

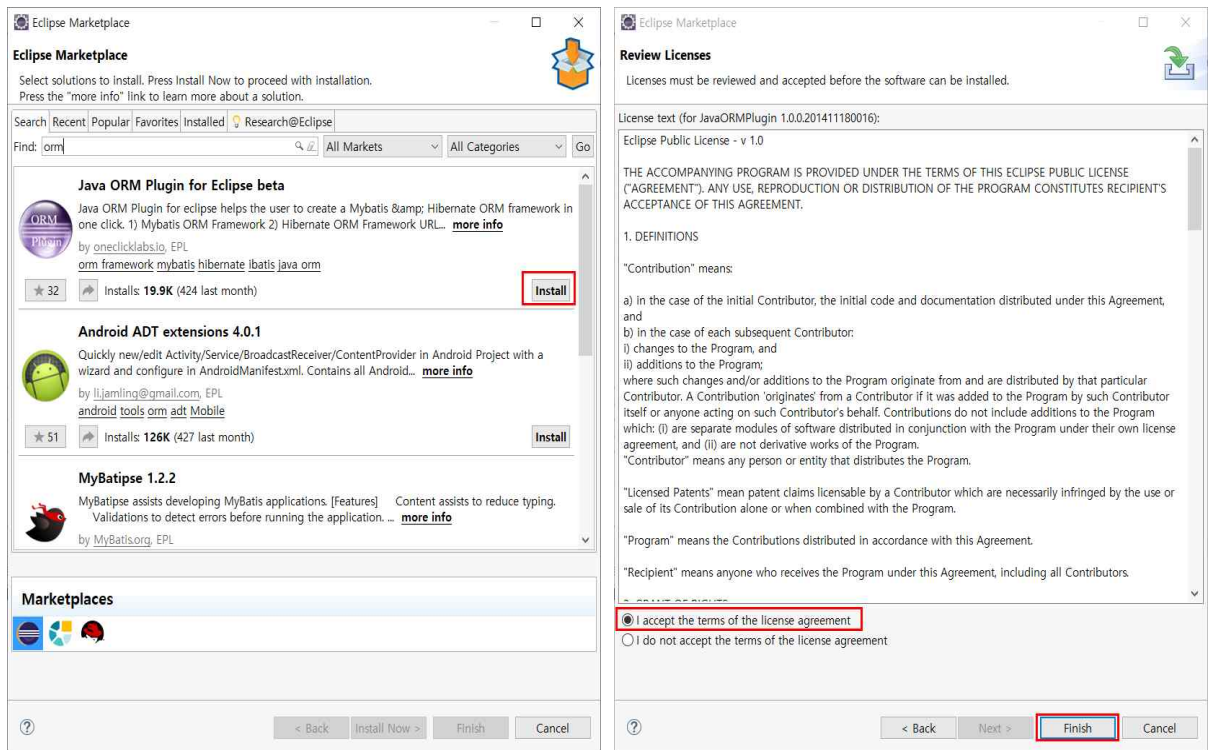
제 14 강 MyBatis

1. MyBatis 프레임워크의 특징

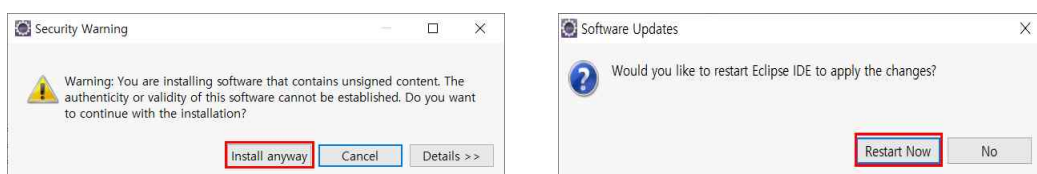
- MyBatis 프레임워크의 특징을 보면 첫째는 한두 줄의 자바 코드로 DB 연동을 처리한다는 점이며 둘째는 SQL 명령어를 자바 코드에서 분리하여 XML 파일로 따로 관리한다는 것이다.
- 사실 유지보수 관점에서 보면 DB연동에 사용된 복잡한 자바 코드는 더 이상 중요하지 않다. 개발자는 실행되는 SQL만 관리하면 되며 MyBatis는 개발자가 이 SQL 관리에만 집중할 수 있도록 도와준다.
- 만약 SQL 명령어가 DAO 같은 자바 클래스에 저장되면 SQL 명령어만 수정하는 상황에서도 자바 클래스를 다시 컴파일 해야 한다. 그리고 SQL 명령어들을 한 곳에 모아서 관리하기도 쉽지 않다. 결국 SQL 명령어에 대한 통합 관리를 위해서라도 자바 소스코드에서 SQL을 분리하는 것은 매우 중요하다.

2. Java ORM Plugin 설치

- 이클립스에서 [Help]-[Eclipse Marketplace...] 메뉴를 선택한 후 검색창에 “orm” 을 입력해 보자. 이때 Java ORM Plugin for Eclipse beta를 찾아 [install] 한다.
- 사용권 계약 동의서가 나오면 동의를 선택하고 [Finish] 버튼을 클릭한다.



- 만약 보안 경고가 뜬다면 무시하고 설치를 진행한다. 설치가 완료되면 이클립스를 재구동하라는 메시지가 나온다. [Restart Now]를 클릭하여 이클립스를 재 구동한다.

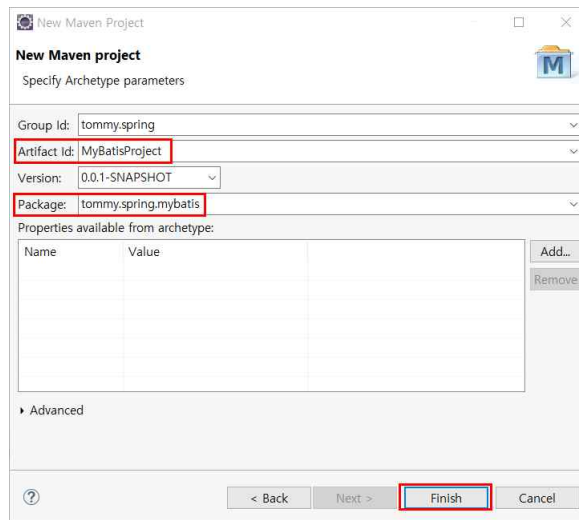


3. MyBatis 프로젝트

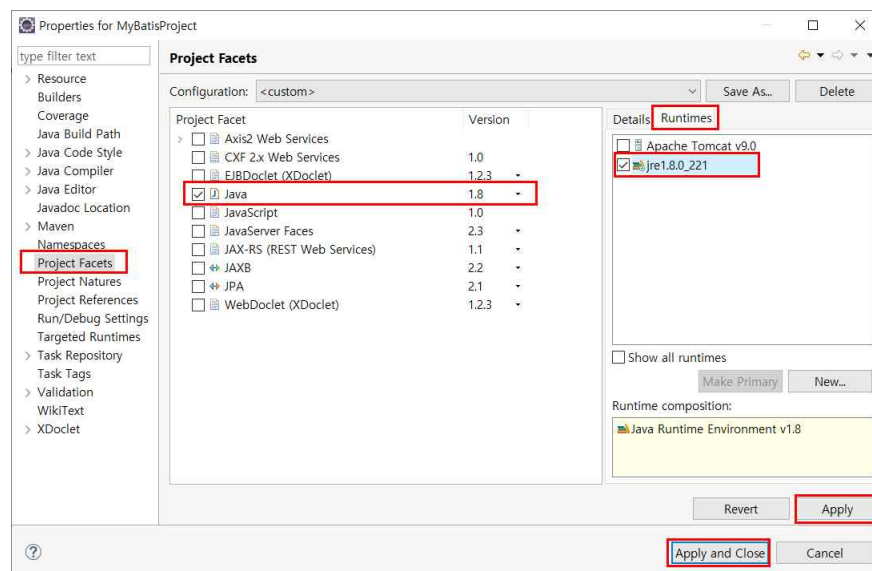
- MyBatis의 개념을 이해하기 위하여 간단한 CRUD 프로그램을 작성해 보자.

① Maven Project 생성

- Eclipse에서 [File]-[New] 메뉴를 선택하고 [Maven]-[Maven Project]를 선택한 후 아래와 같이 “MyBatisProject”를 생성하도록 한다.



- Maven 프로젝트 설정 시 기본 Java의 사용은 1.5로 되어있습니다. java 1.5를 1.8 버전으로 변경해주는 작업이 필요합니다. MyBatisProject 선택 후 우 클릭하여 [Properties] 메뉴 선택 후 아래와 같이 진행



② 의존관계 추가

- 메이븐 설정 스프링 의존관계를 추가한다.

	<!-- 상단 부분 생략 -->
1	<dependencies>
2	<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
3	<dependency>
4	<groupId>org.springframework</groupId>

```

5         <artifactId>spring-context</artifactId>
6         <version>5.1.9.RELEASE</version>
7     </dependency>
8     <dependency>
9         <groupId>junit</groupId>
10        <artifactId>junit</artifactId>
11        <version>4.12</version>
12        <scope>test</scope>
13    </dependency>
14</dependencies>
15<build>
16    <plugins>
17        <plugin>
18            <groupId>org.apache.maven.plugins</groupId>
19            <artifactId>maven-compiler-plugin</artifactId>
20            <version>3.6.2</version>
21            <configuration>
22                <source>1.8</source>
23                <target>1.8</target>
24                <encoding>UTF-8</encoding>
25            </configuration>
26        </plugin>
27    </plugins>
28</build>
29</project>

```

■ 데이터베이스 연동 관련 라이브러리와 Logging 관련 라이브러리를 추가한다.

```

<!-- 상단 부분 생략 -->
1     <dependencies>
2         <!-- 오라클 드라이버 -->
3         <dependency>
4             <groupId>com.oracle</groupId>
5             <artifactId>ojdbc8</artifactId>
6             <version>18.3</version>
7         </dependency>
8         <!-- MyBatis -->
9         <dependency>
10            <groupId>org.mybatis</groupId>
11            <artifactId>mybatis</artifactId>
12            <version>3.5.2</version>
13        </dependency>
14        <!-- commons-logging -->
15        <dependency>
16            <groupId>commons-logging</groupId>
17            <artifactId>commons-logging</artifactId>
18            <version>1.2</version>
19        </dependency>
<!-- 하단 부분 생략 -->

```

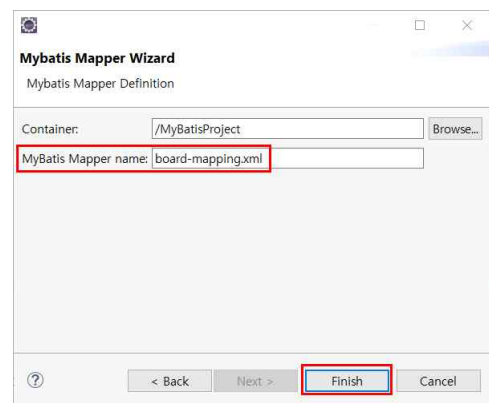
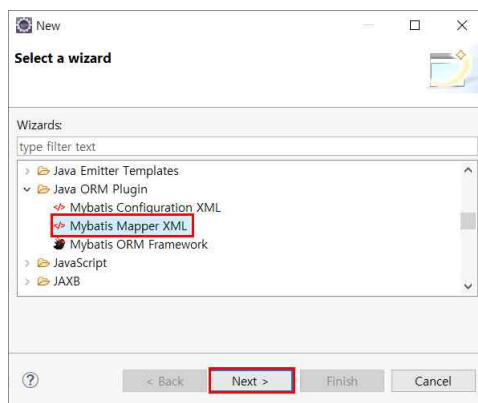
③ VO 클래스 작성

- ☐ 기본적으로 생성된 src/main/java 폴더 안에 tommy.spring.mybatis 패키지를 삭제한다.
- ☐ src/main/test 폴더 안에 tommy.spring.mybatis 패키지를 삭제한다.
- ☐ 만약 MyBatisProject에 에러 마크가 있다면 Maven Update를 한 번 수행해 준다.
- tommy.spring.web.board 패키지를 생성하고 BoardVO 클래스를 작성한다.

```
1 package tommy.spring.web.board;
2 import java.util.Date;
3 public class BoardVO {
4     private int seq;
5     private String title;
6     private String writer;
7     private String content;
8     private Date regDate;
9     private int cnt;
10    private String searchCondition;
11    private String searchKeyword;
12    // getter, setter 추가
13    // Generate toString ... 추가
14 }
```

④ SQL Mapper XML 작성

- ☐ SQL Mapper XML은 MyBatis에서 가장 중요한 파일이다. 이 XML에 DB 연동에 필요한 SQL 명령어들이 저장되기 때문이다.
- ☐ MyBatisProject를 선택한 후에 우 클릭하여 [New]-[Other]를 선택한다.
- ☐ “Java ORM Plugin” 폴더를 선택 후 “MyBatis Mapper XML”을 선택한 후 [Next] 버튼을 클릭.
- ☐ “MyBatis Mapper name” 에 “board-mapping.xml”을 입력하고 [Finish] 버튼을 클릭한다.



- ☐ src/test라는 폴더에 board-mapping.xml 파일이 생성될 것이다. src/main/resources/mappings 폴더를 생성하고 그 안으로 board-mapping.xml 파일을 이동시킨다.

■ board-mapping.xml 파일 작성

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

```

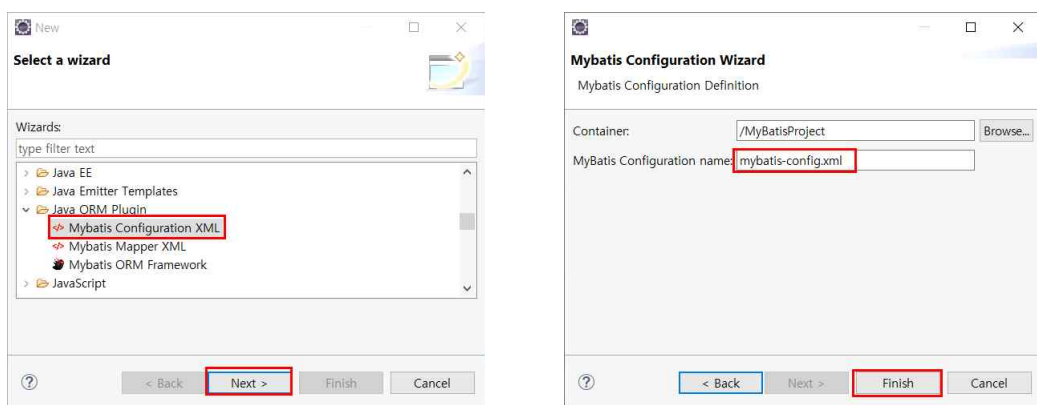
4 <mapper namespace="BoardDAO">
5     <insert id="insertBoard" parameterType="myboard">
6         INSERT INTO myboard (seq, title, writer, content)
7         VALUES((select nvl(max(seq), 0)+1 from myboard), #{title}, #{writer}, #{content})
8     </insert>
9     <update id="updateBoard" parameterType="myboard">
10        UPDATE myboard
11        SET title=#{title}, content=#{content}
12        WHERE seq = #{seq}
13    </update>
14    <delete id="deleteBoard" parameterType="myboard">
15        DELETE FROM myboard WHERE seq = #{seq}
16    </delete>
17    <select id="getBoard" resultType="myboard" parameterType="myboard">
18        SELECT * FROM myboard WHERE seq=#{seq}
19    </select>
20    <select id="getBoardList" resultType="myboard" parameterType="myboard">
21        SELECT * FROM myboard
22        WHERE title LIKE '%| |#{searchKeyword}| |%'
23        ORDER BY seq DESC
24    </select>
25 </mapper>

```

- ☐ SQL Mapper 파일은 <mapper>를 루트 엘리먼트로 사용한다. 그리고 <insert>, <update>, <delete>, <select> 엘리먼트를 이용하여 필요한 SQL문을 등록한다.

⑤ MyBatis 환경 설정 파일

- ☐ MyBatisProject를 선택한 후 마우스 우 클릭하여 [New]-[Other] 메뉴를 선택한다.
- ☐ “Java ORM Plugin” 폴더에서 “Mybatis Configuration XML” 을 선택한 후 [Next] 버튼을 클릭.
- ☐ “Mybatis Configuration name” 에 “mybatis-config.xml” 을 입력 후 [Finish] 버튼을 클릭한다.



- ☐ myabtis-config.xml 파일과 db.properties 파일을 src/main/resources 폴더로 이동시킨다.

■ db.properties 파일 수정

```
jdbc.driverClassName=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@localhost:1521/XEPDB1
jdbc.username=mytest
jdbc.password=mytest
```

■ mybatis-config.xml 파일 수정

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3   PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <properties resource="db.properties" />
7     <typeAliases>
8         <typeAlias alias="myboard" type="tommy.spring.web.board.BoardVO" />
9     </typeAliases>
10    <environments default="development">
11        <environment id="development">
12            <transactionManager type="JDBC" />
13            <dataSource type="POOLED">
14                <property name="driver" value="${jdbc.driverClassName}" />
15                <property name="url" value="${jdbc.url}" />
16                <property name="username" value="${jdbc.username}" />
17                <property name="password" value="${jdbc.password}" />
18            </dataSource>
19        </environment>
20    </environments>
21    <mappers>
22        <mapper resource="mappings/board-mapping.xml" />
23    </mappers>
24 </configuration>
```

- ☐ **<properties>** 엘리먼트는 XML 설정에서 사용할 프로퍼티를 선언하거나 외부 프로퍼티 파일을 참조할 때 사용한다. 이렇게 선언된 프로퍼티는 **`${프로퍼티이름}`**으로 참조하여 사용할 수 있다.
- ☐ **<typeAliases>** 엘리먼트는 **<typeAlias>**를 여러 개 가질 수 있으며 **<typeAlias>**를 이용하여 특정 클래스의 별칭(Alias)을 선언할 수 있다. 이 Alias는 SQL 명령어들이 저장되는 SQL Mapper에서 사용할 수 있으며 이를 통해 SQL Mapper 파일의 크기를 줄여주고 설정을 간단히 처리할 수 있도록 해 준다.
- ☐ **<mappers>** 엘리먼트는 여러 **<mapper>**를 가질 수 있으며 이 **<mapper>**를 이용하여 SQL 명령어들이 저장된 SQL 파일을 등록할 수 있다.

⑥ SqlSession 객체 생성하기

- ☐ MyBatis를 이용하여 DAO를 구현하려면 SqlSession 객체가 필요하다. 그런데 이 SqlSession 객체를 얻으려면 SqlSessionFactory 객체가 필요하다.

■ SqlSessionFactoryBean 클래스 작성

```
1 package tommy.spring.web.util;
2 import java.io.Reader;
3 import org.apache.ibatis.io.Resources;
4 import org.apache.ibatis.session.SqlSession;
5 import org.apache.ibatis.session.SqlSessionFactory;
6 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
7 public class SqlSessionFactoryBean {
8     private static SqlSessionFactory sessionFactory = null;
9     static {
10         try {
11             if(sessionFactory == null) {
12                 Reader reader = Resources.getResourceAsReader(
13                     "mybatis-config.xml");
14                 sessionFactory = new SqlSessionFactoryBuilder().build(reader);
15             }
16         } catch(Exception e) {
17             e.printStackTrace();
18         }
19     }
20     public static SqlSession getSqlSessionInstance() {
21         return sessionFactory.openSession();
22     }
23 }
```

⑦ DAO 클래스 작성

■ BoardDAO 클래스 작성

```
1 package tommy.spring.web.board.impl;
2 import java.util.List;
3 import org.apache.ibatis.session.SqlSession;
4 import tommy.spring.web.board.BoardVO;
5 import tommy.spring.web.util.SqlSessionFactoryBean;
6 public class BoardDAO {
7     private SqlSession sqlSession;
8     public BoardDAO() {
9         sqlSession = SqlSessionFactoryBean.getSqlSessionInstance();
10    }
11    public void insertBoard(BoardVO vo) {
12        sqlSession.insert("BoardDAO.insertBoard", vo);
13    }
14    public void updateBoard(BoardVO vo) {
15        sqlSession.update("BoardDAO.updateBoard", vo);
16    }
17    public void deleteBoard(BoardVO vo) {
18        sqlSession.delete("BoardDAO.deleteBoard", vo);
19    }
20    public BoardVO getBoard(BoardVO vo) {
```

```

21         return (BoardVO) sqlSession.selectOne("BoardDAO.getBoard", vo);
22     }
23     public List<BoardVO> getBoardList(BoardVO vo) {
24         return sqlSession.selectList("BoardDAO.getBoardList", vo);
25     }
26 }

```

- ☐ 위의 코드에서 구현된 각 메서드를 보면 두 개의 정보가 인자로 전달되고 있는데, 첫 번째 인자는 실행될 SQL의 id 정보이다. 이때 SQL Mapper에 선언된 네임스페이스와 아이디를 조합하여 아이디를 지정해야 한다.
- ☐ 그리고 두 번째 인자는 `parameterType` 속성으로 지정된 파라미터 객체이다.
- ☐ 등록, 수정, 삭제는 각각 `insert()`, `update()`, `delete()` 메서드로 처리하며 단 한건의 조회는 `selectOne()` 목록 조회는 `selectList()` 메서드로 처리한다.

⑧ 테스트 클라이언트 작성

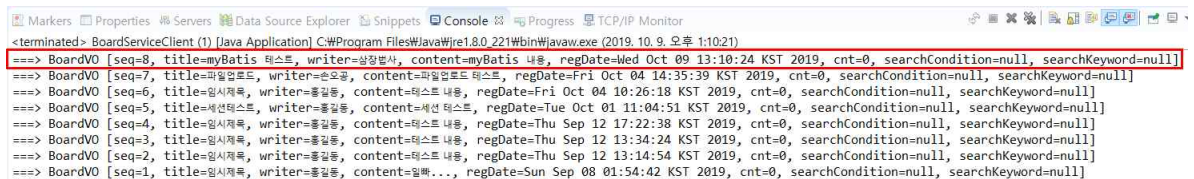
- `src/test/java` 폴더 안에 `BoardServiceClient` 클래스 작성

```

1 package tommy.spring.web.board;
2 import java.util.List;
3 import tommy.spring.web.board.impl.BoardDAO;
4 public class BoardServiceClient {
5     public static void main(String[] args) {
6         BoardDAO boardDAO = new BoardDAO();
7         BoardVO vo = new BoardVO();
8         vo.setTitle("myBatis 테스트");
9         vo.setWriter("삼장법사");
10        vo.setContent("myBatis 내용");
11        boardDAO.insertBoard(vo);
12        vo.setSearchCondition("TITLE");
13        vo.setSearchKeyword("");
14        List<BoardVO> boardList = boardDAO.getBoardList(vo);
15        for (BoardVO board : boardList) {
16            System.out.println("==> " + board.toString());
17        }
18    }
19 }

```

⑨ BoardServiceClient를 실행하여 결과를 확인하자.



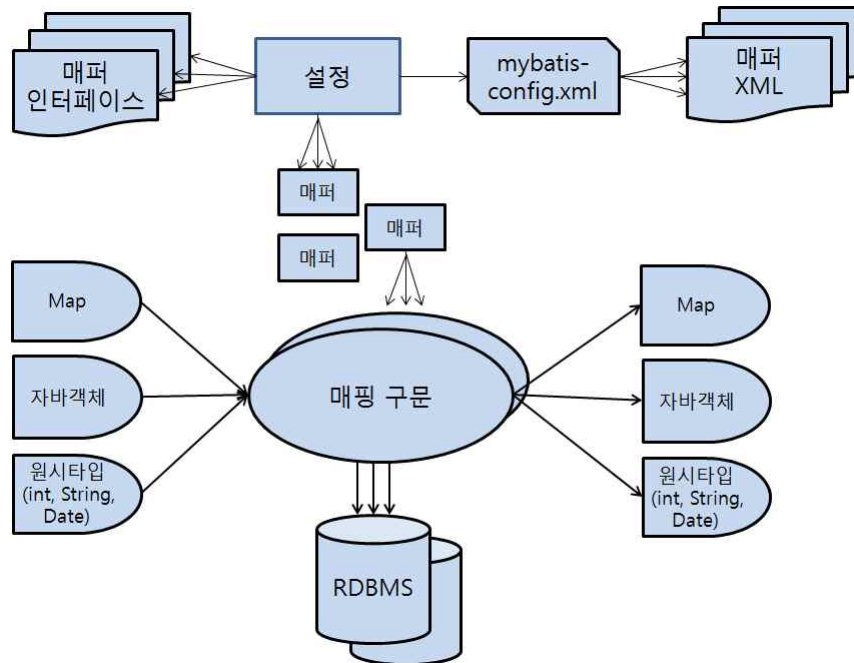
```

<terminated> BoardServiceClient (1) [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 10. 9 오후 1:10:21)
==> BoardVO [seq=8, title=myBatis 테스트, writer=삼장법사, content=myBatis 내용, regDate=Wed Oct 09 13:10:24 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]
==> BoardVO [seq=7, title=파일업로드, writer=손오공, content=파일업로드 테스트, regDate=Fri Oct 04 14:35:39 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]
==> BoardVO [seq=6, title=임시제목, writer=홍길동, content=테스트 내용, regDate=Fri Oct 04 10:26:18 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]
==> BoardVO [seq=5, title=세션테스트, writer=홍길동, content=세션 테스트, regDate=Tue Oct 01 11:04:51 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]
==> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=Thu Sep 12 17:22:38 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]
==> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=Thu Sep 12 13:34:24 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]
==> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=Thu Sep 12 13:14:54 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]
==> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일백..., regDate=Sun Sep 08 01:54:42 KST 2019, cnt=0, searchCondition=null, searchKeyword=null]

```


4. Mapper XML 파일 설정

① MyBatis 구조



② Mapper XML 파일 구조

- Mapper 파일의 구조를 보면 가장 먼저 DTD 선언이 등장하고 그 아래 <mapper> 엘리먼트가 선언된다. <mapper> 엘리먼트는 namespace 속성을 가지는데 이 namespace 속성을 이용하여 더 쉽게 유일한 SQL id를 만들 수 있다.

```
<?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="BoardDAO">
  <insert id="insertBoard"></insert>
  <update id="updateBoard"></update>
  <delete id="deleteBoard"></delete>
  <select id="getBoard"></select>
  <select id="getBoardList"></select>
</mapper>
```

- 네임스페이스가 지정된 Mapper의 SQL을 DAO 클래스에서 참조할 때 네임스페이스와 SQL의 아이디를 결합하여 참조해야 한다.

```
<mapper namespace="BoardDAO">
  <delete id="deleteBoard" parameterType="myboard">
    DELETE FROM myboard WHERE
    seq = #{seq}
  </delete>
</mapper>

public void deleteBoard(BoardVO vo) {
  sqlSession.delete("BoardDAO.deleteBoard", vo);
}
```

③ <select> 엘리먼트

```
<mapper namespace="BoardDAO">
  <typeAliases>
    <typeAlias type="tommy.spring.web.board.BoardVO" alias="myboard" />
  </typeAliases>
  <select id="getBoard" resultType="myboard" parameterType="myboard">
    SELECT * FROM myboard
    WHERE
      seq=#{seq}
  </select>

  <select id="getBoardList" resultType="myboard" parameterType="myboard">
    SELECT * FROM myboard
    WHERE
      title LIKE '%'||#{searchKeyword}||'%'
    ORDER BY seq DESC
  </select>
</mapper>
```

- ☐ <select> 엘리먼트는 테이블을 조회하는 SELECT 구문을 작성할 때 사용한다. parameterType과 resultType 속성을 사용할 수 있다.
- ☐ id 속성은 필수 속성으로 반드시 전체 Mapper 파일들 내에서 유일한 아이디를 등록해야 한다.
- ☐ Mapper 파일에 등록된 SQL을 실행하려면 SQL 실행에 필요한 데이터를 외부로부터 받아야 하는데 이때 parameterType 속성을 사용할 수 있다.
- ☐ parameterType으로 지정된 클래스에는 사용자가 입력한 값들을 저장할 여러 변수가 있다. 변수들을 이용하여 SQL 구문에 사용자 입력 값들을 설정하는데 이때 #{변수명} 표현을 사용한다. 그리고 중요한건 parameterType 속성은 생략할 수 있으며 대부분 생략한다.
- ☐ 검색과 관련된 SQL 구문이 실행되면 ResultSet이 리턴되며 ResultSet에 저장된 검색 결과를 어떤 자바 객체에 매핑할지를 지정해야 하는데 이때 사용하는 것이 resultType 속성이다.
- ☐ resultType 속성은 <select> 엘리먼트에서만 사용할 수 있으며 parameterType 속성과 달리 <select> 엘리먼트에서 절대 생략할 수 없는 속성이다.
- ☐ parameterType과 resultType 속성 모두 <alias>를 사용할 수 있다.

④ <insert> 엘리먼트

- ☐ <insert> 엘리먼트는 데이터베이스에 데이터를 삽입하는 INSERT 구문을 작성하는 요소이다.

```
<insert id="insertBoard" parameterType="myboard">
  <selectKey keyProperty="seq" resultType="int">
    SELECT board_seq.nextval as seq FROM dual
  </selectKey>
  INSERT INTO myboard (seq, title, writer, content)
  VALUES(#{seq}, #{title}, #{writer}, #{content})
</insert>
```

- ☐ <insert> 구문은 자식 요소로 <selectKey> 엘리먼트를 가질 수 있다. 대부분 관계형 데이터베이스에서는 기본 키 필드의 자동 생성을 지원하는데 MyBatis에서는 <insert> 요소의 자식 요소인 <selectKey> 요소를 사용하여 생성된 키를 쉽게 가져올 수 있는 방법을 제공한다.

⑤ <update> 엘리먼트

- <update> 엘리먼트는 데이터를 수정할 때 사용하는 UPDATE 구문을 작성하는 요소이다.

```
<update id="updateBoard" parameterType="myboard">
    UPDATE myboard
    SET title=#{title}, content=#{content}
    WHERE seq = #{seq}
</update>
```

⑥ <delete> 엘리먼트

- <delete> 엘리먼트는 데이터를 삭제할 때 사용하는 DELETE 구문을 작성하는 요소이다.

```
<delete id="deleteBoard" parameterType="myboard">
    DELETE FROM myboard WHERE seq = #{seq}
</delete>
```

⑦ resultMap 속성

- 검색 결과를 특정 자바 객체에 매핑하여 리턴하기 위해서 resultMap 속성을 사용한다. 그러나 검색 결과를 resultMap 속성으로 매핑할 수 없는 몇몇 사례가 있다.
- 예를 들어 검색 결과가 단순 테이블 조회가 아닌 JOIN 구문을 포함할 때는 검색 결과를 정확하게 나타내는 하나의 자바 객체로 매핑할 수 없다.
- 또는 검색된 테이블의 컬럼 이름과 매핑에 사용될 자바 객체의 변수 이름이 다를 때는 검색 결과가 정확하게 자바 객체로 매핑되지 않는다.
- 위와 같은 경우에 <resultMap> 엘리먼트를 사용하여 매핑 규칙을 지정해야 한다.

```
<resultMap type="myboard" id="boardResult">
    <id property="seq" column="SEQ" />
    <result property="title" column="TITLE"/>
    <result property="writer" column="WRITER"/>
    <result property="content" column="CONTENT"/>
    <result property="regDate" column="REGDATE"/>
    <result property="cnt" column="CNT"/>
</resultMap>

<select id="getBoardList" resultMap="boardResult">
    SELECT * FROM myboard
    WHERE title LIKE '%'||#{searchKeyword}||'%'
    ORDER BY seq DESC
</select>
```

- 위 설정에서는 PK에 해당하는 SEQ 컬럼만 <id> 엘리먼트를 사용했고 나머지는 <result> 엘리먼트를 이용하여 검색 결과로 얻어낸 컬럼의 값과 BoardVO 객체의 변수를 매핑하고 있다.

⑧ CDATA Section 사용

- SQL 구문내에서 "<" 기호를 사용한다면 에러가 발생한다. 이는 XML 파서가 XML 파일을 처리할 때 "<" 기호를 작다라는 의미의 연산자가 아니라 또 다른 태그의 시작으로 처리하기 때문이다.
- 이러한 에러를 방지하기 위하여 CDATA Section으로 SQL 구문을 감싸주면 된다.
- CDATA Section은 MyBatis와 상관없는 XML 고유의 문법으로서 CDATA 영역에 작성된 데이터는 단순한 문자 데이터이므로 XML 파서가 해석하지 않도록 한다. 결국 CDATA Section 안에서 작성된 데이터는 XML 파서가 처리하지 않고 그대로 데이터베이스에 전달하므로 문제가 발생하지 않는다.

```

<select id="getBoard" resultType="myboard">
    <![CDATA[
        SELECT * FROM myboard WHERE seq <= #{seq}
    ]]>
</select>

```

⑨ SQL 대문자로 설정하기

- Mapper 파일에 등록되는 SQL 구문은 일반적으로 대문자로 작성한다. 사실 SQL 구문은 대문자 소문자를 구별하지 않는다. 하지만 일반적으로 가독성을 위하여 SQL 구문을 대문자로 표현하는 것을 선호하고 있다.

5. MyBatis JAVA API

① SqlSessionFactoryBuilder 클래스

- MyBatis로 DAO 클래스의 CRUD를 구현하려면 MyBatis에서 제공하는 SqlSession 객체를 사용해야 한다. 그런데 SqlSession 객체는 SqlSessionFactory로부터 얻어야 하므로 가장 먼저 SqlSessionFactory 객체를 생성해야 한다.
- SqlSessionFactory 객체를 생성하려면 SqlSessionFactoryBuilder의 build() 메서드를 이용하는데 build() 메서드는 MyBatis 설정파일(mybatis-config)을 로딩하여 SqlSessionFactory 객체를 생성한다.
- 그리고 mybatis-config.xml 파일을 로딩하려면 입력 스트림인 Reader 객체가 필요하다.
- Reader 객체는 Resources 클래스의 getResourceAsReader() 메서드를 사용하여 얻어낼 수 있다.

```

Reader reader = Resources.getResourceAsReader("mybatis-config.xml");
sessionFactory = new SqlSessionFactoryBuilder().build(reader);

```

② SqlSessionFactory 클래스

- SqlSessionFactory 객체는 openSession()이라는 메서드를 제공하며 이 메서드를 이용해서 SqlSession 객체를 얻을 수 있다.

③ SqlSession 객체

- selectOne() 메서드 : 하나의 데이터를 검색하는 SQL 구문을 실행할 때 사용
 - public Object selectOne(String statement)
 - public Object selectOne(String statement, Object parameter)
- selectList() 메서드 : 여러 개의 데이터가 검색되는 SQL 구문을 실행할 때 사용
 - public List selectList(String statement)
 - public List selectList(String statement, Object parameter)
- insert(), update(), delete() 메서드
 - public int insert(String statement, Object parameter)
 - public int update(String statementId, Object parameterObject) throws SQLException
 - public int delete(String statementId, Object parameterObject) throws SQLException

6. 스프링과 MyBatis 연동하기

① 라이브러리 내려받기

- ☐ mvnrepository에서 mybatis와 mybatis-spring을 검색하여 pom.xml에 의존관계를 추가한다.

```
1 <!-- 상단 부분 생략 -->
2     <dependencies>
3         <!-- mybatis -->
4         <dependency>
5             <groupId>org.mybatis</groupId>
6             <artifactId>mybatis</artifactId>
7             <version>3.5.2</version>
8         </dependency>
9         <!-- mybatis-spring -->
10        <dependency>
11            <groupId>org.mybatis</groupId>
12            <artifactId>mybatis-spring</artifactId>
13            <version>2.0.2</version>
14        </dependency>
15 <!-- 하단 부분 생략 -->
```

② MyBatis 설정 파일 작성

- ☐ src/main/resources 패키지에 mybatis-config.xml을 작성한다.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <typeAliases>
7         <typeAlias type="tommy.spring.web.board.BoardVO" alias="myboard" />
8     </typeAliases>
9     <mappers>
10        <mapper resource="mappings/board-mapping.xml" />
11    </mappers>
12 </configuration>
```

③ SQL Mapper 파일 작성

- ☐ src/main/resources 패키지에 mappings 패키지를 생성하고 board-mapping.xml을 작성한다.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4 <mapper namespace="BoardDAO">
5     <resultMap type="myboard" id="boardResult">
6         <id property="seq" column="SEQ" />
7         <result property="title" column="TITLE"/>
8         <result property="writer" column="WRITER"/>
9         <result property="content" column="CONTENT"/>
```

```

10      <result property="regDate" column="REGDATE"/>
11      <result property="cnt" column="CNT"/>
12  </resultMap>
13  <insert id="insertBoard" parameterType="myboard">
14      INSERT INTO myboard (seq, title, writer, content)
15      VALUES((select nvl(max(seq), 0)+1 from myboard), #{title}, #{writer}, {content})
16  </insert>
17  <update id="updateBoard" parameterType="myboard">
18      UPDATE myboard
19      SET title=#{title}, content=#{content} WHERE seq = #{seq}
20  </update>
21  <delete id="deleteBoard" parameterType="myboard">
22      DELETE FROM myboard WHERE seq = #{seq}
23  </delete>
24  <select id="getBoard" resultType="myboard">
25      SELECT * FROM myboard WHERE seq=#{seq}
26  </select>
27  <select id="getBoardList" resultMap="boardResult">
28      SELECT * FROM myboard
29      WHERE title LIKE '%| |#{searchKeyword}| |%'
30      ORDER BY seq DESC
31  </select>
32 </mapper>

```

④ 스프링 연동설정

- ☐ 스프링과 MyBatis를 연동하려면 우선 스프링 설정 파일에 SqlSessionFactoryBean 클래스를 Bean으로 등록해야 한다.

■ applicationContext.xml 수정

```

1  <!-- 상단 부분 생략-->
2      <bean id="sessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
3          <property name="dataSource" ref="dataSource" />
4          <property name="configLocation" value="classpath:mybatis-config.xml" />
5      </bean>
6  <!-- 하단 부분 생략-->

```

⑤ DAO 클래스 구현 방법 1 - SqlSessionDaoSupport 클래스 이용

■ BoardDAOMybatis 작성

```

1  package tommy.spring.web.board.impl;
2  import java.util.List;
3  import org.apache.ibatis.session.SqlSessionFactory;
4  import org.mybatis.spring.support.SqlSessionDaoSupport;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Repository;
7  import tommy.spring.web.board.BoardVO;
8  @Repository
9  public class BoardDAOMybatis extends SqlSessionDaoSupport {
10      @Autowired

```



```

11      public void setSessionFactory(SqlSessionFactory sqlSessionFactory) {
12          super.setSessionFactory(sqlSessionFactory);
13      }
14      public void insertBoard(BoardVO vo) {
15          System.out.println("---> MyBatis로 insertBoard() 기능 처리");
16          getSession().insert("BoardDAO.insertBoard", vo);
17      }
18      public void updateBoard(BoardVO vo) {
19          System.out.println("---> MyBatis로 updateBoard() 기능 처리");
20          getSession().update("BoardDAO.updateBoard", vo);
21      }
22      public void deleteBoard(BoardVO vo) {
23          System.out.println("---> MyBatis로 deleteBoard() 기능 처리");
24          getSession().delete("BoardDAO.deleteBoard", vo);
25      }
26      public BoardVO getBoard(BoardVO vo) {
27          System.out.println("---> MyBatis로 getBoard() 기능 처리");
28          return (BoardVO) getSession().selectOne("BoardDAO.getBoard", vo);
29      }
30      public List<BoardVO> getBoardList(BoardVO vo) {
31          System.out.println("---> MyBatis로 getBoardList() 기능 처리");
32          return getSession().selectList("BoardDAO.getBoardList", vo);
33      }
34  }

```

- ☐ SqlSessionDaoSupport 클래스를 상속한 후에 가장 먼저 한 작업이 setSessionFactory() 메서드를 재정의 한 것이다. 재 정의한 setSessionFactory() 메서드 위에 @Autowired를 붙였는데 이렇게 하면 스프링 컨테이너가 setSessionFactory() 메서드를 자동으로 호출한다.
- ☐ 이때 스프링 설정파일에 <bean> 등록된 SqlSessionFactoryBean 객체를 인자로 받아 부모인 SqlSessionDaoSupport에 setSessionFactory() 메서드로 실행해 준다.
- ☐ 위와 같이 작업을 해야 SqlSessionDaoSupport 클래스로부터 상속된 getSession() 메서드를 호출하여 SqlSession 객체를 리턴받을 수 있다.

⑥ DAO 클래스 구현 방법 2 - SqlSessionTemplate 클래스 이용

■ applicationContext.xml 수정

```

1  <!-- 상단 부분 생략-->
2      <bean id="sessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
3          <property name="dataSource" ref="dataSource" />
4          <property name="configLocation" value="classpath:mybatis-config.xml" />
5      </bean>
6      <bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
7          <constructor-arg ref="sessionFactory" />
8      </bean>
9  <!-- 하단 부분 생략-->

```

- ☐ 위 설정에서 주의할 점은 SqlSessionTemplate 클래스에는 setter 메서드가 없어서 생성자 인젝션을 사용할 수밖에 없다.

- ☐ 그리고 아래와 같이 DAO 클래스를 구현할 때 SqlSessionTemplate 객체를 @Autowired를 이용하여 의존성 주입 처리를 하면 된다.

■ BoardDAOMybatis 수정

```
1 package tommy.spring.web.board.impl;
2 import java.util.List;
3 import org.mybatis.spring.SqlSessionTemplate;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Repository;
6 import tommy.spring.web.board.BoardVO;
7 @Repository
8 public class BoardDAOMybatis {
9     @Autowired
10    private SqlSessionTemplate sqlSessionTemplate;
11    public void insertBoard(BoardVO vo) {
12        System.out.println("---> MyBatis로 insertBoard() 기능 처리");
13        sqlSessionTemplate.insert("BoardDAO.insertBoard", vo);
14    }
15    public void updateBoard(BoardVO vo) {
16        System.out.println("---> MyBatis로 updateBoard() 기능 처리");
17        sqlSessionTemplate.update("BoardDAO.updateBoard", vo);
18    }
19    public void deleteBoard(BoardVO vo) {
20        System.out.println("---> MyBatis로 deleteBoard() 기능 처리");
21        sqlSessionTemplate.delete("BoardDAO.deleteBoard", vo);
22    }
23    public BoardVO getBoard(BoardVO vo) {
24        System.out.println("---> MyBatis로 getBoard() 기능 처리");
25        return (BoardVO) sqlSessionTemplate.selectOne("BoardDAO.getBoard", vo);
26    }
27    public List<BoardVO> getBoardList(BoardVO vo) {
28        System.out.println("---> MyBatis로 getBoardList() 기능 처리");
29        return sqlSessionTemplate.selectList("BoardDAO.getBoardList", vo);
30    }
31 }
```

⑦ MyBatis 연동 테스트

■ BoardServiceImpl 클래스 수정

```
1 <!-- 상단 부분 생략-->
2 @Service("boardService")
3 public class BoardServiceImpl implements BoardService {
4     @Autowired
5     private BoardDAOMybatis boardDAO;
6 <!-- 하단 부분 생략-->
```

- ☐ 위와 같이 수정하였다면 BoardServiceClient로 테스트를 해도 되고 웹 어플리케이션으로 테스트를

수행할 수 있다. 테스트를 진행하여 결과를 확인해 보자.

The screenshot shows a web browser at `http://localhost:8080/myweb/insertBoard.do` displaying a '게시글 목록' (Board List) page. The page includes a search bar and a table of board entries. A log window on the right shows the following messages:

```

INFO: Initializing Spring DispatcherServlet 'action'
INFO: org.springframework.web.servlet.DispatcherServlet - Initializing Servlet 'action'
INFO: org.springframework.web.servlet.DispatcherServlet - Completed initialization in 1984 ms
로그인 화면으로 이동
로그인 인증 처리
JDBC로 getUser() 기능 처리
글 목록 검색 처리
---> Mybatis로 getBoardList() 기능 처리
글 등록 처리
---> Mybatis로 insertBoard() 기능 처리
글 목록 검색 처리
---> Mybatis로 getBoardList() 기능 처리
  
```

번호	제목	작성자	등록일	조회수
8	myBatis	이승재	2019-10-09	0
7	파일업로드	손오공	2019-10-04	0
6	임시제목	홍길동	2019-10-04	0
5	세션테스트	홍길동	2019-10-01	0
4	임시제목	홍길동	2019-09-12	0
3	임시제목	홍길동	2019-09-12	0
2	임시제목	홍길동	2019-09-12	0
1	임시제목	홍길동	2019-09-08	0

⑧ Dynamic SQL

☐ MyBatis는 SQL의 재사용성과 유연성을 향상하고자 Dynamic SQL을 지원한다. Dynamic SQL을 사용하면 조건에 따라 다양한 쿼리를 데이터베이스에 전송할 수 있다.

☐ 만약 현재 상태에서 검색기능을 추가한다고 하면 아래와 같은 SQL 문이 필요할 것이다.

■ board-mapping.xml

```

1 <!-- 상단 부분 생략-->
2     <select id="getBoardList_T" resultMap="boardResult">
3         SELECT * FROM myboard
4         WHERE title LIKE '% | |#{searchKeyword}| |%'
5         ORDER BY seq DESC
6     </select>
7
8     <select id="getBoardList_C" resultMap="boardResult">
9         SELECT * FROM myboard
10        WHERE content LIKE '% | |#{searchKeyword}| |%'
11        ORDER BY seq DESC
12    </select>
13 <!-- 하단 부분 생략-->
  
```

☐ 위와 같이 두 개의 쿼리를 등록했으면 DAO 클래스의 `getBoardList()` 메서드에 검색 조건에 따른 분기로직을 추가해야 한다.

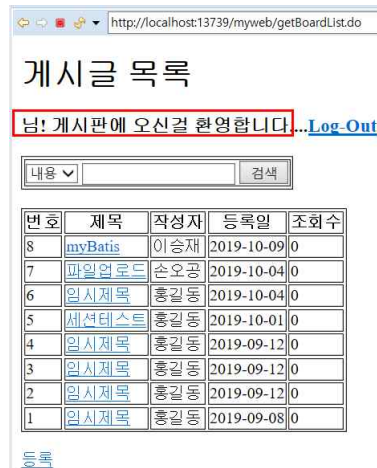
■ BoardDAOmybatis

```

1 <!-- 상단 부분 생략-->
2     public List<BoardVO> getBoardList(BoardVO vo) {
3         System.out.println("---> Mybatis로 getBoardList() 기능 처리");
4         if(vo.getSearchCondition().equals("TITLE")) {
5             return sqlSessionTemplate.selectList("BoardDAO.getBoardList_T", vo);
6         }else if(vo.getSearchCondition().equals("CONTENT")) {
7             return sqlSessionTemplate.selectList("BoardDAO.getBoardList_C", vo);
8         }
9     }
  
```

9	return null;
10	}
11	<!-- 하단 부분 생략-->

- 위와 같이 수정하고 index.jsp 파일을 실행한 후 글 목록 검색기능을 실행해 보면 정상적으로 실행될 것이다.



- 하지만 이런 방식으로 검색 기능을 구현한다면 이후에 추가되는 검색 조건에 대해 비슷한 SQL 구문을 반복해서 작성해야 할 것이다. 결국 유지보수에 어려움이 따르게 된다.
- 또 DAO 클래스의 메서드 역시 수정해야 하는 문제점이 발생한다.
- 이런 SQL문의 중복 문제를 해결하기 위해서 MyBatis에서는 Dynamic SQL을 지원한다.

■ board-mapping.xml 수정

1	<!-- 상단 부분 생략-->
2	<select id="getBoardList" resultMap="boardResult">
3	SELECT * FROM myboard
4	WHERE 1=1
5	<if test="searchCondition=='TITLE'">
6	AND title LIKE '% #{searchKeyword} %'
7	</if>
8	<if test="searchCondition=='CONTENT'">
9	AND content LIKE '% #{searchKeyword} %'
10	</if>
11	ORDER BY seq DESC
12	</select>
13	<!-- 하단 부분 생략-->

- Dynamic SQL을 적용한 구문을 보면 <if>라는 동적 요소를 사용하여 조건에 따른 분기 처리를 하고 있다. 만약 searchCondition이 “TITLE” 을 가지고 있으면 제목 검색에 해당하는 조건이 추가되고 “CONTENT” 라는 값을 가지고 있으면 내용 검색에 해당하는 조건이 추가되어 실행된다.
- 이렇게 동적 엘리먼트를 이용하여 SQL을 처리할 수 있으므로 검색과 관련된 SQL문은 하나만 있으면 된다.
- 그리고 당연히 DAO 클래스의 메서드 역시 원래 코드를 유지할 수 있다. 또 새로운 조건이 추가된다 하더라도 수정할 필요가 없다.

■ BoardDAOMybatis 수정

```

1 <!-- 상단 부분 생략-->
2     public List<BoardVO> getBoardList(BoardVO vo) {
3         System.out.println("---> MyBatis로 getBoardList() 기능 처리");
4         return sqlSessionTemplate.selectList("BoardDAO.getBoardList", vo);
5     }
6 <!-- 하단 부분 생략-->

```

□ 관련기능이 정상적으로 동작하는지 테스트 해 보자.

게시글 목록

이승재님! 게시판에 오신걸 환영합니다....[Log-Out](#)

제목 ▾ 제목 검색

번호	제목	작성자	등록일	조회수
8	myBatis	이승재	2019-10-09	0
7	파일업로드	손오공	2019-10-04	0
6	임시제목	홍길동	2019-10-04	0
5	세션테스트	홍길동	2019-10-01	0
4	임시제목	홍길동	2019-09-12	0
3	임시제목	홍길동	2019-09-12	0
2	임시제목	홍길동	2019-09-12	0
1	임시제목	홍길동	2019-09-08	0

[등록](#)



게시글 목록

이승재님! 게시판에 오신걸 환영합니다....[Log-Out](#)

내용 ▾ 검색

번호	제목	작성자	등록일	조회수
6	임시제목	홍길동	2019-10-04	0
4	임시제목	홍길동	2019-09-12	0
3	임시제목	홍길동	2019-09-12	0
2	임시제목	홍길동	2019-09-12	0
1	임시제목	홍길동	2019-09-08	0

[등록](#)