我不应该用JWT的！

一、前言

大家好呀，我是summo，之前有自学过Shrio框架，网上一搜就有SpringBoot整合Shrio+ JWT的文章，我是在学习Shrio框架的时候顺带学的JWT。后来我还看见有很多博主专门写文章介绍JWT，说这个东西的优点很多，安全性好、去中心化、方便啥的，我就把JWT也应用在我们自己的系统中了。但最近发现这玩意越来越让我觉得别扭，总感觉哪里不太对劲，重新审查我的登录认证逻辑之后才发现：我不应该用JWT的！

这里我用一句解释不该用的原因，省得浪费大家的时间：`我的系统有Redis，而且还用Redis存了JWT，随着系统升级，JWT越来越像普通Token！`明白原理的同学可能心中暗笑，直接跳过看下一篇了，不明白原理的同学，可以看看这个四不像是怎么被我搭出来的。

二、JWT是什么?

看我文章的有很多大神，也有一些小白，所以为了不让小白们看的云里雾里，我还是有必要介绍一些基本原理。JWT是 JSON Web Token 的缩写，可以对JSON对象进行编码(加密)，并通过这个编码传递信息。

1. JWT的结构

（1）头部（Header）

> 头部通常由两部分组成，即令牌的类型（typ）和所使用的算法（alg）。例如，一个头部可能是 {”alg”: ”HS256”, ”typ”: ”JWT”}，表示使用 HMAC SHA-256 算法对令牌进行签名。

（2）载荷（Payload）

> 载荷包含了 JWT 的声明信息，用于描述令牌的相关内容。载荷可以包含标准声明（例如：发行者、主题、过期时间等），也可以包含自定义声明。例如，一个载荷可能是 {”sub”: ”1234567890”, ”name”: ”John Doe”, ”exp”: 1516239022}。

（3）签名（Signature）

> 签名用于验证令牌的完整性和真实性。签名通常由头部、载荷和密钥一起计算而得。验证者可以使用相同的密钥重新计算签名，并将结果与令牌中的签名进行比较，以确认令牌的真实性。

2. SpringBoot使用JWT

（1）maven引入

```
<!-- jwt -->
<dependency>
  <groupId>com.auth0</groupId>
  <artifactId>java-jwt</artifactId>
  <version>3.8.2</version>
</dependency>
```

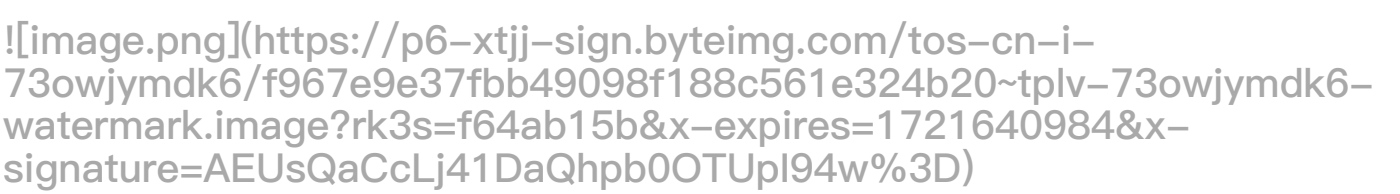
（2）代码示例

```
import java.util.Date;

import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTDecodeException;
import com.auth0.jwt.interfaces.DecodedJWT;
```

```
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;

@Slf4j
public class JWTUtil {
    /**
     * 过期时间
     */
    private static final long EXPIRE_TIME = 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 {
            Algorithm algorithm = Algorithm.HMAC256(secret);
            JWTVerifier verifier = JWT.require(algorithm)
                .withClaim("username", username)
                .build();
            verifier.verify(token);
            log.info("token is valid");
            return true;
        } catch (Exception e) {
            log.info("token is invalid{}", e.getMessage());
            return false;
        }
    }

    /**
     * 从 token中获取用户名
     *
     * @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) {
            log.error("error：{}", e.getMessage());
            return null;
        }
    }
```

```
    /**
     * 生成 token
     *
     * @param username 用户名
     * @param secret   用户的密码
     * @return token
     */
    public static String sign(String username, String secret) {
        try {
            username = StringUtils.lowerCase(username);
            Date date = new Date(System.currentTimeMillis() +
EXPIRE_TIME);
            Algorithm algorithm = Algorithm.HMAC256(secret);
            return JWT.create()
                .withClaim("username", username)
                .withExpiresAt(date)
                .sign(algorithm);
        } catch (Exception e) {
            log.error("error：{}", e);
            return null;
        }
    }

    public static void main(String[] args) {
        //对数据进行加密
        String token = sign("zhangshan", "123456");
        System.out.println(token);
        //对数据进行解密
        System.out.println(getUsername(token));
    }
}
```

运行一下



> JWT还是很简单的，一学就会，很多博主在介绍它的时候都会说它安全、方便、去中心化等等，然后强烈推荐大家使用。但这里我要就要给大家泼冷水了，学是肯定要学的，用就需要看情况了，不能别人说它好，你就无脑用，然后用成一个四不像。至于我为什么说我用来是四不像，接着看！

三、JWT vs Token+Redis

在设计no session系统时，有两种可选方案：JWT与Token+Redis。

1. 原理简介

* JWT: 生成并发给客户端之后，后台是不用存储，客户端访问时会验证其签名、过期时间等再取出里面的信息（如username），再使用该信息直接查询用户信息完成登录验证。jwt自带签名、过期等校验，后台不用存储，缺陷是一旦下发，服务后台无法拒绝携带该jwt的请求（如踢除用户）；
* Token+Redis： 是自己生成个32位的key，value为用户信息，访问时判断redis里是否有该token，如果有，则加载该用户信息完成登录。服务需要存储下发的每个token及对应的value，维持其过期时间，好处是随时可以删除某个token，阻断该token继续使用。

2. 两种方案的优缺点

（1）去中心化的JWT

优点:

1. 去中心化，便于分布式系统使用
2. 基本信息可以直接放在token中。 username，nickname，role
3. 功能权限较少的话，可以直接放在token中。用bit位表示用户所具有的功能权限

缺点:

1. 服务端不能主动让token失效

（2）中心化的Redis+Token

优点：

1. 服务端可以主动让token失效

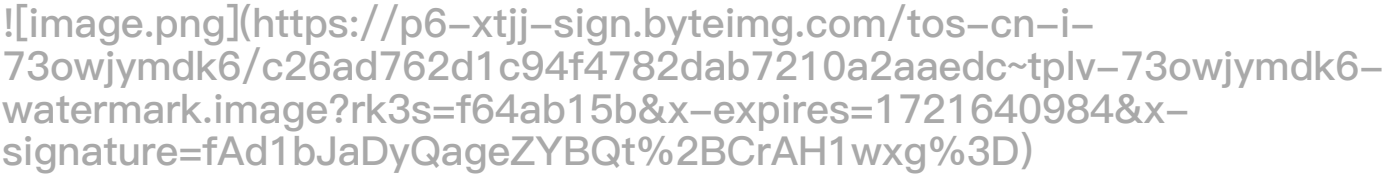
缺点：

1. 依赖内存或redis存储。
2. 分布式系统的话，需要redis调用增加了系统复杂性。

> 光看优缺点的话，JWT优点还比Redis+Token多，在小白时期的我一看：好家伙，JWT这么多优点，用它准没错，看来简历上又可以加上一笔了！

四、我的方案
======

想法是美好的，但现实是残酷的，为什么我会觉得越来越别扭，先上一张流程图，让大家看看我在业务中是怎么做的，如下：



> 上面的方案是JWT或者Redis+Token，而我的方案是JWT+Redis。和普通的流程相比，我还加了一个加解密流程，因为我觉得JWT数据格式太明显了，一眼就知道是用的是什么认证方式，容易被篡改。你说这个一点用都没有吗？好像还有那么点用，最起码JWT变得更安全了...

1. 别扭原因

（1）多余的加解密流程

> 虽然给JWT加一下密提高了安全性，但是导致JWT的自带的过期机制失效了

，必须得加上Redis的缓存失效机制，在安全和方便的选项中我选择了安全。

（2）系统对用户的操作频繁

> 因为我们的系统是一个传统的管理端+C端模式，管理员经常给用户增删权限，刚好命中“服务端不能主动让token失效”这一缺陷，这是最大的原因。

（3）JWT随着系统的升级字符越来越长

> JWT存储的信息比较少的时候，还只有一两百个字符，但是字段一多，直接变成“小作文”，我现在看着这一段长长的token，头都大。

（4）单点登录使用JWT太不可控了

> 实现不了单点登录之后的单点退出功能，即用户在一个系统登出，所有相关系统也自动登出，JWT完全不能满足这个需求。

2. 总结一下

写这篇文章我想肯定会有很多人不服，说我不会用就说人家不好用。诚然，我确实不太会用，不然我也不会用JWT到我的系统中来。我也看了很多介绍JWT的文章，都是说原理和优点，很少有说它的应用场景，知乎上有一篇关于[jwt与token+redis，哪种方案更好用？](http://cxyroad.com/”https://www.zhihu.com/question/274566992”)的辩论，很激烈，感兴趣的同学可以去看看。

这个帖子里面的大部分人都认为JWT不是一个必要的组件，甚至有人说`任何时候都不应该首要考虑JWT`，现在的我深感赞同。里面还有不少老哥推荐使用折中方案，也就是上面我的方案：JWT+Redis，不过我总感觉这种用法很别扭，因为一旦上了Redis那和Redis+Token方案还有什么不同，而且JWT还长的多，白白浪费用户的流量。

里面有个老哥的比喻我觉得非常形象，给大家看下

我觉得这个老哥讲的还挺有道理，JWT本身也只是一个token(有点长的token)，非要让它满足所有的需求属实是难为它了。但是如果仅仅只是创造一个新品，如意大利面，就开始疯狂吹嘘它无所不能，给小白一些错误的引导，那就不对了，毕竟喜欢吃意大利面的人也不多。

原文链接: https://juejin.cn/post/7391699424843710515