第6章 Shiro高级及SaaS-HRM的认证授权

-

1 Shiro在SpringBoot工程的应用

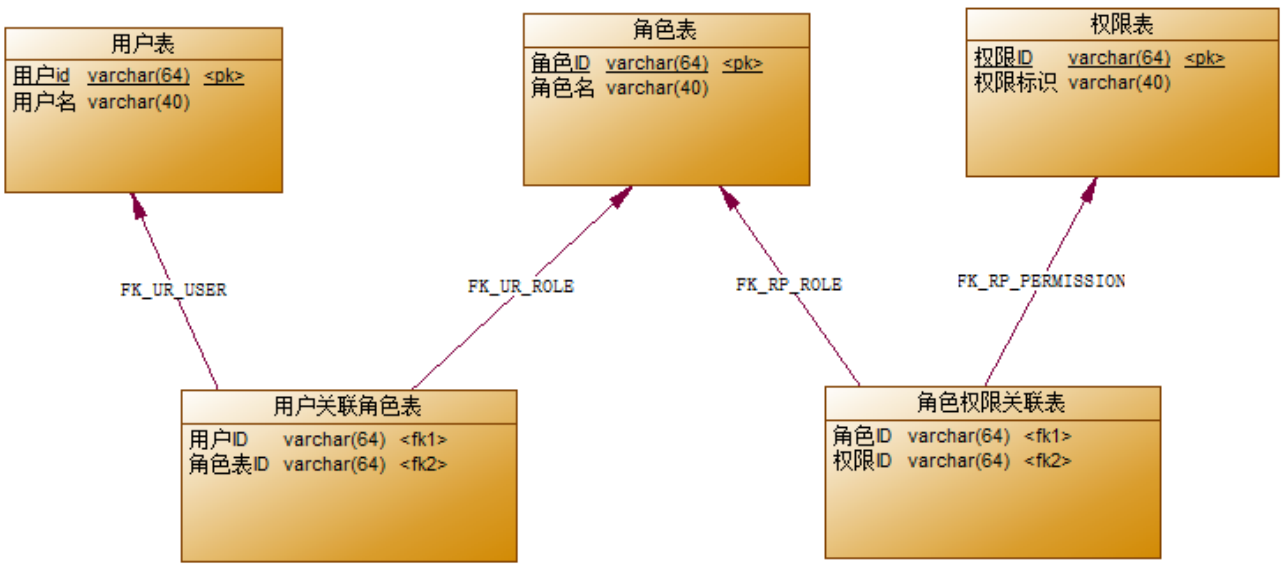
Apache Shiro是一个功能强大、灵活的，开源的安全框架。它可以干净利落地处理身份验证、授权、企业会话管理和加密。越来越多的企业使用Shiro作为项目的安全框架，保证项目的平稳运行。

在之前的讲解中只是单独的使用shiro，方便学员对shiro有一个直观且清晰的认知，我们今天就来看一下shiro在springBoot工程中如何使用以及其他特性

1.1 案例说明

使用springBoot构建应用程序，整合shiro框架完成用户认证与授权。

1.1.1 数据库表



1.1.2 基本工程结构

导入资料中准备的基本工程代码，此工程中实现了基本用户角色权限的操作。我们只需要在此工程中添加Shiro相关的操作代码即可

1.2 整合Shiro

1.2.1 spring和shiro的整合依赖

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.3.2</version>
</dependency>
 <dependency>
     <groupId>org.apache.shiro</groupId>
     <artifactId>shiro-core</artifactId>
     <version>1.3.2</version>
</dependency>
```

1.2.2 修改登录方法

认证：身份认证/登录，验证用户是不是拥有相应的身份。基于shiro的认证，shiro需要采集到用户登录数据使用subject的login方法进入realm完成认证工作。

```
    @RequestMapping(value="/login")
    public String login(String username,String password) {
        try{
            Subject subject = SecurityUtils.getSubject();
            UsernamePasswordToken uptoken = new
UsernamePasswordToken(username,password);
            subject.login(uptoken);
            return "登录成功";
        }catch (Exception e) {
            return "用户名或密码错误";
        }
    }
```

1.2.3 自定义realm

Realm域：Shiro从Realm获取安全数据（如用户、角色、权限），就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作；可以把Realm看成DataSource，即安全数据源

```
public class CustomRealm extends AuthorizingRealm {

    @Override
    public void setName(String name) {
        super.setName("customRealm");
    }

    @Autowired
    private UserService userService;

    /**
     * 构造授权方法
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
```

```
        //1.获取认证的用户数据
        User user = (User)principalCollection.getPrimaryPrincipal();
        //2.构造认证数据
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
        Set<Role> roles = user.getRoles();
        for (Role role : roles) {
            //添加角色信息
            info.addRole(role.getName());
            for (Permission permission:role.getPermissions()) {
                //添加权限信息
                info.addStringPermission(permission.getCode());
            }
        }
        return info;
    }

    /**
     * 认证方法
     */
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        //1.获取登录的upToken
        UsernamePasswordToken upToken = (UsernamePasswordToken)authenticationToken;
        //2.获取输入的用户名密码
        String username = upToken.getUsername();
        String password = new String(upToken.getPassword());
        //3.数据库查询用户
        User user = userService.findByName(username);
        //4.用户存在并且密码匹配存储用户数据
        if(user != null && user.getPassword().equals(password)) {
            return new
SimpleAuthenticationInfo(user,user.getPassword(),this.getName());
        }else {
            //返回null会抛出异常，表明用户不存在或密码不匹配
            return null;
        }
    }
}
```

1.3 Shiro的配置

SecurityManager 是 Shiro 架构的心脏，用于协调内部的多个组件完成全部认证授权的过程。例如通过调用realm完成认证与登录。使用基于springboot的配置方式完成SecurityManager，Realm的装配

```
@Configuration
public class ShiroConfiguration {

    //配置自定义的Realm
    @Bean
    public CustomRealm getRealm() {
        return new CustomRealm();
    }
```

```
    //配置安全管理器
    @Bean
    public SecurityManager securityManager(CustomRealm realm) {
        //使用默认的安全管理器
        DefaultWebSecurityManager securityManager = new
DefaultWebSecurityManager(realm);
        //将自定义的realm交给安全管理器统一调度管理
        securityManager.setRealm(realm);
        return securityManager;
    }

    //Filter工厂，设置对应的过滤条件和跳转条件
    @Bean
    public ShiroFilterFactoryBean shirFilter(SecurityManager securityManager) {
        //1.创建shiro过滤器工厂
        ShiroFilterFactoryBean filterFactory = new ShiroFilterFactoryBean();
        //2.设置安全管理器
        filterFactory.setSecurityManager(securityManager);
        //3.通用配置（配置登录页面，登录成功页面，验证未成功页面）
        filterFactory.setLoginUrl("/autherror?code=1"); //设置登录页面
        filterFactory.setUnauthorizedUrl("/autherror?code=2"); //授权失败跳转页面
        //4.配置过滤器集合
        /**
         * key   ：访问连接
         *       支持通配符的形式
         * value：过滤器类型
         *       shiro常用过滤器
         *           anno     ：匿名访问（表明此链接所有人可以访问）
         *           authc    ：认证后访问（表明此链接需登录认证成功之后可以访问）
         */
        Map<String,String> filterMap = new LinkedHashMap<String,String>();
        // 配置不会被拦截的链接 顺序判断
        filterMap.put("/user/home", "anon");
        filterMap.put("/user/**", "authc");

        //5.设置过滤器
        filterFactory.setFilterChainDefinitionMap(filterMap);
        return filterFactory;
    }

    //配置shiro注解支持
    @Bean
    public AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor(SecurityManager securityManager) {
        AuthorizationAttributeSourceAdvisor advisor = new
AuthorizationAttributeSourceAdvisor();
        advisor.setSecurityManager(securityManager);
        return advisor;
    }
}
```

1.4 shiro中的过滤器

Filter	解释
anon	无参，开放权限，可以理解为匿名用户或游客
authc	无参，需要认证
logout	无参，注销，执行后会直接跳转到`shiroFilterFactoryBean.setLoginUrl();`设置的url
authcBasic	无参，表示 httpBasic 认证
user	无参，表示必须存在用户，当登入操作时不做检查
ssl	无参，表示安全的URL请求，协议为 https
perms[user]	参数可写多个，表示需要某个或某些权限才能通过，多个参数时写 perms[“user, admin”]，当有多个参数时必须每个参数都通过才算通过
roles[admin]	参数可写多个，表示是某个或某些角色才能通过，多个参数时写 roles[“admin，user”]，当有多个参数时必须每个参数都通过才算通过
rest[user]	根据请求的方法，相当于 perms[user:method]，其中 method 为 post，get，delete 等
port[8081]	当请求的URL端口不是8081时，跳转到当前访问主机HOST的8081端口

> 注意：anon, authc, authcBasic, user 是第一组认证过滤器，perms, port, rest, roles, ssl 是第二组授权过滤器，要通过授权过滤器，就先要完成登陆认证操作（即先要完成认证才能前去寻找授权) 才能走第二组授权器（例如访问需要 roles 权限的 url，如果还没有登陆的话，会直接跳转到 shiroFilterFactoryBean.setLoginUrl(); 设置的 url ）

1.5 授权

授权：即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情

shiro支持基于过滤器的授权方式也支持注解的授权方式

1.5.1 基于配置的授权

在shiro中可以使用过滤器的方式配置目标地址的请求权限

```
        //配置请求连接过滤器配置
        //匿名访问（所有人员可以使用）
        filterMap.put("/user/home", "anon");
        //具有指定权限访问
        filterMap.put("/user/find", "perms[user-find]");
        //认证之后访问（登录之后可以访问）
        filterMap.put("/user/**", "authc");
        //具有指定角色可以访问
        filterMap.put("/user/**", "roles[系统管理员]");
```

> 基于配置的方式进行授权，一旦操作用户不具备操作权限，目标地址不会被执行。会跳转到指定的url连接地址。所以需要在连接地址中更加友好的处理未授权的信息提示

1.5.2 基于注解的授权

（1）**RequiresPermissions**

配置到方法上，表明执行此方法必须具有指定的权限

```
    //查询
    @RequiresPermissions(value = "user-find")
    public String find() {
        return "查询用户成功";
    }
```

（2）**RequiresRoles**

配置到方法上，表明执行此方法必须具有指定的角色

```
    //查询
    @RequiresRoles(value = "系统管理员")
    public String find() {
        return "查询用户成功";
    }
```

> 基于注解的配置方式进行授权，一旦操作用户不具备操作权限，目标方法不会被执行，而且会抛出AuthorizationException 异常。所以需要做好统一异常处理完成未授权处理

2 Shiro中的会话管理

在shiro里所有的用户的会话信息都会由Shiro来进行控制，shiro提供的会话可以用于JavaSE/JavaEE环境，不依赖于任何底层容器，可以独立使用，是完整的会话模块。通过Shiro的会话管理器（**SessionManager**）进行统一的会话管理

2.1 什么是shiro的会话管理

SessionManager（会话管理器）：管理所有Subject的session包括创建、维护、删除、失效、验证等工作。SessionManager是顶层组件，由SecurityManager管理

shiro提供了三个默认实现：

1. DefaultSessionManager：用于JavaSE环境
2. ServletContainerSessionManager：用于Web环境，直接使用servlet容器的会话。
3. DefaultWebSessionManager：用于web环境，自己维护会话（自己维护着会话，直接废弃了Servlet容器的会话管理）。

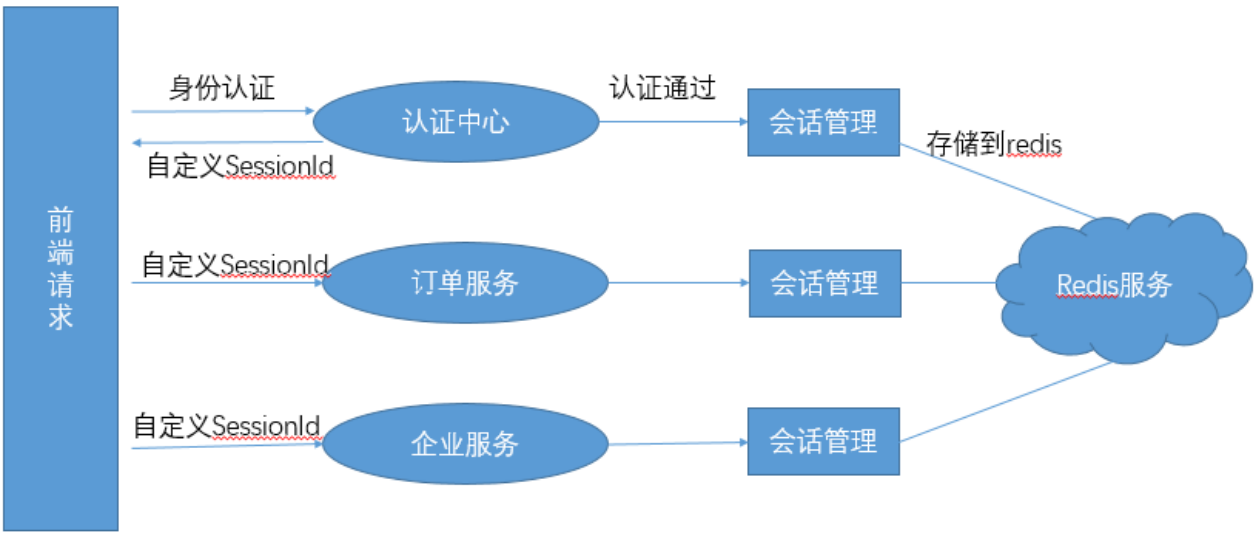
在web程序中，通过shiro的Subject.login()方法登录成功后，用户的认证信息实际上是保存在HttpSession中的通过如下代码验证。

```
    //登录成功后，打印所有session内容
    @RequestMapping(value="/show")
```

```
    public String show(HttpSession session) {
        // 获取session中所有的键值
        Enumeration<?> enumeration = session.getAttributeNames();
        // 遍历enumeration中的
        while (enumeration.hasMoreElements()) {
            // 获取session键值
            String name = enumeration.nextElement().toString();
            // 根据键值取session中的值
            Object value = session.getAttribute(name);
            // 打印结果
            System.out.println("<B>" + name + "</B>=" + value + "<br>/n");
        }
        return "查看session成功";
    }
```

2.2 应用场景分析

在分布式系统或者微服务架构下，都是通过统一的认证中心进行用户认证。如果使用默认会话管理，用户信息只会保存到一台服务器上。那么其他服务就需要进行会话的同步。

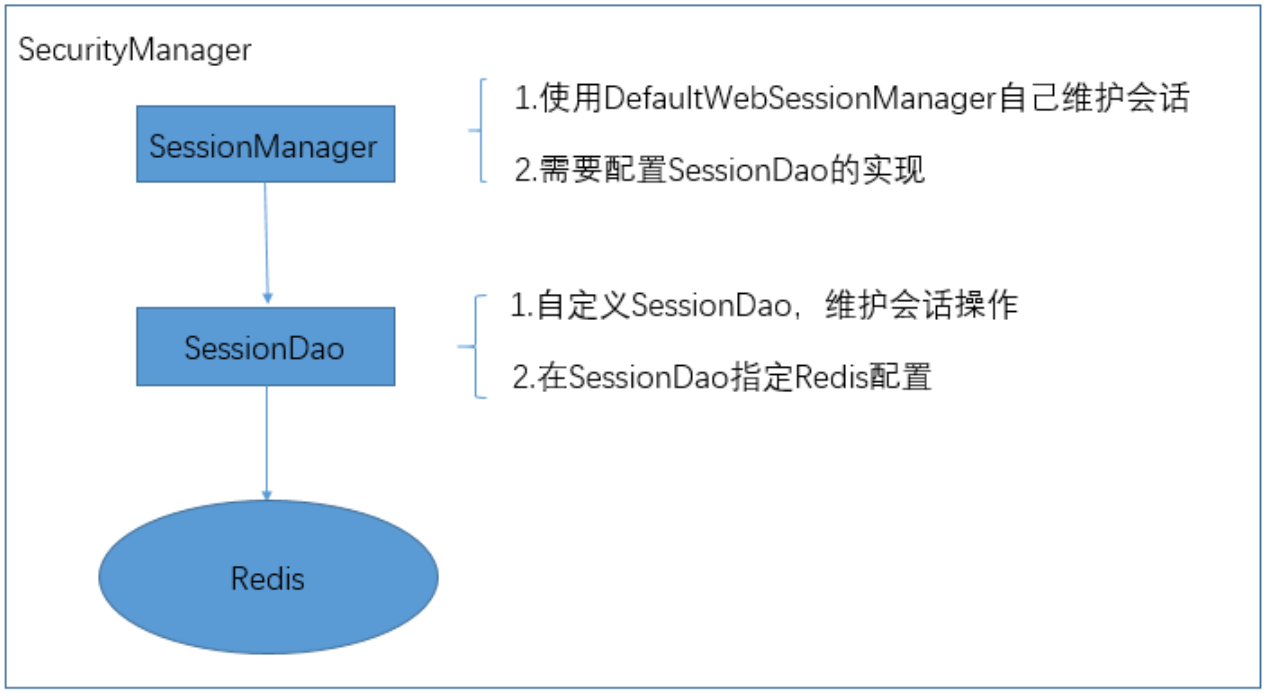


会话管理器可以指定sessionId的生成以及获取方式。

通过sessionDao完成模拟session存入，取出等操作

2.3 Shiro结合redis的统一会话管理

2.3.1 步骤分析



2.3.2 构建环境

（1）使用开源组件Shiro-Redis可以方便的构建shiro与redis的整合工程。

```
<dependency>
    <groupId>org.crazycake</groupId>
    <artifactId>shiro-redis</artifactId>
    <version>3.0.0</version>
</dependency>
```

（2）在springboot配置文件中添加redis配置

```
  redis:
    host: 127.0.0.1
    port: 6379
```

2.3.3 自定义shiro会话管理器

```
/**
 * 自定义的sessionManager
 */
public class CustomSessionManager extends DefaultWebSessionManager {


    /**
     * 头信息中具有sessionid
     *      请求头：Authorization： sessionid
     *
     * 指定sessionId的获取方式
     */
```

```
    protected Serializable getSessionId(ServletRequest request, ServletResponse
response) {

        //获取请求头Authorization中的数据
        String id = WebUtils.toHttp(request).getHeader("Authorization");
        if(StringUtils.isEmpty(id)) {
            //如果没有携带，生成新的sessionId
            return super.getSessionId(request,response);
        }else{
            //返回sessionId；
            request.setAttribute(ShiroHttpServletRequest.REFERENCED_SESSION_ID_SOURCE,
"header");
            request.setAttribute(ShiroHttpServletRequest.REFERENCED_SESSION_ID, id);

 request.setAttribute(ShiroHttpServletRequest.REFERENCED_SESSION_ID_IS_VALID,
Boolean.TRUE);
            return id;
        }
    }
}
```

2.3.4 配置Shiro基于redis的会话管理

在Shiro配置类 `cn.itcast.shiro.ShiroConfiguration` 配置

1. 配置shiro的RedisManager，通过shiro-redis包提供的RedisManager统一对redis操作

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

@Value("${spring.redis.port}")
private int port;
//配置shiro redisManager
public RedisManager redisManager() {
    RedisManager redisManager = new RedisManager();
    redisManager.setHost(host);
    redisManager.setPort(port);
    return redisManager;
}
```

2. Shiro内部有自己的本地缓存机制，为了更加统一方便管理，全部替换redis实现

```
//配置Shiro的缓存管理器
//使用redis实现
public RedisCacheManager cacheManager() {
    RedisCacheManager redisCacheManager = new RedisCacheManager();
    redisCacheManager.setRedisManager(redisManager());
    return redisCacheManager;
}
```

3. 配置SessionDao，使用shiro-redis实现的基于redis的sessionDao

```
/**
 * RedisSessionDAO shiro sessionDao层的实现 通过redis
 * 使用的是shiro-redis开源插件
 */
public RedisSessionDAO redisSessionDAO() {
    RedisSessionDAO redisSessionDAO = new RedisSessionDAO();
    redisSessionDAO.setRedisManager(redisManager());
    return redisSessionDAO;
}
```

4. 配置会话管理器，指定sessionDao的依赖关系

```
    /**
     * 3.会话管理器
     */
    public DefaultWebSessionManager sessionManager() {
        CustomSessionManager sessionManager = new CustomSessionManager();
        sessionManager.setSessionDAO(redisSessionDAO());
        return sessionManager;
    }
```

5. 统一交给SecurityManager管理

```
    //配置安全管理器
    @Bean
    public SecurityManager securityManager(CustomRealm realm) {
        //使用默认的安全管理器
        DefaultWebSecurityManager securityManager = new
DefaultWebSecurityManager(realm);
        // 自定义session管理 使用redis
        securityManager.setSessionManager(sessionManager());
        // 自定义缓存实现 使用redis
        securityManager.setCacheManager(cacheManager());
        //将自定义的realm交给安全管理器统一调度管理
        securityManager.setRealm(realm);
        return securityManager;
    }
```

3 SaaS-HRM中的认证授权

3.1 需求分析

实现基于Shiro的SaaS平台的统一权限管理。我们的SaaS-HRM系统是基于微服务构建，所以在使用Shiro鉴权的时候，就需要将认证信息保存到统一的redis服务器中完成。这样，每个微服务都可以通过指定cookie中的sessionid获取公共的认证信息。

3.2 搭建环境

3.2.1 导入依赖

父工程导入Shiro的依赖

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.3.2</version>
</dependency>
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-core</artifactId>
    <version>1.3.2</version>
</dependency>
<dependency>
    <groupId>org.crazycake</groupId>
    <artifactId>shiro-redis</artifactId>
    <version>3.0.0</version>
</dependency>
```

3.2.2 配置值对象

不需要存入redis太多的用户数据，和获取用户信息的返回对象一致即可，需要实现AuthCachePrincipali接口

```
@Setter
@Getter
public class ProfileResult implements Serializable,AuthCachePrincipal {
    private String mobile;
    private String username;
    private String company;
    private String companyId;
    private Map<String,Object> roles = new HashMap<>();
    //省略
}
```

3.2.3 配置未认证controller

为了在多个微服务中使用，配置公共的未认证未授权的Controller

```
@RestController
@CrossOrigin
public class ErrorController {

    //公共错误跳转
    @RequestMapping(value="autherror")
    public Result autherror(int code) {
        return code ==1?new Result(ResultCode.UNAUTHENTICATED):new
Result(ResultCode.UNAUTHORISE);
    }

}
```

3.2.4 自定义realm授权

ihrm-common模块下创建公共的认证与授权realm，需要注意的是，此realm只处理授权数据即可，认证方法需要在登录模块中补全。

```
public class IhrmRealm extends AuthorizingRealm {

    @Override
    public void setName(String name) {
        super.setName("ihrmRealm");
    }

    //授权方法
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        //1.获取安全数据
        ProfileResult result =
(ProfileResult)principalCollection.getPrimaryPrincipal();
        //2.获取权限信息
        Set<String> apisPerms = (Set<String>)result.getRoles().get("apis");
        //3.构造权限数据，返回值
        SimpleAuthorizationInfo info = new  SimpleAuthorizationInfo();
        info.setStringPermissions(apisPerms);
        return info;
    }

    /**
     * 认证方法
     */
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        return null;
    }
}
```

3.3.5 自定义会话管理器

之前的程序使用jwt的方式进行用户认证，前端发送后端的是请求头中的token。为了适配之前的程序，在shiro中需要更改sessionId的获取方式。很好解决，在shiro的会话管理中，可以轻松的使用请求头中的内容作为sessionid

```
public class IhrmWebSessionManager extends DefaultWebSessionManager {

    private static final String AUTHORIZATION = "Authorization";

    private static final String REFERENCED_SESSION_ID_SOURCE = "Stateless request";

    public IhrmWebSessionManager(){
        super();
    }

    protected Serializable getSessionId(ServletRequest request, ServletResponse
response){
        String id = WebUtils.toHttp(request).getHeader(AUTHORIZATION);
        if(StringUtils.isEmpty(id)){
            //如果没有携带id参数则按照父类的方式在cookie进行获取
            return super.getSessionId(request, response);
        }else{
            id = id.replace("Bearer ", "");
            //如果请求头中有 authToken 则其值为sessionId

 request.setAttribute(ShiroHttpServletRequest.REFERENCED_SESSION_ID_SOURCE,REFERENCED_S
ESSION_ID_SOURCE);
            request.setAttribute(ShiroHttpServletRequest.REFERENCED_SESSION_ID,id);

 request.setAttribute(ShiroHttpServletRequest.REFERENCED_SESSION_ID_IS_VALID,Boolean.TR
UE);
            return id;
        }
    }
}
```

3.3 用户认证

3.3.1 配置用户登录

```
    //用户名密码登录
    @RequestMapping(value="/login",method = RequestMethod.POST)
    public Result login(@RequestBody Map<String,String> loginMap) {
        String mobile = loginMap.get("mobile");
        String password = loginMap.get("password");
        try {
            //1.构造登录令牌 UsernamePasswordToken
            //加密密码
            password = new Md5Hash(password,mobile,3).toString();  //1.密码，盐，加密次数
            UsernamePasswordToken upToken = new UsernamePasswordToken(mobile,password);
            //2.获取subject
            Subject subject = SecurityUtils.getSubject();
            //3.调用login方法，进入realm完成认证
```

```
            subject.login(upToken);
            //4.获取sessionId
            String sessionId = (String)subject.getSession().getId();
            //5.构造返回结果
            return new Result(ResultCode.SUCCESS,sessionId);
        }catch (Exception e) {
            return new Result(ResultCode.MOBILEORPASSWORDERROR);
        }
    }
```

3.3.2 shiro认证

配置用户登录认证的realm域，只需要继承公共的IhrmRealm补充其中的认证方法即可

```
public class UserIhrmRealm extends IhrmRealm {

    @Override
    public void setName(String name) {
        super.setName("customRealm");
    }

    @Autowired
    private UserService userService;

    //认证方法
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        //1.获取用户的手机号和密码
        UsernamePasswordToken upToken = (UsernamePasswordToken) authenticationToken;
        String mobile = upToken.getUsername();
        String password = new String( upToken.getPassword());
        //2.根据手机号查询用户
        User user = userService.findByMobile(mobile);
        //3.判断用户是否存在，用户密码是否和输入密码一致
        if(user != null && user.getPassword().equals(password)) {
            //4.构造安全数据并返回（安全数据：用户基本数据，权限信息 profileResult）
            ProfileResult result = null;
            if("user".equals(user.getLevel())) {
                result = new ProfileResult(user);
            }else {
                Map map = new HashMap();
                if("coAdmin".equals(user.getLevel())) {
                    map.put("enVisible","1");
                }
                List<Permission> list = permissionService.findAll(map);
                result = new ProfileResult(user,list);
            }
            //构造方法：安全数据，密码，realm域名
            SimpleAuthenticationInfo info = new
SimpleAuthenticationInfo(result,user.getPassword(),this.getName());
            return info;
        }
        //返回null，会抛出异常，标识用户名和密码不匹配
```

```
            return null;
        }
    }
```

3.3.3 获取session数据

baseController中使用shiro从redis中获取认证数据

```
    //使用shiro获取
    @ModelAttribute
    public void setResAnReq(HttpServletRequest request,HttpServletResponse response) {
        this.request = request;
        this.response = response;

        //获取session中的安全数据
        Subject subject = SecurityUtils.getSubject();
        //1.subject获取所有的安全数据集合
        PrincipalCollection principals = subject.getPrincipals();
        if(principals != null && !principals.isEmpty()){
            //2.获取安全数据
            ProfileResult result = (ProfileResult)principals.getPrimaryPrincipal();
            this.companyId = result.getCompanyId();
            this.companyName = result.getCompany();
        }
    }
```

3.4 用户授权

在需要使用的接口上配置@RequiresPermissions("API-USER-DELETE")

3.5 配置

构造shiro的配置类

```
@Configuration
public class ShiroConfiguration {

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

    @Value("${spring.redis.port}")
    private int port;

    //配置自定义的Realm
    @Bean
    public IhrmRealm getRealm() {
        return new UserIhrmRealm();
    }

    //配置安全管理器
```

```
    @Bean
    public SecurityManager securityManager() {
        //使用默认的安全管理器
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        // 自定义session管理 使用redis
        securityManager.setSessionManager(sessionManager());
        // 自定义缓存实现 使用redis
        securityManager.setCacheManager(cacheManager());
        //将自定义的realm交给安全管理器统一调度管理
        securityManager.setRealm(getRealm());
        return securityManager;
    }

    //Filter工厂，设置对应的过滤条件和跳转条件
    @Bean
    public ShiroFilterFactoryBean shirFilter(SecurityManager securityManager) {
        //1.创建shiro过滤器工厂
        ShiroFilterFactoryBean filterFactory = new ShiroFilterFactoryBean();
        //2.设置安全管理器
        filterFactory.setSecurityManager(securityManager);
        //3.通用配置（配置登录页面，登录成功页面，验证未成功页面）
        filterFactory.setLoginUrl("/autherror?code=1"); //设置登录页面
        filterFactory.setUnauthorizedUrl("/autherror?code=2"); //授权失败跳转页面
        //4.配置过滤器集合
        /**
         * key   ：访问连接
         *       支持通配符的形式
         * value：过滤器类型
         *       shiro常用过滤器
         *           anno     ：匿名访问（表明此链接所有人可以访问）
         *           authc    ：认证后访问（表明此链接需登录认证成功之后可以访问）
         */
        Map<String,String> filterMap = new LinkedHashMap<String,String>();
        //配置请求连接过滤器配置
        //匿名访问（所有人员可以使用）
        filterMap.put("/frame/login", "anon");
        filterMap.put("/autherror", "anon");
        //认证之后访问（登录之后可以访问）
        filterMap.put("/**", "authc");

        //5.设置过滤器
        filterFactory.setFilterChainDefinitionMap(filterMap);
        return filterFactory;
    }

    //配置shiro注解支持
    @Bean
    public AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor(SecurityManager securityManager) {
        AuthorizationAttributeSourceAdvisor advisor = new
AuthorizationAttributeSourceAdvisor();
        advisor.setSecurityManager(securityManager);
        return advisor;
```

```
    }

    //配置shiro redisManager
    public RedisManager redisManager() {
        RedisManager redisManager = new RedisManager();
        redisManager.setHost(host);
        redisManager.setPort(port);
        return redisManager;
    }

    //cacheManager缓存 redis实现
    public RedisCacheManager cacheManager() {
        RedisCacheManager redisCacheManager = new RedisCacheManager();
        redisCacheManager.setRedisManager(redisManager());
        return redisCacheManager;
    }

    /**
     * RedisSessionDAO shiro sessionDao层的实现 通过redis
     * 使用的是shiro-redis开源插件
     */
    public RedisSessionDAO redisSessionDAO() {
        RedisSessionDAO redisSessionDAO = new RedisSessionDAO();
        redisSessionDAO.setRedisManager(redisManager());
        return redisSessionDAO;
    }

    /**
     * shiro session的管理
     */
    public DefaultWebSessionManager sessionManager() {
        IhrmWebSessionManager sessionManager = new IhrmWebSessionManager();
        sessionManager.setSessionDAO(redisSessionDAO());
        return sessionManager;
    }
}
```