Mybatis 的通用 mapper 的使用

# 1.导入jar包

## 1.1如果是maven项目，可以在pom.xml中加入下列：

|  |
| --- |
| <dependency>  <groupId>tk.mybatis</groupId>  <artifactId>mapper</artifactId>  <version>3.4.0</version>  </dependency> |

## 1.2如果要导入jar包的话，到下列网址下载：

<http://www.mybatis.tk/>

# 使用通用mapper

## 2.1继承通用的Mapper<T>，必须指定泛型<T>

例如下面的例子:

|  |
| --- |
| public interface UserInfoMapper extends Mapper<UserInfo>{} |

一旦继承了Mapper<T>,继承的Mapper 就拥有了Mapper<T> 所有的通用方法。

## 2.2泛型（实体类）<T> 的类型必须符合要求

实体类按照如下规则和数据库表进行转换,注解全部是JPA中的注解:

1. 表名默认使用类名,驼峰转下划线(只对大写字母进行处理),如UserInfo默认对应的表名为user\_info。

2.表名可以使用@Table(name = "tableName")进行指定,对不符合第一条默认规则的可以通过这种方式指定表名.

3.字段默认和@Column一样,都会作为表字段,表字段默认为Java对象的Field名字驼峰转下划线形式.

4.可以使用@Column(name = "fieldName")指定不符合第3条规则的字段名

5.使用@Transient注解可以忽略字段,添加该注解的字段不会作为表字段使用.

6.建议一定是有一个@Id注解作为主键的字段,可以有多个@Id注解的字段作为联合主键.

1. 默认情况下,实体类中如果不存在包含@Id注解的字段,所有的字段都会作为主键字段进行使用(这种效率极低).
2. 实体类可以继承使用,可以参考测试代码中的tk.mybatis.mapper.model.UserLogin2类.
3. 由于基本类型,如int作为实体类字段时会有默认值0,而且无法消除,所以实体类中建议不要使用基本类型.
4. @NameStyle注解，用来配置对象名/字段和表名/字段之间的转换方式，该注解优先于全局配置style，可选值：

1）normal:使用实体类名/属性名作为表名/字段名

2）camelhump:这是默认值，驼峰转换为下划线形式

3）uppercase:转换为大写

4）lowercase:转换为小写

通过[使用Mapper专用的MyBatis生成器插件](http://git.oschina.net/free/Mapper/blob/master/wiki/mapper3/7.UseMBG.md" \t "http://git.oschina.net/free/Mapper/blob/master/wiki/mapper3/_blank)可以直接生成符合要求带注解的实体类。

重点强调@Transient注解

许多人由于不仔细看文档，频繁在这个问题上出错。

如果你的实体类中包含了不是数据库表中的字段，你需要给这个字段加上@Transient注解，这样通用Mapper在处理单表操作时就不会将标注的属性当成表字段处理！

## 2.3主键策略(仅用于insert方法)

通用Mapper还提供了序列(支持Oracle)、UUID(任意数据库,字段长度32)、主键自增(类似Mysql,Hsqldb)三种方式，其中序列和UUID可以配置多个，主键自增只能配置一个。

由于MySql自增主键最常用，所以这里从最简单的配置方式开始。

### 1.@GeneratedValue(generator = "JDBC")

@Id@GeneratedValue(generator = "JDBC")private Integer id;

这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系数据库管理系统的自动递增字段）。 这种情况对应的xml类似下面这样:

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

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

values (#{username},#{password},#{email},#{bio})</insert>

### 2.@GeneratedValue(strategy = GenerationType.IDENTITY)

这个注解适用于主键自增的情况，支持下面这些数据库：

DB2: VALUES IDENTITY\_VAL\_LOCAL()

MYSQL: SELECT LAST\_INSERT\_ID()

SQLSERVER: SELECT SCOPE\_IDENTITY()

CLOUDSCAPE: VALUES IDENTITY\_VAL\_LOCAL()

DERBY: VALUES IDENTITY\_VAL\_LOCAL()

HSQLDB: CALL IDENTITY()

SYBASE: SELECT @@IDENTITY

DB2\_MF: SELECT IDENTITY\_VAL\_LOCAL() FROM SYSIBM.SYSDUMMY1

INFORMIX: select dbinfo('sqlca.sqlerrd1') from systables where tabid=1

JDBC:这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系数据库管理系统的自动递增字段）。

使用GenerationType.IDENTITY需要在全局配置中配置IDENTITY的参数值，并且需要根据数库配置ORDER属性。

举例如下：

//不限于@Id注解的字段,但是一个实体类中只能存在一个(继承关系中也只能存在一个)@Id@GeneratedValue(strategy = GenerationType.IDENTITY)private Integer id;

对应的XML形式为：

<insert id="insertAuthor">

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

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into Author

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

values

(#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})</insert>

注意<selectKey>中的内容就是IDENTITY参数值对应数据库的SQL

### 3.@GeneratedValue(generator = "UUID")

//可以用于任意字符串类型长度超过32位的字段@GeneratedValue(generator = "UUID")private String username;

该字段不会回写。这种情况对应类似如下的XML：

<insert id="insertAuthor">

<bind name="username\_bind" value='@java.util.UUID@randomUUID().toString().replace("-", "")' />

insert into Author

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

values

(#{id}, #{username\_bind}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})</insert>

注意：这种方式不能回写，如果想要回写，请看 **[通用 Mapper UUID 简单示例](http://git.oschina.net/free/Mapper/blob/master/wiki/mapper3/10.Mapper-UUID.md" \t "http://git.oschina.net/free/Mapper/blob/master/wiki/mapper3/_blank)**

### 4.Oracle使用序列

@Id@GeneratedValue(strategy = GenerationType.IDENTITY,generator = "select SEQ\_ID.nextval from dual")private Integer id;

使用Oracle序列的时候，还需要配置:

<property name="ORDER" value="BEFORE"/>

因为在插入数据库前，需要先获取到序列值，否则会报错。 这种情况对于的xml类似下面这样：

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

select SEQ\_ID.nextval from dual</selectKey>

insert into Author

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

values

(#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})</insert>

## 4. 将继承的Mapper接口添加到Mybatis配置中

非Spring项目中在mybatis配置文件中配置，如:

<mappers>

<mapper class="tk.mybatis.mapper.mapper.CountryMapper" />

<mapper class="tk.mybatis.mapper.mapper.UserInfoMapper" />

<mapper class="tk.mybatis.mapper.mapper.UserLoginMapper" />

</mappers>

Spring配置方式

如果你在Spring中配置Mapper接口,不需要像上面这样一个个配置,只需要有下面的这个扫描Mapper接口的这个配置即可:

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

<property name="basePackage" value="com.isea533.mybatis.mapper"/>

</bean>

另外因为通用接口都有顶层的接口，所以你还可以用下面的方式进行配置：

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

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

<property name="markerInterface" value="tk.mybatis.mapper.common.Mapper"/>

</bean>

这样配置后，直接继承了Mapper接口的才会被扫描，basePackage可以配置的范围更大。

如果想在Spring4中使用泛型注入，还需要包含Mapper<T>所在的包，具体请看 [在Spring4中使用通用Mapper](http://git.oschina.net/free/Mapper2/blob/master/wiki/mapper/4.Spring4.md" \t "http://git.oschina.net/free/Mapper/blob/master/wiki/mapper3/_blank)。

## 5. 代码中使用

例如下面这个简单的例子:

SqlSession sqlSession = MybatisHelper.getSqlSession();try {

//获取Mapper

UserInfoMapper mapper = sqlSession.getMapper(UserInfoMapper.class);

UserInfo userInfo = new UserInfo();

userInfo.setUsername("abel533");

userInfo.setPassword("123456");

userInfo.setUsertype("2");

userInfo.setEmail("abel533@gmail.com");

//新增一条数据

Assert.assertEquals(1, mapper.insert(userInfo));

//ID回写,不为空

Assert.assertNotNull(userInfo.getId());

//6是当前的ID

Assert.assertEquals(6, (int)userInfo.getId());

//通过主键删除新增的数据

Assert.assertEquals(1,mapper.deleteByPrimaryKey(userInfo));} finally {

sqlSession.close();}

另一个例子:

SqlSession sqlSession = MybatisHelper.getSqlSession();try {

//获取Mapper

CountryMapper mapper = sqlSession.getMapper(CountryMapper.class);

//查询总数

Assert.assertEquals(183, mapper.selectCount(new Country()));

//查询100

Country country = mapper.selectByPrimaryKey(100);

//根据主键删除

Assert.assertEquals(1, mapper.deleteByPrimaryKey(100));

//查询总数

Assert.assertEquals(182, mapper.selectCount(new Country()));

//插入

Assert.assertEquals(1, mapper.insert(country));} finally {

sqlSession.close();

}

附:Spring使用相关

直接在需要的地方注入Mapper继承的接口即可,和一般情况下的使用没有区别.

# 3.加入通用的dto、service、serviceIml、controller

## 3.1通用的dto

在写实体类的时候可以继承通用的BaseDto，如下代码：

|  |
| --- |
| public class User extends BaseDto {  @Id  private Long id;  private String username;  .......  getter、setter方法略  } |

注：要使用@Id等的注解，需要导入如下jar包:

|  |
| --- |
| <dependency>  <groupId>javax.persistence</groupId>  <artifactId>persistence-api</artifactId>  <version>1.0</version>  </dependency> |

通用的BaseDto代码如下:

|  |
| --- |
| package com.xupf.system.dto;  import java.io.Serializable;  import javax.persistence.Transient;  import javax.xml.bind.annotation.XmlTransient;  import com.fasterxml.jackson.annotation.JsonInclude;  /\*\*  \*  \* @name BaseDto  \* @description 带有标准字段的 DTO 基类.  \* @author xupf@sunshinepaper.com.cn  \* @version 1.0  \* @date 2017年4月21日  \*/  public class BaseDto implements Serializable {  /\*\*  \*  \*/  private static final long serialVersionUID = 1L;    /\*\*  \* 数据状态字段. add,update,delete  \*/  @JsonInclude(JsonInclude.Include.NON\_NULL)  @Transient  @XmlTransient  private String \_\_status;  public String get\_\_status() {  return \_\_status;  }  public void set\_\_status(String \_\_status) {  this.\_\_status = \_\_status;  }  } |

## 3.2通用的service

|  |
| --- |
| package com.xupf.system.service;  import java.util.List;  import org.springframework.transaction.annotation.Transactional;  import com.xupf.system.core.IRequest;  import com.xupf.system.core.StdWho;  /\*\*  \*  \* @name IBaseService  \* @description  \* @author xupf@sunshinepaper.com.cn  \* @version 1.0  \* @date 2017年4月21日  \*/  public interface IBaseService<T> {  List<T> select(IRequest request, T condition, int pageNum, int pageSize);  T insert(IRequest request, @StdWho T record);  T insertSelective(IRequest request, @StdWho T record);  T updateByPrimaryKey(IRequest request, @StdWho T record);  @Transactional(rollbackFor = Exception.class)  T updateByPrimaryKeySelective(IRequest request, @StdWho T record);  T selectByPrimaryKey(IRequest request, T record);  int deleteByPrimaryKey(T record);    List<T> selectAll(IRequest request);  List<T> batchUpdate(IRequest request, @StdWho List<T> list);  int batchDelete(List<T> list);  } |

需要用到的IRequest和@StdWho分别如下:

1. IRequest

|  |
| --- |
| /\*  \* #{copyright}#  \*/  package com.xupf.system.core;  import java.io.Serializable;  import java.util.Map;  import java.util.Set;  /\*\*  \* service 请求上下文.  \* <p>  \* 一些与 session 相关的数据可以放在这里面,一起传给 service 的方法.  \* <p>  \* 还可以通过一个 map 附加一些额外的属性.<br>  \* 注意,这个 map 初始为 null,当首次调用 {@code setAttribute(String)} 时初始化.  \*  \* @author shengyang.zhou@hand-china.com  \*/  public interface IRequest extends Serializable {  String FIELD\_USER\_ID = "userId";  String FIELD\_LOCALE = "locale";  String FIELD\_ROLE\_ID = "roleId";  String FIELD\_LOGIN\_ID="loginId";  String FIELD\_ALL\_ROLE\_ID = "roleIds";  String EMP\_CODE = "employeeCode";  String MDC\_PREFIX = "MDC.";  Long getUserId();  /\*\*  \* 获取扩展属性.  \*  \* @param name  \* 扩展属性名  \* @param <T>  \* 返回类型,自动匹配  \* @return 扩展属性,如果 attributeMap 尚未初始化,返回 null  \*/  <T> T getAttribute(String name);  /\*\*  \* 获取属性Map.  \*  \* @return 一个包含扩展属性的 map,有可能返回 null.(初始为 null)  \*/  Map<String, Object> getAttributeMap();  /\*\*  \* 获取属性Name.  \*  \* @return 属性名 Enumeration  \*/  Set<String> getAttributeNames();  /\*\*  \* 获取公司ID.  \*  \* @return 公司 id  \*/  Long getCompanyId();  /\*\*  \* 获取语言环境.  \*  \* @return lang code  \*/  String getLocale();  /\*\*  \* 获取角色ID.  \*  \* @return 角色 id  \*/  Long getRoleId();  Long[] getAllRoleId();  void setAllRoleId(Long[] roleIds);  /\*\*  \* 设置角色ID.  \*  \* @param accountId  \* 账户 id  \*/  void setUserId(Long accountId);  String getEmployeeCode();  void setEmployeeCode(String employeeCode);  /\*\*  \* 设置扩展属性.  \* <p>  \* 首次调用该方法时,初始化 attributeMap  \*  \* @param name  \* 属性名  \* @param value  \* 属性值  \*/  void setAttribute(String name, Object value);  /\*\*  \* 设置公司ID.  \*  \* @param companyId  \* 公司 id  \*/  void setCompanyId(Long companyId);  /\*\*  \* 设置语言环境.  \*  \* @param locale  \* 语言 code  \*/  void setLocale(String locale);  /\*\*  \* 设置角色ID.  \*  \* @param roleId  \* 角色 id  \*/  void setRoleId(Long roleId);  } |

1. @StdWho

|  |
| --- |
| /\*  \* #{copyright}#  \*/  package com.xupf.system.core;  import java.lang.annotation.ElementType;  import java.lang.annotation.Inherited;  import java.lang.annotation.Retention;  import java.lang.annotation.RetentionPolicy;  import java.lang.annotation.Target;  /\*\*  \* 加在Service 的 方法参数上.  \* <p>  \* 表示这个参数(或者Collection)中的 dto 需要自动填充 who 字段  \*  \* @author shengyang.zhou@hand-china.com  \*/  @Inherited  @Target({ ElementType.PARAMETER })  @Retention(RetentionPolicy.RUNTIME)  public @interface StdWho {  } |

## 3.3通用service的实现类

|  |
| --- |
| package com.xupf.system.service.impl;  import java.util.List;  import org.springframework.aop.framework.AopContext;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Propagation;  import org.springframework.transaction.annotation.Transactional;  import com.github.pagehelper.PageHelper;  import com.xupf.system.core.IRequest;  import com.xupf.system.dto.BaseDto;  import com.xupf.system.dto.DTOStatus;  import com.xupf.system.service.IBaseService;  import tk.mybatis.mapper.common.Mapper;  /\*\*  \*  \* @name BaseServiceImpl  \* @description  \* @author xupf@sunshinepaper.com.cn  \* @version 1.0  \* @date 2017年4月21日  \*/  @Service  public class BaseServiceImpl<T> implements IBaseService<T> {  @Autowired  private Mapper<T> mapper;    @Override  @Transactional(propagation = Propagation.SUPPORTS)  public List<T> select(IRequest request, T condition, int pageNum, int pageSize) {  PageHelper.startPage(pageNum, pageSize);  return mapper.select(condition);  }  @Override  public T insert(IRequest request, T record) {  mapper.insert(record);  return null;  }  @Override  public T insertSelective(IRequest request, T record) {  mapper.insertSelective(record);  return null;  }  @Override  public T updateByPrimaryKey(IRequest request, T record) {  mapper.updateByPrimaryKey(record);  return null;  }  @Override  public T updateByPrimaryKeySelective(IRequest request, T record) {  mapper.updateByPrimaryKeySelective(record);  return null;  }  @Override  public T selectByPrimaryKey(IRequest request, T record) {  return mapper.selectByPrimaryKey(record);  }  @Override  public int deleteByPrimaryKey(T record) {  return mapper.deleteByPrimaryKey(record);  }  @Override  public List<T> selectAll(IRequest request) {  return mapper.selectAll();  }  @Override  @Transactional(rollbackFor = Exception.class)  public List<T> batchUpdate(IRequest request, List<T> list) {  @SuppressWarnings("unchecked")  IBaseService<T> self = ((IBaseService<T>) AopContext.currentProxy());  for (T t : list) {  switch (((BaseDto) t).get\_\_status()) {  case DTOStatus.ADD:  self.insertSelective(request, t);  break;  case DTOStatus.UPDATE:  if (useSelectiveUpdate()) {  self.updateByPrimaryKeySelective(request, t);  } else {  self.updateByPrimaryKey(request, t);  }  break;  case DTOStatus.DELETE:  self.deleteByPrimaryKey(t);  break;  default:  break;  }  }  return list;  }    /\*\*  \* 默认 true,表示在 batchUpdate 中,更新操作,使用updateByPrimaryKeySelective(只更新不为 null  \* 的字段)。<br>  \* 若返回 false,则使用 updateByPrimaryKey(更新所有字段)  \*  \* @return  \*/  protected boolean useSelectiveUpdate() {  return true;  }  @Override  @Transactional(rollbackFor = Exception.class)  public int batchDelete(List<T> list) {  @SuppressWarnings("unchecked")  IBaseService<T> self = ((IBaseService<T>) AopContext.currentProxy());  int c = 0;  for (T t : list) {  c += self.deleteByPrimaryKey(t);  }  return c;  }  } |

注：

1）其中用到的分页插件，如果用的mybatis是3.3.0版本的，请一定使用如下版本的分页插件：

|  |
| --- |
| <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>4.1.6</version>  </dependency> |

1. 其中用到的DTOStatus类如下:

|  |
| --- |
| /\*  \* #{copyright}#  \*/  package com.xupf.system.dto;  /\*\*  \* @author njq.niu@hand-china.com  \*  \* 2016年1月22日  \*/  public final class DTOStatus {    private DTOStatus() {  }    /\*\*  \* Liger UI 记录状态 - 新增.  \*/  public static final String ADD = "add";    /\*\*  \* Liger UI 记录状态 - 更新.  \*/  public static final String UPDATE = "update";    /\*\*  \* Liger UI 记录状态 - 删除.  \*/  public static final String DELETE = "delete";  } |

## 3.4通用controller

|  |
| --- |
| package com.xupf.system.controller;  import javax.servlet.http.HttpServletRequest;  import org.springframework.web.bind.annotation.RestController;  import com.xupf.system.core.IRequest;  import com.xupf.system.core.RequestHelper;  /\*\*  \*  \* @name BaseController  \* @description  \* @author xupf@sunshinepaper.com.cn  \* @version 1.0  \* @date 2017年4月21日  \*/  @RestController  public class BaseController {    protected static final String DEFAULT\_PAGE = "1";  protected static final String DEFAULT\_PAGE\_SIZE = "5";  protected IRequest createRequestContext(HttpServletRequest request) {  return RequestHelper.createServiceRequest(request);  }  } |

其中用到的RequestHelper类如下：

|  |
| --- |
| package com.xupf.system.core;  import java.util.Locale;  import java.util.Map;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpSession;  import org.slf4j.MDC;  import org.springframework.web.servlet.support.RequestContextUtils;  /\*\*  \*  \* @name RequestHelper  \* @description  \* @author xupf@sunshinepaper.com.cn  \* @version 1.0  \* @date 2017年4月21日  \*/  public final class RequestHelper {    private static IRequestListener requestListener = (IRequestListener) new DefaultRequestListener();    public static IRequest createServiceRequest(HttpServletRequest httpServletRequest) {  IRequest requestContext = requestListener.newInstance();  HttpSession session = httpServletRequest.getSession(false);  if (session != null) {  requestContext.setUserId((Long) session.getAttribute(IRequest.FIELD\_USER\_ID));  // requestContext.setRoleId((Long) session.getAttribute(Role.FIELD\_ROLE\_ID));  Object roleIds = session.getAttribute(IRequest.FIELD\_ALL\_ROLE\_ID);  if (roleIds instanceof Long[]) {  requestContext.setAllRoleId((Long[]) roleIds);  }  requestContext.setEmployeeCode((String) session.getAttribute(IRequest.EMP\_CODE));  Locale locale = RequestContextUtils.getLocale(httpServletRequest);  if (locale != null) {  requestContext.setLocale(locale.toString());  }  }  Map<String, String> mdcMap = MDC.getCopyOfContextMap();  if (mdcMap != null) {  mdcMap.forEach((k, v) -> requestContext.setAttribute(IRequest.MDC\_PREFIX.concat(k), v));  }  requestListener.afterInitialize(httpServletRequest, requestContext);  return requestContext;  }  } |

其中的IRequestListener和DefaultRequestListener分别如下

1）IRequestListener

|  |
| --- |
| /\*  \* #{copyright}#  \*/  package com.xupf.system.core;  import javax.servlet.http.HttpServletRequest;  /\*\*  \* IRequestListener.  \*  \* @author shengyang.zhou@hand-china.com  \*/  public interface IRequestListener {    /\*\*  \* 负责提供 IRequest 的实例.  \*  \* @return 实例  \*/  IRequest newInstance();  /\*\*  \* 标准属性初始化完毕以后,扩展属性初始化在这里.  \*  \* @param httpServletRequest  \* HttpServletRequest  \* @param request  \* 实例  \*/  void afterInitialize(HttpServletRequest httpServletRequest, IRequest request);  } |

1. DefaultRequestListener

|  |
| --- |
| /\*  \* #{copyright}#  \*/  package com.xupf.system.core;  import javax.servlet.http.HttpServletRequest;  /\*\*  \* @author shengyang.zhou@hand-china.com  \*/  public class DefaultRequestListener implements IRequestListener {    @Override  public IRequest newInstance() {  return new ServiceRequest();  }  @Override  public void afterInitialize(HttpServletRequest httpServletRequest, IRequest request) {  }  } |

其中的ServiceRequest

|  |
| --- |
| /\*  \* #{copyright}#  \*/  package com.xupf.system.core;  import java.util.HashMap;  import java.util.Map;  import java.util.Set;  import com.fasterxml.jackson.annotation.JsonAnyGetter;  import com.fasterxml.jackson.annotation.JsonAnySetter;  import com.fasterxml.jackson.annotation.JsonIgnore;  /\*\*  \* 默认的 IRequest 实现.  \*  \* @author shengyang.zhou@hand-china.com  \*/  public class ServiceRequest implements IRequest {  private static final String ATTR\_USER\_ID = "\_userId";  private static final String ATTR\_ROLE\_ID = "\_roleId";  private static final String ATTR\_COMPANY\_ID = "\_companyId";  private static final String ATTR\_LOCALE = "\_locale";  private Long userId = -1L;  private Long roleId = -1L;  private Long[] roleIds = {};  private Long companyId = -1L;  private String locale;  private String employeeCode;  @JsonIgnore  private Map<String, Object> attributeMap = new HashMap<>();  @Override  public Long getUserId() {  return userId;  }  @Override  public void setUserId(Long userId) {  this.userId = userId;  setAttribute(ATTR\_USER\_ID, userId);  }  @Override  public String getEmployeeCode() {  return employeeCode;  }  @Override  public void setEmployeeCode(String employeeCode) {  this.employeeCode = employeeCode;  }  @Override  public String getLocale() {  return locale;  }  @Override  public void setLocale(String locale) {  this.locale = locale;  setAttribute(ATTR\_LOCALE, locale);  }  @Override  public Long getRoleId() {  return roleId;  }  @Override  public Long[] getAllRoleId() {  return roleIds;  }  @Override  public void setAllRoleId(Long[] roleIds) {  this.roleIds = roleIds;  }  @Override  public void setRoleId(Long roleId) {  this.roleId = roleId;  setAttribute(ATTR\_ROLE\_ID, roleId);  }  @Override  public Long getCompanyId() {  return companyId;  }  @Override  public void setCompanyId(Long companyId) {  this.companyId = companyId;  setAttribute(ATTR\_COMPANY\_ID, companyId);  }  @Override  @SuppressWarnings("unchecked")  @JsonAnyGetter  public <T> T getAttribute(String name) {  return (T) attributeMap.get(name);  }  @Override  @JsonAnySetter  public void setAttribute(String name, Object value) {  attributeMap.put(name, value);  }  @Override  @JsonIgnore  public Map<String, Object> getAttributeMap() {  return attributeMap;  }  @Override  @JsonIgnore  public Set<String> getAttributeNames() {  return attributeMap.keySet();  }  } |

3.5可能需要用到的ResponseData,作用是可以在controller中用它返回一个rows给前台，在前台可以直接使用rows来遍历输出数据:

|  |
| --- |
| /\*  \* #{copyright}#  \*/  package com.xupf.system.core;  import java.util.List;  import com.fasterxml.jackson.annotation.JsonInclude;  import com.fasterxml.jackson.annotation.JsonInclude.Include;  import com.github.pagehelper.Page;  /\*\*  \* 数据返回对象.  \*  \* @author njq.niu@hand-china.com  \*/  public class ResponseData {  // 返回状态编码  @JsonInclude(Include.NON\_NULL)  private String code;  // 返回信息  @JsonInclude(Include.NON\_NULL)  private String message;  //数据  @JsonInclude(Include.NON\_NULL)  private List<?> rows;  // 成功标识  private boolean success = true;  //总数  @JsonInclude(Include.NON\_NULL)  private Long total;  public ResponseData() {  }  public ResponseData(boolean success) {  setSuccess(success);  }  public ResponseData(List<?> list) {  this(true);  setRows(list);  if (list instanceof Page) {  setTotal(((Page<?>) list).getTotal());  } else {  setTotal((long) list.size());  }  }  public String getCode() {  return code;  }  public String getMessage() {  return message;  }  public List<?> getRows() {  return rows;  }  public Long getTotal() {  return total;  }  public boolean isSuccess() {  return success;  }  public void setCode(String code) {  this.code = code;  }  public void setMessage(String message) {  this.message = message;  }  public void setRows(List<?> rows) {  this.rows = rows;  }  public void setSuccess(boolean success) {  this.success = success;  }  public void setTotal(Long total) {  this.total = total;  }  } |

如在controller中使用的例子：

|  |
| --- |
| @RequestMapping("/com/user/queryAll")  @ResponseBody  public ResponseData queryAll(HttpServletRequest request,@RequestParam(defaultValue = DEFAULT\_PAGE) int page,  @RequestParam(defaultValue = DEFAULT\_PAGE\_SIZE) int pageSize){  IRequest requestContext = createRequestContext(request);  return new ResponseData(userService.selectAll(requestContext));  } |

# spring和mybatis的配置

## 4.1 spring配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:p="http://www.springframework.org/schema/p"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd">  <!-- 数据库连接池 -->  <!-- 加载配置文件 -->  <context:property-placeholder location="classpath:resources/conf.properties" />  <!-- 数据库连接池 -->  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"  destroy-method="close">  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  <property name="driverClassName" value="${jdbc.driver}" />  <property name="maxActive" value="10" />  <property name="minIdle" value="5" />  </bean>  <!-- 配置sqlsessionFactory -->  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <property name="mapperLocations" value="classpath:mapper/\*/\*.xml"></property>  <!-- <property name="plugins">  <array>  <bean class="com.github.pagehelper.PageHelper"/>  </array>  </property> -->  <property name="configLocation" value="classpath:mybatis/SqlMapConfig.xml"></property>  <property name="dataSource" ref="dataSource"></property>  </bean>  <!-- 配置扫描包，加载mapper代理对象 -->  <bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.xupf.\*.mapper"/>  <!-- 3.2.2版本新特性，markerInterface可以起到mappers配置的作用，详细情况需要看Marker接口类 -->  <!-- <property name="markerInterface" value="com.xupf.mybatis.util.MyMapper"/> -->  <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>  </bean>    <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate" scope="prototype">  <constructor-arg index="0" ref="sqlSessionFactory"/>  </bean>    <!-- 扫描包加载Service实现类 -->  <context:component-scan base-package="com.xupf.\*.service"/>    <!-- 事务管理器 -->  <bean id="transactionManager"  class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <!-- 数据源 -->  <property name="dataSource" ref="dataSource" />  </bean>    <!-- 通知 -->  <tx:advice id="txAdvice" transaction-manager="transactionManager">  <tx:attributes>  <!-- 传播行为 -->  <tx:method name="save\*" propagation="REQUIRED" />  <tx:method name="insert\*" propagation="REQUIRED" />  <tx:method name="add\*" propagation="REQUIRED" />  <tx:method name="create\*" propagation="REQUIRED" />  <tx:method name="delete\*" propagation="REQUIRED" />  <tx:method name="update\*" propagation="REQUIRED" />  <tx:method name="find\*" propagation="SUPPORTS" read-only="true" />  <tx:method name="query\*" propagation="SUPPORTS" read-only="true" />  <tx:method name="select\*" propagation="SUPPORTS" read-only="true" />  <tx:method name="get\*" propagation="SUPPORTS" read-only="true" />  </tx:attributes>  </tx:advice>    <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref="txAdvice"  pointcut="execution(\* com.xupf.\*.service.\*.\*(..))" />  </aop:config>  </beans> |

## 4.2 Mybatis配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 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>  </configuration> |

# 5.备注

如果使用它，首先注意导入的分页插件的版本为4.1.6，因为实验了很多，最后确定是用这个版本。

**其次service的实现类中的bachUpdate方法和bachDelete方法没有经过验证。请暂时不要使用**