AutoPro代码自动生成器使用说明书

作者 田守枝

第一部分 前言

# 1 AutoPro代码自动生成插件简介

## 1.1编写AutoPro代码自动生成器的起因

AutoPro是一个以Eclipse插件形式存在的代码生成器，编写这个代码生成器的起因是笔者对前公司使用代码生成器复杂、繁琐的操作感觉到非常的不满意，于是产生了自己编写一个代码生成器的想法，这个想法最终就成为了作为使用者或者潜在使用者的你现在看到的这个Eclipse插件。

目前市面上已经有一些代码自动生成器，最著名应该就是apache软件基金会的Abator代码自动生成工具(Dao层可以使用Mybatis或Ibatis)，而基本上每个软件公司也都有着自己的代码生成器，这些代码生成器或多或少的减少了我们很多重复性工作。到目前为止，笔者见过的代码生成有三类：

1. 第一类：基于xml配置文件。即将数据库表信息配置到一个xml文件中，然后使用程序解析xml配置文件，按照预先设定好的规则生成代码。这个解析程序可能是类似于abator写成一个Eclipse插件，也可能就是普通的一个Java程序。

缺点：必须先设计好数据库，然后将各个表、表中的各个字段配置到xml文件中，如果数据库的表或者字段过多的话，那么将数据库的元信息配置到xml文件中也是一个繁琐的操作。

1. 第二类：基于网页的形式。在网页上可以直接设计数据库的表信息。根据这些设计，可以自动生成数据库的表，和对应的增删改查代码。

从操作上来说，这种方式更简单。

缺点：必须先要将自动生成代码的web应用部署运行起来，然后才能进行工作。

1. 第三类：Eclipse插件形式。点击Eclipse界面上的一个按钮，弹出一个界面，在界面上进行相关配置。鼠标点一点既可以完成代码自动生成的操作。

从笔者个人来说，是比较倾向于第三类的。因为Java程序员使用的开发工具基本上都是Eclipse。以插件形式存在的代码生成器，更加方便，Eclipse打开就可以进行代码自动生成的设置，不需要再填写xml配置文件或者运行起来一个web应用。因此笔者缩写的代码生成器也就是以Eclipse插件的形式存在。

那么既然有了基于Eclipse插件形式的代码生成器，为什么笔者还要写一个呢？

很简单，因为觉得现有的Eclipse插件形式的代码生成器功能还不够强大，操作还比较繁琐。以abator为例，其Dao层只支持Mybatis或者Ibatis，对于其他的框架例如Hibernate、JPA等不支持，同时配置比较麻烦。

更为重要的是，现在的代码自动生成框架基本上只支持一种框架整合方式，例如只支持SSI(SpringMvc、Spring、Ibatis)，这是因为大多数公司的框架选型一般都是固定的。使用SSI的公司很少去使用SSH(Struts、Spring、Hibernate)，所以也就不会花费人力物力再去编写一个支持SSH的代码自动生成器。

## 1.2 AutoPro代码自动生成器的特点

**1、多数据库支持**：数据库支持Mysql、Oracle、SqlServer，DB2等

**2、强大的框架支持**：根据现在流行的三层架构设计，AutoPro的Dao层支持JDBC、Hibernate、Mybatis、Ibatis、JPA；业务层暂时只支持Spring；web层支持SpringMvc、Struts2。

**3、数据库表关联配置信息的生成：**目关联表代码自动生成是代码生成器的一个难点，目前市面上的代码生成器大多数都是基于单表的，对于多表不做处理。因为多表关联涉及到具体的业务逻辑，代码生成器的作者不可能知道你的具体业务逻辑是什么，因此就不处理了。AutoPro的特点在于虽然不生成具体的业务逻辑代码，但是会将实体JavaBean中的关联关系自动生成，同时将配置文件中的关联关系生成。

**4、选择性其他框架整合**：AutoPro除了Spring自动整合Dao层和web层之外，还可以选择性的支持整合其他的第三方类库，例如Activiti、JBPM、Apache POI(默认)、CXF、quartz、redis、memcached、Lucene/Compasss等，AutoPro可以自动生成好Spring整合这些类库的配置信息，并且生成相关的工具类。

**5、完善的功能：**生成的代码默认完成了增删改查功能，具体的包括添加、批量添加、删除、批量删除、基于主键查询、基于条件查询、更新、客户端js校验、服务校验等功能。

**6、测试代码的自动生成**：这是AutoPro的一个亮点，针对每一个表的增删改查都自动生成相应的测试代码，很多插件都没有这个功能哦。

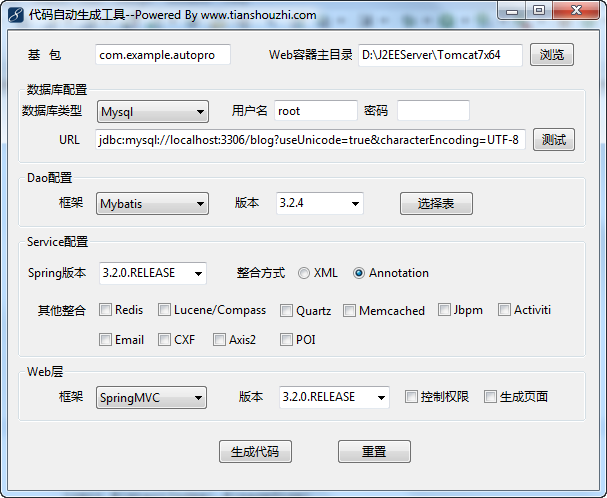
**7、漂亮的界面**：AutoPro生成的JSP界面使用bootstrap3和HTML5，因此生成界面非常好看(从笔者个人的角度来说，是比较好看的).

**8、基于Maven**：一些Eclipse插件将生成代码所依赖的第三方jar包全部放入到插件中，导致插件的体积非常庞大，而且jar包的版本不能升级。AutoPro基于Maven对jar包自动管理功能，只需要生成一个pom.xml文件就可以自动下载相关jar包(不过这就意味着，你要使用AutoPro，就一定要先使用了Maven，这是当前的一个不足点)。

**9、智能记忆功能**：进行过一次配置之后，AutoPro会记住用户之前配置过的信息，再次使用时会自动填充表单，不需要重复配置。例如读者配置好了数据库的信息，选择要生成的代码是基于SSH的，那么AutoPro下次就会自动帮你填充好框架和数据库的配置信息，最大限度的减少你不必要的操作。

**10、创建者个人信息的生成：**基本软件公司都会要求开发人员写代码的时候除了备注之外，还要写上类或者方法的创建者的个人信息，以便于以后的责任确定。AutoPro默认采用”system.user”属性作为创建者的信息。

读者可以对照下面的信息配置界面来理解上面的各个特点。



第二部分 软件使用指南

由于AutoPro是笔者单人开发的，没有经过严格的测试，推荐读者使用AutoPro时的环境如下：

1. Eclipse版本不低于3.5
2. JDK的版本不低于1.6.0.24

## 2.1 插件的下载

下载地址:www.tianshouzhi.com/autopro/download.shtml

插件版本的命名方式是：AutoPro-主版本.次版本.修订版-发布日期.jar

其中次版本是偶数的是稳定版(因为没有经过严格的测试，可能还不够稳定)，奇数则为测试版

例如：AutoPro-1.0.1-20150410.jar

## 2.2 插件的安装

将下载下来的jar包放入Eclipse安装目录的dropins目录下即可。

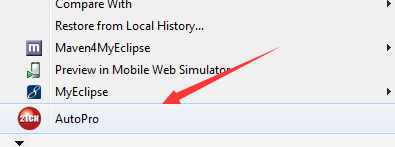
## 2.3 插件的使用步骤

**步骤一：新建一个Maven的Web项目**

此处笔者以新建的项目名为myblog为例，

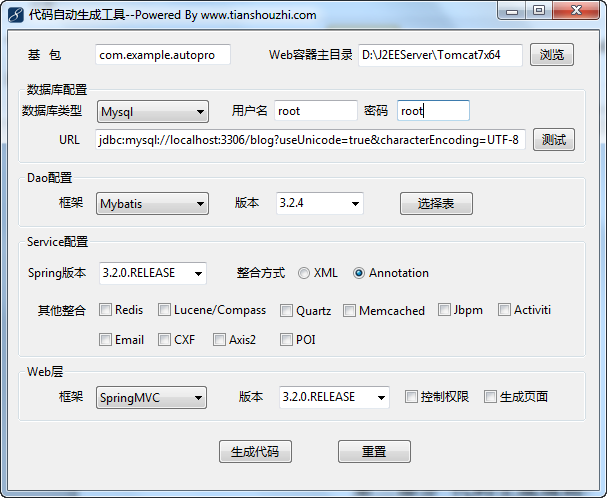
**步骤二：右击项目名**

在弹出的界面中会出现一个AutoPro的选项



**步骤三：点击AutoPro条目**

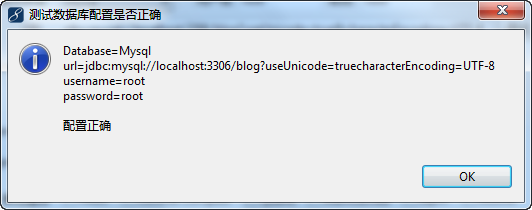
出现界面



在此界面填入必要的配置信息，点击生成代码即可。

## 2.4 配置项详解

1. 基包：按照惯例，基包的命名规范一般是：公司域名倒写.项目名(可以优化，从pom.xml文件中读取)，生成的所有代码都放在这个基包下面
2. Web容器主目录：假设你使用的web容器是tomcat，这个目录就是webapps目录的上一级目录。如果你使用的是MyEclipse，这一项可以忽略，原因你懂的。如果你使用的Eclipse，这个配置会支持将代码自动部署到webapps目录下。
3. 数据库配置：AutoPro的使用是基于你已经设计好了数据表，此处选择数据库类型，数据库实例的url和访问数据库所需的用户名和密码。“测试”按钮用于检测数据库配置是否正确，如果正确的话，会显示类似的界面：

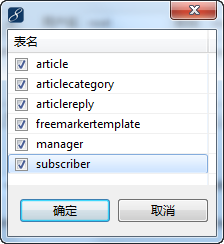


如果不点击测试，配置正确的话也是可以生成代码的，但是建议还是点击一下测试，避免配置错误。

1. Dao配置

用于选择Dao框架的类型和版本号，AutoPro会基于这些信息生成pom.xml文件中的依赖。这里并没有罗列出所有的版本号，没必要，只是把一些典型的稳定版放在这里。数据库连接池默认使用的是C3P0，未来会添加对数据库连接池选择的支持。

“选择表”按钮用于确定哪些表是需要生成相应的代码的，点击之后，出现如下界面：



默认情况下，所有的非关联表都默认选中了，会生成相应的代码。

1. Service配置

由于Service层目前只支持Spring，所以业务层的框架不能进行选择，只能选择Spring的版本号。配置方式支持”XML”和“注解”方式。注意”XML”方式是所有的配置都使用XML。”注解”方式是能使用注解的地方都使用注解，有些地方必须使用XML方式的还是会使用XML。

“其他整合”部分，勾选中的第三方类库，和Spring整合的配置会自动生成，同时会生成相应的工具类。

1. Web层

目前支持Struts2和SpringMvc。

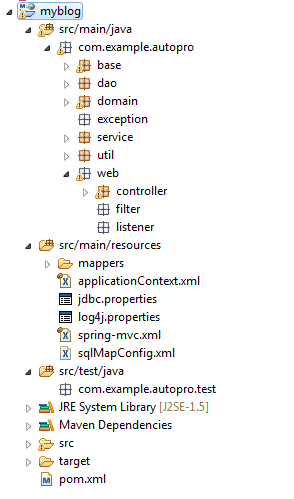
“控制权限”复选框暂时没用

“生成页面”复选框用于确定是否需要生成对应的JSP页面，主要是考虑有一些项目只需要写一些接口，不需要生成JSP页面。

Web层对json支持默认使用的Google的gson，暂时不支持选择。

## 2.5生成的代码目录结构

在所有的配置都生成完成之后，点击生成代码，生成的代码目录结构如下：



相信以你的聪明才智，不用解释就知道这个目录结构的含义了。

第三部分 代码生成规则

# 3、数据库表与实体(JavaBean)映射规则

## 3.1实体名生成规则

1、假设数据库表名为user、User、USER等🡪生成的实体名为User

即将数据库表名都先转换为小写之后，首字母大写

2、数据库表名为 o\_user，(根据很多公司的命名惯例，在实体表之前加上一个o)，则生成的表名依然为User，即将”o\_表名”中的”o\_”去除，然后首字母大写

3、数据库表名为o\_article\_category、article\_category，生成的表名为ArticleCategory。

4、特殊情况，关联表不会生成对应的实体

## 3.2字段生成规则

以数据库表中所有列名都是小写进行讲解

例如一个文章article表如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 类型 | 主键 | 长度 | 外键 | 备注 |
| id | int | 是 |  |  | 主键 |
| title | varchar |  | 50 |  | 标题 |
| keywords | varchar |  | 255(默认) |  | 关键字 |
| content | longtext |  |  |  | 内容 |
| visible | bit(1) |  |  |  | 是否可见 |
| clickCount | int |  |  |  | 点击数 |
| replyCount | int |  |  |  | 回复数 |
| postTime | datetime |  |  |  | 发表时间 |
| category\_id |  |  |  | 是 | 所属类别 |

数据库表中每一个字段对应JavaBean的一个属性，字段的备注对应生成JavaBean的备注，如果没有备注，则默认使用数据库表的字段名作为JavaBean属性名的备注。

int、byte类型转换成的字段类型为Integer

bit(1)转换为的类型为Boolean

longtext、varchar转换为的类型为String

datetime、time、date、timestamp转换为的类型是java.util.Date

float对应Float

long对应Long

double对应Double

外键不生成JavaBean属性

默认生成2个构造方法，一个是无参的，另一个使用除了主键和外键之外的所有字段生成对应的构造方法。

则生成的实体为：

|  |
| --- |
| /\*\*  ,Table: article  \*/  **public** **class** Article **implements** Serializable{    **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** Integer id;  /\*摘要\*/  **private** String abstracts;  /\*点击率\*/  **private** Integer clickCount;  /\*内容\*/  **private** String content;  /\*关键字\*/  **private** String keywords;  /\*发表时间\*/  **private** Date postTime;  /\*回复数\*/  **private** Integer replyCount;  /\*标题\*/  **private** String title;  /\*是否可见\*/  **private** Boolean visible;    **public** Article(){}  **public** Article(String abstracts,Integer clickCount,String content,String keywords,Date postTime,Integer replyCount,String title,Boolean visible){  **this**.abstracts=abstracts;  **this**.clickCount=clickCount;  **this**.content=content;  **this**.keywords=keywords;  **this**.postTime=postTime;  **this**.replyCount=replyCount;  **this**.title=title;  **this**.visible=visible;  }  **public** **void** setId(Integer id){  **this**.id=id;  }  **public** Integer getId(){  **return** **this**.id;  }  **public** **void** setAbstracts(String abstracts){  **this**.abstracts=abstracts;  }  **public** String getAbstracts(){  **return** **this**.abstracts;  }  **public** **void** setClickCount(Integer clickCount){  **this**.clickCount=clickCount;  }  **public** Integer getClickCount(){  **return** **this**.clickCount;  }  **public** **void** setContent(String content){  **this**.content=content;  }  **public** String getContent(){  **return** **this**.content;  }  **public** **void** setKeywords(String keywords){  **this**.keywords=keywords;  }  **public** String getKeywords(){  **return** **this**.keywords;  }  **public** **void** setPostTime(Date postTime){  **this**.postTime=postTime;  }  **public** Date getPostTime(){  **return** **this**.postTime;  }  **public** **void** setReplyCount(Integer replyCount){  **this**.replyCount=replyCount;  }  **public** Integer getReplyCount(){  **return** **this**.replyCount;  }  **public** **void** setTitle(String title){  **this**.title=title;  }  **public** String getTitle(){  **return** **this**.title;  }  **public** **void** setVisible(Boolean visible){  **this**.visible=visible;  }  **public** Boolean getVisible(){  **return** **this**.visible;  }    } |

# 4、DAO生成规则

Dao层框架是可以选择的，预计是要支持使用JDBC、Mybatis、Hibernate、JPA四种ORM框架。

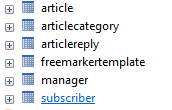
目前DAO的代码自动生成只支持Mybatis的xml配置方式，未来会添加对注解方式配置的支持。

## 4.1 Mybatis生成规则

### 4.1.1 sqlMapConfig生成规则

假设满足以下2点条件

1、数据库中有以下几张表



2、填写的基包的名称为：com.example.autopro

在生成的sqlMapConfig文件中会完成以下两点

1、给每个实体起一个别名，别名就是实体的简单类名

2、列出每个实体映射文件的路径

则生成的sqlMapConfig.xml文件为

|  |
| --- |
| <?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 type=*"com.example.autopro.domain.Article"* alias=*"Article"*/>  <typeAlias type=*"com.example.autopro.domain.Articlecategory"* alias=*"Articlecategory"*/>  <typeAlias type=*"com.example.autopro.domain.Articlereply"* alias=*"Articlereply"*/>  <typeAlias type=*"com.example.autopro.domain.Freemarkertemplate"* alias=*"Freemarkertemplate"*/>  <typeAlias type=*"com.example.autopro.domain.Manager"* alias=*"Manager"*/>  <typeAlias type=*"com.example.autopro.domain.Subscriber"* alias=*"Subscriber"*/>  </typeAliases>  <mappers>  <mapper resource=*"mappers/ArticleMapper.xml"* />  <mapper resource=*"mappers/ArticlecategoryMapper.xml"* />  <mapper resource=*"mappers/ArticlereplyMapper.xml"* />  <mapper resource=*"mappers/FreemarkertemplateMapper.xml"* />  <mapper resource=*"mappers/ManagerMapper.xml"* />  <mapper resource=*"mappers/SubscriberMapper.xml"* />  </mappers>  </configuration> |

### 4.1.2 sqlMapper映射文件的生成规则

映射文件的命名方式为:实体简单名+Mapper.xml

例如生成的实体名为Article，则生成映射文件的名称为ArticleMapper.xml

其他规则：

1、根元素mapper的namespace属性使用实体的全路径

2、resultMap元素的id属性使用”实体名首字母小写+Map”。例如实体为Article，则id属性为articleMap,type属性引用在sqlMapConfig中配置的别名。result子元素分别列出实体每个属性的属性名(property)、对应的数据库列名(column)，对应的java类型全路径(javaType)和对应的jdbcType类型

3、默认生成的操作语句操作语句

insert语句，参数类型为实体(如果主键是自动增长的，则insert语句的主键不会生成)

delete语句，根据主键进行删除一条记录，如果没有主键则不会生成

batchDelete语句,根据主键删除多条记录，参数为实体主键对应的数组，例如实体的主键是id，类型为Integer，则批量删除的参数为Integer[]

update语句 根据id进行修改

selectById语句 根据id查询某一条记录

selectByMap语句，查询分页数据,参数为Map集合，存放是查询条件

同时生成3个用于被包含的语句 where\_Clause，orderBy\_Clause和limit\_Clause，where\_Clause，orderBy\_Clause包含了所有可以被排序的字段

|  |
| --- |
| <?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.example.autopro.domain.Article"* >  <resultMap id=*"articleMap"* type=*"Article"* >  <result column=*"id"* property=*"id"* javaType=*"java.lang.Integer"* jdbcType=*"INTEGER"*/>  <result column=*"abstracts"* property=*"abstracts"* javaType=*"java.lang.String"* jdbcType=*"VARCHAR"*/>  <result column=*"clickCount"* property=*"clickCount"* javaType=*"java.lang.Integer"* jdbcType=*"INTEGER"*/>  <result column=*"content"* property=*"content"* javaType=*"java.lang.String"* jdbcType=*"LONGVARCHAR"*/>  <result column=*"keywords"* property=*"keywords"* javaType=*"java.lang.String"* jdbcType=*"VARCHAR"*/>  <result column=*"postTime"* property=*"postTime"* javaType=*"java.util.Date"* jdbcType=*"TIMESTAMP"*/>  <result column=*"replyCount"* property=*"replyCount"* javaType=*"java.lang.Integer"* jdbcType=*"INTEGER"*/>  <result column=*"title"* property=*"title"* javaType=*"java.lang.String"* jdbcType=*"VARCHAR"*/>  <result column=*"visible"* property=*"visible"* javaType=*"java.lang.Boolean"* jdbcType=*"BIT"*/>  </resultMap>    <insert id=*"insert"* parameterType=*"Article"*>  INSERT INTO article (abstracts,clickCount,content,keywords,postTime,replyCount,title,visible)  VALUES(#{abstracts},#{clickCount},#{content},#{keywords},#{postTime},#{replyCount},#{title},#{visible});  </insert>    <select id=*"selectById"* parameterType=*"java.lang.Integer"* resultMap=*"articleMap"*>  SELECT id,abstracts,clickCount,content,keywords,postTime,replyCount,title,visible FROM article  WHERE id=#{id}  </select>    <delete id=*"deleteById"* parameterType=*"java.lang.Integer"* >  DELETE FROM article  WHERE id =#{id}  </delete>    <update id=*"updateById"* parameterType=*"Article"*>  UPDATE article  <set>  abstracts=#{abstracts},clickCount=#{clickCount},content=#{content},keywords=#{keywords},postTime=#{postTime},replyCount=#{replyCount},title=#{title},visible=#{visible}  </set>  WHERE id=#{id};  </update>  <select id=*"selectByMap"* parameterType=*"java.util.Map"* resultType=*"Article"*>  SELECT id, abstracts,clickCount,content,keywords,postTime,replyCount,title,visible FROM article  <include refid=*"where\_Clause"*/>  <include refid=*"orderBy\_Clause"*/>  <include refid=*"limit\_Clause"*/>  </select>    <select id=*"selectCountByMap"* parameterType=*"java.util.Map"* resultType=*"java.lang.Long"*>  SELECT count(1) FROM article  <include refid=*"where\_Clause"*/>  </select>  <sql id=*"where\_Clause"* >  <where>  <if test=*"id != null"*>  AND id=#{id,javaType=java.lang.Integer,jdbcType=INTEGER}  </if>  <if test=*"abstracts != null"*>  AND abstracts LIKE "%"#{abstracts,javaType=java.lang.String,jdbcType=VARCHAR}"%"  </if>  <if test=*"clickCount != null"*>  AND clickCount=#{clickCount,javaType=java.lang.Integer,jdbcType=INTEGER}  </if>  <if test=*"content != null"*>  AND content LIKE #{content,javaType=java.lang.String,jdbcType=LONGVARCHAR}  </if>  <if test=*"keywords != null"*>  AND keywords LIKE #{keywords,javaType=java.lang.String,jdbcType=VARCHAR}  </if>  <if test=*"postTime != null"*>  AND postTime <![CDATA[ >= ]]> #{postTime\_date\_from,javaType=java.util.Date,jdbcType=TIMESTAMP}  AND postTime <![CDATA[ <= ]]> #{postTime\_date\_to,javaType=java.util.Date,jdbcType=TIMESTAMP}  </if>  <if test=*"replyCount != null"*>  AND replyCount=#{replyCount,javaType=java.lang.Integer,jdbcType=INTEGER}  </if>  <if test=*"title != null"*>  AND title LIKE #{title,javaType=java.lang.String,jdbcType=VARCHAR}  </if>  <if test=*"visible != null"*>  AND visible=#{visible,javaType=java.lang.Boolean,jdbcType=BIT}  </if>  </where>  </sql>  <sql id=*"orderBy\_Clause"*>  <trim prefix=*"ORDER BY"*>  <if test=*"sortName !=null"*>  <if test=*"sortName='id'"*>id</if>  <if test=*"sortName=='abstracts'"*>abstracts</if>  <if test=*"sortName=='clickCount'"*>clickCount</if>  <if test=*"sortName=='content'"*>content</if>  <if test=*"sortName=='keywords'"*>keywords</if>  <if test=*"sortName=='postTime'"*>postTime</if>  <if test=*"sortName=='replyCount'"*>replyCount</if>  <if test=*"sortName=='title'"*>title</if>  <if test=*"sortName=='visible'"*>visible</if>  </if>    <if test=*"sortDirection !=null"*>  <if test=*"sortDirection == 'desc'"*>desc</if>  <if test=*"sortDirection == 'asc'"*>asc</if>  </if>  </trim>  </sql>  <sql id=*"limit\_Clause"*>  <if test=*"startIndex !=null and pageSize!=null"*>  limit #{startIndex},#{pageSize}  </if>  </sql>  </mapper> |