# 什么是Mybatis

ORM框架(Object Relation Mapping),是一个半自动化的持久化框架。

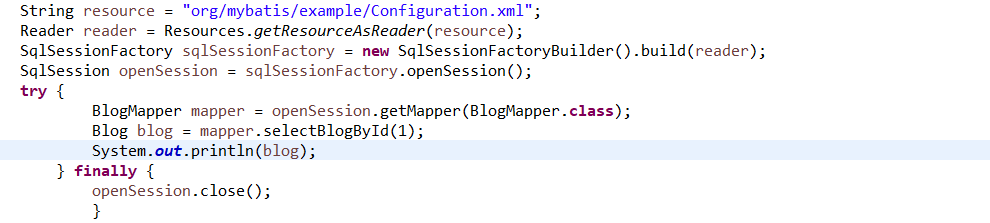
# MYBATIC和Hibernate的区别(为什么使用Mybatis)

Hibernate是全自动的ORM框架，旨在消除SQL(黑箱操作),HQL;

MYbatis是半自动的ORM框架，把编写SQL的权利给了开发人员

# Mybatis Hello word

通过SqlSessionFactory对象获取SqlSession(执行增删改查)



**接口编程**： Mapper接口没有实现类,MYBTIS为接口创建一个代理对象，代理对象去执行增删改查的方法;

**总结**：

SqlSession代表和数据库的一次会话,用完必须关闭。

SqlSession和Connection一样是线程不安全的,每次使用都应该获取新的对象。

Mapper接口没有实现类,Mybatis会为这个接口实现一个代理对象。

MYbatis中重要的配置文件

全局配置文件(可以没有)

SQL映射文件(必须)

# 全局配置文件

## dtd约束

## 4.2properties

1.Mybatis可以使用properties来引入外部properties配置文件的内容。

resource:引入类路径下的资源

url:引入网络路径或者磁盘路径下的资源

<properties resource="org/mybatis/example/config.properties">

<property name="username" value="dev\_user"/>

<property name="password" value="F2Fa3!33TYyg"/>

</properties>

## 4.3settings

设置mybatis 全局属性

<settings>

<setting name="cacheEnabled" value="true"/>

<setting name="lazyLoadingEnabled" value="true"/>

</settings>

## 4.4typeAliases

1.typeAlias：为某个java类型起别名

type:指定要器别名的类型全类名,默认别名就是类名小写；

alias:指定新的别名

<typeAliases>

<typeAlias alias="Author" type="domain.blog.Author"/>

<typeAlias alias="Blog" type="domain.blog.Blog"/>

</typeAliases>

2.pagkage：为某个包下的所有类批量起别名

name:指定包名（当前包及后代包的每一个类都起一个默认的别名（类名小写））

3.批量起名的情况下，使用@Alias注解为某个类型指定新的别名；

<typeAliases>

<package name="domain.blog"/>

</typeAliases>

## 4.5typeHandlers类型处理器

## 4.6plugins 插件（拦截、动态代理）

## 4.7environment

## 4.8 databaseIdProvider

支持多数据库

1.配置databaseIdProvider

2.在SQL上配置databaseId

<databaseIdProvider type="DB\_VENDOR">

<property name="SQL Server" value="sqlserver"/>

<property name="DB2" value="db2"/>

<property name="Oracle" value="oracle" />

</databaseIdProvider>

## 4.9mappers

将SQL映射注册到全局配置中

<mappers>

<mapper resource="org/mybatis/builder/\*.xml"/>

</mappers>

# MYbatis映射文件

## insert, update and delete

1.Mybatis允许增删改直接定义以下类型返回值

Integer、Long、Boolean、void

2.Mysql支持自增主键，自增主键获取，MYbatis利用statement.getGnereatedKeys();

useGeneratedKeys =true：使用自增主键获取主键策略

keyProperty：指定对应的主键属性,也就是Mybatis获取主键以后，将这个值封装给JavaBean。

<insert id="insertAuthor" useGeneratedKeys="true"

keyProperty="id">

insert into Author (username,password,email,bio)

values (#{username},#{password},#{email},#{bio})

</insert>

Oracle不支持自增，Oracle使用序列来模拟自增；

每次插入的数据的主键是从序列中拿到的值；如何获取到这个值？

通过编写selectkey标签查询主键

keProperty：查出的主键封装给javaBean的那个属性；

order="BEFORE":当前SQL在插入SQL之前运行

AFTER:当前SQL在插入SQL之后运行

resultType：查出的数据的返回类型

<insert id="insertAuthor">

<selectKey keyProperty="id" resultType="int" order="BEFORE">

select seq\_author.nextVal from dual

</selectKey>

insert into Author

(id, username, password, email,bio, favourite\_section)

values

(#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})

</insert>

或者

<insert id="insertAuthor">

insert into Author

(id, username, password, email,bio, favourite\_section)

values

( seq\_author.nextVal, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})

</insert>

## Mybatis参数处理

## 单个参数

单个参数时,Mybatis不会做特殊处理，

#{参数},取出参数值

## 多个参数

多个参数：BindingException

Mybatis会做特殊处理。多个参数会被封装为一个map,#{}就是从map中取指定的key值，key：param1...paramN 或参数索引,Value：传入的值;

命名参数：明确指定封装参数时的key,使用@Param("id")注解;

多个参数会被封装成一个map，key使用@Param注解指定的值, Value：传入的参数值, #｛指定的key｝取出对应的参数值;

## pojo:

如果多个参数正好是业务逻辑模型，直接出传入pojo;

#{属性名}:取出传入的pojo的属性值;

## Map：

如果多个参数不是业务逻辑模型中的数据，没有对应的pojo,但是不常使用，为了方便可以传入map;

#{key}:取出传入的Map的值;

## TO：

如果多个参数不是业务逻辑模型中的数据，但是经常要使用， 推荐编写一个TO（Transfer Object）数据传输对象;

Page｛

int index，

int size

｝

**如果是Collection类型的或者数组，也会特殊处理。**

**把传入的List或数组封装到Map中**

# Mybatis参数处理源码

public Object getNamedParams(Object[] args) {

final int paramCount = names.size();

//如果参数为空直接返回

if (args == null || paramCount == 0) {

return null;

//如果只有一个元素并且是没有Param注解，就直接调用arg[0]，单元素直接返回

} else if (!hasParamAnnotation && paramCount == 1) {

return args[names.firstKey()];

} else {

//多个元素或者有Param标注

final Map<String, Object> param = new ParamMap<Object>();

int i = 0;

//遍历names集合：names的values作为key；

//names集合的key作为取值;

for (Map.Entry<Integer, String> entry : names.entrySet()) {

param.put(entry.getValue(), args[entry.getKey()]);

//额外的将每一个参数也保存在map中,使用新的key(param1...paramN)

// add generic param names (param1, param2, ...)

final String genericParamName = GENERIC\_NAME\_PREFIX + String.valueOf(i + 1);

// ensure not to overwrite parameter named with @Param

if (!names.containsValue(genericParamName)) {

param.put(genericParamName, args[entry.getKey()]);

}

i++;

}

return param;

}

}

public ParamNameResolver(Configuration config, Method method) {

//1首先拿到所有的参数类型、注解和标注参数的索引

//1.1获取方法参数列表类型

final Class<?>[] paramTypes = method.getParameterTypes();

//1.2获取方法参数的注解

final Annotation[][] paramAnnotations = method.getParameterAnnotations();

//1.3标注参数的索引

final SortedMap<Integer, String> map = new TreeMap<Integer, String>();

int paramCount = paramAnnotations.length;

// get names from @Param annotations

for (int paramIndex = 0; paramIndex < paramCount; paramIndex++) {

if (isSpecialParameter(paramTypes[paramIndex])) {

// skip special parameters

continue;

}

String name = null;

//如果当前参数的注解是param注解就获取注解的值

for (Annotation annotation : paramAnnotations[paramIndex]) {

if (annotation instanceof Param) {

hasParamAnnotation = true;

name = ((Param) annotation).value();

break;

}

}

if (name == null) {

// @Param was not specified.

if (config.isUseActualParamName()) {

name = getActualParamName(method, paramIndex);

}

if (name == null) {

// use the parameter index as the name ("0", "1", ...)

// gcode issue #71

name = String.valueOf(map.size());

}

}

map.put(paramIndex, name);

}

names = Collections.unmodifiableSortedMap(map);

}

# 7.#和$区别

#{}:可以获取map中的值或pojo对象属性值;

${}:可以获取map中的值或pojo对象属性值;

区别：

#{}:是以预编译的形式，将参数设置到SQL语句中:PreparedStatement；防止SQL注入

${}:取出的值直接拼接在SQL语句中;会有安全问题

大多情况下使用#｛｝

当原生的jdbc不支持占位符的地方我们可以使用${}进行取值

分表:按照年份分分表拆分

select \* from $(year)\_salary where ....

排序：

seelct \* from emp order by ${col} ${order}

# 8.#{}丰富的用法

1. 规定参数的一些规则：

javaType、jdbcType、mode(存储过程)、unmericScale、

resultMap、typeHandler、jdbcTypeName、expression

## jdbcType

在我们数据为null的时候，有一些数据库可能不能识别Mybatis对null的默认处理，比如Oracle(报错); 需要告诉Mybatis jdbcType的类型;

Oracle JdbcType OTHER:无效的类型;因为mybatis对所有的null都映射的是jdbc原生的OTHER类型，Oracle不能正确处理;

解决方案：

1.#｛phone，jdbcType=null｝

2.修改全局配置文件jdbcTypeForNull

由于全局配置中的jdbcTypeForNull =OTHER ，ORACLE不支持；

# Select

## 返回Map

返回一条记录的map，key为列名 ，值为对应的值;

多条记录封装一个map,Map<Integer,JavaBean> 键是这条记录的主键,值为记录封装后的javaBean。

设置resulttype="javaBean"，在Mapper接口上使用@MapKey("id")告诉Mybatis封装的时候使用哪个属性作为map的key。

## 返回resultMap

Tip：在resultMap没有指定的列会自动封装;

## 场景一：查询员工时查询部门信息

### 联合查询：级联属性封装结果集

<resultMap id="userResultMap" type="com.itour.model.Emp">

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

<result property="lastName" column="LAST\_NAME"/>

<result property="dept.did" column="DID"/>

<result property="dept.dname" column="DNAME"/>

</resultMap>

### 嵌套结果集的方式，通过association指定联合对象

**property="dept"指定哪个属性是联合对象**

**javaType:指定属性对象的类型，不可省略；**

<resultMap id="userResultMap" type="com.itour.model.Emp">

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

<result property="lastName" column="LAST\_NAME"/>

<association property="dept" javaType="com.itour.model.Dept">

<id property="did" column="DID" />

<result property="dname" column="DNAME"/>

</associatin>

</resultMap>

### 使用association分步查询

1.先按照员工id查询员工信息

2.根据部门id查询部门信息

3.将部门信息封装到员工中

select:表明属性是调用select指定的方法查出的结果

column：指定将那一列的值传给这个方法

<resultMap id="userResultMap" type="com.iour.model.Emp">

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

<result property="lastName" column="LAST\_NAME"/>

<association property="dept" select="com.itour.DeptMapper.selectOne" column= "did">

<id property="did" column="DID" />

<result property="dname" column="DNAME"/>

</associatin>

</resultMap>

### 使用延迟加载(按需加载)（懒加载）

Emp--dept

我们每次查询Emp对象的时候，Emp,dept都将一起查询出来;

我们希望部门信息在我们使用的时候再去查询，

在分段查询的基础删加上两个配置

全局开启：lazyLoadingEnabled =true和aggressiveLazyLoading=false

## 场景二、查询部门的时候查询该部门下的所有员工

**1.使用Collection定义关联的集合类型的属性的封装规则**

**2.ofType:指定集合里元素的类型**

### 嵌套结果集的方式，使用Collection定义关联的集合类型的属性的封装规则

<resultMap id="userResultMap" type="com.itour.model.Dept">

<id property="did" column="DID" />

<result property="dname" column="DNAME"/>

<collection property="empList" ofType="com.itour.model.Emp" >

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

<result property="lastName" column="LAST\_NAME"/>

</collection>

</resultMap>

### 分布查询

1.查询出对应的部门

2.根据部门id查询出该部门下的员工

<resultMap id="userResultMap" type="com.itour.model.Dept">

<id property="did" column="DID" />

<result property="dname" column="DNAME"/>

<collection property="empList" select="com.itour.EmpMapper.selectByID" column="id" >

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

<result property="lastName" column="LAST\_NAEM"/>

</collection>

</resultMap>

## 拓展：

### 将多列值传过去

将多列的值封装map传递,column="{key1=column1,key2=column2}"

### 使用fetchType开启是否开启延迟加载

fetchType="lazy"表示使用延迟加载；eager:立即

### 鉴别器 <discriminator></discriminator>

鉴别器：mybatis可以使用discriminator判断某列的值，然后根据某列

的值改变封装行为;

column:指定判定的列名

javaType：列值对应的java类型

resultType:指定封装的结果集的类型

<discriminator javaType="String" column="sex">

<case value="0" resultType="com.itour.model.Emp">

指定结果集

</case>

<case value="1">

指定结果集

</case>

</discriminator>

部门编号为12和88的菜查询员工信息

<resultMap id="userResultMap" type="com.itour.model.Dept">

<id property="did" column="DID" />

<result property="dname" column="DNAME"/>

<discriminator javaType="String" column="did">

<case property="12">

<collection property="empList" ofType="com.itour.model.Emp" >

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

<result property="lastName" column="LAST\_NAME"/>

</collection>

</case>

<case property="88">

<collection property="empList" ofType="com.itour.model.Emp" >

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

<result property="lastName" column="LAST\_NAME"/>

</collection>

</case>

<discriminator>

</resultMap>

# 动态SQL

## if

test:判断表达式（OGNL）

查询时如果某一些条件没带可能SQL拼装会有问题

**解决方案：**

1.给where 后面加上1=1 以后的条件用and xxx;

2.Mybatis使用where标签来将索引的查询条件包括在内，

Mybatis会将where标签中拼接的SQL，多出的and或or去除,

where 只会去掉第一个多出来的and或or,可以使用trim标签解决；

## trim(where,set)

后面多出的and或or where标签不能及解决；

prefix=""前缀：trim标签体中是整个字符串拼接后的结果；

prefix给拼接后的整个字符串加一个前缀

prefixOverrides=""前缀覆盖：去掉整个字符串前面多余的字符

suffix=""给拼接后的整个字符串加一个后缀

suffixOverrides=""后缀覆盖：去掉整个字符串后面多余的字符

<trim prefix="WHERE" prefixOverrides="AND |OR ">

...

</trim>

## Choose

<choose>

<when test="title != null">

AND title like #{title}

</when>

<when test="author != null and author.name != null">

AND author\_name like #{author.name}

</when>

<otherwise>

AND featured = 1

</otherwise>

</choose>

## Foreach

collection:指定要遍历的集合

item:将当前遍历出的元素赋值给指定的变量

#｛变量｝就能取出变量的值也就是当前遍历的元素：

separator：每个元素之间的分割符

open:遍历出所有结果拼接一个开始的字符

close:遍历出所有结果拼接一个结束的字符

index:索引，遍历list的时候index是索引，item是指当前值。

遍历map的时候,index表示的是map的key，item就是map的值

<foreach item="item" index="index" collection="list"

open="ID in (" separator="," close=")" nullable="true">

#{item}

</foreach>

## 批量保存

MYSQL：支持values(),(),()...语法

如果需要支持insert...;insert...;

需要在jdbc.url连接属性中配置allowMultiQueries=true;

mysql默认不支持多条SQL后使用";"分割,这种方式分割多个

SQL可以用于其他批量操作(删除、修改)

<insert id="insertAuthor" useGeneratedKeys="true"

keyProperty="id">

insert into Author (username, password, email, bio) values

<foreach item="item" collection="list" separator=",">

(#{item.username}, #{item.password}, #{item.email}, #{item.bio})

</foreach>

</insert>

ORacle：不支持values(),(),()...语法;

Oracle支持的批量方式：

1.多Insert个sql放在在begin- end;中;

2.利用中间表

insert into emp (id,name,email)

select emp\_seq.nextval,name,email from (

select "amy" name,"wwangtao@163.com" from dual

union

select "mike" name,"mike@163.com" from dual

)

<insert id="insertAuthor"

>

insert into Author (id,username, password, email, bio)

select seq\_author.nextVal,username,password,email,bio from

<foreach item="item" collection="list" separator="," >

select (#{item.username}, #{item.password}, #{item.email}, #{item.bio})

</foreach>

</insert>

## 内置参数

mybatis默认的两个内置参数

\_paramter:代表整个参数

单个参数：\_paramter就是这个参数

多个参数：参数会被封装为一个map：\_paramter就是代表这个map;

\_databaseId:如果配置了databaseIdProvider标签，\_databaseId就是代表

当前数据库的别名

## 动态SQL绑定：

bind:可以见OGNL表达式的值绑定到一个变量中,方便引用;

## 抽取可重用的SQL

sql：

使用include标签引入;

include标签里可以自定义一些property，SQL标签内部就能使用自定义的属性;

include-property取值的正确方式：$(prop)，不能使用#｛prop｝

# Mybatis 缓存

## 简介

1.Mybatis默认定义两级缓存，一级缓存和二级缓存。

a.默认情况，只有一级缓存（sqlSession级别的缓存，也成本地缓存）开启;

b.二级缓存需要手动开启和配置，是基于namespace级别的缓存(全局缓存);

c.为了提高拓展性，Mybatis定义了缓存接口Cache。可以通过实现Cache接口来自定义二级缓存;

## 一级缓存

一级缓存(本地缓存)一级缓存是一直开启的;是sqlSession级别的一个map

与数据库同一次会话期间查询到的数据会放在本地缓存中.

以后如果需要获取相同的数据，直接从缓存中获取，无需去查询数据;

## 一级缓存失效的四种情况:

1.sqlSession不同；

2.sqlSession相同,查询条件不同(在一级缓存中没有);

3.sqlSession相同，两次查询之间执行了增删改操作（增删改对当前数据可能会造成影响）;

4.sqlSession相同，手动清除了一级缓存（清除缓存）;

## 二级缓存

二级缓存（局部缓存）是基于namespace级别的缓存;一个namespace对应一个二级缓存;

## 缓存工作机制：

1.一个会话，查询一条数据，这个数据就被放到当前会话的一级缓存中;

2.如果会话关闭,一级缓存中的数据会被保存二级缓存中；新的会话查询信息，

就可以参考二级缓存。

3.不同namespace查询出的数据会放在自己对应的缓存中（map）

## 二级缓存的使用步骤

1.开启全局二级缓存配置;cacheEnabld =true;

2.在Mapper中配置缓存 使用cache标签配置;

<cache eviction="" flushInterval="" readOnly="" size=""></cache>

eviction:缓存的回收策略

lru:最近最少使用，移除最长时间不被使用的对象;

fifo:先进先出：按照对象进入的顺序来移除他们;

soft:软引用，移除基于垃圾回收器的状态和软引用规则的对象；

weak :弱引用：更积极的移除基于垃圾回收器状态和弱引用规则的对象；

默认的是lru

flushInterval:缓存刷新间隔

缓存多长时间清空一次，默认不情况,设置一个毫秒值

readOnly：缓存是否只读;

true:只读;Mybatis认为所有从缓存中获取的数据都是只读操作的，不会修改数据；

Mybatis为了加速获取速度，直接将引用在缓存中的引用交给用户,不安全，速度快;

false：非只读;Mybatis觉得获取的数据可能被修改。

mybatis会利用序列号和反序列化技术克隆一份新数据给用户,安全，速度慢;

size：缓存存放多少元素;

type：指定自定义缓存的全类名；（实现Cach接口）

3.pojo需要实现序列号接口

数据从二级缓存中获取时，

查询出来的数据都会被默认先放到一级缓存中，

只有会话提交或关闭后，一级缓存的数据才会转移到二级缓存中；

## **缓存相关设置和属性**：

1.cacheEnabled=true|false(关闭缓存（二级缓存）（一级缓存一直可用）)

2.每个select标签都有useCache="true"

false:关闭缓存（一级缓存依然可以使用，二级缓存不能使用）

3.每个增删改的标签的：flushCache ="true"

增删改完成后就会清除缓存（一级缓存会失效），二级缓存也会被清空；

4.每个select标签：flushCache ="false",

如果改为true,每次查询之前都会清空缓存，缓存是没有被使用的 ;

5.sqlsession.clearCache()只是清除当前session的一级缓存;

6.localCacheScop:本地缓存作用域(一级缓存SESSION:当期会话的)

STATEMENT:可以禁用一级缓存

# Mybatis运行原理



1. 根据配置文件(全局，sql映射)初始化出Configration对象
2. 创建一个DefaultSqlSession对象，包含了Configration对象及

Executor(根据全局配置中的defaultExecutorType 创建出对应的Ececutor);

1. 获取Mapper对应MapperProxy对象
2. 执行增删改查方法;

1.调用DefaultSqlSession的增删改查(Executor)

2.会创建一个StatementHandler对象（同时创建出Parameterhandler和ResultSetHandler）

3.调用StatementHandler预编译参数以及设置参数值

使用ParameterHandler来给SQL设置参数

4.调用StatementHandler的增删改查方法

5.ResultSetHandler封装结果

# 插件运行原理

1.在四大对象创建的时候

a.每个创建的对象不是之间返回的，而是使用

interceptorChain.pluginAll（parameterHandler）进行封装一层;

2.获取到所有的Interceptor(拦截器)（插件需要实现的接口）

调用interceptor.plugin(target)返回target包装后的对象

3.插件机制，我们可以使用插件为目标对象创建一个代理对象;AOP(面向切面)

插件可以为四大对象创建出代理对象

代理对象就可以拦截到四大对象的每一个执行

# 如何编写插件

1. 编写Interceptor的实现类;
2. 使用@Interceptors注解完成插件签名
3. 将写好的插件注册到全局配置文件中

# 批量更新

defaultExecutorType =batch

# 调用存储过程

