

25-认证机制：应用程序如何进行访问认证？

你好，我是孔令飞，今天我们来聊聊如何进行访问认证。

保证应用的安全是软件开发的最基本要求，我们有多种途径来保障应用的安全，例如网络隔离、设置防火墙、设置IP黑白名单等。不过在我看来，这些更多是从运维角度来解决应用的安全问题。作为开发者，我们也可以从软件层面来保证应用的安全，这可以通过认证来实现。

这一讲，我以HTTP服务为例，来给你介绍下当前常见的四种认证方法：Basic、Digest、OAuth、Bearer。还有很多基于这四种方法的变种，这里就不再介绍了。

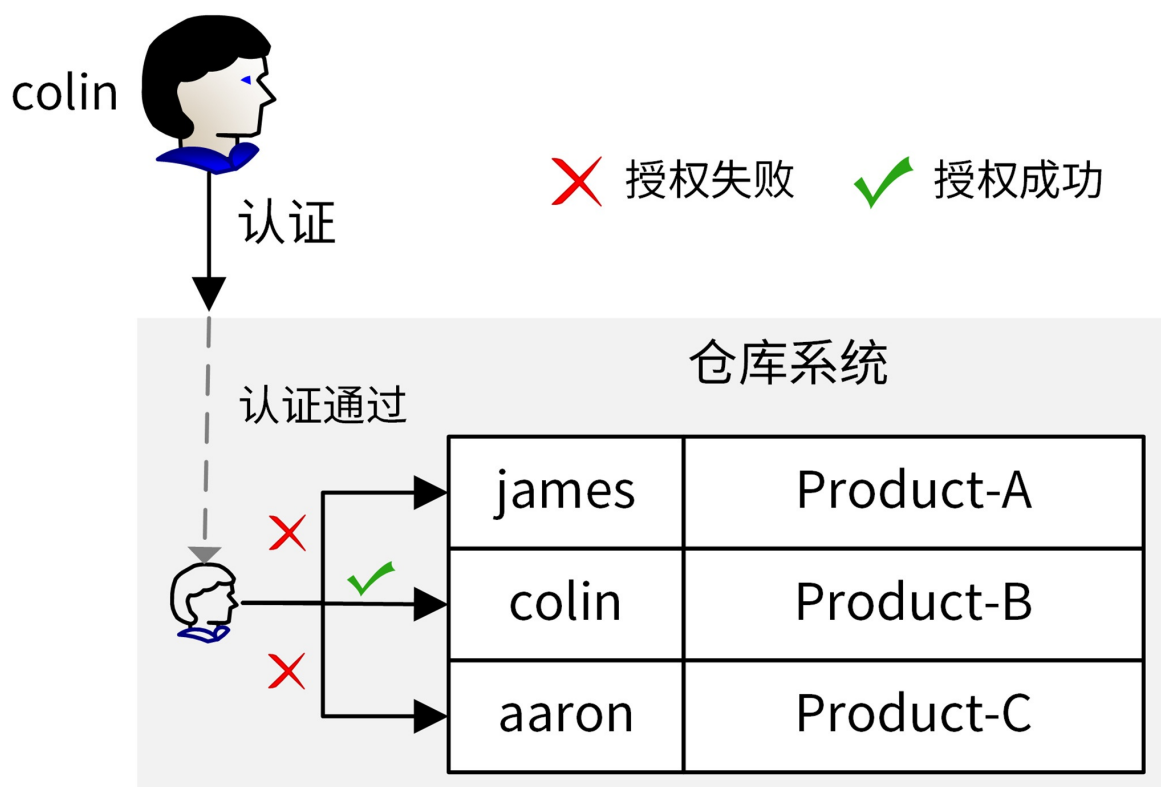
IAM项目使用了Basic、Bearer两种认证方法。这一讲，我先来介绍下这四种认证方法，下一讲，我会给你介绍下IAM项目是如何设计和实现访问认证功能的。

认证和授权有什么区别？

在介绍四种基本的认证方法之前，我想先带你区分下认证和授权，这是很多开发者都容易搞混的两个概念。

- **认证 (Authentication, 英文缩写authn)**：用来验证某个用户是否具有访问系统的权限。如果认证通过，该用户就可以访问系统，从而创建、修改、删除、查询平台支持的资源。
- **授权 (Authorization, 英文缩写authz)**：用来验证某个用户是否具有访问某个资源的权限，如果授权通过，该用户就能对资源做增删改查等操作。

这里，我通过下面的图片，来让你明白二者的区别：



图中，我们有一个仓库系统，用户 james、colin、aaron分别创建了Product-A、Product-B、Product-C。现在用户colin通过用户名和密码（认证）成功登陆到仓库系统中，但他尝试访问Product-A、Product-C失败，因为这两个产品不属于他（授权失败），但他可以成功访问自己创建的资源Product-B（授权成功）。由此可见：**认证证明了你是谁，授权决定了你能做什么。**

上面，我们介绍了认证和授权的区别。那么接下来，我们就回到这一讲的重心：应用程序如何进行访问认证。

四种基本的认证方式

常见的认证方式有四种，分别是 Basic、Digest、OAuth 和 Bearer。先看下Basic认证。

Basic

Basic认证（基础认证），是最简单的认证方式。它简单地将用户名:密码进行base64编码后，放到HTTP Authorization Header中。HTTP请求到达后端服务后，后端服务会解析出Authorization Header中的base64字符串，解码获取用户名和密码，并将用户名和密码跟数据库中记录的值进行比较，如果匹配则认证通过。例如：

```
$ basic=`echo -n 'admin:Admin@2021'|base64`  
$ curl -XPOST -H"Authorization: Basic ${basic}" http://127.0.0.1:8080/login
```

通过base64编码，可以将密码以非明文的方式传输，增加一定的安全性。但是，base64不是加密技术，入侵者仍然可以截获base64字符串，并反编码获取用户名和密码。另外，即使Basic认证中密码被加密，入侵者仍可通过加密后的用户名和密码进行重放攻击。

所以，Basic认证虽然简单，但极不安全。使用Basic认证的唯一方式就是将它和SSL配合使用，来确保整个认证过程是安全的。

IAM项目中，为了支持前端通过用户名和密码登录，仍然使用了Basic认证，但前后端使用HTTPS来通信，保证了认证的安全性。

这里需要注意，在设计系统时，要遵循一个通用的原则：**不要在请求参数中使用明文密码，也不要任何存储中保存明文密码。**

Digest

Digest认证（摘要认证），是另一种 HTTP 认证协议，它与基本认证兼容，但修复了基本认证的严重缺陷。Digest具有如下特点：

- 绝不会用明文方式在网络上发送密码。
- 可以有效防止恶意用户进行重放攻击。
- 可以有选择地防止对报文内容的篡改。

摘要认证的过程见下图：



在上图中，完成摘要认证需要下面这四步：

1. 客户端请求服务端的资源。
2. 在客户端能够证明它知道密码从而确认其身份之前，服务端认证失败，返回401 Unauthorized，并返回WWW-Authentication头，里面包含认证需要的信息。
3. 客户端根据WWW-Authentication头中的信息，选择加密算法，并使用密码随机数nonce，计算出密码摘要response，并再次请求服务端。
4. 服务器将客户端提供的密码摘要与服务器内部计算出的摘要进行对比。如果匹配，就说明客户端知道密码，认证通过，并返回一些与授权会话相关的附加信息，放在Authorization-Info中。

WWW-Authentication头中包含的信息见下表：

字段名	说明
username	用户名
password	密码
realm	服务器返回的realm，一般是域名
method	HTTP请求方法
nonce	服务器发给客户端的随机字符串
nc(nonceCount)	请求的次数，用于标记、计数，防止重放攻击
cnonce(clinetNonce)	客户端发送给服务器的随机字符串
qop	保护质量参数，一般是auth或auth-int，这会影响摘要的算法
uri	请求的uri
response	客户端根据算法算出的密码摘要值

虽然使用摘要可以避免密码以明文方式发送，一定程度上保护了密码的安全性，但是仅仅隐藏密码并不能保证请求是安全的。因为请求（包括密码摘要）仍然可以被截获，这样就可以重放给服务器，带来安全问题。

为了防止重放攻击，服务器向客户端发送了密码随机数nonce，nonce每次请求都会变化。客户端会根据nonce生成密码摘要，这种方式，可以使摘要随着随机数的变化而变化。服务端收到的密码摘要只对特定的随机数有效，而没有密码的话，攻击者就无法计算出正确的摘要，这样我们就可以防止重放攻击。

摘要认证可以保护密码，比基本认证安全很多。但摘要认证并不能保护内容，所以仍然要与HTTPS配合使用，来确保通信的安全。

OAuth

OAuth（开放授权）是一个开放的授权标准，允许用户让第三方应用访问该用户在某一Web服务上存储的私密资源（例如照片、视频、音频等），而无需将用户名和密码提供给第三方应用。OAuth目前的版本是2.0版。

OAuth2.0一共分为四种授权方式，分别为密码式、隐藏式、拼接式和授权码模式。接下来，我们就具体介绍下每一种授权方式。

第一种，密码式。密码式的授权方式，就是用户把用户名和密码直接告诉给第三方应用，然后第三方应用使用用户名和密码换取令牌。所以，使用此授权方式的前提是无法采用其他授权方式，并且用户高度信任某应用。

认证流程如下：

1. 网站A向用户发出获取用户名和密码的请求；
2. 用户同意后，网站A凭借用户名和密码向网站B换取令牌；

3. 网站B验证用户身份后，给出网站A令牌，网站A凭借令牌可以访问网站B对应权限的资源。

第二种，隐藏式。这种方式适用于前端应用。认证流程如下：

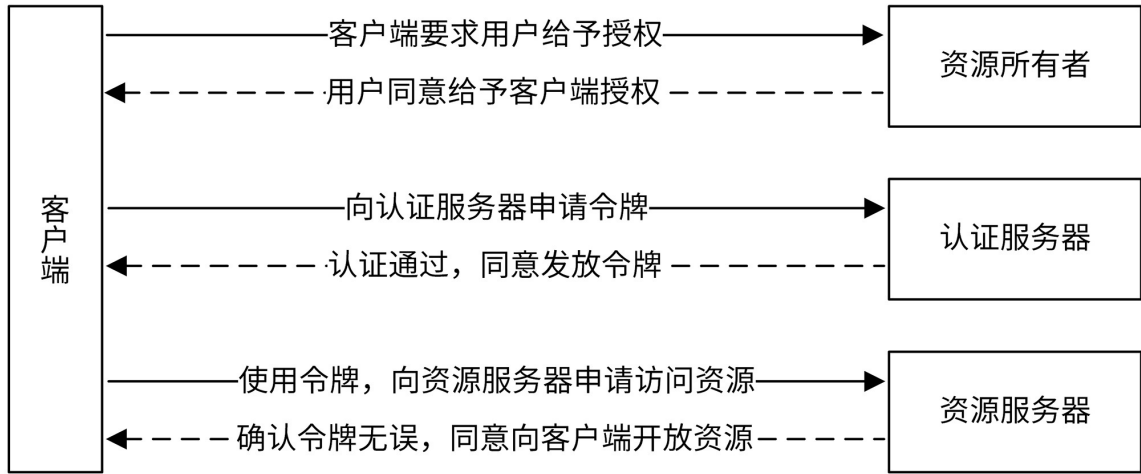
1. A网站提供一个跳转到B网站的链接，用户点击后跳转至B网站，并向用户请求授权；
2. 用户登录B网站，同意授权后，跳转回A网站指定的重定向redirect_url地址，并携带B网站返回的令牌，用户在B网站的数据给A网站使用。

这个授权方式存在着“中间人攻击”的风险，因此只能用于一些安全性要求不高的场景，并且令牌的有效时间要非常短。

第三种，凭借式。这种方式是在命令行中请求授权，适用于没有前端的命令行应用。认证流程如下：

1. 应用A在命令行向应用B请求授权，此时应用A需要携带应用B提前颁发的secretID和secretKey，其中secretKey出于安全性考虑，需在后端发送；
2. 应用B接收到secretID和secretKey，并进行身份验证，验证通过后返回给应用A令牌。

第四种，授权码模式。这种方式就是第三方应用先提前申请一个授权码，然后再使用授权码来获取令牌。相对来说，这种方式安全性更高，前端传送授权码，后端存储令牌，与资源的通信都是在后端，可以避免令牌的泄露导致的安全问题。认证流程如下：



1. A网站提供一个跳转到B网站的链接+redirect_url，用户点击后跳转至B网站；
2. 用户携带向B网站提前申请的client_id，向B网站发起身份验证请求；
3. 用户登录B网站，通过验证，授予A网站权限，此时网站跳转回redirect_url，其中会有B网站通过验证后的授权码附在该url后；
4. 网站A携带授权码向网站B请求令牌，网站B验证授权码后，返回令牌即access_token。

Bearer

Bearer认证，也称为令牌认证，是一种 HTTP 身份验证方法。Bearer认证的核心是bearer token。bearer token是一个加密字符串，通常由服务端根据密钥生成。客户端在请求服务端时，必须在请求头中包含Authorization: Bearer <token>。服务端收到请求后，解析出 <token>，并校验 <token> 的合

法性，如果校验通过，则认证通过。跟基本认证一样，Bearer认证需要配合HTTPS一起使用，来保证认证安全性。

当前最流行的token编码方式是JSON Web Token（JWT，音同 jot，详见 [JWT RFC 7519](#)）。接下来，我通过讲解JWT认证来帮助你了解Bearer认证的原理。

基于JWT的Token认证机制实现

在典型业务场景中，为了区分用户和保证安全，必须对 API 请求进行鉴权，但是不能要求每一个请求都进行登录操作。合理做法是，在第一次登录之后产生一个有一定有效期的 token，并将它存储在浏览器的 Cookie 或 LocalStorage 之中。之后的请求都携带这个 token，请求到达服务器端后，服务器端用这个 token 对请求进行认证。在第一次登录之后，服务器会将这个 token 用文件、数据库或缓存服务器等方法存下来，用于之后请求中的比对。

或者也可以采用更简单的方法：直接用密钥来签发Token。这样，就可以省下额外的存储，也可以减少每一次请求时对数据库的查询压力。这种方法在业界已经有一种标准的实现方式，就是JWT。

接下来，我就来具体介绍下JWT。

JWT简介

JWT是Bearer Token的一个具体实现，由JSON数据格式组成，通过HASH散列算法生成一个字符串。该字符串可以用来进行授权和信息交换。

使用JWT Token进行认证有很多优点，比如说无需在服务端存储用户数据，可以减轻服务端压力；而且采用JSON数据格式，比较易读。除此之外，使用JWT Token还有跨语言、轻量级等优点。

JWT认证流程

使用JWT Token进行认证的流程如下图：



具体可以分为四步：

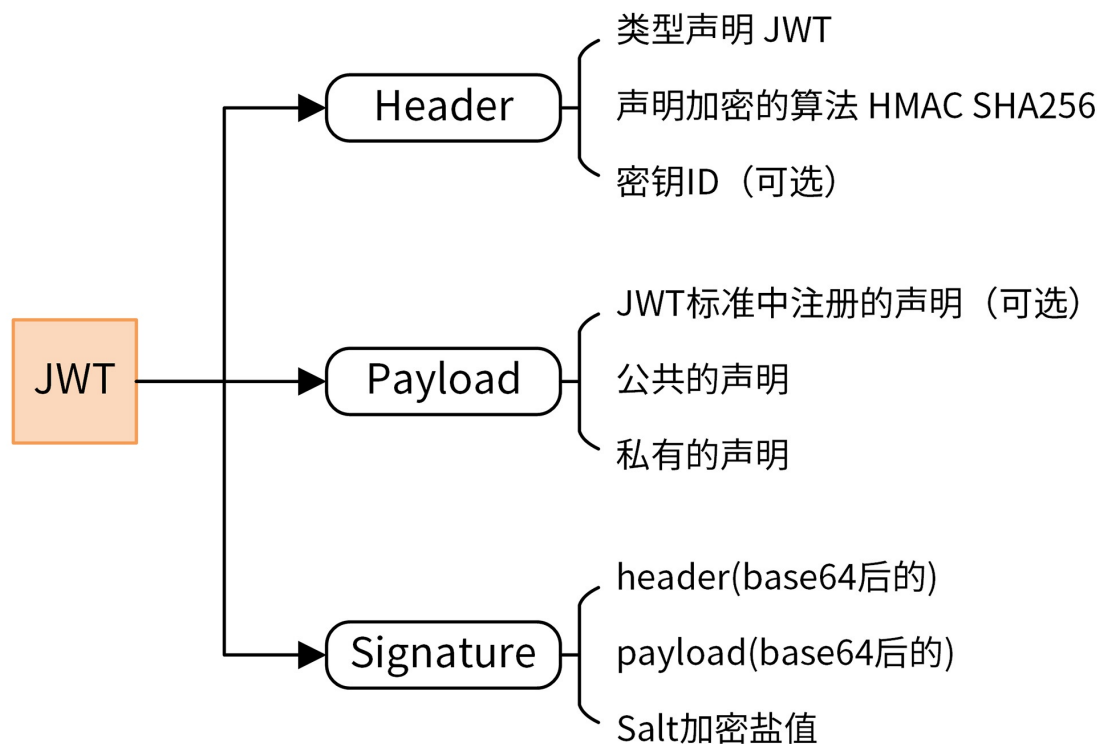
1. 客户端使用用户名和密码请求登录。
2. 服务端收到请求后，会去验证用户名和密码。如果用户名和密码跟数据库记录不一致，则验证失败；如果一致则验证通过，服务端会签发一个Token返回给客户端。
3. 客户端收到请求后会将Token缓存起来，比如放在浏览器Cookie中或者LocalStorage中，之后每次请求都会携带该Token。
4. 服务端收到请求后，会验证请求中的Token，验证通过则进行业务逻辑处理，处理完后返回处理后的结果。

JWT格式

JWT由三部分组成，分别是Header、Payload 和 Signature，它们之间用圆点.连接，并使用Base64编码，例如：

JhdWQi0iJpYW0uYXV0aHoubWFybw90ZWR1LmNvbSIsImV4cCI6MTYwNDE1ODk4NywiaWF0IjoxNjA0MTUxNzg3LCJpc3MiOiJpYW1jdGwiL

JWT中，每部分包含的信息见下图：



下面我来具体介绍下这三部分，以及它们包含的信息。

1. Header

JWT Token的Header中，包含两部分信息：一是Token的类型，二是Token所使用的加密算法。

例如：

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

参数说明：

- typ：说明Token类型是JWT。
- alg：说明Token的加密算法，这里是HS256（alg算法可以有多种）。

这里，我们将Header进行base64编码：

```
$ echo -n '{"typ":"JWT","alg":"HS256"}' | base64
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```


在某些场景下，可能还会有kid选项，用来标识一个密钥ID，例如：

```
{
  "alg": "HS256",
  "kid": "XhbY3aCrfdYcP10FJRu9xcno8JzSbUIvGE2",
  "typ": "JWT"
}
```

2. Payload（载荷）

Payload中携带Token的具体内容由三部分组成：JWT标准中注册的声明（可选）、公共的声明、私有的声明。下面来分别看下。

JWT标准中注册的声明部分，有以下标准字段：



字段名	描述
iss (Issuer)	JWT Token的签发者，其值应为大小写敏感的字符串或Uri
sub (Subject)	主题，sub可以用来鉴别一个用户
exp (Expiration Time)	JWT Token过期时间
aud (Audience)	接收 JWT Token的一方，其值应为大小写敏感的字符串或Uri，一般可以为特定的App、服务或模块。服务器端的安全策略在签发时和验证时，aud必须是一致的
iat (Issued At)	JWT Token签发时间
nbf (Not Before)	JWT Token生效时间
jti (JWT ID)	JWT Token ID，令牌的唯一标识符，通常用于一次性消费的Token

本例中的payload内容为：

```
{
  "aud": "iam.authz.marmotedu.com",
  "exp": 1604158987,
  "iat": 1604151787,
  "iss": "iamctl",
  "nbf": 1604151787
}
```

这里，我们将Payload 进行base64编码：

```
$ echo -n '{"aud":"iam.authz.marmotedu.com","exp":1604158987,"iat":1604151787,"iss":"iamctl","nbf":1604151787}' | base64
eyJhdWQiOiJpYXV0aHoubW90ZW90ZWR1LmNvbSIsImV4cCI6MTYwNDE1ODk4NywiaWF0IjoxNjA0MTUxNzg3LCJpc3MiOiJpYXV1jdGwiLCJuYmYiOiJlE2MDQxNTE3ODd9
```

除此之外，还有公共的声明和私有的声明。公共的声明可以添加任何的需要的信息，一般添加用户的相关信息或其他业务需要的信息，注意不要添加敏感信息；私有声明是客户端和服务端所共同定义的声明，因为base64是对称解密的，所以一般不建议存放敏感信息。

3. Signature (签名)

Signature是Token的签名部分，通过如下方式生成：将Header和Payload分别base64编码后，用.连接。然后再使用Header中声明的加密方式，利用secretKey对连接后的字符串进行加密，加密后的字符串即为最终的Token。

secretKey是密钥，保存在服务器中，一般通过配置文件来保存，例如：

```
# JWT 配置
jwt:
  realm: iam jwt # jwt 标识
  key: dfVpOK8LZeJLZHYmHdb1VdyRrACKpqoo # 服务端密钥
  timeout: 24h # token 过期时间(小时)
  max-refresh: 24h # token 更新时间(小时)
```

这里要注意，**密钥一定不能泄露**。密钥泄露后，入侵者可以使用该密钥来签发JWT Token，从而入侵系统。

最后生成的Token如下：

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJpYXV0aHoubW90ZW90ZWR1LmNvbSIsImV4cCI6MTYwNDE1ODk4NywiaWF0IjoxNjA0MTUxNzg3LCJpc3MiOiJpYXV1jdGwiLCJuYmYiOiJlE2MDQxNTE3ODd9
```

签名后服务端会返回生成的Token，客户端下次请求会携带该Token。服务端收到Token后会解析出header.payload，然后用相同的加密算法和密钥对header.payload再一次加密，并对比加密后的Token和收到的Token是否相同，如果相同则验证通过，不相同则返回HTTP 401 Unauthorized的错误。

最后，关于JWT的使用，我还有两点建议：

- 不要存放敏感信息在Token里；
- Payload中的exp值不要设置的太大，一般开发版本2小时，上线版本7天。当然，你也可以根据需要自行设置。

总结

在开发Go应用时，我们需要通过认证来保障应用的安全。认证，用来验证某个用户是否具有访问系统的权限，如果认证通过，该用户就可以访问系统，从而创建、修改、删除、查询平台支持的资源。业界目前有四种常用的认证方式：Basic、Digest、OAuth、Bearer。其中Basic和Bearer用得最多。

Basic认证通过用户名和密码来进行认证，主要用在用户登录场景；Bearer认证通过Token来进行认证，通常用在API调用场景。不管是Basic认证还是Bearer认证，都需要结合HTTPS来使用，来最大程度地保证请求的安全性。

Basic认证简单易懂，但是Bearer认证有一定的复杂度，所以这一讲的后半部分通过JWT Token，讲解了Bearer Token认证的原理。

JWT Token是Bearer认证的一种比较好的实现，主要包含了3个部分：

- Header：包含了Token的类型、Token使用的加密算法。在某些场景下，你还可以添加kid字段，用来标识一个密钥ID。
- Payload：Payload中携带Token的具体内容，由JWT标准中注册的声明、公共的声明和私有的声明三部分组成。
- Signature：Signature是Token的签名部分，程序通过验证Signature是否合法，来决定认证是否通过。

课后练习

1. 思考下：使用JWT作为登录凭证，如何解决token注销问题？
2. 思考下：Token是存放在LocalStorage中好，还是存放在Cookie中好？

欢迎你在留言区与我交流讨论，我们下一讲见。

精选留言：

- pedro 2021-07-22 08:05:35
问题1：JWT中可新增一个 valid 字段用于表示 token 是否有效，注销后则无效。
问题2：token还是存储在cookie中比较好，可由服务端保存，localStorage在纯前端，中很容易泄露。 [1赞]
- 如来神掌 2021-07-22 21:10:22

方案挺多，如下：

- 1、最简单的，客户端直接删除token，但有弊端
 - 2、token计入数据库，登录后对比，注销时删除
 - 3、注销后记录到黑名单，下次登录后清楚黑名单
 - 4、直接记到Redis，注销后删除，登录后存入，类似数据库
-

作者回复2021-07-23 16:56:58

老哥回答的很赞。主要是删除token在客户端和服务端的缓存

- lark 2021-07-22 18:38:22

在jwt的payload里加一个随机数字段，过期或者失效就重新生成一个随机数。

作者回复2021-07-23 16:59:17

删除客户端的缓存就可以了

- 8.13.3.27.30 2021-07-22 09:35:50

根据文中认证和授权的解释、我理解Oauth应该属于授权？而不是认证