**Spring Security系统安全管理**

**----编写人csx**

目录

[1. Spring Security安全框架介绍 2](#_Toc303681900)

[1.1 Spring Security核心概念 2](#_Toc303681901)

[1.1.1 Authentication (认证）对象 2](#_Toc303681902)

[1.1.2 SecurityContextHolder 3](#_Toc303681903)

[1.1.3 AuthenticationManager 3](#_Toc303681904)

[1.1.4 AccessDecissionManager 3](#_Toc303681905)

[1.1.5 Interceptors(拦截器) 3](#_Toc303681906)

[1.1.6 GrantedAuthority(权限) 4](#_Toc303681907)

[1.2 Spring Security安全管理的原理 4](#_Toc303681908)

[1.2.1 Spring Security对一个请求的认证过程 4](#_Toc303681909)

[1.2.2 用户的认证过程 5](#_Toc303681910)

[2. 系统的安全管理方式 7](#_Toc303681911)

[3. 整合Spring Security 8](#_Toc303681912)

[3.1 表设计 8](#_Toc303681915)

[3.2 与Spring Security集成说明 9](#_Toc303681916)

[3.2.1 身份认证 9](#_Toc303681917)

[*3.2.2* 访问授权 12](#_Toc303681924)

[3.2.3 权限的配置及使用 14](#_Toc303681925)

# Spring Security安全框架介绍

Spring Security（以下简称为SS）其前身是大名鼎鼎的ACEGI，后来正式合并至Spring框架，成为Spring框架中一员，由于它支持安全策略比较多，并且集成容易，现在也越来越多的系统采用它来进行安全管理。本系统中也采用它的基于角色的管理策略，以实现系统灵活的安全管理要求。

## Spring Security核心概念

### Authentication (认证）对象

其实就是一个可以通过Spring Security的认证的身份证明。如实现该接口的类UsernamePasswordAuthenticationToken， 表示可以通过username及password作为身份验证。

Authentication对象包含了

 Principal 标识是哪一个对象，可以认为是用户

 Credentials 信任的对象，如密码

 Authorities 权限的集合，在我们的系统中可以认为是角色的集合 (authorities要赋予给principal的)

### SecurityContextHolder

是Spring Security的核心对象，是安全上下文的访问的入口。如取得当前的登录用户可以从该类中的相应的方法取得。该类中包含ThreadLocal私有属性用于存取SecurityContext, SecurityContext包含Authentication私有属性。如实现弹出窗口登录功能的时候，输入的用户名及密码并没有最终经过SPRING SECURITY的filter,那么如何使用得当前用户可以成功登录呢，其就是利用到这一点，通过该类拿到SecurityContext，然后设置一个认证的对象给它，SPRING SECURITY在看到该认证对象的时候，就会成功经过了身份的认证了。

其实也可以这样理解，为了处理Http请求间认证，Spring Security使用HttpSessionIntegrationFilter，HttpSessionIntegrationFilter用于在HttpSession存储Http请求间的SecurityContext。不过我们可以通过SecurityContextHolder去拿到这个SecurityContext

### AuthenticationManager

通过Providers 验证 在当前 ContextHolder中的Authentication对象是否合法。

### AccessDecissionManager

经过投票机制来审批是否批准操作

### Interceptors(拦截器)

拦截器(如FilterSecurityInterceptor，JoinPoint，MethodSecurityInterceptor等)用于协调授权，认证等操作。

### GrantedAuthority(权限)

是授予相应Principal的应用系统访问权限。

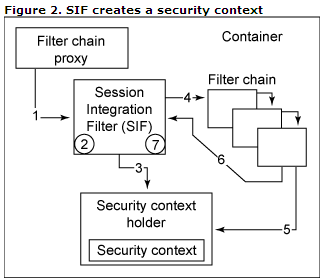
Spring Security是Spring中一个强大的安全管理框架，不过目前在我们系统中使用的仅是其中一部分的功能，则权限过滤安全检查的功能。

【说明】

*如果抛开这个框架，我们实现权限管理的时候，可能使用最多的方案还是使用Filter来进行过滤，在Filter里判断当前的用户是否为登录用户，若是登录用户，则看是否有权限访问当前的资源，若为未登录用户，则跳至登录页面。*

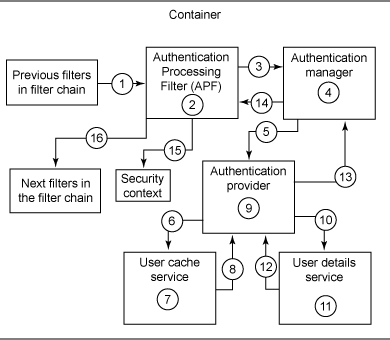
## Spring Security安全管理的原理

### Spring Security对一个请求的认证过程



1. Spring Security Filter Chain 代理触发SIF，并且传入request,response,filter chain对象。在这里一般是认为SIF是你在一串filter中配置的第一个filter。
2. SIF检查是否已经传进去Web Request对象了，如果发现其已经传了，则在同一串中的后面的filter不再会被调用.（看以下的第四步）, 如果SIF发现这个当前的Web 请求是第一次被访问，则会设置一个标识，告诉SIF，该请求已经是被请求过了。
3. SIF检查当前的session中是否已经包括了security context, 它从session中拿到该session context，并且放置到一个叫Security Context Holder的地方去（其实是一跟一个线程绑定）.如果session中不包含该对象，则SIF则创建一个Security Context对象并且把它放置到Security Context Hodler中去。由于Security Context Holder是一个application 的对象，所以其他filter是可以访问的。
4. SIF调用这个Chain中的下一个filter。
5. 其他的filter可有会更新Security Context。
6. 当所有的filter都处理完成后，就会把控制权交还给SIF
7. SIF检查其他的filter是否在处理的过程中改变了Security Context对象（如APF可能会在Security Context中存储了User Details），如果改变了，则就会更新Session Context中的session对象。这意味着你在所有的filter处理中，任何改变都会更新至session（其实就是重置了）。

### 用户的认证过程



1. 前一个filter传入request,response,chain 对象至APF。
2. APF创建一个证明令牌（authentication token） ，包含了用户名，密码及其他的一些来自request的信息
3. APF 把这个证明令牌传至authentication manager。
4. 而authentication manager 可以包含一个或多个认证提供者authentication providers。每个认证者提供确切的每种类型的认证。而这个managere检查哪一个Provider是支持这个来自APF的证明令牌。
5. 把这个证明令牌传给下面的其中合适的一家进行权威认证。.
6. 这个认证机构的提供者从令牌中取出用户名并且传给一个叫User Cache Service，Spring Security会把已经通过认证用户缓存起来，这样下次取用户的时候，包括用户的用户名，密码及权限都会优先从缓存取，而不是首先从数据库或其他数据源取。这样可以提高性能。
7. user cache service checks当前的用户是否已经存在于缓存中.
8. 如果存在该用户，则返回userDetail，如果不存在则返回空。
9. Authentication provider 检查user cache service是否返回空的user details.
10. 如果这个Cache返回为空， authentication provider 会继续传username至另一个service进行认证(*user details service)*。
11. 这个The user details service 通过后台数据库或其他数据源，并且按照username取出user details。一般就是调用UserDetailsService接口的loadUserByUsername方法。
12. user details service 返回user details或在没有找到该用户的时候，就会抛出an authentication exception。
13. 如果user cache service或user details service 返回一个有效的user details, 这个认证的provider重新检查token中的密码跟返回来的user details是否匹配。如果匹配，则authentication provider就会返回user details。否则抛出一个authentication exception。
14. authentication manager 返回一个details of the user 给APF。这样该用户就成功通过认证
15. APF 就会保存user details 当到前的security context 。
16. APF转到下一个filter.

# 系统的安全管理方式

目前系统中已经实现了以下的方式

1. 登录时需要加上验证码
2. 所有的数据展示及访问页需要登录后才能访问
3. 用户的数据库密码存储时使用Sha-256的加密算法
4. 登录后的所有系统的访问URL均需要授权
5. 登录多少次失败后，可锁定IP，约２０分钟后才能自动解锁。（尚未加入，容易实现）

如果需要更强的安全管理策略，可以使用更为复杂的系统配置（如通过防火墙等手段）

权限设计目前是采用基于角色控制的方式，用户需要访问系统的资源，首先必须要授予一个角色，而该角色具有访问系统资源的权限的能力，也可以认为是权限的集合。因此，一个用户要访问系统的某个资源（如产品列表），则首先要授予一个能够访问产品列表资源的角色（如productAdmin）。只要任一个用户拥有了该角色，即可以访问该资源。

系统的安全涉及到两个不同的概念，认证和授权。前者是关于确认用户是否确实是他们所宣称的身份。用户进入系统的时候，首先要进行第一个操作就是进行身份认证，即Authentication。在系统中一般表现为用户用账号跟密码登录。如果都正确了，则可以登录系统。

【说明】在现实中你可以这样理解，员工在进入公司之前，需要进行身份的确认。身份确认通过后，则可以进入公司。进入公司后，并不代表可以随便进入公司的每个办公室。这时就需要每个看当前员工具有哪些角色，即授权。授权则是关于确认用户是否有允许执行一个特定的操作。如当前员工是总经理，则可以进入总经理办公室，并且可以进入普通员工的办公区域。是因为总经理已经授权可以出入这些地方。

在本系统中，权限表现为功能菜单及系统访问的URL。

因而用户、角色、权限之间的关系可以用如下的图描述：



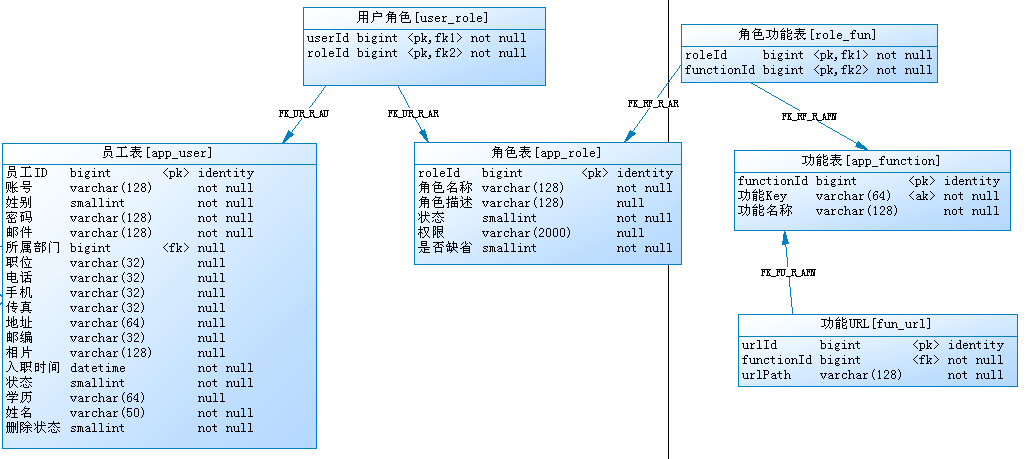
一个用户可以有多个角色，每个角色有多个功能菜单，每个功能菜单会对应多个系统访问的URL

# 整合Spring Security



## 表设计

基于以上的思想，我们设计了以下表：



* app\_user系统用户表，放置系统的所有用户。
* user\_role用户角色，用户所拥有的角色。
* app\_role角色表，放置系统的所有角色
* role\_fun角色对应的功能表，放置角色拥有的功能
* app\_function系统的功能表，放置系统参与授权的所有功能
* fun\_url系统的功能对应的权限URL表

【说明】用户表app\_user中的密码需要保存为密文，可以防止用户的密码泄露

## 与Spring Security集成说明

目前我们需要扩展Spring Security的以下两部分功能

1. 身份认证

2. 授权

### 身份认证

Spring Security是由一组的filter来进行统一的过滤，不同的filter进行相应的权限过滤功能。不过在Security跟Spring集成的过程中，其是由一个代理的类进行这些filter的统一管理。可以在web.xml中进行了查看，如下所示：

<filter>

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

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

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

</filter-mapping>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/index.jsp</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/file-upload</url-pattern>

</filter-mapping>

所有经过springSecurityFilterChain的url，都会转到DelegatingFilterProxy类的bean去处理。而该Bean在Spring Security 2.0中，已经内置于安全管理的缺省的配置当中，我们只需要把app-security.xml加入我们系统管理中来即可。如下：

而该类型的bean则在spring-sercurity.xml中进行了配置，如下所示：

<?xml version="1.0" encoding="UTF-8"?>

<b:beans xmlns="http://www.springframework.org/schema/security" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:b="http://www.springframework.org/schema/beans"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-2.0.4.xsd">

<http auto-config="true" access-denied-page="/403.jsp" lowercase-comparisons="true"

access-decision-manager-ref="accessDecisionManager">

<intercept-url pattern="/images/\*\*" filters="none"/>

<intercept-url pattern="/css/\*\*" filters="none"/>

<intercept-url pattern="/js/\*\*" filters="none"/>

<intercept-url pattern="/403\*" filters="none"/>

<intercept-url pattern="/404\*" filters="none"/>

<intercept-url pattern="/500\*" filters="none"/>

<intercept-url pattern="/ext3/\*\*" filters="none"/>

<intercept-url pattern="/fckeditor/\*\*" filters="none"/>

<intercept-url pattern="/jsonStruts\*\*" filters="none"/>

<form-login default-target-url="/index.jsp" login-page="/login.jsp" authentication-failure-url="/login.jsp?error=true" login-processing-url="/j\_security\_check" />

<logout logout-url="/j\_logout.do" logout-success-url="/login.jsp"/>

<remember-me key="RememberAppUser"/>

</http>

<b:bean id="accessDecisionManager" class="org.springframework.security.vote.AffirmativeBased">

<b:property name="allowIfAllAbstainDecisions" value="false"/>

<b:property name="decisionVoters">

<b:list>

<b:bean class="org.springframework.security.vote.RoleVoter">

<b:property name="rolePrefix" value="" />

</b:bean>

<b:bean class="org.springframework.security.vote.AuthenticatedVoter" />

</b:list>

</b:property>

</b:bean>

<authentication-manager alias="authenticationManager"/>

<authentication-provider user-service-ref="appUserDao">

<password-encoder hash="sha-256" base64="true"/>

</authentication-provider>

***<b:bean id="securityInterceptorFilter" class="com.htsoft.core.web.filter.SecurityInterceptorFilter" >***

***<custom-filter after="FILTER\_SECURITY\_INTERCEPTOR" />***

***<b:property name="securityDataSource" ref="securityDataSource"/>***

***</b:bean>***

***<b:bean id="securityDataSource" class="com.htsoft.core.security.SecurityDataSource">***

***<b:property name="appRoleService" ref="appRoleService"/>***

***<b:property name="anonymousUrls">***

***<b:set>***

***<b:value>/login.do</b:value>***

***<b:value>/check.do</b:value>***

***</b:set>***

***</b:property>***

***<b:property name="publicUrls">***

***<b:set>***

***<b:value>/modelsMenu.do</b:value>***

***<b:value>/itemsMenu.do</b:value>***

***<b:value>/file-upload</b:value>***

***<b:value>/index.jsp</b:value>***

***</b:set>***

***</b:propertiy>***

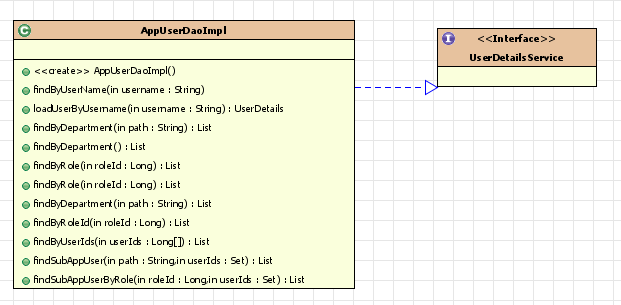
***</b:bean>***

<b:beans>

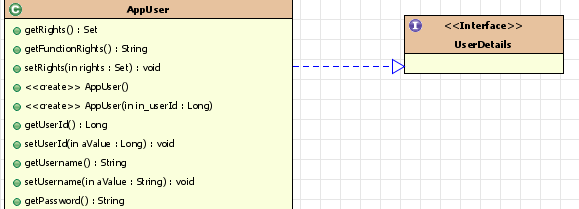
说明：当用户登录时，会根据用户账号及密码进行身份验证，验证由authenticationManager来进，其会调用实现UserDetailsService接口实现类完成，在本系统，是由appUserDaoImpl类来实现。



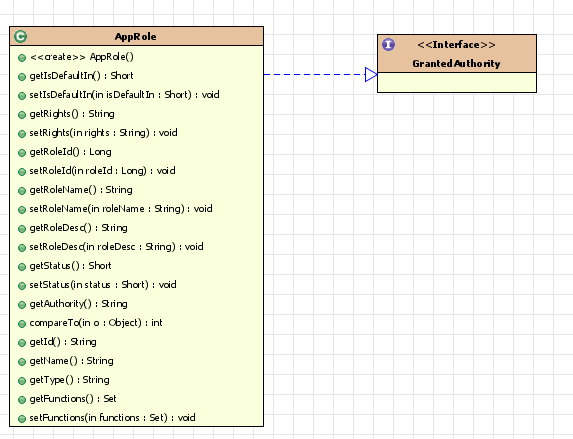
### 访问授权



而我们的用户及角色实体要成为安全框架识别的安全实体，需要相应实现不同的接口，如下所示：



角色需要实现GranteAuthority的接口，如下所示：



授权的管理是通过Filter来进行的，用户访问URL时，均需要经过Spring Security的URL进行授权。在本系统中，这个功能是通过SecurityInterceptorFilter来进行。

系统启动时，会把所有的权限以[角色—URL列表]的形式放置在一个全局的Map中（可参见代码SecurityDataSource中的getDataSource()方法提供实现），用户访问系统的URL时，就会根据当前用户所拥有的角色是否包含此URL。这个全局的权限匹配源则由SecurityDataSource来提供。由于登录用户在进入系统后，都会具备一些常用的功能，所以每个用户均有一个PUBLIC\_ROLE的角色，代表可以访问系统的公告资源。该角色对应的可访问的URL，则配置在SecurityDataSource Bean中的publicUrls属性中。



扩展说明：

在这里我们加入自己的Filter,当以上访问均没有通过认证时，其就会进入SecurityInterceptorFileter，在此类中，我们根据当前用户的所拥有的角色来决定是否进入下一个Filter的处理，若没办法通过权限认证，则会抛出一个拒绝授权访问的错误，系统就会进入403页面。403页中写入响应的文件头“\_\_forbidden”，EXT展示层根据Connection取到的文件头而相应弹出相应的提示信息。

### 权限的配置及使用

以上权限的配置是基于服务端的，实际上，权限的管理并不仅限于服务端的URL拦截，还表现为前台功能的显示及访问。鉴于Spring Security2的配置方式，同时能够结合EXT3来进行整个系统的权限管理，通过使用配置文件，进行整个系统的功能集中管理，包括系统左边的导航菜 单，数据列表中的功能操作权限。我们知道，在传统的Web项目中，我们都是采用基于URL进行权限控制的。基于EXT的应用也不例外，只不过我们是需要结 合前台的功能菜单一起，前台显示出来的功能菜单，其后台均代表可以访问。

#### Menu中的功能菜单配置

先在js/menu/xml/menu-all.xml中加入我们需要的配置模块，如：

<module name="ent" file="menu-ent.xml"/>

然后在menu-ent.xml中再加上我们的模块功能，如下所示：<Menus>

<Items id="SystemSetting" text="系统设置" iconCls="menu-system">

...

<Item id="AppRoleView" iconCls="menu-role" text="角色设置">

<Function id="\_AppRoleList" text="查看角色" iconCls="menu-list">

<url>/system/listAppRole.do</url>

</Function>

<Function id="\_AppRoleAdd" text="添加角色" iconCls="menu-add">

<url>/system/listAppRole.do</url>

<url>/system/saveAppRole.do</url>

</Function>

<Function id="\_AppRoleEdit" text="编辑角色" iconCls="menu-add">

<url>/system/listAppRole.do</url>

<url>/system/saveAppRole.do</url>

</Function>

<Function id="\_AppRoleDel" text="删除角色" iconCls="menu-del">

<url>/system/listAppRole.do</url>

<url>/system/mulDelAppRole.do</url>

</Function>

<Function id="\_AppRoleGrant" text="授权角色">

<url>/system/listAppRole.do</url>

<url>/system/grantAppRole.do</url>

</Function>

</Item>

...

<Item id="ReportTemplateView" iconCls="menu-report" text="报表管理">

...

</Item>

</Items>

<!--其他模块 -->

<Items>

...

</Items>

</Menus>

***<?xml version="1.0" encoding="UTF-8"?>***

***<Menus>***

***<Items id="SystemSetting" text="系统设置" iconCls="menu-system">***

***...***

***<Item id="AppRoleView" iconCls="menu-role" text="角色设置">***

***<Function id="\_AppRoleList" text="查看角色" iconCls="menu-list">***

***<url>/system/listAppRole.do</url>***

***</Function>***

***<Function id="\_AppRoleAdd" text="添加角色" iconCls="menu-add">***

***<url>/system/listAppRole.do</url>***

***<url>/system/saveAppRole.do</url>***

***</Function>***

***<Function id="\_AppRoleEdit" text="编辑角色" iconCls="menu-add">***

***<url>/system/listAppRole.do</url>***

***<url>/system/saveAppRole.do</url>***

***</Function>***

***<Function id="\_AppRoleDel" text="删除角色" iconCls="menu-del">***

***<url>/system/listAppRole.do</url>***

***<url>/system/mulDelAppRole.do</url>***

***</Function>***

***<Function id="\_AppRoleGrant" text="授权角色">***

***<url>/system/listAppRole.do</url>***

***<url>/system/grantAppRole.do</url>***

***</Function>***

***</Item>***

***...***

***<Item id="ReportTemplateView" iconCls="menu-report" text="报表管理">***

***...***

***</Item>***

***</Items>***

***<!--其他模块 -->***

***<Items>***

***...***

***</Items>***

***</Menus>***

menu.xml为整个系统的功能列表，系统的左边菜单也是从该文件显示出来，如：



Menu.xml是在系统启动时，由AppUtil.java中的init方法进行初始化,并构造一个显示左边的菜单的全局Document,同时并通过这个它构造参与授权的Document.每个用户进入系统后，会根据当前用户所拥有的角色及角色对应的权限，进行显示授权的菜单及功能按钮。

#### 界面功能划分

进入某项菜单中，会进入相应的模块的管理界面，如角色管理：



上面的功能“添加角色”，“删除角色”以及管理一栏中的“删除”，“编辑”，“授权”等功能，则对应为menu.xml中的Item下的Function标签的配置，如：

**<Function id="\_AppRoleAdd" text="添加角色" iconCls="menu-add">**

**<url>/system/listAppRole.do</url>**

**<url>/system/saveAppRole.do</url>**

**</Function>**

添加角色对应：

<Function id="\_AppRoleAdd" text="添加角色" iconCls="menu-add">

<url>/system/listAppRole.do</url>

<url>/system/saveAppRole.do</url>

</Function>

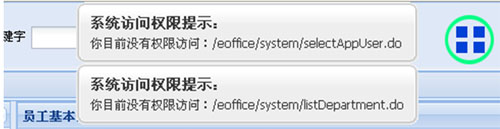
<Function id="\_AppRoleAdd" text="添加角色" iconCls="menu-add">

<url>/system/listAppRole.do</url>

<url>/system/saveAppRole.do</url>

</Function>

其下包括两个Url，一个是/system/listAppRole.do，另一Url /system/saveAppRole.do，表示当我们访问这个功能完成角色添加时，需要访问这两个URL，后台会根据当前用户的角色是否授权访问这 个功能而决定是否开放这两个URL给当前用户访问，否则，前台浏览器会弹出一个类似如下的提示，告诉用户当前没有权限访问这个URL。



当然，这种情况还是要避免出现，因为系统在客户端出现的功能菜单，均是有权访问的。

#### 前台的代码配置

前台的用户登录系统后，会有一个全局的变量存储用户的所有权限（其会把所有的角色的配置抽取出来，去掉重复的权限配置），可以在客户端直接检查用户是否有权限访问某一个菜单的功能，如我们检查用户是否直接有访问“添加角色”的功能，则可以调用系统提供的方法：

    isGranted(‘\_AppRoleAdd’);

该函数返回为true则代表可以访问。

因此我们若要进行更高的权限粒度控制，我们需要把原来的代码进行改装一下。如：

AppRoleView.js需要进行更改。

修改一:

**var toolbar = new Ext.Toolbar({**

**id : 'AppRoleFootBar',**

**height : 30,**

**bodyStyle:'text-align:left',**

**items : []**

**});**

**if(isGranted('\_AppRoleAdd')){**

**toolbar.add(new Ext.Button({**

**iconCls : 'btn-add',**

**text : '添加角色',**

**handler : function() {**

**new AppRoleForm();**

**}**

**}));**

**}**

**if (isGranted('\_AppRoleDel')) {**

**toolbar.add(new Ext.Button({**

**iconCls : 'btn-del',**

**text : '删除角色',**

**handler : function() {**

**var grid = Ext.getCmp("AppRoleGrid");**

**var selectRecords = grid.getSelectionModel().getSelections();**

**if (selectRecords.length == 0) {**

**Ext.Msg.alert("信息", "请选择要删除的记录！");**

**return;**

**}**

**var ids = Array();**

**for (var i = 0; i < selectRecords.length; i++) {**

**ids.push(selectRecords[i].data.roleId);**

**}**

**AppRoleView.remove(ids);**

**}**

**}));**

**}** var toolbar = new Ext.Toolbar({

id : 'AppRoleFootBar',

height : 30,

bodyStyle:'text-align:left',

items : []

});

if(isGranted('\_AppRoleAdd')){

toolbar.add(new Ext.Button({

iconCls : 'btn-add',

text : '添加角色',

handler : function() {

new AppRoleForm();

}

}));

}

if (isGranted('\_AppRoleDel')) {

toolbar.add(new Ext.Button({

iconCls : 'btn-del',

text : '删除角色',

handler : function() {

var grid = Ext.getCmp("AppRoleGrid");

var selectRecords = grid.getSelectionModel().getSelections();

if (selectRecords.length == 0) {

Ext.Msg.alert("信息", "请选择要删除的记录！");

return;

}

var ids = Array();

for (var i = 0; i < selectRecords.length; i++) {

ids.push(selectRecords[i].data.roleId);

}

AppRoleView.remove(ids);

}

}));

}

修改二  
    管理列中的栏目功能：

**if(isGranted('\_AppRoleDel'))**

**str = '<button title="删除" value=" " class="btn-del" onclick="AppRoleView.remove('+ editId + ')"></button>';**

**if(isGranted('\_AppRoleEdit'))**

**str += '&nbsp;<button title="编辑" value=" " class="btn-edit" onclick="AppRoleView.edit('+ editId + ')"></button>';**

**if(isGranted('\_AppRoleGrant'))**

**str += '&nbsp;<button title="授权" value=" " class="btn-grant" onclick="AppRoleView.grant('+ editId + ',\'' + roleName + '\')">&nbsp;</button>';**

**if(isGranted('\_AppRoleDel'))**

**str = '<button title="删除" value=" " class="btn-del" onclick="AppRoleView.remove('+ editId + ')"></button>';**

**if(isGranted('\_AppRoleEdit'))**

**str += '&nbsp;<button title="编辑" value=" " class="btn-edit" onclick="AppRoleView.edit('+ editId + ')"></button>';**

**if(isGranted('\_AppRoleGrant'))**

**str += '&nbsp;<button title="授权" value=" " class="btn-grant" onclick="AppRoleView.grant('+ editId + ',\'' + roleName + '\')">&nbsp;</button>';**

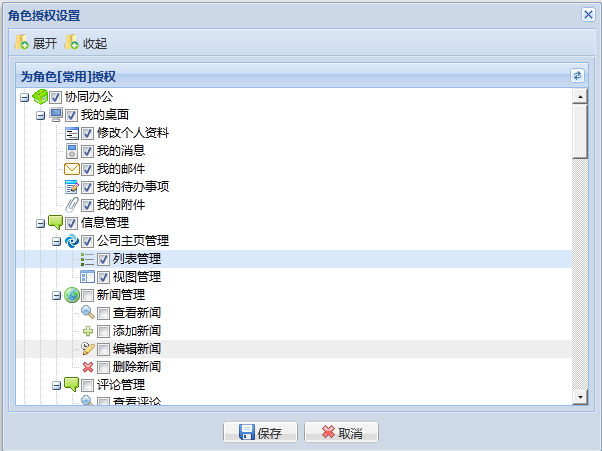
#### 同步menu.xml的数据至数据库

打开配置common/config/下的config.properties的这个配置，如下： isSynMenu=true

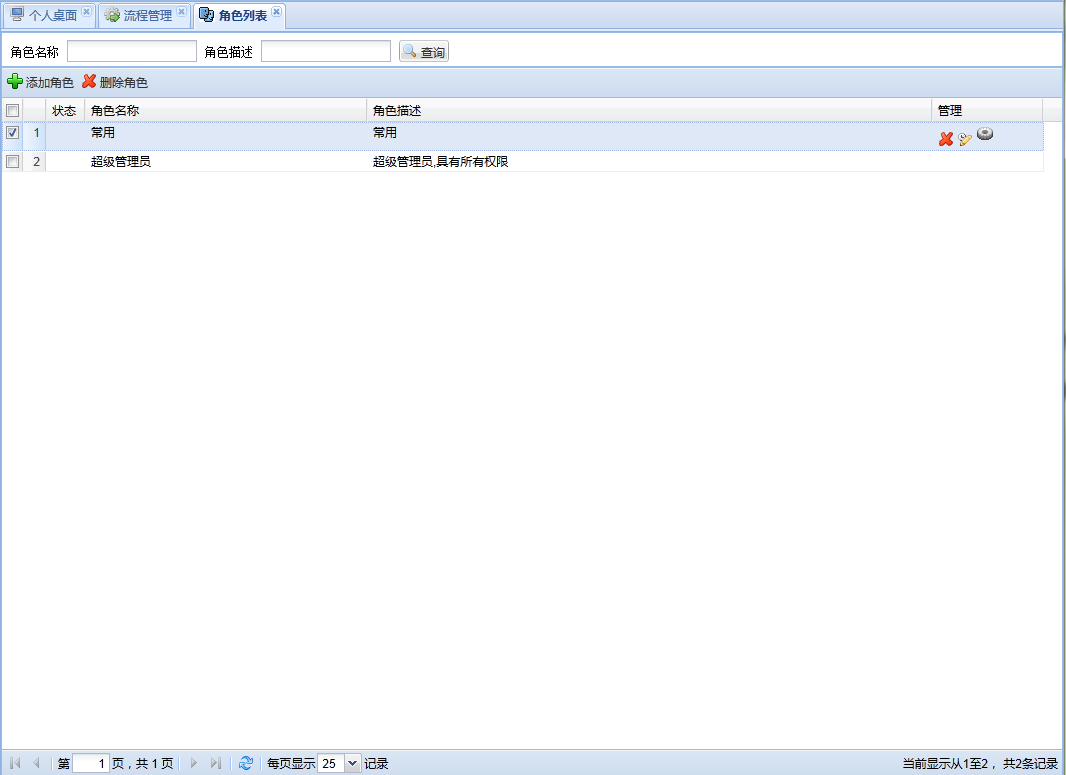
打开此配置则表示会把menu.xml中的所有的Function中的url同步到数据库，转给Spring Security进行匹配，则此时进行角色的权限授予才会真正生效。

这个同步的过程可以在AppUtil类中的init方法可以看到。

#### 为角色进行相应的授权



如按以上给某角色授权，这时该用户访问该模块时，只有“添加角色”这个功能。同样，以上的权限管理也控制着整个系统的左边菜单的权限，如下图所示:



角色授权由类AppRoleAction中的grant方法来完成的。