

chapter03

mybatis

1. **DataSource**
2. MyBatis
3. SQL

■ DataSource

- JDBC를 통해 DB를 사용하려면, Connection 타입의 DB 연결 객체가 필요하다.
- 엔터프라이즈 환경에서는 각 요청마다 Connection을 새롭게 만들고 종료시킨다.
- 애플리케이션과 DB사이의 실제 커넥션을 매번 새롭게 만드는 것은 비효율적이고 성능저하
- 풀링(pooling) 기법 사용
 - 정해진 개수의 DB Connection Pool에 준비하고 애플리케이션 요청때 마다 꺼내서 할당하고 돌려받아 pool에 저장.
 - Spring에서는 DataSource를 하나의 독립된 빈으로 등록하도록 강력하게 권장.
- 엔터프라이즈 시스템에서는 반드시 DB 연결 풀 기능을 지원하는 DataSource를 사용해야 한다.

■ DataSource : Connection Pool

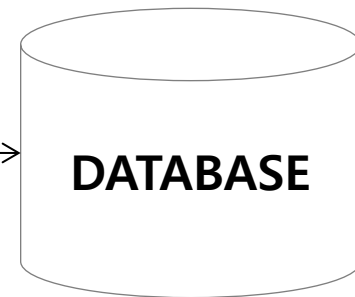
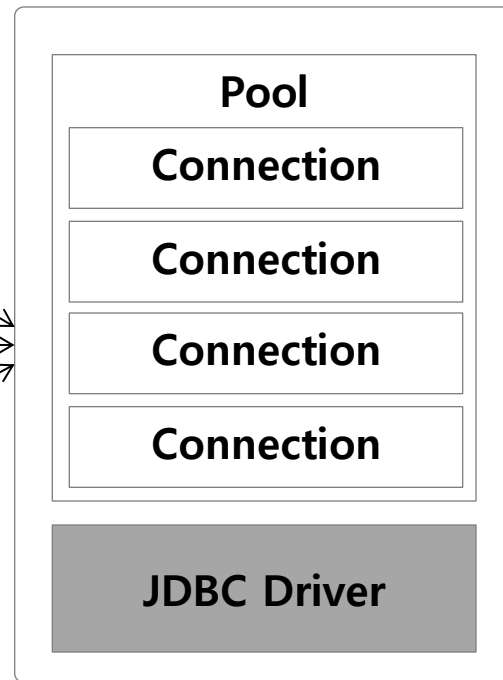
EmaillistDao.java

```
int insert(EmaillistVo vo)  
List<EmaillistVo> getList()
```

EmaillistDao.java

```
int insert(EmaillistVo vo)  
List<EmaillistVo> getList()
```

DataSource



■ pom.xml : 라이브러리 추가

```
<!-- spring jdbc -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
```

■ applicationContext.xml : OracleDataSource bean등록

```
<!-- oracle datasource -->
<bean id="oracleDatasource" class="oracle.jdbc.pool.OracleDataSource" destroy-method="close">
  <property name="URL" value="jdbc:oracle:thin:@localhost:1521:xe" />
  <property name="user" value="webdb" />
  <property name="password" value="webdb" />
  <property name="connectionCachingEnabled" value="true" />
  <qualifier value="main-db" />
</bean>
```

chapter03

mybatis

1. DataSource
2. **MyBatis**
3. SQL

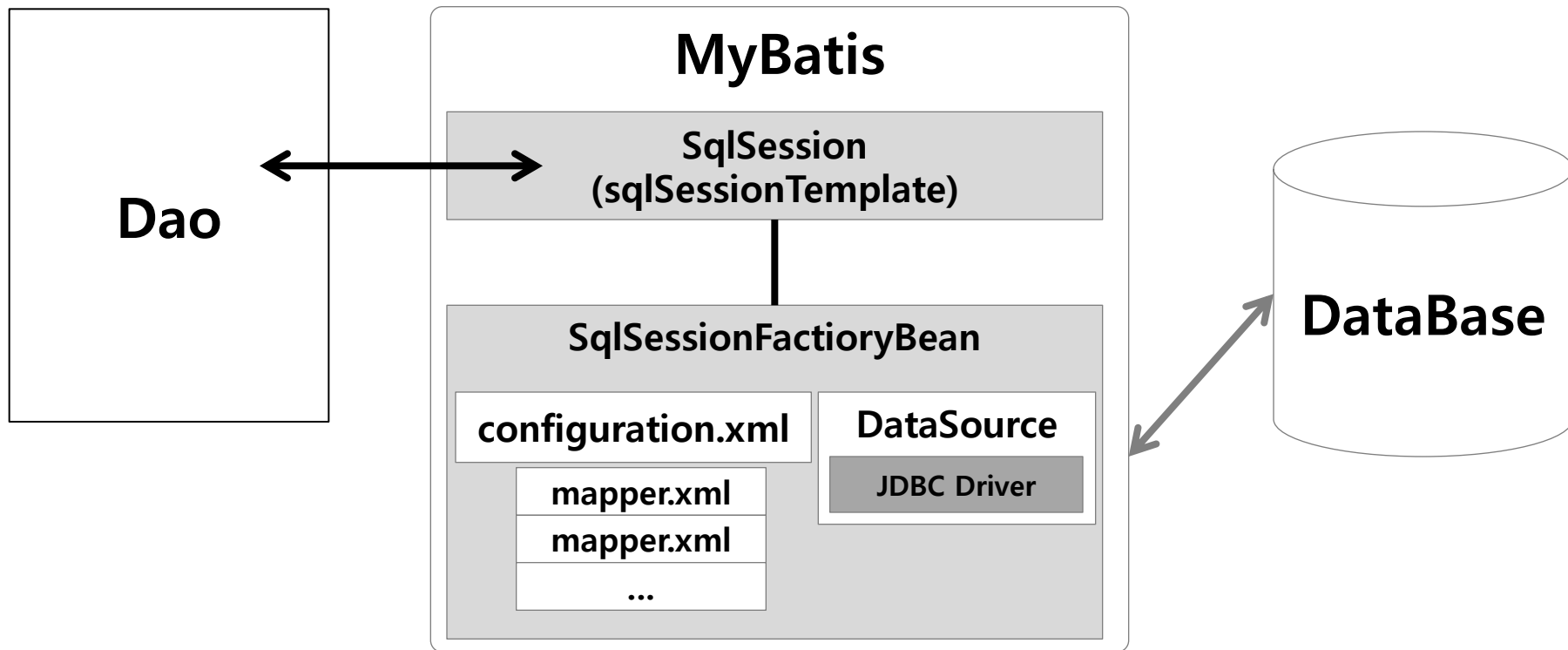
■ MyBatis3.X 소개

- MyBatis2.x(iBatis)의 후속으로 등장한 ORM 프레임워크이다.
- XML를 이용한 SQL과 ORM 을 지원한다.
- 본격적인 ORM인 JPA나 Hibernate 처럼 새로운 DB 프로그래밍 패러다임을 이해해야 하는 것은 아니다. (MyBatis3.x에서는 Mapper 인터페이스를 통해 지원)
- 이미 익숙한 SQL를 그대로 사용하고 JDBC코드의 불편함을 제거
- 가장 큰 특징은 SQL을 자바코드에서 분리해서 별도의 XML 파일 안에 작성하고 관리할 수 있는 것이다.
- 스프링 3.0부터는 MyBatis3.x(iBatis2.x) 버전에 스프링 데이터 액세스 기술 대부분을 지원한다.(DataSource Bean 사용, 스프링 트랜잭션, 예외 자동변환, 템플릿)

■ MyBatis3.X 스프링에서 사용하기

- MyBatis의 DAO는 `SQLSession` 인터페이스를 구현한 클래스의 객체를 DI받아 사용한다.
- MyBatis의 DAO는 `SQLSessionDaoSupport` 추상 클래스를 상속받아 구현하기도 한다.
- 그리고 `Mapper` 인터페이스를 통한 OR 매핑 기능을 지원한다.
- 이 중에 `SQLSession` 인터페이스를 구현한 클래스의 객체의 DI 방식을 주로 사용하게 된다.
- `SQLSession` 인터페이스를 구현한 `SQLSessionTemplate` 클래스를 사용한다.

■ DataSource : Connection Pool



■ pom.xml : 라이브러리 추가

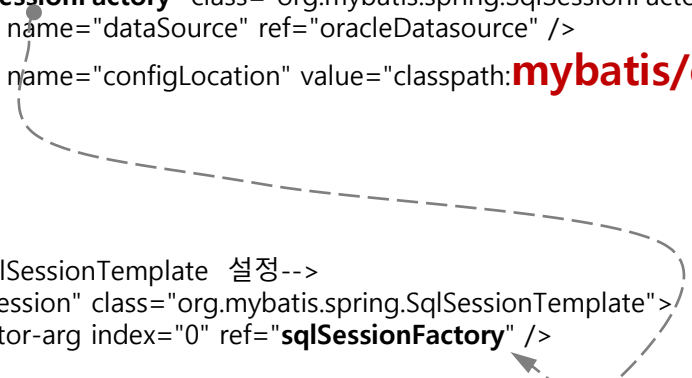
```
<!-- MyBatis -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.2.2</version>
</dependency>

<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.2.0</version>
</dependency>
```

■ applicationContext.xml :

```
<!-- MyBatis SqlSessionFactoryBean 설정-->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="oracleDatasource" />
    <property name="configLocation" value="classpath:mybatis/configuration.xml" />
</bean>

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



A dashed line with an arrow at the end connects the `ref="sqlSessionFactory"` attribute in the `sqlSession` bean definition to the `sqlSessionFactory` bean definition above it, illustrating the dependency.

■ EmaillistDao : SqlSessionTemplate를 DI한다.

```
@Repository
public class EmaillistDao {

    @Autowired
    private SqlSession sqlSession;

    public List<EmaillistVo> getList() {
        List<EmaillistVo> list = sqlSession.selectList("emailist.getList");
        return list;
    }

    public int insert(EmaillistVo vo) {
        return sqlSession.insert("emailist.insert", vo);
    }

    public EmaillistVo get(int no) {
        EmaillistVo vo = sqlSession.selectOne("emailist.get", no);
        System.out.println(vo.toString());

        return vo;
    }
}
```

```
sqlSession.insert("emailist.insert", vo);

sqlSession.update("emailist.update", vo);

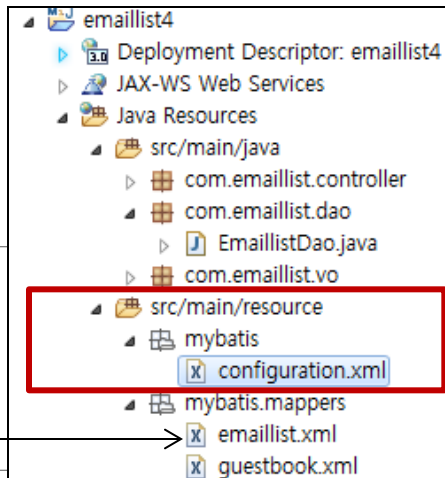
sqlSession.delete("emailist.delete", no);

sqlSession.selectOne("emailist.getbyno", no);

sqlSession.selectList("emailist.getList", page);
```

■ configuration.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>
  <typeAliases>
  </typeAliases>
  <mappers>
    <mapper resource="mybatis/mappers/emaillist.xml" />
    <mapper resource="mybatis/mappers/guestbook.xml" />
  </mappers>
</configuration>
```



■ emaillist.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="emaillist">

  <!-- 쿼리문 작성 -->

</mapper>
```

chapter03

mybatis

1. DataSource
2. MyBatis
3. **SQL**

■ 기본구문

```
<select id="식별자"  parameterType="기본자료형 또는 vo"  resultType="기본자료형 또는 vo">  
    <![CDATA[  
        <!-- 쿼리문 -->  
    ]]>  
</select>
```

```
<insert id="식별자"  parameterType="기본자료형 또는 vo" >  
</insert>
```

```
<update id="식별자"  parameterType="기본자료형 또는 vo" >  
</update>
```

```
<delete id="식별자"  parameterType="기본자료형 또는 vo" >  
</delete>
```

■ select 하기

- 결과의 칼럼 이름과 resultType의 class의 필드명이 같은 경우

```
<select id="list" resultType="com.javaex.vo.EmaillistVo">
  select no,
         first_name as firstName,
         last_name as lastName,
         email
  from   email_list
  order by no desc
</select>
```

■ select 하기

- 결과의 칼럼 이름과 resultMap의 class의 필드명이 다른 경우

```
<resultMap type="com.javaex.vo.EmaillistVo" id="resultMapList">
  <result column="no" property="no" />
  <result column="first_name" property="firstName" />
  <result column="last_name" property="lastName" />
  <result column="email" property="email" />
</resultMap>
```

```
<select id="getList" resultMap="resultMapList">
  <![CDATA[
    select no, last_name, first_name, email
    from emaillist
    order by no desc
  ]]>
</select>
```


■ 파라미터 바인딩

•객체를 이용한 파라미터 바인딩

```
<insert id="insert" parameterType="com.javaex.vo.EmaillistVo">
    <![CDATA[
        insert into emaillist
        values (seq_emaillist_no.nextval, #{lastName}, #{firstName}, #{email} )

    ]]>
</insert>
```

•파라미터가 하나일때 바인딩

```
<select id="getByNo" parameterType="int"    resultType="com.javaex.vo.EmaillistVo">
    <![CDATA[
        select no, last_name, first_name, email
        from emaillist
        where no = #{no}

    ]]>
</select>
```

-#{no} 는 임의지정해도 상관없다. (1개일때는 이름과 상관없이 찾는다.)

-int 는 내장된 alias 이다. (byte, short, long, int, integer, double, float, boolean, string)

■ 파라미터 바인딩

- 파라미터 클래스가 존재하지 않고 여러 값을 파라미터로 넘겨야 할때 → Map이용

```
<select id="getListByMap" parameterType="map" resultType="com.javaex.vo.EmaillistVo">
    <![CDATA[
        select no, last_name lastName, first_name, email
        from emaillist
        where last_name = #{last_name}
        or email = #{email}
    ]]>
</select>
```

•Map사용

```
Map<String, Object> map = new HashMap<String, Object>();
map.put("last_name", "정");
map.put("email", "nnnn");
List<EmaillistVo> list = emailListDao.getListByMap(map);
```

■ Insert 후, 새로들어 간 row의 Primary Key를 받아야 하는 경우


































```
<insert id="insert" parameterType="GuestbookVo">
  <selectKey keyProperty="no" resultType="int" order="BEFORE">
    select seq_guestbook_no.nextval from dual
  </selectKey>
  <![CDATA[
    insert
    into guestbook
    values ( #{no }, #{name }, #{password }, #{content }, SYSDATE )

  ]]>
</insert>
```

■ configuration.xml : alias 사용하여 parameterType, resultType을 짧게 줄인다.

```
<?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>
  <typeAliases>
    <typeAlias alias="EmailListVo" type="com.javaex.vo.EmaillistVo" />
  </typeAliases>
  <mappers>
    <mapper resource="mybatis/mappers/emaillist.xml" />
  </mappers>
</configuration>
```

```
<select id="list" resultType="EmailListVo">
  select no,
    first_name as firstName,
    last_name as lastName,
    email
  from email_list
  order by no desc
</select>
```

- ▼  emailist4
 - ▼  src/main/java
 - ▼  com.javaex.controller
 - ▶  EmailistController.java
 - ▼  com.javaex.dao
 - ▶  EmailistDao.java
 - ▼  com.javaex.vo
 - ▶  EmailistVo.java
 - ▼  src/main/resources
 - ▼  mybatis
 - ▼  mappers
 -  emailist.xml
 -  configuration.xml
 -  src/test/java
 -  src/test/resources
 - ▶  JRE System Library [JavaSE-1.8]
 - ▶  Apache Tomcat v8.5 [Apache Tomcat v8.5]
 - ▶  Maven Dependencies
 - ▶  build
 - ▶  src
 - ▶  target
 - ▼  WebContent
 - ▶  META-INF
 - ▼  WEB-INF
 -  lib
 - ▼  views
 -  hello.jsp
 -  index.jsp
 -  list.jsp
 -  applicationContext.xml
 -  spring-servlet.xml
 -  web.xml
 -  pom.xml

■ 준비할 사항

pom.xml -> mybatis 관련 설정

WEB-INF/web.xml -> Context Listener 와 DB설정 등 비즈니스로직 관리할 xml 설정 경로 지정 필요

WEB-INF/applicationContext.xml -> oracle datasource(DB계정연결), MyBatis SqlSessionFactoryBean, MyBatis SqlSessionTemplate 설정 필요

src/main/resources/mybatis/**configuration.xml** -> mapper 등록

src/main/resources/mybatis/mappers/**emaillist.xml** -> SQL문 등록

Dao, Vo, Controller 준비

■ pom.xml

```
<!-- mybatis -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.1</version>
</dependency>

<!-- mybatis-spring -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.3.0</version>
</dependency>
```

■ web.xml

```
<!-- Context Listener 등록 -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- DB설정 등 비즈니스 로직 설정을 관리할 xml 설정 경로 지정 -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```


■ applicationContext.xml

```
<!-- oracle datasource -->
<bean id= "oracleDataSource" class= "oracle.jdbc.pool.OracleDataSource" destroy-method= "close">
  <property name= "URL" value= "jdbc:oracle:thin:@localhost:1521:xe" />
  <property name= "user" value= "{webdb}" />
  <property name= "password" value= "{webdb}" />
  <property name= "connectionCachingEnabled" value= "true" />
  <qualifier value= "main-db" />
</bean>

<!-- MyBatis SqlSessionFactoryBean -->
<bean id= "sqlSessionFactory" class= "org.mybatis.spring.SqlSessionFactoryBean">
  <property name= "dataSource" ref= "oracleDataSource" />
  <property name= "configLocation" value= "classpath:mybatis/configuration.xml" />
</bean>

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

■ configuration.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>

  <typeAliases>
    <typeAlias alias="emaillist" type="com.javaex.vo.EmaillistVo"> </typeAlias>
  </typeAliases>

  <mappers>
    <mapper resource="mybatis/mappers/emaillist.xml" />
  </mappers>

</configuration>
```

■ emailist.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="EmailistXml">

  <!-- 쿼리문 작성 -->
  <select id="selectList" resultType="emailist">
    select no,
           first_name,
           last_name,
           email
    from emailist
    order by no desc
  </select>

</mapper>
```

03 emaillist app 작성

■ EmaillistDao.java (Vo 는 동일)

```
package com.javaex.dao;
```

```
import java.util.List;
```

```
import org.apache.ibatis.session.SqlSession;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.javaex.vo.EmaillistVo;
```

```
@Repository
```

```
public class EmaillistDao {
```

applicationContext.xml 의 해당 객체를 DI

```
@Autowired
```

```
private SqlSession sqlSession;
```

```
public List<EmaillistVo> getList(){
```

```
    System.out.println("----> sqlSession.selectList()");
```

```
    System.out.println(sqlSession);
```

select id

```
    return sqlSession.selectList("EmaillistXml.selectList");
```

Mapper에 등록된 NameSpace 명

```
}
```

```
}
```

■ EmaillistDao.java (Vo 는 동일)

```
package com.javaex.dao;
```

```
import java.util.List;
```

```
import org.apache.ibatis.session.SqlSession;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.javaex.vo.EmaillistVo;
```

```
@Repository
```

```
public class EmaillistDao {
```

```
@Autowired
```

```
private SqlSession sqlSession;
```

applicationContext.xml 의 해당 객체를 DI

```
public List<EmaillistVo> getList(){
```

```
    System.out.println("----> sqlSession.selectList()");
```

```
    System.out.println(sqlSession);
```

select id

```
    return sqlSession.selectList("EmaillistXml.selectList");
```

Mapper에 등록된 NameSpace 명

```
}
```

```
}
```

■ /WEB-INF/views/list.jsp

```
<body>
```

```
<table border=1 cellpadding=0 cellspacing=0>
```

```
<tr>
```

```
<th>no</th>
```

```
<th>last_name</th>
```

```
<th>first_name</th>
```

```
<th>email</th>
```

```
</tr>
```

```
<c:forEach items="${emaillistList}" var="emaillist">
```

```
<tr>
```

```
<td>${emaillist.no}</td>
```

```
<td>${emaillist.last_name}</td>
```

```
<td>${emaillist.first_name}</td>
```

```
<td>${emaillist.email}</td>
```

```
</tr>
```

```
</c:forEach>
```

```
</table>
```

```
</body>
```

Mapper에 등록된...

```
<select id="selectList" resultType="emaillist">
```