1. Cryptography密码系统

本章主要介绍基于Shiro安全框架中提供的加密功能，以及在身份认证过程中对加密功能的使用。关于本章我们主要把握如下几个方面：

* Shiro密码功能的特点
* 密码功能相关的算法和对应的API
* 使用密码功能进行加密
* 基于密码功能实现身份认证

目录

[7 Cryptography密码系统 1](#_Toc494211017)

[7.1 加密简介 2](#_Toc494211018)

[7.2 编码和解码 2](#_Toc494211019)

[7.3 Hash算法加密 3](#_Toc494211020)

[7.4 对称加密解密 6](#_Toc494211021)

[7.5 加密服务和验证密码服务 7](#_Toc494211022)

[7.6 实现基于加密密码的身份认证 8](#_Toc494211023)

[7.7 总结 12](#_Toc494211024)

在实际应用中，像密码这类信息是需要加密后再存储，而不是明码保存的，否则就很容易出现系统账号信息泄露事件。这样我们的系统就需要提供对应的加密功能和对加密密码进行身份验证的功能。Shiro安全框架的Cryptography模块就提供了这块的支持。

* 1. 加密简介

Shiro安全框架的Cryptography模块的主要目的是将复杂的密码系统的实现提炼成接口，并做好相关的基本实现，方便程序员在这些API的基础上，简单快速的实现加密、解密、密码匹配等强大的密码功能。

Shiro安全框架的Crytography模块有如下特点：

* 简单

1. 设计面向接口，基于POJO进行接口实现

所有的Shiro的密码模块API都是设计成接口，再用简单的POJO类进行实现的。这样我们既可以很方便的配置Crytography模块的组件对象，也方便我们自定义实现相关的接口。

1. 包装简化JCE

Java自带的JCE(Java Crytography Extension)是完整且庞大的标准，理解和使用起来都比较困难，Shiro的Crytography对这套标准进行包装和实现，方便我们更容易理解和使用。

1. 简单对象方式使用

在jdk的jce中，定义了很多抽象类、接口、工厂和实现，理解使用起来很麻烦，Shiro的Crytography可以直接创建一个简单类的对象，就可以基于hash的加密、对称的加密解密、编码和解码功能。

1. 支持运行态异常

同Shiro其他的API一样，Shiro的Crytography模块api以抛出RuntimeException的方法返回操作过程中的各种意外情况，以便应用成做不同的处理。

* 简单的对称的加密解密

1. 直接简单的对象实现

Shiro的Crytography提供了简单直接的密码实现类，比如AbstractSymmetricCipherService, DefaultBlockCipherService, AesCipherService等等，我们直接可以使用。

1. 简单的创建对象

直接用new运算符创建需要的对象。

1. 缺省设置安全参数

创建安全对象的时候，Shiro自动给里面复杂的安全参数做了缺省的设置，省得程序员自己做复杂的设置初始化。

* Hash加密

1. 缺省的Hash实现

对常用的Hash算法提供支持，比如MD5， SHA1， SHA-256等

1. 直接hex和base64的编码

支持基于hex和base64的编码

1. 支持盐

支持带盐的数据加密

* 1. 编码和解码

在实际存储数据过程中，有很多情况需要将二进制数组以字符串的形式保存，需要的时候在还原成二进制数组。这时候我们就需要将二进制数据进行编码和解码。Shiro安全框架中提供了Base64和16进制的编码和解码，同时还提供了CodecSupport类，支持不同编码的二进制和字符串的转换方法。请看下面各种的样例代码了解它们的使用，详细内容请查看随书代码[code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestEncoding  
.java](../code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestEncoding.java)。

* Base64

@Test

public void testBase64EncodeDecode(){

String data = "hello,how are you!";

// 基于Base64进行编码

String base64Encoded = Base64.encode(data.getBytes());

System.out.println(base64Encoded);

// 解码

try {

String result = new String(Base64.decode(base64Encoded));

Assert.assertEquals(data, result);

} catch (Base64DecodingException e) {

e.printStackTrace();

throw new RuntimeException(e);

}

}

* Hex

@Test

public void testHexEncodeDecode(){

String data = "Hello,你好";

String encoded = Hex.encodeToString(data.getBytes());

System.out.println(encoded);

String result = new String(Hex.decode(encoded));

Assert.assertEquals(data, result);

}

* CodecSupport

@Test

public void testCodecSupport(){

String data = "Hello,你好吗!";

byte[] bytes = CodecSupport.toBytes(data, "utf-8");

String result = CodecSupport.toString(bytes,"utf-8");

Assert.assertEquals(data, result);

}

* 1. Hash算法加密

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。

Shiro安全框架提供了对MD5、SHA256（SHA1， SHA512）和通用散列SimpleHash的支持，还提供了了一个DefaultHashService服务封装类，方便我们使用。如下分别用样例代码说明对它们的使用，详细代码请查看随书代码[code/ShiroTP07/src/test/java/  
cn/com/shiro/book/tp07/TestHash.java](../code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestHash.java)。

* MD5

@Test

public void testMD5(){

System.out.println("md5-->");

String data = "Hello,how are you!";

String salt = "你好";

Md5Hash md5Hash = new Md5Hash(data,salt);

String result = md5Hash.toString();

System.out.println("default:" + result);

result = md5Hash.toBase64();

System.out.println("base64:" + result);

result = md5Hash.toHex();

System.out.println("hex:" + result);

md5Hash = new Md5Hash(data,salt,2);

result = md5Hash.toString();

System.out.println("2default:" + result);

result = md5Hash.toBase64();

System.out.println("2base64:" + result);

result = md5Hash.toHex();

System.out.println("2hex:" + result);

}

* SHA256

@Test

public void testSHA256(){

System.out.println("sha256-->");

String data = "Hello,how are you!";

String salt = "你好";

Sha256Hash md5Hash = new Sha256Hash(data,salt);

String result = md5Hash.toString();

System.out.println("default:" + result);

result = md5Hash.toBase64();

System.out.println("base64:" + result);

result = md5Hash.toHex();

System.out.println("hex:" + result);

md5Hash = new Sha256Hash(data,salt,2);

result = md5Hash.toString();

System.out.println("2default:" + result);

result = md5Hash.toBase64();

System.out.println("2base64:" + result);

result = md5Hash.toHex();

System.out.println("2hex:" + result);

}

* SimpleHash

@Test

public void testSimpleHash(){

System.out.println("SimpleHash-->");

String data = "Hello,how are you!";

String salt = "你好";

SimpleHash md5Hash = new SimpleHash("SHA-1",data,salt);//SHA-512

String result = md5Hash.toString();

System.out.println("default:" + result);

result = md5Hash.toBase64();

System.out.println("base64:" + result);

result = md5Hash.toHex();

System.out.println("hex:" + result);

md5Hash = new Sha256Hash(data,salt,2);

result = md5Hash.toString();

System.out.println("2default:" + result);

result = md5Hash.toBase64();

System.out.println("2base64:" + result);

result = md5Hash.toHex();

System.out.println("2hex:" + result);

}

* DefaultHashService

@Test

public void testDefaultHashService(){

System.out.println("\*\*\*defaultHashService-->");

String data = "hello,how are you !";

String privateSalt = "privateSalt";

String publicSalt = "publicSalt";

DefaultHashService hashService = new DefaultHashService(); //默认算法SHA-512

hashService.setHashAlgorithmName("SHA-512");

hashService.setPrivateSalt(new SimpleByteSource(privateSalt)); //私盐，默认无

hashService.setGeneratePublicSalt(true);//是否生成公盐，默认false

hashService.setRandomNumberGenerator(new SecureRandomNumberGenerator());//用于生成公盐。默认就这个

hashService.setHashIterations(1); //生成Hash值的迭代次数

HashRequest request = new HashRequest.Builder()

.setAlgorithmName("MD5").setSource(ByteSource.Util.bytes(data))

.setSalt(ByteSource.Util.bytes(publicSalt)).setIterations(2).build();

String result = hashService.computeHash(request).toHex();

System.out.println(result);

String r = "b475c1dd01076aabc82aba9ce7c63deb";

Assert.assertEquals(r, result);

}

* 1. 对称加密解密

Shiro安全框架处理支持Hash算法的加密支持，Shiro还提供对称式加密/解密算法的支持，如AES、Blowfish等。下面样例代码分别介绍AES和Blowfish的使用，详细内容请查看随书代码[code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestDesEncyption  
.java](../code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestDesEncryption.java)。

* Blowfish

@Test

public void testBlowfishCipherSerivce() {

BlowfishCipherService blowfishCipherSerivce = new BlowfishCipherService();

blowfishCipherSerivce.setKeySize(128);

String text = "你好";

Key key = blowfishCipherSerivce.generateNewKey();

// 加密

String encrptText = blowfishCipherSerivce.encrypt(text.getBytes(),

key.getEncoded()).toHex();

// 解密

String text2 = new String(blowfishCipherSerivce.decrypt(

Hex.decode(encrptText), key.getEncoded()).getBytes());

System.out.println(encrptText);

Assert.assertEquals(text, text2);

}

* AES

@Test

public void testAesCipherService() {

AesCipherService aesCipherService = new AesCipherService();

aesCipherService.setKeySize(128); // 设置key长度

// 生成key

Key key = aesCipherService.generateNewKey();

String text = "hello,how are you";

// 加密

String encrptText = aesCipherService.encrypt(text.getBytes(),

key.getEncoded()).toHex();

// 解密

String text2 = new String(aesCipherService.decrypt(

Hex.decode(encrptText), key.getEncoded()).getBytes());

System.out.println(encrptText);

Assert.assertEquals(text, text2);

}

* 1. 加密服务和验证密码服务

前面几个小节，介绍的是Shiro Crytography对加密系统的底层支持，Shiro还提供了PasswordService及CredentialsMatcher用于提供加密密码及验证密码服务。它们的接口定义分别如下：

PasswordService

public interface PasswordService {

//输入明文密码得到密文密码

String encryptPassword(Object plaintextPassword) throws IllegalArgumentException;

}

CredentialsMatcher

public interface CredentialsMatcher {

//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）

boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info);

}

同时，Shiro也对这些接口提供了默认的实现。PasswordService接口的默认实现是DefaultPasswordService；CredentialsMatcher接口的默认实现是PasswordMatcher和HashedCredentialsMatcher。

下面代码块分别介绍了DefaultPasswordService和PasswordMatcher实现类的使用。详细内容请查看随书代码[code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestEncryptAnd  
Matcher.java](../code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestEncryptAndMatcher.java)。

* DefaultPasswordService

@Test

public void testPasswordService(){

DefaultPasswordService pswService = new DefaultPasswordService();

pswService.setHashService(new DefaultHashService());

pswService.setHashFormat(new Shiro1CryptFormat());

pswService.setHashFormatFactory(new DefaultHashFormatFactory());

String data = "你好，好啊yu";

String result = pswService.encryptPassword(data);

System.out.println(result);

Assert.assertTrue(pswService.passwordsMatch(data, result));

}

* PasswordMatcher

@Test

public void testPasswordMatcher(){

DefaultPasswordService pswService = new DefaultPasswordService();

pswService.setHashService(new DefaultHashService());

pswService.setHashFormat(new Shiro1CryptFormat());

pswService.setHashFormatFactory(new DefaultHashFormatFactory());

PasswordMatcher matcher = new PasswordMatcher();

matcher.setPasswordService(pswService);

AuthenticationToken token = new UsernamePasswordToken("zhangsan","你好，好啊yu","test");

AuthenticationInfo info = new SimpleAuthenticationInfo("zhangsan","$shiro1$SHA-512$1$$MxtK7wQBV1lXC+U+oQegZKyKsKDKm3ZRTM00yoHnojFzTspHMndb9nFe+Ac20nmraBt1Zm+1sRVF+FtIYqV2VQ==","test");// 中间是加密密码

Assert.assertTrue(matcher.doCredentialsMatch(token, info));

}

* 1. 实现基于加密密码的身份认证

接下来我们介绍一下怎么将Shiro Crytography运用到Shiro的身份认证中去。在身份认证中运用Crytography的关键点是把握几点：

1. 保存在安全数据中的密码是加密后的密码
2. 在身份认证的时候，需要将用户输入的明码，加密后同安全数据中的密码进行比对
3. 对用户身份认证时输入的密码进行加密的算法，需要同加密安全数据中密码的算法要一样。

接下来介绍一下简单的Crytography身份认证。

1. 自定义BitPermission类。

这个类的作用和实现，同ShiroTP06中的BitPermission一样，这里就不重复样例代码块了，详细内容请查看随书代码[code/ShiroTP07/src/main/java/cn/com/  
shiro/book/tp07/authz/permission/BitPermission.java](../code/ShiroTP07/src/main/java/cn/com/shiro/book/tp07/authz/permission/BitPermission.java)。

1. 自定义BitAndWildcardPermissionResolver.java类

这个类的作用和实现，同ShiroTP06中的BitAndWildcardPermissionResolver一样，这里就不重复样例代码块了，详细内容请查看随书代码[code/ShiroTP07/ src/main/java/cn/com/shiro/book/tp07/authz/permission/BitAndWildcardPermissionResolver.java](../code/ShiroTP07/src/main/java/cn/com/shiro/book/tp07/authz/permission/BitAndWildcardPermissionResolver.java)。

1. 自定义ShiroRolePermissionResolver.java类。

这个类的作用和实现，同ShiroTP06中的ShiroRolePermissionResolver一样，这里就不重复样例代码块了，详细内容请查看随书代码[code/ShiroTP07/ src/main/java/cn/com/shiro/book/tp07/authz/permission/ShiroRolePermissionResolver.java](../code/ShiroTP07/src/main/java/cn/com/shiro/book/tp07/authz/permission/ShiroRolePermissionResovler.java)。

1. 自定义ShiroRealm.java类

ShiroRealm类继承AuthorizingRealm父类，实现进行身份认证的基本方法(doGetAuthenticationInfo)和授权控制的基本方法(doGetAuthorizationInfo)。

* + 1. doGetAuthentictionInfo(AuthenticationToken)方法的作用是根据应用通过Subject的login方法传入的AuthenticationToken的身份标识(Principals)，从应用安全数据中找出对应的认证Subject身份合法性的凭证信息（Credentials），封装成AuthenticationInfo对象返回，以便后面流程的身份认证。
    2. doGetAuthorizationInfo(PrincipalCollection)反方法的作用是根据传入的Principals从应用安全数据中找出该Subject拥有的角色集合和权限集合信息，封装到AuthorizationInfo对象中返回，以便后面流程的授权判断。

ShiroRealm类的样例代码块如下，详细内容请查看随书代码[code/ShiroTP07/ src/main/java/cn/com/shiro/book/tp07/realm/ShiroRealm.java](../code/ShiroTP07/src/main/java/cn/com/shiro/book/tp07/realm/ShiroRealm.java)。

public class ShiroRealm extends AuthorizingRealm {

private PasswordService passwordService;

private Map<String, String> accountMap = new HashMap<String, String>();

private Map<String, List<String>> userRolesMap = new HashMap<String, List<String>>();

private Map<String, List<String>> userPermsMap = new HashMap<String, List<String>>();

// 模拟数据源初始化数据

{

// 初始化用户角色信息

userRolesMap.put("zhangsan", Arrays.asList("admin"));

userRolesMap.put("lisi", Arrays.asList("role1", "role2"));

userRolesMap.put("wangwu", Arrays.asList("role3"));

// 初始化用户权限信息

userPermsMap.put("wangwu",

Arrays.asList("order:create", "order:delete"));

}

/\*\*

\* 获取指定principals包含的角色和权限信息

\* \*/

@Override

protected AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

if (this.userRolesMap.get(principals.getPrimaryPrincipal()) != null) {

info.addRoles(this.userRolesMap.get(principals

.getPrimaryPrincipal()));

}

if (this.userPermsMap.get(principals.getPrimaryPrincipal()) != null) {

info.addStringPermissions(this.userPermsMap.get(principals

.getPrimaryPrincipal()));

}

return info;

}

/\*\*

\* 根据token的principals返回从数据源中获取的对应AuthenticationInfo

\* \*/

@Override

protected AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) throws AuthenticationException {

SimpleAuthenticationInfo info = null;

String password = this.accountMap.get(token.getPrincipal().toString());

if (password != null) {

info = new SimpleAuthenticationInfo(token.getPrincipal(),

password,

this.getName());

}

return info;

}

public PasswordService getPasswordService() {

return passwordService;

}

public void setPasswordService(PasswordService passwordService) {

this.passwordService = passwordService;

**initAccounts(); ①**

}

// 使用passwordService初始化账号信息，密码加密

private void initAccounts() {

// 初始化用户账号信息

accountMap.put("zhangsan",

**this.passwordService.encryptPassword("abc123")**); **②**

accountMap.put("lisi", **this.passwordService.encryptPassword("123456")**);

accountMap

.put("wangwu", **this.passwordService.encryptPassword("123456")**);

}

}

如上代码中的三个Map成员变量，分别模拟的是应用安全数据中的用户密码集合、用户角色集合和用户权限集合。

注意**①**标注的代码，它的作用是模拟初始化应用安全数据中的账号密码信息；**②**标记部分代码的作用是使用passwordService加密，再将加密后的密码放入账号信息中。

1. 在resources目录下创建shiro\_shiroRealm.ini，在里面配置securityManager，使用我们自定义BitAndWildcardPermissionResolver、ShiroRolePermissionResolver和ShiroRealm对象，完成身份认证和权限授权，同时注意粗体部分标识的是同Shiro Crytography整合配置。样例代码块如下，详细内容请查看随书代码[code/ShiroTP07/src/main/resources/shiro\_shiroRealm.ini](../code/ShiroTP07/src/main/resources/shiro_shiroRealm.ini)。

[main]

**passwordService=org.apache.shiro.authc.credential.DefaultPasswordService**

**hashService=org.apache.shiro.crypto.hash.DefaultHashService**

**passwordService.hashService=$hashService**

**hashFormat=org.apache.shiro.crypto.hash.format.Shiro1CryptFormat**

**passwordService.hashFormat=$hashFormat**

**hashFormatFactory=org.apache.shiro.crypto.hash.format.DefaultHashFormatFactory**

**passwordService.hashFormatFactory=$hashFormatFactory**

**passwordMatcher=org.apache.shiro.authc.credential.PasswordMatcher**

**passwordMatcher.passwordService=$passwordService**

#自定义authorizer

authorizer=org.apache.shiro.authz.ModularRealmAuthorizer

#自定义permissionResolver

permissionResolver=cn.com.shiro.book.tp07.authz.permission.BitAndWildcardPermissionResolver

authorizer.permissionResolver=$permissionResolver

#自定义rolePermissionResolver

rolePermissionResolver=cn.com.shiro.book.tp07.authz.permission.ShiroRolePermissionResovler

authorizer.rolePermissionResolver=$rolePermissionResolver

#设置SecurityManager授权器

securityManager.authorizer=$authorizer

#配置shiroRealm对象(realm名称=realm类名)

shiroRealm=cn.com.shiro.book.tp07.realm.ShiroRealm

**shiroRealm.passwordService=$passwordService**

**shiroRealm.credentialsMatcher=$passwordMatcher**

#设置shiroRealm对象

securityManager.realms=$shiroRealm

1. 编写TestShiroRealmAuthorizer代码进行测试。

因为这个测试代码同前面几个测试代码一样，这里就不重复累赘了，详细内容请查看随书代码[code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestShiroRealmAuthorizer.java](../code/ShiroTP07/src/test/java/cn/com/shiro/book/tp07/TestShiroRealmAuthorizer.java)。

* 1. 总结

本章主要介绍了Shiro Crytography中对密码技术支持的常用接口和实现类，已经它们的基本用法，还介绍了怎么样将Crytography集成到Shiro的身份认证流程去的思路和步骤，以及注意事项。关键点是把握注意事项，具体步骤可以参考7.6节。

注意事项是：

1. 保存在安全数据中的密码是加密后的密码
2. 在身份认证的时候，需要将用户输入的明码，加密后同安全数据中的密码进行比对
3. 对用户身份认证时输入的密码进行加密的算法，需要同加密安全数据中密码的算法要一样。