**퍼시스턴스(Persitence)**

퍼시스턴스는 데이터의 지속성을 의미. 즉 애플리케이션을 종료하고 다시 실행하더라도 이전에 저장한 데이터를 다시 불러올수있는 기술.

퍼시스턴스 프레임

**프레임워크(Framework)**

**라이브러리가(library)**가 개발에 필요한 도구들을 단순히 나열해 놓은 것이라면 **프레임워크(framework)**는 동작에 필요한 구조를 어느 정도 완성해 놓은 반제품 형태의 도구입니다.

**퍼시스턴스 프레임워크**

퍼시스턴스 프레이임워크는 데이터의 저장, 조회, 변경, 삭제를 다루는 클래스 및 삭제를 다루는 클래스 및 설정 파일들의 집합

퍼시턴스 프레임워크를 사용하면 JDBC프로그래밍의 복잡함이나 번거로움 없이 간단한 작업만으로 데이터베이스와 연동되는 시스템을 빠르게 개발할 수 있습니다. 안정적인 구동도 보장되므로 금상첨화가 아닐수 없습니다.

퍼시스턴스 프레임워크에는 SQL문장으로 직접 DB데이터를 다루는 'SQL 맵퍼(mapper)와 자바 객체를 통해 간접적으로 DB데이터를 다루는 객체 관계 맵퍼(Object-Relational mapper)' 가 있습니다. SQL 맵퍼의 대표주자로 'mybatis(이전버전의 이름은 iBATIS)'가 있고, 객체관계 맵퍼의 대표 주자로 하이버네이트(Hibernate)'와 '탑링크('TopLink)'가 있습니다.

**mybatis**

mybatis의 핵심은 개발과 유지보수가 쉽도록 소스코등 박혀있는 SQL을 별도의 파일로 분리하는것. 또한 단순하고 반복적인 JDBC코드를 캡슐화하여 데이터베이스 프로그래밍을 간결하게 만드는 것입니다.

**mybatis 작동 흐름**

1. 데이터 처리를 위해 DAO는 mybatis에 제공하는 객체의 메서드를 호출

2. mybatis는 SQL문이 저장된 맵퍼 파일에서 데이터 처리에 필요한 SQL문을 찾는다.

3. mybatis는 맴퍼 파일에서 찾은 SQL을 서버에 보내고자 JDBC드라이버를 사용

4. JDBC 드라이버는 SQL문을 데이터베이스 서버로 보낸다.

5. mybatis는 select문의 실행 결과를 값 객체에 담아서 반환, insert, update, delete 문인 경우 입력, 변경, 삭제된 레코드의 개수를 반환.

**일반적으로 DAO에서 mybatis를 사용하는 시나리오.**

1. Dao는 **SqlSessionFactory**에게 SQL을 실행할 객체를 요구한다.

2. SqlSessionFactory는 **SqlSession**객체를 생성하여 반환한다.

3. Dao는 **SqlSession**객체에게 SQL실행을 요청

4. SqlSession 객체는 SQL이 저장된 맵퍼 파일(XML) 에서 SQL을 찾는다.

5. SqlSession은 JDBC드라이버를 통해 데이터베이스에 질의를 실행

6. SqlSession은 데이터베이스로부터 가져온 질의 결과를 객체로 생성하여 반환

7. Dao는 사용이 끝난 SqlSession을 닫는다.

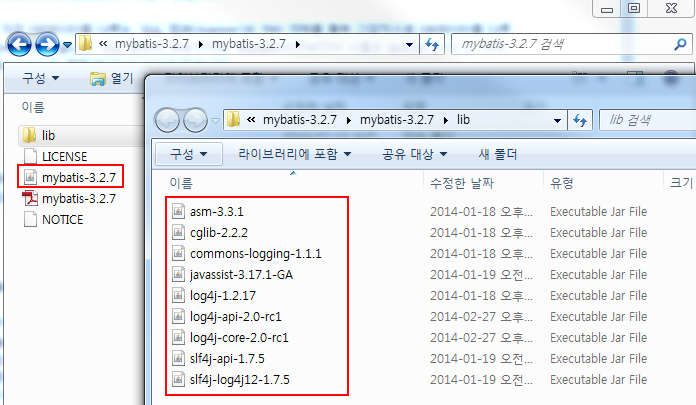
**mybatis 사용 준비**

mybatis를 사용하기위해서 관련 라이브러리 파일을 다음 사이트에서 내려받자.

**mybatis github** : <http://github.com/mybatis>

**jar배포파일 다운받는 페이지** : <https://github.com/mybatis/mybatis-3/releases>

배포된 압축파일을 풀면



mybatis-3.2.7.jar 파일과 의존 라이브러들을 해당 웹프로젝트의 /WebContent/WEB-INF/lib 폴더에 복사.

단 중복 파일이 있으면 버전이 낮은 파일을 삭제

**mybatis 프레임워크의 핵심 컴포넌트**

[표] mybatis의 핵심 컴포넌트

|  |  |
| --- | --- |
| 컴포넌트 | 설명 |
| **SqlSession** | 실제 SQL을 실행하는 객체, 이 객체는 SQL을 처리하기 위해 JDBC드라이버를 사용함. |
| **SqlSessionFactory** | SqlSession 객체를 생성함. |
| **SqlSessionFactoryBuilder** | mysql 설정 파일의 내용을 토대로 SqlSessionFactory를 생성함. |
| **mybatis 설정 파일** | 데이터베이스 연결 정보, 트랜잭션 정보, mybatis제어 정보등의 설정 내용을 포함하고 있음. SqlSessionFactory를 만들 때 사용됨 |
| **SQL 맵퍼 파일** | SQL문을 담고 있는 파일, 보통 XML파일로 작성. SqlSession객체가 참조함. |

**DAO 에서 SqlSessionFactory 사용**

SqlSessionFactory는 SQL을 실행할때 사용할 도구를 만들어줌.

DAO 클래스에서 SqlSessionFactory를 저장할 인스턴스 변수와 셋터 메서드를 선언

SqlSessionFactory sqlSessionFactory;

public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {

this.sqlSessionFactory = sqlSessionFactory;

}

**SqlSession 사용**

SqlSession은 SQL을 실행하는 도구.

이 객체는 직접 생성할수 없고, SqlSessionFactory를 통해서만 얻을수있다.

SqlSession sqlSession = sqlSessionFactory.openSession();

try {

return sqlSession.selectList("*test.dao.testDao.selectList*");

//List<E> selectList(String sqlId)

//sqlId = SQL 맵퍼의 네임스페이스 이름 + 맵퍼파일에 있는 SQL문 ID

//List<E> selectList(String sqlId, Object parameter)

//SELECT문을 실행하는데 값이 필요하다면 위와 같이 두번째 매개변수로 값을 전달.

} finally {

sqlSession.close();

}

 [표] SqlSession의 주요 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| **selectList()** | SELECT문을 실행. 값 객체(Value Object)목록을 반환함 |
| **selectOne()** | SELECT문을 실행. 하나의 값 객체를 반환함 |
| **insert()** | INSERT문을 실행. 반환값은 입력한 데이터 개수. |
| **update()** | UPDATE문을 실행. 반환값은 변경한 데이터 개수. |
| **delete()** | DELETE문을 실행. 반환값은 삭제한 데이터 개수. |

**SqlSession의 메서드와 매개변수 값 전달.**

 DAO 일부 .....

 sqlSession.insert("*test.dao.testDao.insert*", project);

 .....

 맵퍼파일 일부 ...

 <insert id="insert" parameterType="project">

    insert into PROJECTS\_web(PNO, PNAME,CONTENT,STA\_DATE,END\_DATE,STATE,CRE\_DATE,TAGS)

    values ( projects\_web\_pno\_seq.nextval, #{title},#{content},#{startDate},#{endDate},0,sysdate,#{tags})

 </insert>

 .....

=> **#{프로퍼티명}**​자리에 project객체의 프로퍼티 값이 온다.

    객체의 프로퍼티란? 인스턴스 변수를 말하는 것이 아니라 겟터/셋터를 가리키는 용어,  프로퍼티 이름은 겟터/셋터 메서드의 이름에서 추출.

    예를 들어 #{title} 자리에는 project객체의 getTitle() 반환값이 놓이게 된다.

**commit()과 rollback() 메서드**

DBMS는 INSERT, UPDATE, DELETE문을 실행하면 그 작업 결과를임시 데이터베이스에 보관. 클라이언 요청이 있어야만 임시 데이터베이스의 작업물을 운영 데이터베이스에 반영

commit()은 임시 데이터베이스에 보관된 작업결과를 운영 데이터베이스에 적용하라고 요청할때 사용하는 메서드

int count = sqlSession.insert("spms.dao.ProjectDao.insert",project);

sqlSession.**commit();**

return count;

rollback()은 임시 데이터베이스의 작업 결과를 운영 데이터베이스에 반영하지 않고 취소할때 호출.

**자동 커밋** (Auto-commit)

INSERT, UPDATE, DELETE 을 실행할때 자동으로 커밋하고 싶다면 SqlSession객체를 생성할때 true 전달.

SqlSession sqlSession = sqlSessionFactory.openSession(true);

=> 자동 커밋으로 설정해 놓고 쓰면 편리하지만 트랜잭션을 다룰수없음.

**SQL 맵퍼 파일 작성**

|  |
| --- |
| [맵퍼 예시] OracleProjectDaoMapper.xml |
| [Colored By **Color Scripter**™](http://prev.kr/app/ColorScripter)   |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 | <?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="spms.dao.ProjectDao">    <resultMap type="project" id="projectResultMap">     <id column="PNO" property="no"/>     <result column="PNAME"    property="title"/>     <result column="CONTENT"  property="content"/>     <result column="STA\_DATE" property="startDate" javaType="java.sql.Date"/>     <result column="END\_DATE" property="endDate" javaType="java.sql.Date"/>     <result column="STATE"    property="state"/>     <result column="CRE\_DATE" property="createdDate" javaType="java.sql.Date"/>     <result column="TAGS"     property="tags"/>   </resultMap>      <select id="selectList" parameterType="map" resultMap="projectResultMap">       select PNO, PNAME, STA\_DATE, END\_DATE, STATE       from PROJECTS\_web       order by        <choose>          <when test="orderCond == 'TITLE\_ASC'">PNAME asc</when>         <when test="orderCond == 'TITLE\_DESC'">PNAME desc</when>         <when test="orderCond == 'STARTDATE\_ASC'">STA\_DATE asc</when>         <when test="orderCond == 'STARTDATE\_DESC'">STA\_DATE desc</when>         <when test="orderCond == 'ENDDATE\_ASC'">END\_DATE asc</when>         <when test="orderCond == 'ENDDATE\_DESC'">END\_DATE desc</when>         <when test="orderCond == 'STATE\_ASC'">STATE asc</when>         <when test="orderCond == 'STATE\_DESC'">STATE desc</when>         <when test="orderCond == 'PNO\_ASC'">PNO asc</when>          <otherwise>PNO desc</otherwise>           </choose>   </select>      <insert id="insert" parameterType="project">     insert into PROJECTS\_web(PNO, PNAME,CONTENT,STA\_DATE,END\_DATE,STATE,CRE\_DATE,TAGS)     values ( projects\_web\_pno\_seq.nextval, #{title},#{content},#{startDate},#{endDate},0,sysdate,#{tags})   </insert>      <select id="selectOne" parameterType="int" resultMap="projectResultMap">     select PNO, PNAME, CONTENT, STA\_DATE, END\_DATE, STATE, CRE\_DATE, TAGS     from PROJECTS\_web     where PNO=#{value}   </select>      <update id="update" parameterType="project">     update PROJECTS\_web      <set>         <if test="title != null">PNAME=#{title},</if>         <if test="CONTENT != null">CONTENT=#{content},</if>         <if test="STA\_DATE != null">STA\_DATE=#{startDate},</if>         <if test="END\_DATE != null">END\_DATE=#{endDate},</if>         <if test="STATE != null">STATE=#{state},</if>         <if test="TAGS != null">TAGS=#{tags}</if>            <!-- mybatis는 SET절의 끝에 콤마(,)가 있으면 제거 -->                 </set>     where PNO=#{no}   </update>      <delete id="delete" parameterType="int">     delete from PROJECTS\_web      where PNO=#{value}   </delete> </mapper> | |

**동적 SQL의 사용**

SQL문을 동적으로 생성.

동적 SQL엘리먼트 : mybatis는 동적 SQL을 위한 엘리먼트를 제공. JSTL코어 라이브러리에 정의된 태그들과 비슷.

[표] 동적 SQL을 작성할때 사용하는 엘리먼트들

|  |  |
| --- | --- |
| 엘리먼트 예 | 설명 |
| **<if test="조건">SQL 문</if>** | <if> 태그는 어떤 값의 상태를 검사하여 참일경우에만 SQL문을 포함하고 싶을때 사용. test속성에 지정된 조건이 참이면 <if>태그의 내용을 반환 |
| **<choose>**  **<when test="조건1">SQL문</when>**  **<when test="조건2">SQL문</when>**  **<otherwise>SQL 문</otherwise>**  **</choose>** | <choose>태그는 검사할 조건이 여러개일경우에 사용한다. test속성에 지정된 조건이 참이면<when>태그의 내용을 반환. 일하는 조건이 없으면 <otherwise>내용을 반환 |
| **<where>**  **<if test="조건1">SQL 문</if>**  **<if test="조건2">SQL 문</if>**  **</where>** | <where>태그는 WHERE절을 반환한다. <where>안의 하위 태그를 실행하고 나서 반환값이 있으면 WHERE절을 만들어 반환하고, 없으면 WHERE절을 반환하지 않는다. |
| **<trim prefix="단어"**  **prefixOverrides="문자열|문자열">**  **<if test="조건1">SQL 문</if>**  **<if test="조건2">SQL 문</if>**  **</trim>** | <trim>태그는 특정 단어로 시작하는 SQL문을 반환하고 싶을때 사용한다. prefix는 반환값 앞에 붙일 접두어를 지정한다. prefixOverrides는 반환할 값에서 제거해야하는 접두어를 지정한다.    다시말하면 <trim>의 반환값이 있다면, 그 값의 앞 부분이 prefixOverride에 지정된 문자열과 일치할 경우 그 문자열을 제거한다. 그리고 그 값의 앞부분에 prefix로 지정한 접두얼르 붙여 반환한다. |
| **<set>**  **<if test="조건1">SQL 문</if>**  **<if test="조건2">SQL 문</if>**  **</set>** | <set>태그는 UPDATE문의 SET절을 만들 때 사용한다. 조건이 참인<if>의 내용은 SET절에 포함된다. SET절의 항목이 여러개일 경우 자동으로 콤마(,)를 붙인다. |
| **<foreach**  **item="항목"**  **index="인덱스"**  **collection="목록"**  **open="시작문자열"**  **close="종료문자열"**  **separator="구분자">**  **</foreach>** | <foreach>태그는 목록의 값을 가지고 SQL문을 만들때 사용한다. 특히 IN(값, 값, .) 조건을 만들 때 좋다.  item속성에는 항목을 가리킬때 사용할 변수의 이름을 지정한다.  index속성에는 항목의 인덱스 값을 꺼낼때 사용할 변수 이름을 지정한다.  collection속성에는 java.util.List구현체나 배열 객체가 온다.  open속성에는 최종 반환값의 접두어를 지정한다.  close속성에는 최종 반환값의 접미어를 지정한다.  separator속성은 반복으로 생성하는 값을 구분하기 위해 붙이는 문자열을 지정한다. |
| **<bind name="변수명" value="값" />** | <bind>태그는 변수를 생성할때 사용한다. |

**mybatis 설정파일**

mybatis프레임워크의 동작 환경을 설정하는 파일

mybatis설정 파일의 루트 엘리먼트는 configuration 입니다.

[표] configuration의 주요 자식엘리먼트들

|  |  |
| --- | --- |
| 엘리먼트 | 용도 |
| **properties** | 프로퍼티 파일이 있는 경로 설정. <property>를 사용하여 개별 프로퍼티 정의 가능 |
| **settings** | 프레임워크의 실행 환경을 설정 |
| **typeAliases** | 자바클래스 이름 (패키지이름 포함)에 대한 별명 설정 |
| **typehandlers** | 칼럼의 값을 자바 객체로, 자바 객체를 칼럼의 값으로 변환해 주는 클래스를 설정 |
| **environments** | 프레임워크에서 사용할 데이터베이스 정보 (트랜잭션 관리자, 데이터 소스)를 설정. |
| **mappers** | SQL 맵퍼 파일들이 있는 경로 설정  - 자바클래스 경로를 사용하는 방법 :    <mapper resource="spms/dao/OracleProjectDao.xml"/>  - 운영체제의 파일시스템 경로를 사용하는 방법.    <mapper url="file:///c:/project/dao/OracleProjectDao.xml"/> |

|  |
| --- |
| [mybatis 설정파일 예제] mybatis-config.xml |
| [Colored By **Color Scripter**™](http://prev.kr/app/ColorScripter)   |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 | <?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> <!-- <properties resource="spms/dao/db.properties"/> -->    <!--  로그기능 활성 -->   <settings>       <setting name = "logImpl" value="LOG4J"/>   </settings>   <!--  타입 별칭지정 -->   <typeAliases>     <typeAlias type="spms.vo.Project" alias="project"/>     <typeAlias type="spms.vo.MemberVO" alias="member"/>   </typeAliases>      <environments default="development">     <environment id="development">       <transactionManager type="JDBC"/>   <!--            <dataSource type="POOLED">         <property name="driver" value="${driver}"/>         <property name="url" value="${url}"/>         <property name="username" value="${username}"/>         <property name="password" value="${password}"/>       </dataSource>    -->           <dataSource type="JNDI">             <property name="data\_source" value="java:comp/env/jdbc/studydb"/>       </dataSource>          </environment>   </environments>   <!-- 맵퍼파일 지정 -->   <mappers>     <mapper resource="spms/dao/OracleProjectDao.xml"/>   </mappers> </configuration> | |
| [참고] db.properties 내용    driver=oracle.jdbc.driver.OracleDriver  url=jdbc:oracle:thin:@localhost:1521:orcl  username=study  password=study |

**트랜잭션 관리 방식 설정**

트랜잭션(Transaction)이란, 여러개의 데이터변경 작업(insert, update, delete)을 하나의 작업으로 묶은 것입니다. mybatis에서 트랜잭션을 관리하는 방식에는 둘가지가 있습니다.

[표] mybatis의 트랜잭션 관리 유형

|  |  |
| --- | --- |
| 트랜잭션 관리 유형 | 설명 |
| JDBC | 직접 JDBC의 커밋(commit), 롤백(rollback) 기능을 사용하여 mybatis 자체에서 트랜잭션을 관리 |
| MANAGED | 서버의 트랜잭션 관리 기능을 이용. 즉 JavaEE 애플리케이션 서버(JBoss, WebLogic, WebSphere 등) 나 서블릿 컨테이너 (톰캣 서버 등) 에서 트랜잭션을 관리 |

다음과 같이 <transactionManager>를 사용하여 트랜잭션 관리 유형을 설정, type속성에 트랜잭션 관리 유형(JDBC 또는 MANAGED)을 지정

<transactionManager type="JDBC"/>

**데이터 소스 설정**

mybatis 는 JDBC표준 인터페이스인 javax.sql.DataSource구현체를 이용하여 DB커넥션을 다룹니다.

[표] mybatis의 데이터 소스 유형

|  |  |
| --- | --- |
| 데이터 소스 유형 | 설명 |
| UNPOOLED | DB 커넥션을 요청할 때마다 매번 커넥션 객체를 생성한다. 높은 성능을 요구하지 않는 단순한 애플리케이션에 적합하다. |
| POOLED | 미리 DB커넥션 객체를 생성해 두고, 요청하면 즉시 반환한다. 데이터베이스에 연결하는 과정, 즉 연결을 초기화하고 사용자를 인증하는 과정이 없기 때문에 속도가 빠르다. |
| JNDI | Java EE 애플리케이션 서버나 서블릿 컨테이너(톰캣 등) 에서 제공하는 데이터 소스를 사용한다. |

**1) mybatis에서 지원하는 DBConnection Pool 사용**

 <environments default="development">

    <environment id="development">

      <transactionManager type="JDBC"/>

**<dataSource type="POOLED">**

**<property name="driver" value="${driver}"/>**

**<property name="url" value="${url}"/>**

**<property name="username" value="${username}"/>**

**<property name="password" value="${password}"/>**

**</dataSource>**

    </environment>

  </environments>

   => <property>의 value값을 보면 ${}로 되어있는데 이는 프로퍼티를 가리키는 문법이다. <properties>에서 설정한 파일(db.properties)로 부터 프로퍼티의 값을 가져온다.

**2) mybatis에서 JNDI사용하기**

<environments default="development">

   <environment id="development">

      <transactionManager type="JDBC"/>

**<dataSource type="JNDI">**

**<property name="data\_source" value="java:comp/env/jdbc/studydb"/>**

**</dataSource>**

   </environment>

</environments>

[참고]

**web.xml**

<!-- 웹 애플리케이션에서 톰캣서버의 자원 사용 -->

<resource-ref>

<res-ref-name>jdbc/studydb</res-ref-name> <!-- JNDI 이름 -->

<res-type>javax.sql.DataSource</res-type> <!-- 리턴될 자원의 클래스이름 -->

<res-auth>Container</res-auth> <!-- 자원 관리의 주체 -->

</resource-ref>

**context.xml**

<Context>

 <!-- Default set of monitored resources -->

 <WatchedResource>WEB-INF/web.xml</WatchedResource>

<Resource name="jdbc/studydb" auth="Container" type="javax.sql.DataSource"

maxActive = "10" maxIdle="3" maxWait="10000"

username="study"

password="study"

driverClassName="oracle.jdbc.driver.OracleDriver"

url="jdbc:oracle:thin:@localhost:1521:orcl"

closeMethod="close"/>

</Context>

**mybatis 설정 파일에 로그 설정 추가**

<settings>

<setting name = "logImpl" value="LOG4J"/>

</settings>

[표] 로그출력 구현체

|  |  |
| --- | --- |
| value속성값 | 설명 |
| SLF4J | SLF4J |
| LOG4J | Log4j |
| LOG4J2 | Log4j 2 |
| JDK\_LOGGING | JDK logging |
| COMMONS\_LOGGING | Apache Commons Logging |
| STDOUT\_LOGGING | 표준 출력장치로 출력 |
| NO\_LOGGING | 로그 출력 기능 사용안함 |
| 클래스명(패키지명 포함) | org.apache.ibatis.logging.Log 인터페이스의 구현체. |

FATAL < ERROR < WARN < INFO < DEBUG < TRACE

**src/log4j.properties (로그4j 설정 파일)**

log4j.rootLogger=ERROR, stdout

log4j.logger.spms.dao=DEBUG //하위

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.CoversionPattern=%5p [%t] - %m%n

=> DEBUG보다 더 자세한 내용을 보고 싶다면 로그 출력 등급을 **TRACE**로 설정 , 예를 들어 select문의 실행결과를 로그로 출력하고 싶다면 출력등급을 TRACE로 바꾸면된다.

|  |
| --- |
| ==>  Preparing: insert into PROJECTS\_web(PNO, PNAME,CONTENT,STA\_DATE,END\_DATE,STATE,CRE\_DATE,TAGS) values ( projects\_web\_pno\_seq.nextval, ?,?,?,?,0,sysdate,?)  ==> **Parameters: 테스트프로젝트(String), 프로젝트 1234(String), 2013-12-01 00:00:00.0(Timestamp), 2014-07-01** 00:00:00.0(Timestamp), 태그 태그 태그(String)  <==    Updates: 1  /project/list.do  ==>  Preparing: select PNO, PNAME, STA\_DATE, END\_DATE, STATE from PROJECTS\_web order by PNO desc  ==> Parameters:  <==    Columns: PNO, PNAME, STA\_DATE, END\_DATE, STATE  <==       **Row: 7, 테스트프로젝트, 2013-12-01 00:00:00, 2014-07-01 00:00:00, 0**  **<==        Row: 6, 12343333, 2010-01-01 00:00:00, 2010-05-05 00:00:00, 0**  **<==        Row: 4, 프로젝트3, 2013-10-02 00:00:00, 2013-12-30 00:00:00, 0**  **<==        Row: 3, 프로젝트2, 2014-07-01 00:00:00, 2014-09-01 00:00:00, 0**  **<==        Row: 2, 프로젝트1, 2014-07-01 00:00:00, 2014-09-01 00:00:00, 0**  <==      Total: 5 |

**설정파일 불러오기**

​

String resource = "spms/dao/mybatis-config.xml";

InputStream inputStream = Resources.getResourceAsStream(resource);

**SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);**