Spring DI

다룰 내용

- ✓ Spring IoC ?
- ✓ Container
- ✓ Dependency 관리
- ✓ Annotation 활용

IoC

✓ IoC (Inversion of Control) : 제어 역행

- 기존 : 인스턴스의 생성방법에 대한 부분을 개발자가 소스상에서 직접 처리
- IoC : 인스턴스의 생명주기 관리를 개발자가 아닌 컨테이너가 처리함

✓ DI (Dependency Inject) : 의존 주입

- Spring에서 Ioc를 제공하는 형태 중 하나(DL, DI)
- 종류 :
  - : Setter Injection
  - : Contructor Injection

* 의존 : 객체간의 의존관계를 의미

기존 코드 작성 방식

- ✓ 필요한 위치에서 객체 생성
- ✓ 인터페이스를 활용한 객체 생성
- ✓ 별도의 조립기 클래스를 활용하여 객체 생성

- ✓ 개선방법
  - Spring 에서 제공하는 DI 활용
  - 개발자가 코드에서 직접 객체 생성하지 않는 방식
  - XML 환경 설정파일 또는 어노테이션을 이용하여 객체를 주입

Container

✓ Spring Container

- Spring 프레임워크에서 Container 기능을 제공해 주는 클래스를 의미
- Container : Bean 클래스를 관리(생성 , 삭제등) 하는 주체
- Bean : Spring 에서 관리되는 클래스 객체를 나타냄
- Container 초기화 방법 : 설정 정보 Xml 파일을 읽고 Container 에로딩

✓ Container 종류

1. BeanFactory
   - XmlBeanFactory
2. ApplicationContext
   - ClassPathXmlApplicationContext
   - FileSystemXmlApplicationContext
   - XmlWebApplicationContext

Container

클래스명	설명
XmlBeanFactory	Xml 설정파일을 로딩하는 단순한 BeanFactory
ClassPathXmlApplicationContext	클래스패스 설정 경로로부터 Xml 설정파일 로딩
FileSystemXmlApplicationContext	지정된 파일 시스템으로부터 Xml 설정파일 로딩
XmlWebApplicationContext	웹어플리케이션 컨텍스트로부터 Xml 설정파일 로딩

Container - XML

- ✓ Spring XML 파일
  - Spring 은 XML 설정정보를 참조하여 여러가지 Container Service를 제공
  - 유지보수 작업 시 XML 파일을 조정

Container - Bean

✓ Bean 클래스

- Spring 프레임웤에 의해 LifeCycle이 관리되는 클래스
- 일반 POJO 기반의 클래스
- XML에 <bean /> 태그를 이용하여 등록
- <bean> 태그의 속성
  1. id : 여러 개의 Bean 클래스를 식별하기 위한 이름 설정
  2. name : id 속성과 동일한 의미
  3. class : 사용하려는 Bean 클래스의 패키지명을 포함한 클래스명

❖ id 와 name 속성의 차이

설정될 수 있는 값에 차이가 있음. id 속성은 자바 명명 규칙을 따름. 즉, 숫자가 우선 할 수 없고 "/" 와 같은 특수 기호가 사용 될 수 없음

Container – BeanFactory

- Bean의 LifeCycle(생성과 소멸)을 관리
- BeanFactory 객체는 XmlBeanFactory 클래스를 이용하여 초기화
- Resource 클래스를 이용하여 Xml 파일을 로딩한 후 컨테이너 객체를 생성

- 대표적인 Resource 클래스

클래스명	설명
ClassPathResource	**Xml 설정파일을 로딩하는 단순한 BeanFactory**
FileSystemResource	**지정된 파일 시스템으로부터 Xml 설정파일 로딩**

Container – ApplicationContext

- BeanFactory 기능 이외에 추가적인 기능 제공

  : 국제화를 지원, 자원을 로딩하는 범용적 방법 제공, 이벤트 처리

- BeanFactory 와의 차이점

  : 빈을 컨테이너에 로딩하는 시점이 컨텍스트가 시작되기 전에 모든 빈을 미리 로딩하여 빈이 필요할 때 바로 사용할 수 있도록 함

DI

- ✓ Dependency Injection
  - 각 Bean 간의 의존관계 설정을 xml 파일에 등록
  - 프로그램 코드에서는 직접 빈을 획득하기 위한 코드를 사용할 필요가 없음
  - Container 가 자체적으로 필요한 객체를 넘겨줘서 사용하는 방식
  - 사용방식
    1. Constructor Injection
    2. Setter Injection

Constructor Injection

- ✓ Constructor Injection : 생성자를 활용한 값 설정
  1. 문자열을 매개변수로 받는 생성자
  2. 객체를 매개변수로 받는 생성자
  3. 여러개의 매개변수를 받는 생성자
  4. 매개변수 타입 지정 생성자

Setter Injection

- ✓ Setter Injection : set 메소드를 활용한 값 설정
  1. 문자열을 매개변수로 받는 set 메서드
  2. 객체를 매개변수로 받는 set 메서드

Dependency 응용

✓ 집합객체 설정

XML	타입
<list>	java.util.List, 배열
<set>	java.util.Set
<map>	java.util.Map
<props>	Java.util.Properties

Annotation

DI 자동 주입

- ✓ XML 파일이 너무 커지는 것을 방지
- ✓ 자동 주입 기능 사용시 스프링이 알아서 의존 객체를 찾아서 주입
- ✓ 자동 주입 기능 사용 방법
  - XML파일에 <context:annotation-config /> 설정을 추가
  - Java파일에 의존주입대상에 @Autowired 또는 @Resource 설정

XML 설정

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       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
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config />

</beans>
```

@Autowired

- ✓ Java 설정
  - 변수 설정
  - 생성자 설정
  - set 메서드 설정

객체 찾는 순서

1. 타입이 같은 빈을 검색하여 한 개면 그 빈 객체를 사용
2. 두개이상이면 @Qualifier가 명시되어 되어있는 빈객체를 찾는다. 명시된 값과 같은 빈 객체를 사용
3. 두개이상이고 @Qualifier가 없을 경우 이름이 같은 빈객체를 찾아서 사용
4. 위의 경우에 해당하는 객체가 없을 경우 예외가 발생함

@Resource

- ✓ Java 설정
  - 변수 설정
  - 생성자 설정 : 제공되지 않음
  - set 메서드 설정

component-scan

✓ xml 파일 설정을 통해서 자동으로 빈으로 사용 될 객체를 등록한다.

- `<context:component-scan base-package="kr.co.mlec" />`
- 지정된 패키지 하위의 모든 패키지를 스캔하여 빈으로 등록
- 빈으로 등록되려면 자바 클래스에서 어노테이션을 사용해야 한다.
- @Component, @Controller, @Service, @Repository

코드작성

Xml 파일

```
<context:component-scan base-package="xxx" />
```

설정값이 없는 경우 클래스이름의 첫자를 소문자로 적용한 빈으로 등록

```
@Component
public class ColaDrink implements Idrink

@Component("ham")
public class HamSandwiches implements ISandwiches

@Resource(name="colaDrink")
private IDrink drink;

@Resource(name="ham")
public void setSandwiches(ISandwiches sandwiches)
```

Spring AOP

다룰 내용

- ✓ Spring AOP Introduction
- ✓ AspectJ Pointcut
- ✓ Schema Based AOP
- ✓ Annotation Based AOP

기존 OOP 방식 한계

- ✓ 중복되는 코드 발생 : 복사 & 붙이기
- ✓ 지저분한 코드 : 핵심 기능 이외의 공통기능의 코드, 가독성이 안 좋아짐
- ✓ 생산성의 저하
- ✓ 재활용의 저하
- ✓ 변화의 어려움

AOP의 목적

- ✓ 관심의 분리 : 공통 관심사항과 핵심 관심사항
- ✓ 공통관심사항 : 전체 시스템에서 사용되는 기능

  예> 로그처리, 트랜잭션, 인증, 보안, 예외처리 등등…
- ✓ 핵심관심사항 : 일반 업무 프로세스

  예> 게시글 등록, 수정, 삭제, 조회 등등….
- ✓ 핵심관심 사항은 기존의 OOP기술로 쉽게 분리가 가능하나 공통관심은 기존의 OOP 기술로 분리가 쉽지 않음
- ✓ 프로그램 전반에 사용되는 공통 모듈 적용의 효율성을 높여줌
- ✓ 각 소스에 들어있던 공통 모듈을 설정을 통하여 적용함
- ✓ 공통 기능의 변경사항이 있을 경우 사용하는 쪽은 변경할 필요가 없음

AOP 기술

- ✓ 핵심 모듈의 코드를 직접 건들이지 않고 필요한 기능이 동작하도록 하는 것
- ✓ 핵심 모듈의 코드 사이에 공통 모듈이 동작하도록 처리
- ✓ 핵심 기능에 공통 기능을 삽입하기 위한 방법 세가지
    1. 컴파일 시점에 코드에 공통 기능추가
    2. 클래스 로딩 시점에 바이트 코드에 공통 기능 추가
    3. 실행 시에 프록시 객체를 생성해서 공통 기능 추가
- ✓ 스프링 AOP는 프록시 객체를 자동으로 생성, 개발자는 공통 기능만 구현한 클래스 작성

Spring AOP의 특징

- ✓ 표준 자바 클래스로 작성
- ✓ Runtime 시점에 Advice가 적용
- ✓ 메서드 단위의 조인포인트만 가능

AOP 용어

이름	설명
조인포인트(Joinpoint)	공통 관심 모듈의 기능이 삽입되어 동작될 수 있는 위치 메서드 호출, 필드 값 변경시점등, 스프링은 메서드 호출만 가능
포인트컷	어떤 클래스의 조인포인트를 사용할 것인지 결정
어드바이스(Advice)	언제 공통 기능이 핵심 기능에 적용할 지를 정의 예> 게시판 목록 조회 시(핵심 기능)에 실행에 걸린 시간(공통 기능)을 출력한다.
위빙, 크로스컷팅	포인트 컷에 의해서 결정된 조인포인트에 지정된 어드바이스를 삽입하는 과정
애스팩트(Aspect)	공통기능 (포인트 컷과 어드바이스를 합쳐놓은 것)

Advice 용어

이름	설명
Before Advice	대상 객체의 메서드 호출 전에 공통 기능 실행
After Returning Advice	대상 객체의 메서드가 익셉션 없이 실행된 이후에 공통 기능 실행
After Throwing Adivce	대상 객체의 메서드를 실행하는 도중 익셉션이 발생한 경우에 공통 기능 실행
After Advice	대상 객체의 메서드를 실행하는 도중에 익셉션이 발생했는지의 여부에 상관없이 메서드 실행 후 공통 기능을 실행
Around Advice	대상 객체의 메서드 실행 전, 후 또는 익셉션 발생 시점에 공통 기능을 실행

스키마 기반 AOP

✓ <aop:*> 에 해당하는 태그 정보

태그명	설명
<aop:config>	AOP 설정 태그 중 루트, AOP 관련 설정 정보임을 나타냄
<aop:aspect>	Pointcut 과 advice 설정을 위해 사용
<aop:pointcut>	포인트컷을 지정하기 위해 사용, <aop:config>, <aop:aspect> 하위태그
<aop:advisor>	포인트컷과 어드바이스 클래스를 직접 연결할 경우 사용

스키마 기반 AOP

✓ Advice 관련 태그 정보

태그명	설명
<aop:before>	메서드 실행 전에 호출
<aop:after-returning>	메서드가 정상 종료 후 호출
<aop:after-throwing>	메서드가 예외를 발생시킬 때 적용, catch 블록과 유사
<aop:after>	메서드 실행 후 무조건 호출, finally 와 유사
<aop:around>	메서드 호출 이전, 이후, 예외 발생 등 모든 시점에 적용 가능

Pointcut 지시자

✓ execution

수식어패턴? 리턴타입패턴 클래스이름패턴? 메서드이름패턴(파라미터패턴)

[접근제한자] 반환타입 [클래스이름.] 메소드명(파라미터)

생략가능	필수	생략가능	필수
public	*	패키지포함	
protected			
private			

execution (수식어패턴? 리턴타입패턴 클래스이름패턴? 메서드이름패턴(파라미터패턴)

execution ([접근제한자] 반환타입 [클래스이름.]메소드명(파라미터))

execution(public * kr.co.mlec.board.service.BoardService.list (..))

AspectJ 에서의 사용되는 표현식

✓ 와일드 카드 연산자

와일드 카드	설명
*	모든문자
..	0개 이상의 문자
+	주어진 타입의 임의의 서브 클래스나 서브 인터페이스

AspectJ 에서의 사용되는 표현식

✓ 패턴 매칭

형식	설명
set* (..)	set으로 시작하는 모든 메서드
* main(..)	리턴타입 상관없이 이름이 main 인 모든 메서드

AspectJ 에서의 사용되는 표현식

✓ Type

형식	설명
Java.io.*	Java.io 밑에 있는 모든 클래스 지칭
ch4.aop..*	ch4.aop 패키지 및 그 하위 모든 패키지의 클래스 지칭
Number+	+ 는 서브 타입 지칭, Number 타입으로 표현되는 모든 하위 클래스
!(Number+)	Number 타입으로 표현되지 않는 모든 클래스

AspectJ 에서의 사용되는 표현식

✓ Modifier 설정

형식	설명
public static void main(..)	매개변수에 상관없이 접근제한자가 public인 main 함수
!private * * (..)	접근제한자가 private 가 아닌 모든 메서드
* main(..)	접근제한자 상관하지 않는 형식

어노테이션 기반 AOP

- ✓ XML 설정부분을 간소화 하기 위해 어노테이션을 활용하는 방식
- ✓ XML 문서에 어노테이션을 활용하기 위한 선언 필요
  - <aop:aspectj-autoproxy />
- ✓ 클래스 위쪽에 @Aspect 선언

스키마 기반 AOP

✓ 스키마와 관련된 어노테이션

태그명	설명
<aop:before>	`@Before("execution(public * anno.*Controller.*(..))")`
<aop:after-returning>	`@AfterReturning(pointcut="execution(public * anno.*Controller.*(..))", returning="retVal")`
<aop:after-throwing>	`@AfterThrowing( pointcut="execution(public * anno.*Controller.*(..))", throwing="ex")`
<aop:after>	`@After("execution(public * anno.*Controller.*(..))")`
<aop:around>	`@Around("execution(public * anno.*Controller.*(..))")`

myBatis

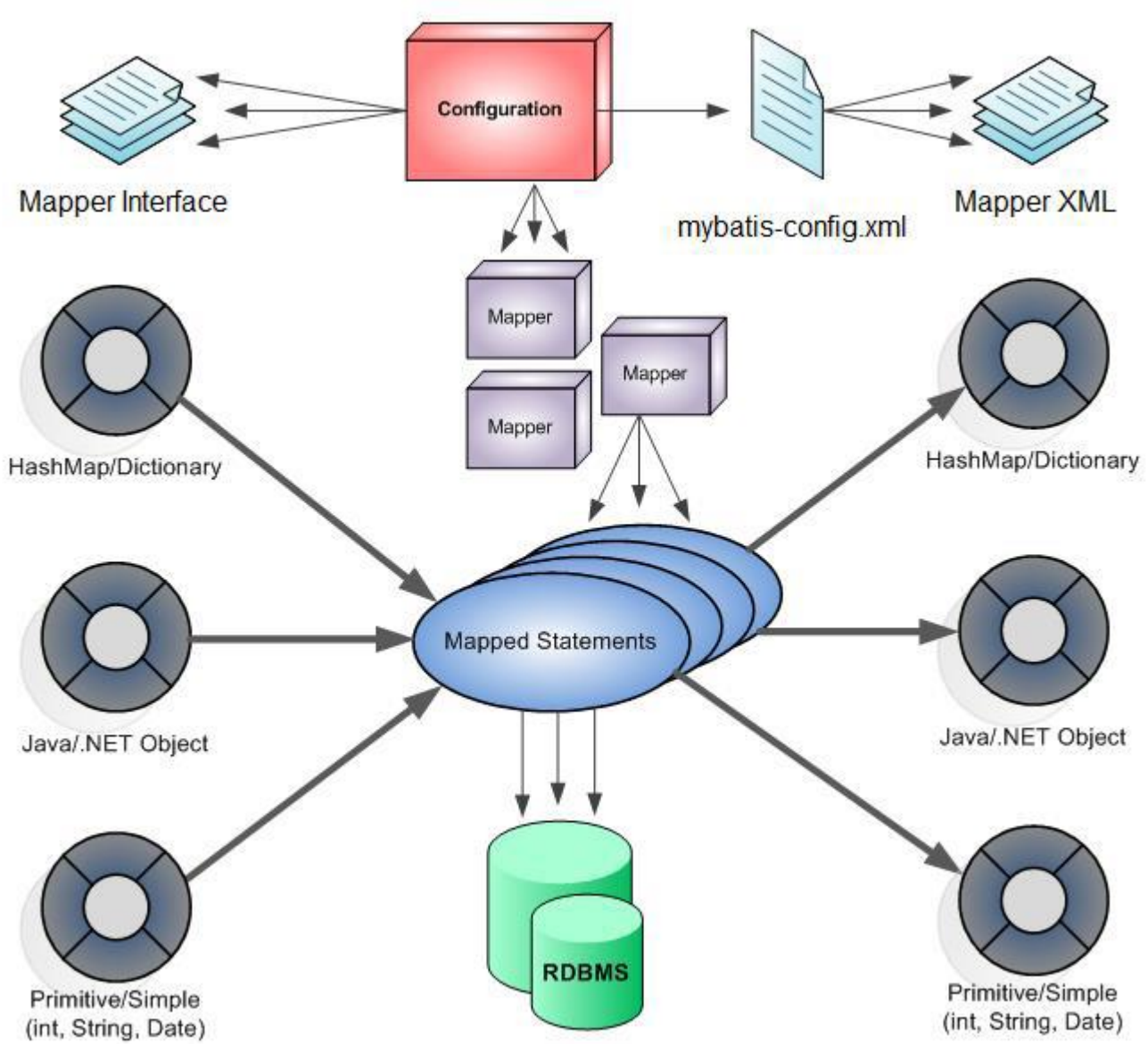
myBatis

- ✓ myBatis의 정의 및 특성
- ✓ myBatis API 및 SQL 관련 XML 태그
- ✓ 실습

myBatis 정의 및 특성

- ✓ 2010년에 iBatis가 Apache를 떠나 Google Code 로 이동하면서 이름이 myBatis로 변경됨.
- ✓ 더 빠른 JDBC 코딩을 위한 일반화된 프레임워크.
- ✓ SQL을 자바 코드가 아닌 XML로 따로 분리
- ✓ SQL의 실행 결과를 Map 또는 자바 클래스로 자동 매핑
- ✓ SQL을 XML 이나 인터페이스내에 어노테이션을 활용하여 처리

myBatis 정의 및 특성



myBatis 적용

- ✓ 환경설정 파일(SqlMapConfig)과 실제 쿼리를 적용할 파일 (Mapper)이 필요(XML, Annotation)
- ✓ 자바파일에서는 설정된 환경설정 정보를 이용하여 SqlSession객체를 얻어온 다음 myBatis와 연동

myBatis 다운로드

- ✓ https://github.com/mybatis/mybatis-3/releases 다운로드
- ✓ 프로젝트에 다운받은 파일을 압축 해재 한 후 mybatis-x.x.x.jar 파일을 클래스패스에 추가(웹 프로젝트인 경우 WEB-INF/lib에 추가)
- ✓ Maven 일 경우 Dependency 추가

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>x.x.x</version>
</dependency>
```

xml 환경 설정

✓ 설정 루트

```
<configuration>
   - vo 객체의 타입 Alias 설정
   <typeAliases>
      <typeAlias alias="xxx" type="xxx.Xxx" />
   </typeAliases>
   - 설정파일 내부에서 사용하는 값을 파일을 활용하여 추출
   <properties resource="db.properties" />
   …… 생략…
</configuration>
```

xml 환경 설정

```
<configuration>
    …… 생략…
    - 트랜잭션 및 데이터베이스 설정
    <environments default="development">
      <environment id="development">
        <transactionManager type="JDBC"/>
        <dataSource type="POOLED">
           <property name="driver" value="…"/>
           <property name="url" value="…"/>
           <property name="username" value="hr"/>
           <property name="password" value="hr"/>
        </dataSource>
      </environment>
    </environments>
</configuration>
```

트랜잭션

UNPOOLED, POOLED, JNDI

데이터베이스

xml 환경 설정

- 매퍼정보 설정

```
    <mappers>
        <mapper resource="common/db/xxx.xml"/>
    </mappers>
</configuration>
```

XML Mapper 사용 태그

구문형식	속성	하위 요소	사용 용도
<select>	id parameteType resultType resultMap	모든 동적 요소	데이터 조회
<sql>	id		공통 쿼리 묶기
<include>	refid		<sql> 로 생성된 컴포넌트 삽입
<insert>	id parameterType	모든 동적 요소 <selectKey>	데이터 입력
<update>	id parameterType	모든 동적 요소	데이터 수정
<delete>	id parameterType	모든 동적 요소	데이터 삭제

XML Mapper 동적 태그

구문형식	속성	사용법
<foreach>	Collection Item Open Separator close	<foreach item="addr" open="(" separator="," close=")" collection="addrs"> #{addr} </foreach>
<if>	test	<if test="id != null">id = #{id}</if>
<choose> <when> <otherwise>	test	<choose> <when test="searchType == 'id'"> id like #{searchWord} </when> <otherwise> addr like #{searchWord} </otherwise> </choose>

XML Mapper 동적 태그

구문형식	속성	사용법
<where>		<where> <if test="writer != null"> writer like #{writer} </if> <if test="title != null"> AND title like #{title} </if> </where>
<set>		<set> <if test="writer != null"> writer = #{writer}, </if> <if test="title != null"> title = #{title} </if> </set>

SqlSession 클래스 중요 메소드

이름	설명
java.util.List	selectList(String id)
java.util.List	selectList(String id, Object parameterObject)
java.lang.Object	selectOne(String id)
java.lang.Object	selectOne(String id, Object parameterObject)
int	insert(String id, Object parameter)
int	delete(String id, Object parameter)
int	update(String id, Object parameter)

Spring db

datasource 설정

✓ DriverManager를 이용한 DataSource 설정

: 일반 JDBC 방식

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.
                                    DriverManagerDataSource"
      p:driverClassName="oracle.jdbc.driver.OracleDriver"
      p:url="jdbc:oracle:thin:@localhost:1521:XE"
      p:username="hr"
      p:password="hr" />
```

datasource 설정

✓ 커넥션 풀을 이용한 DataSource 설정

: DBCP(Jakara Commons Database Connection Pool) API를 이용한 설정

```
<bean id="dataSource"
      class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close"
      p:driverClassName="oracle.jdbc.driver.OracleDriver"
      p:url="jdbc:oracle:thin:@localhost:1521:XE"
      p:username="hr"
      p:password="hr" />
```

datasource 설정

✓ JNDI를 이용한 DataSource 설정

: 웹서버(WebLogic, Tomcat등)에 등록된 JNDI로부터 설정

```
<jee:jndi-lookup id="dataSource"
                 jndi-name="java:comp/env/jdbc/bitdb"/>
```

spring + mybatis 연동 - xml

- ✓ mybatis-spring-x.x.x.jar, mybatis-x.x.x.jar 파일을 클래스 패스 설정
- ✓ DAO에서 SqlSession을 얻기 위해 XML 문서에 *SqlSessionTemplate* *를 선언*

```
<bean id="dataSource" destroy-method="close"
      class="org.apache.commons.dbcp.BasicDataSource">
   …… 생략
</bean>

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

<bean id="sqlSessionTemplate"
      class="org.mybatis.spring.SqlSessionTemplate">
  <constructor-arg ref="sqlSessionFactory" />
</bean>
```

sqlMapConfig.xml

```
<configuration>
    <typeAliases>
        <typeAlias alias="xxx" type="xxx.vo.XxxVO" />
    </typeAliases>
    <mappers>
        <mapper resource="config/sqlmap/oracle/xxx.xml" />
    </mappers>
</configuration>
```

트랜잭션

- ✓ Container 가 제공되는 가장 대표적 서비스
- ✓ 설정파일내 TransactionManager 설정만으로 소스코드 내에 Transaction 관련 코드를 사용하지 않고 자동 Transaction 이 가능
- ✓ Spring 에서는 서로 다른 트랜잭션 API를 지원하며 지원 종류는 JDBC, JTA, JDO, JPA, 하이버네이트 등이 있음
- ✓ 트랜잭션 사용방법은 환경설정 파일인 xml 에 선언하여 사용하는 방법과 프로그램에서 직접 제어하는 방법 2가지가 있음

AOP 방식의 트랜잭션

1. <beans> 태그에 xmlns:tx 관련 부분을 추가

```
<beans xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="“http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd">
```

2. DataSource 추가

```
<bean id="dataSource" 생략…  />
```

3. 트랜잭션 매니져 설정 : 실제 트랜잭션 관리 처리

```
<bean id="transactionManager"
      class=
   "org.springframework.jdbc.datasource.DataSourceTransactionManager"
      p:dataSource-ref="dataSource" />
```

AOP 방식의 트랜잭션

4. 트랜잭션 매니저를 어드바이스로 설정

```
<tx:advice id="txAdvice"
        transaction-manager="transactionManager">
     <tx:attributes>
        <tx:method name="tran*" />
        <tx:method name="select*" read-only="true" />
    </tx:attributes>
</tx:advice>
```

5. 트랜잭션 AOP 설정을 통한 적용

```
<aop:config>
    <aop:pointcut id="tranMethod"
          expression="execution(public * member.*Service.*(..))“/>
    <aop:advisor advice-ref="txAdvice“ pointcut-ref="tranMethod" />
</aop:config>
```

어노테이션 방식의 트랜잭션

1. annotation-driven 설정

```
<tx:annotation-driven
        transaction-manager="transactionManager" />
```

2. 클래스에 어노테이션 표기법 추가

```
@Service("memberService")
public class MemberServiceImpl implements MemberService {
    …… 생략
    @Transactional
    public void registMember(MemberVO memberVO) throws Exception {
        memberDAO.insertMember(memberVO);
    }
}
```

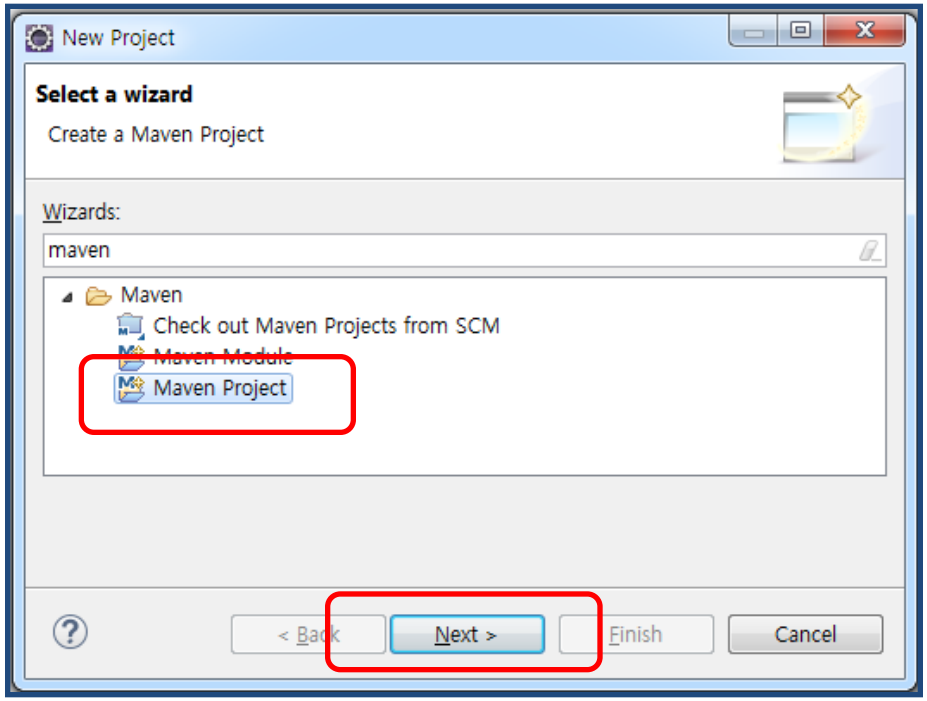
Spring MVC

첫번째 MVC 프로젝트

첫번째 만드는 MVC 프로젝트

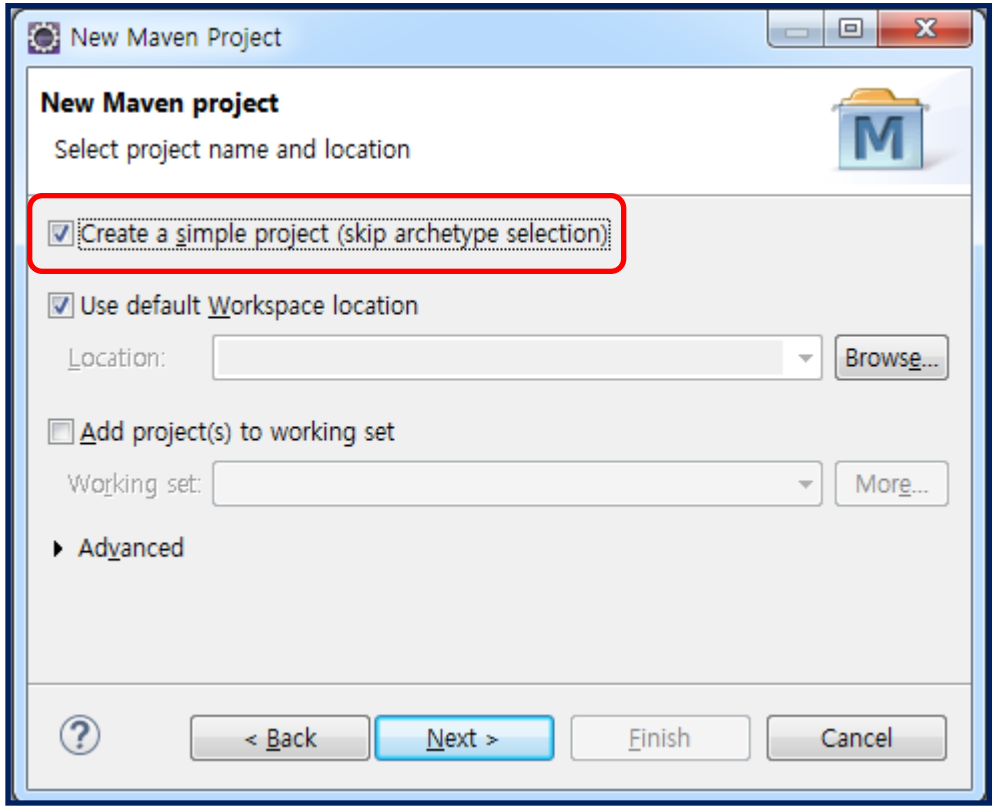
1. 프로젝트 생성

   - Maven 프로젝트를 선택 후 Next 버튼을 클릭



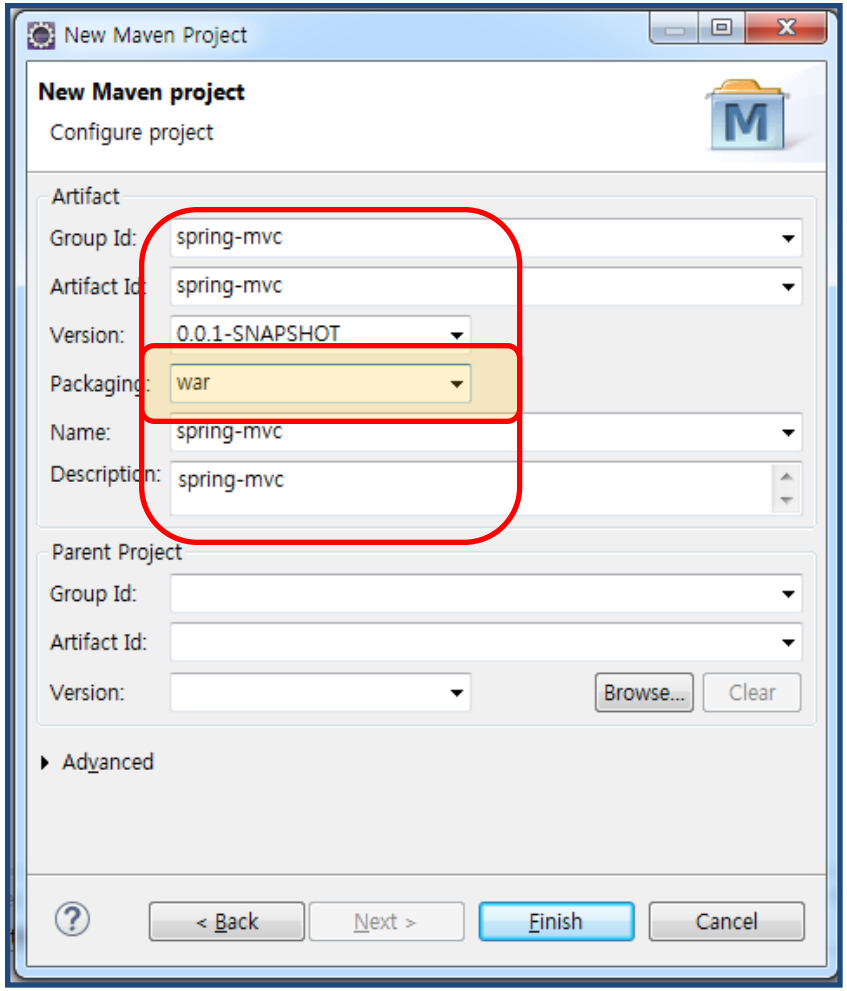
첫번째 만드는 MVC 프로젝트

- Create a simple project 체크 후 Next 클릭



첫번째 만드는 MVC 프로젝트

- 입력창에 다음과 같이 입력 후 Finish 클릭



첫번째 만드는 MVC 프로젝트

2. pom.xml

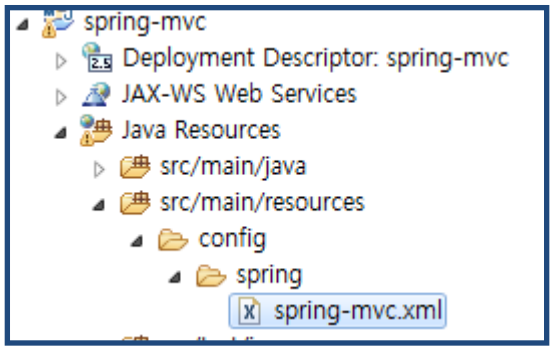
- dependency 추가

```
<dependencies>
  <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.2</version>
      <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.0.1</version>
      <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>4.1.6.RELEASE</version>
  </dependency>
</dependencies>
```

첫번째 만드는 MVC 프로젝트

3. 스프링 설정 파일 작성

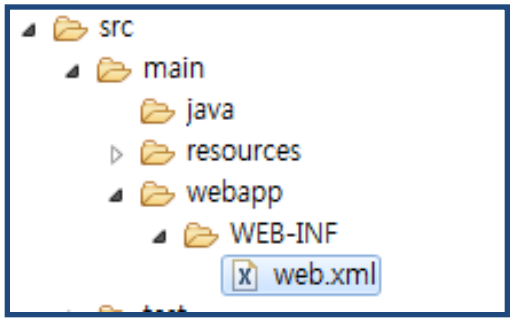
- src ₩ main ₩ resources 하위에 config ₩ spring 폴더 작성
- 제공된 spring-mvc.xml 파일 붙여 넣기



첫번째 만드는 MVC 프로젝트

4. 기본 디렉토리 생성

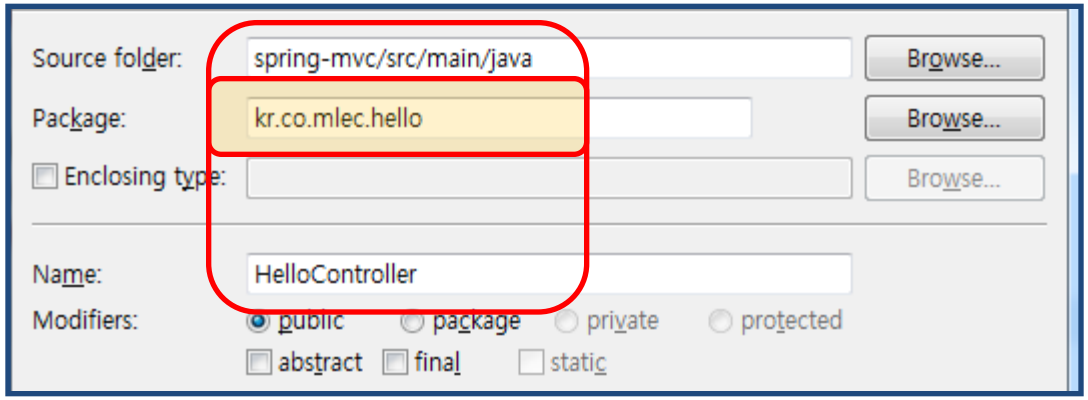
- / spring-mvc / src / main / webapp 에 WEB-INF 폴더 생성
- 제공된 web.xml 파일을 붙여넣기



첫번째 만드는 MVC 프로젝트

5. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 HelloController 생성



첫번째 만드는 MVC 프로젝트

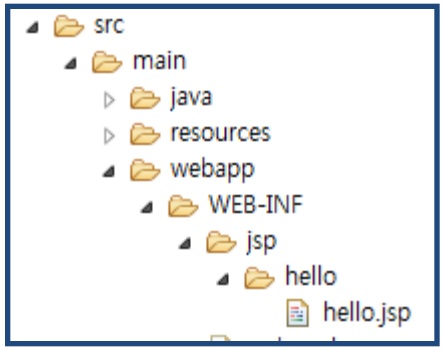
- HelloController 소스 작성

```
@Controller
public class HelloController {
    @RequestMapping("/hello/hello.do")
    public ModelAndView hello() {
        ModelAndView mav = new ModelAndView("hello/hello");
        mav.addObject("msg", "hi 스프링 MVC~~");
        return mav;
    }
}
```

첫번째 만드는 MVC 프로젝트

6. 웹 페이지 만들기

- / spring-mvc / webapp / WEB-INF 에 jsp / hello 폴더 생성
- 생성된 폴더에 hello.jsp 생성



- 소스 작성

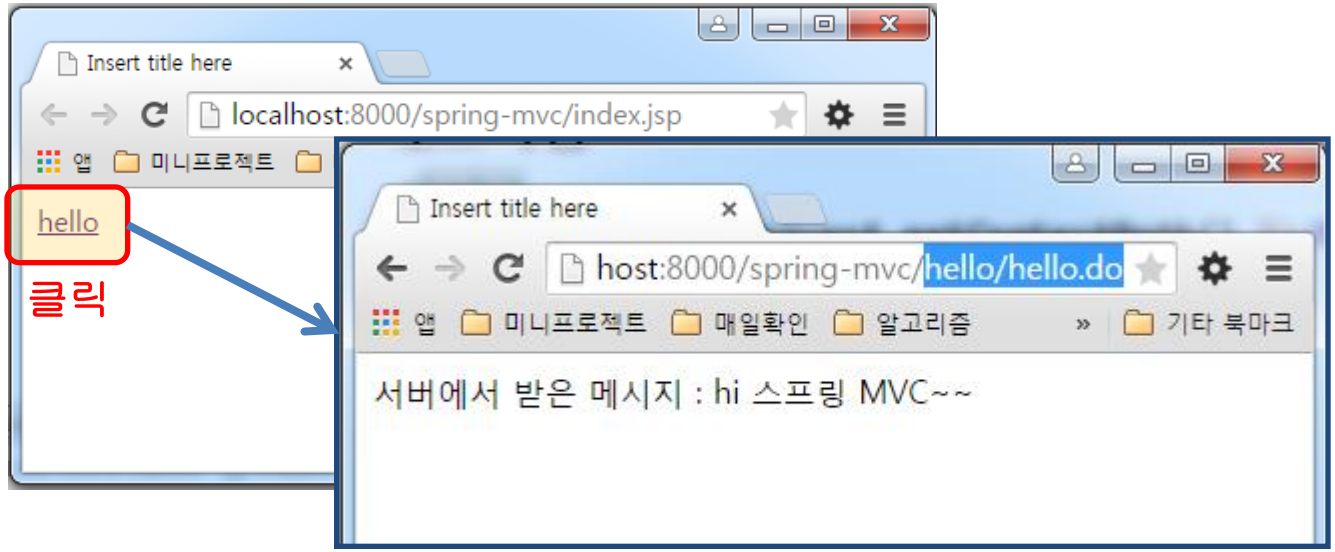
```
<body>
    서버에서 받은 메시지 : ${msg}
</body>
```

첫번째 만드는 MVC 프로젝트

7. 호출

- / spring-mvc / webapp 폴더에 index.jsp 파일 생성
- 소스 작성

```
<body>
    <a href="<%=request.getContextPath() %>/hello/hello.do">
        hello
    </a><br />
</body>
```

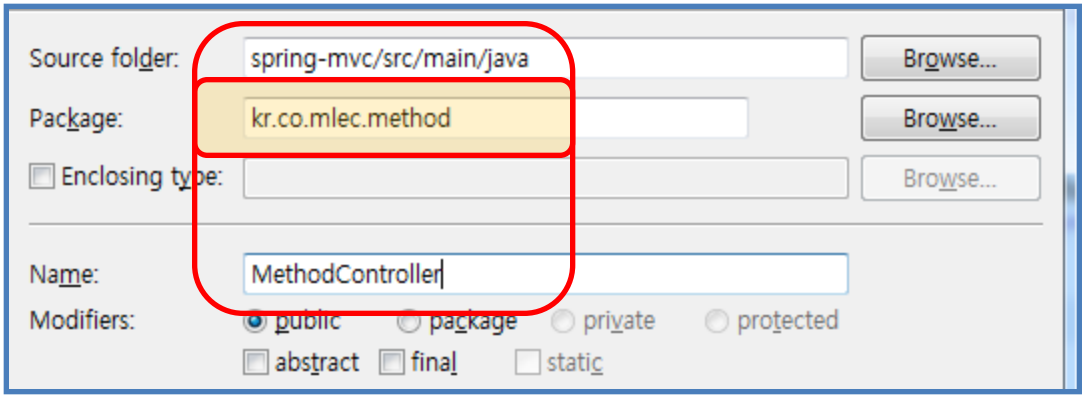


RequestMapping

RequestMapping

1. 컨트롤러 만들기

   - / spring-mvc / src / main / java 폴더에 MethodController 생성



RequestMapping

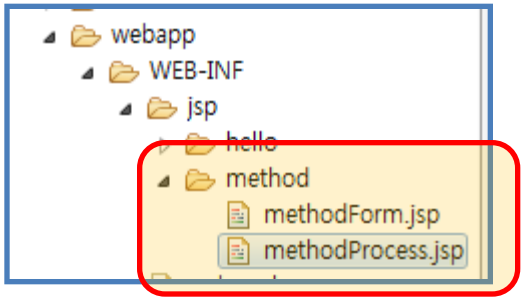
- MethodController 소스 작성

```
@Controller
public class MethodController {
    @RequestMapping(value="/method/method.do",
                    method=RequestMethod.GET)
    public String callGet() {
        return "method/methodForm";
    }
    @RequestMapping(value="/method/method.do",
                    method=RequestMethod.POST)
    public String callPost() {
        return "method/methodProcess";
    }
}
```

RequestMapping

2. 웹 페이지 만들기

- / spring-mvc / webapp / WEB-INF / jsp 에 method 폴더 생성
- 생성된 폴더에 methodForm.jsp, methodProcess.jsp 생성



RequestMapping

- 코드 작성

methodForm.jsp

```
<body>
    <form action="<%= request.getContextPath() %>/method/method.do"
                method="POST">
            <input type="submit" value="호출"  />
    </form>
</body>
```

methodProcess.jsp

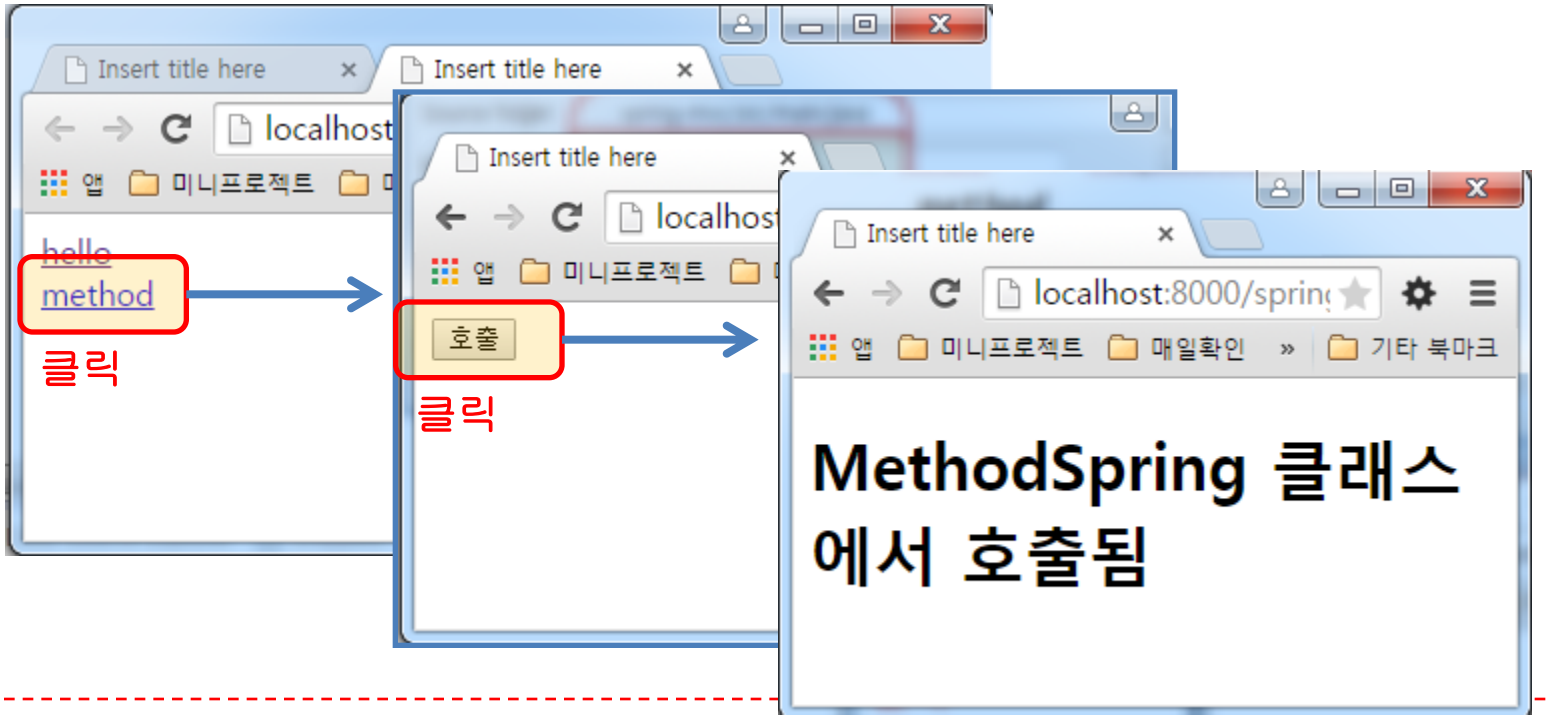
```
 <body>
       <h1>MethodSpring 클래스에서 호출됨</h1>
 </body>
```

RequestMapping

3. 호출

- index.jsp 파일 코드 추가

```
<a href="<%=request.getContextPath() %>/method/method.do">
    method
</a><br />
```



RequestMapping

4. 컨트롤러 코드 변경

```
코드 생략
@RequestMapping(value="/method/method.do")
public class MethodController {
    @RequestMapping(method=RequestMethod.GET)
    public String callGet() {
        코드 생략
    }
    @RequestMapping(method=RequestMethod.POST)
    public String callPost() {
        코드 생략
    }
}
```

RequestMapping

5. 호출

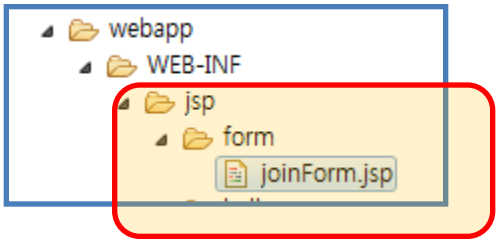
   - index.jsp 파일을 실행하여 페이지 결과 확인

Form 데이터 처리

Form 데이터 처리

1. 웹 페이지 만들기

   - / spring-mvc / webapp / WEB-INF / jsp 에 form 폴더 생성
   - 생성된 폴더에 joinForm.jsp 생성



Form 데이터 처리

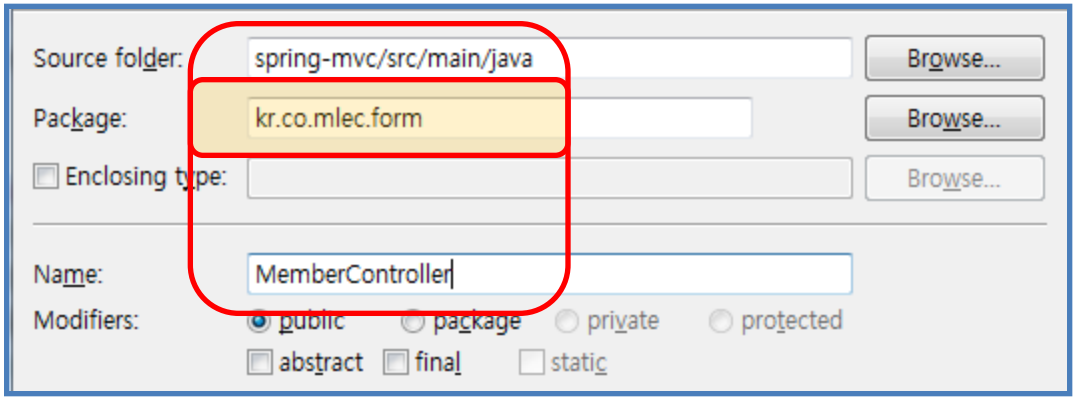
2. joinForm.jsp 코드 작성

```
<form action="<%= request.getContextPath() %>/form/join.do"
      method="POST">
    아이디 : <input type="text" name="id" size="20" /><br />
    암호 : <input type="text" name="password" size="20" /><br />
    이름 : <input type="text" name="name" size="20" /><br />
    <input type="submit" value="저장" />
</form>
```

Form 데이터 처리

3. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 MemberController 생성



Form 데이터 처리

- MemberController 소스 작성

```
@Controller
@RequestMapping("/form")
public class MemberController {
    @RequestMapping("/joinForm.do")
    public String joinForm() {
        return "form/joinForm";
    }
}
```

Form 데이터 처리

4. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/form/joinForm.do">
    form
</a><br />
```

Form 데이터 처리

5. VO 만들기

   - / spring-mvc / src / main / java / form 폴더에 MethodVO 생성

```
private String id;

private String password;

private String name;

set, get 메서드 생성
```

Form 데이터 처리

6. MemberController 코드 추가

```
@RequestMapping("/join.do")
public String join(HttpServletRequest request) {
    String id = request.getParameter("id");
    String password = request.getParameter("password");
    String name = request.getParameter("name");
    MemberVO member = new MemberVO();
    member.setId(id);
    member.setPassword(password);
    member.setName(name);
    request.setAttribute("member", member);
    return "form/memberInfo";
}
```

Form 데이터 처리

7. form 폴더 하위 memberInfo.jsp 페이지 작성

```
<body>
    <h1>회원 정보</h1>
    id : ${member.id}<br />
    password : ${member.password}<br />
    name : ${member.name}<br />
</body>
```

- index.jsp 페이지 실행 후 테스트 진행

Form 데이터 처리

8. @RequestParam - MemberController 소스 수정

```
@RequestMapping("/join.do")
public String join(HttpServletRequest request,
                   @RequestParam("id") String id,
                   @RequestParam("password") String pass,
                   @RequestParam("name") String name) {
    MemberVO member = new MemberVO();
    member.setId(id);
    member.setPassword(pass);
    member.setName(name);
    request.setAttribute("member", member);
    return "form/memberInfo";
}
```

Form 데이터 처리

9. index.jsp 페이지 호출 후 테스트 진행

Form 데이터 처리

10. VO 객체 활용 - MemberController 소스 수정

```
@RequestMapping("/join.do")
public String join(MemberVO member) {
    System.out.println(member.getId());
    System.out.println(member.getPassword());
    System.out.println(member.getName());
    return "form/memberInfo";
}
```

Form 데이터 처리

11. form 폴더 하위 memberInfo.jsp 페이지 코드 수정

```
id : ${memberVO.id}
<br /> password : ${memberVO.password}
<br /> name : ${memberVO.name}
<br />
```

- index.jsp 페이지 실행 후 테스트 진행

Form 데이터 처리

12. @ModelAttribute - JSP에서 사용할 공유 객체명 변경

```
@RequestMapping("/join.do")
public String join(
            @ModelAttribute("member") MemberVO member) {
    System.out.println(member.getId());
    System.out.println(member.getPassword());
    System.out.println(member.getName());
    return "form/memberInfo";
}
```

Form 데이터 처리

13. form 폴더 하위 memberInfo.jsp 페이지 코드 수정

```
id : ${member.id}
<br /> password : ${member.password}
<br /> name : ${member.name}
<br />
```

- index.jsp 페이지 실행 후 테스트 진행

Form 데이터 처리

14. ModelAndView - view 페이지 및 view에서 사용할 객체 공유

```
@RequestMapping("/join.do")
public ModelAndView join(MemberVO member) {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("form/memberInfo");
    mav.addObject("member", member);
    return mav;
}
```

- index.jsp 페이지 실행 후 테스트 진행

Form 데이터 처리

15. Model 객체 및 redirect 사용

```
@RequestMapping("/join.do")
public String join(MemberVO member, Model model) {
    System.out.println(member.getId());
    System.out.println(member.getPassword());
    System.out.println(member.getName());
    model.addAttribute("msg", "가입이 완료되었습니다.");
    return "form/joinForm.do";
    // return "redirect:/form/joinForm.do";
}
```

Form 데이터 처리

16. joinForm.jsp 페이지 코드 수정

```
<script>
    if ("${msg}") {
        alert("${msg}");
    }
</script>
```

- index.jsp 페이지 실행 후 테스트 진행

@ResponseBody

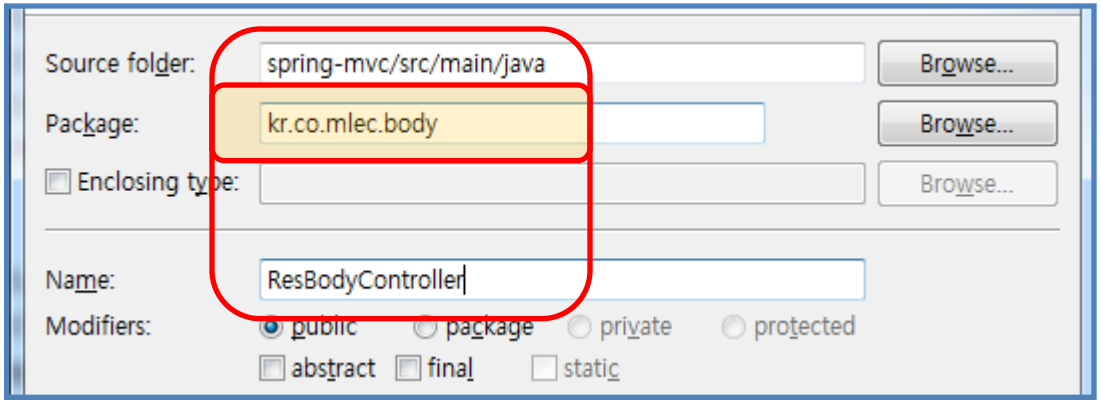
@ResponseBody

@ResponseBody 는 xml 또는 json 과 같은 메시지 기반의 서비스를 만들 경우 사용

예> AJAX 서비스를 제공하는 컨트롤러

1. 컨트롤러 만들기

   - / spring-mvc / src / main / java 폴더에 ResBodyController 생성



@ResponseBody

2. 코드 작성

```
@Controller
@RequestMapping("/ajax")
public class ResBodyController {
    @RequestMapping("/resBody.do")
    @ResponseBody
    public String resStringBody() {
        return "OK, 성공";
    }
}
```

@ResponseBody

3. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/ajax/resBody.do">
    문자열 응답
</a><br />
```

@ResponseBody

4. 한글처리

- spring-mvc.xml 파일 수정 후 페이지 호출 테스트

```
<mvc:annotation-driven>
  <mvc:message-converters>
    <bean class=
     "org.springframework.http.converter.StringHttpMessageConverter">
      <property name="supportedMediaTypes">
        <list>
          <value>text/html; charset=UTF-8</value>
        </list>
      </property>
    </bean>
  </mvc:message-converters>
</mvc:annotation-driven>
```

@ResponseBody

5. JSON 응답 처리

- ResBodyController 코드 추가

```
@RequestMapping("/resBody.json")
@ResponseBody
public Map<String, String> resJsonBody() {
    Map<String, String> result = new HashMap<>();
    result.put("id", "sbc");
    result.put("name", "hong");
    result.put("addr", "서울");
    return result;
}
```

@ResponseBody

6. JSON 처리 컨버터 등록

- spring-mvc.xml 파일 수정

- mvc:message-converters의 하위 태그로 등록

```
<bean class="org.springframework.http.converter
                    .json.MappingJackson2HttpMessageConverter">
  <property name="supportedMediaTypes">
     <list>
        <value>text/html; charset=UTF-8</value>
        <value>application/json; charset=UTF-8</value>
     </list>
  </property>
</bean>
```

@ResponseBody

7. *MappingJackson2HttpMessageConverter 에서 사용하는 라이브러리 다운로드*

- pom.xml 파일 dependency 추가

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.5.3</version>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.5.3</version>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.5.3</version>
</dependency>
```

@ResponseBody

8. JSON URL 매핑 등록

- web.xml 파일 수정

```
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.json</url-pattern>
</servlet-mapping>
```

@ResponseBody

9. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/ajax/resBody.json">
    JSON 응답
</a><br />
```

@ResponseBody

10. JSON 응답 처리 (VO)

- ResBodyController 코드 추가

```
@RequestMapping("/resVOBody.json")
@ResponseBody
public MemberVO resJsonVOBody() {
    MemberVO vo = new MemberVO();
    vo.setId("sbc");
    vo.setName("sbc");
    vo.setPassword("1234");
    return vo;
}
```

@ResponseBody

11. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/ajax/resVOBody.json">
    JSON VO 응답
</a><br />
```

@ResponseBody

12. JSON 응답 처리 (List<String>)

- ResBodyController 코드 추가

```
@RequestMapping("/resStringListBody.json")
@ResponseBody
public List<String> resJsonStringListBody() {
    List<String> list = new ArrayList<>();
    for (int i = 1; i < 4; i++) {
    list.add(String.valueOf(i));
    }
    return list;
}
```

@ResponseBody

13. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="……/ajax/resStringListBody.json">
    JSON List(문자열) 응답
</a><br />
```

@ResponseBody

14. JSON 응답 처리 (List<VO>)

- ResBodyController 코드 추가

```
@RequestMapping("/resVOListBody.json")
@ResponseBody
public List<MemberVO> resJsonVOListBody() {
    List<MemberVO> list = new ArrayList<>();
    for (int i = 1; i < 4; i++) {
        MemberVO vo = new MemberVO();
        vo.setId("sbc");
        vo.setName("sbc");
        vo.setPassword("1234");
        list.add(vo);
    }
    return list;
}
```

@ResponseBody

15. 호출

- index.jsp 파일 코드 추가 후 테스트

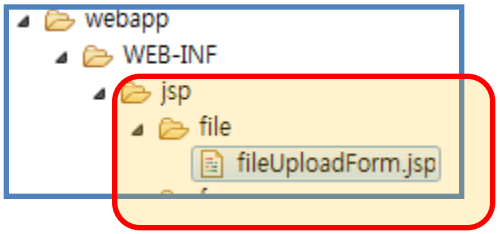
```
<a href="...../ajax/resVOListBody.json">
    JSON List(VO) 응답
</a><br />
```

컨트롤러 없이 페이지 매핑

컨트롤러 없이 페이지 매핑

1. 웹 페이지 만들기

   - / spring-mvc / webapp / WEB-INF / jsp 에 file 폴더 생성
   - 생성된 폴더에 fileUploadForm.jsp 생성



컨트롤러 없이 페이지 매핑

2. uploadForm.jsp 코드 작성

```
<h2>파일 업로드 테스트</h2>
<form method="post" enctype="multipart/form-data"
      action="……/file/upload.do">
    <input type="text" name="id" value="test" /><br />
    <input type="file" name="attachFile1" /><br />
    <input type="file" name="attachFile2" /><br />
    <input type="submit" value="업로드" />
</form>
```

컨트롤러 없이 페이지 매핑

3. spring-mvc.xml 파일에 내용 추가

```
<mvc:view-controller path="/file/uploadForm.do"
                     view-name="file/fileUploadForm"/>
```

컨트롤러 없이 페이지 매핑

4. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="……/upload/uploadForm.do">
    파일 업로드
</a><br />
```

파일 업로드

파일 업로드

1. spring-mvc.xml 파일에 내용 추가

```
<bean id="multipartResolver"
    class= "org.springframework.web.multipart
                        .commons.CommonsMultipartResolver">
     <!-- 최대 업로드 파일 사이즈 : 10MB -->
     <property name="maxUploadSize" value="10485760" />
</bean>
```

파일 업로드

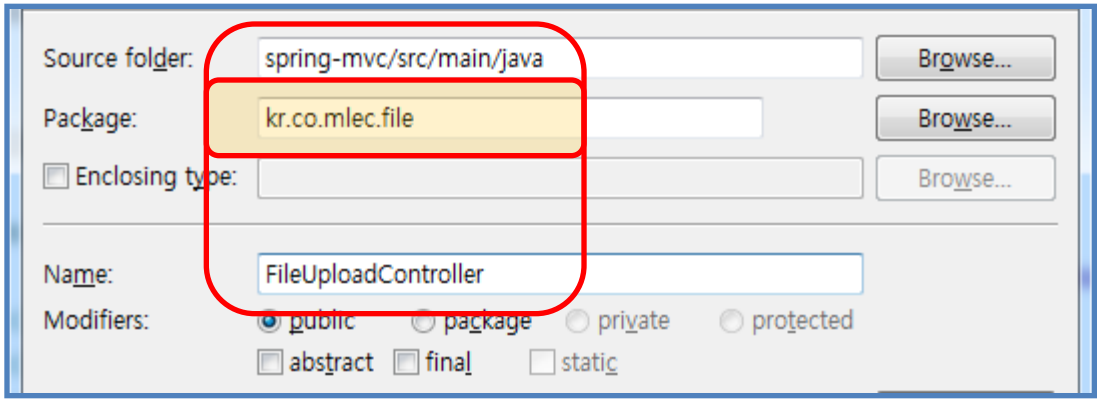
2. pom.xml 파일 dependency 추가

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
```

파일 업로드

3. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 UploadController 생성

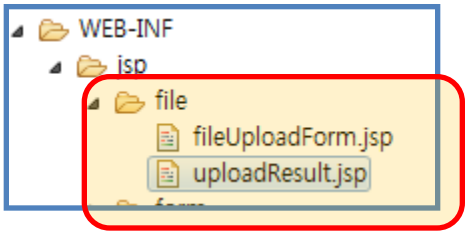


- 제공된 UploadController 자바파일의 내용을 복사해서 붙여넣기 한다.

파일 업로드

4. 결과 페이지 만들기

- / spring-mvc / webapp / WEB-INF / jsp 에 file 폴더 생성
- 생성된 폴더에 uploadResult.jsp 생성



- 코드 작성

```
<body>
    파일 업로드 성공
</body>
```

- index.jsp 페이지를 실행하여 결과 확인