RFC 6749

国际互联网工程任务组(IETF) D. Hardt, Ed.

请求评论 6749 微软

作废 5849 2012 年 10 月

分类 标准跟踪

ISSN: 2070-1721

OAuth2.0 授权框架

抽象

OAuth2.0 授权框架使得第三方应用可以获得对 HTTP 服务的受限访问，或者在资源所有者与 HTTP 服务之间进行一个被批准的交互从而代表该资源所有者，或者通过允许第三方应用程序获得访问来代表该资源所有者。 该文档替换并作废在 RFC5849 描述的 OAuth1.0 协议。

本备忘录状况

本文档是一个标准的跟踪文档。本文档是 IETF 的一个产品。它代表了 IETF 社区的共识。它收到了公众评论、并且获得了 IESG 的认可。在 RFC5741 第 2 部分，可以获得因特网标准的最新信息。

关于该文档当前状况、任何勘误、如何进行反馈，都可以在 http://www.rfc-editor.org/info/rfc6749 进行查询。

版权公告

目 录

第一章 简介

在传统客户端/服务端认证模型中，客户端通过使用资源所有者的凭证在服务器上进行认证来请求服务器上一个访问受限的资源(被保护的资源)。为了给第三方应用提供受限资源的访问，资源所有者与第三方应用共享其凭证数据。这种做法会引起一引起问题与局限：

- 要求第三方应用存储资源所有者的凭证信息以备将来所用，典型应用场景就是以明文存储密码。
- 要求服务器提供密码认证，尽管密码本身存在安全缺陷。
- 第三方应用获得了过于宽泛的资源所有者的资源访问权限，让资源所有者没有任何能力限制使用时间、限制对受限资源子集的访问。
- 资源所有者，只有撤销对所有第三方应用的访问权限才能做到对单个第三方应用的访问权限撤销，并且，通过个性密码的方式来实现访问权限的撤销。
- 任何一个第三方应用的妥协都会导致终端用户的密码以及该密码所保护的数据的泄露。

OAuth2.0 解决了这些问题，它通过引入一个授权层，并从资源所有者中分离出客户端的角色。在 OAuth2.0 中，客户端请求由资源所有者控制的托管在资源服务器上的资源，这些资源由那些资源所有者不同的凭证进行发布。

客户端不再使用资源所有者的凭证去访问受保护的资源，它获得一个访问令牌，这个访问令牌表示一个指定的范围，生命周期，和其它的访问属性。访问令牌由授权服务器并经过资源所有者的批准颁发给第三方客户端。客户端使用这个令牌去访问托管在资源服务器上的受保护的资源。

例如，一个终端用户(资源所有者)可以给一个打印服务(客户端)授权访问她托管在照片共享服务(资源服务器)上的照片的访问权限，而不需要将她的用户名与密码共享给该打印服务。她直接与照片共享服务(授权服务器)信任的服务器进行认证，该服务器颁发给打印服务指定的凭证(访问令牌)。

发布作为一个信息文档的 OAuth1.0 协议(RFC5849)，是一个临时的社区努力的结果。这个标准的跟踪文档建立在 OAuth1.0 开发经验、使用案例、和 IETF

社区的扩展性要求之上的。该文档设计给 HTTP (RFC2616)使用。建立在任何不是 HTTP 协议之上的使用，不再范围之内。OAuth2.0 协议不向后兼容 OAuth1.0。这两个版本的协议在互联网上共存，一些实现可能同时支持这两个版本。然而，新的实现支持 OAuth2.0，OAuth1.0 仅仅用来支持现在的部署，是本文的目标。OAuth2.0 协议共享了 OAuth1.0 协议非常少的实现细节。熟悉 OAuth1.0 的实现厂商在没有对它的结构与细节的条件下，应该熟悉该文档。

1.1 角色

OAuth2.0 定义了 4 个角色：

资源所有者

可以对受保护资源进行授权的实体。当资源所有者是一个人时，它特指一个终端用户。

资源服务器

资源服务器托管受保护资源，可以接收带有访问令牌的受保护资源的请求并进行响应。

客户端

在经过资源所有者的授权之后，代表资源所有者创建受保护资源请求的应用程序。关键字“客户端” 并不暗示任何一个特定的实现特性，尽管，应用程序执行在一个服务器上、桌面上或者其它设备上。

授权服务器

授权服务器给成功认证的资源所有者和获取授权来颁发访问令牌。

在授权服务器与资源服务器之间的交互超出了本文档的范围。授权服务器可能与资源服务器可能是同一台服务器，或者是两台。一个单独的授权服务器可以给多个资源服务器颁发访问令牌。

1.2 协议流

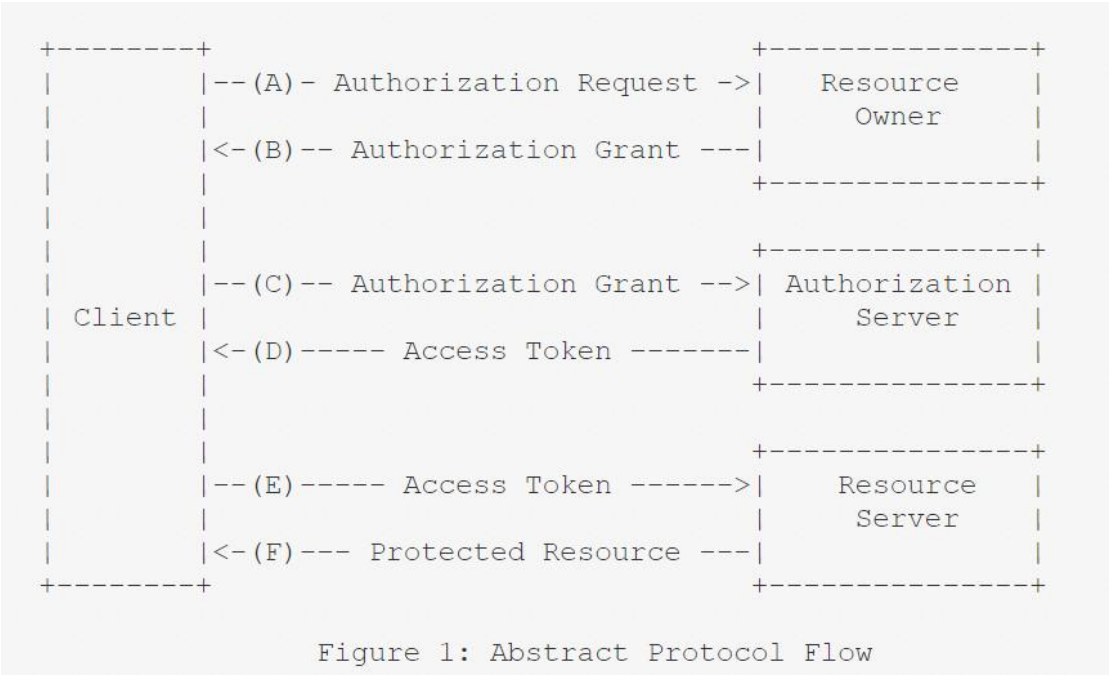

Figure 1: Abstract Protocol Flow

在图 1 的抽象 OAuth2.0 流插件描述了 4 种角色之间的交互过程，包含以下步骤：

(A) 客户端请求来自于资源所有者的授权。授权请求可以直接发送给资源所有者(如上所示)，更好地是间接地通过授权服务器作为一个中介进行转发。

(B) 客户端接收到授权许可，该授权许可是一个代表着资源所有者的授权的一个凭证，表示使用文档中描述的 4 种许可类型之一或者使用扩展的许可类型。授权许可类型依赖于客户端请求许可的方法和授权服务器支持的类型。

(C) 客户端通过与授权服务器进行论证，并携带授权许可来获取访问令牌。

(D) 授权服务器论证客户端并校验授权许可，如果许可有效，就颁发一个访问令牌。

(E) 客户端携带访问令牌，向资源服务器请求受保护的资源。

(F) 资源服务器校验访问令牌，如果令牌有效，则处理请求。

客户端从资源所有者那里获取授权许可一个好的方法是，使用授权服务器作为一个中介，正如 4.2 部分中的图 3 所示。

1.3 授权许可

授权许可是代表着资源拥有者的凭证（访问受保护资源），该凭证被客户端用来获得访问令牌。本文定义了 4 种许可类型：授权码、简单、密码、客户端，也支持定义新的类型的扩展机制。

1.3.1 授权码模式

使用授权服务器作为客户端与资源所有者之间的中介来获取授权码。不是向资源所有者直接请求授权，客户端引导资源所有者到一个授权服务器上(在 RFC2616 是描述的用户代理)，反过来，携带授权码引导资源所有者返回到客户端。

在携带授权码引导资源所有者到客户端之前，授权服务器对资源所有者进行认证，并获取授权。因为资源所有者仅仅与授权服务器进行认证，资源所有者的凭证永远也不会与客户端进行共享。

授权码提供了一上安全上的便捷，例如对客户端进行认证，访问令牌直接传输到客户端而不通过资源所有者的用户代理，而不会暴露给其他人，包括资源所有者。

1.3.2 隐含模式

隐含模式是一种简化的授权码模式，为用浏览器作为客户端实现而优化的，它使用脚本语言，如 JavaScript。在隐含模式中，不再给客户端颁发授权码，直接给客户端颁发访问令牌(资源所有者的结果)。这种授权类型是隐含的，由于没有中间凭证(例如授权码)颁发(被用来获取访问令牌)。

在隐含模式中，颁发一个访问令牌时，授权服务器并不认证客户端。在某些情况下，通过重定向 URI 来验证客户端身份，为了传输给客户端的访问令牌。通过访问资源拥有者的用户代理，访问令牌可能暴露给资源拥有者或者其它应用程序。

隐含模式提高了一些客户端的响应与效率（例如实现中浏览器中的应用），由于它减少了要求用来获取访问令牌的通信次数。但是，应该权衡这种便利，对

于使用隐含模式的安全含义，例如在 10.3 和 10.16 部分描述的那些，尤其是当授权码模式可用时。

1.3.3 密码模式

资源所有者的密码凭证(用户名和密码)可以直接用作授权许可来获取访问令牌。 这种凭证应该仅用在这样的情况，在资源所有者与客户端之间有高信任度(例如，客户端是设备操作系统一部分或者高权限的应用)，并且其他的授权类型不可用（例如授权码模式）。

尽管这种授权模式要求客户端直接访问资源所有者的凭证，资源所有者凭证也是只用于一次请求，用来交换一个访问令牌。这种授权模式，通过使用一个长生命周期的令牌或者刷新令牌，可以消除客户端存储资源所有者凭证的需要。

1.3.4 客户端模式

客户端凭证(或者其它客户认证的形式)用作一种授权许可，当授权范围被限制在受保护资源下。当客户端仅仅代表自己时(客户端也是资源所有者)，或者要求对受保护资源访问时(基于授权服务器的授权)，客户端凭证模式被用途一种典型的授权许可类型。

1.4 访问令牌

访问令牌是用来访问受保护资源的凭证。访问令牌是一个字符串，代表着颁发给客户端的一个授权。访问令牌通过对客户端不透明。访问令牌表示特定的范围，访问的时间，它由资源所有才许可，并被资源服务器和授权服务器强制的。

令牌可能表示一种身份，用来获取授权令牌或者以一种可验证的方式，自包含授权信息。其它的论证凭证，超出本文的范围，可能被要求为了客户端使用令牌。

访问令牌提供提供了一个抽象层，替换了不同的授权结构（例如，用户名与密码），通过一个单一的令牌，可以被资源服务器理解。这个抽象层使得颁发令牌更受限比用来获取他们的授权许可，正如，消除资源服务器需要理解许多不同

的认证方法。

访问令牌基于资源服务器的安全需求，可以有不同的格式、结构、使用方法(例如加密属性)。用于访问受保护资源的访问令牌属性和方法超出了本文的范围，他们在伙伴规范 RFC6750 中定义。

1.5 刷新令牌

Refresh Token 是用来获取 Access Token 的凭证。Refresh Token 由授权服务器来颁发给客户端，当当前 Access Token 变得不可用或者过期，或者为了获取额外的 Access Token，或者减少授权范围(Access Token 可能生命周期更短，权限更少比资源所有者授权的)。由授权服务器来决定是否颁发 Refresh Token。如果授权服务器颁发 Refresh Token，该 Token 包含在 Access Token 里面（在图 1 步骤 D 中）。

Refresh Token 是一个字符串，代表着许可的授权，该授权由资源所有者许可给客户端。该字符串通过对客户端来说是不透明的。该 Token 表示一个身份，用来获取授权信息的。与 Access Token 不同，Refresh Token 仅仅与授权服务器一起使用，永远不会发送给资源服务器。

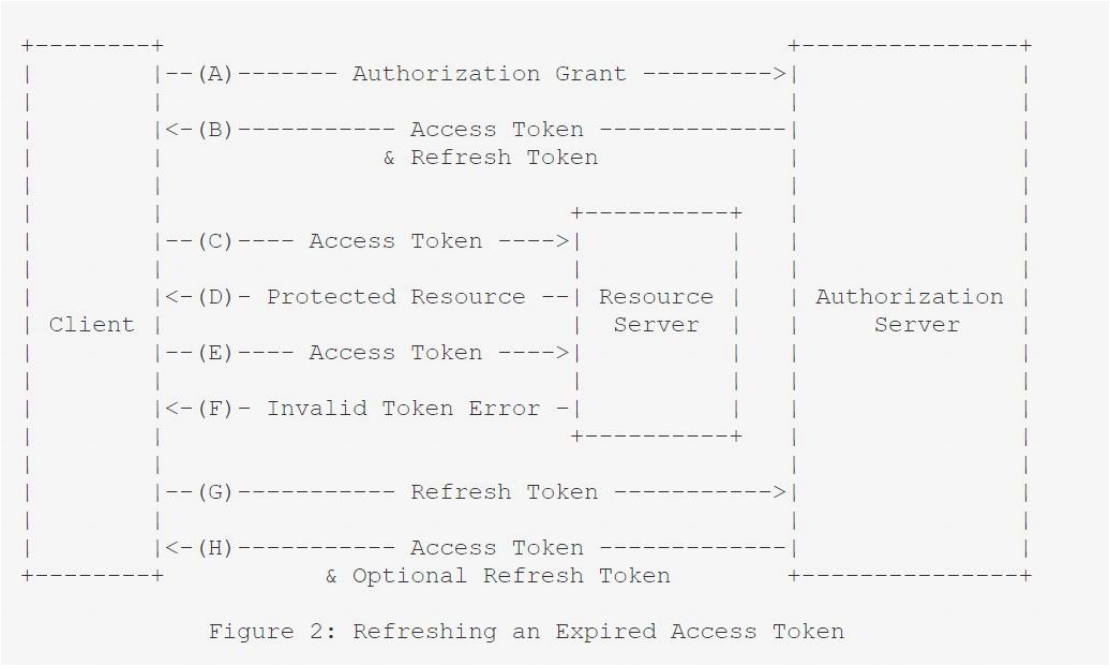

Figure 2: Refreshing an Expired Access Token

图 2 中的流程图包含以下几个步骤：

(A) 客户端携带一个授权许可，通过授权服务器上进行认证，发出一个 Access Token

请求。

(B) 授权服务器认证客户端，并校验授权许可，如果有效，则颁发 Access Token 和 Refresh Token。

(C) 客户端携带 Access Token，向资源服务器，请求一个受保护的资源。

(D) 资源服务器校验 Access Token，如果有效，则处理请求。

(E) 重复步骤(C)与步骤(D)，直到 Access Token 过期。如果客户端知道 Access Token 过期，直接跳转到步骤(G)；否则，发出另一个对受保护资源的请求。

(F) 由于 Access Token 无效，资源服务器向客户端返回 Token 失效错误。

(G) 客户端携带 Refresh Token，在授权服务器上进行认证，请求一个新的 Access Token。客户端认证需求基于客户端类型和授权服务器策略。

(H) 授权服务器认证客户端，并校验 Refresh Token，如果有效，则颁发一个新的 Access Token(可选地，一个新的 Refresh Token)。

步骤(C)，(D)，(E)和(F)如第 7 章描述，超出了本文档的范围。

1.6 TLS 版本

在本文中，无论什么时候用到传输层安全协议时，基于广泛的部署和已知的安全性漏洞，TLS 最合适的版本都会随着时间而变化。在写本文档时，TLS 1.2 是最新的版本，有受限的部署，并可能不太容易实现。TLS1.0 是最广泛部署的版本，并且提供最广泛的互操作性。

实现也可能支持额外的 TLS 机制来满足安全方面的需求。

1.7 HTTP 重定向

本文广泛使用了 HTTP 重定向，在本文中，客户端或者授权服务器引导资源所有者的用户代理重定向到另一个目标地址。虽然本文中案例演示了 HTTP 302 状态码的使用，但是任何其它可以使用的方法(用来实现重定向)也允许使用，并认为是一种实现。

1.8 互操作性

OAuth2.0 提供了一个功能丰富的授权框架，它拥有良好的安全属性。然后，作为一个功能丰富，高度扩展的，有着许多可选组件的框架，独立地，该文档很可能生产许多的不可互操作的实现。

此外，该文档遗留了一些要求的组件，部分没有定义或者完全没有定义(例如，客户端注册，授权服务器功能，终端发现)。没有这些组件，针对一个特定的授权服务器和资源服务器，客户端需要手动并按照规范进行配置，为了进行互操作。

该框架设计带着明确的期望，未来的工作将定义规范和扩展，为了实现 web 规模的互操作性。

1.9 符号约定

本文中出现的关键词"MUST"、"MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY","OPTIONAL"与 RFC2119 中解释一致。

本文使用 RFC5234 中的 ABNF 符号。此外，URI 引用规则包含在 RFC3986 中的 URI。

某些安全相关条款在 RFC4949 中可以理解。这些条款包含，但不限于，“攻击”，“认证”，“授权”，“证书”，“保密”，“凭证”，“加密”，“身份”，“签名”，“签名”，“信任”，“验证”，“验证”。

除非另有说明，所有的协议参数名与参数值都是大小写敏感。

第二章 客户端注册

在初始化协议之前，客户端向授权服务器进行注册。客户端向授权服务器注册的方法，超出了本文档描述的范围，但通常涉及到终端用户以 HTML 形式进行注册。

客户端注册并不要求客户端与授权服务器之间直接交互。当授权服务器支持时，注册可以依赖于其它方法建立信任，并且获得客户端属性(例如，重定向 URI，客户端类型)。例如，可以通过自我颁发，第三方颁发机构或者授权服务器在一个可信频道上实现自动发现，这些方法来实现注册。当注册一个客户端时，客户端开发者应该：

- 指定客户端类型，如 2.1 所描述。
- 提供客户重定向 URIS，如 3.1.2 描述。
- 包含任何其他由授权服务器要求的信息（例如，应用名称，网站，描述，Logo，同意法律条款）。

2.1 客户端类型

OAuth2.0 将客户端定义成两种类型，基于客户端与授权服务器安全认证的能力(例如，维护客户端凭证的保密性的能力)。

保密的

客户端可以维护他们凭证的保密性(例如，客户端实现了对客户端凭证的受限的安全的访问)，或者可以通过其它途径来实现安全的客户端认证。

公开的

客户端不能维护他们凭证的保密性(例如，客户端运行在资源所有者所使用的设备上，例如一个本地应用，或者基于 web 浏览器的应用)，或者不能通过其它途径来实现安全的客户端认证。

客户端类型的设计是基于授权服务器对于安全认证的定义和对客户端凭证暴露可接受程度来考虑的。授权服务器不应该对客户端类型进行假设。

客户端可能由多个分布式的组件来实现的，每个客户端有不同的客户端类型，和安全上下文(例如，拥有基于服务器组件和公开基于浏览器组件组成的一个分布式客户端)。如果授权服务器对于这些的客户端不提供支持，或者关于他们的注册不提供保证，该客户端应该将每一个组件作为一个单独的客户端进行注册。

该规范围绕以下客户端框架进行设计：

Web 应用程序

Web 应用程序是运行在一个 Web 服务器上的保密的客户端。资源所有者通过 HTML 接口，它由资源所有者使用的设备上的用户代理来进行渲染，访问客户端。客户端凭证，与任何颁发给客户端的 Access Token 一样，存储在 Web 服务器上，并不暴露给资源所有者，或者资源所有者可以进行访问。

基于用户代理的应用

基于用户代理的客户端是公开的客户端，在这种客户端上，客户端代码从 web 服务器上下载，并在用户代理上进行执行(例如，web 浏览器)，这些用户代理是由资源所有者使用。协议数据和凭证对于资源所有者来说，是非常容易访问(通常可见)。由于，这些应用程序驻留在用户代理中，当请求认证时，他们可以无缝利用用户代理功能。

原生应用

原生应用是一个公开的客户端，它安装和执行在资源所有者所使用的设备上。协议数据和凭证对于资源所有者来说是可以访问的。普遍认为包含中这种应用程序中的认证凭证是可以被提取的。另一方面，运态颁发的凭证，例如 Access Token 或者 Refresh Token 可以达到保护的可接受级别。最低限度，这些凭证可以被保护免于与恶意服务器的交互。在一些平台上，这些凭

证，可能免于在相同设备上其它应用程序的攻击。

2.2 客户端标识

授权服务器给已经注册的客户端颁发一个客户商身份标识，它是一个唯一的字符串，代表提供给客户端的注册信息。客户端标识不是一个密钥；它暴露给资源所有者，并且不准单独使用。客户端标识对于授权服务器来说是唯一的。

该文档并未定义客户端标识字符串的大小。客户端应该避免假定客户端标识符的大小。授权服务器应该记录它所颁发的任何标识符的大小。

2.3 客户端认证

如果客户端类型是保密的，客户端与授权服务器建立一种适用于授权服务器的认证方法。授权服务器可能接收任何形式的客户端认证请求，这些请求满足授权服务器的安全需求。

保密客户端通常被颁发一组客户端凭证，用来与授权服务器进行认证(例如，密码，公私密钥对)。

授权服务器与可能与公开的客户端建立一个客户端认证方式。然而，考虑到对客户端进行标识，授权服务器不准依赖于公开客户端的认证。

客户端不准在一个请求中，使用超过一种认证方式。

2.3.1 客户端密码

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

例如(额外的换行符仅仅用于显示)：

```
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3
```

可选，授权服务器可能支持包含中请求体的客户端凭证，包含以下参数：

client_id:

必填。在 2.2 小节注册过程中，颁发给客户端的客户端标识符。

client_secret:

必填。如果客户端密钥是一个空字符吕，客户端密钥可能忽略这个参数。

在请求体中，使用以上两个参数来包含客户端凭证并不推荐这种做法。这些参数只能包含中请求体中，不准包含在 URI 中。

举例说明，刷新 Access Token 的一个请求(第 6 章中)，使用了 body 参数(多余的换行符仅用于显示目的)：

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
&client_id=s6BhdRkqt3&client_secret=7Fjfp0ZBr1KtDRbnfVdmIw
```

当使用密码认证来发送请求时，授权服务器必须要求使用第 1.6 节中的 TLS。

由于客户端认证方法涉及密码，授权服务器必须保护任何终端利用它来实施暴力攻击。

2.3.2 其它认证方法

授权服务器可能支持任何适合的 HTTP 认证模式，只要匹配安全需求。当使用其它的认证方法时，授权服务器必须定义在客户端标识符(注册记录)与授权模式之间映射关系，

2.4 未注册客户端

该文档并没有排除未注册客户端的使用。然而，这些客户端的使用超出了该文档的范围，并且要求额外的安全分析与互操作性影响的考虑。

第三章 协议端点(未开始)

在初始化协议之前，客户端向授权服务器进行注册。客户端向授权服务器注册的方法，超出了本文档描述的范围，但通常涉及到终端用户以 HTML 形式进行注册。

客户端注册并不要求客户端与授权服务器之间直接交互。当授权服务器支持时，注册可以依赖于其它方法建立信任，并且获得客户端属性(例如，重定向 URI，客户端类型)。例如，可以通过自我颁发，第三方颁发机构或者授权服务器在一个可信频道上实现自动发现，这些方法来实现注册。当注册一个客户端时，客户端开发者应该：

- 指定客户端类型，如 2.1 所描述。
- 提供客户重定向 URIS，如 3.1.2 描述。
- 包含任何其他由授权服务器要求的信息（例如，应用名称，网站，描述，Logo，同意法律条款）。

3.1 授权端点

OAuth2.0 将客户端定义成两种类型，基于客户端与授权服务器安全认证的能力(例如，维护客户端凭证的保密性的能力)。

保密的

客户端可以维护他们凭证的保密性(例如，客户端实现了对客户端凭证的受限的安全的访问)，或者可以通过其它途径来实现安全的客户端认证。

公开的

客户端不能维护他们凭证的保密性(例如，客户端运行在资源所有者所使用的设备上，例如一个本地应用，或者基于 web 浏览器的应用)，或者不能通过其它途径来实现安全的客户端认证。

3.1.1 响应类型

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.1.2 重定向端点

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.1.2.1 端点请求保密性

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.1.2.2 注册要求

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.1.2.3 动态配置

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.1.2.4 无效端点

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.1.2.5 端点内容

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.2 Token 端点

授权服务器给已经注册的客户端颁发一个客户商身份标识，它是一个唯一的字符串，代表提供给客户端的注册信息。客户端标识不是一个密钥；它暴露给资源所有者，并且不准单独使用。客户端标识对于授权服务器来说是唯一的。

该文档并未定义客户端标识字符串的大小。客户端应该避免假定客户端标识符的大小。授权服务器应该记录它所颁发的任何标识符的大小。

3.2.1 客户端认证

拥有一个客户端密码的客户端可能使用在 RFC 2617 中定义的 HTTP 基本认证模式向授权服务器进行认证。客户端标识符使用 URL 编码算法进行编码(附录 B)，编码结果作为用户名；客户端密码使用同样的算法进行编码，并作为密码。授权服务器必须支持 HTTP 基本的认证模式，用于认证被颁发一个客户端密码的客户端。

3.3 Access Token 范围

如果客户端类型是保密的，客户端与授权服务器建立一种适用于授权服务器的认证方法。授权服务器可能接收任何形式的客户端认证请求，这些请求满足授

权服务器的安全需求。

保密客户端通常被颁发一组客户端凭证，用来与授权服务器进行认证(例如，密码，公私密钥对)。

授权服务器与可能与公开的客户端建立一个客户端认证方式。然而，考虑到对客户端进行标识，授权服务器不准依赖于公开客户端的认证。

客户端不准在一个请求中，使用超过一种认证方式。

```
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3
```

可选，授权服务器可能支持包含中请求体的客户端凭证，包含以下参数：

client_id:

必填。在 2.2 小节注册过程中，颁发给客户端的客户端标识符。

client_secret:

必填。如果客户端密钥是一个空字符吕，客户端密钥可能忽略这个参数。

在请求体中，使用以上两个参数来包含客户端凭证并不推荐这种做法。这些参数只能包含中请求体中，不准包含在 URI 中。

举例说明，刷新 Access Token 的一个请求(第 6 章中)，使用了 body 参数(多余的换行符仅用于显示目的)：