**流程：**

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。

**问题：**

在登录之后，会把token信息保存在数据库中，绑定了client\_id，这样别的系统由于client\_id不同，导致无法执行refresh token，如果所有系统都能调用refresh token，那他们提交的参数必须和登录时候的client\_id和client\_secret一致，但这样就失去了client\_id和client\_secret存在的意义

例如server的client\_id是aaa，登录之后，access\_token绑定了aaa，那么其他系统在调用refresh token的时候，要把aaa提交上去，才能刷新，否则提示refresh\_token doesn't exist or is invalid for the client。

思路：

客户端a，client\_id:aaa,client\_secret:aaa

客户端b，client\_id:bbb,client\_secret:bbb

a登录之后，把(aaa,aaa)保存，b刷新，同时能获取到(aaa,aaa)，或者是某个接口来获取，然后刷新token的时候，把(aaa,aaa)提交上去而不是bbb，这样才能刷新。但这样通过接口的方法就有可能暴露出去。

或者把refresh token改为不验证client\_id，client\_secret，这样的话别人随便提交refresh\_token都能刷新access\_token，但前提是refresh\_token存在在数据库中，所以也不是随便能改的。

第一种方法是比较正常的，因为你的access\_token就是由(aaa,aaa)来生成的，然后要修改token，当然是要提交aaa的数据。然后关闭创建oauth app的功能，由始至终仅有一个client\_id和client\_secret，需要单点登录的系统都填写这个密钥，这样不管哪个系统都具有相同的权限，因为权限时与client\_id绑定在一起的。所以，如果按之前各个系统client\_id都不一样的情况，例如a有2个权限，b有1个权限，当a登录之后，b获取了a的access\_token，就拥有了2个权限，这也是不正确的，所以改为只有一个client\_id，没有注册的页面了。

上面的过程所有系统的都是同一个开发者，不然无法保证client\_id等信息一致

另一种处理思路（未验证是否可行）：

如果授权服务器和对接的系统不是相同的开发者（不同公司），假如有一个授权系统resource（也叫资源系统，当然可以是分开的两个系统），然后开发者前往resource上注册一个账号，并申请一个oauth应用，得到client\_id、client\_secret、scope等信息。

然后开发者就开发自己的系统，假如开发者准备开发一个叫订餐系统、一个叫请假系统，然后为了系统间能单点登录进行访问，那么两个系统都应该接入相同的oauth应用，也就是都具有相同的oauth密钥。不然订餐系统得到的access\_token在请假系统中无法使用。

假如resource部署在192.168.5.159上，订餐系统部署在192.168.5.160，请假系统部署在192.168.5.161上。

进入订餐系统，先提交client\_id到resource上，resource根据cookie、client\_id读取session数据，如果cookie、session不存在的，返回某个状态码如F001，代表未登录，订餐系统收到F001就location到resource的登录页面，必定是登录页面，不然就不会返回F001，登录之后，显示授权页面，resource需要在数据库中多一个表或者一些字段，用来记录用户是否授权了这个client\_id。然后我们先不点确认授权，也就是resource的数据库中user还没对client\_id授权，但resource是已经登录了，所以这个时候我们刷新resource应该要进入主页，同时在刷新请假系统的时候，应该要显示授权页面（请假系统发送client\_id到resource上，resource读取session，已经登录了，然后判断登录的用户是否已对client\_id授权，不是，返回F002，请假系统收到F002就location到resource，本来是location到登录页面，但由于已经登录了，所以应该显示授权页面）。然后回到订餐系统，点击授权，resource中建立起user和client\_id的关系，请假系统也刷新，