[**第五章 编码/加密——《跟我学Shiro》**](http://jinnianshilongnian.iteye.com/blog/2021439)

**博客分类：**

* [跟我学Shiro](http://jinnianshilongnian.iteye.com/category/305053)

[跟我学Shiro](http://www.iteye.com/blogs/tag/%E8%B7%9F%E6%88%91%E5%AD%A6Shiro)

**目录贴：**[**跟我学Shiro目录贴**](http://jinnianshilongnian.iteye.com/blog/2018398)

在涉及到密码存储问题上，应该加密/生成密码摘要存储，而不是存储明文密码。比如之前的600w csdn账号泄露对用户可能造成很大损失，因此应加密/生成不可逆的摘要方式存储。

**5.1 编码/解码**

Shiro提供了base64和16进制字符串编码/解码的API支持，方便一些编码解码操作。Shiro内部的一些数据的存储/表示都使用了base64和16进制字符串。

**Java代码  [收藏代码](javascript:void())**

1. String str = "hello";
2. String base64Encoded = Base64.encodeToString(str.getBytes());
3. String str2 = Base64.decodeToString(base64Encoded);
4. Assert.assertEquals(str, str2);

通过如上方式可以进行base64编码/解码操作，更多API请参考其Javadoc。

**Java代码  [收藏代码](javascript:void())**

1. String str = "hello";
2. String base64Encoded = Hex.encodeToString(str.getBytes());
3. String str2 = **new** String(Hex.decode(base64Encoded.getBytes()));
4. Assert.assertEquals(str, str2);

通过如上方式可以进行16进制字符串编码/解码操作，更多API请参考其Javadoc。

还有一个可能经常用到的类CodecSupport，提供了toBytes(str, "utf-8") / toString(bytes, "utf-8")用于在byte数组/String之间转换。

**5.2 散列算法**

散列算法一般用于生成数据的摘要信息，是一种不可逆的算法，一般适合存储密码之类的数据，常见的散列算法如MD5、SHA等。一般进行散列时最好提供一个salt（盐），比如加密密码“admin”，产生的散列值是“21232f297a57a5a743894a0e4a801fc3”，可以到一些md5解密网站很容易的通过散列值得到密码“admin”，即如果直接对密码进行散列相对来说破解更容易，此时我们可以加一些只有系统知道的干扰数据，如用户名和ID（即盐）；这样散列的对象是“密码+用户名+ID”，这样生成的散列值相对来说更难破解。

**Java代码  [收藏代码](javascript:void())**

1. String str = "hello";
2. String salt = "123";
3. String md5 = **new** Md5Hash(str, salt).toString();//还可以转换为 toBase64()/toHex()

如上代码通过盐“123”MD5散列“hello”。另外散列时还可以指定散列次数，如2次表示：md5(md5(str))：“new Md5Hash(str, salt, 2).toString()”。

**Java代码  [收藏代码](javascript:void())**

1. String str = "hello";
2. String salt = "123";
3. String sha1 = **new** Sha256Hash(str, salt).toString();

使用SHA256算法生成相应的散列数据，另外还有如SHA1、SHA512算法。

Shiro还提供了通用的散列支持：

**Java代码  [收藏代码](javascript:void())**

1. String str = "hello";
2. String salt = "123";
3. //内部使用MessageDigest
4. String simpleHash = **new** SimpleHash("SHA-1", str, salt).toString();

通过调用SimpleHash时指定散列算法，其内部使用了Java的MessageDigest实现。

为了方便使用，Shiro提供了HashService，默认提供了DefaultHashService实现。

**Java代码  [收藏代码](javascript:void())**

1. DefaultHashService hashService = **new** DefaultHashService(); //默认算法SHA-512
2. hashService.setHashAlgorithmName("SHA-512");
3. hashService.setPrivateSalt(**new** SimpleByteSource("123")); //私盐，默认无
4. hashService.setGeneratePublicSalt(**true**);//是否生成公盐，默认false
5. hashService.setRandomNumberGenerator(**new** SecureRandomNumberGenerator());//用于生成公盐。默认就这个
6. hashService.setHashIterations(1); //生成Hash值的迭代次数
8. HashRequest request = **new** HashRequest.Builder()
9. .setAlgorithmName("MD5").setSource(ByteSource.Util.bytes("hello"))
10. .setSalt(ByteSource.Util.bytes("123")).setIterations(2).build();
11. String hex = hashService.computeHash(request).toHex();

1、首先创建一个DefaultHashService，默认使用SHA-512算法；

2、可以通过hashAlgorithmName属性修改算法；

3、可以通过privateSalt设置一个私盐，其在散列时自动与用户传入的公盐混合产生一个新盐；

4、可以通过generatePublicSalt属性在用户没有传入公盐的情况下是否生成公盐；

5、可以设置randomNumberGenerator用于生成公盐；

6、可以设置hashIterations属性来修改默认加密迭代次数；

7、需要构建一个HashRequest，传入算法、数据、公盐、迭代次数。

SecureRandomNumberGenerator用于生成一个随机数：

**Java代码  [收藏代码](javascript:void())**

1. SecureRandomNumberGenerator randomNumberGenerator =
2. **new** SecureRandomNumberGenerator();
3. randomNumberGenerator.setSeed("123".getBytes());
4. String hex = randomNumberGenerator.nextBytes().toHex();

**5.3 加密/解密**

Shiro还提供对称式加密/解密算法的支持，如AES、Blowfish等；当前还没有提供对非对称加密/解密算法支持，未来版本可能提供。

AES算法实现：

**Java代码  [收藏代码](javascript:void())**

1. AesCipherService aesCipherService = **new** AesCipherService();
2. aesCipherService.setKeySize(128); //设置key长度
3. //生成key
4. Key key = aesCipherService.generateNewKey();
5. String text = "hello";
6. //加密
7. String encrptText =
8. aesCipherService.encrypt(text.getBytes(), key.getEncoded()).toHex();
9. //解密
10. String text2 =
11. **new** String(aesCipherService.decrypt(Hex.decode(encrptText), key.getEncoded()).getBytes());
13. Assert.assertEquals(text, text2);

更多算法请参考示例com.github.zhangkaitao.shiro.chapter5.hash.CodecAndCryptoTest。

**5.4 PasswordService/CredentialsMatcher**

Shiro提供了PasswordService及CredentialsMatcher用于提供加密密码及验证密码服务。

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** PasswordService {
2. //输入明文密码得到密文密码
3. String encryptPassword(Object plaintextPassword) **throws** IllegalArgumentException;
4. }

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** CredentialsMatcher {
2. //匹配用户输入的token的凭证（未加密）与系统提供的凭证（已加密）
3. **boolean** doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info);
4. }

Shiro默认提供了PasswordService实现DefaultPasswordService；CredentialsMatcher实现PasswordMatcher及HashedCredentialsMatcher（更强大）。

**DefaultPasswordService配合PasswordMatcher实现简单的密码加密与验证服务**

1、定义Realm（com.github.zhangkaitao.shiro.chapter5.hash.realm.MyRealm）

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MyRealm **extends** AuthorizingRealm {
2. **private** PasswordService passwordService;
3. **public** **void** setPasswordService(PasswordService passwordService) {
4. **this**.passwordService = passwordService;
5. }
6. //省略doGetAuthorizationInfo，具体看代码
7. @Override
8. **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {
9. **return** **new** SimpleAuthenticationInfo(
10. "wu",
11. passwordService.encryptPassword("123"),
12. getName());
13. }
14. }

为了方便，直接注入一个passwordService来加密密码，实际使用时需要在Service层使用passwordService加密密码并存到数据库。

2、ini配置（shiro-passwordservice.ini）

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. passwordService=org.apache.shiro.authc.credential.DefaultPasswordService
3. hashService=org.apache.shiro.crypto.hash.DefaultHashService
4. passwordService.hashService=$hashService
5. hashFormat=org.apache.shiro.crypto.hash.format.Shiro1CryptFormat
6. passwordService.hashFormat=$hashFormat
7. hashFormatFactory=org.apache.shiro.crypto.hash.format.DefaultHashFormatFactory
8. passwordService.hashFormatFactory=$hashFormatFactory
10. passwordMatcher=org.apache.shiro.authc.credential.PasswordMatcher
11. passwordMatcher.passwordService=$passwordService
13. myRealm=com.github.zhangkaitao.shiro.chapter5.hash.realm.MyRealm
14. myRealm.passwordService=$passwordService
15. myRealm.credentialsMatcher=$passwordMatcher
16. securityManager.realms=$myRealm

2.1、passwordService使用DefaultPasswordService，如果有必要也可以自定义；

2.2、hashService定义散列密码使用的HashService，默认使用DefaultHashService（默认SHA-256算法）；

2.3、hashFormat用于对散列出的值进行格式化，默认使用Shiro1CryptFormat，另外提供了Base64Format和HexFormat，对于有salt的密码请自定义实现ParsableHashFormat然后把salt格式化到散列值中；

2.4、hashFormatFactory用于根据散列值得到散列的密码和salt；因为如果使用如SHA算法，那么会生成一个salt，此salt需要保存到散列后的值中以便之后与传入的密码比较时使用；默认使用DefaultHashFormatFactory；

2.5、passwordMatcher使用PasswordMatcher，其是一个CredentialsMatcher实现；

2.6、将credentialsMatcher赋值给myRealm，myRealm间接继承了AuthenticatingRealm，其在调用getAuthenticationInfo方法获取到AuthenticationInfo信息后，会使用credentialsMatcher来验证凭据是否匹配，如果不匹配将抛出IncorrectCredentialsException异常。

3、测试用例请参考com.github.zhangkaitao.shiro.chapter5.hash.PasswordTest。

另外可以参考配置shiro-jdbc-passwordservice.ini，提供了JdbcRealm的测试用例，测试前请先调用sql/shiro-init-data.sql初始化用户数据。

如上方式的缺点是：salt保存在散列值中；没有实现如密码重试次数限制。

**HashedCredentialsMatcher实现密码验证服务**

Shiro提供了CredentialsMatcher的散列实现HashedCredentialsMatcher，和之前的PasswordMatcher不同的是，它只用于密码验证，且可以提供自己的盐，而不是随机生成盐，且生成密码散列值的算法需要自己写，因为能提供自己的盐。

**1、生成密码散列值**

此处我们使用MD5算法，“密码+盐（用户名+随机数）”的方式生成散列值：

**Java代码  [收藏代码](javascript:void())**

1. String algorithmName = "md5";
2. String username = "liu";
3. String password = "123";
4. String salt1 = username;
5. String salt2 = **new** SecureRandomNumberGenerator().nextBytes().toHex();
6. **int** hashIterations = 2;
8. SimpleHash hash = **new** SimpleHash(algorithmName, password, salt1 + salt2, hashIterations);
9. String encodedPassword = hash.toHex();

如果要写用户模块，需要在新增用户/重置密码时使用如上算法保存密码，将生成的密码及salt2存入数据库（因为我们的散列算法是：md5(md5(密码+username+salt2))）。

2、生成Realm（com.github.zhangkaitao.shiro.chapter5.hash.realm.MyRealm2）

**Java代码  [收藏代码](javascript:void())**

1. **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {
2. String username = "liu"; //用户名及salt1
3. String password = "202cb962ac59075b964b07152d234b70"; //加密后的密码
4. String salt2 = "202cb962ac59075b964b07152d234b70";
5. SimpleAuthenticationInfo ai =
6. **new** SimpleAuthenticationInfo(username, password, getName());
7. ai.setCredentialsSalt(ByteSource.Util.bytes(username+salt2)); //盐是用户名+随机数
8. **return** ai;
9. }

此处就是把步骤1中生成的相应数据组装为SimpleAuthenticationInfo，通过SimpleAuthenticationInfo的credentialsSalt设置盐，HashedCredentialsMatcher会自动识别这个盐。

如果使用JdbcRealm，需要修改获取用户信息（包括盐）的sql：“select password, password\_salt from users where username = ?”，而我们的盐是由username+password\_salt组成，所以需要通过如下ini配置（shiro-jdbc-hashedCredentialsMatcher.ini）修改：

**Java代码  [收藏代码](javascript:void())**

1. jdbcRealm.saltStyle=COLUMN
2. jdbcRealm.authenticationQuery=select password, concat(username,password\_salt) from users where username = ?
3. jdbcRealm.credentialsMatcher=$credentialsMatcher

1、saltStyle表示使用密码+盐的机制，authenticationQuery第一列是密码，第二列是盐；

2、通过authenticationQuery指定密码及盐查询SQL；

此处还要注意Shiro默认使用了apache commons BeanUtils，默认是不进行Enum类型转型的，此时需要自己注册一个Enum转换器“BeanUtilsBean.getInstance().getConvertUtils().register(new EnumConverter(), JdbcRealm.SaltStyle.class);”具体请参考示例“com.github.zhangkaitao.shiro.chapter5.hash.PasswordTest”中的代码。

另外可以参考配置shiro-jdbc-passwordservice.ini，提供了JdbcRealm的测试用例，测试前请先调用sql/shiro-init-data.sql初始化用户数据。

3、ini配置（shiro-hashedCredentialsMatcher.ini）

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. credentialsMatcher=org.apache.shiro.authc.credential.HashedCredentialsMatcher
3. credentialsMatcher.hashAlgorithmName=md5
4. credentialsMatcher.hashIterations=2
5. credentialsMatcher.storedCredentialsHexEncoded=**true**
6. myRealm=com.github.zhangkaitao.shiro.chapter5.hash.realm.MyRealm2
7. myRealm.credentialsMatcher=$credentialsMatcher
8. securityManager.realms=$myRealm

1、通过credentialsMatcher.hashAlgorithmName=md5指定散列算法为md5，需要和生成密码时的一样；

2、credentialsMatcher.hashIterations=2，散列迭代次数，需要和生成密码时的意义；

3、credentialsMatcher.storedCredentialsHexEncoded=true表示是否存储散列后的密码为16进制，需要和生成密码时的一样，默认是base64；

此处最需要注意的就是HashedCredentialsMatcher的算法需要和生成密码时的算法一样。另外HashedCredentialsMatcher会自动根据AuthenticationInfo的类型是否是SaltedAuthenticationInfo来获取credentialsSalt盐。

4、测试用例请参考com.github.zhangkaitao.shiro.chapter5.hash.PasswordTest。

**密码重试次数限制**

如在1个小时内密码最多重试5次，如果尝试次数超过5次就锁定1小时，1小时后可再次重试，如果还是重试失败，可以锁定如1天，以此类推，防止密码被暴力破解。我们通过继承HashedCredentialsMatcher，且使用Ehcache记录重试次数和超时时间。

com.github.zhangkaitao.shiro.chapter5.hash.credentials.RetryLimitHashedCredentialsMatcher：

**Java代码  [收藏代码](javascript:void())**

1. **public** **boolean** doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {
2. String username = (String)token.getPrincipal();
3. //retry count + 1
4. Element element = passwordRetryCache.get(username);
5. **if**(element == **null**) {
6. element = **new** Element(username , **new** AtomicInteger(0));
7. passwordRetryCache.put(element);
8. }
9. AtomicInteger retryCount = (AtomicInteger)element.getObjectValue();
10. **if**(retryCount.incrementAndGet() > 5) {
11. //if retry count > 5 throw
12. **throw** **new** ExcessiveAttemptsException();
13. }
15. **boolean** matches = **super**.doCredentialsMatch(token, info);
16. **if**(matches) {
17. //clear retry count
18. passwordRetryCache.remove(username);
19. }
20. **return** matches;
21. }

如上代码逻辑比较简单，即如果密码输入正确清除cache中的记录；否则cache中的重试次数+1，如果超出5次那么抛出异常表示超出重试次数了。