代码地址

<https://github.com/lilewa/mybatisChicken/tree/master/xmlMapper>

本篇讲述使用xml做简单的映射，包括select，insert，update，delete等，以及调用存储过程

**SqlSessionFactory**

每个基于 MyBatis 的应用都是以一个 SqlSessionFactory 的实例为核心的。SqlSessionFactory 的实例可以通过 SqlSessionFactoryBuilder 获得。而 SqlSessionFactoryBuilder 则可以从 XML 配置文件或java代码配置的一个 Configuration 的实例构建出 SqlSessionFactory 的实例。

**1．XML 文件构建 SqlSessionFactory 实例**

建议使用类路径下的资源文件进行配置。 但是也可以使用任意的输入流（InputStream）实例，MyBatis 包含一个名叫 Resources 的工具类，它包含一些实用方法，可使从 classpath 或其他位置加载资源文件更加容易。

String resource = "org/mybatis/example/mybatis-config.xml";

InputStream inputStream = Resources.getResourceAsStream(resource);

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

**2．Java代码构建 SqlSessionFactory实例**

MyBatis 也提供了完整的配置类，提供所有和 XML 文件相同功能的配置项。

DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();

TransactionFactory transactionFactory = new JdbcTransactionFactory();

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

Configuration configuration = new Configuration(environment);

configuration.addMapper(BlogMapper.class);

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(configuration);

**SqlSession**

有了 SqlSessionFactory，我们就可以从中获得 SqlSession 的实例了。SqlSession 完全包含了面向数据库执行 SQL 命令所需的所有方法。可以通过 SqlSession 实例来直接执行已映射的 SQL 语句。

**使用接口关联mapper文件**

Mapper可以使用 XML 配置，也可以使用 Java 注解配置。使用注解来映射简单语句会使代码显得更加简洁，然而对于稍微复杂一点的语句，Java 注解就力不从心了，并且会显得更加混乱。由于 Java 注解的一些限制以及某些 MyBatis 映射的复杂性，要使用大多数高级映射（比如：嵌套联合映射），仍然需要使用 XML 配置

MyBatis 3.0 相比2.0 版本的一个最大变化，就是支持使用接口来调用方法。 以前使用SqlSession 通过命名空间调用MyBatis 方法时，首先需要用到命名空间和方法id 组成的字符串来调用相应的方法。当参数多于1 个的时候，需要将所有参数放到一个Map对象中。通过Map 传递多个参数

使用接口调用方式就会方便很多，它不依赖于字符串字面值，会更安全一点；其次，如果你的 IDE 有代码补全功能，那么代码补全可以帮你快速选择已映射的 SQL 语句。

在Spring 中使用的时候，可以配置为自动扫描所有的接口类，直接将接口注入需要用到的地方

MyBatis 使用Java 的动态代理可以直接通过接口来调用相应的方法，不需要提供接口的实现类。

上一篇Country查询的示例

List<Country> countryList = sqlSession.selectList( "noInterface.mapper.CountryMapper.selectAll" );

改为接口调用的方式查询可以写成

CountryMapper countrMapper = sqlSession.getMapper(CountryMapper.class);

countrMapper.selectAll();

下面演示配置接口的调用方式使用mybatis

**1 准备数据库**

**1.1添加名为mybatis的数据库**

CREATE DATABASE mybatis DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;

**1.2添加table**

CREATE TABLE `sys\_user` (  
 `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '用户ID',  
 `user\_name` varchar(50) DEFAULT NULL COMMENT '用户名',  
 `user\_password` varchar(50) DEFAULT NULL COMMENT '密码',  
 `user\_email` varchar(50) DEFAULT 'test@mybatis.tk' COMMENT '邮箱',  
 `user\_info` text COMMENT '简介',  
 `head\_img` blob COMMENT '头像',  
 `create\_time` datetime DEFAULT NULL COMMENT '创建时间',  
 `country\_id` int(11) DEFAULT NULL COMMENT '国籍',  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=1035 DEFAULT CHARSET=utf8 COMMENT='用户表';

**2 创建实体类**

在src/main/java 下面创建包mybatisChicken.xmlMapper.model，在包下创建实体类SysUser

public class SysUser {  
 private Long id;  
 private String userName;  
 private String userPassword;  
 private String userEmail;  
 private String userInfo;  
 private byte[] headImg;  
 private Date createTime;  
}

**3 创建接口类**

在src/main/java 下面创建包mybatisChicken.xmlMapper.mapper，在该包下创建接口类UserMapper.java

**3.1添加接口方法**

在UserMapper 接口中添加一个selectByid 方法，此方法根据用户id查询用户信息

public interface UserMapper {

SysUser selectById(Long id);

}

**4 配置UserMapper接口对应的mapper文件**

在src/main/resources下新建路径mybatisChicken/xmlMapper/mapper。在路径下新建mapper文件UserMapper.xml

**4.1 将这个mapper文件添加到config文件的mappers元素中**

<mappers>  
 <mapper resource="com/mybatisChicken /mapper/UserMapper.xml"/>  
</mappers>

**4.2设置mapper文件的namespace**

<mapper namespace="mybatisChicken.xmlMapper.mapper.UserMapper">

需要注意的是<mapper>元素的 namespace特性。当Mapper接口和XML文件关联的时候, namespace特性的值就需要配置成接口的完全限定名称,例如 UserMapper接口对应的mybatisChicken.xmlMapper.mapper.UserMapper, MyBatis内部就是通过这个值将接口和XML关联起来的

**4.3 增加查询sql语句**

使用<select>元素配置查询语句。Id特性对应接口的方法名selectById。ResultMap特性指定查询结果的映射规则

<select id="selectById" resultMap="userMap">  
 select \* from sys\_user where id = #{id}  
</select>

接口和mapper文件是通过将namespace 的值设置为接口的全限定名称来进行关联的，mapper文件中的select 标签的id 特性值和定义的接口方法名同名。MyBatis就是通过这种方式将接口方法和XML中定义的SQL语句关联到一起的。

**4.4使用<resultMap>元素配置查询结果映射规则**

id特性用来被其他元素引用，type特性指定查询列映射到的java对象类型。

<resultMap id="userMap" type="com.lile.mybatisExer.model.SysUser">  
 <id property="id" column="id"/>  
 <result property="userName" column="user\_name"/>  
 <result property="userPassword" column="user\_password"/>  
 <result property="userEmail" column="user\_email"/>  
 <result property="userInfo" column="user\_info"/>  
 <result property="headImg" column="head\_img" jdbcType="BLOB"/>  
 <result property="createTime" column="create\_time" jdbcType="TIMESTAMP"/>  
</resultMap>

Result元素定义列和字段的映射关系，它包含的特性

Column特性： 从数据库中得到的列名， 或者是列的别名。

property ：映射到对象的字段名。可以映射简单的如“ username ”这样的特性，也可以映射到对象字段的字段， 例如“ address.street.number ”

Id元素提供和Result元素一样的功能，它还能标识主键列，这会在比较对象实例时用到。id元素可以帮助提高整体性能。

**5 增加测试类方法**

在src/test/java中创建包mybatisChicken.xmlMapper，在这个包下创建测试类UserMapperTest。在测试方法中调用接口方法selectById

//获取 UserMapper 接口  
UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  
//调用 selectById 方法，查询 id = 1 的用户  
SysUser user = userMapper.selectById(1l);

mybatis还支持自动映射，也就是说<select>元素不必指定它的resultMap特性，只需使用resultType特性告诉mybatis要映射的java对象类型。这里的resultType和<resultMap> 元素的type特性都是用来指定java对象类型。使用自动映射时要保证列名或列别名和对象字段名一样

<select id="selectAll" resultType="mybatisChicken.xmlMapper.model.SysUser">  
 select id,  
 user\_name userName,  
 user\_password userPassword,  
 user\_email userEmail,  
 user\_info userInfo,  
 head\_img headImg,  
 create\_time createTime  
 from sys\_user  
</select>

另外，<resultMap> 元素有一个autoMapping特性，可选值true或false。用于指定未在<resultMap> 元素中使用<result>或<id>指定映射关系的列，是否自动映射到对象字段上。此特性可以用于手动指定查询中的复杂嵌套关系后自动映射简单映射关系。

**insert 用法**

UserMapper接口中添加方法

int insert(SysUser sysUser);

方法返回值是执行的SQL 影响的行数，

UserMapper,xml中增加inser语句

<insert id="insert">  
 insert into sys\_user(  
 user\_name, user\_password, user\_email,  
 user\_info, head\_img, create\_time)  
 values(  
 #{userName}, #{userPassword}, #{userEmail},  
 #{userInfo}, #{headImg, jdbcType=BLOB}, #{createTime, jdbcType=DATE})  
</insert>

参数headImg和createTime 使用了jdbcType属性。为了防止类型错误，对于一些特殊的数据类型，建议指定具体的jdbcType 值

由于数据库区分date 、time 、datetime 类型，date 、time 、datetime 对应的JDBC 类型分别为DATE 、TIME 、TIMESTAMP 。但是Java 中一般都使用java.util.Date类型。因此为了保证数据类型的正确，需要手动指定日期类型

#{createTime, jdbcType=TIMESTAMP} 插入的create\_time 的格式是 2019-05-08 23:41:51.602(Timestamp)

#{createTime, jdbcType=DATE} 插入的create\_time 的格式是 2019-05-08(Date)

JDBC 要求，如果一个列允许 null 值，并且可能会传递值 null 的参数，在执行插入、更新和删除时就必须要指定 JDBCType

当insert需要返回主键值时，由于不同数据库的主键生成方式不同， <insert>元素有不同的配置方法

**支持自增主键的数据库**

在UserMapper 接口中增加方法。

int insert2(SysUser sysUser);

在XML 中增加insert2方法对应的sql。

<insert id="insert2" useGeneratedKeys="true" keyProperty="id">  
 insert into sys\_user(  
 user\_name, user\_password, user\_email,  
 user\_info, head\_img, create\_time)  
 values(  
 #{userName}, #{userPassword}, #{userEmail},  
 #{userInfo}, #{headImg, jdbcType=BLOB}, #{createTime, jdbcType=TIMESTAMP})  
</insert>

Insert元素配置了useGeneratedKeys和keyProperty特性

useGeneratedKeys 设置为true 后， MyBatis 会使用JDBC 的getGeneratedKeys 方法来取出由数据库内部生成的主键，获得主键值后将其赋值给keyProperty 指定的id字段，id是对象的字段名。

<insert id="insert2" useGeneratedKeys="true" keyProperty="id">

当主键由多列组成时，还需设置keyColumn 特性，特性值使用逗号隔开不同的列名。keyProperty 特性设置成按逗号隔开的对象字段名，和keyColumn的列一一对应（待定）

有些数据库（如Oracle）不提供主键自增的功能，而是使用序列得到一个值，然后将这个值赋给对象字段，再将数据插入数据库,mybatis使用<selectKey>元素来获取主键的值。这种方式不仅适用于不提供主键自增功能的数据库，也适用于提供主键自增功能的数据库。

增加接口方法

int insert3(SysUser sysUser);

增加insert3方法对应的sql。

<insert id="insert3">  
 insert into sys\_user(  
 user\_name, user\_password, user\_email,  
 user\_info, head\_img, create\_time)  
 values(  
 #{userName}, #{userPassword}, #{userEmail},  
 #{userInfo}, #{headImg, jdbcType=BLOB}, #{createTime, jdbcType=TIMESTAMP})  
 <selectKey keyColumn="id" resultType="long" keyProperty="id" order="AFTER">  
 SELECT LAST\_INSERT\_ID()  
 </selectKey>  
</insert>

selectKey 标签的keyColumn 、keyProperty 和上面useGeneratedKeys 的用法含义相同，这里的resultType 用于设置返回值类型（可以不用设置MyBatis 通常可以推断出来）。order 属性的设置和使用的数据库有关。在MySQL 数据库中， order 属性设置的值是AFTER ，因为当前记录的主键值在insert 语句执行成功后才能获取到。而在Oracle 数据库中， order 的值要设置为BEFORE ，这是因为Oracle中需要先从序列获取值，然后将值作为主键插入到数据库中。

<!-- Oracle 的例子，查询多个列的时候需要 keyColumn -->  
<insert id="insertOracle">  
 <selectKey keyColumn="id" resultType="long" keyProperty="id" order="BEFORE">  
 SELECT SEQ\_USER.nextval from dual  
 </selectKey>  
 insert into sys\_user(  
 id, user\_name, user\_password, user\_email,   
 user\_info, head\_img, create\_time)  
 values(  
 #{id}, #{userName}, #{userPassword}, #{userEmail},   
 #{userInfo}, #{headImg, jdbcType=BLOB}, #{createTime, jdbcType=TIMESTAMP})  
</insert>

Oracle 方式的insert 语句中明确写出了id列和值#｛id｝，因为执行selectKey 中的语句后id 就有值了，我们需要把这个序列值作为主键值入到数据库中，所以必须指定id列，如果不指定这一列，数据库就会因为主键不能为空而抛出异常

更新在mapper文件中使用<update>元素

删除在mapper文件中使用<delete>元素。他们的用法和insert大致相同

**存储过程**

在存储过程中使用参数时， 除了写上必要的属性名，还必须指定参数的mode （模式），可

选值为IN 、OUT 、INOUT 三种，OUT 模式的参数必须指定jdbcType 。这是因为在IN 模式下， MyBatis 提供了默认的jdbcType ，在OUT 模式下没有提供. 另外在使用Oracle 数据库时，如果入参存在null 的情况， 那么In模式参数也必须指定jdbcType

添加存储过程

**CREATE PROCEDURE** `select\_user\_by\_id`

**CREATE PROCEDURE** `select\_user\_page`(

新增接口方法

void selectUserById(SysUser user);

增加接口方法对应的sql

<select id="selectUserById" statementType="CALLABLE" useCache="false">  
 {call select\_user\_by\_id(  
 #{id, mode=IN},  
 #{userName, mode=OUT, jdbcType=VARCHAR},  
 #{userPassword, mode=OUT, jdbcType=VARCHAR},  
 #{userEmail, mode=OUT, jdbcType=VARCHAR},  
 #{userInfo, mode=OUT, jdbcType=VARCHAR},  
 #{headImg, mode=OUT, jdbcType=BLOB, javaType=\_byte[]},  
 #{createTime, mode=OUT, jdbcType=TIMESTAMP}  
 )}  
</select>

接口方法返回类型是void，表示调用存储过程没有返回值。存储过程含有OUT模式的参数，表示调用存储过程会对传入的参数造成影响。调用存储过程可以传入类实体或Map类型参数。存储过程对参数的影响会反应到传入的实体或map上。

在测试类中使用实体类对象作为参数调用存储过程

userMapper.selectUserById(user);

查看存储过程中设置的userName

System.out.println("用户名：" + user.getUserName());

新增接口方法实现分页查询

List<SysUser> selectUserPage(Map<String, Object> params);

增加接口方法对应的sql

<select id="selectUserPage" statementType="CALLABLE" useCache="false" resultMap="userMap">  
 {call select\_user\_page(  
 #{userName, mode=IN},  
 #{offset, mode=IN},  
 #{limit, mode=IN},  
 #{total, mode=OUT, jdbcType=BIGINT}  
 )}  
</select>

存储过程selectUserPage的参数中除了userName 属性在SysUser实体类中，其他3个参数都和SysUser无关，因此为了使用SysUser 而加3个字段也是可以的。更简单的方法是使用map类型的参数

存储过程selectUserPage 中设置了查询结果的记录数total，在最后使用 select \* from sys\_user 语句返回查询结果。因此接口方法使用List<SysUser>作为返回类型。返回的用户信息使用resultMap="userMap"映射到List<SysUser>上。Total是个OUT模式的参数，在传入存储过程的map中获取它的值

在测试类中调用存储过程

Map<String, Object> params = new HashMap<String, Object>();  
params.put("userName", "ad");  
params.put("offset", 0);  
params.put("limit", 10);  
List<SysUser> userList = userMapper.selectUserPage(params);  
Long total = (Long) params.get("total");

使用OUT模式参数时，通常情况下会使用对象中的属性接收出参的值，或者使用Map 类型接收返回值。当使用JavaBean 对象接收出参时，必须保证所有出参在JavaBean 中都有对应的字段存在

**其他**

Java中方法可以重载，那么可以存在相同接口名而参数不同的接口方法。但是mapper文件中的id必须唯一。可能存在如下情况，两个接口方法都使用mapper文件中同一sql语句。

例如接口方法：

int deleteById(Long id);  
int deleteById(SysUser sysUser);

mapper文件：

<delete id="deleteById">/\*参数是id或实体的重载方法都能调用\*/  
 delete from sys\_user where id = #{id}  
</delete>

如果config中要添加很多mapper文件，有一个简化添加mapper文件的方法。使用包名的方式添加mapper文件

<mappers>  
 <package name="mybatisChicken.xmlMapper.mapper"/>  
</mappers>

这种配置方式会先查找mybatisChicken.xmlMapper.mapper包下所有的接口。将接口完全限定名转换为路径,查找该路径下的对应的XML映射文件,例如,将接口mybatisChicken.xmlMapper.mapper.UserMapper转换为 mybatisChicken/xmlMapper/mapper/UserMapper.xml 以xml为后缀搜索XML资源,如果找到就解析XML。

因此，使用接口调用，并使用包名搜索mapper文件时，对mapper文件放置的路径和文件名是有要求的。路径名称+文件名要和接口完全限定名一致，且mapper文件的namespce的值也要和接口完全限定名一致。

Mapper文件路径名==接口完全限定名（使用包名搜索mapper文件要求）

Mapper文件namespace==接口完全限定名（接口调用方式要求）

Log4j包名称== Mapper文件namespace（配置日志输出要求）

Country查询的示例中mapper文件的路径和mapper文件中指定的namespace是无关的。

<mappers>  
 <mapper resource="mapper/CountryMapper.xml"/>  
</mappers>

**<resultMap> 元素中配置的要映射的列，在sql查询中不存在时，不存在的列会被忽略**

<select id="selectMapBiggerCol" resultMap="userMap">  
 select id from sys\_user where id = #{id}  
</select>

<resultMap id="userMap" type="mybatisChicken.xmlMapper.model.SysUser">  
 <id property="id" column="id"/>  
 <result property="userName" column="user\_name"/>  
 <result property="userPassword" column="user\_password"/>  
 <result property="userEmail" column="user\_email"/>  
 <result property="userInfo" column="user\_info"/>  
 <result property="headImg" column="head\_img" jdbcType="BLOB"/>  
 <result property="createTime" column="create\_time" jdbcType="TIMESTAMP"/>  
</resultMap>

**sql查询中的列，在<resultMap> 元素中没有配置映射时，列会被忽略。**

<select id="selectColBiggerMap" resultMap="userMapidname">  
 select \* from sys\_user where id = #{id}  
</select>  
<resultMap id="userMapidname" type="mybatisChicken.xmlMapper.model.SysUser">  
 <id property="id" column="id"/>  
 <result property="userName" column="user\_name"/>  
</resultMap>

引用

<http://www.mybatis.org/mybatis-3/zh/index.html>

<https://github.com/mybatis-book/book>