- Subject

```
Subject主体，外部应用与subject进行交互，subject将用户作为当前操作的主体，这个主体：可以是一个通过浏览器请求的用户，也可能是一个运行的程序。Subject在shiro中是一个接口，接口中定义了很多认证授相关的方法，外部程序通过subject进行认证授，而subject是通过SecurityManager安全管理器进行认证授权
```

- SecurityManager

```
SecurityManager权限管理器，它是shiro的核心，负责对所有的subject进行安全管理。通过SecurityManager可以完成subject的认证、授权等，SecurityManager是通过Authenticator进行认证，通过Authorizer进行授权，通过SessionManager进行会话管理等。SecurityManager是一个接口，继承了Authenticator，Authorizer，SessionManager这三个接口
```

- Authenticator

```
Authenticator即认证器，对用户登录时进行身份认证
```

- Authorizer

```
Authorizer授权器，用户通过认证器认证通过，在访问功能时需要通过授权器判断用户是否有此功能的操作权限。
```

- Realm（数据库读取+认证功能+授权功能实现）

```
Realm领域，相当于datasource数据源，securityManager进行安全认证需要通过Realm获取用户权限数据
比如:
    如果用户身份数据在数据库那么realm就需要从数据库获取用户身份信息。
注意:
    不要把realm理解成只是从数据源取数据，在realm中还有认证授权校验的相关的代码。
```

- SessionManager

```
SessionManager会话管理，shiro框架定义了一套会话管理，它不依赖web容器的session，所以shiro可以使用在非web应用上，也可以将分布式应用的会话集中在一点管理，此特性可使它实现单点登录。
```

- SessionDAO

```
SessionDAO即会话dao，是对session会话操作的一套接口
比如:
    可以通过jdbc将会话存储到数据库
    也可以把session存储到缓存服务器
```

- CacheManager

```
CacheManager缓存管理，将用户权限数据存储在缓存，这样可以提高性能
```

- Cryptography

> Cryptography密码管理，shiro提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能

使用subject

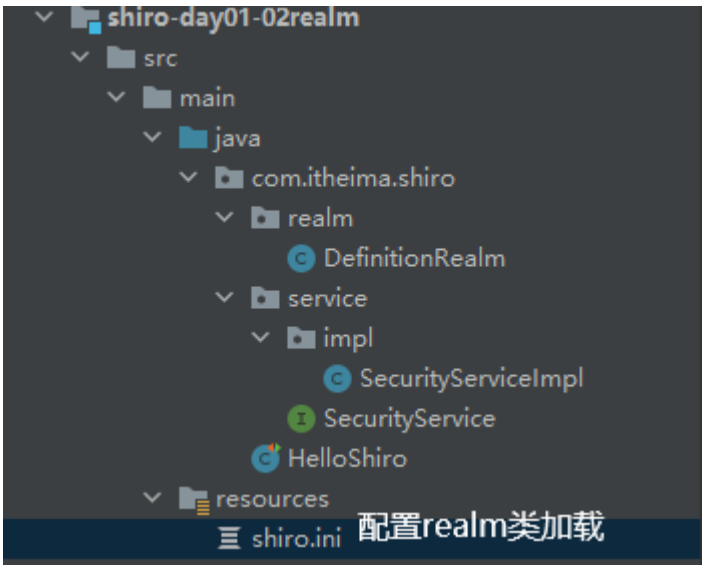
【2.2.4】编写HelloShiro

```
package com.itheima.shiro;

import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.config.IniSecurityManagerFactory;
import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.subject.Subject;
import org.apache.shiro.util.Factory;
import org.junit.Test;

/**
 * @Description: shiro的第一个例子
 */
public class HelloShiro {

    @Test
    public void shiroLogin() {
        //导入权限ini文件构建权限工厂
        Factory<SecurityManager> factory = new
IniSecurityManagerFactory("classpath:shiro.ini");
        //工厂构建安全管理器   -> securityManager
        SecurityManager securityManager = factory.getInstance();
        //使用SecurityUtils工具生效安全管理器 -> SecurityUtils设置securityManager
        SecurityUtils设置securityManager.setSecurityManager(securityManager);
        //使用SecurityUtils工具获得主体 获取subject
        Subject subject = SecurityUtils.getSubject();
        //构建账号token
        UsernamePasswordToken usernamePasswordToken = new UsernamePasswordToken("jay",
"123");
        //登录操作
        subject.login(usernamePasswordToken);
        System.out.println("是否登录成功: " + subject.isAuthenticated());
    }
}
```

配置权限管理管理

在shiro添加配置文件的类加载对象

```
#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
```



编码与解码

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

```
import org.apache.shiro.codec.Base64;
import org.apache.shiro.codec.Hex;

// HEX-byte[]--String转换
Hex.encodeToString();
// HEX-String--byte[]转换
Hex.decode();
// Base64-byte[]--String转换
Base64.encodeToString();
// Base64-String--byte[]转换
Base64.decode();
```

散列算法

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

一般进行散列时最好提供一个salt（盐）

用途: 加密密码 ,数据摘要

shiro支持的散列算法：

Md2Hash、Md5Hash、Sha1Hash、Sha256Hash、Sha384Hash、Sha512Hash

```
import org.apache.shiro.crypto.SecureRandomNumberGenerator;
import org.apache.shiro.crypto.hash.SimpleHash;

private static final String SHA1 = "SHA-1"; // 代表进行加密的算法名称
private static final Integer ITERATIONS =512; // 代表hash迭代次数
SecureRandomNumberGenerator randomNumberGenerator
 = new SecureRandomNumberGenerator(); // 获取随机盐
 private static final String aslt
 = randomNumberGenerator.nextBytes().toHex(); // aslt 代表盐，需要加进一起加密的数据

 SimpleHash(SHA1, input, salt,ITERATIONS).toString();
```

```
package com.itheima.shiro.realm;

import com.itheima.shiro.service.SecurityService;
import com.itheima.shiro.service.impl.SecurityServiceImpl;
import com.itheima.shiro.tools.DigestsUtil;
import org.apache.shiro.authc.*;
import org.apache.shiro.authc.credential.HashedCredentialsMatcher;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.apache.shiro.util.ByteSource;

import java.util.List;
import java.util.Map;

/**
 * @Description:
 */
public class DefinitionRealm extends AuthorizingRealm {

    public DefinitionRealm() {
        //指定密码匹配方式sha1
        HashedCredentialsMatcher hashedCredentialsMatcher = new
HashedCredentialsMatcher(DigestsUtil.SHA1);
        //指定密码迭代此时
        hashedCredentialsMatcher.setHashIterations(DigestsUtil.ITERATIONS);
        //使用父层方法是匹配方式生效
        setCredentialsMatcher(hashedCredentialsMatcher);
    }

    /**
     * @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();
        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");
        return new SimpleAuthenticationInfo(loginName,password,
ByteSource.Util.bytes(salt),getName());
    }

    /**
     * @Description 鉴权方法
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        //拿到用户凭证信息
        String loginName = (String) principalCollection.getPrimaryPrincipal();
        //从数据库中查询对应的角色和权限
        SecurityService securityService = new SecurityServiceImpl();
        List<String> roles = securityService.findRoleByLoginName(loginName);
        List<String> permissions =
securityService.findPermissionByLoginName(loginName);
        //构建资源校验对象
        SimpleAuthorizationInfo simpleAuthorizationInfo = new
SimpleAuthorizationInfo();
        simpleAuthorizationInfo.addRoles(roles);
        simpleAuthorizationInfo.addStringPermissions(permissions);
        return simpleAuthorizationInfo;
    }
}
```

```
@Override
public List<String> findRoleByloginName(String loginName) {
    List<String> list = new ArrayList<>();
    list.add("admin");
    list.add("dev");
    return list;
}

@Override
public List<String>  findPermissionByloginName(String loginName) {
    List<String> list = new ArrayList<>();
    list.add("order:add");
    list.add("order:list");
    list.add("order:del");
    return list;
}
```

Subject用法

```
//判断用户是否已经登录
subject.isAuthenticated() // return boolean
//检查当前用户的角色信息
subject.hasRole("admin")  // 是否有管理员角色
//如果当前用户有此角色，无返回值。若没有此权限，则抛 UnauthorizedException
subject.checkRole("coder");
//检查当前用户的权限信息
subject.isPermitted("order:list");
//如果当前用户有此权限，无返回值。若没有此权限，则抛 UnauthorizedException
subject.checkPermissions("order:add", "order:del");
// 登出
subject.logout();
```

Shiro默认过滤器

【1】认证相关

过滤器	过滤器类	说明	默认
authc	FormAuthenticationFilter	基于表单的过滤器；如“/**=authc”，如果没有登录会跳到相应的登录页面登录	无
logout	LogoutFilter	退出过滤器，主要属性：redirectUrl：退出成功后重定向的地址，如“/logout=logout”	/
anon	AnonymousFilter	匿名过滤器，即不需要登录即可访问；一般用于静态资源过滤；示例“/static/**=anon”	无

【2】授权相关

过滤器	过滤器类	说明	默认
roles	RolesAuthorizationFilter	角色授权拦截器，验证用户是否拥有所有角色；主要属性： loginUrl：登录页面地址（/login.jsp）；unauthorizedUrl：未授权后重定向的地址；示例“/admin/**=roles[admin]”	无
perms	PermissionsAuthorizationFilter	权限授权拦截器，验证用户是否拥有所有权限；属性和roles一样；示例“/user/**=perms["user:create"]”	无
port	PortFilter	端口拦截器，主要属性：port（80）：可以通过的端口；示例“/test= port[80]”，如果用户访问该页面是非80，将自动将请求端口改为80并重定向到该80端口，其他路径/参数等都一样	无
rest	HttpMethodPermissionFilter	rest风格拦截器，自动根据请求方法构建权限字符串（GET=read, POST=create,PUT=update,DELETE=delete,HEAD=read,TRACE=read,OPTIONS=read, MKCOL=create）构建权限字符串；示例“/users=rest[user]”，会自动拼出“user:read,user:create,user:update,user:delete”权限字符串进行权限匹配（所有都得匹配，isPermittedAll）	无
ssl	SslFilter	SSL拦截器，只有请求协议是https才能通过；否则自动跳转会https端口（443）；其他和port拦截器一样；	无

shiro.ini配置信息

```
#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
#用户退出后跳转指定JSP页面
logout.redirectUrl=/login.jsp
#若没有登录，则被authc过滤器重定向到login.jsp页面
authc.loginUrl = /login.jsp
[urls]
/login=anon
#发送/home请求需要先登录
/home= authc
#发送/order/list请求需要先登录
/order-list = roles[admin]
#提交代码需要order:add权限
/order-add = perms["order:add"]
#更新代码需要order:del权限
/order-del = perms["order:del"]
#发送退出请求则用退出过滤器
/logout = logout
```

项目授权

【1】基于代码

【1.1】登录相关

Subject 登录相关方法	描述
isAuthenticated()	返回true 表示已经登录，否则返回false。

【1.2】角色相关

Subject 角色相关方法	描述
hasRole(String roleName)	返回true 如果Subject 被分配了指定的角色，否则返回false。
hasRoles(List roleNames)	返回true 如果Subject 被分配了所有指定的角色，否则返回false。
hasAllRoles(CollectionroleNames)	返回一个与方法参数中目录一致的hasRole 结果的集合。有性能的提高如果许多角色需要执行检查（例如，当自定义一个复杂的视图）。
checkRole(String roleName)	安静地返回，如果Subject 被分配了指定的角色，不然的话就抛出AuthorizationException。
checkRoles(CollectionroleNames)	安静地返回，如果Subject 被分配了所有的指定的角色，不然的话就抛出AuthorizationException。
checkRoles(String... roleNames)	与上面的checkRoles 方法的效果相同，但允许Java5 的var-args 类型的参数

【1.3】资源相关

Subject 资源相关方法	描述
isPermitted(Permission p)	返回true 如果该Subject 被允许执行某动作或访问被权限实例指定的资源，否则返回false
isPermitted(List perms)	返回一个与方法参数中目录一致的isPermitted 结果的集合。
isPermittedAll(Collectionperms)	返回true 如果该Subject 被允许所有指定的权限，否则返回false有性能的提高如果需要执行许多检查（例如，当自定义一个复杂的视图）
isPermitted(String perm)	返回true 如果该Subject 被允许执行某动作或访问被字符串权限指定的资源，否则返回false。
isPermitted(String...perms)	返回一个与方法参数中目录一致的isPermitted 结果的数组。有性能的提高如果许多字符串权限检查需要被执行（例如，当自定义一个复杂的视图）。
isPermittedAll(String...perms)	返回true 如果该Subject 被允许所有指定的字符串权限，否则返回false。
checkPermission(Permission p)	安静地返回，如果Subject 被允许执行某动作或访问被特定的权限实例指定的资源，不然的话就抛出AuthorizationException 异常。
checkPermission(String perm)	安静地返回，如果Subject 被允许执行某动作或访问被特定的字符串权限指定的资源，不然的话就抛出AuthorizationException 异常。
checkPermissions(Collection perms)	安静地返回，如果Subject 被允许所有的权限，不然的话就抛出AuthorizationException 异常。有性能的提高如果需要执行许多检查（例如，当自定义一个复杂的视图）
checkPermissions(String... perms)	和上面的checkPermissions 方法效果相同，但是使用的是基于字符串的权限。

Springboot集成Shiro

、注解方式鉴权

【1】注解介绍

以下为常用注解

注解	说明
@RequiresAuthentication	表明当前用户需是经过认证的用户
@ RequiresGuest	表明该用户需为”guest”用户
@RequiresPermissions	当前用户需拥有指定权限
@RequiresRoles	当前用户需拥有指定角色
@ RequiresUser	当前用户需为已认证用户或已记住用户

例如RoleAction类中我们添加

```
/**
     *@Description: 跳转到角色的初始化页面
     */
    @RequiresRoles(value ={"SuperAdmin","dev"},logical = Logical.OR)
    @RequestMapping(value = "listInitialize")
    public ModelAndView listInitialize(){
        return  new ModelAndView("/role/role-listInitialize");
    }
```

ShiroRedissionSerialize序列化工具

```
package com.itheima.shiro.utils;

import lombok.extern.log4j.Log4j2;
import org.apache.shiro.codec.Base64;

import java.io.*;

/**
 * @Description: 实现shiro会话的序列化存储
 */
@Log4j2
public class ShiroRedissionSerialize {

    public static Object deserialize(String str) {
        if (EmptyUtil.isNullOrEmpty(str)) {
            return null;
        }
        ByteArrayInputStream bis = null;
        ObjectInputStream ois = null;
        Object object=null;
        try {
            bis = new ByteArrayInputStream(EncodesUtil.decodeBase64(str));
            ois = new ObjectInputStream(bis);
            object = ois.readObject();
        } catch (IOException |ClassNotFoundException e) {
            log.error("流读取异常: {}",e);
        } finally {
            try {
                bis.close();
                ois.close();
            } catch (IOException e) {
                log.error("流读取异常: {}",e);
            }
        }
        return object;
    }

    public static String serialize(Object obj) {
```

```
        if (EmptyUtil.isNullOrEmpty(obj)) {
            return null;
        }
        ByteArrayOutputStream bos = null;
        ObjectOutputStream oos = null;
        String base64String = null;
        try {
            bos = new ByteArrayOutputStream();
            oos = new ObjectOutputStream(bos);
            oos.writeObject(obj);
            base64String = EncodesUtil.encodeBase64(bos.toByteArray());
        } catch (IOException e) {
            log.error("流写入异常：{}",e);
        } finally {
            try {
                bos.close();
                oos.close();
            } catch (IOException e) {
                log.error("流写入异常：{}",e);
            }
        }
        return base64String;
    }
}
```

Configuration配置

```
package com.ihrm.system;

import com.ihrm.common.shiro.realm.IhrmRealm;
import com.ihrm.common.shiro.session.CustomSessionManager;
import com.ihrm.system.shiro.realm.UserRealm;
import org.apache.shiro.mgt.SecurityManager;
import
org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;
import org.crazycake.shiro.RedisCacheManager;
import org.crazycake.shiro.RedisManager;
import org.crazycake.shiro.RedisSessionDAO;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.LinkedHashMap;
import java.util.Map;

@Configuration
public class ShiroConfiguration {
```

```
    //1.创建realm
    @Bean
    public IhrmRealm getRealm() {
        return new UserRealm();
    }

    //2.创建安全管理器
    @Bean
    public SecurityManager getSecurityManager(IhrmRealm realm) {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        securityManager.setRealm(realm);

        //将自定义的会话管理器注册到安全管理器中
        securityManager.setSessionManager(sessionManager());
        //将自定义的redis缓存管理器注册到安全管理器中
        securityManager.setCacheManager(cacheManager());

        return securityManager;
    }

    //3.配置shiro的过滤器工厂

    /**
     * 再web程序中，shiro进行权限控制全部是通过一组过滤器集合进行控制
     *
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {
        //1.创建过滤器工厂
        ShiroFilterFactoryBean filterFactory = new ShiroFilterFactoryBean();
        //2.设置安全管理器
        filterFactory.setSecurityManager(securityManager);
        //3.通用配置（跳转登录页面，未授权跳转的页面）
        filterFactory.setLoginUrl("/autherror?code=1");//跳转url地址
        filterFactory.setUnauthorizedUrl("/autherror?code=2");//未授权的url
        //4.设置过滤器集合
        Map<String,String> filterMap = new LinkedHashMap<>();
        //anon -- 匿名访问
        filterMap.put("/sys/login","anon");
        filterMap.put("/autherror","anon");
        //注册
        //authc -- 认证之后访问（登录）
        filterMap.put("/**","authc");
        //perms -- 具有某中权限（使用注解配置授权）
        filterFactory.setFilterChainDefinitionMap(filterMap);

        return filterFactory;
    }


    @Value("${spring.redis.host}")
    private String host;
    @Value("${spring.redis.port}")
```

```
    private int port;

    /**
     * 1.redis的控制器，操作redis
     */
    public RedisManager redisManager() {
        RedisManager redisManager = new RedisManager();
        redisManager.setHost(host);
        redisManager.setPort(port);
        return redisManager;
    }

    /**
     * 2.sessionDao
     */
    public RedisSessionDAO redisSessionDAO() {
        RedisSessionDAO sessionDAO = new RedisSessionDAO();
        sessionDAO.setRedisManager(redisManager());
        return sessionDAO;
    }

    /**
     * 3.会话管理器
     */
    public DefaultWebSessionManager sessionManager() {
        CustomSessionManager sessionManager = new CustomSessionManager();
        sessionManager.setSessionDAO(redisSessionDAO());
        //禁用cookie
        sessionManager.setSessionIdCookieEnabled(false);
        //禁用url重写   url;jsessionid=id
        sessionManager.setSessionIdUrlRewritingEnabled(false);
        return sessionManager;
    }

    /**
     * 4.缓存管理器
     */
    public RedisCacheManager cacheManager() {
        RedisCacheManager redisCacheManager = new RedisCacheManager();
        redisCacheManager.setRedisManager(redisManager());
        return redisCacheManager;
    }



    //开启对shior注解的支持
    @Bean
    public AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor(SecurityManager securityManager) {
        AuthorizationAttributeSourceAdvisor advisor = new
AuthorizationAttributeSourceAdvisor();
        advisor.setSecurityManager(securityManager);
```

```
            return advisor;
        }
    }
```

###