Spring Security3的使用方法有4种：

一种是全部利用配置文件，将用户、权限、资源(url)硬编码在xml文件中。

二种是用户和权限用数据库存储，而资源(url)和权限的对应采用硬编码配置。

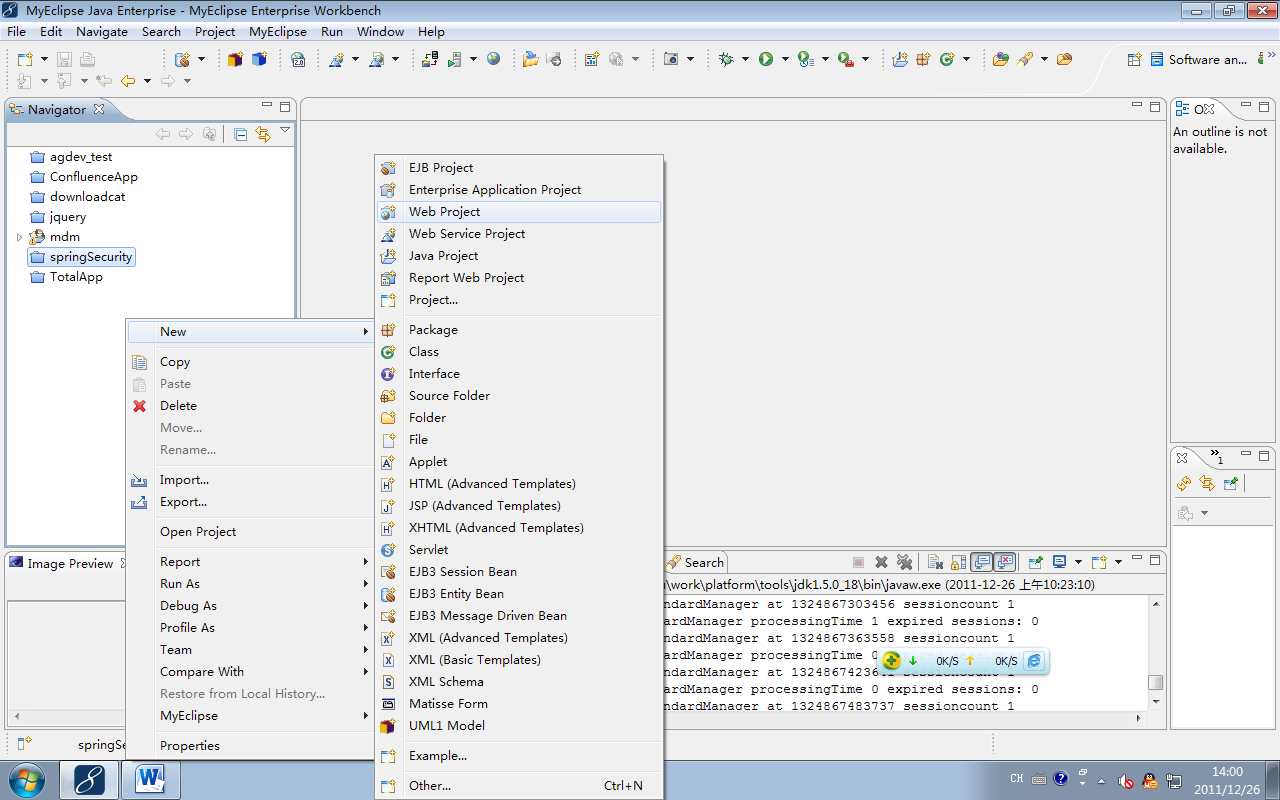
三种是细分角色和权限，并将用户、角色、权限和资源均采用数据库存储，并且自定义过滤器，代替原有的FilterSecurityInterceptor过滤器，并分别实现AccessDecisionManager、InvocationSecurityMetadataSourceService和UserDetailsService，并在配置文件中进行相应配置。

四是修改spring security的源代码，主要是修改InvocationSecurityMetadataSourceService和UserDetailsService两个类。前者是将配置文件或数据库中存储的资源(url)提取出来加工成为url和权限列表的Map供Security使用，后者提取用户名和权限组成一个完整的 (UserDetails)User对象，该对象可以提供用户的详细信息供AuthentationManager进行认证与授权使用。该方法理论上可行，但是比较暴力，不推荐使用。

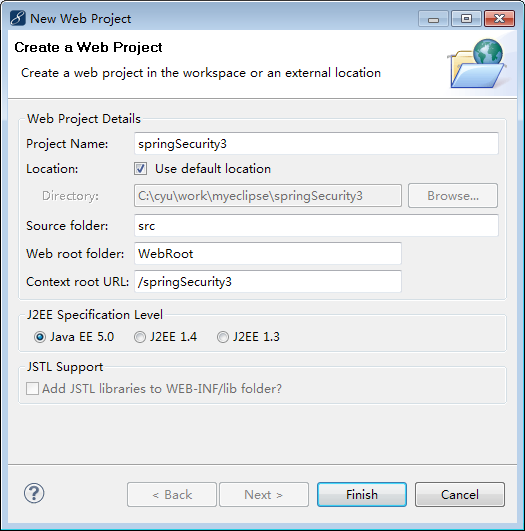
本文有两个例子，我在简单例子章节实现了第一种方法。在复杂例子章节实现了第二种和第三种方法组合使用的例子。简单例子通俗易懂，不再赘述。复杂例子及其使用和扩展，我将穿插详细的配置注释和讲解，包括整个程序的执行过程。

# 简单例子

创建web工程如下图所示：

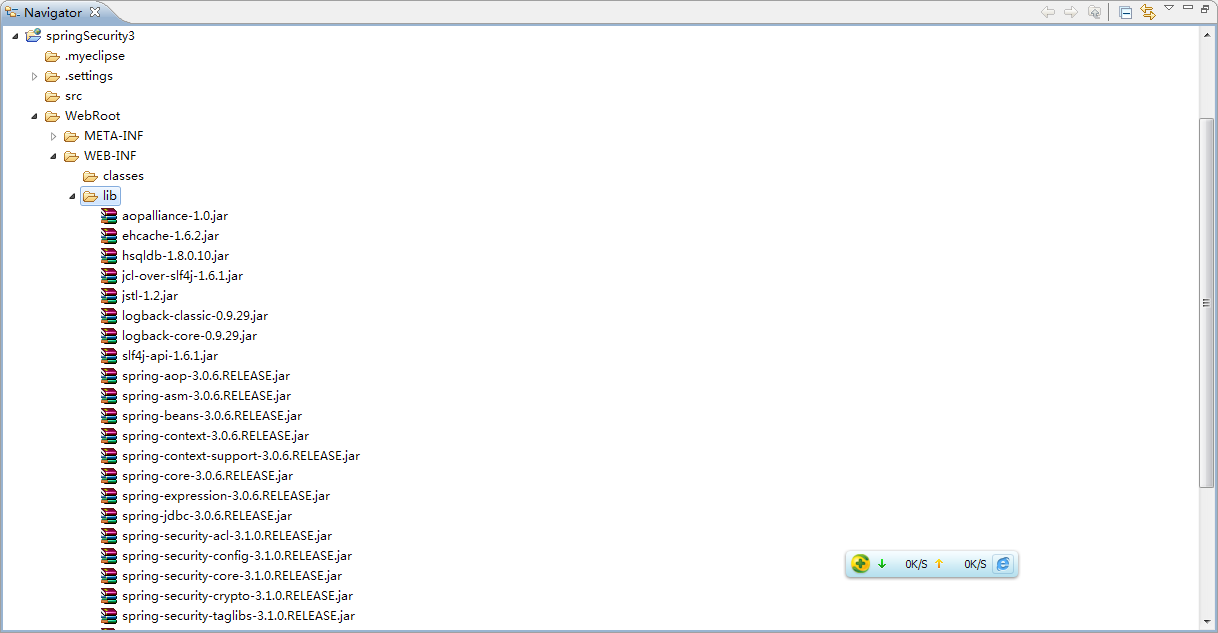


配置如下图所示：



单击Finish即可。

把从spring网站下载的spring-security-3.1.0.RELEASE解压，并将其中的spring-security-samples-contacts-3.1.0.RELEASE.war解压，把WEB-INF\lib中的jar包拷贝到如下图所示：



修改配置web.xml如下：

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

<web-app version=*"2.5"* xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<!-- 加载Spring XML配置文件 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

classpath:securityConfig.xml

</param-value>

</context-param>

<!-- Spring Secutiry3.1的过滤器链配置 -->

<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>/\*</url-pattern>

</filter-mapping>

<!-- Spring 容器启动监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!--系统欢迎页面-->

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

在src中创建securityConfig.xml内容如下：

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

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

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

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.1.xsd"*>

<!-- 登录页面不过滤 -->

<http pattern=*"/login.jsp"* security=*"none"*/>

<http access-denied-page=*"/accessDenied.jsp"* >

<form-login login-page=*"/login.jsp"* />

<!-- 访问/admin.jsp资源的用户必须具有ROLE\_ADMIN的权限 -->

<intercept-url pattern=*"/admin.jsp"* access=*"ROLE\_ADMIN"*/>

<!-- 访问/\*\*资源的用户必须具有ROLE\_USER的权限 -->

<intercept-url pattern=*"/\*\*"* access=*"ROLE\_USER"*/>

<session-management>

<concurrency-control max-sessions=*"1"* error-if-maximum-exceeded=*"false"*/>

</session-management>

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name=*"cyu"* password=*"sap123"* authorities=*"ROLE\_USER"*/>

</user-service>

</authentication-provider>

</authentication-manager>

</b:beans>

在WebRoot中创建login.jsp内容如下：

<%@page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>登录</title>

</head>

<body>

<form action=*"j\_spring\_security\_check"* method=*"POST"*>

<table>

<tr>

<td>用户:</td>

<td><input type=*'text'* name=*'j\_username'*></td>

</tr>

<tr>

<td>密码:</td>

<td><input type=*'password'* name=*'j\_password'*></td>

</tr>

<tr>

<td><input name=*"reset"* type=*"reset"*></td>

<td><input name=*"submit"* type=*"submit"*></td>

</tr>

</table>

</form>

</body>

</html>

在WebRoot中创建accessDenied.jsp，内容如下：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>访问拒绝</title>

</head>

<body>

您的访问被拒绝，无权访问该资源！ <br>

</body>

</html>

在WebRoot中创建admin.jsp内容如下：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>My JSP 'admin.jsp' starting page</title>

</head>

<body>

欢迎来到管理员页面. <br>

</body>

</html>

修改index.jsp内容如下：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

<%@ taglib prefix=*"sec"* uri=*"http://www.springframework.org/security/tags"* %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

</head>

<body>

这是首页，欢迎<sec:authentication property=*"name"*/>!<br>

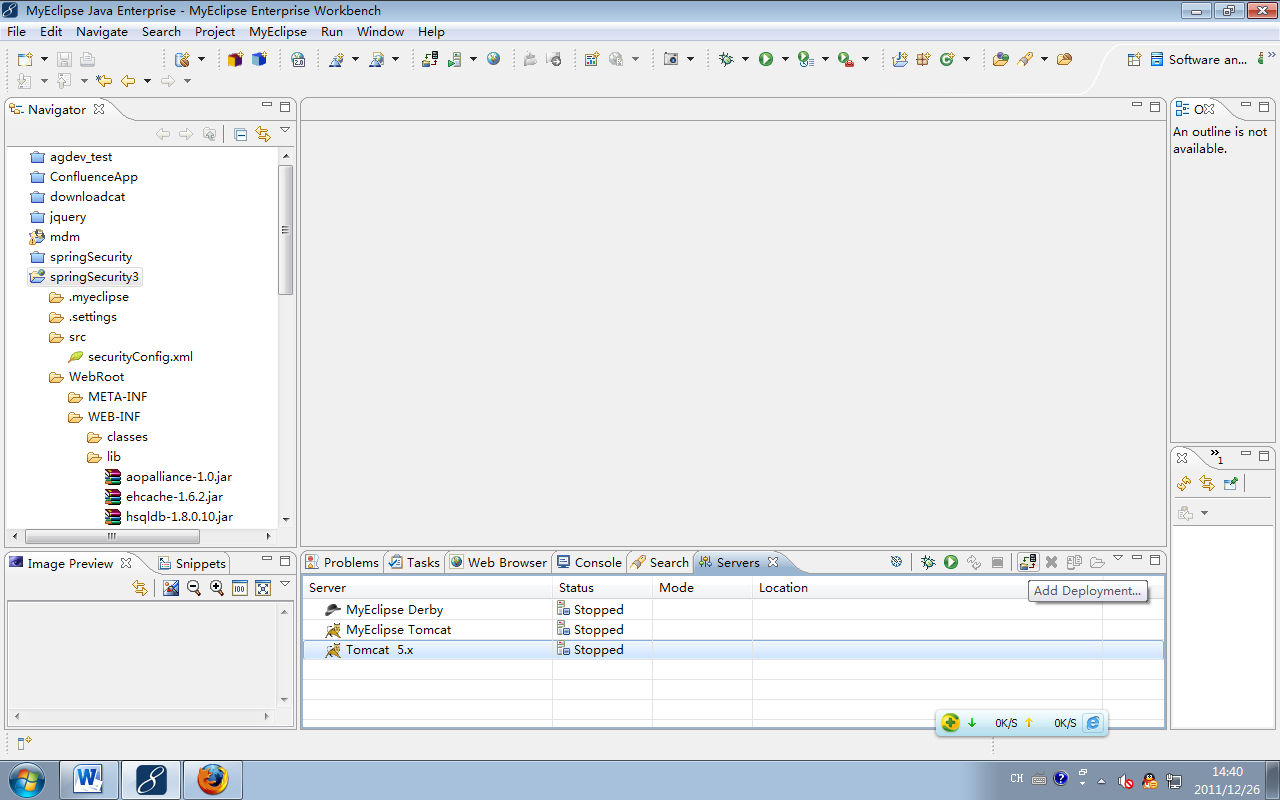
<a href=*"admin.jsp"*>进入admin页面</a>

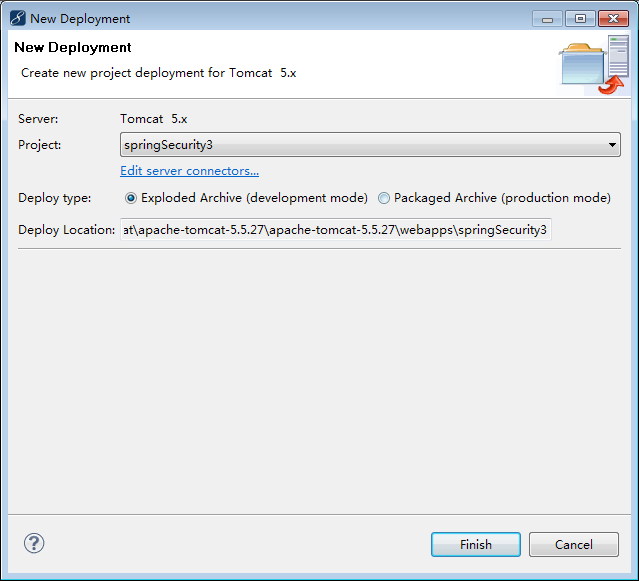
<a href=*"other.jsp"*>进入其它页面</a>

</body>

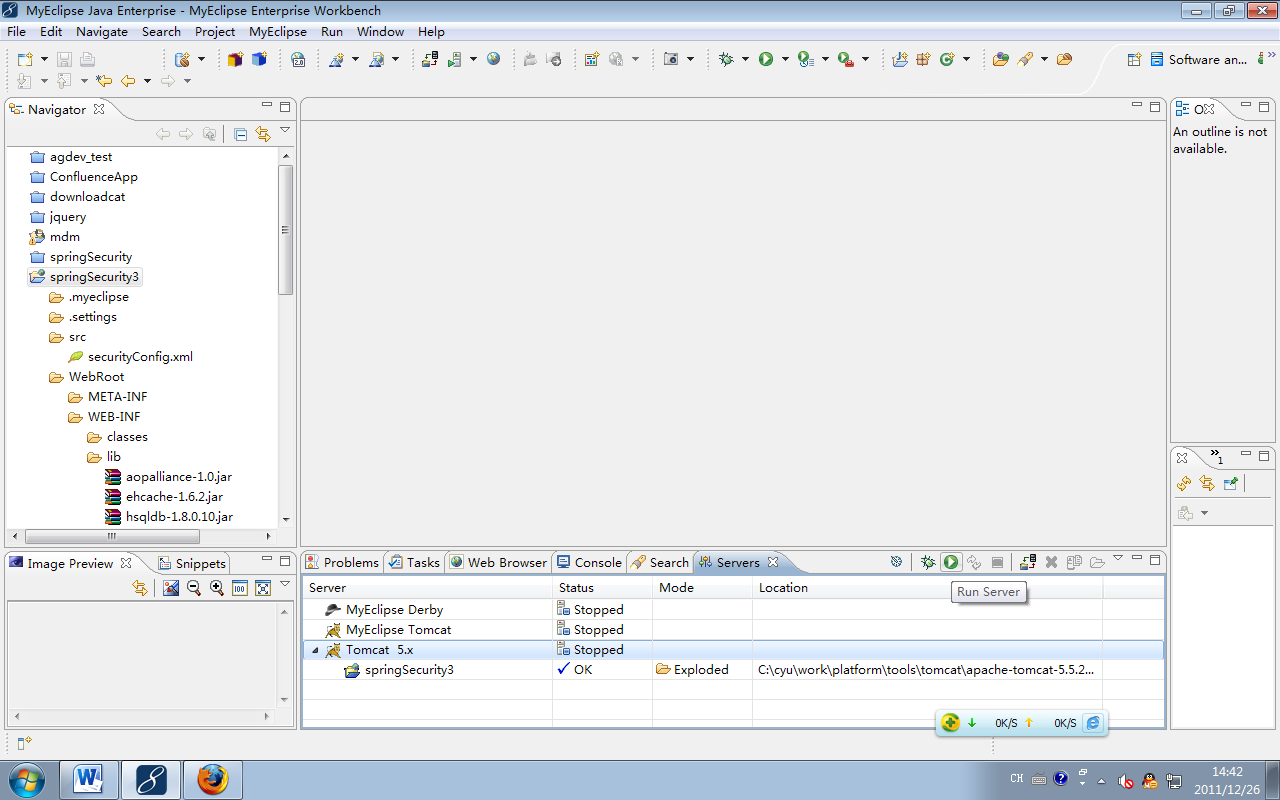
</html>

添加工程如下图所示：





点击Finish即可，然后运行工程如下图所示：



测试页面如下：



输入用户：cyu密码：sap123，然后回车：



点击“进入admin页面”超链，得到如下图所示：



# 复杂例子

修改securityConfig.xml内容如下：

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

<beans:beans xmlns=*"http://www.springframework.org/schema/security"*

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

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.1.xsd"*>

<!-- 登录页面不过滤 -->

<http pattern=*"/login.jsp"* security=*"none"*/>

<http access-denied-page=*"/accessDenied.jsp"* >

<form-login login-page=*"/login.jsp"* />

<!-- 访问/admin.jsp资源的用户必须具有ROLE\_ADMIN的权限 -->

<intercept-url pattern=*"/admin.jsp"* access=*"ROLE\_ADMIN"*/>

<!-- 访问/\*\*资源的用户必须具有ROLE\_USER的权限 -->

<intercept-url pattern=*"/\*\*"* access=*"ROLE\_USER"*/>

<session-management>

<concurrency-control max-sessions=*"1"* error-if-maximum-exceeded=*"false"*/>

</session-management>

<!-- 增加一个filter，这点与Acegi是不一样的，不能修改默认的filter了，

这个filter位于FILTER\_SECURITY\_INTERCEPTOR之前 -->

<custom-filter ref=*"myFilter"* before=*"FILTER\_SECURITY\_INTERCEPTOR"* />

</http>

<!-- 一个自定义的filter，必须包含authenticationManager,accessDecisionManager,securityMetadataSource三个属性，

我们的所有控制将在这三个类中实现，解释详见具体配置 -->

<beans:bean id=*"myFilter"* class=*"com.aostarit.spring.security.MyFilterSecurityInterceptor"*>

<beans:property name=*"authenticationManager"*

ref=*"authenticationManager"* />

<beans:property name=*"accessDecisionManager"*

ref=*"myAccessDecisionManagerBean"* />

<beans:property name=*"securityMetadataSource"*

ref=*"securityMetadataSource"* />

</beans:bean>

<!-- 验证配置 ， 认证管理器，实现用户认证的入口，主要实现UserDetailsService接口即可 -->

<authentication-manager alias=*"authenticationManager"*>

<authentication-provider

user-service-ref=*"myUserDetailService"*>

<!-- 如果用户的密码采用加密的话

<password-encoder hash="md5" />

-->

</authentication-provider>

</authentication-manager>

<!-- 在这个类中，你就可以从数据库中读入用户的密码，角色信息，是否锁定，账号是否过期等 -->

<beans:bean id=*"myUserDetailService"*

class=*"com.aostarit.spring.security.MyUserDetailService"* />

<!-- 访问决策器，决定某个用户具有的角色，是否有足够的权限去访问某个资源 -->

<beans:bean id=*"myAccessDecisionManagerBean"*

class=*"com.aostarit.spring.security.MyAccessDecisionManager"*>

</beans:bean>

<!-- 资源源数据定义，将所有的资源和权限对应关系建立起来，即定义某一资源可以被哪些角色访问 -->

<beans:bean id=*"securityMetadataSource"*

class=*"com.aostarit.spring.security.MyInvocationSecurityMetadataSource"* />

</beans:beans>

编写UrlMatcher接口，内容如下：

**package** com.aostarit.spring.security.tool;

**public** **abstract** **interface** UrlMatcher

{

**public** **abstract** Object compile(String paramString);

**public** **abstract** **boolean** pathMatchesUrl(Object paramObject, String paramString);

**public** **abstract** String getUniversalMatchPattern();

**public** **abstract** **boolean** requiresLowerCaseUrl();

}

**这个接口是以前spring版本中的，现在的spring版本中不存在，由于项目需要且使用方便，故加入到项目当中。**

编写AntUrlPathMatcher类，内容如下：

**package** com.aostarit.spring.security.tool;

**import** org.springframework.util.AntPathMatcher;

**import** org.springframework.util.PathMatcher;

**public** **class** AntUrlPathMatcher

**implements** UrlMatcher

{

**private** **boolean** requiresLowerCaseUrl;

**private** PathMatcher pathMatcher;

**public** AntUrlPathMatcher()

{

**this**(**true**);

}

**public** AntUrlPathMatcher(**boolean** requiresLowerCaseUrl)

{

**this**.requiresLowerCaseUrl = **true**;

**this**.pathMatcher = **new** AntPathMatcher();

**this**.requiresLowerCaseUrl = requiresLowerCaseUrl;

}

**public** Object compile(String path) {

**if** (**this**.requiresLowerCaseUrl) {

**return** path.toLowerCase();

}

**return** path;

}

**public** **void** setRequiresLowerCaseUrl(**boolean** requiresLowerCaseUrl) {

**this**.requiresLowerCaseUrl = requiresLowerCaseUrl;

}

**public** **boolean** pathMatchesUrl(Object path, String url) {

**if** (("/\*\*".equals(path)) || ("\*\*".equals(path))) {

**return** **true**;

}

**return** **this**.pathMatcher.match((String)path, url);

}

**public** String getUniversalMatchPattern() {

**return** "/\*\*";

}

**public** **boolean** requiresLowerCaseUrl() {

**return** **this**.requiresLowerCaseUrl;

}

**public** String toString() {

**return** **super**.getClass().getName() + "[requiresLowerCase='" + **this**.requiresLowerCaseUrl + "']";

}

}

**这个类是以前spring版本中的工具类，现在的spring版本中不存在，由于项目需要且使用方便，故加入到项目当中。**

编写MyFilterSecurityInterceptor类，内容如下：

**package** com.aostarit.spring.security;

**import** java.io.IOException;

**import** javax.servlet.Filter;

**import** javax.servlet.FilterChain;

**import** javax.servlet.FilterConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.ServletRequest;

**import** javax.servlet.ServletResponse;

**import** org.springframework.security.access.SecurityMetadataSource;

**import** org.springframework.security.access.intercept.AbstractSecurityInterceptor;

**import** org.springframework.security.access.intercept.InterceptorStatusToken;

**import** org.springframework.security.web.FilterInvocation;

**import** org.springframework.security.web.access.intercept.FilterInvocationSecurityMetadataSource;

**public** **class** MyFilterSecurityInterceptor **extends** AbstractSecurityInterceptor

**implements** Filter {

**private** FilterInvocationSecurityMetadataSource securityMetadataSource;

/\*\*

\* Method that is actually called by the filter chain. Simply delegates to

\* the {@link #invoke(FilterInvocation)} method.

\*

\* **@param** request

\* the servlet request

\* **@param** response

\* the servlet response

\* **@param** chain

\* the filter chain

\*

\* **@throws** IOException

\* if the filter chain fails

\* **@throws** ServletException

\* if the filter chain fails

\*/

**public** **void** doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) **throws** IOException, ServletException {

FilterInvocation fi = **new** FilterInvocation(request, response, chain);

invoke(fi);

}

**public** FilterInvocationSecurityMetadataSource getSecurityMetadataSource() {

**return** **this**.securityMetadataSource;

}

**public** Class<? **extends** Object> getSecureObjectClass() {

**return** FilterInvocation.**class**;

}

**public** **void** invoke(FilterInvocation fi) **throws** IOException,

ServletException {

InterceptorStatusToken token = **super**.beforeInvocation(fi);

**try** {

fi.getChain().doFilter(fi.getRequest(), fi.getResponse());

} **finally** {

**super**.afterInvocation(token, **null**);

}

}

**public** SecurityMetadataSource obtainSecurityMetadataSource() {

**return** **this**.securityMetadataSource;

}

**public** **void** setSecurityMetadataSource(

FilterInvocationSecurityMetadataSource newSource) {

**this**.securityMetadataSource = newSource;

}

**public** **void** destroy() {

}

**public** **void** init(FilterConfig arg0) **throws** ServletException {

}

}

**核心的是InterceptorStatusToken token = super.beforeInvocation(fi);会调用我们定义的accessDecisionManager:decide(Object object)和securityMetadataSource:getAttributes(Object object)方法。**

编写MyInvocationSecurityMetadataSource类，内容如下：

**package** com.aostarit.spring.security;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Map;

**import** org.springframework.security.access.ConfigAttribute;

**import** org.springframework.security.access.SecurityConfig;

**import** org.springframework.security.web.FilterInvocation;

**import** org.springframework.security.web.access.intercept.FilterInvocationSecurityMetadataSource;

**import** com.aostarit.spring.security.tool.AntUrlPathMatcher;

**import** com.aostarit.spring.security.tool.UrlMatcher;

/\*\*

\*

\* 此类在初始化时，应该取到所有资源及其对应角色的定义

\*

\*/

**public** **class** MyInvocationSecurityMetadataSource

**implements** FilterInvocationSecurityMetadataSource {

**private** UrlMatcher urlMatcher = **new** AntUrlPathMatcher();

**private** **static** Map<String, Collection<ConfigAttribute>> *resourceMap* = **null**;

**public** MyInvocationSecurityMetadataSource() {

loadResourceDefine();

}

**private** **void** loadResourceDefine() {

*resourceMap* = **new** HashMap<String, Collection<ConfigAttribute>>();

Collection<ConfigAttribute> atts = **new** ArrayList<ConfigAttribute>();

ConfigAttribute ca = **new** SecurityConfig("ROLE\_USER");

atts.add(ca);

*resourceMap*.put("/index.jsp", atts);

Collection<ConfigAttribute> attsno = **new** ArrayList<ConfigAttribute>();

ConfigAttribute cano = **new** SecurityConfig("ROLE\_NO");

attsno.add(cano);

*resourceMap*.put("/other.jsp", attsno);

}

// According to a URL, Find out permission configuration of this URL.

**public** Collection<ConfigAttribute> getAttributes(Object object)

**throws** IllegalArgumentException {

// guess object is a URL.

String url = ((FilterInvocation)object).getRequestUrl();

Iterator<String> ite = *resourceMap*.keySet().iterator();

**while** (ite.hasNext()) {

String resURL = ite.next();

**if** (urlMatcher.pathMatchesUrl(resURL, url)) {

**return** *resourceMap*.get(resURL);

}

}

**return** **null**;

}

**public** **boolean** supports(Class<?> clazz) {

**return** **true**;

}

**public** Collection<ConfigAttribute> getAllConfigAttributes() {

**return** **null**;

}

}

**对于资源的访问权限的定义，我们通过实现FilterInvocationSecurityMetadataSource这个接口来初始化数据。看看loadResourceDefine方法，我在这里，假定index.jsp这个资源，需要ROLE\_USER角色的用户才能访问, other.jsp这个资源，需要ROLE\_NO角色的用户才能访问。这个类中，还有一个最核心的地方，就是提供某个资源对应的权限定义，即getAttributes方法返回的结果。注意，我例子中使用的是 AntUrlPathMatcher这个path matcher来检查URL是否与资源定义匹配，事实上你还要用正则的方式来匹配，或者自己实现一个matcher。**

**这里的角色和资源都可以从数据库中获取，建议通过我们封装的平台级持久层管理类获取和管理。**

编写MyAccessDecisionManager类，内容如下：

**package** com.aostarit.spring.security;

**import** java.util.Collection;

**import** java.util.Iterator;

**import** org.springframework.security.access.AccessDecisionManager;

**import** org.springframework.security.access.AccessDeniedException;

**import** org.springframework.security.access.ConfigAttribute;

**import** org.springframework.security.access.SecurityConfig;

**import** org.springframework.security.authentication.InsufficientAuthenticationException;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.GrantedAuthority;

**public** **class** MyAccessDecisionManager **implements** AccessDecisionManager {

//In this method, need to compare authentication with configAttributes.

// 1, A object is a URL, a filter was find permission configuration by this URL, and pass to here.

// 2, Check authentication has attribute in permission configuration (configAttributes)

// 3, If not match corresponding authentication, throw a AccessDeniedException.

**public** **void** decide(Authentication authentication, Object object,

Collection<ConfigAttribute> configAttributes)

**throws** AccessDeniedException, InsufficientAuthenticationException {

**if**(configAttributes == **null**){

**return** ;

}

System.*out*.println(object.toString()); //object is a URL.

Iterator<ConfigAttribute> ite=configAttributes.iterator();

**while**(ite.hasNext()){

ConfigAttribute ca=ite.next();

String needRole=((SecurityConfig)ca).getAttribute();

**for**(GrantedAuthority ga:authentication.getAuthorities()){

**if**(needRole.equals(ga.getAuthority())){ //ga is user's role.

**return**;

}

}

}

**throw** **new** AccessDeniedException("no right");

}

**public** **boolean** supports(ConfigAttribute attribute) {

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

**return** **true**;

}

**public** **boolean** supports(Class<?> clazz) {

**return** **true**;

}

}

**在这个类中，最重要的是decide方法，如果不存在对该资源的定义，直接放行；否则，如果找到正确的角色，即认为拥有权限，并放行，否则throw new AccessDeniedException("no right")。所有的异常建议平台统一进行封装并管理。**

编写MyUserDetailService类，内容如下：

**package** com.aostarit.spring.security;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** org.springframework.dao.DataAccessException;

**import** org.springframework.security.core.GrantedAuthority;

**import** org.springframework.security.core.authority.~~GrantedAuthorityImpl~~;

**import** org.springframework.security.core.userdetails.User;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.core.userdetails.UserDetailsService;

**import** org.springframework.security.core.userdetails.UsernameNotFoundException;

**public** **class** MyUserDetailService **implements** UserDetailsService {

**public** UserDetails loadUserByUsername(String username)

**throws** UsernameNotFoundException, DataAccessException {

Collection<GrantedAuthority> auths=**new** ArrayList<GrantedAuthority>();

~~GrantedAuthorityImpl~~ auth2=**new** GrantedAuthorityImpl("ROLE\_ADMIN");

// auths.add(auth2);

**if**(username.equals("cyu")){

auths=**new** ArrayList<GrantedAuthority>();

~~GrantedAuthorityImpl~~ auth1=**new** GrantedAuthorityImpl("ROLE\_USER");

auths.add(auth1);

auths.add(auth2);

}

// User(String username, String password, boolean enabled, boolean accountNonExpired,

// boolean credentialsNonExpired, boolean accountNonLocked, Collection<GrantedAuthority> authorities) {

User user = **new** User(username,

"sap123", **true**, **true**, **true**, **true**, auths);

**return** user;

}

}

**在这个类中，你就可以从数据库中读入用户的密码，角色信息，是否锁定，账号是否过期等。建议通过我们封装的平台级持久层管理类获取和管理。**

* **整个程序执行的过程如下：**

**1、容器启动(MyInvocationSecurityMetadataSource：loadResourceDefine加载系统资源与权限列表)**

**2、用户发出请求**

**3、过滤器拦截(MyFilterSecurityInterceptor:doFilter)**

**4、取得请求资源所需权限(MyInvocationSecurityMetadataSource:getAttributes)**

**5、匹配用户拥有权限和请求权限(MyAccessDecisionManager:decide)，如果用户没有相应的权限，执行第6步，否则执行第7步**

**6、登录**

**7、验证并授权(MyUserDetailService:loadUserByUsername)**

**8、重复4,5**

发布，测试页面如下：



输入用户：cyu密码：sap123，然后回车：



点击“进入admin页面”超链，得到如下图所示：



点击“进入其它页面”超链，得到如下图所示：

