关于shiro使用密码加密加盐之后序列化失败的问题(十四)

shiro **使用密码加盐之后,序列化失败 ERROR Failed to serialize**

之前的博客一直都是使用的明文存储,一直没有写对密码进行 加密 、加盐处理,有很长时间没有写关于shiro的博客了,期间有很多人加我咨询shiro的问题,今天有个哥们说使用密码加盐后出现序列化失败的问题,找了一下原因，最后记录到博客，希望能给遇到此问题的人一些帮助。

原shiro配置

这里只贴出来造成 序列化 失败相关的配置,完整的原始配置参考：https://blog.csdn.net/qq_34021712/article/details/80791339

```
/**
 * @author: wangsaichao
 * @date: 2018/5/10
 * @description: Shiro配置
 */
@Configuration
public class ShiroConfig {


    /**
     *  身份认证realm; (这个需要自己写，账号密码校验；权限等)
     * @return
     */
    @Bean
    public ShiroRealm shiroRealm(){
        ShiroRealm shiroRealm = new ShiroRealm();
        shiroRealm.setCachingEnabled(true);
        //启用身份验证缓存，即缓存AuthenticationInfo信息，默认false
        shiroRealm.setAuthenticationCachingEnabled(true);
        //缓存AuthenticationInfo信息的缓存名称 在ehcache-shiro.xml中有对应缓存的配置
        shiroRealm.setAuthenticationCacheName("authenticationCache");
        //启用授权缓存，即缓存AuthorizationInfo信息，默认false
        shiroRealm.setAuthorizationCachingEnabled(true);
        //缓存AuthorizationInfo信息的缓存名称  在ehcache-shiro.xml中有对应缓存的配置
        shiroRealm.setAuthorizationCacheName("authorizationCache");
        //配置自定义密码比较器
        shiroRealm.setCredentialsMatcher(retryLimitHashedCredentialsMatcher());
        return shiroRealm;
    }

    /**
     * 配置密码比较器
     * @return
     */
    @Bean("credentialsMatcher")
    public RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher(){
        RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher = new RetryLimitHashedCredentialsMatcher();
        retryLimitHashedCredentialsMatcher.setRedisManager(redisManager());

        //如果密码加密,可以打开下面配置
        //加密算法的名称
        //retryLimitHashedCredentialsMatcher.setHashAlgorithmName("MD5");
        //配置加密的次数
        //retryLimitHashedCredentialsMatcher.setHashIterations(1024);
        //是否存储为16进制
        //retryLimitHashedCredentialsMatcher.setStoredCredentialsHexEncoded(true);

        return retryLimitHashedCredentialsMatcher;
    }

}
```

其中有一个 `RetryLimitHashedCredentialsMatcher` 类 是密码比较器,该类继承于 `SimpleCredentialsMatcher` 由于要进行密码加密加盐处理,所以要更改 `RetryLimitHashedCredentialsMatcher` 继承 `HashedCredentialsMatcher`

```
public class RetryLimitHashedCredentialsMatcher extends HashedCredentialsMatcher {

    // 具体内容省略 ......
}
```

并将 ShiroConfig中的 注释掉的3行打开：

```java
/**
     * 配置密码比较器
     * @return
     */
    @Bean("credentialsMatcher")
    public RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher(){
        RetryLimitHashedCredentialsMatcher retryLimitHashedCredentialsMatcher = new RetryLimitHashedCredentialsMatcher();
        retryLimitHashedCredentialsMatcher.setRedisManager(redisManager());

        //如果密码加密,可以打开下面配置
        //加密算法的名称
        retryLimitHashedCredentialsMatcher.setHashAlgorithmName("MD5");
        //配置加密的次数
        retryLimitHashedCredentialsMatcher.setHashIterations(1024);
        //是否存储为16进制
        retryLimitHashedCredentialsMatcher.setStoredCredentialsHexEncoded(tru);

        return retryLimitHashedCredentialsMatcher;
    }
```

修改ShiroRealm中的验证用户身份代码

```java
/**
 * 验证用户身份
 * @param authenticationToken
 * @return
 * @throws AuthenticationException
 */
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

    //获取用户名密码 第一种方式
    //String username = (String) authenticationToken.getPrincipal();
    //String password = new String((char[]) authenticationToken.getCredentials());

    //获取用户名 密码 第二种方式
    UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) authenticationToken;
    String username = usernamePasswordToken.getUsername();
    String password = new String(usernamePasswordToken.getPassword());

    //从数据库查询用户信息
    User user = this.userMapper.findByUserName(username);

    //可以在这里直接对用户名校验,或者调用 CredentialsMatcher 校验
    if (user == null) {
        throw new UnknownAccountException("用户名或密码错误！");
    }
    //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
    //if (!password.equals(user.getPassword())) {
    //    throw new IncorrectCredentialsException("用户名或密码错误！");
    //}
    if ("1".equals(user.getState())) {
        throw new LockedAccountException("账号已被锁定,请联系管理员！");
    }

    SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(),ByteSource.Util.bytes(user.getUsername()),getName());
    return info;
}
```

然后使用以下代码提前将test用户的密码加密加盐处理放入数据库中,方便测试用:

```
public static void main(String[] args) { System.out.println(md5( password: "123456", salt: "test")); }

public static final String md5(String password, String salt){
    //加密方式
    String hashAlgorithmName = "MD5";
    //盐: 为了即使相同的密码不同的盐加密后的结果也不同
    ByteSource byteSalt = ByteSource.Util.bytes(salt);
    //密码
    Object source = password;
    //加密次数
    int hashIterations = 2;
    SimpleHash result = new SimpleHash(hashAlgorithmName, source, byteSalt, hashIterations);
    return result.toString();
}

}
```

ShiroConfig › main()

ShiroConfig

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java ...
objc[2549]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (0x
.8.0_144.jdk/Contents/Home/jre/lib/libinstrument.dylib (0x1053354e0). One of the two will be used. Which one is undefined.
085de60f32b004402326e5ee425a6167

Process finished with exit code 0
```

https://blog.csdn.net/qq_34021712

最后启动项目，输入 test/123456 进行登录,报以下异常:

```
2018/11/26 21:23:22.381 c.s.t.s.global.utils.SerializeUtils [] ERROR Failed to serialize
java.io.NotSerializableException: org.apache.shiro.util.SimpleByteSource
```

序列化失败, SerializeUtils 是自己写的一个序列化工具,完整内容可以看一下之前的博客，序列化失败的原因就是因为 SimpleByteSource 不能被序列化，原因如下:

首先是 SerializeUtils 的序列化方法,其中 objectOutputStream.writeObject(object); 是真正的执行序列化操作

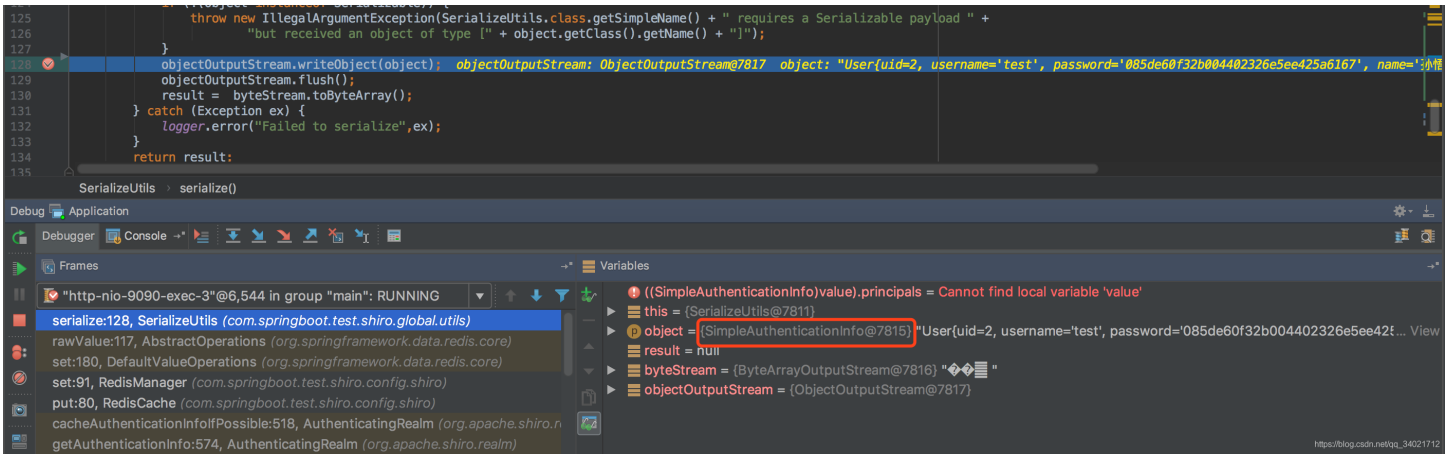
```
/**
 * 序列化
 * @param object
 * @return
 * @throws SerializationException
 */
@Override
public byte[] serialize(Object object) throws SerializationException {
    byte[] result = null;

    if (object == null) {
        return new byte[0];
    }
    try (
            ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
            ObjectOutputStream objectOutputStream = new ObjectOutputStream(byteStream)
    ){

        if (!(object instanceof Serializable)) {
            throw new IllegalArgumentException(SerializeUtils.class.getSimpleName() + " requires a Serializable payload " +
                    "but received an object of type [" + object.getClass().getName() + "]");
        }
        objectOutputStream.writeObject(object);
        objectOutputStream.flush();
        result =  byteStream.toByteArray();
    } catch (Exception ex) {
        logger.error("Failed to serialize",ex);
    }
    return result;
}
```

https://blog.csdn.net/qq_34021712

首先是序列化 SimpleAuthenticationInfo 类



SimpleAuthenticationInfo 类中有个属性 ByteSource， 默认使用的是 SimpleByteSource 就是因为该属性无法序列化导致的

解决方案

第一种：取消authenticationCache

在上面的 shiroRealm 配置中 我们开启了两个缓存： authenticationCache 和 authorizationCache 序列化失败的原因就是 因为开启了 authenticationCache 可以将 authenticationCache 对应的那两行配置 删除,只缓存 authorizationCache 。

第二种：自定义ByteSource的实现类

遇见这种自定义实现类,大家首先肯定想的是：写一个类继承 SimpleByteSource 然后实现序列化接口，如下：

```
/**
 * @author: wangsaichao
 * @date: 2018/11/27
 * @description:
 */
public class MySimpleByteSource extends SimpleByteSource implements Serializable{

    public MySimpleByteSource(String salt) {
        super(salt);
    }

}
```

然后在自定义的 Realm 中的 doGetAuthenticationInfo 方法中，返回 SimpleAuthenticationInfo 如下：

```
return new SimpleAuthenticationInfo(user, user.getPassword(),new MySimpleByteSource(user.getUsername()),getName());
```

注意：经过测试,在序列化的时候不报错,但是在反序列化的时候就报错了：

```
2018/11/27 11:50:21.090 c.s.t.s.global.utils.SerializeUtils [] ERROR Failed to deserialize
java.io.InvalidClassException: com.springboot.test.shiro.config.shiro.MySimpleByteSource; no valid constructor
```

因为在 SimpleByteSource 不存在 默认的 无参构造器，当 不存在无参构造器 或者 访问权限设置为private、默认或protected级别， 会抛出java.io.InvalidException: no valid constructor异常。

正确的解决办法

将 SimpleByteSource 整个类 复制粘贴 给个名字 叫 MyByteSource， 额外实现 Serializable 接口，并添加 无参构造器：

```java
/**
 * @author: wangsaichao
 * @date: 2018/11/27
 * @description: 解决 SimpleByteSource 无法序列化的问题
 */
public class MyByteSource implements ByteSource,Serializable {

    private byte[] bytes;
    private String cachedHex;
    private String cachedBase64;

    public MyByteSource() {
    }

    public MyByteSource(byte[] bytes) {
        this.bytes = bytes;
    }


    public MyByteSource(char[] chars) {
        this.bytes = CodecSupport.toBytes(chars);
    }


    public MyByteSource(String string) {
        this.bytes = CodecSupport.toBytes(string);
    }


    public MyByteSource(ByteSource source) {
        this.bytes = source.getBytes();
    }


    public MyByteSource(File file) {
        this.bytes = new MyByteSource.BytesHelper().getBytes(file);
    }


    public MyByteSource(InputStream stream) {
        this.bytes = new MyByteSource.BytesHelper().getBytes(stream);
    }

    public static boolean isCompatible(Object o) {
        return o instanceof byte[] || o instanceof char[] || o instanceof String ||
                o instanceof ByteSource || o instanceof File || o instanceof InputStream;
    }

    @Override
    public byte[] getBytes() {
        return this.bytes;
    }

    @Override
    public boolean isEmpty() {
        return this.bytes == null || this.bytes.length == 0;
    }

    @Override
    public String toHex() {
        if ( this.cachedHex == null ) {
            this.cachedHex = Hex.encodeToString(getBytes());
        }
        return this.cachedHex;
    }

    @Override
    public String toBase64() {
        if ( this.cachedBase64 == null ) {
            this.cachedBase64 = Base64.encodeToString(getBytes());
        }
        return this.cachedBase64;
    }

    @Override
    public String toString() {
        return toBase64();
    }

```

```
    @Override
    public int hashCode() {
        if (this.bytes == null || this.bytes.length == 0) {
            return 0;
        }
        return Arrays.hashCode(this.bytes);
    }

    @Override
    public boolean equals(Object o) {
        if (o == this) {
            return true;
        }
        if (o instanceof ByteSource) {
            ByteSource bs = (ByteSource) o;
            return Arrays.equals(getBytes(), bs.getBytes());
        }
        return false;
    }

    //will probably be removed in Shiro 2.0.  See SHIRO-203:
    //https://issues.apache.org/jira/browse/SHIRO-203
    private static final class BytesHelper extends CodecSupport {

        /**
         * 嵌套类也需要提供无参构造器
         */
        private BytesHelper() {
        }

        public byte[] getBytes(File file) {
            return toBytes(file);
        }

        public byte[] getBytes(InputStream stream) {
            return toBytes(stream);
        }
    }

}
```

然后在自定义的 `Realm` 中的 `doGetAuthenticationInfo` 方法中，返回 `SimpleAuthenticationInfo` 如下：

```
return new SimpleAuthenticationInfo(user, user.getPassword(),new MyByteSource(user.getUsername()),getName());
```