

스프링

전체 목차

1. 스프링 프레임워크 개요 및 환경설정
2. 의존성 주입
3. MyBatis와 스프링 연동
4. 스프링 Web MVC 기초
5. 스프링 Web MVC 구현하기(CRUD)
6. Rest 방식 데이터 교환
7. AOP와 트랜잭션
8. 파일 업로드
9. Spring Web Security

1. 스프링 프레임워크 개요 및 환경설정

1. 스프링 프레임워크 개요 및 환경설정

- 프레임워크(Framework) 정의

- 사전적 의미는 '어떤 것을 구성하는 구조 또는 뼈대'
- 소프트웨어적 의미로는 '기능을 미리 클래스나 인터페이스 등으로 만들어 제공하는 반제품'

- 프레임워크(Framework) 장점

- 일정한 기준에 따라 개발이 이루어지므로 개발 생산성과 품질이 보장된 애플리케이션을 개발할 수 있음
- 개발 후 유지보수 및 기능의 확장성에서도 고품질 보장

1. 스프링 프레임워크 개요 및 환경설정

- 스프링 프레임워크

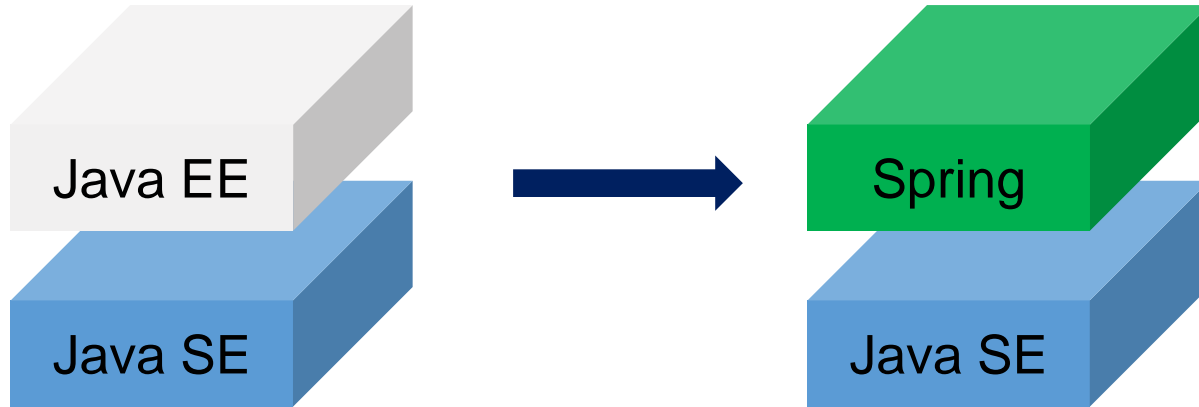
- 스프링 프레임워크(이하 스프링)는 자바 웹 애플리케이션 개발을 위한 오픈 소스 프레임워크
- EJB(Enterprise Java Beans, 엔터프라이즈 자바빈즈)보다 가벼운 경량 프레임워크 (lightweight Framework)

- 컨테이너(Container)란?

- 톰캣은 서블릿 컨테이너라고 부르는데, 그 이유는 톰캣을 실행하면 톰캣은 서블릿의 생성, 초기화, 서비스 실행, 소멸에 관한 모든 권한을 가지고 서블릿을 관리
- 스프링은 애플리케이션에서 사용되는 여러 가지 빈(클래스 객체)을 개발자가 아닌 스프링이 권한을 가지고 직접 관리

1. 스프링 프레임워크 개요 및 환경설정

- 기업형 응용 프로그램을 보조하기 위한 쉬운 프레임워크



- Java EE
분산형, 기업형 응용 프로그램 개발을 위한 API
결합력을 낮추는 DI, DB Transaction 처리, 로그 처리, ...
- Java SE
일반적인 로컬 응용 프로그램 개발을 위한 API
파일 I/O, 콘솔 I/O, 윈도우 I/O, 네트워크 I/O, Thread,...

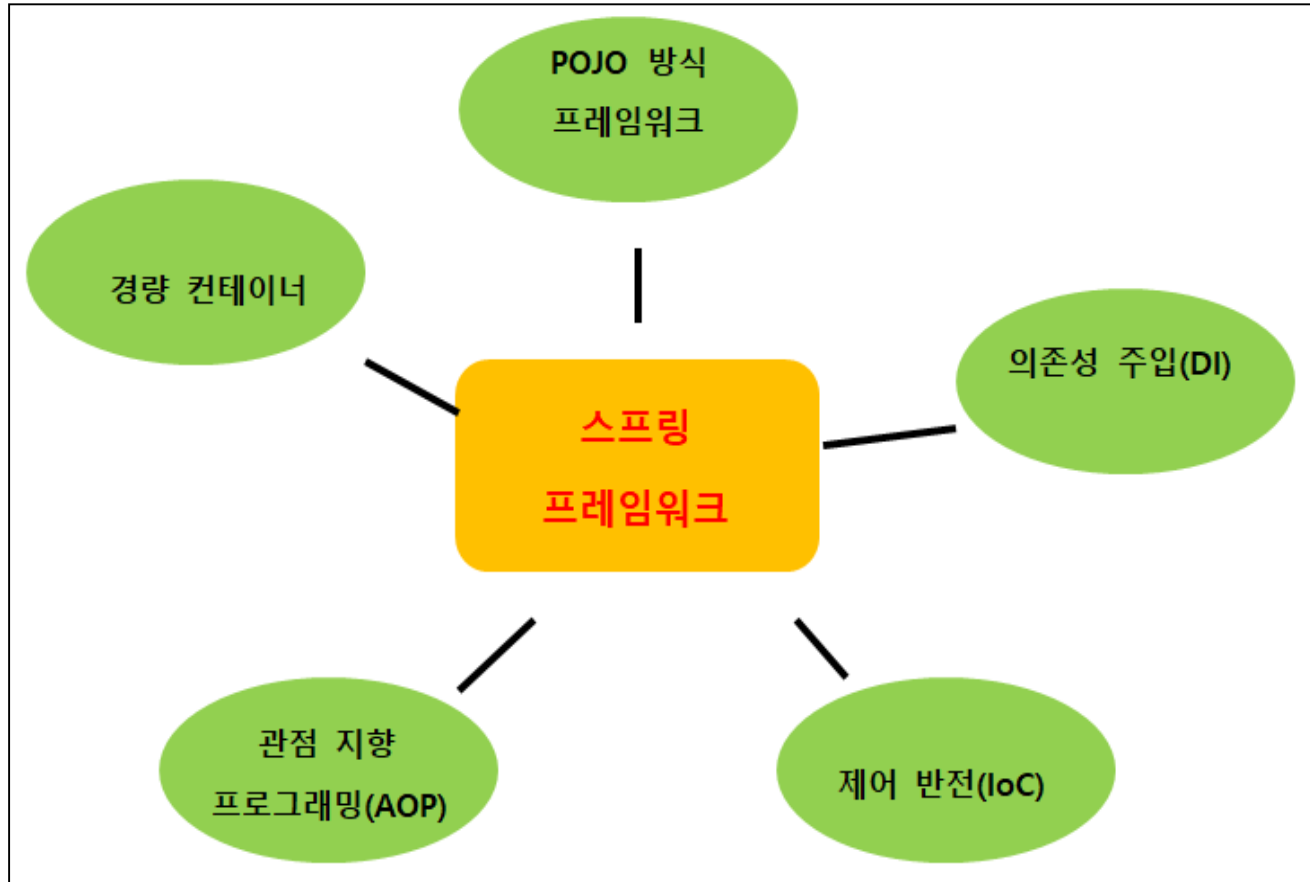
1. 스프링 프레임워크 개요 및 환경설정

- 스프링의 특징

- 가볍고 배우기도 쉬우며 경량 컨테이너의 기능을 수행
- 제어 역행(LoC, Inversion of Control) 기술을 이용해 애플리케이션 간의 느슨한 결합을 제어함
- 의존성 주입(DI, Dependency Injection) 기능을 지원함
- 관점 지향 (AOP, Aspect-Oriented Programming) 기능을 이용해 자원 관리함
- 영속성과 관련된 다양한 서비스를 지원함
- 수많은 라이브러리와 연동 기능을 지원함

1. 스프링 프레임워크 개요 및 환경설정

- 스프링 프레임워크의 특징



1. 스프링 프레임워크 개요 및 환경설정

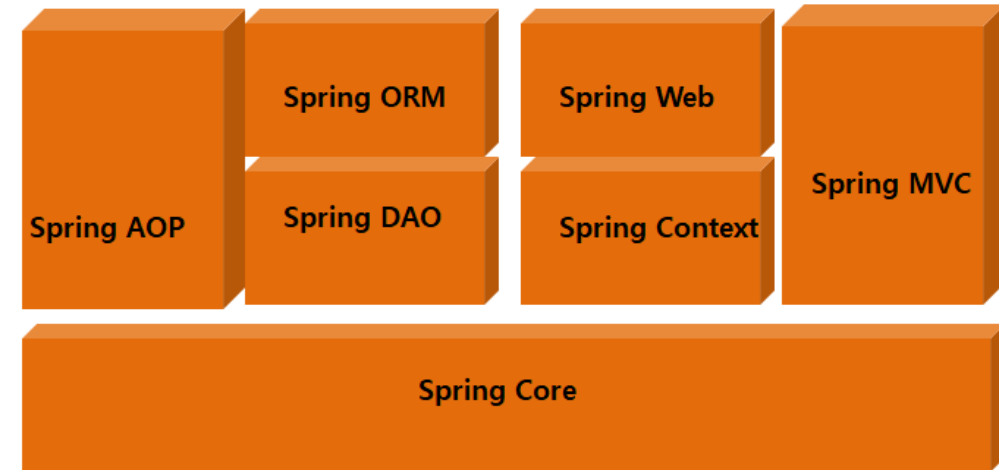
- 용어 정리

- 의존성 주입(DI): 클래스 객체를 개발자가 코드에서 생성하지 않고 프레임워크가 생성하여 사용하는 방법
- 제어 역행(IOC): 서블릿이나 빈 등을 개발자가 코드에서 생성하지 않고 프레임워크가 직접 수행하는 방법
- 관점 지향: 핵심 기능 외 부수 기능들을 분리 구현함으로써 모듈성을 증가시키는 방법
- **POJO(Plain Old Java Object)** : 객체지향적인 원리에 충실하면서, 환경과 기술에 종속되지 않고 필요에 따라 재활용될 수 있는 방식으로 설계된 오브젝트

1. 스프링 프레임워크 개요 및 환경설정

- 스프링 프레임워크의 주요 기능

기능	설명
Core	다른 기능과 설정을 분리하기 위한 IoC 기능을 제공
Context	스프링의 기본 기능으로서 애플리케이션의 각 기능을 하는 빈(Bean)에 대한 접근 방법을 제공
DAO(Data Access Object)	JDBC 기능을 좀 더 편리하게 사용할 수 있도록 합니다.
ORM(Object Relational Mapping)	하이버네이트나 마이바티스 같은 영속성 관련 프레임워크와 연동된 기능을 제공
AOP(Aspect Oriented Programming)	관점 지향 기능을 제공
Web	웹 애플리케이션 개발에 필요한 기능을 제공
WebMVC	스프링에서 MVC 구현에 관련된 기능을 제공



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(DI)
 - 현재 객체의 조력 객체를 외부에서 주입해 주는 방식의 구성
 - 제어의 역행(IoC)

스프링과 스프링 부트

- 스프링

- 자바 기반의 엔터프라이즈 애플리케이션
- 엔터프라이즈 애플리케이션 : 대규모 서비스를 뜻함
- 서버 성능, 안정성, 보안 등을 높은 수준으로 제공

- 스프링 부트

- 스프링의 복잡한 설정을 쉽게 만들어 출시한 일종의 스펀오프
 - 빠르게 스프링 프로젝트 설정 가능
 - 스타터를 사용하면 의존성 사용, 관리 용이

- 차이점

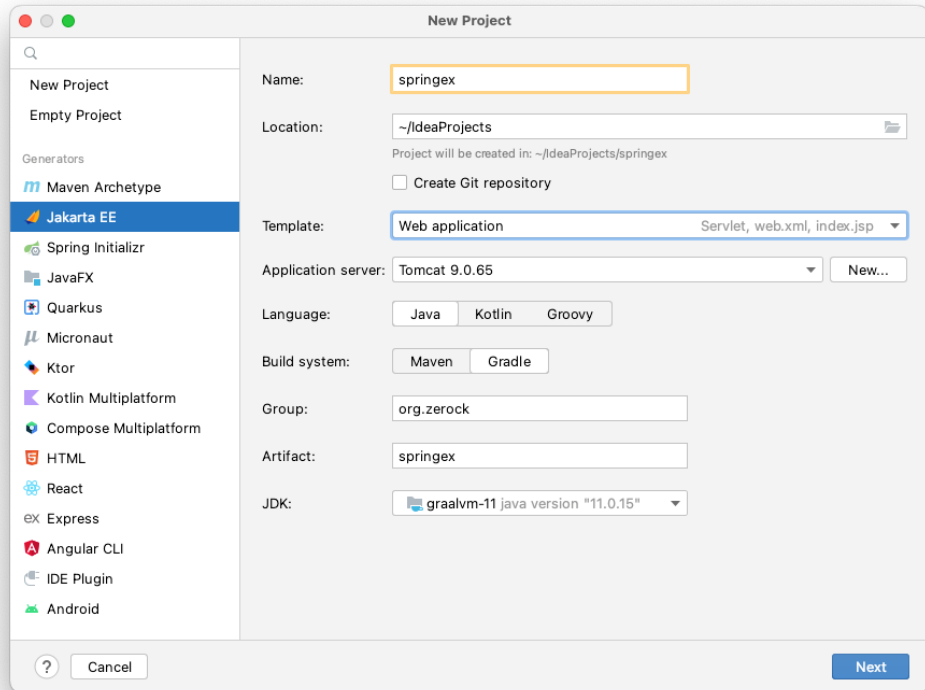
- 구성의 차이 : 스프링은 개발에 필요한 환경을 수동 구성, 스프링 부트는 자동 구성
- 내장 WAS의 유무 : 스프링 부트는 처음부터 WAS를 가지고 있음(tomcat)
 - WAS : 웹 애플리케이션을 실행하기 위한 장치

스프링과 스프링 부트

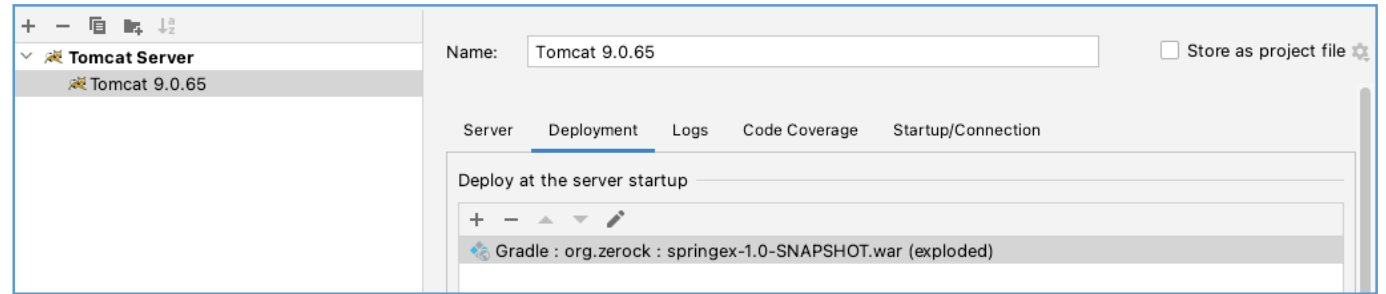
▼ 스프링과 스프링 부트 특징 비교

	스프링	스프링 부트
목적	엔터프라이즈 애플리케이션 개발을 더 쉽게 만들기	스프링의 개발을 더 빠르고 쉽게 하기
설정 파일	개발자가 수동으로 구성	자동 구성
XML	일부 파일은 XML로 직접 생성하고 관리	사용하지 않음
인메모리 데이터베이스 지원	지원하지 않음	인메모리 데이터베이스 자동 설정 지원
서버	프로젝트를 띄우는 서버(예 : 톰캣, 제티)를 별도로 수동 설정	내장형 서버를 제공해 별도의 설정이 필요 없음

프로젝트의 생성



Tomcat 조정



Application context: /

VM options:

-Dfile.encoding=UTF-8

On 'Update' action:

Update classes and resources

☒ Show dialog

On frame deactivation:

Update classes and resources

Version: Java EE 8

Dependencies:

> Specifications
v Implementations

- ☐ Apache CXF REST Server (JAX-RS) (3.4.5)
- ☐ Apache CXF XML Web Service (JAX-WS) (3.4.5)
- ☐ Apache ActiveMQ Client (5.16.3)

Apache CXF REST Server (JAX-RS)

Fully featured Web services framework that supports SOAP, JAX-RS and JAX-WS.

Spring 프레임워크 라이브러리 추가

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

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'

}

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>5.3.31</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.3.31</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>5.3.31</version>
  <scope>test</scope>
</dependency>
```



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

Lombok 추가

```
compileOnly 'org.projectlombok:lombok:1.18.30'  
annotationProcessor 'org.projectlombok:lombok:1.18.30'  
testCompileOnly 'org.projectlombok:lombok:1.18.30'  
testAnnotationProcessor 'org.projectlombok:lombok:1.18.30'
```

```
<dependency>  
  <groupId>org.projectlombok</groupId>  
  <artifactId>lombok</artifactId>  
  <version>1.18.30</version>  
  <scope>provided</scope>  
</dependency>
```


log4j 추가

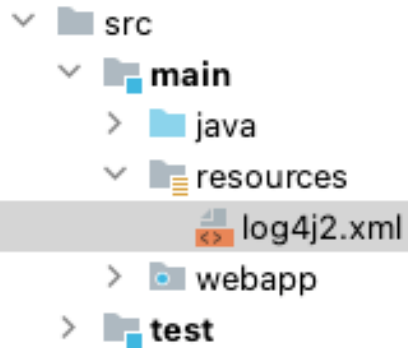
```
implementation group: 'org.apache.logging.log4j', name: 'log4j-core', version: '2.16.0'  
implementation group: 'org.apache.logging.log4j', name: 'log4j-api', version: '2.16.0'  
implementation group: 'org.apache.logging.log4j', name: 'log4j-slf4j-impl', version: '2.16.0'
```

```
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-  
core -->  
<dependency>  
  <groupId>org.apache.logging.log4j</groupId>  
  <artifactId>log4j-core</artifactId>  
  <version>2.22.0</version>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-  
api -->  
<dependency>  
  <groupId>org.apache.logging.log4j</groupId>  
  <artifactId>log4j-api</artifactId>  
  <version>2.22.0</version>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-impl -->  
<dependency>  
  <groupId>org.apache.logging.log4j</groupId>  
  <artifactId>log4j-slf4j-impl</artifactId>  
  <version>2.22.0</version>  
  <scope>test</scope>  
</dependency>
```

log4j추가



```
<?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>
</configuration>
```

jstl 추가

implementation group: 'jakarta.servlet.jsp.jstl', name: 'jakarta.servlet.jsp.jstl-api', version: '3.0.0'

implementation group: 'org.glassfish.web', name: 'jakarta.servlet.jsp.jstl', version: '3.0.1'

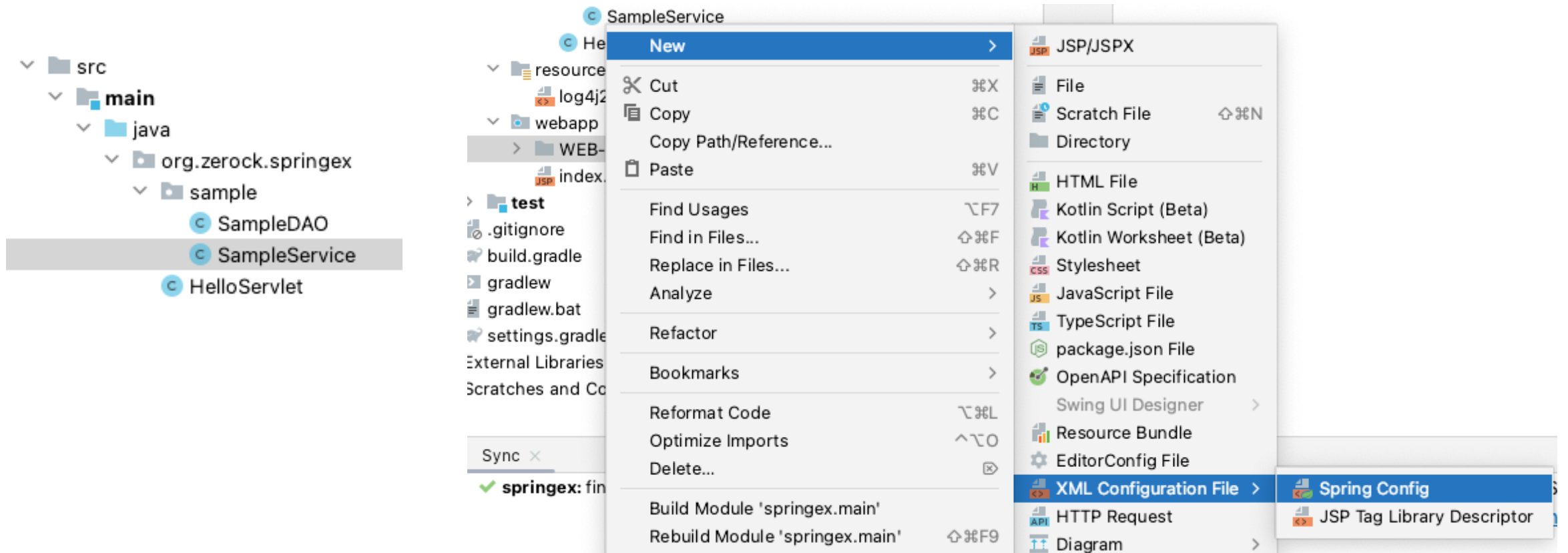
```
<!-- https://mvnrepository.com/artifact/jakarta.servlet.jsp.jstl/jakarta.servlet.jsp.jstl-api -->
<dependency>
  <groupId>jakarta.servlet.jsp.jstl</groupId>
  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
  <version>3.0.0</version>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.jsp.jstl -->
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>3.0.1</version>
</dependency>
```

2. 의존성 주입(DI)

2. 의존성 주입(DI)-의존성 주입 실습

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



2. 의존성 주입(DI)- 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">

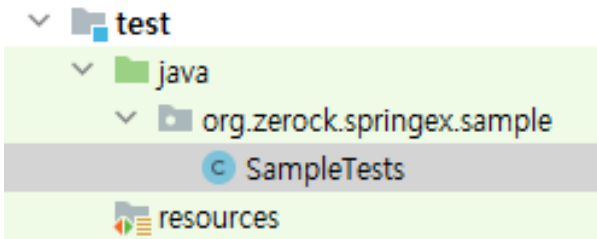
    <bean class="org.zerock.springex.sample.SampleDAO"> </bean>

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

</beans>
```

2. 의존성 주입(DI)- 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 TestExecutionList
[org.springframework.test.context.support.DefaultTestContextBootstrapper] Using TestExecutionListeners: [o
[org.zerock.springex.sample.SampleTests] org.zerock.springex.sample.SampleService@4b54af3d
```

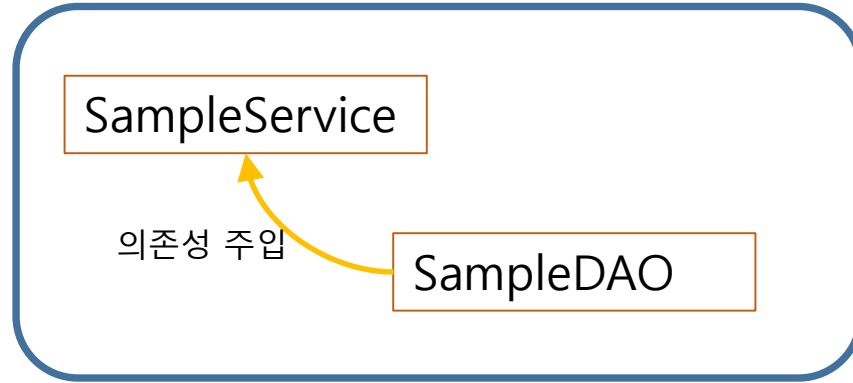
2. 의존성 주입(DI)- ApplicationContext와 빈(Beans)

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

ApplicationContext



ApplicationContext



2. 의존성 주입(DI)- @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

2. 의존성 주입(DI)- 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)
```

2. 의존성 주입(DI)- <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;
}
```

2. 의존성 주입(DI)- 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;

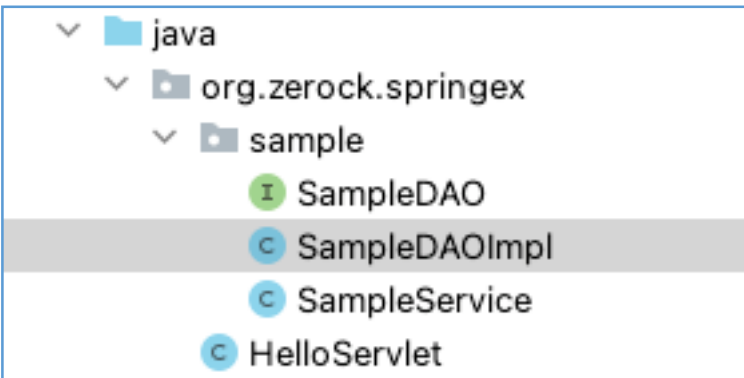
import org.springframework.stereotype.Repository;

@Repository
public class SampleDAO {

}
```

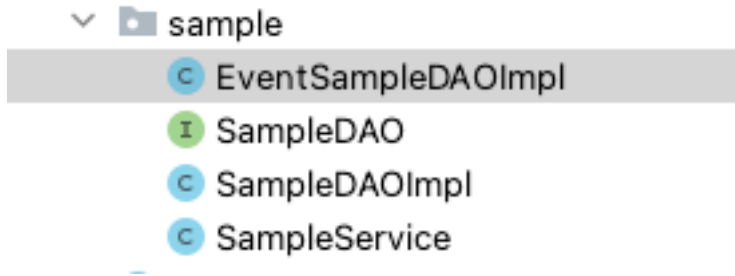
인터페이스를 이용한 느슨한 결합

```
package org.zerock.springex.sample;  
  
public interface SampleDAO {  
}
```



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

다른 SampleDAO 객체로 변경해 보기

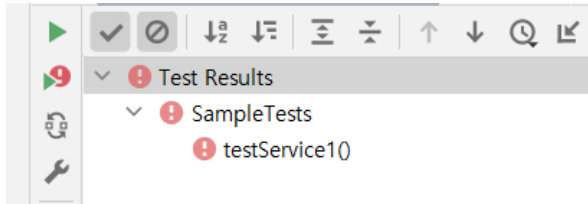


```
package org.zerock.springex.sample;

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 Create breakpoint : 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

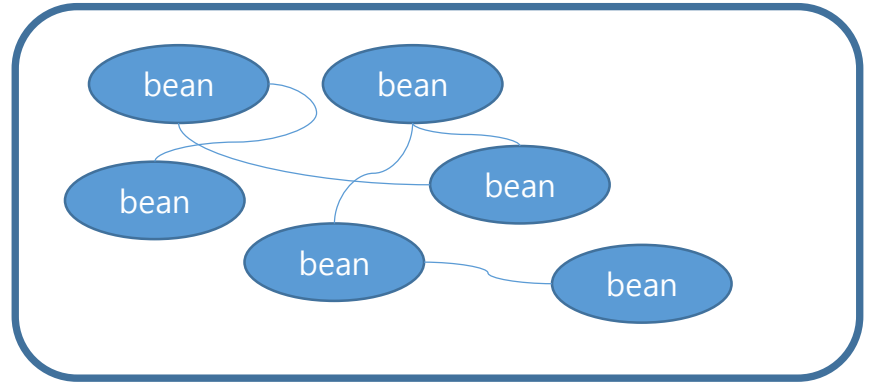
등을 이용해서 동일한 타입의 객체가 여러개 일 경우 특정 클래스를 지정 가능

웹 프로젝트를 위한 스프링 준비

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

```
> test
build.gradle
gradlew
```

ApplicationContext

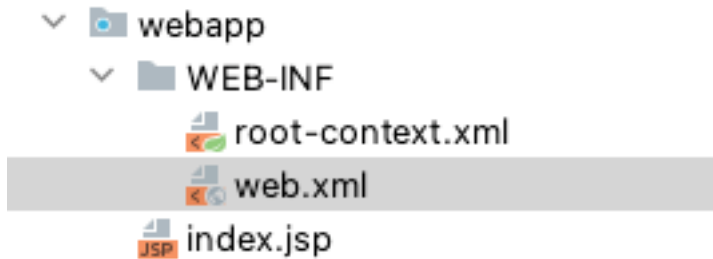


```
implementation group: 'org.springframework', name: 'spring-core', version: '5.3.16'
implementation group: 'org.springframework', name: 'spring-context', version: '5.3.16'
implementation group: 'org.springframework', name: 'spring-test', version: '5.3.16'
implementation group: 'org.springframework',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


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

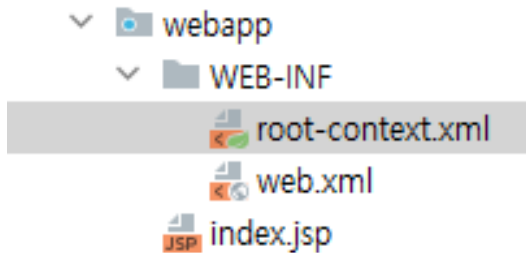
```
dependencies {  
    compileOnly('javax.servlet:javax.servlet-api:4.0.1')
```

...생략...

```
implementation group: 'com.mysql', name: 'mysql-connector-j', version: '8.0.33'
```

```
implementation group: 'com.zaxxer', name: 'HikariCP', version: '5.1.0'
```

```
}
```



```
<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
```

3. MyBatis와 스프링 연동

3. 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

3. MyBatis와 스프링의 연동

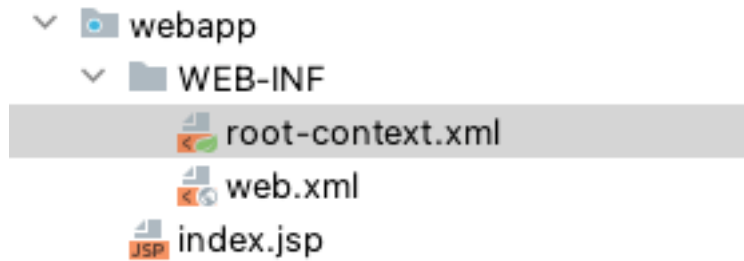
- MyBatis는 단독으로 실행이 가능한 프레임워크지만 mybatis-spring 라이브러리를 이용하면 쉽게 통합해서 사용 가능
- 과거에는 주로 별도의 DAO(Data Access Object)를 구성하고 여기서 MyBatis의 SqlSession을 이용하는 방식
- 최근에는 MyBatis는 인터페이스를 이용하고 실제 코드는 자동으로 생성되는 방식 – Mapper인터페이스와 XML
- 필요한 라이브러리
 - 스프링 관련: spring-jdbc, spring-tx
 - MyBatis 관련: mybatis, mybatis-spring

```
implementation group: 'org.springframework', name: 'spring-jdbc', version: '5.3.31'  
implementation group: 'org.springframework', name: 'spring-tx', version: '5.3.31'
```

```
implementation group: 'org.mybatis', name: 'mybatis', version: '3.5.14'  
implementation group: 'org.mybatis', name: 'mybatis-spring', version: '3.0.3'
```

3. MyBatis와 스프링 연동-MyBatis의 SqlSessionFactory설정

- MyBatis에서 실제 SQL의 처리는 SqlSessionFactory에서 생성하는 SqlSession을 통해서 수행됨

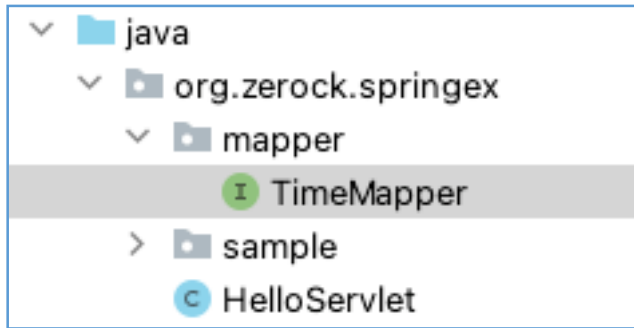


```
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"
    destroy-method="close">
    <constructor-arg ref="hikariConfig" />
</bean>

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

3. MyBatis와 스프링 연동 -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;

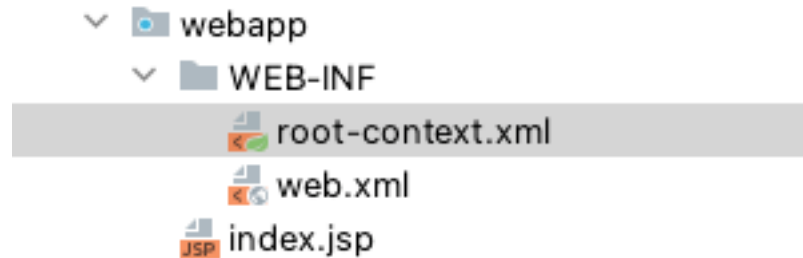
public interface TimeMapper {

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

}
```


3. MyBatis와 스프링 연동 -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>
```

3. MyBatis와 스프링 연동 - 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

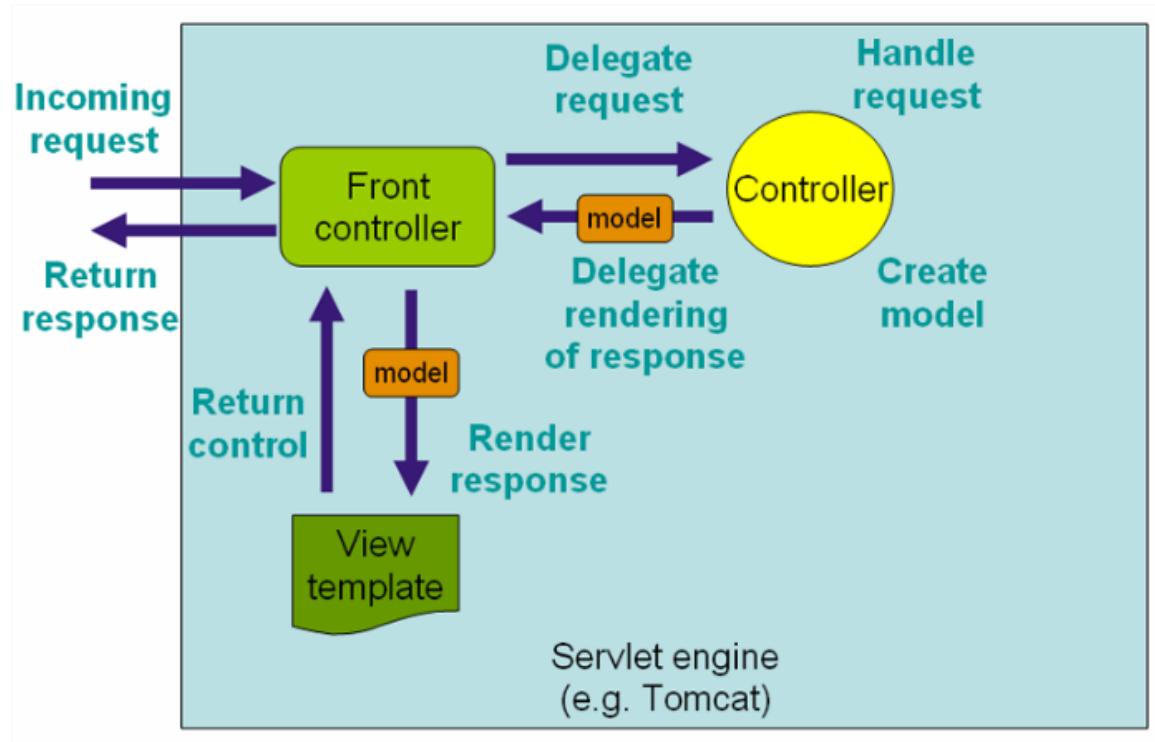
root-context.xml
web.xml

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

4. 스프링 Web MVC 기초

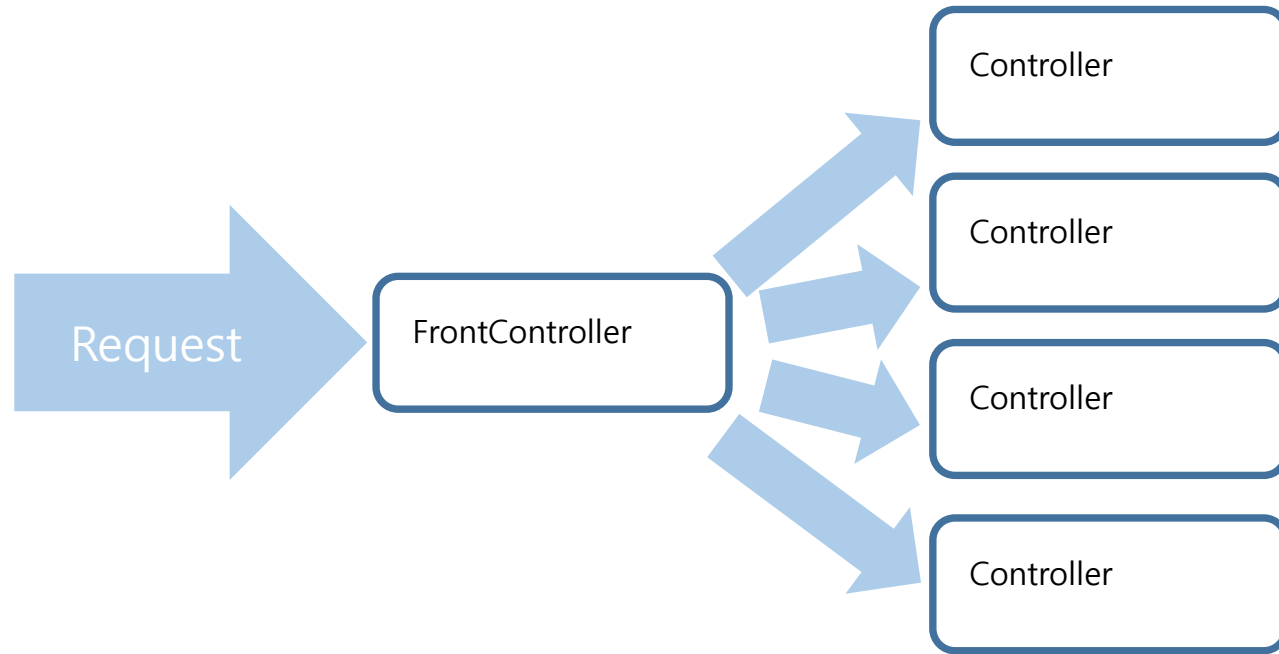
스프링 Web MVC의 특징

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



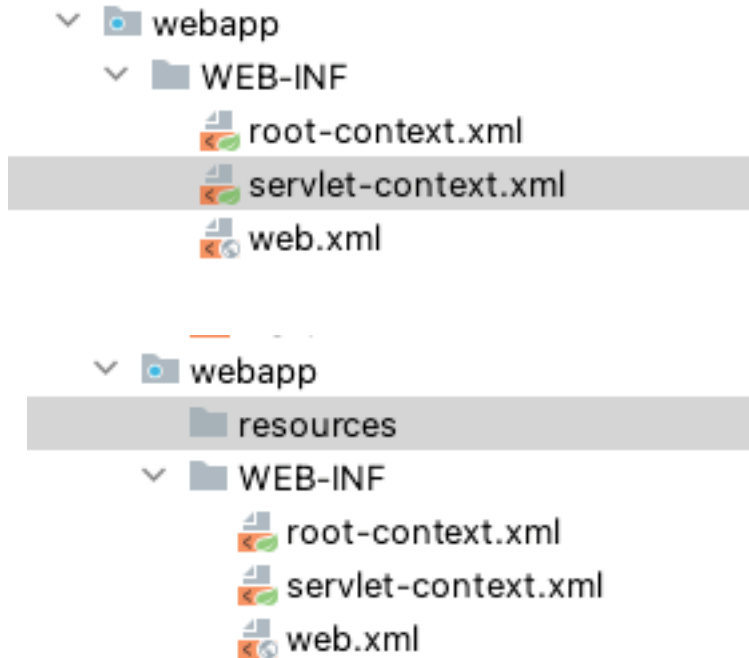
DispatcherServlet과 Front Controller

- Front Controller패턴은 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



```
<?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>

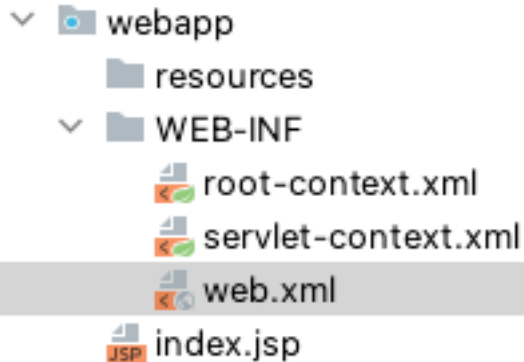
    <mvc:resources mapping="/resources/**" location="/resources/"> </mvc:reso
urces>

    <bean class="org.springframework.web.servlet.view.InternalResourceViewReso
lver">
        <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

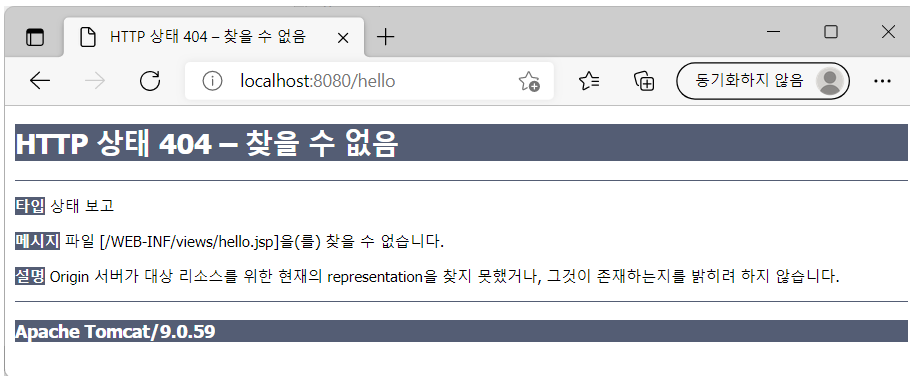
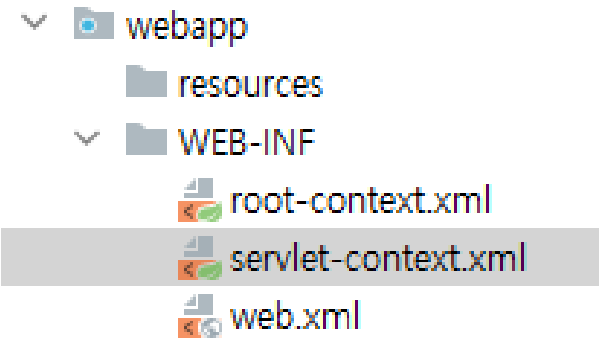
- 상속이나 인터페이스를 구현하는 방식을 사용하지 않고 어노테이션만으로 처리가 가능
- 오버라이드 없이 필요한 메서드들을 정의
- 메서드의 파라미터를 기본자료형이나 객체자료형을 마음대로 지정
- 메서드의 리턴타입도 void, String, 객체 등 다양한 타입을 사용할 수 있음

main
└─ java
 └─ org.zerock.springex
 └─ controller
 > mapper

java
└─ org.zerock.springex
 └─ controller
 └─ SampleController

```
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>
```

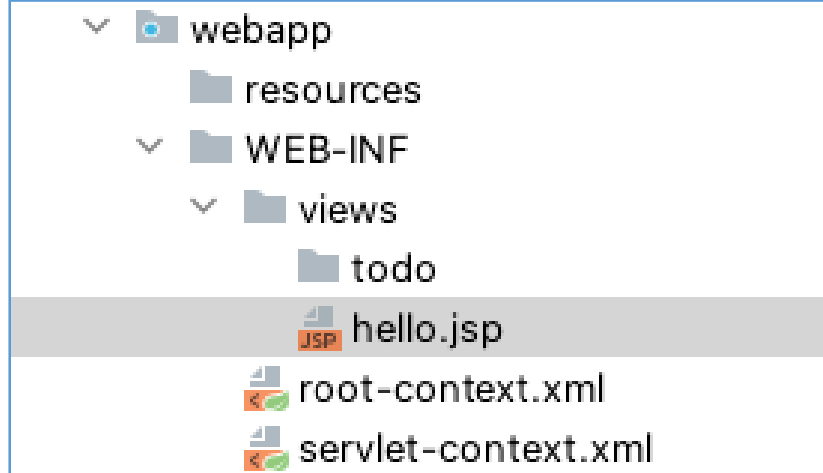
```
<mvc:resources mapping="/resources/**" location="/resources/"></mvc:resources>
```

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

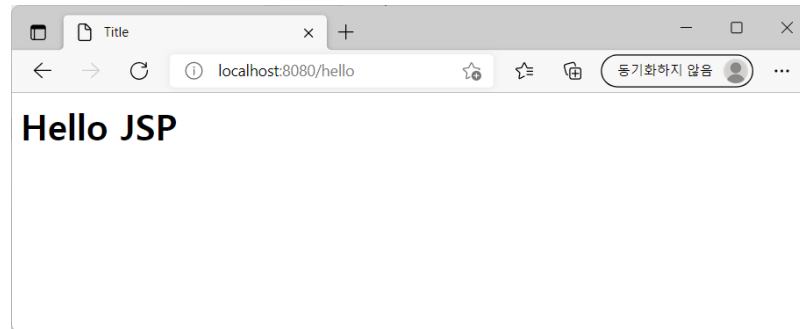
```
<context:component-scan base-package="org.zerock.springex.controller"/>
```

```
</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

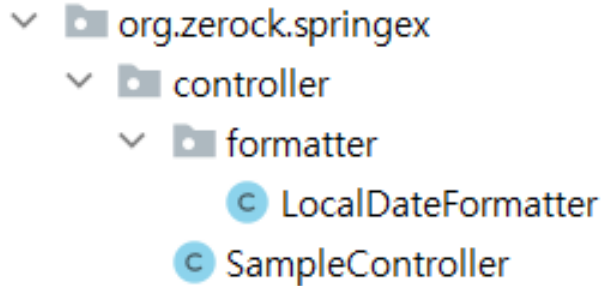
```
@GetMapping("/ex4")
public void ex4(Model model){

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

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

Formatter를 이용한 커스텀 처리

- 날짜나 형 변환을 커스터마이징 해야 하는 경우 주로 사용



```
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 numberFormat
Exception){

        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();
}
```

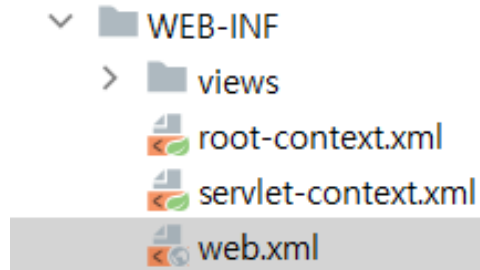
404에러

```
@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>
```