# springmvc3+hibernate4配置+shiro配置+easyUi+ueditor

# 建立web项目

## 不要建立javaee5web项目

javaee5会与hibernate4冲突,导致下面报错不能创建和更新数据库表.

## 导入相关jar包

# 自动生成数据库表测试

## 新建daobean-context.xml文件

### 配置hibernate注解扫描范围

<!-- 配置注解扫描范围 -->

<context:component-scan base-package="org.swinglife"/>

### 配置hibernate dataSource数据源

<!-- 配置数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/test?useUnicode=true&amp;characterEncoding=UTF-8"/>

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

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

<!-- 配置其他 -->

</bean>

### 配置sessionFactory工厂hibernate4

<!-- 配置hibernate4sessionFactory -->

<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

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

<!-- 配置hibernate属性 -->

<property name="hibernateProperties">

<props>

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="hibernate.hbm2ddl.auto">update</prop>

<!-- 使用current\_session -->

<prop key="hibernate.current\_session\_context\_class">org.springframework.orm.hibernate4.SpringSessionContext

</prop>

</props>

</property>

<!-- 注解扫描 -->

<property name="packagesToScan">

<value>org.swinglife.model</value>

</property>

</bean>

### 详细文件

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

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" xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd">

<!-- 配置注解扫描范围 -->

<context:component-scan base-package="org.swinglife"/>

<!-- 配置数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/test?useUnicode=true&amp;characterEncoding=UTF-8"/>

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

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

<!-- 配置其他 -->

</bean>

<!-- 配置hibernate4sessionFactory -->

<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

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

<!-- 配置hibernate属性 -->

<property name="hibernateProperties">

<props>

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="hibernate.hbm2ddl.auto">update</prop>

<prop key="javax.persistence.validation.mode">none</prop>

</props>

</property>

<!-- 注解扫描 -->

<property name="packagesToScan">

<value>org.swinglife.model</value>

</property>

</bean>

</beans>

## 新建org.swinglife.model包存放po对象

### 建立User.java用户实例

详细文件:

@Entity

@Table(name="t\_user")

**public** **class** User {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

**private** Integer id;

**private** String username;

**private** String password;

**private** Integer isDelete;

**private** Date createDate;

/\*\*

\* 多对多用户权限表

\*

\*/

@OneToMany(mappedBy="user",cascade=CascadeType.ALL)

List<UserRole> roles;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

@Temporal(TemporalType.DATE)

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** Integer getIsDelete() {

**return** isDelete;

}

**public** **void** setIsDelete(Integer isDelete) {

**this**.isDelete = isDelete;

}

**public** Date getCreateDate() {

**return** createDate;

}

**public** **void** setCreateDate(Date createDate) {

**this**.createDate = createDate;

}

**public** List<UserRole> getRoles() {

**return** roles;

}

**public** **void** setRoles(List<UserRole> roles) {

**this**.roles = roles;

}

}

### 建立Role.java角色实例

@Entity

@Table(name="t\_role")

**public** **class** Role {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

**private** Integer id;

**private** String description;

**private** String name;

/\*\*

\* 多对一用户权限关系表

\*/

@OneToMany(mappedBy="role", cascade=CascadeType.ALL)

Set<UserRole> userRoles;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Set<UserRole> getUserRoles() {

**return** userRoles;

}

**public** **void** setUserRoles(Set<UserRole> userRoles) {

**this**.userRoles = userRoles;

}

}

### 建立UserRole用户角色关系

@Entity

@Table(name="t\_user\_role")

**public** **class** UserRole {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

**private** Integer id;

@ManyToOne(cascade=CascadeType.ALL)

@JoinColumn(name="userId")

**private** User user;

@ManyToOne(cascade=CascadeType.ALL)

@JoinColumn(name="roleId")

**private** Role role;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** User getUser() {

**return** user;

}

**public** **void** setUser(User user) {

**this**.user = user;

}

**public** Role getRole() {

**return** role;

}

**public** **void** setRole(Role role) {

**this**.role = role;

}

}

### 建立Permission权限实例

@Entity

@Table(name="t\_permission")

**public** **class** Permission {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

**private** Integer id;

**private** String description;

**private** String roleId;

**private** String token;

**private** String url;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** String getRoleId() {

**return** roleId;

}

**public** **void** setRoleId(String roleId) {

**this**.roleId = roleId;

}

**public** String getToken() {

**return** token;

}

**public** **void** setToken(String token) {

**this**.token = token;

}

**public** String getUrl() {

**return** url;

}

**public** **void** setUrl(String url) {

**this**.url = url;

}

}

## 建立测试包和测试类.Test1.java

### 进行JUnit单元测试

@Test

**public** **void** TestHib(){

//测试hibernate配置文件

ApplicationContext ac=**new** ClassPathXmlApplicationContext("daobean-context.xml");

}

如果数据库中表生成则说明测试成功.

# Dao层建立与测试

## 建立Dao层

### 新建BaseDao.java接口

创建一些方法如下

**package** org.swinglife.dao;

**import** java.util.List;

**public** **interface** BaseDao {

//添加

**public** **void** addObject(Object object);

//根据hql查询满足条件的返回一个list列表

**public** List findAllByHQL(String hql);

//根据hql和条件数组查询满足条件的数据

**public** List findAllByHQL(String hql,Object[] args);

//查询满足条件的对象

**public** Object findObjectByHQL(String hql);

**public** Object findObjectByHQL(String hql,Object[] args);

//查询满足条件的对象根据hql返回

**public** Object findObjectBySQL(String sql);

//分页查询

**public** List findPage(String hql,**int** page,**int** size);

/\*\*\*

\* 分页查询 带占位符参数

\*

\* **@param** hql

\* **@param** page

\* **@param** size

\* **@param** args

\* **@return**

\*/

**public** List findPage(String hql, **int** page, **int** size, Object[] args);

/\*\*\*

\* 删除对象

\*/

**public** **void** delObject(Object object);

/\*\*\*

\* 更新对象

\*/

**public** **void** updateObject(Object object);

/\*\*\*

\* 批量更新对象 return int

\*/

**public** **void** updateObjectByHQL(String hql);

**public** **void** updateObjectByHQL(String hql, Object[] params);

/\*\*\*

\* 通过sql查询所有

\*

\* **@param** sql

\* **@return**

\*/

**public** List findAllBySql(String sql);

}

**package** org.swinglife.dao;

**import** java.util.List;

**public** **interface** BaseDao {

//添加

**public** **void** addObject(Object object);

//根据hql查询满足条件的返回一个list列表

**public** List findAllByHQL(String hql);

//根据hql和条件数组查询满足条件的数据

**public** List findAllByHQL(String hql,Object[] args);

//查询满足条件的对象

**public** Object findObjectByHQL(String hql);

**public** Object findObjectByHQL(String hql,Object[] args);

//查询满足条件的对象根据hql返回

**public** Object findObjectBySQL(String sql);

//分页查询

**public** List findPage(String hql,**int** page,**int** size);

/\*\*\*

\* 分页查询 带占位符参数

\*

\* **@param** hql

\* **@param** page

\* **@param** size

\* **@param** args

\* **@return**

\*/

**public** List findPage(String hql, **int** page, **int** size, Object[] args);

/\*\*\*

\* 删除对象

\*/

**public** **void** delObject(Object object);

/\*\*\*

\* 更新对象

\*/

**public** **void** updateObject(Object object);

/\*\*\*

\* 批量更新对象 return int

\*/

**public** **void** updateObjectByHQL(String hql);

**public** **void** updateObjectByHQL(String hql, Object[] params);

/\*\*\*

\* 通过sql查询所有

\*

\* **@param** sql

\* **@return**

\*/

**public** List findAllBySql(String sql);

}

## 创建DaoImpl层

### 实现一些方法

**package** org.swinglife.dao.impl;

**import** java.util.List;

**import** org.hibernate.Query;

**import** org.hibernate.SessionFactory;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Repository;

**import** org.springframework.transaction.annotation.Transactional;

**import** org.swinglife.dao.BaseDao;

@Transactional

@Repository

**public** **class** BaseDaoImpl **implements** BaseDao{

@Autowired

**public** SessionFactory sessionFactory;

**public** **void** addObject(Object object) {

sessionFactory.getCurrentSession().save(object);

}

//执行hql不带条件

@SuppressWarnings({ "unchecked", "rawtypes" })

**public** List findAllByHQL(**final** String hql) {

Query query=sessionFactory.getCurrentSession().createQuery(hql);

**return** query.list();

}

//根据查询条件进行查询,

**public** List findAllByHQL(**final** String hql, **final** Object[] args) {

Query query=sessionFactory.getCurrentSession().createQuery(hql);

setParamter(query, args);

**return** query.list();

}

**public** **void** setParamter(Query query,Object[] args){

**for**(**int** i=0;args!=**null** && i<args.length;i++){

query.setParameter(i, args[i]);

}

}

**public** Object findObjectByHQL(String hql){

Query query=sessionFactory.getCurrentSession().createQuery(hql);

List list=query.list();

**if**(list.size()>0){

**return** list.get(0);

}

**return** **null**;

}

**public** Object findObjectByHQL(**final** String hql, **final** Object[] args){

Query query=sessionFactory.getCurrentSession().createQuery(hql);

setParamter(query, args);

List list=query.list();

**if**(list.size()==1){

**return** list.get(0);

}

**return** **null**;

}

**public** Object findObjectBySQL(String sql) {

Query query=sessionFactory.getCurrentSession().createSQLQuery(sql);

List list=query.list();

**if**(list.size()>0){

**return** list.get(0);

}

**return** **null**;

}

**public** List findPage(String hql, **int** page, **int** size) {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** List findPage(String hql, **int** page, **int** size, Object[] args) {

// **TODO** Auto-generated method stub

**return** **null**;

}

**public** **void** delObject(Object object) {

// **TODO** Auto-generated method stub

}

**public** **void** updateObject(Object object) {

// **TODO** Auto-generated method stub

}

**public** **void** updateObjectByHQL(String hql) {

// **TODO** Auto-generated method stub

}

**public** **void** updateObjectByHQL(String hql, Object[] params) {

// **TODO** Auto-generated method stub

}

**public** List findAllBySql(String sql) {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

### hibernate4 sessionFactory补充说明

1 hibernate4中直接在baseDao层使用sessionFactory

2 在没有创建事务的前提下,只能使用openSession()方法

3 如果要使用getCurrentSession()则需要在配置文件中创建事务,并将baseDaoImpl交给事务处理.

## 修改daobean-context.xml

### 增加事务配置

<!-- 事务管理 -->

<bean id="transactionManager"

class="org.springframework.orm.hibernate4.HibernateTransactionManager">

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

</bean>

<!-- 注解管理事务 -->

<tx:annotation-driven />

## 创建dao层测试方法

### add测试

@Test

**public** **void** TestDao(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("daobean-context.xml");

BaseDao baseDao=(BaseDao) ac.getBean("baseDaoImpl");

User user=**new** User();

user.setUsername("张三");

user.setPassword("123456");

baseDao.addObject(user);

}

查看数据库是否新增数据

### select测试

@Test

**public** **void** TestDaoSelect(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("daobean-context.xml");

BaseDao baseDao=(BaseDao) ac.getBean("baseDaoImpl");

List<User> list=baseDao.findAllByHQL("from User");

**for**(**int** i=0;i<list.size();i++){

System.out.println(list.get(i).getUsername());

}

}

# Service层建立与测试

## 创建service层接口包.

### 创建接口AccountService.java并创建方法

**public** **interface** AccountService{

**public** User getUserByUserName(String username);

### }创建serviceImpl层

### 创建AccountService实现类

@Service

**public** **class** AccountServiceImpl **implements** AccountService{

@Autowired

BaseDao dao;

/\*\*

\* 通过用户名查询对象

\* **@param** username

\* **@author** me

\*/

**public** User getUserByUserName(String username){

User user=(User)dao.findObjectByHQL("from User where username=?", **new** Object[]{ username });

**return** user;

}

}

### 测试类中创建service测试方法

@Test

**public** **void** TestService(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("daobean-context.xml");

AccountService accountService=(AccountService) ac.getBean("accountServiceImpl");

User user=accountService.getUserByUserName("aaa1");

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

}

注意:这里找的是单个对象,数据库中记录应该只对应一条才能顺利测试.

# springmvcweb层建立和测试

## src目录下新建mvc-context.xml配置文件

### beans头文件

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.2.xsd ">

### beans:beans头文件

<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/mvc"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.springframework.org/schema/beans"

xmlns:p="http://www.springframework.org/schema/p" xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd 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>">

### 配置springmvc注解扫描范围

<!-- 注解扫描 -->

<context:component-scan base-package="org.swinglife.controller"></context:component-scan>

### 支持spring3.0新的mvc注解

<!-- 默认的注解映射的支持，自动注册DefaultAnnotationHandlerMapping和AnnotationMethodHandlerAdapter -->

<mvc:annotation-driven />

### 配置静态资源不进行springmvc过滤

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

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

### 配置视图解析器

<!-- 配置模型视图解析器 -->

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

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

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

<!-- 如果使用jstl的话，配置下面的属性 -->

<property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />

</bean>

### web.xml文件中配置springmvc监听器

<!-- Spring MVC 上下文监听 -->

<servlet>

<servlet-name>MVC</servlet-name>

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

<init-param>

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

<param-value>

/WEB-INF/classes/mvc\*.\*,/WEB-INF/classes/daobean\*.\*

</param-value>

</init-param>

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

</servlet>

<!-- Spring MVC 普通页面拦截 -->

<servlet-mapping>

<servlet-name>MVC</servlet-name>

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

</servlet-mapping>

### request作用域监听器配置

<listener>

<listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>

</listener>

### spring字符编码处理过滤器

<filter>

<filter-name>CharactorEncodingFilter</filter-name>

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

<init-param>

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

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharactorEncodingFilter</filter-name>

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

</filter-mapping>

### 创建Controller包和LoginController控制器

简单的配置

@Controller

**public** **class** LoginController {

@RequestMapping(value = "toLogin")

**public** String toLogin(){

**return** "login";

}

}

Web-INF下常见page文件夹并创建login.jsp

浏览器访问<http://localhost:8080/pro_shiro2/toLogin>出现测试页面则测试成功.

## Controller中测试数据

### 修改LoginController.java

@Controller

**public** **class** LoginController {

@Autowired

AccountService accountService;

@RequestMapping(value = "toLogin")

**public** String toLogin(HttpServletRequest request){

User user=accountService.getUserByUserName("aaa1");

request.setAttribute("user",user );

**return** "login";

}

}

页面中加入el表达式${requestScope.user.username }

如果输出结果,则表示成功.

# 级联操作和懒加载测试

## 多对多级联保存测试

role

1 管理员

2 开发人员

userRole

1 1 1

1 1 2

user

1 李四

//级联保存测试

@Test

**public** **void** TestSaveAll(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("daobean-context.xml");

BaseDao baseDao=(BaseDao) ac.getBean("baseDaoImpl");

//建立用户

User user=**new** User();

user.setId(1);

user.setUsername("李四");

user.setPassword("1234");

//建立角色

Role role=**new** Role();

role.setId(1);

role.setName("管理员");

Role role2=**new** Role();

role2.setId(2);

role2.setName("开发人员");

//新建角色权限对象

//建立一个关系

UserRole userRole=**new** UserRole();

userRole.setRole(role);

userRole.setUser(user);

UserRole userRole2=**new** UserRole();

userRole2.setRole(role2);

userRole2.setUser(user);

List<UserRole> list=**new** ArrayList<UserRole>();

list.add(userRole);

list.add(userRole2);

user.setRoles(list);

//保存role

baseDao.addObject(role);

baseDao.addObject(role2);

//保存User

baseDao.addObject(user);

}

1多对多的级联保存需要先保存一的一方,再保存加入了关系的乙方.这样数据库中关系表才不会出现sql错误

2级联保存需要在有关系的地方设上cascade=CascadeType.ALL,all属性为简单的配置.如:

@OneToMany(mappedBy="user",cascade=CascadeType.ALL)

List<UserRole> roles;

@OneToMany(mappedBy="role", cascade=CascadeType.ALL)

Set<UserRole> userRoles;

@ManyToOne(cascade=CascadeType.ALL)

@JoinColumn(name="userId")

**private** User user;

@ManyToOne(cascade=CascadeType.ALL)

@JoinColumn(name="roleId")

**private** Role role;

## 级联删除测试.

### 需求:删除用户的时候将用户角色关系也删除

先删除关系,在删除用户

//级联删除

@Test

**public** **void** deleteAll(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("daobean-context.xml");

BaseDao baseDao=(BaseDao) ac.getBean("baseDaoImpl");

//创建一个用户

User user =**new** User();

user.setId(1);

user.setUsername("李四");

user.setPassword("1234");

String hql="delete UserRole o where o.user.id=1";

baseDao.deleteByHql(hql);

baseDao.delObject(user);

}

## 配置web层中进行级联查询

### 实现方法:延迟session关闭时间至view层

### 去掉daobean-context.xml在springmvc中的加载

### web.xml中更改daobean-context.xml在spring中加载

1 创建spring上下文加载

<context-param>

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

<param-value>classpath:daobean-context.xml</param-value>

</context-param>

2 监听servletContext,启动contextConfigLocation中spring的配置

<!-- 监听servletContext,启动contextConfigLocation中的spring配置信息 -->

<listener>

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

</listener>

### web.xml中增加OpenSessionInViewFilter配置解决懒加载问题

<!-- 使用spring解决懒加载问题 -->

<filter>

<filter-name>OpenSessionInViewFilter</filter-name>

<filter-class>org.springframework.orm.hibernate4.support.OpenSessionInViewFilter</filter-class>

<init-param>

<param-name>singleSession</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>OpenSessionInViewFilter</filter-name>

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

</filter-mapping>

### web配置遇到的问题:

spring配置文件必须使用spring上下文加载的方式.而不能使用springmvc加载配置文件.

否则OpenSessionInViewFilter将不能使用.会报错.

### 增加AccoiuntService接口方法

//根据用户姓名查询用户的所有权限

**public** List<Role> getUserRole(String username);

### AccoiuntServiceImpl实现方法如下

**public** List<Role> getUserRole(String username) {

User user=(User) dao.findObjectByHQL("from User where username=?",**new** Object[]{username} );

List<Role> list=**new** ArrayList<Role>();

**for**(**int** i=0;i<user.getRoles().size();i++){

Role role=user.getRoles().get(i).getRole();

list.add(role);

}

**return** list;

}

### 修改LoginController

List<Role> list=accountService.getUserRole("李四");

request.setAttribute("role", list);

**return** "login";

如果页面读取到数据,说明.懒加载已经成功取消

### 增加daobean-context.xml数据源配置项

<!--解决方法间调用的懒加载问题 -->

<prop key="hibernate.enable\_lazy\_load\_no\_trans">true</prop>

<!-- 使用current\_session -->

<prop key="hibernate.current\_session\_context\_class">org.springframework.orm.hibernate4.SpringSessionContext

# log4j配置

## 新建log4j.properties文件内容如下,

# Global logging configuration，建议开发环境中要用debug

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

## web.xml文件中增加log4j配置

<!-- log4j配置 -->

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>/WEB-INF/classes/log4j.properties</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

# daobean-context.xml改进

## 新建jdbc.properties增加内容如下

driverName=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf-8

name=root

password=root

## bean-context.xml中引入jdbc.properties配置文件

<!-- 加载jdbc资源文件 -->

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

## 修改数据源如下

<!-- 设置数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName" value="${driverName}"></property>

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

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

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

<property name="maxActive" value="30"></property>

<property name="maxIdle" value="10"></property>

<property name="minIdle" value="5"></property>

<property name="maxWait" value="5000"></property>

<property name="timeBetweenEvictionRunsMillis" value="30000"></property>

<property name="numTestsPerEvictionRun" value="16"></property>

</bean>

## 事务管理配置详细配置

<!-- 事务管理 -->

<bean id="transactionManager"

class="org.springframework.orm.hibernate4.HibernateTransactionManager">

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

</bean>

<!-- 注解管理事务 -->

<tx:annotation-driven />

<!-- 配置事务增强处理Bean，指定事务管理器 -->

<tx:advice id="transactionAdvice" transaction-manager="transactionManager">

<!-- 配置详细事务处理语义 -->

<tx:attributes>

<tx:method name="save\*" propagation="REQUIRED" />

<tx:method name="add\*" propagation="REQUIRED" />

<tx:method name="create\*" propagation="REQUIRED" />

<tx:method name="insert\*" propagation="REQUIRED" />

<tx:method name="find\*" propagation="SUPPORTS" read-only="true" />

<!-- 其他采用默认事务方式 -->

<tx:method name="\*" />

</tx:attributes>

</tx:advice>

<!-- Spring aop事务管理 -->

<aop:config>

<!-- 配置切入点 -->

<aop:pointcut id="transactionPointcut"

expression="execution(\* org.swinglife.service.impl.\*.\*(..))" />

<!-- 指定在txAdvice切入点应用txAdvice事务增强处理 -->

<aop:advisor pointcut-ref="transactionPointcut"

advice-ref="transactionAdvice" />

</aop:config>

# Shiro权限控制管理配置

## 新建shiro-context.xml配置文件

### 头文件如下

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

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" xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd

http://www.springframework.org/schema/aop <http://www.springframework.org/schema/aop/spring-aop.xsd>">

### 配置shiro的web过滤器

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

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

<property name="loginUrl" value="/toLogin" />

<property name="successUrl" value="/home" />

<property name="unauthorizedUrl" value="/403.do" />

<!-- 过滤器链定义 ,从上向下顺序执行,一般将/\*\*放到最下面-->

<property name="filterChainDefinitions">

<value>

/myjsp = authc <!-- authc 表示需要认证的链接地址 -->

/home = authc, perms[/home] <!-- perms 表示需要该权限才能访问的链接 -->

</value>

</property>

</bean>

文件说明: loginUrl表示统一的认证地址

successUrl表示认证成功后统一登录地址

securityManager 注入的securityManager工厂

unauthorizedUrl 表示没有该权限的跳转链接地址

filterChainDefinitions 过滤器链,从上向下执行

authc 表示需要认证的地址

perms[/home] 表示需要该权限的地址./home表示权限名,一般从数据库读取权限名

### 配置securityManager工厂

<!-- securityManager配置 -->

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<!-- 配置的自定义的realm -->

<property name="realm" ref="myShiroRealm"></property>

</bean>

### 自定义realm配置

<!-- 自定义realm -->

<bean id="myShiroRealm" class="org.swinglife.shiro.MyShiroRealm"></bean>

注意点:realm中需要使用到service层.如果没用注解需要配置service注入到自定义的realm中,如果使用注解.在其他配置文件中如果没有扫描到该包,需要在配置中添加注解扫描,如果其他配置文件有扫描,.则不需要

<!-- 注解扫描 -->

<context:component-scan base-package="org.swinglife.shiro"></context:component-scan>

## 自定义shiro实现

### 新建包org.swinglife.shiro以及MyShiroRealm.java文件

**public** **class** MyShiroRealm **extends** AuthorizingRealm {

@Autowired

**private** AccountService accountService;

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection pc) {

//用户授权

String username = (String) pc.fromRealm(getName()).iterator().next();

**if** (username != **null**) {

List<String> pers = accountService.getPermissionsByUserName(username);

**if** (pers != **null** && !pers.isEmpty()) {

SimpleAuthorizationInfo info = **new** SimpleAuthorizationInfo();

**for** (String each : pers) {

info.addStringPermission(each);

}

**return** info;

}

}

**return** **null**;

}

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken at) **throws** AuthenticationException {

UsernamePasswordToken token = (UsernamePasswordToken) at;

//用户凭证

String username = token.getUsername();

**if** (username != **null** && !"".equals(username)) {

//service层取出user

User user = accountService.getUserByUserName(username);

**if** (user != **null**) {

System.out.println("user:" + user);

**return** **new** SimpleAuthenticationInfo(user.getUsername(), user.getPassword(), getName());

}

}

**return** **null**;

}

}

## LoginController中增加登录的处理和统一认证处理

@Controller

**public** **class** LoginController {

@Autowired

AccountService accountService;

@RequestMapping(value = "toLogin")

**public** String toLogin(HttpServletRequest request){

**return** "login";

}

@RequestMapping(value = "login", method = RequestMethod.POST)

**public** String Login(String username, String password) {

User user = accountService.getUserByUserName(username);

**if** (user == **null**) {

**return** "error";

}

**if** (!user.getPassword().equals(password)) {

**return** "error";

}

// 登录后存放进shiro token

UsernamePasswordToken token = **new** UsernamePasswordToken(user.getUsername(), user.getPassword());

Subject subject = SecurityUtils.getSubject();

//验证身份和授权信息

subject.login(token);

System.out.println("登录成功");

**return** "redirect:/home";

}

}

## web.xml中增加shiro的过滤器

<filter>

<filter-name>shiroFilter</filter-name>

<filter-class>

org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

</filter>

<filter-mapping>

<filter-name>shiroFilter</filter-name>

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

</filter-mapping>

## web.xml中增加shiro-context.xml加载

<context-param>

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

<param-value>

classpath:daobean-context.xml,classpath:shiro-context.xml

</param-value>

</context-param>

## WEB-INF下page页面下新建login.jsp以及home.jsp内容如下

<%@ page language="java" contentType="text/html; charset=GB18030"

pageEncoding="GB18030"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=GB18030"*>

<title>Insert title here</title>

</head>

<body>

<h1>user login</h1>

<form action=*"login"* method=*"post"*>

username:<input type=*"text"* name=*"username"*><p>

password:<input type=*"password"* name=*"password"*>

<p>

<input type=*"submit"* value=*"submit"*>

</form>

</body>

</html>

**home.jsp**

<%@ page language="java" contentType="text/html; charset=GB18030"

pageEncoding="GB18030"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=GB18030"*>

<title>home</title>

</head>

<body>

<h1>success</h1>

</body>

</html>

## 测试

输入<http://localhost:8080/shiro/home如果跳转到login.jsp>说明拦截成功

再输入<http://localhost:8080/shiro/myjsp是否也会跳转到login.jsp>

## 原理:

### 这是一个简单的shiro配置案例

在shiro拦截链中的请求会根据shiro的配置进行相应的处理.

没有进行登陆时请求的如果在shiro拦截下都会跳转到需要认证的页面

这个例子 点击的请求并没设置在shiro的过滤器中,

调用相应方法后再方法中

创建token信息再取得subject进行login方法调用自定义的realm进行认证和

授权的操作.如果 subject.login(token);并没有出现异常则会继续回到

Controller中继续执行再进行相应的操作,返回视图信息或者其他校验

### 流程图说明

subject.login(token);

Controller

http: /login

Y

抛出

N

没有异常

MyShiroRealm

认证成功后认证信息会被保存在session中,下次访问不会再会跳转到登陆页面

# mvc-config.xml配置增强

## 解决spring3.2 使用@ResponseBody返回值为json或者xml的错误

<bean id="contentNegotiationManager" class="org.springframework.web.accept.ContentNegotiationManagerFactoryBean">

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

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

<!-- 表示不开启AcceptHeader 来解决json返回406的错误 -->

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

<property name="mediaTypes">

<value>

atom=application/atom+xml

html=text/html

json=application/json

\*=\*/\*

</value>

</property>

</bean>

## 解决json乱码

<!-- 解决json乱码以及springmvc支持 -->

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="cacheSeconds" value="0" />

<property name="messageConverters">

<list>

<bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"></bean>

<bean

class="org.springframework.http.converter.StringHttpMessageConverter">

<property name="supportedMediaTypes">

<list>

<value>text/html;charset=UTF-8</value>

</list>

</property>

</bean>

</list>

</property>

</bean>

## json传参

### 新建Controller AjaxTestController.java

@Controller

**public** **class** AjaxController {

@Autowired

AccountService accountService;

@RequestMapping(value="ajaxtest",method=RequestMethod.GET)

**public** @ResponseBody List<Role> AjaxTest(String username){

//根据页面传过来的username查询所有角色信息

System.out.println("Ajax传参为:"+username);

// User user=accountService.getUserByUserName(username);

// //序列化user

// user=accountService.getUserToVo(user);

List<Role> list=accountService.getUserRole(username);

List<Role> listRole=accountService.getRoleToVo(list);

**return** listRole;

}

}

### 关于json序列化问题

hibernate的多表查询会导致json的序列化问题,

**解决办法1**: (自己想的)将所要取的对象进行重新赋值序列化 即转换成Vo对象.

List也要进行序列化,通俗的说就是所有的对象都要是一个新的对象返回回给json

getRoleToVo(list) 方法如下

**public** List<Role> getRoleToVo(List<Role> list) {

List<Role> listRole=**new** ArrayList<Role>();

Role role=**null**;

**for**(**int** i=0;i<list.size();i++){

role=**new** Role();

role.setName(list.get(i).getName());

listRole.add(role);

}

**return** listRole;

}

**解决办法2:**在要查询出来的传值给json的对象类上加上

@JsonIgnoreProperties(value={"userRoles"}) 注解userRoles表示一对多多对一的对象

public class Role implements java.io.Serializable{

json会帮我们过滤掉userRoles这个hibernate一对多 多对1的产物

### 新建jsp页面json.jsp

<%@ page language="java" import="java.util.\*" pageEncoding="utf-8"%>

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<base href="<%=basePath%>">

<title>My JSP 'index.jsp' starting page</title>

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<script type=*"text/javascript"*>

**function** createAjaxObj(){

**var** req;

**if**(window.XMLHttpRequest){

req = **new** XMLHttpRequest();

}**else**{

req = **new** ActiveXObject("Msxml2.XMLHTTP"); //ie

}

**return** req;

}

**function** sendAjaxReq(){

**var** req = createAjaxObj();

req.open("get","ajaxtest?username=admin");

req.setRequestHeader("accept","application/json");

req.onreadystatechange = **function**(){

alert(req.responseText);

eval("var result="+req.responseText);

document.getElementById("div1").innerHTML=result[0].name;

};

req.send(**null**);

}

</script>

</head>

<body>

<a href=*"javascript:void(0);"* onclick="sendAjaxReq()">测试</a>

<div id=*"div1"*></div>

</body>

</html>

如果*div1*出现值表示测试成功..

### 传值时间问题解决办法

**var** da=parseInt(result.createDate);

**var** date =**new** Date(da);

alert(date);

document.getElementById("div1").innerHTML=date.toLocaleString()

# BaseDao层改造

## 对BaseDao层进行泛型改造

# 配置hibernate4 ehcache配置

## jdbc.properties中增加属性

hibernate.cache.use\_second\_level\_cache = true

hibernate.cache.use\_query\_cache = true

hibernate.cache.region.factory\_class = org.hibernate.cache.ehcache.EhCacheRegionFactory

hibernate.cache.provider\_configuration\_file\_resource\_path = ehcache.xml

## 增加ehcache.xml文件

<?xml version="1.0" encoding="UTF-8"?>

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">

<diskStore path="D:/ehcache" />

<!-- DefaultCache setting. -->

<defaultCache

maxElementsInMemory="1000"

eternal="false"

timeToIdleSeconds="120"

timeToLiveSeconds="120"

maxElementsOnDisk="1000000"

overflowToDisk="true"

memoryStoreEvictionPolicy="LRU">

</defaultCache>

<!-- Special objects setting. -->

<cache

name="org.andy.work.entity.AcctUser"

maxElementsInMemory="2"

memoryStoreEvictionPolicy="LRU"

eternal="true"

diskPersistent="false"

overflowToDisk="false"

maxElementsOnDisk="1000000" />

</ehcache>

# easyUi配置json传参

## 设置springmvc不拦截某些请求.因为/的配置会拦截除了.jsp外的所有请求

<mvc:resources location=*"/easyUi/"* mapping=*"/easyUi/\*.html"*/>

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

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

## 新建text.html文件如下

<!DOCTYPE html>

<html>

<head>

<meta http-equiv=*"content-type"* content=*"text/html; charset=UTF-8"*>

<script type=*"text/javascript"*

src=*"../js/jquery-easyui-1.3/jquery-1.7.2.min.js"*></script>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/jquery.easyui.min.js"*></script>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/icon.css"* type=*"text/css"*></link>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/default/easyui.css"* type=*"text/css"*></link>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/locale/easyui-lang-zh\_CN.js"*></script>

<script type=*"text/javascript"* src=*"../js/myjs.js"*></script>

<script type=*"text/javascript"*>

**var** loginAndRegDialog;

$(**function**(){

loginAndRegDialog=$('#loginAndRegDialog').dialog({

closable : **false**,//把右上角叉去掉

modal :**true**,//显示背景灰色

buttons:

[

{text:'注册',

handler:**function**(){

}},

{text:'登陆',

handler:**function**(){

console.info("我点击的是按钮");

$.ajax({

url : sy.bp() + '/easyUiJson',

data : {

name: $('#loginInputForm input[name=name]').val(),

password:$('#loginInputForm input[name=password]').val()

},

cache:**false**,

dataType:'json',

success:**function**(r){

// console.info(r.username);

$.messager.alert('登陆成功','欢迎:'+r.username);

}

});

}}

]

});

});

</script>

</head>

<body>

This is my HTML page. <br>

<div id=*"loginAndRegDialog"* title=*"用户登陆"* style="width:*250px*;height:*200px*;">

<form action=*""* method=*"post"* id=*"loginInputForm"*>

<table>

<tr>

<th align=*"right"*>用户名:</th>

<td><input type=*"text"* name=*"name"* /></td>

</tr>

<tr>

<th align=*"right"*>密码:</th>

<td><input type=*"password"* name=*"password"* /></td>

</tr>

</table>

</form>

</div>

</body>

</html>

### 简单传值方法

data : {

name: $('#loginInputForm input[name=name]').val(),

password:$('#loginInputForm input[name=password]').val()

更改为

data : $('#loginInputForm').serialize(),

## myjs.js中写一个获取项目路径的方法

**function** sy(){ }

sy.bp = **function**(){

**var** curWwwpath=window.document.location.href;

**var** pathName= window.document.location.pathname;

**var** pos=curWwwpath.indexOf(pathName);

**var** localhostPath=curWwwpath.substring(0,pos);

**var** projectName = pathName.substring(0,pathName.substr(1).indexOf('/')+1);

**return** (localhostPath + projectName);

};

## Controller中如下配置

@RequestMapping(value="easyUiJson")

@ResponseBody

**public** User login(String name,String password,HttpServletRequest request){

System.*out*.println("name="+name);

System.*out*.println("password="+password);

User user =**new** User();

user.setUsername("小明");

user.setPassword("11112134");

**return** user;

}

}

查看是否输出 你好,小明

# easyUi表单传参

## Controller:

@RequestMapping(value="uiform")

@ResponseBody

**public** User loginForm(String name,String password){

System.*out*.println("name="+name);

System.*out*.println("password="+password);

User user=accountService.getUserByUserName(name);

**if**(user!=**null**){

**return** user;

}

**return** **null**;

}

## html中:

<!DOCTYPE html>

<html>

<head>

<meta http-equiv=*"content-type"* content=*"text/html; charset=UTF-8"*>

<script type=*"text/javascript"*

src=*"../js/jquery-easyui-1.3/jquery-1.7.2.min.js"*></script>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/jquery.easyui.min.js"*></script>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/icon.css"* type=*"text/css"*></link>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/default/easyui.css"* type=*"text/css"*></link>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/locale/easyui-lang-zh\_CN.js"*></script>

<script type=*"text/javascript"* src=*"../js/myjs.js"*></script>

<script type=*"text/javascript"*>

**var** loginAndRegDialog;

$(**function**(){

loginAndRegDialog=$('#loginAndRegDialog').dialog({

closable : **false**,//把右上角叉去掉

modal :**true**,//显示背景灰色

buttons:

[

{text:'注册',

handler:**function**(){

}},

{text:'登陆',

handler:**function**(){

//提交

loginInputForm.submit();

}}

]

});

loginInputForm=$('#loginInputForm').form({

url: sy.bp()+ '/uiform',

onSubmit: **function**(){

console.info($('#loginInputForm input[name=name]').val());

**if**($('#loginInputForm input[name=name]').val()==""){

$.messager.alert("提示","用户名不能为空");

**return false**;

}

**if**($('#loginInputForm input[name=password]').val()==""){

$.messager.alert("提示","密码不能为空");

**return false**;

}

},

success: **function**(data){

console.info(data);

//需要将data类型转化成object类型

**var** obj = jQuery.parseJSON(data);

**if**(obj!=**null**){

$.messager.alert("登陆成功","欢迎你:"+obj.username);

}

**if**(obj==**null**){

$.messager.alert("提示","您的输入有错误!");

}

}

});

});

</script>

</head>

<body>

This is my HTML page. <br>

<div id=*"loginAndRegDialog"* title=*"用户登陆"* style="width:*250px*;height:*200px*;">

<form method=*"post"* id=*"loginInputForm"*>

<table>

<tr>

<th align=*"right"*>用户名:</th>

<td><input type=*"text"* name=*"name"* /></td>

</tr>

<tr>

<th align=*"right"*>密码:</th>

<td><input type=*"password"* name=*"password"* /></td>

</tr>

</table>

</form>

</div>

</body>

</html>

## 说明:

1 使用form表单提交需要注意将返回的字符类型转换成Object对象才能使用.

2 上面使用json方式,因为用了dataType:'json'所以已经转换成字符串的类型

3 转换对象推荐使用**var** obj = jQuery.parseJSON(data);这种方式,这样在返回为空的时候也能输出null

4 如果使用eval(“(“data”)”)当返回的对象为空时,eval不能正常处理字符串格式.

# easyUi中ValidateBox的使用

## 如何添加一个ValidateBox

<tr>

<th align=*"right"*>密码:</th>

<td><input type=*"password"* name=*"password"* class=*"easyui-validatebox"* required=*"true"* /></td>

</tr>

推荐使用class添加的方法required表示必须填写

## form表单中增加验证

在表单提交前方法中判断对ValidateBox的验证

onSubmit: **function**(){

**if**(!$('#loginInputForm').form('validate')){

**return false**;

}

},

## ajax中如何使用判断

使用 $('#loginInputForm').form('validate'方法

**if**( $('#loginInputForm').form('validate'))

{

$.ajax({

url: sy.bp()+ '/easyUiJson',

data: $('#loginInputForm').serialize(),

dataType:'json',

success: **function**(r){

**if**(r!=**null**){

$.messager.alert('登陆成功','欢迎:'+r.username);

}**else**

$.messager.alert('提示','您的输入有误');

}

});

}

## 如何自定义实现重复密码校验

### js中添加一个方法

$.extend($.fn.validatebox.defaults.rules, {

eqPassword: {

validator: **function**(value, param){

**return** value==$(param[0]).val();

},

message: '密码不一致!'

}

});

### 重复密码中自定义一个类型validType

<tr>

<th align=*"right"*>密码:</th>

<td><input type=*"password"* name=*"password"* class=*"easyui-validatebox"* required=*"true"* /></td>

</tr>

<tr>

<th align=*"right"*>重复密码:</th>

<td><input type=*"password"* name=*"passwordre"* class=*"easyui-validatebox"* required=*"true"* validType=*"eqPassword['#loginInputForm input[name=password]']"*/></td>

</tr>

validType=*"eqPassword['#loginInputForm input[name=password]']"即可以实现*

## 如何添加回车自动跳转未输入项

增加一个方法

loginInputForm.find('input').on('keyup',**function**(event){

**if**(event.keyCode=='13'){

loginInputForm.submit();

}

});

写在里面

## 注意区分javascript中写的loginInputForm

# 注册的实现和Layout的使用

<!DOCTYPE html>

<html>

<head>

<meta http-equiv=*"content-type"* content=*"text/html; charset=UTF-8"*>

<script type=*"text/javascript"*

src=*"../js/jquery-easyui-1.3/jquery-1.7.2.min.js"*></script>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/jquery.easyui.min.js"*></script>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/icon.css"* type=*"text/css"*></link>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/default/easyui.css"* type=*"text/css"*></link>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/locale/easyui-lang-zh\_CN.js"*></script>

<script type=*"text/javascript"* src=*"../js/myjs.js"*></script>

<script type=*"text/javascript"*>

**var** loginAndRegDialog;

**var** loginInputForm;

**var** regDialog;

$(**function**(){

loginAndRegDialog=$('#loginAndRegDialog').dialog({

closable : **false**,//把右上角叉去掉

modal :**true**,//显示背景灰色

buttons:

[

{text:'注册',

handler:**function**(){

regDialog.dialog('open');

}},

{text:'登陆',

handler:**function**(){

//提交

loginInputForm.submit();

}}

]

});

loginInputForm=$('#loginInputForm').form({

url: sy.bp()+ '/uiform',

onSubmit: **function**(){

**if**(!$('#loginInputForm').form('validate')){

**return false**;

}

},

success: **function**(data){

console.info(data);

//需要将data类型转化成object类型

**var** obj = jQuery.parseJSON(data);

**if**(obj!=**null**){

$.messager.alert("登陆成功","欢迎你:"+obj.username);

$.messager.show({

title: '标题',

msg: '登陆成功'

});

}

**if**(obj==**null**){

$.messager.alert("提示","您的输入有错误!");

}

}

});

//给登陆表单绑定事件

loginInputForm.find('input').on('keyup',**function**(event){

**if**(event.keyCode=='13'){

loginInputForm.submit();

}

});

//注册的dialog

regDialog=$('#regDialog').show().dialog({

left: '100px',

top: '100px',

modal: **true**,

closed: **true**,

title: '注册',

buttons: [{

text: '注册',

handler: **function**(){

regForm.submit();

}

}],

//获取焦点

onOpen: **function**(){

setTimeout(**function**(){

$('#regForm').find('input[name=name]').focus();

});

},

onClose: **function**(){

}

});

//注册的表单

regForm=$('#regForm').form({

url: sy.bp()+ '/regForm',

onSubmit: **function**(){

**if**(!$('#regForm').form('validate')){

**return false**;

};

},

success: **function**(data){

**if**(data!=**null**){

$.messager.show({

title: '标题',

msg: '注册成功'

});

regDialog.dialog('close');

}

}

});

//给注册表单绑定时间

regForm.find('input').on('keyup',**function**(event){

**if**(event.keyCode=='13'){

regForm.submit();

}

});

});

</script>

</head>

<body>

This is my HTML page. <br>

<div id=*"loginAndRegDialog"* title=*"用户登陆"* style="width:*250px*;height:*200px*;">

<form method=*"post"* id=*"loginInputForm"*>

<table>

<tr>

<th align=*"right"*>用户名:</th>

<td><input type=*"text"* name=*"name"* class=*"easyui-validatebox"* required=*"true"*/></td>

</tr>

<tr>

<th align=*"right"*>密码:</th>

<td><input type=*"password"* name=*"password"* class=*"easyui-validatebox"* required=*"true"* /></td>

</tr>

</table>

</form>

</div>

<div id=*"regDialog"* style="width:*250px*;height:*200px*; display: *none*">

<form method=*"post"* id=*"regForm"*>

<table>

<tr>

<th align=*"right"*>用户名:</th>

<td><input type=*"text"* name=*"name"* class=*"easyui-validatebox"* required=*"true"*/></td>

</tr>

<tr>

<th align=*"right"*>密码:</th>

<td><input type=*"password"* name=*"password"* class=*"easyui-validatebox"* required=*"true"* /></td>

</tr>

<tr>

<th align=*"right"*>重复密码:</th>

<td><input type=*"password"* name=*"passwordreg"* class=*"easyui-validatebox"* required=*"required"* validType=*"eqPassword['#regForm input[name=password]']"*/></td>

</tr>

</table>

</form>

</div>

</body>

</html>

## layout的使用

<!DOCTYPE html>

<html>

<head>

<meta http-equiv=*"content-type"* content=*"text/html; charset=UTF-8"*>

<script type=*"text/javascript"*

src=*"../js/jquery-easyui-1.3/jquery-1.7.2.min.js"*></script>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/jquery.easyui.min.js"*></script>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/icon.css"* type=*"text/css"*></link>

<link rel=*"stylesheet"* href=*"../js/jquery-easyui-1.3/themes/default/easyui.css"* type=*"text/css"*></link>

<script type=*"text/javascript"* src=*"../js/jquery-easyui-1.3/locale/easyui-lang-zh\_CN.js"*></script>

<script type=*"text/javascript"* src=*"../js/myjs.js"*></script>

<script type=*"text/javascript"*>

**var** cc;

$(**function**(){

cc=$('#layout').layout();//使用js加载在这里不能写其他的方法,否则无法加载成功

cc.layout('collapse','west');//左边的面板隐藏

});

**function** getCenterPane(){

**var** centerPanel=cc.layout('panel','center');//如何使用panel方法.返回一个中间面板.

console.info(centerPanel);

//如何获取中间面板中title或者其他的属性

console.info(centerPanel.panel('options').title);

**if**(!centerPanel.panel('options').title){

centerPanel.panel({

title: '显示标题'

});

}**else**{

centerPanel.panel({

title: **null**

});

}

}

**function** addPanel(){

$('#layout').layout('add',{

region: 'east',

width: 180,

title: 'east Title',

split: **true**,

tools: [{

iconCls:'icon-add',

handler:**function**(){alert('add')}

},{

iconCls:'icon-remove',

handler:**function**(){alert('remove')}

}]

});

}

</script>

</head>

<body id=*"layout"* >

<div data-options=*"region:'north',title:'aaaaa',split:true"* style="height:*100px*;"></div>

<div data-options=*"region:'south',title:'South Title',split:true"* style="height:*100px*;">

<input type=*"button"* onclick="getCenterPane();" value=*"显示隐藏标题"*/>

<input type=*"button"* onclick="$('#layout').layout('expand','west')" value=*"展开左侧面板"*/>

<input type=*"button"* onclick="addPanel()" value=*"右侧添加面板"*/>

</div>

<div data-options=*"region:'west',title:'West',split:true"* style="width:*100px*;"></div>

<div data-options=*"region:'center'"* style="padding:*5px*;background:*#eee*;" href=*"5.html"*>

</div>

</body>

</html>

# dataGrid数据表格的使用

## 创建一个dataGrid

### html中

<script type=*"text/javascript"*>

**var** datagrid;

$(**function**(){

datagrid=$('#datagrid').datagrid({

url: sy.bp()+ '/datagrid',

title: 'User数据测试', //标题

iconCls: 'icon-save',//图标

pagination: **true**,//是否标记页数页码,在下方会出现页数页码显示

pageSize: 5, //每页显示多少条记录

pageList: [5,10,15,20], //分页列表

fit:**true**, //随着窗体的改变而改变大小

fitColumns: **true**,//是否去掉x轴滚动

nowarp: **false**, //文字是否可以自动换行

border: **false**, //边界是否有像素

idField: 'id', //对每行进行的标识

sortName: 'id', //默认的排列id

sortOrder: 'desc',//默认的排列方式

columns: [[

{title:'编号',field:'id',width:100,sortable:**true**},//sortable表示编号可以点击并切换升序降序

{title:'姓名',field:'username',width:100,sortable:**true**},

{title:'密码',field:'password',width:100}

]]

});

});

</script>

<div class=*"easyui-tabs"* fit=*"true"* border=*"false"*>

<div title=*"用户管理"* border=*"false"*>

<table id=*"datagrid"*></table>

</div>

</div>

### Controller中

@RequestMapping(value="datagrid")

@ResponseBody

**public** Map dataGrid(**int** page,**int** rows,String sort,String order){

List<User> list=accountService.getAllUser(page,rows,sort,order);

Long num=accountService.getUserCount();

Map map=**new** HashMap();

map.put("total", num);

map.put("rows", list);

**return** map;

}

### Service方法

/\*\*

\* 返回待条件的用户列表

\*/

**public** List<User> getAllUser(Integer page,Integer rows,String sort, String order) {

String hql="from User order by "+sort+" "+order ;

List<User> list=dao.findPage(hql, page, rows);

**return** list;

}

/\*\*

\* 查询用户总数

\*/

**public** Long getUserCount() {

String hql="select count(\*) from User";

**return** dao.findCounts(hql);

}

### dao层方法

/\*\*

\* 查询总数

\*/

**public** Long findCounts(String hql,Object...params) {

Query query=sessionFactory.getCurrentSession().createQuery(hql);

setParamter(query, params)

**return** (Long) query.uniqueResult();

}

**public** List findPage(String hql, **int** page, **int** size) {

Query query=sessionFactory.getCurrentSession().createQuery(hql);

**return** query.setFirstResult((page-1)\*size).setMaxResults(size).list();

}

## 使用增删改查操作数据库

### 如何增加工具栏

toolbar : [ {

text : '增加',

iconCls : 'icon-add',

handler : **function**() {

}

}, '-', {

text : '修改',

iconCls : 'icon-edit',

handler : **function**() {

}

}, '-', {

text : '删除',

iconCls : 'icon-remove',

handler : **function**() {

}

} ]

### 如何在一个窗口中再创建多个layout

<div class=*"easyui-tabs"* fit=*"true"* border=*"false"*>

<div title=*"用户管理"* border=*"false"*>

<div class=*"easyui-layout"* fit=*"true"* border=*"false"*>

<div region=*"north"* border=*"false"* title=*"过滤"*

style="height:*100%*;width:*100%*">

</div>

<div border=*"false"* region=*"center"*>

<table id=*"datagrid"*></table>

</div>

</div>

</div>

</div>

### dataGrid创建表单以及如何后台传值

表格创建在添加的layout中即*north*

<table class=*"tableForm"* style="width:*100%*;height:*100%*">

<tr>

<th>用户名</th>

<td><input name=*"name"* style="width:*315px*;" /></td>

</tr>

<tr>

<th>创建时间</th>

<td><input name=*"createdatatimeStart"*

class=*"easyui-datetimebox"* style="width:*155px*;" editable=*"false"* />

至<input name=*"createdatatimeEnd"* class=*"easyui-datetimebox"*

editable=*"false"* style="width:*155px*;" /> <a

href=*"javascript:void(0)"* class=*"easyui-linkbutton"*

onclick="my.serch();">查询</a> <a href=*"javascript:void(0)"*

class=*"easyui-linkbutton"* onclick="cleanSerch();">清空</a></td>

</tr>

</table>

### javascript中写上查询的方法

**var** usertableForm=$('#usertableForm').form();

my.serch=**function**(){

//可以序列化所有的表单对象

datagrid.datagrid('load',sy.serializeObject(usertableForm));

};

**sy.serializeObject为将表单对象中的所有元素转换为对象的方法如下:**

sy.serializeObject=**function**(form){/\*将form表单元素的值序列化成对象\*/

**var** o={};

$.each(form.serializeArray(),**function**(index){

//console.info(form.serializeArray());

**if**(o[**this**['name']]){

o[**this**['name']]=o[**this**['name']] + "," + **this**['value'];

}**else**{

o[**this**['name']]=**this**['value'];

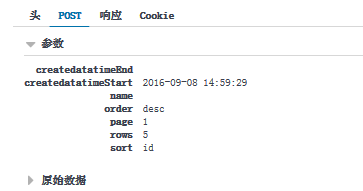
}

});

**return** o;

};

### 通过浏览器验证点击是否传入表单中的元素到后台



### 后台UserDao重写通用查询方法.

**public** **class** UserDaoImpl **extends** BaseDaoImpl **implements** UserDao{

**public** List<User> findPage(String hql, Integer page, Integer rows,

JsonUser userd) {

**final** String finalHql=**this**.setHql(hql,userd);

Query query=**this**.sessionFactory.getCurrentSession().createQuery(finalHql);

**return** query.setFirstResult((page-1)\*rows).setMaxResults(rows).list();

}

**private** String setHql(String hql, JsonUser userd) {

String finalHql=hql;

**if**(StringUtil.*isNotEmpty*(userd.getName())){

finalHql=finalHql+" and username like '%"+userd.getName()+"%'";

}

**if**(StringUtil.*isNotEmpty*(userd.getCreatedatatimeStart())){

finalHql=finalHql+" and createDate > '" + userd.getCreatedatatimeStart()+"'";

}

**if**(userd.getCreatedatatimeEnd()!=**null**&&userd.getCreatedatatimeEnd()!=""){

finalHql=finalHql+" and createDate < '" + userd.getCreatedatatimeEnd()+"'";

}

**if**(StringUtil.*isNotEmpty*(userd.getCreatedatatimeEnd())){

finalHql=finalHql+userd.getOrderby();

}

**return** finalHql;

}

}

### Controller

List<User> list=accountService.getAllUser(page,rows,sort,order,userd);就能实现所有的条件查询

### 实现清空按钮

my.cleanSerch=**function**(){

console.info($('#usertableForm'));

//清空的操作

datagrid.datagrid('load',{});//重新加载组件

//替换表单中inout中所有内容为空

$('#usertableForm input').val('');

}

接下来的easyui请详见easyUI学习文档

# ueditor的配置

## 修改jsp路径下config.json

1 /\* 上传图片配置项 \*/中更改

"imageUrlPrefix": "/pro\_shiro2", /\* 图片访问路径前缀 \*/ ----写项目名称

2 /\* 列出指定目录下的图片 \*/

"imageManagerUrlPrefix": "/pro\_shiro2/", /\* 图片访问路径前缀 \*/

3 ueditor.config.js文件下修改

**var** URL = window.UEDITOR\_HOME\_URL="/pro\_shiro2/ueditor/";

修改url的地址

### 新建页面ueditor.jsp

映入所需要的js和样式

<!-- 默认配置文件 -->

<script type=*"text/javascript"* charset=*"utf-8"* src=*"ueditor/ueditor.config.js"*></script>

<script type=*"text/javascript"* charset=*"utf-8"* src=*"ueditor/ueditor.all.min.js"*> </script>

<script type=*"text/javascript"* charset=*"utf-8"* src=*"ueditor/lang/zh-cn/zh-cn.js"*></script>

<LINK rel=*stylesheet* href=*"ueditor/themes/default/css/ueditor.css"*>

写上测试代码

<form action=*"myueditor"*method=*"post"*>

类别： <input type=*"text"* name=*"category"*/><br/>

标题：<input type=*"text"* name=*"title"*/><br/>

<script id=*"editor"* name=*"content"* type=*"text/plain"* style="width:*1024px*;height:*500px*;">aaaaa</script>

<script type=*"text/javascript"*>

//实例化编辑器

//建议使用工厂方法getEditor创建和引用编辑器实例，如果在某个闭包下引用该

// 编辑器，直接调用UE.getEditor('editor')就能拿到相关的实例

**var** ue = UE.getEditor('editor');

</script>

<input type=*"submit"*value=*"提交"*/>

</form>

### 导入jsp文件下lib包所有

### Controller中填写方法进行测试

# springmvc文件上传的实现

## mvc-context.xml文件中配置

<bean id=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"* >

<property name=*"defaultEncoding"* value=*"gbk"*/> <!-- 默认编码 (ISO-8859-1) -->

<property name=*"maxInMemorySize"* value=*"10240"*/> <!-- 最大内存大小 (10240)-->

<property name=*"uploadTempDir"* value=*"/upload/"*/> <!-- 上传后的目录名 (WebUtils#TEMP\_DIR\_CONTEXT\_ATTRIBUTE) -->

<property name=*"maxUploadSize"* value=*"-1"*/> <!-- 最大文件大小，-1为无限止(-1) -->

</bean>

## 新建测试页面

<body>

<form action=*"uploads"* method=*"post"* enctype=*"multipart/form-data"*>

<input type=*"text"* name=*"name"* /><br>

<input type=*"file"* name=*"file"* />

<input type=*"submit"* />

</form>

</body>

## 新建Controller

@Controller

**public** **class** FileUploadController **implements** ServletContextAware{

**private** ServletContext context;

/\*

@Resource

private AccountService accountService;\*/

**public** **void** setServletContext(ServletContext context) {

**this**.context=context;

}

@RequestMapping(value="uploads",method=RequestMethod.*POST*)

**public** String handleUploadData(HttpServletRequest request,String name,@RequestParam("file") CommonsMultipartFile file){

**if**(!file.isEmpty()){

String path=**this**.context.getRealPath("/tmp/");

System.*out*.println(path);

String fileName=file.getOriginalFilename();

String fileType=fileName.substring(fileName.lastIndexOf("."));

System.*out*.println(fileType);

String saveName=**new** Date().getTime()+fileType;

File file2=**new** File(path,saveName);

**try** {

//将上传的文件写入新建的文件中

file.getFileItem().write(file2);

} **catch** (Exception e) {

e.printStackTrace();

}

//存入数据库的操作

//accountService.add(name, saveName);

System.*out*.println(saveName);

request.setAttribute("imageUrl",saveName);

**return** "redirect:/home";

}**else**

**return** "redirect:/toLogin";

}

}

保存在tmp路径下,所以新建一个tmp文件夹用来存放图片

所有方法均要为post提交

### tomcat下工程路径下temp文件夹下查看是否保存成功

# springmvc拦截器配置

## 新建org.swinglife.interceptor

### 新建自定义拦截器



### 配置文件中配制拦截器

<!-- 配置springmvc拦截器 -->

<mvc:interceptors>

<!-- 配置创建的拦截器,不配置拦截所有的请求 -->

<bean class=*"org.swinglife.interceptor.MyInterceptor"*></bean>

<mvc:interceptor>

<!-- 配置其他拦截器 -->

<mvc:mapping path=*"/login"* />

<bean class=*"org.swinglife.interceptor.MyInterceptor2"*></bean>

</mvc:interceptor>

</mvc:interceptors>