Spring Framework
- JdbcTemplete & MyBatis

Spring Framework
- 데이터베이스 연동

CONTENTS

- JDBC 프로그래밍의 단점을 보완하는 스프링
- DataSource 설정
- JdbcTemplate을 이용한 쿼리 실행
- 트랜잭션 처리
- 스프링의 익셉션

스프링의 데이터베이스 연동 지원

- **스프링은 JDBC, 하이버네이트, iBATIS등의 다양한 기술을 이용해서 손쉽게 DAO 클래스를 구현할 수 있도록 지원**
- Spring은 JDBC를 비롯하여 ORM 프레임워크를 직접적으로 지원하고 있기 때문에 simple하게 JDBC뿐만 아니라 ORM 프레임워크들과의 연동도 매우 쉬움
- Spring은 JDBC, ORM 프레임워크 등의 다양한 기술을 이용해서 손쉽게 DAO클래스를 구현할 수 있도록 지원

  - **템플릿 클래스를 통한 데이터 접근 지원**
  - **의미 있는 예외 클래스 제공**
  - **트렌젝션 처리**

스프링의 데이터베이스 연동 지원

- **데이터 베이스 연동을 위한 템플릿 클래스**
  - 데이터에 접근하는 코드는 Connection 생성,PrepareStatement, ResultSet 등 거의 동일한 코드 구성을 갖는다.
  - 스프링은 데이터베이스 연동을 위한 템플릿 클래스를 제공함으로써 개발자가 중복된 코드를 입력 해야 하는 성가신 작업을 줄일 수 있도록 돕는다.
  - JDBC 뿐 아니라, iBATIS, JMS와 가은 다양한 기술에 대해 템플릿 클래스를 지원하고 있다.
  - **Template 클래스들**
    - **JDBC : JdbcTemplate**
    - **iBatis : SqlMapClientTemplate**
    - **Hibernate : HibernateTemplate**

스프링의 데이터베이스 연동 지원

- **스프링의 예외지원**
  - 스프링은 데이터베이스 처리과정에서 발생하는 예외가 왜 발생을 했는지 좀더 구체적으로 확인 할 수 있도록 하기 위해, 데이터 베이스처리와 관련된 예외클래스를 지원하고 있다.
  - JdbcTemplate 클래스는 처리과정에서 SQLException이 발생하면 스프링이 제공하는 예외 클래스 중 알맞은 예외클래스로 변환해서 발생 시킨다.
  - JdbcTemplat 뿐만 아니라 SqlMapClientTempate과 같이 스프링이 제공하는 템플릿 클래스는 내부적으로 발생하는 예외클래스를 스프링이 제공하는 예외클래스로 알맞게 변환해서 예외를 발생 시킨다.
  - 스프링이 제공하는 테플릿 클래스를 사용하면 데이터베이스 연동을 위해 사용하는 기술에 상관 없이 동일한 방식으로 예외를 처리 할 수 있다.
  - Spring의 모든 예외 클래스들은 DataAccessException을 상속

    ex) BadSqlGrammerException, DataRetrievalFailureException

JDBC 프로그래밍의 단점을 보완하는 스프링

- **JDBC 프로그래밍**

  : 항상 반복적으로 사용하는 코드가 존재

- **반복코드를 줄이기 위해 템플릿 메서드 패턴과 전략 패턴을 사용해서 JdbcTemplate 클래스를 제공.**

- **간단한 트랜잭션 처리**

  트랜잭션의 경우 필요한 메서드 위에 @Transactionoal 애노테이션 사용

  커밋과 롤백을 자동으로 처리하기 때문에, 핵심코드에만 집중가능.

- **JdbcTemplate 클래스**
  - SQL을 실행 하기 위한 메서드를 제공
  - 데이터 조회, 삽입, 수정, 삭제를 위한 SQL 쿼리를 실행 할 수 있다.
  - try~catch~finally 블록 및 커넥션 관리를 위한 중복 코드 삭제로 인한 코드량 감소 및 개발용이

DB 생성

- **Mysql 에서 스키마 생성 : board**
- **member 테이블 DDL**

```
create table board.MEMBER (
    ID int auto_increment primary key,
    EMAIL varchar(255),
    PASSWORD varchar(100),
    NAME varchar(100),
    REGDATE datetime,
    unique key (EMAIL)
)
```

DataSource 설정

- **스프링이 제공하는 DB 연동 기능들은 DataSource를 사용해서 DB Connection을 구하도록 구현되어 있다.**
  - Ex) DB연동에 사용할 DataSource를 스프링 빈으로 등록하고, DB 연동 기능을 구현한 빈 객체(DAO)는 DataSource를 주입 받아 사용.
  - **DataSource 를 제공하는 커넥션 풀 모듈 중 대표적인 모듈 3가지를 볼 수 있다.**
    - **HikariCP**
    - **c3p0**
    - **dbcp**

DataSource 설정 : dbcp

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
                    <dependency>
                            <groupId>mysql</groupId>
                            <artifactId>mysql-connector-java</artifactId>
                            <version>5.1.44</version>
                    </dependency>

<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-pool2 -->
                    <dependency>
                            <groupId>org.apache.commons</groupId>
                            <artifactId>commons-pool2</artifactId>
                            <version>2.4.2</version>
                    </dependency>

<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->
                    <dependency>
                            <groupId>org.apache.commons</groupId>
                            <artifactId>commons-dbcp2</artifactId>
                            <version>2.1.1</version>
                    </dependency>
```

DataSource 설정

```
<bean
    id="dataSource"
    class="org.apache.commons.dbcp2.BasicDataSource"
    p:driverClassName="com.mysql.jdbc.Driver"
    p:url="jdbc:mysql://localhost:3306/board?characterEncoding=utf8"
    p:username="bit"
    p:password="bit" />
```

DataSource 설정 : HikariCP

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
                    <dependency>
                            <groupId>mysql</groupId>
                            <artifactId>mysql-connector-java</artifactId>
                            <version>5.1.44</version>
                    </dependency>

                    <dependency>
                            <groupId>com.zaxxer</groupId>
                            <artifactId>HikariCP</artifactId>
                            <version>3.4.5</version>
                    </dependency>
```

DataSource 설정 : HikariCP

```
<bean id="dataSource"
        class="com.zaxxer.hikari.HikariDataSource"
        p:driverClassName="com.mysql.cj.jdbc.Driver"
        p:jdbcUrl="jdbc:mysql://localhost:3306/project?serverTimezone=UTC"
        p:username="bit"
        p:password="bit"
        destroy-method="close"
>
```

DataSource 설정 : c3p0

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>5.1.44</version>
                </dependency>

                <dependency>
                        <groupId>com.mchange</groupId>
                        <artifactId>c3p0</artifactId>
                        <version>0.9.2.1</version>
                </dependency>
```

DataSource 설정

```
<bean
        id="dataSource"
        class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="com.mysql.jdbc.Driver" />
        <property
        name="jdbcUrl"
        value="jdbc:mysql://localhost/board?characterEncoding=utf8" />
        <property name="user" value="bit" />
        <property name="password" value="bit" />
</bean>
```

JdbcTemplate을 이용한 쿼리 실행

- JdbcTemplate 템플릿 생성

```java
@Repository
public class JdbcTempleteMemberDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    …

}
```

```xml
<bean id="jdbcTemplate"
    class="org.springframework.jdbc.core.JdbcTemplate"
    p:dataSource-ref="dataSource" />
```

JdbcTemplate을 이용한 쿼리 실행

- **JdbcTemplate 템플릿을 이용한 조회 쿼리 실행**
  - **query()**
    - **List<T> query(String sql, RowMapper<T> rowMapper)**
    - **List<T> query(String sql, Object[] args, RowMapper<T> rowMapper)**
    - **List<T> query(String sql, RowMapper<T> rowMapper, Object… args)**
  - sql 파라미터로 전달받은 쿼리를 실행하고, RowMapper를 이용해서 ResultSet의 결과를 자바 객체로 변환한다.
  - sql 파라미터가 인덱스 기반 파라미터(PreparedStatement의 물음표)를 가진 쿼리인 경우, args 파라미터를 이용해서 각 인덱스 파라미터 값을 지정.

JdbcTemplate을 이용한 쿼리 실행

- RowMapper<T> 인터페이스

```
package org.springframework.jdbc.core

Public interface RowMapper<T>{
        T mapRow(ResultSet rs, int rowNum) throws SQLException;
}
```

Dao 구현

```
@Repository
public class JdbcTempleteMemberDao {

@Autowired
private JdbcTemplate jdbcTemplate;

public List<Member> selectList(int startRow, int count) throws SQLException {
    List<Member> memberList = jdbcTemplate.query(
            "select * from project.member order by uname limit ?, ?",
             new RowMapper<Member>() {
                 @Override
                 public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
                     Member member = new Member();
                     member.setIdx(rs.getInt("idx"));
                     member.setUid(rs.getString("uid"));
                     member.setUpw(rs.getString("upw"));
                     member.setUname(rs.getString("uname"));
                     member.setUphoto(rs.getString("uphoto"));
                     return member;
                 }
             }, startRow, count);
    return memberList;
}
```

```java
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.aia.op.member.model.Member;

public class MemberRowMapper implements RowMapper<Member> {

    @Override
    public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
        Member member = new Member();
        member.setIdx(rs.getInt("idx"));
        member.setUid(rs.getString("uid"));
        member.setUpw(rs.getString("upw"));
        member.setUname(rs.getString("uname"));
        member.setUphoto(rs.getString("uphoto"));
        return member;
    }

}
```

Dao 구현

```java
import java.sql.SQLException;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import com.aia.op.member.model.Member;

@Repository
public class JdbcTempleteMemberDao2 {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public List<Member> selectList(int startRow, int count) throws SQLException {
        List<Member> memberList = jdbcTemplate.query(
            "select * from project.member order by uname limit ?, ?",
            new MemberRowMapper(),
            startRow, count);
        return memberList;
    }
}
```

JdbcTemplate을 이용한 쿼리 실행

- 결과가 1생인 경우의 조회 메서드
  - queryForObject(쿼리, 결과 타입 또는 로우맵퍼객체, 매개변수,...)
  - Dao 구현

```
public int selectTotalCount(Connection conn) throws SQLException {

        return jdbcTemplate.queryForObject(
                "select count(*) from project.member",
                Integer.class);

}
```

JdbcTemplate을 이용한 변경 쿼리 실행

- **Insert, Update, Delete 쿼리를 실행할 때는 아래 메서드를 사용.**
  - **int update(Strring sql)**
  - **int update(Strring sql, Object … args)**

```
public int editMember(Member member) throws SQLException {

        int result = 0;
        String sql = "update project.member set upw=?,uname=?,uphoto=? where idx=?";

        result = jdbcTemplate.update(
                        sql,
                        member.getUpw(),
                        member.getUname(),
                        member.getUphoto(),
                        member.getIdx());

        return result;
}
```

DAO 완성

Open Project의 Dao를 완성해봅시다.

@Transactional을 이용한 트랜잭션 처리

- 트랜잭션 범위에서 실행하고 싶은 메서드에 **@Transactional** 애노테이션만 붙이면 된다.
- 정상적인 처리를 위해서 아래 두 가지 설정을 해주어야 한다.
  - **PlatformTransactionManager 빈 설정**
  - **@Transactional 애노테이션 활성화 설정**

```
<!-- PlatformTransactionManager 빈 설정 -->
<bean id="transactionManager"
  class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
</bean>

<!-- @Transactional 애노테이션 활성화 설정 -->
<tx:annotation-driven transaction-manager="transactionManager"/>
```

DB 연동 관련 예외

- ✓ Mysql 서버에 연결할 **권한이 없는 경우**
- ✓ **DB가 실행 중이 아니거나**, 방화벽에 막혀 있는 경우, DB 자체에 대한 네트워크 연결을 못할 경우
- ✓ **Sql문법 오류**, **공백 문자 누락**

로그 메시지

- **Log4j**

  : 로그 메시지를 남기기 위해 사용되는 로깅 프레임워크

- **Pom.xml 에 의존 추가**

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
    <type>bundle</type>
</dependency>
```

- **로그 메시지를 어떤 형식으로 어디에 기록할지에 대한 정보를 설정 파일로 부터 읽어옴.**

로그 메시지

Log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
     <appender name="console" class="org.apache.log4j.ConsoleAppender">
     <layout class="org.apache.log4j.PatternLayout">
     <param name="ConversionPattern"
     value="[%t] [%d{yyyy-MM-dd HH:mm:ss}] %-5p %c:%M - %m%n" />
     </layout>
     </appender>

     <root>
     <priority value="INFO" />
     <appender-ref ref="console" />
     </root>

     <logger name="org.springframework.jdbc">
     <level value="DEBUG" />
     </logger>
</log4j:configuration>
```

Spring Framework
- MyBatis 연동

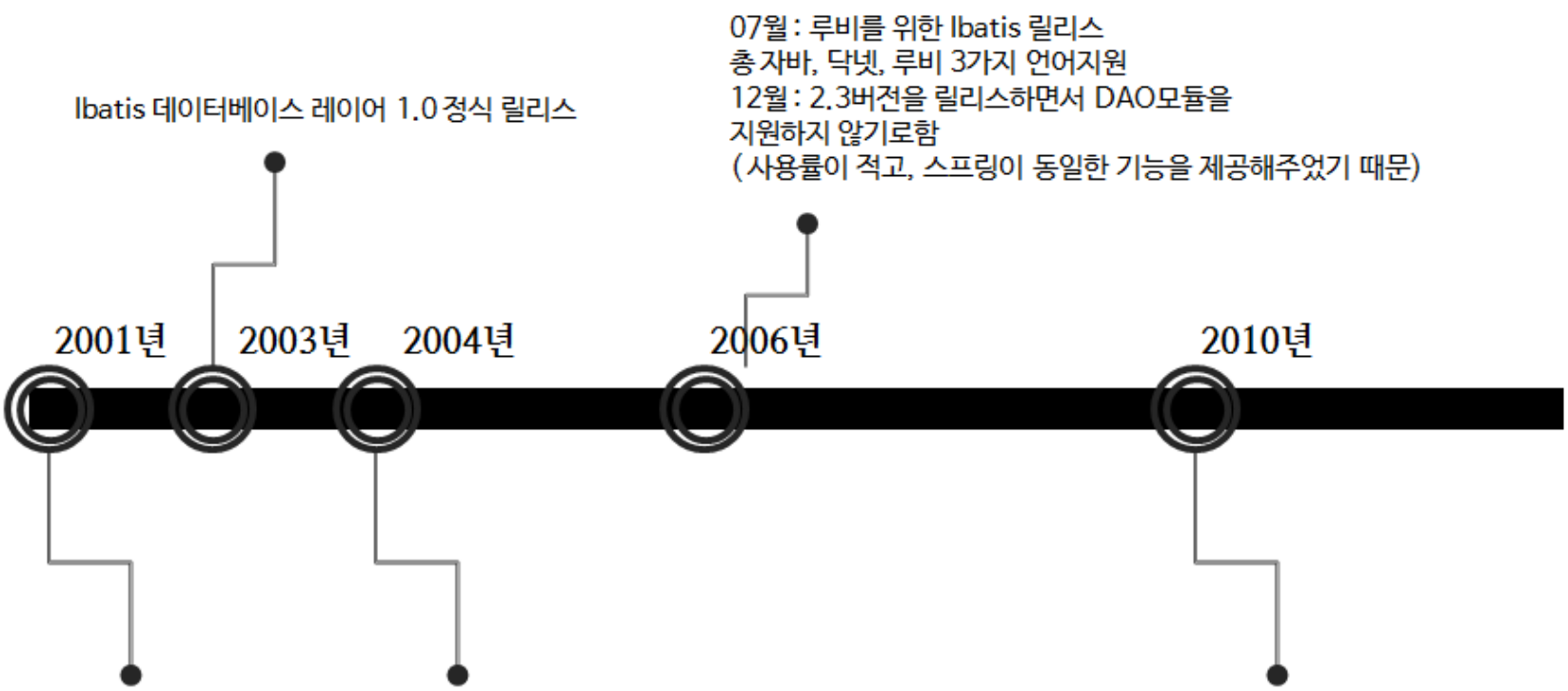
개요

- **JDBC 코드의 패턴**
  - Connection -> Statement ->쿼리전송->연결 close
  - 모든 JDBC 코드는 위의 패턴을 가진다.
  - 이 패턴을 캡슐화 하여 JDBC 코드를 간편하게 사용할 수 있도록 Framework화 가능
- **iBatis(MyBatis) 란**
  - SQL실행 결과를 자바 빈즈 혹은 Map 객체에 매핑해주는 Persistence 솔루션으로 SQL을 소스코드가 아닌 XML로 따로 분리해 관리하도록 지원
- **장점**
  - SQL 문장과 프로그래밍 코드의 분리
  - JDBC 라이브러리를 통해 매개변수를 전달하고 결과를 추출하는 일을 간단히 처리가능
  - 자주 쓰이는 데이터를 변경되지 않는 동안에 임시 보관(Cache) 가능
  - 트랜잭션처리 제공

MyBatis 새로운 기능

- 전반적으로 크게 달라진 점은 없어 기존에 ibatis를 사용하셨던 개발자들은 어려움 없이 사용가능.

MyBatis 역사



- **현재 Mybatis에서 제공하는 최신버전은 3.3.0버전**
- **http://blog.mybatis.org/p/products.html 에서 다운받을 수 있도록 제공**

■ Mybatis와 Ibatis의 차이점

구분	Ibatis(아이바티스)	Mybatis(마이바티스)
네임스페이스	선택사항	필수사항
매핑구문정의	xml만 사용	xml과 에노테이션을 사용
동적 SQL	XML 엘리먼트만 사용 동적 SQL을 위한 XML 엘리먼트는 16개 내외	XML 엘리먼트 및 구문 빌더 사용 동적SQL을 위한 XML엘리먼트는 4개 내외 (if, choose, trim, foreach,set)
스프링 연동	스프링 자체 구현체 사용	마이바티스 별도 모듈 사용
지원계획	향후 아이바티스에 대한 공식적인 릴리스는 없음	향후 계속 릴리스 될 예정

XML과 에노테이션

- **XML**
  - 사용방법은 기존과 동일하나 용어들이 변경

이전용어	변경된 용어
SqlMapConfig	Configuration
sqlMap	mapper

  - XML 엘리먼트가 축소

if	조건문
choose(when otherwise)	반복문
trim(where)	공백제거
foreach	반복문(IN절 사용시 유용)
set	update에서 마지막으로 명시된 칼럼 표기에서 쉼표제거

Mybatis

- Mybatis는 직접 스프링과 연동하기 위한 모듈을 제공하고 있어 이 모듈을 이용해서 스프링이 제공하는 DataSource 및 트랜잭션 관리 기능을 MyBais에 적용이 가능하다.

- **Spring 과 MyBatis와 연동 방법**
  - MyBatis-Spring 모듈 추가
  - SqlSessionFactoryBean을 이용해서 SqlSessionFactory 설정
  - 트랜재개션 설정
  - MyBatis 를 이용해서 DAO 구현
  - SqlSession 을 이용해서 구현
  - 매퍼 동적 생성을 이용해서 구현

Mybatis Spring 모듈 추가

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>${org.springframework-version}</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${org.springframework-version}</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.3.0</version>
</dependency>
```

SqlSessionFactoryBean 과 트랜잭션 관리자 설정

- mybatis-spring 모듈이 제공하는 SqlSessionFactoryBean을 이용해서 mybatis의 SqlSessionFactory를 생성

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

SqlSessionFactoryBean 과 트랜잭션 관리자 설정

- Mapper 작성 (MemberMapper.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="com.aia.op.mapper.mybatis.MemberMapper">

    <select id="selectById"
        resultType="com.aia.op.member.model.Member">
        select * from member where member_id = #{id}
    </select>

    <insert id="insertMember"
        parameterType="com.aia.op.member.model.Member">

        insert into member
        values( #{member_id}, #{member_name}, #{password}, #{photo}, #{regdate})

    </insert>
</mapper>
```

SqlSessionFactoryBean 과 트랜잭션 관리자 설정

- MyBatis 트랜잭션 관리자는 DataSourceTransactionManager를 사용
  servlet-context.xml 아래 코드 적용

```
<!-- 트랜젝션 처리 시작 -->

<beans:bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
          <beans:property name="dataSource">
          <beans:ref bean="dataSource" />
          </beans:property>
</beans:bean>


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


<!-- 트랜젝션 처리 끝 -->
```

MyBatis를 이용한 DAO 구현

- SqlSessionFactoryBean을 이용해서 mybatis의 SqlSessionFactory를 생성하면 MyBatis를 이용해서 DAO 구성 가능

- DAO 구현 방법은 두 가지로 구분
  - **SqlSessionTemplate을 이용한 DAO 구현**
  - **자동 매퍼 생성 기능을 이용한 DAO 구현**

SqlSessionTemplate을 이용한 DAO 구현

- SqlSessionTemplate 클래스는 SqlSession 을 위한 스프링 연동 부분을 구현하고 있고, DAO 는 SqlSessionTemplate를 이용해서 구현

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


<beans:bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
         <beans:constructor-arg index="0" ref="sqlSessionFactory" />
</beans:bean>
```

SqlSessionTemplate을 이용한 DAO 구현

com.aia.op.member.dao.MemberMyBatisDao

```
public class MemberMyBatisDao {

    @Autowired
    private SqlSessionTemplate sqlSessionTemplate;

    public void insert(Member member) {
        String str = "com.aia.op.mapper.mybatis.MemberMapper.insertMember";
        int num = sqlSessionTemplate.update(str, member);
        System.out.println(num);
    }

    public Member selectById(String userid) {
        String str = "com.aia.op.mapper.mybatis.MemberMapper.selectById";
        return (Member) sqlSessionTemplate.selectOne(str, userid);
    }

}
```

자동 매퍼 생성 기능을 이용한 DAO 구현

- MyBatis를 이용해서 DAO를 구현 할 때 대부분의 코드는 단순희 SqlSession의 메서드를 호출하는 것으로 끝난다.
  이러한 단순 코드 작업을 줄여주기 위해서 MyBatis는 인터페이스를 이용해서 런타임에 매퍼 객체를 생성하는 기능을 제공.

자동 매퍼 생성 기능을 이용한 DAO 구현

com.aia.op.member.dao.LoginDao.java

```
package com.bitcamp.openproject.member.dao;

import com.bitcamp.openproject.member.model.Member;

public interface LoginDao {

        public Member selectById(String userId);
        public void insertMember(Member member);
}
```

자동 매퍼 생성 기능을 이용한 DAO 구현

com.aia.op.member.service.LoginService.java

```
public class LoginService {

        // @Autowired
        // private MemberMyBatisDao dao;

        @Autowired
        private SqlSessionTemplate sqlSessionTemplate;

        private LoginDao dao;

        @Transactional
        public User login(
                  String userid,
                  String password,
                  HttpServletRequest req) throws SQLException {

                  dao = sqlSessionTemplate.getMapper(LoginDao.class);
```

자동 매퍼 생성 기능을 이용한 DAO 구현

com.aia.op.member.service.LoginService.java

```
                Member member = dao.selectById(userid);
                System.out.println(member);

                if (member == null) {
                        throw new Exception();
                }
                if (!member.matchPassword(password)) {
                        throw new Exception();
                }
                User user = new User();
                user.setName(member.getName());
                user.setUserid(member.getMemberid());
                user.setPhoto(member.getPhoto());

                return user;
        }

}
```

XML과 에노테이션

- **참고사이트**
  - http://mybatis.github.io/mybatis-3/ko/index.html