// 在公司不让看视频。。。。

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

Mybatis是支持定制化SQL、存储过程以及高级映射的优秀持久层框架。它避免了几乎所有的JDBC代码和手动设置参数（什么叫手动设置参数？）以及获取结果集。这段话是mybatis官网上的，当然在我眼里属于什么都没讲的\_(:з」∠)\_

总之就是个高级JDBC，那JDBC我是怎么用的？

第一步是将MySQL实现的JDBC的jar包添加至classpath。

第二步连接数据库，先确定什么类型的数据库，然后IP地址、用户名、及密码。

第三步事务，setAutoCommit()、commit()、rollback()。

第四步增删查改，PreparedStatement、sql、设置参数。

第五步提取结果，看情况过滤结果集。

第六步释放资源，这个千万不能忘啊！

对没错，就这么多，总结起来就是为了存取数据，mybatis也一样，只是方式不同罢了。接下来，唉，大哥你是怎么连接数据库的呢？

// 以下内容来自<http://www.cnblogs.com/jeffen/p/6248999.html>，还是有人跟我想法一致的嘛。

采用数据库连接池，但连接统一从DataSource里获取，高度解耦因此具体实现不用太在意，安心调用接口。

SQL统一存取，就不像JDBC那样把SQL语句放在DAO类里了。我猜是以键值对形式放在某 .property里。要说好处在哪，由于统一存取，修改SQL语句并不会造成代码的重新编译，而且方便管理，其它的暂时理解不了。

统一存取后会出现一个问题，有一类SQL语句是根据用户传入的参数来确定的，而现在Java代码已经不负责拼凑SQL语句了，要怎么弄？慢慢看小峰的视频吧。一共13讲，还好没有上下，今天晚上快速6讲，最好能全看完。

取结果方式改变。

重复SQL语句的优化，有些语句是很相似的，select \* from t\_a / select \* from t\_b。使用模块化来解决重复问题。

Hello world 结束了，从最简单的增加开始，主要内容有SqlSessionFactory及相应的xml、SqlSesssion、Mapper及相应的xml。

***mybatis-config.xml***

先从SqlSessionFactory开始。

***“每个基于 MyBatis 的应用都是以一个 SqlSessionFactory 的实例为中心的。--官方文档”***

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

InputStream inputStream = Resources.*getResourceAsStream*(resource);

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

获取SqlSessionFactory的代码，推荐将 mybatis-config.xml 放置在源目录下。另外要注意的是获取流的方式，Resources.getResourceAsStream(resource); ，而不是通过FileInputStream，Resources类是Mybatis提供的。最后由 SqlSessionFactoryBuilder 类创建。下面看与之相关的 mybatis-config.xml 文件，这个他妈的才是重头戏啊。

不管是持久层框架还是JDBC，归根结底都是对数据库进行操作，既然如此首先要明确连接的是什么类型的数据库，以及它的URL、用户名、还有密码。因此在开始mybatis-config.xml 前，先准备一个 jdbc.properties ，包含以下4个属性并与mybatis-config.xml 同一目录。

jdbc.driverClassName=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/db\_mybatis

jdbc.username=root

jdbc.password=123456

省的烦，我就直接拷一个xml过来了。

<configuration>

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

<typeAliases>

<typeAlias alias=*"Student"*

type=*"com.java1234.model.Student"*/>

</typeAliases>

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<dataSource type=*"POOLED"*>

<property name=*"driver"*

value=*"${jdbc.driverClassName}"* />

<property name=*"url"*

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

<property name=*"username"*

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

<property name=*"password"*

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

</dataSource>

</environment>

</environments>

<mappers>

<mapper

resource=*"com/java1234/mappers/StudentMapper.xml"* />

</mappers>

</configuration>

从<configuration>开始，子标签肯定不止上面写的那么几个，但我一时半会也用不到。

但在开始介绍前，你要记住，这个顺序不可逆，别名排第一，transactionManager 永远在dataSource 前面。

* <environments default=“”>

环境标签，注意那个 s ，子标签为<environment id=“”>，一般

企业开发至少三个环境开发、测试、正式。<environments default=”development”>的意思是当前环境为id是development的环境。

* <transactionManager type=“”>事务管理器，该标签在<environment>下。Type

有两种，JDBC和MANAGED（托管）。各自含义不同，功能也不同，没空仔细研究，有时间再补，现在记住JDBC就行了。

* <dataSource type=“”>

type值有三：

UNPOOLED，最传统的，每一次操作都创建一个连接，用完后在关闭。

POOLED，连接池。

JNDI，没空研究。

Mybatis不像spring，数据源想用哪个就用哪个，跟换外部数据源的操作很繁琐，还没研究过

http://blog.csdn.net/wangwei\_cq/article/details/8590120

* <properties resource=“”url=“”>

用于引入配置文件，url属性里填真实的文件路径，resource里填。。。。我也不知道要怎么形容，相对路径？

<property name=“” value=“”>

这个东西就是注入属性用的吧，<property>位于<dataSource>下，键值对。

* <typeAliases aliases=“”type=“”>

不用别名的话，mapper.xml里每一个地方（parameterType）都要都要写完整类名，我原本以为没什么，就结而论还是挺烦的。但这种写法还是很辛苦，100个类的话要写100个别名，简单方法是这样的。

<typeAliases>

<package name=“com.jojo.model”>

</typeAliases>

这样就会自动过滤这个包名，只取类名。

* <mapper resource=“”class=“”url=“”>

引入映射文件，这个也要配合着标签看。

resource 属性指定的是相对于跟类路径下的资源

*com/java1234/mappers/StudentMapper.xml*

但与typeAliases同理，这样引入映射文件也是不可取的，有100个映射文件就要写100个<mapper>标签，太累。

<mappers>

<package name=“com.jojo.mappers”>

</mappers>

自动配置所有映射文件。

url 属性指定的是通过URL可以获取到的资源

*[file:///E:/StudentMapper.xml](file://E:\\StudentMapper.xml)*

class 靠类，神奇诶，不过我有一种很不安全的感觉

*Com.java1234.mappers.StudentMapper*

这三个属性一次只能用一个，不能同时出现也不能同时为空。

***XXXMapper.xml***

<mapper namespace=*"com.jojo.mapper.StudentMapper"*>

在mybatis中，映射文件中的namespace是用于绑定Dao接口的，当你的namespace绑定接口后，你可以不用写接口实现类，mybatis会通过该绑定自动帮你找到对应要执行的SQL语句。

先建一个接口，然后里头写增删查改方法。Xml里有<insert>、<delete>、<select>、<update>标签。

resultType与resultMap，在xml中自定义的返回类型就用resultMap，不然就用resultType。

* <resultMap>

阿里巴巴的代码规范里讲明不准用 resultType ，只准用 resultMap，就算结果集中的数据与model类对应一致，也只能用resultMap，不知道为什么。。。。

**jdbcType & javaType的对应关系**

CHAR String

VARCHAR String

LONGVARCHAR String

NUMERIC java.math.BigDecimal

DECIMAL java.math.BigDecimal

BIT boolean

BOOLEAN boolean

TINYINT byte

SMALLINT short

INTEGER int

BIGINT long

REAL float

FLOAT double

DOUBLE double

BINARY byte[]

VARBINARY byte[]

LONGVARBINARY byte[]

DATE java.sql.Date

TIME java.sql.Time

TIMESTAMP java.sql.Timestamp

CLOB Clob

BLOB Blob

ARRAY Array

DISTINCT mapping of underlying type

STRUCT Struct

REF Ref

DATALINK java.net.URL[color=red][/color]

奶奶的，好多啊，jdbcType是要用的，而且mybatis generator也会自动生成，但javaType目前真没见谁用过。

***多表查询***

// 单表查询毕竟简单。

* 一对一映射

// 用法很多啊，四五种。这里我就只记两种模块化的用法，毕竟技术的高低不是由背的方法的多或少来决定的，搞清底层原理才是要紧事。

什么叫一对一呢，一张表的一条记录对应另一张表的一条记录，暂时这么理解。先有因，后有果。上面只练过简单的增删查改，但关联查询乃至更复杂的函数，连接才是项目中经常使用的sql语句，问题来了，这些结果要怎么保存呢？

定义一个resultMap，并在里头写上结果集中列。而resultMap又要对应Javabean（不然数据怎么体现在程序里），由于一个类对应一张表，那么一个Javabean就要包含另一个Javabean。看个例子：

Sql语句

<select id=*"findAllWithAddress"* resultMap=*"personResult"*>

SELECT \* FROM t\_person p JOIN t\_address a

ON p.addressId = a.id

</select>

resultMap，嗯哼，模块化

<resultMap type=*"Person"* id=*"personResult"*>

<id property=*"id"* column=*"id"*/>

<result property=*"name"* column=*"name"*/>

<result property=*"age"* column=*"age"*/>

<result property=*"hobby"* column=*"hobby"*/>

<association property=*"address"*

resultMap=*"addressResult"*></association>

</resultMap>

<resultMap type=*"Address"* id=*"addressResult"*>

<id property=*"id"* column=*"id"*/>

<result property=*"provience"* column=*"provience"*/>

<result property=*"city"* column=*"city"*/>

<result property=*"zone"* column=*"zone"*/>

</resultMap>

Javabean

**public** **class** Person {

**private** **int** id;

**private** String name;

**private** **int** age;

**private** String hobby;

**private** Address address;

现在再看这种写法，确实不好，表面上看是模块化，但！addressResult写在PersonMapper.xml 里，这不是扯淡么？所以在AddressMapper.xml里写addressResult，PersonMapper.xml 里写personResult，需要时再组合才是更为理想的法子。

看着就条理清晰。

<resultMap type=*"Person"* id=*"personResult"*>

<id property=*"id"* column=*"id"*/>

<result property=*"name"* column=*"name"*/>

<result property=*"age"* column=*"age"*/>

<result property=*"hobby"* column=*"hobby"*/>

<association property=*"address"* column=*"addressId"*

select=*"com.jojo.mapper.AddressMapper.findAddressById"*></association>

</resultMap>

<resultMap type=*"Address"* id=*"addressResult"*>

<id property=*"id"* column=*"id"*/>

<result property=*"provience"* column=*"provience"*/>

<result property=*"city"* column=*"city"*/>

<result property=*"zone"* column=*"zone"*/>

</resultMap>

<select id=*"findAddressById"* parameterType=*"Integer"* resultType=*"Address"*>

SELECT \* FROM t\_address WHERE id = #{id}

</select>

这种写法甚至还从逻辑上简化了 SQL 语句，不需要写连接，两个简单的查询语句拼接就搞定了。

最重要的一点就是前后对应一致，两个mapper接口，两个xml，四个文件中关联的名字要统统对应起来！（我的天敌。。。。而且这时候就体现出刷LeetCode的好来了，细心，考虑周全）

* 一对多映射

一对一映射，一张表的一条记录对应另一张表的一条记录；一对多映射，一张表的一条记录对应另一张表的多条记录。

那么，注意逻辑不要紊乱，注意前后名称一致，把<association>换成<collection>，结束。

***动态SQL语句***

// 这个东西，原本以为它有多难呢。。。结果条理异常清晰，基本一看就会。另一方面，它讲解起来有很繁琐，因为没有需求，很难灵活的展示，因此只能列举用法。

* <if test=””>
* <choose>

<when test=””></when>

<when test=””></when>

<otherwise></otherwise>

<choose>

* <where>

<if test="userName != null">and userName = #{userName}</if>

<if test="hobby != null">or hobby = #{hobby}</if>

...... // 有条件时自动加上 WHERE 并去掉第一个 and/or，不过第一个我倾向于直接不加

</where>

* <trim prefix=”WHERE”prefixOverrides=”AND | OR”>

..... // 加前缀与加后缀，用到的次数不是太多 吧？

</trim>

* <set>

<if test="userName != null"> userName = #{userName}, </if>

<if test="hobby != null"> hobby = #{hobby}, </if>

.... // 这个用的也比较多，update语句的那个 SET ，配合 <if>用，最后一个逗号会自动去除

</set>

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

#{item}

// 遍历元素， in(\*, \*, \* ) not in(\*, \*, \* ) 那个index和item一样需要定义，代表遍历的序号，不是从几号开始遍历。。。。

</foreach>

***CLOB/BLOB***

// “大数据”

BLOB 在 Javabean 里对应 byte[] 数组，CLOB 对应 String 就行。

BLOB 涉及到文件，所以又要和 File 还有 I/O 打交道了。从流中获取字节并存入数组，然后调用 setters 赋值，接着直接插入就好。获取时，直接获取就好，但仔细想想，jdbc好像也是直接就用了啊。

结果这两样东西没什么神奇的。

***多个参数***

一个参数的话是 parameterType=“...”然后传类，传Integer，传String。多个参数要怎么办呢？

public Student searchStudents6(String name, Integer age);

方案一：

<select id="searchStudents6" resultMap="StudentResult">

select \* from t\_student where name like #{0} and age=#{1}

</select>

很方便啊，直接0, 1, 2。

方案二：

使用@Param注解

public Student searchStudents6(@Param(“name”)String name,

@Param(“age”)Integer age);

<select id="searchStudents6" resultMap="StudentResult">

select \* from t\_student where name like #{name} and age=#{age}

</select>

方案三：

将参数封装成map，按键值对取 #{key}

将参数封装成list，用foreach遍历

***分页***

* 逻辑分页

主要依靠 RowBounds 类，这个分页功能很弱，它是基于内存的分页（查出所有记录再按偏移量和limit取结果），在大数据量的情况下这样的分页基本上是没有用的。由于分页是在所有数据都取出之后再进行的，因此 XXXMapper.xml 中并不需要做额外的配置。

List<Student> findStudent(RowBounds rowBounds);

RowBounds(int offset, int limit)

函数这么写，然后提供一个 RowBounds 实例就好。

* 物理分页

// 我们的老伙计，sql语句

传一个 Map 进去，然后用 <if> 拼接，limit #{start}， #{size}

***缓存***

缓存分为一级缓存与二级缓存，很强大，用于存放临时数据。

一级缓存的作用范围是一个sqlSession，使用缓存后，下一次调用 select 就不进数据库，直接从缓存中返回。但表中任何一个字段被修改后，缓存便失效了。

***注解配置***

水很深，但按照我一直以来的想法，一个人的技术高低和他能记住多少用法无关，真正精华的是底层的实现，能透彻原理并逻辑清晰的表达出来才是最根本的。

////////////////////////////////////////////////////////////////////////////////////////////

2017.2.28 被逼的没办法，不看了，直接上。啧，我还是不懂，尼玛老子概念一个不会，直接就去抄别人代码，有什么意思，好歹让我记几个

好烦，我第一步应该是先建数据库，建 jdbc.properties 然后设置mybatis-config.xml，接着写model并配置相应的 XXXMapper 及 XXXMapper.xml 文件。最后再回去补mybatis-config.xml。

.properties 内不用双引号，而且不能他妈的有空格。

前后名字要一致。

这两个bug卡了我半个钟头。就这么两个弱智bug。

增删改要提交，sqlSession.Commit();

Mybatis省略了JDBC代码，额不对，是完全屏蔽了JDBC代码。它索性只剩了个接口，连实现都不要！JDBC操作数据库前要先设置参数，Mybatis不需要；JDBC获得数据后要挨个抽取结果集，Mybatis还是不需要。仔细想想确实对，简单SQL语句无非是get,set何苦浪费自己的生命。。。。

2017.3.28

Mybatis不能说看完，但基础用法已经了解完毕。总的来看其实就是两个配置文件的事。

*mybatis-config.xml*

在这个 xml 里配置运行的环境<environment>，要连接的数据库<datasource>，事务管理方式<transactionManager>。存在哪些 DAO <mappers>，给 model 起别名（这个不是必须的，但最好写上）。

*XXXMapper.xml*

增删查改的SQL语句，要返回的结果集类型，动态拼接，可复用片段。

######################### **mybatis generator** #####################

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE generatorConfiguration

PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

<!-- 配置生成器 -->

<generatorConfiguration>

<!-- 可以用于加载配置项或者配置文件，在整个配置文件中就可以使用${propertyKey}的方式来引用配置项 resource：配置资源加载地址，使用resource，MBG从classpath开始找，比如com/myproject/generatorConfig.properties

url：配置资源加载地质，使用URL的方式，比如file:///C:/myfolder/generatorConfig.properties. 注意，两个属性只能选址一个;

另外，如果使用了mybatis-generator-maven-plugin，那么在pom.xml中定义的properties都可以直接在generatorConfig.xml中使用

<properties resource="" url="" /> -->

<!-- 在MBG工作的时候，需要额外加载的依赖包 location属性指明加载jar/zip包的全路径 <classPathEntry location="/Program

Files/IBM/SQLLIB/java/db2java.zip" /> -->

<!-- context:生成一组对象的环境 id:必选，上下文id，用于在生成错误时提示 defaultModelType:指定生成对象的样式

1，conditional：类似hierarchical； 2，flat：所有内容（主键，blob）等全部生成在一个对象中； 3，hierarchical：主键生成一个XXKey对象(key

class)，Blob等单独生成一个对象，其他简单属性在一个对象中(record class) targetRuntime: 1，MyBatis3：默认的值，生成基于MyBatis3.x以上版本的内容，包括XXXBySample；

2，MyBatis3Simple：类似MyBatis3，只是不生成XXXBySample； introspectedColumnImpl：类全限定名，用于扩展MBG -->

<context id=*"mysql"* defaultModelType=*"hierarchical"*

targetRuntime=*"MyBatis3Simple"*>

<!-- 自动识别数据库关键字，默认false，如果设置为true，根据SqlReservedWords中定义的关键字列表； 一般保留默认值，遇到数据库关键字（Java关键字），使用columnOverride覆盖 -->

<property name=*"autoDelimitKeywords"* value=*"false"* />

<!-- 生成的Java文件的编码 -->

<property name=*"javaFileEncoding"* value=*"UTF-8"* />

<!-- 格式化java代码 -->

<property name=*"javaFormatter"*

value=*"org.mybatis.generator.api.dom.DefaultJavaFormatter"* />

<!-- 格式化XML代码 -->

<property name=*"xmlFormatter"*

value=*"org.mybatis.generator.api.dom.DefaultXmlFormatter"* />

<!-- beginningDelimiter和endingDelimiter：指明数据库的用于标记数据库对象名的符号，比如ORACLE就是双引号，MYSQL默认是`反引号； -->

<property name=*"beginningDelimiter"* value=*"`"* />

<property name=*"endingDelimiter"* value=*"`"* />

<!-- 必须要有的，使用这个配置链接数据库 @TODO:是否可以扩展 -->

<jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*

connectionURL=*"jdbc:mysql:///pss"* userId=*"root"* password=*"admin"*>

<!-- 这里面可以设置property属性，每一个property属性都设置到配置的Driver上 -->

</jdbcConnection>

<!-- java类型处理器 用于处理DB中的类型到Java中的类型，默认使用JavaTypeResolverDefaultImpl； 注意一点，默认会先尝试使用Integer，Long，Short等来对应DECIMAL和

NUMERIC数据类型； -->

<javaTypeResolver

type=*"org.mybatis.generator.internal.types.JavaTypeResolverDefaultImpl"*>

<!-- true：使用BigDecimal对应DECIMAL和 NUMERIC数据类型 false：默认, scale>0;length>18：使用BigDecimal;

scale=0;length[10,18]：使用Long； scale=0;length[5,9]：使用Integer； scale=0;length<5：使用Short； -->

<property name=*"forceBigDecimals"* value=*"false"* />

</javaTypeResolver>

<!-- java模型创建器，是必须要的元素 负责：1，key类（见context的defaultModelType）；2，java类；3，查询类

targetPackage：生成的类要放的包，真实的包受enableSubPackages属性控制； targetProject：目标项目，指定一个存在的目录下，生成的内容会放到指定目录中，如果目录不存在，MBG不会自动建目录 -->

<javaModelGenerator targetPackage=*"com.\_520it.mybatis.domain"*

targetProject=*"src/main/java"*>

<!-- for MyBatis3/MyBatis3Simple 自动为每一个生成的类创建一个构造方法，构造方法包含了所有的field；而不是使用setter； -->

<property name=*"constructorBased"* value=*"false"* />

<!-- 在targetPackage的基础上，根据数据库的schema再生成一层package，最终生成的类放在这个package下，默认为false -->

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

<!-- for MyBatis3 / MyBatis3Simple 是否创建一个不可变的类，如果为true， 那么MBG会创建一个没有setter方法的类，取而代之的是类似constructorBased的类 -->

<property name=*"immutable"* value=*"false"* />

<!-- 设置一个根对象， 如果设置了这个根对象，那么生成的keyClass或者recordClass会继承这个类；在Table的rootClass属性中可以覆盖该选项

注意：如果在key class或者record class中有root class相同的属性，MBG就不会重新生成这些属性了，包括： 1，属性名相同，类型相同，有相同的getter/setter方法； -->

<property name=*"rootClass"* value=*"com.\_520it.mybatis.domain.BaseDomain"* />

<!-- 设置是否在getter方法中，对String类型字段调用trim()方法 -->

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

</javaModelGenerator>

<!-- 生成SQL map的XML文件生成器， 注意，在Mybatis3之后，我们可以使用mapper.xml文件+Mapper接口（或者不用mapper接口），

或者只使用Mapper接口+Annotation，所以，如果 javaClientGenerator配置中配置了需要生成XML的话，这个元素就必须配置

targetPackage/targetProject:同javaModelGenerator -->

<sqlMapGenerator targetPackage=*"com.\_520it.mybatis.mapper"*

targetProject=*"src/main/resources"*>

<!-- 在targetPackage的基础上，根据数据库的schema再生成一层package，最终生成的类放在这个package下，默认为false -->

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

</sqlMapGenerator>

<!-- 对于mybatis来说，即生成Mapper接口，注意，如果没有配置该元素，那么默认不会生成Mapper接口 targetPackage/targetProject:同javaModelGenerator

type：选择怎么生成mapper接口（在MyBatis3/MyBatis3Simple下）： 1，ANNOTATEDMAPPER：会生成使用Mapper接口+Annotation的方式创建（SQL生成在annotation中），不会生成对应的XML；

2，MIXEDMAPPER：使用混合配置，会生成Mapper接口，并适当添加合适的Annotation，但是XML会生成在XML中； 3，XMLMAPPER：会生成Mapper接口，接口完全依赖XML；

注意，如果context是MyBatis3Simple：只支持ANNOTATEDMAPPER和XMLMAPPER -->

<javaClientGenerator targetPackage=*"com.\_520it.mybatis.mapper"*

type=*"ANNOTATEDMAPPER"* targetProject=*"src/main/java"*>

<!-- 在targetPackage的基础上，根据数据库的schema再生成一层package，最终生成的类放在这个package下，默认为false -->

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

<!-- 可以为所有生成的接口添加一个父接口，但是MBG只负责生成，不负责检查 <property name="rootInterface"

value=""/> -->

</javaClientGenerator>

<!-- 选择一个table来生成相关文件，可以有一个或多个table，必须要有table元素 选择的table会生成一下文件： 1，SQL

map文件 2，生成一个主键类； 3，除了BLOB和主键的其他字段的类； 4，包含BLOB的类； 5，一个用户生成动态查询的条件类（selectByExample,

deleteByExample），可选； 6，Mapper接口（可选） tableName（必要）：要生成对象的表名； 注意：大小写敏感问题。正常情况下，MBG会自动的去识别数据库标识符的大小写敏感度，在一般情况下，MBG会

根据设置的schema，catalog或tablename去查询数据表，按照下面的流程： 1，如果schema，catalog或tablename中有空格，那么设置的是什么格式，就精确的使用指定的大小写格式去查询；

2，否则，如果数据库的标识符使用大写的，那么MBG自动把表名变成大写再查找； 3，否则，如果数据库的标识符使用小写的，那么MBG自动把表名变成小写再查找；

4，否则，使用指定的大小写格式查询； 另外的，如果在创建表的时候，使用的""把数据库对象规定大小写，就算数据库标识符是使用的大写，在这种情况下也会使用给定的大小写来创建表名；

这个时候，请设置delimitIdentifiers="true"即可保留大小写格式； 可选： 1，schema：数据库的schema； 2，catalog：数据库的catalog；

3，alias：为数据表设置的别名，如果设置了alias，那么生成的所有的SELECT SQL语句中，列名会变成：alias\_actualColumnName

4，domainObjectName：生成的domain类的名字，如果不设置，直接使用表名作为domain类的名字；可以设置为somepck.domainName，那么会自动把domainName类再放到somepck包里面；

5，enableInsert（默认true）：指定是否生成insert语句； 6，enableSelectByPrimaryKey（默认true）：指定是否生成按照主键查询对象的语句（就是getById或get）；

7，enableSelectByExample（默认true）：MyBatis3Simple为false，指定是否生成动态查询语句； 8，enableUpdateByPrimaryKey（默认true）：指定是否生成按照主键修改对象的语句（即update)；

9，enableDeleteByPrimaryKey（默认true）：指定是否生成按照主键删除对象的语句（即delete）； 10，enableDeleteByExample（默认true）：MyBatis3Simple为false，指定是否生成动态删除语句；

11，enableCountByExample（默认true）：MyBatis3Simple为false，指定是否生成动态查询总条数语句（用于分页的总条数查询）；

12，enableUpdateByExample（默认true）：MyBatis3Simple为false，指定是否生成动态修改语句（只修改对象中不为空的属性）；

13，modelType：参考context元素的defaultModelType，相当于覆盖； 14，delimitIdentifiers：参考tableName的解释，注意，默认的delimitIdentifiers是双引号，如果类似MYSQL这样的数据库，使用的是`（反引号，那么还需要设置context的beginningDelimiter和endingDelimiter属性）

15，delimitAllColumns：设置是否所有生成的SQL中的列名都使用标识符引起来。默认为false，delimitIdentifiers参考context的属性

注意，table里面很多参数都是对javaModelGenerator，context等元素的默认属性的一个复写； -->

<table tableName=*"userinfo"*>

<!-- 参考 javaModelGenerator 的 constructorBased属性 -->

<property name=*"constructorBased"* value=*"false"* />

<!-- 默认为false，如果设置为true，在生成的SQL中，table名字不会加上catalog或schema； -->

<property name=*"ignoreQualifiersAtRuntime"* value=*"false"* />

<!-- 参考 javaModelGenerator 的 immutable 属性 -->

<property name=*"immutable"* value=*"false"* />

<!-- 指定是否只生成domain类，如果设置为true，只生成domain类，如果还配置了sqlMapGenerator，那么在mapper

XML文件中，只生成resultMap元素 -->

<property name=*"modelOnly"* value=*"false"* />

<!-- 参考 javaModelGenerator 的 rootClass 属性 <property name="rootClass" value=""/> -->

<!-- 参考javaClientGenerator 的 rootInterface 属性 <property name="rootInterface"

value=""/> -->

<!-- 如果设置了runtimeCatalog，那么在生成的SQL中，使用该指定的catalog，而不是table元素上的catalog

<property name="runtimeCatalog" value=""/> -->

<!-- 如果设置了runtimeSchema，那么在生成的SQL中，使用该指定的schema，而不是table元素上的schema <property

name="runtimeSchema" value=""/> -->

<!-- 如果设置了runtimeTableName，那么在生成的SQL中，使用该指定的tablename，而不是table元素上的tablename

<property name="runtimeTableName" value=""/> -->

<!-- 注意，该属性只针对MyBatis3Simple有用； 如果选择的runtime是MyBatis3Simple，那么会生成一个SelectAll方法，如果指定了selectAllOrderByClause，那么会在该SQL中添加指定的这个order条件； -->

<property name=*"selectAllOrderByClause"* value=*"age desc,username asc"* />

<!-- 如果设置为true，生成的model类会直接使用column本身的名字，而不会再使用驼峰命名方法，比如BORN\_DATE，生成的属性名字就是BORN\_DATE,而不会是bornDate -->

<property name=*"useActualColumnNames"* value=*"false"* />

<!-- generatedKey用于生成生成主键的方法， 如果设置了该元素，MBG会在生成的<insert>元素中生成一条正确的<selectKey>元素，该元素可选

column:主键的列名； sqlStatement：要生成的selectKey语句，有以下可选项： Cloudscape:相当于selectKey的SQL为：

VALUES IDENTITY\_VAL\_LOCAL() DB2 :相当于selectKey的SQL为： VALUES IDENTITY\_VAL\_LOCAL()

DB2\_MF :相当于selectKey的SQL为：SELECT IDENTITY\_VAL\_LOCAL() FROM SYSIBM.SYSDUMMY1

Derby :相当于selectKey的SQL为：VALUES IDENTITY\_VAL\_LOCAL() HSQLDB :相当于selectKey的SQL为：CALL

IDENTITY() Informix :相当于selectKey的SQL为：select dbinfo('sqlca.sqlerrd1') from

systables where tabid=1 MySql :相当于selectKey的SQL为：SELECT LAST\_INSERT\_ID()

SqlServer :相当于selectKey的SQL为：SELECT SCOPE\_IDENTITY() SYBASE :相当于selectKey的SQL为：SELECT

@@IDENTITY JDBC :相当于在生成的insert元素上添加useGeneratedKeys="true"和keyProperty属性

<generatedKey column="" sqlStatement=""/> -->

<!-- 该元素会在根据表中列名计算对象属性名之前先重命名列名，非常适合用于表中的列都有公用的前缀字符串的时候， 比如列名为：CUST\_ID,CUST\_NAME,CUST\_EMAIL,CUST\_ADDRESS等；

那么就可以设置searchString为"^CUST\_"，并使用空白替换，那么生成的Customer对象中的属性名称就不是 custId,custName等，而是先被替换为ID,NAME,EMAIL,然后变成属性：id，name，email；

注意，MBG是使用java.util.regex.Matcher.replaceAll来替换searchString和replaceString的，

如果使用了columnOverride元素，该属性无效； <columnRenamingRule searchString="" replaceString=""/> -->

<!-- 用来修改表中某个列的属性，MBG会使用修改后的列来生成domain的属性； column:要重新设置的列名； 注意，一个table元素中可以有多个columnOverride元素哈~ -->

<columnOverride column=*"username"*>

<!-- 使用property属性来指定列要生成的属性名称 -->

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

<!-- javaType用于指定生成的domain的属性类型，使用类型的全限定名 <property name="javaType" value=""/> -->

<!-- jdbcType用于指定该列的JDBC类型 <property name="jdbcType" value=""/> -->

<!-- typeHandler 用于指定该列使用到的TypeHandler，如果要指定，配置类型处理器的全限定名 注意，mybatis中，不会生成到mybatis-config.xml中的typeHandler

只会生成类似：where id = #{id,jdbcType=BIGINT,typeHandler=com.\_520it.mybatis.MyTypeHandler}的参数描述

<property name="jdbcType" value=""/> -->

<!-- 参考table元素的delimitAllColumns配置，默认为false <property name="delimitedColumnName"

value=""/> -->

</columnOverride>

<!-- ignoreColumn设置一个MGB忽略的列，如果设置了改列，那么在生成的domain中，生成的SQL中，都不会有该列出现 column:指定要忽略的列的名字；

delimitedColumnName：参考table元素的delimitAllColumns配置，默认为false 注意，一个table元素中可以有多个ignoreColumn元素

<ignoreColumn column="deptId" delimitedColumnName=""/> -->

</table>

</context>

</generatorConfiguration>

List<String> warnings = **new** ArrayList<String>();

**boolean** overwrite = **false**;

File configFile = ResourceUtils.*getFile*("classpath:generatorConfigruation.xml");

ConfigurationParser cp = **new** ConfigurationParser(warnings);

Configuration config = cp.parseConfiguration(configFile);

DefaultShellCallback callback = **new** DefaultShellCallback(overwrite);

MyBatisGenerator myBatisGenerator = **new** MyBatisGenerator(config, callback, warnings);

myBatisGenerator.generate(**null**);

**for** (String warning : warnings) {

System.***out***.println(warning);

}

System.***out***.println("done.");