

Spring Day-1

1. 개요

POJO(Plain Old Java Object) 방식: POJO는 Java EE의 EJB 를 사용하면서 해당 플랫폼에 종속되어 있는 무거운 객체들을 만드는 것에 반발하며 나타난 용어다. 별도의 프레임워크 없이 Java EE를 사용할 때에 비해 특정 인터페이스를 직접 구현하거나 상속받을 필요가 없어 기존 라이브러리를 지원하기가 용이하고, 객체가 가볍다.

Spring boot : 스프링부트는 스프링 기반 애플리케이션을 쉽게 생성하고 배포하기 위해 개발된 모듈이다. 기본적으로 의존성 관리와 라이브러리의 설정을 해주며, 웹서버가 하나의 파일에 임베드된 상태로 빌드되기 때문에 배포(Deploy)가 매우 간편해진다

❖ Spring Core

Spring 프레임워크의 근간이 되는 IoC(또는 DI) 기능을 지원하는 영역을 담당
BeanFactory를 기반으로 Bean 클래스들을 제어할 수 있는 기능을 지원

❖ Spring Context

Spring Core에서 지원하는 기능 외에 추가적인 기능들과 좀 더 쉬운 개발이 가능하도록 스프링 기반에서 구현된 Bean 객체에 대한 접근 방법을 제공
JNDI, EJB등을 위한 Adapter를 포함

❖ Spring DAO

JDBC 기반에서 일관된 방법으로 쉽고, 편리하게 DAO를 개발 할 수 있도록 지원
Spring DAO를 사용하면 기존의 DAO보다 적은 코드로 편리하게 DAO 개발이 가능

❖ Spring ORM

Object Relation Mapping 프레임워크인 Hibernate, MyBatis, JDO, JPA와의 연동 지원
Spring ORM을 사용해 Hibernate, MyBatis, JDO 프레임워크와 쉬운 통합이 가능

❖ Spring AOP

Aspect Oriented Programming을 지원하는 기능 AOP Alliance 기반에서 개발됨
애플리케이션 곳곳에 산재된 횡단적 관심사(로그인, CRUD)를 분리하여 모듈에 공통적으로 적용

❖ Spring Web

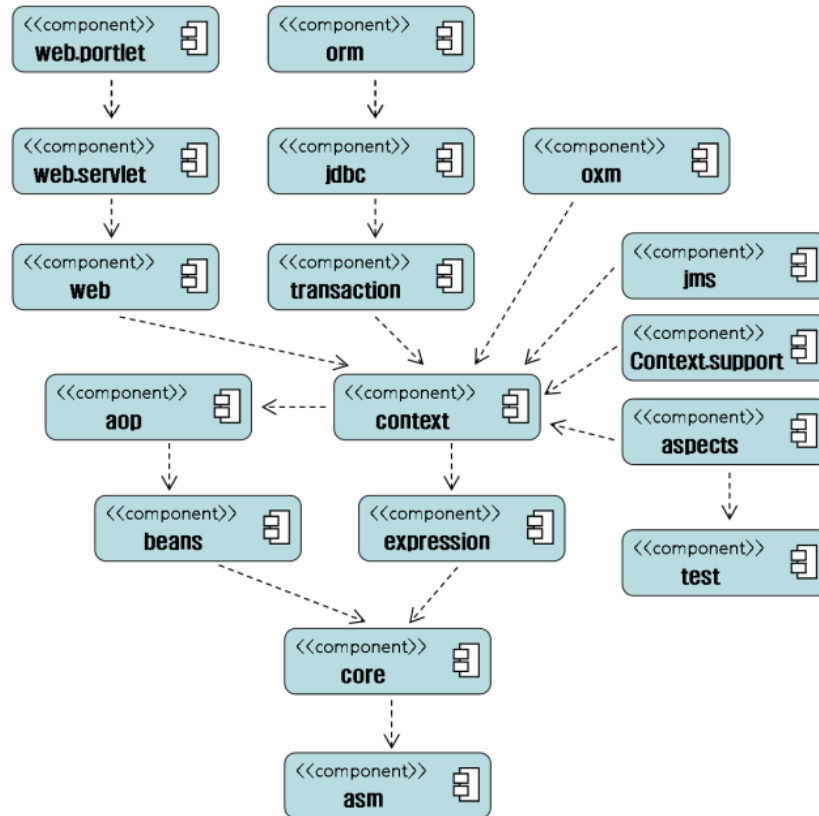
Web Application 개발에 필요한 Web Application Context, Multipart Request 지원
Struts, Webwork 등의 프레임워크와 통합을 지원

❖ Spring Web MVC

Spring에서 독립적으로 Web UI Layer에 Model-View-Controller를 지원
기존에 Struts, Webwork가 지원했던 기능을 Spring Web MVC를 이용하여 대체가능
Velocity, Excel, PDF와 같은 다양한 UI 기술들을 사용하기 위한 API를 제공

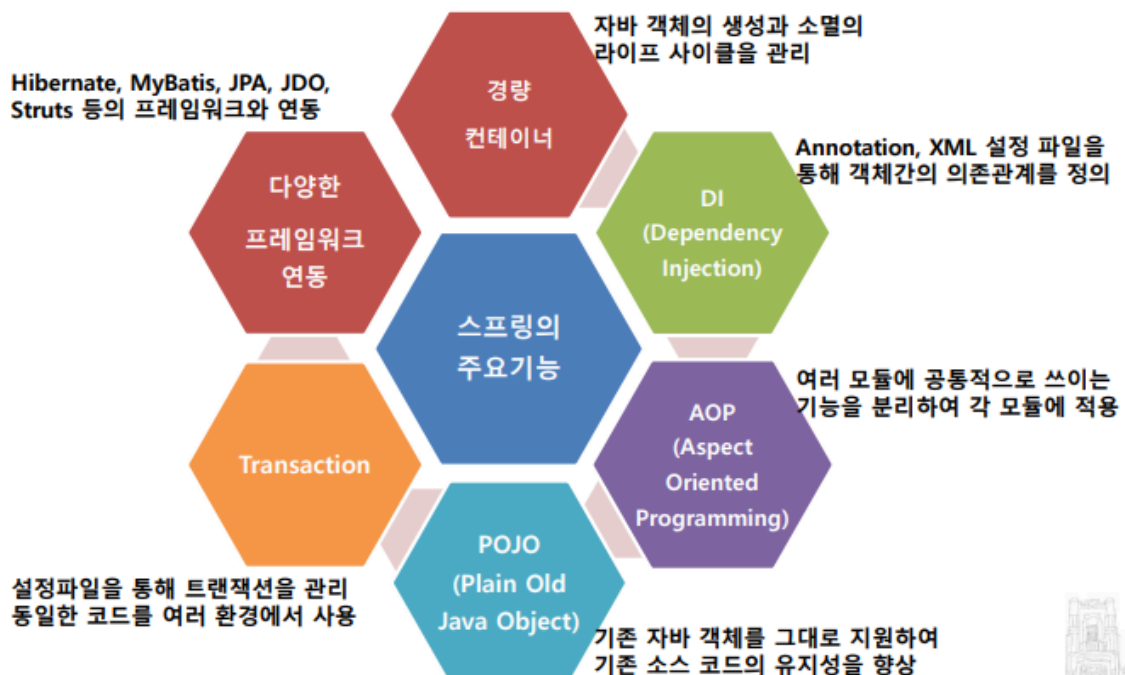
참고사이트

- 스프링 공식 사이트 : <http://spring.io>
- 한국 스프링 사용자 모임 : <http://www.ksug.org>
- 전자정부 프레임워크 포털 : <http://www.egovframe.go.kr>
- 스프링 레퍼런스 가이드 :
<https://docs.spring.io/spring-framework/docs/3.1.x/spring-frameworkreference/html/>
<https://docs.spring.io/spring-framework/docs/4.1.x/spring-frameworkreference/html/>
<https://docs.spring.io/spring-framework/docs/current/reference/html/>
- 스프링 3.1 레퍼런스 가이드(번역) : <http://blog.outsider.ne.kr/729>
- 자바지기의 스프링 워크북 : <http://www.javajigi.net>
- 메이븐 의존성 라이브러리 추가 그룹id, artifactId검색 <http://mvnrepository.com/>



JSP에서는 DAO에 의존해 컨트롤러와 View로 나뉘었다. 모델, 모듈간에는 의존관계가 있다. 스프링은 주로 Context에 의존해 구성된다. Context가 있어서 하위 의존들이 실행될 수 있다.

❖ 스프링 프레임워크 특징



2. Spring 실행전 환경설정

1) 다운로드 : 4는 에러충돌이 심해서 3을 사용할 예정

- 3버전 : <https://github.com/spring-attic/toolsuite-distribution/wiki/Spring-Tool-Suite-3>
- 3버전 최신 LTS : https://download.springsource.com/release/STS/3.9.18.RELEASE/dist/e4.21/spring-tool-suite-3.9.18.RELEASE-e4.21.0-win32-x86_64.zip

1-1) 만약 현장에서 자바8일경우

- 교안 : Spring STS 3버전 설치하기.txt 참고

2) sts-bundle 디렉토리 생성됨 > sts-bundle-3.9.18 로 이름변경해서 관리

3) 기존 DB 충돌 제거

- 중지 : 윈도우 찾기창에서 마우스 우클릭 > 작업관리자 > 서비스 > PID 를 통해 포트번호 확인가능 > MariaDB 서비스열기 > MariaDB 더블클릭, 중지, 시작유형-수동
- 삭제 : 설정 > 앱 > 검색 : MariaDB > 삭제

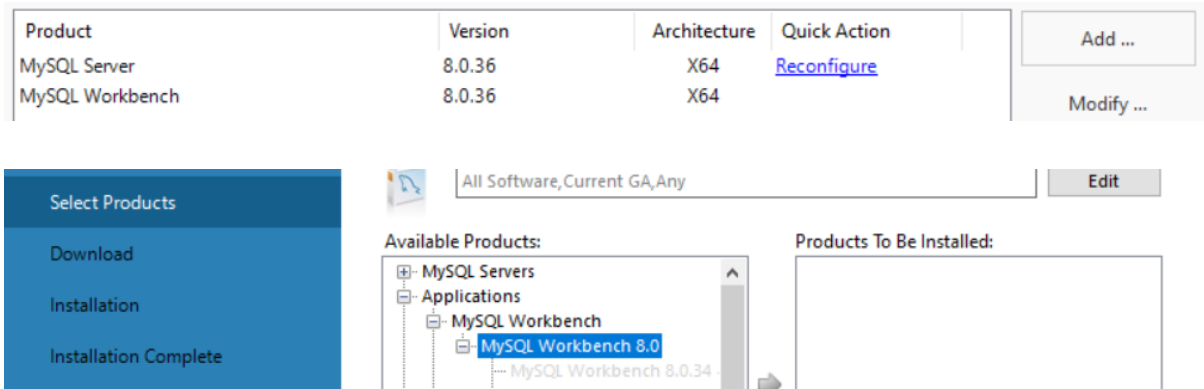
4) 바탕화면 아이콘 : C:\DeveloperTools\sts-bundle-3.9.18\sts-3.9.18.RELEASE > STS.exe 을 바탕화면 아이콘으로 생성, STS-3.9.18 로 이름 변경

3. MySQL 다운로드

- MySQL 홈 > 다운로드 > [MySQL Community \(GPL\) Downloads » MySQL Community Server](#) > 8.0.36
- 최종 다운 경로 : [Go to Download Page : Windows \(x86, 32-bit\), MSI Installer > 285m](#)

1) 기본 설정 그대로 pw 1234 , next 반복

2) mysql-installer-community-8.0.36.0 (다운했던 설치 파일) > Add > workbench도 설치



4. Spring 실행 이후 환경설정 (STS-3.9.18)

1) 실행 디렉토리 지정 : D:\SpringStudy > 기본 파일 자동 설치됨

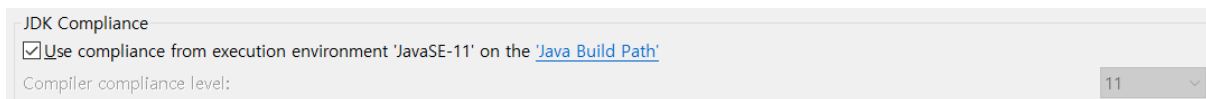
2) Preferences 설정 : 교안 1~3page 참고

3) 기본 학습을 위한 Java Project를 생성

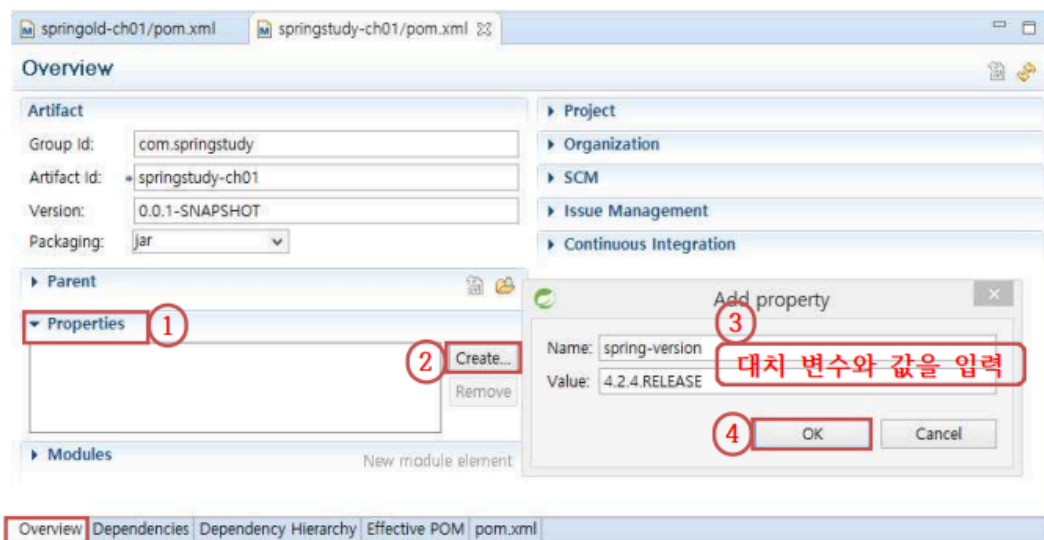
- Package Explorer > New > Maven Project > 위쪽 2개 checkbox 체크 > com.springstudy , springclass-ch01 //예전에는 org.springframework였다고 한다.

3-1) Build Path & Compiler 설정 : java11로 통일

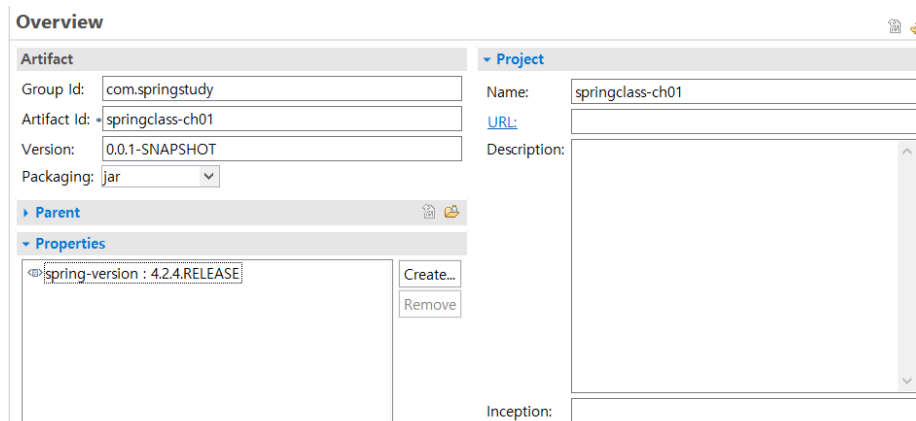
- springclass-ch01 우클릭 > Build Path > Configure Build Path...
- Java Build Path > Libraries > JRE System Library 클릭 > edit > Execution environment > java 11
 - Maven Project는 기본적으로 자바 1.5 버전 으로 생성되기 때문에 우리가 사용할 자바 버전으로 수정을 위해 11 로 통일
- Java Compiler > 11 로 통일



4) Properties 설정 : spring-version : 4.2.4.RELEASE



Maven pom.xml > overview > Properties 설정

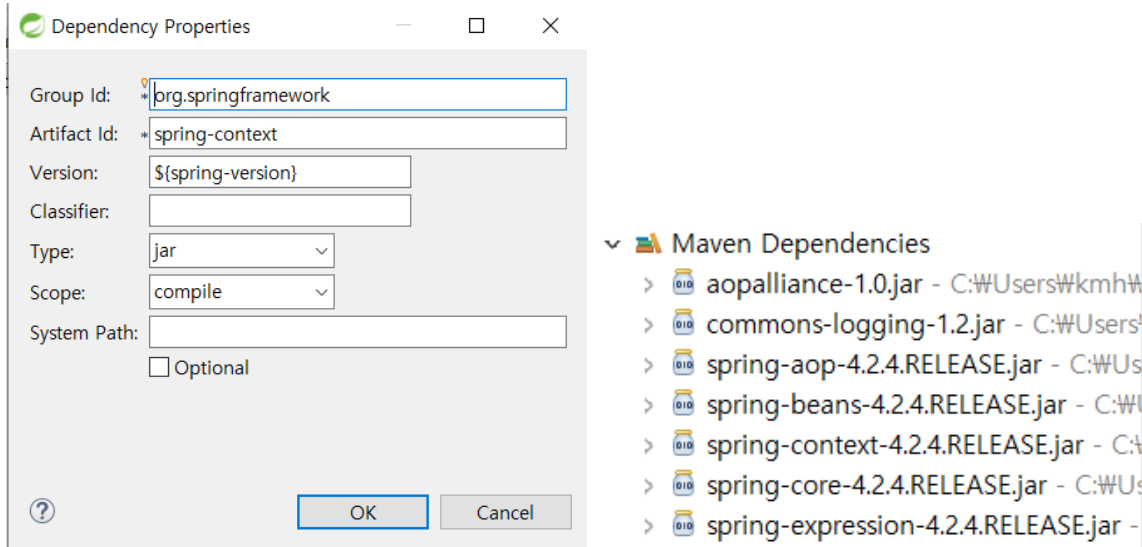


완료된 모습

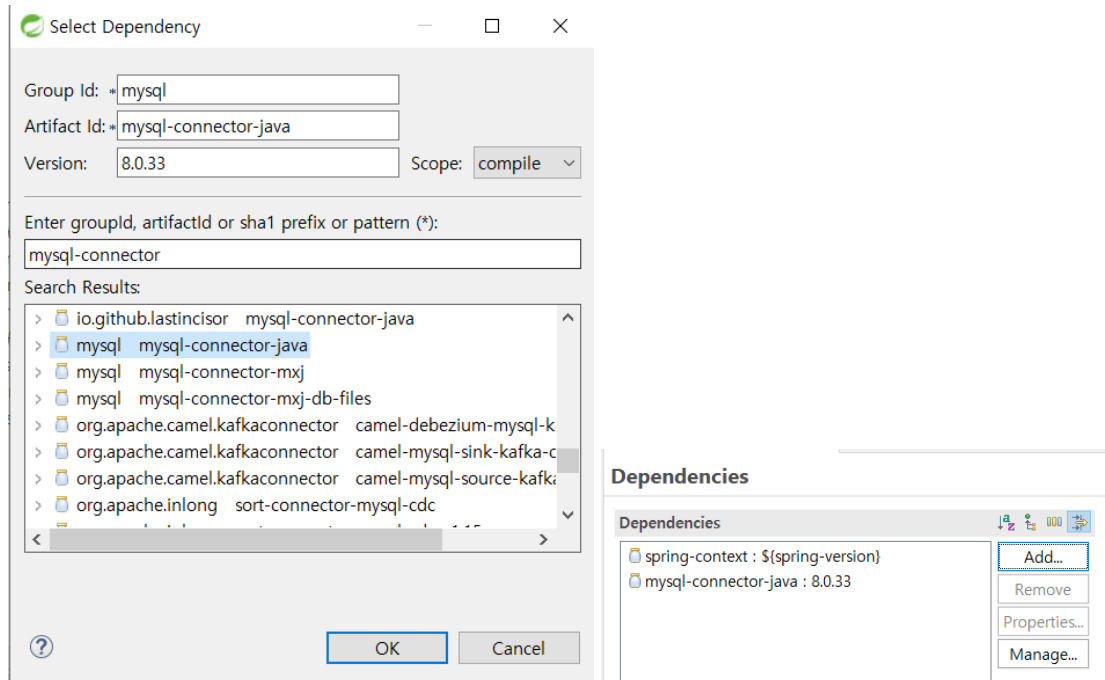
5) Dependencies 설정 : 프로젝트에서 사용하는 라이브러리 의존성을 설정하는 화면 9~11p

애플리케이션이 실행되면서 참조하는(의존하는) 라이브러리를 이 메뉴에서 설정하면 **Maven**이 알아서 메이븐 중앙저장소(Maven Central Repository, <http://www.maven.org>)로부터 다운로드 받아 **BuildPath**에 추가해 준다.

- Dependencies > add > Group Id , Artifact Id , Version 을 좌측 이미지와 동일하게 작성
- spring-context: \${spring-version} 를 Properties 로 다시 열어서 type 를 jar 로 설정 필요
- 완료시 에러는 사라지고 우측이미지처럼 Maven Dependencies 가 확인된다.
 - \${ } 지정없이 특정 버전을 지정할 수도 있지만 만약 애플리케이션에서 참조하는 스프링 버전을 변경해야 한다면 모듈별로 지정된 버전을 변경해야 하므로 불편



- mysql-connector-java 추가 / 추가 이후 완료된 화면



6) Dependencies Hierarchy : 프로젝트에서 참조하는 모듈들의 계층 구조를 한 눈에 파악하는 화면

7) Effective POM : 기본적으로 생성되는 기본 설정 파일

5. 강한 결합 약한 결합 14~16p

```
public class ProductStrongIndex {
    public static void main(String[] args) {
        ProductService service = new ProductService();
        ArrayList<Product> pList = service.getProductList();
        for(Product p : pList){ System.out.println(p); }}
```

ProductService 의 이름을 바꿀 경우, 해당 클래스의 인스턴스의 수정이 불필요
이렇게 클래스들 간의 강한 결합을 느슨한 결합으로 만들어 클래스들 간의 의존성을 해결해야 한다.

6. 인터페이스를 이용한 객체 간 결합 낮추기 17

```
- com.springstudy.ch01.interfaces
public interface ProductService {
    public ArrayList getProductList();
}
```

```
- com.springstudy.ch01.interfaces
//ProductService를 상속 받은 ProductServiceImpl01
public class ProductServiceImpl01 implements ProductService {
    // 상속받은 함수는 그대로 사용되되, 추가 선언이 이루어짐
    public ArrayList getProductList() {
        ProductDAO dao = new ProductDAOImpl();
        System.out.println("impl-service : ProductServiceImpl01.getProductList()");
        return dao.getProductList();
    }
}
```

인터페이스를 이용해 상속과 다형성을 구현하고 특정 객체에 대한 의존성을 낮춰 객체 사용의 대한 유연성을 높였다. 그러나 여전히 ProductServiceImpl01 나 ProductServiceImpl02 클래스가 변경 된다면 ProductImplIndex 클래스의 코드 수정이 불가피 하다.

7. 인터페이스와 팩토리 클래스를 이용한 객체 간 결합 낮추기 20p

ProductFactory 클래스의 단 한 줄만 수정하면 다른 부분은 수정하지 않아도 애플리케이션은 원하는 기능을 제대로 수행 할 수 있게 된다.

애플리케이션이 필요한 객체를 외부의 어떤 주체로부터 주입 받음으로써 의존하는 객체가 변경되더라도 애플리케이션의 수정을 최소화 할 수 있게 되는데 이렇게 외부로부터 필요한 객체를 주입 받는 것을 DI(Dependency Injection, 의존객체 주입)라고 한다.

요약 : 객체간 의존성이 강한 것이 제어 순행, 최소한의 수정이 가능하거나 의존성이 약한 것이 제어역행, DI

1. Spring DI(Dependency Injection) 25p

1) Inversion of Control(IoC)

2) Dependency Injection(DI)

스프링에서 의존성 주입을 설정하는 방법

- 첫 번째는 스프링 Bean 설정 파일인 XML 파일을 이용하여 의존성을 설정 하는 방법
- 두 번째는 스프링이 제공하는 Annotation을 사용해 Bean으로 설정하는 방법
- 세 번째는 자바 클래스를 스프링 빈 설정 클래스로 정의하여 의존성을 주입하는 방법

3) 스프링 DI의 종류

DL은 스프링에게 Bean을 요청해 필요한 객체를 받아오는 것을 의미 DI는 스프링으로부터 자동으로 객체가 주입되는 것을 의미 한다. 우리의 애플리케이션에서 필요한 객체를 스프링으로부터 주입(DI) 받기 위해서는 특별한 설정이 필요한데 바로 이 방법이 앞에서 설명한 XML설정과 Annotation 설정 그리고 자바 클래스를 스프링 빈으로 설정하는 3가지 방법을 말한 것이다.

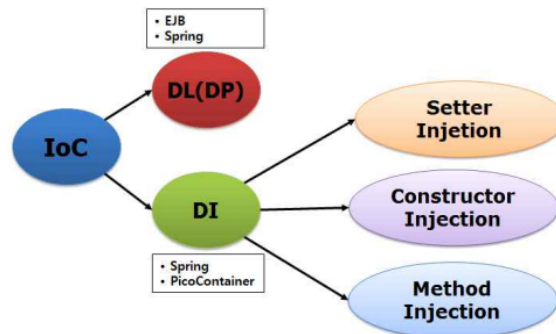


그림 2-3 스프링 DI의 종류

4) 스프링 주요 모듈과 스키마 27p (생략)

5) 스프링 DI(IoC) 컨테이너

5-1) 스프링 DI(Dependency Injection) 컨테이너

스프링 애플리케이션에서는 스프링 DI 컨테이너에서 객체(bean)가 생성되고 관리된다. 스프링 DI 컨테이너는 스프링 프레임워크의 핵심 기능 중 하나로 의존 객체 주입(또는 종속객체 주입이라 함)을 통해 객체를 생성하고 관리하며 객체간의 관계를 맺어 준다. 스프링 프레임워크에는 여러 개의 DI 컨테이너 구현체가 존재하며 크게 두 부류로 나눌 수 있다.

5-2) 빈 팩토리(Bean Factory)

`org.springframework.bean.factory.BeanFactory` 인터페이스로 DI에 대한 가장 기본적인 기능을 제공하는 스프링프레임워크의 가장 단순한 DI 컨테이너 이다.

5-3) 애플리케이션 컨텍스트(Application Context)

`org.springframework.context.ApplicationContext` 인터페이스는 `BeanFactory` 인터페이스의 자손 으로 `ApplicationContext`는 bean을 생성하고 관리하는 스프링 DI 컨테이너 이다. 이 `ApplicationContext`를 상속한 다양한 애플리케이션 컨텍스트 구현체가 존재하며 아래는 많이 사 용되는 `ApplicationContext` 인터페이스의 구현체 이다.

- `ClassPathXmlApplicationContext`, `GenericXmlApplicationContext` 를 수동으로 주로 사용

정리 필요 **8page**는 생략

2. XML 설정을 이용한 DI

어노테이션으로 줄 수 없는 경우는 xml 로 bean을 설정해 해결할 수 있다.

1) 생성자 주입(Constructor Injection) : 3개의 주입중 하나

<https://pamyferret.tistory.com/33>

```
public class MemberDAOImpl implements MemberDAO {
```

```
    private Connection conn;
    private PreparedStatement pstmt;
    private ResultSet rs;
    // 스프링이 제공하는 DriverManagerDataSource 객체 타입의 멤버 선언
    // 스프링 JDBC에 있는애라 에러표시, 라이브러리 의존 설정 필요, pom.xml에서 spring-jdbc
```

추가

```
    private DriverManagerDataSource dataSource;
    // 생성자 주입일 때
    // 스프링이 제공하는 DriverManagerDataSource 객체를 주입받는 생성자 필요
    public MemberDAOImpl(DriverManagerDataSource dataSource) {
```

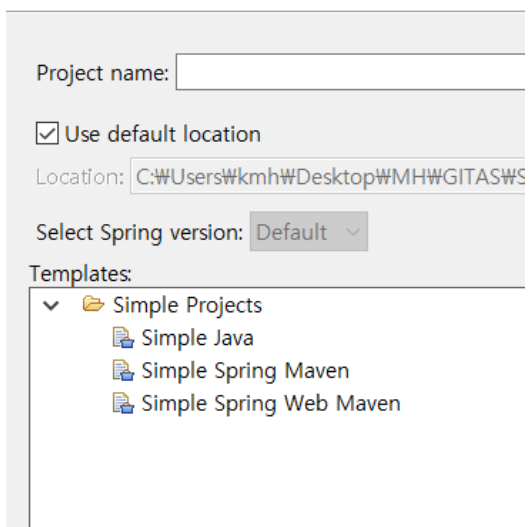
<https://spring.io/tools>

에서의 4는 boot로 유도하기에 학습을 위한 레거시 사용에 필요한 plugin이 오류가 많이 생긴다.

그래서 아래쪽의 3을 다운함 **Spring Tool Suite 3 wiki**

Spring Legacy Project

① Enter a project name.

A screenshot of the Spring Legacy Project wizard. It shows a 'Project name:' text box, a checked 'Use default location' checkbox, a 'Location:' text box with the path 'C:\Users\Wkmh\Desktop\MHWGITAS\...', a 'Select Spring version:' dropdown menu set to 'Default', and a 'Templates:' section with a tree view showing 'Simple Projects' expanded, containing 'Simple Java', 'Simple Spring Maven', and 'Simple Spring Web Maven'.

레거시 프로젝트 생성시 원하는 템플릿이 안뜸

1. Spring MVC 게시판 CRUD 구현을 위한 초기 설정

DB를 보다 쉽고 간편하게 연동할 수 있도록 지원하는 퍼시스턴스 (Persistence) 프레임워크인 MyBatis를 사용해 DB와 연동하는 게시판을 구현하고자 한다.

퍼시스턴스 프레임워크 : SQL Mapper 와 ORM 두가지로 구분

- SQL Mapper : SQL 쿼리와 자바 객체(Object)를 매핑해주는 프레임워크
 - MyBatis : 개발자가 직접 SQL을 작성해줘야 SQL과 자바 객체를 매핑할 수 있다.
 - Spring 프로젝트에서 제공하는 JDBC Template
- ORM(Object Relational Mapping) : DB 데이터와 객체 지향 프로그래밍 언어 간의 호환되지 않는 데이터를 변환하는 프로그래밍 기법을 말하며 자바 진영의 ORM은 JPA(Java Persistence API, 자바 ORM 기술에 대한 API 표준 명세로 ORM을 사용하기 위한 인터페이스의 묶음)를 기준으로 구현되어 있으며 DB 데이터를 자바 객체로 맵핑하여 객체 간의 관계를 바탕으로 SQL을 자동으로 생성해 주는 프레임워크
 - Hibernate
 - EclipseLink 등

MyBatis의 가장 큰 특징은 SQL 쿼리를 프로그램 소스에서 분리해 별도의 Mapper XML 파일이나 Mapper 인터페이스를 작성해 관리한다는 점이다. 국내에서 기업용 애플리케이션을 구축할 때 스프링프레임워크와 MyBatis를 연동한 웹 애플리케이션 구현 기법이 많이 사용된다. 또한 전자정부프레임워크도 스프링프레임워크와 MyBatis 기반으로 설계되어 있다.

1) 프로젝트 생성 및 환경설정

1-2) 의존 라이브러리와 BuildPath 설정 1p

springclass-bbs01 우클릭 > Build Path > Configure BuildPath

- Java Build Path > Libraries > JRE System Library [javaSE-11] <- Edit
- Java Compiler > JDK Compliance > 11
- Project Facets > Dynamic Web Module 3.1 & Java 11 > Runtimes > Apache
- Web Project Settings > Context root : springclass-bbs01

pom.xml

- Overview & Dependencies 설정

The screenshot displays the Eclipse IDE's 'Dependencies' view. On the left, the 'Artifact' tab shows the project configuration for 'springclass-bbs01'. Below it, the 'Properties' tab lists system properties. The main 'Dependencies' view on the right lists the following dependencies:

- spring-context : \${org.springframework-version}
- spring-webmvc : \${org.springframework-version}
- aspectjrt : \${org.aspectj-version}
- slf4j-api : \${org.slf4j-version}
- jcl-over-slf4j : \${org.slf4j-version} [runtime]
- slf4j-log4j12 : \${org.slf4j-version} [runtime]
- log4j : 1.2.15 [runtime]
- javax.inject : 1
- jstl : 1.2
- junit : 4.7 [test]
- mybatis : 3.5.14
- mybatis-spring : 1.3.3
- mysql-connector-java : 8.0.33
- commons-dbcp2 : 2.11.0
- spring-jdbc : \${org.springframework-version}

1-3) DB TABLE 생성 및 데이터 입력 2p

springstudy-bbs01 프로젝트의 src/main/resources/SQL/springbbs.sql 파일을 이용해 MySQL에 테이블을 생성하고 데이터를 추가하자.

MySQL Workbench를 열어 springbbs.sql 을 드래그&드롭

```
1] CREATE DATABASE IF NOT EXISTS spring; //ctrl+Enter
2] use spring; //ctrl+Enter
3] CREATE TABLE IF NOT EXISTS springbbs( 부터
SELECT * FROM springbbs ORDER BY no DESC; 까지 //ctrl+shift+Enter
```

1-4) 웹 애플리케이션 배포 서술자(Deployment Descriptor) 2p~10p

- src/main/webapp/WEB-INF/web.xml

1-2) 이미지 참고, 교안은 주석을 읽어 볼 것

1-5) Spring MVC Bean 정의

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

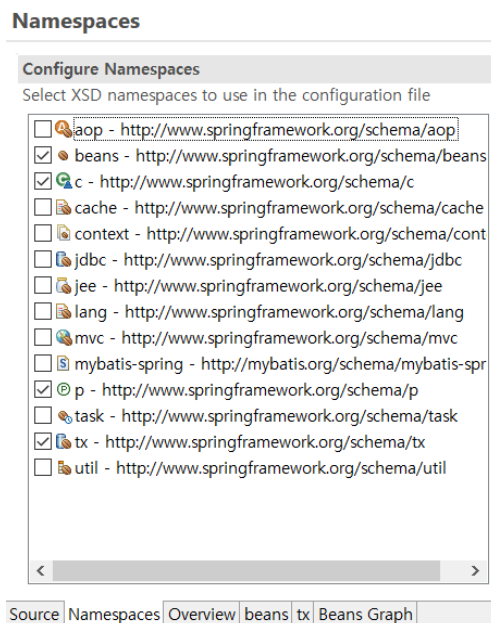
```
<annotation-driven /> //어노테이션 활성화
//ContextRoot/resources/ 로 들어오는 정적 리소스 요청에 대한 경로 매핑
<resources mapping="/resources/**" location="/resources/" />
<context:component-scan base-package="com.springstudy.bbs" /> //b.p 기준으로 자동 bean 생성
```

//id="viewResolver" 로 요청에 접두어와 접미어가 붙여 반환

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

1-6) Spring Bean 정의

- src/main/webapp/WEB-INF/spring/root-context.xml //스프링 bean 설정 파일
네임스페이스는 beans, c, p, tx 네가지를 먼저 체크해준다.



```

//현재의 Bean 설정 파일에 src/main/resource/dbcpdatasource.xml(1-8 설정)을 포함시키는 것
<import resource="classpath:datasource/dbcpdatasource.xml" />

//Mybatis의 전반적 정보를 가진 객체로 스프링프레임워크 연동의 핵심적인 객체 설정
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
//DBCP를 사용하기 위한 DataSource를 참조, 1-8 설정 참조
    <property name="dataSource" ref="dataSource"/>
//Mybatis가 DB작업을 위한 Mapper 인식 경로 설정
    <property name="mapperLocations"
        value="classpath:repository/mappers/**/*Mapper.xml"/>
//value 경로를 간략화, src/main/resource/repository/mappers/BoardMapper.xml 에서 resultType 를
//Board만 줘도 com.springstudy.bbs.domain.Board.java로 인식되는 것
    <property name="typeAliasesPackage" value="com.springstudy.bbs.domain"/>
</bean>

//mybatis-spring 모듈에 의해 SqlSession, SqlSessionFactory 을 스프링과 연동하는 기능을 구현
<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate"
    c:sqlSessionFactory-ref="sqlSessionFactory" />

//JDBC 기반의 트랜잭션 처리를 위한 트랜잭션 매니저 정의
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
    p:dataSource-ref="dataSource" />

```

1-7) DB 접속 정보 설정 20p

- src/main/resources/datasource/datasource.properties
MySQL DB와 DBCP를 사용하기 위한 설정 (생략)

1-8) DataSource Bean 설정 21p

- src/main/resources/datasource/datasource.xml

```

//프로퍼티 대체 변수 설정자(Property Placeholder Configurer)를 설정, 외부 설정을 내부에 적용
<context:property-placeholder location="classpath:datasource/dbcpdatasource.properties" />

<bean id="dataSource" destroy-method="close"
    class="org.apache.commons.dbcp2.BasicDataSource" >
    <property name="driverClassName" value="${db.driverClassName}" />
    <property name="url" value="${db.url}" />
    <property name="username" value="${db.username}" />
    <property name="password" value="${db.password}" />
    <property name="initialSize" value="${db.initialSize}" />
    <property name="maxTotal" value="${db.maxTotal}" />
    <property name="maxIdle" value="${db.maxIdle}" />
</bean>

```

2. Spring MVC와 MyBatis를 활용한 게시판 CRUD 구현

1) 게시글 리스트 보기 요청처리를 위한 DAO class 22~23p (생략)

- com.springstudy.bbs.domain.Board

2) 게시글 CRUD 관련 SQL을 분리한 Mapper 24p

- src/main/resources/repository/mappers/BoardMapper.xml

//맵퍼의 namespace 속성은 맵핑 구문을 그룹핑 하는 역할

```
<mapper namespace="com.springstudy.bbs.mapper.BoardMapper" >
```

// **resultType** : Board를 지정하고 DAO 클래스에서 SqlSession의 selectList() 메서드를 호출하면 List 객체로 받을 수 있다. (return sqlSession.selectList(NAME_SPACE + ".boardList");)

// **resultMap** : boardResultMap 로 지정시 DB의 컬럼명과 표현할 컬럼명이 다를 경우에 사용

```
<select id="boardList" resultType="Board" resultMap="boardResultMap">
```

```
    SELECT * FROM springbbs ORDER BY no DESC
```

```
</select>
```

// 맵퍼 파일 맨 아래에 resultMap을 boardResultMap으로 선언하고 위 select 태그에 resultMap을 boardResultMap으로 지정하면 모든 컬럼의 값을 읽어 올 수 있다

// regDate 를 reg_date 로 지정해주는 곳, PK는 id 로 나머지는 result 태그로 도메인 객체를 맵핑

```
<resultMap id="boardResultMap" type="Board">
    <id property="no" column="no" />
    <result property="title" column="title" />
    <result property="writer" column="writer" />
    <result property="content" column="content" />
    <result property="regDate" column="reg_date" />
    <result property="readCount" column="read_count" />
    <result property="pass" column="pass" />
    <result property="file1" column="file1" />
</resultMap></mapper>
```

3) DAO(Data Access Object) 계층 구현 25p

- com.springstudy.bbs.dao.BoardDao

```
public interface BoardDao {
```

// 현재 페이지에 해당하는 게시글 리스트를 DB에서 읽어와 반환 하는 메서드

```
    public abstract List<Board> boardList();
```

// no에 해당하는 게시글을 DB에서 읽어와 Board 객체로 반환 하는 메서드

```
    public abstract Board getBoard(int no);
```

// 게시 글쓰기 요청 시 게시 글 내용을 Board 객체로 받아 DB에 추가하는 메서드

```
    public abstract void insertBoard(Board board);
```

// 게시 글 수정, 삭제 시 no에 해당하는 비밀번호를 DB에서 읽어와 반환하는 메서드

```
    public String isPassCheck(int no, String pass);
```

// 게시 글 수정 요청 시 수정된 내용을 Board 객체로 받아 DB에 수정하는 메서드

```
    public abstract void updateBoard(Board board);
```

//no에 해당 하는 게시 글을 DB에서 삭제하는 메서드

```
    public abstract void deleteBoard(int no);}
```

4) BoardDao 인터페이스 구현 클래스 26p

게시글 리스트를 MyBatis의 SessionTemplate을 이용해 DB로부터 읽어와 반환하는 메서드를 구현
- com.springstudy.bbs.dao.BoardDaoImpl

// 이 클래스가 데이터 액세스(데이터 저장소) 계층의 컴포넌트(Bean)임을 선언

@Repository

public class BoardDaoImpl implements BoardDao {

// src/main/resources/repository/mappers/BoardMapper.xml에 정의한 Mapper namespace를 상속정의
private final String NAME_SPACE = "com.springstudy.bbs.mapper.BoardMapper";

// mybatis-spring 모듈은 MyBatis의 SqlSession 기능과 스프링 DB 지원 기능을 연동해 주는
SqlSessionTemplate 클래스를 제공한다. SqlSessionTemplate은 SqlSession을 구현해 스프링 연동
부분을 구현하였기 때문에 우리가 만드는 DAO에서 SqlSessionTemplate 객체를 사용해 SqlSession에
정의된 메서드를 사용할 수 있다. SqlSession과 SqlSessionTemplate는 같은 역할을 담당하고 있지만
트랜잭션 처리에서 다른 부분이 있다. SqlSession은 commit(), rollback() 메서드를 명시적으로 호출해
트랜잭션을 처리 하지만 SqlSessionTemplate은 스프링이 트랜잭션을 처리할 수 있도록 구현되어 있기
때문에 별도로 commit(), rollback() 메서드를 호출할 필요가 없다.

private SqlSessionTemplate sqlSession;

@Autowired

public void setSqlSession(SqlSessionTemplate sqlSession) {
 this.sqlSession = sqlSession;
}

// BoardMapper.xml과 연결되어 PreparedStatement 객체를 생성과 파라미터가 연동된다.

// SqlSessionTemplate 객체의 쿼리 결과를 읽어와 Board 객체를 생성하고 값을 설정하게 된다.

* BoardMapper.xml에서 맵핑 구문을 작성하고 아래와 같이 SqlSession 객체의 메서드를 호출하면서
맵핑 설정을 지정하게 되면 이 메서드 안에서 PreparedStatement 객체를 생성하고 PreparedStatement
객체에 필요한 파라미터가 설정된다

* SqlSessionTemplate 객체의 select(), selectOne(), selectList() * 메서드를 호출하면(내장함수들)

PreparedStatement 객체의 executeQuery() 메서드를 실행하고 쿼리를 발행한 결과인 ResultSet
객체에서 데이터를 읽어와 모델 클래스인 Board 객체를 생성하고 이 객체에 값을 설정하게 된다.

* 아래와 같이 SqlSessionTemplate의 메서드가 호출되면 repository/mappers/BoardMapper.xml 맵퍼
파일에서 mapper 태그의 namespace 속성에 지정한 com.springstudy.bbs.mapper.BoardMapper인
맵퍼가 선택되고 그 하부에 <select> 태그의 id 속성에 지정한 boardList인 맵핑 구문이 선택된다. 그리고
MyBatis 내부에서 JDBC 코드로 변환되어 실행된다.

* 매핑 구문에 resultType 속성에 Board를 지정했기 때문에 요청한 페이지에 해당하는 게시 글 리스트가
담긴 List<Board> 객체가 반환된다. Board 테이블에 게시 글 정보가 하나도 없으면 null이 반환 된다.

* 만약 SQL 파라미터를 지정해야 한다면 두 번째 인수에 필요한 파라미터를 지정하면 되는데
파라미터가 여러 개일 경우 Map 객체에 담아 두 번째 인수로 지정하면 된다.

@Override

public List<Board> boardList() {
 return sqlSession.selectList(NAME_SPACE + ".boardList");
}

5) Service 계층 구현 28p

- com.springstudy.bbs.service.BoardService

```
public interface BoardService {  
    public abstract List<Board> boardList();  
    public abstract Board getBoard(int no);  
    public abstract void insertBoard(Board board);  
    public boolean isPassCheck(int no, String pass);  
    public abstract void updateBoard(Board board);  
    public abstract void deleteBoard(int no);  
}
```

6) BoardService 인터페이스 구현 클래스 29p

게시 글 리스트를 반환하는 메서드를 구현한다.

- com.springstudy.bbs.service.BoardServiceImpl

// 이 클래스가 서비스 계층(비즈니스 로직)의 컴포넌트(Bean)임을 선언하고 있다.

```
@Service  
public class BoardServiceImpl implements BoardService {
```

```
@Autowired  
private BoardDao boardDao;
```

```
public void setBoardDao(BoardDao boardDao) {  
    this.boardDao = boardDao;  
}
```

// BoardDao를 이용해 게시판 테이블에서 현재 페이지에 해당하는 게시글 리스트를 반환 하는 메소드

```
@Override  
public List boardList() {  
    return boardDao.boardList();  
}}
```

7) Controller 클래스 29~30p ¹

게시판 관련 요청을 처리하는 BoardController 클래스를 구현

- com.springstudy.bbs.controller.BoardController

//스프링 MVC의 컨트롤러임을 선언하고 있다.

```
@Controller  
public class BoardController {  
  
    @Autowired  
    private BoardService boardService;  
    public void setBoardService(BoardService boardService) {  
        this.boardService = boardService;  
    }  
}
```

* 게시 글 리스트 보기 요청을 처리하는 메서드

¹ [\[Spring\] Controller, Service는 왜 분리해야할까?](#) : 소프트웨어 아키텍처를 유연하게 변경 가능, 단일 책임 원칙 : Controller에서는 사용자의 입력처리와 응답에만 집중하고, Service에서는 실제 기능을 어떤식으로 제공하는지에 대해서만 집중하여야 나중에 다른 팀원이 코드를 수정할 때도 어떤 기능이 어디에 있는지 쉽게 알 수 있다.

- * **@RequestMapping**은 클래스 레벨과 메서드 레벨에 지정할 수 있다.
- * **@RequestMapping**의 ()에 처리할 요청 **URI**만 지정할 때는 **value** 속성을
- * 생략하고 처리할 요청 **URI**를 **String** 또는 **String** 배열을 지정할 수 있지만
- * 다른 속성을 같이 지정할 경우 **value** 속성에 처리할 요청 **URI**를 지정해야 한다.
- * 또한 **method** 속성을 지정해 컨트롤러가 처리할 **HTTP** 요청 방식을
- * 지정할 수 있는데 아래는 **"/boardList"**, **"/list"**로 들어오는 **GET** 방식의
- * 요청을 이 메서드가 처리할 수 있도록 설정한 것이다.
- * **method** 속성을 생략하면 **GET** 방식과 **POST** 방식 모두를 처리할 수 있다.

- * 요청을 처리한 결과를 뷰에 전달하기 위해 사용하는 것이 **Model** 객체이다.
- * 컨트롤러는 요청을 처리한 결과 데이터를 모델에 담아 뷰로 전달하고 뷰는
- * 모델로 부터 데이터를 읽어와 클라이언트로 보낼 결과 페이지를 만들게 된다.

- * 스프링은 컨트롤러에서 모델에 데이터를 담을 수 있는 다양한 방법을 제공하는데
- * 아래와 같이 파라미터에 **Model**을 지정하는 방식이 많이 사용된다.
- * **@RequestMapping** 애노테이션이 적용된 메서드의 파라미터에 **Model**
- * 을 지정하면 스프링이 이 메서드를 호출하면서 **Model** 타입의 객체를 넘겨준다.
- * 우리는 **Model**을 받아 이 객체에 결과 데이터를 담기만 하면 뷰로 전달된다.

```
@RequestMapping(value= {"/boardList", "/list"}, method=RequestMethod.GET)
public String boardList(Model model) {
    // Service 클래스를 이용해 게시 글 리스트를 가져온다.
    List<Board> bList = boardService.boardList();
    // 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다. 모델에는 도메인 객체나 비즈니스 로직을
    // 처리한 결과를 저장한다.
    model.addAttribute("bList", bList);
    // servlet-context.xml에 설정한 ViewResolver에서 prefix와 suffix에 * 지정한 정보를 제외한 뷰 이름을
    // 문자열로 반환하면 된다. 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동한다.
    return "boardList";
}
```

8) 웹 템플릿 View 32~36p

index.jsp, header.jsp, footer.jsp 파일과 bootstrap, css, 자바스크립트 파일 등은 완성 프로젝트인 springstudy-bbs01 프로젝트에서 복사해서 실습용 프로젝트로 가져오자.

- * 메인 템플릿 페이지
 - src/main/webapp/WEB-INF/index.jsp
- * 템플릿 헤더 페이지
 - src/main/webapp/WEB-INF/template/header.jsp
- * 템플릿 푸터 페이지
 - src/main/webapp/WEB-INF/template/footer.jsp
- ▶ 게시 글 리스트 보기 View
 - src/main/webapp/WEB-INF/views/boardList.jsp

3. 게시 글 상세보기 요청처리

1) 게시 글 관련 **SQL**을 분리한 **Mapper** 에 추가 37p

BoardMapper.xml 는 DB 맵핑 구문과 컬럼명과 프로퍼티를 일치시키는 resultMap으로 구성

- src/main/resources/repository/mappers/BoardMapper.xml

//DAO에서 no에 해당하는 게시글을 조회할 때 기본형²인 no를 **selectOne()** 메서드의 두 번째 인수로 지정했기 때문에 parameterType은 생략할 수 있다.

// getBoard를 호출시 Board 객체를 받고, boardResultMap도 받아 쿼리 결과를 반환한다.

```
<select id="getBoard" resultType="Board" resultMap="boardResultMap">
    SELECT * FROM springbbs WHERE no = #{no}
</select>
```

2) **BoardDao** 인터페이스 구현 클래스 37p

BoardDaoImpl 클래스에 no에 해당하는 게시글을 DB로부터 읽어와 Board 객체로 반환하는 메서드

- com.springstudy.bbs.dao.BoardDaoImpl

// getBoard 맵핑 구문을 호출하면서 게시글 번호인 no를 파라미터로 지정했다.

@Override

```
public Board getBoard(int no) {
    return sqlSession.selectOne(NAME_SPACE + ".getBoard", no);
}
```

3) **BoardService** 인터페이스 구현 클래스

BoardServiceImpl 클래스에 BoardDao를 이용해 no에 해당하는 게시 글 상세 내용을 반환하는 메서드

- com.springstudy.bbs.service.BoardServiceImpl

// BoardDao를 이용해 게시판 테이블에서 no에 해당하는 게시 글을 읽어와 반환하는 메서드

@Override

```
public Board getBoard(int no) {
    return boardDao.getBoard(no);
}
```

4) **Controller** 클래스 38p

게시 글 상세보기 요청을 처리하는 아래 메서드를 추가

- com.springstudy.bbs.controller.BoardController

// 요청 파라미터 이름이 같은 경우 @RequestParam 을 지정하지 않아도 스프링으로부터 요청 파라미터를 받을 수 있다.

@RequestMapping("/boardDetail")

```
public String boardDetail(Model model, int no) {
```

// Service 클래스를 이용해 no에 해당하는 게시 글을 가져온다.

```
    Board board = boardService.getBoard(no);
```

// 파라미터로 받은 모델 객체에 뷰로 보낼 모델을 저장한다.

```
    model.addAttribute("board", board);
```

// 아래와 같이 뷰 이름을 반환하면 포워드 되어 제어가 뷰 페이지로 이동

```
    return "boardDetail";
```

```
}
```

5) 게시 글 상세보기 **View 39p** (생략)

- src/main/webapp/WEB-INF/views/boardDetail.jsp

4. 게시글쓰기 폼 요청처리

² 기본형(primitive type) : 계산을 위해 실제 값을 저장, 논리형 (boolean), 문자형 (char), 정수형 (byte, short, int, long) 실수형 (float, double)

- [1] 게시글을 작성품을 보여주는 요청 - 품을 화면에 보여주지만, **Service** 와 **DAO**의 클래스 구현불필요
- [2] 작성한 게시글을 저장요청을 처리

이렇게 클라이언트의 요청이 특별한 처리 없이 단순히 뷰만 보여줘야 하는 경우 **Spring MVC** 설정 파일인 **servlet-context.xml**에 뷰 전용 컨트롤러를 정의해 뷰 페이지를 지정하면 **Controller**에서도 이 요청을 처리하는 메서드를 별도로 구현하지 않아도 된다.

앞에서 **servlet-context.xml**에 게시 글쓰기 요청에 대한 뷰 전용 컨트롤러를 아래와 같이 정의하였기 때문에 **writeForm**과 **boardWrite**로 들어오는 **GET** 방식 요청에 대한 별도의 처리는 필요 없고 아래에서 **view-name**에 지정한 뷰 페이지만 작성하면 된다. 이때 뷰 페이지의 위치는 **Spring MVC** 설정 파일인 **servlet-context.xml**에 **ViewResolver** 클래스를 정의할 때 **prefix**에 지정한 위치(**WEB-INF/views**)를 참고하고 확장자는 **ViewResolver**의 **suffix**에 지정한 **.jsp**를 사용해 파일 명을 지정하면 된다. 그러므로 뷰 페이지의 위치와 파일 명은 **/WEB-INF/views/** 폴더에 **writeForm.jsp** 로 작성하면 된다.

```
<view-controller path="/writeForm" view-name="writeForm" />
<view-controller path="/boardWrite" view-name="writeForm" />
```

- [1] 게시글을 작성품을 보여주는 요청

1) 게시 글쓰기 품 **View** 41p

- src/main/webapp/WEB-INF/views/writeForm.jsp

2) **JavaScript** 42p

- src/main/webapp/resources/js/formcheck.js

- [2] 작성한 게시글을 저장요청을 처리

1) 게시글 관련 **SQL**을 분리한 **Mapper** 에 추가 43p

- src/main/resources/repository/mappers/BoardMapper.xml

// DAO 클래스의 **insertBoard(Board board)** 메서드에서 사용하는 맵핑 구문으로 **parameterType**을 **Board** 타입으로 지정했다.

// **parameterType**을 **Board** 타입으로 명시적 지정, 자바에서 **Board** 값이 들어올 것이라 인지시켜준다.

// 클래스의 프로퍼티(인스턴스 변수)를 **#{}**로 감싸서 지정하면 **MyBatis**가 알아서 처리해 준다.

// **SqlSessionTemplate**의 **insert()** 메서드의 반환 타입이 **int** 이므로 **resultType**은 생략 가능하다.

// 반환 데이터타입이 기본형(**int**)이면 **resultType** 이건 **parameterType**이건 생략이 된다는 것

// **INSERT** 작업을 하면서 생성된 키의 값을 읽어 와야 할 경우 **useGeneratedKeys="true"** 지정

// **parameterType**: 조회조건과 같이 입력받아야 할 데이터

// **resultType**: 쿼리 수행 후 결과를 보내주기 위한 데이터

// **parameterType**을 사용하는 방법

// 파라미터의 값으로 사용: **{변수명}**

// 파라미터 명으로 사용: **\${변수명}**

```
<insert id="insertBoard" parameterType="Board" useGeneratedKeys="true" keyProperty="no">
    INSERT INTO springbbs(title, writer, content, reg_date, read_count, pass)
    VALUES({title}, {writer}, {content}, SYSDATE(), {readCount}, {pass})
</insert>
```

2) **BoardDao** 인터페이스 구현 클래스 44p

게시 글쓰기 요청 시 게시 글 내용을 **Board** 객체로 받아 **DB**에 추가하는 메서드

- com.springstudy.bbs.dao.BoardDaoImpl

// **insertBoard** 맵핑 구문을 호출하면서 **Board** 객체를 파라미터로 지정했다.

@Override

```
public void insertBoard(Board board) {  
    sqlSession.insert(NAME_SPACE + ".insertBoard", board); }
```

3) BoardService 인터페이스 구현 클래스

- com.springstudy.bbs.service.BoardServiceImpl

@Override

```
public void insertBoard(Board board) {  
    boardDao.insertBoard(board); }
```

4) Controller 클래스 45p

- com.springstudy.bbs.controller.BoardController

@RequestMapping(value="/writeProcess", method=RequestMethod.POST)

```
public String insertBoard(Board board) {  
    boardService.insertBoard(board);  
    return "redirect:boardList";}
```

typeAliasesPackage

- > Alias는 Mapper가 XML파일에서 사용되어지는 자바타입에 대한 짧은 별칭
 - > Alias에 해당되는 클래스를 스캔하기 위한 패키지를 지정(예) value="com.springclass.bbs.domain"
-)
- > typeAliasesPackage 속성을 주면 이 하위 디렉터리의 클래스들은 모두 myBatis Mapper XML에서 parameter type 이나 result Type으로 사용가능

selectList & selectOne

- > List selectList(query_id) : id에 대한 select문을 실행한 후 레코드를 List로 반환
 - > List selectList(query_id, '조건') : id에 대한 select문을 실행하면서 조건(쿼리문에서 사용할 인자)를 전달
-
- > selectOne(query_id) : id에 대한 select문을 실행한 후 한 개의 레코드를 지정한 타입으로 반환
 - > selectOne(query_id, '조건') : id에 대한 select문을 실행하면서 조건(쿼리문에서 사용할 인자)를 전달

5. 게시글 수정 폼 요청처리

- [1] 게시 글을 수정폼을 보여주는 요청 - DB에 저장된 게시글 내용을 읽어와 폼에 출력 처리 필요
- [2] 수정한 게시글 저장하는 요청 처리 - 상세보기에서 수정 버튼을 클릭해 수정 폼 이동, 비번 체크

게시 글 수정 폼에 출력해야 할 데이터는 앞에서 구현한 게시 글 상세보기의 `getBoard(int no)` 메서드를 그대로 사용하면 된다. 그러므로 **Mapper** 설정 파일과 **BoardDaoImpl** 클래스에 추가할 기능은 게시 글 상세보기에서 게시 글 수정 폼 요청을 할 때 비밀번호를 체크하는 기능만 추가하면 된다.

- [1] 게시 글을 수정폼을 보여주는 요청

1) JavaScript 46p

상세보기 페이지에서 수정하기 버튼이 클릭되면 게시글 수정 폼을 요청하는 자바스크립트

- src/main/webapp/resources/js/formcheck.js

// hidden 폼을 통해 get 방식으로 요청할 수 있다.

```
$("#detailUpdate").on("click", function() {
    var pass = $("#pass").val();
    if(pass.length <= 0) {
        alert("게시 글을 수정하려면 비밀번호를 입력해주세요");
        return false;
    }
    $("#rPass").val(pass);
    $("#checkForm").attr("action", "update");
    $("#checkForm").attr("method", "post");
    $("#checkForm").submit(); });
```

2) 게시 글 관련 SQL을 분리한 Mapper 에 추가 47p

- src/main/resources/repository/mappers/BoardMapper.xml

```
<select id="isPassCheck" resultType="String">
    SELECT pass FROM springbbs WHERE no = #{no}
</select>
```

3) BoardDao 인터페이스 구현 클래스

게시 글 상세보기에서 게시 글 수정 폼 요청을 할 때 비밀번호를 체크

- com.springstudy.bbs.dao.BoardDaoImpl

```
public String isPassCheck(int no, String pass) {
    return sqlSession.selectOne(NAME_SPACE + ".isPassCheck", no); }
```

4) BoardService 인터페이스 구현 클래스

앞에서 구현한 게시 글 상세보기의 `getBoard(int no)` 메서드를 그대로 사용하면 되고 게시 글 상세보기에서 게시 글 수정 폼 요청을 할 때 비밀번호를 체크하는 메서드만 추가하면 된다.

- com.springstudy.bbs.service.BoardServiceImpl

```
public boolean isPassCheck(int no, String pass) {
    boolean result = false;
    // BoardDao를 이용해 DB에서 no에 해당하는 비밀번호를 읽어온다.
    String dbPass = boardDao.isPassCheck(no, pass);
    if(dbPass.equals(pass)) {
        result = true;
    }
    return result; }
```

5) Controller 클래스 48p

- com.springstudy.bbs.controller.BoardController

* **@RequestMapping** 어노테이션이 적용된 **Controller** 메소드의 파라미터에 **HttpServletResponse**와 **PrintWriter**를 지정했고 요청 파라미터를 받을 **no**와 **pass**도 지정했다. 이렇게 **Controller** 메소드의 파라미터에 필요한 객체나 요청 파라미터 이름과 동일한 이름의 파라미터를 지정하면 스프링이 자동으로 설정해 준다. 만약 요청 파라미터와 메서드의 파라미터 이름이 다른 경우 **Controller** 메서드의 파라미터 앞에 **@RequestParam("요청 파라미터 이름")**을 사용해 요청 파라미터의 이름을 지정하면 스프링이 데이터 형에 맞게 적절히 형 변환까지 해 준다. 형 변환을 할 수 없는 경우 스프링은 **400** 에러를 발생 시킨다. 예를 들면 **Controller** 메소드의 파라미터가 정수형 일 때 요청 파라미터의 값이 정수형으로 형 변환 할 수 없는 경우 **400** 에러를 발생 시킨다.

// 이렇게만 작성해도 spring 이 알아서 설정해주며, 형변환도 해준다. 형변환 안되면 404 에러뜸

```
@RequestMapping(value="/update")
```

```
public String updateBoard(Model model, HttpServletResponse response,
    PrintWriter out, int no, String pass) {
```

// BoardService 클래스를 이용해 게시판 테이블에서 비밀번호가 맞는지 체크한다.

```
boolean result = boardService.isPassCheck(no, pass);
if(! result) {
    response.setContentType("text/html; charset=utf-8");
    out.println("<script>");
    out.println(" alert('비밀번호가 맞지 않습니다.');"");
    out.println(" history.back();"");
    out.println("</script>")
    return null;
}
Board board = boardService.getBoard(no);
model.addAttribute("board", board);
return "updateForm";
}
```

6) 게시글 수정 폼 View 49p (생략)

- src/main/webapp/WEB-INF/views/updateForm.jsp

7) JavaScript 50p (생략)

게시 글 수정 폼에 대한 유효성 검사 자바스크립트 코드

- src/main/webapp/resources/js/formcheck.js

사용자가 게시 글 수정 폼에서 수정한 게시 글을 수정하는 요청은 먼저 기존의 게시 글 비밀번호와 일치하는지 검사한 후 비밀번호가 일치할 때만 게시 글이 수정되도록 구현할 것이다. 그리고 게시 글 수정 요청은 수정된 게시 글을 DB에서 수정한 후 게시 글 리스트로 리다이렉트 시켜야 하므로 별도의 뷰 페이지는 필요 없다.

1) 게시 글 관련 SQL을 분리한 Mapper 에 추가 51p

게시판 테이블에서 no에 해당하는 게시 글을 수정하는 맵핑 구문

- src/main/resources/repository/mappers/BoardMapper.xml

```
<update id="updateBoard" parameterType="Board">
    UPDATE springbbs SET title = #{title},
        content = #{content},
        reg_date = SYSDATE() WHERE no = #{no}
</update>
```

2) BoardDao 인터페이스 구현 클래스 52p

- com.springstudy.bbs.dao.BoardDaoImpl

@Override

```
public void updateBoard(Board board) {
    sqlSession.update(NAME_SPACE + ".updateBoard", board);
}
```

3) BoardService 인터페이스 구현 클래스

- com.springstudy.bbs.service.BoardServiceImpl

@Override

```
public void updateBoard(Board board) {
    boardDao.updateBoard(board);}
}
```

4) Controller 클래스 53p

- com.springstudy.bbs.controller.BoardController

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

```
public String updateBoard(HttpServletRequest response, PrintWriter out, Board board) {
    boolean result = boardService.isPassCheck(board.getNo(), board.getPass());
    if(! result) {
        response.setContentType("text/html; charset=utf-8");
        out.println("<script>");
        out.println(" alert('비밀번호가 맞지 않습니다.');"");
        out.println(" history.back();"");
        out.println("</script>");
        return null;
    }
    boardService.updateBoard(board);
    return "redirect:boardList";
}
```

[03-04] 회원 가입 구현

// prefix, suffix 가 자동 적용되도록 만들었기에 헤더와 푸터가 붙는다. 이를 막기 위해 포워드로 작성
return "forward:WEB-INF/views/member/overlapIdCheck.jsp";

// return false가 버블링과 전파를 막는다.

```
if(id.length == 0) {  
    alert("아이디를 입력해주세요");  
    return false;  
}
```