A、快速开始

wangjie edited this page on 11 Jan · [5 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/A%E3%80%81%E5%BF%AB%E9%80%9F%E5%BC%80%E5%A7%8B/_history)

**pom.xml中添加：**

<dependency>

<groupId>org.jsets</groupId>

<artifactId>jsets-shiro-spring-boot-starter</artifactId>

<version>0.0.1</version>

</dependency>

由于还没有maven坐标您可以下载源码自行编译或者直接下载jar包然后上传至本地仓库。

[**jsets-shiro-spring-boot-starter-1.0.0.jar**](https://github.com/wj596/packages/blob/master/jsets-shiro-spring-boot-starter-1.0.0.jar?_blank)

**application.properties中添加：**

###jsets-shiro配置

#登陆页面

jsets.shiro.login-url=/login

#登陆成功页面

jsets.shiro.login-success-url=/index

#忽略拦截的URL,静态资源

jsets.shiro.filte-rules[0]=/assets/\*\*-->anon

#登陆

jsets.shiro.filte-rules[1]=/login/\*\*-->authc

#退出

jsets.shiro.filte-rules[2]=/logout/\*\*-->logout

#所有路径，断言session中存在用户

jsets.shiro.filte-rules[3]=/\*\*-->user

**创建Controller：**

@Controller

public class IndexAction {

@RequestMapping("/")

public String def() {

return "index";

}

@RequestMapping("/index")

public String index() {

return "index";

}

@RequestMapping("/login")

public String login() {

return "login";

}

}

**创建登陆页面：**

<form class="login-form" action="${ctx}/login" method="post">

<div class="form-item">

用户名：<input type="text" name="username" />

</div>

<div class="form-item">

密码：<input type="password" name="password" />

</div>

<div class="form-item">

<button type="submit">登录</button>

</div>

</form>

<!-- 认证信息 -->

<font color="red">${Request["shiro\_auth\_message"]!}</font>

**创建主页index.html：**

<div class="body">

欢迎您：${Session.shiro\_current\_user.account!}

<br>

退出：<a href="${ctx}/logout">退了</a>

</div>

**Application启动类中添加@EnableJsetsShiro注解：**

@SpringBootApplication

@EnableJsetsShiro

public class Application{

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

启动springboot应用，控制台会打印出组件预置的体验账号和密码：

使用这个账号和密码就可以登陆系统了。

B、ajax响应

wj596 edited this page on 1 Jan · [1 revision](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/B%E3%80%81ajax%E5%93%8D%E5%BA%94/_history)

**401 未登陆**

主要是针对session超时或者未登陆就直接通过浏览器地址访问受保护资源(URL)。

1、普通请求(from提交、url跳转)

处理方式：重定向到登陆页面

2、AJAX请求

处理方式：响应401 HTTP状态码， 返回JSON信息{code:'auth:unauthorized',message:'需要身份验证'}

示例代码：

$.ajax({

url : 请求地址,

type : "POST",

data : 提交的数据,

dataType : 'json',

contentType : 'application/json;charset=UTF-8',

success : 成功响应处理函数,

error : function(xhr, textStatus, errorThrown) {

if (xhr.status == 401) { // 处理未登陆情况

alert(xhr.responseText)

} else if (xhr.status == 403) { // 处理未授权情况

alert(xhr.responseText)

} else { // 处理其他未知情况

alert(xhr.responseText)

}

}

});

如果session超时展示效果：

实际项目中您可以做的更人性化点，弹出一个登陆窗口让用户登陆。假如用户正在填写一个超级复杂的表单，提交的时候刚好session超时，这时候要是跳转到登陆界面，估计用户会崩溃的，这种处理方法让用户不丢弃正在进行的工作。

**403 未授权**

主要是针对用户点击或者提交了一个自己没有权限操作的资源(URL)，比如不具备相应的角色。

1、普通请求(from提交、url跳转)

处理方式：重定向到无权限页面，这个页面是在您的application.properties中配置的

# 无权限页面，当用户操作了无权限的资源跳转至此

jsets.shiro.unauthorized-url=/unauthorized

2、AJAX请求

处理方式：响应403 HTTP状态码， 返回JSON信息{code:'auth:forbidden',message:'权限不足'}

展示效果：

C、启用验证码

wj596 edited this page on 1 Jan · [1 revision](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/C%E3%80%81%E5%90%AF%E7%94%A8%E9%AA%8C%E8%AF%81%E7%A0%81/_history)

**启用验证码验证配置：**

###jsets-shiro配置

#是否启用验证码验证,不配置默认不启用

jsets.shiro.jcaptcha-enable=true

#登陆页面

jsets.shiro.login-url=/login

#登陆成功页面

jsets.shiro.login-success-url=/index

#无权限页面

jsets.shiro.unauthorized-url=/unauthorized

#基本静态过滤规则

jsets.shiro.filte-rules[0]=/assets/\*\*-->anon

jsets.shiro.filte-rules[1]=/login/\*\*-->authc

jsets.shiro.filte-rules[2]=/logout/\*\*-->logout

jsets.shiro.filte-rules[3]=/\*\*-->user

**修改登陆页面：**

<form class="login-form" action="${ctx}/login" method="post">

<div class="form-item">

用户名：<input type="text" name="username" />

</div>

<div class="form-item">

密码：<input type="password" name="password" />

</div>

<div class="form-item">

验证码：<input type="text" name="jcaptcha" />

<img alt="点击切换" src="${ctx}/jcaptcha.jpg"

onclick="this.src='${ctx}/jcaptcha.jpg?d='+new Date()" />

</div>

<div class="form-item">

<button type="submit">登录</button>

</div>

</form>

<!-- 认证信息 -->

<font color="red">${Request["shiro\_auth\_message"]!}</font>

验证码效果：

验证码错误效果：

D、记住我设置

wj596 edited this page on 1 Jan · [1 revision](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/D%E3%80%81%E8%AE%B0%E4%BD%8F%E6%88%91%E8%AE%BE%E7%BD%AE/_history)

**记住我相关配置属性：**

#是否启用验证码验证,不配置默认不启用

jsets.shiro.jcaptcha-enable=true

#记住我有效期(单位为毫秒),不配置默认7天(604800000毫秒)

jsets.shiro.rememberme-max-age=604800000

#记住我 cookie加密秘钥(长度需要为8的倍数),不配置默认一组乱序数字

jsets.shiro.rememberme-secret-key=123456789abcdefh

#登陆页面

jsets.shiro.login-url=/login

#登陆成功页面

jsets.shiro.login-success-url=/index

#无权限页面

jsets.shiro.unauthorized-url=/unauthorized

#基本静态过滤规则

jsets.shiro.filte-rules[0]=/assets/\*\*-->anon

jsets.shiro.filte-rules[1]=/login/\*\*-->authc

jsets.shiro.filte-rules[2]=/logout/\*\*-->logout

jsets.shiro.filte-rules[3]=/\*\*-->user

**修改登陆页面：**

<form class="login-form" action="${ctx}/login" method="post">

<div class="form-item">

用户名：<input type="text" name="username" />

</div>

<div class="form-item">

密码：<input type="password" name="password" />

</div>

<div class="form-item">

验证码：<input type="text" name="jcaptcha" />

<img alt="点击切换" src="${ctx}/jcaptcha.jpg"

onclick="this.src='${ctx}/jcaptcha.jpg?d='+new Date()" />

</div>

<div class="form-item">

记住我：<input name="rememberMe" type="checkbox" />

</div>

<div class="form-item">

<button type="submit">登录</button>

</div>

</form>

<!-- 认证信息 -->

<font color="red">${Request["shiro\_auth\_message"]!}</font>

选中"记住我"复选框， 关闭浏览器，下次系统将会自动登陆。

E、账号唯一用户登陆

wj596 edited this page on 2 Jan · [1 revision](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/E%E3%80%81%E8%B4%A6%E5%8F%B7%E5%94%AF%E4%B8%80%E7%94%A8%E6%88%B7%E7%99%BB%E9%99%86/_history)

**账号唯一用户登陆相关配置属性：**

#是否启用验证码验证,不配置默认不启用

jsets.shiro.jcaptcha-enable=true

#记住我有效期(单位为毫秒),不配置默认7天(604800000毫秒)

jsets.shiro.rememberme-max-age=604800000

#记住我 cookie加密秘钥(长度需要为8的倍数),不配置默认一组乱序数字

jsets.shiro.rememberme-secret-key=123456789abcdefh

#是否启用账号唯一用户登陆(一个账号只允许一个用户登陆，后面登陆的用户踢出前面登陆的用户)

#不配置默认不启用

jsets.shiro.keep-one-enabled=true

#用户被踢出后跳转的页面,keep-one-enabled=true时此项有用,不配置默认login-url

jsets.shiro.kickout-url=/kickout

#登陆页面

jsets.shiro.login-url=/login

#登陆成功页面

jsets.shiro.login-success-url=/index

#无权限页面

jsets.shiro.unauthorized-url=/unauthorized

#基本静态过滤规则

jsets.shiro.filte-rules[0]=/assets/\*\*-->anon

jsets.shiro.filte-rules[1]=/login/\*\*-->authc

jsets.shiro.filte-rules[2]=/logout/\*\*-->logout

jsets.shiro.filte-rules[3]=/\*\*-->user

用户被踢出后的效果：

F、接入用户数据

wangjie edited this page on 11 Jan · [2 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/F%E3%80%81%E6%8E%A5%E5%85%A5%E7%94%A8%E6%88%B7%E6%95%B0%E6%8D%AE/_history)

认证（Authentication）即证明您是账号表示的那个人，基于form的认证中使用口令(密码)，rest api认证中使用令牌(token)。认证的过程需要账号数据，这个数据至少包含两个元素：账号(用户名)、密码，jsets-shiro-spring-boot-starter中使用Account接口表示账号数据、使用ShiroAccountProvider接口为鉴权操作提供Account数据，通过这个接口您可以从数据库、文件、LDAP、远程服务等各种渠道获取认证数据。

**实现Account接口：**

@Entity

@Table(name="t\_user")

public class UserEntity implements Account{

private static final long serialVersionUID = 1L;

// 状态-删除

public final static Short USER\_STATUS\_OK = 1;

// 状态-锁定

public final static Short USER\_STATUS\_LOCKED = 2;

// 状态-删除

private final static Short USER\_STATUS\_DELETED = 9;

@Id

@GeneratedValue(generator=IdGenerators.UUID)

private String id;// 主键 uuid

private String account; // 账号

private String password; // 密码

private String userName; // 用户姓名

private Short status;//状态 1:正常、2：锁定、9：删除

@Override

public String getAccount() {

return account;

}

@Override

public String getPassword() {

return password;

}

//省略其余 getters and setters ... ...

}

**实现ShiroAccountProvider接口：**

@Service

public class AccountProviderImpl implements ShiroAccountProvider {

@Autowired

private UserService userService;

@Autowired

private UserRoleService userRoleService;

/\*\*

\* 加载用户信息

\*/

@Override

public Account loadAccount(String account) throws AuthenticationException {

UserEntity user = userService.getByAccount(account);

// 用户不存在

if(null == user){

throw new AuthenticationException("账号或密码错误");

}

// 对账号做检查

// 当账号异常，如账号被锁定、被禁用等等需要限制登陆，直接抛出AuthenticationException即可

if(UserEntity.USER\_STATUS\_LOCKED == user.getStatus()){

throw new AuthenticationException("账号已被锁定，请联系系统管理员");

}

return user;

}

/\*\*

\* 加载用户持有的角色列表

\*/

@Override

public Set<String> loadRoles(String account) {

return Sets.newHashSet(userRoleService.listUserRoles(account));

}

/\*\*

\* 加载用户的权限列表

\* 系统安全策略是基于角色的权限访问控制(RBAC),故权限列表为空

\*/

@Override

public Set<String> loadPermissions(String account) {

return null;

}

}

**继承JsetsShiroConfigurationAdapter以扩展、定制鉴权组件：**

@Configuration

public class ApplicationSecurityConfig extends JsetsShiroConfigurationAdapter{

// 账号信息提供者实现

@Autowired

private AccountProviderImpl accountProviderImpl;

@Override

protected void configure(SecurityManagerConfig securityManager) {

// 设置账号信息提供者实现

securityManager.setAccountProvider(this.accountProviderImpl);

}

@Override

protected void configure(FilterChainConfig filterChain) {

}

}

您可以自行设置密码的加密方式，相关配置如下：

#密码加密算法,不配置默认MD5

jsets.shiro.passwd-alg=MD5

#密码加密HASH盐 ,不配置默认一组乱序数字

jsets.shiro.passwd-salt=ofaffadfev1234567--090swctewst

#密码加密HASH次数,不配置默认2次

jsets.shiro.passwd-iterations=2

为了保证认证成功，请保证密码加密存储时的加密口径(算法、HASH盐、HASH次数)和认证时的加密口径一直：

@Service

public class UserService {

@Autowired

private JdbcEnhance jdbcEnhance;

@Transactional

public void save(UserEntity user){

if(Strings.isNullOrEmpty(user.getId())){

// 密码明文加密存储，使用ShiroUtils工具会使用您在配置文件中设置的加密算法、HASH盐、HASH次数

user.setPassword(ShiroUtils.password(user.getPassword()));

jdbcEnhance.insert(user);

} else {

jdbcEnhance.update(user);

}

}

// 省略其余方法

}

G、密码错误重试次数限制

wangjie edited this page on 11 Jan · [2 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/G%E3%80%81%E5%AF%86%E7%A0%81%E9%94%99%E8%AF%AF%E9%87%8D%E8%AF%95%E6%AC%A1%E6%95%B0%E9%99%90%E5%88%B6/_history)

**密码错误重试次数相关配置属性：**

#密码错误最大重试次数,不配置默认无限次

jsets.shiro.passwd-max-retries=5

**实现PasswdRetryLimitHandler：**

public class PasswdRetryLimitHandlerImpl implements PasswdRetryLimitHandler{

private static final Logger LOGGER =

LoggerFactory.getLogger(PasswdRetryLimitHandlerImpl.class);

@Autowired

private UserService userService;

@Override

public void handle(String account) {

//锁定账号

userService.lockUser(account);

LOGGER.warn("账号："+account+"密码错误超过5次，已锁定");

}

}

PasswdRetryLimitHandler是一个回调接口，如果密码输入错误次数到达passwd-max-retries设置的阈值，将会调用这个接口的实现类，如果应用系统没有实现或者注入这个接口，则passwd-max-retries的设置无效。

**修改配置适配器ApplicationSecurityConfig：**

@Configuration

public class ApplicationSecurityConfig extends JsetsShiroConfigurationAdapter{

// 账号信息提供者实现

@Autowired

private AccountProviderImpl accountProviderImpl;

// 密码错误次数超限处理器实现

@Autowired

private PasswdRetryLimitHandlerImpl passwdRetryLimitHandlerImpl;

@Override

protected void configure(SecurityManagerConfig securityManager) {

// 设置账号信息提供者实现

securityManager.setAccountProvider(this.accountProviderImpl);

// 设置密码错误次数超限处理器实现

securityManager.setPasswdRetryLimitHandler(this.passwdRetryLimitHandlerImpl);

}

@Override

protected void configure(FilterChainConfig filterChain) {}

}

当用户密码输入错误时会出现如下效果：

密码输入错误次数超过5次用户账号将被锁定：

H、在线用户强制下线

wangjie edited this page on 11 Jan · [2 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/H%E3%80%81%E5%9C%A8%E7%BA%BF%E7%94%A8%E6%88%B7%E5%BC%BA%E5%88%B6%E4%B8%8B%E7%BA%BF/_history)

**强制下线相关配置属性：**

#是否启用强制用户下线(系统管理员通过session操作迫使指定用户下线),不配置默认不启用

jsets.shiro.force-logout-enable=true

#用户被强制下线后跳转的页面,force-logout-enable=true时此项有用,不配置默认login-url

jsets.shiro.force-logout-url=/force\_logout

**在线用户查询：**

@Controller

@RequestMapping("/online")

public class OnlineAction {

/\*\*

\* 当前在线用户

\*/

@RequestMapping("/list")

public String list(Model model) {

// 当前存活的Session数量

int onlineCount = ShiroUtils.getActiveSessionCount();

// 当前存活的Session列表

List<Session> activeSessions = ShiroUtils.getActiveSessions();

// 包装OnlineUserVo

List<OnlineUserVo> onlineUsers = Lists.newArrayList();

// 包装代码见DEMO

model.addAttribute("onlineCount",onlineCount);

model.addAttribute("onlineUsers",onlineUsers);

return "online/online\_list";

}

/\*\*

\* 强制SESSION下线

\*/

@RequestMapping("/do\_force\_fogout")

public String doForceLogout(@RequestParam(name="sessionId") String sessionId) {

ShiroUtils.forceLogout(sessionId);

return "redirect:/online/list";

}

}

在线用户列表:

强制下线效果：

I、缓存设置

wangjie edited this page on 15 Mar · [4 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/I%E3%80%81%E7%BC%93%E5%AD%98%E8%AE%BE%E7%BD%AE/_history)

组件默认使用内存缓存基于ConcurrentHashMap实现的。

如果您的项目中使用了Spring缓存(在配置类或者启动类上添加了@EnableCaching注解)，则会使用Spring缓存。

Spring默认的缓存是guava，如果检测到类路径下有ehcache.xml会自动启用ehcache缓存； 如果类路径下没有ehcache.xml而在application.x(y)ml中有spring.redis的配置则启用redis缓存。

**缓存相关配置属性：**

#是否启用认证\授权数据缓存，不配置默认不启用

jsets.shiro.auth-cache-enabled=true

auth-cache-enabled启用后会将用户的认证信息、用户的授权信息进行缓存，以后使用这些数据会优先从缓存中获取。比如要访问一个URL，shiro需要判断您是否有权限访问这个URL，此时就行需要查询用户的角色，auth-cache-enabled启用了从缓存中查询角色，没启用则从数据库中查询用户角色。

**ehcache**

单机部署的项目建议使用ehcache，简单方便，如果使用ehcache缓存，推荐在ehcache.xml中添加配置如下：

<?xml version="1.0" encoding="UTF-8"?>

<ehcache>

<diskStore path="java.io.tmpdir" />

<!-- ================= jsets-security-shiro相关缓存配置 ====================-->

<!-- 属性说明-->

<!-- name:cache的名称，必须是唯一的-->

<!-- maxEntriesLocalHeap:内存中保持的对象数量 -->

<!-- overflowToDisk:缓存数据数量超过maxElementsInMemory限制后是否缓存到磁盘上，这里设为true确保数据不丢失 -->

<!-- eternal:是否是永恒数据，这里设为true确保数据不丢失，清扫任务交由程序逻辑 -->

<!-- statistics:开启命中统计 -->

<!-- 活跃的SESSION缓存 -->

<cache name="shiro-activeSessionCache"

maxEntriesLocalHeap="2000"

overflowToDisk="true"

eternal="true"

statistics="true">

</cache>

<!-- 密码重试次数缓存 -->

<cache name="shiro-passwordRetryCache"

maxEntriesLocalHeap="2000"

overflowToDisk="true"

eternal="true"

statistics="true">

</cache>

<!-- 保持账号唯一用户登陆缓存 -->

<cache name="shiro-keepOneUserCache"

maxEntriesLocalHeap="2000"

overflowToDisk="true"

eternal="true"

statistics="true">

</cache>

<!-- 认证数据缓存 ，最后一次访问后一小时失效 -->

<cache name="shiro-authenticationCache"

maxEntriesLocalHeap="2000"

overflowToDisk="false"

eternal="false"

timeToIdleSeconds="3600"

statistics="true">

</cache>

<!-- 授权数据缓存 ，最后一次访问后一小时失效-->

<cache name="shiro-authorizationCache"

maxEntriesLocalHeap="2000"

overflowToDisk="false"

eternal="false"

timeToIdleSeconds="3600"

statistics="true">

</cache>

<!-- 已使用过的令牌缓存 ，仅存活一小时-->

<cache name="shiro-tokenBurnersCache"

maxEntriesLocalHeap="2000"

overflowToDisk="true"

eternal="false"

timeToLiveSeconds="3600"

statistics="true">

</cache>

</ehcache>

**redis**

集群部署的项目建议使用redis，集群共享方便

**认证授权信息缓存**

如果开启了认证/授权数据缓存jsets.shiro.auth-cache-enabled=true，请确保用户认证相关信息、用户授权相关信息发生改变时要进行缓存刷新，否则这些变更只能等到用户下次登陆时才能在系统中体现。

认证授权信息缓存同步代码：

@Service

public class UserService {

@Autowired

private JdbcEnhance jdbcEnhance;

@Autowired

private UserRoleService userRoleService;

/\*\*

\* 保存用户信息

\*/

@Transactional

public void save(UserEntity user){

if(Strings.isNullOrEmpty(user.getId())){

// 密码明文加密存储

user.setPassword(shiroSecurityService.password(user.getPassword()));

user.setStatus(UserEntity.USER\_STATUS\_OK);

jdbcEnhance.insert(user);

} else {

userRoleService.deleteByUser(user.getId());

jdbcEnhance.update(user);

}

// 保存用户角色

if(!Strings.isNullOrEmpty(user.getRoleCodes())){

for(String roleName:CommonUtil.split(user.getRoleCodes())){

UserRoleEntity userRole = new UserRoleEntity();

userRole.setUserId(user.getId());

userRole.setRoleId(roleName);

userRoleService.save(userRole);

}

// 清除该用户的认证和授权信息缓存

ShiroUtils.clearAuthCache(user.getAccount());

}

}

/\*\*

\* 修改用户状态(锁定\解锁\禁用\启用)

\*/

public void updateStatus(String account,Short status){

jdbcEnhance.update(SqlBuilder.BUILD()

.UPDATE("T\_USER")

.SET("STATUS = ?")

.WHERE("ACCOUNT = ?"),

status,account);

// 清除该用户的认证和授权信息缓存

ShiroUtils.clearAuthCache(user.getAccount());

}

// 其余操作参见jsets-shiro-demo

}

J、无状态鉴权

wangjie edited this page on 11 Jan · [9 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/J%E3%80%81%E6%97%A0%E7%8A%B6%E6%80%81%E9%89%B4%E6%9D%83/_history)

无状态(Stateless)鉴权通常应用在微服务(REST API)架构中，使用数字摘要(签名)技术生成一个token作为认证和授权的凭证，整个认证和授权过程不依赖于cookie或session，服务端不保留客户端状态因此每次请求都要携带这个token。

jsets-shiro-spring-boot-starter提供两种无状态鉴权方式，分别是散列消息认证码(HMAC)、JSON WEB TOKEN(JWT)。

**HMAC鉴权**

HMAC运算利用哈希算法，以一个密钥和一个消息为输入，生成一个消息摘要作为输出,使用HMAC作为rest api的安全验证协议最显著的两个特点就是防止密码在网络上传递和防止请求信息被篡改。

HMAC请求流程：

服务端接收到请求，也按照和请求发送时一样的摘要生成方式和密码生成服务端摘要，如果请求的摘要和服务端摘要不同则说明是非法请求。

HMAC鉴权相关配置属性：

#是否启用HMAC鉴权，不配置默认不启用

jsets.shiro.hmac-enabled=true

#是否启用HMAC鉴权，不配置默认不启用

jsets.shiro.hmac-enabled=true

#是否启用HMAC签名即时销毁，不配置默认不启用

#jsets.shiro.hmac-burn-enabled=true

#HMAC签名算法,不配置默认HmacMD5,hmac-enabled=true时此项有用

#jsets.shiro.hmac-alg=HmacMD5

#HMAC签名全局秘钥,hmac-enabled=true时此项有用

jsets.shiro.hmac-secret-key=ofaffadfev1234567--090swctewst

#HMAC签名有效期,不配置默认1分钟,hmac-enabled=true时此项有用

#jsets.shiro.hmac-period=60000

HMAC URL过滤规则(过滤器)配置：

#匹配此路径的URL,需要通过hmac认证并具有admin角色

jsets.shiro.filte-rules[3]=/restApi/delete\*-->hmacRoles[admin]

#匹配此路径的URL,需要通过hmac认证

jsets.shiro.filte-rules[4]=/restApi/\*\*-->hmac

**JWT鉴权：**

json web token是一个轻量级开放标准，也是使用HASH算法进行摘要，生成token中包含了头信息和荷载信息。JWT是一个自包含的令牌，即在荷载信息中包含用户鉴权所需所有信息（用户名、角色、权限等等），只需要对token本身进行验签，验签过程中不需要通过数据库查询用户信息。 JWT荷载信息：

Playload//荷载信息

{

"iss": "token-server",//签发者

"exp ": "Mon Nov 13 15:28:41 CST 2017",//过期时间

"sub ": "wangjie",//用户名

"aud": "web-server-1"//接收方,

"nbf": "Mon Nov 13 15:40:12 CST 2017",/生效时间

"jat": "Mon Nov 13 15:20:41 CST 2017",//签发时间

"jti": "0023",//令牌ID标识

"claim": {"auth":"ROLE\_ADMIN"}//访问主张

}

JWT URL鉴权配置属性：

#是否启用JWT鉴权，不配置默认不启用

jsets.shiro.jwt-enabled=true

#是否启用JWT令牌即时销毁，不配置默认不启用,jwt-enabled=true时此项有用

#jsets.shiro.jwt-burn-enabled=true

#JWT签名签名全局秘钥,jwt-enabled=true时此项有用

jsets.shiro.jwt-secret-key=ofaffadfev1234567--090swctewst

URL过滤规则(过滤器)配置：

#匹配此路径的URL,需要通过jwt认证并具有admin角色

jsets.shiro.filte-rules[5]=/restApi2/delete\*-->jwtRoles[admin]

#匹配此路径的URL,需要通过jwt认证

jsets.shiro.filte-rules[6]=/restApi2/\*\*-->jwt

**HMAC摘要和JWT令牌生成工具：**

[CryptoUtil](https://github.com/wj596/jsets-shiro-spring-boot-starter/blob/master/src/main/java/org/jsets/shiro/util/CryptoUtil.java?_blank)

如果您要在客户端或第三放系统生成HMAC摘要，使用这个工具类中的hmacDigest(String plaintext,String secretKey,String algName)放，其中algName属性为算法名称，可以使用类中的常量：

// HMAC 加密算法名称

public static final String HMAC\_MD5 = "HmacMD5";// 128位

public static final String HMAC\_SHA1 = "HmacSHA1";// 126位

public static final String HMAC\_SHA256 = "HmacSHA256";// 256位

public static final String HMAC\_SHA512 = "HmacSHA512";// 512位

请保证秘钥和加密算法和验证端一致。

如果您要在客户端或第三放系统(令牌签发方)生成JWT令牌，先在系统中引入jjwt包：

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.0</version>

</dependency>

然后使用这个工具类中的issueJwt()方法，进行令牌签发，请保证秘钥和验证端一致。

**无状态鉴权的响应状态：**

在"ajax响应"一节中我们用401状态响应未登陆，用403状态响应未授权，无状态鉴权的响应状态和JSON消息与其一致。

**重放攻击：**

所谓重放攻击是指攻击者发送一个目的主机已接收过的包，来达到欺骗系统的目的，主要用于身份认证过程，破坏认证的正确。我们主要是通过时间戳和缓存两种方式来处理。

HMAC签名由于在生成摘要时混入了时间戳(毫秒精度)即使请求的内容一样，每次生成的摘要内容也不一样，所以具备了一次性消费的特性。验证端在验签时会根据上面hmac-period属性的配置(默认是60秒)计算出这个签名生成的时间距离验签时间是否在这个范围内，如果不在就视为失效。JWT的荷载信息中有个exp属性，这个是JWT的有效期，同样在验签时如果过了有效期也视为令牌失效。

基于缓存的方式就是每次验证完成后将签名或令牌ID放入缓存，下次验证时先到缓存中查看是否存在相同的签名或令牌ID，如果存在则视签名和令牌为作废的，不管HMAC和JWT设置的有效期是多少，每个签名或令牌只能使用一次，就是我们通常所说的阅后即焚。您可以在配置文件中启用：

#是否启用HMAC签名即时销毁，不配置默认不启用

#jsets.shiro.hmac-burn-enabled=true

#是否启用JWT令牌即时销毁，不配置默认不启用,jwt-enabled=true时此项有用

#jsets.shiro.jwt-burn-enabled=true

这种处理的方式显然比第一种要安全多了，但是对内存的负载会加大，所以只有您使用的缓存是ehcache或者redis时启用burn-enabled才有效，如果不是对安全性有较高的要求还是推荐使用第一种。

**无状态鉴权的账号数据：**

在"接入用户数据"一节我们用ShiroAccountProvider账号数据提供者接口为有状态鉴权提供账号数据，无状态鉴权默认也是使用这个接口来获取鉴权数据。但是如果您想要无状态鉴权使用独立的账号数据也是可以的，不排除会有这样的需求，比如一个系统即是管理系统同时也行第三方系统提供服务，可能登陆管理系统需要一套账号体系，为第三方系统提供服务需要另外一套账号体系，另外我们在配置文件中的属性hmac-secret-key和jwt-secret-key都是全局秘钥，如果您想要每个无状态鉴权账号都使用自己独立的秘钥，可以使用ShiroStatelessAccountProvider为无状态鉴权提供数据。

实现ShiroStatelessAccountProvider:

@Service

public class StatelessAccountProviderImpl implements ShiroStatelessAccountProvider{

// API客户端管理

@Autowired

private ApiClientService apiClientService;

/\*\*

\* 检查账号是否正常

\* 如果返回false或抛出AuthenticationException则说明账号异常，不予通过认证。

\*/

public boolean checkAccount(String appId) throws AuthenticationException{

return apiClientService.isLocked(appId);

}

/\*\*

\* 获取客户端的签名私钥

\* 如果您要使用全局秘钥，即属性中配置的hmac-secret-key、jwt-secret-key此方法返回null。

\* 如果您要使用各账号自己的秘钥，不要在属性中配置的hmac-secret-key、jwt-secret-key

\*/

public String loadAppKey(String appId){

return apiClientService.getAppKey(appId);

}

/\*\*

\* 根据客户标识加载持有角色

\*/

public Set<String> loadRoles(String appId){

return apiClientService.listRoles(appId);

}

/\*\*

\* 根据客户标识加载持有权限

\*/

public Set<String> loadPermissions(String appId){

return apiClientService.listPermissions(appId);

}

}

修改配置适配器ApplicationSecurityConfig：

@Configuration

public class ApplicationSecurityConfig extends JsetsShiroConfigurationAdapter{

// 账号信息提供者实现

@Autowired

private AccountProviderImpl accountProviderImpl;

// 无状态鉴权账号信息提供者实现

@Autowired

private StatelessAccountProviderImpl statelessAccountProvider;

// 密码错误次数超限处理器实现

@Autowired

private PasswdRetryLimitHandlerImpl passwdRetryLimitHandler;

@Override

protected void configure(SecurityManagerConfig securityManager) {

// 设置账号信息提供者实现

securityManager.setAccountProvider(this.accountProviderImpl);

// 设置密码错误次数超限处理器实现

securityManager.setPasswdRetryLimitHandler(this.passwdRetryLimitHandler);

// 无状态鉴权账号信息提供者实现，如果不设置此项无状态鉴权默认使用this.shiroAccountProvider

securityManager.setStatelessAccountProvider(this.statelessAccountProvider);

}

@Override

protected void configure(FilterChainConfig filterChain) {}

}

这样设置之后，有状态鉴权就使用shiroAccountProvider来获取账号数据，无状态鉴权就使用statelessAccountProvider来获取账号数据，如果不进行setStatelessAccountProvider操作，则有状态鉴权和无状态鉴权都使用shiroAccountProvider获取账号数据。

**HMAC摘要和JWT令牌使用使用场景：**

推荐使用HMAC进行鉴权场景：

推荐使用JWT鉴权的场景：

K、动态URL过滤规则

wangjie edited this page on 12 Jan · [2 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/K%E3%80%81%E5%8A%A8%E6%80%81URL%E8%BF%87%E6%BB%A4%E8%A7%84%E5%88%99/_history)

权限验证（Authorization）即根据判断规则来验证是否能对一个资源执行操作。权限泛指能否操作资源，角色是权限的集合，如果使用权限来表示一个用户能操作的资源显然不方便数量太多，所以大部分系统的安全模型都是RBAC(Role-Based Access Control 基于角色的权限访问控制)，使用用户--角色--资源(URL)的对应关系来表示，对应关系是持久化并且可变更的。

权限验证需要两种数据权限数据和验证规则数据，jsets-shiro-spring-boot-starter使用ShiroAccountProvider接口为鉴权操作提供权限(角色)数据，这个接口在"接入用户数据"一章有说明。

如果没有规则就不存在判断，权限验证也就无从谈起，在shiro中规则表示成这样：

article/update=roles[role\_editor]

article/delete=roles[role\_chief]

article/publish=roles[role\_chief]

这组规则表示，编辑角色(role\_editor)拥有编辑的权限、主编角色(role\_chief)拥有删除和发布的权限，"article/update"就是操作的资源，"roles[role\_editor]"是判断标准即断言用户的角色列表中包含"role\_editor"角色，操作加上判断标准就是规则。

在shiro中您也可以这样表示权限验证规则

@RequiresRoles("role\_editor")

public void update(Article article){

}

这两种方式是有局限的，比如角色-资源的对应关系发生变更，比如新增了一种角色，那就只能修改配置或者代码然后重启动系统才能使得这些变更生效。实际开发中则更希望通数据库查询出角色-资源的对应数据，动态生成URL规则，当角色-资源的对应关系发生变更时能刷新并立即应用这些规则。

jsets-shiro-spring-boot-starter使用ShiroFilteRulesProvider接口为鉴权操作提供权限验证规则数据，支持实时刷新规则。

实现ShiroFilteRulesProvider：

@Service

public class FilteRulesProviderImpl implements ShiroFilteRulesProvider{

@Autowired

private JdbcEnhance jdbcEnhance;

/\*\*

\* 加载基于角色/资源的权限验证

\* RolePermRule包含三个属性(url:资源URL、needRoles:访问需要的角色列表、needPerms:访问需要的权限列表)

\*/

@Override

public List<RolePermRule> loadRolePermRules(){

List<RolePermRule> rolePermRules = this.jdbcEnhance

.selector()

.SELECT("URL,GROUP\_CONCAT(T.ROLE\_ID ORDER BY R.URL) NEED\_ROLES")

.FROM("T\_ROLE\_RESOURCE T")

.JOIN("T\_RESOURCE R ON T.RESOURCE\_ID = R.ID")

.GROUP\_BY("R.URL")

.entityClass(RolePermRule.class)

.list();

return rolePermRules;

}

/\*\*

\* HMAC数字签名权限验证规则配置

\* 为了展示数据原貌使用了硬编码配置

\* 真实场景中可以通过界面配置，比如做一个"接口管理"功能

\*/

@Override

public List<RolePermRule> loadHmacRules() {

// 删除操作需要通过HMAC数字摘要认证并且具有"role\_admin"角色

RolePermRule hmacRule1 = new RolePermRule();

hmacRule1.setUrl("/hmac\_api/delete");

hmacRule1.setNeedRoles("role\_admin");

/// 其余操作需要通过HMAC数字摘要认证

RolePermRule hmacRule2 = new RolePermRule();

hmacRule2.setUrl("/hmac\_api/\*\*");

List<RolePermRule> hmacRules = Lists.newLinkedList();

hmacRules.add(hmacRule1);

hmacRules.add(hmacRule2);

return hmacRules;

}

/\*\*

\* JWT令牌权限验证规则配置

\* 为了展示数据原貌使用了硬编码配置

\* 真实场景中可以通过界面配置，比如做一个"接口管理"功能

\*/

@Override

public List<RolePermRule> loadJwtRules() {

/// 删除操作需要通过JWT令牌认证并且具有"role\_admin"角色

RolePermRule jwtRule1 = new RolePermRule();

jwtRule1.setUrl("/jwt\_api/delete");

jwtRule1.setNeedRoles("role\_admin");

/// 其余操作需要通过JWT令牌认证

RolePermRule jwtRule2 = new RolePermRule();

jwtRule2.setUrl("/jwt\_api/\*\*");

List<RolePermRule> jwtRules = Lists.newLinkedList();

jwtRules.add(jwtRule1);

jwtRules.add(jwtRule2);

return jwtRules;

}

/\*\*

\* 自定义权限验证规则，如果您熟悉shiro的规则表达式，可自定义规则以便定制更简洁有效的规则

\* 如果你扩展了shiro的Filter,通过这个方法为扩展的Filter指定规则

\*/

@Override

public List<CustomRule> loadCustomRules() {

return null;

}

}

loadRolePermRules加载基于角色或者资源的过滤规则，sql语句查询的数据如下：

根据这些数据生成响应的过滤规则。

loadHmacRules、loadJwtRules加载需要HMAC、JWT鉴权的过滤规则，根据上面代码会生成如下shiro过滤规则：

/hmac\_api/delete=hmacRoles[role\_admin]

/hmac\_api/\*\*=hmac

/jwt\_api/delete=hmacRoles[role\_admin]

/jwt\_api/\*\*=jwt

修改配置适配器ApplicationSecurityConfig：

@Configuration

public class ApplicationSecurityConfig extends JsetsShiroConfigurationAdapter{

// 账号信息提供者实现

@Autowired

private ShiroAccountProviderImpl accountProviderImpl;

// 无状态鉴权账号信息提供者实现

@Autowired

private StatelessAccountProviderImpl statelessAccountProvider;

// 密码错误次数超限处理器实现

@Autowired

private PasswdRetryLimitHandlerImpl passwdRetryLimitHandler;

// 动态过滤规则提供者实现

@Autowired

private FilteRulesProviderImpl filteRulesProviderImpl;

@Override

protected void configure(SecurityManagerConfig securityManager) {

// 设置账号信息提供者实现

securityManager.setAccountProvider(this.accountProviderImpl);

// 设置密码错误次数超限处理器实现

securityManager.setPasswdRetryLimitHandler(this.passwdRetryLimitHandler);

// 无状态鉴权账号信息提供者实现，如果不设置此项无状态鉴权默认使用this.shiroAccountProvider

securityManager.setStatelessAccountProvider(statelessAccountProvider);

}

@Override

protected void configure(FilterChainConfig filterChain) {

// 设置过滤规则提供者，实现动态URL鉴权过滤

filterChain.setShiroFilteRulesProvider(this.filteRulesProviderImpl);

}

}

这样在系统启动的时候会通过shiroFilteRulesProvider将所有的过滤规则从数据库加载到shiro中，如果您在角色管理中将角色对应的资源做了修改需要进行过滤规则刷新：

@Service

public class RoleResourceService {

@Autowired

private JdbcEnhance jdbcEnhance;

@Transactional

public void save(String roleId, String resourceIds){

this.deleteResourceByRole(roleId);

for(String resourceId:CommonUtil.split(resourceIds)){

RoleResourceEntity roleResource = new RoleResourceEntity();

roleResource.setRoleId(roleId);

roleResource.setResourceId(resourceId);

this.save(roleResource);

}

}

public void save(RoleResourceEntity roleResource){

jdbcEnhance.insert(roleResource);

// 角色对应的资源改变，要刷新动态过滤规则，以便及时应用更新

ShiroUtils.reloadFilterRules();

}

public void deleteResourceByRole(String roleId){

jdbcEnhance.delete(SqlBuilder.BUILD()

.DELETE\_FROM("T\_ROLE\_RESOURCE")

.WHERE("ROLE\_ID = ?")

, roleId);

// 角色对应的资源改变，要刷新动态过滤规则，以便及时应用更新

ShiroUtils.reloadFilterRules();

}

}

L、扩展接口

wangjie edited this page on 12 Jan · [3 revisions](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/L%E3%80%81%E6%89%A9%E5%B1%95%E6%8E%A5%E5%8F%A3/_history)

组件的扩展是通过JsetsShiroConfigurationAdapter实现的，这是一个适配类。

实现JsetsShiroConfigurationAdapter：

@Configuration

public class ApplicationSecurityConfig extends JsetsShiroConfigurationAdapter{

@Override

protected void configure(SecurityManagerConfig securityManager) {

// 设置账号信息提供者

securityManager.setAccountProvider(ShiroAccountProvider的实现);

// 设置密码错误次数超限处理器

securityManager.setPasswdRetryLimitHandler(PasswdRetryLimitHandler的实现);

// 设置无状态鉴权(HMAC)账号信息提供者

securityManager.setStatelessAccountProvider(StatelessAccountProvider的实现);

// 设置RememberMe Cookie的模板

// 如需要定制RememberMe Cookie的name、domain、httpOnly可设置此项

securityManager.setRememberMeCookie(SimpleCookie);

// 设置SessionDAO

// 如果组件提供的session缓存方式(内存、ehcache、redis)无法满足需求，可设置此项定制session持久化

securityManager.setSessionDAO(SessionDAO);

// 如果组件提供的缓存方式(内存、ehcache、redis)无法满足需求，可设置此项定制缓存实现

securityManager.setCacheManager(CacheManager);

// 添加鉴权控制域

// 组件中提供三个控制域

// AccountPasswdRealm:有状态用户名,密码鉴权控制域

// HmacRealm:无状态hmac签名鉴权控制域

// JwtRealm:无状态jwt令牌鉴权控制域

// 如果无法满足需求，可设置此项添加鉴权控制域

securityManager.addRealm(Realm);

}

@Override

protected void configure(FilterChainConfig filterChain) {

// 设置过滤规则提供者，实现动态URL鉴权过滤

filterChain.setShiroFilteRulesProvider(ShiroAccountProvider的实现);

// 添加鉴权过滤

// 组件中提供的过滤器：

// authc:基于表单的登陆过滤器

// roles:基于角色的验证过滤器

// perms:基于权限的验证过滤器

// user:断言seesion中存在用户的过滤器

// keepOne:账号唯一用户登陆过滤器

// forceLogout:强制用户下线过滤器

// hmac:hmac数字签名认证过滤器

// hmacRoles:hmac数字签名角色验证过滤器

// hmacPerms:hmac数字签名权限验证过滤器

// jwt:jwt令牌认证过滤器

// jwtRoles:jwt令牌角色验证过滤器

// jwtPerms:jwt令牌权限验证过滤器

// 如果无法满足需求，可设置此项覆盖或者添加过滤器

filterChain.addFilter(filterName, Filter);

}

}

M、常见问题

wangjie edited this page on 2 Apr · [1 revision](https://github.com/wj596/jsets-shiro-spring-boot-starter/wiki/M%E3%80%81%E5%B8%B8%E8%A7%81%E9%97%AE%E9%A2%98/_history)

**1、spring boot actuator的监控路径被拦截，如/env、/info、/metrics、/health等。致使spring boot admin server无法获取监控数据**

1、为jmx添加访问上下文

#监控(JMX)配置

management:

context-path: /mgt

2、将/mgt/\*\*使用anon过滤器，也就是过不过滤匹配/mgt/\*\*的路径，这部分的访问安全空间有management自己来。

filte-rules:

- /layui/\*\*,/mgt/\*\*-->anon

- /login/\*\*-->authc

- /logout/\*\*-->logout

- /\*\*-->user