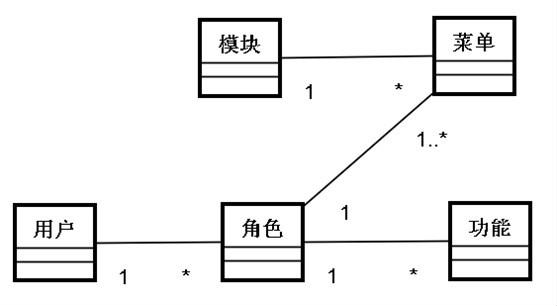
**在项目中集成shiro权限框架(1)**

Shiro是一个功能强大的轻量级权限框架，相对其它权限框架（比如spring security）来说，要易用得很，下面，我给大家讲讲如何在一个项目中简单整合shiro。

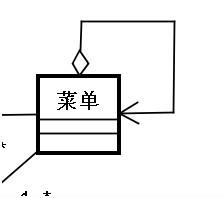
我们通常所说的权限，就是要判断某个操作者是否有操作某个资源的权限，而资源，可以是菜单、链接、功能按钮、业务方法、某类型的数据等等，根据需求，每个项目的权限可能都有所不同，通用万能权限系统是不存在的。

Shrio的权限包括认证、授权、密码管理、会话管理等四个部分，每一个部分，shrio都进行了抽象，以做到和具体应用平台、环境的无关。关于shiro的基础了解，大家可到这里去看看：[http://www.infoq.com/cn/articles/apache-shiro](http://www.infoq.com/cn/articles/apache-shiro" \t "_blank)

权限系统的设计，根据不同的权限粒度，会用所不同，比如：有的系统到菜单级，而有的到每一个功能按钮级，有的到方法级，有的到业务数据级，我在这仅仅讨论前两类。



上面是一个对象关系图，模块在此处仅仅为菜单进行逻辑归类，当然，也可以建立菜单和功能的一对多关系，为功能进行逻辑归类。当然，菜单也可以建成关联的树状结构，以更好扩展。

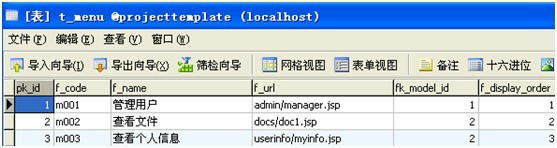


根据对象关系，可以设计出表结构，我直接贴出数据截图：

模块表：



菜单表：



操作表，一般对应页面中的按钮，其中f\_action\_flag对应于restful中的增、删、查、改：



角色表：



角色-菜单关系表：



角色-功能表：



用户表：



用户-角色关系表：



上面截图中，真正有效的表7张，权限控制到页面及功能上。

下篇，贴出上面的类代码及class映射文件。

**在项目中集成shiro权限框架（2）**

基础代码贴上：

对象如下：

/\*\*

 \* 用户对象

 \*

 \* **@author**Ljh

 \*

 \*/

**public** **class**User **implements**Serializable {

    // 学生编码(由系统自动生成)

**private**Long id;

    // 姓名

    @NotEmpty(message = "姓名不可以为空！")

    @Length(min = 2, max = 5)

**private**String name;

    // 出生日期

    @NotNull(message = "生日不能为空")

    @Past(message = "生日输入不正确，请核对！")

**private**Date birthday;

    // 登录密码

**private**String password= "111";

    // 用户角色集合字符

**private**Set<Role> roles= **new**HashSet<Role>();

    /\*\*

     \* 得到该用户可访问的模块，模块中的菜单将实例化

     \* **@return**

     \*/

**public**List<Model> getModels() {

       List<Model> models = **new**ArrayList<Model>();

**for**(Iterator<Menu> its = **this**.getMenus().iterator(); its.hasNext();){

           Menu m = its.next();

**if**(!models.contains(m.getModel())) {

              Model model = **new**Model();

              model.setId(m.getModel().getId());

              model.setDisplayOrder(m.getDisplayOrder());

              model.setModelName(m.getModel().getModelName());

              model.getMenus().add(m);

              models.add(model);

           } **else**{

**for**(Model model : models) {

**if**(model.getId() == m.getModel().getId()) {

                     model.getMenus().add(m);

**break**;

                  }

              }

           }

       }

       Collections.*sort*(models);

**return**models;

    }

    /\*\*

     \* 得到用户可访问的菜单资源

     \* **@return**

     \*/

**public**Set<Menu> getMenus() {

       Set<Menu> menus = **new**HashSet<Menu>();

**for**(Iterator<Role> role = **this**.getRoles().iterator(); role.hasNext();) {

           menus.addAll(role.next().getMenus());

       }

**return**menus;

    }

    // 用户菜操作功能权限集合字符串描述

**public**Set<String> getOperationPermissionsAsString() {

       Set<String> pomissions = **new**HashSet<String>();

       Iterator<Role> roles = **this**.getRoles().iterator();

       Operation op;

       Map<String, HashSet<String>> p\_map = **new**HashMap<String,HashSet<String>>();

**while**(roles.hasNext()) {

           Iterator<Operation> operations =roles.next().getOperations().iterator();

**while**(operations.hasNext()) {

              op = operations.next();

              String key = op.getUrl();

**if**(!key.startsWith("/")) {

                  key = "/"+ key;

              }

**if**(p\_map.get(key) == **null**) {

                  p\_map.put(key, **new**HashSet<String>());

              }

              p\_map.get(key).add(op.getActionFlag());

           }

       }

       //构建形如：[doc:read, moveuser:modify, users:read,user:modify,read,create]的权限字串

**for**(Entry<String, HashSet<String> > entry :p\_map.entrySet()) {

           pomissions.add(entry.getKey() + ":"+ entry.getValue().toString().replace("[", "").replace("]", "").replace(" ", ""));

       }

**return**pomissions;

    }

    // 用户菜单权限集合字符串描述

//  public Set<String> getMenuPermissionsAsString() {

//     Set<String> pomissions= newHashSet<String>();

//     Iterator<Role> roles = this.getRoles().iterator();

//     while (roles.hasNext()) {

//         Iterator<Menu> menus =roles.next().getMenus().iterator();

//         while (menus.hasNext()) {

//            pomissions.add(menus.next().getCode());

//         }

//     }

//     return pomissions;

//  }

    /\*\*

     \* 得到我的全部权限

     \* **@return**

     \*/

**public**Set<String> getPermissionsAsString() {

       Set<String> permissions = **new**HashSet<String>();

       //permissions.addAll(getMenuPermissionsAsString());

       permissions.addAll(getOperationPermissionsAsString());

**return**  permissions;

    }

    // 得到用户角色字符串描述

**public**Set<String> getRolesAsString() {

       Set<String> str\_roles = **new**HashSet<String>();

       Iterator<Role> roles = **this**.getRoles().iterator();

**while**(roles.hasNext()) {

           str\_roles.add(roles.next().getRoleCode());

       }

**return**str\_roles;

    }

角色对象：

**public** **class**Role **implements**Serializable {

    //OID

**private** **int** id;

    //显示名称

**private**String displayName;

    //角色编码，用于生成权限框架的惟一标识

**private**String roleCode;

    //角色可以操作的菜单

**private**Set<Menu> menus= **new**HashSet<Menu>();

    //角色可操作的操作功能【对应于增、删、查、改等功能】

**private**Set<Operation> operations= **new**HashSet<Operation>();

菜单对象：

**public** **class**Menu **implements**Resource, Comparable, Serializable  {

    //OID

**private** **int** id;

**private**String code;

**private**String name;

    //显示顺序

**private** **int** displayOrder;

    //url地址

**private**String url;

    //所属模块

**private**Model model;

@Override

**public** **int**compareTo(Object menu) {

       Menu m = (Menu)menu;

**return**m.getDisplayOrder() - **this**.getDisplayOrder();

    }

操作功能对象：

**public** **class**Operation **implements**Resource, Serializable{

    //OID

**private** **int** id;

    //操作码（保留待用）

**private**String code;

    //名称

**private**String name;

    //操作标志read：读取，create:新增，modify：修改，delete：删除

    //在rest风格中，将请求方式映射为上述四种操作，设计为字符串，以方便今后扩展

**private**String actionFlag;

    //操作url地址,比如:/user/\*

**private**String url;

    //所属菜单

**private**Menu menu;

    //显示顺序，保留待用

**private** **int** displayOrder;

功能模块对象：

/\*\*

 \* 功能模块

 \* **@author**Administrator

 \*

 \*/

**public** **class**Model **implements**Comparable<Model>,Serializable {

//public class Model  {

**private** **int** id;

**private**String modelName;

    //显示顺序

**private** **int** displayOrder;

**private**Set<Menu> menus= **new**HashSet<Menu>();

资源抽象对象，目前暂未使用：

/\*\*

 \* 资源抽象类

 \* **@author**Administrator

 \*

 \*/

**public** **interface**Resource {

    //得到资源编码

**public**String getResCode();

}

最后，贴上hbm文件。

<classname=*"com.my.model.User"*table=*"t\_student"*>

        <idname=*"id"*>

            <columnname=*"pk\_id"*/>

            <generatorclass=*"identity"*></generator>

        </id>

        <propertyname=*"name"*type=*"java.lang.String"*>

            <columnname=*"f\_name"*/>

        </property>

        <propertyname=*"password"*type=*"java.lang.String"*>

            <columnname=*"f\_password"*/>

        </property>

        <propertyname=*"birthday"*>

            <columnname=*"f\_birthday"*/>

        </property>

        <setname=*"roles"*table=*"t\_student\_role"*>

         <keycolumn=*"fk\_student\_id"*></key>

         <many-to-manyclass=*"com.my.model.Role"*column=*"fk\_role\_id"*>

         </many-to-many>

        </set>

</class>

<classname=*"com.my.model.Model"*table=*"t\_model"*>

        <idname=*"id"*>

            <columnname=*"pk\_id"*/>

            <generatorclass=*"identity"*></generator>

        </id>

        <propertyname=*"modelName"*type=*"java.lang.String"*>

            <columnname=*"f\_model\_name"*/>

        </property>

        <propertyname=*"displayOrder"*column=*"f\_display\_order"*>

        </property>

        <setname=*"menus"*table=*"t\_menu"*order-by=*"f\_display\_order"*>

         <keycolumn=*"fk\_model\_id"*></key>

         <one-to-manyclass=*"com.my.model.Menu"*/>

        </set>

    </class>

    <classname=*"com.my.model.Menu"*table=*"t\_menu"*>

        <idname=*"id"*>

            <columnname=*"pk\_id"*/>

            <generatorclass=*"identity"*></generator>

        </id>

        <propertyname=*"code"*type=*"java.lang.String"*>

            <columnname=*"f\_code"*/>

        </property>

        <propertyname=*"name"*type=*"java.lang.String"*>

            <columnname=*"f\_name"*/>

        </property>

        <propertyname=*"displayOrder"*column=*"f\_display\_order"*>

        </property>

         <propertyname=*"url"*type=*"java.lang.String"*>

            <columnname=*"f\_url"*/>

        </property>

        <many-to-onename=*"model"*class=*"com.my.model.Model"*column=*"fk\_model\_id"*>

        </many-to-one>

    </class>

    <classname=*"com.my.model.Operation"*table=*"t\_operation"*>

        <idname=*"id"*>

            <columnname=*"pk\_id"*/>

            <generatorclass=*"identity"*></generator>

        </id>

        <propertyname=*"code"*type=*"java.lang.String"*>

            <columnname=*"f\_code"*/>

        </property>

        <propertyname=*"name"*type=*"java.lang.String"*>

            <columnname=*"f\_name"*/>

        </property>

        <propertyname=*"displayOrder"*column=*"f\_display\_order"*>

        </property>

        <propertyname=*"actionFlag"*column=*"f\_action\_flag"*>

        </property>

         <propertyname=*"url"*type=*"java.lang.String"*>

            <columnname=*"f\_url"*/>

        </property>

        <many-to-onename=*"menu"*class=*"com.my.model.Menu"*column=*"fk\_menu\_id"*>

        </many-to-one>

    </class>

    <classname=*"com.my.model.Role"*table=*"t\_role"*>

        <idname=*"id"*>

            <columnname=*"pk\_id"*/>

            <generatorclass=*"identity"*></generator>

        </id>

        <propertyname=*"displayName"*type=*"java.lang.String"*>

            <columnname=*"f\_displayName"*/>

        </property>

        <propertyname=*"roleCode"*type=*"java.lang.String"*>

            <columnname=*"f\_roleCode"*/>

        </property>

        <setname=*"menus"*table=*"t\_role\_menu"*>

         <keycolumn=*"fk\_role\_id"*></key>

         <many-to-manyclass=*"com.my.model.Menu"*column=*"fk\_menu\_id"*order-by=*"f\_display\_order"*>

         </many-to-many>

        </set>

         <setname=*"operations"*table=*"t\_role\_operation"*>

         <keycolumn=*"fk\_role\_id"*></key>

         <many-to-manyclass=*"com.my.model.Operation"*column=*"fk\_operation\_id"*order-by=*"f\_display\_order"*>

         </many-to-many>

        </set>

    </class>

基础的代码完成了，其实，user中已经有部分和权限验证相关的代码了，后面我详细讲解。

## 在项目中集成shiro权限框架（3）

Shiro与spring已经有完整的整合方法，所以，我们先在web.xml中添对过滤器，将需要验证的请求，拦截到shiro中。

<!—shiro过滤器-->

    <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>

在spring配置中添加下面的支持bean.

<!-- 权限or判定器-->

    <beanid="roleOrFilter"class="com.my.commons.RolesOrFilter">

    </bean>

    <!-- 认证数据库存储-->

    <beanid="myRealm"class="com.my.service.impl.DbAuthRealm">

    </bean>

    <!-- 权限管理器-->

    <beanid="securityManager"class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

       <propertyname="realms">

           <list>

              <refbean="myRealm"/>

           </list>

       </property>

    </bean>

    <beanid="shiroFilter"class="org.apache.shiro.spring.web.ShiroFilterFactoryBean"depends-on="roleOrFilter">

       <propertyname="securityManager"ref="securityManager"/>

       <propertyname="loginUrl"value="/login.jsp"/>

       <propertyname="successUrl"value="/main"/>

       <propertyname="unauthorizedUrl"value="/commons/unauth.jsp"/>

       <!-- 读取自定义权限内容-->

       <propertyname="filterChainDefinitions"value="#{authService.loadFilterChainDefinitions()}"/>

       <propertyname="filters">

           <map>

              <entrykey="roleOrFilter"value-ref="roleOrFilter">

              </entry>

           </map>

       </property>

    </bean>

    <beanid="lifecycleBeanPostProcessor"class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>

其中filterChainDefinitions部分，简单的项目可以直接写死，我们这里采用的是启动时读取生成，如果写死，将会出现类似下面的代码：

<property name="filterChainDefinitions">

           <value>

              /js/\*\* = anon

              /css/\*\* = anon

              /login.jsp = anon

              /login = anon

              /docs/doc1.jsp= authc,roleOrFilter[admin,dev]

              /admin/manager.jsp = authc, roleOrFilter[admin]

              /admin/user/\* = authc, rest[/admin/user/\*]

              /userinfo/myinfo.jsp = authc,roleOrFilter[test,dev,admin]

              /\*\* = authc

           </value>

       </property>

这段静态权限代码中用到了过滤器，我得给大家说一下：

anon部分是不需要进行验证的，即这部分是要放行的（注意\*\*与\*是有区别的，\*\*表示该目录下的所有，当然也包括子目录下的东西，而\*仅指当前目录下的东西）。

Authc,这个是需要登录的人才可以访问的。

roleOrFile，这个是我定义的，只要这个过滤器中的一个角色满足，即可访问，而shiro提供的role过滤器，是当所有角色都满足时才可访问。

Rest是restful过滤器，会将get,post,put,delete转换为对资源的read.create,modify,delete操作。

当然，shiro还提供了其它的过滤器，大家可以自己去看看，比如：permission过滤器，shiro对权限的描述采用wildcard字串，功能强大且可读性强。

另外一点要接出，shiro在里使用最先匹配规则，一旦匹配成功，将不再进行后续的过滤规则检查，因此，在书写时一定要注意顺序，比如，你把/\*\* = anon写到第一行，那么后面的一切都将不会再检测。

好了，下面来讲如何动态生成这个规则，大家看我的配置中有value="#{authService.loadFilterChainDefinitions()}"，这个是spring el表达式语言，表示调用容器中的另一个bean的方法，把这个方法的返回结果，赋值给filterChainDefinitions属性。

在些再补充一些权限字串的知识：

shiro所支持的广义权限字串表达式，共有三种：

1、简单方式

比如：subject.isPermitted("editNews")，表示判断某操作者是否有【编辑新闻】的权限。

2、细粒度方式

比如：subject.isPermitted("News:create")，表示判断某操作者是否有【新建新闻】的权限。

3、实例级访问方式

比如：subject.isPermitted("News:edit:10")，表示判断某操作者是否有【编辑id号是10新闻】的权限。

上面3种方式中，可以用\*表示所有，例如："News:\*"为对所有新闻的操作，"\*:create"对所有事务都可以新增。还可以用 逗号 表示或都，"News:edit:10,11"表示可对10，11号新闻进行编辑。

如果要写页面权限，可参照如下配置：

/index.jsp = anon  
/admin/\*\* = authc, roles[admin]  
 /docs/\*\* = authc, perms[document:read]  
 /\*\* = authc

我定义了一个权限相关的接口，如下：

/\*\*

 \* 权限管理相关方法

 \* **@author**ljh

 \*

 \*/

**public** **interface**IAuthService{

    /\*\*

     \* 加载过滤配置信息

     \* **@return**

     \*/

**public**String loadFilterChainDefinitions();

    /\*\*

     \* 重新构建权限过滤器

     \* 一般在修改了用户角色、用户等信息时，需要再次调用该方法

     \*/

**public** **void**reCreateFilterChains();

}

其中一个方法用于加载生成权限规则字串，另一个，用于用户在系统中更改了角色-菜单，角色-功能关系时，动态重新生效的方法，实现类如下:

@Service(value="authService")

**public** **class**AuthServiceImpl**implements**IAuthService {

**private** **static** **final**Logger log= Logger.getLogger(AuthServiceImpl.**class**);

    //注意/r/n前不能有空格

**private** **static** **final**String CRLF= "\r\n";

**private** **static** **final**String LAST\_AUTH\_STR= "/\*\* =authc\r\n";

    @Resource

**private**ShiroFilterFactoryBean shiroFilterFactoryBean;

    @Resource

**private**IBaseDao dao;

    @Override

**public**String loadFilterChainDefinitions() {

       StringBuffer sb = **new**StringBuffer("");

       sb.append(getFixedAuthRule())

       .append(getDynaAuthRule())

       .append(getRestfulOperationAuthRule())

       .append(LAST\_AUTH\_STR);

**return**sb.toString();

    }

    //生成restful风格功能权限规则

**private**String getRestfulOperationAuthRule() {

       List<Operation> operations = dao.queryEntitys("from Operation o", **new**Object[]{});

       Set<String> restfulUrls = **new**HashSet<String>();

**for**(Operation op : operations) {

           restfulUrls.add(op.getUrl());

       }

       StringBuffer sb  = **new**StringBuffer("");

**for**(Iterator<String> urls =  restfulUrls.iterator(); urls.hasNext(); ) {

           String url = urls.next();

**if**(! url.startsWith("/")) {

              url = "/"+ url ;

           }

           sb.append(url).append("=").append("authc, rest[").append(url).append("]").append(CRLF);

       }

**return**sb.toString();

    }

    //根据角色，得到动态权限规则

**private**String getDynaAuthRule() {

       StringBuffer sb = **new**StringBuffer("");

       Map<String, Set<String>> rules = **new**HashMap<String,Set<String>>();

       List<Role> roles = dao.queryEntitys("from Role r left join fetch r.menus", **new**Object[]{});

**for**(Role role: roles) {

**for**(Iterator<Menu> menus =role.getMenus().iterator(); menus.hasNext();) {

              String url = menus.next().getUrl();

**if**(!url.startsWith("/")) {

                  url = "/"+ url;

              }

**if**(!rules.containsKey(url)) {

                  rules.put(url, **new**HashSet<String>());

              }

              rules.get(url).add((role.getRoleCode()));

           }

       }

**for**(Map.Entry<String, Set<String>> entry :rules.entrySet()) {

           sb.append(entry.getKey()).append(" = ").append("authc,roleOrFilter").append(entry.getValue()).append(CRLF);

       }

**return**sb.toString();

    }

    //得到固定权限验证规则串

**private**String getFixedAuthRule() {

       StringBuffer sb = **new**StringBuffer("");

       ClassPathResource cp = **new**ClassPathResource("fixed\_auth\_res.properties");

       Properties properties = **new**OrderedProperties();

**try**{

           properties.load(cp.getInputStream());

       } **catch**(IOException e) {

           log.error("loadfixed\_auth\_res.properties error!", e);

**throw** **new**RuntimeException("load fixed\_auth\_res.properties error!");

       }

**for**(Iteratorits = properties.keySet().iterator();its.hasNext();) {

           String key = (String)its.next();

           sb.append(key).append(" = ").append(properties.getProperty(key).trim()).append(CRLF);

       }

**return**sb.toString();

    }

    @Override

    //此方法加同步锁

**public** **synchronized** **void**reCreateFilterChains() {

       AbstractShiroFilter shiroFilter = **null**;

**try**{

           shiroFilter = (AbstractShiroFilter)shiroFilterFactoryBean.getObject();

       } **catch**(Exception e) {

           log.error("getShiroFilter from shiroFilterFactoryBean error!", e);

**throw** **new**RuntimeException("get ShiroFilter from shiroFilterFactoryBean error!");

       }

       PathMatchingFilterChainResolver filterChainResolver =(PathMatchingFilterChainResolver)shiroFilter.getFilterChainResolver();

       DefaultFilterChainManager manager =(DefaultFilterChainManager)filterChainResolver.getFilterChainManager();

       //清空老的权限控制

       manager.getFilterChains().clear();

       shiroFilterFactoryBean.getFilterChainDefinitionMap().clear();

       shiroFilterFactoryBean.setFilterChainDefinitions(loadFilterChainDefinitions());

       //重新构建生成

       Map<String, String> chains = shiroFilterFactoryBean.getFilterChainDefinitionMap();

**for**(Map.Entry<String, String> entry :chains.entrySet()) {

            String url = entry.getKey();

            String chainDefinition =entry.getValue().trim().replace(" ", "");

            manager.createChain(url,chainDefinition);

        }

    }

}

## 在项目中集成shiro权限框架（4）

在实现loadFilterChainDefinitions方法时，我把权限规则分成了三个部分，一个是固定规则，比如哪些不需要过渡等，另一部分是菜单级动态规则，最后一部分是基于restful的功能规则（当然，如果系统不是基于restful的，需要根据实际情况修改）。

固定规则保存在一个叫fixed\_auth\_res.properties的文件中，内容大致是这样的：

#fixed authrules

/js/\*\*= anon

/css/\*\*= anon

/login.jsp = anon

/login = anon

下面是DbAuthRealm类，按照我们的规则，从数据库中取出信息，对用户进行验证，密码的加密算法也是在些检测。

/\*\*

 \* 数据库存储认证类信息

 \* **@author**ljh

 \*

 \*/

**public** **class**DbAuthRealm **extends**AuthorizingRealm {

    @Resource(name = "studentService")

**private**IStudentService studentService;

**public**DbAuthRealm() {

**super**();

       // 设置认证token的实现类为用户名密码模式

**this**.setAuthenticationTokenClass(UsernamePasswordToken.**class**);

       //设置验证方式，用户自行设定密码加密方式

**this**.setCredentialsMatcher(**new**CredentialsMatcher() {

           @Override

**public** **boolean**doCredentialsMatch(AuthenticationToken token,AuthenticationInfo info) {

              UsernamePasswordToken upToken = (UsernamePasswordToken)token;

              String pwd = **new**String(upToken.getPassword());

              User student = studentService.getStudentById(Long.parseLong(upToken.getUsername()));

**if**(student.getPassword().equals(DigestUtils.md5Hex(pwd))){

                  //用户名及密码验证通过

**return** **true**;

              }

              //用户名或密码不正确

**return** **false**;

           }

       });

    }

    // 认证

    @Override

**protected**AuthenticationInfo doGetAuthenticationInfo(

           AuthenticationToken token) **throws**AuthenticationException {

       UsernamePasswordToken upToken = (UsernamePasswordToken) token;

       //调用业务方法

       User student = studentService.getStudentById(Long.parseLong(upToken.getUsername()));

**if**(student != **null**) {

           //要放在作用域中的东西，请在这里进行操作

           SecurityUtils.getSubject().getSession().setAttribute("c\_user", student);

**return** **new**SimpleAuthenticationInfo(student.getId(),student.getPassword(), **this**.getName());

       }

       //认证没有通过

**return** **null**;

    }

    // 授权

    @Override

**protected**AuthorizationInfo doGetAuthorizationInfo(

           PrincipalCollection principalCollection) {

       Long loginId = (Long)principalCollection.fromRealm(getName()).iterator().next();

       //取当前用户

       User student = studentService.getStudentById(loginId);

       //添加角色及权限信息

       SimpleAuthorizationInfo sazi = **new**SimpleAuthorizationInfo();

       sazi.addRoles(student.getRolesAsString());

       sazi.addStringPermissions(student.getPermissionsAsString());

**return**sazi;

    }

    @Override

**protected** **void**clearCachedAuthorizationInfo(PrincipalCollection pc) {

       SimplePrincipalCollection principals= **new**SimplePrincipalCollection(pc, getName());

**super**.clearCachedAuthorizationInfo(pc);

    }

最后，来看看入口控制器关于登录、退出的代码：

@Controller

**public** **class**MyAction {

    @Resource

**private**ShiroFilterFactoryBean shiroFilterFactoryBean;

    @Resource

**private**IAuthService authService;

    @RequestMapping(value="/login")

**public**String login(

           @RequestParam(value="username", defaultValue="") String username,

           @RequestParam(value="password", defaultValue="") String password,

           HttpServletRequest req) {

       Subject currentUser = SecurityUtils.getSubject();

**boolean**loginSuccess = **false**;

       //用户已经登录过了，直接进主页面

**if**(currentUser.isAuthenticated()) {

           loginSuccess = **true**;

       }

**try**{

           currentUser.login(**new**UsernamePasswordToken(username, password));

           loginSuccess = **true**;

       } **catch**(Exception e) {

           e.printStackTrace();

           req.setAttribute("error\_info", "用户名或密码错，请核对！");

**return** "login";

       }

**if**(loginSuccess) {

           //生成用户菜单(当然，也可以在main中完成，但在重定向的情况下会有hibernatesession延迟问题)

           User s = (User)currentUser.getSession().getAttribute("c\_user");

           //currentUser.getSession().setAttribute("menus",s.getMenus());

           currentUser.getSession().setAttribute("models", s.getModels());

**return** "redirect:main";

       }

**return** "login";

    }

    @RequestMapping(value="/logout")

**public**String logout() {

       Subject currentUser = SecurityUtils.getSubject();

       currentUser.logout();

**return** "login";

    }

    @RequestMapping(value="/main")

**public**String index() {

       User s = (User)SecurityUtils.getSubject().getSession().getAttribute("c\_user");

       System.out.println(s.getName() + "进入主页面！");

**return** "main\_page";

    }

最后，RolesOrFilter代码贴出：

/\*\*

 \* 在进行角色检测时，只要有一个满足即通

 \* **@author**ljh

 \*

 \*/

**public** **class**RolesOrFilter **extends**AuthorizationFilter {

    @SuppressWarnings({"unchecked"})

**public** **boolean**isAccessAllowed(ServletRequestrequest, ServletResponse response, Object mappedValue) **throws**IOException {

        Subject subject = getSubject(request,response);

        String[] rolesArray = (String[])mappedValue;

**if**(rolesArray == **null**|| rolesArray.length== 0) {

            //no roles specified, so nothing to check - allow access.

**return** **true**;

        }

**for**(String role : rolesArray) {

**if**(subject.hasRole(role)) {

**return** **true**;

         }

        }

**return** **false**;

    }

}

最后一个辅助类，在读取properties时，按key/val在源文件中的顺序读出：

/\*\*

 \* 有序properties

 \* **@author**ljh

 \*/

**public** **class**OrderedProperties **extends**Properties {

**private** **static** **final** **long** serialVersionUID= -4827607240846121968L;

**private** **final**LinkedHashSet<Object> keys= **new**LinkedHashSet<Object>();

**public**Enumeration<Object>keys() {

**return**Collections.<Object> enumeration(keys);

    }

**public**Object put(Object key, Objectvalue) {

        keys.add(key);

**return** **super**.put(key, value);

    }

**public**Set<Object> keySet() {

**return** keys;

    }

**public**Set<String>stringPropertyNames() {

        Set<String> set = **new**LinkedHashSet<String>();

**for**(Object key : **this**.keys) {

            set.add((String) key);

        }

**return**set;

    }

}

好了，shiro的简单的权限整合到此结束，上面的代码实现了用户功能菜单根据权限动态生成，基于url的访问控制，基于restful的访问控制，权限粒度到了功能按钮级别。

另外，如果你要实现基于方法级的权限，shiro也是支持的，它提供了相应的注解【当然你也可以用AOP来自己写，需要用到threadlocal对象传递一些信息哦】。如果你想实现基于数据级的权限，我想最好的办法还是自己在业务实现中进行处理吧。

http://hi.baidu.com/ien\_leo/item/4716ea59e132a0cf9e2667c4