# 概述

项目git地址： <https://github.com/berg-turing/mybatis.git>

# 基础知识

## mapper的DTD(文档类型定义)

约束了mapper文件的写法

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--  Copyright 2009-2017 the original author or authors.  Licensed under the Apache License, Version 2.0 (the "License");  you may not use this file except in compliance with the License.  You may obtain a copy of the License at  http://www.apache.org/licenses/LICENSE-2.0  Unless required by applicable law or agreed to in writing, software  distributed under the License is distributed on an "AS IS" BASIS,  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  See the License for the specific language governing permissions and  limitations under the License.  -->  <!ELEMENT mapper (cache-ref | cache | resultMap\* | parameterMap\* | sql\* | insert\* | update\* | delete\* | select\* )+>  <!ATTLIST mapper  namespace CDATA #IMPLIED  >  <!ELEMENT cache-ref EMPTY>  <!ATTLIST cache-ref  namespace CDATA #REQUIRED  >  <!ELEMENT cache (property\*)>  <!ATTLIST cache  type CDATA #IMPLIED  eviction CDATA #IMPLIED  flushInterval CDATA #IMPLIED  size CDATA #IMPLIED  readOnly CDATA #IMPLIED  blocking CDATA #IMPLIED  >  <!ELEMENT parameterMap (parameter+)?>  <!ATTLIST parameterMap  id CDATA #REQUIRED  type CDATA #REQUIRED  >  <!ELEMENT parameter EMPTY>  <!ATTLIST parameter  property CDATA #REQUIRED  javaType CDATA #IMPLIED  jdbcType CDATA #IMPLIED  mode (IN | OUT | INOUT) #IMPLIED  resultMap CDATA #IMPLIED  scale CDATA #IMPLIED  typeHandler CDATA #IMPLIED  >  <!ELEMENT resultMap (constructor?,id\*,result\*,association\*,collection\*, discriminator?)>  <!ATTLIST resultMap  id CDATA #REQUIRED  type CDATA #REQUIRED  extends CDATA #IMPLIED  autoMapping (true|false) #IMPLIED  >  <!ELEMENT constructor (idArg\*,arg\*)>  <!ELEMENT id EMPTY>  <!ATTLIST id  property CDATA #IMPLIED  javaType CDATA #IMPLIED  column CDATA #IMPLIED  jdbcType CDATA #IMPLIED  typeHandler CDATA #IMPLIED  >  <!ELEMENT result EMPTY>  <!ATTLIST result  property CDATA #IMPLIED  javaType CDATA #IMPLIED  column CDATA #IMPLIED  jdbcType CDATA #IMPLIED  typeHandler CDATA #IMPLIED  >  <!ELEMENT idArg EMPTY>  <!ATTLIST idArg  javaType CDATA #IMPLIED  column CDATA #IMPLIED  jdbcType CDATA #IMPLIED  typeHandler CDATA #IMPLIED  select CDATA #IMPLIED  resultMap CDATA #IMPLIED  name CDATA #IMPLIED  >  <!ELEMENT arg EMPTY>  <!ATTLIST arg  javaType CDATA #IMPLIED  column CDATA #IMPLIED  jdbcType CDATA #IMPLIED  typeHandler CDATA #IMPLIED  select CDATA #IMPLIED  resultMap CDATA #IMPLIED  name CDATA #IMPLIED  >  <!ELEMENT collection (constructor?,id\*,result\*,association\*,collection\*, discriminator?)>  <!ATTLIST collection  property CDATA #REQUIRED  column CDATA #IMPLIED  javaType CDATA #IMPLIED  ofType CDATA #IMPLIED  jdbcType CDATA #IMPLIED  select CDATA #IMPLIED  resultMap CDATA #IMPLIED  typeHandler CDATA #IMPLIED  notNullColumn CDATA #IMPLIED  columnPrefix CDATA #IMPLIED  resultSet CDATA #IMPLIED  foreignColumn CDATA #IMPLIED  autoMapping (true|false) #IMPLIED  fetchType (lazy|eager) #IMPLIED  >  <!ELEMENT association (constructor?,id\*,result\*,association\*,collection\*, discriminator?)>  <!ATTLIST association  property CDATA #REQUIRED  column CDATA #IMPLIED  javaType CDATA #IMPLIED  jdbcType CDATA #IMPLIED  select CDATA #IMPLIED  resultMap CDATA #IMPLIED  typeHandler CDATA #IMPLIED  notNullColumn CDATA #IMPLIED  columnPrefix CDATA #IMPLIED  resultSet CDATA #IMPLIED  foreignColumn CDATA #IMPLIED  autoMapping (true|false) #IMPLIED  fetchType (lazy|eager) #IMPLIED  >  <!ELEMENT discriminator (case+)>  <!ATTLIST discriminator  column CDATA #IMPLIED  javaType CDATA #REQUIRED  jdbcType CDATA #IMPLIED  typeHandler CDATA #IMPLIED  >  <!ELEMENT case (constructor?,id\*,result\*,association\*,collection\*, discriminator?)>  <!ATTLIST case  value CDATA #REQUIRED  resultMap CDATA #IMPLIED  resultType CDATA #IMPLIED  >  <!ELEMENT property EMPTY>  <!ATTLIST property  name CDATA #REQUIRED  value CDATA #REQUIRED  >  <!ELEMENT typeAlias EMPTY>  <!ATTLIST typeAlias  alias CDATA #REQUIRED  type CDATA #REQUIRED  >  <!ELEMENT select (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST select  id CDATA #REQUIRED  parameterMap CDATA #IMPLIED  parameterType CDATA #IMPLIED  resultMap CDATA #IMPLIED  resultType CDATA #IMPLIED  resultSetType (FORWARD\_ONLY | SCROLL\_INSENSITIVE | SCROLL\_SENSITIVE) #IMPLIED  statementType (STATEMENT|PREPARED|CALLABLE) #IMPLIED  fetchSize CDATA #IMPLIED  timeout CDATA #IMPLIED  flushCache (true|false) #IMPLIED  useCache (true|false) #IMPLIED  databaseId CDATA #IMPLIED  lang CDATA #IMPLIED  resultOrdered (true|false) #IMPLIED  resultSets CDATA #IMPLIED  >  <!ELEMENT insert (#PCDATA | selectKey | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST insert  id CDATA #REQUIRED  parameterMap CDATA #IMPLIED  parameterType CDATA #IMPLIED  timeout CDATA #IMPLIED  flushCache (true|false) #IMPLIED  statementType (STATEMENT|PREPARED|CALLABLE) #IMPLIED  keyProperty CDATA #IMPLIED  useGeneratedKeys (true|false) #IMPLIED  keyColumn CDATA #IMPLIED  databaseId CDATA #IMPLIED  lang CDATA #IMPLIED  >  <!ELEMENT selectKey (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST selectKey  resultType CDATA #IMPLIED  statementType (STATEMENT|PREPARED|CALLABLE) #IMPLIED  keyProperty CDATA #IMPLIED  keyColumn CDATA #IMPLIED  order (BEFORE|AFTER) #IMPLIED  databaseId CDATA #IMPLIED  >  <!ELEMENT update (#PCDATA | selectKey | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST update  id CDATA #REQUIRED  parameterMap CDATA #IMPLIED  parameterType CDATA #IMPLIED  timeout CDATA #IMPLIED  flushCache (true|false) #IMPLIED  statementType (STATEMENT|PREPARED|CALLABLE) #IMPLIED  keyProperty CDATA #IMPLIED  useGeneratedKeys (true|false) #IMPLIED  keyColumn CDATA #IMPLIED  databaseId CDATA #IMPLIED  lang CDATA #IMPLIED  >  <!ELEMENT delete (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST delete  id CDATA #REQUIRED  parameterMap CDATA #IMPLIED  parameterType CDATA #IMPLIED  timeout CDATA #IMPLIED  flushCache (true|false) #IMPLIED  statementType (STATEMENT|PREPARED|CALLABLE) #IMPLIED  databaseId CDATA #IMPLIED  lang CDATA #IMPLIED  >  <!-- Dynamic -->  <!ELEMENT include (property+)?>  <!ATTLIST include  refid CDATA #REQUIRED  >  <!ELEMENT bind EMPTY>  <!ATTLIST bind  name CDATA #REQUIRED  value CDATA #REQUIRED  >  <!ELEMENT sql (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST sql  id CDATA #REQUIRED  lang CDATA #IMPLIED  databaseId CDATA #IMPLIED  >  <!ELEMENT trim (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST trim  prefix CDATA #IMPLIED  prefixOverrides CDATA #IMPLIED  suffix CDATA #IMPLIED  suffixOverrides CDATA #IMPLIED  >  <!ELEMENT where (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ELEMENT set (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ELEMENT foreach (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST foreach  collection CDATA #REQUIRED  item CDATA #IMPLIED  index CDATA #IMPLIED  open CDATA #IMPLIED  close CDATA #IMPLIED  separator CDATA #IMPLIED  >  <!ELEMENT choose (when\* , otherwise?)>  <!ELEMENT when (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST when  test CDATA #REQUIRED  >  <!ELEMENT otherwise (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ELEMENT if (#PCDATA | include | trim | where | set | foreach | choose | if | bind)\*>  <!ATTLIST if  test CDATA #REQUIRED  > |

## resultMap标签

说明

用于结果集的映射，是mybatis最强大的功能之一，通过配置resultMap可以根据自己的业务需求映射出复杂的实体对象。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| id | 该resultMap标签的唯一标识 | 是 |
| type | 该resultMap标签映射的结果实体对象类型 | 是 |
| extends | 该resultMap的父级resultMap，他会继承父级所有的内容，是父级所有的子标签而不包含属性 | 否 |
| autoMapping | 自动映射  (true/false) | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| constructor | 见constructor标签详解 | ? |
| id | 见id标签详解 | \* |
| result | 见result标签详解 | \* |
| associate | 见associate标签详解 | \* |
| collection | 见collection标签详解 | \* |
| discriminator | 见discriminator标签详解 | ? |

使用技巧

尽量将所有对象映射的过程交由resultMap来处理，这样可以使查询结果可控。

示例

|  |
| --- |
| <resultMap id=”BaseResultMap” type=”com.berg.dto.Test”>  <constructor>  <idArg name=”id” column=”ID” jdbcType=”BIGINT”/>  <arg name=”name” column=”NAME” jdbcType=”VARCHAR”/>  </constructor>  <result property=”desc” column=”DESC” jdbcType=”VARCHAR”/>  ...  <associate ...>  ...  </associate>  <collection ...>  ...  </collection>  <discriminator ...>  ...  </discriminator>  </resultMap> |

## constructor标签

说明

当结果集在映射成实体对象的时候，需要调用实体对象的指定参数的构造器构造对象时，就可以使用该标签。

属性

无

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| idArg | 见idArg标签详解 | \* |
| arg | 见arg标签详解 | \* |

使用技巧

如果映射的实体对象类需要通过构造器来创建对象，就可以使用该标签。比如一个实体对象的某些属性是必须的，系统只提供了该对象指定参数的构造器，这时候就可以通过使用该标签来确定构造参数。

示例

|  |
| --- |
| <resultMap ...>  <constructor>  <idArg name=”id” column=”ID” jdbcType=”BIGINT”/>  <arg name=”name” column=”NAME” jdbcType=”VARCHAR”/>  ...  </constructor>  ...  </resultMap> |

## idArg标签

说明

该标签是constructor标签的子标签，用于映射一个一个实体的某个属性和对应的表字段，使用该标签与arg标签的不同之处在于标识当前字段为主键标识字段。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| name | 对应类实体的属性名 | 否 |
| column | 对应sql查询结果的列字段名 | 否 |
| jdbcType | 对应sql查询结果的列字段类型 | 否 |
| javaType | 对应类实体的属性类型 | 否 |
| typeHandler | sql查询结果的列字段与java类示例的属性转换器 | 否 |
| select | 对应类实体属性的查询链接(类属性为复杂对象) | 否 |
| resultMap | 对应类实体属性的结果映射链接(类属性为复杂对象) | 否 |

子标签

无

使用技巧

在使用constructor标签的时候，用于标识主键的列的映射

示例

|  |
| --- |
| <resultMap ...>  <constructor>  <idArg name=”id” column=”ID” jdbcType=”BIGINT”/>  </constructor>  ...  </resultMap> |

|  |
| --- |
| <resultMap ...>  <constructor>  <idArg name=”student” column=”STUDENT\_ID”  select=”selectStudentById”/>  </constructor>  </resultMap> |

|  |
| --- |
| <resultMap ...>  <constructor>  <idArg name=”student” column=”STUDENT\_ID”  resultMap=”StudentResultMap”/>  </constructor>  </resultMap> |

## arg标签

说明

arg标签和idArg标签的使用方法一样，只不过idArg带有一种标识的作用，而arg用于普通的属性字段的映射。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| name | 对应类实体的属性名 | 否 |
| column | 对应sql查询结果的列字段名 | 否 |
| jdbcType | 对应sql查询结果的列字段类型 | 否 |
| javaType | 对应类实体的属性类型 | 否 |
| typeHandler | sql查询结果的列字段与java类示例的属性转换器 | 否 |
| select | 对应类实体属性的查询链接(类属性为复杂对象) | 否 |
| resultMap | 对应类实体属性的结果映射链接(类属性为复杂对象) | 否 |

子标签

无

使用技巧

在使用constructor标签的时候，用于标识普通的列的映射

示例

|  |
| --- |
| <resultMap ...>  <constructor>  <arg name=”name” column=”NAME” jdbcType=”VARCHAR”/>  </constructor>  ...  </resultMap> |

|  |
| --- |
| <resultMap ...>  <constructor>  <arg name=”student” column=”STUDENT\_ID”  select=”selectStudentById”/>  </constructor>  </resultMap> |

|  |
| --- |
| <resultMap ...>  <constructor>  <arg name=”student” column=”STUDENT\_ID”  resultMap=”StudentResultMap”/>  </constructor>  </resultMap> |

## id标签

说明

用于resultMap下，标识主键的列与属性的映射

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| property | 对应类实体的属性名 | 否 |
| column | 对应sql查询结果的列字段名 | 否 |
| jdbcType | 对应sql查询结果的列字段类型 | 否 |
| javaType | 对应类实体的属性类型 | 否 |
| typeHandler | sql查询结果的列字段与java类示例的属性转换器 | 否 |

子标签

无

使用技巧

在resultMap下映射主键的列与属性

示例

|  |
| --- |
| <resultMap ...>  <id property=”id” column=”ID” jdbcType=”BIGINT”/>  ...  </resultMap> |

## result标签

说明

也是resultMap标签下用于映射列与属性的标签，与id标签不同之处在于，其映射只是用于普通的列与属性的映射。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| property | 对应类实体的属性名 | 否 |
| column | 对应sql查询结果的列字段名 | 否 |
| jdbcType | 对应sql查询结果的列字段类型 | 否 |
| javaType | 对应类实体的属性类型 | 否 |
| typeHandler | sql查询结果的列字段与java类示例的属性转换器 | 否 |

子标签

无

使用技巧

在resultMap下映射普通的列与属性

示例

|  |
| --- |
| <resultMap ...>  <result property=”name” column=”NAME” jdbcType=”VARCHAR”/>  ...  </resultMap> |

## associate标签

说明

用于resultMap映射关联的单个对象，属于mybatis的高级查询标签之一，主要用于处理业务系统中存在的实体对象之间一对一的关系，通过配置该标签，可以关联处理一对一的实体关系。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| property | 对应类实体的属性名 | 是 |
| column | 对应sql查询结果的列字段名 | 否 |
| jdbcType | 对应sql查询结果的列字段类型 | 否 |
| javaType | 对应类实体的属性类型 | 否 |
| typeHandler | sql查询结果的列字段与java类示例的属性转换器 | 否 |
| select | 对应类实体属性的查询链接(类属性为复杂对象) | 否 |
| resultMap | 对应类实体属性的结果映射链接(类属性为复杂对象) | 否 |
| notNullColumn |  | 否 |
| columnPrefix | 给每一列的列名加的前缀 | 否 |
| resultSet |  | 否 |
| foreignColumn |  | 否 |
| autoMapping | 是否开启自动映射  (true/false) | 否 |
| fetchType | 获取数据的类型  (lazy/eager) | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| constructor | 见constructor标签详解 | ? |
| id | 见id标签详解 | \* |
| result | 见result标签详解 | \* |
| associate | 见associate标签详解 | \* |
| collection | 见collection标签详解 | \* |
| discriminator | 见discriminator标签详解 | ? |

使用技巧

用在resultMap下面，通过select方式，引用其他的关联查询语句，在使用懒加载的时候，需要使用该方式。

用在resultMap下面，通过resulltMap方式，在查询的时候使用关联查询，然后通过该方式来映射结果。

示例

更详细示例请参照[项目](https://github.com/berg-turing/mybatis.git)路径: com.berg.base.xmlmappers.associate

|  |
| --- |
| <resultMap ...>  ...  <associate property=”dep” column=”DEP\_ID” select=”selectDepById”/>  </resultMap> |

|  |
| --- |
| <resultMap ...>  ...  <associate property=”dep” column=”DEP\_ID” columnPrefix=”d\_”  resultMap=”DepBaseResultMap”/>  </resultMap> |

## collection标签

说明

用于resultMap映射关联的多个对象，属于mybatis的高级查询标签之一，主要用于处理业务系统中存在的实体对象之间一对多的关系，通过配置该标签，可以关联处理一对多的实体关系。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| property | 对应类实体的属性名 | 是 |
| column | 对应sql查询结果的列字段名 | 否 |
| jdbcType | 对应sql查询结果的列字段类型 | 否 |
| ofType |  | 否 |
| javaType | 对应类实体的属性类型 | 否 |
| typeHandler | sql查询结果的列字段与java类示例的属性转换器 | 否 |
| select | 对应类实体属性的查询链接(类属性为复杂对象) | 否 |
| resultMap | 对应类实体属性的结果映射链接(类属性为复杂对象) | 否 |
| notNullColumn |  | 否 |
| columnPrefix | 为每一列的列名加的前缀 | 否 |
| resultSet |  | 否 |
| foreignColumn |  | 否 |
| autoMapping | 是否开启自动映射  (true/false) | 否 |
| fetchType | 获取数据的方式  (lazy/eager) | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| constructor | 见constructor标签详解 | ? |
| id | 见id标签详解 | \* |
| result | 见result标签详解 | \* |
| associate | 见associate标签详解 | \* |
| collection | 见collection标签详解 | \* |
| discriminator | 见discriminator标签详解 | ? |

使用技巧

该标签用于项目中映射关系为一对多的情况

示例

更详细示例请参照[项目](https://github.com/berg-turing/mybatis.git)路径: com.berg.base.xmlmappers.collection

|  |
| --- |
| <resultMap ...>  ...  <collection column=”STUDENT\_ID” property=”studentList”  jdbcType=”BIGINT” select=”selectStudentByDepartmentId”/>  </resultMap> |

## discriminator标签

说明

鉴别器discriminator最主要面临的场景是根据某些字段的值映射出不同的对象。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| column | 对应sql查询结果的列字段名 | 否 |
| jdbcType | 对应sql查询结果的列字段类型 | 否 |
| javaType | 将column转换成的java类型，用于与case标签的value进行比较 | 是 |
| typeHandler | sql查询结果的列字段与java类示例的属性转换器 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **约束** |
| case | 见case标签详解 | + |

使用技巧

在查询数据之后，需要根据查出信息的不同值创建不同的对象

示例

更详细示例请参照[项目](https://github.com/berg-turing/mybatis.git)路径: com.berg.base.xmlmappers.discriminator

|  |
| --- |
| <resultMap ...>  ...  <discriminator column=”SEX” javaType=”int” jdbcType=”INTEGER”>  <case value=”1” resultMap=”MaleResultMap”/>  <case value=”2” resultMap=”FemaleResultMap”/>  </discriminator>  </resultMap> |

|  |
| --- |
| <resultMap ...>  ...  <discriminator column=”sex” javaType=”int” jdbcType”INTEGER”>  <case value=”1” resultType=”Male”>  <associate property=”maleAttr” column=”ATTR\_ID”  jdbcType=”BIGINT” resultMap=”MaleAttrResultMap”/>  </case>  <case value=”2” resultType=”Female”>  <associate property=”femaleAttr” column=”ATTR\_ID”  jdbcType=”BIGINT” resultMap=”FemaleAttrResultMap”/>  </case>  </resultMap> |

## case标签

说明

case标签用于处理discriminator标签不同的分支，相当于java中switch语句中的case关键字。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| value | 对比的每一项的值，相当于switch语句的case后面接的值 | 是 |
| resultMap | 当符合该项时，对应使用的结果集 | 否 |
| resultType | 当符合该项时，对应使用的结果类型 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| constructor | 见constructor标签详解 | ? |
| id | 见id标签详解 | \* |
| result | 见result标签详解 | \* |
| associate | 见associate标签详解 | \* |
| collection | 见collection标签详解 | \* |
| discriminator | 见discriminator标签详解 | ? |

使用技巧

使用在discriminator标签下面，用于提供鉴别器针对不同的值执行不同的映射结果。

示例

|  |
| --- |
| <resultMap ...>  ...  <discriminator column=”sex” javaType=”int” jdbcType”INTEGER”>  <case value=”1” resultType=”Male”>  <associate property=”maleAttr” column=”ATTR\_ID”  jdbcType=”BIGINT” resultMap=”MaleAttrResultMap”/>  </case>  <case value=”2” resultType=”Female”>  <associate property=”femaleAttr” column=”ATTR\_ID”  jdbcType=”BIGINT” resultMap=”FemaleAttrResultMap”/>  </case>  </resultMap> |

## select标签

说明

是mybatis的mapper文件中，活动性最强的标签，mybatis也提供了对该标签的最大的支持。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| id | 该select语句的唯一标识，也是对应mapper接口的方法名 | 是 |
| parameterType | 传入的参数的类型 | 否 |
| parameterMap | 传入的参数的映射对象(基本已经废弃) | 否 |
| resultType | 查询结果的结果类型，如果是多个结果，只需要写集合存储的数据的类型，与resultMap不能同时存在 | 否 |
| resultMap | 使用resultMap来映射查询出来的每一行数据，与resultType不能同时存在 | 否 |
| resultSetType | 使用的结果集类型，如果没有配置，默认使用  (FORWARD\_ONLY/SCROLL\_INSENSITIVE/SCROLL\_SENSITIVE) | 否 |
| statementType | 在执行sql语句的时候，JDBC所使用的statement类型  (STATEMENT/PREPARED/CALLABLE) | 否 |
| fetchSize |  | 否 |
| timeout |  | 否 |
| flushCache | 是否刷新缓存，如果设置为true，则一级缓存和二级缓存都会被刷新  (true/false) | 否 |
| useCache | 是否使用缓存  (true/false) | 否 |
| databaseId | 数据库提供商的id，用于将sql语句指定给特定的数据库执行。比如说这里设置数据库提供商为mysql，那么这里的语句只会在mysql中执行，而在其他的数据库不会执行 | 否 |
| lang | 解析mapper文件时候的驱动，一般使用mybatis的默认设置，如果有自己定制的mapper文件解析驱动，可以在这里设置 | 否 |
| resultOrdered | (true/false) | 否 |
| resultSets |  | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

详情见项目内的代码

示例

|  |
| --- |
| <select id=”select” parameterType=”Dto” resultMap=”BaseResultMap”>  SELECT  <include refid=”BaseSql”>  <property name=”prefix” value=”t”/>  <property name=”suffix” value=” ”/>  </include>  FROM <include refid=”TableName”/> t  <where>  <include refid=”BaseCondition”>  <property name=”prefix” value=”t”/>  </include>  </where>  </select> |

## insert标签

说明

向数据库中插入数据。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| id | 该insert语句的唯一标识，也是对应mapper接口的方法名 | 是 |
| parameterType | 参数类型，也就是相应的mapper接口方法传入的参数，如果有多个参数，就可以不写 | 否 |
| parameterMap | 现在基本不用 | 否 |
| timeout |  | 否 |
| flushCache | 是否刷新缓存，如果设置为true，则一级缓存和二级缓存都会被刷新  (true/false) | 否 |
| statementType | 在执行sql语句的时候，JDBC所使用的statement类型  (STATEMENT/PREPARED/CALLABLE) | 否 |
| keyProperty | 主键的属性名称 | 否 |
| useGeneratedKeys | 是否使用主键自动生成，其实就是调用的数据库本身的自动生成，如果数据库在创建表的时候，没有对主键设置自增，那么这个属性值就没有什么作用，这个时候，可以使用selectKey标签来做定制化的设置  (true/false) | 否 |
| keyColumn | 数据库中主键列的名称 | 否 |
| databaseId | 数据库提供商的id，用于将sql语句指定给特定的数据库执行。比如说这里设置数据库提供商为mysql，那么这里的语句只会在mysql中执行，而在其他的数据库不会执行 | 否 |
| lang | 解析mapper文件时候的驱动，一般使用mybatis的默认设置，如果有自己定制的mapper文件解析驱动，可以在这里设置 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| selectKey | 见selectKey标签详情 | \* |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

详情见项目内代码

示例

|  |
| --- |
| <insert id="insert" parameterType="MaleAttr">  <selectKey keyProperty="id" keyColumn="ID"  resultType="long" order="BEFORE">  SELECT if(max(ID) IS NULL, 1, max(ID)+1)  NEW\_ID FROM <include refid="TableName"/>  </selectKey>  INSERT INTO  <include refid="TableName"/>  (  <trim suffixOverrides=",">  <if test="studentId!=null and studentId!=0">  STUDENT\_ID,  </if>  <if test="game!=null and !game.equals('')">  GAME  </if>  </trim>  )  VALUES  (  <trim suffixOverrides=",">  <if test="studentId!=null and studentId!=0">  #{studentId},  </if>  <if test="game!=null and !game.equals('')">  #{game}  </if>  </trim>  )  </insert> |

## selectKey标签

说明

自定义主键生成方式

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| resultType | 执行该标签内的sql语句之后，返回的数据的数据类型 | 否 |
| statementType | 在执行sql语句的时候，JDBC所使用的statement类型  (STATEMENT/PREPARED/CALLABLE) | 否 |
| keyProperty | 主键的属性名称 | 否 |
| keyColumn | 数据库中主键列的名称 | 否 |
| order | 执行语句的先后  (BEFORE/AFTER) | 否 |
| databaseId | 数据库提供商的id，用于将sql语句指定给特定的数据库执行。比如说这里设置数据库提供商为mysql，那么这里的语句只会在mysql中执行，而在其他的数据库不会执行 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

可以通过该方式来自定义主键的生成方式，可能是按照某种自定义的算法来生成主键。

示例

|  |
| --- |
| <insert ...>  <selectKey keyProperty=”id” keyColumn=”ID”  order=”BEFORE” resultType=”long”>  select if(max(ID) IS NULL, 1, max(ID) + 2) FROM tableName  </selectKey>  ...  </insert> |

## update标签

说明

更新语句块

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| id | 该update语句的唯一标识，也是对应mapper接口的方法名 | 是 |
| parameterType | 参数类型，也就是相应的mapper接口方法传入的参数，如果有多个参数，就可以不写 | 否 |
| parameterMap | 现在基本不用 | 否 |
| timeout |  | 否 |
| flushCache | 是否刷新缓存，如果设置为true，则一级缓存和二级缓存都会被刷新  (true/false) | 否 |
| statementType | 在执行sql语句的时候，JDBC所使用的statement类型  (STATEMENT/PREPARED/CALLABLE) | 否 |
| keyProperty | 主键的属性名称 | 否 |
| useGeneratedKeys | 是否使用主键自动生成，其实就是调用的数据库本身的自动生成，如果数据库在创建表的时候，没有对主键设置自增，那么这个属性值就没有什么作用，这个时候，可以使用selectKey标签来做定制化的设置  (true/false) | 否 |
| keyColumn | 数据库中主键列的名称 | 否 |
| databaseId | 数据库提供商的id，用于将sql语句指定给特定的数据库执行。比如说这里设置数据库提供商为mysql，那么这里的语句只会在mysql中执行，而在其他的数据库不会执行 | 否 |
| lang | 解析mapper文件时候的驱动，一般使用mybatis的默认设置，如果有自己定制的mapper文件解析驱动，可以在这里设置 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| selectKey | 见selectKey标签详情 | \* |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

详情见项目内代码

示例

|  |
| --- |
| <update id="updateByCondition">  <if test="condition!=null">  UPDATE <include refid="TableName"/>  <set>  <if test="maleAttr.studentId!=null and maleAttr.studentId!=0">  STUDENT\_ID = #{maleAttr.studentId},  </if>  <if test="maleAttr.game!=null and !maleAttr.game.equals('')">  GAME = #{maleAttr.game}  </if>  </set>  <where>  <if test="condition.id!=null and condition.id!=0">  ID = #{condition.id}  </if>  <if test="condition.studentId!=null and condition.studentId!=0">  AND STUDENT\_ID = #{condition.studentId}  </if>  <if test="condition.game!=null and !condition.game.equals('')">  AND GAME = #{condition.game}  </if>  </where>  </if>  </update> |

## delete标签

说明

删除语句块

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| id | 该delete语句的唯一标识，也是对应mapper接口的方法名 | 是 |
| parameterType | 参数类型，也就是相应的mapper接口方法传入的参数，如果有多个参数，就可以不写 | 否 |
| parameterMap | 现在基本不用 | 否 |
| timeout |  | 否 |
| flushCache | 是否刷新缓存，如果设置为true，则一级缓存和二级缓存都会被刷新  (true/false) | 否 |
| statementType | 在执行sql语句的时候，JDBC所使用的statement类型  (STATEMENT/PREPARED/CALLABLE) | 否 |
| databaseId | 数据库提供商的id，用于将sql语句指定给特定的数据库执行。比如说这里设置数据库提供商为mysql，那么这里的语句只会在mysql中执行，而在其他的数据库不会执行 | 否 |
| lang | 解析mapper文件时候的驱动，一般使用mybatis的默认设置，如果有自己定制的mapper文件解析驱动，可以在这里设置 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

详情见项目内代码

示例

|  |
| --- |
| <delete id="deleteById" parameterType="long">  DELETE FROM <include refid="TableName"/>  WHERE ID = #{id}  </delete> |

## sql标签

说明

sql标签主要用于书写一些通用的sql语句，用于其他标签内部引用，一般sql标签都会与include标签结合使用。

在sql标签内，可以使用${name}的方式进行占位，在include标签引用该sql标签对象的时候，可以在include内定义这些变量。

属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| id | sql标签的id，是该sql标签的唯一标识。 |
| databaseId | 数据库提供商的id，用于将sql语句指定给特定的数据库执行。比如说这里设置数据库提供商为mysql，那么这里的语句只会在mysql中执行，而在其他的数据库不会执行 |
| lang | 解析mapper文件时候的驱动，一般使用mybatis的默认设置，如果有自己定制的mapper文件解析驱动，可以在这里设置 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

1. mapper文件所对应的数据表的表名;
2. 定义查询语句需要查询的字段;
3. 查询语句的查询条件。

示例

|  |
| --- |
| <sql id=”TableName”>  test\_table\_name  </sql> |

|  |
| --- |
| <sql id=”BaseSql”>  ${prefix}.ID,  ${prefix}.NAME,  ${prefix}.DESCRIPTION  </sql> |

|  |
| --- |
| <sql id=”BaseCondition”>  <if test=”id!=null and id!=0”>  AND ${prefix}.ID = #{id}  </if>  <if test=”name!=null and !’’.equals(name)”>  AND ${prefix}.NAME like  CONCAT(‘%’, #{name, jdbcType=VARCHAR}, ‘%’)  </if>  <if test=”description!=null and !’’.equals(description)”>  AND ${prefix}.DESCRIPTION like  CONCAT(‘%’, #{description, jdbcType=VARCHAR}, ‘%’)  </if>  </sql> |

## include标签

说明

include标签主要用于引用sql标签，同时可以在include标签内设置一些参数，用于替换sql标签内部的占位符。

属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| refid | 引用的sql标签的id。 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| property | 用于给sql标签传递的参数 | + |

使用技巧

需要引用sql标签的地方，就需要使用include标签

1. select标签内部的查询的字段;
2. 需要使用表名的地方，通过引用的方式使用表名，可以预防表名变化;
3. select标签内部的查询条件部分

示例

|  |
| --- |
| <select ...>  SELECT  <include refid=”BaseSql”>  <property name=”prefix” value=”t”/>  </include>  FROM  <include refid=”TableName”/> t  <where>  <include refid=”BaseCondition”>  <property name=”prefix” value=”t”/>  </include>  </where>  </select> |

## property标签

说明

用于include标签下，给include所引用的sql标签内传递一定的参数

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| name | 属性名 | 是 |
| value | 属性值 | 是 |

子标签

无

使用技巧

详情见项目内代码

示例

|  |
| --- |
| <include refid=”BaseSql”>  <property name=”prefix” value=”t”/>  <property name=”suffix” value=” ”/>  </include> |

## bind标签

说明

用于写一些复杂的语句

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| name | 属性名 | 是 |
| value | 属性值 | 是 |

子标签

无

使用技巧

示例

|  |
| --- |
| <bind name=”” |

## if标签

说明

条件判断语句

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| test | 判断条件 | 是 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

动态sql的基础，一般通过该标签与实际的条件来动态的生成sql语句

示例

|  |
| --- |
| <if test=”id!=null and id!=0”>  ID = #{id}  </if> |

|  |
| --- |
| <if test=”name!=null and !name.equals(‘’)”>  NAME like CONCAT(‘%’, #{name, jdbcType=VARCHAR}, ‘%’)  </if> |

## where标签

说明

是为了处理使用WHERE条件语句时，使用if标签条件生成的条件语句出现多余的AND或者OR而设置的语句。

比如：

|  |
| --- |
| WHERE  <if test=”id!=null and id!=0”>  ID = #{id}  </if>  <if test=”name!=null and !name.equals(‘’)”>  AND NAME like CONCAT(‘%’, #{name, jdbcType=VARCHAR}, ‘%’)  </if> |

当传入的参数都存在或者只传入了id值不会出错

当没有传值的时候，就生成下面的语句

|  |
| --- |
| WHERE |

或者只传入了name只，就生成了下面的语句

|  |
| --- |
| WHERE  AND NAME like CONCAT(‘%’, ‘name’, ‘%’) |

这两种情况都是会导致sql语句执行出错的，上面的问题可以使用下面的方法解决：

|  |
| --- |
| WHERE  1=1  <if test=”id!=null and id!=0”>  AND ID = #{id}  </if>  <if test=”name!=null and !name.equals(‘’)”>  AND NAME like CONCAT(‘%’, #{name, jdbcType=VARCHAR}, ‘%’)  </if> |

但是这种方法及其不好看，还感觉那个1=1有点碍眼，我们可以使用下面的方法来代替上面的方法

|  |
| --- |
| <where>  <if test=”id!=null and id!=0”>  AND ID = #{id}  </if>  <if test=”name!=null and !name.equals(‘’)”>  AND NAME like CONCAT(‘%’, #{name, jdbcType=VARCHAR}, ‘%’)  </if>  </where> |

where标签的作用就是，当where标签下没有任何内容的时候，就不输出任何东西。

如果where标签下有内容的时候，就会在那些内容前面加一个WHERE，并且会去掉内部内容最开始的AND和OR字段。

上面的方法与下面的方法等同

|  |
| --- |
| <trim prefix=”WHERE” prefixOverrides=”AND |OR ”>  <if test=”id!=null and id!=0”>  AND ID = #{id}  </if>  <if test=”name!=null and !name.equals(‘’)”>  AND NAME like CONCAT(‘%’, #{name, jdbcType=VARCHAR}, ‘%’)  </if>  </trim> |

上面的代码与使用where标签的效果是一样的，但是要注意prefixOverrides属性内的值的写法，内部的内容是”AND |OR ”。

注意：AND和OR后面都有一个空格

属性

无

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

见说明

示例

见说明

## set标签

说明

set标签和where标签的目地是一样的，也是为了解决一些现有动态sql语句存在的异常。

set标签解决的是SET语句中出现的异常。如下：

|  |
| --- |
| UPDATE tableName  SET  <if test=”age!=null and age!=0”>  AGE =#{age},  </if>  <if test=”name!=null and !name.equals(‘’)”>  NAME = #{name}  </if>  WHERE  ID = #{id} |

当没有参数的时候，会生成下面的语句

|  |
| --- |
| UPDATE tableName  SET |

当传入的参数只有age而没有name的时候，就会生成下面的语句

|  |
| --- |
| UPDATE tableName  SET  AGE = 12,  WHERE  ID = 1 |

在AGE = 12后面就多了一个逗号，这样sql语句就会报错

我么使用set标签对该语句进行一定的改造

|  |
| --- |
| <set>  <if test=”age!=null and age!=0”>  AGE =#{age},  </if>  <if test=”name!=null and !name.equals(‘’)”>  NAME = #{name}  </if>  </set> |

set标签会判断内部有无内容，如果没有内容，那么set标签就不会做处理，如果有内容，set标签就是在内部内容前面加上SET，并去除内部内容最后的逗号(,)。

下面的语句同set标签的作用

|  |
| --- |
| <trim prefix=”SET” suffixOverrides=”,”>  <if test=”age!=null and age!=0”>  AGE =#{age},  </if>  <if test=”name!=null and !name.equals(‘’)”>  NAME = #{name}  </if>  </trim> |

属性

无

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

见说明

示例

见说明

## foreach标签

说明

主要用于遍历一个集合内的数据

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| collection | 处理的集合对象 | 是 |
| item | 集合中的每一项的名字 | 否 |
| index | 每一项索引的名字 | 否 |
| open | 内部内容以什么内容开头 | 否 |
| close | 内部内容以什么内容结尾 | 否 |
| separator | 内部每一项内容以什么分割 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

主要用于对集合的遍历

示例

|  |
| --- |
| <insert ...>  INSERT INTO tableName  (SEX, NAME)  VALUES  <foreach collection=”list” item=”item” close=”;” separator=”,”>  (#{item.sex}, #{item.name})  </foreach>  </insert> |

|  |
| --- |
| <select ...>  SELECT ID, SX, NAME  FROM tableName  WHERE  ID IN  <foreach collection=”list” item=”item”  open=”(” close=”)” separator=”,”>  #{item}  </foreach>  </select> |

## trim标签

说明

trim语句是一个自由度很高的标签，上面展示了如何使用trim来代替where标签和set标签。

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| prefix | 增加的前缀 | 否 |
| prefixOverrides | 需要去掉的前缀 | 否 |
| suffix | 增加的后缀 | 否 |
| suffixOverrieds | 需要去掉的后缀 | 否 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

替代where标签

替代set标签

示例

见where与set标签

## choose标签

说明

选择执行的标签，代码会按照choose标签下第的when标签的顺序执行，只要执行到满足条件的when就使用该when标签下的语句，如果没有when标签满足条件，就会使用otherwise中的语句，和java内的switch语句相像，主要语句与逻辑体现在when和otherwise中。

属性

无

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| when | 见when标签详解 | \* |
| otherwise | 见otherwise标签详解 | ? |

使用技巧

当需要根据当某些状态不同时，执行的查询条件也不相同，就可以使用该标签。

示例

|  |
| --- |
| <choose>  <when test=”sex==1”>  ...  </when>  <otherwise>  ...  </otherwise>  </choose> |

## when标签

说明

用于处理具体满足条件的语句

属性

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **是否必须** |
| test | 判断条件 | 是 |

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

和if标签结合使用，可以产生很复杂的查询语句

示例

|  |
| --- |
| <choose>  <when test=”id!=null and id!=0”>  ID = #{id}  </when>  <when test=”name!=null and !name.equals(‘’)”>  NAME = #{name}  </when>  <otherwise>  NAME like CONCAT(‘BERG’, ‘%’)  </otherwise>  </choose> |

## otherwise标签

说明

choose标签的默认分支，如果没有when标签满足条件，就会执行otherwise标签。

属性

无

子标签

|  |  |  |
| --- | --- | --- |
| **标签名** | **描述** | **约束** |
| include | 见include标签详情 | \* |
| trim | 见trim标签详情 | \* |
| where | 见where标签详情 | \* |
| set | 见set标签详情 | \* |
| foreach | 见foreach标签详情 | \* |
| choose | 见choose标签详情 | \* |
| if | 见if标签详情 | \* |
| bind | 将bind标签详情 | \* |

使用技巧

无

示例

|  |
| --- |
| <choose>  <when test=”id!=null and id!=0”>  ID = #{id}  </when>  <when test=”name!=null and !name.equals(‘’)”>  NAME = #{name}  </when>  <otherwise>  NAME like CONCAT(‘BERG’, ‘%’)  </otherwise>  </choose> |

# 应用

## 方法一

说明：

该方法是最符合MyBatis规范的，在映射文件中，即保证了各个对象之间的相互独立性，又保证了整个系统的可扩展性。但是这种方式会导致数据的冗余与N+1的问题的出现。

什么是N+1问题？

概括地讲,N+1 查询问题可以是这样引起的:

你执行了一个单独的 SQL 语句来获取结果列表(就是“+1”)。

对返回的每条记录,你执行了一个查询语句来为每个加载细节(就是“N”)。

这个问题会导致成百上千的 SQL 语句被执行。这通常不是期望的。

书写方法：

MyDto.class

|  |
| --- |
| public class MyDto{  /\*\*  \* id  \*/  private Long id;  /\*\*  \* 单个关联对象id  \*/  private Long extendsOneId;  /\*\*  \* 名称  \*/  private String name;  ...  /\*\*  \* 关联的单个对象  \*/  private ExtendsOne extendsOne;  /\*\*  \* 关联的多个对象  \*/  private List<ExtendsTwo> extendsTwos;  //setter和getter方法  ...  } |

MyMapper.class

|  |
| --- |
| public interface MyMapper{  /\*\*  \* 根据id查找一个实体对象  \* @param id 实体的id  \* @return 查找到的实体对象，没有找到就返回null  \*/  public MyDto selectOne(Long id);  /\*\*  \* 根据条件查询满足条件的实体对象  \* @param myDto 条件对象  \* @return 查找到的实体对象，如果没有找到就返回一个size为0的list  \*/  public List<MyDto> select(MyDto myDto);  /\*\*  \* 根据条件查询满足条件的实体对象(包括关联的对象)  \* @param myDto 条件对象  \* @return 查找到的实体对象，如果没有找到就返回一个size为0的list  \*/  public List<MyDto> selectWithExtends(MyDto myDto);  } |

MyMapper.xml

|  |
| --- |
| <?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.berg.xxx.mapper.MyMapper">  <!-- 当前实体对象所对应的数据库表的表名 -->  <sql id=”TableName”>  MyTableName  </sql>  <!-- 基本的结果映射 -->  <resultMap id=”BaseResultMap” type=”com.berg.xxx.dto.MyDto”>  <id property=”id” column=”ID” jdbcType=”BIGINT”/>  <result property=”extendsOneId” column=”EXTENDS\_ONE\_ID”  jdbcType=”BIGINT”/>  <result property=”name” column=”NAME” jdbcType=”VARCHAR”/>  ...  </resultMap>  <!-- 带关联对象的结果映射 -->  <resultMap id=”ExtendsResultMap” type=”com.berg.xxx.dto.MyDto”  extends=”com.berg.xxx.mapper.MyMapper.BaseResultMap”>  <association property=”extendsOne” column=”EXTENDS\_ONE\_ID”  select=”com.berg.xxx.mapper.ExtendsOneMapper.selectOne”/>  <collection property=”extendsTwos” column=”ID”  ofType=”com.berg.xxx.dto.ExtendTwo”  select=”com.berg.xxx.mapper.ExtendsTwoMapper.selectMore”/>  </resultMap>  <!-- 基本的属性字段 -->  <sql id=”BaseSql”>  ${prefix}.ID,  ${prefix}.EXTENDS\_ONE\_ID,  ${prefix}.NAME  </sql>    <!-- 基本查询条件 -->  <sql id=”BaseCondition”>  <if test=”id!=null and id!=0”>  AND ${prefix}.ID = #{id}  </if>  <if test=”extendsOneId!=null and extendsOneId!=0”>  AND ${prefix}.EXTENDS\_ONE\_ID = #{extendsOneId}  </if>  <if test=”name!=null and !name.equals(‘’)”>  AND ${prefix}.NAME like  CONCAT(‘%’, CONCAT(#{name, jdbcType=VARCHAR}, ‘%’))  </if>  ...  </sql>  <!-- 按照id查询实体对象 -->  <select id=”selectOne” paramterType=”long”  resultMap=”com.berg.xxx.MyMapper.BaseResultMap”>  SELECT  <include refid=”com.berg.xxx.MyMapper.BaseSql”>  <property name=”prefix” value=”m”/>  </include>  FROM  <include refid=”com.berg.xxx.MyMapper.TableName”/> m  WHERE  m.ID = #{id}  </select>  <!-- 按照条件查询实体对象 -->  <select id=”select” paramterType=”long”  resultMap=”com.berg.xxx.MyMapper.BaseResultMap”>  SELECT  <include refid=”com.berg.xxx.MyMapper.BaseSql”>  <property name=”prefix” value=”m”/>  </include>  FROM  <include refid=”com.berg.xxx.MyMapper.TableName”/> m  <trim prefix=”WHERE” prefixOverrides=”AND |OR ”>  <include refid=”com.berg.xxx.MyMapper.BaseCondition”>  <property name=”prefix” value=”m”/>  </include>  </trim>  </select>  <!-- 按照条件查询实体对象(带关联对象) -->  <select id=”selectWithExtends” paramterType=”long”  resultMap=”com.berg.xxx.MyMapper.ExtendsResultMap”>  SELECT  <include refid=”com.berg.xxx.MyMapper.BaseSql”>  <property name=”prefix” value=”m”/>  </include>  FROM  <include refid=”com.berg.xxx.MyMapper.TableName”/> m  <trim prefix=”WHERE” prefixOverrides=”AND |OR ”>  <include refid=”com.berg.xxx.MyMapper.BaseCondition”>  <property name=”prefix” value=”m”/>  </include>  </trim>  </select>  </mapper> |

## 方法二

说明：该方法主要是用于解决方法一中存在的N+1的问题，通过另外一种对数据的组装与关联处理的方式，解决了N+1的问题，但是该方法书写时较为繁琐，基本满足规范。

书写方式：

MyDto.class 同方法一

MyMapper.class

|  |
| --- |
| public interface MyMapper{  /\*\*  \* 根据条件查询满足条件的实体对象  \* @param myDto 条件对象  \* @return 查找到的实体对象，如果没有找到就返回一个size为0的list  \*/  public List<MyDto> select(MyDto myDto);  /\*\*  \* 根据条件查询满足条件的实体对象(包括关联的对象)  \* @param myDto 条件对象  \* @return 查找到的实体对象，如果没有找到就返回一个size为0的list  \*/  public List<MyDto> selectWithExtends(MyDto myDto);  } |

在方法二中，不需要selectOne这个方法。

MyMapper.xml

|  |
| --- |
| <?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.berg.xxx.mapper.MyMapper">  <!-- 当前实体对象所对应的数据库表的表名 -->  <sql id=”TableName”>  MyTableName  </sql>  <!-- 基本的结果映射 -->  <resultMap id=”BaseResultMap” type=”com.berg.xxx.dto.MyDto”>  <id property=”id” column=”ID” jdbcType=”BIGINT”/>  <result property=”extendsOneId” column=”EXTENDS\_ONE\_ID”  jdbcType=”BIGINT”/>  <result property=”name” column=”NAME” jdbcType=”VARCHAR”/>  ...  </resultMap>  <!-- 带关联对象的结果映射 -->  <resultMap id=”ExtendsResultMap” type=”com.berg.xxx.dto.MyDto”  extends=”com.berg.xxx.mapper.MyMapper.BaseResultMap”>  <association property=”extendsOne” column=”EXTENDS\_ONE\_ID”  columnPrefix=”o\_”  resultMap=  ”com.berg.xxx.mapper.ExtendsOneMapper.BaseResultMap”/>  <collection property=”extendsTwos” column=”ID” columnPrefix=”t\_”  ofType=”com.berg.xxx.dto.ExtendTwo”  resultMap=  ”com.berg.xxx.mapper.ExtendsTwoMapper.BaseResultMap”/>  </resultMap>  <!-- 基本的属性字段 -->  <sql id=”BaseSql”>  ${prefix}.ID,  ${prefix}.EXTENDS\_ONE\_ID,  ${prefix}.NAME  </sql>  <!-- 扩展的sql -->  <sql id=”ExtendsSql”>  ${prefix}.ID ${suffix}\_ID,  ${prefix}.EXTENDS\_ONE\_ID ${suffix}\_EXTENDS\_ONE\_ID,  ${prefix}.NAME ${suffix}\_NAME  </sql>  <!-- 基本查询条件 -->  <sql id=”BaseCondition”>  <if test=”id!=null and id!=0”>  AND ${prefix}.ID = #{id}  </if>  <if test=”extendsOneId!=null and extendsOneId!=0”>  AND ${prefix}.EXTENDS\_ONE\_ID = #{extendsOneId}  </if>  <if test=”name!=null and !name.equals(‘’)”>  AND ${prefix}.NAME like  CONCAT(‘%’, CONCAT(#{name, jdbcType=VARCHAR}, ‘%’))  </if>  ...  </sql>    <!-- 按照条件查询实体对象 -->  <select id=”select” paramterType=”long”  resultMap=”com.berg.xxx.MyMapper.BaseResultMap”>  SELECT  <include refid=”com.berg.xxx.MyMapper.BaseSql”>  <property name=”prefix” value=”m”/>  </include>  FROM  <include refid=”com.berg.xxx.MyMapper.TableName”/> m  <trim prefix=”WHERE” prefixOverrides=”AND |OR ”>  <include refid=”com.berg.xxx.MyMapper.BaseCondition”>  <property name=”prefix” value=”m”/>  </include>  </trim>  </select>  <!-- 按照条件查询实体对象(带关联对象) -->  <select id=”selectWithExtends” paramterType=”long”  resultMap=”com.berg.xxx.MyMapper.ExtendsResultMap”>  SELECT  <include refid=”com.berg.xxx.MyMapper.BaseSql”>  <property name=”prefix” value=”m”/>  </include>,  <include refid=”com.berg.xxx.ExtendsOneMapper.BaseSql”>  <property name=”prefix” value=”o”/>  <property name=”suffix” value=”o”/>  </include>,  <include refid=”com.berg.xxx.ExtendsTwoMapper.BaseSql”>  <property name=”prefix” value=”t”/>  <property name=”suffix” value=”t”/>  </include>  FROM  <include refid=”com.berg.xxx.MyMapper.TableName”/> m  LEFT JOIN  <include refid=”com.berg.xxx.ExtendsOneMapper.TableName”/> o  ON m.EXTENDS\_ONE\_ID = o.EXTENDS\_ONE\_ID  LEFT JOIN  <include refid=”com.berg.xxx.ExtendsTwoMapper.TableName”/> t  ON m.ID = t.MY\_DTO\_ID  <trim prefix=”WHERE” prefixOverrides=”AND |OR ”>  <include refid=”com.berg.xxx.MyMapper.BaseCondition”>  <property name=”prefix” value=”m”/>  </include>  </trim>  </select>  </mapper> |

## 方法三

说明：方法三与前两种方法具有本质上的不同，是一个阉割版本，但是更加适合我们现在的开发方式。不满足规范，扩展性一般。

书写方式：

MyDto.class

|  |
| --- |
| public class MyDto{  /\*\*  \* id  \*/  private Long id;  /\*\*  \* 单个关联对象id  \*/  private Long extendsOneId;  /\*\*  \* 名称  \*/  private String name;  ...  /\*\*  \* ExtendsOne的名称  \*/  private String extendsOneName;  //setter和getter方法  ...  } |

MyMapper.class 同方法二中的mapper

MyMapper.xml

|  |
| --- |
| <?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.berg.xxx.mapper.MyMapper">  <!-- 当前实体对象所对应的数据库表的表名 -->  <sql id=”TableName”>  MyTableName  </sql>  <!-- 基本的结果映射 -->  <resultMap id=”BaseResultMap” type=”com.berg.xxx.dto.MyDto”>  <id property=”id” column=”ID” jdbcType=”BIGINT”/>  <result property=”extendsOneId” column=”EXTENDS\_ONE\_ID”  jdbcType=”BIGINT”/>  <result property=”name” column=”NAME” jdbcType=”VARCHAR”/>  ...  </resultMap>  <!-- 带关联属性的结果映射 -->  <resultMap id=”ExtendsResultMap” type=”com.berg.xxx.dto.MyDto”  extends=”com.berg.xxx.mapper.MyMapper.BaseResultMap”>  <result property=”extendsOneName” jdbcType=”VARCHAR”  column=”EXTENDS\_ONE\_NAME”/>  </resultMap>  <!-- 基本的属性字段 -->  <sql id=”BaseSql”>  ${prefix}.ID,  ${prefix}.EXTENDS\_ONE\_ID,  ${prefix}.NAME  </sql>  <!-- 扩展的sql -->  <sql id=”ExtendsSql”>  ${prefixA}.EXTENDS\_ONE\_NAME EXTENDS\_ONE\_NAME  </sql>  <!-- 基本查询条件 -->  <sql id=”BaseCondition”>  <if test=”id!=null and id!=0”>  AND ${prefix}.ID = #{id}  </if>  <if test=”extendsOneId!=null and extendsOneId!=0”>  AND ${prefix}.EXTENDS\_ONE\_ID = #{extendsOneId}  </if>  <if test=”name!=null and !name.equals(‘’)”>  AND ${prefix}.NAME like  CONCAT(‘%’, CONCAT(#{name, jdbcType=VARCHAR}, ‘%’))  </if>  ...  </sql>  <!-- 扩展的查询条件 -->  <sql id=”ExtendsCondition”>  <if test=”extendsOneName!=null and !extendsOneName.equals(‘’)”>  AND ${prefixA}.EXTENDS\_ONE\_NAME like  CONCAT(‘%’,  CONCAT(#{extendsOneName, jdbcType=VARCHAR}, ‘%’))  </if>  </sql>    <!-- 按照条件查询实体对象 -->  <select id=”select” paramterType=”long”  resultMap=”com.berg.xxx.MyMapper.BaseResultMap”>  SELECT  <include refid=”com.berg.xxx.MyMapper.BaseSql”>  <property name=”prefix” value=”m”/>  </include>  FROM  <include refid=”com.berg.xxx.MyMapper.TableName”/> m  <trim prefix=”WHERE” prefixOverrides=”AND |OR ”>  <include refid=”com.berg.xxx.MyMapper.BaseCondition”>  <property name=”prefix” value=”m”/>  </include>  </trim>  </select>  <!-- 按照条件查询实体对象(带关联对象) -->  <select id=”selectWithExtends” paramterType=”long”  resultMap=”com.berg.xxx.MyMapper.ExtendsResultMap”>  SELECT  <include refid=”com.berg.xxx.MyMapper.BaseSql”>  <property name=”prefix” value=”m”/>  </include>,  <include refid=”com.berg.xxx.MyMapper.ExtendsSql”>  <property name=”prefixA” value=”o”/>  </include>  FROM  <include refid=”com.berg.xxx.MyMapper.TableName”/> m  LEFT JOIN  <include refid=”com.berg.xxx.ExtendsOneMapper.TableName”/> o  ON m.EXTENDS\_ONE\_ID = o.EXTENDS\_ONE\_ID  <trim prefix=”WHERE” prefixOverrides=”AND |OR ”>  <include refid=”com.berg.xxx.MyMapper.BaseCondition”>  <property name=”prefix” value=”m”/>  </include>  <include refid=”com.berg.xxx.MyMapper.ExtendsCondition”>  <property name=”prefixA” value=”o”/>  </include>  </trim>  </select>  </mapper> |