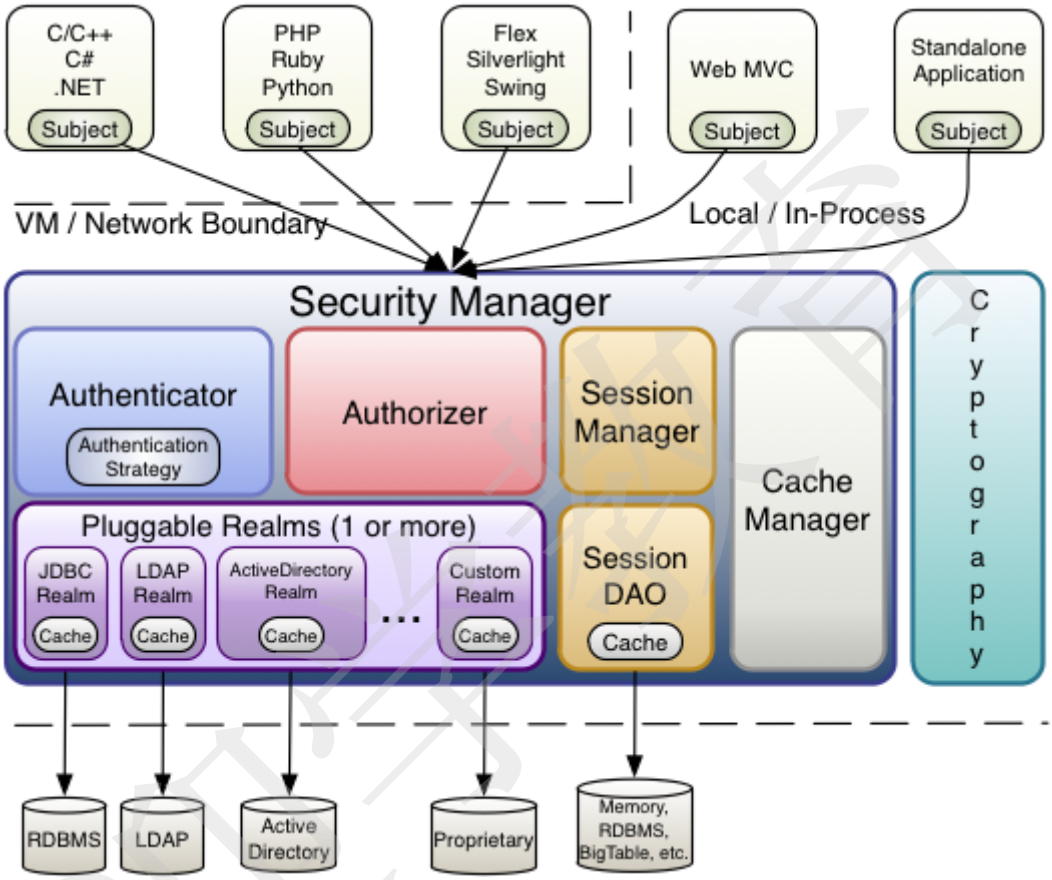
第七章 Spring Boot 权限框架零基础入门到实战

1.从零开始认识 shiro

1.shiro 简介

shiro是apache的一个开源框架，而且呢是一个权限管理的框架，用于实现用户认证、用户授权。spring 中也有一个权限框架 spring security (原名Acegi)，它和 spring 依赖过于紧密，没有 shiro 使用简单。shiro 不依赖于 spring，shiro 不仅可以实现 web应用的权限管理，还可以实现c/s系统，分布式系统权限管理，shiro属于轻量框架，越来越多企业项目开始使用shiro。使用shiro实现系统的权限管理，有效提高开发效率，从而降低开发成本。

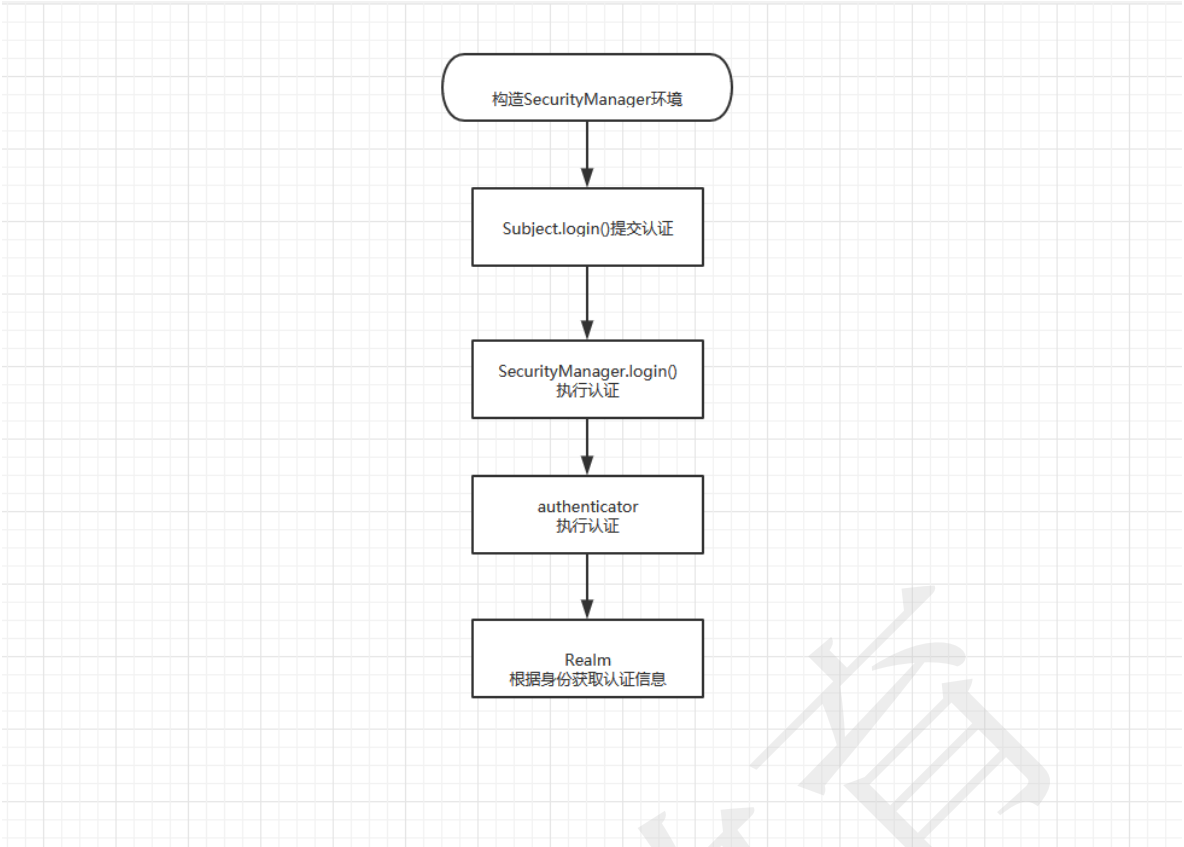
2.shiro 基本功能点



- subject：主体，可以是用户也可以是程序，主体要访问系统，系统需要对主体进行认证、授权。
- security Manager：安全管理器，主体进行认证和授权都是通过securityManager进行。
- authenticator：认证器，主体进行认证最终通过authenticator进行的。
- authorizer：授权器，主体进行授权最终通过authorizer进行的。
- sessionManager：web应用中一般是用web容器对session进行管理，shiro也提供一套session管理的方式。
- SessionDao： 通过SessionDao管理session数据，针对个性化的session数据存储需要使用sessionDao。
- cache Manager：缓存管理器，主要对session和授权数据进行缓存，比如将授权数据通过cacheManager进行缓存管理，和ehcache整合对缓存数据进行管理。
- Cryptography：加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储。
- realm：域，领域，相当于数据源，通过realm存取认证、授权相关数据。

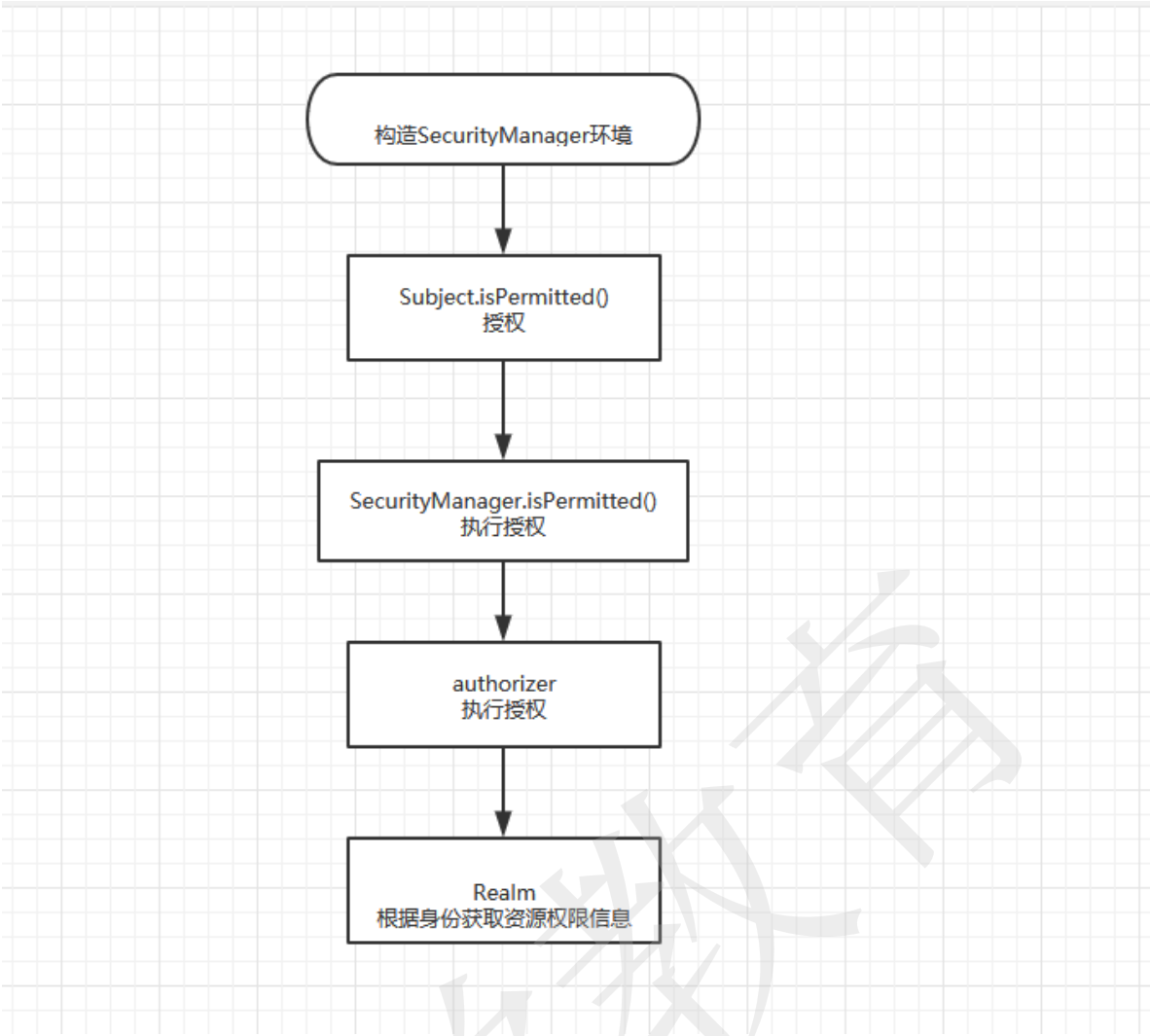
记住一点，Shiro 不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给Shiro 即可。

3.认证流程



1. 构建SecurityManager环境
2. 主体提交认证
3. SecurityManager 处理
4. 流转到 Authenticator 执行认证
5. 通过 Realm 获取相关的用户信息（获取验证数据进行验证）

4.授权流程

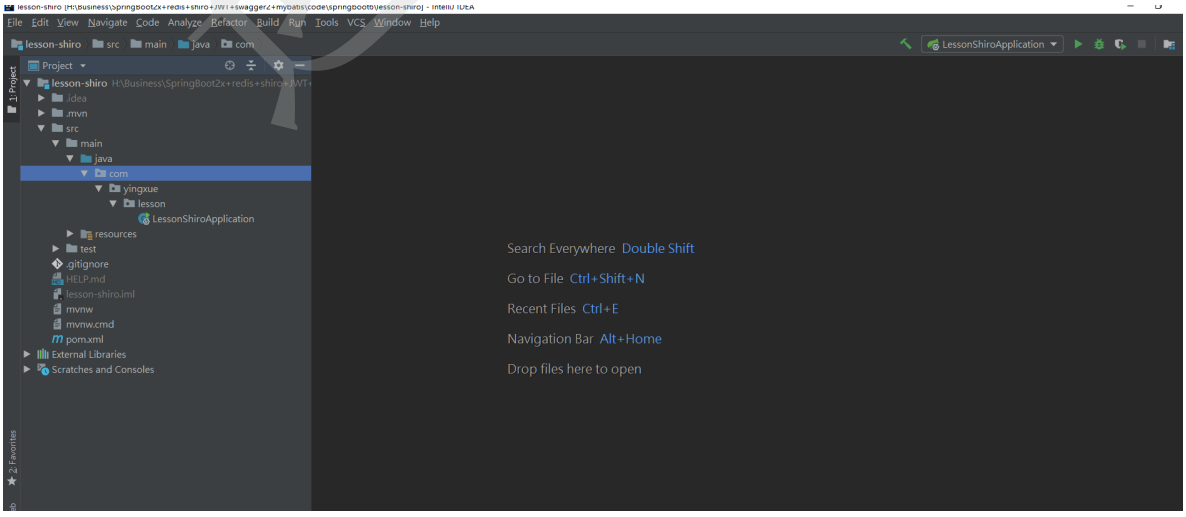


1. 创建构建SecurityManager环境
2. 主体提交授权认证
3. SecurityManager 处理
4. 流转到 Authorizor 授权器执行授权认证
5. 通过 Realm 从数据库或配置文件获取角色权限数据返回给授权器，进行授权。

2.Spring Boot 集成 shiro 快速入门

2.1 idea Spring Initializr 快速创建项目

New----->Project----->Spring Initializr 一路默认后就创建好一个以spring boot构建的web项目了



2.2 加入shiro 所需的依赖

- 加入jar包

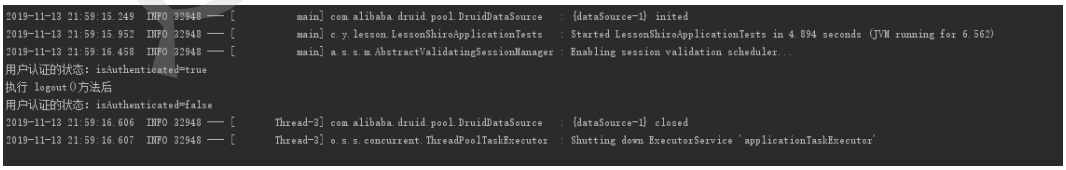
```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.4.1</version>
</dependency>
```

2.3 shiro 用户认证

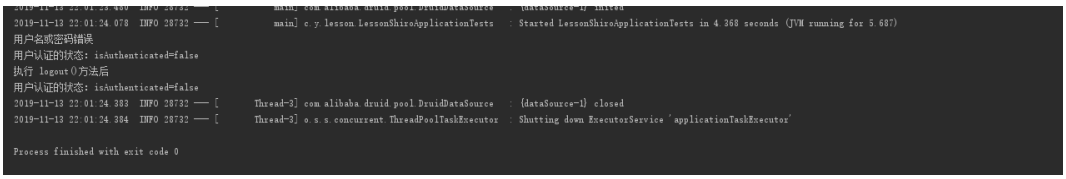
- 在junit单元测试类创建shiro认证的测试方法

```
    @Test
    public void authentication(){
        //构建SecurityManager环境
        DefaultWebSecurityManager defaultSecurityManager = new DefaultWebSecurityManager();
        //创建一个SimpleAccountRealm 域
        SimpleAccountRealm simpleAccountRealm = new SimpleAccountRealm();
        //添加一个测试账号(后面可以做成读取动态读取数据库)
        simpleAccountRealm.addAccount("zhangsan", "123456");
        //设置Realm
        defaultSecurityManager.setRealm(simpleAccountRealm);
        SecurityUtils.setSecurityManager(defaultSecurityManager);
        //获取主体
        Subject subject = SecurityUtils.getSubject();
        //用户名和密码(用户输入的用户名密码)生成token
        UsernamePasswordToken token = new UsernamePasswordToken("zhangsan", "123456");
        try {
            //进行登入(提交认证)
            subject.login(token);

        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名密码不匹配");
        }catch (LockedAccountException exception){
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }

        System.out.println("用户认证的状态: isAuthenticated=" + subject.isAuthenticated());
        //登出logout
        System.out.println("执行 logout()方法后");
        subject.logout();
        System.out.println("用户认证的状态: isAuthenticated=" + subject.isAuthenticated());

    }
```

- 测试
  - 当用户输入用户名密码为“zhangsan”、“123456” 程序输出



  - 当用户输入的用户名密码为‘zhangsan“、”1234567“(密码错误) 程序输出



  - 当用户输入的用户名密码为"zhangsan1"、"123456"(用户名错误) 程序输出

2.4 shiro 用户授权

上面的小 demo 是 shiro 实现认证的一个过程，做权限管理的时候，分为两块，一个认证，一个是授权，认证是判断用户是否账号密码正确，授权是判断用户登入以后有什么权限

- 在 junit 单元测试类创建 shiro 授权的测试方法

```
@Test
    public void authorization(){
        //构建SecurityManager环境
        DefaultSecurityManager defaultSecurityManager = new DefaultSecurityManager();
        //创建一个SimpleAccountRealm 域
        SimpleAccountRealm simpleAccountRealm = new SimpleAccountRealm();
        //添加一个测试账号、和所拥有的角色(后面可以做成读取动态读取数据库)
        simpleAccountRealm.addAccount("zhangsan", "123456","admin","user");
        //设置Realm
        defaultSecurityManager.setRealm(simpleAccountRealm);
        SecurityUtils.setSecurityManager(defaultSecurityManager);
        //获取主体
        Subject subject = SecurityUtils.getSubject();
        //用户名和密码(用户输入的用户名密码)生成token
        UsernamePasswordToken token = new UsernamePasswordToken("zhangsan", "123456");
        try {
            //进行登入(提交认证)
            subject.login(token);
            subject.checkRoles("admin","user");

        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名密码不匹配");
        }catch (LockedAccountException exception){
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }catch ( UnauthorizedException ae ) {
            System.out.println("用户没有权限");
        }
        System.out.println("用户认证的状态：isAuthenticated=" + subject.isAuthenticated());
        //登出logout
        System.out.println("执行 logout()方法后");
        subject.logout();
        System.out.println("用户认证的状态：isAuthenticated=" + subject.isAuthenticated());

    }
```

- 测试
  - 当用户要 subject.checkRoles("admin","user"); 检测是否拥有 admin、user 角色的时候 程序输出

  - 当用户要 subject.checkRoles("test","user"); 检测是否拥有 test、user 角色的时候 程序输出

  - checkRoles 方法就是检测用户是否拥有传入的角色，即只要有一个不是用户所拥有的角色就会抛出异常。请看下列源码。

```
protected void checkRoles(Collection<String> roles, AuthorizationInfo info) {
    if (roles != null && !roles.isEmpty()) {
        Iterator var3 = roles.iterator();

        while(var3.hasNext()) {
            String roleName = (String)var3.next();
            this.checkRole(roleName, info);
        }
    }
}
```

```
protected void checkRole(String role, AuthorizationInfo info) {
    if (!this.hasRole(role, info)) {
        String msg = "User does not have role [" + role + "]";
        throw new UnauthorizedException(msg);
    }
}
```

3. SpringBoot 使用IniRealm进行认证授权

上节课中我们利用`SimpleAccountRealm`在程序中写死了用户安全数据，接下来我们使用`.ini`将数据移到配置文件中。

3.1概述

IniRealm是Shiro提供一种Realm实现。用户、角色、权限等信息集中在一个.ini文件那里。

3.2配置 .ini 文件

- 在 resources 目录下创建一个 shiro.ini 文件

```
#账号信息
[users]
#账号=密码，角色
test=123456,test
admin=123456,admin

#角色信息
[roles]
test=user:list,user:deleted,user:edit
admin= *
```

3.3 测试

- 在 junit 单元测试类创建 testIniRealm 方法

```
@Test
    public void testIniRealm(){
        //配置文件中的用户权限信息，文件在类路径下
        IniRealm iniRealm = new IniRealm("classpath:shiro.ini");

        //1,构建SecurityManager环境
        DefaultSecurityManager defaultSecurityManager = new DefaultSecurityManager();
        //设置Realm
        defaultSecurityManager.setRealm(iniRealm);
```

```
        SecurityUtils.setSecurityManager(defaultSecurityManager);
        //获取主体
        Subject subject = SecurityUtils.getSubject();
        //用户名和密码的token
        UsernamePasswordToken token = new UsernamePasswordToken("test", "123456");
        try {
            //2,主体提交认证请求
            subject.login(token);
            System.out.println("用户认证的状态：isAuthenticated=" + subject.isAuthenticated());
            //检查是否有角色
            subject.checkRoles("test");
            System.out.println("有admin角色");
            //检查是否有权限
            subject.checkPermissions("user:list");
            System.out.println("有user:delete权限");

        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名或密码错误");
        }catch (LockedAccountException exception){
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }catch ( UnauthorizedException ae ) {
            System.out.println("用户没有权限");
        }

    }
```

- 当用户 test 进入到程序时候我们首先用 IniRealm 读取 shiro.ini 配置获得 test 用户的一些安全信息。

  subject.checkRoles("test")：即是判断该用户是否拥有 test 角色。

  subject.checkPermissions("user:list")：即是检查用户是否拥有 user:list 的权限

  很明显test 这个用户都满足这些条件，最后程序输出

- 假如把 subject.checkRoles("test") 改成 subject.checkRoles("amin") 即验证用户 test 是否拥有 admin 角色呢？

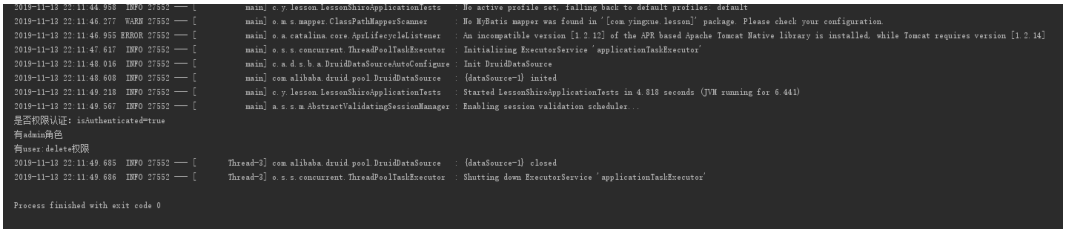
  这个时候很明显会抛出异常，因为我们在 shiro.ini 配置里面只给 test 用户 配置了 test 的角色。

  最后程序输出

- 假如是用户 admin 登录进来呢？

  很明显不会抛异常，因为用户 admin 拥有了 admin 这个角色，而 admin 这角色设置了 '*' 的通配符即拥有所有的权限所以不会抛出异常。

  程序输出



4. Spring Boot 使用 JdbcRealm 进行认证授权

4.1概述

上一节课我们主要讲了把用户安全信息(相应的角色/权限)配置在 .ini 文件，使用 IniRealm 去读取 .ini 文件获得用户的安全信息。也是有局限性的。因为我们得事先把所有用户信息配置在.ini 文件，这样显然是行不通的，我们的系统用户都是动态的不固定的，它的一些用户信息权限信息都是变化的，所以固定在.ini 配置文件显然是行不通的。这些数据通常我们都是把它存入到DB 中，那shiro 有没有提供直接从DB读取用户安全信息的域呢？(Realm)

shiro 作为一个优秀的开源框架，显然是可以的。

下面就该 JdbcRealm 出场了。

4.2 JdbcRealm 实践

4.2.1 数据库驱动

很明显看到 jdbc 就明白是跟数据有关的吧，所以我们要引入 mysql 数据库驱动

```
<!--数据库驱动-->
  <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.47</version>
  </dependency>
```

4.2.2 数据源

有了mysql数据库驱动是不是得加个数据源呀？这里我们用阿里的druid

```
  <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.1.10</version>
  </dependency>
```

4.2.3 数据库表结构

下面我们创建一个名为 shiro 的数据库、分别创建三张表 users、user_roles、roles_permissions

```
CREATE DATABASE IF NOT EXISTS shiro DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
```

- users

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(25) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- user_roles

```
CREATE TABLE `user_roles` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `role_name` varchar(25) DEFAULT NULL,
  `username` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- roles_permissions

```
CREATE TABLE `roles_permissions` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `permission` varchar(255) DEFAULT NULL,
  `role_name` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- 分别插入如下几条数据

```
USE shiro;
INSERT INTO `shiro`.`users` (`id`, `username`, `password`) VALUES ('1', 'admin', '123456');
INSERT INTO `shiro`.`users` (`id`, `username`, `password`) VALUES ('2', 'test', '123456');
INSERT INTO `shiro`.`user_roles` (`id`, `role_name`, `username`) VALUES ('1', 'admin', 'admin');
INSERT INTO `shiro`.`user_roles` (`id`, `role_name`, `username`) VALUES ('2', 'test', 'test');
INSERT INTO `shiro`.`roles_permissions` (`id`, `permission`, `role_name`) VALUES ('1',
'user:deleted', 'test');
INSERT INTO `shiro`.`roles_permissions` (`id`, `permission`, `role_name`) VALUES ('2',
'user:list', 'test');
INSERT INTO `shiro`.`roles_permissions` (`id`, `permission`, `role_name`) VALUES ('3', '*',
'admin');
INSERT INTO `shiro`.`roles_permissions` (`id`, `permission`, `role_name`) VALUES ('4',
'user:edit', 'test');
```

4.2.4 创建 testJdbcRealm方法

```
@Test
    public void testJdbcRealm(){
        //配置数据源
        DruidDataSource dataSource = new DruidDataSource();
        dataSource.setUrl("jdbc:mysql://localhost:3306/shiro");
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUsername("root");
        dataSource.setPassword("root");
        //配置文件中的用户权限信息，文件在类路径下
        JdbcRealm jdbcRealm = new JdbcRealm();
        jdbcRealm.setDataSource(dataSource);
        //使用JdbcRealm下面的值需要为true不然无法查询用户权限
        jdbcRealm.setPermissionsLookupEnabled(true);

        //1,构建SecurityManager环境
        DefaultSecurityManager defaultSecurityManager = new DefaultSecurityManager();
        //设置Realm
        defaultSecurityManager.setRealm(jdbcRealm);


        SecurityUtils.setSecurityManager(defaultSecurityManager);
        //获取主体
        Subject subject = SecurityUtils.getSubject();
        //用户名和密码的token
        UsernamePasswordToken token = new UsernamePasswordToken("admin", "123456");
        try {
            //2,主体提交认证请求
            subject.login(token);
            System.out.println("认证状态：isAuthenticated=" + subject.isAuthenticated());
            //检查是否有角色
            subject.checkRoles("admin");
            System.out.println("有admin角色");
```

```
            //检查是否有权限
            subject.checkPermissions("user:delete");
            System.out.println("有user:delete权限");

        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名或密码错误");
        }catch (LockedAccountException exception){
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }catch ( UnauthorizedException ae ) {
            System.out.println("用户没有权限");
        }
    }
```

4.2.5 测试

当用户 admin 登录进来的时候 程序输出

```
2019-11-19 11:24:09.536  INFO 63368 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2019-11-19 11:24:09.962  INFO 63368 --- [           main] c.y.lesson.LessonShiroApplicationTests   : Started LessonShiroApplicationTests in 3.08 seconds (JVM running for 4.164)
2019-11-19 11:24:10.408  INFO 63368 --- [           main] com.alibaba.druid.pool.DruidDataSource   : {dataSource-1} inited
2019-11-19 11:24:10.694  INFO 63368 --- [           main] a.s.s.m.AbstractValidatingSessionManager : Enabling session validation scheduler...
认证状态: isAuthenticated=true
有admin角色
有user:delete权限
2019-11-19 11:24:10.819  INFO 63368 --- [       Thread-2] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorService 'applicationTaskExecutor'

Process finished with exit code 0
```

当用户 test 登录进来的时候 程序输出

```
UsernamePasswordToken token = new UsernamePasswordToken("test", "123456");
```

```
2019-11-19 11:25:34.861  INFO 61980 --- [           main] a.s.s.m.AbstractValidatingSessionManager : Enabling session validation scheduler...
认证状态: isAuthenticated=true
用户没有权限
2019-11-19 11:25:34.963  INFO 61980 --- [       Thread-2] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorService 'applicationTaskExecutor'

Process finished with exit code 0
```

4.2.6 源码分析

看了这个例子，为什么我们没有写 sql 怎么查询用户信息和角色，权限信息的呢？我们来看JdbcRealm源码

```
public class JdbcRealm extends AuthorizingRealm {
    protected static final String DEFAULT_AUTHENTICATION_QUERY = "select password from users where username = ?";
    protected static final String DEFAULT_SALTED_AUTHENTICATION_QUERY = "select password, password_salt from users where username = ?";
    protected static final String DEFAULT_USER_ROLES_QUERY = "select role_name from user_roles where username = ?";
    protected static final String DEFAULT_PERMISSIONS_QUERY = "select permission from roles_permissions where role_name = ?";
    private static final Logger log = LoggerFactory.getLogger(JdbcRealm.class);
    protected DataSource dataSource;
    protected String authenticationQuery = "select password from users where username = ?";
    protected String userRolesQuery = "select role_name from user_roles where username = ?";
    protected String permissionsQuery = "select permission from roles_permissions where role_name = ?";
    protected boolean permissionsLookupEnabled = false;
    protected JdbcRealm.SaltStyle saltStyle;
```

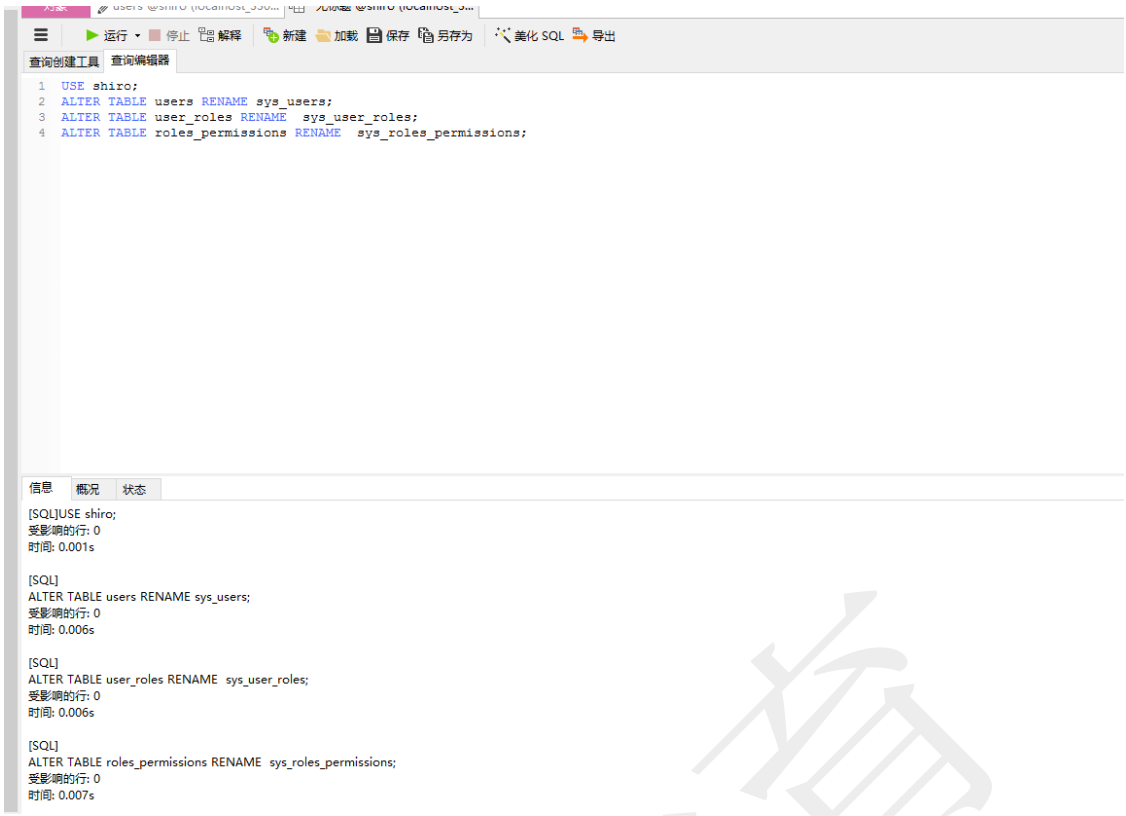
```
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationExceptio
    UsernamePasswordToken upToken = (UsernamePasswordToken)token;
    String username = upToken.getUsername();
    if (username == null) {
        throw new AccountException("Null usernames are not allowed by this realm.");
    } else {
        Connection conn = null;
        SimpleAuthenticationInfo info = null;

        try {
            String salt;
            try {
                conn = this.dataSource.getConnection();
                String password = null;
                salt = null;
                switch(this.saltStyle) {
                case NO_SALT:
                    password = this.getPasswordForUser(conn, username)[0];
                    break;
                case CRYPT:
                    throw new ConfigurationException("Not implemented yet");
                case COLUMN:
                    String[] queryResults = this.getPasswordForUser(conn, username);
                    password = queryResults[0];
                    salt = queryResults[1];
```

```
private String[] getPasswordForUser(Connection conn, String username) throws SQLException {
    boolean returningSeparatedSalt = false;
    String[] result;
    switch(this.saltStyle) {
    case NO_SALT:
    case CRYPT:
    case EXTERNAL:
        result = new String[1];
        break;
    case COLUMN:
    default:
        result = new String[2];
        returningSeparatedSalt = true;
    }

    PreparedStatement ps = null;
    ResultSet rs = null;

    try {
        ps = conn.prepareStatement(this.authenticationQuery);
        ps.setString( parameterIndex: 1, username);
        rs = ps.executeQuery();

        for(boolean foundResult = false; rs.next(); foundResult = true) {
            if (foundResult) {
                throw new AuthenticationException("More than one user row found for user [" + username + "]. Usern
            }
```

从以上源码得知其实 JdbcRealm 已经帮我们写好查询语句了，所以我们就要在数据库创建与之对应的表结构，这样才能查出数据，但是有的同学可能就有疑问了，这里只能使用他默认的 sql，在实际的开发中，我们不可能就简单的使用 JdbcRealm 默认的 sql 语句，而是自己自定义的 sql 语句，更多时候我们的数据库以及数据表都是根据业务需要自己创建的。

1. 首先把上面三张表分别修改一下表名

```
USE shiro;
ALTER TABLE users RENAME sys_users;
ALTER TABLE user_roles RENAME  sys_user_roles;
ALTER TABLE roles_permissions RENAME  sys_roles_permissions;
```

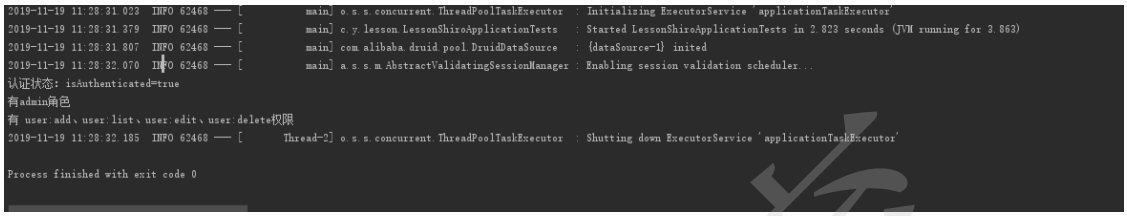


2. 新建testNewJdbcRealm

```
@Test
  public void testNewJdbcRealm(){
      //配置数据源
      DruidDataSource dataSource = new DruidDataSource();
      //如果使用的是新版的驱动 配置driver的时候要注意
      dataSource.setUrl("jdbc:mysql://localhost:3306/shiro?serverTimezone=UTC");
      dataSource.setDriverClassName("com.mysql.jdbc.Driver");
      dataSource.setUsername("root");
      dataSource.setPassword("root");
      //配置文件中的用户权限信息，文件在类路径下
      JdbcRealm jdbcRealm = new JdbcRealm();
      jdbcRealm.setDataSource(dataSource);
      //使用JdbcRealm下面的值需要为true不然无法查询用户权限
      jdbcRealm.setPermissionsLookupEnabled(true);
      //使用自定义sql查询用户信息
      String sql="select password from sys_users where username = ?";
      jdbcRealm.setAuthenticationQuery(sql);
      String roleSql = "select role_name from sys_user_roles where username = ?";
      jdbcRealm.setUserRolesQuery(roleSql);
      String permissionsSql = "select permission from sys_roles_permissions where role_name = ?";
      jdbcRealm.setPermissionsQuery(permissionsSql);

      //1,构建SecurityManager环境
      DefaultSecurityManager defaultSecurityManager = new DefaultSecurityManager();
      //设置Realm
      defaultSecurityManager.setRealm(jdbcRealm);


      SecurityUtils.setSecurityManager(defaultSecurityManager);
      //获取主体
      Subject subject = SecurityUtils.getSubject();
      //用户名和密码的token
      UsernamePasswordToken token = new UsernamePasswordToken("admin", "123456");
      try {
          //2,主体提交认证请求
          subject.login(token);
          System.out.println("认证状态：isAuthenticated=" + subject.isAuthenticated());
          //检查是否有角色
          subject.checkRoles("admin");
          System.out.println("有admin角色");
          //检查是否有权限
          subject.checkPermissions("user:delete","user:edit","user:list","user:add");
```

```
            System.out.println("有 user:add、user:list、user:edit、user:delete权限");

        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名或密码错误");
        }catch (LockedAccountException exception){
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }catch ( UnauthorizedException ae ) {
            System.out.println("用户没有权限");
        }
    }
```

3. 最后程序输出



5. Spring Boot 使用自定义 Realm 进行认证授权

5.1 概述

虽然 jdbcRealm 已经实现了从数据库中获取用户的验证信息，但是 jdbcRealm灵活性也是稍差一些的，如果要实现自己的一些特殊应用时将不能支持，这个时候可以通过自定义realm来实现身份的认证功能。

通常自定义Realm只需要继承：AuthorizingRealm重写 doGetAuthenticationInfo(用户认证)、doGetAuthorizationInfo(用户授权) 这两个方法即可。

5.2 自定义 Realm

5.2.1 创建 CustomRealm

```
package com.yingxue.lesson.shiro;


import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
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 java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @ClassName: CustomRealm
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class CustomRealm extends AuthorizingRealm {
    /**
     * 模拟数据库中的用户名和密码
     */
    private Map<String, String> userMap =new HashMap<>();

    /**
     * 使用代码块初始化数据
     */
```

```
    {
        userMap.put("admin", "123456");
        userMap.put("test","123456");
    }

    /**
     * 用户授权
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param principalCollection
     * @return       org.apache.shiro.authz.AuthorizationInfo
     * @throws
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        //这个就是SimpleAuthenticationInfo(username,password,getName()); 第一个参数
        String username= (String) SecurityUtils.getSubject().getPrincipal();
        System.out.println("开始执行*******doGetAuthorizationInfo方法啦****");
        System.out.println(SecurityUtils.getSubject().getPrincipal());
        String username = (String) principalCollection.getPrimaryPrincipal();
        //从数据库或者缓存中获取角色数据
        List<String> roles = getRolesByUsername(username);
        //从数据库或者缓存中获取权限数据
        List<String> permissions = getPerminssionsByUsername(username);
        //创建AuthorizationInfo，并设置角色和权限信息
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
        authorizationInfo.addStringPermissions(permissions);
        authorizationInfo.addRoles(roles);
        return authorizationInfo;

    }
    /**
     * 用户人证
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param authenticationToken
     * @return       org.apache.shiro.authc.AuthenticationInfo
     * @throws
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws
AuthenticationException {
        System.out.println("开始执行*******doGetAuthenticationInfo方法啦****");
        //获取登录用户名
        String username = (String) authenticationToken.getPrincipal();
        //通过用户名到数据库获取用户信息
        String password = getPasswordByUsername(username);
        if (password == null) {
            return null;
        }
        SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,password,getName());
        return authenticationInfo;

    }

    /**
     * 模拟通过数据库获取权限数据
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param username
     * @return       java.util.List<java.lang.String>
     * @throws
     */
    private List<String> getPerminssionsByUsername(String username) {
        List<String> permissions = new ArrayList<>();
        /**
         * 只有是 admin 用户才有 新增、删除权限
         */
        if(username.equals("admin")){
            permissions.add("user:delete");
```

```
            permissions.add("user:add");
        }
        permissions.add("user:edit");
        permissions.add("user:list");
        return permissions;
    }

    /**
     * 模拟通过数据库获取用户角色信息
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param username
     * @return       java.util.List<java.lang.String>
     * @throws
     */
    private List<String> getRolesByUsername(String username) {
        List<String> roles = new ArrayList<>();
        if(username.equals("admin")){
            roles.add("admin");
        }
        roles.add("test");
        return roles;
    }
    /**
     * 通过用户名查询密码，模拟数据库查询
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param username
     * @return       java.lang.String
     * @throws
     */
    private String getPasswordByUsername(String username) {

        return userMap.get(username);
    }


}
```

5.2.2 创建 testCustomRealm 方法

```
    @Test
    public void testCustomRealm(){
        CustomRealm customRealm=new CustomRealm();
        //1,构建SecurityManager环境
        DefaultSecurityManager defaultSecurityManager = new DefaultSecurityManager();
        //设置Realm
        defaultSecurityManager.setRealm(customRealm);


        SecurityUtils.setSecurityManager(defaultSecurityManager);
        //获取主体
        Subject subject = SecurityUtils.getSubject();
        //用户名和密码的token
        UsernamePasswordToken token = new UsernamePasswordToken("admin", "123456");
        try {
            //2,主体提交认证请求
            subject.login(token);
            System.out.println("认证状态：isAuthenticated=" + subject.isAuthenticated());
            //检查是否有角色
            subject.checkRoles("admin");
            System.out.println("有admin角色");
            //检查是否有权限
            subject.checkPermissions("user:delete","user:edit","user:list","user:add");
            System.out.println("有 user:add、user:list、user:edit、user:delete权限");

        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名或密码错误");
        }catch (LockedAccountException exception){
```

```
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }catch ( UnauthorizedException ae ) {
            System.out.println("用户没有权限");
        }
    }
```

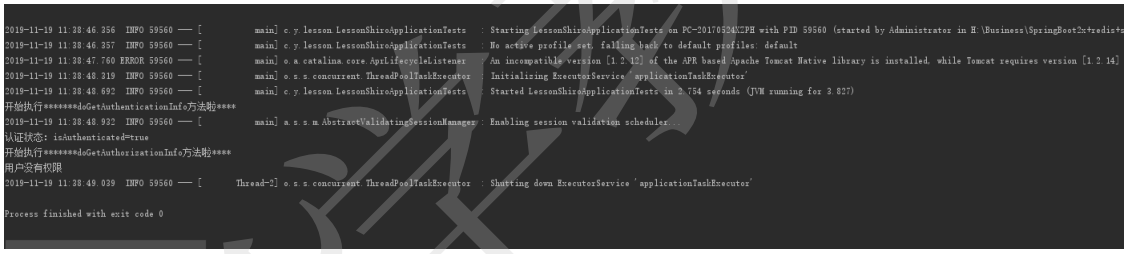
5.2.3 测试

1. 当用户 admin 登录进来的时候 程序输出

```
UsernamePasswordToken token = new UsernamePasswordToken("admin", "123456");
```

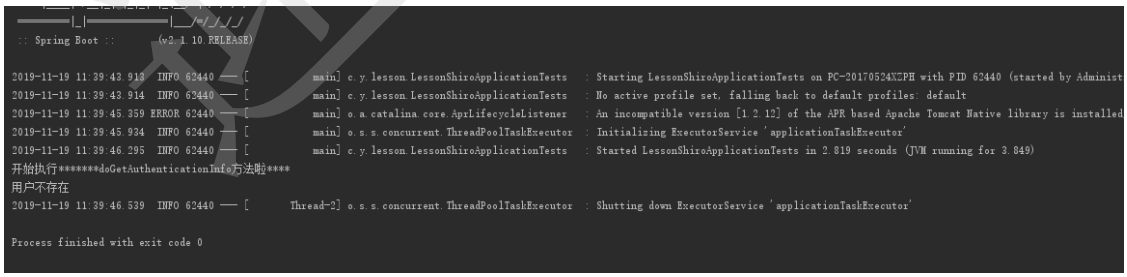
2. 当用户 test 登录进来的时候 程序输出

```
UsernamePasswordToken token = new UsernamePasswordToken("test", "123456");
```



3. 当用户 dev 登陆进来的时候 程序输出

```
UsernamePasswordToken token = new UsernamePasswordToken("dev", "123456");
```



通过上述验证过程我们可以发现，shiro 更多的是帮助我们完成验证过程。我们需要从数据库查询当前用户的角色、权限，把这些信息告诉 shiro 框架。当我们执行用户认证的时候首先调用 doGetAuthenticationInfo 进行用户认证，当我们要校验权限的时候 就会执行 doGetAuthorizationInfo 进行授权操作。

6. SpringBoot整合shiro之盐值加密认证详解

6.1 概述

上节课我们主要讲了自定义 Realm，里面的用户认证所使用的密码都是明文，这种方式是不可取的往往我们在实战开发中用户的密码都是以密文形势进行存储，并且要求加密算法是不可逆的，著名的加密算法有MD5、SHA1等。所以这节课我们主要介绍 Shiro 安全框架学习它的加密方案。这里我们采用md5的方式加密，然后呢。md5加密又分为加盐，和不加盐。

6.2 不加盐认证

6.2.1创建 testMatcher 方法

```
    @Test
    public void testMatcher(){
        CustomRealm customRealm=new CustomRealm();
        //1,构建SecurityManager环境
        DefaultSecurityManager defaultSecurityManager = new DefaultSecurityManager();
        //设置Realm
        defaultSecurityManager.setRealm(customRealm);
        SecurityUtils.setSecurityManager(defaultSecurityManager);
        //获取主体
        Subject subject = SecurityUtils.getSubject();
        //用户名和密码的token
        UsernamePasswordToken token = new UsernamePasswordToken("dev", "123456");
        try {
            //2,主体提交认证请求
            subject.login(token);
            System.out.println("是否权限认证：isAuthenticated=" + subject.isAuthenticated());
            //检查是否有角色
            subject.checkRoles("admin");
            System.out.println("有admin角色");
            //检查是否有权限
            subject.checkPermissions("user:delete","user:edit","user:list","user:add");
            System.out.println("有 user:add、user:list、user:edit、user:delete权限");

        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名或密码错误");
        }catch (LockedAccountException exception){
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }catch ( UnauthorizedException ae ) {
            System.out.println("用户没有权限");
        }
    }
```

6.2.2加入密码认证核心代码

```
        //进行加密
        HashedCredentialsMatcher matcher = new HashedCredentialsMatcher();
        //采用md5加密
        matcher.setHashAlgorithmName("md5");
        //设置加密次数
        matcher.setHashIterations(2);
        customRealm.setCredentialsMatcher(matcher);
```

6.2.3 修改 CustomRealm 新增获取密文的方法

```
/**
     * 获得密文密码
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param currentPassword
     * @return       java.lang.String
     * @throws
     */
    private String getPasswordMatcher(String currentPassword){
        return new Md5Hash(currentPassword, null,2).toString();
    }
```

6.2.4 修改 doGetAuthenticationInfo

```
/**
     * 用户人证
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
```

```
     * @param authenticationToken
     * @return       org.apache.shiro.authc.AuthenticationInfo
     * @throws
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws
AuthenticationException {
        System.out.println("开始执行*******doGetAuthenticationInfo方法啦****");
        //获取登录用户名
        String username = (String) authenticationToken.getPrincipal();
        //通过用户名到数据库获取用户信息
        String password = getPasswordByUsername(username);
        if (password == null) {
            return null;
        }
        String matcherPwd=getPasswordMatcher(password);
        System.out.println(matcherPwd);
        SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,matcherPwd,getName());
        return authenticationInfo;

    }
```

6.2.5 测试

1. 当用户 admin 登录进来的时候 程序输出

```
UsernamePasswordToken token = new UsernamePasswordToken("admin", "123456");
```

- 程序输出

- 假如改成还是用明文检验

```
 //        String matcherPwd=getPasswordMatcher(password);
//        System.out.println(matcherPwd);
//        SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,matcherPwd,getName());
        SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,password,getName());
```

- 程序输出

6.3加盐认证

当两个用户的密码相同时，单纯使用不加盐的MD5加密方式，会发现数据库中存在相同结构的密码，这样也是不安全的。我们希望即便是两个人的原始密码一样，加密后的结果也不一样。如何做到呢？其实就好像炒菜一样，两道一样的鱼香肉丝，加的盐不一样，炒出来的味道就不一样。MD5加密也是一样，需要进行盐值加密。

6.3.1 创建 testSaltMatcher 方法

```
    @Test
    public void testSaltMatcher(){
```

```
        CustomRealm customRealm=new CustomRealm();
        //进行加密
        HashedCredentialsMatcher matcher = new HashedCredentialsMatcher();
        //采用mdf加密
        matcher.setHashAlgorithmName("md5");
        //设置加密次数
        matcher.setHashIterations(1);
        customRealm.setCredentialsMatcher(matcher);
        //1,构建SecurityManager环境
        DefaultSecurityManager defaultSecurityManager = new DefaultSecurityManager();
        //设置Realm
        defaultSecurityManager.setRealm(customRealm);
        SecurityUtils.setSecurityManager(defaultSecurityManager);
        //获取主体
        Subject subject = SecurityUtils.getSubject();
        //用户名和密码的token
        UsernamePasswordToken token = new UsernamePasswordToken("admin", "123456");
        try {
            //2,主体提交认证请求
            subject.login(token);
            System.out.println("是否权限认证：isAuthenticated=" + subject.isAuthenticated());
            //检查是否有角色
            subject.checkRoles("admin");
            System.out.println("有admin角色");
            //检查是否有权限
            subject.checkPermissions("user:delete","user:edit","user:list","user:add");
            System.out.println("有 user:add、user:list、user:edit、user:delete权限");
            //if no exception, that's it, we're done!
        }catch (IncorrectCredentialsException exception){
            System.out.println("用户名或密码错误");
        }catch (LockedAccountException exception){
            System.out.println("账号已被锁定");
        }catch (DisabledAccountException exception){
            System.out.println("账号已被禁用");
        }catch (UnknownAccountException exception){
            System.out.println("用户不存在");
        }catch ( UnauthorizedException ae ) {
            System.out.println("用户没有权限");
        }
    }
```

6.3.2 加盐验证核心代码

```
        //设置盐的值
         authenticationInfo.setCredentialsSalt(ByteSource.Util.bytes(username));
```

6.3.3 修改 CustomRealm 新增获取加盐密文的方法

```
/**
     * 获得密文密码
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param currentPassword
     * @return       java.lang.String
     * @throws
     */
    private String getPasswordMatcher(String currentPassword,String username){
        return new Md5Hash(currentPassword, username).toString();
    }
```

6.3.4修改 doGetAuthenticationInfo

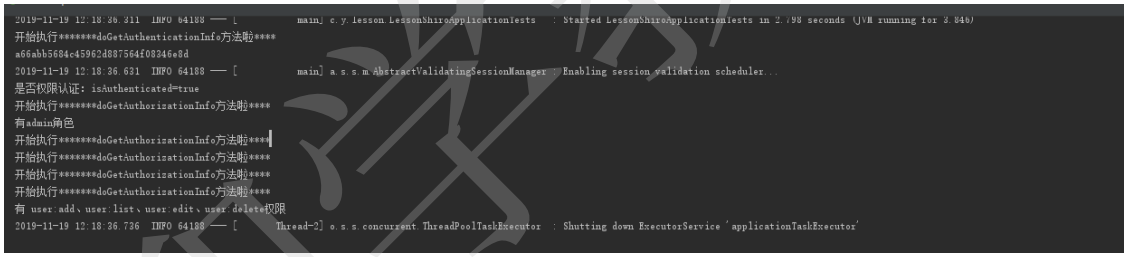
```
 /**
     * 用户人证
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param authenticationToken
     * @return       org.apache.shiro.authc.AuthenticationInfo
```

```
     * @throws
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws
AuthenticationException {
        System.out.println("开始执行*******doGetAuthenticationInfo方法啦****");
        //获取登录用户名
        String username = (String) authenticationToken.getPrincipal();
        //通过用户名到数据库获取用户信息
        String password = getPasswordByUsername(username);
        if (password == null) {
            return null;
        }
//        String matcherPwd=getPasswordMatcher(password);
//        System.out.println(matcherPwd);
//        SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,matcherPwd,getName());
////        SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,password,getName());
//        return authenticationInfo;
        String matcherPwd=getPasswordMatcher(password,username);
        System.out.println(matcherPwd);
        SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,matcherPwd,getName());
        authenticationInfo.setCredentialsSalt(ByteSource.Util.bytes(username));
        return authenticationInfo;

    }
```

6.3.5 测试

1. 当用户 admin 登录进来 程序输出

```
UsernamePasswordToken token = new UsernamePasswordToken("admin", "123456");
```

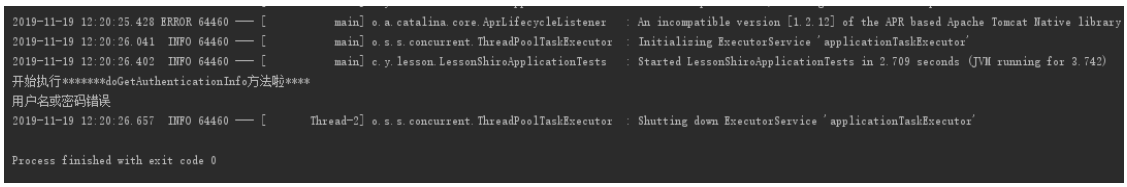


2. 当用户 admin 登录但是密码却输成1234567 程序输出

```
UsernamePasswordToken token = new UsernamePasswordToken("admin", "1234567");
```

3. 当用户 admin 登录进来密码也输入正确了但是 验证的时候改用明文 程序输出

```
        //把下面两个方法注释掉
       // String matcherPwd=getPasswordMatcher(password);
        //System.out.println(matcherPwd);
      SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(username,password,getName());
```



shiro 加密就讲到这里，其实有的同学会有疑问，假如我们公司用 sessionID 作为登录状态的凭证那么我们该怎么办呢，怎么才能让用户通过认证呢？这里我们可以自定义密码匹配类继承 HashedCredentialsMatcher类 重写它的 doCredentialsMatch (密码匹配的方法) 即可。后面我们讲解实战开发的时候会详细的讲解。

7. Spring Boot 2.x+shiro前后端分离实战-骨架搭建

需求：

SpringBoot+shiro+redis 整合起来实现用户认证授权实战

响应码约定：

code=0：服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。

code=4010001：（授权异常）请求要求身份验证。 客户端需要跳转到登录页面重新登录

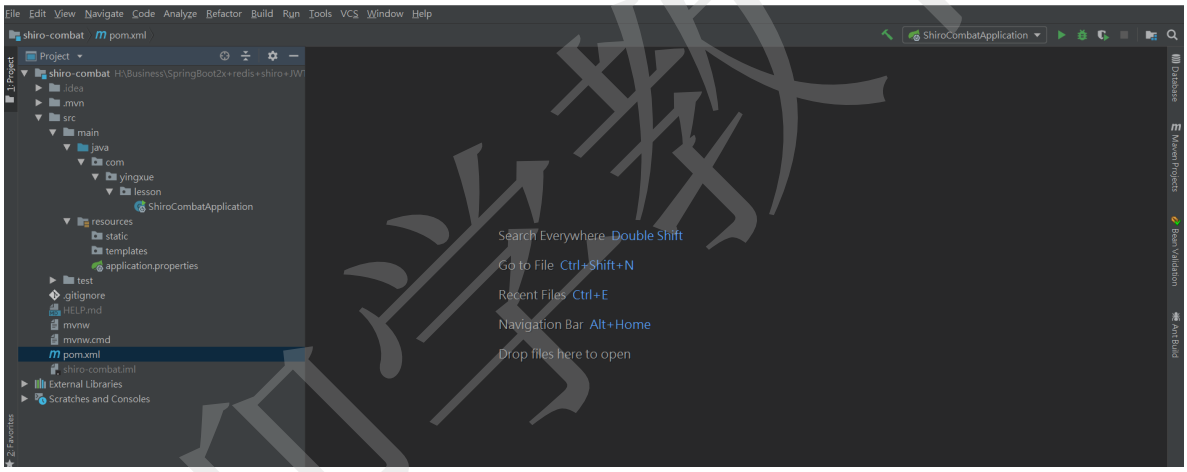
code=4010002：(凭证过期) 客户端请求刷新凭证接口

code=4030001：没有权限禁止访问

code=400xxxx：系统主动抛出的业务异常

code=5000001：系统异常

7.1 创建工程



7.2 加入相关依赖

7.2.1 数据库相关

```
<!--数据库驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!--数据源-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.1.10</version>
        </dependency>
        <!--mybatis -->
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>
```

7.2.2 shiro相关

```
<!--shiro 相关-->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.4.1</version>
        </dependency>
```

7.2.3 redis 相关

```
<!--redis 相关-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>
```

7.2.4 swagger 相关

```
<!--swagger 相关-->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.9.2</version>
        </dependency>
```

7.2.5 其它有用的插件配置

```
       <!--lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <!--fastJson-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.49</version>
        </dependency>
        <!--热部署-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
```

7.2.6 完整的pom

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.10.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.yingxue.lesson</groupId>
    <artifactId>shiro-combat</artifactId>
```

```
    <version>0.0.1-SNAPSHOT</version>
    <name>shiro-combat</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--shiro 相关-->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.4.1</version>
        </dependency>
        <!--redis 相关-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>
        <!--swagger 相关-->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.9.2</version>
        </dependency>
        <!--lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <!--fastJson-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.49</version>
        </dependency>
        <!--热部署-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!--数据库驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!--数据源-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.1.10</version>
        </dependency>
```

```
        <!--mybatis -->
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <fork>true</fork>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

7.3 配置swagger

1. swagger 配置

```
package com.yingxue.lesson.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.ParameterBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.schema.ModelRef;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Parameter;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;
import java.util.List;

/**
 * @ClassName: SwaggerConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@EnableSwagger2
@Configuration
public class SwaggerConfig {

    @Value("${swagger2.enable}")
    private boolean enable;

    @Bean
    public Docket createRestApi() {
        /**
         * 这是为了我们在用 swagger 测试接口的时候添加头部信息
         */
        List<Parameter> pars = new ArrayList<Parameter>();
        ParameterBuilder tokenPar = new ParameterBuilder();
        tokenPar.name("sessionId").description("swagger测试用(模拟sessionId传入)非必填
header").modelRef(new ModelRef("string")).parameterType("header").required(false);
        /**
         * 多个的时候 就直接添加到 pars 就可以了
         */
        pars.add(tokenPar.build());
```

```
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.yingxue.lesson.controller"))
                .paths(PathSelectors.any())
                .build()
                .globalOperationParameters(pars)
                .enable(enable)
                ;
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("迎学教育--shiro 实战")
                .description("迎学教育-spring boot 实战系列")
                .termsOfServiceUrl("")
                .version("1.0")
                .build();
    }

}
```

2. 开关配置

```
#swagger 开关
swagger2.enable=true
```

8 Spring Boot 2.x+shiro前后端分离实战-整合 redis

8.1 自定义 MyStringRedisSerializer 序列化

```
package com.yingxue.lesson.serializer;

import com.alibaba.fastjson.JSON;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.util.Assert;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

/**
 * @ClassName: MyStringRedisSerializer
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class MyStringRedisSerializer implements RedisSerializer<Object> {
    private final Charset charset;

    public MyStringRedisSerializer() {
        this(StandardCharsets.UTF_8);
    }

    public MyStringRedisSerializer(Charset charset) {
        Assert.notNull(charset, "Charset must not be null!");
        this.charset = charset;
    }

    @Override
    public String deserialize(byte[] bytes) {
        return (bytes == null ? null : new String(bytes, charset));
    }

    @Override
    public byte[] serialize(Object object) {
        if (object == null) {
            return new byte[0];
        }
        if(object instanceof String){
```

```
            return object.toString().getBytes(charset);
        }else {
            String string = JSON.toJSONString(object);
            return string.getBytes(charset);
        }

    }



}
```

8.2 注入 RedisTempalet

```
package com.yingxue.lesson.config;

import com.yingxue.lesson.serializer.MyStringRedisSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.StringRedisSerializer;

/**
 * @ClassName: RedisConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
public class RedisConfig {
    @Bean
    public RedisTemplate<String,Object> redisTemplate(RedisConnectionFactory redisConnectionFactory){
        RedisTemplate<String,Object> redisTemplate=new RedisTemplate<>();
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        MyStringRedisSerializer myStringRedisSerializer=new MyStringRedisSerializer();
        StringRedisSerializer stringRedisSerializer=new StringRedisSerializer();
        redisTemplate.setValueSerializer(myStringRedisSerializer);
        redisTemplate.setKeySerializer(stringRedisSerializer);
        redisTemplate.setHashValueSerializer(myStringRedisSerializer);
        redisTemplate.setHashKeySerializer(stringRedisSerializer);
        return redisTemplate;
    }
}
```

8.3 自定义运行时异常

```
package com.yingxue.lesson.exception;

/**
 * @ClassName: BusinessException
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class BusinessException extends RuntimeException{
    /**
     * 异常编号
     */
    private final int messageCode;

    /**
     * 对messageCode 异常信息进行补充说明
     */
    private final String detailMessage;

    public BusinessException(int messageCode, String message) {
```

```
        super(message);
        this.messageCode = messageCode;
        this.detailMessage = message;
    }

    public int getMessageCode() {
        return messageCode;
    }

    public String getDetailMessage() {
        return detailMessage;
    }
}
```

8.4 引入RedisService 工具类

```
package com.yingxue.lesson.service;

import com.yingxue.lesson.exception.BusinessException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.concurrent.TimeUnit;

/**
 * @ClassName: RedisService
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class RedisService {
    @Autowired
    private RedisTemplate<String,Object> redisTemplate;
    /** -------------------key相关操作--------------------- */

    /**
     * 是否存在key
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean hasKey(String key) {
        if (null==key){
            return false;
        }
        return redisTemplate.hasKey(key);
    }

    /**
     * 删除key
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       Boolean  成功返回true 失败返回false
     * @throws
     */
    public Boolean delete(String key) {
        if (null==key){
            return false;
        }
        return redisTemplate.delete(key);
    }

    /**
```

```
     * 批量删除key
     * @Author:      小霍
     * @CreateDate:  2019/8/27 20:27
     * @UpdateUser:
     * @UpdateDate:  2019/8/27 20:27
     * @Version:     0.0.1
     * @param keys
     * @return       Long 返回成功删除key的数量
     * @throws
     */
    public Long delete(Collection<String> keys) {
        return redisTemplate.delete(keys);
    }


    /**
     * 设置过期时间
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param timeout
     * @param unit
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean expire(String key, long timeout, TimeUnit unit) {
        if (null==key||null==unit){
            return false;
        }
        return redisTemplate.expire(key, timeout, unit);
    }

    /**
     * 查找匹配的key
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param pattern
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<String> keys(String pattern) {
        if (null==pattern){
            return null;
        }
        return redisTemplate.keys(pattern);
    }


    /**
     * 移除 key 的过期时间，key 将持久保持
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean persist(String key) {
        if (null==key){
            return false;
        }
        return redisTemplate.persist(key);
    }

    /**
     * 返回 key 的剩余的过期时间
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param unit
```

```
     * @return       java.lang.Long 当 key 不存在时，返回 -2 。当 key 存在但没有设置剩余生存时间时，返回 -1 。否则，以秒为单位，返回 key的剩余生存时间
     * @throws
     */
    public Long getExpire(String key, TimeUnit unit) {
        if(null==key||null==unit){
            throw new BusinessException(4001004,"key or TomeUnit 不能为空");
        }
        return redisTemplate.getExpire(key, unit);
    }

    //*************String相关数据类型**************************
    /**
     * 设置指定 key 的值
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param value
     * @return       void
     * @throws
     */
    public void set(String key, Object value) {

        if(null==key||null==value){
            return;
        }
        redisTemplate.opsForValue().set(key, value);
    }
    /**
     * 设置key 的值 并设置过期时间
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param value
     * @param time
     * @param unit
     * @return       void
     * @throws
     */
    public void set(String key,Object value,long time,TimeUnit unit){

        if(null==key||null==value||null==unit){
            return;
        }
        redisTemplate.opsForValue().set(key,value,time,unit);
    }
    /**
     * 设置key 的值 并设置过期时间
     * key存在 不做操作返回false
     * key不存在设置值返回true
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param value
     * @param time
     * @param unit
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean setifAbsen(String key,Object value,long time,TimeUnit unit){

        if(null==key||null==value||null==unit){
            throw new BusinessException(4001004,"kkey、value、unit都不能为空");
        }
        return redisTemplate.opsForValue().setIfAbsent(key,value,time,unit);
    }
    /**
     * 获取指定Key的Value。如果与该Key关联的Value不是string类型，Redis将抛出异常，
     * 因为GET命令只能用于获取string Value，如果该Key不存在，返回null
     * @Author:      小霍
     * @UpdateUser:
```

```
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Object
     * @throws
     */
    public Object get(String key){

        if(null==key){
            return null;
        }
        return  redisTemplate.opsForValue().get(key);
    }
    /**
     * 很明显先get再set就说先获取key值对应的value然后再set 新的value 值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param value
     * @return       java.lang.Object
     * @throws
     */
    public Object getSet(String key,Object value){

        if(null==key){
            return null;
        }
        return redisTemplate.opsForValue().getAndSet(key,value);
    }
    /**
     * 通过批量的key获取批量的value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param keys
     * @return       java.util.List<java.lang.Object>
     * @throws
     */
    public List<Object> mget(Collection<String> keys){

        if(null==keys){
            return Collections.emptyList();
        }
        return redisTemplate.opsForValue().multiGet(keys);
    }
    /**
     *  将指定Key的Value原子性的增加increment。如果该Key不存在，其初始值为0，在incrby之后其值为increment。
     *  如果value的值不能转换为整型值，如Hi，该操作将执行失败并抛出相应异常。操作成功则返回增加后的value值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param increment
     * @return       long
     * @throws
     */
    public long incrby(String key,long increment){
        if(null==key){
            throw new BusinessException(4001004,"key不能为空");
        }
        return redisTemplate.opsForValue().increment(key,increment);
    }
    /**
     *
     * 将指定Key的Value原子性的减少decrement。如果该Key不存在，其初始值为0，
     * 在decrby之后其值为-decrement。如果Value的值不能转换为整型值，
     * 如Hi，该操作将执行失败并抛出相应的异常。操作成功则返回减少后的value值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param decrement
     * @return       java.lang.Long
     * @throws
```

```
     */
    public Long decrby(String key,long decrement){
        if(null==key){
            throw new BusinessException(4001004,"key不能为空");
        }
        return redisTemplate.opsForValue().decrement(key,decrement);
    }
    /**
     *  如果该Key已经存在，APPEND命令将参数Value的数据追加到已存在Value的末尾。如果该Key不存在，
     *  APPEND命令将会创建一个新的Key/Value。返回追加后Value的字符串长度。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param value
     * @return       java.lang.Integer
     * @throws
     */
    public Integer append(String key,String value){
        if(key==null){
            throw new BusinessException(4001004,"key不能为空");
        }
        return redisTemplate.opsForValue().append(key,value);
    }
//******************hash数据类型**********************
    /**
     * 通过key 和 field 获取指定的 value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param field
     * @return       java.lang.Object
     * @throws
     */
    public Object hget(String key, Object field) {
        if(null==key||null==field){
            return null;
        }
        return redisTemplate.opsForHash().get(key,field);
    }

    /**
     * 为指定的Key设定Field/Value对，如果Key不存在，该命令将创建新Key以用于存储参数中的Field/Value对，
     * 如果参数中的Field在该Key中已经存在，则用新值覆盖其原有值。
     * 返回1表示新的Field被设置了新值，0表示Field已经存在，用新值覆盖原有值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param field
     * @param value
     * @return
     * @throws
     */
    public void hset(String key, Object field, Object value) {
        if(null==key||null==field){
            return;
        }
        redisTemplate.opsForHash().put(key,field,value);
    }

    /**
     * 判断指定Key中的指定Field是否存在，返回true表示存在，false表示参数中的Field或Key不存在。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param field
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean hexists(String key, Object field) {
        if(null==key||null==field){
```

```
            return false;
        }
        return redisTemplate.opsForHash().hasKey(key,field);
    }

    /**
     * 从指定Key的Hashes Value中删除参数中指定的多个字段，如果不存在的字段将被忽略，
     * 返回实际删除的Field数量。如果Key不存在，则将其视为空Hashes，并返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param fields
     * @return       java.lang.Long
     * @throws
     */
    public Long hdel(String key, Object... fields) {
        if(null==key||null==fields||fields.length==0){
            return 0L;
        }
        return redisTemplate.opsForHash().delete(key,fields);
    }


    /**
     * 通过key获取所有的field和value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.util.Map<java.lang.Object,java.lang.Object>
     * @throws
     */
    public Map<Object, Object> hgetall(String key) {
        if(key==null){
            return null;
        }
        return redisTemplate.opsForHash().entries(key);
    }

    /**
     * 逐对依次设置参数中给出的Field/Value对。如果其中某个Field已经存在，则用新值覆盖原有值。
     * 如果Key不存在，则创建新Key，同时设定参数中的Field/Value。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param hash
     * @return
     * @throws
     */
    public void hmset(String key, Map<String, Object> hash) {

        if(null==key||null==hash){
            return;
        }
        redisTemplate.opsForHash().putAll(key,hash);
    }

    /**
     * 获取和参数中指定Fields关联的一组Values，其返回顺序等同于Fields的请求顺序。
     * 如果请求的Field不存在，其值对应的value为null。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param fields
     * @return       java.util.List<java.lang.Object>
     * @throws
     */
    public List<Object> hmget(String key, Collection<Object> fields) {

        if(null==key||null==fields){
            return null;
```

```
        }

        return redisTemplate.opsForHash().multiGet(key,fields);
    }

    /**
     * 对应key的字段自增相应的值
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param field
     * @param increment
     * @return       java.lang.Long
     * @throws
     */
    public Long hIncrBy(String key,Object field,long increment){
        if (null==key||null==field){
            throw new BusinessException(4001004,"key or field 不能为空");
        }
        return redisTemplate.opsForHash().increment(key,field,increment);

    }
    //***************List数据类型***************
    /**
     * 向列表左边添加元素。如果该Key不存在，该命令将在插入之前创建一个与该Key关联的空链表，之后再将数据从链表的头部插入。
     * 如果该键的Value不是链表类型，该命令将将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param strs 可以使一个string 也可以使string数组
     * @return       java.lang.Long 返回操作的value个数
     * @throws
     */
    public Long lpush(String key, Object... strs) {
        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().leftPushAll(key,strs);
    }

    /**
     * 向列表右边添加元素。如果该Key不存在，该命令将在插入之前创建一个与该Key关联的空链表，之后再将数据从链表的尾部插入。
     * 如果该键的Value不是链表类型，该命令将将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param strs 可以使一个string 也可以使string数组
     * @return       java.lang.Long 返回操作的value个数
     * @throws
     */
    public Long rpush(String key, Object... strs) {
        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().rightPushAll(key,strs);
    }
    /**
     * 返回并弹出指定Key关联的链表中的第一个元素，即头部元素。如果该Key不存在，
     * 返回nil。LPOP命令执行两步操作：第一步是将列表左边的元素从列表中移除，第二步是返回被移除的元素值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Object
     * @throws
     */
    public Object lpop(String key) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().leftPop(key);
```

```
    }

    /**
     * 返回并弹出指定Key关联的链表中的最后一个元素，即头部元素。如果该Key不存在，返回nil。
     * RPOP命令执行两步操作：第一步是将列表右边的元素从列表中移除，第二步是返回被移除的元素值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Object
     * @throws
     */
    public Object rpop(String key) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().rightPop(key);
    }

    /**
     *该命令的参数start和end都是0-based。即0表示链表头部(leftmost)的第一个元素。
     * 其中start的值也可以为负值，-1将表示链表中的最后一个元素，即尾部元素，-2表示倒数第二个并以此类推。
     * 该命令在获取元素时，start和end位置上的元素也会被取出。如果start的值大于链表中元素的数量，
     * 空链表将会被返回。如果end的值大于元素的数量，该命令则获取从start(包括start)开始，链表中剩余的所有元素。
     * 注：Redis的列表起始索引为0。显然，LRANGE numbers 0 -1 可以获取列表中的所有元素。返回指定范围内元素的列表。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param start
     * @param end
     * @return       java.util.List<java.lang.Object>
     * @throws
     */
    public List<Object> lrange(String key, long start, long end) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().range(key,start,end);
    }

    /**
     * 让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。
     * 下标 0 表示列表的第一个元素，以 1 表示列表的第二个元素，以此类推。
     * 你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param start
     * @param end
     * @return
     * @throws
     */
    public void ltrim(String key, long start, long end) {
        if(null==key){
            return;
        }
        redisTemplate.opsForList().trim(key,start,end);
    }

    /**
     * 该命令将返回链表中指定位置(index)的元素，index是0-based，表示从头部位置开始第index的元素，
     * 如果index为-1，表示尾部元素。如果与该Key关联的不是链表，该命令将返回相关的错误信息。 如果超出index返回这返回nil。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param index
     * @return       java.lang.Object
     * @throws
     */
    public Object lindex(String key, long index) {
        if(null==key){
```

```
            return null;
        }
        return redisTemplate.opsForList().index(key,index);
    }

    /**
     * 返回指定Key关联的链表中元素的数量，如果该Key不存在，则返回0。如果与该Key关联的Value的类型不是链表，则抛出相关的异常。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Long
     * @throws
     */
    public Long llen(String key) {

        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().size(key);
    }
    //***************Set数据类型*************
    /**
     * 如果在插入的过程用，参数中有的成员在Set中已经存在，该成员将被忽略，而其它成员仍将会被正常插入。
     * 如果执行该命令之前，该Key并不存在，该命令将会创建一个新的Set，此后再将参数中的成员陆续插入。返回实际插入的成员数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param members 可以是一个String 也可以是一个String数组
     * @return       java.lang.Long 添加成功的个数
     * @throws
     */
    public Long sadd(String key, Object... members) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForSet().add(key, members);

    }

    /**
     * 返回Set中成员的数量，如果该Key并不存在，返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Long
     * @throws
     */
    public Long scard(String key) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForSet().size(key);

    }

    /**
     * 判断参数中指定成员是否已经存在于与Key相关联的Set集合中。返回true表示已经存在，false表示不存在，或该Key本身并不存在。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param member
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean sismember(String key, Object member) {
        if (null==key){
            return false;
        }
```

```
        return redisTemplate.opsForSet().isMember(key,member);

    }

    /**
     * 和SPOP一样，随机的返回Set中的一个成员，不同的是该命令并不会删除返回的成员。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.String
     * @throws
     */
    public Object srandmember(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().randomMember(key);

    }
    /**
     * 和SPOP一样，随机的返回Set中的一个成员，不同的是该命令并不会删除返回的成员。
     * 还可以传递count参数来一次随机获得多个元素，根据count的正负不同，具体表现也不同。
     * 当count 为正数时，SRANDMEMBER 会随机从集合里获得count个不重复的元素。
     * 如果count的值大于集合中的元素个数，则SRANDMEMBER 会返回集合中的全部元素。
     * 当count为负数时，SRANDMEMBER 会随机从集合里获得|count|个的元素，如果|count|大与集合中的元素，
     * 就会返回全部元素不够的以重复元素补齐，如果key不存在则返回nil。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param count
     * @return       java.util.List<java.lang.String>
     * @throws
     */
    public List<Object> srandmember(String key,int count) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForSet().randomMembers(key,count);

    }

    /**
     * 通过key随机删除一个set中的value并返回该值
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.String
     * @throws
     */
    public Object spop(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().pop(key);

    }

    /**
     * 通过key获取set中所有的value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> smembers(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().members(key);
```

```
}
/**
 * 从与Key关联的Set中删除参数中指定的成员，不存在的参数成员将被忽略，
 * 如果该Key并不存在，将视为空Set处理。返回从Set中实际移除的成员数量，如果没有则返回0。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @param members
 * @return       java.lang.Long
 * @throws
 */
public Long srem(String key, Object... members) {
    if (null==key){
        return 0L;
    }
    return redisTemplate.opsForSet().remove(key,members);

}

/**
 * 将元素value从一个集合移到另一个集合
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param srckey
 * @param dstkey
 * @param member
 * @return       java.lang.Long
 * @throws
 */
public Boolean smove(String srckey, String dstkey, Object member) {
    if (null==srckey||null==dstkey){
        return false;
    }
    return redisTemplate.opsForSet().move(srckey,member,dstkey);

}


/**
 * 获取两个集合的并集
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @param otherKeys
 * @return       java.util.Set<java.lang.Object> 返回两个集合合并值
 * @throws
 */
public Set<Object> sUnion(String key, String otherKeys) {
    if (null==key||otherKeys==null){
        return null;
    }
    return redisTemplate.opsForSet().union(key, otherKeys);
}
//**********Sorted Set 数据类型*******************
/**
 *添加参数中指定的所有成员及其分数到指定key的Sorted Set中，在该命令中我们可以指定多组score/member作为参数。
 * 如果在添加时参数中的某一成员已经存在，该命令将更新此成员的分数为新值，同时再将该成员基于新值重新排序。
 * 如果键不存在，该命令将为该键创建一个新的Sorted Set Value，并将score/member对插入其中。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @param score
 * @param member
 * @return       java.lang.Long
 * @throws
 */
public Boolean zadd(String key, double score, Object member) {
    if (null==key){
        return false;
```

```
        }
        return redisTemplate.opsForZSet().add(key,member,score);

    }


    /**
     * 该命令将移除参数中指定的成员，其中不存在的成员将被忽略。
     * 如果与该Key关联的Value不是Sorted Set，相应的错误信息将被返回。 如果操作成功则返回实际被删除的成员数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param members 可以使一个string 也可以是一个string数组
     * @return       java.lang.Long
     * @throws
     */
    public Long zrem(String key, Object... members) {
        if(null==key||null==members){
            return 0L;
        }
        return redisTemplate.opsForZSet().remove(key,members);

    }

    /**
     * 返回Sorted Set中的成员数量，如果该Key不存在，返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Long
     * @throws
     */
    public Long zcard(String key) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForZSet().size(key);
    }

    /**
     * 该命令将为指定Key中的指定成员增加指定的分数。如果成员不存在，该命令将添加该成员并假设其初始分数为0，
     * 此后再将其分数加上increment。如果Key不存在，该命令将创建该Key及其关联的Sorted Set，
     * 并包含参数指定的成员，其分数为increment参数。如果与该Key关联的不是Sorted Set类型，
     * 相关的错误信息将被返回。如果不报错则以串形式表示的新分数。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param score
     * @param member
     * @return       java.lang.Double
     * @throws
     */
    public Double zincrby(String key, double score, Object member) {
        if (null==key){
            throw new BusinessException(4001004,"key 不能为空");
        }
        return redisTemplate.opsForZSet().incrementScore(key,member,score);
    }

    /**
     * 该命令用于获取分数(score)在min和max之间的成员数量。
     * （min=<score<=max）如果加上了“(”着表明是开区间例如zcount key (min max 则 表示（min<score=<max）
     * 同理zcount key min (max 则表明(min=<score<max) 返回指定返回数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param min
     * @param max
     * @return       java.lang.Long
     * @throws
```

```
     */
    public Long zcount(String key, double min, double max) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForZSet().count(key, min, max);

    }

    /**
     * Sorted Set中的成员都是按照分数从低到高的顺序存储，该命令将返回参数中指定成员的位置值，
     * 其中0表示第一个成员，它是Sorted Set中分数最低的成员。 如果该成员存在，则返回它的位置索引值。否则返回nil。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param member
     * @return       java.lang.Long
     * @throws
     */
    public Long zrank(String key, Object member) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().rank(key,member);

    }

    /**
     * 如果该成员存在，以字符串的形式返回其分数，否则返回null
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param member
     * @return       java.lang.Double
     * @throws
     */
    public Double zscore(String key, Object member) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().score(key,member);
    }

    /**
     * 该命令返回顺序在参数start和stop指定范围内的成员，这里start和stop参数都是0-based，即0表示第一个成员，-1表示最后一个成员。如果start大于该Sorted
     * Set中的最大索引值，或start > stop，此时一个空集合将被返回。如果stop大于最大索引值，
     * 该命令将返回从start到集合的最后一个成员。如果命令中带有可选参数WITHSCORES选项，
     * 该命令在返回的结果中将包含每个成员的分数值，如value1,score1,value2,score2...。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param min
     * @param max
     * @return       java.util.Set<java.lang.String> 指定区间内的有序集成员的列表。
     * @throws
     */
    public Set<Object> zrange(String key, long min, long max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().range(key, min, max);

    }
    /**
     * 该命令的功能和ZRANGE基本相同，唯一的差别在于该命令是通过反向排序获取指定位置的成员，
     * 即从高到低的顺序。如果成员具有相同的分数，则按降序字典顺序排序。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
```

```
     * @param start
     * @param end
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zReverseRange(String key, long start, long end) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().reverseRange(key, start, end);

    }

    /**
     * 该命令将返回分数在min和max之间的所有成员，即满足表达式min <= score <= max的成员，
     * 其中返回的成员是按照其分数从低到高的顺序返回，如果成员具有相同的分数，
     * 则按成员的字典顺序返回。可选参数LIMIT用于限制返回成员的数量范围。
     * 可选参数offset表示从符合条件的第offset个成员开始返回，同时返回count个成员。
     * 可选参数WITHSCORES的含义参照ZRANGE中该选项的说明。*最后需要说明的是参数中min和max的规则可参照命令ZCOUNT。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param max
     * @param min
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zrangebyscore(String key, double min, double max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().rangeByScore(key, min, max);

    }



    /**
     * 该命令除了排序方式是基于从高到低的分数排序之外，其它功能和参数含义均与ZRANGEBYSCORE相同。
     * 需要注意的是该命令中的min和max参数的顺序和ZRANGEBYSCORE命令是相反的。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param max
     * @param min
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zrevrangeByScore(String key, double min, double max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().reverseRangeByScore(key, min, max);
    }
}
```

8.5 配置redis连接池

```
# Redis 服务器地址
spring.redis.host=localhost
# Redis 服务?连接端?
spring.redis.port=6379
# 连接池最大连接数（使用负值表示没有限制） 默认 8
spring.redis.lettuce.pool.max-active=100
# 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
spring.redis.lettuce.pool.max-wait=PT10S
# 连接池中的最大空闲连接 默认 8
spring.redis.lettuce.pool.max-idle=30
# 连接池中的最小空闲连接 默认 0
spring.redis.lettuce.pool.min-idle=1
```

```
#链接超时时间
spring.redis.timeout=PT10S
```

9 Spring Boot 2.x+shiro前后端分离实战-整合mybatis

9.1 逆向生成代码

1. generatorConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration
1.0//EN"
        "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
    <!--classPathEntry:数据库的JDBC驱动,换成你自己的驱动位置  -->
    <classPathEntry location="F:\mvnrepository\mysql\mysql-connector-java\5.1.28\mysql-connector-
java-5.1.28.jar" />

    <!-- 一个数据库一个context -->
    <!--defaultModelType="flat" 大数据字段，不分表 -->
    <context id="MysqlTables" targetRuntime="MyBatis3" defaultModelType="flat">
        <property name="autoDelimitKeywords" value="true"/>
        <property name="beginningDelimiter" value="`"/>
        <property name="endingDelimiter" value="`"/>
        <property name="javaFileEncoding" value="utf-8"/>
        <plugin type="org.mybatis.generator.plugins.SerializablePlugin"/>

        <plugin type="org.mybatis.generator.plugins.ToStringPlugin"/>

        <!-- 注释 -->
        <commentGenerator>
            <property name="suppressAllComments" value="true"/><!-- 是否取消注释 -->
            <property name="suppressDate" value="true"/> <!-- 是否生成注释代时间戳-->
        </commentGenerator>

        <!-- jdbc连接 -->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
connectionURL="jdbc:mysql://localhost:3306/test_mybatis" userId="root"
                        password="root"/>
        <!-- 类型转换 -->
        <javaTypeResolver>
            <!-- 是否使用bigDecimal， false可自动转化以下类型（Long, Integer, Short, etc.） -->
            <property name="forceBigDecimals" value="false"/>
        </javaTypeResolver>

        <!-- 生成实体类地址 -->
        <javaModelGenerator targetPackage="com.yingxue.lesson.entity"
targetProject="src/main/java">
            <property name="enableSubPackages" value="false"/>
            <property name="trimStrings" value="true"/>
        </javaModelGenerator>
        <!-- 生成mapxml文件 -->
        <sqlMapGenerator targetPackage="mapper" targetProject="src/main/resources">
            <property name="enableSubPackages" value="false"/>
        </sqlMapGenerator>
        <!-- 生成mapxml对应client，也就是接口dao -->
        <javaClientGenerator targetPackage="com.yingxue.lesson.mapper"
targetProject="src/main/java" type="XMLMAPPER">
            <property name="enableSubPackages" value="false"/>
        </javaClientGenerator>


        <table tableName="sys_user" domainObjectName="SysUser"
               enableCountByExample="false"
               enableUpdateByExample="false"
               enableDeleteByExample="false"
               enableSelectByExample="false"
               selectByExampleQueryId="true">
            <columnOverride column="sex" javaType="java.lang.Integer"/>
            <columnOverride column="status" javaType="java.lang.Integer"/>
            <columnOverride column="create_where" javaType="java.lang.Integer"/>
            <columnOverride column="deleted" javaType="java.lang.Integer"/>
        </table>
```

```
    </context>
</generatorConfiguration>
```

2. 配置插件

```
<!--配置mybatis代码生成工具-->
            <!--使用生成工具可以直接使用maven的命令提示符，
            生成语句是mvn mybatis-generator:generate ，
            一旦数据库进行了更改，都需使用这句代码重新生成bean、dao、mapper文件-->
            <plugin>
                <groupId>org.mybatis.generator</groupId>
                <artifactId>mybatis-generator-maven-plugin</artifactId>
                <version>1.3.5</version>
                <configuration>
                    <configurationFile>src/main/resources/generatorConfig.xml</configurationFile>
                    <verbose>true</verbose>
                    <overwrite>true</overwrite>
                </configuration>
                <executions>
                    <execution>
                        <phase>deploy</phase>
                        <id>Generate MyBatis Artifacts</id>
                        <goals>
                            <goal>generate</goal>
                        </goals>
                    </execution>
                </executions>
                <dependencies>
                    <dependency>
                        <groupId>org.mybatis.generator</groupId>
                        <artifactId>mybatis-generator-core</artifactId>
                        <version>1.3.5</version>
                    </dependency>
                </dependencies>
            </plugin>
```

9.2 数据库链接配置

```
#数据库配置
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.druid.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.druid.url=jdbc:mysql://localhost:3306/test_mybatis?
useUnicode=true&characterEncoding=utf-8&useSSL=false
spring.datasource.druid.username=root
spring.datasource.druid.password=root
##################    连接池配置    ################
#连接池建立时创建的初始化连接数
spring.datasource.druid.initial-size=5
#连接池中最大的活跃连接数
spring.datasource.druid.max-active=20
#连接池中最小的活跃连接数
spring.datasource.druid.min-idle=5
# 配置获取连接等待超时的时间
spring.datasource.druid.max-wait=60000
# 打开PSCache，并且指定每个连接上PSCache的大小
spring.datasource.druid.pool-prepared-statements=true
spring.datasource.druid.max-pool-prepared-statement-per-connection-size=20
spring.datasource.druid.validation-query=SELECT 1 FROM DUAL
spring.datasource.druid.validation-query-timeout=30000
#是否在获得连接后检测其可用性
spring.datasource.druid.test-on-borrow=false
#是否在连接放回连接池后检测其可用性
spring.datasource.druid.test-on-return=false
#是否在连接空闲一段时间后检测其可用性
spring.datasource.druid.test-while-idle=true
# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
spring.datasource.druid.time-between-eviction-runs-millis=60000
```

```
# 配置一个连接在池中最小生存的时间，单位是毫秒
spring.datasource.druid.min-evictable-idle-time-millis=300000
```

9.3 mybatis 动态 sql 配置

```
mybatis.mapper-locations=classpath:mapper/*.xml
```

9.4 扫描mapper

```
@MapperScan("com.yingxue.lesson.mapper")
```

10 Spring Boot 2.x+shiro前后端分离实战-自定义 AccessControlFilter token校验

10.1 自定义 token

```
package com.yingxue.lesson.shiro;

import org.apache.shiro.authc.UsernamePasswordToken;

/**
 * @ClassName: CustomPasswordToken
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class CustomPasswordToken extends UsernamePasswordToken {
    private String token;

    public CustomPasswordToken(String token) {
        this.token = token;
    }

    @Override
    public Object getPrincipal() {
        return token;
    }
}
```

10.2 自定义 AccessControlFilter 用户凭证校验

```
package com.yingxue.lesson.shiro;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONPObject;
import com.yingxue.lesson.constants.Constant;
import com.yingxue.lesson.exception.BusinessException;
import lombok.extern.slf4j.Slf4j;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.web.filter.AccessControlFilter;
import org.springframework.util.StringUtils;

import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
import java.io.OutputStream;
import java.util.HashMap;
import java.util.Map;

/**
 * @ClassName: CustomAccessControlFilter
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
```

```
 * @Version: 0.0.1
 */
@Slf4j
public class CustomAccessControlFilter extends AccessControlFilter {
    /**
     * 是否允许访问
     * true：允许，交下一个Filter处理
     * false：回往下执行onAccessDenied
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param servletRequest
     * * @param servletResponse
     * * @param o
     * @return       boolean
     * @throws
     */
    @Override
    protected boolean isAccessAllowed(ServletRequest servletRequest, ServletResponse servletResponse,
Object o) {
        return false;
    }
    /**
     * 表示访问拒绝时是否自己处理，
     * 如果返回true表示自己不处理且继续拦截器链执行，
     * 返回false表示自己已经处理了（比如重定向到另一个页面）。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param servletRequest
     * @param servletResponse
     * @return       boolean
     * @throws
     */
    @Override
    protected boolean onAccessDenied(ServletRequest servletRequest, ServletResponse servletResponse)
throws IOException, ServletException {
        HttpServletRequest request= (HttpServletRequest) servletRequest;
        try {
            log.info(request.getMethod());
            log.info(request.getRequestURL().toString());
            //获取用户凭证
            String accessToken=request.getHeader(Constant.SESSION_ID);
            if(StringUtils.isEmpty(accessToken)){
                throw new BusinessException(4001002,"用户凭证为空请重新登录");
            }
            CustomPasswordToken customPasswordToken=new CustomPasswordToken(accessToken);
            // 委托给Realm进行登录
            getSubject(servletRequest, servletResponse).login(customPasswordToken);
        }catch (BusinessException e) {
            customResponse(e.getMessageCode(),e.getDefaultMessage(),servletResponse);
            return false;
        } catch (AuthenticationException e) {
            if(e.getCause() instanceof BusinessException){
                BusinessException exception= (BusinessException) e.getCause();

 customResponse(exception.getMessageCode(),exception.getDefaultMessage(),servletResponse);
                return false;
            }else {
                customResponse(4000001,"用户认证失败",servletResponse);
                return false;
            }
        }catch (AuthorizationException e){
            if(e.getCause() instanceof BusinessException){
                BusinessException exception= (BusinessException) e.getCause();

 customResponse(exception.getMessageCode(),exception.getDefaultMessage(),servletResponse);
                return false;
            }else {
                customResponse(4030001,"没有访问的权限",servletResponse);
                return false;
            }
        }
        catch (Exception e){
```

```
            if(e.getCause() instanceof BusinessException){
                BusinessException exception= (BusinessException) e.getCause();

 customResponse(exception.getMessageCode(),exception.getDefaultMessage(),servletResponse);
                return false;
            }else {
                customResponse(5000001,"系统异常",servletResponse);
                return false;
            }
        }
        return true;
    }

    /**
     * 自定义错误响应
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param code
     * @param msg
     * @param response
     * @return       void
     * @throws
     */
    private void customRsponse(int code, String msg, ServletResponse response){
        // 自定义异常的类，用户返回给客户端相应的JSON格式的信息
        try {
            Map<String,Object> result=new HashMap<>();
            result.put("code",code);
            result.put("msg",msg);
            response.setContentType("application/json; charset=utf-8");
            response.setCharacterEncoding("UTF-8");

            String userJson = JSON.toJSONString(result);
            OutputStream out = response.getOutputStream();
            out.write(userJson.getBytes("UTF-8"));
            out.flush();
        } catch (IOException e) {
            log.error("eror={}",e);
        }
    }

}
```

10.3 自定义 CredentialsMatcher

```
package com.yingxue.lesson.shiro;

import com.yingxue.lesson.exception.BusinessException;
import com.yingxue.lesson.service.RedisService;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.credential.SimpleCredentialsMatcher;
import org.springframework.beans.factory.annotation.Autowired;


/**
 * @ClassName: CustomHashedCredentialsMatcher
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class CustomHashedCredentialsMatcher extends HashedCredentialsMatcher {

    @Autowired
    private RedisService redisService;
    @Override
    public boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {
        CustomPasswordToken customPasswordToken= (CustomPasswordToken) token;
        String accessToken = (String) customPasswordToken.getPrincipal();
        if(!redisService.hasKey(accessToken)){
```

```
            throw new BusinessException(4001002,"授权信息信息无效请重新登录");
        }
        return true;
    }
}
```

11 Spring Boot 2.x+shiro前后端分离实战-自定义 Realm

```
package com.yingxue.lesson.shiro;


import com.yingxue.lesson.service.RedisService;
import lombok.extern.slf4j.Slf4j;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
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.springframework.beans.factory.annotation.Autowired;

import java.util.*;

/**
 * @ClassName: CustomRealm
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Slf4j
public class CustomRealm extends AuthorizingRealm {
    @Autowired
    private RedisService redisService;
    @Override
    public boolean supports(AuthenticationToken token) {
        return token instanceof CustomPasswordToken;
    }
    /**
     * 主要业务:
     * 系统业务出现要验证用户的角色权限的时候，就会调用这个方法
     * 来获取该用户所拥有的角色/权限
     * 这个用户授权的方法我们可以缓存起来不用每次都调用这个方法。
     * 后续的课程我们会结合 redis 实现它
     */

    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        SimpleAuthorizationInfo authorizationInfo=new SimpleAuthorizationInfo();
        String accessToken= (String) SecurityUtils.getSubject().getPrincipal();
        String userId= (String) redisService.get(accessToken);
        authorizationInfo.addRoles(getRolesByUserId(userId));
        authorizationInfo.setStringPermissions(getPermissionByUserId(userId));
        return authorizationInfo;
    }

    /**
     * 主要业务:
     * 当业务代码调用 subject.login(customPasswordToken); 方法后
     * 就会自动调用这个方法 验证用户名/密码
     * 这里我们改造成 验证 token 是否有效 已经自定义了 shiro 验证
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws
AuthenticationException {
        CustomPasswordToken token= (CustomPasswordToken) authenticationToken;
        SimpleAuthenticationInfo simpleAuthenticationInfo=new
SimpleAuthenticationInfo(token.getPrincipal(),token.getPrincipal().getName());
```

```
        return simpleAuthenticationInfo;
    }

    /**
     * 获取用户的角色
     * 这里先用伪代码代替
     * 后面我们讲到权限管理系统后 再从 DB 读取
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param userId
     * @return        java.util.List<String>
     * @throws
     */
    private List<String> getRolesByUserId(String userId) {
        List<String> roles = new ArrayList<>();
        if(userId.equals("9a26f5f1-cbd2-473d-82db-1d6dcf4598f8")){
            roles.add("admin");
        }else {
            roles.add("test");
        }

        return roles;
    }

    /**
     * 获取用户的权限
     * 这里先用伪代码代替
     * 后面我们讲到权限管理系统后 再从 DB 读取
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param userId
     * @return        java.util.List<String>
     * @throws
     */
     private List<String> getPermissionByUserId(String userId) {
        List<String> permissions = new ArrayList<>();
        /**
         * 只有是 admin 用户才拥有所有权限
         */
        if(userId.equals("9a26f5f1-cbd2-473d-82db-1d6dcf4598f8")){
            permissions.add("*");
        }else {
            permissions.add("sys:user:edit");
            permissions.add("sys:user:list");
        }
        return permissions;
    }
}
```

12 Spring Boot 2.x+shiro前后端分离实战-shiro核心配置

```
package com.yingxue.lesson.config;

import com.yingxue.lesson.shiro.CustomAccessControlFilter;
import com.yingxue.lesson.shiro.CustomHashedCredentialsMatcher;
import com.yingxue.lesson.shiro.CustomRealm;
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.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.servlet.Filter;
import java.util.LinkedHashMap;
import java.util.Map;
```

```java
/**
 * @ClassName: ShiroConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
public class ShiroConfig {
    /**
     * 自定义密码 校验
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       com.yingxue.lesson.shiro.CustomHashedCredentialsMatcher
     * @throws
     */
    @Bean
    public CustomHashedCredentialsMatcher customHashedCredentialsMatcher(){
        return new CustomHashedCredentialsMatcher();
    }
    /**
     * 自定义域
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       com.yingxue.lesson.shiro.CustomRealm
     * @throws
     */
    @Bean
    public CustomRealm customRealm(){
        CustomRealm customRealm=new CustomRealm();
        customRealm.setCredentialsMatcher(customHashedCredentialsMatcher());
        return customRealm;
    }


    /**
     * 安全管理
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       org.apache.shiro.mgt.SecurityManager
     * @throws
     */
    @Bean
    public SecurityManager securityManager(){
        //构建 SecurityManager环境
        DefaultWebSecurityManager securityManager=new DefaultWebSecurityManager();
        //自定义 Realm
        securityManager.setRealm(customRealm());
        return securityManager;
    }

    /**
     * shiro过滤器，配置拦截哪些请求
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param securityManager
     * @return       org.apache.shiro.spring.web.ShiroFilterFactoryBean
     * @throws
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager){
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        shiroFilterFactoryBean.setSecurityManager(securityManager);
        //自定义拦截器限制并发人数,参考博客:
        LinkedHashMap<String, Filter> filtersMap = new LinkedHashMap<>();
        //用来校验token
```

```
        filtersMap.put("token", new CustomAccessControlFilter());
        shiroFilterFactoryBean.setFilters(filtersMap);
        Map<String, String> filterChainDefinitionMap = new LinkedHashMap<>();
        filterChainDefinitionMap.put("/api/user/login", "anon");
        //放开swagger-ui地址
        filterChainDefinitionMap.put("/swagger/**", "anon");
        filterChainDefinitionMap.put("/v2/api-docs", "anon");
        filterChainDefinitionMap.put("/swagger-ui.html", "anon");
        filterChainDefinitionMap.put("/swagger-resources/**", "anon");
        filterChainDefinitionMap.put("/webjars/**", "anon");
        filterChainDefinitionMap.put("/favicon.ico", "anon");
        filterChainDefinitionMap.put("/captcha.jpg", "anon");
        filterChainDefinitionMap.put("/csrf","anon");
        filterChainDefinitionMap.put("/**","token,authc");
        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
        return shiroFilterFactoryBean;
    }

    /**
     * 开启shiro aop注解支持.
     * 使用代理方式;所以需要开启代码支持;
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param securityManager
     * @return       org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor
     * @throws
     */
    @Bean
    public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager
securityManager) {
        AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new
AuthorizationAttributeSourceAdvisor();
        authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
        return authorizationAttributeSourceAdvisor;
    }
    @Bean
    @ConditionalOnMissingBean
    public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {
        DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator = new
DefaultAdvisorAutoProxyCreator();
        defaultAdvisorAutoProxyCreator.setProxyTargetClass(true);
        return defaultAdvisorAutoProxyCreator;
    }


}
```

13 Spring Boot 2.x+shiro前后端分离实战-实现用户登录认证访问授权

13.1 登录接口业务

```
@Override
    public LoginRespVO login(LoginReqVO vo) {
        SysUser userByName = sysUserMapper.getUserByName(vo.getUsername());
        if(userByName==null){
            throw new BusinessException(4001004,"用户名密码不匹配");
        }
        if(userByName.getStatus()==2){
            throw new BusinessException(4001004,"该账户已经被锁定，请联系系统管理员");
        }
//
if(!getPasswordMatcher(vo.getPassword(),userByName.getSalt()).equals(userByName.getPassword())){
//            throw new BusinessException(4001004,"用户名密码不匹配");
//        }
        if(!PasswordUtils.matches(userByName.getSalt(),vo.getPassword(),userByName.getPassword())){
            throw new BusinessException(4001004,"用户名密码不匹配");
        }
        LoginRespVO loginRespVO=new LoginRespVO();
        loginRespVO.setId(userByName.getId());
        String token= UUID.randomUUID().toString();
        loginRespVO.setToken(token);
```

```
        redisService.set(token,userByName.getId(),60, TimeUnit.MINUTES);
        return loginRespVO;
    }
     /**
     * 获得密文密码
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param currentPassword
     * @return       java.lang.String
     * @throws
     */
    private String getPasswordMatcher(String currentPassword,String salt){
        return new Md5Hash(currentPassword, salt).toString();
    }
```

13.2 用户详情业务

```
   @Override
    public SysUser detail(String id) {
        return sysUserMapper.selectByPrimaryKey(id);
    }
```

13.3 RESTful 控制层接口

```
package com.yingxue.lesson.controller;

import com.yingxue.lesson.entity.SysUser;
import com.yingxue.lesson.service.UserService;
import com.yingxue.lesson.vo.req.LoginReqVO;
import com.yingxue.lesson.vo.resp.LoginRespVO;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiModelProperty;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import org.apache.shiro.authz.annotation.RequiresPermissions;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

/**
 * @ClassName: UserController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/api")
@Api(tags = "用户模块")
public class UserController {
    @Autowired
    private UserService userService;

    @PostMapping("/user/login")
    @ApiOperation(value ="用户登录接口")
    public Map<String, Object> login(@RequestBody LoginReqVO vo){
        Map<String,Object> result=new HashMap<>();
        result.put("code",0);
        result.put("data",userService.login(vo));
        return result;
    }

    @GetMapping("/user/{id}")
    @ApiModelProperty(value = "查询用户详情接口")
    @RequiresPermissions("sys:user:detail")
    public Map<String, Object> detail(@PathVariable("id") @ApiParam(value = "用户Id") String id){
        Map<String,Object> result=new HashMap<>();
        result.put("code",0);
        result.put("data",userService.detail(id));
        return result;
    }
```

```
}
```

13.4 配置登录接口白名单

修改com.yingxue.lesson.config.ShiroConfig#shiroFilterFactoryBean加入shiro 忽略拦截

```
        filterChainDefinitionMap.put("/api/user/login", "anon");
```