MyBatis

#01. 사용 절차

1. Maven 저장소를 통해 라이브러리 정보를 `build.gradle`에 등록
2. MyBatis를 관리할 패키지 생성
    - ex) kr.hossam.myschool
3. 데이터베이스 접속 정보를 저장하고 있는 환경설정 파일 생성
    - `config.xml`
4. 처리하고자 하는 테이블의 구조를 본딴 Beans를 생성
    - 생성자는 정의하지 않는다.
    - `kr.hossam.model.Department`
5. SQL문을 저장하고 있는 파일을 테이블 단위로 생성
    - `~~~Mapper.xml`
6. 환경설정 정보와 SQL문이 저장된 파일들을 로드해서 DB연동을 처리할 객체를 생성하는 클래스 작성
    - kr.hossam.myschool.MyBatisConnectionFactory

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>
    <!-- MySQL 접속 정보를 지정한다. -->
    <properties>
        <property name="hostname" value="DB가_설치된_컴퓨터_주소" />
        <property name="portnumber" value="MYSQL_포트번호" />
        <property name="database" value="데이터베이스_이름" />
        <property name="username" value="접속자_아이디" />
        <property name="password" value="비밀번호" />
    </properties>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver" />
                <property name="url"
value="jdbc:mysql://${hostname}:${portnumber}/${database}?
characterEncoding=UTF8&amp;serverTimezone=UTC" />
                <property name="username" value="${username}" />
                <property name="password" value="${password}" />
            </dataSource>
        </environment>
    </environments>

    <!-- 실행할 SQL문을 정의한 Mapper XML의 경로를 지정한다. -->
```

```
    <mappers>
        <mapper resource="kr/hossam/mapper/DepartmentMapper.xml" />
    </mappers>
</configuration>
```

~~~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">

<mapper namespace="이파일의_내용을_대표하는_이름(통상적으로_파일명)">

    <!-- Beans 클래스의 객체이름(id)과 클래스이름(type)을 명시한다. -->
    <resultMap type="kr.hossam.model.Department" id="department">
        <!-- Beans의 멤버변수(property)이름과 대상 테이블의 컬럼(column)을 연결한다.
-->
        <result property="deptno" column="deptno" />
        <result property="dname" column="dname" />
        <result property="loc" column="loc" />
    </resultMap>

    <!-- 데이터 저장을 위한 기능 정의 -->
    <insert id="메서드이름에 해당하는 식별자"
            parameterType="파라미터의 데이터 타입(Beans의 클래스명)"
            useGeneratedKeys="자동증가 일련번호값 리턴 여부"
            keyProperty="자동증가 일련번호 컬럼명">
        SQL문
    </insert>

    <!-- 데이터 삭제를 위한 기능 정의 -->
    <delete id="메서드이름에 해당하는 식별자"
            parameterType="파라미터의 데이터 타입(Beans의 클래스명)">
        SQL문
    </delete>

    <!-- 데이터 갱신을 위한 기능 정의 -->
    <update id="메서드이름에 해당하는 식별자"
            parameterType="파라미터의 데이터 타입(Beans의 클래스명)">
        SQL문
    </update>

    <!-- 단일행 조회를 위한 기능 정의 -->
    <select id="메서드이름에 해당하는 식별자"
            parameterType="파라미터의 데이터 타입(Beans의 클래스명)"
            resultMap="리턴될 Beans이름">
        SQL문
    </select>

    <!-- 다중행 조회를 위한 기능 정의 -->
```
~~~

```
    <select id="메서드이름에 해당하는 식별자"
            resultMap="리턴될 Beans이름">
        SQL문
    </select>
</mapper>
```

MyBatisConnectionFactory.java

```
package kr.hossam;

import java.io.IOException;
import java.io.Reader;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class MyBatisConnectionFactory {
    /** 데이터베이스 접속 객체 */
    // import org.apache.ibatis.session.SqlSessionFactory;
    private static SqlSessionFactory sqlSessionFactory;

    /** XML에 명시된 접속 정보를 읽어들인다. */
    // 클래스 초기화 블럭 : 클래스 변수의 복잡한 초기화에 사용된다.
    // 클래스가 처음 로딩될 때 한번만 수행된다.
    static {
        // 접속 정보를 명시하고 있는 XML의 경로 읽기
        // --> import java.io.Reader;
        // --> import org.apache.ibatis.io.Resources;
        try {
            Reader reader = Resources.getResourceAsReader("환경설정파일의_경로");

            // sqlSessionFactory가 존재하지 않는다면 생성한다.
            if (sqlSessionFactory == null) {
                sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /** 데이터베이스 접속 객체를 통해 DATABASE에 접속한 세션를 리턴한다. */
    // --> import org.apache.ibatis.session.SqlSession;
    public static SqlSession getSqlSession() {
        return sqlSessionFactory.openSession();
    }
}
```

Log4j2 설정

Gradle 설정 추가

- Log4j core 2.2 (보다 상위 버전)
- Log4j API 2.2 (보다 상위 버전)

환경설정 파일 추가

프로젝트 안에 `src/main/resources` 폴더를 생성하고 그 안에 `logj2.xml` 파일을 생성한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml>
<Configuration status="INFO">
    <Appenders>
        <RollingFile name="RollingFile"
            fileName="log/log.log" filePattern="log/log-%d{yyyy-MM-dd}.log"
append="false">
            <PatternLayout charset="UTF-8" pattern="%d %5p [%c] %m%n" />
            <Policies>
                <TimeBasedTriggeringPolicy interval="1" modulate="true" />
            </Policies>
        </RollingFile>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss} [%-5level] %logger -
%msg%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Root level="DEBUG" additivity="false">
            <AppenderRef ref="Console" />
            <AppenderRef ref="RollingFile" />
        </Root>
        <logger name="org.apache.ibatis" level="DEBUG" additivity="false">
            <AppenderRef ref="Console" />
            <AppenderRef ref="RollingFile" />
        </logger>
    </Loggers>
</Configuration>
```

Log4j의 로그 수준

중요도 혹은 심각도를 의미하는 단어

우선순위	수준	의미
5	DEBUG	중요도 최하위. 개발자가 프로그램의 실행 과정을 모니터링하기 위해 직접 출력하는 메시지(개발자가 직접 사용)
4	INFO	프로그램의 동작 정보, 시스템 정보 등을 출력하는 용도 (잘 사용안함)
3	WARN	경고메시지 (잘 사용안함)

우선순위	수준	의미
2	ERROR	에러메시지 catch 블록에서 출력하는 내용이 사용(활용도 높음)
1	FATAL	최고 심각도. 시스템이 다운될 수준의 에러 내용을 기술

`log4j2.xml`에 설정된 수준값보다 우선순위가 높은 로그 메시지는 모두 출력된다.

예

- 설정값이 `DEBUG`로 설정되어 있는 경우 모든 메시지가 출력된다.
- 설정값이 `WARN`으로 설정되어 있는 경우 INFO 수준과 DEBUG 수준은 출력되지 않는다.

> 수준을 의미하는 단어는 모두 `logger`객체 안에 메서드로 존재