**1、本章面试题**

     什么是shiro，shiro有什么作用，web项目中如何使用

项目中中的安全问题是怎么解决的？用户的认证和授权是如何处理？

**2、知识点**

**2.1、课程回顾**

shiro 是功能强大，使用简单的java安全框架

认证，授权 ，加密，会话管理

核心概念： subject (用户)，SecurityManager(幕后主使)，Realm(连接器)

**2.2、本章重点**

什么是shiro,为什么要使用

核心概念（组件）

简单入门

自定义realm用法（加密，加盐）

jdbc realm用法（使用数据库）

SSM(spring springmvc mybatis)整合shiro

SM(springboot mybatis) 整合shiro

**3、具体内容**

**3.1 什么是shiro**

[**http://shiro.apache.org/**](http://shiro.apache.org/)

**Apache Shiro™**是一个功能强大且易于使用的Java安全框架，用于执行身份验证，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。

**3.2 为什么要使用shiro(和它同级spring security)**

**易于使用** -易于使用是该项目的最终目标。应用程序安全性可能非常令人困惑和沮丧，并被视为“必要的邪恶”。如果您使它易于使用，以使新手程序员可以开始使用它，那么就不必再痛苦了。

**全面** -Apache Shiro声称没有其他具有范围广度的安全框架，因此它很可能是满足安全需求的“一站式服务”。

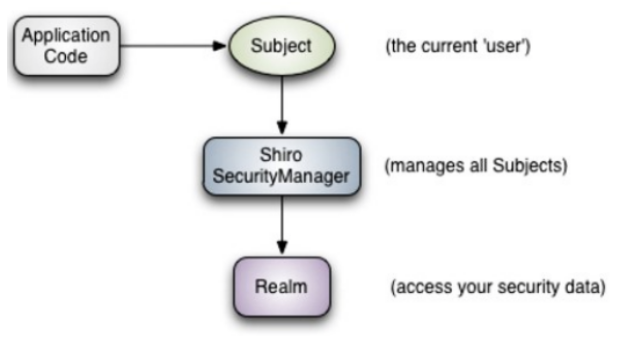
**灵活** -Apache Shiro可以在任何应用程序环境中工作。尽管它可以在Web，EJB和IoC环境中运行，但并不需要它们。Shiro也不要求任何规范，甚至没有很多依赖性。

**具有Web功能 -**Apache Shiro具有出色的Web应用程序支持，允许您基于应用程序URL和Web协议（例如REST）创建灵活的安全策略，同时还提供一组JSP库来控制页面输出。

**可插拔** -Shiro干净的API和设计模式使它易于与许多其他框架和应用程序集成。您会看到Shiro与Spring，Grails，Wicket，Tapestry，Mule，Apache Camel，Vaadin等框架无缝集成。

**支持** -Apache Shiro是Apache Software Foundation（Apache软件基金会）的一部分，该组织被证明以其社区的最大利益行事。项目开发和用户群体友好的公民随时可以提供帮助。如果需要，像Katasoft这样的商业公司也可以提供专业的支持和服务。

**3.3 核心概念**



**Subject**

Subject一词是一个安全术语，基本上表示“当前正在执行的用户”。它只是不被称为“用户”，因为“用户”一词通常与人类相关联。在安全的世界，术语“主题”可以指一个人，但也有会谈进程，守护进程帐户，或任何类似。

import org.apache.shiro.subject.Subject;
import org.apache.shiro.SecurityUtils;
Subject currentUser = SecurityUtils.getSubject();

**SecurityManager**

主题的“幕后”对应对象是SecurityManager。主题代表当前用户的安全操作，而SecurityManager管理*所有*用户的安全操作。它是Shiro体系结构的核心，并充当一种“伞”对象，引用了许多内部嵌套的安全组件，这些安全组件构成了一个对象图。每个应用程序几乎总是有一个SecurityManager实例。它本质上是一个应用程序单例（尽管不必是*静态*单例）普通的Java代码，Spring XML，YAML，.properties和.ini文件等进行配置

ini文件方式，实例化SecurityManager 配置：

[main]

#CredentialsMatcher 认证匹配器

cm = org.apache.shiro.authc.credential.HashedCredentialsMatcher

cm.hashAlgorithm = SHA-512

cm.hashIterations = 1024

# Base64 encoding (less text):

cm.storedCredentialsHexEncoded = false

[users]

jdoe = TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJpcyByZWFzb2

asmith = IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbXNoZWQsIG5vdCB

[roles]

import org.apache.shiro.SecurityUtils;

import org.apache.shiro.config.IniSecurityManagerFactory;

import org.apache.shiro.mgt.SecurityManager;

import org.apache.shiro.util.Factory;

// 1。加载INI配置

Factory <SecurityManager> factory =

new IniSecurityManagerFactory（“ classpath：shiro.ini”）;

// 2。创建SecurityManager

SecurityManager securityManager = factory.getInstance（）;

// 3。使它可访问

SecurityUtils.setSecurityManager（securityManager）;

**Realms**

Realms充当Shiro与应用程序的安全数据之间的“桥梁”或“连接器”,也就是说，当需要真正与安全性相关的数据（例如用户帐户）进行交互以执行身份验证（登录）和授权（访问控制）时，Shiro会从为应用程序配置的一个或多个Realms中查找其中的许多内容。本质上是特定于安全性的DAO：它封装了数据源的连接详细信息，并根据需要使关联数据可用于Shiro。

#数据源

dataSource=org.apache.commons.dbcp.BasicDataSource

#数据源链接要素

dataSource.driverClassName=oracle.jdbc.driver.OracleDriver

dataSource.url=jdbc:oracle:thin:@localhost:1521:orcl

dataSource.username=scott

dataSource.password=tiger

#jdbcrealm

jdbcrealm=org.apache.shiro.realm.jdbc.JdbcRealm

#jdbcrealm需要用到的数据源

jdbcrealm.dataSource=$dataSource

#开启查找权限

jdbcrealm.permissionsLookupEnabled=true

#设置安全管理器使用的jdbcrealm

securityManager.realms=$jdbcrealm

**3.4 认证和授权**

**认证(Authentication)：**

身份验证是验证用户身份的过程。也就是说，当用户通过应用程序进行身份验证时，他们在证明自己实际上就是他们所说的身份。有时也称为“登录”。这通常是一个三步过程。

收集用户的标识信息（称为*主体 用户名*）和支持身份的*凭证*（称为*凭据 密码）*。

AuthenticationToken token =

new UsernamePasswordToken(username, password);

将主体和凭据提交到系统。

Subject currentUser = SecurityUtils.getSubject();

currentUser.login(token);

如果提交的凭据与系统对该用户身份的期望匹配，则认为该用户已通过身份验证。如果它们不匹配，则不认为用户已通过身份验证。

try {

currentUser.login(token);

//通过身份验证，执行其他业务

} catch (IncorrectCredentialsException ice) {

//密码错误

} catch (LockedAccountException lae) {

//账户锁定（默认系统不提供，可以编写代码过程中自己实现）

}

… //许多其他异常

catch (AuthenticationException ae) {

//上面配置的异常的父类

}

**授权（authorisation）**

授权本质上是访问控制-控制用户可以在应用程序中访问的内容（例如资源，网页等）。大多数用户通过使用角色和权限等概念来执行访问控制。也就是说，通常根据分配给他们的角色和/或权限，允许用户执行某项操作或不执行某项操作。

判断角色：

if ( subject.hasRole(“administrator”) ) {

//show the ‘Create User’ button

} else {

//grey-out the button?

}

判断权限：

if ( subject.isPermitted(“user:create”) ) {

//show the ‘Create User’ button

} else {

//grey-out the button?

}

**3.5 入门示例**

**1, 创建项目，引入jar**

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.5.1</version>

</dependency>

<!-- commons-logging 日志包-->

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.0.4</version>

</dependency>

**2, 创建ini文件（shiro初始化文件），复制下面内容：**

#用户对象

[users]

#用户名密码

scott=tiger,role1

smith=tiger

#角色

[roles]

#角色role1对用户有create、update权限

role1=user:create,user:update

#角色role2对用户有create、delete权限

role2=user:create,user:delete

#角色role3对用户有create权限

role3=user:create

**3，创建java类,加载shiro.ini 实例化SecurityManager 获取subject ,进行认证和授权的操作**

// 加载配置文件，创建工厂类

Factory<SecurityManager> factory =

new IniSecurityManagerFactory("classpath:shiro.ini");

//使用工厂类获取SecurityManager对象

SecurityManager securityManager = factory.getInstance();

// 使SecurityManager 使它可访问

SecurityUtils.setSecurityManager(securityManager);

// 获取当前执行的用户

Subject currentUser = SecurityUtils.getSubject();

// 获取session

Session session = currentUser.getSession();

//使用session 存储任何信息

session.setAttribute("userName","zhangsan");

//判断用户是否被认证过

if(!currentUser.isAuthenticated()){

//收集用户的标识信息（称为主体 用户名）和支持身份的凭证（称为凭据 密码）

UsernamePasswordToken token =

new UsernamePasswordToken("scott","tiger");

// 记住密码

token.setRememberMe(true);

// 将主体和凭据提交到系统。如果提交的凭据与系统对该用户身份的期望匹配，则认为该用户已通过身份验证。如果它们不匹配，则不认为用户已通过身份验证

try {

currentUser.login(token);

System.out.println("认证成功！");

//授权

//判断角色

System.out.println("测试scott是否有用role1角色："+currentUser.hasRole("role1"));//true

System.out.println("测试scott是否有用role2角色："+currentUser.hasRole("role2"));//false

// hasAllRoles拥有为true

System.out.println("测试scott是否有用role1,role2角色："+

currentUser.hasAllRoles(Arrays.asList("role1","role2")));//false

//判断权限

System.out.println("测试scott是否拥有user:create权限:"+

currentUser.isPermitted("user:create"));//true

System.out.println("测试scott是否拥有dept:del权限:"+

currentUser.isPermitted("dept:del"));//true

System.out.println("测试scott是否拥有dept:update权限:"+

currentUser.isPermitted("dept:update"));

System.out.println("测试scott是否拥有多个权限1:"+

currentUser.isPermittedAll("user:create","user:update"));//true

System.out.println("测试scott是否拥有多个权限2:"+

currentUser.isPermittedAll("dept:update","dept:create"));//false

} catch (AccountException e){

System.out.println("用户名错误！");

}catch (IncorrectCredentialsException e){

System.out.println("密码错误！");

}

catch (AuthenticationException e) {

System.out.println("用户名或者密码错误");

}

}

//退出

currentUser.logout();

**3.6 自定义realm用法 （加密，加盐）**

**1，创建自定义的ini文件 shiro\_custom\_realm.ini 复制下面内容**

[main]

#引入加密类 cm CredentialsMatcher 认证匹配器

cm = org.apache.shiro.authc.credential.HashedCredentialsMatcher

#规定加密方式 hash哈希 AlgorithmName 算法名称

cm.hashAlgorithmName = SHA-512

#哈希次数

cm.hashIterations = 10

# Base64 encoding (less text):如果credential 哈希是Hex 编码的话为true，base64的话为false

#cm.storedCredentialsHexEncoded = false

#配置自定义realm

myRealm=com.aaa.shiro.util.MyCustomRealm

# $cm 引用上面cm配置

myRealm.credentialsMatcher = $cm

#把安全数据交给SecurityManager认证或者授权

securityManager.realms=$myRealm

**2，编写自定义MyRealm类继承AuthorizingRealm类，重写认证和授权方法**

//需要重写的授权方法

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

return null;

}

//需要重写的认证方法

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

// 获取已经收集用户的标识信息中的用户名

Object userName = authenticationToken.getPrincipal();

//模拟调用编写的服务类获取用户信息

// UserInfo userInfo = userService.getUserInfo(userName);

String userNameByDB = "scott";

//判断是否匹配

if(!userNameByDB.equals(userName)){

//账号不配置，直接抛出账户异常

throw new AccountException();

}

//加密过的密码

//tiger 经过加盐（123456）后10次hash结果

String myDBPassWord ="5cf729699270e380bfc886aedc3fe8f2acf13751c938ac2e46d281510eedf77242b6cd8e5013d505b072fa63bcaa3b7bcd593fce1e6d5e5a5487a0cee33b5fd2";

//盐 从数据库中取出的

String myDBSalt="1234567";

//实例化返回值

//SimpleAuthenticationInfo(Object principal, Object hashedCredentials, ByteSource credentialsSalt, String realmName)

// 参数1 principal 从数据库中取出的用户信息userInfo

// 参数2 hashedCredentials 加密过的密码

// 参数3 credentialsSalt 盐值 为了提高密码安全度

//参数4 realmName 获取当前操作类的名称MyCustomRealm

AuthenticationInfo authenticationInfo =

new SimpleAuthenticationInfo(userNameByDB,myDBPassWord, ByteSource.Util.bytes(myDBSalt),getName());

return authenticationInfo;

}

**3，编写测试类加载 shiro\_custom\_realm.ini调用MyCustomRealm,完成加密加盐的认证**

//加载配置文件，生成工厂类

Factory<SecurityManager> factory =

new IniSecurityManagerFactory("classpath:shiro\_custom\_realm.ini");

// 使用工厂获取SecurityManager

SecurityManager securityManager = factory.getInstance();

// 使SecurityManager可以访问

SecurityUtils.setSecurityManager(securityManager);

//获取Subject(当前用户)

Subject currentUser = SecurityUtils.getSubject();

//判断是否已经认证

if(!currentUser.isAuthenticated()) {

//收集用户用户名和密码数据 多态

AuthenticationToken authenticationToken =

new UsernamePasswordToken("scott","tiger");

//提交收集到信息，让securityManager进行认证

try {

currentUser.login(authenticationToken);

System.out.println("登录成功");

} catch (AccountException e){

System.out.println("用户名错误");

}catch (IncorrectCredentialsException e){

System.out.println("密码错误");

} catch (AuthenticationException e) {

System.out.println("用户名或者密码错误！");

}

}

//登出

currentUser.logout();

**模拟注册时生成密码的工具类：**

//Sha512Hash(Object source, Object salt, int hashIterations)

//参数1 原始密码

//参数2 盐值

//参数3 hash次数

String registerSalt="123456";

Sha512Hash sha512Hash= new Sha512Hash("tiger",registerSalt,

10);

// 打印出 tiger 经过加盐（123456）后10次hash结果

System.out.println(sha512Hash.toString());

// 5cf729699270e380bfc886aedc3fe8f2acf13751c938ac2e46d281510eedf77242b6cd8e5013d505b072fa63bcaa3b7bcd593fce1e6d5e5a5487a0cee33b5fd2

//存入数据库

**3.7 jdbcRealm的用法**

**1，新加，引入连接池(dbcp,druid)和数据库驱动包（oracle和mysql）**

[**http://mvnrepository.com/**](http://mvnrepository.com/)

<!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.alibaba/druid -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.21</version>

</dependency>

<!-- https://mvnrepository.com/artifact/oracle/oracle-jdbc -->

<dependency>

<groupId>oracle</groupId>

<artifactId>oracle-jdbc</artifactId>

<version>12.1.0.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.48</version>

</dependency>

**2 ，查看org.apache.shiro.realm.jdbc.JdbcRealm的源码，创建数据库**

**oracle:**

-- 用户表users

create table users(

username varchar2(20),

password varchar2(50),

age number(3)

);

-- 添加数据

insert into users values('scott','tiger',10);

insert into users values('admin','tiger',20);

insert into users values('smith','tiger',30);

-- 用户角色关联表 user\_roles

create table user\_roles(

role\_name varchar2(20),

username varchar2(20)

);

-- 添加数据

insert into user\_roles values('role1','scott');

insert into user\_roles values('role2','scott');

insert into user\_roles values('role2','admin');

insert into user\_roles values('role3','smith');

-- 角色权限关联表 roles\_permissions

create table roles\_permissions(

role\_name varchar2(20),

permission varchar2(100)

);

-- 添加数据

insert into roles\_permissions values('role1','dept:select');

insert into roles\_permissions values('role1','dept:update');

insert into roles\_permissions values('role1','dept:delete');

insert into roles\_permissions values('role2','user:select');

insert into roles\_permissions values('role2','user:update');

insert into roles\_permissions values('role3','user:delete');

-- 检查

select \* from users;

select \* from user\_roles;

select \* from roles\_permissions;

**mysql:**

-- 用户表users

create table users(

username varchar(20),

password varchar(50),

age int(3)

);

-- 添加数据

insert into users values('scott','tiger',10);

insert into users values('admin','tiger',20);

insert into users values('smith','tiger',30);

-- 用户角色关联表 user\_roles

create table user\_roles(

role\_name varchar(20),

username varchar(20)

);

-- 添加数据

insert into user\_roles values('role1','scott');

insert into user\_roles values('role2','scott');

insert into user\_roles values('role2','admin');

insert into user\_roles values('role3','smith');

-- 角色权限关联表 roles\_permissions

create table roles\_permissions(

role\_name varchar(20),

permission varchar(100)

);

-- 添加数据

insert into roles\_permissions values('role1','dept:select');

insert into roles\_permissions values('role1','dept:update');

insert into roles\_permissions values('role1','dept:delete');

insert into roles\_permissions values('role2','user:select');

insert into roles\_permissions values('role2','user:update');

insert into roles\_permissions values('role3','user:delete');

-- 检查

select \* from users;

select \* from user\_roles;

select \* from roles\_permissions;

**3 ，编写shiro\_jdbcrealm\_oracle.ini shiro\_jdbcrealm\_mysql.ini**

**shiro\_jdbcrealm\_oracle.ini (oracle+dbcp)**

#数据源

dataSource=org.apache.commons.dbcp.BasicDataSource

#数据源链接要素

dataSource.driverClassName=oracle.jdbc.driver.OracleDriver

#mysql 3306 sqlserver 1433

dataSource.url=jdbc:oracle:thin:@localhost:1521:orcl

dataSource.username=scott

dataSource.password=tiger

#jdbcrealm

jdbcrealm=org.apache.shiro.realm.jdbc.JdbcRealm

#jdbcrealm需要用到的数据源

jdbcrealm.dataSource=$dataSource

#开启查找权限

jdbcrealm.permissionsLookupEnabled=true

#设置安全管理器使用的jdbcrealm

securityManager.realms=$jdbcrealm

**shiro\_jdbcrealm\_mysql.ini (mysql+druid)**

#数据源

dataSource=com.alibaba.druid.pool.DruidDataSource

#数据源链接要素

dataSource.driverClassName=com.mysql.jdbc.Driver

#mysql 3306 sqlserver 1433

dataSource.url=jdbc:mysql://localhost:3306/szqy07

dataSource.username=root

dataSource.password=root

#jdbcrealm

jdbcrealm=org.apache.shiro.realm.jdbc.JdbcRealm

#jdbcrealm需要用到的数据源

jdbcrealm.dataSource=$dataSource

#开启查找权限

jdbcrealm.permissionsLookupEnabled=true

#设置安全管理器使用的jdbcrealm

securityManager.realms=$jdbcrealm

**4，编写测试类ShiroTestC ShiroTestD**

// 加载ini创建factory shiro\_jdbcrealm\_mysql.ini

Factory<SecurityManager> factory =

new IniSecurityManagerFactory("classpath:shiro\_jdbcrealm\_oracle.ini");

// 工厂类获取SecurityManager

SecurityManager securityManager = factory.getInstance();

//使他可以访问

SecurityUtils.setSecurityManager(securityManager);

//获取当前Subject

Subject currentUser = SecurityUtils.getSubject();

//判断是否认证

if(!currentUser.isAuthenticated()){

//收集客户端提交的用户和密码

AuthenticationToken token =

new UsernamePasswordToken("scott","tiger");

//new UsernamePasswordToken("smith","tiger");

//提交收集的数据给SecurityManager ,做认证处理 this.securityManager.login(this, token);

//org.apache.shiro.authc.Authenticator authenticate方法，认证

try {

currentUser.login(token);

System.out.println("认证成功");

//授权

//角色

System.out.println("scott用户是否具有role1角色："+currentUser.hasRole("role1"));//true

System.out.println("scott用户是否具有role3角色："+currentUser.hasRole("role3"));//false

//权限

System.out.println("scott用户是否具有dept:select权限："+currentUser.isPermitted("dept:select"));//true

System.out.println("scott用户是否具有dept:select权限："+currentUser.isPermitted("dept:alter"));//false

} catch (AccountException e){

System.out.println("用户名错误");

}catch (IncorrectCredentialsException e){

System.out.println("密码错误");

}catch (AuthenticationException e) {

e.printStackTrace();

}

}

//当前用户退出

currentUser.logout();

**4、本章总结**

**4.1 总结本章知识点**

shiro 认证流程：

1）收集认证信息（用户和密码）UsernamePasswordToken,subject调用login方法将信息提交SecurityManager

2) SecurityManager将认证信息交给org.apache.shiro.authc.Authenticator

3）realm 访问数据源，获取安全数据，并且做封装（AuthenticationInfo）

4） SecurityManager获取realm从数据库（或者其他数据源）拿到的安全数据

5)org.apache.shiro.authc.Authenticator调用Authenticate方法，进行验证

授权流程：

org.apache.shiro.authz.Authorizer

**4.2 面试题答案**

**4.3 预习下一章重点**