链滴

关于 JWT Token 自动续期的解决方案

作者：jianzh5

原文链接：https://ld246.com/article/1594861530113

来源网站：链滴

前言

在前后端分离的开发模式下，前端用户登录成功后后端服务会给用户颁发一个jwt token。前端(如vue在接收到jwt token后会将token存储到LocalStorage中。

后续每次请求都会将此token放在请求头中传递到后端服务，后端服务会有一个过滤器对token进行截校验，校验token是否过期，如果token过期则会让前端跳转到登录页面重新登录。

因为jwt token中一般会包含用户的基础信息，为了保证token的安全性，一般会将token的过期时间置的比较短。

但是这样又会导致前端用户需要频繁登录（token过期），甚至有的表单比较复杂，前端用户在填写单时需要思考较长时间，等真正提交表单时后端校验发现token过期失效了不得不跳转到登录页面。

如果真发生了这种情况前端用户肯定是要骂人的，用户体验非常不友好。本篇内容就是在前端用户无知的情况下实现token的自动续期，避免频繁登录、表单填写内容丢失情况的发生。

实现原理

jwt token自动续期的实现原理如下：

1. 登录成功后将用户生成的 jwt token 作为key、value存储到cache缓存里面 (这时候key、value一样)，将缓存有效期设置为 token有效时间的2倍。

2. 当该用户再次请求时，通过后端的一个 jwt Filter 校验**前端token**是否是有效token，如果token效表明是非法请求，直接抛出异常即可；

3. 根据规则取出cache token，判断cache token是否存在，此时主要分以下几种情况：

- cache token 不存在

这种情况表明该用户账户空闲超时，返回用户信息已失效，请重新登录。

- cache token 存在，则需要 **使用jwt工具类验证该cache token 是否过期超时**，不过期无需处。

过期则表示该用户一直在操作只是token失效了，后端程序会给token对应的key映射的value值重新成jwt token并覆盖value值，该缓存生命周期重新计算。

实现逻辑的核心原理：

前端请求Header中设置的token保持不变，校验有效性以缓存中的token为准。

代码实现（伪码）

1. 登录成功后给用户签发token，并设置token的有效期

```
...
SysUser sysUser = userService.getUser(username,password);
if(null !== sysUser){
    String token = JwtUtil.sign(sysUser.getUsername(),
sysUser.getPassword());
}
...
```

```
public static String sign(String username, String secret) {
    //设置token有效期为30分钟
    Date date = new Date(System.currentTimeMillis() + 30 * 60 * 1000);
    //使用HS256生成token,密钥则是用户的密码
    Algorithm algorithm = Algorithm.HMAC256(secret);
    // 附带username信息
    return JWT.create().withClaim("username", username).withExpiresAt(date).sign(algorithm);
}
```

2. 将token存入redis，并设定过期时间，将redis的过期时间设置成token过期时间的两倍

```
Sting tokenKey = "sys:user:token" + token;
redisUtil.set(tokenKey, token);
redisUtil.expire(tokenKey, 30 * 60 * 2);
```

3. 过滤器校验token，校验token有效性

```
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOExce
tion, ServletException {
    //从header中获取token
    String token = httpServletRequest.getHeader("token")
    if(null == token){
        throw new RuntimeException("illegal request，token is necessary!")
    }
    //解析token获取用户名
    String username = JwtUtil.getUsername(token);
    //根据用户名获取用户实体，在实际开发中从redis取
    User user = userService.findByUser(username);
    if(null == user){
        throw new RuntimeException("illegal request，token is Invalid!")
    }
    //校验token是否失效，自动续期
    if(!refreshToken(token,username,user.getPassword())){
        throw new RuntimeException("illegal request，token is expired!")
    }
    ...
}
```

4. 实现token的自动续期

```
public boolean refreshToken(String token, String userName, String passWord) {
    Sting tokenKey = "sys:user:token" + token ;
    String cacheToken = String.valueOf(redisUtil.get(tokenKey));
    if (StringUtils.isNotEmpty(cacheToken)) {
        // 校验token有效性，注意需要校验的是缓存中的token
        if (!JwtUtil.verify(cacheToken, userName, passWord)) {
            String newToken = JwtUtil.sign(userName, passWord);
            // 设置超时时间
            redisUtil.set(tokenKey, newToken) ;
            redisUtil.expire(tokenKey, 30 * 60 * 2);
        }
        return true;
    }
```

原文链接：关于 JWT Token 自动续期的解决方案

```
    return false;
}
...

public static boolean verify(String token, String username, String secret) {
    try {
        // 根据密码生成JWT效验器
        Algorithm algorithm = Algorithm.HMAC256(secret);
        JWTVerifier verifier = JWT.require(algorithm).withClaim("username", username).build();
        // 效验TOKEN
        DecodedJWT jwt = verifier.verify(token);
        return true;
    } catch (Exception exception) {
        return false;
    }
}
```

本文中jwt的相关操作是基于 com.auth0.java-jwt 实现，大家可以通过阅读原文获取 JWTUtil 工具类。

小结

jwt token实现逻辑的核心原理是 **前端请求Header中设置的token保持不变，校验有效性以缓存中的oken为准，千万不要直接校验Header中的token。**实现原理部分大家好好体会一下，思路比实现更要！

JwtUtil

```
public class JwtUtil {
    // Token过期时间30分钟（用户登录过期时间是此时间的两倍，以token在reids缓存时间为准）
    public static final long EXPIRE_TIME = 30 * 60 * 1000;

    /**
     * 校验token是否正确
     * @param token  密钥
     * @param secret 用户的密码
     * @return 是否正确
     */
    public static boolean verify(String token, String username, String secret) {
        try {
            // 根据密码生成JWT效验器
            Algorithm algorithm = Algorithm.HMAC256(secret);
            JWTVerifier verifier = JWT.require(algorithm).withClaim("username", username).build();
            // 效验TOKEN
            DecodedJWT jwt = verifier.verify(token);
            return true;
        } catch (Exception exception) {
            return false;
        }
    }

    /**
     * 获得token中的信息无需secret解密也能获得
     * @return token中包含的用户名
```

```
     */
    public static String getUsername(String token) {
        try {
            DecodedJWT jwt = JWT.decode(token);
            return jwt.getClaim("username").asString();
        } catch (JWTDecodeException e) {
            return null;
        }
    }

    /**
     * 生成签名,30min后过期
     */
    public static String sign(String username, String secret) {
        Date date = new Date(System.currentTimeMillis() + EXPIRE_TIME);
        //使用HS256生成token,密钥则是用户的密码
        Algorithm algorithm = Algorithm.HMAC256(secret);
        // 附带username信息
        return JWT.create().withClaim("username", username).withExpiresAt(date).sign(algorithm)

    }

    /**
     * 根据request中的token获取用户账号
     * @param request
     * @return
     */
    public static String getUserNameByToken(HttpServletRequest request) {
        String accessToken = request.getHeader("X-Access-Token");
        String username = getUsername(accessToken);
        if (StringUtils.isEmpty(username)) {
            throw new RuntimeException("无法获取有效用户！");
        }
        return username;
    }

}
```