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

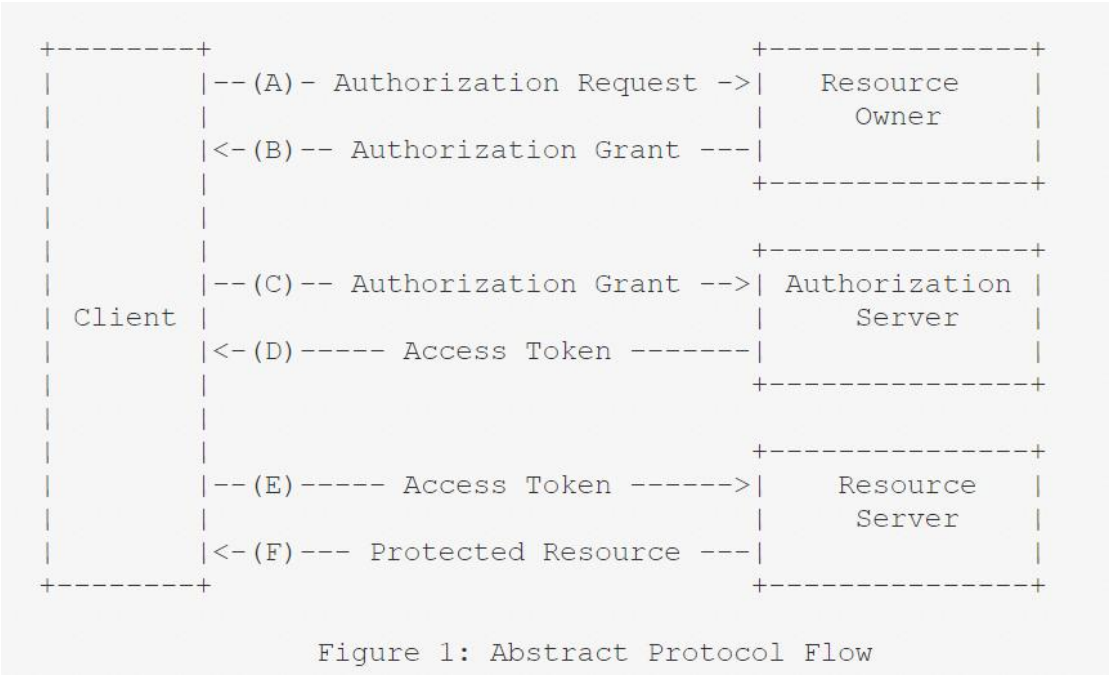


Figure 1: Abstract Protocol Flow

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。

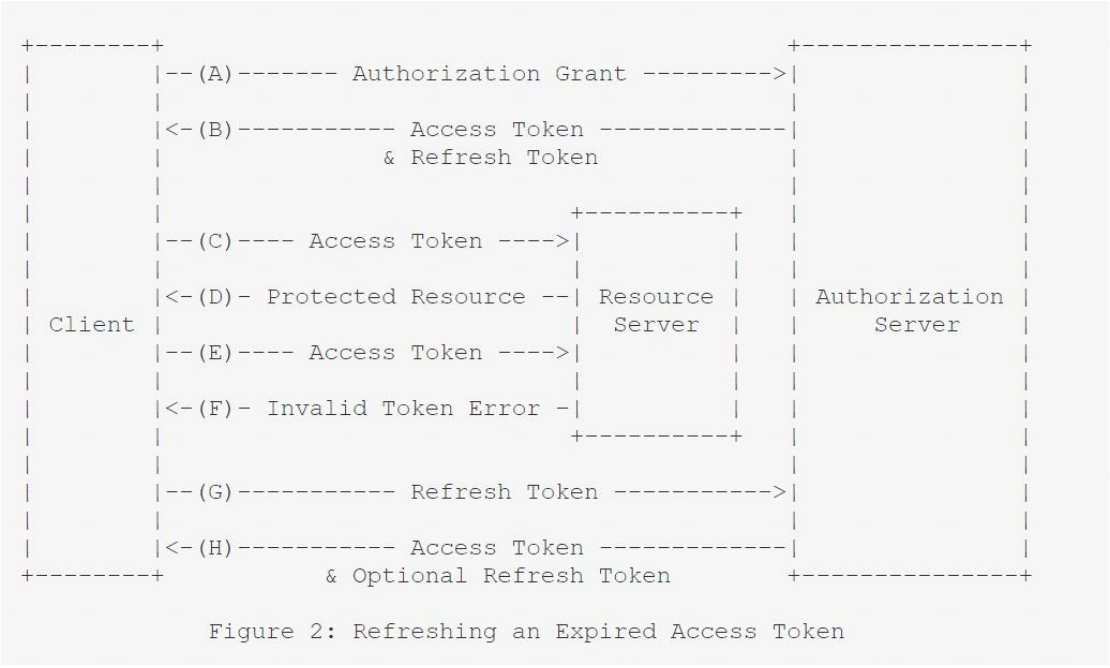


Figure 2: Refreshing an Expired Access Token

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

授权过程利用两个授权服务器端点(HTTP 资源)：

- 授权端，给客户端通过用户代理重定向从资源所有者获取授权。
- 令牌端，客户端用来交换访问令牌的授权许可，通常伴随着客户端认证。

还有一个客户端端点：

- 重定向端，授权服务器用来向客户端返回包含授权凭证的响应信息，客户端是通过资源所有者的用户代理来请求的。

3.1 授权端点

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

客户端获取授权端点位置的方法超出了本文档描述的范围，但是该位置通常在服务文档中提供。

该授权端 URI 可能包含一个 URLEncoded 格式化的(附录 B)的查询组件(RFC3986 第 3.4 节)，当增加额外查询参数时，必须获取该查询组件。

由于发送给该授权端的请求会导致用户认证和明文凭证的传输(在 HTTP 响应中)，因此，授权服务器必须要求使用在 1.6 节中提到的 TLS。

授权服务器对于授权端必须支持 GET 方法，可能也支持 POST 方法。

没有值的参数，必须被视为从请求中忽略。授权服务器必须忽略不可识别的请求参数。请求和响应参数不能包含多次。

3.1.1 响应类型

授权端点只被授权码模式和简易模式使用。客户端向授权服务器使用以下参数通过期望的授权许可类型：

响应类型(response_type)

必填。该值要么是 code(用来请求一个授权码在 4.1.1 中描述)，要么是 token

（用来请求一个访问令牌，在 4.2.1 中描述的简易模式），或者是在 8.4 中描述的一个已注册的扩展值。

扩展的响应类型可能包含一个空格分隔的值列表，这些值的顺序没有影响(例如响应类型”a b”与”b a”相同)。这些复合的响应类型的意义在相应的规范文档中定义。

3.1.2 重定向端点

授权服务器在完成与资源所有者的交互之后，它将资源所有者的用户代理重定向至客户端应用上。授权服务器将用户代理重定向到客户端应用的重定向端上，也就是在客户端应用注册过程或者发送授权请求与授权服务器建立的。

重定向端点的 URI 必须是一个绝对的 URI，在 4.3 部分 RFC3986 中定义的。该端 URI 可能包含一个 URLEncoded 格式化的查询组件(附录 B)，在增加客户的查询参数时，必须保留。该端 URI 不能包含碎片组件。

3.1.2.1 端点请求保密性

当请求的响应类型是授权码、令牌、或者重定向请求会导致一些敏感数据在开放网络上进行传输时，重定向端要求使用在 1.6 中描述的 TLS。本文并不强制使用 TLS，因为在撰写本文档时，考虑到对于许多客户端应用来核准，要求客户端部署 TLS 是一件比较麻烦的事情。如果不使用 TLS，授权服务器应该提醒资源所有者该重定向是不安全的（在授权请求过程中，显示一条消息）。

缺少 TLS 可能对于客户端应用的安全与被授权访问权限的资源有很严重的影响。当授权过程被委托用于终端用户的论证(第三方登录),TLS 的使用特别重要。

3.1.2.2 注册要求

授权服务器必须要求以下客户端应用注册他们的重定向端点：

- 公开的客户端。
- 使用简单授权模式的保密的客户端

授权服务器要求所有的客户端应用在使用授权端点之前，注册它们的重定向端点。

授权服务器应该要求客户端应用提供完整的重定向 URI(客户端可能使用 state 请求参数来达到每次自定义请求的效果)。如果，不太可能要求完整的 URI 注册，授权服务器应该要求注册 URI 的模式，授权注册，路径(当请求授权时，允许客户端应用动态改变查询组件)。

客户端服务器可能允许客户端应用注册多个重定向端点。

缺少对于重定向 URI 的注册可能引起攻击者将授权端点作为一个开放的重定向器，如 10.15 中描述。

3.1.2.3 动态配置

如果注册了多个重定向地址，如果重定向地址一部分被注册，如果没有注册重定向地址，客户端应用必须在授权请求中，包含请求参数 redirect_uri 作为重定向 URI。

当在授权请求中包含了重定向 URI，授权服务器必须比较与匹配接收到的访参数值和之前注册过的重定向值，在第 6 部分中描述的。如果客户端注册信息中包含了完整的重定向 URI，授权服务器必须使用简单字符串比较算法(在 6.2.1 中 RFC3968)比较两个重定向 URI 值。

3.1.2.4 无效端点

如果授权请求由于缺失重定向 URI，或者无效，或者不匹配，而未通过验证，授权服务器应该向资源所有者通知错误信息，并且，不允许将用户代理重定向到该无效的重定向 URI。

3.1.2.5 端点内容

发送给客户端应用端点的请求会导致一个 HTML 文档的响应，由用户代理进行处理。如果重定向请求被直接处理成一个 HTML 响应，包含在 HTML 文档中的任何脚本都可以拥有重定向访问权限和其包含的凭证进行执行。

3.2 Token 端点

Token 端点被客户端用来获取访问令牌，展示授权许可或者刷新令牌。令牌端点伴随每一次的授权许可而使用，除了简单授权模式(由于直接颁发令牌)。

客户端获取 Token 端点访问路径的方法超出了本文档的范围，但是有服务描述文档中提供。

该 Token 端点可能包含 URL 编码格式化的查询组件(3.4 节中 RFC3986)，该查询组件在增加额外的查询参数时，必须保留。该 Token 端点不准包含碎片组件。

由于，发送给 Token 端点的请求，会导致明文凭证的传输(在 HTTP 请求与响应中)，授权服务器必须要求在 1.6 节中描述的 TLS，当向 Token 端点发送请求时。

当客户端发送 Access Token 请求时，必须使用 POST 方法。

没有值的参数，必须被视为可以从请求中进行忽略。授权服务器必须忽略不可识别的参数。请求和响应参数只能包含一次。

3.2.1 客户端认证

当向 Token 端点发送请求时，被颁发客户端凭证的保密客户端和其他客户端必须向授权服务器进行认证，如 2.3 节中描述。客户端认证用于以下场景：

- 强制将颁发给客户端的刷新 token 与授权码与客户端进行绑定。当授权码在一个不安全的通道中进行传输或者重定向 URI 并非完整路径注册时，客户端认证是比较重要的。
- 从被支持的客户端中恢复，通过禁用客户端或者个性其凭证令牌，这样就可以阻止攻击者滥用被盗的 Refresh Token。修改一组客户端凭证比撤销整组 Refresh Token 更快。
- 实现认证管理最佳实践是周期性地更换凭证。更换整组 Refresh Token 比较难，但是只更换单组凭证却是非常简单。

客户端在向 Token 端点发送请求时，可能使用 client_id 来标识自己。

3.3 Access Token 范围

授权端点与令牌端点允许客户端指定访问请求的范围，通过使用请求参数 scope。反过来，授权服务器使用 scope 响应参数来通知客户端 token 的范围。

scope 参数值是由空格分隔的值列表，大小写敏感的字符串。该字符串由授权服务器定义。如果该值包含多个空格分隔的字符串，它们之间的顺序没有关系，每一个字符串增加了一个请求的范围。

授权服务器可能完全或者部分忽略客户端的 scope 请求，基于授权服务器的策略或者资源拥有者的指令。如果被颁发的 Access Token 中的 scope 不同于客户端请求中的 scope，授权服务器必须包含 scope 参数通知客户端实际被授权的 scope.

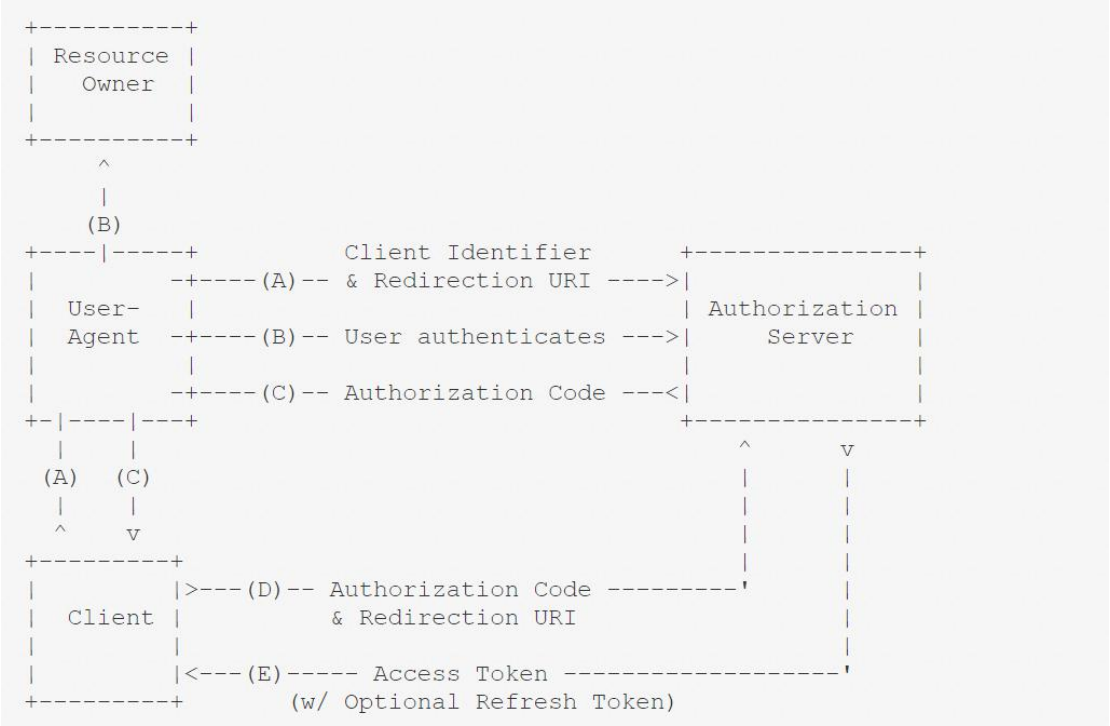
在客户端请求授权时，如果客户端忽略了 scope 参数，授权服务器必须要么以一个预先定义的值来处理请求，要么不通过该请求，表明是一个无效的 scope。授权服务器应该用文档来注明 scope 要求，和默认值。

第四章 获得授权

为了请求一个 Access Token，客户端从资源所有者那里获取授权。授权以授权许可的形式来表示，客户端使用这个授权许可来请求 Access Token。OAuth2.0 定义了 4 种许可类型：授权码模式、隐含模式、密码模式、客户端模式。它也提供了一个扩展机制，用于定义其他的许可类型。

4.1 授权码模式

授权码模式用来获取 Access Token 和 Refresh Token，并被优化于保密客户端。由于这是一个重定义的流程图，客户端必须可以与资源所有者的用户代理进行交互(通常是一个 Web 浏览器)，并且能接收来自于授权服务器的接下来的请求(通过重定向)。



注意：描述步骤的行 A B C 由于通过了用户代理，被分成了两部分。

(A) 客户端应用通过重定向资源所有者的用户代理到授权服务器，来初始化数据流。客户端包含了客户端标识符，请求范围，局部状态，和重定向 URI，一旦授权服务器授权通过或者拒绝，授权服务器会向客户端用户代理响应给客户端。

(B) 授权服务器通过用户代理对资源所有者进行认证，并且无论资源所有者授权通过或者拒绝，都会建立连接。

(C) 假定资源所有者授权通过，授权服务器引导用户代理重定向到之前提供的重定向 URI。重定义 URI 包含授权码和之前客户端提供的 state。

(D) 客户端应用向授权服务器的 Token 端请求一个 Access Token，请求参数包含上一步接收到的授权码 Authorization Code。当发送请求时，客户端向授权服务器进行认证。客户端包含重定向 URI 以用于获取授权码。

(E) 授权服务器对客户端应用进行认证，校验授权码，并确保接收到的重向向 URI 与步骤(C)中的 URI 相匹配。如果授权码有效，授权服务器会响应一个 Access Token，可选地，一个 Refresh Token。

4.1.1 授权请求

客户端通过向查询组件增加以下参数来构造一个请求的 URI，使用 URL 编码格式，附录 B：

响应类型(response_type)

必填。值固定为 code。

客户端 ID(client_id)

必填。2.2 小节中描述的客户端标识符。

重定向 URI(redirect_uri)

可选。3.1.2 小节中描述。

范围(scope)

可选。访问请求的范围在 3.3 小节中描述。

状态(state)

推荐。一个不透明值，被客户端用来维持在请求与回调之间的状态。授权服务器当重定向用户代理到客户端时，会带上这个参数。该参数应该用来阻止跨站请求伪造，如 10.12 描述。

客户端引导资源所有者到一个构造的 URI，通过一个 HTTP 重定向响应，或者通过其他可用方式到该 URI。

例如，客户端引导用户代理构造使用 TLS 的 HTTP 请求(多余的行分隔符仅用于显示)。

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

授权服务器会校验请求，以确保所有的请求参数出现并且有效。如果请求有效，授权服务器会认证资源所有者，并且获取一个授权决定结果(通过询问资源所有者或者通过其他方法建立批准机制)。

4.1.2 授权响应

如果资源所有者授权了这个访问请求，授权服务器会颁发一个授权码，并且通过增加以下参数到查询组件上来将其传递给客户端，这些参数以 URL 编码，见附录 B：

Code

必填。授权码由授权服务器生成。在授权码被颁发之后，授权码必须在很短的一段时间之内过期，以减少泄露的风险。推荐 10 分钟作为授权码最长的生命周期。客户端只能使用授权码一次。如果授权码使用超过 1 次，授权服务器必须拒绝请求，并且撤销所有基于该授权码颁发的所有令牌。授权码被绑定到客户端标识符和重定向 URI 上。

State

必填。如果在客户端的授权请求中，出现了 state 参数。

例如，授权服务器通过发送以下 HTTP 响应，来重定向用户代理。

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA
          &state=xyz
```

客户端必须忽略不可识别的响应参数。该文档并未定义 Authorization code 字符串的长度。客户端应避免假定 Authorization code 字符串的长度。授权服务器应该在文档中注明它所颁发的授权码的大小。

4.1.2.1 错误响应

如果请求失败，是因为，缺少重定向 URI，无法的重定向 URI，或者不匹配的重定向 URI，或者如果客户端标识符缺失、无效，授权服务器应该通知资源所有者错误信息，并且，不允许自动重定向用户代理到这个无效的 URI。

如果资源所有者拒绝访问请求，或者请求因为某些非缺失、无效重定向 URI 原因而失败时，授权服器通过向查询组件增加以下 URL 编码格式的参数，来通知客户端，附录 B:

error

必填。单个的 ASCII 码错误码，来自于以下值：

invalid_request：请求缺少必要参数，包括无效的参数值，同一个参数出现多次，或者格式错误。

unauthorized_client：客户端没有授权使用该方法来请求一个授权码。

access_denied：资源所有者或者授权服务器拒绝该请求。

unsupported_response_type：授权服务器不支持通过这种方法来获取授权码。

invalid_scope：scope 是无效的，未知的，或者格式错误。

server_error：授权服务器遇到未知错误，从而不能处理请求。(该错误码是必要的，因为 500 服务器内部错误的 HTTP 响应状态码不能通过一个重定向来返回给客户端)。

temporarily_unavailable：授权服务器由于临时过载或者服务器维护，当前不能处理请求(该错误码是必要的，因为 503 服务不可达的 HTTP 响应状态码不能通过一个重定向来返回给客户端)。

error_description

可选。人可读的 ASCII 文本，提供了额外的信息，被用来客户端开发者理解错误。error_description 的值不能包含超出%x20-21 / %x23-5B / %x5D-7E 的字符。

error_uri

可选。一个 URI，标识一个人可读的 web 页面，提供了额外错误信息，被用来客户端开发者理解错误。error_uri 的参数值必须遵守 URI 参考语法，不能包含超出%x20-21 / %x23-5B / %x5D-7E 的字符。

state

如果在客户端的授权请求参数中出现了 state 参数，则必填。该值与从客户端接收过来的值相同。

例如，授权服务器通过发送以下 HTTP 响应来重定向用户代理：

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=access_denied&state=xyz
```

4.1.3 Access Token 请求

客户端向 Token 端点发送一个请求，通过发送以 URL 编码的参数，附录 B，在 HTTP 请求体中，以 UTF-8 编码。

grant_type

必填。固定为 authorization_code。

code

必填。从授权服务器接收的 authorization code。

redirect_uri

必填。如果在授权请求中包含了 redirect_uri 参数，在 4.1.1 中描述，则他们的值必须完全相同。

client_id

如果客户端没有向授权服务器认证，如 3.2.1 中描述，则必填。

如果客户端类型是保密类型的，或者客户端被颁发客户端凭证(或者赋予认证要求)，则客户端必须向授权服务器认证，如 3.2.1 描述。

例如，客户端使用 TLS 来构造请求(多余行仅用于显示目的)：

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

授权服务器必须做到：

- 对于保密的客户端，或被颁发客户端凭证的客户端(或者其它认证要求)，要求客户端进行认证。
- 如果客户端认证被要求，则认证客户端
- 确保授权码被颁发给认证过的保密的客户端，或如果客户端是公开的，确保授权码被颁发给请求参数中的 client_id。
- 验证授权是否有效
- 如果 redirect_uri 参数在初始的授权请求中出现，如 4.1.1 中描述，确保 redirect_uri 出现，并且两个值要求完全相同。

4.1.4 Access Token 响应

如果 Access Token 请求是有效的，并且被授权的，授权服务器会颁发一个 Access Token 和一个可选的 Refresh Token，5.1 节中描述。如果客户认证失败或者该请求无效，授权服务器会如 5.2 小节中描述的，返回一个错误的响应。

一个成功的响应案例：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

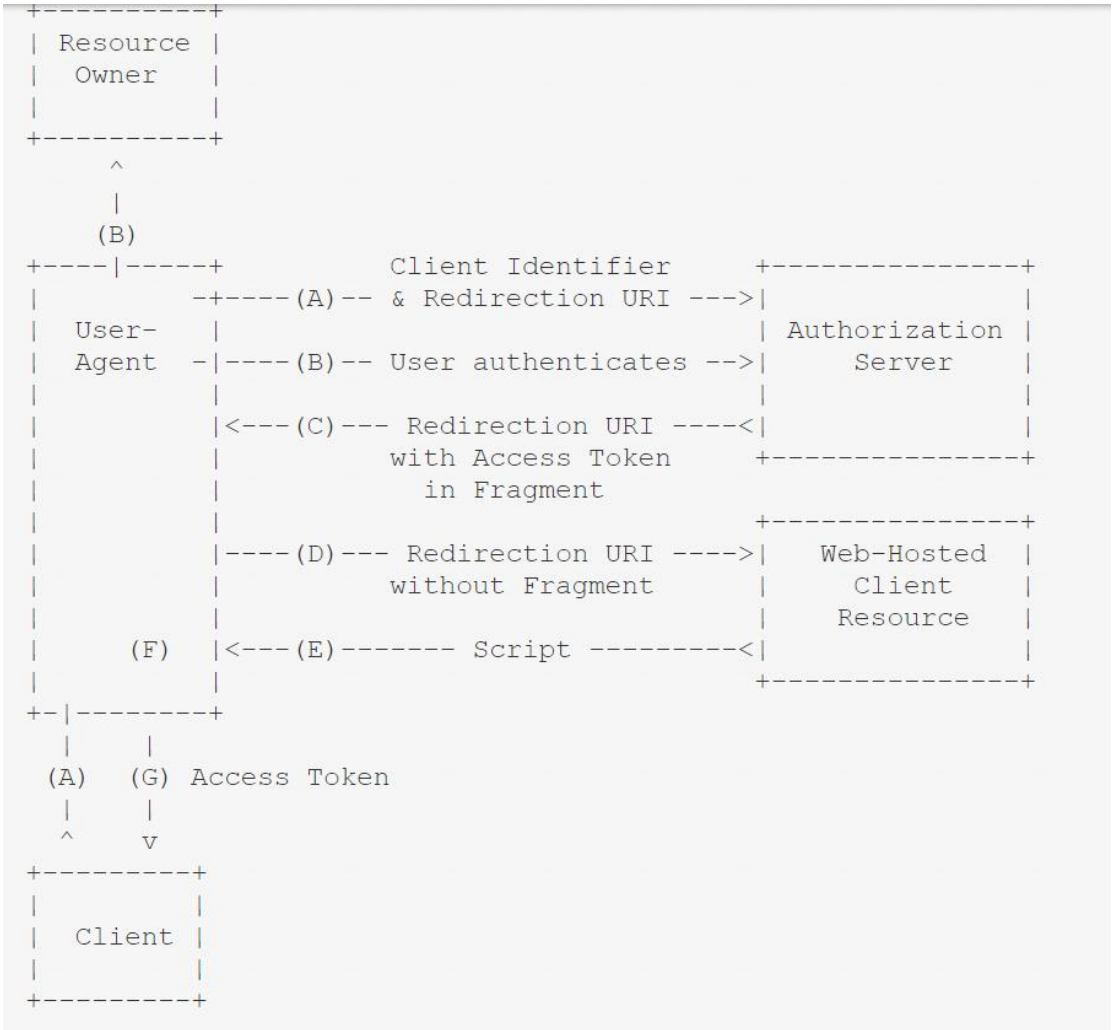
4.2 隐式模式

隐式许可模式用于获取 Access Token（不支持颁发 Refresh Token），并且，为公开通用的客户端优化实现特定场景下的重定向 URI。这些客户端通常是宿主在浏览器中，使用一种脚本语言，如 JavaScript。

由于这是一个基于重定向的数据流，客户端应用必须可以与资源所有者的用户代理进行交互(通常是一个 Web 浏览器)，并且可以接收来自于授权服务器的请求(通过重定向)。

与授权码模式不一样，在授权码模式中，客户端对于授权与 Access Token 需要构造不同的请求，而隐式模式，接收授权请求的结果作为 Access Token。

隐式模式不包含客户端的论证，它依赖于资源所有者的存在，重定向 URI 的注册。由于 Access Token 被编码进重定向 URI 中，它可能被暴露到资源所有者，其它应用也驻留在同一设备中。



注意：描述步骤(A)与(B)的行被分成了两部分，因为，他们通过了用户代理。

在图 4 中描述的数据流包含以下步骤：

(A) 客户端通过引导资源所有者的用户代理到授权端点到完成初始化。客户端包含了客户端标识符，请求范围，局部状态，和重定向 URI，一旦授权服务器授权通过或者拒绝，授权服务器会向客户端用户代理响应给客户端。

(B) 授权服务器通过用户代理对资源所有者进行认证，并且无论资源所有者授权通过或者拒绝，都会建立连接。

(C) 假定资源所有者授权通过，授权服务器引导用户代理重定向到之前提供的重定向 URI。重定义 URI 包含的片段中包含了 Access Token。

(D) 用户代理通过向 web 托管的客户端资源发出请求(不包含片段 RFC2616)，遵循重定向指令。用户代理在局部范围内获取片段信息。

(E) web 托管的客户端资源向用户代理返回一个 web 页面(通常是包含嵌入式脚本的 HTML 文档)，它可以访问完整的重定向 URI，该 URI 包含了由用户代理获

得的片段信息，也可以提取包含在片段中的 Access Token（和其它参数）。

(F) 用户代理执行由 web 托管客户端提供的脚本，该脚本可以提取 Access Token。

(G) 用户代理传递 Access Token 给客户端。

阅读 1.3.2 和 9 了解隐式模式的使用背景。阅读 10.3 和 10.16 了解重要的安全考虑，当使用隐式模式时。

4.2.1 授权请求

客户端通过向查询组件增加以下参数来构造一个请求的 URI，使用 URL 编码格式，附录 B：

响应类型(response_type)

必填。值固定为 token。

客户端 ID(client_id)

必填。2.2 小节中描述的客户端标识符。

重定向 URI(redirect_uri)

可选。3.1.2 小节中描述。

范围(scope)

可选。访问请求的范围在 3.3 小节中描述。

状态(state)

推荐。一个不透明值，被客户端用来维持在请求与回调之间的状态。授权服务器当重定向用户代理到客户端时，会带上这个参数。该参数应该用来阻止跨站请求伪造，如 10.12 描述。

客户端引导资源所有者到一个构造的 URI，通过一个 HTTP 重定向响应，或者通过其他可用方式到该 URI。

例如，客户端引导用户代理构造使用 TLS 的 HTTP 请求(多余的行分隔符仅用于显示)。

```
GET /authorize?response_type=token&client_id=s6BhdRkqt3&state=xyz
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

授权服务器会校验请求，以确保所有的请求参数出现并且有效。如果请求有效，授权服务器会认证资源所有者，并且获取一个授权决定结果(通过询问资源

所有者或者通过其他方法建立批准机制)。

当决定建立时，授权服务器会通过一个 HTTP 重定向响应来引导用户代理重定向到一个 URI，或者通过其它方法来实现重定向。

4.2.2 Access Token 响应

如果资源所有者给该访问请求授权，授权服务器会颁发一个 Access Token 并且通过向查询组件增加以下 url 编码的参数，将其传递给客户端，附录 B：

access_token

必填。由授权服务器颁发的访问令牌。

token_type

必填。在 7.1 中描述的，颁发的令牌的类型，该值大小写不敏感。

expires_in

推荐。Access Token 的生命周期，秒。例如，3600 表示该 Access Token 会在响应之后的 1 小时之后失效。如果忽略，授权服务器应该通过其它方法来提供过期时间，或者其它方法或者文档来注明这个默认值。

scope

如果与客户端请求的相同，则可选；否则，必填。在 3.3 小节中描述了，Access Token 的 scope。

state

必填。如果在客户端的授权请求中，出现了 state 参数。

例如，授权服务器通过发送以下 HTTP 响应，来重定向用户代理。

```
HTTP/1.1 302 Found
Location: http://example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA
          &state=xyz&token_type=example&expires_in=3600
```

开发者应该清楚，某些用户代理并不支持在 HTTP Location 响应头部中的片段组件中的包含。这些客户端会要求使用其它的方法来重定向客户端，而不是 3xx 重定向的响应，例如，返回一个 HTML 页面，包含了一个继续的按钮，链接到一个重定向的 URI。

客户端必须忽略不可识别的响应参数。该文档并未定义 Authorization code

字符串的长度。客户端应避免假定 Authorization code 字符串的长度。授权服务器应该在文档中注明它所颁发的授权码的大小。

4.2.2.1 错误响应

如果请求失败，是因为，缺少重定向 URI，无法的重定向 URI，或者不匹配的重定向 URI，或者如果客户端标识符缺失、无效，授权服务器应该通知资源所有者错误信息，并且，不允许自动重定向用户代理到这个无效的 URI。

如果资源所有者拒绝访问请求，或者请求因为某些非缺失、无效重定向 URI 原因而失败时，授权服器通过向查询组件增加以下 URL 编码格式的参数，来通知客户端，附录 B:

error

必填。单个的 ASCII 码错误码，来自于以下值：

invalid_request：请求缺少必要参数，包括无效的参数值，同一个参数出现多次，或者格式错误。

unauthorized_client：客户端没有授权使用该方法来请求一个授权码。

access_denied：资源所有者或者授权服务器拒绝该请求。

unsupported_response_type：授权服务器不支持通过这种方法来获取授权码。

invalid_scope：scope 是无效的，未知的，或者格式错误。

server_error：授权服务器遇到未知错误，从而不能处理请求。(该错误码是必要的，因为 500 服务器内部错误的 HTTP 响应状态码不能通过一个重定向来返回给客户端)。

temporarily_unavailable：授权服务器由于临时过载或者服务器维护，当前不能处理请求(该错误码是必要的，因为 503 服务不可达的 HTTP 响应状态码不能通过一个重定向来返回给客户端)。

error_description

可选。人可读的 ASCII 文本，提供了额外的信息，被用来客户端开发者理解错误。error_description 的值不能包含超出%x20-21 / %x23-5B / %x5D-7E 的字

符。

error_uri

可选。一个 URI，标识一个人可读的 web 页面，提供了额外错误信息，被用来客户端开发者理解错误。error_uri 的参数值必须遵守 URI 参考语法，不能包含超出%x20-21 / %x23-5B / %x5D-7E 的字符。

state

如果在客户端的授权请求参数中出现了 state 参数，则必填。该值与从客户端接收过来的值相同。

例如，授权服务器通过发送以下 HTTP 响应来重定向用户代理：

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=access_denied&state=xyz
```

4.3 密码模式

密码许可模式适用于资源所有者与客户端应用有着信任关系的情况，例如设备操作系统或者高权限的应用。当启用此模式，并且其它模式不支持时，授权服务器应该特别注意。

密码模式适用于这样的客户端，它们可以获得资源所有者的凭证(用户名与密码，通常使用可以交互的形式)。它也适用于通过直接认证模式例如基本 HTTP 认证或者摘要认证来迁移已经存在的客户端到 OAuth2.0 协议上来，伴随着将已经存储的凭证信息转换为 Access Token。

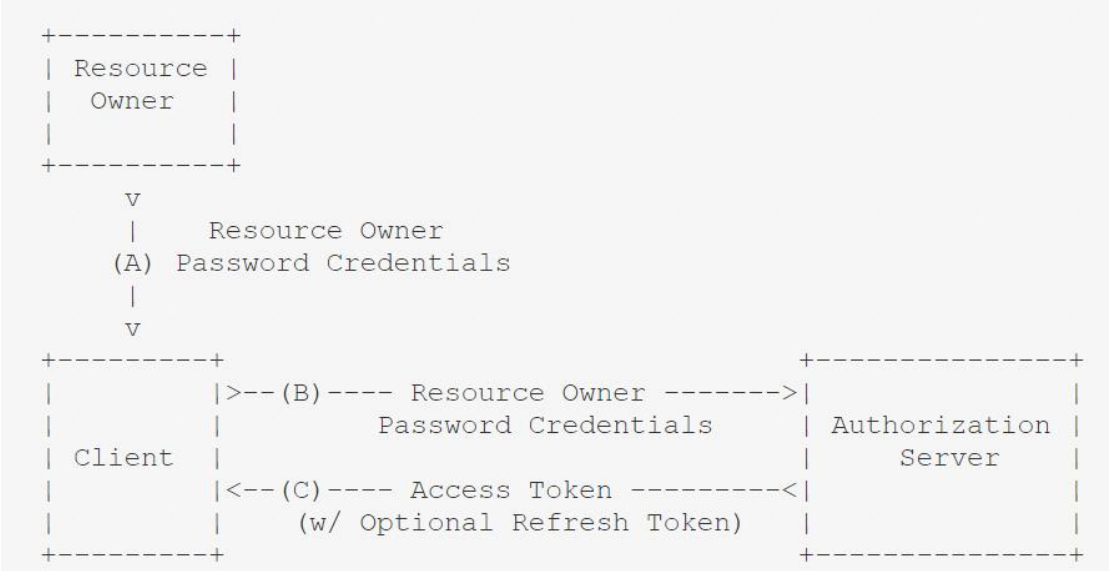


Figure 5: Resource Owner Password Credentials Flow

(A) 资源所有者向客户端提供用户名与密码。

(B) 客户端携带从资源所有者那里接收到的凭证信息，向授权服务器的 Token 端点那里请求一个 Access Token。当构造请求时，客户端向授权服务器进行认证。

(C) 授权服务器认证客户端，并且验证资源所有者的凭证信息，如果有效，颁发 Access Token。

4.3.1 授权请求和响应

客户端获取资源所有者凭证信息的方法超出了本文档的范围。客户端一时获得了 Access Token，必须丢弃凭证。

4.3.2 Access Token 请求

客户端向 Token 端点发送一个请求，通过发送以 URL 编码的参数，附录 B，在 HTTP 请求体中，以 UTF-8 编码。

grant_type

必填。固定为 password。

username

必填。资源所有者用户名。

password

必填。资源所有者密码。

scope

可选。访问请求的范围，在 3.3 中有描述。

如果客户端类型是保密类型的，或者客户端被颁发客户端凭证(或者赋予认证要求)，则客户端必须向授权服务器认证，如 3.2.1 描述。

例如，客户端使用 TLS 来构造请求(多余行仅用于显示目的)：

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=johndoe&password=A3ddj3w
```

授权服务器必须做到：

- 对于保密的客户端，或被颁发客户端凭证的客户端(或者其它认证要求)，要求客户端进行认证。
- 如果客户端认证被要求，则认证客户端
- 使用存在的密码校验算法来校验资源所有者的密码凭证。

由于这样的 Access Token 请求利用了资源所有者的密码，授权服务器必须保护端点免于蛮力攻击（例如，限流或者生成警报）。

4.3.3 Access Token 响应

如果 Access Token 请求是有效的并且被授权的，授权服务器会颁发一个 Access Token，和一个可选的 Refresh Token，如 5.1 中描述。如果请求没有通过客户端认证，或者请求无效，授权服务器会返回一个错误响应，如 5.2 中描述。

一个成功的响应如下所示：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

4.4 客户端模式

当客户端请求在自己控制下的受保护资源或者由其它资源所有者控制的资源时，客户端可以只使用客户端凭证(或者其它认证方法)来请求一个 Access Token。客户端模式必须只能适用于保密的客户端。

Figure 6: Client Credentials Flow

(A) 客户端向授权服务器进行论证，并从 token 端点那里请求一个 Access Token。

(B) 授权服务器对客户端进行认证，如果有效，则颁发 Access Token。

4.4.1 授权请求和响应

由于客户端认证被用作授权许可，所以不再需要授权请求。

4.4.2 Access Token 请求

客户端向 Token 端点发送一个请求，通过发送以 URL 编码的参数，附录 B，在 HTTP 请求体中，以 UTF-8 编码：

grant_type

必填。固定为 client_credentials。

scope

可选。访问请求的范围，在 3.3 中有描述。

客户端必须向授权服务器进行认证，在 3.2.1 中描述。

例如，客户端使用 TLS 来构造请求(多余行仅用于显示目的)：

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

4.4.3 Access Token 响应

如果 Access Token 请求是有效的并且被授权的，授权服务器会颁发一个 Access Token，和一个可选的 Refresh Token，如 5.1 中描述。如果请求没有通过客户端认证，或者请求无效，授权服务器会返回一个错误响应，如 5.2 中描述。

一个成功的响应如下所示：

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "example_parameter":"example_value"
}
```

4.5 扩展模式

客户端通过使用绝对 URI（由授权服务器来定义）指定许可类型作为 token

端点的 grant_type 参数值，并且需要的话，增加一些其他参数，来实现扩展许可类型。例如，使用 SAML2.0 断言许可类型(OAuth-SAML2 中定义)来请求一个 Access Token，客户端可以使用 TLS 来构造以下的 HTTP 请求：

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

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-
bearer&assertion=PEFzc2VydGlvbiBJc3N1ZUluc3RhbnQ9IjIwMTEtMDU
[...omitted for brevity...]aG5TdGF0ZW1lbnQ-PC9Bc3NlcnRpb24-
```

如果 Access Token 请求是有效的并且被授权的，授权服务器会颁发一个 Access Token，和一个可选的 Refresh Token，如 5.1 中描述。如果请求没有通过客户端认证，或者请求无效，授权服务器会返回一个错误响应，如 5.2 中描述。

第五章 颁发 Access Token

如果 Access Token 请求是有效的并且被授权的，授权服务器会颁发一个 Access Token，和一个可选的 Refresh Token，如 5.1 中描述。如果请求没有通过客户端认证，或者请求无效，授权服务器会返回一个错误响应，如 5.2 中描述。

5.1 成功响应

授权服务器颁发一个 Access Token 和一个可选的 Refresh Token，通过以 200 响应状态码的响应体中追加以下参数来构造一个响应：

access_token

必填。由授权服务器颁发的访问令牌。

token_type

必填。在 7.1 中描述的，颁发的令牌的类型，该值大小写不敏感。

expires_in

推荐。Access Token 的生命周期，秒。例如，3600 表示该 Access Token 会在响应之后的 1 小时之后失效。如果忽略，授权服务器应该通过其它方法来提供过期时间，或者其它方法或者文档来注明这个默认值。

refresh_token

可选的。Access Token 可以用来获取新的 Access Token，如第 6 章中描述。

scope

如果与客户端请求的相同，则可选；否则，必填。在 3.3 小节中描述了。

在 HTTP 响应实体中，以"application/json"数据类型，在 RFC4627 中定义，包含这些参数。这些参数被序列化为一个 JSON 数据结构的字符串，位于顶层。参数名与参数值包含在 JSON 字符串。数字类型的值也被包含作为 JOSN 数字。参数的顺序没有关系，可以变化。

授权服务器必须包含”Cache-Control”响应头域(RFC2616)，值为”no-store”，在任何包含 tokens，凭证，或者其它敏感信息的响应中，“pragma”响应问域也是一样，值为”no-cache”。

例如:

```
For example:

   HTTP/1.1 200 OK
   Content-Type: application/json;charset=UTF-8
   Cache-Control: no-store
   Pragma: no-cache

   {
     "access_token":"2YotnFZFEjr1zCsicMWpAA",
     "token_type":"example",
     "expires_in":3600,
     "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
     "example_parameter":"example_value"
   }
```

客户端必须忽略不可识别的响应参数。该文档并未定义 Authorization code 字符串的长度。客户端应避免假定 Authorization code 字符串的长度。授权服务器应该在文档中注明它所颁发的授权码的大小。

5.2 错误响应

授权服务器返回一个 400 状态码(错误的请求)，并且包含以下参数:

error

必填。单个的 ASCII 码错误码，来自于以下值：

invalid_request：请求缺少必要参数，包括无效的参数值，同一个参数出现多次，或者格式错误。

unauthorized_client：客户端没有授权使用该方法来请求一个授权码。

access_denied：资源所有者或者授权服务器拒绝该请求。

unsupported_response_type：授权服务器不支持通过这种方法来获取授权码。

invalid_scope：scope 是无效的，未知的，或者格式错误。

server_error：授权服务器遇到未知错误，从而不能处理请求。(该错误码是必要的，因为 500 服务器内部错误的 HTTP 响应状态码不能通过一个重定向来返回给客户端)。

temporarily_unavailable：授权服务器由于临时过载或者服务器维护，当前不能处理请求(该错误码是必要的，因为 503 服务不可达的 HTTP 响应状态码不能通过

一个重定向来返回给客户端)。

error_description

可选。人可读的 ASCII 文本，提供了额外的信息，被用来客户端开发者理解错误。error_description 的值不能包含超出%x20-21 / %x23-5B / %x5D-7E 的字符。

error_uri

可选。一个 URI，标识一个人可读的 web 页面，提供了额外错误信息，被用来客户端开发者理解错误。error_uri 的参数值必须遵守 URI 参考语法，不能包含超出%x20-21 / %x23-5B / %x5D-7E 的字符。

如果与客户端请求的相同，则可选；否则，必填。在 3.3 小节中描述了。在 HTTP 响应实体中，以"application/json"数据类型，在 RFC4627 中定义，包含这些参数。这些参数被序列化为一个 JSON 数据结构的字符串，位于顶层。参数名与参数值包含在 JSON 字符串。数字类型的值也被包含作为 JOSN 数字。参数的顺序没有关系，可以变化。

例如：

```
For example:

  HTTP/1.1 400 Bad Request
  Content-Type: application/json;charset=UTF-8
  Cache-Control: no-store
  Pragma: no-cache

  {
    "error":"invalid_request"
  }
```

第六章 刷新 Access Token

如果授权服务器向客户端颁发了一个 Refresh Token，客户端可以通过在 url 编码的参数中(附录 B)以 UTF-8 编码，在 HTTP 请求体中追加以下参数来构造一个刷新 Access Token 的请求：

grant_type

必填。固定为 refresh_token。

refresh_token 必填。颁发给客户端的 refresh_token。

scope

可选。访问请求的范围，在 3.3 中有描述。

因为 Refresh Token 通常是持久的凭证，被用来请求新的 Access Token，该 Refresh Token 被绑定到被颁发的客户端。如果客户端类型是保密的，或者客户端被颁发了客户端凭证，客户端必须向授权服务器进行认证，在 3.2.1 中描述。

例如，客户端使用 TLS 来构造请求(多余行仅用于显示目的)：

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
```

授权服务器必须做到：

- 对于保密的客户端，或被颁发客户端凭证的客户端(或者其它认证要求)，要求客户端进行认证。
- 如果客户端被要求认证，并确保 Refresh Token 是颁发给被论证的客户端，则要求对客户端进行认证。
- 校验 Refresh Token。

如果客户端有效并授权通过，授权服务器会颁发一个如 5.1 中描述的 Access Token。如果请求没有通过认证或者请求无效，授权服务器会返回一个如 5.2 中描述的错误响应。

授权服务器可能会颁发一个新的 Refresh Token，在这种场景下，客户端必须丢弃旧的 Refresh Token 并用新的来替换旧的 Refresh Token。授权服务器在颁发

新的 Refresh Token 给客户端之后，可能撤销旧的 Refresh Token。如果颁发一个新的 Refresh Token，那么该 Refresh Token 的范围与被包含在客户端请求中的 Refresh Token 范围相同。

第七章 访问受保护资源

客户端通过向资源服务器展示 Access Token 来访问受保护资源。资源服务器必须校验 Access Token，并确保它没有过期，并且它的范围覆盖了请求的资源。资源服务器校验 Access Token 的方法(有一些错误响应)，超出了本文档所描述的范围，但一般来讲，涉及到资源服务器与授权服务器之间的交互与协调。

客户端利用 Access Token 向资源服务器去认证的方法，依赖于授权服务器所颁发的 Access Token 的类型。通常情况下，它涉及到使用 HTTP 请求头中的”Authorization”，伴随着认证模式。

7.1 Access Token 类型

Access Token 类型给客户端提供了一些信息，这些用来利用构造一个受保护资源的请求(伴随着指定类型的属性)。客户端如果不理解 Access Token 类型，就不准使用 Access Token。

例如，通过在请求中包含 Access Token 来使用 Access Token 类型，在 RFC6750 中定义。

```
GET /resource/1 HTTP/1.1
Host: example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

然而，在 OAuth-HTTP-MAC 中定义的 MAC 令牌类型通过颁发一个消息认证码与 Access Token 一起来使用，该 MAC 被用于对 HTTP 请求中的一些组件进行签名。

上述举例仅用于描述目的。建议开发者在使用之前参考 RFC6750 和 OAuth-HTTP-MAC 规范。

每一种 Access Token 类型指定了额外的属性，这些属性伴随着 access_token 响应一起被发送给客户端。在请求一个受保护资源时，它也可以定义一种用于包含 Access Token 的 HTTP 认证方法。

7.2 错误响应

如果一个资源请求失败，资源服务器应该通知客户端错误信息。尽管这些错误响应的规范超出了本文的范围，但是本文在 11.4 节中创建一个公共的注册中心，用于在 OAuth2.0 令牌认证模式中共享这些错误信息。

主要为 OAuth2.0 令牌认证的新的认证模式应该建立一种机制用来向客户端提供错误状态码，这种机制允许错误信息在注册中心进行注册。

这种模式可能将有效的错误码集限制到一被注册错误码的子集。如果用命名参数来命名错误码，参数名应该是 error。

其它可以用作 OAuth 令牌的认证模式，但不是主要用于这个目的的模式，可能以相同的方式将这些错误码绑定到注册中心。

在本文中，新的认证方法可能选择指定 error_description 和 error_uri 的使用，用来一起返回错误信息。

第八章 扩展

8.1 自定义 Access Token 类型

有两种方法来定义 Access Token 类型：在 Access Token 类型注册中心进行注册(遵行 11.1 中描述的步骤)，或者使用唯一的绝对的 URI 作为其名称。

利用 URI 名称的 Access Token 类型应该被限制在特定供应商的实现，并不通用，并且具体到了他们所使用的资源服务器的细节。

所有其它类型必须被注册。类型名称必须遵守类型名称 ABNF。如果类型定义包含了一种新的 HTTP 认证名称，该类型名称应该与 HTTP 认证模式相同(RFC2417 定义)。”example”类型被保留作为案例中使用。

```
type-name  = 1*name-char
name-char  = "-" / "." / "_" / DIGIT / ALPHA
```

8.2 自定义新的端点参数

在授权端点或者 token 端点中使用的新的请求、响应参数是定义和注册在 OAuth 参数注册中心的，遵循 11.2 中描述的过程。

参数名称必须遵守参数名 ABNF，并且参数值语法必须是定义好的(例如 ABNF，或者一个已经存在的参数的语法引用)。

```
param-name  = 1*name-char
name-char   = "-" / "." / "_" / DIGIT / ALPHA
```

未注册的特定于供应商的参数扩展，没有通用性，且局限于所用授权服务器的实现细节，这些参数扩展应该利用特定供应商的前缀，不太可能与其它已经注册的参数值冲突(例如以公司名称打头)。

8.3 自定义新的授权许可类型

可以通过给 grant_type 赋予一个唯一的绝对的 URI 参数，来定义一种新的授权许可类型。如果这个扩展的许可类型要求额外的 token 端点参数，他们必须在 OAuth 参数注册中心进行注册，如 11.2 中描述。

8.4 自定义新的授权许可端点响应类型

在授权端点中使用的新的响应类型在授权端点响应类型注册中心进行定义与注册，遵循 11.3 中描述的过程。响应类型必须遵守其 ABNF。

```
response-type  = response-name *( SP response-name )
response-name  = 1*response-char
response-char  = "_" / DIGIT / ALPHA
```

如果一个响应类型包含一个或者多个空格字符，它被作为空格分隔的值列表，顺序没有影响。只有一种顺序可以被注册，覆盖了同一个值集合的多个排列组合。

例如，响应类型”token code”这个文档没有定义。然而，一个扩展可以定义与注册这个”token code”类型。一旦完成注册，相同的组件就不能再注册为”code token”，但是这些值可以用来表示相同的响应类型。

8.5 自定义新的错误码

在这样的场景中，协议扩展(Access Token 类型，扩展参数，扩展许可类型)要求附加的错误码来与授权码许可错误响应一起使用，隐含许可错误响应，token 错误响应，这些错误码可能要定义。

如果扩展错误码与一个已注册的 Access Token 类型，已注册的的端点参数，扩展的许可类型结合使用，就必须被注册(遵循 11.4 的过程)。

错误码必须遵守其 ABNF，并且如果可能应该使用一个标识名称作为前缀。例如，一个对参数”exmple”标识无效值集合的错误码应该命名无”

example_invalid”。

```
error      = 1*error-char
error-char = %x20-21 / %x23-5B / %x5D-7E
```

第九章 原生应用

原生应用是安装与执行在资源所有者设备上的客户端程序（例如，桌面应用，原生手机应用）。原生应用要求一些特定的考虑，关于安全，平台能力，和整个终端体验。

授权端点要求客户端与资源所有者用户代理之间的交互。原生应用可以撤销外部用户代理或者嵌入在应用中的用户代理。例如：

- 外部用户代理-原生应用可以捕获来自于授权服务器的响应，原生应用使用注册在操作系统的重定向 URI 来撤销客户端，原生应用通过手动的复制与粘贴来实现该撤销，运行在局部的安装了一个用户代理扩展的 web 服务器上，或者通过提供一个标识了托管服务器的重定向 URI，反过来，可以响应原生应用。
- 嵌入式的用户代理-原生应用通过直接与嵌入式的用户代理进行通信，监控在资源加载，访问用户代理 cookies 存储时的状态变更来获取响应。

当在外部用户代理与嵌入式用户代理之间进行选择时，开发者必须考虑以下因素：

- 由于资源所有者已经与授权服务器有一个活动的会话，不需要再次认证，外部用户代理可能提高完成率。它提供了一处非常熟悉的终端用户的体验与功能。资源所有者可能依赖于用户代理的特征或者扩展来辅助认证(例如，密码管理器，2 因子设备管理器)。
- 嵌入式的用户代理可能可用性更好，因为它消除了切换上下文与打开新窗口的情况 。
- 因为资源所有者在一个不需要访问可见的保护的身份不明的窗口进行认证，嵌入式的用户代理构成了一个安全上的挑战。嵌入式的用户代理培养终端用户信息身份不明的认证请求(使得钓鱼攻击更简单执行)。

当在隐含授权模式与授权码模式中选择时，应该考虑以下因素：

- 使用授权码模式的原生应用应该这样做，不使用客户端凭证，因为原生应用没有能力维护凭证的保密性。

- 当使用隐含模式时，不再返回 refresh token，一时 access token 过期，就会要求重复授权过程。

第十章 安全考虑因素

作为一个灵活的和可扩展的框架，OAuth 的安全考虑依赖于许多因素。下面给实现者提供了一些安全指导原则，在 2.1 中三种客户端的场景：web 应用、基于用户代理的应用、原生应用。

OAuth-THREATMODEL 提供了一个综合的安全模型、分析，和协议设计背景。

10.1 客户端认证(未开始)

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.2 客户端假冒

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.3 Access Token

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.4 Refresh Token

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.5 Authorization Code

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.6 Authorization Code Redirection URI 操作

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.7 资源所有者密码凭证

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.8 请求保密性

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.9 确保端点真实性

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.10 凭证猜测攻击

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.11 钓鱼攻击

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.12 跨站请求伪造

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.13 点击劫持

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.14 代码侵入与输入校验

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.15 开放的重定向器

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

10.16 在隐含模式中滥用 Access Token 模仿资源所有者

授权端点是用来与资源所有者进行交互，并且用来获取一个授权许可。授权服务器首先必须验证资源所有者的身份。授权服务器认证资源所有者的的方式(用户名与密码登录，会话 cookies)超出了本文档描述的范围。

第十二章 IANA 考虑因素(未开始)

11.1 OAuth Access Token 类型注册中心

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。

11.1.1 注册模板

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。

11.2 OAuth 参数注册中心

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。

11.2.1 注册模板

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。

11.2.2 初始的注册中心内容

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。

11.3 OAuth 授权端点响应类型注册中心

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。

11.3.1 注册模板

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。

11.3.2 初始的注册中心内容

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。

11.4 OAuth 扩展错误码注册中心

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。

11.4.1 注册模板

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。

第十三章 参考文档(未开始)

12.1 规范性参考文档

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。

12.2 信息参考文档

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。