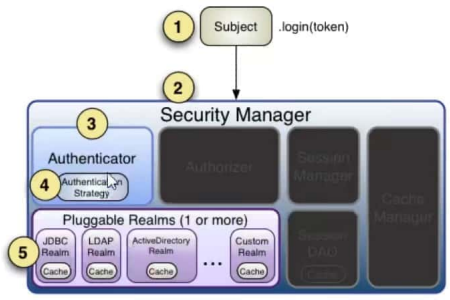
第二章：Shiro 的基本使用方式

一、身份认证

1. 基本流程



(1) 首先调用 Subject.login(token) 进行登录，其会自动委托给 Security Manager，调用之前必须通过 SecurityUtils.setSecurityManager() 设置

(2) SecurityManager 负责真正的身份验证逻辑，它会委托给 Authenticator 进行身份验证

(3) Authenticator 才是真正的身份验证者，Shiro API 中核心的身份认证入口点，此处可以自定义插入自己的实现

(4) Authenticator 可能会委托给相应的 AuthenticationStrategy 进行多 Realm 身份验证，默认 ModularRealmAuthenticator 会调用 AuthenticationStrategy 进行多 Realm 身份验证

(5) Authenticator 会把相应的 token 传入 Realm，从 Realm 获取身份验证信息，如果没有返回/抛出异常表示身份验证失败了。此处可以配置多个 Realm，将按照相应的顺序及策略进行访问

2. 示例：这个示例主要演示了外部信息与 Shiro 内部交互的过程。先获得了 Subject，之后用 token 进行登录

(1) 新建项目并导入依赖

(2) 编写 Shiro.ini

```
#声明用户账号
[users]
jay=123
```

(3) 编写登录操作

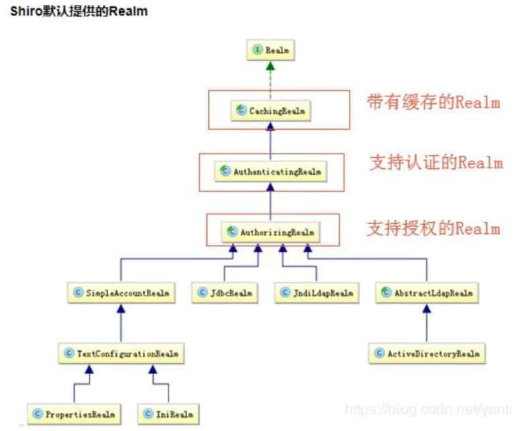
```
public class HelloShiro {

    @Test
    public void shiroLogin(){
        //导入INI配置创建工厂
        Factory<SecurityManager> factory = new IniSecurityManagerFactory( iniResourcePath: "classpath:shiro.ini");
        //工厂构建安全管理器
        SecurityManager securityManager = factory.getInstance();
        //使用工具生效安全管理器
        SecurityUtils.setSecurityManager(securityManager);
        //使用工具获得subject主体
        Subject subject = SecurityUtils.getSubject();
        //构建账户密码
        UsernamePasswordToken usernamePasswordToken = new UsernamePasswordToken( username: "jay", password: "123");
        //使用subject主体去登录
        subject.login(usernamePasswordToken);
        //打印登录信息
        System.out.println("登录结果"+subject.);
    }

}
```

二、Realm

1. Realm 接口介绍：通过 Realm，可以获取安全数据（如用户、角色、权限），也可以完成认证和鉴权的操作



2. 自定义 Realm：需要覆写其中的认证与鉴权方法，这里主要讲解了认证方法。先从 token 中获取登录名，之后数据库中获取对应的密码。

```
/**
 * @Description 认证方法
 */
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {
    //获取登录名
    String loginName = (String) authenticationToken.getPrincipal();
    SecurityService securityService = new SecurityServiceImpl();
    String password = securityService.findPasswordByLoginName(loginName);
    if("".equals(password)){
        throw new UnknownAccountException("账户不存在");
    }
    return new SimpleAuthenticationInfo(loginName,password,getName());
}
```

编写 Shiro.ini，声明自定义的 Realm

```
#声明自定义的realm，且为安全管理器指定realms
[main]
definitionRealm=com.itheima.shiro.realm.DefinitionRealm
securityManager.realms=$definitionRealm
#声明用户账号
#[users]
#jay=123
```

三、编码和散列算法

(一) 编码和解码

1. 作用：加密
2. 过程：创建项目；添加 Shiro 坐标；写编码工具类；进行测试

```
/**
 * @Description: 编码工具类
 */
public class EncodesUtil {

    /**
     * @Description HEX-string-byte[]
     * @param input 输入数组
     * @return 字符串
     */
    public static String encodeHex(byte[] input){
        return Hex.encodeToString(input);
    }

    /**
     * @Description HEX-byte[]-string
     * @param input 输入字符串
     * @return byte数字
     */
    public static byte[] decodeHex(String input){
        return Hex.decode(input);
    }

    /**
     * @Description HEX-string-byte[]
     * @param input 输入数组
     * @return 字符串
     */
    public static String encodeBase64(byte[] input){
        return Base64.encodeToString(input);
    }
}
```

(二) 散列算法

1. 概述

   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salt（即盐）；这样散列的对象是“密码+salt”，这样生成的散列值相对来说更难破解。

2. 方法

```
public class DigestsUtil {

    private static final String SHA1="SHA-1";

    private static final Integer ITERATIONS=512;

    /**
     * @Description sha1摘要算法
     * @param input 明文字符串
     * @param salt 干扰数据
     * @return
     */
    public static String sha1(String input,String salt){
        return  new SimpleHash(SHA1,input,salt,ITERATIONS).toString();
    }

    /**
     * @Description 随机生成salt
     * @return hex编码的slat
     */
    public  static  String generateSalt(){
        SecureRandomNumberGenerator randomNumberGenerator = new SecureRandomNumberGenerator();
        return randomNumberGenerator.nextBytes().toHex();
    }

}
```

(三) 使用散列算法的 Realm

1. 创建项目

2. 创建密文密码

   使用ClientTest的testDigestsUtil创建密码为“123”的password密文和salt密文

```
password:56265d624e481ca62c6dfbc523e6d6fc7932d0d5
salt:845a66ac80174c0e486db9354cf84f9a
```

3. 编写 SecurityService

```
package com.itheima.shiro.service.impl;

import com.itheima.shiro.service.SecurityService;

import java.util.HashMap;
import java.util.Map;

/**
 * @Description: 权限服务层
 */
public class SecurityServiceImpl implements SecurityService {

    @Override
    public Map<String,String> findPassWordByLoginName(String loginName) {
        //模拟数据库中存储的密文信息
       return  DigestsUtil.entryptPassword("123");
    }
}
```

4. 指定密码的匹配方式

   为DefinitionRealm类添加构造方法如下：

```
    /**
     * @Description 构造函数
     */
public DefinitionRealm() {
    //指定密码匹配方式为sha1
    HashedCredentialsMatcher matcher = new HashedCredentialsMatcher(DigestsUtil.SHA1);
    //指定密码迭代次数
    matcher.setHashIterations(DigestsUtil.ITERATIONS);
    //使用父亲方法使匹配方式生效
    setCredentialsMatcher(matcher);
}
```

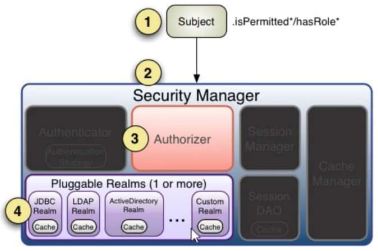
修改DefinitionRealm类的认证doGetAuthenticationInfo方法如下

java

```
    /**
     * @Description 认证接口
     * @param token 传递登录token
     * @return
     */
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws
AuthenticationException {
    //从AuthenticationToken中获得登录名称
    String loginName = (String) token.getPrincipal();
    SecurityService securityService = new SecurityServiceImpl();
    Map<String, String> map = securityService.findPasswordByLoginName(loginName);
    if (map.isEmpty()){
        throw new UnknownAccountException("账户不存在");
    }
    String salt = map.get("salt");
    String password = map.get("password");
    //传递账号和密码:参数1：缓存对象，参数2：明文密码，参数三：字节salt,参数4：当前DefinitionRealm名
称
    return  new SimpleAuthenticationInfo(loginName,password,
ByteSource.Util.bytes(salt),getName());
}
```

四、身份授权

1. 基本流程：在进行鉴权之前需要先完成认证



1、首先调用Subject.isPermitted/hasRole接口，其会委托给SecurityManager。

2、SecurityManager接着会委托给内部组件Authorizer；

3、Authorizer再将其请求委托给我们的Realm去做；Realm才是真正干活的；

4、Realm将用户请求的参数封装成权限对象。再从我们重写的doGetAuthorizationInfo方法中获取从数据库中查询到的权限集合。

5、Realm将用户传入的权限对象，与从数据库中查出来的权限对象，进行一一对比。如果用户传入的权限对象在从数据库中查出来的权限对象中，则返回true，否则返回false。

2. 创建项目
3. 覆写 Realm 中的鉴权方法

在DefinitionRealm中修改doGetAuthorizationInfo方法如下

```
/**
  * @Description 授权方法
  */
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
    //拿到用户认证凭证信息
    String loginName = (String) principals.getPrimaryPrincipal();
    //从数据库中查询对应的角色和资源
    SecurityService securityService = new SecurityServiceImpl();
    List<String> roles = securityService.findRoleByloginName(loginName);
    List<String> permissions = securityService.findPermissionByloginName(loginName);
    //构建资源校验
    SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
    authorizationInfo.addRoles(roles);
    authorizationInfo.addStringPermissions(permissions);
    return authorizationInfo;
}
```

4. 测试

```
@Test
public void testPermissionRealm(){
    Subject subject = shiroLogin();
    //打印登录信息
    System.out.println("登录结果:"+subject.isAuthenticated());
    //————校验当前用户是否拥有管理员的架势
    System.out.println("是否有管理员角色"+subject.hasRole( roleIdentifier: "admin"));
    //————校验当前用户没有的角色
    try {
        subject.checkRole( roleIdentifier: "coder");
        System.out.println("当前用户有coder角色");
    }catch (Exception ex){
        System.out.println("当前用户没有coder角色");
    }
    //————校验当前用户的权限信息
    System.out.println("是否有查看订单的权限"+subject.isPermitted( permission: "order:list"));
    //————校验当前用户满意的权限
    try {
        subject.checkPermission("order:update");
        System.out.println("当前用户有更改的权限");
    }catch (Exception ex){
        System.out.println("当前用户没有coder角色");
    }
}
```