mybatis的文档 <http://mybatis.github.io/mybatis-3/zh/index.html>

http://haohaoxuexi.iteye.com/blog/1851081

# 一、MyBatis动态SQL

MyBatis的动态SQL是基于OGNL表达式的，它可以帮助我们方便的在SQL语句中实现某些逻辑。

MyBatis中用于实现动态SQL的元素主要有：

if

choose（when，otherwise）

trim

where

set

foreach

## 1.1、If语句

if就是简单的条件判断，利用if语句我们可以实现某些简单的条件选择。先来看如下一个例子：

Xml代码：

<select id="dynamicIfTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog where 11 = 1

<if test="title != null">

and title = #{title}

</if>

<if test="content != null">

and content = #{content}

</if>

<if test="owner != null">

and owner = #{owner}

</if>

</select>

这条语句的意思非常简单，如果你提供了title参数，那么就要满足title=#{title}，同样如果你提供了Content和Owner的时候，它们也需要满足相应的条件，之后就是返回满足这些条件的所有Blog，这是非常有用的一个功能，以往我们使用其他类型框架或者直接使用JDBC的时候， 如果我们要达到同样的选择效果的时候，我们就需要拼SQL语句，这是极其麻烦的，比起来，上述的动态SQL就要简单多了。

## 1.2、Choose语句

choose元素的作用就相当于JAVA中的switch语句，基本上跟JSTL中的choose的作用和用法是一样的，通常都是与when和otherwise搭配的。看如下一个例子：

Xml代码：

<select id="dynamicChooseTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog where 1 = 1

<choose>

<when test="title != null">

and title = #{title}

</when>

<when test="content != null">

and content = #{content}

</when>

<otherwise>

and owner = "owner1"

</otherwise>

</choose>

</select>

when元素表示当when中的条件满足的时候就输出其中的内容，跟JAVA中的switch效果差不多的是按照条件的顺序，当when中有条件满足的时候，就会跳出choose，即所有的when和otherwise条件中，只有一个会输出，当所有的我很条件都不满足的时候就输出otherwise中的内容。所以上述语句的意思非常简单， 当title!=null的时候就输出and titlte = #{title}，不再往下判断条件，当title为空且content!=null的时候就输出and content = #{content}，当所有条件都不满足的时候就输出otherwise中的内容。

## 1.3、Where语句

where语句的作用主要是简化SQL语句中where中的条件判断的，先看一个例子，再解释一下where的好处。

Xml代码：

<select id="dynamicWhereTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog

<where>

<if test="title != null">

title = #{title}

</if>

<if test="content != null">

and content = #{content}

</if>

<if test="owner != null">

and owner = #{owner}

</if>

</where>

</select>

where元素的作用是会在写入where元素的地方输出一个where，另外一个好处是你不需要考虑where元素里面的条件输出是什么样子的，MyBatis会智能的帮你处理，如果所有的条件都不满足那么MyBatis就会查出所有的记录，如果输出后是and 开头的，MyBatis会把第一个and忽略，当然如果是or开头的，MyBatis也会把它忽略；此外，在where元素中你不需要考虑空格的问题，MyBatis会智能的帮你加上。像上述例子中，如果title=null， 而content != null，那么输出的整个语句会是select \* from t\_blog where content = #{content}，而不是select \* from t\_blog where and content = #{content}，因为MyBatis会智能的把首个and 或 or 给忽略。

## 1.4、Trim语句

trim元素的主要功能是可以在自己包含的内容前加上某些前缀，也可以在其后加上某些后缀，与之对应的属性是prefix和suffix；可以把包含内容的首部某些内容覆盖，即忽略，也可以把尾部的某些内容覆盖，对应的属性是prefixOverrides和suffixOverrides；正因为trim有这样的功能，所以我们也可以非常简单的利用trim来代替where元素的功能，示例代码如下：

Xml代码：

<select id="dynamicTrimTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog

<trim prefix="where" prefixOverrides="and |or">

<if test="title != null">

title = #{title}

</if>

<if test="content != null">

and content = #{content}

</if>

<if test="owner != null">

or owner = #{owner}

</if>

</trim>

</select>

## 1.5、Set语句

set元素主要是用在更新操作的时候，它的主要功能和where元素其实是差不多的，主要是在包含的语句前输出一个set，然后如果包含的语句是以逗号结束的话将会把该逗号忽略，如果set包含的内容为空的话则会出错。有了set元素我们就可以动态的更新那些修改了的字段。下面是一段示例代码：

Xml代码：

<update id="dynamicSetTest" parameterType="Blog">

update t\_blog

<set>

<if test="title != null">

title = #{title},

</if>

<if test="content != null">

content = #{content},

</if>

<if test="owner != null">

owner = #{owner}

</if>

</set>

where id = #{id}

</update>

上述示例代码中，如果set中一个条件都不满足，即set中包含的内容为空的时候就会报错。

## 1.6、Foreach语句

foreach的主要用在构建in条件中，它可以在SQL语句中进行迭代一个集合。foreach元素的属性主要有item，index，collection，open，separator，close。item表示集合中每一个元素进行迭代时的别名，index指定一个名字，用于表示在迭代过程中，每次迭代到的位置，open表示该语句以什么开始，separator表示在每次进行迭代之间以什么符号作为分隔符，close表示以什么结束，在使用foreach的时候最关键的也是最容易出错的就是collection属性，该属性是必须指定的，但是在不同情况下，该属性的值是不一样的，主要有一下3种情况：

如果传入的是单参数且参数类型是一个List的时候，collection属性值为list

如果传入的是单参数且参数类型是一个array数组的时候，collection的属性值为array

如果传入的参数是多个的时候，我们就需要把它们封装成一个Map了，当然单参数也可以封装成map，实际上如果你在传入参数的时候，在MyBatis里面也是会把它封装成一个Map的，map的key就是参数名，所以这个时候collection属性值就是传入的List或array对象在自己封装的map里面的key

下面分别来看看上述三种情况的示例代码：

### 1.6.1、单参数List的类型

Xml代码：

<select id="dynamicForeachTest" resultType="Blog">

select \* from t\_blog where id in

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

#{item}

</foreach>

</select>

Java代码：

public List<Blog> dynamicForeachTest(List<Integer> ids);

### 1.6.2、单参数array数组的类型

Xml代码：

<select id="dynamicForeach2Test" resultType="Blog">

select \* from t\_blog where id in

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

#{item}

</foreach>

</select>

Java代码：

public List<Blog> dynamicForeach2Test(int[] ids);

### 1.6.3、参数封装成Map的类型

Xml代码：

<select id="dynamicForeach3Test" resultType="Blog">

select \* from t\_blog where title like "%"#{title}"%" and id in

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

#{item}

</foreach>

</select>

Java代码：

public List<Blog> dynamicForeach3Test(Map<String, Object> params);

# 二、Mybatis+Log4j控制台输出SQL语句

## 2.1、pom.xml添加依赖项

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

## 2.2、web.xml中增加listener

<!--Log4j配置-->

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>/WEB-INF/log4j.xml</param-value>

</context-param>

<context-param>

<param-name>log4jRefreshInterval</param-name>

<param-value>60000</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

## 2.3、log4j.xml配置文件

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

<!DOCTYPE log4j:configuration SYSTEM "http://toolkit.alibaba-inc.com/dtd/log4j/log4j.dtd">

<log4j:configuration xmlns:log4j=*'http://jakarta.apache.org/log4j/'*>

<!-- ===================================================================== -->

<!-- 定义全局输出适配器 -->

<!-- ===================================================================== -->

<appender name=*"consoleAppender"* class=*"org.apache.log4j.ConsoleAppender"*>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"[%d{yyyy-MM-dd HH:mm:ss.SSS}]-[%c]-[%t] %x - %m%n"*/>

</layout>

</appender>

<!-- Mybatis and SQL -->

<logger name=*"org.apache.ibatis"* additivity=*"true"*>

<level value=*"DEBUG"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<logger name=*"org.mybatis.spring"* additivity=*"true"*>

<level value=*"ERROR"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<logger name=*"java.sql"* additivity=*"true"*>

<level value=*"DEBUG"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<logger name=*"java.sql.Connection"* additivity=*"true"*>

<level value=*"DEBUG"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<logger name=*"java.sql.Statement"* additivity=*"true"*>

<level value=*"DEBUG"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<logger name=*"java.sql.PreparedStatement"* additivity=*"true"*>

<level value=*"DEBUG"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<!-- Spring -->

<logger name=*"org.springframework"* additivity=*"true"*>

<level value=*"ERROR"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<!-- ActiveMQ -->

<logger name=*"org.apache.activemq"* additivity=*"true"*>

<level value=*"ERROR"*/>

<appender-ref ref=*"consoleAppender"*/>

</logger>

<!-- ===================================================================== -->

<!-- 根日志记录器 -->

<!-- ===================================================================== -->

<root>

<level value=*"DEBUG"*/>

<appender-ref ref=*"consoleAppender"*/>

</root>

</log4j:configuration>