

코드로 배우는 스프링 웹프로젝트

작성일: 2020.02.05

작성자: 도원진

[목차]


1. 스프링 개발환경 구성
 - 1-1. 프로젝트 생성
 - 1-2. Lombok라이브러리 설치
 - 1-3. DB설정
2. 기본적인 웹 게시물 관리
 - 2-1. 디자인
 - 2-2. 개발
 - A. 영속 영역의 CRUD 구현
 - B. 페이징 화면처리
 - C. 검색처리
 - D. @RestController의 반환 타입
3. Ajax 댓글처리
4. 파일업로드 처리
 - 4-1. 업로드를 위한 기본 설정
 - 4-2. Ajax를 통한 업로드


1. 스프링 개발환경 구성

1-1. 프로젝트생성

- File > New > **Spring Legacy Project** 선택 > **Spring Legacy Project** 창에서 **Spring MVC Project** 선택
- 이후 화면에서 패키지 경로 3단 지정

Spring Legacy Project



 Click 'Next' to load the template contents.

Project name:


☒ Use default location

Location:

Select Spring version:

Templates:

- Simple Projects
 - Simple Java
 - Simple Spring Maven
 - Simple Spring Web Maven
- Batch
- GemFire
- Integration
- Persistence
- Simple Spring Utility Project
- Spring MVC Project**

 requires downloading

Description:


A new Spring MVC web application development project

URL: <https://dist.springsource.com/release/STS/help/org.springframework.templates.mvc-3.2.2.zip>

Working sets

☐ Add project to working sets

Working sets:



1-2. pom.xml 을 이용한 dependency 관리

- pom.xml 에 설정해 놓은 dependency로 일일이 Library를 프로젝트로 다운받아 복사하는 수고를 덜어준다.

1-1과정의 프로젝트 생성직후 스프링 버전은 4.3 이후로 변경해준다.

```
<java-version>1.8</java-version>
<org.springframework-version>5.0.7.RELEASE</org.springframework-version>
```

각 코드들은 **Maven Repository** 홈페이지를 통해 버전별로 확인할 수 있다.

```

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>${org.springframework-version}</version>
  <scope>test</scope>
</dependency>

```

1-3. Lombok 라이브러리 설치

- 생성자, getter/setter, toString 함수를 작성하는 수고를 덜어준다.

```

-->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId> <!--test를 위해 추가해야할 의존성
  <version>${org.springframework-version}</version>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>

```

1-4. 프로젝트의 JDBC 연결

- jdbc.jar 파일을 다운받는 수고를 하지 말고 **Maven**의 도움을 받자

```

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.48</version>
</dependency>

```

1-5. 커넥션 풀 설정

설정 방법

- XML 설정 root-context.xml

(hikari DBCP를 이용하기 때문에 dataSource설정이 위의 코드와 살짝 다름에 유의)

```
<!-- HikariCP configuration -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <!-- 오라클인 경우 -->
    <!--<property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"></property>
    <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:XE">
</property> -->
    <property name="driverClassName" value="com.mysql.jdbc.Driver">
</property>
    <property name="jdbcUrl" value="jdbc:mysql://localhost/bizspring">
</property>
    <property name="username" value="root"></property>
    <property name="password" value="1234"></property>
</bean>

<!-- HikariCP DataSource -->
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-
method="close">
    <constructor-arg ref="hikariConfig" />
</bean>
```

1-6. MyBatis와 스프링 연동

- 라이브러리
 - mybatis
 - mybatis-spring
 - spring-jdbc
 - spring-tx
- xml 설정 root-context.xml

```
<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

1-7. 스프링과의 연동 처리

- Mapper인터페이스

```
package org.zerock.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {
    @Select("SELECT now() FROM dual")
    public String getTime();
}
```

- 위 인터페이스를 Bean등록 root-context.xml
 - namespace 로 mybatis-spring추가해야함.

```
<mybatis-spring:scan base-package="org.zerock.mapper" />
```

- 4.3 이후)

Mapper인터페이스에 대응하는 메소드 선언 (단, Annotation은 없다)

```
package org.zerock.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {
    //@Select("SELECT sysdate FROM dual")
    public String getTime();
}
```

mapper.xml작성

위치 : Mapper인터페이스가 있는 패키지 또는

src/main/resources/org/zerock/mapper/

1. namespace 속성값은 **Mapper인터페이스의 패키지경로**
2. sql 태그의 id 속성값은 **Mapper인터페이스에 있는 해당 메소드명**

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

  <select id="getTime2" resultType="string">
    SELECT sysdate FROM dual
  </select>

</mapper>
```

2.기본적인 웹 게시물 관리

2-1. 디자인

bootstrap실습파일_ 카페자료실에 있는 압축파일 다운로드

HTML, CSS, JavaScript Library 다운로드 후 이클립스 프로젝트로 복사 붙여넣기

2-2. 개발

A. 영속 영역의 CRUD 구현

XMLmapper 와 Mapper 인터페이스 함께사용 (MyBatis이용하기)

- 쓰기1

useGeneratedKeys="true" keyProperty="bno" 속성 두개로 Oracle의 nextVal()을 대체한다.

```
<insert id="insert" useGeneratedKeys="true" keyProperty="bno">
    insert into tbl_board (bno,title,content,writer)
    values ({bno}, #{title}, #{content}, #{writer})
</insert>
```

- 쓰기2

MySQL의 AutoIncrement 를 활용할 수 있기 때문에 Insert를 진행할 때
AutoIncrement가 설정된 칼럼은 제외시킨다.

(참고) MySQL 함수 LAST_INSERT_ID() : insert한 최종 bno 값

```
<insert id="insertSelectKey">
    insert into tbl_board (title,content, writer)
    values ({title}, #{content}, #{writer})
</insert>
```

- 읽기

```
<select id="getList" resultType="org.zerock.domain.BoardVO">
    <![CDATA[
        select * from tbl_board where bno > 0
    ]]>
</select>
```

- 수정

```
<update id="update">
    update tbl_board
    set title= #{title},
    content=#{content},
    writer = #{writer},
    updateDate = now()
    where bno =
    #{bno}
</update>
```

- 삭제

```
<delete id="delete">
    delete from tbl_board where bno = #{bno}
</delete>
```

B. 페이징 화면 처리

[페이징 처리할 때 필요한 정보들]

변수	의미	비고
pageNum	현재 보고있는 페이지 위치, (사용자 input)	
total	총 게시물의 수 (DB질의 통해)	
amount	한 페이지에서 보여줄 게시글 수, (상수, 일반적으로 10)	
endPage	pageNum 를 통해 계산	
startPage	pageNum 를 통해 계산	
realEnd	total, amount 를 통해 계산	

예)

아래 게시글 amount = 4, pageNum = 2

게시글 번호	제목	글쓴인	날짜	조회수
5	A	원진	2017-10-31	5
6	B	Kevin	2020-10-31	7
7	C	Merry	2019-10-31	3
8	D	Max	2020-10-31	1

Navigation Per Page)

Navigation Per Page = 1

startPage	pageNum			endPage
1	2	3	4	5

Navigation Per Page = 2

startPage				endPage
6	7	8	9	10

Navigation Per Page = 3 (마지막 페이지)

startPage	realEnd			endPage
11	12			

- PageDTO에서 계산된 값

- 페이징

- endPage

- 하단 페이지 번호칸의 마지막 번호

- 현재 pageNum이 결정함

- `int endPage = (int) (Math.ceil(cri.getPageNum() / 10.0)) * 10`

- startPage

- `int startPage = endPage - (pageLength - 1)`

(pageLength : 하단 페이지 번호칸들의 길이)

- prev, next

- prev

- startPage가 결정

- `this.prev = this.startPage > 1;`

- next

- endPage 와 realEnd 가 결정

- `this.next = this.endPage < realEnd;`

- 마지막 페이지 단까지 온 상황

- endPage값을 더 세밀하게 조정해줘야 함(값을 더 내림)

```
int realEnd = (int) (Math.ceil((total * 1.0) /
cri.getAmount()));
if (realEnd <= this.endPage) {
    this.endPage = realEnd;
}
```

C. 검색 처리

[검색 조건 처리를 위한 Criteria의 변화]

- Criteria

- 기존

- pageNum

- amount

- 추가

- type

- amount

- String[] getTypeArr() : type(eg. "T", "W", "C", "TW", "TC", "TWC")를 각 문자로 분리시키는 메소드
- ```
<foreach item='type' collection="typeArr">
 <trim prefix="OR">

 </trim>
</foreach>
```
- Mapper interface에 getListwithPaging(Criteria cri)메소드에 파라미터 cri가 있다.
- cri의 getTypeArr 메소드의 실행으로 반환값을 자동으로 받음
- SQL 모듈화
  - <sql id="criteria"> 와 <include refid="criteria"> 를 통해 sql문을 모듈화할 수 있다
  - 이렇게 모듈화된 sql쿼리문은 where 절에 편리하게 반복사용됨.

## D. @RestController의 반환 타입

```
public class SampleController {

 @GetMapping(value = "/getText", produces = "text/plain; charset=UTF-8")
 public String getText() {

 log.info("MIME TYPE: " + MediaType.TEXT_PLAIN_VALUE);

 return "안녕하세요";
 }
}
```

- produces 속성
  - MediaType의 상수를 사용할 수 있음.
  - 생략가능
    - 기본 xml 리턴
    - url주소 맨 뒤에 .json을 붙이면 JSON객체 리턴
- ResponseEntity<> 리턴타입
  - Controller안 메소드 리턴타입의 한 종류
  - 구성요소
    - 결과데이터
    - StatusCode

```
○ @GetMapping(value = "/check", params = { "height", "weight" })
 public ResponseEntity<SampleVO> check(Double height, Double weight)
 {

 SampleVO vo = new SampleVO(000, "" + height, "" + weight);

 ResponseEntity<SampleVO> result = null;

 if (height < 150) {
```

```

 result =
 ResponseEntity.status(HttpStatus.BAD_GATEWAY).body(vo);
 } else {
 result = ResponseEntity.status(HttpStatus.OK).body(vo);
 }

 return result;
}

```

## 3. Ajax 댓글 처리

### 3-1. 댓글 처리를 위한 영속 영역

- ReplyVO 클래스 추가
  - rno
  - bno
  - reply
  - replyer
  - replyDate
  - updateDate

### 3-2. 두개 이상의 파라미터를 Mapper가 사용할 상황

- 이유
  - bno : 게시물의 번호
  - Cri : 그 게시물에 관련된 모든 댓글목록 및 페이지징처리

```

public List<ReplyVO> getListWithPaging(@Param("cri") Criteria cri,
@Param("bno") Long bno);

```

@Param("a") Type x 해석: 인터페이스에서 x로 받은 데이터를 Mapper.xml 에서 a로 사용하겠다.

### 3-3. 서비스 영역과 Controller 처리

- REST방식으로 동작하는 URL을 설계할 때는 PK를 기준으로 작성하는 것이 좋습니다.
- @RequestBody를 하는 이유 : String이 전송될 때는 어노테이션없이도 메소드의 파라미터로 오는 vo 객체의 필드에 자동 바인딩이 되나, Json String이 전송될 때는 @RequestBody 어노테이션을 꼭 붙여야한다.

```

@RequestMapping(method = { RequestMethod.PUT,
 RequestMethod.PATCH }, value =("/{rno}", consumes =
"application/json", produces = {
 MediaType.TEXT_PLAIN_VALUE })
public ResponseEntity<String> modify(
 @RequestBody ReplyVO vo, //ReplyVO에 rno 가 있으나, request body에서
만 맵핑하기 때문에
 @PathVariable("rno") Long rno) {

```

```

 vo.setRno(rno);

 log.info("rno: " + rno);
 log.info("modify: " + vo);

 return service.modify(vo) == 1
 ? new ResponseEntity<>("success", HttpStatus.OK)
 : new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
 }
}

```

ReplyVO에 rno 가 있으나, request body에서만 맵핑하기 때문에 vo 객체의 rno는 null임.

따라서, @RequestBody를 항상 Post와 같이 사용하는건 아니다. 서버가 JSONString을 전송받을 때 사용

```

var replyService = (function() {

 function add(reply, callback, error) {
 console.log("add reply.....");

 $.ajax({
 type : 'post',
 url : '/replies/new',
 data : JSON.stringify(reply),
 contentType : "application/json; charset=utf-8",
 success : function(result, status, xhr) {
 if (callback) {
 callback(result);
 }
 },
 error : function(xhr, status, er) {
 if (error) {
 error(er);
 }
 }
 })
 }

 return {
 add : add,
 };

})();
}

```

## 4. 파일 업로드 처리

### 4-1. 스프링의 첨부파일을 위한 기본설정

web.xml 변경(첨부파일 처리에 대한 톰캣WAS자체 설정)

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
 id="WebApp_ID" version="3.1">
```

추가

```
<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>
 <load-on-startup>1</load-on-startup>

 <multipart-config>
 <location>C:\\upload\\temp</location> //업로드 경로
 <max-file-size>20971520</max-file-size> <!--1MB * 20 -->
 <max-request-size>41943040</max-request-size><!-- 40MB -->
 <file-size-threshold>20971520</file-size-threshold> <!-- 20MB -->
 </multipart-config>

</servlet>
```

multipartResolver Bean 등록

```
<context:component-scan
 base-package="org.zerock.controller" />

<beans:bean id="multipartResolver"

class="org.springframework.web.multipart.support.StandardServletMultimediaResolve
r">

</beans:bean>
```

## 4-2. Ajax 파일 업로드

파일 업로드 방식 : 주로 Ajax를 사용함 따라서, FormData

브라우저 에서 Ajax로 FormData를 전송

```
<div class='uploadDiv'>
 <input type='file' name='uploadFile' multiple>
</div>

<div class='uploadResult'>

```

```


</div>

<button id='uploadBtn'>Upload</button>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"
 integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
 crossorigin="anonymous"></script>

<script>
 $(document).ready(function(){
 var regex = new RegExp("(.*?)\.(exe|sh|zip|alz)$");
 var maxSize = 5242880; //5MB
 //boolean을 리턴하는 사전처리함수 (파일사이즈, 확장자검사해줌)
 function checkExtension(fileName, fileSize) {
 if (fileSize >= maxSize) {
 alert("파일 사이즈 초과");
 return false;
 }
 if (regex.test(fileName)) {
 alert("해당 종류의 파일은 업로드할 수 없습니다.");
 return false;
 }
 return true;
 }

 });
</script>

```

서버에서 처리

```

<Form>방식과 거의일
@PostMapping("/uploadFormAction")
public void uploadFormPost(MultipartFile[] uploadFile, Model model) {
 String uploadFolder = "C:\\upload";
 for (MultipartFile multipartFile : uploadFile) {
 log.info("-----");
 log.info("Upload File Name: " +
multipartFile.getOriginalFilename());
 log.info("Upload File Size: " + multipartFile.getSize());
 String uploadFileName = multipartFile.getOriginalFilename();//IE원 경
우 전체경로를 리턴해옴. // IE 일 경우 substring작업을 추가로 요함
 // uploadFileName =
uploadFileName.substring(uploadFileName.lastIndexOf("\\") + 1);
 File saveFile = new File(uploadFolder, uploadFileName);
 try {
 multipartFile.transferTo(saveFile);//업로드 진행
 } catch (Exception e) {
 log.error(e.getMessage());
 } // end catch
 } // end for
}

```