**一、MyBatis简介与配置MyBatis+Spring+MySql**

**1.1MyBatis简介**

MyBatis 是一个可以自定义SQL、存储过程和高级映射的持久层框架。MyBatis 摒除了大部分的JDBC代码、手工设置参数和结果集重获。MyBatis 只使用简单的XML 和注解来配置和映射基本数据类型、Map 接口和POJO 到数据库记录。相对Hibernate和Apache OJB等“一站式”ORM解决方案而言，Mybatis 是一种“半自动化”的ORM实现。  
需要使用的Jar包：mybatis-3.0.2.jar（mybatis核心包）。mybatis-spring-1.0.0.jar(与Spring结合包)。

下载地址：  
<http://ibatis.apache.org/tools/ibator>  
<http://code.google.com/p/mybatis/>

**1.2MyBatis+Spring+MySql简单配置**

**1.2.1搭建Spring环境**

1,建立maven的web项目；  
2,加入Spring框架、配置文件；  
3,在pom.xml中加入所需要的jar包（spring框架的、mybatis、mybatis-spring、junit等）；  
4,更改web.xml和spring的配置文件；  
5,添加一个jsp页面和对应的Controller；  
6,测试。

可参照：<http://limingnihao.iteye.com/blog/830409>。[**使用Eclipse的Maven构建SpringMVC项目**](http://limingnihao.iteye.com/blog/830409)

**1.2.2建立MySql数据库**

建立一个学生选课管理数据库。  
表：学生表、班级表、教师表、课程表、学生选课表。  
逻辑关系：每个学生有一个班级；每个班级对应一个班主任教师；每个教师只能当一个班的班主任；

使用下面的sql进行建数据库，先建立学生表，插入数据（2条以上）。

更多sql请下载项目源文件，在resource/sql中。

Sql代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. /\* 建立数据库 \*/
2. CREATE DATABASE STUDENT\_MANAGER;
3. USE STUDENT\_MANAGER;
4. /\*\*\*\*\* 建立student表 \*\*\*\*\*/
5. CREATE TABLE STUDENT\_TBL
6. (
7. STUDENT\_ID VARCHAR(255) PRIMARY KEY,
8. STUDENT\_NAME VARCHAR(10) NOT NULL,
9. STUDENT\_SEX VARCHAR(10),
10. STUDENT\_BIRTHDAY DATE,
11. CLASS\_ID VARCHAR(255)
12. );
13. /\*插入学生数据\*/
14. INSERT INTO STUDENT\_TBL (STUDENT\_ID,
15. STUDENT\_NAME,
16. STUDENT\_SEX,
17. STUDENT\_BIRTHDAY,
18. CLASS\_ID)
19. VALUES (123456,
20. '某某某',
21. '女',
22. '1980-08-01',
23. 121546
24. )

/\* 建立数据库 \*/

CREATE DATABASE STUDENT\_MANAGER;

USE STUDENT\_MANAGER;

/\*\*\*\*\* 建立student表 \*\*\*\*\*/

CREATE TABLE STUDENT\_TBL

(

STUDENT\_ID VARCHAR(255) PRIMARY KEY,

STUDENT\_NAME VARCHAR(10) NOT NULL,

STUDENT\_SEX VARCHAR(10),

STUDENT\_BIRTHDAY DATE,

CLASS\_ID VARCHAR(255)

);

/\*插入学生数据\*/

INSERT INTO STUDENT\_TBL (STUDENT\_ID,

STUDENT\_NAME,

STUDENT\_SEX,

STUDENT\_BIRTHDAY,

CLASS\_ID)

VALUES (123456,

'某某某',

'女',

'1980-08-01',

121546

)

创建连接MySql使用的配置文件mysql.properties。

Mysql.properties代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. jdbc.driverClassName=com.mysql.jdbc.Driver
2. jdbc.url=jdbc:mysql://localhost:3306/student\_manager?user=root&password=limingnihao&useUnicode=true&characterEncoding=UTF-8

jdbc.driverClassName=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/student\_manager?user=root&password=limingnihao&useUnicode=true&characterEncoding=UTF-8

**1.2.3搭建MyBatis环境**

顺序随便，现在的顺序是因为可以尽量的少的修改写好的文件。

**1.2.3.1创建实体类: StudentEntity**

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public class StudentEntity implements Serializable {
2. private static final long serialVersionUID = 3096154202413606831L;
3. private ClassEntity classEntity;
4. private Date studentBirthday;
5. private String studentID;
6. private String studentName;
7. private String studentSex;
8. public ClassEntity getClassEntity() {
9. return classEntity;
10. }
11. public Date getStudentBirthday() {
12. return studentBirthday;
13. }
14. public String getStudentID() {
15. return studentID;
16. }
17. public String getStudentName() {
18. return studentName;
19. }
20. public String getStudentSex() {
21. return studentSex;
22. }
23. public void setClassEntity(ClassEntity classEntity) {
24. this.classEntity = classEntity;
25. }
26. public void setStudentBirthday(Date studentBirthday) {
27. this.studentBirthday = studentBirthday;
28. }
29. public void setStudentID(String studentID) {
30. this.studentID = studentID;
31. }
32. public void setStudentName(String studentName) {
33. this.studentName = studentName;
34. }
35. public void setStudentSex(String studentSex) {
36. this.studentSex = studentSex;
37. }
38. }

public class StudentEntity implements Serializable {

private static final long serialVersionUID = 3096154202413606831L;

private ClassEntity classEntity;

private Date studentBirthday;

private String studentID;

private String studentName;

private String studentSex;

public ClassEntity getClassEntity() {

return classEntity;

}

public Date getStudentBirthday() {

return studentBirthday;

}

public String getStudentID() {

return studentID;

}

public String getStudentName() {

return studentName;

}

public String getStudentSex() {

return studentSex;

}

public void setClassEntity(ClassEntity classEntity) {

this.classEntity = classEntity;

}

public void setStudentBirthday(Date studentBirthday) {

this.studentBirthday = studentBirthday;

}

public void setStudentID(String studentID) {

this.studentID = studentID;

}

public void setStudentName(String studentName) {

this.studentName = studentName;

}

public void setStudentSex(String studentSex) {

this.studentSex = studentSex;

}

}

**1.2.3.2创建数据访问接口**

Student类对应的dao接口：StudentMapper。

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public interface StudentMapper {
2. public StudentEntity getStudent(String studentID);
3. public StudentEntity getStudentAndClass(String studentID);
4. public List<StudentEntity> getStudentAll();
5. public void insertStudent(StudentEntity entity);
6. public void deleteStudent(StudentEntity entity);
7. public void updateStudent(StudentEntity entity);
8. }

public interface StudentMapper {

public StudentEntity getStudent(String studentID);

public StudentEntity getStudentAndClass(String studentID);

public List<StudentEntity> getStudentAll();

public void insertStudent(StudentEntity entity);

public void deleteStudent(StudentEntity entity);

public void updateStudent(StudentEntity entity);

}

**1.2.3.3创建SQL映射语句文件**

Student类的sql语句文件StudentMapper.xml  
resultMap标签：表字段与属性的映射。  
Select标签：查询sql。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3. <mapper namespace="com.manager.data.StudentMapper">
4. <resultMap type="StudentEntity" id="studentResultMap">
5. <id property="studentID" column="STUDENT\_ID"/>
6. <result property="studentName" column="STUDENT\_NAME"/>
7. <result property="studentSex" column="STUDENT\_SEX"/>
8. <result property="studentBirthday" column="STUDENT\_BIRTHDAY"/>
9. </resultMap>
10. <!-- 查询学生，根据id -->
11. <select id="getStudent" parameterType="String" resultType="StudentEntity" resultMap="studentResultMap">
12. <![CDATA[
13. SELECT \* from STUDENT\_TBL ST
14. WHERE ST.STUDENT\_ID = #{studentID}
15. ]]>
16. </select>
17. <!-- 查询学生列表 -->
18. <select id="getStudentAll" resultType="com.manager.data.model.StudentEntity" resultMap="studentResultMap">
19. <![CDATA[
20. SELECT \* from STUDENT\_TBL
21. ]]>
22. </select>
23. </mapper>

<?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="com.manager.data.StudentMapper">

<resultMap type="StudentEntity" id="studentResultMap">

<id property="studentID" column="STUDENT\_ID"/>

<result property="studentName" column="STUDENT\_NAME"/>

<result property="studentSex" column="STUDENT\_SEX"/>

<result property="studentBirthday" column="STUDENT\_BIRTHDAY"/>

</resultMap>

<!-- 查询学生，根据id -->

<select id="getStudent" parameterType="String" resultType="StudentEntity" resultMap="studentResultMap">

<![CDATA[

SELECT \* from STUDENT\_TBL ST

WHERE ST.STUDENT\_ID = #{studentID}

]]>

</select>

<!-- 查询学生列表 -->

<select id="getStudentAll" resultType="com.manager.data.model.StudentEntity" resultMap="studentResultMap">

<![CDATA[

SELECT \* from STUDENT\_TBL

]]>

</select>

</mapper>

**1.2.3.4创建MyBatis的mapper配置文件**

在src/main/resource中创建MyBatis配置文件：mybatis-config.xml。  
typeAliases标签：给类起一个别名。com.manager.data.model.StudentEntity类，可以使用StudentEntity代替。  
Mappers标签：加载MyBatis中实体类的SQL映射语句文件。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
3. <configuration>
4. <typeAliases>
5. <typeAlias alias="StudentEntity" type="com.manager.data.model.StudentEntity"/>
6. </typeAliases>
7. <mappers>
8. <mapper resource="com/manager/data/maps/StudentMapper.xml" />
9. </mappers>
10. </configuration>

<?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="StudentEntity" type="com.manager.data.model.StudentEntity"/>

</typeAliases>

<mappers>

<mapper resource="com/manager/data/maps/StudentMapper.xml" />

</mappers>

</configuration>

**1.2.3.5修改Spring 的配置文件**

主要是添加SqlSession的制作工厂类的bean：SqlSessionFactoryBean，（在mybatis.spring包中）。需要指定配置文件位置和dataSource。  
和数据访问接口对应的实现bean。通过MapperFactoryBean创建出来。需要执行接口类全称和SqlSession工厂bean的引用。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 导入属性配置文件 -->
2. <context:property-placeholder location="classpath:mysql.properties" />
3. <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
4. <property name="driverClassName" value="${jdbc.driverClassName}" />
5. <property name="url" value="${jdbc.url}" />
6. </bean>
7. <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
8. <property name="dataSource" ref="dataSource" />
9. </bean>
10. <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
11. <property name="configLocation" value="classpath:mybatis-config.xml" />
12. <property name="dataSource" ref="dataSource" />
13. </bean>
14. <!— mapper bean -->
15. <bean id="studentMapper" class="org.mybatis.spring.MapperFactoryBean">
16. <property name="mapperInterface" value="com.manager.data.StudentMapper" />
17. <property name="sqlSessionFactory" ref="sqlSessionFactory" />
18. </bean>

<!-- 导入属性配置文件 -->

<context:property-placeholder location="classpath:mysql.properties" />

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

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

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

</bean>

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

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

<property name="configLocation" value="classpath:mybatis-config.xml" />

<property name="dataSource" ref="dataSource" />

</bean>

<!— mapper bean -->

<bean id="studentMapper" class="org.mybatis.spring.MapperFactoryBean">

<property name="mapperInterface" value="com.manager.data.StudentMapper" />

<property name="sqlSessionFactory" ref="sqlSessionFactory" />

</bean>

**也可以不定义mapper的bean，使用注解：**

将StudentMapper加入注解

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. @Repository
2. @Transactional
3. public interface StudentMapper {
4. }

@Repository

@Transactional

public interface StudentMapper {

}

对应的需要在dispatcher-servlet.xml中加入扫描：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
2. <property name="annotationClass" value="org.springframework.stereotype.Repository"/>
3. <property name="basePackage" value="com.liming.manager"/>
4. <property name="sqlSessionFactory" ref="sqlSessionFactory"/>
5. </bean>

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

<property name="annotationClass" value="org.springframework.stereotype.Repository"/>

<property name="basePackage" value="com.liming.manager"/>

<property name="sqlSessionFactory" ref="sqlSessionFactory"/>

</bean>

**1.2.4测试StudentMapper**

使用SpringMVC测试，创建一个TestController，配置tomcat，访问index.do页面进行测试：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. @Controller
2. public class TestController {
3. @Autowired
4. private StudentMapper studentMapper;
5. @RequestMapping(value = "index.do")
6. public void indexPage() {
7. StudentEntity entity = studentMapper.getStudent("10000013");
8. System.out.println("name：" + entity.getStudentName());
9. }
10. }

@Controller

public class TestController {

@Autowired

private StudentMapper studentMapper;

@RequestMapping(value = "index.do")

public void indexPage() {

StudentEntity entity = studentMapper.getStudent("10000013");

System.out.println("name：" + entity.getStudentName());

}

}

使用Junit测试：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781671)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. 使用Junit测试：
2. Java代码
3. @RunWith(value = SpringJUnit4ClassRunner.class)
4. @ContextConfiguration(value = "test-servlet.xml")
5. public class StudentMapperTest {
6. @Autowired
7. private ClassMapper classMapper;
8. @Autowired
9. private StudentMapper studentMapper;
10. @Transactional
11. public void getStudentTest(){
12. StudentEntity entity = studentMapper.getStudent("10000013");
13. System.out.println("" + entity.getStudentID() + entity.getStudentName());
14. List<StudentEntity> studentList = studentMapper.getStudentAll();
15. for( StudentEntity entityTemp : studentList){
16. System.out.println(entityTemp.getStudentName());
17. }
18. }
19. }

**二、SQL语句映射文件(1)resultMap**

SQL 映射XML 文件是所有sql语句放置的地方。需要定义一个workspace，一般定义为对应的接口类的路径。写好SQL语句映射文件后，需要在MyBAtis配置文件mappers标签中引用，例如：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <mappers>
2. <mapper resource="com/liming/manager/data/mappers/UserMapper.xml" />
3. <mapper resource="com/liming/manager/data/mappers/StudentMapper.xml" />
4. <mapper resource="com/liming/manager/data/mappers/ClassMapper.xml" />
5. <mapper resource="com/liming/manager/data/mappers/TeacherMapper.xml" />
6. </mappers>

<mappers>

<mapper resource="com/liming/manager/data/mappers/UserMapper.xml" />

<mapper resource="com/liming/manager/data/mappers/StudentMapper.xml" />

<mapper resource="com/liming/manager/data/mappers/ClassMapper.xml" />

<mapper resource="com/liming/manager/data/mappers/TeacherMapper.xml" />

</mappers>

当Java接口与XML文件在一个相对路径下时，可以不在myBatis配置文件的mappers中声明。

SQL 映射XML 文件一些初级的元素：

1. cache – 配置给定模式的缓存  
2. cache-ref – 从别的模式中引用一个缓存  
3. resultMap – 这是最复杂而却强大的一个元素了，它描述如何从结果集中加载对象  
4. sql – 一个可以被其他语句复用的SQL 块  
5. insert – 映射INSERT 语句  
6. update – 映射UPDATE 语句  
7. delete – 映射DELEETE 语句  
8. select - 映射SELECT语句

**2.1 resultMap**

resultMap 是MyBatis 中最重要最强大的元素了。你可以让你比使用JDBC 调用结果集省掉90%的代码，也可以让你做许多JDBC 不支持的事。现实上，要写一个等同类似于交互的映射这样的复杂语句，可能要上千行的代码。ResultMaps 的目的，就是这样简单的语句而不需要多余的结果映射，更多复杂的语句，除了只要一些绝对必须的语句描述关系以外，再也不需要其它的。

resultMap属性：type为java实体类；id为此resultMap的标识。

resultMap可以设置的映射：

1. constructor – 用来将结果反射给一个实例化好的类的构造器

a) idArg – ID 参数；将结果集标记为ID，以方便全局调用  
b) arg –反射到构造器的通常结果

2. id – ID 结果，将结果集标记为ID，以方便全局调用

3. result – 反射到JavaBean 属性的普通结果

4. association – 复杂类型的结合；多个结果合成的类型

a) nested result mappings – 几resultMap 自身嵌套关联，也可以引用到一个其它上

5. collection –复杂类型集合a collection of complex types

6. nested result mappings – resultMap 的集合，也可以引用到一个其它上

7. discriminator – 使用一个结果值以决定使用哪个resultMap

a) case – 基本一些值的结果映射的case 情形

i. nested result mappings –一个case 情形本身就是一个结果映射，因此也可以包括一些相同的元素，也可以引用一个外部resultMap。

**2.1.1 id、result**

id、result是最简单的映射，id为主键映射；result其他基本数据库表字段到实体类属性的映射。  
最简单的例子：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="liming.student.manager.data.model.StudentEntity" id="studentResultMap">
2. <id property="studentId" column="STUDENT\_ID" javaType="String" jdbcType="VARCHAR"/>
3. <result property="studentName" column="STUDENT\_NAME" javaType="String" jdbcType="VARCHAR"/>
4. <result property="studentSex" column="STUDENT\_SEX" javaType="int" jdbcType="INTEGER"/>
5. <result property="studentBirthday" column="STUDENT\_BIRTHDAY" javaType="Date" jdbcType="DATE"/>
6. <result property="studentPhoto" column="STUDENT\_PHOTO" javaType="byte[]" jdbcType="BLOB" typeHandler="org.apache.ibatis.type.BlobTypeHandler" />
7. </resultMap>

<resultMap type="liming.student.manager.data.model.StudentEntity" id="studentResultMap">

<id property="studentId" column="STUDENT\_ID" javaType="String" jdbcType="VARCHAR"/>

<result property="studentName" column="STUDENT\_NAME" javaType="String" jdbcType="VARCHAR"/>

<result property="studentSex" column="STUDENT\_SEX" javaType="int" jdbcType="INTEGER"/>

<result property="studentBirthday" column="STUDENT\_BIRTHDAY" javaType="Date" jdbcType="DATE"/>

<result property="studentPhoto" column="STUDENT\_PHOTO" javaType="byte[]" jdbcType="BLOB" typeHandler="org.apache.ibatis.type.BlobTypeHandler" />

</resultMap>

id、result语句属性配置细节：

|  |  |  |
| --- | --- | --- |
| 属性 | 描述 |  |
| property | 需要映射到JavaBean 的属性名称。 |  |
| column | 数据表的列名或者标签别名。 |  |
| javaType | 一个完整的类名，或者是一个类型别名。如果你匹配的是一个JavaBean，那MyBatis 通常会自行检测到。然后，如果你是要映射到一个HashMap，那你需要指定javaType 要达到的目的。 |  |
| jdbcType | 数据表支持的类型列表。这个属性只在insert,update 或delete 的时候针对允许空的列有用。JDBC 需要这项，但MyBatis 不需要。如果你是直接针对JDBC 编码，且有允许空的列，而你要指定这项。 |  |
| typeHandler | 使用这个属性可以覆写类型处理器。这项值可以是一个完整的类名，也可以是一个类型别名。 |  |

支持的JDBC类型  
为了将来的引用，MyBatis 支持下列JDBC 类型，通过JdbcType 枚举：  
BIT，FLOAT，CHAR，TIMESTAMP，OTHER，UNDEFINED，TINYINT，REAL，VARCHAR，BINARY，BLOB，NVARCHAR，SMALLINT，DOUBLE，LONGVARCHAR，VARBINARY，CLOB，NCHAR，INTEGER，NUMERIC，DATE，LONGVARBINARY，BOOLEAN，NCLOB，BIGINT，DECIMAL，TIME，NULL，CURSOR

**2.1.2 constructor**

我们使用id、result时候，需要定义java实体类的属性映射到数据库表的字段上。这个时候是使用JavaBean实现的。当然我们也可以使用实体类的构造方法来实现值的映射，这个时候是通过构造方法参数的书写的顺序来进行赋值的。  
使用construcotr功能有限（例如使用collection级联查询）。  
上面使用id、result实现的功能就可以改为：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="StudentEntity" id="studentResultMap" >
2. <constructor>
3. <idArg javaType="String" column="STUDENT\_ID"/>
4. <arg javaType="String" column="STUDENT\_NAME"/>
5. <arg javaType="String" column="STUDENT\_SEX"/>
6. <arg javaType="Date" column="STUDENT\_BIRTHDAY"/>
7. </constructor>
8. </resultMap>

<resultMap type="StudentEntity" id="studentResultMap" >

<constructor>

<idArg javaType="String" column="STUDENT\_ID"/>

<arg javaType="String" column="STUDENT\_NAME"/>

<arg javaType="String" column="STUDENT\_SEX"/>

<arg javaType="Date" column="STUDENT\_BIRTHDAY"/>

</constructor>

</resultMap>

当然，我们需要定义StudentEntity实体类的构造方法：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public StudentEntity(String studentID, String studentName, String studentSex, Date studentBirthday){
2. this.studentID = studentID;
3. this.studentName = studentName;
4. this.studentSex = studentSex;
5. this.studentBirthday = studentBirthday;
6. }

public StudentEntity(String studentID, String studentName, String studentSex, Date studentBirthday){

this.studentID = studentID;

this.studentName = studentName;

this.studentSex = studentSex;

this.studentBirthday = studentBirthday;

}

**2.1.3 association联合**

联合元素用来处理“一对一”的关系。需要指定映射的Java实体类的属性，属性的javaType（通常MyBatis 自己会识别）。对应的数据库表的列名称。如果想覆写的话返回结果的值，需要指定typeHandler。  
不同情况需要告诉MyBatis 如何加载一个联合。MyBatis 可以用两种方式加载：

1. select: 执行一个其它映射的SQL 语句返回一个Java实体类型。较灵活；  
2. resultsMap: 使用一个嵌套的结果映射来处理通过join查询结果集，映射成Java实体类型。

例如，一个班级对应一个班主任。  
首先定义好班级中的班主任属性：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. private TeacherEntity teacherEntity;

private TeacherEntity teacherEntity;

**2.1.3.1使用select实现联合**

例：班级实体类中有班主任的属性，通过联合在得到一个班级实体时，同时映射出班主任实体。

这样可以直接复用在TeacherMapper.xml文件中定义好的查询teacher根据其ID的select语句。而且不需要修改写好的SQL语句，只需要直接修改resultMap即可。

ClassMapper.xml文件部分内容：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="ClassEntity" id="classResultMap">
2. <id property="classID" column="CLASS\_ID" />
3. <result property="className" column="CLASS\_NAME" />
4. <result property="classYear" column="CLASS\_YEAR" />
5. <association property="teacherEntity" column="TEACHER\_ID" select="getTeacher"/>
6. </resultMap>
7. <select id="getClassByID" parameterType="String" resultMap="classResultMap">
8. SELECT \* FROM CLASS\_TBL CT
9. WHERE CT.CLASS\_ID = #{classID};
10. </select>

<resultMap type="ClassEntity" id="classResultMap">

<id property="classID" column="CLASS\_ID" />

<result property="className" column="CLASS\_NAME" />

<result property="classYear" column="CLASS\_YEAR" />

<association property="teacherEntity" column="TEACHER\_ID" select="getTeacher"/>

</resultMap>

<select id="getClassByID" parameterType="String" resultMap="classResultMap">

SELECT \* FROM CLASS\_TBL CT

WHERE CT.CLASS\_ID = #{classID};

</select>

TeacherMapper.xml文件部分内容：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="TeacherEntity" id="teacherResultMap">
2. <id property="teacherID" column="TEACHER\_ID" />
3. <result property="teacherName" column="TEACHER\_NAME" />
4. <result property="teacherSex" column="TEACHER\_SEX" />
5. <result property="teacherBirthday" column="TEACHER\_BIRTHDAY"/>
6. <result property="workDate" column="WORK\_DATE"/>
7. <result property="professional" column="PROFESSIONAL"/>
8. </resultMap>
9. <select id="getTeacher" parameterType="String" resultMap="teacherResultMap">
10. SELECT \*
11. FROM TEACHER\_TBL TT
12. WHERE TT.TEACHER\_ID = #{teacherID}
13. </select>

<resultMap type="TeacherEntity" id="teacherResultMap">

<id property="teacherID" column="TEACHER\_ID" />

<result property="teacherName" column="TEACHER\_NAME" />

<result property="teacherSex" column="TEACHER\_SEX" />

<result property="teacherBirthday" column="TEACHER\_BIRTHDAY"/>

<result property="workDate" column="WORK\_DATE"/>

<result property="professional" column="PROFESSIONAL"/>

</resultMap>

<select id="getTeacher" parameterType="String" resultMap="teacherResultMap">

SELECT \*

FROM TEACHER\_TBL TT

WHERE TT.TEACHER\_ID = #{teacherID}

</select>

**2.1.3.2使用resultMap实现联合**

与上面同样的功能，查询班级，同时查询器班主任。需在association中添加resultMap（在teacher的xml文件中定义好的），新写sql（查询班级表left join教师表），不需要teacher的select。

修改ClassMapper.xml文件部分内容：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="ClassEntity" id="classResultMap">
2. <id property="classID" column="CLASS\_ID" />
3. <result property="className" column="CLASS\_NAME" />
4. <result property="classYear" column="CLASS\_YEAR" />
5. <association property="teacherEntity" column="TEACHER\_ID" resultMap="teacherResultMap"/>
6. </resultMap>
7. <select id="getClassAndTeacher" parameterType="String" resultMap="classResultMap">
8. SELECT \*
9. FROM CLASS\_TBL CT LEFT JOIN TEACHER\_TBL TT ON CT.TEACHER\_ID = TT.TEACHER\_ID
10. WHERE CT.CLASS\_ID = #{classID};
11. </select>

<resultMap type="ClassEntity" id="classResultMap">

<id property="classID" column="CLASS\_ID" />

<result property="className" column="CLASS\_NAME" />

<result property="classYear" column="CLASS\_YEAR" />

<association property="teacherEntity" column="TEACHER\_ID" resultMap="teacherResultMap"/>

</resultMap>

<select id="getClassAndTeacher" parameterType="String" resultMap="classResultMap">

SELECT \*

FROM CLASS\_TBL CT LEFT JOIN TEACHER\_TBL TT ON CT.TEACHER\_ID = TT.TEACHER\_ID

WHERE CT.CLASS\_ID = #{classID};

</select>

其中的teacherResultMap请见上面TeacherMapper.xml文件部分内容中。

**2.1.4 collection聚集**

聚集元素用来处理“一对多”的关系。需要指定映射的Java实体类的属性，属性的javaType（一般为ArrayList）；列表中对象的类型ofType（Java实体类）；对应的数据库表的列名称；  
不同情况需要告诉MyBatis 如何加载一个聚集。MyBatis 可以用两种方式加载：

1. select: 执行一个其它映射的SQL 语句返回一个Java实体类型。较灵活；  
2. resultsMap: 使用一个嵌套的结果映射来处理通过join查询结果集，映射成Java实体类型。

例如，一个班级有多个学生。  
首先定义班级中的学生列表属性：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. private List<StudentEntity> studentList;

private List<StudentEntity> studentList;

**2.1.4.1使用select实现聚集**

用法和联合很类似，区别在于，这是一对多，所以一般映射过来的都是列表。所以这里需要定义javaType为ArrayList，还需要定义列表中对象的类型ofType，以及必须设置的select的语句名称（需要注意的是，这里的查询student的select语句条件必须是外键classID）。

ClassMapper.xml文件部分内容：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="ClassEntity" id="classResultMap">
2. <id property="classID" column="CLASS\_ID" />
3. <result property="className" column="CLASS\_NAME" />
4. <result property="classYear" column="CLASS\_YEAR" />
5. <association property="teacherEntity" column="TEACHER\_ID" select="getTeacher"/>
6. <collection property="studentList" column="CLASS\_ID" javaType="ArrayList" ofType="StudentEntity" select="getStudentByClassID"/>
7. </resultMap>
8. <select id="getClassByID" parameterType="String" resultMap="classResultMap">
9. SELECT \* FROM CLASS\_TBL CT
10. WHERE CT.CLASS\_ID = #{classID};
11. </select>

<resultMap type="ClassEntity" id="classResultMap">

<id property="classID" column="CLASS\_ID" />

<result property="className" column="CLASS\_NAME" />

<result property="classYear" column="CLASS\_YEAR" />

<association property="teacherEntity" column="TEACHER\_ID" select="getTeacher"/>

<collection property="studentList" column="CLASS\_ID" javaType="ArrayList" ofType="StudentEntity" select="getStudentByClassID"/>

</resultMap>

<select id="getClassByID" parameterType="String" resultMap="classResultMap">

SELECT \* FROM CLASS\_TBL CT

WHERE CT.CLASS\_ID = #{classID};

</select>

StudentMapper.xml文件部分内容：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- java属性，数据库表字段之间的映射定义 -->
2. <resultMap type="StudentEntity" id="studentResultMap">
3. <id property="studentID" column="STUDENT\_ID" />
4. <result property="studentName" column="STUDENT\_NAME" />
5. <result property="studentSex" column="STUDENT\_SEX" />
6. <result property="studentBirthday" column="STUDENT\_BIRTHDAY" />
7. </resultMap>
8. <!-- 查询学生list，根据班级id -->
9. <select id="getStudentByClassID" parameterType="String" resultMap="studentResultMap">
10. <include refid="selectStudentAll" />
11. WHERE ST.CLASS\_ID = #{classID}
12. </select>

<!-- java属性，数据库表字段之间的映射定义 -->

<resultMap type="StudentEntity" id="studentResultMap">

<id property="studentID" column="STUDENT\_ID" />

<result property="studentName" column="STUDENT\_NAME" />

<result property="studentSex" column="STUDENT\_SEX" />

<result property="studentBirthday" column="STUDENT\_BIRTHDAY" />

</resultMap>

<!-- 查询学生list，根据班级id -->

<select id="getStudentByClassID" parameterType="String" resultMap="studentResultMap">

<include refid="selectStudentAll" />

WHERE ST.CLASS\_ID = #{classID}

</select>

**2.1.4.2使用resultMap实现聚集**

使用resultMap，就需要重写一个sql，left join学生表。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="ClassEntity" id="classResultMap">
2. <id property="classID" column="CLASS\_ID" />
3. <result property="className" column="CLASS\_NAME" />
4. <result property="classYear" column="CLASS\_YEAR" />
5. <association property="teacherEntity" column="TEACHER\_ID" resultMap="teacherResultMap"/>
6. <collection property="studentList" column="CLASS\_ID" javaType="ArrayList" ofType="StudentEntity" resultMap="studentResultMap"/>
7. </resultMap>
8. <select id="getClassAndTeacherStudent" parameterType="String" resultMap="classResultMap">
9. SELECT \*
10. FROM CLASS\_TBL CT
11. LEFT JOIN STUDENT\_TBL ST
12. ON CT.CLASS\_ID = ST.CLASS\_ID
13. LEFT JOIN TEACHER\_TBL TT
14. ON CT.TEACHER\_ID = TT.TEACHER\_ID
15. WHERE CT.CLASS\_ID = #{classID};
16. </select>

<resultMap type="ClassEntity" id="classResultMap">

<id property="classID" column="CLASS\_ID" />

<result property="className" column="CLASS\_NAME" />

<result property="classYear" column="CLASS\_YEAR" />

<association property="teacherEntity" column="TEACHER\_ID" resultMap="teacherResultMap"/>

<collection property="studentList" column="CLASS\_ID" javaType="ArrayList" ofType="StudentEntity" resultMap="studentResultMap"/>

</resultMap>

<select id="getClassAndTeacherStudent" parameterType="String" resultMap="classResultMap">

SELECT \*

FROM CLASS\_TBL CT

LEFT JOIN STUDENT\_TBL ST

ON CT.CLASS\_ID = ST.CLASS\_ID

LEFT JOIN TEACHER\_TBL TT

ON CT.TEACHER\_ID = TT.TEACHER\_ID

WHERE CT.CLASS\_ID = #{classID};

</select>

其中的teacherResultMap请见上面TeacherMapper.xml文件部分内容中。studentResultMap请见上面StudentMapper.xml文件部分内容中。

**2.1.5discriminator鉴别器**

有时一个单独的数据库查询也许返回很多不同（但是希望有些关联）数据类型的结果集。鉴别器元素就是被设计来处理这个情况的，还有包括类的继承层次结构。鉴别器非常容易理解，因为它的表现很像Java语言中的switch语句。

定义鉴别器指定了column和javaType属性。列是MyBatis查找比较值的地方。JavaType是需要被用来保证等价测试的合适类型（尽管字符串在很多情形下都会有用）。

下面这个例子为，当classId为20000001时，才映射classId属性。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781878)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <resultMap type="liming.student.manager.data.model.StudentEntity" id="resultMap\_studentEntity\_discriminator">
2. <id property="studentId" column="STUDENT\_ID" javaType="String" jdbcType="VARCHAR"/>
3. <result property="studentName" column="STUDENT\_NAME" javaType="String" jdbcType="VARCHAR"/>
4. <result property="studentSex" column="STUDENT\_SEX" javaType="int" jdbcType="INTEGER"/>
5. <result property="studentBirthday" column="STUDENT\_BIRTHDAY" javaType="Date" jdbcType="DATE"/>
6. <result property="studentPhoto" column="STUDENT\_PHOTO" javaType="byte[]" jdbcType="BLOB" typeHandler="org.apache.ibatis.type.BlobTypeHandler" />
7. <result property="placeId" column="PLACE\_ID" javaType="String" jdbcType="VARCHAR"/>
8. <discriminator column="CLASS\_ID" javaType="String" jdbcType="VARCHAR">
9. <case value="20000001" resultType="liming.student.manager.data.model.StudentEntity" >
10. <result property="classId" column="CLASS\_ID" javaType="String" jdbcType="VARCHAR"/>
11. </case>
12. </discriminator>
13. </resultMap>

**二、SQL语句映射文件(2)增删改查、参数、缓存**

**2.2 select**

一个select 元素非常简单。例如：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 查询学生，根据id -->
2. <select id="getStudent" parameterType="String" resultMap="studentResultMap">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.CLASS\_ID
8. FROM STUDENT\_TBL ST
9. WHERE ST.STUDENT\_ID = #{studentID}
10. </select>

<!-- 查询学生，根据id -->

<select id="getStudent" parameterType="String" resultMap="studentResultMap">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.CLASS\_ID

FROM STUDENT\_TBL ST

WHERE ST.STUDENT\_ID = #{studentID}

</select>

这条语句就叫做‘getStudent，有一个String参数，并返回一个StudentEntity类型的对象。  
注意参数的标识是：#{studentID}。

select 语句属性配置细节：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 描述 | 取值 | 默认 |
| id | 在这个模式下唯一的标识符，可被其它语句引用 |  |  |
| parameterType | 传给此语句的参数的完整类名或别名 |  |  |
| resultType | 语句返回值类型的整类名或别名。注意，如果是集合，那么这里填写的是集合的项的整类名或别名，而不是集合本身的类名。（resultType 与resultMap 不能并用） |  |  |
| resultMap | 引用的外部resultMap 名。结果集映射是MyBatis 中最强大的特性。许多复杂的映射都可以轻松解决。（resultType 与resultMap 不能并用） |  |  |
| flushCache | 如果设为true，则会在每次语句调用的时候就会清空缓存。select 语句默认设为false | true|false | false |
| useCache | 如果设为true，则语句的结果集将被缓存。select 语句默认设为false true|false false timeout 设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定 | true|false | false |
| timeout | 设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定 | 正整数 | 未设置 |
| fetchSize | 设置一个值后，驱动器会在结果集数目达到此数值后，激发返回，默认为不设值，由驱动器自己决定 | 正整数 | 驱动器决定 |
| statementType | statement，preparedstatement，callablestatement。 预准备语句、可调用语句 | STATEMENT PREPARED CALLABLE | PREPARED |
| resultSetType | forward\_only，scroll\_sensitive，scroll\_insensitive 只转发，滚动敏感，不区分大小写的滚动 | FORWARD\_ONLY SCROLL\_SENSITIVE SCROLL\_INSENSITIVE | 驱动器决定 |

**2.3 insert**

一个简单的insert语句：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 插入学生 -->
2. <insert id="insertStudent" parameterType="StudentEntity">
3. INSERT INTO STUDENT\_TBL (STUDENT\_ID,
4. STUDENT\_NAME,
5. STUDENT\_SEX,
6. STUDENT\_BIRTHDAY,
7. CLASS\_ID)
8. VALUES (#{studentID},
9. #{studentName},
10. #{studentSex},
11. #{studentBirthday},
12. #{classEntity.classID})
13. </insert>

<!-- 插入学生 -->

<insert id="insertStudent" parameterType="StudentEntity">

INSERT INTO STUDENT\_TBL (STUDENT\_ID,

STUDENT\_NAME,

STUDENT\_SEX,

STUDENT\_BIRTHDAY,

CLASS\_ID)

VALUES (#{studentID},

#{studentName},

#{studentSex},

#{studentBirthday},

#{classEntity.classID})

</insert>

insert可以使用数据库支持的自动生成主键策略，设置useGeneratedKeys=”true”，然后把keyProperty 设成对应的列，就搞定了。比如说上面的StudentEntity 使用auto-generated 为id 列生成主键.  
还可以使用selectKey元素。下面例子，使用mysql数据库nextval('student')为自定义函数，用来生成一个key。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 插入学生 自动主键-->
2. <insert id="insertStudentAutoKey" parameterType="StudentEntity">
3. <selectKey keyProperty="studentID" resultType="String" order="BEFORE">
4. select nextval('student')
5. </selectKey>
6. INSERT INTO STUDENT\_TBL (STUDENT\_ID,
7. STUDENT\_NAME,
8. STUDENT\_SEX,
9. STUDENT\_BIRTHDAY,
10. CLASS\_ID)
11. VALUES (#{studentID},
12. #{studentName},
13. #{studentSex},
14. #{studentBirthday},
15. #{classEntity.classID})
16. </insert>

<!-- 插入学生 自动主键-->

<insert id="insertStudentAutoKey" parameterType="StudentEntity">

<selectKey keyProperty="studentID" resultType="String" order="BEFORE">

select nextval('student')

</selectKey>

INSERT INTO STUDENT\_TBL (STUDENT\_ID,

STUDENT\_NAME,

STUDENT\_SEX,

STUDENT\_BIRTHDAY,

CLASS\_ID)

VALUES (#{studentID},

#{studentName},

#{studentSex},

#{studentBirthday},

#{classEntity.classID})

</insert>

insert语句属性配置细节：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 描述 | 取值 | 默认 |
| id | 在这个模式下唯一的标识符，可被其它语句引用 |  |  |
| parameterType | 传给此语句的参数的完整类名或别名 |  |  |
| flushCache | 如果设为true，则会在每次语句调用的时候就会清空缓存。select 语句默认设为false | true|false | false |
| useCache | 如果设为true，则语句的结果集将被缓存。select 语句默认设为false true|false false timeout 设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定 | true|false | false |
| timeout | 设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定 | 正整数 | 未设置 |
| fetchSize | 设置一个值后，驱动器会在结果集数目达到此数值后，激发返回，默认为不设值，由驱动器自己决定 | 正整数 | 驱动器决定 |
| statementType | statement，preparedstatement，callablestatement。 预准备语句、可调用语句 | STATEMENT PREPARED CALLABLE | PREPARED |
| useGeneratedKeys | 告诉MyBatis 使用JDBC 的getGeneratedKeys 方法来获取数据库自己生成的主键（MySQL、SQLSERVER 等  关系型数据库会有自动生成的字段）。默认：false | true|false | false |
| keyProperty | 标识一个将要被MyBatis 设置进getGeneratedKeys 的key 所返回的值，或者为insert 语句使用一个selectKey  子元素。 |  |  |

selectKey语句属性配置细节：

|  |  |  |
| --- | --- | --- |
| 属性 | 描述 | 取值 |
| keyProperty | selectKey 语句生成结果需要设置的属性。 |  |
| resultType | 生成结果类型，MyBatis 允许使用基本的数据类型，包括String 、int类型。 |  |
| order | 可以设成BEFORE 或者AFTER，如果设为BEFORE，那它会先选择主键，然后设置keyProperty，再执行insert语句；如果设为AFTER，它就先运行insert 语句再运行selectKey 语句，通常是insert 语句中内部调用数据库（像Oracle）内嵌的序列机制。 | BEFORE AFTER |
| statementType | 像上面的那样， MyBatis 支持STATEMENT，PREPARED和CALLABLE 的语句形式， 对应Statement ，PreparedStatement 和CallableStatement 响应 | STATEMENT PREPARED CALLABLE |

**2.4 update、delete**

一个简单的update：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 更新学生信息 -->
2. <update id="updateStudent" parameterType="StudentEntity">
3. UPDATE STUDENT\_TBL
4. SET STUDENT\_TBL.STUDENT\_NAME = #{studentName},
5. STUDENT\_TBL.STUDENT\_SEX = #{studentSex},
6. STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},
7. STUDENT\_TBL.CLASS\_ID = #{classEntity.classID}
8. WHERE STUDENT\_TBL.STUDENT\_ID = #{studentID};
9. </update>

<!-- 更新学生信息 -->

<update id="updateStudent" parameterType="StudentEntity">

UPDATE STUDENT\_TBL

SET STUDENT\_TBL.STUDENT\_NAME = #{studentName},

STUDENT\_TBL.STUDENT\_SEX = #{studentSex},

STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},

STUDENT\_TBL.CLASS\_ID = #{classEntity.classID}

WHERE STUDENT\_TBL.STUDENT\_ID = #{studentID};

</update>

一个简单的delete：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 删除学生 -->
2. <delete id="deleteStudent" parameterType="StudentEntity">
3. DELETE FROM STUDENT\_TBL WHERE STUDENT\_ID = #{studentID}
4. </delete>

<!-- 删除学生 -->

<delete id="deleteStudent" parameterType="StudentEntity">

DELETE FROM STUDENT\_TBL WHERE STUDENT\_ID = #{studentID}

</delete>

update、delete语句属性配置细节：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 描述 | 取值 | 默认 |
| id | 在这个模式下唯一的标识符，可被其它语句引用 |  |  |
| parameterType | 传给此语句的参数的完整类名或别名 |  |  |
| flushCache | 如果设为true，则会在每次语句调用的时候就会清空缓存。select 语句默认设为false | true|false | false |
| useCache | 如果设为true，则语句的结果集将被缓存。select 语句默认设为false true|false false timeout 设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定 | true|false | false |
| timeout | 设置驱动器在抛出异常前等待回应的最长时间，默认为不设值，由驱动器自己决定 | 正整数 | 未设置 |
| fetchSize | 设置一个值后，驱动器会在结果集数目达到此数值后，激发返回，默认为不设值，由驱动器自己决定 | 正整数 | 驱动器决定 |
| statementType | statement，preparedstatement，callablestatement。 预准备语句、可调用语句 | STATEMENT PREPARED CALLABLE | PREPARED |

**2.5 sql**

Sql元素用来定义一个可以复用的SQL 语句段，供其它语句调用。比如：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 复用sql语句 查询student表所有字段 -->
2. <sql id="selectStudentAll">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.CLASS\_ID
8. FROM STUDENT\_TBL ST
9. </sql>

<!-- 复用sql语句 查询student表所有字段 -->

<sql id="selectStudentAll">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.CLASS\_ID

FROM STUDENT\_TBL ST

</sql>

这样，在select的语句中就可以直接引用使用了，将上面select语句改成：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 查询学生，根据id -->
2. <select id="getStudent" parameterType="String" resultMap="studentResultMap">
3. <include refid="selectStudentAll"/>
4. WHERE ST.STUDENT\_ID = #{studentID}
5. </select>

<!-- 查询学生，根据id -->

<select id="getStudent" parameterType="String" resultMap="studentResultMap">

<include refid="selectStudentAll"/>

WHERE ST.STUDENT\_ID = #{studentID}

</select>

**2.6parameters**

上面很多地方已经用到了参数，比如查询、修改、删除的条件，插入，修改的数据等，MyBatis可以使用的基本数据类型和Java的复杂数据类型。  
基本数据类型，String，int，date等。  
但是使用基本数据类型，只能提供一个参数，所以需要使用Java实体类，或Map类型做参数类型。通过#{}可以直接得到其属性。

**2.6.1基本类型参数**

根据入学时间，检索学生列表：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 查询学生list，根据入学时间 -->
2. <select id="getStudentListByDate" parameterType="Date" resultMap="studentResultMap">
3. SELECT \*
4. FROM STUDENT\_TBL ST LEFT JOIN CLASS\_TBL CT ON ST.CLASS\_ID = CT.CLASS\_ID
5. WHERE CT.CLASS\_YEAR = #{classYear};
6. </select>

<!-- 查询学生list，根据入学时间 -->

<select id="getStudentListByDate" parameterType="Date" resultMap="studentResultMap">

SELECT \*

FROM STUDENT\_TBL ST LEFT JOIN CLASS\_TBL CT ON ST.CLASS\_ID = CT.CLASS\_ID

WHERE CT.CLASS\_YEAR = #{classYear};

</select>

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. List<StudentEntity> studentList = studentMapper.getStudentListByClassYear(StringUtil.parse("2007-9-1"));
2. for (StudentEntity entityTemp : studentList) {
3. System.out.println(entityTemp.toString());
4. }

List<StudentEntity> studentList = studentMapper.getStudentListByClassYear(StringUtil.parse("2007-9-1"));

for (StudentEntity entityTemp : studentList) {

System.out.println(entityTemp.toString());

}

**2.6.2Java实体类型参数**

根据姓名和性别，检索学生列表。使用实体类做参数：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 查询学生list，like姓名、=性别，参数entity类型 -->
2. <select id="getStudentListWhereEntity" parameterType="StudentEntity" resultMap="studentResultMap">
3. SELECT \* from STUDENT\_TBL ST
4. WHERE ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName}),'%')
5. AND ST.STUDENT\_SEX = #{studentSex}
6. </select>

<!-- 查询学生list，like姓名、=性别，参数entity类型 -->

<select id="getStudentListWhereEntity" parameterType="StudentEntity" resultMap="studentResultMap">

SELECT \* from STUDENT\_TBL ST

WHERE ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName}),'%')

AND ST.STUDENT\_SEX = #{studentSex}

</select>

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. StudentEntity entity = new StudentEntity();
2. entity.setStudentName("李");
3. entity.setStudentSex("男");
4. List<StudentEntity> studentList = studentMapper.getStudentListWhereEntity(entity);
5. for (StudentEntity entityTemp : studentList) {
6. System.out.println(entityTemp.toString());
7. }

StudentEntity entity = new StudentEntity();

entity.setStudentName("李");

entity.setStudentSex("男");

List<StudentEntity> studentList = studentMapper.getStudentListWhereEntity(entity);

for (StudentEntity entityTemp : studentList) {

System.out.println(entityTemp.toString());

}

**2.6.3Map参数**

根据姓名和性别，检索学生列表。使用Map做参数：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 查询学生list，=性别，参数map类型 -->
2. <select id="getStudentListWhereMap" parameterType="Map" resultMap="studentResultMap">
3. SELECT \* from STUDENT\_TBL ST
4. WHERE ST.STUDENT\_SEX = #{sex}
5. AND ST.STUDENT\_SEX = #{sex}
6. </select>

<!-- 查询学生list，=性别，参数map类型 -->

<select id="getStudentListWhereMap" parameterType="Map" resultMap="studentResultMap">

SELECT \* from STUDENT\_TBL ST

WHERE ST.STUDENT\_SEX = #{sex}

AND ST.STUDENT\_SEX = #{sex}

</select>

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. Map<String, String> map = new HashMap<String, String>();
2. map.put("sex", "女");
3. map.put("name", "李");
4. List<StudentEntity> studentList = studentMapper.getStudentListWhereMap(map);
5. for (StudentEntity entityTemp : studentList) {
6. System.out.println(entityTemp.toString());
7. }

Map<String, String> map = new HashMap<String, String>();

map.put("sex", "女");

map.put("name", "李");

List<StudentEntity> studentList = studentMapper.getStudentListWhereMap(map);

for (StudentEntity entityTemp : studentList) {

System.out.println(entityTemp.toString());

}

**2.6.4多参数的实现**

如果想传入多个参数，则需要在接口的参数上添加@Param注解。给出一个实例：  
接口写法：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public List<StudentEntity> getStudentListWhereParam(@Param(value = "name") String name, @Param(value = "sex") String sex, @Param(value = "birthday") Date birthdar, @Param(value = "classEntity") ClassEntity classEntity);

public List<StudentEntity> getStudentListWhereParam(@Param(value = "name") String name, @Param(value = "sex") String sex, @Param(value = "birthday") Date birthdar, @Param(value = "classEntity") ClassEntity classEntity);

SQL写法：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 查询学生list，like姓名、=性别、=生日、=班级，多参数方式 -->
2. <select id="getStudentListWhereParam" resultMap="studentResultMap">
3. SELECT \* from STUDENT\_TBL ST
4. <where>
5. <if test="name!=null and name!='' ">
6. ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{name}),'%')
7. </if>
8. <if test="sex!= null and sex!= '' ">
9. AND ST.STUDENT\_SEX = #{sex}
10. </if>
11. <if test="birthday!=null">
12. AND ST.STUDENT\_BIRTHDAY = #{birthday}
13. </if>
14. <if test="classEntity!=null and classEntity.classID !=null and classEntity.classID!='' ">
15. AND ST.CLASS\_ID = #{classEntity.classID}
16. </if>
17. </where>
18. </select>

<!-- 查询学生list，like姓名、=性别、=生日、=班级，多参数方式 -->

<select id="getStudentListWhereParam" resultMap="studentResultMap">

SELECT \* from STUDENT\_TBL ST

<where>

<if test="name!=null and name!='' ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{name}),'%')

</if>

<if test="sex!= null and sex!= '' ">

AND ST.STUDENT\_SEX = #{sex}

</if>

<if test="birthday!=null">

AND ST.STUDENT\_BIRTHDAY = #{birthday}

</if>

<if test="classEntity!=null and classEntity.classID !=null and classEntity.classID!='' ">

AND ST.CLASS\_ID = #{classEntity.classID}

</if>

</where>

</select>

进行查询：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. List<StudentEntity> studentList = studentMapper.getStudentListWhereParam("", "",StringUtil.parse("1985-05-28"), classMapper.getClassByID("20000002"));
2. for (StudentEntity entityTemp : studentList) {
3. System.out.println(entityTemp.toString());
4. }

List<StudentEntity> studentList = studentMapper.getStudentListWhereParam("", "",StringUtil.parse("1985-05-28"), classMapper.getClassByID("20000002"));

for (StudentEntity entityTemp : studentList) {

System.out.println(entityTemp.toString());

}

**2.6.5字符串代入法**

默认的情况下，使用#{}语法会促使MyBatis 生成PreparedStatement 属性并且使用PreparedStatement 的参数（=？）来安全的设置值。尽量这些是快捷安全，也是经常使用的。但有时候你可能想直接未更改的字符串代入到SQL 语句中。比如说，对于ORDER BY，你可能会这样使用：ORDER BY ${columnName}但MyBatis 不会修改和规避掉这个字符串。  
注意：这样地接收和应用一个用户输入到未更改的语句中，是非常不安全的。这会让用户能植入破坏代码，所以，要么要求字段不要允许客户输入，要么你直接来检测他的合法性 。

**2.7 cache缓存**

MyBatis 包含一个强在的、可配置、可定制的缓存机制。MyBatis 3 的缓存实现有了许多改进，既强劲也更容易配置。默认的情况，缓存是没有开启，除了会话缓存以外，它可以提高性能，且能解决全局依赖。开启二级缓存，你只需要在SQL 映射文件中加入简单的一行：<cache/>

这句简单的语句的作用如下：

1. 所有在映射文件里的select 语句都将被缓存。  
2. 所有在映射文件里insert,update 和delete 语句会清空缓存。  
3. 缓存使用“最近很少使用”算法来回收  
4. 缓存不会被设定的时间所清空。  
5. 每个缓存可以存储1024 个列表或对象的引用（不管查询出来的结果是什么）。  
6. 缓存将作为“读/写”缓存，意味着获取的对象不是共享的且对调用者是安全的。不会有其它的调用  
7. 者或线程潜在修改。

例如，创建一个FIFO 缓存让60 秒就清空一次，存储512 个对象结果或列表引用，并且返回的结果是只读。因为在不用的线程里的两个调用者修改它们可能会导致引用冲突。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <cache eviction="FIFO" flushInterval="60000" size="512" readOnly="true">
2. </cache>

<cache eviction="FIFO" flushInterval="60000" size="512" readOnly="true">

</cache>

还可以在不同的命名空间里共享同一个缓存配置或者实例。在这种情况下，你就可以使用cache-ref 来引用另外一个缓存。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/781911)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <cache-ref namespace="com.liming.manager.data.StudentMapper"/>

<cache-ref namespace="com.liming.manager.data.StudentMapper"/>

Cache 语句属性配置细节：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 说明 | 取值 | 默认值 |
| eviction | 缓存策略： LRU - 最近最少使用法：移出最近较长周期内都没有被使用的对象。 FIFI- 先进先出：移出队列里较早的对象 SOFT - 软引用：基于软引用规则，使用垃圾回收机制来移出对象 WEAK - 弱引用：基于弱引用规则，使用垃圾回收机制来强制性地移出对象 | LRU FIFI SOFT WEAK | LRU |
| flushInterval | 代表一个合理的毫秒总计时间。默认是不设置，因此使用无间隔清空即只能调用语句来清空。 | 正整数 | 不设置 |
| size | 缓存的对象的大小 | 正整数 | 1024 |
| readOnly | 只读缓存将对所有调用者返回同一个实例。因此都不能被修改，这可以极大的提高性能。可写的缓存将通过序列  化来返回一个缓存对象的拷贝。这会比较慢，但是比较安全。所以默认值是false。 | true|false | false |

**三、动态SQL语句**

有些时候，sql语句where条件中，需要一些安全判断，例如按某一条件查询时如果传入的参数是空，此时查询出的结果很可能是空的，也许我们需要参数为空时，是查出全部的信息。使用Oracle的序列、mysql的函数生成Id。这时我们可以使用动态sql。

下文均采用mysql语法和函数（例如字符串链接函数CONCAT）。

**3.1 selectKey 标签**

在insert语句中，在Oracle经常使用序列、在MySQL中使用函数来自动生成插入表的主键，而且需要方法能返回这个生成主键。使用myBatis的selectKey标签可以实现这个效果。

下面例子，使用mysql数据库自定义函数nextval('student')，用来生成一个key，并把他设置到传入的实体类中的studentId属性上。所以在执行完此方法后，边可以通过这个实体类获取生成的key。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 插入学生 自动主键-->
2. <insert id="createStudentAutoKey" parameterType="liming.student.manager.data.model.StudentEntity" keyProperty="studentId">
3. <selectKey keyProperty="studentId" resultType="String" order="BEFORE">
4. select nextval('student')
5. </selectKey>
6. INSERT INTO STUDENT\_TBL(STUDENT\_ID,
7. STUDENT\_NAME,
8. STUDENT\_SEX,
9. STUDENT\_BIRTHDAY,
10. STUDENT\_PHOTO,
11. CLASS\_ID,
12. PLACE\_ID)
13. VALUES (#{studentId},
14. #{studentName},
15. #{studentSex},
16. #{studentBirthday},
17. #{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},
18. #{classId},
19. #{placeId})
20. </insert>

<!-- 插入学生 自动主键-->

<insert id="createStudentAutoKey" parameterType="liming.student.manager.data.model.StudentEntity" keyProperty="studentId">

<selectKey keyProperty="studentId" resultType="String" order="BEFORE">

select nextval('student')

</selectKey>

INSERT INTO STUDENT\_TBL(STUDENT\_ID,

STUDENT\_NAME,

STUDENT\_SEX,

STUDENT\_BIRTHDAY,

STUDENT\_PHOTO,

CLASS\_ID,

PLACE\_ID)

VALUES (#{studentId},

#{studentName},

#{studentSex},

#{studentBirthday},

#{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},

#{classId},

#{placeId})

</insert>

调用接口方法，和获取自动生成key

Java代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. StudentEntity entity = new StudentEntity();
2. entity.setStudentName("黎明你好");
3. entity.setStudentSex(1);
4. entity.setStudentBirthday(DateUtil.parse("1985-05-28"));
5. entity.setClassId("20000001");
6. entity.setPlaceId("70000001");
7. this.dynamicSqlMapper.createStudentAutoKey(entity);
8. System.out.println("新增学生ID: " + entity.getStudentId());

StudentEntity entity = new StudentEntity();

entity.setStudentName("黎明你好");

entity.setStudentSex(1);

entity.setStudentBirthday(DateUtil.parse("1985-05-28"));

entity.setClassId("20000001");

entity.setPlaceId("70000001");

this.dynamicSqlMapper.createStudentAutoKey(entity);

System.out.println("新增学生ID: " + entity.getStudentId());

selectKey语句属性配置细节：

|  |  |  |
| --- | --- | --- |
| 属性 | 描述 | 取值 |
| keyProperty | selectKey 语句生成结果需要设置的属性。 |  |
| resultType | 生成结果类型，MyBatis 允许使用基本的数据类型，包括String 、int类型。 |  |
| order | 1：BEFORE，会先选择主键，然后设置keyProperty，再执行insert语句；  2：AFTER，就先运行insert 语句再运行selectKey 语句。 | BEFORE  AFTER |
| statementType | MyBatis 支持STATEMENT，PREPARED和CALLABLE 的语句形式， 对应Statement ，PreparedStatement 和CallableStatement 响应 | STATEMENT  PREPARED  CALLABLE |

**3.2 if标签**

if标签可用在许多类型的sql语句中，我们以查询为例。首先看一个很普通的查询：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 查询学生list，like姓名 -->
2. <select id="getStudentListLikeName" parameterType="StudentEntity" resultMap="studentResultMap">
3. SELECT \* from STUDENT\_TBL ST
4. WHERE ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName}),'%')
5. </select>

<!-- 查询学生list，like姓名 -->

<select id="getStudentListLikeName" parameterType="StudentEntity" resultMap="studentResultMap">

SELECT \* from STUDENT\_TBL ST

WHERE ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName}),'%')

</select>

但是此时如果studentName或studentSex为null，此语句很可能报错或查询结果为空。此时我们使用if动态sql语句先进行判断，如果值为null或等于空字符串，我们就不进行此条件的判断，增加灵活性。

参数为实体类StudentEntity。将实体类中所有的属性均进行判断，如果不为空则执行判断条件。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 2 if(判断参数) - 将实体类不为空的属性作为where条件 -->
2. <select id="getStudentList\_if" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.STUDENT\_PHOTO,
8. ST.CLASS\_ID,
9. ST.PLACE\_ID
10. FROM STUDENT\_TBL ST
11. WHERE
12. <if test="studentName !=null ">
13. ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')
14. </if>
15. <if test="studentSex != null and studentSex != '' ">
16. AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}
17. </if>
18. <if test="studentBirthday != null ">
19. AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}
20. </if>
21. <if test="classId != null and classId!= '' ">
22. AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}
23. </if>
24. <if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">
25. AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}
26. </if>
27. <if test="placeId != null and placeId != '' ">
28. AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}
29. </if>
30. <if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">
31. AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}
32. </if>
33. <if test="studentId != null and studentId != '' ">
34. AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}
35. </if>
36. </select>

<!-- 2 if(判断参数) - 将实体类不为空的属性作为where条件 -->

<select id="getStudentList\_if" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

WHERE

<if test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')

</if>

<if test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</if>

<if test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</if>

<if test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</if>

<if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</if>

<if test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</if>

<if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</if>

<if test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</if>

</select>

使用时比较灵活， new一个这样的实体类，我们需要限制那个条件，只需要附上相应的值就会where这个条件，相反不去赋值就可以不在where中判断。

Java代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public void select\_test\_2\_1() {
2. StudentEntity entity = new StudentEntity();
3. entity.setStudentName("");
4. entity.setStudentSex(1);
5. entity.setStudentBirthday(DateUtil.parse("1985-05-28"));
6. entity.setClassId("20000001");
7. //entity.setPlaceId("70000001");
8. List<StudentEntity> list = this.dynamicSqlMapper.getStudentList\_if(entity);
9. for (StudentEntity e : list) {
10. System.out.println(e.toString());
11. }
12. }

public void select\_test\_2\_1() {

StudentEntity entity = new StudentEntity();

entity.setStudentName("");

entity.setStudentSex(1);

entity.setStudentBirthday(DateUtil.parse("1985-05-28"));

entity.setClassId("20000001");

//entity.setPlaceId("70000001");

List<StudentEntity> list = this.dynamicSqlMapper.getStudentList\_if(entity);

for (StudentEntity e : list) {

System.out.println(e.toString());

}

}

**3.3 if + where 的条件判断**

当where中的条件使用的if标签较多时，这样的组合可能会导致错误。我们以在3.1中的查询语句为例子，当java代码按如下方法调用时：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. @Test
2. public void select\_test\_2\_1() {
3. StudentEntity entity = new StudentEntity();
4. entity.setStudentName(null);
5. entity.setStudentSex(1);
6. List<StudentEntity> list = this.dynamicSqlMapper.getStudentList\_if(entity);
7. for (StudentEntity e : list) {
8. System.out.println(e.toString());
9. }
10. }

@Test

public void select\_test\_2\_1() {

StudentEntity entity = new StudentEntity();

entity.setStudentName(null);

entity.setStudentSex(1);

List<StudentEntity> list = this.dynamicSqlMapper.getStudentList\_if(entity);

for (StudentEntity e : list) {

System.out.println(e.toString());

}

}

如果上面例子，参数studentName为null，将不会进行STUDENT\_NAME列的判断，则会直接导“WHERE AND”关键字多余的错误SQL。

这时我们可以使用where动态语句来解决。这个“where”标签会知道如果它包含的标签中有返回值的话，它就插入一个‘where’。此外，如果标签返回的内容是以AND 或OR 开头的，则它会剔除掉。

上面例子修改为：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 3 select - where/if(判断参数) - 将实体类不为空的属性作为where条件 -->
2. <select id="getStudentList\_whereIf" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.STUDENT\_PHOTO,
8. ST.CLASS\_ID,
9. ST.PLACE\_ID
10. FROM STUDENT\_TBL ST
11. <where>
12. <if test="studentName !=null ">
13. ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')
14. </if>
15. <if test="studentSex != null and studentSex != '' ">
16. AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}
17. </if>
18. <if test="studentBirthday != null ">
19. AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}
20. </if>
21. <if test="classId != null and classId!= '' ">
22. AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}
23. </if>
24. <if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">
25. AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}
26. </if>
27. <if test="placeId != null and placeId != '' ">
28. AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}
29. </if>
30. <if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">
31. AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}
32. </if>
33. <if test="studentId != null and studentId != '' ">
34. AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}
35. </if>
36. </where>
37. </select>

<!-- 3 select - where/if(判断参数) - 将实体类不为空的属性作为where条件 -->

<select id="getStudentList\_whereIf" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

<where>

<if test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')

</if>

<if test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</if>

<if test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</if>

<if test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</if>

<if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</if>

<if test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</if>

<if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</if>

<if test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</if>

</where>

</select>

**3.4 if + set 的更新语句**

当update语句中没有使用if标签时，如果有一个参数为null，都会导致错误。

当在update语句中使用if标签时，如果前面的if没有执行，则或导致逗号多余错误。使用set标签可以将动态的配置SET 关键字，和剔除追加到条件末尾的任何不相关的逗号。

使用if+set标签修改后，如果某项为null则不进行更新，而是保持数据库原值。如下示例：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 4 if/set(判断参数) - 将实体类不为空的属性更新 -->
2. <update id="updateStudent\_if\_set" parameterType="liming.student.manager.data.model.StudentEntity">
3. UPDATE STUDENT\_TBL
4. <set>
5. <if test="studentName != null and studentName != '' ">
6. STUDENT\_TBL.STUDENT\_NAME = #{studentName},
7. </if>
8. <if test="studentSex != null and studentSex != '' ">
9. STUDENT\_TBL.STUDENT\_SEX = #{studentSex},
10. </if>
11. <if test="studentBirthday != null ">
12. STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},
13. </if>
14. <if test="studentPhoto != null ">
15. STUDENT\_TBL.STUDENT\_PHOTO = #{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},
16. </if>
17. <if test="classId != '' ">
18. STUDENT\_TBL.CLASS\_ID = #{classId}
19. </if>
20. <if test="placeId != '' ">
21. STUDENT\_TBL.PLACE\_ID = #{placeId}
22. </if>
23. </set>
24. WHERE STUDENT\_TBL.STUDENT\_ID = #{studentId};
25. </update>

<!-- 4 if/set(判断参数) - 将实体类不为空的属性更新 -->

<update id="updateStudent\_if\_set" parameterType="liming.student.manager.data.model.StudentEntity">

UPDATE STUDENT\_TBL

<set>

<if test="studentName != null and studentName != '' ">

STUDENT\_TBL.STUDENT\_NAME = #{studentName},

</if>

<if test="studentSex != null and studentSex != '' ">

STUDENT\_TBL.STUDENT\_SEX = #{studentSex},

</if>

<if test="studentBirthday != null ">

STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},

</if>

<if test="studentPhoto != null ">

STUDENT\_TBL.STUDENT\_PHOTO = #{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},

</if>

<if test="classId != '' ">

STUDENT\_TBL.CLASS\_ID = #{classId}

</if>

<if test="placeId != '' ">

STUDENT\_TBL.PLACE\_ID = #{placeId}

</if>

</set>

WHERE STUDENT\_TBL.STUDENT\_ID = #{studentId};

</update>

**3.5 if + trim代替where/set标签**

trim是更灵活的去处多余关键字的标签，他可以实践where和set的效果。

**3.5.1trim代替where**

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 5.1 if/trim代替where(判断参数) - 将实体类不为空的属性作为where条件 -->
2. <select id="getStudentList\_if\_trim" resultMap="resultMap\_studentEntity">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.STUDENT\_PHOTO,
8. ST.CLASS\_ID,
9. ST.PLACE\_ID
10. FROM STUDENT\_TBL ST
11. <trim prefix="WHERE" prefixOverrides="AND|OR">
12. <if test="studentName !=null ">
13. ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')
14. </if>
15. <if test="studentSex != null and studentSex != '' ">
16. AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}
17. </if>
18. <if test="studentBirthday != null ">
19. AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}
20. </if>
21. <if test="classId != null and classId!= '' ">
22. AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}
23. </if>
24. <if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">
25. AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}
26. </if>
27. <if test="placeId != null and placeId != '' ">
28. AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}
29. </if>
30. <if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">
31. AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}
32. </if>
33. <if test="studentId != null and studentId != '' ">
34. AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}
35. </if>
36. </trim>
37. </select>

<!-- 5.1 if/trim代替where(判断参数) - 将实体类不为空的属性作为where条件 -->

<select id="getStudentList\_if\_trim" resultMap="resultMap\_studentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

<trim prefix="WHERE" prefixOverrides="AND|OR">

<if test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')

</if>

<if test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</if>

<if test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</if>

<if test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</if>

<if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</if>

<if test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</if>

<if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</if>

<if test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</if>

</trim>

</select>

**3.5.2 trim代替set**

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 5.2 if/trim代替set(判断参数) - 将实体类不为空的属性更新 -->
2. <update id="updateStudent\_if\_trim" parameterType="liming.student.manager.data.model.StudentEntity">
3. UPDATE STUDENT\_TBL
4. <trim prefix="SET" suffixOverrides=",">
5. <if test="studentName != null and studentName != '' ">
6. STUDENT\_TBL.STUDENT\_NAME = #{studentName},
7. </if>
8. <if test="studentSex != null and studentSex != '' ">
9. STUDENT\_TBL.STUDENT\_SEX = #{studentSex},
10. </if>
11. <if test="studentBirthday != null ">
12. STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},
13. </if>
14. <if test="studentPhoto != null ">
15. STUDENT\_TBL.STUDENT\_PHOTO = #{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},
16. </if>
17. <if test="classId != '' ">
18. STUDENT\_TBL.CLASS\_ID = #{classId},
19. </if>
20. <if test="placeId != '' ">
21. STUDENT\_TBL.PLACE\_ID = #{placeId}
22. </if>
23. </trim>
24. WHERE STUDENT\_TBL.STUDENT\_ID = #{studentId}
25. </update>

<!-- 5.2 if/trim代替set(判断参数) - 将实体类不为空的属性更新 -->

<update id="updateStudent\_if\_trim" parameterType="liming.student.manager.data.model.StudentEntity">

UPDATE STUDENT\_TBL

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

<if test="studentName != null and studentName != '' ">

STUDENT\_TBL.STUDENT\_NAME = #{studentName},

</if>

<if test="studentSex != null and studentSex != '' ">

STUDENT\_TBL.STUDENT\_SEX = #{studentSex},

</if>

<if test="studentBirthday != null ">

STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},

</if>

<if test="studentPhoto != null ">

STUDENT\_TBL.STUDENT\_PHOTO = #{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},

</if>

<if test="classId != '' ">

STUDENT\_TBL.CLASS\_ID = #{classId},

</if>

<if test="placeId != '' ">

STUDENT\_TBL.PLACE\_ID = #{placeId}

</if>

</trim>

WHERE STUDENT\_TBL.STUDENT\_ID = #{studentId}

</update>

**3.6 choose (when, otherwise)**

有时候我们并不想应用所有的条件，而只是想从多个选项中选择一个。而使用if标签时，只要test中的表达式为true，就会执行if标签中的条件。MyBatis提供了choose 元素。if标签是与(and)的关系，而choose比傲天是或（or）的关系。

choose标签是按顺序判断其内部when标签中的test条件出否成立，如果有一个成立，则choose结束。当choose中所有when的条件都不满则时，则执行otherwise中的sql。类似于Java 的switch 语句，choose为switch，when为case，otherwise则为default。

例如下面例子，同样把所有可以限制的条件都写上，方面使用。choose会从上到下选择一个when标签的test为true的sql执行。安全考虑，我们使用where将choose包起来，放置关键字多于错误。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 6 choose(判断参数) - 按顺序将实体类第一个不为空的属性作为where条件 -->
2. <select id="getStudentList\_choose" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.STUDENT\_PHOTO,
8. ST.CLASS\_ID,
9. ST.PLACE\_ID
10. FROM STUDENT\_TBL ST
11. <where>
12. <choose>
13. <when test="studentName !=null ">
14. ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')
15. </when >
16. <when test="studentSex != null and studentSex != '' ">
17. AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}
18. </when >
19. <when test="studentBirthday != null ">
20. AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}
21. </when >
22. <when test="classId != null and classId!= '' ">
23. AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}
24. </when >
25. <when test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">
26. AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}
27. </when >
28. <when test="placeId != null and placeId != '' ">
29. AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}
30. </when >
31. <when test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">
32. AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}
33. </when >
34. <when test="studentId != null and studentId != '' ">
35. AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}
36. </when >
37. <otherwise>
38. </otherwise>
39. </choose>
40. </where>
41. </select>

<!-- 6 choose(判断参数) - 按顺序将实体类第一个不为空的属性作为where条件 -->

<select id="getStudentList\_choose" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

<where>

<choose>

<when test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')

</when >

<when test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</when >

<when test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</when >

<when test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</when >

<when test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</when >

<when test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</when >

<when test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</when >

<when test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</when >

<otherwise>

</otherwise>

</choose>

</where>

</select>

**3.7 foreach**

对于动态SQL 非常必须的，主是要迭代一个集合，通常是用于IN 条件。List 实例将使用“list”做为键，数组实例以“array” 做为键。

foreach元素是非常强大的，它允许你指定一个集合，声明集合项和索引变量，它们可以用在元素体内。它也允许你指定开放和关闭的字符串，在迭代之间放置分隔符。这个元素是很智能的，它不会偶然地附加多余的分隔符。

注意：你可以传递一个List实例或者数组作为参数对象传给MyBatis。当你这么做的时候，MyBatis会自动将它包装在一个Map中，用名称在作为键。List实例将会以“list”作为键，而数组实例将会以“array”作为键。

这个部分是对关于XML配置文件和XML映射文件的而讨论的。下一部分将详细讨论Java API，所以你可以得到你已经创建的最有效的映射。

**3.7.1参数为array示例的写法**

接口的方法声明：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public List<StudentEntity> getStudentListByClassIds\_foreach\_array(String[] classIds);

public List<StudentEntity> getStudentListByClassIds\_foreach\_array(String[] classIds);

动态SQL语句：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!— 7.1 foreach(循环array参数) - 作为where中in的条件 -->
2. <select id="getStudentListByClassIds\_foreach\_array" resultMap="resultMap\_studentEntity">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.STUDENT\_PHOTO,
8. ST.CLASS\_ID,
9. ST.PLACE\_ID
10. FROM STUDENT\_TBL ST
11. WHERE ST.CLASS\_ID IN
12. <foreach collection="array" item="classIds" open="(" separator="," close=")">
13. #{classIds}
14. </foreach>
15. </select>

<!— 7.1 foreach(循环array参数) - 作为where中in的条件 -->

<select id="getStudentListByClassIds\_foreach\_array" resultMap="resultMap\_studentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

WHERE ST.CLASS\_ID IN

<foreach collection="array" item="classIds" open="(" separator="," close=")">

#{classIds}

</foreach>

</select>

测试代码，查询学生中，在20000001、20000002这两个班级的学生：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. @Test
2. public void test7\_foreach() {
3. String[] classIds = { "20000001", "20000002" };
4. List<StudentEntity> list = this.dynamicSqlMapper.getStudentListByClassIds\_foreach\_array(classIds);
5. for (StudentEntity e : list) {
6. System.out.println(e.toString());
7. }
8. <p>}<span style="font-size: 14px; font-weight: bold; white-space: normal;"> </span></p>

@Test

public void test7\_foreach() {

String[] classIds = { "20000001", "20000002" };

List<StudentEntity> list = this.dynamicSqlMapper.getStudentListByClassIds\_foreach\_array(classIds);

for (StudentEntity e : list) {

System.out.println(e.toString());

}

}

**3.7.2参数为list示例的写法**

接口的方法声明：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public List<StudentEntity> getStudentListByClassIds\_foreach\_list(List<String> classIdList);

public List<StudentEntity> getStudentListByClassIds\_foreach\_list(List<String> classIdList);

动态SQL语句：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 7.2 foreach(循环List<String>参数) - 作为where中in的条件 -->
2. <select id="getStudentListByClassIds\_foreach\_list" resultMap="resultMap\_studentEntity">
3. SELECT ST.STUDENT\_ID,
4. ST.STUDENT\_NAME,
5. ST.STUDENT\_SEX,
6. ST.STUDENT\_BIRTHDAY,
7. ST.STUDENT\_PHOTO,
8. ST.CLASS\_ID,
9. ST.PLACE\_ID
10. FROM STUDENT\_TBL ST
11. WHERE ST.CLASS\_ID IN
12. <foreach collection="list" item="classIdList" open="(" separator="," close=")">
13. #{classIdList}
14. </foreach>
15. </select>

<!-- 7.2 foreach(循环List<String>参数) - 作为where中in的条件 -->

<select id="getStudentListByClassIds\_foreach\_list" resultMap="resultMap\_studentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

WHERE ST.CLASS\_ID IN

<foreach collection="list" item="classIdList" open="(" separator="," close=")">

#{classIdList}

</foreach>

</select>

测试代码，查询学生中，在20000001、20000002这两个班级的学生：

Java代码 [复制代码](http://limingnihao.iteye.com/blog/782190)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. @Test
2. public void test7\_2\_foreach() {
3. ArrayList<String> classIdList = new ArrayList<String>();
4. classIdList.add("20000001");
5. classIdList.add("20000002");
6. List<StudentEntity> list = this.dynamicSqlMapper.getStudentListByClassIds\_foreach\_list(classIdList);
7. for (StudentEntity e : list) {
8. System.out.println(e.toString());
9. }
10. }

**四、MyBatis主配置文件**

在定义sqlSessionFactory时需要指定MyBatis主配置文件：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
2. <property name="configLocation" value="classpath:mybatis-config.xml" />
3. <property name="dataSource" ref="dataSource" />
4. </bean>

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

<property name="configLocation" value="classpath:mybatis-config.xml" />

<property name="dataSource" ref="dataSource" />

</bean>

MyBatis配置文件中大标签configuration下子标签包括：

configuration

|--- properties

|--- settings

|--- typeAliases

|--- typeHandlers

|--- objectFactory

|--- plugins

|--- environments

|--- |--- environment

|--- |--- |--- transactionManager

|--- |--- |\_\_ dataSource

|\_\_ mappers

**4.1 properties属性**

properties和java的.properties的配置文件有关。配置properties的resource指定.properties的路径，然后再在properties标签下配置property的name和value，则可以替换.properties文件中相应属性值。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <!-- 属性替换 -->
2. <properties resource="mysql.properties">
3. <property name="jdbc.driverClassName" value="com.mysql.jdbc.Driver"/>
4. <property name="jdbc.url" value="jdbc:mysql://localhost:3306/student\_manager"/>
5. <property name="username" value="root"/>
6. <property name="password" value="limingnihao"/>
7. </properties>

<!-- 属性替换 -->

<properties resource="mysql.properties">

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

<property name="jdbc.url" value="jdbc:mysql://localhost:3306/student\_manager"/>

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

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

</properties>

**4.2 settings设置**

这是MyBatis 修改操作运行过程细节的重要的步骤。下方这个表格描述了这些设置项、含义和默认值。

|  |  |  |  |
| --- | --- | --- | --- |
| 设置项 | 描述 | 允许值 | 默认值 |
| cacheEnabled | 对在此配置文件下的所有cache 进行全局性开/关设置。 | true | false | true |
| lazyLoadingEnabled | 全局性设置懒加载。如果设为‘false’，则所有相关联的都会被初始化加载。 | true | false | true |
| aggressiveLazyLoading | 当设置为‘true’的时候，懒加载的对象可能被任何懒属性全部加载。否则，每个属性都按需加载。 | true | false | true |
| multipleResultSetsEnabled | 允许和不允许单条语句返回多个数据集（取决于驱动需求） | true | false | true |
| useColumnLabel | 使用列标签代替列名称。不同的驱动器有不同的作法。参考一下驱动器文档，或者用这两个不同的选项进行测试一下。 | true | false | true |
| useGeneratedKeys | 允许JDBC 生成主键。需要驱动器支持。如果设为了true，这个设置将强制使用被生成的主键，有一些驱动器不兼容不过仍然可以执行。 | true | false | false |
| autoMappingBehavior | 指定MyBatis 是否并且如何来自动映射数据表字段与对象的属性。PARTIAL将只自动映射简单的，没有嵌套的结果。FULL 将自动映射所有复杂的结果。 | NONE,  PARTIAL,  FULL | PARTIAL |
| defaultExecutorType | 配置和设定执行器，SIMPLE 执行器执行其它语句。REUSE 执行器可能重复使用prepared statements 语句，BATCH执行器可以重复执行语句和批量更新。 | SIMPLE  REUSE  BATCH | SIMPLE |
| defaultStatementTimeout | 设置一个时限，以决定让驱动器等待数据库回应的多长时间为超时 | 正整数 | Not Set  (null) |

例如：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <settings>
2. <setting name="cacheEnabled" value="true" />
3. <setting name="lazyLoadingEnabled" value="true" />
4. <setting name="multipleResultSetsEnabled" value="true" />
5. <setting name="useColumnLabel" value="true" />
6. <setting name="useGeneratedKeys" value="false" />
7. <setting name="enhancementEnabled" value="false" />
8. <setting name="defaultExecutorType" value="SIMPLE" />
9. </settings>

<settings>

<setting name="cacheEnabled" value="true" />

<setting name="lazyLoadingEnabled" value="true" />

<setting name="multipleResultSetsEnabled" value="true" />

<setting name="useColumnLabel" value="true" />

<setting name="useGeneratedKeys" value="false" />

<setting name="enhancementEnabled" value="false" />

<setting name="defaultExecutorType" value="SIMPLE" />

</settings>

**4.3 typeAliases类型别名**

类型别名是Java 类型的简称。

它仅仅只是关联到XML 配置，简写冗长的JAVA 类名。例如：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <typeAliases>
2. <typeAlias alias="UserEntity" type="com.manager.data.model.UserEntity" />
3. <typeAlias alias="StudentEntity" type="com.manager.data.model.StudentEntity" />
4. <typeAlias alias="ClassEntity" type="com.manager.data.model.ClassEntity" />
5. </typeAliases>

<typeAliases>

<typeAlias alias="UserEntity" type="com.manager.data.model.UserEntity" />

<typeAlias alias="StudentEntity" type="com.manager.data.model.StudentEntity" />

<typeAlias alias="ClassEntity" type="com.manager.data.model.ClassEntity" />

</typeAliases>

使用这个配置，“StudentEntity”就能在任何地方代替“com.manager.data.model.StudentEntity”被使用。

对于普通的Java类型，有许多内建的类型别名。它们都是大小写不敏感的，由于重载的名字，要注意原生类型的特殊处理。

|  |  |
| --- | --- |
| 别名 | 映射的类型 |
| \_byte | byte |
| \_long | long |
| \_short | short |
| \_int | int |
| \_integer | int |
| \_double | double |
| \_float | float |
| \_boolean | boolean |
| string | String |
| byte | Byte |
| long | Long |
| short | Short |
| int | Integer |
| integer | Integer |
| double | Double |
| float | Float |
| boolean | Boolean |
| date | Date |
| decimal | BigDecimal |
| bigdecimal | BigDecimal |
| object | Object |
| map | Map |
| hashmap | HashMap |
| list | List |
| arraylist | ArrayList |
| collection | Collection |
| iterator | Iterator |

**4.4 typeHandlers类型句柄**

无论是MyBatis在预处理语句中设置一个参数，还是从结果集中取出一个值时，类型处理器被用来将获取的值以合适的方式转换成Java类型。下面这个表格描述了默认的类型处理器。

|  |  |  |
| --- | --- | --- |
| 类型处理器 | Java类型 | JDBC类型 |
| BooleanTypeHandler | Boolean，boolean | 任何兼容的布尔值 |
| ByteTypeHandler | Byte，byte | 任何兼容的数字或字节类型 |
| ShortTypeHandler | Short，short | 任何兼容的数字或短整型 |
| IntegerTypeHandler | Integer，int | 任何兼容的数字和整型 |
| LongTypeHandler | Long，long | 任何兼容的数字或长整型 |
| FloatTypeHandler | Float，float | 任何兼容的数字或单精度浮点型 |
| DoubleTypeHandler | Double，double | 任何兼容的数字或双精度浮点型 |
| BigDecimalTypeHandler | BigDecimal | 任何兼容的数字或十进制小数类型 |
| StringTypeHandler | String | CHAR和VARCHAR类型 |
| ClobTypeHandler | String | CLOB和LONGVARCHAR类型 |
| NStringTypeHandler | String | NVARCHAR和NCHAR类型 |
| NClobTypeHandler | String | NCLOB类型 |
| ByteArrayTypeHandler | byte[] | 任何兼容的字节流类型 |
| BlobTypeHandler | byte[] | BLOB和LONGVARBINARY类型 |
| DateTypeHandler | Date（java.util） | TIMESTAMP类型 |
| DateOnlyTypeHandler | Date（java.util） | DATE类型 |
| TimeOnlyTypeHandler | Date（java.util） | TIME类型 |
| SqlTimestampTypeHandler | Timestamp（java.sql） | TIMESTAMP类型 |
| SqlDateTypeHandler | Date（java.sql） | DATE类型 |
| SqlTimeTypeHandler | Time（java.sql） | TIME类型 |
| ObjectTypeHandler | Any | 其他或未指定类型 |
| EnumTypeHandler | Enumeration类型 | VARCHAR-任何兼容的字符串类型，作为代码存储（而不是索引）。 |

你可以重写类型处理器或创建你自己的类型处理器来处理不支持的或非标准的类型。要这样做的话，简单实现TypeHandler接口（org.mybatis.type），然后映射新的类型处理器类到Java类型，还有可选的一个JDBC类型。然后再typeHandlers中添加这个类型处理器。

新定义的类型处理器将会覆盖已经存在的处理Java的String类型属性和VARCHAR参数及结果的类型处理器。要注意MyBatis不会审视数据库元信息来决定使用哪种类型，所以你必须在参数和结果映射中指定那是VARCHAR类型的字段，来绑定到正确的类型处理器上。这是因为MyBatis直到语句被执行都不知道数据类型的这个现实导致的。

Java代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public class LimingStringTypeHandler implements TypeHandler {
2. @Override
3. public void setParameter(PreparedStatement ps, int i, Object parameter, JdbcType jdbcType) throws SQLException {
4. System.out.println("setParameter - parameter: " + ((String) parameter) + ", jdbcType: " + jdbcType.TYPE\_CODE);
5. ps.setString(i, ((String) parameter));
6. }
7. @Override
8. public Object getResult(ResultSet rs, String columnName) throws SQLException {
9. System.out.println("getResult - columnName: " + columnName);
10. return rs.getString(columnName);
11. }
12. @Override
13. public Object getResult(CallableStatement cs, int columnIndex) throws SQLException {
14. System.out.println("getResult - columnIndex: " + columnIndex);
15. return cs.getString(columnIndex);
16. }
17. }

public class LimingStringTypeHandler implements TypeHandler {

@Override

public void setParameter(PreparedStatement ps, int i, Object parameter, JdbcType jdbcType) throws SQLException {

System.out.println("setParameter - parameter: " + ((String) parameter) + ", jdbcType: " + jdbcType.TYPE\_CODE);

ps.setString(i, ((String) parameter));

}

@Override

public Object getResult(ResultSet rs, String columnName) throws SQLException {

System.out.println("getResult - columnName: " + columnName);

return rs.getString(columnName);

}

@Override

public Object getResult(CallableStatement cs, int columnIndex) throws SQLException {

System.out.println("getResult - columnIndex: " + columnIndex);

return cs.getString(columnIndex);

}

}

在配置文件的typeHandlers中添加typeHandler标签。

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <typeHandlers>
2. <typeHandler javaType="String" jdbcType="VARCHAR" handler="liming.student.manager.type.LimingStringTypeHandler"/>
3. </typeHandlers>

<typeHandlers>

<typeHandler javaType="String" jdbcType="VARCHAR" handler="liming.student.manager.type.LimingStringTypeHandler"/>

</typeHandlers>

**4.5 ObjectFactory对象工厂**

每次MyBatis 为结果对象创建一个新实例，都会用到ObjectFactory。默认的ObjectFactory 与使用目标类的构造函数创建一个实例毫无区别，如果有已经映射的参数，那也可能使用带参数的构造函数。

如果你重写ObjectFactory 的默认操作，你可以通过继承org.apache.ibatis.reflection.factory.DefaultObjectFactory创建一下你自己的。

ObjectFactory接口很简单。它包含两个创建用的方法，一个是处理默认构造方法的，另外一个是处理带参数构造方法的。最终，setProperties方法可以被用来配置ObjectFactory。在初始化你的ObjectFactory实例后，objectFactory元素体中定义的属性会被传递给setProperties方法。

Java代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. public class LimingObjectFactory extends DefaultObjectFactory {
2. private static final long serialVersionUID = -399284318168302833L;
3. @Override
4. public Object create(Class type) {
5. return super.create(type);
6. }
7. @Override
8. public Object create(Class type, List<Class> constructorArgTypes, List<Object> constructorArgs) {
9. System.out.println("create - type: " + type.toString());
10. return super.create(type, constructorArgTypes, constructorArgs);
11. }
12. @Override
13. public void setProperties(Properties properties) {
14. System.out.println("setProperties - properties: " + properties.toString() + ", someProperty: " + properties.getProperty("someProperty"));
15. super.setProperties(properties);
16. }
17. }

public class LimingObjectFactory extends DefaultObjectFactory {

private static final long serialVersionUID = -399284318168302833L;

@Override

public Object create(Class type) {

return super.create(type);

}

@Override

public Object create(Class type, List<Class> constructorArgTypes, List<Object> constructorArgs) {

System.out.println("create - type: " + type.toString());

return super.create(type, constructorArgTypes, constructorArgs);

}

@Override

public void setProperties(Properties properties) {

System.out.println("setProperties - properties: " + properties.toString() + ", someProperty: " + properties.getProperty("someProperty"));

super.setProperties(properties);

}

}

配置文件中添加objectFactory标签

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <objectFactory type="liming.student.manager.configuration.LimingObjectFactory">
2. <property name="someProperty" value="100"/>
3. </objectFactory>

<objectFactory type="liming.student.manager.configuration.LimingObjectFactory">

<property name="someProperty" value="100"/>

</objectFactory>

**4.6 plugins插件**

MyBatis允许你在某一点拦截已映射语句执行的调用。默认情况下，MyBatis允许使用插件来拦截方法调用：

* Executor(update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
* ParameterHandler(getParameterObject, setParameters)
* ResultSetHandler(handleResultSets, handleOutputParameters)
* StatementHandler(prepare, parameterize, batch, update, query)

这些类中方法的详情可以通过查看每个方法的签名来发现，而且它们的源代码在MyBatis的发行包中有。你应该理解你覆盖方法的行为，假设你所做的要比监视调用要多。如果你尝试修改或覆盖一个给定的方法，你可能会打破MyBatis的核心。这是低层次的类和方法，要谨慎使用插件。

使用插件是它们提供的非常简单的力量。简单实现拦截器接口，要确定你想拦截的指定签名。

**4.7 environments环境**

MyBatis 可以配置多个环境。这可以帮助你SQL 映射对应多种数据库等。

**4.8 mappers映射器**

这里是告诉MyBatis 去哪寻找映射SQL 的语句。可以使用类路径中的资源引用，或者使用字符，输入确切的URL 引用。

例如：

Xml代码 [复制代码](http://limingnihao.iteye.com/blog/1060764)[[收藏代码http://limingnihao.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <mappers>
2. <mapper resource="com/manager/data/maps/UserMapper.xml" />
3. <mapper resource="com/manager/data/maps/StudentMapper.xml" />
4. <mapper resource="com/manager/data/maps/ClassMapper.xml" />
5. </mappers>

<mappers>

<mapper resource="com/manager/data/maps/UserMapper.xml" />

<mapper resource="com/manager/data/maps/StudentMapper.xml" />

<mapper resource="com/manager/data/maps/ClassMapper.xml" />

</mappers>

@Test

public void test7\_2\_foreach() {

ArrayList<String> classIdList = new ArrayList<String>();

classIdList.add("20000001");

classIdList.add("20000002");

List<StudentEntity> list = this.dynamicSqlMapper.getStudentListByClassIds\_foreach\_list(classIdList);

for (StudentEntity e : list) {

System.out.println(e.toString());

}

}

<resultMap type="liming.student.manager.data.model.StudentEntity" id="resultMap\_studentEntity\_discriminator">

<id property="studentId" column="STUDENT\_ID" javaType="String" jdbcType="VARCHAR"/>

<result property="studentName" column="STUDENT\_NAME" javaType="String" jdbcType="VARCHAR"/>

<result property="studentSex" column="STUDENT\_SEX" javaType="int" jdbcType="INTEGER"/>

<result property="studentBirthday" column="STUDENT\_BIRTHDAY" javaType="Date" jdbcType="DATE"/>

<result property="studentPhoto" column="STUDENT\_PHOTO" javaType="byte[]" jdbcType="BLOB" typeHandler="org.apache.ibatis.type.BlobTypeHandler" />

<result property="placeId" column="PLACE\_ID" javaType="String" jdbcType="VARCHAR"/>

<discriminator column="CLASS\_ID" javaType="String" jdbcType="VARCHAR">

<case value="20000001" resultType="liming.student.manager.data.model.StudentEntity" >

<result property="classId" column="CLASS\_ID" javaType="String" jdbcType="VARCHAR"/>

</case>

</discriminator>

</resultMap>

使用Junit测试：

Java代码

@RunWith(value = SpringJUnit4ClassRunner.class)

@ContextConfiguration(value = "test-servlet.xml")

public class StudentMapperTest {

@Autowired

private ClassMapper classMapper;

@Autowired

private StudentMapper studentMapper;

@Transactional

public void getStudentTest(){

StudentEntity entity = studentMapper.getStudent("10000013");

System.out.println("" + entity.getStudentID() + entity.getStudentName());

List<StudentEntity> studentList = studentMapper.getStudentAll();

for( StudentEntity entityTemp : studentList){

System.out.println(entityTemp.getStudentName());

}

}

}