GoogleOAuth2.0 认证指南

极客学院出版

前言

Google API 使用 OAuth 2.0 协议来进行验证和授权。谷歌支持常见的 OAuth 2.0 方案，包括使用 Web 服务器，本地安装的，和客户端类型的应用。

本指南是 Google Developers 官网 OAuth 2.0 Authorzation 的中文翻译版本。

适用人群

本教程是给那些想详细了解 Google OAuth 2.0 认证机制并调用 Google API 的开发人员编写的。

学习前提

在学习本教程之前，你需要对 OAuth 2.0 认证机制相关的知识有一定了解。

你将学会

- Google OAuth 2.0 认证机制
- 不同类型的应用程序中使用 OAuth 2.0
- OAuth 2.0 认证后调用 Google API

更新日期	更新内容
2015-06-26	Google OAuth 2.0 认证指南

目录

使用 OAuth 2.0 来访问谷歌 API

谷歌 APIs 使用 OAuth 2.0 协议来进行验证和授权。谷歌支持常见的 OAuth 2.0 方案，例如 web 服务器使用的，本地安装的，和客户端方面的应用。

首先，从谷歌开发者控制台获得 OAuth 2.0 客户端凭证。然后您的客户端应用程序向谷歌授权服务器请求获得一个访问令牌，接下来从回应中提取出令牌，并将其发送给你想要访问的谷歌 API。 若想动手演练如何通过 OAuth 2.0 访问谷歌，您可以在 OAuth 2.0 游乐场里面做实验（您可以选择使用自己的客户端凭证）。

该页面为您概要地展示各种谷歌支持的 OAuth 2.0 授权方案，并提供详细内容的链接。若想详细了解关于如何使用 OAuth 2.0 来进行验证，请参阅 OpenID 连接。

> 注意：由于能否正确实现该功能牵涉到安全问题，所以当您和谷歌的 OAuth 2.0 端点进行互动时，我们强烈建议您使用 OAuth 2.0 库。使用他人经过良好除错的代码是一项最佳实践，这样做能有助于保护您自己和您的用户。更多信息请参阅客户端库。

#

内容

#

基本步骤

所有应用程序在通过 OAuth 2.0 访问谷歌 API 时都遵循一个基本模式。抽象地说，你需要遵循下面四步:

1. 从谷歌开发者控制台获得 OAuth 2.0 凭证。

访问谷歌开发者控制台来获得 OAuth 2.0 凭证，例如应用程序和谷歌都知道的一个客户端 ID 和客户端 secret。数据集会根据你所建立的应用程序类型而有所不同，举例来说，一个 JavaScript 应用程序并不要求一个 secret，但是一个 Web 服务器应用程序就会要求。

2. 从谷歌授权服务器获得一个访问密钥。

在你的应用程序可以通过谷歌 API 访问私有数据之前，它必须获得一个访问令牌来许可你访问那个 API。一个单独的访问令牌可以为多个 API 许可不同程度的访问权限。一个叫做 `scope`（域）的变量参数可用来控制一个访问令牌所许可的控制资源集和操作集。在一个有访问令牌的请求中，你的应用程序可发送一个或多个 `scope` 参数。

有多种方法可以发出这个请求，这些方法根据你所创建的应用程序类型不同而有所不同。举例来说，一个 JavaScript 应用程序可以通过浏览器重定向到谷歌要求访问令牌，而安装在一个没有浏览器的设备上的应用程序则会使用 web 服务请求。

一些请求会要求用户登陆到他们的谷歌账户作为一个验证步骤。登陆之后，用户会被询问他们是否愿意许可你的应用程序所要求的权限。这个过程叫做“用户准许”。

如果用户批准了请求，谷歌授权服务器会发送一个访问令牌给您的应用程序（或一个应用程序可以使用的授权码来获得访问令牌）。如果用户没有批准请求，服务器会返回一个错误信息。

通常来说渐进地要求各种 scope 是一项最佳实践，比起一次性要求所有权限，当需要进行这种访问时才发出相应的请求会更好。 举例来说，一个想要支持购买功能的应用程序不应该在用户点击“购买”按钮之前要求谷歌钱包访问权; 详情参阅渐进式授权。

3. 发送访问令牌给一个 API。

当一个应用程序获取访问令牌时，会在一个 HTTP 授权头里嵌入令牌发送给谷歌 API。当然，以 URI 查询串参数来发送令牌也是可行的，但我们不推荐这样做，因为 URI 参数会在 log 中留下痕迹，而 log 本身并不是十分安全。同时，避免创建不必要的 URI 参数名是一项 REST 良好实践。

访问令牌仅对令牌所请求的 `scope` 中有描述的操作集和资源集有效。举例来说，如果一个用于 Google+ API 的访问令牌被分发下来，这个令牌并不能许可您访问谷歌通讯录 API。然而，你可以，将那个访问令牌多次发送给 Google+ API 来实现相似操作。Web

4. 刷新访问令牌，如果必要的话。

访问令牌的生命期是有限的。如果你的应用程序需要比一个访问令牌的生命期更长的时间来访问谷歌 API，可以通过获得一个刷新令牌(refresh token)来实现。刷新令牌可以让您的应用程序获得新的访问令牌。

> 注意: 将刷新令牌保存在一个安全的长期存储器内并持续地使用他们直到令牌过期。每对客户端-用户（client-user）能获得的刷新令牌的数量是有限的，同一个用户在多个客户端上能获得的刷新令牌也是有限的，而且这些限制都不一样。如果你的应用程序请求的刷新令牌数量超过上述的任何一个限制，限制外较旧的刷新令牌会自动失效。

#

方案

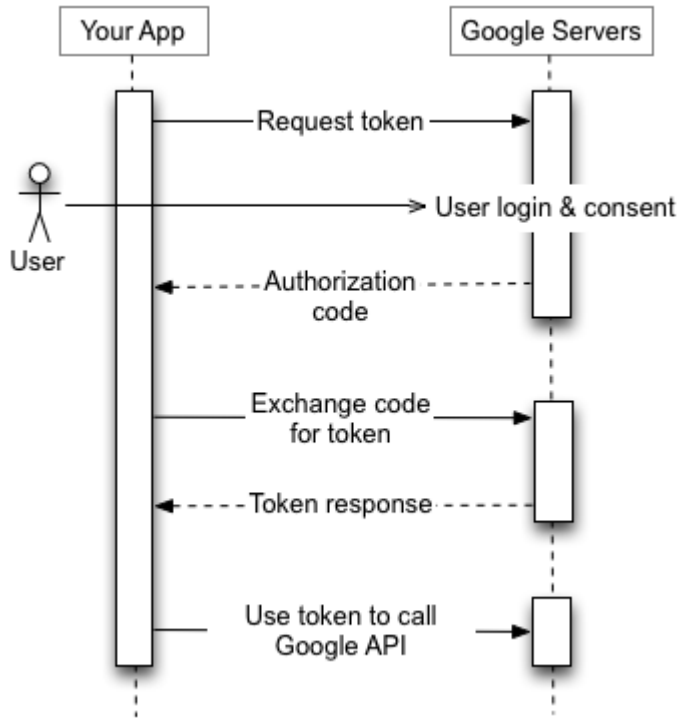
#

Web 服务器应用程序

谷歌 OAuth 2.0 端点支持使用下列语言和框架编写的 Web 服务器应用程序：PHP，Java，Python，Ruby，和 ASP.NET。

授权程序会在您的应用程序将浏览器重定向到谷歌 URL 时启动; URL 包含了查询参数来指出请求的访问类型。谷歌负责用户验证，会话选择，和用户准许。这一程序的结果是授权码，应用程序可以通过这个码来获得一个访问令牌和一个刷新令牌。

应用程序应存储刷新令牌供未来使用，并使用访问令牌来访问谷歌 API。一旦访问令牌过期，应用程序可使用刷新令牌来获得新的访问令牌。



详情请参阅使用 OAuth 2.0 for Web 服务器应用程序。

#

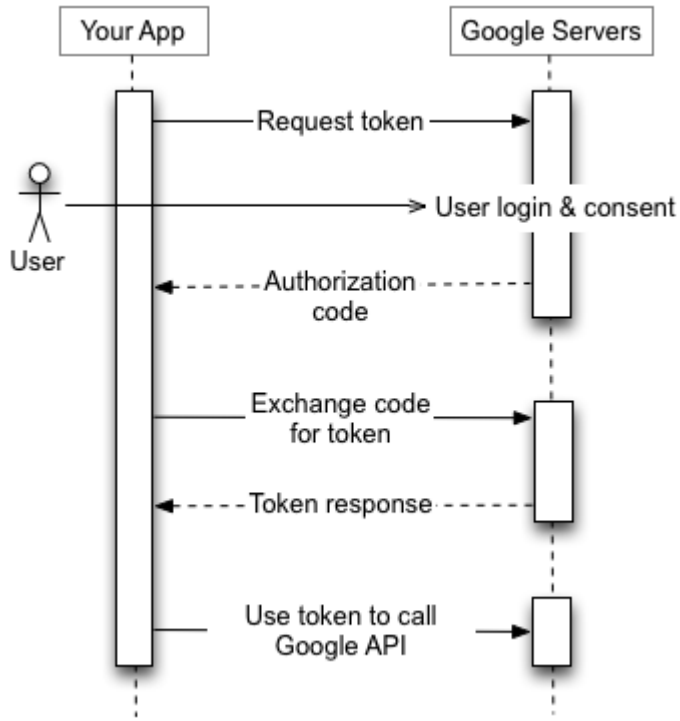
本地安装的应用程序

谷歌 OAuth 2.0 端点支持安装在各种设备上的应用程序，例如电脑、移动设备、和平板。当你通过谷歌开发者控制台创建客户端 ID 时，声明这是一个本地安装的应用程序，然后选择根据你的应用程序类别选择 Android，Chrome，iOS，或其他。

这个过程的结果是一个客户端 ID，在一些情况下，还会有一个客户端 secret，你需要将其嵌入到你的应用程序源代码中。（在这个情况下，客户端 secret 显然没被当做秘密看待。）

授权程序会在您的应用程序将浏览器重定向到谷歌 URL 时启动; URL 包含了查询参数来指出请求的访问类型。谷歌 负责用户验证，会话选择，和用户准许。这一程序的结果是授权码，应用程序可以通过这个码来获得一个访问令牌和一个刷新令牌。

应用程序应存储刷新令牌供未来使用，并使用访问令牌来访问谷歌 API。一旦访问令牌过期，应用程序可使用刷新令牌来获得新的访问令牌.



详情请参阅使用 OAuth 2.0 for 本地安装的应用程序。

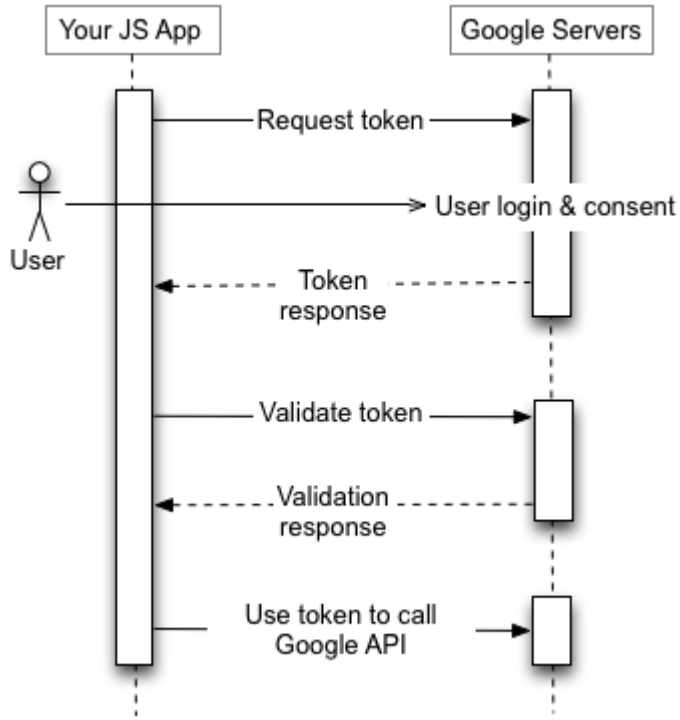
#

客户端方面（JavaScript）应用程序

谷歌 OAuth 2.0 端点支持在浏览器里运行的 JavaScript 应用程序。

授权程序会在您的应用程序将浏览器重定向到谷歌 URL 时启动; URL 包含了查询参数来指出请求的访问类型。谷歌 负责用户验证，会话选择，和用户准许。

这一程序的结果是访问令牌，客户端应该先对令牌进行验证再将其包含在谷歌 API 请求内。一旦令牌过期，应用程序可重复同一步骤。



详情请参阅使用 OAuth 2.0 for 客户端方面应用程序.

#

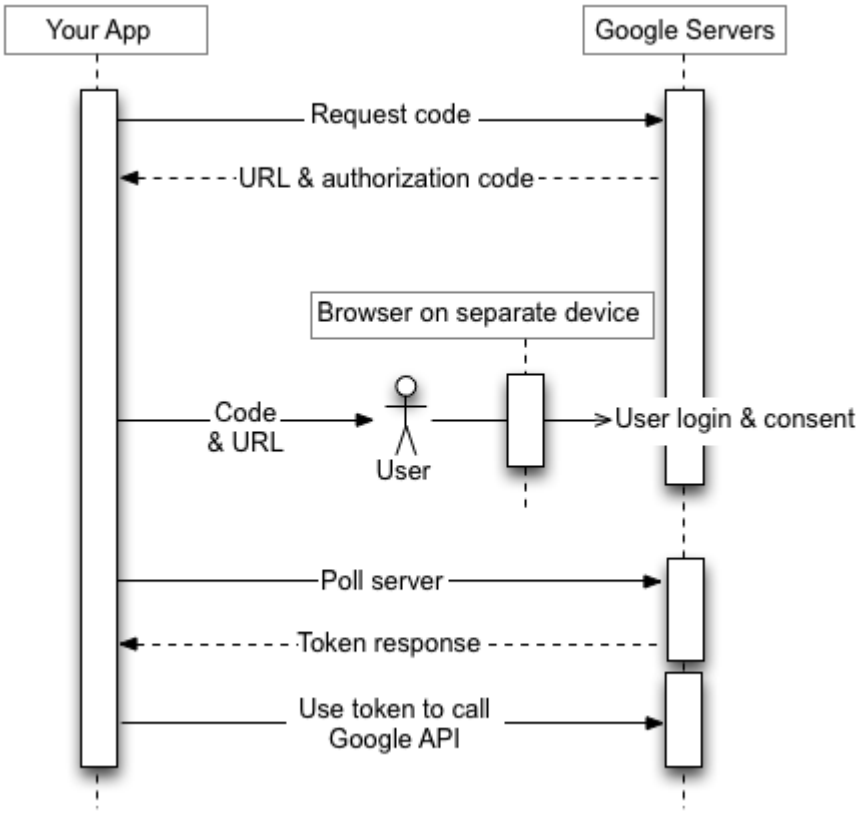
输入有限设备上的应用程序

谷歌 OAuth 2.0 端点支持在输入有限的设备上运行的应用程序，例如家用游戏机、摄影机、打印机等。

授权程序会在您的应用程序通过向谷歌 URL 发送 Web 服务请求来获取授权码时启动；服务器回应包含多个参数，包括了一个 URL 和一个代码让应用程序来显示给用户。

用户从设备获得 URL 和代码后，应切换到另一个拥有更强大输入功能的设备或者电脑，然后启动浏览器，访问获得的 URL 并登陆，然后输入获得的代码。

与此同时，应用程序会定期轮询谷歌 URL。若用户同意访问，谷歌服务器便会发回包含一个访问令牌和刷新令牌的回应。应用程序应存储刷新令牌供未来使用，并使用访问令牌来访问谷歌 API。一旦访问令牌过期，应用程序可使用刷新令牌来获得新的访问令牌.



详情请参阅使用 OAuth 2.0 for 设备。

#

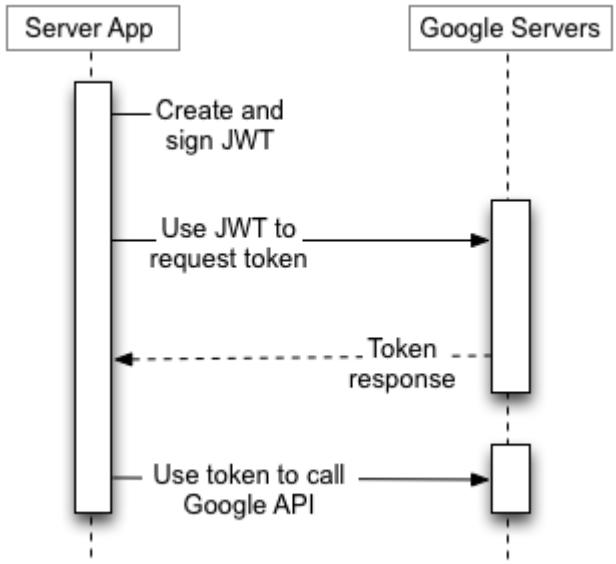
服务账户

谷歌 API 中例如预测 API 和谷歌云存储可以在您不访问用户信息的情形下代您的应用程序实现相关功能。在这种情况下你的应用程序需要向 API 证明自己的身份，不过这样做不需要用户准许。相似地，在企业应用中，你的应用程序可以要求代理来访问一些资源。

要实现这些服务器与服务器之间的互动，你需要一个**服务账户**，该账户是属于你的应用程序的而不是最终用户个体。你的应用程序以服务账户名义来调用谷歌 API，这种情形下也不需要用户准许。（如果不使用服务账户，而让应用程序以最终用户的名义来调用谷歌 API，则在一些情况下会要求用户准许。）

> 注意: 这些服务账户方案要求应用程序自行创建 JSON Web 令牌（JWTs）并且自己进行密码学签名。我们强烈建议您使用库来完成这些工作。如果您不使用抽象化令牌创建和签名的库，自己编写这部分代码的话，则有犯错误的风险，这些错误可能会对您的应用程序的安全照成严重冲击。若想了解这种方案所需的库的列表，请参阅关于服务账户的文档。

从谷歌开发者控制台获得的一个服务账户的凭证，包括了一个自动生成的唯一的电子邮箱地址、一个客户端 ID，和至少一对公钥和私钥。使用客户端 ID 和一个私钥来创建一个经过签名的 JWT 并以正确的格式构造一个访问令牌请求。然后应用程序将令牌请求发送给谷歌 OAuth 2.0 授权服务器，并获得访问令牌。然后应用程序使用访问令牌来访问谷歌 API。当访问令牌过期，应用程序重复这个步骤来获得新令牌。



详情请参阅关于服务账户的文档。

#

令牌有效期限

您的代码应该要能预料一个颁发下来的令牌有失效的可能性。下面几种原因都能导致令牌失效:

- 用户废除了访问权。
- 令牌闲置时间超过 6 个月。
- 用户账户的令牌请求超过了限值。

目前每个谷歌用户账户都有令牌的数量限制，最多 25 个。如果一个用户拥有 25 个有效令牌，则下一个验证请求成功时，会在没有任何用户可见的警告的情况下废除最后一次使用时间最老的令牌。

如果你需要许可多个程序、机器、或设备，其中一个解决方法就是限制你许可给单个用户的客户端数量在 15 或 20 内。如果你是谷歌 Apps 管理员，你可以创建额外的管理员用户并使用他们来授权一些客户端。

#

客户端库

下面的客户端库整合了一些流行的框架，让您更简单地应用 OAuth 2.0。随着时间的推移，会有更多的新功能添加到这些库中。

- 谷歌 API 客户端库 for Java
- 谷歌 API 客户端库 for Python
- 谷歌 API 客户端库 for .NET
- 谷歌 API 客户端库 for Ruby
- 谷歌 API 客户端库 for PHP
- 谷歌 API 客户端库 for JavaScript
- 谷歌工具箱 for Mac OAuth 2.0 控制器

2

Web 服务器应用程序中使用 OAuth 2.0

谷歌 OAuth 2.0 端点支持使用下列语言和框架编写的 Web 服务器应用程序：PHP，Java，Python，Ruby，和 ASP.NET。 这些应用程序可能会在用户使用的时候或用户没在使用的时候访问谷歌 API。这种工作流需要应用程序有能力保存一个 secret。

这个文档描述了如何从 Web 服务器应用程序使用 OAuth 2.0 访问谷歌 API。

#

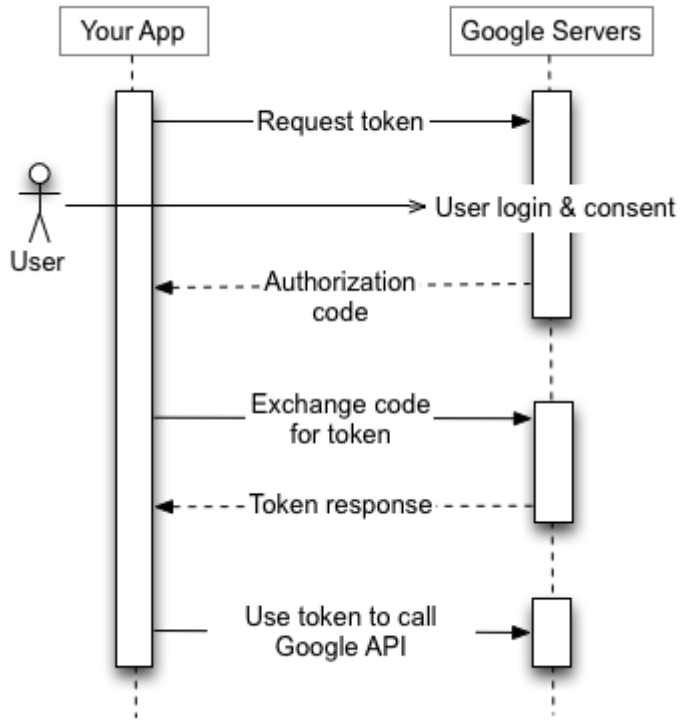
概览

授权程序会在你的应用程序将一个浏览器重定向到一个谷歌 URL；一个包含查询参数的 URL 显示出正在请求的访问类型。和其他情形一样，谷歌会处理用户认证，会话选择，用户准许（User consent）。这一程序的结果是授权码，谷歌会将授权码以查询串的形式返回给您的应用程序。

接收到授权码之后，你的应用程序可以通过密码交换（同时也会进行客户端 ID 和客户端 secret 交换）来获得访问令牌,在某种情形下还会得到一个刷新令牌。

应用程序可以使用访问令牌来访问谷歌 API。

如果一个刷新令牌在授权码交换时存在，那么它可以在任何时候用来获取新的访问令牌。这叫做离线访问，因为应用程序用这种方法取得新的访问令牌时不需要用户值守浏览器。



#

生成 URL

用于认证用户的 URL 是 `https://accounts.google.com/o/oauth2/auth` 。这个端点只能通过 SSL 访问，HTTP 连接会被拒绝。

端点：https://accounts.google.com/o/oauth2/auth
描述：这个端点是初次请求的目标。它负责处理寻找活动会话，认证用户和用户准许。当您的应用程序对该端点发送请求时，回应会包含一个访问令牌，一个刷新令牌，和一个授权码。

--

对于 web 服务器应用程序，谷歌认证服务器支持的查询串参数集为：

参数： response_type（ 响应类型 ）
值： code（ 密码 ）
描述： 决定 Google OAuth 2.0 端点是否要返回授权码。 对于 Web 服务器应用程序，参数应该使用 code。

--

参数： client_id（ 客户端 ID ）
值： 你从开发者控制台处获得的客户端 ID。
描述： 确定是哪个客户端正在发出请求。传过去的这个参数值必须要与开发者控制台里显示的完全一致。

--

参数： redirect_uri（ 重定向 URI ）
值： 在开发者控制台里列出的这个工程的 redirect_urivalues 其中一个值。
描述： 决定回应(Response)会发向哪里。这个参数的值必须和谷歌开发者控制台为这个工程所显示的值的其中一个完全一致（ 包括完整的 HTTP 或 HTTPS 格式、大小写、和末尾的'/'符号 ）。

参数： scope（域）

值： 用空格分隔该应用程序所请求的权限集。

描述： 确认您的应用程序请求的谷歌 API 访问权。传过去的参数值会以用户准许页面的方式向用户显示。请求的权限数量和获得用户准许的可能性有逆相关关系。若想了解可用的登陆域，请参见登陆域。若想了解所有谷歌 API 的可用域，请访问 API 浏览器。基本上渐进地请求域是一项最佳实践，比起提前一次性请求所有权限，按需请求更好。举例来说，一个想要支持购买功能的应用程序应该在用户点击“购买”按钮的时候才要求谷歌钱包访问权; 详情参阅渐进式授权。

参数： state（状态）

值： 任意字符串

描述： 当收到回应时提供任何可能对您的应用程序有用的状态。谷歌认证服务器会回传这个参数，所以您的应用程序会收到和它发出去一样的内容。若想防止跨站请求伪造攻击(CSRF), 我们强烈推荐您在状态中包含防伪造令牌，并且在回应中进行确认。详情请参见OpenID 连接 来获得实现这个功能的示例。

参数： access_type（访问类型）

值： online 或 offline（在线或离线）

描述： 表示当用户不在浏览器前时，您的应用程序是否需要访问谷歌 API。这个参数默认值是 online。如果您的应用程序需要在用户不在浏览器前的时候刷新访问令牌，那么请使用 offline，这样做会让您的应用程序在第一次为用户获取授权码的时候获得一个刷新令牌。

--

参数： approval_prompt（准许提示）

值： force 或 auto（强制或自动）

描述： 决定用户是否应该再次进行用户准许。默认值是 auto（自动），表示用户只需要在第一次权限请求的时候看见用户许可页面。如果这个值是 force（强制），那么用户即使以前许可了您的应用程序的相关权限，也会再次见到用户许可页面。

参数： login_hint（登陆提示）

值： 电子邮箱地址或子标识符

描述： 当您的应用程序知道哪个用户正在尝试进行认证时，可以提供这个参数作为对认证服务器的提示。将这个参数传过去会在登陆页面自动填写用户邮箱地址，或者选择适合的多用户登录会话，从而简化登陆工作流。

参数： include_granted_scopes（包含已许可的域）

值： true 或 false

描述： 如果这个参数值被设为 true，同时认证请求被许可，那么该次认证会许可所有曾经许可给这组用户和应用程序的其他域；请参见渐进式授权。

下面是一串示例 URL。（包含换行和空格来加强可读性）

```
https://accounts.google.com/o/oauth2/auth?
scope=email%20profile&
state=security_token%3D138r5719ru3e1%26url%3Dhttps://oa2cb.example.com/myHome&
redirect_uri=https%3A%2F%2Foauth2-login-demo.appspot.com%2Fcode&,
response_type=code&
client_id=812741506391.apps.googleusercontent.com&
approval_prompt=force
```

#

处理回应

回应会被发送到请求 URL 中 `redirect_uri` 所指定的目的地。如果用户准许访问请求，那么回应会包含授权码和状态参数（如果请求中有包含状态参数的话）。如果用户没有准许该次请求，那么回应会包含一个错误信息。所有回应都会发送到查询串中的 web 服务器，如下例所示：

一个包含错误的回应：

```
https://oauth2-login-demo.appspot.com/code?error=access_denied&state=security_token%3D138r5719ru3e1%26url%
```

一个授权码回应：

```
https://oauth2-login-demo.appspot.com/code?state=security_token%3D138r5719ru3e1%26url%3Dhttps://oa2cb.examp
```

> 重要信息: 如果您接收回应的端点会渲染 HTML 页面，那么页面的所有资源都可以看见 URL 中的授权码。其中脚本能直接读取 URL，页面上任何资源都可能收到带有授权码的 URL 的 Referer HTTP头。请认真考虑您是否真的想发送认证凭证给那个页面上的所有资源（特别是第三方脚本，例如社交网络插件和统计分析插件）。若想避免这个问题，我们推荐让服务器先单独处理请求，然后再重定向到另外一个不包含回应参数的 URL。

服务器收到授权码后，它就能用授权码来交换一个访问令牌和一个刷新令牌。这种请求实际上是将一个 HTTPS `POST` 发送到 URL： `https://www.googleapis.com/oauth2/v3/token` ，其中包含以下参数：

字段： code（密码）
描述： 从初次请求获得的授权码

--

字段： client_id（客户端 ID）
描述： 从开发者控制台处获得的客户端 ID。

--

字段： client_secret（客户端 secret）
描述： 从开发者控制台处获得的客户端 secret。

--

字段： redirect_uri（重定向 URI）
描述： 这个工程在开发者控制台处列出的重定向 URI 项之一。

--

字段： grant_type（许可类型）

描述： 正如 OAuth 2.0 规格中定义的，这个字段必须包含 authorization_code 中的一个值。

--

实际请求可能会像下面这样：

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7&
client_id=8819981768.apps.googleusercontent.com&
client_secret={client_secret}&
redirect_uri=https://oauth2-login-demo.appspot.com/code&
grant_type=authorization_code
```

一个针对这个请求的成功的回应应该包含以下字段：

字段： access_token（访问令牌）

描述： 可以发送给谷歌 API 的令牌。

--

字段： refresh_token（刷新令牌）

描述： 用于获得新的访问令牌的令牌。刷新令牌会一直有效直到用户对其废除访问权。该字段只有在授权码请求中的 access_type = offline 的时候才可能出现。

--

字段： expires_in（有效期）

描述： 访问令牌剩余的生命期。

--

字段： token_type（令牌类型）

描述： 确认返回的令牌类型。在这次示例中，这个字段总是会拥有值 Bearer。

--

一个成功的回应会以 JSON 数组的形式发送回来，和下面的例子类似：

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer"
}
```

> 注意: 有时回应可能会包含其他字段，您的程序不应该将这种情况视为错误。上面示例中的集合是最小集。

#

调用谷歌 API

您的程序获得访问令牌之后，您可以使用令牌以用户或者服务账户的名义来对谷歌 API 进行调用。要做到这一点，请将访问令牌包含到发给 API 的请求中，可以通过包含 `access_token` 查询参数或者一个 `Authorization: Bearer` HTTP 头来实现。如果可能的话，我们更欢迎 HTTP 头的方法，因为查询串更容易在服务器记录中可见。在大多数情况下你可以使用客户端库来设置您的谷歌 API 调用（例如，当对谷歌人脉 API 进行调用时）。

你可以在OAuth 2.0 游乐场中自行尝试所有的谷歌 API 并查看他们所对应的域。

#

示例

一个通过使用 `access_token` 查询串参数对 people.get 端点（谷歌人脉 API）的调用会和下例相类似，当然在实际情况中您需要提供您自己的访问令牌：

```
GET https://www.googleapis.com/plus/v1/people/userId?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

而对于已经认证的用户（me）而言，通过 `Authorization: Bearer` HTTP 头调用同样的 API 就会像下面这样：

```
GET /plus/v1/people/me HTTP/1.1
Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg
Host: googleapis.com
```

您可以尝试 `curl` 命令行应用程序。下面是使用 HTTP 头的方法（推荐）的示例：

```
curl -H "Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg" https://www.googleapis.com/plus/v1/people/me
```

或者，您也可以使用查询串参数的方法实现：

```
curl https://www.googleapis.com/plus/v1/people/me?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

#

渐进式授权

在 OAuth 协议中，您的应用程序为了访问资源所请求的认证是以域来区分的，如果用户已经被认证而且同意权限请求，您的应用程序会接收到一个生命期比较短的访问令牌，这些令牌能让它访问目标资源，而刷新令牌（可选）可以让应用程序拥有长期的访问权限。

按需请求资源访问权限通常被认为是一项最佳实践。举例来说，一个让人们可以对音乐进行采样并且创建混音的应用程序也许在登陆时候只需要非常少量的资源，说不定除了登陆者的名字之外就没别的了。但是，要保存一个完整的混音可能会需要访问他们的谷歌网盘。大多数人都会觉得在应用程序需要存储文件时申请谷歌网盘的访问权限是非常自然的。

这种情况下，在登录时应用程序可以请求这个域：`https://www.googleapis.com/auth/plus.loginto` 来实现一个基本的社交登陆功能，然后在需要保存混音的时候才申请这个域：`https://www.googleapis.com/auth/drive.file` 。

同时使用在使用 OpenID 连接和使用 OAuth 2.0 来访问谷歌 API 里描述的步骤通常会让你的应用程序不得不管理两个不同的访问令牌。如果您希望避开这种复杂性，则需要在所有 OAuth 2.0 工作流的第一步时，将发送给 `https://accounts.google.com/o/oauth2/auth` 的认证 URI 里添加一个额外的参数。这个参数是 `include_granted_scopes` ，其中值可以被赋为 `true` 或 `false` （默认值是 `false` ），当这个值为 `true` 时，如果您的域认证要求被批准，谷歌认证服务器会将这次认证和所有之前已经成功的认证为这组用户-应用程序合并。这类请求的 URI 可能看上去会像下面这样（下例有插入换行和空格来增强可读性）：

```
https://accounts.google.com/o/oauth2/auth?
  scope=https://www.googleapis.com/auth/drive.file&
  state=security_token%3D138r5719ru3e1%26url%3Dhttps://oa2cb.example.com/myHome&
  redirect_uri=https%3A%2F%2Fmyapp.example.com%2Fcallback&
  response_type=code&
  client_id=8127352506391.apps.googleusercontent.com&
  approval_prompt=force&
  include_granted_scopes=true
```

我们现在把上面这种认证叫做“组合认证”；关于组合认证：

- 您可以使用刚才得到的访问令牌来访问任何在组合认证中已经合并的域。
- 当您使用刷新令牌来进行组合认证时，返回的新访问令牌也代表了组合认证，所以也可以用于访问其任意域。

- 组合认证包括了所有过去已经被许可的权限，即使这些权限请求曾经是从不同的客户端发出的。举例来说，如果您在桌面应用程序请求了 `https://www.googleapis.com/auth/plus.loginscope` ，然后又向同一个用户的移动端发送了同样的请求，那么后面这个请求会被自动批准，因为组合认证会包含两边的域。
- 当您废除了一个代表组合认证的的令牌，其所有的认证都会被同时废除；这意味着如果你还保留着任何以往的权限的令牌，他们也会跟着失效。

#

离线访问

在某些情况下，你的应用程序可能需要在用户不在的时候访问谷歌 API。例如备份服务或者一些可以让博客的帖子可以在周一早上准时8点钟发布的应用程序。 这种类型的访问被称作离线的，而 Web 服务器应用程序可能会向用户要求离线访问权。反之，正常和默认的情况下访问被称作在线的。

如果您的应用程序需要离线访问谷歌 API，那么在授权码的请求中就应该包含 `access_type` 参数，并将其值设定为 `offline` 。离线访问请求的例子在下方，含有换行和空格来增强可读性：

```
https://accounts.google.com/o/oauth2/auth?
 scope=email%20profile&
 state=security_token%3D138r5719ru3e1%26url%3Dhttps://oa2cb.example.com/myHome&
 redirect_uri=https%3A%2F%2Foauth2-login-demo.appspot.com%2Fcode&
 response_type=code&
 client_id=812741506391.apps.googleusercontent.com&
 access_type=offline
```

当目标用户的浏览器第一次被指向这串 URL 时，他们会看见用户准许页面。如果他们批准访问，那么回应中就会包含一个授权码，授权码可以用来交换获得一个访问令牌和一个刷新令牌。

下面是授权码交换的示例：

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7&
client_id=8819981768.apps.googleusercontent.com&
client_secret={client_secret}&
redirect_uri=https://oauth2-login-demo.appspot.com/code&
grant_type=authorization_code
```

如果这是应用程序第一次为用户进行授权码交换，那么回应中就会包含一个访问令牌和一个刷新令牌，如下例所示：

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
```

```
  "refresh_token":"1/xEoDL4iW3cxlI7yDbSRFYNG01kVKM2C-259HOF2aQbI"
}
```

> 重要：当您的应用程序获得一个刷新令牌时，将刷新令牌保存起来供未来使用时很重要的。因为一旦您的应用程序丢失了刷新令牌，它就只能重新向用户进行用户准许才能获得另一个刷新令牌了。如果你需要重新向用户进行用户准许，请在授权码请求里面包含 `approval_prompt` 参数，并将其值设定为 `force` 。

您的应用程序接收刷新令牌之后，就可以在任意时间获得新的访问令牌了。请参见刷新令牌的相关章节 来获得更多信息。

下一次您的应用程序为同一个用户请求授权码时，用户不会再一次被要求同意准许了（假设他们以前已经同意了这次请求所包含的的域）。和预料的一样，回应会包含一个用于交换的授权码。不过，和第一次为用户交换授权码不一样，返回的内容中不会包括刷新令牌。下面是这类请求的一个示例；

```
{
  "access_token":"1/fFAGRNJru1FQd77BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
}
```

#

使用刷新令牌

和之前章节提到的一样，一个刷新令牌是在使用参数 `access_type` 为 `offline` 的第一次授权码交换中获得的。在这类情况下，您的应用程序可以通过发送刷新令牌到谷歌 OAuth 2.0 认证服务器来获得一个新的访问令牌。

若要通过这种方式获取访问令牌，您的应用程序应该发送一个 HTTPS `POST` 请求到 `https://www.googleapis.com/oauth2/v3/token` 。并且请求必须包含以下参数：

字段： refresh_token（刷新令牌）
描述： 从授权码交换返回的刷新令牌

--

字段： client_id（客户端 ID）
描述： 从开发者控制台处获得的客户端 ID。

--

字段： client_secret（客户端 secret）

描述： 从开发者控制台处获得的客户端 secret。

--

字段： grant_type（许可类型）

描述： 正如 OAuth 2.0 规格中定义的，这个字段必须包含 authorization_code 中的一个值。

--

这种请求如下例所示：

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

client_id=8819981768.apps.googleusercontent.com&
client_secret={client_secret}&
refresh_token=1/6BMfW9j53gdGImsiyUH5kU5RsR4zwI9lUVX-tqf8JXQ&
grant_type=refresh_token
```

只要用户没有废除您的应用程序的访问权限，回应中就会包括一个新的访问令牌。对于上述请求，回应应该和下例相似：

```
{
  "access_token":"1/fFBGRNJru1FQd44AzqT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
}
```

请注意刷新令牌的发放数量是有限制的；限制的单位是每一组客户端-用户，你应该在长期存储器中保存刷新令牌并一直使用它直到过期。如果您的应用程序请求太多刷新令牌，就可能会触发限制，这种情况下最旧的令牌会失效。

#

废除令牌

在某些情况下，一个用户也许会想要废除已经授予给应用程序的权限。用户可以用过访问以下 URL 来显式地废除权限 https://accounts.google.com/b/0/IssuedAuthSubTokens。 同时，通过编程让应用程序自行废除已经获得的权限也是可能的。程序化废除工作在用户取消订阅或者卸载程序时是很重要的。换句话来说，卸载的过程可以包含一个 API 请求，用于确保废除被卸载的应用程序已经获得的权限。

若想程序化废除令牌，您的应用程序需要向 `https://accounts.google.com/o/oauth2/revoke` 发送一个请求，并且将令牌作为参数发送出去：

```
curl https://accounts.google.com/o/oauth2/revoke?token={令牌}
```

其中的“令牌”可以是访问令牌，也可以是刷新令牌。如果令牌是一个访问令牌，并且有成对的刷新令牌，那么所对应的刷新令牌也会被废除。

如果废除工作被成功执行，那么回应的状态码会是 `200` 。如果发生错误，回应的状态码会是 `400` ，并且回应会包含一个错误码。

注意： 在接收到废除工作成功的回应之后，可能需要隔一段时间废除才会完全生效。

#

客户端库

下面的客户端库整合了一些流行的框架，让您更简单地应用 OAuth 2.0。随着时间的推移，会有更多的新功能添加到这些库中。

- 谷歌 API 客户端库 for Java
- 谷歌 API 客户端库 for Python
- 谷歌 API 客户端库 for .NET
- 谷歌 API 客户端库 for Ruby
- 谷歌 API 客户端库 for PHP
- 谷歌 API 客户端库 for JavaScript
- 谷歌工具箱 for Mac OAuth 2.0 控制器

3

本地安装的应用程序中使用 OAuth 2.0

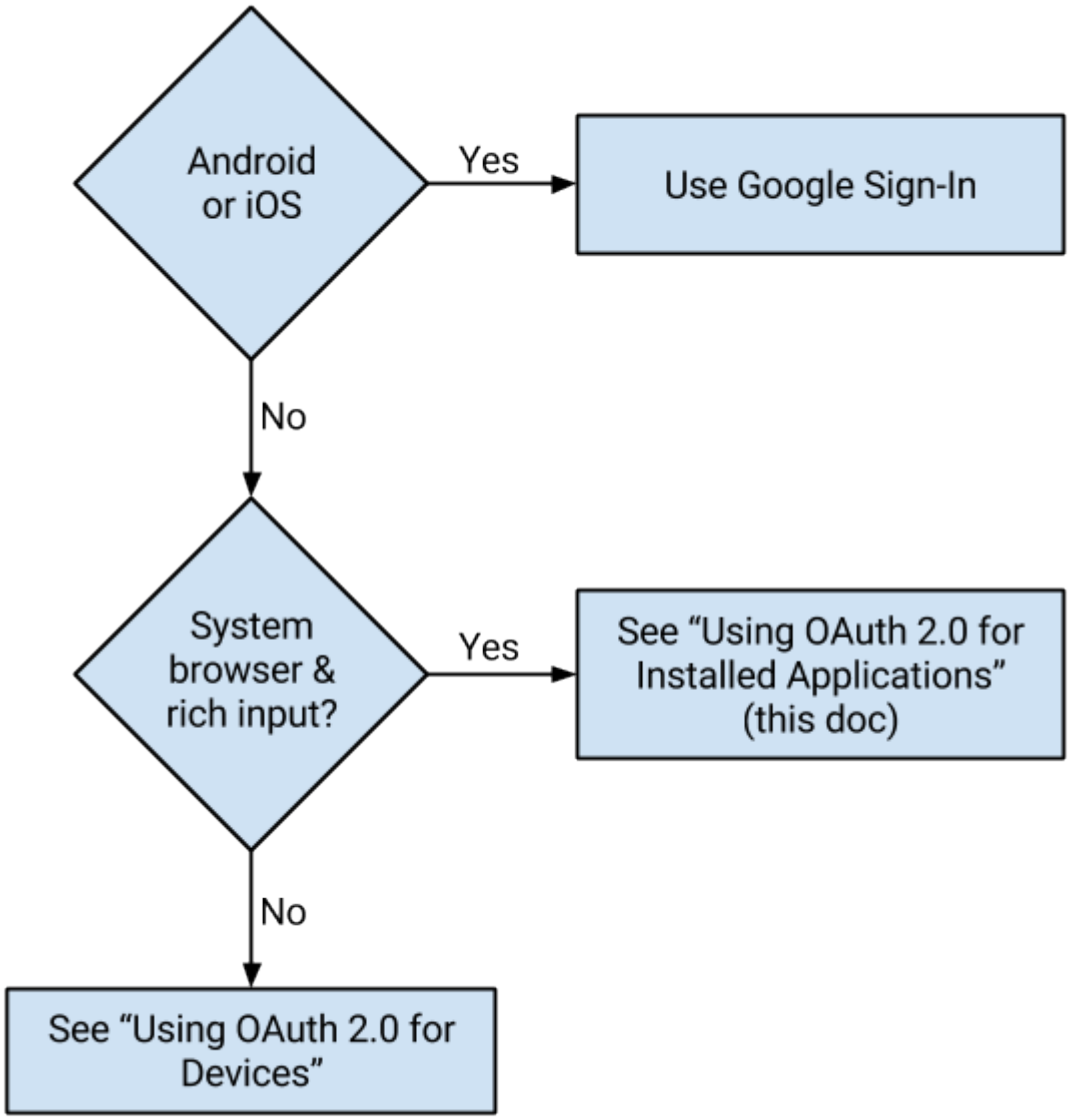
谷歌 OAuth 2.0 端点支持各种平台上的应用程序，例如计算机、移动电话、平板等。本地安装的应用程序会被分发到每个独立的机器上，而且通常被认为不能用于保存 secret。这些应用程序可以在用户值守时，或者在后台运行时访问谷歌 API。

如果您满足下述条件，那么这个文档就是为您准备的：

- 您正在编写一个本地安装的应用程序，且该程序的平台不是 Android 或 iOS，同时
- 您的本地安装的应用程序会运行在拥有良好输入性能并且系统具有浏览器的设备上，例如拥有全键盘的设备。

如果你正在 Android 或 iOS 编写应用程序，请使用谷歌登陆来认证您的用户。谷歌登陆按钮会管理 OAuth 2.0 工作流，包括认证和获取谷歌 API 授权。所有有谷歌账户的人都能用谷歌登陆，不论他们有没有升级到 Google+。若要添加谷歌登陆按钮，请根据平台分别参照 Android 或 iOS 或网站。

如果您的应用程序运行在没有浏览器的设备上，或者输入性能受限（例如运行在家用游戏机、视频摄录机、打印机上的应用程序），那么请参照在设备上使用 OAuth 2.0。

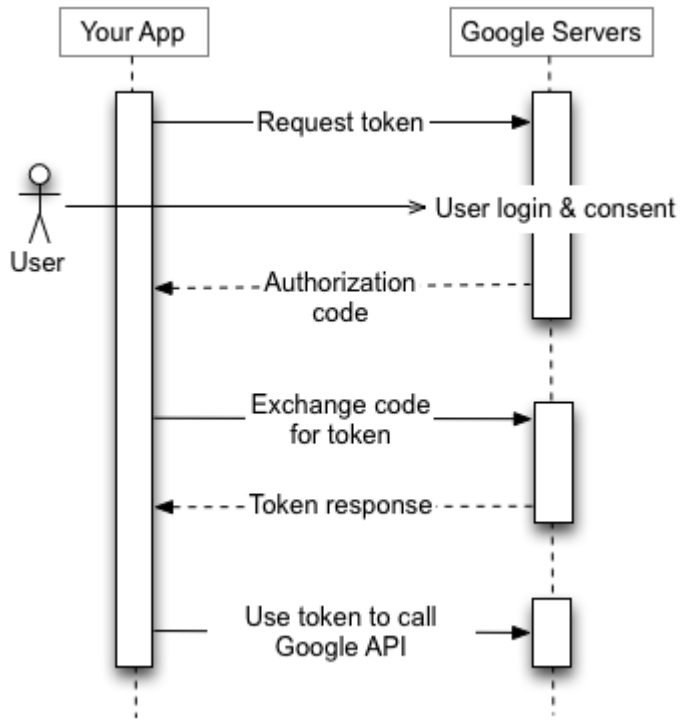

Android or iOS
Yes
Use Google Sign-In
No
System browser & rich input?
Yes
See “Using OAuth 2.0 for Installed Applications” (this doc)
No
See “Using OAuth 2.0 for Devices”

#

概览

在这个工作流中，在本地安装的应用程序必须能使用浏览器，或者有嵌入式浏览器作为 Web 视图。在本地安装的应用程序上的 OAuth 2.0 工作流如下：

1. 您的应用程序将浏览器重定向到一个谷歌 URL。URL 查询参数会指出应用程序所需要的谷歌 API 访问权限。
2. 和其他方案一样，谷歌会负责用户认证和用户准许（User consent），这一系列程序的结果是一个授权码。授权码会返回到浏览器的标题栏上或者作为一个查询串参数被返回，这取决于您的应用程序在请求中所发送的参数。
3. 您的应用程序会进行授权码交换来获得访问令牌和刷新令牌。在交换过程中，应用程序会提交他的客户端 ID 和客户端 secret（这两者都是从开发者控制台获得的）。
4. 您的应用程序会使用访问令牌来对谷歌 API 进行调用，应用程序也会储存刷新令牌以供未来使用。



这个工作流和在 Web 服务器应用程序上使用 OAuth 2.0的工作流比较类似，但有以下三点不同：

- 当创建一个客户端 ID时，你需要指明您的应用程序是一个在本地安装的应用程序。这会影响 redirect_uri 参数的值的结果。
- 从开发者控制台获取的客户端 ID 和客户端 secret 是嵌入在您的应用程序源代码内的。在这种情况下，客户端 secret 显然没被当做秘密看待。
- 授权码可以通过浏览器的标题栏，或者 http://localhost 端口的查询串来返回到您的应用程序。

#

为认证请求生成 URL

用于认证用户的 URL 是 https://accounts.google.com/o/oauth2/auth 。这个端点只能通过 SSL 访问，HTTP（非-SSL）连接会被拒绝。

--

端点：https://accounts.google.com/o/oauth2/auth
描述：这个端点是初次请求的目标。它负责处理寻找活动会话，认证用户和用户准许。当您的应用程序对该端点发送请求时，回应会包含一个访问令牌，一个刷新令牌，和一个授权码。

--

对于在本地安装的应用程序，谷歌认证服务器支持的查询串参数集为：

参数： response_type（响应类型）
值： code（密码）
描述： 决定 Google OAuth 2.0 端点是否要返回授权码。 对于在本地安装的应用程序，参数应该使用 code。

--

参数： client_id（客户端 ID）
值： 你从开发者控制台处获得的客户端 ID。
描述： 确定是哪个客户端正在发出请求。传过去的这个参数值必须要与开发者控制台里显示的完全一致。

--

参数： redirect_uri（重定向 URI）
值： 在开发者控制台里列出的这个工程的 redirect_urivalues 其中一个值。
描述： 决定回应(Response)会发向哪里。这个参数的值必须和谷歌开发者控制台为这个工程所显示的值的其中一个完全一致（包括完整的 HTTP 或 HTTPS 格式、大小写、和末尾的'/'符号）。您能使用 urn:ietf:wg:oauth:2.0:oob, urn:ietf:wg:oauth:2.0:oob:auto, 或者一个 http://localhost 端口。更多详情，请参照选择一个重定向 URI。

--

参数： scope（域）
值： 用空格分隔的该应用程序所请求的权限集。

描述： 确认您的应用程序请求的谷歌 API 访问权。传过去的参数值会以用户准许页面的方式向用户显示。请求的权限数量和获得用户准许的可能性有逆相关关系。若想了解可用的登陆域，请参见登陆域。若想了解所有谷歌 API 的可用域，请访问 API 浏览器。基本上渐进地请求域是一项最佳实践，比起提前一次性请求所有权限，按需请求更好。举例来说，一个想要支持购买功能的应用程序应该在用户点击“购买”按钮的时候才要求谷歌钱包访问权; 详情参阅渐进式授权。

--

参数： state（状态）
值： 任意字符串
描述： 当收到回应时提供任何可能对您的应用程序有用的状态。谷歌认证服务器会回传这个参数，所以您的应用程序会收到和它发出去一样的内容。若想防止跨站请求伪造攻击(CSRF), 我们强烈推荐您在状态中包含防伪造令牌，并且在回应中进行确认。详情请参见OpenID 连接 来获得实现这个功能的示例。

--

参数： login_hint（登陆提示）
值： 电子邮箱地址 或 子标识符
描述： 当您的应用程序知道哪个用户正在尝试进行认证时，可以提供这个参数作为对认证服务器的提示。将这个参数传过去会在登陆页面自动填写用户邮箱地址，或者选择适合的多用户登录会话，从而简化登陆工作流。

参数： include_granted_scopes（包含已许可的域）

值： true 或 false

描述： 如果这个参数值被设为 true，同时认证请求被许可，那么该次认证会许可所有 曾经许可给这组用户和应用程序的其他域；请参见渐进式授权。

--

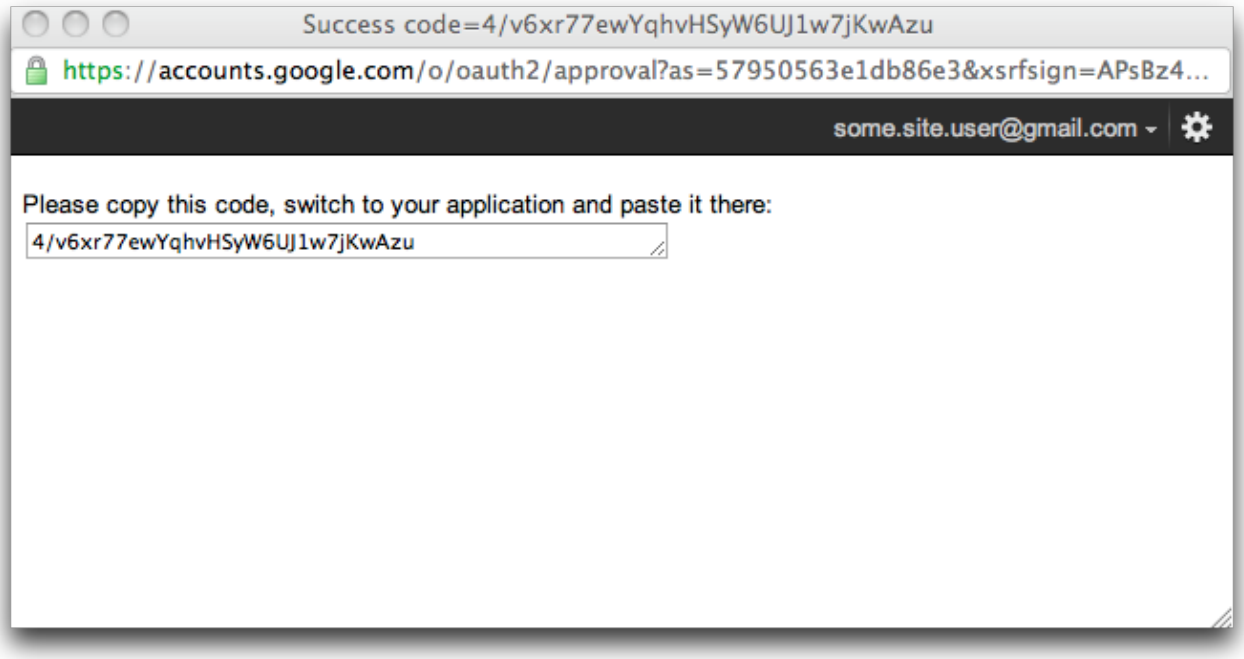
#

示例 URL

下面是一个示例 URL，包含换行和空格来加强可读性：

```
https://accounts.google.com/o/oauth2/auth?
  scope=email%20profile&
  redirect_uri=urn:ietf:wg:oauth:2.0:oob&
  response_type=code&
  client_id=812741506391-h38jh0j4fv0ce1krdkiq0hfvt6n5amrf.apps.googleusercontent.com
```

如果用户通过类似上述的 URL 进行登陆并且批准了访问权限，那么其结果会是类似下面这样的对话框：



下面是另外一个示例 URL，包含换行和空格来加强可读性：

```
https://accounts.google.com/o/oauth2/auth?
 scope=email%20profile&
 redirect_uri=http://localhost:9004&
 response_type=code&
 client_id=812741506391-h38jh0j4fv0ce1krdkiq0hfvt6n5amrf.apps.googleusercontent.com
```

这两个 URL 的区别只有 `redirect_uri` 参数的值。第一个会让回应通过页面的标题返回授权码，而第二个则会使授权码作为查询串的一部分返回到 `http://localhost` 地址。

#

选择一个重定向 URI

当您在谷歌开发者控制台中创建一个客户端 ID 时，控制台会为您创建两个 redirect_uri（重定向 URI）：

urn:ietf:wg:oauth:2.0:oob 和 http://localhost

您的应用程序所使用的参数值会决定授权码以何种方式传回。

#

http://localhost

这个参数值会告知谷歌认证服务器授权码应该以查询串参数的形式返回到客户端的 Web 服务器上。您可以在不需要变更谷歌开发者控制台配置的情况下指定端口号。如果要使用这个 URL 接收授权码，您的应用程序必须监听本地 Web 服务器。这种方案在许多平台上可行，但不是所有平台都可行。如果您的平台支持这种方式，这是推荐的获取授权码的方式。

> 注意：在一些情况下，尽管可以进行监听，但其他软件（例如 Windows 防火墙）如果没有作出显著配置可能会对通讯照成影响。

#

urn:ietf:wg:oauth:2.0:oob

这个参数值会告知谷歌认证服务器应该用浏览器的标题栏将授权码返回到客户端，并且还带有一个文本页面提示用户将密码复制并粘贴到应用程序中（就像上面的截图一样）。这在当客户端（例如一个 Windows 应用程序）在系统不作出显著配置的前提下无法监听 HTTP 端口时十分有用。

当您使用这个参数值，您的应用程序可以检测到页面已经载入，并且可以读取 HTML 页面的标题来获得授权码。接下来就要靠您的应用程序来关闭浏览器窗口了，如果您想确保用户永远都不会看见包含授权码的页面的话，其实现方式根据平台的不同而不同。

如果您的目标平台不允许您检测页面是否已经载入，或者不允许您检测页面的标题，那么您可以要求用户手动将密码复制粘贴到您的应用程序内，就和页面本身的文字说明一样。

#

urn:ietf:wg:oauth:2.0:oob:auto

该参数和 `urn:ietf:wg:oauth:2.0:oob` 是一致的，但返回的确认页面不会有文字提示用户手动复制授权码，取而代之的是要求用户关闭浏览器窗口。

在您的应用程序能读取 HTML 标题（例如，通过在桌面上检查窗口标题）获得授权码，但是却没法自动关闭页面的时候，这种方法就显得有用。

#

处理回应并发出令牌请求

初次认证请求的回应会包含一个授权码（密码），您的应用程序可以用授权码交换获得一个访问令牌和一个刷新令牌。

若要发出这种令牌请求，您的应用程序应发送一个 HTTP `POST` 请求到 `/oauth2/v3/token` 。并且包含以下参数：

字段： code
描述： 初次认证请求返回的授权码

--

字段： client_id（客户端 ID）
描述： 从开发者控制台处获得的客户端 ID。

--

字段： client_secret（客户端 secret）
描述： 从开发者控制台处获得的客户端 secret。（对于注册为 Android，iOS 或 Chrome 应用程序的客户端而言，这是可选项。）

--

字段： redirect_uri（重定向 URI）
描述： 从开发者控制台处获得的重定向 URI。

--

字段： grant_type（许可类型）
描述： 正如 OAuth 2.0 规格中定义的，这个字段必须包含 authorization_code 中的一个值。

--

一个实际的请求应该类似下例：

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded
```

```
code=4/v6xr77ewYqhvHSyW6UJ1w7jKwAzu&
client_id=8819981768.apps.googleusercontent.com&
client_secret=

your_client_secret&
redirect_uri=https://oauth2-login-demo.appspot.com/code&
grant_type=authorization_code
```

针对上述请求的成功回应会包含下列字段：

字段： access_token（访问令牌）

描述： 可以发送给谷歌 API 的令牌。

--

字段： refresh_token（刷新令牌）

描述： 用于获得新的访问令牌的令牌，对于本地安装的应用程序，该令牌默认已经包含，刷新令牌会一直有效直到用户对其废除访问权。

--

字段： expires_in（有效期）

描述： 访问令牌剩余的生命期。

--

字段： token_type（令牌类型）

描述： 确认返回的令牌类型。目前，这个字段总是会拥有值 Bearer。

--

> 注意: 有时回应可能会包含其他字段，您的程序不应该将这种情况视为错误。上面示例中的集合是最小集。

一个成功的回应会以 JSON 数组的形式发送回来，和下面的例子类似：

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
  "refresh_token":"1/xEoDL4iW3cxlI7yDbSRFYNG01kVKM2C-259HOF2aQbI"
}
```

#

调用谷歌 API

您的程序获得访问令牌之后，您可以使用令牌以用户或者服务账户的名义来对谷歌 API 进行调用。要做到这一点，请将访问令牌包含到发给 API 的请求中，可以通过包含 `access_token` 查询参数或者一个 `Authorization: Bearer` HTTP 头来实现。如果可能的话，我们更欢迎 HTTP 头的方法，因为查询串更容易在服务器记录中可见。在大多数情况下你可以使用客户端库来设置您的谷歌 API 调用（例如，当对谷歌人脉 API 进行调用时）。

你可以在OAuth 2.0 游乐场中自行尝试所有的谷歌 API 并查看他们所对应的域。

#

示例

一个通过使用 access_token 查询串参数对 people.get 端点（谷歌人脉 API）的调用会和下例相类似，当然在实际情况中您需要提供您自己的访问令牌：

```
GET https://www.googleapis.com/plus/v1/people/userId?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

而对于已经认证的用户（me）而言，通过 `Authorization: Bearer` HTTP 头调用同样的 API 就会像下面这样：

```
GET /plus/v1/people/me HTTP/1.1
Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg
Host: googleapis.com
```

您可以尝试 `curl` 命令行应用程序。下面是使用 HTTP 头的方法（推荐）的示例：

```
curl -H "Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg" https://www.googleapis.com/plus/v1/people/me
```

或者，您也可以使用查询串参数的方法实现：

```
curl https://www.googleapis.com/plus/v1/people/me?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

#

渐进式授权

在 OAuth 协议中，您的应用程序为了访问资源所请求的认证是以域来区分的，如果用户已经被认证而且同意权限请求，您的应用程序会接收到一个生命期比较短的访问令牌，这些令牌能让它访问目标资源，而刷新令牌（可选）可以让应用程序拥有长期的访问权限。

按需请求资源访问权限通常被认为是一项最佳实践。举例来说，一个让人们可以对音乐进行采样并且创建混音的应用程序也许在登陆时候只需要非常少量的资源，说不定除了登陆者的名字之外就没别的了。但是，要保存一个完整的混音可能会需要访问他们的谷歌网盘。大多数人都会觉得在应用程序需要存储文件时申请谷歌网盘的访问权限是非常自然的。

这种情况下，在登录时应用程序可以请求这个域：`https://www.googleapis.com/auth/plus.loginto` 来实现一个基本的社交登陆功能，然后在需要保存混音的时候才申请这个域：`https://www.googleapis.com/auth/drive.file` 。

同时使用在使用 OpenID 连接和使用 OAuth 2.0 来访问谷歌 API 里描述的步骤通常会让你的应用程序不得不管理两个不同的访问令牌。如果您希望避开这种复杂性，则需要在所有 OAuth 2.0 工作流的第一步时，将发送给 `https://accounts.google.com/o/oauth2/auth` 的认证 URI 里添加一个额外的参数。这个参数是 `include_granted_scopes` ，其中值可以被赋为 `true` 或 `false` （默认值是 `false` ），当这个值为 `true` 时，如果您的域认证要求被批准，谷歌认证服务器会将这次认证和所有之前已经成功的认证为这组用户-应用程序合并。这类请求的 URI 可能看上去会像下面这样（下例有插入换行和空格来增强可读性）：

```
https://accounts.google.com/o/oauth2/auth?
  scope=https://www.googleapis.com/auth/drive.file&
  state=security_token%3D138r5719ru3e1%26url%3Dhttps://oa2cb.example.com/myHome&
  redirect_uri=https%3A%2F%2Fmyapp.example.com%2Fcallback&
  response_type=code&
  client_id=8127352506391.apps.googleusercontent.com&
  approval_prompt=force&
  include_granted_scopes=true
```

我们现在把上面这种认证叫做“组合认证”；关于组合认证：

- 您可以使用刚才得到的访问令牌来访问任何在组合认证中已经合并的域。
- 当您使用刷新令牌来进行组合认证时，返回的新访问令牌也代表了组合认证，所以也可以用于访问其任意域。

- 组合认证包括了所有过去已经被许可的权限，即使这些权限请求曾经是从不同的客户端发出的。举例来说，如果您在桌面应用程序请求了 `https://www.googleapis.com/auth/plus.loginscope` ，然后又向同一个用户的移动端发送了同样的请求，那么后面这个请求会被自动批准，因为组合认证会包含两边的域。
- 当您废除了一个代表组合认证的的令牌，其所有的认证都会被同时废除；这意味着如果你还保留着任何以往的权限的令牌，他们也会跟着失效。

#

使用刷新令牌

若要获取访问令牌，您的应用程序应该发送一个 HTTPS `POST` 请求到 `https://www.googleapis.com/oauth2/v3/token` 。并且请求必须包含以下参数：

字段： refresh_token（刷新令牌）
描述： 从授权码交换返回的刷新令牌

字段： client_id（客户端 ID）

描述： 从开发者控制台处获得的客户端 ID。

--

字段： client_secret（客户端 secret）

描述： 从开发者控制台处获得的客户端 secret。

--

字段： grant_type（许可类型）

描述： 正如 OAuth 2.0 规格中定义的，这个字段必须包含 authorization_code 中的一个值。

--

这种请求如下例所示：

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

client_id=8819981768.apps.googleusercontent.com&
client_secret={client_secret}&
refresh_token=1/6BMfW9j53gdGImsiyUH5kU5RsR4zwI9lUVX-tqf8JXQ&
grant_type=refresh_token
```

只要用户没有废除您的应用程序的访问权限，回应中就会包括一个新的访问令牌。对于上述请求，回应应该和下例相似：

```
{
  "access_token":"1/fFBGRNJru1FQd44AzqT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
}
```

请注意刷新令牌的发放数量是有限制的；限制的单位是每一组客户端-用户，你应该在长期存储器中保存刷新令牌并一直使用它直到过期。如果您的应用程序请求太多刷新令牌，就可能会触发限制，这种情况下最旧的令牌会失效。

#

客户端库

下面的客户端库整合了一些流行的框架，让您更简单地应用 OAuth 2.0。随着时间的推移，会有更多的新功能添加到这些库中。

- 谷歌 API 客户端库 for Java
- 谷歌 API 客户端库 for Python
- 谷歌 API 客户端库 for .NET
- 谷歌 API 客户端库 for Ruby
- 谷歌 API 客户端库 for PHP
- 谷歌 API 客户端库 for JavaScript
- 谷歌工具箱 for Mac OAuth 2.0 控制器

4

客户端应用程序中使用 OAuth 2.0

谷歌 OAuth 2.0 端点支持以 JavaScript 为中心的应用程序。这些应用程序可以在用户值守时访问谷歌 API，并且这类应用程序不能保存 secret。

该文章描述如何通过 OAuth 2.0 让 JavaScript 应用程序访问谷歌 API。

#

概览

这种方案下，程序会将浏览器（完整页面或者弹窗）重定向至一个谷歌 URL 并附带一系列查询参数来告知该应用程序所请求的谷歌 API 权限。和其他方案一样，谷歌会负责用户认证和用户准许（User consent），这一系列程序的结果是一个访问令牌。谷歌会将该访问令牌作为回应的一部分发回，然后客户端类型的脚本应用程序将访问令牌从回应中提取出来。

应用程序在接收到访问令牌之后就可以访问谷歌 API 了。

注意: 在这种方案下，您的应用程序应该总是使用 HTTPS 连接。

#

生成 URL

用于认证用户的 URL 是 `https://accounts.google.com/o/oauth2/auth` 。这个端点只能通过 SSL 访问，HTTP（非-SSL）连接会被拒绝。

端点：https://accounts.google.com/o/oauth2/auth
描述：这个端点是初次请求的目标。它负责处理寻找活动会话，认证用户和用户准许。

--

对于客户端类型应用程序而言，谷歌认证服务器支持的查询串参数集为：

参数： response_type（响应类型）
值： token（令牌）
描述： 该值会告知谷歌认证服务器在回应片段中返回访问令牌。

--

参数： client_id（客户端 ID）
值： 你从开发者控制台处获得的客户端 ID。
描述： 确定是哪个客户端正在发出请求。传过去的这个参数值必须要与开发者控制台里显示的完全一致。

--

参数： redirect_uri（重定向 URI）
值： 在开发者控制台里列出的这个工程的 redirect_urivalues 其中一个值。
描述： 决定回应(Response)会发向哪里。这个参数的值必须和谷歌开发者控制台为这个工程所显示的值的其中一个完全一致（包括完整的 HTTP 或 HTTPS 格式、大小写、和末尾的'/'符号）。

参数： scope（域）

值： 用空格分隔的该应用程序所请求的权限集。

描述： 确认您的应用程序请求的谷歌 API 访问权。传过去的参数值会以用户准许页面的方式向用户显示。请求的权限数量和获得用户准许的可能性有逆相关关系。若想了解可用的登陆域，请参见登陆域。若想了解所有谷歌 API 的可用域，请访问 API 浏览器。基本上渐进地请求域是一项最佳实践，比起提前一次性请求所有权限，按需请求更好。举例来说，一个想要支持购买功能的应用程序应该在用户点击“购买”按钮的时候才要求谷歌钱包访问权; 详情参阅渐进式授权。

参数： state（状态）

值： 任意字符串

描述： 当收到回应时提供任何可能对您的应用程序有用的状态。谷歌认证服务器会回传这个参数，所以您的应用程序会收到和它发出去一样的内容。可以用于将用户重定向到正确的站点上的资源，Nonce(一次性加密串)，用于防止跨站请求伪造攻击的防伪造令牌。

参数： approval_prompt（准许提示）

值： force 或 auto **描述：** 决定用户是否应该再次进行用户准许。默认值是 auto（自动），表示用户只需要在第一次权限请求的时候看见用户许可页面。如果这个值是 force（强制），那么用户即使以前许可了您的应用程序的相关权限，也会再次见到用户许可页面。

参数： login_hint（登陆提示）

值： 电子邮箱地址 或 子标识符

描述： 当您的应用程序知道哪个用户正在尝试进行认证时，可以提供这个参数作为对认证服务器的提示。将这个参数传过去会在登陆页面自动填写用户邮箱地址，或者选择适合的多用户登录会话，从而简化登陆工作流。

参数： include_granted_scopes（包含已许可的域）

值： true 或 false

描述： 如果这个参数值被设为 true，同时认证请求被许可，那么该次认证会许可所有 曾经许可给这组用户和应用程序的其他域；请参见渐进式授权。

下面是一个示例 URL，加入了换行符和空格来增加可读性。

```
https://accounts.google.com/o/oauth2/auth?
 scope=email%20profile&
 state=%2Fprofile&
 redirect_uri=https%3A%2F%2Foauth2-login-demo.appspot.com%2Foauthcallback&
 response_type=token&
 client_id=812741506391.apps.googleusercontent.com
```

#

处理回应

如果用户准许您的应用程序权限请求，谷歌会回应一个访问令牌给您的应用程序。访问令牌在返回的片段中的 `access_token` 参数中。由于片段不会整个返回给服务器，所以客户端类型脚本程序必须分析片段并且从 `access_token` 参数中提取值。在回应中还存在其他参数，包括 `expires_in` 和 `token_type` 。这些参数描述了令牌的有效时限（单位为秒），返回的令牌的类型。如果请求中包含状态参数，那么回应中也会包含状态参数。

一个 User Agent 流回应的例子如下：

```
https://oauth2-login-demo.appspot.com/oauthcallback#access_token=1/fFBGRNJru1FQd44AzqT3Zg&token_type=Bea
```

> 注意: 除了上面提到的字段之外，可能会有别的字段也会存在于回应内，您的应用程序不能将这种情况视为错误。上述字段集仅是最小集。

下面是一小段 JavaScript 脚本用于分析回应并且将参数返回给服务器。这个代码被托管在这个 URL 上： `https://oauth2-login-demo.appspot.com/oauthcallback` 。

```
// First, parse the query string
var params = {}, queryString = location.hash.substring(1),
    regex = /([^&=]+)=([^&]*)/g, m;
while (m = regex.exec(queryString)) {
  params[decodeURIComponent(m[1])] = decodeURIComponent(m[2]);
}

// And send the token over to the server
var req = new XMLHttpRequest();
// consider using POST so query isn't logged
req.open('GET', 'https://' + window.location.host + '/catchtoken?' + queryString, true);
```

```
req.onreadystatechange = function (e) {
  if (req.readyState == 4) {
     if(req.status == 200){
       window.location = params['state']
   }
  else if(req.status == 400) {
        alert('There was an error processing the token.')
    }
    else {
      alert('something else other than 200 was returned')
    }
  }
};
req.send(null);
```

这段代码将从接收到的片段中的参数通过 XMLHttpRequest 发送给服务器，并且将访问令牌写入浏览器的本地存储中。后者是一个可选操作，这取决于应用程序是否请求其他 JavaScript 代码来调用谷歌 API。并且请注意这个代码将参数发往 /accepttoken 端点，并且他们是通过 HTTPS 通道传输的。

#

错误回应

如果用户没有许可应用程序所请求的权限，谷歌认证服务器会回应一个错误。这个错误以片段形式返回。

一个错误回应示例如下：

```
https://oauth2-login-demo.appspot.com/oauthcallback#error=access_denied
```

#

验证令牌

以片段形式接收的令牌**必须**被显式地确认。如果没能验证这种方式获取的令牌，将会让您的应用程序更可能遭受责任混淆问题(Confused Deputy Problem)。

您可以通过发送一个 Web 服务请求给谷歌认证服务器来确认令牌，并且对该 Web 服务请求的结果进行一次字符串比对。

#

TokenInfo 验证

通过谷歌认证服务器来验证一个令牌相对来说简单。您的应用程序包含了一个访问令牌在access_token 参数，这个令牌可用于以下端点：

端点：https://accounts.google.com/o/oauth2/auth
描述：接收一个访问令牌并且返回访问令牌的相关信息，包括该令牌所授权的程序，该令牌的目标，用户已经许可的域，令牌的剩余生命期，用户 ID 等。

--

下面是这类请求的一个示例：

```
https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

TokenInfo端点会回应一个 JSON 对象来描述令牌或错误。下面是在没有发生错误的情况下，该对象所包含的字段表：

--

字段： audience（ 受众 ）
描述： 令牌的目标应用程序。

--

字段： scope（ 域 ）
描述： 空格为间隔符的用户已经许可的域集。

字段： userid（用户 ID）

描述： 这个字段只会在 profile 域存在于请求内时才会出现。该字段的值是一个已登录用户的一个恒定的标识符，用于创建和管理您的应用程序的用户会话。这个标识符和客户端 ID 是相互独立的。这意味着在同一个组织内多个应用程序联系同一个 profile 信息成为可能。

--

字段： expires_in（有效期至）

描述： 令牌的剩余生命期，以秒为单位。

下面是回应的一个示例：

```
{
  "audience":"8819981768.apps.googleusercontent.com",
  "user_id":"123456789",
  "scope":"profile email",
  "expires_in":436
}
```

> 注意： 当验证一个令牌时，请务必确认回应中的 audience 字段精确吻合你在开发者控制台中获得的客户端 ID。这个步骤是绝对关键的，因为这是防止混淆责任问题的一个手段。

如果令牌已经过期、或者被篡改、或者权限已经被废除，谷歌认证服务器都会返回一个错误。错误会包含 400 状态码，而 JSON 对象内容如下：

```
{"error":"invalid_token"}
```

认证服务器不会提供任何额外信息来指出失败原因，这是特意的。

#

调用谷歌 API

您的程序获得访问令牌之后，您可以使用令牌以用户或者服务账户的名义来对谷歌 API 进行调用。要做到这一点，请将访问令牌包含到发给 API 的请求中，可以通过包含 access_token 查询参数或者一个 Authorization: Bearer HTTP 头来实现。如果可能的话，我们更欢迎 HTTP 头的方法，因为查询串更容易在服务器记录中可见。在大多数情况下你可以使用客户端库来设置您的谷歌 API 调用（例如，当对谷歌人脉 API 进行调用时）。

你可以在OAuth 2.0 游乐场中自行尝试所有的谷歌 API 并查看他们所对应的域。

#

示例

一个通过使用 access_token 查询串参数对 people.get 端点（谷歌人脉 API）的调用会和下例相类似，当然在实际情况中您需要提供您自己的访问令牌：

```
GET https://www.googleapis.com/plus/v1/people/userId?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

而对于已经认证的用户（me）而言，通过 Authorization: Bearer HTTP 头调用同样的 API 就会像下面这样：

```
GET /plus/v1/people/me HTTP/1.1
Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg
Host: googleapis.com
```

您可以尝试 `curl` 命令行应用程序。下面是使用 HTTP 头的方法（推荐）的示例：

```
curl -H "Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg" https://www.googleapis.com/plus/v1/people/me
```

或者，您也可以使用查询串参数的方法实现：

```
curl https://www.googleapis.com/plus/v1/people/me?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

#

渐进式授权

在 OAuth 协议中，您的应用程序为了访问资源所请求的认证是以域来区分的，如果用户已经被认证而且同意权限请求，您的应用程序会接收到一个生命期比较短的访问令牌，这些令牌能让它访问目标资源，而刷新令牌（可选）可以让应用程序拥有长期的访问权限。

按需请求资源访问权限通常被认为是一项最佳实践。举例来说，一个让人们可以对音乐进行采样并且创建混音的应用程序也许在登陆时候只需要非常少量的资源，说不定除了登陆者的名字之外就没别的了。但是，要保存一个完整的混音可能会需要访问他们的谷歌网盘。大多数人都会觉得在应用程序需要存储文件时申请谷歌网盘的访问权限是非常自然的。

这种情况下，在登录时应用程序可以请求这个域：`https://www.googleapis.com/auth/plus.loginto` 来实现一个基本的社交登陆功能，然后在需要保存混音的时候才申请这个域：`https://www.googleapis.com/auth/drive.file` 。

同时使用在使用 OpenID 连接和使用 OAuth 2.0 来访问谷歌 API 里描述的步骤通常会让你的应用程序不得不管理两个不同的访问令牌。如果您希望避开这种复杂性，则需要在所有 OAuth 2.0 工作流的第一步时，将发送给 `https://accounts.google.com/o/oauth2/auth` 的认证 URI 里添加一个额外的参数。这个参数是 `include_granted_scopes` ，其中值可以被赋为 `true` 或 `false` （默认值是 `false` ），当这个值为 `true` 时，如果您的域认证要求被批准，谷歌认证服务器会将这次认证和所有之前已经成功的认证为这组用户-应用程序合并。这类请求的 URI 可能看上去会像下面这样（下例有插入换行和空格来增强可读性）：

```
https://accounts.google.com/o/oauth2/auth?
  scope=https://www.googleapis.com/auth/drive.file&
  state=security_token%3D138r5719ru3e1%26url%3Dhttps://oa2cb.example.com/myHome&
  redirect_uri=https%3A%2F%2Fmyapp.example.com%2Fcallback&
  response_type=code&
  client_id=8127352506391.apps.googleusercontent.com&
  approval_prompt=force&
  include_granted_scopes=true
```

我们现在把上面这种认证叫做“组合认证”；关于组合认证：

- 您可以使用刚才得到的访问令牌来访问任何在组合认证中已经合并的域。
- 当您使用刷新令牌来进行组合认证时，返回的新访问令牌也代表了组合认证，所以也可以用于访问其任意域。

- 组合认证包括了所有过去已经被许可的权限，即使这些权限请求曾经是从不同的客户端发出的。举例来说，如果您在桌面应用程序请求了 `https://www.googleapis.com/auth/plus.loginscope` ，然后又向同一个用户的移动端发送了同样的请求，那么后面这个请求会被自动批准，因为组合认证会包含两边的域。
- 当您废除了一个代表组合认证的的令牌，其所有的认证都会被同时废除；这意味着如果你还保留着任何以往的权限的令牌，他们也会跟着失效。

5

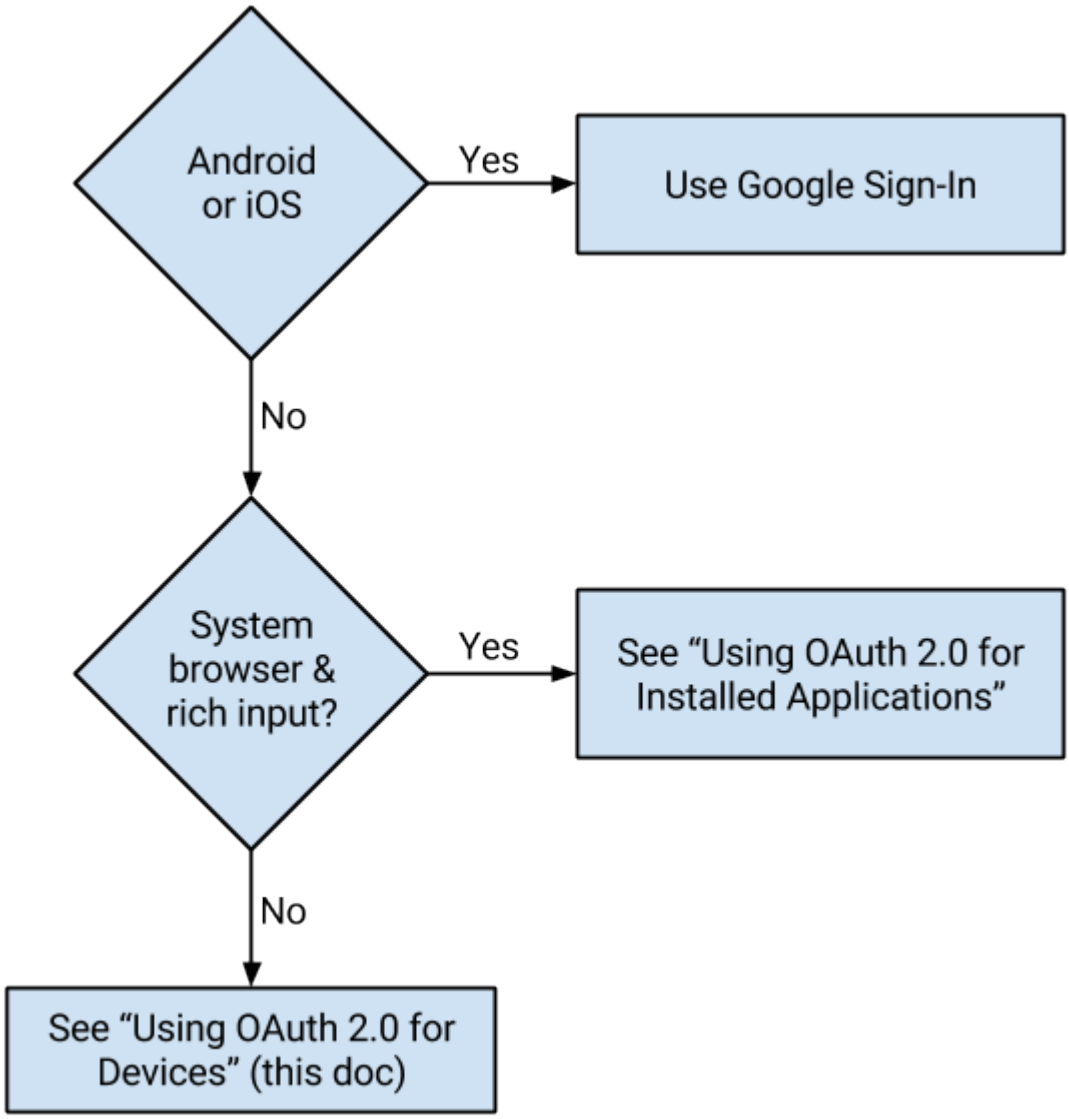
设备中使用 OAuth 2.0

如果您：

- 正在编写一个在本地安装的应用程序，并且该程序不在 Android 或 iOS 上运行。
- 您的应用程序会运行在一个没有系统浏览器、或者输入能力有限的设备上，例如家用游戏机、摄像机、打印机等。
- 您的应用仅需要访问这种工作流所允许的域。

如果您正在为 Android 或 iOS 编写应用程序，请使用谷歌登陆来认证您的用户。谷歌登陆按钮会管理 OAuth 2.0 工作流，包括认证和获取谷歌 API 授权。所有有谷歌账户的人都能用谷歌登陆，不论他们有没有升级到 Google+。若要添加谷歌登陆按钮，请根据平台分别参照 Android 或 iOS。

如果您正在为其他平台开发程序，并且应用程序会运行在拥有系统浏览器和强输入能力的设备（例如拥有键盘的设备）上，那么请参照在本地安装的应用程序上使用 OAuth 2.0。



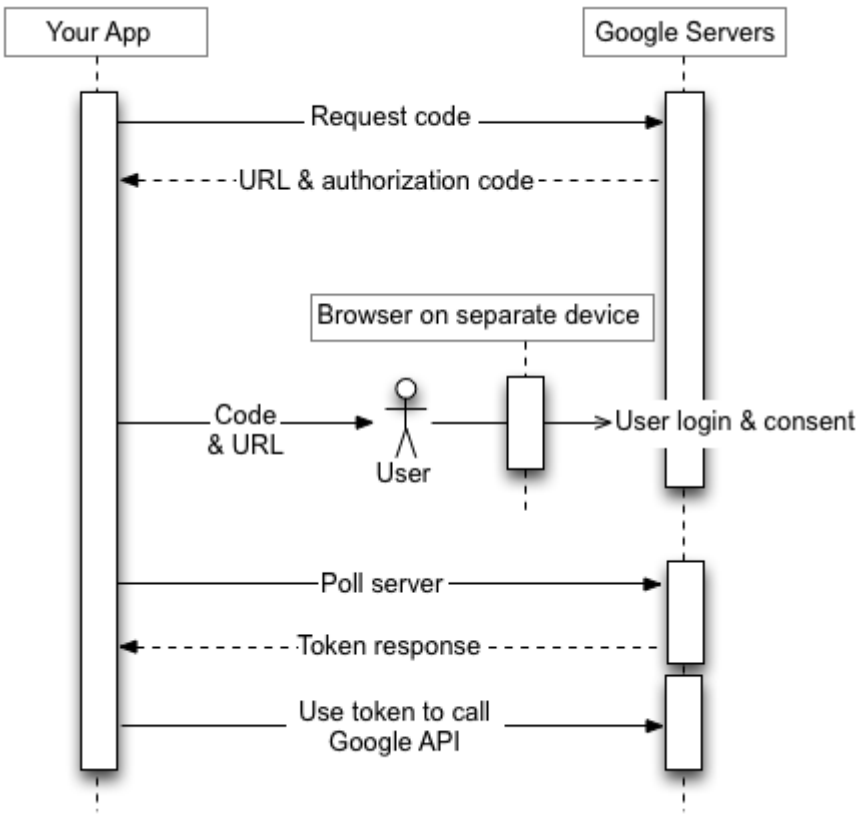
#

概览

运行在输入能力有限的设备（例如家用游戏机、摄像机、打印机等）上的应用程序可以以用户名义访问谷歌 API，不过用户必须通过电脑或者有输入能力的设备来完成剩余操作。工作流如下：

- 您的应用程序发送一个带参数的请求到谷歌 URL 来开始这一工作流。回应会包含一个设备码、一个用户码、一个 URL、一个有效期、和一个建议轮询间隔时间。
- 接收到上述回应之后，您的应用程序将 URL 和用户码展示给用户，并且指引用户打开浏览器，导航到该 URL，并输入代码。
- 用户这时切换到一个拥有更强输入能力的设备或电脑，启动浏览器，导航到先前设备指引的 URL，登陆账户，然后输入代码。
- 在后台，您的应用程序会轮询一个谷歌端点来获取访问令牌和刷新令牌。这些令牌只会在用户登陆和许可请求之后才会被返回到您的应用程序。

使用这种工作流，您只能获取有限的 scope 集合。



考虑到这些设备的能力多样性和运行环境多样性，本文档的示例会使用 `curl` 命令行工具。因为将 `curl` 命令移植到不同的编程语言和运行环境通常都是简单琐碎的。

#

获取用户码

和所有 OAuth 2.0 方案一样，你首先需要在谷歌开发者控制台建立一个工程并且获取一个客户端 ID 和客户端 secret。

获得客户端 ID 和客户端 secret 之后，请发送一个 HTTP `POST` 给 OAuth 2.0 设备端点：`https://accounts.google.com/o/oauth2/device/code` 请求中请包含你的 `client_id`（客户端 ID）和 scopes 的列表。和其他 OAuth 2.0 工作流不一样的是，`response_type`（回应种类）和 `redirect_uri`（重定向 URI）在设备工作流中式不需要的。

下面是请求用户码的一个示例：

```
POST /o/oauth2/device/code HTTP/1.1
Host: accounts.google.com
Content-Type: application/x-www-form-urlencoded

client_id=812741506391-h38jh0j4fv0ce1krdkiq0hfvt6n5amrf.apps.googleusercontent.com&
scope=email%20profile
```

或者你也可以使用 `curl`：

```
curl -d "client_id=812741506391-h38jh0j4fv0ce1krdkiq0hfvt6n5amrf.apps.googleusercontent.com&scope=email profile" h
```

回应会以 JSON 对象的形式返回：

```
{
  "device_code" : "4/4-GMMhmHCXhWEzkobqIHGG_EnNYYsAkukHspeYUk9E8",
  "user_code" : "GQVQ-JKEC",
  "verification_url" : "https://www.google.com/device",
  "expires_in" : 1800,
  "interval" : 5
}
```

JSON 对象内的 `user_code`（用户码）和 `verification_url`（验证 URL）值应该被展示给用户。整个概念就是让用户自己去浏览器并导航到 `verification_url` 中的 URL，然后输入 `user_code` 里面的代码。`user_code` 是大小写敏感的，所以用户需要准确地输入回应中给出的代码。`device_code`、`expires_in` 和 `interval` 字段会在获取访问令牌和刷新令牌 章节中使用到。

#

显示用户码

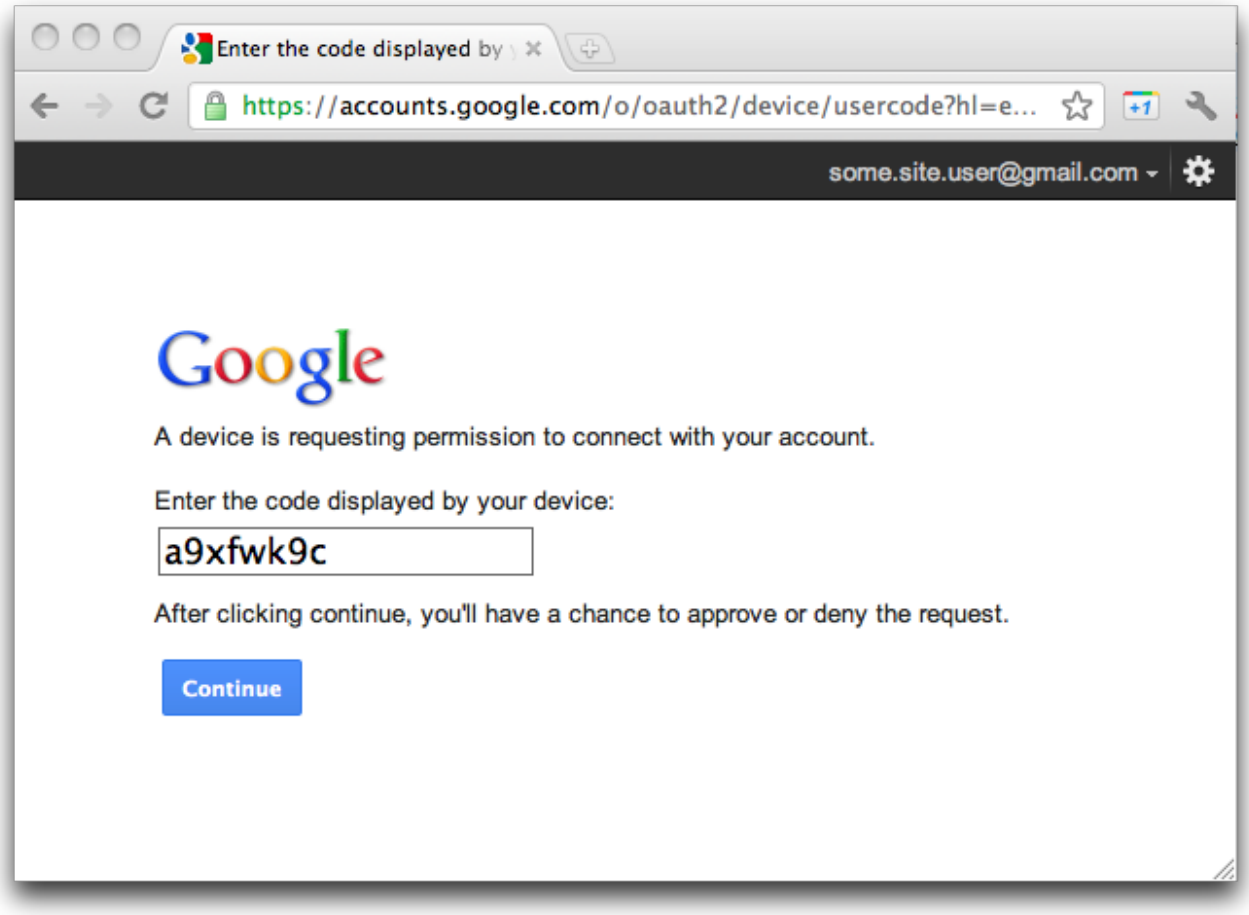
verification_url 和 user_code 的值可能会变更。所以请将您的用户界面设计成有能力应对以下限制：

- user_code 必须在一个可以显示 15 个 ‘W’ 字符宽度的区域中显示。换句话说，如果你能正确显示 WWWWWWWWWWWWWW，那么您的用户界面是有效的。
- 显示 verification_url 的区域需要有足够宽度来应对一个 40 个字符长的 URL 串。

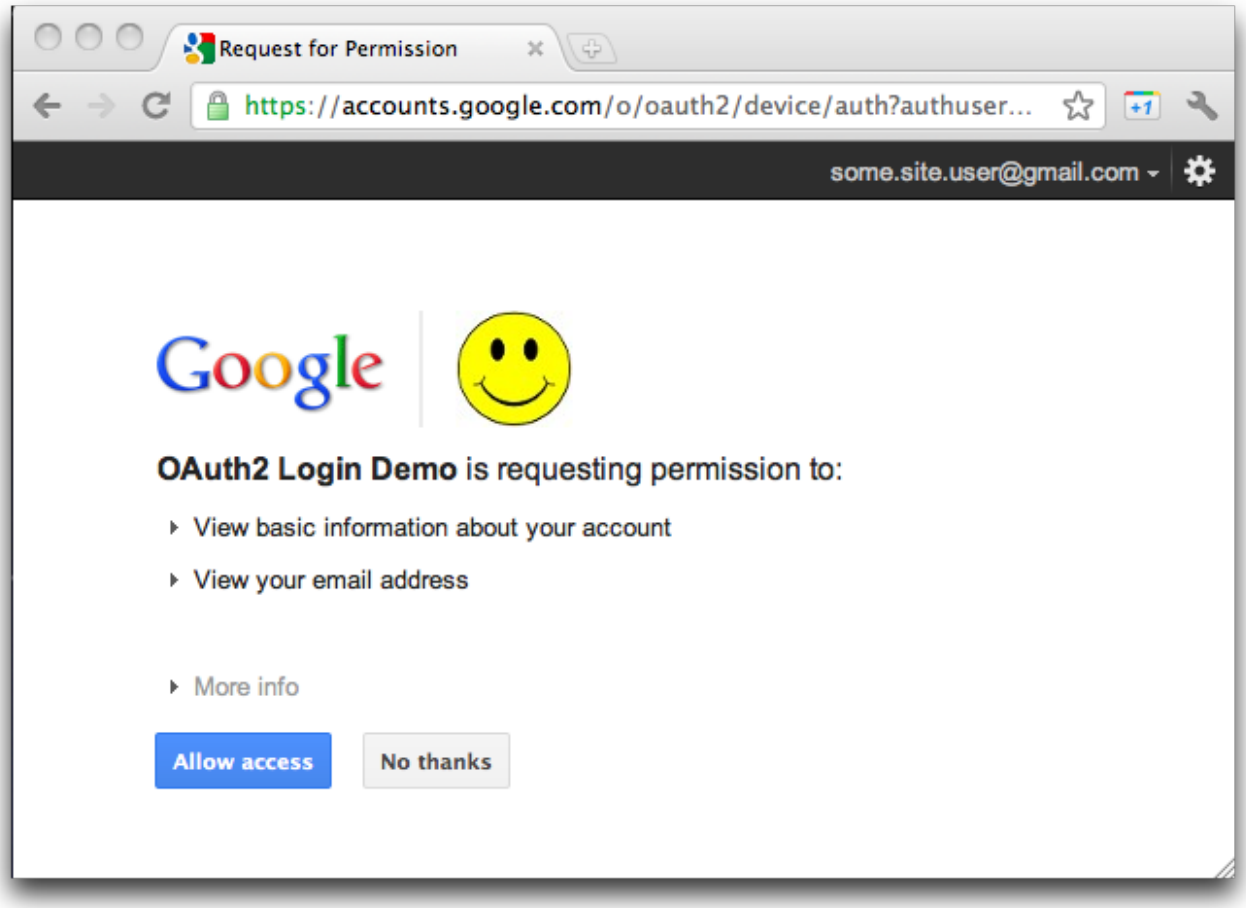
上述两个字符串可能会包含任意可打印的 US-ASCII 字符集中的字符。不要做出任何关于字符串内容的多余的假设。特别的，不要以任何形式修改 user_code 字符串（例如变更大小写或者加入格式化字符），这样做可能会导致未来我们修改代码的时候连同破坏您的程序正常运作。也请不要修改 verification_url，除非选择性的去除 URL 方案（scheme）来优化显示，如果您打算从 URL 中去除 URL 方案（例如“http://”），请确保您的应用程序可以处理 “http” 和 “https” 两种协议。

> 备注：为了防止您的应用程序在未来无法正常工作，请用 15 个字符长度的'W' 用户码来测试您的应用程序，并且在显示时不要修改字符串。

当用户导航到 verification_url 中的 URL 时，他们会见到类似下图的页面：



在用户输入 user_code 之后，他们会被要求登录谷歌，并且会看见用户准许页面（如下图所示）。



如果用户点击“许可权限”，那么您的应用程序就可以获得一个访问令牌和刷新令牌，这两个令牌可以让您的应用程序访问一个或多个谷歌 API。和所有 OAuth 2.0 方案一样，域（ `scope` ）参数会控制您的应用程序拥有哪些访问谷歌 API 的权限。

#

获取访问令牌和刷新令牌

当您的应用程序把 `user_code` 和 `verification_url` 展示给用户之后，就可以用当时和 `user_code` 和 `verification_url` 一同返回的 `device_code` 参数来轮询谷歌端点。被轮询的端点 URL 是 `https://www.googleapis.com/oauth2/v3/token` ，而每次请求之间的轮询间隔在 `interval` 参数中定义，单位为秒。

这种请求如下例所示：

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

client_id=812741506391-h38jh0j4fv0ce1krdkiq0hfvt6n5amrf.apps.googleusercontent.com&
client_secret={clientSecret}&
code=4/4-GMMhmHCXhWEzkobqIHGG_EnNYYsAkukHspeYUk9E8&
grant_type=http://oauth.net/grant_type/device/1.0
```

或者使用 curl:

```
curl -d "client_id=812741506391-h38jh0j4fv0ce1krdkiq0hfvt6n5amrf.apps.googleusercontent.com&
client_secret={clientSecret}&
code=4/4-GMMhmHCXhWEzkobqIHGG_EnNYYsAkukHspeYUk9E8&
grant_type=http://oauth.net/grant_type/device/1.0"
https://www.googleapis.com/oauth2/v3/token
```

如果用户还没有同意请求，那么回应会像下面这样：

```
{
  "error" : "authorization_pending"
}
```

那么您的应用程序应该以不快于 interval 字段值的速率重复发送请求。如果您的应用程序以过快的速度轮询，那么回应会变成下面这样：

```
{
  "error" : "slow_down"
}
```

一旦用户登陆谷歌并且许可您的应用程序访问谷歌 API，那么下一次轮询请求会得到一个访问令牌和刷新令牌（如下所示）。

```
{
  "access_token" : "ya29.AHES6ZSuY8f6WFLswSv0HELP2J4cCvFSj-8GiZM0Pr6cgXU",
  "token_type" : "Bearer",
  "expires_in" : 3600,
  "refresh_token" : "1/551G1yXUqgkDGnkfFk6ZbjMLMDIMxo3JFc8lY8CAR-Q"
}
```

接到这个回应之后，您的应用程序就可以在谷歌 API 请求中使用访问令牌了。访问令牌的生命期是有限的。如果您的应用程序需要在长时间之后访问 API，那么可以使用刷新令牌来获得一个新的访问令牌（请参见使用刷新令牌）。如果您的应用程序需要这种权限，那么就需要将刷新令牌保存下来以便日后使用。

#

调用谷歌 API

您的程序获得访问令牌之后，您可以使用令牌以用户或者服务账户的名义来对谷歌 API 进行调用。要做到这一点，请将访问令牌包含到发给 API 的请求中，可以通过包含 `access_token` 查询参数或者一个 `Authorization: Bearer` HTTP 头来实现。如果可能的话，我们更欢迎 HTTP 头的方法，因为查询串更容易在服务器记录中可见。在大多数情况下你可以使用客户端库来设置您的谷歌 API 调用（例如，当对谷歌人脉 API 进行调用时）。

你可以在OAuth 2.0 游乐场中自行尝试所有的谷歌 API 并查看他们所对应的域。

#

示例

一个通过使用 access_token 查询串参数对 people.get 端点（谷歌人脉 API）的调用会和下例相类似，当然在实际情况中您需要提供您自己的访问令牌：

```
GET https://www.googleapis.com/plus/v1/people/userId?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

而对于已经认证的用户（me）而言，通过 Authorization: Bearer HTTP 头调用同样的 API 就会像下面这样：

```
GET /plus/v1/people/me HTTP/1.1
Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg
Host: googleapis.com
```

您可以尝试 `curl` 命令行应用程序。下面是使用 HTTP 头的方法（推荐）的示例：

```
curl -H "Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg" https://www.googleapis.com/plus/v1/people/me
```

或者，您也可以使用查询串参数的方法实现：

```
curl https://www.googleapis.com/plus/v1/people/me?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

#

使用刷新令牌

访问令牌终会过期。API 会通过返回 `401` 状态码来指示一个访问令牌已经过期。要获得一个新的访问令牌，发送一个请求给令牌端点，并且包含 `client_id` 、 `client_secret` 、 `refresh_token` 和 `grant_type` 参数（如下所示）。

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

client_id=812741506391-h38jh0j4fv0ce1krdkiq0hfvt6n5amrf.apps.googleusercontent.com&
client_secret={clientSecret}&
refresh_token=1/551G1yXUqgkDGnkfFk6ZbjMLMDIMxo3JFc8lY8CAR-Q&
grant_type=refresh_token
```

请注意刷新令牌的发放数量是有限制的；限制的单位是每一组客户端-用户，你应该在长期存储器中保存刷新令牌并一直使用它直到过期。如果您的应用程序请求太多刷新令牌，就可能会触发限制，这种情况下最旧的令牌会失效。

#

允许的域

当您在设备上使用这种 OAuth 2.0 工作流，你只能获取以下域：

分析工具配置和报告 API

https://www.googleapis.com/auth/analytics

https://www.googleapis.com/auth/analytics.readonly

日历 API

https://www.googleapis.com/auth/calendar

https://www.googleapis.com/auth/calendar.readonly

通讯录 API (只读)

https://www.googleapis.com/auth/contacts

云打印 API

https://www.googleapis.com/auth/cloudprint

云存储 API

https://www.googleapis.com/auth/devstorage.full_control

https://www.googleapis.com/auth/devstorage.read_write

健身 REST API

https://www.googleapis.com/auth/fitness.activity.read

https://www.googleapis.com/auth/fitness.activity.write

https://www.googleapis.com/auth/fitness.body.read

https://www.googleapis.com/auth/fitness.body.write

https://www.googleapis.com/auth/fitness.location.read

https://www.googleapis.com/auth/fitness.location.write

聚合表 API

https://www.googleapis.com/auth/fusiontables

谷歌登陆

email

profile

https://www.googleapis.com/auth/plus.me

YouTube 数据和直播流 API

https://www.googleapis.com/auth/youtube

https://www.googleapis.com/auth/youtube.readonly

https://www.googleapis.com/auth/youtube.upload

6

服务类应用中使用 OAuth 2.0

谷歌的 OAuth 2.0 系统支持 server-to-server 的交互，类似于 Web 应用于谷歌服务那样的交互。为了使用这类方案，你需要一个服务账号，这个账号是属于你的应用程序的，用以代替个人终端用户。你的应用程序被服务账号代替调用谷歌的应用接口，所以用户不用直接参与。这个方案有时被称为 two-legged OAuth 或者 2LO。（相关词 three-legged OAuth 指的是一个应用程序代替终端用户调用谷歌应用接口的方案，有时是需要用户同意的）

特别地，当应用程序调用谷歌应用接口时，应用程序使用服务账号，更多是处理应用程序本身的数据而不是用户的数据。例如应用程序使用谷歌的云存储服务来储存持久化数据时，将会通过服务账号来鉴定应用的调用。

如果你拥有一个谷歌应用域名，又或者你使用谷歌应用来工作，例如，谷歌应用的管理员可以在的应用域名里授权应用代表用户访问用户数据。再例如，调用谷歌日历接口的应用，将在域名内代替用户添加事项到所有用户的日历中去。在域名内，服务账号被授权代替用户访问数据，有时称为域范围内的授权认证。

> 注意：当你使用谷歌应用市场来安装应用程序到你的设备上时，请求的权限将会自动授予给应用。你不需要手动为应用使用的服务账号授权。

以下文档描述了一个应用程序如何使用谷歌应用接口的客户端程序库或者 http，来完成 server-to-server 的 OAuth 2.0 协议。

#

内容

#

概述

为了支持 server-to-server 的交互，首先在开发者面板中为你的项目创建一个服务账号。如果你想在谷歌应用中，让用户可以访问用户数据，然后代理就会为服务账号使用域内访问。

接着，准备你的应用程序，通过服务账号的证书来调用授权接口，向 OAuth 2.0 auth 服务发出请求，获取访问令牌。

最后，你的应用程序能够使用访问令牌来调用谷歌应用接口。

> 建议：你的应用程序可以通过你熟悉的编程语言的谷歌接口库或者通过使用 OAuth 2.0 系统和 HTTP 来完成这些任务。然而，server-to-server 的授权交互的体系结构会要求应用创建用于存储密钥的 JSON Web Tokens，这种做法非常容易出现错误，严重影响到你应用的安全。
>
> 因为这个原因，我们强烈建议你使用封装好的库，例如 Google APIs client libraries，它从你的代码中抽象了加密过程。

#

创建服务账号

服务账号的证书包含了唯一的电子邮件地址，一个客户 ID，至少一对公共/私人密钥。

如果你的程序运行在谷歌应用引擎上，当你创建项目时就会自动建立起一个服务账号。

如果你的应用运行在谷歌计算引擎上，当你创建项目时也会自动建立起一个服务账号，但是在你创建谷歌计算引擎实例时，你必须指定应用可以访问的范围。浏览更多信息，请查看 Preparing an instance to use service accounts。

如果你的应用并没有运行在谷歌应用引擎或者谷歌计算引擎上时，你必须在谷歌开发者面板中获取证书。为了生成一个服务账号的证书，或者浏览一个你早已生成公共证书，按如下步骤执行：

1. 访问谷歌开发者面板。
2. 选中一个项目或者新建一个项目。
3. 在滑动条的左边，展开 APIs & auth 选项。然后点击 APIs。在 API 部分选中 Enabled APIs 来罗列所有可用的 APIs。如果你没有让 API 处于可用状态，在 APIs 列表中选中 Enabled API 按钮 ，然后使其可用。
4. 在滑动条的左边，选中证书。
5. 为了建立一个新的服务账号，按如下步骤：

   a. 在 OAuth 下，选中 Create new Client ID。

   b. 当出现提示时，选中 Service Account 和 Create Client ID。

   c. 弹出一个对话框，点击 OK，就行了。

你创建了新的密钥，然后你就可以下载到你的机器上了。下载的仅仅只是副本。你负责安全地保存它。控制台只会在你初始化服务账号时显示你的私有密钥的密码，然后密码就不会再次显示了。你现在获取了一个 Generate New JSON Key 和 Generate New P12 Key 以及删除密钥的权利。

在任何时刻你都可以返回开发者面板去查看 client ID，电子邮件地址，公共密钥指纹或者生成额外的公共密钥/私有钥对。了解更多服务账号证书的细节请打开开发者面板，参阅 Service accounts 的帮助文件。

留意服务账号的电子邮箱地址，存储服务账号的 P12 私有密钥到本地，让你的应用程序可以访问到这些东西。你的应用程序在调用授权接口时需要用到它们。

> 注意：在开发环境或者产品环境中，你都必须存储以及严密地管理自己的私有密钥。谷歌不会保存你的私有密钥，只会保存公共密钥。

#

服务账号域内授权

如果你的应用想访问用户数据，你创建的服务账号应确保有访问那些你想要访问的谷歌应用域内用户数据的权限。

以下步骤必须由管理员执行:

1. 前往你的谷歌应用的管理员面板。

2. 在列表中选中 Security。如果你不能看见 Security 列表，在页面底部的灰色栏中选中 More controls，然后在列表中选中 Security，确保你是以管理员身份执行。
3. 选中 Show more，然后在选项中选中 Advanced settings。
4. 在 Authentication 中选中 Manage API client access。
5. 在 Client Name 字段中输入服务帐户的 Client ID。
6. 在 One or More API Scopes 字段输入你应用可以访问的范围。例如，你的应用需要域内访问谷歌的导航应用接口以及谷歌日历应用接口，便可以输入： https://www.googleapis.com/auth/drive，https://www.googleapis.com/auth/calendar。
7. 点击 Authorize。

您的应用程序现在有应用接口调用权限，就像你域内的一个用户(模拟用户)。每当你准备使用授权接口时，指定了模拟用户。

#

准备做一个授权 API 调用

#

Java

在你从开发者面板获取客户电子邮箱和私有密钥之后，使用 Java 版的 Google APIs Client Library，根据服务账号的证书以及你的应用需要访问的范围来创建 `GoogleCredential` 对象。例如：

平台： Google App Engine

代码：

```
import com.google.api.client.googleapis.extensions.appengine.auth.oauth2.AppIdentityCredential;
import com.google.api.services.sqladmin.SQLAdminScopes;

// ...

AppIdentityCredential credential =
    new AppIdentityCredential(SQLAdminScopes.SQLSERVICE_ADMIN);
```

> 注意：如果你的你应用运行在 Google App Engine，你只能使用 AppIdentityCredential 证书对象。如果你的应用需要在其他运行环境上运行，在本地测试你的应用时，必须检测环境以及使用不同的证书机制（查看其他平台）。

平台：Google Compute Engine

代码：

```
import com.google.api.client.googleapis.compute.ComputeCredential;
import com.google.api.client.googleapis.javanet.GoogleNetHttpTransport;
import com.google.api.client.http.HttpTransport;
import com.google.api.client.json.JsonFactory;
import com.google.api.client.json.jackson2.JacksonFactory;

// ...

JsonFactory JSON_FACTORY = JacksonFactory.getDefaultInstance();
HttpTransport httpTransport = GoogleNetHttpTransport.newTrustedTransport();
ComputeCredential credential = new ComputeCredential.Builder()
    .setTransport(httpTransport)
    .setJsonFactory(JSON_FACTORY)
    .build();
```

> 注意：如果你的应用程序运行在 `Google Compute Engine`，你只能使用 `ComputeCredential` 证书对象。如果你的应用需要在其他运行环境上运行，在本地测试你的应用时，必须检测环境以及使用不同的证书机制（查看其他平台）。你可以使用默认证书来简化流程。

平台：其他平台

代码：

```
import com.google.api.client.googleapis.auth.oauth2.GoogleCredential;
import com.google.api.client.googleapis.javanet.GoogleNetHttpTransport;
import com.google.api.client.http.HttpTransport;
import com.google.api.client.json.JsonFactory;
import com.google.api.client.json.jackson2.JacksonFactory;
import com.google.api.services.sqladmin.SQLAdminScopes;

// ...

String emailAddress = "123456789000-abc123def456@developer.gserviceaccount.com";
JsonFactory JSON_FACTORY = JacksonFactory.getDefaultInstance();
HttpTransport httpTransport = GoogleNetHttpTransport.newTrustedTransport();
```

```
GoogleCredential credential = new GoogleCredential.Builder()
    .setTransport(httpTransport)
    .setJsonFactory(JSON_FACTORY)
    .setServiceAccountId(emailAddress)
    .setServiceAccountPrivateKeyFromP12File(new File("MyProject.p12"))
    .setServiceAccountScopes(Collections.singleton(SQLAdminScopes.SQLSERVICE_ADMIN))
    .build();
```

如果你委托域内访问给服务账号，然后你想模仿一个用户账号，要指定用户的电子邮箱给 `GoogleCredential` 对象的 `setServiceAccountUser` 方法。例如:

```
GoogleCredential credential = new GoogleCredential.Builder()
    .setTransport(httpTransport)
    .setJsonFactory(JSON_FACTORY)
    .setServiceAccountId(emailAddress)
    .setServiceAccountPrivateKeyFromP12File(new File("MyProject.p12"))
    .setServiceAccountScopes(Collections.singleton(SQLAdminScopes.SQLSERVICE_ADMIN))
    .setServiceAccountUser("user@example.com")
    .build();
```

在你的应用中使用 `GoogleCredential` 对象来调用谷歌应用接口。

#

Python

在你从开发者面板获取客户电子邮箱和私有密钥之后，使用 Python 版的 Google APIs Client Library 完成如下步骤:

1. 根据服务账号的证书以及你的应用需要访问的范围来创建 `GoogleCredential` 对象。例如:

   平台: Google App Engine

   代码:

   ``` from oauth2client.appengine import AppAssertionCredentials

   credentials = AppAssertionCredentials( 'https://www.googleapis.com/auth/sqlservice.admin')

   ```

   > 注意：如果你的应用运行在谷歌应用引擎，你只能使用 AppIdentityCredential 证书对象。如果你的应用需要在其他运行环境上运行，在本地测试你的应用时，必须测试环境以及使用不同的证书机制（查看其他平台）。

平台： Google Compute Engine

代码：

```
from oauth2client.gce import AppAssertionCredentials

credentials = AppAssertionCredentials( 'https://www.googleapis.com/auth/sqlservice.admin')
```

> 注意：如果你的你应用运行在 `Google Compute Engine`，你只能使用 `ComputeCredential` 证书对象。如果你的应用需要在其他运行环境上运行，在本地测试你的应用时，必须测试环境以及使用不同的证书机制（查看其他平台）。你可以使用默认的证书来简化流程。

平台： Other

代码：

```
from oauth2client.client import SignedJwtAssertionCredentials

client_email = '123456789000-abc123def456@developer.gserviceaccount.com' with open("MyProject.p12") as f: private_key = f.read()

credentials = SignedJwtAssertionCredentials(client_email, private_key, 'https://www.googleapis.com/auth/sqlservice.admin')
```

如果你委托域内访问给服务账号，然后你想模仿一个用户账号，在创建 `Credentials` 对象时要指定用户的电子邮箱给到 `sub` 参数，例如：

```
credentials = SignedJwtAssertionCredentials(client_email, private_key, 'https://www.googleapis.com/auth/sqlservice.admin', sub='user@example.org')
```

2. 使用 `Credentials` 对象的 `authorize` 方法来适配 `httplib2.Http` 实例所有请求的必要证书头。

```
from httplib2 import Http

http_auth = credentials.authorize(Http())
```

在你的应用中使用被授权后的 `Http` 对象来调用谷歌应用接口。

#

HTTP/REST

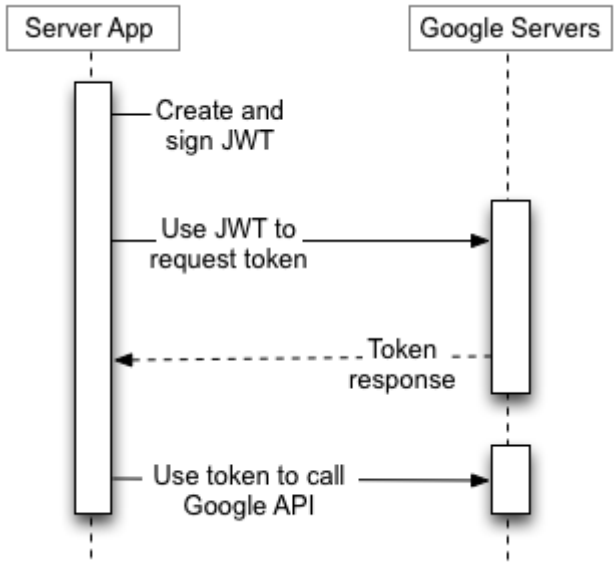
> 建议：你的应用程序可以通过使用 OAuth 2.0 系统和 HTTP 来完成这些任务。然而 server-to-server 的授权交互的体系结构会要求应用创建用于存储密钥的 JSON Web Tokens，这种做法非常容易出现错误，严重影响到你应用程序的安全。
>
> 因为这个原因，我们强烈建议你使用封装好的库，例如 Google APIs client libraries，他从你的代码中抽象了加密过程。

在你从开发者面板获取到 client ID 和私人密钥后，你需要完成如下步骤：

1. 一个 JSON Web Tokens，包含了一个头部，一个设置，一个签名。
2. 从 Google OAuth 2.0 授权服务请求一个访问码。
3. 处理授权服务返回的 JSON 响应。

如果响应包含了一个访问码，你可以使用这个访问码来调用谷歌应用接口。（如果响应没有包含访问令牌，你的JSON Web Token 和令牌请求或许不符合格式，或者服务账号没有权限去访问被请求的范围。）

当访问码的过了有效期，你的应用会生成另一个 JWT，记下他，然后使用它来请求另一个访问码。



图片 6.1 serviceaccount

本章节余下内容包括如下：创建一个 JWT 的细节，记录 JWT，格式化访问令牌请求，处理响应。

#
创建一个 JWT

一个 JWT 是由三部分组成：一个头部，一个要求设置，一个签名。头部和要求设置是 JSON 对象，这些 JSON 对象被序列化成 UTF-8 字节，然后通过 Base64url 进行编码。由于重复编码操作对编码的变化提供了弹性。头部，要求设置，签名通过（.）字符来串接在一起。

一个 JWT 的组成如下：

```
{Base64url encoded header}.{Base64url encoded claim set}.{Base64url encoded signature}
```

签名的组成如下：

```
{Base64url encoded header}.{Base64url encoded claim set}
```

#
格式化 JWT 的头部

头部由两个字段组成，一个是签名的算法，一个是格式的类型。全部的字段都是强制性的，每个字段只有一个值。说明了其他的算法和格式，头部也会相应的变化。

服务账号依赖于 RSA SHA-256 算法和 JWT token 格式。一个头部的 JSON 示例如下：

```
{"alg":"RS256","typ":"JWT"}
```

Base64url 编码后的展示如下：

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9
```

#
格式化 JWT 的要求设置

JWT 的要求设置包含了 JWT 的相关信息，包含了请求的权限，目标的令牌，发行者，令牌被生产时的时间，令牌的有效期。大部分字段是必要的，就像 JWT 的头部，JWT 的要求设置是一个 JSON 对象，用于签名的计算。

#
请求要求

在下面展示了请求要求的字段。他们可以以任意的顺序出现。

字段名	描述
iss	服务账号的电子邮箱
scope	一个用空格分隔应用程序请求的权限列表。
aud	目标的描述符的声明。做一个访问令牌请求时这个值总是 https://www.googleapis.com/oauth2/v3/token.
exp	声明的过期时间，指定为秒就是从UTC，1970年1月1日。这个值最多1小时后发布时间。
iat	声明时，指定为秒就是从 UTC，1970年1月1日。

一个 JWT 的要求设置的 JSON 如下：

```
{
  "iss":"761326798069-r5mljlln1rd4lrbhg75efgigp36m78j5@developer.gserviceaccount.com",
  "scope":"https://www.googleapis.com/auth/devstorage.readonly",
  "aud":"https://www.googleapis.com/oauth2/v3/token",
  "exp":1328554385,
  "iat":1328550785
}
```

#

额外的要求

在一些企业的案例中，在组织内应用程序能代表一个特定的用户请求权限。在允许执行这种模拟类型的操作前，通常是由域管理员授权应用程序模拟一个用户。查看更多有关于域管理员的信息，请前往 Managing API client access。

为了获取访问码，使应用可以访问到包括用户的电子邮箱在内的资源，JWT 的要求设置的 `sub` 字段描述如下：

字段名	描述
sub	应用程序请求授权访问用户的电子邮件地址。

如果应用程序没有权限去模拟用户，那么响应的访问码包含的 `sub` 字段将是一个错误。

包含 `sub` 字段的 JWT 要求设置如下：

```
{
  "iss":"761326798069-r5mljlln1rd4lrbhg75efgigp36m78j5@developer.gserviceaccount.com",
  "sub":"some.user@example.com",
  "scope":"https://www.googleapis.com/auth/prediction",
  "aud":"https://www.googleapis.com/oauth2/v3/token",
  "exp":1328554385,
  "iat":1328550785
}
```

#
对 JWT 的要求设置进行编码

就像 JWT 的头部，要求设置应该被序列化成 UTF-8 字节，然后通过 Base64url 进行编码。下面的就是一个 JWT 的要求设置的 JSON 例子和 Base64url-safe 例子：

```
{
  "iss":"761326798069-r5mljlln1rd4lrbhg75efgigp36m78j5@developer.gserviceaccount.com",
  "scope":"https://www.googleapis.com/auth/prediction",
  "aud":"https://www.googleapis.com/oauth2/v3/token",
  "exp":1328554385,
  "iat":1328550785
}
```

```
eyJpc3MiOiI3NjEzMjY3OTgwNjktcjVtbGpsbG4xcmQ0bHJiaGc3NWVmZ2lncDM2bTc4ajVAZGV2ZWxvcGVyLmdzZXJ2a
```

#
计算签名

JSON Web Signature是用来为 JWT 生成签名的。输入的签名就如下列的字符数组：

```
{Base64url encoded header}.{Base64url encoded claim set}
```

在计算签名的时候，必须使用 JWT 头部中指定的签名算法。Google OAuth 2.0 授权服务只支持使用 SHA-256 哈希算法的 RSA。这个算法在 JWT 头部的 `alg` 字段表示成 RS256。

使用 SHA256withRSA（就是使用 SHA-256 哈希函数的 RSASSA-PKCS1-V1_5-SIGN）和从开发者面板获取的私人密钥计算输入的 UTF-8 序列。将输出字符数组。

签名必须被 Base64url 编码。头部，要求设置，签名，通过使用实心圆点（.）进行串接，其结果就是 JWT。他应该像如下展示那样：

```
{Base64url encoded header}.
{Base64url encoded claim set}.
{Base64url encoded signature}
```

下面是 JWT 被 Base64url 编码前的例子：

```
{"alg":"RS256","typ":"JWT"}.
{
"iss":"761326798069-r5mljlln1rd4lrbhg75efgigp36m78j5@developer.gserviceaccount.com",
"scope":"https://www.googleapis.com/auth/prediction",
"aud":"https://www.googleapis.com/oauth2/v3/token",
"exp":1328554385,
"iat":1328550785
```

```
}.
[signature bytes]
```

下面是 JWT 被 Base64url 编码后，并准备传输的例子：

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiI3NjEzMjY3OTgwNjktcjVtbGpsbG4xcmQ0bHJiaGc3NWVmZ2IncDM2bTc4ajVAZGV2ZWxvcGVyLmdzZXJ2a
ixOUGehweEVX_UKXv5BbbwVEdcz6AYS-6uQV6fGorGKrHf3LlJnyREw9evE-gs2bmMaQI5_UbabvI4k-mQE4kBqtmS
```

#

创建访问令牌请求

在生成一个 JWT 之后，应用可以使用这个 JWT 来请求访问码。访问码请求是一个 HTTPS 的 `POST` 请求， POST 的数据包会被 URL 编码。 URL 是：

```
https://www.googleapis.com/oauth2/v3/token
```

在 HTTPS `POST` 请求中的参数如下：

字段名	描述
grant_type	使用如下的字符串，如有必要请进行 URL 编码：urn:ietf:params:oauth:grant-type:jwt-bearer。
assertion	JWT，包括签名。

下面是用来请求访问码的 HTTPS `POST` 的原生数据：

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer&assertion=eyJhbGciOiJSUzI1NiIsInR5cCI
```

下面使用使用 `curl` 做相同的请求：

```
curl -d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer&assertion=eyJhbGciOiJSUzI1NiIs
' https://www.googleapis.com/oauth2/v3/token
```

#

处理响应

如果 JWT 和访问码请求被适当的编码，以及服务账号拥有执行的权限，从授权服务返回 JSON 响应包含一个访问码，一个响应例子如下：

```
{
  "access_token" : "1/8xbJqaOZXSUZbHLl5EOtu1pxz3fmmetKx9W8CV4t79M",
  "token_type" : "Bearer",
```

```
  "expires_in" : 3600
}
```

访问码的有效期为一个小时，在有限期内可以被有效使用的。

#

调用谷歌应用接口

#

JAVA

使用 GoogleCredential 对象完成调用谷歌应用接口的步骤如下：

1. 通过 GoogleCredential 对象创建一个服务对象调用你想调用的接口。例如：

   ``` SQLAdmin sqladmin = new SQLAdmin.Builder(httpTransport, JSON_FACTORY, credential).build();

   ```

2. 使用 interface provided by the service object 为应用接口服务创建一个请求。例如，列出 exciting-example-123 工程的实例化的云数据库：

   ``` SQLAdmin.Instances.List instances = sqladmin.instances().list("exciting-example-123").execute();

   ```

#

Python

使用被授权了的 HTTP 对象完成调用谷歌应用接口，步骤如下：

1. 创建一个服务对象调用你想调用的接口。通过创建函数以及名字、接口的版本以及被授权的 HTTP 对象来创建一个服务对象。例如，调用版本号为 1beta3 的云数据库管理员接口的代码如下：

   ``` from apiclient.discovery import build

   sqladmin = build('sqladmin', 'v1beta3', http=http_auth)
   ```

```

2. 使用 interface provided by the service object 为应用接口服务创建一个请求。例如，列出 exciting-example-123 工程的实例化的云数据库：

   ``` response = sqladmin.instances().list(project='exciting-example-123').execute()

   ```

#

HTTP/REST

在你的应用获取到访问令牌之后，你能使用这个令牌代替用户或者服务账号来调用谷歌应用接口。为了完成以上功能，在调用应用接口的请求通过 access_token 字段查询参数或者 Authorization: Bearer HTTP 头部包含访问码。如果允许，使用 HTTP 头部包含访问令牌会更好，因为查询字符串容易被泄露。你可以通过封装的库来建立更多的谷歌应用调用的事例。（例如调用 People API ）。

你可以尝试所有的谷歌应用接口，在 OAuth 2.0 Playground 查看它们的应用范围。

#
例子

如下，调用 people.get 的终端通过使用 access_token 查询字符串参数，你需要指定属于你自己的访问码：

```
GET https://www.googleapis.com/plus/v1/people/userId?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

下面是授权用户使用 Authorization: Bearer 头部完成相同的调用：

```
GET /plus/v1/people/me HTTP/1.1
Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg
Host: googleapis.com
```

你可以尝试在控制台应用中使用 cur 。下面是使用 HTTP 头部完成相同的调用功能：

```
curl -H "Authorization: Bearer 1/fFBGRNJru1FQd44AzqT3Zg" https://www.googleapis.com/plus/v1/people/me
```

或者，作为一种完成相同的调用功能选择，使用查询字符串参数：

```
curl https://www.googleapis.com/plus/v1/people/me?access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

#
访问码过期
```

通过 Google OAuth 2.0 Authorization Server 发布的访问码的有效期是一个小时。当访问码失效后，应用应该生成另一个 JWT 来请求另一个新的访问码。

7

跨客户端身份凭证

当开发者构建软件时，会经常包含一些模块，例如，运行 Web 服务的模块，依赖浏览器的模块，运行 native 移动应用的模块。开发人员和使用软件的人，通常认为这些模块应该是应用程序的一部分。

谷歌的 OAuth 2.0 支持上述习惯。你必须在谷歌开发者面板中构建软件。谷歌开发者面板中的单位组织叫做 project，可以对应于一个多组件的应用程序。对每一个工程，你可以写上一些品牌信息，你必须指定应用程序将会访问的应用接口。开发者面板会生成一个独一无二的字符串 client ID，用来标志每一个多组件应用程序的组件。

#

跨客户端授权的目标

被一个或多个 `scope` 字符串标识的应用程序，当其使用 OAuth 2.0 授权时，它将代替用户来请求一个 OAuth 2.0 的访问令牌来访问资源。通常要由用户来批准应用程序的访问。

当应用程序被用户授予特定 `scope` 内的访问权限时，用户会留意到用户准许界面，这个准许界面包含了你在谷歌开发者面板中构建的工程的版本号和产品品牌信息。（关于如何构建同意（协议）界面的更多信息，请查看在开发者控制面板的 Setting up OAuth 2.0 ）。因此，谷歌认为在一个项目中，当用户授权任何访问特定 `scope` 的 client ID 时,既表示用户信任整个 `scope` 的申请。

效果就是，无论何时，当应用程序的组件被谷歌授权服务系统可靠地授权后，就不会多次提醒用户批准同一个应用访问资源。目前支持上诉效果的有 Web 客户端, JavaScript 的客户端和 Android 应用程序。

#

跨客户端访问令牌

依赖于运行代码的平台，软件能够通过多种多样的途径获取 OAuth 2.0 的访问令牌。更多信息，请查看 Using OAuth 2.0 to Access Google APIs 和 Google Play Services Authorization。通常应用程序获取一个访问令牌需要得到用户审批。

幸运的是，无论何时，在同一个项目中给其他组件授权，谷歌授权服务系统都能够使用到与用户批准授权的，给定项目内的 client ID 的相关信息。

效果就是，在同一个项目的同一个范围中，如果安卓应用请求了一个访问令牌，而且用户早就给 Web 组件授权了，用户就不会被再次询问授权。如下途径也是适用的：在安卓应用上，如果访问的范围已经被授权过了，用户就不用再次授权 Web 组件。

#

安卓 ID 令牌

跨客户端身份标志的一个好处就是，无需用户登录，仅利用它以及 ID 令牌，就可以让安卓应用和他们的主服务端沟通。

#

例子

假设开发者控制面板中有一个项目，这个项目包含了一个客户 ID 为 `9414861317621.apps.googleusercontent.com` 的服务端控件（一个应用程序）。 再假设在这个项目中还有一个原生安卓应用。

这个安卓应用能够代替设备（手机）上任意一个谷歌账号调用 GoogleAuthUtil.getToken() 方法，调用时， `audience:server:client_id:` 这个前缀会通过 Web 组件的客户 ID 来进行修正补全，也就是 `scope` 参数设置为 `audience:server:client_id:9414861317621.apps.googleusercontent.com` 。

在这个方案中， `GoogleAuthUtil` 这个对象将会察觉到在同一个项目中的安卓应用和 Web 组件的客户 ID ，不用用户同意，就可以提供谷歌验证过的 ID 令牌给应用。 这个 ID 令牌包含了 一些字段，其中一些比较重要的字段如下：

- `iss` ：总是 `accounts.google.com`
- `aud` ：项目中 Web 组件的客户 ID
- `azp` ：项目中安卓应用的客户 ID
- `email` ：用来标识用户请求令牌的电子邮件

ID 令牌被设计成通过 HTTPS 传输，所以使用 web 组件必须按如下步骤进行：

1. 验证加密签名。因为令牌的形式是一个 JSON Web 令牌或者是 JWT，并且还有大多数流行的编程语言库来验证 JWT，这是简单而有效的。
2. 确保 `aud` 字段的值是属于你自己的客户 ID。

完成相关操作后， web 组件的令牌就有下列必然的特征：

1. 令牌是通过谷歌发布的。
2. 令牌被放置在 `email` 字段，然后发送到设备。

Web 组件可以通过安卓应用程序鉴定放置在 `azp` 字段的客户 ID。然而非兼容的或者高权限的安卓设备或许会伪造 `azp` 字段。

效果就是，Web 组件把来自安卓客户端的数据当做来自项目中的用户的数据，使用 ID 令牌来代替用户来访问和更新用户数据，而不用多次提醒用户进行身份验证。

#

安卓应用的 Web 端获得离线访问权限

再次假设一个包含了客户 ID 为 `9414861317621.apps.googleusercontent.com` 的 Web 组件，在这个事例中想代替用户访问两个不同的资源，但此时用户并不在线。然而，软件访问资源通常是通过一个安卓应用的交互来实现的。让我们更深入地假设，资源是被 Google Drive scope 内的字符串 `https://www.googleapis.com/auth/drive.file` 和 Google+ scope 内的登陆字符串 `https://www.googleapis.com/auth/plus.login` 标志的，我们将上述两个字符串标记为 `resource-1` 和 `resource-2` 。

在这个场景中，安卓应用就像是在开发者面板项目中的 web 组件那样，在设备上指定的任何的谷歌账户，都能够调用 GoogleAuthUtil.getToken() 方法，其 `scope` 参数的之为 `oauth2:server:client_id:9414861317621.apps.googleusercontent.com:api_scope:resource-1 resource-2.` 。

在这个案例中，用户已经授权这个项目访问前面所说的那两个范围， `GoogleAuthUtil.getToken()` 方法将首次请求那两个范围的令牌。如果他可以这么做，那么这个方法并不会返回 OAuth 令牌，但是会放回一个临时的授权令牌，用来更新访问令牌和刷新令牌。

安卓应用能通过 HTTPS 来发送授权码给自己的 web 组件。web 组件可以这个授权令牌返回访问令牌和刷新令牌，在 Handling the response 中有相关描述。以下规则适用于 web 组件:

- 当 web 组件通过授权码交换令牌时，在 `POST` 请求中不应该包含 `redirect_uri` 参数。
- 当 web 组件接收到令牌时，必须检查 ID 令牌。接受来自谷歌安全通道的令牌，但是请求中并不含有签名检查。所以 web 组件还必须作如下操作:
    - 核实来自谷歌的 ID 令牌中的 `aud` 字段，是否与 谷歌开发者面板 中的客户 ID 一致。
    - 核实来自谷歌的 ID 令牌中的 `sub` 字段，是否与从客户端接受到的 ID 令牌中的 `sub` 字段一致。
- 当刷新令牌还未过期时， web 组件的任务就是安全地保存刷新令牌，使其能够长期使用。
- 重要的是，安卓应用本身不会企图使用授权码来更换刷新令牌。这将需要应用存储服务端的客户 ID 和客户密钥。这会导致你的安卓应用有安全漏洞，因为应用会极其简单地被破解。

你能够在一个较短的时间内，大概一个钟，使用访问令牌来访问请求的范围。刷新令牌不会过期，除非被明确地取消，你可以使用刷新令牌随时随地地来获取新的访问令牌。

#

建议的流程

为了有效使用资源减少不必要的工作，当安卓程序通过自己的后端开启一个网络会话时，我们推荐如下的程序流程：

1. 所有的请求都使用 HTTP 的 `POST` 请求，每个 `POST` 请求主体都要包含 ID 令牌。你获取的是在之前章节描述的 ID 令牌，要告知给用户的后端。
2. 会话的首次请求，不论它是否拥有在线访问必要范围的适当等级，通常都会有查询后端对其进行校验。
3. web 后端必须知道他是否拥有一个有效的刷新令牌。如果它没有刷新令牌或者发现了它的刷新令牌已经失效（你可以使用刷新令牌来获取新的访问令牌来测试其是否失效），web 后端就会传递缺少刷新令牌的信息给客户端。
4. 如果客户端接收到了来自后端的信息，得知了后端没有有效的刷新令牌。在安卓应用获取了授权令牌，就像前面说的那样，会将这个授权令牌传送回 web 后端来获取有效的刷新令牌。
5. 客户端和服务器继续他们之间的会话。

在这些步骤中，控制请求离线访问的功能是通过服务端的代码来维护的，最好能够知道什么时候需要离线访问。并且，如果应用总是请求刷新令牌，却不去检查服务端是否已经拥有了刷新令牌，这将导致有限的刷新令牌被消耗，这反倒可以突出用户/应用程序组合是优秀的。

上诉效果就是，用户只要经过少数的确认，就可以让项目的移动应用的组件，为其他组件取得永久的访问离线资源的权限.

8

谷歌应用程序默认凭证

为调用谷歌应用接口，谷歌应用程序的默认凭证提供了一个简单途径来获取授权证书。

他们有最适合这些事例的解决方案，当应用程序独立于用户去调用接口时，需要拥有相同的的身份标识和授权等级。这里推荐授予应用调用谷歌云应用接口的权限，特别是在你要发布到谷歌应用引擎或者谷歌计算引擎虚拟机上的应用程序时。

# #

内容

#

何时使用应用程序的默认凭证

出现下列任一情形时，我们推荐你使用应用程序的默认凭证：

- 你的代码运行在谷歌应用引擎或者谷歌计算引擎上。当部署应用程序时，其默认凭证提供了访问内置服务账号的权限，但是也提供了部署之前，测试应用程序时可供选择的凭证。
- 为避免在应用程序代码中嵌入的鉴定信息。通常一个最佳使用方式是，以避免包含密钥认证的相关源代码，如测试和生产时，需要在不同的上下文中使用不同的凭证。使用环境变量，在应用程序外可以定义凭证。
- 正在访问的应用接口，其中的数据是关联到一个云项目或整个应用程序的，而不是个人用户数据。所以最好使用一个认证流程，最终用户明确同意访问（查看 使用 OAuth 2.0 访问谷歌应用接口）。

#

应用程序的默认凭证如何工作

通过调用客户端代码库，你可以获取到应用程序的默认凭证。返回的凭证是由环境中运行的代码决定的。按照以下顺序进行检查:

1. 检测环境变量 GOOGLE_APPLICATION_CREDENTIALS。如果环境变量是规定的，它应该指明了定义凭证的文件。通过这个简单的方法可以获取到凭证，其目的是使用 谷歌开发者面板 的 APIs & Auth 选项中的子选项 Credentials 来创建一个服务帐户。创建一个服务账号或者选择一个现存的，选中 Generate new JSON key 选项。为下载的 JSON 文件设置环境变量。
2. 如果你在你的设备上安装了谷歌云 SDK，需要运行命令 `gcloud auth login`，你的身份可以用作代理，从那台机器测试代码调用应用接口。
3. 如果你的应用程序时运行在谷歌应用引擎上的，内置的服务账号要关联到应用程序。
4. 如果你的应用程序时运行在谷歌计算引擎上的，内建的服务账号必须关联到虚拟机。

5. 如果不遵循上述约束，就会发生错误。

#

在应用程序代码中调用应用程序的默认凭证

默认凭证被集成在谷歌应用接口客户端库中。支持的环境有安装在 Linux，Windows 和 Mac OS 上的应用，以及 GCE（Google Compute Engine ）虚拟机。

#

Java

版本为 1.19 的 谷歌应用接口客户端库 Java 版 中集成了默认凭证。

```
import com.google.api.client.googleapis.auth.oauth2.GoogleCredential;

...

GoogleCredential credential = GoogleCredential.getApplicationDefault();
```

如下，它可以用来访问一个应用接口服务:

```
Compute compute = new Compute.Builder
    (transport, jsonFactory, credential).build();
```

一些凭证类型向你请求某些 scopes，但是服务实体入口点并不受理。如果你偶尔遭遇了这个情节，就像下面所做的那样，你需要注入 scopes:

```
Collection COMPUTE_SCOPES =
    Collections.singletonList(ComputeScopes.COMPUTE);
if (credential.createScopedRequired()) {
    credential = credential.createScoped(COMPUTE_SCOPES);
}
```

参考：谷歌应用接口客户端库 Java 版文档.

#

Python

版本为 1.3 的 谷歌应用接口客户端库 Python 版 中集成了默认凭证。

```
from oauth2client.client import GoogleCredentials
credentials = GoogleCredentials.get_application_default()
```

可以这样使用服务:

```
from googleapiclient.discovery import build

...

service = build('compute', 'v1', credentials=credentials)
```

`build()` 方法在给定的服务内留意了注入的适当范围，虽然 `create_scoped` 方法可以显式地做到这一点。

参考：谷歌应用接口客户端库 Python 版文档.

#

工具支持

#

谷歌云 SDK

自谷歌云 SDK 0.9.51 以来，当在本地测试应用程序代码时，使用命令 `gcloud auth login` ，可以让你的身份凭证用来当做应用程序默认凭证的代理。

这是最快速便捷地在本地校验代码的方法。有一点需要注意，你的个人身份凭证通常有很多的权限，所以你该找一个场景，这个场景是一个应用接口在本地被调用而不是在发布之后被调用。它只能在谷歌云应用接口许可范围内工作。如果这个方法引发了问题，考虑下载服务账号密钥来设置环境变量 environment variable 。

#

谷歌应用引擎 SDK

自谷歌应用引擎SDK 1.9.18 以来，当在本地运行开发用应用程序服务，可以使用应用程序的默认凭证。注意，这个仅仅支持通过使用命令 `gcloud preview app run` 来完成上诉效果，不支持在老旧的特定语言的开发服务器。

#

发现并修理故障

在运行环境的上下文中，应用程序的默认凭证允许你使用不同的身份凭证。这有时意味着需要调整来确保所有正在使用的身份拥有权限。

#

权限

使用应用程序的默认凭证的代码，可以像用户或者服务账号那样的身份凭证运行，包括内置的服务账号的身份凭证。如果使用了一个以上的身份凭证，他们必须都有权限调用。配置权限，请打开谷歌开发者面板的 Permissions 选项。

#

域（ Scopes ）

他们采取 URI 的形式，名字一组给定的 API 的功能，OAuth2 的权限 scope 的定义是在一个给定的上下文中，可被调用的应用接口。他们采取 URI 的形式，为给定的应用接口命名，例如，"https://www.googleapis.com/auth/compute.readonly"。不同类型的身份凭证处理不同的 scope。

下载的服务账号的密钥和谷歌应用引擎内建的服务账号，其 scope 必须在代码中指定。应用接口的封装或许该这么做，但是在使用这个证书时，会出错。获取更多信息如何在你的代码中注入 scope，请查看 Calling the Application Default Credentials in application code。

谷歌计算引擎虚拟机的服务账号，其支持的范围必须在虚拟机创建时指定。如果使用谷歌云 SDK ，就要使用命令 `gcloud compute instances create` 以及指定 `--scopes` 参数来完成。

如果使用谷歌云 SDK 的命令 `gcloud auth login` ，在本地提供自己的身份凭证，scope 就会被固定，即使 scope 包含了所有的谷歌云应用接口的范围。如果你需要不在这里谈及的 scope，建议你下载服务中的密钥。