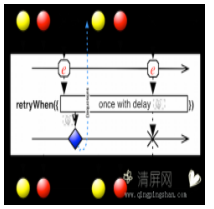
Spring Security整合JSON Web Token(JWT)提升REST安全性

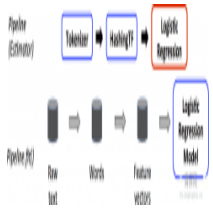
2017-04-17 10:44 出处：清屏网 人气：451 评论（0）

一、背景

由于本人最近在维护 SSM 的项目，因为上一个人使用 Spring Security 这个安全框架，所以在这里也不改了，继续使用这个框架（勇敢地往坑里跳吧），而我们的这个项目主要是为移动 APP 搭建的后台。我们都知道 APP 对 cookie 的操作都是力不用心的，不同于 WEB 开发，浏览器会主动来维护这个状态，这个 cookie 的名称为 JSESSIONID，而 HTTP 是无状态的，但也可以通过 ***http://api.example.com/user;jsessionid=xxxx?p=a*** 这样的方式传递给服务器，服务器就能识别出对应的 session。现在的状况是 APP 的每个请求都用这个方式传递，不然服务器就认为该请求未登录，所以会被安全框架拦截，从而拿不到正常的数据。而服务器每次都要维护这么多的 session，会占用内存，以后做集群也不方便（用 Spring Session 缓存到 Rides 中也是可以做集群的，基于 WEB 应用），那有没有更好的解决方案呢？答案当然是有的啦！（这不是废话，哈哈）我们可以基于 token，也可以用 OAuth2（但对于我们这些只给自己应用而不是提供给第三方使用的，用这个就有点臃肿了），然后最近使用某歌、某度，发现有个简单安全的一个标准，deng deng deng deng，那就是 JSON Web Token 简称 JWT，所以决定用它来代替这个被我们的安卓小伙吐槽得不能再吐槽的基于 cookie 的认证方式了，Here we go!

本类最热新闻


对RxJava中.repeatWhen()和.retr


MLlib1.6指南笔记

- RxDownload：基于RxJava打造的下载工具, 支持多...
- HoloLens开发手记：使用Visual Studio Using Vis...
- dotnet core搭建持续集成环境
- RocketMQ源码三：Producer消息发送过程
- java集成pdf.js实现pdf文件在线预览

什么是 JWT **JSON Web Token** 呢？网上的大神们已经描述得非常清晰了，毕竟人家是专业人士，我只是一只在不断学习的菜鸟，所以这里就不说了。这里提供一些我通过 xx 度搜到技术文章，里面有更通俗易懂的解说。

48小时最热 7天最热 7天热评 月榜

方程式0day ETERNALBLUE复现过程

使用C++ STL库统计一散文中单词出现次数和行..

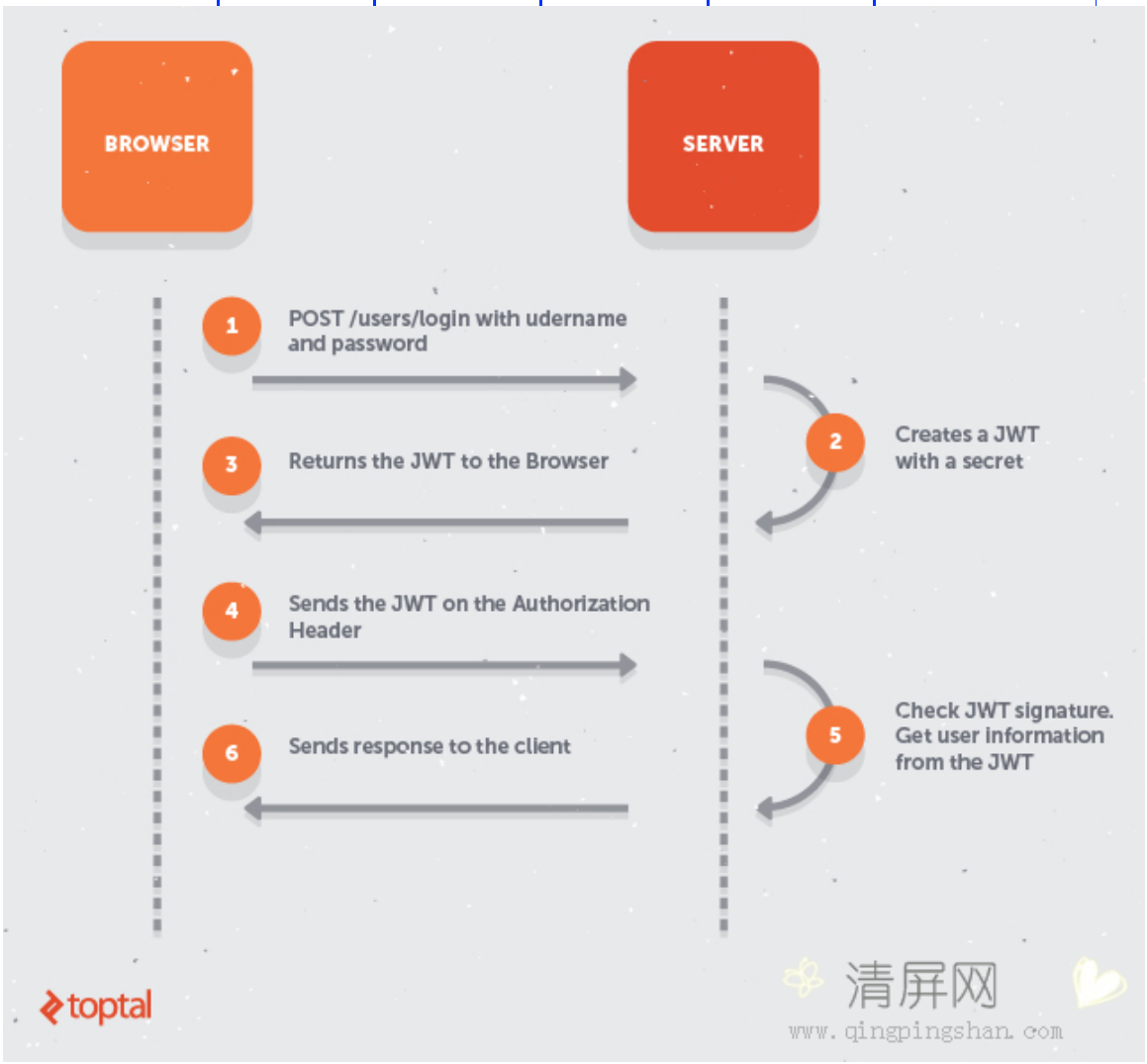
Win10 16176更新内容大全汇总

详解Android中ListView实现图文并列并且自定...

HTTPS中的密码技术

女性经期为什么会上火？

自适应Tab宽度可以滑动文字逐渐变色的TabLa...



TGP运行穿越火线没有声音怎么办？

Win10开热点总会莫名的自动关闭如何解决？

你为什么愿意为了一杯喜茶排队四五个钟头

推荐阅读：

- *基于Token的WEB后台认证机制 - 红心李*

- ***REST Security with JWT, Spring Security and Java***

三、Spring Security 与 JWT

为 **JSESSIONID** 的字段，因为在 登录成功 后会把验证信息保存到这个对应的 session 中，所以根据 JSESSIONID 的值取出对应的 session 就能获取是否已认证，和相应的用户权限了。

而这里想要集成 **JWT** ，这里就使用自己提供的 **Filter** ，由我们来自定义一个基于 JWT 的 Filter。


```
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {

    private final Log logger = LogFactory.getLog(this.getClass());

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Value("${jwt.header}")
    private String tokenHeader;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletRe
sponse response, FilterChain chain) throws ServletException, IOException {
        String authToken = request.getHeader(this.tokenHeader);
        // authToken.startsWith("Bearer ")
        // String authToken = header.substring(7);
        String username = jwtTokenUtil.getUsernameFromToken(authToken);
```

```
        if (username != null && SecurityContextHolder.getContext().getAuthent
ication() == null) {

            // It is not compelling necessary to load the use details from th
e database. You could also store the information
            // in the token and read it from it. It's up to you ;)
            UserDetails userDetails = this.userDetailsService.loadUserByUsern
ame(username);

            // For simple validation it is completely sufficient to just chec
k the token integrity. You don't have to call
            // the database compellingly. Again it's up to you ;)
            if (jwtTokenUtil.validateToken(authToken, userDetails)) {
                UsernamePasswordAuthenticationToken authentication = new User
namePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                logger.info("authenticated user " + username + ", setting sec
urity context");
                SecurityContextHolder.getContext().setAuthentication(authenti
cation);
            }
        }

        chain.doFilter(request, response);
    }
}
```

这个代码是国外的大神写的

当然这个代码只能做参考，因为我们要基于自己的业务逻辑进行修改的，客官先别急，下面再来实现我们自己的 Filter，这里先做一下铺垫。

AuthenticationEntryPoint

这个是拦截成功，需要登录权限时会走这个，上面的大神中有使用这个

```
<http pattern="/api/**" entry-point-ref="restAuthenticationEntryPoint" create
-session="stateless"> (3)
      <csrf disabled="true"/>  (4)
      <custom-filter before="FORM_LOGIN_FILTER"
ref="jwtAuthenticationFilter"/>  (5)
</http>
```

然后他自定义了一个实现类

RestAuthenticationEntryPoint.java

```
public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint
 {

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse resp
onse, AuthenticationException authException) throws IOException {
        // This is invoked when user tries to access a secured REST resource
 without supplying any credentials
        // We should just send a 401 Unauthorized response because there is n
o 'login page' to redirect to
```

```
    }
}
```

他这里只是在认证失败时返回 401 状态码，其实在这里我们也可以重定向到某个页面，这个就是相当于我们在 **spring-security.xml** 中配置的 **form-login** 标签的 **login-page** 属性，认证失败会把这个跳转动作交给入口点来处理， *详情* ，现在我觉得就使用认的自动跳转到我们配置文件中 **login-page** 就可以了，所以这个就不需要引用了。

- ***Spring Security** 使用自定义控制器来完成登陆验证*
- ***spring-security**中的**entry-point-ref**属性*

这里依然使用可缓存的 UserDetails， *缓存* ***UserDetails*** 。

缓存 UserDetails

- *初识 **Spring Security - Filter***
- *初识 **Spring Security - 缓存 UserDetails***

四、自定义实现

好了，开始撸吧！

自定义 UserDetails

由于这里整合 JWT 时利用 token 传输一些非敏感的关键信息


```
{
   "exp": 1491547421,
   "enabled": true,
   "sub": "13800138000",
   "scope": ["ROLE_USER"],
   "non_locked": true,
   "non_expired": true,
   "jti": "41dca17b-ce77-4308-8cc3-f9a56bffe81c",
   "user_id": 66666,
   "iat": 1491541421
}
```

因为这些信息在验证时会还原成一个框架要求的完整用户，框架自带的子类无法满足，所以我们自定一个实现类来保存这些信息

```
/**
 * JWT保存的用户信息
```

```
 * @since 2017-04-05
 */
public class JWTUserDetails implements UserDetails {

    private Long userId;
    private String password;
    private final String username;
    private final Collection<? extends GrantedAuthority> authorities;
    private final boolean accountNonExpired;
    private final boolean accountNonLocked;
    private final boolean credentialsNonExpired;
    private final boolean enabled;


    public JWTUserDetails(long userId, String username, String password, Coll
ection<? extends GrantedAuthority> authorities) {
        this(userId, username, password, true, true, true, true,
authorities);
    }

    public JWTUserDetails(long userId, String username, String password, bool
ean enabled, boolean accountNonExpired, boolean credentialsNonExpired, boolea
n accountNonLocked, Collection<? extends GrantedAuthority> authorities) {
        if (username != null && !"".equals(username) && password != null) {
            this.userId = userId;
            this.username = username;
            this.password = password;
            this.enabled = enabled;
            this.accountNonExpired = accountNonExpired;
            this.credentialsNonExpired = credentialsNonExpired;
            this.accountNonLocked = accountNonLocked;
            this.authorities = authorities;
```

```
ues to constructor");
        }
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    public long getUserId() {
        return userId;
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return accountNonExpired;
    }

    @Override
    public boolean isAccountNonLocked() {
        return accountNonLocked;
    }
```

```
    public boolean isCredentialsNonExpired() {
        return credentialsNonExpired;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }
}
```

修改相应的 UserDetailsService 返回我们自定义的 UserDetails `JWTUserDetails`


```
/**
 * 提供认证所需的用户信息
 *
 * @author ybin
 * @since 2017-03-08
 */
public class UserDetailsServiceCustom implements UserDetailsService {

    protected final Log logger = LogFactory.getLog(this.getClass());

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNot
FoundException {

        // 1. 根据用户标识获取用户
        ...

        if (user == null) {
```

```
        }

        // 2. 获取用户权限
        ...

        UserDetails userDetails = new JWTUserDetails(userId, username, passwo
rd,
                enabled, accountNonExpired, credentialsNonExpired, accountNon
Locked, authorities);

        return userDetails;
    }
```

现在有了自定义的 UserDetails，那我们怎么从 token 中还原成一个实体呢？那么就要提供一个自己 Filter 了

自定义 Filter

```
/**
 * JWT认证令牌过滤器
 *
 * @author ybin
 * @since 2017-04-05
 */
public class JWTAuthenticationFilter extends OncePerRequestFilter {

    @Value("${jwt.header}")
    private String token_header;

    @Resource
```

```
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletRe
sponse response, FilterChain chain) throws ServletException, IOException {

        String auth_token = request.getHeader(this.token_header);
        final String auth_token_start = "Bearer ";
        if (StringUtils.isNotEmpty(auth_token) && auth_token.startsWith(auth_
token_start)) {
            auth_token = auth_token.substring(auth_token_start.length());
        } else {
            // 不按规范,不允许通过验证
            auth_token = null;
        }

        String username = jwtUtils.getUsernameFromToken(auth_token);
        logger.info(String.format("Checking authentication for user %s.", use
rname));

        if (username != null && SecurityContextHolder.getContext().getAuthent
ication() == null) {
            // It is not compelling necessary to load the use details from th
e database. You could also store the information
            // in the token and read it from it. It's up to you ;)
            // UserDetails userDetails = this.userDetailsService.loadUserByUs
ername(username);
            UserDetails userDetails = jwtUtils.getUserFromToken(auth_token);

            // For simple validation it is completely sufficient to just chec
k the token integrity. You don't have to call
            // the database compellingly. Again it's up to you ;)
            if (jwtUtils.validateToken(auth_token, userDetails)) {
                UsernamePasswordAuthenticationToken authentication = new User
```

```
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                logger.info(String.format("Authenticated user %s, setting sec
urity context", username));
                SecurityContextHolder.getContext().setAuthentication(authenti
cation);
            }
        }

        chain.doFilter(request, response);
    }

}
```

这样就可以还原这个已登录的用户了，就不用每次去利用 **this.userDetailsService.loadUserByUsername(username);** 调用数据库了。

Filter 也有了，接下来的怎么用呢？配置 **spring-security.xml**

```
<http use-expressions="false" create-session="stateless">
    <!-- 关闭 CSRF 保护 -->
    <csrf disabled="true"/>

    <custom-filter before="FORM_LOGIN_FILTER" ref="jwtAuthenticationFilter"/>

    <!-- ###################### 不需要控制权限 start ###################### -->
    <!-- 登录页面 -->
    <intercept-url pattern="/account/login" access="IS_AUTHENTICATED_ANONYMOU
SLY"/>
```

```
CATED_ANONYMOUSLY"/>-->
    <!-- ##################### 不需要控制权限  end  ##################### -->

    <!-- ##################### 需要控制权限  start  ##################### -->
    <!-- 访问其他所有页面都需要有USER权限 -->
    <intercept-url pattern="/**" access="ROLE_USER" />
    <!-- ##################### 需要控制权限   end   ##################### -->

    <!-- 配置登录页面地址login-page、登录失败后的跳转地址authentication-failure-u
rl -->
    <form-login login-page="/account/login" />
    <!--<!- 已经超时的 sessionId 进行请求需要重定向的页面 ->-->
    <!--<session-management invalid-session-url="/account/invalid/session">--
>
        <!--<!- 设置一个帐号同时允许登录多少次 ->-->
        <!--<concurrency-control max-sessions="1" />-->
    <!--</session-management>-->
</http>

<beans:bean id="jwtUtils" class="org.spring.security.jwt.JWTUtils"/>
<beans:bean id="jwtAuthenticationFilter" class="org.spring.security.filter.JW
TAuthenticationFilter"/>
```

因为我们这里基于 JWT 了，所以不再需要以前那恶心的 session 了，所以设为 **create-session="stateless"** ，对应之前的 session 管理的相关标签也要注释掉，否则会在运行项目时发生如下冲突

```
07-Apr-2017 13:41:19.869 严重 [RMI TCP Connection(3)-127.0.0.1] org.apache.cat
alina.core.StandardContext.listenerStart Exception sending context initialize
d event to listener instance of class org.springframework.web.context.Context
```

```
nfiguration problem: session-management  cannot be used in combination with c
reate-session='STATELESS' Offending resource: file .../spring-security.xml
```

JWTUtils.java

```
/**
 * jwt-token工具类
 *
 * @author ybin
 * @since 2017-04-04
 */
public class JWTUtils {

    public static final String ROLE_REFRESH_TOKEN = "ROLE_REFRESH_TOKEN";

    private static final String CLAIM_KEY_USER_ID = "user_id";
    private static final String CLAIM_KEY_AUTHORITIES = "scope";
    private static final String CLAIM_KEY_ACCOUNT_ENABLED = "enabled";
    private static final String CLAIM_KEY_ACCOUNT_NON_LOCKED = "non_locked";
    private static final String CLAIM_KEY_ACCOUNT_NON_EXPIRED =
"non_expired";

    @Value("${jwt.secret}")
    private String secret;

    @Value("${jwt.access_token.expiration}")
    private Long access_token_expiration;

    @Value("${jwt.refresh_token.expiration}")
    private Long refresh_token_expiration;
```

```
    public JWTUserDetails getUserFromToken(String token) {
        JWTUserDetails user;
        try {
            final Claims claims = getClaimsFromToken(token);
            long userId = getUserIdFromToken(token);
            String username = claims.getSubject();
            List roles = (List) claims.get(CLAIM_KEY_AUTHORITIES);
            Collection<? extends GrantedAuthority> authorities = parseArrayTo
Authorities(roles);
            boolean account_enabled = (Boolean) claims.get(CLAIM_KEY_ACCOUNT_
ENABLED);
            boolean account_non_locked = (Boolean) claims.get(CLAIM_KEY_ACCOU
NT_NON_LOCKED);
            boolean account_non_expired = (Boolean) claims.get(CLAIM_KEY_ACCO
UNT_NON_EXPIRED);

            user = new JWTUserDetails(userId, username, "password", account_e
nabled, account_non_expired, true, account_non_locked, authorities);
        } catch (Exception e) {
            user = null;
        }
        return user;
    }

    public long getUserIdFromToken(String token) {
        long userId;
        try {
            final Claims claims = getClaimsFromToken(token);
            userId = (Long) claims.get(CLAIM_KEY_USER_ID);
        } catch (Exception e) {
            userId = 0;
```

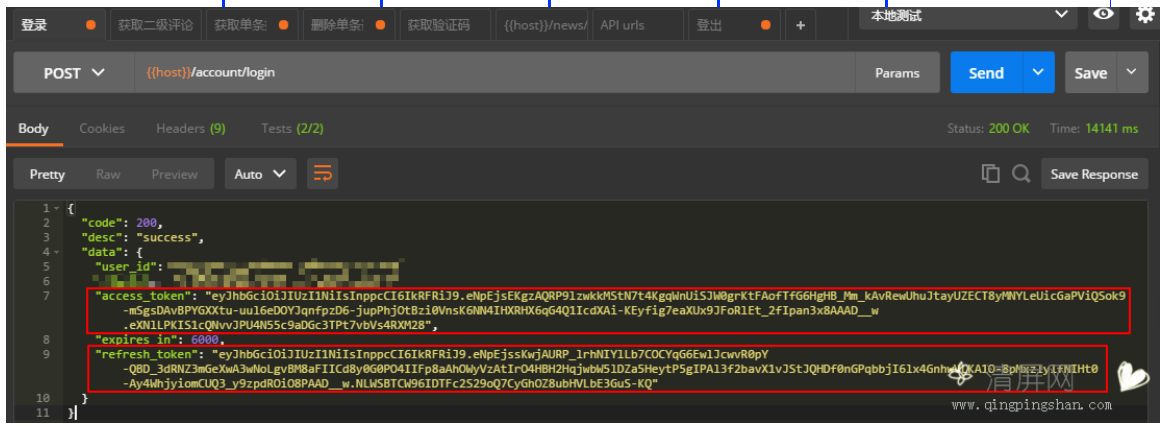
```
}

public String getUsernameFromToken(String token) {
    String username;
    try {
        final Claims claims = getClaimsFromToken(token);
        username = claims.getSubject();
    } catch (Exception e) {
        username = null;
    }
    return username;
}

public Date getCreatedDateFromToken(String token) {
    Date created;
    try {
        final Claims claims = getClaimsFromToken(token);
        created = claims.getIssuedAt();
    } catch (Exception e) {
        created = null;
    }
    return created;
}

public Date getExpirationDateFromToken(String token) {
    Date expiration;
    try {
        final Claims claims = getClaimsFromToken(token);
        expiration = claims.getExpiration();
    } catch (Exception e) {
        expiration = null;
    }
```

```
    private Claims getClaimsFromToken(String token) {
        Claims claims;
        try {
            claims = Jwts.parser()
                    .setSigningKey(secret)
                    .parseClaimsJws(token)
                    .getBody();
        } catch (Exception e) {
            claims = null;
        }
        return claims;
    }

    private Date generateExpirationDate(long expiration) {
        return new Date(System.currentTimeMillis() + expiration * 1000);
    }

    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }

    private Boolean isCreatedBeforeLastPasswordReset(Date created, Date lastP
asswordReset) {
        return (lastPasswordReset != null &&
created.before(lastPasswordReset));
    }

    public String generateAccessToken(UserDetails userDetails) {
        JWTUserDetails user = (JWTUserDetails) userDetails;
        Map<String, Object> claims = generateClaims(user);
```

```
        return generateAccessToken(user.getUsername(), claims);
    }

    private Map<String, Object> generateClaims(JWTUserDetails user) {
        Map<String, Object> claims = new HashMap<>();
        claims.put(CLAIM_KEY_USER_ID, user.getUserId());
        claims.put(CLAIM_KEY_ACCOUNT_ENABLED, user.isEnabled());
        claims.put(CLAIM_KEY_ACCOUNT_NON_LOCKED, user.isAccountNonLocked());
        claims.put(CLAIM_KEY_ACCOUNT_NON_EXPIRED,
user.isAccountNonExpired());
        return claims;
    }

    private String generateAccessToken(String subject, Map<String, Object> cl
aims) {
        return generateToken(subject, claims, access_token_expiration);
    }

    private List authoritiesToArray(Collection<? extends GrantedAuthority> au
thorities) {
        List<String> list = new ArrayList<>();
        for (GrantedAuthority ga : authorities) {
            list.add(ga.getAuthority());
        }
        return list;
    }

    private Collection<? extends GrantedAuthority> parseArrayToAuthorities(Li
st roles) {
        Collection<GrantedAuthority> authorities = new ArrayList<>();
        SimpleGrantedAuthority authority;
        for (Object role : roles) {
```

```
        }
        return authorities;
    }

    public String generateRefreshToken(UserDetails userDetails) {
        JWTUserDetails user = (JWTUserDetails) userDetails;
        Map<String, Object> claims = generateClaims(user);
        // 只授于更新 token 的权限
        String roles[] = new String[]{JWTUtils.ROLE_REFRESH_TOKEN};
        claims.put(CLAIM_KEY_AUTHORITIES, JSON.toJSON(roles));
        return generateRefreshToken(user.getUsername(), claims);
    }

    private String generateRefreshToken(String subject, Map<String, Object> c
laims) {
        return generateToken(subject, claims, refresh_token_expiration);
    }

    public Boolean canTokenBeRefreshed(String token, Date lastPasswordReset)
{
        final Date created = getCreatedDateFromToken(token);
        return !isCreatedBeforeLastPasswordReset(created, lastPasswordReset)
                && (!isTokenExpired(token));
    }

    public String refreshToken(String token) {
        String refreshedToken;
        try {
            final Claims claims = getClaimsFromToken(token);
            refreshedToken = generateAccessToken(claims.getSubject(),
claims);
        } catch (Exception e) {
```

```
        return refreshedToken;
    }

    private String generateToken(String subject, Map<String, Object> claims,
long expiration) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
                .setId(UUID.randomUUID().toString())
                .setIssuedAt(new Date())
                .setExpiration(generateExpirationDate(expiration))
                .compressWith(CompressionCodecs.DEFLATE)
                .signWith(SIGNATURE_ALGORITHM, secret)
                .compact();
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        JWTUserDetails user = (JWTUserDetails) userDetails;
        final long userId = getUserIdFromToken(token);
        final String username = getUsernameFromToken(token);
        // final Date created = getCreatedDateFromToken(token);
        // final Date expiration = getExpirationDateFromToken(token);
        return (userId == user.getUserId()
                && username.equals(user.getUsername())
                && !isTokenExpired(token)
                /* && !isCreatedBeforeLastPasswordReset(created, userDetails.
getLastPasswordResetDate()) */
        );
    }

}
```



测试

利用 **Postman** 测试一下接口，为了后续使用其他接口，这里不像 cookie 在 Postman 设置中默认会自动带上，而我们基于 JWT 后，需要每次设置到 **header** 中，这样每次手动设置（想砸电脑的心都有了），这里推荐个小技巧，使用 Postman 自带的功能实现请求成功后自动帮我们设置一个全局的变量，然后在其他接口就能引用了，再也不用一个个 **Ctrl + C** 、 **Ctrl + V** 了，是不是很棒棒呢

skill

cookie 的方式了，仿佛看到新大陆。

refresh token

细心的人可能已经发现上图中的 API 请求返回了一个 `refresh_token` ，这玩意有啥子用？？可以参考下 *微信的开放文档* 里面有提及到这个思想。

刷新access_token有效期

access_token是调用授权关系接口的调用凭证，由于access_token有效期（目前为2个小时）较短，当access_token超时后，可以使用refresh_token进行刷新，access_token刷新结果有两种：

1. 若access_token已超时，那么进行refresh_token会获取一个新的access_token，新的超时时间；
2. 若access_token未超时，那么进行refresh_token不会改变access_token，但超时时间会刷新，相当于续期access_token。

refresh_token拥有较长的有效期（30天），当refresh_token失效的后，需要用户重新授权。

请求方法

获取第一步的code后，请求以下链接进行refresh_token：

https://api.weixin.qq.com/sns/oauth2/refresh_token?appid=APPID&grant_type=refresh_token&refresh_token=REFRESH_TOKEN

参数说明

参数	是否必须	说明
appid	是	应用唯一标识
grant_type	是	填refresh_token
refresh_token	是	填写通过access_token获取到的refresh_token参数

返回说明

正确的返回：

```
{
    "access_token":"ACCESS_TOKEN",
    "expires_in":7200,
    "refresh_token":"REFRESH_TOKEN",
    "openid":"OPENID",
    "scope":"SCOPE"
}
```

参数	说明
access_token	接口调用凭证
expires_in	access_token接口调用凭证超时时间，单位（秒）
refresh_token	用户刷新access_token
openid	授权用户唯一标识
scope	用户授权的作用域，使用逗号（,）分隔

```
{"errcode":40030,"errmsg":"invalid refresh_token"}
```

检验授权凭证（access_token）是否有效

请求说明

http请求方式: GET ***https://api.weixin.qq.com/sns/auth?access_token=ACCESS_TOKEN&openid=OPENID***

参数说明	参数	是否必须
access_token	是	调用接口凭证
openid	是	普通用户标识，对该公众帐号唯一 返回说明

说明

正确的 Json 返回结果： { "errcode":0, "errmsg":"ok" }

错误的 Json 返回示例： { "errcode":40003, "errmsg":"invalid openid" }

就是在 access_token 过期或还没过期的情况下刷新 access_token，而不用使用帐号密码进行登录获取了。

刷新 access_token 的权限

能不能使用 refresh_token 来代替 access_token 呢？所以问题就来，我们要在业务中加入只有 使用 **refresh_token** 进行刷新。细心的你可能已经发现了这两个 token 的 Payload（内容） 中有 **scope** 这个字段，对，这个就是当前用户的所拥有的权限

access_token	refresh_token
"scope": ["ROLE_USER"]	**"scope": ["ROLE_REFRESH_TOKEN"]**

我们生成 **refresh_token** 时，只赋予 **ROLE_REFRESH_TOKEN** 权限，那就各施其职了。

现在权限也相应赋予了，但我们使用 refresh_token 请求时都是返回 403

403

这是因为我们要配置文件配置了所有接口都要有 **ROLE_USER** 这个权限才能访问，所以识别不了 **ROLE_REFRESH_TOKEN** 这个权限，这时我们在配置文件上加上

```
<intercept-url pattern="/account/auth/refresh_token" access="ROLE_REFRESH_TOK
EN" />
```

这样即可正常使用 refresh_token 进行访问了。而我们使用 access_token 访问时就出现 403，因为它只有 ROLE_USER 权限，但是我们现在是面向移动 APP 的后台服务，这样不友好，那咋办呢?

我目前是这么干的，给 **/account/auth/refresh_token** 这个路径设置两个访问权限

```
<intercept-url pattern="/account/auth/refresh_token" access="ROLE_USER,ROLE_R
EFRESH_TOKEN" />
```

然后使用方法级别的权限校验，如果没有这个权限框架就会抛出 **AccessDeniedException** ，然后我们在 统一异常处理 处识别与返回 json 格式的数据了

```
@GetMapping("/account/auth/refresh_token")
@PreAuthorize("hasRole('" + JWTUtils.ROLE_REFRESH_TOKEN + "')")
public String refreshAuthToken(...) {
    ...
}
```

注意

由于我们使用 spring security 的注解，而且是在控制层 **controller** 中，所以我们还要打开 spring security 对注解的支持（目前有三种不同的注解，要分别打开），这里涉及到一个细节的问题，要在 controller 层与只在 service 层使用配置的方式不一样

了。我们这里就要使用第一种了，配置如下：

spring-security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ···
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="
       http://www.springframework.org/schema/security
       http://www.springframework.org/schema/security/spring-security.xsd
       ···">

    <security:global-method-security pre-post-annotations="enabled" proxy-target-class="true"/>

    ···

</beans>
```

这里要引入 spring-security 相应的 头文件 。

还搞不清Spring 与 Spring MVC 容器之间的关系?

到此，我们就完成了，只能使用 **ROLE_REFRESH_TOKEN** 权限才能访问了，不拥有这个权限的就会返回权限不足。

参阅:

但是目前这样也有不足，就是

1. 只能是在这个接口使用 **ROLE_REFRESH_TOKEN** 是正常访问， **ROLE_USER** 能返回 json 提示。
2. 当使用 **ROLE_REFRESH_TOKEN** 权限访问其他接口时就是返回网页版的 403，而不是抛出权限不足的异常，从而 统一异常处理 也处理不了。

提出问题

如果你知道怎么在没有对应权限时能统一返回 json 数据，而不是 403 网页版的，欢迎在下方留言或私聊我，在这先谢谢你！

本文标签： Spring，JSON

相关文章

springboot读取配置文件（application.yml...	04.18	SpringBoot如何加载properties和yml配置文..	04.18
SpringData JPA 简单查询：方法定义规则	04.18	SpringBoot与Mybatis实现SpringMVC We...	04.18
SpringMVC注解版前台向后台传值的两种方...	04.18	springboot图片上传与显示功能实例详解	04.18
springboot与redis实现session共享步骤详解	04.18	spring与mybatis三种整合方法详细教程	04.18
Spring Data JPA调用存储过程实例代码实例	04.18	Spring+SpringMVC+Mybatis高性能web构..	04.18

发表评论

来说两句吧...

还没有评论，快来抢沙发吧！

畅言