主题: Spring Security 3的一些体会

精华帖 (0) :: 良好帖 (0) :: 新手帖 (0) :: 隐藏帖 (0)

作者 | 正文

Jclick
等级: 初级会员

性别:
文章: 50
积分: 70
来自: 北京

发表时间: 2011-09-23 最后修改: 2011-09-23

相关文章:
- Spring源代码解析（九）：Spring Acegi框架鉴权的实现
- spring security 源码解读 1
- 扩展acegi以支持验证码等

推荐群组: JBoss SEAM
更多相关推荐

之前，项目中用到了Spring Security 3，以前没用过的，用过之后，感觉到框架用起来真的很简单，为什么说简单呢？因为只要你看了官方的example之后就可以用，如果遇到不熟悉的也可以翻翻文档解决问题。但是如果为了长远发展，带来的负面效应也是很大的，如果用一个框架不求甚解，只是为了框架而去用框架，后患无穷啊~~因为有可能你只用到框架的一小部分内容，却引入了庞大的框架，如果你想改变一些东西，却发现和框架起了冲突，多么可笑。。。

言归正传，讲讲我的辛酸历程吧~不过也算得上是一种收获。

项目中用了Spring Security 3 （简称SS3）之后，其实项目中并没用到那么详细的权限控制，我感觉Spring Security 3的好处之一就是权限控制可以非常的细化。可以说只是用到了登录的控制。但是现在用到了这么一个需求，就是单点登录，从另外一个系统直接可以登入到这个系统，用户数据可以是同步的，初步的想法就是一个URL加上参数，也就是用户账号密码了，加密出来，然后用来登录。当然安全等级不是很高，这样已经满足了需求，有兴趣了解单点登录的朋友可以去看看淘宝，登入支付宝的时候就是单点登录，很好的一个例子，不过人家的安全等级较高。继续回到问题上来，本来以为这么简单就解决了需求，然后配置之后，但是却发现该怎么登进来呢，我有了账号密码，该怎么直接跳到这个系统？首先想到的就是直接跳转到一个action里，取出用户数据放到Session里（当时还仔细看SS3），太天真了。结果通不过SS3的身份验证，后来仔细看了看，原来有一个过滤器，也就是下边的：

Java代码

```
1.  <s:intercept-url pattern="/user/**" access="isAuthenticated()" />
```

其它的页面基本上都有这个过滤，所以是通不过的，原来有一个接口：

Java代码

```
1.  public interface Authentication extends Principal, Serializable {
2.       Collection<GrantedAuthority> getAuthorities();
3.       Object getCredentials();
4.       Object getDetails();
5.       Object getPrincipal();
6.       boolean isAuthenticated();
7.       void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException;
8.  }
```

存放的是用户经过SS3身份验证后一些信息。而isAuthenticated();指的是是否通过认证。于是我就想，是不是取到Authentication之后改一下isAuthenticated();就可以通过认证了，我试了试不行。想必用过SS3的朋友都知道这个j_spring_security_check，必然会说为什么不直接用这个在后缀上加上用户信息呢，这个就是SS3开放的登录借口，接受用户名密码的，格式都是j_username类型的。不是不想用，关键是不行，为什么不行呢？因为之前我试了在上边加用户名密码就是通

不过认证，我也很纳闷，一样的表单的，后来看了源码发现了原来这个SS3过滤的URL只接受POST请求的。接下来我们一步步解开SS3登录的过程吧。

首先看SS3配置文件中登录的配置吧

Java代码

```
<s:form-login login-page="/login.jsp" default-target-url="/index.jsp" always-use-default-target="true" aut
hentication-failure-url="/login.jsp?error=1" />
```

首先这里边其实有些默认的，比如login-processing-url=""这个参数其实就参数内容就是默认j_spring_security_check，这样的话，SS3就会自动给他一拦截器，当连接是j_spring_security_check的时候，就会调用

Java代码

```
/* Copyright 2004, 2005, 2006 Acegi Technology Pty Limited
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.springframework.security.web.authentication;


import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.security.authentication.AuthenticationServiceException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.util.TextEscapeUtils;
import org.springframework.util.Assert;


/**
 * Processes an authentication form submission. Called {@code AuthenticationProcessingFilter} prior to Spr
ing Security
 * 3.0.
 * <p>
 * Login forms must present two parameters to this filter: a username and
 * password. The default parameter names to use are contained in the
 * static fields {@link #SPRING_SECURITY_FORM_USERNAME_KEY} and {@link #SPRING_SECURITY_FORM_PASSWORD_KEY}.

 * The parameter names can also be changed by setting the {@code usernameParameter} and {@code passwordPar
ameter}
 * properties.
 * <p>
 * This filter by default responds to the URL {@code /j_spring_security_check}.
 *
 * @author Ben Alex
 * @author Colin Sampaleanu
 * @author Luke Taylor
 * @since 3.0
 */
public class UsernamePasswordAuthenticationFilter extends AbstractAuthenticationProcessingFilter {
    //~ Static fields/initializers =====================================================================
=============
```

```
50.
51.     public static final String SPRING_SECURITY_FORM_USERNAME_KEY = "j_username";
52.     public static final String SPRING_SECURITY_FORM_PASSWORD_KEY = "j_password";
53.     public static final String SPRING_SECURITY_LAST_USERNAME_KEY = "SPRING_SECURITY_LAST_USERNAME";
54.
55.     private String usernameParameter = SPRING_SECURITY_FORM_USERNAME_KEY;
56.     private String passwordParameter = SPRING_SECURITY_FORM_PASSWORD_KEY;
57.     private boolean postOnly = true;
58.
59.     //~ Constructors ===================================================================================
=============
60.
61.     public UsernamePasswordAuthenticationFilter() {
62.         super("/j_spring_security_check");
63.     }
64.
65.     //~ Methods ========================================================================================
=============
66.
67.     public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) t
hrows AuthenticationException {
68.         if (postOnly && !request.getMethod().equals("POST")) {
69.             throw new AuthenticationServiceException("Authentication method not supported: " + request.get
Method());
70.         }
71.
72.         String username = obtainUsername(request);
73.         String password = obtainPassword(request);
74.
75.         if (username == null) {
76.             username = "";
77.         }
78.
79.         if (password == null) {
80.             password = "";
81.         }
82.
83.         username = username.trim();
84.
85.         UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(username,
 password);
86.
87.         // Place the last username attempted into HttpSession for views
88.         HttpSession session = request.getSession(false);
89.
90.         if (session != null || getAllowSessionCreation()) {
91.             request.getSession().setAttribute(SPRING_SECURITY_LAST_USERNAME_KEY, TextEscapeUtils.escapeEnti
ties(username));
92.         }
93.
94.         // Allow subclasses to set the "details" property
95.         setDetails(request, authRequest);
96.
97.         return this.getAuthenticationManager().authenticate(authRequest);
98.     }
99.
100.     /**
101.      * Enables subclasses to override the composition of the password, such as by including additional val
ues
102.      * and a separator.<p>This might be used for example if a postcode/zipcode was required in addition t
o the
103.      * password. A delimiter such as a pipe (|) should be used to separate the password and extended valu
e(s). The
104.      * <code>AuthenticationDao</code> will need to generate the expected password in a corresponding manne
r.</p>
105.      *
106.      * @param request so that request attributes can be retrieved
107.      *
```

```
108.      * @return the password that will be presented in the <code>Authentication</code> request token to the
109.      *         <code>AuthenticationManager</code>
110.      */
111.     protected String obtainPassword(HttpServletRequest request) {
112.         return request.getParameter(passwordParameter);
113.     }
114.
115.     /**
116.      * Enables subclasses to override the composition of the username, such as by including additional val
ues
117.      * and a separator.
118.      *
119.      * @param request so that request attributes can be retrieved
120.      *
121.      * @return the username that will be presented in the <code>Authentication</code> request token to the
122.      *         <code>AuthenticationManager</code>
123.      */
124.     protected String obtainUsername(HttpServletRequest request) {
125.         return request.getParameter(usernameParameter);
126.     }
127.
128.     /**
129.      * Provided so that subclasses may configure what is put into the authentication request's details
130.      * property.
131.      *
132.      * @param request that an authentication request is being created for
133.      * @param authRequest the authentication request object that should have its details set
134.      */
135.     protected void setDetails(HttpServletRequest request, UsernamePasswordAuthenticationToken authRequest)
{
136.         authRequest.setDetails(authenticationDetailsSource.buildDetails(request));
137.     }
138.
139.     /**
140.      * Sets the parameter name which will be used to obtain the username from the login request.
141.      *
142.      * @param usernameParameter the parameter name. Defaults to "j_username".
143.      */
144.     public void setUsernameParameter(String usernameParameter) {
145.         Assert.hasText(usernameParameter, "Username parameter must not be empty or null");
146.         this.usernameParameter = usernameParameter;
147.     }
148.
149.     /**
150.      * Sets the parameter name which will be used to obtain the password from the login request..
151.      *
152.      * @param passwordParameter the parameter name. Defaults to "j_password".
153.      */
154.     public void setPasswordParameter(String passwordParameter) {
155.         Assert.hasText(passwordParameter, "Password parameter must not be empty or null");
156.         this.passwordParameter = passwordParameter;
157.     }
158.
159.     /**
160.      * Defines whether only HTTP POST requests will be allowed by this filter.
161.      * If set to true, and an authentication request is received which is not a POST request, an exceptio
n will
162.      * be raised immediately and authentication will not be attempted. The <tt>unsuccessfulAuthentication(
)</tt> method
163.      * will be called as if handling a failed authentication.
164.      * <p>
165.      * Defaults to <tt>true</tt> but may be overridden by subclasses.
166.      */
167.     public void setPostOnly(boolean postOnly) {
168.         this.postOnly = postOnly;
169.     }
```

```
170.
171.     public final String getUsernameParameter() {
172.         return usernameParameter;
173.     }
174.
175.     public final String getPasswordParameter() {
176.         return passwordParameter;
177.     }
178. }
```

在attemptAuthentication(HttpServletRequest request, HttpServletResponse response)中我们可以看到，此方法只接受POST请求的，这个方法的主要作用就是取出用户名密码，然后根据SS3配置文件中的authenticationManager获取用户的信息，具体细节不在赘述，相信用过的朋友都知道他取的过程。

Java代码

```
1.  <s:authentication-manager alias="authenticationManager">
2.                  <s:authentication-provider user-service-ref="userDetailsService">
3.                          <s:password-encoder hash="plaintext" />
4.                  </s:authentication-provider>
5.          </s:authentication-manager>
```

而这个类的父类就是控制登录是否成功或失败的AbstractAuthenticationProcessingFilter。他主要控制登录是否成功并且控制成功后的跳转URL。下边是这个类的过滤器过滤内容:

Java代码

```
1.  public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
2.            throws IOException, ServletException {
3.
4.        HttpServletRequest request = (HttpServletRequest) req;
5.        HttpServletResponse response = (HttpServletResponse) res;
6.
7.        if (!requiresAuthentication(request, response)) {
8.            chain.doFilter(request, response);
9.
10.           return;
11.       }
12.
13.       if (logger.isDebugEnabled()) {
14.           logger.debug("Request is to process authentication");
15.       }
16.
17.       Authentication authResult;
18.
19.       try {
20.           authResult = attemptAuthentication(request, response);
21.           if (authResult == null) {
22.               // return immediately as subclass has indicated that it hasn't completed authentication
23.               return;
24.           }
25.           sessionStrategy.onAuthentication(authResult, request, response);
26.       }
27.       catch (AuthenticationException failed) {
28.           // Authentication failed
29.           unsuccessfulAuthentication(request, response, failed);
30.
31.           return;
32.       }
33.
34.       // Authentication success
35.       if (continueChainBeforeSuccessfulAuthentication) {
36.           chain.doFilter(request, response);
37.       }
38.
39.       successfulAuthentication(request, response, authResult);
```

```
40.     }
```

我们可以看到它取出了用户的数据然后放在了Authentication中了。如果成功则调用 successfulAuthentication(request, response, authResult);这个方法，失败则调用unsuccessfulAuthentication(request, response, failed);

不再过多的贴代码了，接着说的就是successfulAuthentication(request, response, authResult);这个方法主要做的就是首先查看是否勾选了记住密码选项然后跳转页面也就是目标页，当然其中还有很多机制，比如没条用一个URL就会经国LogoutFilter判断URL了，比如注入了很多Bean，再比如用了大量的回调方法，比如对角色权限等等的细分。很多很多。但是只看了两三天的源码，也只是懂了点SS3的基本原理以及一些内部的实现机制，还差好多。以后继续再看看，再跟大家分享咯。。

最后想说的就是框架其实只是用来快速开发用的，很多框架基本上都封装了实现的细节，我们只知道怎么用是远远不够的，那样干的只是体力活，更多的我们应该去知道它为什么这么用，怎么实现的，然后我们可以不用框架，也可以写出很好的代码。其实说白了，我们最需要掌握的就是解决问题的能里，能不是去跟别人说我会用什么什么框架之类的！

最后送给大家的是很不错Spring security 3 学习文档~~

Spring_Security_3.rar (2.1 MB)
下载次数: 85

返回顶楼 主页 资料 短信 留言 关注

uncle_bacon
等级:

发表时间：3 小时前

这个是蛮好的，相互学习

性别:
文章: 26
积分: 110
来自: 上海
我现在离线

返回顶楼 主页 资料 短信 留言 关注 回帖地址 0 0 请登录后投票

george_space
等级: 初级会员

发表时间：38 分钟前

我感觉Spring Security 3还是不太满意的，一开始我使用Spring Security 3，费了很大功夫才实现了权限定义从数据库读取、多页

性别:
文章: 235
积分: 10
来自: 上海

我现在离线

面登陆、验证码、附加登录信息、自定义超时提示文本.....等等一系列功能，后来就一直用它，然后突然有一天发现，Spring Security 3保护的url竟然失效了，也不知道怎么搞得，直接输入受保护的url就可以进入后台，算了，不用它了。

后来又换了apache shiro，用过一段时间后，也感觉很不爽，所以就干脆自己实现了一个简单的权限管理模块，用到现在，还没有出现莫名url保护不起作用的事。

返回顶楼 主页 资料 短信 留言 关注 回帖地址 0 0 请登录后投票

发表回复

论坛首页 → Java编程和Java企业应用版 → Spring

跳转论坛: Java编程和Java企业应用