

权限管理框架

**讲师：传智.燕青**

# 课程目标

通过学习本课程掌握权限管理的设计思想及方法，使用Shiro框架完成权限管理功能开发。

1. 理解基于资源的权限管理方法。
2. 掌握权限管理的数据模型。
3. 掌握不使用shiro开发基于url的权限管理方法。
4. 掌握Shiro进行用户认证的常用方法。
5. 掌握Shiro进行授权的常用方法。
6. 掌握Shiro整合企业应用开发的方法。

# 权限管理

## 什么是权限管理

基本上涉及到用户参与的系统都要进行权限管理，权限管理属于系统安全的范畴，权限管理实现对用户访问系统的控制，按照安全规则或者[安全策略](http://baike.baidu.com/view/160028.htm)控制用户可以访问而且只能访问自己被授权的资源。

权限管理包括用户身份认证和授权两部分，简称认证授权。对于需要访问控制的资源用户首先经过身份认证，认证通过后用户具有该资源的访问权限方可访问。

## 用户身份认证

### 概念

身份认证，就是判断一个用户是否为合法用户的处理过程。最常用的简单身份认证方式是系统通过核对用户输入的用户名和口令，看其是否与系统中存储的该用户的用户名和口令一致，来判断用户身份是否正确。对于采用[指纹](http://baike.baidu.com/view/5628.htm)等系统，则出示指纹；对于硬件Key等刷卡系统，则需要刷卡。

### 用户名密码身份认证流程



### 关键对象

上边的流程图中需要理解以下关键对象：

* **Subject：主体**

访问系统的用户，主体可以是用户、程序等，进行认证的都称为主体；

* **Principal：身份信息**

是主体（subject）进行身份认证的标识，标识必须具有唯一性，如用户名、手机号、邮箱地址等，一个主体可以有多个身份，但是必须有一个主身份（Primary Principal）。

* **credential：凭证信息**

是只有主体自己知道的安全信息，如密码、证书等。

## 授权

### 概念

授权，即访问控制，控制谁能访问哪些资源。主体进行身份认证后需要分配权限方可访问系统的资源，对于某些资源没有权限是无法访问的。

### 授权流程

下图中橙色为授权流程。



### 关键对象

授权可简单理解为who对what(which)进行How操作：

* Who，即主体（Subject），主体需要访问系统中的资源。
* What，即资源（Resource），如系统菜单、页面、按钮、类方法、系统商品信息等。资源包括资源类型和资源实例，比如商品信息为资源类型，类型为t01的商品为资源实例，编号为001的商品信息也属于资源实例。
* How，权限/许可（Permission），规定了主体对资源的操作许可，权限离开资源没有意义，如用户查询权限、用户添加权限、某个类方法的调用权限、编号为001用户的修改权限等，通过权限可知主体对哪些资源都有哪些操作许可。

权限分为粗颗粒和细颗粒，粗颗粒权限是指对资源类型的权限，细颗粒权限是对资源实例的权限。

主体、资源、权限关系如下图：



### 权限模型

对上节中的主体、资源、权限通过数据模型表示。

主体（账号、密码）

资源（资源名称、访问地址）

权限（权限名称、资源id）

角色（角色名称）

角色和权限关系（角色id、权限id）

主体和角色关系（主体id、角色id）

如下图：



通常企业开发中将资源和权限表合并为一张权限表，如下：

资源（资源名称、访问地址）

权限（权限名称、资源id）

合并为：

权限（权限名称、资源名称、资源访问地址）



上图常被称为权限管理的通用模型，不过企业在开发中根据系统自身的特点还会对上图进行修改，但是用户、角色、权限、用户角色关系、角色权限关系是需要去理解的。

### 权限分配

对主体分配权限，主体只允许在权限范围内对资源进行操作，比如：对u01用户分配商品修改权限，u01用户只能对商品进行修改。

权限分配的数据通常需要持久化，根据上边的数据模型创建表并将用户的权限信息存储在数据库中。

### 权限控制

用户拥有了权限即可操作权限范围内的资源，系统不知道主体是否具有访问权限需要对用户的访问进行控制。

#### 基于角色的访问控制

RBAC基于角色的访问控制（Role-Based Access Control）是以角色为中心进行访问控制，比如：主体的角色为总经理可以查询企业运营报表，查询员工工资信息等，访问控制流程如下：



上图中的判断逻辑代码可以理解为：

if(主体.hasRole("总经理角色id")){

查询工资

}

缺点：以角色进行访问控制粒度较粗，如果上图中查询工资所需要的角色变化为总经理和部门经理，此时就需要修改判断逻辑为“判断主体的角色是否是总经理或部门经理”，系统可扩展性差。

修改代码如下：

if(主体.hasRole("总经理角色id") || 主体.hasRole("部门经理角色id")){

查询工资

}

#### 基于资源的访问控制

RBAC基于资源的访问控制（Resource-Based Access Control）是以资源为中心进行访问控制，比如：主体必须具有查询工资权限才可以查询员工工资信息等，访问控制流程如下：

上图中的判断逻辑代码可以理解为：

if(主体.hasPermission("查询工资权限标识")){

查询工资

}

优点：系统设计时定义好查询工资的权限标识，即使查询工资所需要的角色变化为总经理和部门经理也只需要将“查询工资信息权限”添加到“部门经理角色”的权限列表中，判断逻辑不用修改，系统可扩展性强。

# 权限管理解决方案

## 粗颗粒度和细颗粒度

### 什么是粗颗粒度和细颗粒度

对资源类型的管理称为粗颗粒度权限管理，即只控制到菜单、按钮、方法，粗粒度的例子比如：用户具有用户管理的权限，具有导出订单明细的权限。对资源实例的控制称为细颗粒度权限管理，即控制到数据级别的权限，比如：用户只允许修改本部门的员工信息，用户只允许导出自己创建的订单明细。

### 如何实现粗颗粒度和细颗粒度

对于粗颗粒度的权限管理可以很容易做系统架构级别的功能，即系统功能操作使用统一的粗颗粒度的权限管理。

对于细颗粒度的权限管理不建议做成系统架构级别的功能，因为对数据级别的控制是系统的业务需求，随着业务需求的变更业务功能变化的可能性很大，建议对数据级别的权限控制在业务层个性化开发，比如：用户只允许修改自己创建的商品信息可以在service接口添加校验实现，service接口需要传入当前操作人的标识，与商品信息创建人标识对比，不一致则不允许修改商品信息。

## 基于url拦截

基于url拦截是企业中常用的权限管理方法，实现思路是：将系统操作的每个url配置在权限表中，将权限对应到角色，将角色分配给用户，用户访问系统功能通过Filter进行过虑，过虑器获取到用户访问的url，只要访问的url是用户分配角色中的url则放行继续访问。

如下图：

 使用权限管理框架

对于权限管理基本上每个系统都有，使用权限管理框架完成权限管理功能的开发可以节省系统开发时间，并且权限管理框架提供了完善的认证和授权功能有利于系统扩展维护，但是学习权限管理框架是需要成本的，所以选择一款简单高效的权限管理框架显得非常重要。

# 基于url拦截实现

## 环境准备

jdk：1.7.0\_72

web容器：tomcat7

系统框架：springmvc3.2.0+mybatis3.2.7（详细参考springmvc教案）

前台UI：jquery easyUI1.2.2

## 数据库

创建mysql5.1数据库

创建用户表、角色表、权限表、角色权限关系表、用户角色关系表。

导入脚本，先导入shiro\_sql\_talbe.sql再导入shiro-sql\_table\_data.sql



## activeUser用户身份类

用户登陆成功记录activeUser信息并将activeUser存入session。

**public** **class** ActiveUser **implements** java.io.Serializable {

**private** String userid;//用户id

**private** String usercode;// 用户账号

**private** String username;// 用户名称

**private** List<SysPermission> menus;// 菜单

**private** List<SysPermission> permissions;// 权限

## anonymousURL.properties

anonymousURL.properties公开访问地址，无需身份认证即可访问。

## commonURL.properties

commonURL.properties公共访问地址，身份认证通过无需分配权限即可访问。

## 用户身份认证拦截器

使用springmvc拦截器对用户身份认证进行拦截，如果用户没有登陆则跳转到登陆页面，本功能也可以使用filter实现 。

**public** **class** LoginInterceptor **implements** HandlerInterceptor {

// 在进入controller方法之前执行

// 使用场景：比如身份认证校验拦截，用户权限拦截，如果拦截不放行，controller方法不再执行

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) **throws** Exception {

// 校验用户访问是否是公开资源地址(无需认证即可访问)

List<String> open\_urls = ResourcesUtil.*gekeyList*("anonymousURL");

// 用户访问的url

String url = request.getRequestURI();

**for** (String open\_url : open\_urls) {

**if** (url.indexOf(open\_url) >= 0) {

// 如果访问的是公开 地址则放行

**return** **true**;

}

}

// 校验用户身份是否认证通过

HttpSession session = request.getSession();

ActiveUser activeUser = (ActiveUser) session.getAttribute("activeUser");

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

// 用户已经登陆认证，放行

**return** **true**;

}

// 跳转到登陆页面

request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request,

response);

**return** **false**;

}

## 用户授权拦截器

使用springmvc拦截器对用户访问url进行拦截，如果用户访问的url没有分配权限则跳转到无权操作提示页面（refuse.jsp），本功能也可以使用filter实现。

**public** **class** PermissionInterceptor **implements** HandlerInterceptor {

// 在进入controller方法之前执行

// 使用场景：比如身份认证校验拦截，用户权限拦截，如果拦截不放行，controller方法不再执行

// 进入action方法前要执行

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) **throws** Exception {

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

// 用户访问地址：

String url = request.getRequestURI();

// 校验用户访问是否是公开资源地址(无需认证即可访问)

List<String> open\_urls = ResourcesUtil.*gekeyList*("anonymousURL");

// 用户访问的url

**for** (String open\_url : open\_urls) {

**if** (url.indexOf(open\_url) >= 0) {

// 如果访问的是公开 地址则放行

**return** **true**;

}

}

//从 session获取用户公共访问地址（认证通过无需分配权限即可访问）

List<String> common\_urls = ResourcesUtil.*gekeyList*("commonURL");

// 用户访问的url

**for** (String common\_url : common\_urls) {

**if** (url.indexOf(common\_url) >= 0) {

// 如果访问的是公共地址则放行

**return** **true**;

}

}

// 从session获取用户权限信息

HttpSession session = request.getSession();

ActiveUser activeUser = (ActiveUser) session.getAttribute("activeUser");

// 取出session中权限url

// 获取用户操作权限

List<SysPermission> permission\_list = activeUser.getPermissions();

// 校验用户访问地址是否在用户权限范围内

**for** (SysPermission sysPermission : permission\_list) {

String permission\_url = sysPermission.getUrl();

**if** (url.contains(permission\_url)) {

**return** **true**;

}

}

// 跳转到页面

request.getRequestDispatcher("/WEB-INF/jsp/refuse.jsp").forward(

request, response);

**return** **false**;

}

## 用户登陆

用户输入用户账号和密码登陆，登陆成功将用户的身份信息（用户账号、密码、权限菜单、权限url等）记入activeUser类，并写入session。

### controller

//用户登陆提交

@RequestMapping("/loginsubmit")

**public** String loginsubmit(HttpSession session,String usercode,String password,String randomcode) **throws** Exception{

//校验验证码

//从session获取正确的验证码

String validateCode = (String)session.getAttribute("validateCode");

**if**(!randomcode.equals(validateCode)){

//抛出异常：验证码错误

**throw** **new** CustomException("验证码 错误 ！");

}

//用户身份认证

ActiveUser activeUser = sysService.authenticat(usercode, password);

//记录session

session.setAttribute("activeUser", activeUser);

**return** "redirect:first.action";

}

### service接口

/\*\*

\*

\* <p>

\* Title: authenticat

\* </p>

\* <p>

\* Description:用户认证

\* </p>

\*

\* **@param** usercode

\* 用户账号

\* **@param** password

\* 用户密码

\* **@return** ActiveUser 用户身份信息

\* **@throws** Exception

\*/

**public** ActiveUser authenticat(String usercode, String password)

**throws** Exception;

// 根据账号查询用户

**public** SysUser findSysuserByUsercode(String usercode) **throws** Exception;

// 根据用户id获取权限

**public** List<SysPermission> findSysPermissionList(String userid)

**throws** Exception;

// 根据用户id获取菜单

**public** List<SysPermission> findMenuList(String userid) **throws** Exception;

# shiro介绍

## 什么是shiro

Shiro是apache旗下一个开源框架，它将软件系统的安全认证相关的功能抽取出来，实现用户身份认证，权限授权、加密、会话管理等功能，组成了一个通用的安全认证框架。

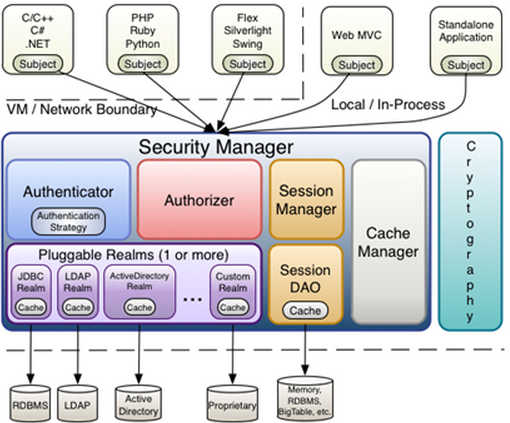
## 为什么要学shiro

既然shiro将安全认证相关的功能抽取出来组成一个框架，使用shiro就可以非常快速的完成认证、授权等功能的开发，降低系统成本。

shiro使用广泛，shiro可以运行在web应用，非web应用，集群分布式应用中越来越多的用户开始使用shiro。

java领域中spring security(原名Acegi)也是一个开源的权限管理框架，但是spring security依赖spring运行，而shiro就相对独立，最主要是因为shiro使用简单、灵活，所以现在越来越多的用户选择shiro。

## Shiro架构



### Subject

Subject即主体，外部应用与subject进行交互，subject记录了当前操作用户，将用户的概念理解为当前操作的主体，可能是一个通过浏览器请求的用户，也可能是一个运行的程序。 Subject在shiro中是一个接口，接口中定义了很多认证授相关的方法，外部程序通过subject进行认证授，而subject是通过SecurityManager安全管理器进行认证授权

### SecurityManager

SecurityManager即安全管理器，对全部的subject进行安全管理，它是shiro的核心，负责对所有的subject进行安全管理。通过SecurityManager可以完成subject的认证、授权等，实质上SecurityManager是通过Authenticator进行认证，通过Authorizer进行授权，通过SessionManager进行会话管理等。

SecurityManager是一个接口，继承了Authenticator, Authorizer, SessionManager这三个接口。

### Authenticator

Authenticator即认证器，对用户身份进行认证，Authenticator是一个接口，shiro提供ModularRealmAuthenticator实现类，通过ModularRealmAuthenticator基本上可以满足大多数需求，也可以自定义认证器。

### Authorizer

Authorizer即授权器，用户通过认证器认证通过，在访问功能时需要通过授权器判断用户是否有此功能的操作权限。

### realm

Realm即领域，相当于datasource数据源，securityManager进行安全认证需要通过Realm获取用户权限数据，比如：如果用户身份数据在数据库那么realm就需要从数据库获取用户身份信息。

注意：不要把realm理解成只是从数据源取数据，在realm中还有认证授权校验的相关的代码。

### sessionManager

sessionManager即会话管理，shiro框架定义了一套会话管理，它不依赖web容器的session，所以shiro可以使用在非web应用上，也可以将分布式应用的会话集中在一点管理，此特性可使它实现单点登录。

### SessionDAO

SessionDAO即会话dao，是对session会话操作的一套接口，比如要将session存储到数据库，可以通过jdbc将会话存储到数据库。

### CacheManager

CacheManager即缓存管理，将用户权限数据存储在缓存，这样可以提高性能。

### Cryptography

Cryptography即密码管理，shiro提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能。

## shiro的jar包

与其它java开源框架类似，将shiro的jar包加入项目就可以使用shiro提供的功能了。shiro-core是核心包必须选用，还提供了与web整合的shiro-web、与spring整合的shiro-spring、与任务调度quartz整合的shiro-quartz等，下边是shiro各jar包的maven坐标。

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-ehcache</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-quartz</artifactId>

<version>1.2.3</version>

</dependency>

也可以通过引入shiro-all包括shiro所有的包：

<dependency>

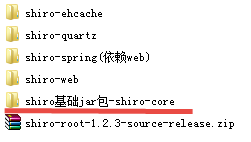
<groupId>org.apache.shiro</groupId>

<artifactId>shiro-all</artifactId>

<version>1.2.3</version>

</dependency>

参考lib目录 ：



# shiro认证

## 认证流程



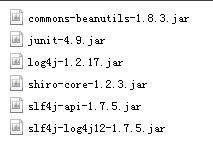
## 入门程序（用户登陆和退出）

### 创建java工程

jdk版本：1.7.0\_72

eclipse：elipse-indigo

### 加入shiro-core的Jar包及依赖包



### log4j.properties日志配置文件

log4j.rootLogger=debug, stdout

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

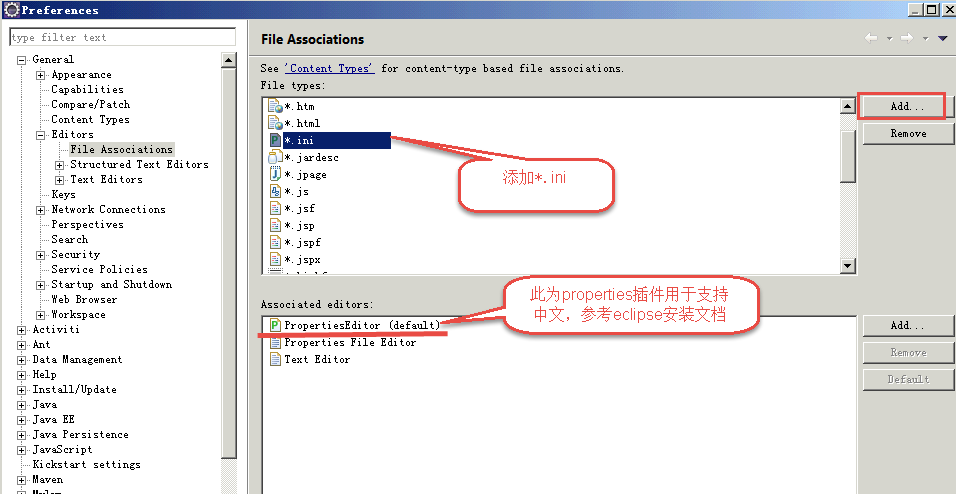
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m %n

### shiro.ini

通过Shiro.ini配置文件初始化SecurityManager环境。

配置 eclipse支持ini文件编辑：



在eclipse配置后，在classpath创建shiro.ini配置文件，为了方便测试将用户名和密码配置的shiro.ini配置文件中：

[users]

zhang=123

lisi=123

### 认证代码

// 用户登陆、用户退出

@Test

**public** **void** testLoginLogout() {

// 构建SecurityManager工厂，IniSecurityManagerFactory可以从ini文件中初始化SecurityManager环境

Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(

"classpath:shiro.ini");

// 通过工厂创建SecurityManager

SecurityManager securityManager = factory.getInstance();

// 将securityManager设置到运行环境中

SecurityUtils.*setSecurityManager*(securityManager);

// 创建一个Subject实例，该实例认证要使用上边创建的securityManager进行

Subject subject = SecurityUtils.*getSubject*();

// 创建token令牌，记录用户认证的身份和凭证即账号和密码

UsernamePasswordToken token = **new** UsernamePasswordToken("zhang", "123");

**try** {

// 用户登陆

subject.login(token);

} **catch** (AuthenticationException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

// 用户认证状态

Boolean isAuthenticated = subject.isAuthenticated();

System.*out*.println("用户认证状态：" + isAuthenticated);

// 用户退出

subject.logout();

isAuthenticated = subject.isAuthenticated();

System.*out*.println("用户认证状态：" + isAuthenticated);

}

### 认证执行流程

1. 创建token令牌，token中有用户提交的认证信息即账号和密码
2. 执行subject.login(token)，最终由securityManager通过Authenticator进行认证
3. Authenticator的实现ModularRealmAuthenticator调用realm从ini配置文件取用户真实的账号和密码，这里使用的是IniRealm（shiro自带）
4. IniRealm先根据token中的账号去ini中找该账号，如果找不到则给ModularRealmAuthenticator返回null，如果找到则匹配密码，匹配密码成功则认证通过。

### 常见的异常

* UnknownAccountException

账号不存在异常如下：

org.apache.shiro.authc.UnknownAccountException: No account found for user。。。。

* IncorrectCredentialsException

当输入密码错误会抛此异常，如下：

org.apache.shiro.authc.IncorrectCredentialsException: Submitted credentials for token [org.apache.shiro.authc.UsernamePasswordToken - zhangsan, rememberMe=false] did not match the expected credentials.

更多如下：

DisabledAccountException（帐号被禁用）

LockedAccountException（帐号被锁定）

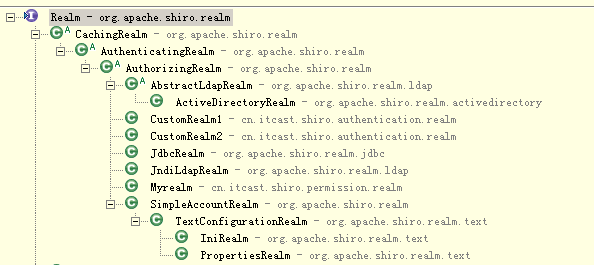
ExcessiveAttemptsException（登录失败次数过多）

ExpiredCredentialsException（凭证过期）等

## 自定义Realm

上边的程序使用的是Shiro自带的IniRealm，IniRealm从ini配置文件中读取用户的信息，大部分情况下需要从系统的数据库中读取用户信息，所以需要自定义realm。

### shiro提供的realm



最基础的是Realm接口，CachingRealm负责缓存处理，AuthenticationRealm负责认证，AuthorizingRealm负责授权，通常自定义的realm继承AuthorizingRealm。

### 自定义Realm

**public** **class** CustomRealm1 **extends** AuthorizingRealm {

@Override

**public** String getName() {

**return** "customRealm1";

}

//支持UsernamePasswordToken

@Override

**public** **boolean** supports(AuthenticationToken token) {

**return** token **instanceof** UsernamePasswordToken;

}

//认证

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) **throws** AuthenticationException {

//从token中 获取用户身份信息

String username = (String) token.getPrincipal();

//拿username从数据库中查询

//....

//如果查询不到则返回null

**if**(!username.equals("zhang")){//这里模拟查询不到

**return** **null**;

}

//获取从数据库查询出来的用户密码

String password = "123";//这里使用静态数据模拟。。

//返回认证信息由父类AuthenticatingRealm进行认证

SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(

username, password, getName());

**return** simpleAuthenticationInfo;

}

//授权

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

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

**return** **null**;

}

}

### shiro-realm.ini

[main]

#自定义 realm

customRealm=cn.itcast.shiro.authentication.realm.CustomRealm1

#将realm设置到securityManager

securityManager.realms=$customRealm

思考：这里为什么不用配置[users]了？？

### 测试代码

测试代码同入门程序，将ini的地址修改为shiro-realm.ini。

分别模拟账号不存在、密码错误、账号和密码正确进行测试。

## 散列算法

散列算法一般用于生成一段文本的摘要信息，散列算法不可逆，将内容可以生成摘要，无法将摘要转成原始内容。散列算法常用于对密码进行散列，常用的散列算法有MD5、SHA。

一般散列算法需要提供一个salt（盐）与原始内容生成摘要信息，这样做的目的是为了安全性，比如：111111的md5值是：96e79218965eb72c92a549dd5a330112，拿着“96e79218965eb72c92a549dd5a330112”去md5破解网站很容易进行破解，如果要是对111111和salt（盐，一个随机数）进行散列，这样虽然密码都是111111加不同的盐会生成不同的散列值。

### 例子

//md5加密，不加盐

String password\_md5 = **new** Md5Hash("111111").toString();

System.*out*.println("md5加密，不加盐="+password\_md5);

//md5加密，加盐，一次散列

String password\_md5\_sale\_1 = **new** Md5Hash("111111", "eteokues", 1).toString();

System.*out*.println("password\_md5\_sale\_1="+password\_md5\_sale\_1);

String password\_md5\_sale\_2 = **new** Md5Hash("111111", "uiwueylm", 1).toString();

System.*out*.println("password\_md5\_sale\_2="+password\_md5\_sale\_2);

//两次散列相当于md5(md5())

//使用SimpleHash

String simpleHash = **new** SimpleHash("MD5", "111111", "eteokues",1).toString();

System.*out*.println(simpleHash);

### 在realm中使用

实际应用是将盐和散列后的值存在数据库中，自动realm从数据库取出盐和加密后的值由shiro完成密码校验。

#### 自定义realm

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) **throws** AuthenticationException {

//用户账号

String username = (String) token.getPrincipal();

//根据用户账号从数据库取出盐和加密后的值

//..这里使用静态数据

//如果根据账号没有找到用户信息则返回null，shiro抛出异常“账号不存在”

//按照固定规则加密码结果 ，此密码 要在数据库存储，原始密码 是111111，盐是eteokues

String password = "cb571f7bd7a6f73ab004a70322b963d5";

//盐，随机数，此随机数也在数据库存储

String salt = "eteokues";

//返回认证信息

SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(

username, password, ByteSource.Util.*bytes*(salt),getName());

**return** simpleAuthenticationInfo;

}

#### realm配置

配置shiro-cryptography.ini

[main]

#定义凭证匹配器

credentialsMatcher=org.apache.shiro.authc.credential.HashedCredentialsMatcher

#散列算法

credentialsMatcher.hashAlgorithmName=md5

#散列次数

credentialsMatcher.hashIterations=1

#将凭证匹配器设置到realm

customRealm=cn.itcast.shiro.authentication.realm.CustomRealm2

customRealm.credentialsMatcher=$credentialsMatcher

securityManager.realms=$customRealm

#### 测试代码

测试代码同上个章节，注意修改ini路径。

# shiro授权

## 授权流程



## 授权方式

Shiro 支持三种方式的授权：

* 编程式：通过写if/else 授权代码块完成：

Subject subject = SecurityUtils.getSubject();

if(subject.hasRole(“admin”)) {

//有权限

} else {

//无权限

}

* 注解式：通过在执行的Java方法上放置相应的注解完成：

@RequiresRoles("admin")

public void hello() {

//有权限

}

* JSP/GSP 标签：在JSP/GSP 页面通过相应的标签完成：

<shiro:hasRole name="admin">

<!— 有权限—>

</shiro:hasRole>

本教程序授权测试使用第一种编程方式，实际与web系统集成使用后两种方式。

## 授权测试

### shiro-permission.ini

创建存放权限的配置文件shiro-permission.ini，如下：

[users]

#用户zhang的密码是123，此用户具有role1和role2两个角色

zhang=123,role1,role2

wang=123,role2

[roles]

#角色role1对资源user拥有create、update权限

role1=user:create,user:update

#角色role2对资源user拥有create、delete权限

role2=user:create,user:delete

#角色role3对资源user拥有create权限

role3=user:create

在ini文件中用户、角色、权限的配置规则是：“用户名=密码，角色1，角色2...” “角色=权限1，权限2...”，首先根据用户名找角色，再根据角色找权限，角色是权限集合。

### 权限字符串规则

权限字符串的规则是：“资源标识符：操作：资源实例标识符”，意思是对哪个资源的哪个实例具有什么操作，“:”是资源/操作/实例的分割符，权限字符串也可以使用\*通配符。

例子：

用户创建权限：user:create，或user:create:\*

用户修改实例001的权限：user:update:001

用户实例001的所有权限：user：\*：001

### 测试代码

测试代码同认证代码，注意ini地址改为shiro-permission.ini，主要学习下边授权的方法，**注意：在用户认证通过后执行下边的授权代码。**

@Test

**public** **void** testPermission() {

// 从ini文件中创建SecurityManager工厂

Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(

"classpath:shiro-permission.ini");

// 创建SecurityManager

SecurityManager securityManager = factory.getInstance();

// 将securityManager设置到运行环境

SecurityUtils.*setSecurityManager*(securityManager);

// 创建主体对象

Subject subject = SecurityUtils.*getSubject*();

// 对主体对象进行认证

// 用户登陆

// 设置用户认证的身份(principals)和凭证(credentials)

UsernamePasswordToken token = **new** UsernamePasswordToken("zhang", "123");

**try** {

subject.login(token);

} **catch** (AuthenticationException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

// 用户认证状态

Boolean isAuthenticated = subject.isAuthenticated();

System.*out*.println("用户认证状态：" + isAuthenticated);

// 用户授权检测 基于角色授权

// 是否有某一个角色

System.*out*.println("用户是否拥有一个角色：" + subject.hasRole("role1"));

// 是否有多个角色

System.*out*.println("用户是否拥有多个角色：" + subject.hasAllRoles(Arrays.*asList*("role1", "role2")));

// subject.checkRole("role1");

// subject.checkRoles(Arrays.asList("role1", "role2"));

// 授权检测，失败则抛出异常

// subject.checkRole("role22");

// 基于资源授权

System.*out*.println("是否拥有某一个权限：" + subject.isPermitted("user:delete"));

System.*out*.println("是否拥有多个权限：" + subject.isPermittedAll("user:create:1", "user:delete"));

//检查权限

subject.checkPermission("sys:user:delete");

subject.checkPermissions("user:create:1","user:delete");

}

### 基于角色的授权

// 用户授权检测 基于角色授权

// 是否有某一个角色

System.*out*.println("用户是否拥有一个角色：" + subject.hasRole("role1"));

// 是否有多个角色

System.*out*.println("用户是否拥有多个角色：" + subject.hasAllRoles(Arrays.*asList*("role1", "role2")));

对应的check方法：

subject.checkRole("role1");

subject.checkRoles(Arrays.*asList*("role1", "role2"));

上边check方法如果授权失败则抛出异常：

org.apache.shiro.authz.UnauthorizedException: Subject does not have role [.....]

### 基于资源授权

// 基于资源授权

System.*out*.println("是否拥有某一个权限：" + subject.isPermitted("user:delete"));

System.*out*.println("是否拥有多个权限：" + subject.isPermittedAll("user:create:1", "user:delete"));

对应的check方法：

subject.checkPermission("sys:user:delete");

subject.checkPermissions("user:create:1","user:delete");

上边check方法如果授权失败则抛出异常：

org.apache.shiro.authz.UnauthorizedException: Subject does not have permission [....]

## 自定义realm

与上边认证自定义realm一样，大部分情况是要从数据库获取权限数据，这里直接实现基于资源的授权。

### realm代码

在认证章节写的自定义realm类中完善doGetAuthorizationInfo方法，此方法需要完成：根据用户身份信息从数据库查询权限字符串，由shiro进行授权。

// 授权

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

// 获取身份信息

String username = (String) principals.getPrimaryPrincipal();

// 根据身份信息从数据库中查询权限数据

//....这里使用静态数据模拟

List<String> permissions = **new** ArrayList<String>();

permissions.add("user:create");

permissions.add("user.delete");

//将权限信息封闭为AuthorizationInfo

SimpleAuthorizationInfo simpleAuthorizationInfo = **new** SimpleAuthorizationInfo();

**for**(String permission:permissions){

simpleAuthorizationInfo.addStringPermission(permission);

}

**return** simpleAuthorizationInfo;

}

### shiro-realm.ini

ini配置文件还使用认证阶段使用的，不用改变。

思考：shiro-permission.ini中的[roles]为什么不需要了？？

### 测试代码

同上边的授权测试代码，注意修改ini地址为shiro-realm.ini。

### 授权执行流程

1. 执行subject.isPermitted("user:create")
2. securityManager通过ModularRealmAuthorizer进行授权
3. ModularRealmAuthorizer调用realm获取权限信息
4. ModularRealmAuthorizer再通过permissionResolver解析权限字符串，校验是否匹配

# shiro与项目集成开发

## shiro与spring web项目整合

shiro与springweb项目整合在“基于url拦截实现的工程”基础上整合，基于url拦截实现的工程的技术架构是springmvc+mybatis，整合注意两点：

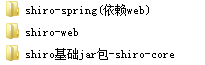
1、shiro与spring整合

2、加入shiro对web应用的支持

### 取消原springmvc认证和授权拦截器

去掉springmvc.xml中配置的LoginInterceptor和PermissionInterceptor拦截器。

### 加入shiro的 jar包



### web.xml添加shiro Filter

<!-- shiro过虑器，DelegatingFilterProx会从spring容器中找shiroFilter -->

<filter>

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

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

<init-param>

<param-name>targetFilterLifecycle</param-name>

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

</init-param>

</filter>

<filter-mapping>

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

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

</filter-mapping>

### applicationContext-shiro.xml

<!-- Shiro 的Web过滤器 -->

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

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

<!-- 如果没有认证将要跳转的登陆地址，http可访问的url，如果不在表单认证过虑器FormAuthenticationFilter中指定此地址就为身份认证地址 -->

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

<!-- 没有权限跳转的地址 -->

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

<!-- shiro拦截器配置 -->

<property name=*"filters"*>

<map>

<entry key=*"authc"* value-ref=*"formAuthenticationFilter"* />

</map>

</property>

<property name=*"filterChainDefinitions"*>

<value>

<!-- 必须通过身份认证方可访问，身份认 证的url必须和过虑器中指定的loginUrl一致 -->

/loginsubmit.action = authc

<!-- 退出拦截，请求logout.action执行退出操作 -->

/logout.action = logout

<!-- 无权访问页面 -->

/refuse.jsp = anon

<!-- roles[XX]表示有XX角色才可访问 -->

/item/list.action = roles[item],authc

/js/\*\* anon

/images/\*\* anon

/styles/\*\* anon

<!-- user表示身份认证通过或通过记住我认证通过的可以访问 -->

/\*\* = user

<!-- /\*\*放在最下边，如果一个url有多个过虑器则多个过虑器中间用逗号分隔，如：/\*\* = user,roles[admin] -->

</value>

</property>

</bean>

<!-- 安全管理器 -->

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

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

</bean>

<!-- 自定义 realm -->

<bean id=*"userRealm"* class=*"cn.itcast.ssm.realm.CustomRealm1"*>

</bean>

<!-- 基于Form表单的身份验证过滤器，不配置将也会注册此过虑器，表单中的用户账号、密码及loginurl将采用默认值，建议配置 -->

<bean id=*"formAuthenticationFilter"*

class=*"org.apache.shiro.web.filter.authc.FormAuthenticationFilter"*>

<!-- 表单中账号的input名称 -->

<property name=*"usernameParam"* value=*"usercode"* />

<!-- 表单中密码的input名称 -->

<property name=*"passwordParam"* value=*"password"* />

<!-- <property name="rememberMeParam" value="rememberMe"/> -->

<!-- loginurl：用户登陆地址，此地址是可以http访问的url地址 -->

<property name=*"loginUrl"* value=*"/loginsubmit.action"* />

</bean>

*securityManager*：这个属性是必须的。

*loginUrl*：没有登录认证的用户请求将跳转到此地址，不是必须的属性，不输入地址的话会自动寻找项目web项目的根目录下的”/login.jsp”页面。

*unauthorizedUrl*：没有权限默认跳转的页面。

### 使用shiro注解授权

在springmvc.xml中配置shiro注解支持，可在controller方法中使用shiro注解配置权限：

<!-- 开启aop，对类代理 -->

<aop:config proxy-target-class=*"true"*></aop:config>

<!-- 开启shiro注解支持 -->

<bean

class=*"*

*org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor"*>

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

</bean>

修改Controller代码，在方法上添加授权注解，如下：

// 查询商品列表

@RequestMapping("/queryItem")

@RequiresPermissions("item:query")

**public** ModelAndView queryItem() **throws** Exception {

上边代码@RequiresPermissions("item:query")表示必须拥有“item:query”权限方可执行。

其它的方法参考示例添加注解，一边添加一边思考这比基于url拦截有什么好处。

### 自定义realm

此realm先不从数据库查询权限数据，当前需要先将shiro整合完成，在上边章节定义的realm基础上修改。

**public** **class** CustomRealm1 **extends** AuthorizingRealm {

@Autowired

**private** SysService sysService;

@Override

**public** String getName() {

**return** "customRealm";

}

// 支持什么类型的token

@Override

**public** **boolean** supports(AuthenticationToken token) {

**return** token **instanceof** UsernamePasswordToken;

}

// 认证

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) **throws** AuthenticationException {

// 从token中 获取用户身份信息

String username = (String) token.getPrincipal();

// 拿username从数据库中查询

// ....

// 如果查询不到则返回null

**if** (!username.equals("zhang")) {// 这里模拟查询不到

**return** **null**;

}

// 获取从数据库查询出来的用户密码

String password = "123";// 这里使用静态数据模拟。。

// 根据用户id从数据库取出菜单

//...先用静态数据

List<SysPermission> menus = **new** ArrayList<SysPermission>();;

SysPermission sysPermission\_1 = **new** SysPermission();

sysPermission\_1.setName("商品管理");

sysPermission\_1.setUrl("/item/queryItem.action");

SysPermission sysPermission\_2 = **new** SysPermission();

sysPermission\_2.setName("用户管理");

sysPermission\_2.setUrl("/user/query.action");

menus.add(sysPermission\_1);

menus.add(sysPermission\_2);

// 构建用户身体份信息

ActiveUser activeUser = **new** ActiveUser();

activeUser.setUserid(username);

activeUser.setUsername(username);

activeUser.setUsercode(username);

activeUser.setMenus(menus);

// 返回认证信息由父类AuthenticatingRealm进行认证

SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(

activeUser, password, getName());

**return** simpleAuthenticationInfo;

}

// 授权

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

// 获取身份信息

ActiveUser activeUser = (ActiveUser) principals.getPrimaryPrincipal();

//用户id

String userid = activeUser.getUserid();

// 根据用户id从数据库中查询权限数据

// ....这里使用静态数据模拟

List<String> permissions = **new** ArrayList<String>();

permissions.add("item:query");

permissions.add("item:update");

// 将权限信息封闭为AuthorizationInfo

SimpleAuthorizationInfo simpleAuthorizationInfo = **new** SimpleAuthorizationInfo();

**for** (String permission : permissions) {

simpleAuthorizationInfo.addStringPermission(permission);

}

**return** simpleAuthorizationInfo;

}

}

### 登录

//用户登陆页面

@RequestMapping("/login")

**public** String login()**throws** Exception{

**return** "login";

}

// 用户登陆提交

@RequestMapping("/loginsubmit")

**public** String loginsubmit(Model model, HttpServletRequest request)

**throws** Exception {

// shiro在认证过程中出现错误后将异常类路径通过request返回

String exceptionClassName = (String) request

.getAttribute("shiroLoginFailure");

**if** (UnknownAccountException.**class**.getName().equals(exceptionClassName)) {

**throw** **new** CustomException("账号不存在");

} **else** **if** (IncorrectCredentialsException.**class**.getName().equals(

exceptionClassName)) {

**throw** **new** CustomException("用户名/密码错误");

} **else**{

**throw** **new** Exception();//最终在异常处理器生成未知错误

}

}

### 首页

由于session由shiro管理，需要修改首页的controller方法：

//系统首页

@RequestMapping("/first")

**public** String first(Model model)**throws** Exception{

//主体

Subject subject = SecurityUtils.*getSubject*();

//身份

ActiveUser activeUser = (ActiveUser) subject.getPrincipal();

model.addAttribute("activeUser", activeUser);

**return** "/first";

}

### 退出

由于使用shiro的sessionManager，不用开发退出功能，使用shiro的logout拦截器即可。

<!-- 退出拦截，请求logout.action执行退出操作 -->

/logout.action = logout

### 无权限refuse.jsp

当用户无操作权限，shiro将跳转到refuse.jsp页面。

参考：applicationContext-shiro.xml

## realm连接数据库

### 添加凭证匹配器

添加凭证匹配器实现md5加密校验。

修改applicationContext-shiro.xml：

<!-- 凭证匹配器 -->

<bean id=*"credentialsMatcher"*

class=*"org.apache.shiro.authc.credential.HashedCredentialsMatcher"*>

<property name=*"hashAlgorithmName"* value=*"md5"* />

<property name=*"hashIterations"* value=*"1"* />

</bean>

<!-- 自定义 realm -->

<bean id=*"userRealm"* class=*"cn.itcast.ssm.realm.CustomRealm1"*>

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

</bean>

### realm代码

修改realm代码从数据库中查询用户身份信息和权限信息，将sysService注入realm。

**public** **class** CustomRealm1 **extends** AuthorizingRealm {

@Autowired

**private** SysService sysService;

@Override

**public** String getName() {

**return** "customRealm";

}

// 支持什么类型的token

@Override

**public** **boolean** supports(AuthenticationToken token) {

**return** token **instanceof** UsernamePasswordToken;

}

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) **throws** AuthenticationException {

// 从token中获取用户身份

String usercode = (String) token.getPrincipal();

SysUser sysUser = **null**;

**try** {

sysUser = sysService.findSysuserByUsercode(usercode);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

// 如果账号不存在

**if** (sysUser == **null**) {

**throw** **new** UnknownAccountException("账号找不到");

}

// 根据用户id取出菜单

List<SysPermission> menus = **null**;

**try** {

menus = sysService.findMenuList(sysUser.getId());

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

// 用户密码

String password = sysUser.getPassword();

//盐

String salt = sysUser.getSalt();

// 构建用户身体份信息

ActiveUser activeUser = **new** ActiveUser();

activeUser.setUserid(sysUser.getId());

activeUser.setUsername(sysUser.getUsername());

activeUser.setUsercode(sysUser.getUsercode());

activeUser.setMenus(menus);

SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(

activeUser, password, ByteSource.Util.*bytes*(salt),getName());

**return** simpleAuthenticationInfo;

}

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

//身份信息

ActiveUser activeUser = (ActiveUser) principals.getPrimaryPrincipal();

//用户id

String userid = activeUser.getUserid();

//获取用户权限

List<SysPermission> permissions = **null**;

**try** {

permissions = sysService.findSysPermissionList(userid);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//构建shiro授权信息

SimpleAuthorizationInfo simpleAuthorizationInfo = **new** SimpleAuthorizationInfo();

**for**(SysPermission sysPermission:permissions){

simpleAuthorizationInfo.addStringPermission(sysPermission.getPercode());

}

**return** simpleAuthorizationInfo;

}

}

## 缓存

shiro每个授权都会通过realm获取权限信息，为了提高访问速度需要添加缓存，第一次从realm中读取权限数据，之后不再读取，这里Shiro和Ehcache整合。

### 添加Ehcache的jar包



### 配置

在applicationContext-shiro.xml中配置缓存管理器。

<!-- 安全管理器 -->

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

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

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

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

</bean>

<!-- 缓存管理器 -->

<bean id=*"cacheManager"* class=*"org.apache.shiro.cache.ehcache.EhCacheManager"*>

</bean>

## session管理

在applicationContext-shiro.xml中配置sessionManager：

<!-- 安全管理器 -->

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

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

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

</bean>

<!-- 会话管理器 -->

<bean id=*"sessionManager"* class=*"org.apache.shiro.web.session.mgt.DefaultWebSessionManager"*>

<!-- session的失效时长，单位毫秒 -->

<property name=*"globalSessionTimeout"* value=*"600000"*/>

<!-- 删除失效的session -->

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

</bean>

## 验证码

### 自定义FormAuthenticationFilter

需要在验证账号和名称之前校验验证码。

**public** **class** MyFormAuthenticationFilter **extends** FormAuthenticationFilter {

**protected** **boolean** onAccessDenied(ServletRequest request,

ServletResponse response, Object mappedValue) **throws** Exception {

// 校验验证码

// 从session获取正确的验证码

HttpSession session = ((HttpServletRequest)request).getSession();

//页面输入的验证码

String randomcode = request.getParameter("randomcode");

//从session中取出验证码

String validateCode = (String) session.getAttribute("validateCode");

**if** (!randomcode.equals(validateCode)) {

// randomCodeError表示验证码错误

request.setAttribute("shiroLoginFailure", "randomCodeError");

//拒绝访问，不再校验账号和密码

**return** **true**;

}

**return** **super**.onAccessDenied(request, response, mappedValue);

}

}

### 修改FormAuthenticationFilter配置

修改applicationContext-shiro.xml中对FormAuthenticationFilter的配置。

<bean id=*"formAuthenticationFilter"*

class=*"org.apache.shiro.web.filter.authc.FormAuthenticationFilter"*>

改为

<bean id=*"formAuthenticationFilter"*

class=*"cn.itcast.ssm.shiro.MyFormAuthenticationFilter"*>

### 登陆页面

添加验证码：

<TR>

<TD>验证码：</TD>

<TD><input id=*"randomcode"* name=*"randomcode"* size=*"8"* /> <img

id=*"randomcode\_img"* src=*"*${baseurl}*validatecode.jsp"* alt=*""*

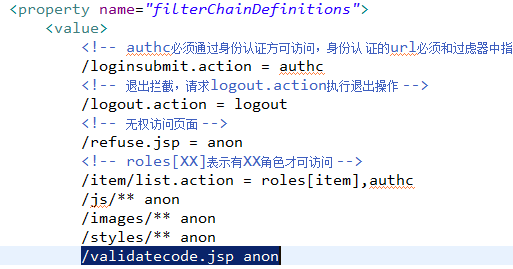
width=*"56"* height=*"20"* align=*'absMiddle'* /> <a

href=*javascript:randomcode\_refresh()*>刷新</a></TD>

</TR>

### 配置validatecode.jsp匿名访问

修改applicationContext-shiro.xml：



## 记住我

用户登陆选择“自动登陆”本次登陆成功会向cookie写身份信息，下次登陆从cookie中取出身份信息实现自动登陆。

### 用户身份实现java.io.Serializable接口

向cookie记录身份信息需要用户身份信息对象实现序列化接口，如下：





### 配置

<!-- 安全管理器 -->

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

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

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

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

<!-- 记住我 -->

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

</bean>

<!-- rememberMeManager管理器 -->

<bean id=*"rememberMeManager"* class=*"org.apache.shiro.web.mgt.CookieRememberMeManager"*>

<property name=*"cookie"* ref=*"rememberMeCookie"* />

</bean>

<!-- 记住我cookie -->

<bean id=*"rememberMeCookie"* class=*"org.apache.shiro.web.servlet.SimpleCookie"*>

<constructor-arg value=*"rememberMe"* />

<!-- 记住我cookie生效时间30天 -->

<property name=*"maxAge"* value=*"2592000"* />

</bean>

修改formAuthenticationFitler添加页面中“记住我checkbox”的input名称：

<bean id=*"formAuthenticationFilter"*

class=*"cn.itcast.ssm.shiro.MyFormAuthenticationFilter"*>

<!-- 表单中账号的input名称 -->

<property name=*"usernameParam"* value=*"usercode"* />

<!-- 表单中密码的input名称 -->

<property name=*"passwordParam"* value=*"password"* />

<property name=*"rememberMeParam"* value=*"rememberMe"*/>

<!-- loginurl：用户登陆地址，此地址是可以http访问的url地址 -->

<property name=*"loginUrl"* value=*"/loginsubmit.action"* />

</bean>

### 登陆页面

在login.jsp中添加“记住我”checkbox。

<TR>

<TD></TD>

<TD>

<input type=*"checkbox"* name=*"rememberMe"* />自动登陆

</TD>

</TR>

# 附：

## shiro过虑器

|  |  |
| --- | --- |
| 过滤器简称 | 对应的java类 |
| anon | org.apache.shiro.web.filter.authc.AnonymousFilter |
| authc | org.apache.shiro.web.filter.authc.FormAuthenticationFilter |
| authcBasic | org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter |
| perms | org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter |
| port | org.apache.shiro.web.filter.authz.PortFilter |
| rest | org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter |
| roles | org.apache.shiro.web.filter.authz.RolesAuthorizationFilter |
| ssl | org.apache.shiro.web.filter.authz.SslFilter |
| user | org.apache.shiro.web.filter.authc.UserFilter |
| logout | org.apache.shiro.web.filter.authc.LogoutFilter |

anon:例子/admins/\*\*=anon 没有参数，表示可以匿名使用。

authc:例如/admins/user/\*\*=authc表示需要认证(登录)才能使用，没有参数

roles：例子/admins/user/\*\*=roles[admin],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，当有多个参数时，例如admins/user/\*\*=roles["admin,guest"],每个参数通过才算通过，相当于hasAllRoles()方法。

perms：例子/admins/user/\*\*=perms[user:add:\*],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，例如/admins/user/\*\*=perms["user:add:\*,user:modify:\*"]，当有多个参数时必须每个参数都通过才通过，想当于isPermitedAll()方法。

rest：例子/admins/user/\*\*=rest[user],根据请求的方法，相当于/admins/user/\*\*=perms[user:method] ,其中method为post，get，delete等。

port：例子/admins/user/\*\*=port[8081],当请求的url的端口不是8081是跳转到schemal://serverName:8081?queryString,其中schmal是协议http或https等，serverName是你访问的host,8081是url配置里port的端口，queryString

是你访问的url里的？后面的参数。

authcBasic：例如/admins/user/\*\*=authcBasic没有参数表示httpBasic认证

ssl:例子/admins/user/\*\*=ssl没有参数，表示安全的url请求，协议为https

user:例如/admins/user/\*\*=user没有参数表示必须存在用户，当登入操作时不做检查

注：

anon，authcBasic，auchc，user是认证过滤器，

perms，roles，ssl，rest，port是授权过滤器

## shiro的jsp标签

Jsp页面添加：

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

|  |  |
| --- | --- |
| 标签名称 | 标签条件（均是显示标签内容） |
| <shiro:authenticated> | 登录之后 |
| <shiro:notAuthenticated> | 不在登录状态时 |
| <shiro:guest> | 用户在没有RememberMe时 |
| <shiro:user> | 用户在RememberMe时 |
| <shiro:hasAnyRoles name="abc,123" > | 在有abc或者123角色时 |
| <shiro:hasRole name="abc"> | 拥有角色abc |
| <shiro:lacksRole name="abc"> | 没有角色abc |
| <shiro:hasPermission name="abc"> | 拥有权限资源abc |
| <shiro:lacksPermission name="abc"> | 没有abc权限资源 |
| <shiro:principal> | 显示用户身份名称 |

<shiro:principal property="username"/> 显示用户身份中的属性值