❑ About

- 이 문서는 행전안전부의 전자정부 프레임위크의 DataAccess 매뉴얼을 정리한 것입니다.
  또한 iBATIS 개발자 문서에서 필요한 부분을 인용합니다.
- iBATIS를 Persistant Framework, ORM Tool, SQL Mapper, Data Mapper 등 혼용되어 사용되나 여기서는 Data Mapper 라고 하겠습니다.
- 개발자 문서 및 전자정부 매뉴얼을 정리한 것이므로 개발자 문서를 꼭 참고 하시기 바랍니다.

❑ 참고

- Appache 사이트에서 iBATIS 는 Attic 상태
- MyBatis
  - http://www.mybatis.org
- iBATIS-SqlMaps-2 개발자 가이드(영문)
  - http://ibatis.apache.org/docs/java/pdf
- iBATIS-SqlMaps-2 개발자 가이드(한글, 이동국님 번역)
  - http://www.kldp.net/projects/kfwdp/download
- Spring Framework - Reference Documentation
  - http://static.springframework.org/spring/docs/2.5.6/reference/orm.html#orm-ibatis

❑ 목차

❑ intro

- iBATIS Data Mapper Framework는 DBMS에 접근할 때 필요한 자바코드를 현저하게 줄일 수 있다.
  자바코드의 20%를 사용하여 JDBC기능의 80%를 제공하는 간단한 프레임워크라는 뜻이다.
- iBATIS는 다른 프레임워크, ORM(Object Relation Mapping)에 비해 가장 큰 장점은 심플함(Simplicity)이다.
- XML 기술을 사용해서 간단하게 JavaBean(또는 Map)객체를 PreparedStatement의 파라미터와 ResultSets으로 쉽게 mapping할 수 있다.

❑ history

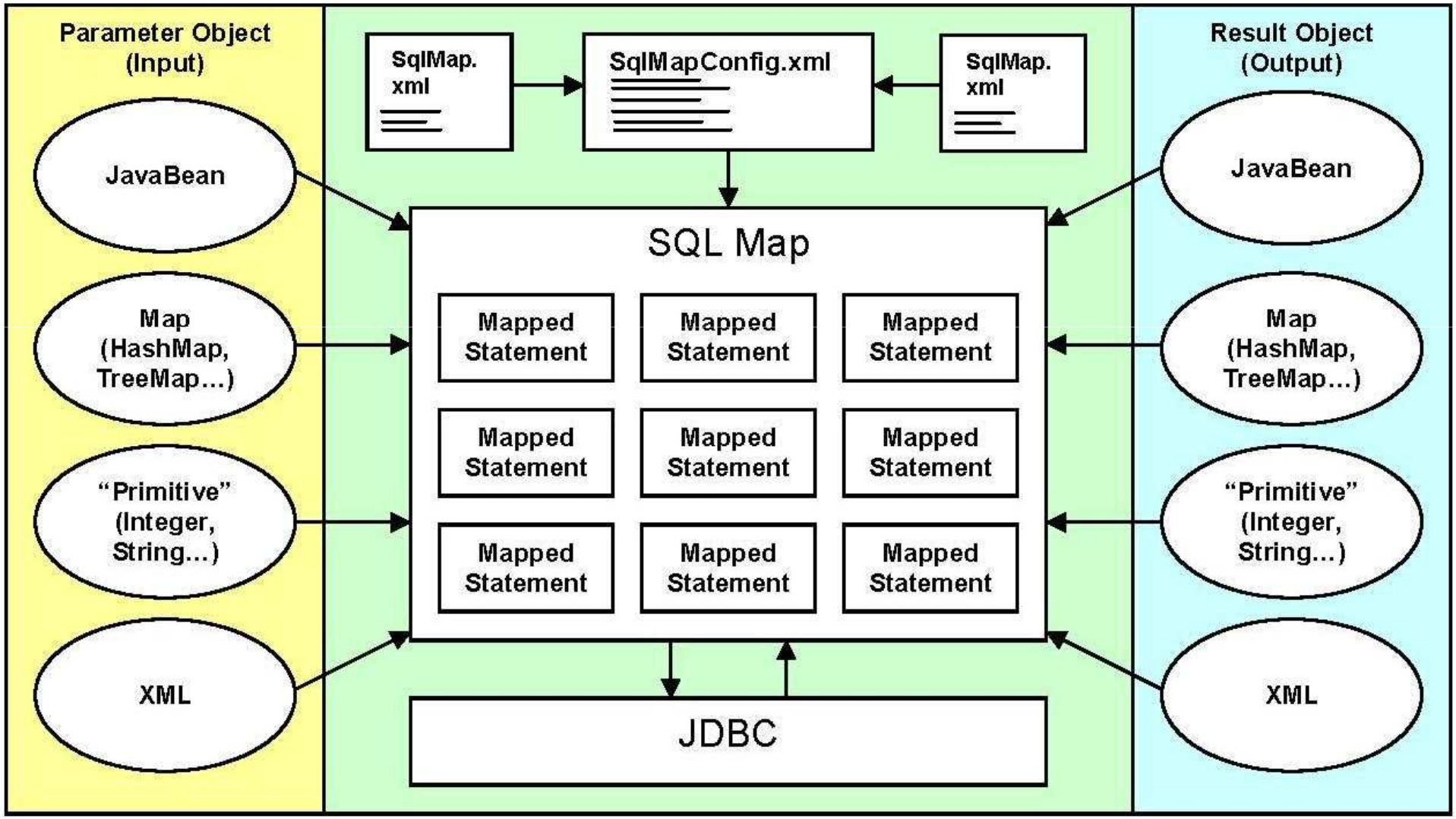
- 개발자인 Clinton Begin에 의해 java기반의 iBATIS 프로젝트 시작
- 2002년 ASF(Apache Software Foundation) Java 기반 정식 1.0 버전 발표
  .Net 기반 프로젝트 시작
- 2004년 ASF의 Top Level project로 승인
  .Net 기반 정식1.0 버전 발표
- 2010년 6월 Google Code로 프로젝트 이전
  버전이 3.x 로 업그레이드 하면서 iBatis에서 myBatis로 명칭 변경

❑ feature

- 추상화된 접근 방식 제공
  - JDBC 데이터 억세스에 대한 추상화된 접근 방식으로 간편하고 쉬운 API, 자원 연결/해제, 공통 에러 처리 등을 통합 지원함
- 코드로부터 SQL 분리 지원
  - 소스코드로부터 SQL 문을 분리하여 별도의 repository(의미있는 문법의 XML)에 유지하고 이에 대한 빠른 참조구조를 내부적으로 구현하여 관리/유지보수/튜닝의 용이성을 보장함.
- 쿼리 실행의 입/출력 객체 바인딩/매핑 지원
  - 쿼리문의 입력 파라미터에 대한 바인딩과 실행결과 resultset의 가공(매핑)처리시 객체(VO, Map, List)수준의 자동화를 지원함
- Dynamic SQL 지원
  - 코드 작성, API 직접 사용 없이 입력 조건에 따른 동적인 쿼리문 변경을 지원함
- 다양한 DB 처리 지원
  - 기본 질의 외에 Batch SQL, Paging, Callable Statement, BLOB/CLOB 등 다양한 DB처리를 지원함

❑ Architecture

– iBATIS Architecture Flow



❑ Architecture

– iBATIS Architecture 구성요소

Architecture 구성 요소	설 명
Parameter Object (Input)	파라미터 객체는 JavaBean, Map, Primitive 객체로서, update문 내에 입력값을 셋팅하기 위해 사용되거나 쿼리문의 where절을 셋팅하기 위해 사용된다.
SqlMapConfig.xml	Data Mapper에서 사용하는 설정을 담고 있는 파일로서, DataSource, Data Mapper 및 Thread Management등과 같은 상세 설정 정보를 담고있다.
SqlMap. xml	하나의 SqlMap.xml파일은 많은 CacheMdel, Parameter Maps, Result Maps, Statements 정보를 담고있다.
SQL Map	Data Mapper프레임워크는 PreparedStatement인스턴스를 생성하고 제공된 파라미터 객체를 사용해서 파라미터를 셋팅한다. 그리고 statement를 실행하고 ResultSet으로부터 결과객체를 생성한다.
Mapped Statement	Mapped Statement는 Data Mapper 프레임워크의 핵심으로서, Parameter Maps과 Result Maps를 이용하여 SQL statement로 치환된다.
Result Object (Output)	결과 객체는 JavaBean, Map, Primitive객체로서, 쿼리문의 결과값을 담고 있다.

❑ 실습 단계

- library 설치 : iBATIS를 실행 하기 위한 파일 다운로드 및 설치
- SQL Map Config 파일 작성 : DB 연결정보, 설정정보등을 관리
- SQL Mapping 파일 작성 : 실제 쿼리문을 관리
- SqlMapClient 생성 파일 작성 : SqlMapClientBuilder를 이용 SqlMapClient인스턴스 생성
- DAO 파일 구현 : dao객체에서 SqlMapClient 실행 API 를 통해 실행

❑ library 설치

- http://www.mybatis.org 또는 http://code.google.com/p/mybatis/
- 현업에서 현재 사용되는 버전은 2.x 이므로 ibatis-2.3.4.726.zip을 다운로드 한다.
- 지정된 CLASSPATH나 웹 어플리케이션의 /WEB-INF/lib 에 jar파일을 배치한다.
- iBATIS는 자체 jar 파일 하나만으로도 실행될 수 있게끔 다른 파일과의 의존성은 없다.
- 만약 성능향상을 위해서 추가적인 라이브러리를 설치할 수 있다.

❑ library 설치

- 필수 library 및 추가적인 기능 또는 성능향상을 위한 library

이 름	옵 션	설 명
ibatis-2.3.4.726.jar	*필수	iBATIS 라이브러리(필수) http://www.mybatis.org/
DBMS JDBC 드라이버	*필수	DBMS사 에서 제공하는 JDBC 드라이버
commons-dbcp-1.x.jar	선택	database connection pooling 지원 라이브러리 dbcp 의존파일 commons-pool-1.x.jar 필요 http://commons.apache.org/
commons-logging-1.x.x.jar	*선택	Jakarta Commons Logging http://commons.apache.org/
log4j-1.x.x.jar	*선택	Log4j http://logging.apache.org/
oscache-2.x.x.jar	선택	중앙집중 또는 분산 캐시 지원 라이브러리 http://www.opensymphony.com/
cglib-nodep-2.x.x.jar	선택	Runtime Bytecode Enhancing 필요 시 http://cglib.sourceforge.net/

- 일반적으로 필수 library 외에 로깅, 커넥션풀을 위한 library도 설치한다.

❑ SQL Map Config 파일

– SQL Map XML Configuration파일은 데이터소스, 데이터 매퍼에 대한 설정, 쓰레드 관리와 같은 SQL Maps와 다른 옵션에 대한 설정을 제공하는 중앙집중적인 XML설정 파일을 사용해서 설정된다. 다음은 SQL Maps설정 파일을 작성한다. (sql-map-config-2.dtd)

- 소스폴더에 org/ddit/ibatis/SqlMapConfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
  <properties resource="org/ddit/ibatis/db.properties" />
  <settings />
  <transactionManager type="JDBC" commitRequired="false" >
    <dataSource type="SIMPLE" >
      <property name="JDBC.Driver" value="${driver}" />
      <property name="JDBC.ConnectionURL" value="${url}" />
      <property name="JDBC.Username" value="${username}" />
      <property name="JDBC.Password" value="${password}" />
    </dataSource>
  </transactionManager>
  <sqlMap resource="org/ddit/ibatis/mapping/member.xml" />
</sqlMapConfig>
```

❑ SQL Map Config 파일

– DB 연결정보를 별도의 파일로 작성한다.

• 소스폴더에 org/ddit/ibatis/db.properties

```
# org/ddit/ibatis/db.properties
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@127.0.0.1:1521:xe
username=ddit
password=java
```

❑ SQL Mapping 파일

– sql mapping 파일을 작성한다. (sql-map-2.dtd)

• 소스폴더에 org/ddit/ibatis/mapping/member.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap
    PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="member">
  <!-- // 완전한 이름에 대한 별칭 설정 -->
  <typeAlias alias="memberBean" type="org.ddit.member.model.MemberBean" />

  <!-- // resultClass에 완전한 클래스이름이나 별칭을 사용한다. -->
  <select id="getMemberList" resultClass="org.ddit.member.model.MemberBean">
    SELECT *
      FROM member
  </select>

  <select id="getMember" parameterClass="string"  resultClass="memberBean">
    SELECT *
      FROM member
     WHERE mem_id = #mem_id#
   </select>
```

❑ SQL Mapping 파일

– sql mapping 파일을 작성한다. (sql-map-2.dtd)

• 소스폴더에 org/ddit/ibatis/mapping/member.xml

```
  <insert id="insertMember" parameterClass="memberBean">
       INSERT INTO member
             ( mem_id,       mem_pwd,        mem_name,
               mem_regno1,   mem_regno2,     mem_birth,
               mem_zip,      mem_addr1,      mem_addr2,
               mem_hometel,  mem_comtel,     mem_cellphone,
               mem_email,    mem_reg_date )
       VALUES( #mem_id#,     #mem_pwd#,      #mem_name#,
               #mem_regno1#, #mem_regno2#,   #mem_birth#,
               #mem_zip#,    #mem_addr1#,    #mem_addr2#,
               #mem_hometel#,#mem_comtel#,   #mem_cellphone#,
               #mem_email#,  SYSDATE )
  </insert>

  <!-- // 업데이트 구문을 완성하세요!!!  -->

</sqlMap>
```

❑ SqlMapClient 생성 파일 작성

– 어플리케이션에서 iBATIS를 사용하기 위해 SqlMapClientBuilder를 이용 SqlMapClient인스턴스 생성

- 소스폴더에 org/ddit/ibatis/SqlDataMapper.java

```
package org.ddit.ibatis;
import java.io.Reader;
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;

public class SqlDataMapper {
  private static SqlMapClient sqlMapper;
  static {
    try {
      Reader reader = Resources.getResourceAsReader("org/ddit/ibatis/SqlMapConfig.xml");
      sqlMapper = SqlMapClientBuilder.buildSqlMapClient(reader);
      reader.close();
    } catch (IOException e) {
      throw new RuntimeException("iBATIS initialize error:" + e, e);
    }
  }
  public static SqlMapClient getSqlMapper() {
    return sqlMapper;
  }
} // class
```

❑ DAO 파일 구현

– 어플리케이션에서 iBATIS를 사용하기 위해 SqlMapClientBuilder를 이용 SqlMapClient인스턴스 생성

- 소스폴더에 org/ddit/member/dao/MemberDaoiBatis.java

```
package org.ddit.member.dao;
import java.sql.*;
import java.util.List;
import org.ddit.ibatis.SqlDataMapper;
import com.ibatis.sqlmap.client.SqlMapClient;

public class MemberDaoiBatis implements IMemberDao {
  // SqlMapClient를 SqlDataMappper로 부터 구한다
  private SqlMapClient mapper = SqlDataMapper.getSqlMapper();
  @Override
  public MemberBean getMember(Connection con, String memId) throws SQLException {
    MemberBean bean = (MemberBean)mapper.queryForObject("getMember",memId) ;
    return bean;
    // return  (MemberBean)mapClient.queryForObject("getMember",memId) ;
  }
  @Override
  public List<MemberBean> getMemberList(Connection con) throws SQLException {
    return mapper.queryForList("getMemberList");
  }
  ...
} // class
```

❑ JDBC 코드 비교

– JDBC를 직접 구현

```
public MemberBean getMember(Connection conn, String memId) throws SQLException {
  PreparedStatement pstmt = null;
  ResultSet rs = null;
  StringBuffer sb = new StringBuffer();
  try{
    sb.append(" Select * From member ");
    sb.append("  Where mem_id = ? ");
    pstmt = conn.prepareStatement(sb.toString());
    pstmt.setString(1, memId);                          // 파라미터(?)를 설정
    rs = pstmt.executeQuery();
    if(rs.next()){
      MemberBean bean = new MemberBean();               // bean을 초기화
      bean.setMem_id(rs.getString("mem_id"));
      bean.setMem_pwd(rs.getString("mem_pwd"));
      ....
      bean.setMem_email(rs.getString("mem_email"));
    } // if
    return bean;
  }finally{
    if(rs != null) try{ rs.close(); }catch(SQLException e){}
    if(pstmt != null) try{ pstmt.close(); }catch(SQLException e){}
  }
} // getMember
```

❑ JDBC 코드 비교

– iBATIS를 이용한 JDBC 구현

```
// dao 객체
public MemberBean getMember(Connection conn, String memId) throws SQLException {
  return (MemberBean)mapper.queryForObject("getMember",memId) ;
}

// iBATIS 매핑 구문
<select id="getMember" parameterClass="string"
                        resultClass="org.ddit.member.model.MemberBean">
    SELECT * FROM member
     WHERE mem_id = #mem_id#
</select>
```

❑ SQL Map Config 파일

– iBATIS 메인 설정 파일

- SqlMapClient 설정관련 상세 내역을 제어할 수 있는 메인 설정 파일로 주로 transaction 관리 관련 설정 및 다양한 옵션 설정, Sql Mapping 파일들에 대한 path 설정 등을 포함한다.
- DTD 파일 : sql-map-config-2.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig  PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
  <properties resource="" />
  <settings ... />
  <resultObjectFactory type="">
    <property name="" value="" />
  </resultObjectFactory>
  <typeAlias alias="" type="" />
  <typeHandler javaType="" jdbcType="" callback="" />
  <transactionManager type="">
    <dataSource type="">
      <property name="" value="" />
    </dataSource>
  </transactionManager>

  <sqlMap resource="" />
</sqlMapConfig>
```

- 설정 태그 설명

태그 요소	설 명
<sqlMapConfig>	iBATIS 설정 파일의 root 태그
<properties />	표준 자바 속성파일(name=value)을 이용하여 SQL Maps설정 파일 내에 참조될 수 있는 변수가 될 수 있고 모든 SQL Maps는 내부에서 ${key} 형식으로 참조된다. resource 속성으로 classpath, url 속성으로 유효한 URL 상에 있는 자원을 지정 가능하다.
<settings />	SqlMapClient 인스턴스를 위해 다양한 옵션과 최적화를 설정하도록 한다. 모든 속성은 선택사항(optional) 이다.
<resultObjectFactory />	SQL구문의 실행으로 결과객체를 생성하기 위한 factory클래스를 명시한다. 이 요소가 기술되지 않는다면 iBATIS는 결과객체를 생성하기 위해 일반적인 방법을 (class.newInstance()) 사용할 것이다.
<typeAlias />	풀 팩키지명에 대한 간략한 별칭 설정, 전역적으로 모든 SQL Mapping 파일에서 사용 가능
<typeHandler />	javaType ↔ jdbcType 간의 타입변환(prepared statement 의 파라메터 세팅/resultSet 의 값 얻기) 을 처리하는 typeHandler 구현체를 등록할 수 있다.
<transactionManager >	SQL Maps를 위한 트랜잭션 관리 서비스를 설정하도록 한다. type 속성에 트랜잭션 관리자 JDBC, JTA, EXTERNAL 중에 하나를 기술 한다.
<dataSource >	transactionManager 설정의 일부 영역으로 DataSource 에 대한 설정이다. type 속성으로 어떤 DataSourceFactory 를 사용할지 지시할 수 있는데, SIMPLE, DBCP, JNDI 의 세가지 설정이 가능하다.
<sqlMap />	명시적으로 각 SQL Mapping 파일을 포함하도록 설정한다. classpath (resource 속성으로 지정) 나 url (url 속성으로 지정) 상의 자원을 stream 형태로 로딩하게 된다.

- SQL Map Config 파일에서 setting 속성

이 름	기본값	설 명
maxRequests	512	같은 시간대에 SQL 문을 실행한 수 있는 thread 의 최대 갯수 지정. maxRequests="256",
maxSessions	128	주어진 시간에 활성화될 수 있는 session(또는 client) 수 지정. maxSessions="64"
maxTransactions	32	같은 시간대에 SqlMapClient.startTransaction() 에 들어갈 수 있는 최대 갯수 지정. maxTransactions="16",
cacheModelsEnabled	true	SqlMapClient 에 대한 모든 cacheModel 에 대한 사용 여부를 global 하게 지정. cacheModelsEnabled="true", (enabled)
lazyLoadingEnabled	true	SqlMapClient 에 대한 모든 lazy loading 에 대한 사용 여부를 global 하게 지정. lazyLoadingEnabled="true", (enabled)
enhancementEnabled	false	runtime bytecode enhancement 기술 사용 여부 지정. enhancementEnabled="true", (disabled)
useStatementNamespaces	false	mapped statements 에 대한 참조 시 namespace 조합 사용 여부 지정. true 인 경우 queryForObject("sqlMapName.statementName"); 과 같이 사용함. useStatementNamespaces="false", (disabled)
defaultStatementTimeout		모든 JDBC 쿼리에 대한 timeout 시간(초) 지정, 각 statement 의 설정으로 override 가능함. 모든 driver가 이 설정을 지원하는 것은 아님에 유의할 것. 지정하지 않는 경우 timeout 없음(cf. 각 statement 설정에 따라)
classInfoCacheEnabled	true	introspected(java 의 reflection API에 의해 내부 참조된) class의 캐쉬를 유지할지에 대한 설정 classInfoCacheEnabled="true", true (enabled)
statementCachingEnabled	true	prepared statement 의 local cache 를 유지할지에 대한 설정 statementCachingEnabled="true", true (enabled)

❑ SQL Mapping 파일

- iBATIS의 mapped statement 생성, sql문을 관리하며 객체와 컬럼간의 매핑정의, 캐시 설정 등을 지정한다.
- DTD 파일 : sql-map-2.dtd

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap   PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
      "http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="">
  <typeAlias alias="" type="" />
  <cacheModel id="" type="" />

  <resultMap id="" class="" />
  <parameterMap id="" class=""  />

  <sql id="" />

  <statement id="" />
  <insert id="" />
  <update id="" />
  <delete id="" />
  <select id="" />
  <procedure id="" />
</sqlMap>
```

- 설정 태그 설명

태그 요소	설 명
<sqlMap>	sql 매핑파일의 root 태그
<typeAlias />	현재 매핑 파일내에서 객체에 대한 간략한 alias 명을 지정함. (cf. 매우 자주 쓰이는 class 의 경우 SQL Map 설정파일 에 global 하게 등록하는 것이 좋음)
<cacheModel>	mapped statement 의 결과를 캐시하여 빠른 응답을 함. DBMS에 서비스 요청을 하지 않으므로 전체적인 성능 향상
<resultMap>	DB 칼럼명(select 문의 칼럼 alias) 과 결과 객체의 attribute 에 대한 매핑 및 추가 옵션을 정의함.
<parameterMap>	객체에 대한 prepared ststement에 대한 바인드 변수 매핑, 추가 옵션을 정의함.
<sql>	반복적으로 사용되는 sql 구문(일부 또는 전체)을 id에 등록하여 재사용함
<statement>	각 statement 타입에 따른 mapped statement 정의 요소 예시. 유형에 따라 insert/ update/ delete/ select/ procedure/ statement 요소 사용 가능

- iBATIS에서 미리 지정되어 있는 type alias

java type	alias
java.lang.String	string
int/ java.lang.Ineger	int/integer
long/ java.lang.Long	long
java.util.Date	date
java.util.List	list
java.util.ArrayList	arraylist
java.util.Map	map
java.util.HashMap	hashmap
java.util.Collection	collection
java.util.Iterator	iterator
기타	byte, short, double, decimal, object 등

  - type alias는 대소문자를 구분하지는 않는다.
    "string", "String", "StrinG" 모두 java.lang.String 에 매핑된다.

❑ Mapped Statement

- Data Mapper 사상의 핵심으로 Mapped Statement 는 parameterMaps(input) 과 resultMaps(output) 을 가질 수 있는 어떤 SQL 문이라도 될 수 있다. 단순하게는 파라미터나 결과에 대한 class를 직접적으로 설정할 수 있으며, in/out 매핑, 결과의 cache 유지 등에 대한 상세한 설정이 가능하다.
  - statement 구문

```
<statement id="statementName"
          [parameterClass="some.class.Name"]
          [resultClass="some.class.Name"]
          [parameterMap="nameOfParameterMap"]
          [resultMap="nameOfResultMap"]
          [cacheModel="nameOfCache"]
          [timeout="5"] >
    select * from PRODUCT where PRD_ID = [?|#propertyName#]
    order by [$simpleDynamic$]
</statement>
```

  - 위 statement 요소 위치에는 sql 문 유형에 따라 insert, update, delete, select, procedure 가 올 수 있다. 위에서 대괄호[] 내에 있는 속성은 선택 사항이다.
  - parameterClass 나 resultClass 속성으로 In/Out 에 대한 객체를 직접 지정할 수 있다. 이는 일반적으로 표준JavaBeans 객체 또는 Map (result 인 경우에는 Map 구현체 명시할 것) 이 될수 있으며, 단일 변수인 경우에는 primitive 래퍼 클래스로 지정할 수도 있다.

❑ Mapped Statement

- prepared statement 에 대한 바인드 변수 매핑을 위한 매핑 정의를 별도의 parameterMap 태그로 따로 지정한 경우 위의 parameterMap 속성으로 해당 id 를 지정한다. parameterMap 을 지정한 경우 sql 문의 바인드 변수 영역은 ? 로 작성하며 parameterMap 매핑 설정의 순서와 갯수가 맞아야 함에 유의한다.
- parameterMap 을 쓰지 않고 Inline Parameter 로 쓰는 것을 더 선호하면 #속성명# 과 같은 형태로 간략히 기술할 수 있다.
- resultSet 에 대한 결과 객체 매핑을 위해 별도의 resultMap 태그로 따로 지정한 경우 위의 resultMap 속성으로 해당 id 를 지정한다. resultMap 의 사용은 성능상, 상세 옵션의 적용 기능성 측면에서 추천하는 바이다.
- 한번 조회 한 결과를 캐쉬하기 위해 별도의 cacheModel 태그로 관련 설정을 한 경우, 대상 statement 에 해당 cacheModel id 를 위의 cacheModel 속성과 같이 지시한다. DB 데이터의 변경 시 cache 를 갱신할 수 있도록 cacheModel 설정의 flush 관련 설정에 유의해야 한다.
- 사용 DBMS 와 JDBC 드라이버가 지원하는 경우 timeout 을 명시할 수 있으며 이는 SQL Map 설정파일의 defaultStatementTimeout 에 우선한다.
- 위의 order by 절에 쓰인 $변수명$ 은 replaced Text 처리로 input 객체에 해당 변수에 대한 값으로 전달된 String을 SQL 의 동적 변경 요소로 대치(replace)하여 처리한다. 바인드 변수로 처리할 수 없는 order by 절이나 from 절, 혹은 전체 sql 문의 동적 변경 등을 위해 사용될 수 있으나, SQL Injection 의 보안 위험이 있고, 테이블이나 칼럼이 변경되는 경우 내부적으로 자동 매핑된 결과 객체에 대한 resultSet metadata 의 캐슁내역과 맞지 않아 오류를 일으킬 수 있으므로 유의한다.

❑ parameterMap

- parameterMap은 JavaBeans 객체에 prepared statement 에 대한 바인드 변수 매핑을 처리한다.
  하지만 parameterClass 나 Inline Parameter 에 비해 많이 사용되지 않는다.
  - 바인드되는 변수(?) 와 순서 및 갯수가 정확이 맞아야 함
  - Dynamic SQL (동적 구문을 위한) 사용 불가능
- 일부 저장프로시저(OUT 파라미터를 받아야 하는)를 제외한다면 inline parameter를 사용하는게 편하다.

```
<parameterMap id="parameterMapName" [class="com.domain.Product"]>
  <parameter property ="propertyName" [jdbcType="VARCHAR"] [javaType="string"]
             [nullValue="-9999"]
             [typeName="{REF or user-defined type}"]
             [resultMap=someResultMap]
             [mode=IN|OUT|INOUT]
             [typeHandler=someTypeHandler]
             [numericScale=2]/>
  <parameter …… />
  <parameter …… />
</parameterMap>
```

  - nullValue : 해당 프로퍼티가 null 일때 대체값이 아니라 해당 프로퍼티의 값이 지정된 값일 때 null 로 대체하라는 것임
    위의 예시에서 프로퍼티의 값이 만약 "-9999" 라면 null로 입력하라는 것임

❑ parameterMap

– parameterMap 예시

```
<parameterMap id="boardInsertMap" class="boardBean"  >
  <parameter property="bo_seq"   javaType="int" jdbcType="NUMERIC" />
  <parameter property="bo_title"   javaType="string" jdbcType="VARCHAR" />
  <parameter property="bo_writer"  javaType="string" jdbcType="VARCHAR" />
  <parameter property="bo_pwd"     javaType="string" jdbcType="VARCHAR" />
  <parameter property="bo_content" javaType="string" jdbcType="CLOB"/>
  <parameter property="bo_hit" javaType="int"    jdbcType="NUMERIC" nullValue="-9999" />
  <parameter property="bo_ip"  javaType="string" jdbcType="VARCHAR" nullValue="127.0.0.1" />
</parameterMap>

<insert id="insertBoard" parameterMap="board.boardInsertMap" >
    INSERT INTO board
           ( bo_seq, bo_title, bo_writer,
             bo_pwd, bo_content, bo_reg_date,
             bo_hit, bo_ip,  bo_type )
     VALUES (  ?, ?, ?,
               ?, ?, SYSDATE,
               ?, ?, 'board' )
</insert>
```

– 여기서 만약 bo_hit, bo_ip 프로퍼티가 "-9999", "127.0.0.1" 라면 null로 인서트됨

❑ Inline Paramerter

- prepared statement 에 대한 바인드 변수 매핑 처리를 위해서 가장 많이 사용되는 방식이다.
  자바객체의 프로퍼티에 대한 이름으로 간단하게 바인드 된다.
- 추가적으로 jdbcType, nullValue를 지정할 수 있다. (콜론 사용 ":")

```
#propertyName# - OR -
#propertyName:jdbcType# - OR -
#propertyName:jdbcType:nullValue#
```

- 상세한 옵션이 필요한 경우에도 사용 가능하다. (콤마 사용 "," 및 필요한 요소만)

```
#propertyName,javaType=?,jdbcType=?,mode=?,nullValue=?,handler=?,numericScale=?#
```

❑ java타입에 따른 이름

- Primitive 형(String, int,..) : 어떠한 이름도 사용가능 cf. #value#
- JavaBean 객체 (get/set) : 프로퍼티 이름으로 cf. #mem_id#
- Map : 맵의 key 이름으로 cf. #startRow#
- List/배열 : 배열상의 첨자번호 cf. #[0]#, #list[0]#

❑ Inline Paramerter

– Inline Parameter 예시

```
<insert id="insertBoard" parameterClass="boardBean" >
  INSERT INTO board
        ( bo_seq,     bo_title,
          bo_writer,  bo_pwd,
          bo_content, bo_reg_date,
          bo_hit,
          bo_ip,
          bo_type )
  VALUES (#bo_seq#,     #bo_title#,
          #bo_writer#,  #bo_pwd#,
          #bo_content:CLOB#, SYSDATE,
          #bo_hit:NUMERIC:-9999#,
          #bo_ip,javaType=string,jdbcType=VARCHAR,nullValue=127.0.0.1#,
          'board' )
</insert>
```

❑ resultMap

- SQL Maps의 중요한 요소이다. resultMap은 컬럼과 객체 프로퍼터간의 매핑, 타입 명시, null 값 대체, typeHandler 처리, complex property 매핑(JavaBean, List, Set, Collection)을 처리한다.

```
<resultMap id="resultMapName" class="some.domain.Class"
        [extends="parent-resultMap"]
        [groupBy="some property list"]>
       <result property="propertyName" column="COLUMN_NAME"
              [columnIndex="1"] [javaType="int"] [jdbcType="NUMERIC"]
              [nullValue="-999999"] [select="someOtherStatement"]
              [resultMap="someOtherResultMap"]
              [typeHandler="com.mydomain.MyTypehandler"]
              />
       <result ... />
</resultMap>
```

- extends : 상위 resultMap이 있는 경우, 상속을 받을 수 있다. 꼭 상위 class와 같을 필요는 없다.
- groupBy : 1:1관계, 1:N관계에서 사용되며, 특히 1:N관계일 경우 iBATIS가 레코드를 내부적으로 그룹화하기 위해 사용된다.
- jdbcType : java.sql.Types에 지정된 타입(VARCHAR, CHAR, NUMERIC, INTEGER, 등)
- nullValue : 해당 컬럼이 null인 경우 자바 프로퍼티에 지정한 값으로 대체한다.
- select : 해당 complex property 타입을 로드 하기 위해 지정된 mapped statement ID를 기술한다.
- resutlMap : 해당 complex property 타입을 로드 하기 위해 namespace이름이 포함된 resultMap ID를 기술한다. 1:N 관계인 겨우 iBATIS가 내부적으로 그룹화 하기위한 groupBy 속성을 사용한다.

❑ resultMap

– resultMap 예시

```
<typeAlias alias="boardBean" type="org.ddit.board.model.BoardBean"  />
<resultMap id="getBoardListMap" class="boardBean" >
  <result property="bo_seq"    column="BO_SEQ"    javaType="int"    jdbcType="NUMERIC" />
  <result property="bo_writer" column="BO_WRITER" javaType="string" jdbcType="VARCHAR" />
  <result property="bo_title"  column="BO_TITLE"  javaType="string" jdbcType="VARCHAR" />
  <result property="bo_pwd"    column="BO_PWD" javaType="string" jdbcType="VARCHAR" />
  <result property="bo_ip"     column="BO_IP"  javaType="string" jdbcType="VARCHAR" />
  <result property="bo_hit"    column="BO_HIT"     javaType="int" nullValue="0" />
  <result property="bo_grp_id"  column="BO_GRP_ID"  javaType="int" nullValue="0" />
  <result property="bo_grp_ord" column="BO_GRP_ORD" javaType="int" nullValue="0" />
  <result property="bo_grp_lvl" column="BO_GRP_LVL" javaType="int" nullValue="0" />
  <result property="bo_type"    column="BO_TYPE"  nullValue="board" />
  <result property="bo_reg_date" column="BO_REG_DATE" javaType="string" jdbcType="DATE" />
</resultMap>

<resultMap id="getBoardMap" class="boardBean" extends="boardListMap">
  <result property="bo_content" column="BO_CONTENT" javaType="string" jdbcType="CLOB"/>
</resultMap>
```

– mapped statement 에서 resultMap속성을 사용한다.

```
<select id="getBoard" parameterClass="int" resultMap="getBoardMap" >
<select id="getBoardList" parameterClass="java.util.Map" resultMap="getBoardListMap" >
```

❑ Don't forget about this

- Sql 매핑 파일에서 넘겨진 parameter 정보 설정을 위해 사용되는 지시자는?
  - ## : inline parameter , PreaparedStatement 의 ? 의 역할
  - $$: Replaced Text, 직접 문자열로 입력
  - ? : ParameterMap과 순서 매핑 할 때, 실제적으로 불편한 점이 많아 사용 안함
- inliner parameter(##) / Replaced Text($$) 사용 할 때 이름은?
  - Primitive 형(String, int,..) : 어떠한 이름도 사용가능 cf. #value#
  - JavaBean 객체 (get/set) : 프로퍼티 이름으로 cf. #mem_id#
  - Map : 맵의 key 이름으로 cf. #startRow#
  - List/배열 : 배열상의 첨자번호 cf. #[0]#, #list[0]#
- Replaced Text($$)는 언제 사용?
  - sql 구문중에 테이블명, 컬럼명등이 넘겨진 파라미터 정보에 있는 경우
  - 만약 Map에 "tb"라는 key에 값이 "member" 라는 문자열이 들어있다면
    Select * from #tb# ->Select * from ? , pstmt.setString(1, tb) => Select * from 'member'
    Select * from tb => Select * from member
- sql 구문과 Sql 매핑 파일 (xml문서)에서 공통적으로 사용되는 "<", ">" 등의 처리는?
  - < > 등으로 변환해서 sql 구문을 작성한다. cf. WHERE bo_count > 10
  - <![CDATA[...]]> 이용하여 작성한다. cf. WHERE bo_count <![CDATA[bo_count > 10]]>

❑ query (Select) 관련 메서드

- 일반적으로 sql 구문이 "select" 인 경우 사용된다.

- Object queryForObject (String id [, Object paramObject [, Object resultObject]]) throws SQLException
  - 하나의 레코드를 읽어서 결과객체에 담아 반환, 만약 레코드가 없는 경우 null 반환
  - resultObject : 자바객체를 생성하여 넘겨주면 iBATIS는 해당 객체에 결과정보를 저장한다.
    해당 클래스의 생성자가 private, 기본 생성자가 없는 경우, factory객체를 통해 생성되는 객체 등에 사용

- List queryForList (String id [, Object paramObject] [, int skip, int max]) throws SQLException
  - 여러 레코드를 읽어서 List객체에 담아 반환, 만약 레코드가 없는 경우 빈 List가 반환
  - skip : 읽은 레코드에서 건너뛸 레코드 행수
  - max : 반환할 최대 행수

- Map queryForMap (String id, Object paramObject, String keyProp [, String valueProp]) throws SQLException
  - 결과물을 map(key, value)으로 리턴한다.
  - keyProp : 결과 집합 내에서 다른 결과들과 구별 할 수 있는(key) 프로퍼티 설정 , 보통 PK와 관련된 프로퍼티
  - valueProp : 결과집합에서 어느 하나의 프로퍼티 정보만 map의 value에 넣을 때 사용

❑ non-query (insert, update, delete) 관련 메서드

- sql 구문이 "insert", "update", "delete" 무엇이든 실행 메서드는 아래의 어느 것을 사용해도 된다.
  단지, 의미적으로 매치 시키면 된다. 단, insert() 메서드는 반환이 Object인 것 주의

- Object insert (String id [, Object paramObject]) throws SQLException
  - 일반적인 sql구문을 사용한 경우 null 리턴한다.
  - 매핑파일의 <insert> 태그의 하위 <selectKey >태그를 사용한 경우에만 return 객체를 받을 수 있다.

- int update (String id [, Object paramObject]) throws SQLException
  - sql 구문에 의해 영향 받은 갯수 리턴

- int delete (String id [, Object paramObject]) throws SQLException
  - sql 구문에 의해 영향 받은 갯수 리턴
  - 내부적으로는 update() 메서드를 호출한다.

❑ iBATIS의 Logging

- iBATIS는 내부적으로 log factory를 사용하여 로깅 정보를 제공한다.
  내부적인 로그 팩토리는 다음의 로그 구현체중 하나로 로깅 정보를 위임한다.
  - Jakarta Commons Logging
  - Log4J
  - JDK Logging (JRE 1.4 이상이 요구됨)
- iBATIS 로그 팩토리는 위 구현체중 순서대로 체크를 해서 처음으로 찾아진 로깅 구현체를 사용하게 된다.
  위 구현체중 하나도 찾지 못하면 로깅은 사용되지 않는다.
- 다운로드
  - Jakarta Commons Logging : http://commons.apache.org/
    실제 Commons Logging은 추상 API이다. 즉, 이것은 구현체를 제공하지는 않는다.
  - Log4J : http://logging.apache.org/
    실제 로깅을 하는 구현체.
    기본 설정파일은 프로퍼티 파일인 log4j.properties나 xml형식의 log4j.xml 을 클래스패스에서 찾는다.
- 위의 2개의 jar 파일을 /WEB-INF/lib 폴더에 배치한다.

❑ Log4J 설정파일

– Log4j를 이용한 간단한 설정 예시 (위치 : /WEB-INF/classes/log4j.properties)

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout, file

# iBATIS & SQL Map logging level
#log4j.logger.com.ibatis=DEBUG
#log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG
log4j.logger.java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
log4j.logger.java.sql.ResultSet=DEBUG

# Console Appender
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p %C{1}:%L - %m%n

# DailyRolling File Appender
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
log4j.appender.file.Threshold=WARN
log4j.appender.file.File=C:\\ddit.log
log4j.appender.file.DatePattern='.'yyyyMMdd
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=[%-5p %d{yyyy-MM-dd HH:mm:ss} %l] %m%n
```

❑ Log4J 주요 컴포넌트

- Logger : Log4j 패키지의 핵심클래스이다.
  로그 메세지를 Appender에 전달하는 역할을 하며 debug(), info(), error()등의 로그 메서드를 사용한다.
- Appender : Logger로부터 전달된 메시지를 기록하는 클래스이다.
  콘솔에 출력할 것이지, 파일에 기록할 것인지, email로 보낼것인지등 다양한 방식을 지원한다.
- Layout : 로그를 기록할 때 어떤 형식으로 출력할 것이지에 대한 레이아웃에 대한 결정을 하는 클래스이다.

❑ 설정 파일 설명

- 설정파일 2번째라인에서 log4j의 전역레벨은 ERROR이며 Appender는 stdout, file 두개가 설정됐다.
  DEBUG로 설정한다면 다음 처럼 하면 된다.
  - log4j.rootLogger=DEBUG, stdout, file
  - 위처럼 바꾸면 iBATIS의 로그외에도 모든 로그가 리포팅된다.
- 전역레벨을 ERROR로 설정하고 개별클래스, 팩키지 단위로 레벨을 변경하려면 다음처럼 한다
  - log4j.logger.java.sql.Connection=DEBUG
  - log4j.logger.java.sql.PreparedStatement=DEBUG
  - log4j.logger.java.sql.Statement=DEBUG
  - log4j.logger.java.sql.ResultSet=DEBUG
  - 위처럼 바꾸면 실제 운영 중에도 로그가 리포팅되므로 개발 중에만 사용하고 주석처리 한다.

❑ Dynamic Mapped Statements

- JDBC 작업은 상황에 따라서 또는 조건에 따라서 쿼리는 변한다. 즉 프로그램 코딩영역에서 if, switch, for문을 사용하게 되는데 이를 위해 iBATIS는 동적인 구문작성을 위한 요소(태그)를 제공해 준다.

- Dynamic Element
  - dynamic태그는 다른 dynamic 구문의 결과를 전체적으로 묶을때 사용된다.
  - 다른 dynamic 구문이 나타나지 않았다면 dynamic 태그는 수행되지 않는다.
  - 이 태그를 사용할 때, removeFirstPrepend속성 기능이 강제로 수행된다.

태 그	설 명
<dynamic>	다른 dynamic 구문의 결과를 전체적으로 묶을때 사용된다.

  - 속성
    1. prepend – statement에 붙을 오버라이딩 가능한 SQL부분(옵션)
    2. open – 전체 을 체 결과물내용물을 열기 위한 문자열(옵션)
    3. close – 결과적인 전체내용물을 닫기 위한 문자열(옵션)

- 두개 이상의 테이블을 읽어야 하는 상황인데. 이것을 iBatis 에서는 어떻게 처리를 하는냐??
- 크게 3가지 정도로 나누겠습니다.
  - 선수조건 : 메인(master) VO객체에 sub 객체에 대한 정보를 가지게끔
  - 1:1 : sub VO가 맴버 변수로 main VO 객체
  - 1:N : sub VO 타입을 같는 List 맴버변수로 main VO에 있으면 iBatis 자체적으로 실제 해당 객체를 사용하고자 할때 작업을 합니다. (lazy-loading)
- 1. 두개의 구문을 연결
  - 각가의 구문이 별도로 존재
- 2. 두개이상의 테이블을 JOIN 으로 가져오는 방법
- 양쪽 테이블에 최소한 1개이상 존재하면 inner join 으로 한쪽(서브)테이블이 레코드가 없을수 있다면 outer join 단, 둘다 모두 resultMap을 선언하셔야 합니다.

❑ queryForMap 예제

- 지금까지는 iBATIS 에서 모델객체(bean, dto, vo)를 사용하여 파라미터를 받거나 결과를 저장하였다. 마찬가지로 Map을 사용하여 파라미터를 받거나 결과를 저장할 수 있다. 하지만 Map 을 사용하여 처리할 때 다음을 인지하고 사용해야 한다.
  - JavaBean객체를 사용할 때는 컴파일 시점에 해당 클래스의 프로퍼티가 있는지, 적합한지 체크를 한다.
  - Map을 사용할 때는 컴파일 시점이 아니라 실행시점에 하기 때문에 잠재적인 오류가 있을 수 있다.
  - Map은 모델객체 보다 자원을 많이 사용한다.
  - Map을 사용하면 해당 타입(Integer, String,..)을 보증하지 않는다.
- 하지만 Map을 사용해야 할 경우가 생기기 마련이다. 예를 들자면 Member빈이 있는데 월별 생일자 집계를 생성해야 할 경우 Member빈을 사용할 수는 없다. 이럴 경우 통계용 모델객체를 만드는 것이 좋겠지만 그렇지 않다면 Map 을 사용해야 한다.

❑ queryForMap 예제

– 실행 결과가 한 레코드인 경우 처리

```
<!-- // iBatis 구문 파일:회원들의 월별 생일자 집계용 -->
<select id="getMemberMonthTotal" resultClass="java.util.HashMap" >
  SELECT SUM( IF(MONTH(mem_bir) = 1, 1,0)) mem_jan,
         SUM( IF(MONTH(mem_bir) = 2, 1,0)) mem_feb,
         SUM( IF(MONTH(mem_bir) = 3, 1,0)) mem_mar,
         SUM( IF(MONTH(mem_bir) = 4, 1,0)) mem_apr,
         SUM( IF(MONTH(mem_bir) = 5, 1,0)) mem_may,
         SUM( IF(MONTH(mem_bir) = 6, 1,0)) mem_jun,
         SUM( IF(MONTH(mem_bir) = 7, 1,0)) mem_jly,
         SUM( IF(MONTH(mem_bir) = 8, 1,0)) mem_aug,
         SUM( IF(MONTH(mem_bir) = 9, 1,0)) mem_sep,
         SUM( IF(MONTH(mem_bir) = 10, 1,0)) mem_oct,
         SUM( IF(MONTH(mem_bir) = 11, 1,0)) mem_nov,
         SUM( IF(MONTH(mem_bir) = 12, 1,0)) mem_dec,
         COUNT(*) AS mem_total
    FROM member
</select>
<!-- // Oracle 을 사용할 경우 -->
<!-- // SELECT  SUM(DECODE(TO_NUMBER(TO_CHAR(mem_bir,'MM')), 1,1,0)) mem_jan,  -->
```

- 워의 query는 각 월별로 통계를 내는 것이며 컬럼명은 mem_jan, mem_feb, 이런식으로 처리하였으므로
  해당결과를 담을 자바객체가 없다. 그래서 resultClass를 HashMap으로 지정하였다.
  iBatis가 결과물을 생성해야 하므로 Map 이라는 인터페이스를 사용하면 안되고 Map의 구현체인 HashMap 을 사용한다.

❑ queryForMap 예제

```
//DAO 에서의 처리
public Map getMemberMonthTotal() throws SQLException {
  return  (java.util.Map)mapClient.queryForObject("member.getMemberMonthTotal");
}
```

```
// View 단에서 처리, oneMap은 Action단에 선언된 java.util.Map 변수
<c:if test="${not empty oneMap}">
  <tr>
    <td>${oneMap.mem_jan}</td>
    <td>${oneMap.mem_feb}</td>
    <td>${oneMap.mem_mar}</td>
    <td>${oneMap.mem_apr}</td>
    <td>${oneMap.mem_may}</td>
    <td>${oneMap.mem_jun}</td>
    <td>${oneMap.mem_jly}</td>
    <td>${oneMap.mem_aug}</td>
    <td>${oneMap.mem_sep}</td>
    <td>${oneMap.mem_oct}</td>
    <td>${oneMap.mem_nov}</td>
    <td>${oneMap.mem_dec}</td>
    <td>${oneMap.mem_total}</td>
  </tr>
</c:if>
```

❑ queryForMap 예제

– 결과 화면

월별 생일자 통계

1월	2월	3월	4월	5월	6월	7월	8월	9월	10월	11월	12월	Tot
6	0	4	1	2	0	0	1	1	3	3	3	24

❑ queryForMap 예제

- 실행 결과가 여러 레코드인 경우 처리
  - 위의 예제는 단순히 월변에 대한 집계일 뿐이었다. 즉 레코드가 하나였지만 지역 및 월별로 통계를 내려고 한다면 여러 행이 나올 것이다.

```
<!-- // iBatis 구문 파일:회원들의 지역별, 월별 생일자 집계용 -->
<select id="getMemberAreaMonthTotal" resultClass="java.util.HashMap" >
  SELECT LEFT(mem_add1,2) AS mem_address,
         SUM(IF(MONTH(mem_bir) = 1, 1,0)) mem_jan,
         SUM(IF(MONTH(mem_bir) = 2, 1,0)) mem_feb,
                ...
         SUM(IF(MONTH(mem_bir) = 11, 1,0)) mem_nov,
         SUM(IF(MONTH(mem_bir) = 12, 1,0)) mem_dec,
         COUNT(*) AS mem_total
    FROM member
   GROUP BY LEFT(mem_add1,2)
</select>
<!-- // Oracle 을 사용할 경우    -->
<!-- // SELECT SUBSTR(mem_add1,0 ,2) AS mem_address,
                SUM(DECODE(TO_NUMBER(TO_CHAR(mem_bir,'MM')), 1,1,0)) mem_jan,
                 ....
        GROUP BY SUBSTR(mem_add1,0 ,2)      -->
```

❑ queryForMap 예제

```
//DAO 에서의 처리
public Map getMemberAreaMonthTotal() throws SQLException {
  return mapClient.queryForMap("member.getMemberAreaMonthTotal", null,"mem_address");
}
```


```
//DAO 에서의 처리
public Map getMemberAreaMonthTotal() throws SQLException {
  return mapClient.queryForMap("member.getMemberAreaMonthTotal", null,"mem_address");
}
```


- 여기서는 queryForMap() 메서드를 사용했으며 인자에 따라 두 가지 형태가 있다.
  - queryForMap(String id, Object parameterObject, String keyProp)
  - queryForMap(String id, Object parameterObject, String keyProp, String valueProp)
- 세번째 인자는 Map의 키가 될 필드를 지정해 주어야 한다는 것이다. Map은 배열처럼 첨자로 동작하는게 아니라 키를 근거로 접근하기 때문에 key (다른것과 구분이 되는 것, 보통 PK 필드를 주면 된다.)가 있어야 한다. 여기서는 "mem_address" 로 했으며 네번재 인자는 결과물에서 어떤 한 개의 값만 필요한 경우 해당 필드를 적어주면 해당 값만 담는다.

❑ queryForMap 예제

- var 속성 을 이용해서

```
// View 단, var 속성 사용, allMap은 Action단에 선언되 java.util.Map 변수
<c:forEach items="${allMap}" var="eachMap" varStatus="st">
  <tr>
    <td>${st.count}</td>
    <td>${eachMap.value.mem_address}</td>
    <td>${eachMap.value.mem_jan}</td>
    <td>${eachMap.value.mem_feb}</td>
                ...
    <td>${eachMap.value.mem_nov}</td>
    <td>${eachMap.value.mem_dec}</td>
    <td>${eachMap.value.mem_total}</td>
  </tr>
</c:forEach>
```

- varStatus 속성 사용

```
// View 단, varStatus 속성 사용
<c:forEach items="${allMap}" varStatus="st">
<tr>
  <td>${st.count}</td>
  <td>${st.current.value.mem_address}</td>
  <td>${st.current.value.mem_jan}</td>
      ...
```

❑ queryForMap 예제

- 기타 표현 방법

```
// forEach 문 내에서
${allMap[eachMap.key].mem_address}

// 키를 알고 있는 경우
${allMap["대전"].mem_address}
```

- 결과 화면

지역별, 월별 생일자 통계

No.	지역	1월	2월	3월	4월	5월	6월	7월	8월	9월	10월	11월	12월	Tot
1	대구	0	0	0	0	0	0	0	0	0	2	0	0	2
2	대전	4	0	3	1	2	0	0	1	1	1	3	2	18
3	충남	1	0	1	0	0	0	0	0	0	0	0	1	3
4	서울	1	0	0	0	0	0	0	0	0	0	0	0	1

❑ CacheModel 적용

- 반복적으로 자주 사용되는 query 같은 경우 매번 DataBase와 I/O 가 발생하게 된다. 이러한 경우 첫 번째 실행된 Mapped Statement의 결과객체를 캐시하여 재사용 한다면 DataBase I/O가 발생하지 않아 어플리케이션의 전체적인 성능개선을 할 수 있다.
- 동일 Mapped Statement라도 파라미터값에 따라서 개별적으로 캐시 된다.

```
<typeAlias alias="memberBean" type="org.ddit.member.model.MemberBean" />
<cacheModel type="LRU" id="memberCache" readOnly="true" >
   <flushInterval hours="3" />
   <flushOnExecute statement="insertMember"/>
   <flushOnExecute statement="updateMember"/>
   <flushOnExecute statement="deleteMember"/>
   <property name="size" value="1000"/>
</cacheModel>

<select id="getMemberList"  resultClass="memberBean" cacheModel="memberCache">
   SELECT *
    FROM member
</select>
<select id="getMember" resultClass="memberBean" cacheModel="memberCache" >
   SELECT *
     FROM member
    WHERE mem_id = #mem_id#
</select>
```

❑ CacheModel 적용

- type 속성 : 캐시 모델이 저장되는 형태은 다음 네 가지 중 선택한다.
  - MEMORY - GC가 메모리에서 삭제할 때까지 캐시를 메모리에 저장해둔다.
  - FIFO - 캐시가 가득 차게 되면 가장 오래된 객체가 캐시에서 제거된다.
  - LRU - 객체를 캐시에서 자동으로 제거하기 위해 LRU 알고리즘을 (가장 최근에 적게 사용된) 사용한다.
  - OSCACHE - OpenSymphony Cache를 사용한다.
- flushInterval : 특정 시간단위마다 cache를 삭제하기 위한 옵션이다.
  - hours, minutes, seconds, milliseconds등을 지정할 수 있다.
- flushOnExecute : 지정한 mapped statement가 실행될때 cache를 삭제하는것을 의미한다.
  - Namespace를 사용하는 상황이라면 Namespace.statement 로 지정을 해줘야 한다.
- property의 size : 해당 cacheModel이 소유하고 있을 cache의 최대 항목 갯수를 의미한다.

- DML작업이 빈번한 경우에는 주의
- 한번의 결과객체가 너무 많은 경우 메모리 부족현상이 발생하므로 주의