# 以大连理工的项目为例

# 1.编写实体类(公司) 主要是随机盐

@Entity  
@Table(name = **"rec\_info"**)  
**public class** CompanyInfo {  
 *//编号，主键UUID* @Id  
 @GenericGenerator(name = **"idGenerator"**, strategy = **"uuid"**)  
 @GeneratedValue(generator = **"idGenerator"**)  
 **private** String **id**;  
  
 *//公司名称* @Column(length = 32,nullable = **false**)  
 **private** String **name**;

*//联系邮箱 用于登录*@Column(length = 50,nullable = **false**)  
**private** String **email**;  
  
*//登录密码*@Column(length = 32,nullable = **false**)  
**private** String **pwd**;  
  
*//随机盐*@Column(length = 50,nullable = **false**)  
**private** String **salt**;

等等好多

# 2.编写注册impl方法（获取随机盐，使用它使密码加密）

## 解释：

### 1.利用方法获取随机盐

### 2.得到前台传来的密码

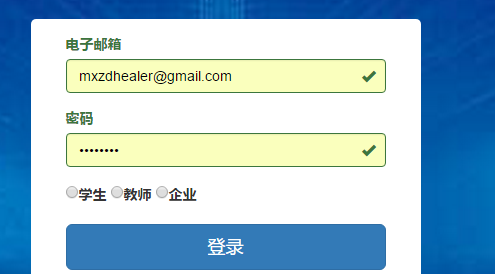
### 3.使用Md5Hash将密码和随机盐进行整合加密

### 4.重新设置密码，进行保存

@Override  
**public void** saveCompany(CompanyInfo companyInfo) {  
  
 *//获得随机盐* SecureRandomNumberGenerator secureRandomNumberGenerator = **new** SecureRandomNumberGenerator();  
 String salt = secureRandomNumberGenerator.nextBytes().toHex();  
 *//对密码加密后,将加密后的密码和盐存入对象* String password = companyInfo.getPwd();  
 password = **new** Md5Hash(password, salt).toString();  
 companyInfo.setPwd(password);  
 companyInfo.setSalt(salt);  
 **companyInfoRepository**.save(companyInfo);  
  
}

# 2.编写登录界面，action可不写其实

<**form action="/login.html" method="post"**>  
   
  
 <**label**>电子邮箱</**label**>  
 <**input type="text" name="username" id="loginName"** />  
  
 <**label** >密码</**label**>  
 <**input type="password" name="password"** />  
  
 <**label**<**input type="radio" name="type" value="STUDENT"**/>学生</**label**>  
 <**label** ><**input type="radio" name="type" value="TEACHER"**/>教师</**label**>  
 <**label** ><**input type="radio" name="type" value="COMPANY"**/>企业</**label**>  
  
 <**button type="submit"** >登录</**button**>



# 2.impl实现查找用户功能，通过邮箱找到用户，但是我们不需要，也没有必要获取它的全部信息，只要具备下面几个登录之后所携带的属性即可 这个将来用到了UserReam中

## 1.impl方法

@Override  
**public** RecLoginDTO findLoginInfo(String email) {  
 **return companyInfoRepository**.findLoginInfo(email);  
}

## 2.数据层查询，并且将结果通过构造器，进行绑定domian

@Query(value = **"select new cn.edu.dlut.career.dto.company.RecLoginDTO (c.id ,c.name,c.email,c.pwd ,c.salt )from CompanyInfo c where c.email=?1"**)  
RecLoginDTO findLoginInfo(String email);

## 3.绑定信息的实体类RecLoginDTO

**public class** RecLoginDTO {  
 **private** String **id**;  
 **private** String **name**;  
 **private** String **email**;  
 **private** String **pwd**;  
 **private** String **salt**;

get set 空构造器

下面这个用在了通过邮箱查询用户中中进行用户身份携带属性的绑定和判断

**public** RecLoginDTO(String id, String name, String email, String pwd, String salt) {  
 **this**.**id** = id;  
 **this**.**name** = name;  
 **this**.**email** = email;  
 **this**.**pwd** = pwd;  
 **this**.**salt** = salt;  
}

# 3.开始编写shiro 的配置文件Ream（用于和登录用户身份的判断），这里使用UserReam继承AuthorizingRealm

## 1.编写用户通过邮箱获取数据库用户的一些类companyInfoService

**public class** UserRealm **extends** AuthorizingRealm {  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(UserRealm.**class**);  
  
 @Autowired  
 **private** CompanyInfoService **companyInfoService**;

## 2.编写用户认证的方法，需要重写doGetAuthenticationInfo方法

### 1. 获取前台传来的authcToken标志，也就是前台的 邮箱，密码，类型，但是这里本身这个变量的属性没有getType方法，所以我们使用了自定义的UserLoginToken一个类用于继承UsernamePasswordToken ，重新添加了type属性

### 2.判断用户名和密码是否为空，然后，抛出一个异常（正常情况下，前台就可以自动判断，这里添加我认为就是防止注入）未完，看3

**protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authcToken)  
 **throws** AuthenticationException {  
 SimpleAuthenticationInfo info = **null**;  
 UserLoginToken token = (UserLoginToken) authcToken;  
 String username = token.getUsername();   
 *//通过type判断登录的类型，决定调用哪一个认证方法* String type = token.getType();  
 **if** (username == **null**) {  
 **throw new** AccountException(**"Null usernames are not allowed by this realm."**);  
 }  
 **if** (token.getPassword().**length** < 1) {  
 **throw new** UnknownAccountException(**"Null password are not allowed by this realm."**);  
 }

### 3.这个项目中，我有三个身份，分别是学生，老师，和企业，这里的type是企业

#### 1.一个简单的用户登录，我们没有必要，也不需要将关于企业的所有信息全部查出来，只要查到登录所需要的常用信息即可。所以之类我们是用了RecLoginDTO ，企业登录常用的携带的信息即可（也就是一个domain，id; name; g email; pwd; salt;） 通过用户名查询获得RecLoginDTO

#### 2.通过得到的RecLoginDTO对象查找到盐，用于和用户登录给的简单密码进行结合Md5Hash，然后重新赋值给toaken，

#### 3.然后真正开始一个用户登录后携带的对象（这里我是自定义）UserPrincipal

**public class** UserPrincipal **implements** Serializable {  
 **private static final long *serialVersionUID*** = 1L;  
 **private** String **id**;  
 **private** String **recName**;  
 **private** String **email**;  
 **private** String **role**;

### ）

#### 4. SimpleAuthenticationInfo 这个会自动帮我们进行用户身份密码进行核实，并且设置一个全局的用户UserPrincipal

*//调用企业认证* **if** (**"COMPANY"**.equals(type)) {  
 **try** {  
RecLoginDTO recLoginDTO = **companyInfoService**.findLoginInfo(username);  
 **if**(recLoginDTO!=**null**){  
 *//对原密码用盐加密处理后认证* String salt = recLoginDTO.getSalt();  
**char**[] pwd = **new** Md5Hash(token.getPassword(), salt).toString().toCharArray();  
 token.setPassword(pwd);  
 info= **new** SimpleAuthenticationInfo(**new** UserPrincipal(recLoginDTO.getId(),recLoginDTO.getName(),recLoginDTO.getEmail(),**"COMPANY"**),  
 recLoginDTO.getPwd(),

getName());

这块对比逻辑是先对比username，但是username肯定是相等的，所以真正对比的是password。从这里传入的password（这里是从数据库获取的）和token（filter中登录时生成的）中的password做对比，如果相同就允许登录，不相同就抛出异常。

***logger***.info(**"用户:"** + username + **"登陆"**);  
 **return** info;  
 } **else**{  
 **throw new** AuthenticationException(**"No this user."**);  
 }  
 } **catch** (Exception e) {  
 **throw new** AuthenticationException(e.getMessage());  
 }  
 }**else if**(**"STUDENT"**.equals(type)){  
 *//* ***TODO: 2017/4/6 学生认证实现* return null**;  
 }**else if**(**"TEACHER"**.equals(type)){  
 *//* ***TODO: 2017/4/6 教师认证实现* return null**;  
 }**else**{  
 **return null**;  
 }  
}

## 3.编写用户授权 重写doGetAuthorizationInfo

### 1.将传来的principals 强制转化为我们自定义的UserPrincipal（其实就是我们上面进行验证登录时候的UserPrincipal）

**protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  
 **if** (principals == **null**) {  
 **throw new** AuthorizationException(**"PrincipalCollection method argument cannot be null."**);  
 }

|  |
| --- |
| Object obj = getAvailablePrincipal(principals); |
|  | UserPrincipal principal = (UserPrincipal) obj; |
|  | SimpleAuthorizationInfo info = new SimpleAuthorizationInfo(); |
|  | if (principal.getRole()!=null) { |
|  | principal.getRole(); |
|  | info.addRole(principal.getRole()); |
|  | } |
|  | // if (userRole.equals("COMPANY")){ |
|  | // |
|  | // }else if(userRole.equals("STUDENT")){ |
|  | // |
|  | // }else if(userRole.equals("TEACHER")){ |
|  | // |
|  | // } |

**return** info;  
}

# 4.开始配置自动启动装置ShiroConfig，进行用户名和密码的拦截，以及安全控制

## 1.配置常用变量和获取UserReam

### 1这里的login在contrller中有两个，一个是，没有登录点击随意路径get方法自动进入的

## 2.另外一个是post方法携带参数进入的， 正儿八经的登录，

@Configuration  
**public class** ShiroConfig {  
 **private static** String *LOGIN\_URL* = **"/login.html"**;  
 **private static** String *SUCCESS\_URL* = **"/index.html"**;  
 **private static** String *UNAUTHORIZED\_URL* = **"/login.html"**;  
 @Bean(name = **"userRealm"**)  
 **public** UserRealm getShiroRealm() {  
 **return new** UserRealm();  
 }

## 2.添加安全管理器，将生命的UserReam加入其中，用于管理，并且加入一些讲来能用到的缓存、session的管理者

@Bean(name = **"securityManager"**)  
**public** DefaultWebSecurityManager defaultWebSecurityManager() {  
 DefaultWebSecurityManager dsm = **new** DefaultWebSecurityManager();  
 *//CAS dsm.setRealm(myRealm());* List<Realm> realms = **new** ArrayList<>();  
 realms.add(getShiroRealm());  
 dsm.setRealms(realms);  
 *//CAS  
 //dsm.setSubjectFactory(defaultWebSubjectFactory());* dsm.setSessionManager(defaultWebSessionManager());  
 dsm.setCacheManager(ehCacheManager());  
 **return** dsm;  
}

## 3.缓存和session（现在还不太明白，还是先写上吧）

*/\*\*  
 \* redis 共享session , 修改session的共享域  
 \** ***@return*** *\*/*@Bean  
**public** DefaultWebSessionManager defaultWebSessionManager() {  
 DefaultWebSessionManager d = **new** DefaultWebSessionManager();  
 *//d.getSessionIdCookie().setDomain(cookie\_domain);* **return** d;  
}

@Bean(name = **"ehCacheManager"**)  
**public** EhCacheManager ehCacheManager() {  
 **return new** EhCacheManager();  
}

## 4.真正的启动，拦截器、配置安全管理器，

/\*\*

     \* ShiroFilterFactoryBean 处理拦截资源文件问题。

     \* 注意：单独一个ShiroFilterFactoryBean配置是或报错的，以为在

     \* 初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager

     \*

        Filter Chain定义说明

       1、一个URL可以配置多个Filter，使用逗号分隔

       2、当设置多个过滤器时，全部验证通过，才视为通过

       3、部分过滤器可指定参数，如perms，roles

     \*

     \*/

@Bean(name = **"shiroFilterFactoryBean"**)  
**public** ShiroFilterFactoryBean shiroFilterFactoryBean() {  
 ShiroFilterFactoryBean sf = **new** ShiroFilterFactoryBean();  
 sf.setSecurityManager(defaultWebSecurityManager());  
 sf.setSuccessUrl(*SUCCESS\_URL*);  
 sf.setLoginUrl(*LOGIN\_URL*);  
 sf.setUnauthorizedUrl(*UNAUTHORIZED\_URL*);  
       //拦截器.  
 Map<String, String> chainMap = **new** LinkedHashMap<>();  
 chainMap.put(**"/login.html"**, **"anon"**);  
 chainMap.put(**"/register/\*\*"**, **"anon"**);  
 chainMap.put(**"/js/\*\*"**, **"anon"**);  
 chainMap.put(**"/css/\*\*"**, **"anon"**);  
 chainMap.put(**"/img/\*\*"**, **"anon"**);  
 chainMap.put(**"/fonts/\*\*"**, **"anon"**);  
 chainMap.put(**"/less/\*\*"**, **"anon"**);  
 chainMap.put(**"/\*\*.ico"**,**"anon"**);  
 chainMap.put(**"/\*\*"**, **"authc"**);  
 sf.setFilterChainDefinitionMap(chainMap);  
 **return** sf;  
}

## 解释：

### 1. 拦截器 chainMap.put("/login.html", "anon");

#### anon:所有url都都可以匿名访问;

#### authc: 需要认证才能进行访问;

#### user:配置记住我或认证通过可以访问；

## 2.url模式使用Ant风格模式

Ant路径通配符支持?、\*、\*\*，注意通配符匹配不包括目录分隔符“/”：

**?：匹配一个字符**，如”/admin?”将匹配/admin1，但不匹配/admin或/admin2；

**\*：匹配零个或多个字符串**，如/admin\*将匹配/admin、/admin123，但不匹配/admin/1；

**\*\*：匹配路径中的零个或多个路径**，如/admin/\*\*将匹配/admin/a或/admin/a/b。

## 3. 控制角色访问的的路径，

### 第一个，只有student 角色的才能访问url为student 下面的东西，

chainMap.put(**"/student/\*\*"**,**"authc,roles[STUDENT]"**);

### 第二个，只有同时满足角色0，和1，才能访问teacher路径下面的 chainMap.put("/teacher/\*\*","authc,roles[0，1]");

如/admins/user/\*\*=roles["admin,guest"]

但是这个设置方法是需要每个参数满足才算通过，相当于hasAllRoles()方法。

也就是我们的角色必须同时拥有admin和guest权限才可以。

1，   
/adminUser/\*\* = roles[admin]   
/adminUser/\*\* = roles[test]   
这种情况只能对角色test有效   
3，   
/adminUser/\*\* = roles[admin],roles[test]   
这种情况需要用户拥有二个角色时才有效，

### 第三个，角色为其中之一就可以访问

#### 1.首先 写一个认证的过滤器

**public class** CustomRolesAuthorizationFilter **extends** AuthorizationFilter {  
  
 Logger **logger** = LoggerFactory.*getLogger*(CustomRolesAuthorizationFilter.**class**);  
 @Override  
 **protected boolean** isAccessAllowed(ServletRequest req, ServletResponse resp, Object mappedValue) **throws** Exception {  
 Subject subject = getSubject(req, resp);  
 String[] rolesArray = (String[]) mappedValue;  
 **logger**.info(rolesArray[0]+rolesArray[1]+**"\*\*\*\*\*\*\* config"**+rolesArray.**length**+**""**+subject.getPrincipal());  
 **if** (rolesArray == **null** || rolesArray.**length** == 0) { *//没有角色限制，有权限访问* **return true**;  
 }  
 UserPrincipal userPrincipal = (UserPrincipal) subject.getPrincipal();  
 **if** (userPrincipal!=**null**) { //判断是不是为空，防止首页刷新失败  
 **for** (**int** i = 0; i < rolesArray.**length**; i++) {  
 **if** (userPrincipal.getRole().equals(rolesArray[i])) { *//若当前用户是rolesArray中的任何一个，则有权限访问* **return true**;  
 }  
 }  
 }  
 **return false**;  
 }

2.在ShiroConfig 中的 **shiroFilterFactoryBean 进行填写相关属性**

javax.servlet.Filter;

Map<String ,Filter> cusFileter = **new** HashMap<>();  
cusFileter.put(**"customrole"**, **new** CustomRolesAuthorizationFilter());  
sf.setFilters(cusFileter);

Map<String, String> chainMap = **new** LinkedHashMap<>();  
chainMap.put(**"/login.html"**, **"anon"**);  
chainMap.put(**"/less/\*\*"**, **"anon"**);  
chainMap.put(**"/\*\*.ico"**,**"anon"**);  
chainMap.put(**"/student/\*\*"**,**"authc,roles[STUDENT]"**);  
chainMap.put(**"/teacher/\*\*"**,**"customrole, customrole [0,1]"**);

chainMap.put(**"/\*\*"**, **"authc"**);  
sf.setFilterChainDefinitionMap(chainMap);

# 5.编写登录controler

## 1.第一个login是get参数类型的，点击任何路径只要没有登录都会进入。

## 2.第二个是写好了参数后进入的

@Controller  
**public class** LoginController {  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(LoginController.**class**);  
  
 @GetMapping(**"/login.html"**)  
 **public** String login() {  
 ***logger***.info(**"Show Login Page"**);  
 Subject currentUser = SecurityUtils.*getSubject*();  
 **if** (currentUser.isAuthenticated()) {  
 **return "redirect:index.html"**;  
 }  
 **return "login"**;  
 }

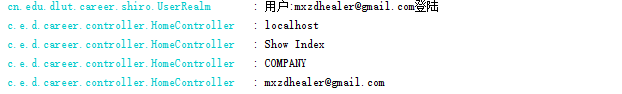
@PostMapping(**"/login.html"**)  
**public** String login(String username, String password,String type, Model model) {  
 Subject currentUser = SecurityUtils.*getSubject*();  
 **if** (!currentUser.isAuthenticated()) {  
 //拓展token，加入登录类型，实现统一登录入口UserLoginToken token = **new** UserLoginToken(username, password,type);  
 **try** {  
 ***logger***.info(**"Begin dto process user login ..."**);  
 currentUser.login(token);  
 **return "redirect:index.html"**;  
 } **catch** (Exception e) {  
 ***logger***.info(e.getMessage());  
 model.addAttribute(**"errorMessage"**, **"您的用户名或密码错误，请重新登录！"**);  
 }  
 }  
 **return "login"**;  
}

# 6.上面的Logincontroler跳转到index页面（HomeController）

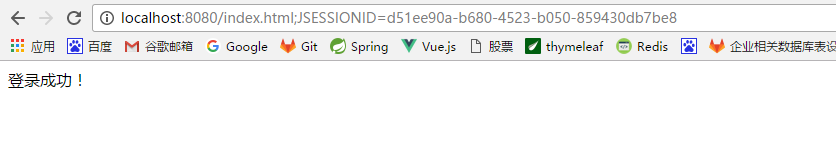
*/\*\*  
 \* 登录后展示的首页  
 \** ***@return*** *\*/* @GetMapping({**"/"**, **"index.html"**})  
 **public** String dashboard() {  
 Subject currentUser = SecurityUtils.*getSubject*();  
 ***logger***.info(**request**.getServerName());  
***logger***.info(**"Show Index"**);  
UserPrincipal userPrincipal = (UserPrincipal) currentUser.getPrincipal();  
 ***logger***.info(userPrincipal.getRole());  
 ***logger***.info(userPrincipal.getEmail());  
 **return "index"**;  
}

# 7.验证成功

## 1.控制台



## 2.浏览器



# 总结：

## 我们会发现login有两个，而且html登录action没有也可以走通，因为它是自动进入shiro的。而且我们会发现如果第二个login里面没有验证登录（就像网上看的学习一样，没有验证登录，而是直接检查异常，也是可以的，但是不严谨，至于为什么，我不知道。反正就是好吧。）下面是异常的类

* DisabledAccountException （禁用的帐号）
* LockedAccountException （锁定的帐号）
* UnknownAccountException（错误的帐号）
* ExcessiveAttemptsException（登录失败次数过多）
* IncorrectCredentialsException （错误的凭证）
* ExpiredCredentialsException （过期的凭证）
* **kaptchaValidateFailed -- > 验证码错误"**);

网上的login

*/ 登录提交地址和applicationontext-shiro.xml配置的loginurl一致。 (配置文件方式的说法)* @RequestMapping(value=**"/login"**,method=RequestMethod.***POST***)  
 **public** String login(HttpServletRequest request, Map<String, Object> map) **throws** Exception {  
 System.***out***.println(**"HomeController.login()"**);  
 *// 登录失败从request中获取shiro处理的异常信息。  
 // shiroLoginFailure:就是shiro异常类的全类名.* String exception = (String) request.getAttribute(**"shiroLoginFailure"**);  
  
 System.***out***.println(**"exception="** + exception);  
 String msg = **""**;  
 **if** (exception != **null**) {  
 **if** (UnknownAccountException.**class**.getName().equals(exception)) {  
 System.***out***.println(**"UnknownAccountException -- > 账号不存在："**);  
 msg = **"UnknownAccountException -- > 账号不存在："**;  
 } **else if** (IncorrectCredentialsException.**class**.getName().equals(exception)) {  
 System.***out***.println(**"IncorrectCredentialsException -- > 密码不正确："**);  
 msg = **"IncorrectCredentialsException -- > 密码不正确："**;  
 } **else if** (**"kaptchaValidateFailed"**.equals(exception)) {  
 System.***out***.println(**"kaptchaValidateFailed -- > 验证码错误"**);  
 msg = **"kaptchaValidateFailed -- > 验证码错误"**;  
 } **else** {  
 msg = **"else >> "**+exception;  
 System.***out***.println(**"else -- >"** + exception);  
 }  
 }  
 map.put(**"msg"**, msg);  
 *// 此方法不处理登录成功,由shiro进行处理.* **return "/login"**;  
 }

# 8.至此登录验证就是成功了，后续就是角色和权限的搭配

|  |
| --- |
| public class UserLoginToken extends UsernamePasswordToken{ |
|  |  |
|  | //身份类型 |
|  | private String type; |
|  |  |
|  | public UserLoginToken(String username, String password, String type) { |
|  | super(username, password); |
|  | this.type = type; |
|  | } |
|  |  |
|  | public String getType() { |
|  | return type; |
|  | } |
|  |  |
|  | public void setType(String type) { |
|  | this.type = type; |
|  | } |
|  | } |