基于 Apache Shiro 的站群角色管理①

刘全飞[1,2], 周相兵[1,3]

[1](阿坝师范高等专科学校, 汶川 623002)
[2](西华大学 数学与计算机学院, 成都 611743)
[3](成都理工大学 地球物理学院, 成都 611743)

摘 要: 在站群角色管理中, 易出现权限管理复杂、角色分配多样, 访问效率低下等问题, 因此, 设计实现一种基于 Apache Shiro 开源框架的 RBAC 管理系统来解决站群角色管理中的不足; 首先全面概述 Shiro 原理, 然后以 Shiro 为基础设计实现站群权限角色管理功能. 最终使用户可以定制不同的角色与权限功能, 且整个过程操作简单便捷.

关键词: 站群系统; 角色管理; Apache Shiro; RBAC

Station Group Role Management Based on Apache Shiro

LIU Quan-Fei[1,2], ZHOU Xiang-Bing[1,3]

[1](Library of Aba Thechers College, Wenchuan 623002, China)
[2](Mathematics and Computer College, Xihua University, Chengdu 611743, China)
[3](Colledge of Geophysical, Chengdu University of Technology, Chengdu 611743, China)

Abstract: In the role of management station group, it is prone to complex privilege management, role assignment problem, diversity, access to low efficiency etc. Therefore, this paper designs and realizes a RBAC management system of Apache Shiro based on open source framework to solve the problem of role management station. First, a comprehensive overview of the principle of Shiro is brought out. Then we design and implement the station group role management function based on Shiro. Finally, users can customize different roles and permissions function, and the whole process is simple and convenient.

Key words: station group system; role management; Apache Shiro; RBAC

目前信息业务系统安全策略的应用主要是以基于角色的访问控制(Role-Based Access Control, RBAC)来实现的[1]; 对于不同的应用系统需求, 需要对 RBAC 进行改进和优化, 以达到更简单、方便的访问控制, 从而减轻软件系统安全策略设计的繁琐; 但传统 RBAC 的实现, 易出现权限管理复杂、角色分配多样, 访问效率低下等问题, 造成整个应用系统安全策略不稳定. 因此, 赵明斌等人[2]提出的云计算访问控制模型能将动态可变机制与主客体安全等级引入访问控制策略中, 既保证云环境下数据的安全性和可靠性, 又具有一定的灵活性. 赵卫东[3], 等人提出的改进基于角色的细粒度访问控制模型, 该模型引入属性约束和用户组, 在分析和设计上从访问控制的粒度出发, 有效减少了角色和权限的授予数量, 降低了权限管理复杂性, 确保了细粒度的访问控制. 从文献[2-3]可以看出改进和优化后 RBAC 的优点: 通过创建角色, 将具有相同角色(功能权限相同, 操作数据对象不同)的用户集合组成组, 然后直接为用户组分配角色, 这就保留了为用户指定一个或多个角色, 而且也使用户通过角色间接地获得操作权限, 减少了授予权限的工作量, 并具备了组织变化给权限-角色带来了更多的灵活性.

然而对站群信息系统来说, 角色管理也是一个复杂的问题, 这是因为站群系统中角色多、管理的功能也多样, 而且不同的功能需求所对应用户也不相同, 不

① 基金项目:四川省应用基础研究项目(2014JY0005);阿坝师范高等专科学校青年基金(ASC10-06);四川省教育厅自然科学项目(15ZB0349)
收稿时间:2014-10-16;收到修改稿时间:2014-11-28

同的用户又所具有的角色也不相同, 使得不同的角色所具有的权限又不一样, 所以直接赋予用户权限复杂度大大增加. 因此, 结合基于 RBAC 模型的优势和站群角色管理的需求, 本文利用 Apache Shiro 对站群系统角色管理进行设计, 以减少站群系统权限赋予的难度, 提高系统的应用效率和能力, 让站群安全控制策略管理变得简单和快捷.

1 Apache Shiro原理

在 Apache Shiro 发布之前, 开发人员通常使用 JAAS 和 Spring Security 进行授权, 但它们的配置相当繁琐. Shiro 的出现, 以其灵活、易用、开放源代码的特点被广泛运用于各类应用系统的安全控制中, 而且 Shiro 有效汇聚了各种改进和优化的 RBAC 结果. 从而使开发人员在使用 Apache Shiro 过程中, 只需要调用 Apache Shiro 提供的 API 就能完成复杂、繁琐的认证和授权过程.

1.1 Apache Shiro 的主要功能

Apache Shiro 是一个开源的 JAVA 安全框架, 其功能强大, 灵活性较好. 它主要提供了访问授权、身份认证、会话管理和数据加密等功能, 能为任何应用系统提供安全认证和身份控制, 很适合在大型应用中对复杂访问控制的操作.

(1)Authenticator(身份认证): 用户身份识别, 常被称为用户“登录”.

(2)Authorizer(访问控制): 对用户进行访问控制, 判断用户对某项操作是否有权限, 判断用户对特定的安全角色是否进行了分配. 同时 shiro 对角色的简单的授权控制, 支持细粒度的签权, 即对某一条数据的访问权进行控制, 也能对单个资源的访问权限进行控制.

(3)Cryptography(数据加密): 通过对数据进行加密算法对敏感数据进行安全保护. Shiro 提供大量易于使用、易于理解的 RSA、Hasher 以及不同编译器实现的加密算法, 开发人员只需要简单的调用这些加密算法, 就可保证数据的安全性.

(4)SessionManager(会话管理): shiro 的特色就是会话管理, 它提供一个强健的会话机制, 能够通过 API 创建和管理所有环境下用户的 Session 周期. Shiro 内置的企业级会话管理功能保证了在任何环境下, Shiro 能够对用户 Session 进行本地管理.

1.2 Apache Shiro 框架结构

Apache Shiro 框架结构如图 1 所示, shiro 由三个核心组件 Subject, SecurityManager 和 Realm 组成[4].

Subject: 指“与当前软件交互的东西”, 可以使人、第三方进程、后台帐户(DaemonAccount)或其他类似事物. 考虑到大多数目的和用途, 我们可以把它认为是 Shiro 的“用户”概念. Subject 代表了当前用户的安全操作, SecurityManager 则管理所有用户的安全操作.

SecurityManager: 它是一个 Façade 接口, 由 Authenticator、 Authorizer 、essionFactory 组成. Shiro 框架的核心是 SecurityManager, 并由它来管理内部组件, 并通过它来提供安全管理的各种服务.

Realm: 它充当 Shiro 与应用安全数据间的“桥梁”. 当用户执行认证和授权验证时, Shiro 会从应用配置的 Realm 中查找用户及权限信息. Realm 封装了数据源的连接细节, 并在需要时将相关数据提供给 Shiro. Shiro 内置了可以连接大量安全数据源的 Realm, 如 LDAP、关系数据库(JDBC)、类似 INI 的文本配置资源以及属性文件等. 如果缺省的 Realm 不能满足需求, 你还可以插入代表自定义数据源的自己的 Realm 实现[5].

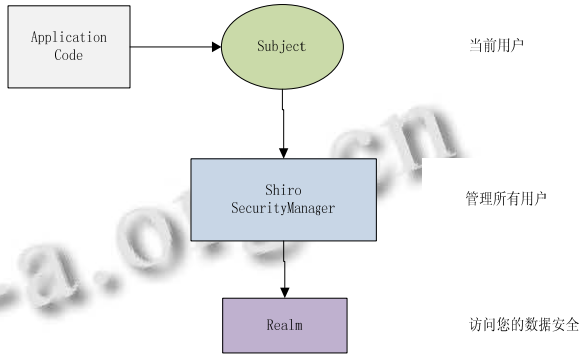

图 1 Shiro 架构图

2 站群角色及访问管理模型

站群访问控制管理通过构建角色, 赋予角色权限; 构建会员组, 赋予会员组权限; 根据用户的功能需求对用户划分角色和会员组, 通过角色和会员组间接赋予用户权限. 而角色管理逻辑设计主要由两方面组成: 站群角色管理体系结构和角色管理模型的实现, 其中角色管理体系结构包括对用户管理, 角色管理, 会员组管理, 工作流组管理的定义和各自组成; 角色管理模型的实现是对角色管理系统进行创建, 编程的过程.

而站群管理角色是将站群中的每个站点的用户角色做为每个站点的角色, 这样使不同站点用户可以管理所属的站点, 也可以管理其它授权的功能, 如图 2 所示.

图 2 站群角色

2.1 构建站群角色管理模型体系结构

站群角色管理模型结构先要从站群角色管理的功能出发, 提取管理的流程, 分解其步骤, 最后确定角色管理的组成. 站群角色管理由多个模块构成. 如图 3 定义了 4 种角色管理模型: 用户管理, 角色管理, 会员组管理, 工作流管理.

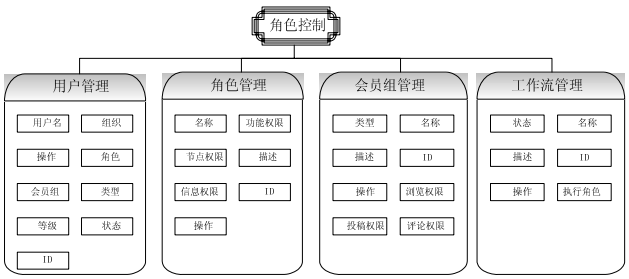

图 3 角色管理模型体系结构

用户管理模型是在角色管理模型和会员组管理模型的基础上建立的, 它直接面对的是站群系统的用户, 属于最底层的模型. 用户管理可以进行除了常规的增加、删除、修改、查询外还可以进行审核、锁定、解锁、删除密码等操作. 用户的权限通过角色和会员组的权限来控制. 用户管理还包括用户的一些基本信息和登录信息. 实现结果如图 4 所示.

图 4 用户管理模型实现结果

角色管理主要完成构建角色名称、完成角色的描述、自定义角色的权限(节点权限、功能权限和信息权限). 所有的权限定义都通过下拉菜单, 选择的模式进行. 角色权限的多少, 根据角色具有的事务特征决定. 角色管理修改功能实现结果如图 5 所示.

图 5 角色管理修改功能实现结果

会员组管理就是把站群权限分为大的几个方面, 让具有相同功能但角色又不同的用户划分在一起. 会员组的权限是针对功能特征而设定, 角色管理权限针对具体事务而设定, 但它们都是为用户管理服务. 实现结果如图 6 所示.

图 6 会员组管理实现结果

工作流管理模型主要是为了站群信息发布而设计的, 在站群信息发布过程中有“审核中”和“待审核”步骤. 在审核的过程中也需要角色来完成, 工作流管理就是对工作流程分配角色, 这里的角色同样在角色管理中已经定义过.

2.2 角色管理实现

在模型的体系结构设计的基础上, 需要对角色管理的设计进行实现. 即使用 Apache Shiro 框架对管理模型进行实现要经过设计数据模型、创建工程, 新建实体、配置 web.xml, 添加过滤器、配置 INI 文件、配置登录页面等过程. 但通常在对 Shiro 框架进行配置时, 通常集成到的 web application 中. 设计数据模型时, 一个账号可以拥有多个角色, 一个角色包含多个权限. 创建实体主要是添加 Shiro 相关的 Jar 包. 把所有过程做完就可以实现过程.

2.2.1 面向 Shiro 用户认证定义

主要完成自定义 Authentication 对象, 使得 Subject 除了携带用户的登录名外还可以携带其它相关信息.

```
public class ShiroUser implements Serializable {
private static final long serialVersionUID = 1L;
public Integer id;
public String username;
public ShiroUser(Integer id, String username) {
    this.id = id;
    this.username = username;
}
/* 本函数输出将作为默认的<shiro:principal/>输出.*/
```

2.2.2 面向 Shiro 用户访问控制

主要实现角色登陆及权限控制.

```
public class ShiroDbRealm extends
AuthorizingRealm implements InitializingBean {
protected UserShiroService userShiroService;
private CredentialsDigest credentialsDigest;
 /* 认证回调函数,登录时调用.*/
@Override protected AuthenticationInfo
doGetAuthenticationInfo(AuthenticationToken
authcToken) throws AuthenticationException {
UsernamePasswordToken token =
(UsernamePasswordToken) authcToken;
        User user =
 userShiroService.findByUsername(token.getUsername())
;
        /*前后台登录共用, 非管理员也可登录.*/
if (user != null && user.isNormal())
{byte[] salt = user.getSaltBytes();
return new SimpleAuthenticationInfo(new
ShiroUser(user.getId(),
        user.getUsername()), user.getPassword(),
            ByteSource.Util.bytes(salt), getName());
        }
    else {
            return null;
        }
    }
    @Override
Protected AuthorizationInfo doGetAuthorizationInfo(
            PrincipalCollection principals) {
    ShiroUser shiroUser = (ShiroUser)
    principals.getPrimaryPrincipal();
    User user = userShiroService.get(shiroUser.id);
SimpleAuthorizationInfo auth
= new SimpleAuthorizationInfo();
        Site site = Context.getCurrentSite();
        if (user != null && site != null) {
    Set<String> perms = user.getPerms(site.getId());
            if (!CollectionUtils.isEmpty(perms)) {
                auth.setStringPermissions(perms);
            }
            if (user.isSuper()) {
                auth.addRole("super");
            }
        }
        return auth;
    }
    /*设定 Password 校验的 Hash 算法与迭代次数.*/
```

2.2.3 面向 Shiro 的权限控制

主要完成整个站群内容的权限控制操作.

```
 public class CmsAuthenticationFilter extends
FormAuthenticationFilter {
    private Logger logger = LoggerFactory
.getLogger(CmsAuthenticationFilter.class);
 public static final String
DEFAULT_BACK_SUCCESS_URL =
"/cmscp/index.do";
    /*是否需要验证码*/
    public static final String
CAPTCHA_REQUIRED_KEY="shiroCaptchaRequired";
    /*验证码错误次数*/
    public static final String
    CAPTCHA_ERROR_COUNT_KEY =
    "shiroCaptchaErrorCount";
    /*验证码名称*/
    public static final String CAPTCHA_PARAM =
"captcha";
    /*返回 URL*/
    public static final String
```

```
FALLBACK_URL_PARAM = "fallbackUrl";
/*后台路径*/
```

2.2.4 面向 Shiro 应用配置

```
<bean
id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
<property
name="securityManager" ref="securityManager" />
    <property
name="loginUrl" value="/login.jspx" />
    <property name="filters">
        <util:map>
<entry key="backSite" value-ref="backSiteFilter" />
<entry key="authc" value-ref="authcFilter" />
    <entry key="user" value-ref="userFilter" />
<entry key="logout" value-ref="logoutFilter" />
        </util:map>
    </property>
    <property name="filterChainDefinitions">
        <value>
            / = anon
            *.jspx = anon
            *.jsp = anon
            /login.jspx = authc
            /logout.jspx = logout
            /my.jspx = user
            /my/** = user
            /cmscp/ = backSite,anon
            /cmscp/index.do = backSite,anon
            /cmscp/login.do = backSite,authc
            /cmscp/logout.do = backSite,logout
            /cmscp/** = backSite,user
        </value>
    </property>
</bean>
<!-- Shiro Filter -->
<bean
 id="backSiteFilter"
class="com.abtc.core.support.BackSiteFilter"
depends-on="siteDao">
<property
                    name="siteShiroService"
ref="siteShiroServiceImpl"/>
<property

name="cacheManager"ref="shiroEhcacheManager"/>
</bean>
<bean
id="authcFilter"
class="com.abtc.core.security.CmsAuthenticationFilter"/
>
 <bean
 id="userFilter"
class="com.abtc.core.security.CmsUserFilter"/>
<bean
 id="logoutFilter"
class="com.abtc.core.security.CmsLogoutFilter"/>
<!-- Shiro's main business-tier object for
web-enabled applications -->
<bean
 id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityMa
nager">
        <property name="realm" ref="shiroDbRealm"
/>
        <property
name="cacheManager" ref="shiroEhcacheManager"
/>
</bean>
<!-- 保证实现了 Shiro 内部 lifecycle 函数的 bean
执行 -->
<bean
 id="lifecycleBeanPostProcessor"
class="org.apache.shiro.spring.LifecycleBeanPostProcess
or" />
<!-- 项目自定义的 Realm, 所有 userRealmService
依赖的 dao 都需要用 depends-on 声明 -->
<bean
 id="shiroDbRealm"
class="com.abtc.core.security.ShiroDbRealm"
```

```
depends-on="userDao">
            <property
        name="userShiroService"
ref="userShiroServiceImpl" />
    </bean>
    <!-- 用户授权信息 Cache, 采用 EhCache -->
 <bean
id="shiroEhcacheManager"class="org.apache.shiro.cache
.ehcache.EhCacheManager">
    <property
     name="cacheManagerConfigFile"
value="classpath:ehcache-shiro.xml"/>
    </bean>
```

3 结语

本文设计实现了基于 Apache Shiro 开源框架的站群访问控制策略. Shiro 是通过 RBAC 的思想所具有强有力角色管理的优势开发的, 而且能与其他诸如 SpringMVC 等框架集成, 也可以基于 Shiro 进行独立开发, 从而就有效避免了在信息系统研发过程角色管理的繁琐和复杂. 而在具体应用 Shiro 时, 直接将用户、角色、会员组和权限通过 Shiro 所提供的 API 调用相应功能即可, 这就大大减轻了程序开发工作的复杂度, 操作更加方便.

参考文献

1 Bertino E, Bonatti P, Ferrar E. TRBAC: A temporal rolebased access control model. ACM Trans. on Information and Systems Security, 2001, 4(3): 191–233.

2 赵明斌,姚志强.基于 RBAC 的云计算访问控制模型.计算机应用,2012,32(S2):267–270.

3 赵卫东,毕晓清,卢新明.基于角色的细粒度访问控制模型的设计与实现.计算机工程与设计,2013,34(2):474–479.

4 刘铁行,及俊川,任玉平.基于 SSH2 与 Apache Shiro 整合的代码生成器的研究.科研信息化技术与应用,2013,4(4):82–88.

5 李双,袁丁.基于角色的多级安全政策.计算机工程与设计,2012,33(6):2166–2171.