- 1.简介 (1. Introduction)

简介

  - 角色 (1.1. Roles)
  - 协议流程 (1.2. Protocol Flow)
- 1.3. 授权许可 (1.3. Authorization Grant)
- 1.3.1. 授权码 (1.3.1. Authorization Code)
- 1.3.2. 隐式许可 (1.3.2. Implicit)
- 1.3.3. 资源所有者密码凭据 (1.3.3. Resource Owner Password Credentials)
- 1.3.4. 客户端凭据 (1.3.4. Client Credentials)
- 1.4. 访问令牌 (1.4. Access Token)
- 1.5. 刷新令牌 (1.5. Refresh Token)
- 1.6. TLS版本 (1.6. TLS Version)
- 1.7. HTTP重定向 (1.7. HTTP Redirections)
- 1.8. 互操作性 (1.8. Interoperability)
- 1.9. 符号约定

1.简介 (1. Introduction)

1. 在传统的客户端-服务器身份验证模式中，客户端请求服务器上访问受限的资源（受保护的资源）时，需要使用资源所有者的凭据在服务器上进行身份验证。资源所有者为了给第三方应用提供受限资源的访问权限，需要与第

OAuth通过引入授权层以及从资源所有者角色分离出客户端角色来解决这些问题。在OAuth中，客户端请求对受资源所有者控制且托管在资源服务器上的资源的访问权限，并授予一组不同于资源所有者所拥有的凭据。

作为使用资源所有者的凭据访问受保护资源的替代，客户端获得一个访问令牌——一个代表特定作用域、生命周期以及其他访问权限属性的字符串。访问令牌由授权服务器在资源所有者认可的情况下颁发给第三方客户端。客户

例如，终端用户（资源所有者）可以许可一个打印服务（客户端）访问她存储在图片共享网站（资源服务器）上的受保护图片，而无需与打印服务分享自己的用户名和密码，而是，她直接与图片分享网站信任的服务器（授权服务器）进

本规范是为RFC 2616协议设计的。在性质非HTTP协议上使用OAuth不在本规范的范围之内。

OAuth 1.0协议（RFC5849）作为一个指导性文档发布，是一个小的特设团体的工作成果。本标准化规范在OAuth 1.0的部署经验之上构建，也包括从更广泛的IETF社区收集到其他用户案例和可扩展性需

1.0的实现者应该理解本文档，而非对有关OAuth 2.0的结构和细节做任何假设。

1. Introduction

```
   In the traditional client-server authentication model, the client   requests an access-
restricted resource (protected resource) on the   server by authenticating with the server using
the resource owner's
   credentials.  In order to provide third-party applications access to   restricted resources,
the resource owner shares its credentials with   the third party.  This creates several problems
and limitations:
   o  Third-party applications are required to store the resource      owner's credentials for
future use, typically a password in
      clear-text.
   o  Servers are required to support password authentication, despite      the security
weaknesses inherent in passwords.
   o  Third-party applications gain overly broad access to the resource      owner's protected
```

resources, leaving resource owners without any ability to restrict duration or access to a limited subset of
resources.

- Resource owners cannot revoke access to an individual third party without revoking access to all third parties, and must do so by
changing the third party's password.

- Compromise of any third-party application results in compromise of the end-user's password and all of the data protected by that
password.

OAuth addresses these issues by introducing an authorization layer and separating the role of the client from that of the resource owner. In OAuth, the client requests access to resources controlled
by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource
owner.

Instead of using the resource owner's credentials to access protected resources, the client obtains an access token -- a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the
approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

For example, an end-user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo- sharing service (resource server), without sharing her username and
password with the printing service. Instead, she authenticates directly with a server trusted by the photo-sharing service (authorization server), which issues the printing service delegation-
specific credentials (access token).

This specification is designed for use with HTTP ([RFC2616]). The use of OAuth over any protocol other than HTTP is out of scope.

The OAuth 1.0 protocol ([RFC5849]), published as an informational document, was the result

of a small ad hoc community effort. This Standards Track specification builds on the OAuth 1.0 deployment
experience, as well as additional use cases and extensibility requirements gathered from the wider IETF community. The OAuth 2.0 protocol is not backward compatible with OAuth 1.0. The two versions
may co-exist on the network, and implementations may choose to support both. However, it is the intention of this specification that new implementations support OAuth 2.0 as specified in this
document and that OAuth 1.0 is used only to support existing deployments. The OAuth 2.0 protocol shares very few implementation details with the OAuth 1.0 protocol. Implementers familiar with
OAuth 1.0 should approach this document without any assumptions as to
its structure and details.

1.1. 角色 (1.1. Roles)

OAuth定义了四种角色:

- 资源所有者能够授予对受保护资源的访问权限的实体。当资源所有者是人时，它被称为最终用户。
  使用资源所有者的授权代表资源所有者发起对受保护资源的请求的应用程序。术语“客户端”并非特指任何特定的的实现特点（例如：应用程序是否是在服务器、台式机或其他设备上执行）。

授权服务器和资源服务器之间的交互超出了本规范的范围。授权服务器可以和资源服务器是同一台服务器，也可以是分离的个体。一个授权服务器可以颁发被多个资源服务器接受的访问令牌。

1.1. Roles

```
   OAuth defines four roles:
   resource owner
      An entity capable of granting access to a protected resource.      When the resource owner
is a person, it is referred to as an
      end-user.
   resource server
      The server hosting the protected resources, capable of accepting      and responding to
protected resource requests using access tokens.
   client
      An application making protected resource requests on behalf of the      resource owner and
with its authorization.  The term "client" does      not imply any particular implementation
characteristics (e.g.,      whether the application executes on a server, a desktop, or
other
      devices).
   authorization server      The server issuing access tokens to the client after
successfully
      authenticating the resource owner and obtaining authorization.
   The interaction between the authorization server and resource server   is beyond the scope of
this specification.  The authorization server   may be the same server as the resource server or
a separate entity.
```

```
    A single authorization server may issue access tokens accepted by   multiple resource servers.
```

1.2. 协议流程 (1.2. Protocol Flow)

1.2. 协议流程

图1: 抽象的协议流程

图1中所示的抽象的OAuth 2.0流程描述了四种角色之间的交互，包括以下步骤：

- (A) 客户端从资源所有者处请求授权。授权请求可以直接向资源所有者发起（如图所示），或者更可取的是通过授权服务器作为中介间接发起。
- (B) 客户端收到授权许可，这是一个代表资源所有者的授权的凭据，使用本规范中定义的四种许可类型之一或者使用扩展许可类型表示。授权许可类型取决于客户端请求授权所使用的方法以及授权服务器支持的类型。

- □C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □D□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □E□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □F□□□□□□□□□□□□□□□□□□□□□□□

□□4.1□3□□□□□□□□□□□□A□□□B□□□□□□□□□□□□□□□□□□□□□□□□

--4.1. □ □3

```
+----------+
| Resource |
|   Owner  |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier      +---------------+
|         -+----(A)-- & Redirection URI ---->|               |
|  User-   |                                 | Authorization |
|  Agent  -+----(B)-- User authenticates --->|     Server    |
|          |                                 |               |
|         -+----(C)-- Authorization Code ---<|               |
+-|----|---+                                 +---------------+
  |    |                                         ^      v
 (A)  (C)                                        |      |
  |    |                                         |      |
  ^    v                                         |      |
+---------+                                      |      |
|         |>---(D)-- Authorization Code ---------'      |
|  Client |          & Redirection URI                  |
|         |                                             |
|         |<---(E)----- Access Token -------------------'
```

```
+---------+       (w/ Optional Refresh Token)
```

注: 说明步骤 (A) 、 (B) 和 (C) 的直线因为通过用户代理而被分为两部分。

图3: 授权码流程

1.2. Protocol Flow

```
     +--------+                               +---------------+     |        |--(A)-
Authorization Request ->|   Resource    |     |        |                               |
Owner     |
     |        |<-(B)-- Authorization Grant ---|               |     |
|                               +---------------+
     |        |
     |        |                               +---------------+     |        |--(C)--
Authorization Grant -->| Authorization |     | Client |                               |
Server    |
     |        |<-(D)----- Access Token -------|               |     |
|                               +---------------+
     |        |
     |        |                               +---------------+     |        |--(E)----- Access
Token ------>|    Resource   |     |        |                               |     Server
|
     |        |<-(F)--- Protected Resource ---|               |     +--------
+                               +---------------+
                     Figure 1: Abstract Protocol Flow
   The abstract OAuth 2.0 flow illustrated in Figure 1 describes the   interaction between the
four roles and includes the following steps:
   (A)  The client requests authorization from the resource owner.  The        authorization
request can be made directly to the resource owner        (as shown), or preferably indirectly
via the authorization
        server as an intermediary.
   (B)  The client receives an authorization grant, which is a        credential representing
```

the resource owner's authorization, expressed using one of four grant types defined in this
specification or using an extension grant type. The authorization grant type depends on the method used by the client to request authorization and the types supported by the
authorization server.

(C) The client requests an access token by authenticating with the authorization server and presenting the authorization grant.

(D) The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.

(E) The client requests the protected resource from the resource server and authenticates by presenting the access token.

(F) The resource server validates the access token, and if valid, serves the request.

The preferred method for the client to obtain an authorization grant from the resource owner (depicted in steps (A) and (B)) is to use the authorization server as an intermediary, which is illustrated in Figure 3 in Section 4.1.

1.3. 授权许可（1.3. Authorization G

1.3. 授权许可

授权许可是一个代表资源所有者授权（访问受保护资源）的凭据，客户端用它来获取访问令牌。本规范定义了四种许可类型——授权码、隐式许可、资源所有者密码凭据和客户端凭据——以及用于定义其他类型的可扩展性机制_

1.3. Authorization Grant

```
   An authorization grant is a credential representing the resource   owner's authorization (to
access its protected resources) used by the   client to obtain an access token.  This
specification defines four   grant types -- authorization code, implicit, resource owner
password
   credentials, and client credentials -- as well as an extensibility
   mechanism for defining additional types.
```

1.3.1. [illegible] [illegible]1.3.1. Authorizatior Code[illegible]

1.3.1. [illegible]

[illegible]RFC2616[illegible]

[illegible]

1.3.1. Authorization Code

```
   The authorization code is obtained by using an authorization server   as an intermediary
between the client and resource owner.  Instead of   requesting authorization directly from the
resource owner, the client
   directs the resource owner to an authorization server (via its   user-agent as defined in
[RFC2616]), which in turn directs the   resource owner back to the client with the authorization
code.
   Before directing the resource owner back to the client with the   authorization code, the
authorization server authenticates the   resource owner and obtains authorization.  Because the
resource owner
   only authenticates with the authorization server, the resource   owner's credentials are
never shared with the client.
   The authorization code provides a few important security benefits,   such as the ability to
authenticate the client, as well as the   transmission of the access token directly to the
```

```
client without
   passing it through the resource owner's user-agent and potentially
   exposing it to others, including the resource owner.
```

1.3.2. [illegible]1.3.2. Implicit[illegible]

1.3.2. [illegible]

[illegible]JavaScript[illegible]

[illegible]URI[illegible]

[illegible]10.310.16[illegible]

1.3.2. Implicit

```
   The implicit grant is a simplified authorization code flow optimized   for clients
implemented in a browser using a scripting language such   as JavaScript.  In the implicit flow,
instead of issuing the client   an authorization code, the client is issued an access token
directly
Hardt                            Standards Track                    [Page 8]
```

```
RFC 6749                          OAuth 2.0                    October 2012
   (as the result of the resource owner authorization).  The grant type   is implicit, as no
intermediate credentials (such as an authorization   code) are issued (and later used to obtain
an access token).
```

When issuing an access token during the implicit grant flow, the authorization server does not authenticate the client. In some cases, the client identity can be verified via the redirection URI

used to deliver the access token to the client. The access token may be exposed to the resource owner or other applications with access to

the resource owner's user-agent.

Implicit grants improve the responsiveness and efficiency of some clients (such as a client implemented as an in-browser application), since it reduces the number of round trips required to obtain an

access token. However, this convenience should be weighed against the security implications of using implicit grants, such as those described in Sections 10.3 and 10.16, especially when the authorization code grant type is available.

1.3.3. □□□□□□□□ □1.3.3. Resour

Owner Password Credentials□

1.3.3. Resource Owner Password Credentials

```
   The resource owner password credentials (i.e., username and password)   can be used directly
as an authorization grant to obtain an access   token.  The credentials should only be used when
there is a high
   degree of trust between the resource owner and the client (e.g., the   client is part of the
device operating system or a highly privileged   application), and when other authorization
grant types are not
   available (such as an authorization code).
   Even though this grant type requires direct client access to the   resource owner
credentials, the resource owner credentials are used   for a single request and are exchanged
for an access token.  This
   grant type can eliminate the need for the client to store the   resource owner credentials
for future use, by exchanging the   credentials with a long-lived access token or refresh token.
```

1.3.4. 客户端凭据 (1.3.4. Client Credentials)

当授权范围，限于客户端控制下的受保护资源或事先与 授权服务器商定的受保护资源时，客户端凭据，可以被用作为一种授权许可。典型的场景，当客户端代表自己的（功能角色acting）（时（acting on its ov

1.3.4. Client Credentials

```
   The client credentials (or other forms of client authentication) can   be used as an
authorization grant when the authorization scope is   limited to the protected resources under
the control of the client,   or to protected resources previously arranged with the
authorization
   server.  Client credentials are used as an authorization grant   typically when the client is
acting on its own behalf (the client is   also the resource owner) or is requesting access to
protected
   resources based on an authorization previously arranged with the
   authorization server.
```

1.4. [illegible] （1.4. Access Token）

1.4. [illegible]

[illegible]

[illegible] RFC6750

1.4. Access Token

```
   Access tokens are credentials used to access protected resources.  An   access token is a
string representing an authorization issued to the   client.  The string is usually opaque to
the client.  Tokens
   represent specific scopes and durations of access, granted by the   resource owner, and
enforced by the resource server and authorization
   server.
   The token may denote an identifier used to retrieve the authorization   information or may
self-contain the authorization information in a   verifiable manner (i.e., a token string
consisting of some data and a   signature).  Additional authentication credentials, which are
beyond
   the scope of this specification, may be required in order for the
   client to use a token.
   The access token provides an abstraction layer, replacing different   authorization
constructs (e.g., username and password) with a single   token understood by the resource
```

server. This abstraction enables   issuing access tokens more restrictive than the authorization grant

   used to obtain them, as well as removing the resource server's need   to understand a wide range of authentication methods.

   Access tokens can have different formats, structures, and methods of   utilization (e.g., cryptographic properties) based on the resource   server security requirements. Access token attributes and the

   methods used to access protected resources are beyond the scope of   this specification and are defined by companion specifications such   as [RFC6750].

1.5. 刷新令牌（1.5. Refresh Token）

1.5. 刷新令牌

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颁发给客户端，用于在当前访问令牌失效或过期时，获取一个新的访问令牌，或者获得相等或更窄范围的额外的访问令牌（访问令牌可能具有比资源所有者所授权的更短的生_

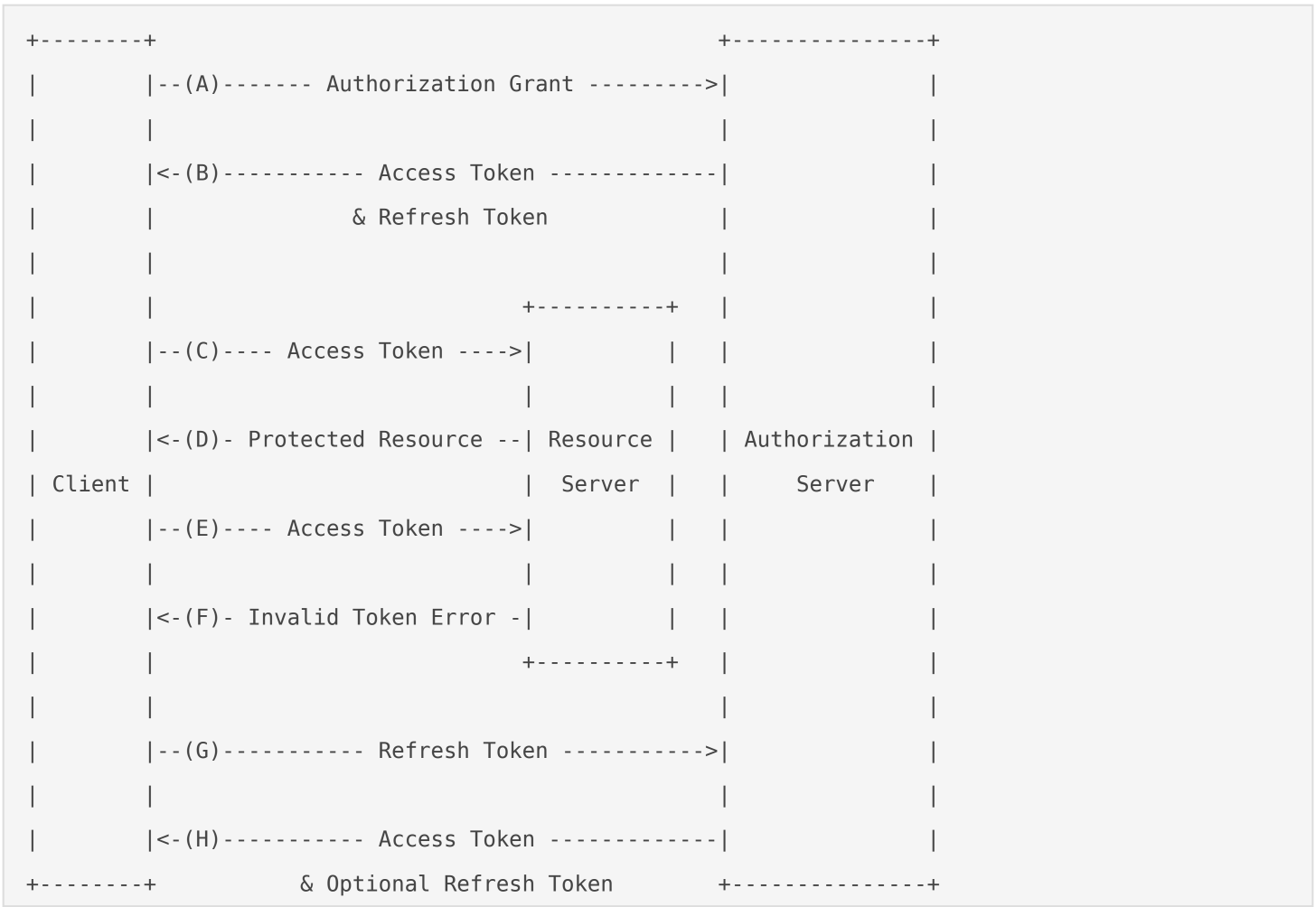

图2：刷新过期的访问令牌

图2中的所示流程包含以下步骤：

- (A) 客户端通过与授权服务器进行身份验证并出示授权许可请求访问令牌。
- (B) 授权服务器对客户端进行身份验证并验证授权许可，若有效则颁发访问令牌和刷新令牌。
- (C) 客户端通过出示访问令牌向资源服务器发起受保护资源的请求。
- (D) 资源服务器验证访问令牌，若有效则满足该要求。
- (E) 步骤 (C) 和 (D) 重复进行，直到访问令牌到期。如果客户端知道访问令牌已过期，跳到步骤 (G)，否则它将继续发起另一个对受保护资源的请求。
- (F) 由于访问令牌是无效的，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验证并出示刷新令牌，请求一个新的访问令牌。客户端身份验证要求基于客户端的类型和授权服务器的策略。
- (H) 授权服务器对客户端进行身份验证并验证刷新令牌，若有效则颁发一个新的访问令牌（和——可选地——一个新的刷新令牌）。

步骤 (C) 第[7]节广、(E) 和 (F) 在本规范的范围以外，如[5]所述。

1.5. Refresh Token

```
   Refresh tokens are credentials used to obtain access tokens.  Refresh
   tokens are issued to the client by the authorization server and are
   used to obtain a new access token when the current access token
   becomes invalid or expires, or to obtain additional access tokens
   with identical or narrower scope (access tokens may have a shorter
   lifetime and fewer permissions than authorized by the resource
   owner).  Issuing a refresh token is optional at the discretion of the
   authorization server.  If the authorization server issues a refresh
   token, it is included when issuing an access token (i.e., step (D) in
   Figure 1).
   A refresh token is a string representing the authorization granted to
   the client by the resource owner.  The string is usually opaque to
   the client.  The token denotes an identifier used to retrieve the
Hardt                        Standards Track                   [Page 10]
```

authorization information. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers.

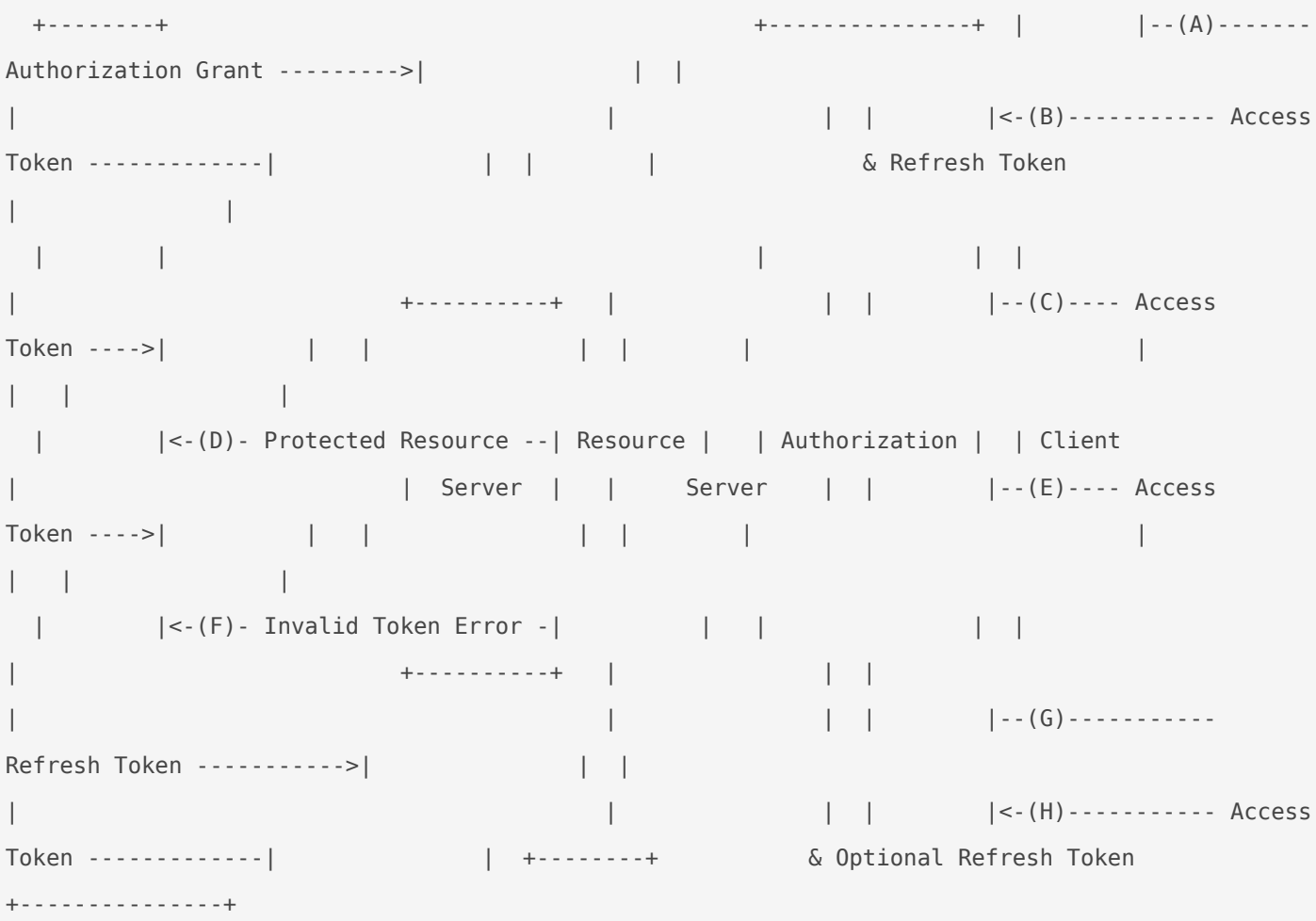

Figure 2: Refreshing an Expired Access Token

The flow illustrated in Figure 2 includes the following steps:

(A) The client requests an access token by authenticating with the authorization server and presenting an authorization grant.

(B) The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token and a refresh token.

(C) The client makes a protected resource request to the resource server by presenting the access token.

(D) The resource server validates the access token, and if valid, serves the request.

(E) Steps (C) and (D) repeat until the access token expires. If the client knows the

access token expired, it skips to step (G); otherwise, it makes another protected resource request.

(F) Since the access token is invalid, the resource server returns an invalid token error.

(G) The client requests a new access token by authenticating with the authorization server and presenting the refresh token. The client authentication requirements are based on the client type and on the authorization server policies.

(H) The authorization server authenticates the client and validates the refresh token, and if valid, issues a new access token (and, optionally, a new refresh token).

Steps (C), (D), (E), and (F) are outside the scope of this specification, as described in Section 7.

1.6. TLS□□□1.6. TLS Version□

1.6. TLS□□

□□□□□□□□□□□□□□□RFC5246□□□□□□□□RFC2246□□□□□□□□□□□□□□□□□TLS 1.0□□□□□TLS 1.2□

□□□□□□□□□□□□□□□□□□□□□□□□□

1.6. TLS Version

```
   Whenever Transport Layer Security (TLS) is used by this   specification, the appropriate
version (or versions) of TLS will vary   over time, based on the widespread deployment and known
security
   vulnerabilities.  At the time of this writing, TLS version 1.2   [RFC5246] is the most recent
version, but has a very limited
   deployment base and might not be readily available for   implementation.  TLS version 1.0
[RFC2246] is the most widely   deployed version and will provide the broadest
interoperability.
   Implementations MAY also support additional transport-layer security
   mechanisms that meet their security requirements.
```

1.7. HTTP重定向 (1.7. HTTP Redirections)

1.7. HTTP重定向

本规范广泛采用了HTTP重定向，有此客户端或授权服务器引导资源所有者的用户代理到另一个目的地址。虽然本规范中的例子演示了HTTP 302状态码的使用，但是任何其他通过用户代理完成重定向的方法都是允许的_

1.7. HTTP Redirections

```
   This specification makes extensive use of HTTP redirections, in which   the client or the
authorization server directs the resource owner's   user-agent to another destination.  While
the examples in this
   specification show the use of the HTTP 302 status code, any other   method available via the
user-agent to accomplish this redirection is
   allowed and is considered to be an implementation detail.
```

1.8. 互操作性 (1.8. Interoperability

1.8. 互操作性

OAuth 2.0提供了丰富的具有明确的安全性质的授权框架。然而，尽管在其自身看来是一个带有许多可选择组件的丰富且高度可扩展的框架，本规范有可能产生许多非可互操作的实现。

此处，本规范中留下一些必需组件部分或完全没有定义（例如，客户注册，授权服务器性能，端点发现等）。没有这些组件，客户端必须针对特定的授权服务器和资源服务器被手动并专门配置，以进行互操作。本框架设计具有一个明确的期望，即未来工作将定义实现完整的Web范围的互操作性所需的规范性的配置和扩展。

1.8. Interoperability

```
   OAuth 2.0 provides a rich authorization framework with well-defined   security properties.
However, as a rich and highly extensible   framework with many optional components, on its own,
this
   specification is likely to produce a wide range of non-interoperable
   implementations.
   In addition, this specification leaves a few required components   partially or fully
undefined (e.g., client registration,   authorization server capabilities, endpoint discovery).
Without
Hardt                        Standards Track                   [Page 12]
```

```
RFC 6749                         OAuth 2.0                   October 2012
   these components, clients must be manually and specifically   configured against a specific
```

```
authorization server and resource
   server in order to interoperate.
   This framework was designed with the clear expectation that future   work will define
prescriptive profiles and extensions necessary to   achieve full web-scale interoperability.
```

1.9. [illegible]

1.9. [illegible]

[illegible] RFC2119 [illegible] RFC5234 [illegible] (ABNF) [illegible] RFC3986 [illegible] URI [illegible] URI [illegible]

RFC4949 [illegible]

[illegible]

1.9. Notational Conventions

```
   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",   "SHOULD", "SHOULD NOT",
"RECOMMENDED", "MAY", and "OPTIONAL" in this   specification are to be interpreted as described
in [RFC2119].
   This specification uses the Augmented Backus-Naur Form (ABNF)   notation of [RFC5234].
Additionally, the rule URI-reference is   included from "Uniform Resource Identifier (URI):
Generic Syntax"
   [RFC3986].
   Certain security-related terms are to be understood in the sense   defined in [RFC4949].
These terms include, but are not limited to,   "attack", "authentication", "authorization",
"certificate",
   "confidentiality", "credential", "encryption", "identity", "sign",   "signature", "trust",
"validate", and "verify".
   Unless otherwise noted, all the protocol parameter names and values
   are case sensitive.
```