1 base64 编码

1.1 什么是 Base64

所谓 Base64，就是说选出 64 个字符：小写字母 a-z、大写字母 A-Z、数字 0-9、符号"+"、"/"（再加上作为垫字的"="，实际上是使用 65 个字符），作为一个基本字符集。然后，其他所有符号都转换成这个字符集中的字符。

1.2 linux base64 命令

1.2.1 Linux 下用 base64 命令编解码字符串

编码：

```
echo -n 'Hello World' | base64

SGVsbG8gV29ybGQ=
```

解码：

```
echo -n 'SGVsbG8gV29ybGQ=' | base64 -d

Hello World
```

备注：

- echo 命令是带换行符的
- echo -n 不换行输出
- echo -n '{"alg":"HS256","typ":"JWT"}' | base64

1.2.2 base64 编解码文件

```
#base64 编码
# base64 待编码的文件名 > 编码后的文件名
base64   1.mp3 > mymp3
#base64 解码
#base64 -d 待解码的文件名 >解码后的文件名
base64 -d mymp3>88.mp3
```

1.3 Base64 和 Base64Url 的区别

Base64Url 是一种在 Base64 的基础上编码形成新的编码方式，为了编码能在网络中安全

顺畅传输，需要对 Base64 进行的编码，特别是互联网中。

Base64Url 编码的流程：

1、明文使用 BASE64 进行编码

2、在 Base64 编码的基础上进行以下的处理：

1)去除尾部的"="

2)把"+"替换成"-"

3)斜线"/"替换成下划线"_"

2 跨域认证问题和 JWT 实现登录原理图

2.1 跨域认证问题

互联网服务离不开用户认证。一般流程是下面这样。

- 用户向服务器发送用户名和密码。
- 服务器验证通过后，在当前对话（session）里面保存相关数据，比如用户角色、登录时间等等。
- 服务器向用户返回一个 jsession_id，写入用户的 Cookie。
- 用户随后的每一次请求，都会通过 Cookie，将 session_id 传回服务器。
- 服务器收到 session_id，找到前期保存的数据，由此得知用户的身份。

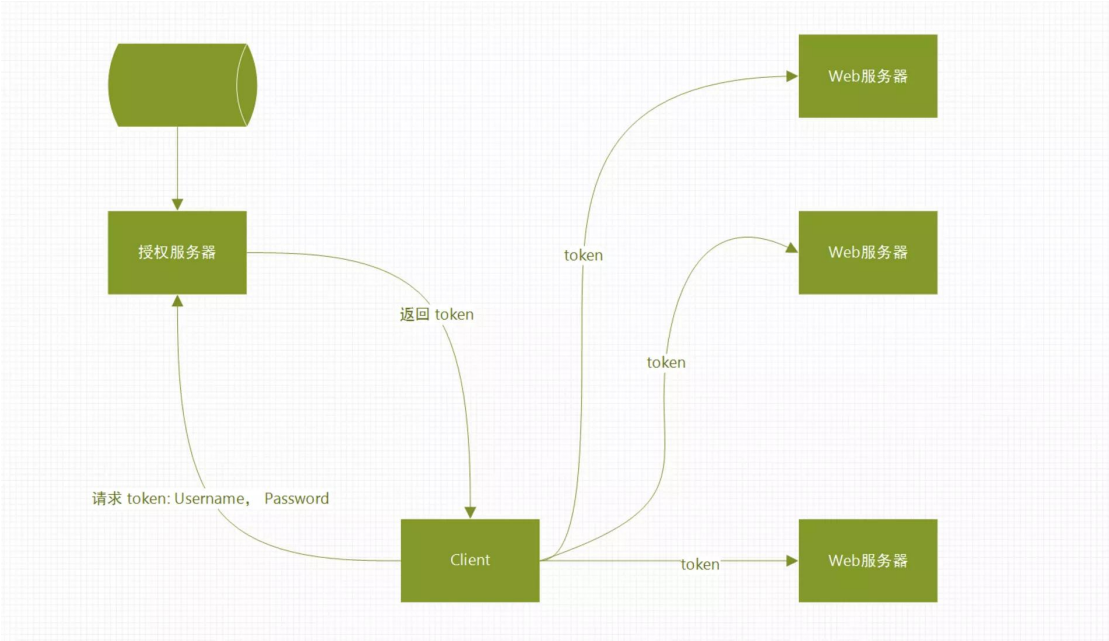
这种模式的问题在于，扩展性（scaling）不好。单机当然没有问题，如果是服务器集群，或者是跨域的服务导向架构，就要求 session 数据共享，每台服务器都能够读取 session。

举例来说，A 网站和 B 网站是同一家公司的关联服务。现在要求，用户只要在其中一个网站登录，再访问另一个网站就会自动登录，请问怎么实现？

一种解决方案是 session 数据持久化，写入数据库或别的持久层。各种服务收到请求后，都向持久层请求数据。这种方案的优点是架构清晰，缺点是工程量比较大。另外，持久层万一挂了，就会单点失败。

另一种方案是服务器索性不保存 session 数据了，所有数据都保存在客户端，每次请求都发回服务器。JWT 就是这种方案的一个代表。 服务器不存数据，客户端存，服务器解析就行了

2.2 JWT 实现登录原理图



说明：

JWT 只通过算法实现对 Token 合法性的验证，不依赖数据库，Memcached 的等存储系统，因此可以做到跨服务器验证，只要密钥和算法相同，不同服务器程序生成的 Token 可以互相验证。

3 JWT 学习

3.1 简介

JSON Web Token（JWT）是一个开放标准（RFC 7519），它定义了一种紧凑且独立的方式，用于在各方之间作为 JSON 对象安全地传输信息。 此信息可以通过数字签名进行验证和信任。 JWT 可以使用密钥（使用 HMAC 算法）或使用 RSA 或 ECDSA 的公钥/私钥对进行签名。

- 官方网址：https://jwt.io/
- 调试页面：https://jwt.io/
- 学习文档：https://jwt.io/introduction/

3.2 用途

- 授权：这是我们使用 JWT 最广泛的应用场景。一次用户登录，后续请求将会包含 JWT，对于那些合法的 token，允许用户连接路由，服务和资源。目前 JWT 广泛应用在 SSO（Single

Sign On）(单点登录)上。因为他们开销很小并且可以在不同领域轻松使用。

- 信息交换：JSON Web Token 是一种在各方面之间安全信息传输的好的方式 因为 JWT 可以签名 - 例如，使用公钥/私钥对 - 您可以确定发件人是他们所说的人。 此外，由于使用标头和有效负载计算签名，您还可以验证内容是否未被篡改。

3.3 JWT 组成

一个 JWT 由三部分组成，各部分以点分隔：

```
Header(头部）-----base64Url 编码的 Json 字符串
Payload(载荷）---base64url 编码的 Json 字符串
Signature(签名)---使用指定算法，通过 Header 和 Playload 加盐计算的字符串
```

一个 JWT 看起来像下面这样：

```
xxxxx.yyyyy.zzzzz

下面这样：
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG
4gRG9lIiwiaWFOIjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

3.3.1 Header

此部分有两部分组成：

- 一部分是 token 的类型，目前只能是 JWT
- 另一部分是签名算法，比如 HMAC 、 SHA256 、 RSA

示例：

```
{
  "alg":"HS256",
  "typ":"JWT"
}
```

base64 编码命令：

```
echo -n '{"alg":"HS256","typ":"JWT"}' | base64
```


3.3.2 Payload

token 的第二部分是 payload（有效负载），其中包含 claims（声明）。Claims 是关于一个实体（通常是用户）和其他数据类型的声明。

claims 有三种类型：registered,public,and private claims。

- Registered（已注册的声明）：这些是一组预定义声明，不是强制性的，但建议使用，以提供一组有用的，可互操作的声明。 其中一些是：iss（发行人），exp（到期时间），sub（主题），aud（观众）and others。（请注意，声明名称只有三个字符，因为 JWT 意味着紧凑。）

JWT 规定了 7 个官方字段，供选用。
iss (issuer)：签发人
exp (expiration time)：过期时间
sub (subject)：主题
aud (audience)：受众
nbf (Not Before)：生效时间
iat (Issued At)：签发时间
jti (JWT ID)：编号
除了官方字段，你还可以在这个部分定义私有字段，下面就是一个例子。

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

注意，JWT 默认是不加密的，任何人都可以读到，所以不要把**秘密信息（密码，手机号等）**放在这个部分。
这个 JSON 对象也要使用 Base64URL 算法转成字符串。

- Public(公开声明)：这些可以由使用 JWT 的人随意定义。 但为避免冲突，应在 IANA JSON Web Token Registry 中定义它们，或者将其定义为包含防冲突命名空间的 URI。

- private (私人声明)：这些声明是为了在同意使用它们的各方之间共享信息而创建的，并且既不是注册声明也不是公开声明。

示例：

```
{
  "sub": "1234567890",
```



```
  "name": "John Doe",
  "admin": true
}
```

3.3.3 Signature（保证数据安全性的）

Signature 部分是对前两部分的签名，**防止数据篡改**。

首先，需要指定一个**密钥（secret）**。这个密钥只有服务器才知道，不能泄露给用户。然后，使用 Header 里面指定的签名算法（默认是 HMAC SHA256），按照下面的公式产生签名。

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

算出签名以后，把 Header、Payload、Signature 三个部分拼成一个字符串，每个部分之间用"点"（.）分隔，就可以返回给用户。

示例：

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

3.4 JWT 的使用方式【重点】

客户端收到服务器返回的 JWT，可以储存在 Cookie 里面，也可以储存在 localStorage。此后，客户端每次与服务器通信，都要带上这个 JWT。你可以把它放在 Cookie 里面自动发送，但是这样不能跨域，所以更好的做法是放在 HTTP 请求的头信息 Authorization 字段里面。

```
Authorization: Bearer jwt
```

另一种做法是，跨域的时候，JWT 就放在 POST 请求的数据体里面。

3.5 JWT 的几个特点

JWT 默认是不加密，但也是可以加密的。生成原始 Token 以后，可以用密钥再加密一次。

JWT 不加密的情况下，不能将秘密数据写入 JWT。

JWT 不仅可以用于认证，也可以用于交换信息。有效使用 JWT，可以降低服务器查询数据

库的次数。

JWT 的最大缺点是，由于服务器不保存 session 状态，因此无法在使用过程中**废止**某个 token，或者更改 token 的权限。也就是说，一旦 JWT 签发了，在到期之前就会始终有效，除非服务器部署额外的逻辑（JWT 的登出问题）。就是因为服务端无状态了

正常情况下 修改了密码后就会跳转到登录页面 ： 修改成功后清空浏览器保存的 token 了

后端怎么玩？ 因为服务端不保留 token 我用之前的 token 还是可以继续访问的

从有状态（后端也会存一个）的变成无状态的了

我们就要把它从无状态再变成有状态了

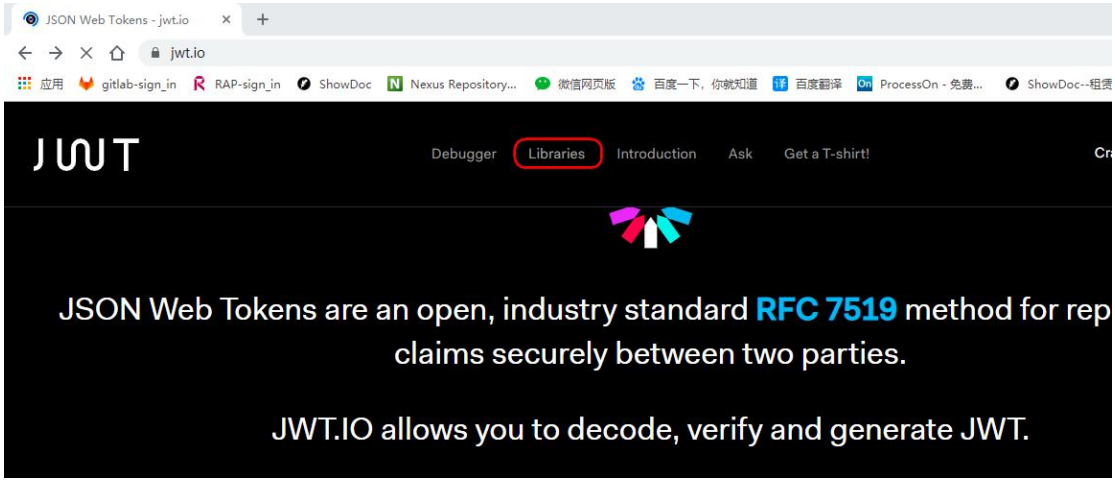
JWT 本身包含了认证信息，一旦泄露，任何人都可以获得该令牌的所有权限。为了减少盗用，JWT 的有效期应该设置得比较短。对于一些比较重要的权限，使用时应该再次对用户进行认证。

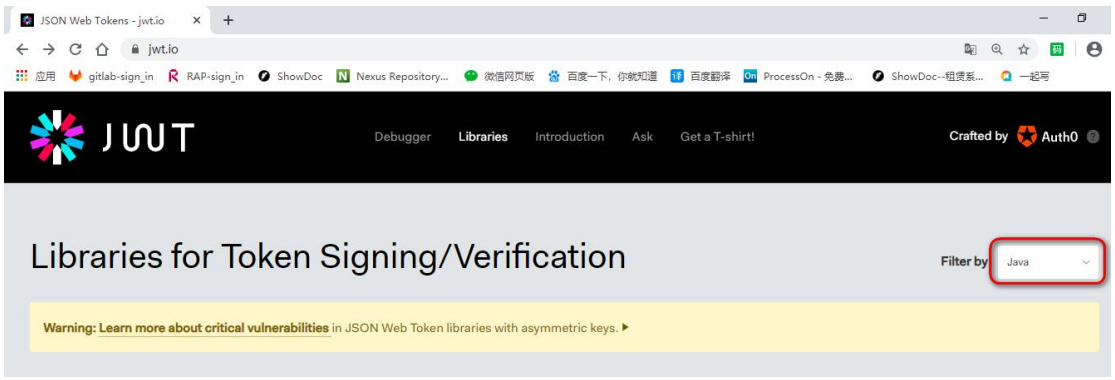
为了减少盗用，JWT 不应该使用 HTTP 80 协议明码传输，要使用 HTTPS 443 协议传输。

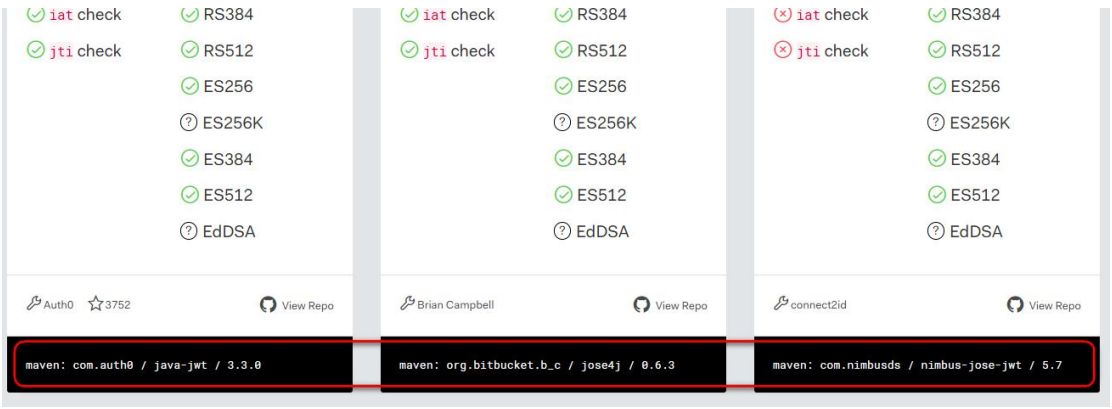
我们颁发一个令牌 用户名称 用户的权限信息 这个令牌 2 个小时有效

Jwt 只要能解析 就认为你是可用的 做不了 登出 后端不存储用户信息了 后端无状态了

4 Java 类库







5 java 中使用 jwt

5.1 新建 maven 工程 jwt-learn1

5.2 添加依赖

```
<!-- 添加 jwt 的依赖 -->
<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>3.11.0</version>
</dependency>
```

5.3 编写功能类

```
package com.powernode.utils;

import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.TokenExpiredException;
import com.auth0.jwt.interfaces.Claim;
import com.auth0.jwt.interfaces.DecodedJWT;

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 用于生成和解析 JWT
 */
public class JWTUtils {
```



```
    /**
     * 声明一个秘钥
     */
    private static final String SECRET = "leige";


    /**
     * 生成JWT
     *
     * @param userId   用户编号
     * @param username 用户名
     * @param auth     用户权限
     */
    public String createToken(Integer userId, String username,
List<String> auth) {
        //得到当前的系统时间
        Date currentDate = new Date();
        //根据当前时间计算出过期时间 定死为5分钟
        Date expTime = new Date(currentDate.getTime() + (1000 * 60 * 5));
        //组装头数据
        Map<String, Object> header = new HashMap<>();
        header.put("alg", "HS256");
        header.put("typ", "JWT");
        return JWT.create()
                .withHeader(header) //头
                .withClaim("userId", userId) //自定义数据
                .withClaim("username", username) //自定义数据
                .withClaim("auth", auth) //自定义数据
                .withIssuedAt(currentDate) //创建时间
                .withExpiresAt(expTime)//过期时间
                .sign(Algorithm.HMAC256(SECRET));
    }

    /**
     * 验证JWT并解析
     *
     * @param token 要验证的jwt的字符串
     */
    public static Boolean verifyToken(String token) {
        try{
            // 使用秘钥创建一个解析对象
            JWTVerifier
jwtVerifier=JWT.require(Algorithm.HMAC256(SECRET)).build();
            //验证JWT
            DecodedJWT decodedJWT = jwtVerifier.verify(token);
//            String header = decodedJWT.getHeader();
//            String payload = decodedJWT.getPayload();
//            String signature = decodedJWT.getSignature();
//            System.out.println("header = " + header);
//            System.out.println("payload = " + payload);
//            System.out.println("signature = " + signature);
```

```
//
//            Date expiresAt = decodedJWT.getExpiresAt();
//            System.out.println("expiresAt = " + expiresAt);
//            Claim userId = decodedJWT.getClaim("userId");
//            System.out.println("userId = " + userId.asInt());
//            Claim username = decodedJWT.getClaim("username");
//            System.out.println("username = " + username.asString());
//            Claim auth = decodedJWT.getClaim("auth");
//            System.out.println("auth = " +
auth.asList(String.class));
            return true;
        }catch (TokenExpiredException e){
            e.printStackTrace();
        }
        return false;
    }

    /**
     * 获取 JWT 里面相前的用户编号
     */
    public Integer getUserId(String token){
        try{
            // 使用秘钥创建一个解析对象
            JWTVerifier
jwtVerifier=JWT.require(Algorithm.HMAC256(SECRET)).build();
            //验证 JWT
            DecodedJWT decodedJWT = jwtVerifier.verify(token);
            Claim userId = decodedJWT.getClaim("userId");
            return userId.asInt();
        }catch (TokenExpiredException e){
            e.printStackTrace();
        }
        return null;
    }
    /**
     * 获取 JWT 里面相前的用户名
     */
    public static String getUsername(String token){
        try{
            // 使用秘钥创建一个解析对象
            JWTVerifier
jwtVerifier=JWT.require(Algorithm.HMAC256(SECRET)).build();
            //验证 JWT
            DecodedJWT decodedJWT = jwtVerifier.verify(token);
            Claim username = decodedJWT.getClaim("username");
            return username.asString();
        }catch (TokenExpiredException e){
            e.printStackTrace();
        }
        return null;
    }
    /**
     * 获取 JWT 里面相前权限
```



```
     */
    public List<String> getAuth(String token){
        try{
            // 使用秘钥创建一个解析对象
            JWTVerifier
jwtVerifier=JWT.require(Algorithm.HMAC256(SECRET)).build();
            //验证 JWT
            DecodedJWT decodedJWT = jwtVerifier.verify(token);
            Claim auth = decodedJWT.getClaim("auth");
            return auth.asList(String.class);
        }catch (TokenExpiredException e){
            e.printStackTrace();
        }
        return null;
    }

}
```

5.4 写主类测试一下

6 JWT 的总结

JWT 就是一个加密的带用户信息的字符串，没学习 JWT 之前，我们在项目中都是返回一个基本的字符串，然后请求时带上这个字符串，再从 session 或者 redis 中（共享 session）获取当前用户，学过 JWT 以后我们可以把用户信息直接放在字符串返回给前端，然后用户请求时带过来，我们是在服务器进行解析拿到当前用户，这就是两种登录方式，这两种方式有各自的优缺点。