

# Spring

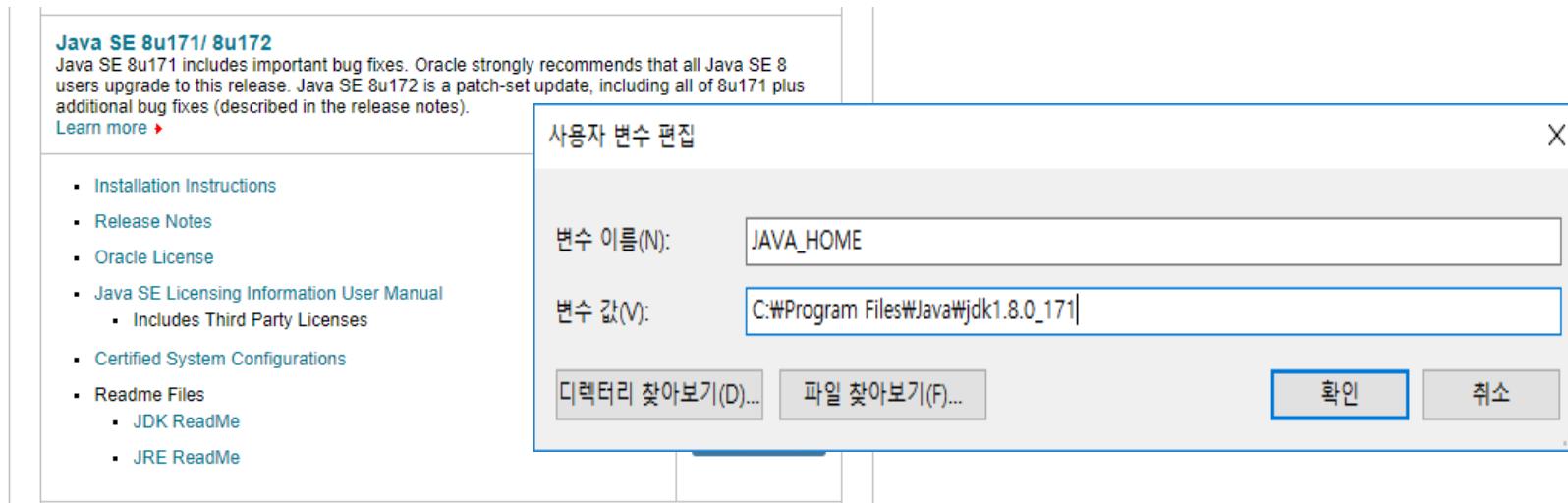
개요

# Objectvies

- 스프링의 개발 환경 (STS(혹은 Eclipse), Lombok 등)
- 오라클 데이터베이스 설치 및 계정 설정
- 스프링과 MyBatis의 연동 설정
- 스프링 MVC의 구성 설정 및 테스트

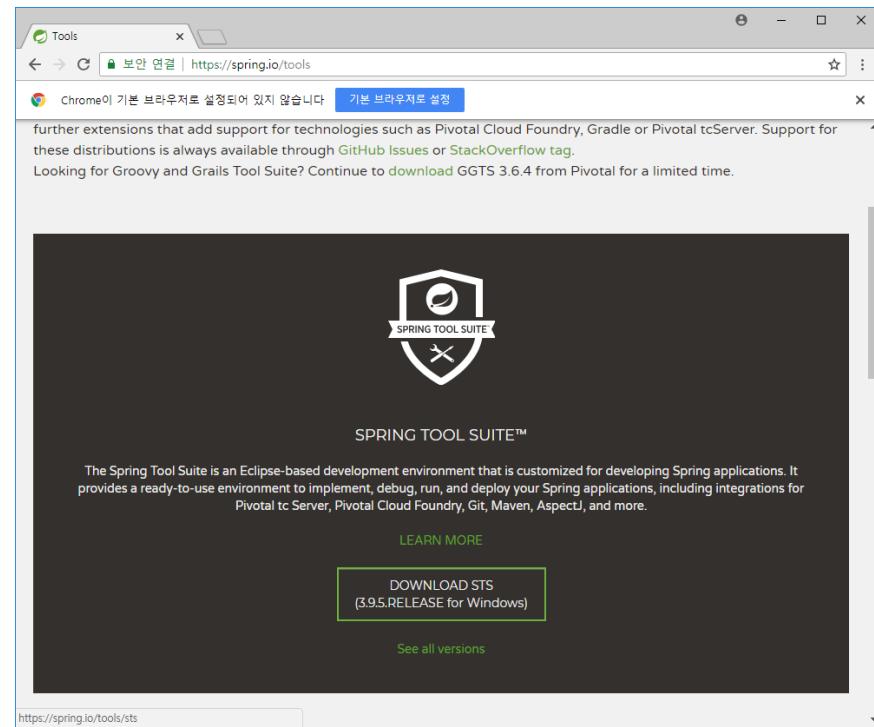
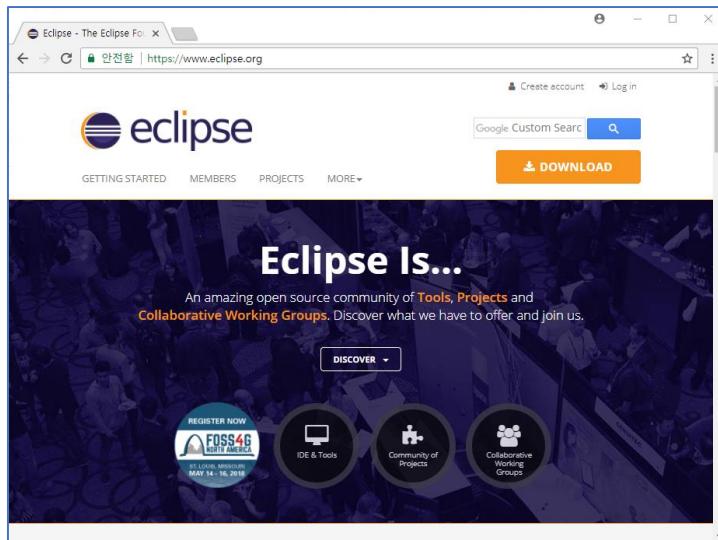
# 개발을 위한 준비 (JDK1.8)

- JDK1.8 사용
- 환경 변수 설정도 같이 진행할 것



# 개발을 위한 준비 (Eclipse or STS)

- 개발 도구 – Eclipse, STS(Spring Tool Suite), IntelliJ Ultimate 등
- Eclipse의 경우
  - STS 플러그인 추가 설치 후 사용
- STS의 경우
  - Eclipse와 별도로 다운로드 및 압축 해제



# STS의 플러그인

- <https://spring.io/tools/sts/all>
- Eclipse 메뉴에서 'Help -> Install New Software'

## Update Site Archives

You can download archived versions of the update sites, if you want to install STS into an existing Eclipse installation or if you want to update an existing STS installation without accessing the hosted version of the update repository.

ECLIPSE	ARCHIVE	SIZE
4.8.0	<a href="#">springsource-tool-suite-3.9.5.RELEASE-e4.8.0-updatesite.zip</a>	163MB
4.7.3a	<a href="#">springsource-tool-suite-3.9.5.RELEASE-e4.7.3a-updatesite.zip</a>	149MB

## Update Sites

If you want to install STS in an existing Eclipse installation, you can use one of the following update sites. Please choose the one that matches the Eclipse version you're running:

ECLIPSE	UPDATE SITES
4.8	<a href="http://download.springsource.com/release/TOOLS/update/e4.8/">http://download.springsource.com/release/TOOLS/update/e4.8/</a>
4.7	<a href="http://download.springsource.com/release/TOOLS/update/e4.7/">http://download.springsource.com/release/TOOLS/update/e4.7/</a>
4.6	<a href="http://download.springsource.com/release/TOOLS/update/e4.6/">http://download.springsource.com/release/TOOLS/update/e4.6/</a>

# Tomcat 다운로드

- JDK버전에 맞게 다운로드 및 압축해제

## Apache Tomcat Versions

Apache Tomcat® is an open source software implementation of the Java Servlet and JavaServer Pages technologies. Different versions of Apache Tomcat are available for different versions of the Servlet and JSP specifications. The mapping between [the specifications](#) and the respective Apache Tomcat versions is:

Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	JASPIC Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.10	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.32	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.53 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x	7.0.90	6 and later (7 and later for WebSocket)
2.5	2.1	2.1	N/A	N/A	6.0.x (archived)	6.0.53 (archived)	5 and later
2.4	2.0	N/A	N/A	N/A	5.5.x (archived)	5.5.36 (archived)	1.4 and later
2.3	1.2	N/A	N/A	N/A	4.1.x (archived)	4.1.40 (archived)	1.3 and later
2.2	1.1	N/A	N/A	N/A	3.3.x (archived)	3.3.2 (archived)	1.1 and later

# 스프링 프로젝트 생성

- Spring 프로젝트 중에서 'Spring Legacy Project' 생성

**Spring Legacy Project**

Click 'Next' to load the template contents.

Project name: ex00

Use default location

Location: C:\Users\cooki\wsbook\ex00

Select Spring version: Default

Templates:

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

requires downloading

Description:  
A new Spring MVC web application development project

URL: http://dist.springsource.com/release/STS/help/org-c-3.2.2.zip

Please specify the top-level package e.g. com.mycompany.myapp\*

org.zeorck.controller

# pom.xml의 수정

- 스프링 프로젝트의 버전 변경 => 5.0.7
- Java 버전 변경 및 'Maven -> Update Project'

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.5.1</version>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <compilerArgument>-Xlint:all</compilerArgument>
        <showWarnings>true</showWarnings>
        <showDeprecation>true</showDeprecation>
    </configuration>
</plugin>
```

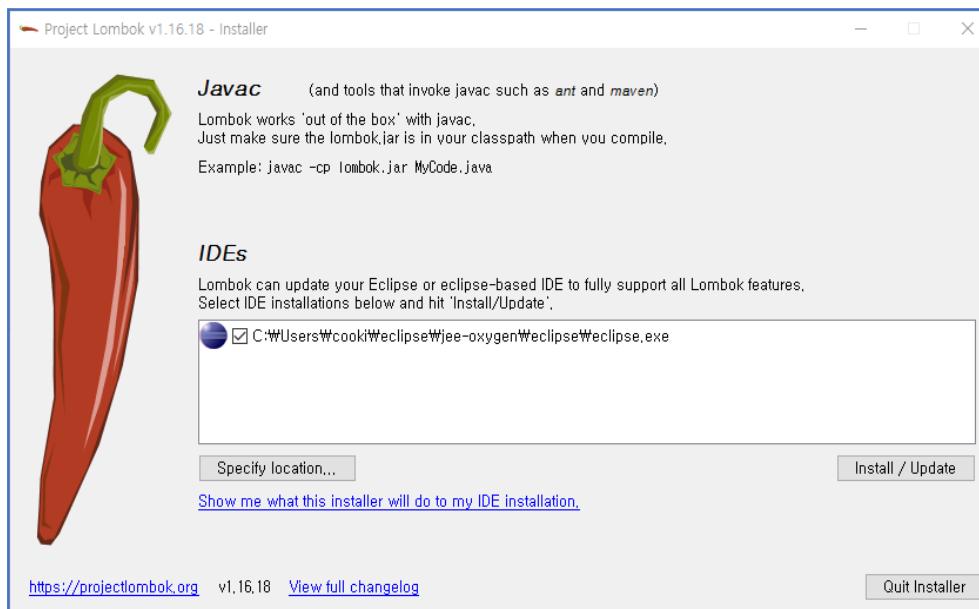
# 프로젝트의 실행 확인 및 경로 조정

- Tomcat을 이용한 프로젝트 실행 및 경로를 '/'로 조정
  - Tomcat의 'Modules'를 이용
  - 프로젝트의 'Web Settings'를 이용해서 조정

```
정보: Initializing Spring FrameworkServlet 'appServlet'
INFO : org.springframework.web.servlet.DispatcherServlet - FrameworkServlet 'appServlet': initialization started
INFO : org.springframework.web.context.support.XmlWebApplicationContext - Refreshing WebApplicationContext for namespace 'appSe
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from ServletContext resourc
INFO : org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor - JSR-330 'javax.inject.Inject' annota
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "[[],methods=[GET]]" onto p
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Looking for @ControllerAdvice: WebA
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Looking for @ControllerAdvice: WebA
INFO : org.springframework.web.servlet.handler.SimpleUrlHandlerMapping - Mapped URL path [/resources/**] onto handler 'org.spri
INFO : org.springframework.web.servlet.DispatcherServlet - FrameworkServlet 'appServlet': initialization completed in 832 ms
6월 14, 2018 11:34:34 오전 org.apache.coyote.AbstractProtocol start
정보: Starting ProtocolHandler ["http-nio-8080"]
6월 14, 2018 11:34:34 오전 org.apache.coyote.AbstractProtocol start
정보: Starting ProtocolHandler ["ajp-nio-8009"]
6월 14, 2018 11:34:34 오전 org.apache.catalina.startup.Catalina start
정보: Server startup in 6577 ms
INFO : org.zerock.controller.HomeController - Welcome home! The client locale is ko_KR.
```

# 개발을 위한 준비 (Lombok 라이브러리)

- 컴파일시에 getter/setter, 생성자, `toString()` 등을 자동으로 생성해 주기 때문에 편리함
- <https://projectlombok.org/>
- 다운로드 후에 Eclipse에 추가 설치 필요



# Java 설정을 이용하는 경우

- 기존의 XML파일들을 제거하고 pom.xml에 아래의 설정을 추가한 후에 프로젝트를 업그레이드

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.2.0</version>
  <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
  </configuration>
</plugin>
```

# Java 설정을 이용하는 경우의 설정

- root-context.xml => RootConfig클래스로 처리
- web.xml => WebConfig클래스로 처리
- @Configuration 을 이용하는 설정

# RootConfig 클래스

## RootConfig 클래스

```
package org.zerock.config;

import org.springframework.context.annotation.Configuration;

@Configuration
public class RootConfig {

}
```

# WebConfig 클래스

## WebConfig 클래스

```
package org.zerock.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebConfig extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[] {RootConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        // TODO Auto-generated method stub
        return null;
    }

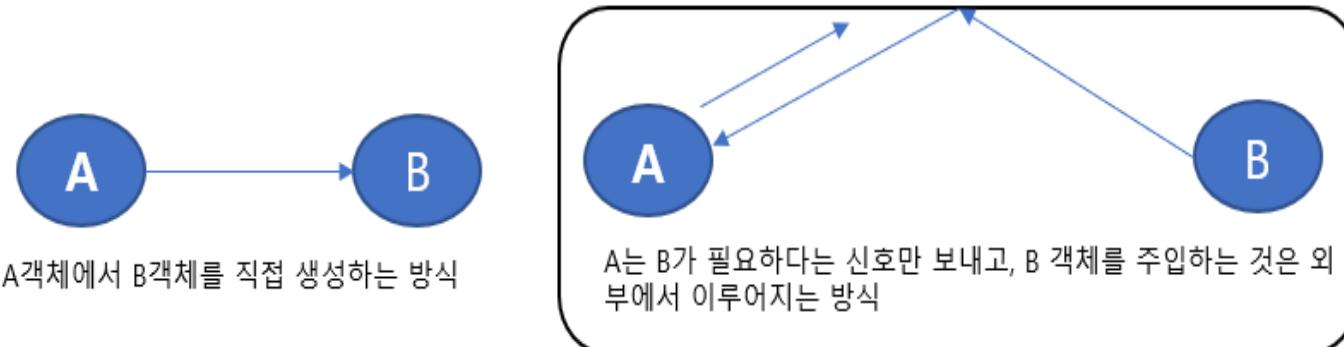
}
```

# 2장. 스프링 특징과 의존성 주입

PART 1

# 의존성 주입 (Dependency Injection)

- 마틴 파울러
  - <https://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection>
- 코드의 내부에서 객체간의 연결을 이루지 않고, 외부에서 설정을 통해서 객체간을 연결하는 패턴
- 컴파일시가 아닌 실행시에 의존 관계가 완성되는 방식
- 스프링의 경우 의존성 주입을 쉽게 적용할 수 있는 프레임워크



# Heavy / LightWeight Framework

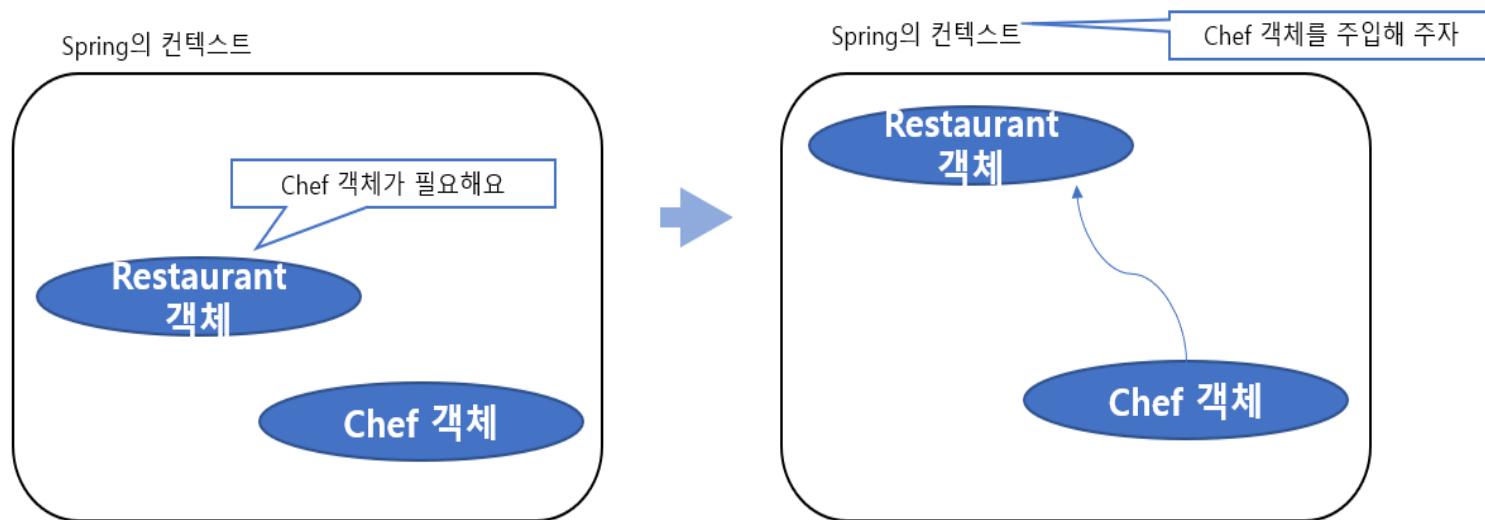
- 2000년대 초반의 분위기
  - EJB
  - 비싼 WAS
  - 많은 것의 통합
- 2000년대 중반 이후
  - 빠르고 가벼운 개발 방식
  - 작은 서비스의 군집화
- 2010년 이후
  - microservice

# AOP(Aspect-Oriented Programming)지원

- 시스템 전반에 필요한 기능들을 모듈화 시키고, 비즈니스 로직을 가지는 객체와 결합하는 방식
- crosss-concern: 횡단 관심사로 번역
  - 보안이나 로깅과 같이 시스템 여기저기서 필요한 공통적인 기능
- AOP는 횡단 관심사를 분리하고, 이를 결합하는 기능이 필요한데 스프링은 이러한 기능을 프레임워크에서 지원
- Spring AOP는 Proxy 객체를 생성

# 의존성 주입 예제

- 스프링을 이용하는 환경에서 각각의 객체를 생성하고, 이를 스프링의 설정을 통해서 연결해 보도록 한다.



### org.zerock.sample.Chef 클래스

```
package org.zerock.sample;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component
@Data
public class Chef {

}
```

### org.zerock.sample.Restaurant 클래스

```
package org.zerock.sample;

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

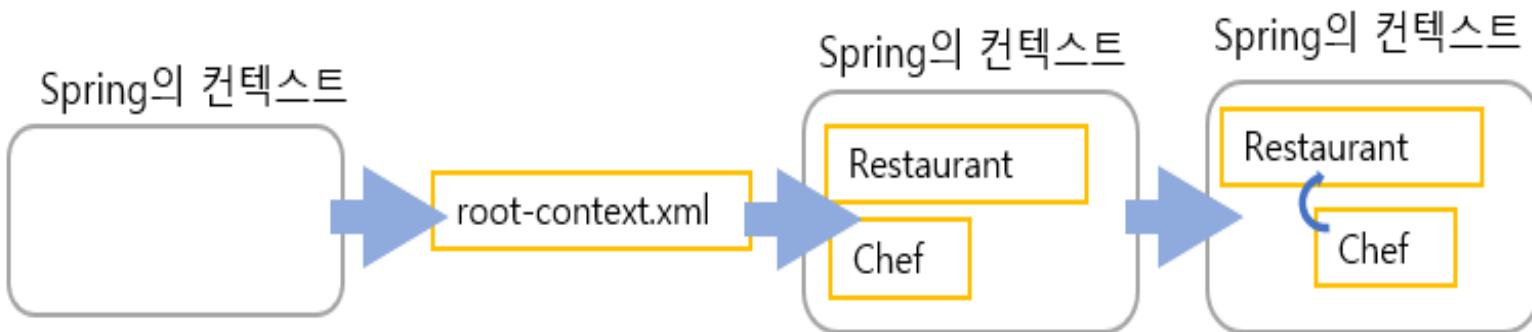
import lombok.Data;
import lombok.Setter;

@Component
@Data
public class Restaurant {

@Setter(onMethod_ = @Autowired)
private Chef chef;

}
```

# 스프링이 동작하면서 생기는 일



root-context.xml의 설정 내용이 동작하면서 필요한 인스턴스 들(bean)을 생성하고, 의존 관계를 파악해서 주입시켜 주는 방식

# 테스트 환경의 구성 및 테스트

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
@Log4j
public class SampleTests {

    @Setter(onMethod_ = { @Autowired })
    private Restaurant restaurant;

    @Test
    public void testExist() {

        assertNotNull(restaurant);

        log.info(restaurant);
        log.info("-----");
        log.info(restaurant.getChef());

    }
}
```

# 스프링 4.3이후의 자동 주입

- 기존에는 @Inject나 @Autowired, @Resource 등을 이용해서 주입에 대한 정보를 어노테이션으로 설정
- 스프링 4.3이후에는 단일 파라미터를 이용하는 생성자에 한해서 특정 타입의 객체를 자동으로 주입 가능

```
public class SampleHotel {  
    private Chef chef;  
  
    public SampleHotel(Chef chef) {  
        this.chef = chef;  
    }  
}
```

# 3장. 스프링과 Oracle Database 연동

PART 1

# Objectives

- Oracle 11g Express Edition 설치 및 설정
- JDBC 연결 확인
- Connection Pool(Hikari CP) 설정

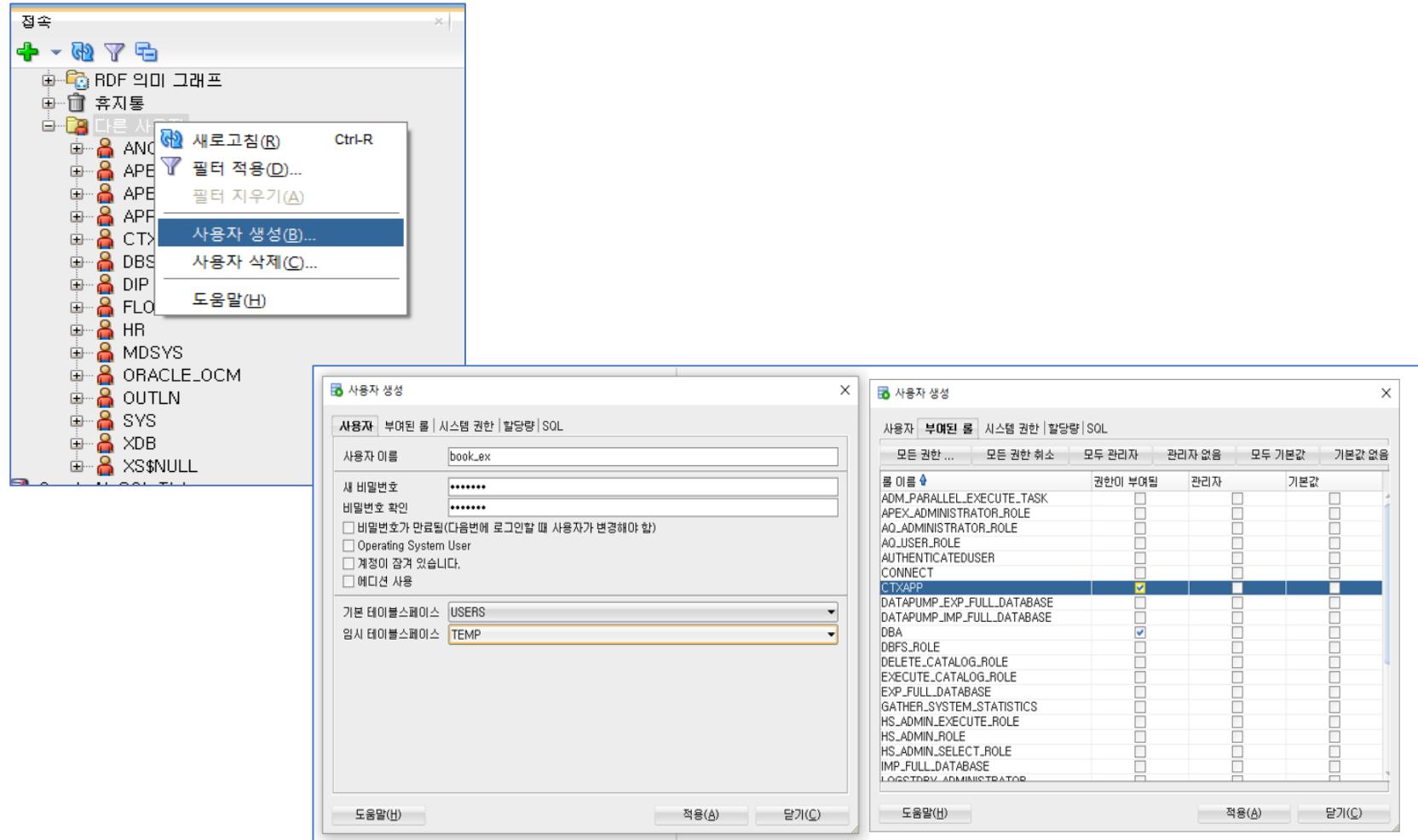
# Oracle 11g Express Edition

- <https://www.oracle.com/database/technologies/appdev/xe.html>
  - Oracle Database 11g Express Edition (Oracle Database XE) is an entry-level, small-footprint database based on the Oracle Database 11g Release 2 code base. It's free to develop, deploy, and distribute; fast to download; and simple to administer.

The image is a composite screenshot illustrating the process of downloading and installing Oracle Database 11g Express Edition. It consists of three main sections:

- Top Left:** A screenshot of the Oracle homepage. A blue arrow points from the "Database" section of the top navigation bar to the URL <http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html>.
- Middle Left:** A screenshot of the "Oracle Database 11g Express Edition" download page. It shows the release date (June 4, 2014), download links for Windows x64, Windows x32, Linux x64, and Linux x32, and a list of related downloads at the bottom.
- Bottom Right:** A screenshot of the "Preparing to Install..." step of the Oracle Database 11g Express Edition - InstallShield Wizard. It shows the progress bar for extracting the setup file.

# SQL Developer 설치 및 계정 설정



# JDBC연결 및 테스트

## JDBCTests 테스트 코드

```
package org.zerock.persistence;

import org.junit.Log4jTestRunner;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBCTests {

    static {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Test
    public void testConnection() {

        try (Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "book_ex",
                "book_ex")) {

            log.info(con);
        } catch (Exception e) {
            fail(e.getMessage());
        }
    }
}
```

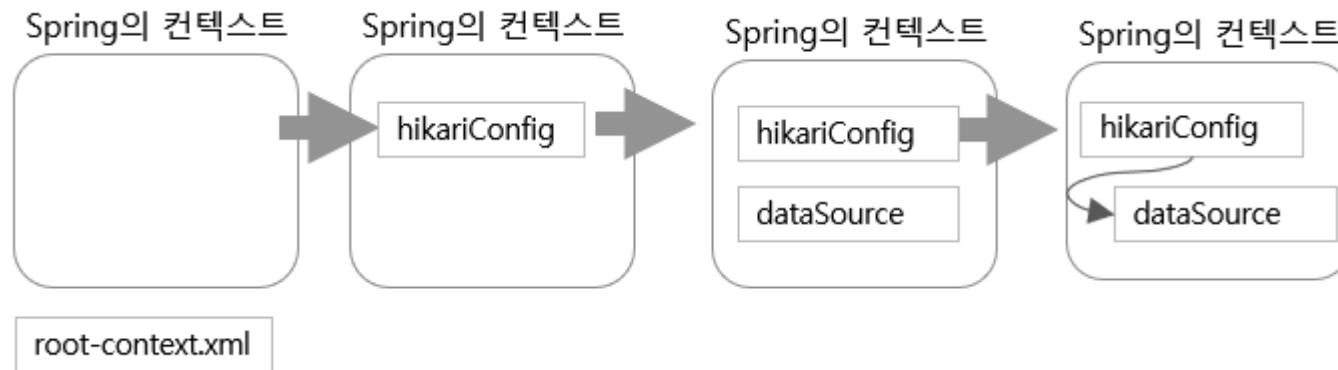
# DataSource 설정

- DB와 Connection을 맺고 끊는 작업은 리소스의 소모가 많은 작업
- Pooling이라는 기법을 통해서 객체를 미리 생성하고 빌려 쓰는 방식으로 이용해서 연결 시간을 단축
- Commons DBCP나 HikariCP등을 활용

```
<dependency>
<groupId>com.zaxxer</groupId>
<artifactId>HikariCP</artifactId>
<version>2.7.4</version>
</dependency>
```

```
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
<property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"></property>
<property name="jdbcUrl"
value="jdbc:oracle:thin:@localhost:1521:XE"></property>
<property name="username" value="book_ex"></property>
<property name="password" value="book_ex"></property>
</bean>
```

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



# 4장. MyBatis와 스프링 연동

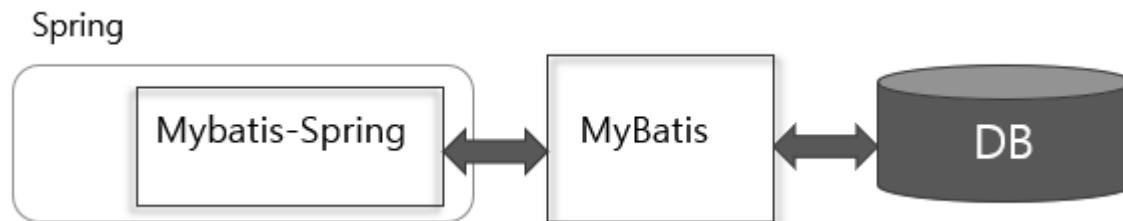
PART 1

# Objectives

- MyBatis 세팅 : ORM(Object Relation Mapping)
- spring-mybatis 라이브러리 설정
- XML Mapper 파일
- Mapper 인터페이스 설정 확인

# MyBatis

- SQL Mapping 프레임워크
  - SQL과 Object간의 관계를 매핑해주는 역할
  - JDBC코드에 비해 처리하는 부분이 간결해지고, close처리등이 지원
- Spring에서의 사용
  - 스프링은 MyBatis와의 연결을 위한 mybatis-spring 라이브러리를 이용해서 연동 처리



# mybatis-spring의 설정

- 기본적으로 dataSource의 설정이 필요
- pom.xml에 라이브러리 추가

```
<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis</artifactId>
<version>3.4.6</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis-spring</artifactId>
<version>1.3.2</version>
</dependency>
```

# SqlSessionFactory의 설정

- root-context.xml에 MyBatis설정
- MyBatis의 핵심 객체는 SqlSessionFactory타입의 객체
- SqlSessionFactoryBean은 내부적으로 MyBatis의 SqlSessionFactory를 생성

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

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

# Mapper인터페이스

- MyBatis를 사용하는 방식으로는 Mapper인터페이스를 작성하고 자동으로 생성되는 클래스를 사용하는 방식을 이용

```
TimeMapper 인터페이스
```

```
package org.zerock.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {

    @Select("SELECT sysdate FROM dual")
    public String getTime();

}
```

# Mapper인터페이스의 설정/인식

**Namespaces**

Configure Namespaces

Select XSD namespaces to use in the configuration file

- aop - http://www.springframework.org/schema/aop
- beans - http://www.springframework.org/schema/beans
- c - http://www.springframework.org/schema/cache
- cache - http://www.springframework.org/schema/cache
- context - http://www.springframework.org/schema/context
- jdbc - http://www.springframework.org/schema/jdbc
- jee - http://www.springframework.org/schema/jee
- lang - http://www.springframework.org/schema/lang
- mvc - http://www.springframework.org/schema/mvc
- mybatis-spring - http://mybatis.org/schema/mybatis-spring
- p - http://www.springframework.org/schema/p
- task - http://www.springframework.org/schema/task
- tx - http://www.springframework.org/schema/tx
- util - http://www.springframework.org/schema/util

Source Namespaces Overview beans context

**root-context.xml의 일부**

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

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

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

# Mapper인터페이스의 설정 확인

- spring-test를 이용해서 테스트 코드 작성

```
@Log4j  
public class TimeMapperTests {  
  
    @Setter(onMethod_ = @Autowired)  
    private TimeMapper timeMapper;  
  
    @Test  
    public void testgetTime() {  
        log.info(timeMapper.getClass().getName());  
        log.info(timeMapper.getTime());  
    }  
}
```

# XML Mapper + Mapper 인터페이스

## TimeMapper 인터페이스

```
package org.zerock.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {

    @Select("SELECT sysdate FROM dual")
    public String getTime();

    public String getTime2();
}
```

## TimeMapper.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="org.zerock.mapper.TimeMapper">

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

</mapper>
```

# log4jdbc-log4j2 설정

- MyBatis는 내부적으로 PreparedStatement를 이용하기 때문에 좀 더 쉽게 SQL의 로그를 보기 위한 설정

pom.xml 일부

```
<!-- https://mvnrepository.com/artifact/org.bgee.log4jdbc-log4j2/log4jdbc-log4j2-jdbc4 -->
<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4</artifactId>
    <version>1.16</version>
</dependency>
```

```
<property name="driverClassName"
    value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
<property name="jdbcUrl"
    value="jdbc:log4jdbc:oracle:thin:@localhost:1521:XE"></property>
```

# 스프링 웹 프로젝트

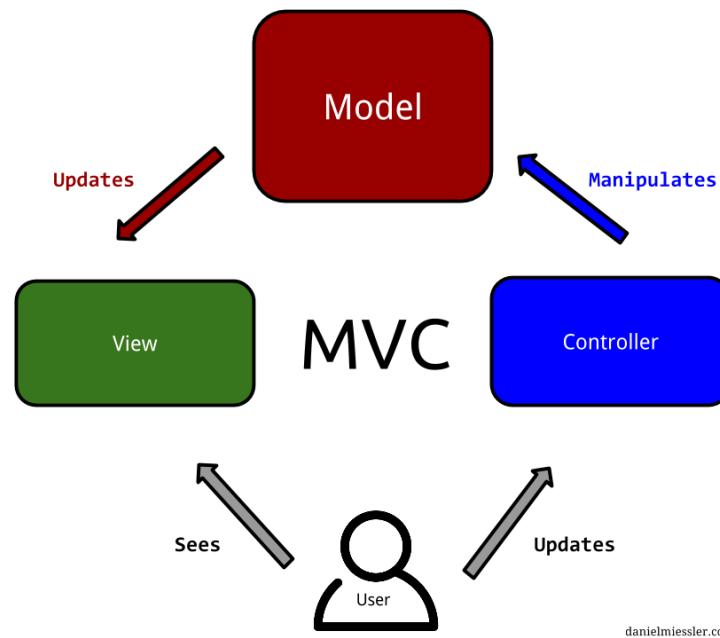
Spring MVC

# Objectives

- MVC구조의 이해
- 스프링 MVC의 다양한 예제의 학습
- 스프링 MVC를 이용하는 파일 업로드 연습
- 스프링 MVC의 예외처리

# MVC(Model-View-Controller)

- 대부분의 서블릿 기반 프레임워크들이 사용하는 방식
- 데이터와 처리, 화면을 분리하는 방식
- 웹에서는 Model 2 방식으로 표현

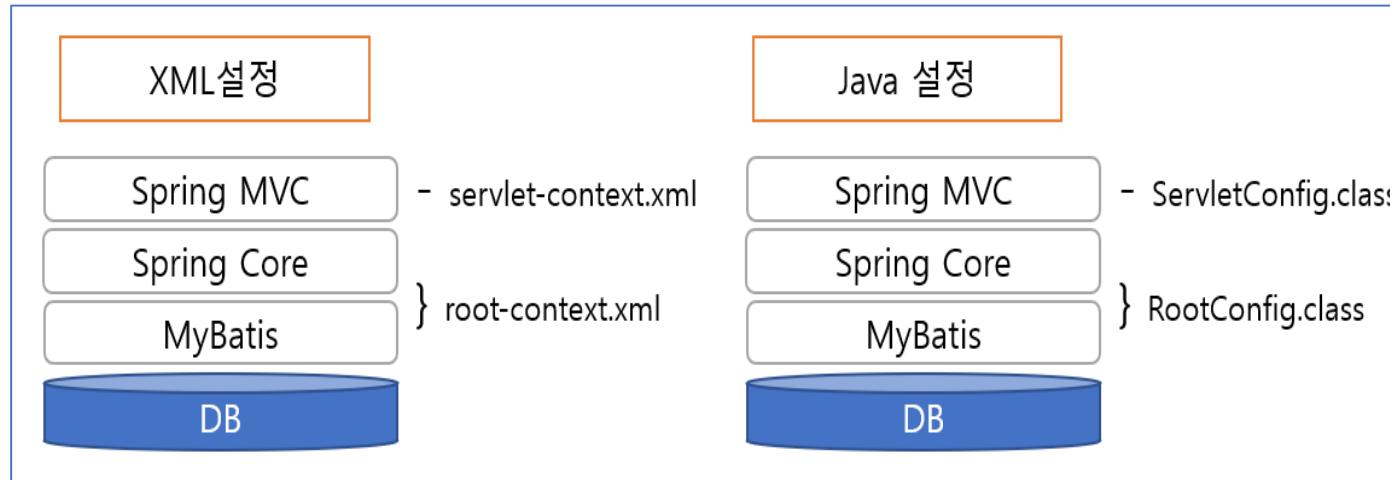


# 스프링과 스프링 MVC

- 스프링 프레임워크 Core + 여러 Sub 프로젝트들
- <https://spring.io/projects>
- 별도로 결합해서 사용하기 때문에 설정 역시 별도로 처리 가능



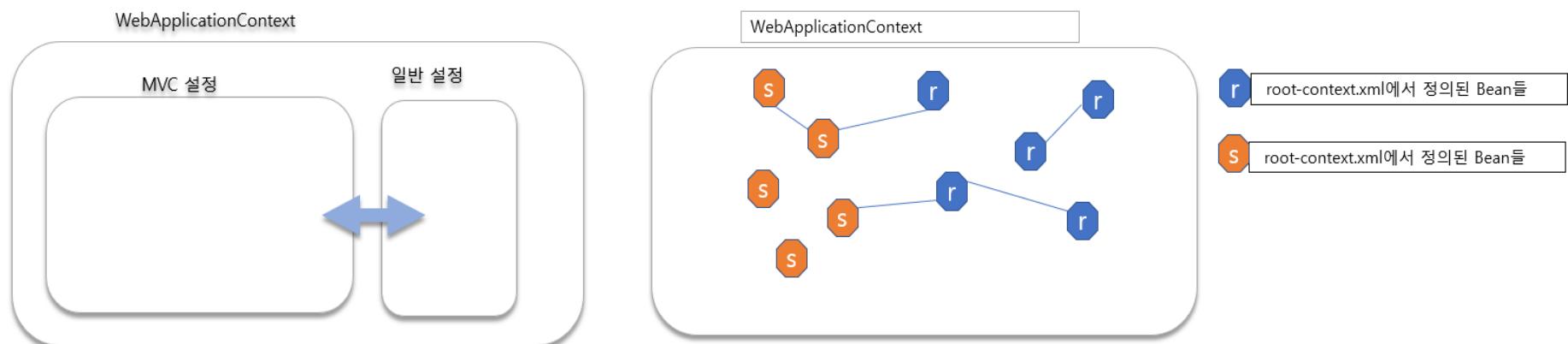
# 일반적인 스프링 + 스프링 MVC



\* XML이나 Java 설정 이용시에 설정 분리

# 웹 프로젝트의 구조

- 스프링을 실행하는 존재
  - ApplicationContext => WebApplicationContext
  - 같이 연동되는 방식으로 동작하기 때문에 설정을 분리해도 통합해서 사용 가능



# 프로젝트 설정후 변경

- Servlet은 3.0이상으로 변경

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

# 프로젝트의 경로(path)

- Tomcat을 이용하는 경로 변경
- Web Project Setting을 이용하는 변경

Path	Document Base
/controller	ex01

Properties for ex01

type filter text

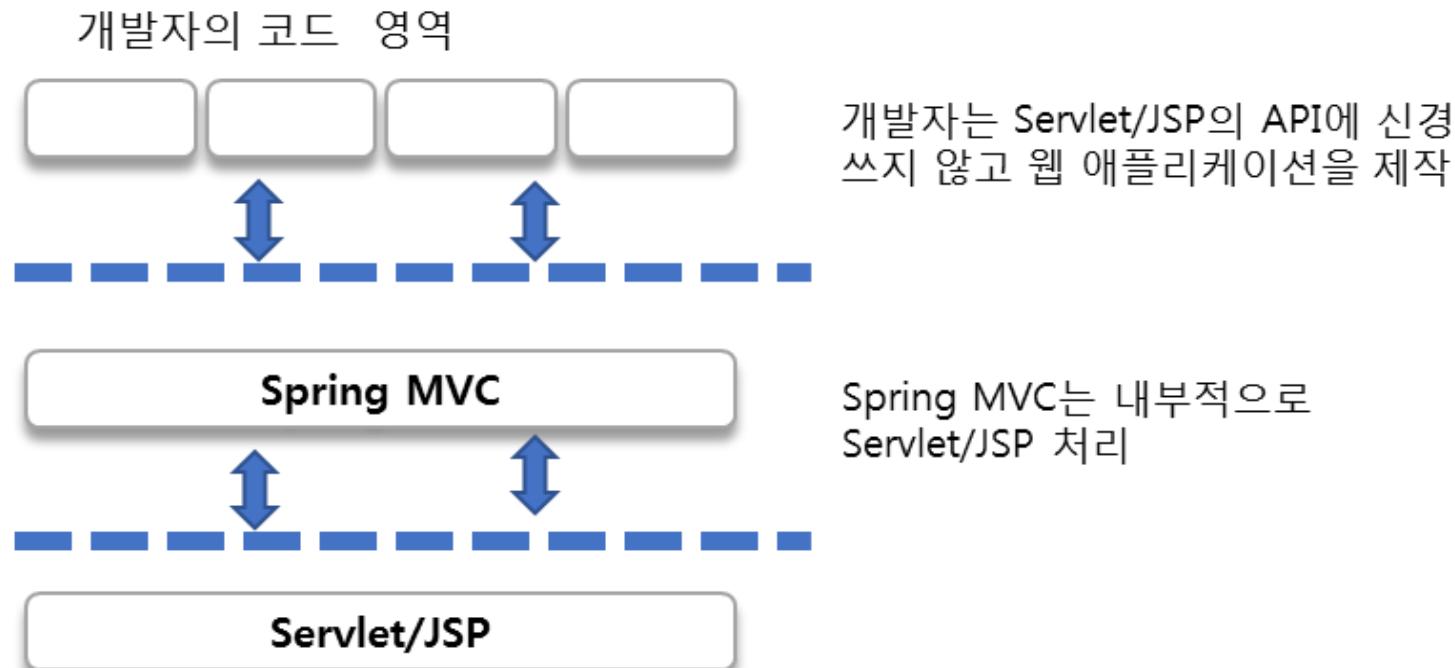
Java Build Path  
Java Code Style  
Java Compiler  
Java Editor  
Javadoc Location  
JavaScript  
JSP Fragment  
Maven  
Project Facets  
Project Natures  
Project References  
Run/Debug Settings  
Server  
Service Policies  
Spring  
Targeted Runtimes  
Task Repository  
Task Tags  
Validation  
Web Content Settings  
Web Page Editor  
Web Project Settings

Web Project Settings

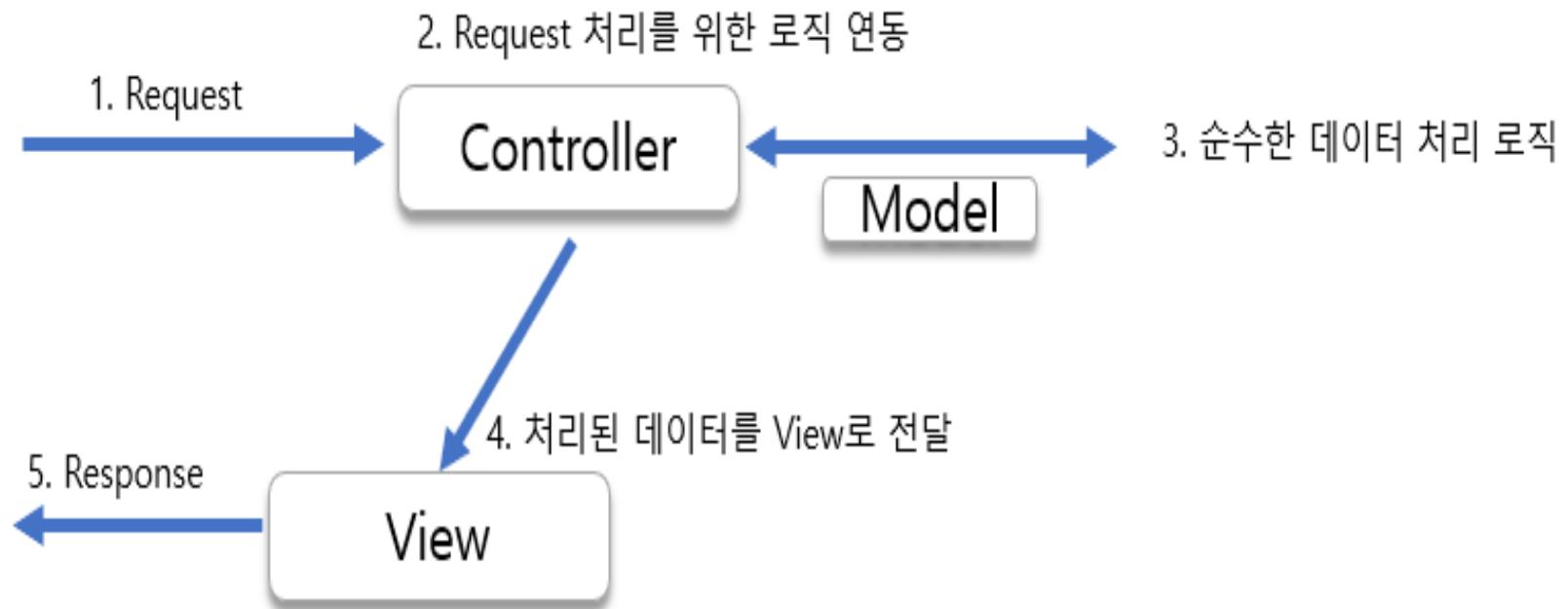
Context root: /

# 스프링 MVC의 기본 사상

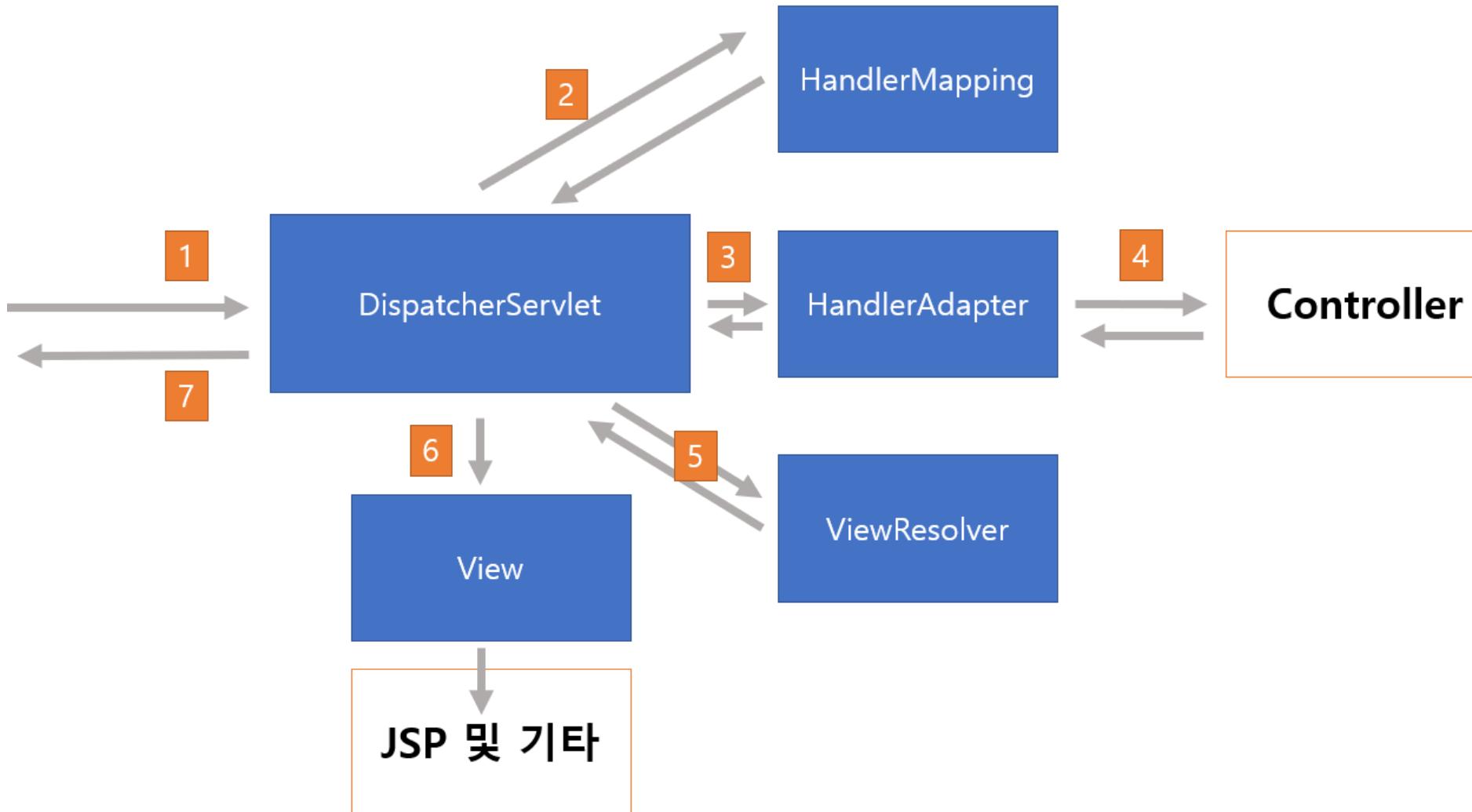
- 서블릿 기반이긴 하지만 한 단계 더 추상화된 수준의 개발 지향
- 서블릿 API 없이도 개발이 가능한 수준



# 모델2 방식과 스프링 MVC



# 스프링 MVC의 기본 흐름



# Controller

- HttpServletRequest, HttpServletResponse를 거의 사용할 필요 없이 필요 한 기능 구현
- 다양한 타입의 파라미터 처리, 다양한 타입의 리턴 타입 사용 가능
- GET 방식, POST 방식 등 전송 방식에 대한 처리를 어노테이션으로 처리 가능
- 상속/인터페이스 방식 대신에 어노테이션만으로도 필요한 설정 가능

# @Controller, @RequestMapping

- `@Controller` – 해당 클래스의 인스턴스를 스프링의 빈으로 등록하고 컨트롤러로 사용
  - `<component-scan>`과 같이 활용
- `@RequestMapping` – 특정한 URI에 대한 처리를 해당 컨트롤러나 메서드에서 처리

## SampleController 클래스

```
package org.zerock.controller;

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

@Controller
@RequestMapping("/sample/*")
public class SampleController {

}
```

# @RequestMapping의 변화

- 스프링 4.3 전까지는 @RequestMapping( method ='POST') 방식으로 사용
- 스프링 4.3이후에는 @GetMapping, @PostMapping등으로 간단히 표현 가능

SampleController 클래스의 일부

```
@RequestMapping(value = "/basic", method = {RequestMethod.GET, RequestMethod.POST})
public void basicGet() {

    log.info("basic get.....");
}

@GetMapping("/basicOnlyGet")
public void basicGet2() {

    log.info("basic get only get.....");
}
```

# 컨트롤러의 파라미터 수집

- 스프링 MVC의 컨트롤러는 메서드의 파라미터를 자동으로 수집, 변환하는 편리한 기능을 제공
- Java Beans 규칙에 맞게 작성되어야 한다.
  - 생성자가 없거나 빈 생성자
  - 올바른 규칙으로 만들어진 Getter/Setter

```
@GetMapping("/ex01")
public String ex01(SampleDTO dto) {
    log.info("+" + dto);
    return "ex01";
}
```

```
@Data
public class SampleDTO {

    private String name;
    private int age;
}
```

# 리스트/배열

```
@GetMapping("/ex02List")
public String ex02List(@RequestParam("ids")ArrayList<String> ids) {

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

    return "ex02List";
}
```

# @InitBinder

```
public class TodoDTO {  
  
    private String title;  
    private Date dueDate; //날짜 타입의 변환 필요  
}
```

```
@InitBinder  
public void initBinder(WebDataBinder binder) {  
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  
    binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, false));  
}  
  
...생략...  
  
@GetMapping("/ex03")  
public String ex03(TodoDTO todo) {  
    log.info("todo: " + todo);  
    return "ex02";  
}
```

# @DateTimeFormat

- @InitBinder 외에도 날짜에 대한 처리가 쉽게 추가된 어노테이션

```
package org.zerock.domain;

import java.util.Date;

import org.springframework.format.annotation.DateTimeFormat;

import lombok.Data;

@Data
public class TodoDTO {

    private String title;

    @DateTimeFormat(pattern = "yyyy/MM/dd")
    private Date dueDate;
}
```

# Model이라는 데이터전달자

- Model 객체는 JSP에 컨트롤러에서 생성된 데이터를 담아서 전달하는 역할을 하는 존재
- 모델 2 방식에서 사용하는 `request.setAttribute()`와 유사한 역할

```
public String home(Model model) {  
  
    model.addAttribute("serverTime", new java.util.Date());  
  
    return "home";  
}
```

# @ModelAttribute

- 컨트롤러에서 메서드의 파라미터는 기본자료형을 제외한 객체형 타입은 다시 화면으로 전달
- @ModelAttribute는 명시적으로 화면에 전달되도록 지정

```
@GetMapping("/ex04")
public String ex04(SampleDTO dto, @ModelAttribute("page") int page) {

    log.info("dto: " + dto);
    log.info("page: " + page);

    return "/sample/ex04";
}
```

```
<h2>SAMPLEDTO ${sampleDTO }</h2>
<h2>PAGE ${page }</h2>
```



# RedirectAttribute

- 화면에 한번만 전달되는 파라미터를 처리하는 용도
- 내부적으로 HttpSession 객체에 담아서 한번만 사용되고, 폐기

```
rttr.addFlashAttribute("name", "AAA");
rttr.addFlashAttribute("age", 10);

return "redirect:/";
```

```
response.sendRedirect("/home?name=aaa
&age=10");
```

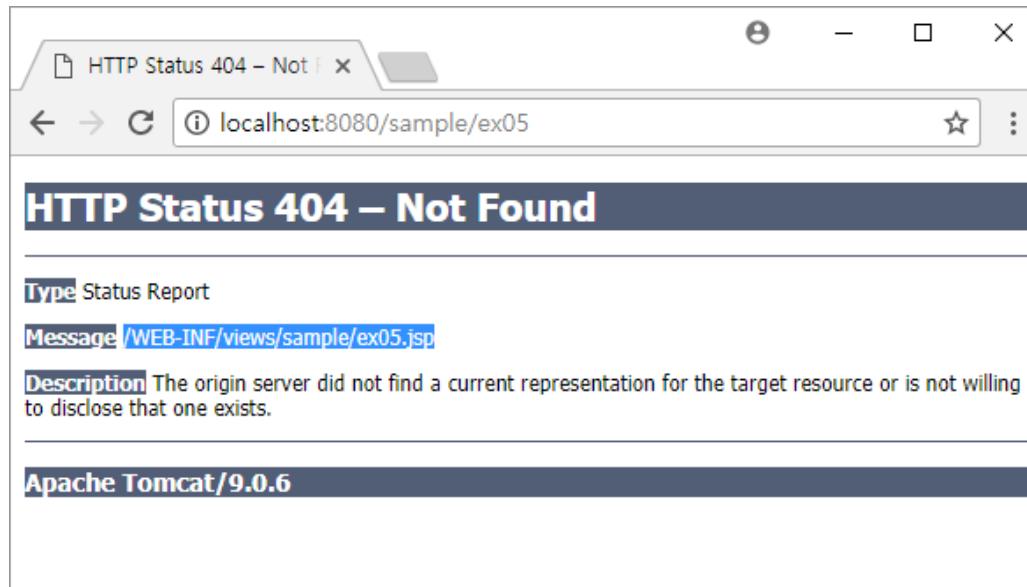
# Controller의 리턴타입

- String: jsp를 이용하는 경우에는 jsp 파일의 경로와 파일이름을 나타내기 위해서 사용
- void: 호출하는 URL과 동일한 이름의 jsp를 의미
- VO, DTO 타입: 주로 JSON 타입의 데이터를 만들어서 반환하는 용도로 사용 (추가적인 라이브러리 필요)
- ResponseEntity 타입: response할 때 Http 헤더 정보와 내용을 가공하는 용도로 사용 (추가적인 라이브러리 필요).
- Model, ModelAndView: Model로 데이터를 반환하거나 화면까지 같이 지정하는 경우에 사용 (최근에는 많이 사용하지 않습니다.).
- HttpHeaders: 응답에 내용 없이 Http 헤더 메시지만 전달하는 용도로 사용

# void타입

- 호출하는 URL과 동일한 이름의 jsp파일

```
@GetMapping("/ex05")
public void ex05() {
    log.info("/ex05.....");
}
```



# String 타입

- 상황에 따라 다른 화면을 보여줄 필요가 있을 경우에 유용하게 사용
- String 타입에는 다음과 같은 특별한 키워드를 붙여서 사용할 수 있음
  - redirect: 리다이렉트 방식으로 처리하는 경우
  - forward: 포워드 방식으로 처리하는 경우

# 객체 타입

- XML이나 JSON으로 처리
- @RepsoneBody 어노테이션과 같이 사용

```
<!--  
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackso  
n-databind -->  
<dependency>  
    <groupId>com.fasterxml.jackson.core</groupId>  
    <artifactId>jackson-databind</artifactId>  
    <version>2.9.4</version>  
</dependency>
```

```
@GetMapping("/ex06")  
public @ResponseBody SampleDTO ex06() {  
    log.info("/ex06.....");  
  
    SampleDTO dto = new SampleDTO();  
    dto.setAge(10);  
    dto.setName("홍길동");  
  
    return dto;  
}
```



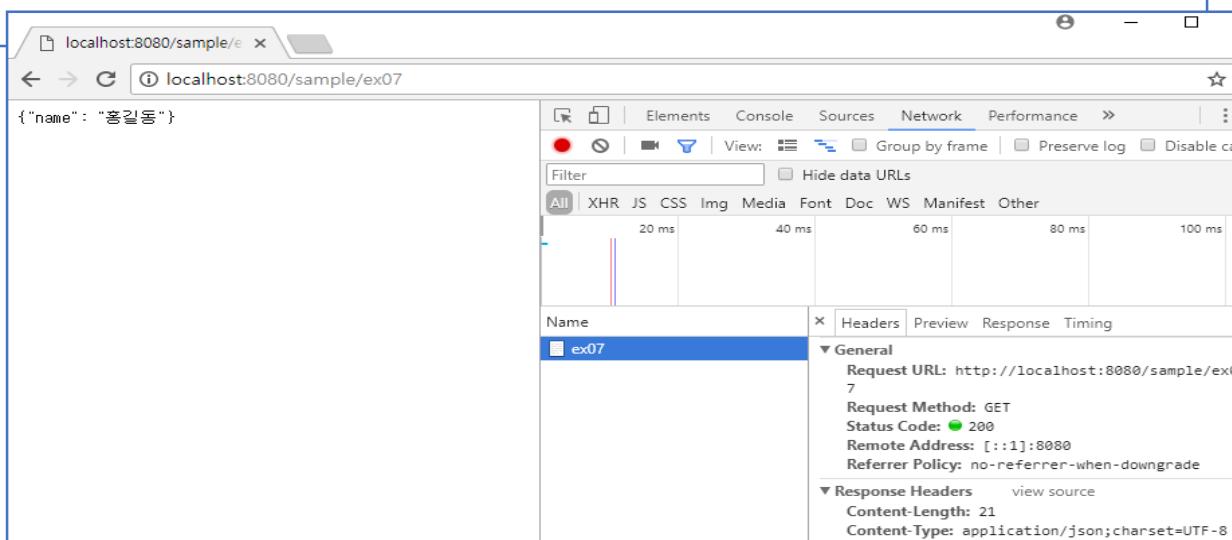
# ResponseEntity

- HTTP헤더 정보와 추가적인 데이터를 전달할 때 사용

```
@GetMapping("/ex07")
public ResponseEntity<String> ex07() {
    log.info("/ex07.....");
    // {"name": "홍길동"}
    String msg = "{\"name\": \"홍길동\"}";

    HttpHeaders header = new HttpHeaders();
    header.add("Content-Type", "application/json;charset=UTF-8");

    return new ResponseEntity<>(msg, header, HttpStatus.OK);
}
```



# 파일업로드 처리

- Servlet 3.0이후(Tomcat 7.0)에는 기본적으로 업로드 되는 파일을 처리할 수 있는 기능이 추가
- 별도로 commons-fileupload 라이브러리 등을 사용

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.3</version>
</dependency>
```

# 파일 업로드를 위한 servlet-context.xml

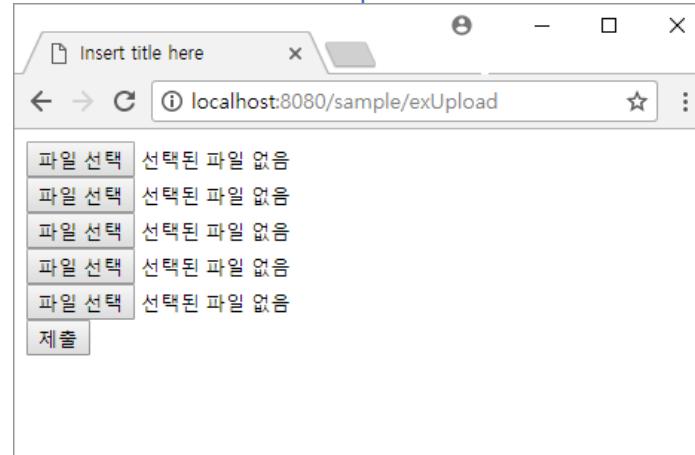
- multipartResolver라는 이름으로 스프링 빈 설정

```
<beans:bean id= "multipartResolver"
class= "org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <beans:property name= "defaultEncoding" value= "utf-8"></beans:property>
    <!-- 1024 * 1024 * 10 bytes 10MB -->
    <beans:property name= "maxUploadSize" value= "104857560"> </beans:property>
    <!-- 1024 * 1024 * 2 bytes 2MB -->
    <beans:property name= "maxUploadSizePerFile" value= "2097152"> </beans:property>
    <beans:property name= "uploadTempDir" value = "file:/C:/upload/tmp "> </beans:property>
    <beans:property name= "maxInMemorySize" value= "10485756"> </beans:property>
</beans:bean>
```

# 파일업로드를 위한 HTML

- <form> 태그내 enctype='multipart'

```
<form action="/sample/exUploadPost" method="post"  
enctype="multiPart/form-data">  
  
<div>  
  <input type='file' name='files'>  
</div>  
<div>  
  <input type='submit'>  
</div>  
</form>
```

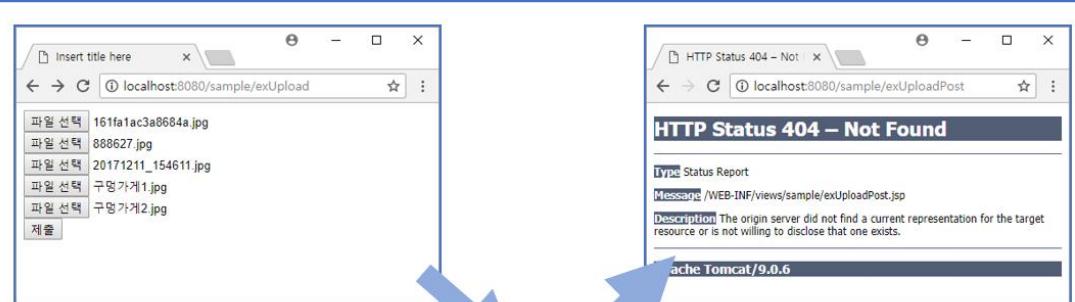


# 업로드되는 파일의 처리

- MultipartFile을 이용해서 처리

```
@PostMapping("/exUploadPost")
public void exUploadPost(ArrayList<MultipartFile> files) {

    files.forEach(file -> {
        log.info("-----");
        log.info("name:" + file.getOriginalFilename());
        log.info("size:" + file.getSize());
    });
}
```



```
INFO : org.zerock.controller.SampleController - name:161fa1ac3a8684a.jpg
INFO : org.zerock.controller.SampleController - size:1018843
INFO : org.zerock.controller.SampleController - name:888627.jpg
INFO : org.zerock.controller.SampleController - size:232402
INFO : org.zerock.controller.SampleController - name:20171211_154611.jpg
INFO : org.zerock.controller.SampleController - size:413005
INFO : org.zerock.controller.SampleController - name:구영가게1.jpg
INFO : org.zerock.controller.SampleController - size:121493
INFO : org.zerock.controller.SampleController - name:구영가게2.jpg
INFO : org.zerock.controller.SampleController - size:304061
```

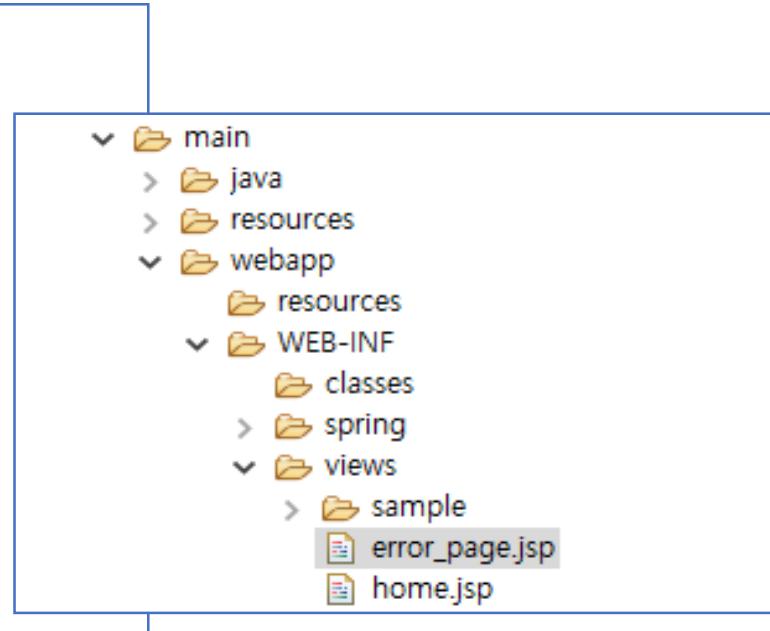
# 컨트롤러의 예외(Exception)처리

- @ExceptionHandler와 @ControllerAdvice를 이용한 처리
- @ResponseBody를 이용하는 예외 메시지 구성

# @ControllerAdvice: AOP 공통관심사

- 예외처리와 원래의 컨트롤러가 혼합된 형태의 클래스가 작성되는 방식
- @ExceptionHandler는 해당 메서드가 () 들어가는 예외 타입을 처리한다는 것을 의미

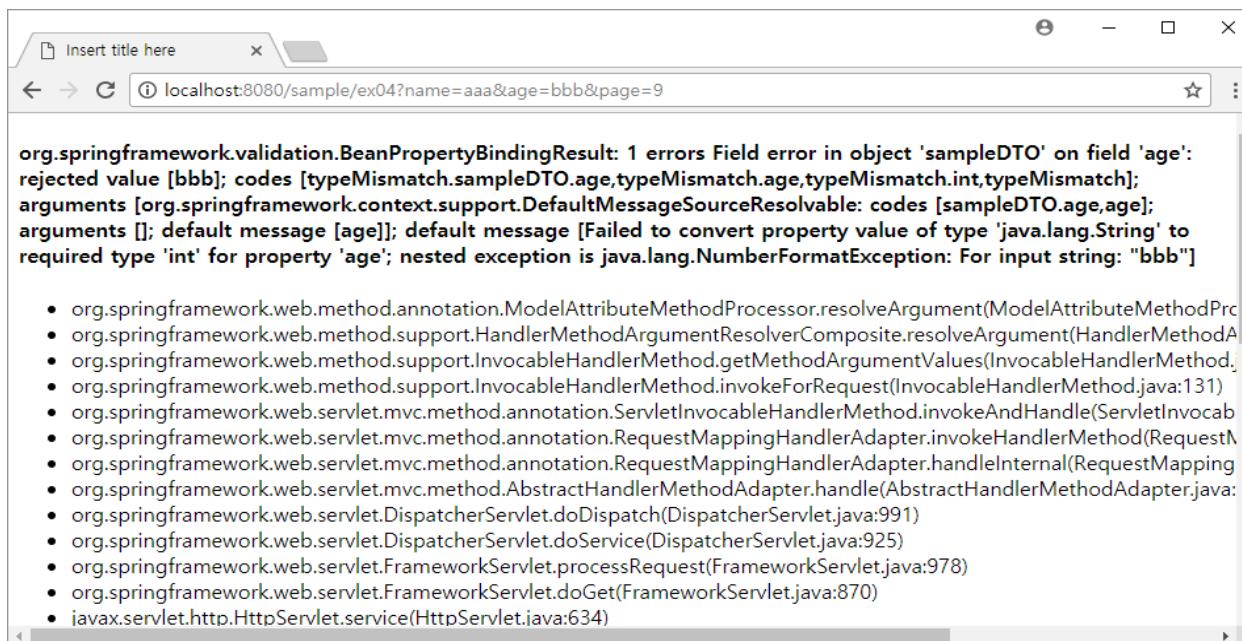
```
@ControllerAdvice  
@Log4j  
public class CommonExceptionAdvice {  
  
    @ExceptionHandler(Exception.class)  
    public String except(Exception ex, Model model) {  
  
        log.error("Exception ..... " + ex.getMessage());  
        model.addAttribute("exception", ex);  
        log.error(model);  
        return "error_page";  
    }  
}
```



# error\_page.jsp의 내용

```
<h4><c:out value="${exception.getMessage()}"></c:out></h4>

<ul>
<c:forEach items="${exception.getStackTrace() }" var="stack">
<li><c:out value="${stack}"></c:out></li>
</c:forEach>
</ul>
```



# 스프링 웹 프로젝트

게시판 프로젝트

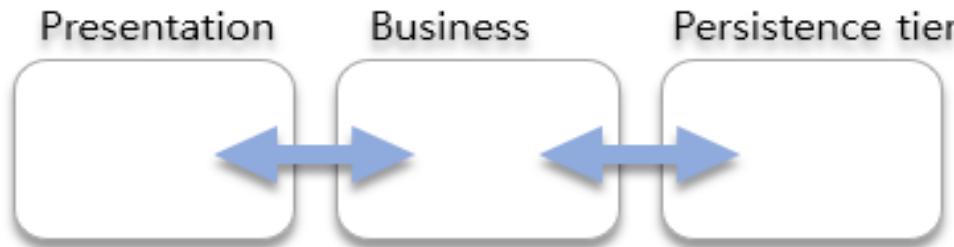
# Objectives

- 스프링 MVC를 이용하는 웹 프로젝트 전체 구조에 대한 이해
- 개발의 각 단계에 필요한 설정 및 테스트 환경
- 기본적인 등록, 수정, 삭제, 조회, 리스트 구현
- 목록(리스트) 화면의 페이징(paging) 처리
- 검색 처리와 페이지 이동

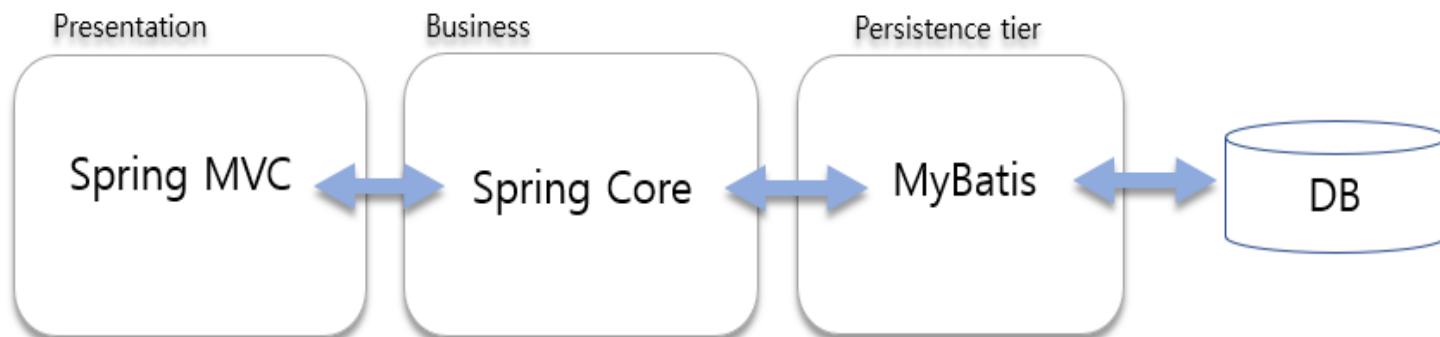
## 7. 프로젝트의 구성

# 프로젝트의 구성

- 일반적인 웹 프로젝트의 구조는 3-Tier의 구조를 활용



- 스프링 MVC를 이용하는 예제의 구성



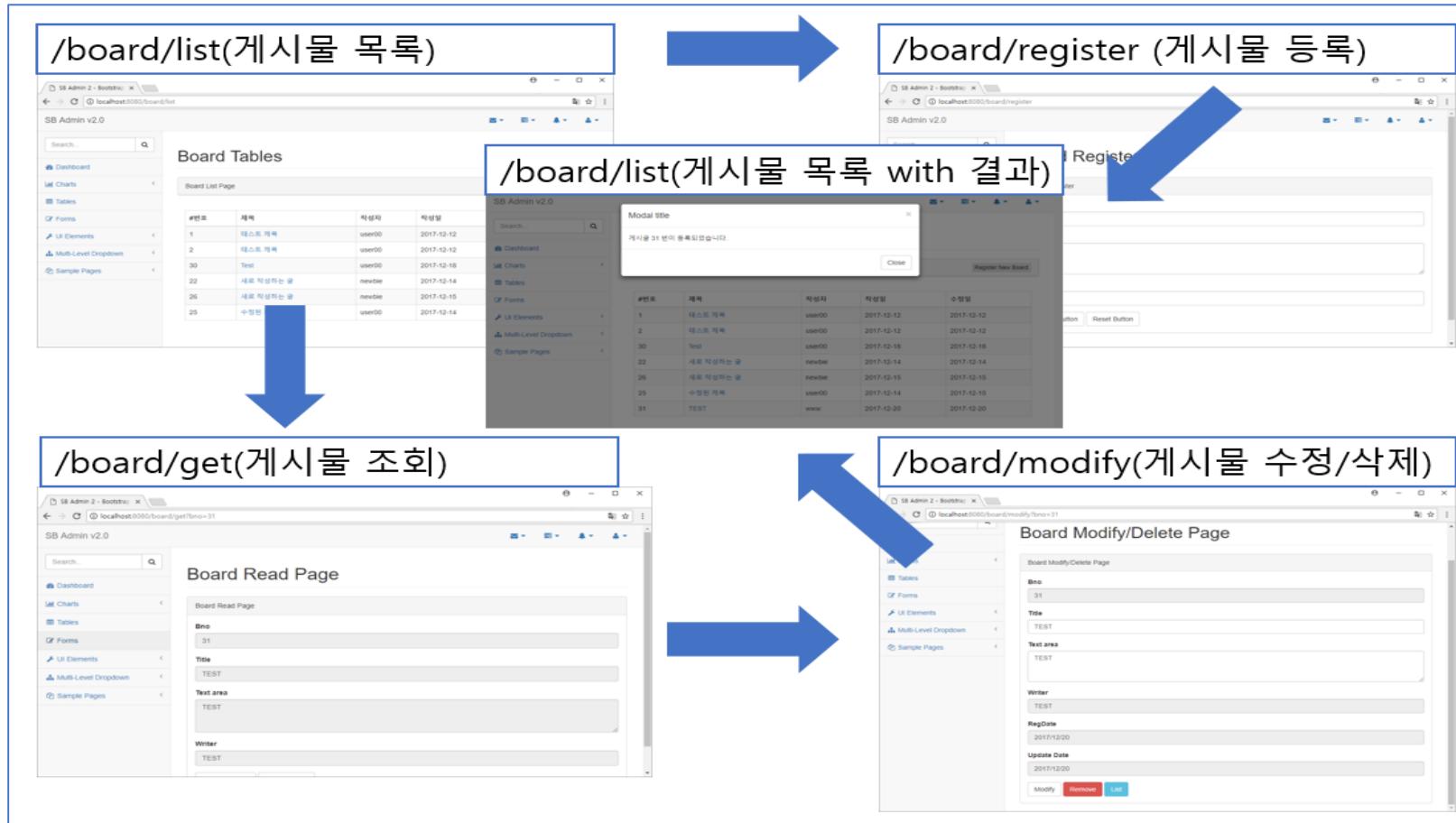
# 각 영역의 네이밍 규칙

- xxxController: 스프링 MVC에서 동작하는 Controller 클래스
- xxxService, xxxServiceImpl: 비즈니스 영역을 담당하는 인터페이스는 'xxxService'라는 방식을 사용하고, 인터페이스를 구현한 클래스는 'xxxServiceImpl'이라는 이름을 사용
- xxxDAO, xxxRepository: DAO(Data-Access-Object)나 Repository(저장소)라는 이름으로 영역을 따로 구성하는 것이 보편적. 예제에서는 별도의 DAO를 구성하는 대신에 MyBatis의 Mapper 인터페이스를 활용.
- VO, DTO: VO의 경우는 주로 Read Only의 목적이 강하고, 데이터 자체도 Immutable(불변)하게 설계. DTO는 주로 데이터 수집의 용도

# 프로젝트 패키지의 구성

org.zerock.config	프로젝트와 관련된 설정 클래스들의 보관 패키지
org.zerock.controller	스프링 MVC의 Controller들의 보관 패키지
org.zerock.service	스프링의 Service 인터페이스와 구현 클래스 패키지
org.zerock.domain	VO, DTO 클래스들의 패키지
org.zerock	MyBatis Mapper 인터페이스 패키지
org.zerock.exception	웹 관련 예외처리 패키지
org.zerock.aop	스프링의 AOP 관련 패키지
org.zerock.security	스프링 Security 관련 패키지
org.zerock.util	각종 유틸리티 클래스 관련 패키지

# 기본적인 게시물의 CRUD 흐름



# 프로젝트의 생성 및 준비

- Spring Legacy Project의 생성
- pom.xml에서 스프링 버전 변경
- spring-test,spring-jdbc,spring-tx 추가
- junit버전 변경
- Servlet 버전 변경
- HikariCP, MyBatis, mybatis-spring, Log4jdbc 추가
- JDBC드라이버 프로젝트내 추가
- 기타 Lombok의 설정 등

# 데이터베이스내 테이블 생성

```
create sequence seq_board;
```

일련 번호를 위한 sequence 생성

```
create table tbl_board (
    bno number(10,0),
    title varchar2(200) not null,
    content varchar2(2000) not null,
    writer varchar2(50) not null,
    regdate date default sysdate,
    updatedate date default sysdate
);
```

게시물 저장을 위한 테이블 생성

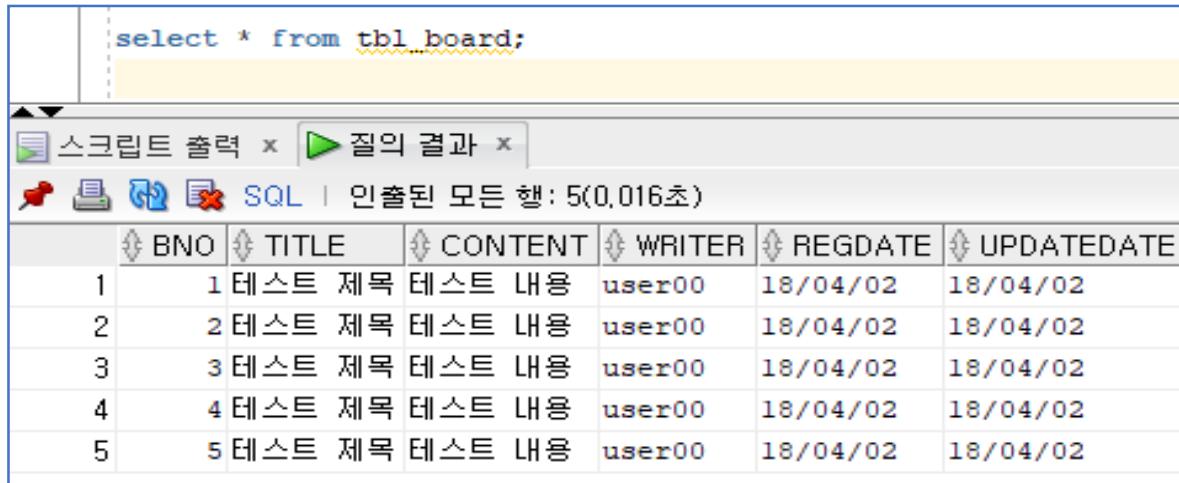
```
alter table tbl_board add constraint
pk_board
primary key (bno);
```

게시물의 PK 지정

# Dummy(더미) 데이터의 추가

```
insert into tbl_board (bno, title, content, writer)
values (seq_board.nextval, '제목 테스트 내용 테스트', 'user00');
```

반복적인 실행으로 여러 개의 데이터 생성 및 확인 및 commit



The screenshot shows the Oracle SQL Developer interface. In the top-left query editor, the following SQL statement is written:

```
select * from tbl_board;
```

In the bottom-right results tab, the output of the query is displayed in a grid format. The results show five rows of data, each with a unique BNO and a title/content pair that includes the word '테스트' (Test). The writer is consistently 'user00' and the dates are all '18/04/02'. The tabs at the top of the results window indicate the current view is '질의 결과' (Query Results).

BNO	TITLE	CONTENT	WRITER	REGDATE	UPDATEDATE
1	1 테스트 제목 테스트 내용	user00	18/04/02	18/04/02	
2	2 테스트 제목 테스트 내용	user00	18/04/02	18/04/02	
3	3 테스트 제목 테스트 내용	user00	18/04/02	18/04/02	
4	4 테스트 제목 테스트 내용	user00	18/04/02	18/04/02	
5	5 테스트 제목 테스트 내용	user00	18/04/02	18/04/02	

# 데이터베이스 설정 및 테스트

- root-context.xml
  - DataSource의 설정
  - SqlSessionFactory 설정

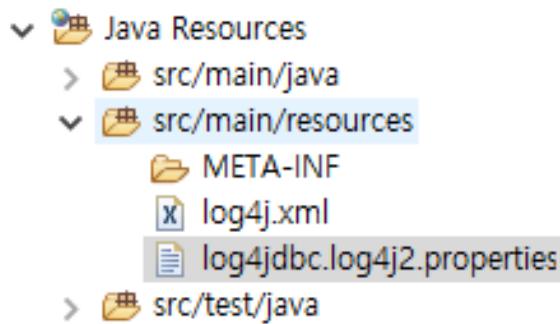
```
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">

    <property name="driverClassName"
    value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
    <property name="jdbcUrl"
    value="jdbc:log4jdbc:oracle:thin:@localhost:1521:XE"></property>
    <property name="username" value="book_ex"></property>
    <property name="password" value="book_ex"></property>

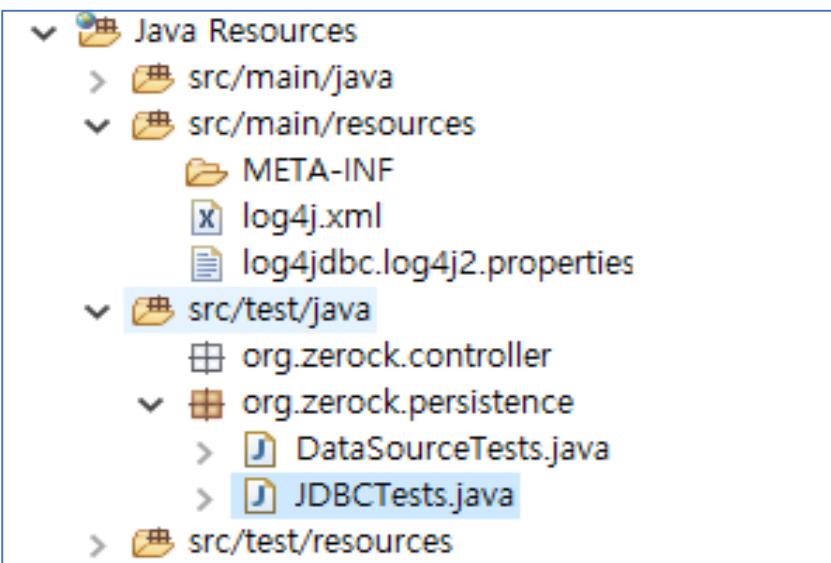
</bean>

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

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



## JDBC연결과 DataSource에 대한 테스트 실행



# 8. 영속 계층 구현

# 영속 계층의 처리

- 테이블을 반영하는 VO(Value Object) 클래스의 생성
- MyBatis의 Mapper 인터페이스의 작성/XML 처리
- 작성한 Mapper 인터페이스의 테스트

# BoardVO 클래스

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID
1 BNO	NUMBER(10, 0)	No	(null)	1
2 TITLE	VARCHAR2(200 BYTE)	No	(null)	2
3 CONTENT	VARCHAR2(2000 BYTE)	No	(null)	3
4 WRITER	VARCHAR2(50 BYTE)	No	(null)	4
5 REGDATE	DATE	Yes	sysdate	5
6 UPDATEDATE	DATE	Yes	sysdate	6

```
package org.zerock.domain;

import java.util.Date;

import lombok.Data;

@Data
public class BoardVO {

    private Long bno;
    private String title;
    private String content;
    private String writer;
    private Date regdate;
    private Date updateDate;
}
```

# Mapper인터페이스

```
package org.zerock.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Select;
import org.zerock.domain.BoardVO;

public interface BoardMapper {

    @Select("select * from tbl_board where bno > 0")
    public List<BoardVO> getList();

}
```

# BoardMapper의 테스트

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
@Log4j
public class BoardMapperTests {

    @Setter(onMethod = @_({ @Autowired }))
    private BoardMapper mapper;

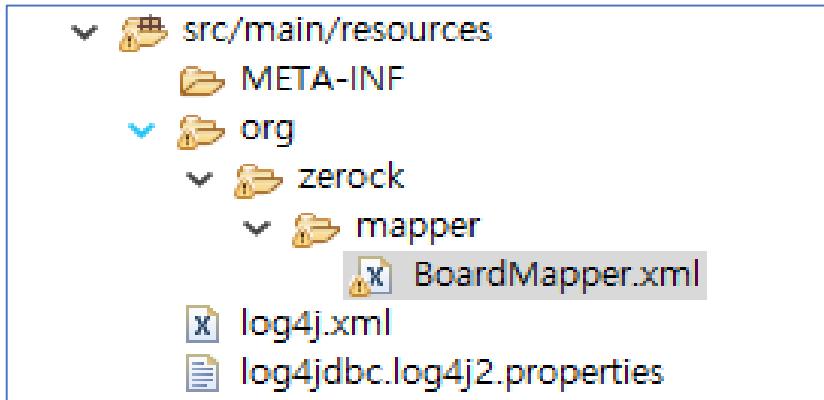
    @Test
    public void testGetList() {

        mapper.getList().forEach(board -> log.info(board));

    }
}
```

```
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|
|bno |title |content |writer |regdate           |updatedate
|-----|-----|-----|-----|-----|
|1  |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:30:51.0 |2018-06-22 11:30:51.0 |
|2  |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|3  |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|4  |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|5  |테스트 제목 |테스트 내용 |user00 |2018-06-22 11:31:21.0 |2018-06-22 11:31:21.0 |
|-----|-----|-----|-----|-----|
INFO : jdbc.resultset - 1. ResultSet.next() returned false
INFO : jdbc.resultset - 1. ResultSet.close() returned void
INFO : jdbc.audit - 1. Connection.getMetaData() returned oracle.jdbc.driver.OracleDatabaseMetaData@4044fb95
INFO : jdbc.audit - 1. PreparedStatement.isClosed() returned false
INFO : jdbc.audit - 1. PreparedStatement.close() returned
```

# Mapper XML파일



## 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="org.zerock.mapper.BoardMapper">

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

</mapper>
```

# 게시물 등록(Create)

- 생성된 게시물의 번호를 사용하는지에 따른 구분
  - insert만 처리되고 생성된 PK 값을 알 필요가 없는 경우
  - insert문이 실행되고, 생성된 PK 값을 알아야 하는 경우
    - <selectkey> 사용

```
public interface BoardMapper {  
  
    // @Select("select * from tbl_board where bno > 0")  
    public List<BoardVO> getList();  
  
    public void insert(BoardVO board);  
  
    public void insertSelectKey(BoardVO board);  
  
}
```

# BoardMapper.xml

...

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

```
<insert id="insertSelectKey">
```

```
<selectKey keyProperty="bno" order="BEFORE"
    resultType="long">
    select seq_board.nextval from dual
</selectKey>
```

```
insert into tbl_board (bno,title,content, writer)
values (#{bno}, #{title}, #{content}, #{writer})
</insert>
```

# <selectkey>

- SQL이 실행되기 전에 별도의 PK값등을 얻기 위해서 사용
- order='before' 를 이용해서 insert구문이 실행되기 전에 호출
- keyProperty를 통해 BoardVO의 bno값으로 세팅

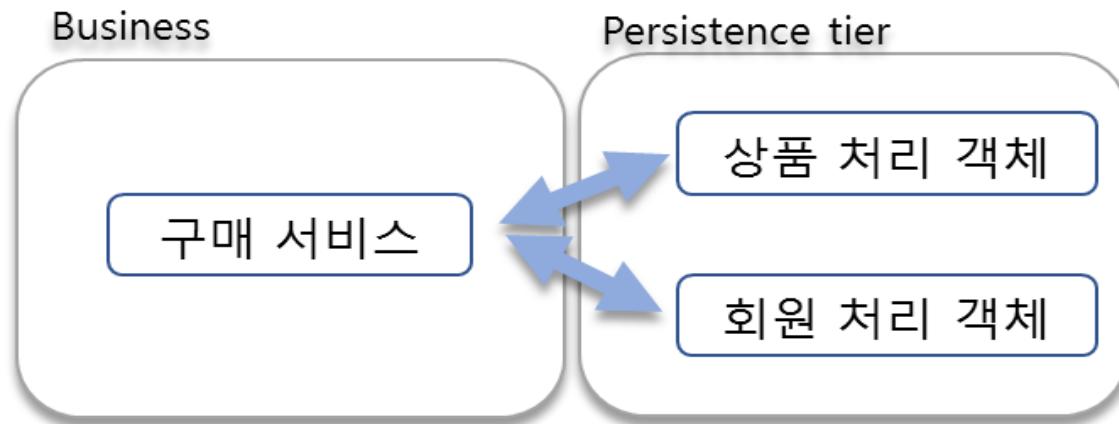
# BoardMapper의 insert테스트

- spring-test를 이용한 테스트 환경을 이용해서 동작 확인
- <selectkey>를 이용하는 경우와 그렇지 않은 경우의 비교

```
@Test  
public void testInsert() {  
  
    BoardVO board = new BoardVO();  
    board.setTitle("새로 작성하는 글");  
    board.setContent("새로 작성하는 내용");  
    board.setWriter("newbie");  
  
    mapper.insert(board);  
  
    log.info(board);  
}
```

# 게시물의 조회(read)

- BoardMapper에 read관련 메서드의 추가
- BoardMapper.xml의 SQL 추가
- 테스트를 통한 확인



```
public interface BoardMapper {  
  
    ...  
  
    public BoardVO read(Long bno);  
  
}
```

```
<select id="read" resultType="org.zerock.domain.BoardVO">  
    select * from tbl_board where bno = #{bno}  
</select>
```

```
@Test  
public void testRead() {  
  
    // 존재하는 게시물 번호로 테스트  
    BoardVO board = mapper.read(5L);  
  
    log.info(board);  
  
}
```

# 게시물의 삭제

- BoardMapper인터페이스에 메서드 추가 – 파라미터는 PK
- BoardMapper.xml의 수정

```
public interface BoardMapper {  
    ...  
    public int delete(Long bno);  
}
```

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

# 게시물의 수정

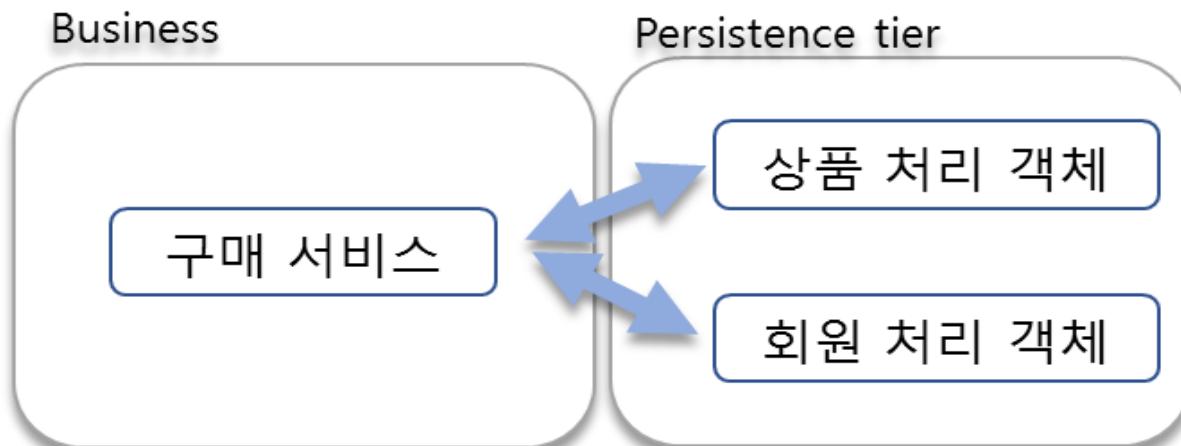
```
public interface BoardMapper {  
    ...  
    public int update(BoardVO board);  
}
```

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

# 9. 비즈니스 계층 구현

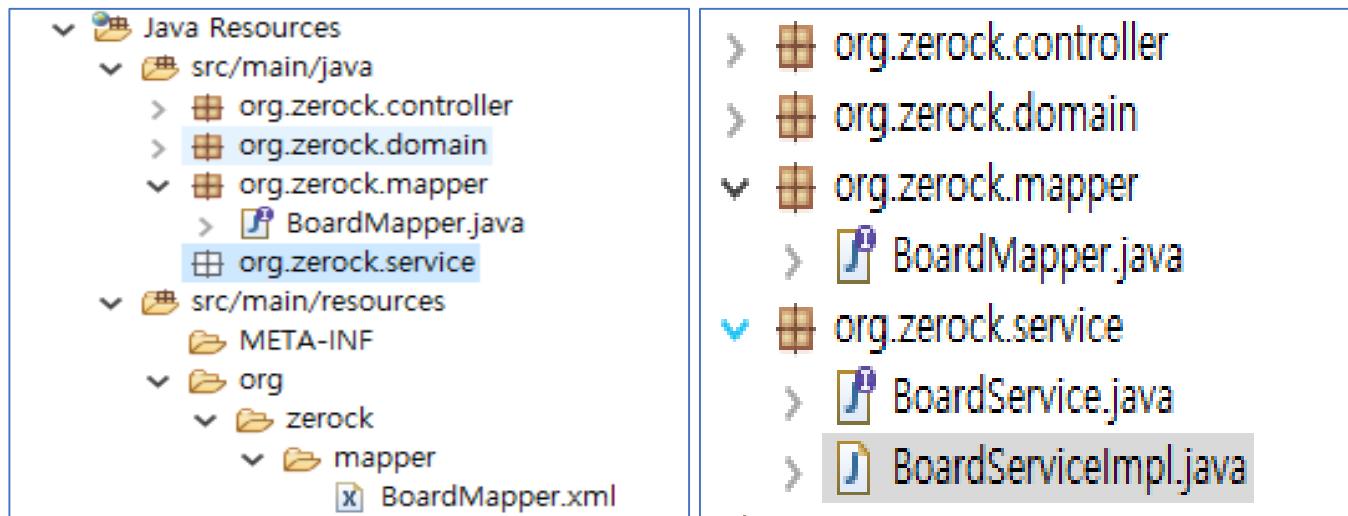
# 비즈니스 계층(서비스 계층)

- 고객의 요구사항을 반영하는 계층
- 업무의 단위로 설계
  - 트랜잭션의 단위
- 여러 개의 Mapper나 DAO를 사용하는 경우가 존재함
- xxxService의 형태로 작성



# 서비스 패키지 설정

- 인터페이스와 클래스를 설정하고, root-context.xml에 등록



```
> org.zerock.controller  
> org.zerock.domain  
<org.zerock.mapper>  
> BoardMapper.java  
<org.zerock.service>  
> BoardService.java  
> BoardServiceImpl.java
```

## root-context.xml의 일부

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

```
public interface BoardService {  
  
    public void register(BoardVO board);  
  
    public BoardVO get(Long bno);  
  
    public boolean modify(BoardVO board);  
  
    public boolean remove(Long bno);  
  
    public List<BoardVO> getList();  
  
}
```

```
@Log4j  
@Service  
@AllArgsConstructor  
public class BoardServiceImpl implements BoardService {  
  
    //spring 4.3 이상에서 자동 처리  
    private BoardMapper mapper;  
  
  
    @Override  
    public void register(BoardVO board) { }  
  
..이하 생략..
```

# @Service 어노테이션

- @Service는 스프링에 빈으로 등록되는 서비스객체의 어노테이션
- XML의 경우에는 <component-scan>에서 조사하는 패키지의 클래스들 중에 @Service가 있는 클래스의 인스턴스를 스프링의 빈으로 설정

# 서비스 계층의 구현과 테스트 진행

- 원칙적으로는 서비스 계층 역시 Mapper나 DAO와 같이 별도로 테스트를 진행하는 것이 바람직
- 하나의 Mapper나 DAO를 이용하는 경우에는 테스트를 생략하는 경우도 많은 편

# 10. 프레젠테이션(웹) 계층의 CRUD 구현

# 웹 계층의 구현

- 웹 계층에서 가장 먼저 설계하는 것은 URI의 설계

Task	URL	Method	Parameter	From	URL 이동
전체 목록	/board/list	GET			
등록 처리	/board/register	POST	모든 항목	입력화면 필요	이동
조회	/board/read	GET	bno=123		
수정 처리	/board/modify	POST	bno	입력화면 필요	이동
삭제 처리	/board/remove	POST	모든 항목	입력화면 필요	이동

# 진행 작업의 순서

- 목록 페이지 - 모든 진입 경로인 동시에 입력을 하는 링크
- 등록 입력/처리 - 게시물 등록 및 처리, 처리후 이동
- 조회 - 목록 페이지에서 특정 게시물로 이동
- 수정/삭제 - 조회 페이지에서 수정/삭제 선택해 처리

# BoardController 목록의 처리

- 게시물(BoardVO)의 목록을 Model에 담아서 전달

```
@Controller  
@Log4j  
@RequestMapping("/board/*")  
@AllArgsConstructor  
public class BoardController {  
  
    private BoardService service;  
  
    @GetMapping("/list")  
    public void list(Model model) {  
  
        log.info("list");  
        model.addAttribute("list", service.getList());  
  
    }  
}
```

# BoardController의 테스트

```
@RunWith(SpringJUnit4ClassRunner.class)

//Test for Controller
@WebAppConfiguration

@ContextConfiguration({
    "file:src/main/webapp/WEB-INF/spring/root-context.xml",
    "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml"})
// Java Config
// @ContextConfiguration(classes = {
//         org.zerock.config.RootConfig.class,
//         org.zerock.config.ServletConfig.class} )
@Log4j
public class BoardControllerTests {

    @Setter(onMethod_ = {@Autowired} )
    private WebApplicationContext ctx;

    private MockMvc mockMvc;
```

```
@Before
public void setup() {
    this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
}

@Test
public void testList() throws Exception {

    Log.info(
        mockMvc.perform(
            MockMvcRequestBuilders.get("/board/list"))
        .andReturn()
        .getModelAndView()
        .getModelMap());
}

}
```

# BoardController의 등록 처리

```
@PostMapping("/register")
public String register(BoardVO board, RedirectAttributes rttr) {

    Log.info("register: " + board);

    service.register(board);

    rttr.addFlashAttribute("result", board.getBno());

    return "redirect:/board/list";
}
```

- POST방식으로 처리되는 데이터를 BoardVO 타입의 인스턴스로 바인딩해서 메서드에서 활용
- BoardService를 이용해서 등록 처리
- 'redirect: '를 이용해서 다시 목록으로 이동

# 등록처리의 테스트

```
@Test  
public void testRegister()throws Exception{  
  
    String resultPage = mockMvc.perform(MockMvcRequestBuilders.post("/board/register")  
        .param("title", "테스트 새글 제목")  
        .param("content", "테스트 새글 내용")  
        .param("writer", "user00")  
    ).andReturn().getModelAndView().getViewName();  
  
    Log.info(resultPage);  
  
}
```

# BoardController의 조회/테스트

```
@GetMapping("/get")
public void get(@RequestParam("bno") Long bno, Model model) {

    Log.info("/get");
    model.addAttribute("board", service.get(bno));
}
```

```
@Test
public void tetGet() throws Exception {

    Log.info(mockMvc.perform(MockMvcRequestBuilders
        .get("/board/get")
        .param("bno", "2"))
        .andReturn()
        .getModelAndView().getModelMap());
}
```

# BoardController의 수정/테스트

```
@PostMapping("/modify")
public String modify(BoardVO board, RedirectAttributes rttr) {
    Log.info("modify:" + board);

    if (service.modify(board)) {
        rttr.addFlashAttribute("result", "success");
    }
    return "redirect:/board/list";
}
```

```
@Test
public void testModify() throws Exception {

    String resultPage = mockMvc
        .perform(MockMvcRequestBuilders.post("/board/modify")
            .param("bno", "1")
            .param("title", "수정된 테스트 새글 제목")
            .param("content", "수정된 테스트 새글 내용")
            .param("writer", "user00"))
        .andReturn().getModelAndView().getViewName();

    Log.info(resultPage);
}
```

# BoardController의 삭제/테스트

```
@PostMapping("/remove")
public String remove(@RequestParam("bno") Long bno, RedirectAttributes rttr) {
    Log.info("remove..." + bno);
    if (service.remove(bno)) {
        rttr.addFlashAttribute("result", "success");
    }
    return "redirect:/board/list";
}
```

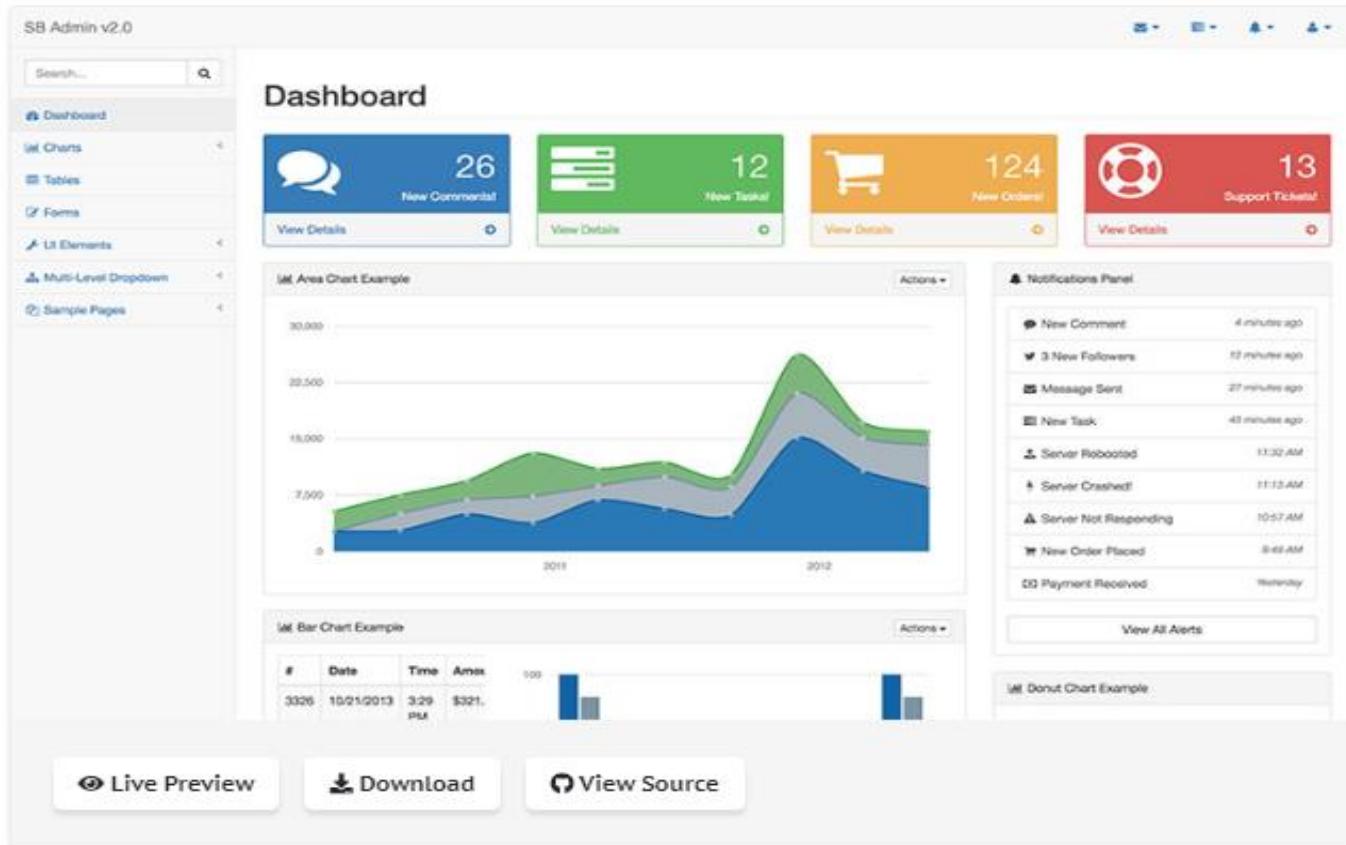
```
@Test
public void testRemove() throws Exception{
    //삭제전 데이터베이스에 게시물 번호 확인할 것
    String resultPage = mockMvc.perform(MockMvcRequestBuilders.post("/board/remove")
        .param("bno", "25")
        .andReturn().getModelAndView().getViewName();

    Log.info(resultPage);
}
```

# 11. 화면 처리

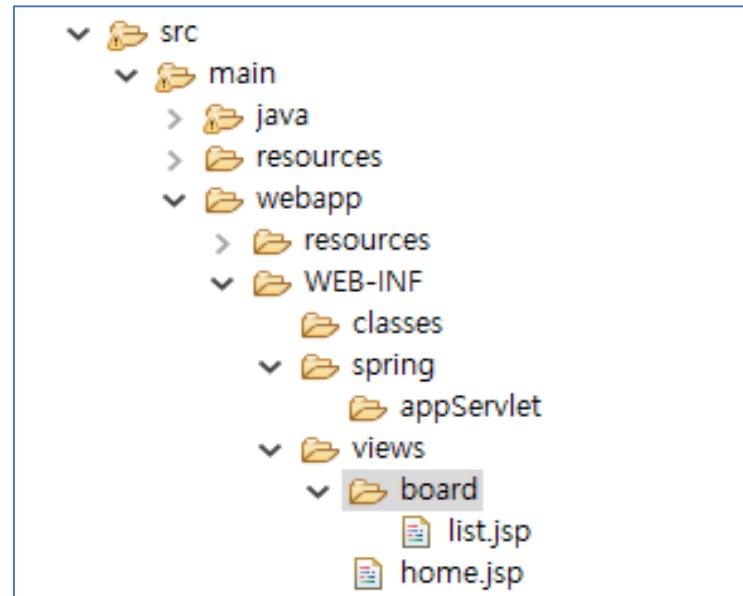
# 화면의 처리 : HTML/CSS JS/JQUERY, Bootstrap

- SB-Admin 2(MIT 라이센스)를 이용해서 페이지 디자인
- <https://startbootstrap.com/template-overviews/sb-admin-2/>



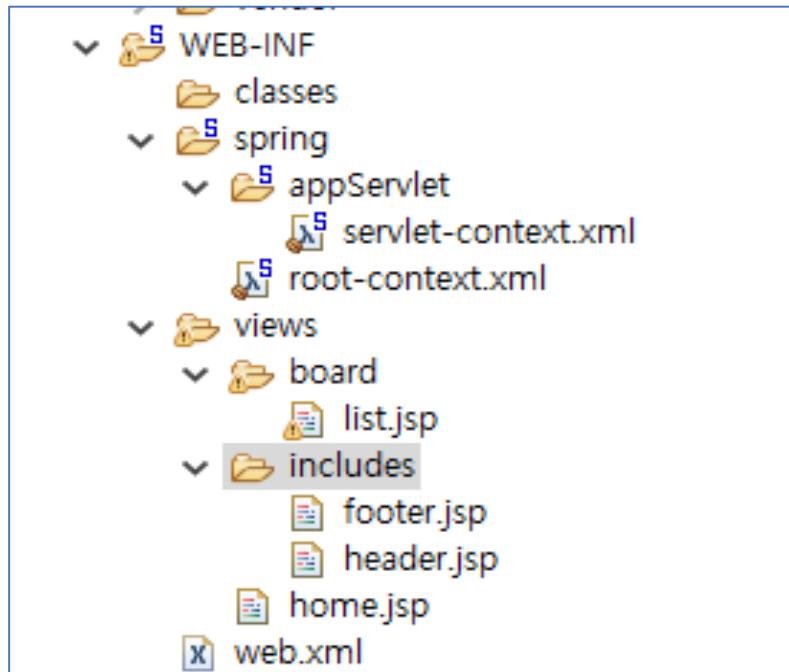
# 목록 페이지 작성

- 프로젝트의 경로는 '/'를 이용하도록 수정
- Tomcat등을 이용해서 실제로 JSP 처리에 문제 없는지 확인 후 진행



# includes 적용

- JSP페이지의 공통으로 사용되는 페이지의 일부를 header.jsp와 footer.jsp로 분리해서 각 페이지에서 include 하는 방식으로 사용



# list.jsp의 적용

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

<%@include file="../includes/header.jsp" %>

<div class="row">
    <div class="col-lg-12">
        <h1 class="page-header">Tables</h1>
...생략...
```

# jQuery 버전 변경과 문제

The image displays two side-by-side screenshots of the SB Admin v2.0 dashboard, specifically the 'Tables' section. Both screenshots show a table with columns for 'Rendering engine', 'Browser', 'Platform(s)', and 'Engine version'. The left screenshot shows the standard layout with a search bar positioned above the table. The right screenshot shows a layout issue where the search bar has moved directly above the table's header row, overlapping it.

jQuery 교체 후 모바일 크기에서 새로 고침 시에 메뉴가 펼쳐지는 문제

Rendering engine	Browser	Platform(s)	Engine version
Gecko	Firefox 1.0	Win 98+ / OSX.2+	1.7
Gecko	Firefox 1.5	Win 98+ / OSX.2+	1.8
Gecko	Firefox 2.0	Win 98+ / OSX.2+	1.8
Gecko	Firefox 3.0	Win 2k+ / OSX.3+	1.9
Gecko	Camino 1.0	OSX.2+	1.8

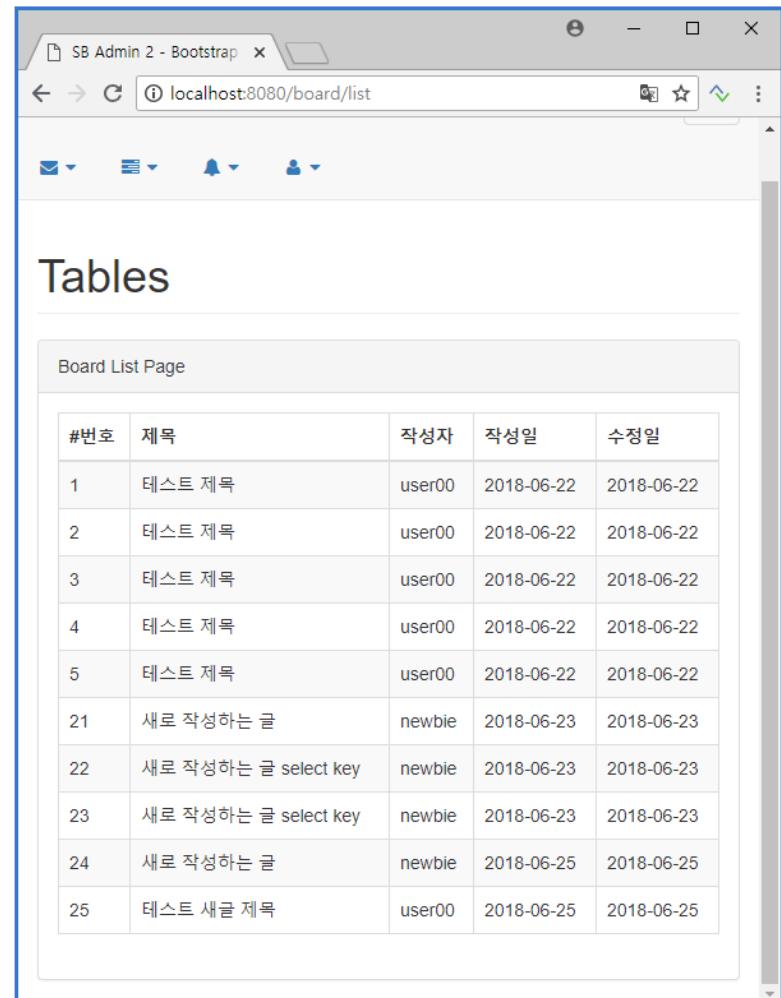
localhost:8080/board/list# Search:

```
<script>
$(document).ready(function() {
    $('#dataTables-example').DataTable({
        responsive: true
    });
    $(".sidebar-nav")
        .attr("class","sidebar-nav navbar-collapse collapse")
        .attr("aria-expanded",'false')
        .attr("style","height:1px");
});
</script>
```

# 목록 화면의 처리

```
<thead>
    <tr>
        <th>#번호</th>
        <th>제목</th>
        <th>작성자</th>
        <th>작성일</th>
        <th>수정일</th>
    </tr>
</thead>

<c:forEach items="${list}" var="board">
    <tr>
        <td><c:out value="${board.bno}" /></td>
        <td><c:out value="${board.title}" /></td>
        <td><c:out value="${board.writer}" /></td>
        <td><fmt:formatDate pattern="yyyy-MM-dd"
            value="${board.regdate}" /></td>
        <td><fmt:formatDate pattern="yyyy-MM-dd"
            value="${board.updateDate}" /></td>
    </tr>
</c:forEach>
</table>
```



The screenshot shows a web browser window titled "SB Admin 2 - Bootstrap" with the URL "localhost:8080/board/list". The page displays a table titled "Tables" under the heading "Board List Page". The table has columns: #번호, 제목, 작성자, 작성일, and 수정일. There are 25 rows of data, each representing a board entry. The first few rows show entries like "테스트 제목" by "user00" on "2018-06-22". Some rows have titles like "새로 작성하는 글" and "새로 작성하는 글 select key".

#번호	제목	작성자	작성일	수정일
1	테스트 제목	user00	2018-06-22	2018-06-22
2	테스트 제목	user00	2018-06-22	2018-06-22
3	테스트 제목	user00	2018-06-22	2018-06-22
4	테스트 제목	user00	2018-06-22	2018-06-22
5	테스트 제목	user00	2018-06-22	2018-06-22
21	새로 작성하는 글	newbie	2018-06-23	2018-06-23
22	새로 작성하는 글 select key	newbie	2018-06-23	2018-06-23
23	새로 작성하는 글 select key	newbie	2018-06-23	2018-06-23
24	새로 작성하는 글	newbie	2018-06-25	2018-06-25
25	테스트 새글 제목	user00	2018-06-25	2018-06-25

# 등록 입력 페이지와 등록

- GET방식으로 게시물 등록 화면 제공
- POST방식으로 실제 게시물 등록 처리
- 이후 목록 페이지로 이동

```
@GetMapping("/register")
public void register() {
```

```
}
```

SB Admin 2 - Bootstrap

localhost:8080/board/register

SB Admin v2.0

Board Register

Board Register

Title  
テ스트

Text area  
テスト

Writer  
user00

Submit Button Reset Button



# 한글 깨짐과 필터 설정

The image shows two screenshots of a web application. On the left is the 'Board Register' page, and on the right is the 'Board List Page'. A large black box highlights a specific row in the list table.

**Board Register**

Board Register

Title  
테스트

Text area  
테스트

Writer  
user00

Submit Button Reset Button

**게시글이 등록되지만 한글이 깨지는 문제가 발생하는 경우**

**Board List Page**

#번호	제목	작성자	작성일	수정일
1	테스트 제목	user00	2018-06-22	2018-06-22
2	테스트 제목	user00	2018-06-22	2018-06-22
3	테스트 제목	user00	2018-06-22	2018-06-22
4	테스트 제목	user00	2018-06-22	2018-06-22
5	테스트 제목	user00	2018-06-22	2018-06-22
21	새로 작성하는 글	newbie	2018-06-23	2018-06-23
22	새로 작성하는 글 select key	newbie	2018-06-23	2018-06-23
	를 select key	newbie	2018-06-23	2018-06-23
	를	newbie	2018-06-25	2018-06-25
25	테스트 새글 제목	user00	2018-06-25	2018-06-25
26	íñóíññíñ,	user00	2018-06-25	2018-06-25

# UTF-8 필터 처리

- web.xml을 이용한 필터 설정

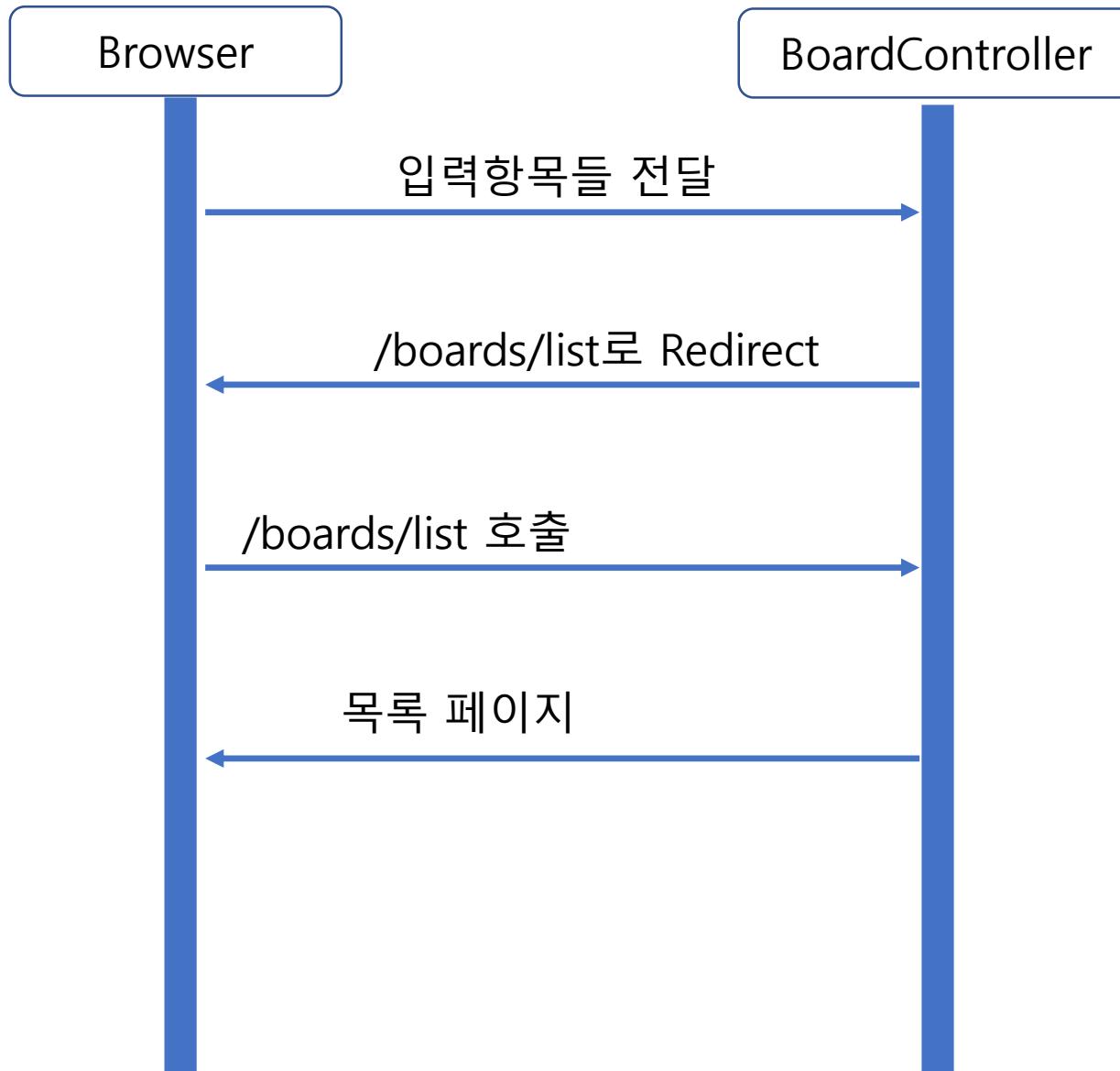
```
<filter>
    <filter-name>encoding</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>encoding</filter-name>
    <servlet-name>appServlet</servlet-name>
</filter-mapping>
```

```
@Override
protected Filter[] getServletFilters() {
    CharacterEncodingFilter characterEncodingFilter = new CharacterEncodingFilter();
    characterEncodingFilter.setEncoding("UTF-8");
    characterEncodingFilter.setForceEncoding(true);

    return new Filter[] { characterEncodingFilter };
}
```

# 재전송(redirect) 처리



# 게시물 작업 이후 재전송

- 게시물의 등록, 수정, 삭제의 경우에 해당
- 작업이 완료된 후에는 리스트 페이지로 다시 이동
- BoardController에서는 RedirectAttributes의 addFlashAttribute( )를 이용해서 단 한번만 전송되는 데이터 저장후 전송
- JSP등의 화면에서는 JavaScript를 이용해서 경고창이나 모달창 등을 보여주는 형식

```
@PostMapping("/register")
public String register(BoardVO board, RedirectAttributes rttr) {

    Log.info("register: " + board);

    service.register(board);

    rttr.addFlashAttribute("result", board.getBno());

    return "redirect:/board/list";
}
```

# 화면의 처리

```
<script type="text/javascript">

$(document).ready(function(){

    var result = '<c:out value="${result}" />';

});

</script>
```

## 작업후 생성되는 결과

```
<script type="text/javascript">

$(document).ready(function(){

    var result = '15';

});

</script>
```

## 일반 호출의 결과

```
<script type="text/javascript">

$(document).ready(function(){

    var result = '';

});

</script>
```

# 모달창 보여주기

- BoardController에서 특정한 데이터가 RedirectAttribute에 포함된 경우에 모달창을 보여주기

```
<script type="text/javascript">
$(document).ready(function() {

    var result = '<c:out value="${result}" />';

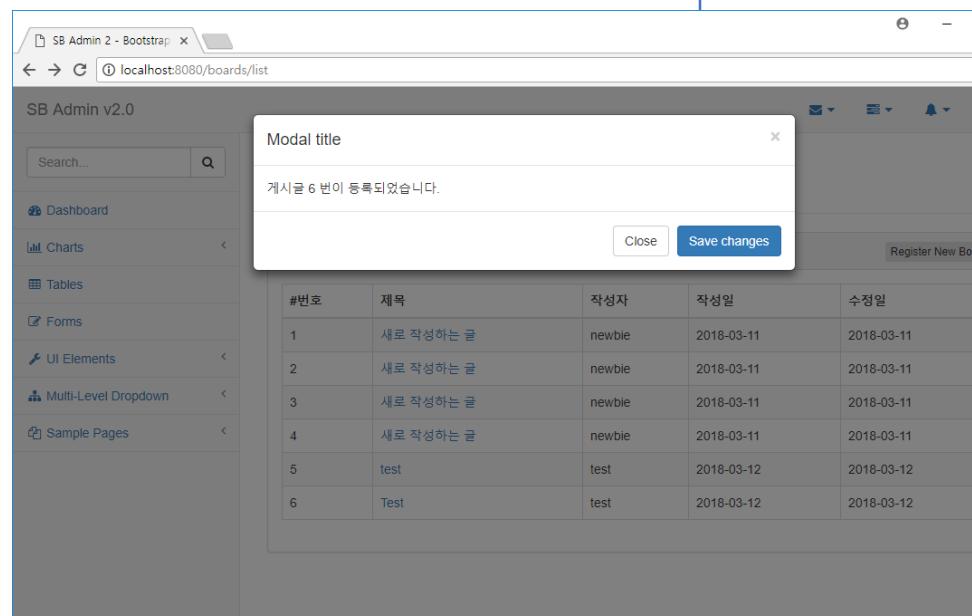
    checkModal(result);

    function checkModal(result) {

        if (result === '') {
            return;
        }

        if (parseInt(result) > 0) {
            $(".modal-body").html(
                "게시글 " + parseInt(result) + " 번이 등록되었습니다.");
        }
        $("#myModal").modal("show");
    }
});

</script>
```

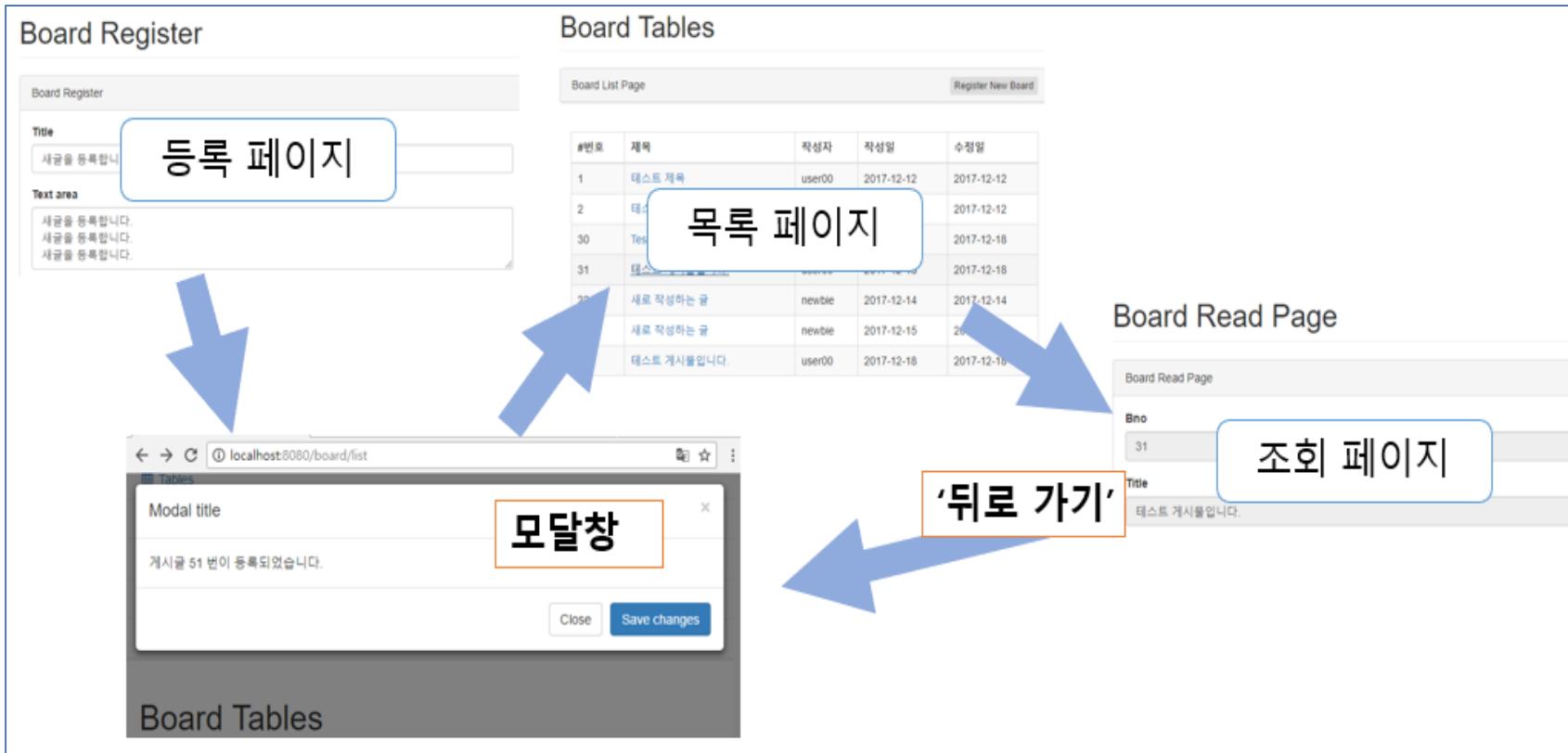


# 조회 페이지와 이동

- 목록에서 특정 게시물 선택후 이동 처리
- 조회의 경우는 반드시 GET방식으로만 처리

```
<tr>
  <td><c:out value= '${board.bno}' /></td>
  <td><a href= '/board/get?bno=<c:out value="${board.bno}" />'><c:out
value= '${board.title}' /></a></td>
  <td><c:out value= '${board.writer}' /></td>
  <td><fmt:formatDate pattern = "yyyy-MM-dd" value = '${board.regdate}' /></td>
  <td><fmt:formatDate pattern = "yyyy-MM-dd" value = '${board.updateDate}' /></td>
</tr>
```

# 뒤로 가기와 window의 history 객체



1

/boards/list ★

2

/boards/register

/boards/list

3

/boards/list ★

/boards/register

/boards/list ★

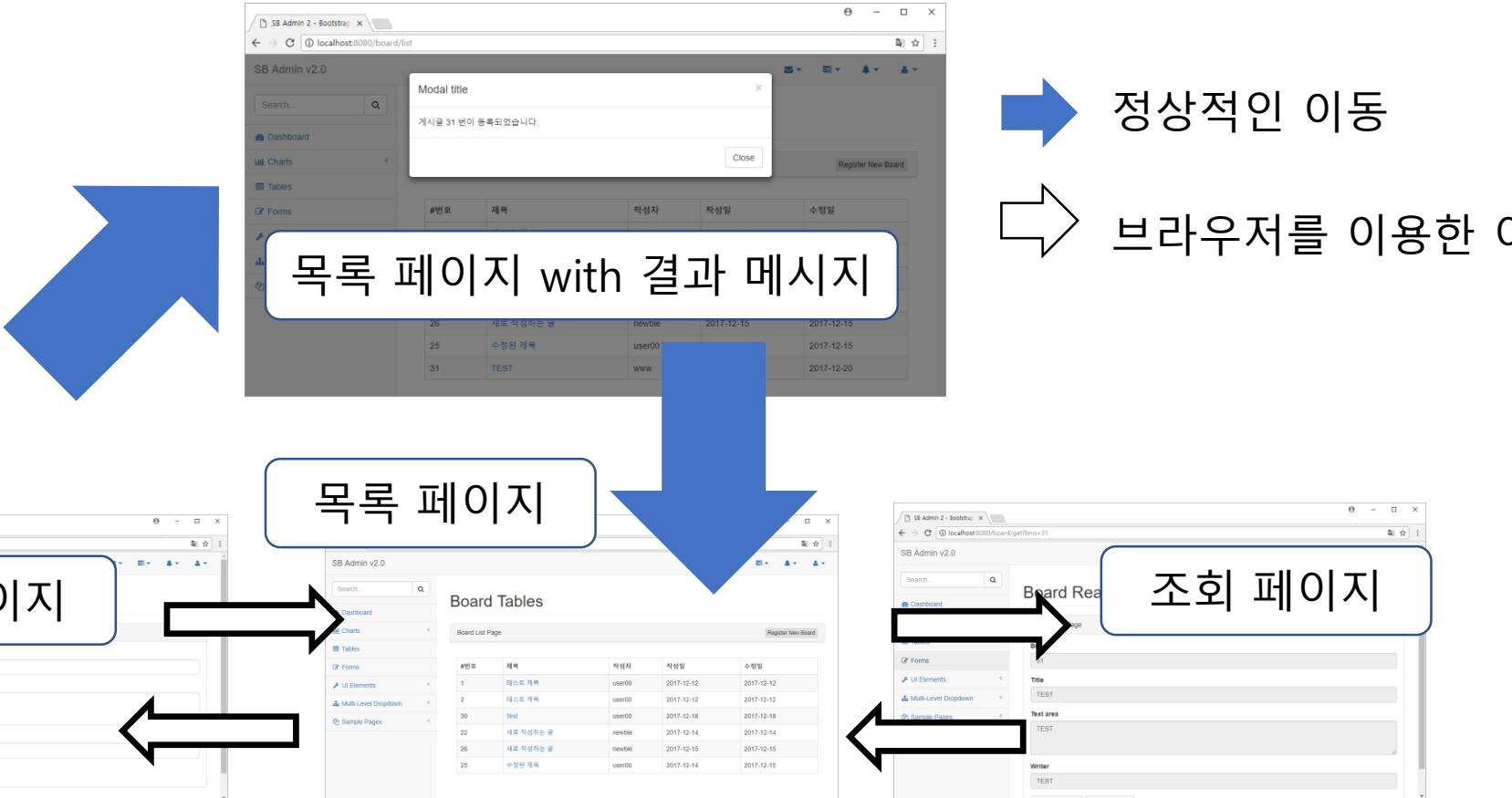
4

/boards/register

/boards/list ★

/boards/register

/boards/list



# 게시물의 수정/삭제 처리

- 게시물 조회이후 수정/삭제 페이지
- 사용자의 선택에 따라서 작업을 처리
- 수정/삭제는 POST방식으로 처리

Board Modify Page

Bno  
19

Title  
Test

Text area  
test

Writer  
Test

Modify Remove List

```
<script type="text/javascript">
$(document).ready(function() {

    var formObj = $("form");

    $('button').on("click", function(e){

        e.preventDefault();

        var operation = $(this).data("oper");

        console.log(operation);

        if(operation === 'remove'){
            formObj.attr("action", "/board/remove");
        }else if(operation === 'list'){
            //move to list
            self.location= "/board/list";
            return;
        }
        formObj.submit();
    });
});</script>
```

# 12. 오라클 DB 페이지 처리

# order by의 고민

- SQL의 실행 과정
  - SQL 파싱
  - SQL 최적화
  - SQL 실행
- SQL의 처리 과정에서 SQL의 실행 계획(execution plan)이 수립
- 데이터의 정렬이 많은 경우에는 order by가 성능에 나쁜 영향

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			7	4
SORT		ORDER BY	7	4
TABLE ACCESS	TBL_BOARD	FULL	7	3
Other XML				
{info}				
info type="db_version"				

# order by 대신에 index

```
select /*+INDEX_DESC(tbl_board pk_board) */
       rownum rn, bno, title, content
  from
    tbl_board
 where rounum <= 10;
```

계획 설명 x 질의 결과 x

SQL | 인출된 모든 행: 10(0,001초)

RN	BNO	TITLE
1	1	3670041 Test
2	2	3670040 테스트 제목
3	3	3670039 테스트 제목
4	4	3670038 TEST
5	5	3670037 수정된 제목
6	6	3670036 TEST
7	7	3670035 수정된 제목
8	8	3670034 새로 작성하는 글
9	9	3670033 새로 작성하는 글
10	10	3670032 Test

SELECT STATEMENT  
  | TABLE ACCESS  
  | INDEX  
    | Access Predicates  
      | BNO>0  
    | Filter Predicates  
      | BNO>0

TBL\_BOARD BY INDEX ROWID  
PK\_BOARD RANGE SCAN DESCENDING

# 식별키(PK)와 인덱스

- PK를 생성하면 자동으로 인덱스가 생성

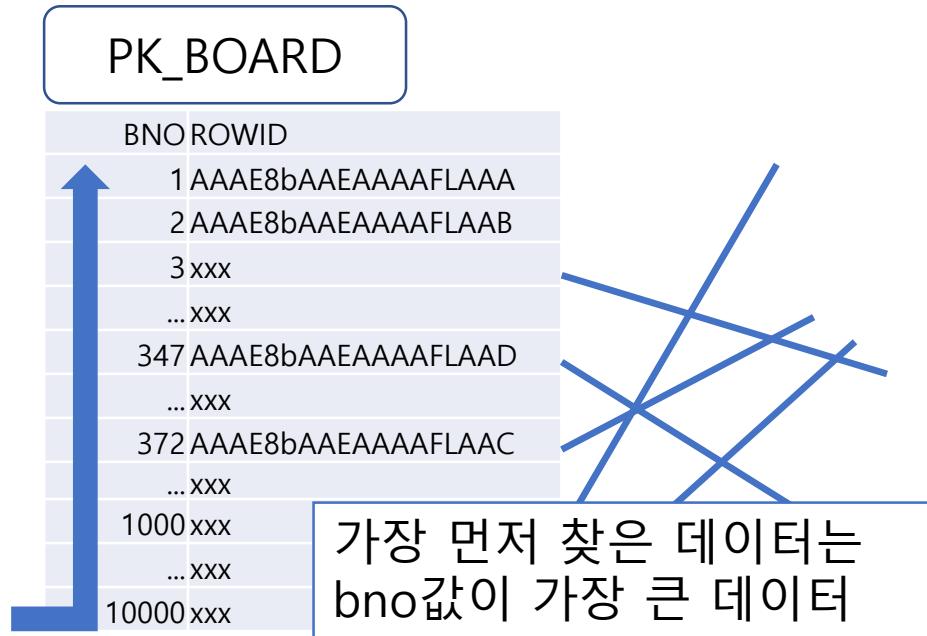
식별키로 만들어진 인덱스

BN	ROWID
1	AAAAE8bAAEAAAAFLAAA
2	AAAAE8bAAEAAAAFLAAB
3 xxx	
... xxx	
347	AAAAE8bAAEAAAAFLAAD
... xxx	
372	AAAAE8bAAEAAAAFLAAC
... xxx	
1000	xxx
... xxx	
10000	xxx

테이블의 데이터

ROWID	BNO	TITLE	CONTENT
AAAAE8bAAEAAAAFLAAA	1	테스트 제목	테스트 내용
AAAAE8bAAEAAAAFLAAB	2	테스트 제목	테스트 내용
AAAAE8bAAEAAAAFLAAC	372	테스트 제목	테스트 내용
AAAAE8bAAEAAAAFLAAD	347	새로 작성하는 글	새로 작성하는 내용
AAAAE8bAAEAAAAFLAAE	348	테스트 제목	테스트 내용
AAAAE8bAAEAAAAFLAAF	349	테스트 제목	테스트 내용
AAAAE8bAAEAAAAFLAAG	350	Test	Test
AAAAE8bAAEAAAAFLAAH	351	새로 작성하는 글	새로 작성하는 내용
AAAAE8bAAEAAAAFLAAI	352	새로 작성하는 글	새로 작성하는 내용
AAAAE8bAAEAAAAFLAAJ	353	수정된 제목	수정된 내용
AAAAE8bAAEAAAAFLAAK	354	TEST	TEST
AAAAE8bAAEAAAAFLAAL	355	수정된 제목	수정된 내용
AAAAE8bAAEAAAAFLAM	356	TEST	TEST
AAAAE8bAAEAAAAFLAN	357	TEST	TEST

# 인덱스를 이용한 정렬



SELECT STATEMENT		
TABLE ACCESS	TBL_BOARD	BY INDEX ROWID
INDEX	PK_BOARD	RANGE SCAN DESCENDING
Access Predicates		
BNO>0		
Filter Predicates		
BNO>0		

# 인덱스를 이용하기 위한 힌트

- 개발자의 의도를 힌트를 이용해서 전달
- 힌트 구문은 잘못되어도 SQL처리에는 지장을 주지 않음
- 여러 종류의 힌트가 존재
  - FULL
  - INDEX\_DESC, INDEX\_ASC

```
SELECT  
/*+ Hint name ( param...) */ column name, ....  
FROM  
table name  
....
```

# ROWNUM과 인덱스

- 테이블에서 최종적으로 나오면서 붙이는 컬럼
- 인덱스를 통해서 나오는지, FULL 스캔을 통하는지에 따라서 ROWNUM이 다르게 나옴

```
select /*+ INDEX_ASC(tbl_board pk_board) */  
    rownum rn, bno, title, content  
  from tbl_board;
```

## PK\_BOARD

BN0	ROWID
1	AAAE8bAAEAAAAFLAAA
2	AAAE8bAAEAAAAFLAAB
3	xxx
...	xxx
347	AAAE8bAAEAAAAFLAAD
...	xxx
372	AAAE8bAAEAAAAFLAAC
...	xxx
1000	xxx
...	xxx
10000	xxx

```
select /*+ INDEX_ASC(tbl_board pk_board) */  
    rownum rn, bno, title, content  
  from tbl_board;
```

가장 먼저 찾은 데이터  
부터 ROWNUM이 1부터  
시작

RN	BNO	TITLE	CONTENT
208	211	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
209	212	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
210	213	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
211	214	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
212	215	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
213	216	수정테스트	test수정테스트수정테스트
214	217	새로 작성하는 글	새로 작성하는 내용
215	218	새로 작성하는 글	새로 작성하는 내용
216	219	테스트 제목	테스트 내용
217	220	테스트 제목	테스트 내용
218	221	테스트 제목	테스트 내용
219	222	테스트 제목	테스트 내용
220	223	수정된 제목	수정된 내용
221	224	새로 작성하는 글	select key 새로 작성하는 내용 select key
222	225	새로 작성하는 글	새로 작성하는 내용

# 인라인뷰와 페이징

```
select /*+INDEX_DESC(tbl_board pk_board) */
       rownum rn, bno, title, content
  from
    tbl_board
 where rownum <= 10;
```

RN	BNO	TITLE	CONTENT	STOPKEY
1	2359299	테스트 게시물입니다.	테스트 게시물입니다.	
2	2359298	테스트 게시물입니다.	테스트 게시물입니다.	
3	2359297	테스트 게시물입니다.	SELECT STATEMENT	
4	2359296	테스트 게시물입니다.	COUNT	
5	2359295	테스트 게시물입니다.	Filter Predicates	
6	2359294	테스트 게시물입니다.	ROWNUM<=10	
7	2359293	테스트 게시물입니다.	TABLE ACCESS	TBL_BOARD BY INDEX ROWID
8	2359292	테스트 게시물입니다.	INDEX	PK_BOARD FULL SCAN DESCENDING
9	2359291	테스트 게시물입니다.	테스트 게시물입니다.	
10	2359290	수정된 테스트 게시물입니다.	테스트 게시물입니다.	

# ROWNUM은 1이 포함된 조건으로

```
select /*+INDEX_DESC(tbl_board pk_board) */  
    rownum rn, bno, title, content  
from  
    tbl_board  
where rownum <= 20;
```

RN	BNO	TITLE	CONTENT
1	2359299	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
2	2359298	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
3	2359297	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
4	2359296	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
5	2359295	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
6	2359294	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
7	2359293	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
8	2359292	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
9	2359291	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
10	2359290	수정된 테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다.
11	2359289	ㅁㄴㅇㄹ	ㅁㄴㅇㄹ
12	2359288	Test	Test
13	2359287	테스트 새글 제목	테스트 새글 내용
14	2359286	새로 작성하는 글	새로 작성하는 내용
15	2359285	새로 작성하는 글 select key	새로 작성하는 내용 select key
16	2359284	수정된 제목	수정된 내용
17	2359283	테스트 제목	테스트 내용
18	2359282	테스트 제목	테스트 내용
19	2359281	테스트 제목	테스트 내용
20	2359280	테스트 제목	테스트 내용

SELECT ....

FROM (

**SELECT ....  
FROM....**

**인라인뷰**

)

```

select
    bno, title, content
from
(
    select /*+INDEX_DESC(tbl_board pk_board) */
        rounum rn, bno, title, content
    from
        tbl_board
    where rounum <= 20
)
where rn > 10;

```

RN	BNO	TITLE	CONTENT
1	2359299	테스트 게시물입니다.	테스트 게시물입니다. 테스트 게시물입니다. 테스트 게시물입니다.
		인라인뷰의 내용	나. 테스트 게시물입니다. 테스트 게시물입니다.
4	2359296	테스트 게시물입니다.	나. 테스트 게시물입니다. 테스트 게시물입니다.
5	2359295	테스트 게시물입니다.	나. 테스트 게시물입니다. 테스트 게시물입니다. 테스트 게시물입니다.
6	2359294	테스트 게시물입니다.	나. 테스트 게시물입니다. 테스트 게시물입니다. 테스트 게시물입니다.
7	2359293	테스트 게시물입니다.	나. 테스트 게시물입니다. 테스트 게시물입니다. 테스트 게시물입니다.
8	2359292	테스트 게시물입니다.	나. 테스트 게시물입니다. 테스트 게시물입니다. 테스트 게시물입니다.
9	2359291	테스트 게시물입니다.	나. 테스트 게시물입니다. 테스트 게시물입니다. 테스트 게시물입니다.
10	2359290	수정된 테스트 게시물입니다.	나. 테스트 게시물입니다. 테스트 게시물입니다. 테스트 게시물입니다.
11	2359289	ㅁㄴㅇㄹ	ㅁㄴㅇㄹ
12	2359288	Test	Test
13	2359287	테스트 새글 제목	테스트 새글 내용
14	2359286	새로 작성하는 글	새로 작성하는 내용
15	2359285	새로 작성하는 글	새로 작성하는 내용
16	2359284	수정된 제목	수정된 내용
17	2359283	테스트 제목	테스트 내용
18	2359282	테스트 제목	테스트 내용
19	2359281	테스트 제목	테스트 내용
20	2359280	테스트 제목	테스트 내용

인라인뷰에서 필요한 부분만 추출

# 13. MyBatis와 스프링 페이징 처리

# 검색에 필요한 내용을 담는 Criteria클래스

- 검색에 사용되는 여러 종류의 데이터를 하나의 객체로 묶기 위한 용도

```
@Getter  
@Setter  
@ToString  
public class Criteria {  
  
    private int pageNum;  
    private int amount;  
  
    public Criteria() {  
        this(1,10);  
    }  
  
    public Criteria(int pageNum, int amount) {  
        this.pageNum = pageNum;  
        this.amount = amount;  
    }  
}
```

# MyBatis의 처리

```
public interface BoardMapper {  
  
    public List<BoardVO> getList();  
  
    public List<BoardVO> getListWithPaging(Criteria cri);  
  
    public void insert(BoardVO board);  
  
    public Integer insertSelectKey(BoardVO board);  
  
    public BoardVO read(Long bno);  
  
    public int delete(Long bno);  
  
    public int update(BoardVO board);  
  
}
```

```
<select id="getListWithPaging"  
       resultType="org.zerock.domain.BoardVO">  
  <![CDATA[  
    select  
      bno, title, content, writer, regdate, updatedate  
    from  
      (  
        select /*+INDEX_DESC(tbl_board pk_board) */  
          rownum rn, bno, title, content, writer, regdate, updatedate  
        from  
          tbl_board  
        where rownum <= #{pageNum} * #{amount}  
      )  
    where rn > (#{pageNum} -1) * #{amount}  
  ]]>  
</select>
```

# 페이지 처리 테스트

```
@Test  
public void testGetList() {  
  
    // service.getList().forEach(board -> log.info(board));  
    service.getList(new Criteria(2, 10)).forEach(board -> Log.info(board));  
}
```

cri.setPageNum(1);  
cri.setAmount(10);

cri.setPageNum(2);  
cri.setAmount(10);

cri.setPageNum(3);  
cri.setAmount(10);

2359299  
2359298  
2359297  
2359296  
2359295  
2359294  
2359293  
2359292  
2359291  
2359290

2359289  
2359288  
2359287  
2359286  
2359285  
2359284  
2359283  
2359282  
2359281  
2359280

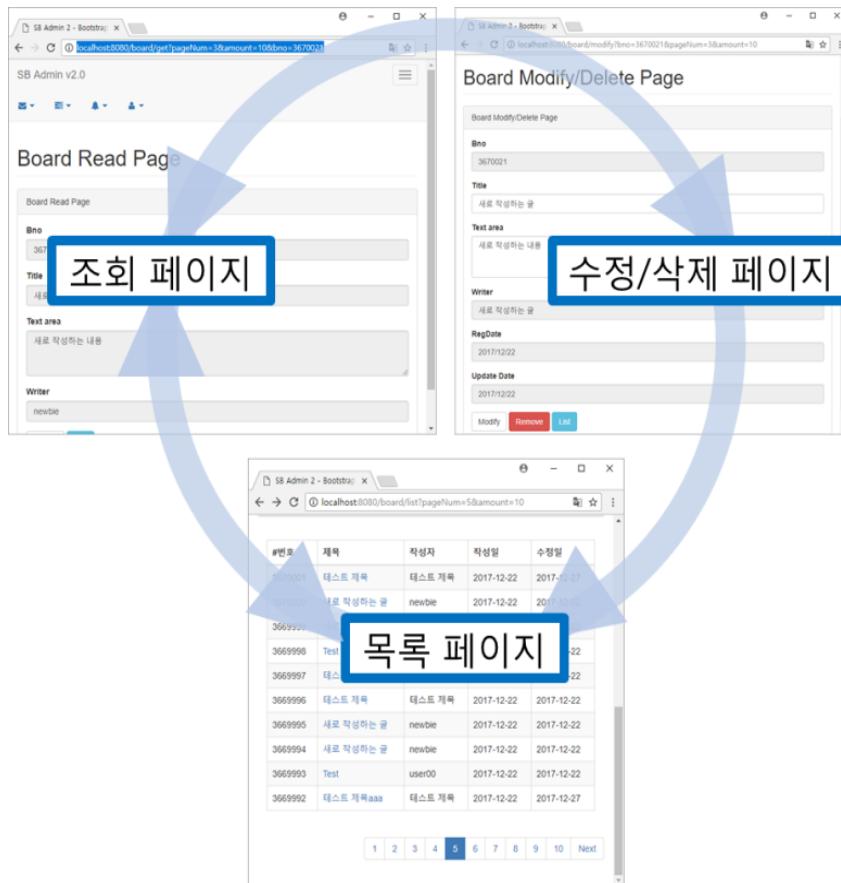
2359279  
2359278  
2359277  
2359276  
2359275  
2359274  
2359273  
2359272  
2359271  
2359270



# 14. 화면 페이징 처리

# JSP 처리

- BoardController/BoardService/BoardServiceImpl 처리
- list.jsp 처리



# 페이지 처리에 필요한 정보들

- 현재 페이지 번호(page)
- 이전과 다음으로 이동 가능한 링크의 표시 여부(prev, next)
- 화면에서 보여지는 페이지의 시작 번호와 끝 번호(startPage, endPage)

페이지의 끝 번호(endPage) 계산

```
this.endPage = (int)(Math.ceil(페이지번호 / 10.0)) * 10;
```

페이지의 시작 번호(startPage) 계산

```
this.startPage = this.endPage - 9;
```

total을 통한 endPage의 재계산

```
realEnd = (int) (Math.ceil((total * 1.0) / amount));
```

```
if(realEnd < this.endPage) {  
    this.endPage = realEnd;  
}
```

## 이전(prev) 계산

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

## 다음(next) 계산

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

```
@Getter  
@ToString  
public class PageDTO {  
  
    private int startPage;  
    private int endPage;  
    private boolean prev, next;  
  
    private int total;  
    private Criteria cri;  
  
    public PageDTO(Criteria cri, int total) {  
  
        this.cri = cri;  
        this.total = total;  
  
        this.endPage = (int) (Math.ceil(cri.getPageNum() / 10.0)) * 10;  
  
        this.startPage = this.endPage - 9;  
  
        int realEnd = (int) (Math.ceil((total * 1.0) / cri.getAmount()));  
  
        if (realEnd <= this.endPage) {  
            this.endPage = realEnd;  
        }  
  
        this.prev = this.startPage > 1;  
  
        this.next = this.endPage < realEnd;  
    }  
}
```

# JSP에서 화면 번호 출력

```
<div class='pull-right'>
    <ul class="pagination">

        <c:if test="${pageMaker.prev}">
            <li class="paginate_button previous"><a href="#">Previous</a>
            </li>
        </c:if>

        <c:forEach var="num" begin="${pageMaker.startPage}"
            end="${pageMaker.endPage}">
            <li class="paginate_button"><a href="#">${num}</a></li>
        </c:forEach>

        <c:if test="${pageMaker.next}">
            <li class="paginate_button next"><a href="#">Next</a></li>
        </c:if>
    </ul>
</div>
<!--  end Pagination -->
</div>
```

SB Admin 2 - Bootstrap

localhost:8080/board/list?pageNum=5

## Tables

Board List Page

Register New Board

#번호	제목	작성자	작성일	수정일
2359259	새로 작성하는 글 select key	newbie	2018-04-03	2018-04-03
2359258	수정된 제목	user00	2018-04-03	2018-04-03
2359257	<a href="http://localhost:8080/board/list?pageNum=5">http://localhost:8080/board/list? pageNum=5</a>			
2359256				
2359255				
2359254	테스트 제목	user00	2018-04-03	2018-04-03
2359253	새로 작성하는 글	newbie	2018-04-03	2018-04-03
2359252	수정테스트	Test	2018-04-03	2018-04-03
2359251	Test	Test	2018-04-03	2018-04-03
2359250	테스트 새글 제목	user00	2018-04-03	2018-04-03

1 2 3 4 5 6 7 8 9 10 Next

SB Admin 2 - Bootstrap

localhost:8080/board/list?pageNum=5&amount=20

2359214	테스트 게시글입니다.	user00	2018-04-03	2018-04-03
2359213	테스트 게시글입니다.	user00	2018-04-03	2018-04-03
2359212	테스트 게시글입니다.	user00	2018-04-03	2018-04-03
2359211	테스트 게시글입니다.	user00	2018-04-03	2018-04-03
2359210	수정된 테스트 게시글입니다.	user00	2018-04-03	2018-04-03
2359209	ㅁㄴㅇㄹ	ㅁㄴㅇㄹ	2018-04-03	2018-04-03
2359208	Test	Test	2018-04-03	2018-04-03
2359207	<a href="http://localhost:8080/board/list?&lt;br/&gt;pageNum=5&amp;amount=20">http://localhost:8080/board/list? pageNum=5&amp;amount=20</a>			
2359206				
2359205				
2359204	새로 작성하는 글 select key	newbie	2018-04-03	2018-04-03
2359203	수정된 제목	user00	2018-04-03	2018-04-03
2359202	테스트 제목	user00	2018-04-03	2018-04-03
2359201	테스트 제목	user00	2018-04-03	2018-04-03
2359200	테스트 제목	user00	2018-04-03	2018-04-03

1 2 3 4 5 6 7

# 페이지 번호 이벤트 처리

- 페이지번호의 링크는 페이지 번호만 가지도록 하고
- 별도의 <form>태그와 이벤트 처리를 통해서 이동

```
<form id='actionForm' action="/board/list" method='get'>
  <input type='hidden' name='pageNum' value = '${pageMaker.cri.pageNum}'>
  <input type='hidden' name='amount' value = '${pageMaker.cri.amount}'>
</form>
```

```
var actionForm = $("#actionForm");

$(".paginate_button a").on("click", function(e) {

  e.preventDefault(); //기본 동작 제한
  console.log( ' click ' );
  //<form>태그의 내용 변경후 submit
  actionForm.find("input[name='pageNum']").val($(this).attr("href"));
  actionForm.submit();
});
```

# 조회 페이지로 이동

- 조회 페이지 이동시 현재 페이지 번호를 같이 전달하는 방식으로 처리되어야 함

The diagram illustrates the flow of data between three pages:

- Top Left:** A screenshot of a browser window titled "Admin 2 - Bootstrap" showing a list of board posts. The URL in the address bar is `http://localhost:8080/board/list?pageNum=3&amount=10`. A red box highlights the first post in the list.
- Top Right:** A screenshot of a browser window titled "Admin 2 - Bootstrap" showing the details of the selected post. The URL is `http://localhost:8080/board/get?bno=2359279`. The page title is "Board Read Page". The post details show the title "수정테스트" and the content "Test". A red box highlights the "Edit" button in the bottom right corner.
- Bottom:** A screenshot of a browser window titled "Admin 2 - Bootstrap" showing the list of board posts again. The URL is `http://localhost:8080/board/list?`. A large blue arrow points from the "Edit" button on the detail page down to the list page, indicating that the current page number ( pageNum=3 ) is lost during the transition.
- Text Box:** A box at the bottom center contains the text "조회 페이지에서 목록 페이지로 이동시 문제가 생김" (Problem occurs when moving from the search page to the list page).

```

<td>
  <a class='move' href='<c:out value='${board.bno}'/>'>
    <c:out value='${board.title}' /></a>
  </td>

```

2359291	테스트 게시물입니다.
2359290	수정된 테스트 게시물입니다.

1 2 3

localhost:8080/board/2359291

## list.jsp 게시물 조회를 위한 이벤트 처리 추가

```

$( ".move" ).on( "click", function(e){

  e.preventDefault();
  actionForm.append(" <input type='hidden' name='bno' value='"+ $(this).attr("href")+"'" );
  actionForm.attr("action", "/board/get");
  actionForm.submit();

});

```

The diagram illustrates the interaction between two JSP pages:

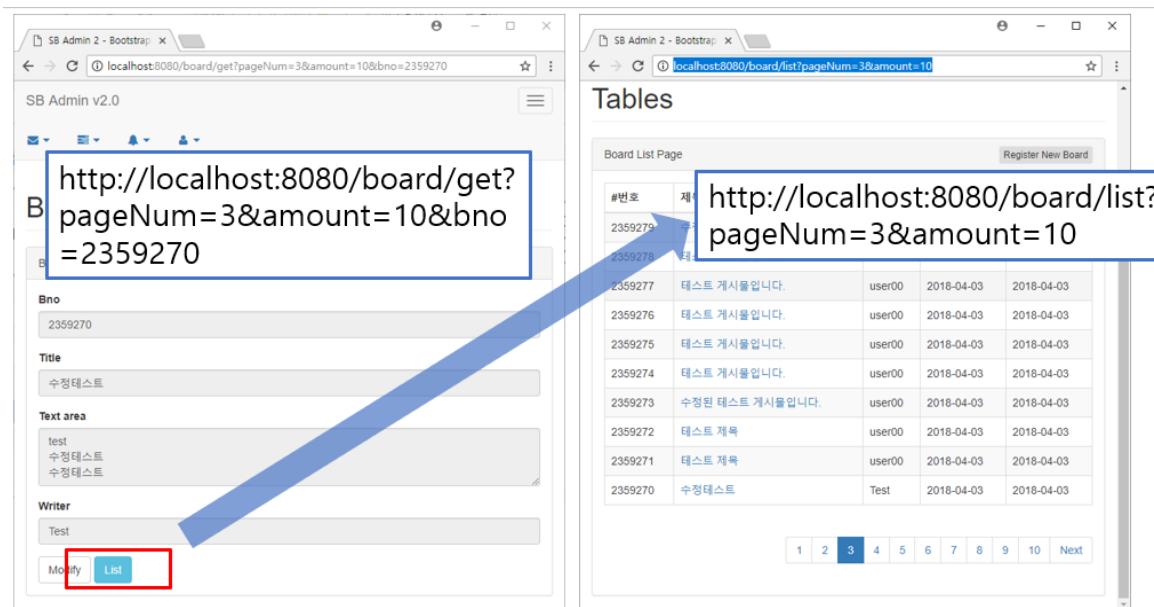
- Left Window (Board List Page):** Shows a table of board entries. An arrow points from the row with bno 2359285 to the right window.
- Right Window (Board Read Page):** Shows a form with the URL `http://localhost:8080/board/get?pageNum=2&amount=10&bno=2359285`. The 'Bno' field is populated with '2359285'.

# 조회 페이지에서 목록페이지로

- 전달받은 페이지 번호를 이용해서 원래 페이지로 이동

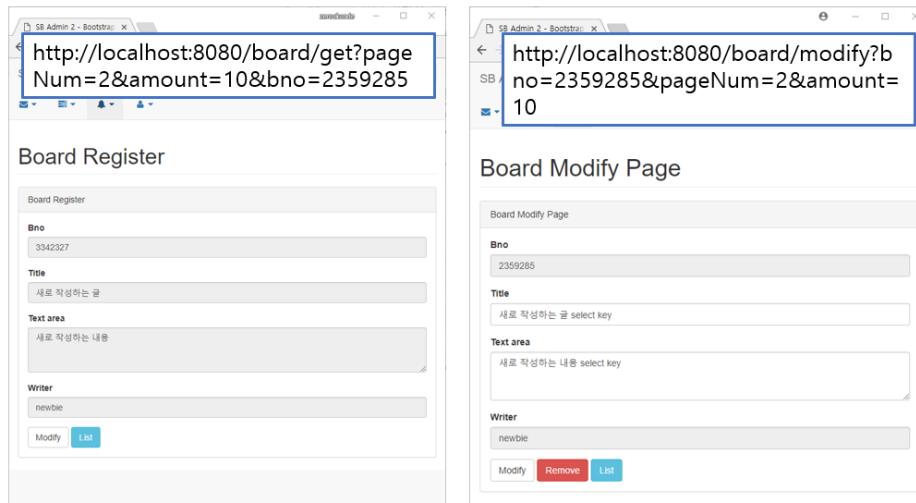
views/board/get.jsp의 일부

```
<form id='openForm' action="/board/modify" method="get">
    <input type='hidden' id='bno' name='bno' value='<c:out value='${board.bno}'/>'>
    <input type='hidden' name='pageNum' value='<c:out value='${cri.pageNum}'/>'>
    <input type='hidden' name='amount' value='<c:out value='${cri.amount}'/>'>
</form>
```



# 수정/삭제

- 수정/삭제 페이지에서도 목록으로 이동할 수 있도록 전달



# 수정/삭제 후 이동

```
@PostMapping("/remove")
public String remove(@RequestParam("bno") Long bno, @ModelAttribute("cri") Criteria cri, RedirectAttributes rttr)
{
    Log.info("remove..." + bno);
    if (service.remove(bno)) {
        rttr.addFlashAttribute("result", "success");
    }
    rttr.addAttribute("pageNum", cri.getPageNum());
    rttr.addAttribute("amount", cri.getAmount());

    return "redirect:/board/list";
}
```

<http://localhost:8080/boards/list?pageNum=5&amount=10>

번호	제목	작성자	작성일	수정일
1	수정된 제목	user00	2018-04-03	2018-04-03
2359257	테스트 제목	user00	2018-04-03	2018-04-03
2359256	테스트 제목	user00	2018-04-03	2018-04-03
2359255	테스트 제목	user00	2018-04-03	2018-04-03
2359254	테스트 제목	user00	2018-04-03	2018-04-03
2359253	새로 작성하는 글	newbie	2018-04-03	2018-04-03
2359252	수정테스트	Test	2018-04-03	2018-04-03
2359251	Test	Test	2018-04-03	2018-04-03
2359250	테스트 새글 제목	user00	2018-04-03	2018-04-03

<http://localhost:8080/board/get?pageNum=5&amount=10&bno=2359251>

Board Read Page

Bno	2359251
Title	Test
Text area	Test
Writer	Test

Modify List

<http://localhost:8080/board/modify?bno=2359251&pageNum=5&amount=10>

Board Modify Page

Bno	2359251
Title	Test
Text area	Test
Writer	Test

Modify Remove List

1

Board Read Page

2

Board Modify Page

3

# 게시물의 숫자 처리

```
public interface BoardMapper {  
  
    ...생략...  
  
    <select id="getTotalCount" resultType="int">  
        select count(*) from tbl_board where bno > 0  
    </select>  
  
    public int getTotalCount(Criteria cri);  
}
```

## BoardController 클래스의 일부

```
@GetMapping("/list")  
public void list(Criteria cri, Model model) {  
  
    log.info("list: " + cri);  
    model.addAttribute("list", service.getList(cri));  
    //model.addAttribute("pageMaker", new PageDTO(cri, 123));  
  
    int total = service.getTotal(cri);  
  
    log.info("total: " + total);  
  
    model.addAttribute("pageMaker", new PageDTO(cri, total));  
}
```

# 15. 검색 처리

# 검색의 유형

- 제목, 내용, 작성자와 같은 단일 항목
- 제목 + 내용, 제목 + 작성자와 같은 복합 항목
- 검색 항목에 따라서 매번 다른 SQL이 처리될 필요가 있는 상황
- MyBatis의 동적쿼리기능을 이용해서 처리
  - <http://www.mybatis.org/mybatis-3/ko/dynamic-sql.html>

# MyBatis의 동적 태그들

- if
  - if 는 test라는 속성과 함께 특정한 조건이 true가 되었을 때 포함된 SQL을 사용하고자 할 때 작성
- choose (when, otherwise)
  - if와 달리 choose는 여러 상황들 중 하나의 상황에서만 동작
- trim (where, set)
  - trim, where, set은 단독으로 사용되지 않고, <if>, <choose>와 같은 태그들을 내포하여 SQL들을 연결해 주고, 앞 뒤에 필요한 구문들(AND, OR, WHERE 등)을 추가하거나 생략하는 역할
- foreach
  - foreach는 List, 배열, 맵 등을 이용해서 루프를 처리

# foreach 태그

```
Map<String, String> map = new HashMap<>();  
  
map.put("T", "TTTT");  
map.put("C", "CCCC");
```

```
select * from tbl_board  
  
<trim prefix="where (" suffix=")" prefixOverrides="OR" >  
<foreach item="val" index="key" collection="map">  
  
    <trim prefix = "OR" >  
        <if test="key == 'C'.toString()">  
            content = #{val}  
        </if>  
        <if test="key == 'T'.toString()">  
            title = #{val}  
        </if>  
        <if test="key == 'W'.toString()">  
            writer = #{val}  
        </if>  
    </trim>  
</foreach>  
</trim>
```

# 검색조건처리를 위한 Criteria클래스 변경

- 검색항목(type)과 검색 키워드(keyword)추가

```
public class Criteria {  
  
    private int pageNum;  
    private int amount;  
  
    private String type;  
    private String keyword;  
  
}
```

# BoardMapper.xml의 변경

```
<select id="getListWithPaging" resultType="org.zerock.domain.BoardVO">
...생략..
<trim prefix= "(" suffix= ")" AND prefixOverrides="OR ">
<foreach item='type' collection="typeArr">
<trim prefix="OR ">
<choose>
<when test="type == 'T'.toString()">
    title like '%' ||#{keyword}|| '%'
</when>
<when test="type == 'C'.toString()">
    content like '%' ||#{keyword}|| '%'
</when>
<when test="type == 'W'.toString()">
    writer like '%' ||#{keyword}|| '%'
</when>
</choose>
</trim>
</foreach>
</trim>
...생략..
</select>
```

# 테스트

```
@Test  
public void testSearch() {  
  
    Criteria cri = new Criteria();  
    cri.setKeyword("새로");  
    cri.setType("TC");  
  
    List<BoardVO> list = mapper.getListWithPaging(cri);  
  
    list.forEach(board -> Log.info(board));  
}
```

```
Criteria cri = new Criteria();  
cri.setKeyword("키워드");  
cri.setType("T");
```

```
pk_board) /* rounum rn, bno, title, content, writer, regdate, updatedate from  
( title like '%'||'키워드'|| '%' ) AND rounum <= 1 * 10 ) where rn > (1 -1) * 10
```

```
Criteria cri = new Criteria();  
cri.setKeyword("키워드");  
cri.setType("TC");
```

```
INFO : jdbc.sqltiming - select bno, title, content, writer, regdate, updatedate from ( select  
pk_board) /* rounum rn, bno, title, content, writer, regdate, updatedate from tbl_board where  
( title like '%'||'키워드'|| '%' OR content like '%'||'키워드'|| '%' ) AND rounum <= 1 * 10 ) where  
rn > (1 -1) * 10
```

```
Criteria cri = new Criteria();  
cri.setKeyword("키워드");  
cri.setType("TCW");
```

```
INFO : jdbc.sqltiming - select bno, title, content, writer, regdate, updatedate from ( select /*  
pk_board) /* rounum rn, bno, title, content, writer, regdate, updatedate from tbl_board where  
( title like '%'||'키워드'|| '%' OR content like '%'||'키워드'|| '%' OR writer like '%'||'키워드'|| '%'  
) AND rounum <= 1 * 10 ) where rn > (1 -1) * 10
```

# <sql> 조각과 <include>

- 게시물의 검색과 게시물의 숫자 카운트에 공통으로 사용되므로 <sql> 조각으로 분리하고, 필요한 곳에서 <include>하는 방식으로 적용

```
<sql id="criteria">
    <trim prefix="(" suffix=") AND " prefixOverrides="OR">
        ...생략...
    </trim>
</sql>
```

```
<select id="getListWithPaging" resultType="org.zerock.domain.BoardVO">
    <![CDATA[
    ...생략...

    <include refid="criteria"></include>

    <![CDATA[
        rnum <= #{pageNum} * #{amount}
    )
    where rn > (#{pageNum} -1) * #{amount}
    ]]>
</select>
```

# 화면에서의 검색처리

```
<form id='searchForm' action="/board/list" method='get'>
  <select name='type'>
    <option value="">--</option>
    <option value="T">제목</option>
    <option value="C">내용</option>
    <option value="W">작성자</option>
    <option value="TC">제목 or 내용</option>
    <option value="TW">제목 or 작성자</option>
    <option value="TWC">제목 or 내용 or 작성자</option>
  </select>
  <input type='text' name='keyword' />
  <input type='hidden' name='pageNum' value='${pageMaker.cri.pageNum}'>
  <input type='hidden' name='amount' value='${pageMaker.cri.amount}'>
  <button class='btn btn-default'>Search</button>
</form>
```

The diagram illustrates the search process. On the left, a user interface is shown with a dropdown menu set to '제목' (Title), a text input field containing '작성하는', and a 'Search' button. A large blue arrow points from this interface to a browser window on the right. The browser window displays a table of board list results. Below the browser window, the URL and search parameters are shown.

URL and Search Parameters:

```
http://localhost:8080/board/list?type=T&keyword=%EC%9E%91%EC%84%B1%ED%95%98%EB%8A%94&pageNum=1&amount=10
```

Table Data (Screenshot from the browser window):

#번호	제목	작성자	작성일	수정일
3670052	새로 작성하는 글 select key	newbie	2017-12-30	2017-12-30
3670051	새로 작성하는 글	newbie	2017-12-30	2017-12-30
3670034	새로 작성하는 글	newbie	2017-12-22	2017-12-22
3670033	새로 작성하는 글	newbie	2017-12-22	2017-12-22
	wble		2017-12-22	2017-12-22
	wble		2017-12-22	2017-12-22
	로 작성하는 글 Test		2017-12-22	2017-12-27
	로 작성하는 글		2017-12-22	2017-12-27
	wble		2017-12-22	2017-12-22

User Interface Fields (Left):

- 제목 (Selected)
- 작성하는
- Search

User Interface Fields (Bottom):

- 
- Search

Pagination (Bottom):

- 1 2 3 4 5 6 7 8 9 10 Next

# 검색이벤트 처리

```
$( "#searchForm button").on("click", function(e){

    if(!searchForm.find("option:selected").val()){
        alert("검색종류를 선택하세요");
        return false;
    }

    if(!searchForm.find("input[name='keyword']").val()){
        alert("키워드를 입력하세요");
        return false;
    }

    searchForm.find("input[name='pageNum']").val("1");
    e.preventDefault();

    searchForm.submit();

});
```

# 검색후 <select> 처리

- 'Search'를 클릭하면 무조건 1페이지로
- 이후에는 검색 항목 유지

검색 항목이 선택되지 않은 경우

localhost:8080/board/list?type=C&keyword=&pageNum=1&amount=10

ID	제목	출처	작성자	날짜
2359294	테스트	출처: localhost:8080		2018-04-03
2359293	테스트	검색종류를 선택하세요		2018-04-03
2359292	테스트			2018-04-03
2359291	테스트 게시글입니다.	user00	2018-04-03	2018-04-03
2359290	수정된 테스트 게시물입니다.	user00	2018-04-03	2018-04-04

-- ▾  Search

1 2 3 4 5 6 7 8 9 10 Next

키워드를 입력하지 않은 경우

localhost:8080/board/list?type=C&keyword=&pageNum=1&amount=10

ID	제목	출처	작성자	날짜
2359294	테스트	출처: localhost:8080		2018-04-03
2359293	테스트	키워드를 입력하세요		2018-04-03
2359292	테스트			2018-04-03
2359291	테스트 게시글입니다.	user00	2018-04-03	2018-04-03
2359290	수정된 테스트 게시물입니다.	user00	2018-04-03	2018-04-04

제목 ▾  Search

1 2 3 4 5 6 7 8 9 10 Next

검색시에는 무조건 1페이지로

localhost:8080/board/list?type=C&keyword=Test&pageNum=1&amount=10

ID	제목	출처	작성자	날짜
2359162	Test	Test	2018-04-03	2018-04-03
2359145	Test	Test	2018-04-03	2018-04-03
2359118	Test	Test	2018-04-03	2018-04-03
2359114	Test	Test	2018-04-03	2018-04-03

제목 ▾ Test Search

1 2 3 4 5 6 7 8 9 10 Next

# 기타 검색 처리

- 검색후 조회,수정,삭제 페이지 이동시 검색 항목 유지
  - RedirectAttribute에 검색 관련 내용을 추가하는 방식
  - UriComponentsBuilder를 이용해서 Criteria에서 링크를 생성하는 방식

## Criteria 클래스의 일부

```
public String getListLink() {  
  
    UriComponentsBuilder builder = UriComponentsBuilder.fromPath("")  
        .queryParam("pageNum", this.pageNum)  
        .queryParam("amount", this.getAmount())  
        .queryParam("type",this.getType())  
        .queryParam("keyword", this.getKeyword());  
  
    return builder.toUriString();  
  
}
```

# 스프링 웹 프로젝트

Restful 방식

# Objectives

- REST방식의 데이터 교환 방식의 이해
- Ajax를 통한 JSON 데이터 통신
- jQuery를 이용하는 Ajax 처리
- JavaScript의 모듈 패턴

# 16. REST방식으로 전환

# 웹의 과거와 현재

- 과거의 웹 서비스
  - 고정된 브라우저의 주소창
  - 특정한 확장자를 이용하는 모델 2 방식(ex> \*.do)
  - 특정한 파라미터에 의한 분기 구조
- 현재의 웹 서비스
  - URI + 식별데이터
  - GET/POST외에 PUT/DELETE 등의 다양한 전송 방식 사용
  - 서버에서는 순수한 데이터만을 서비스 하는 방식



# REST방식

- REST는 'Representational State Transfer'의 약어로 하나의 URI는 하나의 고유한 리소스(Resource)를 대표하도록 설계된다는 개념에 전송방식을 결합해서 원하는 작업을 지정
- 스프링에서는 다양한 어노테이션과 기능을 통해서 REST방식의 서비스를 간편하게 구축할 수 있음

어노테이션	기능
@RestController	Controller가 REST 방식을 처리하기 위한 것임을 명시합니다.
@ResponseBody	일반적인 JSP와 같은 뷰로 전달되는 게 아니라 데이터 자체를 전달하기 위한 용도
@PathVariable	URL 경로에 있는 값을 파라미터로 추출하려고 할 때 사용
@CrossOrigin	Ajax의 크로스 도메인 문제를 해결해주는 어노테이션
@RequestBody	JSON 데이터를 원하는 타입으로 바인딩 처리

# @RestController

- 스프링 4에서부터는 @Controller 외에 @RestController라는 어노테이션을 추가해서 해당 Controller의 모든 메서드의 리턴 타입을 기존과 다르게 처리한다는 것을 명시
- @RestController는 메서드의 리턴 타입으로 사용자가 정의한 클래스 타입을 사용할 수 있고, 이를 JSON이나 XML로 자동으로 처리

# 예제프로젝트의 준비

- 데이터의 처리는 XML과 JSON을 이용할 것이므로 pom.xml을 변경
- jackson-databind와 Jackson-dataformat-xml 라이브러리 활용
- Java 객체를 JSON으로 쉽게 변환할 수 있는 gson 라이브러리

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.6</version>
</dependency>

<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.2</version>
</dependency>
```

# @RestController의 반환 타입

- @RestController를 사용하는 컨트롤러에서는 다음과 같은 반환 타입들을 사용한다.
  - String 혹은 Integer 등의 타입들
  - 사용자 정의 타입
  - ResponseEntity<> 타입
- 주로 ResponseEntity 타입을 이용하는 것이 일반적

# SampleVO 클래스와 SampleController

- JSON 혹은 XML로 변환될 데이터

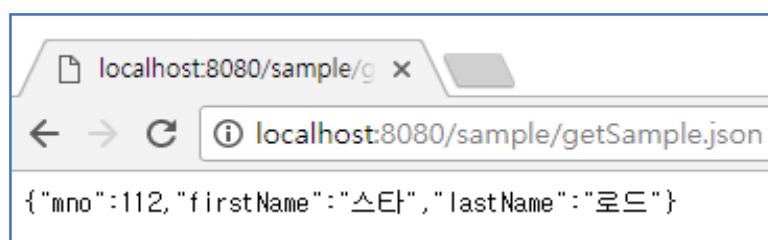
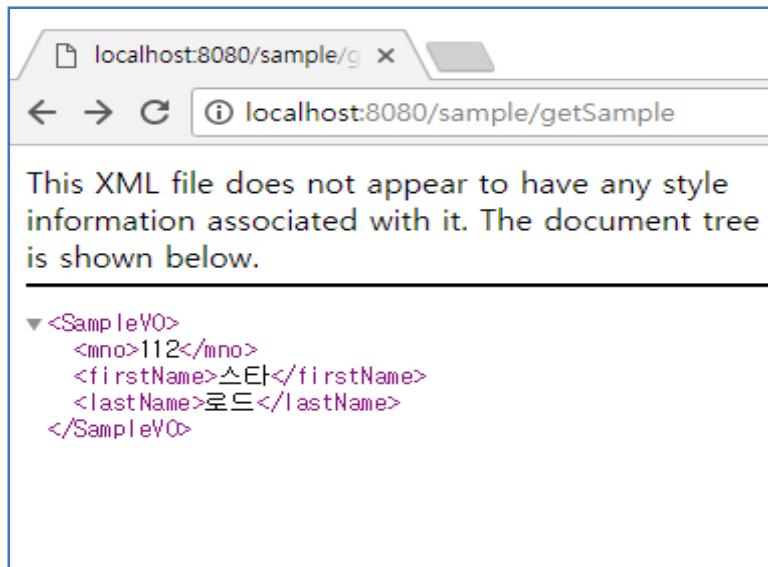
```
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class SampleVO {  
  
    private Integer mno;  
    private String firstName;  
    private String lastName;  
}
```

- SampleVO를 서비스하는 SampleController

```
@RestController  
@RequestMapping("/sample")  
@Log4j  
public class SampleController {  
  
}
```

# JSON/XML의 테스트

```
@GetMapping(value = "/getSample", produces = { MediaType.APPLICATION_JSON_UTF8_VALUE,  
    MediaType.APPLICATION_XML_VALUE })  
  
public SampleVO getSample() {  
  
    return new SampleVO(112, "스타", "로드");  
  
}
```



확장자에 따라 다른 타입으로 서비스

# Collection타입의 객체 반환-List

```
@GetMapping(value = "/getList")  
public List<SampleVO> getList() {  
  
    return IntStream.range(1, 10).mapToObj(i -> new SampleVO(i, i + "First", i + " Last"))  
        .collect(Collectors.toList());  
  
}
```

The image displays two browser windows side-by-side, both showing the URL <http://localhost:8080/sample/getList>.

**Left Browser Window (XML Response):**

This XML file does not appear to have any style information associated with it.

```
<List>
  <item>
    <mno>1</mno>
    <firstName>1First</firstName>
    <lastName>1 Last</lastName>
  </item>
  <item>
    <mno>2</mno>
    <firstName>2First</firstName>
    <lastName>2 Last</lastName>
  </item>
  <item>
    <mno>3</mno>
    <firstName>3First</firstName>
    <lastName>3 Last</lastName>
  </item>
  <item>
    <mno>4</mno>
    <firstName>4First</firstName>
    <lastName>4 Last</lastName>
  </item>
  <item>
    <mno>5</mno>
```

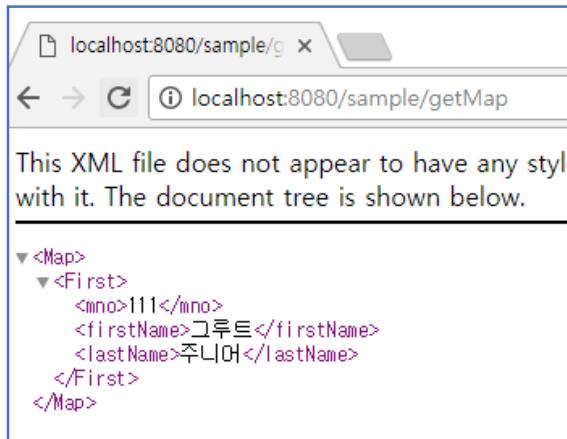
**Right Browser Window (JSON Response):**

```
[{"mno":1, "firstName":"1First", "lastName":"1 Last"}, {"mno":2, "firstName":"2First", "lastName":"2 Last"}, {"mno":3, "firstName":"3First", "lastName":"3 Last"}, {"mno":4, "firstName":"4First", "lastName":"4 Last"}, {"mno":5, "firstName":"5First", "lastName":"5 Last"}, {"mno":6, "firstName":"6First", "lastName":"6 Last"}, {"mno":7, "firstName":"7First", "lastName":"7 Last"}, {"mno":8, "firstName":"8First", "lastName":"8 Last"}, {"mno":9, "firstName":"9First", "lastName":"9 Last"}]
```

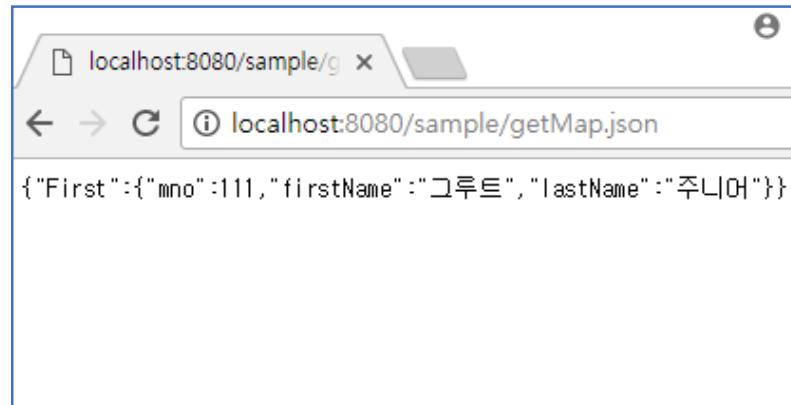
# Collection타입의 객체 반환-Map

```
@GetMapping(value = "/getMap")  
  
public Map<String, SampleVO> getMap() {  
  
    Map<String, SampleVO> map = new HashMap<>();  
    map.put("First", new SampleVO(111, "그루트", "주니어"));  
  
    return map;  
}
```

http://localhost:8080/sample/getMap



http://localhost:8080/sample/getMap.json



# ResponseEntity 탑

- 단순히 데이터뿐만 아니라 브라우저에 HTTP상태 코드등 추가적인 데이터를 전달할 수 있다는 장점

```
@GetMapping(value = "/check", params = { "height", "weight" })  
  
public ResponseEntity<SampleVO> check(Double height, Double weight) {  
  
    SampleVO vo = new SampleVO(000, "" + height, "" + weight);  
  
    ResponseEntity<SampleVO> result = null;  
  
    if (height < 150) {  
        result = ResponseEntity.status(HttpStatus.BAD_GATEWAY).body(vo);  
    } else {  
        result = ResponseEntity.status(HttpStatus.OK).body(vo);  
    }  
  
    return result;  
}
```

← → ⌂ ⓘ localhost:8080/sample/check.json?height=140&weight=60

```
{"mno":0,"firstName":"140.0","lastName":"60.0"}
```

Elements Console ⌂

Filter  Hide

20 ms

Name	Status
check.json?height=140&...	502

# @RestController의 파라미터

- `@PathVariable`: 일반 컨트롤러에서도 사용이 가능하지만 REST 방식에서 자주 사용됩니다. URL 경로의 일부를 파라미터로 사용할 때 이용
- `@RequestBody`: JSON 데이터를 원하는 타입의 객체로 변환해야 하는 경우에 주로 사용
- 일반 `<form>` 방식으로 처리된 데이터

# @PathVariable

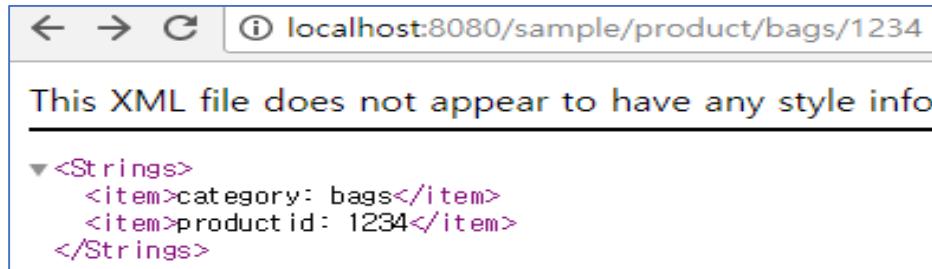
- URI경로 중간에 들어간 값을 얻기 위해서 사용

```
http://localhost:8080/sample/{sno}
```

```
http://localhost:8080/sample/{sno}/page/{pno}
```

```
@GetMapping("/product/{cat}/{pid}")
public String[] getPath(
    @PathVariable("cat") String cat,
    @PathVariable("pid") Integer pid) {

    return new String[] { "category: " + cat, "productid: " + pid };
}
```



# @RequestBody

- 전송된 데이터가 JSON이고, 이를 컨트롤러에서는 사용자 정의 타입의 객체로 변환할 때 사용

```
@Data  
public class Ticket {  
  
    private int tno;  
    private String owner;  
    private String grade;  
}  
  
@PostMapping("/ticket")  
public Ticket convert(@RequestBody Ticket ticket) {  
  
    Log.info("convert.....ticket" + ticket);  
  
    return ticket;  
}
```

일반적으로 브라우저에서는 JSON 형태의 데이터를 전송할 수 없으므로 별도의 REST 관련 도구를 이용해서 테스트를 진행해야 함

# REST 방식의 테스트

- JUnit기반의 테스트
- 별도의 프로그램이나 크롬 확장 프로그램등을 이용하는 테스트

```
@Test

public void testConvert() throws Exception {

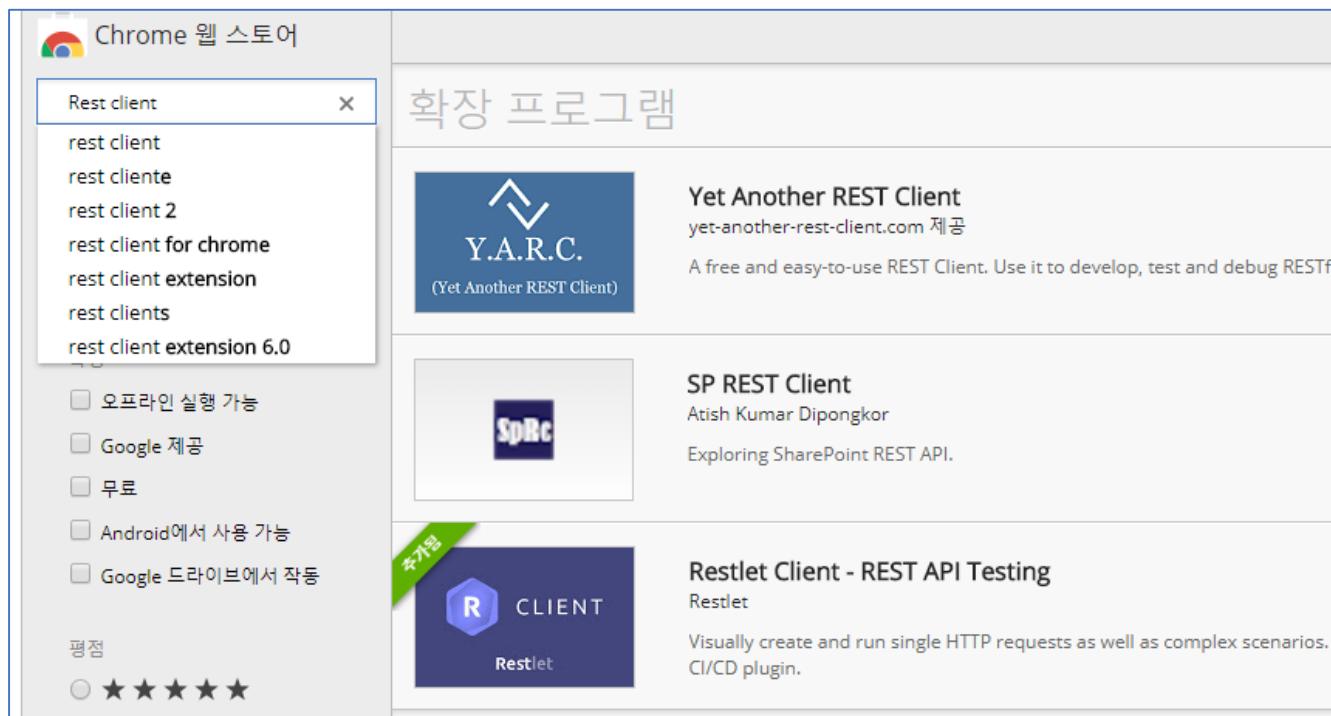
    Ticket ticket = new Ticket();
    ticket.setTno(123);
    ticket.setOwner("Admin");
    ticket.setGrade("AAA");
    //GSON 라이브러리를 이용해서 JSON 데이터로 변환
    String jsonStr = new Gson().toJson(ticket);

    Log.info(jsonStr);

    mockMvc.perform(post("/sample/ticket")
        .contentType(MediaType.APPLICATION_JSON)
        .content(jsonStr))
        .andExpect(status().is(200));
}
```

# 크롬 확장 프로그램을 이용하는 테스트

- 크롬 앱 스토어에서 'REST'방식 호출이 가능한 다양한 프로그램들이 존재
  - Yet Another REST Client
  - Restlet Client
  - etc



# Restlet을 이용한 테스트

DRAFT

METHOD: POST | SCHEME: http://HOST[:PORT][PATH/?QUERY]

http://localhost:8080/sample/ticket

Send | Save as | length: 35 bytes

QUERY PARAMETERS

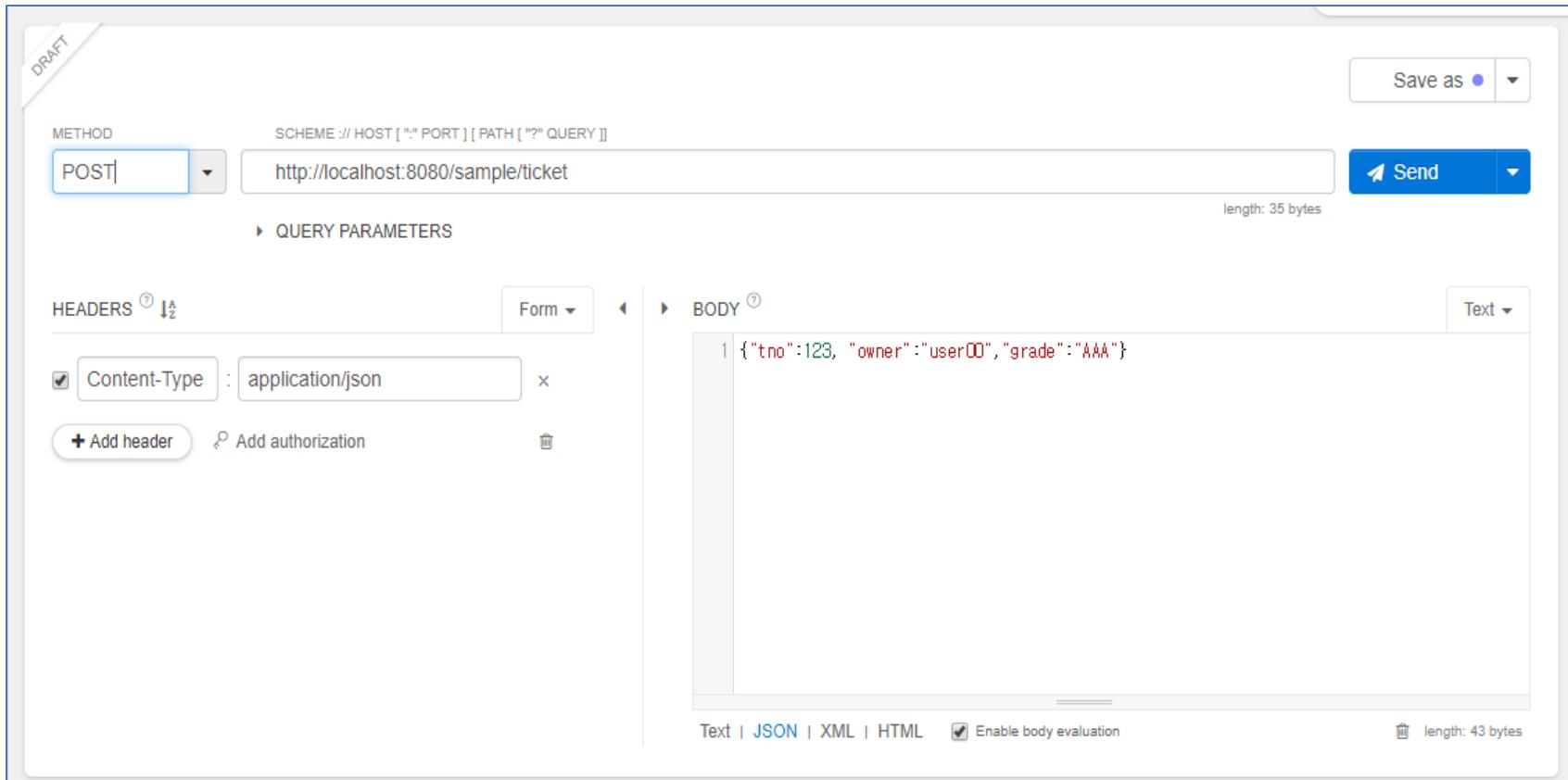
HEADERS: Content-Type: application/json

Add header | Add authorization | Remove

BODY: Text

```
1 {"tno":123, "owner":"user00", "grade":"AAA"}
```

Text | JSON | XML | HTML | Enable body evaluation | length: 43 bytes



# 다양한 전송방식과 URI설계

- REST 방식의 데이터 교환에서 가장 특이한 점은 기존의 GET/POST 외에 다양한 방식으로 데이터를 전달한다는 점

작업	전송방식
Create	POST
Read	GET
Update	PUT
Delete	DELETE

작업	전송방식	URI
등록	POST	/members/new
조회	GET	/members/{id}
수정	PUT	/members/{id} + body (json 데이터 등)
삭제	DELETE	/member/{id}

# 17. Ajax를 이용하는 댓글 처리

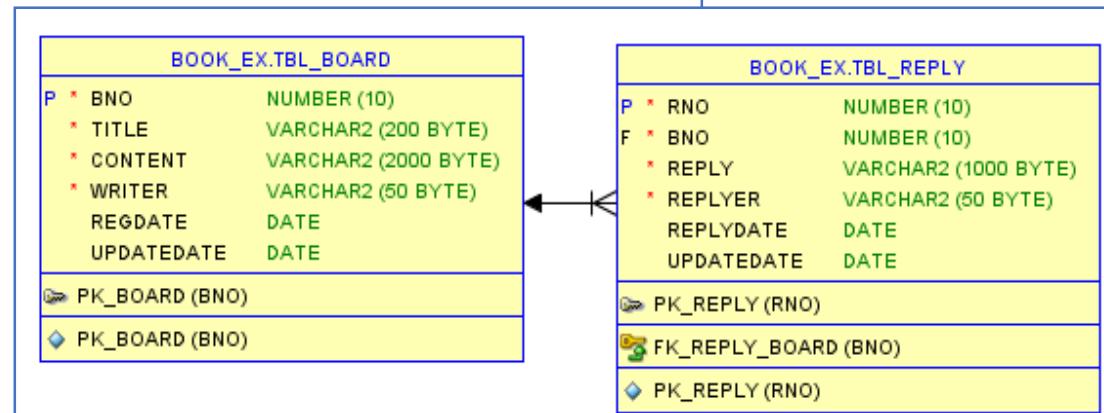
# 댓글 처리를 위한 테이블 설계

```
create table tbl_reply (
    rno number(10,0),
    bno number(10,0) not null,
    reply varchar2(1000) not null,
    replyer varchar2(50) not null,
    replyDate date default sysdate,
    updateDate date default sysdate
);
```

```
create sequence seq_reply;
```

```
alter table tbl_reply add constraint pk_reply primary key (rno);
```

```
alter table tbl_reply add constraint fk_reply_board
foreign key (bno) references tbl_board (bno);
```



식별키(PK) 지정

외래키(FK) 지정

# ReplyVO클래스의 추가/Mapper 준비

```
@Data  
public class ReplyVO {  
  
    private Long rno;  
    private Long bno;  
  
    private String reply;  
    private String replyer;  
    private Date replyDate;  
    private Date updateDate;  
  
}
```

```
public interface ReplyMapper {  
  
}
```

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

# Mapper CRUD작업-create

- 외래키가 걸려있으므로 실제 존재하는 게시물 번호를 이용해서 테스트를 진행해야 함

```
public interface ReplyMapper {  
  
    public int insert(ReplyVO vo);  
}
```

```
<mapper namespace="org.zerock.mapper.ReplyMapper">  
  
    <insert id="insert">  
  
        insert into tbl_reply (rno, bno, reply, replyer)  
        values (seq_reply.nextval, #{bno}, #{reply}, #{replyer})  
  
    </insert>  
  
</mapper>
```

# Mapper 특정 댓글 조회

- 댓글의 번호를 이용해서 특정 댓글 조회

```
public interface ReplyMapper {  
  
    public int insert(ReplyVO vo);  
  
    public ReplyVO read(Long bno);  
}
```

```
<select id="read" resultType="org.zerock.domain.ReplyVO">  
  
    select * from tbl_reply where rno = #{rno}  
  
</select>
```

# Mapper 특정 댓글 삭제/수정

- 댓글의 번호로 삭제

```
public interface ReplyMapper {  
    ...생략...  
    public int delete(Long bno);  
}
```

```
<delete id="delete">  
  
    delete from tbl_reply where rno = #{rno}  
  
</delete>
```

- 특정 댓글 수정

```
public interface ReplyMapper {  
    ...  
    public int update(ReplyVO reply);  
}
```

```
<update id="update">  
  
    update tbl_reply set reply = #{reply}, updatedate =  
    sysdate where rno =  
    #{rno}  
  
</update>
```

# @Param 어노테이션과 댓글 목록

- MyBatis의 파라미터는 1개만 허용
- 이를 해결하기 위해 1)Map형태를 사용하거나, 2)별도의 클래스를 이용하거나, 3)@Param을 이용할 수 있음

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

```
<select id="getListWithPaging"  
    resultType="org.zerock.domain.ReplyVO">  
  
    select rno, bno, reply, replyer, replyDate, updatedate  
    from tbl_reply  
    where bno = #{bno}  
    order by rno asc  
  
</select>
```

# 서비스영역과 컨트롤러 처리

- ReplyService 인터페이스와 ReplyServiceImpl 클래스 생성

```
public interface ReplyService {  
  
    public int register(ReplyVO vo);  
  
    public ReplyVO get(Long rno);  
  
    public int modify(ReplyVO vo);  
  
    public int remove(Long rno);  
  
    public List<ReplyVO> getList(Criteria cri, Long bno);  
  
}
```

```
@Service  
@Log4j  
public class ReplyServiceImpl implements ReplyService {  
  
    @Setter(onMethod_ = @Autowired)  
    private ReplyMapper mapper;  
  
    @Override  
    public int register(ReplyVO vo) {  
  
        Log.info("register....." + vo);  
  
        return mapper.insert(vo);  
    }  
..생략...
```

# ReplyController의 설계

작업	URL	HTTP 전송방식
등록	/replies/new	POST
조회	/replies/:rno	GET
삭제	/replies/:rno	DELETE
수정	/replies/:rno	PUT or PATCH
페이지	/replies/pages/:bno/:page	GET

```
@RequestMapping("/replies/")
@RestController
@Log4j
@AllArgsConstructor
public class ReplyController {

    private ReplyService service;

}
```

# 등록작업과 테스트

- 댓글 등록의 경우 브라우저에서는 JSON 타입으로 된 댓글 데이터를 전송하고, 서버에서는 댓글의 처리 결과가 정상적으로 되었는지 문자열로 결과를 알려 주는 방식으로 처리

```
@PostMapping(value = "/new",
    consumes = "application/json",
    produces = { MediaType.TEXT_PLAIN_VALUE })

public ResponseEntity<String> create(@RequestBody ReplyVO vo) {

    log.info("ReplyVO: " + vo);

    int insertCount = service.register(vo);

    log.info("Reply INSERT COUNT: " + insertCount);

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

DRAFT

METHOD  
POST

SCHEME :// HOST [ ":" PORT ] [ PATH [ "?" QUERY ] ]  
http://localhost:8080/replies/new

Save as ● ▾

Send

QUERY PARAMETERS

HEADERS ⚠ Form ▾

Content-Type : application/json

+ Add header    Add authorization

BODY ⚠ Text ▾

1 { "bno":3145745, "reply": "Hello Reply", "replyer": "user00" }

존재하는 게시물 번호를 사용

JSON타입으로 지정

Text | **JSON** | XML | HTML     Enable body evaluation

length: 56 bytes

The screenshot shows a REST client interface with a red box highlighting the 'Content-Type' header field in the 'HEADERS' section. A blue box labeled '존재하는 게시물 번호를 사용' (Use existing post ID) points to the 'bno' value in the JSON body. Another blue box labeled 'JSON타입으로 지정' (Specify as JSON type) points to the 'JSON' tab at the bottom of the body editor.

# 특정 게시물의 댓글 목록

```
@GetMapping(value = "/pages/{bno}/{page}",
    produces = {
        MediaType.APPLICATION_XML_VALUE,
        MediaType.APPLICATION_JSON_UTF8_VALUE })
public ResponseEntity<List<ReplyVO>> getList(
    @PathVariable("page") int page,
    @PathVariable("bno") Long bno) {

    Log.info("getList.....");
    Criteria cri = new Criteria(page,10);
    Log.info(cri);

    return new ResponseEntity<>(service.getList(cri, bno), HttpStatus.OK);
}
```

XML과 JSON타입으로 서비스

This XML file does not appear to have any style information associated

```
▼<ReplyPageDTO>
  <replyCnt>8</replyCnt>
  ▼<list>
    ▼<list>
      <rno>5</rno>
      <bno>3145745</bno>
      <reply>댓글 테스트 5</reply>
      <replier>replier5</replier>
      <replyDate>1534550815000</replyDate>
      <updateDate>1534550815000</updateDate>
    </list>
    ▼<list>
      <rno>10</rno>
      <bno>3145745</bno>
      <reply>댓글 테스트 10</reply>
      <replier>replier10</replier>
      <replyDate>1534550815000</replyDate>
      <updateDate>1534550815000</updateDate>
    </list>
    ▼<list>
      <rno>11</rno>
      <bno>3145745</bno>
      <reply>Hello Reply</reply>
      <replier>user00</replier>
      <replyDate>1534552080000</replyDate>
      <updateDate>1534552080000</updateDate>
    </list>
    ▼<list>
      <rno>12</rno>
      <bno>3145745</bno>
      <reply>Hello Reply</reply>
      <replier>user00</replier>
      <replyDate>1534552193000</replyDate>
      <updateDate>1534552193000</updateDate>
    </list>
```

# 댓글의 삭제/조회

```
@GetMapping(value = "/{rno}",
    produces = { MediaType.APPLICATION_XML_VALUE, MediaType.APPLICATION_JSON_UTF8_VALUE })
public ResponseEntity<ReplyVO> get(@PathVariable("rno") Long rno) {

    log.info("get: " + rno);

    return new ResponseEntity<>(service.get(rno), HttpStatus.OK);
}

@DeleteMapping(value= "/{rno}" ,produces = { MediaType.TEXT_PLAIN_VALUE })
public ResponseEntity<String> remove(@PathVariable("rno") Long rno) {

    log.info("remove: " + rno);

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

# 댓글의 수정

```
@RequestMapping(method = { RequestMethod.PUT, RequestMethod.PATCH },
    value = "/{rno}",
    consumes = "application/json",
    produces = { MediaType.TEXT_PLAIN_VALUE })

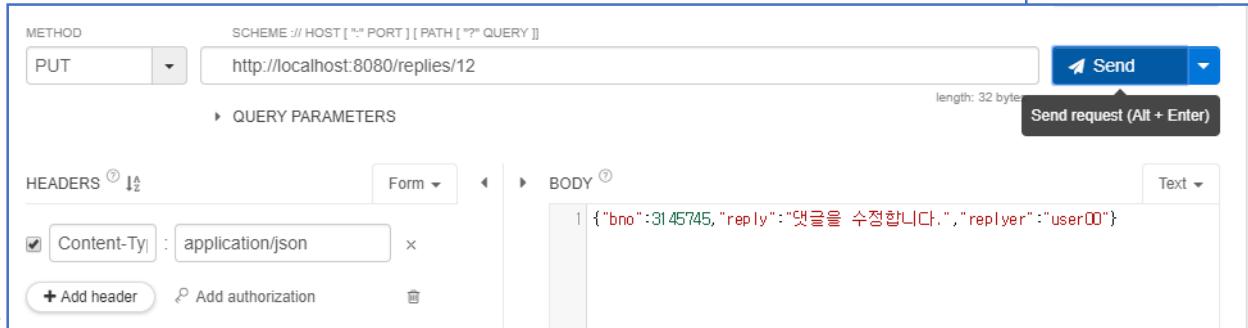
public ResponseEntity<String> modify(@RequestBody ReplyVO vo, @PathVariable("rno") Long rno) {

    vo.setRno(rno);

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

    return service.modify(vo) == 1
        : new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);

}
```



# JavaScript의 준비 - 모듈화

- JS의 모듈 패턴
  - 여러 기능들을 모아서 하나의 모듈화
  - 클로저를 이용해서 상태 유지
  - 여러 함수들이 메서드화 되므로 객체지향 구조에 적합

reply.js

```
console.log("Reply Module.....");

var replyService = (function(){

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

    return {add:add};
})();
```

# reply.js 댓글 등록

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

Ajax처리후 동작해야 하는 함수

Ajax로 ReplyController호출

모듈 패턴으로 외부에 노출하는 정보

# 조회 화면에서 호출

```
script type="text/javascript" src="/resources/js/reply.js"></script>

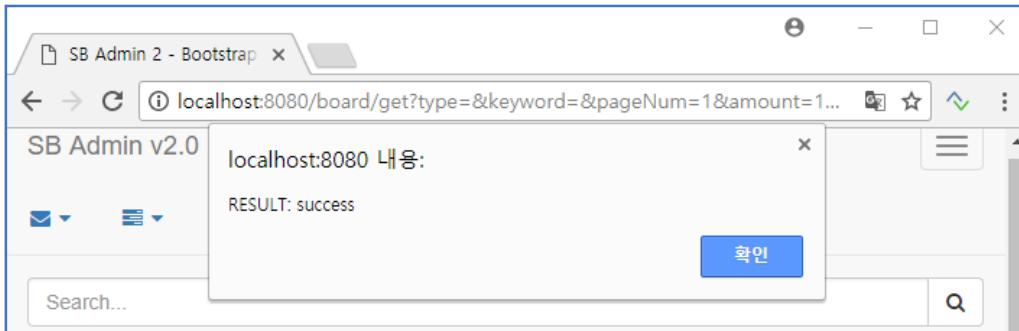
<script>

console.log("=====");
console.log("JS TEST");

var bnoValue = '<c:out value="${board.bno}" />';

//for replyService add test
replyService.add(
    {reply:"JS Test", replyer:"tester", bno:bnoValue} ,
    function(result){
        alert("RESULT: " + result);
    }
);

</script>
```



## JSON으로 처리되는지 확인

A screenshot of a browser developer tools Network tab. A request labeled "new" is selected. The Headers section shows the following:

```
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection: keep-alive
Content-Length: 52
Content-Type: application/json; charset=UTF-8
Cookie: _ga=GA1.1.1053734745.1513683555; JSESSIONID=11809FA874BBCC4935DCA90A164EABB
Host: localhost:8080
Origin: http://localhost:8080
Referer: http://localhost:8080/board/get?type=&keyword=&pageNum=1&amount=10&bno=3670051
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
239.132 Safari/537.36
X-Requested-With: XMLHttpRequest
```

The Request Payload section shows the JSON response:

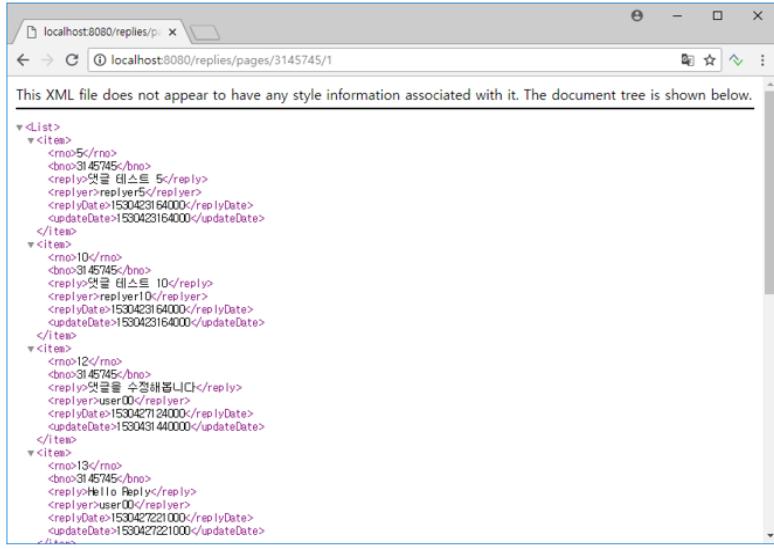
```
{reply: "JS Test", replyer: "tester", bno: 3670051}
  bno: 3670051
  reply: "JS Test"
  replyer: "tester"
```

At the bottom of the Network tab, it says "11 requests | 22.5 KB transferred ...".

# 댓글의 목록 처리

댓글의 처리 전에 우선적으로 확인한 후에 진행

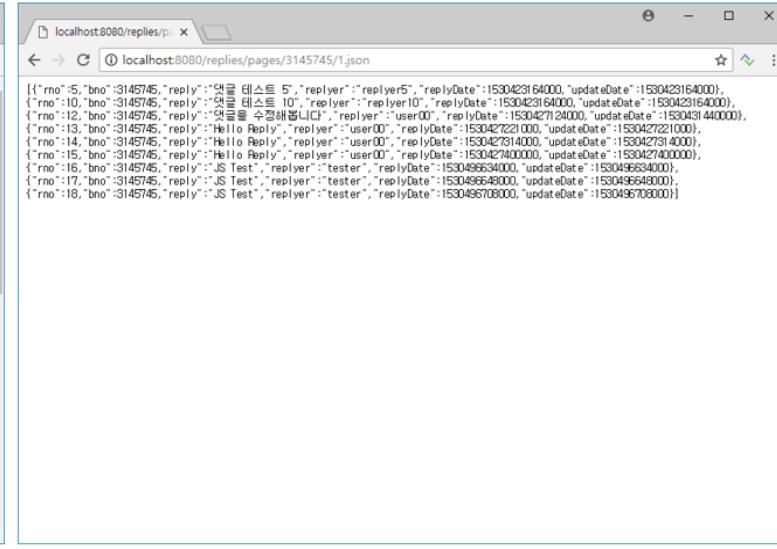
<http://localhost:8080/replies/pages/3145745/1>



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<List>
  <item>
    <bno>5</bno>
    <reply>댓글 테스트 5</reply>
    <replyer>replyer5</replyer>
    <replyDate>1530423164000</replyDate>
    <updateDate>1530423164000</updateDate>
  </item>
  <item>
    <bno>10</bno>
    <reply>댓글 테스트 10</reply>
    <replyer>replyer10</replyer>
    <replyDate>1530423164000</replyDate>
    <updateDate>1530423164000</updateDate>
  </item>
  <item>
    <bno>12</bno>
    <reply>댓글을 수정했습니다</reply>
    <replyer>user00</replyer>
    <replyDate>1530427124000</replyDate>
    <updateDate>153043144000</updateDate>
  </item>
  <item>
    <bno>13</bno>
    <reply>Hello Reply</reply>
    <replyer>user00</replyer>
    <replyDate>1530427221000</replyDate>
    <updateDate>1530427221000</updateDate>
  </item>
```

<http://localhost:8080/replies/pages/3145745/1.json>



```
[{"rno":5,"bno":"3145745","reply":"댓글 테스트 5","replyer":"replyer5","replyDate":1530423164000,"updateDate":1530423164000}, {"rno":10,"bno":"3145745","reply":"댓글 테스트 10","replyer":"replyer10","replyDate":1530423164000,"updateDate":1530423164000}, {"rno":12,"bno":"3145745","reply":"댓글을 수정했습니다","replyer":"user00","replyDate":1530427124000,"updateDate":153043144000}, {"rno":13,"bno":"3145745","reply":"Hello Reply","replyer":"user00","replyDate":1530427221000,"updateDate":1530427221000}, {"rno":14,"bno":"3145745","reply":"Hello Reply","replyer":"user00","replyDate":1530427314000,"updateDate":1530427314000}, {"rno":15,"bno":"3145745","reply":"Hello Reply","replyer":"user00","replyDate":153042740000,"updateDate":153042740000}, {"rno":16,"bno":"3145745","reply":"JS Test","replyer":"tester","replyDate":1530496634000,"updateDate":1530496634000}, {"rno":17,"bno":"3145745","reply":"JS Test","replyer":"tester","replyDate":1530496648000,"updateDate":1530496648000}, {"rno":18,"bno":"3145745","reply":"JS Test","replyer":"tester","replyDate":1530496708000,"updateDate":1530496708000}]
```

# getJSON( )처리 –reply.js

```
function getList(param, callback, error) {  
  
    var bno = param.bno;  
    var page = param.page || 1;  
  
    $.getJSON("/replies/pages/" + bno + "/" + page + ".json",  
        function(data) {  
            if (callback) {  
                callback(data);  
            }  
        }).fail(function(xhr, status, err) {  
            if (error) {  
                error();  
            }  
        });  
}
```

Ajax로 ReplyController호출

## 화면에서의 호출 테스트

```
replyService.getList({bno:bnoValue, page:1}, function(list){  
  
    for(var i = 0, len = list.length||0; i < len; i++ ){  
        console.log(list[i]);  
    }  
});
```

```
▶ {rno: 5, bno: 2359299, reply: "댓글 테스트 5", replyer: "replyer5", replyDate: 1523635917000, ...} get?page  
▶ {rno: 10, bno: 2359299, reply: "Update Reply ", replyer: "replyer10", replyDate: 1523635917000, ...} get?page  
▶ {rno: 21, bno: 2359299, reply: "JS Test", replyer: "tester", replyDate: 1523715248000, ...} get?page  
▶ {rno: 22, bno: 2359299, reply: "JS Test", replyer: "tester", replyDate: 1523717132000, ...} get?page  
▶ {rno: 23, bno: 2359299, reply: "JS Test", replyer: "tester", replyDate: 1523717136000, ...} get?page
```

# 댓글의 삭제와 갱신

## reply.js의 일부

```
function remove(rno, callback, error) {
    $.ajax({
        type : 'delete',          전송방식은 DELETE 방식 사용
        url : '/replies/' + rno,
        success : function(deleteResult, status, xhr) {
            if (callback) {
                callback(deleteResult);
            }
        },
        error : function(xhr, status, er) {
            if (error) {
                error(er);
            }
        }
    });
}
```

# 댓글 삭제 테스트

- 존재하는 댓글의 번호를 이용해서 처리

```
//23번 댓글 삭제 테스트  
replyService.remove(23, function(count) {  
  
    console.log(count);  
  
    if (count === "success") {  
        alert("REMOVED");  
    }  
}, function(err) {  
    alert('ERROR...');  
});
```

삭제전

RN	RNO	REPLY	REPLIER	REPLYDATE	UPDATEDATE
1	5	댓글 테스트 5	replyer5	18/04/14	18/04/14
2	10	Update Reply	replyer10	18/04/14	18/04/14
3	21	JS Test	tester	18/04/14	18/04/14
4	22	JS Test	tester	18/04/14	18/04/14
5	23	JS Test	tester	18/04/14	18/04/14

삭제후

RN	RNO	REPLY	REPLIER	REPLYDATE	UPDATEDATE
1	5	댓글 테스트 5	replyer5	18/04/14	18/04/14
2	10	Update Reply	replyer10	18/04/14	18/04/14
3	21	JS Test	tester	18/04/14	18/04/14
4	22	JS Test	tester	18/04/14	18/04/14

# 댓글 수정/테스트

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

PUT방식으로 호출  
전달하는 데이터는 JSON데이터

```
//22번 댓글 수정  
replyService.update({  
    rno : 22,  
    bno : bnoValue,  
    reply : "Modified Reply...."  
}, function(result) {  
  
    alert("수정 완료...");  
});
```

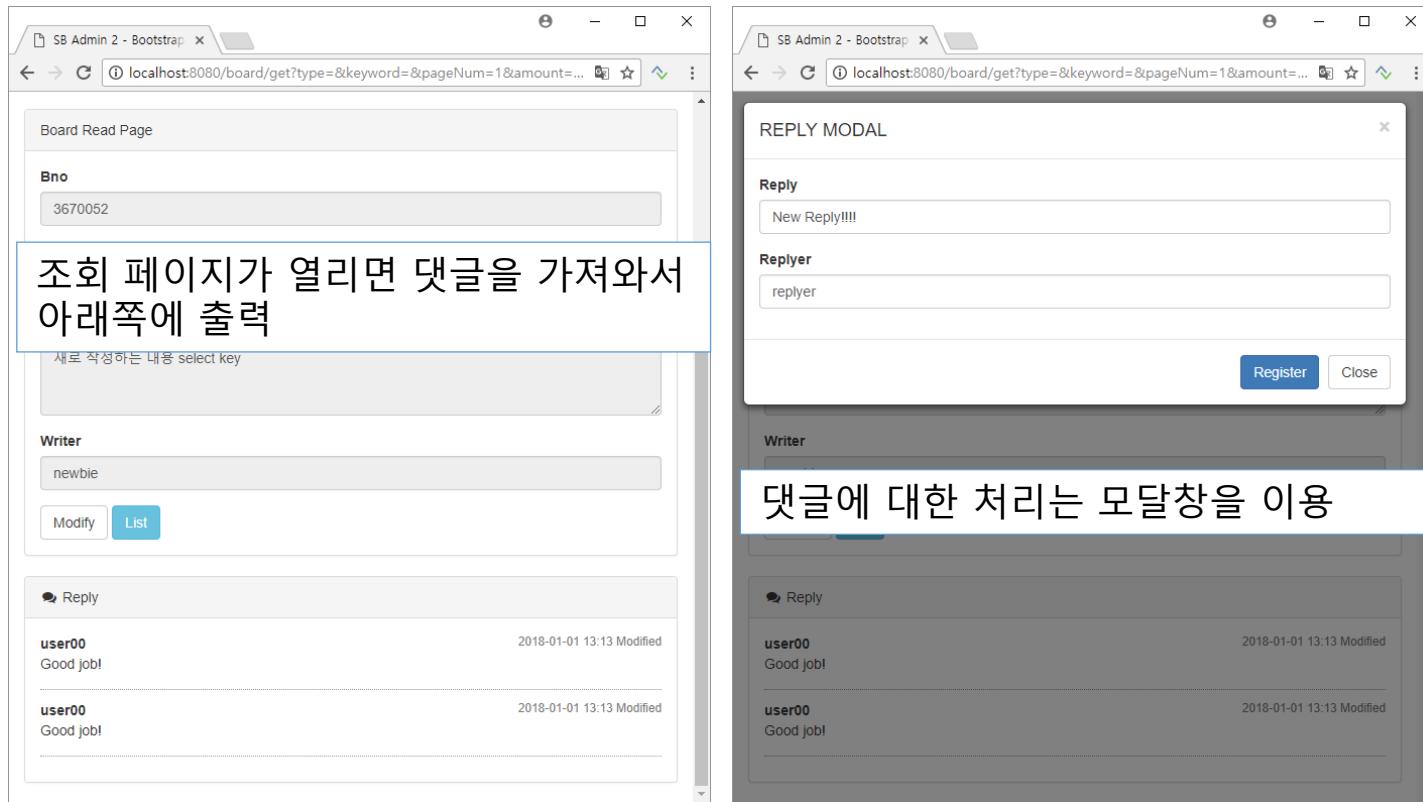
# 특정댓글조회/테스트

```
function get(rno, callback, error) {  
  
    $.get("/replies/" + rno + ".json", function(result) {  
  
        if (callback) {  
            callback(result);  
        }  
  
    }).fail(function(xhr, status, err) {  
        if (error) {  
            error();  
        }  
    });  
}
```

```
replyService.get(10, function(data){  
    console.log(data);  
});
```

# 이벤트 처리와 HTML처리

- Ajax호출 이벤트 처리와 후 처리



# 댓글 목록의 처리

- 댓글의 목록은 <ul> 태그 내에 <li> 태그를 이용해서 처리
- <li> 태그는 하나의 댓글을 의미
- 수정이나 삭제 시에는 반드시 댓글의 번호(rno)가 필요하므로 'data-rno' 속성을 이용

The screenshot shows a list of comments in a web application. At the top left is a 'Reply' button with a microphone icon, and at the top right is a 'New Reply' button. The comments are listed in reverse chronological order:

- replyer5** 댓글 테스트 5 2018/04/13
- replyer10** Update Reply 01:11:57
- tester** JS Test 23:14:08
- tester** Modified Reply.... 23:45:32
- Tester** Test 00:52:44

```

function showList(page){

    replyService.getList({bno:bnoValue,page: page|| 1 }, function(list) {

        var str="";
        if(list == null || list.length == 0){

            replyUL.html("");

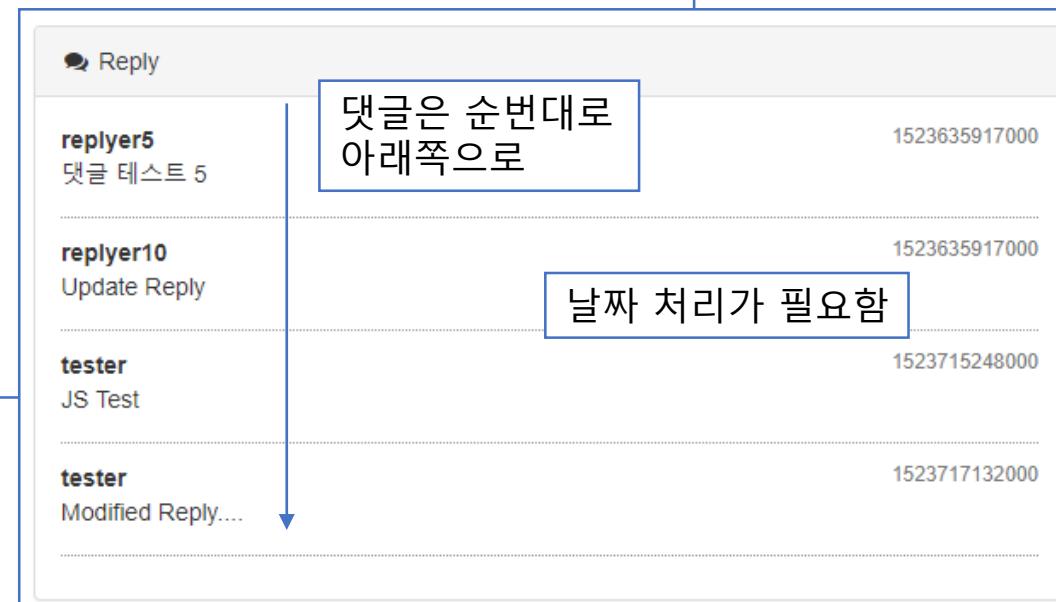
            return;
        }

        for (var i = 0, len = list.length || 0; i < len; i++) {
            str += "<li class='left clearfix' data-rno='"+list[i].rno+">";
            str += " <div><div class='header'><strong class='primary-font'>" +list[i].replyer+"</strong>";
            str += " <small class='pull-right text-muted'>" +list[i].replyDate+"</small></div>";
            str += " <p>" +list[i].reply+"</p></div></li>";
        }

        replyUL.html(str);
    });
}

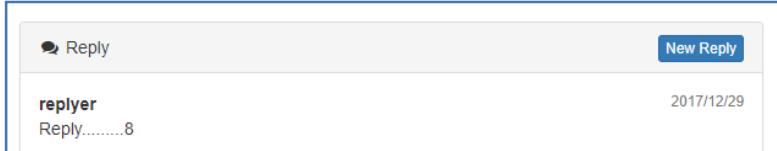
```

showList(페이지번호)는 해당 게시글의 댓글을 가져온 후 <li>태그를 만들어서 화면에 보여준다.



# 새로운 댓글의 처리

- 모달창을 이용해서 댓글 추가



The screenshot shows a web browser window with two modal dialogs open over a list of comments. The left modal is titled 'REPLY MODAL' and contains fields for 'Reply' (text input), 'Replyer' (text input), and 'Reply Date' (date input). The right modal is also titled 'REPLY MODAL' and contains fields for 'Reply' (text input) and 'Replyer' (text input). Both modals have 'Register' and 'Close' buttons. The background shows a list of comments with columns for 'Replyer', 'Reply', and 'BNO'. The first comment in the list has a BNO of 3670052.

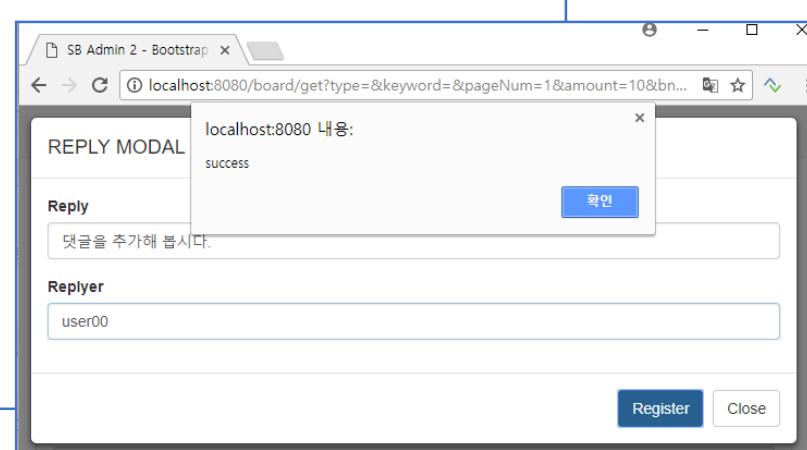
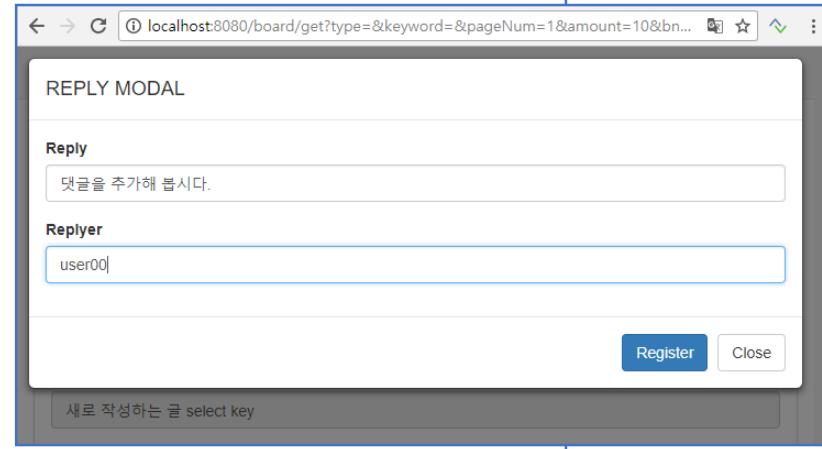
원래의 모달창

새로운 댓글 등록의 모달창

Replyer	Reply	BNO
replyer	Reply.....8	3670052
replyer	Reply.....4	
replyer	Reply.....8	
replyer	Reply.....12	

# 댓글 추가후 처리

```
modalRegisterBtn.on("click",function(e){  
  
    var reply = {  
        reply: modalInputReply.val(),  
        replyer:modalInputReplyer.val(),  
        bno:bnoValue  
    };  
  
    replyService.add(reply, function(result){  
  
        alert(result);  
  
        modal.find("input").val("");  
        modal.modal("hide");  
  
    });  
  
});
```



# 특정 댓글의 클릭 이벤트

- 댓글은 Ajax로 가져온 결과를 DOM에 추가하는 형태이므로 이벤트 위임(delegation) 방식을 이용해서 처리

A screenshot of a comment section from a website. At the top left is a 'Reply' button with a speech bubble icon. At the top right is a 'New Reply' button. Below these, there are two visible comments:

- replyer5** 댓글 테스트 5 2018/04/13 01:11:57
- replyer10** Update Reply 23:14:08

The second comment, by replyer10, has a red rectangular box drawn around its entire content area. Inside this box, the text "이벤트의 대상은 <ul>" is written in black.

Below the comments, there is a single line of text: "JS Test".

이벤트 위임으로 <li>가  
이벤트의 주인공

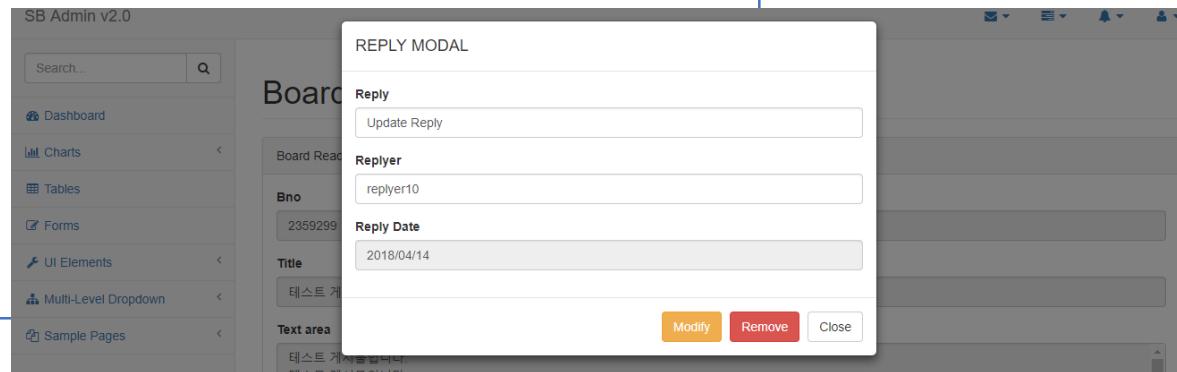
A screenshot of a list of numbers enclosed in a blue rectangular border. The numbers are:

- 5
- 10
- 21
- 22
- 24

The first number, '5', is highlighted with a blue background.

```
//댓글 조회 클릭 이벤트 처리
```

```
$(".chat").on("click", "li", function(e){  
  
    var rno = $(this).data("rno");  
  
    replyService.get(rno, function(reply){  
  
        modalInputReply.val(reply.reply);  
        modalInputReplyer.val(reply.replyer);  
        modalInputReplyDate.val(replyService.displayTime( reply.replyDate ))  
            .attr("readonly","readonly");  
        modal.data("rno", reply.rno);  
  
        modal.find("button[id !='modalCloseBtn']").hide();  
        modalModBtn.show();  
        modalRemoveBtn.show();  
  
        $(".modal").modal("show");  
    });  
});
```



# 댓글의 수정/삭제 처리 이벤트

- 수정/삭제 처리는 Ajax로 하고, 모달창 close
- 수정/삭제 후에는 다시 최신 댓글 목록 갱신

```
modalModBtn.on("click", function(e){  
  
    var reply = {rno:modal.data("rno"), reply: modalInputReply.val()};  
  
    replyService.update(reply, function(result){  
  
        alert(result);  
        modal.modal("hide");  
        showList(1);  
  
    });  
  
});
```

# 댓글의 페이지 처리

- 데이터베이스의 인덱스 설계
  - 모든 댓글의 조회는 게시물 번호를 이용해서 처리하므로, 게시물 번호의 정렬된 구조가 필요

IDX_REPLY			TBL_REPLY			
bno	rno	ROWID	ROWID	rno	bno	reply
300	200	AAAXXK	AAAXXA	1	100	xxx
200	3	AAAXXB	AAAXXE	22	200	xxx
200	13	AAAXXD	AAAXXD	13	200	xxx
200	22	AAAXXE	AAAXXB	3	200	xxx
200	43	AAAXXF	AAAXXF	43	200	xxx
200	89	AAAXXG	AAAXXC	5	100	xxx
200	100	AAAXXH	AAAXXG	89	200	xxx
100	1	AAAXXA	AAAXXH	100	200	xxx
100	5	AAAXXC	AAAXXK	200	300	xxx
100	123	AAAXXC	AAAXXI	123	100	xxx

# 인덱스를 이용하는 댓글 페이지 처리

```
select /*+INDEX(tbl_reply idx_reply) */  
    rownum rn, bno, rno, reply, replyer, replyDate, updatedate  
  from tbl_reply  
 where bno = 3145745(게시물 번호)  
   and rno > 0;
```



# 댓글의 수(카운트)

- 댓글 페이지를 위한 숫자 파악

```
<select id="getCountByBno" resultType="int">  
  <![CDATA[  
    select count(rno) from tbl_reply where bno = #{bno}  
  ]]>  
</select>
```

# ReplyServiceImpl 처리

- 댓글의 숫자와 댓글 목록을 처리하는 ReplyPageDTO 클래스

```
@Data  
@AllArgsConstructor  
@Getter  
public class ReplyPageDTO {  
  
    private int replyCnt;  
    private List<ReplyVO> list;  
}
```

```
public interface ReplyService {
```

...생략...

```
    public ReplyPageDTO getListPage(Criteria cri, Long bno);
```

```
}
```

```
public class ReplyServiceImpl implements ReplyService {
```

```
    @Setter(onMethod_ = @Autowired)
```

```
    private ReplyMapper mapper;
```

...생략...

```
    @Override
```

```
    public ReplyPageDTO getListPage(Criteria cri, Long bno)
    {
```

```
        return new ReplyPageDTO(
```

```
            mapper.getCountByBno(bno),
```

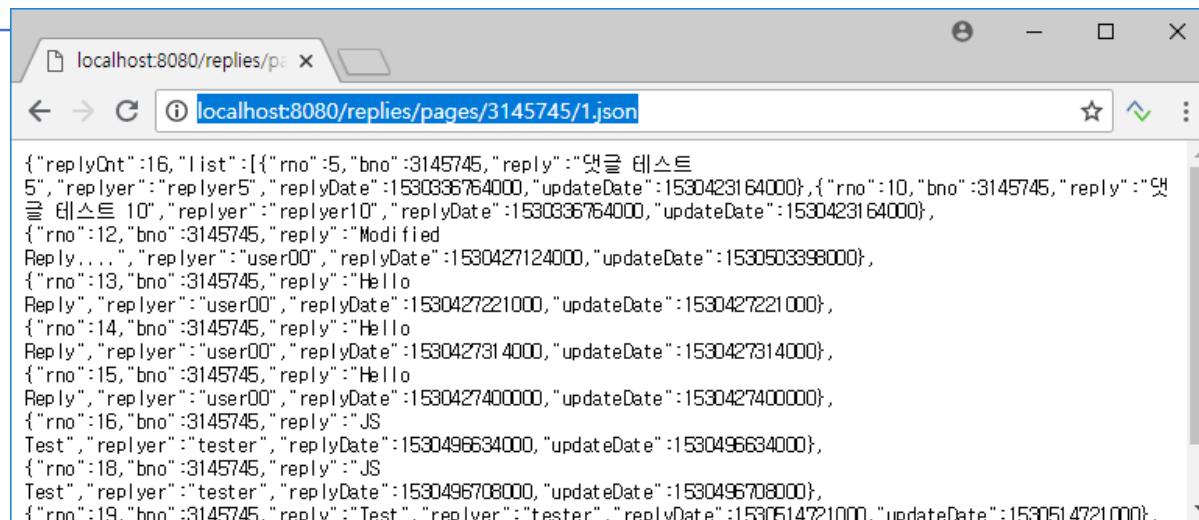
```
            mapper.getListWithPaging(cri, bno));
```

```
    }
```

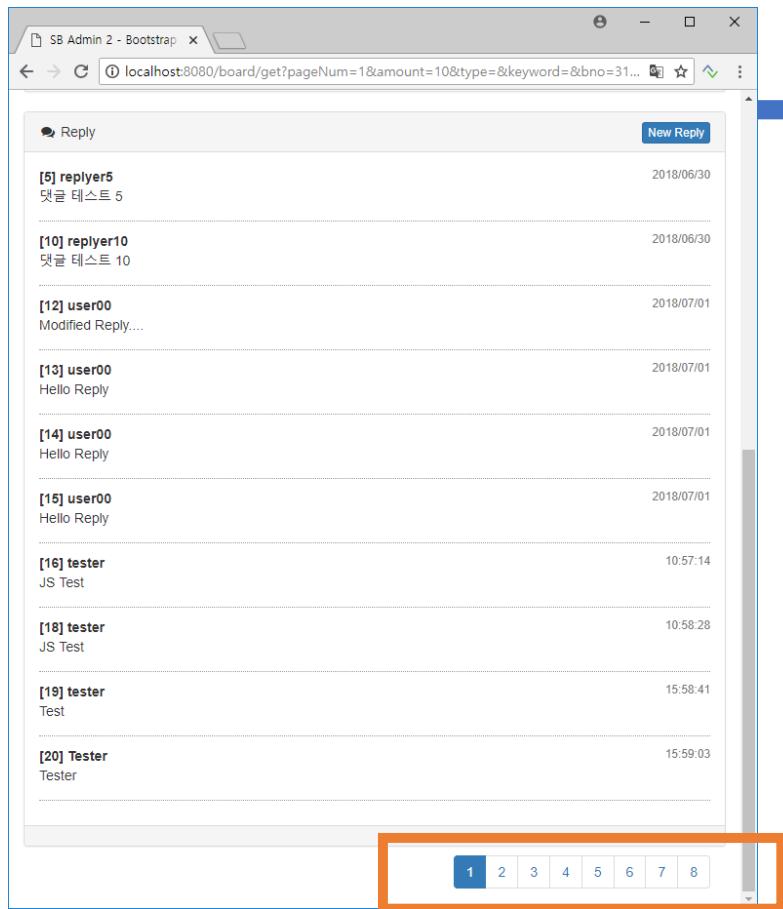
```
}
```

# ReplyController의 수정

```
@GetMapping(value = "/pages/{bno}/{page}",  
    produces = { MediaType.APPLICATION_XML_VALUE,  
        MediaType.APPLICATION_JSON_UTF8_VALUE })  
  
public ResponseEntity<ReplyPageDTO> getList(@PathVariable("page") int page, @PathVariable("bno") Long bno) {  
  
    Criteria cri = new Criteria(page, 10);  
  
    Log.info("get Reply List bno: " + bno);  
  
    Log.info("cri:" + cri);  
  
    return new ResponseEntity<>(service.getListPage(cri, bno), HttpStatus.OK);  
}
```

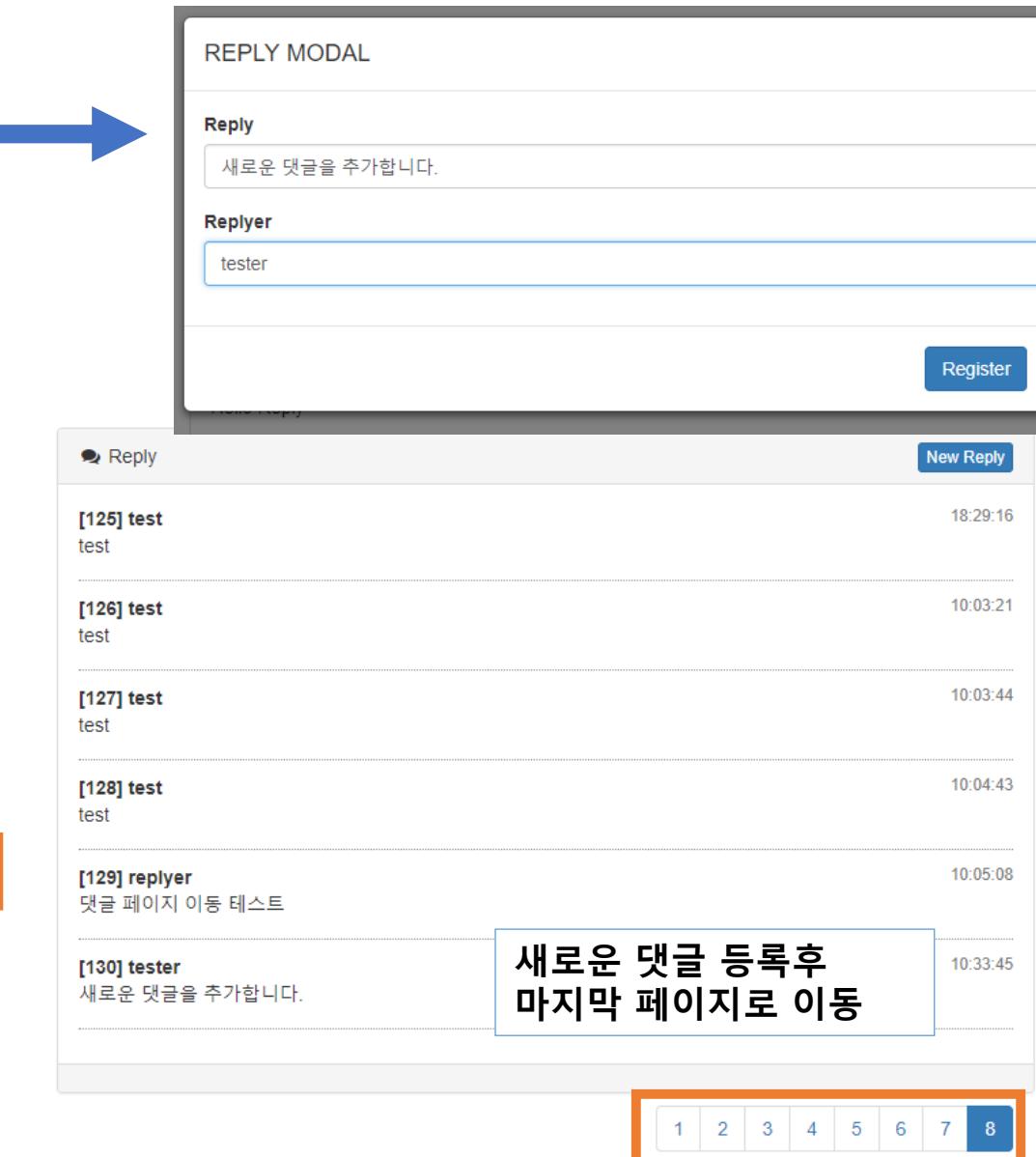


# 댓글의 화면 처리



Screenshot of a web browser displaying a list of comments. The browser title is "SB Admin 2 - Bootstrap". The URL is "localhost:8080/board/get?pageNum=1&amount=10&type=&keyword=&bno=31...". The page shows a list of comments with their IDs, posters, content, and timestamps. At the bottom, there is a navigation bar with page numbers 1 through 8, where page 1 is highlighted.

ID	Poster	Content	Date
[5]	replyer5	댓글 테스트 5	2018/06/30
[10]	replyer10	댓글 테스트 10	2018/06/30
[12]	user00	Modified Reply....	2018/07/01
[13]	user00	Hello Reply	2018/07/01
[14]	user00	Hello Reply	2018/07/01
[15]	user00	Hello Reply	2018/07/01
[16]	tester	JS Test	10:57:14
[18]	tester	JS Test	10:58:28
[19]	tester	Test	15:58:41
[20]	Tester	Tester	15:59:03



The screenshot shows a "REPLY MODAL" window with a "Reply" input field containing "새로운 댓글을 추가합니다." and a "Replier" input field containing "tester". Below the modal is a list of comments. A blue box highlights the text "새로운 댓글을 추가합니다." in the modal. Another blue box highlights the page number 8 in the pagination at the bottom of the comment list.

REPLY MODAL

Reply  
새로운 댓글을 추가합니다.

Replier  
tester

Register

New Reply

ID	Poster	Content	Date
[125]	test	test	18:29:16
[126]	test	test	10:03:21
[127]	test	test	10:03:44
[128]	test	test	10:04:43
[129]	replyer	댓글 페이지 이동 테스트	10:05:08
[130]	tester	새로운 댓글을 추가합니다.	10:33:45

새로운 댓글 등록후 마지막 페이지로 이동

# 댓글의 페이지 계산과 출력

- reply.js 는 게시물의 번호와 페이지를 사용하도록 수정

```
function getList(param, callback, error) {  
  
    var bno = param.bno;  
    var page = param.page || 1;  
  
    $.getJSON("/replies/pages/" + bno + "/" + page + ".json",  
        function(data) {  
            if (callback) {  
                //callback(data); // 댓글 목록만 가져오는 경우  
                callback(data.replyCnt, data.list); //댓글 숫자와 목록을 가져오는 경우  
            }  
        }).fail(function(xhr, status, err) {  
            if (error) {  
                error();  
            }  
        });  
    }  
}
```

# 새로운 댓글 추가

- 새로운 댓글을 추가하면 page값을 -1로 전송하고, 댓글의 전체 숫자를 파악한 후에 페이지 이동

```
modalRegisterBtn.on("click",function(e){  
    var reply = {  
        reply: modalInputReply.val(),  
        replyer:modalInputReplyer.val(),  
        bno:bnoValue  
    };  
    replyService.add(reply, function(result){  
        alert(result);  
        modal.find("input").val("");  
        modal.modal("hide");  
        //showList(1);  
        showList(-1);  
    });  
});
```

# 댓글의 페이지 번호 처리

```
<!-- /.panel-heading 기존에 존재하는 부분 -->
```

```
<div class="panel-body">
```

```
    <ul class="chat">  
        </ul>
```

```
</div>
```

```
<!-- /.panel .chat-panel 추가-->
```

```
<div class="panel-footer">
```

```
</div>
```



SB Admin 2 - Bootstrap

localhost:8080/board/get?pageNum=1&amount=10&type=&keyword=&bno=31...

Reply

New Reply

[5] replyer5	댓글 테스트 5	2018/06/30
[10] replyer10	댓글 테스트 10	2018/06/30
[12] user00	Modified Reply....	2018/07/01
[13] user00	Hello Reply	2018/07/01
[14] user00	Hello Reply	2018/07/01
[15] user00	Hello Reply	2018/07/01
[16] tester	JS Test	10:57:14
[18] tester	JS Test	10:58:28
[19] tester	Test	15:58:41
[20] Tester	Tester	15:59:03

1 2 3 4 5 6 7 8

REPLY MODAL

새로운 댓글을 추가합니다.

Replyer

tester

Register Close

Reply

New Reply

[125] test	test	18:29:16
[126] test	test	10:03:21
[127] test	test	10:03:44
[128] test	test	10:04:43
[129] replyer	댓글 페이지 이동 테스트	10:05:08
[130] tester	새로운 댓글을 추가합니다.	10:33:45

새로운 댓글 등록후  
마지막 페이지로 이동

1 2 3 4 5 6 7 8

# 스프링 웹 프로젝트

AOP

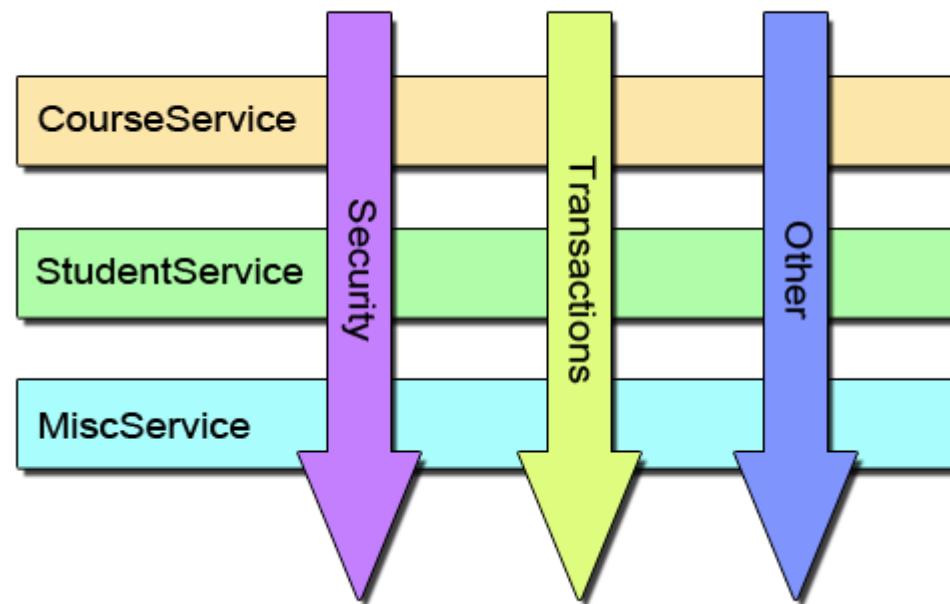
# Objectives

- AOP의 개념 이해와 적용
- AOP의 용어와 기법
- 트랜잭션의 처리 적용

# 18. AOP라는 패러다임

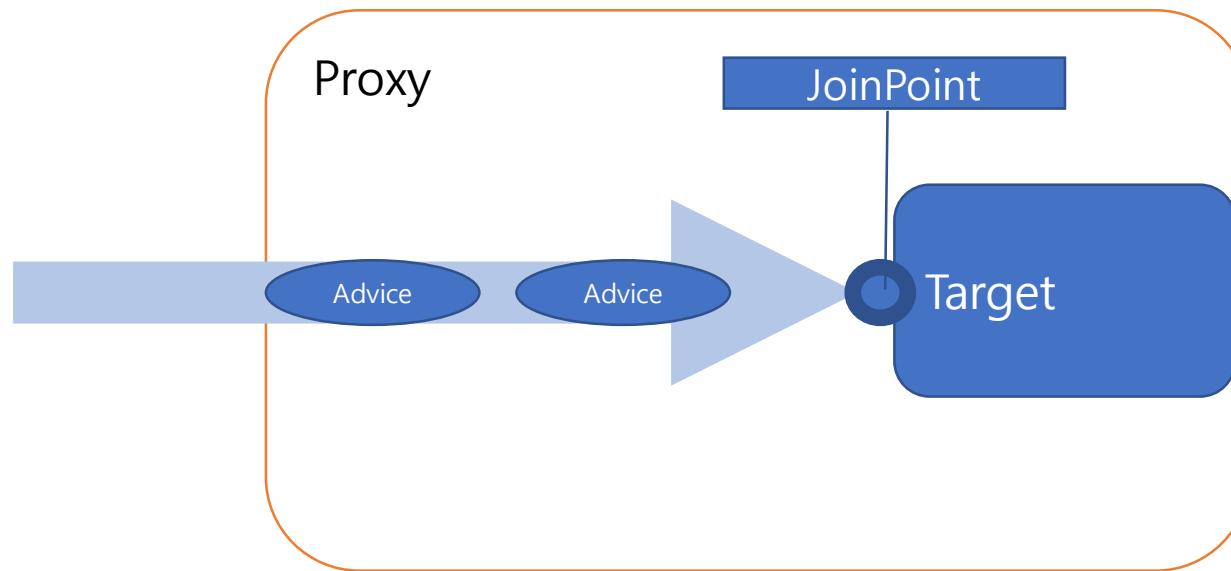
# 관점 지향 프로그래밍

- 관심사의 분리
  - 핵심로직은 아니지만 코드에 전반적으로 반복적으로 사용하며, 필요한 로직들
  - 횡단 관심사(cross-concerns)

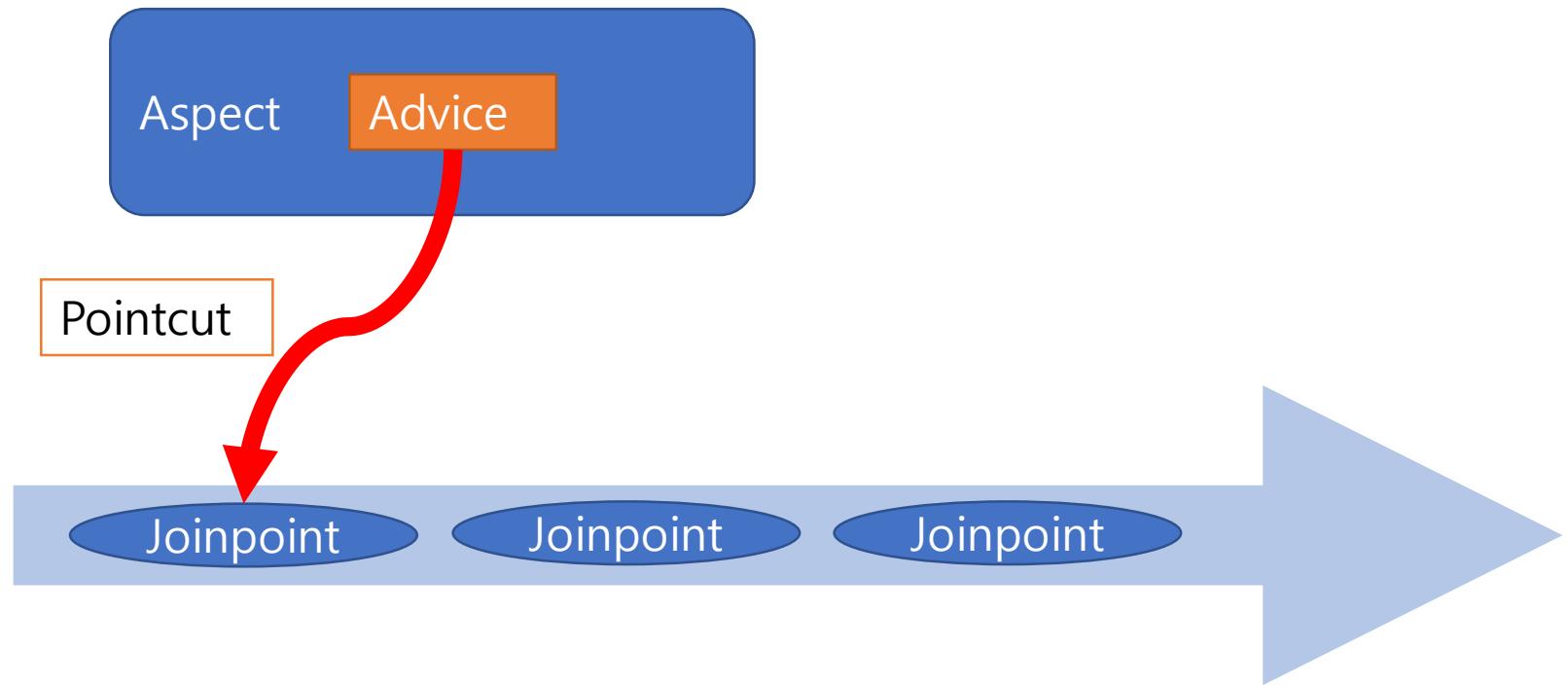


# 주요 용어

- Aspect: 추상 명사로 획단 관심사를 의미
  - ex> 로깅, 보안, 트랜잭션등
- Advice: 획단 관심사를 구현한 객체
- Target: 핵심로직을 가지고 있는 객체
- Proxy 객체: Target객체 + Advice



- JoinPoint
  - Advice의 적용대상 – 스프링에서는 target의 메서드
- Pointcut
  - 여러 JoinPoint들 중에서 Advice가 적용되는 select 기준



# Advice의 종류

- 실제로 개발하는 관심사 코드

구분	설명
Before Advice	Target의 JoinPoint를 호출하기 전에 실행되는 코드입니다. 코드의 실행 자체에는 관여할 수 없습니다.
After Returning Advice	모든 실행이 정상적으로 이루어진 후에 동작하는 코드입니다.
After Throwing Advice	예외가 발생한 뒤에 동작하는 코드입니다.
After Advice	정상적으로 실행되거나 예외가 발생했을 때 구분 없이 실행되는 코드입니다.
Around Advice	메서드의 실행 자체를 제어할 수 있는 가장 강력한 코드입니다. 직접 대상 메서드를 호출하고 결과나 예외를 처리할 수 있습니다.

# Pointcut

- Advice를 어떤 JoinPoint에 결합할 것인지를 결정하는 설정

구분	설명
<b>execution(@execution)</b>	메서드를 기준으로 Pointcut을 설정합니다.
<b>within(@within)</b>	특정한 타입(클래스)을 기준으로 Pointcut을 설정합니다.
<b>this</b>	주어진 인터페이스를 구현한 객체를 대상으로 Pointcut을 설정합니다.
<b>args(@args)</b>	특정한 파라미터를 가지는 대상들만을 Pointcut으로 설정합니다.
<b>@annotation</b>	특정한 어노테이션이 적용된 대상들만을 Pointcut으로 설정합니다.

# AOP의 실습

- pom.xml 설정

```
<properties>
    <java-version>1.8</java-version>
    <org.springframework-version>5.0.7.RELEASE</org.springframework-
version>
    <org.aspectj-version>1.9.0</org.aspectj-version>
    <org.slf4j-version>1.7.25</org.slf4j-version>
</properties>

<!-- AspectJ -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>
```

# 서비스 계층 설계/Advice 작성/Pointcut

- Advice에는 어노테이션을 이용해서 Pointcut 설정

```
@Aspect  
@Log4j  
@Component  
public class LogAdvice {  
  
    @Before( "execution(* org.zerock.service.SampleService.*(..))")  
    public void logBefore() {  
  
        Log.info("=====");  
    }  
}
```

# AOP처리를 위한 설정

root-context.xml의 일부

```
<context:annotation-config></context:annotation-config>

<context:component-scan
    base-package="org.zerock.service"></context:component-scan>
<context:component-scan
    base-package="org.zerock.aop"></context:component-scan>

<aop:aspectj-autoproxy></aop:aspectj-autoproxy>
```

```
8 @Override
9 public Integer doAdd(String str1, String str2) throws Exception {
10
11     return Integer.parseInt(str1) + Integer.parseInt(str2);
12 }
13 }
```

# AOP 테스트

- AOP가 적용되면 일반 객체가 아니라 Proxy처리가 된 객체가 생성되는 것을 확인할 수 있음

```
@Setter(onMethod = @_({ @Autowired }))  
private SampleService service;  
  
@Test  
public void testClass() {  
  
    Log.info(service);  
    Log.info(service.getClass().getName());  
  
}
```

```
INFO : org.zerock.service.SampleServiceTests - org.zerock.service.SampleServiceImpl@31ea9581  
INFO : org.zerock.service.SampleServiceTests - com.sun.proxy.$Proxy20
```

# args를 이용하는 파라미터 추적

- args라는 특별한 변수를 이용해서 파라미터를 설정하고 기록 가능

```
@Before("execution(* org.zerock.service.SampleService*.doAdd(String, String)) && args(str1, str2)")

public void logBeforeWithParam(String str1, String str2) {

    Log.info("str1: " + str1);
    Log.info("str2: " + str2);
}
```

```
INFO : org.zerock.aop.LogAdvice - =====
INFO : org.zerock.aop.LogAdvice - str1: 123
INFO : org.zerock.aop.LogAdvice - str2: 456
INFO : org.zerock.service.SampleServiceTests - 579
```

# @AfterThrowing

- 예외 발생을 감지해서 AOP로 처리

```
@AfterThrowing(pointcut = "execution(* org.zerock.service.SampleService.*(..))",
throwing="exception")

public void logException(Exception exception) {

    Log.info("Exception....!!!!");
    Log.info("exception: "+ exception);

}
```

# @Around와 ProceedingJoinPoint

- @Around의 경우는 직접 해당 메소드를 실행할 것을 결정할 수 있음
- 파라미터로 ProceedingJoinPoint로 지정하고 사용해야 함
- ProceedingJoinPoint의 메서드
  - getTarget( ): 실제로 실행해야 하는 객체
  - proceed( ): 실제 메서드의 실행

```
@Around("execution(* org.zerock.service.SampleService.*(..))")  
public Object logTime( ProceedingJoinPoint pjp) {  
  
    long start = System.currentTimeMillis();  
    Log.info("Target: " + pjp.getTarget());  
    Log.info("Param: " + Arrays.toString(pjp.getArgs()));  
    //invoke method  
    Object result = null;  
  
    try {  
        result = pjp.proceed();  
    } catch (Throwable e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
  
    long end = System.currentTimeMillis();  
    Log.info("TIME: " + (end - start));  
  
    return result;  
}
```

# 19. 스프링에서 트랜잭션 처리

# 트랜잭션

- 비즈니스 용어에서 '거래'의 의미
- 하나의 '거래'는 여러 번의 데이터베이스 관련 작업이 이루어지므로 이런 작업들을 '하나의 트랜잭션으로 처리'한다고 표현
- 트랜잭션의 원칙 ACID

원자성(Atomicity)	하나의 트랜잭션은 모두 하나의 단위로 처리되어야 합니다. 좀 더 쉽게 말하자면 어떤 트랜잭션이 A와 B로 구성된다면 항상 A, B의 처리결과는 동일한 결과이어야 합니다. 즉 A는 성공했지만, B는 실패할 경우 A, B는 원래의 상태로 되돌려져야만 합니다. 어떤 작업이 잘못되는 경우 모든 것은 다시 원점으로 되돌아가야만 합니다.
일관성(Consistency)	트랜잭션이 성공했다면 데이터베이스의 모든 데이터는 일관성을 유지해야만 합니다. 트랜잭션으로 처리된 데이터와 일반 데이터 사이에는 전혀 차이가 없어야만 합니다.
격리(Isolation)	트랜잭션으로 처리되는 중간에 외부에서의 간섭은 없어야만 합니다.
영속성(Durability)	트랜잭션이 성공적으로 처리되면, 그 결과는 영속적으로 보관되어야 합니다.

# 댓글과 게시물의 반정규화

- 정규화를 하면 여러번의 조인이 필요하고, 성능의 저하가 오는 경우 '반 정규화'를 통해서 해결
- 반정규화는 자주 사용하는 값을 컬럼으로 작성해서 유지하는 방식
- 게시물과 댓글의 숫자의 경우

BOOK_EX.TBL_BOARD	
P *	BNO NUMBER (10)
*	TITLE VARCHAR2 (200 BYTE)
*	CONTENT VARCHAR2 (2000 BYTE)
*	WRITER VARCHAR2 (50 BYTE)
	REGDATE DATE
	UPDATEDATE DATE
☞	PK_BOARD (BNO)
◆	PK_BOARD (BNO)

BOOK_EX.TBL_REPLY	
P *	RNO NUMBER (10)
F *	BNO NUMBER (10)
*	REPLY VARCHAR2 (1000 BYTE)
*	REPLIER VARCHAR2 (50 BYTE)
	REPLYDATE DATE
	UPDATEDATE DATE
☞	PK_REPLY (RNO)
☞	FK_REPLY_BOARD (BNO)
◆	PK_REPLY (RNO)

# 트랜잭션 설정

- pom.xml의 트랜잭션 관련 설정 추가

```
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
  </bean>

<tx:annotation-driven />
```

# 트랜잭션 설정의 테스트

- 2 개 이상의 테이블에 insert 작업을 하나의 '트랜잭션' 이라고 가정하고 트랜잭션 설정이 없는 경우와 있는 경우를 비교
- 스프링의 경우 @Transactional을 이용해서 설정 가능
  - 메서드의 @Transactional 설정이 가장 우선시 됩니다.
  - 클래스의 @Transactional 설정은 메서드보다 우선순위가 낮습니다.
  - 인터페이스의 @Transactional 설정이 가장 낮은 우선순위입니다.

# 댓글과 트랙잭션 설정

- tbl\_board 테이블에 댓글 수를 의미하는 replycnt컬럼 추가
- 댓글의 추가와 삭제시에 replycnt는 트랜잭션하에 관리되어야 함

```
alter table tbl_board add (replycnt number default 0);
```

```
update tbl_board set replycnt = (select count(rno) from tbl_reply  
where tbl_reply.bno = tbl_board.bno);
```

```
<update id="updateReplyCnt">  
    update tbl_board set replycnt = replycnt + #{amount} where bno = #{bno}  
</update>
```

# ReplyServiceImpl의 수정

...생략...

```
@Transactional  
@Override  
public int register(ReplyVO vo) {  
  
Log.info("register....." + vo);  
  
    boardMapper.updateReplyCnt(vo.getBno(), 1);  
  
    return mapper.insert(vo);  
  
}
```

...생략...

```
@Transactional  
@Override  
public int remove(Long rno) {  
  
Log.info("remove...." + rno);  
  
    ReplyVO vo = mapper.read(rno);  
  
    boardMapper.updateReplyCnt(vo.getBno(), -1);  
    return mapper.delete(rno);  
  
}
```

# 스프링 웹 프로젝트

File upload

# Objectives

- 스프링에서의 파일 업로드 처리
- <form>과 Ajax를 이용하는 파일 업로드
- 썸네일 처리
- 파일 다운로드의 처리

# 21. 파일 업로드 방식

# 일반적인 파일 업로드 처리 방식

- <form> 태그를 이용하는 방식: 브라우저의 제한이 없어야 하는 경우에 사용
  - 일반적으로 페이지 이동과 동시에 첨부파일을 업로드하는 방식
  - <iframe>을 이용해서 화면의 이동 없이 첨부파일을 처리하는 방식
- Ajax를 이용하는 방식: 첨부파일을 별도로 처리하는 방식
  - <input type='file'>을 이용하고 Ajax로 처리하는 방식
  - Drag And Drop이나 jQuery 라이브러리들을 이용해서 처리하는 방식

# 파일 업로드 라이브러리들

- cos.jar: 2002년도 이후에 개발이 종료되었으므로, 더 이상 사용하는 것을 권장하지 않음
- commons-fileupload: 가장 일반적으로 많이 활용되고, 서블릿 스펙 3.0 이전에도 사용 가능
- 서블릿3.0 이상 – 3.0 이상부터는 자체적인 파일 업로드 처리가 API 상에서 지원

# Servlet 3.0 이상의 경우 web.xml 설정

- 어노테이션 설정 혹은 web.xml의 설정

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

# servlet-context.xml의 설정

- multipartResolver 설정

```
<context:component-scan  
    base-package="org.zerock.controller" />  
  
<beans:bean id="multipartResolver"  
    class="org.springframework.web.multipart.support.StandardServletMultipartResolver">  
  
</beans:bean>
```

# <form> 방식

```
<form action="uploadFormAction" method="post" enctype="multipart/form-data">  
  
<input type='file' name='uploadFile' multiple>  
  
<button>Submit</button>
```

multiple 설정은 브라우저에 제약이 있음

## Browser Support

The numbers in the table specify the first browser version that fully supports the attribute.

Attribute					
multiple	6.0	10.0	3.6	5.0	11.0

# Multipart 탑

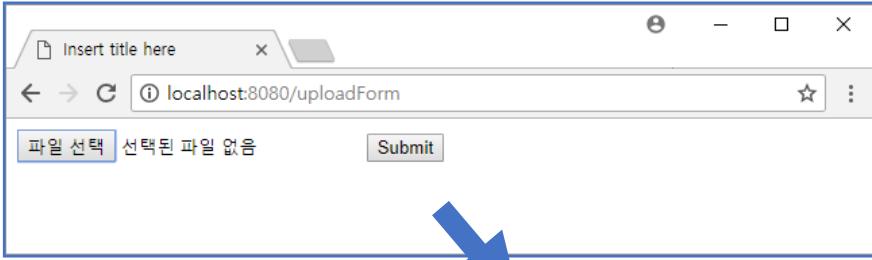
- 업로드되는 파일 데이터 처리에 사용
- 스프링의 컨트롤러의 파라미터로 처리

```
@PostMapping("/uploadFormAction")
public void uploadFormPost(MultipartFile[] uploadFile, Model model) {

    for (MultipartFile multipartFile : uploadFile) {

        log.info("-----");
        log.info("Upload File Name: " +multipartFile.getOriginalFilename());
        log.info("Upload File Size: " +multipartFile.getSize());

    }
}
```



Insert title here

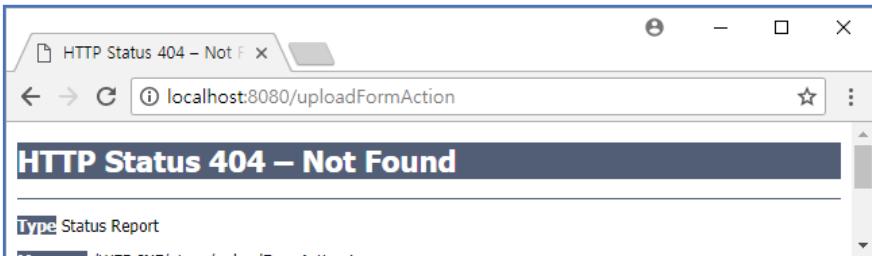
localhost:8080/uploadForm

파일 선택 선택된 파일 없음

Submit

## UploadController

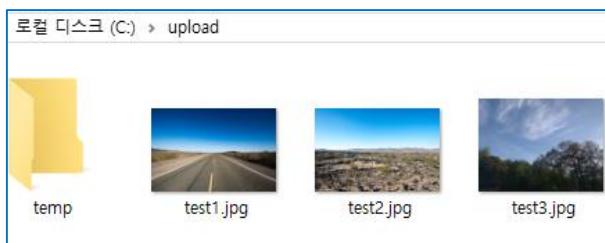
```
INFO : org.zerock.controller.HomeController - Welcome home! The client locale is ko_KR.  
INFO : org.zerock.controller.UploadController - -----  
INFO : org.zerock.controller.UploadController - Upload File Name: jeju.jpg  
INFO : org.zerock.controller.UploadController - Upload File Size: 112701  
INFO : org.zerock.controller.UploadController - -----  
INFO : org.zerock.controller.UploadController - Upload File Name: popart.jpg  
INFO : org.zerock.controller.UploadController - Upload File Size: 1819412
```



# MultipartFile의 주요 메서드

<b>String getName( )</b>	파라미터의 이름 <input> 태그의 이름
<b>String getOriginalFileName( )</b>	업로드되는 파일의 이름
<b>boolean isEmpty( )</b>	파일이 존재하지 않는 경우 true
<b>long getSize( )</b>	업로드되는 파일의 크기
<b>byte[ ] getBytes( )</b>	byte[ ]로 파일 데이터 반환
<b>InputStream getInputStream( )</b>	파일데이터와 연결된 InputStream을 반환
<b>transferTo(File file)</b>	파일의 저장

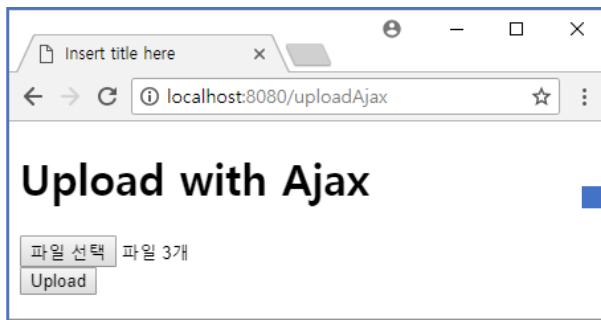
```
File saveFile = new File(uploadFolder, multipartFile.getOriginalFilename());  
  
try {  
    multipartFile.transferTo(saveFile);  
} catch (Exception e) {  
    log.error(e.getMessage());  
}//end catch
```



# Ajax를 이용한 파일 업로드

- Ajax를 이용하는 경우에는 FormData 객체를 이용
- FormData 역시 브라우저별로 지원 여부 확인

```
<script>
$(document).ready(function(){
    $("#uploadBtn").on("click", function(e){
        var formData = new FormData();
        var inputFile = $("input[name='uploadFile']");
        var files = inputFile[0].files;
        console.log(files);
    });
});
</script>
```

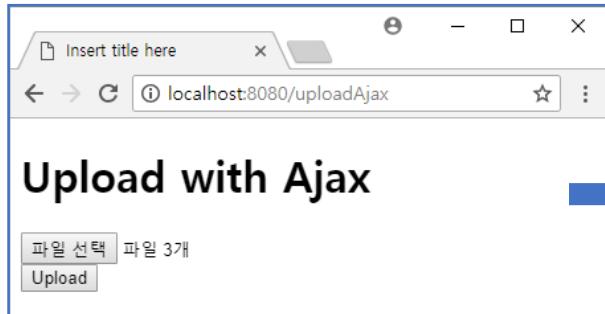


```
▼ fileList 1
▶ 0: File(6316680) {name: "test1.jpg", lastModified: 1525336999000,
▶ 1: File(6838790) {name: "test2.jpg", lastModified: 1525337010000,
▶ 2: File(154349) {name: "test5.jpg", lastModified: 1525337242000,
  length: 3
▶ __proto__: fileList
```

# jQuery를 이용한 파일 전송

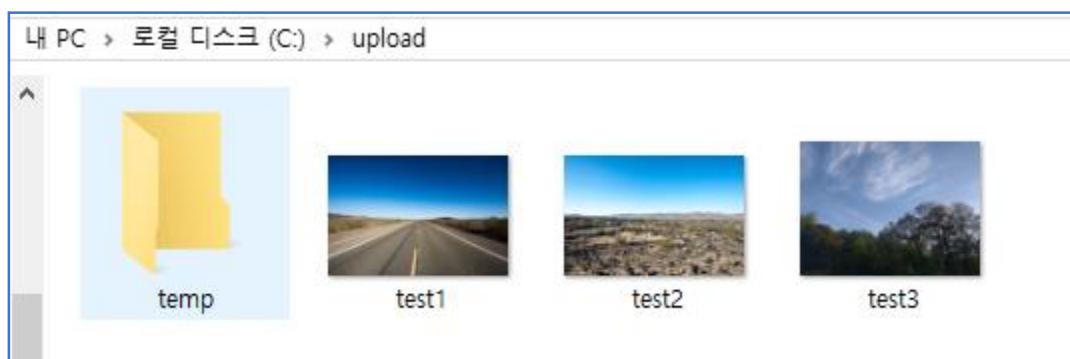
- processData속성과 contentType속성의 조절이 필수

```
$.ajax({  
    url: '/uploadAjaxAction',  
    processData: false,  
    contentType: false,  
    data: formData,  
    type: 'POST',  
    success: function(result){  
        alert("Uploaded");  
    }  
}); //$.ajax
```



## UploadController

```
INFO : org.zerock.controller.UploadController - -----
INFO : org.zerock.controller.UploadController - Upload File Name: test3.jpg
INFO : org.zerock.controller.UploadController - Upload File Size: 264315
INFO : org.zerock.controller.UploadController - -----
INFO : org.zerock.controller.UploadController - Upload File Name: test4.jpg
INFO : org.zerock.controller.UploadController - Upload File Size: 161588
INFO : org.zerock.controller.UploadController - -----
INFO : org.zerock.controller.UploadController - Upload File Name: test5.jpg
INFO : org.zerock.controller.UploadController - Upload File Size: 154349
```



# 파일 전송시 고려 사항들

- 동일한 이름으로 파일이 업로드 된다면 기존 파일이 사라지는 문제
- 이미지 파일의 경우에는 원본 파일의 용량이 큰 경우 섬네일 이미지를 생성해야 하는 문제
- 이미지 파일과 일반 파일을 구분해서 다운로드 혹은 페이지에서 조회하도록 처리하는 문제
- 첨부파일 공격에 대비하기 위한 업로드 파일의 확장자 제한

## 22. 파일 업로드 상세 처리

# 파일의 확장자나 크기 사전처리

- 정규식을 이용해서 파일 확장자 체크

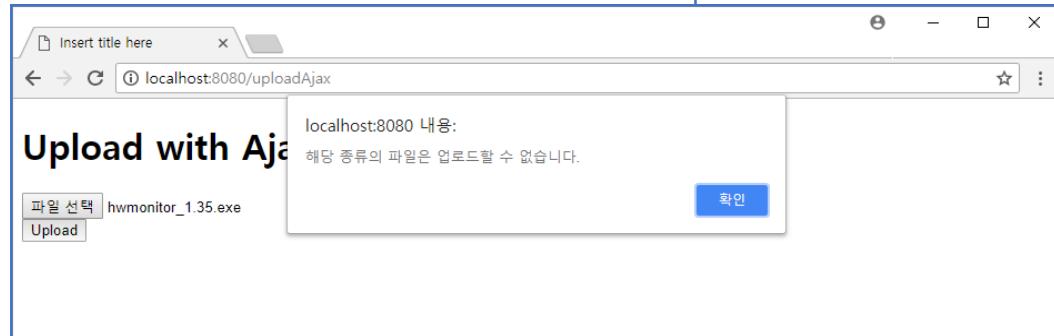
```
var regex = new RegExp("(.*?)\.(exe|sh|zip|alz)$");
var maxSize = 5242880; //5MB

function checkExtension(fileName, fileSize){

    if(fileSize >= maxSize){
        alert("파일 사이즈 초과");
        return false;
    }

    if(regex.test(fileName)){
        alert("해당 종류의 파일은 업로드할 수 없습니다.");
        return false;
    }

    return true;
}
```



# 중복된 이름의 파일 처리

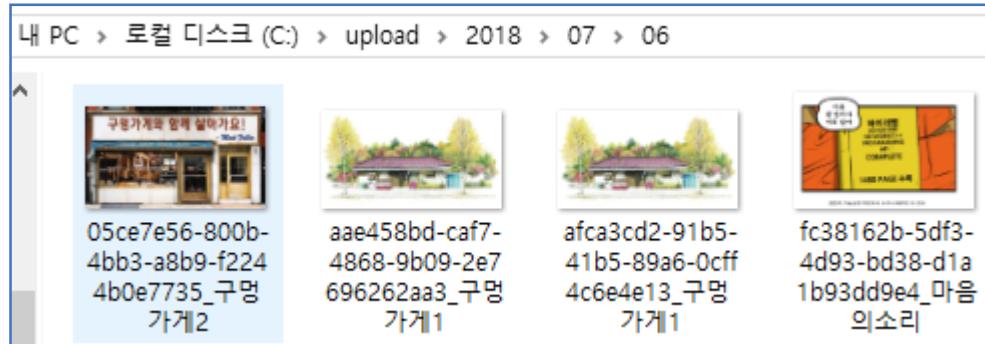
- UUID를 이용해서 고유한 문자열 생성후 업로드된 파일의 이름으로 사용
- 하나의 폴더에 너무 많은 파일이 업로드 되지 않도록 '년/월/일' 폴더 생성

```
private String getFolder() {  
  
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  
  
    Date date = new Date();  
  
    String str = sdf.format(date);  
  
    return str.replace("-", File.separator);  
}
```

```
// make folder -----  
File uploadPath = new File(uploadFolder, getFolder());  
Log.info("upload path: " + uploadPath);
```

```
UUID uuid = UUID.randomUUID();  
  
uploadFileName = uuid.toString() + "_" + uploadFileName;  
  
File saveFile = new File(uploadPath, uploadFileName);  
  
try {  
  
    multipartFile.transferTo(saveFile);  
} catch (Exception e) {  
    Log.error(e.getMessage());  
} // end catch
```

### UUID를 이용해서 고유한 파일명 생성



# 섬네일(thumbnail) 이미지 생성

- 원본 이미지를 그냥 사용하는 경우 브라우저에 너무 많은 데이터들이 전송되는 문제
- 일반적으로 작은 이미지(섬네일)를 생성해서 처리
- JDK의 ImageIO를 이용할 수도 있지만, 해상도 등의 문제로 인해 별도의 라이브러리 활용
  - Thumbnailator 라이브러리 활용

Home » net.coobird » thumbnailator

**Thumbnailator**  
Thumbnailator - a thumbnail generation library for Java

**License** MIT  
**Used By** 48 artifacts

Central (8) Jahia (1)

Version
0.4.8
0.4.7
0.4.6



```
<dependency>
    <groupId>net.coobird</groupId>
    <artifactId>thumbnailator</artifactId>
    <version>0.4.8</version>
</dependency>
```

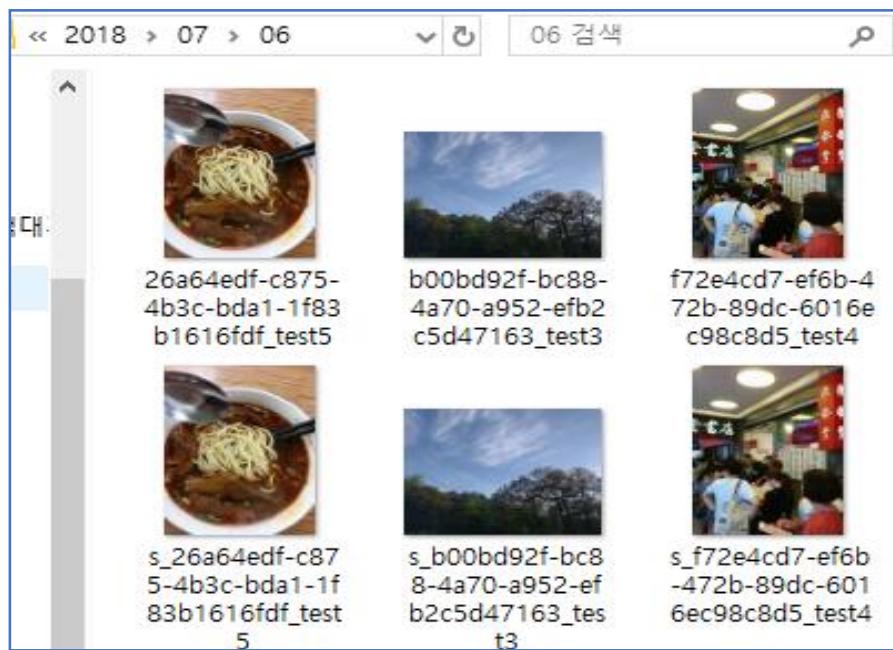
# 섬네일을 처리하는 단계

- 이미지 파일 여부의 판단
- 섬네일 생성 및 저장

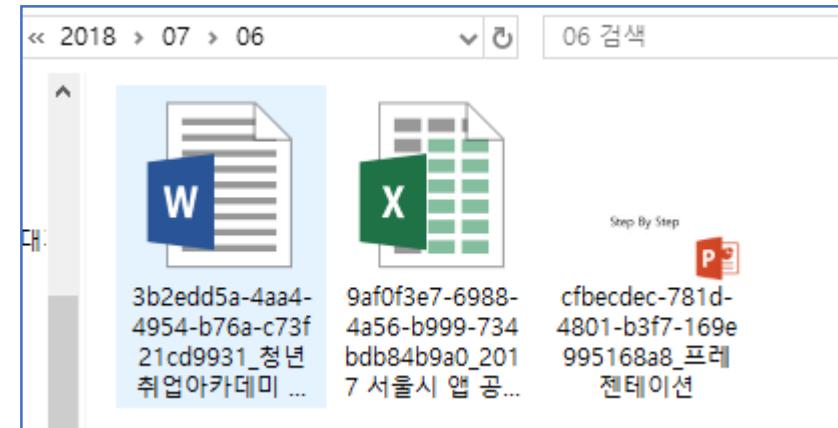
```
try {  
    String contentType = Files.probeContentType(file.toPath());  
  
    return contentType.startsWith("image");  
  
} catch (IOException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}
```

```
// check image type file  
  
if (checkImageType(saveFile)) {  
  
    FileOutputStream thumbnail =  
        new FileOutputStream(new File(uploadPath, "s_" + uploadFileName));  
  
    Thumbnailator.createThumbnail(multipartFile.getInputStream(), thumbnail, 100, 100);  
  
    thumbnail.close();  
}
```

## 이미지파일의 경우



## 일반파일의 경우



# 업로드된 파일의 데이터 반환

- 업로드 이후에 반환해야 하는 정보
  - 업로드된 파일의 이름과 원본 파일의 이름
  - 파일이 저장된 경로
  - 업로드된 파일이 이미지인지 아닌지에 대한 정보
- 정보들을 객체로 처리하고 JSON으로 전송

```
@Data  
public class AttachFileDTO {  
  
    private String fileName;  
    private String uploadPath;  
    private String uuid;  
    private boolean image;  
  
}
```

# 브라우저에서 Ajax의 처리

```
$.ajax({  
    url: '/uploadAjaxAction',  
    processData: false,  
    contentType: false,  
    data: formData,  
    type: 'POST',  
    dataType: 'json',  
    success: function(result){  
  
        console.log(result);  
  
    }  
}); //$.ajax
```

```
▼ (2) [ {...}, {...} ] ⓘ  
  ▼ 0:  
    fileName: "test4.jpg"  
    image: true  
    uploadPath: "2018\07\06"  
    uuid: "5ec7b354-690f-48f9-ae88-8fdfa6061ccf"  
    ► __proto__: Object  
  ▼ 1:  
    fileName: "test5.jpg"  
    image: true  
    uploadPath: "2018\07\06"  
    uuid: "1586d515-2921-4c43-8a8f-3c1a1f600e41"  
    ► __proto__: Object  
    length: 2  
    ► __proto__: Array(0)
```

```
$ajax({  
    url : '/uploadAjaxAction',  
    processData : false,  
    contentType : false,  
    data : formData,  
    type : 'POST',  
    dataType : 'json',  
    success : function(result) {  
  
        console.log(result);  
  
        showUploadedFile(result);  
  
    }  
}); //$.ajax
```

```
function showUploadedFile(uploadResultArr) {  
  
    var str = "";  
  
    $(uploadResultArr).each(function(i, obj) {  
  
        str += "<li>" + obj.fileName + "</li>";  
  
    });  
  
    uploadResult.append(str);  
}
```

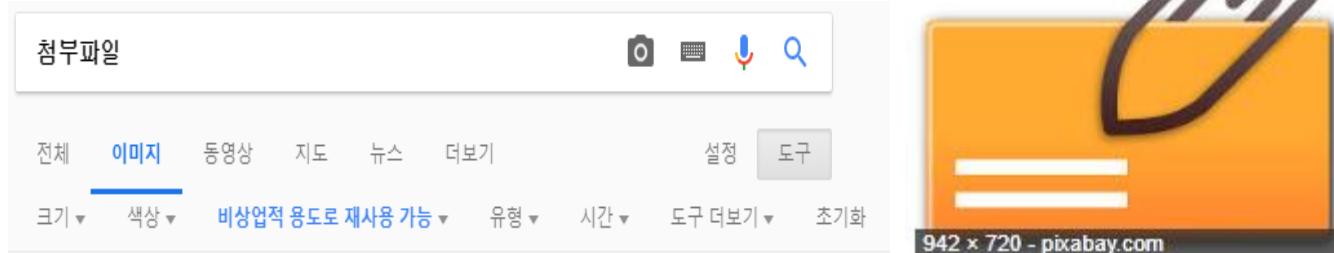
## Upload with Ajax

파일 선택 선택된 파일 없음  
Upload

- test4.jpg
- test4.jpg
- test5.jpg

# 일반 파일의 처리

- 이미지가 아닌 경우에는 화면에서는 간단한 아이콘 등을 이용해서 첨부파일 표시



## Upload with Ajax

선택된 파일 없음

그림파트2.pptx   그림파트3.pptx   그림파트4.pptx

# 섬네일 이미지 보여주기

- 파일의 확장자에 따라 적당한 MIME타입 데이터를 지정

```
HttpHeaders header = new HttpHeaders();  
  
header.add("Content-Type", Files.probeContentType(file.toPath()));
```

파일의 확장자에 따라서 **MIME**타입이 다르게 처리

▼ Response Headers [view source](#)

Content-Length: 1190  
Content-Type: image/png  
Date: Mon, 07 May 2018 05:34:40 GMT

▼ Response Headers [view source](#)

Content-Length: 264315  
Content-Type: image/jpeg  
Date: Mon, 07 May 2018 05:39:37 GMT

# 첨부파일의 다운로드

- 일반 파일의 경우 클릭시에 다운로드 처리

```
@GetMapping(value = "/download", produces = MediaType.APPLICATION_OCTET_STREAM_VALUE)
@ResponseBody
public ResponseEntity<Resource> downloadFile(String fileName) {

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

    Resource resource = new FileSystemResource("c:\\\\upload\\\\" + fileName);

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

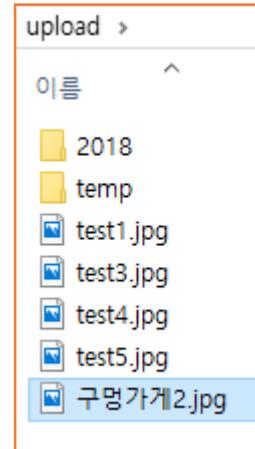
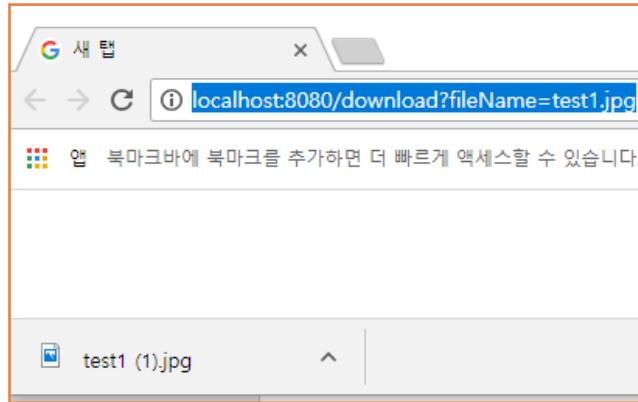
    String resourceName = resource.getFilename();

    HttpHeaders headers = new HttpHeaders();

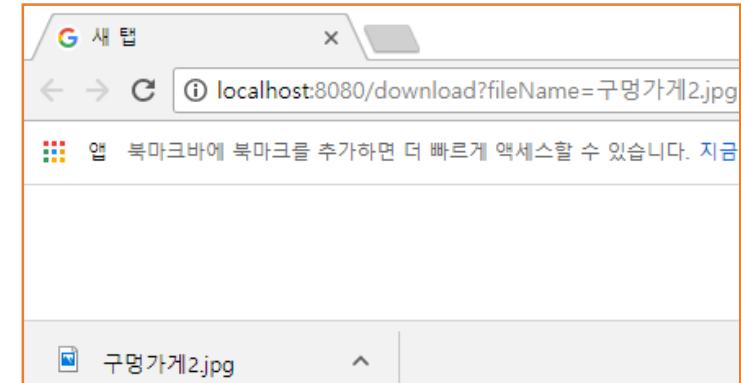
    try {
        headers.add("Content-Disposition",
                    "attachment; filename=" + new String(resourceName.getBytes("UTF-8"), "ISO-8859-1"));
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }

    return new ResponseEntity<Resource>(resource, headers, HttpStatus.OK);
}
```

## 자동으로 해당 파일을 다운로드



## 한글이름 파일 다운로드



## IE의 경우에 한글 파일 이름 처리 주의

```
boolean checkIE = (userAgent.indexOf("MSIE") > -1 || userAgent.indexOf("Trident") > -1);

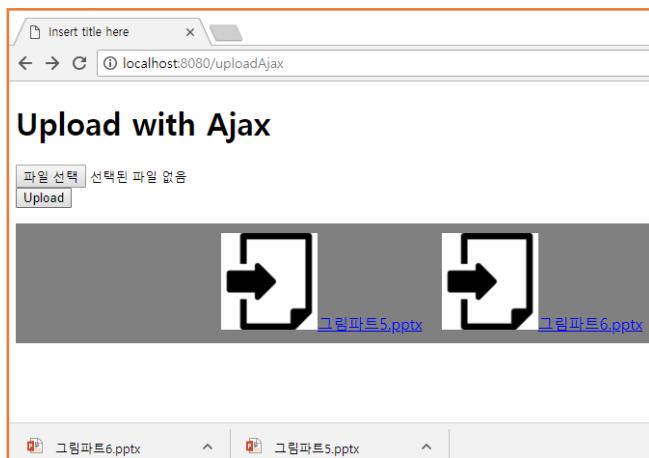
String downloadName = null;

if (checkIE) {
    downloadName = URLEncoder.encode(resourceName, "UTF8").replaceAll("\\+", " ");
} else {
    downloadName = new String(resourceName.getBytes("UTF-8"), "ISO-8859-1");
}
```

# 화면에서의 다운로드 처리

- 화면에서는 '/download' 경로로 요청

```
if(!obj.image){  
  
    var fileCallPath = encodeURIComponent( obj.uploadPath+"/"+ obj.uuid +"_"+obj.fileName);  
  
    str += "<li><a href='/download?fileName="+fileCallPath+"'>  
        +<img src='/resources/img/attach.png'>"+obj.fileName+"</a></li>"  
}
```



# 원본 이미지 보여주기

- <div>를 이용해서 화면 내에 원본 이미지를 보여주도록 처리
- jQuery의 animate( )를 이용해서 처리

```
$(".bigPictureWrapper").css("display","flex").show();  
  
$(".bigPicture")  
    .html("<img src='/display?fileName="+fileCallPath+"'>")  
    .animate({width: '100%', height: '100%'}, 1000);
```

## Upload with Ajax

파일 선택 선택된 파일 없음

Upload



# 첨부파일의 삭제

- 첨부파일 삭제시 고려해야 하는 사항들
  - 이미지 파일의 경우에는 섬네일까지 같이 삭제되어야 하는 점
  - 파일을 삭제한 후에는 브라우저에서도 섬네일이나 파일 아이콘이 삭제되도록 처리하는 점
  - 비정상적으로 브라우저의 종료 시 업로드된 파일의 처리
- 화면에서의 삭제
  - 첨부파일의 <div> 자체를 삭제
- 서버에서의 삭제
  - 이미지의 경우에는 섬네일 삭제/ 원본 삭제
  - 일반 파일의 경우에는 원본 삭제

# 첨부파일 삭제의 고민

- 첨부파일은 삭제했지만, 원래의 게시글을 수정/삭제하지 않은 경우
- 비정상적으로 수정/삭제 중에 브라우저가 종료된 경우
- 무난한 방식은 DB에 있는 첨부파일의 목록과 실제 업로드 폴더에 있는 파일의 목록을 비교해서 처리하는 작업을 주기적으로 처리
- Quartz라이브러리를 이용해서 스프링에서 주기적으로 처리

# 등록을 위한 화면 처리

- 추가된 첨부파일은 이미 업로드가 완료된 상황
- <form> 전송시 <input type='hidden'> 태그들을 첨부된 파일의 수 만큼 생성해서 같이 전송

JSON정보는 <input type='hidden'>으로 변환

```
<input type="hidden" name="attachList[0].fileName" value="test3.jpg">
<input type="hidden" name="attachList[0].uuid" value="cf4a5227-e9c5-460f-b8f9-1e9f672a330b">
<input type="hidden" name="attachList[0].uploadPath" value="2018\05\16">
<input type="hidden" name="attachList[0].fileType" value="true">
<input type="hidden" name="attachList[1].fileName" value="test4.jpg">
<input type="hidden" name="attachList[1].uuid" value="94c87808-604c-48b4-8b34-b35ee7af1036">
<input type="hidden" name="attachList[1].uploadPath" value="2018\05\16">
<input type="hidden" name="attachList[1].fileType" value="true">
<input type="hidden" name="attachList[2].fileName" value="test5.jpg">
<input type="hidden" name="attachList[2].uuid" value="c4b0c2ac-abdd-49f3-890f-c7ba1a24b3cc">
<input type="hidden" name="attachList[2].uploadPath" value="2018\05\16">
<input type="hidden" name="attachList[2].fileType" value="true">
```

```
create table tbl_attach (
    uuid varchar2(100) not null,
    uploadPath varchar2(200) not null,
    fileName varchar2(100) not null,
    filetype char(1) default 'I',
    bno number(10,0)
);

alter table tbl_attach add constraint pk_attach primary key (uuid);

alter table tbl_attach add constraint fk_board_attach foreign key (bno)
references tbl_board(bno);
```

tbl\_board와는 외래키로 설정

```
@Data  
public class BoardAttachVO {  
  
    private String uuid;  
    private String uploadPath;  
    private String fileName;  
    private boolean fileType;  
  
    private Long bno;  
  
}
```

<input type='hidden'>으로 만들어진  
파일 정보들을 BoardAttachVO로 변환

```
public class BoardVO {  
  
    private Long bno;  
    private String title;  
    private String content;  
    private String writer;  
    private Date regdate;  
    private Date updateDate;  
  
    private int replyCnt;  
  
    private List<BoardAttachVO> attachList;  
}
```

BoardVO는 여러 개의 첨부 파일을 가지도록

```
@Transactional  
@Override  
public void register(BoardVO board) {  
  
    Log.info("register....." + board);  
  
    mapper.insertSelectKey(board);  
  
    if(board.getAttachList() == null || board.getAttachList().size() <= 0) {  
        return;  
    }  
  
    board.getAttachList().forEach(attach ->{  
  
        attach.setBno(board.getBno());  
        attachMapper.insert(attach);  
    });  
}
```

트랜잭션하에 여러 개의 첨부 파일 정보도 DB에 저장

test3.jpg,test4.jpg,test5.jpg 를 게시물 등록 시에 추가한 경우

UUID	UPLOADPATH	FILENAME	FILETYPE	BNO
1 e7f922a7-1456-40fa-85ae-2dabb9f85b73	2018\07\08	test3.jpg	1	3145784
2 9fa37bc2-415f-4551-9fb2-97892e219f02	2018\07\08	test5.jpg	1	3145784
3 d4f32238-7f00-4a50-b4e6-2d1441501e62	2018\07\08	test4.jpg	1	3145784

Files

파일 선택 선택된 파일 없음



test3.jpg ×



test4.jpg ×



test5.jpg ×

# 게시물의 조회와 첨부파일

- 게시물 조회시에는 게시물의 정보와 첨부파일들의 정보를 같이 가져오도록 해야 함
  - Ajax로 해당 게시물의 첨부파일들만 조회하는 방식

```
@GetMapping(value = "/getAttachList", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
```

```
@ResponseBody
```

```
public ResponseEntity<List<BoardAttachVO>> getAttachList(Long bno) {
```

```
    log.info("getAttachList " + bno);
```

```
    return new ResponseEntity<>(service.getAttachList(bno), HttpStatus.OK);
```

```
}
```

```
    var bno = '<c:out value="${board.bno}" />';
```

```
    $.getJSON("/board/getAttachList", {bno: bno}, function(arr){
```

```
        console.log(arr);
```

```
    }); //end getjson
```

# 게시물의 삭제와 첨부파일

- 게시물의 삭제시에는 데이터베이스 상의 파일 정보의 삭제
- 실제 업로드된 파일 (이미지의 경우 썸네일도) 삭제

게시물 수정 삭제

Board Modify Page

Board Modify Page	
Bno	2097170
Title	Remove Test
Text area	Remove Test
Writer	writer
<button>Modify</button> <button>Remove</button> <button>List</button>	

데이터베이스내 첨부 파일의 기록

UUID	UPLOAD...	FILENAME	FILETYPE	BNO

폴더내 썸네일과 일반 파일 삭제



# 서버측 게시물 수정과 첨부파일의 삭제

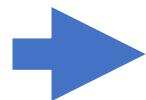
- 첨부파일은 사실상 수정이라는 개념이 존재하지 않음
- 삭제후 다시 추가하는 방식

test3.jpg,test4.jpg,test5.jpg 를 게시물 등록 시에 추가한 경우

The screenshot shows a file selection interface and a database table. On the left, a 'Files' panel displays three selected files: 'test3.jpg', 'test4.jpg', and 'test5.jpg'. On the right, a table lists these files with their details:

LOADPATH	FILENAME	FILETYPE	BNO
\07\08	test3.jpg	1	3145784
\07\08	test5.jpg	1	3145784
\07\08	test4.jpg	1	3145784

test3.jpg,test4.jpg를 삭제하고 jeju.jpg파일을 추가한 경우



The screenshot shows a file selection interface and a database table. On the left, a 'Files' panel displays two selected files: 'test5.jpg' and 'jeju.jpg'. On the right, a table lists the files with their details:

UUID	UPLOADPATH	FILENAME	FILETYPE	BNO
1 9fa37bc2-415f-4551-9fb2-97892e219f02	2018\07\08	test5.jpg	1	3145784
2 7fe55ea1-8688-4f90-9625-b3db746aa9aa	2018\07\08	jeju.jpg	1	3145784

# 잘못된 첨부파일들의 삭제

- 첨부파일정보와 DB상의 정보가 일치하지 않는 상황
  - 첨부파일만을 등록하고 게시물을 등록하지 않았을 때의 문제 - 파일은 이미 서버에 업로드되었지만, 게시물을 등록하지 않았으므로 의미 없이 파일들만 서버에 업로드된 상황
  - 게시물을 수정할 때 파일을 삭제했지만 실제로 폴더에서 기존 파일은 삭제되지 않은 문제  
– 데이터베이스에는 기존 파일이 삭제되었지만, 실제 폴더에는 남는 문제

어제 날짜로 등록된 첨부파일의 목록  
을 구한다.



어제 업로드가 되었지만, 데이터베이스에는 존재하지 않는 파일들을 찾는다.



데이터베이스와 비교해서 필요 없는  
파일들을 삭제한다.

UUID	UPLOADPATH	FILENAME	FILETYPE
e941b1c6-6fec-458...	2018#05#21	test3.jpg	1
e7c71a10-7aab-4a...	2018#05#21	test4.jpg	1
b4d2bd62-03bf-45...	2018#05#21	test5.jpg	1
2726675a-c695-4d...	2018#05#21	0c2229a.jpg	1
20a6b192-3bad-43...	2018#05#21	888627.jpg	1
97a57a7b-a982-4f...	2018#05#21	c0022996_491ae4059d9...	1
e8f3ed3d-43e5-45...	2018#05#21	freesoul_76669_1[9].jpg	1
758f8d17-e89b-42...	2018#05#21	jeju.jpg	1
212aba25-e5b0-49...	2018#05#21	z7676.jpg	1
ba6c8831-80ba-4fa...	2018#05#21	z7677.jpg	1

2f51c91d-6a26-48d4-4713-9e33929902b1\_20180318  
3c057106-d2f3-4add-8647-0b1c4c04573b\_NCS평가양식0222  
20a6b192-3bad-4315-8962-9738c9db4dd4\_888627  
97a57a7b-a982-4f7e-a938-ac1af338a40\_c0022996\_491ae4059d958  
212aba25-e5b0-494e-bccc-2cc850a98a03\_z7676  
758f8d17-e89b-42d8-b5a6-15cd0cb01bff\_jeju  
1069d80b-c4bb-47a6-af97-5ed20482f431\_프리젠테이션  
2726675a-c695-4d51-8a1c-d8c6d29d3b2\_0c2229a  
b4d2bd62-03bf-45e4-b23c-ff698c440c34\_test5  
ba6c8831-80ba-4fa9-bd92-43753a9c94f8\_z7677  
bc96f685-5598-48f6-b1ea-67b884d9a29\_NCS평가양식0226  
bd01b459-9f14-44b5-839e-ae16b988304f\_c0022996\_491ae4059d9...  
e7c71a10-7aab-4a45-9517-05b220fb6d0\_test4  
e8f3ed3d-43e5-4542-9453-73e4a8ab7007\_freesoul\_76669\_1[9]  
e941b1c6-6fec-4589-9c11-35c365026e91\_test3  
s\_20a6b192-3bad-4315-8962-9738c9db4dd4\_888627  
s\_97a57a7b-a982-4f7e-a938-ac1af338a40\_c0022996\_491ae4059d9...  
s\_212aba25-e5b0-494e-bccc-2cc850a98a03\_z7676  
s\_758f8d17-e89b-42d8-b5a6-15cd0cb01bff\_jeju

20a6b192-3bad-4315-8962-9738c9db4dd4\_888627  
97a57a7b-a982-4f7e-a938-ac1af338a40\_c0022996\_491ae4059d958  
212aba25-e5b0-494e-bccc-2cc850a98a03\_z7676  
758f8d17-e89b-42d8-b5a6-15cd0cb01bff\_jeju  
2726675a-c695-4d51-8a1c-d8c6d29d3b2\_0c2229a  
b4d2bd62-03bf-45e4-b23c-ff698c440c34\_test5  
ba6c8831-80ba-4fa9-bd92-43753a9c94f8\_z7677  
e7c71a10-7aab-4a45-9517-05b220fb6d0\_test4  
e8f3ed3d-43e5-4542-9453-73e4a8ab7007\_freesoul\_76669\_1[9]  
e941b1c6-6fec-4589-9c11-35c365026e91\_test3  
s\_20a6b192-3bad-4315-8962-9738c9db4dd4\_888627  
s\_97a57a7b-a982-4f7e-a938-ac1af338a40\_c0022996\_491ae4059d9...  
s\_212aba25-e5b0-494e-bccc-2cc850a98a03\_z7676  
s\_758f8d17-e89b-42d8-b5a6-15cd0cb01bff\_jeju  
s\_2726675a-c695-4d51-8a1c-d8c6d29d3b2\_0c2229a  
s\_b4d2bd62-03bf-45e4-b23c-ff698c440c34\_test5  
s\_ba6c8831-80ba-4fa9-bd92-43753a9c94f8\_z7677  
s\_e7c71a10-7aab-4a45-9517-05b220fb6d0\_test4  
s\_e8f3ed3d-43e5-4542-9453-73e4a8ab7007\_freesoul\_76669\_1[9]  
s\_e941b1c6-6fec-4589-9c11-35c365026e91\_test3

# Quartz라이브러리 설정

- 주기적으로 반복해야 하는 작업을 Java를 이용해서 처리할 수 있도록 하는 경우에 사용

```
<!-- https://mvnrepository.com/artifact/org.quartz-scheduler/quartz -->
```

```
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz</artifactId>
    <version>2.3.0</version>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.quartz-scheduler/quartz-jobs -->
```

```
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz-jobs</artifactId>
    <version>2.3.0</version>
</dependency>
```

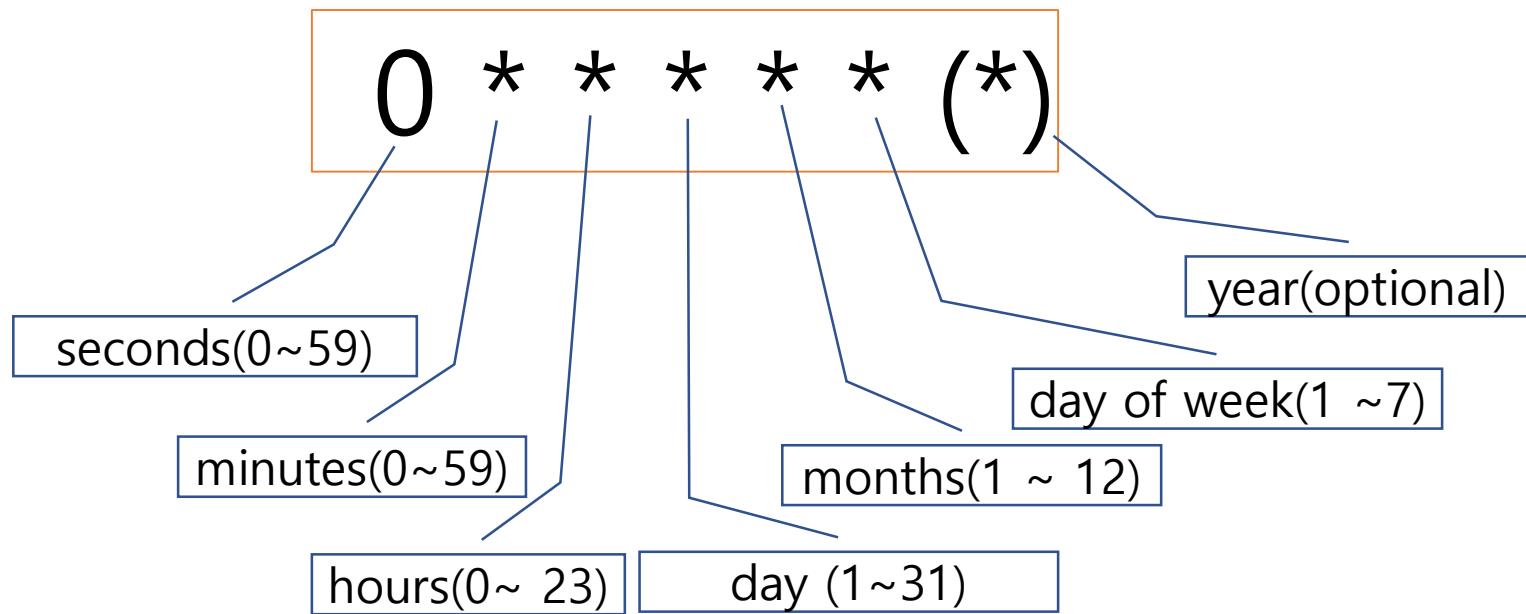
# Task설정

- root-context.xml에 설정 추가

```
xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring  
http://mybatis.org/schema/mybatis-spring-1.2.xsd  
    http://www.springframework.org/schema/task  
    http://www.springframework.org/schema/task/spring-task-4.3.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-4.3.xsd  
                http://www.springframework.org/schema/aop  
                http://www.springframework.org/schema/aop/spring-aop-4.3.xsd  
                    http://www.springframework.org/schema/tx  
                    http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">  
...생략...  
<task:annotation-driven/>
```

# cron 설정

- 시간 단위의 값을 조정해서 주기 설정



# 스프링 웹 프로젝트

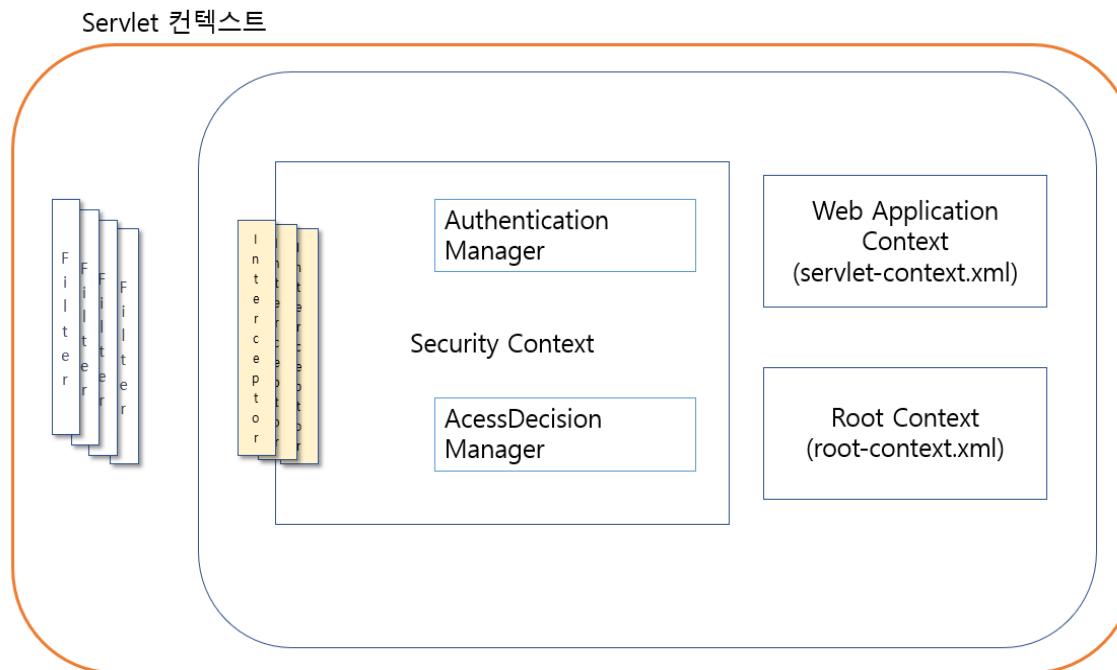
Spring Security

# Objectives

- 스프링 웹 시큐리티의 이해
- 인증(Authentication) VS 인가(Authorization)
- 로그인 처리와 자동 로그인
- CSRF 공격과 CSRF 토큰

# Spring Web Security 소개

- 스프링 시큐리티의 시작
  - Acegi 프레임워크에서 시작
  - 필터를 이용한 처리
- 스프링 시큐리티의 기본 구조



# 스프링 시큐리트를 위한 설정

- 스프링 시큐리티 관련 라이브러리
- 스프링 시큐리티를 JSP에서 적용하기 위한 taglib

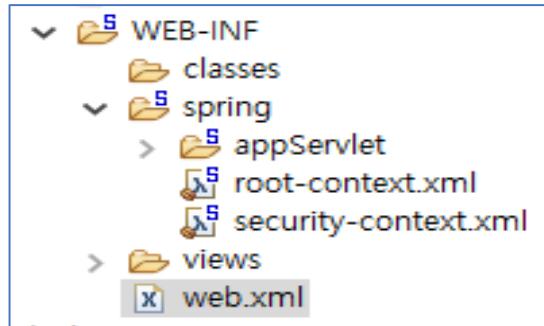
```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-taglibs -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>5.0.6.RELEASE</version>
</dependency>
```

# security-config.xml의 추가 /web.xml 변경

- root-context.xml등에 추가하는 설정도 가능하지만 별도로 분리 할 수 있음



## web.xml의 일부

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

# web.xml에서 security-context.xml 인식

## web.xml의 일부

```
<!-- The definition of the Root Spring Container shared by all Servlets  
and Filters -->  
<context-param>  
    <param-name>contextConfigLocation</param-name>  
    <param-value>/WEB-INF/spring/root-context.xml  
    /WEB-INF/spring/security-context.xml  
    </param-value>  
</context-param>
```

# 최초의 security-context.xml의 설정

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

    <security:http>

        <security:form-login />

    </security:http>

    <security:authentication-manager>

    </security:authentication-manager>

</beans>
```

# 시큐리티 적용이 필요한 URI설계

- 스프링 시큐리티의 최소한의 설정이 완료되었다면 시큐리티에 의해 제어가 필요한 URI를 설계
  - /sample/all -> 로그인을 하지 않은 사용자도 접근 가능한 URI
  - /sample/member -> 로그인 한 사용자들만이 접근할 수 있는 URI
  - /sample/admin -> 로그인 한 사용자들 중에서 관리자 권한을 가진 사용자만이 접근할 수 있는 URI

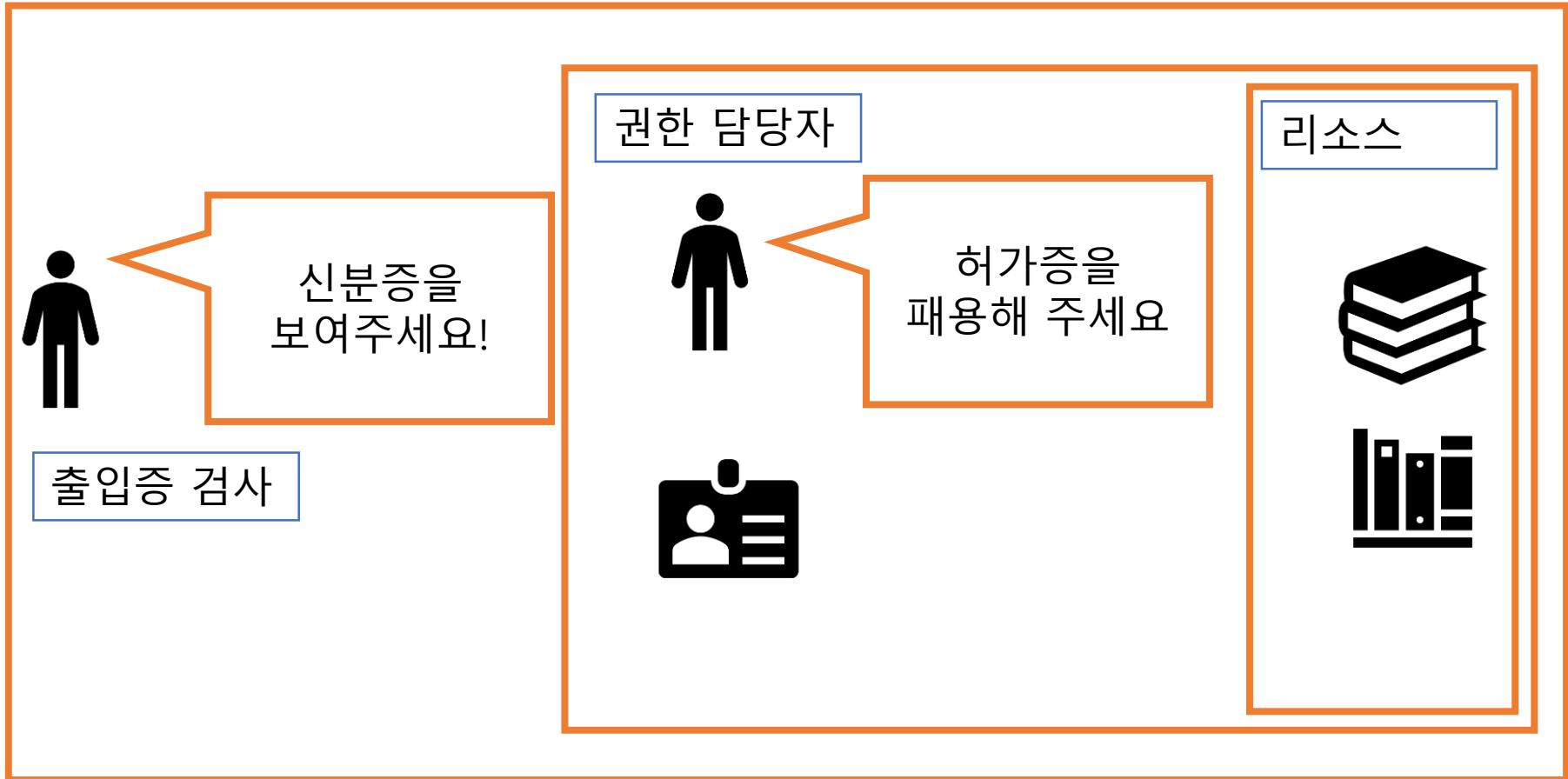
```
@Log4j
@RequestMapping("/sample/*")
@Controller
public class SampleController {

    @GetMapping("/all")
    public void doAll() {
        Log.info("do all can access everybody");
    }

    @GetMapping("/member")
    public void doMember() {
        Log.info("logined member");
    }

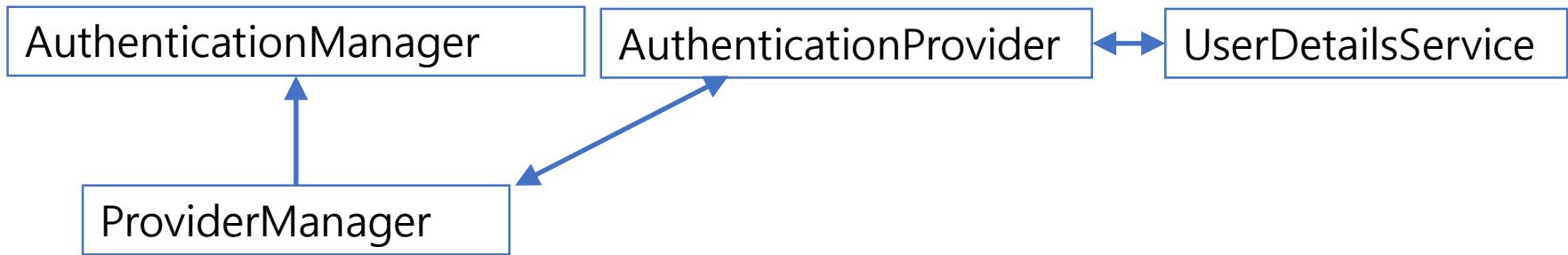
    @GetMapping("/admin")
    public void doAdmin() {
        Log.info("admin only");
    }
}
```

# 인증(Authentication)과 권한 부여(Authorization - 인가)



# 스프링 시큐리티의 핵심 구조

- AuthenticationManager 는 다양한 방식의 AuthenticationProvider 를 이용해서 사용자 정보를 확인



# 로그인과 로그아웃처리

- security-context.xml 접근제한 설정
- access 속성은 기본적으로 표현식(expression)이나 문자열을 이용해서 처리 가능

```
<security:http>

    <security:intercept-url pattern="/sample/all" access="permitAll"/>

    <security:intercept-url pattern="/sample/member" access="hasRole('ROLE_MEMBER')"/>

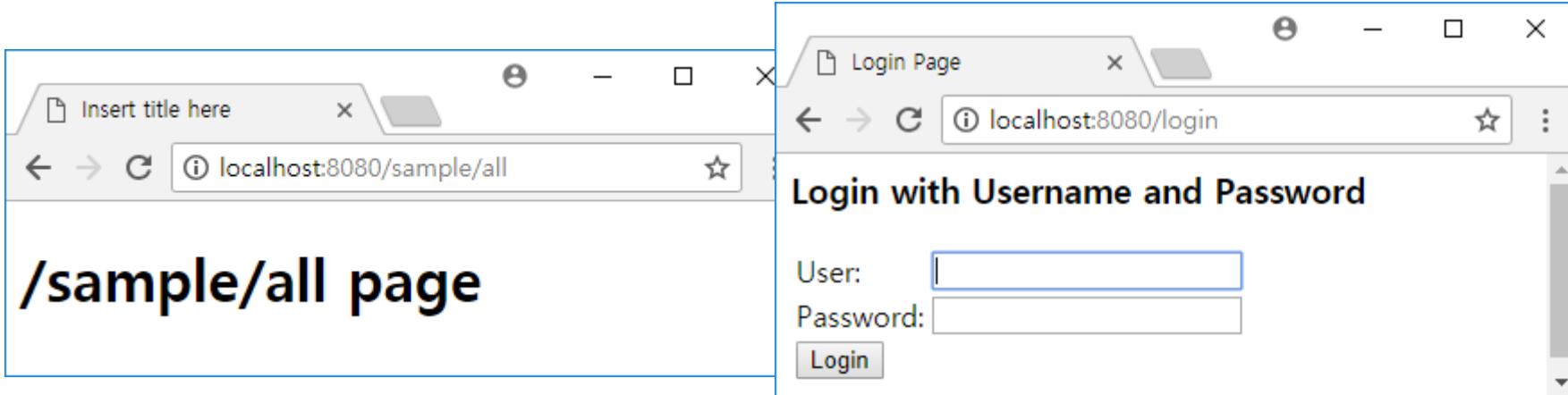
    <security:form-login />

</security:http>

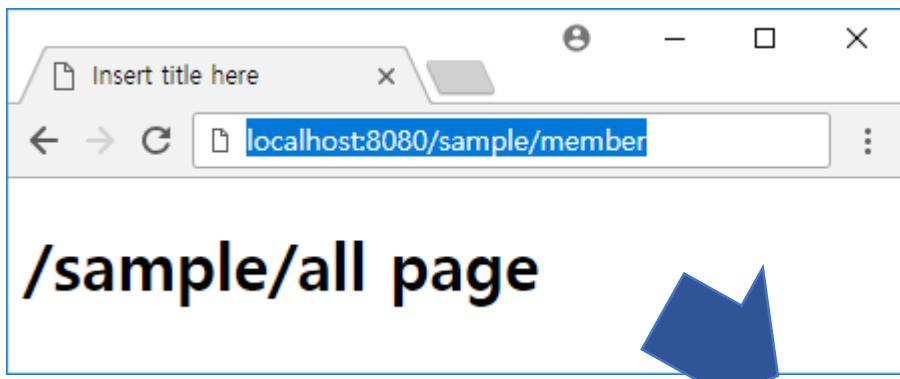
<security:authentication-manager>

</security:authentication-manager>

</beans>
```



/sample/member 호출 시 강제로 로그인 페이지로 이동



# 단순로그인 처리

- 단순한 문자열로 권한을 지정할 수 있지만 5버전이상에서는 반드시 PasswordEncoder작업이 필요하므로 주의
- {noop}을 이용해서 임시로 처리 가능

```
<security:authentication-manager>

    <security:authentication-provider>
        <security:user-service>

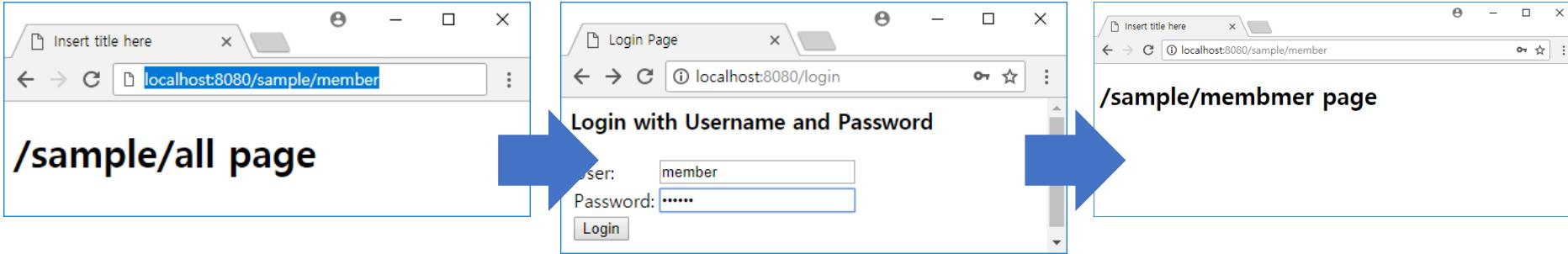
            <security:user name="member" password="{noop}member" authorities="ROLE_MEMBER"/>

        </security:user-service>

    </security:authentication-provider>

</security:authentication-manager>
```

# 로그인 후 이동 확인



A screenshot of a browser window titled "Login Page" with the URL "localhost:8080/login". The page displays a form titled "Login with Username and Password". It has two input fields: "User:" and "Password:", both currently empty. Below the fields is a "Login" button.

JSESSIONID 쿠키를 삭제한 후  
동일 URI 접근 시에는 다시 로그인 필요

A screenshot of the Chrome DevTools Application tab. The left sidebar shows "Application" with options for Manifest, Service Workers, and Clear storage. The main area is titled "Cookies" and lists a cookie for the domain "localhost" with the path "/" and the value "D3190413CBA8DBCD9FB00E2E648687A1".

Name	Value	Domain	Path	Ex
JSESSIONID	D3190413CBA8DBCD9FB00E2E648687A1	localh...	/	19

# 여러 권한을 가지는 사용자 설정

```
<security:authentication-manager>

    <security:authentication-provider>
        <security:user-service>

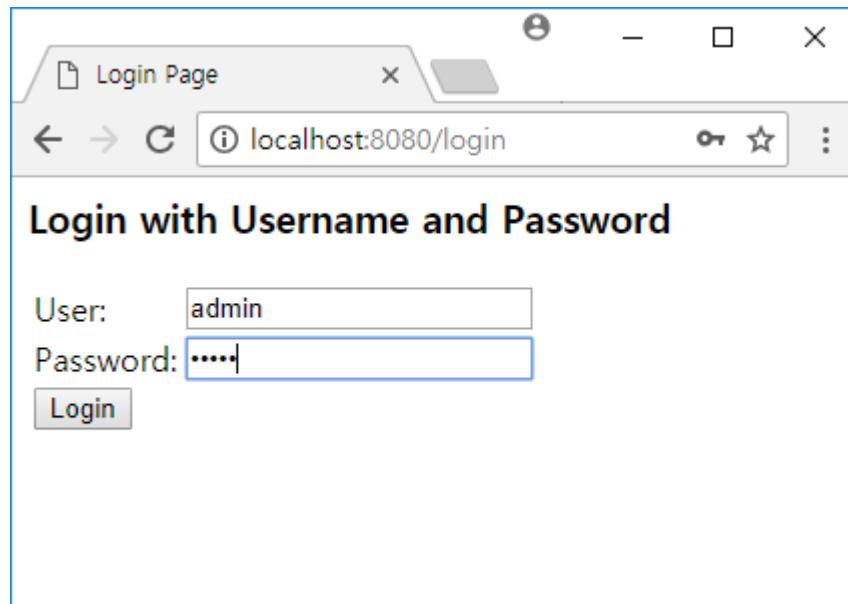
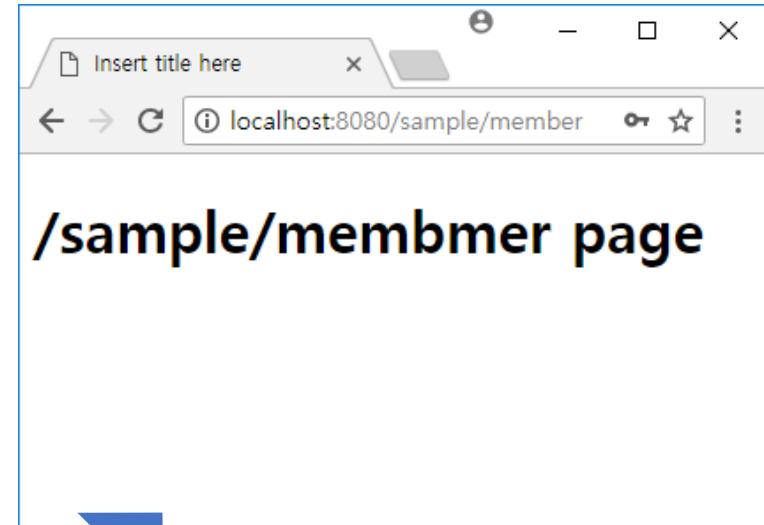
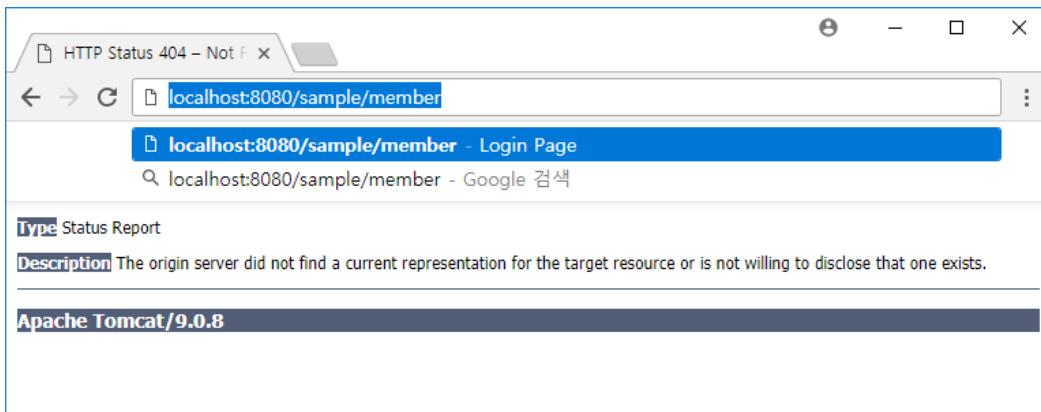
            <security:user name="member" password="{noop}member" authorities="ROLE_MEMBER"/>

            <security:user name="admin" password="{noop}admin" authorities="ROLE_MEMBER, ROLE_ADMIN"/>

        </security:user-service>

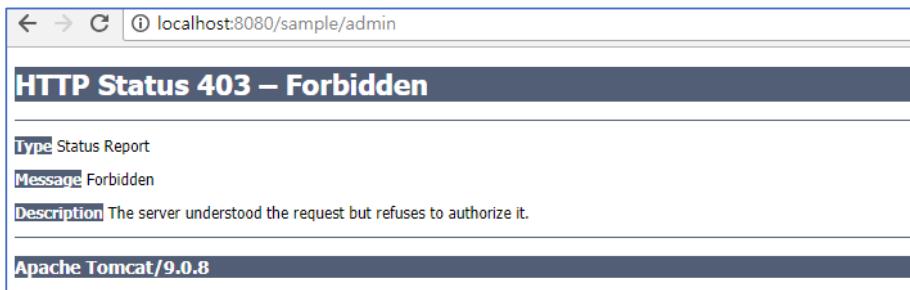
    </security:authentication-provider>

</security:authentication-manager>
```



# 접근 제한 메시지의 처리

- AccessDeniedHandler를 구현하거나 URI를 지정해서 처리



```
<security:access-denied-handler  
error-page="/accessError"/>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  
pageEncoding="UTF-8"%>  
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>  
...  
  
<h2><c:out value="${SPRING_SECURITY_403_EXCEPTION.getMessage()}"/></h2>  
  
<h2><c:out value="${msg}"/></h2>  
  
</ul>  
</body>  
</html>
```

# 커스텀 로그인 페이지

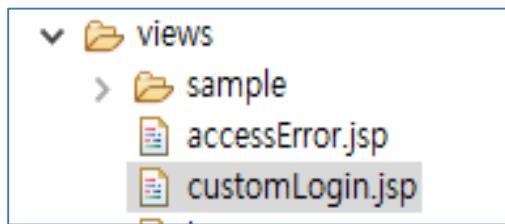
```
<security:form-login login-page="/customLogin" />
```

```
@GetMapping("/customLogin")
public void loginInput(String error, String logout, Model model) {

    log.info("error: " + error);
    log.info("logout: " + logout);

    if (error != null) {
        model.addAttribute("error", "Login Error Check Your Account");
    }

    if (logout != null) {
        model.addAttribute("logout", "Logout!!!");
    }
}
```



```

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

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>

<h1>Custom Login Page</h1>
<h2><c:out value="${error}" /></h2>
<h2><c:out value="${logout}" /></h2>

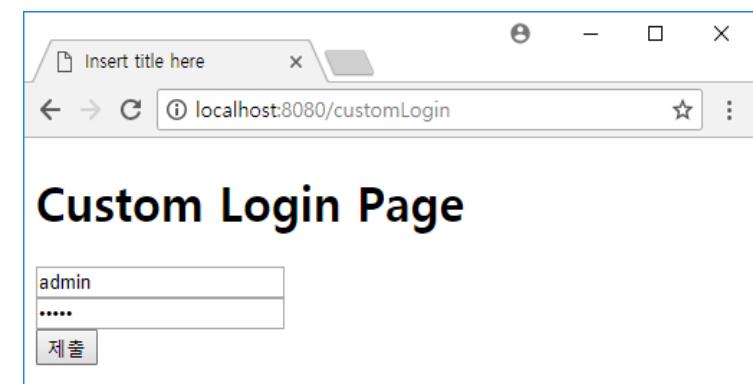
<form method='post' action="/Login">

<div>
    <input type='text' name='username' value='admin'>
</div>
<div>
    <input type='password' name='password' value='admin'>
</div>
<div>
    <input type='submit'>
</div>
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />

</form>

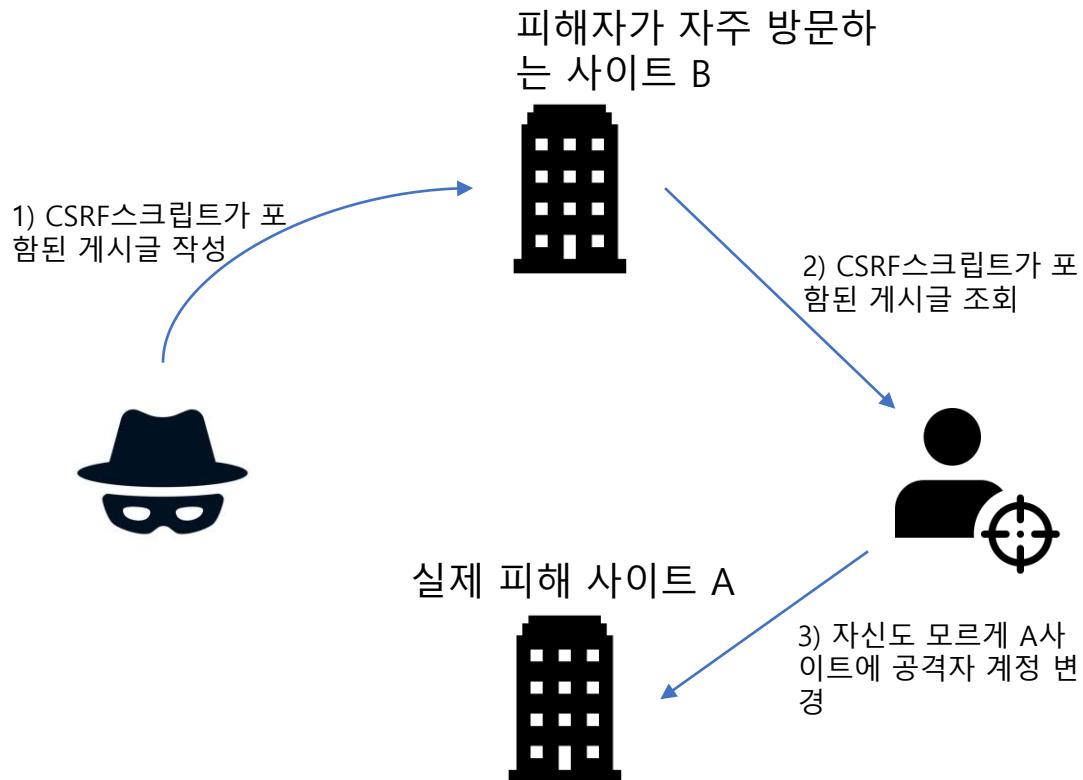
</body>
</html>

```



# CSRF 공격과 CSRF 토큰

- 사이트간 요청 위조(Cross-site request forgery) 공격
- 웹 기본적으로 출처를 따지지 않는다는 점을 이용



# CSRF토큰

- 스프링 시큐리티는 기본적으로 GET방식을 제외하고 모든 요청에 CSRF토큰 사용
- <form>등의 데이터 전송시에 CSRF토큰을 같이 전송하도록 처리

```
▼<form method="post" action="/login">
  ▶<div>...</div>
  ▶<div>...</div>
  ▶<div>...</div>
    <input type="hidden" name="_csrf" value="5a0ded0c-a151-4f6d-95f5-4c66664308d1">
  </form>
```

```
▼<form method="post" action="/login">
  ▶<div>...</div>
  ▶<div>...</div>
  ▶<div>...</div>
    <input type="hidden" name="_csrf" value="3b5b49da-11ba-44a4-a021-8117ebef5b7">
  </form>
  ..
```

# 로그인 성공과 AuthenticationSuccessHandler

- 로그인 성공후 특정 URI로 이동하거나 쿠키 처리 등의 추가적인 작업

```
@Log4j
public class CustomLoginSuccessHandler implements AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication auth)
        throws IOException, ServletException {
        log.warn("Login Success");

        List<String> roleNames = new ArrayList<>();

        auth.getAuthorities().forEach(authority -> {
            roleNames.add(authority.getAuthority());
        });

        log.warn("ROLE NAMES: " + roleNames);

        if (roleNames.contains("ROLE_ADMIN")) {
            response.sendRedirect("/sample/admin");
            return;
        }

        if (roleNames.contains("ROLE_MEMBER")) {
            response.sendRedirect("/sample/member");
            return;
        }

        response.sendRedirect("/");
    }
}
```

```

<bean id="customAccessDenied" class="org.zerock.security.CustomAccessDeniedHandler"></bean>
<bean id="customLoginSuccess" class="org.zerock.security.CustomLoginSuccessHandler"></bean>

<security:http>

    <security:intercept-url pattern="/sample/all" access="permitAll"/>

    <security:intercept-url pattern="/sample/member" access="hasRole('ROLE_MEMBER')"/>

    <security:intercept-url pattern="/sample/admin" access="hasRole('ROLE_ADMIN')"/>

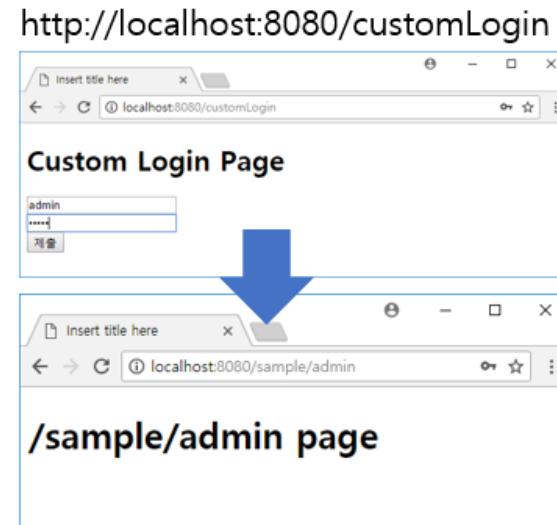
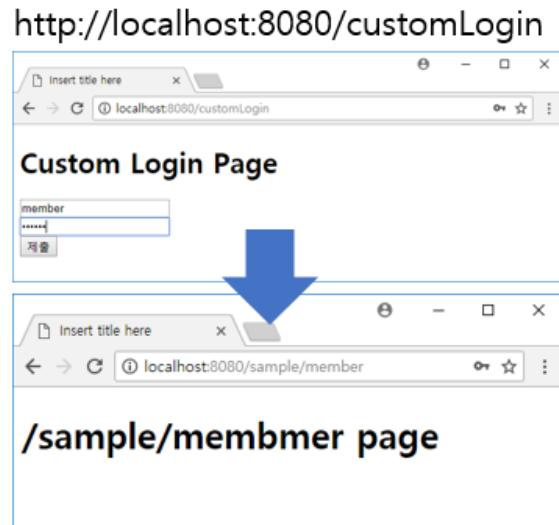
    <security:access-denied-handler ref="customAccessDenied"/>

    <security:form-login login-page="/customLogin" authentication-success-handler-ref="customLoginSuccess" />

    <!-- <security:csrf disabled="true"/> -->

</security:http>

```



# 로그아웃의 처리와 LogoutSuccessHandler

```
<security:logout logout-  
url="/customLogout" invalidate-  
session="true" />
```

```
<h1> Logout Page</h1>  
  
<form action="/customLogout" method='post'>  
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />  
  <button>로그아웃</button>  
</form>
```

# JDBC를 이용하는 간편 인증/권한 처리

- 현실적으로 데이터베이스에 회원 정보를 이용해서 로그인 처리
- 패스워드는 PasswordEncoder를 지정해서 처리
- <https://github.com/spring-projects/spring-security/blob/master/core/src/main/java/org/springframework/security/provisioning/JdbcUserDetailsManager.java>

```
// GroupManager SQL
public static final String DEF_FIND_GROUPS_SQL = "select group_name from groups";
public static final String DEF_FIND_USERS_IN_GROUP_SQL = "select username from group_members gm, groups g "
    + "where gm.group_id = g.id" + " and g.group_name = ?";
public static final String DEF_INSERT_GROUP_SQL = "insert into groups (group_name) values (?)";
public static final String DEF_FIND_GROUP_ID_SQL = "select id from groups where group_name = ?";
public static final String DEF_INSERT_GROUP_AUTHORITY_SQL = "insert into group_authorities (group_id, authority) values (?,?)";
public static final String DEF_DELETE_GROUP_SQL = "delete from groups where id = ?";
public static final String DEF_DELETE_GROUP_AUTHORITIES_SQL = "delete from group_authorities where group_id = ?";
public static final String DEF_DELETE_GROUP_MEMBERS_SQL = "delete from group_members where group_id = ?";
public static final String DEF_RENAME_GROUP_SQL = "update groups set group_name = ? where group_name = ?";
public static final String DEF_INSERT_GROUP_MEMBER_SQL = "insert into group_members (group_id, username) values ( ?,? )";
public static final String DEF_DELETE_GROUP_MEMBER_SQL = "delete from group_members where group_id = ? and username = ?";
public static final String DEF_GROUP_AUTHORITIES_QUERY_SQL = "select g.id, g.group_name, ga.authority "
    + "from groups g, group_authorities ga "
    + "where g.group_name = ? "
    + "and g.id = ga.group_id ";
public static final String DEF_DELETE_GROUP_AUTHORITY_SQL = "delete from group_authorities where group_id = ? and authority = ?";
```

# 회원 테이블의 설계

```
create table users(
    username varchar2(50) not null primary key,
    password varchar2(50) not null,
    enabled char(1) default '1');

create table authorities (
    username varchar2(50) not null,
    authority varchar2(50) not null,
    constraint fk_authorities_users foreign key(username) references users(username));

create unique index ix_auth_username on authorities (username,authority);

insert into users (username, password) values ('user00','pw00');
insert into users (username, password) values ('member00','pw00');
insert into users (username, password) values ('admin00','pw00');

insert into authorities (username, authority) values ('user00','ROLE_USER');
insert into authorities (username, authority) values ('member00','ROLE_MANAGER');
insert into authorities (username, authority) values ('admin00','ROLE_MANAGER');
insert into authorities (username, authority) values ('admin00','ROLE_ADMIN');
commit;
```

# Security-context 설정

```
<security:authentication-manager>

    <security:authentication-provider>
        <security:jdbc-user-service data-source-ref="dataSource" />
    </security:authentication-provider>

</security:authentication-manager>
```

# PasswordEncoder의 설정

- 4버전까지는 위와 같이 별도의 PasswordEncoder를 이용하고 싶지 않을 때 NoOpPasswordEncoder를 이용해서 처리할 수 있었지만, 5버전부터는 Deprecated되어서 더 이상 사용할 수 없음으로 주의
- 암호화를 피하고 싶다면 직접 PasswordEncoder를 구현

```
<security:authentication-manager>

    <security:authentication-provider>
        <security:jdbc-user-service
            data-source-ref="dataSource" />

        <security:password-encoder
            ref="customPasswordEncoder" />

    </security:authentication-provider>

</security:authentication-manager>-
```

# 기존의 테이블을 이용하는 경우

```
create table tbl_member(
    userid varchar2(50) not null primary key,
    userpw varchar2(100) not null,
    username varchar2(100) not null,
    regdate date default sysdate,
    updatedate date default sysdate,
    enabled char(1) default '1');

create table tbl_member_auth (
    userid varchar2(50) not null,
    auth varchar2(50) not null,
    constraint fk_member_auth foreign key(userid) references tbl_member(userid)
);
```

# BCryptPasswordEncoder

- bcrypt는 태생 자체가 패스워드를 저장하는 용도로 설계된 해시 함수로 특정 문자열을 암호화하고, 체크하는 쪽에서는 암호화된 패스워드가 가능한 패스워드인지만 확인

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

```
<security:authentication-provider>
    <security:jdbc-user-service
        data-source-ref="dataSource" />

    <security:password-encoder
        ref="bcryptPasswordEncoder" />

</security:authentication-provider>

</security:authentication-manager>
```

# 인코딩된 패스워드를 가지는 사용자 생성

- 테스트 코드를 이용해서 패스워드를 인코딩한 후 insert

```
@Setter(onMethod_ = @Autowired)
private PasswordEncoder pwencoder;

@Setter(onMethod_ = @Autowired)
private DataSource ds;

...
stmt.setString(2, pwencoder.encode("pw" + i));
```

# 쿼리를 이용하는 인증

```
<security:jdbc-user-service  
    data-source-ref="dataSource"  
  
    users-by-username-query="select userid , userpw , enabled from tbl_member where userid = ? "  
    authorities-by-username-query="select userid, auth from tbl_member_auth where userid = ? " />  
  
<!-- <security:password-encoder ref="customPasswordEncoder" /> -->  
  
<security:password-encoder  
    ref="bcryptPasswordEncoder" />
```

The screenshot shows a browser window titled "Custom Login Page". In the input fields, "admin90" is entered into the "userid" field and "...." is entered into the "userpw" field. Below the inputs is a "제출" (Submit) button. To the right of the browser window is a Java console window displaying the following log output:

```
INFO : jdbc.sqltiming - select userid , userpw , enabled from tbl_member where userid = 'admin90'  
{executed in 1 msec}  
INFO : jdbc.resultset - 6. ResultSet.new ResultSet returned  
INFO : jdbc.audit - 6. PreparedStatement.executeQuery() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@255b5041  
INFO : jdbc.resultset - 6. ResultSet.next() returned true  
INFO : jdbc.resultset - 6. ResultSet.getString(1) returned admin90  
INFO : jdbc.resultset - 6. ResultSet.getString(2) returned $2a$10$MZvuOyEysYp9NlVdcyNu5h6i.ldfXrTZ.J6h9K1AHNDtiDwGls  
INFO : jdbc.resultset - 6. ResultSet.getBoolean(3) returned true  
INFO : jdbc.resultsettable -  
|-----|-----|-----|  
|userid |userpw |enabled |  
|-----|-----|-----|  
|admin90|$2a$10$MZvuOyEysYp9NlVdcyNu5h6i.ldfXrTZ.J6h9K1AHNDtiDwGls|true|  
|-----|-----|-----|  
INFO : jdbc.sqltiming - select userid, auth from tbl_member_auth where userid = 'admin90'  
{executed in 0 msec}  
INFO : jdbc.resultset - 6. ResultSet.new ResultSet returned  
INFO : jdbc.audit - 6. PreparedStatement.executeQuery() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@39706d6d  
INFO : jdbc.resultset - 6. ResultSet.next() returned true  
INFO : jdbc.resultset - 6. ResultSet.getString(2) returned ROLE_ADMIN  
INFO : jdbc.resultsettable -  
|-----|-----|  
|userid |auth |  
|-----|-----|  
|[unread]|ROLE_ADMIN|  
|-----|-----|
```

# 커스텀 UserDetailsService 활용

- 사용자가 원하는 방식으로 인가/인증 처리를 하기 위해서는 직접 UserDetailsService 인터페이스를 구현해서 처리

Method Summary	
All Methods	Instance Methods
Modifier and Type	Method and Description
UserDetails	<code>loadUserByUsername(java.lang.String username)</code> Locates the user based on the username.

`loadUserByUsername()`이라는 메서드의 반환 타입인 `UserDetails` 역시 인터페이스로, 사용자의 정보와 권한 정보 등을 담는 타입

UserDetails 타입은 getAuthorities( ), getPassword( ), getUsername( ) 등의 여러 추상 메서드를 가지고 있어서, 개발 전에 이를 직접 구현할 것인지 UserDetails 인터페이스를 구현해둔 스프링 시큐리티의 여러 하위 클래스를 이용할 것인지 판단해야 함

# 회원 도메인, 회원 Mapper 설계

```
src/main/java  
  org.zerock.controller  
  org.zerock.domain  
    AuthVO.java  
    MemberVO.java
```

```
package org.zerock.domain;  
  
import java.util.Date;  
import java.util.List;  
  
import lombok.Data;  
  
@Data  
public class MemberVO {  
  
    private String userid;  
    private String userpw;  
    private String userName;  
    private boolean enabled;  
  
    private Date regDate;  
    private Date updateDate;  
    private List<AuthVO> authList;  
}
```

```
package org.zerock.domain;  
  
import lombok.Data;  
  
@Data  
public class AuthVO {  
  
    private String userid;  
    private String auth;  
}
```

# MemberMapper

- MyBatis의 <resultMap>을 활용해 Join처리시에 발생하는 1:N 문제를 해결

```
<mapper namespace= "org.zerock.mapper.MemberMapper">

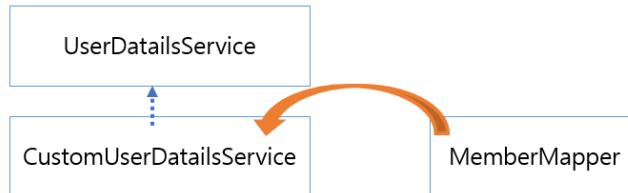
<resultMap type= "org.zerock.domain.MemberVO" id= "memberMap">
    <id property= "userid" column= "userid"/>
    <result property= "userid" column= "userid"/>
    <result property= "userpw" column= "userpw"/>
    <result property= "userName" column= "username"/>
    <result property= "regDate" column= "regdate"/>
    <result property= "updateDate" column= "updatedate"/>
    <collection property= "authList" resultMap= "authMap">
        </collection>
    </resultMap>

    <resultMap type= "org.zerock.domain.AuthVO" id= "authMap ">
        <result property= "userid" column= "userid"/>
        <result property= "auth" column= "auth"/>
    </resultMap>

    <select id= "read" resultMap= "memberMap">
SELECT
    mem.userid, userpw, username, enabled, regdate, updatedate, auth
FROM
    tbl_member mem LEFT OUTER JOIN tbl_member_auth auth on mem.userid = auth.userid
WHERE mem.userid = #{userid}
    </select>
</mapper>
```

# CustomUserDetailsService 구성

- CustomUserDetailsService는 스프링 시큐리티의 UserDetailsService를 구현하고, MemberMapper 타입의 인스턴스를 주입 받아서 실제 기능을 구현



```
@Log4j
public class CustomUserDetailsService implements UserDetailsService {

    @Setter(onMethod_ = { @Autowired })
    private MemberMapper memberMapper;

    @Override
    public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {

        Log.warn("Load User By UserName : " + userName);

        return null;
    }
}
```

# security-context.xml의 설정

- 변경된 방식으로 로그인 처리를 하는지 우선적으로 확인

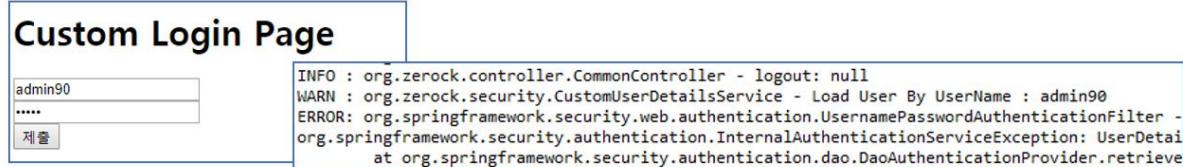
```
<security:authentication-manager>

    <security:authentication-provider
        user-service-ref="customUserDetailsService">

        <security:password-encoder
            ref="bcryptPasswordEncoder" />

    </security:authentication-provider>

</security:authentication-manager>
```



# MemberVO를 UserDetails 타입으로 변환하기

- MemberVO에 UserDetails 인터페이스를 추가하거나 확장하는 방식을 사용

```
public class CustomUser extends User {

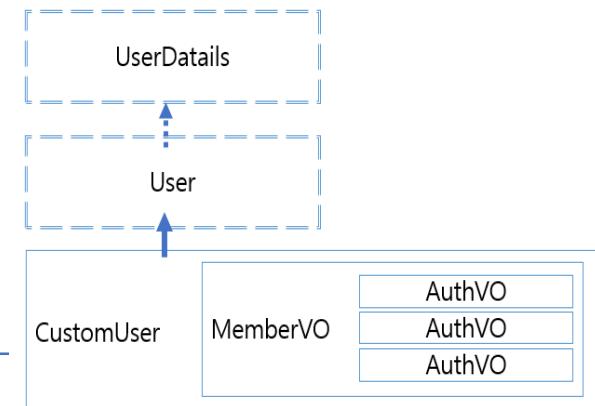
    private static final long serialVersionUID = 1L;

    private MemberVO member;

    public CustomUser(String username, String password, Collection<? extends GrantedAuthority> authorities) {
        super(username, password, authorities);
    }

    public CustomUser(MemberVO vo) {
        super(vo.getUserid(), vo.getUserpw(), vo.getAuthList().stream()
            .map(auth -> new SimpleGrantedAuthority(auth.getAuth()))).collect(Collectors.toList()));

        this.member = vo;
    }
}
```



```
@Log4j
public class CustomUserDetailsService implements UserDetailsService {

    @Setter(onMethod_ = { @Autowired })
    private MemberMapper memberMapper;

    @Override
    public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {

        Log.warn("Load User By UserName : " + userName);

        // userName means userid
        MemberVO vo = memberMapper.read(userName);

        Log.warn("queried by member mapper: " + vo);

        return vo == null ? null : new CustomUser(vo);
    }
}
```

# 스프링 시큐리티를 JSP에서 활용하기

- JSP에서는 시큐리티 태그들을 이용해서 처리
- <sec:authentication> 태그와 principal이라는 이름의 속성을 사용

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>

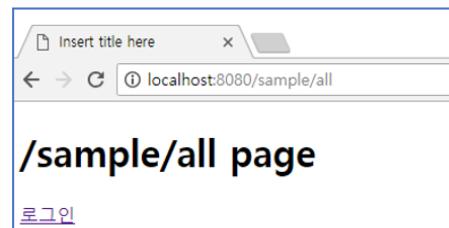
<p>principal : <sec:authentication property="principal"/></p>
<p>MemberVO : <sec:authentication property="principal.member"/></p>
<p>사용자이름 : <sec:authentication property="principal.member.userName"/></p>
<p>사용자아이디 : <sec:authentication property="principal.username"/></p>
<p>사용자 권한 리스트 : <sec:authentication property="principal.member.authList"/></p>
```

# 표현식을 이용하는 동적 화면 구성

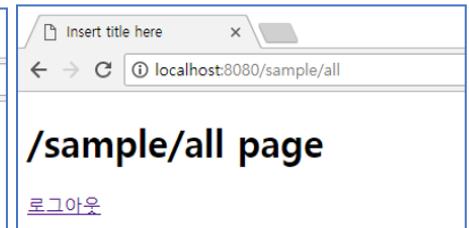
표현식	설명
<code>hasRole( [role] )</code> <code>hasAuthority( [authority] )</code>	해당 권한이 있으면 true
<code>hasAnyRole( [role,role2])</code> <code>hasAnyAuthority([authority])</code>	여러 권한들 중에서 하나라도 해당하는 권한이 있으면 true
<code>principal</code>	현재 사용자 정보를 의미
<code>permitAll</code>	모든 사용자에게 허용
<code>denyAll</code>	모든 사용자에게 거부
<code>isAnonymous( )</code>	익명의 사용자의 경우(로그인을 하지 않은 경우도 해당)
<code>isAuthenticated( )</code>	인증된 사용자면 true
<code>isFullyAuthenticated( )</code>	Remember-me로 인증된 것이 아닌 인증된 사용자인 경우 true

```
<sec:authorize access="isAnonymous()">  
  <a href="/customLogin">로그인</a>  
</sec:authorize>  
  
<sec:authorize access="isAuthenticated()">  
  <a href="/customLogout">로그아웃</a>  
</sec:authorize>
```

로그인하지 않은 사용자



로그인된 사용자



# 자동 로그인(remember-me)

- 스프링 시큐리티의 경우 'remember-me' 기능을 메모리상에서 처리하거나, 데이터베이스를 이용하는 형태로 약간의 설정만으로 구현이 가능
- security-context.xml에는 <security:remember-me> 태그를 이용해서 기능을 구현
- <security:remember-me>에는 아래와 같이 여러 속성
  - key: 쿠키에 사용되는 값을 암호화하기 위한 키(key)값
  - data-source-ref: DataSource를 지정하고 테이블을 이용해서 기존 로그인 정보를 기록(옵션)
  - remember-me-cookie: 브라우저에 보관되는 쿠키의 이름을 지정합니다. 기본값은 'remember-me'입니다.
  - remember-me-parameter: 웹 화면에서 로그인할 때 'remember-me'는 대부분 체크박스를 이용해서 처리합니다. 이 때 체크박스 태그의 name속성을 의미합니다.
  - tokenValiditySeconds: 쿠키의 유효시간을 지정합니다.

# 데이터베이스를 이용하는 자동로그인

- 별도의 코드 생성없이 테이블 생성만으로도 처리 가능

```
create table persistent_logins (
    username varchar(64) not null,
    series varchar(64) primary key,
    token varchar(64) not null,
    last_used timestamp not null);
```

```
<security:remember-me
    data-source-ref="dataSource" token-validity-seconds="604800" />
```

# 로그인 화면에서 자동 로그인

```
<form method='post' action="/Login">

<div>
  <input type='text' name='username' value='admin'>
</div>
<div>
  <input type='password' name='password' value='admin'>
</div>
<div>
  <input type='checkbox' name='remember-me'> Remember Me
</div>

<div>
  <input type='submit'>
</div>
<input type="hidden" name="${_csrf.parameterName}"
       value="${_csrf.token}" />
```

The screenshot illustrates the process of automatic login. On the left, a custom login page titled "Custom Login Page" is displayed. It contains two text input fields for "username" and "password", both pre-filled with "admin". A checkbox labeled "Remember Me" is checked. On the right, the browser's developer tools are open, specifically the Network tab, which shows the current session cookies. Two cookies are listed:

Name	Value	Domain	Path	Expires	Size	HTTP
JSESSIONID	675E1947B03A0DF7C9F6C67B5D34A1BC	localhost	/	1969-01-01T00:00:00Z	42	✓
remember-me	a2JasVFLR1VuZGdkMUsxSV09SMIQzUSUzRCU...	localhost	/	2018-01-01T00:00:00Z	90	✓

# 로그아웃 처리

- 자동 로그인 기능을 이용하는 경우에 사용자가 로그아웃을 하면 기존과 달리 자동 로그인에 사용하는 쿠키도 삭제해 주도록 쿠키를 삭제하는 항목을 security-context.xml에 지정

```
<security:logout logout-url="/customLogout"  
invalidate-session="true" delete-cookies="remember-me,JSESSION_ID" />
```