shiro第一天 基于url权限管理 shiro基础

# 课程目标：

1、了解基于资源的权限管理方式

2、掌握权限数据模型

3、掌握基于url的权限管理（不使用shiro实现权限管理）

4、shiro实现用户认证

5、shiro实现用户授权

6、shiro与企业web项目整合开发的方法

# 课程安排

整个课程是系统架构设计相关的课程。

第一天：基于url权限管理 shiro基础

原理知识：对权限管理的理解

什么是权限管理

用户认证

用户授权

权限管理的解决方案

基于url的权限管理开发实现（重点）

shiro基础：

shiro架构（重点）

shiro进行用户认证的方法（重点）

shiro进行用户授权的方法（重点）

第二天：shiro应用

shiro与企业web项目整合开发的方法

# 权限管理原理知识

## 什么是权限管理

只要有用户参与的系统一般都要有权限管理，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" \t "_blank)控制用户可以访问而且只能访问自己被授权的资源。

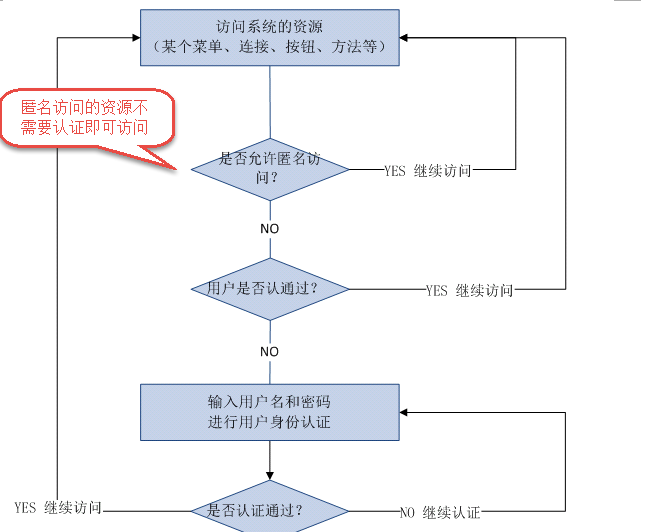
权限管理包括用户认证和授权两部分。

## 用户认证

### 概念

用户认证，用户去访问系统，系统要验证用户身份的合法性。最常用的用户身份验证的方法：1、用户名密码方式、2、指纹打卡机、3、基于证书验证方法。。系统验证用户身份合法，用户方可访问系统的资源。

### 用户认证流程



### 关键对象

subject：主体，理解为用户,可能是程序，都要去访问系统的资源，系统需要对subject进行身份认证。

principal：身份信息，通常是唯一的，一个主体还有多个身份信息，但是都有一个主身份信息（primary principal）

credential：凭证信息，可以是密码、证书、指纹。

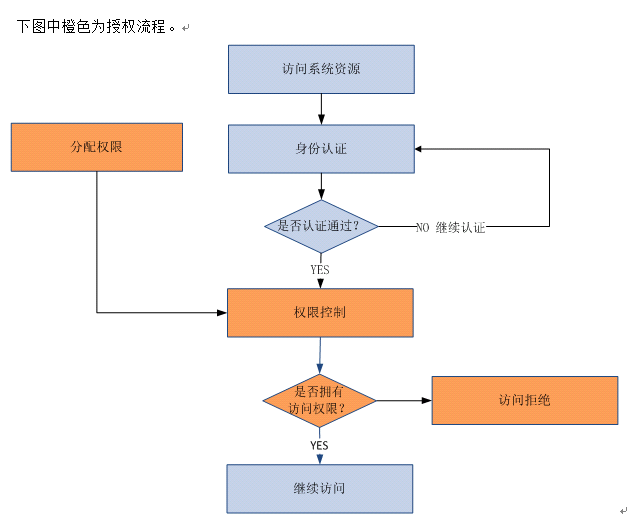
**总结：主体在进行身份认证时需要提供身份信息和凭证信息。**

## 用户授权

### 概念

用户授权，简单理解为访问控制，在用户认证通过后，系统对用户访问资源进行控制，用户具有资源的访问权限方可访问。

### 授权流程



### 关键对象

授权的过程理解为：who对what(which)进行how操作。

who：主体即subject，subject在认证通过后系统进行访问控制。

what(which)：资源(**Resource**)，subject必须具备资源的访问权限才可访问该 资源。资源比如：系统用户列表页面、商品修改菜单、商品id为001的商品信息。

资源分为**资源类型和资源实例**：

系统的用户信息就是资源类型，相当于java类。

系统中id为001的用户就是资源实例，相当于new的java对象。

how：权限/许可(**permission**) ，针对资源的权限或许可，subject具有permission访问资源，如何访问/操作需要定义permission，权限比如：用户添加、用户修改、商品删除。

### 权限模型

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。

### 分配权限

用户需要分配相应的权限才可访问相应的资源。权限是对于资源的操作许可。

通常给用户分配资源权限需要将权限信息持久化，比如存储在关系数据库中。

把用户信息、权限管理、用户分配的权限信息写到数据库（权限数据模型）

### 权限控制(授权核心)

#### 基于角色的访问控制

RBAC(role based access control)，基于角色的访问控制。

比如：

系统角色包括：部门经理、总经理。（角色针对用户来划分）

系统代码中实现：

//如果该user是部门经理则可以访问if中的代码

if(user.hasRole('部门经理')){

//系统资源内容

//用户报表查看

}

问题：

角色针对人划分的，人作为用户在系统中属于活动内容，如果该角色可以访问的资源出现变更，需要修改你的代码了，比如：需要变更为部门经理和总经理都可以进行用户报表查看，代码改为：

if(user.hasRole('部门经理') || user.hasRole('总经理') ){

//系统资源内容

//用户报表查看

}

基于角色的访问控制是不利于系统维护(可扩展性不强)。

#### 基于资源的访问控制

RBAC(Resource based access control)，基于资源的访问控制。

资源在系统中是不变的，比如资源有：类中的方法，页面中的按钮。

对资源的访问需要具有permission权限，代码可以写为：

if(user.hasPermission ('用户报表查看（权限标识符）')){

//系统资源内容

//用户报表查看

}

上边的方法就可以解决用户角色变更不用修改上边权限控制的代码。

如果需要变更权限只需要在分配权限模块去操作，给部门经理或总经理增或删除权限。

**建议使用基于资源的访问控制实现权限管理。**

# 权限管理解决方案

## 什么是粗粒度和细粒度权限

粗粒度权限管理，对资源类型的权限管理。资源类型比如：菜单、url连接、用户添加页面、用户信息、类方法、页面中按钮。。

粗粒度权限管理比如：超级管理员可以访问户添加页面、用户信息等全部页面。

部门管理员可以访问用户信息页面包括 页面中所有按钮。

细粒度权限管理，对资源实例的权限管理。资源实例就资源类型的具体化，比如：用户id为001的修改连接，1110班的用户信息、行政部的员工。

**细粒度权限管理就是数据级别的权限管理。**

细粒度权限管理比如：部门经理只可以访问本部门的员工信息，用户只可以看到自己的菜单，大区经理只能查看本辖区的销售订单。。

粗粒度和细粒度例子：

系统有一个用户列表查询页面，对用户列表查询分权限，如果粗颗粒管理，张三和李四都有用户列表查询的权限，张三和李四都可以访问用户列表查询。

进一步进行细颗粒管理，张三（行政部）和李四(开发部)只可以查询自己本部门的用户信息。张三只能查看行政部 的用户信息，李四只能查看开发部门的用户信息。**细粒度权限管理就是数据级别的权限管理。**

## 如何实现粗粒度和细粒度权限管理

如何实现粗粒度权限管理？

粗粒度权限管理比较容易将权限管理的代码抽取出来在系统架构级别统一处理。比如：通过springmvc的拦截器实现授权。

如何实现细粒度权限管理？

对细粒度权限管理在数据级别是没有共性可言，针对细粒度权限管理就是系统业务逻辑的一部分，如果在业务层去处理相对比较简单，如果将细粒度权限管理统一在系统架构级别去抽取，比较困难，即使抽取的功能可能也存在扩展不强。

建议细粒度权限管理在业务层去控制。

比如：部门经理只查询本部门员工信息，在service接口提供一个部门id的参数，controller中根据当前用户的信息得到该 用户属于哪个部门，调用service时将部门id传入service，实现该用户只查询本部门的员工。

## 基于url拦截的方式实现

基于url拦截的方式实现在实际开发中比较常用的一种方式。

对于web系统，通过filter过虑器实现url拦截，也可以springmvc的拦截器实现基于url的拦截。

## 使用权限管理框架实现

对于粗粒度权限管理，建议使用优秀权限管理框架来实现，节省开发成功，提高开发效率。

shiro就是一个优秀权限管理框架。

# 基于url的权限管理

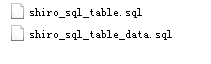
## 基于url权限管理流程



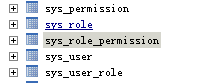
## 搭建环境

### 数据库

mysql5.1数据库中创建表：用户表、角色表、权限表(实质上是权限和资源的结合 )、用户角色表、角色权限表。



完成权限管理的数据模型创建。



### 开发环境

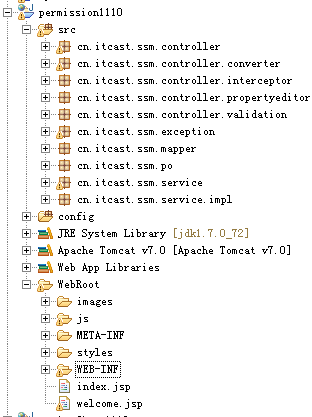
jdk1.7.0\_72

eclipse 3.7 indigo

技术架构：springmvc+mybatis+jquery easyui

### 系统工程架构

springmvc+mybatis+jquery easyui



### 系统登陆

系统 登陆相当 于用户身份认证，用户成功，要在session中记录用户的身份信息.

操作流程：

用户进行登陆页面

输入用户名和密码进行登陆

进行用户名和密码校验

如果校验通过，在session记录用户身份信息

#### 用户的身份信息

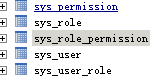
创建专门类用于记录用户身份信息。

**private** String **userId**;*//用户id***private** String **userCode**;*// 用户账号***private** String **username**;*// 用户名称*

#### mapper

mapper接口：　根据用户账号查询用户（sys\_user）信息（使用逆向工程生成的mapper）

使用逆向工程生成以下表的基础代码：



#### service（进行用户名和密码校验）

接口功能：根据用户的身份和密码 进行认证，如果认证通过，返回用户身份信息

认证过程：

根据用户身份（账号）查询数据库，如果查询不到用户不存在

对输入的密码 和数据库密码 进行比对，如果一致，认证通过

*/\*\*  
 \* 根据用户的身份和密码 进行认证，如果认证通过，返回用户身份信息  
 \*  
 \** ***@param userCode*** *账号  
 \** ***@param password*** *密码  
 \** ***@return*** *ActiveUser  
 \** ***@throws*** *Exception Exception  
 \*/*@Override  
**public** ActiveUser authenticat(String userCode, String password) **throws** Exception {}

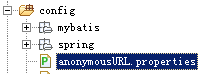
#### controller（记录session）

*/\*\*  
 \* 用户登陆提交方法  
 \*  
 \** ***@param session*** *session  
 \** ***@param randomcode*** *输入的验证码  
 \** ***@param usercode*** *用户账号  
 \** ***@param password*** *用户密码  
 \** ***@return*** *页面  
 \** ***@throws*** *Exception Exception  
 \*/*@RequestMapping(**"/login"**)  
**public** String login(HttpSession session, String randomcode, String usercode, String password) **throws** Exception {  
 *//校验验证码，防止恶性攻击  
 //从session获取正确验证码* String validateCode = (String) session.getAttribute(**"validateCode"**);  
  
 *//输入的验证和session中的验证进行对比* **if** (!randomcode.equals(validateCode)) {  
 *//抛出异常* **throw new** CustomException(**"验证码输入错误"**);  
 }  
  
 *//调用service校验用户账号和密码的正确性* ActiveUser activeUser = **sysService**.authenticat(usercode, password);  
  
 *//如果service校验通过，将用户身份记录到session* session.setAttribute(**"activeUser"**, activeUser);  
 *//重定向到商品查询页面* **return "redirect:/product/all"**;  
}

### 用户认证拦截器

#### anonymousURL.properties

配置可以匿名访问的url



#### 编写认证拦截器

*//在执行handler之前来执行的  
//用于用户认证校验、用户权限校验*@Override  
**public boolean** preHandle(HttpServletRequest request,  
 HttpServletResponse response, Object handler) **throws** Exception {  
  
 *//得到请求的url* String url = request.getRequestURI();  
  
 *//判断是否是公开 地址  
 //实际开发中需要公开 地址配置在配置文件中  
 //从配置中取逆名访问url* List<String> open\_urls = ResourcesUtil.*gekeyList*(**"anonymousURL"**);  
 *//遍历公开 地址，如果是公开 地址则放行* **for** (String open\_url : open\_urls) {  
 **if** (url.contains(open\_url)) {  
 *//如果是公开 地址则放行* **return true**;  
 }  
 }  
  
 *//判断用户身份在session中是否存在* HttpSession session = request.getSession();  
 ActiveUser activeUser = (ActiveUser) session.getAttribute(**"activeUser"**);  
 *//如果用户身份在session中存在放行* **if** (activeUser != **null**) {  
 **return true**;  
 }  
 *//执行到这里拦截，跳转到登陆页面，用户进行身份认证* request.getRequestDispatcher(**"/WEB-INF/jsps/user/login.jsp"**).forward(request, response);  
  
 *//如果返回false表示拦截不继续执行handler，如果返回true表示放行* **return false**;  
}

#### 配置拦截器

在springmvc.xml中配置拦截器

*<!--拦截器 -->*<**mvc:interceptors**>  
  
 <**mvc:interceptor**>  
 *<!-- 用户认证拦截 -->* <**mvc:mapping path="/\*\*"**/>  
 <**mvc:exclude-mapping path="/js/\*\*"**/>  
 <**mvc:exclude-mapping path="/styles/\*\*"**/>  
 <**mvc:exclude-mapping path="/images/\*\*"**/>  
 <**bean class="cn.qdgxy.shiro.web.interceptor.LoginInterceptor"**/>  
 </**mvc:interceptor**>  
</**mvc:interceptors**>

### 授权

#### commonURL.properties

在此配置文件配置公用访问地址，公用访问地址只要通过用户认证，不需要对公用访问地址分配权限即可访问。

#### 获取用户权限范围的菜单

思路：

在用户认证时，认证通过，根据用户id从数据库获取用户权限范围的菜单，将菜单的集合存储在session中。

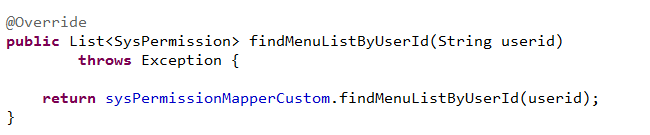
**public class** ActiveUser **implements** Serializable {  
  
 **private** Long **userId**;*//用户id* **private** String **userCode**;*// 用户账号* **private** String **username**;*// 用户名称* **private** List<SysPermission> **menus**;*// 菜单* **private** List<SysPermission> **permissions**;*// 权限*

mapper接口：根据用户id查询用户权限的菜单

*<!-- 根据用户id查询菜单 -->* <**select id="findMenuListByUserId" parameterType="string" resultType="SysPermission"**>  
 SELECT *\** FROM sys\_permission  
 WHERE TYPE = 'menu'  
 AND id IN (  
 SELECT sys\_permission\_id  
 FROM sys\_role\_permission  
 WHERE sys\_role\_id IN (  
 SELECT sys\_role\_id  
 FROM sys\_user\_role  
 WHERE sys\_user\_id = #{id}  
 )  
 )  
</**select**>

*/\*\*  
 \* 根据用户id查询菜单  
 \*  
 \** ***@param userId*** *用户id  
 \** ***@return*** *List*<*SysPermission*>  
 *\** ***@throws*** *Exception Exception  
 \*/*List<SysPermission> findMenuListByUserId(String userId) **throws** Exception;

service接口：根据用户id查询用户权限的菜单



#### 获取用户权限范围的url

思路：

在用户认证时，认证通过，根据用户id从数据库获取用户权限范围的url，将url的集合存储在session中。

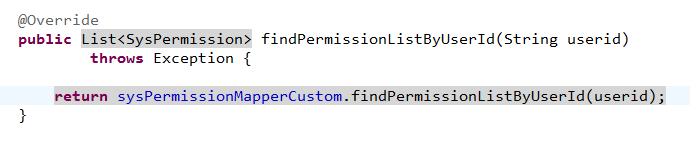
**public class** ActiveUser **implements** Serializable {  
  
 **private** Long **userId**;*//用户id* **private** String **userCode**;*// 用户账号* **private** String **username**;*// 用户名称* **private** List<SysPermission> **menus**;*// 菜单* **private** List<SysPermission> **permissions**;*// 权限*

mapper接口：根据用户id查询用户权限的url

*<!-- 根据用户id查询url -->* <**select id="findPermissionListByUserId" parameterType="string" resultType="SysPermission"**>  
 SELECT *\** FROM sys\_permission  
 WHERE TYPE = 'permission'  
 AND id IN (  
 SELECT sys\_permission\_id  
 FROM sys\_role\_permission  
 WHERE sys\_role\_id IN (  
 SELECT sys\_role\_id  
 FROM sys\_user\_role  
 WHERE sys\_user\_id = #{id}  
 )  
 )  
</**select**>

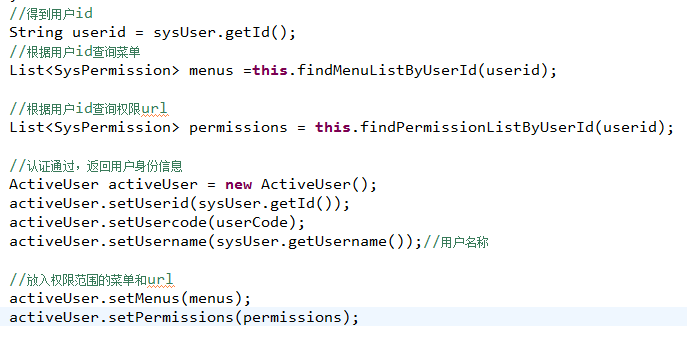
*/\*\*  
 \* 根据用户id查询菜单  
 \*  
 \** ***@param userId*** *用户id  
 \** ***@return*** *List*<*SysPermission*>  
 *\** ***@throws*** *Exception Exception  
 \*/*List<SysPermission> findMenuListByUserId(String userId) **throws** Exception;

service接口：根据用户id查询用户权限的url



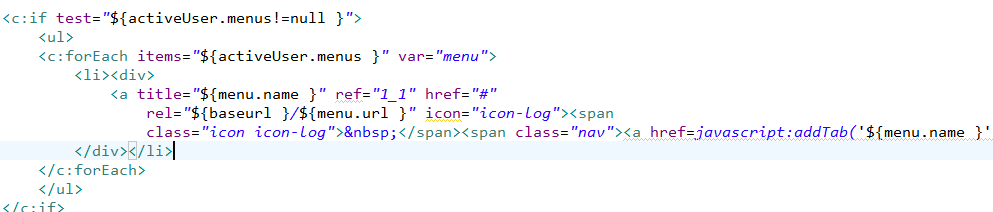
#### 用户认证通过取出菜单和url放入session

修改service认证代码：



#### 菜单动态显示

修改first.jsp，动态从session中取出菜单显示：



#### 授权拦截器

//在执行handler之前来执行的

//用于用户认证校验、用户权限校验

@Override

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

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

//得到请求的url

String url = request.getRequestURI();

//判断是否是公开 地址

//实际开发中需要公开 地址配置在配置文件中

//从配置中取逆名访问url

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

//遍历公开 地址，如果是公开 地址则放行

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

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

//如果是公开 地址则放行

**return** **true**;

}

}

//从配置文件中获取公共访问地址

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

//遍历公用 地址，如果是公用 地址则放行

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

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

//如果是公开 地址则放行

**return** **true**;

}

}

//获取session

HttpSession session = request.getSession();

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

//从session中取权限范围的url

List<SysPermission> permissions = activeUser.getPermissions();

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

//权限的url

String permission\_url = sysPermission.getUrl();

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

//如果是权限的url 地址则放行

**return** **true**;

}

}

//执行到这里拦截，跳转到无权访问的提示页面

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

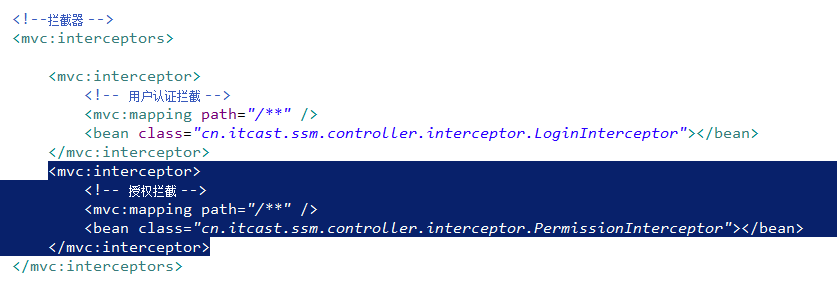
//如果返回false表示拦截不继续执行handler，如果返回true表示放行

**return** **false**;

}

#### 配置授权拦截器

注意：将授权拦截器配置在用户认证拦截的下边。



## 小结

使用基于url拦截的权限管理方式，实现起来比较简单，不依赖框架，使用web提供filter就可以实现。

问题：

需要将所有的url全部配置起来，有些繁琐，不易维护，url(资源)和权限表示方式不规范。

# shiro介绍

## 什么是shiro

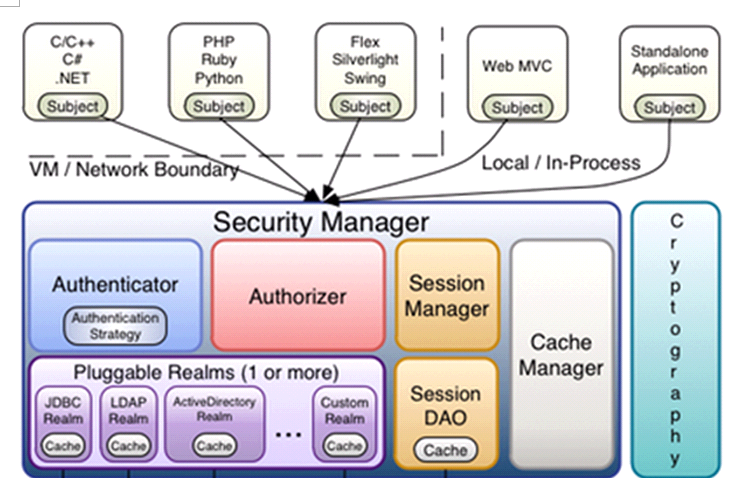
shiro是apache的一个开源框架，是一个权限管理的框架，实现 用户认证、用户授权。

spring中有spring security (原名Acegi)，是一个权限框架，它和spring依赖过于紧密，没有shiro使用简单。

shiro不依赖于spring，shiro不仅可以实现 web应用的权限管理，还可以实现c/s系统，分布式系统权限管理，shiro属于轻量框架，越来越多企业项目开始使用shiro。

使用shiro实现系统的权限管理，有效提高开发效率，从而降低开发成本。

## shiro架构



subject：主体，可以是用户也可以是程序，主体要访问系统，系统需要对主体进行认证、授权。

securityManager：安全管理器，主体进行认证和授权都 是通过securityManager进行。

authenticator：认证器，主体进行认证最终通过authenticator进行的。

authorizer：授权器，主体进行授权最终通过authorizer进行的。

sessionManager：web应用中一般是用web容器对session进行管理，shiro也提供一套session管理的方式。

SessionDao：通过SessionDao管理session数据，针对个性化的session数据存储需要使用sessionDao。

cache Manager：缓存管理器，主要对session和授权数据进行缓存，比如将授权数据通过cacheManager进行缓存管理，和ehcache整合对缓存数据进行管理。

realm：域，领域，相当于数据源，通过realm存取认证、授权相关数据。

**注意：在realm中存储授权和认证的逻辑。**

cryptography：密码管理，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。

比如 md5散列算法。

## 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。

*<!--核心包-->*<**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-core</**artifactId**>  
 <**version**>${shiro.version}</**version**>  
</**dependency**>  
  
*<!--整合web-->*<**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-web</**artifactId**>  
 <**version**>${shiro.version}</**version**>  
</**dependency**>  
  
*<!--整合Spring-->*<**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-spring</**artifactId**>  
 <**version**>${shiro.version}</**version**>  
</**dependency**>  
  
*<!--整合缓存-->*<**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-ehcache</**artifactId**>  
 <**version**>${shiro.version}</**version**>  
</**dependency**>  
  
*<!--整合任务调度-->*<**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-quartz</**artifactId**>  
 <**version**>${shiro.version}</**version**>  
</**dependency**>

也可以通过引入shiro-all包括shiro所有的包：

<dependency>

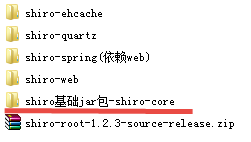
<groupId>org.apache.shiro</groupId>

<artifactId>shiro-all</artifactId>

<version>1.2.3</version>

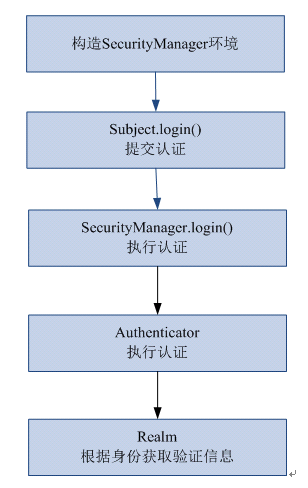
</dependency>

参考lib目录 ：



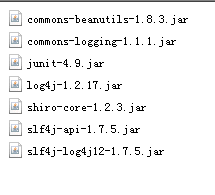
# shiro认证

## shiro认证流程

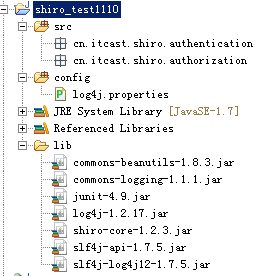


## shiro入门程序工程 环境

jar包：shiro-core.jar



工程结构：

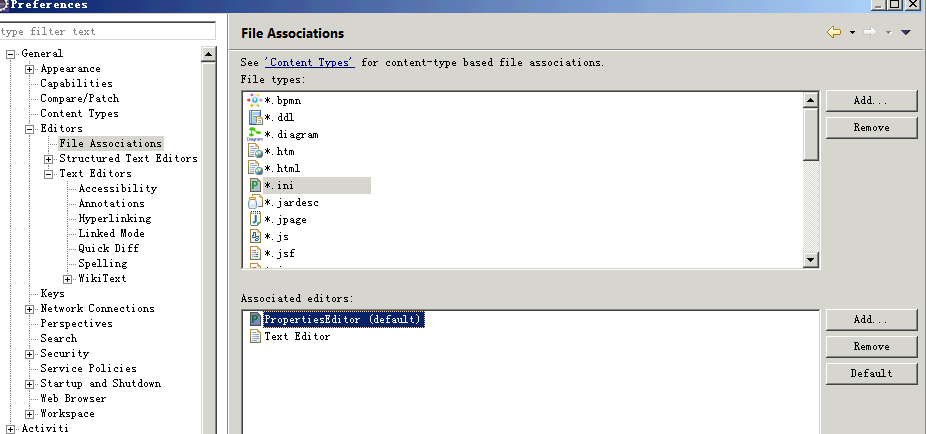


## shiro认证入门程序

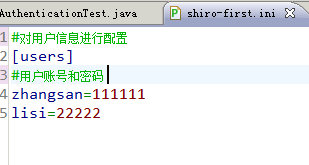
### shiro-first.ini

通过此配置文件创建securityManager工厂。

需要修改eclipse的ini的编辑器:



配置数据：



### 入门程序代码

*/\*\*  
 \* 用户登陆和退出  
 \*/*@Test  
**public void** testLoginAndLogout() {  
 *// 创建securityManager工厂，通过ini配置文件创建securityManager工厂* Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(**"classpath:shiro-first.ini"**);  
  
 *// 创建SecurityManager* SecurityManager securityManager = factory.getInstance();  
  
 *// 将securityManager设置当前的运行环境中* SecurityUtils.*setSecurityManager*(securityManager);  
  
 *// 从SecurityUtils里边创建一个subject* Subject subject = SecurityUtils.*getSubject*();  
  
 *// 在认证提交前准备token（令牌）  
 // 这里的账号和密码 将来是由用户输入进去* UsernamePasswordToken token = **new** UsernamePasswordToken(**"zhangsan"**, **"111111"**);  
  
 **try** {  
 *// 执行认证提交* subject.login(token);  
 } **catch** (AuthenticationException e) {  
 e.printStackTrace();  
 }  
  
 *// 是否认证通过* **boolean** isAuthenticated = subject.isAuthenticated();  
  
 System.***out***.println(**"是否认证通过："** + isAuthenticated);  
  
 *// 退出操作* subject.logout();  
  
 *// 是否认证通过* isAuthenticated = subject.isAuthenticated();  
  
 System.***out***.println(**"是否认证通过："** + isAuthenticated);  
}

### 执行流程

1、通过ini配置文件创建securityManager

2、调用subject.login方法主体提交认证，提交的token

3、securityManager进行认证，securityManager最终由ModularRealmAuthenticator进行认证。

4、ModularRealmAuthenticator调用IniRealm(给realm传入token)去ini配置文件中查询用户信息

5、IniRealm根据输入的token（UsernamePasswordToken）从shiro-first.ini查询用户信息，根据账号查询用户信息（账号和密码）

如果查询到用户信息，就给ModularRealmAuthenticator返回用户信息（账号和密码）

如果查询不到，就给ModularRealmAuthenticator返回null

6、ModularRealmAuthenticator接收IniRealm返回Authentication认证信息

如果返回的认证信息是null，ModularRealmAuthenticator抛出异常（org.apache.shiro.authc.UnknownAccountException）

如果返回的认证信息不是null（说明inirealm找到了用户），对IniRealm返回用户密码（在ini文件中存在）和 token中的密码进行对比，如果不一致抛出异常（org.apache.shiro.authc.IncorrectCredentialsException）

### 小结：

ModularRealmAuthenticator作用进行认证，需要调用realm查询用户信息（在数据库中存在用户信息）

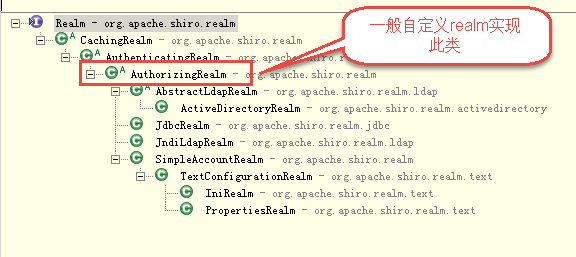
ModularRealmAuthenticator进**行密码对比**（认证过程）。

realm：需要根据token中的身份信息去查询数据库（入门程序使用ini配置文件），如果查到用户返回认证信息，如果查询不到返回null。

## 自定义realm

将来实际开发需要realm从数据库中查询用户信息。

### realm接口

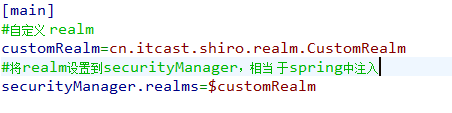


### 自定义realm

*// 用于认证* @Override  
 **protected** AuthenticationInfo doGetAuthenticationInfo(  
 AuthenticationToken token) **throws** AuthenticationException {  
  
 *// token是用户输入的  
 // 第一步从token中取出身份信息 账号* String userCode = (String) token.getPrincipal();  
  
 *// 第二步：根据用户输入的userCode从数据库查询  
 // ....  
  
  
 // 如果查询不到返回null  
 //数据库中用户账号是zhangsansan  
 /\*if(!userCode.equals("zhangsansan")){//  
 return null;  
}\*/  
  
  
 // 模拟从数据库查询到密码* String password = **"111111"**;  
  
 *// 如果查询到返回认证信息AuthenticationInfo* SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(  
 userCode, password, **this**.getName());  
  
 **return** simpleAuthenticationInfo;  
 }

### 配置realm

需要在shiro-realm.ini配置realm注入到securityManager中。



### 测试

同上边的入门程序，需要更改ini配置文件路径：

Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(**"classpath:shiro-realm.ini"**);

## 散列算法

通常需要对密码 进行散列，常用的有md5、sha，

对md5密码，如果知道散列后的值可以通过穷举算法，得到md5密码对应的明文。

建议对md5进行散列时加salt（盐），进行加密相当于对原始密码+盐进行散列。

正常使用时散列方法：

在程序中对原始密码+盐进行散列，将散列值存储到数据库中，并且还要将盐也要存储在数据库中。

如果进行密码对比时，使用相同方法，将原始密码+盐进行散列，进行比对。

### md5散列测试程序：

**public class** MD5Test {  
  
 **public static void** main(String[] args) {  
 *//原始 密码* String source = **"111111"**;  
 *//盐* String salt = **"qwerty"**;  
 *//散列次数* **int** hashIterations = 2;  
 *//上边散列1次：f3694f162729b7d0254c6e40260bf15c  
 //上边散列2次：36f2dfa24d0a9fa97276abbe13e596fc  
  
  
 //构造方法中：  
 //第一个参数：明文，原始密码  
 //第二个参数：盐，通过使用随机数  
 //第三个参数：散列的次数，比如散列两次，相当 于md5(md5(''))* Md5Hash md5Hash = **new** Md5Hash(source, salt, hashIterations);  
  
 String password\_md5 = md5Hash.toString();  
 System.***out***.println(password\_md5);  
  
 *//第一个参数：散列算法* SimpleHash simpleHash = **new** SimpleHash(**"md5"**, source, salt, hashIterations);  
 System.***out***.println(simpleHash.toString());  
 }  
  
}

### 自定义realm支持散列算法

需求：实际开发时realm要进行md5值（明文散列后的值）的对比。

#### 新建realm(CustomRealmMd5)

*// 用于认证* @Override  
 **protected** AuthenticationInfo doGetAuthenticationInfo(  
 AuthenticationToken token) **throws** AuthenticationException {  
  
 *// token是用户输入的  
 // 第一步从token中取出身份信息* String userCode = (String) token.getPrincipal();  
  
 *// 第二步：根据用户输入的userCode从数据库查询  
 // ....  
  
 // 如果查询不到返回null  
 // 数据库中用户账号是zhangsansan  
 /\*  
\* if(!userCode.equals("zhangsansan")){// return null; }  
\*/  
  
 // 模拟从数据库查询到密码,散列值* String password = **"f3694f162729b7d0254c6e40260bf15c"**;  
 *// 从数据库获取salt* String salt = **"qwerty"**;  
 *//上边散列值和盐对应的明文：111111  
  
 // 如果查询到返回认证信息AuthenticationInfo* SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(  
 userCode, password, ByteSource.Util.*bytes*(salt), **this**.getName());  
  
 **return** simpleAuthenticationInfo;  
 }

#### 在realm中配置凭证匹配器

[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.qdgxy.shiro.realm.CustomRealmMd5  
customRealm.credentialsMatcher=$credentialsMatcher  
securityManager.realms=$customRealm