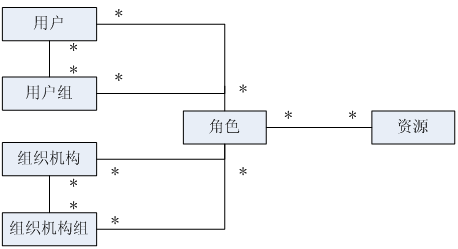
**简单的实体关系图**



**简单数据字典**

用户(sys\_user)

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 描述 |
| id | bigint |  | 编号 主键 |
| username | varchar | 100 | 用户名 |
| password | varchar | 100 | 密码 |
| salt | varchar | 50 | 盐 |
| role\_ids | varchar | 100 | 角色列表 |
| locked | bool |  | 账户是否锁定 |

组织机构(sys\_organization)

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 描述 |
| id | bigint |  | 编号 主键 |
| name | varchar | 100 | 组织机构名 |
| priority | int |  | 显示顺序 |
| parent\_id | bigint |  | 父编号 |
| parent\_ids | varchar | 100 | 父编号列表 |
| available | bool |  | 是否可用 |

资源(sys\_resource)

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 描述 |
| id | bigint |  | 编号 主键 |
| name | varchar | 100 | 资源名称 |
| type | varchar | 50 | 资源类型， |
| priority | int |  | 显示顺序 |
| parent\_id | bigint |  | 父编号 |
| parent\_ids | varchar | 100 | 父编号列表 |
| permission | varchar | 100 | 权限字符串 |
| available | bool |  | 是否可用 |

角色(sys\_role)

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 描述 |
| id | bigint |  | 编号 主键 |
| role | varchar | 100 | 角色名称 |
| description | varchar | 100 | 角色描述 |
| resource\_ids | varchar | 100 | 授权的资源 |
| available | bool |  | 是否可用 |

**资源**：表示菜单元素、页面按钮元素等；菜单元素用来显示界面菜单的，页面按钮是每个页面可进行的操作，如新增、修改、删除按钮；使用type来区分元素类型（如menu表示菜单，button代表按钮），priority是元素的排序，如菜单显示顺序；permission表示权限；如用户菜单使用user:\*；也就是把菜单授权给用户后，用户就拥有了user:\*权限；如用户新增按钮使用user:create，也就是把用户新增按钮授权给用户后，用户就拥有了user:create权限了；available表示资源是否可用，如菜单显示/不显示。

**角色**：role表示角色标识符，如admin，用于后台判断使用；description表示角色描述，如超级管理员，用于前端显示给用户使用；resource\_ids表示该角色拥有的资源列表，即该角色拥有的权限列表（显示角色），即角色是权限字符串集合；available表示角色是否可用。

**组织机构**：name表示组织机构名称，priority是组织机构的排序，即显示顺序；available表示组织机构是否可用。

**用户**：username表示用户名；password表示密码；salt表示加密密码的盐；role\_ids表示用户拥有的角色列表，可以通过角色再获取其权限字符串列表；locked表示用户是否锁定。

此处如资源、组织机构都是树型结构：

|  |  |  |  |
| --- | --- | --- | --- |
| id | name | parent\_id | parent\_ids |
| 1 | 总公司 | 0 | 0/ |
| 2 | 山东分公司 | 1 | 0/1/ |
| 3 | 河北分公司 | 1 | 0/1/ |
| 4 | 济南分公司 | 2 | 0/1/2/ |

parent\_id表示父编号，parent\_ids表示所有祖先编号；如0/1/2/表示其祖先是2、1、0；其中根节点父编号为0。

为了简单性，如用户-角色，角色-资源关系直接在实体（用户表中的role\_ids，角色表中的resource\_ids）里完成的，没有建立多余的关系表，如要查询拥有admin角色的用户时，建议建立关联表，否则就没必要建立了。在存储关系时如role\_ids=1,2,3,；多个之间使用逗号分隔。

用户组、组织机构组本实例没有实现，即可以把一组权限授权给这些组，组中的用户/组织机构就自动拥有这些角色/权限了；另外对于用户组可以实现一个默认用户组，如论坛，不管匿名/登录用户都有查看帖子的权限。

更复杂的权限请参考我的《JavaEE项目开发脚手架》：<http://github.com/zhangkaitao/es>。

**表/数据SQL**

具体请参考

sql/ shiro-schema.sql （表结构）

sql/ shiro-data.sql  （初始数据）

默认用户名/密码是admin/123456。

**实体**

具体请参考com.github.zhangkaitao.shiro.chapter16.entity包下的实体，此处就不列举了。

**DAO**

具体请参考com.github.zhangkaitao.shiro.chapter16.dao包下的DAO接口及实现。

**Service**

具体请参考com.github.zhangkaitao.shiro.chapter16.service包下的Service接口及实现。以下是出了基本CRUD之外的关键接口：

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** ResourceService {
2. Set<String> findPermissions(Set<Long> resourceIds); //得到资源对应的权限字符串
3. List<Resource> findMenus(Set<String> permissions); //根据用户权限得到菜单
4. }

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** RoleService {
2. Set<String> findRoles(Long... roleIds); //根据角色编号得到角色标识符列表
3. Set<String> findPermissions(Long[] roleIds); //根据角色编号得到权限字符串列表
4. }

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** UserService {
2. **public** **void** changePassword(Long userId, String newPassword); //修改密码
3. **public** User findByUsername(String username); //根据用户名查找用户
4. **public** Set<String> findRoles(String username);// 根据用户名查找其角色
5. **public** Set<String> findPermissions(String username);// 根据用户名查找其权限
6. }

Service实现请参考源代码，此处就不列举了。

**UserRealm实现**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** UserRealm **extends** AuthorizingRealm {
2. @Autowired **private** UserService userService;
3. **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
4. String username = (String)principals.getPrimaryPrincipal();
5. SimpleAuthorizationInfo authorizationInfo = **new** SimpleAuthorizationInfo();
6. authorizationInfo.setRoles(userService.findRoles(username));
7. authorizationInfo.setStringPermissions(userService.findPermissions(username));
8. System.out.println(userService.findPermissions(username));
9. **return** authorizationInfo;
10. }
11. **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {
12. String username = (String)token.getPrincipal();
13. User user = userService.findByUsername(username);
14. **if**(user == **null**) {
15. **throw** **new** UnknownAccountException();//没找到帐号
16. }
17. **if**(Boolean.TRUE.equals(user.getLocked())) {
18. **throw** **new** LockedAccountException(); //帐号锁定
19. }
20. **return** **new** SimpleAuthenticationInfo(
21. user.getUsername(), //用户名
22. user.getPassword(), //密码
23. ByteSource.Util.bytes(user.getCredentialsSalt()),//salt=username+salt
24. getName()  //realm name
25. );
26. }
27. }

此处的UserRealm和《第六章Realm及相关对象》中的UserRealm类似，通过UserService获取帐号及角色/权限信息。

**Web层控制器**

**Java代码  [收藏代码](javascript:void())**

1. @Controller
2. **public** **class** IndexController {
3. @Autowired
4. **private** ResourceService resourceService;
5. @Autowired
6. **private** UserService userService;
7. @RequestMapping("/")
8. **public** String index(@CurrentUser User loginUser, Model model) {
9. Set<String> permissions = userService.findPermissions(loginUser.getUsername());
10. List<Resource> menus = resourceService.findMenus(permissions);
11. model.addAttribute("menus", menus);
12. **return** "index";
13. }
14. }

IndexController中查询菜单在前台界面显示，请参考相应的jsp页面；

**Java代码  [收藏代码](javascript:void())**

1. @Controller
2. **public** **class** LoginController {
3. @RequestMapping(value = "/login")
4. **public** String showLoginForm(HttpServletRequest req, Model model) {
5. String exceptionClassName = (String)req.getAttribute("shiroLoginFailure");
6. String error = **null**;
7. **if**(UnknownAccountException.**class**.getName().equals(exceptionClassName)) {
8. error = "用户名/密码错误";
9. } **else** **if**(IncorrectCredentialsException.**class**.getName().equals(exceptionClassName)) {
10. error = "用户名/密码错误";
11. } **else** **if**(exceptionClassName != **null**) {
12. error = "其他错误：" + exceptionClassName;
13. }
14. model.addAttribute("error", error);
15. **return** "login";
16. }
17. }

LoginController用于显示登录表单页面，其中shiro authc拦截器进行登录，登录失败的话会把错误存到shiroLoginFailure属性中，在该控制器中获取后来显示相应的错误信息。

**Java代码  [收藏代码](javascript:void())**

1. @RequiresPermissions("resource:view")
2. @RequestMapping(method = RequestMethod.GET)
3. **public** String list(Model model) {
4. model.addAttribute("resourceList", resourceService.findAll());
5. **return** "resource/list";
6. }

在控制器方法上使用@RequiresPermissions指定需要的权限信息，其他的都是类似的，请参考源码。

**Web层标签库**

com.github.zhangkaitao.shiro.chapter16.web.taglib.Functions提供了函数标签实现，有根据编号显示资源/角色/组织机构名称，其定义放在src/main/webapp/tld/zhang-functions.tld。

**Web层异常处理器**

**Java代码  [收藏代码](javascript:void())**

1. @ControllerAdvice
2. **public** **class** DefaultExceptionHandler {
3. @ExceptionHandler({UnauthorizedException.**class**})
4. @ResponseStatus(HttpStatus.UNAUTHORIZED)
5. **public** ModelAndView processUnauthenticatedException(NativeWebRequest request, UnauthorizedException e) {
6. ModelAndView mv = **new** ModelAndView();
7. mv.addObject("exception", e);
8. mv.setViewName("unauthorized");
9. **return** mv;
10. }
11. }

如果抛出UnauthorizedException，将被该异常处理器截获来显示没有权限信息。

**Spring配置——spring-config.xml**

定义了context:component-scan来扫描除web层的组件、dataSource（数据源）、事务管理器及事务切面等；具体请参考配置源码。

**Spring配置——spring-config-cache.xml**

定义了spring通用cache，使用ehcache实现；具体请参考配置源码。

**Spring配置——spring-config-shiro.xml**

定义了shiro相关组件。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="userRealm" **class**="com.github.zhangkaitao.shiro.chapter16.realm.UserRealm">
2. <property name="credentialsMatcher" ref="credentialsMatcher"/>
3. <property name="cachingEnabled" value="false"/>
4. </bean>

userRealm组件禁用掉了cache，可以参考<https://github.com/zhangkaitao/es/tree/master/web/src/main/java/com/sishuok/es/extra/aop>实现自己的cache切面；否则需要在修改如资源/角色等信息时清理掉缓存。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="sysUserFilter"
2. **class**="com.github.zhangkaitao.shiro.chapter16.web.shiro.filter.SysUserFilter"/>

sysUserFilter用于根据当前登录用户身份获取User信息放入request；然后就可以通过request获取User。

**Java代码  [收藏代码](javascript:void())**

1. <property name="filterChainDefinitions">
2. <value>
3. /login = authc
4. /logout = logout
5. /authenticated = authc
6. /\*\* = user,sysUser
7. </value>
8. </property>

如上是shiroFilter的filterChainDefinitions定义。

**Spring MVC配置——spring-mvc.xml**

定义了spring mvc相关组件。

**Java代码  [收藏代码](javascript:void())**

1. <mvc:annotation-driven>
2. <mvc:argument-resolvers>
3. <bean **class**="com.github.zhangkaitao.shiro.chapter16
4. .web.bind.method.CurrentUserMethodArgumentResolver"/>
5. </mvc:argument-resolvers>
6. </mvc:annotation-driven>

此处注册了一个@CurrentUser参数解析器。如之前的IndexController，从request获取shiro sysUser拦截器放入的当前登录User对象。

**Spring MVC配置——spring-mvc-shiro.xml**

定义了spring mvc相关组件。

**Java代码  [收藏代码](javascript:void())**

1. <aop:config proxy-target-**class**="true"></aop:config>
2. <bean **class**="org.apache.shiro.spring.security
3. .interceptor.AuthorizationAttributeSourceAdvisor">
4. <property name="securityManager" ref="securityManager"/>
5. </bean>

定义aop切面，用于代理如@RequiresPermissions注解的控制器，进行权限控制。

**web.xml配置文件**

定义Spring ROOT上下文加载器、ShiroFilter、及SpringMVC拦截器。具体请参考源码。

**JSP页面**

**Java代码  [收藏代码](javascript:void())**

1. <shiro:hasPermission name="user:create">
2. <a href="${pageContext.request.contextPath}/user/create">用户新增</a><br/>
3. </shiro:hasPermission>

使用shiro标签进行权限控制。具体请参考源码。

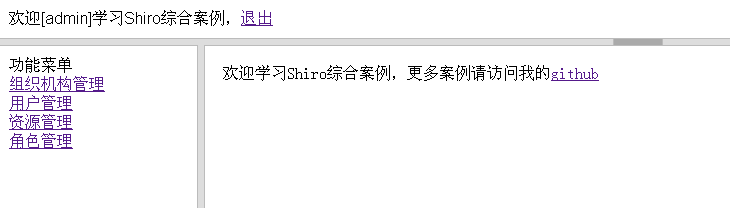
**系统截图**

访问<http://localhost:8080/chapter16/>；

首先进入登录页面，输入用户名/密码（默认admin/123456）登录：



登录成功后到达整个页面主页，并根据当前用户权限显示相应的菜单，此处菜单比较简单，没有树型结构显示



然后就可以进行一些操作，如组织机构维护、用户修改、资源维护、角色授权

