## 1为什么学习Mybatis？

学习过的持久层框架：DBUtils（JdbcTemplate）/ Hibernate

mybatis是类似与hibernate的orm持久层框架。

框架的底层都时对jdbc的封装。

### 1.1 原因？

简单来说： 1 国内用mybatis的多

2 mybatis比hibernate门槛更低

3 sql优化方面，hibernate不如mybatis

4 如果hibernate使用了sqlquery，则sql语句和代码耦合，不利于后期维护。

* + 1. 目前最主流的持久层框架为hibernate和Mybatis，而且国内目前情况使用mybatis的公司比hibernate更多。
    2. hibernate学习门槛不低，要精通门槛更高。

== 门槛高在怎么设计O/R映射，在性能和对象模型之间如何权衡取得平衡，

== 以及怎样用好hibernate缓存与数据加载策略方面需要你的经验和能力都很强才行。

国内目前的情况精通hibernate技术大牛较少。

* + 1. sql优化方面，hibernate的查询会将表中的所有字段查询出来，这一点会有性能消耗。

当然了，hibernate也可以自己写sql来指定需要查询的字段，但是这样就破坏了hibernate开发的简洁性。

* + 1. 说的更深入一些，如果有个查询要关联多张表，比如10张表时，而且我们要取的字段知识其中几张表的部分字段，这是用hibernate时就会显得力不从心。如果使用hibernate的sqlquery，但是后续的维护工作也会让人发狂。

### 1.2问题研究？

* 1. Jdbc访问数据库存在问题？
     1. 频繁创建、打开和关闭连接，消耗资 🡪 解决：使用连接池
     2. sql语句硬编码 ：prepareStatement(“select \* from user where name = ?”),这样条件改变时，就需要重写sql语句。🡪解决： 使用配置文件实现解耦
     3. 结果集获取和遍历困难，存在硬编码问题；从ResultSet中遍历获取数据比较麻烦。——>使用框架hibernate/mybatis ,内部包装成POJO对象。

## 2 mybatis入门

### 2.1主要目标：

解决连接频繁创捷、打开和关闭问题

解决sql语句和参数硬编码问题

解决结果集处理麻烦问题

解决方法：

连接池 ：解决自主创建、打开关闭连接池问题

配置文件（sqlMapperConfig.xml） ：统一管理sql，降低耦合性。便于管理

映射文件 (sqlMapper.xml)：根据映射文件，输出对应的POJO

### 2.2 mybatis底层架构

sqlMapConfig.xml

sqlMapper1.xml

sqlMapper2.xml

SqlSessionFactoryBuilder加载核心配置文件

SqlSessionFactory

SqlSession是个接口

Executor（执行器：真正做事的人）根据每个sql语句创建一个对象MappedStatement，并对该对象进行参数等设置。

MappedStatement(一个sql语句，对应一个该对象)

输出映射：

参数类型

基本类型：int、string等

对象：POJO

集合：map、list

输入映射：

参数类型

基本类型：int、string等

对象：POJO

集合：map

访问数据库前，设置参数

访问数据库后，包装数据

## 3 sqlMapConfig.xml

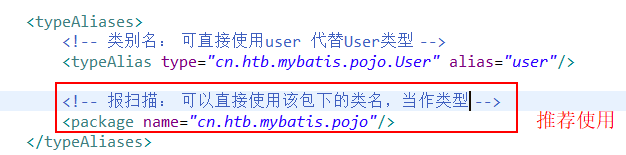


#### 2.2.1 mybatis环境配置：

比如：开发环境/生产环境/MySql连接/Oricle连接，等都可以配置。还可以配置多个，根据需要选择不同的环境。

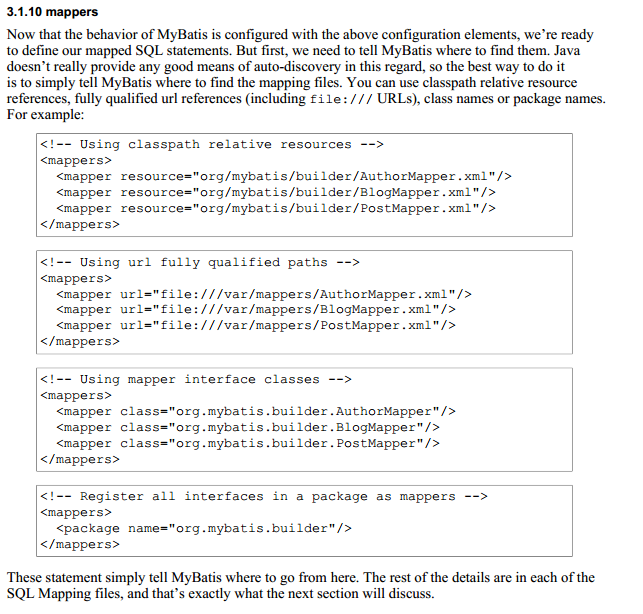


别名



示例





<mappers>

<mapper class=”接口的path”>

<mappers>

<mappers>

<package name=”包名”>

<mappers>

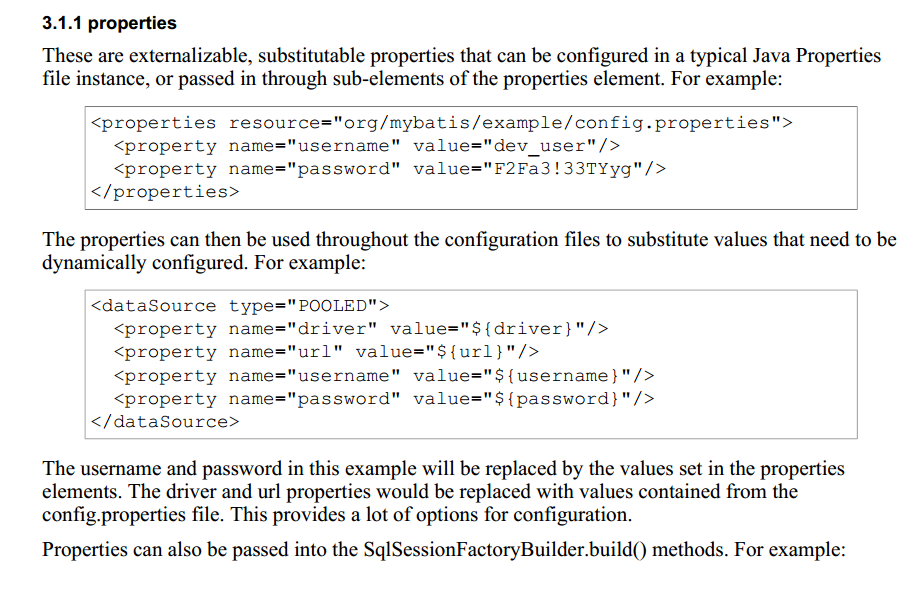
要求：

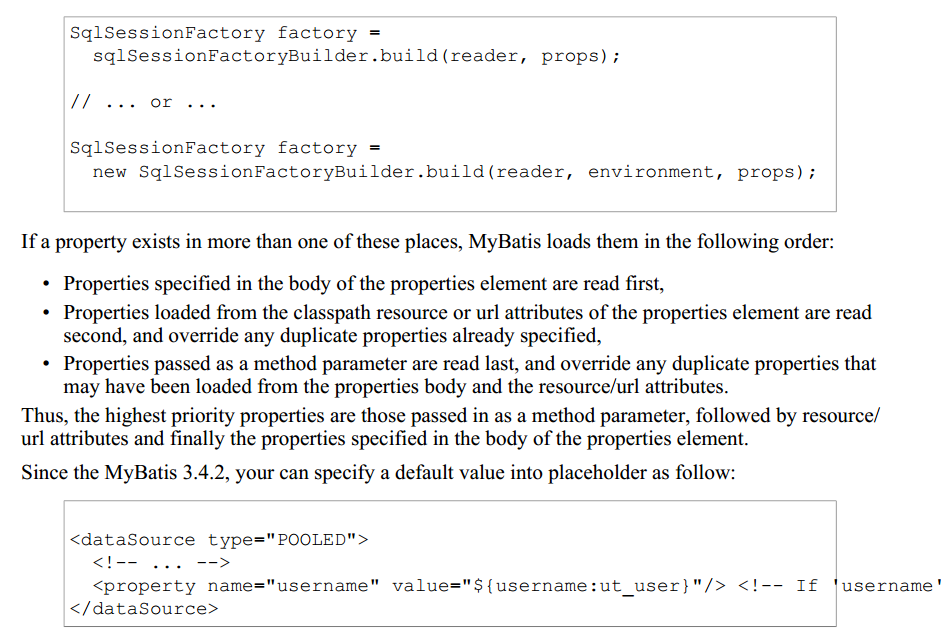
1 接口必须和映射文件在同一目录下

2 接口必须和映射文件名称一致



#### 2.2.2 sqlMapConfig.xml 资源加载顺序





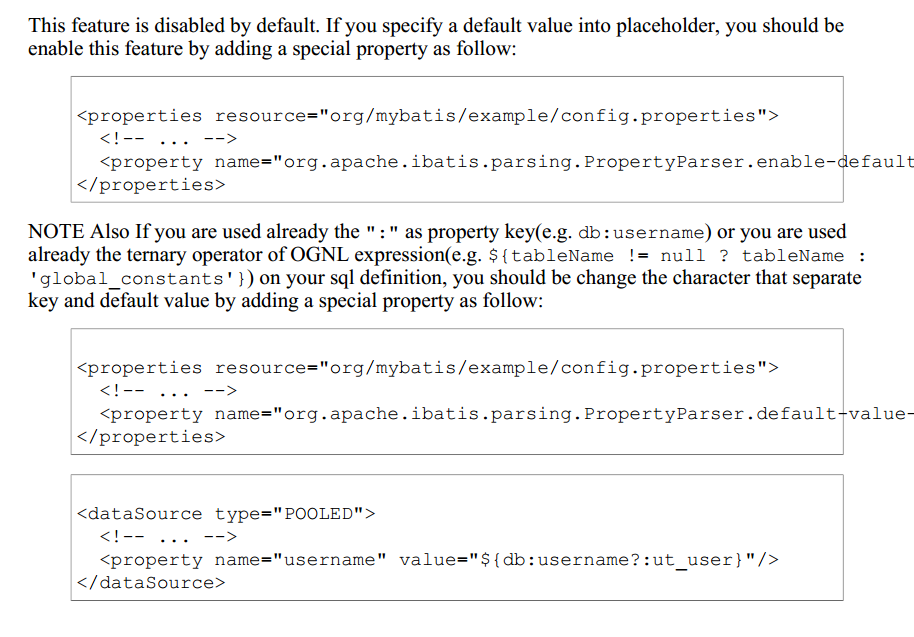
加载顺序：

1 先加载文件本身标签内容：<property name=”username” value=”root2”>

2 在加载classpath下的resource文件的内容（jdbc.properties）,覆盖名字一样的属性。

3 最后加载方法参数的内容，同样覆盖名字一样的值。

因此，优先级最高的是方法参数，其次是classpath的资源内容，最后是文件中的标签内容。



## 4 sqlMapper.xml

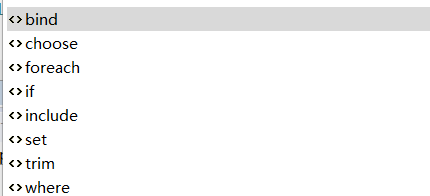
输入类型：基本类型（int，string，long）、

pojo、

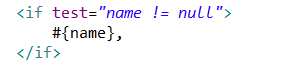
pojo包装类型（map、list、array）

输出类型：基本类型、pojo、pojo列表

### 4.1 动态SQL ：MyBatis 中的sql标签



#### 4.1.1 if标签：用于判断



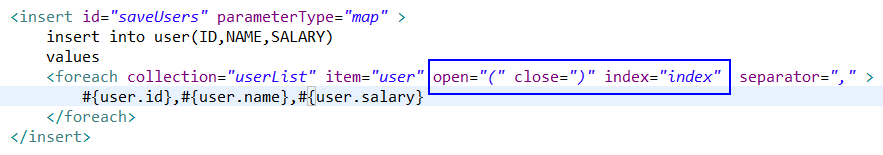
#### 4.1.2 foreach标签：动态组装sql。多用于遍历,比如：map的遍历



3.2.1结果如下：



3.2.2如果改成下图：



则结果如图，说明：open 和 close分别表示，整个组装sql的开始和结束.要合理使用。

separator则表示每个foreach中的sql的分隔。



#### 4.1.3 trim标签：动态组装sql

3.3.1 prefix 和suffix表示组装完整sql的前缀和后缀

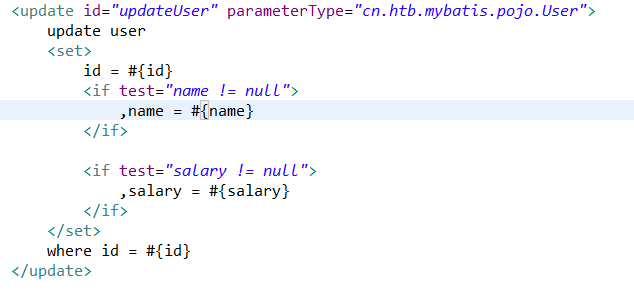
3.3.2 suffixOverrides是 表示覆盖/去除trim标签内，末尾的与之匹配的字符。

如下:去除最后一个 逗号

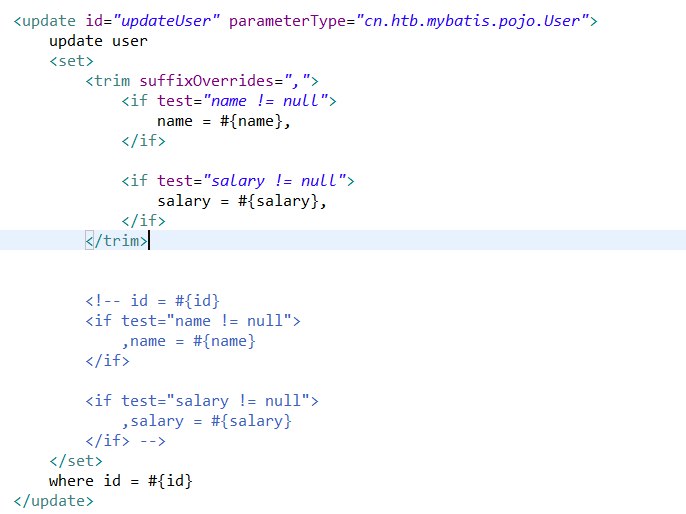
3.3.3 prefixOverrides是 表示覆盖/去除trim标签内，句首的与之匹配的字符。



#### 4.1.4 set标签：用户更新时使用



使用trim标签：



#### 4.1.5 choose标签：

<choose>

<when test=””><when/>

<otherwise><otherwise/>

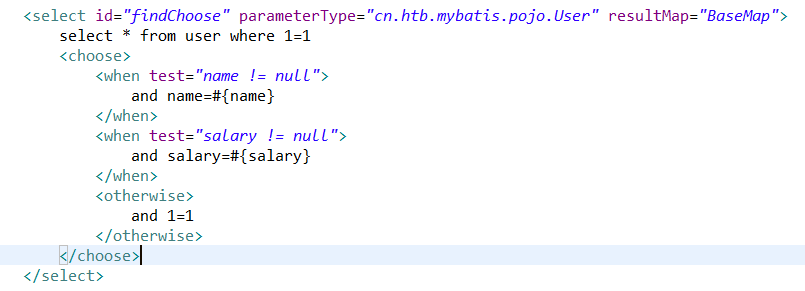
<choose/>

类似：

if{ }

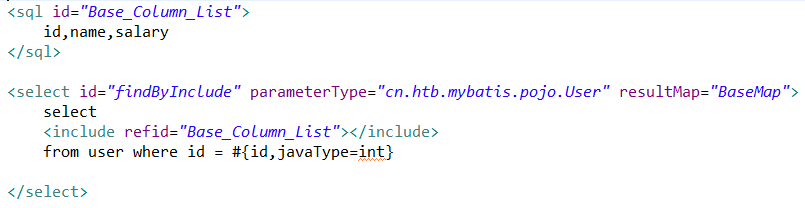
else if{ }

else{ }

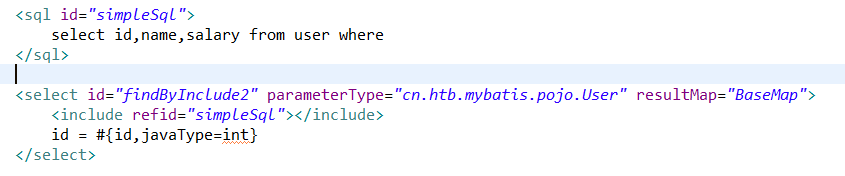


#### 4.1.6 include标签：用于引入sql语句

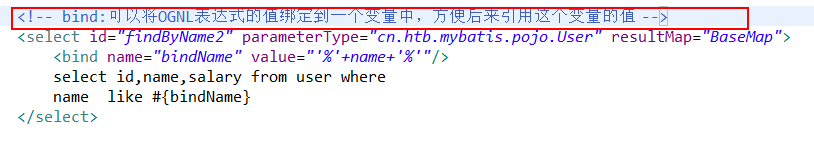
例一



例二



#### 4.1.7 bind标签使用



### 4.2mybatis传入参数类型

#### 4.2.1传入参数类型主要两类：

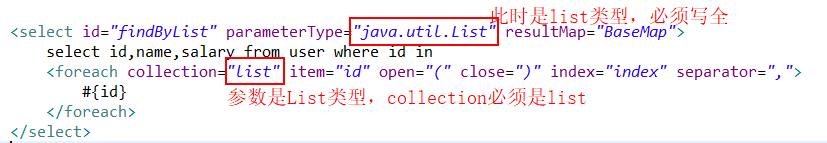
基本数据类型：int string long Date

复杂数据类型：类，Map、List 、Arrays (当时List和Arrays时，parameter值必须写全：java.util.Arrays java.util.List)

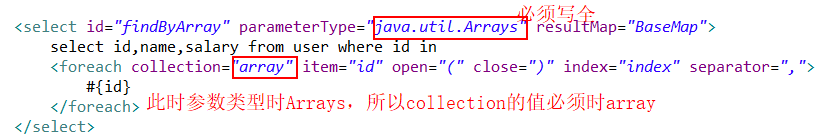
#### 4.2.2 foreach中的collection

collection属性是在使用foreach的时候最关键的也是最容易出错的，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：

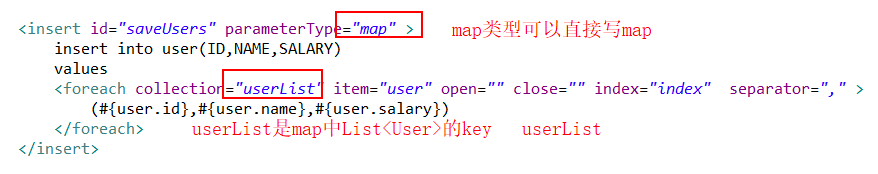
（1）传入的是单参数且参数类型是一个List的时候，collection属性值必须为list .



（2）传入的是单参数且参数类型是一个array数组，collection的属性值必须为array .



（3）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.



### 4.3 sql标签测试和参数类型测试

简单参数类型：



#### 4.3.1xml配置文件

<?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=*"cn.htb.mybatis.dao.UserMapper"*>

<resultMap type=*"cn.htb.mybatis.pojo.User"* id=*"BaseMap"*>

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

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

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

</resultMap>

<!-- 普通查询测试========================= -->

<select id=*"findById"* parameterType=*"int"* resultMap=*"BaseMap"*>

select \* from user where id = #{id};

</select>

<!-- 模糊查询测试=========================== -->

<select id=*"findByName"* parameterType=*"string"* resultMap=*"BaseMap"*>

select \* from user where name like "%${value}%"

</select>

<!--

======================

sql标签测试

=========================

-->

<!--

map.put("list", List<User>)

foreach遍历集合来组装sql

collection：map中集合的key

open:以某种字符开始

close:以某种字符结束

item：集合中的元素

separator：以某种字符分隔

index：当前遍历到的索引号

-->

<!-- 测试=========================== foreach -->

<insert id=*"saveUsers"* parameterType=*"map"* >

insert into user(ID,NAME,SALARY)

values

<foreach collection=*"userList"* item=*"user"* open=*""* close=*""* index=*"index"* separator=*","* >

(#{user.id},#{user.name},#{user.salary})

</foreach>

</insert>

<!-- 测试=========================== trim -->

<insert id=*"saveUser"* parameterType=*"cn.htb.mybatis.pojo.User"*>

insert into user

<trim prefix=*"("* suffix=*")"* suffixOverrides=*","* prefixOverrides=*""*>

<if test=*"id != null"*>

ID,

</if>

<if test=*"name != null"*>

NAME,

</if>

<if test=*"salary != null"*>

SALARY,

</if>

</trim>

<trim prefix=*"values("* suffix=*")"* suffixOverrides=*","* >

<if test=*"id != null"*>

#{id},

</if>

<if test=*"name != null"*>

#{name},

</if>

<if test=*"salary != null"*>

#{salary},

</if>

</trim>

</insert>

<select id=*"findByCondition"* parameterType=*"cn.htb.mybatis.pojo.User"* resultMap=*"BaseMap"*>

select \* from user

<trim prefix=*"where"* prefixOverrides=*"or"*>

<if test=*"id != null"*>

or id = #{id}

</if>

<if test=*"name != null"*>

or name=#{name}

</if>

<if test=*"salary != null"*>

or salary=#{salary}

</if>

</trim>

</select>

<!-- 测试=========================== set -->

<update id=*"updateUser"* parameterType=*"cn.htb.mybatis.pojo.User"*>

update user

<set>

<trim suffixOverrides=*","*>

<if test=*"name != null"*>

name = #{name},

</if>

<if test=*"salary != null"*>

salary = #{salary},

</if>

</trim>

<!-- id = #{id}

<if test="name != null">

,name = #{name}

</if>

<if test="salary != null">

,salary = #{salary}

</if> -->

</set>

where id = #{id}

</update>

<!-- 测试=========================== choose -->

<select id=*"findChoose"* parameterType=*"cn.htb.mybatis.pojo.User"* resultMap=*"BaseMap"*>

select \* from user where 1=1

<choose>

<when test=*"name != null"*>

and name=#{name}

</when>

<when test=*"salary != null"*>

and salary=#{salary}

</when>

<otherwise>

and 1=1

</otherwise>

</choose>

</select>

<!-- 测试=========================== include -->

<sql id=*"Base\_Column\_List"*>

id,name,salary

</sql>

<select id=*"findByInclude"* parameterType=*"cn.htb.mybatis.pojo.User"* resultMap=*"BaseMap"*>

select

<include refid=*"Base\_Column\_List"*></include>

from user where id = #{id,javaType=int}

</select>

<sql id=*"simpleSql"*>

select id,name,salary from user where

</sql>

<select id=*"findByInclude2"* parameterType=*"cn.htb.mybatis.pojo.User"* resultMap=*"BaseMap"*>

<include refid=*"simpleSql"*></include>

id = #{id,javaType=int}

</select>

<!-- 测试=========================== bind -->

<!-- bind:可以将OGNL表达式的值绑定到一个变量中，方便后来引用这个变量的值 -->

<select id=*"findByName2"* parameterType=*"cn.htb.mybatis.pojo.User"* resultMap=*"BaseMap"*>

<bind name=*"bindName"* value=*"'%'+name+'%'"*/>

select id,name,salary from user where

name like #{bindName}

</select>

<!--

======================

参数类型测试

=========================

-->

<!-- 测试=========================== 数组Arrays -->

<select id=*"findByArray"* parameterType=*"java.util.Arrays"* resultMap=*"BaseMap"*>

select id,name,salary from user where id in

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

#{id}

</foreach>

</select>

<!-- 测试=========================== 集合List -->

<select id=*"findByList"* parameterType=*"java.util.List"* resultMap=*"BaseMap"*>

select id,name,salary from user where id in

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

#{id}

</foreach>

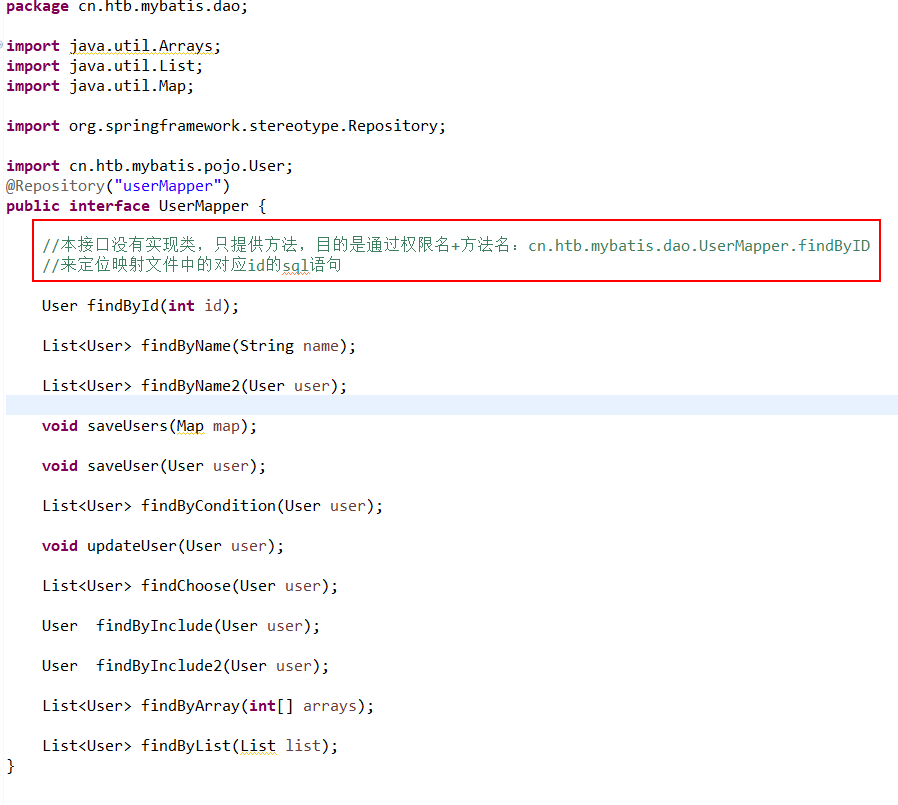
</select>

<!-- 测试=========================== map和基本类型上边已经涉及到了 -->

</mapper>

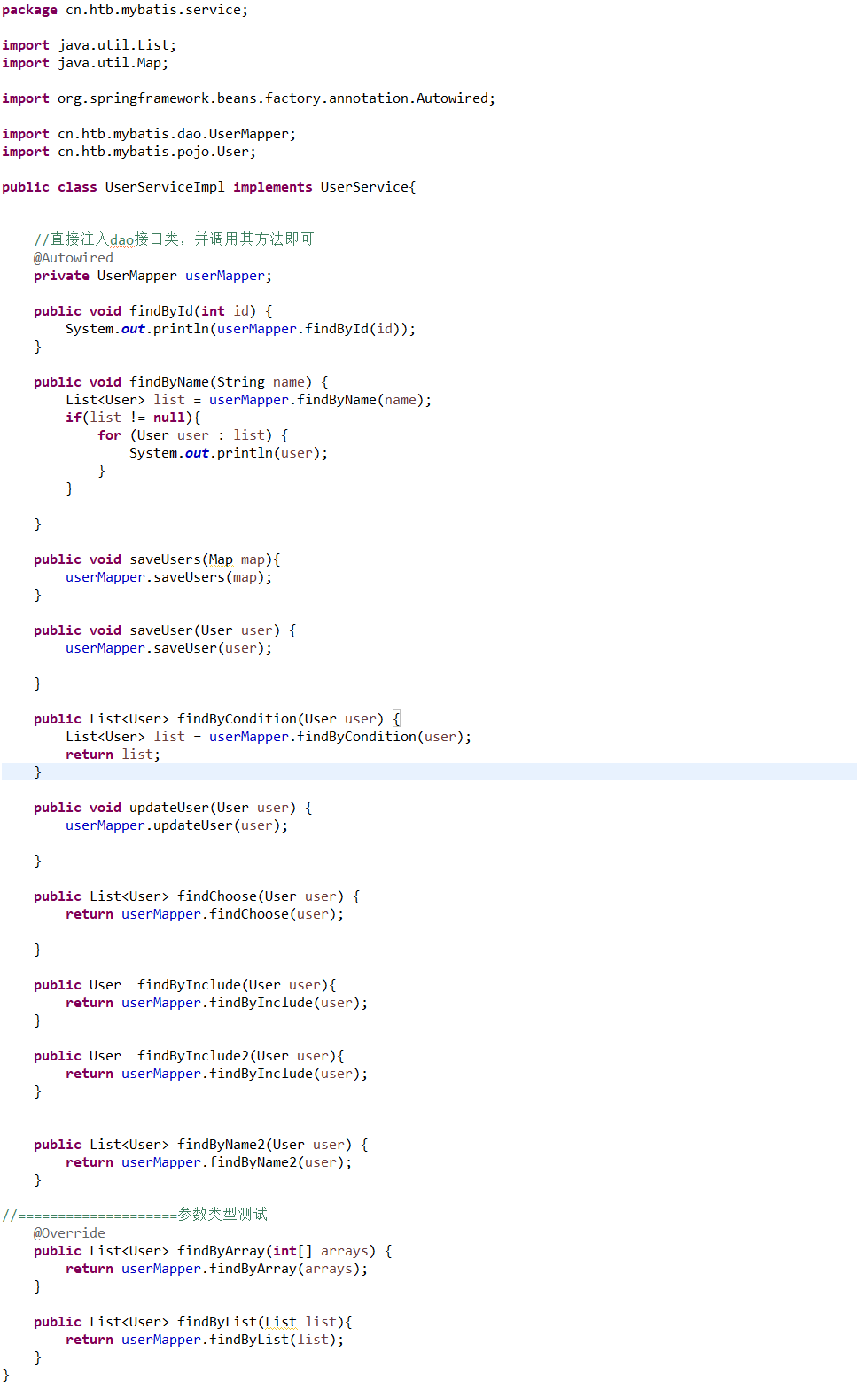


5.2 dao接口代码



5.3 service代码





5.4 测试代码

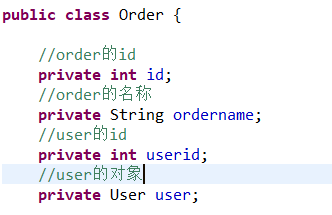


## 5查询相关

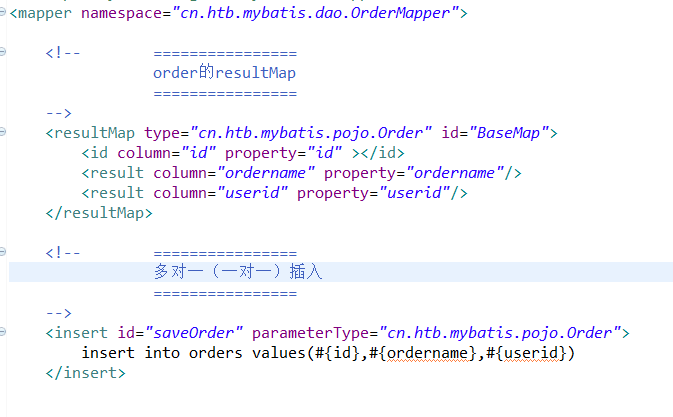
### 5.1 关联关系

#### 5.1.1 一对一（多对一也是一种一对一）

Order---->User



##### 5.1.1.1插入：



##### 5.1.1.2查询：

重点是resultMap中的

association标签：一对一

property是对应的属性，javaType是该属性对应类型

<association property=*"user"* javaType=*"cn.htb.mybatis.pojo.User"*>

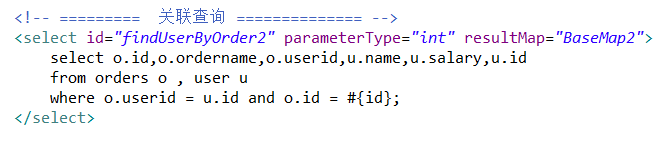
collection 标签：一对多

property是对应的集合属性，ofType是该集合属性包含对象的类型

<collection property=*"orderList"* ofType=*"cn.htb.mybatis.pojo.Order"*>

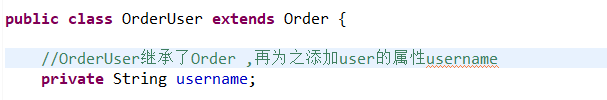
###### 5.1.1.2.1查询方式一：采用ResultMap建立关联关系

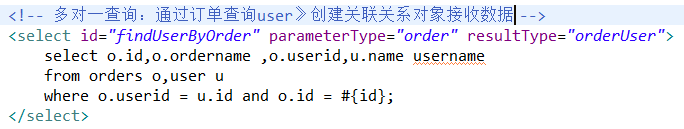




###### 5.1.1.2.2查询方式二：resultType 创建新的关联关系对象

OrderUser;其属性为查询order和user属性的综合



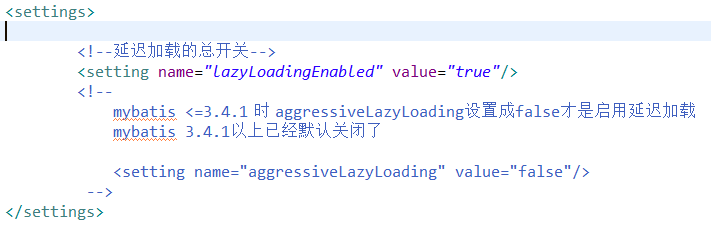


#### 5.1.2 一对多（多对多也是一种一对多，从中间表来看）



### 5.2 延迟加载

#### 5.2.1开启延迟加载开关 sqlMapConfig.xml

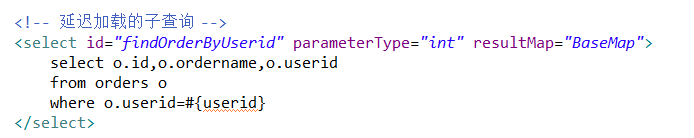


#### 5.2.2 sqlMapper.xml中使用延迟加载

本例是一对多延迟加载，一对一（多对一）延迟加载同理。



子查询语句：



### 5.3 缓存

一个项目中肯定会存在很多共用的查询数据，对于这一部分的数据，没必要

每一个用户访问时都去查询数据库，因此配置二级缓存将是非常必要的。

#### 5.3.1 开启缓存sqlMapConfig.xml

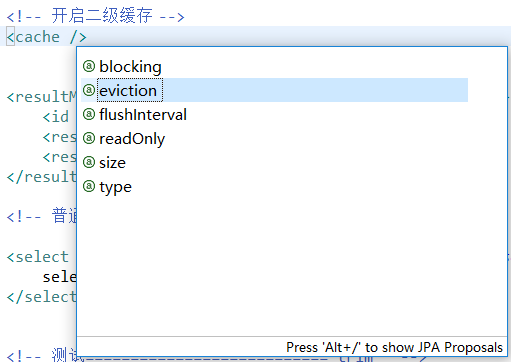


#### 5.3.2 mapper.xml中开启缓存

##### 5.3.2.1有这个标签就代表开启，不设置的话就是按照默认机制



##### 5.3.2.2<cache>标签内容也可以自己设置



1 eviction

这个更高级的配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会 导致冲突。可用的收回策略有, 默认的是 LRU:

1. LRU – 最近最少使用的:移除最长时间不被使用的对象。
2. FIFO – 先进先出:按对象进入缓存的顺序来移除它们。
3. SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
4. WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

2 flushInterval(刷新间隔)

可以被设置为任意的正整数,而且它们代表一个合理的毫秒 形式的时间段。默认情况是不设置,也就是没有刷新间隔,缓存仅仅调用语句时刷新。

3 size (引用数目)可以被设置为任意正整数,要记住你缓存的对象数目和你运行环境的 可用内存资源数目。默认值是1024。

4 readOnly(只读)属性可以被设置为 true 或 false。只读的缓存会给所有调用者返回缓 存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存 会返回缓存对象的拷贝(通过序列化) 。这会慢一些,但是安全,因此默认是 false。

5 type 修改缓存类型：比如第三方

<cache type=*"org.mybatis.caches.ehcache.EhcacheCache"*/>

##### 5.3.2.3 精度设置

在select语句中useCache=false可以禁用当前的语句的二级缓存，即每次查询夸session 的查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。

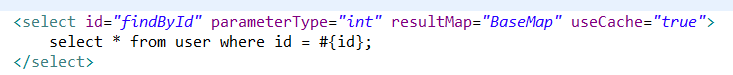
映射文件中所有的insert、update和delete语句将刷新缓存，三个语句中有flushCache=*"true"* 属性，默认情况下为true，即同个sessionFactory中当有sql语句更新时缓存自动被刷新以保证数据的实时性，如果改成false则不会刷新。

使用缓存时如果手动修改数据库表中的查询数据会出现脏读， 缓存将使用LRU（Least Recently Used）最近最少使用策略算法来回.

<cache  eviction="FIFO"  flushInterval="60000"  size="512"  readOnly="true"/>

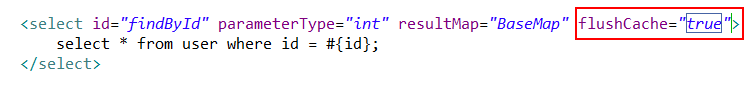
#### 5.3.3精确到sql是否启用缓存：

useCache为true则启用，false不启用



#### 5.3.4 是否刷新缓存

flushCach为true刷新（默认），false不刷新



#### 5.3.5二级缓存框架

采用mybatis的二级缓存框架

##### 5.3.5.1：引入缓存的依赖包

C:\Users\renliang\AppData\Roaming\Tencent\Users\491793587\QQ\WinTemp\RichOle\JOZ58Q@U{X_}F`EN}BLVU1A.jpg

##### 5.3.5.2：引入缓存配置文件

ehcache.xml

defaultCache配置说明：

maxElementsInMemory 内存中最大缓存对象数.当超过最大对象数的时候,ehcache会按指定的策略去清理内存  
eternal 缓存对象是否永久有效,一但设置了,timeout将不起作用.  
timeToIdleSeconds 设置Element在失效前的允许闲置时间.仅当element不是永久有效时使用,可选属性,默认值是0,也就是可闲置时间无穷大.  
timeToLiveSeconds：设置Element在失效前允许存活时间.最大时间介于创建时间和失效时间之间.仅当element是永久有效时使用,默认是0.,也就是element存活时间无穷大.  
overflowToDisk 配置此属性,当内存中Element数量达到maxElementsInMemory时,Ehcache将会Element写到磁盘中.  
diskSpoolBufferSizeMB 这个参数设置DiskStore(磁盘缓存)的缓存区大小.默认是30MB.每个Cache都应该有自己的一个缓冲区.  
maxElementsOnDisk 磁盘中最大缓存对象数,若是0表示无穷大.  
diskPersistent 是否在重启服务的时候清楚磁盘上的缓存数据.true不清除.  
diskExpiryThreadIntervalSeconds 磁盘失效线程运行时间间隔.  
memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时,Ehcache将会根据指定的策略去清理内存.默认策略是LRU(最近最少使用).你可以设置为FIFO(先进先出)或是LFU(较少使用).

##### 5.3.5.3：修改mapper文件中缓存类型

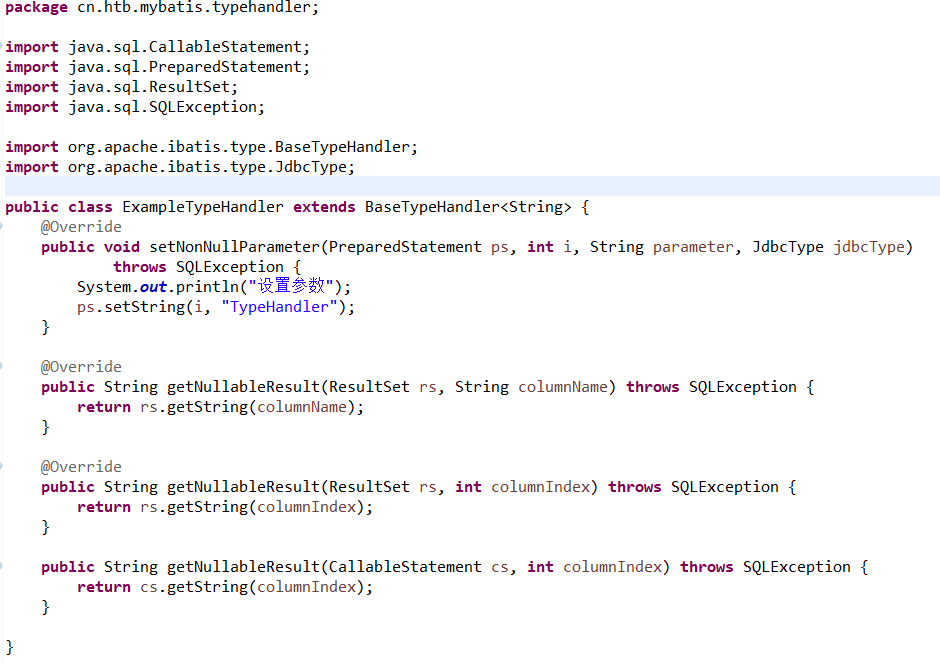
<cache type=*"org.mybatis.caches.ehcache.EhcacheCache"*/>

## 6扩展（TypeHandler和Plugins）

### 6.1 TypeHandler自定义

#### 6.1.1方式一：

##### 6.1.1.1自定义TypeHandler



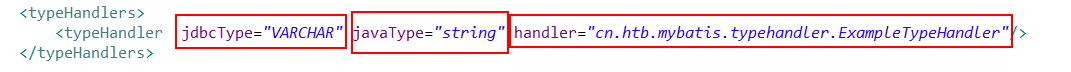
##### 6.1.1.2sqlMapCofig配置：

如果自定一处理器中没有添加注解：

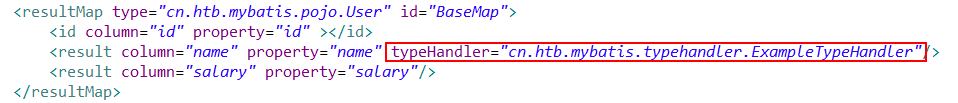
@MappedJdbcTypes(JdbcType.***VARCHAR***)

@MappedTypes(String.**class**)

则需要在配置文件中指明：哪种java类型（javaType=”string”），对应哪种数据库类型(jdbcType=”VARCHAR”)



##### 6.1.1.3 自定义typeHandler的使用



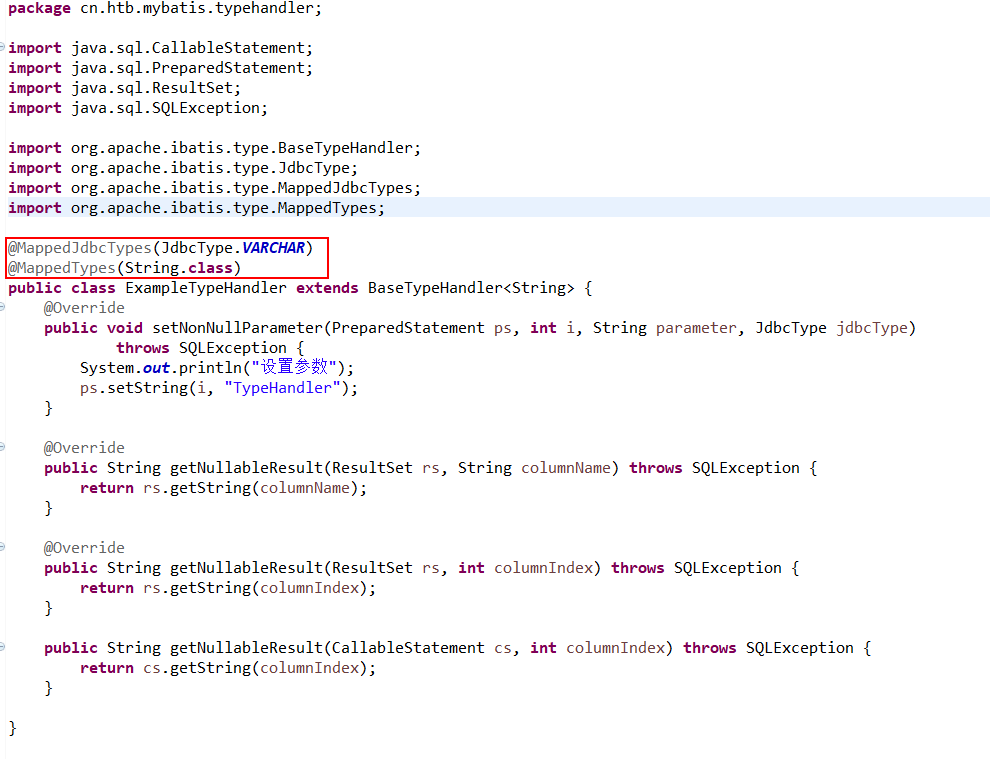


#### 6.1.2方式二

##### 6.1.2.1在自定义处理器上添加注解

这样就可以，指明Java类型对应的数据库类型。

无论在接口添加注解，还是配置文件添加属性。最终就是要指明哪种java类型对应哪种数据库类型



### 6.2 plugins 插件案例（分页）

代码：

#### 6.2.1 分页对象模型

**package** cn.htb.page;

**import** java.io.Serializable;

**import** java.util.List;

**public** **class** Pager<T> **implements** Serializable{

/\*\*

\* 开始位置

\*/

**private** **int** startPos;

/\*\*

\* 当前页码

\*/

**private** **int** curPage;

/\*\*

\* 每页大小

\*/

**private** **int** pageSize;

/\*\*

\* 每一页的数据

\*/

**private** List<T> datas;

/\*\*

\* 总页数

\*/

**private** **int** totalPage;

/\*\*

\* 总数量

\*/

**private** **int** totalCount;

**public** Pager(**int** curPage, **int** pageSize) {

**this**.curPage = curPage;

**this**.pageSize = pageSize;

**this**.startPos = (**this**.curPage - 1) \* **this**.pageSize;

}

**public** **int** getStartPos() {

**return** startPos;

}

**public** **void** setStartPos(**int** startPos) {

**this**.startPos = startPos;

}

**public** **int** getCurPage() {

**return** curPage;

}

**public** **void** setCurPage(**int** curPage) {

**this**.curPage = curPage;

}

**public** **int** getPageSize() {

**return** pageSize;

}

**public** **void** setPageSize(**int** pageSize) {

**this**.pageSize = pageSize;

}

**public** List<T> getDatas() {

**return** datas;

}

**public** **void** setDatas(List<T> datas) {

**this**.datas = datas;

}

**public** **int** getTotalPage() {

**return** totalPage;

}

**public** **void** setTotalPage(**int** totalPage) {

**this**.totalPage = totalPage;

}

**public** **int** getTotalCount() {

**return** totalCount;

}

**public** **void** setTotalCount(**int** totalCount) {

**this**.totalCount = totalCount;

**this**.totalPage = (**this**.totalCount - 1) / **this**.pageSize + 1;

}

@Override

**public** String toString() {

**return** "Pager{" +

"startPos=" + startPos +

", curPage=" + curPage +

", pageSize=" + pageSize +

", datas=" + datas +

", totalPage=" + totalPage +

", totalCount=" + totalCount +

'}';

}

}

#### 6.2.2 当前线程工具类

**package** cn.htb.page;

**public** **class** ThreadLocalUtil {

**public** **final** **static** ThreadLocal<Pager<?>> ***threadLocal*** = **new** ThreadLocal<Pager<?>>();

}

#### 6.2.3 插件代码

**package** cn.htb.page;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.util.Properties;

**import** org.apache.ibatis.executor.parameter.ParameterHandler;

**import** org.apache.ibatis.executor.statement.StatementHandler;

**import** org.apache.ibatis.mapping.BoundSql;

**import** org.apache.ibatis.mapping.MappedStatement;

**import** org.apache.ibatis.plugin.Interceptor;

**import** org.apache.ibatis.plugin.Intercepts;

**import** org.apache.ibatis.plugin.Invocation;

**import** org.apache.ibatis.plugin.Plugin;

**import** org.apache.ibatis.plugin.Signature;

**import** org.apache.ibatis.reflection.DefaultReflectorFactory;

**import** org.apache.ibatis.reflection.MetaObject;

**import** org.apache.ibatis.reflection.SystemMetaObject;

@Intercepts(value = {@Signature(

type = StatementHandler.**class**,

method = "prepare",

args = {

Connection.**class**,

Integer.**class**

}

)})

**public** **class** ThreadLocalPagePlugin **implements** Interceptor{

/\*\*

\* 这个方法是实际的拦截逻辑，我们的目的是在这里来实现分页，需要达到什么程度的使用。

\* 假设从ThreadLocal获取分页信息，来进行分页操作；

\* **@param** invocation

\* **@return**

\* **@throws** Throwable

\*/

@Override

**public** Object intercept(Invocation invocation) **throws** Throwable {

// 获取目标对象，注意StatementHandler中的属性都是protected

// 不能直接访问，因此需要通过其他的方式来获取，就是MetaObject

// 其基本实现是BaseStatementHandler其中最重要的属性是MappedStatment

// 包含了SQL相关信息

// 实际返回的是RoutingStatementHandler

StatementHandler handler = (StatementHandler) invocation.getTarget();

// 获取指定对象的元信息

MetaObject metaObject = MetaObject.*forObject*(

handler,

SystemMetaObject.***DEFAULT\_OBJECT\_FACTORY***,

SystemMetaObject.***DEFAULT\_OBJECT\_WRAPPER\_FACTORY***,

**new** DefaultReflectorFactory()

);

// 然后就可以通过MetaObject获取对象的属性

// 获取RoutingStatementHandler->PrepareStatementHandler->BaseStatementHandler中的mappedStatement

// mappedStatement 包含了Sql的信息

MappedStatement mappedStatement = (MappedStatement) metaObject.getValue("delegate.mappedStatement");

// 获取statement id

String statementId = mappedStatement.getId();

// 会拦截每个属性

**if** (statementId.endsWith("ByPage")){

// ByPage 表示的是分页查询

BoundSql boundSql = handler.getBoundSql();

String sql = boundSql.getSql();

// 获取当前线程分页信息

Pager<?> pager = ThreadLocalUtil.***threadLocal***.get();

String countSql = "SELECT COUNT(\*) " + sql.substring(sql.indexOf("FROM"));

Connection conn = (Connection) invocation.getArgs()[0];

PreparedStatement ps = conn.prepareStatement(countSql);

// 获取参数处理器来处理参数

ParameterHandler ph = (ParameterHandler) metaObject.getValue("delegate.parameterHandler");

ph.setParameters(ps);

// 执行查询

ResultSet rs = ps.executeQuery();

**if**(rs.next()){

pager.setTotalCount(rs.getInt(1));

}

String pageSql = sql + " LIMIT " + pager.getStartPos() + ", " + pager.getPageSize();

metaObject.setValue("delegate.boundSql.sql", pageSql);

}

**return** invocation.proceed();

}

// 指定需要拦截的对象

@Override

**public** Object plugin(Object target) {

**return** Plugin.*wrap*(target, **this**);

}

// 初始化属性

@Override

**public** **void** setProperties(Properties properties) {

}

}

#### 6.2.4 sqlMapConfig.xm配置

<plugins>

<plugin interceptor=*"cn.htb.page.ThreadLocalPagePlugin"* />

</plugins>

#### 6.2.5 sqlMapper.xml 配置

<select id=*"findByPage"* resultType=*"User"*>

SELECT \*

FROM user

</select>

#### 6.2.6测试代码

@Test

**public** **void** test(){

Pager<User> pager = **new** Pager<User>(1, 4);

ThreadLocalUtil.***threadLocal***.set(pager);

List<User> list = userService.findByPage();

**for** (User user : list) {

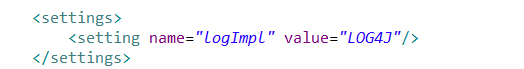
System.***out***.println(user);

}

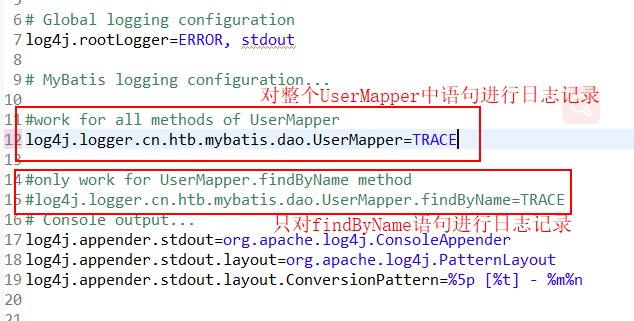
}

### 6.3 mybatis日志记录

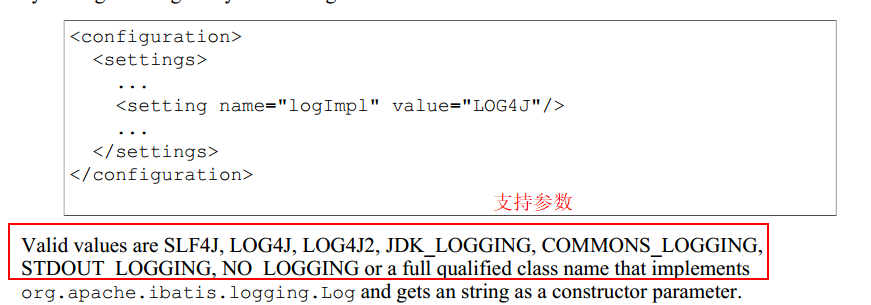
#### 6.3.1 sqlMapConfig.xml配置

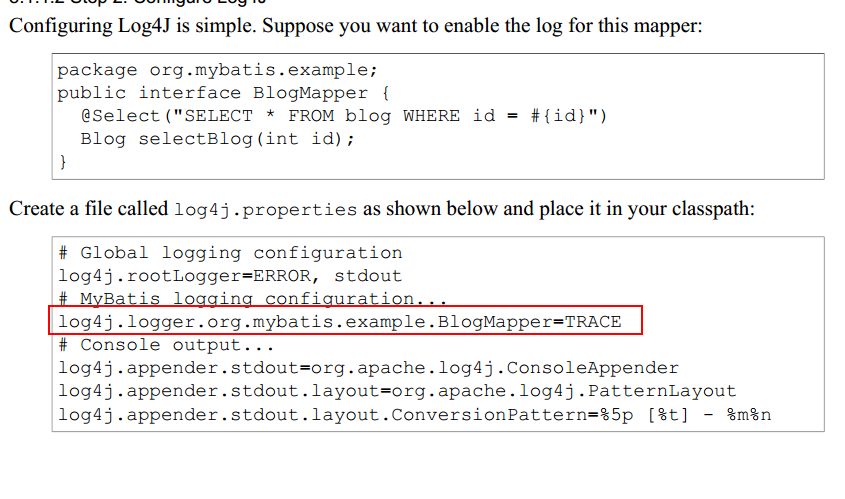


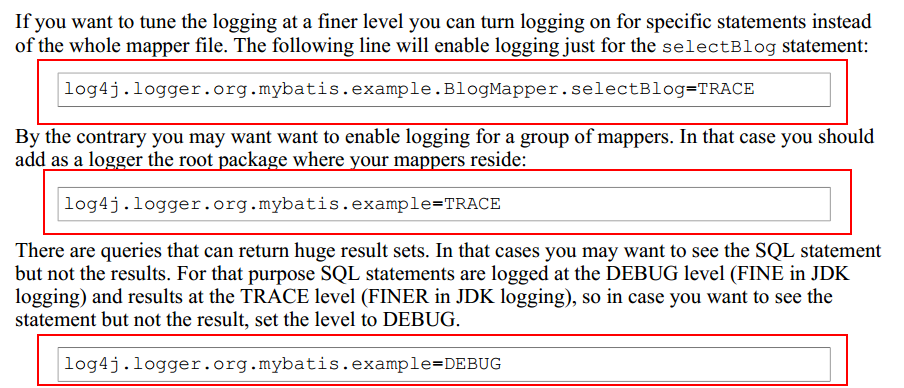
#### 6.3.2 log4j.properties内容



#### 6.3.3 官方文档







## 7 开发方式

### 7.1 传统开发

SqlSessionFactoryBuilder ssfb = new SqlSessionFactoryBuilder( );

InputStream inpuStream = Resources.getResourceAsStream(“classpath:sqlMapConfig.xml”);

SqlSessionFactory ssf = ssfb.build(inputStream);

SqlSession sqlSession = ssf.openSession( );

sqlSession.selectList();

sqlSession.selectOne();

sqlSession.update();

### 7.2动态代理开发

#### 7.2.1开发规则

7.2.1.1 namespace必须是接口的全路径名

7.2.1.2 接口的方法名必须与sql 语句的id一致

7.2.1.3 接口的入参与parameterType类型一致

7.2.1.4 接口的返回值必须与resultType类型一致

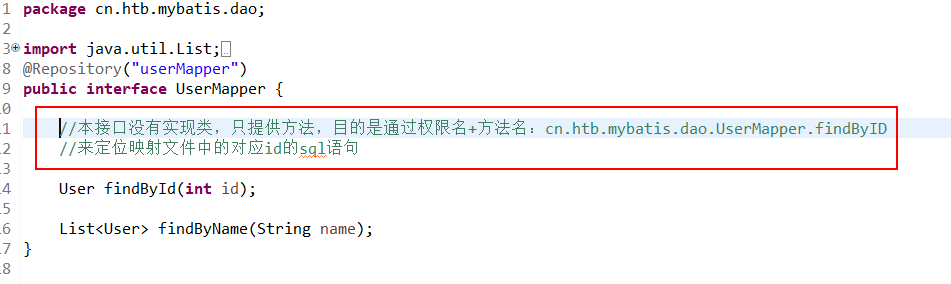
#### 7.2.2 mybatis整合spring面向dao接口编程



spring完全整合mybaits



dao代码：



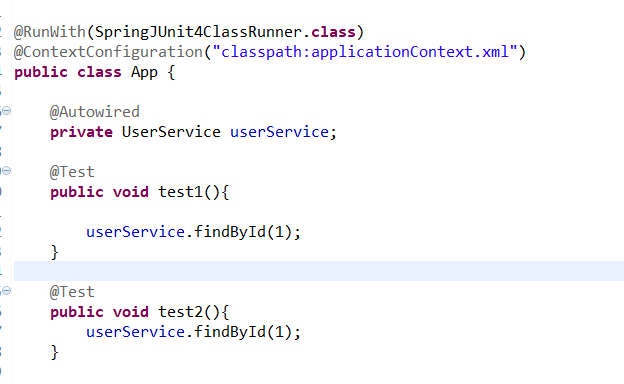
mapper映射文件



service代码



测试类：



对象和数据的转换

对象->数据

：映射关系（resultMap）

== association

== collection

：TypeHandler

== BaseTypeHandler

数据->对象

：SQL语句（动态标签）

：延迟加载

== LazyLoadingEnabled ->true

：开启缓存

== CacheEnabled ->true

== Cache

：Plugns

== Interceptor

开发方式：

：传统开发

：动态代理开发（DAO接口）

## 8分页处理

逆向工程生成的代码是不支持分页处理的，如果想进行分页需要自己编写mapper，这样就失去逆向工程的意义了。为了提高开发效率可以使用mybatis的分页插件PageHelper。

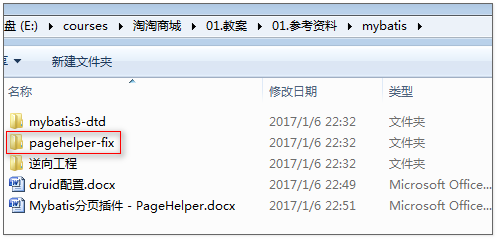
## 分页插件PageHelper

### Mybatis分页插件 - PageHelper说明

如果你也在用Mybatis，建议尝试该分页插件，这个一定是最方便使用的分页插件。该插件目前支持Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库分页。

### 使用方法

第一步：把PageHelper依赖的jar包添加到工程中。官方提供的代码对逆向工程支持的不好，使用参考资料中的pagehelper-fix。



使用参考资料中的工程 ，将其导入到eclipse 使用maven 安装到本地仓库。

第二步：在Mybatis的全局文件中配置SqlMapConfig.xml中配置拦截器插件:

<plugins>

<!-- com.github.pagehelper为PageHelper类所在包名 -->

<plugin interceptor="com.github.pagehelper.PageHelper">

<!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库-->

<property name="dialect" value="mysql"/>

</plugin>

</plugins>

PageHelper4.0以后使用这个

<plugins>

<!-- 分页插件 ：其实就是一个拦截器 com.github.pagehelper为PageHelper类所在包名 -->

<plugin interceptor=*"com.github.pagehelper.PageInterceptor"*>

<!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库

数据库自动匹配，不用配置了

-->

<!-- <property name="dialect" value="mysql"/> -->

</plugin>

</plugins>

第三步：在代码中使用

1、设置分页信息：

//获取第1页，10条内容，默认查询总数count

PageHelper.startPage(1, 10);

//紧跟着的第一个select方法会被分页

List<Country> list = countryMapper.selectIf(1);

//第二个不会被分页

List<Country> list = countryMapper.selectIf(1);

2、取分页信息第一种方法

//分页后，实际返回的结果list类型是Page<E>，如果想取出分页信息，需要强制转换为Page<E>，

Page<Country> listCountry = (Page<Country>)list;

listCountry.getTotal();

1. 取分页信息的第二种方法，推荐使用第二种

//获取第1页，10条内容，默认查询总数count

PageHelper.startPage(1, 10);

List<Country> list = countryMapper.selectAll();

//用PageInfo对结果进行包装

PageInfo page = new PageInfo(list);

//测试PageInfo全部属性

//PageInfo包含了非常全面的分页属性

assertEquals(1, page.getPageNum());

assertEquals(10, page.getPageSize());

assertEquals(1, page.getStartRow());

assertEquals(10, page.getEndRow());

assertEquals(183, page.getTotal());

assertEquals(19, page.getPages());

assertEquals(1, page.getFirstPage());

assertEquals(8, page.getLastPage());

assertEquals(true, page.isFirstPage());

assertEquals(false, page.isLastPage());

assertEquals(false, page.isHasPreviousPage());

assertEquals(true, page.isHasNextPage());

#### 分页测试

|  |
| --- |
| @Test  **public** **void** testPageHelper() **throws** Exception {  //初始化spring容器  ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  //获得Mapper的代理对象  TbItemMapper itemMapper = applicationContext.getBean(TbItemMapper.**class**);  //设置分页信息  PageHelper.*startPage*(1, 30);  //执行查询  TbItemExample example = **new** TbItemExample();  List<TbItem> list = itemMapper.selectByExample(example);  //取分页信息  PageInfo<TbItem> pageInfo = **new** PageInfo<>(list);  System.***out***.println(pageInfo.getTotal());  System.***out***.println(pageInfo.getPages());  System.***out***.println(pageInfo.getPageNum());  System.***out***.println(pageInfo.getPageSize());  } |

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import com.taotao.manager.mapper.TbItemMapper;

import com.taotao.manager.pojo.TbItem;

import com.taotao.manager.pojo.TbItemExample;

/\*\*

\*

\* @ClassName App

\* @Description 测试PageHelper分页api

\* @Author [浅醉]

\* @Date 2019年1月4日 下午3:32:26

\*/

public class App {

public static void main(String[] args) {

System.out.println("dd");

ApplicationContext context = new ClassPathXmlApplicationContext("classpath:spring/applicationContext-dao.xml");

//获取TbItemMapper

TbItemMapper itemMapper = context.getBean(TbItemMapper.class);

//获取条件模板 :添加查询条件时使用

TbItemExample example = new TbItemExample();

//设置信息

PageHelper.startPage(1,3); //每页展示三行数据

//只有紧跟在设置分页信息后的第一个查询语句可以完成分页

List<TbItem> list = itemMapper.selectByExample(example);

//下面的不能完成分页

List<TbItem> list2 = itemMapper.selectByExample(example);

System.out.println("第一个list大小："+list.size());

System.out.println("第二个list大小："+list2.size());

//输出结果如下：可见第二个查询语句没有分页功能

//第一个list大小：3

//第二个list大小：3096

for(TbItem temp:list){

System.out.println(temp.getId());

}

//获取页面信息

PageInfo<TbItem> info = new PageInfo<TbItem>(list);

//使用PageInfo包装结果集后，可以根据其众多的方法，获取更多信息

}

}

一月 04, 2019 5:26:59 下午 com.alibaba.com.caucho.hessian.io.SerializerFactory getDeserializer

警告: Hessian/Burlap: 'com.github.pagehelper.Page' is an unknown class in WebappClassLoader

context:

delegate: false

repositories:

----------> Parent Classloader:

ClassRealm[plugin>org.apache.tomcat.maven:tomcat7-maven-plugin:2.2, parent: sun.misc.Launcher$AppClassLoader@5c647e05]

:

java.lang.ClassNotFoundException: com.github.pagehelper.Page