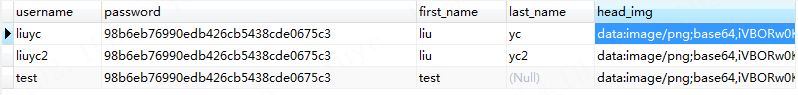
服务器地址 <http://192.168.5.159/oauth2-server/>

客户端地址 <http://192.168.5.159/oauth2-client/>

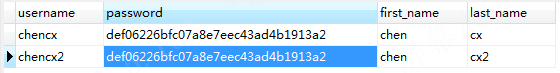
服务端：登录、查看应用、增删改应用，最重要的内容是Scope（权限）、Client\_id、Client\_secret，目前有三个用户：

密码是123456



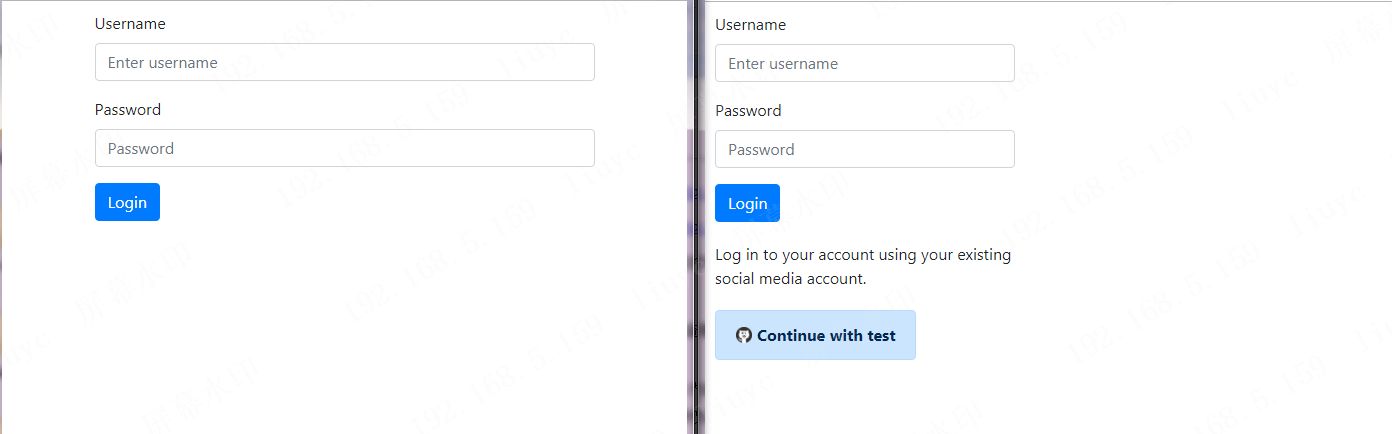
客户端：登录（内部用户登录，使用其他账户登录），获取用户详细信息（如果是使用其他账户登录，需要权限），目前有两个用户：

密码是123456



**图文说明：（左为服务端，右为客户（第三方）端）**

1. 一开始，两边都没有登录：



1. 然后客户端先测试内部登录：

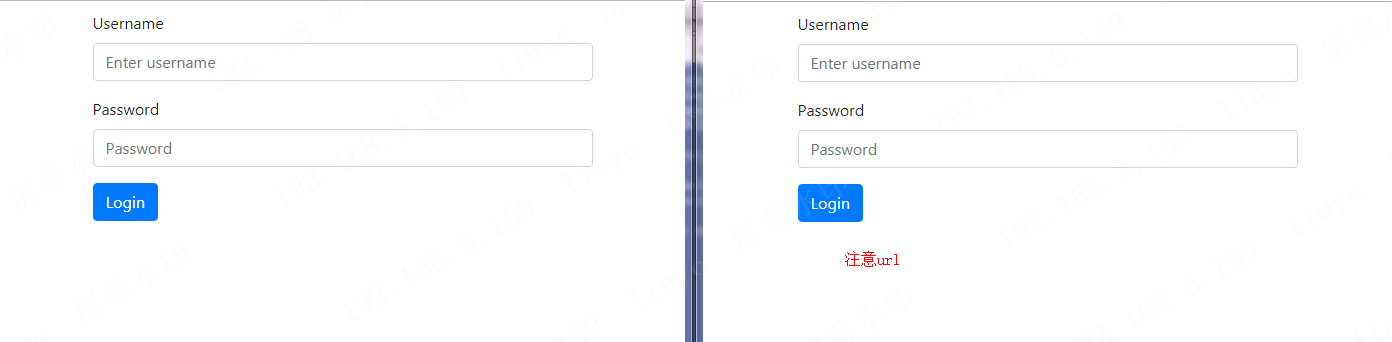
账户 密码 chencx 123456



1. 登录成功后可以获取用户详细信息，而客户端的登录不会影响到服务端的

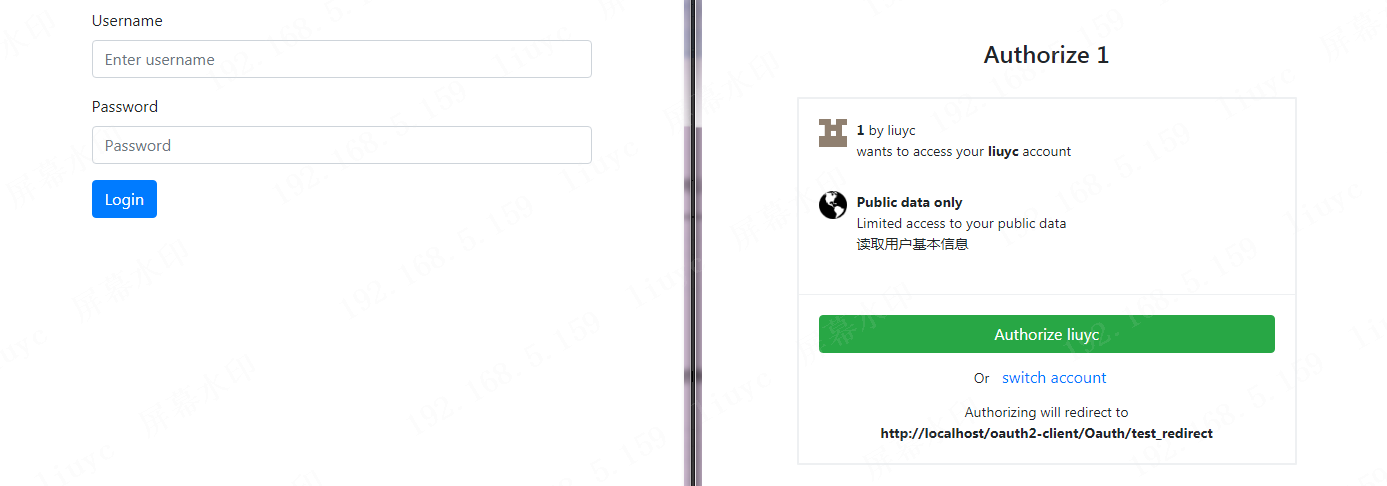


1. 退出客户端的账户，点击continue with test，意思是使用服务端的账户来登录



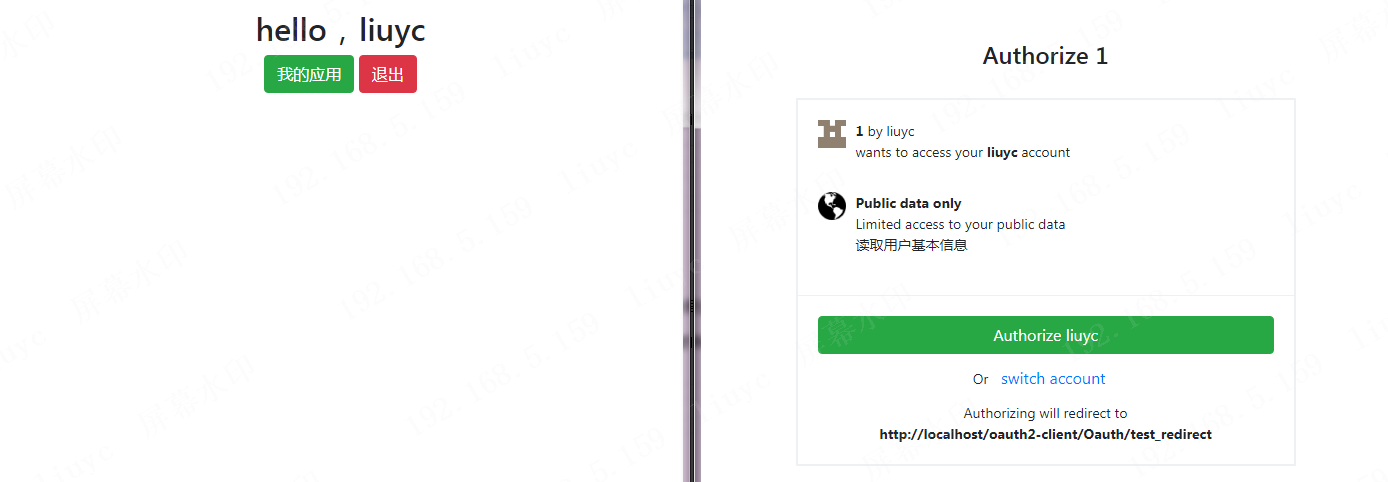
右边客户端显示需要登录到服务端，这个时候url是带有client\_id这类的参数的

1. 使用服务端的账户 liuyc 123456 登录

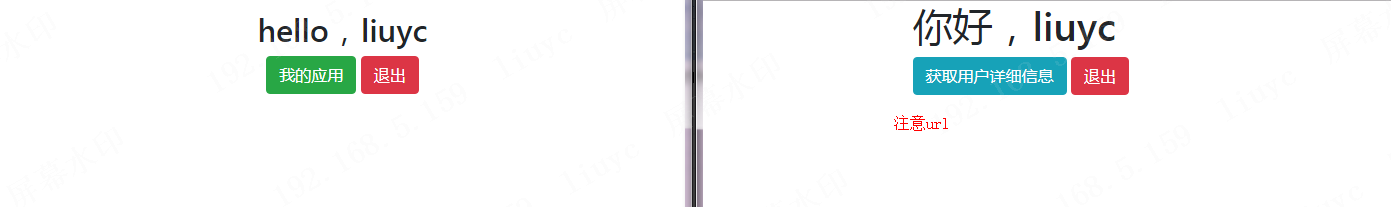


可以看到授权页面，包含应用的信息，应用拥有的权限，当前要授权的用户

1. 这个时候，可以刷新一下左边的浏览器，会进入了主页，登录的用户就是刚才在右边浏览器的登录用户



1. 点击授权，右边进入到客户端主页



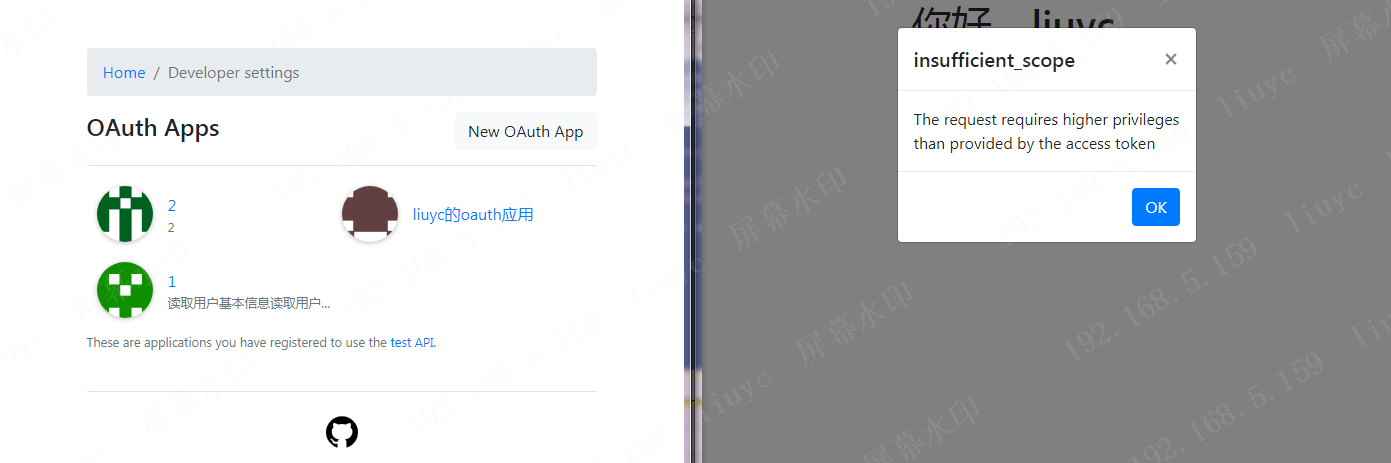
这个时候，注意右边的url，是客户端的地址，但使用的是服务端的账户来的登录，客户端是没有liuyc这个用户的

1. 点击一下获取用户的详细信息，报了错误

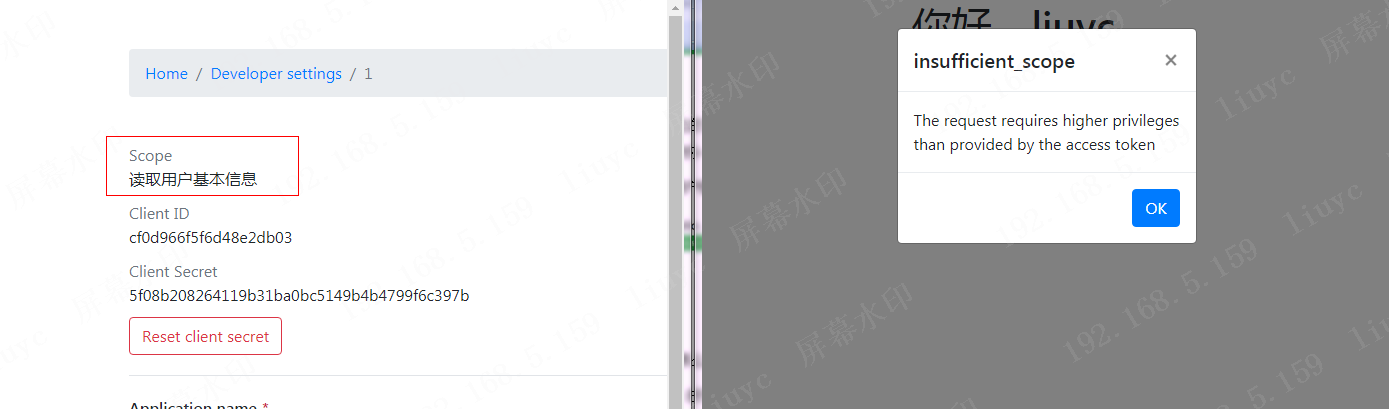


意思是权限不足，那么我们前往服务端看看应用的权限

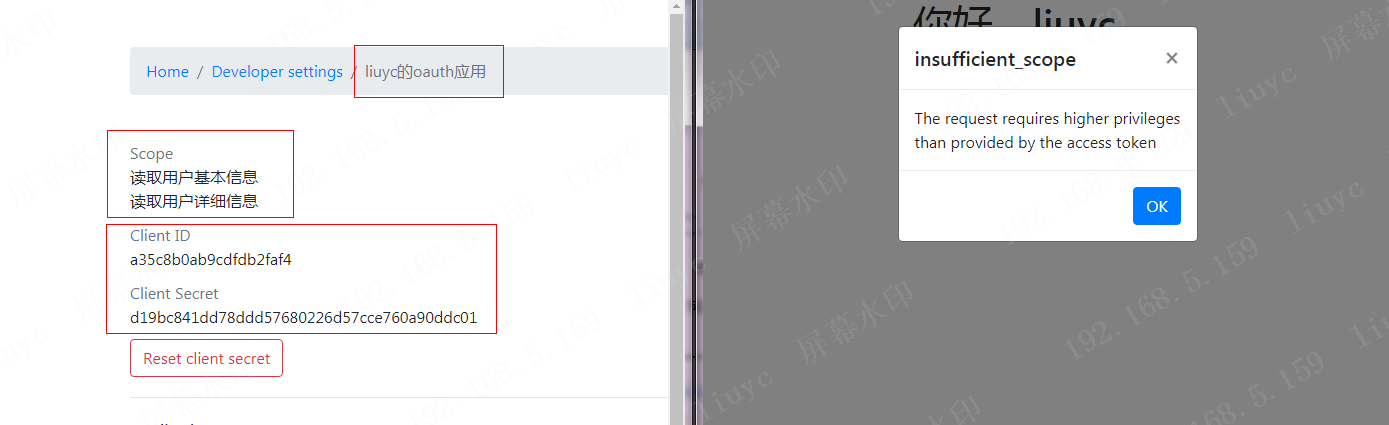
1. 点击我的应用，进入应用列表，这里多个应用没关系，我们找回上图授权页面，授权的是1的应用



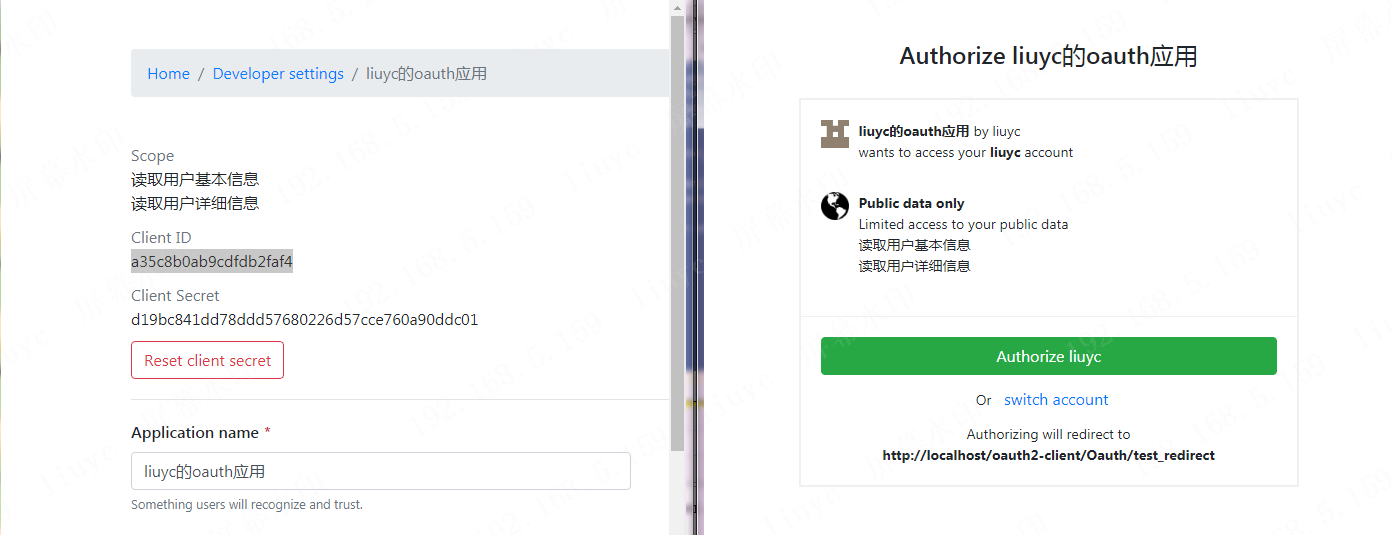
1. 进入1应用，可以看到该应用的权限只有读取基本信息，所以不能读取用户的详细信息



1. 在客户端中修改授权的应用，目前只能代码修改，把client\_id和client\_secret改为“liuyc的oauth应用”的值，还需要修改scope的值包含userdetail，否则验证不通过



1. 退出客户端的登录，再次点击continue with test



不需要再次登录的服务端，因为已经登录过了，所以直接显示授权页面，可以看到授权页面的信息显示的是“liuyc的oauth应用”，而且权限具有读取详细信息。

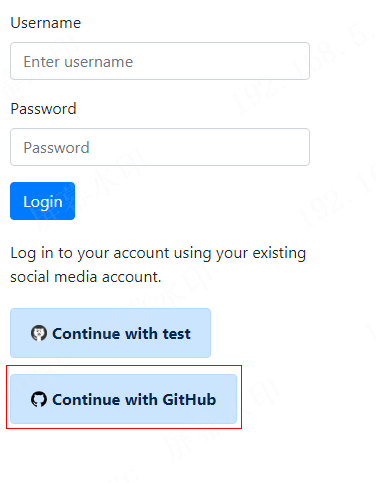
1. 点击授权，进入主页，并且能获取到详细信息



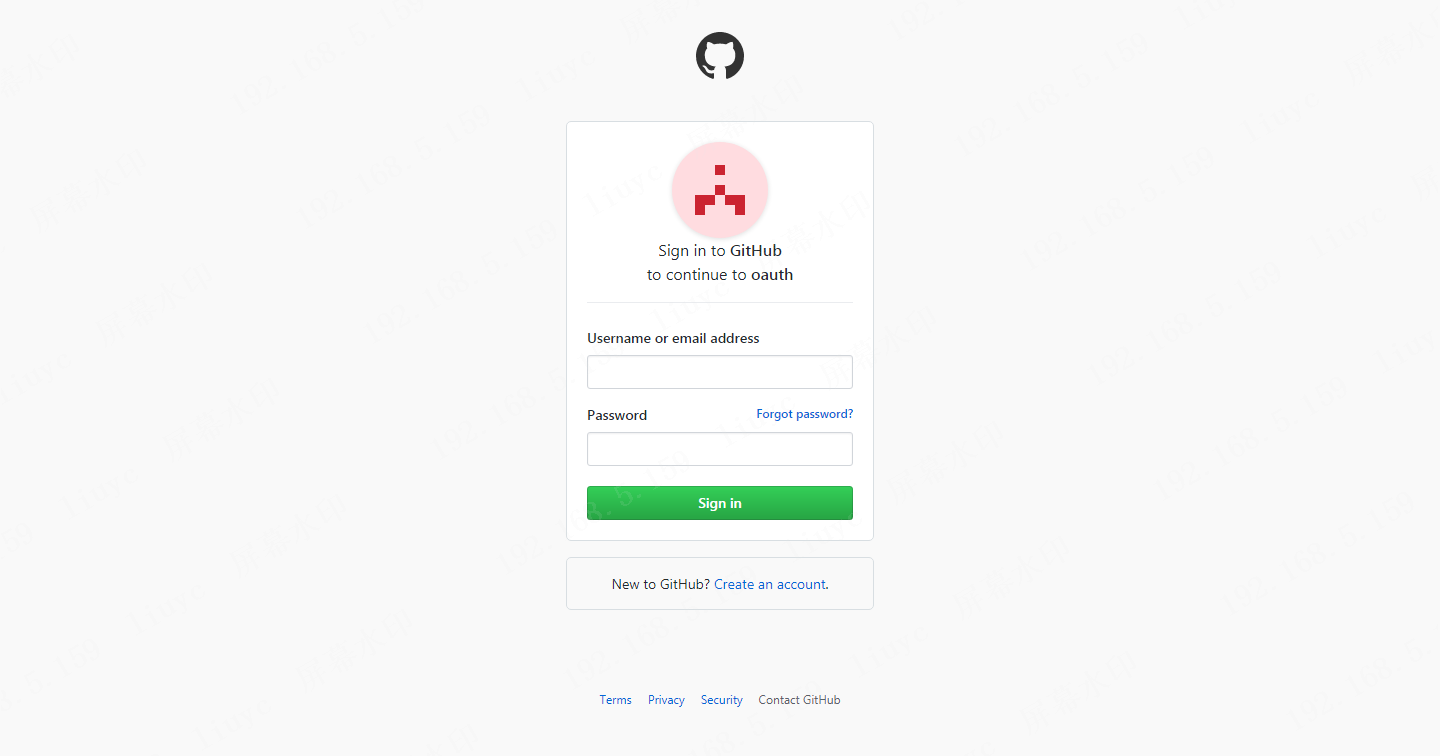
1. 其他的功能这里不作说明了，例如授权页面中的切换账号，应用的创建、修改、client\_secret的修改、api的查看（<http://192.168.5.159/oauth2-server/test>）

**真实例子：**

1、点击使用github账户登录这个demo的客户端



2、如果这个时候，你的github还没有登录，要先登录到github，它才知道是哪个用户



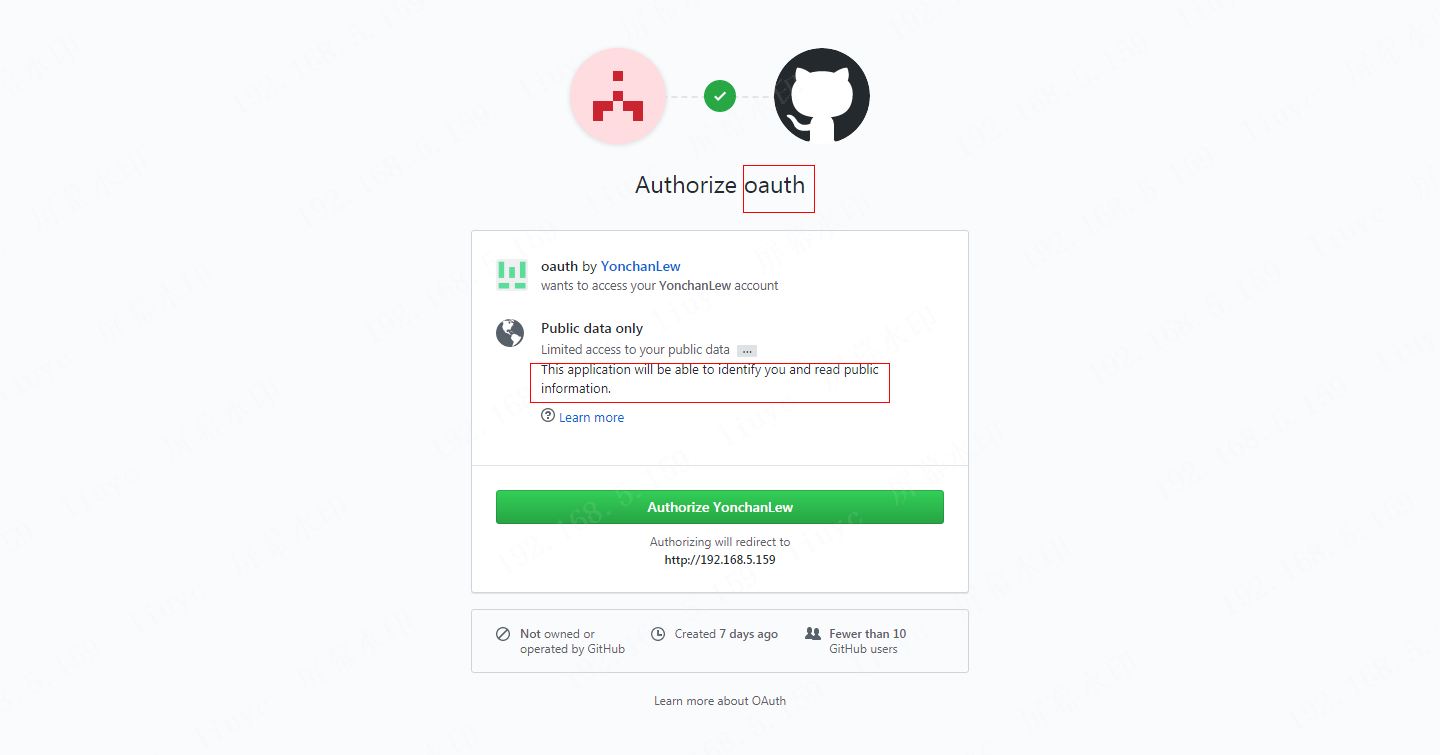
3、然后显示授权界面，让用户确定，是否允许oauth这个应用读取你的信息，这个时候，如果你不信任oauth这个应用，就不要授权。这里要搞清楚4个东西：

demo系统客户端：就是第一张图的登录界面，相对github来说是一个第三方的系统；

github服务端：拥有用户信息的服务端；

oauth应用：demo客户端开发者在github上注册的应用，中间媒介

github用户：用户本体

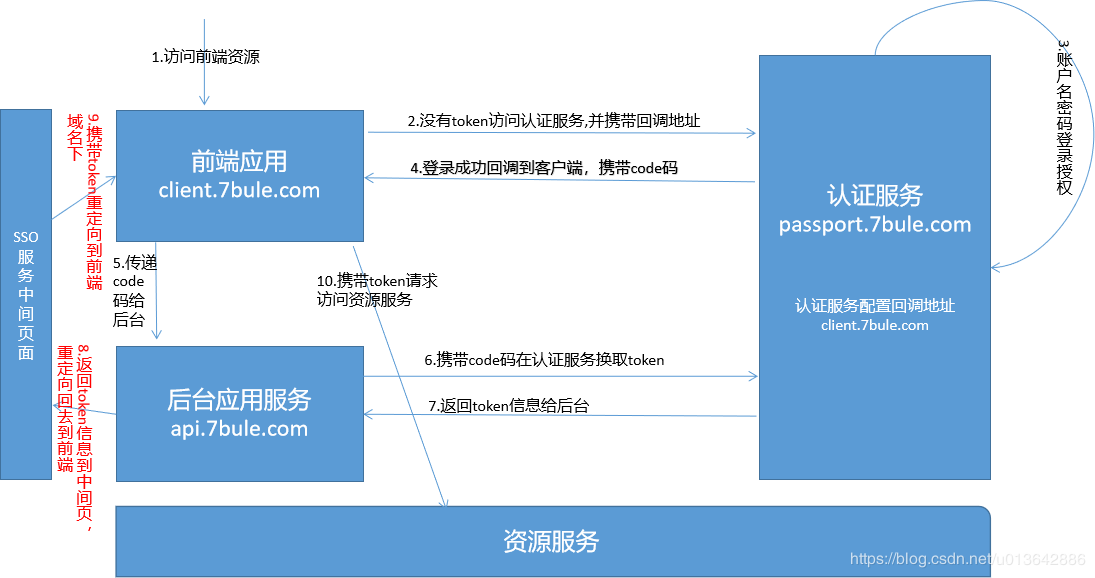


而这个授权界面的必要性就是，当一个用户在使用demo客户端的时候，嫌麻烦不想在demo客户端上注册一个账号，为了方便，使用github来登录demo客户端，然后显示这个授权页面，看到这个oauth的应用会读取你的公开信息，如果这个时候，用户并不希望透露github上的公开信息给demo客户端，那就不能授权，应该返回登录页面自己注册一个账号进入系统，或者是使用test服务器的账号来登录。

4、授权成功之后，demo客户端就会收到github提供的access\_token，那么demo客户端就可以使用access\_token来调用github的api获取用户的公开信息



单点登录：



要单点登录，那么所有的客户端都必须是使用test认证，没有了内部登录功能，那么登录界面可以省去，每次进入系统A之前，都判断是否已登录，如果未登录就相当于加载授权登录的a标签，然后test服务器判断是否已登录，如果未登录就显示登录页面，是test提供的，登录了之后，显示授权页面

先（注册一个用户，现在没有注册）登录server，新建一个应用，得到client\_id和client\_secret，然后开发者开发自己的系统，为了自己的系统能使用server的账户来登录，就要对接server的oauth接口，现在接口使用密码模式，即第三方系统通过server的账号用户名和密码加上client\_id和client\_secret来验证，通过之后返回用户信息和token信息，就进入第三方系统，第三方系统后续的请求都要请求server判断session还是否存在来判断用户是否已登录，这个判断的请求同时都返回token信息（如果不想每次都请求，网上推荐最多的是使用redis来共享session）。

进入第三方系统之后，进行系统自己的各种操作，基本上操作都是与用户相关，所以第三方系统在登录了之后，应该在自己的数据库上新增一个用户，自己生成uuid，每次使用server账户登录之后，能够根据server返回的用户信息对应自己系统数据库的用户，从而对应自己系统其他的数据表。

接着第三方系统需要请求server的资源时（如果第三方系统并不需要server的资源，那么token信息可以无视，只需要在退出的时候调用server接口删除一下token保障安全就可以了），需要提交access\_token，第三方要保证access\_token的有效性，所以在请求server资源的时候，应该先调用判断access\_token是否还有效的接口（这里修改了server的返回值，直接返回到期时间，这样通过时间判断就知道是否过期了），返回有效/无效，如果无效，就再使用refresh\_token调用获取新access\_token接口，如果refresh\_token有效，返回新的token信息，server那边也要更新token信息，如果refresh\_token无效，就返回重新登录的信息（当access\_token和refresh\_token都无效时，就只有重新授权才能获取token了）

上面一直描述的都是oauth的过程，其实已经包含了sso，在对接server的oauth接口后，当登录是成功的，server就会保存session信息，当另一个第三方系统（域相同）刷新，请求server的判断session是否存在的接口（上方蓝色文字），如果返回不是空，就是已经登录了，未登录的时候是返回null，这样就能实现单点登录了。

退出的时候，删除session和token信息，这样第三方系统刷新调用蓝色文字的接口，就会返回null，这时回到登录页面就实现退出系统了

无论是oauth还是sso，网上说的例子都是登录页面都应该是服务器提供，然后通过参数回调到原来的第三方系统，这里我并没有这样做，我使用了密码模式，即需要用户高度信任第三方系统，能把账号密码发送给第三方系统的方式，第三方系统使用发过来的账户密码进行登录得到token。