[译] 深入 OAuth2.0 和 JWT

原创 云前端 云前端 2019-11-20

> 原文链接：https://uriotnews.com/?s=oauth2.0+and+jwt

I. 认证和授权

从基于计算机的应用出现伊始，几乎每个开发者在其职业生涯内都会面对的一个最常见也是最复杂的问题，就是**安全性（security）**。这类问题意味着要考虑理解由谁提供什么数据/信息，此外还有关乎时间、校验、再校验等诸如此类的很多其他方面的事情。

而和安全性相关的所有关注点都可以被分解成两类问题：**认证（Authentication）** 和 **授权（Authorization）**。

虽然这两个术语常常交替着使用，但它们本质上表示了不同的功用。让我们试着擦亮记忆，再一次来定义这些概念。

认证

认证是这样一种验证过程：通过让用户、网站、应用程序通过提供合法证书或验证方式，以证明他们符合自己所宣称的身份。认证经常通过用户名和密码证实，有时也会辅以一些其他的只为用户所知的信息。这类信息或元素称为**因子（factors）**。基于这些因子，任何认证机制都可以划分为以下三类：

1. **单因子认证**: 只依赖用户名和密码
2. **双因子认证**: 除了用户名和密码，也需要一块保密信息（比如银行网站可能要求用户输入一个只有自己知道的 PIN）
3. **多因子认证 (MFA)**: 使用两个或多个、来自不同类别的安全性因子（如医院系统需要用户名密码 + 用户智能手机收到的安全验证码 + 指纹信息）

授权

授权指的是一个验证某用户能访问什么的过程。在授权过程中，某用户/应用程序的权限级别被确定后，才被允许访问特定的 APIs/模块。通常，授权发生在用户身份被 认证 之后。

授权是通过使用“策略（policies）”和“规则（rules）”来实现的。

认证 vs 授权

虽然认证和授权常常交替着使用，但可以试着用一个“苏打水和鸡尾酒”的比喻来理解：二者殊为不同 -- 苏打水作为一种原材料，可以被用来制作多种不同的饮料，也可以单独饮用；而鸡尾酒则是一种由多种成分构成的混合品，苏打水也可能是其中之一，但不会只包含这一种。

如此说来，说苏打水等同于鸡尾酒或鸡尾酒就是冒泡的苏打水都是不正确的。相似的是，认证和授权也不是同样的术语；实现得好的话它们可以相得益彰，但本质上是不同的。

	认证	授权
1	确定用户所宣称的身份	确定用户可访问的权限
2	通过合法凭证校验用户	通过规则和策略校验访问
3	早于授权	在认证成功后执行
4	通过 ID tokens 实现	用 Access Tokens 实现

在真实场景中，要结合使用认证和授权以保护资源。当你能证明自己的身份之前，不应该被允许访问资源；而即使证明了身份，若无访问权限，依然应被拒绝。

实现机制

要实现认证和授权有多种途径，但时下最流行的是 “基于令牌（*token-based*）” 的方法。

什么是基于令牌的认证？

基于令牌的认证和授权（Token-based authentication/authorization）是这样一种技术：当用户在某处输入一次其用户名和密码后，作为交换会得到一个唯一生成的已加密令牌。该令牌随后会替代登陆凭证，用以访问受保护的页面或资源。

这种方式的要点在于确保每个发往服务器的请求都伴随着一个已签名的令牌，服务器利用该令牌核验真实性之后才对当次请求做出响应。

令牌是什么？

一个“令牌”就是服务器生成的一段数据，包含了唯一性识别一个用户的信息，一般被生成为一长串随机字符和数字。

比如看起来可能像这样：`cc7112734bbde748b7708b0284233419` ，或更复杂些比如：`eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJtZXNzYWdlIjoiSldUIFJ1bGVzISIsImlhdCI6M`。

这个令牌本身是无意义和无用的，但结合适当的令牌化系统，就会变成保证应用安全性的重要一环。

为何要使用令牌？

比之于传统的 cookies 等手段，使用令牌有如下好处：

- **无状态**：令牌是自包含（self-contained）的，其包含了所有用于认证的信息。这对于可扩展性是极佳的，可以让服务器从不得不存储 session 的境地中解脱
- **可以在任何地方生成**：令牌的生成和校验是解耦的，让使用单独的服务器甚至不同的厂商来完成令牌的签名成为了可能的选项，如 Auth0（译注：一家 ‘Identity-as-a-service’ 提供商）

- **细粒度的访问控制**：通过令牌负荷（token payload），不仅是用户可访问的资源这一项，也可以轻易制定更多用户角色和权限

基于令牌的实现

尽管具体实现各有不同，但基本上都涉及以下步骤：

1. 用户通过用户名和密码请求访问
2. 应用验证凭证
3. 应用向客户端发放已签名的令牌
4. 客户端存储令牌，并将其附加在其后的每次请求中一同发送
5. 服务器验证令牌并响应数据

虽说并无关于该如何实现你的应用的限制，但 *IETF*（*Internet Engineering Task Force*，互联网工程任务组）还是定义了一些标准。其中最流行的有两个：

1. OAuth 2.0 (RFC 6749 and RFC 6750).
2. JWT (RFC 7519).

II. 了解 OAuth 2.0

我们已经刷新了关于认证和授权的认知，并将了解基于令牌认证的常识。在本章节中，来看看最常用的一种实现：`OAuth 2.0`。

OAuth 2.0 简介

在传统 C/S 模型中，客户端（client）通过让服务器认证资源拥有者（resource owner）的凭证来请求服务端受保护的资源。资源拥有者会将自己的凭证分享给第三方应用（third-party applications），让后者得以访问受限资源。这种凭证分享行为会造成若干问题和限制，其中的一些如下所列：

- 第三方应用需要存储资源拥有者的凭证以持续利用，典型的如存储一个明文的密码
- 尽管存在密码固有的安全性弱点，服务器仍得支持密码认证
- 对于资源拥有者的受限资源，第三方应用得到了过于宽泛的访问权限；置资源拥有者于无力约束访问时长或限制访问资源子集的境地
- 资源拥有者无法撤回个别第三方的访问权限，除非改变所有第三方的密码

OAuth 针对这些问题提出了引入一个认证层，并把客户端（client）的角色与资源拥有者的角色分离开来。所以：

> OAuth 是一种授权协议，以允许用户将对其在一个站点上的资源的受限访问许可给另一个站点，而不必公开其凭据

OAuth 为客户端提供一种“安全代理访问”能力，用以代表资源拥有者访问服务器资源。OAuth 指定了这样一个过程：资源拥有者在不分享其凭证的前提下授权第三方访问其服务器资源。

下面举个例子来说明：

你知道有些小轿车的“泊车钥匙”吧？如果还未曾耳闻的话，那就是有些车型（没错，就是特别奢侈的那些！）附有一种特别的钥匙，可以在泊车时交给服务员。和你的正常钥匙不同的是，这种钥匙不允许汽车开出去一两英里那么远。

某些泊车钥匙打不开后备箱，另一些则访问不了车载电话的通讯录。无论此类限制是什么，思路都是一样清晰的：你让某人通过特殊的钥匙有限访问你的车，但你的正常钥匙能解锁一切。

类似的，OAuth 中的“泊车钥匙”就是 **访问令牌（Access Tokens）**，通过其允许对资源的不同级别的访问。

OAuth 2.0 术语

角色（Roles）：OAuth2.0 规范定义了四种角色。

1. **资源拥有者 Resource Owner**：一个有能力对访问受保护资源授权的实体（entity）。当这个实体是一个人时，它就表示终端用户（end-user）。
2. **资源服务器 Resource Server**：存储受保护资源的服务器，能接受和响应使用访问令牌的受保护资源请求。
3. **客户端 Client**：一个发起对受保护资源请求的应用程序，其代表了资源拥有者并持有其凭证。术语 “client” 并不意味着任何实现特征（如该应用程序是否运行在服务器上、桌面端，或是其他设备上）。

4. **授权服务器 Authorization Server**：当资源拥有者认证成功并获得授权之后，该服务器向客户端授予访问令牌。

令牌（Tokens）：令牌有两种类型。

1. **访问令牌 Access Token：** 访问令牌即表示颁发给客户端之授权的一个字符串。对用户端来说这个字符串一般是晦涩的。令牌代表了特殊的访问范围和持续时间，由资源拥有者授予，被资源服务器和授权服务器实施。

   令牌可能表示一个用来取回认证信息的标识符，也可能以一种可验证的方式（如包含一些数据和签名）自包含认证信息。

2. **更新令牌 Refresh Token：** 更新令牌是用来获得访问令牌的凭证。更新令牌由授权服务器向客户端发出，并在当访问令牌无效或过期后，用更新令牌获得一个新的访问令牌；也可能用其获得访问范围相同或更窄的附加访问令牌（这些访问令牌和经过资源拥有者授权的访问令牌相比，可能有更短的生存时间和更少的权限）。

   是否发放一个更新令牌是由授权服务器酌情处理的；如果发放了则会用在后续发放访问令牌时。

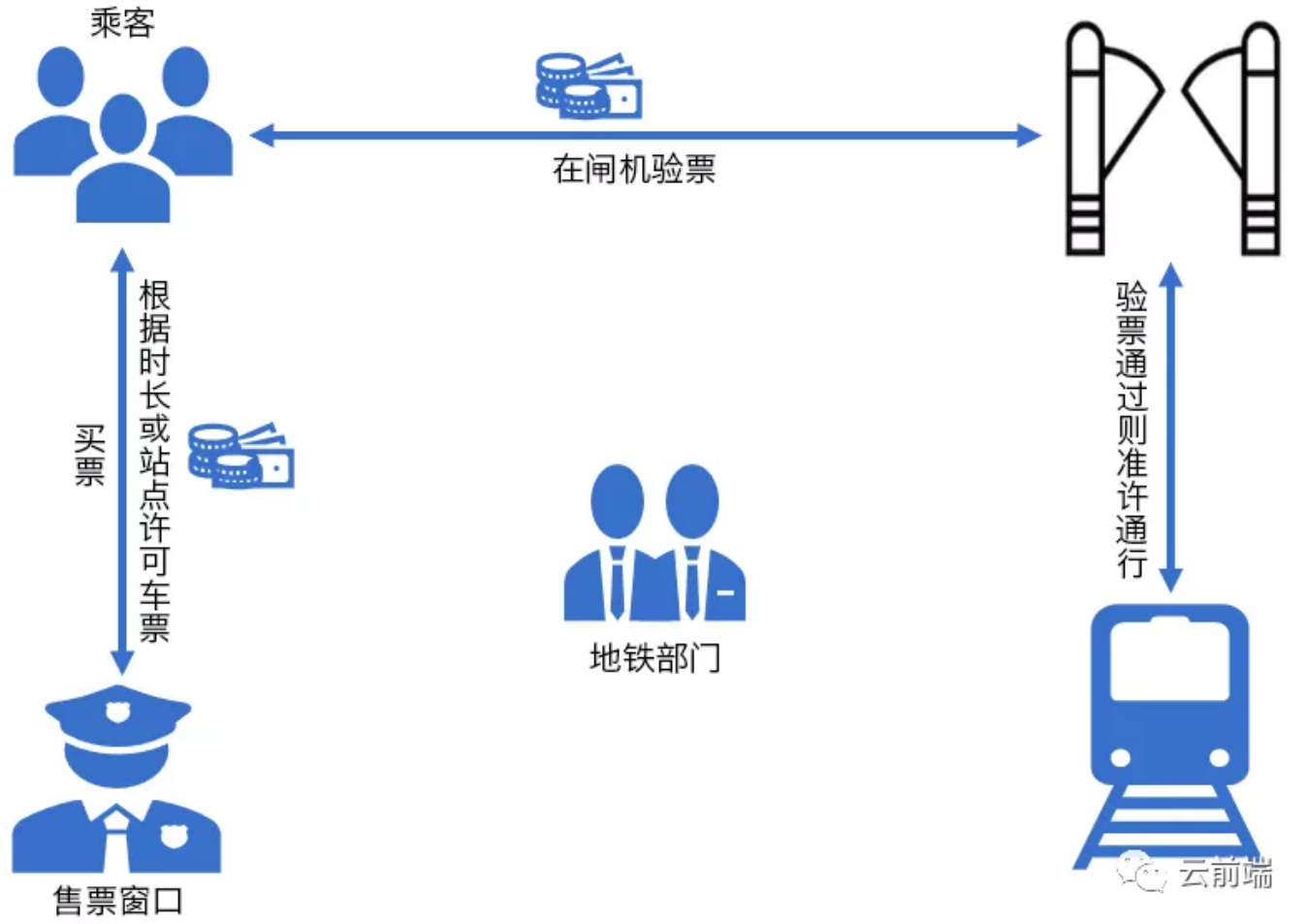
不同于请求令牌，更新令牌专为授权服务器设计，不会发送给资源服务器。

授权许可（Authorization Grant）：授权许可是一种表示资源拥有者之认可（访问其受保护资源）的凭证，被客户端用于获取访问令牌。OAuth 2.0 规范定义了四种许可类型：

1. **授权代码 Authorization Code:** 授权代码由使用一个作为客户端和资源拥有者之中间人的授权服务器处获取。不同于从资源拥有者那里直接请求授权，客户端将资源拥有者引导至授权服务器，后者又将资源拥有者伴随一个授权代码引导回客户端。
2. **隐式许可 Implicit Grant:** 不向客户端发送授权代码，而是由客户端直接获取访问令牌。
3. **资源拥有者密码凭证 Resource owner password credentials (ROPC):** 资源拥有者密码凭证（如用户名和密码）可被直接作为授权许可以获取访问令牌。ROPC 只应被使用在资源拥有者和客户端（如客户端是所在设备操作系统的一部分，或是一个高权限的应用）之间需要高信任等级，且其他几种授权许可（如授权代码）不可用的时候。
4. **客户端凭证 Client Credentials:** 当授权范围限于客户端控制之下的受保护资源，或是与授权服务器事先约定的受保护资源的时候，客户端凭证（或其他客户端认证形式）可被用来作为一种授权许可。典型的是客户端代表自己（其同时也是资源拥有者）或客户端正在请求访问基于事先和资源服务器约定好的受保护资源的时候。

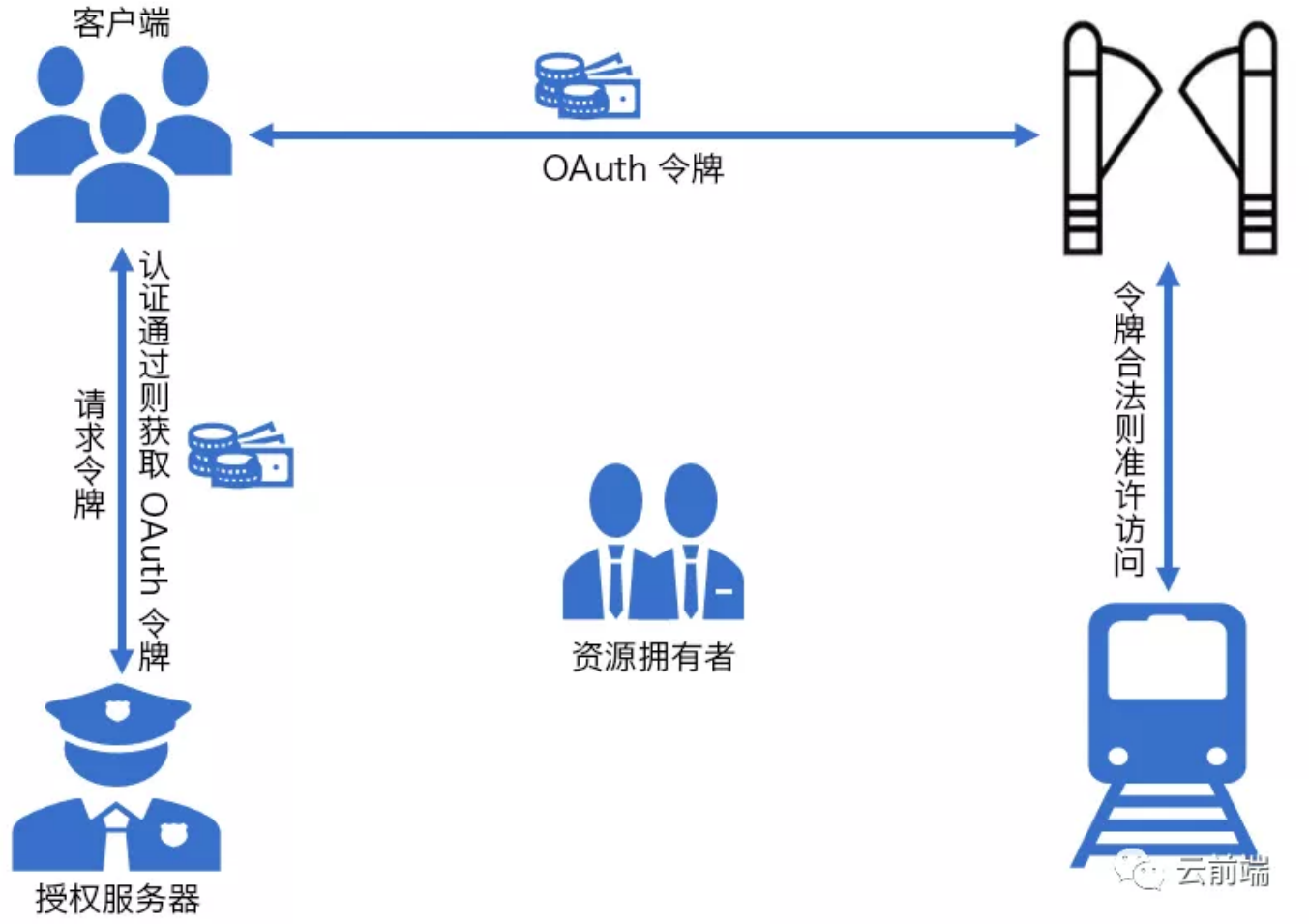
处理 OAuth 2.0 时理解这些术语是至关重要的。所以，也试着用一个例子来说明。

想象一个地铁运输系统。典型的引导流程如下：一位乘客（commuter）从售票机或售票窗口购买车票，制票系统许可这张车票在有限的时间或站点数量之间是合法的。而后，乘客在闸机验票，车票合法则准许进入，即可乘坐列车。



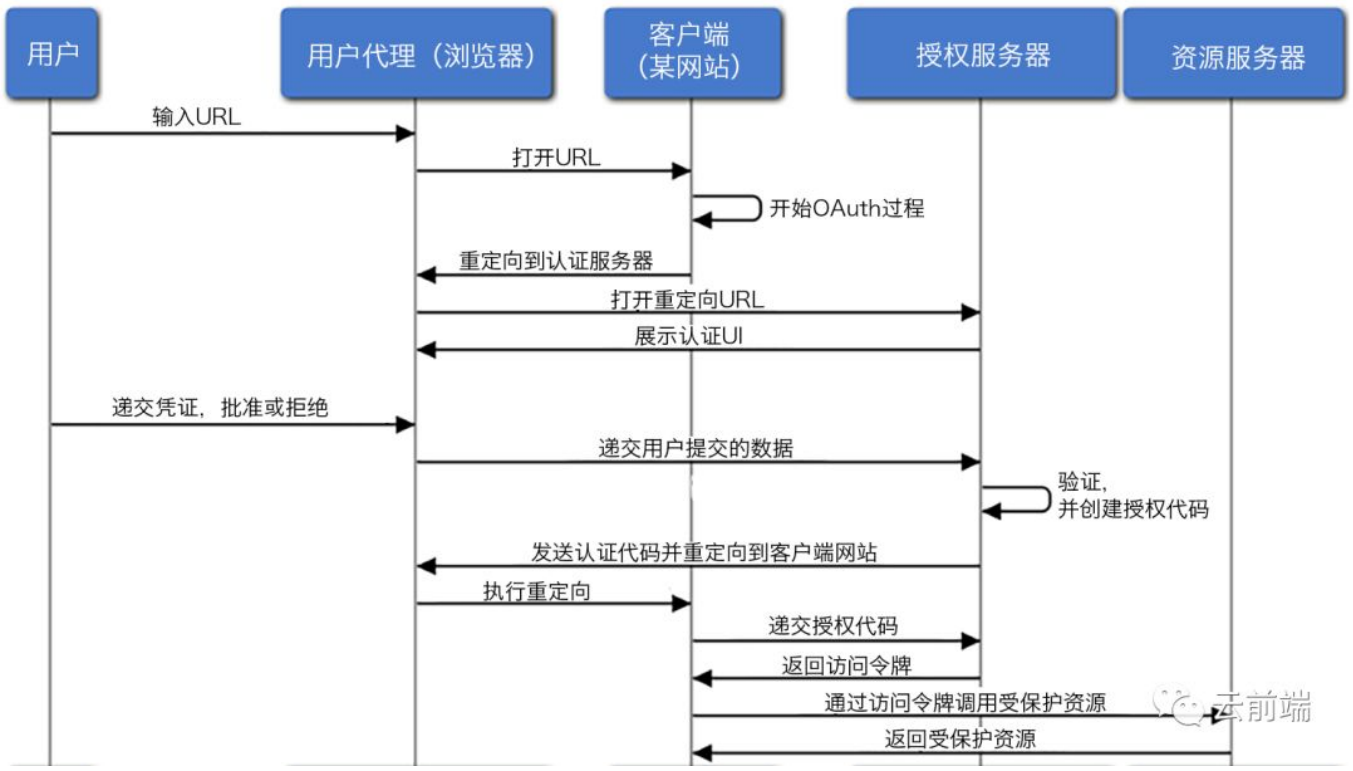
以上场景可以和下面的 OAuth 2.0 中的角色对应起来：

乘客 (**客户端**) 打算利用地铁 (**受保护的资源**)，所以他/她得先向售票机或售票窗口 (**资源服务器**) 买票。制票系统 (**授权服务器**) 代表地铁部门 (**资源拥有者**) 以车票 (**令牌**) 为依据许可访问。



OAuth 2.0 控制流

一次 OAuth 2.0 的流程可用下图表示:



OAuth 2.0 用例

OAuth 2.0 把认证从授权决策中解耦。恰当设计的 OAuth 2.0 令牌既可以支持细粒度授权，也可以支持粗粒度授权。对于任何从另一处（服务器/应用）访问存储在某处的资源/数据的场景，OAuth 2.0 可说是最适用的方法之一了。

以下列出一些场景，我们将尝试通过一个可穿戴设备的例子理解 OAuth 2.0 用例。

就拿运动手环来说吧，假设 Alice 买了一个，并用移动端上配套的手环 app 跟踪并分析运动过程。那么流程会是什么样呢？

- 首先，Alice 需要在手环 app 中创建个人档案。一种方法是用手环 app 提供的档案创建表单，另一种方法是让手环 app 访问其他 app 并拉取 Alice 已经存储在那里的档案信息 -- 就拿 FriendBook 这个明显是虚构的社交媒介网站来举例吧。
- 手环 app 将重定向到 FriendBook 的登录界面。一旦 Alice 成功使用其凭证登录，她将看到一个准许同意的页面，询问她或请她验证她的哪些信息要分享，以及她允许访问哪些存储在 FriendBook 上的东西。在确认之后，手环 app 就可以使用 OAuth 2.0 从 FriendBook 拉取并使用数据了。
- 可穿戴设备会向手环 app 发送数据，其后，手环 app 会同步数据到服务器，以期存档和分析。

客户端密码： (尽管使用了 *OAuth 2.0* 的认证应该被避免). Alice 不必创建一个新密码；取而代之的是，她使用自己在 FriendBook 服务器上已经创建的密码。

Web 服务器： 可穿戴设备的 app 不必每次操作都发起登录。Alice 要从 FriendBook 上分享或拉取数据，手环 app 将能够以服务器对服务器的方式访问那些数据。

用户代理： 手环 app 扮演了其应用服务器的代理人的角色，用来从主服务器上同步数据。由于使用了 OAuth 2.0 对此授权，该代理可以准确访问服务器上的资源（数据）。

3. 了解 JWT

下面来看看 JWT。

JSON Web Token (JWT)，通常读作 “**jot**”，是一个定义了以 JSON 对象紧凑而自包含的在各方之间安全传输信息的标准。其包含了声明方面的信息，特别的被用于如 HTTP 等空间受约束的环境；该信息可被验证，也是可信的，因为经过了数字化签名。JWT 可以用 密钥（如 *HMAC*） 或 公钥私钥对（*RSA* 或 *ECDSA*） 签名。

JWT 的两个特性是：

- **紧凑 Compact：** 因为其相对较小的尺寸，JWT 可以借由 URL 发送， 作为一个 POST 参数，或在一个 HTTP header 内。
- **自包含 Self-contained：** 一个 JWT 包含了所有关于一个实体的所需信息，以避免多次查询数据库。JWT 的接纳者同样无需调用服务器以验证令牌。

这些令牌可以是被签名的、被加密的，或两者皆有。签名过的令牌被用来验证令牌完整性，而加密过的令牌用来隐藏声明。

注意：正如名称所暗示的，JWT 是 JSON 形式的，也就意味着其包含键值对。虽说在 JSON 合法和有关方一致性方面，对键和值有多长并无限制，但大多数标准都遵循了 *3*个字母 的键格式。

JWT 术语

JWT 表现为由点（ `.` ）分割的三个字符串组成的一个序列，典型的格式看起来如下：

> AAAAA.BBBBB.CCCCC

三个子串分别称作 **头部（Header）** 、 **负载（Payload）** 和 **签名（Signature）**，下面逐一讲解：

头部 Header

虽说只要相关几方之间有共识，则在头部中放什么是没有限制的，但通常由两部分组成：

1. **typ**: 表示令牌类型（type），值为 `JWT`
2. **alg**: 表示签名此令牌的算法（algorithm），如 `HMAC` 、 `RSA` 、 `SHA`

负载 Payload

JWT 的第二部分表示负载，这部分由声明（claims）组成。

所谓声明就是关于实体和任意附加数据的信息。在一段 JWT 中，声明由键表示。这些声明是依赖上下文的，且应该相应的被处理和被理解，但依每种规范会有若干标准规则应用于声明：

1. 在一个 JWT 声明集合中，每个声明的名称必须是唯一的
2. 对于 JWT 的处理逻辑，必须 保证这种唯一性，要么拒绝重复的名字，要么用一个 JSON 处理器返回重复项中词法上最后一个名字
3. 使用 JWT 的应用要明确其选用的声明标准，并定义必须项和可选项
4. 因为 JWT 的核心目标之一就是精简，故所有名字也应该简短

一个可能的负载例子：

```
{
    "sub": "1234567890", "name": "Alice", "admin": true
}
```

负载中的声明又可以细分为以下三种类型：

已注册的声明

有一些声明注册在 IANA(https://dzone.com/refcardz/core-json) 的 “JSON Web 令牌声明” 注册表中。这些声明并非是在所有情况下都要求强制使用或实现的，准确的说它们是作为提供一个有用的集合的起始点而被注册的。

其中一些有必要了解的是：

1. **iss (issuer):** 声明了发行人，也就是发行 JWT 的主体。处理此声明通常是因应用而异的。“iss” 值是一个大小写敏感的字符串，包含一个普通字符串或者一个 URL。该声明是可选的
2. **sub (subject):** 表示 JWT 的主体 (用户)。值必须要么是全局唯一的，要么在发行人上下文范围内局部唯一。处理该声明通常也是因应用而异的。“sub” 值是一个大小写敏感的字符串，包含一个普通字符串或者一个 URL。该声明是可选的
3. **aud (audience):** 表示 JWT 的目标接收方。如果当该声明存在且处理该声明的一方不能通过 “aud” 的值进行自我身份验证时，则 JWT 必须被拒绝。大多数情况下，这个值是由大小写敏感的字符串（包含一个普通字符串或者一个 URL）组成的数组。该声明是可选的
4. **exp (expiration):** 表示过期时间，即等于或晚于那个时刻再处理 JWT 则绝不可被接受。其值通常是以秒记的时间戳（译注：按 POSIX 中定义的 “seconds since epoch” 标准，也就是 PHP 等语言中常用的那种）。该声明是可选的
5. **nbf (not before) :** 表示一个时间，即早于那个时刻再处理 JWT 则绝不可被接受。其值通常是以秒记的时间戳。该声明是可选的
6. **iat (issued at):** 表示发出 JWT 的时刻。可用于判断 JWT 的寿命。必须是一个时间戳。该声明是可选的
7. **jti (JWT ID):** 为 JWT 提供一个唯一的身份识别符，其值必须难以重复，以防 JWT 被重复执行。该声明是可选的

公开声明

此类声明的名字可被 JWT 使用者任意定义。但为了预防冲突，任何新名字都应该注册在 `IANA “JSON Web Token Claims”` 注册表中，或将其定义为包含防冲突命名空间的 URI 等。

在任何情况下，对名字和值的定义都要考虑到合理的预防措施，以确保它们在其定义的命名空间中受控。

私有声明

这可以理解为是创建自定义声明以在应用内共享信息规格，可以是除以上两种外的任意声明名字。与公有声明不同，私有声明受制于冲突问题，要小心使用。

签名

签名先是通过对头部和负载 Base64 编码而生成，其后会与一个密钥联合，最好被头部中指定的算法签名。

签名被用于校验 JWT 的发送者是否名实相符，以及信息在传送过程中是否被更改。比如，如果创建了一个使用 HMAC SHA256 算法之令牌的签名，你会像下面这样做：

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

一个更完整的例子

观察如下 JWT 签名：

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJzYXRpc2giLCJhdWQiOiJteWFwcCIsIkNVU1QiOiI
```

该签名使用了 HS512 算法编码，并包含了如下信息：

```
Header: {
    "alg": "HS512",
    "typ": "JWT"
}
Payload: {
    "sub": "satish",
    "aud": "myapp",
    "CUST": "1",
    "exp": 1566214585,
    "iss": "my-auth-app"
}
```

JWT 用例

认证

当用户使用其凭证成功登录后，一个 ID 令牌会被返回。按照 OpenID Connect (OIDC) 规范，该 ID 令牌就是一个 JWT。

授权

一旦用户登录成功，应用就可能会代表用户请求访问路由、服务、资源等。为此，将使用一个访问令牌，形式上可能就是 JWT。每个后续的请求也都包含该访问令牌。由于 JWT 开销很小，也能轻易用于跨域名访问，单点登录（SSO，Single Sign-on）广泛使用这项技术。

信息交换

由于可被签名，JWT 是一种在多方间安全传递信息的良好方式，这意味着你能确定发送者名实相符。另外，一个 JWT 结构允许你验证内容没有被篡改过。

为何使用 JWT ？

解耦

JWT 最大的优势（比之于使用内存内随机令牌的用户 session 管理）就是其使得对第三方服务器认证逻辑的代理可以：

- 一个集中式的、内部自定义开发的认证服务器
- 更典型的是，使用 LDAP 这种可以发出 JWT 的商业产品
- 甚至可以使用一个纯第三方的认证提供商

认证逻辑/服务器可以从应用服务器完全分离，无需在应用间再分享密码摘要。

无状态

由于 JWT 是自包含的，且无需在内存中保持请求之间的令牌，所以应用服务器可以做到完全无状态（stateless）。认证服务器可以颁发令牌，将其发回后就立即丢弃掉。

紧凑

JSON 比 XML 简介，所以当其被编码后，一个 JWT 比 SAML 令牌更小。这使得 JWT 成为一个在 HTML 和 HTTP 环境中传送的好选择。

更安全

为了签名，JWT 可以使用一个公钥/私钥对，表现为 X.509 证书的形式。一个 JWT 也可以通过分享使用了 HMAC 算法的密钥而被对称签名。同时虽然 SAML 令牌也可以使用 JWT 这样的公钥/私钥对，但相比于签名 JSON 的简单性，想用 XML 数字签名算法签名 XML 却不会引入未知的安全漏洞是非常困难的。

更通用

因为直接映射到对象，JSON 处理器在大多数编程语言中都更常见。相反，XML 没有自然的 文档到对象 的映射。这意味着 JWT 比 SAML 更易用。

更易处理

JWT 为互联网规模而设计，意思就是其在用户设备上更易处理，特别是移动端。

JWT：要考虑到的点

除去以上说过的优缺点，JWT 标准也有其自身的问题：

- 如果需要封锁或冻结一个用户账号，应用就不得不等待令牌过期才能完全停工。
- 如果用户要更新密码（例如在账户劫持的情况下）且一个认证在之前已经被执行过的话，那么由之前的密码产生的令牌会在过期前持续有效。
- 在标准实现中，没有“更新”令牌被指定。因此过期后用户将重新认证。
- 在不违背 JWT 令牌的“无状态”方面的前提下，是不可能破坏一个令牌的，即便令牌已从浏览器被删除，它也会在过期前一直有效。

为了应对这些调整，一些 JWT 库在标准实现之上增加了一个层，并允许更新令牌机制，同时也包含一些特性如在必要情况下强制用户重新认证等。

JWT：最佳实践

在动手实现 JWT 之前，让我们了解一些最佳实践，以确保基于令牌的认证恰当地用于你的应用中。

1. 保证安全。签名 key 应该同其他任何凭证一样被处理，并只出示给必须需要它的服务。

2. 不要在负载中加入敏感信息。令牌被签名为难操作易解码的形式。向负载中添加最少的声明以保证性能和安全性。
3. 给令牌设置过期时间。技术上来说，一旦令牌被签名 -- 它就是永久有效的，除非用来签名的 key 改变，或明确的设置了过期时间。这会造成隐患，所以应该有令牌的过期、撤销策略。
4. 拥抱 HTTPS。不要向非 HTTPS 的连接发送令牌，因为那些请求可以被拦截从而连累到令牌。
5. 考察你所有的授权用例。增加一个次要的令牌验证系统以确保令牌能从你的服务器上生成，举例来说，也许不是通用做法，但可能对实现需求是很必要的。

更多

用 Spring Boot 2 和 JWT 实现基于角色的访问控制

--End--

搜索 fewelife 关注公众号

转载请注明出处

发朋友圈5个赞和100个赞，差别到底在哪儿?

原来是西门大嫂

张杰谢娜婚姻？刘诗诗杨幂复合？杨紫张艺兴在一起?

花姐超好看