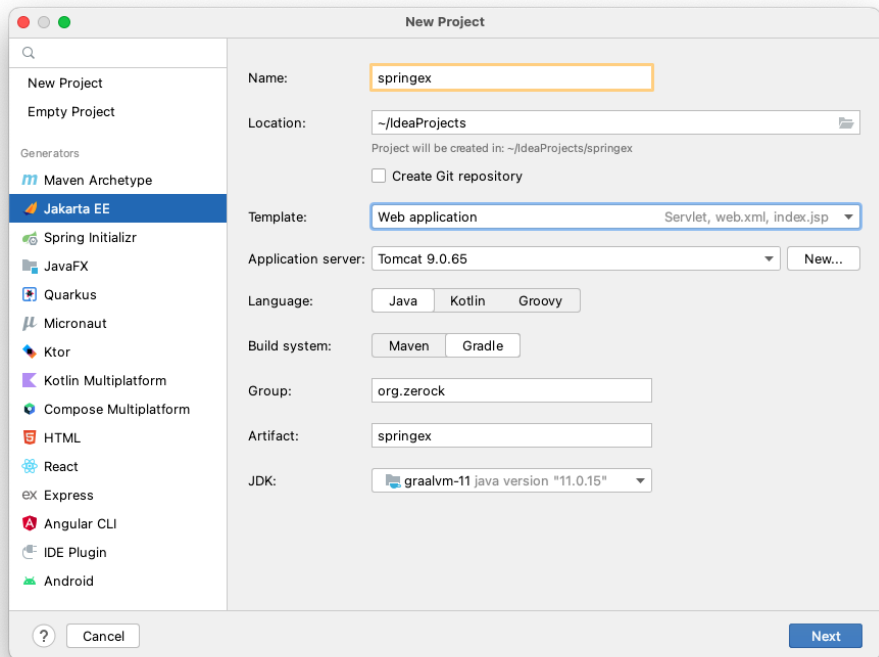


PART4 - 스프링

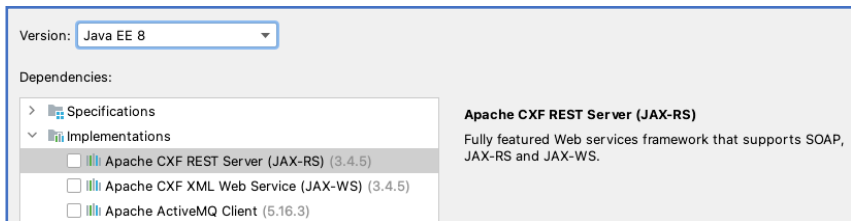
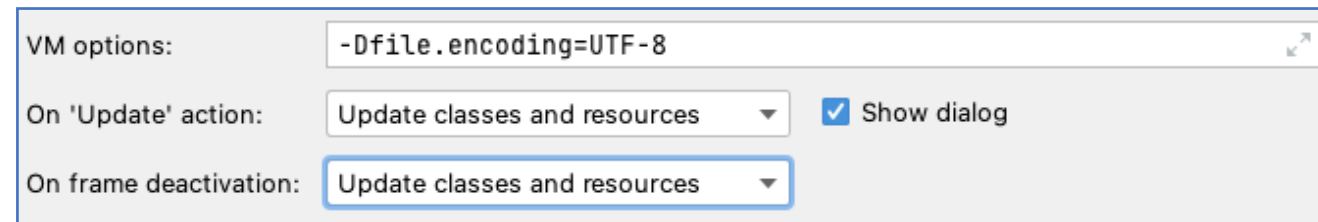
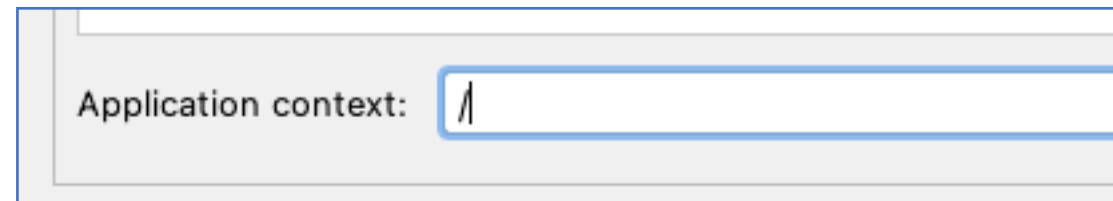
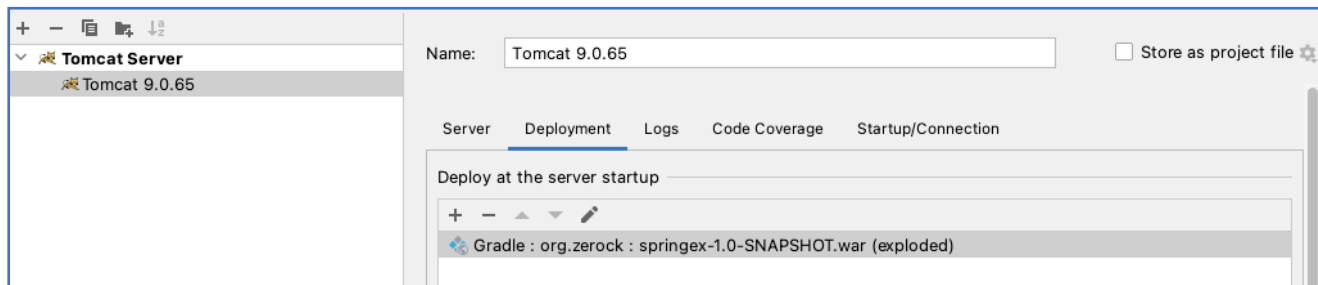
스프링

- 2000년대 등장한 경량(lightweight) 프레임워크의 일종
- Spring core에 여러 서브 프로젝트의 결합 형태로 구성
 - Spring Web MVC
 - Spring Data JDBC
- 다른 프레임워크를 배척하지 않고 통합구성 가능
- 의존성 주입을 통한 객체지향 구성
 - Dependency injections
 - 현재 객체의 조력 객체를 외부에서 주입해 주는 방식의 구성
 - 제어의 역행

프로젝트의 생성



Tomcat 조정



Spring 프레임워크 라이브러리 추가

- 경량 프레임워크들은 대부분 jar파일의 형태로 구성
- 'spring - core' 라이브러리 추가

 **Spring Core**
Spring Core

License	Apache 2.0
Categories	Core Utilities
Tags	spring
Used By	7,024 artifacts

[Central \(221\)](#) [AtlassianPkgs \(1\)](#) [Atlassian 3rd-P Old \(3\)](#) [Spring Plugins \(51\)](#) [Spring Lib M \(3\)](#) [Spring Milestones \(7\)](#)
[ICM \(16\)](#) [Grails Core \(5\)](#) [Geomajas \(1\)](#) [EBIPublic \(5\)](#) [Alfresco \(11\)](#) [Cambridge \(1\)](#) [Gradle Releases \(1\)](#)

Version	Vulnerabilities	Repository	Usage
5.3.16		Central	128
5.3.15		Central	591

```
dependencies {  
    compileOnly('javax.servlet:javax.servlet-api:4.0.1')
```

```
    testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
```

```
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")
```

```
    implementation group: 'org.springframework', name: 'spring-core', version: '5.3.16'
```

```
    implementation group: 'org.springframework', name: 'spring-context', version: '5.3.16'
```

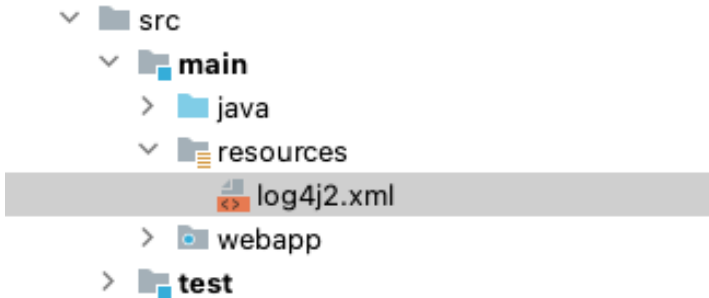
```
    implementation group: 'org.springframework', name: 'spring-test', version: '5.3.16'
```

```
}
```

Lombok/Log4j2/JSTL추가

```
compileOnly 'org.projectlombok:lombok:1.18.22'  
annotationProcessor 'org.projectlombok:lombok:1.18.22'  
testCompileOnly 'org.projectlombok:lombok:1.18.22'  
testAnnotationProcessor 'org.projectlombok:lombok:1.18.22'
```

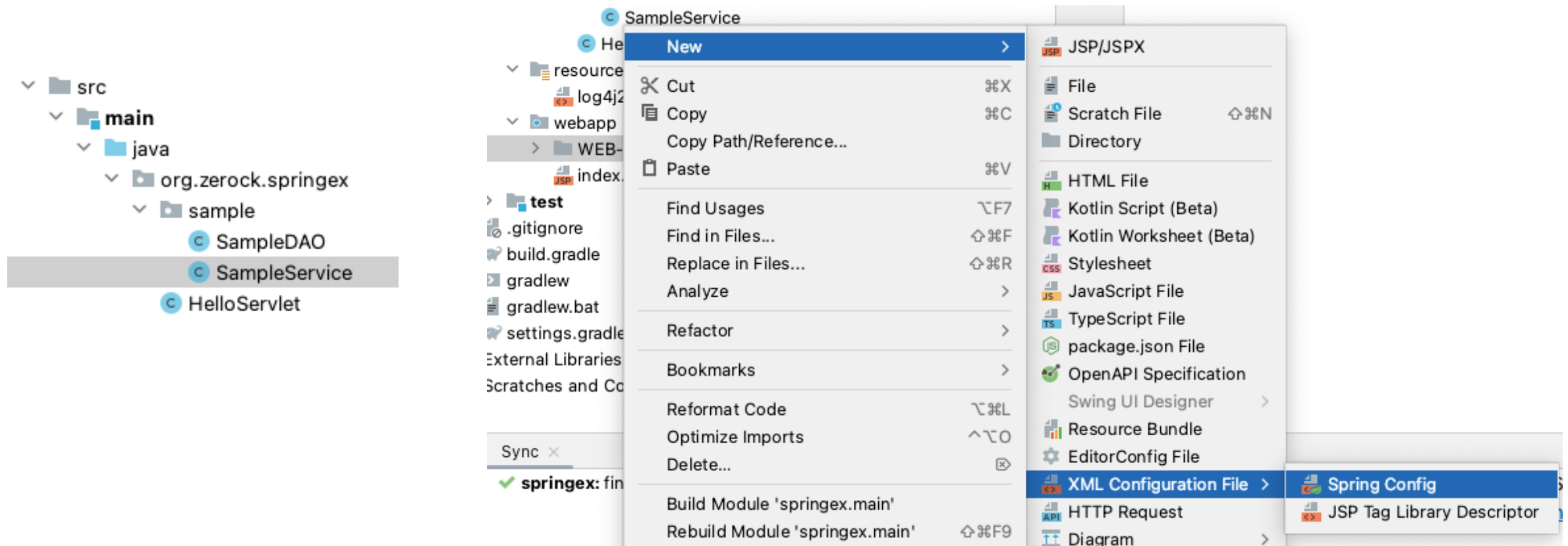
```
implementation group: 'org.apache.logging.log4j', name: 'log4j-core', version: '2.16.0'  
implementation group: 'org.apache.logging.log4j', name: 'log4j-api', version: '2.16.0'  
implementation group: 'org.apache.logging.log4j', name: 'log4j-slf4j-impl', version: '2.16.0'  
  
implementation group: 'jstl', name: 'jstl', version: '1.2'
```



```
<?xml version="1.0" encoding="UTF-8"?>  
  
<configuration status="INFO">  
  
  <Appenders>  
    <!-- 콘솔 -->  
    <Console name="console" target="SYSTEM_OUT">  
      <PatternLayout charset="UTF-8" pattern="%d{hh:mm:ss} %5p [%c] %m%n"/>  
    </Console>  
  </Appenders>  
  
  <loggers>  
    <logger name="org.springframework" level="INFO" additivity="false">  
      <appender-ref ref="console" />  
    </logger>  
  
    <logger name="org.zerock" level="INFO" additivity="false">  
      <appender-ref ref="console" />  
    </logger>  
  
    <root level="INFO" additivity="false">  
      <AppenderRef ref="console"/>  
    </root>  
  
  </loggers>
```

의존성 주입 실습

- 서비스 객체(SampleService)에 DAO 객체의 주입 예제
- 클래스 생성 후 스프링의 설정 파일 추가



root-context.xml

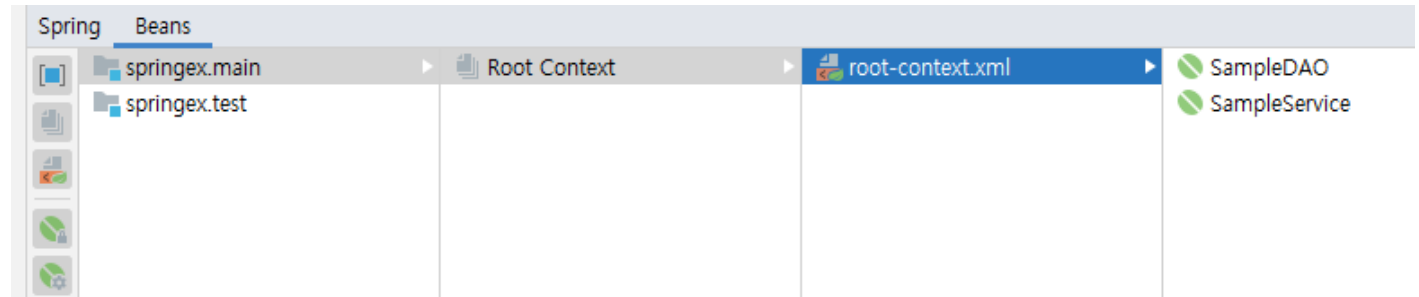
- 일반적으로 스프링 프레임워크 이용시 사용하는 기본 설정 파일
- 주로 POJO에 대한 설정
- 별도의 라이브러리들을 활용하는 경우에는 별도의 파일을 추가하는 방식을 이용

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

    <bean class="org.zerock.springex.sample.SampleDAO"></bean>

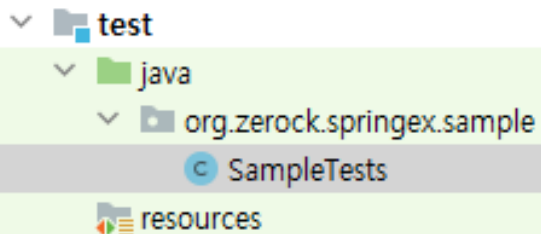
    <bean class="org.zerock.springex.sample.SampleService"></bean>

</beans>
```



설정 테스트

- 스프링이 관리하는 객체를 빈(bean)
- 테스트 코드를 추가해서 설정에 문제가 없는지 확인



```
package org.zerock.springex.sample;

import lombok.extern.log4j.Log4j2;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@Log4j2
@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations="file:src/main/webapp/WEB-INF/root-context.xml")
public class SampleTests {

    @Autowired
    private SampleService sampleService;

    @Test
    public void testService1() {
        log.info(sampleService);
        Assertions.assertNotNull(sampleService);
    }
}
```

```
[org.springframework.test.context.support.DefaultTestContextBootstrapper] Loaded default TestExecutionListene
[org.springframework.test.context.support.DefaultTestContextBootstrapper] Using TestExecutionListeners: [org.
[org.zerock.springex.sample.SampleTests] org.zerock.springex.sample.SampleService@4b54af3d
```

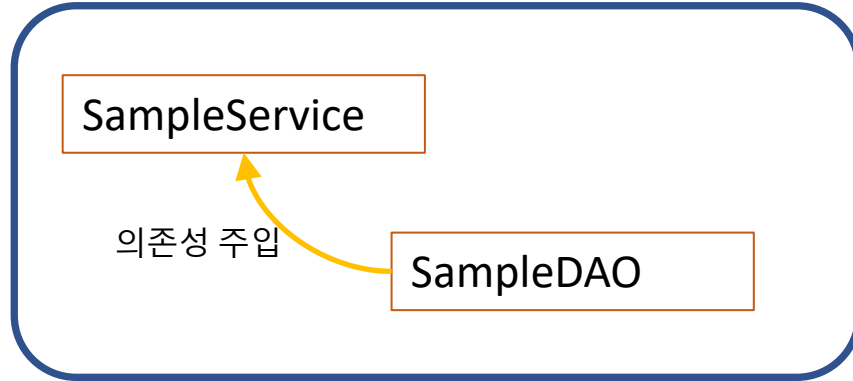

ApplicationContext와 빈(Beans)

- 스프링이 빈들을 관리하는 공간 – ApplicationContext
- root-context.xml을 읽어서 해당 클래스들을 인스턴스화 시켜서 ApplicationContext 내부에서 관리

ApplicationContext



ApplicationContext



@Autowired의 의미와 필드 주입

- @Autowired 가 있는 필드의 경우 해당 타입의 객체가 스프링의 컨텍스트내 존재한다면 실행시 주입된다.

```
@Log4j2
@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations="file:src/main/webapp/WEB-INF/root-context.xml")
public class SampleTests {

    @Autowired
    private SampleService sampleService;

    @Test
    public void testService1() {
        log.info(sampleService);
        Assertions.assertNotNull(sampleService);
    }
}
```

의존성 주입

ApplicationContext

SampleService

SampleDAO

SampleDAO 객체 주입 실습

```
package org.zerock.springex.sample;
```

```
import lombok.ToString;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
@ToString
```

```
public class SampleService {
```

```
    @Autowired
```

```
    private SampleDAO sampleDAO;
```

```
}
```

```
INFO [org.springframework.test.context.support.DefaultTestContextBootstrapper] Loaded default TestExecutionListener class
INFO [org.springframework.test.context.support.DefaultTestContextBootstrapper] Using TestExecutionListeners: [org.spring
INFO [org.zerock.springex.sample.SampleTests] SampleService(sampleDAO=org.zerock.springex.sample.SampleDAO@203c20cf)
```

<context:component-scan>

- 패키지를 지정해서 해당 패키지내 클래스의 인스턴스들을 스프링의 빈으로 등록하기 위해서 사용
- 특정 어노테이션을 이용해서 스프링의 빈으로 관리될 객체를 표시
 - @Controller
 - @Service
 - @Repository
 - @Component

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

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

</beans>
```

```
package org.zerock.springex.sample;

import org.springframework.stereotype.Repository;

@Repository
public class SampleDAO {
}
```

```
package org.zerock.springex.sample;

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

@Service
@ToString
public class SampleService {

  @Autowired
  private SampleDAO sampleDAO;
}
```

생성자 주입 방식

- 주입받는 타입을 final로 선언하고 생성자를 통해서 의존성 주입
- lombok의 @RequiredArgsConstructor 를 통해서 생성자 자동 생성

```
package org.zerock.springex.sample;

import lombok.RequiredArgsConstructor;
import lombok.ToString;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
@ToString
@RequiredArgsConstructor
public class SampleService {

    private final SampleDAO sampleDAO;

}
```

```
package org.zerock.springex.sample;

import org.springframework.stereotype.Repository;

@Repository
public class SampleDAO {

}
```

인터페이스를 이용한 느슨한 결합

```
package org.zerock.springex.sample;
```

```
public interface SampleDAO {  
}
```

```
▼ java  
  ▼ org.zerock.springex  
    ▼ sample  
      I SampleDAO  
      c SampleDAOImpl  
      c SampleService  
      c HelloServlet
```

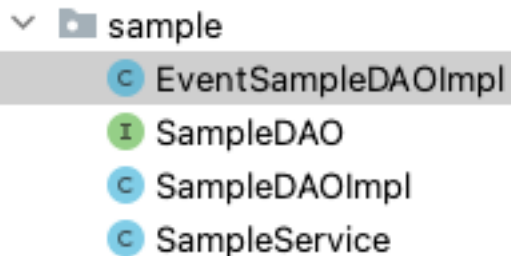
```
package org.zerock.springex.sample;
```

```
import org.springframework.stereotype.Repository;
```

```
@Repository
```

```
public class SampleDAOImpl implements SampleDAO{  
}
```

다른 SampleDAO 객체로 변경해 보기

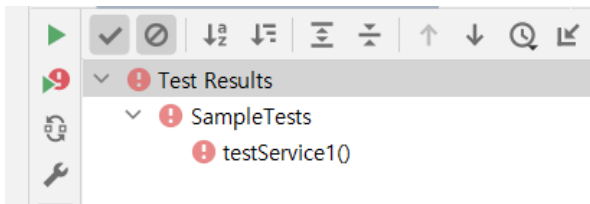


sample

- EventSampleDAOImpl
- SampleDAO
- SampleDAOImpl
- SampleService

```
package org.zerock.springex.sample;  
  
import org.springframework.stereotype.Repository;  
  
@Repository  
public class EventSampleDAOImpl implements SampleDAO {  
}
```

SampleDAO타입의 빈이 두개가 되는 상황이 되므로 에러 발생



Caused by: org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'sampleService'
at org.springframework.beans.factory.support.ConstructorResolver.createArgumentArray(ConstructorResolver.java:800) ~[spring-beans-5
at org.springframework.beans.factory.support.ConstructorResolver.autowireConstructor(ConstructorResolver.java:229) ~[spring-beans-5

@Primary

@Qualifier

등을 이용해서 동일한 타입의 객체가 여러개 일 경우 특정 클래스를 지정 가능

웹 프로젝트를 위한 스프링 준비

- 스프링의 ApplicationContext는 여러 빈들을 관리
- Web의 경우 web.xml을 이용해서
- 리스너를 통해서 ApplicationContext등록

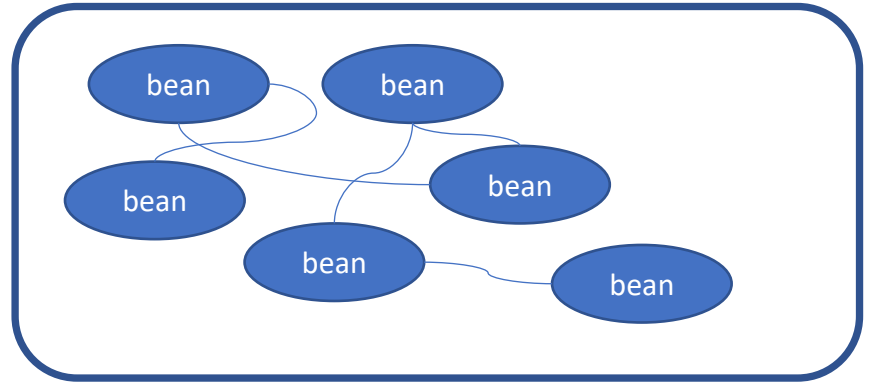
> test

build.gradle

gradlew

```
implementation group: 'org.springframework', name: 'spring-core', version: '5.3.16'  
implementation group: 'org.springframework', name: 'spring-context', version: '5.3.16'  
implementation group: 'org.springframework', name: 'spring-test', version: '5.3.16'  
implementation group: 'org.springframework', name: 'spring-webmvc', version: '5.3.16'
```

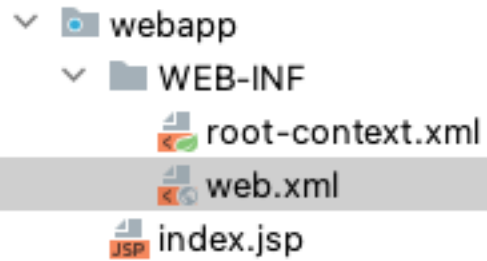
ApplicationContext



web.xml의 설정

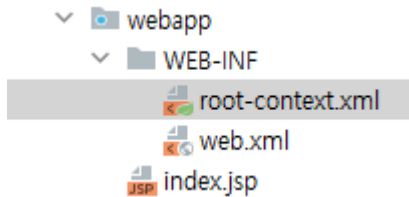
```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/root-context.xml</param-value>
</context-param>

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



DataSource 구성하기

```
dependencies {  
    compileOnly('javax.servlet:javax.servlet-api:4.0.1')  
  
    ...생략...  
    implementation 'org.mariadb.jdbc:mariadb-java-client:3.0.3'  
    implementation group: 'com.zaxxer', name: 'HikariCP', version: '5.0.1'  
}
```



```
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">  
    <property name="driverClassName" value="org.mariadb.jdbc.Driver"></property>  
    <property name="jdbcUrl" value="jdbc:mariadb://localhost:3306/webdb"></property>  
    <property name="username" value="webuser"></property>  
    <property name="password" value="webuser"></property>  
    <property name="dataSourceProperties">  
        <props>  
            <prop key="cachePrepStmts">true</prop>  
            <prop key="prepStmtCacheSize">250</prop>  
            <prop key="prepStmtCacheSqlLimit">2048</prop>  
        </props>  
    </property>  
</bean>  
  
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"  
    destroy-method="close">  
    <constructor-arg ref="hikariConfig" />  
</bean>
```

```
@Log4j2
@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations="file:src/main/webapp/WEB-INF/root-context.xml")
public class SampleTests {
```

```
    @Autowired
    private SampleService sampleService;
```

```
    @Autowired
    private DataSource dataSource;
```

```
    @Test
    public void testService1() {
        log.info(sampleService);
        Assertions.assertNotNull(sampleService);
    }
```

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

```
        Connection connection = dataSource.getConnection();
        log.info(connection);
        Assertions.assertNotNull(connection);
        connection.close();
```

```
    }
```

```
}
```

```
INFO [org.springframework.test.context.support.DefaultTestContextBootstrapper] Loaded default TestExecutionListener class name
INFO [org.springframework.test.context.support.DefaultTestContextBootstrapper] Using TestExecutionListeners: [org.springframework
INFO [com.zaxxer.hikari.HikariDataSource] HikariPool-1 - Starting...
INFO [com.zaxxer.hikari.pool.HikariPool] HikariPool-1 - Added connection org.mariadb.jdbc.Connection@63411512
INFO [com.zaxxer.hikari.HikariDataSource] HikariPool-1 - Start completed.
INFO [org.zerock.springex.sample.SampleTests] HikariProxyConnection@1118998513 wrapping org.mariadb.jdbc.Connection@63411512
```

MyBatis와 스프링 연동

- 'Sql Mapping Framework' - SQL의 처리를 객체와 매핑해서 처리
- JDBC를 이용해서 PreparedStatement/ResultSet에 대한 객체 처리를 자동으로 수행
- Connection등의 JDBC자원들에 대한 자동 close()
- SQL은 별도의 XML등을 이용해서 분리

MyBatis와 스프링의 연동 방식

- MyBatis는 단독으로 실행이 가능한 프레임워크지만 mybatis-spring 라이브러리를 이용하면 쉽게 통합해서 사용 가능
- 과거에는 주로 별도의 DAO(Data Access Object)를 구성하고 여기서 MyBatis의 SqlSession을 이용하는 방식
- 최근에는 MyBatis는 인터페이스를 이용하고 실제 코드는 자동으로 생성되는 방식 – Mapper인터페이스와 XML
- 필요한 라이브러리
 - 스프링 관련: spring-jdbc, spring-tx
 - MyBatis 관련: mybatis, mybatis-spring

```
implementation group: 'org.springframework', name: 'spring-jdbc',  
version: '5.3.16'
```

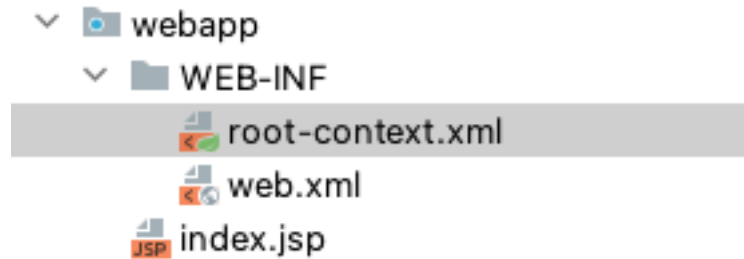
```
implementation group: 'org.springframework', name: 'spring-tx',  
version: '5.3.16'
```

```
implementation 'org.mybatis:mybatis:3.5.9'
```

```
implementation 'org.mybatis:mybatis-spring:2.0.7'
```

MyBatis의 SqlSessionFactory 설정

- MyBatis에서 실제 SQL의 처리는 SqlSessionFactory에서 생성하는 SqlSession을 통해서 수행됨

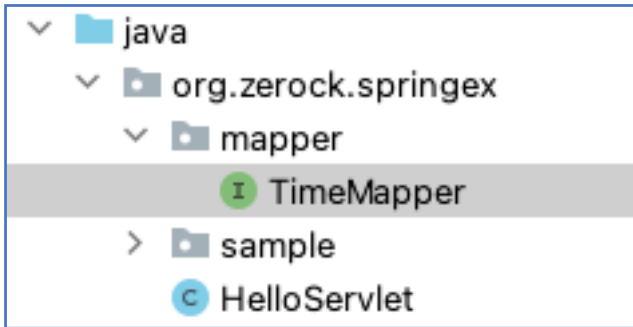


```
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"
    destroy-method="close">
    <constructor-arg ref="hikariConfig" />
</bean>

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

Mapper인터페이스 활용하기

- MyBatis를 통해서 수행해야 하는 기능을 매퍼 인터페이스로 작성
- 어노테이션 혹은 XML로 SQL 작성
- 스프링의 설정에서



```
package org.zerock.springex.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {

    @Select("select now()")
    String getTime();
}
```


root-context.xml 설정

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

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

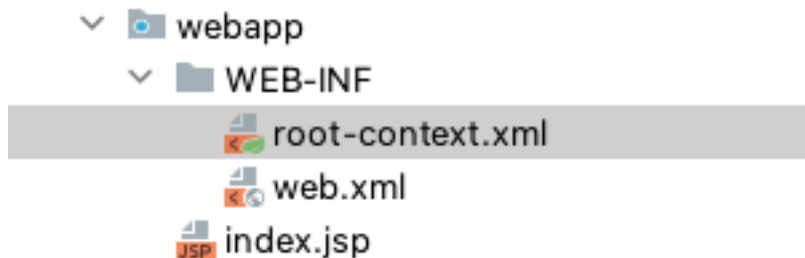
```
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <property name="driverClassName" value="org.mariadb.jdbc.Driver"></property>
    <property name="jdbcUrl" value="jdbc:mariadb://localhost:3306/webdb"></property>
    <property name="username" value="webuser"></property>
    <property name="password" value="webuser"></property>
    <property name="dataSourceProperties">
        <props>
            <prop key="cachePrepStmts">true</prop>
            <prop key="prepStmtCacheSize">250</prop>
            <prop key="prepStmtCacheSqlLimit">2048</prop>
        </props>
    </property>
</bean>
```

```
<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" />
</bean>
```

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

```
</beans>
```



XML로 SQL분리하기

- SQL이 길고 복잡한 경우 XML을 이용해서 SQL을 분리
- XML을 이용하는 방식
 - 매퍼인터페이스에 메소드를 선언
 - XML파일을 작성하고 메서드의 이름과 네임스페이스를 작성
 - <select>, <insert>.. 을 이용할때 id 속성값은 메서드의 이름으로 지정

org.zerock.springex
mapper
TimeMapper
TimeMapper2

```
package org.zerock.springex.mapper;  
  
public interface TimeMapper2 {  
  
    String getNow();  
}
```

resources
mappers
log4j2.xml
webapp
WEB-INF

resources
mappers
TimeMapper2.xml
log4j2.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.springex.mapper.TimeMapper2">  
  
    <select id="getNow" resultType="string">  
        select now()  
    </select>  
  
</mapper>
```

main
java
resources
webapp
WEB-INF

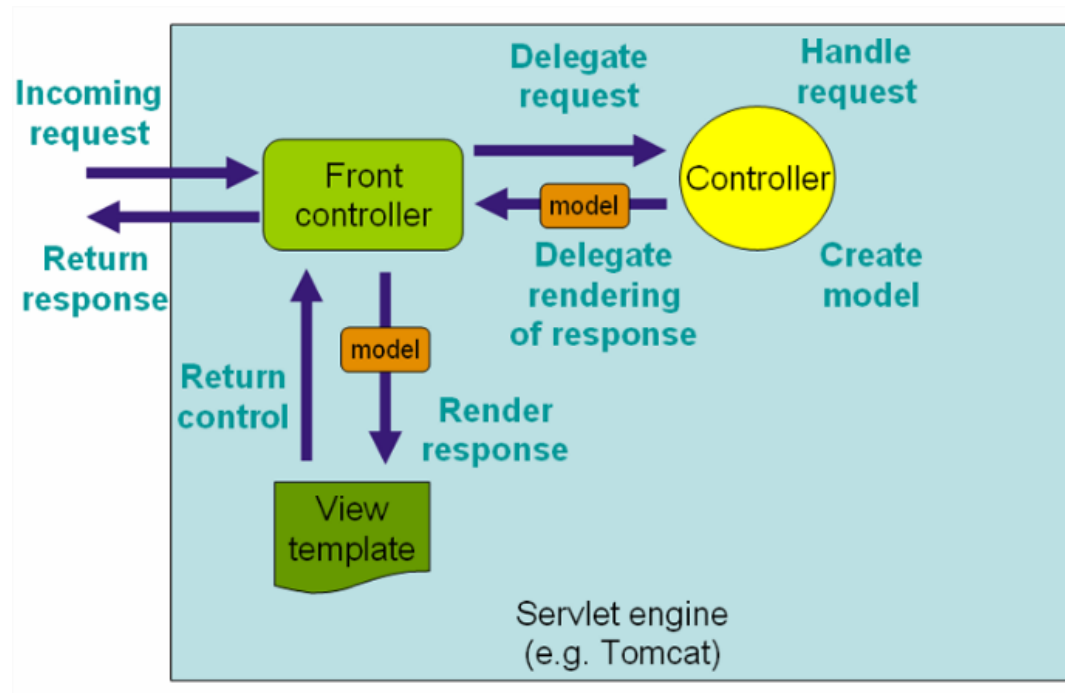
root-context.xml
web.xml

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


스프링 Web MVC 기초

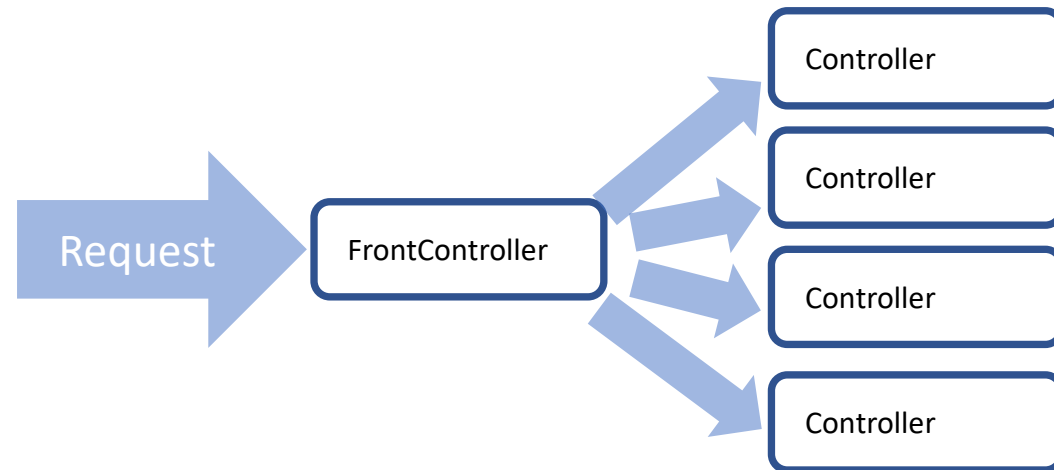
스프링 Web MVC의 특징

- 기존의 MVC구조에서 추가적으로 Front-Controller패턴 적용
- 어노테이션의 적극적인 활용
- 파라미터나 리턴타입에 대한 자유로운 형식
- 추상화된 api들의 제공



DispatcherServlet과 Front Controller

- Front Controller 패턴은 모든 요청을 하나의 컨트롤러를 거치는 구조로 일관된 흐름을 작성하는데 도움이 됨
- 스프링 Web MVC에서는 DispatcherServlet이 Front Controller



servlet-context.xml

- spring-core와 달리 웹과 관련된 처리를 분리하기 위해서 작성하는 설정파일
- 반드시 구분할 필요는 없으나 일반적으로 계층별로 분리하는 경우가 많음

▼ webapp
▼ WEB-INF

root-context.xml
servlet-context.xml
web.xml

▼ webapp

resources

▼ WEB-INF

root-context.xml
servlet-context.xml
web.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:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <mvc:annotation-driven></mvc:annotation-driven>

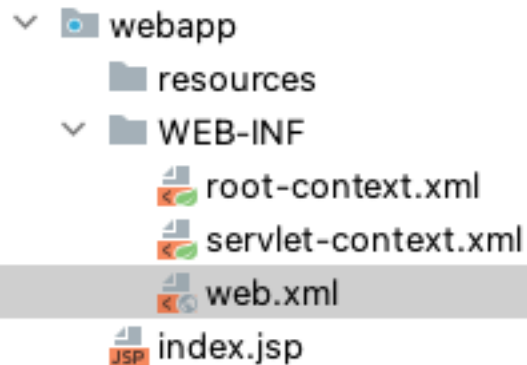
    <mvc:resources mapping="/resources/**" location="/resources/"></mvc:resources>

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

</beans>
```


web.xml 설정

- 스프링 Web MVC에서 사용하는 DispatcherServlet을 web.xml에 설정



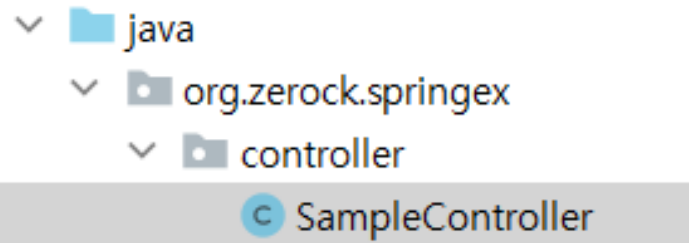
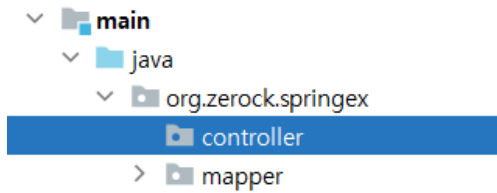
```
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

```
INFO [com.zaxxer.hikari.pool.HikariPool] HikariPool-1 - Added connection org.mariadb.jdbc.Connection@27eb
INFO [com.zaxxer.hikari.HikariDataSource] HikariPool-1 - Start completed.
INFO [org.springframework.web.context.ContextLoader] Root WebApplicationContext initialized in 388 ms
INFO [org.springframework.web.servlet.DispatcherServlet] Initializing Servlet 'appServlet'
```

실습 -스프링 MVC에서 컨트롤러

- 상속이나 인터페이스를 구현하는 방식을 사용하지 않고 어노테이션만으로 처리가 가능
- 오버라이드 없이 필요한 메서드들을 정의
- 메서드의 파라미터를 기본자료형이나 객체자료형을 마음대로 지정
- 메서드의 리턴타입도 void, String, 객체 등 다양한 타입을 사용할 수 있음



```
package org.zerock.springex.controller;

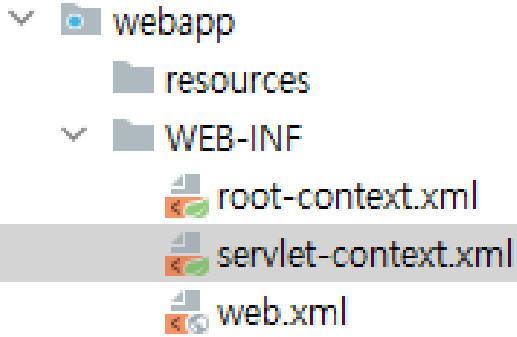
import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
@Log4j2
public class SampleController {

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

}
```

servlet-context.xml의 component-scan



```
<?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:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">

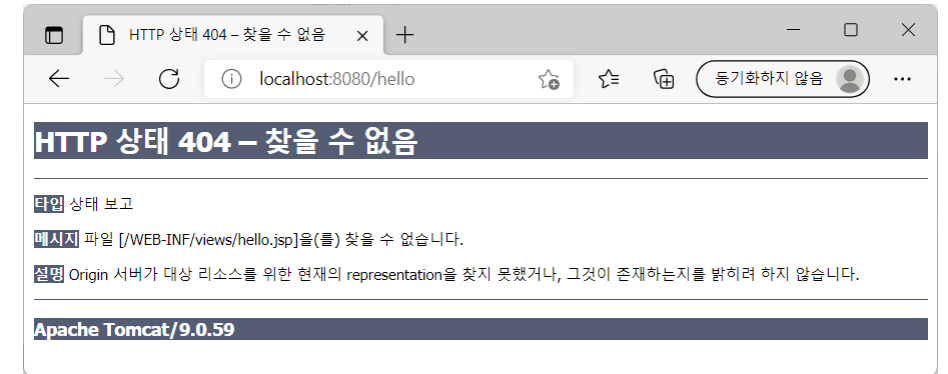
    <mvc:annotation-driven></mvc:annotation-driven>

    <mvc:resources mapping="/resources/**"
location="/resources/"></mvc:resources>

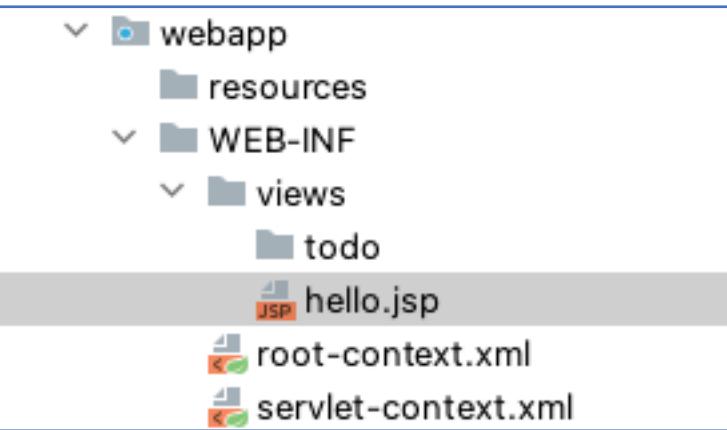
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
>
        <property name="prefix" value="/WEB-INF/views/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

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

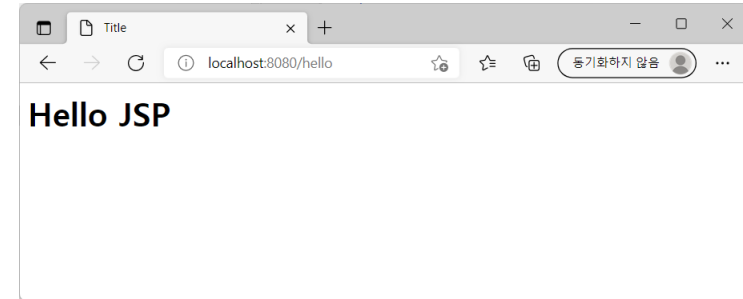
</beans>
```



화면 처리



```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Title</title>
</head>
<body>
<h1>Hello JSP </h1>
</body>
</html>
```



@RequestMapping과 파생 어노테이션들

- @RequestMapping은 경로를 처리하기 위한 용도로 사용
- 스프링 MVC의 경우 하나의 클래스로 여러 경로를 처리
- 4버전 이후 @GetMapping, @PostMapping 등이 추가

파라미터의 자동 수집과 변환

- DTO나 VO등으로 자동으로 HttpServletRequest의 파라미터 수집
 - 기본자료형의 경우는 자동으로 형 변환처리가 가능
 - 객체자료형의 경우는 setXXX()의 동작을 통해서 처리
 - 객체자료형의 경우 생성자가 없거나 파라미터가 없는 생성자가 필요
- @RequestParam: 파라미터의 이름을 지정하거나 기본값(defaultValue)를 지정할 수 있음
- @ModelAttribute를 이용해서 명시적으로 해당 파라미터를 view까지 전달하도록 구성할 수 있음

```
@GetMapping("/ex4")
public void ex4(Model model){

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

    model.addAttribute("message", "Hello World");
}
```

Formatter를 이용한 커스텀 처리

- 날짜나 형 변환을 커스터마이징 해야 하는 경우 주로 사용

▼ org.zerock.springex
▼ controller
▼ formatter
 LocalDateFormatter
 SampleController

```
package org.zerock.springex.controller.formatter;  
  
import org.springframework.format.Formatter;  
  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
import java.util.Locale;  
  
public class LocalDateFormatter implements Formatter<LocalDate> {  
  
    @Override  
    public LocalDate parse(String text, Locale locale) {  
        return LocalDate.parse(text, DateTimeFormatter.ofPattern("yyyy-MM-dd"));  
    }  
  
    @Override  
    public String print(LocalDate object, Locale locale) {  
        return DateTimeFormatter.ofPattern("yyyy-MM-dd").format(object);  
    }  
  
}
```


- ▼ WEB-INF
 - ▼ views
 - todo
 - hello.jsp
 - root-context.xml
 - servlet-context.xml

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
  <property name="prefix" value="/WEB-INF/views/"></property>  
  <property name="suffix" value=".jsp"></property>  
</bean>  
<mvc:annotation-driven conversion-service="conversionService" />
```

객체 자료형 파라미터 수집

- Java Beans 형식으로 만들어진 클래스 타입은 자동으로 객체가 생성되고 setXXX()등을 자동으로 호출
- Lombok의 @Setter 혹은 @Data를 활용

TodoDTO

▼ org.zerock.springex
 > controller
 ▼ dto
 TodoDTO
 > mapper

```
package org.zerock.springex.dto;

import lombok.*;
import java.time.LocalDate;

@ToString
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class TodoDTO {

    private Long tno;

    private String title;

    private LocalDate dueDate;

    private boolean finished;

    private String writer; // 새로 추가됨

}
```

Model이라는 특별한 파라미터

- 예전 서블릿에서 `request.setAttribute()`로 처리했던 모델대신 사용
- 메소드의 파라미터에 Model을 선언하면 자동으로 객체 생성
- `addAttribute()`를 이용해서 view까지 전달할 객체 저장

RedirectAttributes와 리다이렉션

- 스프링 MVC의 경우 반환타입이 문자열이고 redirect: 로 시작하는 경우 리다이렉트 처리
- RedirectAttributes는 리다이렉트시에 필요한 쿼리 스트링을 구성하기 위한 객체
- addFlashAttribute(): 일회성으로 전달되는 값 전달을 위해 사용
- addAttribute(): 리다이렉트시에 쿼리 스트링으로 작성되는 값

다양한 리턴타입

- 상속이나 인터페이스와 달리 다양한 리턴 타입을 사용할 수 있다.
 - void
 - String
 - 사용자 정의 타입
 - 배열/컬렉션
 - ResponseEntity 등

스프링 MVC의 예외처리

- @ControllerAdvice를 이용해서 처리
- 예외에 따라서 @ExceptionHandler를 메서드에 활용

controller

exception

CommonExceptionAdvice

formatter

LocalDateFormatter

```
package org.zerock.springex.controller.exception;

import lombok.extern.log4j.Log4j2;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

@ControllerAdvice
@Log4j2
public class CommonExceptionAdvice {

    @ResponseBody
    @ExceptionHandler(NumberFormatException.class)
    public String exceptNumber(NumberFormatException numberFormatException){

        log.error("-----");
        log.error(numberFormatException.getMessage());

        return "NUMBER FORMAT EXCEPTION";
    }

}
```

범용적인 예외처리

- 예외 발생시 상위 타입인 Exception을 이용해서 예외 메시지를 구성
- 디버깅 용도로 활용

```
@ResponseBody
@ExceptionHandler(Exception.class)
public String exceptCommon(Exception exception){

    log.error("-----");
    log.error(exception.getMessage());

    StringBuffer buffer = new StringBuffer("<ul>");

    buffer.append("<li>" +exception.getMessage()+"</li>");

    Arrays.stream(exception.getStackTrace()).forEach(stackTraceElement -> {
        buffer.append("<li>" +stackTraceElement+"</li>");
    });
    buffer.append("</ul>");

    return buffer.toString();
}
```

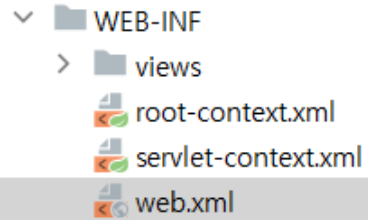

404에러

```
@ExceptionHandler({NoHandlerFoundException.class})
@ResponseStatus(HttpStatus.NOT_FOUND)
public String notFound(){

    return "custom404";
}
```



```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>Oops! 페이지를 찾을 수 없습니다!</h1>
</body>
</html>
```



```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/servlet-context.xml</param-value>
    </init-param>

    <init-param>
        <param-name>throwExceptionIfNoHandlerFound</param-name>
        <param-value>true</param-value>
    </init-param>

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


프로젝트의 구현 목표

Search

☐완료여부

☐제목 ☐작성자

년-월-일 년-월-일

Register

Todo List

Tno	Title	Writer	DueDate	Finished
1525	스프링 공부하기	user13	2022-01-20	false
1524	한글테스트	user1	2021-12-31	false
1523	한글테스트	user1	2021-12-31	false
1522	스프링 테스트	user00	2022-10-10	false
1521	스프링 공부하기	user13	2022-01-20	false
1520	한글테스트	user1	2021-12-31	true
1518	스프링 테스트	user00	2022-10-10	false
1516	한글테스트	user1	2021-12-31	true
1515	한글테스트	user1	2021-12-31	false
1514	스프링 테스트	user00	2022-10-10	false

1 2 3 4 5 6 7 8 9 10 Next

Navbar

Featured

Title 스프링 MVC 테스트

DueDate 2021-12-31

Writer user01

Footer

Navbar

Featured

TNO 1525

Title 스프링 공부하기

DueDate 2022-01-20

Writer user13

☐ Finished

Footer

Navbar

Featured

TNO 1525

Title 스프링 공부하기

DueDate 2022-01-20

Writer user13

☐ Finished

Footer

1) 검색과 필터링을 적용할 수 있는 화면을 구성하고
MyBatis의 동적 쿼리를 이용해서 상황에 맞는 Todo들을 검색합니다.

- 1) 새로운 Todo를 등록할 때 문자열/boolean/LocalDate를 자동으로 처리하도록 합니다.
- 2) 목록에서 조회 화면으로 이동할 때 모든 검색/필터링/페이징 조건을 유지하도록 구성합니다.
- 3) 조회화면에서는 모든 조건을 유지한 채로 수정/삭제화면으로 이동하도록 구성합니다.
- 4) 삭제 시에는 다시 목록 화면으로 이동합니다.
- 5) 수정 시에는 다시 조회 화면으로 이동하지만 검색/필터링/페이징 조건은 초기화 합니다.

데이터베이스 테이블 수정

```
drop table tbl_todo;
```

```
create table tbl_todo (  
    tno int auto_increment primary key ,  
    title varchar(100) not null,  
    dueDate date not null,  
    writer varchar(50) not null,  
    finished tinyint default 0  
);
```

ModelMapper 설정과 @Configuration

스프링의 경우 XML파일을 통한 설정 외에도 @Configuration이 있는 클래스를 이용해서 설정을 지정할 수 있음

```
package org.zerock.springex.config;

import org.modelmapper.ModelMapper;
import org.modelmapper.convention.MatchingStrategies;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

@Configuration

```
public class ModelMapperConfig {
```

@Bean

```
public ModelMapper getMapper() {
    ModelMapper modelMapper = new ModelMapper();
    modelMapper.getConfiguration()
        .setFieldMatchingEnabled(true)
        .setFieldAccessLevel(org.modelmapper.config.Configuration.AccessLevel.PRIVATE)
        .setMatchingStrategy(MatchingStrategies.LOOSE);

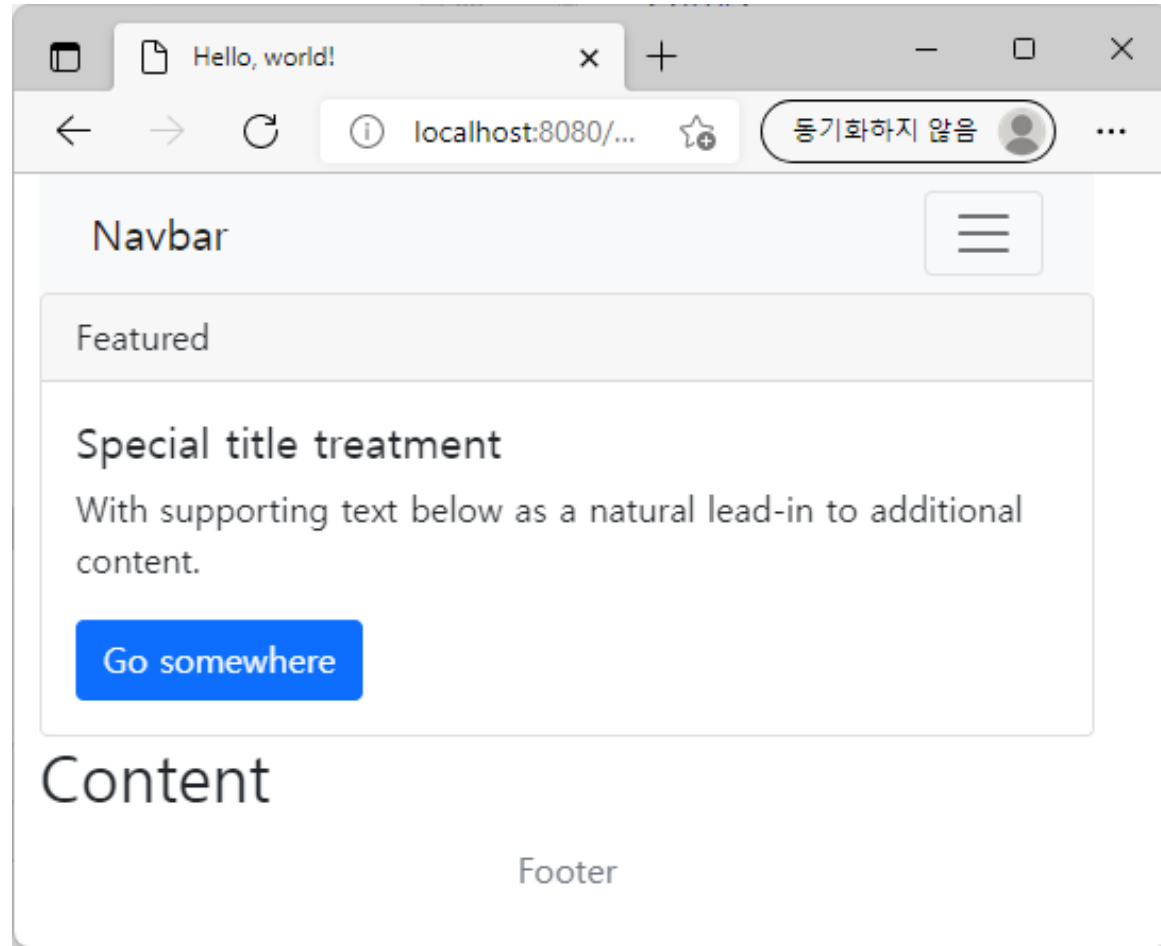
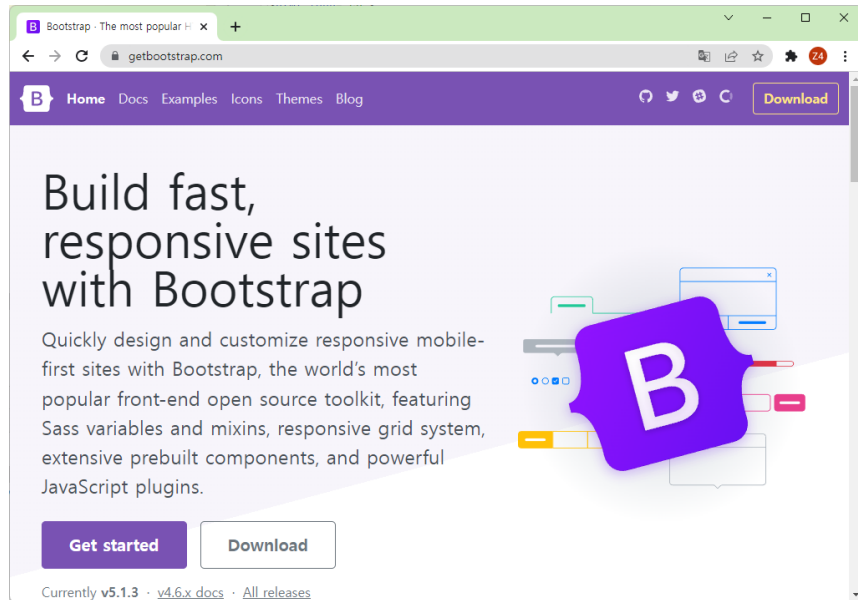
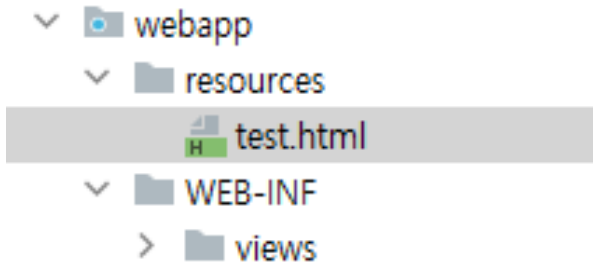
    return modelMapper;
}
}
```



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

화면디자인 - 부트스트랩

- 부트스트랩을 이용해서 간단한 화면 구성
- 컴포넌트를 이용해서 레이아웃이나 화면 디자인 가능



MyBatis와 스프링을 이용한 영속 처리

- MyBatis를 이용해서 SQL을 처리하고 테스트
- 개발의 단계
 - VO 클래스 개발
 - Mapper인터페이스 개발
 - XML을 이용해서 SQL 작성
 - 테스트 코드의 개발

java

- org.zerock.springex
 - config
 - controller
 - domain
 - TodoVO
 - dto

```
package org.zerock.springex.domain;

import lombok.*;

import java.time.LocalDate;

@Getter
@ToString
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class TodoVO {

    private Long tno;

    private String title;

    private LocalDate dueDate;

    private String writer;

    private boolean finished;
}
```

mapper

- TimeMapper
- TimeMapper2
- TodoMapper

```
package org.zerock.springex.mapper;

public interface TodoMapper {

    String getTime();
}
```

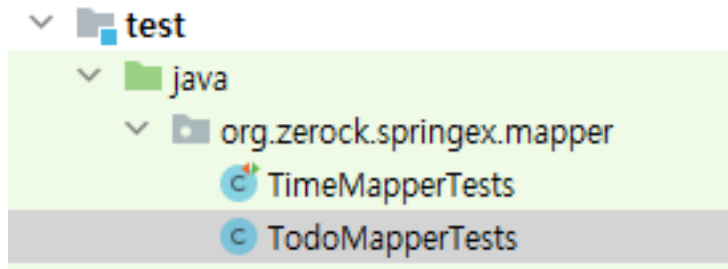
resources

- mappers
 - TimeMapper2.xml
 - TodoMapper.xml
 - log4j2.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.springex.mapper.TodoMapper">

    <select id="getTime" resultType="string">
        select now()
    </select>

</mapper>
```

```
package org.zerock.springex.mapper;
```

```
import lombok.extern.log4j.Log4j2;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
```

```
@Log4j2
```

```
@ExtendWith(SpringExtension.class)
```

```
@ContextConfiguration(locations="file:src/main/webapp/WEB-INF/root-context.xml")
```

```
public class TodoMapperTests {
```

```
    @Autowired(required = false)
```

```
    private TodoMapper todoMapper;
```

```
    @Test
```

```
    public void testGetTime() {
```

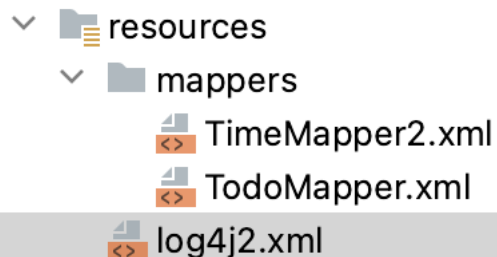
```
        log.info(todoMapper.getTime());
    }
```

```
}
```

```
09:14:34 INFO [org.springframework.test.context.support.DefaultTestContextBootstrapper] Loaded default TestExecutionLi:
09:14:34 INFO [org.springframework.test.context.support.DefaultTestContextBootstrapper] Using TestExecutionListeners:
09:14:35 INFO [com.zaxxer.hikari.HikariDataSource] HikariPool-1 - Starting...
09:14:35 INFO [com.zaxxer.hikari.pool.HikariPool] HikariPool-1 - Added connection org.mariadb.jdbc.Connection@7e7f0216
09:14:35 INFO [com.zaxxer.hikari.HikariDataSource] HikariPool-1 - Start completed.
09:14:35 INFO [org.zerock.springex.mapper.TodoMapperTests] 2022-03-09 21:14:35
BUILD SUCCESSFUL in 2s
```

로그 레벨의 조정

- MyBatis와 JDBC의 상세 로그를 위한 로그 레벨 조정



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

<configuration status="INFO">

    <Appenders>
        <!-- 콘솔 -->
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout charset="UTF-8" pattern="%d{hh:mm:ss} %5p [%c] %m%n"/>
        </Console>
    </Appenders>

    <loggers>
        <logger name="org.springframework" level="INFO" additivity="false">
            <appender-ref ref="console" />
        </logger>

        <logger name="org.zerock" level="INFO" additivity="false">
            <appender-ref ref="console" />
        </logger>

        <logger name="org.zerock.springex.mapper" level="TRACE" additivity="false">
            <appender-ref ref="console" />
        </logger>

        <root level="INFO" additivity="false">
            <AppenderRef ref="console"/>
        </root>

    </loggers>

</configuration>
```

Todo 기능 개발(insert)

- 개발 순서
 - TodoMapper -> TodoService -> TodoController -> JSP

```
package org.zerock.springex.mapper;

import org.zerock.springex.domain.TODOVO;

public interface TodoMapper {

    String getTime();

    void insert(TODOVO todoVO);
}
```

```
<?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.springex.mapper.TODOMapper">

    <select id="getTime" resultType="string">
        select now()
    </select>

    <insert id="insert">
        insert into tbl_todo (title, dueDate, writer) values ( #{title}, #{dueDate}, #{writer})
    </insert>

</mapper>
```

TodoService와 TodoServiceImpl

> mapper
▼ service
 I TodoService

```
package org.zerock.springex.service;

import org.zerock.springex.dto.TODO;

public interface TodoService {

    void register(TODO todo);
}
```

```
package org.zerock.springex.service;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Service;
import org.zerock.springex.domain.TODO;
import org.zerock.springex.dto.TODO;
import org.zerock.springex.mapper.TODO;
```

```
@Service
@Log4j2
@RequiredArgsConstructor
public class TodoServiceImpl implements TodoService{
```

```
    private final TODO todo;
```

```
    private final ModelMapper modelMapper;
```

```
@Override
public void register(TODO todo) {
```

```
    log.info(modelMapper);
```

```
    TODO todo = modelMapper.map(todo, TODO.class);
```

```
    log.info(todo);
```

```
    todoMapper.insert(todo);
```

```
    }
}
```

TodoController의 GET/POST 처리

- controller
 - exception
 - formatter
 - SampleController
 - TodoController

- WEB-INF
 - views
 - todo
 - register.jsp
 - custom404.jsp

```
package org.zerock.springex.controller;

import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@Controller
@RequestMapping("/todo")
@Log4j2
public class TodoController {

    @RequestMapping("/list")
    public void list(Model model){
        log.info("todo list.....");
    }

    @GetMapping("/register")
    public void registerGET() {
        log.info("GET todo register.....");
    }

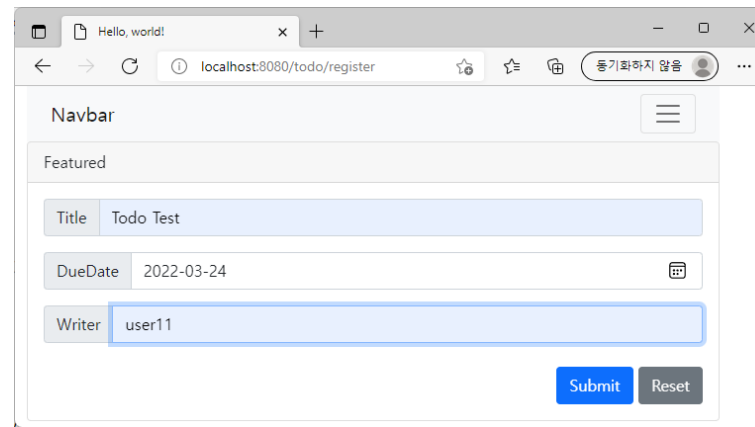
}
```

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/todo/register'. The page layout includes a 'Navbar' at the top, a 'Featured' section below it, and a main content area. The 'Featured' section contains a form with three input fields: 'Title', 'DueDate' (with a date picker icon), and 'Writer'. Below these fields are two buttons: 'Submit' (blue) and 'Reset' (grey). The main content area is labeled 'Content' and the footer is labeled 'Footer'.

```
@PostMapping("/register")
public String registerPost(TodoDTO todoDTO, RedirectAttributes redirectAttributes) {
    log.info("POST todo register.....");

    log.info(todoDTO);

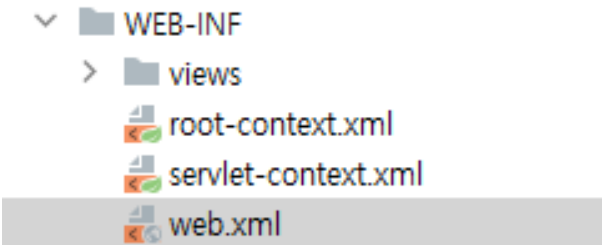
    return "redirect:/todo/list";
}
```



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/todo/register'. The page has a 'Navbar' at the top and a 'Featured' section below it. The 'Featured' section contains a form with three input fields: 'Title' (containing 'Todo Test'), 'DueDate' (containing '2022-03-24'), and 'Writer' (containing 'user11'). There are 'Submit' and 'Reset' buttons at the bottom right of the form.

’/todo/list’에 대한 처리가 아직 이루어지지 않은 상황이지만
한글 깨짐등의 문제를 먼저 처리하도록 구성

UTF-8 필터 처리



← → ↻ ⓘ localhost:8080/todo/register

Navbar ☰

Featured

Title

DueDate 📅

Writer

```
<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>
```

POST todo register.....

TodoDTO(tno=null, title=Todo Test 한글, dueDate=2022-03-31, finished=false, writer=user11)

todo list.....

@Valid를 이용한 서버사이드 검증

- hibernate-validate 라이브러리를 이용해서 서버사이드에서 검증

> test
build.gradle
gradlew

```
@ToString
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class TodoDTO {

    private Long tno;

    @NotEmpty
    private String title;

    @Future
    private LocalDate dueDate;

    private boolean finished;

    @NotEmpty
    private String writer;

}
```

```
@PostMapping("/register")
public String registerPost(@Valid TodoDTO todoDTO,
                           BindingResult bindingResult,
                           RedirectAttributes redirectAttributes) {

    log.info("POST todo register.....");

    if(bindingResult.hasErrors()) {
        log.info("has errors.....");
        redirectAttributes.addFlashAttribute("errors", bindingResult.getAllErrors());
        return "redirect:/todo/register";
    }

    log.info(todoDTO);

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

Featured

Title	Todo Test 한글
DueDate	2022-03-31
Writer	Writer

Submit Reset

```
[org.zerock.springex.controller.TODOController] has errors.....
[org.zerock.springex.controller.TODOController] GET todo register.....
```


서버 검증 메시지 처리

- BindResult의 에러 메시지를 좀 더 편리하게 JavaScript 로 처리
- 상황에 따라서 처리가 가능하다는 장점

```
<script>

const serverValidResult = {}

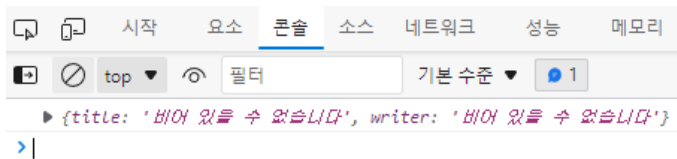
<c:forEach items="${errors}" var="error">

serverValidResult['${error.getField()}'] = '${error.defaultMessage}'

</c:forEach>

console.log(serverValidResult)

</script>
```



```
{title: '비어 있을 수 없습니다', writer: '비어 있을 수 없습니다'}
```

```
<script>

const serverValidResult = {}

console.log(serverValidResult)

</script>
```

```
<script>

const serverValidResult = {}

serverValidResult['title'] = '비어 있을 수 없습니다'

serverValidResult['writer'] = '비어 있을 수 없습니다'

console.log(serverValidResult)

</script>
```

Todo 목록 기능 개발

TodoMapper 기능 개발

```
public interface TodoMapper {  
  
    String getTime();  
  
    void insert(TodoVO todoVO);  
  
    List<TodoVO> selectAll();  
  
}
```

```
<select id="selectAll"  
resultType="org.zerock.springex.domain.TODOVO">  
    select * from tbl_todo order by tno desc  
</select>
```

TodoService 기능 개발

```
package org.zerock.springex.service;  
  
import org.zerock.springex.dto.TODODTO;  
  
import java.util.List;  
  
public interface TodoService {  
  
    void register(TODODTO todoDTO);  
  
    List<TODODTO> getAll();  
  
}
```

```
@Override  
public List<TODODTO> getAll() {  
  
    List<TODODTO> dtoList = todoMapper.selectAll().stream()  
        .map(vo -> modelMapper.map(vo, TODODTO.class))  
        .collect(Collectors.toList());  
  
    return dtoList;  
}
```

TodoController의 목록 처리

```
@RequestMapping("/list")
public void list(Model model){

    log.info("todo list.....");

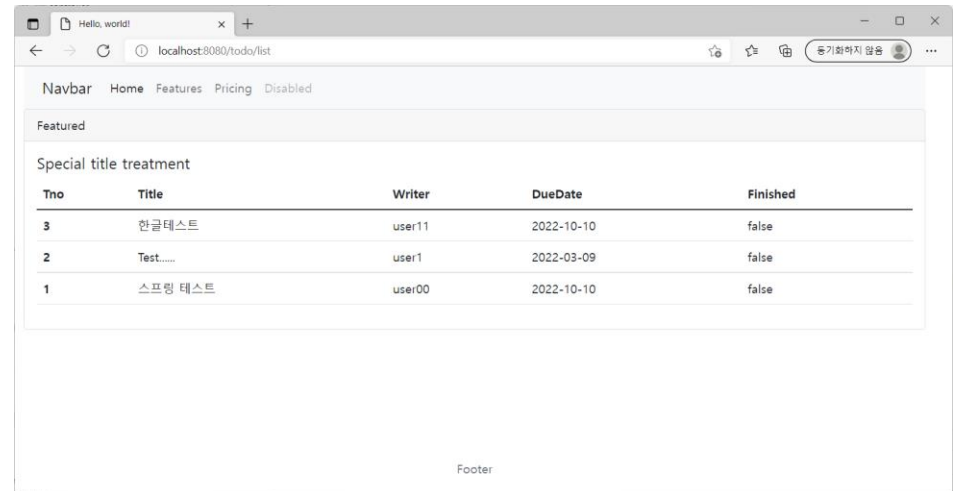
    model.addAttribute("dtoList", todoService.getAll());

}
```



```
<div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <table class="table">
        <thead>
            <tr>
                <th scope="col">Tno</th>
                <th scope="col">Title</th>
                <th scope="col">Writer</th>
                <th scope="col">DueDate</th>
                <th scope="col">Finished</th>
            </tr>
        </thead>
        <tbody>
            <c:forEach items="${dtoList}" var="dto">
                <tr>
                    <th scope="row"><c:out value="${dto.tno}"/></th>
                    <td><c:out value="${dto.title}"/></td>
                    <td><c:out value="${dto.writer}"/></td>
                    <td><c:out value="${dto.dueDate}"/></td>
                    <td><c:out value="${dto.finished}"/></td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
</div>

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



Todo 조회 기능 개발

- '/todo/read?tno=xx ' 와 같이 TodoController호출시 동작

TodoMapper 기능 개발

```
public interface TodoMapper {  
  
    String getTime();  
  
    void insert(TodoVO todoVO);  
  
    List<TodoVO> selectAll();  
  
    TodoVO selectOne(Long tno);  
  
}
```

```
<mapper namespace="org.zerock.springex.mapper.TodoMapper">  
  
    ...  
  
    <select id="selectOne"  
        resultType="org.zerock.springex.domain.TodoVO">  
        select * from tbl_todo where tno = #{tno}  
    </select>  
  
</mapper>
```

TodoService 기능 개발

```
public interface TodoService {  
  
    void register(TodoDTO todoDTO);  
  
    List<TodoDTO> getAll();  
  
    TodoDTO getOne(Long tno);  
  
}
```

```
@Override  
public TodoDTO getOne(Long tno) {  
  
    TodoVO todoVO = todoMapper.selectOne(tno);  
  
    TodoDTO todoDTO = modelMapper.map(todoVO, TodoDTO.class);  
  
    return todoDTO;  
  
}
```

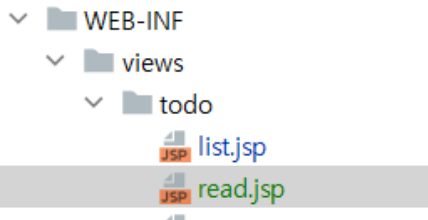
TodoController의 개발

```
@GetMapping("/read")
public void read(Long tno, Model model){

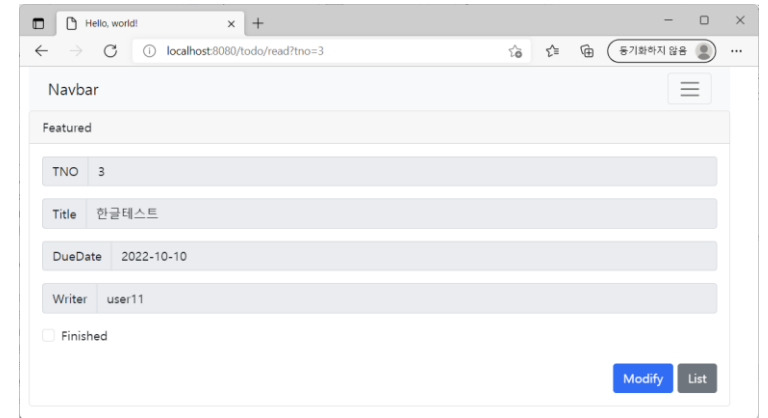
    TodoDTO todoDTO = todoService.getOne(tno);
    log.info(todoDTO);

    model.addAttribute("dto", todoDTO);

}
```



```
<div class="card-body">
  <div class="input-group mb-3">
    <span class="input-group-text">TNO</span>
    <input type="text" name="tno" class="form-control"
      value=<c:out value="${dto.tno}"></c:out> readonly>
  </div>
  <div class="input-group mb-3">
    <span class="input-group-text">Title</span>
    <input type="text" name="title" class="form-control"
      value=<c:out value="${dto.title}"></c:out> readonly>
  </div>
  <div class="input-group mb-3">
    <span class="input-group-text">DueDate</span>
    <input type="date" name="dueDate" class="form-control"
      value=<c:out value="${dto.dueDate}"></c:out> readonly>
  </div>
  <div class="input-group mb-3">
    <span class="input-group-text">Writer</span>
    <input type="text" name="writer" class="form-control"
      value=<c:out value="${dto.writer}"></c:out> readonly>
  </div>
  <div class="form-check">
    <label class="form-check-label" >
      Finished &nbsp;
    </label>
    <input class="form-check-input" type="checkbox" name="finished"
      <c:out value="${dto.finished?checked:''}" disabled >
    </div>
  <div class="my-4">
    <div class="float-end">
      <button type="button" class="btn btn-primary">Modify</button>
      <button type="button" class="btn btn-secondary">List</button>
    </div>
  </div>
```



수정/삭제를 위한 링크 처리

- 화면상에서 'Modify'버튼을 클릭해서 GET방식으로 이동

```
<div class="my-4">
  <div class="float-end">
    <button type="button" class="btn btn-primary">Modify</button>
    <button type="button" class="btn btn-secondary">List</button>
  </div>
</div>

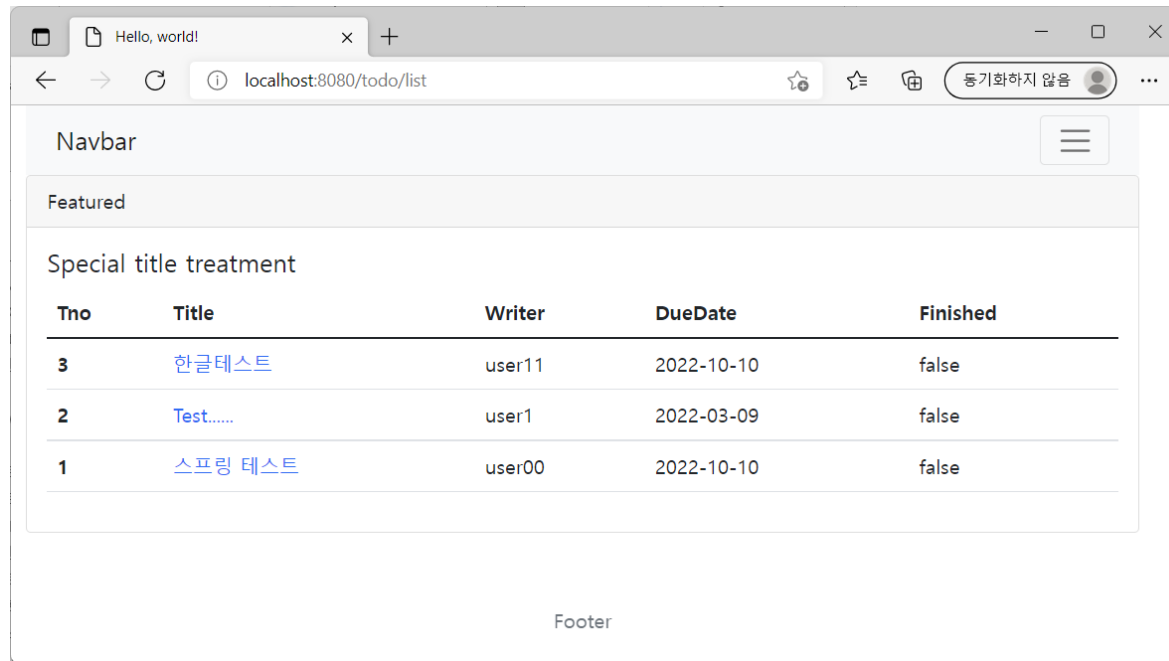
<script>
  document.querySelector(".btn-primary").addEventListener("click", function(e){
    self.location = "/todo/modify?tno="+${dto.tno}
  },false)

  document.querySelector(".btn-secondary").addEventListener("click", function(e){
    self.location = "/todo/list";
  },false)
</script>
```

list.jsp의 링크 처리

- list.jsp에서는 제목부분에 링크를 통해서 조회로 이동하도록 수정

```
<tr>
  <th scope="row"><c:out value="${dto.tno}" /></th>
  <td><a href="/todo/read?tno=${dto.tno}" class="text-decoration-none"><c:out value="${dto.title}" /></a></td>
  <td><c:out value="${dto.writer}" /></td>
  <td><c:out value="${dto.dueDate}" /></td>
  <td><c:out value="${dto.finished}" /></td>
</tr>
```



Todo의 삭제 기능 개발

- 수정 화면에서 POST방식을 통해서 삭제 처리

/todo/register(GET)

Navbar

Featured

Title Title

DueDate 년-월-일

Writer Writer

Submit Reset

Footer

/todo/list(GET)

Navbar

Featured

Special title treatment

Tno	Title	Writer	DueDate	Finished
3	등록 기능 테스트	user01	2021-12-31	false
2	Test.....	user1	2021-12-23	false
1	스프링 테스트	user00	2022-10-10	false

/todo/read?tno=xxx(GET)

Navbar

Featured

TNO 2

Title Test.....

DueDate 2021-12-23

Writer user1

☐ Finished

Modify List

/todo/remove
(POST)
tno=xxx

/todo/modify
(POST)
tno=xxx&title=x
xxx

/todo/modify?tno=xxx(GET)

Navbar

Featured

TNO 2

Title Test.....

DueDate 2021-12-23

Writer user1

☐ Finished

Remove Modify List

수정/삭제 화면

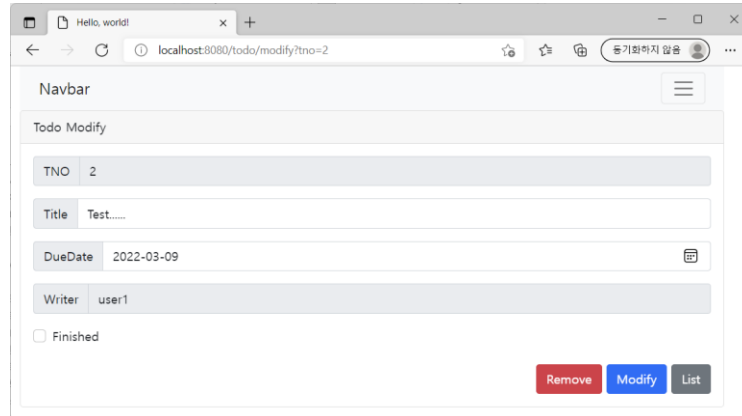
- controller
 - exception
 - formatter
 - SampleController
 - TodoController

- WEB-INF
 - views
 - todo
 - list.jsp
 - modify.jsp
 - read.jsp
 - register.jsp

```
@GetMapping({"/read", "/modify"})
public void read(Long tno, Model model){

    TodoDTO todoDTO = todoService.getOne(tno);
    log.info(todoDTO);

    model.addAttribute("dto", todoDTO );
}
```



```
<div class="card-body">
  <form action="/todo/modify" method="post">
    <div class="input-group mb-3">
      <span class="input-group-text">TNO</span>
      <input type="text" name="tno" class="form-control"
        value=<c:out value="${dto.tno}"></c:out> readonly>
    </div>
    <div class="input-group mb-3">
      <span class="input-group-text">Title</span>
      <input type="text" name="title" class="form-control"
        value=<c:out value="${dto.title}"></c:out> readonly>
    </div>
    <div class="input-group mb-3">
      <span class="input-group-text">DueDate</span>
      <input type="date" name="dueDate" class="form-control"
        value=<c:out value="${dto.dueDate}"></c:out> >
    </div>
    <div class="input-group mb-3">
      <span class="input-group-text">Writer</span>
      <input type="text" name="writer" class="form-control"
        value=<c:out value="${dto.writer}"></c:out> readonly>
    </div>
    <div class="form-check">
      <label class="form-check-label" >
        Finished &nbsp;  
      </label>
      <input class="form-check-input" type="checkbox" name="finished"
        ${dto.finished?"checked":""}>
    </div>
    <div class="my-4">
      <div class="float-end">
        <button type="button" class="btn btn-danger">Remove</button>
        <button type="button" class="btn btn-primary">Modify</button>
        <button type="button" class="btn btn-secondary">List</button>
      </div>
    </div>
  </form>
</div>
```

Remove버튼의 처리

```
</form>
</div>

<script>

    const formObj = document.querySelector("form")

    document.querySelector(".btn-danger").addEventListener("click", function(e)
    {

        e.preventDefault()
        e.stopPropagation()

        formObj.action = "/todo/remove"
        formObj.method = "post"

        formObj.submit()

    }, false);

</script>
```

```
@PostMapping("/remove")
public String remove(Long tno, RedirectAttributes redirectAttributes){

    log.info("-----remove-----");
    log.info("tno: " + tno);

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

```
INFO [org.zerock.springex.controller.TODOController] -----remove-----
INFO [org.zerock.springex.controller.TODOController] tno: 2
INFO [org.zerock.springex.controller.TODOController] todo list.....
```

삭제 – TodoMapper/TodoService/TodoController

```
public interface TodoMapper {  
    String getTime();  
    void insert(TodoVO todoVO);  
    List<TodoVO> selectAll();  
    TodoVO selectOne(Long tno);  
    void delete(Long tno);  
}
```

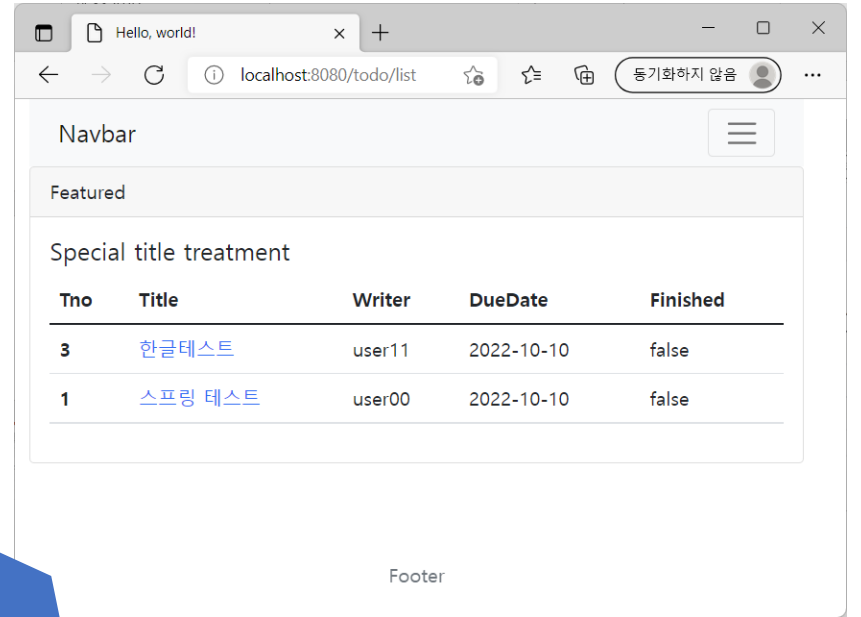
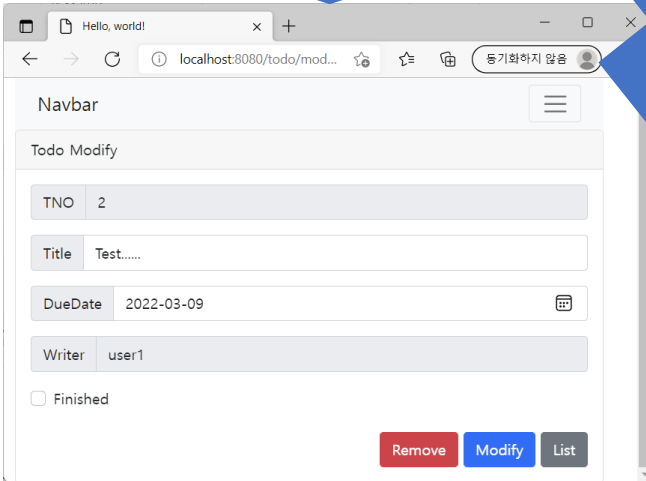
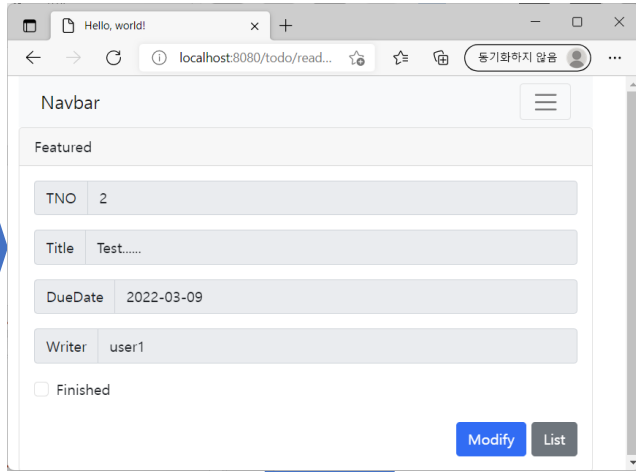
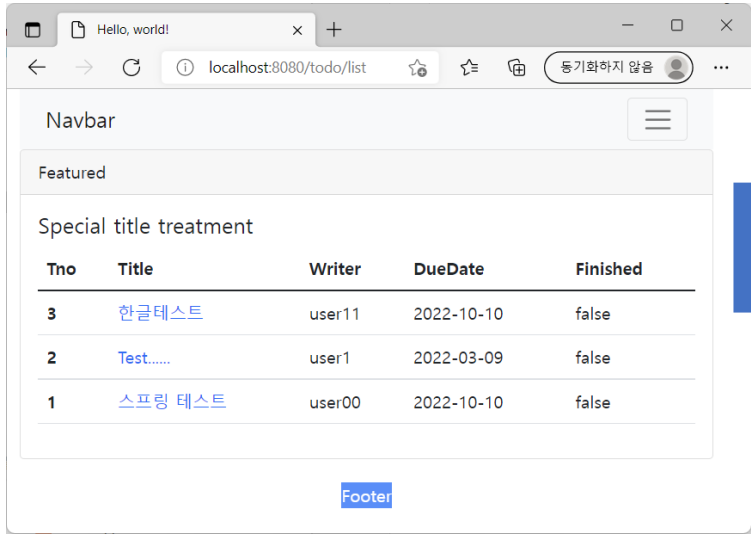
```
<mapper  
namespace="org.zerock.springex.mapper.TodoMapper">  
  
    ...  
  
    <delete id="delete">  
        delete from tbl_todo where tno = #{tno}  
    </delete>  
  
</mapper>
```

```
public interface TodoService {  
    void register(TodoDTO todoDTO);  
    List<TodoDTO> getAll();  
    TodoDTO getOne(Long tno);  
    void remove(Long tno);  
}
```

```
public class TodoServiceImpl implements TodoService{  
  
    ...  
    @Override  
    public void remove(Long tno) {  
  
        todoMapper.delete(tno);  
  
    }  
}
```

TodoController

```
@PostMapping("/remove")  
public String remove(Long tno, RedirectAttributes  
    redirectAttributes){  
  
    log.info("-----remove-----");  
    log.info("tno: " + tno);  
  
    todoService.remove(tno);  
  
    return "redirect:/todo/list";  
}
```



Todo 수정 기능 개발

TodoMapper

```
public interface TodoMapper {  
  
    String getTime();  
  
    void insert(TodoVO todoVO);  
  
    List<TodoVO> selectAll();  
  
    TodoVO selectOne(Long tno);  
  
    void delete(Long tno);  
  
    void update(TodoVO todoVO);  
}
```

```
<update id="update">  
    update tbl_todo set title = #{title} , dueDate = #{dueDate},  
    finished= #{finished} where tno = #{tno}  
</update>
```

TodoService/TodoServiceImpl

```
public interface TodoService {  
  
    void register(TodoDTO todoDTO);  
  
    List<TodoDTO> getAll();  
  
    TodoDTO getOne(Long tno);  
  
    void remove(Long tno);  
  
    void modify(TodoDTO todoDTO);  
}
```

```
@Override  
public void modify(TodoDTO todoDTO) {  
  
    TodoVO todoVO = modelMapper.map(todoDTO, TodoVO.class);  
  
    todoMapper.update(todoVO);  
}
```

checkbox를 위한 Formatter

```
package org.zerock.springex.controller.formatter;
import org.springframework.format.Formatter;

import java.text.ParseException;
import java.util.Locale;

public class CheckboxFormatter implements Formatter<Boolean> {

    @Override
    public Boolean parse(String text, Locale locale) throws ParseException {
        if(text == null ) {
            return false;
        }
        return text.equals("on");
    }

    @Override
    public String print(Boolean object, Locale locale) {
        return object.toString();
    }
}
```

```
<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="formatters">
        <set>
            <bean class="org.zerock.springex.controller.formatter.LocalDateFormatter"/>
            <bean class="org.zerock.springex.controller.formatter.CheckboxFormatter"/>
        </set>
    </property>
</bean>
```

- controller
 - exception
 - formatter
 - CheckboxFormatter
 - LocalDateFormatter
 - SampleController
 - TodoController

- WEB-INF
 - views
 - root-context.xml
 - servlet-context.xml

TodoController

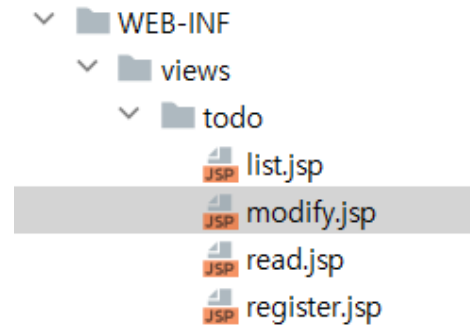
```
@PostMapping("/modify")
public String modify(@Valid TodoDTO todoDTO,
                    BindingResult bindingResult,
                    RedirectAttributes redirectAttributes){

    if(bindingResult.hasErrors()) {
        log.info("has errors.....");
        redirectAttributes.addFlashAttribute("errors", bindingResult.getAllErrors() );
        redirectAttributes.addAttribute("tno", todoDTO.getTno() );
        return "redirect:/todo/modify";
    }

    log.info(todoDTO);

    todoService.modify(todoDTO);

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



```
<script>

    const serverValidResult = {}

    <c:forEach items="${errors}" var="error">

        serverValidResult[${error.getField()}] = '${error.defaultMessage}'

    </c:forEach>

    console.log(serverValidResult)
</script>
```

```
<script>

    const formObj = document.querySelector("form")

    document.querySelector(".btn-danger").addEventListener("click",function(e) {

        ...

    },false);

    document.querySelector(".btn-primary").addEventListener("click",function(e) {

        e.preventDefault()
        e.stopPropagation()

        formObj.action = "/todo/modify"
        formObj.method = "post"

        formObj.submit()

    },false);

</script>
```

Navbar

Todo Modify

TNO 1

Title 스프링 제목 수정

DueDate 2022-10-10

Writer user00

☒ Finished

Remove Modify List

Navbar

Featured

Special title treatment

Tno	Title	Writer	DueDate	Finished
3	한글테스트	user11	2022-10-10	false
1	스프링 제목 수정	user00	2022-10-10	true

Footer

localhost:8080/todo/read?tno=1

```
<script>
    const serverValidResult = {}

    serverValidResult['dueDate'] = '미래 날짜여야 합니다'

    console.log(serverValidResult)
</script>
```


페이징처리를 위한 TodoMapper

- MariaDB/MySQL에서는 limit를 이용해서 페이징 쿼리 작성

```
select * from tbl_todo order by tno desc limit 10;
```

가져오는 데이터의 수(fetch)

```
select * from tbl_todo order by tno desc limit 10, 10;
```

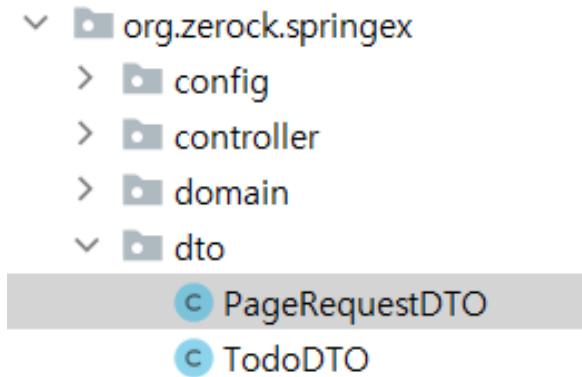
건너뛰는 데이터의 수(skip)

가져오는 데이터의 수(fetch)

limit기능은 뒤에 값만을 사용할 수 있고 식(expression)을 사용할 수 없다는 단점

페이지 처리를 위한 DTO

- 현재 페이지 번호(page)
- 한 페이지당 데이터의 수(size)



```
package org.zerock.springex.dto;

import lombok.*;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Positive;

@Builder
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PageRequestDTO {

    @Builder.Default
    @Min(value = 1)
    @Positive
    private int page = 1;

    @Builder.Default
    @Min(value = 10)
    @Max(value = 100)
    @Positive
    private int size = 10;

    public int getSkip(){

        return (page - 1) * 10;
    }
}
```

TodoMapper의 목록/count 처리

```
public interface TodoMapper {
```

```
    ...
```

```
    List<TodoVO> selectList(PageRequestDTO pageRequestDTO);  
}
```

```
<select id="selectList" resultType="org.zerock.springex.domain.TODOVO">  
    select * from tbl_todo order by tno desc limit #{skip}, #{size}  
</select>
```

```
public interface TodoMapper {
```

```
    String getTime();
```

```
    ...
```

```
    List<TodoVO> selectList(PageRequestDTO pageRequestDTO);
```

```
    int getCount(PageRequestDTO pageRequestDTO);
```

```
}
```

```
<select id="getCount" resultType="int">  
    select count(tno) from tbl_todo  
</select>
```

목록 데이터를 위한 DTO/서비스 계층

- 목록 화면에서 필요한 데이터를 하나의 DTO객체로 만들어서 사용하면 나중에 재사용이 가능
 - TodoDTO의 목록
 - 전체 데이터의 수
 - 페이지 번호의 처리를 위한 데이터들 (시작 페이지 번호/ 끝 페이지 번호...)

> domain

▼ dto

PageRequestDTO

PageResponseDTO

TodoDTO

```
package org.zerock.springex.dto;

import java.util.List;

public class PageResponseDTO<E> {

    private int page;
    private int size;
    private int total;

    //시작 페이지 번호
    private int start;
    //끝 페이지 번호
    private int end;

    //이전 페이지의 존재 여부
    private boolean prev;
    //다음 페이지의 존재 여부
    private boolean next;

    private List<E> dtoList;

}
```

PageResponseDTO의 생성자

```
@Builder(builderMethodName = "withAll")
public PageResponseDTO(PageRequestDTO pageRequestDTO,
    List<E> dtoList, int total){

    this.page = pageRequestDTO.getPage();
    this.size = pageRequestDTO.getSize();

    this.total = total;
    this.dtoList = dtoList;

}
```

페이지 번호의 계산

- page가 1인 경우: 시작 페이지(start)는 1, 마지막 페이지(end)는 10
- page가 10인 경우: 시작 페이지(start)는 1, 마지막 페이지(end)는 10
- page가 11인 경우: 시작 페이지(start)는 11, 마지막 페이지(end)는 20

시작 페이지 번호/마지막 페이지 번호 계산

```
this.end = (int)(Math.ceil(this.page / 10.0 )) * 10;
```

page를 10으로 나눈 값을 올림 처리 한 후 * 10

1 / 10 => 0.1 => 1 => 10

11 / 10 => 1.1 => 2 => 20

10 / 10 => 1.0 => 1 => 10

```
this.end = (int)(Math.ceil(this.page / 10.0 )) * 10;
```

```
this.start = this.end - 9;
```

```
int last = (int)(Math.ceil((total/(double)size)));
```

이전/다음 페이지 계산

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

```
this.next = total > this.end * this.size;
```

```
@Getter
@ToString
public class PageResponseDTO<E> {

    private int page;
    private int size;
    private int total;

    //시작 페이지 번호
    private int start;
    //끝 페이지 번호
    private int end;

    //이전 페이지의 존재 여부
    private boolean prev;
    //다음 페이지의 존재 여부
    private boolean next;

    private List<E> dtoList;

    @Builder(builderMethodName = "withAll")
    public PageResponseDTO(PageRequestDTO pageRequestDTO, List<E> dtoList,
int total){

        this.page = pageRequestDTO.getPage();
        this.size = pageRequestDTO.getSize();

        this.total = total;
        this.dtoList = dtoList;

        this.end = (int) (Math.ceil(this.page / 10.0 )) * 10;

        this.start = this.end - 9;

        int last = (int) (Math.ceil((total/(double) size)));

        this.end = end > last ? last: end;

        this.prev = this.start > 1;

        this.next = total > this.end * this.size;

    }
}
```

TodoService/TodoServiceImpl에서의 목록 처리

```
package org.zerock.springex.service;

import org.zerock.springex.dto.PageRequestDTO;
import org.zerock.springex.dto.PageResponseDTO;
import org.zerock.springex.dto.TODODTO;

public interface TodoService {

    void register(TODODTO todoDTO);

    //List<TODODTO> getAll();

    PageResponseDTO<TODODTO> getList(PageRequestDTO pageRequestDTO);

    TODODTO getOne(Long tno);

    void remove(Long tno);

    void modify(TODODTO todoDTO);
}
```

```
@Override
public PageResponseDTO<TODODTO> getList(PageRequestDTO pageRequestDTO) {

    List<TODOVO> voList = todoMapper.selectList(pageRequestDTO);
    List<TODODTO> dtoList = voList.stream()
        .map(vo -> modelMapper.map(vo, TODODTO.class))
        .collect(Collectors.toList());

    int total = todoMapper.getCount(pageRequestDTO);

    PageResponseDTO<TODODTO> pageResponseDTO = PageResponseDTO.<TODODTO>withAll()
        .dtoList(dtoList)
        .total(total)
        .pageRequestDTO(pageRequestDTO)
        .build();

    return pageResponseDTO;
}
```


TodoController와 JSP처리

- controller
 - exception
 - formatter
 - SampleController
 - TodoController

```
@GetMapping("/list")
public void list(@Valid PageRequestDTO pageRequestDTO, BindingResult bindingResult, Model model){

    log.info(pageRequestDTO);

    if(bindingResult.hasErrors()){
        pageRequestDTO = PageRequestDTO.builder().build();
    }
    model.addAttribute("responseDTO", todoService.getList(pageRequestDTO));
}
```

- WEB-INF
 - views
 - todo
 - list.jsp
 - modify.jsp

```
<c:forEach items="${responseDTO.dtoList}" var="dto">
<tr>
    <th scope="row"><c:out value="${dto.tno}"/></th>
    <td><a href="/todo/read?tno=${dto.tno}" class="text-decoration-none"><c:out value="${dto.title}"/></a></td>
    <td><c:out value="${dto.writer}"/></td>
    <td><c:out value="${dto.dueDate}"/></td>
    <td><c:out value="${dto.finished}"/></td>
</tr>
</c:forEach>
```

페이지 번호 출력과 이동

부트스트랩의 pagination 컴포넌트 활용

Disabled and active states

Pagination links are customizable for different circumstances. Use `.disabled` for links that appear un-clickable and `.active` to indicate the current page.

While the `.disabled` class uses `pointer-events: none` to try to disable the link functionality of `<a>`s, that CSS property is not yet standardized and doesn't account for keyboard navigation. As such, you should always add `tabindex="-1"` on disabled links and use custom JavaScript to fully disable their functionality.

Previous 1 2 3 Next

```
<div class="float-end">
  <ul class="pagination flex-wrap">
    <:forEach begin="${responseDTO.start}" end="${responseDTO.end}" var="num">
      <li class="page-item"><a class="page-link" href="#">${num}</a></li>
    </forEach>
  </ul>
</div>
```

http://localhost:8080/todo/list

Featured				
Special title treatment				
Tno	Title	Writer	DueDate	Finished
1527	한글테스트	user11	2022-10-10	false
1526	스프링 제목 수정	user00	2022-10-10	false
1525	한글테스트	user11	2022-10-10	false
1524	스프링 제목 수정	user00	2022-10-10	false
1523	한글테스트	user11	2022-10-10	false
1522	스프링 제목 수정	user00	2022-10-10	false
1521	한글테스트	user11	2022-10-10	false
1520	스프링 제목 수정	user00	2022-10-10	false
1519	한글테스트	user11	2022-10-10	false
1518	스프링 제목 수정	user00	2022-10-10	false

1 2 3 4 5 6 7 8 9 10

http://localhost:8080/todo/list?page=12

Featured				
Special title treatment				
Tno	Title	Writer	DueDate	Finished
1417	한글테스트	user11	2022-10-10	false
1416	스프링 제목 수정	user00	2022-10-10	false
1415	한글테스트	user11	2022-10-10	false
1414	스프링 제목 수정	user00	2022-10-10	false
1413	한글테스트	user11	2022-10-10	false
1412	스프링 제목 수정	user00	2022-10-10	false
1411	한글테스트	user11	2022-10-10	false
1410	스프링 제목 수정	user00	2022-10-10	false
1409	한글테스트	user11	2022-10-10	false
1408	스프링 제목 수정	user00	2022-10-10	false

11 12 13 14 15 16 17 18 19 20

페이지 번호의 클릭 이벤트 처리를 위한 'data-num' 속성 추가

```
<ul class="pagination flex-wrap">
  <c:if test="${responseDTO.prev}">
    <li class="page-item">
      <a class="page-link" data-num="${responseDTO.start - 1}">Previous</a>
    </li>
  </c:if>

  <c:forEach begin="${responseDTO.start}" end="${responseDTO.end}" var="num">
    <li class="page-item ${responseDTO.page == num? "active":""}">
      <a class="page-link" data-num="${num}">${num}</a></li>
    </c:forEach>

  <c:if test="${responseDTO.next}">
    <li class="page-item">
      <a class="page-link" data-num="${responseDTO.end + 1}">Next</a>
    </li>
  </c:if>
</ul>
```

```
<script>

document.querySelector(".pagination").addEventListener("click", function (e) {
  e.preventDefault()
  e.stopPropagation()

  const target = e.target

  if(target.tagName !== 'A') {
    return
  }
  const num = target.getAttribute("data-num")

  self.location = `/todo/list?page=${num}` //백틱(` `)을 이용해서 템플릿 처리
},false)

</script>
```

목록에서 조회 페이지 이동

- 조회 페이지에서 다시 목록으로 돌아올 수 있도록 현재 페이지를 같이 쿼리스트링으로 유지할 필요가 있음
- PageResponseDTO를 이용해서 링크를 생성하는 기능을 미리 구성

```
private String link;

public int getSkip(){

    return (page - 1) * 10;
}

public String getLink() {
    if(link == null){
        StringBuilder builder = new StringBuilder();

        builder.append("page=" + this.page);

        builder.append("&size=" + this.size);
        link = builder.toString();
    }
    return link;
}
```

```
<c:forEach items="${responseDTO.dtoList}" var="dto">
    <tr>
        <th scope="row"><c:out value="${dto.tno}"/></th>
        <td>
            <a href="/todo/read?tno=${dto.tno}&${pageRequestDTO.link}" class="text-decoration-
none" data-tno="${dto.tno}" >
                <c:out value="${dto.title}"/>
            </a>
        </td>
        <td><c:out value="${dto.writer}"/></td>
        <td><c:out value="${dto.dueDate}"/></td>
        <td><c:out value="${dto.finished}"/></td>
    </tr>
</c:forEach>
```

```
<tr>
    <th scope="row">1385</th>
    <td>
        <a href="/todo/read?tno=1385&page=15&size=10" class="text-decoration-none" data-tno="1385">...</a>
    </td>
    <td>user11</td>
    <td>2022-10-10</td>
    <td>false</td>
</tr>
<tr>
    <th scope="row">1384</th>
    <td>
        <a href="/todo/read?tno=1384&page=15&size=10" class="text-decoration-none" data-tno="1384">...</a>
    </td>
    <td>user00</td>
```

페이지 정보 유지를 위한 수정

- 조회 페이지 역시 추가적으로 PageRequestDTO를 이용하도록 수정
- 수정/삭제의 경우에도 페이지 정보 유지가 필요하다면 파라미터로 PageRequestDTO를 지정

검색/필터링 조건의 정의

완료여부와 기간은 AND 필터링

'제목/작성자' 는 OR 검색

Search

☒완료여부

☒제목 ☐작성자

년-월-일 년-월-일

Search Clear

- 제목(title)과 작성자(writer)는 키워드(keyword)를 이용하는 검색 처리
- 완료여부를 필터링 처리
- 특정한 기간을 지정 (from, to) 필터링 처리

검색과 필터링에 필요한 데이터는 다음과 같이 구분

- 완료 여부에 사용되는 boolean 타입 (finished)
- 제목, 작성자 검색에 사용하는 문자열(keyword)
- 특정 기간 검색을 위한 LocalDate 변수 2개 (from, to)

- > config
- > controller
- > domain
- ▼ dto

- PageRequestDTO
- PageResponseDTO
- TodoDTO

```
public class PageRequestDTO {  
  
    ...  
    private String[] types;  
  
    private String keyword;  
  
    private boolean finished;  
  
    private LocalDate from;  
  
    private LocalDate to;  
  
    ...  
}
```

types에 따른 동적 쿼리

- MyBatis의 동적 쿼리(dynamic query)기능을 이용해서 상황에 따라서 다른 코드를 만들어 낼 수 있음
 - if
 - choose(when, otherwise)
 - trim(where, set)
 - foreach

```
<select id="selectList" resultType="org.zerock.springex.domain.TODOVO">
  select * from tbl_todo
  <foreach collection="types" item="type">
    #{type}
  </foreach>
  order by tno desc limit #{skip}, #{size}
</select>
```

```
<select id="selectList" resultType="org.zerock.springex.domain.TODOVO">
  select * from tbl_todo
  <foreach collection="types" item="type">
    <if test="type == 't'.toString()">
      title like concat('%', #{keyword}, '%')
    </if>
    <if test="type == 'w'.toString()">
      writer like concat('%', #{keyword}, '%')
    </if>
  </foreach>
  order by tno desc limit #{skip}, #{size}
</select>
```

```
<foreach collection="types" item="type" open="(" close=")" separator=" OR ">
  <if test="type == 't'.toString()">
    title like concat('%', #{keyword}, '%')
  </if>
  <if test="type == 'w'.toString()">
    writer like concat('%', #{keyword}, '%')
  </if>
</foreach>
```


<where>처리

```
<select id="selectList" resultType="org.zerock.springex.domain.TODOVO">
  select * from tbl_todo
  <where>
    <if test="types != null and types.length > 0">
      <foreach collection="types" item="type" open="(" close=")" separator=" OR ">
        <if test="type == 't'.toString()">
          title like concat('%', #{keyword}, '%')
        </if>
        <if test="type == 'w'.toString()">
          writer like concat('%', #{keyword}, '%')
        </if>
      </foreach>
    </if>
  </where>
  order by tno desc limit #{skip}, #{size}
</select>
```

<trim>과 완료 여부/완료일 필터링

```
<where>
  <if test="types != null and types.length > 0">
    <foreach collection="types" item="type" open="(" close=")" "
separator=" OR ">
      <if test="type == 't'.toString()">
        title like concat('%', #{keyword}, '%')
      </if>
      <if test="type == 'w'.toString()">
        writer like concat('%', #{keyword}, '%')
      </if>
    </foreach>
  </if>

  <if test="finished">
    <trim prefix="and">
      finished = 1
    </trim>
  </if>
</where>
```

```
<where>
  <if test="types != null and types.length > 0">
    <foreach collection="types" item="type" open="(" close=")" separator=" OR ">
      <if test="type == 't'.toString()">
        title like concat('%', #{keyword}, '%')
      </if>
      <if test="type == 'w'.toString()">
        writer like concat('%', #{keyword}, '%')
      </if>
    </foreach>
  </if>

  <if test="finished">
    <trim prefix="and">
      finished = 1
    </trim>
  </if>

  <if test="from != null and to != null">
    <trim prefix="and">
      dueDate between #{from} and #{to}
    </trim>
  </if>

</where>
```

검색 조건을 위한 화면처리

```
<!--추가하는 코드-->
<div class="row content">
  <div class="col">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Search </h5>
        <form action="/todo/list" method="get">
          <input type="hidden" name="size" value="${pageRequestDTO.size}">
          <div class="mb-3">
            <input type="checkbox" name="finished" >완료여부
          </div>
          <div class="mb-3">
            <input type="checkbox" name="types" value="t">제목
            <input type="checkbox" name="types" value="w">작성자
            <input type="text" name="keyword" class="form-control" >
          </div>
          <div class="input-group mb-3 dueDateDiv">
            <input type="date" name="from" class="form-control">
            <input type="date" name="to" class="form-control">
          </div>
          <div class="input-group mb-3">
            <div class="float-end">
              <button class="btn btn-primary" type="submit">Search</button>
              <button class="btn btn-info" type="reset">Clear</button>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

The screenshot shows a web browser window with the URL `localhost:8080/todo/list?page=4&size=10`. The page has a navbar with a search section and a featured section.

Search Section:

- Search input field
- ☐ 완료여부
- ☐ 제목 ☐ 작성자
- Date range selector: 년-월-일
- Buttons: Search, Clear

Featured Section:

Special title treatment

Tno	Title	Writer	DueDate	Finished
1497	한글테스트	user11	2022-10-10	false
1496	스프링 제목 수정	user00	2022-10-10	false
1495	한글테스트	user11	2022-10-10	false
1494	스프링 제목 수정	user00	2022-10-10	false
1493	한글테스트	user11	2022-10-10	false
1492	스프링 제목 수정	user00	2022-10-10	false
1491	한글테스트	user11	2022-10-10	false
1490	스프링 제목 수정	user00	2022-10-10	false
1489	한글테스트	user11	2022-10-10	false
1488	스프링 1488	user00	2022-10-10	true

Page navigation: 1 2 3 4 5 6 7 8 9 10 Next

Footer

Search

☒완료여부

☒제목 ☒작성자

테스트

2022-03-01 2022-12-30

Search Clear

PageRequestDTO로 수집된 결과

INFO [org.zerock.springex.controller.TODOController]

PageRequestDTO(page=1, size=10, link=page=1&size=10, types=[t, w], keyword=테스트, finished=true, from=2022-03-01, to=2022-12-30)

실행되는 SQL

```
select * from tbl_todo WHERE ( title like concat('%', ?, '%') OR writer like concat('%', ?, '%') ) and finished = 1 and dueDate between ? and ? order by tno desc limit ?, ?
```

```
select count(tno) from tbl_todo WHERE ( title like concat('%', ?, '%') OR writer like concat('%', ?, '%') ) and finished = 1 and dueDate between ? and ?
```