9月13日更新缓存模块

# Mybatis需掌握

Mapper接口配置文件

# 简单入门步骤

单myBatis连接数据库获取数据操作

第一步 导包 导入mybatis-3.2.7包下lib文件夹全部和数据库的驱动即可，不用数据源包

路径：mybatis-3.2.7\lib下的全部jar和mysql-connector-java-5.1.46.jar

第二步 在类根目录配置核心文件：mybatis-config.xml

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!—1、配置别名： 包其子包下所有类有效，作用：第三步mapper.xml下parameterType、resultType不用写全路径类名-->

<typeAliases>

<package name="com.wwl.pojo"/>

</typeAliases>

<!—2、和spring整合后 environments配置的连接数据库等都将废除 下面不用学习(了解 ) -->

<environments default="development">

<environment id="development">

<!-- 使用jdbc事务管理 -->

<transactionManager type="JDBC" />

<!—内置数据库连接池 -->

<dataSource type="POOLED">

<property name="driver" value="${jdbc.driver}" />

<property name="url"

value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8" />

<property name="username" value="root" />

<property name="password" value="root" />

</dataSource>

</environment>

</environments>

<!—3、 Mapper的位置 Mapper.xml 写Sql语句的文件的位置 -<!-- 加载映射文件 -->->

<mappers>

<!-- <mapper class="com.itheima.mybatis.mapper.UserMapper" /> -->

<package name="com.itheima.mybatis.mapper"/> //使用包方式扫描

<mapper resource="sqlmap/User.xml" /> //加载单个xml

</mappers>

</configuration>

第三步 创建mapper的XML文件 ：两种方式一般都用第一种方式，第二种方式了解

<?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">

第一种配置方式(Mapper动态代理方式)需要写接口类，类中的方法名要和语句中的id一致

//1、配置和接口类连接的属性namespac值：接口类的全路径

<mapper namespace="wwl.mapper.UsersMapper">

//2、 select查询语句，这里的id名称要和接口方法一致，如果第二步没有配置别名返回值和请求参数需要全路径名称

<select id="getUsersById" parameterType="Users" resultType="Users"> //param参数可以：pojo、基本类型’

select \* from users where userid=#{id} //#{}是占位符：id是Users类中的属性，

</select>

</mapper>

第二种配置方式 不用写接口类（了解）

<mapper namespace="users"> //因为没有接口类：这里的值随便写：防止有重名的方法

//方法id的名称需要配合调用的时候使用

<select id="getUsersById" parameterType="int" resultType="com.wwl.pojo.Users">

select \* from users where userid=#{v} #{} 占位符 值：不是pojo类这里随便写一个字母

</select>

</mapper>

第四步 调用操作 配合第三步的两种方式

public static void main(String[] args) throws Exception {

1、获取SqlSession对象

InputStream is=Resources.getResourceAsStream("mybatis-config.xml");

SqlSessionFactory s= new SqlSessionFactoryBuilder().build(is);

SqlSession sqlSession=s.openSession();// 获取sqlSession，和spring整合后由spring管理

2、调用方法操作

第一种方式的调用(重要)

UsersMapper user=sqlSession.getMapper(UsersMapper.class); //传入第三步<mapper namespace=" wwl.mapper.UsersMapper">

Users u= new Users();u.setId(15);

Users result=user.getUsersById(u); //获取到接口对象后：调用类中的方法得到返回值

System.out.println(result.getUserName()); //有值 pojo和数据库字段一致

System.out.println(result.getId()); //null 数据库的字段和pojo的字段不一致

sqlSession.close(); // 和spring整合后由spring管理

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

第二种方式的调用(了解)

Users u=sqlSession.selectOne("users.getUsersById", 15); //和第一种方式的方法不一样：直接传xml中的id名称，和值

System.out.println(u.getUserName()); //有值

System.out.println(u.getId()); //null 数据库的字段和pojo的字段不一致

}

# SSM整合步骤

第一步 导入jar包

1、mybatis3.2.7下的核心包还有lib下的全部导入和spring的整合包：mybatis-spring-1.3.2

2、springmvc是spring框架中的子框架(web层框架)所以不用整合，完美融合

3、spring的框架包：aop、jdbc(用于mybatis事务需要)、comtext、core、web、等等

4、日志Log4j：1.2.16和commons-logging-1.2还有配置文件log4j.properties(放到src下)

第二步 web中的配置

说明：如果是单一工程(不是分布式工程)监听器ContextLoaderListener不用配置，下面servlet已经都有这功能

<!-- 配置监听器ContextLoaderListener用于：配置数据源、mapper扫描、mybatis相关配置-->

<!-- 加载application-context.xml文件 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:application-\*.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 配置springmvc的servlet：用于配置扫描注册bean、mvc的注解驱动 -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<!-- springmvc配置文件加载 -->

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc-servlet.xml</param-value>

</init-param>

<!-- 让此servlet在服务启动就加载 (默认是第一次请求访问此servlet才初始化)-->

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!-- 设置成为默认的servlet(tomcat就有全局的默认寻找静态资源、后缀.jsp的解析器) -->

<url-pattern>/</url-pattern>

</servlet-mapping>

<!-- 配置post请求编码：此过滤器jar文件在spring.web的filter包下 -->

<filter>

<filter-name>uncode</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>uncode</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

第三步 springMvc-config.xml的配置

<!-- 配置注册扫描bean 此标签包含了<context:annotation-config>功能 -->

<context:component-scan base-package="com.wwl"/>

<!-- 此标签配置：3.1以后新版本注解映射器和注解适配器(不配置也可以有默认加载旧的)-->

<mvc:annotation-driven/>

<!-- 配置视图解析器进行优化：前缀和后缀。(系统默认加载的进行配置是为了优化功能) -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<!-- 配置逻辑视图的前缀 -->

<property name="prefix" value="/WEB-INF/jsp/"/>

<!-- 配置逻辑视图的后缀 -->

<property name="suffix" value=".jsp"/>

</bean>

<!-- 配置静态资源不被拦截 -->

<mvc:resources location="/img/" mapping="/img/\*\*"/>

<mvc:resources location="/js/" mapping="/js/\*\*"/>

<mvc:resources location="/css/" mapping="/css/\*\*"/>

第四步 mybatis-config.xml 的配置

1、配置mapper：由于使用spring和mybatis的整合包进行mapper扫描，这里不需要配置了

但是必须遵循：mapper.xml和mapper.java文件同名且在一个目录

2、配置别名

<!-- 1、配置别名:包其子包下所有类有效，大小写不敏感 作用:mapper.xml下parameterType、resultType不用写全路径类名-->

<typeAliases>

<package name="com.wwl.entity"/>

</typeAliases>

3、配置全局settiing的配置：根据需要添加

第五步 applicationContext-config.xml的配置

<!-- 导入properties文件 ：注意key命名要有一定的特殊规则：不要单user命名,jdbc.user -->

<context:property-placeholder location="classpath:mysql-config.properties"/>

<!-- 第一步 c3p0连接池数据源 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<property name="driverClass" value="${t.driverClass}"></property>

<property name="jdbcUrl" value="${t.url}"></property>

<property name="user" value="${t.username}"></property>

<property name="password" value="${t.password}"></property>

</bean>

<!-- 第二步 配置sqlSessionFactoryBean <!-- Mybatis的工厂 --> -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<!-- 加载mybatis的核心配置文件 -->

<property name="configLocation" value="classpath:mybatis/mybatis-config.xml"/>

</bean>

<!-- 第三步 配置Mapper动态代理开发 扫描 如果需要扫描多个包，中间使用英文逗号隔开 -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="com.wwl.mapper"/>

</bean>

---------------------------------------------------------第三步的另外两种方式：了解，更好的明白原理------------------------------------------------

<!-- Dao原始Dao 直接加载类 -->

<bean id="userDao" class="com.itheima.mybatis.dao.UserDaoImpl">

<property name="sqlSessionFactory" ref=" sqlSessionFactory "/>

</bean>

<!-- Mapper动态代理开发 扫描单个xml -->

<bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">

<property name="sqlSessionFactory" ref=" sqlSessionFactory "/>

<property name="mapperInterface" value="com.itheima.mybatis.mapper.UserMapper"/>

</bean>

---------------------------------------------------------end ----------------------------------------------------------------------------------------

以上就可以正常连接数据库不像hibernate要在事务中才能操作，但是企业开发事务还是要配置的

<!-- 注解事务：使用到的是jdbc的注解 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"/>

</bean>

<!-- 开启注解的事务支持配置 -->

<tx:annotation-driven transaction-manager="transactionManager"/>

<!—还有xml的注解事务操作 -->

# 增删改查操作

增加(insert)操作 注意获取主键问题

第一步 XML操作

<!-- 获取插入数据后的id值，一个表插入数据后，因为id主键是自增长的，没有插入需要获取到ID，可以下面的方式 -->

<insert id="insertUser" parameterType="Users">

<!-- selectKey 标签实现主键返回 -->

<!-- keyProperty：主键对应的pojo中的哪一个属性 -->

<!-- 获取主键id操作：order有两种方式：一种主键自增长方式(先数据后插入主键)AFTER，一种先数据插入后生成主键BEFORE -->

<selectKey keyProperty="id" resultType="Integer" order="AFTER">

<!-- LAST\_INSERT\_ID() 是MySql函数方法 得到刚 insert 进去记录的主键值，只适用与自增主键-->

select LAST\_INSERT\_ID()

</selectKey>

insert into user(username,birthdat,address,sex)

values(#{username},#{birthdat},#{address},#{sex})

</insert>

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

如果是BEFORE的设置返回主键id操作：

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

SELECT LAST\_INSERT\_ID()

</selectKey>

第二步 代码部分

public static void insertUser() {

try {

InputStream is = Resources.getResourceAsStream("mybatis-config.xml");

SqlSessionFactory s= new SqlSessionFactoryBuilder().build(is);

SqlSession sqlSession=s.openSession();

UsersMapper usersMapper=sqlSession.getMapper(UsersMapper.class);

Users user= new Users();

user.setUserName("曾老婆");

user.setPhone("15326584545");

usersMapper.getInsertUser(user);

//获取主键值

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

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

修改(upadte)

第一步 XML操作

<!-- 更新用户 -->

<update id="updateUserById" parameterType="cn.itcast.mybatis.pojo.User">

UPDATE `user` SET username = #{username} WHERE id = #{id}

</update>

第二步 代码部分

SqlSession sqlSession = sqlSessionFactory.openSession();

执行SqlSession对象执行更新

User user = new User(); // 创建需要更新的User

user.setId(26);

user.setUsername("关羽");

user.setBirthday(new Date());

user.setAddress("蜀国");

usersMapper.getUpdateUser(user);

查询操作

两个条件 根据性别和名字查询用户

<select id="queryUserByWhere" parameterType="user" resultType="user">

SELECT id, username, birthday, sex, address FROM `user`

<!-- where标签可以自动添加where，同时处理sql语句中第一个and关键字 -->

<where>

<if test="sex != null">

AND sex = #{sex}

</if>

<if test="username != null and username != ''">

AND username LIKE

'%${username}%'

</if>

</where>

</select>

根据多个id查询用户: 可以配合<foreach>的使用

------------------------------------------------------代码操作 --------------------------------------------------------------------------------------------

参数如果是list集合： public List<Users> selectUserByIds(List<Integer> i);

调用操作

List<Integer> lis = new ArrayList<Integer>();

lis.add(1);lis.add(15);

List<Users> result=userMapper.selectUserByIds(lis);

for(Users u :result) {

System.out.println(u.getUserName());

}

------------------------------------------------------ XMl操作--------------------------------------------------------------------------------------------

sql语句 ：select \* from users where userid in (1,2,3,….)

<!-- 查询多个id （1,2,3） -->

<select id="selectUserByIds" resultType="Users" parameterType=" Uuuu ">

<!-- foreach标签，进行遍历 -->

<!-- collection：遍历的集合，这里是QueryVo的ids属性 -->

<!-- item：遍历的项目，可以随便写，，但是和后面的#{}里面要一致 -->

<!-- open：在前面添加的sql片段 -->

<!-- close：在结尾处添加的sql片段 -->

<!-- separator：指定遍历的元素之间使用的分隔符 -->

<include refid="userSql"/> where userd in

<select id="selectUserByIds" resultType="Users" parameterType="Integer">

<include refid="userSql"/> where userId in

<foreach collection="list" item="a" separator="," open="(" close=")">// 也可以open="userid in (" close=")

#{a}

</foreach>

</select>

public List<Users> selectUserByIds(Integer[] i); 参数是数组

sql语句 ：select \* from users where userid in (1,2,3)

<!-- 查询多个id （1,2,3） -->

<select id="selectUserByIds" resultType="Users" parameterType="Integer">

select \* from users where userd in

<foreach collection="array" item="id" separator="," open="(" close=")">//最后拼接成（1,2,3）

#{id}

</foreach>

</select>

# 关联查询

一对一：一个订单对应一个用户

有两个pojo类：A和B，A类中维护了B类：

public class A(){

private B b();

}

一对多：一个用户对应多个订单

# 映射文件配置

mapper映射XML配置

根标签<mapper>配置

1、属性namespace：命名空间，用于隔离sql -->

2、还有一个很重要的作用，使用动态代理开发DAO， namespace必须和Mapper接口类路径一致 -

<mapper namespace="cn.itcast.mybatis.mapper.UserMapper">

查询标签<select>

1、属性id必须和Mapper接口方法名一致

2、属性parameterType必须和接口方法参数类型一致

3、属性resultType必须和接口方法返回值类型一致

<select id="queryUserById" parameterType="int"

resultType="cn.itcast.mybatis.pojo.User">

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

</select>

resultMap标签的配置

如果sql查询字段名和pojo的属性名不一致，可以通过resultMap将字段名和属性名作一个对应关系 ，resultMap实质上还需要将查询结果映射到pojo对象中。

resultMap可以实现将查询结果映射为复杂类型的pojo，比如在查询结果映射对象中包括pojo和list实现一对一查询和一对多查询。

<resultMap type="order" id="orderResultMap">

<!-- 定义主键 ,非常重要。如果是多个字段,则定义多个id -->

<!-- property：主键在pojo中的属性名 column：主键在数据库中的列名-->

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

<!-- 定义普通属性 -->

<result property="userId" column="user\_id" />

<result property="number" column="number" />

</resultMap>

第一种 情况 如果只有一个字段和表不一致可以只写一个

<resultMap type="order" id="orderResultMap">

<result property="userId" column="user\_id" />

</resultMap>

<sql>标签：sql片段

引入本身XML的操作

<mapper namespace="com.wwl.mapper.UsersMapper">

<sql id="userSql"> 第一步 声明一个sql片段

select \* from users

</sql>

<select id="getByUserId" parameterType="Integer" resultType="Users">

<include refid="userSql"/> 第二步 引入sql片段

where userId=#{v}

</select>

</mapper>

引入别的XML的操作：在refid前面加上对应的Mapper.xml的namespace

<include refid="com.wwl.mapper.AbcMapper.AbcSql"/

<where>语句标签和if语句标签

<!-- 根据条件查询用户 -->

<select id="queryUserByWhere" parameterType="user" resultType="user">

SELECT id, username, birthday, sex, address FROM `user`

<!-- where标签可以自动添加where，同时处理sql语句中第一个and关键字 -->

<where>

<if test="sex != null">

AND sex = #{sex}

</if>

<if test="username != null and username != ''">注意字符串类型的数据需要要做不等于空字符串校验。

AND username LIKE

'%${username}%'

</if>

</where>

</select>

<foreach>标签的使用

collection属性：遍历的集合 mybatis给我们封装了list、array属性的

item属性：遍历的项目，可以随便写，，但是和后面的#{}里面要一致

open属性：在前面添加的sql片段

close属性：在结尾处添加的sql片段

separator属性：指定遍历的元素之间使用的分隔符，如 逗号，等

<foreach collection="list" item="item" open="id IN (" close=")"

separator=",">

#{item}

</foreach>

核心配置文件 ：1数据源、2事务管理 、3找映射文件、4配置别名

配置顺序 必须是下面顺序来不然报错

properties（属性）

settings（全局配置参数）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

<mappers>加载映射文件

<mappers>

1使用mapper接口类路径：此方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中

<mapper class="com.itheima.mybatis.mapper.UserMapper" /> //指定接口类

2、注册指定包下的所有mapper接口：此方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中

<package name="com.itheima.mybatis.mapper"/>

3、使用相对于类路径的资源：此配置没有硬性要求

<mapper resource="sqlmap/User.xml" />

</mappers>

<typeAliases>配置别名

<typeAliases>

<typeAlias type="com.wwl.pojo.Users" alias="h"/> //单个匹配类

<package name="com.wwl.pojo"/>!-- 批量别名定义，扫描整个包下的类，别名为类名（大小写不敏感） -->

</typeAliases>

然后在resultType或：parameterType中就可以不用使用全类路径的名字了

# 核心类

SqlSession的使用范围

SqlSession中封装了对数据库的操作，如：查询、插入、更新、删除等。

SqlSession通过SqlSessionFactory创建。

SqlSessionFactory是通过SqlSessionFactoryBuilder进行创建。

SqlSession是一个面向用户的接口，sqlSession中定义了数据库操作方法。

每个线程都有它自己的SqlSession实例。SqlSession的实例不能共享使用，它是线程不安全的。因此最佳的范围是请求或方法范围。绝对不能将SqlSession实例的引用放在一个类的静态字段或实例字段中。

打开一个 SqlSession；使用完毕就要关闭它。通常把这个关闭操作放到 finally 块中以确保每次都能执行关闭。如下：

SqlSession session = sqlSessionFactory.openSession();

try {

// do work

} finally {

session.close();

}

SqlSessionFactoryBuilder 可以作为工具类，创建完就不需要了

SqlSessionFactoryBuilder用于创建SqlSessionFacoty，SqlSessionFacoty一旦创建完成就不需要SqlSessionFactoryBuilder了，因为SqlSession是通过SqlSessionFactory创建的。所以可以将SqlSessionFactoryBuilder当成一个工具类使用，最佳使用范围是方法范围即方法体内局部变量。

SqlSessionFactory 适用范围运行期间、重复使用、设置为单例

SqlSessionFactory是一个接口，接口中定义了openSession的不同重载方法，SqlSessionFactory的最佳使用范围是整个应用运行期间，一旦创建后可以重复使用，通常以单例模式管理SqlSessionFactory。

# 分页插件

第一步 导包 mybatis-3.4.3.jar jsqlparser-0.9.4.jar pagehelper-4.1.1.jar

注意了：mybatis版本问题，测试了很久才发现这个这三个版本合起来可以用

初始化 dialect [mysql]时出错:null

那就是sql解析工具--jsqlparser.jar4.1.0及以后版本需要0.9.4版本

http://repo1.maven.org/maven2/com/github/jsqlparser/jsqlparser/0.9.4/

4.1.0以前版本需要0.9.1版本

http://repo1.maven.org/maven2/com/github/jsqlparser/jsqlparser/0.9.1/

<https://blog.csdn.net/weixin_41382994/article/details/80205865?utm_source=copy>

第二步 配置 mybatis.config.xml

<!-- —1、配置别名： 包其子包下所有类有效，作用：第三步mapper.xml下parameterType、resultType不用写全路径类名-->

<typeAliases>

<package name="com.wwl.pojo"/>

</typeAliases>

<!-- 配置分页插件 -->

<plugins>

<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>

-------------------------------------------------网上找到的-------------------------------------------------------------------------------------------

<plugins>

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

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

<!-- 4.0.0以后版本可以不设置该参数 -->

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

<!-- 该参数默认为false -->

<!-- 设置为true时，会将RowBounds第一个参数offset当成pageNum页码使用 -->

<!-- 和startPage中的pageNum效果一样-->

<property name="offsetAsPageNum" value="true"/>

<!-- 该参数默认为false -->

<!-- 设置为true时，使用RowBounds分页会进行count查询 -->

<property name="rowBoundsWithCount" value="true"/>

<!-- 设置为true时，如果pageSize=0或者RowBounds.limit = 0就会查询出全部的结果 -->

<!-- （相当于没有执行分页查询，但是返回结果仍然是Page类型）-->

<property name="pageSizeZero" value="true"/>

<!-- 3.3.0版本可用 - 分页参数合理化，默认false禁用 -->

<!-- 启用合理化时，如果pageNum<1会查询第一页，如果pageNum>pages会查询最后一页 -->

<!-- 禁用合理化时，如果pageNum<1或pageNum>pages会返回空数据 -->

<property name="reasonable" value="false"/>

<!-- 3.5.0版本可用 - 为了支持startPage(Object params)方法 -->

<!-- 增加了一个`params`参数来配置参数映射，用于从Map或ServletRequest中取值 -->

<!-- 可以配置pageNum,pageSize,count,pageSizeZero,reasonable,orderBy,不配置映射的用默认值 -->

<!-- 不理解该含义的前提下，不要随便复制该配置 -->

<property name="params" value="pageNum=pageHelperStart;pageSize=pageHelperRows;"/>

<!-- 支持通过Mapper接口参数来传递分页参数 -->

<property name="supportMethodsArguments" value="false"/>

<!-- always总是返回PageInfo类型,check检查返回类型是否为PageInfo,none返回Page -->

<property name="returnPageInfo" value="none"/>

</plugin>

</plugins>

第三步 代码中的使用

先按没有分页的正常查询：如下面模糊查询，它是先查询完后才给你分页的

把分页步骤部署上去：参数分页的两个参数pageNo,pageSize

//分页插件的使用:共两步，第一步

PageHelper.startPage(Integer.parseInt(pageNo),Integer.parseInt(pageSize));

UsersExample example=new UsersExample();

Criteria criteria = example.createCriteria();

//设置模糊查询username

if(username ==null) {

username="%%";

}else {

username="%"+username+"%";

}

criteria.andUsernameLike(username);

List<Users> list = usersMapper.selectByExample(example);

if(!Judge.isNullAndEmpty(list)) {

//分页插件的使用:共两步，第二步,把模糊查询的结果当参数放进去

PageInfo<Users> pageInfo = new PageInfo<>(list);

List<Users> rlist = pageInfo.getList();

customResult.setListData(rlist);

customResult.setStatus(200);

}else {

customResult.setStatus(404);

}

return customResult;

# 缓存

一级缓存

工作原理：

第一次查询id为1的用户 ，先去找缓存是否有id为1的用户信息，没有从数据库查询

注意 ：如果sqlSession去执行commit操作（插入，删除，修改），会清空一级缓存（避免赃读）

第二次查询id如果有，先去找缓存，就不从数据库查

测试 ：默认支持一级缓存，不用去配置

1. 设置两次查询一样的id，进行跟踪控制台信息， 第二次，中间加入一个插入功能

# 其它

通配符 ${} #{}

#{para}引用参数的时候，Mybatis会把这个参数认为是一个字符串, 自动进行java类型和jdbc类型转换，可以防止SQL注入，如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。

${param}引用参数的时候，Mybatis会把这个参数认为是sql一部分，不会当成字符串，表示拼接sql串，不进行jdbc类型转换， ${}可以接收简单类型值或pojo属性值，如果parameterType传输单个简单类型值，${}括号中值只能是value。

使用例子 模糊查询操作

#{}操作

<!-- 如果返回多个结果，mybatis会自动把返回的结果放在list容器中 -->

<!-- resultType的配置和返回一个结果的配置一样 -->

<select id="queryUserByUsername1" parameterType="string"

resultType="cn.itcast.mybatis.pojo.User">

SELECT \* FROM `user` WHERE username LIKE #{username}

</select>

代码里面写百分号：List<Object> list = sqlSession.selectList("queryUserByUsername1", "%王%");

第二种---------------------------------------------------------------------------------------------------------------------------

直接sql语句这样写：SELECT \* FROM `user` WHERE username LIKE “%” #{username} “%”

${}操作

<!-- 如果传入的参数是简单数据类型，${}里面必须写value -->

<select id="queryUserByUsername2" parameterType="string"

resultType="cn.itcast.mybatis.pojo.User">

SELECT \* FROM `user` WHERE username LIKE '%${value}%'

</select>

MySQL ：after和before

after :自增长用：如先把数据插入数据库，然后再主键生成

before：先用UUID生成主键ID，再把数据插入数据库

CDATA的使用

CDATA 部件以"<![CDATA[" 标记开始，以"]]>"标记结束。在"<![CDATA["和"]]>"之间 的特殊字符的意义都不起作用，而转变为普通字符串内容。

在 MyBatis 的 XML 映射语句配置文件中，如果 SQL 语句有特殊字符，使用CDTA 部件括起来，如：

<select id= "selectBlog\_use\_collection"

resultMap= "blogResult" >

<![CDATA[ SELECT id , title, author\_id as authored FROM BLOG WHERE ID > 0 and ID < 10 ]]> </select>