KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

# 스프링과 MySQL Database 연동

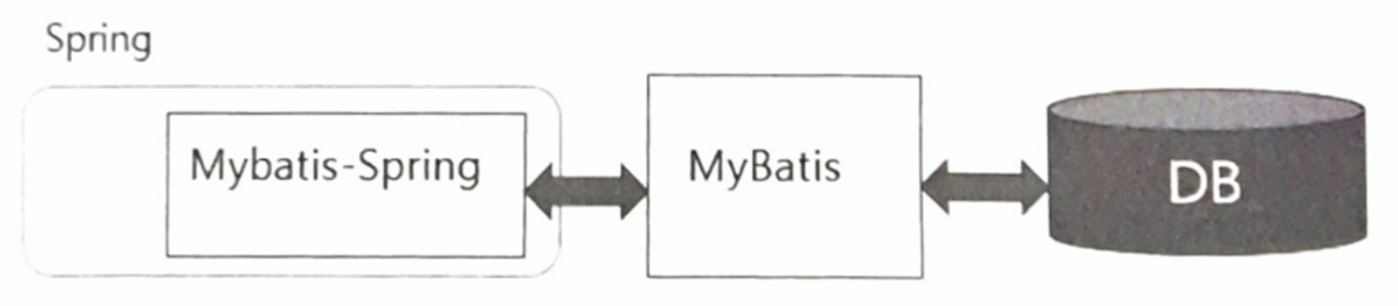
## [KB] IT's Your Life

mybatis 연동. mybatis란 자바를 위한 오픈소스 sql매퍼 프레임워크. 데이터베이스 쿼리 코드를 간결하게 작성하고 유지보수성을 높이는 데 도움을 준다. 아래는 주요 기능들.
1 jdbc 코드 간소화 : jdbc코드를 캡슐화하여 복잡한 jdbc코드를 줄이고, 불필요하게 "여러가지 상황에서 거의 또는 전혀 변경하지 않고 재사용할 수 있는" 코드를 제거한다.
2 sql쿼리와 자바코드 분리 : xml파일에 sql쿼리를 저장하여 java코드와 sql쿼리를 분리함. sql쿼리의 변경과 java코드는 무관해짐
3 동적 sql지원 : xml파일에서 if for choose foreach 등의 태그를 사용하여 동적 sql 작성 작성 가능해짐.
4 객체와 데이터베이스 매핑 : sql 쿼리 결과를 java객체로 매핑하여 데이터베이스를 객체 형태로 다룰 수 있다

KB 국민은행

## MyBatis

- https://mybatis.org/mybatis-3/
- SQL 매핑 프레임워크




스프링과 디비와 연결

## MyBatis 관련 라이브러리 추가

- spring-jdbc/spring-tx
  - 스프링에서 데이터베이스 처리와 트랜잭션 처리
- mybatis/mybatis-spring
  - MyBatis와 스프링 연동용 라이브러리

트랜젝션 처리가 필요함

mybatis를 위한 연동 라이브러리가 필요하다

## build.gradle

```
                    ...
// 데이터베이스
implementation 'com.mysql:mysql-connector-j:8.1.0'
implementation 'com.zaxxer:HikariCP:2.7.4'

implementation "org.springframework:spring-tx:${springVersion}"
implementation "org.springframework:spring-jdbc:${springVersion}"

implementation 'org.mybatis:mybatis:3.4.6'
implementation 'org.mybatis:mybatis-spring:1.3.2'
                    ...
```

트랜젝션 관리를 스프링이 하겠다!라는 의미

→ gradle sync


## SQLSessionFactory

- SQLSession 마이바티스의 중심 객체
  - Connection 생성, SQL 전달, 결과 리턴 등 처리
- SQLSessionFactory
  - SQLSession 객체 생성

sql 세션에 디비와 연결하는 conn객체 풀이 보관된다.
또한 마이바티스 관련 정보들이 안에서 구축된다.

세션은 connection과 1대1 연결된다.

sql세션은 멀티스레딩 안전하지 않다. 동기화가 필요

이를 위해 sql세션 책토리 패턴을 이용한다.

sql세션은 conn객체만큼 만들어지고 쓰레드 안전하지 않다

sql세션팩토리는 싱글톤으로 운용되고 쓰레드 안전하다!

마이 바티스 설정파일을 만들자! 이것도 외부로 BBAAM

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

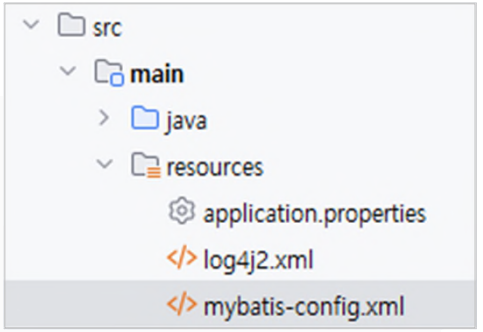




## resources/mybatis-config.xml

```
<?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">

<configuration>

</configuration>
```

분석할때 여기에 있는 규칙을 참고함




번외 xml. 태그기반인 마크업랭기쥐. x는 extensible 약자. 확장 가능하다. 다른 태그기반랭기쥐와 다르게 태그를 만들어서 사용할 수 있다. 사용자 정의 태그 사용 가능.

2가지 정류로 나뉘다. 규칙이 있는것과 없는 것.

이번건 규칙 있는 거.

!DOCTYPE태그는 규칙이 어디있느냐를 표기. 규칙이 없는 xml이면 해당 캐그 필요X

dtd와 schema는 규칙정의 형식이다 스키마는 .xml형식이다

configuratoin이 root태그.

## config.RootConfig.java

```
...
@Configuration
public class RootConfig {
        ...
        @Autowired
        ApplicationContext applicationContext;

 @Bean
 public SqlSessionFactory sqlSessionFactory() throws Exception {
   SqlSessionFactoryBean sqlSessionFactory = new SqlSessionFactoryBean();
   sqlSessionFactory.setConfigLocation(applicationContext.getResource("classpath:/mybatis-config.xml"));
   sqlSessionFactory.setDataSource(dataSource());
   return (SqlSessionFactory) sqlSessionFactory.getObject();
 }

        @Bean
        public DataSourceTransactionManager transactionManager(){
                DataSourceTransactionManager manager = new DataSourceTransactionManager(dataSource());

                return manager;
        }
}
```

DI 요청. ApplicationContext는 스프링에서 자동으로 빈에 등록되어있음!

sqlsessionfactory도 빈등록!

앞에서 등록된 컨텍스트에 있는 dataSource빈 달라. 없으면 생성메서드 호출해서 달라 라는 의미



마이바티스 관련 코드는 이게 다야. 나머지는 스프링이 알아서 다해줌

아무자바객체나 autowired할수있는게 아니야
할수있는건 빈에 등록된 객체에 한해서 가능

디비와 연결된 커넥션들은
디비커넥션풀로
히카리가 관리.

각 커넥션은 sql세션이
품어서 관리
해당 sql세션은
sql세션 팩토리가 만들고
관리함

sql세션팩토리가 sql세션을
생성할때 참고하는 설정은
아까 만든
mybatis_config.xml파일


# 1 MyBatis

## test:: RootConfigTest.java

```
class RootConfigTest {
                    ...
   @Autowired
   private SqlSessionFactory sqlSessionFactory;
                    ...
   @Test
   public void testSqlSessionFactory() {
      try (
         SqlSession session = sqlSessionFactory.openSession();
         Connection con = session.getConnection();
      ) {
         log.info(session);
         log.info(con);
      } catch (Exception e) {
         fail(e.getMessage());
      }
   }
}
```

openSession하면 sqlsession하나 나옴

그리고 그안에 겟 커넥션있음. 그리고호출한다면 이는 히카리가 관리하는 디비커넥션 풀에서 하나 꺼낸거임




sql세션 인터페이스를 보면 쿼리를 매개변수로 받고 쿼리를 실행해주는 메서드들이 많다!
where절에 들어가는 것처럼 조건이 있을경우는 해당 조건들을 캡슐화한 오브젝트 변수를 넘겨주는 메서드도 있다.
여튼 다양함; 트랜젝션 관련된 메서드도 있다!

jdbc할때 쿼리를 작성할때 힘들었던 점! string sql= ",,,"; 여러줄에 걸쳐서 만들때 별로였음. 또 물음표는 가독성 떨어져;

이런 표현을 다 xml로 빨거야. 그러면 하드코딩도 해결함. 분리되네. 그걸 우리는 매퍼라한다.
태그를 지정해서 어느 메서드를 위한 태그인지 지정도 가능





## Mapper

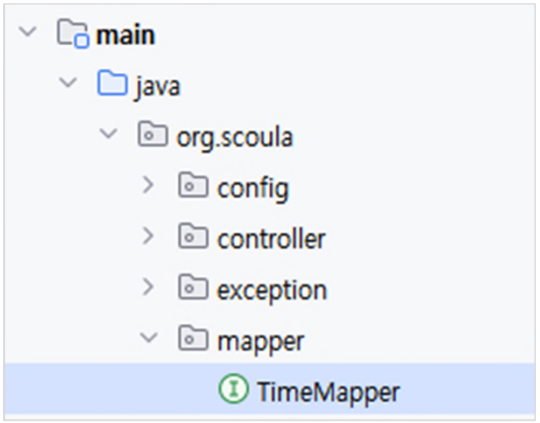
- SQL과 그에 대한 처리를 지정하는 역할
- XML과 인터페이스+어노테이션 형태로 작성

mybatis를 제대로 사용해보자

이번 예제가 mybatis사용하는 좋은 이유를 보여줌

## Mapper 인터페이스

- org.scoula.mapper 패키지
  - TimeMapper 인터페이스 추가



## RootConfig.java – Mapper 설정

```
…
import org.mybatis.spring.annotation.MapperScan;
…

@Configuration
@PropertySource({"classpath:/application.properties"})
@MapperScan(basePackages  = {"org.scoula.mapper"})
public class RootConfig {

                …

}
```

### @MapperScan

- Mapper 인터페이스를 검색할 패키지 목록 지정
- 해당 인터페이스를 빈으로 등록
- 해당 인터페이스의 구현체를 동적으로 자동 생성

마이바티스가 자동으로 구현해줌

## mapper.TimeMapper.java

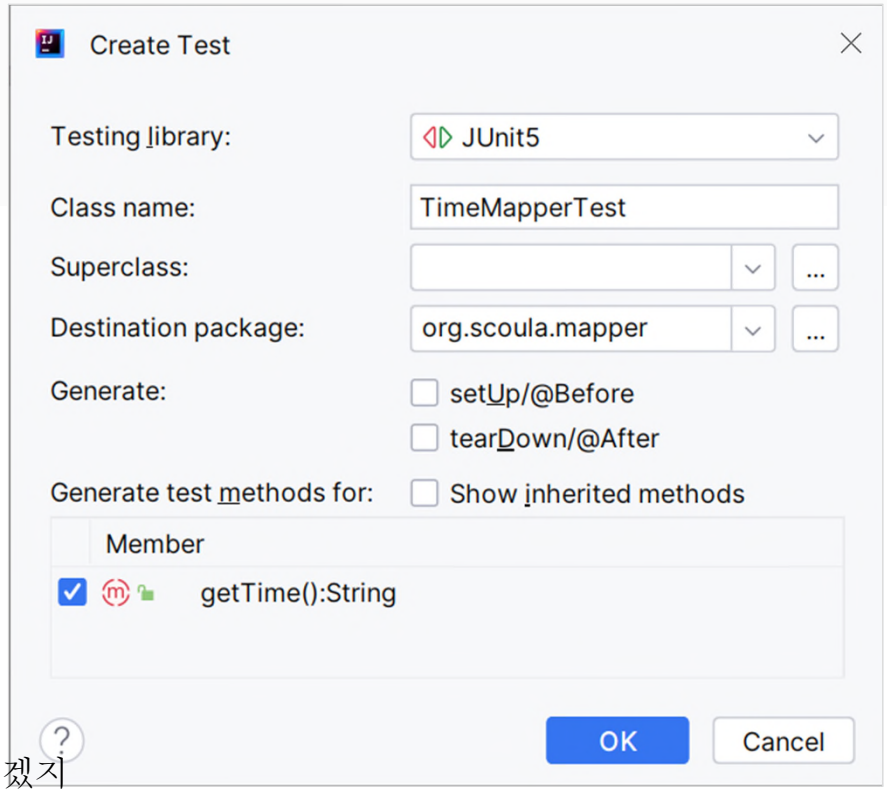
```
package org.scoula.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {
    @Select("SELECT sysdate()")
    public String getTime();

}
```

애도 빈 등록됐으니 DI후보임

메서드하나 추가

테스트하러 ㄱ

Ctrl+Shift+T




자동으로 구현된 TimeMapperProxy 를 쑤도코드로 표현하자면

```
TimeMapperProxy {
    @Autowired
    SQLSession session;

    selectOne
```

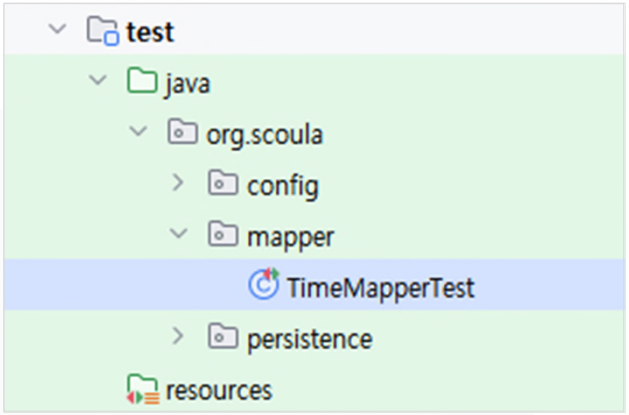
sql세션 팩토리에서 얻은 sql세션

//selectOne 메서드있음.. @Select때문에
//왜 하필 selectOne이냐
//지정 메서드 반환타입등 시그니처를 봐서
//어울리는 메서드하나 지정함.
//만약 반환이 리스트타입이면 selectList같은거겠지

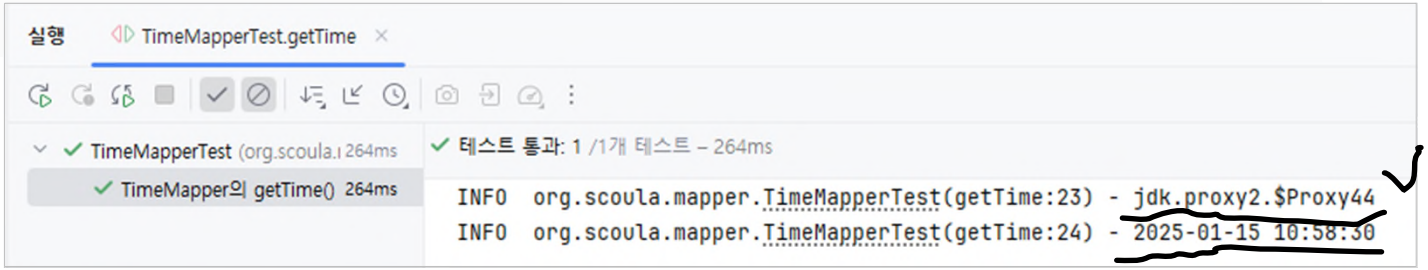
//지정된 메서드는 동작할때 필요한 첫 매개변수는
//쿼리일텐데 이는 @Select어노테이션의 속성값인 "select sysdate()"이다.

마이바티스가 인터페이스를 구자동으로 구현한고 메서드도 어노테이션정보로 구현하여 바로 사용할 수 있게해준다.

## test :: mapper.TimeMapperTest.java




```
...

@ExtendWith(SpringExtension.class)
@ContextConfiguration(classes = { RootConfig.class })
@Log4j2
public class TimeMapperTest {

        @Autowired
        private TimeMapper timeMapper;

        @Test
 @DisplayName("TimeMapper의 getTime()")
        public void getTime () {
                log.info(timeMapper.getClass().getName());
                log.info(timeMapper.getTime());
        }
}
```



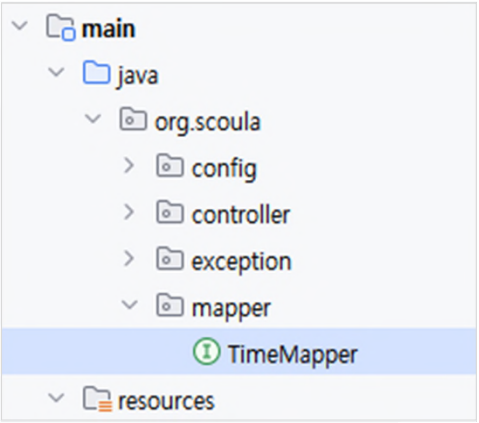

메서드
구현 안했는데
뭔가 행동하네?

## TimeMapper.java

```
package org.scoula.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {
        @Select("SELECT sysdate FROM dual")
        public String getTime();

        public String getTime2();
}
```

mybatis코드를 잘 활용했지만
이런 형태는 sql쿼리와 자바코드가
완전히 분리되지 않은 형태.
게다가 하드코딩임!

selectIOne메서드랑 매칭 되겠죠



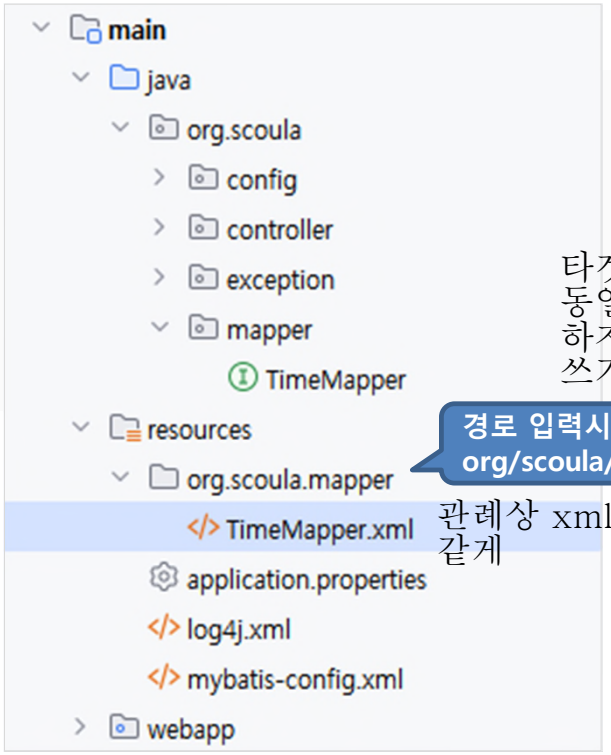

- getTime2()에 연관시킬 sql은 xml 매퍼로 정의해서 처리

## XML 매퍼와 같이 쓰기

- src/main/resources
  - org/scoula/mapper --> Mapper 인터페이스와 같은 패키지 경로여야 함
    - TimeMapper.xml --> 인터페이스명과 파일명이 같아야 함

```
<?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.scoula.mapper.TimeMapper">

</mapper>
```

TimerMapper 인터페이스 경로




```
<?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">
```

## resources:: TimeMapper.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.scoula.mapper.TimeMapper">

        <select id="getTime2" resultType="string">
                SELECT sysdate()
        </select>
</mapper>
```

String getTime2()의 메서드명 — 인터페이스의 메서드 이름이 곧 아이디가 됨.

@Select와 대응하는 select태그.

ㅏ<insert> <update>등
다양한 동작 태그도 있다/

String getTime2()의 리턴타입

리턴 타입과 쿼리ㅡ 내용을 봐서

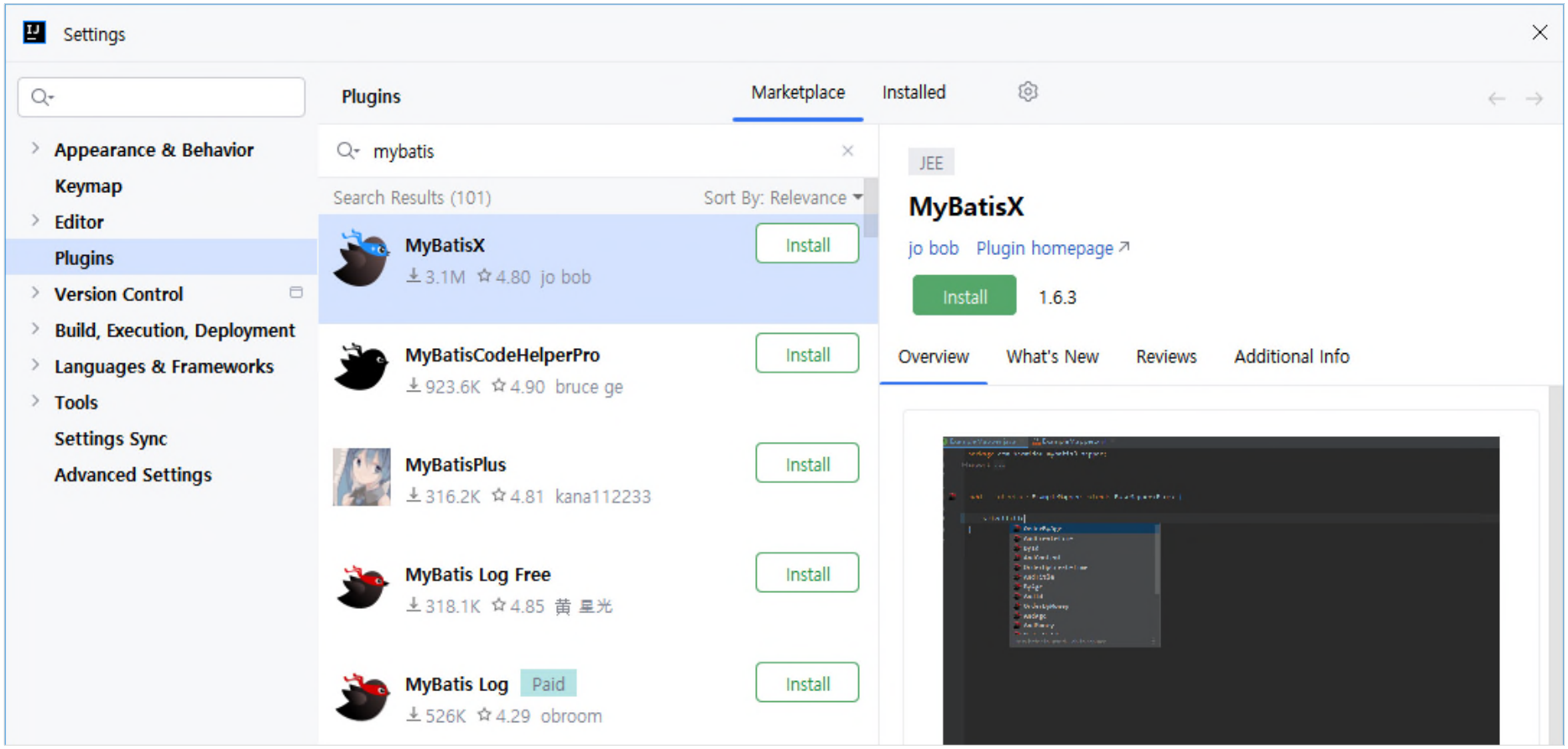
알맞은 sql세션 메서드가 호출활용됨

플러그인을 통해 해당 과정을 쉽게할 수 있다!!

왔다갔다 쉽게할 수 있게

## MyBatisX 플러그인 설치

- mybatis 검색



아까썼던 select태그 내용 지워버리고 일단

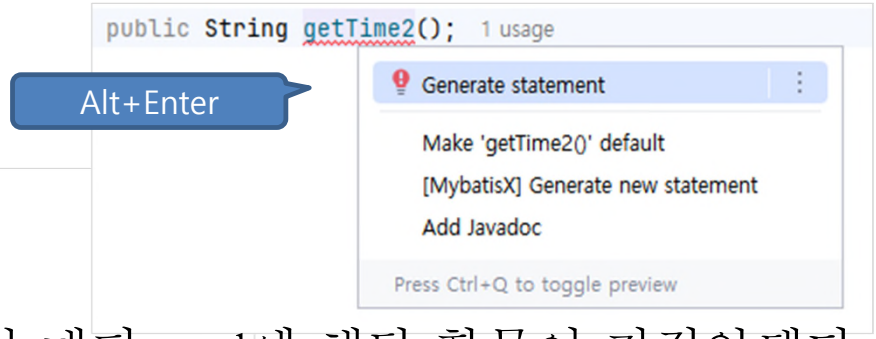




## resources:: TimeMapper.xml

```
<?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.scoula.mapper.TimeMapper">

</mapper>
```

## TimeMapper.java


자바 문법이 아닌 매퍼 xml에 해당 항목이 지정안됐다 으미.

## resources:: TimeMapper.xml

왔다갔다 쉬움

TimeMapper.java로 이동

getTime2() 선언으로 이동

```xml
<mapper namespace="org.scoula.mapper.TimeMapper">

    <select id="getTime2" resultType="java.lang.String">
        SELECT sysdate()
    </select>
</mapper>
```

## TimeMapper.java

Mapper xml의 getTime2로 이동

```java
public interface TimeMapper {  2 usages
    @Select("SELECT sysdate()")  1 usage
    public String getTime();

    public String getTime2();  1 usage
}
```

기본 골격 만들어줌

쿼리만 채워주면돼

;쓰지말고

## test::TimeMapperTest.java

```
package org.scoula.persistence;
…
public class TimeMapperTest {
        …

        @Test
 @DisplayName("TimeMapper의 getTime2()")
        public void getTime2() {

                log.info("getTime2");
                log.info(timeMapper.getTime2());

        }
}
```

## JDBC 처리 관련 로그 세부 사항 출력

- PreparedStatement로 SQL 처리시 ?에 실제 치환 값을 알아야 하는 상황이 발생 가능
- log4jdbc-log4j2 라이브러리
  - PreparedStatement로 SQL 처리시 ?에 실제 치환 값을 출력해줌

# 3 log4jdbc-log4j2 설정

## build.gradle

```
// logging
implementation 'org.slf4j:slf4j-api:2.0.9'
runtimeOnly 'org.slf4j:jcl-over-slf4j:2.0.9'
runtimeOnly 'org.slf4j:slf4j-log4j12:2.0.9'
implementation 'log4j:log4j:1.2.17'

implementation 'org.bgee.log4jdbc-log4j2:log4jdbc-log4j2-jdbc4:1.16'
implementation 'org.apache.logging.log4j:log4j-api:2.0.1'
implementation 'org.apache.logging.log4j:log4j-core:2.0.1'
```

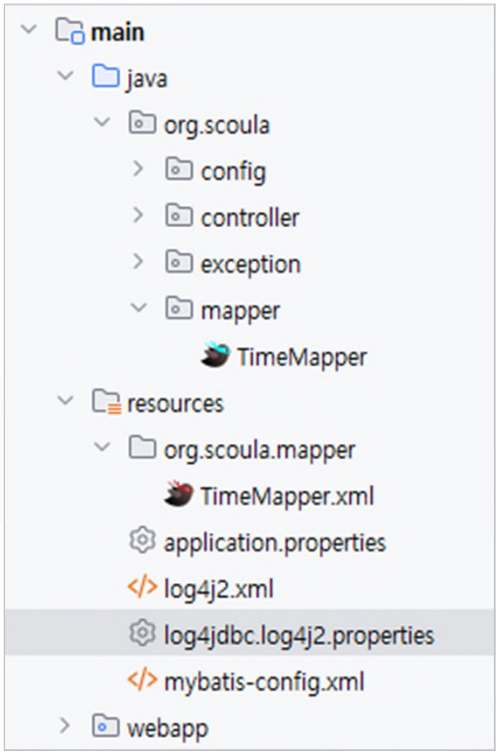
→ gradle sync

implementation 'org.bgee.log4jdbc-log4j2:log4jdbc-log4j2-jdbc4:1.16'
implementation 'org.apache.logging.log4j:log4j-api:2.0.1'
implementation 'org.apache.logging.log4j:log4j-core:2.0.1'

# 3 log4jdbc-log4j2 설정

## log4jdbc-log4j2 설정

- src/main/resources
  - log4jdbc.log4j2.properties




## resources/log4jdbc.log4j2.properties

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
log4jdbc.auto.load.popular.drivers=false
log4jdbc.drivers=com.mysql.cj.jdbc.Driver
```

설정 간단한 설명
로그메세지를 가로채서 살짝 조작해서 넘기는 녀석

log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
log4jdbc.auto.load.popular.drivers=false
log4jdbc.drivers=com.mysql.cj.jdbc.Driver

## resources/application.properties

```
#driver=com.mysql.cj.jdbc.Driver
#url=jdbc:mysql://127.0.0.1:3306/scoula_db
jdbc.driver=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
jdbc.url=jdbc:log4jdbc:mysql://localhost:3306/scoula_db
jdbc.username=scoula
jdbc.password=1234
```

# 주석처리

jdbc.driver=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
jdbc.url=jdbc:log4jdbc:mysql://localhost:3306/scoula_db

## 로그의 레벨 설정

- 너무 많은 로그 메시지 출력 → 로그 레벨 이용하여 수정 가능
- 로그레벨
  - FATAL : 가장 크리티컬한 에러가 일어 났을 때만 로깅
  - ERROR : 일반 에러가 일어 났을 때만 로깅
  - WARN : 에러는 아니지만 주의할 필요가 있을 때만 로깅
  - INFO : 일반 정보를 나타낼 때 로깅 (INFO + ERROR)
  - DEBUG : 일반 정보를 상세히 나타낼 때 로깅 (디버깅 정보)
  - TRACE : 경로추적을 위해 사용

## test :: resources/log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <!-- Appender, Layout 설정 -->
  <Appenders>
    <Console name="console" target="SYSTEM_OUT">
      <PatternLayout pattern=" %-5level %c(%M:%L) - %m%n"/>
    </Console>
  </Appenders>

  <!-- Logger 설정 -->
  <Loggers>
    <Root level="INFO">
      <AppenderRef ref="console"/>
    </Root>
    <Logger name="org.scoula" level="INFO" additivity="false" >
      <AppenderRef ref="console"/>
    </Logger>

    <Logger name="org.springframework" level="INFO" additivity="false">
      <AppenderRef ref="console"/>
    </Logger>
  </Loggers>

</Configuration>
```

## 테스트 로그 출력

마이바티스가 내보내는 로그들

```
...
INFO  jdbc.audit(methodReturned:175) - 1. Connection.prepareStatement(SELECT sysdate()) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@31a3f4de
INFO  jdbc.sqlonly(sqlOccurred:228) - SELECT sysdate()
INFO  jdbc.sqltiming(sqlTimingOccurred:373) - SELECT sysdate()
{executed in 24 msec}
INFO  jdbc.audit(methodReturned:175) - 1. PreparedStatement.execute() returned true
INFO  jdbc.resultset(methodReturned:175) - 1. ResultSet.new ResultSet returned
INFO  jdbc.audit(methodReturned:175) - 1. PreparedStatement.getResultSet() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@302edb74
INFO  jdbc.resultset(methodReturned:175) - 1. ResultSet.getMetaData() returned com.mysql.cj.jdbc.result.ResultSetMetaData@67c5ac52 - Field level information:
    com.mysql.cj.result.Field@36417a54[dbName=null,tableName=null,originalTableName=null,columnName=sysdate(),originalColumnName=null,mysqlType=12(FIELD_TYPE_DATETIME),sqlType=93,flags= BINARY, charsetIndex=63, charsetName=ISO-8859-1]
...
|--------------------|
|sysdate()           |
|--------------------|
|2025-01-15 11:19:37 |
|--------------------|
...
```

preparedstatement준비 표시

쿼리

결과 나온거 표시

결과를 테이블로 보여줌

저 4개정도만 나오게끔 하자

# 3 log4jdbc-log4j2 설정

## 로그의 레벨 설정

### log4jdbc 출력 로그 조절

- <Logger> 태그 지정

### 핵심 로그만 출력하기

- jdbc.sqlonly

→ 로그 레벨 info로 설정

테스트용 log4j2.xml

- jdbc.audit
- jdbc.connection
- jdbc.resultset 등

→ 해당 패키지의 로그 레벨을 warn으로 설정

# 3 log4jdbc-log4j2 설정

## resources/log4j2.xml

```
        ...
    <Logger name="jdbc" level="WARN" additivity="false">
      <AppenderRef ref="console"/>
    </Logger>

    <Logger name="jdbc.sqlonly" level="INFO" additivity="false">
      <AppenderRef ref="console"/>
    </Logger>

  </Loggers>

</Configuration>
```

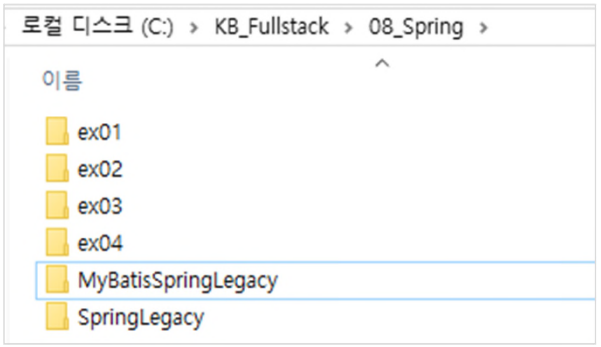
jdbc로 시작하는 로그는 warn레벨로

그중 sqlonly는 info로 레벨 낮추기

### ○ 다시 테스트 실행

## 프로젝트 템플릿 만들기

- ex04 → MyBatisSpringLegacy로 복사

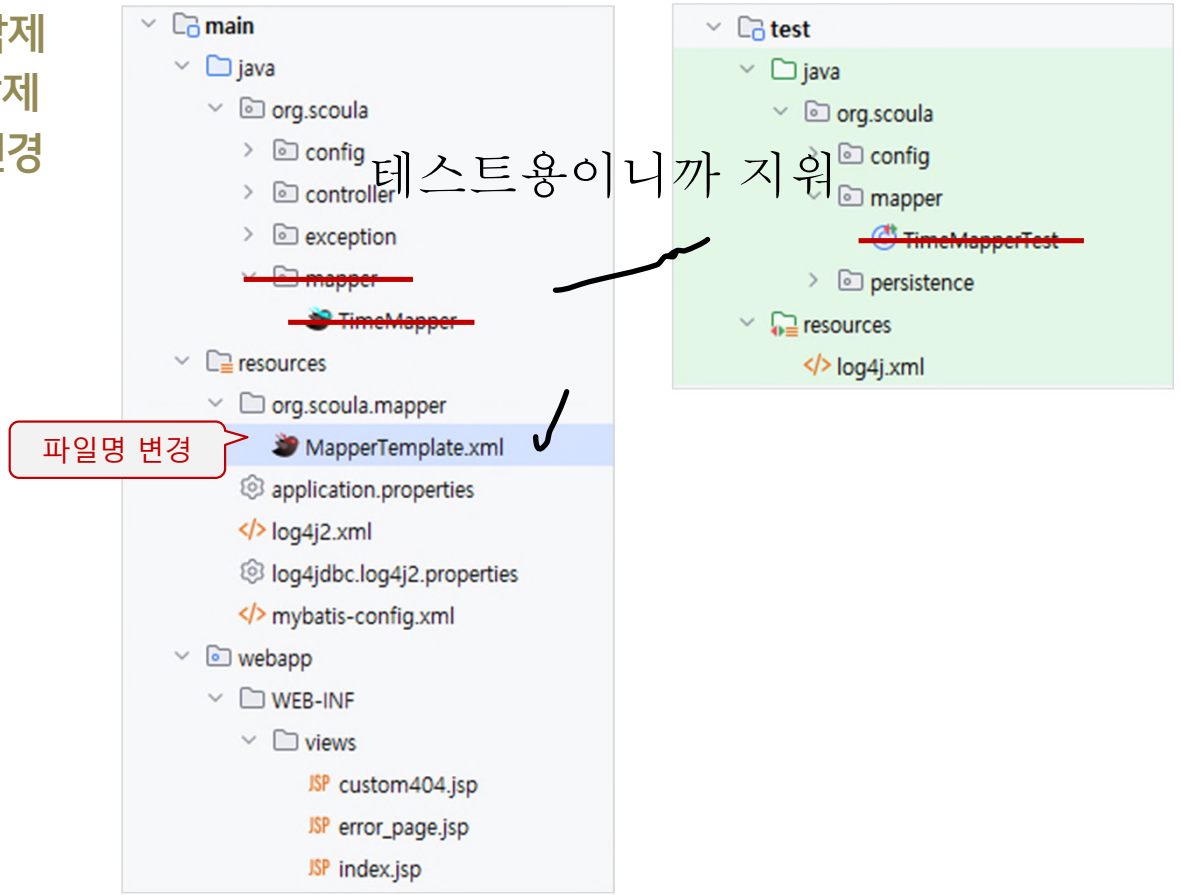


## settings.gradle

```
rootProject.name = "MyBatisSpringLegacy"
```

## 파일 정리

- mapper 관련 패키지/파일 삭제
- mapper 관련 테스트 파일 삭제
- TimeMapper.xml 파일명 변경
  → MapperTemplate.xml



## RootConfig.java

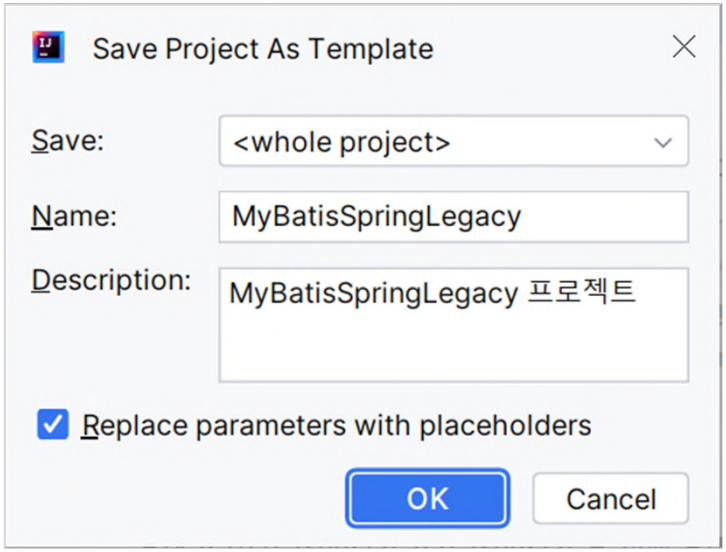
```
@Configuration
@PropertySource({"classpath:/application.properties"})
// @MapperScan(basePackages  = {})
public class RootConfig {
                ...
}
```

## resources/org/scoula/mapper/MapperTemplate.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="">

</mapper>
```

## 프로젝트 템플릿 만들기

- File > New Project Setup > Save Project As Template...




DB연동 작업

DB생성
테이블 생성
디비워크벤치에서 하고!

필요한 코딩
1 인터페이스 와 메서드 시그니처정의
2 xml에 쿼리정의
를 하면된ㄷ다!