**MyBatis**

# 目錄

[概念](#_概念) [MyBatis-Spring](#_MyBatis-Spring)

[實作](#_實作) [PageHelper](#_PageHelper)

[多參數條件](#_多參數條件) [MyBatipse – MyBatis Tool for Eclipse](#_MyBatipse_–_MyBatis)

[傳回自動產生的Key (GeneratedKey)](#_傳回自動產生的Key_(GeneratedKey)) [參考資料](#_參考資料)

[JOIN處理](#_JOIN處理)

[顯示SQL-use Log4j](#_顯示SQL-use_Log4j)

[MyBatis動態SQL](#_MyBatis動態SQL)

[MyBatis進階設定](#_MyBatis進階設定)

[映射檔(xxxMapper.xml) 標籤細節](#_映射檔(xxxMapper.xml)_標籤細節)

[MyBatis API for Java](#_MyBatis_API_for)

[MyBatis-@Annotation](#_MyBatis-@Annotation)

[SQL Builder – SQL類別](#_SQL_Builder_–)

# 概念

角色

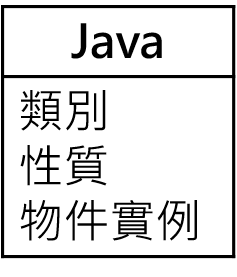
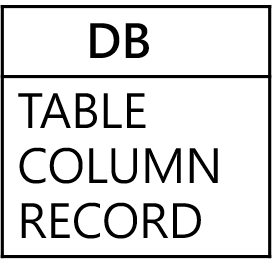
MyBatis在專案裡就是Hibernate的角色，所以一般來說，一個專案只會使用Hibernate**或**Mybatis，不會同時使用

持久層(永續層) : 實作**ORM**技術

**DAO層**

**持久層**

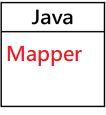
**DataBase**

Hibernate的ORM

MyBatis優點：**效率**較Hibernate好

註 : 但業界Hibernate用的較多(三大框架之一)

MyBatis特色 : 將Java方法(Method)與**SQL**語句關聯



MyBatis架構圖

**核心組態檔**

**(mybatis-config.xml)**

**映射檔**

**(xxxMapper.xml)**

**映射器介面**

**(xxxMapper.java)**

**JavaBean**

**(xxxBean.java)**

MyBatis vs. Hibernate(系統架構)

|  |  |  |
| --- | --- | --- |
|  | MyBatis | Hibernate |
| 核心組態檔 | mybatis-config.xml | hibernate.cfg.xml |
| 映射檔 | xxxMapper.xml | xxxBean.hbm.xml |
| 資料容器 | JavaBean | JavaBean |
| 資料存取物件 | xxxMapper.java(使用MyBatis-Spring時) | DAO |

# 實作

## 0. 撰寫JavaBean

ex. MemberBean

## 1. 加入jar檔

1.1 DB Driver

1.2 MyBatis : **mybatis-x.y.z.jar**

## 2. 建立及撰寫組態檔(mybatis-config.xml)

**\*註 : xml檔管理的良好習慣 : 利用Source Folder**

**專案 🡪 右鍵 🡪 New 🡪 Other 🡪 Source Folder 🡪 輸入resource 🡪 Finish**

**2.1 建立組態檔 : 在resource下, 新增mybatis-config.xml**

**2.2 撰寫組態檔**

<?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>

<!-- 設定類別別名, 僅縮短名稱用 -->

<typeAliases>

<typeAlias alias=*"Member"* type=*"model.pojo.MemberBean"* />

<typeAlias alias=*"Boolean"* type=*"java.lang.Boolean"* />

<typeAlias alias=*"String"* type=*"java.lang.String"* />

<typeAlias alias=*"Timestamp"* type=*"java.sql.Timestamp"* />

</typeAliases>

<!-- 環境設定 -->

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<!-- DataSource設定 -->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/ipo"* />

<property name=*"username"* value=*"root"* />

<property name=*"password"* value=*"password"* />

</dataSource>

<!-- 用JNDI設定DataSource -->

<!-- <dataSource type="JNDI"> -->

<!-- <property name="data\_source" value="java:comp/env/jdbc/ipo" /> -->

<!-- <property name="initial\_context" value="java:comp/env/jdbc/ipo" /> -->

<!-- </dataSource> -->

</environment>

</environments>

<!-- 指定映射檔(xxxMapper.xml) -->

<mappers>

<mapper resource=*"mapper/MemberMapper.xml"* /> <--指定xxxMapper.xml位置和檔名

</mappers>

</configuration>

## 3. 建立及撰寫映射檔

3.1 建立 : 依步驟2.2所指定映射檔路徑、檔名, 建立**xxxMapper.xml**

3.2 撰寫映射檔

3.2.1 聲明使用mybatis-3-mapper.dtd

3.2.2 設定resultMap (JavaBean性質與DB欄位映射)

\*註 : MyBatis有自動映射機制, 前提是性質與欄位**名稱一樣(忽略大小寫)**;另可在mybatis-config.xml<settings>內加入<setting

name=*"mapUnderscoreToCamelCase"* value=*"true"* />啟用Java駝峰式命名與DB典型式命名自動映射

(ex. createTime<-->CREATE\_TIME)

3.2.3 撰寫DML(INSERT、DELETE、UPDATE、SELECT)

|  |  |
| --- | --- |
| **MySQL建議的資料型態** | |
| **Data Type Mapping** | |
| **MySQL** | **Java** |
| **BIT(1)** | **Boolean** |
| **INT,INTEGER** | **Integer** |
| **BIGINT** | **Long** |
| **FLOAT** | **Float** |
| **DOUBLE** | **Double** |
| **DECIMAL** | **BigDecimal** |
| **DATE** | **java.sql.Date** |
| **DATETIME** | **java.sql.Timestamp** |
| **TIMESTAMP** | **java.sql.Timestamp** |
| **VARCHAR** | **String** |
| **BLOB** | **byte[]** |
| **TEXT** | **String** |

XML格式注意

小於 : **&lt;**

大於 : **&gt;**

另可用**<![CDATA[**xxx**]]>**表示大小於

<?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=*"model.mapper.MemberMapper"*> <--指定映射器

<resultMap id=*"memberMap"* type=*"Member"*> <--指定JavaBean全名, 或<typeAlias>設定裡的類別別名

<id property=*"id"* column=*"ID"* javaType=*"Integer"* jdbcType=*"INTEGER"* />

<result property=*"account"* column=*"ACCOUNT"* javaType=*"String"* jdbcType=*"VARCHAR"* />

<result property=*"password"* column=*"PASSWORD"* javaType=*"String"* jdbcType=*"VARCHAR"* />

<result property=*"nickname"* column=*"NICKNAME"* javaType=*"String"* jdbcType=*"VARCHAR"* />

<result property=*"createTime"* column=*"CREATE\_TIME"* javaType=*"Timestamp"* jdbcType=*"TIMESTAMP"* />

<result property=*"pass"* column=*"PASS"* javaType=*"Boolean"* jdbcType=*"BIT"* />

<result property=*"sessionId"* column=*"SESSION\_ID"* javaType=*"String"* jdbcType=*"VARCHAR"* />

</resultMap>

<insert id=*"insert"* parameterType=*"Member"*>

INSERT INTO MEMBER

(ACCOUNT,

PASSWORD,

NICKNAME,

CREATE\_TIME,

PASS,

SESSION\_ID)

VALUES

(#{account},

#{password},

#{nickname},

#{createTime},

#{pass},

#{sessionId})

</insert>

<delete id=*"delete"* parameterType=*"Integer"*>

DELETE FROM MEMBER WHERE ID = #{id}

</delete>

<update id=*"update"* parameterType=*"Member"*>

UPDATE MEMBER SET

ACCOUNT = #{account},

PASSWORD = #{password},

NICKNAME= #{nickname},

CREATE\_TIME = #{createTime},

PASS = #{pass},

SESSION\_ID = #{sessionId}

WHERE

ID = #{id}

</update>

<select id=*"selectByKey"* parameterType=*"Integer"* resultType=*"Member"*>

SELECT \* FROM MEMBER WHERE ID = #{id}

</select>

<select id=*"selectAll"* resultMap=*"memberMap"*>

SELECT \* FROM MEMBER

</select>

</mapper>

## 4. 建立及撰寫映射器(Mapper)介面

4.1 建立 : 依步驟3.2 **namespace**所指定套件、名稱, 建立interface **xxxMapper.java**

4.2 撰寫 : 步驟3.2 定義之DML SQL, 撰寫對應方法, 規則如下

回傳型態 : **resultType**="xxx” 或 **resultMap**所對應的類別

\*註 : SELECT回傳多筆時, 用List<xxx>

方法名稱 : **id**="xxx"

參數型態 : **parameterType**="xxx"

參數名稱 : **#{**xxx**}**. 若參數是一個JavaBean, 名稱則不受限

範例

**public** **interface** MemberMapper {

**int** insert(MemberBean member);

**int** delete(Integer id);

**int** update(MemberBean member);

MemberBean selectByKey(Integer id);

List<MemberBean> selectAll();

}

## 5. 建立及撰寫產生Session類別

5.1 建立 : 新增套件util, 建立類別MyBatisUtil.java

5.2 撰寫

**import** java.io.InputStream;

**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** MyBatisUtil {

**private** **static** SqlSessionFactory *sessionFactory*;

**private** MyBatisUtil() {

}

**private** **static** SqlSessionFactory getSessionFactory() {

**if** (*sessionFactory* == **null**) {

**try**(InputStream is = Resources.*getResourceAsStream*("mybatis-config.xml")){

*sessionFactory* = **new** SqlSessionFactoryBuilder().build(is);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**return** *sessionFactory*;

}

**public** **static** SqlSession getSession() {

**return** *getSessionFactory*().openSession();

}

}

## 6. 建立及撰寫DAO (使用MyBatis-Spring時省略)

**public** **class** MemberDAO {

**public** **int** insert(MemberBean bean) {

**基本步驟**

1. **透過MyBatisUtil取得SqlSession**
2. **透過SqlSession取得xxxMapper**
3. **呼叫xxxMapper對應的Method**
4. **確認交易(select時則略)**
5. **回傳結果**

**try**(SqlSession session = MyBatisUtil.*getSession*()){

MemberMapper mapper = session.getMapper(MemberMapper.**class**);

**int** result = mapper.insert(bean);

session.commit();

**return** result;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** -1;

}

**public** **int** delete(Integer id) {

**try**(SqlSession session = MyBatisUtil.*getSession*()){

MemberMapper mapper = session.getMapper(MemberMapper.**class**);

**int** result = mapper.delete(id);

session.commit();

**return** result;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** -1;

}

**public** **int** update(MemberBean bean) {

**try**(SqlSession session = MyBatisUtil.*getSession*()){

MemberMapper mapper = session.getMapper(MemberMapper.**class**);

**int** result = mapper.update(bean);

session.commit();

**return** result;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** -1;

}

**public** MemberBean selectByKey(Integer id) {

**try**(SqlSession session = MyBatisUtil.*getSession*()){

MemberMapper mapper = session.getMapper(MemberMapper.**class**);

**return** mapper.selectByKey(id);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

**public** List<MemberBean> selectAll() {

**try**(SqlSession session = MyBatisUtil.*getSession*()){

MemberMapper mapper = session.getMapper(MemberMapper.**class**);

**return** mapper.selectAll();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

}

# 多參數條件

## 方式1 : 使用JavaBean

**MemberMapper.xml**

<select id=*"multiParameter"* parameterType=*"model.pojo.MemberBean"*>

SELECT \* FROM MEMBER

WHERE

ACCOUNT LIKE #{account} AND

NICKNAME LIKE #{nickname}

</select>

**MemberMapper.java**

List<MemberBean> multiParameter(MemberBean member);

## 方式2 : 使用@Param

**\*註 : 套件為org.apache.ibatis.annotations.Param**

**MemberMapper.xml**

<select id=*"multiParameter"*>

SELECT \* FROM MEMBER

WHERE

CREATE\_TIME BETWEEN #{startTime} AND #{endTime}

</select>

**MemberMapper.java**

List<MemberBean> multiParameter(@Param("startTime") Timestamp startTime

, @Param("endTime") Timestamp endTime);

# 傳回自動產生的Key (GeneratedKey)

## 單筆INSERT

**MemberMapper.xml**

<insert id=*"insert"* parameterType=*"Member"* useGeneratedKeys=*"true"* keyProperty=*"id"* keyColumn=*"ID"*>

INSERT INTO MEMBER

(ACCOUNT,

PASSWORD,

**註 : 亦可在mybatis-config.xml裡設定(影響所有映射器)**

<settings>

<setting name=*"useGeneratedKeys"* value=*"true"* />

</settings>

NICKNAME,

CREATE\_TIME,

PASS,

SESSION\_ID)

VALUES

(#{account},

#{password},

#{nickname},

#{createTime},

#{pass},

#{sessionId})

</insert>

**in java code**

MemberBean member = **new** MemberBean(…);

memberMapper.insert(member);

Integer id = member.getId();

**\*註 : 自動產生的Key會存回member物件**

## 多筆INSERT(BATCH INSERT)

**MemberMapper.xml**

<insert id=*"insert"* parameterType=*"List"* useGeneratedKeys=*"true"* keyProperty=*"id"* keyColumn=*"ID"*>

INSERT m\_member

(m\_account,

m\_password,

m\_nickname,

m\_createTime,

m\_pass,

m\_sessionID)

VALUES

<foreach item=*"member"* collection=*"memberList"* separator=*","*>

(#{member.m\_account},

#{member.m\_password},

#{member.m\_nickname},

#{member.m\_createTime},

#{member.m\_pass},

#{member.m\_sessionID})

</foreach>

</insert>

**MemberMapper.java**

**int** insertBatch(@Param("memberList") List<MemberBean> memberList);

# JOIN處理

## 0. 確認關聯為一對一或一對多

## 1. 修改主JavaBean : 加入關聯Bean, 和setter、getter

**一對一 : private** OrderBean order;

**一對多 : private** List<OrderBean> orderList;

## 2. 修改映射檔(xxxMapepr.xml)

### 2.1 在<resultMap>裡加入..

**一對一** <association property=*"order"* javaType=*"model.pojo.OrderBean"*>

<id property=*"no"* column=*"NO"* />

<result property=*"itemName"* column=*"ITEM\_NAME"* />

<result property=*"amount"* column=*"AMOUNT"* />

...

</association>

**一對多** <collection property=*"orderList"* javaType=*"ArrayList"* ofType=*"model.pojo.OrderBean"*>

<id property=*"no"* column=*"NO"* />

<result property=*"itemName"* column=*"ITEM\_NAME"* />

<result property=*"amount"* column=*"AMOUNT"* />

...

</collection>

### 2.2 修改<select>裡的SQL

<select id=*"selectAll"* resultType=*"Member"*>

SELECT M.\*, O.NO AS NO, O.ITEM\_NAME AS ITEM\_NAME, O.AMOUNT AS AMOUNT

FROM MEMBER M

LEFT JOIN ORDER O

ON M.ID = O.M\_ID

</select>**\*註 : 關聯TABLE的欄位別名需與步驟2.1設定** column=*"xxx"***相符**

## 經由再次select取得關聯Bean

**ex. MemberMapper.xml**

<resultMap id=*"MemberMap"* type=*"Member"*>

<id property=*"id"* column=*"ID"* javaType=*"Integer"* jdbcType=*"INTEGER"*/>

<result property=*"account"* column=*"ACCOUNT"* javaType=*"Stirng"* jdbcType=*"VARCHAR"* />

…

<association property=*"order"* column=*"ID"* javaType=*"Order"* select=*"selectOrderByKey"* fetchType=*"lazy"*/>

</resultMap>

<select id=*"selectAllMember"* resultMap=*"MemberMap"*>

SELECT \* FROM MEMBER

</select>

<select id=*"selectOrderByKey"* resultType=*"Order"*>

SELECT \* FROM ORDER WHERE ID = #{id}

</select>

# 顯示SQL-use Log4j

## 1. 加入jar檔 : log4j-x.y.zz.jar

MyBatis還支援許多Log框架

* SLF4J
* Apache Commons Logging
* Log4j 2
* JDK logging

## 2. 建立log4j組態檔

預設 : classpath\log4j.properties

自訂 : 在程式裡則需以PropertyConfigurator.configure("pathAndFileName"), 指定載入

## 3. 撰寫log4j組態設定檔

log4j.rootLogger=FATAL, mybatisAppender <--自訂Appender名稱

log4j.logger.映射器介面全名=TRACE <--監控單個Mapper

#ex.. log4j.logger.model.mapper.MemberMapper

log4j.logger.映射器所在套件=TRACE <--監控多個Mapper(套件下全部Mapper)

#ex.. log4j.logger.model.mapper

log4j.appender.mybatisAppender=org.apache.log4j.ConsoleAppender

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

log4j.appender.mybatisAppender.layout.ConversionPattern=%m%n

## 4. 加入至MyBatis組態檔 : 在mybaits-config.xml, <configuration>裡加入..

<settings>

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

</settings>

# MyBatis動態SQL

**動態TABLE及動態COLUMN條件 : ${TABLE\_NAME}、${COLUMN\_NAME}**

<select id=*"dynamicTableNameOrColumnName"*>

SELECT \*

FROM ${tableName}

ORDER BY ${columnName}

</select>

**多選條件 : <if test="xxx">**

<select id=*"ifExample"* parameterType=*"Member"* resultMap=*"memberMap"*>

SELECT \*

FROM MEMBER

WHERE 1 = 1

<if test=*"id != 0"*>AND ID = #{id}</if>

<if test=*"account != null"*>AND ACCOUNT = #{account}</if>

</select>

**\*註 : test="xxx"裡的注意事項**

**1. 或(or)、及(and)需用全小寫表示**

**2. 若對"基本型態"判斷不可用==null、!=null (基本型態永遠不為null)**

**單選條件 : <choose><when><otherwise>**

<select id=*"chooseExample"* parameterType=*"Member"* resultMap=*"memberMap"*>

SELECT \*

FROM MEMBER

WHERE

<choose>

<when test=*"id != null"*> ID = #{id}</when>

<when test=*"account != null"*>AND ACCOUNT = #{account}</when>

<when test=*"password != null"*>AND PASSWORD = #{password}</when>

<otherwise>AND NICKNAME = #{nickname}</otherwise>

</choose>

</select>

**動態條件 : <where>**

**\*註 : 標籤內有一個以上條件成立, 才會加入"WHERE", 且會自動刪除多餘的token(ex.. AND、OR)**

<select id=*"whereExample"* parameterType=*"Member"* resultMap=*"memberMap"*>

SELECT \*

FROM MEMBER

<where>

<if test=*"id != null"*>ID = #{id}</if>

<if test=*"account != null"*>AND ACCOUNT = #{account}</if>

</where>

</select>

**更活的條件 : <trim prefix="xxx" suffix="xxx" prefixOverrides="xxx" suffixOverrides="xxx">**

**\*註**

**prefix : 執行時自動加入的前置詞**

**suffix : 執行時自動加入的後綴詞**

**prefixOverrides : 需被去除的多餘前置詞**

**suffixOverrides : 需被去除的多餘後綴詞**

**EX1**

<select id=*"trim1Example"* parameterType=*"Member"* resultMap=*"memberMap"*>

SELECT \*

FROM MEMBER

<trim prefix=*"WHERE"* prefixOverrides=*"AND | OR"* suffixOverrides=*","*>

<if test=*"id != null"*>AND ID = #{id}</if>

<if test=*"account != null"*>OR ACCOUNT = #{account}</if>

</trim>

</select>

**EX2**

<update id=*"trim2Example"* parameterType=*"Member"*>

UPDATE MEMBER

<trim prefix=*"SET"* suffixOverrides=*","*>

<if test=*"account != null"*>ACCOUNT = #{account},</if>

<if test=*"password != null"*>PASSWORD = #{password},</if>

<if test=*"nickname != null"*>NICKNAME = #{nickname},</if>

<if test=*"createTime != null"*>CREATE\_TIME = #{createTime},</if>

<if test=*"pass != null"*>PASS = #{pass},</if>

<if test=*"sessiodId != null"*>SESSION\_ID = #{sessionId}</if>

</trim>

WHERE

ID = #{id}

</update>

**指定值 : <set>**

<update id=*"setExample"* parameterType=*"Member"*>

UPDATE MEMBER

<set>

<if test=*"account != null"*>ACCOUNT = #{account},</if>

<if test=*"password != null"*>PASSWORD = #{password},</if>

<if test=*"nickname != null"*>NICKNAME = #{nickname},</if>

<if test=*"createTime != null"*>CREATE\_TIME = #{createTime},</if>

<if test=*"pass != null"*>PASS = #{pass},</if>

<if test=*"sessiodId != null"*>SESSION\_ID = #{sessionId}</if>

</set>

WHERE

ID = #{id}

</update>

**迭代迴圈 : <foreach collection="xxx" index="xxx" item="xxx" open="xxx" separator="x" close="xxx">**

**\*註**

**1. collection 可指定Array、Collection、Map等可迭代物件**

**2. 映射器(xxxMapper.java)裡需以@Param("xxx")對應collection="xxx"**

**EX1**

**MemberMapper.xml**

<select id=*"foreachExample"* resultMap=*"memberMap"*>

SELECT \*

FROM MEMBER

WHERE

ID IN

<foreach collection=*"idlist"* index=*"index"* item=*"item"* open=*"("* separator=*","* close=*")"*>

#{item}

</foreach>

</select>

**MemberMapper.java**

List<MemberBean> foreachExample(@Param("idList") **int**[] ids);

**EX2. 配合動態COLUMN使用**

<select id=*"foreachAndDynamicColumnExample"* resultMap=*"memberMap"*>

SELECT \*

FROM m\_member

WHERE

<foreach collection=*"dynamicColumnList"* index=*"index"* item=*"item"* separator=*"AND"*>

${item.columnName} LIKE #{item.value}

</foreach>

</select>

**綁定 : <bind>**

**\*註 : 綁定一個值, 即宣告一個常數, 使用OGNL表示式**

<select id=*"bindExample"* resultType=*"Member"*>

<bind name=*"account"* value=*"'%' + \_parameter.getAccount() + '%'"* />

SELECT \*

FROM MEMBER

WHERE

ACCOUNT LIKE #{account}

</select>

**DBMS TYPE : \_databaseId**

**\*註 : 對應到mybatis-config.xml裡<databaseIdProvider>取得之值, 可應用於多資料源管理**

**ex.**

**mybatis-config.xml**

<databaseIdProvider type=*"DB\_VENDOR"*>

<property name=*"Oracle"* value=*"oracle"* />

<property name=*"SQl Server"* value=*"sqlserver"*/>

<property name=*"MySQL"* value=*"mysql"* />

<property name=*"DB2"* value=*"db2"* />

<property name=*"Adaptive Server Enterprise"* value=*"sybase"* />

</databaseIdProvider>

**xxxMapper.xml**

<select id=*"databaseIdExample"* resultType=*"String"*>

<if test=*"\_databaseId == 'mysql'"*>

SELECT NICKNAME FROM MEMBER

</if>

<if test=*"\_databaseId != 'mysql'"*>

SELECT 'ERROR'

</if>

</select>

**CALL STORED PROCEDURE : <select statementType="CALLABLE"> {…}**

<select id=*"callStoredProcedureExample"* statementType=*"CALLABLE"* resultType=*"回傳型態依內儲程序"*>

{CALL procedureName(#{param1}, #{param2})}

</select>

# MyBatis進階設定

## mybatis-config.xml的結構

**\*註 : 順序不可變動**

**configuration—根目錄**

* + - * **properties—外部設定檔來源**
      * **settings—MyBatis的系統設定**

**setting**

* + - * **typeAliases—類別別名**

**typeAlias**

**package**

* + - * **typeHandlers—型態處理器**

**typeHandler**

**package**

* + - * **objectFactory—物件工廠**
      * **plugins—攔截器(外掛)**

**plugin**

* + - * **environments—環境設定**

**environment**

**transactionManager—交易管理者**

**dataSource—資料源**

* + - * **databaseIdProvider—DBMS廠商ID提供者**
      * **mappers—映射檔(xxxMapper.xml)來源**

## properties

**ex. 把JDBC設定值寫在jdbc.properties, 用mybatis-config.xml裡的properties引入**

**jdbc.properties**

jdbc.**driverClassName**=com.mysql.jdbc.Driver

jdbc.**url**=jdbc:mysql://localhost:3306/ipo

jdbc.**username**=root

jdbc.**password**=0000

**mybatis.config.xml**

<properties resource=*"****jdbc.properties****"* />

…

<dataSource type=*"POOLED"*>

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

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

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

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

</dataSource>

## Setting

### 映射相關

**autoMappingBehavior—設定自動映射(性質<一>欄位)模式**

NONE:關閉自動映射 PARTIAL(預設):只自動映射未被設定項 FULL:全自動映射

**autoMappingUnknownColumnBehavior—設定發現未知性質或欄位時之行為模式**

NONE:無 WARNING:輸出警告(WARN)log FAILING:拋出例外(SqlSessionException)

**mapUnderscoreToCamelCase—允許Java命名方式(aColumn)自動映射DB典型命名方式(A\_COLUMN)**

**useActualParamName—允許Mapper.java<一>Mapper.xml裡的參數名稱使用真實參數名映射**

### 查詢相關

**multipleResultSetsEnabled—允許傳回多ResultSet(driver需支援. 預設為true)**

**useGeneratedKeys—使用返回自動生成key**

**useColumnLabel—使用Column "Label"取代Column Name (不同的DBMS會有不同的支援)**

**jdbcTypeForNull—設定預設jdbcType. (無指定jdbcType又遇null時會自動補上)**

**defaultScriptingLanguage—設定動態 SQL 生成的預設語言**

**callSettersOnNulls—當查詢結果為null時, 是否呼叫setter**

**returnInstanceForEmptyRow—當查詢結果為null時, 是否傳回一空的實例**

### 延遲載入相關

**lazyLoadingEnabled—開啟延遲載入**

**aggressiveLazyLoading—開啟延遲載入細節. 若開啟, 任何延遲載入相關屬性都會被載入**

**proxyFactory—設定Mybatis 建立具有延遲載入的物件所用的代理工具(CGLIB | JAVASSIST)**

**lazyLoadTriggerMethods—設定哪個物件的哪個方法觸發延遲載入**

### 系統相關

**cacheEnabled—開啟cache**

**defaultStatementTimeout—設定driver對DB的等待時間, 值需為正整數**

**defaultFetchSize—設定fetch大小. (大量取值時可能發生OOM, 藉由此設定可稍微改善. 預設為沒設定)**

**localCacheScope—設定利用本機cache模式 (防止循環參考和加速含有子查詢的查詢)**

SESSION:會暫存一個session,用來處理所有查詢 STATEMENT:用local session執行查詢

### Log相關

**logPrefix—設定MyBatis輸出Log的前置詞**

**logImpl—設定實作Log的框架**

### 其他

**defaultExecutorType—設定執行器(executor)**

SIMPLE:一般的執行器 REUSE:會重複使用SQL語句(Prepared Statement) BATCH:會重複使用SQL語句, 且批量執行

**safeRowBoundsEnabled—允許使用分界(分頁)RowBounds. false為允許**

**safeResultHandlerEnabled—允許使用結果處理器(ResultHandler)**

**vfsImpl—設定VFS(Virual File System)實作者**

## typeAliases

**一次設一個 :** <typeAlias alias=*"Member"* type=*"model.pojo.MemberBean"* />

**一次設多個(指定套件) :** <package name=*"java.model.pojo"* />

## environment

**transactionManager—設定交易管理者**

JDBC : 官方解釋為依JDBC的設定, 但MySQL的JDBC Driver預設是關閉交易控制, 測試結果卻是會自動開啟交易控制, 剛好**相反**

MANAGED : 交由容器(Container)管理, 預設會關閉連線, 可用<property name="closeConnection" value="false"/>更改

**\*註 : 使用MyBatis-Spring時不需設定**

**dataSource—設定資料源**

UNPOOLED : 無連線池連線(較慢), 被用到時才開啟, 用完時即關閉

POOLED : 連線池連線(較快), 重複使用連線, 省去了建立連線時初始化的時間

JNDI : 使用JDNI方式取得DataSource

**dataSource的屬性**

**\*driver—設定驅動程式類別**

**\*url—設定資料庫連線位置**

**\*username—設定資料庫連線帳號**

**\*password—設定資料庫連線帳號**

**driver.useSSL—使用SSL**

**driver.useUnicode—使用Unicode**

**driver.encoding—設定編碼**

**\*POOLED模式時, dataSource的進階設定**

**poolMaximumActiveConnections—設定使用中的最大連線數**

**poolMaximumIdleConnections—設定空閒中的最大連線數**

**poolMaximumCheckoutTime—設定被強制返回條件(時間)**

**poolTimeToWait—設定重新連接條件(時間)**

**poolPingEnabled—啟用偵測查詢**

**poolPingQuery—設定偵測查詢, 用以測試連線是否正常, 可設定一般Query**

**poolPingConnectionsNotUsedFor—設定偵測查詢的使用頻率**

**\*JNDI模式時, dataSource的設定(兩屬性擇一設定即可)**

**\*data\_source—指定DataSource物件的JNDI名稱**

**initial\_context—同上, 但用InitialContext載入**

## databaseIdProvider

**\*註 : 管理不同的DBMS. 配合變數\_databaseId使用可處理方言問題**

**ex.**

**Mybatis-config.xml**

<databaseIdProvider type=*"DB\_VENDOR"*>

<property name=*"Oracle"* value=*"oracle"* />

<property name=*"SQl Server"* value=*"sqlserver"*/>

<property name=*"MySQL"* value=*"mysql"* />

<property name=*"DB2"* value=*"db2"* />

<property name=*"Adaptive Server Enterprise"* value=*"sybase"* />

</databaseIdProvider>

**MemberMapper.xml**

<select id=*"databaseIdExample"* resultType=*"Integer"*>

SELECT

<choose>

<when test=*"\_databaseId == oracle"*>LENGTH(NICKNAME)</when>

<when test=*"\_databaseId == sqlserver"*>LEN(NICKNAME)</when>

<otherwise>'ERROR'</otherwise>

</choose>

FROM MEMBER

</select>

## mappers

**一次設一個**

**classpath :** <mapper resource=*"MemberMapper.xml"* />

**檔案系統路徑 :** <mapper url=*"file:///D:/workspace/MybatisDemo/resource/MemberMapper.xml"* />

**類別套件(需配合註釋使用) :** <mapper class=*"model.mapper.MemberMapper"* />

**一次設多個(指定套件, 需配合註釋使用)**

<package name=*"model.mapper"*/>

# 映射檔(xxxMapper.xml) 標籤細節

**映射標籤<resultMap>**

**屬性**

id**—唯一識別名, 代表此映射**

type**—映射的JavaBean**

autoMapping**—使用自動映射, 預設 : unset**

extends**—指定本resultMap的父resultMap**

**子標籤**

<constructor>**—透過建構子, 將結果注入到JavaBean. 所有參數的順序、型態需完全對應**

**\*註 : 若使用**<constructor>**, JavaBean可無預設建構子, 但若用setter注入就一定要有預設建構子**

<idArg column=*"xxx"* javaType=*"xxx"*>**—鍵值結果參數, 標記出鍵值可提高效能**

<arg>**—普通結果參數**

<id>**—透過setter注入的鍵值結果, 標記出鍵值可提高效能**

property**—JavaBean性質名稱**

column**—資料庫的欄位名稱或別名, 即resultSet.getXxx("column")**

javaType**—Java資料型態, 通常可不設定(由MyBatis自動辨識), 但若映射到HashMap就必需指定**

jdbcType**—JDBC資料型態, 僅對null資料有作用, 所以一般只對接受null的欄位(性質)設定**

typeHandler**—指定型態處理器**

<result>**—透過setter注入的普通結果**

<association>**—設定一對一關聯的結果**

property**—JavaBean性質名稱**

column**—關聯Table鍵值欄位名稱, 可用**column=*"{prop1=col1,prop2=col2}"***設定複合鍵**

javaType**—JavaBean類別名稱**

typeHandler**—指定型態處理器**

select**—藉由另一個**<select>**取得關聯Bean**

fetchType**—lazy | eager**

resultMap**—參考(指向)其他已定義的**<resultMap id=*"xxx"*>

columnPrefix**—設定欄位名的前置詞, 用於標示來自不同Table, 但名稱相同的欄位**

**ex. 表格MEMBER和ORDER都有ID欄位, SELECT時將ORDER的ID取別名為O\_ID, 設定ORDER關聯時就可設定**<association columnPrefix=*"*O\_*"*>

notNullColumn**—指定某欄位不為null時才建立關聯Bean, 可設多欄位(以逗號隔開), 預設 : unset**

autoMapping**—使用自動映射**

foreignColumn**—指定外來鍵**

resultSet**—對應**<select resultSets=*"xxx, yyy"*>

<collection>**—設定一對多關聯的結果集**

**\*註 : 屬性大致上同**<association>**, 以下只列出部分**

javaType**—用於**<collection>**時, 通常設為ArrayList, 亦可不設(MyBatis可自動辨別)**

ofType**—JavaBean類別名稱**

<discriminator>**—映射鑑別器(結構似Java的switch case), 可依不同狀況使用不同的資料型態**

javaType**—受鑑識欄位的型態**

column**—需要鑑識的欄位**

<case>**—狀況, 當鑑識欄位的值為**value=*"xxx"***時, 回傳**resultType **/** resultMap**指定的資料型態**

value**—值**

resultType**—回傳型態**

resultMap**—回傳型態**

**DML標籤補充**

1. <selectKey>**—<insert>跟<update>的子標籤, 使用來取得自動產生的鍵值**

keyProperty**—自動產生的鍵值存回的性質**

resultType**—keyProperty的型態**

order**—設定執行順序為外標籤(<insert>或<update>)之前還是之後, BEFORE | AFTER**

statementType**—SQL語句的類型, STATEMENT | PREPARED | CALLABLE(呼叫Stored Procedure時使用). 預設 :**

**PREPARED**

1. <sql>**—定義可重複使用的SQL語句, ex.**

**\*註 : 某些環境下使用**<sql>**容易發生不明錯誤**

<sql id=*"sqlExample"*>

${tableName}.ACCOUNT, ${tableName}.PASSWORD, ${tableName}.NICKNAME

</sql>

<select id=*"useSqlExample"*>

SELECT

<include refid=*"sqlExample"*><property name=*"tableName"* value=*"T1"* /></include>

<include refid=*"sqlExample"*><property name=*"tableName"* value=*"T2"* /></include>

FROM TABLE1 T1

LEFT JOIN TABLE2 T2

ON T1.ID = T2.ID

</select>

1. <select resultType=*"Map"*>**—傳回Map的select**

兩種型式

* 1. 傳回一筆

MemberMapper.xml

<select id=*"selectByKeyReturnMap"* resultType=*"Map"*>

SELECT \* FROM MEMBER WHERE ID = #{id}

</select>

MemberMapper.java

Map<String, Object> selectByKeyReturnMap(@Param("id") Integer id);

MainApp.java

Map<String, Object> memberMap = *dao*.selectByKeyReturnMap(1);

**for** (String key : memberMap.keySet()) {

System.***out***.println(key + " : " + memberMap.get(key));

}

* 1. 傳回多筆

MemberMapper.xml

<select id=*"selectAllReturnMap"* resultType=*"Map"*>

SELECT \* FROM MEMBER

</select>

MemberMapper.java

方式1

@MapKey("ID")

Map<Integer, Member> selectAllReturnMap();

方式2

List<Map<String, Object>> selectAllReturnMap();

MainApp.java

Map<Integer, Member> resultMap = *dao*.selectAllReturnMap();

**for** (Integer id : resultMap.keySet()) {

System.***out***.println(resultMap.get(id));

}

**DML標籤的屬性**

**基本**

id**—唯一識別名, 對應映射器方法名稱**

parameterType**—參數型態**

resultType**—回傳型態**

resultMap**—回傳型態, 引用另一resultMap**

order**—只對<selectKey>有作用, <selectKey>在<insert>前或後(BEFORE或AFTER)**

**GeneratedKeys**

useGeneratedKeys**—只對<insert>、<update>有作用, 使用返回自動產生的鍵值**

keyProperty**—只對<insert>、<update>有作用, 指定自動產生的鍵值所存放性質**

keyColumn**—只對<insert>、<update>有作用, 指定自動產生的鍵值所屬欄位**

**系統**

flushCache**—對應方法被呼叫時, 清空LocalCache和LocalCache2. 預設 : false**

useCache**—使用LocalCache2緩存SQL. 預設 : <select>時為true**

timeout**—Driver等待資料庫的時間上限. 預設 : unset(依Driver設定)**

fetchSize**—Dirver向資料庫的批量資料筆數. 預設 : unset(依Driver設定)**

**JDBC**

statementType**—底層JDBC使用的Statement種類, STATEMENT，PREPARED 或 CALLABLE(呼叫Stored Procedure時使**

**用). 預設 : PREPARED**

resultSetType**—底層JDBC使用的ResultSet種類, FORWARD\_ONLY，SCROLL\_SENSITIVE 或SCROLL\_INSENSITIVE.**

**預設 : unset(依Driver設定)**

**其他**

databaseId**—DBMS廠商ID(配合databaseIdProvider使用)**

resultOrdered**—只對子查詢select有作用, 避免記憶體不足. 預設 : false**

resultSets**—只對多ResultSet有作用, 列出所有ResultSet, 並用逗號隔開**

**緩存標籤**

<cache>**—二級緩存, 預設狀況特性如下..**

1. **所有select語句被緩存**
2. **所有insert、update、delete語句刷新(flush)緩存**
3. **使用LRU回收機制(Least Recently Use)**
4. **The cache will not flush on any sort of time based schedule (i.e. no Flush Interval)**
5. **緩存1024個返回物件參考**
6. **緩存的返回物件為可讀可寫, 但並非共享的, 為呼叫者獨享**

eviction**—回收模式, LRU(預設) | FIFO | SOFT | WEAK**

flushInterval**—更新週期(毫秒)**

size**—儲存返回物件的數量上限, 預設 : 1024**

readOnly**—只能讀取, 設為false(預設)時較慢但較安全, 回傳物件不能修改內容, 設定為true時才可修改內容**

type**—指定自訂的緩存類別. 自訂的緩存類別需實作org.mybatis.cache.Cache介面**

**自訂JavaBean初始化物件-InitializingObject**

**註**

1. **版本 : 3.4.2後**
2. **實作介面 : org.apache.ibatis.builder.InitializingObject**
3. **行為 : 在注入所有性質後被呼叫**

<cache-ref namespace=*"xxx"*>**—指定參考其他xxxMapper.java所定義的cache**

# MyBatis API for Java

**連線**

**建立過程 : XML或@Annotation或Java code -> SqlSessionFactoryBuilder -> SqlSessionFactory -> SqlSession**

**\*註 : 使用MyBatis-Spring(或MyBatis-Guice)時, SqlSession則由Spring(或Guice)所建立**

**SqlSessionFactoryBuilder**

● build(InputStream inputStream)

●build(InputStream inputStream, String environment, Properties properties)

inputStream : 指定組態檔(mybatis-config.xml), 可透過Resources.*getResourceAsStream*("mybatis-config.xml")取得

environment : 指定環境, 對應<environment id=*"xxx"*>, 若無指定則使用<environments default=*"xxx"*>

properties : 指定.properties檔, 等同在mybatis-config.xml裡加<properties resource=*"xxx"* />

\*Proerties的優先權 : 參數 > <properties resource=*"xxx"* /> > mybatis-config.xml裡直接指定的值

●build(Configuration config)

**ex.**

DataSource dataSource = (DataSource) **new** InitialContext().lookup("java:comp/env/jdbc/ipo");

JdbcTransactionFactory transactionFactory = **new** JdbcTransactionFactory();

Environment environment = **new** Environment("development", transactionFactory, dataSource);

Configuration config = **new** Configuration(environment);

config.setLazyLoadingEnabled(**true**);

// config.setEnhancementEnabled(true);

TypeAliasRegistry typeAliasRegistry = config.getTypeAliasRegistry();

typeAliasRegistry.registerAlias(MemberBean.**class**);

typeAliasRegistry.registerAlias(ArticleBean.**class**);

typeAliasRegistry.registerAlias(Timestamp.**class**);

config.addMapper(MemberMapper.**class**);

config.addMapper(ArticleMapper.**class**);

SqlSessionFactory sessionFactory = **new** SqlSessionFactoryBuilder().build(config);

**SqlSessionFactory**

●openSession()

**特性**

1.不自動commit

2.Connection由當下環境取得

3.TransactionIsolationLevel為Driver或DataSource預設

4. PreparedStatement不重複使用, 也不批次執行更新(insert、delete、update)

● openSession(**boolean** autoCommit)

● openSession(Connection connection)

● openSession(TransactionIsolationLevel level)

● openSession(ExecutorType execType)

● openSession(ExecutorType execType, **boolean** autoCommit)

● openSession(ExecutorType execType, Connection connection)

● openSession(ExecutorType execType, TransactionIsolationLevel level)

autoCommit : 自動交易否

connection : 自訂的連線物件

level : 交易隔離等級, NONE | READ\_UNCOMMITTED | READ\_COMMITTED | REPEATABLE\_READ | SERIALIZABLE

execType : 執行器種類, SIMPLE(每次建一新的語句) | REUSE(重複使用語句) | BATCH(批量執行所有更新語句)

● getConfiguration() : 取得MyBatis組態設定, 可用於執行中自我檢查

**SqlSession**

**DML相關方法**

**註**

**ResultHandler : 結果處理器, 可自訂對結果的動作**

**new** ResultHandler<String>() {

@Override

**public** **void** handleResult(ResultContext<?

**extends** String> resultContext) {

}//ResultContext : 用來控制結果物件

}

**RowBounds : 指定要取回的範圍**

**new** RowBounds(offset, limit)

● insert(String statement)

● delete(String statement)

● update(String statement)

● selectOne(String statement)

● selectList(String statement)

● selectCursor(String statement)

● selectMap(String statement, String mapKey)

● select(String statement, ResultHandler handler)

● insert(String statement, Object parameter)

● delete(String statement, Object parameter)

● update(String statement, Object parameter)

● selectOne(String statement, Object parameter)

● selectList(String statement, Object parameter)

● selectMap(String statement, Object parameter, String mapKey)

● select(String statement, Object parameter, [ResultHandler](eclipse-javadoc:%E2%98%82=MybatisDemo/D:%5C/ipo%5C/workspace%5C/MybatisDemo%5C/WebContent%5C/WEB-INF%5C/lib%5C/mybatis-3.4.2.jar%3Corg.apache.ibatis.session(SqlSession.class%E2%98%83SqlSession~select~Ljava.lang.String;~Ljava.lang.Object;~Lorg.apache.ibatis.session.ResultHandler;%E2%98%82org.apache.ibatis.session.ResultHandler) handler)

● selectCursor(String statement, Object parameter)

● selectList(String statement, Object parameter, RowBounds rowBounds)

● selectMap(String statement, Object parameter, String mapKey, RowBounds rowBounds)

● select(String statement, Object parameter, RowBounds rowBounds, ResultHandler handler)

● selectCursor(String statement, Object parameter, RowBounds rowBounds)

**執行相關方法**

● flushStatements() : 當<setting name=*"defaultExecutorType"* value=*"BATCH"* />時, 用此方法執行儲存在JDBC類

別裡的批量更新SQL語句

**交易控制**

**註 : 當使用auto-commit或其他外部交易管理者時, 以下方法無效**

●commit()

●commit(**boolean** force)

●rollback()

●rollback(**boolean** force)

**連線相關**

● clearCache() : 清除本機cache. MyBatis會替每個SqlSessio附加本機cache, 理論上當呼叫update()、commit()、

rollback()、close()等方法時會被清除, 但要確切清除可使用clearCache()

● close()

**範例**

**MemberMapper.java**

**註 : 使用MyBatis API for Java注意事項**

1. **所有註冊至Configuration裡的Mapper不可有重複的方法名稱**

@Select("SELECT \* FROM MEMBER WHERE ID = #{id}")

Member selectByKey(@Param("id") Integer id);

**MainApp.java**

MysqlDataSource dataSource = **new** MysqlDataSource();

dataSource.setURL("jdbc:mysql://localhost:3306/ipo");

dataSource.setUser("root");

dataSource.setPassword("0000");

JdbcTransactionFactory transactionFactory = **new** JdbcTransactionFactory();

Environment environment = **new** Environment("development", transactionFactory, dataSource);

Configuration config = **new** Configuration(environment);

// TypeAliasRegistry typeAliasRegistry = config.getTypeAliasRegistry();

// typeAliasRegistry.registerAlias(Member.class);

config.addMapper(MemberMapper.**class**);

// config.addMappers("model.mapper");

SqlSessionFactory sessionFactory = **new** SqlSessionFactoryBuilder().build(config);

**try**(SqlSession session = sessionFactory.openSession()){

Member member = session.selectOne("selectByKey", 1);

System.***out***.println(member.getNickname());

} **catch** (Exception e) {

e.printStackTrace();

}

# MyBatis-@Annotation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DML相關 | | | | |
| @Annotation | Target | XML對應 | Desc | Example |
| @Insert | ***METHOD*** | <insert> | 設定INSERT SQL語句 | @Insert("INSERT INTO MEMBER (ACCOUNT, …)") |
| @Delete | ***METHOD*** | <delete> | 設定DELETE SQL語句 | @Delete("DELETE FROM MEMBER WHERE ID = #{id}") |
| @Update | ***METHOD*** | <update> | 設定UPDATE SQL語句 | @Update("UPDATE MEMBER SET PASS = #{pass}, …") |
| @Select | ***METHOD*** | <select> | 設定SELECT SQL語句 | @Select("SELECT \* FROM MEMBER") |
| @SelectKey | ***METHOD*** | <selectKey> | 設定返回自動產生的鍵值之SQL語句 | @SelectKey(before = **false**, keyProperty = "id", resultType = Integer.**class**, statement = "SELECT LAST\_INSERT\_ID()") |
| @InsertProvider | ***METHOD*** | <insert> | 依指定的類別和方法回傳動態的SQL語句. **需另外定義Provider類別** | @SelectProvider(type = MyProvider.**class**, method = "getSelectByKeyStatement") |
| @DeleteProvider | ***METHOD*** | <delete> |
| @UpdateProvider | ***METHOD*** | <update> |
| @SelectProvider | ***METHOD*** | <select> |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 映射相關 | | | | |
| @Annotation | Target | XML對應 | Desc | Example |
| @Results | ***METHOD*** | <resultMap> | 設定性質與欄位映射集 |  |
| @ConstructorArgs |  | <constructor> | 透過建構子, 將結果注入到JavaBean |  |
| @Arg |  | <arg> <idArg> | 指定透過建構子注入時的參數 |  |
| @Result |  | <result> <id> | 設定性質與欄位映射關係 |  |
| @One |  | <association> | 設定一對一關聯的結果 |  |
| @Many |  | <collection> | 設定一對多關聯的結果 |  |
| @TypeDiscriminator | ***METHOD*** | <discriminator> | 使用映射鑑別器 |  |
| @Case |  | <case> | 設定映射鑑別器狀況 |  |
| @MapKey | ***METHOD*** |  | 用在傳回Map<K, V>的方法, 設定JavaBean的哪一個性質為鍵值 | @MapKey("ID") |
| @ResultMap | ***METHOD*** |  | 指定參考定義好的@Results(映射集) | @ResultMap("resultsId") |
| @ResultType | ***METHOD*** |  | 當使用結果處理器時, 方法的回傳型態設為void, 必須用此註釋設定型態 |  |
| @Param | ***PARAMETER*** |  | 設定SQL語句裡對應的變數 | @Param("id") |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系統與進階 | | | | |
| @Annotation | Target | XML對應 | Desc | Example |
| @CacheNamespace | ***TYPE*** | <cache> | 設定緩存 |  |
| @CacheNamespaceRef | ***TYPE*** | <cache-ref> | 指定參考其他xxxMapper.java所定義的cache |  |
| @Options | ***METHOD*** | DML標籤的屬性 | 設定可選擇的進階設定, 一定使用此註釋, 未被設定的屬性會被設為預設值 | @Options(fetchSize = 1) |
| @Flush | ***METHOD*** |  | 用此方法執行儲存在JDBC類別裡的批量更新SQL語句, 等同呼叫SqlSession.flushStatements() | @Flush |
| @Property |  | <property> | 引用外部設定, 會被mybatis.xml裡的<properties>設定覆蓋 |  |

**範例**

**\*註 : 註釋寫法可完全取代映射檔** **(xxxMapper.xml)寫法, 即不用寫映射檔, 但太複雜的SQL較不適用**

**基本**

@Results(id = "memberMap", value = {

@Result(id = **true**, property = "id", column="ID"),

@Result(property = "account", column = "ACCOUNT"),

@Result(property = "password", column = "PASSWORD"),

@Result(property = "nickname", column = "NICKNAME"),

@Result(property = "createTime", column = "CREATE\_TIME"),

@Result(property = "pass", column = "PASS"),

@Result(property = "sessionId", column = "SESSION\_ID")

})

@Select("SELECT \* FROM MEMBER WHERE ID = #{id}")

MemberBean selectByKey(Integer id);

@ResultMap("memberMap")

@Select("SELECT \* FROM MEMBER")

List<MemberBean> selectAll();

@Insert("INSERT INTO MEMBER(ACCOUNT, PASSWORD, NICKNAME, CREATE\_TIME, PASS, SESSION\_ID)"

+ "VALUES(#{account}, #{password}, #{nickname}, #{createTime}, #{pass}, #{sessionId})")

@SelectKey(before = **false**, keyProperty = "m\_id", resultType = Integer.**class**, statement = "SELECT LAST\_INSERT\_ID()")

**int** insertWithSelectKey(MemberBean member);

**進階**

1. **@ConstructorArgs**

@Results(id = "MemberMap")

@ConstructorArgs({

@Arg(column = "ID", javaType = Integer.**class**, id = **true**),

@Arg(column = "ACCOUNT", javaType = String.**class**),

@Arg(column = "PASSWORD", javaType = String.**class**),

@Arg(column = "NICKNAME", javaType = String.**class**),

@Arg(column = "CREATE\_TIME", javaType = Timestamp.**class**),

@Arg(column = "PASS", javaType = Boolean.**class**),

@Arg(column = "SESSION\_ID", javaType = String.**class**)

}

)

@Select("SELECT \* FROM MEMBER WHERE ID = #{id}")

Member selectWithConstructorArgs(Integer id);

1. **@SelectProvider**

@ResultMap("memberMap")

@SelectProvider(type = SQLStatementProvider.**class**, method = "getSelectByKeyStatement")

Member selectWithProvider(Integer id);

SQLStatementProvider.java

**public** **class** SQLStatementProvider {

**public** String getSelectByKeyStatement(Integer id) {

**return** **new** SQL(){{

SELECT("\*");

FROM("MEMBER");

WHERE("ID = #{id}");

}}.toString();

}

}

# SQL Builder – SQL類別

**Insert Join**

INSERT\_INTO(String tableName) JOIN(String... joins)

INTO\_COLUMNS(String... columns) INNER\_JOIN(String... joins)

INTO\_VALUES(String... values) LEFT\_OUTER\_JOIN(String... joins)

或 RIGHT\_OUTER\_JOIN(String... joins)

VALUES(String columns, String values)

**Delete Condition Concatenate**

DELETE\_FROM(String table) OR()

WHERE(String... conditions) AND()

**Update 註**

UPDATE(String table) 1. 版本3.4.2+ 支援

SET(String... sets) 2. WHERE()、HAVING()預設用**AND**為分隔符號

WHERE(String... conditions) 3. 沒有"ON "敘述的方法

**Select**

SELECT(String... columns)

SELECT\_DISTINCT(String... columns)

FROM(String... tables)

WHERE(String... conditions)

GROUP\_BY(String... columns)

HAVING(String... conditions)

ORDER\_BY(String... columns)

**範例**

MemberSQLProvider.java

**public** **class** MemberSQLProvider {

**public** **static** String getInsertSQL1() {

**return** **new** SQL()

.INSERT\_INTO("MEMBER")

.INTO\_COLUMNS("ACCOUNT", "PASSWORD", "NICKNAME", "CREATE\_TIME", "PASS", "SESSION\_ID")

.INTO\_VALUES("#{account}", "#{password}", "#{nickname}", "#{createTime}", "#{pass}",

"#{sessionId}")

.toString();

}

**public** **static** String getInsertSQL2() {

**return** **new** SQL()

.INSERT\_INTO("MEMBER")

.VALUES("ACCOUNT", "#{account}")

.VALUES("PASSWORD", "#{password}")

.VALUES("NICKNAME", "#{nickname}")

.VALUES("CREATE\_TIME", "#{createTime}")

.VALUES("PASS", "#{pass}")

.VALUES("SESSION\_ID", "#{sessionId}")

.toString();

}

**public** **static** String getDeleteSQL() {

**return** **new** SQL()

.DELETE\_FROM("MEMBER")

.WHERE("ID = #{id}")

.toString();

}

**public** **static** String getUpdateSQL() {

**return** **new** SQL()

.UPDATE("MEMBER")

.SET("ACCOUNT = #{account}", "PASSWORD = #{password}", "NICKNAME = #{nickname}",

"CREATE\_TIME = #{createTime}", "PASS = #{pass}", "SESSION\_ID = #{sessionId}")

.WHERE("ID = #{id}")

.toString();

}

**public** **static** String getSelectByKeySQL() {

**return** **new** SQL()

.SELECT("\*")

.FROM("MEMBER")

.WHERE("ID = #{id}")

.toString();

}

**public** **static** String getSelectAllSQL() {

**return** **new** SQL()

.SELECT("\*")

.FROM("MEMBER")

.toString();

}

}

MemberMapper.java

**public** **interface** MemberMapper {

@InsertProvider(method = "getInsertSQL1", type = MemberSQLProvider.**class**)

**int** insert1(Member bean);

@InsertProvider(method = "getInsertSQL2", type = MemberSQLProvider.**class**)

**int** insert2(Member bean);

@DeleteProvider(method = "getDeleteSQL", type = MemberSQLProvider.**class**)

**int** delete(Integer id);

@UpdateProvider(method = "getUpdateSQL", type = MemberSQLProvider.**class**)

**int** update(Member bean);

@Results(id = "MemberMap")

@ConstructorArgs({

@Arg(column = "ID", javaType = Integer.**class**, id = **true**),

@Arg(column = "ACCOUNT", javaType = String.**class**),

@Arg(column = "PASSWORD", javaType = String.**class**),

@Arg(column = "NICKNAME", javaType = String.**class**),

@Arg(column = "CREATE\_TIME", javaType = Timestamp.**class**),

@Arg(column = "PASS", javaType = Boolean.**class**),

@Arg(column = "SESSION\_ID", javaType = String.**class**)

})

@SelectProvider(method = "getSelectByKeySQL", type = MemberSQLProvider.**class**)

Member selectByKey(Integer id);

@ResultMap("MemberMap")

@SelectProvider(method = "getSelectAllSQL", type = MemberSQLProvider.**class**)

List<Member> selectAll();

}

# MyBatis-Spring

## 0. 完成Spring及MyBatis各自基本設定

**\*註 : 使用Mybatis-Spring, 可將幾乎全部MyBatis組態檔內容移植到Spring組態檔裡**

## 1. 加入jar檔 : mybatis-spring-x.y.z.jar

## 2. 修改mybatis-config.xml

**將**<environments>**及**<mappers>**註解**

## 3. 修改applicationContext.xml

### 3.1 宣告DataSource

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"username"* value=*"root"* />

**組態檔**

**Spring : WEB-INF/applicationContext.xml**

**MyBatis : resource/mybatis-config.xml**

<property name=*"password"* value=*"password"* />

<property name=*"url"* value=*"*jdbc:mysql://localhost:3306/ipo*"* />

</bean>

**或**

<bean id=*"dataSource"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"java:comp/env/jdbc/ipo"* />

</bean>

### 3.2 宣告SqlSessionFactory

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* /> <--指定DataSource

<property name=*"configLocation"* value=*"classpath:mybatis-config.xml"* /> <--指定MyBatis組態檔位置

<property name=*"mapperLocations"* value=*"classpath:mapper/\*Mapper.xml"* /><--指定映射檔位置(可設多個)

</bean>

### 3.3 宣告映射器(Mapper)

<bean id=*"MemberMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>

<property name=*"mapperInterface"* value=*"model.mapper.MemberMapper"* /> <--指定xxxMapper

<property name=*"sqlSessionFactory"* ref=*"sqlSessionFactory"* /> <--指定SqlSessionFactory

</bean>

## 4. Controller裡透過@Autowired使用映射器(Mapper)

@Controller

@RequestMapping("member")

**public** **class** MemberController {

@Autowired

**private** MemberMapper memberMapper;

@RequestMapping("findAllMembers")

**public** String findAllMembers(Model model) {

List<Member> members = memberMapper.selectAll();

model.addAttribute("members", members);

**return** "result";

}

}

## 補充

### 1. 自動掃描映射器介面(Mapper.java)

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 可設多個路徑(用分號隔開) -->

<property name=*"basePackage"* value=*"sys1.model.mapper;sys2.model.\*.mapper; "* />

<property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"* />

</bean>

### 2. 類別別名 : 在SqlSessionFactory裡設定

**\*註 : 不可用米字號("\*")**

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 可設多個套件(用逗號隔開), 不可用萬用字元(\*) -->

<property name=*"typeAliasesPackage"* value=*"sys1.model.pojo,sys2.model.pojo"* />

</bean>

### 3. MyBatis-SqlSessionTemplate

<bean id=*"mybatisSessionTemplate"* class=*"org.mybatis.spring.SqlSessionTemplate"*>

<constructor-arg index=*"0"* ref=*"sqlSessionFactory"*/>

<constructor-arg index=*"1"* value=*"BATCH"*/> <--支援批量執行

</bean>

### 4. 多個映射檔位置 : 在SqlSessionFactory裡設定

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"mapperLocations"*>

<array>

<value>classpath\*:sys1/mapepr/\*Mapper.xml</value>

<value>classpath\*:sys2/mapper/\*Mapper.xml</value>

</array>

</property>

</bean>

# PageHelper

## 前置作業

### 加入jar檔

**手動**

[pagehelper-x.y.z.jar](http://repo1.maven.org/maven2/com/github/pagehelper/pagehelper/) : <http://repo1.maven.org/maven2/com/github/pagehelper/pagehelper/>

[jsqlparser-x.y.z.jar](http://repo1.maven.org/maven2/com/github/jsqlparser/jsqlparser/0.9.5/) : <http://repo1.maven.org/maven2/com/github/jsqlparser/jsqlparser/0.9.5/>

**Maven**

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

<version>x.y.z</version>

</dependency>

### 指定攔截器(引入外掛)

**MyBatis :** 在mybatis-config.xml裡, <configuration>中加入

<plugins>

<plugin interceptor=*"com.github.pagehelper.PageInterceptor"*>

<property name=*"細節設定1"* value=*"值1"* />

<property name=*"細節設定2"* value=*"值2"* />

...

</plugin>

</plugins>

**MyBatis-Spring :** 在applicationContext.xml裡,

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>中加入

<property name=*"plugins"*>

<array>

<bean class=*"com.github.pagehelper.PageInterceptor"*>

<property name=*"properties"*>

<value>

細節設定1=值1

細節設定2=值2

...

</value>

</property>

</bean>

</array>

</property>

**注意事項**

1. 指定攔截器(引入外掛), 應在mybatis-config.xml或applicationContext.xml擇一設定, 不可重複設定

2. PageHelper不支援含有FOR UPDATE的sql script

3. 含有子查詢的sql script可能會導致錯誤

4. PageHelper.startPage()只影響此方法後的**第一個**MyBatis SELECT方法, 所以若有判斷敘述時, 應寫在同區塊, ex..

List<Member> memberList;

**if** (status == 1) {

PageHelper.*startPage*(1, 3);//此行與下行必需放在同區塊

memberList = mapper.selectAll();

}

5. 手動清除分頁參數 : PageHelper.*clearPage*();

## 細節設定

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名稱** | **說明** | **值** | **備註** | **範例** |
| dialect | 指定分頁方式 | com.github.pagehelper.PageHelper(預設)  ,自訂的**分頁類別**全名 | 自訂需實作com.github.pagehelper.Dialect | dialect=x.y.z.MyDialect |
| helperDialect | 指定分頁方式的方言 | oracle,mysql,mariadb,sqlite,hsqldb,postgresql,  db2,**sqlserver**,informix,h2,**sqlserver2012**,derby,**自訂的分頁方言類別全名** | 1.PageHelper會依當前DBMS,選擇適當的分頁方式; 亦可直接指定  2.sqlserver表示SqlServer2005  3.自訂需繼承AbstractHelperDialect | helperDialect=mysql |
| offsetAsPageNum | true時,將RowBounds中的offset當成pageNum使用,進行分頁 | false(預設),true | 1.只對RowBounds作為分頁參數時才有效  2.false時,reasonable強制為false | offsetAsPagenum=true |
| rowBoundsWithCount | true時,且使用RowBounds分頁會進行count查詢 | false(預設),true | 只對RowBounds作為分頁參數時才有效 | rowBoundsWithCount=true |
| pageSizeZero | true時,且pageSize=0或RowBounds.limit=0,表示查全部 | false(預設),true | 相當於沒有執行分頁查詢 | pageSizeZero=true |
| reasonable | 分頁合理化;true時,且pageNum<=0表示查第一頁,pageNum>總頁數表示查最後一頁 | false(預設),true | 1.false時,則直接依據參數查詢  2. offsetAsPageNum=true時, 強制reasonable=false | reasonable=true |
| params | PageHelper.startPage(Object params)或Mapper.select(int pageNum, int PageSize)裡的參數映射 | pageNum=自訂名稱, pageSize=自訂名稱, count=自訂名稱, pageSizeZero=自訂名稱, reasonable=自訂名稱 | 1.各參數映射以分號隔開  2.預設為pageNum=pageNum;pageSize=pageSize;count=countSql;reasonable=reasonable;pageSizeZero=pageSizeZero | params=pageNum=num; pageSize= size |
| supportMethodsArguments | 支援通過Mapper參數指定分頁參數;找到適當的值就會自動分頁 | false(預設),true | 1. 映射規則依上params設定  2. 配合@Param("pageNum") **int** pageNum,  @Param("pageSize") **int** pageSize使用 | supportMethodsArguments=true |
| autoRuntimeDialect | 允許在執行中自動切換分頁方言 | false(預設),true | sqlserver2012不支援 | autoRuntimeDialect=true |
| closeConn | 是否關閉自動取得的連線 | false,true(預設) | 當動態資料源且無指定方言(helperDialect)時,自動取得一個連線 | closeConn=false |

## 使用方式

### RowBounds類別

#### 透過session.selectOne() / session.selectList()

Member member = session.selectOne("selectByKey", 1);

List<Member> memberList = session.selectList("selectAll", **null**, **new** RowBounds(1, 3));

#### 透過mapper.select()

步驟

1. 修改Mapper.java : 在欲分頁的select方法中加入參數..

List<Member> selectAll(RowBounds rowBounds);//MemberMapper.java

2. 使用方法

List<Member> memberList = mapper.selectAll(**new** RowBounds(1, 3));//call

### PageHepler類別的靜態方法

#### PageHelper.*startPage*(int pageNum, int pageSize)

PageHelper.*startPage*(1, 3);//第1頁, 每頁3筆

List<Member> memberList = memberMapper.selectAll();

#### PageHelper.*offsetPage*(int start, int length)

PageHelper.*offsetPage*(0, 5);//第0筆開始, 取5筆 (start從0開始)

List<Member> memberList = memberMapper.selectAll();

### 透過Mapper.java的參數

#### **\*註 : 需先開啟supportMethodsArguments功能**

MyBatis :在mybatis-config.xml裡, <plugin interceptor=*"..PageInterceptor"*>中加入

<property name=*"supportMethodsArguments"* value=*"true"* />

MyBatis-Spring : 在applicationContext.xml裡

<bean id=*"sqlSessionFactory"* class=*"..SqlSessionFactoryBean"* >

<property name=*"plugins"*>

<array>

<bean class=*"..PageInterceptor"*>

<property name=*"properties"*>

<value>

psupportMethodsArguments=true

#### select(@Param("pageNum") **int** pageNum, @Param("pageSize") **int** pageSize)

**\*註 : 可在組態檔利用params設定, 自訂參數名稱**

步驟

1. 修改Mapper.java : 在欲分頁的select方法中加入參數..

List<Member> selectAll(@Param("pageNum") **int** pageNum, @Param("pageSize") **int** pageSize);//預設參數名稱

2. 使用方法

List<Member> memberList = memberMapper.selectAll(1, 3);

#### Pojo.java中加入pageNum、pageSize性質

步驟

1. 修改Pojo.java : 加入pageNum、pageSize及setter()、getter()

**public** **class** Member {

...//other properties here

**private** Integer pageNum;

**private** Integer pageSize;

//pageNum and pageSize setter()、getter() here

}

2. 使用方法

Member member = **new** Member();

member.setPageNum(1);

member.setPageSize(3);

List<Member> memberList = memberMapper.selectAll(member);

**\*註 : 當pageNum != null && pageSize != null時, 才會自動分頁**

### ISelect介面

#### Page<T>類別

Page<Member> page = PageHelper.*startPage*(1, 3).doSelectPage(() -> {

mapper.selectAll();

});

#### PageInfo<T>類別

PageInfo<Member> pageInfo = PageHelper.*startPage*(1, 3).doSelectPageInfo(() -> {

mapper.selectAll();

});

#### 查數量

**long** count = PageHelper.*count*(() -> {

mapper.selectAll();

});

#### 方法集

|  |  |  |  |
| --- | --- | --- | --- |
| Page<T> | | PageInfo<T> | |
| **boolean** isCount() | 包含count查詢否 | **boolean** isFirstPage() | 為第一頁否 |
| Boolean getReasonable() | 使用分頁合理化否 | **boolean** isLastPage() | 為第最後頁否 |
| Boolean getPageSizeZero() | 使用pageSizeZero否 | **boolean** hasPreviousPage() | 有前頁否 |
| String getCountColumn() | 取得count()查詢的欄位名 | **boolean** hasNextPage() | 有次頁否 |
| String getOrderBy() | 取得排序 | **boolean** isFirstPage() | 為第一頁否 |
| **int** getPageNum() | 取得當前頁數 | **int** getPageNum() | 取得當前頁數 |
| **boolean** isOrderByOnly() | 只允許遞增排序否 | **int** getPageSize() | 取得頁面大小 |
| **int** getSize() | 取得當前頁大小 | **int** getSize() | 取得當前頁大小 |
| **int** getStartRow() | 取得當前第一筆資料在資料庫的ROWNUM | **int** getStartRow() | 取得當前第一筆資料在資料庫的ROWNUM |
| **int** getEndRow() | 取得當前最後一筆資料在資料庫的ROWNUM | **int** getEndRow() | 取得當前最後一筆資料在資料庫的ROWNUM |
| **long** getTotal() | 取得資料總數量 | **long** getTotal() | 取得資料總數量 |
| **int** getPages() | 取得頁總數量 | **int** getPages() | 取得頁總數量 |
|  |  | **int** getPrePage() | 取得前頁頁數 |
|  |  | **int** getNextPage() | 取得次頁頁數 |
|  |  | **int** getPrePage() | 取得前頁頁數 |
|  |  | **int** getNavigatePages() | 取得導航頁數(??) |
|  |  | **int** getNavigateFirstPage() | 取得導航第一頁頁數 |
|  |  | **int** getNavigateLastPage() | 取得導航最後一頁頁數 |
|  |  | **int**[] getNavigatepageNums() | 取得導航頁數集 |
|  |  | List<T>getList() | 取得當前頁結果集 |

# MyBatipse – MyBatis Tool for Eclipse

**註**

**1. 此工具主要是協助XML文件的編寫**

**2. 只支援Eclipse Luna(4.4+)以上**

**安裝 :** Eclipse -> Help -> Eclipse Marketplace -> Find文字框輸入“mybatipse” -> Go -> 找到MyBatipse x.y.z -> Install

**設定**

1. **即時同步檢查 :** 專案 -> 右鍵 -> Configure -> Add Mybatis Nature
2. **類別別名無法對應修正 :** 專案 -> 右鍵 -> Properties -> MyBatipse -> 選Use project settings -> New -> 輸入方式三種

**a. 類別全名 :** 表示此類別有設定別名, ex.. model.pojo.MemberBean

**b. 套件 :** 表示此套件下所有類別都有設定別名, ex.. model.pojo

**c. 不規則別名 :** 類別別名並非使用MyBatis預設命名方式時使用, ex.. model.pojo.MemberBean**:Member**

**註 : 1. MyBatis類別別名預設命名方式為 model.pojo.MemberBean -> MemberBean**

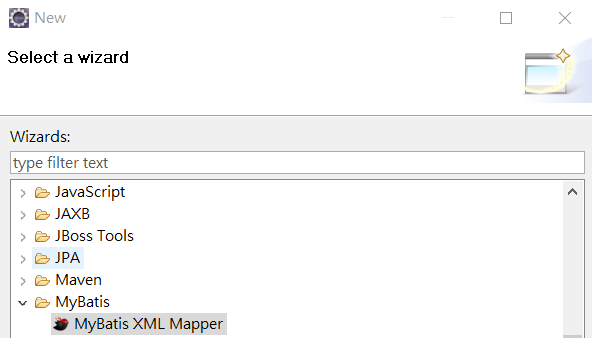
**2. 修改完要按 Project ->Clean, 才會修正**

1. **開啟MyBatis Declaration視景 : Windows -> Show View -> Other -> MyBatis Declaration -> OK**

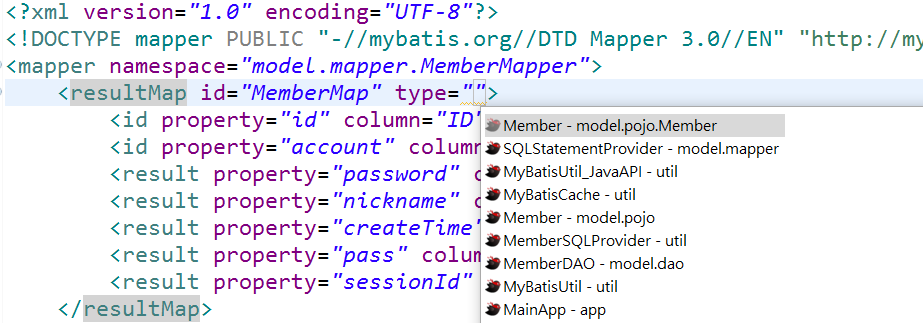
**註 : 編寫SQL語句時, 可用此視景顯示完整SQL語句**

**使用**

建立映射檔(xxxMapper.xml) : (在要建立xxxMapper.xml的資料夾) 右鍵 -> New -> Other -> **MyBatis XML Mapper**



編寫XML文件時, 按**Alt + /**, 即會出現自動完成框

****

# 參考資料

MyBatis(英文) : http://www.mybatis.org/mybatis-3/

MyBatis (簡中) : <http://www.mybatis.org/mybatis-3/zh/>

MyBatis-Spring(英文) : <http://www.mybatis.org/spring/>

MyBatis-Spring(簡中) : <http://www.mybatis.org/spring/zh/>

MyBatis API Documents : <http://www.mybatis.org/mybatis-3/zh/apidocs/index.html>

MySQL建議的資料型態 : <https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-type-conversions.html>

MyBatipse分享 : <http://www.jianshu.com/p/4a309d9882ca>

MyBatis問題討論區 : <https://github.com/mybatis/mybatis-3/issues>

MyBatis-PageHelper : <https://github.com/pagehelper/Mybatis-PageHelper/blob/master/README_zh.md>