安全问题：凭据共享与凭据盗用

在传统的客户端–服务器身份认证模型中，客户端通过使用资源所有者的凭据与服务器进行身份认证来请求服务器上的访问受限资源(受保护的资源)。
为了向第三方应用程序提供对受限资源的访问权限，资源所有者与第三方共享其凭据。这会产生几个问题和限制：
1.第三方应用程序需要存储资源所有者的凭据以供将来使用，通常是明文密码。
2.服务器必须支持密码验证，尽管密码本身存在安全漏洞。
3.第三方应用程序获得了对资源所有者受保护资源的过于广泛的访问权限，使资源所有者无法限制持续时间或访问有限的资源子集。
4.资源所有者不能在不撤销所有第三方的访问权限的情况下撤销对单个第三方的访问权限，并且必须通过更改第三方的密码来实现。
5.任何第三方应用程序的泄露都会导致最终用户的密码以及受该密码保护的所有数据的泄露。

解决问题的关键思想：角色分离、授权访问、访问控制

OAuth 通过引入授权层并将客户端的角色与资源所有者的角色分离来解决这些问题。

在 OAuth 中，客户端请求访问由资源所有者控制并由资源服务器托管的资源，并获得与资源所有者不同的一组凭据。

客户端不是使用资源所有者的凭证来访问受保护的资源，而是获得一个访问令牌——一个表示特定范围、生命周期和其他访问属性的字符串。

访问令牌由授权服务器在资源所有者的批准下颁发给第三方客户端。客户端使用访问令牌访问资源服务器托管的受保护资源。

例如，最终用户(资源所有者) 可以授权打印服务(客户端) 访问她存储在照片共享服务(资源服务器) 中的受保护照片，而无需与打印服务共享她的用户名和密码。相反，她直接向照片共享服务(授权服务器) 信任的服务器进行身份认证，该服务器颁发打印服务委托特定的凭据(访问令牌)。

OAuth 2.0 设计的重要假设：不受控的客户端总比授权服务器或者受保护资源多出好几个数量级

方案设计&协议

OAuth 2.0 的优缺点

优点

更安全，客户端不接触用户密码，服务端更易集中保护
广泛传播并被持续采用
模块化和可扩展性好，支持多种客户端架构场景
短寿命和封装的令牌
资源服务器和授权服务器解耦
集中式授权，简化客户端
HTTP/JSON友好，易于请求和传递令牌
客户可以有不同的信任级别

缺点

OAuth 2.0只是一个协议，不是具体实现
不同产品/框架有不同的实现，会出现兼容性问题

OAuth 2.0 不能做什么?

OAuth 是一个授权协议，不是身份认证协议

OAuth 没有定义 HTTP 协议之外的情形

OAuth 没有定义用户对用户的授权机制

OAuth 没有定义授权处理机制

OAuth 没有定义令牌格式

OAuth 没有定义加密算法

OAuth 不是单体协议：该规范被分成了多个定义和流程，每个定义和流程都有各种适用的场景

OAuth 2.0 的优缺点

OAuth 2.0 不能做什么?

OAuth 2.0 (RFC6749)：让最终用户通过OAuth将他们在受保护资源上的部分权限委托给客户端应用，使客户端应用代表他们执行操作。
https://datatracker.ietf.org/doc/html/rfc6749

OAuth 2.0 角色定义

资源所有者 (Resource Owner)
能够授予对受保护资源的访问权限的实体。当资源所有者是一个人时，它被称为最终用户

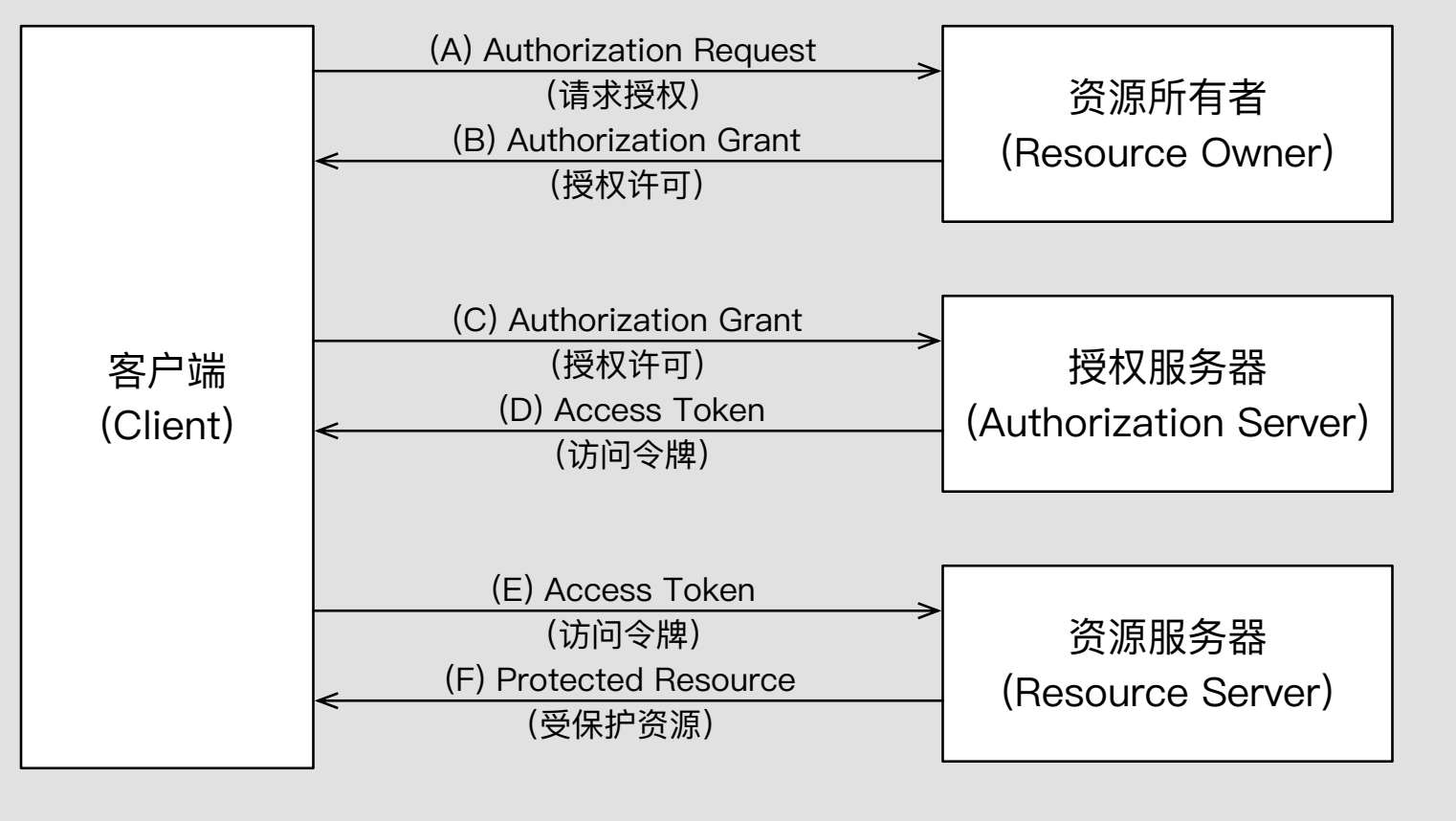
授权服务器 (Authorization Server)
服务器在成功验证资源所有者并获得授权后向客户端颁发访问令牌

客户端 (Client)
代表资源所有者并经其授权发出受保护资源请求的应用程序。术语「客户端」并不意味着任何特定的实现特征(例如，应用程序是否在服务器、桌面或其他设备上执行)

资源服务器 (Resource Server)
托管受保护资源的服务器，能够使用访问令牌接受和响应受保护资源请求

核心流程&关系

OAuth 2.0 Protocol Flow



OAuth 2.0 Protocol Flow 描述了四个角色之间的交互，包括以下步骤：
(A) 客户端向资源所有者请求授权。授权请求可以直接发送给资源所有者(如图所示)，或者最好通过授权服务器作为中介间接发送。
(B) 客户端接收到授权许可，它是代表资源所有者授权的凭证，使用RFC6749中定义的四种授权类型之一或使用扩展授权类型表示。授权类型取决于客户端请求授权的方式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身份认证并提供授权许可来请求访问令牌。
(D) 授权服务器对客户端进行**身份认证(authenticate)**并**验证授权许可(validate the authorization grant)**，如果有效，则颁发访问令牌。
(E) 客户端向资源服务器请求受保护的资源，并通过提供访问令牌进行身份认证。
(F) 资源服务器验证访问令牌，如果有效，则为请求提供服务。

客户端从资源所有者获得授权许可的首选方法（如步骤(A)和(B)所示）是使用授权服务器作为中介

关键概念

OAuth 2.0 关键概念

访问令牌 (Access Token)
访问令牌是用于访问受保护资源的凭据。访问令牌是一个字符串，表示颁发给客户端的授权。该字符串通常对客户端是不透明的。令牌代表特定的访问范围和持续时间，由资源所有者授予，并由资源服务器和授权服务器强制执行。

令牌可以表示用于检索授权信息的标识符，也可以以可验证的方式自包含授权信息(即，由一些数据和签名组成的令牌字符串)。为了让客户端使用令牌，可能需要其他**身份认证凭据(authenthication credentials)**。

访问令牌提供了一个抽象层，用资源服务器可以理解的单个令牌替换不同的授权结构(例如，用户名和密码)。这种抽象使得发布访问令牌比用于获取它们的授权许可更具限制性，并且消除了资源服务器对了解各种身份认证(authenthication)方法的需要。

根据资源服务器的安全要求，访问令牌可以具有不同的格式、结构和使用方法(例如，加密属性)。

刷新令牌 (Refresh Token)
刷新令牌是用于获取访问令牌的凭据。

刷新令牌由授权服务器下发给客户端，用于在当前访问令牌失效或过期时获取新的访问令牌，或者获取范围相同或更窄的额外访问令牌（访问令牌的生命周期可能较短，并且权限少于资源所有者授权的权限）。 授权服务器可自行决定是否发布刷新令牌。如果授权服务器发出刷新令牌，则在发出访问令牌时会包含它 (即下面 OAuth 2.0 Refresh Token Flow 中的步骤(D)) 。

刷新令牌是一个字符串，表示资源所有者授予客户端的授权。该字符串通常对客户端是不透明的。令牌表示用于检索授权信息的标识符。 **与访问令牌不同，刷新令牌仅用于授权服务器，并且永远不会发送到资源服务器。**

Protocol Endpoints
授权过程使用两个**授权服务器端点**（HTTP 资源）：

授权端点（Authorization Endpoint)
客户端用于通过用户代理重定向从资源所有者获得授权。

令牌端点 (Token Endpoint)
客户端用于交换访问令牌的授权授权，通常使用客户端身份验证。

以及一个**客户端端点**：

重定向端点 (Redirection Endpoint)
授权服务器用于通过资源所有者用户代理向客户端返回包含授权凭据的响应。

并非每种授权授予类型都使用这两个端点。
扩展授权类型可以根据需要定义额外的端点。

客户端注册

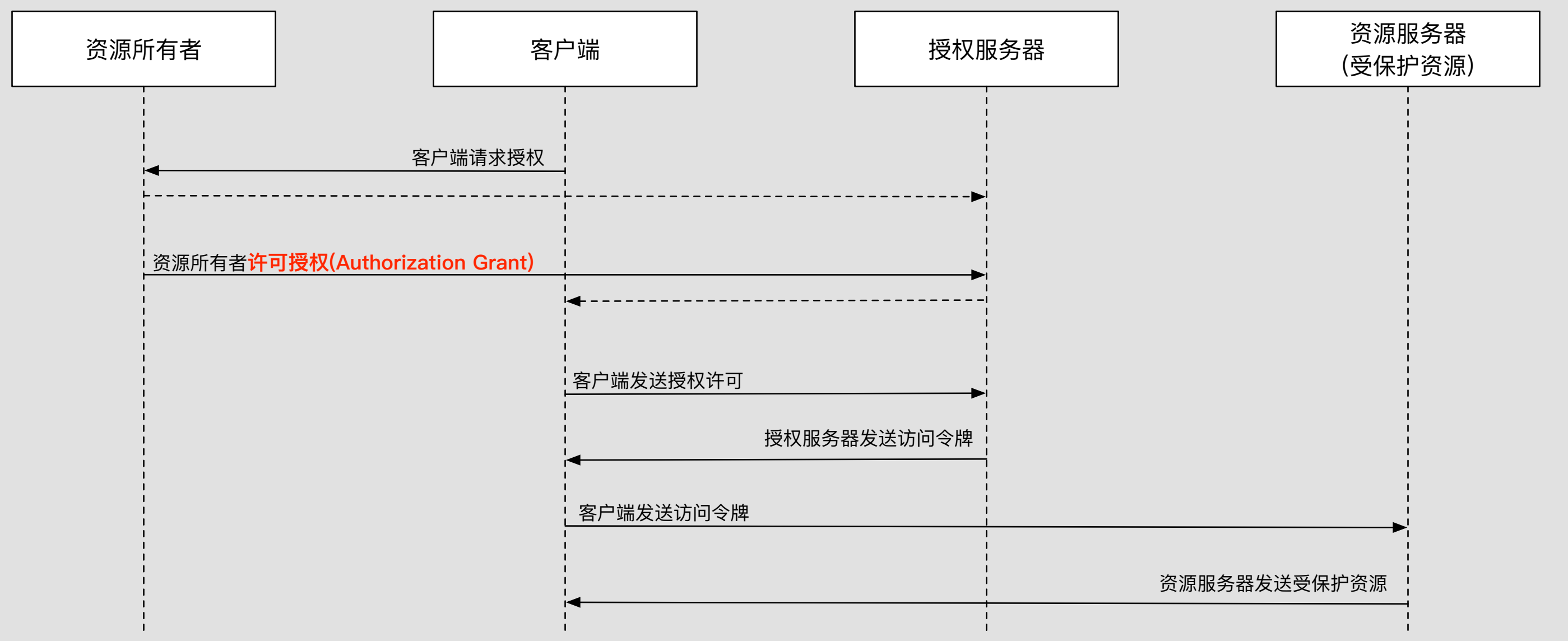
基本授权&访问

OAuth 2.0 Client Registration(客户端注册)

客户端注册不需要客户端和授权服务器之间的直接交互。当授权服务器支持时，注册可以依赖其他方式来建立信任并获得所需的客户端属性（例如，重定向 URI、客户端类型）。例如，注册可以使用自发布或第三方发布的断言来完成，或者通过授权服务器使用可信通道执行客户端发现来完成。

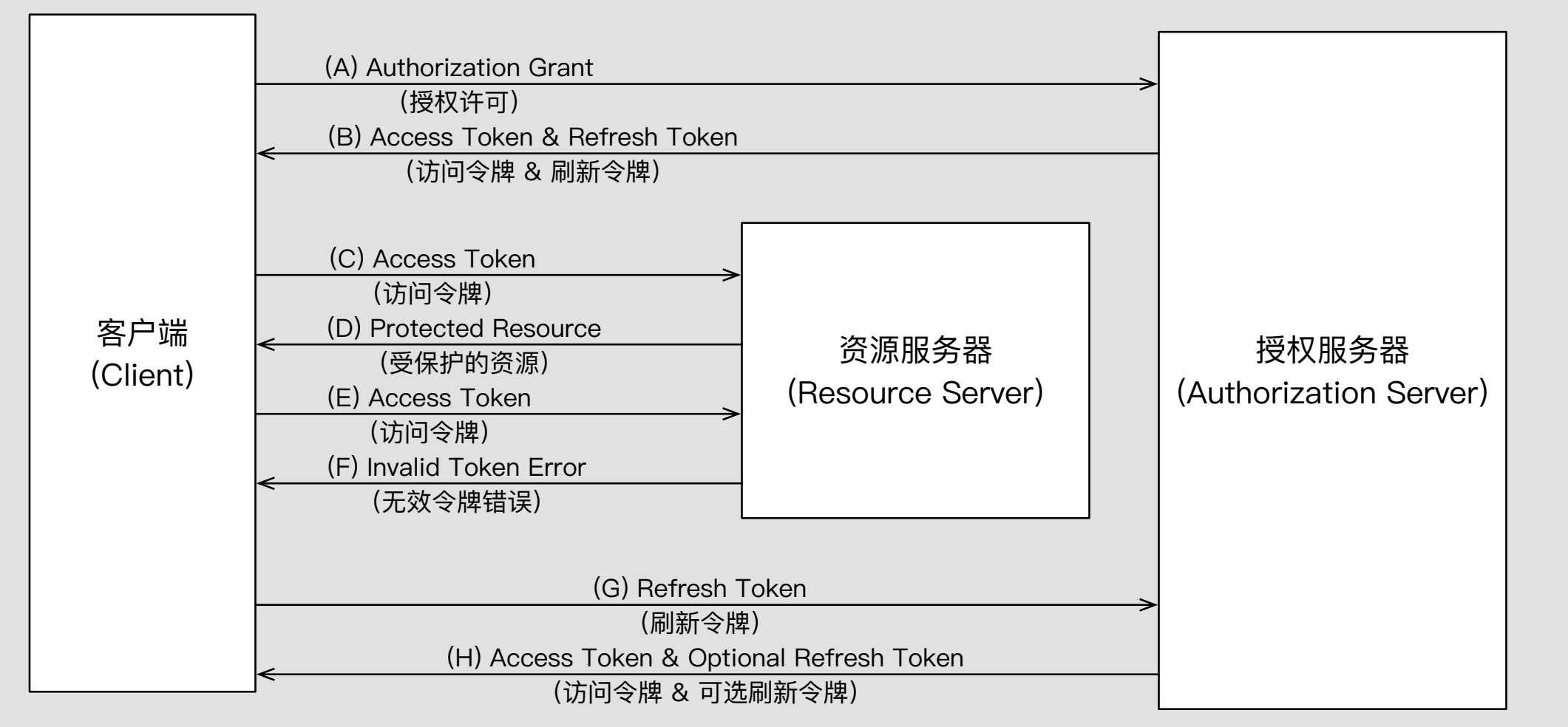
注册客户端时，客户端开发人员应：
1. 如指定客户端类型
2. 提供其客户端重定向 URI，以及
包括授权服务器要求的任何其他信息（例如，应用程序名称、网站、描述、徽标图像、接受法律条款）。

OAuth 2.0 工作流程 (许可授权&资源访问)



刷新令牌

OAuth 2.0 Refresh Token (刷新令牌) Flow



OAuth 2.0 Refresh Token (刷新令牌)，包括以下步骤：
((A) 客户端通过与授权服务器进行**身份认证(authenticate)**并提供授权许可来请求访问令牌。
(B) 授权服务器对客户端进行**身份认证(authenticate)**并**验证授权许可(validate the authorization grant)**，如果有效，则颁发访问令牌和刷新令牌。
(C) 客户端通过提供访问令牌向资源服务器发出受保护的资源请求。
(D) 资源服务器验证访问令牌，如果有效，则为请求提供服务。
(E) **重复步骤 (C) 和 (D) 直到访问令牌过期**。如果客户端知道访问令牌过期，则跳到步骤(G)；否则，它会发出另一个受保护的资源请求。
(F) 由于访问令牌无效，资源服务器返回无效令牌错误。
(G) 客户端通过与授权服务器进行**身份认证(authenticate)**并提供刷新令牌来请求新的访问令牌。客户端**身份认证(authenticate)**要求基于客户端类型和授权服务器策略。
(H) 授权服务器对客户端进行**身份认证(authenticate)**并验证刷新令牌，如果有效，则发出新的访问令牌(可选，新的刷新令牌)。

(C), (D), (E), and (F) 不在 OAuth 2.0 的范畴

客户端关键概念

许可授权(Authorization Grant)

OAuth 2.0 Client 关键概念

客户端类型 (Client Type)

OAuth 定义了两种客户端类型，基于它们与授权服务器进行安全身份验证的能力(即，保持其客户端凭据机密性的能力)

客户端类型指定基于授权服务器对安全身份验证的定义及其可接受的客户端凭据暴露级别。授权服务器不应该对客户端类型做出假设。

Confidential
客户端**能够维护其凭据的机密性**（例如，客户端在安全服务器上实现，对客户端凭据的访问受限），或能够使用其他方式保护客户端身份验证。

Public
无法维护其凭据机密性的客户端（例如，在资源所有者使用的设备上执行的客户端，例如已安装的本机应用程序或基于 Web 浏览器的应用程序），并且无法通过任何其他方式进行安全客户端身份验证。

客户端类型指定基于授权服务器对安全身份认证的定义及其可接受的客户端凭据暴露级别。授权服务器不应该对客户端类型做出假设。

客户端可以实现为一组分布式组件，每个组件具有不同的客户端类型和安全上下文（例如，具有基于机密服务器的组件和基于公共浏览器的组件的分布式客户端）。如果授权服务器不为此类客户端提供支持或不提供有关其注册的指导，则客户端应将每个组件注册为单独的客户端。

客户端标识(Client Identifier)

授权服务器向注册的客户端发出一个客户端标识符——一个代表客户端提供的注册信息的唯一字符串。客户端标识符不是秘密；它向资源所有者公开，不得单独用于客户端身份认证。客户端标识符对于授权服务器是唯一的。

rfc6749 未定义客户端标识符字符串大小。客户应避免对标识符大小做出假设。授权服务器应该记录它发布的任何标识符的大小。

客户端认证(Client Authentication)

如果客户端类型是 Confidential，客户端和授权服务器建立适合授权服务器安全要求的客户端认证方法。授权服务器可以接受满足其安全要求的任何形式的客户端认证。

Confidential 客户端通常会发布(或建立)一组客户端凭据，用于与授权服务器进行身份认证(例如，密码、公钥/私钥对)。

授权服务器可以与公共客户端建立客户端认证方法。但是，授权服务器不得依赖公共客户端身份认证来识别客户端。

客户端不得在每个请求中使用多个身份认证方法。

客户端密码(Client Password)

拥有客户端密码的客户端可以使用 [RFC2617] 中定义的 HTTP 基本身份认证方案(HTTP Basic authentication scheme)与授权服务器进行身份认证。客户端标识符使用“application/x-www-form-urlencoded”编码算法进行编码，编码值用作用户名；客户端密码使用相同的算法进行编码并用作密码。授权服务器必须支持 HTTP 基本身份认证方案，以对已获得客户端密码的客户端进行身份认证。

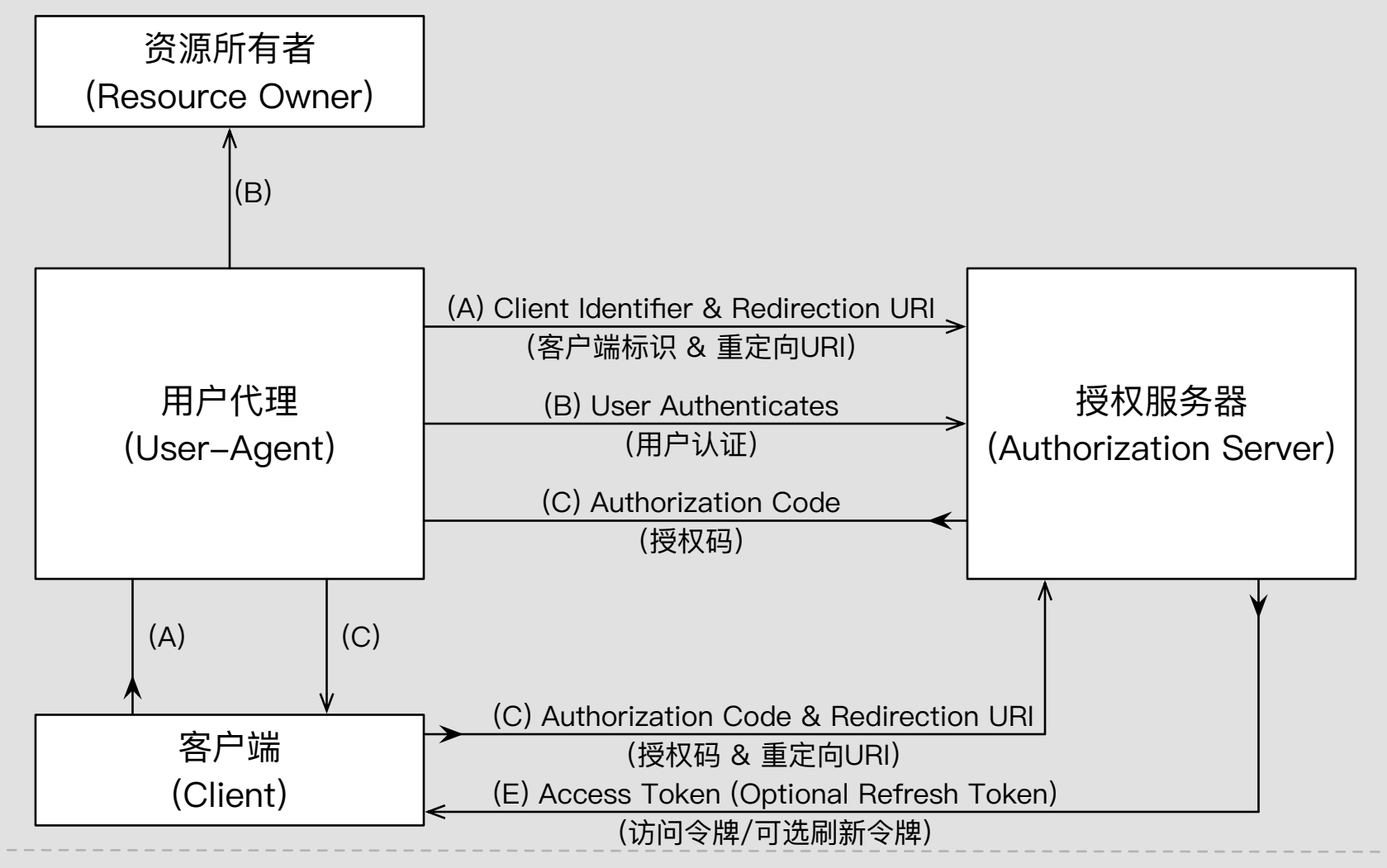
OAuth 2.0 定义的许可授权(Authorization Grant) 的主要类型 & 实现

授权码许可类型
(Authorization Code)
授权码是通过使用授权服务器作为客户端和资源所有者之间的中介来获得的。客户端不是直接从资源所有者请求授权，而是将资源所有者引导到授权服务器，然后将资源所有者引导回具有授权码的客户端。

在使用授权码将资源所有者引导回客户端之前，授权服务器对资源所有者进行身份认证(authenticate)并获得授权。因为资源所有者只向授权服务器进行身份认证，所以资源所有者的凭据永远不会与客户端共享。

授权码提供了一些重要的安全优势，例如认证客户端的能力，以及将访问令牌直接传输到客户端，而无需通过资源所有者的用户代理传递它并可能将其暴露给其他人，包括资源所有者。

实现



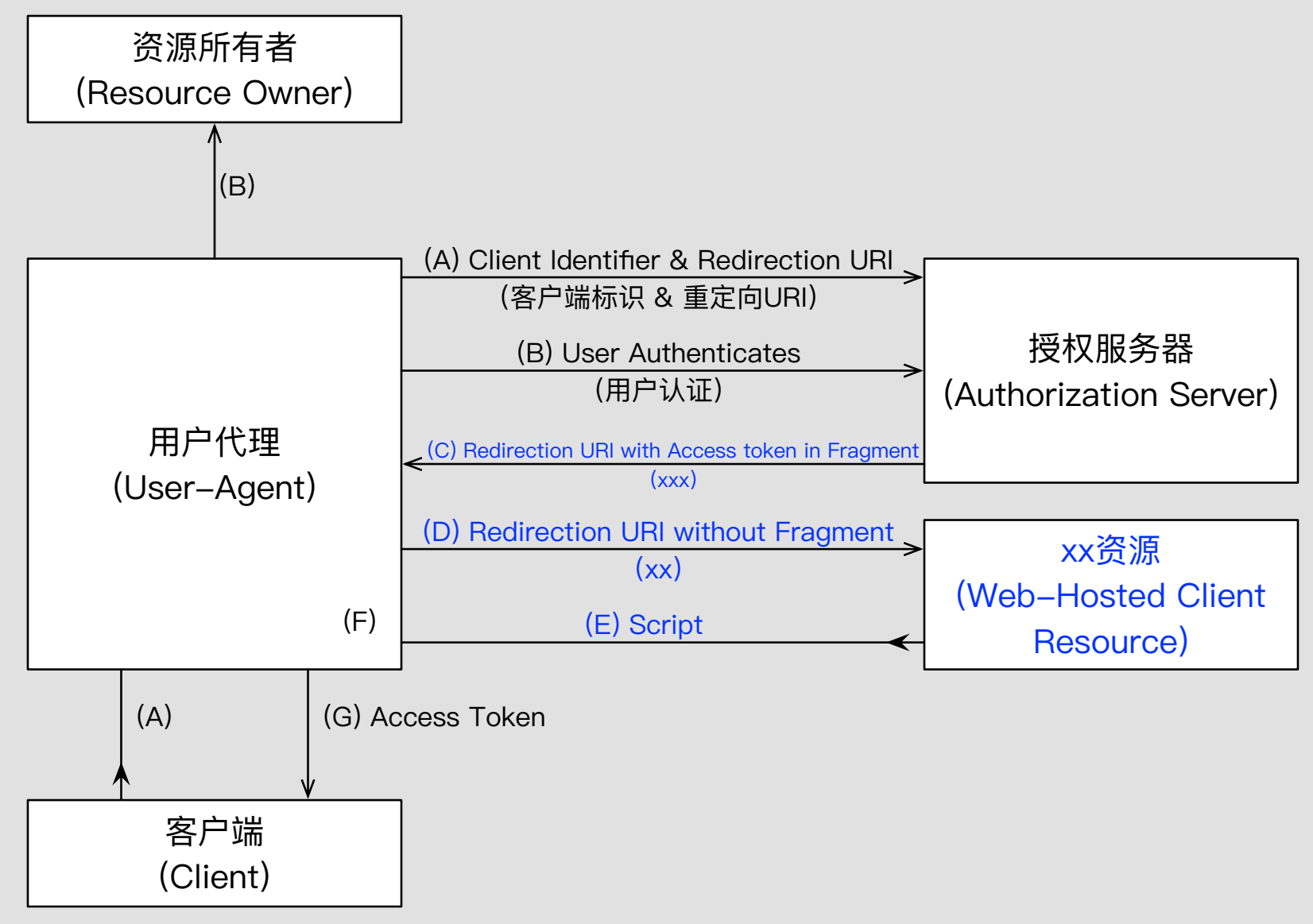
授权码模式的许可授权(Authorization Grant)过程，包括以下步骤：
(A) 客户端通过将资源所有者的用户代理定向到**授权端点(the authorization endpoint)**来启动流程。客户端包括客户端标识符、请求的范围、本地状态和重定向 URI，一旦授予(或拒绝)访问权限，授权服务器会将用户代理发送回该 URI。
(B) 授权服务器对资源所有者(通过用户代理)进行身份认证，并确定资源所有者是允许还是拒绝客户端的访问请求。
(C) 假设资源所有者授予访问权限，授权服务器使用之前提供的重定向 URI (在请求中或在客户端注册期间)将用户代理重定向回客户端。重定向 URI 包括授权码和客户端之前提供的所有本地状态。
(D) 客户端通过包含在上一步中收到的授权码，从授权服务器的**令牌端点(token endpoint)**请求访问令牌。发出请求时，客户端向授权服务器进行身份验证。客户端包含用于获取授权码进行验证的重定向 URI。
(E) 授权服务器对客户端进行身份认证，验证授权码，并确保接收到的重定向 URI 与步骤 (C) 中用于重定向客户端的 URI 匹配。如果有效，授权服务器将使用访问令牌和可选的刷新令牌进行响应。

隐式许可类型
(Implicit)
隐式授权是一种简化的授权码流程，针对使用 JavaScript 等脚本语言在浏览器中实现的客户端进行了优化。在隐式流程中，不是向客户端颁发授权码，而是直接向客户端颁发访问令牌(作为资源所有者授权的结果)。授权类型是隐式的，因为没有发布中间凭证(例如授权代码)(稍后用于获取访问令牌)。

在隐式授权流程期间发布访问令牌时，授权服务器不会对客户端进行身份认证。在某些情况下，可以通过用于将访问令牌传递给客户端的重定向 URI 来认证客户端身份。访问令牌可能会暴露给资源所有者或其他有权访问资源所有者的用户代理的应用程序。

隐式授权提高了某些客户端（例如作为浏览器内应用程序实现的客户端）的响应能力和效率，因为它减少了获取访问令牌所需的往返次数。然而，这种便利性应该与使用隐式授权的安全影响进行权衡。

实现

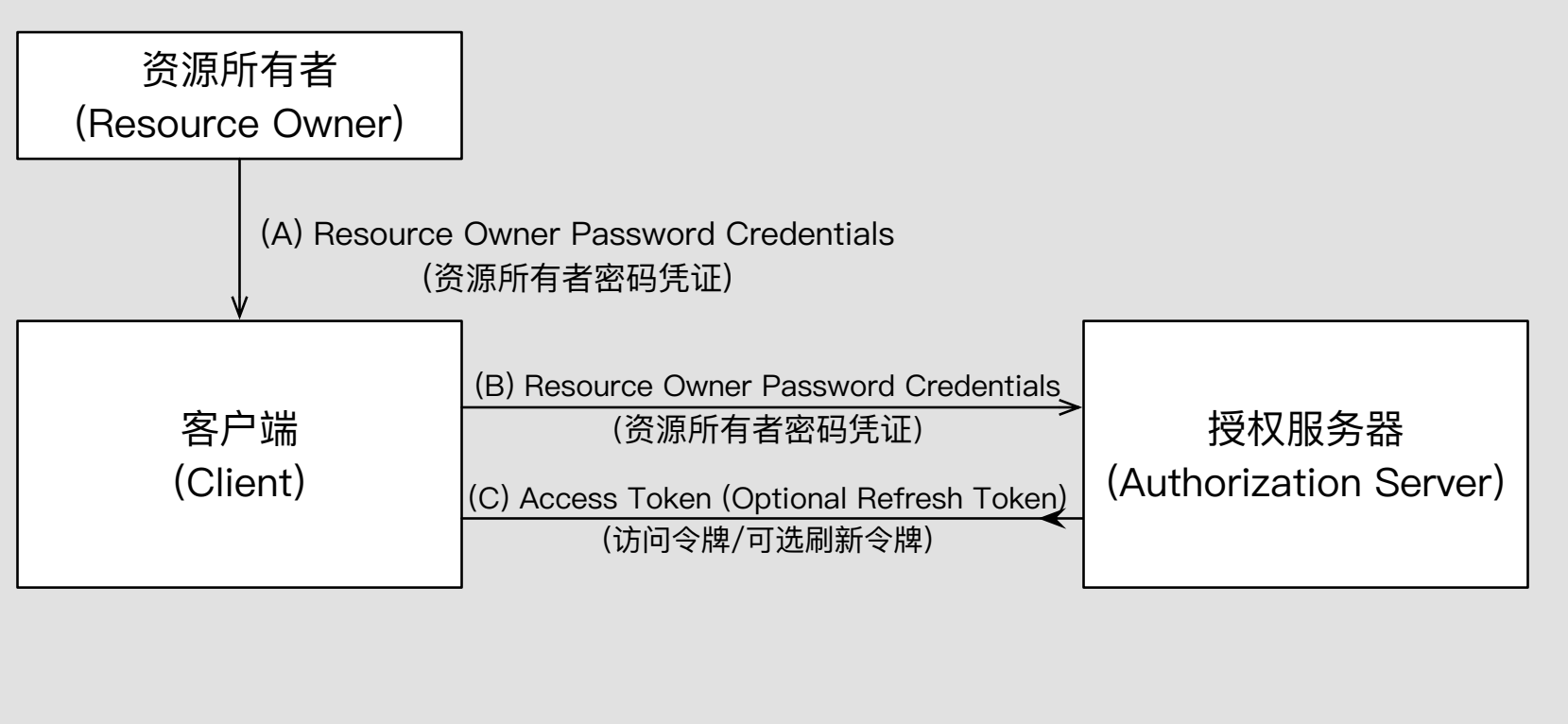


隐式模式的许可授权(Authorization Grant)过程，包括以下步骤：
(A) 客户端通过将资源所有者的用户代理定向到**授权端点(the authorization endpoint)**来启动流程。客户端包括其客户端标识符、请求的范围、本地状态和重定向 URI，一旦授予（或拒绝）访问权限，授权服务器会将用户代理发送回该 URI。
(B) 授权服务器对资源所有者(通过用户代理)进行身份认证，并确定资源所有者是允许还是拒绝客户端的访问请求。
(C) 假设资源所有者授予访问权限，授权服务器使用之前提供的重定向 URI 将用户代理重定向回客户端。重定向 URI 在 URI 片段(URI Fragment)中包含访问令牌。
(D) 用户代理通过向 Web 托管的客户端资源(不包括 [RFC2616] 定义的片段)发出请求来遵循重定向指令。用户代理在本地保留片段(Fragment)信息。
(E) Web 托管的客户端资源返回一个网页(通常是带有嵌入脚本的 HTML 文档)，该网页能够访问完整的重定向 URI，包括用户代理保留的片段，并提取访问令牌(和其他参数)包含在片段中。
(F) 用户代理在本地执行由 Web 托管的客户端资源提供的脚本，提取访问令牌。
(G) 用户代理将访问令牌传递给客户端。

资源所有者凭据许可类型
(Resource Owner Password Credentials)
资源所有者密码凭证（即用户名和密码）可以直接用作授权授予来获取访问令牌。 仅当资源所有者和客户端之间存在高度信任时（例如，客户端是设备操作系统或高特权应用程序的一部分）以及其他授权授予类型不可用时，才应使用凭据（ 例如授权码）。

尽管此授权类型需要客户端直接访问资源所有者凭据，但资源所有者凭据用于单个请求并交换访问令牌。 这种授权类型可以通过与长期访问令牌或刷新令牌交换凭证来消除客户端存储资源所有者凭证以供将来使用的需要。

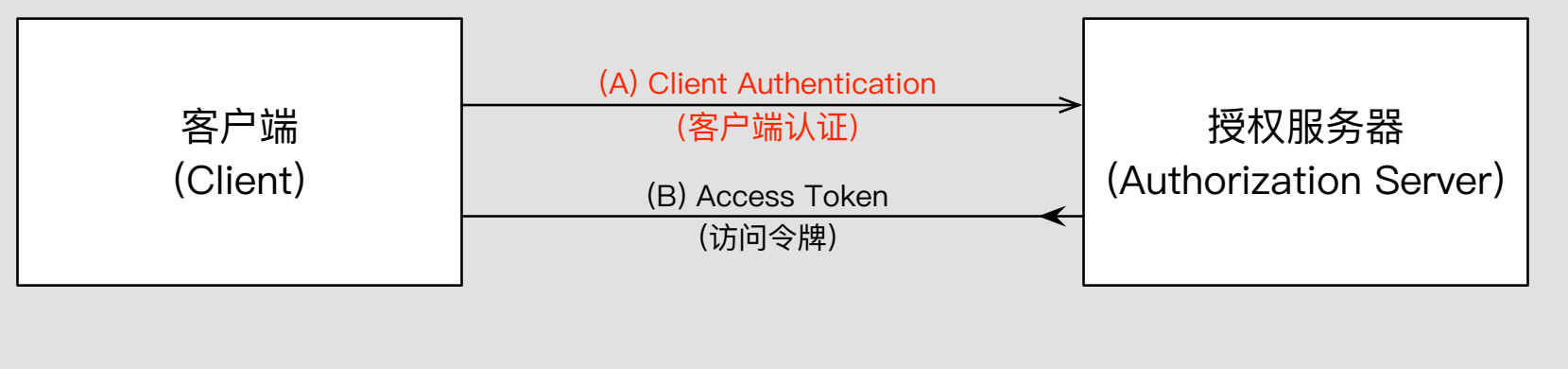
实现



资源所有者凭据模式的许可授权(Authorization Grant)过程，包括以下步骤：
(A) 资源所有者向客户端提供其用户名和密码。
(B) 客户端通过包含从资源所有者收到的凭据，从授权服务器的**令牌端点(token endpoint)**请求访问令牌。 发出请求时，客户端向授权服务器进行**身份认证**。
(C) 授权服务器对客户端进行**身份验证并验证资源所有者凭据**，如果有效，则颁发访问令牌。

客户端凭据许可类型
(Client Credentials)
当授权范围仅限于客户端控制下的受保护资源，或授权服务器之前安排的受保护资源时，客户端凭据（或其他形式的客户端身份验证）可以用作授权许可。 客户端凭据适用的场景：当客户端代表自己行事（客户端也是资源所有者）或基于先前与授权服务器安排的授权请求访问受保护资源时。

实现



客户端凭据模式的许可授权(Authorization Grant)过程，包括以下步骤：
(A) 客户端向授权服务器进行身份认证，并从**令牌端点(token endpoint)**请求访问令牌。
(B) 授权服务器对客户端进行身份认证，如果有效，则颁发访问令牌。

扩展机制

扩展类型

授权选型

扩展授权许可类型

客户端通过使用绝对 URI（由授权服务器定义）作为令牌端点的 ”grant_type” 参数的值来指定授权类型，并添加任何必要的附加参数，从而使用扩展授权类型。

例如，要使用 [OAuth–SAML2] 定义的安全断言标记语言 (SAML) 2.0 断言授权类型请求访问令牌，客户端可以使用 TLS 发出以下 HTTP 请求（额外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=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-
    bearer&assertion=PEFzc2VydGlvbiBJc3N1ZUluc3RhbnQ9IjIwMTEtMDU
    [...omitted for brevity...]aG5TdGF0ZW1lbnQ-PC9Bc3NlcnRpb24-
```

如果访问令牌请求有效且已授权，则授权服务器会发出访问令牌和可选的刷新令牌。
如果请求未通过客户端身份认证或无效，授权服务器将返回错误响应。

断言许可类型(Assertion)
Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants
https://datatracker.ietf.org/doc/html/rfc7521

断言许可类型是由OAuth 工作组发布的第一个官方扩展许可类型。在这种许可类型下，客户端会得到一条结构化的且被加密保护的信息，叫做“断言”，使用断言向授权服务器换取令牌。可以把断言想象为某种经过认证的文档，例如文凭或者许可证。只要你信任认证机构能确保声明的真实性，就可以相信文档中的内容也是真实的。

如何选择授权许可(Authorization Grant)类型?