

springboot整合shiro-对密码进行MD5并加盐处理(十五)

原文地址，转载请注明出处：https://blog.csdn.net/qq_34021712/article/details/84571067 ©王赛超

数据库中密码相关字段都不是明文,肯定是 加密 之后的，传统方式一般是使用MD5加密。

单纯使用不加盐的MD5 加密方式，当两个用户的密码相同时，会发现数据库中存在相同内容的密码，这样也是不安全的。我们希望即便是两个人的原始密码一样，加密后的结果也不一样。

下面进行 shiro 密码 加密加盐配置：

1.ShiroConfig中添加密码比较器

```
1  /**
2   * 配置密码比较器
3   * @return
4   */
5  @Bean("credentialsMatcher")
6  public RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher(){
7      RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher = new RetryLimitHashedCredentialsMatcher();
8      retryLimitHashedCredentialsMatcher.setRedisManager(redisManager());
9
10     // 如果密码加密,可以打开下面配置
11     // 加密算法的名称
12     retryLimitHashedCredentialsMatcher.setHashAlgorithmName("MD5");
13     // 配置加密的次数
14     retryLimitHashedCredentialsMatcher.setHashIterations(2);
15     // 是否存储为16进制
16     retryLimitHashedCredentialsMatcher.setStoredCredentialsHexEncoded(true);
17
18     return retryLimitHashedCredentialsMatcher;
19 }
```

2.将密码比较器配置给ShiroRealm

```
1  /**
2   * 身份认证realm; (这个需要自己写, 账号密码校验; 权限等)
3   * @return
4   */
5  @Bean
6  public ShiroRealm shiroRealm(){
7      ShiroRealm shiroRealm = new ShiroRealm();
8      shiroRealm.setCachingEnabled(true);
9      // 启用身份验证缓存, 即缓存AuthenticationInfo信息, 默认false
10     shiroRealm.setAuthenticationCachingEnabled(true);
11     // 缓存AuthenticationInfo信息的缓存名称 在ehcache-shiro.xml中有对应缓存的配置
12     shiroRealm.setAuthenticationCacheName("authenticationCache");
13     // 启用授权缓存, 即缓存AuthorizationInfo信息, 默认false
14     shiroRealm.setAuthorizationCachingEnabled(true);
15     // 缓存AuthorizationInfo信息的缓存名称 在ehcache-shiro.xml中有对应缓存的配置
16     shiroRealm.setAuthorizationCacheName("authorizationCache");
17     // 配置自定义密码比较器
18     shiroRealm.setCredentialsMatcher(retryLimitHashedCredentialsMatcher());
19     return shiroRealm;
20 }
```

3.密码比较器RetryLimitHashedCredentialsMatcher

自定义的密码比较器，跟前面博客中逻辑没有变化,唯一变的是 继承的类从 SimpleCredentialsMatcher 变为 HashedCredentialsMatcher

在密码比较器中做了：如果用户输入密码连续错误5次，将锁定账号,具体参考博客：https://blog.csdn.net/qq_34021712/article/details/80461177

RetryLimitHashedCredentialsMatcher完整内容如下：

```
1  package com.springboot.test.shiro.config.shiro;
2
3  import java.util.concurrent.atomic.AtomicInteger;
4
5  import com.springboot.test.shiro.modules.user.dao.UserMapper;
6  import com.springboot.test.shiro.modules.user.dao.entity.User;
7  import org.apache.log4j.Logger;
8  import org.apache.shiro.authc.AuthenticationInfo;
9  import org.apache.shiro.authc.AuthenticationToken;
10 import org.apache.shiro.authc.LockedAccountException;
11 import org.apache.shiro.shiro.authc.credentials.HashedException;
12 import org.springframework.beans.factory.annotation.Autowired;
```

```

13
14
15 /**
16  * @author: WangSaiChao
17  * @date: 2018/5/25
18  * @description: 登陆次数限制
19  */
20 public class RetryLimitHashedCredentialsMatcher extends HashedCredentialsMatcher {
21
22     private static final Logger logger = Logger.getLogger(RetryLimitHashedCredentialsMatcher.class);
23
24     public static final String DEFAULT_RETRYLIMIT_CACHE_KEY_PREFIX = "shiro:cache:retrylimit:";
25     private String keyPrefix = DEFAULT_RETRYLIMIT_CACHE_KEY_PREFIX;
26     @Autowired
27     private UserMapper userMapper;
28     private RedisManager redisManager;
29
30     public void setRedisManager(RedisManager redisManager) {
31         this.redisManager = redisManager;
32     }
33
34     private String getRedisKickoutKey(String username) {
35         return this.keyPrefix + username;
36     }
37
38     @Override
39     public boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {
40
41         // 获取用户名
42         String username = (String)token.getPrincipal();
43         // 获取用户登录次数
44         AtomicInteger retryCount = (AtomicInteger)redisManager.get(getRedisKickoutKey(username));
45         if (retryCount == null) {
46             // 如果用户没有登陆过, 登陆次数加1 并放入缓存
47             retryCount = new AtomicInteger(0);
48         }
49         if (retryCount.incrementAndGet() > 5) {
50             // 如果用户登陆失败次数大于5次 抛出锁定用户异常 并修改数据库字段
51             User user = userMapper.findByUserName(username);
52             if (user != null && "0".equals(user.getState())){
53                 // 数据库字段 默认为 0 就是正常状态 所以 要改为1
54                 // 修改数据库的状态字段为锁定
55                 user.setState("1");
56                 userMapper.update(user);
57             }
58             logger.info("锁定用户" + user.getUsername());
59             // 抛出用户锁定异常
60             throw new LockedAccountException();
61         }
62         // 判断用户账号和密码是否正确
63         boolean matches = super.doCredentialsMatch(token, info);
64         if (matches) {
65             // 如果正确, 从缓存中将用户登录计数 清除
66             redisManager.del(getRedisKickoutKey(username));
67         }{
68             redisManager.set(getRedisKickoutKey(username), retryCount);
69         }
70         return matches;
71     }
72
73     /**
74     * 根据用户名 解锁用户
75     * @param username
76     * @return
77     */
78     public void unlockAccount(String username){
79         User user = userMapper.findByUserName(username);
80         if (user != null){
81             // 修改数据库的状态字段为锁定
82             user.setState("0");
83             userMapper.update(user);
84             redisManager.del(getRedisKickoutKey(username));
85         }
86     }
87
88 }

```

4.修改ShiroRealm中doGetAuthenticationInfo方法

```
1 package com.springboot.test.shiro.config.shiro;
2
3 import com.springboot.test.shiro.modules.user.dao.PermissionMapper;
4 import com.springboot.test.shiro.modules.user.dao.RoleMapper;
5 import com.springboot.test.shiro.modules.user.dao.entity.Permission;
6 import com.springboot.test.shiro.modules.user.dao.entity.Role;
7 import com.springboot.test.shiro.modules.user.dao.UserMapper;
8 import com.springboot.test.shiro.modules.user.dao.entity.User;
9 import org.apache.shiro.SecurityUtils;
10 import org.apache.shiro.authc.*;
11 import org.apache.shiro.authz.AuthorizationInfo;
12 import org.apache.shiro.authz.SimpleAuthorizationInfo;
13 import org.apache.shiro.realm.AuthorizingRealm;
14 import org.apache.shiro.subject.PrincipalCollection;
15 import org.springframework.beans.factory.annotation.Autowired;
16
17 import java.util.Set;
18
19 /**
20  * @author: wangsaichao
21  * @date: 2018/5/10
22  * @description: 在Shiro中, 最终是通过Realm来获取应用程序中的用户、角色及权限信息的
23  * 在Realm中会直接从我们的数据源中获取Shiro需要的验证信息。可以说, Realm是专用于安全框架的DAO。
24  */
25 public class ShiroRealm extends AuthorizingRealm {
26
27     @Autowired
28     private UserMapper userMapper;
29
30     @Autowired
31     private RoleMapper roleMapper;
32
33     @Autowired
34     private PermissionMapper permissionMapper;
35
36     /**
37      * 验证用户身份
38      * @param authenticationToken
39      * @return
40      * @throws AuthenticationException
41      */
42     @Override
43     protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {
44
45         // 获取用户名密码 第一种方式
46         //String username = (String) authenticationToken.getPrincipal();
47         //String password = new String((char[]) authenticationToken.getCredentials());
48
49         // 获取用户名 密码 第二种方式
50         UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) authenticationToken;
51         String username = usernamePasswordToken.getUsername();
52         String password = new String(usernamePasswordToken.getPassword());
53
54         // 从数据库查询用户信息
55         User user = this.userMapper.findByUserName(username);
56
57         // 可以在这里直接对用户名校验, 或者调用 CredentialsMatcher 校验
58         if (user == null) {
59             throw new UnknownAccountException("用户名或密码错误! ");
60         }
61         // 这里将 密码对比 注销掉, 否则 无法锁定 要将密码对比 交给 密码比较器
62         //if (!password.equals(user.getPassword())) {
63         //    throw new IncorrectCredentialsException("用户名或密码错误! ");
64         //}
65         if ("1".equals(user.getState())) {
66             throw new LockedAccountException("账号已被锁定, 请联系管理员! ");
67         }
68
69         SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(), new MyByteSource(user.getUsername()), get
70         return info;
71     }
72
73     /**
74      * 授权用户权限
75      * 授权的方法是在碰到<shiro:hasPermission name=' '></shiro:hasPermission>标签的时候调用的
76      * 它会去检测shiro框架中的权限(这里的permissions)是否包含有该标签的name值, 如果有, 里面的内容显示
77      * 如果没有, 里面的内容不予显示(这就完成了对于权限的认证。)
78      *
79      * shiro的权限授权是通过继承AuthorizingRealm抽象类, 重载doGetAuthorizationInfo();
```

```

80 * 当访问到页面的时候, 链接配置了相应的权限或者shiro标签才会执行此方法否则不会执行
81 * 所以如果只是简单的身份认证没有权限的控制的话, 那么这个方法可以不进行实现, 直接返回null即可。
82 *
83 * 在这个方法中主要是使用类: SimpleAuthorizationInfo 进行角色的添加和权限的添加。
84 * authorizationInfo.addRole(role.getRole()); authorizationInfo.addStringPermission(p.getPermission());
85 *
86 * 当然也可以添加set集合: roles是从数据库查询的当前用户的角色, stringPermissions是从数据库查询的当前用户对应的权限
87 * authorizationInfo.setRoles(roles); authorizationInfo.setStringPermissions(stringPermissions);
88 *
89 * 就是说如果在shiro配置文件中添加了filterChainDefinitionMap.put("/add", "perms[权限添加]");
90 * 就说明访问/add这个链接必须要有“权限添加”这个权限才可以访问
91 *
92 * 如果在shiro配置文件中添加了filterChainDefinitionMap.put("/add", "roles[100002], perms[权限添加]");
93 * 就说明访问/add这个链接必须要有 “权限添加” 这个权限和具有 “100002” 这个角色才可以访问
94 * @param principalCollection
95 * @return
96 */
97 @Override
98 protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
99
100     System.out.println("查询权限方法调用了!!!");
101
102     // 获取用户
103     User user = (User) SecurityUtils.getSubject().getPrincipal();
104
105     // 获取用户角色
106     Set<Role> roles = this.roleMapper.findRolesByUserId(user.getId());
107     // 添加角色
108     SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
109     for (Role role : roles) {
110         authorizationInfo.addRole(role.getRole());
111     }
112
113     // 获取用户权限
114     Set<Permission> permissions = this.permissionMapper.findPermissionsByRoleId(roles);
115     // 添加权限
116     for (Permission permission:permissions) {
117         authorizationInfo.addStringPermission(permission.getPermission());
118     }
119
120     return authorizationInfo;
121 }
122
123 /**
124 * 重写方法, 清除当前用户的 授权缓存
125 * @param principals
126 */
127 @Override
128 public void clearCachedAuthorizationInfo(PrincipalCollection principals) {
129     super.clearCachedAuthorizationInfo(principals);
130 }
131
132 /**
133 * 重写方法, 清除当前用户的 认证缓存
134 * @param principals
135 */
136 @Override
137 public void clearCachedAuthenticationInfo(PrincipalCollection principals) {
138     super.clearCachedAuthenticationInfo(principals);
139 }
140
141 @Override
142 public void clearCache(PrincipalCollection principals) {
143     super.clearCache(principals);
144 }
145
146 /**
147 * 自定义方法: 清除所有 授权缓存
148 */
149 public void clearAllCachedAuthorizationInfo() {
150     getAuthorizationCache().clear();
151 }
152
153 /**
154 * 自定义方法: 清除所有 认证缓存
155 */
156 public void clearAllCachedAuthenticationInfo() {
157     getAuthenticationCache().clear();
158 }

```

```

159
160     /**
161      * 自定义方法: 清除所有的 认证缓存 和 授权缓存
162      */
163     public void clearAllCache() {
164         clearAllCachedAuthenticationInfo();
165         clearAllCachedAuthorizationInfo();
166     }
167
168 }

```

跟之前的 `ShiroRealm` 相比,唯一改变的了

`SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(),new MyByteSource(user.getUsername()),getName());` 这一行代码,添加了 加盐参数。

注意：大家可能看到了使用了 `MyByteSource` 而不是 `ByteSource.Util.bytes(user.getUsername())` 具体原因参考博客：

https://blog.csdn.net/qq_34021712/article/details/84567437

5.下面是生成密码加密加盐的方法,可以在注册的时候对明文进行加密 加盐 入库

```

1 package com.springboot.test.shiro;
2
3 import org.apache.shiro.crypto.hash.SimpleHash;
4 import org.apache.shiro.util.ByteSource;
5 import org.junit.Test;
6
7 /**
8  * @author: WangSaiChao
9  * @date: 2018/11/27
10  * @description: 给 密码进行 加密加盐 盐值默认为 用户名
11  */
12 public class PasswordSaltTest {
13
14     @Test
15     public void test() throws Exception {
16         System.out.println(md5("123456", "admin"));
17     }
18
19     public static final String md5(String password, String salt){
20         // 加密方式
21         String hashAlgorithmName = "MD5";
22         // 盐: 为了即使相同的密码不同的盐加密后的结果也不同
23         ByteSource byteSalt = ByteSource.Util.bytes(salt);
24         // 密码
25         Object source = password;
26         // 加密次数
27         int hashIterations = 2;
28         SimpleHash result = new SimpleHash(hashAlgorithmName, source, byteSalt, hashIterations);
29         return result.toString();
30     }
31
32 }

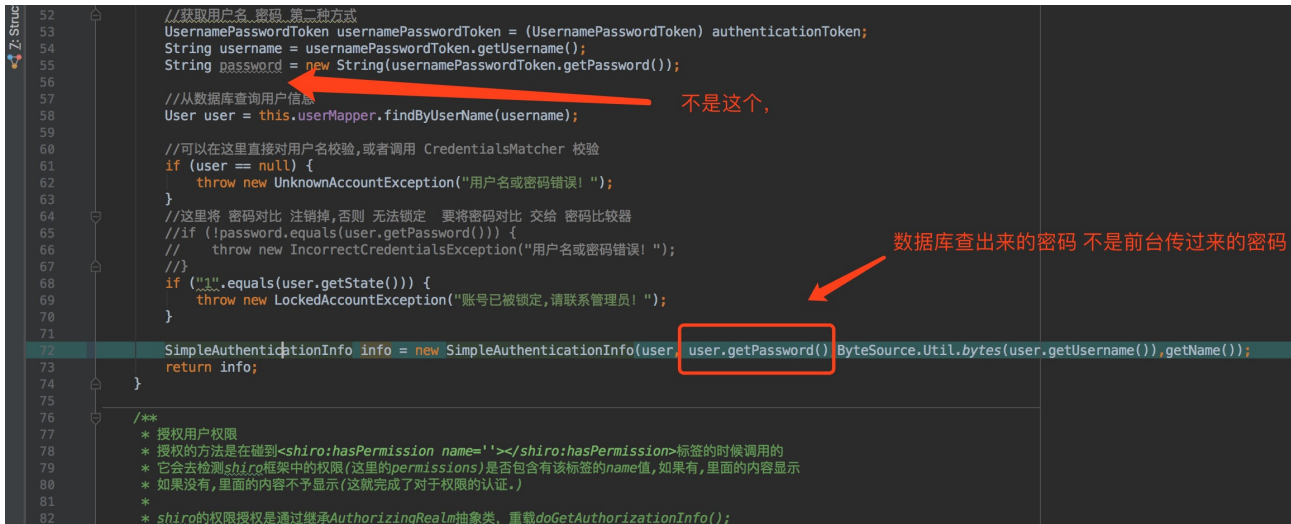
```

可能出现的问题

可能会发生这种情况, 测试发现密码不对, 具体原因debug都可以发现,这里直接把结果发出来:

第一种:

debug发现 传入的密码 经过加密加盐之后是对的,但是 从数据库中 获取的密码 却是明文,原因是在 `ShiroRealm` 中 `doGetAuthenticationInfo` 方法中,最后返回的 `SimpleAuthenticationInfo` 第二个参数 是密码,这个密码 不是从前台传过来的密码,而是从数据库中查询出来的



```
52 //获取用户名、密码 第二种方式
53 UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) authenticationToken;
54 String username = usernamePasswordToken.getUsername();
55 String password = new String(usernamePasswordToken.getPassword());
56
57 //从数据库查询用户信息
58 User user = this.userMapper.findByUserName(username);
59
60 //可以在这里直接对用户名称校验,或者调用 CredentialsMatcher 校验
61 if (user == null) {
62     throw new UnknownAccountException("用户名或密码错误! ");
63 }
64 //这里将 密码对比 注销掉,否则 无法锁定 要将密码对比 交给 密码比较器
65 //if (!password.equals(user.getPassword())) {
66 //    throw new IncorrectCredentialsException("用户名或密码错误! ");
67 //}
68 if ("1".equals(user.getState())) {
69     throw new LockedAccountException("账号已被锁定,请联系管理员! ");
70 }
71
72 SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(), ByteSource.Util.bytes(user.getUsername()), getName());
73 return info;
74 }
75
76 /**
77  * 授权用户权限
78  * 授权的方法是在碰到<shiro:hasPermission name='></shiro:hasPermission>标签的时候调用的
79  * 它会去检测shiro框架中的权限(这里的permissions)是否包含有该标签的name值,如果有,里面的内容显示
80  * 如果没有,里面的内容不予显示(这就完成了对于权限的认证.)
81  *
82  * shiro的权限授权是通过继承AuthorizingRealm抽象类,重载doGetAuthorizationInfo();
```

第二种:

debug发现 传入的密码 经过加密加盐之后是对的,但是 从数据库中 获取的密码 却是更长的一段密文,原因是在 `ShiroConfig` 中配置的 `RetryLimitHashedCredentialsMatcher` 一个属性:

```
1 //是否存储为16进制
2 retryLimitHashedCredentialsMatcher.setStoredCredentialsHexEncoded(true);
```

默认是 `true`, 如果改为 `false`, 则会出现 对比的时候从数据库拿出密码,然后转 `base64` 变成了另外一个更长的字符串,所以怎么对比都是不通过的。