Shiro框架

shiro代码示例

一、权限的管理

1.1 什么是权限管理

基本上涉及到用户参与的系统都要进行权限管理，权限管理属于系统安全的范畴，权限管理实现对用户访问系统的控制，按照安全规则或者安全策略控制用户可以访问而且只能访问自己被授权的资源。

权限管理包括用户身份认证和授权两部分，简称认证授权。对于需要访问控制的资源用户首先经过身份认证，认证通过后用户具有该资源的访问权限方可访问。

1.2 什么是身份认证

身份认证，就是判断一个用户是否为合法用户的处理过程。最常用的简单身份认证方式是系统通过核对用户输入的用户名和口令，看其是否与系统中存储的该用户的用户名和口令一致，来判断用户身份是否正确。对于采用指纹等系统，则出示指纹；对于硬件Key等刷卡系统，则需要刷卡。

1.3 什么是授权

授权，即访问控制，控制谁能访问哪些资源。主体进行身份认证后需要分配权限方可访问系统的资源，对于某些资源没有权限是无法访问的

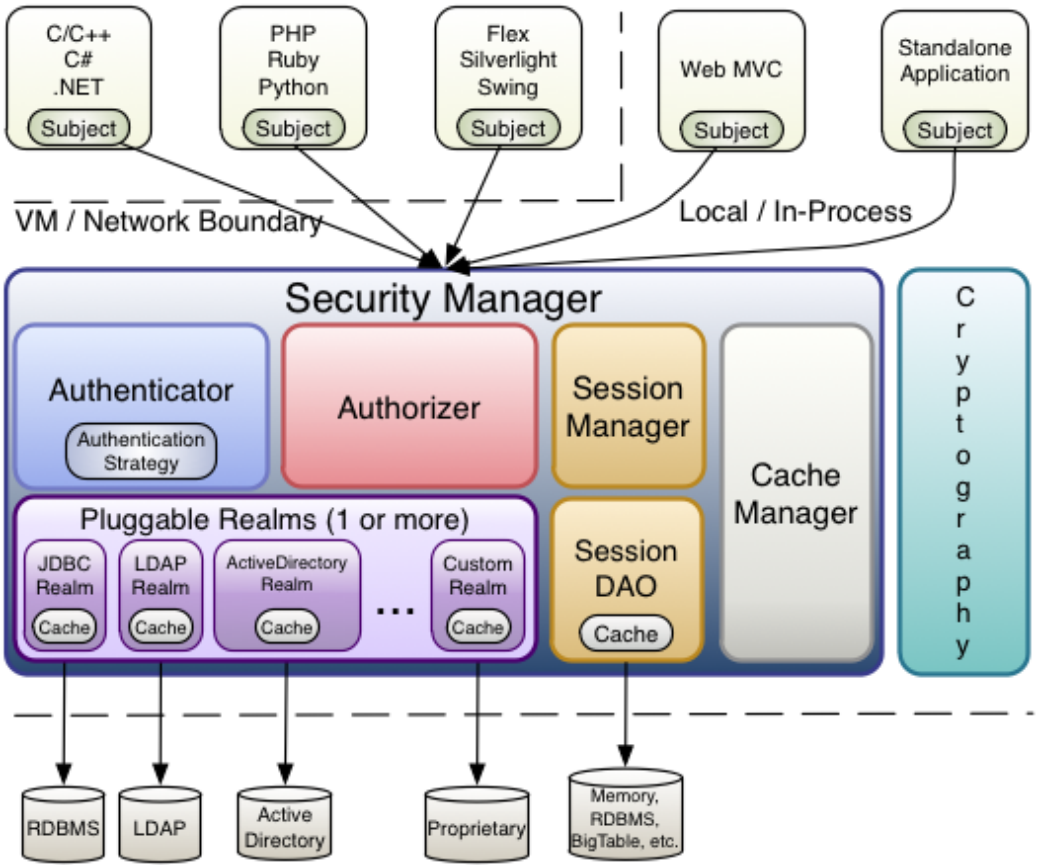
二、什么是shiro

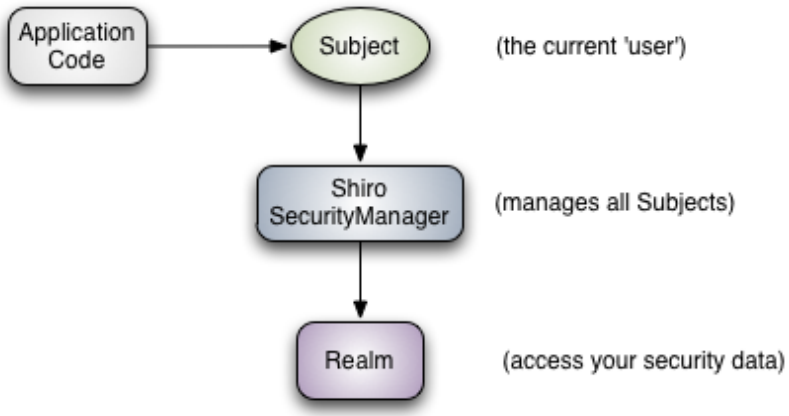
Apache Shiro™ is a powerful and easy-to-use Java security framework that performs authentication, authorization, cryptography, and session management. With Shiro’ s easy-to-understand API, you can quickly and easily secure any application – from the smallest mobile applications to the largest web and enterprise applications.

Shiro 是一个功能强大且易于使用的Java安全框架，它执行身份验证、授权、加密和会话管理。使用Shiro易于理解的API，您可以快速轻松地保护任何应用程序—从最小的移动应用程序到最大的web和企业应用程序。

Shiro是apache旗下一个开源框架，它将软件系统的安全认证相关的功能抽取出来，实现用户身份认证，权限授权、加密、会话管理等功能，组成了一个通用的安全认证框架。

三、shiro的核心架构





3.1 Subject

Subject即主体，外部应用与subject进行交互，subject记录了当前操作用户，将用户的概念理解为当前操作的主体，可能是一个通过浏览器请求的用户，也可能是一个运行的程序。

Subject在shiro中是一个接口，接口中定义了很多认证授相关的方法，外部程序通过subject进行认证授，而subject是通过SecurityManager安全管理器进行认证授权

3.2 SecurityManager

SecurityManager即安全管理器，对全部的subject进行安全管理，它是shiro的核心，负责对所有的subject进行安全管理。通过SecurityManager可以完成subject的认证、授权等，实质上SecurityManager是通过Authenticator进行认证，通过Authorizer进行授权，通过SessionManager进行会话管理等。

SecurityManager是一个接口，继承了Authenticator, Authorizer, SessionManager这三个接口。

3.3 Authenticator

Authenticator即认证器，对用户身份进行认证，Authenticator是一个接口，shiro提供ModularRealmAuthenticator实现类，通过ModularRealmAuthenticator基本上可以满足大多数需求，也可以自定义认证器。

3.4 Authorizer

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

3.5 Realm

Realm即领域，相当于datasource数据源，securityManager进行安全认证需要通过Realm获取用户权限数据，比如：如果用户身份数据在数据库那么realm就需要从数据库获取用户身份信息。

注意：不要把realm理解成只是从数据源取数据，在realm中还有认证授权校验的相关的代码。

3.6 SessionManager

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

3.7 SessionDAO

SessionDAO即会话dao，是对session会话操作的一套接口，比如要将session存储到数据库，可以通过jdbc将会话存储到数据库。

3.8 CacheManager

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

3.9 Cryptography

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

四、shiro中的认证

4.1 认证

身份认证，就是判断一个用户是否为合法用户的处理过程。最常用的简单身份认证方式是系统通过核对用户输入的用户名和口令，看其是否与系统中存储的该用户的用户名和口令一致，来判断用户身份是否正确。

4.2 shiro中认证的关键对象

1. Subject：主体

访问系统的用户，主体可以是用户、程序等，进行认证的都称为主体；

2. Principal：身份信息

是主体（subject）进行身份认证的标识，标识必须具有唯一性，如用户名、手机号、邮箱地址等，一个主体可以有多个身份，但是必须有一个主身份（Primary Principal）。

3. credential：凭证信息

是只有主体自己知道的安全信息，如密码、证书等。

4.3 认证流程

认证不是根据数据库中的数据，而是根据配置文件shiro.ini。

Java

```java
public class AuthenticatorTest {
    public static void main(String[] args) {
        // 1.创建安全管理器
        DefaultSecurityManager securityManager = new DefaultSecurityManager();

        // 2.给安全管理器设置realm  域对象
        securityManager.setRealm(new IniRealm("classpath:shiro.ini"));

        // 3.SecurityUtils给全局工具类设置安全管理器
        SecurityUtils.setSecurityManager(securityManager);

        // 4.关键对象subject主体
        Subject subject = SecurityUtils.getSubject();

        // 5.创建令牌  认证可能会抛出两种异常，用户名不对UnknownAccountException，密码不对IncorrectCredentialsException
        UsernamePasswordToken token = new UsernamePasswordToken("yjiewei", "123456");
        subject.login(token);

        // 6.查看验证结果
        System.out.println("认证结果: " + subject.isAuthenticated());
    }
}
```

源码关键

- 用户名校验

Java

```java
// org.apache.shiro.realm.SimpleAccountRealm#doGetAuthenticationInfo 获得一个账户信息
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
    UsernamePasswordToken upToken = (UsernamePasswordToken)token;
    SimpleAccount account = this.getUser(upToken.getUsername());
    if (account != null) {
        if (account.isLocked()) {
            throw new LockedAccountException("Account [" + account + "] is locked.");
        }

        if (account.isCredentialsExpired()) {
            String msg = "The credentials for account [" + account + "] are expired";
            throw new ExpiredCredentialsException(msg);
        }
    }

    return account;
}
```

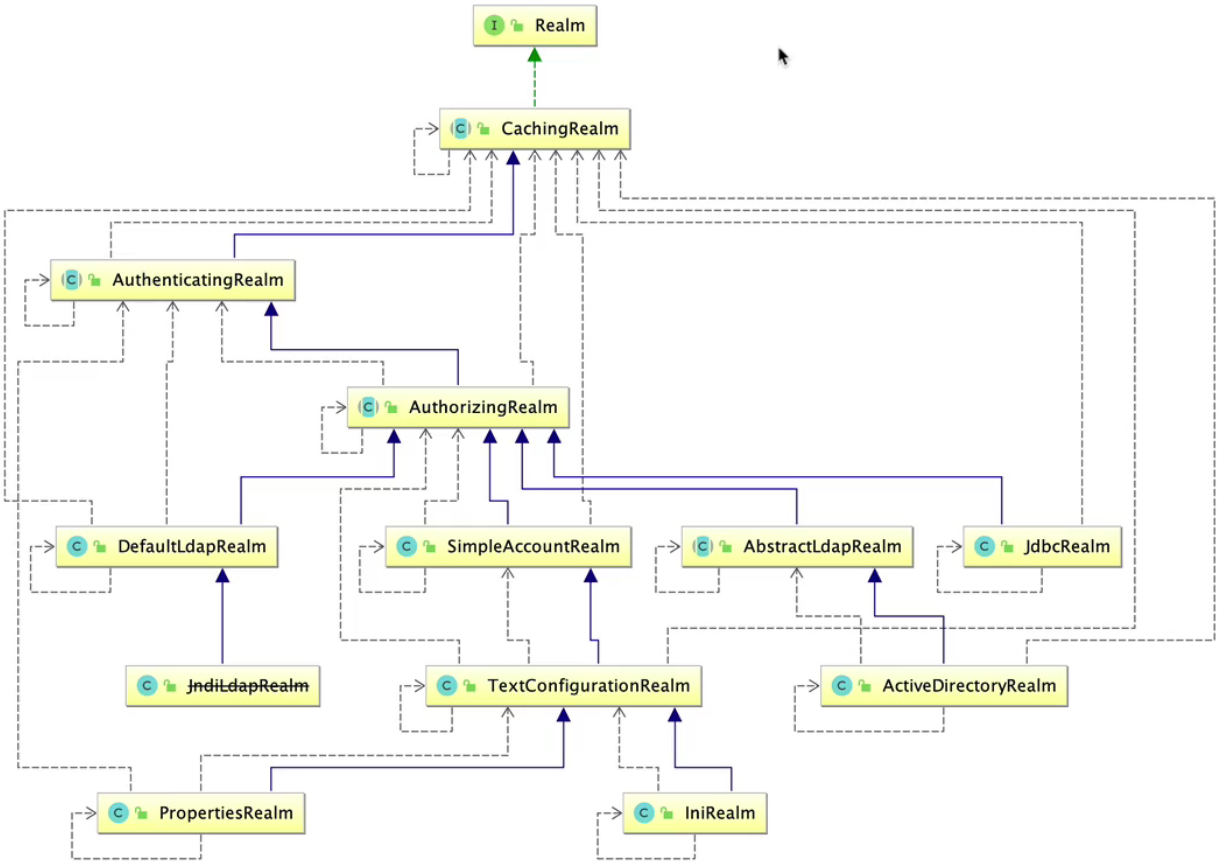
- 密码校验

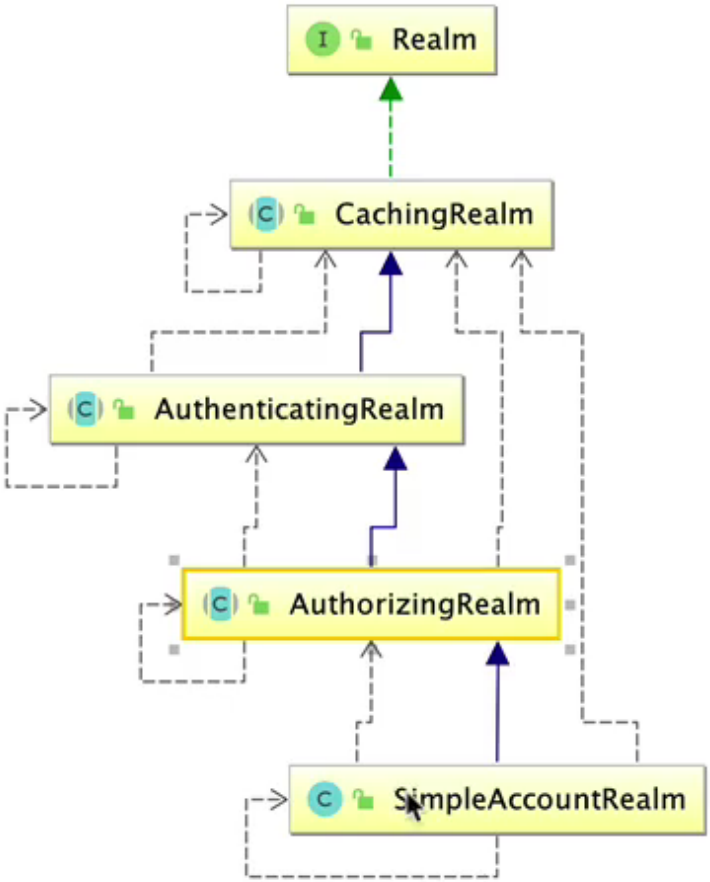
Java

```java
// org.apache.shiro.realm.AuthenticatingRealm#assertCredentialsMatch
// 最终比较是在这里
protected void assertCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) throws AuthenticationException {
    CredentialsMatcher cm = this.getCredentialsMatcher();
    if (cm != null) {
        if (!cm.doCredentialsMatch(token, info)) {
            String msg = "Submitted credentials for token [" + token + "] did not match the expected credentials.";
            throw new IncorrectCredentialsException(msg);
        }
    } else {
        throw new AuthenticationException("A CredentialsMatcher must be configured in order to verify credentials during authentication.  If you do not wish for credentials to be examined, you can configure an " + AllowAllCredentialsMatcher.class.getName() + " instance.");
    }
}
```

想法：是不是把查用户名部分的实现替换成自己的，从数据库中查找呢？

org.apache.shiro.realm.AuthorizingRealm





目前这个SimpleAccountRealm继承了AuthorizingRealm，重写了认证和授权方法。

4.4 总结

4.4.1 认证

1. 最终执行用户名比较 org.apache.shiro.realm.SimpleAccountRealm#doGetAuthenticationInfo
2. 最终密码校验是在 org.apache.shiro.realm.AuthenticatingRealm#assertCredentialsMatch

4.4.2 结论

1. AuthenticatingRealm 是认证realm，
   org.apache.shiro.realm.AuthenticatingRealm#doGetAuthenticationInfo
2. AuthorizingRealm 是授权realm，
   org.apache.shiro.realm.AuthorizingRealm#doGetAuthorizationInfo

4.5 自定义realm

TypeScript

```
public class CustomerRealmAuthenticatorTest {
    public static void main(String[] args) {
        // 1.创建安全管理器
        DefaultSecurityManager securityManager = new DefaultSecurityManager();

        // 2.给安全管理器设置realm  域对象
        securityManager.setRealm(new CustomerRealm());

        // 3.SecurityUtils给全局工具类设置安全管理器
        SecurityUtils.setSecurityManager(securityManager);

        // 4.关键对象subject主体
        Subject subject = SecurityUtils.getSubject();

        // 5.创建令牌  认证可能会抛出两种异常，用户名不对UnknownAccountException，密码不对IncorrectCredentialsException
        UsernamePasswordToken token = new UsernamePasswordToken("yjiewei", "123456");
        subject.login(token);

        // 6.查看验证结果
        System.out.println("认证结果: " + subject.isAuthenticated());
    }
}
```

Scala

```scala
/**
 * 自定义的realm实现，将认证/授权数据的来源转为数据库的实现
 * @author yjiewei
 * @date 2021/7/17
 */
public class CustomerRealm extends AuthorizingRealm {

    /**
     * 授权
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        return null;
    }

    /**
     * 认证
     * @param authenticationToken
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {
        String username = (String) authenticationToken.getPrincipal();
        System.out.println("自定义的realm获得登录的用户名是 " + username);
        // 假设我们从数据库中查询一个数据记录，然后这里和用户名进行比较
        if ("yjiewei".equals(username)) {
            // 参数 1.数据库用户名  2.返回数据库中的正确密码（再去别的地方比较）  3.当前realm名字
            SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo("yjiewei", "123456", this.getName());
            return authenticationInfo;
        }
        return null;
    }
}
```

4.6 使用MD5和salt

1. MD5是单向散列函数，只有原文到散列值，而散列值到原文基本不可能，不可破解。（摘要）

2. 加盐是为了防止用户用简单密码，导致穷举破解。
3. 同时salt还需要保存到数据库中，在验证用户信息的时候进行比对。

TypeScript

```
public class CustomerMD5RealmAuthenticatorTest {
    public static void main(String[] args) {
        // 1.创建安全管理器
        DefaultSecurityManager securityManager = new DefaultSecurityManager();

        // 2.给安全管理器设置realm  域对象
        CustomerMD5Realm realm = new CustomerMD5Realm();

        // 2.1 设置realm使用hash凭证匹配器 并设置使用密码
        HashedCredentialsMatcher hashedCredentialsMatcher = new
HashedCredentialsMatcher();
        hashedCredentialsMatcher.setHashAlgorithmName("MD5");
        // 2.2 告知散列次数，如果是md5+salt+hash1024
        hashedCredentialsMatcher.setHashIterations(1024);

        realm.setCredentialsMatcher(hashedCredentialsMatcher);
        securityManager.setRealm(realm);

        // 3.SecurityUtils给全局工具类设置安全管理器
        SecurityUtils.setSecurityManager(securityManager);

        // 4.关键对象subject主体
        Subject subject = SecurityUtils.getSubject();

        // 5.创建令牌  认证可能会抛出两种异常，用户名不对UnknownAccountException，密码
不对IncorrectCredentialsException
        UsernamePasswordToken token = new UsernamePasswordToken("yjiewei",
"123456");
        try{
            subject.login(token);
            System.out.println("登录成功");
        }catch (IncorrectCredentialsException exception) {
            exception.printStackTrace();
            System.out.println("密码错误");
        }catch (UnknownAccountException exception) {
            exception.printStackTrace();
            System.out.println("用户名错误");
        }
    }
}
```

Java

```java
/**
 * 使用自定义realm加入md5和salt和hash
 * @apiNote 这里假设都是从数据库中根据token中的用户名来查找数据库中匹配的用户信息，后续做凭证比对
 *          注意，这是默认的凭证比对是SimpleCredentialsMaster，如果我们的密码用了hash值，那么这个
 *          凭证比对器也需要修改，也就是具体比较的细节得换掉
 *          另外，验证时加入第三个参数的盐值，传入时不需要，凭证比较器会自动获取进行比较，反正凭证器只需要你指定hash的算法
 *          第四个参数是realm的名字
 *          如果散列1024次呢？ md5+salt+hash1024
 * @author yjiewei
 * @date 2021/7/17
 */
public class CustomerMD5Realm extends AuthorizingRealm {
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        return null;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {
        String principal = (String) authenticationToken.getPrincipal();
        // 第三个参数为随机盐
        if ("yjiewei".equals(principal)) {
            return new SimpleAuthenticationInfo(principal, "458c2980fa62f02a98280bb8c4795425", ByteSource.Util.bytes("1234567890"), this.getName());
        }
        return null;
    }
}
```

Java

```java
public class ShiroMD5Test {
    public static void main(String[] args) {
        // 1.不地道的用法
        Md5Hash md5Hash = new Md5Hash();
        md5Hash.setBytes("123456".getBytes(StandardCharsets.UTF_8));
        String s = md5Hash.toHex();
        System.out.println(s);

        // 2.构造方法使用
        Md5Hash hash = new Md5Hash("123456");
        System.out.println(hash.toHex());

        // 3.加盐构造方法使用，通常是随机数值或者字符串
        Md5Hash hash1 = new Md5Hash("123456", "1234567890");
        System.out.println(hash1.toHex());

        // 4.md5+salt+hash
        Md5Hash hash2 = new Md5Hash("123456", "1234567890", 1024);
        System.out.println(hash2.toHex());
    }
}
```

五、shiro中的授权

5.1 授权

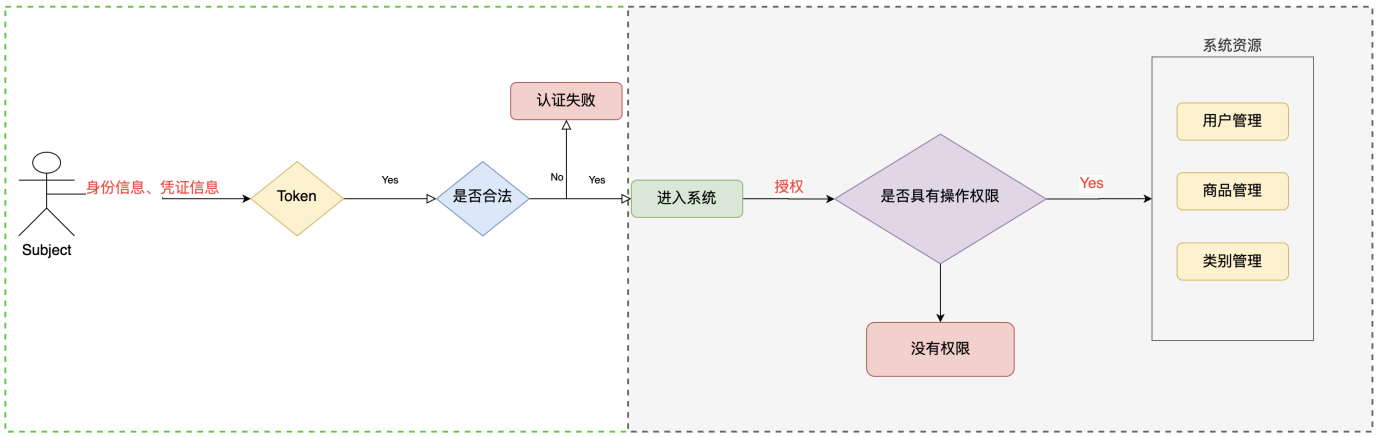
授权，即访问控制，控制谁能访问哪些资源。主体进行身份认证后需要分配权限方可访问系统的资源，对于某些资源没有权限是无法访问的。

5.2 关键对象

授权可简单理解为who对what(which)进行How操作：

1. **Who**，即主体（Subject），主体需要访问系统中的资源。
2. **What**，即资源（Resource)，如系统菜单、页面、按钮、类方法、系统商品信息等。资源包括资源类型和资源实例，比如商品信息为资源类型，类型为t01的商品为资源实例，编号为001的商品信息也属于资源实例。
3. **How**，权限/许可（Permission)，规定了主体对资源的操作许可，权限离开资源没有意义，如用户查询权限、用户添加权限、某个类方法的调用权限、编号为001用户的修改权限等，通过权限可知主体对哪些资源都有哪些操作许可。

5.3 授权流程



5.4 授权方式

- **基于角色的访问控制**
  - RBAC基于角色的访问控制（Role-Based Access Control）是以角色为中心进行访问控制

Java

```java
if (subject.hasRole("admin")) {
    // 操作哪些资源
}
```

- **基于资源的访问控制**
  - RBAC基于资源的访问控制（Resource-Based Access Control）是以资源为中心进行访问控制

Java

```java
if (subject.isPermission("user:*:create")) { // user:find:*  user:*:01  01号用户具有权限
    // 谁.权限.资源
    user:update:01   （资源实例）对01号用户具有修改权限
    user:*:01  01号用户具有权限
}

if (subject.isPermission("user:update:01")) {
    //  （资源实例）对01号用户具有修改权限
}

if (subject.isPermission("user:update:*")) {
    //  （资源类型）对01号用户具有修改权限
}
```

5.5 权限字符串

权限字符串的规则是：资源标识符：操作：资源实例标识符，意思是对哪个资源的哪个实例具有什么操作，“:”是资源/操作/实例的分割符，权限字符串也可以使用*通配符。

例子：

1. 用户创建权限：user:create，或user:create:*
2. 用户修改实例001的权限：user:update:001
3. 用户实例001的所有权限：user:*：001

5.6 shiro中授权编程实现方式

- 编程式

JavaScript

```
Subject subject = SecurityUtils.getSubject();
if(subject.hasRole(“admin”)) {
        //有权限
} else {
        //无权限
}
```

- 注解式

TypeScript

```
@RequiresRoles("admin")
public void hello() {
        //有权限
}
```

- 标签式

XML

```
JSP/GSP 标签：在JSP/GSP 页面通过相应的标签完成：
<shiro:hasRole name="admin">
        <!— 有权限—>
</shiro:hasRole>
注意： Thymeleaf 中使用shiro需要额外集成！
```

Java

```
public class CustomerMD5RealmAuthenticatorTest {
    public static void main(String[] args) {
        // 1.创建安全管理器
        DefaultSecurityManager securityManager = new DefaultSecurityManager();
```

```
        DefaultSecurityManager securityManager = new DefaultSecurityManager();

        // 2.给安全管理器设置realm  域对象
        CustomerMD5Realm realm = new CustomerMD5Realm();

        // 2.1 设置realm使用hash凭证匹配器 并设置使用密码
        HashedCredentialsMatcher hashedCredentialsMatcher = new
HashedCredentialsMatcher();
        hashedCredentialsMatcher.setHashAlgorithmName("MD5");
        // 2.2 告知散列次数，如果是md5+salt+hash1024
        hashedCredentialsMatcher.setHashIterations(1024);

        realm.setCredentialsMatcher(hashedCredentialsMatcher);
        securityManager.setRealm(realm);

        // 3.SecurityUtils给全局工具类设置安全管理器
        SecurityUtils.setSecurityManager(securityManager);

        // 4.关键对象subject主体
        Subject subject = SecurityUtils.getSubject();

        // 5.创建令牌  认证可能会抛出两种异常，用户名不对UnknownAccountException，密码
不对IncorrectCredentialsException
        UsernamePasswordToken token = new UsernamePasswordToken("yjiewei",
"123456");
        try{
            subject.login(token);
            System.out.println("登录成功");
        }catch (IncorrectCredentialsException exception) {
            exception.printStackTrace();
            System.out.println("密码错误");
        }catch (UnknownAccountException exception) {
            exception.printStackTrace();
            System.out.println("用户名错误");
        }

        // 6.认证用户进行授权
        if (subject.isAuthenticated()) {
            // 6.1 基于角色权限校验
            System.out.println(subject.hasRole("super"));

            // 6.2 基于多角色权限校验
            System.out.println(subject.hasAllRoles(Arrays.asList("admin",
"user")));

            // 6.3 是否拥有其中一个角色
            boolean[] booleans = subject.hasRoles(Arrays.asList("admin",
"super"));
            for (boolean aBoolean : booleans) {
```

```
                System.out.println(aBoolean);
            }

            // 6.4 基于权限字符串的访问控制  资源标识符：操作：资源类型
            boolean permitted = subject.isPermitted("user:update:01");// user模块的所有权限
            System.out.println("目前是否拥有user权限："+ permitted);

            // 6.5 分别有什么权限，全部权限
        }
    }
}
```

TypeScript

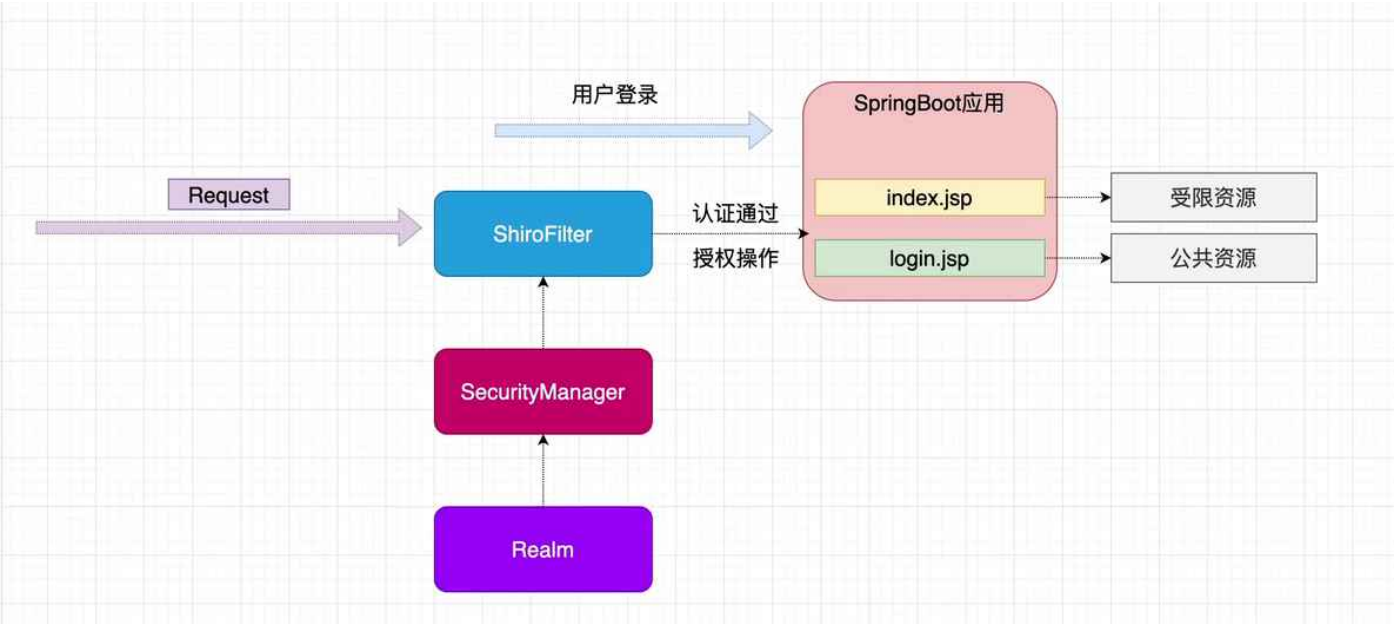
```
/**
 * 使用自定义realm加入md5和salt和hash
 * @apiNote 这里假设都是从数据库中根据token中的用户名来查找数据库中匹配的用户信息，后续做凭证比对
 *          注意，这是默认的凭证比对是SimpleCredentialsMaster，如果我们的密码用了hash值，那么这个
 *          凭证比对器也需要修改，也就是具体比较的细节得换掉
 *          另外，验证时加入第三个参数的盐值，传入时不需要，凭证比较器会自动获取进行比较，反正凭证器只需要你指定hash的算法
 *          第四个参数是realm的名字
 *          如果散列1024次呢？ md5+salt+hash1024
 * @author yjiewei
 * @date 2021/7/17
 */
public class CustomerMD5Realm extends AuthorizingRealm {
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        String primaryPrincipal = (String) principalCollection.getPrimaryPrincipal();
        System.out.println("当前拿到的身份是："+ primaryPrincipal);
        // 从数据库中获得权限信息，用户信息 yjiewei  admin  user
        Set<String> set = new HashSet<>();
        set.add("admin");
        set.add("user");
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo(set);
        // 另外一种设置角色的方法
        authorizationInfo.addRole("super");
        authorizationInfo.addStringPermission("user:*:01");
        return authorizationInfo;
    }

```

```
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {
        String principal = (String) authenticationToken.getPrincipal();
        // 第三个参数为随机盐
        if ("yjiewei".equals(principal)) {
            return new SimpleAuthenticationInfo(principal, "458c2980fa62f02a98280bb8c4795425", ByteSource.Util.bytes("1234567890"), this.getName());
        }
        return null;
    }
}
```

六、整合SpringBoot项目实战

spirngboot通过jsp整合shiro代码示例

6.1 整合思路



6.2 配置环境

6.2.1.创建项目

springboot+shiro+jsp

6.2.2 引入依赖

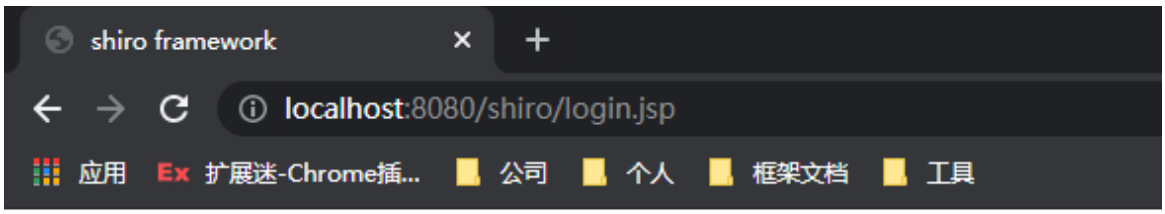
XML

```xml
<!--引入jsp解析依赖-->
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId> <!--好家伙，这里不能写版本号，原因不明-->
</dependency>
<!--引入shiro整合Springboot依赖-->
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring-boot-starter</artifactId>
    <version>1.5.3</version>
</dependency>
```

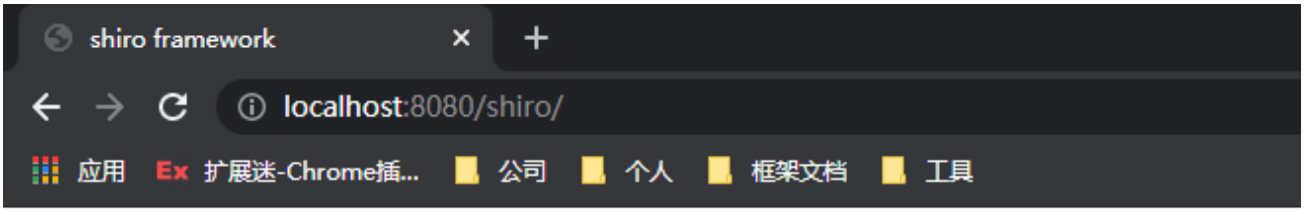
6.2.3 修改视图

XML

```xml
server.port=8080
server.servlet.context-path=/shiro
spring.application.name=shiro

spring.mvc.view.prefix=/
spring.mvc.view.suffix=.jsp
```



用户登录



系统主页V1.0

- 用户管理
- 商品管理
- 订单管理
- 物流管理

6.3 简单使用

6.3.1 创建配置类

用来整合shiro框架相关的配置类

Java

```java
/**
 * 整合shiro框架相关的配置类
 * @apiNote shiroFilter拦截到所有的请求之后，通过web安全管理器对请求进行处理，安全管理器需要realm来
 *          决定使用什么方式来做检验等工作，比如用的是jdbc还是mybatis等等
 * @author yjiewei
 * @date 2021/7/18
 */
@Configuration
public class ShiroConfig {

    // 1.创建shiroFilter，负责拦截所有请求
    @Bean
    public ShiroFilterFactoryBean getShiroFilterFactoryBean(DefaultWebSecurityManager defaultWebSecurityManager){
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        // 1.1配置安全管理器
        shiroFilterFactoryBean.setSecurityManager(defaultWebSecurityManager);

        // 1.2配置完安全管理器，你还得告诉拦截器你的系统那些是受限资源哪些是公共资源
        Map<String, String> map = new HashMap<>();
        map.put("/index.jsp", "authc"); // 访问资源，受限资源 authc（过滤器）代表请求这个资源需要认证和授权
```

```
        shiroFilterFactoryBean.setFilterChainDefinitionMap(map);

        // 1.3默认认证界面路径
        shiroFilterFactoryBean.setLoginUrl("/login.jsp");

        return shiroFilterFactoryBean;
    }

    // 2.创建web安全管理器
    @Bean
    public DefaultWebSecurityManager getDefaultWebSecurityManager(Realm realm){
        DefaultWebSecurityManager defaultWebSecurityManager = new
DefaultWebSecurityManager();
        defaultWebSecurityManager.setRealm(realm);

        return defaultWebSecurityManager;
    }

    // 3.创建自定义的realm
    @Bean
    @Primary
    public Realm getRealm(){
        return new CustomerRealm();
    }
}
```

Java

```
/**
 * 自定义realm
 * @author yjiewei
 * @date 2021/7/18
 */
public class CustomerRealm extends AuthorizingRealm {
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {
        return null;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        return null;
    }
}
```

HTML

```
<!-- index.jsp -->
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>shiro framework</title>
</head>
<body>
    <h1>系统主页V1.0</h1>
    <ul>
        <li><a href=""> 用户管理</a> </li>
        <li><a href=""> 商品管理</a> </li>
        <li><a href=""> 订单管理</a> </li>
        <li><a href=""> 物流管理</a> </li>
    </ul>
</body>
</html>
```

HTML

```
<!-- login.jsp -->
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>shiro framework</title>
</head>
<body>
    <h1>用户登录</h1>
</body>
</html>
```

6.4 常见过滤器

注意: shiro提供和多个默认的过滤器，我们可以用这些过滤器来配置控制指定url的权限：

	A	B	C
1	**anon**	**AnonymousFilter**	**指定url可以匿名访问（访问时不需要认证授权）**
2	**authc**	**FormAuthenticationFilter**	**指定url需要form表单登录，默认会从请求中获取username、password,rememberMe等参数并尝试登录，如果登录不了就跳转到loginUrl配置的路径。我们也可以用这个过滤器做默认的登录逻辑，但是一般都是我们自己在控制器写登录逻辑的，自己写的话出错返回的信息都可以定制嘛**
3	authcBasic	BasicHttpAuthenticationFilter	指定url需要basic登录
4	logout	LogoutFilter	登出过滤器，配置指定url就可以实现退出功能，非常方便
5	noSessionCreation	NoSessionCreationFilter	禁止创建会话
6	perms	PermissionsAuthorizationFilter	需要指定权限才能访问
7	port	PortFilter	需要指定端口才能访问
8	rest	HttpMethodPermissionFilter	将http请求方法转化成相应的动词来构造一个权限字符串，这个感觉意义不大，有兴趣自己看源码的注释
9	roles	RolesAuthorizationFilter	需要指定角色才能访问
10	ssl	SslFilter	需要https请求才能访问
11	user	UserFilter	需要已登录或“记住我”的用户才能访问

6.5 认证和退出实现

6.5.1 登录实现

1.login.jsp

XML

```xml
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>shiro framework</title>
</head>
<body>
    <h1>用户登录</h1>

    <form action="${pageContext.request.contextPath}/user/login" method="post">
        用户名: <input type="text" name="username" > <br>
        密码: <input type="password" name="password"> <br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

2.UserController

Java

```java
/**
 * 用户身份验证
 * @param username
 * @param password
 * @return
 */
@RequestMapping("login")
public String login(String username, String password){
    // 1.获取主体对象
    Subject subject = SecurityUtils.getSubject();
    try{
        subject.login(new UsernamePasswordToken(username, password));
        return "redirect:/index.jsp";
    }catch (UnknownAccountException e){
        System.out.println("用户名错误");
    }catch (IncorrectCredentialsException e){
        System.out.println("密码错误");
    }
    return "redirect:/login.jsp";

}
```

4.ShiroConfig

主要的Shiro配置类中声明：哪些是需要验证的资源，哪些是公开的资源

TypeScript

```
/**
 * 整合shiro框架相关的配置类
 * @apiNote shiroFilter拦截到所有的请求之后，通过web安全管理器对请求进行处理，安全管理器需要realm来
 *          决定使用什么方式来做检验等工作，比如用的是jdbc还是mybatis等等
 * @author yjiewei
 * @date 2021/7/18
 */
@Configuration
public class ShiroConfig {

    // 1.创建shiroFilter，负责拦截所有请求
    @Bean
    public ShiroFilterFactoryBean getShiroFilterFactoryBean(DefaultWebSecurityManager defaultWebSecurityManager){
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        // 1.1配置安全管理器
        shiroFilterFactoryBean.setSecurityManager(defaultWebSecurityManager);

        // 1.2配置完安全管理器，你还得告诉拦截器你的系统那些是受限资源哪些是公共资源
        //    对于受限资源，或者公共资源，通常使用通配符来匹配，不然代码看起来可不太优雅
        Map<String, String> map = new HashMap<>();
        //map.put("/index.jsp", "authc"); // 访问资源，受限资源 authc（过滤器）代表请求这个资源需要认证和授权
        map.put("/user/login", "anon");
        map.put("/user/register", "anon");
        map.put("/register.jsp", "anon");
        map.put("/**", "authc");
        shiroFilterFactoryBean.setFilterChainDefinitionMap(map);

        // 1.3默认认证界面路径
        shiroFilterFactoryBean.setLoginUrl("/login.jsp");

        return shiroFilterFactoryBean;
    }

    // 2.创建web安全管理器
    @Bean
    public DefaultWebSecurityManager getDefaultWebSecurityManager(Realm realm){
        DefaultWebSecurityManager defaultWebSecurityManager = new DefaultWebSecurityManager();
        defaultWebSecurityManager.setRealm(realm);

```

```
        return defaultWebSecurityManager;
    }

    // 3.创建自定义的realm
    @Bean
    @Primary
    public Realm getRealm(){
        return new CustomerRealm();
    }

}
```

6.5.2 退出认证

1.index.jsp

添加登出链接

HTML

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>shiro framework</title>
</head>
<body>
<h1>系统主页V1.0</h1><a href="${pageContext.request.contextPath}/user/logout">退出登录</a>
    <ul>
        <li><a href=""> 用户管理</a> </li>
        <li><a href=""> 商品管理</a> </li>
        <li><a href=""> 订单管理</a> </li>
        <li><a href=""> 物流管理</a> </li>
    </ul>
</body>
</html>
```

2.UserController

Java

```java
/**
 * 退出登录
 * @return 重定向到登录界面
 */
@RequestMapping("/logout")
public String logout(){
    Subject subject = SecurityUtils.getSubject();
    subject.logout();
    return "redirect:/login.jsp";
}
```

6.6 MD5、Salt的认证实现

6.6.1 用户注册+随机盐处理

1.导入依赖

HTML

```html
<!-- mybatis依赖 -->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.1</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.16</version>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.2.6</version>
</dependency>
```

2.application.properties

Java

```
server.port=8080
server.servlet.context-path=/shiro
spring.application.name=shiro

spring.mvc.view.prefix=/
spring.mvc.view.suffix=.jsp

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# 这里我废了很大的劲，一个是时区问题，一个是mapper-locations的路径问题，由于idea没有将目录展开，导致框架识别不到
# 另外，mapper文件和dao类的名字要保持一致
spring.datasource.url=jdbc:mysql:///shiro?characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
spring.datasource.username=root
spring.datasource.password=root

mybatis.type-aliases-package=com.yjiewei.entity
mybatis.mapper-locations=classpath:com/yjiewei/mapper/*.xml
```

3.数据库

SQL

```sql
/*
Navicat MySQL Data Transfer

Source Server         : netca
Source Server Version : 80016
Source Host           : localhost:3306
Source Database       : shiro

Target Server Type    : MYSQL
Target Server Version : 80016
File Encoding         : 65001

Date: 2021-07-18 14:04:40
*/

SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for t_user
-- ----------------------------
DROP TABLE IF EXISTS `t_user`;
CREATE TABLE `t_user` (
  `id` int(6) NOT NULL AUTO_INCREMENT,
  `username` varchar(40) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `salt` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

4.创建entity

Java

```java
@Data
@Accessors(chain = true)
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private Integer id;
    private String username;
    private String password;
    private String salt;
}
```

5.创建DAO接口

Java

```java
@Mapper
public interface UserDao {

    void save(User user);
}
```

6.开发mapper配置文件

注意：mapper文件的位置要在 application.properties配置的目录下面

注意：mapper文件的命名 与 Dao接口保持一致

XML

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.yjiewei.dao.UserDao">
    <insert id="save" parameterType="User" useGeneratedKeys="true"
keyProperty="id">
        insert into t_user values(#{id},#{username},#{password},#{salt})
    </insert>
</mapper>
```

7.开发service接口

Java

```java
public interface UserService {
    // 注册用户方法
    void register(User user);
}

// 实现类
@Component
public class UserServiceImpl implements UserService{

    @Resource
    private UserDao userDao;

    @Override
    public void register(User user) {
        // 处理业务 md5+salt+hash
        String salt = SaltUtil.getSalt(10);
        user.setSalt(salt);
        Md5Hash md5Hash = new Md5Hash(user.getPassword(), salt, 1024);
        user.setPassword(md5Hash.toHex());
        userDao.save(user);
    }
}
```

8.创建salt工具类

Java

```java
public class SaltUtil {
    /**
     * 生成salt的静态方法
     * @param n
     * @return
     */
    public static String getSalt(int n) {
        StringBuilder sb = new StringBuilder();
        char[] chars =
"QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm7894561230!@#$%^&*
()".toCharArray();
        for (int i = 0; i < n; i++) {
            char c = chars[new Random().nextInt(chars.length)];
            sb.append(c);
        }
        return sb.toString();
    }
}
```

9.开发Controller

Java

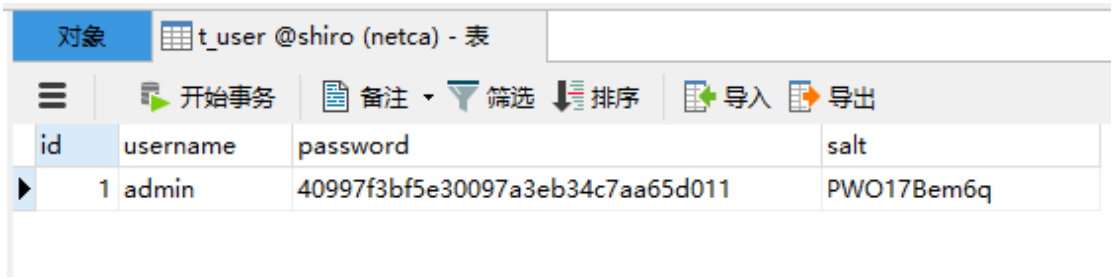
```java
/**
 * 用户注册
 * @param user
 * @return
 */
@RequestMapping("register")
public String register(User user){
    try{
        userService.register(user);
        return "redirect:/login.jsp";
    }catch (Exception e) {
        e.printStackTrace();
        return "redirect:/register.jsp";
    }
}
```

10.设置公共资源

在ShiroConfig中添加

```Go
map.put("/user/register","anon");//anon 设置为公共资源
map.put("/register.jsp","anon");//anon 设置为公共资源
```

11.测试注册



6.6.2 开发数据库认证

1. 为什么不直接注入，要写一个工具类？工具类中如何注入？自动注入吗

CustomerRealm不是放入工厂中的，所以没有办法直接注入userService对象，只能从工厂中获取进行使用。

1.开发DAO

```Java
@Mapper
public interface UserDao {

    void save(User user);

    // 根据身份信息认证的方法
    User findByUserName(String username);
}
```

2.开发mapper配置文件

```XML
<select id="findByUserName" parameterType="String" resultType="User">
  select id,username,password,salt from t_user
  where username = #{username}
</select>
```

3.开发Service接口

Java

```java
public interface UserService {
    // 注册用户方法
    void register(User user);

    // 根据用户名查询业务的方法
    User findByUserName(String username);
}
```

4.开发Service实现类

Java

```java
@Service("userService")
@Transactional
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDAO userDAO;
    @Override
    public User findByUserName(String username) {
        return userDAO.findByUserName(username);
    }
}
```

5.开发工厂工具类

Java

```java
/**
 * ApplicationContext工具类
 * @author yjiewei
 * @date 2021/7/21
 */
@Component
public class ApplicationContextUtils implements ApplicationContextAware {

    private static ApplicationContext context;

    @Override  // 这里根据注入情况添加
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
        context = applicationContext;
    }

    // 根据bean的名字获取容器中的bean
    public static Object getBean(String beanName){
        return context.getBean(beanName);
    }
}
```

6.修改自定义realm

Scala

```scala
/**
 * 自定义realm
 * @author yjiewei
 * @date 2021/7/18
 */
public class CustomerRealm extends AuthorizingRealm {
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        return null;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

        String username = (String) authenticationToken.getPrincipal();
        // 模拟实现
        /*if ("yjiewei".equals(username)){
            return new SimpleAuthenticationInfo(username, "123456", this.getName());
        }*/
        // 从数据库中获取对应的用户名及密码等信息做认证
        // 1.在工厂中获取service对象
        UserService userService = (UserService) ApplicationContextUtils.getBean("userService");

        // 2.根据身份信息做查询
        User user = userService.findByUserName(username);

        // 3.用户不为空，则返回数据库信息
        if (!ObjectUtils.isEmpty(user)) {
            // 3.1返回数据库中的信息，底层和用户输入的信息做校验
            SimpleAuthenticationInfo simpleAuthenticationInfo = new SimpleAuthenticationInfo(user.getUsername(),
                    user.getPassword(),
                    ByteSource.Util.bytes(user.getSalt()),
                    this.getName());
            return simpleAuthenticationInfo;
        }
        return null;
    }
}
```

7.修改ShiroConfig中realm

这里只修改最后一个方法

TypeScript

```
/**
 * 整合shiro框架相关的配置类
 * @apiNote shiroFilter拦截到所有的请求之后，通过web安全管理器对请求进行处理，安全管理器需要realm来
 *          决定使用什么方式来做检验等工作，比如用的是jdbc还是mybatis等等
 * @author yjiewei
 * @date 2021/7/18
 */
@Configuration
public class ShiroConfig {

    // 1.创建shiroFilter，负责拦截所有请求
    @Bean
    public ShiroFilterFactoryBean getShiroFilterFactoryBean(DefaultWebSecurityManager defaultWebSecurityManager){
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        // 1.1配置安全管理器
        shiroFilterFactoryBean.setSecurityManager(defaultWebSecurityManager);

        // 1.2配置完安全管理器，你还得告诉拦截器你的系统那些是受限资源哪些是公共资源
        //    对于受限资源，或者公共资源，通常使用通配符来匹配，不然代码看起来可不太优雅
        Map<String, String> map = new HashMap<>();
        //map.put("/index.jsp", "authc"); // 访问资源，受限资源 authc（过滤器）代表请求这个资源需要认证和授权
        map.put("/user/login", "anon");
        map.put("/user/register", "anon");
        map.put("/register.jsp", "anon");
        map.put("/**", "authc");
        shiroFilterFactoryBean.setFilterChainDefinitionMap(map);

        // 1.3默认认证界面路径
        shiroFilterFactoryBean.setLoginUrl("/login.jsp");

        return shiroFilterFactoryBean;
    }

    // 2.创建web安全管理器
    @Bean
    public DefaultWebSecurityManager getDefaultWebSecurityManager(Realm realm){
        DefaultWebSecurityManager defaultWebSecurityManager = new DefaultWebSecurityManager();
        defaultWebSecurityManager.setRealm(realm);
```

```
39
40          return defaultWebSecurityManager;
41      }
42
43      // 3.创建自定义的realm
44      // 为什么要设置凭证匹配器呢?
45      //      首先你放入数据库的密码使用了散列，以及散列次数并加了盐，由于盐不变可以从数据
    库中获取，
46      //      再者重新登录做校验时，相当于再次对你的密码做加密，但是realm是不知道你之前怎么
    做的加密，你得告诉它，
47      //      最后，凭证匹配器是shiro内部实现的，相当于你要用什么方式来做校验。
48      @Bean
49      @Primary
50      public Realm getRealm(){
51
52          // realm是用来定义你的校验方式，域对象，设置你的凭证匹配器，加密方式，散列次数
53          CustomerRealm customerRealm = new CustomerRealm();
54
55          // 1.设置凭证匹配器
56          HashedCredentialsMatcher credentialsMatcher = new
    HashedCredentialsMatcher();
57          // 2.匹配器中使用的摘要算法
58          credentialsMatcher.setHashAlgorithmName("md5");
59          // 3.散列的次数
60          credentialsMatcher.setHashIterations(1024);
61
62          customerRealm.setCredentialsMatcher(credentialsMatcher);
63          return customerRealm;
64      }
65
66  }
```

8.测试

未登录则会跳转到登录页面，anon为放行资源。

测试成功。

6.7授权实现

6.7.1 没有数据库

修改认证后的权限。

Java

```java
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

    /**
     * @note 当你设置资源受限时，后台授权得有匹配的角色和权限去获取相对应的资源，前面是锁，这个是看你有没有钥匙🔑
     */
    SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
    // 新增两个角色
    authorizationInfo.addRole("user");
    authorizationInfo.addRole("user_manager");
    // user用户对所有资源类型都有新增和删除功能
    authorizationInfo.addStringPermission("user:add:*");
    authorizationInfo.addStringPermission("user:delete:*");

    return authorizationInfo;
}
```

1. 页面资源限制

修改前面的页面访问，只有对应的权限才能访问。

XML

```xml
<%@ taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>shiro framework</title>
</head>
<body>
<%-- 设置资源访问受限 --%>
<h1>系统主页V1.0</h1><a href="${pageContext.request.contextPath}/user/logout">退出登录</a>
    <ul>
        <shiro:hasAnyRoles name="user_manager, admin, addinfo_manager">
            <li><a href=""> 用户管理</a>
                <ul>
                    <shiro:hasPermission name="user:add:*">
                        <li><a href=""> 添加 </a> </li>
                    </shiro:hasPermission>
                    <shiro:hasPermission name="user:delete:*">
                        <li><a href=""> 删除 </a> </li>
                    </shiro:hasPermission>
```

```
                    <shiro:hasPermission name="user:update:*">
                        <li><a href=""> 修改 </a> </li>
                    </shiro:hasPermission>
                    <shiro:hasPermission name="user:find:*">
                        <li><a href=""> 查询 </a> </li>
                    </shiro:hasPermission>
                </ul>
            </li>
        </shiro:hasAnyRoles>

        <shiro:hasAnyRoles name="user_manager, admin, addinfo_manager">
            <li><a href=""> 订单管理</a>
                <ul>
                    <shiro:hasPermission name="user:add:*">
                        <li><a href=""> 添加 </a> </li>
                    </shiro:hasPermission>
                    <shiro:hasPermission name="user:delete:*">
                        <li><a href=""> 删除 </a> </li>
                    </shiro:hasPermission>
                    <shiro:hasPermission name="user:update:*">
                        <li><a href=""> 修改 </a> </li>
                    </shiro:hasPermission>
                    <shiro:hasPermission name="user:find:*">
                        <li><a href=""> 查询 </a> </li>
                    </shiro:hasPermission>
                </ul>
            </li>
        </shiro:hasAnyRoles>

        <shiro:hasRole name="admin">
            <li><a href=""> 商品管理</a> </li>
            <li><a href=""> 物流管理</a> </li>
        </shiro:hasRole>

        <shiro:hasRole name="user">
            <li><a href=""> 仅普通用户可见</a> </li>
            <li><a href=""> 公共资源 </a> </li>
        </shiro:hasRole>
    </ul>
</body>
</html>
```

2. 代码方式控制访问

Java

```java
/**
 * 模拟保存方法
 * 实际上知识为了测试页面访问URL时是否有权限访问
 * @return
 */
@RequestMapping("save")
public String save(){
    System.out.println("进入方法");

    //1.基于角色
    //获取主体对象
    Subject subject = SecurityUtils.getSubject();
    //代码方式
    if (subject.hasRole("admin")) {
        System.out.println("保存订单!");
    }else{
        System.out.println("无权访问!");
    }
    //2.基于权限字符串
    // ...
    return "redirect:/index.jsp";
}
```

3. 通过方法注解限制

- @RequiresRoles 用来基于角色进行授权
- @RequiresPermissions 用来基于权限进行授权

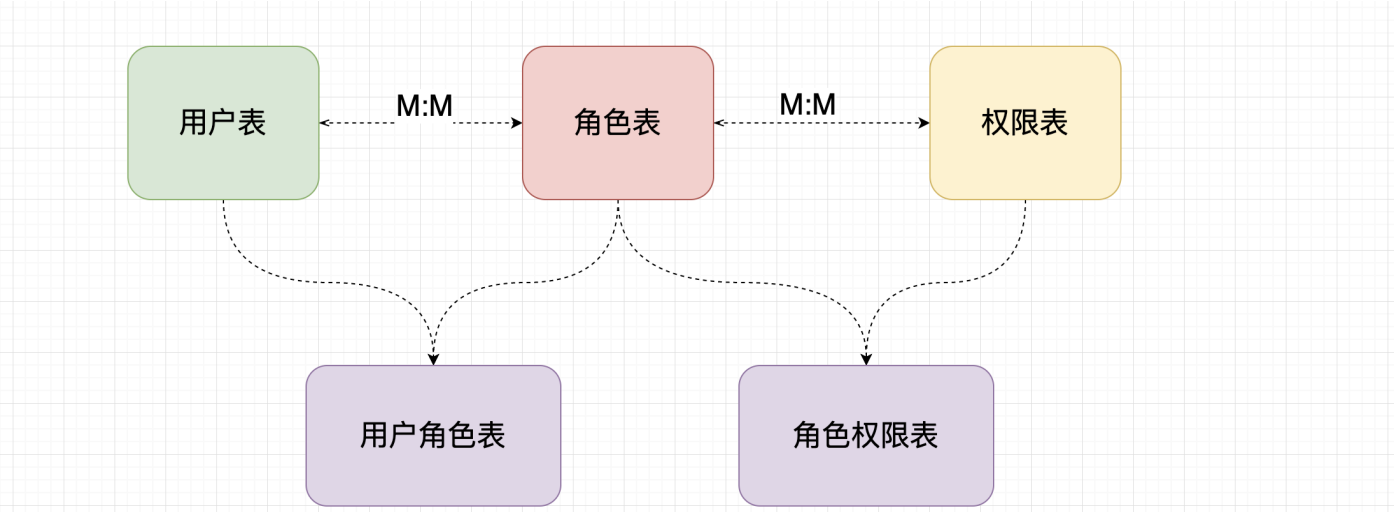
Java

```java
@Controller
@RequestMapping("order")
public class OrderController {

    @RequiresRoles(value={"admin","user"})//用来判断角色  同时具有 admin user
    @RequiresPermissions("user:update:01") //用来判断权限字符串
    @RequestMapping("save")
    public String save(){
        System.out.println("进入方法");
        return "redirect:/index.jsp";
    }

}
```

没有权限直接就报错了。

6.7.2 连接数据库做授权

1. 授权数据持久化

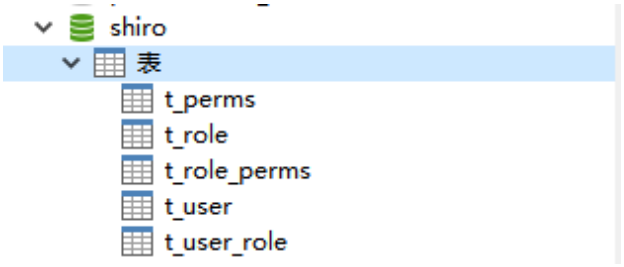


数据库的建表语句（这一块的数据库表设计可以回看）

SQL

```sql
/*
Navicat MySQL Data Transfer

Source Server         : 本机数据库
Source Server Version : 80016
Source Host           : localhost:3306
Source Database       : shiro

Target Server Type    : MYSQL
Target Server Version : 80016
File Encoding         : 65001

Date: 2021-07-22 12:01:49
*/

SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for t_perms
-- ----------------------------
DROP TABLE IF EXISTS `t_perms`;
CREATE TABLE `t_perms` (
  `id` int(6) NOT NULL AUTO_INCREMENT,
  `name` varchar(80) DEFAULT NULL,
  `url` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for t_role
-- ----------------------------
DROP TABLE IF EXISTS `t_role`;
CREATE TABLE `t_role` (
  `id` int(6) NOT NULL AUTO_INCREMENT,
  `name` varchar(60) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for t_role_perms
-- ----------------------------
DROP TABLE IF EXISTS `t_role_perms`;
CREATE TABLE `t_role_perms` (
  `id` int(6) NOT NULL,
  `roleid` int(6) DEFAULT NULL,
  `permsid` int(6) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for t_user
-- ----------------------------
DROP TABLE IF EXISTS `t_user`;
CREATE TABLE `t_user` (
  `id` int(6) NOT NULL AUTO_INCREMENT,
  `username` varchar(40) DEFAULT NULL,
  `password` varchar(40) DEFAULT NULL,
  `salt` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Table structure for t_user_role
-- ----------------------------
DROP TABLE IF EXISTS `t_user_role`;
CREATE TABLE `t_user_role` (
  `id` int(6) NOT NULL,
  `userid` int(6) DEFAULT NULL,
  `roleid` int(6) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```



2. 创建实体类

Java

```java
@Data
@Accessors(chain = true)
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private Integer id;
    private String username;
    private String password;
    private String salt;

    //定义角色集合
    private List<Role> roles;
}
```

Java

```java
@Data
@Accessors(chain = true)
@AllArgsConstructor
@NoArgsConstructor
public class Role implements Serializable {
    private String id;
    private String name;

    //定义权限的集合
    private List<Perms> perms;

}
```

Java

```java
@Data
@Accessors(chain = true)
@AllArgsConstructor
@NoArgsConstructor
public class Perms implements Serializable {
    private String id;
    private String name;
    private String url;
}
```

3. 创建dao方法

C++

```cpp
//根据用户名查询所有角色
User findRolesByUserName(String username);

//根据角色id查询权限集合
List<Perms> findPermsByRoleId(String id);
```

4. service接口

Java

```java
//根据用户名查询所有角色
User findRolesByUserName(String username);
//根据角色id查询权限集合
List<Perms> findPermsByRoleId(String id);
```

5. service实现

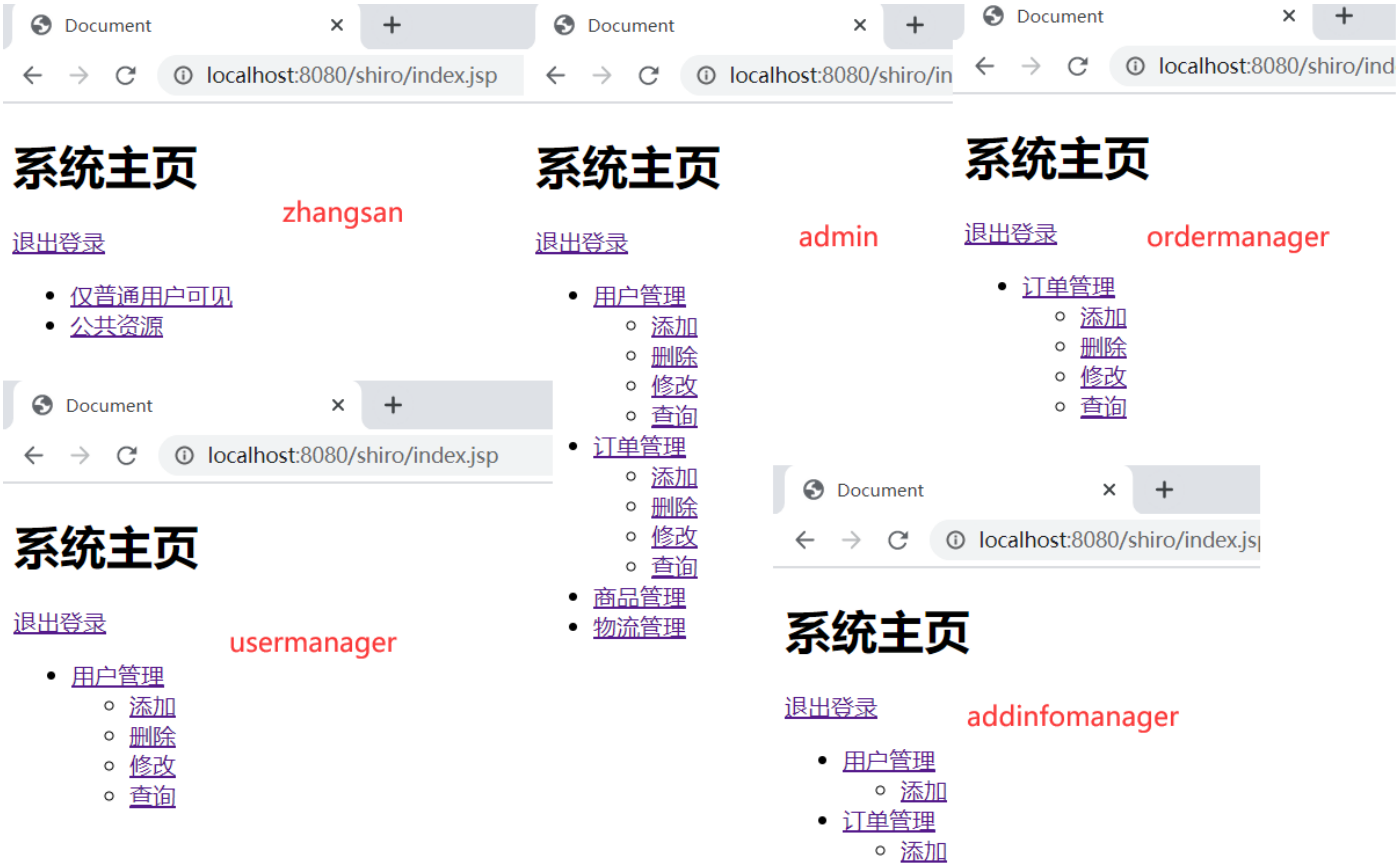
Java

```java
@Override
public List<Perms> findPermsByRoleId(String id) {
  return userDAO.findPermsByRoleId(id);
}

@Override
public User findRolesByUserName(String username) {
  return userDAO.findRolesByUserName(username);
}
```

6. mapper实现

Java

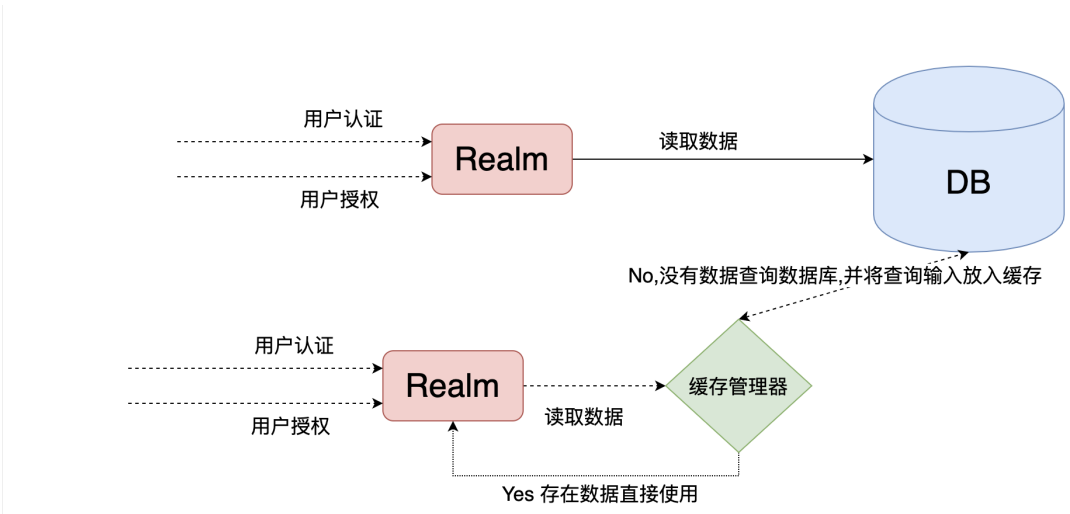
```
<resultMap id="userMap" type="User">
  <id column="uid" property="id"/>
  <result column="username" property="username"/>
  <!--角色信息-->
  <collection property="roles" javaType="list" ofType="Role">
    <id column="id" property="id"/>
    <result column="rname" property="name"/>
  </collection>
</resultMap>

<select id="findRolesByUserName" parameterType="String" resultMap="userMap">
  SELECT u.id uid,u.username,r.id,r.name rname
  FROM t_user u
  LEFT JOIN t_user_role ur
  ON u.id=ur.userid
  LEFT JOIN t_role r
  ON ur.roleid=r.id
  WHERE u.username=#{username}
</select>

<select id="findPermsByRoleId" parameterType="String" resultType="Perms">
  SELECT p.id,p.NAME,p.url,r.name
  FROM t_role r
  LEFT JOIN t_role_perms rp
  ON r.id=rp.roleid
  LEFT JOIN t_perms p ON rp.permsid=p.id
  WHERE r.id=#{id}
</select>
```



6.8 使用CacheManager

6.8.1 Cache 作用

- Cache 缓存: **计算机内存中一段数据**
- 作用: **用来减轻DB的访问压力,从而提高系统的查询效率**
- 流程:



6.8.2 使用shiro中默认EhCache实现缓存

1. 引入依赖

XML

```xml
<!--引入shiro和ehcache-->
<dependency>
  <groupId>org.apache.shiro</groupId>
  <artifactId>shiro-ehcache</artifactId>
  <version>1.5.3</version>
</dependency>
```

2. 开启缓存

Java

```java
@Bean
@Primary
public Realm getRealm(){

    // realm是用来定义你的校验方式，域对象，设置你的凭证匹配器，加密方式，散列次数
    CustomerRealm customerRealm = new CustomerRealm();

    // 1.设置凭证匹配器
    HashedCredentialsMatcher credentialsMatcher = new
HashedCredentialsMatcher();
    // 2.匹配器中使用的摘要算法
    credentialsMatcher.setHashAlgorithmName("md5");
    // 3.散列的次数
    credentialsMatcher.setHashIterations(1024);

    // 4.开启缓存管理器
    customerRealm.setCachingEnabled(true);
    customerRealm.setAuthenticationCachingEnabled(true);
    customerRealm.setAuthorizationCachingEnabled(true);
    customerRealm.setCacheManager(new EhCacheManager());

    customerRealm.setCredentialsMatcher(credentialsMatcher);
    return customerRealm;
}
```

3. 启动刷新页面进行测试

- 注意:如果控制台没有任何sql展示说明缓存已经开启

```
SpringbootJspShiroApplication
Console  Endpoints
2021-07-26 12:34:32.420  INFO 8800 --- [  restartedMain] o.a.catalina.core.AprLifecycleListener
2021-07-26 12:34:32.420  INFO 8800 --- [  restartedMain] o.a.catalina.core.AprLifecycleListener
2021-07-26 12:34:32.422  INFO 8800 --- [  restartedMain] o.a.catalina.core.AprLifecycleListener
2021-07-26 12:34:32.587  INFO 8800 --- [  restartedMain] org.apache.jasper.servlet.TldScanner
2021-07-26 12:34:32.591  INFO 8800 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/shiro]
2021-07-26 12:34:32.592  INFO 8800 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext
2021-07-26 12:34:33.076  INFO 8800 --- [  restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping
2021-07-26 12:34:33.221  INFO 8800 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer
2021-07-26 12:34:33.255  INFO 8800 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer
2021-07-26 12:34:33.263  INFO 8800 --- [  restartedMain] c.yjiewei.SpringbootJspShiroApplication
2021-07-26 12:35:26.726  INFO 8800 --- [nio-8081-exec-3] o.a.c.c.C.[Tomcat].[localhost].[/shiro]
2021-07-26 12:35:26.726  INFO 8800 --- [nio-8081-exec-3] o.s.web.servlet.DispatcherServlet
2021-07-26 12:35:26.727  INFO 8800 --- [nio-8081-exec-3] o.s.web.servlet.DispatcherServlet
2021-07-26 12:35:26.790  INFO 8800 --- [nio-8081-exec-3] com.alibaba.druid.pool.DruidDataSource
调用授权验证：admin
perms:[Perms(id=1, name=user:*:*, url=null), Perms(id=2, name=order:*:*, url=null)]
```

- 没开启缓存的效果如下

```
SpringbootJspShiroApplication
Console  Endpoints
2021-07-26 12:36:38.210  INFO 19176 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer
2021-07-26 12:36:38.220  INFO 19176 --- [  restartedMain] c.yjiewei.SpringbootJspShiroApplication
2021-07-26 12:36:43.150  INFO 19176 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/shiro]
2021-07-26 12:36:43.151  INFO 19176 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
2021-07-26 12:36:43.152  INFO 19176 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
2021-07-26 12:36:47.680  INFO 19176 --- [nio-8081-exec-6] com.alibaba.druid.pool.DruidDataSource
调用授权验证：admin
perms:[Perms(id=1, name=user:*:*, url=null), Perms(id=2, name=order:*:*, url=null)]
调用授权验证：admin
perms:[Perms(id=1, name=user:*:*, url=null), Perms(id=2, name=order:*:*, url=null)]
调用授权验证：admin
perms:[Perms(id=1, name=user:*:*, url=null), Perms(id=2, name=order:*:*, url=null)]
调用授权验证：admin
perms:[Perms(id=1, name=user:*:*, url=null), Perms(id=2, name=order:*:*, url=null)]
调用授权验证：admin
perms:[Perms(id=1, name=user:*:*, url=null), Perms(id=2, name=order:*:*, url=null)]
调用授权验证：admin
perms:[Perms(id=1, name=user:*:*, url=null), Perms(id=2, name=order:*:*, url=null)]
调用授权验证：admin
```

6.8.3 使用Redis作为缓存实现

> 相比于EhCache，Redis不用害怕突然宕机给应用带来的不便，Redis相当于存在另外一块内存中的数据，断电后依旧能够获取到之前存储的数据。

1. 引入依赖

Java

```
<!--redis整合springboot-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

2. 配置Redis连接

XML

```
spring.redis.port=6379
spring.redis.host=localhost
spring.redis.database=0
```

3. 启动redis服务

4. 开发RedisCacheManager

Java

```
//自定义shiro缓存管理器
public class RedisCacheManager implements CacheManager {

    //参数1:认证或者是授权缓存的统一名称
    @Override
    public <K, V> Cache<K, V> getCache(String cacheName) throws CacheException {
        System.out.println(cacheName);
        return new RedisCache<K,V>(cacheName);
    }
}
```

5. 开RedisCache实现

自定义redis缓存的实现

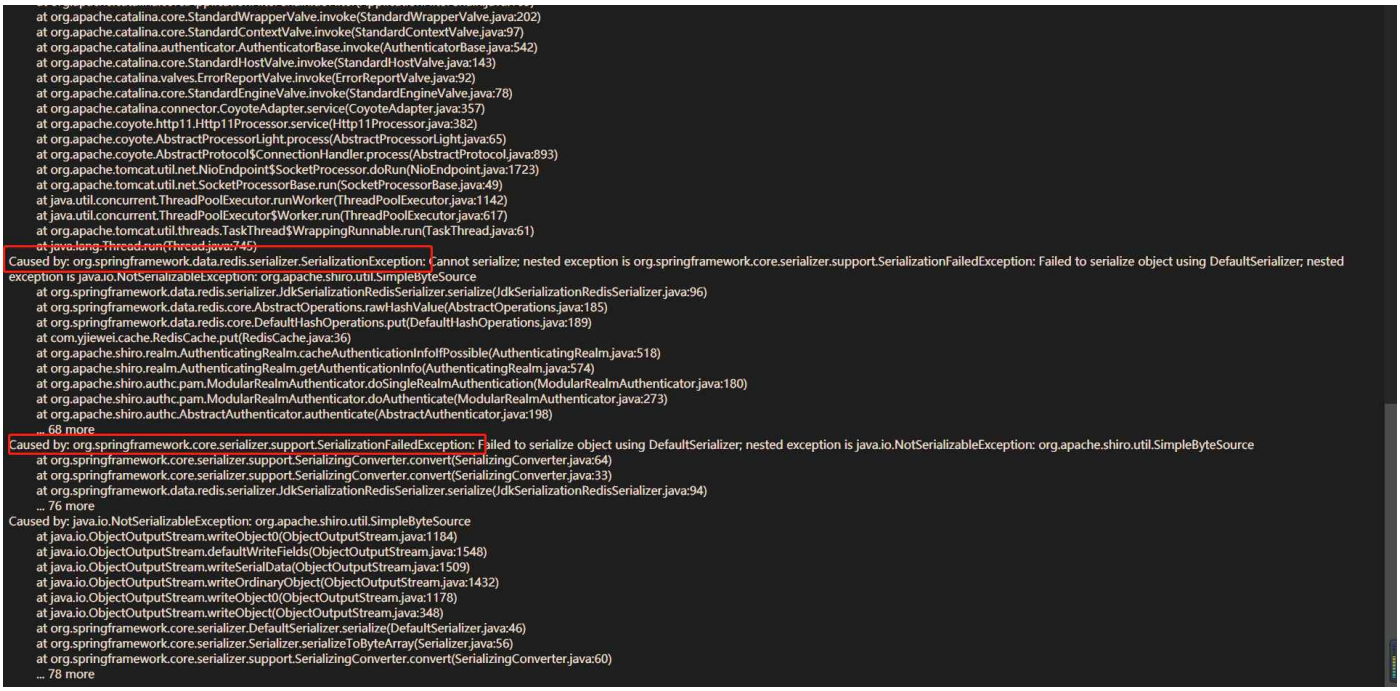
Kotlin

```
//自定义redis缓存的实现
public class RedisCache<k,v> implements Cache<k,v> {
    private String cacheName;

    public RedisCache() {
    }
```

```
7
8       public RedisCache(String cacheName) {
9           this.cacheName = cacheName;
10      }
11
12      @Override
13      public v get(k k) throws CacheException {
14          System.out.println("get key:"+k);
15          return (v)
    getRedisTemplate().opsForHash().get(this.cacheName,k.toString());
16      }
17
18      @Override
19      public v put(k k, v v) throws CacheException {
20          System.out.println("put key: "+k);
21          System.out.println("put value:"+v);
22          getRedisTemplate().opsForHash().put(this.cacheName,k.toString(),v);
23          return null;
24      }
25
26      @Override
27      public v remove(k k) throws CacheException {
28          System.out.println("=============remove==============");
29          return (v)
    getRedisTemplate().opsForHash().delete(this.cacheName,k.toString());
30      }
31
32      @Override
33      public void clear() throws CacheException {
34          System.out.println("=============clear==============");
35          getRedisTemplate().delete(this.cacheName);
36      }
37
38      @Override
39      public int size() {
40          return getRedisTemplate().opsForHash().size(this.cacheName).intValue();
41      }
42
43      @Override
44      public Set<k> keys() {
45          return getRedisTemplate().opsForHash().keys(this.cacheName);
46      }
47
48      @Override
49      public Collection<v> values() {
50          return getRedisTemplate().opsForHash().values(this.cacheName);
51      }
52
```

```
53      private RedisTemplate getRedisTemplate(){
54          RedisTemplate redisTemplate = (RedisTemplate)
    ApplicationContextUtils.getBean("redisTemplate");
55          redisTemplate.setKeySerializer(new StringRedisSerializer());
56          redisTemplate.setHashKeySerializer(new StringRedisSerializer());
57          return redisTemplate;
58      }
59   }
```

6. 启动项目测试发现报错



由于shiro中提供的simpleByteSource实现没有实现序列化,所有在认证时出现错误信息，所以我们需要自己去实现盐的序列化。

Java

```java
//自定义salt实现 实现序列化接口

public class MyByteSource implements ByteSource, Serializable {
  private  byte[] bytes;
  private String cachedHex;
  private String cachedBase64;

  public MyByteSource(){

  }

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

```

```
16    public MyByteSource(char[] chars) {
17        this.bytes = CodecSupport.toBytes(chars);
18    }
19
20    public MyByteSource(String string) {
21        this.bytes = CodecSupport.toBytes(string);
22    }
23
24    public MyByteSource(ByteSource source) {
25        this.bytes = source.getBytes();
26    }
27
28    public MyByteSource(File file) {
29        this.bytes = (new
com.lut.shiro.salt.MyByteSource.BytesHelper()).getBytes(file);
30    }
31
32    public MyByteSource(InputStream stream) {
33        this.bytes = (new
com.lut.shiro.salt.MyByteSource.BytesHelper()).getBytes(stream);
34    }
35
36    public static boolean isCompatible(Object o) {
37        return o instanceof byte[] || o instanceof char[] || o instanceof String
|| o instanceof ByteSource || o instanceof File || o instanceof InputStream;
38    }
39
40    public byte[] getBytes() {
41        return this.bytes;
42    }
43
44    public boolean isEmpty() {
45        return this.bytes == null || this.bytes.length == 0;
46    }
47
48    public String toHex() {
49        if (this.cachedHex == null) {
50            this.cachedHex = Hex.encodeToString(this.getBytes());
51        }
52
53        return this.cachedHex;
54    }
55
56    public String toBase64() {
57        if (this.cachedBase64 == null) {
58            this.cachedBase64 = Base64.encodeToString(this.getBytes());
59        }
60
```

```
61          return this.cachedBase64;
62      }
63
64      public String toString() {
65          return this.toBase64();
66      }
67
68      public int hashCode() {
69          return this.bytes != null && this.bytes.length != 0 ?
    Arrays.hashCode(this.bytes) : 0;
70      }
71
72      public boolean equals(Object o) {
73          if (o == this) {
74              return true;
75          } else if (o instanceof ByteSource) {
76              ByteSource bs = (ByteSource)o;
77              return Arrays.equals(this.getBytes(), bs.getBytes());
78          } else {
79              return false;
80          }
81      }
82
83      private static final class BytesHelper extends CodecSupport {
84          private BytesHelper() {
85          }
86
87          public byte[] getBytes(File file) {
88              return this.toBytes(file);
89          }
90
91          public byte[] getBytes(InputStream stream) {
92              return this.toBytes(stream);
93          }
94      }
95  }
```

实现不需要修改，要修改的是类的实现，实现序列化接口即可。

Java

```java
// 使用自己写好的类来获取盐并进行序列化。
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
throws AuthenticationException {
  System.out.println("==========================");
  //根据身份信息
  String principal = (String) token.getPrincipal();
  //在工厂中获取service对象
  UserService userService = (UserService)
ApplicationContextUtils.getBean("userService");
  User user = userService.findByUserName(principal);
  if(!ObjectUtils.isEmpty(user)){
    return new SimpleAuthenticationInfo(user.getUsername(),user.getPassword(),
                                        new
MyByteSource(user.getSalt()),this.getName());
  }
  return null;
}
```

7. 再次启动测试,发现可以成功放入redis缓存

Keys * 查看所有键值对。

6.8.4 加入验证码验证

1. 验证码工具类

Java

```java
public class VerifyCodeUtils{

    //使用到Algerian字体，系统里没有的话需要安装字体，字体只显示大写，去掉了1,0,i,o几个
容易混淆的字符
    public static final String VERIFY_CODES =
"23456789ABCDEFGHJKLMNPQRSTUVWXYZ";
    private static Random random = new Random();


    /**
     * 使用系统默认字符源生成验证码
     * @param verifySize    验证码长度
     * @return
     */
    public static String generateVerifyCode(int verifySize){
```

```
14          return generateVerifyCode(verifySize, VERIFY_CODES);
15      }
16      /**
17       * 使用指定源生成验证码
18       * @param verifySize    验证码长度
19       * @param sources   验证码字符源
20       * @return
21       */
22      public static String generateVerifyCode(int verifySize, String sources){
23          if(sources == null || sources.length() == 0){
24              sources = VERIFY_CODES;
25          }
26          int codesLen = sources.length();
27          Random rand = new Random(System.currentTimeMillis());
28          StringBuilder verifyCode = new StringBuilder(verifySize);
29          for(int i = 0; i < verifySize; i++){
30              verifyCode.append(sources.charAt(rand.nextInt(codesLen-1)));
31          }
32          return verifyCode.toString();
33      }
34
35      /**
36       * 生成随机验证码文件,并返回验证码值
37       * @param w
38       * @param h
39       * @param outputFile
40       * @param verifySize
41       * @return
42       * @throws IOException
43       */
44      public static String outputVerifyImage(int w, int h, File outputFile, int
    verifySize) throws IOException{
45          String verifyCode = generateVerifyCode(verifySize);
46          outputImage(w, h, outputFile, verifyCode);
47          return verifyCode;
48      }
49
50      /**
51       * 输出随机验证码图片流,并返回验证码值
52       * @param w
53       * @param h
54       * @param os
55       * @param verifySize
56       * @return
57       * @throws IOException
58       */
59      public static String outputVerifyImage(int w, int h, OutputStream os, int
    verifySize) throws IOException{
```

```
60          String verifyCode = generateVerifyCode(verifySize);
61          outputImage(w, h, os, verifyCode);
62          return verifyCode;
63      }
64
65      /**
66       * 生成指定验证码图像文件
67       * @param w
68       * @param h
69       * @param outputFile
70       * @param code
71       * @throws IOException
72       */
73      public static void outputImage(int w, int h, File outputFile, String code)
    throws IOException{
74          if(outputFile == null){
75              return;
76          }
77          File dir = outputFile.getParentFile();
78          if(!dir.exists()){
79              dir.mkdirs();
80          }
81          try{
82              outputFile.createNewFile();
83              FileOutputStream fos = new FileOutputStream(outputFile);
84              outputImage(w, h, fos, code);
85              fos.close();
86          } catch(IOException e){
87              throw e;
88          }
89      }
90
91      /**
92       * 输出指定验证码图片流
93       * @param w
94       * @param h
95       * @param os
96       * @param code
97       * @throws IOException
98       */
99      public static void outputImage(int w, int h, OutputStream os, String code)
    throws IOException{
100         int verifySize = code.length();
101         BufferedImage image = new BufferedImage(w, h,
    BufferedImage.TYPE_INT_RGB);
102         Random rand = new Random();
103         Graphics2D g2 = image.createGraphics();
104
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALI
```

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_ON);
        Color[] colors = new Color[5];
        Color[] colorSpaces = new Color[] { Color.WHITE, Color.CYAN,
                Color.GRAY, Color.LIGHT_GRAY, Color.MAGENTA, Color.ORANGE,
                Color.PINK, Color.YELLOW };
        float[] fractions = new float[colors.length];
        for(int i = 0; i < colors.length; i++){
            colors[i] = colorSpaces[rand.nextInt(colorSpaces.length)];
            fractions[i] = rand.nextFloat();
        }
        Arrays.sort(fractions);

        g2.setColor(Color.GRAY);// 设置边框色
        g2.fillRect(0, 0, w, h);

        Color c = getRandColor(200, 250);
        g2.setColor(c);// 设置背景色
        g2.fillRect(0, 2, w, h-4);

        //绘制干扰线
        Random random = new Random();
        g2.setColor(getRandColor(160, 200));// 设置线条的颜色
        for (int i = 0; i < 20; i++) {
            int x = random.nextInt(w - 1);
            int y = random.nextInt(h - 1);
            int xl = random.nextInt(6) + 1;
            int yl = random.nextInt(12) + 1;
            g2.drawLine(x, y, x + xl + 40, y + yl + 20);
        }

        // 添加噪点
        float yawpRate = 0.05f;// 噪声率
        int area = (int) (yawpRate * w * h);
        for (int i = 0; i < area; i++) {
            int x = random.nextInt(w);
            int y = random.nextInt(h);
            int rgb = getRandomIntColor();
            image.setRGB(x, y, rgb);
        }

        shear(g2, w, h, c);// 使图片扭曲

        g2.setColor(getRandColor(100, 160));
        int fontSize = h-4;
        Font font = new Font("Algerian", Font.ITALIC, fontSize);
        g2.setFont(font);
        char[] chars = code.toCharArray();
        for(int i = 0; i < verifySize; i++){
```

```
            AffineTransform affine = new AffineTransform();
            affine.setToRotation(Math.PI / 4 * rand.nextDouble() *
(rand.nextBoolean() ? 1 : -1), (w / verifySize) * i + fontSize/2, h/2);
            g2.setTransform(affine);
            g2.drawChars(chars, i, 1, ((w-10) / verifySize) * i + 5, h/2 +
fontSize/2 - 10);
        }

        g2.dispose();
        ImageIO.write(image, "jpg", os);
    }

    private static Color getRandColor(int fc, int bc) {
        if (fc > 255)
            fc = 255;
        if (bc > 255)
            bc = 255;
        int r = fc + random.nextInt(bc - fc);
        int g = fc + random.nextInt(bc - fc);
        int b = fc + random.nextInt(bc - fc);
        return new Color(r, g, b);
    }

    private static int getRandomIntColor() {
        int[] rgb = getRandomRgb();
        int color = 0;
        for (int c : rgb) {
            color = color << 8;
            color = color | c;
        }
        return color;
    }

    private static int[] getRandomRgb() {
        int[] rgb = new int[3];
        for (int i = 0; i < 3; i++) {
            rgb[i] = random.nextInt(255);
        }
        return rgb;
    }

    private static void shear(Graphics g, int w1, int h1, Color color) {
        shearX(g, w1, h1, color);
        shearY(g, w1, h1, color);
    }

    private static void shearX(Graphics g, int w1, int h1, Color color) {

```

```
198         int period = random.nextInt(2);
199 
200         boolean borderGap = true;
201         int frames = 1;
202         int phase = random.nextInt(2);
203 
204         for (int i = 0; i < h1; i++) {
205             double d = (double) (period >> 1)
206                     * Math.sin((double) i / (double) period
207                     + (6.2831853071795862D * (double) phase)
208                     / (double) frames);
209             g.copyArea(0, i, w1, 1, (int) d, 0);
210             if (borderGap) {
211                 g.setColor(color);
212                 g.drawLine((int) d, i, 0, i);
213                 g.drawLine((int) d + w1, i, w1, i);
214             }
215         }
216 
217     }
218 
219     private static void shearY(Graphics g, int w1, int h1, Color color) {
220 
221         int period = random.nextInt(40) + 10; // 50;
222 
223         boolean borderGap = true;
224         int frames = 20;
225         int phase = 7;
226         for (int i = 0; i < w1; i++) {
227             double d = (double) (period >> 1)
228                     * Math.sin((double) i / (double) period
229                     + (6.2831853071795862D * (double) phase)
230                     / (double) frames);
231             g.copyArea(i, 0, 1, h1, 0, (int) d);
232             if (borderGap) {
233                 g.setColor(color);
234                 g.drawLine(i, (int) d, i, 0);
235                 g.drawLine(i, (int) d + h1, i, h1);
236             }
237 
238         }
239 
240     }
241     public static void main(String[] args) throws IOException {
242         //获取验证码
243         String s = generateVerifyCode(4);
244         //将验证码放入图片中
245         outputImage(260,60,new File("aa.jpg"),s);
```

```
246             System.out.println(s);
247         }
248     }
```

2. 开发页面加入验证码

Java

```
1  <form action="${pageContext.request.contextPath}/user/login" method="post">
2      用户名:<input type="text" name="username" > <br/>
3      密码  : <input type="text" name="password"> <br>
4      请输入验证码: <input type="text" name="code"><img
   src="${pageContext.request.contextPath}/user/getImage" alt=""><br>
5      <input type="submit" value="登录">
6  </form>
```

3.开发控制器

Java

```
1  @RequestMapping("getImage")
2  public void getImage(HttpSession session, HttpServletResponse response) throws
   IOException {
3    //生成验证码
4    String code = VerifyCodeUtils.generateVerifyCode(4);
5    //验证码放入session
6    session.setAttribute("code",code);
7    //验证码存入图片
8    ServletOutputStream os = response.getOutputStream();
9    response.setContentType("image/png");
10   VerifyCodeUtils.outputImage(220,60,os,code);
11 }
```

4.放行验证码

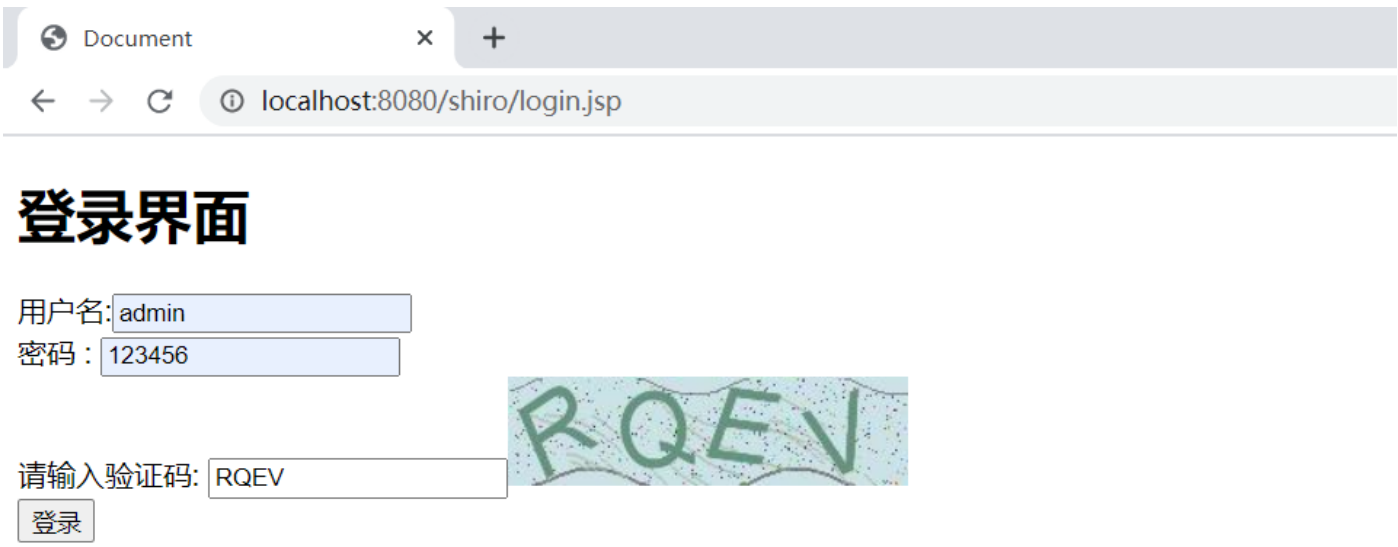
`` `map.put("/user/getImage","anon");//验证码 ` ``

5.修改认证流程

TypeScript

```
@RequestMapping("login")
public String login(String username, String password,String code,HttpSession session) {
    //比较验证码
    String codes = (String) session.getAttribute("code");
    try {
        if (codes.equalsIgnoreCase(code)){
            //获取主体对象
            Subject subject = SecurityUtils.getSubject();
                subject.login(new UsernamePasswordToken(username, password));
                return "redirect:/index.jsp";
        }else{
            throw new RuntimeException("验证码错误!");
        }
    } catch (UnknownAccountException e) {
        e.printStackTrace();
        System.out.println("用户名错误!");
    } catch (IncorrectCredentialsException e) {
        e.printStackTrace();
        System.out.println("密码错误!");
    }catch (Exception e){
        e.printStackTrace();
        System.out.println(e.getMessage());
    }
    return "redirect:/login.jsp";
}
```

6. 启动测试



6.8.5 JSP中Shiro常用标签

HTML

```
<shiro:guest>
    游客访问 <a href = "login.jsp"></a>
</shiro:guest>

user 标签：用户已经通过认证\记住我 登录后显示响应的内容
<shiro:user>
    欢迎[<shiro:principal/>]登录 <a href = "logout">退出</a>
</shiro:user>

authenticated标签：用户身份验证通过，即 Subjec.login 登录成功 不是记住我登录的
<shiro:authenticted>
    用户[<shiro:principal/>] 已身份验证通过
</shiro:authenticted>

notAuthenticated标签：用户未进行身份验证，即没有调用Subject.login进行登录,包括"记住我"也属于未进行身份验证
<shiro:notAuthenticated>
    未身份验证(包括"记住我")
</shiro:notAuthenticated>


principal 标签：显示用户身份信息，默认调用
Subjec.getPrincipal()获取，即Primary Principal
<shiro:principal property = "username"/>

hasRole标签：如果当前Subject有角色将显示body体内的内容
<shiro:hashRole name = "admin">
    用户[<shiro:principal/>]拥有角色admin
</shiro:hashRole>

hasAnyRoles标签：如果Subject有任意一个角色(或的关系)将显示body体里的内容
<shiro:hasAnyRoles name = "admin,user">
    用户[<shiro:pricipal/>]拥有角色admin 或者 user
</shiro:hasAnyRoles>

lacksRole:如果当前 Subjec没有角色将显示body体内的内容
<shiro:lacksRole name = "admin">
    用户[<shiro:pricipal/>]没有角色admin
</shiro:lacksRole>

hashPermission:如果当前Subject有权限将显示body体内容
<shiro:hashPermission name = "user:create">
    用户[<shiro:pricipal/>] 拥有权限user:create
</shiro:hashPermission>
```

```
44
45 lacksPermission:如果当前Subject没有权限将显示body体内容
46 <shiro:lacksPermission name = "org:create">
47     用户[<shiro:pricipal/>] 没有权限org:create
48 </shiro:lacksPermission>
```

七、shiro 整合thymeleaf 权限控制

代码示例

7.1 引入依赖

Java

```
1  <!--引入thymeleaf模板引擎，扎样才能在html页面中使用shiro标签-->
2  <dependency>
3      <groupId>com.github.theborakompanioni</groupId>
4      <artifactId>thymeleaf-extras-shiro</artifactId>
5      <version>2.0.0</version>
6  </dependency>
7
8  <!--thymeleaf整合到springboot中不可少的-->
9  <dependency>
10     <groupId>org.springframework.boot</groupId>
11     <artifactId>spring-boot-starter-thymeleaf</artifactId>
12 </dependency>
```

7.2 修改页面

login.html

HTML

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>用户登录</h1>

    <form th:action="@{/user/login}" method="post">
        用户名：<input type="text" name="username" > <br>
        密码：<input type="password" name="password"> <br>
        请输入验证码：<input type="text" name="code"><img th:src="@{/user/getImage}" alt=""><br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

7.3 视图控制器

jsp页面改成html页面

Java

```java
@Controller
@RequestMapping("user")
public class UserController {

    @Resource
    private UserService userService;

    @RequestMapping("loginView")
    public String loginView(){
        System.out.println("loginView页面跳转");
        return "login";
    }

    @RequestMapping("registerView")
    public String registerView(){
        System.out.println("registerView页面跳转");
        return "register";
    }

    /**
```

```
21      * 用户身份验证
22      * @param username
23      * @param password
24      * @return
25      */
26     @RequestMapping("login")
27     public String login(String username, String password, String code,
    HttpSession session){
28         // 1.获取主体对象
29         Subject subject = SecurityUtils.getSubject();
30         //比较验证码
31         String codes = (String) session.getAttribute("code");
32         try{
33             if (codes.equals(code)) {
34                 subject.login(new UsernamePasswordToken(username, password));
35                 return "index";
36             }else {
37                 throw new RuntimeException("验证码错误");
38             }
39         }catch (UnknownAccountException e){
40             System.out.println("用户名错误");
41         }catch (IncorrectCredentialsException e){
42             System.out.println("密码错误");
43         }
44         return "login";
45     }
46
47     /**
48      * 退出登录
49      * @return 重定向到登录界面
50      */
51     @RequestMapping("/logout")
52     public String logout(){
53         Subject subject = SecurityUtils.getSubject();
54         subject.logout();
55         return "login";
56     }
57
58     /**
59      * 用户注册
60      * @param user
61      * @return
62      */
63     @RequestMapping("register")
64     public String register(User user){
65         try{
66             userService.register(user);
67             return "login";
```

```
68          }catch (Exception e) {
69              e.printStackTrace();
70              return "register";
71          }
72      }
73
74      /**
75       * 模拟保存方法
76       * 实际上知识为了测试页面访问URL时是否有权限访问
77       * @return
78       */
79      @RequestMapping("save")
80      public String save(){
81          System.out.println("进入方法");
82
83          //1.基于角色
84          //获取主体对象
```