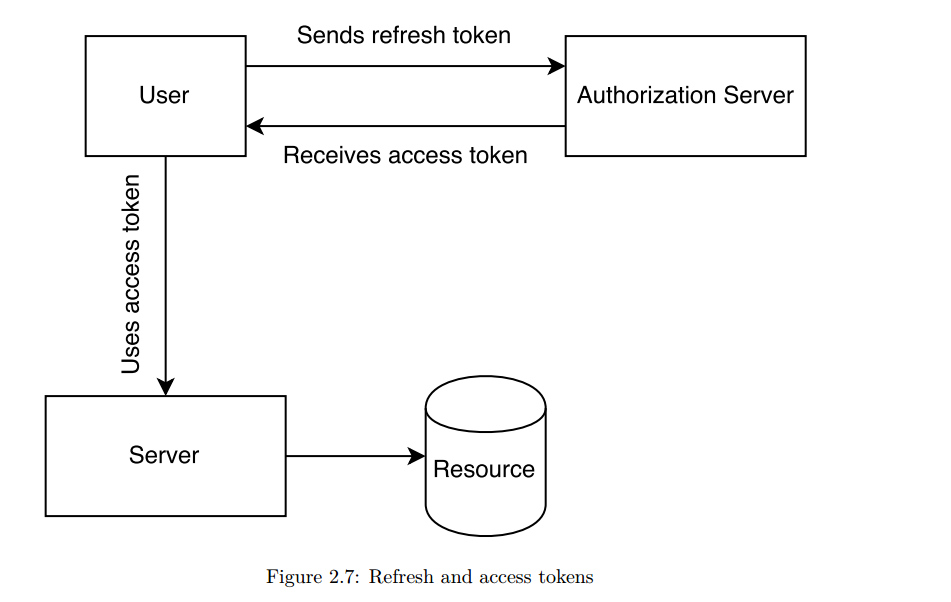
2.2.1访问和刷新令牌  
 访问令牌和刷新令牌是在分析不同联合身份解决方案时经常会遇到的两种令牌类型。我们将简要解释它们是什么以及它们如何在身份验证和授权上下文中提供帮助。

这两个概念通常在OAuth2规范的背景下实施。OAuth2规范定义了一系列必要的步骤，通过将访问与所有权分离来提供对资源的访问(换句话说，它允许具有不同访问级别的多个方访问相同的资源)。这些步骤的几个部分是由实现定义的。也就是说，相互竞争的OAuth2实现可能无法互操作。例如，令牌的实际二进制格式未指定，但其目的和功能已被定义为：

访问令牌是允许拥有它们的人访问受保护资源的令牌。这些令牌通常具有短暂的生命周期，并且可能会嵌入一个截止日期。它们还可能携带或与其他信息相关联(例如，访问令牌可能携带允许请求的IP地址)。这些额外的数据是由具体实现定义的。

另一方面，刷新令牌允许客户端请求新的访问令牌。例如，在访问令牌过期后，客户端可以向授权服务器请求新的访问令牌。为了满足这个请求，需要使用刷新令牌。与访问令牌不同，刷新令牌通常具有较长的生命周期。



分离访问令牌和刷新令牌的关键在于使访问令牌易于验证。带有签名(例如签名的JWT)的访问令牌可以由资源服务器自己进行验证。无需为此目的联系授权服务器。

另一方面，刷新令牌需要访问授权服务器。通过将验证与对授权服务器的查询分开，可以实现更好的延迟和更简单的访问模式。在令牌泄漏的情况下，通过使访问令牌尽可能短暂并在其中嵌入额外的检查(如客户端检查)来实现适当的安全性。

由于刷新令牌的长寿命，必须保护其不被泄漏。在发生泄漏的情况下，可能需要在服务器上进行黑名单处理（短寿命的访问令牌最终会强制使用刷新令牌，因此在对资源进行黑名单处理并且所有访问令牌都过期后，资源得到保护）。

注意：访问令牌和刷新令牌的概念是在OAuth2中引入的。OAuth 1.0和1.0a使用 "token" 这个词的含义有所不同。

## JSON Web令牌详解

所有JWT都由三个不同的元素构成：头部（header）、载荷（payload）和签名/加密数据。前两个元素是特定结构的JSON对象。第三个元素取决于用于签名或加密的算法，在未加密的JWT中可能会被省略。JWT可以以一种称为JWS/JWE紧凑序列化的方式进行编码。

JWS（JSON Web Signature）和JWE（JSON Web Encryption）规范定义了第三种序列化格式，称为JSON序列化，这是一种非紧凑表示法，允许在同一个JWT中存在多个签名或接收者。这在规范的第4章和第5章中有详细说明。

紧凑的序列化是对第一个两个JSON元素（头部和载荷）以及用于签名或加密的数据（该数据本身不是JSON对象）的UTF-8字节的Base64 URL安全编码，如有需要。这些数据也进行了Base64-URL编码。这三个元素之间用点号（“.”）分隔。

JWT使用一种适用于URL的Base64编码的变体。该编码基本上用 "-" 和 "\_" 字符替换了 "+" 和 "/" 字符，而且移除了填充。这个变体被称为 base64url。请注意，在本文档中所有关于Base64编码的引用都指的是这个变体。

得到的序列是一个可打印的字符串，如下所示（为了可读性插入了换行符）：

**eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.**

**eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.**

**TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ**

请注意，点号将JWT的三个元素分隔开来（顺序为：头部、载荷和签名）。

在这个例子中，解码后的头部是：

{

"alg": "HS256",

"typ": "JWT"

}

解码后的有效载荷是：

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

而用于验证签名的密钥是 "secret"。JWT.io 是一个交互式的平台，可用于深入了解JWT。复制上面的令牌，然后尝试进行编辑，看看会发生什么。

3.1 头部  
 每个JWT都携带一个头部（也称为JOSE头部），其中包含有关自身的声明。这些声明确定了所使用的算法、JWT是否已签名或加密，以及通常如何解析JWT的其余部分。

根据所涉及的JWT类型，头部中可能会有更多的字段是强制性的。例如，加密的JWT携带有关用于密钥加密和内容加密的密码算法的信息。这些字段在未加密的JWT中是不存在的。

对于未加密的JWT头，唯一的强制声明是**alg**声明:

* **alg**：用于签名和/或解密此JWT的主要算法

对于未加密的JWT，此声明必须设置为值**none**

可选的头部声明包括 typ 和 cty 声明：

* **typ：**JWT本身的媒体类型。此参数仅意为在JWT可能与携带JOSE头部的其他对象混合的情况下使用。实际上，这种情况很少发生。当存在时，此声明应设置为值 JWT。
* **cty:** 内容类型。大多数JWT携带特定的声明以及作为其载荷一部分的任意数据。对于这种情况，不得设置内容类型声明。对于载荷本身是JWT（嵌套JWT）的情况，此声明必须存在并携带值 JWT。这告诉实现需要进一步处理嵌套JWT。嵌套JWT很少见，因此 cty 声明在头部中很少出现。

因此，对于未加密的JWT，头部简单地为：

{

"alg": "none"

}

它被编码为:

**eyJhbGciOiJub25lIn0**

可以向头部添加额外的用户定义的声明。通常情况下，除非在解密之前需要特定于用户的元数据，否则这样做的用途有限。

3.2载荷

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

载荷是通常添加所有有趣用户数据的元素。此外，规范中定义的某些声明也可能存在。与头部一样，载荷是一个JSON对象。没有强制性的声明，尽管特定的声明具有明确的含义。JWT规范规定，实现不理解的声明应该被忽略。具有特定含义的声明称为注册声明。

**3.2.1 注册声明**

* **iss：**来自单词 "issuer"。一个区分大小写的字符串或URI，用于唯一标识发出JWT的一方。其解释是特定于应用程序的（没有中央管理发行方的权威机构）。
* **sub：**来自单词 "subject"。一个区分大小写的字符串或URI，用于唯一标识此JWT携带信息的一方。换句话说，此JWT中包含的声明是关于此一方的陈述。JWT规范规定，该声明在发行方的上下文中必须是唯一的，或者在不可能的情况下，全局唯一。此声明的处理方式是特定于应用程序的。
* **aud：**来自单词 "audience"。可以是单个区分大小写的字符串或URI，也可以是一组这样的值，用于唯一标识此JWT的预期接收方。换句话说，当存在此声明时，读取此JWT中数据的一方必须在 aud 声明中找到自己，或者忽略JWT中包含的数据。与 iss 和 sub 声明一样，此声明是特定于应用程序的。
* **exp：**来自单词 "expiration"（时间）。一个表示特定日期和时间的数字，格式为 POSIX 规定的“自纪元以来的秒数”。此声明设置了JWT被视为无效的确切时刻。一些实现可能允许在时钟之间存在一定偏差（即在到期日期后的几分钟内将此JWT视为有效）。
* **nbf：**来自 "not before"（时间）的缩写。与 exp 声明相反。一个表示特定日期和时间的数字，格式为 POSIX 规定的“自纪元以来的秒数”。此声明设置了JWT被视为有效的确切时刻。当前时间和日期必须等于或晚于此日期和时间。一些实现可能允许存在一定的偏差。
* **iat：**来自 "issued at"（时间）。一个表示特定日期和时间的数字（与 exp 和 nbf 具有相同的格式），表示此JWT的发行时间。
* **jti：**来自 "JWT ID"。一个字符串，表示此JWT的唯一标识符。此声明可用于区分具有类似内容的JWT（例如，防止重播攻击）。确保唯一性是由实现负责的。

正如您可能已经注意到的那样，所有的名称都很短。这符合设计要求之一：使JWT尽可能小。

字符串或URI：根据JWT规范，URI被解释为包含冒号（:）字符的任何字符串。由实现提供有效的值。

**3.2.2 公共和私有声明**

所有不属于注册声明部分的声明都是私有或公共声明。

* 私有声明：由JWT的用户（消费者和生产者）定义的声明。换句话说，这些是为特定情况使用的特设声明。因此，必须注意防止冲突。
* 公共声明：是要么在IANA JSON Web Token Claims注册表（用户可以在其中注册其声明以防止冲突的注册表）中注册的声明，要么使用具有防冲突名称的命名（例如，通过在其名称前添加命名空间）。

实际上，大多数声明要么是注册声明，要么是私有声明。一般而言，大多数JWT都是为特定目的发行的，并考虑了明确的一组潜在用户。这使得选择具有防冲突名称的任务变得简单。

3.3 不安全的JWT

根据我们目前学到的知识，可以构建不安全的JWT。这些是最简单的JWT，由一个简单的（通常是静态的）头部组成：

{

"alg": "none"

}

以及用户定义的有效载荷。例如：

{

"sub": "user123",

"session": "ch72gsb320000udocl363eofy",

"name": "Pretty Name",

"lastpage": "/views/settings"

}

由于没有签名或加密，这个JWT被编码为简单的两个元素（为了可读性插入了换行符）：

**eyJhbGciOiJub25lIn0.**

**eyJzdWIiOiJ1c2VyMTIzIiwic2Vzc2lvbiI6ImNoNzJnc2IzMjAwMDB1ZG9jbDM2M**

**2VvZnkiLCJuYW1lIjoiUHJldHR5IE5hbWUiLCJsYXN0cGFnZSI6Ii92aWV3cy9zZXR0aW5ncyJ9.**

像上面展示的不安全的JWT可能适用于客户端使用。例如，如果会话ID是一个难以猜测的数字，并且其余数据仅由客户端用于构建视图，那么使用签名就是多余的。这些数据可以被单页Web应用程序用于构建带有用户的“漂亮”名称的视图，而无需在用户被重定向到其最后访问的页面时与后端交互。即使恶意用户尝试修改这些数据，也不会获得任何好处。

请注意紧凑表示中的尾随点号（.）。由于没有签名，它只是一个空字符串。尽管如此，点号仍然被添加。1

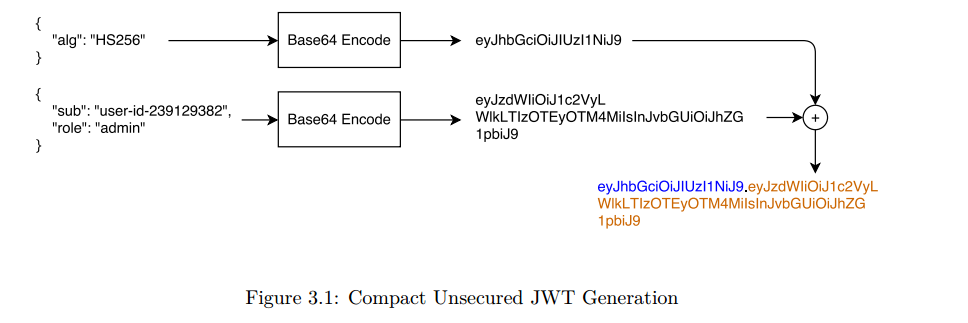
然而，在实际应用中，不安全的JWT是比较罕见的。

3.4 创建不安全的JWT

为了从头部和载荷的JSON版本得到紧凑表示，执行以下步骤：

1. 将头部作为其UTF-8表示的字节数组。JWT规范不要求在编码之前将JSON最小化或剥离无意义的字符（如空格）。
2. 使用Base64-URL算法对字节数组进行编码，去除尾随的等号（=）。
3. 将载荷作为其UTF-8表示的字节数组。JWT规范不要求在编码之前将JSON最小化或剥离无意义的字符（如空格）。
4. 使用Base64-URL算法对字节数组进行编码，去除尾随的等号（=）。
5. 连接生成的字符串，将头部放在首位，后跟一个“.”字符，然后是载荷。

在进行编码之前，必须对头部和载荷进行验证（验证是否存在所需的声明以及每个声明的正确使用）。



**3.4.1样例代码**

// URL安全的Base64变体

function b64(str) {

return new Buffer(str).toString('base64')

.replace(/=/g, '')

.replace(/\+/g, '-')

.replace(/\//g, '\_');

}

function encode(h, p) {

const headerEnc = b64(JSON.stringify(h));

const payloadEnc = b64(JSON.stringify(p));

return `${headerEnc}.${payloadEnc}`;

}

完整的示例在附带的样例代码文件coding.js中。

3.5 解析不安全的JWT

为了从紧凑序列化形式得到JSON表示，执行以下步骤：

1. 找到第一个点号“.”字符。取该字符之前的字符串（不包括该字符）。
2. 使用Base64-URL算法对字符串进行解码。结果是JWT的头部。
3. 取步骤1中点号之后的字符串。
4. 使用Base64-URL算法对字符串进行解码。结果是JWT的载荷。

生成的JSON字符串可以通过根据需要添加空格来“美化”。

**3.5.1样例代码**

function decode(jwt) {

const [headerB64, payloadB64] = jwt.split('.');

// 这些支持解析Base64的URL安全变体

const headerStr = new Buffer(headerB64, 'base64').toString();

const payloadStr = new Buffer(payloadB64, 'base64').toString();

return {

header: JSON.parse(headerStr),

payload: JSON.parse(payloadStr)

};

}

完整的示例在附带的样例代码文件coding.js中。