

Spring Framework

- Mybatis
- 1.Mybatis
- 2.Mybatis 추가하기
- 3.Mybatis Spring 연결
- 4.Mybatis XML 속성

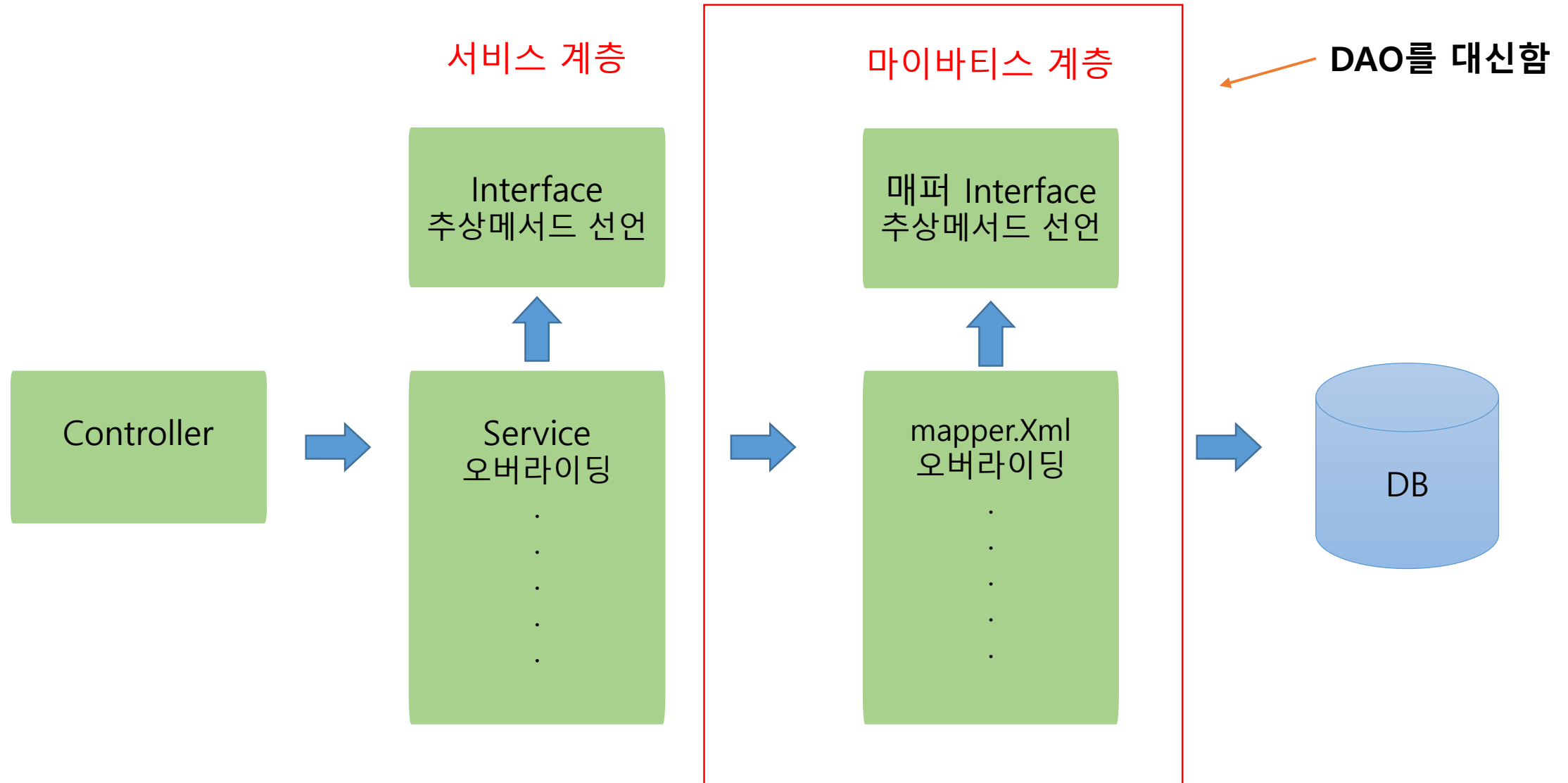
## 1. MyBatis

- MyBatis 는 개발자가 지정한 **SQL, 고급 매핑을 지원하는 프레임워크**이다.
- MyBatis 는 JDBC 코드와 수동으로 셋팅하는 파라미터와 결과 매핑을 제거합니다.
- MyBatis 는 복잡한 JDBC코드를 걷어내며 깔끔한 소스코드를 유지합니다.
- MyBatis 는 **DAO계층을 대신**합니다.
- MyBatis 는 기존 **DAO의 Interface의 구현클래스를 xml파일이 대신**합니다.
  
- 스프링에서 사용하려면 **MyBatis-Spring module을 다운로드** 받아야 한다.

### 관련 API

<http://www.mybatis.org/mybatis-3/>

## 1. MyBatis



## 1. MyBatis

전통적인 JDBC 프로그램	MyBatis
<ol style="list-style-type: none"><li>1. 직접 Connection 생성</li><li>2. 직접 Close() 처리</li><li>3. 직접 PreparedStatement 생성</li><li>4. Pstmt의 setxxx() 직접 처리</li><li>5. Select의 경우 ResultSet 처리</li></ol>	<ol style="list-style-type: none"><li>1. 자동 Connection 생성</li><li>2. 자동 Close() 처리</li><li>3. 자동 PreparedStatement 처리</li><li>4. #{name} 을 통한 ? 처리</li><li>5. 리턴 타입으로 자동 ResultSet 처리</li></ol>

## 2. MyBatis 추가하기

SQLSessionFactory

- 마이바티스 **핵심객체**
- 스프링 컨테이너에 생성해 놓으면 된다

- 마이바티스 사용시 기본적으로 **spring-jdbc 라이브러리가 있어야 함**

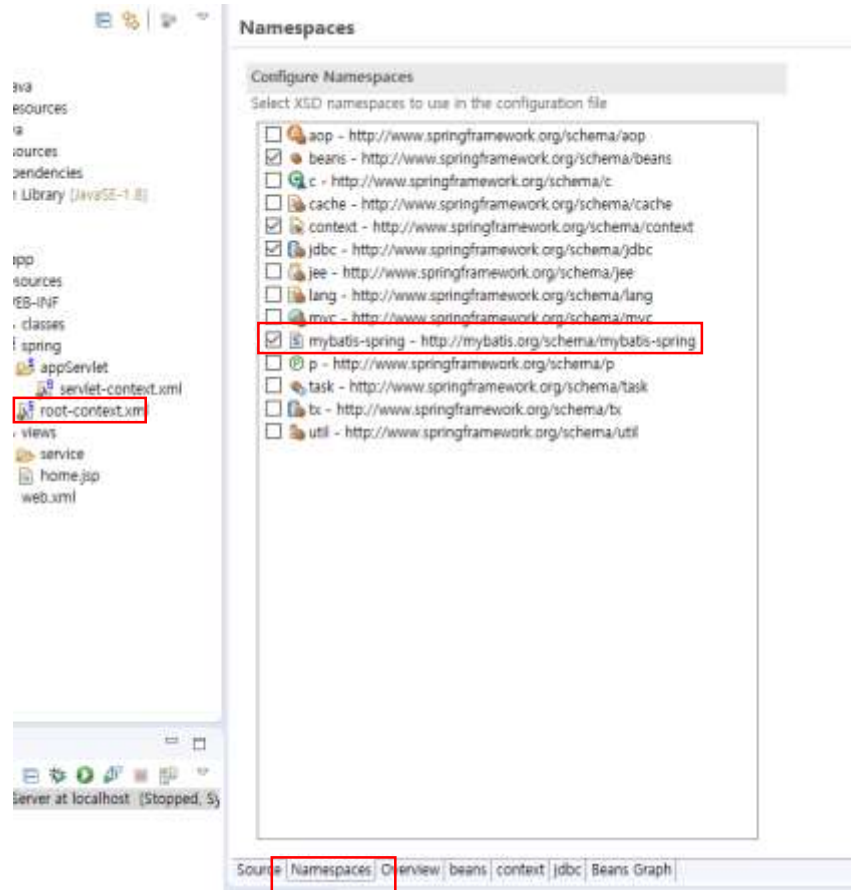
MyBatis를 사용하기 위한  
pom.xml 설정 추가

```
<!-- MyBatis 관련 라이브러리 추가 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.6</version>
</dependency>

<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.2</version>
</dependency>
```

## 2. MyBatis 추가하기

MyBatis를 사용하기 위한  
namespace 추가



MyBatis를 사용하기 위한  
Root-context.xml 설정 추가

```
<!-- MyBatis의 사용의 핵심 객체 sessionFactory 추가 -->  
<bean id="sqlSessionFactory"  
      class="org.mybatis.spring.SqlSessionFactoryBean">  
    <property name="dataSource" ref="dataSource"></property>  
</bean>
```

<mybatis-spring:scan base-package="com.zerock.testmapper"/>  
코드 해석

- > sqlSessionFactory 이름으로 컨테이너에 객체생성
- > dataSource 메서드에 앞서 생성한 dataSource를 주입시킨다.
- > ref는 참조속성 입니다.

- > mybatis-scan 추가
- > 해당 패키지를 스캔하여 base-package하위 모든 인터페이스를 마이바티스에 등록합니다.
- > @Mapper로 마이바티스 인터페이스만 지정하는 전략도 사용됩니다.

## 2. MyBatis 설정 추가하기

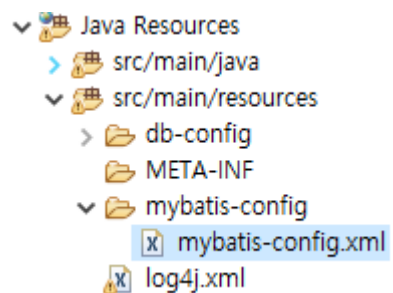
MyBatis를 사용하기 위한  
Root-context.xml 설정 추가

```
<!-- MyBatis의 사용의 핵심 객체 sessionFactory 추가 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="configLocation" value="classpath:/mybatis-config/mybatis-config.xml"/>
</bean>

<mybatis-spring:scan base-package="com.zerock.testmapper"/>
```

### 코드 해석

`configLocation`은 마이바티스 설정 속성입니다  
`classpath:/`는 `src/main/resources`파일 아래의 파일의 경로를 참조하는 방법입니다  
`classpath`아래의 파일을 맵핑시켜 마이바티스 설정을 추가할 수 있습니다.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<!-- 마이바티스 설정관련 xml파일이 됩니다 -->
<configuration>
    <typeAliases>
        <typeAlias type="com.myweb.command.FreeBoardVO" alias="FreeBoardVO"/>
    </typeAliases>
</configuration>
```

### 3. MyBatis Spring 연결

#### MyBatis와 Spring의 연결

마이바티스 계층

매퍼 Interface  
추상메서드 선언

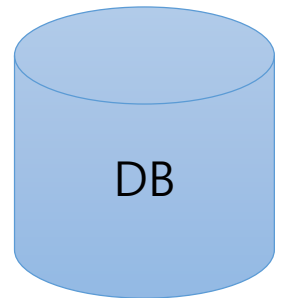


mapper.Xml  
오버라이딩

.  
. .  
. .  
. .  
. .

```
package com.zerock.testmapper;  
  
public interface TestMapper {  
  
    public String getTime();  
  
}
```

```
<?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">  
  
<!--위 설정을 추가 -->  
  
<mapper namespace="com.zerock.testmapper.TestMapper">  
  
    <select id="getTime" resultType="String">  
        select sysdate() from dual  
    </select>  
  
</mapper>
```





### 3. MyBatis Mapper 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">

<!--위 설정을 추가 -->

<mapper namespace="com.zerock.testmapper.TestMapper">
  <select id="getTime" resultType="String">
    select sysdate() from dual
  </select>
</mapper>
```



*Namespace* – 인터페이스의 경로와 동일하게 작성

*Id* – 인터페이스 추상메서드와 동일하게 작성

*resultType* – 추상메서드의 리턴타입과 동일하게 작성

### 3. MyBatis Mapper XML의 주요 속성(중요)

#### Mapper태그

*Namespace* – 인터페이스 전체경로 작성(인터페이스 동일한 이름으로 병합해서 처리 함)

#### Select 속성

*Id* – 메서드를 찾기위한 구분자(인터페이스 메서드명과 동일)

*parameterType* – 구문에 전달될 파라미터 타입 (패키지 경로 포함, 전체 클래스명)

*resultType* – 결과 반환 타입 (패키지 경로 포함, 전체 클래스명)

*resultMap* – 외부 Map타입을 이용한 반환 타입

#### Insert, Update, Delete 속성

*Id* – 메서드를 찾기위한 구분자(인터페이스 메서드명과 동일)

*parameterType* – 구문에 전달될 파라미터 타입 (패키지 경로 포함, 전체 클래스명)

#### Sql 구분의 값의 전달

*#{name}*

#### 부등호 같은 문자를 문자열로 인식시킴

```
<![CDATA[  
    코드  
]]>
```

#### 마이바티스의 두개이상 데이터 맵핑 처리

1. VO클래스 자동맵핑
2. HashMap 자동맵핑
3. @Param 이름지정을 사용해서 맵핑

#### 동적쿼리 지원

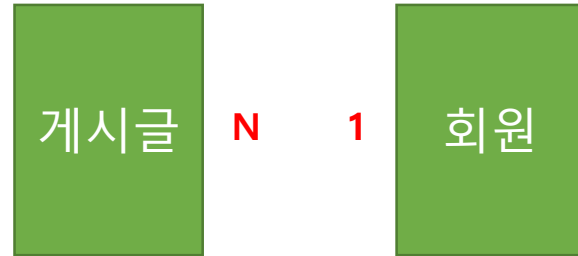
1. if
2. chose(when, otherwise)
3. foreach

## 4. 조인의 처리

다대일 조인 (N : 1)  
화면에서 글 목록을 나타내는데  
작성자의 정보도 같이 나타낸다.

N쪽에 FK가 들어간다.  
ORM작업에 알맞은 모형.

많이 사용된다.



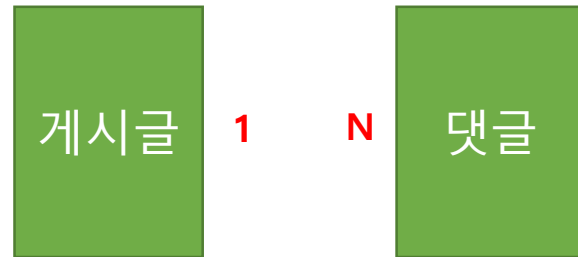
게시글 left join 회원 on 키 = 키

일대다 조인 (1 : N)  
화면에서 글 내용 1개에 여러  
댓글을 나타낸다.

ORM작업에 알맞지 않은 모형

부분적으로 사용된다.

select 2번으로 대체 될 수도 있다. 게시글 left join 회원 on 키 = 키



글	댓글
1번글	1댓글
1번글	2댓글
1번글	3댓글
1번글	4댓글

화면에서는 하나의 글만 필요한데  
조인 작업에 결과는 이렇게 나오게 됩니다

## 4.조인의 처리

```
CREATE TABLE MEMO(  
    MNO INT PRIMARY KEY AUTO_INCREMENT,  
    WRITER VARCHAR(50) NOT NULL,  
    MEMO VARCHAR(200) NOT NULL  
);
```

**N**

```
insert into MEMO(writer, memo) values('aaa', 'aaa');  
insert into MEMO(writer, memo) values('aaa', 'bbb');
```

```
CREATE TABLE USERS(  
    ID VARCHAR(50)  
    PRIMARY KEY,  
    PW VARCHAR(50) NOT NULL,  
    NAME VARCHAR(50),  
    EMAIL VARCHAR(50) UNIQUE  
);
```

**1**

```
insert into users(ID, pw, name, email) values('aaa', '1234', '홍길동', 'aaa@naver.com');
```

## 4. 조인의 처리

N:1 형태의 조인은 N의 VO에 조인에 필요한 값을 넣어주면 된다.



```
public class MemoVO {  
  
    private int mno;  
    private String writer;  
    private String memo;  
  
    //N : 1 조인에서 N쪽에 처리할 데이터를 추가  
    private String name;  
    private String email;  
  
}
```

```
select * from MEMO m      N  
left outer join USERS u    1  
on m.writer = u.id;
```

ID	PW	NAME	EMAIL	MNO	WRITER	MEMO
aaa	1234	홍길동	aaa@naver.com	2	aaa	bbb
aaa	1234	홍길동	aaa@naver.com	1	aaa	aaa

```
<select id="joinOne" resultType="MemoVO">  
    select m.*,  
           u.name,  
           u.email  
    from memo m left join users u  
    on m.writer = u.id  
</select>
```

## 4. 조인의 처리

1:N 형태의 조인은 ORM에 맞지 않기 때문에  
표현하고 싶다면 VO쪽에 N쪽을 표현한다.  
그리고 Mybatis에 ORM작업을 직접 명시한다.



```
public class UsersVO<T> {
```

```
    private String id;
    private String pw;
    private String name;
    private String email;
```

//1: N 조인에서는 N쪽을 list로 처리

```
    private List<T> memoList;
```



**resultMap:** 조인의 ORM을 직접 지정  
**Id:** resultMap의 이름  
**Type:** 리턴의결과 타입,  
**Column:** 데이터베이스의 컬럼값  
**Property:** 처리할 VO의 속성값

```
select * from USERS u
left outer join MEMO m
on m.writer = u.id;
```

	MNO	WRITER	MEMO	ID	PW	NAME	EMAIL
▶	1	aaa	aaa	aaa	1234	홍길동	aaa@naver.com
	2	aaa	bbb	aaa	1234	홍길동	aaa@naver.com

```
<select id="joinTwo" resultMap="joinResult">
    select *
    from users u left join memo m
    on u.id = m.writer
</select>
```

```
<resultMap type="UsersVO" id="joinResult">
    <result column="id" property="id"/>
    <result column="name" property="name"/>
    <result column="email" property="email"/>

    <collection property="memoList" resultMap="memoListResult" />
</resultMap>

<resultMap type="MemoVO" id="memoListResult">
    <result column="mno" property="mno"/>
    <result column="writer" property="writer"/>
    <result column="memo" property="memo"/>
</resultMap>
```