/* 게시 글 수정 요청 시 호출되는 메서드
 * 게시 글 수정 요청 시 수정된 내용을 Board 객체로 받아 DB에 수정하는 메서드
 **/
@Override
public void updateBoard(Board board) {

    // updateBoard 맵핑 구문을 호출하면서 Board 객체를 파라미터로 지정했다.
    sqlSession.update(NAME_SPACE + ".updateBoard", board);
}
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardDao를 이용해 게시 글을 수정하는 메서드를 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl

```
// BoardDao를 이용해 게시 글을 수정하는 메서드
@Override
public void updateBoard(Board board) {
    boardDao.updateBoard(board);
}
```

▶ Controller 클래스

BoardController 클래스에 게시 글 수정 요청을 처리하는 메서드를 추가한다.

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 수정 폼에서 들어오는 게시 글 수정 요청을 처리하는 메서드
 *
 * @RequestMapping의 ()에 value="/updateProcess", method=RequestMethod.POST를
 * 지정해 "/updateProcess"로 들어오는 POST 방식의 요청을 처리하도록 설정한 것이다.
```

```
 *
 * @RequestMapping 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.
 *
 * @RequestMapping 애노테이션이 적용된 컨트롤러 메서드에 커맨드 객체를
 * 파라미터로 지정하면 커맨드 객체의 프로퍼티와 동일한 이름을 가진 요청
 * 파라미터의 데이터를 스프링이 자동으로 설정해 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 정수형으로 형 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 **/
@RequestMapping(value="updateProcess", method=RequestMethod.POST)
public String updateBoard(HttpServletResponse response,
        PrintWriter out, Board board) {

    // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
    boolean result = boardService.isPassCheck(board.getNo(), board.getPass());

    // 비밀번호가 맞지 않으면
    if(! result) {

        /* 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.
         **/
        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;
    }
```

```
        // BoardService 클래스를 이용해 게시판 테이블에서 게시 글을 수정한다.
        boardService.updateBoard(board);

        /* 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";
    }
```

2-7) 게시 글 삭제하기

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

▶ JavaScript

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

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

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

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

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

▶ 게시 글 관련 SQL을 분리한 Mapper 에 추가

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

```
<!--
    게시판 테이블에서 no에 해당하는 게시 글을 삭제하는 맵핑 구문

    아래는 DAO 클래스의 deleteBoard(int no) 메서드에서 사용하는
    맵핑 구문으로 parameterType은 int 형이므로 생략했다.

    SqlSessionTemplate의 delete() 메서드의 반환 타입이 int 이므로
    resultType은 생략 가능하다.
-->
<delete id="deleteBoard">
    DELETE FROM springbbs
    WHERE no = #{no}
</delete>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 사용자가 삭제를 요청한 게시 글 번호를 받아 그 번호에 해당하는 게시 글을 테이블에서 삭제하는 메서드를 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
/* 게시 글 삭제 요청 시 호출되는 메서드
 * no에 해당 하는 게시 글을 DB에서 삭제하는 메서드
 **/
@Override
public void deleteBoard(int no) {

    // deleteBoard 맵핑 구문을 호출하면서 no를 파라미터로 지정했다.
    sqlSession.delete(NAME_SPACE + ".deleteBoard", no);
}
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardDao를 이용해 게시 글을 삭제하는 메서드를 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl

```
// BoardDao를 이용해 no에 해당하는 게시 글을 삭제하는 메서드
@Override
public void deleteBoard(int no) {
    boardDao.deleteBoard(no);
}
```

▶ Controller 클래스

BoardController 클래스에 게시 글 삭제 요청을 처리하는 메서드를 추가한다.

- com.springstudy.bbs.controller.BoardController

```
    /* 게시 글 상세보기에서 들어오는 게시 글 삭제 요청을 처리하는 메서드
     *
     * 아래는 "/delete", "/deleteBoard"로 들어오는 요청을 처리하는 메서드를 지정한 것이다.
     * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
     *
     * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
     * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
     * no와 pass도 지정했다. 이렇게 Controller 메서드의 파라미터에 필요한
     * 객체나 요청 파라미터 이름과 동일한 이름의 파라미터를 지정하면 스프링이 자동으로
     * 설정해 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.
     *
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
     * 대한 설명은 setBoardService() 메서드 바로 아래에 있는 주석을 참고하고
     * @RequestParam 애노테이션을 사용해 요청 파라미터를 읽어오는 방법은
     * boardDetail() 메서드의 주석을 참고하기 바란다.
     **/
    @RequestMapping({"/delete", "deleteBoard"})
    public String deleteBoard(HttpServletResponse response,
            PrintWriter out, int no, String pass) {

        // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
        boolean result = boardService.isPassCheck(no, pass);

        // 비밀번호가 맞지 않으면
        if(! result) {

            /* 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.
             **/
            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;
        }

        // BoardService 클래스를 이용해 게시판 테이블에서 게시 글을 수정한다.
        boardService.deleteBoard(no);

        /* 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. 게시 글 리스트 페이징 처리 구현

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

1) 프로젝트 생성 및 환경설정

이번 예제는 앞에서 구현한 springstudy-bbs01 프로젝트를 프로젝트 이름만 springclass-bbs02로 변경하고 페이징 처리에 필요한 부분만 수정할 것이다. 그렇기 때문에 프로젝트 생성과 설정 그리고 의존 라이브러리 설정 등은 앞의 예제에서 설명한 주석을 참고 하길 바란다.
이번 예제는 앞에서 구현한 게시판에 페이징 기능을 추가하는 것이므로 설정파일과 소스가 거의 동일하다. 그러므로 꼭 필요한 부분만 교안에 작성하고 추가되거나 수정되는 부분은 붉은색 볼드체로 표시할 것이다.

- **실습용 프로젝트 : springclass-bbs02**
- **완성 프로젝트 : springstudy-bbs02**

2) 게시 글 리스트 페이징 처리 구현

2-1) 게시 글 리스트 요청처리

이번 예제는 앞 예제에서 게시 글 리스트에 페이징 처리를 하는 것이므로 Domain 클래스는 앞 예제와 동일하기 때문에 교안에는 작성하지 않았다.

▶ DAO(Data Access Object) 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에 한 페이지에 표시할 게시 글 리스트를 반환하는 boardList() 추상 메서드를 아래와 같이 수정하고 페이징 처리를 위해 DB로부터 게시 글의 수를 카운트해서 가져오는 getBoardCount() 추상 메서드를 추가한다.

- com.springstudy.bbs.dao.BoardDao

```
/* 한 페이지에 보여 질 게시 글 리스트 요청 시 호출되는 메소드
 * 현재 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환 하는 메소드
 **/
public abstract List<Board> boardList(int startRow, int num);
```

```
/* 전체 게시 글 수를 계산하기 위해 호출되는 메서드 - paging 처리에 사용
 * DB 테이블에 등록된 모든 게시 글의 수를 반환 하는 메서드
 **/
public abstract int getBoardCount();
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 BoardDao 인터페이스에서 수정한 boardList(int startRow, int num) 메서드를 수정하고 페이징 처리를 위해 DB로부터 게시 글의 수를 가져오는 getBoardCount() 메서드를 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
/* 한 페이지에 보여 질 게시 글 리스트 요청 시 호출되는 메소드
 * 현재 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환 하는 메소드
 **/
@Override
public List<Board> boardList(int startRow, int num) {

    // SQL 파라미터가 여러 개일 경우 Map을 이용하여 지정한다.
    Map<String, Integer> params = new HashMap<String, Integer>();
    params.put("startRow", startRow);
    params.put("num", num);

    /* BoardMapper.xml에서 맵핑 구문을 작성하고 아래와 같이 SqlSession
     * 객체의 메서드를 호출하면서 맵핑 설정을 지정하게 되면 이 메서드 안에서
     * PreparedStatement 객체를 생성하고 PreparedStatement 객체에
     * 필요한 파라미터가 설정된다.
     *
     * SqlSessionTemplate 객체의 select(), selectOne(), selectList()
     * 메서드를 호출하면 PreparedStatement 객체의 executeQuery() 메서드를
     * 실행하고 쿼리를 발행한 결과인 ResultSet 객체에서 데이터를 읽어와 모델
     * 클래스인 Board 객체를 생성하고 이 객체에 값을 설정하게 된다.
     *
     * 아래와 같이 SqlSessionTemplate의 메서드가 호출되면
     * repository/mappers/BoardMapper.xml 맵퍼 파일에서
     * mapper 태그의 namespace 속성에 지정한
     * com.springstudy.bbs.mapper.BoardMapper인 맵퍼가 선택되고
     * 그 하부에 <select> 태그의 id 속성에 지정한 boardList인 맵핑 구문이
     * 선택된다. 그리고 MyBatis 내부에서 JDBC 코드로 변환되어 실행된다.
     *
     * 매핑 구문에 resultType 속성에 Board를 지정했기 때문에 요청한
     * 페이지에 해당하는 게시 글 리스트가 담긴 List<Board> 객체가
     * 반환된다. Board 테이블에 게시 글 정보가 하나도 없으면 null이 반환 된다.
     *
     * 만약 SQL 파라미터를 지정해야 한다면 두 번째 인수에 필요한 파라미터를
```

```
         * 지정하면 되는데 파라미터가 여러 개일 경우 Map 객체에 담아 두 번째
         * 인수로 지정하면 된다.
         **/
        return sqlSession.selectList(NAME_SPACE + ".boardList", params);
    }

    /* 전체 게시 글 수를 계산하기 위해 호출되는 메서드 - paging 처리에 사용
     * DB 테이블에 등록된 모든 게시 글의 수를 반환 하는 메서드
     **/
    @Override
    public int getBoardCount() {
        return sqlSession.selectOne(NAME_SPACE + ".getBoardCount");
    }
```

▶ SQL을 분리한 Mapper 에 추가

게시판 관련 맵퍼에 한 페이지에 해당하는 게시 글을 게시판 테이블로부터 가져오는 boardList 맵핑 구문을 수정하고 페이징 처리를 위해 DB로부터 게시 글의 수를 가져오는 getBoardCount 맵핑 구문을 추가한다.

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

```
    <!--
        한 페이지에 해당하는 게시 글 리스트를 가져오는 맵핑 구문

        테이블의 컬럼명은 일반적으로 언더스코어 표기법("_")을 사용하는 경우가
        많고 클래스의 인스턴스 맴버는 카멜표기법을 사용한다.
        테이블의 컬럼명과 모델 클래스의 프로퍼티 이름이 다른 경우 아래와 같이
        SELECT 쿼리에 별칭을 사용해 모델 클래스의 프로퍼티 이름과 동일하게
        맞춰야 한다. 그렇지 않으면 오류는 발생하지 않지만 데이터를 읽어 올
        수 없다.

        SELECT read_count AS readCount FROM springbbs

        아래는 테이블에 언더스코어 표기법으로 작성된 컬럼이 있기 때문에 이 컬럼은
        별칭을 지정할 때 자바 도메인 객체의 프로퍼티와 동일하게 지정하였다.
        이렇게 resultType에 Board를 지정하고 DAO 클래스에서 SqlSession의
        selectList() 메서드를 호출하면 List<Board> 객체로 반환된다.

        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.

        DAO에서 현재 페이지에 해당하는 게시 글 리스트를 조회할 startRow와 num을
        HashMap에 저장해 넘겨줬기 때문에 parameterType="hashmap"
```

```
    으로 설정하고  SQL 쿼리에서 LIMIT 명령 다음에 HashMap의 키로 지정한
    #{startRow}와 #{num}을 지정하였다.
-->
<select id="boardList" parameterType="hashmap" 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 게시 글 수를 반환하는 맵핑 구문

    SQL 쿼리 결과가 정수 이므로 resultType을 int형으로 지정했다.
    아래에서 resultType을 생략하면 예외가 발생한다.
-->
<select id="getBoardCount" resultType="int">
    SELECT
        COUNT(no)
    FROM springbbs
</select>
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 한 페이지에 표시할 게시 글 리스트와 페이징 처리를 위해 필요한 데이터를 계산해 Map으로 반환하는 boardList() 메서드를 수정한다.

- com.springstudy.bbs.service.BoardService

```
/* BoardDao를 이용해 게시판 테이블에서 한 페이지에 해당하는 게시 글
 * 리스트와 페이징 처리에 필요한 데이터를 Map 객체로 반환 하는 메소드
 **/
public abstract Map<String, Object> boardList(int pageNum);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardService 인터페이스에서 수정한 boardList(int pageNum) 메서

드를 아래와 같이 수정한다. 또한 페이징 처리와 관련해서 한 페이지에 보여 줄 게시 글의 수와 페이지 이동을 위해 한 페이지에 표시할 페이지 링크(페이지 그룹)를 상수로 정의 했다.

- com.springstudy.bbs.service.BoardServiceImpl

```
// 이 클래스가 서비스 계층(비즈니스 로직)의 컴포넌트(Bean) 임을 선언하고 있다.
@Service
public class BoardServiceImpl implements BoardService {

    // 한 페이지에 보여 줄 게시 글의 수를 상수로 선언
    private static final int PAGE_SIZE = 10;

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

    ... 중략 ...

    /* BoardDao를 이용해 게시판 테이블에서 한 페이지에 해당하는 게시 글
     * 리스트와 페이징 처리에 필요한 데이터를 Map 객체로 반환 하는 메소드
     **/
    @Override
    public Map<String, Object> boardList(int pageNum) {

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

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

        // BoardDao를 이용해 전체 게시 글 수를 얻어온다.
        int listCount = boardDao.getBoardCount();


        /* 현재 페이지에 해당하는 게시 글 리스트를 BoardDao를 이용해 DB에서 읽어온다.
         * Oracle에서는 페이징 처리를 위해 의사컬럼인 ROWNUM을 사용했지만
         * MySQL은 검색된 데이터에서 특정 행 번호부터 지정한 개수 만큼 행을 읽어오는
```

```
 * LIMIT 명령을 제공하고 있다. LIMIT의 첫 번째 매개변수에 가져올 데이터의
 * 시작 행을 지정하고 두 번째 매개변수에 가져올 데이터의 개수를 지정하면 된다.
 **/
List<Board> boardList = boardDao.boardList(startRow, PAGE_SIZE);

/* 페이지 그룹 이동 처리를 위해 전체 페이지를 계산 한다.
 * [이전] 11 12 13...   또는   ... 8 9 10 [다음] 과 같은 페이징 처리
 * 전체 페이지 = 전체 게시 글 수 / 한 페이지에 표시할 게시 글 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
 * 페이지 그룹 별 시작 페이지 : 1, 11, 21, 31...
 * 첫 번째 페이지 그룹에서 페이지 리스트는 1 ~ 10이 되므로 currentPage가
 * 1 ~ 10 사이에 있으면 startPage는 1이 되고 11 ~ 20 사이면 11이 된다.
 *
 * 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 연자자를
 * 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 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
         * 게시 글 리스트의 수, 한 페이지에 보여 줄 게시 글 리스트의 데이터를 Map에
         * 저장해 컨트롤러로 전달한다.
         **/
        Map<String, Object> modelMap = new HashMap<String, Object>();

        modelMap.put("boardList", 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;
    }

    ... 중략 ...

}
```

▶ Controller 클래스

BoardController 클래스에 게시 글 리스트 요청을 처리하는 메서드를 아래와 같이 수정한다.

- com.springstudy.bbs.controller.BoardController

```
    /* @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입
     *
     * 1. Controller 메서드의 파라미터 타입으로 지정할 수 있는 객체와 애노테이션
     * @RequestMapping 애노테이션이 적용된 컨트롤러 메서드의 파라미터에
     * 아래와 같은 객체와 애노테이션을 사용할 수 있도록 지원하고 있다.
     *
     * - HttpServletRequest, HttpServletResponse
     *   요청/응답을 처리하기 위한 서블릿 API
     *
     * - HttpSession
     *   HTTP 세션을 위한 서블릿 API
     *
     * - org.springframework.ui.Model, ModelMap, java.util.Map
     *   뷰에 모델 데이터를 전달하기 위한 모델 객체
     *
     * - 커맨드 객체(VO, DTO)
     *   요청 데이터를 저장할 객체
```

```
*
* - Errors, BindingResult
*   검증 결과를 저장할 객체로 커맨드 객체 바로 뒤에 위치 시켜야 한다.
*
* - @RequestParam
*   HTTP 요청 파라미터의 값을 메서드의 파라미터로 매핑하기 위한 애노테이션
*
* - @RequestHeader
*   HTTP 요청 헤더의 값을 파라미터로 받기 위한 애노테이션
*
* - @RequestCookie
*   Cookie 데이터를 파라미터로 받기 위한 애노테이션
*
* - @RequestVariable
*   RESTful API 방식의 파라미터를 받기 위한 경로 변수 설정 애노테이션
*
* - @RequestBody
*   요청 몸체의 데이터를 자바 객체로 변환하기 위한 애노테이션
*   String이나 JSON으로 넘어오는 요청 몸체의 데이터를 자바 객체로
*   변환하기 위한 사용하는 애노테이션 이다.
*
* - Writer, OutputStream
*   응답 데이터를 직접 작성할 때 메서드의 파라미터로 지정해 사용한다.
*
* 2. Controller 메서드의 반환 타입으로 지정할 수 있는 객체와 애노테이션
* - String
*   뷰 이름을 반환할 때 메서드의 반환 타입으로 지정
*
* - void
*   컨트롤러의 메서드에서 직접 응답 데이터를 작성할 경우 지정
*
* - ModelAndView
*   모델과 뷰 정보를 함께 반환해야 할 경우 지정
*   이전의 컨트롤는 스프링이 지원한는 Controller 인터페이스를
*   구현해야 했는데 이때 많이 사용하던 반환 타입이다.
*
* - 자바 객체
*   메서드에 @ResponseBody가 적용된 경우나 메서드에서 반환되는
*   객체를 JSON 또는 XML과 같은 양식으로 응답을 변환 할 경우에 사용한다.
**/

/* 게시 글 리스트 보기 요청을 처리하는 메서드
*
* @RequestMapping은 클래스 레벨과 메서드 레벨에 지정할 수 있다.
* @RequestMapping의 ()에 처리할 요청 URI만 지정할 때는 value 속성을
```

* 생략하고 처리할 요청 URI를 String 또는 String 배열을 지정할 수 있지만
* 다른 속성을 같이 지정할 경우 value 속성에 처리할 요청 URI를 지정해야 한다.
* 또한 method 속성을 지정해 컨트롤러가 처리할 HTTP 요청 방식을
* 지정할 수 있는데 아래는 "/boardList", "/list"로 들어오는 GET 방식의
* 요청을 이 메서드가 처리할 수 있도록 설정한 것이다.
* method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
*
* 요청을 처리한 결과를 뷰에 전달하기 위해 사용하는 것이 Model 객체이다.
* 컨트롤러는 요청을 처리한 결과 데이터를 모델에 담아 뷰로 전달하고 뷰는
* 모델로 부터 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 애노테이션이 적용된 메서드의 파라미터에 Model
* 을 지정하면 스프링이 이 메서드를 호출하면서 Model 타입의 객체를 넘겨준다.
* 우리는 Model을 받아 이 객체에 결과 데이터를 담기만 하면 뷰로 전달된다.
*
* 스프링은 클라이언트로부터 넘어 오는 요청 파라미터를 받을 수 있는 여러 가지
* 방법을 제공하고 있다. 아래와 같이 Controller 메서드에 요청 파라미터 이름과
* 동일한 이름의 메서드 파라미터를 지정하면 스프링으로부터 요청 파라미터를 넘겨
* 받을 수 있다. 만약 요청 파라미터와 메서드의 파라미터 이름이 다른 경우 아래와
* 같이 메서드의 파라미터 앞에 @RequestParam("요청 파라미터 이름")을
* 사용해 요청 파라미터의 이름을 지정하면 된다.
*
* @RequestMapping 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.
*
* @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터
* 이름과 요청 파라미터의 이름이 같은 경우 @RequestParam 애노테이션을
* 지정하지 않아도 스프링으로부터 요청 파라미터를 받을 수 있다.
*

```
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
     * 대한 설명은 boardList() 메서드 주석 위쪽에서 설명한 주석을 참고하기 바란다.
     *
     * 아래는 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.
     **/
    @RequestMapping(value= {"/boardList", "/list"}, method=RequestMethod.GET)
    public String boardList(Model model,
            @RequestParam(value="pageNum", required=false,
            defaultValue="1") int pageNum) {


        // Service 클래스를 이용해 게시 글 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);


        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "boardList";
    }
```

▶ 웹 템플릿 View

웹 템플릿의 index.jsp와 header.jsp, footer.jsp는 앞의 예제와 동일하기 때문에 교안에는 작성하지 않았다.

▶ 게시 글 리스트 View

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

- src/main/webapp/WEB-INF/views/boardList.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!-- 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">게시 글 리스트</h2>
            </div>
        </div>
        <form name="searchForm" id="searchForm" action="#"
            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 href="writeForm" class="btn btn-outline-success">글쓰기</a>
            </div>
        </div>
        <div class="row my-3">
            <div class="col">
                <table class="table table-hover">
                    <thead>
                        <tr class="table-dark">
                            <th>NO</th>
                            <th>제목</th>
                            <th>작성자</th>
                            <th>작성일</th>
                            <th>조회수</th>
                        </tr>
```

```
                    </thead>
                    <tbody class="text-secondary">
                        <!-- 게시 글이 있는 경우 -->
                    <c:if test="${ not empty boardList }">
                        <c:forEach var="b" items="${boardList}" varStatus="status">
                        <tr>
                            <td>${ b.no }</td>
                            <td><a href="boardDetail?no=${b.no}&pageNum=${currentPage}"
                                class="text-decoration-none link-secondary">${ b.title }</a></td>
                            <td>${ b.writer }</td>
                            <td>${ b.regDate }</td>
                            <td>${ b.readCount }</td>
                        </tr>
                        </c:forEach>
                    </c:if>
                    <!-- 게시 글이 없는 경우 -->
                    <c:if test="${ empty boardList }">
                        <tr>
                            <td colspan="5" class="text-center">게시 글이 존재하지 않습니다.</td>
                        </tr>
                    </c:if>
                    </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 페이지에
                     * pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
                     * startPage로 이동할 수 있다.
                     **/
                     --%>
                    <c:if test="${ startPage > pageGroup }">
                        <li class="page-item">
                            <a class="page-link" href="boardList?pageNum=${ startPage - pageGroup }">Pre</a>
                        </li>
                    </c:if>
                    <%--
```

```
                    /* 현재 페이지 그룹의 startPage 부터 endPage 만큼 반복하면서
                    * 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
                    * 현재 페이지는 링크를 설정하지 않는다.
                    **/
                    --%>
                    <c:forEach var="i" begin="${startPage}" end="${endPage}">
                        <c:if test="${i == currentPage }">
                            <li class="page-item active" aria-current="page">
                                <span class="page-link">${i}</span>
                            </li>
                        </c:if>
                        <c:if test="${i != currentPage }">
                            <li class="page-item"><a class="page-link"
href="boardList?pageNum=${ i }">${i}</a></li>
                        </c:if>
                    </c:forEach>
                    <%--
                    /* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
                     * 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
                     * pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
                     * startPage로 이동할 수 있다.
                     **/
                     --%>
                    <c:if test="${ endPage < pageCount }">
                        <li class="page-item">
                            <a class="page-link" href="boardList?pageNum=${ startPage +
pageGroup }">Next</a>
                        </li>
                    </c:if>
                </ul>
            </nav>
        </div>
    </div>
</div>
```

2-2) 게시 글 상세보기 요청처리

게시 글 리스트의 페이징 기능이 추가되면 페이징 처리와 관련해 몇 가지 더 추가적인 작업이 더 필요하다. 예를 들면 게시 글 상세보기에서 게시 글 리스트로 이동할 경우 사용자가 바로 이전에 있었던 게시 글 리스트로 이동시켜야 사용자 편의성을 제공할 수 있을 것이다. 또한 게시 글 수정 폼에서 게시 글 리스트로 이동할 경우 바로 이전에 있었던 게시 글 리스트로 이동시켜야할 필요도 있다. 그리고 게시 글 수정하기가 완료되었을 때 이전에 있었던 게시 글 리스트로 리다이렉트 시켜야 할 필요도 있으며 게시 글이 삭제되었을 때도 이전에 있었던 게시 글 리스트로 리다이렉트 시켜야

할 필요도 있다. 이를 위해 요청을 받는 Controller에서 pageNum을 받아 모델에 담고 각각의 뷰 페이지에서 게시 글 리스트로 이동하는 링크에 pageNum에 해당하는 요청 파라미터를 추가해 줘야 한다.
이번 예제에는 이전의 프로젝트에서 게시판을 구현할 때 게시 글 상세보기에서 처리해야 하는 게시 글 읽은 횟수를 증가하는 기능을 구현하지 않았다. 이번 프로젝트에서는 게시 글 상세보기 요청에서 게시 글 읽을 횟수를 증가하는 기능도 추가할 것이다. 이 기능을 추가하기 위해 몇 가지 추가적인 작업이 필요하다.
Service 계층에서 게시 글 하나의 정보를 처리하는 getBoard(int no)는 게시 글 상세보기와 게시 글 수정 폼 요청 모두에서 호출된다. 그렇기 때문에 Controller에서 Service 계층의 getBoard(int no) 메서드를 호출 할 때 게시 글 상세보기인지 아니면 게시 글 수정 폼 요청인지를 구분해서 호출해야 할 필요가 있다.

▶ DAO(Data Access Object) 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에서 기존에 게시 글 하나의 정보를 읽어와 반환하는 getBoard() 메서드는 앞의 예제와 동일하기 때문에 생략했다. 이번에 새롭게 추가되는 게시 글 읽은 횟수를 증가시키는 메서드만 아래와 같이 추가하면 된다.

- com.springstudy.bbs.dao.BoardDao

```
/* 게시 글을 조회한 횟수를 증가시키는 메서드
 * 게시 글 상세보기 요청 시 게시 글을 조회한 횟수를 DB에서 증가시켜주는 메서드
 **/
public abstract void incrementReadCount(int no);
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 BoardDao 인터페이스에서 새로 추가한 수정한 incrementReadCount(int no) 메서드를 아래와 같이 오버라이딩 하면 된다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
/* 게시 글을 조회한 횟수를 증가시키는 메서드
 * 게시 글 상세보기 요청 시 게시 글을 조회한 횟수를 DB에서 증가시켜주는 메서드
 **/
@Override
public void incrementReadCount(int no) {

    // incrementReadCount 맵핑 구문을 호출하면서 no를 파라미터로 지정했다.
    sqlSession.update(NAME_SPACE + ".incrementReadCount", no);
}
```

▶ SQL을 분리한 Mapper 에 추가

게시판 관련 맵퍼에 게시 글 읽은 횟수를 1 증가시키는 incrementReadCount 맵핑 구문을 아래와 같이 추가한다. **no에 해당하는 게시 글을 DB에서 읽어오는 매핑 구문인 getBoard는 앞의 예제와 동일하기 때문에 교안에서 생략했다.**

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

```
<!--
    게시판 테이블에서 no에 해당하는 게시 글의 읽은 횟수를 증가시키는 맵핑 구문

    아래는 DAO 클래스의 deleteBoard(int no) 메서드에서 사용하는
    맵핑 구문으로 parameterType은 int 형이므로 생략했다.

    SqlSessionTemplate의 update() 메서드의 반환 타입이 int 이므로
    resultType은 생략 가능하다.
-->
<update id="incrementReadCount">
    UPDATE springbbs
        SET read_count = read_count + 1
    WHERE no = #{no}
</update>
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 게시 글 읽은 횟수를 증가시키고 게시 글 하나의 정보를 반환하는 getBoard(int no, boolean isCount) 메서드를 아래와 같이 수정한다.

- com.springstudy.bbs.service.BoardService

```
/* BoardDao를 이용해 게시판 테이블에서
 * no에 해당하는 게시 글 을 읽어와 반환하는 메서드
 * isCount == true 면 게시 상세보기, false 면 게시 글 수정 폼 요청임
 **/
public abstract Board getBoard(int no, boolean isCount);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardService 인터페이스에서 수정한 getBoard(int no, boolean isCount) 메서드를 아래와 같이 수정한다.

- com.springstudy.bbs.service.BoardServiceImpl

```
/* BoardDao를 이용해 게시판 테이블에서
 * no에 해당하는 게시 글 을 읽어와 반환하는 메서드
 * isCount == true 면 게시 상세보기, false 면 게시 글 수정 폼 요청임
 **/
@Override
public Board getBoard(int no, boolean isCount) {

    // 게시 글 상세보기 요청만 게시 글 읽은 횟수를 증가시킨다.
```

```
        if(isCount) {
            boardDao.incrementReadCount(no);
        }
        return boardDao.getBoard(no);
    }
```

▶ Controller 클래스

- com.springstudy.bbs.controller.BoardController

```
    /* 게시 글 상세보기 요청을 처리하는 메서드
     *
     * 아래 @RequestMapping에서 method를 생략했기 때문에 "/boardDetail"로
     * 들어오는 GET 방식과 POST 방식의 요청 모두를 처리할 수 있다.
     *
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
     * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
     * 참고하기 바란다.
     *
     * 아래는 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.
     **/
    @RequestMapping("/boardDetail")
    public String boardDetail(Model model, int no,
            @RequestParam(value="pageNum", required=false, defaultValue="1")
            int pageNum) {

        /* Service 클래스를 이용해 no에 해당하는 게시 글 하나의 정보를 읽어온다.
         * 두 번째 인수에 true를 지정해 게시 글 읽은 횟수를 1 증가 시킨다.
         **/
        Board board = boardService.getBoard(no, true);

        /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
         * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
         **/
        model.addAttribute("board", board);
        model.addAttribute("pageNum", pageNum);

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
```

```
        return "boardDetail";
    }
```

▶ 게시 글 상세보기 View

게시 글 상세보기에서 게시 글 수정 폼으로 넘어가는 기능과 게시 글 삭제 기능으로 연결되는 기능을 제공하고 있기 때문에 각각 연결되는 요청에 pageNum을 요청 파라미터로 보내야 한다. 왜냐하면 게시 글 수정 폼에서 게시 글 리스트로 넘어갈 때 사용자가 이전에 있었던 페이지로 이동할 수 있도록 해야 하고 게시 글이 수정되거나 삭제되었을 때도 이전에 있었던 페이지로 이동할 수 있도록 서비스를 제공하기 위해서 필요하다.

- src/main/webapp/WEB-INF/views/boardDetail.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!-- content -->
<div class="row my-5" id="global-content">
    <div class="col">
        <form name="checkForm" id="checkForm">
            <input type="hidden" name="no" id="no" value="${ board.no }"/>
            <input type="hidden" name="pass" id="rPass" />
            <input type="hidden" name="pageNum" value="${ pageNum }" />
        </form>
        <div class="row text-center">
            <div class="col">
                <h2 class="fs-3 fw-bold">게시 글 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">${ board.title }</td>
                        </tr>
                        <tr>
                            <th>글쓴이</th>
                            <td>${ board.writer }</td>
                            <th>작성일</th>
                            <td><fmt:formatDate value="${ board.regDate }"
                                pattern="yyyy-MM-dd HH:mm:ss" /></td>
                        </tr>
                        <tr>
```

```
                        <th>비밀번호</th>
                        <td>
                            <div class="col-sm-8">
                                <input class="form-control" type="password"
name="pass" id="pass">
                            </div>
                        </td>
                        <th>조회수</th>
                        <td>${ board.readCount }</td>
                    </tr>
                    <tr>
                        <th>파    일</th>
                        <td colspan="3">
                        <c:if test="${ empty board.file1 }">
                            첨부파일 없음
                        </c:if>
                        <c:if test="${ not empty board.file1 }">
                            <a href="upload/${ board.file1 }">파일 다운로드</a>
                        </c:if>
                        </td>
                    </tr>
                    <tr>
                        <td colspan="4">
                            <pre>${ board.content }</pre>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
    <div class="row my-3">
        <div class="col text-center">
            <input class="btn btn-warning" type="button" id="detailUpdate" value="수정
하기"/>
              <input class="btn btn-danger"  type="button" id="detailDelete"
value="삭제하기" />
              <input class="btn btn-primary" type="button" value="목록보기"
                onclick="location.href='boardList?pageNum=${pageNum}'"/>

        </div>
    </div>
</div>
</div>
```

2-3) 게시 글쓰기 요청처리

앞에서 게시 글쓰기 폼 요청은 모델 데이터가 필요 없기 때문에 Controller에서 이 요청에 대한 처리를 하지 않고 Spring MVC 설정 파일인 "WEB-INF/spring/appServlet/servlet-context.xml" 파일에 뷰 전용 컨트롤러를 설정하였다. 이번 페이징 처리에서도 게시 글쓰기 폼에서 게시 글 리스트로 이동할 때는 게시 글 리스트의 첫 페이지로 이동하도록 할 것이다. 또한 사용자가 게시 글쓰기 폼에서 작성한 게시 글을 저장하는 요청은 게시 글을 DB에 저장한 후 게시 글 리스트로 리다이렉트 될 때 사용자가 있었던 페이지로 보내는 것이 아니라 새로운 게시 글이 등록된 것을 확인할 수 있는 첫 페이지로 리다이렉트 하면 된다. 일반적으로 게시판에서 게시 글 리스트는 항상 최신 글을 맨 위에 보여주도록 설계되기 때문에 사용자가 자신이 작성한 신규 게시 글을 확인하려면 첫 페이지로 이동해야 하므로 앞에서와 마찬가지로 게시 글쓰기가 완료되면 게시 글 리스트의 첫 페이지로 리다이렉트 시킬 것이다.

게시 글쓰기 기능은 페이징 처리가 추가되더라도 앞에서 구현한 Controller, Service 계층과 DAO 계층 클래스 그리고 맵핑 구문과 뷰 페이지가 동일하기 때문에 교안에서 생략했다.

2-4) 게시 글 수정 폼 요청처리

게시 글 수정하기는 게시 글을 수정할 수 있는 폼을 보여주는 요청과 게시 글 수정 폼에서 사용자가 수정한 게시 글을 저장하는 요청으로 나누어 처리해야 한다. 게시 글 수정 폼을 보여주는 요청은 이전에 작성한 게시 글을 폼에 출력하여 보여줘야 하므로 게시 글쓰기와는 다르게 DB에 저장된 게시 글 내용을 읽어와 폼에 출력하는 처리가 필요하다. 그리고 게시 글 수정 기능은 게시 글 상세보기에서 수정하기 버튼을 클릭해 수정 폼으로 이동하는 방식으로 구현할 것이다. 이 때 이전의 비밀번호를 검사해 비밀번호가 일치해야 게시 글 수정 폼으로 이동할 수 있도록 구현할 것이므로 비밀번호를 체크하는 기능도 필요하다.

이번 예제는 페이징 처리를 구현하는 것이므로 게시 글 수정 폼에서 게시 글 리스트로 돌아갈 때 사용자가 이전에 있었던 페이지로 이동시키기 위해서 pageNum을 Controller 클래스에서 받아 모델에 담아 게시 글 수정 폼에서 사용할 수 있도록 구현해야 하며 게시 글 수정 폼에서 게시 글을 수정한 후 수정 요청을 할 때도 pageNum을 요청 파라미터에 추가해야 한다.

게시 수정 폼 요청은 앞에서 구현한 Service 계층과 DAO 계층 클래스 그리고 맵핑 구문이 동일하기 때문에 교안에서 생략되었고 Controll와 뷰 페이지만 수정하면 된다.

▶ Controller 클래스

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 수정 폼 요청을 처리하는 메서드
 *
 * 아래는 "/update"로 들어오는 요청을 처리하는 메서드를 지정한 것이다.
 * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
 *
 * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
 * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
 * no와 pass도 지정했다. 이렇게 Controller 메서드의 파라미터에 필요한
 * 객체나 요청 파라미터 이름과 동일한 이름의 파라미터를 지정하면 스프링이 자동으로
```

```
 * 설정해 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.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 **/
@RequestMapping(value="/update")
public String updateBoard(Model model, HttpServletResponse response,
        PrintWriter out, int no, String pass,
        @RequestParam(value="pageNum", required=false, defaultValue="1")
        int pageNum) {

    // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
    boolean result = boardService.isPassCheck(no, pass);

    // 비밀번호가 맞지 않으면
    if(! result) {

        /* 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.
         **/
        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;
    }

    /* Service 클래스를 이용해 no에 해당하는 게시 글 하나의 정보를 읽어온다.
     * 두 번째 인수로 false를 지정해 게시 글 읽은 횟수를 증가시키지 않는다.
     **/
    Board board = boardService.getBoard(no, false);

    /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
```

```
         * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
         **/
        model.addAttribute("board", board);
        model.addAttribute("pageNum", pageNum);

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "updateForm";
    }
```

▶ 게시 글 수정 폼 View

게시 글 수정 폼 페이지에서도 이 페이지로 들어올 때 받은 페이지 정보를 게시 글 수정 요청을 하면서 서버로 보내줘야 서버에서 게시 글 수정이 완료된 후 사용자가 이전에 있었던 게시 글 리스트로 연결되도록 서비스 할 수 있다.

- src/main/webapp/WEB-INF/views/updateForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- 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">게시 글 수정하기</h2>
            </div>
        </div>
        <form name="updateForm" action="updateProcess" id="updateForm"
            class="row g-3 border-primary" method="post">
            <input type="hidden" name="no" value="${board.no}">
            <input type="hidden" name="pageNum" value="${ pageNum }" />
            <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" value="${ board.writer }">
            </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" 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"/>
                      <input type="button" value="목록보기"
                        onclick="location.href='boardList?pageNum=${pageNum}'"
class="btn btn-primary"/>
            </div>
        </form>
    </div>
</div>
```

2-5) 게시 글 수정 요청처리

사용자가 게시 글 수정 폼에서 수정한 게시 글을 수정하는 요청은 먼저 기존의 게시 글 비밀번호와 일치하는지 검사한 후 비밀번호가 일치할 때만 게시 글이 수정되도록 구현할 것이다. 그리고 게시 글 수정 요청은 수정된 게시 글을 DB에서 수정한 후 게시 글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다. 이번 예제는 페이징 처리를 구현하는 것이므로 게시 글 수정 요청을 처리한 후 리다이렉트할 때 pageNum을 파라미터로 추가해야 사용자가 이전에 있었던 게시 글 리스트 페이지로 이동시킬 수 있다.

게시 수정 요청은 앞에서 구현한 Service 계층과 DAO 계층 클래스 그리고 맵핑 구문이 동일하기 때문에 교안에서 생략되었고 Controll만 수정하면 된다.

▶ Controller 클래스

- com.springstudy.bbs.controller.BoardController

```
    /* 게시 글 수정 폼에서 들어오는 게시 글 수정 요청을 처리하는 메서드
     *
     * @RequestMapping의 ()에 value="/updateProcess", method=RequestMethod.POST를
     * 지정해 "/updateProcess"로 들어오는 POST 방식의 요청을 처리하도록 설정한 것이다.
     *
     * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
     * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
     * Board 객체를 지정했다. 또한 리다이렉트할 때 pageNum을 파라미터로 보내기
     * 위해서 RedirectAttributes 객체를 메서드의 파라미터로 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.
 *
 * @RequestMapping 애노테이션이 적용된 컨트롤러 메서드에 커맨드 객체를
 * 파라미터로 지정하면 커맨드 객체의 프로퍼티와 동일한 이름을 가진 요청
 * 파라미터의 데이터를 스프링이 자동으로 설정해 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 정수형으로 형 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 **/
@RequestMapping(value="/updateProcess", method=RequestMethod.POST)
public String updateBoard(HttpServletResponse response,
        PrintWriter out, Board board,
        RedirectAttributes reAttrs,
        @RequestParam(value="pageNum", required=false, defaultValue="1")
        int pageNum) {

    // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
    boolean result = boardService.isPassCheck(board.getNo(), board.getPass());

    // 비밀번호가 맞지 않으면
    if(! result) {

        /* 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.
         **/
        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;
        }

        // BoardService 클래스를 이용해 게시판 테이블에서 게시 글을 수정한다.
        boardService.updateBoard(board);

        /* 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 주소를 지정한다.
         *
         * 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.
         *
         * 아래와 같이 RedirectAttributs의 addAttribute() 메서드를 사용해
         * 지속적으로 사용할 파라미터를 지정하면 자동으로 주소 뒤에 파라미터로
         * 추가되며 addFlashAttribute() 메서드를 사용해 파라미터로 지정하면
         * 한 번만 사용할 수 있도록 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 URLEncoding을 코드로 구현해야 하지만
         * web.xml에서 스프링프레임워크가 지원하는 CharacterEncodingFilter를
         * 설정했기 때문에 Filter에 의해 UTF-8로 인코딩 되어 클라이언트로 응답된다.
         *
         * 아래는 게시 글 리스트로 Redirect 되면서 같이 보내야할 pageNum을
         * RedirectAttributs를 이용해 파라미터로 전달하는 예이다.
         **/
        reAttrs.addAttribute("pageNum", pageNum);
        //reAttrs.addFlashAttribute("test", "1회용 파라미터 받음 - test");
        return "redirect:boardList";
    }
```

2-6) 게시 글 삭제 요청처리

게시 글 삭제 기능은 게시 글 상세보기에서 삭제하기 버튼을 클릭하면 먼저 비밀번호를 검사해 비밀번호가 일치해야 게시 글을 삭제할 수 있도록 구현할 것이다. 그리고 게시 글 삭제 요청은 해당

게시 글을 DB에서 삭제한 후 게시 글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다. 이번 예제는 페이징 처리를 구현하는 것이므로 게시 글 삭제 요청을 처리한 후 리다이렉트할 때 pageNum을 파라미터로 추가해야 사용자가 이전에 있었던 게시 글 리스트 페이지로 이동시킬 수 있다.

게시 삭제 요청은 앞에서 구현한 Service 계층과 DAO 계층 클래스 그리고 맵핑 구문이 동일하기 때문에 교안에서 생략되었고 Controllr만 수정하면 된다.

▶ Controller 클래스

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 상세보기에서 들어오는 게시 글 삭제 요청을 처리하는 메서드
 *
 * 아래는 "/delete", "/deleteBoard"로 들어오는 요청을 처리하는 메서드를 지정한 것이다.
 * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
 *
 * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
 * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
 * no와 pass도 지정했다. 그리고 리다이렉트 할 때 pageNum을 파라미터로 보내기
 * 위해서 RedirectAttributes 객체를 메서드의 파라미터로 지정했다.
 * 이렇게 Controller 메서드의 파라미터에 필요한 객체나 요청 파라미터 이름과 동일한
 * 이름의 파라미터를 지정하면 스프링이 자동으로 설정해 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.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 **/
@RequestMapping({"/delete", "deleteBoard"})
public String deleteBoard(HttpServletResponse response,
        PrintWriter out, int no, String pass,
        RedirectAttributes reAttrs,
        @RequestParam(value="pageNum", required=false, defaultValue="1")
        int pageNum) {

    // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
    boolean result = boardService.isPassCheck(no, pass);

    // 비밀번호가 맞지 않으면
    if(! result) {
```

```
        /* 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.
         **/
        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;
    }

    // BoardService 클래스를 이용해 게시판 테이블에서 게시 글을 수정한다.
    boardService.deleteBoard(no);

    /* 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 주소를 지정한다.
     *
     * 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.
     *
     * 아래와 같이 RedirectAttributs의 addAttribute() 메서드를 사용해
     * 지속적으로 사용할 파라미터를 지정하면 자동으로 주소 뒤에 파라미터로
     * 추가되며 addFlashAttribute() 메서드를 사용해 파라미터로 지정하면
     * 한 번만 사용할 수 있도록 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 URLEncoding을 코드로 구현해야 하지만
     * web.xml에서 스프링프레임워크가 지원하는 CharacterEncodingFilter를
     * 설정했기 때문에 Filter에 의해 UTF-8로 인코딩 되어 클라이언트로 응답된다.
     *
```

```
        * 아래는 게시 글 리스트로 Redirect 되면서 같이 보내야할 pageNum을
        * RedirectAttributs를 이용해 파라미터로 전달하는 예이다.
        **/
        reAttrs.addAttribute("pageNum", pageNum);
        //reAttrs.addFlashAttribute("test", "1회용 파라미터 받음 - test");
        return "redirect:boardList";
    }
```

3. 게시 글 검색 기능 구현

앞에서 게시 글 리스트에 페이징 기능을 추가하는 방법에 대해 알아보았다.
이번에는 게시 글 리스트에서 사용자가 특정 검색어에 해당하는 게시 글을 검색하는 기능을 추가할 것이다. 게시판에 검색 기능을 구현하기 전에 이 기능을 제공하려면 애플리케이션이 어떻게 동작해야 하는지 먼저 생각해 보자.
게시 글 검색 기능이 포함된 게시판에서 게시 글 리스트는 검색이 아닌 경우와 특정 검색어로 검색한 경우에 따라서 게시 글 리스트를 출력하는 부분이 다르기 때문에 서버에서 요청을 처리하는 부분도 각각에 맞게 처리해야 한다. 그래서 현재 요청이 게시 글 리스트만 보여 달라는 것인지 아니면 검색 결과에 대한 리스트를 보여 달라는 것인지 먼저 판단하고 단순 게시 글 리스트 요청인지 아니면 검색 리스트 요청인지에 따라서 다르게 처리하는 코드를 작성해야 한다.

1) 프로젝트 생성 및 환경설정

이번 예제도 springstudy-bbs02 프로젝트를 복사해 springclass-bbs03 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 검색 기능을 추가할 것이다. 그렇기 때문에 프로젝트 생성과 설정 그리고 의존 라이브러리 설정 등에 대한 자세한 내용은 앞의 예제에서 설명한 주석을 참고 하길 바란다.
이번 예제도 앞에서 구현한 게시판에 기능을 추가하는 것이므로 설정파일과 소스가 거의 동일하기 때문에 추가되거나 수정되는 부분은 붉은색 볼드체로 표시할 것이다.

- **실습용 프로젝트 : springclass-bbs03**
- **완성 프로젝트 : springstudy-bbs03**

2) 게시 글 검색 기능 구현

2-1) 게시 글 리스트와 검색 리스트 요청처리

게시판에 검색 기능이 추가되면 게시 글 리스트의 페이징 처리와 검색 리스트의 페이징 처리를 구분해서 처리해야 한다. 왜냐하면 게시 글 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.
게시 글 리스트는 앞에서와 마찬가지로 전체 게시 글의 수를 이용해 페이지 수를 계산하고 페이징 처리에 필요한 데이터를 만들면 되겠지만 검색 리스트는 제목, 작성자, 내용으로 검색하는 경우에 따라서 사용자가 입력한 검색어가 포함된 게시 글의 수를 기준으로 페이징 처리에 필요한 데이터를 만들어야 한다. 위와 같이 동작하도록 구현하기 위해서 게시 글 리스트와 검색 리스트 처리를 별도의 메서드로 분리해서 처리할 수도 있지만 우리는 하나의 메서드에서 게시 글 리스트 요청인지, 검색 리스트 요청인지를 구분해서 각각에 맞게 구현할 것이다. 이렇게 하나의 메서드에서 두 가지 상황에 따라서 처리하기 위해서는 각 상황에 맞게 동적으로 변환되는 SQL 쿼리를 작성해야 한다. 그러므로 우리는 MyBatis 맵퍼 설정 파일에서 동적 쿼리를 구현하는 방법에 대해 알아볼 것이다.
이번 예제는 게시 글 페이징 처리가 구현된 앞의 예제에서 게시 글 검색 리스트 기능을 추가로 구

현하는 것이므로 Domain 클래스는 앞의 예제와 동일하므로 교안에는 작성하지 않았다.

▶ DAO(Data Access Object) 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에 게시 글 리스트와 검색 리스트에 따라서 한 페이지에 표시할 게시 글 리스트를 반환하는 boardList() 메서드와 각 상황에 맞는 페이징 처리를 위해 DB로부터 게시 글의 수를 가져오는 getBoardCount() 메서드를 다음과 같이 수정한다.

- com.springstudy.bbs.dao.BoardDao

```
/* 한 페이지에 보여 질 게시 글 리스트와 검색 리스트 요청 시 호출되는 메소드
 * 현재 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환 하는 메소드
 **/
public abstract List<Board> boardList(
        int startRow, int num, String type, String keyword);

/* 게시 글 수를 계산하기 위해 호출되는 메서드 - paging 처리에 사용
 * 게시 글 리스트와 검색 리스트에 대한 게시 글 수를 반환 하는 메서드
 **/
public abstract int getBoardCount(String type, String keyword);
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 BoardDao 인터페이스에서 수정한 boardList(int startRow, int num, String type, String keyword) 메서드를 수정하고 게시 글 리스트와 검색 리스트의 페이징 처리를 위해 각각의 상황에 따라서 DB로부터 게시 글의 수를 가져오는 getBoardCount(String type, String keyword) 메서드를 수정한다.

- com.springstudy.bbs.dao.BoardDaoImpl

```
/* 한 페이지에 보여 질 게시 글 리스트와 검색 리스트 요청 시 호출되는 메소드
 * 현재 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환 하는 메소드
 **/
@Override
public List<Board> boardList(
        int startRow, int num, String type, String keyword) {

    // SQL 파라미터가 여러 개일 경우 Map을 이용하여 지정한다.
    Map<String, Object> params = new HashMap<String, Object>();
    params.put("startRow", startRow);
    params.put("num", num);
    params.put("type", type);
    params.put("keyword", keyword);

    /* BoardMapper.xml에서 맵핑 구문을 작성하고 아래와 같이 SqlSession
     * 객체의 메서드를 호출하면서 맵핑 설정을 지정하게 되면 이 메서드 안에서
```

```
         * PreparedStatement 객체를 생성하고 PreparedStatement 객체에
         * 필요한 파라미터가 설정된다.
         *
         * SqlSessionTemplate 객체의 select(), selectOne(), selectList()
         * 메서드를 호출하면 PreparedStatement 객체의 executeQuery() 메서드를
         * 실행하고 쿼리를 발행한 결과인 ResultSet 객체에서 데이터를 읽어와 모델
         * 클래스인 Board 객체를 생성하고 이 객체에 값을 설정하게 된다.
         *
         * 아래와 같이 SqlSessionTemplate의 메서드가 호출되면
         * repository/mappers/BoardMapper.xml 맵퍼 파일에서
         * mapper 태그의 namespace 속성에 지정한
         * com.springstudy.bbs.mapper.BoardMapper인 맵퍼가 선택되고
         * 그 하부에 <select> 태그의 id 속성에 지정한 boardList인 맵핑 구문이
         * 선택된다. 그리고 MyBatis 내부에서 JDBC 코드로 변환되어 실행된다.
         *
         * 매핑 구문에 resultType 속성에 Board를 지정했기 때문에 요청한
         * 페이지에 해당하는 게시 글 리스트가 담긴 List<Board> 객체가
         * 반환된다. Board 테이블에 게시 글 정보가 하나도 없으면 null이 반환 된다.
         *
         * 만약 SQL 파라미터를 지정해야 한다면 두 번째 인수에 필요한 파라미터를
         * 지정하면 되는데 파라미터가 여러 개일 경우 Map 객체에 담아 두 번째
         * 인수로 지정하면 된다.
         **/
        return sqlSession.selectList(NAME_SPACE + ".boardList", params);
    }

    /* 게시 글 수를 계산하기 위해 호출되는 메서드 - paging 처리에 사용
     * 게시 글 리스트와 검색 리스트에 대한 게시 글 수를 반환 하는 메서드
     **/
    @Override
    public int getBoardCount(String type, String keyword) {

        // SQL 파라미터가 여러 개일 경우 Map을 이용해 지정하면 된다.
        Map<String, String> params = new HashMap<String, String>();
        params.put("type", type);
        params.put("keyword", keyword);

        return sqlSession.selectOne(NAME_SPACE + ".getBoardCount", params);
    }
```

▶ 게시 글 관련 SQL을 분리한 Mapper 에 추가

게시판 관련 맵퍼에 게시 글 리스트와 검색 리스트에 따라서 한 페이지에 표시할 게시 글을 게시판 테이블로부터 가져오는 boardList 맵핑 구문을 수정하고 각각의 상황에 따라서 페이징 처리를 위해 DB로부터 게시 글의 수를 가져오는 getBoardCount 맵핑 구문도 수정한다.

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

```
<!--
    한 페이지에 해당하는 게시 글 리스트, 검색리스트를 가져오는 맵핑 구문

    테이블의 컬럼명은 일반적으로 언더스코어 표기법("_")을 사용하는 경우가
    많고 클래스의 인스턴스 맴버는 카멜표기법을 사용한다.
    테이블의 컬럼명과 모델 클래스의 프로퍼티 이름이 다른 경우 아래와 같이
    SELECT 쿼리에 별칭을 사용해 모델 클래스의 프로퍼티 이름과 동일하게
    맞춰야 한다. 그렇지 않으면 오류는 발생하지 않지만 데이터를 읽어 올
    수 없다.

    SELECT read_count AS readCount FROM springbbs

    아래는 테이블에 언더스코어 표기법으로 작성된 컬럼이 있기 때문에 이 컬럼은
    별칭을 지정할 때 자바 도메인 객체의 프로퍼티와 동일하게 지정하였다.
    이렇게 resultType에 Board를 지정하고 DAO 클래스에서 SqlSession의
    selectList() 메서드를 호출하면 List<Board> 객체로 반환된다.

    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.

    DAO에서 현재 페이지에 해당하는 게시 글 리스트를 조회할 startRow와 num을
    HashMap에 저장해 넘겨줬기 때문에 parameterType="hashmap"
    으로 설정하고  SQL 쿼리에서 LIMIT 명령 다음에 HashMap의 키로 지정한
    #{startRow}와 #{num}을 지정하였다.

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

    게시 글 리스트와 검색 리스트 요청에 따라서 각각을 처리하기 위해 동적으로
    변환되는 SQL 쿼리를 작성해야 한다. 검색 리스트도 제목, 작성자, 내용을
    기준으로 검색어가 포함된 검색 리스트만 구성하기 위해서 동적으로 변환되는
    SQL이 필요하기 때문에 각 상황에 맞게 조건절이 동적으로 생성되도록 했다.
-->
<select id="boardList" resultType="Board" parameterType="hashmap">
    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.

    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 boardList()
    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 해야 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 게시 글 수와 검색 리스트에 대한 게시 글 수를 반환하는 맵핑 구문

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

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

    동적 SQL 참고 :
    http://www.mybatis.org/mybatis-3/ko/dynamic-sql.html

    DAO에서 검색에 필요한 type과 keyword를 HashMap에 저장해
    넘겨줬기 때문에 parameterType="hashmap"으로 설정하고
    SQL 쿼리의 파라미터를 HashMap의 키 이름으로 지정하였다.

    SQL 쿼리의 결과가 정수이기 때문에 resultType은 int형을 지정했다.
    아래에서 resultType을 생략하면 예외가 발생한다.

    아래에서 parameterType="hashmap"을 생략해도 map의 key 값으로
    지정한 이름과 #{}에 지정한 이름이 같은 파라미터에 데이터가 바인딩 된다.
-->
<select id="getBoardCount" parameterType="hashmap" 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 해야 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>
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 한 페이지에 표시할 게시 글 리스트와 검색 리스트 그리고 각각의 상황에 따라서 페이징 처리를 위해 필요한 데이터를 Map으로 반환하는 boardList() 메서드를 아래와 같이 수정한다.

- com.springstudy.bbs.service.BoardService

```
    /* BoardDao를 이용해 게시판 테이블에서 한 페이지에 해당하는 게시 글
     * 리스트와 페이징 처리에 필요한 데이터를 Map 객체로 반환 하는 메소드
     **/
    public abstract Map<String, Object> boardList(
            int pageNum, String type, String keyword);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardService 인터페이스에서 수정한 boardList(int pageNum, String type, String keyword) 메서드를 아래와 같이 수정한다.

- com.springstudy.bbs.service.BoardServiceImpl

```
/* BoardDao를 이용해 게시판 테이블에서 한 페이지에 해당하는 게시 글
 * 리스트와 페이징 처리에 필요한 데이터를 Map 객체로 반환 하는 메소드
 **/
@Override
public Map<String, Object> boardList(
        int pageNum, String type, String keyword) {

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

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

    /* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
     * 게시 글 리스트를 요청하는 것으로 간주하여 false 값을 갖게 한다.
     * Controller에서 type이나 keyword의 요청 파라미터가 없으면
     * 기본 값을 "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 게시 글의
     * 수를 반환하고, type이 존재하면 제목이나 내용 또는 작성자를 기준으로
     * 검색어가 포함된 게시 글 수를 반환한다.
     **/
    listCount = boardDao.getBoardCount(type, keyword);
    System.out.println("listCount : " + listCount + ", type : "
                + type + ", keyword : " + keyword);

    /* 현재 페이지에 해당하는 게시 글 리스트를 BoardDao를 이용해 DB에서 읽어온다.
     * Oracle에서는 페이징 처리를 위해 의사컬럼인 ROWNUM을 사용했지만
     * MySQL은 검색된 데이터에서 특정 행 번호부터 지정한 개수 만큼 행을 읽어오는
```

```
 * LIMIT 명령을 제공하고 있다. LIMIT의 첫 번째 매개변수에 가져올 데이터의
 * 시작 행을 지정하고 두 번째 매개변수에 가져올 데이터의 개수를 지정하면 된다.
 **/
List<Board> boardList = boardDao.boardList(
        startRow, PAGE_SIZE, type, keyword);

/* 페이지 그룹 이동 처리를 위해 전체 페이지를 계산 한다.
 * [이전] 11 12 13...   또는   ... 8 9 10 [다음] 과 같은 페이징 처리
 * 전체 페이지 = 전체 게시 글 수 / 한 페이지에 표시할 게시 글 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
 * 페이지 그룹 별 시작 페이지 : 1, 11, 21, 31...
 * 첫 번째 페이지 그룹에서 페이지 리스트는 1 ~ 10이 되므로 currentPage가
 * 1 ~ 10 사이에 있으면 startPage는 1이 되고 11 ~ 20 사이면 11이 된다.
 *
 * 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 연자자를
 * 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 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
         * 게시 글 리스트의 수, 한 페이지에 보여 줄 게시 글 리스트의 데이터를 Map에
         * 저장해 컨트롤러로 전달한다.
         **/
        Map<String, Object> modelMap = new HashMap<String, Object>();

        modelMap.put("boardList", 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 클래스에 일반 게시 글 리스트와 검색 리스트에 대한 요청을 처리하는 메서드를 아래와 같이 수정한다.

- com.springstudy.bbs.controller.BoardController

```
    /* 게시 글 리스트 보기 요청을 처리하는 메서드(게시 글 리스트, 검색 리스트)
     *
     * @RequestMapping은 클래스 레벨과 메서드 레벨에 지정할 수 있다.
     * @RequestMapping의 ()에 처리할 요청 URI만 지정할 때는 value 속성을
     * 생략하고 처리할 요청 URI를 String 또는 String 배열을 지정할 수 있지만
     * 다른 속성을 같이 지정할 경우 value 속성에 처리할 요청 URI를 지정해야 한다.
     * 또한 method 속성을 지정해 컨트롤러가 처리할 HTTP 요청 방식을
     * 지정할 수 있는데 아래는 "/boardList", "/list"로 들어오는 GET 방식의
     * 요청을 이 메서드가 처리할 수 있도록 설정한 것이다.
     * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
     *
     * 요청을 처리한 결과를 뷰에 전달하기 위해 사용하는 것이 Model 객체이다.
```

* 컨트롤러는 요청을 처리한 결과 데이터를 모델에 담아 뷰로 전달하고 뷰는
* 모델로 부터 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 애노테이션이 적용된 메서드의 파라미터에 Model
* 을 지정하면 스프링이 이 메서드를 호출하면서 Model 타입의 객체를 넘겨준다.
* 우리는 Model을 받아 이 객체에 결과 데이터를 담기만 하면 뷰로 전달된다.
*
* 스프링은 클라이언트로부터 넘어 오는 요청 파라미터를 받을 수 있는 여러 가지
* 방법을 제공하고 있다. 아래와 같이 Controller 메서드에 요청 파라미터 이름과
* 동일한 이름의 메서드 파라미터를 지정하면 스프링으로부터 요청 파라미터를 넘겨
* 받을 수 있다. 만약 요청 파라미터와 메서드의 파라미터 이름이 다른 경우 아래와
* 같이 메서드의 파라미터 앞에 @RequestParam("요청 파라미터 이름")을
* 사용해 요청 파라미터의 이름을 지정하면 된다.
*
* @RequestMapping 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.
*
* @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터
* 이름과 요청 파라미터의 이름이 같은 경우 @RequestParam 애노테이션을
* 지정하지 않아도 스프링으로부터 요청 파라미터를 받을 수 있다.
*
* @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
* 대한 설명은 boardList() 메서드 주석 위쪽에서 설명한 주석을 참고하기 바란다.
*
* 아래는 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.
     **/
    @RequestMapping(value= {"/boardList", "/list"})
    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 게시 글 리스트를 가져온다.
         * Service 클래스 안에서 일반 리스트 요청인지, 검색 요청인지를
         * 체크해서 각각에 맞는 필요한 데이터를 반환하도록 구현하면 된다.
         **/
        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);

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "boardList";
    }
```

▶ 웹 템플릿 View

웹 템플릿의 index.jsp와 header.jsp, footer.jsp는 앞의 예제와 동일하기 때문에 교안에는 작성하지 않았다.

▶ 게시 글 리스트 View

게시 글 리스트를 보여주는 페이지에서 일반 게시 글 리스트와 검색 리스트를 같이 처리하기 때문에 게시 글 상세보기로 넘어갈 때 일반 게시 글 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 알려주

기 위해서 keyword와 type이라는 파라미터를 같이 보내줘야 한다. 왜냐하면 사용자가 일반 검색 리스트에서 게시 글 상세페이지로 이동한 후 다시 게시 글 리스트로 돌아 올 경우 이전에 있었던 게시 글 리스트의 페이지로 돌아 올 수 있어야 하고 검색 리스트라면 이전에 있었던 검색 리스트로 돌아 올 수 있도록 서비스하기 위해 필요하다.

- src/main/webapp/WEB-INF/views/boardList.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!-- 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">게시 글 리스트</h2>
            </div>
        </div>
        <form name="searchForm" id="searchForm" action="#"
            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 아래를 화면에 표시 -->
        <c:if test="${ searchOption }">
            <div class="row my-3">
                <div class="col text-center">
                    "${ keyword  }" 검색 결과
                </div>
            </div>

            <%-- 검색 요청일 경우 일반 게시 글 리스트로 이동할 수 있도록 링크를 설정했다.
--%>
```

```
    <div class="row my-3">
        <div class="col-6">
            <a href="boardList" class="btn btn-outline-success">리스트</a>
        </div>
        <div class="col-6 text-end">
            <a href="writeForm" class="btn btn-outline-success">글쓰기</a>
        </div>
    </div>
</c:if>

<!-- 검색 요청이 아닐 경우 아래를 화면에 표시 -->
<c:if test="${ not searchOption }">
    <div class="row my-3">
        <div class="col text-end">
            <a href="writeForm" class="btn btn-outline-success">글쓰기</a>
        </div>
    </div>
</c:if>

<div class="row my-3">
    <div class="col">
        <table class="table table-hover">
            <thead>
                <tr class="table-dark">
                    <th>NO</th>
                    <th>제목</th>
                    <th>작성자</th>
                    <th>작성일</th>
                    <th>조회수</th>
                </tr>
            </thead>
            <tbody class="text-secondary">
            <%--
                검색 요청 이면서 검색된 리스트가 존재할 경우
                게시 글 상세보기로 링크를 적용할 때 type과 keyword
                파라미터를 적용해 링크를 설정한다.
            --%>
            <c:if test="${ searchOption and not empty boardList }">
                <c:forEach var="b" items="${boardList}" varStatus="status">
                <tr>
                    <td>${ b.no }</td>
                    <td>
                        <a href="boardDetail?no=${b.no}&pageNum=${currentPage}
                        &type=${ type }&keyword=${ keyword }"
                        class="text-decoration-none    link-secondary">${    b.title
```

```
}</a>
                        </td>
                        <td>${ b.writer }</td>
                        <td>${ b.regDate }</td>
                        <td>${ b.readCount }</td>
                    </tr>
                    </c:forEach>
                </c:if>

                <%--
                    일반 게시 글 리스트 요청 이면서 게시 글 리스트가 존재할 경우
                    게시 글 상세보기로 링크를 적용할 때 type과 keyword
                    파라미터는 필요 없다.
                --%>
                <c:if test="${ not searchOption and not empty boardList }">
                    <c:forEach var="b" items="${boardList}" varStatus="status">
                    <tr>
                        <td>${ b.no }</td>
                        <td><a href="boardDetail?no=${b.no}&pageNum=${currentPage}"
                            class="text-decoration-none    link-secondary">${    b.title
}</a></td>
                        <td>${ b.writer }</td>
                        <td>${ b.regDate }</td>
                        <td>${ b.readCount }</td>
                    </tr>
                    </c:forEach>
                </c:if>
                <%-- 검색 요청이면서 검색된 리스트가 존재하지 않을 경우 --%>
                <c:if test="${ searchOption and empty boardList }">
                    <tr>
                        <td colspan="5" class="text-center">
                            "${ keyword }"가 포함된 게시 글이 존재하지 않습니다.
                        </td>
                    </tr>
                </c:if>

                <%-- 일반 게시 글 리스트 요청이면서 게시 글 리스트가 존재하지 않을 경우
--%>
                <c:if test="${ not searchOption and empty boardList }">
                    <tr>
                        <td  colspan="5"  class="text-center">게시  글이  존재하지  않습니
다.</td>
                    </tr>
                </c:if>
                </tbody>
```

```
            </table>
        </div>
    </div>

    <!-- 검색 요청이면서 검색된 리스트가 존재할 경우 페이지네이션  -->
    <c:if test="${ searchOption and not empty boardList }">
        <div class="row">
            <div class="col">
                <nav aria-label="Page navigation">
                  <ul class="pagination justify-content-center">
                    <%--
                    /* 현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
                     * 이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
                     * pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
                     * startPage로 이동할 수 있다.
                     **/
                     --%>
                    <c:if test="${ startPage > pageGroup }">
                        <li class="page-item">
                            <a class="page-link" href="boardList?pageNum=${ startPage - pageGroup }
                              &type=${ type }&keyword=${ keyword }">Pre</a>
                        </li>
                    </c:if>

                    <%--
                    /* 현재 페이지 그룹의 startPage 부터 endPage 만큼 반복하면서
                     * 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
                     * 현재 페이지는 링크를 설정하지 않는다.
                     **/
                    --%>
                    <c:forEach var="i" begin="${startPage}" end="${endPage}">
                        <c:if test="${i == currentPage }">
                        <li class="page-item active" aria-current="page">
                            <span class="page-link">${i}</span>
                        </li>
                        </c:if>
                        <c:if test="${i != currentPage }">
                            <li class="page-item">
                                <a class="page-link" href="boardList?pageNum=${ i }&type=${ type }&keyword=${ keyword }">${i}</a>
                            </li>
                        </c:if>
                    </c:forEach>
```

```
                        <%--
                        /* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
                         * 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이
지에
                         * pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
                         * startPage로 이동할 수 있다.
                         **/
                         --%>
                        <c:if test="${ endPage < pageCount }">
                            <li class="page-item">
                                <a class="page-link" href="boardList?pageNum=${ startPage
+ pageGroup }
                                  &type=${ type }&keyword=${ keyword }">Next</a>
                            </li>
                        </c:if>
                    </ul>
                </nav>
            </div>
        </div>
    </c:if>

    <!-- 일반 게시글 요청이면서 검색된 리스트가 존재할 경우 페이지네이션  -->
    <c:if test="${ not searchOption and not empty boardList }">
        <div class="row">
            <div class="col">
                <nav aria-label="Page navigation">
                  <ul class="pagination justify-content-center">
                    <%--
                    /* 현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
                     * 이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이
지에
                     * pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
                     * startPage로 이동할 수 있다.
                     **/
                     --%>
                    <c:if test="${ startPage > pageGroup }">
                        <li class="page-item">
                            <a class="page-link" href="boardList?pageNum=${ startPage
- pageGroup }">Pre</a>
                        </li>
                    </c:if>
                    <%--
                    /* 현재 페이지 그룹의 startPage 부터 endPage 만큼 반복하면서
                     * 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
```

```
                        * 현재 페이지는 링크를 설정하지 않는다.
                        **/
                        --%>
                        <c:forEach var="i" begin="${startPage}" end="${endPage}">
                            <c:if test="${i == currentPage }">
                            <li class="page-item active" aria-current="page">
                                <span class="page-link">${i}</span>
                            </li>
                            </c:if>
                            <c:if test="${i != currentPage }">
                                <li class="page-item"><a class="page-link"
href="boardList?pageNum=${ i }">${i}</a></li>
                            </c:if>
                        </c:forEach>

                        <%--
                        /* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
                        * 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이
지에
                        * pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
                        * startPage로 이동할 수 있다.
                        **/
                        --%>
                        <c:if test="${ endPage < pageCount }">
                            <li class="page-item">
                                <a class="page-link" href="boardList?pageNum=${ startPage
+ pageGroup }">Next</a>
                            </li>
                        </c:if>
                    </ul>
                </nav>
            </div>
        </div>
    </c:if>
</div>
</div>
```

2-2) 게시 글 상세보기 요청처리

게시 글 검색 기능이 추가되면 일반 게시 글 리스트와 검색 리스트의 페이징 처리와 관련해 몇 가지 더 추가적인 작업이 더 필요하다. 예를 들면 게시 글 상세보기에서 게시 글 리스트로 이동할 경우 사용자가 바로 이전에 있었던 게시 글 리스트와 검색 리스트로 이동할 수 있도록 구분해서 처리해야한다. 또한 게시 글 수정 폼에서 게시 글 리스트나 검색 리스트로 이동할 경우 바로 이전에 있었던 게시 글 리스트나 검색 리스트로 구분해서 이동시켜야 사용자 편의성을 제공할 수 있다. 그리

고 게시 글 수정하기가 완료되었을 때도 이전에 있었던 게시 글 리스트나 검색 리스트 로 구분해서 리다이렉트 시켜야할 필요도 있으며 게시 글이 삭제되었을 때도 수정과 마찬가지로 이전에 있었던 게시 글 리스트나 검색 리스트로 리다이렉트 시켜야할 필요도 있다. 이를 위해 요청을 받는 Controller에서 pageNum과 type 그리고 keyword를 받아 모델에 담고 각각의 뷰 페이지에서 게시 글 리스트나 검색 리스트로 이동하는 링크에 pageNum과 type 그리고 keyword를 요청 파라미터에 추가해 줘야 한다.

게시 글 상세보기는 앞에서 구현한 Service 계층과 DAO 계층 클래스 그리고 맵핑 구문이 동일하기 때문에 교안에는 작성하지 않았고 요청을 처리하는 Controller와 뷰 페이지만 작성하였다.

▶ Controller 클래스

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 상세보기 요청을 처리하는 메서드
 *
 * 아래 @RequestMapping에서 method를 생략했기 때문에 "/boardDetail"로
 * 들어오는 GET 방식과 POST 방식의 요청 모두를 처리할 수 있다.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 *
 * 아래는 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.
 **/
@RequestMapping("/boardDetail")
public String boardDetail(Model model, int no,
        @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) throws Exception {

    /* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
     * 게시 글 리스트를 요청하는 것으로 간주하여 false 값을 갖게 한다.
     * Controller에서 type이나 keyword의 요청 파라미터가 없으면
     * 기본 값을 "null"로 지정했기 때문에 아래와 같이 체크했다.
     **/
```

```
        boolean searchOption = (type.equals("null")
                || keyword.equals("null")) ? false : true;

        /* Service 클래스를 이용해 no에 해당하는 게시 글 하나의 정보를 읽어온다.
         * 두 번째 인수에 true를 지정해 게시 글 읽은 횟수를 1 증가 시킨다.
         **/
        Board board = boardService.getBoard(no, true);

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

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "boardDetail";
    }
```

▶ 게시 글 상세보기 View

- src/main/webapp/WEB-INF/views/boardDetail.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!-- content -->
<div class="row my-5" id="global-content">
    <div class="col">
        <form name="checkForm" id="checkForm">
            <input type="hidden" name="no" id="no" value="${ board.no }"/>
            <input type="hidden" name="pass" id="rPass" />
            <input type="hidden" name="pageNum" value="${ pageNum }" />

            <%--
```

```
            검색 리스트에서 들어온 요청일 경우 다시 keyword에 해당하는
            검색 리스트로 돌려보내기 위해서 아래의 파라미터가 필요하다.
         --%>
        <c:if test="${ searchOption }">
            <input type="hidden" name="type" value="${ type }" />
            <input type="hidden" name="keyword" value="${ keyword }" />
        </c:if>
    </form>
    <div class="row text-center">
        <div class="col">
            <h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
        </div>
    </div>
    <div class="row my-3">
        <div class="col">
            <table class="table table-bordered" >
                <tbody>
                    <tr>
                        <th class="table-secondary">제 목</th>
                        <td colspan="3">${ board.title }</td>
                    </tr>
                    <tr>
                        <th>글쓴이</th>
                        <td>${ board.writer }</td>
                        <th>작성일</th>
                        <td><fmt:formatDate value="${ board.regDate }"
                            pattern="yyyy-MM-dd HH:mm:ss" /></td>
                    </tr>
                    <tr>
                        <th>비밀번호</th>
                        <td>
                            <div class="col-sm-8">
                                <input class="form-control" type="password"
name="pass" id="pass">
                            </div>
                        </td>
                        <th>조회수</th>
                        <td>${ board.readCount }</td>
                    </tr>
                    <tr>
                        <th>파&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;일</th>
                        <td colspan="3">
                        <c:if test="${ empty board.file1 }">
                            첨부파일 없음
                        </c:if>
```

```
                        <c:if test="${ not empty board.file1 }">
                            <a href="upload/${ board.file1 }">파일 다운로드</a>
                        </c:if>
                        </td>
                    </tr>
                    <tr>
                        <td colspan="4">
                            <pre>${ board.content }</pre>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
    <div class="row my-3">
        <div class="col text-center">
            <input class="btn btn-warning" type="button" id="detailUpdate" value="수정
하기"/>
              <input class="btn btn-danger"  type="button" id="detailDelete"
value="삭제하기" />
            <%--
                일반 게시 글 리스트에서 온 요청이면 일반 게시 글 리스트로 돌려 보낸다.
            --%>
            <c:if test="${ not searchOption }">
              <input class="btn btn-primary" type="button" value="목록보기"
                    onclick="location.href='boardList?pageNum=${pageNum}'"/>
            </c:if>
            <%--
                검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
            --%>
            <c:if test="${ searchOption }">
              <input class="btn btn-primary" type="button" value="목록보기"
                    onclick="location.href='boardList?pageNum=${pageNum}&type=${
type }&keyword=${ keyword }'"/>
            </c:if>
        </div>
    </div>
</div>
```

2-3) 게시 글쓰기 요청처리

게시 글쓰기 폼 요청은 모델 데이터가 필요 없기 때문에 Controller에서 이 요청에 대한 처리를 하지 않고 Spring MVC 설정 파일인 “WEB-INF/spring/appServlet/servlet-context.xml” 파일에

뷰 전용 컨트롤러를 아래와 같이 설정하였다.

```
<view-controller path="/writeForm" view-name="writeForm" />
<view-controller path="/boardWrite" view-name="writeForm" />
```

이번 검색 기능에서도 게시 글쓰기 폼에서 게시 글 리스트로 이동할 때는 게시 글 리스트의 첫 페이지로 이동하도록 할 것이다. 그러므로 **writeForm.jsp는 이전 예제와 동일한 파일이므로 교안에서 생략하였다.** 그리고 사용자가 게시 글쓰기 폼에서 작성한 게시 글을 저장하는 요청은 게시 글을 DB에 저장한 후 게시 글 리스트로 리다이렉트 될 때 사용자가 있었던 페이지로 보내는 것이 아니라 새로운 게시 글이 등록된 것을 확인할 수 있는 첫 페이지로 리다이렉트 하면 된다. 일반적으로 게시판에서 게시 글 리스트는 항상 최신 글을 맨 위에 보여주도록 설계되기 때문에 사용자가 자신이 작성한 신규 게시 글을 확인하려면 첫 페이지로 이동해야 하므로 앞에서와 마찬가지로 게시 글 쓰기가 완료되면 게시 글 리스트의 첫 페이지로 리다이렉트 시킬 것이다.
게시 글쓰기 폼에서 들어오는 게시 글쓰기 요청은 Controller, Service 계층과 DAO 계층 클래스 그리고 맵핑 구문과 뷰 페이지 모두가 이전 예제와 동일하기 때문에 교안에서 생략되었다.

2-4) 게시 글 수정 폼 요청처리

게시 글 수정하기는 게시 글을 수정할 수 있는 폼을 보여주는 요청과 게시 글 수정 폼에서 사용자가 수정한 게시 글을 저장하는 요청으로 나누어 처리해야 한다. 게시 글 수정 폼을 보여주는 요청은 이전에 작성한 게시 글을 폼에 출력하여 보여줘야 하므로 게시 글쓰기와는 다르게 DB에 저장된 게시 글 내용을 읽어와 폼에 출력하는 처리가 필요하다. 그리고 게시 글 수정 기능은 게시 글 상세보기에서 수정하기 버튼을 클릭해 수정 폼으로 이동하는 방식으로 구현할 것이다. 이 때 이전의 비밀번호를 검사해 비밀번호가 일치해야 게시 글 수정 폼으로 이동할 수 있도록 구현할 것이므로 비밀번호를 체크하는 기능도 필요하다.
이번 예제는 게시 글 검색 기능을 구현하는 것이므로 게시 글 리스트와 검색 리스트를 구분해서 처리해야 한다. 게시 글 수정 폼에서 게시 글 리스트와 검색 리스트로 돌아갈 때 사용자가 이전에 있었던 각각의 페이지로 이동시키기 위해서 pageNum과 type 그리고 keyword를 Controller 클래스에서 받아 모델에 담아 게시 글 수정 폼에서 사용할 수 있도록 구현해야 하며 게시 글 수정 폼에서 게시 글을 수정한 후 수정 요청을 할 때도 pageNum과 type 그리고 keyword를 요청 파라미터에 추가해야 한다.
게시 글 수정 폼 요청은 앞에서 구현한 Service 계층과 DAO 계층 클래스 그리고 맵핑 구문이 동일하기 때문에 교안에서 생략되었고 Controll와 뷰 페이지만 다음과 같이 수정하면 된다.

▶ Controller 클래스

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 수정 폼 요청을 처리하는 메서드
 *
 * 아래는 "/update"로 들어오는 요청을 처리하는 메서드를 지정한 것이다.
 * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
 *
```

```
 * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
 * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
 * no와 pass도 지정했다. 이렇게 Controller 메서드의 파라미터에 필요한
 * 객체나 요청 파라미터 이름과 동일한 이름의 파라미터를 지정하면 스프링이 자동으로
 * 설정해 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.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 **/
@RequestMapping(value="/update")
public String updateBoard(Model model, HttpServletResponse response,
        PrintWriter out, int no, String pass,
        @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) throws Exception {

    // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
    boolean result = boardService.isPassCheck(no, pass);

    // 비밀번호가 맞지 않으면
    if(! result) {

        /* 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.
         **/
        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;
```

```
        }

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

        /* Service 클래스를 이용해 no에 해당하는 게시 글 하나의 정보를 읽어온다.
         * 두 번째 인수로 false를 지정해 게시 글 읽은 횟수를 증가시키지 않는다.
         **/
        Board board = boardService.getBoard(no, false);

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

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "updateForm";
    }
```

▶ 게시 글 수정 폼 View

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

- src/main/webapp/WEB-INF/views/updateForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

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

<!-- 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">게시 글 수정하기</h2>
            </div>
        </div>
        <form name="updateForm" action="updateProcess" id="updateForm"
            class="row g-3 border-primary" method="post">
            <%--
                no는 DB에서 게시 글을 수정하기 위해 필요하고 pageNum은 게시 글이
                수정된 후에 이전에 사용자가 머물렀던 게시 글 리스트의 동일한 페이지로
                보내기 위해 필요한 정보이다.
            --%>
            <input type="hidden" name="no" value="${board.no}">
            <input type="hidden" name="pageNum" value="${ pageNum }" />

            <%--
                검색 요청일 경우 다시 keyword에 해당하는 검색 리스트로
                돌려보내기 위해서 아래의 파라미터가 필요하다.
             --%>
            <c:if test="${ searchOption }">
                <input type="hidden" name="type" value="${ type }" />
                <input type="hidden" name="keyword" value="${ keyword }" />
            </c:if>

            <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" value="${ board.writer }">
            </div>
            <div class="col-4 ">
                <label for="pass" class="form-label">비밀번호</label>
                <input type="password" class="form-control" name="pass"
                    id="pass">
            </div>
            <div class="col-8 offset-md-2">
                <label for="title" class="form-label">제 목</label>
                <input type="text" class="form-control" name="title"
                    id="title" value="${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">${ board.content }</textarea>
            </div>
            <div class="col-8 offset-md-2 text-center mt-5">
               <input type="submit" value="수정하기" class="btn btn-primary"/>
                <%--
                    일반 게시 글 리스트에서 온 요청이면 일반 게시 글 리스트로 돌려 보낸다.
                --%>
                <c:if test="${ not searchOption }">
                  <input class="btn btn-primary" type="button" value="목록보기"
                        onclick="location.href='boardList?pageNum=${pageNum}'"/>
                </c:if>
                <%--
                    검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
                --%>
                <c:if test="${ searchOption }">
                  <input class="btn btn-primary" type="button" value="목록보기"
                        onclick="location.href='boardList?pageNum=${pageNum}&type=${
type }&keyword=${ keyword }'"/>
                </c:if>
            </div>
        </form>
    </div>
</div>
```

2-5) 게시 글 수정 요청처리

사용자가 게시 글 수정 폼에서 수정한 게시 글을 수정하는 요청은 먼저 기존의 게시 글 비밀번호와 일치하는지 검사한 후 비밀번호가 일치할 때만 게시 글이 수정되도록 구현할 것이다. 그리고 게시 글 수정 요청은 수정된 게시 글을 DB에서 수정한 후 게시 글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다.

이번 예제는 게시 글 검색 기능을 구현하는 것이므로 게시 글 수정 요청을 처리한 후 게시 글 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 게시 글 리스트나 검색 리스트로 이동시킬 수 있다.

게시 수정 폼에서 들어오는 게시 글 수정 요청은 앞에서 구현한 Service 계층과 DAO 계층 클래스 그리고 맵핑 구문이 동일하기 때문에 교안에서 생략되었고 Controll만 수정하면 된다.

▶ Controller 클래스

게시 글 수정 폼에서 들어오는 게시 글 수정 요청을 처리하는 메서드와 게시 글 삭제 요청을 처리하는 메서드를 다음과 같이 수정하고 이들 메서드 안에서 리다이렉트할 때 RedirectAttributes객체를 이용해 pageNum과 type 그리고 keyword를 파라미터로 추가하는 부분도 다음과 같이 수정한

다.

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 수정 폼에서 들어오는 게시 글 수정 요청을 처리하는 메서드
 *
 * @RequestMapping의 ()에 value="/updateProcess", method=RequestMethod.POST를
 * 지정해 "/updateProcess"로 들어오는 POST 방식의 요청을 처리하도록 설정한 것이다.
 *
 * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
 * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
 * Board 객체를 지정했다. 또한 리다이렉트할 때 pageNum을 파라미터로 보내기
 * 위해서 RedirectAttributes 객체를 메서드의 파라미터로 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.
 *
 * @RequestMapping 애노테이션이 적용된 컨트롤러 메서드에 커맨드 객체를
 * 파라미터로 지정하면 커맨드 객체의 프로퍼티와 동일한 이름을 가진 요청
 * 파라미터의 데이터를 스프링이 자동으로 설정해 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 정수형으로 형 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 **/
@RequestMapping(value="/updateProcess", method=RequestMethod.POST)
public String updateBoard(HttpServletResponse response,
        PrintWriter out, Board board,
        RedirectAttributes reAttrs,
        @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) throws Exception {

    // BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
```

```
boolean result = boardService.isPassCheck(board.getNo(), board.getPass());

// 비밀번호가 맞지 않으면
if(! result) {

    /* 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.
     **/
    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;
}

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

// BoardService 클래스를 이용해 게시판 테이블에서 게시 글을 수정한다.
boardService.updateBoard(board);

/* 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 주소를 지정한다.
 *
 * 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.
 *
```

```
     * 아래와 같이 RedirectAttributs의 addAttribute() 메서드를 사용해
     * 지속적으로 사용할 파라미터를 지정하면 자동으로 주소 뒤에 파라미터로
     * 추가되며 addFlashAttribute() 메서드를 사용해 파라미터로 지정하면
     * 한 번만 사용할 수 있도록 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 URLEncoding을 코드로 구현해야 하지만
     * web.xml에서 스프링프레임워크가 지원하는 CharacterEncodingFilter를
     * 설정했기 때문에 Filter에 의해 UTF-8로 인코딩 되어 클라이언트로 응답된다.
     *
     * 아래는 게시 글 리스트로 Redirect 되면서 같이 보내야할 searchOption을
     * RedirectAttributs를 이용해 파라미터로 전달하는 예이다.
     **/
    reAttrs.addAttribute("searchOption", searchOption);

    // 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.
         *
         * 게시 글 상세 보기 요청을 처리하는 boardDetail() 메서드에서 뷰 페이지에서
         * 링크에 사용할 keyword를 java.net 패키지의 URLEncoder 클래스를
         * 이용해 수동으로 인코딩한 후 모델에 담아 뷰 페이지로 전달하였다.
         *
         * 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 인코딩 해줘야 하지만
         * web.xml에서 스프링프레임워크가 지원하는 CharacterEncodingFilter를
         * 설정했기 때문에 Filter에 의해 UTF-8로 인코딩 되어 클라이언트로 응답된다.
         *
         * 아래는 검색 리스트로 Redirect 되면서 같이 보내야할 keyword와 type을
         * RedirectAttributs를 이용해 파라미터로 전달하는 예이다.
         **/
        reAttrs.addAttribute("type", type);
        reAttrs.addAttribute("keyword", keyword);
    }

    reAttrs.addAttribute("pageNum", pageNum);
```

```
        //reAttrs.addFlashAttribute("test", "1회용 파라미터 받음 - test");
        return "redirect:boardList";
    }
```

2-6) 게시 글 삭제 요청처리

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

이번 예제는 게시 글 검색 기능을 구현하는 것이므로 게시 글 삭제 요청을 처리한 후 게시 글 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 게시 글 리스트나 검색 리스트로 이동시킬 수 있다.

게시 삭제 요청은 앞에서 구현한 Service 계층과 DAO 계층 클래스 그리고 맵핑 구문이 동일하기 때문에 교안에서 생략되었고 Controll만 수정하면 된다.

```
    /* 게시 글 상세보기에서 들어오는 게시 글 삭제 요청을 처리하는 메서드
     *
     * 아래는 "/delete", "/deleteBoard"로 들어오는 요청을 처리하는 메서드를 지정한 것이다.
     * method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
     *
     * @RequestMapping 애노테이션이 적용된 Controller 메서드의 파라미터에
     * HttpServletResponse와 PrintWriter를 지정했고 요청 파라미터를 받을
     * no와 pass도 지정했다. 그리고 리다이렉트 할 때 pageNum을 파라미터로 보내기
     * 위해서 RedirectAttributes 객체를 메서드의 파라미터로 지정했다.
     * 이렇게 Controller 메서드의 파라미터에 필요한 객체나 요청 파라미터 이름과 동일한
     * 이름의 파라미터를 지정하면 스프링이 자동으로 설정해 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.
     *
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
     * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
     * 참고하기 바란다.
     **/
    @RequestMapping({"/delete", "deleteBoard"})
    public String deleteBoard(HttpServletResponse response,
            PrintWriter out, int no, String pass,
            RedirectAttributes reAttrs,
```

```
        @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) throws Exception {

// BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.
boolean result = boardService.isPassCheck(no, pass);

// 비밀번호가 맞지 않으면
if(! result) {

    /* 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.
     **/
    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;
}

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

// BoardService 클래스를 이용해 게시판 테이블에서 게시 글을 수정한다.
boardService.deleteBoard(no);

/* 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 주소를 지정한다.
 *
 * 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.
 *
 * 아래와 같이 RedirectAttributs의 addAttribute() 메서드를 사용해
 * 지속적으로 사용할 파라미터를 지정하면 자동으로 주소 뒤에 파라미터로
 * 추가되며 addFlashAttribute() 메서드를 사용해 파라미터로 지정하면
 * 한 번만 사용할 수 있도록 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 URLEncoding을 코드로 구현해야 하지만
 * web.xml에서 스프링프레임워크가 지원하는 CharacterEncodingFilter를
 * 설정했기 때문에 Filter에 의해 UTF-8로 인코딩 되어 클라이언트로 응답된다.
 *
 * 아래는 게시 글 리스트로 Redirect 되면서 같이 보내야할 searchOption을
 * RedirectAttributs를 이용해 파라미터로 전달하는 예이다.
 **/
reAttrs.addAttribute("searchOption", searchOption);

// 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.
     *
     * 게시 글 상세 보기 요청을 처리하는 boardDetail() 메서드에서 뷰 페이지에서
     * 링크에 사용할 keyword를 java.net 패키지의 URLEncoder 클래스를
     * 이용해 수동으로 인코딩한 후 모델에 담아 뷰 페이지로 전달하였다.
     *
     * 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 인코딩 해줘야 하지만
     * web.xml에서 스프링프레임워크가 지원하는 CharacterEncodingFilter를
     * 설정했기 때문에 Filter에 의해 UTF-8로 인코딩 되어 클라이언트로 응답된다.
     *
```

```
            * 아래는 검색 리스트로 Redirect 되면서 같이 보내야할 keyword와 type을
            * RedirectAttributs를 이용해 파라미터로 전달하는 예이다.
            **/
            reAttrs.addAttribute("type", type);
            reAttrs.addAttribute("keyword", keyword);
        }

        reAttrs.addAttribute("pageNum", pageNum);
        //reAttrs.addFlashAttribute("test", "1회용 파라미터 받음 - test");
        return "redirect:boardList";
    }
```

4. 회원 로그인/로그아웃과 파일업로드 구현

앞에서 게시판을 구현하면서 기본 적인 CRUD 기능과 페이징 처리 그리고 게시 글 검색 기능을 구현하는 방법에 대해 알아보았다.
이번 챕터에서는 회원관련 서비스를 위한 별도의 Controller를 추가해 회원 로그인 기능과 로그아웃 기능을 구현하면서 HttpSession을 사용하는 방법에 대해 알아 볼 것이다. 또한 실무에서는 주민번호나 비밀번호와 같은 민감한 회원 정보는 암호화하여 저장하는 것이 일반적이므로 회원 로그인 요청을 처리할 때 스프링 시큐리티의 PasswordEncoder를 사용하여 로그인을 처리하는 방법에 대해 알아볼 것이다.
회원가입에 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.
파일 업로드를 위해서 Commons FileUpload API를 사용할 것이며 이 API를 사용할 경우 스프링 프레임워크에서 지원하는 CommonsMultipartResolver를 사용해 파일 업로드를 처리하는 방법에 대해 알아 볼 것이다.

1) 프로젝트 생성 및 환경설정

이번 예제는 앞에서 게시판 검색 기능까지 구현되어 있는 springstudy-bbs03 프로젝트를 복사해 springclass-bbs04 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 로그인과 로그아웃 기능을 추가할 것이다. 그리고 이어서 파일 업로드와 다운로드 기능도 추가할 것이다. 그렇기 때문에 프로젝트 생성과 설정 그리고 의존 라이브러리 설정 등에 대한 자세한 내용은 앞의 예제에서 설명한 주석을 참고 하길 바란다.
이번 예제도 앞에서 구현한 게시판에 회원 로그인/로그아웃과 파일 업로드/다운로드 기능을 추가하는 것이므로 설정파일과 소스가 거의 동일하기 때문에 전체 소스 코드는 추가하지 않고 필요하거나 수정되는 부분의 소스 코드를 추가하고 붉은색 볼드체로 표시할 것이다.

1-1) 프로젝트 생성

- **실습용 프로젝트 : springclass-bbs04**
- **완성 프로젝트 : springstudy-bbs04**

1-2) 테이블 생성 및 데이터 입력

이번 예제는 앞에서 사용한 springbbs 테이블 수정 없이 그대로 사용하고 회원관련 테이블을 새롭게 생성해 회원정보를 추가하고 사용할 것이다. 아래 member.sql 파일을 살펴보면 회원정보 중에서 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 spring;
use spring;

# 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/0iMuemBeF4Y8.7WZeVKAoudv/VzgQx697lYlZgQxr/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$10$b3t8sn6QZGHYaRx3OS5KUuPxzWZdY5yHPRxlSdAgByQ7v0BlCLzrO',
    '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('servlet', '강감찬',
'$2a$10$.g6l.wyIFO1.j4u4gvVtKOnG9ACBUT1GRlDwlMZcjBxZPrCAURLaG',
    'midas@daum.net', '010-5687-5678', '06043',
    '서울 강남구 강남대로146길 28 (논현동, 논현아파트)', '논현신동아파밀리에아파트 111동 1234
호',
    '02-5326-5678', '1', '2022-06-05 12:10:30');

commit;
SELECT * FROM member;
```

1-3) 의존 라이브러리와 BuildPath 설정

Spring Legacy Project 메뉴를 이용해 Spring MVC Project를 생성하게 되면 자바 버전은 1.6,

스프링프레임워크 버전 3.1.1, Dynamic Web Module 2.5가 기본 설정되어 있다.
우리는 스프링프레임워크 버전 4.2.4 버전을 사용할 것이므로 Maven을 통해 버전을 변경해야 한다. 그리고 자바 1.8 버전과 Dynamic Web Module 3.0을 사용할 것이므로 Configure Build Path를 통해 Java Build Path의 자바 버전과 Java Compiler 버전을 1.8로 설정하고 Server Runtime 설정을 변경해야 한다. 그리고 프로젝트를 생성할 때 패키지의 3번째 단계에 지정한 bbs가 ContextRoot와 Artifact Id로 사용되기 때문에 이를 변경해야 할 필요도 있을 것이다. 또한 이번 프로젝트는 MyBatis와 스프링프레임워크를 연동하고 스프링 시큐리티의 PasswordEncoder를 이용해 사용자가 입력한 데이터와 암호화되어 저장된 데이터를 비교해 로그인을 처리하고 파일 업로드 기능을 구현할 것이므로 mybatis 모듈과 mybatis-spring 모듈이 필요하고 spring-security-web 모듈과 commons-fileupload 라이브러리가 필요하다.
우리는 commons-fileupload 라이브러리 버전은 1.3.3을 사용해 파일 업로드를 구현할 것이므로 pom.xml에서 이 라이브러리를 의존설정 해야 한다.
이에 대한 자세한 내용은 springstudy-bbs04의 web.xml의 주석을 참고하길 바란다.

1-4) Spring MVC Bean 정의

Spring MVC Bean 정의 파일은 로그인 폼 요청을 처리하는 뷰 전용컨트롤러 설정을 제외하고 앞 예제와 동일하기 때문에 아래에서 붉은색 볼드체로 표시한 로그인 폼에 대한 뷰 전용컨트롤러 만 Spring MVC Bean 정의 파일에 추가하면 된다.

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

... 중략 ...

```
<!--
    Redirect View 설정
    특정 URI에 대한 클라이언트의 요청을 다른 페이지로 Redirect 시켜야할 경우에
    아래와 같이 리다이렉트 전용 컨트롤러를 설정할 수 있다.

    아래는 ContextRoot("/"), ContextRoot/index, ContextRoot/default로
    들어오는 요청을 게시 글 리스트로 리다이렉트 시키는 예 이다.
-->
<view-controller path="/" view-name="redirect:/boardList" />
<redirect-view-controller path="/index" redirect-url="/boardList"/>
<redirect-view-controller path="/default" redirect-url="/boardList"/>
<view-controller path="loginForm" view-name="loginForm" />
```

... 중략...

1-5) 로그인 처리를 위한 Spring-Security관련 Bean 정의

비밀번호 암호화 처리를 위해서 Spring이 지원하는 BCryptPasswordEncoder를 Bean으로 정의하는 spring-security.xml 파일을 새롭게 만들고 아래 코드를 참고하여 작성해 보자.

- src/main/webapp/WEB-INF/spring/spring-security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:security="http://www.springframework.org/schema/security"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security-4.2.xsd">

    <bean id="bcryptPasswordEncoder"
        class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />

</beans>
```

1-6) 파일 업로드를 위한 Spring Bean 정의

파일 업로드를 위해 Spring이 지원하는 CommonsMultipartResolver를 Bean으로 정의하는 코드를 아래 붉은색 볼드체로 작성된 부분을 참고해 root-context.xml 파일 맨 아래에 추가하면 된다.

- src/main/webapp/WEB-INF/spring/root-context.xml

```
... 중략 ...

    <!--
        9. Commons FileUpload API를 지원하기 위해 스프링프레임워크가 제공하는
           CommonsMultipartResolver를 Bean으로 정의

        파일 업로드를 위해 스프링이 제공하는 멀티파트 지원 기능을 사용하려면
        MultipartResolver를 스프링 설정 파일에 빈으로 등록해야 한다.
        MultipartResolver는 멀티파트 형식으로 encoding된 데이터가
        전송된 경우 해당 데이터를 스프링 MVC에서 사용할 수 있도록 변환하는
        역할을 한다. 예를 들자면 @RequestParam 애노테이션을 이용해
        멀티파트로 전송된 파라미터 값과 파일 데이터를 사용할 수 있게 해준다.

        스프링에서 기본 제공하는 MultipartResolver는 아래 두 개가 있다.
        이 두 개의 MultipartResolver 구현체 중 사용하고자 하는 클래스를
        스프링 빈으로 등록하면 된다.  한 가지 주의할 점은 빈의 이름(id)은
        반드시 multipartResolver 로 지정해야 한다는 것이다.
        DispatcherServlet이 이 이름을 가진 빈을 찾아 사용하기 때문에 다른
        이름을 지정할 경우 MultipartResolver로 사용할 수 없다.

        - Commons FileUpload API를 사용해 멀티파트 데이터를 처리할 경우
```

```
        org.springframework.web.multipart.commons.CommonsMultipartResolver

        - 서블릿 3.0의 Part를 사용해 멀티파트 데이터를 처리할 경우
        org.springframework.web.multipart.support.StandardServletMultipartResolver

        아래는 CommonsMultipartResolver 클래스에서 많이 사용하는 프로퍼티의 설명이다.
        - maxUploadSize
        최대 업로드 가능한 파일의 바이트 크기로 업로드 하는 파일의 크기를
        제한하지 않으려면 -1을 지정하면 된다. 기본 값은 -1 이다.

        - maxInMemorySize
        디스크에 임시 파일을 생성하기 전에 메모리에 보관할 수 있는 최대 바이트
        크기로 기본 값은 10240Byte 이다.

        - defaultEncoding
        요청을 파싱할 때 사용할 문자 셋으로 지정하지 않을 경우
        HttpServletRequest.setCharacterEncoding() 메소드로 지정한
        문자 셋이 적용된다. 기본 값은 ISO-8859-1이 적용된다.
    -->
    <bean id="multipartResolver" class=
        "org.springframework.web.multipart.commons.CommonsMultipartResolver" />

... 중략 ...
```

1-7) 웹 애플리케이션 배포 서술자(Deployment Descriptor) 수정

ContextLoaderListener가 읽어서 초기화 작업을 수행할 수 있도록 web.xml 파일에서 웹 애플리케이션 초기화 파라미터 설정 부분에 spring-security.xml 파일을 아래와 같이 추가하자.

- src/main/webapp/WEB-INF/web.xml

```
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring/root-context.xml
            /WEB-INF/spring/spring-security.xml
        </param-value>
    </context-param>
```

2) 회원 로그인/로그아웃 구현

먼저 회원 한 명의 정보를 저장하는 Domain 클래스를 다음을 참고해 작성하고 MemberDao 인터페이스와 그 구현체인 MemberDaoImpl 클래스, SQL 쿼리를 분리해 작성한 MemberMapper, 그리고 MemberService 인터페이스와 그 구현체인 MemberServiceImpl을 차례로 구현하고 회원관련 요청을 전담하여 처리하는 MemberController를 새로 추가하여 회원 로그인/로그아웃 처리를 구현할 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 입력 받을 수 있도록 구현할 것이고 또 다른 하나는 로그인 요청이 발생한 현재 페이지에서 팝업 형식으로 로그인 폼 모달 창을 표시하여 회원 아이디와 비밀번호를 입력 받을 수 있도록 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 만들뿐 로그인을 처리하는 로직은 하나만 구현하면 된다. 다시 말해 Controll, Service, Dao 클래스에서 로그인 기능을 처리하는 메서드는 하나만 작성하면 된다는 말이다.

2-1) 회원 로그인/로그아웃 처리

▶ Domain 클래스

```
/* 한 명의 회원 정보를 저장하는 클래스(VO, Beans, DTO)
 * 회원 정보를 저장하고 있는 테이블의 필드와 1:1 맵핑되는 Domain 클래스
 **/
public class Member {

    private String name, id, pass, email, mobile;
    private String phone, zipcode, address1, address2;
    private boolean  emailGet;
    private Timestamp regDate;

    public Member() { }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
```

```
public String getPass() {
    return pass;
}
public void setPass(String pass) {
    this.pass = pass;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
public String getMobile() {
    return mobile;
}
public void setMobile(String mobile) {
    this.mobile = mobile;
}
public String getPhone() {
    return phone;
}
public void setPhone(String phone) {
    this.phone = phone;
}
public String getZipcode() {
    return zipcode;
}
public void setZipcode(String zipcode) {
    this.zipcode = zipcode;
}
public String getAddress1() {
    return address1;
}
public void setAddress1(String address1) {
    this.address1 = address1;
}
public String getAddress2() {
    return address2;
}
public void setAddress2(String address2) {
    this.address2 = address2;
}
public boolean isEmailGet() {
    return emailGet;
}
```

```
    public void setEmailGet(boolean emailGet) {
        this.emailGet = emailGet;
    }
    public Timestamp getRegDate() {
        return regDate;
    }
    public void setRegDate(Timestamp regDate) {
        this.regDate = regDate;
    }
}
```

▶ DAO(Data Access Object) 계층 구현

com.springstudy.bbs.dao 패키지에 MemberDao 인터페이스를 새롭게 만들고 회원 로그인과 한 명의 회원 정보를 반환하는 추상메서드를 정의한다.

- com.springstudy.bbs.dao.MemberDao

```
public interface MemberDao {
    /**
     * 한 명의 회원 정보를 반환하는 메서드
     * @param id는 회원 아이디
     * @return id에 해당하는 회원 정보를 Member 객체로 반환
     **/
    public Member getMember(String id);
}
```

▶ 회원 관련 SQL을 분리한 Mapper

src/main/resources/repository/mappers/에 있는 Mapper-Template.xml을 복사해 파일 이름을 MemberMapper.xml로 변경하고 다음의 코드를 참고해 Mapper를 작성하자.

- src/main/resources/repository/mappers/MemberMapper.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">

<!--
    맵퍼의 namespace 속성은 맵핑 구문을 그룹핑 하는 역할을 한다.
    여러 맵퍼에서 맵핑 구문의 id 속성 값이 중복되더라도 namespace와
    맵핑 구문의 id 속성에 지정한 값을 합쳐서 호출하기 때문에 맵핑 구문이
    중복되지 않게 분류하여 관리할 수 있다.
    테이블 이름을 바탕으로 namespace를 지정하고 맵핑 구문의 id 속성의
    값은 SQL 문의 맥락에 따라서 명명하는 것이 일반적 이다.

    parameterType 속성에는 주로 SQL 문의 조건에 사용할 파라미터의
```

```
    데이터 타입을 지정하는 속성으로 자바 원시 타입(기본 타입, String)과 자바빈,
    Map과 같은 타입을 지정할 수 있다. resultType 속성도 parameterType
    속성에서 지정한 타입을 많이 사용한다. parameterType과 resultType에
    지정할 데이터 타입이 자바 원시 타입이면 생략해도 잘 동작한다.

    SQL문의 조건에 사용할 파라미터는 아래와 같이 #{} 로 감싸서 지정하면 된다.
-->
<mapper namespace="com.springstudy.bbs.mapper.MemberMapper" >

    <!--
        회원 id에 해당하는 회원 정보를 반환하는 맵핑 구문
        테이블의 컬럼명은 일반적으로 언더스코어 표기법("_")을 사용하는 경우가
        많고 클래스의 인스턴스 맴버는 카멜표기법을 사용한다.
        테이블의 컬럼명과 모델 클래스의 프로퍼티 이름이 다른 경우 아래와 같이
        SELECT 쿼리에 별칭을 사용해 모델 클래스의 프로퍼티 이름과 동일하게
        맞춰야 한다. 그렇지 않으면 오류는 발생하지 않지만 데이터를 읽어 올
        수 없다.

        SELECT member_id AS memberId FROM Member
    -->
    <select id="getMember" resultType="Member">
        SELECT * FROM member WHERE id = #{id}
    </select>
</mapper>
```

▶ MemberDao 인터페이스 구현 클래스

com.springstudy.bbs.dao 패키지에 MemberDao를 구현하는 MemberDaoImpl 클래스를 새롭게 만들고 MyBatis의 SessionTemplate을 이용해 회원 로그인을 체크에 필요한 비밀번호를 반환하는 메서드와 회원 한 명의 정보를 반환하는 메서드를 다음과 같이 구현한다.

- com.springstudy.bbs.dao.MemberDaoImpl

```
// 이 클래스가 데이터 액세스(데이터 저장소) 계층의 컴포넌트임을 선언한다.
@Repository
public class MemberDaoImpl implements MemberDao {

    /* 이 예제는 MyBatis가 제공하는 SqlSessionTemplate 객체를 사용하기
     * 때문에 스프링으로부터 DI 받을 수 있도록 생성자나 setter를 준비해야 한다.
     *
     * mybatis-spring 모듈은 MyBatis의 SqlSession 기능과 스프링 DB 지원 기능을
     * 연동해 주는 SqlSessionTemplate 클래스를 제공한다. SqlSessionTemplate은
     * SqlSession을 구현해 스프링 연동 부분을 구현하였기 때문에 우리가 만드는 DAO에서
     * SqlSessionTemplate 객체를 사용해 SqlSession에 정의된 메서드를 사용할 수 있다.
     *
     * SqlSession과 SqlSessionTemplate는 같은 역할을 담당하고 있지만 트랜잭션
```

```
     * 처리에서 다른 부분이 있다. SqlSession은 commit(), rollback() 메서드를
     * 명시적으로 호출해 트랜잭션을 처리 하지만 SqlSessionTemplate은 스프링이
     * 트랜잭션을 처리할 수 있도록 구현되어 있기 때문에 별도로 commit(), rollback()
     * 메서드를 호출할 필요가 없다.
     **/
    private SqlSessionTemplate sqlSession;

    /* src/main/resources/repository/mappers/MemberMapper.xml에
     * 정의한 Mapper namespace를 상수로 정의
     **/
    private final String NAME_SPACE = "com.springstudy.bbs.mapper.MemberMapper";

    @Autowired
    public void setSqlSession(SqlSessionTemplate sqlSession) {
        this.sqlSession = sqlSession;
    }

    // member 테이블에서 id에 해당하는 회원 정보를 읽어오는 메서드 반환하는 메서드
    @Override
    public Member getMember(String id) {

        /* 아래와 같이 SqlSessionTemplate의 메서드가 호출되면
         * repository/mappers/MemberMapper.xml 맵퍼 파일에서
         * mapper 요소의 namespace 속성에 지정한
         * com.springstudy.bbs.mapper.MemberMapper인 맵퍼가
         * 선택되고 그 하부에 <select> 요소의 id 속성에 지정한 getMember인
         * 맵핑 구문이 선택되어 MyBatis 내부에서 JDBC 코드로 변환되어 실행된다.
         *
         * 매핑 구문에 resultType 속성에 Member를 지정했기 때문 Member 객체가
         * 반환된다. id가 존재하지 않으면 검색된 결과가 없으므로 null이 반환 된다.
         **/
        return sqlSession.selectOne(NAME_SPACE + ".getMember", id);
    }
}
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 MemberService 인터페이스를 새롭게 만들고 회원 로그인과 한 명의 회원 정보를 반환하는 아래 추상메서드를 정의한다.

- com.springstudy.bbs.service.MemberService

```
public interface MemberService {

    /**
     * 회원 로그인을 처리하는 메서드
```

```
     * @param id는 회원 아이디
     * @param pass는 회원 비밀번호
     * @return 로그인 처리 결과를 정수로 반환
     */
    public int login(String id, String pass);

    /**
     * 한 명의 회원 정보를 반환하는 메서드
     * @param id는 member 테이블의 Primary Key
     * @return id에 해당하는 회원 정보를 Member 객체로 반환
     **/
    public Member getMember(String id);
}
```

▶ MemberService 인터페이스 구현 클래스

com.springstudy.bbs.service 패키지에 MemberService를 구현하는 MemberServiceImpl 클래스를 새롭게 만들고 회원 로그인을 처리하는 메서드와 회원 한 명의 정보를 반환하는 메서드를 다음과 같이 구현한다. 회원 로그인을 처리하는 login() 메서드에서는 사용자가 입력한 비밀번호와 DB 테이블에 암호화되어 저장된 데이터를 스프링 시큐리티에서 제공하는 BCryptPasswordEncoder 객체를 이용해 비밀번호가 맞는지를 체크하는 기능을 구현하고 있다.

- com.springstudy.bbs.service.MemberServiceImpl

```
// 이 클래스가 서비스(비즈니스 로직) 계층의 컴포넌트임을 선언한다.
@Service
public class MemberServiceImpl implements MemberService {

    /* 인스턴스 필드에 @Autowired annotation을 사용하면 접근지정자가
     * private이고 setter 메서드가 없다고 하더라도 문제없이 주입 할 수 있다.
     * 기본 생성자가 반드시 존재해야 스프링이 이 클래스의 인스턴스를 생성한 후
     * setter 주입 방식으로 주입해 준다. 이 클래스에는 다른 생성자가 존재하지
     * 않으므로 컴파일러에 의해 자동으로 기본 생성자가 만들어 진다.
     **/
    @Autowired
    private MemberDao memberDao;

    /* 회원 비밀번호에 대한 암호화 인코딩관련 스프링 시큐리티의 PasswordEncoder
     * 회원 로그인 요청시 DB 테이블에 암호화 인코딩되어 저장된 비밀번호와 사용자가
     * 입력한 일반 문자열 비밀번호를 비교하는데 사용되고 회원 가입과 회원 정보 수정에서
     * 사용자가 입력한 비밀번호를 암호화 인코딩하여 저장하는데도 사용된다.
     **/
    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    public void setMemberDao(MemberDao memberDao) {
```

```
        this.memberDao = memberDao;
    }

    // MemberDao를 이용해 로그인 요청 처리 결과를 반환하는 메서드
    @Override
    public int login(String id, String pass) {

        Member m = memberDao.getMember(id);

        int result = -1;

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

        /* 로그인 성공
         * BCryptPasswordEncoder 객체의 matches 메소드를 이용해 암호가 맞는지 확인
         * matches() 메소드의 첫 번 인수로 인코딩이 않된 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
                result = 0;
        }

        return result;
    }

    // MemberDao를 이용해 id에 해당하는 회원 정보를 가져오는 메서드
    @Override
    public Member getMember(String id) {
        return memberDao.getMember(id);
    }
}
```

▶ Controller 클래스

com.springstudy.bbs.controller 패키지에 MemberController 클래스를 새롭게 만들고 회원 로그인과 로그아웃 요청을 처리하는 메서드를 아래 코드를 참고해 작성한다.

- com.springstudy.bbs.controller.MemberController

```
// 스프링 MVC의 컨트롤러임을 선언하고 있다.
```

```
@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 {

    private MemberService memberService;

    /* @Autowired annotation을 사용해 MemberService 구현체를 셋터 주입하고 있다.
     * 스프링이 기본 생성자를 통해 이 클래스의 인스턴스를 생성한 후 setter 주입
     * 방식으로 MemberService 구현체를 주입해 준다. 셋터 주입은 반드시 기본 생성자가
     * 존재해야 하지만 이 클래스에는 다른 생성자가 존재하지 않으므로 컴파일러에 의해
     * 자동으로 기본 생성자가 만들어 진다.
     **/
    @Autowired
    public void setMemberService(MemberService memberService) {
        this.memberService = memberService;
    }

    /* @RequestMapping의 ()에 method=RequestMethod.Post를 지정해
     * "/login"으로 들어오는 POST 방식의 요청을 처리할 수 있도록 설정하고 있다.
     *
     * 요청을 처리한 결과를 뷰에 전달하기 위해 사용하는 것이 모델 객체이다.
     * 컨트롤러는 요청을 처리한 결과 데이터를 모델에 담아 뷰로 전달하고 뷰는
     * 모델로 부터 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 애노테이션이 적용된 메서드의 파라미터에 Model
     * 을 지정하면 스프링이 이 메서드를 호출하면서 Model 타입의 객체를 넘겨준다.
     * 우리는 Model을 받아 이 객체에 결과 데이터를 담기만 하면 뷰로 전달된다.
     *
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터에
     * @RequestParam 애노테이션에 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 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="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.
 *
 * @RequestMapping 애노테이션이 적용된 메서드에 요청 파라미터
 * 이름과 메서드의 파라미터 이름이 같은 경우 @RequestParam 애노테이션을
 * 지정하지 않아도 스프링으로부터 요청 파라미터를 받을 수 있다.
 **/
@RequestMapping(value="/login", method=RequestMethod.POST)
public String login(Model model, @RequestParam("userId") String id,
        @RequestParam("pass") String pass,
        HttpSession session, HttpServletResponse response)
                throws ServletException, IOException {

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

    if(result == -1) {
        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) {
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<script>");
        out.println("alert('비밀번호가 다릅니다.');");
        out.println("location.href='loginForm'");
        out.println("</script>");
```

```
            return null;
        }

        Member member = memberService.getMember(id);
        session.setAttribute("isLogin", true);

        /* 클래스 레벨에 @SessionAttributes("member") 애노테이션을
         * 지정하고 그 컨트롤러의 메서드에서 아래와 같이 동일한 이름으로 모델에
         * 추가하면 스프링이 세션 영역에 데이터를 저장해 준다.
         **/
        model.addAttribute("member", member);
        System.out.println("member.name : " + member.getName());

        /* 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.
         *
         * 로그인이 성공하면 게시 글 리스트로 리다이렉트 된다.
         **/
        return "redirect:/boardList";
    }

    /* @RequestMapping의 ()에 method 속성을 지정하지 않았으므로
     * "/logout" 으로 들어오는 GET과 POST 방식 요청 모두를 처리할 수 있다.
     **/
    @RequestMapping("/logout")
    public String 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 게시 글 리스트로 리다이렉트 된다./
```

```
         **/
        return "redirect:/boardList";
    }
}
```

▶ 웹 템플릿 View

웹 템플릿에 사용되는 파일들도 회원 로그인과 로그아웃 처리 부분을 제외하면 앞의 예제와 거의 동일하다. 앞에서 언급했던 것처럼 이번에 회원 로그인을 구현할 때 사용할 로그인 폼은 별도의 로 로그인 폼 페이지를 표시하는 방식과 현재 페이지에서 팝업 형식으로 로그인 폼 모달 창을 표시하는 2가지 유형을 사용할 것이므로 별도의 로그인 폼 페이지는 따로 작성하면 되지만 팝업 형식으로 로그인 폼 모달 창을 표시하기 위해서는 전체 뷰 페이지에 적용될 수 있는 곳에 로그인 폼 모달 창을 작성해야 하므로 index.jsp 파일에 로그인 폼 모달 창을 작성해야 한다. 앞의 예제에서 **추가되거나 수정되는 부분은 붉은색 볼드체로 표시하였다.**

* 메인 템플릿 페이지

- src/main/webapp/WEB-INF/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Spring MVC 게시판</title>
    <!--
        BoardController에서 클래스 레벨에 @RequestMapping 애노테이션을 사용해
        별도로 경로 매핑을 하지 않고 boardList(Model model) 메서드에만
        @RequestMapping({"/boardList", "/list"}) 애노테이션으로 요청 매핑을 설정했다.
        그리고 WEB-INF/spring/appServlet/servlet-context.xml에서 정적 리소스와
        관련된 url 맵핑을 아래와 같이 설정했기 때문에
        <mvc:resources mapping="/resources/**" location="/resources/" />
        css의 위치를 "resources/css/index.css"와 같이 지정해야 한다.

        브라우저 주소 표시줄에 http://localhost:8080/springstudy-bbs01/list
        또는 http://localhost:8080/springstudy-bbs01/boardList 등으로
        표시되므로 css 디렉터리는 ContextRoot/resources/css에 위치하기 때문에 현재
        위치를 기준으로 상대 참조 방식으로 "resources/css/index.css"를 지정해야 한다.
    -->
    <title>스프링 게시판</title>
    <link href="resources/bootstrap/bootstrap.min.css" rel="stylesheet" >
    <link rel="stylesheet" type="text/css" href="resources/css/member.css" />
    <style>
    </style>
```

```
    <script src="resources/js/jquery-3.2.1.min.js"></script>
    <script src="resources/js/formcheck.js"></script>
    <script src="resources/js/member.js"></script>
</head>
<body>
    <div class="container">
        <%@ include file="template/header.jsp" %>
        <jsp:include page="${ param.body }" />
        <%@ include file="template/footer.jsp" %>
    </div>
    <script src="resources/bootstrap/bootstrap.bundle.min.js"></script>

    <!-- 로그인 모달 -->
    <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 action="login" method="post">
                    <div class="modal-body">
                        <div class="mb-3">
                            <label for="userId" class="col-form-label fw-bold">아이디 :
</label>
                            <input type="text" class="form-control" id="userId"
name="userId">
                        </div>
                        <div class="mb-3">
                            <label for="pass" class="col-form-label fw-bold">비밀번호 :
</label>
                            <input type="password" class="form-control" id="pass"
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>
```

*** 템플릿 헤더 페이지**

- src/main/webapp/WEB-INF/template/header.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header  -->
<div class="row border-bottom border-primary" id="global-header">
    <div class="col-4">
        <p><img src="https://via.placeholder.com/200x100"></p>
    </div>
    <div class="col-8">
        <div class="row mt-1">
            <div class="col">
                <ul class="nav justify-content-end">
                    <li class="nav-item">
                        <a class="nav-link"
                            href='${ sessionScope.isLogin ? "logout" : "loginForm" }'>
                            ${ sessionScope.isLogin ? "로그아웃" : "로그인-폼" }
                        </a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link "
                            ${ not sessionScope.isLogin ? "data-bs-toggle='modal'
data-bs-target='#loginModal'" : ""}
                            href='${ sessionScope.isLogin ? "logout" : "#" }'>
                            ${ sessionScope.isLogin ? "로그아웃" : "로그인-모달" }
                        </a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="boardList">게시 글 리스트</a>
                    </li>
                    <li class="nav-item">
                        <c:if test="${ not sessionScope.isLogin }" >
                            <a class="nav-link" href="#">회원가입</a>
                        </c:if>
                        <c:if test="${ sessionScope.isLogin }" >
                            <a class="nav-link" href="#">정보수정</a>
                        </c:if>
                    </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"> </div>
        </div>
        <div class="row">
            <div class="col text-end pe-5 text-primary">
                <c:if test="${ sessionScope.isLogin }" >
                    <div>안녕하세요 ${ sessionScope.member.name }님</div>
                </c:if>
            </div>
        </div>
    </div>
</div>
```

▶ 로그인 폼 View

아래의 로그인 폼을 새롭게 추가한다.

- src/main/webapp/WEB-INF/views/loginForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<div class="row my-5" id="global-content">
    <div class="col">
        <form class="my-5" id="loginForm" action="login" method="post">
            <h2 class="fw-bold">Member Login</h2>
            <fieldset>
                <legend>Member Loin</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" />
                <p id="btn1">
                    <input type="checkbox" id="saveId" value="savedIdYes" />
                    <label for="saveId">아이디저장</label>
                    <input type="checkbox" id="secure" value="secureYes" />
                    <label for="secure">보안접속</label>
                </p>
                <p id="btn2">
                    <input type="button" value="회원가입" id="btnJoin" />
                    <input type="button" value="아이디/비밀번호 찾기" id="btnSearch" />
                </p>
            </fieldset>
        </form>
    </div>
</div>
```

▶ 스타일(CSS)

회원 로그인 폼에서 사용되는 스타일도 새롭게 추가한다.

- src/main/webapp/resources/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 #btn1 {
    clear: both;
    margin-left: 60px;
    padding: 10px 0px;
}
#loginForm #btn1 label {
    font-size: 11px;
    vertical-align: middle;
}
#loginForm #btn1 input{
    height: 20px;
}
#loginForm #btn2 {
    margin-left: 60px;
```

```
}
#loginForm #btn2 input {
    background-color: #666;
    border-style: none;
    border-radius: 5px;
    color: #fff;
    height: 25px;
    padding: 5px 10px;
}
```

▶ 자바스크립트

회원 로그인 폼이 submit 될 때 아이디와 비밀번호가 제대로 입력되었는지 유효성 검사를 수행하는 **member.js 자바스크립트 파일을 새롭게 만들어 아래 경로에 추가한다.**

- src/main/webapp/resources/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;
        }
    });
});
```

3) 파일 업로드/다운로드 구현

게시 글쓰기에서 파일 업로드를 구현하고 게시 글 상세보기에서 파일 다운로드를 구현할 것이다.
파일 업로드 기능이 추가되면 게시 글 수정하기에서도 기존에 업로드된 파일을 수정하는 기능도 제공해야 할 것이다. 이 부분은 게시 글쓰기의 파일 업로드 부분을 참고해 직접 구현해 보자.

3-1) 파일 업로드

스프링은 MultipartResolver를 사용해 Multipart 데이터에 접근할 수 있는 아래와 같은 다양한 방법을 제공하고 있다.

- MultipartFile 인터페이스를 이용한 접근
- @RequestParam 애노테이션을 이용해 MultipartFile에 접근
- MultipartHttpServletRequest를 이용해 MultipartFile에 접근
- 커맨드 객체를 이용한 접근(커맨드 객체에 MultipartFile 타입의 프로퍼티가 있어야 함)
- Servlet 3.0의 Part를 이용한 접근

우리는 이중에서 @RequestParam 애노테이션과 MultipartHttpServletRequest 객체를 이용해 MultipartFile에 접근해 파일 업로드를 처리하는 방법에 대해 알아 볼 것이다.
먼저 업로드 되는 파일을 저장할 "src/main/webapp/resources/"에 upload 폴더를 만들자.
이번 예제도 앞에서 구현 기능에 파일 업로드/다운로드를 구현하는 것이므로 Service와 DAO 클래스는 동일하기 때문에 교안에서 생략할 것이다.

▶ 게시 글쓰기 폼 View

게시 글쓰기 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 추가하면 된다.

- src/main/webapp/WEB-INF/views/writeForm.jsp에서 수정

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

    ... 중략...

    <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">
        <label for="file1" class="form-label">파 일</label>
        <input type="file" class="form-control" name="file1"  id="file1" >
    </div>
```

```
        ... 중략...

    </form>
```

▶ Service, DAO 클래스 및 BoardMapper.xml

Service 계층의 클래스와 DAO 계층의 클래스는 앞에서 사용한 클래스 그대로 사용하면 되고 BoardMapper.xml 파일에서 게시 글을 추가하는 insertBoard 매핑 구문에 파일을 추가하는 부분만 다음과 같이 추가하면 된다.

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

```
<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>
```

▶ Controller 클래스

BoardController 클래스에 MultipartFile 인터페이스를 이용해 파일 업로드를 처리할 기존의 insertBoar() 메서드를 아래와 같이 수정하고 MultipartHttpServletRequest를 이용해 파일 업로드를 처리할 addBoard() 메서드를 새롭게 추가한다. 그리고 업로드 되는 파일을 저장할 폴더 위치를 DEFAULT_PATH 라는 상수로 정의한다.
게시 글쓰기가 완료된 후 그 결과를 화면에 표시하는 것이 아니라 게시 글 리스트로 리다이렉트 되도록 구현할 것이므로 별도의 뷰 페이지는 필요 없다.

- com.springstudy.bbs.controller.BoardController

```
//스프링 MVC의 컨트롤러임을 선언하고 있다.
@Controller
public class BoardController {

    // 업로드한 파일을 저장할 폴더 위치를 상수로 선언하고 있다.
    private final static String DEFAULT_PATH = "/resources/upload/";

    ... 중략 ...

    /* 게시 글쓰기 폼에서 들어오는 게시 글쓰기 요청을 처리하는 메서드
     * @RequestParam 애노테이션을 이용해 MultipartFile에 접근하기
     *
     * @RequestMapping의 ()에 value="/writeProcess", method=RequestMethod.Post를
     * 지정해 "/writeProcess"로 들어오는 POST 방식의 요청을 처리하는 메서드를
```

```
 * 지정한 것이다.
 *
 * 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.
 *
 * @RequestMapping 애노테이션이 적용된 컨트롤러 메서드에 커맨드 객체를
 * 파라미터로 지정하면 커맨드 객체의 프로퍼티와 동일한 이름을 가진 요청
 * 파라미터의 데이터를 스프링이 자동으로 설정해 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 정수형으로 형 변환 할 수 없는 경우 400 에러를 발생 시킨다.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 *
 * 아래에서 title, writer, content, pass 등은 요청 파라미터의 이름과
 * 동일하기 때문에 메서드의 파라미터에 @RequestParam 애노테이션을
 * 적용하지 않아도 스프링이 요청 파라미터의 값을 자동으로 바인딩 시켜준다.
 * 하지만 multipartFile은 요청 파라미터가 file1이므로 서로 이름이 다르기
 * 때문에 @RequestMapping을 이용해 value 속성에 요청 파라미터의
 * 이름을 지정해야 스프링이 MultipartFile 객체를 바이딩 시켜준다.
 **/
@RequestMapping(value="/writeProcess", method=RequestMethod.POST)
public String insertBoard(
        HttpServletRequest request,
        String title, String writer, String content, String pass,
        @RequestParam(value="file1", required=false) MultipartFile multipartFile)
                throws IOException {

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

    Board board = new Board();
    board.setTitle(title);
    board.setWriter(writer);
    board.setContent(content);
    board.setPass(pass);
```

```
/* 업로드한 Multipart 데이터(파일)에 접근하기
 * 스프링은 MultipartResolver를 사용해 멀티파트 데이터에
 * 접근할 수 있는 아래와 같은 다양한 방법을 제공하고 있다.
 *
 * MultipartFile 인터페이스를 이용한 접근
 * @RequestParam 애노테이션을 이용해 MultipartFile에 접근
 * MultipartHttpServletRequest를 이용해 MultipartFile에 접근
 * 커맨드 객체를 이용한 접근(커맨드 객체에 MultipartFile 타입의 프로퍼티가 있어함)
 * 서블릿 3.0의 Part를 이용한 접근
 *
 * 이 예제는 @RequestParam 애노테이션을 이용해 MultipartFile에 접근하는
 * 파일 업로드 방법을 소개하고 있다.
 **/
if(!multipartFile.isEmpty()) { // 업로드된 파일 데이터가 존재하면

    // Request 객체를 이용해 파일이 저장될 실제 경로를 구한다.
    String filePath =
            request.getServletContext().getRealPath(DEFAULT_PATH);

    /* UUID(Universally Unique Identifier, 범용 고유 식별자)
     * 소프트웨어 구축에 쓰이는 식별자의 표준으로 네트워크상에서 서로 모르는
     * 개체들을 식별하고 구별하기 위해서 사용된다. UUID 표준에 따라 이름을
     * 부여하면 고유성을 완벽하게 보장할 수는 없지만 실제 사용상에서 중복될
     * 가능성이 거의 없다고 인정되기 때문에 실무에서 많이 사용되고 있다.
     *
     * 파일 이름의 중복을 막고 고유한 파일 이름으로 저장하기 위해 java.util
     * 패키지의 UUID 클래스를 이용해 랜덤한 UUID 값을 생성한다.
     **/
    UUID uid = UUID.randomUUID();
    String saveName =
            uid.toString() + "_" + multipartFile.getOriginalFilename();

    File file = new File(filePath, saveName);
    System.out.println("insertBoard - newName : " + file.getName());

    // 업로드 되는 파일을 upload 폴더로 저장한다.
    multipartFile.transferTo(file);

    /* 아래와 같이 스프링이 지원하는 FileCopyUtils 클래스를
     * 이용해 업로드 되는 파일을 upload 폴더로 저장할 수 있다.
     **/
    //byte[] in = multipartFile.getBytes();
    //FileCopyUtils.copy(in, file);

    // 업로드된 파일 명을 Board 객체에 저장한다.
```

```
            board.setFile1(saveName);
        }

        /* BoardService 클래스를 이용해
         * 폼에서 넘어온 게시 글 정보를 게시 글 테이블에 추가한다.
         **/
        boardService.insertBoard(board);


        /* 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 주소를 지정한다.
         **/
        return "redirect:boardList";
    }

    /* 게시 글쓰기 폼에서 들어오는 게시 글쓰기 요청을 처리하는 메서드
     * MultipartHttpServletRequest를 이용해 MultipartFile에 접근하기
     *
     * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
     * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
     * 참고하기 바란다.
     */
    @RequestMapping(value="/addProcess", method=RequestMethod.POST)
    public String addBoard(MultipartHttpServletRequest request)
            throws IOException {

        /* MultipartHttpServletRequest 객체를 사용하는 것도 업로드된
         * 파일에 접근하기 위해서는 MultipartFile을 이용해 접근해야 한다.
         **/
        MultipartFile multipartFile = request.getFile("file1");
        System.out.println("originName : " + multipartFile.getOriginalFilename());
        System.out.println("name : " + multipartFile.getName());

        Board board = new Board();
        board.setTitle(request.getParameter("title"));
        board.setWriter(request.getParameter("writer"));
        board.setContent(request.getParameter("content"));
        board.setPass(request.getParameter("pass"));

        /* 업로드한 Multipart 데이터(파일)에 접근하기
```

```
 * 스프링은 MultipartResolver를 사용해 멀티파트 데이터에
 * 접근할 수 있는 아래와 같은 다양한 방법을 제공하고 있다.
 *
 * MultipartFile 인터페이스를 이용한 접근
 * @RequestParam 애노테이션을 이용해 MultipartFile에 접근
 * MultipartHttpServletRequest를 이용해 MultipartFile에 접근
 * 커맨드 객체를 이용한 접근(커맨드 객체에 MultipartFile 타입의 프로퍼티가 있어함)
 * 서블릿 3.0의 Part를 이용한 접근
 *
 * 이 예제는 MultipartHttpServletRequest를 이용해 MultipartFile에 접근하는
 * 파일 업로드 방법을 소개하고 있다.
 **/
if(!multipartFile.isEmpty()) { // 업로드된 파일 데이터가 존재하면

    // Request 객체를 이용해 파일이 저장될 실제 경로를 구한다.
    String filePath =
            request.getServletContext().getRealPath(DEFAULT_PATH);

    /* UUID(Universally Unique Identifier, 범용 고유 식별자)
     * 소프트웨어 구축에 쓰이는 식별자의 표준으로 네트워크 상에서 서로 모르는
     * 개체들을 식별하고 구별하기 위해서 사용된다. UUID 표준에 따라 이름을
     * 부여하면 고유성을 완벽하게 보장할 수는 없지만 실제 사용상에서 중복될
     * 가능성이 거의 없다고 인정되기 때문에 실무에서 많이 사용되고 있다.
     *
     * 파일 이름의 중복을 막고 고유한 파일 이름으로 저장하기 위해 java.util
     * 패키지의 UUID 클래스를 이용해 랜덤한 UUID 값을 생성한다.
     **/
    UUID uid = UUID.randomUUID();
    String saveName =
            uid.toString() + "_" + multipartFile.getOriginalFilename();

    File file = new File(filePath, saveName);
    System.out.println("addBoard - newName : " + file.getName());

    // 업로드한 파일을 upload 폴더로 저장한다.
    multipartFile.transferTo(file);

    // 업로드된 파일 명을 Board 객체에 저장한다.
    board.setFile1(saveName);
}

/* BoardService 클래스를 이용해
 * 폼에서 넘어온 게시 글 정보를 게시 글 테이블에 추가한다.
 **/
boardService.insertBoard(board);
```

```
        return "redirect:boardList";
    }

    ... 중략 ...

}
```

3-2) 파일 다운로드

파일 다운로드는 게시 글 상세보기에서 링크 방식으로 Controller에 다운로드를 요청하는 형태로 구현할 것이다. Controller에 다운로드를 요청할 때 어떤 파일을 원하는지 알려줘야 하는데 이때 파일 이름을 파라미터로 전송해 다운로드를 요청할 것이다.
브라우저는 데이터를 서버로 전송할 때 웹 페이지의 ContentType 속성에 지정된 문자 셋과 동일하게 URLEncoding을 처리한 후 데이터를 전송한다. 그러므로 요청을 보내는 페이지와 요청을 받는 컨트롤러에서 동일한 문자 셋으로 처리될 수 있도록 주의를 기울여야 한다.

▶ 게시 글 상세보기 Controller 클래스

게시 글 상세보기를 처리하는 BoardController 클래스의 boardDetail() 메서드는 **앞의 프로젝트와 동일하기 때문에 코드는 생략했다.**

▶ 게시 글 상세보기 View

게시 글 상세보기 뷰 페이지도 파일 다운로드를 링크로 설정하는 부분을 제외하고는 앞의 프로젝트와 동일하므로 아래의 코드를 참고해 파일을 다운로드 할 수 있도록 링크 부분을 수정하자.

- src/main/webapp/WEB-INF/views/boardDetail.jsp

```
    ... 중략 ...

    <tr>
        <th>파    일</th>
        <td colspan="3">
        <c:if test="${ empty board.file1 }">
            첨부파일 없음
        </c:if>
        <c:if test="${ not empty board.file1 }">
            <a href="fileDownload?fileName=${ board.file1 }">파일 다운로드</a>
        </c:if>
        </td>
    </tr>
```

... 중략 ...

▶ 게시 글 다운로드 Controller 클래스

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

- com.springstudy.bbs.controller.BoardController

... 중략 ...

```
// 게시 글 상세보기에서 들어오는 파일 다운로드 요청을 처리하는 메서드
@RequestMapping("/fileDownload")
public void download(HttpServletRequest request,
        HttpServletResponse response) throws Exception {

    String fileName = request.getParameter("fileName");
    System.out.println("fileName : " + fileName);

    String filePath =
            request.getServletContext().getRealPath(DEFAULT_PATH);

    File file = new File(filePath, fileName);
    System.out.println("file.getName() : " + file.getName());

    // 응답 데이터에 파일 다운로드 관련 컨텐츠 타입 설정이 필요하다.
    response.setContentType("application/download; charset=UTF-8");
    response.setContentLength((int) file.length());

    // 한글 파일명을 클라이언트로 바로 내려 보내기 때문에 URLEncoding이 필요하다.
    fileName = URLEncoder.encode(file.getName(), "UTF-8");
    System.out.println("다운로드 fileName : " + fileName);

    // 전송되는 파일 이름을 한글 그대(원본파일 이름 그대로)로 보내주기 위한 설정이다.
    response.setHeader("Content-Disposition",
            "attachment; filename=\"" + fileName + "\";");

    // 파일로 전송되야 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();
    }

    ... 중략 ...
```

5. 회원 가입 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 정보 수정 기능을 구현할 것이다. 또한 로그인을 하지 않으면 게시글 리스트만 볼 수 있고 검색을 하거나 게시 글 상세보기 같은 기능은 이용할 수 없도록 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 적용해야할 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 HandlerIntercepter를 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 사용한 springbbs, member 테이블을 수정 없이 그대로 사용하기 때문에 별도의 테이블 생성은 필요 없다. 또한 웹 애플리케이션 배포 서술자(web.xml)에 추가되는 설정도 없다.

1-1) 프로젝트 생성

이번 예제도 springstudy-bbs04 프로젝트를 복사해 springclass-bbs05 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 회원 가입, 회원 정보 수정, 그리고 Interceptor를 활용한 로그인 체크 기능을 구현할 것이다. 그렇기 때문에 프로젝트 생성과 설정 그리고 의존 라이브러리 설정 등에 대한 자세한 내용은 앞의 예제에서 설명한 주석을 참고 하길 바란다.

이번 예제도 앞에서 구현한 게시판에 회원 가입과 HandlerInterceptor를 활용해 로그인 체크 기능을 추가하는 것이므로 설정파일과 소스가 거의 동일하기 때문에 추가되거나 수정되는 부분은 붉은색 볼드체로 표시할 것이다.

- **실습용 프로젝트 : springclass-bbs05**
- **완성 프로젝트 : springstudy-bbs05**

1-2) 의존 라이브러리와 BuildPath 설정

Spring Legacy Project 메뉴를 이용해 Spring MVC Project를 생성하게 되면 자바 버전은 1.6, 스프링프레임워크 버전 3.1.1, Dynamic Web Module 2.5가 기본 설정되어 있다.

우리는 스프링프레임워크 버전 4.2.4 버전을 사용할 것이므로 Maven을 통해 버전을 변경해야 한다. 그리고 자바 1.8 버전과 Dynamic Web Module 3.0을 사용할 것이므로 Configure Build Path를

통해 Java Build Path의 자바 버전과 Java Compiler 버전을 1.8로 설정하고 Server Runtime 설정을 변경해야 한다. 그리고 프로젝트를 생성할 때 패키지의 3번째 단계에 지정한 bbs가 ContextRoot와 Artifact Id로 사용되기 때문에 이를 변경해야 할 필요도 있을 것이다. 또한 이번 프로젝트는 앞에서 작성한 모든 기능이 포함되어 있기 때문에 MyBatis와 스프링프레임워크를 연동하고 스프링 시큐리티의 PasswordEncoder를 이용해 사용자가 입력한 데이터와 암호화되어 저장된 데이터를 비교해 로그인을 처리하고 파일 업로드를 구현하는 예제가 포함되어 있으므로 mybatis 모듈과 mybatis-spring 모듈이 필요하고 spring-security-web 모듈과 commons-fileupload 라이브러리가 필요하다.
이 번 프로젝트의 회원 정보 수정 폼에서 기존의 비밀번호 확인을 위해 스프링이 지원하는 Ajax 요청 처리에 대해 간단히 알아 볼 것이다. 이를 위해서 스프링이 제공하는 자바 객체를 JSON 형식으로 변환 해 주는 MappingJackson2HttpMessageConverter라는 다소 이름이 긴 클래스를 사용할 것이다. 이 클래스를 사용하기 위해서 별도의 Bean 설정은 필요하지 않는다. 다만 Spring MVC 설정에서 <mvc:annotation-driven />을 설정하는 것으로 스프링이 위의 클래스를 스프링 Bean으로 자동 등록해 준다. 하지만 이 클래스는 Jackson2 라이브러리를 사용해 자바 객체를 JSON 형식으로 변환하기 때문에 pom.xml을 통해 Jackson2 라이브러리를 아래와 같이 의존설정 해야 한다.

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.8.5</version>
</dependency>
```

이에 대한 자세한 설명은 springstudy-bbs05 프로젝트에서 com.springstudy.bbs.ajax 패키지의 AjaxProcessController 클래스의 주석을 참고하길 바란다.

1-3) Spring MVC Bean 정의

Spring MVC Bean 정의 파일에 다음과 같이 회원 가입 폼 요청을 처리하는 뷰 전용 컨트롤러 설정을 추가한다. 로그인 여부를 체크하는 Interceptor 설정도 필요하지만 이 부분은 뒤에서 인터셉터를 설명할 때 추가할 것이다.
이외에 root-context.xml 파일과 spring-security.xml 파일은 예제와 동일하다.

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

```
<!--
    회원가입 폼 요청을 처리하는 뷰 전용 컨트롤러 설정
-->
<view-controller path="/joinForm" view-name="member/memberJoinForm" />
```

2) 회원 가입 구현

▶ 회원 가입 폼 View

아래의 코드를 참고해 회원 가입 폼을 작성해 보자.

- src/main/webapp/WEB-INF/views/member/memberJoinForm.jsp

```
<%-- 회원가입 폼 요청 처리 결과를 출력할 View 페이지 --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!--
    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="id" 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="중복확인">
                        </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="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>
            </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">* 이 메 일 : </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>
            <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>
                                <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="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>
            <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>
```

▶ 아이디 중복확인 View

아래의 코드를 참고해 아이디 중복확인 페이지를 작성해 보자.

- src/main/webapp/WEB-INF/views/member/overlapIdCheck.jsp

```
<%--
    회원 가입시 아이디 중복검사 요청에 대한 처리 결과를 출력할 View 페이지
    이 페이지는 새창으로 실행되고 중복 아이디 체크를 할 수 있는 폼을 제공한다.
--%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link href="resources/bootstrap/bootstrap.min.css" rel="stylesheet" >
<script type="text/javascript" src="resources/js/jquery-3.2.1.min.js"></script>
<script type="text/javascript" src="resources/js/member.js"></script>
<title>중복 아이디 체크</title>
</head>
<body>
    <div class="row my-5" id="global-content">
        <c:choose>
            <c:when test="${ overlap }" >
            <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>
            </c:when>
            <c:otherwise>
                <div class="col">
```

```
                    <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 ${id}는 사용할 수 있는 아이디 입니다.
                        </div>
                    </div>
                    <div class="row mt-5">
                        <div class="col  text-center">
                            <input type="button" value="${ id }을(를) 아이디로 사용하기"
                                   id="btnIdCheckClose" data-id-value="${ id }" class="btn
btn-primary"/>
                        </div>
                    </div>
                </div>
            </c:otherwise>
        </c:choose>
    </div>
</body>
</html>
```

▶ 자바스크립트

member.js 파일에 회원 가입 폼에서 데이터가 제대로 입력되었는지 유효성 검사를 수행하는 아래의 자바스크립트 코드를 추가한다.

- src/main/webapp/resources/js/member.js

```
// DOM이 준비되면 실행될 콜백 함수
$(function() {

    ... 앞부분 생략 ...

    /* 회원 가입 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 눌려질 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);


/* 회원 가입 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, location=no, "
            + "status=no, memubar=no, width=400, height=200");
});


/* 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 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();
    });
});

/* 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, ""));
    }
}

/* 이 메일 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, ""));
    }
}

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

```
 **/
function joinFormCheck() {
    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(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 분기 한다.
                var addr = ''; // 주소 변수
                var extraAddr = ''; // 참고 항목 변수

                //사용자가 선택한 주소 타입에 따라 해당 주소 값을 가져온다.
                // 사용자가 도로명 주소를 선택했을 경우
                if (data.userSelectedType === 'R') {
                    addr = data.roadAddress;

                } else { // 사용자가 지번 주소를 선택했을 경우(J)
                    // 모두 도로명 주소가 선택되도록 변경했음
                    addr = data.roadAddress;
                    //addr = data.jibunAddress;
                }

                // 사용자가 선택한 주소가 도로명 타입일때 참고 항목을 조합한다.
                if(data.userSelectedType === 'R'){
```

```
                // 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();
}
```

▶ DAO 계층 구현

com.springstudy.bbs.dao 패키지의 MemberDao 인터페이스에 회원 가입을 처리하는 추상 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.MemberDao

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

▶ 게시판 관련 SQL을 분리한 Mapper

MemberMapper.xml에 회원 가입을 처리하는 맵핑 구문을 추가한다.

- src/main/resources/repository/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>
```

▶ MemberDao 인터페이스 구현 클래스

MemberDaoImpl 클래스에 회원 가입을 처리하는 아래와 같은 메서드를 추가한다.

- com.springstudy.bbs.service.MemberDaoImpl

```
// 회원 정보를 회원 테이블에 저장하는 메서드
@Override
public void addMember(Member member) {
    sqlSession.insert(NAME_SPACE + ".addMember", member);
}
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 MemberService 인터페이스에 회원 가입을 처리하는 아래 메서드를 추가한다.

- com.springstudy.bbs.service.MemberService

```
// 회원 가입시 DAO를 이용해 아이디 중복을 체크하는 메서드
public boolean overlapIdCheck(String id);

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

▶ MemberService 인터페이스 구현 클래스

MemberServiceImpl 클래스에 회원 아이디 중복을 체크하는 메서드와 회원 가입을 처리하는 아래와 같은 메서드를 추가한다. 회원 가입을 처리하는 메서드를 살펴보면 회원이 입력한 원본 문자열을 스프링 시큐리티가 제공하는 BCrytPasswordEncoder 객체를 이용해 암호화하여 저장하는 코드를 볼 수 있다.

- com.springstudy.bbs.service.MemberServiceImpl

```
... 앞부분 생략 ...

// 회원 가입시 아이디 중복을 체크하는 메서드
@Override
```

```
    public boolean overlapIdCheck(String id) {
        Member member = memberDao.getMember(id);
        System.out.println("overlapIdCheck - member : " + member);
        if(member == null) {
            return false;
        }
        return true;
    }

    // 회원 정보를 DAO를 이용해 회원 테이블에 저장하는 메서드
    @Override
    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/VzgQx697lYlZgQxr/pe
        // $2a$10$b3t8sn6QZGHYaRx3OS5KUuPxzWZdY5yHPRxlSdAgByQ7v0BlCLzrO
        // $2a$10$.g6l.wyIFO1.j4u4gvVtKOnG9ACBUT1GRlDwlMZcjBxZPrCAURLaG
        // $2a$10$l37iiJWozST9.2EI11vjqOmSk9rus.5cawhTiPuQagCzVNTZcoFDa
        // $2a$10$qtxKvIgk.URhyqgx93KYFuw4e8Rh3IhIkR0CTZhd.HazLal7d3rqK
        System.out.println(member.getPass());
        memberDao.addMember(member);
    }
```

▶ Controller 클래스

com.springstudy.bbs.controller.MemberController 클래스에 회원 가입 폼에서 들어오는 요청을 처리하는 메서드를 추가한다. 그리고 회원 가입 폼에서 회원 아이디 중복확인 버튼이 클릭되면 회원 아이디 중복에 대한 요청을 처리해 주는 메서드도 추가해야 한다.

- com.springstudy.bbs.controller.MemberController

```
// 스프링 MVC의 컨트롤러임을 선언하고 있다.
@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 {

... 앞부분 생략 ...

    /* 회원가입 폼에서 들어오는 요청을 처리하는 메서드
     * 아래는 "/joinResult"로 들어오는 GET 방식 요청을 처리하는 메서드를 지정한
     * 것이다. method 속성을 생략하면 GET 방식과 POST 방식 모두를 처리할 수 있다.
     *
     * 실제 memberJoinForm.jsp에서 폼을 전송할 때 POST 방식으로 폼을 전송
     * 했지만 아래의 joinResult() 메서드에서 정상적으로 처리된다.
     * method 속성이 생략되면 스프링은 GET 방식 요청을 먼저 적용해 처리 하지만
     * POST 방식의 요청도 처리해 준다.
     *
     * 만약 아래에서 method=RequestMethod.GET 를 명시적으로 지정했거나
     * /joinInfo 요청을 처리하는 메서드가 존재하지 않는다면 아래와 같이 405
     * 익셉션이 발생하게 된다.
     *
     * HTTP Status 405 - Request method 'POST' not supported
     *
     * 스프링이 자동으로 처리해 주긴 하지만 GET 방식과 POST 방식을 명확하게
     * 처리하는 것이 바람직한 코딩 방법이다.
     **/
    @RequestMapping("/joinResult")
    public String joinResult(Model model, Member member,
            String pass1, String emailId, String emailDomain,
            String mobile1, String mobile2, String mobile3,
            String phone1, String phone2, 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);
        System.out.println("joinResult : " + member.getName());
```

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

    // 회원가입 폼에서 들어오는 중복 아이디 체크 요청을 처리하는 메서드
    @RequestMapping("/overlapIdCheck")
    public String overlapIdCheck(Model model, String id) {

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

        // model에 id와 회원 아이디 중복 여부를 저장 한다.
        model.addAttribute("id", id);
        model.addAttribute("overlap", overlap);

        /* 회원 가입 폼에서 아이디 중복확인 버튼을 클릭하면 새창으로 뷰가 보이게
         * 해야 하므로 뷰 이름을 반환 할 때 "forward:" 접두어를 사용했다.
         * "forwrad:" 접두어가 있으면 뷰 리졸버 설정에 지정한 prefix, suffix를
         * 적용하지 않고 "forwrad:" 뒤에 붙인 뷰 페이지로 포워딩 된다.
         **/
        return "forward:WEB-INF/views/member/overlapIdCheck.jsp";
    }
}
```

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. 이때 비밀번호 수정에 대한 부분은 기존의 비밀번호를 입력하여 "비밀번호확인"을 하게 되는데 이 기능을 Ajax로 구현할 것이다.

▶ DAO 계층 구현

com.springstudy.bbs.dao 패키지의 MemberDao 인터페이스에 회원 정보 수정과 관련된 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.MemberDao

```
// 회원 정보 수정 시에 기존 비밀번호가 맞는지 체크하는 메서드
public String memberPassCheck(String id);

// 회원 정보를 회원 테이블에서 수정하는 메서드
public void updateMember(Member member);
```

▶ 게시판 관련 SQL을 분리한 Mapper

MemberMapper.xml에 회원 정보 수정과 관련된 맵핑 구문을 추가한다.

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

```
<!--
    회원 테이블에서 id에 해당하는 비밀번호를 가져오는 맵핑 구문

    아래는 DAO 클래스의 memberPassCheck(int no, String pass)
    메서드에서 사용하는 맵핑 구문으로 DAO에서 회원 id에 해당하는 비밀번호를
    조회할 때 selectOne() 메서드의 두 번째 인수로 기본형인 id를 지정했기
    때문에 parameterType은 생략할 수 있다.

    SQL문의 조건에 사용할 파라미터는 아래와 같이 #{} 로 감싸서 지정하면 된다.
-->
<select id="memberPassCheck" resultType="String">
    SELECT
        pass
    FROM member
    WHERE id = #{id}
</select>

<!--
    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>
```

▶ MemberDao 인터페이스 구현 클래스

MemberDaoImpl 클래스에 회원 정보 수정과 관련된 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.MemberDaoImpl

```
    // 회원 정보 수정 시에 기존 비밀번호가 맞는지 체크하는 메서드
    public String memberPassCheck(String id) {

        // memberPassCheck 맵핑 구문을 호출하면서 회원 아이디를 파라미터로 지정했다.
        return sqlSession.selectOne(NAME_SPACE + ".memberPassCheck",    id);
    }

    // 회원 정보를 DAO를 이용해 회원 테이블에서 수정하는 메서드
    public void updateMember(Member member) {
        sqlSession.update(NAME_SPACE + ".updateMember", member);
    }
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 MemberService 인터페이스에 회원 정보 수정을 처리하는 아래 메서드를 추가한다.

- com.springstudy.bbs.service.MemberService

```
    // 회원 정보 수정 시에 기존 비밀번호가 맞는지 체크하는 메서드
    public boolean memberPassCheck(String id, String pass);

    // 회원 정보를 DAO를 이용해 회원 테이블에서 수정하는 메서드
    public void updateMember(Member member);
```

▶ MemberService 인터페이스 구현 클래스

MemberServiceImpl 클래스에 회원 정보 수정을 처리하는 메서드를 아래와 같이 추가한다. memberPassCheck() 메서드의 코드를 살펴보면 BCyrtPasswordEncoder 객체의 matches() 메서드를 이용해 사용자가 입력한 비밀번호와 DB 테이블에 암호화되어 저장된 비밀번호를 비교해서 맞은면 true를 틀리면 false를 반환하는 코드를 볼 수 있을 것이다. 또한 회원 정보를 수정하는

updateMember() 메서드에서는 사용자가 새롭게 입력한 새 비밀번호를 암호화하여 저장하는 부분도 확인할 수 있다.

- com.springstudy.bbs.service.MemberServiceImpl

```
// 회원 정보 수정 시에 기존 비밀번호가 맞는지 체크하는 메서드
public boolean memberPassCheck(String id, String pass) {

    String dbPass = memberDao.memberPassCheck(id, pass);
    boolean result = false;

    /* 비밀번호가 맞으면 true를 반환하도록 작성한다.
     * BCryptPasswordEncoder 객체의 matches 메소드를 이용해 암호가 맞는지 확인
     * matches() 메소드의 첫 번 인수로 인코딩이 않된 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;
}

// 회원 정보를 DAO를 이용해 회원 테이블에서 수정하는 메서드
public void updateMember(Member member) {

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

    memberDao.updateMember(member);
}
```

▶ Controller 클래스

com.springstudy.bbs.controller 패키지의 MemberController 클래스에 회원 정보 수정 폼 요청을 처리하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.controller.MemberController

```
// 회원 정보 수정 폼 요청을 처리하는 메서드
@RequestMapping("/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(새로 작성)

- src/main/webapp/WEB-INF/views/member/memberUpdateForm.jsp

```
<%-- 회원정보 수정 폼 요청 처리 결과를 출력할 View 페이지 --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!--
    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">* 이 름 : </label>
                    <input type="text" class="form-control" name="name" id="name"
                        value="${sessionScope.member.name}" readonly>
                </div>
            </div>
            <div class="row my-3">
                <div class="col-8 offset-2">
                    <label for="userId" class="form-label">* 아이디 : </label>
                    <div class="row">
                        <div class="col">
                            <input type="text" class="form-control" name="id" id="userId"
                                value="${sessionScope.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>
            <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="password"  class="form-control"  name="zipcode"
id="zipcode"
                                   maxlength="5"                                    readonly
value="${sessionScope.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 value="${sessionScope.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"
                            value="${sessionScope.member.address2}">
                    </div>
                </div>
                <div class="row my-3">
                    <div class="col-8 offset-2">
                        <label for="emailId" class="form-label">* 이 메 일 : </label>
                        <div class="row">
                            <div class="col-md-4">
                                <input type="text" class="form-control" name="emailId"
id="emailId"
                                    value="${sessionScope.member.email.split('@')[0]}">
                            </div> @
                            <div class="col-md-4">
                                <input type="text" class="form-control" name="emailDomain"
id="emailDomain"
                                    value="${sessionScope.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>
                <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 ${ member.mobile.split('-')[0] == 010 ? "selected" :
"" }>
                                    010</option>
                                <option ${ member.mobile.split('-')[0] == 011 ? "selected" :
"" }>
                                    011</option>
                                <option ${ member.mobile.split('-')[0] == 016 ? "selected" :
"" }>
                                    016</option>
                                <option ${ member.mobile.split('-')[0] == 017 ? "selected" :
"" }>
                                    017</option>
                                <option ${ member.mobile.split('-')[0] == 018 ? "selected" :
"" }>
                                    018</option>
                                <option ${ member.mobile.split('-')[0] == 019 ? "selected" :
"" }>
                                    019</option>
                            </select>
                        </div>-
                        <div class="col-md-4">
                            <input     type="text"     class="form-control"     name="mobile2"
id="mobile2" maxlength="4"
                                value="${ sessionScope.member.mobile.split('-')[1] }">
                        </div>-
                        <div class="col-md-4">
                            <input     type="text"     class="form-control"     name="mobile3"
id="mobile3" maxlength="4"
                                value="${ sessionScope.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" ${ member.emailGet ? "checked" : "" }>
```

```
                                <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" ${ member.emailGet ? "" : "checked" }>
                                <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 ${ member.phone.split('-')[0] == 02 ? "selected" :
"" }>
                                    02</option>
                                <option ${ member.phone.split('-')[0] == 031 ? "selected" :
"" }>
                                    031</option>
                                <option ${ member.phone.split('-')[0] == 032 ? "selected" :
"" }>
                                    032</option>
                                <option ${ member.phone.split('-')[0] == 033 ? "selected" :
"" }>
                                    033</option>
                                <option ${ member.phone.split('-')[0] == 041 ? "selected" :
"" }>
                                    041</option>
                                <option ${ member.phone.split('-')[0] == 042 ? "selected" :
"" }>
                                    042</option>
                                <option ${ member.phone.split('-')[0] == 043 ? "selected" :
"" }>
                                    043</option>
                                <option ${ member.phone.split('-')[0] == 044 ? "selected" :
"" }>
```

```
                    044</option>
                <option ${ member.phone.split('-')[0] == 051 ? "selected" :
"" }>
                    051</option>
                <option ${ member.phone.split('-')[0] == 052 ? "selected" :
"" }>
                    052</option>
                <option ${ member.phone.split('-')[0] == 053 ? "selected" :
"" }>
                    053</option>
                <option ${ member.phone.split('-')[0] == 054 ? "selected" :
"" }>
                    054</option>
                <option ${ member.phone.split('-')[0] == 055 ? "selected" :
"" }>
                    055</option>
                <option ${ member.phone.split('-')[0] == 061 ? "selected" :
"" }>
                    061</option>
                <option ${ member.phone.split('-')[0] == 062 ? "selected" :
"" }>
                    062</option>
                <option ${ member.phone.split('-')[0] == 063 ? "selected" :
"" }>
                    063</option>
                <option ${ member.phone.split('-')[0] == 064 ? "selected" :
"" }>
                    064</option>
                <option ${ member.phone.split('-')[0] == 010 ? "selected" :
"" }>
                    010</option>
                <option ${ member.phone.split('-')[0] == 011 ? "selected" :
"" }>
                    011</option>
                <option ${ member.phone.split('-')[0] == 016 ? "selected" :
"" }>
                    016</option>
                <option ${ member.phone.split('-')[0] == 017 ? "selected" :
"" }>
                    017</option>
                <option ${ member.phone.split('-')[0] == 018 ? "selected" :
"" }>
                    018</option>
                <option ${ member.phone.split('-')[0] == 019 ? "selected" :
"" }>
```

```
                                019</option>
                            </select>
                        </div> -
                        <div class="col-md-4">
                            <input     type="text"     class="form-control"     name="phone2"
id="phone2"
                                maxlength="4" value="${ member.phone.split('-')[1] }">
                        </div> -
                        <div class="col-md-4">
                            <input     type="text"     class="form-control"     name="phone3"
id="phone3"
                                maxlength="4" value="${ 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>
```

▶ 자바스크립트

member.js 파일의 DOM이 준비되었을 때 동작하는 $(function() { } 함수 안에 다음과 같은 이벤트 처리 코드를 추가로 작성한다.

- src/main/webapp/resources/js/member.js 에 추가

```
    /* 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": "method",
            "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 다릅니다.\n비밀번호를 다시 확인해주세요");
                    $("#oldPass").val("").focus();
                }
            },
            "error": function() {
                console.log("error");
            }
        });
    });

    // 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;
        }

        /* joinFormChcek() 함수에서 폼 유효성 검사를 통과하지 못하면
         * false가 반환되기 때문에 그대로 반환하면 폼이 서브밋 되지 않는다.
         **/
        return joinFormCheck();
    });
});
```

▶ Ajax 요청처리 Controller 클래스

프로젝트에 com.springstudy.bbs.ajax 패키지를 추가하고 이 패키지에 Ajax 요청을 처리하는

AjaxProcessController 클래스 새롭게 추가한다. 그리고 회원 정보 수정 폼에서 회원의 기존 비밀번호가 맞는지 확인하는 Ajax 요청이 들어오면 이를 처리하는 memberPassCheck() 메서드를 AjaxProcessController 클래스에 다음과 같이 추가한다.

- com.springstudy.bbs.ajax.AjaxProcessController

```
// Ajax 요청을 처리하는 컨트롤러
@Controller
public class AjaxProcessController {

    @Autowired
    private MemberService memberService;

    /* 스프링 mvc에서 xml이나 json 형식의 응답 데이터를 만들려면
     * xml 또는 json 형식의 응답을 생성하는 뷰 클래스를 사용하거나
     * HttpServletResponse 객체를 직접 사용해 필요한 응답 데이터를
     * 생성할 수있다.
     * 스프링 mvc가 지원 하는 아래와 같은 애노테이션을 사용하면 text, xml,
     * json 형식의 요청 데이터를 자바 객체로 변환해 주거나 자바 객체를 text,
     * xml, json 형식으로 변환해 응답 본문에 추가해 준다.
     *
     * @RequestBody
     * 클라이언트가 요청할 때 요청 본문으로 넘어오는 데이터를 자바 객체로 변환할
     * 때 사용한다. 예를 들면 post 방식 요청에서 본문에 실어 넘어오는 파라미터를
     * 자바의 String으로 변환하거나 json 형식의 데이터를 자바 객체로 변환하기
     * 위해 사용한다.
     *
     * @ResponseBody
     * 자바 객체를 json 형식이나 xml 형식의 문자열로 변환하여 응답 본문에 실어
     * 보내기 위해 사용한다.
     * 컨트롤러 메서드에 @ResponseBody가 적용되면 메서드의 반환 값은 스프링
     * mvc에 의해서 Http 응답 본문에 포함 된다.
     *
     *
     * MappingJackson2HttpMessageConverter를 이용한 json 응답처리
     *
     * 요청 본문의 json 형식의 데이터를 자바 객체로 변환해 주거나
     * 자바 객체를 응답 본문의 json 형식의 데이터로 변환해 준다.
     * 지원하는 요청 컨텐츠 타입은 아래와 같다.
     * application/json, application/*+json
     *
     * 참고사이트 : https://github.com/FasterXML
     * http://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/
     *
     * 스프링이 제공하는 MappingJackson2HttpMessageConverter는
     * Jackson2 라이브러리를 이용해서 자바 객체와 json 데이터를 변환하기
```

```
     * 때문에 pom.xml에서 "jackson-databind"으로 검색해 아래와 같이
     * Jackson2 라이브러리 의존 설정을 해야 한다.
     * 그렇지 않으면 json 타입의 요청을 자바 객체로 변환할 수 없기 때문에
     * 415 Unsupported Media Type 응답을 받게 된다.
     *
     * <dependency>
     *       <groupId>com.fasterxml.jackson.core</groupId>
     *           <artifactId>jackson-databind</artifactId>
     *           <version>2.8.5</version>
     *  </dependency>
     *
     * 스프링 빈 설정 파일에 HttpMessageConvert 인터페이스를
     * 구현한 MappingJackson2HttpMessageConverter가 빈으로
     * 등록되어 있어야 응답 본문으로 들어오는 json 형식의 데이터를 자바 객체로
     * 변환하거나 자바 객체를 json 형식의 데이터로 변환해 응답 본문에 포함 시킬
     * 수 있다. 하지만 스프링 빈 설정 파일에 <mvc:annotation-driven />이
     * 적용되었기 때문에 MappingJackson2HttpMessageConverter를
     * 포함한 다수의 HttpMessageConverter 구현체를 빈으로 등록해 준다.
     *
     * @RequestMapping 애노테이션이 적용된 Controller의 메서드에
     * 아래와 같이 @ResponseBody를 적용하고 Map 객체를 반환하면
     * MappingJackson2HttpMessageConverter에 의해서 JSON
     * 형식으로 변환된다.
     **/
    @RequestMapping("/passCheck.ajax")
    @ResponseBody
    public Map<String, Boolean> memberPassCheck(String id, String pass) {

        /* 요청 파라미터로 전달 받은 id와 pass를 입력해
         * 비밀번호가 맞는지 여부를 JSON 형식으로 변환해 응답 본문에 추가한다.
         **/
        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;
    }
}
```

▶ Controller 클래스

com.springstudy.bbs.controller 패키지의 MemberController 클래스에 회원 정보 수정 폼에서 들어오는 수정 요청을 처리하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.controller.MemberController

```
// 회원 수정 폼에서 들어오는 요청을 처리하는 메서드
@RequestMapping("/memberUpdateResult")
public String memberUpdateInfo(Model model, Member member,
        String pass1, String emailId, String emailDomain,
        String mobile1, String mobile2, String mobile3,
        String phone1, String phone2, 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);
    System.out.println("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);

    // 게시 글 리스트로 리다이렉트 시킨다.
    return "redirect:boardList";
}
```

4) HandlerInterceptor를 이용한 로그인 체크 기능 구현

이번 프로젝트에는 앞에서 언급한 것처럼 로그인을 하지 않으면 게시 글 리스트만 볼 수 있고 검색을 하거나 게시 글 상세보기 같은 기능은 이용할 수 없도록 구현할 것이다. 앞에서는 회원이 로그인 상태인지를 체크하기 위해서 컨트롤러 메서드마다 세션을 체크해 로그인 상태를 체크해야 했기 때문에 로그인을 체크하는 동일한 코드의 중복이 많이 발생했다. 이를 스프링프레임워크가 제공하는 HandlerInterceptor를 구현해 해결할 것이다. 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 마다 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 처리해야할 기능이 필요하다면 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 전체를 구현하지 않아도 될 때는 HandlerInterceptor를 구현하였지만 메서드의 내용이 없는 HandlerInterceptorAdapter를 상속 받아 필요한 메서드만 구현 할 수 있다.
HandlerInterceptor를 구현하고 동작시키기 위해서는 SpringMVC 설정에 Bean으로 등록하고 <interceptors> 태그를 사용해 이 인터셉터가 동작할 경로 매핑을 해야 한다.

▶ HandlerInterceptor 클래스

프로젝트에 com.springstudy.bbs.interceptor 패키지를 추가하고 이 패키지에 로그인 여부를 체크하는 HandlerInterceptor를 상속한 LoginCheckInterceptor 클래스 추가한다.

- com.springstudy.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 전체를
```

```
 * 구현하지 않아도 될 때는 HandlerInterceptor를 구현하였지만 메서드의 내용이
 * 없는 HandlerInterceptorAdapter를 상속 받아 필요한 메서드만 구현 할 수 있다.
 *
 * HandlerInterceptor를 구현하고 동작시키기 위해서는 SpringMVC 설정에
 * Bean으로 등록하고 <interceptors> 태그를 사용해 이 인터셉터가 동작할
 * 경로 매핑을 해야 제대로 동작한다.
 **/
// 접속자가 로그인 상태인지 체크하는 인터셉터
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 {

        // 세션에 isLogin란 이름의 데이터가 없으면 로그인 상태가 아님
        if(request.getSession().getAttribute("isLogin") == null) {
            response.sendRedirect("loginForm");
            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 {
    }

    /* 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 {
    }
}
```

▶ Spring MVC Bean 정의

앞에서 구현한 HandlerInterceptor가 제대로 동작하기 위해서는 Spring MVC Bean 정의 파일에 아래와 같이 스프링이 관리하는 Bean으로 등록해야 한다.

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

```
    <!-- 로그인 체크 인터셉터 설정 -->
    <!--
        HandlerInterceptor는 아래와 같이 별도의 Bean으로 등록하고 <interceptors>
        태그 안에서 <interceptor> 태그를 통해 경로 매핑과 Bean 참조를 설정할 수 있다.
        또한 <interceptor> 태그 안에서 경로 매핑과 Bean 설정을 같이 할 수도 있다.
    -->
    <beans:bean id="loginCheckInterceptor"
        class="com.springstudy.bbs.interceptor.LoginCheckInterceptor" />
    <interceptors>
        <interceptor>
            <mapping path="/boardDetail" />
            <mapping path="/add*" />
            <mapping path="/write*" />
            <mapping path="/update*" />
            <mapping path="/memberUpdate*" />
            <beans:ref bean="loginCheckInterceptor" />
        </interceptor>
        <!--
        <interceptor>
            <mapping path="/member/*" />
            <mapping path="/board/*" />
            <beans:bean class=
                "com.springstudy.bbs.interceptor.LoginCheckInterceptor" />
        </interceptor>
        -->
    </interceptors>
```

6. 게시 글 상세보기의 추천/땡큐 및 댓글 Ajax 구현

이번 예제는 맘에 드는 게시 글을 추천하거나 자료실 게시판이나 좋은 자료를 올려서 고맙다는 의사를 업로더에게 표시할 수 있는 땡큐 기능을 추가해 볼 것이다.
대부분의 추천/땡큐 기능은 게시 글 상세보기에서 이루어지는데 사용자가 추천이나 땡큐를 클릭했을 때 게시 글 상세보기 페이지 전체를 새롭게 갱신하는 것 보다 추천이나 땡큐가 표시되는 부분만 갱신하는 것이 네트워크 트래픽도 줄이고 페이지 이동 없이 서비스 할 수 있다는 장점이 있다. 그래서 이번 추천/땡큐와 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/땡큐 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. 또한 웹 애플리케이션 배포 서술자(web.xml)에 추가되는 설정도 없고 Spring MVC Bean 정의파일(servlet-context.xml)과 Spring Bean 정의파일(root-context.xml)에 추가되는 설정도 없다.

1-1) 프로젝트 생성

이번 예제도 springstudy-bbs05 프로젝트를 복사해 springclass-bbs06 프로젝트를 만들고 프로젝트에 필요한 라이브러리 의존성을 설정한 후 게시 글 상세보기 페이지에서 추천/땡큐 기능과 댓글 기능을 Ajax로 구현할 것이다.

- **실습용 프로젝트 : springclass-bbs06**
- **완성 프로젝트 : springstudy-bbs06**

1-2) DB TABLE 생성 및 데이터 입력

먼저 springstudy-bbs06 프로젝트의 src/main/resources/SQL/springbb.sql을 참고해 데이터베이스에서 springbbs 테이블을 수정하고 reply 테이블 생성하자.

1-3) 의존 라이브러리와 BuildPath 설정

Spring Legacy Project 메뉴를 이용해 Spring MVC Project를 생성하게 되면 자바 버전은 1.6,

스프링프레임워크 버전 3.1.1, Dynamic Web Module 2.5가 기본 설정되어 있다.
우리는 스프링프레임워크 버전 4.2.4 버전을 사용할 것이므로 Maven을 통해 버전을 변경해야 한다. 그리고 자바 1.8 버전과 Dynamic Web Module 3.0을 사용할 것이므로 Configure Build Path를 통해 Java Build Path의 자바 버전과 Java Compiler 버전을 1.8로 설정하고 Server Runtime 설정을 변경해야 한다. 그리고 프로젝트를 생성할 때 패키지의 3번째 단계에 지정한 bbs가 ContextRoot와 Artifact Id로 사용되기 때문에 이를 변경해야 할 필요도 있을 것이다. 또한 이번 프로젝트는 앞에서 작성한 모든 기능이 포함되어 있기 때문에 MyBatis와 스프링프레임워크를 연동하고 다중 파일 업로드를 구현하는 예제가 포함되어 있으므로 mybatis 모듈과 mybatis-spring 모듈이 필요하고 commons-fileupload 라이브러리가 필요하다.
이 번 프로젝트의 추천/땡큐와 댓글 기능은 스프링이 지원하는 Ajax 요청 처리 방식을 사용할 것이다. 이를 위해서 스프링이 제공하는 MappingJackson2HttpMessageConverter라는 다소 이름이 긴 클래스를 사용할 것이다. 이 클래스는 자바 객체를 JSON 형식으로 변환 해 주는 클래스로 별도의 설정은 필요 없다. 다만 Spring MVC 설정(servlet-context.xml)에서 <mvc:annotation-driven /> 을 설정하는 것으로 스프링이 위의 클래스를 스프링 Bean으로 자동 등록해 준다. 하지만 이 클래스는 Jackson2 라이브러리를 사용해 자바 객체를 JSON 형식으로 변환하기 때문에 pom.xml을 통해 Jackson2 라이브러리를 아래와 같이 의존설정 해야 한다.

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.8.5</version>
</dependency>
```

이에 대한 자세한 설명은 springstudy-bbs06 프로젝트에서 com.springstudy.bbs.ajax 패키지의 AjaxProcessController 클래스의 주석을 참고하길 바란다.

2) 게시 글 상세보기 페이지 댓글 리스트 구현

추천/땡큐 기능을 Ajax로 구현하기 위해서 앞에서 구현한 게시 글 상세보기 화면을 수정할 것이다. 이 게시 글 상세보기 화면에서 추천/땡큐 기능과 댓글 기능을 제공할 것이므로 추천/땡큐 버튼과 댓글 리스트를 출력하는 부분을 먼저 구현하고 추천/땡큐 기능을 Ajax로 구현하게 될 것이다. 게시 글 상세보기 화면을 구성할 때 게시 글의 추천/땡큐 수도 출력해야 하므로 기존에 사용하던 springbbs 테이블에 추천, 땡큐 수를 저장할 수 있는 컬럼을 추가하고 Board 클래스에 추천/땡큐를 저장하는 인스턴스 변수가 추가 되었다. 또한 게시 글의 댓글을 저장할 테이블인 reply 테이블이 추가되었기 때문에 댓글 하나를 저장할 수 있는 도메인 객체인 replay 클래스도 추가되었다. 아래를 참고해 Board 클래스를 수정하고 Reply 클래스를 추가하자.

▶ 게시 글 하나의 정보를 저장하는 Domain 클래스(수정)

- com.jspstudy.bbs.domain.Board

```
/* 하나의 게시 글 정보를 저장하는 클래스(Domain, VO, Beans, DTO)
 * 게시 글 정보를 저장하고 있는 테이블의 필드와 1:1 맵핑되는 Domain 클래스
 **/
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;

    public Board() { }
    public Board(int no, String title, String writer, String content,
            Timestamp regDate, int readCount, String pass, String file1) {
        this.no = no;
        this.title = title;
        this.writer = writer;
        this.content = content;
        this.regDate = regDate;
        this.readCount = readCount;
        this.pass = pass;
        this.file1 = file1;
    }

    public int getNo() {
        return no;
    }
```

```
public void setNo(int no) {
    this.no = no;
}
public String getTitle() {
    return title;
}
public void setTitle(String title) {
    this.title = title;
}
public String getWriter() {
    return writer;
}
public void setWriter(String writer) {
    this.writer = writer;
}
public String getContent() {
    return content;
}
public void setContent(String content) {
    this.content = content;
}
public Timestamp getRegDate() {
    return regDate;
}
public void setRegDate(Timestamp regDate) {
    this.regDate = regDate;
}
public int getReadCount() {
    return readCount;
}
public void setReadCount(int readCount) {
    this.readCount = readCount;
}
public String getPass() {
    return pass;
}
public void setPass(String pass) {
    this.pass = pass;
}
public String getFile1() {
    return file1;
}
public void setFile1(String file1) {
    this.file1 = file1;
}
```

```
    public int getRecommend() {
        return recommend;
    }
    public void setRecommend(int recommend) {
        this.recommend = recommend;
    }
    public int getThank() {
        return thank;
    }
    public void setThank(int thank) {
        this.thank = thank;
    }
}
```

▶ 댓글 하나의 정보를 저장하는 Domain 클래스(새로 작성)

- com.jspstudy.bbs.domain.Reply

```
/* 하나의 댓글 정보를 저장하는 클래스(Domain, VO, Beans, DTO)
 * 댓글 정보를 저장하고 있는 테이블의 필드와 1:1 맵핑되는 Domain 클래스
 **/
public class Reply {
    private int no;
    private int bbsNo;
    private String replyContent;
    private String replyWriter;
    private Timestamp regDate;

    public Reply() { }
    public Reply(int bbsNo, String replyContent, String replyWriter) {
        this.bbsNo = bbsNo;
        this.replyContent = replyContent;
        this.replyWriter = replyWriter;
    }
    public Reply(int no, int bbsNo, String replyContent,
            String replyWriter, Timestamp regDate) {

        this.no = no;
        this.bbsNo = bbsNo;
        this.replyContent = replyContent;
        this.replyWriter = replyWriter;
        this.regDate = regDate;
    }
    public int getNo() {
        return no;
    }
```

```
    public void setNo(int no) {
        this.no = no;
    }
    public int getBbsNo() {
        return bbsNo;
    }
    public void setBbsNo(int bbsNo) {
        this.bbsNo = bbsNo;
    }
    public String getReplyContent() {
        return replyContent;
    }
    public void setReplyContent(String replyContent) {
        this.replyContent = replyContent;
    }
    public String getReplyWriter() {
        return replyWriter;
    }
    public void setReplyWriter(String replyWriter) {
        this.replyWriter = replyWriter;
    }
    public Timestamp getRegDate() {
        return regDate;
    }
    public void setRegDate(Timestamp regDate) {
        this.regDate = regDate;
    }
}
```

▶ DAO 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에 게시 글 번호에 해당하는 댓글 리스트를 가져오는 추상 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDao 에 추가

```
    // 게시 글 번호에 해당하는 댓글 리스트를 가져오는 메서드
    public abstract List<Reply> replyList(int no);
```

▶ 게시판 관련 SQL을 분리한 Mapper

BoardMapper.xml에 게시 글 번호에 해당하는 댓글 리스트를 가져오는 맵핑 구문을 추가한다.

- src/main/resources/repository/mappers/BoardMapper.xml 에 추가

```
    <!--
        게시 글 번호에 해당하는 댓글 리스트를 가져오는 맵핑 구문
```

```
        테이블의 컬럼명은 일반적으로 언더스코어 표기법("_")을 사용하는 경우가
        많고 클래스의 인스턴스 맴버는 카멜표기법을 사용한다.
        테이블의 컬럼명과 모델 클래스의 프로퍼티 이름이 다른 경우 아래와 같이
        SELECT 쿼리에 별칭을 사용해 모델 클래스의 프로퍼티 이름과 동일하게
        맞춰야 한다. 그렇지 않으면 오류는 발생하지 않지만 데이터를 읽어 올
        수 없다.
    -->
    <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>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 게시 글 번호에 해당하는 댓글 리스트를 가져오는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl 에 추가

```
    // 게시 글 번호에 해당하는 댓글 리스트를 가져오는 메서드
    public List<Reply> replyList(int no) {
        return sqlSession.selectList(NAME_SPACE + ".replyList", no);
    }
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 게시 글 번호에 해당하는 댓글 리스트를 반환하는 추상 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardService 에 추가

```
    // 게시 글 번호에 해당하는 댓글 리스트를 반환하는 메서드
    public abstract List<Reply> replyList(int no);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 게시 글 번호에 해당하는 댓글 리스트를 반환하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl 에 추가

```
// 게시 글 번호에 해당하는 댓글 리스트를 반환하는 메서드
public List<Reply> replyList(int no) {
    return boardDao.replyList(no);
}
```

▶ Controller 클래스

com.springstudy.bbs.controller 패키지의 BoardController 클래스에서 게시 글 상세보기 요청을 처리하는 메서드를 아래 붉은색 볼드체로 표시한 부분만 수정한다.

나머지 부분은 앞의 예제와 동일하기 때문에 추가되는 부분은 붉은색 볼드체로 표시하였다.

- com.springstudy.bbs.controller.BoardController

```
/* 게시 글 상세보기 요청을 처리하는 메서드
 *
 * 아래 @RequestMapping에서 method를 생략했기 때문에 "/boardDetail"로
 * 들어오는 GET 방식과 POST 방식의 요청 모두를 처리할 수 있다.
 *
 * @RequestMapping 애노테이션이 적용된 메서드의 파라미터와 반환 타입에
 * 대한 설명과 @RequestParam에 대한 설명은 boardList() 메서드의 주석을
 * 참고하기 바란다.
 *
 * 아래는 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.
 **/
@RequestMapping("/boardDetail")
public String boardDetail(Model model, int no,
        @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) throws Exception {

    /* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
     * 게시 글 리스트를 요청하는 것으로 간주하여 false 값을 갖게 한다.
     * Controller에서 type이나 keyword의 요청 파라미터가 없으면
     * 기본 값을 "null"로 지정했기 때문에 아래와 같이 체크했다.
```

```
         **/
        boolean searchOption = (type.equals("null")
                || keyword.equals("null")) ? false : true;

        /* Service 클래스를 이용해 no에 해당하는 게시 글 하나의 정보를 읽어온다.
         * 두 번째 인수에 true를 지정해 게시 글 읽은 횟수를 1 증가 시킨다.
         **/
        Board board = boardService.getBoard(no, true);

        // 현재 게시 글에 해당하는 댓글 리스트
        List<Reply> replyList = boardService.replyList(no);

        /* 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.
         * 모델에는 도메인 객체나 비즈니스 로직을 처리한 결과를 저장한다.
         **/
        model.addAttribute("board", board);
        model.addAttribute("replyList", replyList);
        model.addAttribute("pageNum", pageNum);
        model.addAttribute("searchOption", searchOption);

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

        /* servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에
         * 지정한 정보를 제외한 뷰 이름을 문자열로 반환하면 된다.
         *
         * 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
         **/
        return "boardDetail";
    }
```

▶ 메인 템플릿 페이지

게시 글 상세보기에서 댓글 리스트를 출력할 때 부트스트랩이 제공하는 아이콘을 사용할 예정이므로 부트스트랩의 아이콘 설명 페이지에서 맨 아래 부분에 있는 "설치" 부분의 내용을 참고해서 index.jsp 페이지에 부트스트랩 아이콘을 사용하기 위한 css 외부 링크를 추가하자. index.jsp 페이지도 부트스트랩 아이콘 사용을 위한 외부 링크를 설정하는 부분을 제외하고는 앞의 예제와 동일하기 때문에 **추가되거나 수정되는 부분은 붉은색 볼드체로 표시했다.**

https://icons.getbootstrap.kr/

- src/main/webapp/WEB-INF/index.jsp

```
... 앞부분 생략 ...

    <title>스프링 게시판</title>
    <link href="resources/bootstrap/bootstrap.min.css" rel="stylesheet" >

    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-icons.css">

    <link rel="stylesheet" type="text/css" href="resources/css/member.css" />

... 뒷부분 생략 ...
```

▶ 게시 글 상세보기 View(수정)

게시 글 상세보기 페이지도 댓글 관련 부분을 제외하고는 앞의 예제와 동일하기 때문에 **추가되거나 수정되는 부분은 붉은색 볼드체로 표시했다.**

- src/main/webapp/WEB-INF/views/boardDetail.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<script src="resources/js/reply.js"></script>

<!-- content -->
<div class="row my-5" id="global-content">
    <div class="col">
        <form name="checkForm" id="checkForm">
            <input type="hidden" name="no" id="no" value="${ board.no }"/>
            <input type="hidden" name="pass" id="rPass" />
            <input type="hidden" name="pageNum" value="${ pageNum }" />

            <%--
                검색 리스트에서 들어온 요청일 경우 다시 keyword에 해당하는
                검색 리스트로 돌려보내기 위해서 아래의 파라미터가 필요하다.
             --%>
            <c:if test="${ searchOption }">
                <input type="hidden" name="type" value="${ type }" />
                <input type="hidden" name="keyword" value="${ keyword }" />
            </c:if>
        </form>
        <div class="row text-center">
```

```
            <div class="col">
                <h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
            </div>
        </div>

        <!-- 게시 글 상세보기 영역  -->
        <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">${ board.title }</td>
                        </tr>
                        <tr>
                            <th>글쓴이</th>
                            <td>${ board.writer }</td>
                            <th>작성일</th>
                            <td><fmt:formatDate value="${ board.regDate }"
                                pattern="yyyy-MM-dd HH:mm:ss" /></td>
                        </tr>
                        <tr>
                            <th>비밀번호</th>
                            <td>
                                <div class="col-sm-8">
                                    <input        class="form-control"        type="password"
name="pass" id="pass">
                                </div>
                            </td>
                            <th>조회수</th>
                            <td>${ board.readCount }</td>
                        </tr>
                        <tr>
                            <th>파&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;일</th>
                            <td colspan="3">
                            <c:if test="${ empty board.file1 }">
                                첨부파일 없음
                            </c:if>
                            <c:if test="${ not empty board.file1 }">
                                <a  href="fileDownload?fileName=${ board.file1 }">파일 다운
로드</a>
                            </c:if>
                            </td>
                        </tr>
                        <tr>
```

```
                        <td colspan="4">
                            <pre>${ board.content }</pre>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>

    <!-- 상세보기 버튼 영역 -->
    <div class="row my-3">
        <div class="col text-center">
            <input class="btn btn-warning" type="button" id="detailUpdate" value="수정
하기"/>
              <input class="btn btn-danger"  type="button" id="detailDelete"
value="삭제하기" />
            <%--
                일반 게시 글 리스트에서 온 요청이면 일반 게시 글 리스트로 돌려 보낸다.
            --%>
            <c:if test="${ not searchOption }">
              <input class="btn btn-primary" type="button" value="목록보기"
                    onclick="location.href='boardList?pageNum=${pageNum}'"/>
            </c:if>
            <%--
                검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
            --%>
            <c:if test="${ searchOption }">
              <input class="btn btn-primary" type="button" value="목록보기"
                    onclick="location.href='boardList?pageNum=${pageNum}&type=${
type }&keyword=${ keyword }'"/>
            </c:if>
        </div>
    </div>

    <!--
        아래부터 추천/땡큐와 댓글 기능을 처리하는 부분으로
        기존의 게시 글 상세보기 뷰 페이지의 아래쪽에 다음의 코드를 추가하면 된다.
    -->
    <!-- 추천/땡큐 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;">
                    <img      src="resources/images/recommend.png"      alt="추천하기"
```

```
/> 추천
                        <span class="recommend" >(${ board.recommend })</span>
                    </span> |
                    <span  id="thank"  class="btnCommend  text-primary"  style="cursor:
pointer;">
                        <img src="resources/images/smile.png" alt="땡큐"/> 땡큐
                        <span class="recommend">(${ 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> 댓글쓰기
                    </span>
                </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 -->
        <c:if test="${ not empty replyList }" >
        <div class="row mb-3">
            <div class="col" id="replyList">
                <c:forEach var="reply" items="${ replyList }" >
                <div class="replyRow row border border-top-0">
                    <div class="col">
                        <div class="row bg-light p-2">
                            <div class="col-4">
                                <span>${ reply.replyWriter }</span>
                            </div>
                            <div class="col-8 text-end">
                                <span class="me-3">
                                    <fmt:formatDate       value="${       reply.regDate}"
pattern="yyyy-MM-dd HH:mm:ss" />
                                </span>
                                <button    class="modifyReply    btn    btn-outline-success
btn-sm" data-no="${ reply.no }">
                                    <i class="bi bi-journal-text">수정</i>
```

```
                                </button>
                                <button class="deleteReply btn btn-outline-warning
btn-sm" data-no="${ reply.no }">
                                    <i class="bi bi-trash">삭제</i>
                                </button>
                                <button class="btn btn-outline-danger btn-sm"
onclick="reportReply('${ reply.no }')">
                                    <i class="bi bi-telephone-outbound">신고</i>

                                </button>
                            </div>
                        </div>
                        <div class="row">
                            <div class="col p-3">
                                <pre>${ reply.replyContent }</pre>
                            </div>
                        </div>
                    </div>
                </div>
                </c:forEach>
            </div>
        </div>
        </c:if>

        <!-- 댓글이 존재하지 않는 경우 -->
        <c:if test="${ empty replyList }" >
        <div class="row mb-5" id="replyList">
            <div class="col text-center border p-5">
                <div>이 게시 글에 대한 댓글이 존재하지 않습니다.</div>
            </div>
        </div>
        </c:if>

        <!-- 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" value="${ board.no }"/>
                    <input type="hidden" name="replyWriter" value="${
sessionScope.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>
                                </div>
                            </div>
                            <div class="row my-3">
                                <div class="col-md-10">
                                    <textarea      name="replyContent"      id="replyContent"
class="form-control" rows="4"></textarea>
                                </div>
                                <div class="col-md">
                                    <input type="submit" value="댓글쓰기"          class="btn
btn-primary h-100 w-100" id="replyWriteButton">
                                </div>
                            </div>
                        </div>
                    </div>
                </form>
            </div>
        </div><!-- end replyForm -->

    </div>
</div>
```

3) 게시 글 상세보기 페이지 추천/땡큐 Ajax 구현

앞에서 게시 글 상세보기 페이지에 추천/땡큐를 할 수 있는 버튼과 댓글 리스트를 출력하는 기능을 추가하였다. 앞에서 구성한 게시 글 상세보기 페이지에서 추천/땡큐를 Ajax를 처리하는 부분을 구현할 것이다.

▶ DAO 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에 게시 글 번호에 해당하는 추천/땡큐를 업데이트 하는 메서드와 새롭게 반영된 추천/땡큐 정보를 가져오는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDao 에 추가

```
// 게시 글 번호에 해당하는 추천/땡큐를 업데이트 하는 메서드
public abstract void updateRecommend(int no, String recommend);

// 게시 글 번호에 해당하는 추천/땡큐 정보를 가져오는 메서드
public abstract Board getRecommend(int no);
```

▶ 게시판 관련 SQL을 분리한 Mapper

BoardMapper.xml에 게시 글 번호에 해당하는 추천/땡큐를 업데이트 하는 맵핑 구문과 새롭게 반영된 추천/땡큐 정보를 가져오는 맵핑 구문을 아래와 같이 추가한다.

- src/main/resources/repository/mappers/BoardMapper.xml 에 추가

```
<!--
    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 하도록 맵핑 구문을
    작성했다.
    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 updateRecommend()
    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.

    HashMap에 저장한 데이터가 int와 같이 자바 기본 형과 String 데이터이므로
    parameterType을 생략했다.
-->
<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 반화할 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>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 게시 글 번호에 해당하는 추천/땡큐를 업데이트 하는 메서드와 새롭게 반영된 추천/땡큐 정보를 가져오는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl 에 추가

```
    // 게시 글 번호에 해당하는 추천/땡큐를 업데이트 하는 메서드
    public void updateRecommend(int no, String recommend) {

        // 맵퍼로 보낼 파라미터가 하나 이상이라 Map에 담아 맵퍼로 보냈다.
        Map<String, Object> params = new HashMap<String, Object>();
        params.put("no", no);
        params.put("recommend", recommend);
        sqlSession.update(NAME_SPACE + ".updateRecommend", params);
    }

    // 게시 글 번호에 해당하는 추천/땡큐 정보를 가져오는 메서드
    public Board getRecommend(int no) {
        return sqlSession.selectOne(NAME_SPACE + ".getRecommend", no);
    }
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 게시 글 번호에 해당하는 추천/땡큐를 업데이트 하고 새롭게 반영된 추천/땡큐 정보를 가져오는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardService 에 추가

```
// 추천/땡큐 정보를 업데이트하고 갱신된 추천/땡큐를 가져오는 메서드
public Map<String, Integer> recommend(int no, String recommend);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 게시 글 번호에 해당하는 추천/땡큐를 업데이트 하고 새롭게 반영된 추천/땡큐 정보를 가져오는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl 에 추가

```
// 추천/땡큐 정보를 업데이트하고 갱신된 추천/땡큐를 가져오는 메서드
public Map<String, Integer> recommend(int no, String recommend) {

    boardDao.updateRecommend(no, recommend);
    Board board = boardDao.getRecommend(no);

    Map<String, Integer> map = new HashMap<String, Integer>();
    map.put("recommend", board.getRecommend());
    map.put("thank", board.getThank());
    return map;
}
```

▶ Ajax 요청처리 Controller 클래스(새로 작성)

com.springstudy.bbs.ajax 패키지에 BoardAjaxController 클래스를 새롭게 만들고 추천/땡큐 요청을 처리하는 메서드를 아래와 같이 구현한다.

- com.springstudy.bbs.ajax.BoardAjaxController

```
@Controller
public class BoardAjaxController {

    @Autowired
    BoardService boardService;

    /* 추천/탱큐 요청을 처리하는 메서드
     *
     * 스프링 mvc에서 xml이나 json 형식의 응답 데이터를 만들려면
     * xml 또는 json 형식의 응답을 생성하는 뷰 클래스를 사용하거나
     * HttpServletResponse 객체를 직접 사용해 필요한 응답 데이터를
```

```
* 생성할 수있다.
* 스프링 mvc가 지원 하는 아래와 같은 애노테이션을 사용하면 text, xml,
* json 형식의 요청 데이터를 자바 객체로 변환해 주거나 자바 객체를 text,
* xml, json 형식으로 변환해 응답 본문에 추가해 준다.
*
* @RequestBody
* 클라이언트가 요청할 때 요청 본문으로 넘어오는 데이터를 자바 객체로 변환할
* 때 사용한다. 예를 들면 post 방식 요청에서 본문에 실어 넘어오는 파라미터를
* 자바의 String으로 변환하거나 json 형식의 데이터를 자바 객체로 변환하기
* 위해 사용한다.
*
* @ResponseBody
* 자바 객체를 json 형식이나 xml 형식의 문자열로 변환하여 응답 본문에 실어
* 보내기 위해 사용한다.
* 컨트롤러 메서드에 @ResponseBody가 적용되면 메서드의 반환 값은 스프링
* mvc에 의해서 Http 응답 본문에 포함 된다.
*
*
* MappingJackson2HttpMessageConverter를 이용한 json 응답처리
*
* 요청 본문의 json 형식의 데이터를 자바 객체로 변환해 주거나 자바 객체를
* 응답 본문의 json 형식의 데이터로 변환해 준다.
* 지원하는 요청 컨텐츠 타입은 아래와 같다.
* application/json, application/*+json
*
* 참고사이트 : https://github.com/FasterXML
* http://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/
*
* 스프링이 제공하는 MappingJackson2HttpMessageConverter는
* Jackson2 라이브러리를 이용해서 자바 객체와 json 데이터를 변환하기
* 때문에 pom.xml에서 "jackson-databind"으로 검색해 아래와 같이
* Jackson2 라이브러리 의존 설정을 해야 한다.
* 그렇지 않으면 json 타입의 요청을 자바 객체로 변환할 수 없기 때문에
* 415 Unsupported Media Type 응답을 받게 된다.
*
* <dependency>
*       <groupId>com.fasterxml.jackson.core</groupId>
*           <artifactId>jackson-databind</artifactId>
*           <version>2.8.5</version>
*  </dependency>
*
* 스프링 빈 설정 파일에 HttpMessageConvert 인터페이스를
* 구현한 MappingJackson2HttpMessageConverter가 빈으로
* 등록되어 있어야 응답 본문으로 들어오는 json 형식의 데이터를 자바 객체로
* 변환하거나 자바 객체를 json 형식의 데이터로 변환해 응답 본문에 포함 시킬
```

```
     * 수 있다. 하지만 스프링 빈 설정 파일에 <mvc:annotation-driven />이
     * 적용되었기 때문에 MappingJackson2HttpMessageConverter를
     * 포함한 다수의 HttpMessageConverter 구현체를 빈으로 등록해 준다.
     *
     * @RequestMapping 애노테이션이 적용된 Controller의 메서드에
     * 아래와 같이 @ResponseBody를 적용하고 Map 객체를 반환하면
     * MappingJackson2HttpMessageConverter에 의해서 JSON
     * 형식으로 변환된다.
     **/
    @RequestMapping("/recommend.ajax")
    @ResponseBody
    public Map<String, Integer> recommend(int no, String recommend) {

        /* 위에서도 언급했지만 @RequestMapping 애노테이션이 적용된
         * Controller 메서드에 @ResponseBody 애노테이션이 적용되면
         * 반환 타입이 String 일 경우 HttpMessageConverter를 사용해
         * String 객체를 직렬화 하고 반환 타입이 위와 같이 Map이거나 자바
         * 객체인 경우 MappingJackson2HttpMessageConverter를
         * 사용해 JSON 형식으로 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);
    }
}
```

▶ 자바스크립트(새로 작성)

- src/main/webapp/resources/js/reply.js 에 추가

```
// DOM(Document Object Model)이 준비 되었다면
$(document).ready(function() {

    // 추천/땡큐 Ajax
    $(".btnCommend").click(function() {

        var com = $(this).attr("id");
        console.log("com : " + com);
```

```
        $.ajax({
            url: "recommend.ajax",

            // type을 지정하지 않으면 get 방식 요청이다.
            type: "post",

            // 파라미터로 보낼 데이터를 객체 리터럴로 지정하고 있다.
            data : { recommend: com, no : $("#no").val()},

            /* RecommendAction 클래스에서 Gson 라이브러리를 이용해
             * 응답 데이터를 json 형식으로 출력했기 때문에 dataType을 json
             * 으로 지정해야 한다. 응답 데이터를 json 형식으로 받았기 때문에
             * Ajax 통신이 성공하면 실행될 함수의 첫 번째 인수로 지정한
             * data는 자바스크립트 객체이므로 닷(.) 연산자를 이용해 접근할 수 있다.
             **/
            dataType: "json",
            success: function(data) {
                /* 추천/땡큐가 반영된 것을 사용자에게 알리고
                 * 응답으로 받은 갱신된 추천하기 데이터를 화면에 표시한다.
                 **/
                var msg = recom == '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);
            }
        });
    });
}
```

4) 게시 글 상세보기 페이지 댓글 Ajax 구현

앞에서 게시 글 상세보기 페이지에서 댓글 리스트를 출력하는 기능을 추가하였다. 이번에는 게시 글 상세보기 페이지에서 댓글을 쓰고 수정하고 삭제하는 기능을 구현할 것이다.

4-1) 댓글 쓰기

▶ DAO 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에 댓글을 추가하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDao 에 추가

```
// 게시 글 번호에 해당하는 댓글을 DB에 등록하는 메서드
public void addReply(Reply reply);
```

▶ 게시판 관련 SQL을 분리한 Mapper

BoardMapper.xml에 댓글을 추가하는 맵핑 구문을 아래와 같이 추가한다.

- src/main/resources/repository/mappers/BoardMapper.xml 에 추가

```
<!--
    댓글을 추가하는 맵핑 구문
-->
<insert id="addReply" parameterType="Reply">
    INSERT INTO reply(bbs_no, reply_content, reply_writer, reg_date)
    VALUES(#{bbsNo}, #{replyContent}, #{replyWriter}, SYSDATE())
</insert>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 댓글을 추가하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl 에 추가

```
// 게시 글 번호에 해당하는 댓글을 DB에 등록하는 메서드
public void addReply(Reply reply) {
    sqlSession.insert(NAME_SPACE + ".addReply", reply);
}
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 댓글을 추가하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardService 에 추가

```
// BoardDao를 이용해 게시 글 번호에 해당하는 댓글을 등록하는 메서드
public void addReply(Reply reply);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 댓글을 추가하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl 에 추가

```
// BoardDao를 이용해 게시 글 번호에 해당하는 댓글을 등록하는 메서드
public void addReply(Reply reply) {
    boardDao.addReply(reply);
}
```

▶ Controller 클래스

BoardAjaxController 클래스에 댓글 쓰기 요청을 처리하는 메서드를 아래와 같이 구현한다.

- com.springstudy.bbs.ajax.BoardAjaxController 에 추가

```
// 댓글 쓰기 요청을 처리하는 메서드
@RequestMapping("/replyWrite.ajax")
@ResponseBody
public List<Reply> addReply(Reply reply) {

    // 새로운 댓글을 등록한다.
    boardService.addReply(reply);

    /* 댓글 쓰기가 완료되면 새롭게 추가된 댓글을 포함해서 게시 글
     * 상세보기에 다시 출력해야 하므로 갱신된 댓글 리스트를 가져와 반환한다.
     *
     * 아래는 게시 글 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 동감이여..", replyWriter: "midas"},
     *  {bbsNo: 100, no: 20, ... },
     *     ...
     *  {bbsNo: 100, no: 1, regDate: 1462682672000,
     *       replyContent: "항상 감사합니다...", replyWriter: "midas"}
     * ]
     **/
```

```
        return boardService.replyList(reply.getBbsNo());
    }
```

▶ 자바스크립트

$(document).ready(function() { } 안에 댓글쓰기 버튼이 클릭되었을 때와 댓글쓰기 폼이 Submit 될 때 처리하는 자바스크립트 이벤트 처리 코드를 아래와 같이 추가한다.

- src/main/webapp/resources/js/reply.js 에 추가

```
    // 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 상태이면
        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 상태이면
            $("#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 그 때 그 때 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);
            }
        });

        // 폼의 전송을 취소
        return false;
    });
```

4-2) 댓글 수정하기

▶ DAO 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에 댓글을 수정하는 추상 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDao 에 추가

```
// DB에서 댓글 번호에 해당하는 댓글을 수정하는 메서드
public void updateReply(Reply reply);
```

▶ 게시판 관련 SQL을 분리한 Mapper

BoardMapper.xml에 댓글을 수정하는 맵핑 구문을 아래와 같이 추가한다.

- src/main/resources/repository/mappers/BoardMapper.xml 에 추가

```
<!--
    댓글을 수정하는 맵핑 구문
-->
<update id="updateReply" parameterType="Reply">
    UPDATE reply
        SET reply_content = #{replyContent},
            reg_date = SYSDATE()
    WHERE no = #{no}
</update>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 댓글을 수정하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl 에 추가

```
// DB에서 댓글 번호에 해당하는 댓글을 수정하는 메서드
public void updateReply(Reply reply) {
    sqlSession.update(NAME_SPACE + ".updateReply", reply);
```

```
}
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 댓글을 수정하는 추상 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardService 에 추가

```
// BoardDao를 이용해 댓글을 수정하는 메서드
public void updateReply(Reply reply);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 댓글을 수정하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl 에 추가

```
// BoardDao를 이용해 댓글을 수정하는 메서드
public void updateReply(Reply reply) {
    boardDao.updateReply(reply);
}
```

▶ Controller 클래스

BoardAjaxController 클래스에 댓글 수정 요청을 처리하는 메서드를 아래와 같이 구현한다.

- com.springstudy.bbs.ajax.BoardAjaxController 에 추가

```
// 댓글 수정 요청을 처리하는 메서드
@RequestMapping("/replyUpdate.ajax")
@ResponseBody
public List<Reply> updateReply(Reply reply) {

    // 새로운 댓글을 등록한다.
    boardService.updateReply(reply);

    // 새롭게 갱신된 댓글 리스트를 가져와 반환한다.
    return boardService.replyList(reply.getBbsNo());
}
```

▶ 자바스크립트

$(document).ready(function() { } 안에 수정 버튼이 클릭되었을 때와 댓글수정 폼이 Submit 될 때 처리하는 자바스크립트 이벤트 처리 코드를 아래와 같이 추가한다.

- src/main/webapp/resources/js/reply.js 에 추가

```
/* 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 상태이면
    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 상태이면
        $("#replyForm").removeClass("d-none")
            .css("display", "none").insertAfter($replyRow).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": "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 백업을 받아야 한다.
         **/
        $replyForm = $("#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: "post",
            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 쓰기가 완료되면 폼을 숨긴다.
                $("#replyContent").val("");
                $replyForm.css("display", "none");
                $("#global-content > div.col").append($replyForm);

            },
            error: function(xhr, status, error) {
                alert("ajax 실패 : " + status + " - " + xhr.status);
            }

        });

        // 폼이 submit 되는 것을 취소한다.
        return false;
    });
```

4-3) 댓글 삭제하기

▶ DAO 계층 구현

com.springstudy.bbs.dao 패키지의 BoardDao 인터페이스에 댓글을 삭제하는 추상 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDao 에 추가

```
// DB에서 댓글 번호에 해당하는 댓글을 삭제하는 메서드
public void deleteReply(int no);
```

▶ 게시판 관련 SQL을 분리한 Mapper

BoardMapper.xml에 댓글을 삭제하는 맵핑 구문을 아래와 같이 추가한다.

- src/main/resources/repository/mappers/BoardMapper.xml 에 추가

```
<!--
    댓글을 삭제하는 맵핑 구문
-->
<delete id="deleteReply">
    DELETE FROM reply
    WHERE no = #{no}
</delete>
```

▶ BoardDao 인터페이스 구현 클래스

BoardDaoImpl 클래스에 댓글을 삭제하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.dao.BoardDaoImpl 에 추가

```
// DB에서 댓글 번호에 해당하는 댓글을 삭제하는 메서드
public void deleteReply(int no) {
    sqlSession.delete(NAME_SPACE + ".deleteReply", no);
}
```

▶ Service 계층 구현

com.springstudy.bbs.service 패키지의 BoardService 인터페이스에 댓글을 삭제하는 추상 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardService 에 추가

```
// BoardDao를 이용해 댓글 번호에 해당하는 댓글을 삭제하는 메서드
public void deleteReply(int no);
```

▶ BoardService 인터페이스 구현 클래스

BoardServiceImpl 클래스에 댓글을 삭제하는 메서드를 아래와 같이 추가한다.

- com.springstudy.bbs.service.BoardServiceImpl 에 추가

```
// BoardDao를 이용해 댓글 번호에 해당하는 댓글을 삭제하는 메서드
public void deleteReply(int no) {
    boardDao.deleteReply(no);
}
```

▶ Controller 클래스

BoardAjaxController 클래스에 댓글 삭제 요청을 처리하는 메서드를 아래와 같이 구현한다.

- com.springstudy.bbs.ajax.BoardAjaxController 에 추가

```
// 댓글 삭제 요청을 처리하는 메서드
@RequestMapping("/replyDelete.ajax")
@ResponseBody
public List<Reply> deleteReply(int no, int bbsNo) {

    // 새로운 댓글을 등록한다.
    boardService.deleteReply(no);

    // 새롭게 갱신된 댓글 리스트를 가져와 반환한다.
    return boardService.replyList(bbsNo);
}
```

▶ 자바스크립트

$(document).ready(function() { } 안에 댓글 삭제 관련 자바스크립트 코드를 추가한다.

- src/main/webapp/resources/js/reply.js 에 추가

```
/* 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() {

    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);

    /* 아래에서 $("#replyTable").empty(); 가 호출되면
     * 댓글 쓰기 폼이 문서에서 삭제될 수 있으므로 백업을 받아야 한다.
     **/
    $replyForm = $("#replyForm").slideUp(300);

    var result = confirm(writer + "님이 작성한 " + no +"번 댓글을 삭제하시겠습니까?");

    if(result) {
        $.ajax({
            url: "replyDelete.ajax",
            type: "post",
            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 쓰기가 완료되면 폼을 숨긴다.
                    $("#replyContent").val("");
                    $replyForm.css("display", "none");
                    $("#global-content > div.col").append($replyForm);

                },
                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 */
function reportReply(elemId) {
    var result = confirm("이 댓글을 신고하시겠습니까?");
    if(result == true) {
        alert("report - " + result);
    }
}
```