

# 목 차

## 파트 1 . 스프링 5 기본 환경설정 및 라이브러리 사용법

1. 스프링 5 MVC 시작 ----- 2 페이지
2. 스프링의 폼 전송객체 ----- 12 페이지
3. 세션과 인터셉터 ----- 17 페이지
4. 데이터베이스 사용과 MyBatis ----- 28 페이지
5. JSON 을 반환하는 Response 만들기 ----- 39 페이지
6. 뷰 템플릿 엔진 타임리프 사용법 ----- 40 페이지

## 파트 2. 스프링 5 를 사용해 웹사이트 제작하기

1. 프로젝트 준비 ----- 45 페이지
2. 메인페이지 레이아웃 만들기 ----- 55 페이지
3. 회원가입과 로그인구현하기 ----- 61 페이지
4. 인터셉터를 활용한 권한관리 ----- 90 페이지
5. 게시판 글쓰기 폼 구성하기 ----- 91 페이지
6. 게시판 검색 만들기 ----- 133 페이지

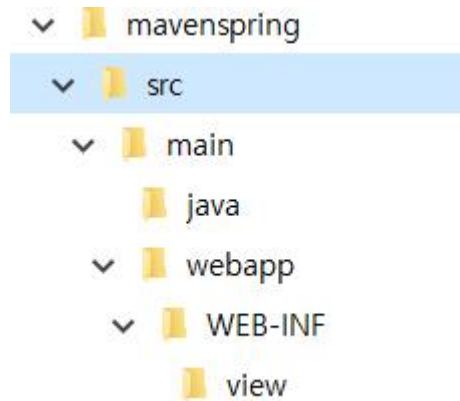
## 파트 3 웹사이트 완성도 올리기

1. 게시판 버튼들의 보완 ----- 141 페이지
2. 사용자 레벨업 시스템 만들기 ----- 144 페이지
3. 기타사항 정리하기 및 차후 개발사항 ----- 150 페이지

# **PART 1. 스프링 5 를 기본 환경설정 및 라이브러리 사용법**

## 1. 스프링5 MVC 시작

스프링 프로젝트로 사용할 폴더를 만들고 다음과 같이 폴더 구조를 만들어줍니다.



프로젝트 폴더명은 mavenspring이고 src/main/java는 소스코드를 작성할 폴더입니다. (테스트를 위해서는 src/test/java 도 만들어야 합니다.)

이제 mavenspring 폴더에 다음과 같이 pom.xml 을 만들어줍니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>net.gondr</groupId>
  <artifactId>mavenSpring</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <dependencies>

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

    <!-- jsp 2.3 -->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.2-b02</version>
      <scope>provided</scope>
    </dependency>

  </dependencies>
</project>
```

```

<!-- jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>

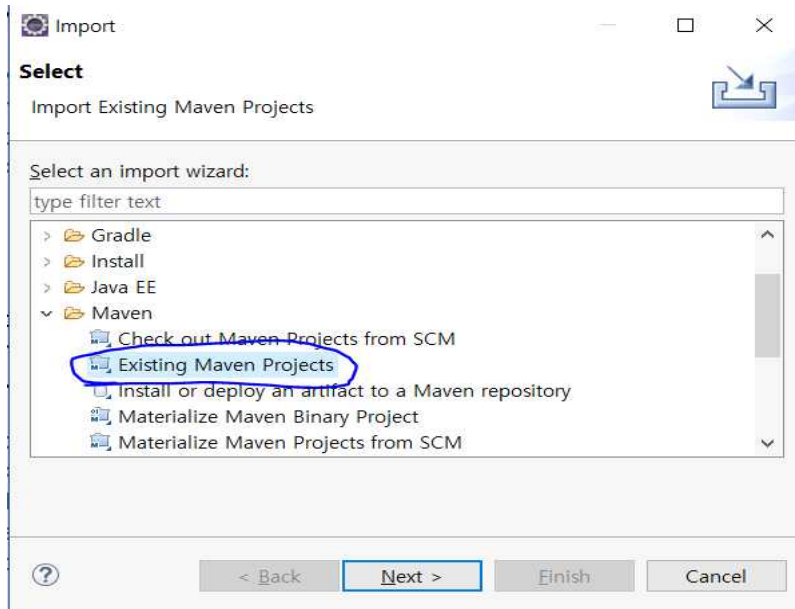
<!-- springframework -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.2.RELEASE</version>
</dependency>
</dependencies>

<!-- 빌드시 필요한 메이븐 플러그인-->
<build>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>utf-8</encoding>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

다 만들어졌다면 이클립스에서 이녀석을 maven 프로젝트로 임포트 해주면 됩니다.

[File] -> [Import...] 메뉴를 이용해서 다음과 같은 창에서 Existing maven project를 선택해주면 됩니다.



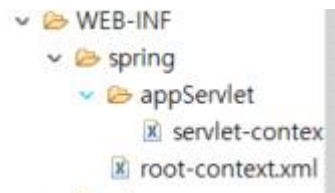
우리가 만든 mavenproject1 폴더를 선택하면 메이븐에 의해서 의존성들이 다운로드 되고 프로젝트 구성이 완료될 것입니다. 이클립스에서 로딩한 모습은 다음과 같습니다.



프로젝트에 톰캣 8.5 서버를 추가하고 웹모듈로서 추가해줍니다.

환경설정을 위해 2개의 파일과 폴더를 만들어주겠습니다. WEB-INF 밑에 spring 이라는 폴더를 만들고 그안에 appServlet 폴더를 만들어줍니다.

spring 폴더 안에는 root-context.xml 파일을 만들고 appServlet 폴더 안에는 servlet-context.xml 파일을 만들어줍니다. 완성된 모습은 다음과 같습니다.



각각의 파일의 내용은 다음과 같이 채워줍니다.

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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Root Context: 모든 웹컴포넌트에 공유될 빈객체들을 생성하는 곳 -->

</beans>
```

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"
             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">

    <!-- DispatcherServlet Context: 서블릿 디스패처가 요청을 처리하는 구조속에서
필요한 빈을 선언하는 곳 -->

    <!-- 스프링의 @Controller 를 비롯한 다양한 애노테이션이 동작할 수 있도록
Enable 해줌-->
    <annotation-driven />

    <!-- 웹애플릿/resources 로 오는 모든 요청을 /resources 경로로 연결해준다. -->
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Controller들이 보내는 반환 String 으로 만들어질 View Resolver 셋팅 -->
    <beans:bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/view/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <!-- Controller, Repository, Service 등을 검색할 패키지 -->
    <context:component-scan base-package="net.gondr.controller" />

</beans:beans>
```

각각의 파일들의 역할은 다음과 같습니다.

Root-context.xml : 웹어플리케이션 전반에서 사용될 스프링 Bean들을 생성해주는 곳

Servlet-context.xml : 사용자의 요청에 대한 처리과정에서의 로직에서 사용되는 Bean을 생성해주는 곳.

작업이 완료되면 이러한 환경설정하에 돌아가는 메인 디스패처 서블릿을 만들기 위해서 web.xml을 WEB-INF 폴더에 만들어주고 다음과 같이 코드를 작성합니다.

/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

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

    <!-- 전체 환경에 대한 설정파일 설정 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>

    <!-- 전체 환경에서 로드할 리스너 설정 -->
    <listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>

    <!-- URI에 따라 라우팅을 담당할 디스패처를 서블릿으로 등록함. -->
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>

    <!-- 사용자가 지정한 환경설정 클래스들의 위치를 지정해줌. -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring/appServlet/<u>servlet</u>-context.xml
        </param-value>
    </init-param>
```

```


    <!-- 가장 첫번째로 로드되도록 만들어줌. -->
    <load-on-startup>1</load-on-startup>
</servlet>

<!-- 서블릿을 /주소에 맵핑하여 모든 요청을 해당 서블릿이 받을 수 있도록 함. -->
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

<filter>
    <filter-name>encodingFilter</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>
</filter>
<!-- 모든 url에 해당 인코딩 필터가 동작하도록 맵핑 -->
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>

```

이제 요청을 처리할 컨트롤러를 만들어봅시다.  
 net.gondr.controller 패키지를 만들고 해당 패키지에  
 MainController 클래스를 만들어줍니다.



```

src/main/java
└── net.gondr.controller
    └── MainController.java
    
```

```

package net.gondr.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class MainController {
    @GetMapping("/")
    public String hello(Model model, @RequestParam(value="name",
required=false) String name ) {
        model.addAttribute("msg", "안녕하세요! " + name);
        return "index";
    }
}

```



```
}
```

이 클래스는 컨트롤러로 등록되었으며 uri 주소 "/" 로 GET 요청시 hello 라는 매서드가 수행되게 됩니다. hello 수행시에 name이라는 파라미터는 존재해도 되고 안해도 되며 존재시에는 자동으로 String 으로 형 변환되어 name이라는 변수에 넣어지게 됩니다.

이제 view를 만들어줄 차례입니다. 메인 컨트롤러가 index를 리턴하기 때문에 여기서도 index.jsp 를 view 폴더에 만들 것입니다. 다음과 같이 코딩합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>메인 페이지</title>
</head>
<body>
    <h1>서버 메시지</h1>
    <p>${msg}</p>
</body>
</html>
```

다 되었습니다. 이제 실행해볼까요?

접속하면 다음과 같이 메시지가 뜨고 null이라고 나옵니다.

## 서버 메시지

안녕하세요! null

그리고 접속 주소뒤에 다음과 같이 Get 파라미터 값을 보내면 해당 이름이 뜰 것입니다.

```
http://localhost:8080/?name=gondr
```

만약 넘길 값의 기본값을 지정하고 싶다면 다음과 같이 defaultValue 값을 지정하면 됩니다.

```
@GetMapping("/")
public String hello(Model model, @RequestParam(value="name", required=false,
defaultValue="gondr") String name ) {
    model.addAttribute("msg", "안녕하세요! " + name);
    return "index";
}
```

만약 다음과 같이 inter값을 받기로 하고 문자열을 보내면

```
@GetMapping("/")
public String hello(Model model, @RequestParam(value="name", required=false,
defaultValue="10") int name ) {
    model.addAttribute("msg", "안녕하세요! " + name);
    return "index";
}
```

## HTTP Status 400 – Bad Request

**Type** Status Report

**Description** The server cannot or will not process the request due to something that is perc

### Apache Tomcat/8.5.34

위와 같은 결과를 얻을 수 있습니다. 자동으로 형변환을 하기 때문에 숫자에 적합하지 않은 데이터를 보내면 변환할 때 익셉션이 발생하기 때문입니다. 이 익셉션을 올바르게 처리하려면 다음과 같이 MainController에 추가하면 됩니다.

net.gondr.controller/MainController.java

```
package net.gondr.controller;

import org.springframework.beans.TypeMismatchException;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class MainController {
    @GetMapping("/")
    public String hello(Model model, @RequestParam(value="name",
required=false, defaultValue="10") int name ) {
        model.addAttribute("msg", "안녕하세요! " + name);
        return "index";
    }

    @ExceptionHandler(TypeMismatchException.class)
    public String handleTypeMismatchException() {
        return "exception";
    }
}
```

이제 exception 페이지를 만들기 위해 view 폴더에 exception.jsp 를 다음과 같이 만들어줍니다.

/WEB-INF/view/exception.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>에러페이지</h1>
    <p>name 파라미터의 값으로는 오로지 숫자만이 들어갈 수 있습니다.</p>
</body>
</html>
```

이제 고의적으로 name파라미터에 문자를 넣으면 우리가 설정한 곳으로 오게됩니다.

이렇게 쉽게 익셉션의 처리가 가능합니다. 만약 모든 컨트롤러에서 동작하는 익셉션처리를 원한다면 다음과 같이 작성할 수 있습니다.

net.gondr.exception 패키지를 만들고 그 밑에 CommonExceptionHandler 를 만들어줍니다.

net.gondr.exception /CommonExceptionHandler

```
package net.gondr.exception;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice("net.gondr.controller")
public class CommonExceptionHandler {
    @ExceptionHandler(Exception.class)
    public String handleException() {
        return "globalException";
    }
}
```

위의 핸들러는 net.gondr.controller에 있는 모든 클래스와 그 하위 패키지의 클래스에서 Exception이 발생시 작동하게 됩니다. 이 클래스가 동작하도록 servlet-context.xml 에 다음과 같이 등록합니다.

```
<!-- 전역 익셉션 처리기 -->
<beans:bean class="net.gondr.exception.CommonExceptionHandler">
</beans:bean>
```

실험을 위해 controller에 있는 익셉션을 지워줍니다.

이제 익셉션을 발생시키면 에러페이지로 이동할 것입니다. (물론 jsp뷰가 없기 때문에 404 에러가 뜹습니다.)

GET방식이 아닌 /main/gondr 이렇게 넘겨주면 gondr이란 값이 name으로 들어가게 하고 싶다면 다음과 같이 코드를 작성하면 됩니다.

```
@GetMapping("/main/{name}")
public String hello(Model model, @PathVariable String name ) {
    model.addAttribute("msg", "안녕하세요! " + name);
    return "index";
}
```

주의 : 만약 위의 코드에서 /main으로 안하고 /{name} 등으로 했다면 /board 같은 url도 전부 name으로 맵핑되어 작동하지 않게 됩니다. 이를 주의해야합니다.

## 2. 스프링의 폼 전송 객체

회원가입하는 페이지를 만들고 스프링에서는 기존의 jsp와 다르게 어떤 것들을 사용할 수 있는지 보겠습니다. 다음과 같이 UserController를 net.gondr.controller에 추가합니다.

net.gondr.controller/UserController.java

```
package net.gondr.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/user/")
public class UserController {
    @RequestMapping(value="regist", method=RequestMethod.GET)
    public String viewRegistPage() {
        return "user/regist";
    }
}
```

별도로 이 컨트롤러를 활성화하지 않아도 servlet-context.xml에서 해당 패키지를 스캔하도록 해두었기 때문에 정상적으로 작동합니다.

이제 view 폴더에 user폴더를 만들고 그곳에 regist.jsp 를 다음과 같이 작성합니다.

WEB-INF/view/user/regist.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원가입 페이지</title>
</head>
<body>
    <h1>회원가입</h1>
    <form method="post">
        <input type="text" name="userid" />
        <input type="number" name="code"/>
    </form>
</body>
</html>
```

```

        <input type="submit" value="전송"/>
    </form>
</body>
</html>

```

이 값이 전송될 컨트롤러 값을 만들어야 합니다. UserController 에 다음과 같이 추가 매서드를 만들어서 post 전송을 받을 수 있습니다.

net.gondr.controller/UserController.java

```

@RequestMapping(value="regist", method=RequestMethod.POST)
public String registProcess(HttpServletRequest req) {

    System.out.println(req.getParameter("userid"));
    System.out.println(req.getParameter("code"));
    return "redirect:/";
}

```

이전에는 이처럼 request 객체로부터 직접 파라미터를 받아서 이를 VO객체로 만들어서 저장하고 리다이렉트 해주곤 했습니다.

하지만 이건 전송하는 값이 많아질수록 무척이나 귀찮고 코드의 길이를 늘이는 주범이 됩니다. 스프링에서는 이를 커맨드 객체라는 개념을 통해서 해결하고 있습니다. 커맨드 객체가 엄청난건 아닙니다. 넘어오는 값을 받을 VO객체를 만들고 이를 파라미터에 적어주기만하면 됩니다.

vo, dto 등을 만들기 위해서 net.gondr.domain 패키지를 만들고 그 안에 GondrVO 라는 vo객체를 만들어줍니다.

net.gondr.domain/GondrVO

```

package net.gondr.domain;

public class GondrVO {
    private String userid;
    private Integer code;

    public String getUserid() {
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    public Integer getCode() {
        return code;
    }
    public void setCode(Integer code) {

```

```

        this.code = code;
    }
}

```

(VO이름은 아무렇게나 해도 된다는 뜻에서 GondrVO로 지었습니다. 실제 프로젝트에서 저런 이름을 지으면 안됩니다.)

이제 UserController를 다음과 같이 변경합니다.

net.gondr.controller/UserController.java

```

@RequestMapping(value="regist", method=RequestMethod.POST)
public String registProcess( GondrVO gondrVO ) {

    System.out.println(gondrVO.getUserid());
    System.out.println(gondrVO.getCode());

    return "redirect:/";
}

```

이제 회원가입을 눌러주면 값이 정상적으로 VO객체에 들어가는 것을 볼 수 있습니다. 넘어오는 파라미터의 이름에 맞추어서 getter, setter를 자동으로 호출하여 값을 넣어주기 때문에 VO객체에 값이 채워지게 됩니다.

만약 이 값을 view에서 사용하고 싶다면 registProcess 를 다음과 같이 변경하여줍니다.

net.gondr.controller/UserController.java

```

@RequestMapping(value="regist", method=RequestMethod.POST)
public String registProcess( GondrVO gondrVO ) {

    System.out.println(gondrVO.getUserid());
    System.out.println(gondrVO.getCode());

    return "user/registok";
}

```

이제 registok.jsp 파일을 user 폴더에 다음과 같이 만들어줍니다.

/WEB-INF/view/user/registok.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

```

```

        입력한 id : ${gondrVO.userid} <br>
        코드 : ${gondrVO.code } <br>
    </body>
</html>

```

기본적으로 커맨드 객체는 그냥 써주기만하면 request객체에 실려서 뷰로 넘어갑니다.

만약 name 이 빈값이거나 올바르지 않은 값이 들어왔을 때는 걸러주는 로직도 필요합니다. 일반적으로 이를 하기 위해서는 if문으로 들어온 값을 비교해야 합니다. 하지만 스프링에서는 이를 조금 더 편하게 수행할 수 있는 방법이 있습니다.

스프링에서 MVC 커맨드 객체의 값이 올바른지 검사하는 2개의 인터페이스가 있습니다.

org.springframework.validation.Validator

org.springframework.validation.Errors

이 인터페이스를 사용하는 구현 클래스를 net.gondr.validator 패키지를 만들고 그안에 GondrValidator를 다음과 같이 작성합니다.

net.gondr.validator/GondrValidator.java

```

package net.gondr.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import net.gondr.domain.GondrVO;

public class GondrValidator implements Validator {
    @Override
    public boolean supports(Class<?> clazz) {
        return GondrVO.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "userid",
"required", "유저 아이디 값은 필수입니다.");

        GondrVO vo = (GondrVO) target;

        if (vo.getCode() < 0 ) {
            errors.rejectValue("code", "bad", "0이하의 숫자는 올 수
없습니다");
        }
    }
}

```



2가지의 검사를 수행하는 벨리데이션 코드입니다. 첫번째는 userid가 비었거나 공백인지 검사하고 code값이 음수인지 검사합니다.

중요한 점은 반드시 supports 매서드를 통해서 검사할 VO클래스를 올바르게 지정해주어야 스프링에서 자동 validation을 수행할 수 있습니다.

UserController를 다음과 같이 변경합니다.

<net.gondr.controller/UserController.java>

```
@RequestMapping(value="regist", method=RequestMethod.POST)
public String registProcess(GondrVO gondrVO, Errors errors, Model model ) {
    new GondrValidator().validate(gondrVO, errors);
    if(errors.hasErrors()) {
        model.addAttribute("errors", errors);
        return "error"; //에러가 존재할시 error페이지로 이동
    }

    return "user/registok";
}
```

이제 다시한번 서버를 작동하고 비어있는 아이디로 입력해봅시다.

## HTTP Status 404 – Not Found

**Type** Status Report

**Message** /WEB-INF/view/error.jsp

**Description** The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.

Apache Tomcat/8.5.34

위와 같이 error.jsp로 이동하게 됩니다. 하지만 아직 뷰가 없기 때문에 notfound가 나오게 되죠.

이제 errors.jsp를 보도록 합시다.

</WEB-INF/view/error.jsp>

```
<%@ 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">
<title>Insert title here</title>
</head>
<body>
    <h2>에러 페이지</h2>
    <c:if test="${errors.hasFieldErrors('userid')}">
        <c:out
value="${errors.getFieldError('userid').defaultMessage}"></c:out><br>
    </c:if>

    <c:if test="${errors.hasFieldErrors('code')}">
        <c:out
value="${errors.getFieldError('code').defaultMessage}"></c:out><br>
    </c:if>

</body>
</html>
```

이제 정상적으로 나오게 됩니다.

### 3. 세션과 인터셉터

사용자 로그인을 처리하기 위한 VO 몇가지를 domain에 추가하도록 하겠습니다.

net.gondr.domain/UserVO.java

```
package net.gondr.domain;

public class UserVO {
    private String userid;
    private String username;
    private String password;

    public String getUserid() {
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

로그인 화면을 만들기 위해서 UserController에 다음과 같이 코드를 추가합니다.

net.gondr.controller/UserController.java

```
@RequestMapping(value="login", method=RequestMethod.GET)
public String viewLoginPage() {
    return "user/login";
}

@RequestMapping(value="login", method=RequestMethod.POST)
public String loginProcess() {
    return "redirect:/";
}
```

```
}
```

로그인 뷰페이지를 만들기 위해서 view 폴더에 login 페이지를 다음과 같이 만들어줍니다.

```
</WEB-INF/view/user/login.jsp>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>로그인 페이지</h1>
    <form method="post">
        <input type="text" name="userid"/><br/>
        <input type="password" name="password"/><br/>
        <input type="submit" value="전송"/>
    </form>
</body>
</html>
```

그리고 이 요청을 받아줄 수 있도록 UserController의 loginProcess메서드를 다음과 같이 수정합니다.

```
@RequestMapping(value="login", method=RequestMethod.POST)
public String loginProcess(UserVO user, HttpSession session) {
    if(user.getUserid().equals("gondr") && user.getPassword().equals("1234"))
    {
        session.setAttribute("user", user);
        //로그인 성공시 메인 화면으로 이동
        return "redirect:/";
    }else {
        //실패시 다시 로그인 화면으로 이동
        return "redirect:/user/login";
    }
}
```

HttpSession 을 파라미터에 추가하기만 하면 세션이 자동으로 주입되어 들어옵니다. 이는 스프링에서 Controller로 지정되어 있는 클래스에 메서드들에 알려진 오브젝트들이 파라미터로 존재하면 자동으로 인젝션을 해줍니다.

꽤 간단하게 세션을 이용할 수 있습니다. 물론 코드를 다음과 같이 작성해도 됩니다.

```
@RequestMapping(value="login", method=RequestMethod.POST)
```

```

public String loginProcess(UserVO user, HttpServletRequest req) {
    if(user.getUserid().equals("gondr") && user.getPassword().equals("1234"))
    {
        HttpSession session = req.getSession();
        session.setAttribute("user", user);
        //로그인 성공시 메인 화면으로 이동
        return "redirect:/";
    }else {
        //실패시 다시 로그인 화면으로 이동
        return "redirect:/user/login";
    }
}
}

```

이 방법은 장점은 로그인되지 않았을 경우에는 세션정보를 생성하지 않는다는 것입니다. 큰 차이는 없기 때문에 편한 방법인 1번을 사용해도 아무 문제 없습니다.

이제 세션의 유무에 따라서 메인 페이지가 달라지도록 해보겠습니다.

</WEB-INF/view/index.jsp>

```

<%@ 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">
<title>메인 페이지</title>
</head>
<body>
    <h1>서버 메시지</h1>
    <p>${msg}</p>

    <c:if test="${! empty user}">
        <p>${user.userid} 님 환영합니다.</p>
        <span><a href="/user/logout">[로그아웃]</a></span>
    </c:if>

    <c:if test="${empty user }">
        <p>로그인 하시려면 로그인 페이지로 이동하세요</p>
        <span><a href="/user/login">[로그인]</a></span>
    </c:if>
</body>
</html>

```

이제 로그아웃만 구현하면 됩니다. UserController에 다음과 같이 매서드를 추가하여 세션에 있는 user값을 지워줍니다.

```

@RequestMapping(value="logout", method=RequestMethod.GET)
public String logoutProcess(HttpSession session) {
    if(session.getAttribute("user") != null) {
        session.removeAttribute("user");
        return "redirect:/";
    }else {
        return "redirect:/user/login";
    }
}

```

물론 session.invalidate 매서드를 사용하여 세션에 있는 정보를 모조리 지워버려도 상관없습니다. 유저정보를 볼 수 있는 user/info 페이지를 만들고 로그인한 유저만 여기에 접근할 수 있도록 해 보겠습니다.

먼저 usercontroller에 다음과 같이 viewInfopage 매서드를 추가합니다.

net.gondr.controller/UserController.java

```

@RequestMapping(value="info", method=RequestMethod.GET)
public String viewInfoPage(HttpSession session){
    if(session.getAttribute("user") == null) {
        //로그인되지 않은 유저는 로그인 페이지로 리다이렉트!
        return "redirect:/user/login";
    }
    return "user/info";
}

```

회원정보 페이지를 view폴더에 다음과 같이 만들어줍니다.

/WEB-INF/view/user/info.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h2>유저 정보 페이지</h2>

    아이디 : <span>${user.userid}</span><br/>
    이름 : <span>${user.username }</span><br/>
</body>
</html>

```

물론 지금은 이름은 없기 때문에 나오지 않을 것입니다. 앞으로 웹사이트를 만들면서 지속적으로 로그인 인증이 필요한 페이지들이 나올 것입니다. 그렇다면 그 모든 페이지마다 이런식으로 if문을 만들어서 걸러주는 것은 코드의 중복이며, 실수로 한군데에 안만든다면 문제가 발생하게 됩니

다. 따라서 요청을 중간에서 가로채서 보안을 관리하는 모듈이 필요합니다. 이를 스프링에서는 `Interceptor`라고 부릅니다.

인터셉터는 기본적으로 `HttpServletRequest`를 중간에 가로채기 위해 만들어져 있기 때문에 시점별로 3개의 매서드를 가지고 있습니다.

요청 처리전에 검사하는 `preHandle`과 요청 수행후에 검사하는 `postHandle` 사용자에게 View를 전송하고 눈 후 처리해야할 `afterCompletion` 입니다.

`net.gondr.interceptor` 패키지를 만들고 그안에 `AuthInterceptor` 를 다음과 같이 작성합니다.

`net.gondr.interceptor/AuthInterceptor.java`

```
package net.gondr.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.web.servlet.HandlerInterceptor;

public class AuthInterceptor implements HandlerInterceptor{
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
        throws Exception {
        HttpSession session = request.getSession();
        if(session != null ) {
            Object userVO = session.getAttribute("user");
            if(userVO != null) {
                return true;
            }
        }
        //로그인되지 않은 경우
        response.sendRedirect("/user/login");
        return false;
    }
}
```

위의 인터셉터는 3가지 중에 `preHandle`을 사용했으며 세션에 정보가 없을 경우 `false`를 리턴하여 더 이상 진행되지 않도록 막았습니다. 또한 실패한경우는 로그인 페이지로 이동하게 했습니다.

이제 이 인터셉터를 적용할 url을 지정해야 합니다. 이는 `servlet-context.xml` 에서 처리하도록 하겠습니다. 다음과 같이 인터셉터를 bean에 추가합니다.

`/WEB-INF/spring/appServlet/servlet-context.xml`

```
<!-- 인터셉터 등록 -->
```

```

<beans:bean id="authInterceptor"
class="net.gondr.interceptor.AuthInterceptor"></beans:bean>

<!-- 인터셉터 매핑 -->
<interceptors>
    <interceptor>
        <mapping path="/user/info"/>
        <beans:ref bean="authInterceptor"/>
    </interceptor>
</interceptors>

```

AuthInterceptor를 별도의 빈으로 등록해서 매핑해주었습니다.

net.gondr.controller/UserController

```

@RequestMapping(value="info", method=RequestMethod.GET)
public String viewInfoPage(HttpSession session){
    return "user/info";
}

```

동작은 정상적으로 이루어집니다. 만약 추가적으로 경로를 지정한다면 다음과 같이 만들어주면 됩니다.

```

<!-- 인터셉터 등록 -->
<beans:bean id="authInterceptor"
class="net.gondr.interceptor.AuthInterceptor"></beans:bean>

<!-- 인터셉터 매핑 -->
<interceptors>
    <interceptor>
        <mapping path="/user/info"/>
        <mapping path="/board/**"/>
        <exclude-mapping path="/board/list"/>
        <exclude-mapping path="/board/view/*/"/>
        <beans:ref bean="authInterceptor"/>
    </interceptor>
</interceptors>

```

위의 코드는 /board 로 시작하는 아래 경로 모두를 로그인 검사를 하되 list, view 는 제외하는 것을 말합니다.

\*\* 는 하위 폴더를 포함한 모든 폴더와 텍스트를 의미합니다.

\*는 0개 또는 그 이상의 글자를 말합니다.

?: 1개 글자를 의미합니다.

이 3개의 패턴들로 원하는 url 에 인터셉터를 적용하는 것이 가능합니다. 반복적으로 나오는 인증



검사 로직을 이것으로 크게 단축시킬 수 있습니다. (정말 간단하죠?)

마지막으로 최근에는 로컬스토리지로 인해 그 쓰임새가 거의 사라져가고 있는 쿠키에 대해서 알아보겠습니다.

쿠키를 통해 아이디를 기억하게 만드는 기능을 만들어보겠습니다. 다음과 같이 로그인 폼을 정정합니다.

/WEB-INF/view/user/login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>로그인 페이지</h1>
    <form method="post">
        <input type="text" name="userid"/><br/>
        <input type="password" name="password"/><br/>
        <input type="checkbox" name="remember"/> 이메일 기억하기 <br/>
        <input type="submit" value="전송"/>
    </form>
</body>
</html>
```

이제 이 정보를 받아줄 UserController를 조금 변경해야 합니다. 왜냐하면 UserVO 에는 remember라는 멤버변수는 없기 때문입니다. 그렇다고 UserVO에 그것을 만들기는 애매합니다. UserVO는 말그대로 유저 정보 데이터인데 거기에 remember라는 값을 넣는 것은 맞지 않기 때문입니다. 따라서 화면전송 객체와 DB 저장객체를 분리할 필요가 있습니다. 그래서 이경우 화면 전송 객체를 DTO라고 부르고 DB 저장객체를 VO라고 부릅니다. LoginDTO를 다음과 같이 만들어봅시다.

net.gondr.domain/LoginDTO.java

```
package net.gondr.domain;

public class LoginDTO {
    private String userid;
    private String password;
    private boolean remember;
    public String getUserid() {
        return userid;
    }
}
```

```

    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public boolean isRemember() {
        return remember;
    }
    public void setRemember(boolean remember) {
        this.remember = remember;
    }
}
}

```

그리고 UserController의 loginProcess 부분을 다음과 같이 변경합니다.

```

@RequestMapping(value="login", method=RequestMethod.POST)
public String loginProcess(LoginDTO login, HttpSession session) {
    if(login.getUserid().equals("gondr") &&
    login.getPassword().equals("1234")) {
        //이 부분은 향후 데이터베이스 로직으로 변경됩니다.
        UserVO user = new UserVO();
        user.setUserid(login.getUserid());
        user.setUsername("최선탄");
        //여기까지 데이터베이스 로직으로 변경 예정

        session.setAttribute("user", user);
        //로그인 성공시 메인 화면으로 이동
        return "redirect:/";
    }else {
        //실패시 다시 로그인 화면으로 이동
        return "redirect:/user/login";
    }
}
}

```

이 상태에서 쿠키를 설정하여 로그인 성공시에는 아이디를 기억하도록 하겠습니다. 먼저 매서드에 다음과 같이 HttpServletResponse 객체를 추가하여 받아줍니다.

net.gondr.controller/UserController.java

```

@RequestMapping(value="login", method=RequestMethod.POST)
public String loginProcess(LoginDTO login, HttpSession session,
HttpServletResponse res) {

```

```

        if(login.getUserid().equals("gondr") &&
login.getPassword().equals("1234")) {
            //이 부분은 향후 데이터베이스 로직으로 변경됩니다.
            UserVO user = new UserVO();
            user.setUserid(login.getUserid());
            user.setUsername("최선훈");
            //여기까지 데이터베이스 로직으로 변경 예정

            Cookie cookie = new Cookie("REMEMBER", login.getUserid());
            cookie.setPath("/");

            if(login.isRemember()) {
                cookie.setMaxAge(60 * 60 * 24 * 30); //30일간 기억
            }else {
                cookie.setMaxAge(0); //쿠키 삭제
            }
            res.addCookie(cookie);

            session.setAttribute("user", user);
            //로그인 성공시 메인 화면으로 이동
            return "redirect:/";
        }else {
            //실패시 다시 로그인 화면으로 이동
            return "redirect:/user/login";
        }
    }
}

```

로그인시 기억버튼을 클릭하면 쿠키를 설정하고 그렇지 않으면 기존 쿠키의 지속시간을 0으로 바꾸는 코드입니다. 이제 로그인후 기억체크후 실제로 쿠키가 생성되는 지를 크롬 개발자도구를 통해 확인합시다.

Name	Value
JSESSIONID	A88EDA7B1EA38679043C9F88A0811E59
REMEMBER	gondr

이 쿠키 값을 로그인 페이지에서 읽어서 처리하도록 하려면 다음과 같이 viewLoginPage매서드를 수정하면 됩니다.

net.gondr.controller/UserController.java

```

@RequestMapping(value="login", method=RequestMethod.GET)
public String viewLoginPage(Model model, @CookieValue(value="REMEMBER",
required=false)Cookie cookie) {
    if(cookie != null) {
        model.addAttribute("email", cookie.getValue());
    }
    return "user/login";
}

```

```
}
```

그리고 로그인 페이지인 login.jsp를 다음과 같이 만들어줍니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>로그인 페이지</h1>
    <form method="post">
        <input type="text" name="userid" value="${email}"/><br/>
        <input type="password" name="password"/><br/>
        <input type="checkbox" name="remember"/> 이메일 기억하기 <br/>
        <input type="submit" value="전송"/>
    </form>
</body>
</html>
```

(만들다보니 습관처럼 userid 대신 email을 써버렸습니다. -\_-; 그러려니 하고 넘어가기 바랍니다.)

이제 기억후 들어가면 아이디가 기억되어 나오는 것을 볼 수 있습니다.

## 4. 데이터베이스 사용과 MyBatis

데이터베이스 사용에 필요한 의존성들을 pom.xml 에 다음과 같이 추가합니다.

<pom.xml>

```
<!-- 스프링 jdbc 관련 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.1.2.RELEASE</version>
</dependency>

<!-- mysql 연결관련 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>6.0.6</version>
</dependency>

<!-- mybatis 사용 관련 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.6</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.2</version>
</dependency>

<!-- JUnit 테스트를 위한 라이브러리 -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

이제 가져온 라이브러리들을 셋팅해야합니다. 데이터베이스 테이블을 만드는 것부터 시작해보시다.

데이터베이스에 접속하여 회원을 생성하는 것을 연습해보시다. 이를 위해 먼저 spring\_users 테이블을 다음과 같이 작성합니다.

#	이름	종류	데이터정렬방식	모기	Null	기본 값	설명	수가
1	userid	varchar(100)	utf8mb4_general_ci		아니오	없음		
2	username	varchar(100)	utf8mb4_general_ci		아니오	없음		
3	password	varchar(100)	utf8mb4_general_ci		아니오	없음		

데이터베이스에 연결할 수 있도록 데이터 소스를 만들어야 합니다. 다음과 같이 데이터 소스를 root-context.xml 에 추가합니다.

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

    <!-- Root Context: 모든 웹컴포넌트에 공유될 빈객체들을 생성하는 곳 -->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"
value="com.mysql.cj.jdbc.Driver"></property>
        <property name="url"
value="jdbc:mysql://gondr.asuscomm.com/devdb?useSSL=false&serverTimezone=Asia/Seoul"></property>
        <property name="username" value="devdb"></property>
        <property name="password" value="1234"></property>
    </bean>
</beans>
```

XML에서는 &가 특수기호이기 때문에 이를 표현하기 위해 &amp; 를 사용하였습니다.

우리가 DB연결설정을 올바르게 적었는지를 테스트하려면 이제부터 컨트롤러를 만들고 결과를 보여줄 뷰도 만들고 요청을 받을 뷰도 만들어야 합니다. 그저 오타없이 DataSource를 잘 찾는지 확인하기 위해 해야할 일들이 너무 많습니다. 이를 쉽게 해결해주는 것이 유닛단위 테스트를 지원하는 JUnit입니다.

Src/test/java 에 net.gondr.test 패키지를 만들고 그곳에 DataSourceTest 클래스를 다음과 같이 작성합니다.

src/test/java/net.gondr.test/DataSourceTest.java(앞으로 테스트 폴더는 전부 test로 단축하여표시합니다.)

```
package net.gondr.test;

import java.sql.Connection;
```

```

import javax.sql.DataSource;

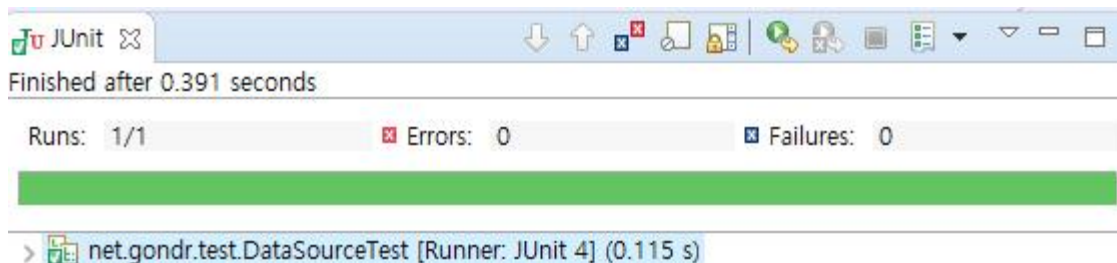
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations=
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})
public class DataSourceTest {
    @Autowired
    private DataSource ds;

    @Test
    public void testConnection() throws Exception {
        try {
            Connection con = ds.getConnection();
            //연결이 잘되었는지 출력해봄.
            System.out.println(con);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

이제 이 파일에서 우클릭 하고 Run as -> Junit 테스트를 클릭하면 테스트된 결과가 junit창에 띄워집니다. 정상적으로 잘 작성했다면 다음과 같은 모습을 보입니다.



그리고 콘솔창에는 다음과 같이 데이터 연결이 찍혔을 것입니다.

```
com.mysql.cj.jdbc.ConnectionImpl@73a1e9a9
```

여기까지 되었다면 데이터 소스 설정은 올바르게 이루어졌습니다.

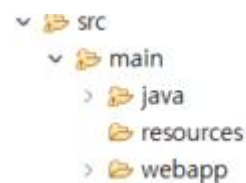
아마도 상단에 있는 Runwith, ContextConfiguration, Autowired 의 역할이 궁금할 것입니다. 앞의 2가지는 Junit 테스트에 관련된 것이고 마지막 3번째는 스프링의 의존성 주입과 관련되어있습니다.

RunWith 해당 테스트를 수행하는데 있어서 함께 실행되어야 하는 클래스를 지정하는 것으로 지정된 클래스는 테스트 수행전에 메모리에 로드되어 Junit 테스트에 사용됩니다. (즉 Junit의 Test 기능이 SpringJUnit4ClassRunner를 사용합니다.) ContextConfiguration은 테스트시 구동되는 스프링환경에서 로드되어야 할 환경설정 파일을지정합니다.(또는 클래스를 지정하기도합니다.) 이에 따라 해당 테스트 수행전에 SpringJUnit4ClassRunner 클래스를 로딩하고 환경설정 파일로 root-context.xml 을 이용하여 실행환경 셋팅후 Test가 지정된 매서드를 실행하는 것입니다.

마지막으로 Autowired는 스프링에 등록된 Bean중에서 해당 타입에 가장 적합한 인스턴스를 자동으로 주입시켜주는 역할을 합니다. 이 때 주의해야할 사항은 무엇을 주입해야할지 모르는 상황을 만들어서는 안된다는 것입니다.

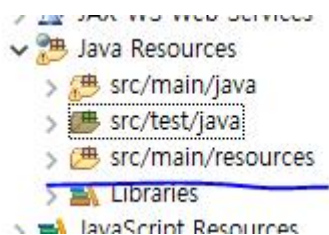
예를 들어 List 형식의 변수에 Autowired를 작성해놓고 Bean으로 ArrayList와 LinkedList를 만들게 되면 스프링입장에서는 2개의 빈 모두 List형식에 들어갈 수 있기 때문에 모호한 주입상황에 부딪히게 되고 이때는 익셉션을 내면서 로딩을 중지하게 됩니다.

이제 마이바티스 빈을 설정해봅시다. 빈설정에 앞서서 MyBatis의 환경설정 파일을 만들어야 합니다. 일반적으로는 src/main/resources 폴더에 작성합니다. 다만 우리는 초기단계에서 이를 만들지 않았습니다. 따라서 다음과 같이 만들어주면 됩니다.



하지만 아직은 해당 내용이 클래스패스에 등록된 상태가 아닙니다.

간단하게 프로젝트에 우클릭하고 maven -> update project를 한번 클릭하면 메이븐 구조에 맞게 resource를 클래스패스에 넣어줍니다. 그 결과는 아래와 같이 나오게 됩니다.



좌측의 모습이 메이븐 또는 그라들 프로젝트의 기본적인 모습입니다.

resources 패키지에 mybatis-config.xml 파일을 다음과 같이 만들어줍니다.

src/main/resource/mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
```



```
<!-- 아직 환경설정은 작성할 것이 없으니 비워둡니다. -->
</configuration>
```

그리고 이 환경설정을 로드해 만들어지는 마이바티스 빈을 다음과 같이 root-context.xml에 만들어줍니다.

/WEB-INF/spring/root-context.xml

```
<!-- sqlSessionFactroy -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation"
value="classpath:/mybatis-config.xml"></property>
</bean>
```

이제 세션팩토리가 잘 나오는지 확인해보기 위한 테스트 코드를 작성해봅시다. net.gondr.test 패키지에 MybatisTest.java 파일을 다음과 같이 작성합니다.

test/net.gondr.test/MybatisTest

```
package net.gondr.test;

import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations=
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})
public class MybatisTest {
    @Autowired
    private SqlSessionFactory factory;

    @Test
    public void testFactory() {
        //SQL 연결 공장이 올바르게 주입되었는지 확인하는 매서드
        System.out.println(factory);
    }

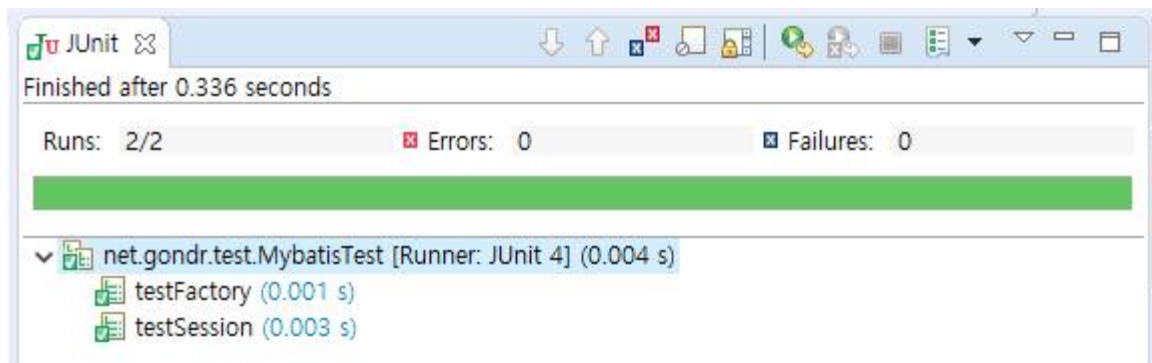
    @Test
    public void testSession() {
        //실제 데이터베이스 연결이 올바르게 출력되는지를 확인하는 매서드
        try {
            SqlSession session = factory.openSession();
            System.out.println(session);
        }
    }
}
```

```

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

테스트를 수행해봅시다.



```

org.apache.ibatis.session.defaults.DefaultSqlSessionFactory@5b8dfcc1
org.apache.ibatis.session.defaults.DefaultSqlSession@689604d9

```

위와 같이 테스트가 수행되면 성공적으로 수행된 것입니다.

마이바티스 매퍼는 XML로 설정할 때는 매퍼를 한곳에 넣어두고 sqlSessionSessionFactoryBean을 만들 때 property로 mapperLocations 값으로 classpath를 넣어주면됩니다. 다음과 같이 말이죠.

```

<!-- sqlSessionSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation"
value="classpath:/mybatis-config.xml"></property>
    <property name="mapperLocations"
value="classpath:/mappers/**/*.Mapper.xml"></property>
</bean>

```

위와 같이 작성하면 resources 패키지 안에 있는 mappers패키지 안에 있는 Mapper.xml로 끝나는 모든 파일을 매퍼로 등록하게 됩니다.

그외에도 인터페이스로 매퍼를 설정할 수도 있습니다. 인터페이스로 매퍼를 설정하는 방법은 다음 MyBatis-Spring 페이지(<http://www.mybatis.org/spring/ko/getting-started.html>)를 참고하여 공부해볼만합니다. 고맙게도 누군가가 한글화를 마쳐둔 상태입니다.

다만 2018.11월 현재 대부분의 인터넷의 글들은 XML기반으로 설정을 하고 있기 때문에 이 강의에서도 XML기반으로 작성하도록 하겠습니다.

사용자 매퍼를 만들어봅시다. resources 패키지에 mappers 패키지를 만들고 그안에 다음과 같이 UserMapper.xml 을 만들어줍니다.

src/main/resources/mappers/UserMapper.xml

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

<mapper namespace="net.gondr.mappers.UserMapper">
  <insert id="insertUser">
    INSERT INTO spring_users (userid, username, password) VALUES (
    #{userid}, #{username}, #{password})
  </insert>
</mapper>
```

추가적으로 SqlSession 템플릿 빈을 하나 추가해줍니다. 이 빈은 팩토리부터 연결을 받아와서 만들어지는 데이터베이스 세션을 말합니다. 다만 우리는 여기서 MyBatis의 기본 클래스인 SqlSessionTemplate를 활용해서 다 사용한 세션을 별다른 동작없이도 알아서 클로징을 해주도록 할 것입니다. root-context.xml에 다음과 같이 빈을 추가합니다.

```
<!-- sqlSession : 팩토리로부터 연결을 가져와서 사용후 자동으로 반환까지 해줌. -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"
  destroy-method="clearCache">
  <!-- 빈생성시 파라미터로 sqlSessionSessionFactory빈을 넘겨줌. -->
  <constructor-arg name="sqlSessionFactory"
  ref="sqlSessionFactory"></constructor-arg>
</bean>
```

이제 openSession을 호출해줄 필요도 없이 바로 session을 주입해줄 수 있는 준비가 끝났습니다.

DB 서버에 연결해서 데이터를 가져올 DAO를 만들기 위해 net.gondr.dao 패키지를 만들고 그안에 UserDAO.java를 다음과 같이 작성합니다.

```
package net.gondr.dao;

import net.gondr.domain.UserVO;

public interface UserDAO {

    public void insertUser(UserVO user);

}
```

그리고 이 인터페이스를 구현한 UserDaoImpl.java를 net.gondr.dao 패키지에 다음과 같이 작성합니다.

net.gondr.dao/UserDaoImpl.java

```
package net.gondr.dao;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import net.gondr.domain.UserVO;

@Repository
public class UserDaoImpl implements UserDao {
    @Autowired
    private SqlSession session;

    private static final String namespace = "net.gondr.mappers.UserMapper";

    @Override
    public void insertUser(UserVO user) {
        session.insert(namespace + ".insertUser", user);
    }
}
```

session 변수는 Autowired를 사용하여 스프링으로부터 우리가 등록한 템플릿이 바로 주입되게 되고 여기에 insert를 호출하면 자동으로 알맞은 세션하나를 open하여 insert 작업을 수행하게 됩니다. 프로그래머는 이 과정에서 아무것도 신경쓰지 않아도 됩니다.

상단에 Repository는 해당 클래스가 DAO 객체라는 것을 스프링에 알려주는 역할입니다. 정확히는 스프링은 Repository, Service, Controller 전부 Component로 보고 Spring Bean으로 등록합니다. 다만 그 역할에 있어서는 구분해서 사용할 것을 권장하고 있습니다.

또한 3가지를 바꾸어 사용해도 정상적으로 작동은 하나 DAO에 Repository대신 Component나 Service, Controller를 사용시 DAO에서 발생하는 특정 예외들을 잡을 수 없는 현상이 발생할 수도 있습니다.

그리고 마지막으로 굳이 UserDao만 만들면 되는데 왜 UserDaoImpl을 만드는데 대해서 의문이 들 것입니다.

인터페이스를 중간의 매개체로 두게되면 2개의 객체간의 결합도가 낮아지게 됩니다. 낮은 결합도

는 향후 프로그램의 교체 및 수정 비용을 줄이게 되는데 결정적인 역할을 합니다. UserDaoImpl의 변경이 UserDao를 사용하는 다른 모든 객체에 영향을 전혀 주지 않게 되는거죠. 변화로 인한 부작용도 없앨 수 있게 되구요. 따라서 객체와 객체의 연결을 만들때는 인터페이스로 만드는 것이 프로그래밍의 층을 만들어낼 수 있는 방법입니다.

(아마도 이 말을 온전히 이해할 수 있게 된다면 어느정도 프로그래밍에 경험이 쌓이고, 객체지향도 많은 부분이 이해가 된 걸꺼예요. 당장 이해가 안간다고 좌절할 필요는 전혀 없습니다. 걱정말아요.)

dao들을 스프링이 빈으로 읽어들이고 등록할 수 있도록 root-context.xml에 다음과 같이 넣어줍니다. (이것을 root-context.xml에 넣는 이유는 DAO는 웹로직과 관계없이 API 로직에서 활용될 수 있기 때문입니다. 즉 사용자의 요청에 따라 움직이지 않고 일정시간마다 서버의 데이터를 저장하는 등의 로직에도 사용되기 때문입니다.)

WEB-INF/spring/root-context.xml

```
<context:component-scan base-package="net.gondr.dao"></context:component-scan>
```

아마도 이 내용을 치고나면 예러가 나면서 붉은 줄이 그어질 것입니다.

servlet-context.xml과 비교해보면 차이점을 바로 알 수 있습니다. context에 대한 xmlns가 추가되지 않았기 때문입니다. root-contxt.xml의 상단의 네임스페이스에 다음과 같이 spring의 context 관련 ns를 추가합니다.

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:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="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">

  <!-- Root Context: 모든 웹컴포넌트에 공유될 빈객체들을 생성하는 곳 -->
  <bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"
      value="com.mysql.cj.jdbc.Driver"></property>
    <property name="url"
      value="jdbc:mysql://gondr.asuscomm.com/devdb?useSSL=false&serverTimezone=Asia/Seoul"></property>
    <property name="username" value="devdb"></property>
    <property name="password" value="1234"></property>
```

```

        </bean>

        <!-- sqlSessionFactory -->
        <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
            <property name="dataSource" ref="dataSource" />
            <property name="configLocation"
value="classpath:/mybatis-config.xml"></property>
            <property name="mapperLocations"
value="classpath:/mappers/**/*.xml"></property>
        </bean>

        <!-- sqlSession : 팩토리로부터 연결을 가져와서 사용후 자동으로 반환까지 해줌.
-->
        <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"
destroy-method="clearCache">
            <!-- 빈생성시 파라미터로 sqlSessionFactory빈을 넘겨줌. -->
            <constructor-arg name="sqlSessionFactory"
ref="sqlSessionFactory"></constructor-arg>
        </bean>

        <context:component-scan
base-package="net.gondr.dao"></context:component-scan>
    </beans>

```

위와같이 네임스페이스를 추가해주면 에러가 사라질 것입니다.

이제 이 DAO를 테스트해봅시다. net.gondr.test 패키지에 다음과 같이 UserDaoTest.java를 작성합니다.

```

package net.gondr.test;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import net.gondr.dao.UserDAO;
import net.gondr.domain.UserVO;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{"file:src/main/webapp/WEB-INF/**/*.root-context.xml"})
public class UserDaoTest {
    @Autowired
    private UserDAO dao;

    @Test
    public void testInsertUser() {
        UserVO user = new UserVO();
    }
}

```

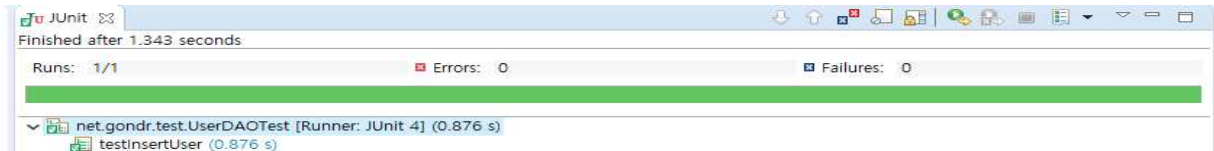
```

        user.setUserid("gondr");
        user.setPassword("1234");
        user.setUsername("최선햀");

        dao.insertUser(user);
    }
}

```

테스트를 수행하면 다음과 같이 나올 것입니다. (여러 번 수행하면 예러가 납니다. userid가 primary키이기 때문입니다.)



그리고 데이터베이스를 확인해보면 다음과 같이 데이터가 삽입된 것을 확인할 수 있습니다.

+ 옵션

	userid	username	password
수정	gondr	최선햀	1234

이것으로 데이터베이스 연결은 마무리 됩니다. (이후 프로젝트에서 제대로 다룹니다)

## 5. JSON을 반환하는 Response 만들기

일반적인 웹페이지는 viewpage를 반환하는 것이 일반적이거나 RESTAPI의 경우 JSON형태의 데이터를 반환하게 됩니다.

VO 객체를 JSON 으로 전달하는 매서드를 만들어보겠습니다. 먼저 다음과 같이 UserController 에 매서드를 만들어줍니다.

net.gondr.controller/UserController.java

```
@RequestMapping(value="data", method=RequestMethod.GET)
public @ResponseBody UserVO getUserData() {
    UserVO temp = new UserVO();
    temp.setUserid("tempId");
    temp.setPassword("12345");
    temp.setUsername("임시 아이디");

    return temp;
}
```

ResponseBody 애노테이션은 해당 매서드가 view 로 가지 않고 바로 데이터를 반환한다는 것을 말합니다.

이제 푸른 기대를 안고 해당 /user/data 에 접속해봅시다.

아마도 이전에 만들어둔 전역 익셉션 처리기로 들어가면서 페이지가 없다는 에러가 날 것입니다. 즉 예외상황이 발생했다는 것입니다. 해당 객체를 응답객체로 만들어서 보내줄 라이브러리가 필요한데 그게 없는 것입니다.

객체를 JSON 로 변환해주는 라이브러는 많지만 가장 많이 사용되는 것은 Jackson 과 Gson 입니다. pom.xml 에 다음과 같이 의존성을 추가해줍니다.

<pom.xml>

```
<!-- json전송을 위한 jackson 라이브러리 -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.7</version>
</dependency>
```

다시한번 서버를 키고 접속하면 json 형태로 전환된 데이터를 볼 수 있습니다.



과제. 여태까지 배운 것을 활용하여 /user/data/gondr 이라고 접속하면 gondr 의 정보를 /user/data/admin 이라 하면 admin 을 만약 해당 유저가 없다면 공백을 반환하는 컨트롤러를 만들어보세요. (정답은 다음페이지에 있습니다. 가급적 보지 말고 해결하세요)

## 정답코드

먼저 userMapper 부터 설정하세요. 다음과 같이 select 를 하나 추가합니다.

src/main/resource/mappers/userMapper.xml

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

<mapper namespace="net.gondr.mappers.UserMapper">
  <insert id="insertUser">
    INSERT INTO spring_users (userid, username, password) VALUES (
    #{userid}, #{username}, #{password})
  </insert>

  <select id="selectUser" resultType="net.gondr.domain.UserVO">
    SELECT * FROM spring_users WHERE userid = #{userid}
  </select>
</mapper>
```

그리고 DAO 에서 이 매퍼를 사용하도록 다음과 같이 인터페이스를 수정합니다.

net.gondr.dao/UserDAO.java

```
package net.gondr.dao;

import net.gondr.domain.UserVO;

public interface UserDAO {

    public void insertUser(UserVO user);

    public UserVO selectUser(String userid);
}
```

그리고 추가한 매서드를 구현해야하니 UserDAOImpl.java 에 다음과 같이 추가합니다.

net.gondr.dao/UserDAOImpl.java

```
package net.gondr.dao;
```

```

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import net.gondr.domain.UserVO;

@Repository
public class UserDaoImpl implements UserDao {
    @Autowired
    private SqlSession session;

    private static final String namespace = "net.gondr.mappers.UserMapper";

    @Override
    public void insertUser(UserVO user) {
        session.insert(namespace + ".insertUser", user);
    }

    @Override
    public UserVO selectUser(String userid) {
        return session.selectOne(namespace + ".selectUser", userid);
    }
}

```

그리고 UserHandler 를 다음과 같이 작성합니다.

net.gondr.controller/UserHandler.java

```

@Autowired
private UserDao dao;

@RequestMapping(value="data/{userid}", method=RequestMethod.GET)
public @ResponseBody UserVO getUserData(@PathVariable String userid) {
    UserVO user = dao.selectUser(userid);
    return user;
}

```

이제 접속해서 해당 url 로 요청하면 정상적으로 자료가 나오는 것을 볼 수 있습니다.

스프링의 개념을 이해하고 있다면 손쉽게 여기까지 작성이 가능합니다.

## 6. 뷰 템플릿 엔진 타임리프 사용법.

스프링에서는 뷰 템플릿 엔진으로 타임리프(Thymeleaf)라는 뷰 엔진을 사용합니다. 타임리프는 뷰계층에서 로직(반복, 출력 등)을 좀 더 편하게 작성할 수 있게 해주며, 마스터 템플릿과의 병합등을 편리하게 해줍니다. 사실 이런 설명을 보는 것보다 직접 설치하고 보는게 더 빠릅니다. 스프링에 설치를 위해 다음과 같이 pom.xml 에 작성하여 줍니다.

```
<!-- 타임리프 뷰템플릿 사용을 위한 의존성 -->
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf</artifactId>
    <version>3.0.11.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf-spring5</artifactId>
    <version>3.0.11.RELEASE</version>
</dependency>
```

스프링을 3 버전이나 4 버전을 사용시에는 thymeleaf-spring3 또는 4 를 설치해야 합니다.

이제 이를 servlet-context.xml 에 빈으로 다음과 같이 추가합니다.

```
<!-- 타임리프 관련 템플릿 빈들 -->
    <beans:bean id="templateResolver"
class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver">
    <beans:property name="prefix"
value="/WEB-INF/template/"></beans:property>
    <beans:property name="suffix" value=".html"></beans:property>
    <beans:property name="characterEncoding"
value="UTF-8"></beans:property>
    <beans:property name="templateMode" value="HTML"></beans:property>
    <beans:property name="cacheable" value="true"></beans:property>
    </beans:bean>

    <beans:bean id="templateEngine"
class="org.thymeleaf.spring5.SpringTemplateEngine">
    <beans:property name="templateResolver"
ref="templateResolver"></beans:property>
    <beans:property name="enableSpringELCompiler"
value="true"></beans:property>
    </beans:bean>

    <beans:bean id="viewResolver"
class="org.thymeleaf.spring5.view.ThymeleafViewResolver">
```

```

        <beans:property name="templateEngine"
ref="templateEngine"></beans:property>
        <beans:property name="characterEncoding"
value="UTF-8"></beans:property>
        <beans:property name="order" value="1"></beans:property>
    </beans:bean>

```

물론 당연히 이전에 작성했던 jsp 관련 View Resolver 는 주석처리해주어야 합니다.

```

<!-- Controller들이 보내는 반환 String 으로 만들어질 View Resolver 셋팅 -->
<!--
<beans:bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/view/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>
-->

```

이제 우리의 메시지를 표현해줄 타임리프 템플릿을 만들어봅시다. index.jsp 대신에 index.html 을 WEB-INF/template 폴더에 만들어 줍니다.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>메인페이지</title>
</head>
<body>
    <h1>서버 메시지 Hello world</h1>
    <p th:text="${msg}"></p>
</body>
</html>

```

타임리프는 th 로 시작하는 고유의 명령을 가지고 있으며 이를 사용해서 뷰를 작성하게 됩니다.

데이터 출력하는 th:text, th:html 외에도 반복문을 작성하는 th:each, 폼작성을 할 수 있는 th:action, th:field 등 다양한 태그들을 지원합니다.

물론 본인이 JSTL 이 좀 더 편하다면 그걸 사용해도 아무 문제 없습니다. JSTL 을 사용할 사람들은 위쪽의 설정대로 그대로 쓰고 뷰파일로 jsp 를 사용하면 됩니다.

사실 타임리프 자체가 가지는 장점이 커서 이 강의에서 타임리프를 사용한다기보단 현재 스프링 진영에서 뷰 템플릿엔진으로 밀고 있기에 다루보는 것입니다. (전 Jade 가 더 편합니다)

타임리프에 대한 좀 더 자세한 사용법은 2부에서 스프링을 사용해서 웹사이트를 개발해보면서  
사용하겠습니다.

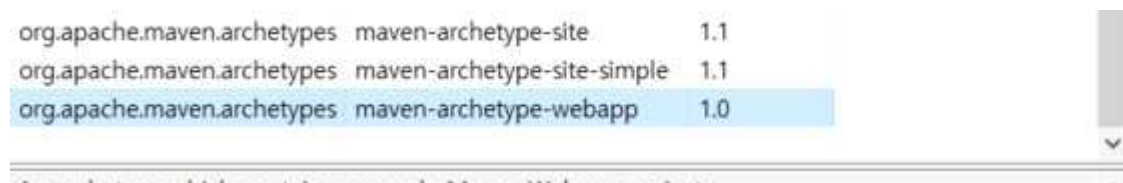
## **PART 2. 스프링 5 를 이용해 웹사이트 제작하기**

## 1. 프로젝트 준비

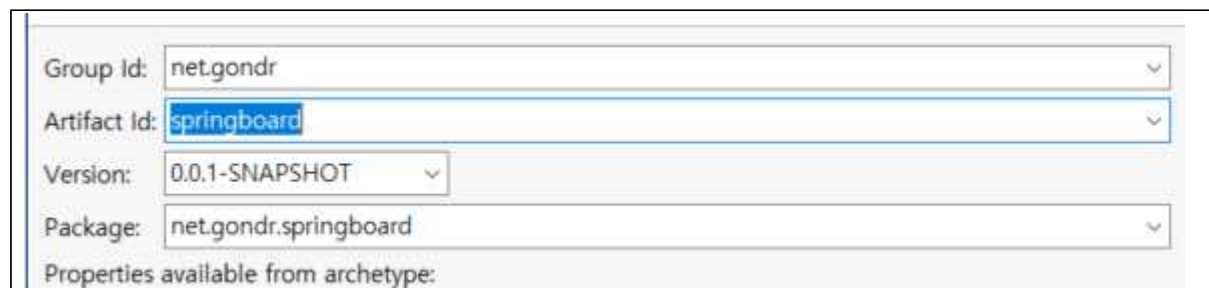
이번에는 이전과 다르게 이클립스에서 메이븐 프로젝트를 직접 만드는 방법으로 작업을 시작하겠습니다.

File->new -> maven project 를 선택합니다.

첫번째는 그냥 next를 누르고 두번째 선택상자에서는 기본으로 되어있는 maven-archetype-webapp 를 선택하고 next를 누릅니다.



다음 창에서는 아래와 같이 정보를 입력하고 Next를 눌러줍니다. (자신의 패키지 명으로 변경해도 좋습니다만 진행되면서 코드를 복사해서 사용할 것이라면 이대로 가는 것이 정신건강에 좋을 것이라 생각합니다.)



완료 버튼을 누르면 메이븐으로부터 필요한 자료들이 다운로드되면서 프로젝트가 구성될 것입니다.

에러가 있는건 무시하고 pom.xml 부터 작성하도록 합시다. 다소 길지만 다음과 같이 pom.xml을 작성합니다.

<pom.xml>

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>net.gondr</groupId>
  <artifactId>springboard</artifactId>
  <packaging>war</packaging>
```

```

<version>1.0.0</version>
<name>springboard</name>

<properties>
<org.springframework-version>5.1.2.RELEASE</org.springframework-version>
  <java-version>1.8</java-version>
</properties>

<dependencies>
  <!-- 서블릿 규약 -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
  </dependency>

  <!-- jsp 2.3 -->
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.2-b02</version>
    <scope>provided</scope>
  </dependency>

  <!-- springframework -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
  </dependency>

  <!-- 스프링 jdbc 관련 -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
  </dependency>

  <!-- Spring에서 Junit을 테스트 하기 위한 라이브러리 -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${org.springframework-version}</version>
    <scope>test</scope>
  </dependency>

  <!-- 벨리데이션 관련 api -->
  <dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>1.1.0.Final</version>
  </dependency>

```

```

<!-- mysql 연결관련 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>6.0.6</version>
</dependency>

<!-- mybatis 사용 관련 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.6</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.2</version>
</dependency>

<!-- JUnit 테스트를 위한 라이브러리 -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>

<!-- json전송을 위한 jackson 라이브러리 -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.7</version>
</dependency>

<!-- 타임리프 뷰템플릿 사용을 위한 의존성 -->
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf</artifactId>
    <version>3.0.11.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf-spring5</artifactId>
    <version>3.0.11.RELEASE</version>
</dependency>

</dependencies>

<!-- 빌드시 필요한 메이븐 플러그인-->
<build>

```



```

        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>${java-version}</source>
                    <target>1.8</target>
                    <encoding>utf-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>

```

저장하는 순간부터 메이븐인 프로젝트를 빌드하기 시작합니다. 하지만 안전을 위해서 프로젝트에 우클릭후 maven -> update project를 클릭해줍니다.

업데이트가 되었다면 스프링에 관련된 폴더를 추가해줍니다.

WEB-INF에 spring폴더를 만들고 해당 폴더에 root-context.xml 을 다음과 같이 작성합니다.

<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:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="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">

    <!-- Root Context: 모든 웹컴포넌트에 공유될 빈객체들을 생성하는 곳 -->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"
value="com.mysql.cj.jdbc.Driver"></property>
        <property name="url"
value="jdbc:mysql://gondr.asuscomm.com/devdb?useSSL=false&serverTimezone=Asia/Seoul"></property>
        <property name="username" value="devdb"></property>
        <property name="password" value="1234"></property>
    </bean>

    <!-- sqlSessionSessionFactory -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation"
value="classpath:/mybatis-config.xml"></property>
        <property name="mapperLocations"
value="classpath:/mappers/**/*.xml"></property>

```

```

        </bean>

        <!-- sqlSession : 팩토리로부터 연결을 가져와서 사용후 자동으로 반환까지 해줌.
-->
        <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"
destroy-method="clearCache">
            <!-- 빈생성시 파라미터로 sqlSessionFactroy빈을 넘겨줌. -->
            <constructor-arg name="sqlSessionFactory"
ref="sqlSessionFactory"></constructor-arg>
        </bean>

</beans>

```

데이터베이스에 관련된 빈들은 값을 각자에 맞게 변경해야 한다는 것을 잊지 마세요.

spring 폴더에 appServlet폴더를 만들고 그안에 servlet-context.xml 을 다음과 같이 만들어주세요

<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"
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">

    <!-- DispatcherServlet Context: 서블릿 디스패처가 요청을 처리하는 구조속에서
필요한 빈을 선언하는 곳 -->

    <!-- 스프링의 @Controller 를 비롯한 다양한 애노테이션이 동작할 수 있도록
Enable 해줌-->
    <annotation-driven />

    <!-- 전역 익셉션 처리기 -->
    <beans:bean class="net.gondr.exception.CommonExceptionHandler">
    </beans:bean>

    <!-- 타임리프 관련 템플릿 빈들 -->
    <beans:bean id="templateResolver"
class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver">
        <beans:property name="prefix"
value="/WEB-INF/template/"></beans:property>

```

```

        <beans:property name="suffix" value=".html"></beans:property>
        <beans:property name="characterEncoding"
value="UTF-8"></beans:property>
        <beans:property name="templateMode" value="HTML"></beans:property>
        <beans:property name="cacheable" value="true"></beans:property>
    </beans:bean>

    <beans:bean id="templateEngine"
class="org.thymeleaf.spring5.SpringTemplateEngine">
        <beans:property name="templateResolver"
ref="templateResolver"></beans:property>
        <beans:property name="enableSpringELCompiler"
value="true"></beans:property>
    </beans:bean>

    <beans:bean id="viewResolver"
class="org.thymeleaf.spring5.view.ThymeleafViewResolver">
        <beans:property name="templateEngine"
ref="templateEngine"></beans:property>
        <beans:property name="characterEncoding"
value="UTF-8"></beans:property>
        <beans:property name="order" value="1"></beans:property>

    </beans:bean>

    <!-- Controller, Repository, Service 등을 검색할 패키지 -->
    <context:component-scan base-package="net.gondr.controller" />

</beans:beans>

```

web.xml 에서 기본 컨텍스트 값으로 root-context를 불러오고 servlet-context를 불러서 dispatcher를 만들어주기 위해 다음과 같이 작성합니다.

/WEB-INF/web.xml

```

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

    <!-- 전체 환경에 대한 설정파일 설정 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>

    <!-- 전체 환경에서 로드할 리스너 설정 -->
    <listener>

```

```

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>

<!-- URI에 따라 라우팅을 담당할 디스패처를 서블릿으로 등록함. -->
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>

  <!-- 사용자가 지정한 환경설정 클래스들의 위치를 지정해줌. -->
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/spring/appServlet/servlet-context.xml
    </param-value>
  </init-param>

  <!-- 가장 첫번째로 로드되도록 만들어줌. -->
  <load-on-startup>1</load-on-startup>
</servlet>

<!-- 서블릿을 /주소에 매핑하여 모든 요청을 해당 서블릿이 받을 수 있도록 함. -->
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<filter>
  <filter-name>encodingFilter</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>
</filter>

<!-- 모든 url에 해당 인코딩 필터가 동작하도록 매핑 -->
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>

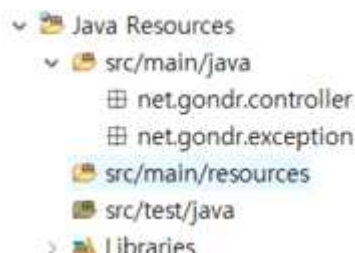
```

src/main/java 폴더와 /src/test/java 폴더를 만들어줍니다.



만들어지는 순간 자동으로 클래스 패스 상으로 추가될 것입니다. (만약 안된다면 update project를 수행하면됩니다.)

그리고 2개의 패키지를 만들어줍니다. net.gondr.controller, net.gondr.exception 완성된 모습은 다음과 같습니다.



net.gondr.exception 패키지에 CommonExceptionHandler 를 다음과 같이 작성합니다.

<net.gondr.exception/CommonExceptionHandler.java>

```
package net.gondr.exception;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice("net.gondr.controller")
public class CommonExceptionHandler {
    @ExceptionHandler(Exception.class)
    public String handleException(Exception e) {
        e.printStackTrace();
        return "errorpage";
    }
}
```

메인페이지를 만들지 않았으므로 아직 익셉션 페이지는 만들지 않습니다.

net.gondr.controller 패키지에 MainController를 만들고 다음과 같이 작업합니다.

<net.gondr.controller/MainController.java>

```
package net.gondr.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
```

```
@Controller
public class MainController {
    @RequestMapping(value="/", method=RequestMethod.GET)
    public String indexPage() {
        return "index";
    }
}
```

mybatis-config도 src/main/resources에 다음과 같이 작성합니다.

<src/main/resources/mybatis-config.xml>

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 아직 환경설정은 작성할 것이 없으니 비워둡니다. -->
</configuration>
```

그리고 resource폴더에 mappers 패키지도 함께 작성해줍니다.

이제 템플릿 폴더에 index.html을 다음과 같이 작성해보도록 하겠습니다.

먼저 공통적으로 사용될 header부분을 template폴더에 fragments 폴더를 만들고 그안에 layout.html 로 다음과 같이 코딩합니다.

<WEB-INF/template/fragments/layout.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org" >
<head th:fragment="head">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js">
</script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>

<footer th:fragment="footer">
    Copyright &copy; Gondr.net
```

```
</footer>
</html>
```

index.html을 template폴더에 다음과 같이 만들어줍니다.

<WEB-INF/template/index.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head th:replace="fragments/layout :: head"></head>

<body>
    <div class="container">
        <div class="jumbotron">
            <h1 class="display-4">양영디지털고 CodeBook!</h1>
            <p class="lead">코드북은 우리가 일상생활에서 코드를 작성하면서
부딪혔던 문제들과 해결책을 공유하기 위해 올리는 공간입니다.</p>
            <hr class="my-4">
            <p>내가 고생하며 해결했던 일은 언젠가 다른 프로그래머에게도
고통으로 다가옵니다. 이를 도와줄 수 있는 사이트! Code Book입니다.</p>
            <a class="btn btn-primary btn-lg" href="#"
role="button">보러가기</a>
        </div>
    </div>
    <footer th:replace="fragments/layout :: footer"></footer>
</body>
</html>
```

출력되는 모습은 다음과 같습니다.

## 양영디지털고 CodeBook!

코드북은 우리가 일상생활에서 코드를 작성하면서 부딪혔던 문제들과 해결책을 공유하기 위해 올리는 공간입니다.

내가 고생하며 해결했던 일은 언젠가 다른 프로그래머에게도 고통으로 다가옵니다. 이를 도와줄 수 있는 사이트! Code Book입니다.

보러가기

copyright © Gondr.net

여기까지 작업했다면 프로젝트를 시작할 준비가 끝났습니다. 이 강의를 통해서 여러분은 코드를 공유하고 서로 댓글을 보면서 함께 문제점을 해결할 수 있는 회원제 웹사이트를 개발하게 될 것입니다.

## 2. 메인페이지 레이아웃 만들기

지금은 단순한 푸터와 헤드밖에 없지만 header부분인 nav바와 좀더 그럴듯한 푸터를 만들어서 전체적인 웹사이트의 구조를 완성해보겠습니다. (물론 Bootstrap을 쓸 것입니다.)

먼저 정적인 리소스들을 더 이상 웹페이지로부터 불러오는 cdn이 아닌 자체 리소스로 넣고 불러오도록 하겠습니다.

기존에 했던 방식대로라면 다운로드해서 압축을 풀고 정적 디렉토리에 넣어주고 이를 링크를 걸어주는 방식으로 할 것입니다. 하지만 이는 매우 귀찮습니다.

프론트 엔드 관리도구인 npm + webpack을 쓰면 이 모든 것이 깔끔해집니다만, 이걸 여기서 다루기에는 너무 분량이 많습니다. (만약 이 방법을 이미 알고 있다면 이 방법을 쓰세요!!)

여기서는 jsp만의 독특한 프론트 라이브러리 해결방법은 webjar를 사용해보도록 하겠습니다. 서블릿 3버전부터는 WEB-INF/lib에 있는 파일을 리소스 주소를 통해 접근할 수 있게 되었습니다. webjars는 이 점을 이용하여 프론트 라이브러리들을 jar로 패키징하여 pom.xml을 통해 관리할 수 있게 해주는 툴입니다.

먼저 우리가 원하는 라이브러리를 웹자에서 찾아봅시다. 주소는 다음과 같습니다.

<https://www.webjars.org/>

검색을 해서 찾아도 되지만 우리가 원하는 webjar 2개가 바로 상단에 있습니다. 바로 제이쿼리와 부트스트랩입니다. 간단하게 추가해 봅시다. 우리는 메이븐 빌드툴을 사용중이니 상단의 메이븐 탭을 선택합니다.

Name	Versions	Build Tool: SBT / Play 2 / Maven / Ivy / Grape / Gradle / Buildr / Leiningen	Files
npm	5.0.0-2	<pre>&lt;dependency&gt;   &lt;groupId&gt;org.webjars&lt;/groupId&gt;   &lt;artifactId&gt;npm&lt;/artifactId&gt;   &lt;version&gt;5.0.0-2&lt;/version&gt; &lt;/dependency&gt;</pre>	3071 Fi
jquery	3.3.1-1	<pre>&lt;dependency&gt;   &lt;groupId&gt;org.webjars&lt;/groupId&gt;   &lt;artifactId&gt;jquery&lt;/artifactId&gt;   &lt;version&gt;3.3.1-1&lt;/version&gt; &lt;/dependency&gt;</pre>	4 Files
Bootstrap	4.1.3	<pre>&lt;dependency&gt;   &lt;groupId&gt;org.webjars&lt;/groupId&gt;   &lt;artifactId&gt;bootstrap&lt;/artifactId&gt;   &lt;version&gt;4.1.3&lt;/version&gt; &lt;/dependency&gt;</pre>	210 File

이제 2개를 pom.xml에 dependency로 추가합니다.

<pom.xml>

```
<!-- webjars 관련 의존성들 -->
<dependency>
  <groupId>org.webjars</groupId>
```



```

    <artifactId>jquery</artifactId>
    <version>3.3.1-1</version>
  </dependency>
  <dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>4.1.3</version>
  </dependency>
  <dependency>
    <groupId>org.webjars</groupId>
    <artifactId>popper.js</artifactId>
    <version>1.14.4</version>
  </dependency>

```

이제 layout.html의 링크를 다음과 같이 고쳐주자.

<WEB-INF/template/layout.html>

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
  xmlns:th="http://www.thymeleaf.org" >
<head th:fragment="head">
  <link rel="stylesheet"
href="/webjars/bootstrap/4.1.3/css/bootstrap.min.css">
  <script src="/webjars/jquery/3.3.1-1/jquery.min.js"></script>
  <script src="/webjars/popper.js/1.14.4/umd/popper.min.js"></script>
  <script src="/webjars/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>

<footer th:fragment="footer">
  Copyright &copy; Gondr.net
</footer>
</html>

```

물론 접근시 정상적으로 되지 않습니다. 이는 기본 라우트가 없기 때문에 스프링 dispatcher가 모든 요청을 처리함으로 인해 접근이 안되는 것입니다. 다음과 같이 servlet-context.xml에 한줄을 추가하여 기본 라우트 핸들러를 활성화시켜줍시다

<WEB-INF/spring/appServlet/servlet-contxt.xml>

```

<!-- 기본 라우터를 활성화 시켜준다. -->
<default-servlet-handler/>

```

이제 정상적으로 로드 될 것입니다. 위에서 webjars의 경로는 maven dependency를 열어보면 나오는 경로에서 찾아 볼 수 있습니다.

프론트엔드 빌드 도구를 사용하지 않고도 자바의

강점을 이용해 만들어져 상당히 편리합니다. 또한 webjar의 모든 패키지는 npm에 올라와있는 것으로 만들어지기 때문에 npm에 있는 패키지는 webjars에도 있다고 보면 됩니다.

이제 준비가 끝났으니 헤더를 만들어봅시다. layout.html에 다음과 같이 작성합니다.

<WEB-INF/template/layout.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head th:fragment="head">
<link rel="stylesheet"
      href="/webjars/bootstrap/4.1.3/css/bootstrap.min.css">
<script src="/webjars/jquery/3.3.1-1/jquery.min.js"></script>
<script src="/webjars/popper.js/1.14.4/umd/popper.min.js"></script>
<script src="/webjars/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>

<header th:fragment="header" class="mb-4">
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="/">
      
      CodeBook
    </a>
    <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarContent">
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarContent">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
          <a class="nav-link" href="/">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link"
href="/code">CodeBook</a>
        </li>
        <li class="nav-item">
          <a class="nav-link"
href="/board">FreeBoard</a>
        </li>
      </ul>

      <form class="form-inline my-2 my-lg-0">
        <input class="form-control mr-sm-2" type="text"
placeholder="회원아이디" name="userid">
        <input class="form-control mr-sm-2" type="password"
placeholder="비밀번호" name="password">
        <button class="btn btn-outline-success my-2 mr-2
my-sm-0" type="submit">로그인</button>
      </form>
    </div>
  </nav>
</header>
```

```

                                <a class="btn btn-outline-info my-2
my-sm-0">회원가입</a>
                                </form>
                                </div>
                                </nav>
</header>

<footer class="bg-primary text-white fixed-bottom" th:fragment="footer">
    <div class="text-center py-3">© 2018 Copyright: www.gondr.net</div>
</footer>
</html>

```

그리고 index.html에 헤더를 다음과 같이 추가하여 만들어줍니다.

<WEB-INF/template/index.html>

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head th:replace="fragments/layout :: head"></head>

<body>
    <header th:replace="fragments/layout :: header"></header>

    <div class="container">
        <div class="jumbotron">
            <h1 class="display-4">양영디지털고 CodeBook!</h1>
            <p class="lead">코드북은 우리가 일상생활에서 코드를 작성하면서
부딪혔던 문제들과 해결책을 공유하기 위해 올리는 공간입니다.</p>
            <hr class="my-4">
            <p>내가 고생하며 해결했던 일은 언젠가 다른 프로그래머에게도
고통으로 다가옵니다. 이를 도와줄 수 있는 사이트! Code Book입니다.</p>
            <a class="btn btn-primary btn-lg" href="#"
role="button">보러가기</a>
        </div>
    </div>
    <footer th:replace="fragments/layout :: footer"></footer>
</body>
</html>

```

이제 개략적으로 웹사이트 구축과 타임리프에 대해 감이 올 것입니다. 내친김에 에러페이지도 만들어봅시다.

CommonExceptionHandler.java를 다음과 같이 고쳐봅시다.

<net.gondr.exception/CommonExceptionHandler.java>

```

package net.gondr.exception;

```

```

import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice("net.gondr.controller")
public class CommonExceptionHandler {
    @ExceptionHandler(Exception.class)
    public String handleException(Exception e, Model model) {
        model.addAttribute("title", "에러 클래스 : " +
e.getClass().getName());
        model.addAttribute("errorMsg", e.getMessage());
        e.printStackTrace();
        return "errorpage";
    }
}

```

template 폴더에 errorpage.html 을 다음과 같이 작성합니다.

/WEB-INF/template/errorpage.html

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">

<head th:replace="fragments/layout :: head"></head>

<body>
    <header th:replace="fragments/layout :: header"></header>

    <div class="container">
        <div class="row">
            <div class="col-12">
                <div class="card">
                    <div class="card-header">오류 메시지</div>
                    <div class="card-body">
                        <h5 class="card-title">
th:text="${title}">오류의 클래스</h5>
                        <p class="card-text">
th:text="${errorMsg}">오류 메시지</p>
                        <a href="javascript:history.back()"
class="btn btn-primary">이전</a>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <footer th:replace="fragments/layout :: footer"></footer>
</body>

```

```
</html>
```

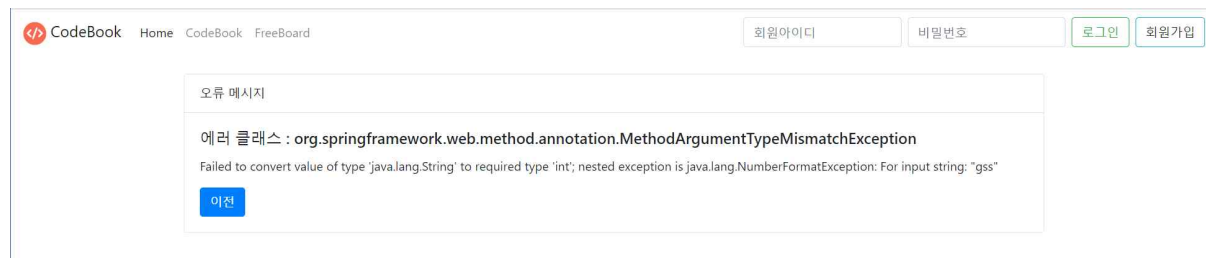
이제 고의적으로 익셉션을 발생시키는 상황을 만들기 위해 MainController.java를 다음과 같이 변경해봅시다.

```
package net.gondr.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class MainController {
    @RequestMapping(value="/", method=RequestMethod.GET)
    public String indexPage(@RequestParam(value="name", required=false,
    defaultValue="10") int name) {
        return "index";
    }
}
```

이제 접속해서 고의적으로 name 속성에 string을 넣으면 다음과 같은 익셉션 페이지를 보게 될 것입니다.



이제 메인페이지의 레이아웃 구성은 끝났습니다. 회원가입과 로그인을 만들어봅시다.

### 3. 회원가입과 로그인 구현하기.

회원가입을 위해서 먼저 백엔드 단부터 차근차근 만들어보겠습니다.

제일 먼저 할 일은 회원 데이터베이스를 만드는 일입니다. 다음과 같이 spring\_users 를 다시 만들어 줍시다.

#	이름	종류	데이터정렬방식	보기	Null	기본값	설명	추가	실행
<input type="checkbox"/>	1	<b>userid</b>	varchar(100)	utf8mb4_general_ci	아니오	없음		변경  삭제  더보기	
<input type="checkbox"/>	2	<b>name</b>	varchar(100)	utf8mb4_general_ci	아니오	없음		변경  삭제  더보기	
<input type="checkbox"/>	3	<b>password</b>	varchar(100)	utf8mb4_general_ci	아니오	없음		변경  삭제  더보기	
<input type="checkbox"/>	4	<b>exp</b>	int(11)		아니오	없음		변경  삭제  더보기	
<input type="checkbox"/>	5	<b>level</b>	int(11)		아니오	없음		변경  삭제  더보기	
<input type="checkbox"/>	6	<b>img</b>	varchar(300)	utf8mb4_general_ci	아니오	없음		변경  삭제  더보기	

이 회원DB와 일치하는 UserVO객체를 만들어줄 차례입니다. net.gondr.domain 패키지를 만들고 해당 패키지 안에 UserVO.java를 다음과 같이 만들어줍니다.

<net.gondr.domain/UserVO.java>

```
package net.gondr.domain;

public class UserVO {
    private String userid;
    private String name;
    private String password;
    private Integer exp;
    private Integer level;
    private String img;

    public String getUserid() {
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
```

```

        this.password = password;
    }
    public Integer getExp() {
        return exp;
    }
    public void setExp(Integer exp) {
        this.exp = exp;
    }
    public Integer getLevel() {
        return level;
    }
    public void setLevel(Integer level) {
        this.level = level;
    }
    public String getImg() {
        return img;
    }
    public void setImg(String img) {
        this.img = img;
    }
    @Override
    public String toString() {
        return "UserVO [userid=" + userid + ", name=" + name + ",
password=" + password + ", exp=" + exp + ", level="
        + level + ", img=" + img + "]\n";
    }
}
}

```

이제 회원가입을 위해 mybatis mapper를 만들어야 합니다. userMapper.xml을 mappers 디렉토리에 다음과 같이 추가합니다.

<src/main/resources/mappers/userMapper.xml>

```

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

<mapper namespace="net.gondr.mappers.UserMapper">
    <insert id="insertUser">
        INSERT INTO spring_users (userid, name, password, exp, level, img)
        VALUES ( #{userid}, #{name}, PASSWORD(#{password}), 0, 1, #{img})
    </insert>

    <select id="selectUser" resultType="UserVO">
        SELECT * FROM spring_users WHERE userid = #{userid}
    </select>

    <select id="loginUser" resultType="UserVO">
        SELECT * FROM spring_users
        WHERE userid = #{userid} AND password = PASSWORD(#{password})
    </select>
</mapper>

```

```
        </select>
    </mapper>
```

net.gondr.domain.UserVO라고 안쓰고도 정상적으로 작동하기 위해서는 별칭을 주어야 합니다. mybatis-config.xml 에 다음과 같이 별칭을 검색할 디렉토리를 입력해주면 domain에 들어가는 모든 패키지는 정상적으로 별칭으로 검색됩니다.

<src/main/resources/mybatis-config.xml>

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 데이터타입 별칭 지정을 위한 Alias -->
    <typeAliases>
        <package name="net.gondr.domain"/>
    </typeAliases>
</configuration>
```

맵퍼는 별도로 설정을 하지 않아도 root-context.xml에 설정해둔 맵퍼 설정으로 인해 자동으로 불러와질 것입니다.

이제 이 맵퍼를 사용하는 dao를 작성해보도록 하겠습니다. net.gondr.dao 패키지를 만들고 해당 패키지 안에 UserDAO 인터페이스를 다음과 같이 만들어줍니다.

<net.gondr.dao/UserDAO.java>

```
package net.gondr.dao;

import net.gondr.domain.UserVO;

public interface UserDAO {
    //유저 정보 불러오는 메서드
    public UserVO getUser(String userid);

    //로그인 정보 불러오는 메서드
    public UserVO loginUser(String userid, String password);

    //회원가입용 메서드
    public void insertUser(UserVO user);
}
```

인터페이스를 구현한 UserDAOImpl 클래스를 다음과 같이 작성합니다.

<net.gondr.dao/UserDAOImpl.java>



```

package net.gondr.dao;

import java.util.HashMap;
import java.util.Map;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import net.gondr.domain.UserVO;

@Repository
public class UserDaoImpl implements UserDao{

    @Autowired
    private SqlSession session;

    private final String namespace = "net.gondr.mappers.UserMapper";

    @Override
    public UserVO getUser(String userid) {
        return session.selectOne(namespace + ".selectUser", userid);
    }

    @Override
    public UserVO loginUser(String userid, String password) {
        Map<String, String> loginMap = new HashMap<String, String>();
        loginMap.put("userid", userid);
        loginMap.put("password", password);

        return session.selectOne(namespace + ".loginUser", loginMap);
    }

    @Override
    public void insertUser(UserVO user) {
        session.insert(namespace + ".insertUser", user);
    }
}

```

여기까지 만들었다면 이 DAO를 자동으로 스프링 빈으로 등록할 수 있도록 root-context.xml 에 다음과 같이 한 줄을 등록합니다.

<WEB-INF/spring/root-contet.xml>

```
<context:component-scan base-package="net.gondr.dao"></context:component-scan>
```

이제 UserDaoImpl은 빈으로 등록되어 있기 때문에 어디서든 Autowired 될 수 있습니다.

Service 계층을 만들기에 앞서서 DAO가 올바른지 테스트를 작성해보도록 하겠습니다. src/test/java 폴더에 net.gondr.dao 패키지를 만들고 다음과 같이 UserDaoTest 를 작성하겠습니다.

<test/net.gondr.dao/UserDAOTest.java>

```

package net.gondr.dao;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import net.gondr.domain.UserVO;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})
public class UserDaoTest {

    @Autowired
    private UserDao dao;

    @Test
    public void testInsertUser() throws Exception {
        UserVO user = new UserVO();
        user.setUserid("gondr");
        user.setPassword("1234");
        user.setImg("");
        user.setName("최선훈");
        dao.insertUser(user);
    }

    @Test
    public void testSelectUser() throws Exception {
        UserVO user = dao.getUser("gondr");
        System.out.println(user);
    }

    @Test
    public void testLogin() throws Exception {
        UserVO user = dao.loginUser("gondr", "1234");
        System.out.println(user);
    }
}

```

주의해야 할 사항은 Test 애노테이션은 순차적인 실행을 보장하지 않는다는 것입니다. 따라서 위와 같은 경우 로그인이 먼저 실행되고 인서트 되고 셀렉트 될 수 도 있습니다. 만약 그렇게 실행된다면 결과는 다음과 같습니다.

```

null
UserVO [userid=gondr, name=최선훈, password=*A4B6157319038724E3560894F7F932C8886EBFCF, exp=0, level=1, img=]

```

따라서 테스트시에는 순차적으로 @Test 를 주석처리해가면서 테스트해야 합니다.

정상적으로 사용자가 들어가고 로그인도 되고 가져오는 것도 된다면 이제 회원가입 서비스를 만들 차례입니다.

net.gondr.service 패키지를 만들고 그안에 UserService 인터페이스를 다음과 같이 만들어줍니다.

```
package net.gondr.service;

import net.gondr.domain.UserVO;

public interface UserService {
    //로그인 서비스
    public UserVO login(String userid, String password);

    //회원가입
    public void register(UserVO user);

    //회원정보 불러오기
    public UserVO getUserInfo(String userid);
}
```

이제 이 서비스를 구현한 UserServiceImpl 클래스를 다음과 같이 만들어줍니다. (지금 현재는 DAO 와 크게 다를 것이 없습니다.)

<net.gondr.service/UserServiceImpl.java>

```
package net.gondr.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import net.gondr.dao.UserDAO;
import net.gondr.domain.UserVO;

@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDAO userDao;

    @Override
    public UserVO getUserInfo(String userid) {
        return userDao.getUser(userid);
    }

    @Override
    public UserVO login(String userid, String password) {
        return userDao.loginUser(userid, password);
    }

    @Override
    public void register(UserVO user) {
        userDao.insertUser(user);
    }
}
```

이 Service도 자동으로 로드될 수 있도록 root-context.xml에 다음과 같이 한 줄을 추가로 작성해 줍니다.

```
<context:component-scan  
base-package="net.gondr.service"></context:component-scan>
```

사실 DAO와 큰 차이점이 없기 때문에 지금은 Test 를 작성하지 않아도 무방하지만 차후 테스트를 지속적으로 수행하기 위해서 Service 테스트도 만들어두겠습니다. 유닛단위로 테스트를 만들어두면 차후 기능추가시 문제점을 발견하는게 무척 쉬워지기 때문에 귀찮더라도 처음 만들때부터 정석대로 만드는 것이 더 좋습니다.

src/test/java에 net.gondr.service 패키지를 만들고 그안에 다음과 같이 UserServiceTest 클래스를 작성합니다.

<test/net.gondr.service/UserServiceTest.java>

```
package net.gondr.service;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.test.context.ContextConfiguration;  
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
  
import net.gondr.domain.UserVO;  
  
@RunWith(SpringJUnit4ClassRunner.class)  
@ContextConfiguration(locations =  
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})  
public class UserServiceTest {  
    @Autowired  
    private UserService service;  
  
    // @Test  
    public void insertUser(){  
        UserVO user = new UserVO();  
        user.setName("최선훈");  
        user.setPassword("1234");  
        user.setUserid("gondr99");  
        user.setImg("");  
  
        service.register(user);  
    }  
}
```

```

@Test
public void loginUser() {
    UserVO user = service.login("gondr99", "1234");
    System.out.println(user);
}

@Test
public void userInfo() {
    UserVO user = service.getUserInfo("gondr99");
    System.out.println(user);
}
}

```

이제 회원가입 메뉴를 만들어봅시다. UserController를 net.gondr.controller 패키지에 다음과 같이 작성합니다.

<net.gondr.controller/UserController.java>

```

package net.gondr.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/user/")
public class UserController {

    @RequestMapping(value="register", method=RequestMethod.GET)
    public String viewRegisterPage() {

        return "user/registerPage";
    }

    @RequestMapping(value="register", method=RequestMethod.POST)
    public String userRegist() {

        return "user/registerPage";
    }
}

```

아직은 post전송시에도 그냥 페이지를 보여줍니다. template 폴더에 user 폴더를 만들고 그 안에 registerPage.html파일을 다음과 같이 만들어 줍니다.

<WEB-INF/template/user/registerPage.html>

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">

```

```

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>

    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">
                <h2>회원 가입</h2>
                <form method="post" enctype="multipart/form-data">
                    <div class="form-group">
                        <label
for="userid">이메일(아이디)</label>
                        <input type="email"
class="form-control" id="userid" placeholder="아이디로 사용할 이메일 주소"
name="userid">
                        <small id="emailHelp"
class="form-text text-muted">사용자 아이디로 쓸 이메일 주소를 입력하세요.</small>
                    </div>

                    <div class="form-group">
                        <label for="username">회원이름</label>
                        <input type="text"
class="form-control" id="username" placeholder="사용자의 이름" name="username">
                    </div>

                    <div class="form-group">
                        <label for="password">비밀번호</label>
                        <input type="password"
class="form-control" id="password" name="password" placeholder="비밀번호를
입력하세요">
                    </div>

                    <div class="form-group">
                        <label for="password">비밀번호
확인</label>
                        <input type="password"
class="form-control" id="passwordConfirm" name="passwordConfirm"
placeholder="비밀번호를 확인 입력하세요">
                    </div>

                    <div class="form-group">
                        <label for="profileImg">프로필 이미지
업로드</label>
                        <input type="file"
class="form-control-file" id="profileImg" name="profileImg">
                    </div>
                </form>
            </div>
        </div>
    </div>

```

```

<button type="submit" class="btn
btn-primary">회원가입</button>
    </form>
</div>
</div>
</div>
<footer th:replace="/fragments/layout :: footer"></footer>
</body>
</html>

```

회원가입페이지로부터 받아들이는 정보를 담을 DTO 객체를 생성하겠습니다. net.gondr.domain 패키지에 다음과 같이 RegisterDTO를 만들어 줍니다.

```

package net.gondr.domain;

import org.springframework.web.multipart.MultipartFile;

public class RegisterDTO {
    private String userid;
    private String username;
    private String password;
    private String passwordConfirm;
    private MultipartFile profileImg;

    @Override
    public String toString() {
        return "RegisterDTO [userid=" + userid + ", password=" + password
+ ", passwordConfirm=" + passwordConfirm
        + ", profileImg=" + profileImg + "];"
    }

    public String getUserid() {
        return userid;
    }

    public void setUserid(String userid) {
        this.userid = userid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {

```

```

        this.password = password;
    }

    public String getPasswordConfirm() {
        return passwordConfirm;
    }

    public void setPasswordConfirm(String passwordConfirm) {
        this.passwordConfirm = passwordConfirm;
    }

    public MultipartFile getProfileImg() {
        return profileImg;
    }

    public void setProfileImg(MultipartFile profileImg) {
        this.profileImg = profileImg;
    }
}

```

회원이입시 넘어오는 멀티파트 파일 데이터를 처리해주기 위해서는 스프링의 멀티파트 환경설정을 빈으로 등록해주어야 하고, 이에 필요한 패키지도 추가로 설치해야 합니다.

pom.xml에 다음과 같이 2개의 패키지를 추가합니다.

<pom.xml>

```

<!-- 파일업로드 관련 의존성 -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.3</version>
</dependency>
<dependency>
    <groupId>org.imgscalr</groupId>
    <artifactId>imgscalr-lib</artifactId>
    <version>4.2</version>
</dependency>

```

첫번째는 파일업로드 관련이고 두번째는 이미지 썸네일 제작을 위한 리사이징 라이브러리 입니다.

이제 servlet-context.xml에 multipart로 넘어오는 데이터를 처리할 수 있도록 multipartResolver를 활성화시켜주어야 합니다. 다음과 같이 빈을 등록합니다.

<WEB-INF/spring/appServlet/servlet-context.xml>

```

<!-- 파일업로드를 위해 멀티파트 데이터 리졸버를 빈으로 등록 -->
<beans:bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

```



```
<beans:property name="maxUploadSize" value="10485760"></beans:property>
</beans:bean>
```

이제 파일 업로드와 회원가입을 위한 기초준비는 끝났습니다.

업로드한 파일이 잘 넘어오는지 확인하기 위해서 다음과 같이 UserController를 수정합니다.

<net.gondr.controller/UserController.java>

```
package net.gondr.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import net.gondr.domain.RegisterDTO;

@Controller
@RequestMapping("/user/")
public class UserController {

    @RequestMapping(value="register", method=RequestMethod.GET)
    public String viewRegisterPage() {

        return "user/registerPage";
    }

    @RequestMapping(value="register", method=RequestMethod.POST)
    public String userRegist(RegisterDTO registerDTO) {

        System.out.println(registerDTO.toString());

        return "user/registerPage";
    }
}
```

회원가입을 시도해보면 다음과 같이 콘솔창에 정보가 나올 것입니다.

```
RegisterDTO [userid=gondr@nate.com, password=asd, passwordConfirm=asd, profileImg=org.springframework.web.multipart.commons.CommonsMultipartFile@cfd700a]
```

간단하게 MultipartFile을 커맨드 객체에 포함하는 것만으로 즉시 파일이 업로드되어 들어옵니다.

이제 이 받은 파일을 저장해야 합니다. 저장에 있어서는 몇가지 문제점이 생깁니다. 파일이름이 중복되는 문제와 저장경로에 대한 문제입니다. 차근 차근 이를 해결해봅시다.

파일업로드는 회원가입 뿐 아니라 게시판에서도 사용할 것이기 때문에 모듈화해서 어디서든 사용할 수 있도록 만들어보겠습니다.

먼저 업로드 된 파일이 이미지 파일인지 검사할 수 있는 MediaUtil을 만들겠습니다. net.gondr.util이라는 패키지를 만들고 그안에 MediaUtil.java를 다음과 같이 만들어줍니다.

<net.gondr.util/MediaUtil.java>

```

package net.gondr.util;

import java.util.HashMap;
import java.util.Map;

import org.springframework.http.MediaType;

public class MediaUtil {
    private static Map<String, MediaType> mediaMap;

    static {
        //정적(클래스) 생성자
        mediaMap = new HashMap<String, MediaType>();
        mediaMap.put("JPG", MediaType.IMAGE_JPEG);
        mediaMap.put("GIF", MediaType.IMAGE_GIF);
        mediaMap.put("PNG", MediaType.IMAGE_PNG);
    }

    public static MediaType getMediaType(String type) {
        return mediaMap.get(type.toUpperCase()); //없다면 null반환
    }
}

```

위의 코드는 mediaMap이라는 해시맵을 만들어서 확장파일별로 해당 미디어의 타입을 알려주는 역할을 합니다.

그리고 같은 패키지에 FileUtil.java도 다음과 같이 작성합니다.

<net.gondr.util/FileUtil.java>

```

package net.gondr.util;

import java.awt.image.BufferedImage;
import java.io.File;
import java.util.UUID;

import javax.imageio.ImageIO;

import org.imgscalr.Scalr;
import org.springframework.util.FileCopyUtils;

public class FileUtil {

    private static String makeThumbnail(String uploadPath, String filename)
    throws Exception{
        //원본파일 읽어오기
        BufferedImage sourceImage = ImageIO.read(new File(uploadPath,
        filename));

        //이미지 비율 유지하고 높이 128에 맞추어 리사이징
        BufferedImage destImage = Scalr.resize(sourceImage,

```

```

    Scalr.Method.AUTOMATIC, Scalr.Mode.FIT_TO_HEIGHT, 128);

    String thumbnailName = uploadPath + File.separator + "s_" +
filename;

    File newFile = new File(thumbnailName);
    String extension = filename.substring(filename.lastIndexOf(".") +
1); //확장자
    ImageIO.write(destImage, extension.toUpperCase(), newFile);
//새로운 파일에 해당 확장자로 다시 써줌.

    //경로구분자를 /로 치환하여 윈도우에서도 잘 동작하도록 함.
    return
thumbnailName.substring(uploadPath.length()).replace(File.separatorChar, '/');
}

    public static String uploadFile(String uploadPath, String originalName,
byte[] fileData) throws Exception{
        UUID uid = UUID.randomUUID(); //파일의 고유 이름을 만들기 위해
UUID를 사용함.

        String saveName = uid.toString() + "_" + originalName;

        File upDir = new File(uploadPath);
        if(!upDir.exists()) {
            upDir.mkdir(); //업로드 경로가 없다면 생성함.
        }
        File target = new File(uploadPath, saveName); //업로드 경로에
랜덤으로 생성한 파일명으로 저장

        FileCopyUtils.copy(fileData, target); //파일카피 유틸리티로 넘어온
데이터를 target으로 복사.

        String extension =
originalName.substring(originalName.lastIndexOf(".") + 1); //확장자 알아내고

        String uploadFilename = null;
        if(MediaUtil.getMediaType(extension) != null) { //만약 이미지
파일이 업로드 된거라면
            uploadFilename = makeThumbnail(uploadPath, saveName);
        }else {
            uploadFilename = (uploadPath + File.separator +
saveName).replace(File.separatorChar, '/');
        }

        return uploadFilename; //저장된 파일명을 보내줌.(이미지면
    }
}

```

다소 복잡하기에 최대한 주석을 달아 설명을 해두었습니다.

실질적인 업로드는 uploadFile 매서드에서 일어나고 makeThumbnail은 64X64 사이즈로 리사이징된 이미지를 반환합니다.

이제 UserController에서 실제 파일이 업로드 되도록 다음과 같이 코드를 만들어줍니다.

```
package net.gondr.controller;

import javax.servlet.ServletContext;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.multipart.MultipartFile;

import net.gondr.domain.RegisterDTO;
import net.gondr.util.FileUtil;

@Controller
@RequestMapping("/user/")
public class UserController {

    @Autowired
    private ServletContext context;

    @RequestMapping(value="register", method=RequestMethod.GET)
    public String viewRegisterPage() {

        return "user/registerPage";
    }

    @RequestMapping(value="register", method=RequestMethod.POST)
    public String userRegist(RegisterDTO registerDTO) throws Exception{

        String uploadPath = context.getRealPath("/WEB-INF/upload");
        MultipartFile file = registerDTO.getProfileImg();
        String upFile = FileUtil.uploadFile(uploadPath,
file.getOriginalFilename(), file.getBytes());

        System.out.println(uploadPath + "에 " + upFile + "로 업로드 됨");
        return "user/registerPage";
    }
}
```

업로드할 Path를 알기 위해 ServletContext를 Autowired 시켰습니다. 그리고 실제 WAS가 구동되고 있는 경로상의 WEB-INF 폴더에 upload 폴더 밑에 파일을 업로드했습니다. 해당 폴더에 가보면 이미지 파일의 경우 썸네일 이미지와 원본이미지가 저장된 것을 볼 수 있습니다. (썸네일은 원

본이름앞에 s\_ 가 붙어있습니다.)

이제 파일 업로드 기능을 구현했으니 회원가입이 되도록 컨트롤러를 만들어봅시다.

<net.gondr.controller/UserController.java>

```
package net.gondr.controller;

import javax.servlet.ServletContext;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.multipart.MultipartFile;

import net.gondr.domain.RegisterDTO;
import net.gondr.domain.UserVO;
import net.gondr.service.UserService;
import net.gondr.util.FileUtil;

@Controller
@RequestMapping("/user/")
public class UserController {

    @Autowired
    private ServletContext context;

    @Autowired
    private UserService uService;

    @RequestMapping(value="register", method=RequestMethod.GET)
    public String viewRegisterPage() {

        return "user/registerPage";
    }

    @RequestMapping(value="register", method=RequestMethod.POST)
    public String userRegist(RegisterDTO registerDTO) throws Exception{

        String uploadPath = context.getRealPath("/WEB-INF/upload");
        MultipartFile file = registerDTO.getProfileImg();
        String upFile = "";
        if(!file.getOriginalFilename().equals("")) {
            upFile = FileUtil.uploadFile(uploadPath,
file.getOriginalFilename(), file.getBytes());
        }

        UserVO user = new UserVO();
        user.setImg(upFile);
        user.setName(registerDTO.getUsername());
        user.setPassword(registerDTO.getPassword());
        user.setUserid(registerDTO.getUserid());
    }
}
```

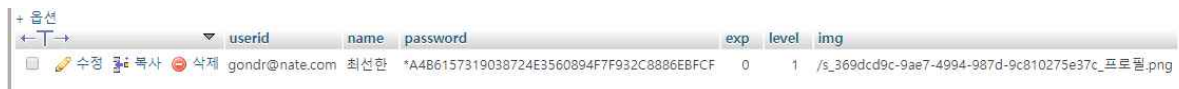
```

        uService.register(user);

        //return "user/registerPage";
        return "redirect:/";
    }
}

```

이제 아무 정보나 입력하고 제대로 회원가입이 되는지 확인해봅시다.



위와 같이 가입되어 있다면 성공입니다.

하지만 이대로 끝내면 안되겠지요? 회원가입시 놓아야할 문장은 넣었는지, 비밀번호와 비밀번호 확인이 일치하는지 등을 검사해야 합니다. 이 부분은 이전에 배웠던 벨리데이터를 활용해보겠습니다. net.gondr.validator 패키지를 만들고 다음과 같이 RegisterValidator를 만들어줍니다.

<net.gondr.validator/RegisterValidator.java>

```

package net.gondr.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
import org.springframework.web.multipart.MultipartFile;

import net.gondr.domain.RegisterDTO;
import net.gondr.util.MediaUtil;

public class RegisterValidator implements Validator{

    @Override
    public boolean supports(Class<?> clazz) {
        return RegisterDTO.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "userid",
"required", "유저 아이디의 값은 필수입니다.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username",
"required", "유저 이름은 필수입니다.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password",
"required", "비밀번호 입력은 필수 입니다.");

        RegisterDTO dto = (RegisterDTO) target;
        if(!dto.getPassword().equals(dto.getPasswordConfirm())) {
            errors.rejectValue("password", "bad", "비밀번호와 비밀번호

```

```

        확인이 일치하지 않습니다");
    }

    MultipartFile img = dto.getProfileImg();
    if(!img.getOriginalFilename().equals("")) {
        String filename = img.getOriginalFilename();
        String extension =
filename.substring(filename.lastIndexOf(".") + 1);
        if(MediaUtil.getMediaType(extension) == null) {
            errors.rejectValue("profileImg", "bad", "프로필
이미지는 이미지 파일만 업로드 가능합니다.");
        }
    }
}
}
}

```

위의 코드는 회원가입하는 정보가 올바른지를 검사하고 올바르지 않은 정보의 경우 errors객체에 넣어줍니다. Controller를 다음과 같이 정정하여서 위의 벨리데이터를 사용합시다.

```

package net.gondr.controller;

import javax.servlet.ServletContext;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.multipart.MultipartFile;

import net.gondr.domain.RegisterDTO;
import net.gondr.domain.UserVO;
import net.gondr.service.UserService;
import net.gondr.util.FileUtil;
import net.gondr.validator.RegisterValidator;

@Controller
@RequestMapping("/user/")
public class UserController {

    @Autowired
    private ServletContext context;

    @Autowired
    private UserService uService;

    @RequestMapping(value="register", method=RequestMethod.GET)
    public String viewRegisterPage(Model model) {
        model.addAttribute("registerDTO", new RegisterDTO()); //빈 객체를
모델에 실어서 보낸다.
    }
}

```

```

        return "user/registerPage";
    }

    @RequestMapping(value="register", method=RequestMethod.POST)
    public String userRegist(RegisterDTO registerDTO, Errors errors) throws
    Exception{

        //올바른 값인지 벨리데이팅
        new RegisterValidator().validate(registerDTO, errors);
        if(errors.hasErrors()) {
            return "user/registerPage";
        }

        String uploadPath = context.getRealPath("/WEB-INF/upload");
        MultipartFile file = registerDTO.getProfileImg();
        String upFile = "";
        if(!file.getOriginalFilename().equals("")) {
            upFile = FileUtil.uploadFile(uploadPath,
file.getOriginalFilename(), file.getBytes());
        }

        UserVO user = new UserVO();
        user.setImg(upFile);
        user.setName(registerDTO.getUsername());
        user.setPassword(registerDTO.getPassword());
        user.setUserid(registerDTO.getUserid());
        userService.register(user);

        return "redirect:/";
    }
}

```

굵은 글씨와 하이라이팅 처리된 부분이 변경된 부분입니다.

먼저 페이지로 이동할 때 비어있는 RegisterDTO객체를 보내서 해당 객체를 벨리데이팅 객체로 쓸 수 있도록 합니다. (이는 차후 나올 폼의 타임리프와 연동이 됩니다.)

이후 벨리데이터를 만들어서 자료를 검증하고 에러가 있을 경우 다시 registerPage로 보냅니다.

registerPage는 다음과 같이 정정하여 보내준 객체와 연동되어진 에러들을 표기할 수 있도록 합니다.

<WEB-INF/template/user/registerPage.html>

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/Layout :: head"></head>

<body>

```



```

<header th:replace="/fragments/layout :: header"></header>

<div class="container">
    <div class="row">
        <div class="col-10 offset-1">
            <h2>회원 가입</h2>
            <form method="post" th:action="@{/user/register}"
enctype="multipart/form-data" th:object="${registerDTO}">
                <div class="form-group">
                    <label
for="userid">이메일(아이디)</label>

                    <input type="email"
class="form-control" th:errorclass="is-invalid" id="userid" placeholder="아이디로
사용할 이메일 주소" th:field="*{userid}">

                    <small id="emailHelp"
class="form-text text-muted">사용자 아이디로 쓸 이메일 주소를 입력하세요.</small>
                    <div th:errors="*{userid}"
class="invalid-feedback">

                        사용자 아이디 오류시 보여줄
메시지

                    </div>
                </div>

                <div class="form-group">
                    <label for="username">회원이름</label>
                    <input type="text"
class="form-control" th:errorclass="is-invalid" id="username"
placeholder="사용자의 이름" th:field="*{username}">
                    <div th:errors="*{username}"
class="invalid-feedback">

                        회원이름 관련 에러시 보여줄
메시지

                    </div>
                </div>

                <div class="form-group">
                    <label for="password">비밀번호</label>
                    <input type="password"
class="form-control" id="password" th:errorclass="is-invalid"
th:field="*{password}" placeholder="비밀번호를 입력하세요">
                    <div th:errors="*{password}"
class="invalid-feedback">

                        비밀번호 관련 에러시 보여줄
메시지

                    </div>
                </div>

                <div class="form-group">

```



에는 `*{}` 를 사용했다는 것을 꼭 기억하세요. 그리고 그 밑에는 `th:errors` 를 통해서 해당 속성에 에러가 났을 경우 표시해줄 `div`를 만들었습니다. 만약 해당 속성에 에러가 존재한다면 `div`안에 있는 "회원이름 관련 에러시 보여줄 메시지" 대신에 에러메시지가 보여지게 됩니다. 물론 평상시에는 이 메시지는 보이지 않습니다. 부트스트랩에서 `invalid-feedback` 클래스는 동위선택자인 `input` 태그에 `is-invalid` 클래스가 있을 경우에만 활성화되고, 그렇지 않으면 `display:none`처리 되기 때문입니다.(부트스트랩 설정상 그렇습니다.) 이 `is-invalid`는 `th:errorclass` 에 의해서 에러가 났을때만 클래스에 추가되기 때문에 에러가 나지 않으면 메시지는 보이지 않게 되는 것입니다.

위의 2가지 코드가 반복되어 폼을 만들고 폼을 밸리데이팅 하게 됩니다. 굉장히 간단하죠? 더불어서 `post`로 전송하더라도 해당 값이 그대로 실려서 다시 표현되기 때문에 잘못된 입력을 했더라도 입력한 값이 사라지지 않습니다.

과제. 밸리데이터를 수정해서 이메일 형식이 아니면 받아들이지 않도록 만들어보세요.(정답은 다음페이지에 있습니다.)

```

package net.gondr.validator;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
import org.springframework.web.multipart.MultipartFile;

import net.gondr.domain.RegisterDTO;
import net.gondr.util.MediaUtil;

public class RegisterValidator implements Validator{

    private static final String emailRegExp =
    "^[_A-Za-z0-9-\\+](\\.[_A-Za-z0-9-])*@[_A-Za-z0-9-](\\.[_A-Za-z0-9-])*([A-Za-z]{2,})$";
    private Pattern pattern;

    public RegisterValidator() {
        pattern = Pattern.compile(emailRegExp);
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return RegisterDTO.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "userid",
"required", "유저 아이디의 값은 필수입니다.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username",
"required", "유저 이름은 필수입니다.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password",
"required", "비밀번호 입력은 필수 입니다.");

        RegisterDTO dto = (RegisterDTO) target;
        if(!dto.getPassword().equals(dto.getPasswordConfirm())) {
            errors.rejectValue("password", "bad", "비밀번호와 비밀번호
확인 이 일치하지 않습니다");
        }

        if(!dto.getUserid().isEmpty()) {
            Matcher matcher = pattern.matcher(dto.getUserid());
            if(!matcher.matches()) {
                errors.rejectValue("userid", "bad", "이메일 형식이
올바르지 않습니다.");
            }
        }
    }
}

```

```

    }
}

    MultipartFile img = dto.getProfileImg();
    if(!img.getOriginalFilename().equals("")) {
        String filename = img.getOriginalFilename();
        String extension =
filename.substring(filename.lastIndexOf(".") + 1);
        if(MediaUtil.getMediaType(extension) == null) {
            errors.rejectValue("profileImg", "bad", "프로필
이미지는 이미지 파일만 업로드 가능합니다.");
        }
    }
}
}
}

```

이제 로그인 기능을 구현해보겠습니다. fragment에 있는 layout.html을 다음과 같이 변경하여 로그인 url을 지정합니다.

WEB-INF/template/fragments/layout.html 중 로그인 form 부분

```

<form class="form-inline my-2 my-lg-0" action="/user/Login" method="post">
    <input class="form-control mr-sm-2" type="text" placeholder="회원아이디/"
name="userid">
    <input class="form-control mr-sm-2" type="password"
placeholder="비밀번호" name="password">
    <button class="btn btn-outline-success my-2 mr-2 my-sm-0"
type="submit">로그인</button>
    <a href="/user/register" class="btn btn-outline-info my-2
my-sm-0">회원가입</a>
</form>

```

이 로그인 요청을 담을 커맨드 객체를 다음과 같이 만들어줍니다.

net.gondr.domain/LoginDTO.java

```

package net.gondr.domain;

public class LoginDTO {
    private String userid;
    private String password;

    public String getUserid() {
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
}

```

```

    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

그리고 UserController에 다음과 같이 매서드를 추가하여 로그인관련한 매서드를 추가합니다.

<net.gondr.controller/UserController.java>

```

@RequestMapping(value="login", method=RequestMethod.POST)
public String userLogin(LoginDTO loginDTO, HttpSession session, Model model) {
    String msg = "로그인 실패, 아이디와 비밀번호를 확인하세요";

    if(loginDTO.getUserid() == null || loginDTO.getPassword() == null) {
        return "user/login";
    }
    UserVO user = uService.login(loginDTO.getUserid(),
loginDTO.getPassword());

    if(user == null) {
        return "user/login";
    }

    session.setAttribute("user", user);
    return "redirect:/"; //메인화면으로 리다이렉트 (차후 이부분은 수정가능)
}

@RequestMapping(value="login", method=RequestMethod.GET)
public String viewLoginPage(Model model) {
    model.addAttribute("loginDTO", new LoginDTO());
    return "user/login";
}

```

그리고 로그인 실패시 보여질 login.html을 다음과 같이 작성합니다.

<WEB-INF/template/user/login.html>

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>

    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">

```

```

<h2>로그인</h2>
<form method="post" th:action="@{/user/Login}"
th:object="${LoginDTO}">
    <div class="form-group">
        <label
for="userid">이메일 (아이디)</label>
        <input type="email"
class="form-control" th:errorclass="is-invalid" id="userid" placeholder="아이디/로
사용할 이메일 주소" th:field="*{userid}">
        <div th:errors="*{userid}"
class="invalid-feedback">
            사용자 아이디 오류시 보여줄
에러메시지
        </div>
    </div>
    <div class="form-group">
        <label for="password">비밀번호</label>
        <input type="password"
class="form-control" id="password" th:errorclass="is-invalid"
th:field="*{password}" placeholder="비밀번호를 입력하세요">
        <div th:errors="*{password}"
class="invalid-feedback">
            비밀번호 관련 에러시 보여줄
메시지
        </div>
    </div>
    <button type="submit" class="btn
btn-primary">로그인</button>
</form>
</div>
</div>
<footer th:replace="/fragments/layout :: footer"></footer>
</body>
</html>

```

이제 로그인이 성공적으로 수행되면 루트 페이지로 오게됩니다. 하지만 로그인이 되었을 때 페이지의 변화가 없기 때문에 로그인이 된 건지 알 수가 없습니다. layout.html에서 header부분을 로그인 변수 여부에 따라 변화하게 만들어봅시다.

layout.html을 다음과 같이 변경합니다.

```
<WEB-INF/template/fragments/layout.html>
```

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head th:fragment="head">
<link rel="stylesheet"
      href="/webjars/bootstrap/4.1.3/css/bootstrap.min.css">
<script src="/webjars/jquery/3.3.1-1/jquery.min.js"></script>
<script src="/webjars/popper.js/1.14.4/umd/popper.min.js"></script>
<script src="/webjars/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>

<header th:fragment="header" class="mb-4">
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="/">
      
      CodeBook
    </a>
    <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarContent">
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarContent">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
          <a class="nav-link" href="/">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link"
href="/code">CodeBook</a>
        </li>
        <li class="nav-item">
          <a class="nav-link"
href="/board">FreeBoard</a>
        </li>
      </ul>

      <form th:if="${session.user == null}" class="form-inline
my-2 my-lg-0" action="/user/login" method="post">
        <input class="form-control mr-sm-2" type="text"
placeholder="회원아이디" name="userid">
        <input class="form-control mr-sm-2" type="password"
placeholder="비밀번호" name="password">
        <button class="btn btn-outline-success my-2 mr-2
my-sm-0" type="submit">로그인</button>
        <a href="/user/register" class="btn
btn-outline-info my-2 my-sm-0">회원가입</a>
      </form>
      <div th:if="${session.user != null}">
        <button class="btn btn-outline-success"
th:text="${session.user.name} + '님'"></button>
        <a href="/user/logout" class="btn

```



```

    btn-outline-warning">로그아웃</a>
    </div>
</div>
</nav>
</header>

<footer class="bg-primary text-white fixed-bottom" th:fragment="footer">
    <div class="text-center py-3">© 2018 Copyright: www.gondr.net</div>
</footer>
</html>

```

다른부분은 변한부분이 없고 로그인창을 띄워주는 부분만 변경되었습니다.

이제 세션값에 따라서 화면이 변할 것입니다. 타임리프 템플릿에서는 th:if 를 통해서 해당 값을 테스트 할 수 있습니다. 또한 JSTL과의 차이점은 JSTL은 세션변수와 request 변수에 대한 접근이 다릅니다. 세션변수는 별도로 앞에 session이라는 말을 써주어야 합니다. 위에서도 session에 있는 user변수를 체크하기 위해서 session.user 로 선언했습니다.

이제 로그아웃 기능도 만들어봅시다.

UserController에 다음과 같이 매서드를 추가합니다.

<net.gondr.controller/UserController.java>

```

@RequestMapping(value="logout", method=RequestMethod.GET)
public String userLogout(HttpSession session) {
    session.removeAttribute("user");
    return "redirect:/";
}

```

## 4. 인터셉터를 활용한 권한관리

이제 회원의 회원가입, 로그인, 로그아웃 기능이 마무리 되었습니다. 이제 파트1에서 다루었던 인터셉터를 만들어서 로그인 한 사용자는 회원가입에 접근할 수 없고, 로그인하지 않은 사용자는 로그아웃에 접근할 수 없도록 만들어보겠습니다.

먼저 net.gondr.interceptor 패키지를 만들고 그 안에 AuthInterceptor를 다음과 같이 만들어줍니다.

<net.gondr.interceptor/AuthInterceptor.java>

```
package net.gondr.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.web.servlet.HandlerInterceptor;

public class AuthInterceptor implements HandlerInterceptor{
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        HttpSession session = request.getSession();
        if(session != null) {
            Object user = session.getAttribute("user");
            if(user != null) {
                return true;
            }

            response.sendRedirect("/user/login");
            return false;
        }
    }
}
```

거꾸로 로그인한 사용자가 접근 못하도록 NonAuthInterceptor 도 다음과 같이 작성합니다.

<net.gondr.interceptor/NonAuthInterceptor.java>

```
package net.gondr.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.web.servlet.HandlerInterceptor;

public class NonAuthInterceptor implements HandlerInterceptor{
    @Override
```

```

    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler)
        throws Exception {

        HttpSession session = request.getSession();

        if(session != null) {
            Object user = session.getAttribute("user");
            if(user != null) {
                response.sendRedirect("/");
                return false;
            }
        }
        return true;
    }
}

```

이제 2개의 인터셉터를 servlet-context.xml에 등록해서 로그인한 사용자와 로그인하지 않은 사용자를 구분하여 처리할 수 있게 되었습니다. 다음과 같이 servlet-context.xml에 인터셉터의 등록과 url 등록을 해주도록 하겠습니다.

<WEB-INF/spring/appServlet/servlet-context.xml>

```

<!-- 인터셉터 등록-->
<beans:bean id="authInterceptor"
class="net.gondr.interceptor.AuthInterceptor"></beans:bean>
<beans:bean id="nonAuthInterceptor"
class="net.gondr.interceptor.NonAuthInterceptor"></beans:bean>

<!-- 인터셉터 맵핑 -->
<interceptors>
    <interceptor>
        <mapping path="/user/register"/>
        <beans:ref bean="nonAuthInterceptor"/>
    </interceptor>

    <interceptor>
        <mapping path="/user/Logout"/>
        <beans:ref bean="authInterceptor"/>
    </interceptor>
</interceptors>

```

아직 인터셉터에서 메시지를 실어서 보내는 것은 하고 있지 않습니다. 이부분은 차후 헤더에 메시지를 구현하고 난뒤에 작성하도록 하겠습니다.

위와 같이 작성하면 로그인 여부에 따라 페이지가 갈리게 될 것입니다.

## 5. 게시판 글 쓰기 폼 구성하기.

게시판 글쓰기를 관리하기 글전송객체를 만들어주겠습니다.

pom.xml에 글작성 폼을 위한 2개의 의존성을 추가하겠습니다.

pom.xml

```
<!-- lucy-xss 크로스 사이트 스크립팅 공격 방지를 위한-->
<dependency>
  <groupId>com.navercorp.lucy</groupId>
  <artifactId>lucy-xss-servlet</artifactId>
  <version>2.0.0</version>
</dependency>
```

```
<dependency>
  <groupId>org.webjars.bower</groupId>
  <artifactId>tinymce</artifactId>
  <version>4.8.5</version>
</dependency>
```

첫번째는 XSS 공격방지를 위해 네이버에서 만든 플러그인을 설치하는 것이고 두번째는 tinymce 라는 textEditor를 설치하는 것입니다.

데이터베이스 저장을 위해 BoardVO를 다음과 같이 작성합니다.

<net.gondr.domain/BoardVO.java>

```
package net.gondr.domain;

import java.sql.Date;

public class BoardVO {
    private Integer id;
    private String title;
    private String content;
    private String writer;
    private Date writeDate;

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
}
```

```

    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public String getWriter() {
        return writer;
    }
    public void setWriter(String writer) {
        this.writer = writer;
    }
    public Date getWriteDate() {
        return writeDate;
    }
    public void setWriteDate(Date writeDate) {
        this.writeDate = writeDate;
    }
}

```

이제 글쓰기 화면을 만들기 위해서 BoardController.java 를 만들어줍시다.

net.gondr.controller/BoardController.java

```

package net.gondr.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import net.gondr.domain.BoardVO;

@Controller
@RequestMapping("/board/")
public class BoardController {

    @RequestMapping(value="write", method=RequestMethod.GET)
    public String viewWritePage(Model model) {
        model.addAttribute("boardVO", new BoardVO());
        return "board/write";
    }
}

```

그리고 template 폴더에 board 폴더를 만들고 그안에 write.html을 다음과 같이 작성합니다.

<WEB-INF/template/board/write.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>
    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">
                <h2>글작성</h2>
                <form method="post" th:action="@{/board/write}"
enctype="multipart/form-data" th:object="${boardVO}">
                    <div class="form-group">
                        <label for="title">글 제목</label>
                        <input type="text"
class="form-control" th:errorclass="is-invalid" id="title" placeholder="글 제목을
입력하세요" th:field="*{title}">
                            <div th:errors="*{title}"
class="invalid-feedback">
                                글 제목에 오류가 있을 때 보여줄
메시지
                            </div>
                        </div>
                    <div class="form-group">
                        <textarea class="form-control "
id="content" th:errorclass="is-invalid" placeholder="글 내용을 입력하세요"
th:field="*{content}"></textarea>
                            <div th:errors="*{content}"
class="invalid-feedback">
                                글 내용 관련 오류가 있을 때
보여줄 메시지
                            </div>
                        </div>
                    <button type="submit" class="btn
btn-primary">글작성</button>
                </form>
            </div>
        </div>
    </div>
    <footer th:replace="/fragments/layout :: footer"></footer>
    <script src="/webjars/tinymce/4.8.5/tinymce.min.js"></script>
    <script src="/resources/js/app.js"></script>
</body>
```

```
</html>
```

마지막에 tinymce 플러그인을 추가하고 app.js를 추가했습니다. app.js에서 tinymce를 불러서 초기화시키기만하면 그럴듯한 에디터가 만들어집니다. 다음과 같이 resource 폴더에 app.js를 만들어 줍니다.

/webapp/resources/js/app.js

```
$(function(){
    tinymce.init({
        selector:"#content"
    });
});
```



이제 tinymce에 이미지 파일 업로드 기능과 코드 조각 더하기 기능을 추가해봅시다. 타이니MCE (이하 타이니)에 메뉴를 추가하려면 다음과 같이 app.js를 수정합니다.

<webapp/resources/js/app.js>

```
$(function(){
    tinymce.init({
        selector:"#content",
        plugins : 'advlist autolink link image lists charmap print
preview codesample emoticons textcolor',
        codesample_languages: [
            {text: 'HTML/XML', value: 'markup'},
            {text: 'JavaScript', value: 'javascript'},
            {text: 'CSS', value: 'css'},
            {text: 'PHP', value: 'php'},
            {text: 'Python', value: 'python'},
            {text: 'Java', value: 'java'},
            {text: 'C', value: 'c'},
```

```

        {text: 'C#', value: 'csharp'},
        {text: 'C++', value: 'cpp'},
    ],
    toolbar: [
        'undo redo | styleselect | bold italic | link image
codesample | alignleft aligncenter alignright | forecolor backcolor emoticons'
    ],
    height:400,
    menubar:false
    });
});

```

타이니는 이런식으로 init에 여러가지 옵션들과 플러그인을 추가하여 메뉴를 구성할 수 있습니다. 좀 더 자세한 설명과 한글화 등을 원한다면 타이니 홈페이지(<https://www.tiny.cloud/>)를 참조하기 바랍니다.

이제 실행하고 board/write로 접속해보면 다음과 같이 코드 샘플과 기타 여러가지 옵션들을 넣을 수 있는 화면이 생성될 것입니다.

## 글작성

글 제목

글 제목을 입력하세요

↶ ↷
Formats
B I
🔗 🖼️ {i}
☰ ☷ ☹️
A A ☺️

코드입니다. 📄

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
  xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

```

POWERED BY TINYMCE

글작성

파일업로드도 가능하게 해봅시다. 이전과는 달리 커맨드 객체로 넘어가지 않고 tinymce에서 업로드시 ajax로 업로드 되도록 할 것입니다.

기존의 이미지 업로드는 별도의 링크를 가져와서 업로드하는 방식이기 때문에 좀 더 간단하게 해당 이미지 업로드를 툴바에서 제거하고 버튼을 클릭하면 바로 이미지를 선택하고 파일을 업로드



하는 방식으로 변경하겠습니다.

먼저 파일의 업로드 요청에 대한 응답객체인 UploadResponse 를 다음과 같이 net.gondr.domain 패키지에 만들어줍니다.

<net.gondr.domain/UploadResponse.java>

```
package net.gondr.domain;

public class UploadResponse {
    private boolean result;
    private String msg;
    private String uploadImage;
    private String thumbImage;

    public boolean isResult() {
        return result;
    }
    public void setResult(boolean result) {
        this.result = result;
    }
    public String getMsg() {
        return msg;
    }
    public void setMsg(String msg) {
        this.msg = msg;
    }
    public String getUploadImage() {
        return uploadImage;
    }
    public void setUploadImage(String uploadImage) {
        this.uploadImage = uploadImage;
    }
    public String getThumbImage() {
        return thumbImage;
    }
    public void setThumbImage(String thumbImage) {
        this.thumbImage = thumbImage;
    }
}
```

이제 이 응답객체를 리턴할 업로드 컨트롤러를 BoardController에 다음과 같이 작성합니다.

<net.gondr.controller/BoardController.java>

```
@RequestMapping(value="upload", method=RequestMethod.POST)
@ResponseBody
public UploadResponse handleImageUpload(@RequestParam("file") MultipartFile file,
    HttpServletResponse res) {
    String uploadPath = context.getRealPath("/resources/images");

    UploadResponse response = new UploadResponse();
    try {
        String originalName = file.getOriginalFilename();
```

```

        String extension =
originalName.substring(originalName.lastIndexOf(".") + 1); //확장자 알아내고
        if(MediaUtil.getMediaType(extension) == null) {
            throw new Exception("올바르지 않은 파일 형식");
        }
        String upFile = FileUtil.uploadFile(uploadPath,
file.getOriginalFilename(), file.getBytes());

        //썸네일 경로 셋팅
        response.setThumbImage("/resources/images" + upFile);
        //실제 파일 경로셋팅
        upFile = "/resources/images/" + upFile.substring(3,
upFile.length());
        response.setUploadImage(upFile);
        response.setMsg("성공적으로 업로드 됨");
        response.setResult(true);
    } catch (Exception e) {
        e.printStackTrace();
        response.setMsg(e.getMessage());
        response.setResult(false);
        res.setStatus(HttpServletResponse.SC_BAD_REQUEST);
    }

    return response;
}

```

해당 매서드는 UploadResponse를 직접 반환하기 때문에 @ResponseBody 애노테이션을 붙여주었으며 이전에 작성한 FileUtil을 적극적으로 활용해서 작성하고 있습니다. 성공했다면 썸네일 이미지와 일반 업로드 이미지의 경로가 담긴 응답객체가 전송될 것입니다. 이제 타이니 에디터에서 아이콘을 누르고 파일을 선택하면 자동으로 해당 파일이 업로드 되고 MCE 에디터로 전송될 수 있도록 코드를 작성해보겠습니다. app.js를 다음과 같이 작성합니다.

<webapp/resources/js/app.js>

```

$(function(){
    tinymce.init({
        selector:"#content",
        plugins : 'advlist autolink link image lists charmap print
preview codesample emoticons textcolor',
        codesample_languages: [
            {text: 'HTML/XML', value: 'markup'},
            {text: 'JavaScript', value: 'javascript'},
            {text: 'CSS', value: 'css'},
            {text: 'PHP', value: 'php'},
            {text: 'Python', value: 'python'},
            {text: 'Java', value: 'java'},
            {text: 'C', value: 'c'},
            {text: 'C#', value: 'csharp'},
            {text: 'C++', value: 'cpp'},
        ],
    },

```

```

        toolbar: [
            'undo redo | styleselect | bold italic | link imageupload
codesample | alignleft aligncenter alignright | forecolor backcolor emoticons'
        ],
        height:400,
        menubar:false,
        setup: function(editor) {
            //파일 업로드를 수행할 input 태그를 만들어둠
            var inp = $('<input id="tinymce-uploader" type="file"
name="pic" accept="image/*" style="display:none">');
            $(editor.getElement()).parent().append(inp);

            // 이미지 업로드 버튼을 툴바에 만들고 해당 아이콘을 이미지
아이콘으로 변경함.
            editor.addButton('imageupload', {
                icon: 'image',
                onclick: function(e) { // 툴바 버튼 클릭시 input 태그가 클릭되도록
설정함.

                    inp.trigger('click');
                }
            });

            // 파일이 선택되면 바로 서버로 업로드 시작
            inp.on("change", function(e){
                uploadFile($(this), editor);
            });

            function uploadFile(inp, editor) {
                var input = inp.get(0);
                var data = new FormData();
                data.append('file', input.files[0]);

                $.ajax({
                    url: '/board/upload',
                    type: 'POST',
                    data: data,
                    enctype: 'multipart/form-data',
                    dataType : 'json',
                    processData: false, // 데이터 처리 꺼주고
(이걸 안하면 서버로 데이터 보낼때 getParameter형식으로 변경해서 보냄
                    contentType: false, // 전송 데이터의 콘텐츠
타입을 꺼서 제이쿼리가 자동으로 판단하도록 함.
                    success: function(data, textStatus, jqXHR)
{
                        editor.insertContent(``);
                    },
                    error: function(jqXHR, textStatus,
errorThrown) {

```

```

        console.log(jqXHR);
        if(jqXHR.responseJSON) {
            data = jqXHR.responseJSON;
            alert('이미지 업로드 중 오류
발생: ' + data.msg);
        }
    }); //AJAX 전송 종료
} //업로드 파일 함수 종료
} //setup 종료
}); //tinymce init종료
});

```

기존에 있던 코드에 파일 전송부분만을 추가했습니다. 자세한 설명은 주석으로 대체하였기에 주석을 보고 이해하기 바랍니다. AJAX에 대한 설명은 하지 않습니다.

editor 객체를 활용해서 타이니에디터 안에 업로드된 이미지의 경로를 넣은 img 태그를 삽입하여 업로드와 동시에 화면에 나타나도록 했습니다. 이제 서버를 키고 파일을 업로드해봅시다.

성공적으로 업로드 되면 그림이 나타날 것입니다. 이제 실제 글을 썼을 때 글이 정상적으로 쓰일 수 있도록 만들어보겠습니다.

회원가입시와 동일하게 Mapper와 DAO부터 차근차근 만들면서 올라오겠습니다. 먼저 데이터베이스에 다음과 같이 spring\_boards 테이블을 만들겠습니다.

#	이름	종류	데이터정렬방식	보기	Null	기본값	설명	추가	실행
1	id	int(11)			아니오	없음		AUTO_INCREMENT	변경 삭제 더보기
2	title	varchar(200)	utf8mb4_general_ci		아니오	없음			변경 삭제 더보기
3	content	text	utf8mb4_general_ci		아니오	없음			변경 삭제 더보기
4	writer	varchar(100)	utf8mb4_general_ci		아니오	없음			변경 삭제 더보기
5	writeDate	datetime			아니오	없음			변경 삭제 더보기

그리고 boardMapper.xml 파일을 다음과 같이 작성하겠습니다.

<src/main/resources/mappers/boardMapper.xml>

```

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

<mapper namespace="net.gondr.mappers.BoardMapper">
    <insert id="write">
        INSERT INTO spring_boards (title, content, writer, writeDate)

```

```
VALUES ( #{title}, #{content}, #{writer}, NOW() )
</insert>
</mapper>
```

간단하게 글쓰기에 관한 매퍼만 만들었습니다. 이제 이 글쓰기를 하는 BoardDao를 다음과 같이 작성합니다.

<net.gondr.dao/BoardDAO.java>

```
package net.gondr.dao;

import net.gondr.domain.BoardVO;

public interface BoardDAO {
    //글을 쓰는 매서드
    public void write(BoardVO data);
}
```

그리고 이 인터페이스에 구현체를 다음과 같이 작성합니다.

<net.gondr.dao/BoardDAOImpl.java>

```
package net.gondr.dao;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import net.gondr.domain.BoardVO;

public class BoardDAOImpl implements BoardDAO{
    @Autowired
    private SqlSession session;

    private final String namespace = "net.gondr.mappers.BoardMapper";

    @Override
    public void write(BoardVO data) {
        session.insert(namespace + ".write", data);
    }
}
```

그럼 정상적으로 글이 쓰여지는지 확인하기 위해서 BoardDAOTest를 작성하겠습니다.

<src/test/java/net.gondr.dao/BoardDAOTest.java>

```
package net.gondr.dao;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
```

```

import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import net.gondr.domain.BoardVO;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})
public class BoardDAOTest {
    @Autowired
    private BoardDAO dao;

    @Test
    public void createBoard() {
        BoardVO board = new BoardVO();
        board.setContent("글내용입니다. 테스트용 내용");
        board.setTitle("글 제목입니다. 테스트용");
        board.setWriter("gondr");

        dao.write(board);
    }
}

```

테스팅을 돌려서 글이 잘 써지는 지 확인하세요.

BoardService도 만들어 봅시다. 다음과 같이 BoardService 인터페이스는 net.gondr.service패키지에 만들어줍니다.

<net.gondr.service/BoardService>

```

package net.gondr.service;

import net.gondr.domain.BoardVO;

public interface BoardService {
    //글쓰기
    public void writeArticle(BoardVO board);
}

```

<net.gondr.service/BoardServiceImpl.java>

```

package net.gondr.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import net.gondr.dao.BoardDAO;
import net.gondr.domain.BoardVO;

```

```

@Service
public class BoardServiceImpl implements BoardService{
    @Autowired
    private BoardDAO dao;

    @Override
    public void writeArticle(BoardVO board) {
        dao.write(board);
    }
}

```

역시 마찬가지로 테스트를 위해 BoardServiceTest.java를 다음과 같이 작성후 Junit 로 테스트를 돌려줍니다.

<src/test/java/net.gondr.service/BoardServiceTest.java>

```

package net.gondr.service;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import net.gondr.domain.BoardVO;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})
public class BoardServiceTest {
    @Autowired
    BoardService service;

    @Test
    public void writeBoardTest() {
        BoardVO board = new BoardVO();
        board.setTitle("이번에도 테스트");
        board.setContent("이번에도 테스트 내용입니다. 테스트 테스트");
        board.setWriter("gondr");

        service.writeArticle(board);
    }
}

```

정상적으로 데이터베이스에 삽입되는지 확인해봅시다.

테스트를 잘 통과했다면 실제 글이 쓰여질 때 board객체를 validating할 validator를 만들어보겠습니다

니다.

net.gondr.validator에 다음과 같이 validator를 만들어줍니다.

<net.gondr.validator/BoardValidator.java>

```
package net.gondr.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import net.gondr.domain.BoardVO;

public class BoardValidator implements Validator{

    @Override
    public boolean supports(Class<?> clazz) {
        return BoardVO.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "title",
"required", "글의 제목은 필수값입니다.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "content",
"required", "글의 내용은 필수값입니다.");
    }
}
```

밸리데이터도 만들었으니 실제 글이 쓰여질 수 있도록 BoardController에 writeProcess 매서드를 만들어보겠습니다.

<net.gondr.controller/BoardController.java>

```
@Controller
@RequestMapping("/board/")
public class BoardController {
    @Autowired
    private ServletContext context;

    @Autowired
    private BoardService boardService;

    //...중간 코드 생략...

    @RequestMapping(value="write", method=RequestMethod.POST)
```



```

    public String writeProcess(BoardVO board, HttpSession session, Errors
errors) {
        //올바른 값인지 벨리데이팅
        new BoardValidator().validate(board, errors);
        if(errors.hasErrors()) {
            return "board/write"; //에러가 존재하면 글쓰기 페이지로 보냄.
        }
        //여기는 인터셉터에 의해서 로그인하지 않은 사용자는 막히게 될 것이기
        때문에 그냥 에러처리 없이 user를 불러써도 된다.
        UserVO user = (UserVO)session.getAttribute("user");
        //로그인한 사용자의 아이디를 글쓴이로 등록하고
        board.setWriter(user.getUserid());

        //실제 DB에 글을 기록함.
        boardService.writeArticle(board);
        return "redirect:/board";
    }
    //.....중간 코드 생략.....
}

```

주석을 제외하면 실제 코드는 얼마 되지 않습니다. 이제 글을 작성하고 실제로 기록되는지 봅시다.

만약 기존에 웹페이지를 개발했다면 지금쯤 드는 생각이 아마도 XSS공격에 대한 대비가 되어 있는가입니다. 다행인건 타이니 에디터가 자동으로 script태그는 이스케이프 시켜서 넣어줍니다. 실험삼아서 스크립트가 들어가는 코드를 작성해서 글을 써보세요.

```

제목 <p>&lt;script>alert('a');&lt;/script>
</p>
<...

```

타이니가 자동으로 스크립트공격이 가능한 태그들만 이스케이프 처리해줍니다. 하지만 웹사이트에서 자바스크립트를 disable시키면 스크립트 기반으로 작동하는 타이니는 이를 처리해주지 못하고 값은 그대로 넘어가게 됩니다.

수정 ✎ 복사 🗑 삭제 5 스크립트 공격 <script>alert('a');</script>

gondr@nate.com 2018-11-26 21:09:14

즉 타이니만으로 보안을 믿는 것은 불가능하다는것이죠. 애초에 보안은 클라이언트가 아닌 서버단에서 관리해야 합니다. 이전에 깔아두었던 lucy-xss 의 필터기능을 사용할 차례입니다.

web.xml에 다음과 같이 servlet-filter를 추가합니다.

<WEB-INF/web.xml>

```
<!-- lucy-xss 필터 적용 반드시 Charencoding 필터 뒤쪽에 선언할 것.-->
<filter>
    <filter-name>lucy</filter-name>
    <filter-class>com.navercorp.lucy.security.xss.servletfilter.XssEscapeServletFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>lucy</filter-name>
    <url-pattern>/board/write</url-pattern>
</filter-mapping>
```

주의사항은 반드시 encodingFilter 뒤쪽에 이를 선언해줘야 정상적으로 작동한다는 것입니다.

그리고 src/main/resources에 다음과 같이 lucy-xss-servlet-filter-rule.xml 파일을 작성합니다.

<src/main/resources/lucy-xss-servlet-filter-rule.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- version 20180725-1 -->
<config xmlns="http://www.nhncorp.com/lucy-xss"
    extends="lucy-xss-default-sax.xml">
    <elementRule>
        <element name="body" disable="true" /> <!-- <BODY
ONLOAD=alert("XSS")>, <BODY BACKGROUND="javascript:alert('XSS')"> -->
        <element name="embed" disable="true" />
        <element name="iframe" disable="true" /> <!-- <IFRAME
SRC="http://hacker-site.com/xss.html"> -->
        <element name="meta" disable="true" />
        <element name="object" disable="true" />
        <element name="script" disable="true" /> <!-- <SCRIPT>
alert("XSS"); </SCRIPT> -->
        <element name="style" disable="true" />
        <element name="link" disable="true" />
        <element name="base" disable="true" />
    </elementRule>
    <attributeRule>
        <attribute name="data" base64Decoding="true">
<notAllowedPattern><![CDATA[(?i:s\\*c\\*r\\*i\\*p\\*t\\*:.)]></notAllowedPattern>
<notAllowedPattern><![CDATA[(?i:d\\*a\\*t\\*a\\*:.)]></notAllowedPattern>
<notAllowedPattern><![CDATA[&[#\\%x]+[\\da-fA-F][\\da-fA-F]+]]></notAllowedPattern>
        </attribute>
        <attribute name="src" base64Decoding="true">
<notAllowedPattern><![CDATA[(?i:s\\*c\\*r\\*i\\*p\\*t\\*:.)]></notAllowedPattern>
<notAllowedPattern><![CDATA[(?i:d\\*a\\*t\\*a\\*:.)]></notAllowedPattern>
<notAllowedPattern><![CDATA[&[#\\%x]+[\\da-fA-F][\\da-fA-F]+]]></notAllowedPattern>
        </attribute>
        <attribute name="style">
<notAllowedPattern><![CDATA[(?i:j\\*a\\*v\\*a\\*s\\*c\\*r\\*i\\*p\\*t\\*:.)]></notAllowedPattern>
<notAllowedPattern><![CDATA[(?i:e\\*x\\*p\\*r\\*e\\*s\\*s\\*i\\*o\\*n)]></notAllowedPattern>
<notAllowedPattern><![CDATA[&[#\\%x]+[\\da-fA-F][\\da-fA-F]+]]></notAllowedPattern>
        </attribute>
        <attribute name="href">
```

```

<notAllowedPattern><![CDATA[(?i:j\\*a\\*v\\*a\\*s\\*c\\*r\\*i\\*p\\*t\\*:.)]]></notAllowedPattern>
<notAllowedPattern><![CDATA[&[#\\%x]+[\\da-fA-F][\\da-fA-F]+]]></notAllowedPattern>
    </attribute>
    <attribute name="formaction" disable="true"/>
</attributeRule>
</config>

```

역시 마찬가지로 같은 폴더에 lucy-xss-sax.xml을 다음과 같이 작성합니다.

<net.gondr.resources/lucy-xss-sax.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="http://www.nhncorp.com/lucy-xss"
extends="lucy-xss-superset-sax.xml">
    <elementRule>
        <element name="applet" disable="true" />
        <element name="base" disable="true" />
        <element name="body" disable="true" />
        <element name="form" disable="true" />
        <element name="html" disable="true" />
        <element name="iframe" disable="true" />
        <element name="meta" disable="true" />
        <element name="script" disable="true" />
        <element name="style" disable="true" />
        <element name="textarea" disable="true" />
        <element name="xml" disable="true" />
        <element name="xmp" disable="true" />
        <element name="object" disable="true" />

        <element name="td" disable="false" />
        <element name="tr" disable="false" />
        <element name="table" disable="false" />
        <element name="img" disable="false" />
        <element name="a" disable="false" />
        <element name="pre" disable="false" />
        <element name="code" disable="false" />
        <element name="embed" disable="false" />

    </elementRule>

    <attributeRule>
        <attribute name="src">
<allowedPattern><![CDATA[["']?\\s*http://.*]]></allowedPattern>
        </attribute>
        <attribute name="href">
<notAllowedPattern><![CDATA[(?i:script)]]></notAllowedPattern>
<notAllowedPattern><![CDATA[(?i:\\.css)]]></notAllowedPattern>
        </attribute>
        <attribute name="style">
<notAllowedPattern><![CDATA[(?i:expression)]]></notAllowedPattern>
<notAllowedPattern><![CDATA[(?i:alert)]]></notAllowedPattern>
        </attribute>
    </attributeRule>

```

```
</config>
```

이제 이XML을 이용해서 글을 쓰기전에 걸러주는 로직을 만들겠습니다.

BoardController에 writeProcess 매서드를 다음과 같이 변경합니다.

```
<net.gondr.controller/BoardController.java>
```

```
@RequestMapping(value="write", method=RequestMethod.POST)
public String writeProcess(BoardVO board, HttpSession session, Errors errors) {
    //올바른 값인지 벨리데이팅
    new BoardValidator().validate(board, errors);
    if(errors.hasErrors()) {
        return "board/write"; //에러가 존재하면 글쓰기 페이지로 보냄.
    }
    //여기는 인터셉터에 의해서 로그인하지 않은 사용자는 막히게 될 것이기 때문에
    그냥 에러처리 없이 user를 불러써도 된다.
    UserVO user = (UserVO)session.getAttribute("user");
    //로그인한 사용자의 아이디를 글쓴이로 등록하고
    board.setWriter(user.getUserid());

    LucyXssFilter filter = XssSaxFilter.getInstance("lucy-xss-sax.xml");
    String clean = filter.doFilter(board.getContent());
    board.setContent(clean);
    //실제 DB에 글을 기록함.
    boardService.writeArticle(board);
    return "redirect:/board";
}
```

위와 같이 3줄이면 설정에 의해서 위험한태그들은 알아서 걸러지게 됩니다. 셋팅에서는 일일이 해주었지만 공격가능성이 없는 h1, img, a 태그등은 별도의 설정을 하지 않아도 filter에서 허용됩니다. 만약 모든 것을 다 차단하고 싶다면 XssPreventer를 사용하면 됩니다. 해당 태그는 모든 HTML 태그들을 무력화시킵니다. 또한 공격가능성이 있는 변형된 HTML태그들도 찾아서 무력화합니다. 하지만 우리가 만들 게시판에는 어울리지 않기 때문에 Filter를 사용하였습니다. 이제 글을 작성하고 잘 써지는 것을 확인하면 됩니다. 다양한 스마트에디터의 기능을 활용해서 정상적으로 잘 쓰여지는지 확인해보세요.

이제 이 글을 보여지는 뷰페이지를 만들어보겠습니다.

글이 쓰여지는 것을 만들었던 순서대로 올라오면 됩니다. 다만 이번에는 read외에도 list, delete, update 등의 기능도 같이 만들면서 하겠습니다. (하나하나 하려면 너무 오래걸려요...)

boardMapper.xml에 다음과 같이 기능을 추가합니다.

```
<src/main/resources/mappers/boardMapper.xml>
```

```

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

<mapper namespace="net.gondr.mappers.BoardMapper">
  <insert id="write">
    INSERT INTO spring_boards (title, content, writer, writeDate)
    VALUES ( #{title}, #{content}, #{writer}, NOW() )
  </insert>
  <select id="list" resultType="BoardVO">
    SELECT * FROM spring_boards ORDER BY id DESC LIMIT #{start},
    #{cnt}
  </select>
  <update id="update">
    UPDATE spring_boards SET content = #{content}, title = #{title}
    WHERE id = #{id}
  </update>
  <delete id="delete">
    DELETE FROM spring_boards WHERE id = #{id}
  </delete>
  <select id="view" resultType="BoardVO">
    SELECT * FROM spring_boards WHERE id = #{id}
  </select>
  <select id="cnt" resultType="Integer">
    SELECT count(*) FROM spring_boards
  </select>
</mapper>

```

이제 이에 맞게 BoardDAO 인터페이스를 변경합니다.

<net.gondr.dao/BoardDAO.java>

```

package net.gondr.dao;

import java.util.List;

import net.gondr.domain.BoardVO;

public interface BoardDAO {
    //글을 쓰는 매서드
    public void write(BoardVO data);
    //글보기 매서드
    public BoardVO view(Integer id);
    //글리스트 보기
    public List<BoardVO> list(Integer start, Integer cnt);
    //글삭제
    public void delete(Integer id);
    //글 수정
    public void update(BoardVO data);

    //현재 글의 갯수

```

```
        public Integer getCnt();  
    }  
}
```

그리고 DAO구현은 다음과 같이 만들어줍니다.

```
package net.gondr.dao;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
import org.apache.ibatis.session.SqlSession;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
  
import net.gondr.domain.BoardVO;  
  
@Repository  
public class BoardDAOImpl implements BoardDAO{  
    @Autowired  
    private SqlSession session;  
  
    private final String namespace = "net.gondr.mappers.BoardMapper";  
  
    @Override  
    public void write(BoardVO data) {  
        session.insert(namespace + ".write", data);  
    }  
  
    @Override  
    public BoardVO view(Integer id) {  
        return session.selectOne(namespace + ".view", id);  
    }  
  
    @Override  
    public List<BoardVO> list(Integer start, Integer cnt) {  
        Map<String,Integer> paramMap = new HashMap<>();  
        paramMap.put("start", start);  
        paramMap.put("cnt", cnt);  
        return session.selectList(namespace + ".list", paramMap);  
    }  
  
    @Override  
    public void delete(Integer id) {  
        session.delete(namespace + ".delete", id);  
    }  
  
    @Override  
    public void update(BoardVO data) {  
        session.update(namespace + ".update", data);  
    }  
  
    @Override  
    public Integer getCnt() {  
        return session.selectOne(namespace + ".cnt");  
    }  
}
```

```
}  
  
}
```

이제 이 DAO가 올바르게 만들어졌는지 테스트하기 위해서 BoardDAOTest를 다음과 같이 변경후 @Test 애노테이션을 하나씩 주석처리하면서 테스트 해보기 바랍니다.

<src/test/java/net.gondr.dao/BoardDAOTest.java>

```
package net.gondr.dao;  
  
import java.util.List;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.test.context.ContextConfiguration;  
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
  
import net.gondr.domain.BoardVO;  
  
@RunWith(SpringJUnit4ClassRunner.class)  
@ContextConfiguration(locations =  
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})  
public class BoardDAOTest {  
    @Autowired  
    private BoardDAO dao;  
  
    @Test  
    public void createBoard() {  
        BoardVO board = new BoardVO();  
        board.setContent("글내용입니다. 테스트용 내용");  
        board.setTitle("글 제목입니다. 테스트용");  
        board.setWriter("gondr");  
  
        dao.write(board);  
    }  
  
    @Test  
    public void readBoard() {  
        BoardVO data = dao.view(1);  
  
        System.out.println(data.getTitle());  
        System.out.println(data.getContent());  
    }  
  
    @Test  
    public void getListBoard() {  
        List<BoardVO> list = dao.list(0, 10);  
  
        for(BoardVO board : list) {  
            System.out.println(board.getTitle());  
        }  
    }  
}
```

```

    }

    @Test
    public void getCnt() {
        Integer cnt = dao.getCnt();
        System.out.println(cnt);
    }

    @Test
    public void update() {
        BoardVO data = dao.view(1);
        data.setTitle("수정된 제목입니다");
        data.setContent("수정된 내용입니다.");
        dao.update(data);
    }

    @Test
    public void delete() {
        dao.delete(1);
    }
}

```

\*주의 : 한꺼번에 테스트를 돌리면 안됩니다. 한 개씩 순차적으로 주석을 하고 지워가며 테스트 하세요.

전부 올바르게 만들어졌다면 Service의 제작으로 넘어가면 됩니다.

BoardService를 다음과 같이 추가적으로 제작합니다.

<net.gondr.service/BoardService.java>

```

package net.gondr.service;

import java.util.List;

import net.gondr.domain.BoardVO;

public interface BoardService {
    //글쓰기
    public void writeArticle(BoardVO board);
    //글보기
    public BoardVO viewArticle(Integer id);
    //글 리스트 보기
    public List<BoardVO> getArticleList(Integer start, Integer cnt);
    //글 수정하기
    public void updateArticle(BoardVO board);
    //글 삭제하기
    public void deleteArticle(Integer id);
}

```



```
        //글 갯수 가져오기
        public Integer countArticle();
    }
}
```

그리고 다음과 같이 BoardServiceImpl을 구현합니다.

<net.gondr.service/BoardServiceImpl.java>

```
package net.gondr.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import net.gondr.dao.BoardDAO;
import net.gondr.domain.BoardVO;

@Service
public class BoardServiceImpl implements BoardService {
    @Autowired
    private BoardDAO dao;

    @Override
    public void writeArticle(BoardVO board) {
        dao.write(board);
    }

    public BoardVO viewArticle(Integer id) {
        return dao.view(id);
    }

    public List<BoardVO> getArticleList(Integer start, Integer cnt) {
        return dao.list(start, cnt);
    }

    public void updateArticle(BoardVO board) {
        dao.update(board);
    }

    public void deleteArticle(Integer id) {
        dao.delete(id);
    }

    public Integer countArticle() {
        return dao.getCnt();
    }
}
```

서비스에 대한 테스트 코드도 다음과 같이 작성할 수 있습니다.

<src/test/java/net.gondr.service/BoardServiceTest.java>

```
package net.gondr.service;
```

```

import java.util.List;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import net.gondr.domain.BoardVO;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{"file:src/main/webapp/WEB-INF/**/root-context.xml"})
public class BoardServiceTest {
    @Autowired
    BoardService service;

    @Test
    public void writeBoardTest() {
        BoardVO board = new BoardVO();
        board.setTitle("이번에도 테스트");
        board.setContent("이번에도 테스트 내용입니다. 테스트 테스트");
        board.setWriter("gondr");

        service.writeArticle(board);
    }

    @Test
    public void viewArticle() {
        BoardVO board = service.viewArticle(2);
        System.out.println(board.getTitle());
    }

    @Test
    public void getArticleList() {
        List<BoardVO> list = service.getArticleList(0, 10);

        for(BoardVO data : list) {
            System.out.println(data.getTitle());
        }
    }

    @Test
    public void updateArticle() {
        BoardVO board = service.viewArticle(2);
        board.setTitle("서비스에서 수정한 제목입니다.");
        board.setContent("서비스에서 수정한 내용입니다.");
        service.updateArticle(board);
    }

    @Test
    public void deleteArticle() {

```

```

        service.deleteArticle(2);
    }

    @Test
    public void countArticle() {
        System.out.println(service.countArticle());
    }
}

```

이제 서비스까지 만들었으니 글 보기 페이지의 Controller를 만들어줍시다. 다음과 같이 BoardController에 매서드를 추가합니다.

<net.gondr.controller/BoardController.java>

```

@RequestMapping(value="view/{id}", method=RequestMethod.GET)
public String viewArticle(@PathVariable Integer id, Model model) {
    BoardVO board = boardService.viewArticle(id);
    model.addAttribute("board", board);

    return "board/view";
}

```

이제 타임리프로 board/view 템플릿을 만들 차례입니다.

<WEB-INF/template/board/view.html>

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>
    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">
                <h2 th:text="${board.title}">글제목</h2>
                <div class="card">
                    <div class="card-header">
                        <span
th:text="${board.title}">글제목</span>
                        <span class="badge badge-primary"
th:text="${board.writer}">글쓴이</span>
                        <span class="badge badge-warning"
th:text="${#dates.format(board.writeDate, 'yyyy-MM-dd')}">작성일</span>
                    </div>
                    <div class="card-body"
th:utext="${board.content}">
                </div>
            </div>
        </div>
    </div>

```

```

        </div>
    </div>
</div>
<div class="row mt-3">
    <div class="col-10 offset-1 text-right">
        <a href="/board/list" class="btn
btn-primary">목록보기</a>
    </div>
</div>
</div>
<footer th:replace="/fragments/layout :: footer"></footer>
</body>
</html>

```

이전까지는 th:text를 썼으나 이번에는 utext를 써서 html이 그대로 출력될 수 있도록 해주었습니다. 또한 내장함수인 #dates를 써서 날짜 형식을 맞추어 출력되도록 해주었습니다.

**과제. 글보기 페이지에 작성한 사용자의 프로필 이미지가 나오게 하려면 어떻게 해야할까요?**

일단 boardMapper.xml에서 사용자의 이미지파일, 작성자 이름등을 가져오도록 변경해야 합니다. 또한 이 작업은 BoardVO를 변경시키게 됩니다. 정답은 다음장에 있습니다.

먼저 다음과 같이 BoardVO를 변경합니다.

<net.gondr.domain/BoardVO.java>

```

//상단 생략
    private String name;
    private String img;
    private Integer level;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getImg() {
        return img;
    }
//하단 생략

```

이제 boardMapper.xml의 view를 다음과 같이 변경합니다.

```
<select id="view" resultType="BoardVO">
    SELECT b.*, u.name, u.img, u.level FROM spring_boards AS b,
spring_users AS u WHERE id = #{id} AND userid = writer
</select>
```

이후 view.html을 다음과 같이 변경합니다.

<WEB-INF/template/board/view.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>
    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">
                <div class="row d-flex mb-2">
                    <div class="ml-2"
style="width:128px;height:128px;"
                    
                    </div>
                    <div class="col">
                        <h4 class="card-title"
th:text="${board.title}">글제목</h4>
                        <p class="card-text">
                            <span class="badge
badge-primary" th:text="@{${board.name} + '(' + ${board.writer} + ')'"
}">글쓴이</span><br>
                            <span class="badge
badge-secondary" th:text="@{'LV.[ ' + ${board.level} + ' ]' }">레벨</span><br>
                            <span class="badge
badge-warning" th:text="@{' 작성일 : ' + ${#dates.format(board.writeDate,
'yyyy-MM-dd')} }">작성일</span>
                        </p>
                    </div>
                </div>
                <div class="card">
                    <div class="card-body"
th:utext="${board.content}">
                    </div>
                </div>
            </div>
            <div class="row mt-3">
                <div class="col-10 offset-1 text-right">
```

```

<a href="/board/list" class="btn
btn-primary">목록보기</a>
</div>
</div>
</div>
<footer th:replace="/fragments/layout :: footer"></footer>
</body>
</html>

```

정상적으로 이미지가 표시되도록 UserController에 다음과 같이 이미지 라우팅 매서드를 하나 추가합니다.

<net.gondr.controller/UserController.java>

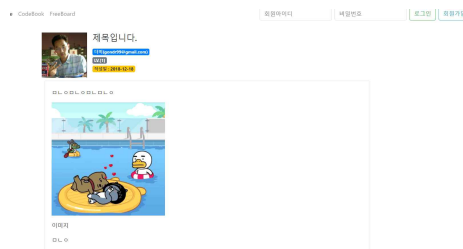
```

@RequestMapping(value="/profile/{file:.+}", method=RequestMethod.GET)
@ResponseBody
public byte[] getUserProfile(@PathVariable String file) throws IOException {
    String uploadPath = context.getRealPath("/WEB-INF/upload");
    System.out.println(file);
    File profile = new File(uploadPath + File.separator + file );
    FileInputStream in = new FileInputStream(profile);

    return IOUtils.toByteArray(in);
}

```

이제 프로필 이미지를 업로드한 회원으로 글을 작성후 해당 글을 보려고 하면 다음과 같이 나타날 것입니다.



PathVariable에서 {file 뒤쪽에 .+ 표시는 한 개 이상의 점문자를 허용한다는 뜻입니다. 우리가 요청하는 파일명이 확장자까지 함께 있기 때문에 이를 반드시 표기해줘야 정상적으로 구동됩니다. 그렇지 않으면 스프링이 자동으로 확장자를 제거하고 파일명만 변수에 담아주기 때문입니다.

수정과 삭제는 차후 구현하기로 하고 먼저 목록부터 구현해봅시다. 글목록을 불러올 때 페이지를 처리할 수 있도록 net.gondr.domain에 다음과 같이 Criteria 클래스를 작성합니다.

<net.gondr.domain/Criteria.java>

```
package net.gondr.domain;

public class Criteria {
    private Integer page;
    private Integer perPageNum;
    private Integer perChapterNum;
    private String keyword;

    public Criteria () {
        this.page = 1;
        this.perPageNum = 10;
        this.perChapterNum = 5;
        this.keyword = null;
    }
    public Integer getPage() {
        return page;
    }
    public void setPage(Integer page) {
        this.page = page;
    }
    public Integer getPerPageNum() {
        return perPageNum;
    }
    public void setPerPageNum(Integer perPageNum) {
        this.perPageNum = perPageNum;
    }
    public Integer getPerChapterNum() {
        return perChapterNum;
    }
    public void setPerChapterNum(Integer perChapterNum) {
        this.perChapterNum = perChapterNum;
    }
    public String getKeyword() {
        return keyword;
    }
    public void setKeyword(String keyword) {
        this.keyword = keyword;
    }
}
```

이제 이 객체를 기반으로 하는 /board/list에 대한 컨트롤러를 BoardController에 다음과 같이 작성합니다.

<net.gondr.controller/BoardController.java>

```
@RequestMapping(value="list", method=RequestMethod.GET)
public String viewList(Criteria criteria, Model model) {
    List<BoardVO> list = boardService.getArticleList( (criteria.getPage() - 1)
* criteria.getPerPageNum(), criteria.getPerPageNum());
```

```
        model.addAttribute("list", list); //리스트에 더해서
    return "board/list";
}
```

이제 이 값을 가지고 보여줄 수 있도록 list.html을 다음과 같이 작성합니다.

```
</WEB-INF/template/board/list.html>
```

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
  <header th:replace="/fragments/layout :: header"></header>
  <div class="container">
    <div class="row">
      <div class="col-10 offset-1">
        <h2>코드북 게시판</h2>

        <table class="table table-striped">
          <tr>
            <th>글번호</th>
            <th width="50%">글제목</th>
            <th>작성자</th>
            <th>작성일</th>
          </tr>
          <tr th:each="board : ${list}">
            <td th:text="${board.id}">글번호</td>
            <td><a th:href="@{'/board/view/' +
${board.id} }" th:text="${board.title}">글제목</a></td>
            <td
th:text="${board.writer}">작성자</td>
            <td
th:text="${board.writeDate}">작성일</td>
          </tr>
        </table>

        </div>
        <div class="row mt-3">
          <div class="col-10 offset-1 text-right">
            <a href="/board/write" class="btn
btn-primary">글작성</a>
          </div>
        </div>
      </div>
    </div>
  </div>
  <footer th:replace="/fragments/layout :: footer"></footer>
</body>
```



</html>

타임리프의 th:each는 \* 로 반복할 변수를 표시해줍니다. 그리고 마치 java의 foreach문처럼 사용이 가능합니다.

이를 통해 tr을 반복적으로 그려줌으로서 게시글들을 그려줄 수 있습니다. 좀 더 자세한 설명은 타임리프 공식문서를 참조하세요(영어입니다.)

/board/list에 접근하면 다음과 같이 화면이 나오게 됩니다.

 CodeBook [Home](#) [CodeBook](#) [FreeBoard](#)

### 코드북 게시판

글번호	글제목	작성자	작성일
14	제목입니다.	gondr99@gmail.com	2018-12-18
13	아주 긴 제목입니다. 어찌될런지요아주 긴 제목입니다. 어찌될런지요아주 긴 제목입니다. 어찌될런지요아주 긴 제목입니다. 어찌될런지요	gondr@nate.com	2018-11-27
12	이번에도 테스트	gondr	2018-11-27
11	글 제목입니다. 테스트용	gondr	2018-11-27
4	스크립트 글제목 테스트	gondr@nate.com	2018-11-26
3	글 작성입니다. 이번엔 컨트롤러	gondr@nate.com	2018-11-26
2	서비스에서 수정한 제목입니다.	gondr	2018-11-26

글작성

criteria를 활용하여 페이징을 처리하였기 때문에 페이지별로 글을 보여주는 것까지 자동으로 처리됩니다.

이제 페이지네이션을 처리해보도록 하겠습니다. 지금 사용한 Criteria객체를 조금만 더 활용하면 됩니다. Criteria.java 를 다음과 같이 작성합니다.

<net.gondr.domain/Criteria.java>

```
package net.gondr.domain;

public class Criteria {
    private Integer page;
    private Integer perPageNum;
    private Integer perChapterNum;
    private String keyword;

    private boolean prev;
    private boolean next;
    private Integer start;
    private Integer end;
    private Integer totalPage;

    public Criteria () {
```

```

        this.page = 1;
        this.perPageNum = 10;
        this.perChapterNum = 5;
        this.keyword = null;
        this.prev = true;
        this.next = true;
    }

    public void Calculate(Integer total) {
        //전체 글의 갯수인 total을 기반으로 전체 페이지수와 페이지 시작, 끝
        그리고 이전, 다음 챕터또한 만들어낸다.
        this.totalPage = (int) Math.ceil((double)total / this.perPageNum);
        if (this.totalPage == 0) this.totalPage = 1;
        this.end = (int) Math.ceil( (double)this.page / this.perChapterNum
    ) * this.perChapterNum;
        this.start = this.end - this.perChapterNum + 1; //시작번호

        if(this.end > this.totalPage) {
            this.end = this.totalPage;
            this.next = false;
        }

        if(this.start == 1) {
            this.prev = false;
        }
    }

    //getter setter 생략
}

```

Criteria의 Calculate를 실행하기 위해 다음과 같이 Controller의 매서드를 변경합니다.

<net.gondr.controller/BoardController.java>

```

@RequestMapping(value="list", method=RequestMethod.GET)
public String viewList(Criteria criteria, Model model) {
    List<BoardVO> list = boardService.getArticleList( (criteria.getPage() - 1)
* criteria.getPerPageNum(), criteria.getPerPageNum());
    model.addAttribute("list", list); //리스트에 더해서

    Integer cnt = boardService.countArticle();
    criteria.Calculate(cnt);

    return "board/list";
}

```

이제 이 Criteria를 화면에 표시하기 위해 List.html을 다음과 같이 변경합니다.

<WEB-INF/template/board/list.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>
    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">
                <h2>코드북 게시판</h2>

                <table class="table table-striped">
                    <tr>
                        <th>글번호</th>
                        <th width="50%">글제목</th>
                        <th>작성자</th>
                        <th>작성일</th>
                    </tr>
                    <tr th:each="board : ${list}">
                        <td th:text="${board.id}">글번호</td>
                        <td><a th:href="@{'/board/view/' +
${board.id} }" th:text="${board.title}">글제목</a></td>
                        <td
th:text="${board.writer}">작성자</td>
                        <td
th:text="${board.writeDate}">작성일</td>
                    </tr>
                </table>
            </div>
            <div class="row mt-3">
                <div class="col-10 offset-1 text-right">
                    <a href="/board/write" class="btn
btn-primary">글작성</a>
                </div>
            </div>
        </div>
        <nav>
            <ul class="pagination justify-content-center">
                <li class="page-item" th:if="${criteria.prev}">
                    <a class="page-link"
th:href="@{'/board/list?page=' + ${criteria.start - 1} }" aria-label="Previous">
                        <span
aria-hidden="true">&laquo;</span>
                    </a>
                </li>
                <li>
                    <th:block th:each="i : ${#numbers.sequence(criteria.start,
```

```

criteria.end)}}">
                <li class="page-item" >
                    <a class="page-link"
th:href="@{'/board/List?page=' + ${i} }" th:text="${i}">인덱스 번호</a>
                </li>
            </th:block>
            <li class="page-item" th:if="${criteria.next}">
                <a class="page-link"
th:href="@{'/board/List?page=' + ${criteria.end + 1} }" aria-label="Next">
                    <span
aria-hidden="true">&raquo;</span>
                </a>
            </li>
        </ul>
    </nav>
</div>
<footer th:replace="/fragments/layout :: footer"></footer>
</body>
</html>

```

페이지네이션을 작성했습니다. 다만 이번에는 만들어진 리스트를 순회하지 않고 시작점부터 끝점까지를 for문을 돌려야 했기 때문에 th:block 태그를 활용하여 시작부터 끝까지 리스트를 생성시켜서 페이지네이션을 출력했습니다.

이렇게 게시판의 리스트 페이지와 페이지네이션까지 완성했습니다. 하지만 아직 글보기에서 우리가 작성한 코드가 올바르게 나오지 않는다는 것을 볼 수 있습니다. 다음과 같이 말이죠.

```

그림도 있고요

코드도 있어요

@Test
public void createBoard() {
    BoardVO board = new BoardVO();
    board.setContent("글내용입니다. 테스트용 내용");
    board.setTitle("글 제목입니다. 테스트용");
    board.setWriter("gondr");

    dao.write(board);
}

```

이부분의 html코드를 보면 다음과 같이 작성되어 있습니다.

```

<pre class="language-java"> == $0
<code>
    "@Test
    public void createBoard() {
        BoardVO board = new BoardVO();
        board.setContent("글내용입니다. 테스트용 내용");
        board.setTitle("글 제목입니다. 테스트용");
        board.setWriter("gondr");

        dao.write(board);
    }"
</code>
</pre>

```

이 code태그들은 prism이라는 css 파일이 있어야 정상적으로 보여집니다. 타이니 MCE의 코드 하이라이팅은 prism으로 만들어져 있기 때문입니다. prism 홈페이지에 가서 다운받아 보겠습니다.

<https://prismjs.com/>

우리는 app.js에서 우리가 지원하는 언어를 markup, javascript, css, php, python, java, c, csharp, cpp 로 지정했습니다. 이에 맞추어서 다운로드 페이지에서 테마와 지원언어를 선택해주세요.

아래에서는 Okaidia 테마와 위에서 설정된 언어들을 체크했습니다.

Compression level:

- ☐ Development version
- ☒ Minified version

**Core**

- ☒ Core (6.42KB)

**Themes**

- ☐ Default (2.19KB)
- ☐ Dark (2KB)
- ☐ Funky (1.98KB)
- ☒ Okaidia ocoadia (1.75KB)
- ☐ Twilight remybach (3.99KB)
- ☐ Coy tshedor (4.06KB)
- ☐ Solarized Light hectormatos2011 (2.52KB)
- ☐ Tomorrow Night Rosey (1.71KB)

**Languages**

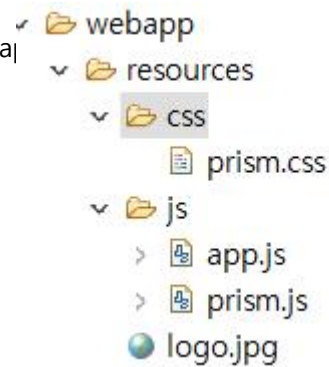
- ☐ Select/unselect all
- ☒ Markup + HTML + XML + SVG + MathML (0.97KB)
- ☒ CSS (1.02KB)
- ☒ C-like (0.7KB)
- ☒ JavaScript (1.67KB)
- ☐ GameMaker Language LiarOnce
- ☐ Go arnehormann (0.71KB)
- ☐ GraphQL Golmote (0.43KB)
- ☐ Groovy robfletcher (1.6KB)
- ☐ Haml Golmote (2.12KB)
- ☐ Handlebars Golmote (0.79KB)
- ☐ PowerShell nauzilus (4.17KB)
- ☐ Processing Golmote (0.58KB)
- ☐ Prolog Golmote (0.34KB)
- ☐ .properties Golmote (0.27KB)
- ☐ Protocol Buffers just-boris (0.3KB)
- ☐ Pug Golmote (2.7KB)

플러그인으로는 Line Numbers와 Line Highlight, Toolbar, Copy to Clipboard Button을 체크했습니다.

## Plugins

- |   |   |  |
|---|---|--|
| <input checked="" type="checkbox"/> <a href="#">Line Highlight</a> (3.63KB)           | <input type="checkbox"/> <a href="#">JSONP Highlight</a><br>nauzilus (2.19KB)         | <input type="checkbox"/> <a href="#">Command Line</a><br>chriswells0 (3.2KB)                       |
| <input checked="" type="checkbox"/> <a href="#">Line Numbers</a> kuba-kubula (2.66KB) | <input type="checkbox"/> <a href="#">Highlight Keywords</a><br>vkbansal (0.19KB)      | <input type="checkbox"/> <a href="#">Unescaped Markup</a> (1.51KB)                                 |
| <input type="checkbox"/> <a href="#">Show Invisibles</a> (0.67KB)                     | <input type="checkbox"/> <a href="#">Remove initial line feed</a><br>Golmote (0.32KB) | <input type="checkbox"/> <a href="#">Normalize Whitespace</a><br>zeitgeist87 (2.44KB)              |
| <input type="checkbox"/> <a href="#">Autolinker</a> (1.14KB)                          | <input type="checkbox"/> <a href="#">Previewers</a><br>Golmote (17.01KB)              | <input type="checkbox"/> <a href="#">Data-URI Highlight</a><br>Golmote (1.46KB)                    |
| <input type="checkbox"/> <a href="#">WebPlatform Docs</a> (3.37KB)                    | <input type="checkbox"/> <a href="#">Autoloader</a> Golmote (3.07KB)                  | <input checked="" type="checkbox"/> <a href="#">Toolbar</a><br>mAdhaTTah (2.67KB)                  |
| <input type="checkbox"/> <a href="#">Custom Class</a><br>dvkndn (0.36KB)              | <input type="checkbox"/> <a href="#">Keep Markup</a><br>Golmote (1.24KB)              | <input checked="" type="checkbox"/> <a href="#">Copy to Clipboard Button</a><br>mAdhaTTah (0.91KB) |
| <input type="checkbox"/> <a href="#">File Highlight</a> (1.44KB)                      |   |  |
| <input type="checkbox"/> <a href="#">Show Language</a><br>nauzilus (1.94KB)           |   |  |

전부되었다면 js 파일과 css파일을 다운받아서 프로젝트 webapp  
어줍니다.



이렇게 넣어준 파일을 view.html에 다음과 같이 링크를 추가  
합니다.

<WEB-INF/template/board/view.html>

```
<link rel="stylesheet" href="/resources/css/prism.css">
<script src="/resources/js/prism.js"></script>
```

이제 다시 글보기 페이지를 가보면 다음과 같이 적용된 것을 볼 수 있습니다.

코드도 있어요

```
@Test
public void createBoard() {
    BoardVO board = new BoardVO();
    board.setContent("글내용입니다. 테스트용 내용");
    board.setTitle("글 제목입니다. 테스트용");
    board.setWriter("gondr");

    dao.write(board);
}
```

(2018년 12월 현재 line number 플러그인이 정상적으로 적용되지 않습니다. tinymce 자체의 코드를 수정해야하나, webjar 방식은 그것이 불가능하여 미결상태로 진행합니다.)

이제 인터셉터를 붙여서 글쓰기페이지의 경우 로그인한 사용자만 나오도록 변경해보겠습니다. servlet-context.xml에 다음과 같이 인터셉터의 url을 추가합니다.

<spring/appServlet/servlet-context.xml>

```
<!-- 인터셉터 맵핑 -->
<interceptors>
    <interceptor>
        <mapping path="/user/register"/>
        <beans:ref bean="nonAuthInterceptor"/>
    </interceptor>

    <interceptor>
        <mapping path="/user/logout"/>
        <mapping path="/board/write"/>
        <beans:ref bean="authInterceptor"/>
    </interceptor>
</interceptors>
```

위와 같이 한줄만 추가해주면 글쓰는 부분에 대한 권한처리는 끝나게 됩니다.

이제 로그인한 사용자가 해당 글의 작성자와 동일할 때 수정, 삭제 버튼이 나오도록 변경해보겠습니다.

글보기에서 정보 일치 여부에 따라 if를 써서 보여주게 하면 됩니다. 이부분은 java 컨트롤러의 수정없이 view 단의 타임리프트만으로도 충분히 가능합니다. view.html을 다음과 같이 수정합니다.

<WEB-INF/template/board/view.html>

```
<div class="row mt-3">
    <div class="col-10 offset-1 text-right">
        <th:block th:if="${session.user != null and session.user.userid
== board.writer}">
            <a th:href="@{'/board/write/' + ${board.id}}" class="btn
btn-success">수정하기</a>
            <a th:href="@{'/board/delete/' + ${board.id}}" class="btn
btn-danger">삭제하기</a>
        </th:block>
        <a href="/board/list" class="btn btn-primary">목록보기</a>
    </div>
</div>
```

상단부와 하단부의 코드는 생략했습니다. 기존에 목록보기 버튼이 있던 곳에 위와 같이 코드를 작성하여 로그인 상태이고 해당 상태의 id가 글쓴이와 동일하다면 write와 delete에 대한 버튼을

부여합니다. 권한은 나중에 일괄적으로 수정하기로 하고 일단 수정을 구현해보겠습니다. write에 대한 컨트롤러 로직을 변화시키기 위해서 BoardController를 변경시켜야 합니다. BoardController의 viewModPage를 다음과 같이 작성합니다.

<net.gondr.controller/BoardController.java>

```
@RequestMapping(value="write/{id}", method=RequestMethod.GET)
public String viewModPage(Model model, @PathVariable("id") Integer id,
    HttpSession session, RedirectAttributes rttr) {
    BoardVO data = boardService.viewArticle(id);
    UserVO user = (UserVO)session.getAttribute("user");

    //해당 글의 수정 권한여부 판단
    if(!data.getWriter().equals(user.getUserid())) {
        rttr.addFlashAttribute("msg", "수정할 권한이 없습니다");
        return "redirect:/board/list";
    }else {
        model.addAttribute("boardVO", data);
        return "board/write";
    }
}
```

요청이 들어오면 해당 글을 불러와서 현재 로그인된 사용자의 아이디와 일치하는지를 검사후 수정페이지로 보낼 것인지를 결정하게 됩니다. 따라서 이 페이지도 차후에 로그인 인터셉터가 사전 검사를 해주어야 합니다. 아직 사용하지는 않았으나 메시지를 위해서 RedirectAttribute에 메시지에 추가해주었습니다.

일단 자신이 올바른 사용자라면 글쓰기 페이지로 이동할 것입니다. 글쓰기 페이지를 이제 다음과 같이 수정하여 글 쓰기와 수정상태가 다르도록 하겠습니다.

<WEB-INF/template/board/write.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>
    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">
                <h2 th:if="${boardVO.id == null}">글작성</h2>
                <h2 th:if="${boardVO.id != null}">수정하기</h2>
                <form method="post" th:action="@{/board/write}"
                    enctype="multipart/form-data" th:object="${boardVO}">
```





```

RedirectAttributes rttr) {
    //올바른 값인지 벨리데이팅
    new BoardValidator().validate(board, errors);
    if(errors.hasErrors()) {
        return "board/write"; //에러가 존재하면 글쓰기 페이지로 보냄.
    }

    //여기는 인터셉터에 의해서 로그인하지 않은 사용자는 막히게 될 것이기 때문에
    그냥 에러처리 없이 user를 불러써도 된다.
    UserVO user = (UserVO)session.getAttribute("user");

    //글 수정시에는 권한 검사
    if(board.getId() != null) {
        BoardVO data = boardService.viewArticle(board.getId());
        if(data == null || !user.getUserid().equals(data.getWriter())) {
            rttr.addFlashAttribute("msg", "권한이 없습니다.");
            return "redirect:/board/view/" + board.getId(); //해당 글로
다시 돌려보내기
        }
    } //로그인한 사용자의 아이디를 글쓴이로 등록하고
    board.setWriter(user.getUserid());

    LucyXssFilter filter = XssSaxFilter.getInstance("lucy-xss-sax.xml");
    String clean = filter.doFilter(board.getContent());
    board.setContent(clean);

    //실제 DB에 글을 기록함.
    if(board.getId() != null) {
        //글 수정
        boardService.updateArticle(board);
    }else {
        //글 작성
        boardService.writeArticle(board);
    }

    return "redirect:/board/list";
}

```

글 수정요청이 들어왔을 때 다시한번 해당 글에 대한 수정권한이 있는 사용자인지를 검색해서 권한이 있을 경우에만 update를 수행하게 해줍니다. 이미 짜여진 틀안에서 만드는 것이기 때문에 수정코드가 그리 많지 않으면서 제 기능을 다 하도록 만들어줄 수 있습니다.

여기까지 프로젝트를 진행하면서 아마도 다음과 같은 에러메시지를 보고 있는 사람들이 있을 것입니다.

s\_dbe61973-1cba-48cc-b011-b41cce75f987\_프로필.png  
[java.io.FileNotFoundException: D:\workspace\spring\metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\springboard\WEB-INF\upload\s\\_dbe61973-1cba-48cc-b011-b](#)

이미지 프로필 파일을 찾을 수 없을 때 나오게 되는 오류입니다. 이미지가 없는 사용자의 경우에도 위와 같은 메시지를 볼 수 있습니다.

과제 . 이미지가 없는 사용어나 오류로 인해 이미지를 표시할 수 없을 때 표시할 기본이미지를 하나 놓고 이를 이용해서 오류시 기본 이미지가 표시되도록 코드를 작성하세요

정답은 하단에 있습니다. 가급적 보지 말고 해결하세요.

기본 사용자 이미지로 사용할 파일을 다운받고 이름을 default.png로 하여 WEB-INF의 upload 폴더에 넣어줍니다. 구글 이미지 검색에서 default profile image 라고 검색하면 상당히 많은 이미지들이 나올 것입니다.

그리고 UserController의 파일이미지 서비스 부분을 다음과 같이 변경합니다.

```
@RequestMapping(value={"profile", "profile/{file:.+}"}, method=RequestMethod.GET)
@ResponseBody
public byte[] getUserProfile(@PathVariable Optional<String> file) throws
IOException {
    String uploadPath = context.getRealPath("/WEB-INF/upload");
    String imgFile = "default.png";
    if(file.isPresent()) {
        imgFile = file.get();
    }
    try {
        File profile = new File(uploadPath + File.separator + imgFile );
        FileInputStream in = new FileInputStream(profile);
        return IOUtils.toByteArray(in);
    } catch (FileNotFoundException e) {
        File profile = new File(uploadPath + File.separator +
"default.png");
        FileInputStream in = new FileInputStream(profile);
        return IOUtils.toByteArray(in);
    }
}
```

```
}
```

경로 2개에 대해서 모두 받도록 설정하고 Optional을 이용해서 선택적으로 들어오는 경로변수를 받아서 저장한후 이를 이용해서 이미지를 보여주도록 했습니다. 또한 이 때 만약 이미지 파일이 존재하지 않아서 에러가 발생한다면 try catch에 의해서 기본 이미지가 서비스되도록 변경했기 때문에 어떤 경우에도 이미지가 안나오는 경우는 없게 됩니다.(default.png 파일이 없는 경우는 예외입니다....--)

이제 삭제만 구현하면 게시판의 crud는 모두 끝납니다.

BoardController에서 다음과같이 deleteArticle 매서드를 만들어줍니다.

<net.gondr.controller/BoardController.java>

```
@RequestMapping(value="delete/{id}", method=RequestMethod.GET)
public String deleteArticle(@PathVariable("id") Integer id, HttpSession session,
RedirectAttributes rttr) {
    UserVO user = (UserVO)session.getAttribute("user");
    BoardVO data = boardService.viewArticle(id);

    if(!user.getUserid().equals(data.getWriter())) {
        rttr.addFlashAttribute("msg", "삭제 권한이 없습니다.");
        return "redirect:/board/view/" + data.getId();
    }

    boardService.deleteArticle(id);
    rttr.addFlashAttribute("msg", "성공적으로 삭제되었습니다.");
    return "redirect:/board/list";
}
```

여기까지 만들었던 기능들을 인터셉터에 등록하여 로그인한 사람만 접근할 수 있도록 수정합니다.

<WEB-INF/spring/appServlet/servlet-context.xml>

```
<!-- 인터셉터 맵핑 -->
<interceptors>
    <interceptor>
        <mapping path="/user/register"/>
        <beans:ref bean="nonAuthInterceptor"/>
    </interceptor>

    <interceptor>
        <mapping path="/user/logout"/>
        <mapping path="/board/write/*"/>
        <mapping path="/board/delete/*"/>
        <beans:ref bean="authInterceptor"/>
    </interceptor>
</interceptors>
```

```
</interceptors>
```

추가적인 기능을 구현하기 위해 앞서서 편의성 증진을 위해 2가지 기능을 추가해보겠습니다. 첫번째로 풋터의 유동적인 위치입니다. 현재 풋터가 fixed로 놓여져있기 때문에 화면이 늘어나게 되면 화면 하단부가 footer에 가려지게 됩니다. 이를 스크립트를 이용해서 처리하도록 하겠습니다.

layout.html에 다음과 같이 스크립트를 header에 추가합니다.

<WEB-INF/template/fragment/layout.html>

```
<head th:fragment="head">
    <link rel="stylesheet"
          href="/webjars/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="/webjars/jquery/3.3.1-1/jquery.min.js"></script>
    <script src="/webjars/popper.js/1.14.4/umd/popper.min.js"></script>
    <script src="/webjars/bootstrap/4.1.3/js/bootstrap.min.js"></script>
    <script>
        $(function(){
            if(document.body.clientHeight > window.innerHeight - 24){
                $(".footer").removeClass("fixed-bottom");
                $(".footer").addClass("mt-3");
            }
        });
    </script>
</head>
```

다음은 플래시 메시지를 띄워주는 부분입니다.

기존에 구조가 있기 때문에 간단하게 만들어집니다. 여태까지 보내주었던 플래시 메시지를 띄워주는 부분을 만들어보겠습니다.

다음과 같이 layout.html의 header부분에 메시징에 관련 된 코드를 추가로 작성합니다.

```
<header th:fragment="header" class="mb-4">
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        //네비게이션 내부 코드 생략
    </nav>
    <div th:if="${msg != null}" class="alert alert-warning alert-dismissible
    fade show" role="alert">
        <strong>알림!</strong> <span th:text="${msg}">메시지</span>
        <button type="button" class="close" data-dismiss="alert"
        aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
</header>
```

위와 같이 만들면 Redirect메시지가 있을 경우에 해당 메시지를 네비게이션바 아래에 표기하게 됩니다.

## 6. 게시판 검색 만들기

우리가 목표로했던 게시판도 이제 거의 만들어졌습니다. 게시판의 검색기능을 만들고 권한부분을 보강후 게시판에 대한 구현은 마무리 짓도록 하겠습니다.(댓글기능은 여러분의 재량으로 남겨둡니다)

게시판의 검색기능을 만들어보겠습니다.(사실 이미 만들어져있습니다. 조금만 수정하면 됩니다) 이전까지는 컨트롤러에서 크리테리아에 있는 값을 직접 꺼내서 이를 바탕으로 글을 불러왔습니다. 하지만 mybatis의 기능을 활용하여 아예 Criteria 객체 자체를 넘겨서 mybatis가 이를 처리하도록 해보겠습니다.

Criteria 객체에 다음과 같이 매서드 하나를 추가합니다. (페이지의 시작값을 알아내기 위한 매서드입니다.)

<net.gondr.domain/Criteria.java>

```
public Integer getPageStart() {  
    return (page - 1) * perPageNum;  
}
```

검색어인 키워드는 이미 셋팅되어 있습니다. BoardController를 다음과 같이 변경합니다.

<net.gondr.controller/BoardController.java>

```
@RequestMapping(value="list", method=RequestMethod.GET)  
public String viewList(Criteria criteria, Model model) {  
    List<BoardVO> list = boardService.getArticleList(criteria);  
    model.addAttribute("list", list); //리스트에 더해서  
  
    Integer cnt = boardService.countArticle(criteria);  
    criteria.Calculate(cnt);  
  
    return "board/list";  
}
```

이렇게 변경시에 getArticleList 와 countArticle 에서 에러가 발생합니다. 이를 정정하기 위해 BoardService부분도 다음과 같이 변경합니다.

<net.gondr.service/BoardService.java>

```
package net.gondr.service;  
  
import java.util.List;  
  
import net.gondr.domain.BoardVO;  
import net.gondr.domain.Criteria;
```

```

public interface BoardService {
    //글쓰기
    public void writeArticle(BoardVO board);
    //글보기
    public BoardVO viewArticle(Integer id);
    //글 리스트 보기
    public List<BoardVO> getArticleList(Criteria cri);
    //글 수정하기
    public void updateArticle(BoardVO board);
    //글 삭제하기
    public void deleteArticle(Integer id);
    //글 갯수 가져오기
    public Integer countArticle(Criteria cri);
}

```

이제 아마 BoardServiceImpl에서 맞지 않다고 오류를 낼 것입니다. 해당 파일도 변경해줍니다.

<net.gondr.service/BoardServiceImpl.java>

```

package net.gondr.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import net.gondr.dao.BoardDAO;
import net.gondr.domain.BoardVO;
import net.gondr.domain.Criteria;

@Service
public class BoardServiceImpl implements BoardService {
    @Autowired
    private BoardDAO dao;

    @Override
    public void writeArticle(BoardVO board) {
        dao.write(board);
    }

    public BoardVO viewArticle(Integer id) {
        return dao.view(id);
    }

    public List<BoardVO> getArticleList(Criteria cri) {
        return dao.list(cri);
    }

    public void updateArticle(BoardVO board) {
        dao.update(board);
    }
}

```



```

        public void deleteArticle(Integer id) {
            dao.delete(id);
        }

        public Integer countArticle(Criteria cri) {
            return dao.getCnt(cri);
        }
    }
}

```

이에 맞추어 BoardDao도 변경합니다.

<net.gondr.dao/BoardDao.java>

```

package net.gondr.dao;

import java.util.List;

import net.gondr.domain.BoardVO;
import net.gondr.domain.Criteria;

public interface BoardDAO {
    //글을 쓰는 매서드
    public void write(BoardVO data);
    //글보기 매서드
    public BoardVO view(Integer id);
    //글리스트 보기
    public List<BoardVO> list(Criteria cri);
    //글삭제
    public void delete(Integer id);
    //글 수정
    public void update(BoardVO data);

    //현재 글의 갯수
    public Integer getCnt(Criteria cri);
}

```

이제 BoardDaoImpl을 다음과 같이 변경합니다.

<net.gondr.dao/BoardDaoImpl.java>

```

package net.gondr.dao;

import java.util.List;

import org.apache.ibatis.session.SqlSession;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import net.gondr.domain.BoardVO;
import net.gondr.domain.Criteria;

@Repository
public class BoardDAOImpl implements BoardDAO{
    @Autowired
    private SqlSession session;

    private final String namespace = "net.gondr.mappers.BoardMapper";

    @Override
    public void write(BoardVO data) {
        session.insert(namespace + ".write", data);
    }

    @Override
    public BoardVO view(Integer id) {
        return session.selectOne(namespace + ".view", id);
    }

    @Override
    public List<BoardVO> list(Criteria cri) {
        return session.selectList(namespace + ".list", cri);
    }

    @Override
    public void delete(Integer id) {
        session.delete(namespace + ".delete", id);
    }

    @Override
    public void update(BoardVO data) {
        session.update(namespace + ".update", data);
    }

    @Override
    public Integer getCnt(Criteria cri) {
        return session.selectOne(namespace + ".cnt", cri);
    }
}

```

(그외의 Test코드들의 오류는 알아서 잡도록 합시다. 매개변수만 변경해주면 됩니다.)

마지막으로 핵심이 되는 BoardMapper를 다음과 같이 변경합시다.

<src/main/resource/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">

<mapper namespace="net.gondr.mappers.BoardMapper">
    <sql id="search">
        <if test="keyword != null">
            where title LIKE CONCAT('%', #{keyword}, '%') OR content
            LIKE CONCAT('%', #{keyword}, '%')
        </if>
    </sql>
    <insert id="write">
        INSERT INTO spring_boards (title, content, writer, writeDate)
        VALUES ( #{title}, #{content}, #{writer}, NOW() )
    </insert>
    <select id="list" resultType="BoardVO">
        SELECT id, title, writer, writeDate FROM spring_boards
        <include refid="search"></include>
        ORDER BY id DESC LIMIT #{pageStart}, #{perPageNum}
    </select>
    <update id="update">
        UPDATE spring_boards SET content = #{content}, title = #{title}
        WHERE id = #{id}
    </update>
    <delete id="delete">
        DELETE FROM spring_boards WHERE id = #{id}
    </delete>
    <select id="view" resultType="BoardVO">
        SELECT b.*, u.name, u.img, u.level FROM spring_boards AS b,
        spring_users AS u WHERE id = #{id} AND userid = writer
    </select>
    <select id="cnt" resultType="Integer">
        SELECT count(*) FROM spring_boards
        <include refid="search"></include>
    </select>
</mapper>

```

MyBatis의 동적-SQL 기능(Dynamic SQL)을 활용하여 조건이 있을 때와 없을때가 다르게 SQL이 실행되도록 하였습니다. 동적 SQL에 대해서 좀 더 자세히 알고 싶다면 링크에 들어가서 직접 읽어보기 바랍니다.

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

이제 keyword값으로 데이터를 전송하면 검색된 결과만 나올 것입니다. 예를 들어 다음과 같이 url 주소를 입력했다면

<http://localhost:9000/board/list?page=1&keyword=abc>

abc가 들어간 글만을 검색하고 페이징까지 완료될 것입니다. 이제 이 검색어를 입력할 공간, 즉 폼만 만들어주면 끝납니다.

다음과 같이 list.html을 변경시켜서 검색 폼을 만들어줍니다.

<WEB-INF/template/board/list.html>

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/layout :: head"></head>

<body>
    <header th:replace="/fragments/layout :: header"></header>
    <div class="container">
        <div class="row">
            <div class="col-10 offset-1">
                <h2>코드북 게시판</h2>
                <div class="row">
                    <div class="col-12 text-right">
                        <form class="form-inline
justify-content-end" id="searchForm">
                            <input type="text"
class="form-control mb-2 mr-sm-2" id="keyword" placeholder="검색어를 입력하세요">
                            <button type="button"
id="btnSearch" class="btn btn-primary mb-2">검색</button>
                        </form>
                    </div>
                </div>
                <table class="table table-striped">
                    <tr>
                        <th>글번호</th>
                        <th width="50%">글제목</th>
                        <th>작성자</th>
                        <th>작성일</th>
                    </tr>
                    <tr th:each="board : ${list}">
                        <td th:text="${board.id}">글번호</td>
                        <td><a th:href="@{'/board/view/' +
${board.id} }" th:text="${board.title}">글제목</a></td>
                        <td
th:text="${board.writer}">작성자</td>
                        <td
th:text="${board.writeDate}">작성일</td>
                    </tr>
                </table>
            </div>
            <div class="row mt-3">
                <div class="col-10 offset-1 text-right">
                    <a href="/board/write" class="btn
btn-primary">글작성</a>
                </div>
            </div>
        </div>
    </div>
</body>
```

```

        </div>
        <nav>
            <ul class="pagination justify-content-center">
                <li class="page-item" th:if="${criteria.prev}">
                    <a class="page-link"
th:href="@{'/board/List?page=' + ${criteria.start - 1} }" aria-label="Previous">
                        <span
aria-hidden="true">&laquo;</span>
                    </a>
                </li>
                <th:block th:each="i : ${#numbers.sequence(criteria.start,
criteria.end)}">
                    <li class="page-item" >
                        <a class="page-link"
th:href="@{'/board/List?page=' + ${i} }" th:text="${i}">인덱스 번호</a>
                    </li>
                </th:block>
                <li class="page-item" th:if="${criteria.next}">
                    <a class="page-link"
th:href="@{'/board/List?page=' + ${criteria.end + 1} }" aria-label="Next">
                        <span
aria-hidden="true">&raquo;</span>
                    </a>
                </li>
            </ul>
        </nav>
    </div>
    <footer th:replace="/fragments/layout :: footer"></footer>
    <script>
        $(function(){
            $("#btnSearch").on("click", function(e){
                let text = $("#keyword").val();
                location.href = '/board/list?keyword=' + text;
            });
            $("#searchForm").on("submit", function(e){
                $("#btnSearch").click();
                return false;
            });
        });
    </script>
</body>
</html>

```

여기까지 검색코드입니다.

여러분은 간단한 코드 게시판을 만들면서 스프링의 개략적인 기능들을 다뤄봤습니다. 다음 파트에서는 웹사이트의 사용성을 증가시키는 여러가지 작업들을 수행하겠습니다.

## **PART 3. 웹사이트의 완성도 올리기**

## 1. 게시판 버튼들의 보완

이전에 만들어둔 게시판은 검색이후 글을 들어가거나 다른 페이지로 이동시에 해당 내용이 유지되지 않는 문제를 가지고 있습니다. 버튼링크에 검색어가 유지되지 않기 때문입니다. 링크에 지속적으로 검색결과가 들어가야 합니다.

Criteria에 다음과 같이 매서드를 추가합니다.

<net.gondr.domain/Criteria.java>

```
public String getQuery(Integer page) {
    String q = "?page=" + page;
    if(this.keyword != null) {
        q += "&keyword=" + this.keyword;
    }
    return q;
}
```

그리고 이를 활용해 list.html의 출력부분을 다음과 같이 변경합니다.

<WEB-INF/template/board/list.html>

```
<nav>
    <ul class="pagination justify-content-center">
        <li class="page-item" th:if="${criteria.prev}">
            <a class="page-link" th:href="@{'/board/list' +
                ${criteria.getQuery(criteria.start - 1)} }" aria-label="Previous">
                <span aria-hidden="true">&laquo;</span>
            </a>
        </li>
        <th:block th:each="i : ${#numbers.sequence(criteria.start,
            criteria.end)}">
            <li class="page-item" >
                <a class="page-link" th:href="@{'/board/list' +
                    ${criteria.getQuery(i)} }" th:text="${i}">인덱스 번호</a>
            </li>
        </th:block>
        <li class="page-item" th:if="${criteria.next}">
            <a class="page-link" th:href="@{'/board/list' +
                ${criteria.getQuery(criteria.end + 1)} }" aria-label="Next">
                <span aria-hidden="true">&raquo;</span>
            </a>
        </li>
    </ul>
</nav>
```

getQuery함수를 이용해서 검색 키워드가 존재할 경우에는 키워드를 출력하게 합니다. 이를 통해

페이지를 클릭했을 때도 검색결과가 유지되어 페이지징되도록 합니다. 이러한 페이지징과 검색결과를 글보기에서 글을 보았다가 목록보기를 클릭해도 동일하게 유지되어야 합니다. 쿼리스트링을 글보기 페이지로 어떻게 가져갈 수 있을까요?

title에 링크를 찍어줄 때 단순히 글번호만을 찍는 것이 아니라 쿼리스트링도 함께 보내도록 다음과 같이 변경합니다.

<WEB-INF/template/board/list.html>

```
<table class="table table-striped">
  <tr>
    <th>글번호</th>
    <th width="50%">글제목</th>
    <th>작성자</th>
    <th>작성일</th>
  </tr>
  <tr th:each="board : ${list}">
    <td th:text="${board.id}">글번호</td>
    <td><a th:href="@{'/board/view/' + ${board.id} +
    ${criteria.getQuery(criteria.page)} }" th:text="${board.title}">글제목</a></td>
    <td th:text="${board.writer}">작성자</td>
    <td th:text="${board.writeDate}">작성일</td>
  </tr>
</table>
```

이제 글보기페이지에서 이를 받아줘야 하기 때문에 BoardController를 다음과 같이 변경해야 합니다.

<net.gondr.controller/BoardController.java>

```
@RequestMapping(value="/view/{id}", method=RequestMethod.GET)
public String viewArticle(@PathVariable Integer id, Model model, Criteria
criteria) {
    BoardVO board = boardService.viewArticle(id);
    model.addAttribute("board", board);

    return "board/view";
}
```

단순히 커맨드 객체를 적어주기만 하면 파라미터가 알맞게 들어가게 됩니다. 이제 View 페이지에서 이 객체를 활용할 수 있습니다. view.html에서 다음과 같이 코드를 변경합니다.

<WEB-INF/template/board/view.html>

```
<div class="row mt-3">
  <div class="col-10 offset-1 text-right">
    <th:block th:if="${session.user != null and session.user.userid
== board.writer}">
```






```
        <a th:href="@{'/board/write/' + ${board.id}}" class="btn
btn-success">수정하기</a>
        <a th:href="@{'/board/delete/' + ${board.id}}" class="btn
btn-danger">삭제하기</a>
    </th:block>
    <a th:href="@{'/board/List' + ${criteria.getQuery(criteria.page)}}
}" class="btn btn-primary">목록보기</a>
</div>
</div>
```

이제 목록보기 버튼을 눌렀을 때 이전에 보던 페이지로 그대로 이동하게 됩니다. 이를 통해 사용자 경험을 좀 더 증진시킬 수 있습니다.

## 2. 사용자 레벨업 시스템 만들기

사용자 컬럼에 레벨과 exp라는 칸을 만들었습니다. 이제 이 값을 글을 쓸 때마다 올려주는 방식으로 변경시켜보고, 일정이상 경험치가 쌓이면 레벨을 증가시키는 방식으로 프로그램을 만들어 보겠습니다.

먼저 레벨 테이블을 만들기 위해 다음과 같이 테이블을 만들어줍니다.

#	이름	종류	데이터정렬방식	보기	Null	기본값	설명	추가	실행
<input type="checkbox"/>	1	level 	int(11)		아니오	없음			
<input type="checkbox"/>	2	exp	int(11)		아니오	없음			

테이블 이름은 spring\_level로 합니다. 이 테이블에 레벨별 필요 경험치를 기록할 것입니다. 물론 일일이 기록하지는 않을 것입니다. userMapper.xml에 다음과 같이 쿼리를 추가합니다.

<src/main/resources/mappers/userMapper.xml>

```
<insert id="LevelData">
    INSERT INTO spring_level(level, exp) VALUES ( #{level}, #{exp} )
</insert>

<delete id="clearData">
    DELETE FROM `spring_level`
</delete>
```

이를 활용해서 레벨업 테이블을 자동으로 만들어주는 DAO와 서비스를 만들어주겠습니다. UserDao에 다음과 같이 2개의 매서드를 추가합니다.

<net.gondr.dao/UserDao.java>

```
//레벨업 테이블 클리어
public void deleteLevelTable();

//레벨업 테이블에 데이터 추가
public void insertLevelData(Integer level, Integer exp);
```

그리고 이 2개의 매서드에 대한 구현을 다음과 같이 작성합니다.

<net.gondr.dao/UserDaoImpl.java>

```
@Override
```

```

public void deleteLevelTable() {
    session.delete(namespace + ".clearData");
}

@Override
public void insertLevelData(Integer level, Integer exp) {
    Map<String, Integer> levelMap = new HashMap<String, Integer>();
    levelMap.put("level", level);
    levelMap.put("exp", exp);

    session.insert(namespace + ".levelData", levelMap);
}

```

이제 이 2개의 DAO를 아우르는 fillLevelTable이라는 서비스를 만들어봅시다.

<net.gondr.service/UserService.java>

```

//레벨테이블 채워주기
public void fillLevelTable(Integer max);

```

그리고 다음과 같이 구현을 합니다.

<net.gondr.service/UserServiceImpl.java>

```

@Override
public void fillLevelTable(Integer max) {
    userDao.deleteLevelTable(); //레벨데이터 삭제하고

    for(int i = 1; i <= max; i++) {
        Integer exp = (int)Math.floor(Math.pow( ((double)i -1) * 50 / 49,
2.5) * 10);
        userDao.insertLevelData(i, exp);
    }
}

```

경험치 산출식은 스프링이라기보단 대부분의 rpg 게임에서 사용하고 있는 산출식입니다. service에서 반복적으로 지정된 레벨까지의 필요 경험치를 집어넣어줍니다. 물론 넣기 전에 테이블을 삭제해주고 있구요.

이 서비스를 실행해주는 컨트롤러를 UserController에 다음과 같이 작성합니다.

<net.gondr.controller/UserController.java>

```

@RequestMapping(value="level/make", method=RequestMethod.GET)
public String makeLevel(RedirectAttributes rttr) {
    userService.fillLevelTable(200); //200레벨까지 경험치 생성
    rttr.addFlashAttribute("msg", "레벨 생성이 완료되었습니다.");
}

```

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

서버를 키고 /user/level/make 로 접속하면 레벨 200까지의 필요경험치 삽입후 리다이렉션 됩니다. 물론 이 코드는 최초 한번만 실행하고 주석처리해두는 것이 좋습니다. 아무나 들어와서 경험치 테이블을 리셋시키면 안되니까요. (관리자를 만들고 관리자만 해당 url에 접근 가능하도록 만드는 방법도 있습니다.

이제 경험치 테이블이 만들어졌습니다. 이제 글을 쓰고 난뒤 경험치가 올라가고 레벨업이 가능한지 체크해보도록 해보겠습니다.

경험치를 체크하고 올려주는 쿼리를 다음과 같이 userMapper.xml에 추가합니다.

<src/main/resources/mappers/userMapper.xml>

```
<select id="requireExp" resultType="Integer">
    SELECT exp FROM spring_level WHERE level = #{level}
</select>

<update id="setLevelAndExp">
    UPDATE spring_users SET level = #{level}, exp = #{exp} WHERE userid =
    #{userid}
</update>
```

그리고 UserDao에도 당연히 매서드를 추가해주어야겠지요?

<net.gondr.dao/UserDAO.java>

```
//필요경험치 알아내기
public Integer getRequireExp(Integer level);

//레벨과 경험치 셋팅하기
public void setLevelAndExp(UserVO user);
```

실제 구현도 다음과 같이 작성합니다.

<net.gondr.dao/UserDAOImpl.java>

```
@Override
public Integer getRequireExp(Integer level) {
    return session.selectOne(namespace + ".requireExp", level);
}

@Override
public void setLevelAndExp(UserVO user) {
    session.update(namespace + ".setLevelAndExp", user);
}
```

이제 서비스를 만들차례입니다. 서비스는 한 개의 매서드로 필요레벨과 경험치 확인후 레벨업 필요시 레벨업을 시키도록 할 것입니다.

<net.gondr.service/UserService.java>

```
//회원에게 경험치 추가하기
public UserVO addExp(String userId, Integer exp);
```

이 매서드의 구현은 다음과 같습니다.

<net.gondr.service/UserServiceImpl.java>

```
@Override
public UserVO addExp(String userId, Integer exp) {
    UserVO user = userDao.getUser(userId);
    user.setExp(user.getExp() + exp);
    Integer requireExp = userDao.getRequireExp(user.getLevel() + 1);

    if(user.getExp() >= requireExp ) {
        user.setExp(user.getExp() - requireExp);
        user.setLevel(user.getLevel() + 1);
    }

    //경험치 증가 처리후 DB에 저장
    userDao.setLevelAndExp(user);
}
```

이제 글이 써진 후에 이 addExp를 호출만 해주면 됩니다.

BoardController에 글이 써지는 부분을 다음과 같이 정정합니다.

<net.gondr.controller/BoardController.java>

```
package net.gondr.controller;

//임포트 생략

@Controller
@RequestMapping("/board/")
public class BoardController {
    @Autowired
    private ServletContext context;

    @Autowired
    private BoardService boardService;
```

```

@Autowired
private UserService userService;

//코드 중략

@RequestMapping(value="write", method=RequestMethod.POST)
public String writeProcess(BoardVO board, HttpSession session, Errors
errors, RedirectAttributes rttr) {
    //올바른 값인지 벨리데이팅
    new BoardValidator().validate(board, errors);
    if(errors.hasErrors()) {
        return "board/write"; //에러가 존재하면 글쓰기 페이지로 보냄.
    }

    //여기는 인터셉터에 의해서 로그인하지 않은 사용자는 막히게 될 것이기
    때문에 그냥 에러처리 없이 user를 불러써도 된다.
    UserVO user = (UserVO)session.getAttribute("user");

    //글 수정시에는 권한 검사
    if(board.getId() != null) {
        BoardVO data = boardService.viewArticle(board.getId());
        if(data == null ||
!user.getUserid().equals(data.getWriter())) {
            rttr.addFlashAttribute("msg", "권한이 없습니다.");
            return "redirect:/board/view/" + board.getId();
        }
    }
    //로그인한 사용자의 아이디를 글쓴이로 등록하고
    board.setWriter(user.getUserid());

    LucyXssFilter filter =
XssSaxFilter.getInstance("lucy-xss-sax.xml");
    String clean = filter.doFilter(board.getContent());
    board.setContent(clean);

    //실제 DB에 글을 기록함.
    if(board.getId() != null) {
        //글 수정
        boardService.updateArticle(board);
    }else {
        //글 작성
        boardService.writeArticle(board);
        user = userService.addExp(user.getUserid(), 5); //글을 한번
        쓸때마다 5의 exp를 지급
        session.setAttribute("user", user); //갱신후 세션값을

```

```

재설정해줌.
    }

    return "redirect:/board/list";
}

//코드 하단 생략
}

```

위와 같이 글작성후 exp5를 추가하여 우리가 만든 레벨업 루틴이 성공적으로 구동될 수 있도록 합니다.

가급적이면 저 5라는 숫자도 상수로 사용하기보단 상수변수로 활용하는 것이 더 좋습니다. (Ex. EXPData라는 스텁 클래스를 만들어서 해당 클래스에 스텁 변수로서 값을 셋팅하고 그 값을 이용하는 방법 - 이 예제의 해결방법은 아래에 코드로 적어두었습니다.)

먼저 ExpData라는 클래스를 domain패키지에 만들어줍니다.

<net.gondr.domain/ExpData.java>

```

package net.gondr.domain;

public class ExpData {
    public static Integer SMALL = 2;
    public static Integer MEDIUM = 5;
    public static Integer LARGE = 10;
}

```

그리고 이전에 BoardController에서 직접 값 5를 넣었던 부분을 다음과 같이 변경합니다.

<net.gondr.controller/BoardController.java>

```

//글 작성
boardService.writeArticle(board);
user = userService.addExp(user.getUserId(), ExpData.MEDIUM); //글을 한번 쓸때마다
5의 exp를 지급
session.setAttribute("user", user); //갱신후 세션값을 재설정해줌.

```

물론 임포트도 당연히 해주어야겠지요?

### 3. 기타 사항 정리하기

메인 메뉴의 링크들을 정리합시다. 다음과 같이 header의 링크들을 변경합시다.

<WEB-INF/template/fragment/layout.html>

```
<header th:fragment="header" class="mb-4">
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="/">
            
            CodeBook
        </a>
        <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarContent">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarContent">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="/">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link"
href="/board/list">CodeBook</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link"
href="/free/list">FreeBoard</a>
                </li>
            </ul>

            <form th:if="${session.user == null}" class="form-inline
my-2 my-lg-0" action="/user/Login" method="post">
                <input class="form-control mr-sm-2" type="text"
placeholder="회원아이디" name="userid">
                <input class="form-control mr-sm-2" type="password"
placeholder="비밀번호" name="password">
                <button class="btn btn-outline-success my-2 mr-2
my-sm-0" type="submit">로그인</button>
                <a href="/user/register" class="btn
btn-outline-info my-2 my-sm-0">회원가입</a>
            </form>
            <div th:if="${session.user != null}">
                <button class="btn btn-outline-success"
th:text="${session.user.name} + '님'"></button>
                <a href="/user/Logout" class="btn
```



```

    btn-outline-warning">로그아웃</a>
    </div>
  </div>
</nav>
<div th:if="${msg != null}" class="alert alert-warning alert-dismissible
fade show" role="alert">
    <strong>알림!</strong> <span th:text="${msg}">메시지</span>
    <button type="button" class="close" data-dismiss="alert"
aria-label="Close">
        <span aria-hidden="true">&times;</span>
    </button>
</div>
</header>

```

이제 게시글을 누르면 올바르게 게시글로 이동합니다. 자유게시판의 경우는 일부러 만들지 않았 습니다. 여태까지 배운 것을 활용해서 한번 만들어보라는 의미입니다. 자유게시판의 경우는 코드 탑재 기능만 제외하고 만들면 됩니다.

메인화면에서 코드보기 버튼의 링크도 다음과 같이 정정합니다.

<WEB-INF/template/index.html>

```

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">

<head th:replace="fragments/Layout :: head"></head>

<body>
    <header th:replace="fragments/Layout :: header"></header>

    <div class="container">
        <div class="jumbotron">
            <h1 class="display-4">양영디지털고 CodeBook!</h1>
            <p class="lead">코드북은 우리가 일상생활에서 코드를 작성하면서
부딪혔던 문제들과 해결책을 공유하기 위해 올리는 공간입니다.</p>
            <hr class="my-4">
            <p>내가 고생하며 해결했던 일은 언젠가 다른 프로그래머에게도
고통으로 다가옵니다. 이를 도와줄 수 있는 사이트! Code Book입니다.</p>
            <a class="btn btn-primary btn-lg" href="/board/List"
role="button">보러가기</a>
        </div>
    </div>
    <footer th:replace="fragments/Layout :: footer"></footer>
</body>
</html>

```

## 4. 차후 개발사항

가. 첫번째로 개발할 사항은 자유게시판입니다. 자유게시판은 사이트의 성격상 이미지 첨부 없이 오로지 텍스트로만 작성하는 게시판이 될 것입니다.

나. 두번째로 개발할 것은 댓글기능입니다. 각각의 글에 댓글이 작성될 수 있도록 해야 합니다. 이는 코드북과 자유게시판 모두 마찬가지입니다. 또한 댓글에는 반드시 페이징 기능도 포함되어야 합니다.

다. 세번째는 레벨업에 따른 권한 관리입니다. 사용자가 레벨업이 될수록 특정 권한이 생기도록 하는 부분들을 프로그래밍해준다면 좀 더 나은 결과를 볼 수 있습니다.

라. 마지막으로 네번째는 관리자 도구를 만드는 것입니다. 관리자도구는 프로그래밍으로 하지 않아도 웹 UI로만 웹사이트를 관리할 수 있도록 만들어주어야 합니다.

위의 4가지 사항들을 보완한다면 여러분이 지금 만든 사이트도 충분히 배포가 가능한 수준의 사이트입니다. 여기서 끝내지 말고 조금더 연습해서 수준있는 웹사이트를 개발하기 바랍니다. 수고하셨습니다.

# 튜토리얼로 배우는 스프링5

Spring5 & WebJars & ThymeLeaf & MyBatis

발행일 : 2018년 12월 28일

집필자 : 교사 최 선 한

교사 전 재 남

발행처 : 양영디지털고등학교