# Shrio的基本介绍：

shiro安全框架是目前为止作为登录注册最常用的框架，因为它十分的强大简单，提供了认证、授权、加密和会话管理等功能 。

**shiro能做什么？**

　　　　　　　认证：验证用户的身份

　　　　　　　授权：对用户执行访问控制：判断用户是否被允许做某事

　　　　　　　会话管理：在任何环境下使用 Session API，即使没有 Web 或EJB 容器。

　　　　　　　加密：以更简洁易用的方式使用加密功能，保护或隐藏数据防止被偷窥

　　　　　　　Realms：聚集一个或多个用户安全数据的数据源

　　　　　　　单点登录（SSO）功能。

　　　　　　　为没有关联到登录的用户启用 "Remember Me“ 服务

**Shiro 的四大核心部分**

　　　　　　Authentication(身份验证)：简称为“登录”，即证明用户是谁。

　　　　　　Authorization(授权)：访问控制的过程，即决定是否有权限去访问受保护的资源。

　　　　　　Session Management(会话管理)：管理用户特定的会话，即使在非 Web 或 EJB 应用程序。

　　　　　　Cryptography(加密)：通过使用加密算法保持数据安全

**shiro的三个核心组件：**

　　　　　　Subject ：正与系统进行交互的人，或某一个第三方服务。所有 Subject 实例都被绑定到（且这是必须的）一个SecurityManager 上。

　　　　　　SecurityManager：Shiro 架构的心脏，用来协调内部各安全组件，管理内部组件实例，并通过它来提供安全管理的各种服务。当 Shiro 与一个 Subject 进行交互时，实质上是幕后的 SecurityManager 处理所有繁重的 Subject 安全操作。

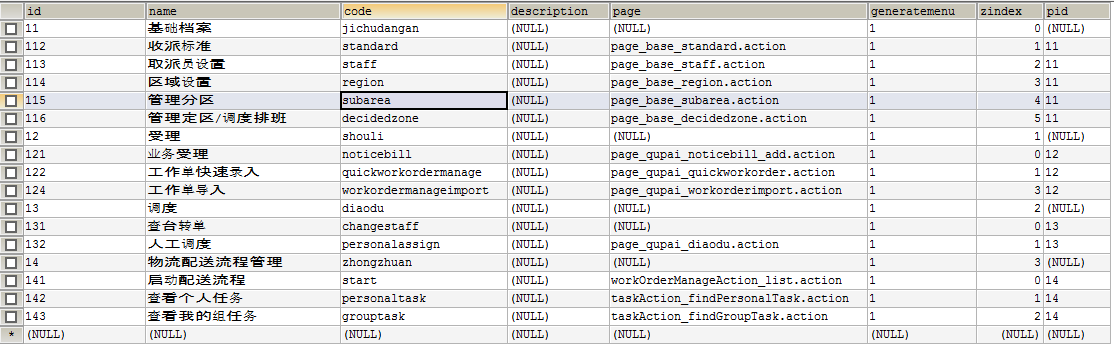
　　　　　　Realms ：本质上是一个特定安全的 DAO。当配置 Shiro 时，必须指定至少一个 Realm 用来进行身份验证和/或授权。Shiro 提供了多种可用的 Realms 来获取安全相关的数据。如关系数据库(JDBC)，INI 及属性文件等。可以定义自己 Realm 实现来代表自定义的数据源。（和数据库打交道，有两个方法一个用于授权一个用于认证）

# 权限管理：

## 设计表

1.设计表：五张表 角色表 用户表 资源表 角色资源表 用户角色表(角色和用户之间是多对多 资源和角色之间也是多对多)

2.初始化资源数据 权限改有的字段（资源 id pid（这个如果是根目录顶级（模块名）pid设置为null 连接也设置为null 其他的都有pid和链接））



## 2.给用户分配权限（往往需要一个管理系统） 当然也可以初始化数据

先创建角色 给角色分配资源 在给用户分配角色

## 3.登录 认证 授权

认证：

在action中：在原来登录的基础上 ：user是否存在不让你去判断了 由shrio去判断 你只需要提供用户名和密码给他就可以了 把带认证的token 加入到subject中

自己建立Realm（和数据库打交道）：从token取出user 根据名字查询数据库是否有该用户 如果有 再由shrio自己确认密码是否正确

授权：

从subject中获取到user对象 先判断用户是否是初始化用户（是查询所有权限） 不是只查询用户改有的权限（注意初始化的用户默认可以有所有权限 表中没有分配权限）

代码如下：

获取对象

/\*\*

\* 用户登录,使用shiro框架提供的方式进行认证操作

\*/

public String login(){

//从Session中获取生成的验证码

String validatecode = (String) ServletActionContext.getRequest().getSession().getAttribute("key");

//校验验证码是否输入正确

if(StringUtils.isNotBlank(checkcode) && checkcode.equals(validatecode)){

//使用shiro框架提供的方式进行认证操作

Subject subject = SecurityUtils.getSubject();//获得当前用户对象,状态为“未认证”

AuthenticationToken token = new UsernamePasswordToken(model.getUsername(),MD5Utils.md5

(model.getPassword()));//创建用户名密码令牌对象

try{

subject.login(token);

}catch(Exception e){

e.printStackTrace();

return LOGIN;

}

User user = (User) subject.getPrincipal();

ServletActionContext.getRequest().getSession().setAttribute("loginUser", user);

return HOME;

}else{

//输入的验证码错误,设置提示信息，跳转到登录页面

this.addActionError("输入的验证码错误！");

return LOGIN;

}

}

public class BOSRealm extends AuthorizingRealm{

@Autowired

private IUserDao userDao;

@Autowired

private IFunctionDao functionDao;

//认证方法

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException

{

System.out.println("自定义的realm中认证方法执行了。。。。");

UsernamePasswordToken passwordToken = (UsernamePasswordToken)token;

//获得页面输入的用户名

String username = passwordToken.getUsername();

//根据用户名查询数据库中的密码

User user = userDao.findUserByUsername(username);

if(user == null){

//页面输入的用户名不存在

return null;

}

//简单认证信息对象

AuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(), this.getName());

//框架负责比对数据库中的密码和页面输入的密码是否一致

return info;

}

//授权方法

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

//获取当前登录用户对象

User user = (User) SecurityUtils.getSubject().getPrincipal();

//User user2 = (User) principals.getPrimaryPrincipal();

// 根据当前登录用户查询数据库，获取实际对应的权限

List<Function> list = null;

if(user.getUsername().equals("admin")){

DetachedCriteria detachedCriteria = DetachedCriteria.forClass(Function.class);

//超级管理员内置用户，查询所有权限数据

list = functionDao.findByCriteria(detachedCriteria);

}else{

list = functionDao.findFunctionListByUserId(user.getId());

}

for (Function function : list) {

info.addStringPermission(function.getCode());

}

return info;

}

}

注意需要导包

配置文件配置：

**在web.Xml中**

<!-- 配置spring框架提供的用于整合shiro框架的过滤器 -->

<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>

**在application中**

<!-- 注册安全管理器对象 -->

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

<property name=*"realm"* ref=*"bosRealm"*/>

<!-- 注入缓存管理器 -->

<property name=*"cacheManager"* ref=*"cacheManager"*/>

</bean>

<!-- 注册realm -->

<bean id=*"bosRealm"* class=*"com.itheima.bos.realm.BOSRealm"*>

</bean>

## 权限控制：

### 使用shiro的方法注解方式权限控制

第一步：在spring配置文件中开启shiro注解支持

<!-- 开启shiro框架注解支持 -->

<bean id=*"defaultAdvisorAutoProxyCreator"*

class=*"org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"*>

<!-- 必须使用cglib方式为Action对象创建代理对象 -->

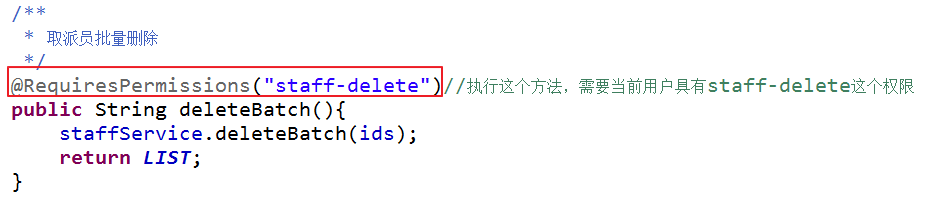
<property name=*"proxyTargetClass"* value=*"true"*/>

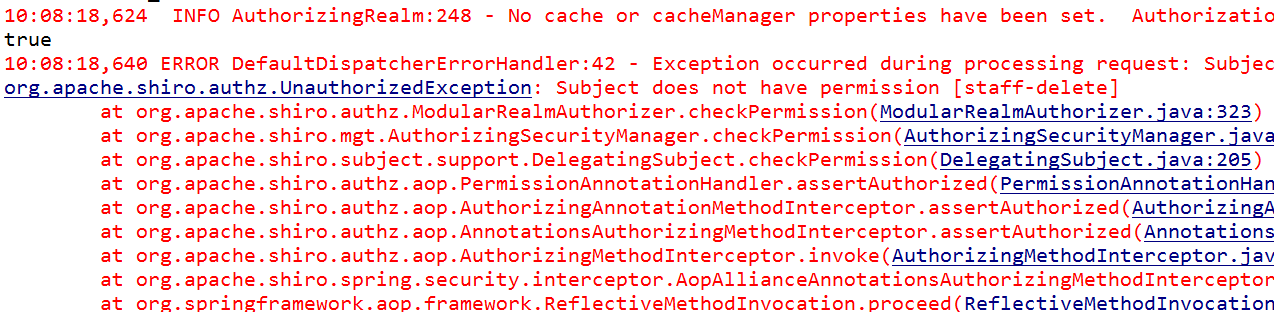
</bean>

<!-- 配置shiro框架提供的切面类，用于创建代理对象 -->

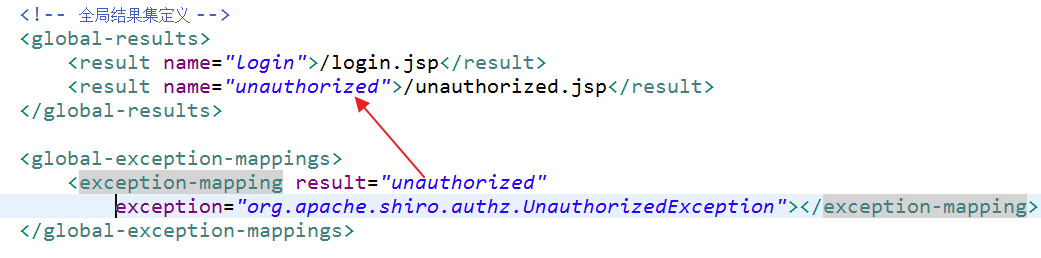
<bean class=*"org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor"*/>

第二步：在Action的方法上使用shiro注解





第三步：在struts.xml中配置全局异常捕获，当shiro框架抛出权限不足异常时，跳转到权限不足提示页面

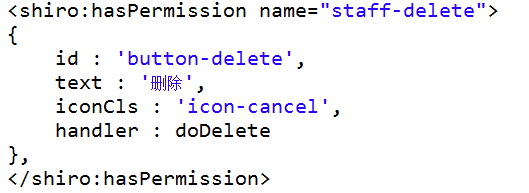


### 使用shiro提供的页面标签方式权限控制

第一步：在jsp页面中引入shiro的标签库

<%@ taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>

第二步：使用shiro的标签控制页面元素展示



### 使用过滤器的方式：（有注解可以不使用这一种）

<!-- 配置shiro框架的过滤器工厂对象 -->

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

<!-- 注入安全管理器对象 -->

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

<!-- 注入相关页面访问URL -->

<property name=*"loginUrl"* value=*"/login.jsp"*/>

<property name=*"successUrl"* value=*"/index.jsp"*/>

<property name=*"unauthorizedUrl"* value=*"/unauthorized.jsp"*/>

<!--注入URL拦截规则 -->

<property name=*"filterChainDefinitions"*>

<value>

/css/\*\* = anon

/js/\*\* = anon

/images/\*\* = anon

/validatecode.jsp\* = anon

/login.jsp = anon

/userAction\_login.action = anon

/page\_base\_staff.action = perms["staff-list"]

/\* = authc

</value>

</property>

</bean>