适用范围

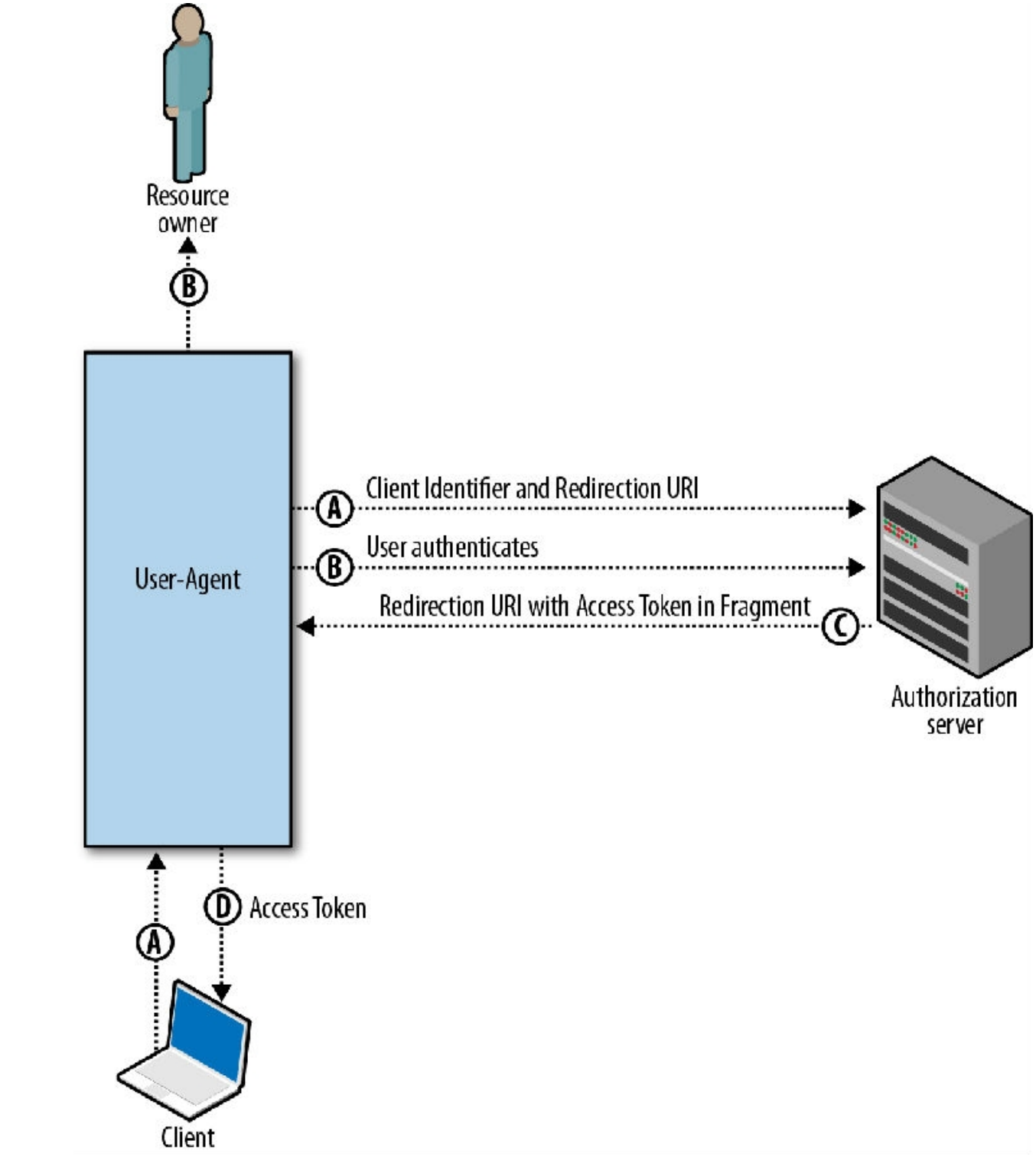
仅需临时访问的场景

用户会定期在API提供者那里进行登录

OAuth客户端运行在浏览器中（Javascript、Flash等）

浏览器绝对可信，因为该类型可能会将访问令牌泄露给恶意用户或应用程序

流程剖析



1. 让用户明白所做的操作并请求认证

这一步与授权码认证模式中的操作类似，即当牵涉到OAuth认证时，应首先让用户明确该操作。然后将用户引导至授权页面。
该授权接口的URL会在开发者文档中给出，以谷歌为例：

https://accounts.google.com/o/oauth2/auth

在请求该页面时还需附带几个参数：

client_id

在应用注册时提供

redirect_uri

授权认证后的重定向地址

scope

应用所请求访问的数据，一般由空格分隔的多个字符串组成

response_type

对于此授权类型来说为“token”，即在授权成功后会返回access token

2. 从URL中解析access token

当授权顺利完成后，用户会被重定向到redirect_uri中指定的URL并附带access_token等重要数据，它们会包含在url hash中，例如：

http://example.com/callback#access_token=ya29GAHES6ZSzX&token_type=Bearer&expires_in=3600

JavaScript是不会自动将解析hash段中的元素解析成键/值对，因此我们需要手动进行该操作：

```
var oauthParams = {};
// parse the query string
// from http://oauthssodemo.appspot.com/step/2
var params = {}, queryString = location.hash.substring(1), regex = /([^&=]+)=([^&]*)/g, m;
while (m = regex.exec(queryString)) {
    oauthParams[decodeURIComponent(m[1])] = decodeURIComponent(m[2]);
}
...
```

3. 访问API

拿到了access_token，API接口就向你敞开了大门，接下来的操作我想就无需阐述了。

从整个流程可以看出，相比授权码授权，隐式授权少了第一步获取Authorization Code的过程，因此变得更为简单。但正因为如此也降低了安全性。