[**http://blog.csdn.net/k10509806/article/details/6369131**](http://blog.csdn.net/k10509806/article/details/6369131)

**一、数据库结构**

先来看一下数据库结构，采用的是基于角色-资源-用户的权限管理设计。(MySql数据库)

    为了节省篇章，只对比较重要的字段进行注释。

    1.用户表Users

    CREATE TABLE `users` (

       -- 账号是否有限 1. 是 0.否  
       `enable` int(11) default NULL,  
       `password` varchar(255) default NULL,  
       `account` varchar(255) default NULL,  
       `id` int(11) NOT NULL auto\_increment,  
       PRIMARY KEY  (`id`)  
    )

   2.角色表Roles

   CREATE TABLE `roles` (  
     `enable` int(11) default NULL,  
     `name` varchar(255) default NULL,  
     `id` int(11) NOT NULL auto\_increment,  
     PRIMARY KEY  (`id`)  
   )

   3 用户\_角色表users\_roles

   CREATE TABLE `users\_roles` (

     --用户表的外键  
     `uid` int(11) default NULL,

     --角色表的外键  
     `rid` int(11) default NULL,  
     `urId` int(11) NOT NULL auto\_increment,  
     PRIMARY KEY  (`urId`),  
     KEY `rid` (`rid`),  
     KEY `uid` (`uid`),  
    CONSTRAINT `users\_roles\_ibfk\_1` FOREIGN KEY (`rid`) REFERENCES `roles` (`id`),  
    CONSTRAINT `users\_roles\_ibfk\_2` FOREIGN KEY (`uid`) REFERENCES `users` (`id`)  
   )

   4.资源表resources

   CREATE TABLE `resources` (  
     `memo` varchar(255) default NULL,

     -- 权限所对应的url地址  
     `url` varchar(255) default NULL,

     --优先权  
     `priority` int(11) default NULL,

     --类型  
     `type` int(11) default NULL,

     --权限所对应的编码，例201代表发表文章  
     `name` varchar(255) default NULL,  
     `id` int(11) NOT NULL auto\_increment,  
     PRIMARY KEY  (`id`)  
   )

   5.角色\_资源表roles\_resources

    CREATE TABLE `roles\_resources` (  
      `rsid` int(11) default NULL,  
      `rid` int(11) default NULL,  
      `rrId` int(11) NOT NULL auto\_increment,  
      PRIMARY KEY  (`rrId`),  
      KEY `rid` (`rid`),  
      KEY `roles\_resources\_ibfk\_2` (`rsid`),  
      CONSTRAINT `roles\_resources\_ibfk\_2` FOREIGN KEY (`rsid`) REFERENCES `resources` (`id`),  
      CONSTRAINT `roles\_resources\_ibfk\_1` FOREIGN KEY (`rid`) REFERENCES `roles` (`id`)  
      )

**二、系统配置**

所需要的jar包，请自行到官网下载，我用的是Spring Security3.1.0.RC1版的。把dist下的除了源码件包导入就行了。还有那些零零碎的   数据库驱动啊，log4j.jar等等，我相信在用Spring Security之前，大家已经会的了。

  1） web.xml

1. <!-- Spring -->
2. **<context-param>**
3. **<param-name>**contextConfigLocation**</param-name>**
4. **<param-value>**classpath:applicationContext.xml,classpath:applicationContext-security.xml**</param-value>**
5. **</context-param>**

8. **<listener>**
9. **<listener-class>**org.springframework.web.context.ContextLoaderListener**</listener-class>**
10. **</listener>**
11. <!-- 权限 -->
12. **<filter>**
13. **<filter-name>**springSecurityFilterChain**</filter-name>**
14. **<filter-class>**
15. org.springframework.web.filter.DelegatingFilterProxy
16. **</filter-class>**
17. **</filter>**
18. **<filter-mapping>**
19. **<filter-name>**springSecurityFilterChain**</filter-name>**
20. **<url-pattern>**/\***</url-pattern>**
21. **</filter-mapping>**

 这里主要是配置了让容器启动的时候加载application-security.xml和Spring Security的权限过滤器代理,让其过滤所有的客服请求。

 2）application-security.xml

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans:beans** xmlns="http://www.springframework.org/schema/security"
3. xmlns:beans="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
6. http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.1.xsd"**>**
8. **<global-method-security** pre-post-annotations="enabled" **/>**
9. <!-- 该路径下的资源不用过滤 -->
10. **<http** pattern="/js/\*\*" security="none"**/>**
11. **<http** use-expressions="true" auto-config="true"**>**
13. **<form-login** **/>**
14. **<logout/>**
15. <!-- 实现免登陆验证 -->
16. **<remember-me** **/>**
17. **<session-management** invalid-session-url="/timeout.jsp"**>**
18. **<concurrency-control** max-sessions="10" error-if-maximum-exceeded="true" **/>**
19. **</session-management>**
20. **<custom-filter** ref="myFilter" before="FILTER\_SECURITY\_INTERCEPTOR"**/>**
21. **</http>**
22. <!-- 配置过滤器 -->
23. **<beans:bean** id="myFilter" class="com.huaxin.security.MySecurityFilter"**>**
24. <!-- 用户拥有的权限 -->
25. **<beans:property** name="authenticationManager" ref="myAuthenticationManager" **/>**
26. <!-- 用户是否拥有所请求资源的权限 -->
27. **<beans:property** name="accessDecisionManager" ref="myAccessDecisionManager" **/>**
28. <!-- 资源与权限对应关系 -->
29. **<beans:property** name="securityMetadataSource" ref="mySecurityMetadataSource" **/>**
30. **</beans:bean>**
31. <!-- 实现了UserDetailsService的Bean -->
32. **<authentication-manager** alias="myAuthenticationManager"**>**
33. **<authentication-provider** user-service-ref="myUserDetailServiceImpl" **/>**
34. **</authentication-manager>**
35. **<beans:bean** id="myAccessDecisionManager" class="com.huaxin.security.MyAccessDecisionManager"**></beans:bean>**
36. **<beans:bean** id="mySecurityMetadataSource" class="com.huaxin.security.MySecurityMetadataSource"**>**
37. **<beans:constructor-arg** name="resourcesDao" ref="resourcesDao"**></beans:constructor-arg>**
38. **</beans:bean>**
39. **<beans:bean** id="myUserDetailServiceImpl" class="com.huaxin.security.MyUserDetailServiceImpl"**>**
40. **<beans:property** name="usersDao" ref="usersDao"**></beans:property>**
41. **</beans:bean>**
42. **</beans:beans>**

我们在第二个http标签下配置一个我们自定义的继承了org.springframework.security.access.intercept.AbstractSecurityInterceptor的Filter，并注入其

必须的3个组件authenticationManager、accessDecisionManager和securityMetadataSource。其作用上面已经注释了。

<custom-filter ref="myFilter" before="FILTER\_SECURITY\_INTERCEPTOR"/> 这里的FILTER\_SECURITY\_INTERCEPTOR是Spring Security默认的Filter，

我们自定义的Filter必须在它之前，过滤客服请求。接下来看下我们最主要的myFilter吧。

3）myFilter

  (1) MySecurityFilter.java 过滤用户请求

**[java]** [view plaincopyprint?](http://blog.csdn.net/k10509806/article/details/6369131)

1. public class MySecurityFilter extends AbstractSecurityInterceptor implements Filter {
2. //与applicationContext-security.xml里的myFilter的属性securityMetadataSource对应，
3. //其他的两个组件，已经在AbstractSecurityInterceptor定义
4. private FilterInvocationSecurityMetadataSource securityMetadataSource;
6. @Override
7. public SecurityMetadataSource obtainSecurityMetadataSource() {
8. return this.securityMetadataSource;
9. }
11. public void doFilter(ServletRequest request, ServletResponse response,
12. FilterChain chain) throws IOException, ServletException {
13. FilterInvocation fi = new FilterInvocation(request, response, chain);
14. invoke(fi);
15. }
17. private void invoke(FilterInvocation fi) throws IOException, ServletException {
18. // object为FilterInvocation对象
19. //super.beforeInvocation(fi);源码
20. //1.获取请求资源的权限
21. //执行Collection<ConfigAttribute> attributes = SecurityMetadataSource.getAttributes(object);
22. //2.是否拥有权限
23. //this.accessDecisionManager.decide(authenticated, object, attributes);
24. InterceptorStatusToken token = super.beforeInvocation(fi);
25. try {
26. fi.getChain().doFilter(fi.getRequest(), fi.getResponse());
27. } finally {
28. super.afterInvocation(token, null);
29. }
30. }
32. public FilterInvocationSecurityMetadataSource getSecurityMetadataSource() {
33. return securityMetadataSource;
34. }
36. public void setSecurityMetadataSource(FilterInvocationSecurityMetadataSource securityMetadataSource) {
37. this.securityMetadataSource = securityMetadataSource;
38. }
40. public void init(FilterConfig arg0) throws ServletException {
41. // TODO Auto-generated method stub
42. }
44. public void destroy() {
45. // TODO Auto-generated method stub
47. }
49. @Override
50. public Class<? extends Object> getSecureObjectClass() {
51. //下面的MyAccessDecisionManager的supports方面必须放回true,否则会提醒类型错误
52. return FilterInvocation.class;
53. }
54. }

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

  :getAttributes(Object object)方法。

 （2）MySecurityMetadataSource.java

**[java]** [view plaincopyprint?](http://blog.csdn.net/k10509806/article/details/6369131)

1. //1 加载资源与权限的对应关系
2. public class MySecurityMetadataSource implements FilterInvocationSecurityMetadataSource {
3. //由spring调用
4. public MySecurityMetadataSource(ResourcesDao resourcesDao) {
5. this.resourcesDao = resourcesDao;
6. loadResourceDefine();
7. }
9. private ResourcesDao resourcesDao;
10. private static Map<String, Collection<ConfigAttribute>> resourceMap = null;
12. public ResourcesDao getResourcesDao() {
13. return resourcesDao;
14. }
16. public void setResourcesDao(ResourcesDao resourcesDao) {
17. this.resourcesDao = resourcesDao;
18. }
20. public Collection<ConfigAttribute> getAllConfigAttributes() {
21. // TODO Auto-generated method stub
22. return null;
23. }
25. public boolean supports(Class<?> clazz) {
26. // TODO Auto-generated method stub
27. return true;
28. }
29. //加载所有资源与权限的关系
30. private void loadResourceDefine() {
31. if(resourceMap == null) {
32. resourceMap = new HashMap<String, Collection<ConfigAttribute>>();
33. List<Resources> resources = this.resourcesDao.findAll();
34. for (Resources resource : resources) {
35. Collection<ConfigAttribute> configAttributes = new ArrayList<ConfigAttribute>();
36. //以权限名封装为Spring的security Object
37. ConfigAttribute configAttribute = new SecurityConfig(resource.getName());
38. configAttributes.add(configAttribute);
39. resourceMap.put(resource.getUrl(), configAttributes);
40. }
41. }
43. Set<Entry<String, Collection<ConfigAttribute>>> resourceSet = resourceMap.entrySet();
44. Iterator<Entry<String, Collection<ConfigAttribute>>> iterator = resourceSet.iterator();
46. }
47. //返回所请求资源所需要的权限
48. public Collection<ConfigAttribute> getAttributes(Object object) throws IllegalArgumentException {
50. String requestUrl = ((FilterInvocation) object).getRequestUrl();
51. System.out.println("requestUrl is " + requestUrl);
52. if(resourceMap == null) {
53. loadResourceDefine();
54. }
55. return resourceMap.get(requestUrl);
56. }
58. }

 这里的resourcesDao，熟悉Dao设计模式和Spring 注入的朋友应该看得明白。

（3）MyUserDetailServiceImpl.java

**[java]** [view plaincopyprint?](http://blog.csdn.net/k10509806/article/details/6369131)

1. public class MyUserDetailServiceImpl implements UserDetailsService {
3. private UsersDao usersDao;
4. public UsersDao getUsersDao() {
5. return usersDao;
6. }
8. public void setUsersDao(UsersDao usersDao) {
9. this.usersDao = usersDao;
10. }
12. public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
13. System.out.println("username is " + username);
14. Users users = this.usersDao.findByName(username);
15. if(users == null) {
16. throw new UsernameNotFoundException(username);
17. }
18. Collection<GrantedAuthority> grantedAuths = obtionGrantedAuthorities(users);
20. boolean enables = true;
21. boolean accountNonExpired = true;
22. boolean credentialsNonExpired = true;
23. boolean accountNonLocked = true;
25. User userdetail = new User(users.getAccount(), users.getPassword(), enables, accountNonExpired, credentialsNonExpired, accountNonLocked, grantedAuths);
26. return userdetail;
27. }
29. //取得用户的权限
30. private Set<GrantedAuthority> obtionGrantedAuthorities(Users user) {
31. Set<GrantedAuthority> authSet = new HashSet<GrantedAuthority>();
32. Set<Roles> roles = user.getRoles();
34. for(Roles role : roles) {
35. Set<Resources> tempRes = role.getResources();
36. for(Resources res : tempRes) {
37. authSet.add(new GrantedAuthorityImpl(res.getName()));
38. s           }
39. }
40. return authSet;
41. }
42. }

(4) MyAccessDecisionManager.java

**[java]** [view plaincopyprint?](http://blog.csdn.net/k10509806/article/details/6369131)

1. public class MyAccessDecisionManager implements AccessDecisionManager {
3. public void decide(Authentication authentication, Object object, Collection<ConfigAttribute> configAttributes) throws AccessDeniedException, InsufficientAuthenticationException {
4. if(configAttributes == null) {
5. return;
6. }
7. //所请求的资源拥有的权限(一个资源对多个权限)
8. Iterator<ConfigAttribute> iterator = configAttributes.iterator();
9. while(iterator.hasNext()) {
10. ConfigAttribute configAttribute = iterator.next();
11. //访问所请求资源所需要的权限
12. String needPermission = configAttribute.getAttribute();
13. System.out.println("needPermission is " + needPermission);
14. //用户所拥有的权限authentication
15. for(GrantedAuthority ga : authentication.getAuthorities()) {
16. if(needPermission.equals(ga.getAuthority())) {
17. return;
18. }
19. }
20. }
21. //没有权限
22. throw new AccessDeniedException(" 没有权限访问！ ");
23. }
25. public boolean supports(ConfigAttribute attribute) {
26. // TODO Auto-generated method stub
27. return true;
28. }
30. public boolean supports(Class<?> clazz) {
31. // TODO Auto-generated method stub
32. return true;
33. }
35. }

**三、流程**

 1）容器启动(MySecurityMetadataSource：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(MySecurityFilter:doFilter)  
 4）取得请求资源所需权限(MySecurityMetadataSource: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(MyUserDetailServiceImpl:loadUserByUsername)  
 8）重复4,5

**四、结束语**

好了，终于写完了，回头看了一下，感觉不是怎么行。等我弄明白Spring Security它的原理之后，再回头修改下注释吧。大家觉得不妥的地方，可以留言，我会回复大家的。

我已经把源码上传到CSDN了。<http://download.csdn.net/source/3283687>

一、前言

      在上一篇<http://blog.csdn.net/k10509806/archive/2011/04/28/6369131.aspx>文章中，提到的MyUserDetailServiceImpl获取用户权限，在用户没有登陆的时候，Spring Security会让我们自动跳转到默认的登陆界面，但在实际应用绝大多数是用我们自己的登陆界面的，其中就包括一些我们自己的逻辑，比如验证码。所以本人又研究一下，终于摸清了一些如何配置自己的登陆界面的办法。在这里献丑了。

二、Spring Security的过滤器

      通过DEBUG可以看到Spring Security的Filter的顺序

Security filter chain: [  
  ConcurrentSessionFilter  
  SecurityContextPersistenceFilter  
  LogoutFilter  
  MyUsernamePasswordAuthenticationFilter  
  RequestCacheAwareFilter  
  SecurityContextHolderAwareRequestFilter  
  RememberMeAuthenticationFilter  
  AnonymousAuthenticationFilter  
  SessionManagementFilter  
  ExceptionTranslationFilter  
  MySecurityFilter  
  FilterSecurityInterceptor  
]

Spring Security的登陆验证用的就是MyUsernamePasswordAuthenticationFilter，所以要实现我们自己的验证，可以写一个类并继承MyUsernamePasswordAuthenticationFilter类，重写attemptAuthentication方法。

三、applicationContext-Security.xml配置

**[xhtml]** [view plaincopyprint?](http://blog.csdn.net/k10509806/article/details/6436987)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans:beans** xmlns="http://www.springframework.org/schema/security"
3. xmlns:beans="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
6. http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.1.xsd"**>**
8. **<debug/>**
9. **<http** pattern="/js/\*\*" security="none"**/>**
10. **<http** pattern="/resource/\*\*" security="none"**></http>**
11. **<http** pattern="/login.jsp" security="none"**/>**
13. **<http** use-expressions="true" entry-point-ref="authenticationProcessingFilterEntryPoint"**>**
14. **<logout/>**
15. <!-- 实现免登陆验证 -->
16. **<remember-me** **/>**
17. **<session-management** invalid-session-url="/timeout.jsp"**>**
18. **<concurrency-control** max-sessions="10" error-if-maximum-exceeded="true" **/>**
19. **</session-management>**
21. **<custom-filter** ref="loginFilter" position="FORM\_LOGIN\_FILTER"  **/>**
22. **<custom-filter** ref="securityFilter" before="FILTER\_SECURITY\_INTERCEPTOR"**/>**
23. **</http>**
25. <!-- 登录验证器 -->
26. **<beans:bean** id="loginFilter"
27. class="com.huaxin.security.MyUsernamePasswordAuthenticationFilter"**>**
28. <!-- 处理登录的action -->
29. **<beans:property** name="filterProcessesUrl" value="/j\_spring\_security\_check"**></beans:property>**
30. <!-- 验证成功后的处理-->
31. **<beans:property** name="authenticationSuccessHandler" ref="loginLogAuthenticationSuccessHandler"**></beans:property>**
32. <!-- 验证失败后的处理-->
33. **<beans:property** name="authenticationFailureHandler" ref="simpleUrlAuthenticationFailureHandler"**></beans:property>**
34. **<beans:property** name="authenticationManager" ref="myAuthenticationManager"**></beans:property>**
35. <!-- 注入DAO为了查询相应的用户 -->
36. **<beans:property** name="usersDao" ref="usersDao"**></beans:property>**
37. **</beans:bean>**
38. **<beans:bean** id="loginLogAuthenticationSuccessHandler"
39. class="org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler"**>**
40. **<beans:property** name="defaultTargetUrl" value="/index.jsp"**></beans:property>**
41. **</beans:bean>**
42. **<beans:bean** id="simpleUrlAuthenticationFailureHandler"
43. class="org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHandler"**>**
44. <!-- 可以配置相应的跳转方式。属性forwardToDestination为true采用forward false为sendRedirect -->
45. **<beans:property** name="defaultFailureUrl" value="/login.jsp"**></beans:property>**
46. **</beans:bean>**
48. <!-- 认证过滤器 -->
49. **<beans:bean** id="securityFilter" class="com.huaxin.security.MySecurityFilter"**>**
50. <!-- 用户拥有的权限 -->
51. **<beans:property** name="authenticationManager" ref="myAuthenticationManager" **/>**
52. <!-- 用户是否拥有所请求资源的权限 -->
53. **<beans:property** name="accessDecisionManager" ref="myAccessDecisionManager" **/>**
54. <!-- 资源与权限对应关系 -->
55. **<beans:property** name="securityMetadataSource" ref="mySecurityMetadataSource" **/>**
56. **</beans:bean>**
57. <!-- 实现了UserDetailsService的Bean -->
58. **<authentication-manager** alias="myAuthenticationManager"**>**
59. **<authentication-provider** user-service-ref="myUserDetailServiceImpl" **/>**
60. **</authentication-manager>**
62. **<beans:bean** id="myAccessDecisionManager" class="com.huaxin.security.MyAccessDecisionManager"**></beans:bean>**
63. **<beans:bean** id="mySecurityMetadataSource" class="com.huaxin.security.MySecurityMetadataSource"**>**
64. **<beans:constructor-arg** name="resourcesDao" ref="resourcesDao"**></beans:constructor-arg>**
65. **</beans:bean>**
66. **<beans:bean** id="myUserDetailServiceImpl" class="com.huaxin.security.MyUserDetailServiceImpl"**>**
67. **<beans:property** name="usersDao" ref="usersDao"**></beans:property>**
68. **</beans:bean>**
70. <!-- 未登录的切入点 -->
71. **<beans:bean** id="authenticationProcessingFilterEntryPoint" class="org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint"**>**
72. **<beans:property** name="loginFormUrl" value="/login.jsp"**></beans:property>**
73. **</beans:bean>**
74. **</beans:beans>**

这里特别要说明一下，我们的<http>标签不能配置auto-config，因为这样配置后，依然会采用Spring Security的Filter Chain会与下面我们配的custom-filter冲突，最好会抛异常。还有配置一个切入点entry-point-ref="authenticationProcessingFilterEntryPoint"，为了在未登陆的时候，跳转到哪个页面，不配也会抛异常。

 <custom-filter ref="loginFilter" position="FORM\_LOGIN\_FILTER"  /> position表示替换掉Spring Security原来默认的登陆验证Filter。

四、MyUsernamePasswordAuthenticationFilter

**[java]** [view plaincopyprint?](http://blog.csdn.net/k10509806/article/details/6436987)

1. package com.huaxin.security;
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import javax.servlet.http.HttpSession;
7. import org.apache.commons.lang.xwork.StringUtils;
8. import org.springframework.security.authentication.AuthenticationServiceException;
9. import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
10. import org.springframework.security.core.Authentication;
11. import org.springframework.security.core.AuthenticationException;
12. import org.springframework.security.web.WebAttributes;
13. import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
15. import com.huaxin.bean.Users;
16. import com.huaxin.dao.UsersDao;
18. /\*
19. \*
20. \* UsernamePasswordAuthenticationFilter源码
21. attemptAuthentication
22. this.getAuthenticationManager()
23. ProviderManager.java
24. authenticate(UsernamePasswordAuthenticationToken authRequest)
25. AbstractUserDetailsAuthenticationProvider.java
26. authenticate(Authentication authentication)
27. P155 user = retrieveUser(username, (UsernamePasswordAuthenticationToken) authentication);
28. DaoAuthenticationProvider.java
29. P86 loadUserByUsername
30. \*/
31. public class MyUsernamePasswordAuthenticationFilter extends UsernamePasswordAuthenticationFilter{
32. public static final String VALIDATE\_CODE = "validateCode";
33. public static final String USERNAME = "username";
34. public static final String PASSWORD = "password";
36. private UsersDao usersDao;
37. public UsersDao getUsersDao() {
38. return usersDao;
39. }
40. public void setUsersDao(UsersDao usersDao) {
41. this.usersDao = usersDao;
42. }
44. @Override
45. public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException {
46. if (!request.getMethod().equals("POST")) {
47. throw new AuthenticationServiceException("Authentication method not supported: " + request.getMethod());
48. }
49. //检测验证码
50. checkValidateCode(request);
52. String username = obtainUsername(request);
53. String password = obtainPassword(request);
55. //验证用户账号与密码是否对应
56. username = username.trim();
58. Users users = this.usersDao.findByName(username);
60. if(users == null || !users.getPassword().equals(password)) {
61. /\*
62. 在我们配置的simpleUrlAuthenticationFailureHandler处理登录失败的处理类在这么一段
63. 这样我们可以在登录失败后，向用户提供相应的信息。
64. if (forwardToDestination) {
65. request.setAttribute(WebAttributes.AUTHENTICATION\_EXCEPTION, exception);
66. } else {
67. HttpSession session = request.getSession(false);
69. if (session != null || allowSessionCreation) {
70. request.getSession().setAttribute(WebAttributes.AUTHENTICATION\_EXCEPTION, exception);
71. }
72. }
73. \*/
74. throw new AuthenticationServiceException("用户名或者密码错误！");
75. }
77. //UsernamePasswordAuthenticationToken实现 Authentication
78. UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(username, password);
79. // Place the last username attempted into HttpSession for views
81. // 允许子类设置详细属性
82. setDetails(request, authRequest);
84. // 运行UserDetailsService的loadUserByUsername 再次封装Authentication
85. return this.getAuthenticationManager().authenticate(authRequest);
86. }
88. protected void checkValidateCode(HttpServletRequest request) {
89. HttpSession session = request.getSession();
91. String sessionValidateCode = obtainSessionValidateCode(session);
92. //让上一次的验证码失效
93. session.setAttribute(VALIDATE\_CODE, null);
94. String validateCodeParameter = obtainValidateCodeParameter(request);
95. if (StringUtils.isEmpty(validateCodeParameter) || !sessionValidateCode.equalsIgnoreCase(validateCodeParameter)) {
96. throw new AuthenticationServiceException("验证码错误！");
97. }
98. }
100. private String obtainValidateCodeParameter(HttpServletRequest request) {
101. Object obj = request.getParameter(VALIDATE\_CODE);
102. return null == obj ? "" : obj.toString();
103. }
105. protected String obtainSessionValidateCode(HttpSession session) {
106. Object obj = session.getAttribute(VALIDATE\_CODE);
107. return null == obj ? "" : obj.toString();
108. }
110. @Override
111. protected String obtainUsername(HttpServletRequest request) {
112. Object obj = request.getParameter(USERNAME);
113. return null == obj ? "" : obj.toString();
114. }
116. @Override
117. protected String obtainPassword(HttpServletRequest request) {
118. Object obj = request.getParameter(PASSWORD);
119. return null == obj ? "" : obj.toString();
120. }

123. }

五、login.jsp

**[php]** [view plaincopyprint?](http://blog.csdn.net/k10509806/article/details/6436987)

1. <body>
2. <span style="color:red"><%=session.getAttribute(WebAttributes.AUTHENTICATION\_EXCEPTION) %></span>
3. <form action="j\_spring\_security\_check" method="post">
4. Account：<Input name="username"/><br/>
5. Password：<input name="password" type="password"/><br/>
6. <input value="submit" type="submit"/>
7. </form>
8. </body>