MyBatis
(board-mybatis.war)

동의과학대학교 컴퓨터정보과
김진숙

목차

- Persistence Framework
- MyBatis란?
- MyBatis 설치
- MyBatis 설정
  - 설정 파일 : board-config.xml
  - SQL 맵퍼 파일 : board-mapper.xml
- 기존 게시판 프로젝트 변경
  - boardDao.java

Persistence Framework

- JDBC 프로그래밍의 복잡함 없이 간단한 작업만으로 DB와 연동되는 시스템을 빠르게 개발
- 모든 Persistence Framework은 내부적으로 JDBC API 사용
- 종류
  - SQL Mapper
    - SQL 문장으로 직접 데이터베이스를 다룸
    - SQL Mapper는 SQL을 명시
    - Mybatis, JdbcTemplate 등
  - ORM(Object-Relational Mapping)
    - 객체와 데이터베이스의 데이터를 자동으로 연결해주는 것
    - ORM을 이용하면 SQL이 아닌 코드(메소드)로 데이터를 조작
    - Hibernate(JPA의 구현체) 등

MyBatis(iBatis)란?

- 개발자가 지정한 SQL, 저장프로시저 그리고 몇가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크 중 SQL Mapper
- Mybatis의 이전 버전인 iBatis는 Clinton Begin 의해 2002년에 시작된 프로젝트
  - 원래는 암호 관련 소프트웨어 개발에 초점을 맞춘 프로젝트
  - iBatis -> internet + abatis(적의 공격을 방어하기 위한 장애물)
  - 2010년에 은퇴(http://ibatis.apache.org/)
  - MyBatis가 iBatis를 계승하여 발전시킴(https://blog.mybatis.org/p/about.html)
  - 아파치 라이선스 2.0으로 배포되는 자유 소프트웨어이다

MyBatis란?

- 등장 배경
  - JDBC는 관계형 데이터 베이스를 사용하기 위해 다양한 API 제공
  - 기존의 JDBC의 한계
    - 인터넷 사용자가 폭발적으로 증가하고 애플리케이션 기능이 복잡해 짐
    - SQL문이 프로그래밍 코드에 섞여 있음
    - 애플리케이션 기능이 복잡해지면 SQL 문 길이가 엄청 늘어남
    - SQL문 실행 시 오류도 많고 유지보수 문제도 발생

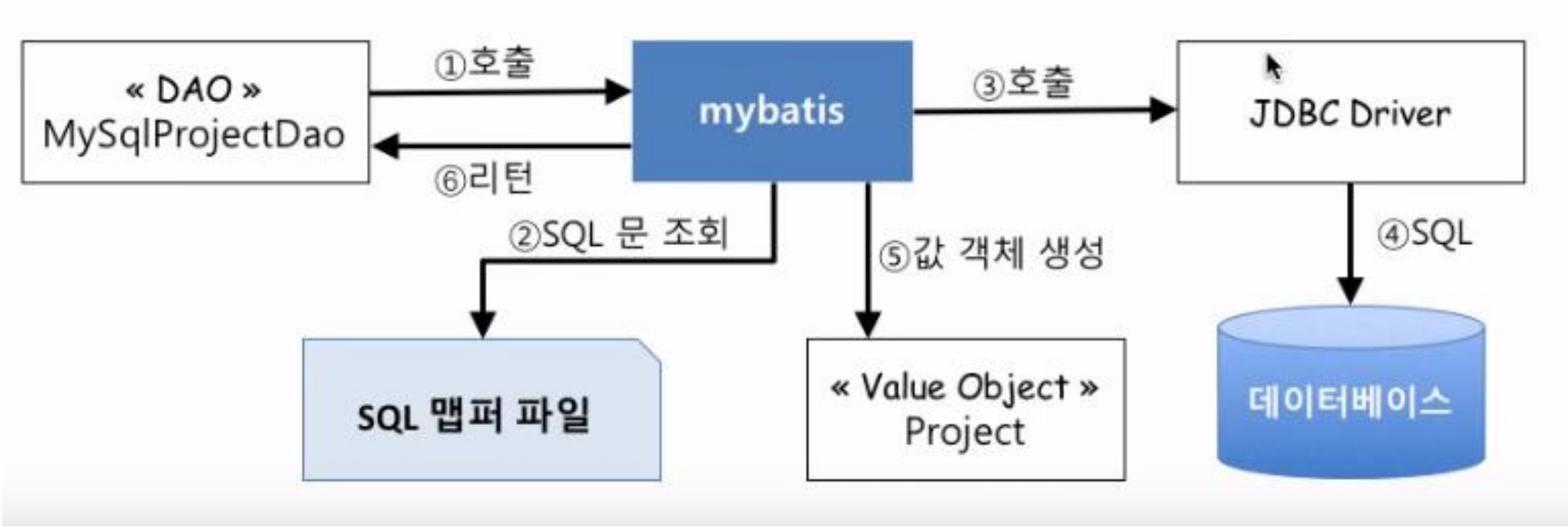
복잡한 SQL문 예

```
...
public void addGoods(GoodsVO goodsVO) throws SQLException {
  Connection con = dataFactory.getConnection();
  Statement stmt = con.createStatement();
  String query = "insert into t_Goods_info (goods_id,"+
                 "goods_sort,"+
                 "goods_title,"+
                 "goods_writer,"+
                 "goods_publisher,"+
                 "goods_price,"+
                 "goods_sales_price,"+
                 "goods_point,"+
                 "goods_published_date,"+
                 "goods_total_page,"+
                 "goods_isbn,"+
                 "goods_delivery_price,"+
```

```
                 "goods_delivery_date,"+
                 "goods_type,"+
                 "goods_writer_intro,"+
                 "goods_intro,"+
                 "goods_publisher_comment,"+
                 "goods_recommendation,"+
                 "goods_contents_order)";
query+=" values('"+
                 goodsVO.getGoods_id()+"','"+
                 goodsVO.getGoods_sort()+"','"+
                 goodsVO.getGoods_title()+"','"+
                 goodsVO.getGoods_writer()+"','"+
                 goodsVO.getGoods_publisher()+"',"+
                 goodsVO.getGoods_price()+","+
                 goodsVO.getGoods_sales_price()+","+
                 goodsVO.getGoods_point()+",'"+
                 goodsVO.getGoods_published_date()+"',"+
                 goodsVO.getGoods_page_total()+",'"+
                 goodsVO.getGoods_isbn()+"',"+
                 goodsVO.getGoods_delivery_price()+",'"+
                 goodsVO.getGoods_delivery_date()+"','"+
                 goodsVO.getGoods_type()+"','"+
                 goodsVO.getGoods_writer_intro()+"','"+
                 goodsVO.getGoods_intro()+"','"+
                 goodsVO.getGoods_publisher_comment()+"','"+
                 goodsVO.getGoods_recommendation()+"','"+
                 goodsVO.getGoods_contents_order()+"')";
                 System.out.println(query);
                 stmt.executeUpdate(query);
                 String goods_id=goodsVO.getGoods_id();
}
...
```

MyBatis를 사용하는 이유

- 자바 코드 내 **SQL문 분리**
- 반복되는 JDBC 코드를 객체화하여 단순화 시킴



MyBatis의 특징

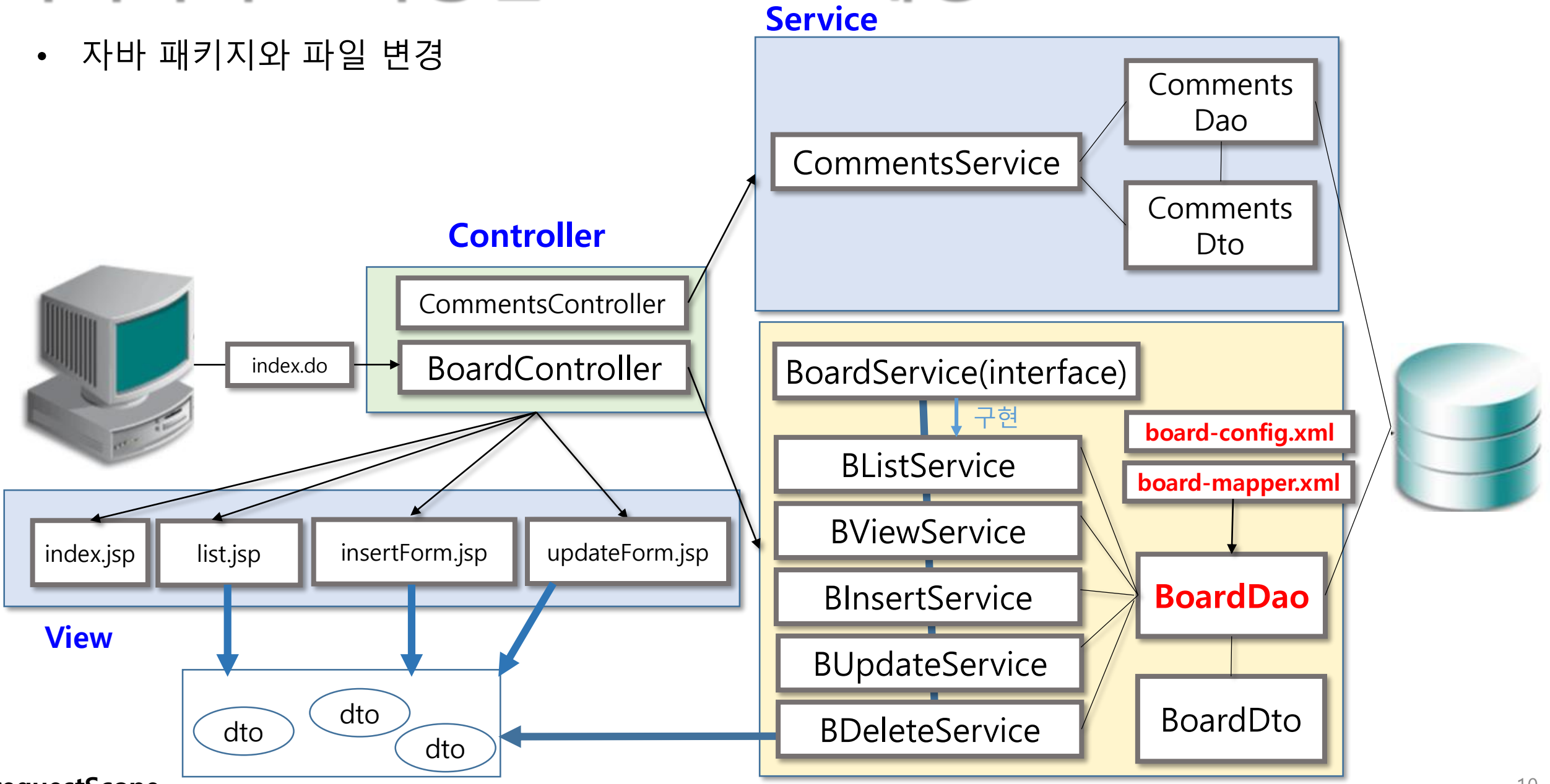
- 간단한 퍼시스턴스(영속객체) 프레임워크
- **SQL문**이 애플리케이션 소스 코드로부터 완전 **분리**
  - SQL을 소스 코드가 아닌 XML로 분리
- **생산성** : 62%정도 줄어드는 코드 , 간단한 설정
- **작업의 분배** : 팀을 세분화하는 것을 도움
- **이식성** : 어떤 프로그래밍 언어로도 구현 가능 (자바, C#, .NET, RUBY)
- **오픈소스**로 무료

Mybatis의 특징(상세)

- 쉬운 접근성과 코드의 **간결함**
  - 가장 간단한 퍼시스턴스(DB) 프레임워크
  - XML 형태로 서술된 JDBC코드라고 생각될 만큼 JDBC의 모든 기능을 제공
  - 복잡한 JDBC 코드를 걷어내어 깔끔한 소스코드 유지 가능
- **SQL문과 프로그래밍 코드의 분리**
  - SQL에 변경이 있어도 Java 코드 수정하고 다시 컴파일하지 않아도 됨
  - SQL 작성과 관리를 분리하여 DBA와 같은 개발자가 아닌 사람과 협업할 수 있음
- 다양한 프로그래밍 언어로 구현 가능
  - java, C#, .NET, Ruby
- mybatis 프레임워크는 **자체 커넥션풀 구축** 가능
- 여러 개의 DB연결정보를 설정해 두고 실행 상황(개발, 테스트, 운영)에 따라 DB 지정 가능
- 실행 성능을 높이기 위해 select 결과를 캐싱(임시저장)해 둘 수 있음
- SQL 맵퍼 파일에서 사용할 값 객체에 대해 별명(alias)를 부여할 수 있음

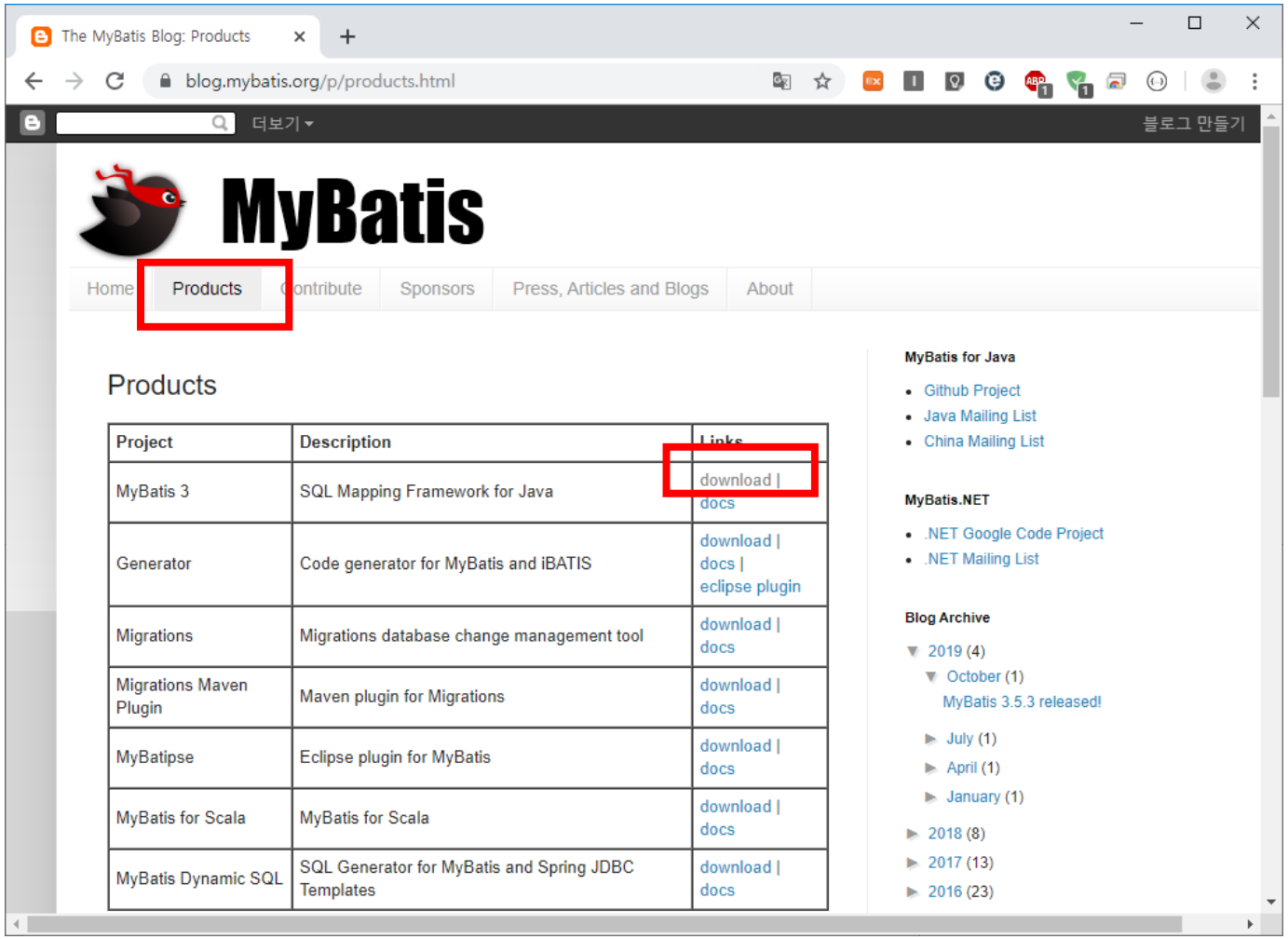
마이바티스 사용한 db프로그래밍

- 자바 패키지와 파일 변경



MyBatis 설치

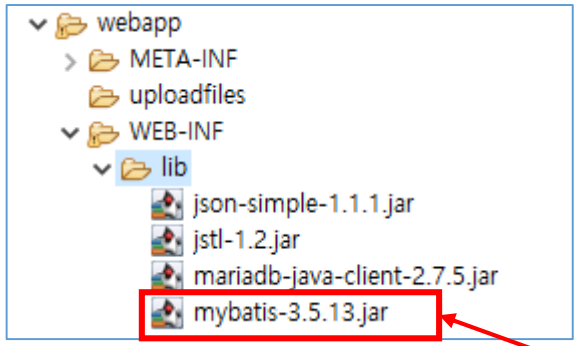
- http://mybatis.org

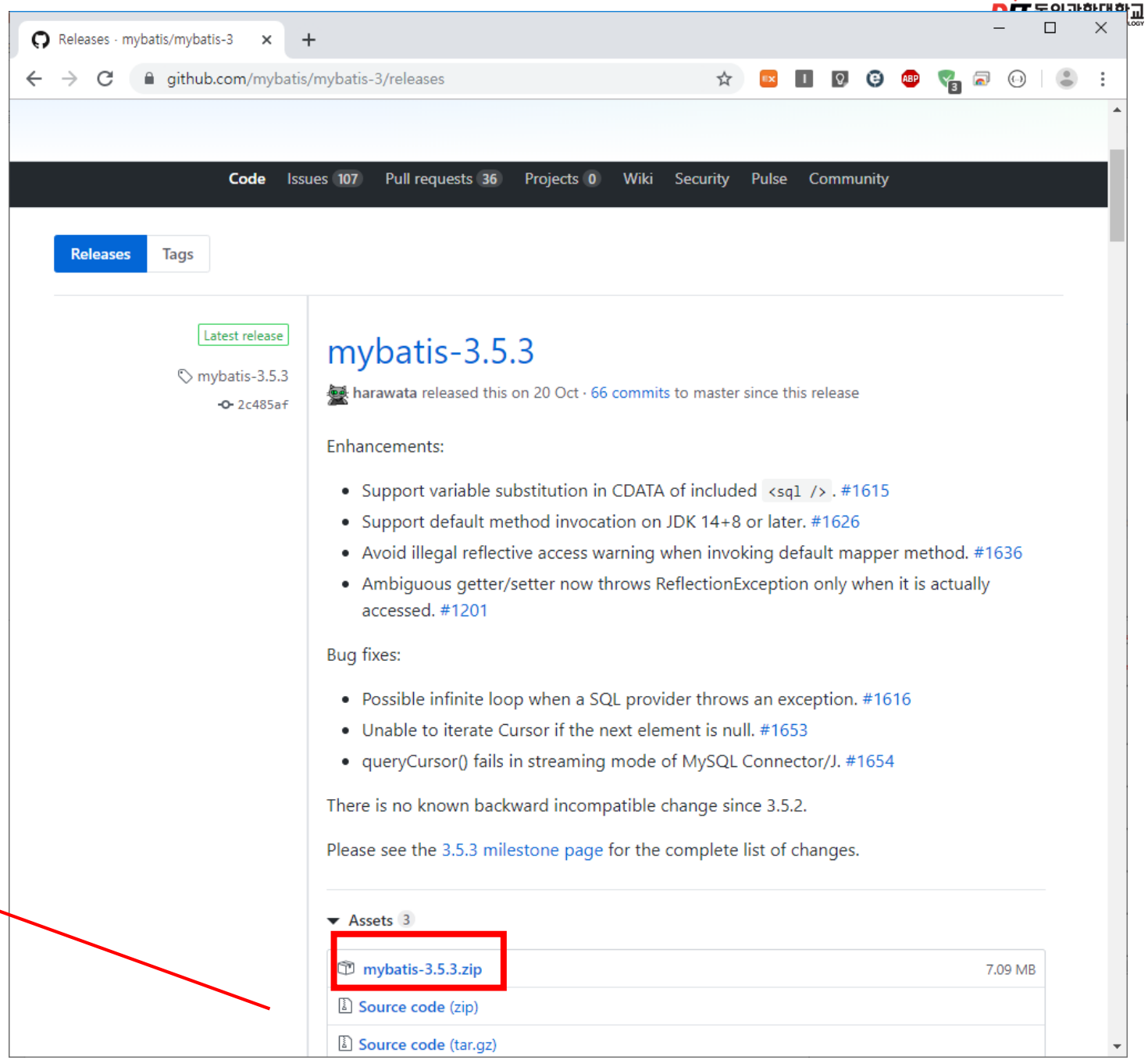


- mybatis 도움말 사이트
  - https://mybatis.org/mybatis-3/ko/index.html

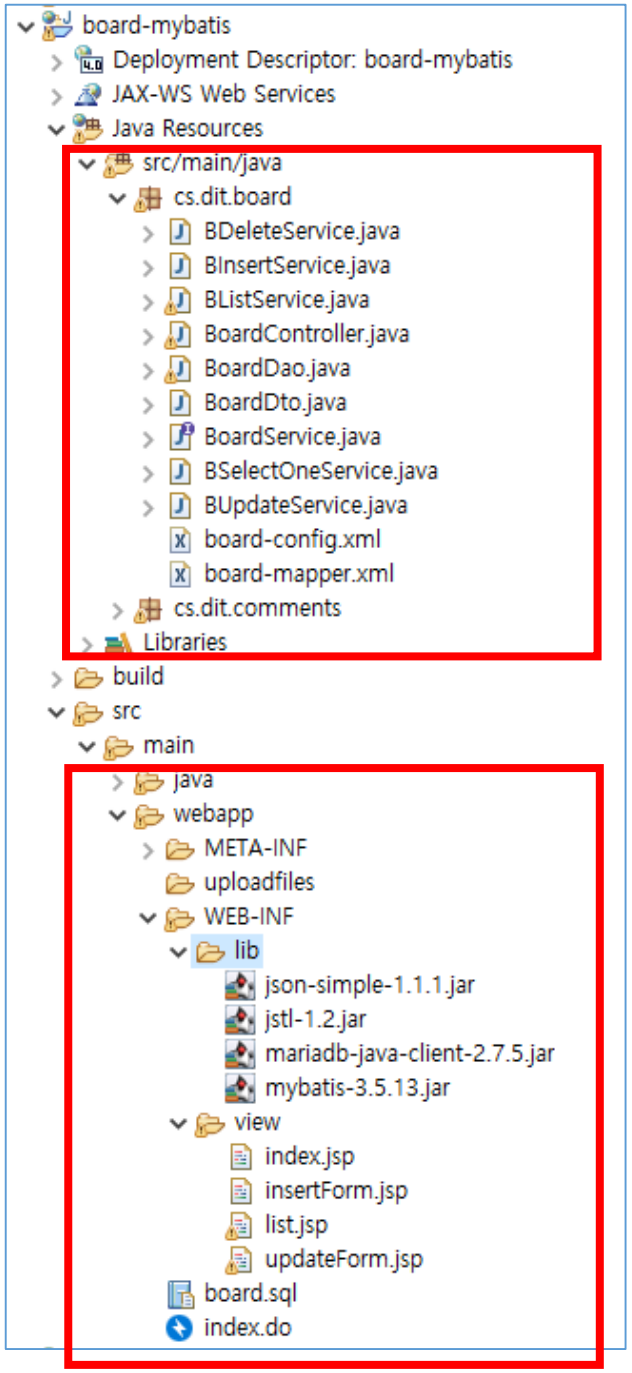
MyBatis 설치

- mybatis-3.5.3.zip 파일의 압축을 푼다.
- 이클립스의 WebContent-WEB-INF/lib 에 파일을 복사하여 붙여넣는다.

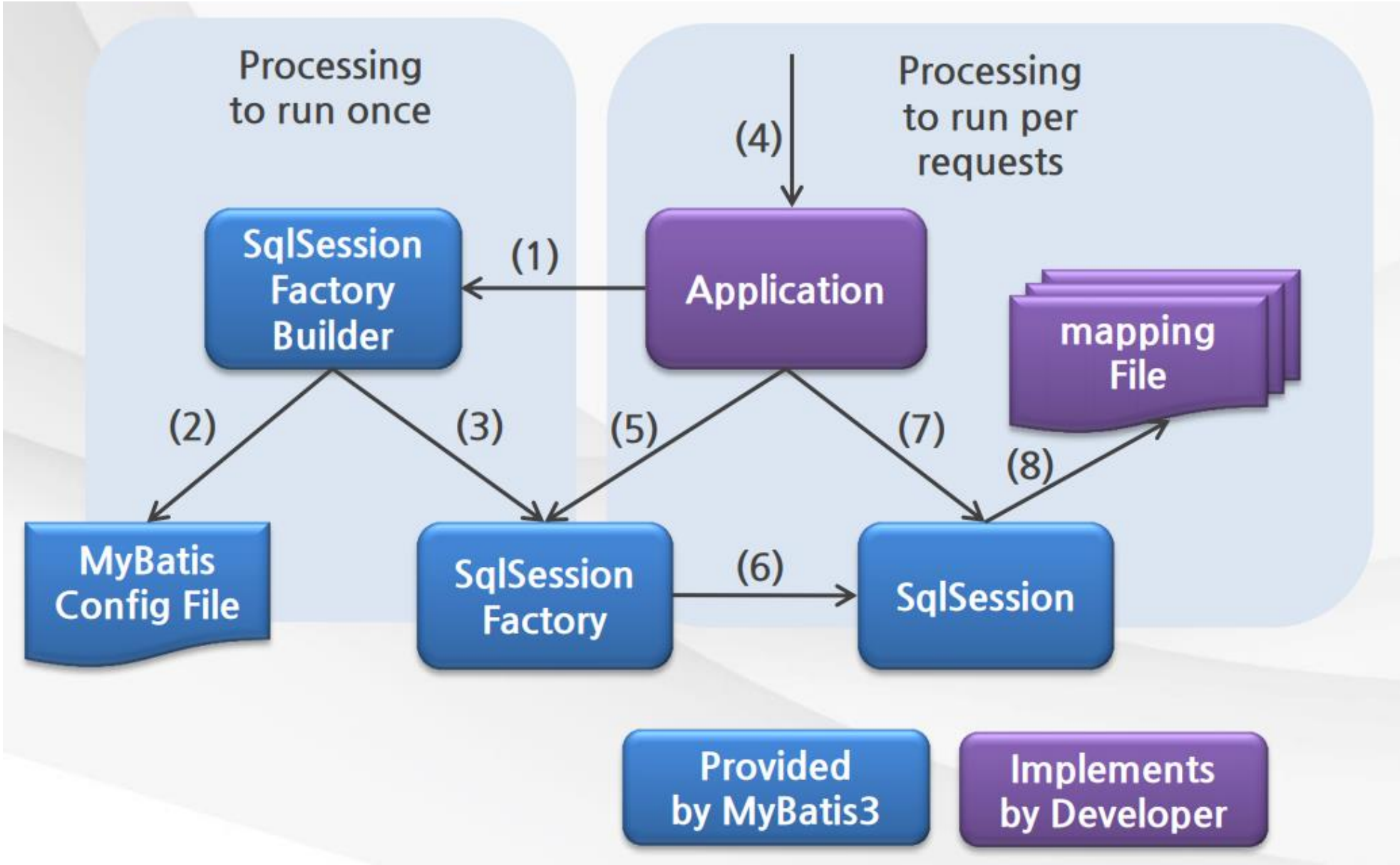




전체 파일 구조


board-mybatis
Deployment Descriptor: board-mybatis
JAX-WS Web Services
Java Resources
src/main/java
cs.dit.board
BDeleteService.java
BInsertService.java
BListService.java
BoardController.java
BoardDao.java
BoardDto.java
BoardService.java
BSelectOneService.java
BUpdateService.java
board-config.xml
board-mapper.xml
cs.dit.comments
Libraries
build
src
main
java
webapp
META-INF
uploadfiles
WEB-INF
lib
json-simple-1.1.1.jar
jstl-1.2.jar
mariadb-java-client-2.7.5.jar
mybatis-3.5.13.jar
view
index.jsp
insertForm.jsp
list.jsp
updateForm.jsp
board.sql
index.do

mybatis 구성 요소



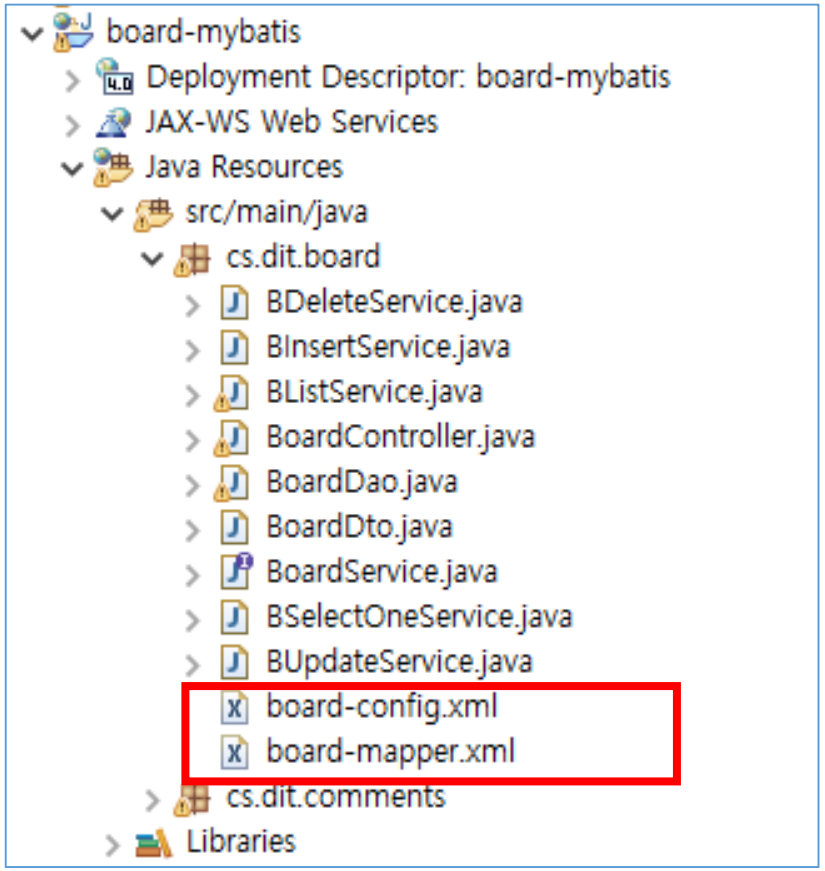
mybatis 설정 파일

설정 파일(파일명은 임의)	설명
설정 파일 (**board-config.xml**)	• DB 연결 정보 • 트랜잭션 정보 • mybatis 제어 정보 등의 설정 내용을 포함 • SqlSessionFactory를 작성할 때 사용
맵퍼 파일 (**board-mapper.xml**)	• SQL문을 담고 있는 파일 • SqlSession 객체가 참조

https://mybatis.org/mybatis-3/ko/getting-started.html

설정 파일 : board-config.xml

- 위치 : src 내 xml 파일로 작성
  - New – Other – XML - XML file에서 파일명을 mybatis-config.xml로 하여 작성
  - 파일명은 임의로 작성 가능(단, 기능과 의미를 생각해 작명할 것)



• <configuration> 루트 엘리먼트

엘리먼트	설명
properties	• 프로퍼티 파일이 있는 경로 설정 • <property>를 사용하여 개별 프로퍼티 정의 기능
settings	• 프레임워크의 실행 환경 설정
typeAliases	• 자바 클래스 이름(패키지명 포함)에 대한 별칭 설정 • SQL 맴퍼 파일에서 매개변수 타입(parameterType)이나 결과타입(resultType)을 지정할 때 긴 이름 대신 짧은 이름의 별명 사용 가능
typeHandlers	• 컬럼 값을 자바 객체로, 자바 객체를 컬럼의 값으로 변화하는 클래스 설정
environments	• DB 환경정보 설정 태그 • 프레임워크에서 사용할 DB정보(트랜잭션 관리자, 데이터소드) 설정 • 여러 개의 DB 접속 정보 설정
mappers	• SQL 맴버 파일이 있는 경로 설정

- mybatis 도움말 사이트
  https://mybatis.org/mybatis-3/ko/index.html

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "https://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias type="cs.dit.board.BoardDto" alias = "board"/>
  </typeAliases>

  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="org.mariadb.jdbc.Driver"/>
        <property name="url" value="jdbc:mariadb://localhost:3306/jinsookdb"/>
        <property name="username" value="jinsook"/>
        <property name="password" value="1111"/>
      </dataSource>
    </environment>
  </environments>

  <mappers>
    <mapper resource="cs/dit/board/board-mapper.xml"/>
  </mappers>                                    // 맴퍼파일을 지정(패키지명과 함께)
</configuration>
```

<environment> 엘리먼트

- 트랜잭션 관리 및 데이터 소스 설정 태그
- <transactionManager> 태그의 type 속성 값

트랜잭션 관리 유형	설명
JDBC	• 직접 jdbc의 commit, rollback 기능을 사용하여 mybatis 자체에서 트랜잭션 관리
MANAGED	• 서버의 트랜잭션 관리기능을 이용, 즉 JavaEE 애플리케이션 서버(서블릿 컨테이너 등)에서 트랜잭션 관리

mybatis가 connection pool을 운영함으로 dbcp 설정 필요 없음

<dataSource> 엘리먼트

- mybatis는 JDBC 표준 인터페이스인 javax.sql.DataSource 구현체를 이용하여 DB 커넥션을 다룸
- mybatis에서 가능한 데이터 소스 유형

트랜잭션 관리 유형	설명
UNPOOLED	• DB 커넷션을 요청할 때마다 매번 커넥션 객체를 생성한다. • 단순한 애플리케이션에 적합하다.
POOLED	• 미리 DB 커넥션 객체를 생성해 두고, 요청하면 즉시 반환한다 (mybatis 자체 커넥션 풀) • DB 연결 과정(초기화 등)이 없기 때문에 속도가 빠르다.
JNDI	• JavaEE 애플리케이션 서버나 서블릿 컨테이너에서 제공하는 데이터소스를 사용한다.

<mappers> 엘리먼트

- SQL 맵퍼 파일들의 정보 설정

```
<!-- 클래스 경로에 있을 경우 -->
<mappers>
    <mapper resource = "cs/dit/board/board-mapper.xml" />
</mappers>

<!-- 아니면 url 속성 사용 -->
<mappers>
    <mapper url = "file:///c:/conf/sql-mapper.xml" />
</mappers>
```

SQL 맵퍼 파일 : login-mapper.xml

- XML 선언

```
<?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> 루트 엘리먼트
  - namespace 속성 : 자바의 패키지와 같이 SQL문을 묶는 용도로 사용
- <select><insert><update><delete> 엘리먼트

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "https://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="cs.dit.board">

  <select id="selectOne" resultType="board">
    SELECT * FROM board WHERE bcode = #{bcode}
  </select>

  <select id="selectAll" resultType="board">
    SELECT * FROM board ORDER BY bcode desc LIMIT #{page}, #{numOfRecords}
  </select>

  <insert id="insert" parameterType="board">
    INSERT INTO board(SUBJECT, CONTENT, WRITER, REGDATE, filename)
    VALUES(#{subject}, #{content}, #{writer}, SYSDATE(), #{filename})
  </insert>

  <select id = "recordCount" resultType="int">
    SELECT COUNT(bcode) FROM board
  </select>

  <update id = "update" parameterType="board">
    UPDATE board SET subject = #{subject}, content = #{content},
        writer = #{writer}, regDate = SYSDATE(), filename= #{filename}
        WHERE bcode = #{bcode}
  </update>

  <delete id="delete" parameterType="int">
    DELETE FROM board WHERE bcode = #{bcode}
  </delete>

</mapper>
```

DAO 클래스

- 매퍼 파일의 설정이 적용되는 객체는 DAO 객체임
- mybatis의 핵심 컴포넌트

컴포넌트	설명	생성 순서
MyBatis 설정파일	• 데이터베이스 접속주소정보나 Mapping 파일경로 등의 고정된 환경정보를 설정한다.	0
mapping 파일	• SQL문과 Mapping을 설정한다.	
SqlSessionFactoryBuilder	• 설정 파일의 내용을 토대로 SqlSessionFactory 를 생성한다.	1
SqlSessionFactory	• SqlSession 객체를 생성한다(sqlMapper).	2
SqlSession	• 핵심적인 역할을 하는 클래스로 SQL실행이나 트랜잭션 관리를 실행한다. • Thread-safe 하지 않아 Thread 마다 필요에 따라 생성한다(request scope). • SQL 처리를 위해 JDBC 드라이버를 사용한다.	3

스코프(Scope) 와 생명주기(Lifecycle)

이제부터 다룰 스코프와 생명주기에 대해서 이해하는 것은 매우 중요하다. 스코프와 생명주기를 잘못 사용하는 것은 다양한 동시성 문제를 야기할 수 있다.

참고 **객체 생명주기와 의존성 삽입 프레임워크**

의존성 삽입 프레임워크는 쓰레드에 안전하도록 해준다. 트랜잭션 성질을 가지는 SqlSessions과 매퍼들 그리고 그것들을 직접 빈에 삽입하면 생명주기에 대해 기억하지 않아도 되게 해준다. DI프레임워크와 마이바티스를 사용하기 위해 좀더 많은 정보를 보기 위해서는 MyBatis-Spring이나 MyBatis-Guice 하위 프로젝트를 찾아보면 된다.

SqlSessionFactoryBuilder

이 클래스는 인스턴스화되어 사용되고 던져질 수 있다. SqlSessionFactory 를 생성한 후 유지할 필요는 없다. 그러므로 SqlSessionFactoryBuilder 인스턴스의 가장 좋은 스코프는 메소드 스코프(예를들면 메소드 지역변수)이다. 여러개의 SqlSessionFactory 인스턴스를 빌드하기 위해 SqlSessionFactoryBuilder를 재사용할 수도 있지만 유지하지 않는 것이 가장 좋다.

SqlSessionFactory

한번 만든뒤 SqlSessionFactory는 애플리케이션을 실행하는 동안 존재해야만 한다. 그래서 삭제하거나 재생성할 필요가 없다. 애플리케이션이 실행되는 동안 여러 차례 SqlSessionFactory 를 다시 빌드하지 않는 것이 가장 좋은 형태이다. 재빌드하는 형태는 결과적으로 "나쁜냄새" 가 나도록 한다. 그러므로 SqlSessionFactory 의 가장 좋은 스코프는 애플리케이션 스코프이다. 애플리케이션 스코프로 유지하기 위한 다양한 방법이 존재한다. 가장 간단한 방법은 싱글턴 패턴이나 static 싱글턴 패턴을 사용하는 것이다. 또는 구글 쥬스나 스프링과 같은 의존성 삽입 컨테이너를 선호할 수도 있다. 이러한 프레임워크는 SqlSessionFactory의 생명주기를 싱글턴으로 관리할 것이다.

SqlSession

각각의 쓰레드는 자체적으로 SqlSession인스턴스를 가져야 한다. SqlSession인스턴스는 공유되지 않고 쓰레드에 안전하지도 않다. 그러므로 가장 좋은 스코프는 요청 또는 메소드 스코프이다. SqlSession 을 static 필드나 클래스의 인스턴스 필드로 지정해서는 안된다. 그리고 서블릿 프레임워크의 HttpSession 과 같은 관리 스코프에 둬서도 안된다. 어떠한 종류의 웹 프레임워크를 사용한다면 HTTP 요청과 유사한 스코프에 두는 것으로 고려해야 한다. 달리 말해서 HTTP 요청을 받을때마다 만들고 응답을 리턴할때마다 SqlSession 을 닫을 수 있다. SqlSession 을 닫는 것은 중요하다. 언제나 finally 블록에서 닫아야만 한다. 다음은 SqlSession을 닫는 것을 확인하는 표준적인 형태다.

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  // do work
}
```

코드전반에 이런 형태를 사용하는 것은 모든 데이터베이스 자원을 잘 닫는 것으로 보장하게 할 것이다.

Mapper 인스턴스

Mapper는 매핑된 구문을 바인딩 하기 위해 만들어야 할 인터페이스이다. mapper 인터페이스의 인스턴스는 SqlSession 에서 생성한다. 그래서 mapper 인스턴스의 가장 좋은 스코프는 SqlSession 과 동일하다. 어쨌든 mapper 인스턴스의 가장 좋은 스코프는 메소드 스코프이다. 사용할 메소드가 호출되면 생성되고 끝난다. 명시적으로 닫을 필요는 없다.

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  BlogMapper mapper = session.getMapper(BlogMapper.class);
  // do work
}
```

SqlSessionFactoryBuilder

- SqlSessionFactory 를 생성한 후 계속 유지할 필요 없음
- 그러므로 SqlSessionFactoryBuilder 인스턴스의 가장 좋은 스코프는 메소드 스코프(예를들면 메소드 지역변수)이다. 여러 개의 SqlSessionFactory 인스턴스를 빌드하기 위해 SqlSessionFactoryBuilder를 재사용할 수도 있지만 유지하지 않는 것이 가장 좋다.
- SqlSessionFactoryBuilder의 주요 메서드

주요 메서드	설명
build()	• mybatis-config.xml 설정 파일을 로딩하여 SqlSessionFactory 객체를 생성

SqlSessionFactory

- SqlSessionFactory는 애플리케이션을 실행하는 동안 유효해야 함
- SqlSessionFactory는 스코프는 애플리케이션 스코프에 저장해야 함
  - static 싱글톤으로 생성
- SqlSessionFactory의 주요 메서드

주요 메서드	설명
openSession()	• SqlSession 객체에 대한 팩토리 객체로 이 메소드를 통해 SqlSession 객체를 얻을 수 있음 • SqlSession 객체는 실제 member-mapper.xml 의 SQL문을 호출하는데 사용됨

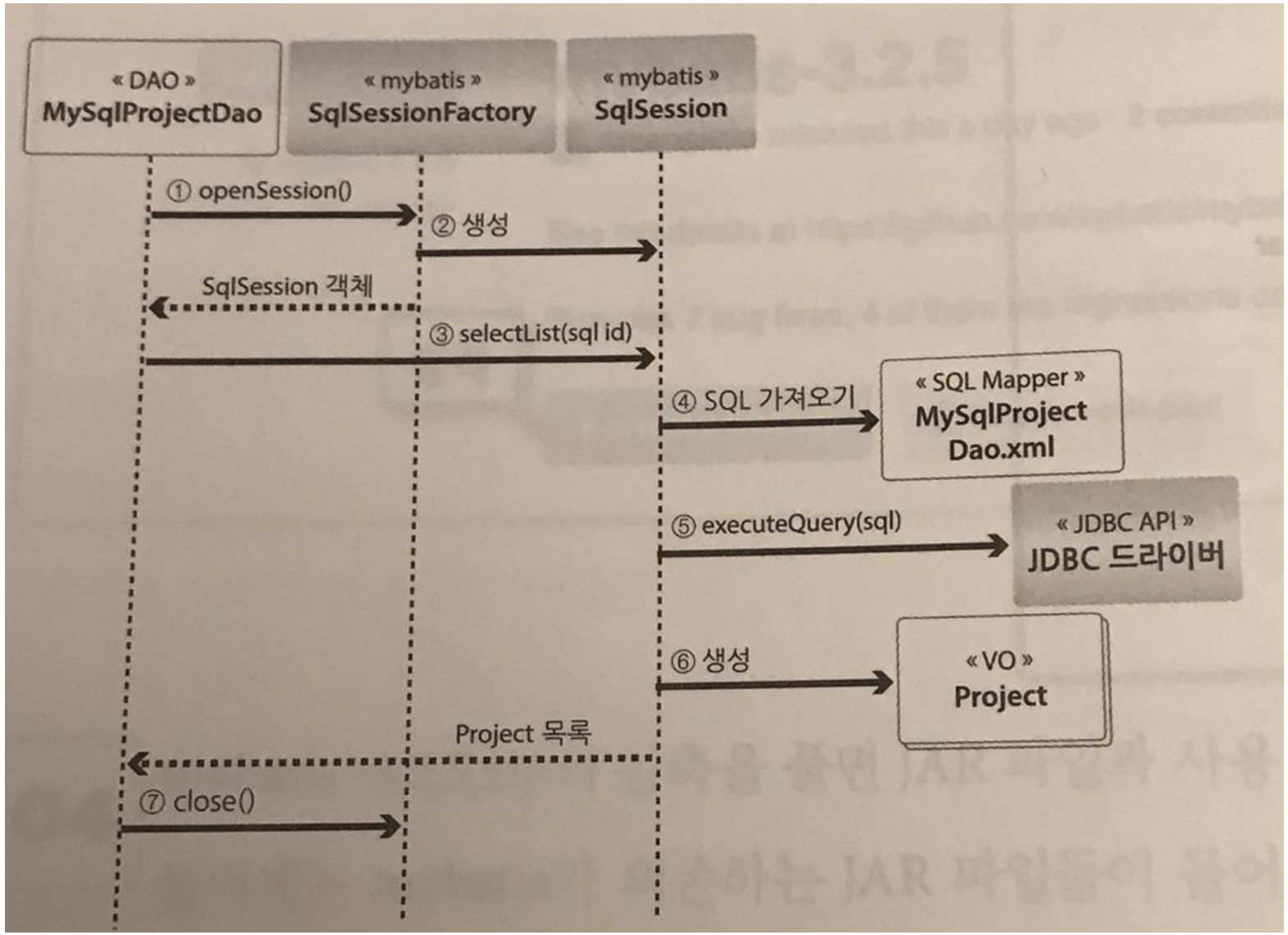
싱글톤 패턴
- 단 하나의 인스턴스를 생성해 사용하는 디자인 패턴이다.
- 인스턴스가 필요 할 때 똑같은 인스턴스를 만들어 내는 것이 아니라, 동일(기존) 인스턴스를 사용하게 함

SqlSession

- SqlSession인스턴스는 공유되지 않고 쓰레드에 안전하지도 않음
- SqlSession 은 지역변수로 지정해야 함
- SqlSession 을 닫는 것은 중요함으로 finally 블록에서 닫음
- SqlSession의 주요 메서드

주요 메서드	설명
List **selectList**(String stmt, Object param)	• SELECT 문 실행. VO 객체 목록을 반환
Object **selectOne**(String stmt, Object param)	• SELECT 문 실행. 하나의 VO 객체 반환
int **insert**(String stmt, Object param)	• INSERT 문 실행. 반환값은 입력한 레코드의 갯수
int **update**(String stmt, Object param)	• UPDATE 문 실행. 반환값은 수정한 레코드의 갯수
int **delete**(String stmt, Object param)	• DELETE 문 실행. 반환값은 삭제한 레코드의 갯수

Mybatis 동작 과정



BoardDao.java 변경

```java
package cs.dit.board;

import java.io.InputStream;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

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 BoardDao {

    private static SqlSessionFactory sqlMapper = null;

    public static SqlSessionFactory getInstance() {
        if(sqlMapper==null) {
            try {
                InputStream inputStream = Resources.getResourceAsStream("cs/dit/board/board-config.xml");
                sqlMapper = new SqlSessionFactoryBuilder().build(inputStream);
                inputStream.close();
            }catch(Exception e) {
                e.printStackTrace();
            }
        }

        return sqlMapper;
    }
```

목록보기 / 목록 자세히 보기

```java
public BoardDto selectOne(int bcode) {
    sqlMapper = getInstance();
    SqlSession session = sqlMapper.openSession();
    try {
        return session.selectOne("cs.dit.board.selectOne", bcode);
    }finally {
        session.close();
    }
}

public List<BoardDto> list(int page, int numOfRecords){
    sqlMapper = getInstance();

    SqlSession session = sqlMapper.openSession();
    Map<String, Object> param = new HashMap<>();
    param.put("page", (page-1)*numOfRecords);
    param.put("numOfRecords", numOfRecords);

    System.out.println("sqlMapper 사용!!!");
    try {
        return session.selectList("cs.dit.board.selectAll", param);
    }finally {
        session.close();
    }
}
public int recordCount() {
    sqlMapper = getInstance();
    SqlSession session = sqlMapper.openSession();
    try {
        return session.selectOne("cs.dit.board.recordCount");
    }finally {
        session.close();
    }
}
```

```xml
<mapper namespace="cs.dit.board">

  <select id="selectOne" resultType="board">
    SELECT * FROM board WHERE bcode = #{bcode}
  </select>

  <select id="selectAll" resultType="board">
    SELECT * FROM board ORDER BY bcode desc LIMIT #{page}, #{numOfRecords}
  </select>

  <select id = "recordCount" resultType="int">
    SELECT COUNT(bcode) FROM board
  </select>

  <insert id="insert" parameterType="board">
    INSERT INTO board(SUBJECT, CONTENT, WRITER, REGDATE, filename)
    VALUES(#{subject}, #{content}, #{writer}, SYSDATE(), #{filename})
  </insert>

  <update id = "update" parameterType="board">
    UPDATE board SET subject = #{subject}, content = #{content},
        writer = #{writer}, regDate = SYSDATE(), filename= #{filename}
        WHERE bcode = #{bcode}
  </update>

  <delete id="delete" parameterType="int">
    DELETE FROM board WHERE bcode = #{bcode}
  </delete>

</mapper>
```

데이터 입력 / 데이터 변경

```java
68    public void insert(BoardDto dto) {
69        sqlMapper = getInstance();
70        SqlSession session = sqlMapper.openSession();
71
72        try {
73            session.insert("cs.dit.board.insert", dto);
74            session.commit();
75        }finally {
76            session.close();
77        }
78    }
79
80    public void update(BoardDto dto) {
81        sqlMapper = getInstance();
82        SqlSession session = sqlMapper.openSession();
83        try {
84            session.update("cs.dit.board.update", dto);
85            session.commit();
86            System.out.println("dao의 update");
87        }finally {
88            session.close();
89        }
90
91    }
92    public void delete(int bcode) {
93        sqlMapper = getInstance();
94        SqlSession session = sqlMapper.openSession();
95        try {
96            session.delete("cs.dit.board.delete", bcode);
97            session.commit();
98        }finally {
99            session.close();
100       }
101
102   }
```

```xml
6  <mapper namespace="cs.dit.board">
7
8    <select id="selectOne" resultType="board">
9      SELECT * FROM board WHERE bcode = #{bcode}
10   </select>
11
12   <select id="selectAll" resultType="board">
13     SELECT * FROM board ORDER BY bcode desc LIMIT #{page}, #{numOfRecords}
14   </select>
15
16   <select id = "recordCount" resultType="int">
17     SELECT COUNT(bcode) FROM board
18   </select>
19
20   <insert id="insert" parameterType="board">
21     INSERT INTO board(SUBJECT, CONTENT, WRITER, REGDATE, filename)
22     VALUES(#{subject}, #{content}, #{writer}, SYSDATE(), #{filename})
23   </insert>
24
25   <update id = "update" parameterType="board">
26     UPDATE board SET subject = #{subject}, content = #{content},
27         writer = #{writer}, regDate = SYSDATE(), filename= #{filename}
28         WHERE bcode = #{bcode}
29   </update>
30
31   <delete id="delete" parameterType="int">
32     DELETE FROM board WHERE bcode = #{bcode}
33   </delete>
34
35 </mapper>
```