在线实验，请到PC端体验

OAuth 2.0 授权原理与实战

一、课程介绍

1. 内容简介

OAuth 2.0 是时下最流行的授权认证方式，其典型的应用有第三方账号登陆，获取认证开发第三方应用等。本课程的前半段主要讲解 OAuth 2.0 授权的原理。后半段则会基于 Flask 与 Github 授权接口实现一个第三方留言本应用。

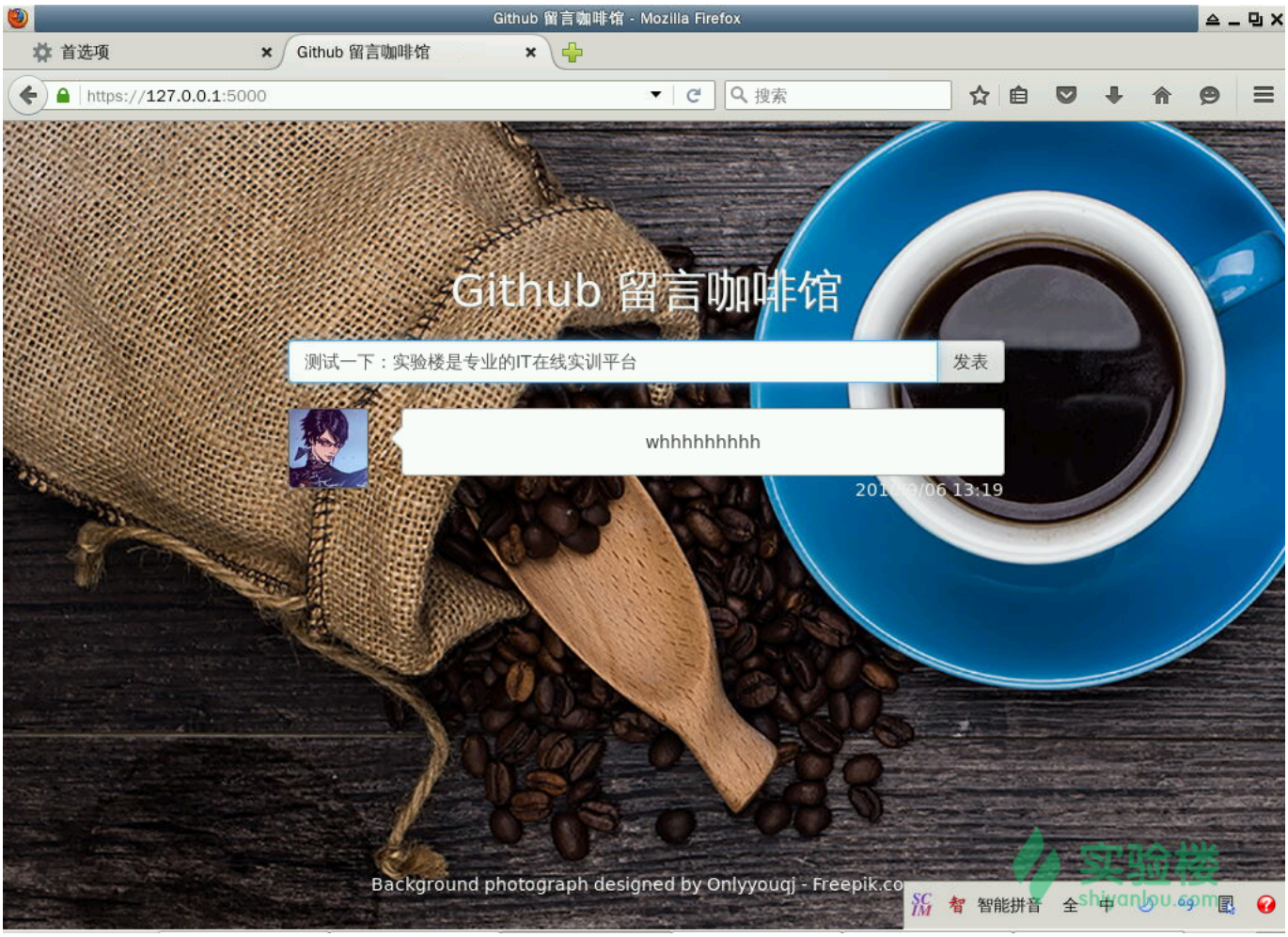
2. 课程知识点

本课程项目完成过程中，我们将学习：

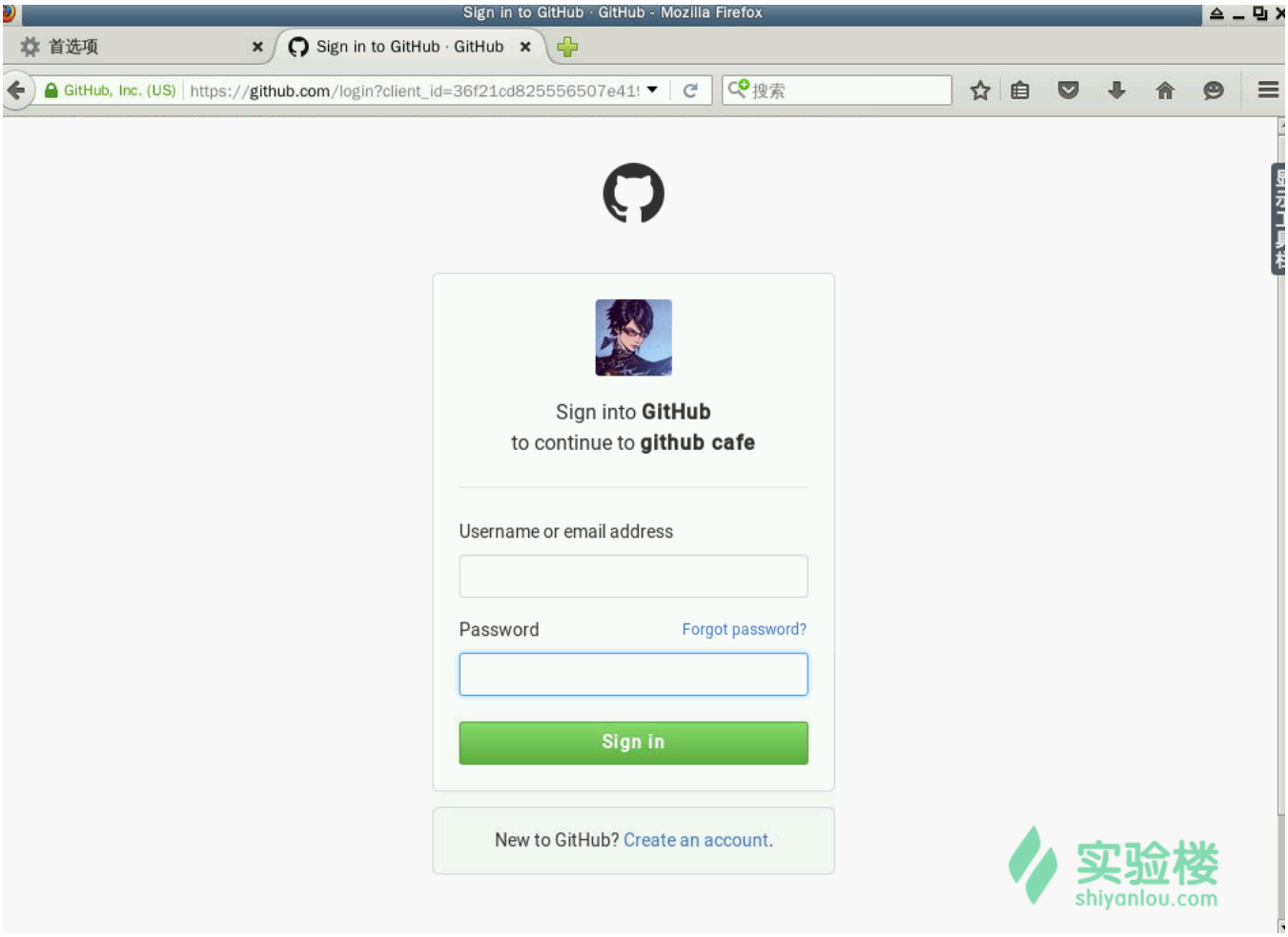
1. OAuth 2.0 授权认证的流程
2. OAuth 2.0 授权代码的编写
3. 基于 flask + mongodb 实现一个简单的第三方留言本应用

3. 效果图

清空缓存后尝试发布一条新留言：



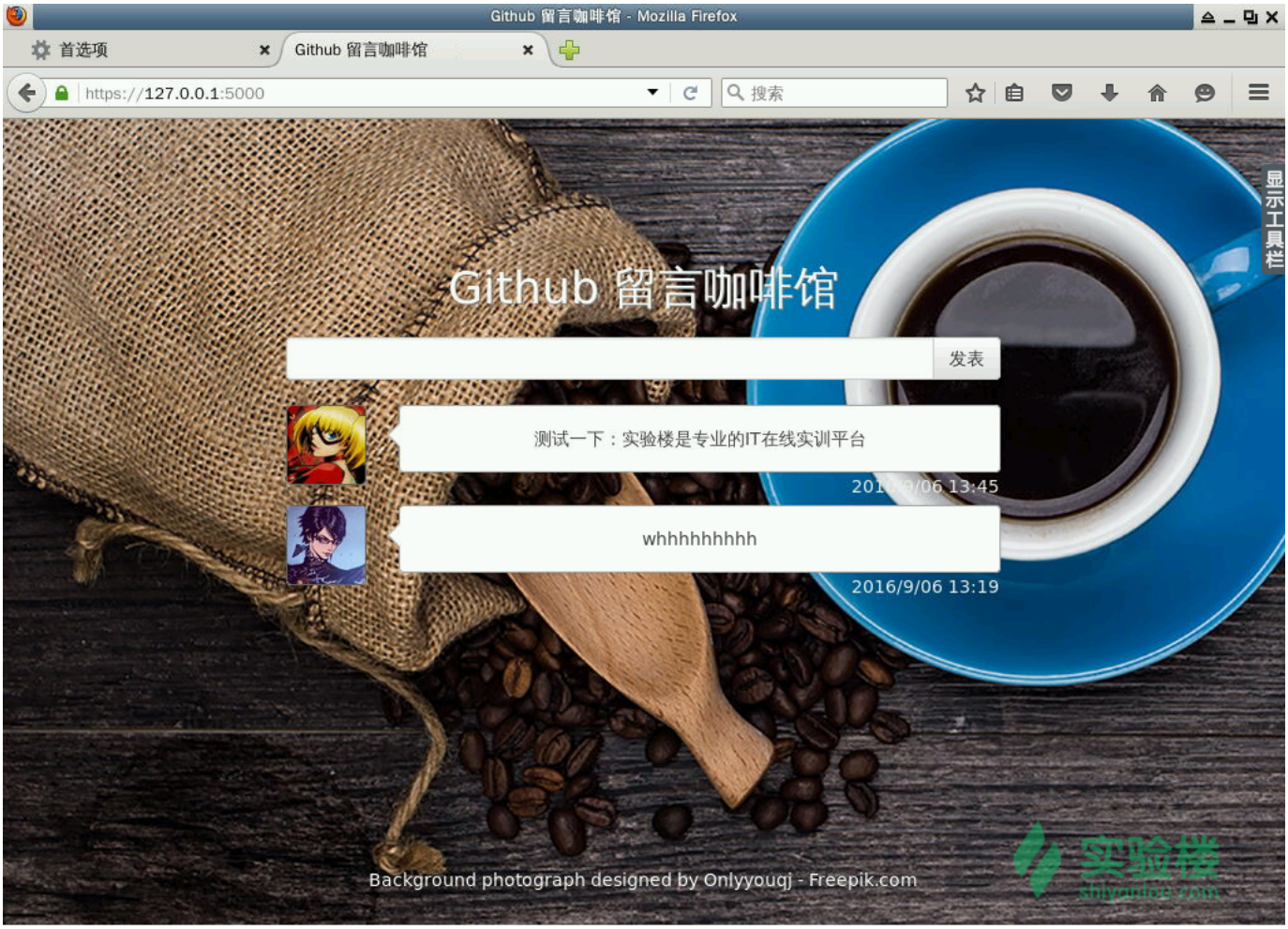
点击发表后跳转到授权页：



授权页的图其实是应用的logo，没有设置应用logo的话就会使用应用作者的头像。

发表后：

我用另一个账号发表了一条留言。



二、实验环境

打开终端，进入 Code 目录，创建 oauth2.0 文件夹, 并将其作为我们的工作目录。

```
$ cd Code
$ mkdir oauth2.0 && cd oauth2.0
```

三、OAuth 2.0 授权原理

1. 什么是 OAuth 2.0

OAuth 2.0 目前广泛使用的授权认证协议，它使得第三方应用能够在不必得到用户账号密码的情况下通过申请授权（也就是取得用户和服务商的同意）来访问到用户的私人资源。当第三方应用申请授权成功时，它会得到一个访问资源用的 token（以下简称访问 token），之后便可使用访问 token 来获取资源了。

2. OAuth 2.0 的四种授权类型

OAuth 2.0 有四种授权类型：

1. 授权码授权（适用于 Web 应用）
2. 隐式授权 （适用于移动应用）
3. 用户密码授权 （不推荐使用）
4. 客户端授权 (适用于后端应用)

接下来从简单到复杂依次讲解这四种授权类型。

客户端授权

客户端授权类型是四种授权类型里流程最简单的。因为这种类型的目的并不是为了获取用户的私人资源，而是客户端自身希望通过授权得到访问 token 来访问服务提供者提供的服务接口。

认证流程如下图（参考：https://tools.ietf.org/html/rfc6749#section-4.4 (https://tools.ietf.org/html/rfc6749#section-4.4)）：

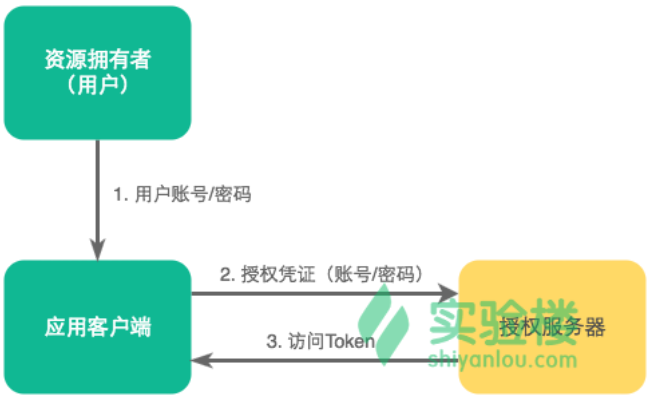
1. 客户端发送授权用的凭证给执行授权的服务器
2. 授权服务器返回访问 token 给客户端

授权认证所需的凭证一般就是一个 `client_id` 和一个 `client_key`，你在服务提供者（各类开放平台）那里申请创建应用时就会得到，它提供给你的 id 和 key 也可能会叫 API Key，Secret Key 等，这就需要你阅读该平台的文档进行确认了。

用户密码授权

用户密码授权类型需要用户提供账号密码给第三方应用。这种方式过于邪恶所以不推荐使用。

认证流程如下图（参考：https://tools.ietf.org/html/rfc6749#section-4.3 (https://tools.ietf.org/html/rfc6749#section-4.3)）：



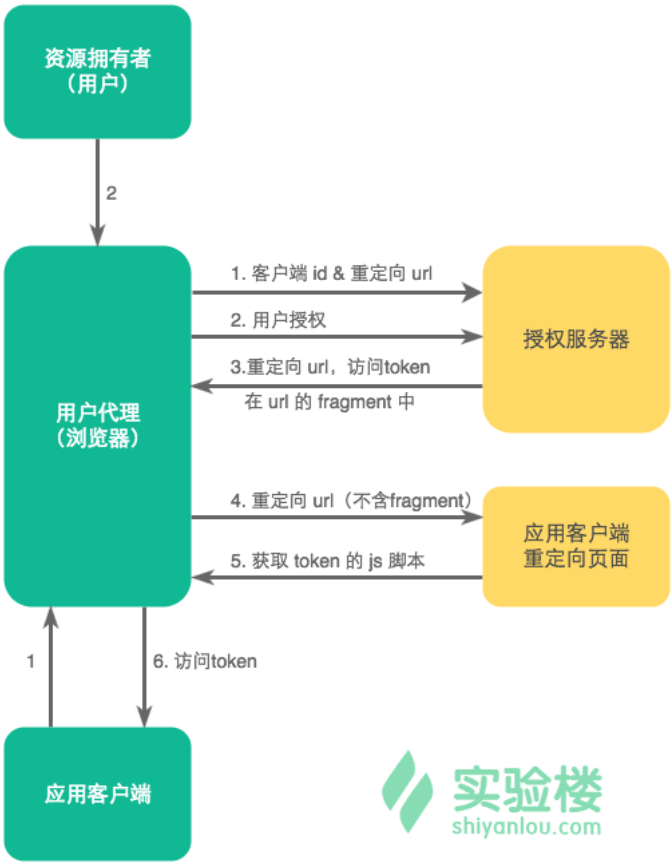
1. 客户端从用户那里得到用户名和用户密码
2. 客户端发送授权用的凭证给授权服务器
3. 授权服务器返回访问 token 给客户端

这种方式下的认证凭证就是用户名与用户密码。

隐式授权

隐式授权可以理解为简化版的授权码授权，它通过浏览器将用户导向授权页面，用户授权后就会重定向到你指定的页面URI。授权服务器会在重定向的URI的最后接上一个包含了访问 token 的 fragment，fragment 举个例子就是 `http://www.example.org/foo.html#bar` 里的 `#bar`，最后你通过重定向后的页面里的 js 脚本拿到这个 token 再传给应用客户端。

认证流程如下图（参考：https://tools.ietf.org/html/rfc6749#section-4.2 (https://tools.ietf.org/html/rfc6749#section-4.2)）：



（User-Agent 大多指浏览器，Client 指应用客户端）

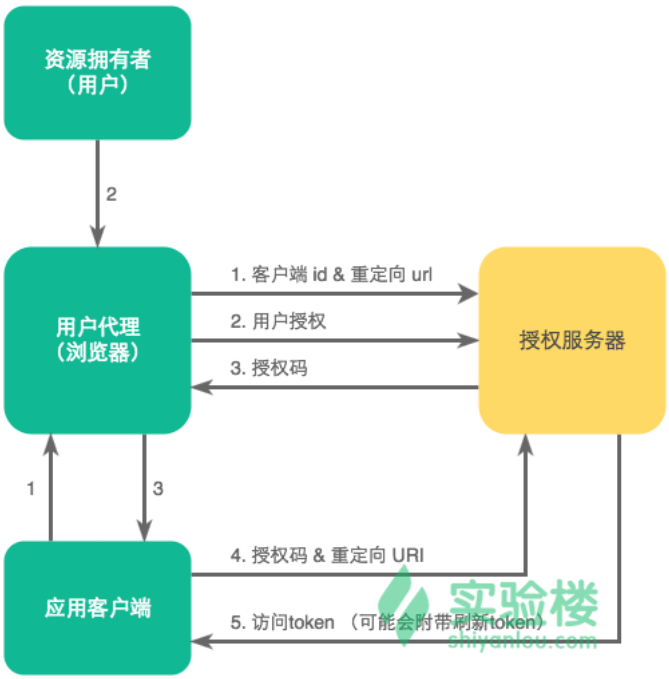
（这里需要说明一下：一般我们说后台是服务器，浏览器是客户端。但是实际上应该这么理解：享受服务的那一方叫客户端，提供服务的那一方叫服务端）

1. 应用客户端通过浏览器将用户导向授权页面
2. 用户授权
3. 授权服务器返回重定向 URI，并在 URI 上携带访问 token
4. 浏览器访问重定向 URI（不包含访问 token，但实际上 token 已经保存在浏览器本地了）
5. 通过页面的 js 脚本获取访问 token
6. 将访问 token 传给应用客户端

授权码授权

授权码授权需要先获得授权码再获取访问 token。

认证流程如下图（参考：https://tools.ietf.org/html/rfc6749#section-4.1 (https://tools.ietf.org/html/rfc6749#section-4.1)）：



1. 应用客户端通过浏览器将用户导向授权页面
2. 用户授权
3. 授权服务器返回授权码
4. 应用客户端使用授权码作为授权凭证，重定向 URI 作为辅助验证向授权服务器请求访问 token
5. 授权服务器返回访问 token （可能附带刷新token）给应用客户端

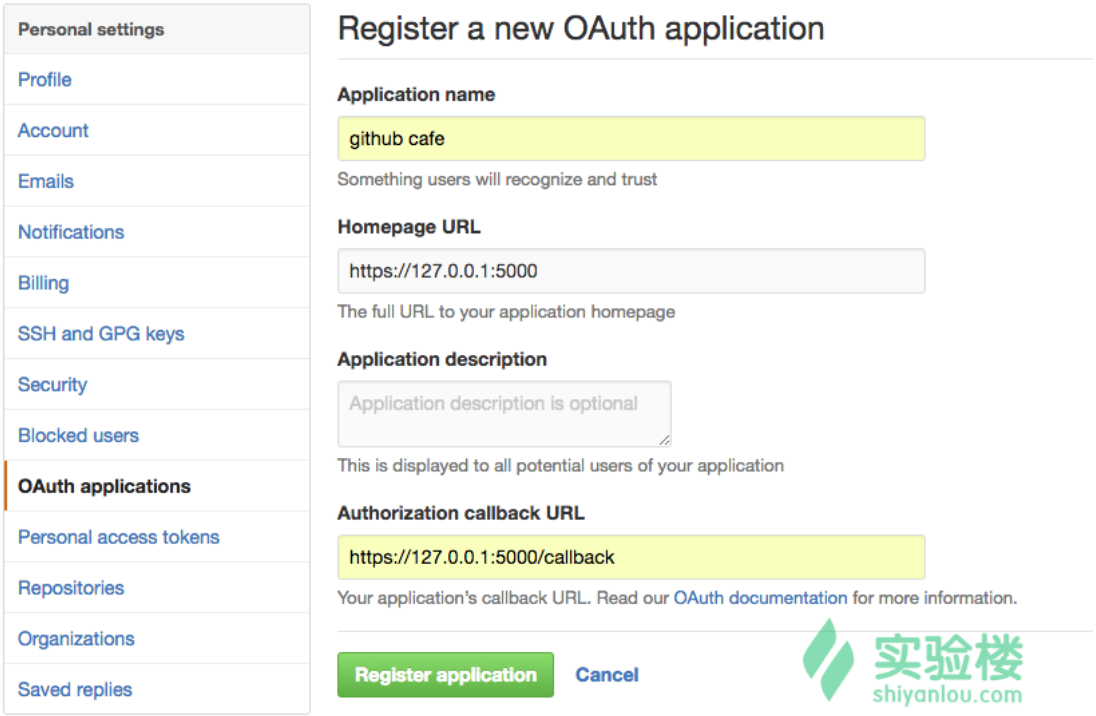
访问 token 的使用一般都有时间限制，刷新 token 可用来获取新的访问 token（不需要用户再次授权）。

到目前为止还没有提到认证过程中的那些 http 报文要如何构造。没有提是怕导致阅读时不连贯，容易打断思路。而且虽然协议是规定好的，但是各大网站对协议的实现都是有差异的，有兴趣的同学可以参考这篇博文：各大網站 OAuth 2.0 實作差異 (https://blog.yorkxin.org/2013/09/30/oauth2-implementation-differences-among-famous-sites)。所以具体还是要通过阅读别人家的文档才知道报文该如何构造。在实战部分，笔者会带大家阅读 Github 的 API 文档，通过这种方式来学习 OAuth 2.0 中报文的构造。

四、项目实战

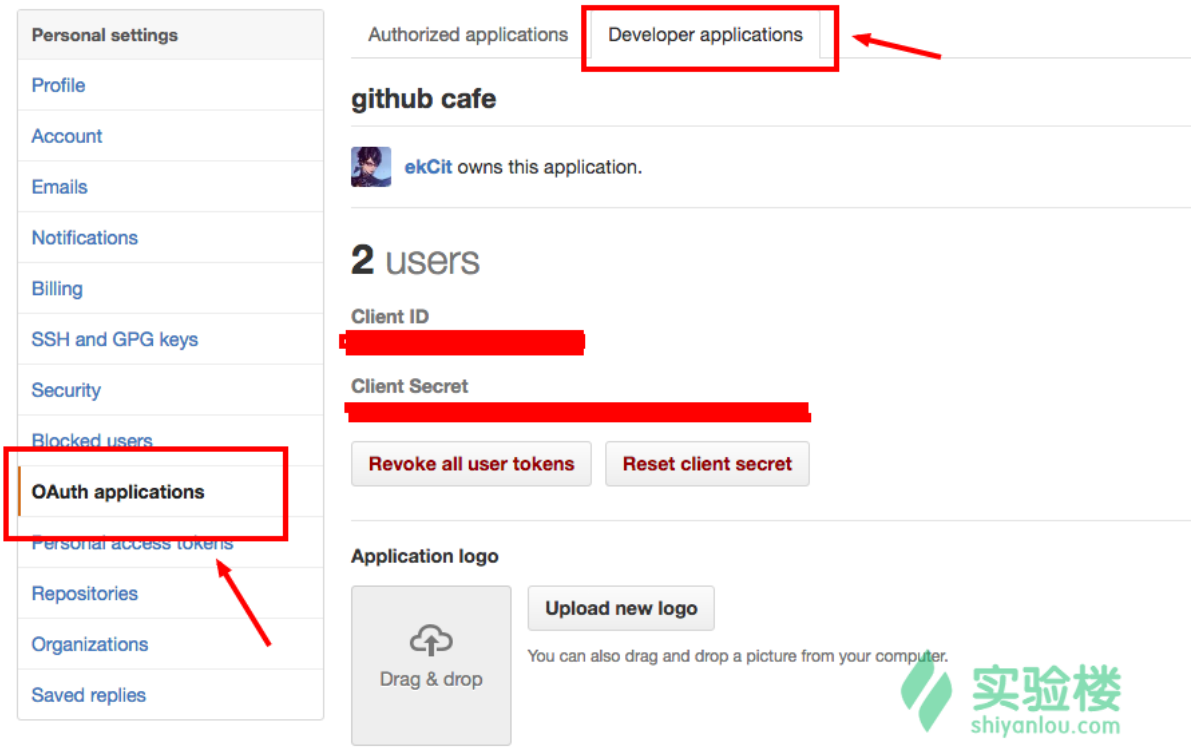
下面我们通过实现一个 web 应用来加深对 OAuth 2.0 协议的理解，授权方式为授权码授权。

首先在 Github 上申请一个新的应用，地址：https://github.com/settings/applications/new (https://github.com/settings/applications/new)



注意主页 URL 和 Callback URL 需指明使用 https 协议，因为 Oauth2 是工作在 SSL 层上的，这样才能保证传输的安全性。当然直接使用 http 协议也是可以的，这里我们还是用 https。

查看客户端 id 与 secret



接着阅读这部分文档：https://developer.github.com/v3/oauth/#web-application-flow (https://developer.github.com/v3/oauth/#web-application-flow)

下面对文档稍作翻译。

Github Oauth 2.0 接口

1.重定向用户到 Github 授权页面

对授权页面做出的请求如下：

```
GET https://github.com/login/oauth/authorize
```

列出参数：

- client_id（必选）：客户端ID，可在应用页面 (https://github.com/settings/applications/)访问查看
- redirect_uri：重定向 url，该地址必须在 `callback_url` 的子目录下
- scope：scope 用来指定你要获取哪些权限。如果为空则只能读取用户的一些公开信息，详见Scopes (https://developer.github.com/v3/oauth/#scopes)
- state：防止 CSRF 攻击用的随机字符串
- allow_signup：认证过程中允许用户注册 Github，默认为 true

其中只有 client_id 是必选的。

根据以上的信息构造的 GET 请求的访问地址可能长这个样子：

```
https://github.com/login/oauth/authorize?
  client_id=...&
  scope=user%20public_repo
```


获取的权限是读写用户信息与读写公开仓库的能力。

2.重定向回你的网页

当授权页面重定向回你的应用页面时，重定向的 URL 后面会接上授权码参数，大概长这样：

```
https://你的重定向url?code=你的一串授权码&state=防止CSRF攻击的随机字串
```

有了授权码就能够申请获取 访问token 了。

对 访问token 发放页面做出请求：

```
POST https://github.com/login/oauth/access_token
```

列出参数：

- client_id（必选）：客户端id，可在应用页面 (https://github.com/settings/applications/)访问查看
- client_secret（必选）：客户端secret，可在应用页面 (https://github.com/settings/applications/)访问查看
- code（必选）：授权码
- redirect_uri：需与之前的重定向 url 一致
- state：防止 CSRF 攻击用的随机字符串

响应默认是以下形式：

```
access_token=e72e16c7e42f292c6912e7710c838347ae178b4a&scope=user%2Cgist&token_type=bearer
```

你也可以在报文头部指定想要的响应形式：

```
Accept: application/json
{"access_token":"e72e16c7e42f292c6912e7710c838347ae178b4a", "scope":"repo,gist", "token_type":"bearer"}

Accept: application/xml
<OAuth>
  <token_type>bearer</token_type>
  <scope>repo,gist</scope>
  <access_token>e72e16c7e42f292c6912e7710c838347ae178b4a</access_token>
</OAuth>
```

3. 使用访问 token 访问 API

你可以将访问 token 用在 URL 参数上：

```
GET https://api.github.com/user?access_token=...
```

不过更推荐你在报文头部存放 token：

```
Authorization: 访问token OAUTH-TOKEN
```

项目架构

因为本项目主要是为了新手快速上手 OAuth 2.0 授权用的，为了避免其它的学习成本，选用的都是简单易用的数据库和 Python 库，这些我都会给出文档地址。不过不阅读文档也不影响实验进行，因为它们的代码都属于你看一眼就知道是在做什么的。

后台数据库用 Mongodb，不用手动创建数据库，NoSQL，操作 Mongodb 就跟操作词典一样简单。

使用的 Python 库如下：

- flask： 轻量级 Python 框架，如果没学过它，还是建议通过Flask - 快速上手 (http://dormousehole.readthedocs.io/en/latest/quickstart.html)快速入门一下。
- flask_pymongo：flask 下操作 mongodb 数据库的库，官方文档 (https://flask-pymongo.readthedocs.io/en/latest/)
- requests_oauthlib：一个对人类友好的 oauth 库，官方文档 (https://requests-oauthlib.readthedocs.io/en/latest/index.html)

项目环境

首先下载项目需要的资料包：

```
$ wget http://labfile.oss.aliyuncs.com/courses/644/material.zip
$ unzip material.zip
$ rm -rf material.zip __MACOSX
```

资料包的内容如下：

```
.
├── requirements.txt
├── ssl.crt
├── ssl.key
├── static
│   ├── css
│   │   └── style.css
│   └── image
│       └── bg2.jpg
└── templates
    └── index.html
```

- `static` 与 `templates` 是关于网页模板的文件，这里直接给出。
- `ssl.crt` 与 `ssl.key` 是为了使用 **https** 协议所需要的文件，你也可以自己生成。参考这个网址：**http://werkzeug.pocoo.org/docs/0.11/serving/#ssl (http://werkzeug.pocoo.org/docs/0.11/serving/#ssl)**
- `requrements.txt` 包含了所需要安装的库。

安装 **mongodb**：

```
$ sudo apt-get install mongodb
```

项目代码是 **python2** 与 **python3** 通用的，同学可以根据自己的喜好来搭建环境：

Python2：

```
$ sudo pip install virtualenv
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirements.txt
```

Python3：

```
$ sudo pip3 install virtualenv
$ virtualenv -p python3 venv3
$ source venv3/bin/activate
$ pip3 install -r requirements.txt
```

项目实现

在工作目录下创建 **app.py** 文件，我会先给出它的代码再一一讲解。（事实上，这部分就是项目的全部代码了）

```
#-*- coding:utf-8 -*-
from flask import Flask, request, session, redirect, url_for, render_template
from flask_pymongo import PyMongo
from requests_oauthlib import OAuth2Session
import datetime, time
import os

app = Flask(__name__)
mongo = PyMongo(app)
app.config["MONGO_HOST"] = "127.0.0.1"
app.config["MONGO_PORT"] = 27017
app.config["MONGO_DBNAME"] = "github_cafe"

client_id = "替换成你申请的应用的客户端id"
client_secret = "替换成你申请的应用的客户端secret"
authorization_base_url = "https://github.com/login/oauth/authorize"
token_url = "https://github.com/login/oauth/access_token"

@app.route('/', methods=["GET", "POST"])
@app.route('/page/<int:page>', methods=["GET", "POST"])
def index(page = 1):
    if request.method == "POST":
        session["geek_wisdom"] = request.form["geek_wisdom"]
        try:
            githuber_say()
        except:
            github = OAuth2Session(client_id)
            authorization_url, state = github.authorization_url(authorization_base_url)
            session["oauth_state"] = state
            return redirect(authorization_url)

    wisdom_list = get_githuber_wisdom(page=page)
    return render_template("index.html", wisdom_list=wisdom_list, page=page, page_count=get_page_count())


@app.route('/callback', methods=["GET"])
def callback():
    github = OAuth2Session(client_id, state=session['oauth_state'])
    token = github.fetch_token(token_url, client_secret=client_secret,
                               authorization_response=request.url)
    session['oauth_token'] = token
    githuber_say()
    return redirect(url_for("index"))

def githuber_say():
    github = OAuth2Session(client_id, token=session['oauth_token'])
    profile = github.get('https://api.github.com/user').json()

    wisdom_dict = {
        "username": profile["name"],
        "avatar_url": profile["avatar_url"],
        "html_url": profile["html_url"],
        "geek_wisdom": session["geek_wisdom"],
        "datetime" : datetime.datetime.today().strftime("%Y/%-m/%d %H:%M"),
        "timestamp" : time.time()
    }

    del session["geek_wisdom"]
    mongo.db.wisdom.insert(wisdom_dict)

def get_githuber_wisdom(count=10, page=1):
    return mongo.db.wisdom.find({}).sort([('timestamp', -1)]).skip(count * (page-1)).limit(count)

def get_page_count(count=10):
    # python3 请将 / 改成 //动手实践是学习 IT 技术最有效的方式!          开始实验
    return mongo.db.wisdom.find({}).count() / count + 1

if __name__ == '__main__':
    app.secret_key = os.urandom(24)
    app.run(debug=True, ssl_context=("ssl.crt", "ssl.key"), threaded=True)
```

flask 应用的配置和一些常量信息:

```
app = Flask(__name__)
mongo = PyMongo(app)
# mongodb 做完这些配置即可，不用手动连接
app.config["MONGO_HOST"] = "127.0.0.1"
app.config["MONGO_PORT"] = 27017
# 数据库名可随意取，它会自动创建
app.config["MONGO_DBNAME"] = "github_cafe"

client_id = "替换成你申请的应用的客户端id"
client_secret = "替换成你申请的应用的客户端secret"
# 授权页面的地址
authorization_base_url = "https://github.com/login/oauth/authorize"
# 分发访问 token 的地址
token_url = "https://github.com/login/oauth/access_token"
```

首页的表单如图所示，输入框的 name 值是 geek_wisdom：

首页的代码逻辑：

```
@app.route('/', methods=["GET", "POST"])
@app.route('/page/<int:page>', methods=["GET", "POST"])
def index(page = 1):
    if request.method == "POST":
        # 获取表单数据，这里由于之后要重定向，先在 session 里暂存一下。
        session["geek_wisdom"] = request.form["geek_wisdom"]
        try:
            # 这一步会尝试直接获取授权资源完成留言提交，如果失败就进入授权步骤
            githuber_say()
        except:
            # ... 构造带了参数的授权页面地址 ...
            # ... 这部分在之后说 ...
            return redirect(authorization_url)

    # 从数据库取得 githuber 的留言
    wisdom_list = get_githuber_wisdom(page=page)
    return render_template("index.html", wisdom_list=wisdom_list, page=page, page_count=get_page_count())
```

构造带了参数的授权页面地址：

```
github = OAuth2Session(client_id)
authorization_url, state = github.authorization_url(authorization_base_url)
session["oauth_state"] = state
```

从之前阅读文档我们知道这步只有 client_id 是必选项，我们的留言本应用只需要读用户的基本信息即可，因此用不到 scope，如果需要，在 OAuth2Session 里带上 scope 参数就可以了。

authorization_url 函数会帮我们构造带了参数的 url，同时会带上 state 参数，保存该参数用于防止 CSRF 攻击。

回调页面的逻辑：

```
@app.route('/callback', methods=["GET"])
def callback():
    github = OAuth2Session(client_id, state=session['oauth_state'])
    # 得到访问 token
    token = github.fetch_token(token_url, client_secret=client_secret,
                               authorization_response=request.url)
    session['oauth_token'] = token
    # 访问授权资源，完成一次留言的提交
    githuber_say()
    # 重定向回主页
    return redirect(url_for("index"))
```

授权码就在重定向的地址之中。

获取访问 token 的这一步只需要 client_id，client_secret，state 和 授权码信息即可。requests_oauthlib 默认返回的响应格式是 json 格式，不过它都帮我们封装好了，我们直接就能拿到访问 token。

`githuber_say()` 的实现

```
def githuber_say():
    github = OAuth2Session(client_id, token=session['oauth_token'])
    # 获取用户个人信息
    profile = github.get('https://api.github.com/user').json()

    # 一条留言所需要的信息数据
    wisdom_dict = {
        "username": profile["name"],
        "avatar_url": profile["avatar_url"],
        "html_url": profile["html_url"],
        "geek_wisdom": session["geek_wisdom"],
        "datetime" : datetime.datetime.today().strftime("%Y/%-m/%d %H:%M"),
        "timestamp" : time.time()
    }

    del session["geek_wisdom"]

    # 数据库插入一条留言数据
    mongo.db.wisdom.insert(wisdom_dict)
```

关于有哪些接口可使用，都是什么功能，参考：https://developer.github.com/v3/users/ (https://developer.github.com/v3/users/) 右边一列。

`mongo.db` 是配置的数据库， `mongo.db.wisdom` 是数据库下的一个集合（很像 SQL 数据库里的表），集合的名字任取（在这里是 wisdom ），当你对集合做插入操作时会自动在数据库中创建该集合。

最后应用运行的参数需指定 `ssl_context` ，这样应用就能 ssl 层上通信了。

```
if __name__ == '__main__':
    app.secret_key = os.urandom(24)
    app.run(debug=True, ssl_context=("ssl.crt", "ssl.key"), threaded=True)
```

动手实践是学习 IT 技术最有效的方式！ 开始实验

运行结果

新开一个标签页启动 mongodb：


```
$ mkdir db
$ mongod --dbpath ./db
```

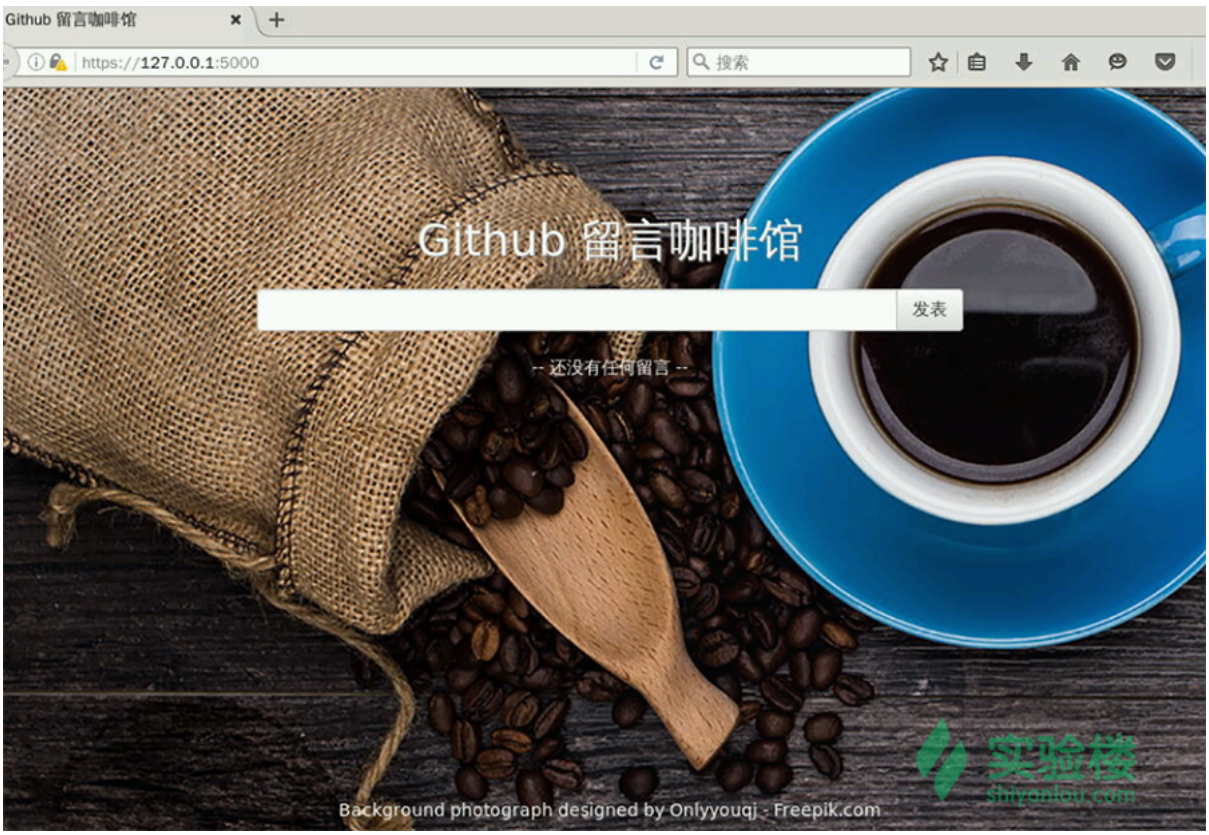
运行 **flask** 应用（请确保在 **virtualenv** 的虚拟环境下）：

```
(venv)$ python app.py
```

访问 `https://127.0.0.1:5000` （一定要加上 **https** 协议头）

显示连接不安全，这时只要添加例外即可：

应用首页：



动手实践是学习 IT 技术最有效的方式！ 开始实验

五、总结

在本课程中，我们学习了 OAuth 2.0 的四种授权流程，实现了一个简单的第三方应用，但还是有部分内容没有讲到，比如刷新 token 的使用，这一部分可参考协议 https://tools.ietf.org/html/rfc6749#section-6 (https://tools.ietf.org/html/rfc6749#section-6) 或是 requests-oauthlib 官方文档中的内容：https://requests-oauthlib.readthedocs.io/en/latest/oauth2_workflow.html#refreshing-tokens (https://requests-oauthlib.readthedocs.io/en/latest/oauth2_workflow.html#refreshing-tokens) 。在实际操作中，如何授权还是以服务提供者提供的文档为准。

六、参考资料