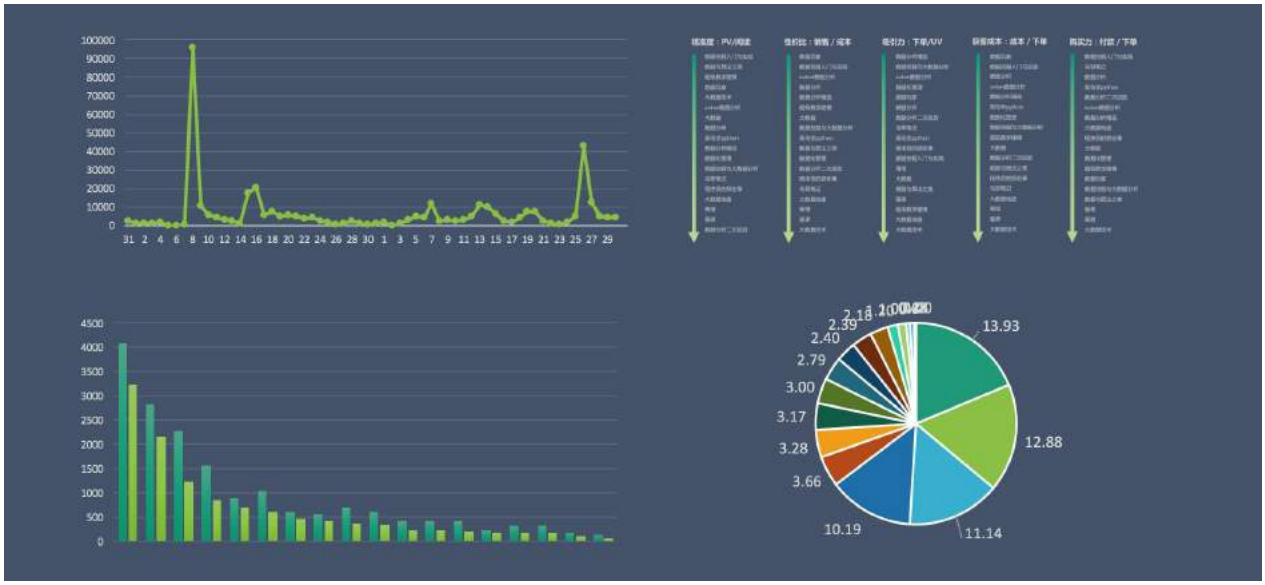
Table of Contents

为什么要学爬虫?

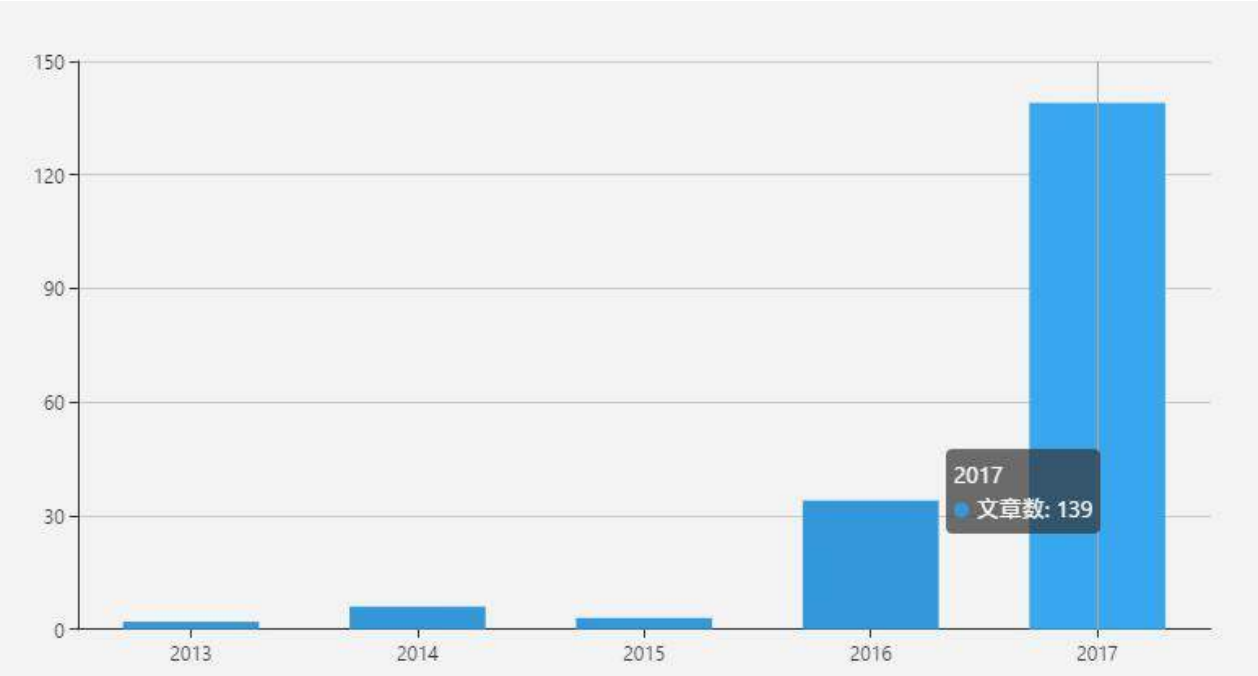
爬虫是一个非常具有实践性的编程技能，它并不是程序员的专属技能，任何具有一定编程基础的人都可以学习爬虫，写个爬虫分析一下股票走势，写个爬虫YouTube下载视频，上链家爬个房源数据分析房价趋势，爬知乎、爬豆瓣、爬新浪微博、爬影评，爬虫有太多可以做的事情，人工智能时代，对数据的依赖越来越重要。

> 马云说：数据是新一轮技术革命最重要的生产资料。

数据主要的来源就是通过爬虫获取，通过爬虫获取数据可以进行市场调研和数据分析，可以作为机器学习和数据挖掘的原始数据，我们通过微信公众号爬虫得到的数据对新媒体内容提供运营策略。



通过爬虫发现原来我4年前就在公众号写的文章，最近一年写了一百多篇，这些数据在微信平台是没法统计的，只有通过爬虫自己来统计分析。



对小白来说，爬虫可能是一个很复杂的事情，现在我们带着一个具体的目标（以爬虫微信公众号文章为例），在目标的驱动下，跟着这本小册一步一步学会爬虫，同时，那些所谓的前置知识也在这个过程中学会了。在这本小册中，我将以手把手的方式教会你如何进行网络爬虫。

为什么要学Python

Python 作为一门连小学生都可以学会的语言，非常适合没有编程基础的同学。它可以让你更快的理解编程的思想，能让你体会到通过编程来解决问题带来快乐，它没有复杂的语法，最为接近伪代码的语言，没有繁琐的编译过程，也不需要你手动管理内存，类库非常丰富，解决各种问题都有很多现成的工具，无需自己造轮子。

> Python之父说：人生苦短，我用Python。

你会学到什么?

- 爬虫基本原理
- 爬虫工具 Requests 的基本使用
- 数据抓包分析工具 Fiddler 的基本使用
- MongoDB 数据库的基本使用
- 使用 Pandas 进行数据分析
- 使用 Matplotlib 进行数据可视化展示

你需要准备什么?

任何对网络爬虫感兴趣者，或者是对微信公众号数据感兴趣的人都可以参与到这本小册中来，你需要准备的东西包括：

- 一台移动设备（Android或者iOS手机）
- 一个可登录的微信帐号
- 一台可以联网的电脑
- 还需要会一点点Python编程基础

温馨提醒
最后还是要声明一下，爬虫与反爬虫就像矛与盾，它们之间的较量是一场没

微信公众号爬虫的基本原理

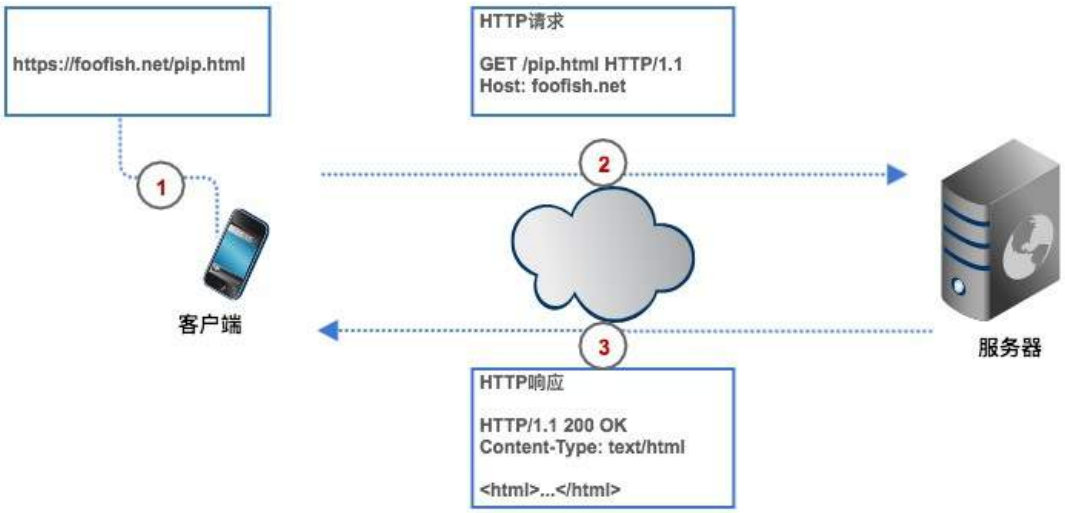
网上关于爬虫的教程多如牛毛，但很少有看到微信公众号爬虫教程，而且最多的就是使用搜狗微信，不过它有诸多的弊端，比如文章链接是临时的，没有阅读量等指标，所以我想写一个比较系统的关于如何通过手机客户端利用 Python 爬微信公众号文章的教程，并对公众号文章做数据分析，为更好的运营公众号提供决策。

爬虫的基本原理

所谓爬虫就是一个自动化数据采集工具，你只要告诉它要采集哪些数据，丢给它一个 URL，就能自动地抓取数据了。其背后的基本原理就是通过爬虫程序向目标服务器发起 HTTP 请求，目标服务器返回响应结果，爬虫收到响应并从中提取数据，再进行数据清洗、数据存储。

爬虫的基本流程

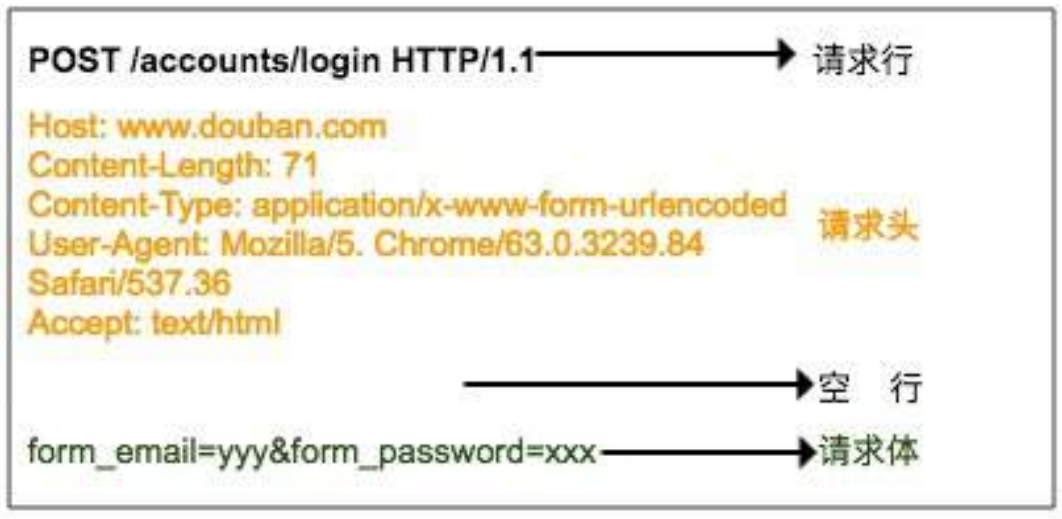
爬虫的基本流程其实就是一个 HTTP 请求的过程，以浏览器访问一个网址为例，从用户输入 URL 开始，客户端通过 DNS 解析查询到目标服务器的 IP 地址，然后与之建立 TCP 连接，连接成功后，浏览器就构造一个 HTTP 请求发送给服务器，服务器收到请求之后，从数据库查到相应的数据并封装成一个 HTTP 响应，然后将响应结果返回给浏览器，浏览器对响应内容进行数据解析、提取、渲染并最终展示在你面前。



HTTP 协议的请求和响应都必须遵循固定的格式，只有遵循统一的 HTTP 请求格式，服务器才能正确解析不同客户端发的请求，同样地，服务器遵循统一的响应格式，客户端才得以正确解析不同网站发过来的响应ni。

HTTP 请求格式

HTTP 请求由请求行、请求头、空行、请求体组成。



请求行由三部分组成：

1. 第一部分是请求方法，常见的请求方法有 GET、POST、PUT、DELETE、HEAD
2. 第二部分是客户端要获取的资源路径
3. 第三部分是客户端使用的 HTTP 协议版本号

请求头是客户端向服务器发送请求的补充说明，比如 User-Agent 向服务器说明客户端的身份。

请求体是客户端向服务器提交的数据，比如用户登录时需要提高的账号密码信息。请求头与请求体之间用空行隔开。请求体并不是所有的请求都有的，比如一般的GET都不会带有请求体。

上图就是浏览器登录豆瓣时向服务器发送的HTTP POST 请求，请求体中指定了用户名和密码。

HTTP 响应格式

HTTP 响应格式与请求的格式很相似，也是由响应行、响应头、空行、响应体组成。

响应行也包含三部分，分别是服务端的 HTTP 版本号、响应状态码、状态说明，响应状态码常见有 200、400、404、500、502、304 等等，一般以 2 开头的表示服务器正常响应了客户端请求，4 开头表示客户端的请求有问题，5 开头表示服务器出错了，没法正确处理客户端请求。状态码说明就是对该状态码的一个简短描述。

第二部分就是响应头，响应头与请求头对应，是服务器对该响应的一些附加说明，比如响应内容的格式是什么，响应内容的长度有多少、什么时间返回给客户端的、甚至还有一些 Cookie 信息也会放在响应头里面。

第三部分是响应体，它才是真正的响应数据，这些数据其实就是网页的 HTML 源代码。

小结

这仅仅只是一个爬虫基本原理的介绍，涉及的 HTTP 协议的内容也非常有限，但不可能用一篇文章事无巨细的介绍完，因为 HTTP 协议是一个很大的话题，用一本书也写不完，深入了解推荐两本书《图解HTTP》、《HTTP权威指南》。

使用 Requests 实现一个简单网页爬虫

> 友情提示：小册代码全部基于 `Python3.6` 实现

第一节我们简单介绍了爬虫的基本原理，理解原理可以帮助我们更好的实现代码。Python 提供了非常多工具去实现 HTTP 请求，但第三方开源库提供的功能更丰富，你无需从 socket 通信开始写，比如使用Pyton内建模块 `urllib` 请求一个 URL 代码示例如下：

```
import ssl

from urllib.request import Request
from urllib.request import urlopen

context = ssl._create_unverified_context()

# HTTP 请求
request = Request(url="https://foofish.net/pip.html",
                  method="GET",
                  headers={"Host": "foofish.net"},
                  data=None)

# HTTP 响应
response = urlopen(request, context=context)
headers = response.info()  # 响应头
content = response.read() # 响应体
code = response.getcode() # 状态码
```

发起请求前首先要构建请求对象 Request，指定 url 地址、请求方法、请求头，这里的请求体 data 为空，因为你不需要提交数据给服务器，所以你也可以不指定。urlopen 函数会自动与目标服务器建立连接，发送 HTTP 请求，该函数的返回值是一个响应对象 Response，里面有响应头信息，响应体，状态码之类的属性。

但是，Python 提供的这个内建模块过于低级，需要写很多代码，使用简单爬虫可以考虑 Requests，Requests 在GitHub 有近30k的Star，是一个很Pythonic的框架。先来简单熟悉一下这个框架的使用方式

安装 requests

```
pip install requests
```

GET 请求

```
>>> r = requests.get("https://httpbin.org/ip")
>>> r
<Response [200]> # 响应对象
>>> r.status_code  # 响应状态码
200

>>> r.content  # 响应内容
'{\n  "origin": "183.237.232.123"\n}\n'
```

POST 请求

```
>>> r = requests.post('http://httpbin.org/post', data =
```

自定义请求头

这个经常会用到，服务器反爬虫机制会判断客户端请求头中的User-Agent是否来源于真实浏览器，所以，我们使用Requests经常会指定UA伪装成浏览器发起请求

```
>>> url = 'https://httpbin.org/headers'
>>> headers = {'user-agent': 'Mozilla/5.0'}
>>> r = requests.get(url, headers=headers)
```

参数传递

很多时候URL后面会有一串很长的参数，为了提高可读性，requests 支持将参数抽离出来作为方法的参数（params）传递过去，而无需附在 URL 后面，例如请求 url http://bin.org/get?key=val ，可使用

```
>>> url = "http://httpbin.org/get"
>>> r = requests.get(url, params={"key":"val"})
>>> r.url
u'http://httpbin.org/get?key=val'
```

指定Cookie

Cookie 是web浏览器登录网站的凭证，虽然 Cookie 也是请求头的一部分，我们可以从中剥离出来，使用 Cookie 参数指定

```
>>> s = requests.get('http://httpbin.org/cookies', cook
>>> s.text
u'{\n  "cookies": {\n    "from-my": "browser"\n  }\n}\r
```

设置超时

当发起一个请求遇到服务器响应非常缓慢而你又不希望等待太久时，可以指定 timeout 来设置请求超时时间，单位是秒，超过该时间还没有连接服务器成功时，请求将强行终止。

```
r = requests.get('https://google.com', timeout=5)
```

设置代理

一段时间nei发送的请求太多容易被服务器判定为爬虫，所以很多时候我们使用代理IP来伪装客户端的真实IP。

```
import requests

proxies = {
    'http': 'http://127.0.0.1:1080',
    'https': 'http://127.0.0.1:1080',
}

r = requests.get('http://www.kuaidaili.com/free/', prox
```

Session

如果想和服务器一直保持登录（会话）状态，而不必每次都指定 cookies，那么可以使用 session，Session 提供的API和 requests 是一样的。

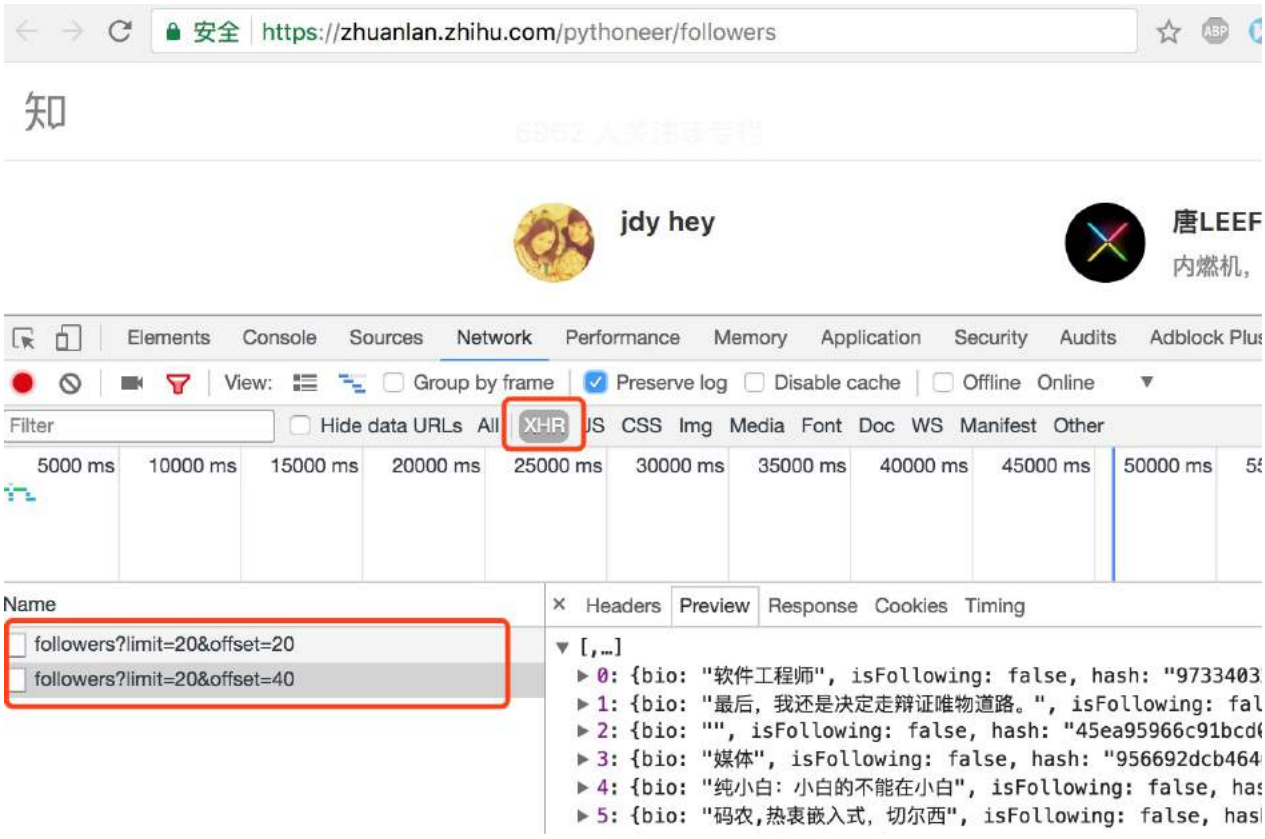
```
import requests

s = requests.Session()
s.cookies = requests.utils.cookiejar_from_dict({"a": "c
r = s.get('http://httpbin.org/cookies')
print(r.text)
# '{"cookies": {"a": "c"}}'

r = s.get('http://httpbin.org/cookies')
print(r.text)
# '{"cookies": {"a": "c"}}'
```

小试牛刀

现在我们使用Requests完成一个爬取知乎专栏用户关注列表的简单爬虫为例，找到任意一个专栏，打开它的关注列表。用 Chrome 找到获取粉丝列表的请求地址：https://zhuanlan.zhihu.com/api/columns/pythoneer/followers?limit=20&offset=20



然后我们用 Requests 模拟浏览器发送请求给服务器

```
import json

import requests


class SimpleCrawler:
    init_url = "https://zhuanlan.zhihu.com/api/columns/
    offset = 0

    def crawl(self, params=None):
        # 必须指定UA，否则知乎服务器会判定请求不合法
        headers = {
            "Host": "zhuanlan.zhihu.com",
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0
                          "AppleWebKit/537.36 (KHTML, I
        }
```

```
        response = requests.get(self.init_url, headers=
        print(response.url)
        data = response.json()
        # 7000表示所有关注量
        # 分页加载更多，递归调用
        while self.offset < 7000:
            self.parse(data)
            self.offset += 20
            params = {"limit": 20, "offset": self.offse
            self.crawl(params)

    def parse(self, data):
    # 以json格式存储到文件
        with open("followers.json", "a", encoding="utf-
            for item in data:
                f.write(json.dumps(item))
                f.write('\n')


if __name__ == '__main__':
    SimpleCrawler().crawl()
```

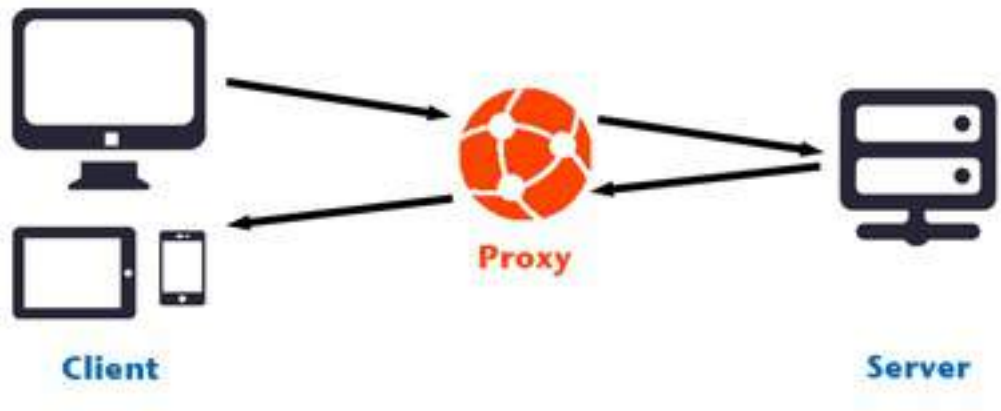
小结

这就是一个最简单的基于 Requests 的单线程知乎专栏粉丝列表的爬虫，requests 非常灵活，请求头、请求参数、Cookie 信息都可以直接指定在请求方法中，返回值 response 如果是 json 格式可以直接调用json()方法返回 python 对象。关于 Requests 的更多使用方法可以参考官方文档：http://docs.python-requests.org/en/master/

使用 Fiddler 抓包分析公众号请求过程

上一节我们熟悉了 Requests 基本使用方法，配合 Chrome 浏览器实现了一个简单爬虫，但因为微信公众号的封闭性，微信公众平台并没有对外提供 Web 端入口，只能通过手机客户端接收、查看公众号文章，所以，为了窥探到公众号背后的网络请求，我们需要借以代理工具的辅助。

代理工具又称之为抓包工具，主流的抓包工具 Windows 平台有 Fiddler，macOS 有 Charles，阿里开源了一款工具叫 AnyProxy。它们的基本原理都是类似的，就是通过在手机客户端设置好代理IP和端口，客户端所有的 HTTP、HTTPS 请求就会经过代理工具，在代理工具中就可以清晰地看到每个请求的细节，然后可以分析出每个请求是如何构造的，弄清楚这些之后，我们就可以用 Python 模拟发起请求，进而得到我们想要的数据。

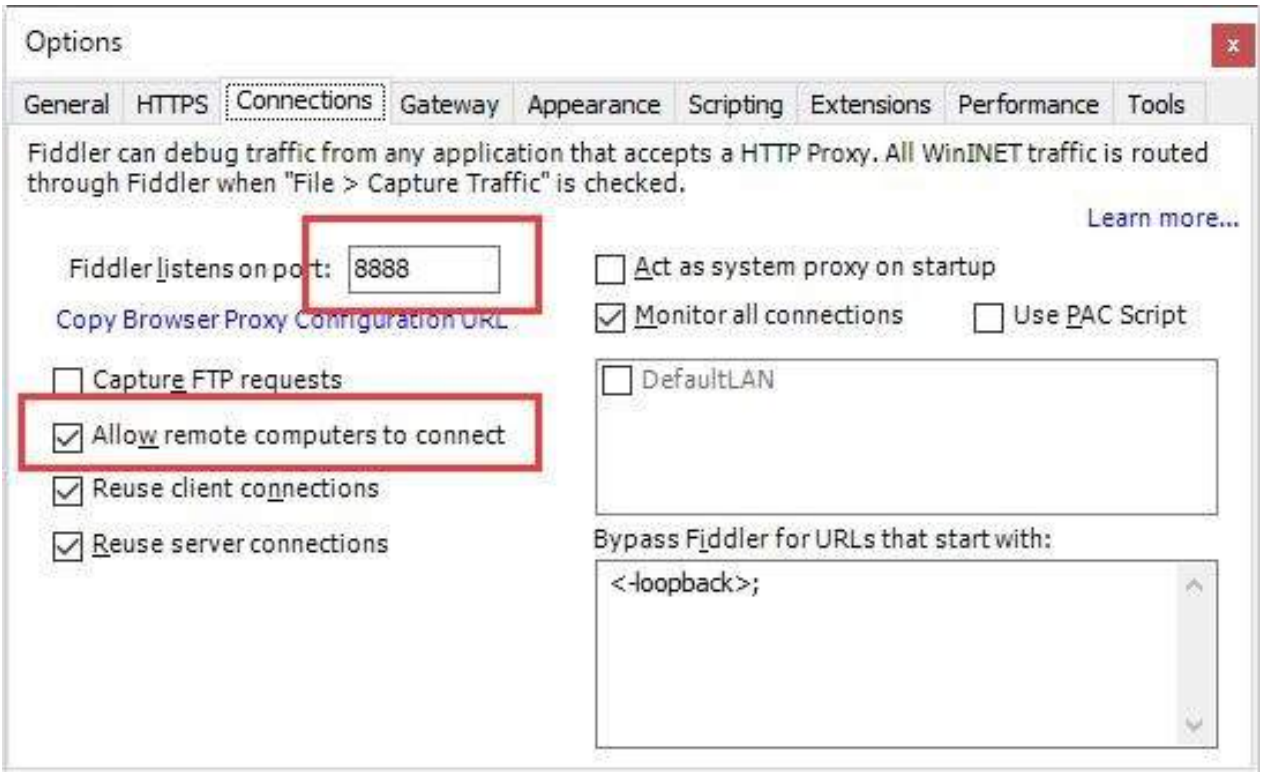


Fiddler 下载地址是 https://www.telerik.com/download/fiddler，安装包就 4M 多，在配置之前，首先要确保你的手机和电脑在同一个局域网，如果不再同一个局域网，你可以买个随身WiFi，在你电脑上搭建一个极简无线路由器。安装过程一路点击下一步完成就可以了。

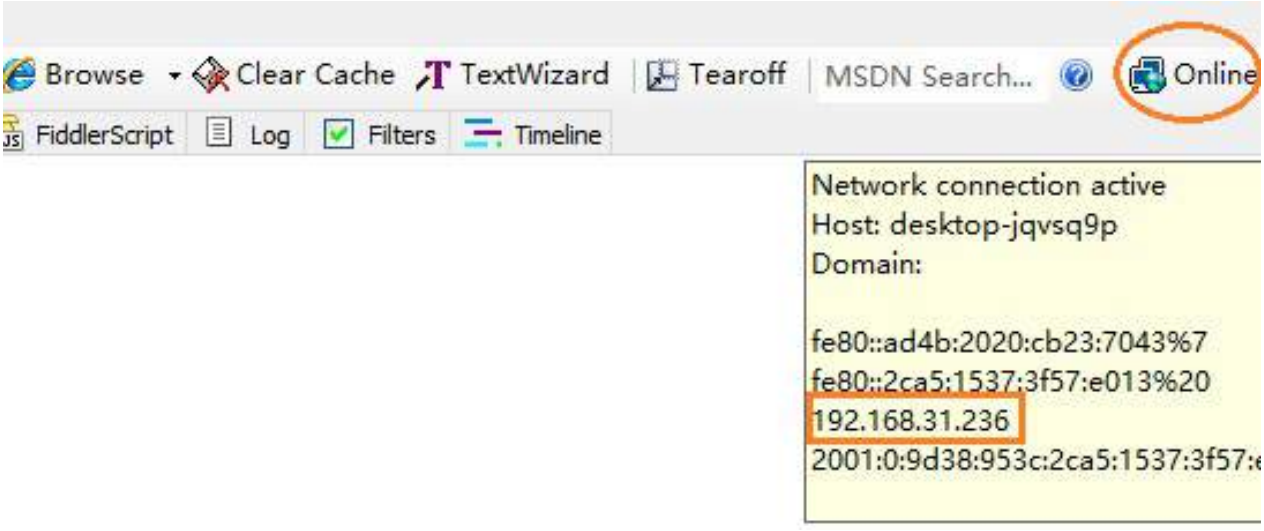
Fiddler 配置

选择 **Tools > Fiddler Options > Connections**

Fiddler 默认的端口是使用 8888，如果该端口已经被其它程序占用了，你需要手动更改，勾选 Allow remote computers to connect，其它的选择默认配置就好，配置更新后记得重启 Fiddler。一定要重启 Fiddler，否则代理无效。



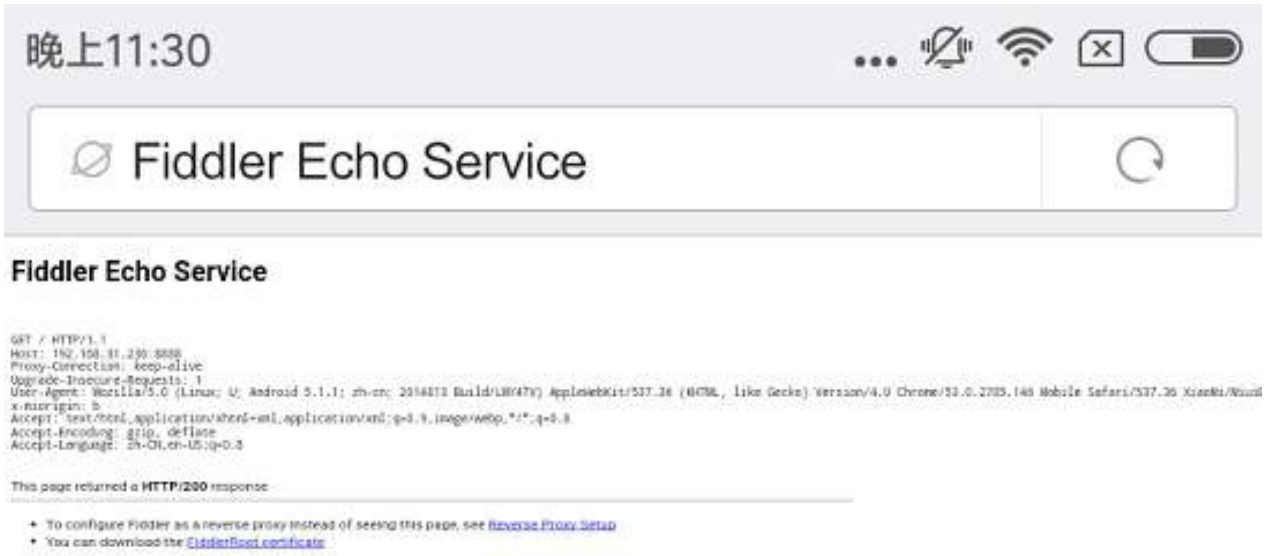
接下来你需要配置手机，我们以 Android 设备为例，现在假设你的手机和电脑已经在同一个局域网（只要连的是同一个路由器就在同局域网内），找到电脑的 IP 地址，在 Fiddler 右上角有个 Online 图标，鼠标移过去就能看到IP了，你也可以在CMD窗口使用 `ipconfig` 命令查看到



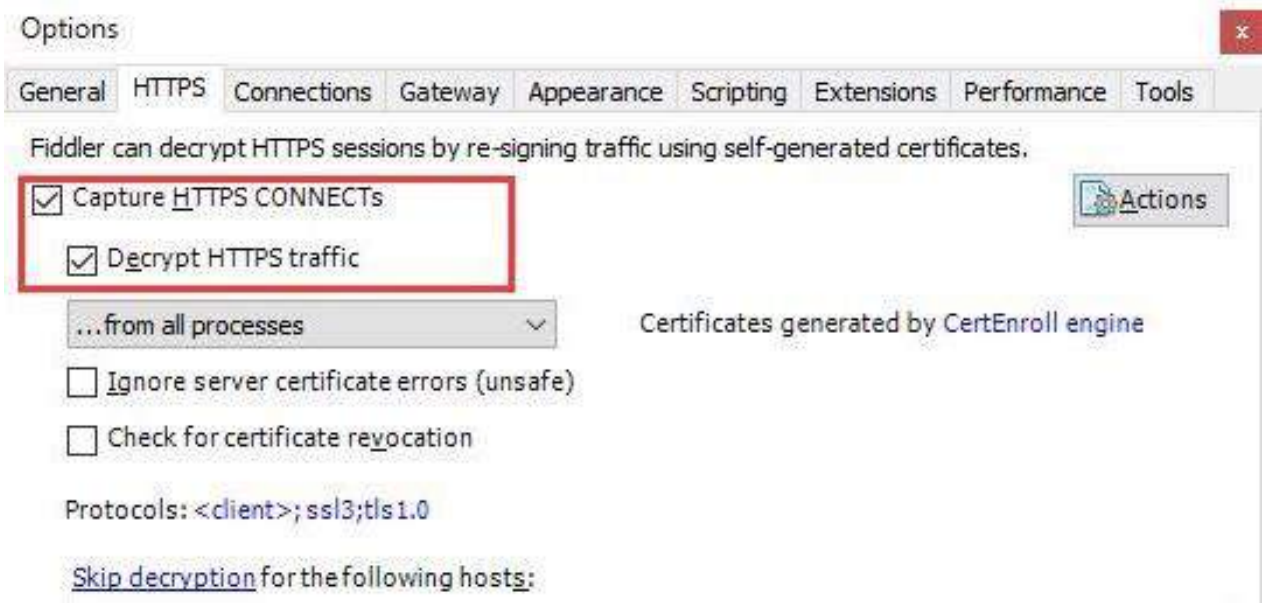
Android 手机代理配置

进入手机的 WLAN 设置，选择当前所在局域网的 WiFi 链接，设置代理服务器的 IP 和端口，我这是以小米设备为例，其它 Android 手机的配置过程大同小异。

测试代理有没有设置成功可以在手机浏览器访问你配置的地址：http://192.168.31.236:8888/ 会显示 Fiddler 的回显页面，说明配置成功。

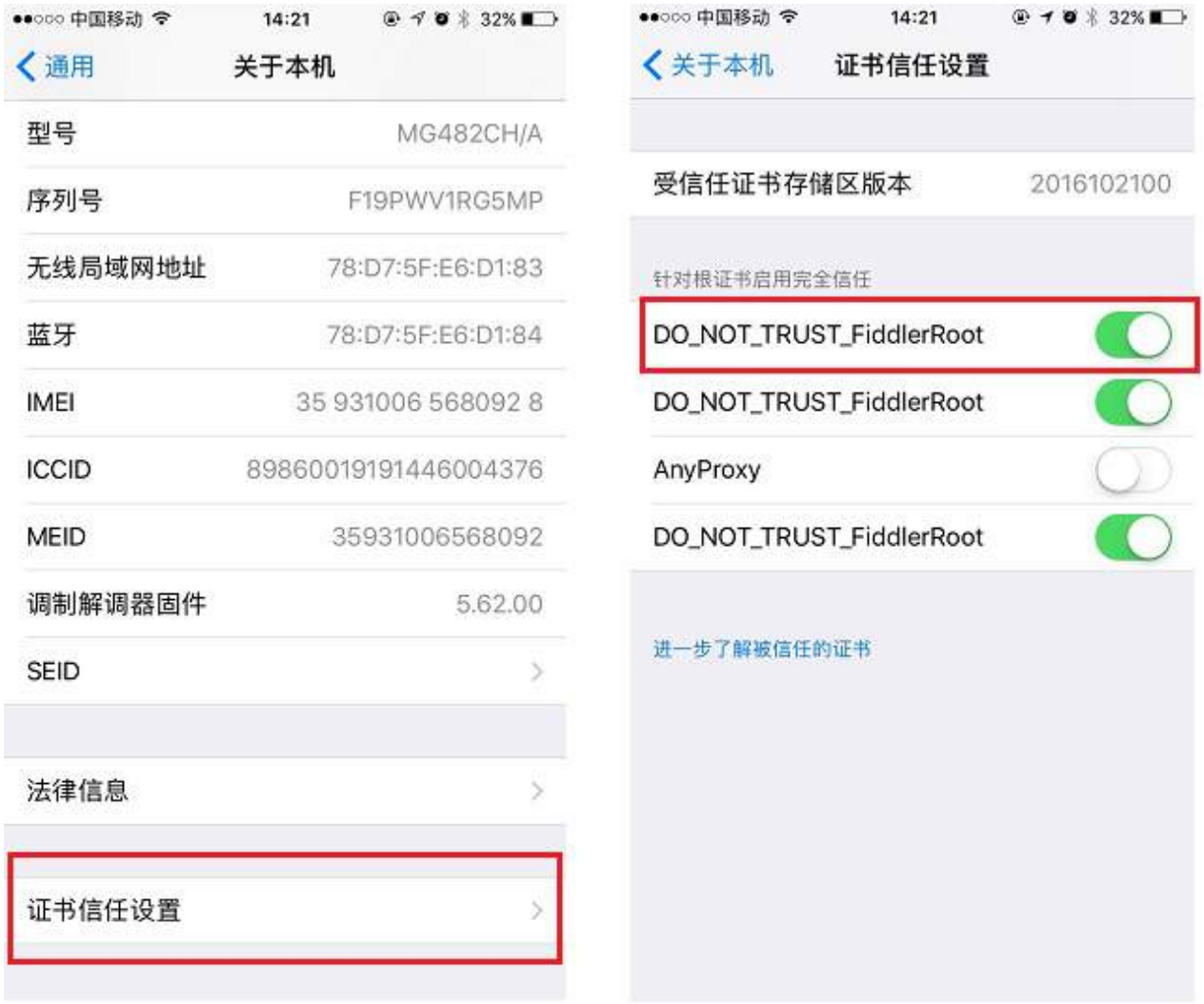


现在你打开任意一个HTTP协议的网站都能看到请求会出现在 Fiddler 窗口，但是 HTTPS 的请求并没有出现在 Fiddler 中，其实还差一个步骤，需要在 Fiddler 中激活 HTTPS 抓取设置。在 Fiddler 选择 **Tools > Fiddler Options > HTTPS > Decrypt HTTPS traffic**， 重启 Fiddler。



为了能够让 Fiddler 截取 HTTPS 请求，客户端都需要安装且信任 Fiddler 生成的 CA 证书，否则会出现“网络出错，轻触屏幕重新加载:-1200”的错误。在浏览器打开 Fiddler 回显页面 http://192.168.31.236:8888/ 下载 **FiddlerRoot certificate**，下载并安装证书，并验证通过。

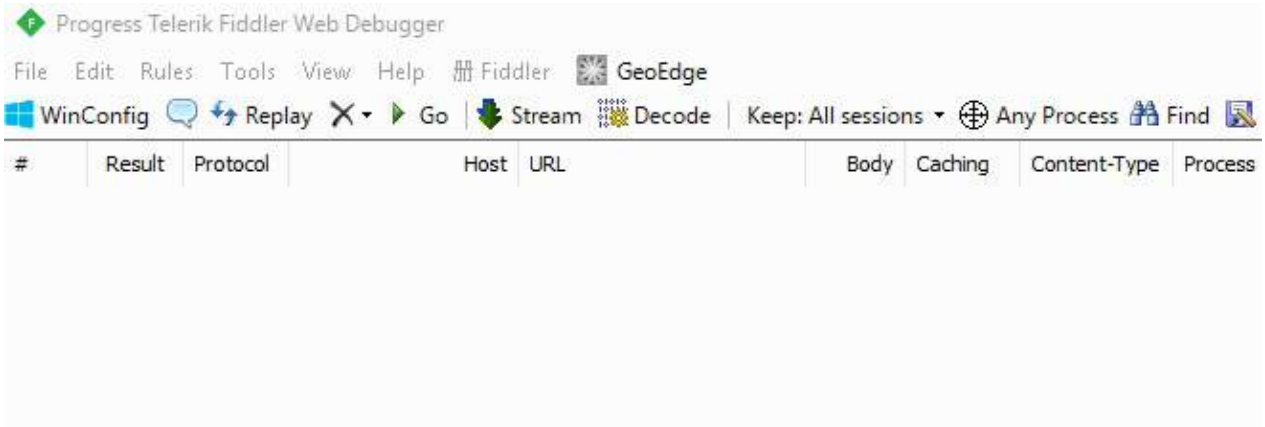
iOS下载安装完成之后还要从 **设置->通用->关于本机->证书信任设置** 中把 Fiddler 证书的开关打开



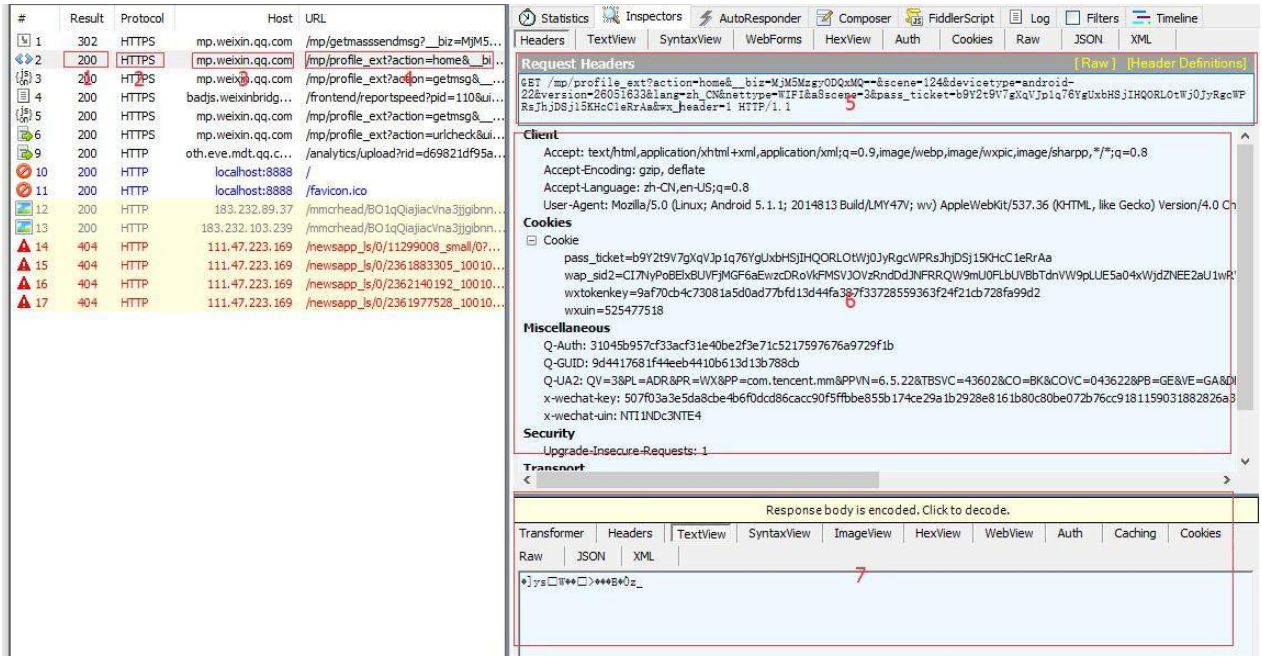
Android 手机下载保存证书后从系统设置里面找到系统安全，从SD卡安装证书，如果没有安装证书，打开微信公众号的时候会弹出警告。

至此，所有的配置都完成了，现在打开微信随便选择一个公众号，进入公众号的【查看历史消息】

同时观察 Fiddler 的主面板



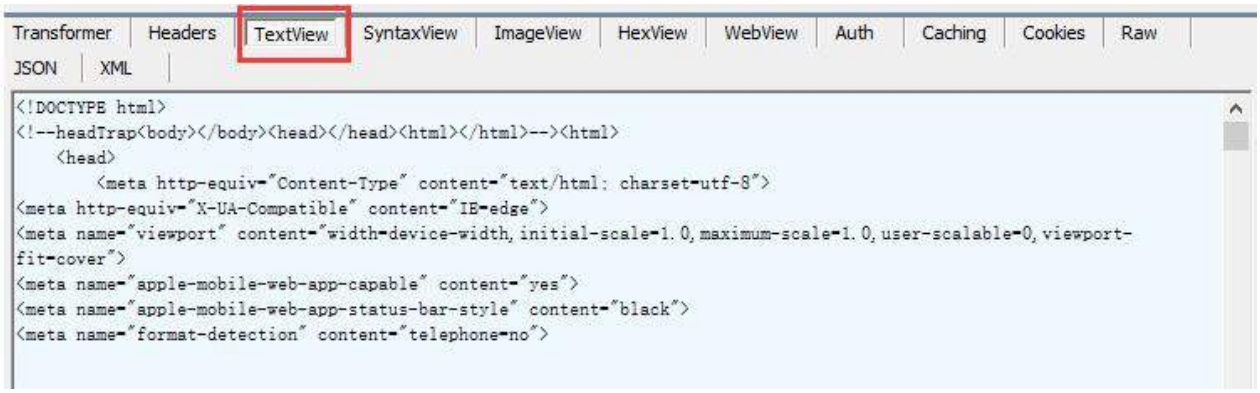
当微信从公众号介绍页进入历史消息页面时，在 Fiddler 上已经能看到有请求进来了，这些请求就是微信 APP 向服务器发送的请求，现在简单介绍一下这个请求面板上的每个模块的意义。



我把上面的主面板划分为 7 大块，你需要理解每块的内容，后面才有可能会用 Python 代码来模拟微信请求。

1、服务器的响应结果，200 表示服务器对该请求响应成功
2、请求协议，微信的请求协议都是基 于HTTPS 的，所以前面一定要配置好，不然你看不到 HTTPS 的请求。
3、微信服务器主机名
4、请求路径
5、请求行，包括了请求方法（GET），请求协议（HTTP/1.1），请求路径（/mp/profile_ext...后面还有很长一串参数） 6、包括Cookie信息在内的请求头。
7、微信服务器返回的响应数据，我们分别切换成 TextView 和 WebView 看一下返回的数据是什么样的。

TextView 模式下的预览效果是服务器返回的 HTML 源代码



WebView 模式是 HTML 代码经过渲染之后的效果，其实就是我们在手机微信中看到的效果，只不过因为缺乏样式，所以没有手机上看到的美化效果。

如果服务器返回的是 Json格式或者是 XML，你还可以切换到对应的页面预览查看。

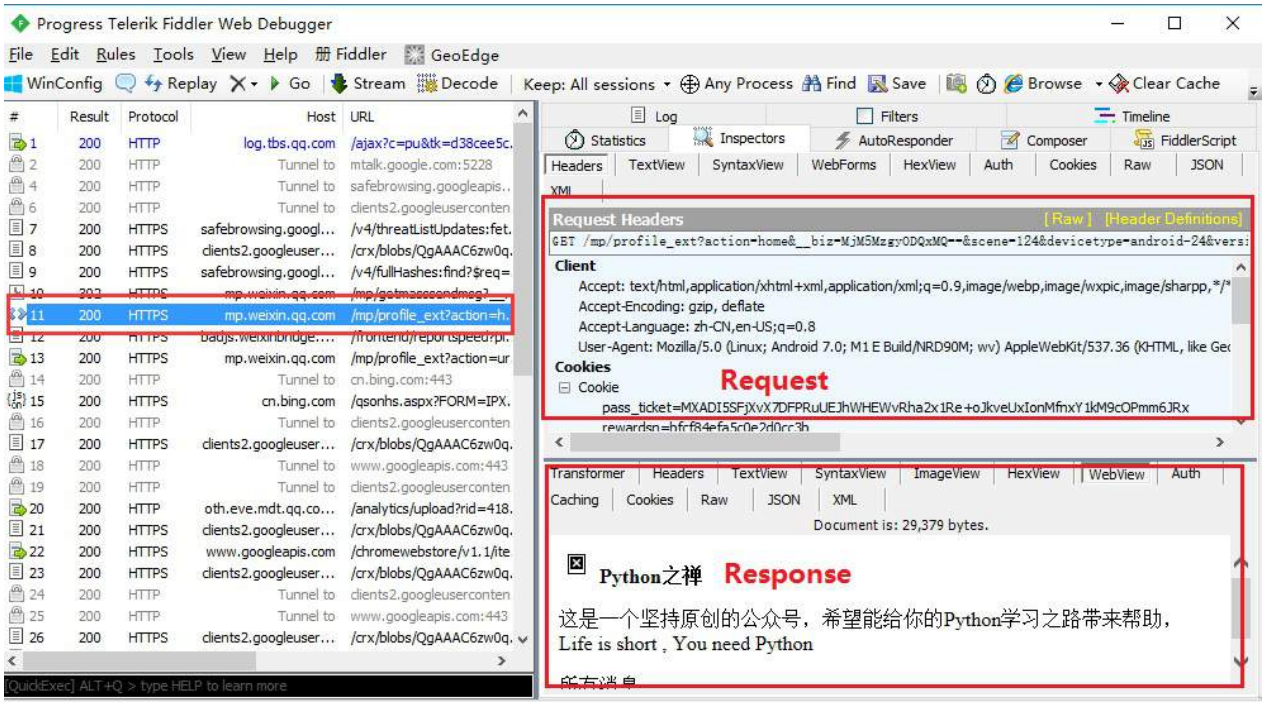
小结

配置好Fiddler的几个步骤主要包括指定监控的端口，开通HTTPS流量解密功能，同时，客户端需要安装CA证书。下一节我们基于Requests模拟像微信服务器发起请求。

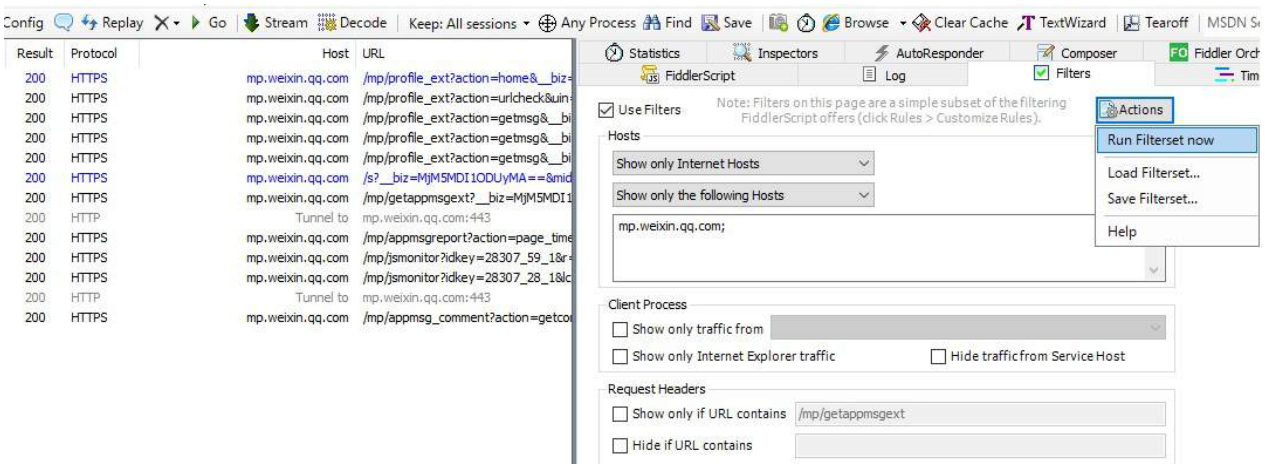
抓取第一篇微信公众号文章

上一节我们熟悉了 Fiddler 的基本操作以及每个模块所代表的意义，这节我们详细了解获取微信公众号历史文章接口的请求情况，以及如何使用 Python 模拟微信发送请求获取公众号文章的基本信息。

打开微信历史消息页面时，我们从 Fiddler 看到了很多请求，为了找到微信历史文章的接口，我们要逐个查看 Response 返回的内容，最后发现第 11 个请求就是我们要寻找的



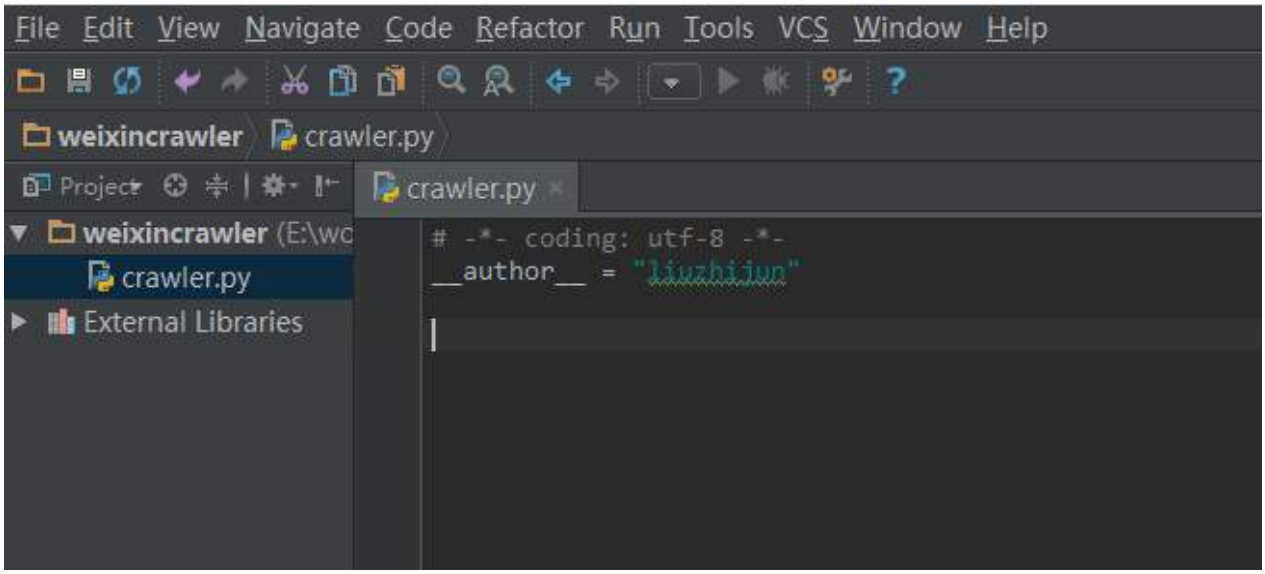
确定微信公众号的请求HOST是 mp.weixin.qq.com 之后，我们可以使用过滤器来过滤掉不相关的请求。



爬虫的基本原理就是模拟浏览器发送 HTTP 请求，然后从服务器得到响应结果，现在我们就用 Python 实现如果发送一个 HTTP 请求。这里我们使用 requests 库来发送请求。

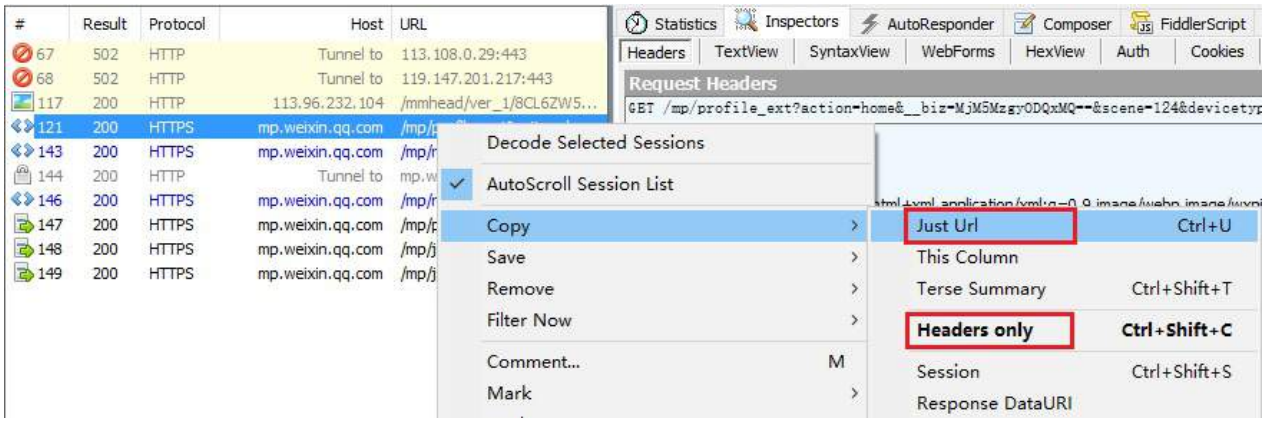
创建一个 Pycharm 项目

我们使用 Pycharm 作为开发工具，你也可以使用其它你熟悉的工具，Python 环境是 Python3（推荐使用 Python3.6），先创建一个项目 weixincrawler



现在我们来编写一个最粗糙的版本，第一是要找到请求的完整URL地址，第二是找到完整的请求头（headers）信息，头信息里面包括了cookie、User-agent、Host 等信息。

我们直接从 Fiddler 请求中拷贝 URL 和 Headers， **右键 -> Copy -> Just Url/Headers Only**



最终拷贝出来的URL很长，它包含了很多的参数

```
url = "https://mp.weixin.qq.com/mp/profile_ext" \
      "?action=home" \
      "&__biz=MjM5MzgyODQxMQ==" \
      "&scene=124" \
      "&devicetype=android-24" \
      "&version=26051633&lang=zh_CN" \
      "&nettype=WIFI&a8scene=3" \
      "&pass_ticket=MXADI5SFjXvX7DFPRuUEJhWHEWvRha2x1Re
      "&wx_header=1"
```

暂且不去分析（猜测）每个参数的意义。然后把 Headers 拷贝出来，发现 Fiddler 把 请求行、响应行、响应头都包括进来了，我们只需要中间的请求头部分。

因为 requests.get 方法里面的 headers 参数必须是字典对象，所以，先要写个函数把刚刚拷贝的字符串转换成字典对象。

```
def headers_to_dict(headers):
    """
    将字符串
    '''
    Host: mp.weixin.qq.com
    Connection: keep-alive
    Cache-Control: max-age=
    '''
    转换成字典对象
    {
        "Host": "mp.weixin.qq.com",
        "Connection": "keep-alive",
        "Cache-Control":"max-age="
    }
    :param headers: str
    :return: dict
    """
    headers = headers.split("\n")
    d_headers = dict()
    for h in headers:
        if h:
            k, v = h.split(":", 1)
            d_headers[k] = v.strip()
    return d_headers
```

最终 v0.1 版本出来的，不出意外的话，公众号历史文章数据就在 response.text 中。如果返回的内容非常短，而且title标签是 `<title>验证</title>` ，那么说明你的请求参数或者请求头有误，最有可能的一种请求就是 Headers 里面的 Cookie 字段过期，从手机微信端重新发起一次请求获取最新的请求参数和请求头试试。

```
# v0.1
def crawl():
    url = "https://mp.weixin.qq.com/..." # 省略了
    headers = """  # 省略了
Host: mp.weixin.qq.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
    """
    headers = headers_to_dict(headers)
    response = requests.get(url, headers=headers, verif
    print(response.text)
```

最后，我们顺带把响应结果另存为html文件，以便后面重复使用，分析里面的内容

```
with open("weixin_history.html", "w", encoding="utf-8")
    f.write(response.text)
```

用浏览器打开 weixin_history.html 文件，查看该页面的源代码，搜索微信历史文章标题的关键字 "11月赠书"（就是我以往发的文章），你会发现，历史文章封装在叫 msgList 的数组中（实际上该数组包装在字典结构中），这是一个 Json 格式的数据，但是里面还有 html 转义字符需要处理

接下来我们就来写一个方法提取出历史文章数据，分三个步骤，首先用正则提取数据内容，然后 html 转义处理，最终得到一个列表对象，返回最近发布的10篇文章。

```
def extract_data(html_content):
    """
    从html页面中提取历史文章数据
    :param html_content 页面源代码
    :return: 历史文章列表
    """
    import re
    import html
    import json

    rex = "msgList = '({.*?})'"
    pattern = re.compile(pattern=rex, flags=re.S)
    match = pattern.search(html_content)
    if match:
        data = match.group(1)
        data = html.unescape(data)
        data = json.loads(data)
        articles = data.get("list")
        for item in articles:
            print(item)
        return articles
```

最终提取出来的数据总共有10条，就是最近发表的10条数据，我们看看每条数据返回有哪些字段。

```
article = {'app_msg_ext_info':
                {'title': '11月赠书，总共10本，附Python书单
                 'copyright_stat': 11,
                 'is_multi': 1,
                 'content': '',
                 'author': '刘志军',
                 'subtype': 9,
                 'del_flag': 1,
                 'fileid': 502883895,
                 'content_url': 'http:\\/\\/mp.weixin.q
                 ''
                 'digest': '十一月份赠书福利如期而至，更多惊
                 'cover': 'http:\\/\\/mmbiz.qpic.cn\\..
                 'multi_app_msg_item_list': [{'fileid':
                                             'content_
                                             'content'
                                             'cover':
                                             'del_flag
                                             'digest':
                                             'source_u
                                             'title':
                                             'author':
                                             }],
                 'source_url': 'https:\\/\\/github.com\
                 },
      'comm_msg_info': {'datetime': 1511827200,
                        'status': 2,
                        'id': 1000000161,
                        'fakeid': '2393828411',
                        'content': '',
                        'type': 49}}
```

我们结合下面这张图来看:

上面这张图是微信作为单次推送发给用户的多图文消息，有发送时间对应 `comm_msg_info.datetime` ， `app_msg_ext_info` 中的字段信息就是第一篇文章的字段信息，分别对应:

- title：文章标题
- content_url：文章链接
- source_url：原文链接，有可能为空
- digest：摘要
- cover：封面图
- datetime：推送时间

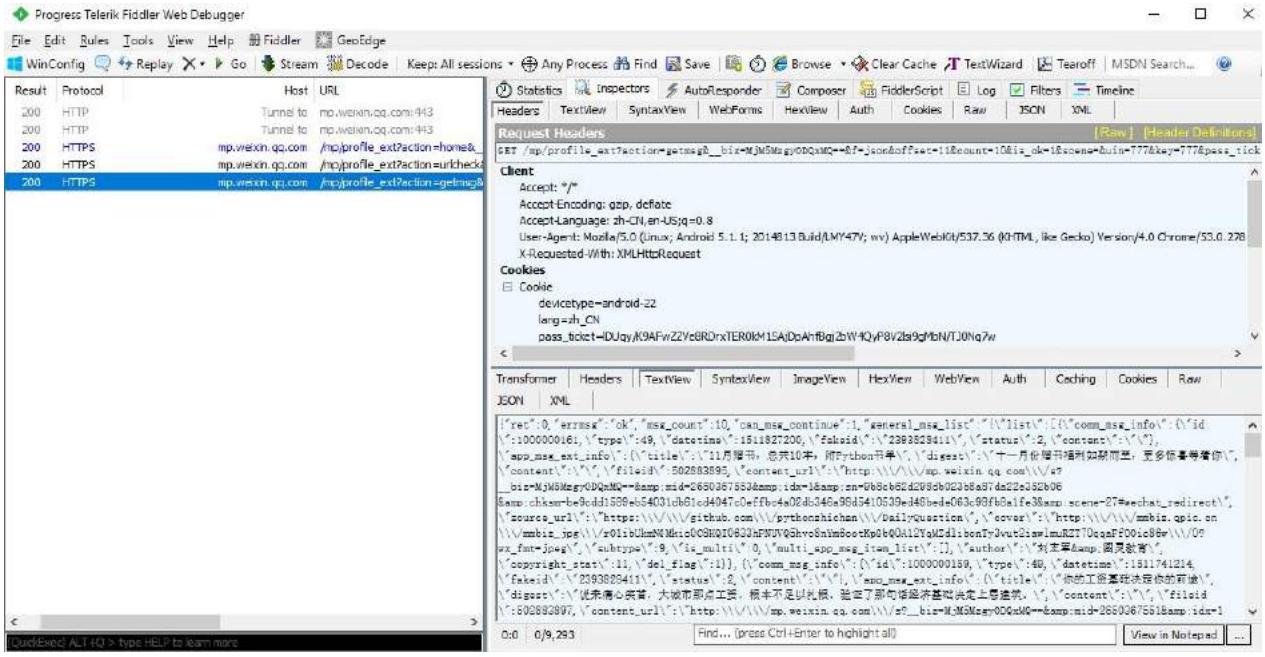
后面几篇文章以列表的形式保存在 `multi_app_msg_item_list` 字段中。

到此，公众号文章的基本信息就抓到了，但也仅仅只是公众号的前10条推送，如何获取更多历史文章等问题放在下节讲解。

本节完整代码可以在GitHub仓库weixincrawler_v查看。

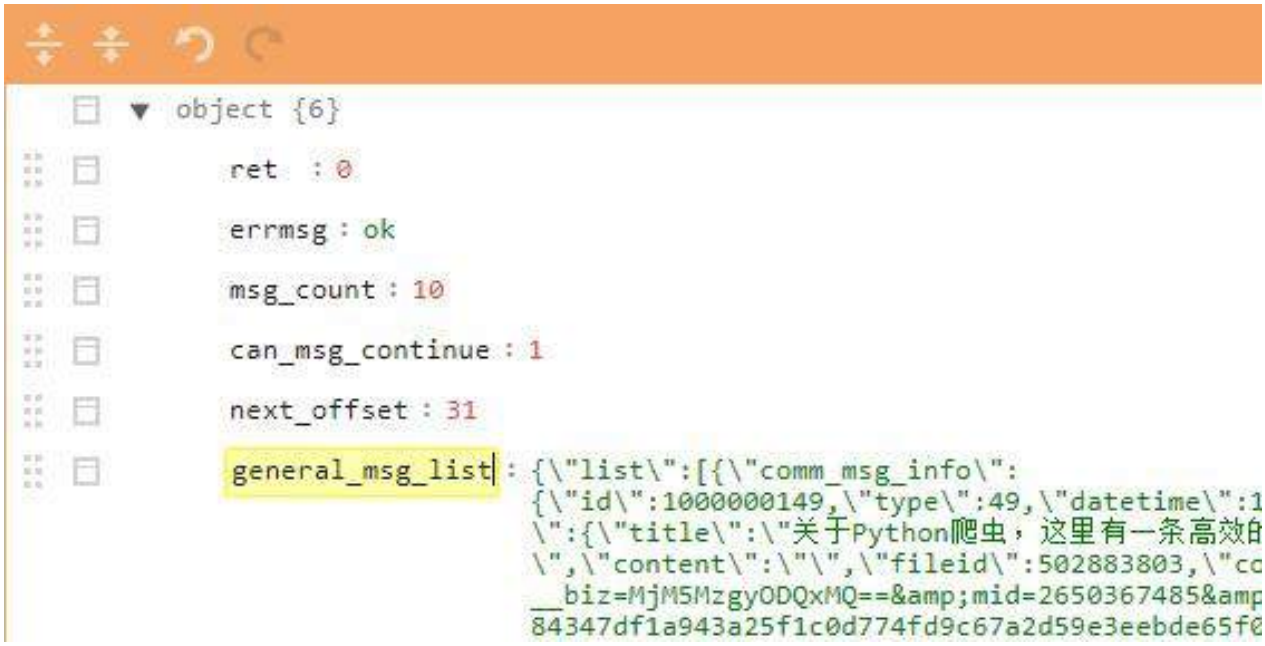
抓取公众号所有历史文章

我们按照第三节使用 Fiddler 抓包方式，打开手机某个微信公众号历史文章列表，上拉加载更多，找到加载更多文章的 URL 请求地址，你会看到 Fiddler 会有一个加载更多文章列表的请求。



分析抓包数据

该接口返回的数据是 JSON 格式，这种数据格式处理起来非常方便，首先我们把数据拷贝到 Chrome 插件 JSON Editor 或者找一个JSON Online Formatter 对返回的数据进行格式化处理，以便查看每个字段所代表的意义。



你可以大概猜出来每个字段的意思

- ret：请求是否成功，0就表示成功
- msg_count： 返回的数据条数
- can_msg_continue： 是否还有下一页数据
- next_offset： 下一次请求的起始位置
- general_msg_list：真实数据

`general_msg_list` 是历史文章里面的基本信息，包括每篇文章的标题、发布时间、摘要、链接地址、封面图等，而像文章的阅读数、点赞数、评论数、赞赏数这些数据都需要通过额外接口获取。

代码实现

分析完后，用代码实现其实非常简单，按照上节的方式，我们把 URL 和 Header 信息直接从 Fiddler 中拷贝过来。

```
# crawler.py
# -*- coding: utf-8 -*-

import logging
import utils
```

```
import requests

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(__name__)


class WeiXinCrawler:
    def crawl(self):
        """
        爬取更多文章
        :return:
        """
        url = "https://mp.weixin.qq.com/mp/profile_ext?
                "action=getmsg&" \
                "__biz=MjM5MzgyODQxMQ==" \
                "&f=json&" \
                "offset=11&" \
                "count=10&" \
                "is_ok=1" \
                "&scene=124&" \
                "uin=777&key=777&" \
                "pass_ticket=25llsA6zWUPC9KHOvP4oE%2BQv
                "appmsg_token=936_qKN8I1KSEO%252BWB2YUS
                "x5=0&" \
                "f=json"


        headers = """
                    Host: mp.weixin.qq.com
                    .... 省略了，自己补充 ...
                    """
        headers = utils.str_to_dict(headers)
```

```
            response = requests.get(url, headers=headers, v
            result = response.json()
            if result.get("ret") == 0:
                msg_list = result.get("general_msg_list")
                logger.info("抓取数据: offset=%s, data=%s" %
            else:
                # 错误消息
                # {"ret":-3,"errmsg":"no session","cookie_c
                logger.error("无法正确获取内容，请重新从Fiddler
                exit()


if __name__ == '__main__':
    crawler = WeiXinCrawler()
    crawler.crawl()
```

成功爬取了第二页的数据，那么第三页呢，第四页呢？所以，我们还需要对该方法进行重构，使得它可以抓取公众号全部历史文章。通过字段 can_msg_continue 确定是否继续抓取，再结合 next_offset 就可以加载更多数据，我们需要把 url 中可变的参数 offset 用变量来代替，递归调用直到 `can_msg_continue` 为 0 说明所有文章都爬取完了。

```
    def crawl(self, offset=0):
        """
        爬取更多文章
        :return:
        """
        url = "https://mp.weixin.qq.com/mp/profile_ext?
              "action=getmsg&" \
              "__biz=MjM5MzgyODQxMQ==&" \
              "f=json&" \
              "offset={offset}&" \
```

```
                "count=10&" \
                "is_ok=1&" \
                "scene=&" \
                "uin=777&" \
                "key=777&" \
                "pass_ticket=25llsA6zWUPC9KHOvP4oE+QwJ3nS
                "wxtoken=&" \
                "appmsg_token=936_qKN8I1KSEO%2BWB2YUShHV8
                "x5=1&" \
                "f=json".format(offset=offset)  # 请将appr

        headers = """
Host: mp.weixin.qq.com
.... 省略了，自己补充 ...
"""
        headers = utils.str_to_dict(headers)
        response = requests.get(url, headers=headers, v
        result = response.json()
        if result.get("ret") == 0:
            msg_list = result.get("general_msg_list")
            logger.info("抓取数据: offset=%s, data=%s" %
            # 递归调用
            has_next = result.get("can_msg_continue")
            if has_next == 1:
                next_offset = result.get("next_offset")
                time.sleep(2)
                self.crawl(next_offset)
        else:
            # 错误消息
            # {"ret":-3,"errmsg":"no session","cookie_c
            logger.error("无法正确获取内容，请重新从Fiddler
            exit()
```

当 has_next 为 0 时，说明已经到了最后一页，这时才算爬完了一个公众号的所有历史文章，现在把所有的文章摘要数据抓取下来了，但是数据还没有存储，下一节，我们将使用MongoDB将数据进行持久化。

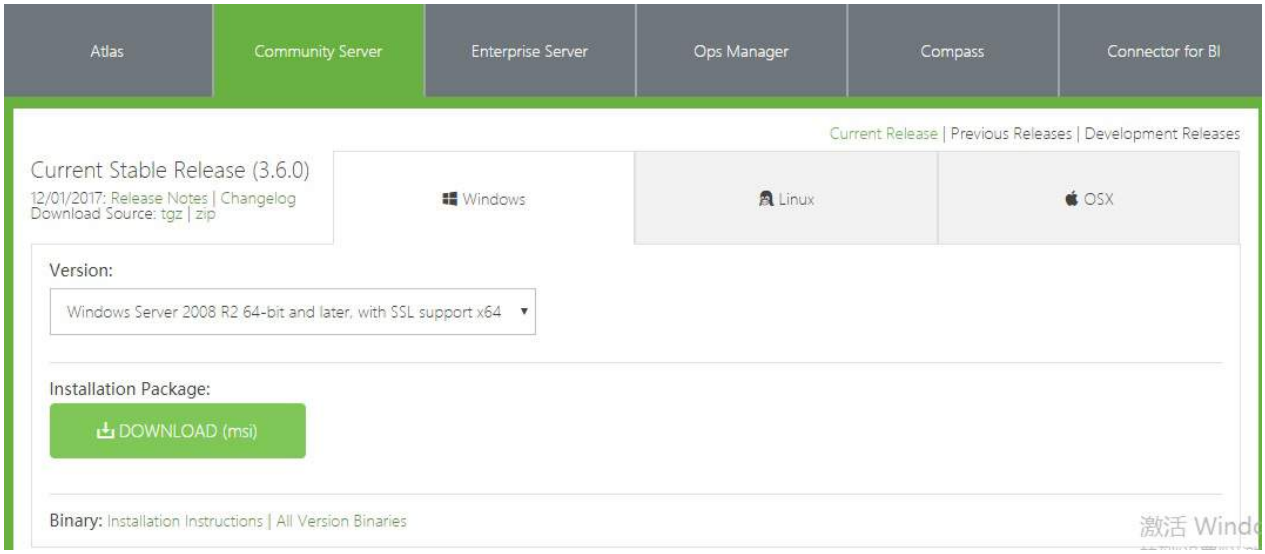
本节完整代码地址：weixincrawler/v0.2

将爬取的文章存储到MongoDB

关于数据的存储有很多选择，最简单的方式就是直接保存到 CSV 文件中，这种方式操作简单，适合数据量少的情况，Python的标准库 `csv` 模块就可以直接支持。如果遇到数据量非常大的情况，就必须要用到专业的数据库系统，你既可以使用 MySQL 这样的关系型数据库，也可以使用 MongoDB 一类的文档型数据库。用Python 操作 MongoDB 非常方便，无需定义表结构就可以直接将数据插入，所以我们在这一节采用用 MongoDB 来存储数据。

MongoDB 安装

MongoDB 目前最新版本是3.6，在官网地址https://www.mongodb.com/download-center#community选择相应平台下载安装。



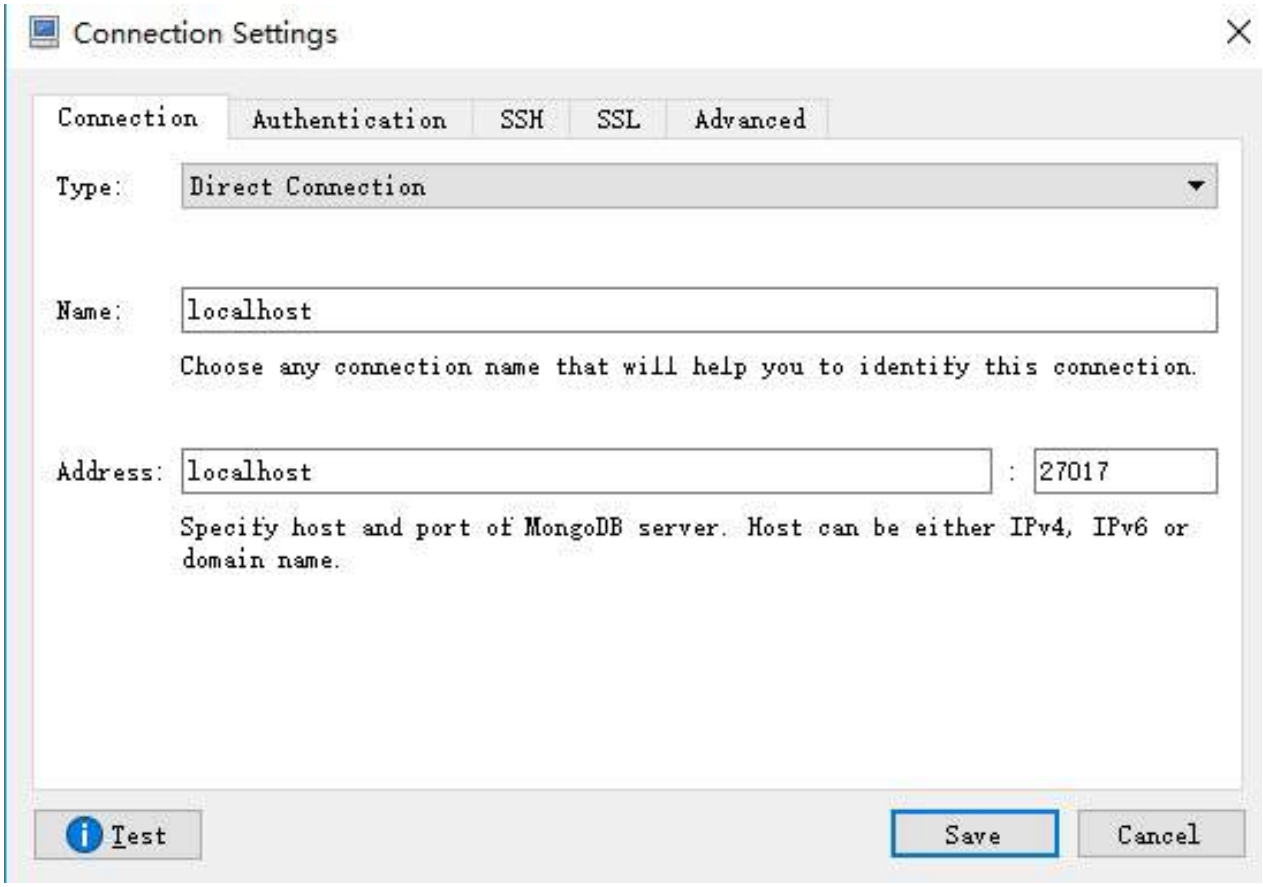
Windows 默认安装在 C:\Program Files\MongoDB\Server\3.6\，macOS 也可以直接通过 brew 命令安装，Linux平台直接下载压缩包解压即可。

```
brew install mongodb --with-openssl
```

启动 MongoDB

```
mongod --dbpath <path to data directory>
```

默认端口是 27017，为了更好的查看数据，我们可以装一个 MongoDB 客户端， 官方自带有 compass，也可以下载第三方工具 Robo 3T https://robomongo.org/，这里推荐大家使用免费的 Robo 3T。



MongoEngine

MongoEngine 是 MongoDB 的 DOM（Document-Object Mapper）框架，一种类似于关系型数据库中的ORM框架，使用它可以更方便并写出简洁的代码

安装

```
$ pip install mongoengine
```

连接

```
from mongoengine import connect
# 连接 mongodb, 无需事先创建数据库
connect('weixin', host='localhost', port=27017)
```

定义数据模型

```
# -*- coding: utf-8 -*-
from datetime import datetime

from mongoengine import DateTimeField
from mongoengine import Document
from mongoengine import IntField
from mongoengine import StringField
from mongoengine import URLField
from mongoengine import connect

__author__ = "liuzhijun"

# 连接 mongodb
connect('weixin2', host='localhost', port=27017)


class Post(Document):
    """
    文章信息
    """
    title = StringField()  # 文章标题
    content_url = StringField()  # 文章链接
    content = StringField()  # 文章内容
    source_url = StringField()  # 原文链接
    digest = StringField()  # 文章摘要
    cover = URLField(validation=None)  # 封面图
```

```
    p_date = DateTimeField()   # 推送时间

    read_num = IntField(default=0)   # 阅读数
    like_num = IntField(default=0)   # 点赞数
    comment_num = IntField(default=0)   # 评论数
    reward_num = IntField(default=0)   # 赞赏数
    author = StringField()   # 作者

    c_date = DateTimeField(default=datetime.now)   # 数据
    u_date = DateTimeField(default=datetime.now)   # 最后
```

数据保存

在第五小节中，我们只是把抓取的数据简单的打印出来，现在我们就把它存数据库，因为抓取的数据中有很多无用的字段，所以，这里我们写一个工具函数叫 `sub_dict` 用于获取指定字段信息。

```
import html
def sub_dict(d, keys):
    return {k: html.unescape(d[k]) for k in d if k in k

d = {"a": "1", "b": 2, "c": 3}
sub_dict(d, ["a", "b"]) # {"a":"1", "b": "2"}
```

获取字典的子字典可以用字典推导式实现，我这里还导入了 `html.unescape` 方法是希望保存到数据库的数据都是经过反转义处理的。

```
    @staticmethod
    def save(msg_list):

        msg_list = msg_list.replace("\/", "/")
        data = json.loads(msg_list)
        msg_list = data.get("list")
        for msg in msg_list:
            p_date = msg.get("comm_msg_info").get("date
            msg_info = msg.get("app_msg_ext_info")  # 
            if msg_info:
                WeiXinCrawler._insert(msg_info, p_date)
                multi_msg_info = msg_info.get("multi_ap
                for msg_item in multi_msg_info:
                    WeiXinCrawler._insert(msg_item, p_c
            else:
                logger.warning(u"此消息不是图文推送，data=

    @staticmethod
    def _insert(item, p_date):
        keys = ('title', 'author', 'content_url', 'dige
        sub_data = utils.sub_dict(item, keys)
        post = Post(**sub_data)
        p_date = datetime.fromtimestamp(p_date)
        post["p_date"] = p_date
        logger.info('save data %s ' % post.title)
        try:
            post.save()
        except Exception as e:
            logger.error("保存失败 data=%s" % post.to_js
```

如果是文字推送就没有 `app_msg_ext_info` 字段，无需保存， `multi_app_msg_item_list` 是多图文推送字段，而且和外层的 `app_msg_ext_info` 字段是一致的，有标题、封面图、摘要、链接等信息，所以我们把插入数据库的代码 `_insert` 作为私有方法抽离出来共用。

最后我们看一下保存的数据。

```
db.getCollection('post').find({}).sort({"p_date":-1})
```

post 0.006 sec.

Key	Value	Type
(1) ObjectId("5a4539c7a54d75940d090fd...	{ 14 fields }	Object
_id	ObjectId("5a4539c7a54d75940d090fd3")	ObjectId
title	5个酷毙的Python工具	String
content_url	http://mp.weixin.qq.com/s?__biz=MjM5MzgyODQ...	String
source_url		String
digest	工欲善其事必先利其器，一个好的工具能让起到事半...	String
cover	http://mmbiz.qpic.cn/mmbiz_jpg/rO1ibUkmNGMm...	String
p_date	2017-12-27 08:00:00.000Z	Date
read_num	0	Int32
like_num	0	Int32
comment_num	0	Int32
reward_num	0	Int32
author	刘志军	String
c_date	2017-12-29 02:36:55.787Z	Date
u_date	2017-12-29 02:36:55.787Z	Date
(2) ObjectId("5a4539c7a54d75940d090fd...	{ 14 fields }	Object
(3) ObjectId("5a4539c7a54d75940d090fd...	{ 14 fields }	Object
(4) ObjectId("5a4539c7a54d75940d090fd...	{ 14 fields }	Object

小结

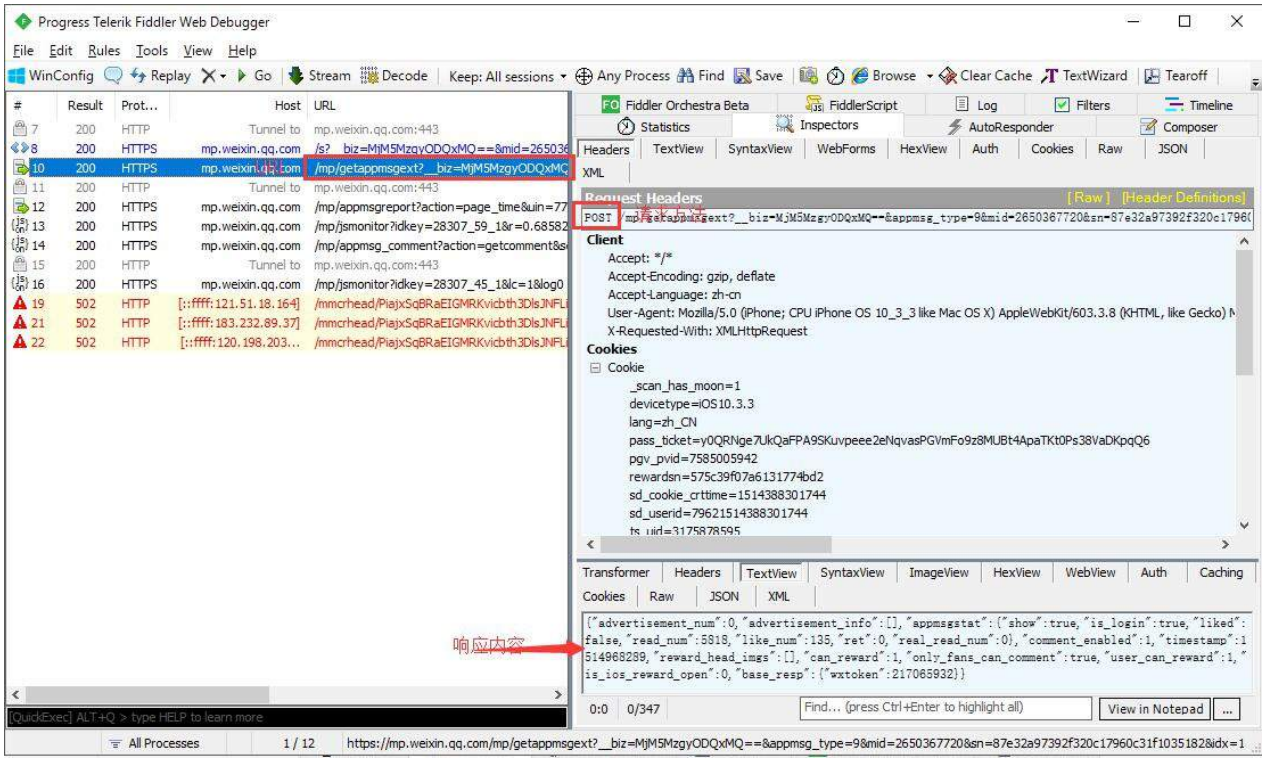
本节完成代码在GitHub v0.3，这小节我们主要熟悉是 Mongodb 以及 如果用 Python连接 Mongodb 进行数据存储，推荐两个资源，第一个是：《MongoDB 入门指南》，第二个是 MongoEngine 教程，如果你想进行系统的学习 MongoDB，推荐两本书籍《MongoDB权威指南》和《MongoDB实战》

获取文章阅读数、点赞数、评论数、赞赏数

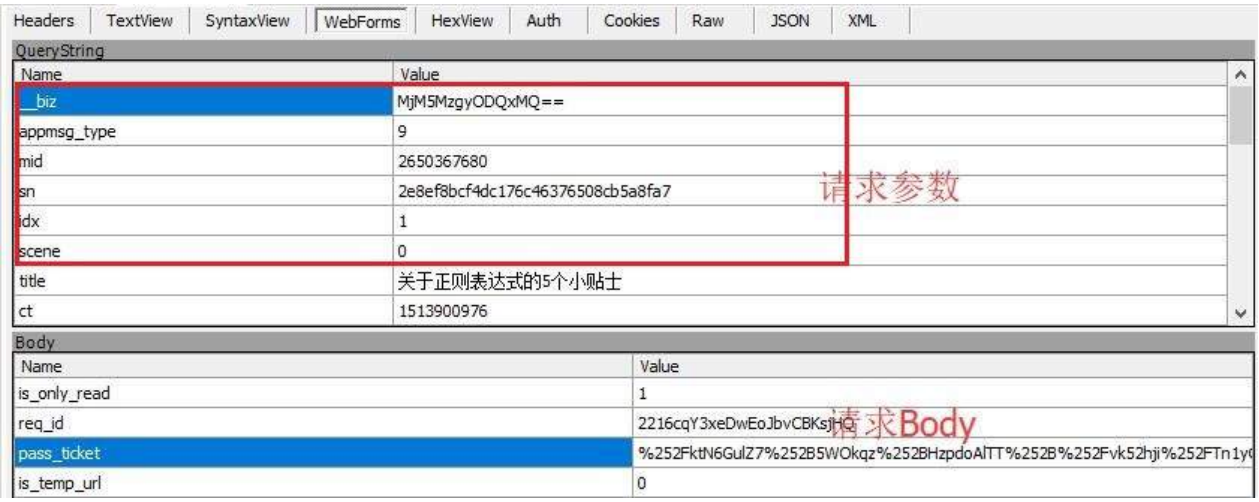
如果只是获取所有文章的基本信息价值并不大，最多能对文章做检索，只有得到文章的阅读数、点赞数、评论数和赞赏数之后数据才有数据分析的价值。这节就来讨论如何获取这些数据。

抓包分析

点开任意一篇文章，通过 Fiddler 或 Charles 抓包分析，逐个分析每个请求，通过观察发现获取文章阅读数、点赞数的URL接口为（我们命名为 data_url）：https://mp.weixin.qq.com/mp/getappmsgext ，后面有很多查询参数，请求方法为 POST



该请求的查询参数有28个之多，另外还附有请求 Body。



返回的响应数据是JSON格式，根据字段名称基本能猜出其中的意义，阅读数、点赞数、赞赏数都包含在其中

```
{
    "advertisement_num": 0,
    "advertisement_info": [ ],
    "appmsgstat": {
        "show": true,
        "is_login": true,
        "liked": false,
        "read_num": 6395,   # 阅读数
        "like_num": 190,    # 点赞数
        "ret": 0,
        "real_read_num": 0
    },
    "comment_enabled": 1,
    "timestamp": 1514972862,
    "reward_head_imgs": [   # 赞赏头像列表
        "http://wx.qlogo.cn/mmhead/V3bYdzb7P4DLf3e7Xf74
        "http://wx.qlogo.cn/mmhead/Q3auHgzwzM7KF8PIsOic
    ],
    "reward_total_count": 16,   # 赞赏数
    "can_reward": 1,
    "only_fans_can_comment": true,
    "user_can_reward": 1,
    "reward_qrcode_ticket": "%2B%2FfLw%2BXXGQwDD0ik6Gwp
    "base_resp": {
        "wxtoken": 723698581
    }
}
```


确定了请求的URL及查询参数，请求方法，请求体，请求头也能查看到，返回的数据也有了，剩下的问题是如何批量获取不同文章的数据，这需要从请求的 data_url 着手分析。

为了找出 data_url 中查询参数的规律，先对比文章详情的 content_url （就是在上一节得到的文章详情URL）

```
# 文章的URL
content_url = "http://mp.weixin.qq.com/s?" \
              "__biz=MjM5MzgyODQxMQ==&" \
              "mid=2650367413&idx=1&" \
              "sn=637de06b162c21605eef3db41ee4a1bb&" \
              "chksm=be9cdee189eb57f78994371ce1b5b42656
              "scene=27"
```

不得而知， `__biz` ，mid，idx，sn，scene，chksm 是构成一篇文章的完整URL，而文章阅读数的URL是：

```
# 阅读数URL
data_url = "https://mp.weixin.qq.com/mp/getappmsgext?"
           "__biz=MjM5MzgyODQxMQ==&" \
           "appmsg_type=9&" \
           "mid=2650367720&" \
           "sn=87e32a97392f320c17960c31f1035182&" \
           "idx=1&" \
           "scene=27&" \
           "title=2018%20%E5%B9%B4%EF%BC%8C%E5%AD%A6%E7
           "ct=1514505600&" \
           "abtest_cookie=AwABAAoADAANAAkAJIgeAGKIHgD8i
           "devicetype=iOS10.3.3&" \
           "version=/mmbizwap/zh_CN/htmledition/js/appm
           "f=json&" \
           "r=0.341679623927889&is_need_ad=0&" \
           "comment_id=2810810222&" \
           "is_need_reward=1&" \
           "both_ad=0&" \
           "reward_uin_count=27&" \
           "msg_daily_idx=1&" \
           "is_original=0&" \
           "uin=777&" \
           "key=777&" \
           "pass_ticket=y0QRNge7UkQaFPA9SKuvpeee2eNqvas
           "wxtoken=217065932&" \
           "devicetype=iOS10.3.3&" \
           "clientversion=16060123&" \
           "appmsg_token=938_vSjt2FisNybtpLZCYw5DGyn0L2
           "x5=0&" \
           "f=json"
```

对比两个URL，你会发现 content_url 中的参数除了 chksm 其它几个参数都在 data_url 中，我们把 content_url 中的参数替换到 data_url 再来验证请求会不会正常返回数据。至于其他参数要不要改，怎么改我们先放一边（这是一个不断猜想、验证的过程，经过我的多次试验，除了 appmsg_token 有一定的时效之外，其它值可以保持不变，也就是说不同的文章，只要把content_url中的参数替换到 data_url 中就可以获取该文章的数据了。）

代码实现

```
    @staticmethod
    def update_post(post):
        """
        post 参数是从mongodb读取出来的一条数据
        稍后就是对这个对象进行更新保存
        :param post:
        :return:
        """

        # 这个参数是我从Fiddler中拷贝出 URL，然后提取出查询参
        # 稍后会作为参数传给request.post方法
        data_url_params = {'__biz': 'MjM5MzgyODQxMQ==',
                           'sn': '08ce54f6f36873e74c638
                           'title': '2017%E5%B9%B4%EF%B
                           'ct': '1514796292',
                           'abtest_cookie': 'AwABAAoADA
                           'devicetype': 'android-24',
                           'version': '/mmbizwap/zh_CN/
                           'r': '0.6452677228890584', '
                           'is_need_reward': '1', 'both
                           'is_original': '0', 'uin': '
                           'pass_ticket': 'mXHYjLnkYux1
                           'wxtoken': '1805512665', 'cl
```

```
                                'appmsg_token': '938_VN3Rr7C
                                'x5': '1'}

        # url转义处理
        content_url = html.unescape(post.content_url)
        # 截取content_url的查询参数部分
        content_url_params = urlsplit(content_url).quer
        # 将参数转化为字典类型
        content_url_params = utils.str_to_dict(content_
        # 更新到data_url
        data_url_params.update(content_url_params)
        body = "is_only_read=1&req_id=03230SZyTR8kQlPVk
        data = utils.str_to_dict(body, "&", "=")

        headers = """
Host: mp.weixin.qq.com
Connection: keep-alive
Content-Length: 155
Origin: https://mp.weixin.qq.com
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Linux; Android 7.0; M1 E Build
Content-Type: application/x-www-form-urlencoded; charse
Accept: */*
Referer: https://mp.weixin.qq.com/s?__biz=MjM5MzgyODQxM
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,en-US;q=0.8
Cookie: rewardsn=05c38771473771b68376; wxtokenkey=92c03
Q-UA2: QV=3&PL=ADR&PR=WX&PP=com.tencent.mm&PPVN=6.6.1&T
Q-GUID: 0fd685fa8c515a30dd9f7caf13b788cb
Q-Auth: 31045b957cf33acf31e40be2f3e71c5217597676a9729f1
        """

        headers = utils.str_to_dict(headers)
```

```
        data_url = "https://mp.weixin.qq.com/mp/getappm

        r = requests.post(data_url, data=data, verify=F

        result = r.json()
        if result.get("appmsgstat"):
            post['read_num'] = result.get("appmsgstat")
            post['like_num'] = result.get("appmsgstat")
            post['reward_num'] = result.get("reward_tot
            post['u_date'] = datetime.now()
            logger.info(" 「%s」 read_num: %s like_num: %
                        (post.title, post['read_num'],
            post.save()
        else:
            logger.warning(u"没有获取的真实数据，请检查请求参
```

需要注意的是 iOS 没有赞赏功能，所以如果要获取赞赏数据，我们必须用 Android 设备来抓取数据。现在就来遍历更新每条数据的内容：

```
crawler = WeiXinCrawler()
  for post in Post.objects(read_num=0):
      crawler.update_post(post)
      time.sleep(1) # 防止恶意刷
```

不出意外的话，能正常获取到数据，在抓取的过程中，微信会有反爬虫限制，爬了一段时间后，返回的数据成了：

```
{"base_resp":{"ret":301,"errmsg":"default"}}
```

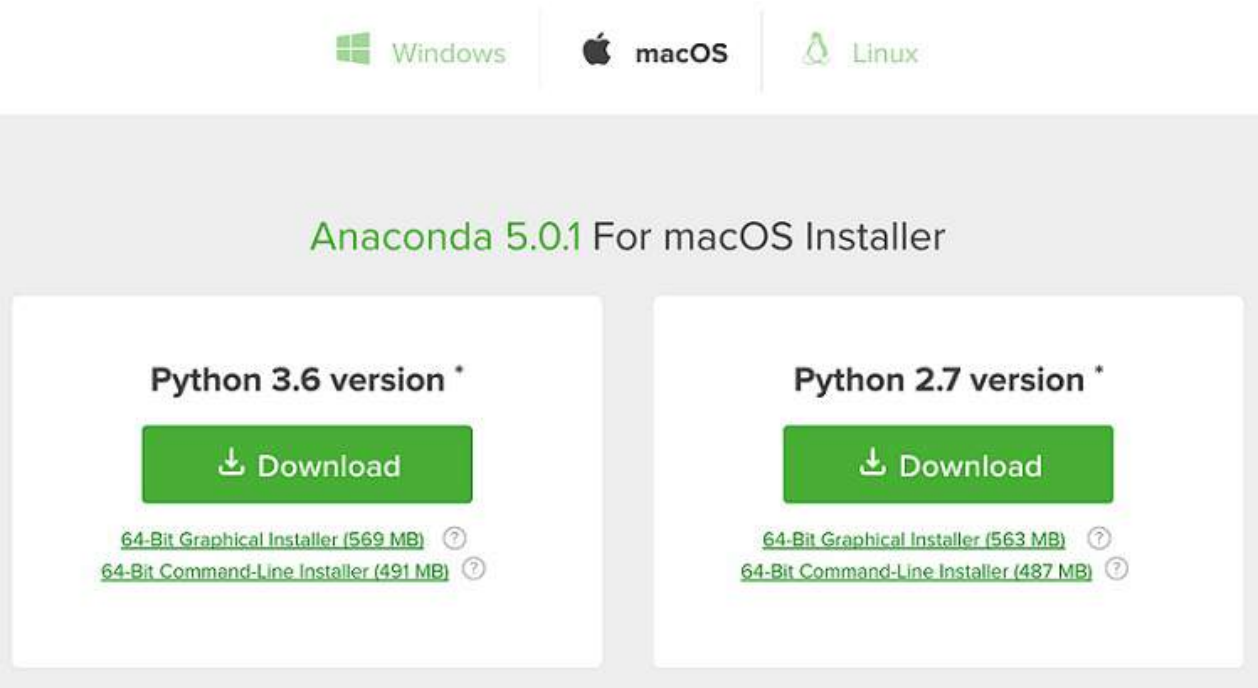
这个时候需要休息一会儿才能继续爬虫，换IP也没有，如果需要大规模爬虫就有必要准备多个微信号来操作。本节完整代码：weixincrawler-v0.4

搭建数据分析环境：Anaconda、Jupyter Notebook

Anaconda 是一个针对数据分析领域 Python 发行版本，它提供了包管理（packages）工具和虚拟环境（environment）管理， `conda` 命令可用于安装、卸载、更新包、创建不同版本的 Python 独立环境，可用于替换 `pip` 和 `virtualenv` 这两个工具。此外，Anaconda 自带了很多数据科学的依赖包以及Juypter Notebook 等工具。

Anaconda 下载安装

可直接从 Anaconda 官方网站进行下载，选择 Python3.6 的版本，因为 Python2.7即将被废弃，下载后根据提示安装即可



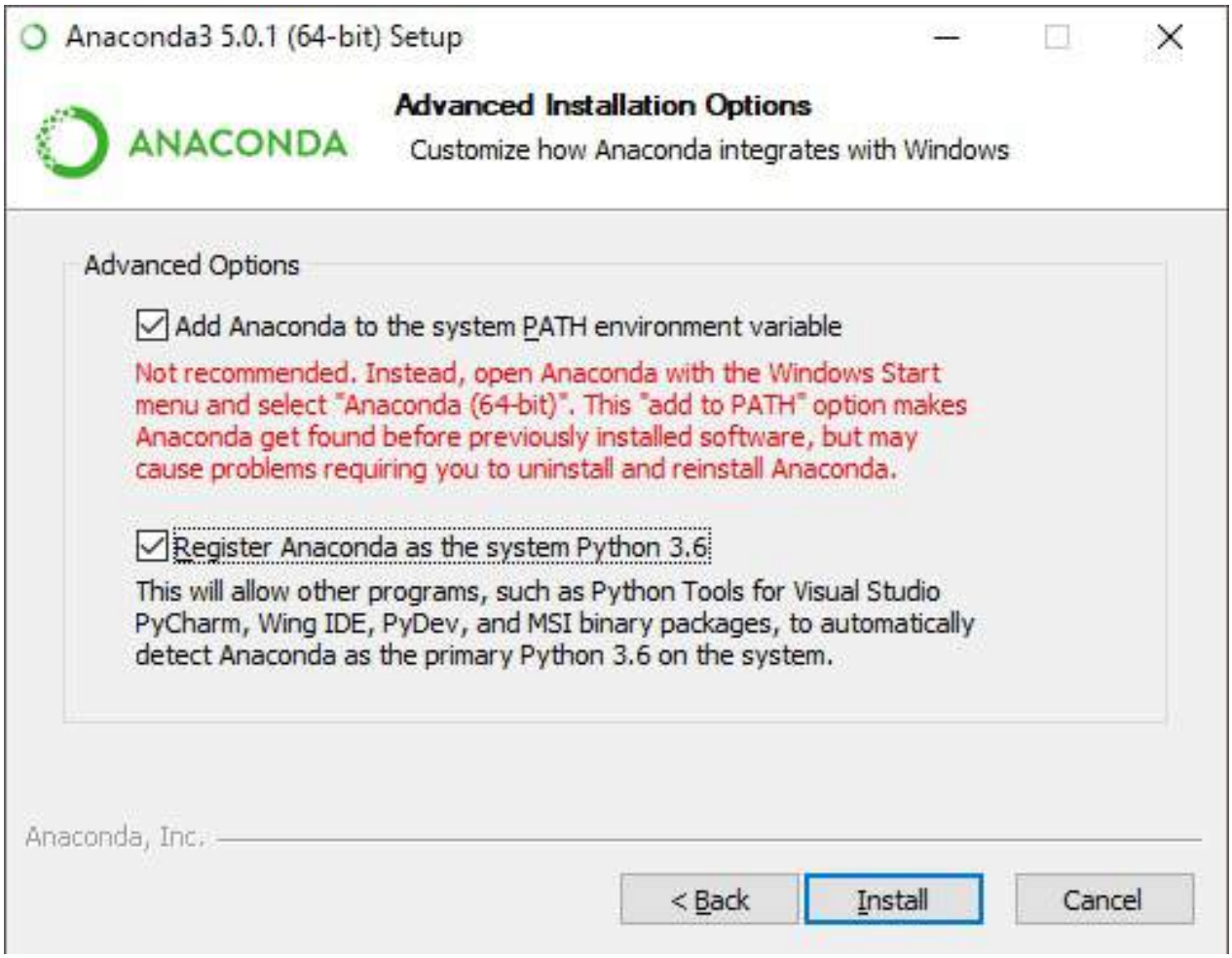
macOS/Linux 安装完成之后会自动把 Anaconda 添加到 PATH 环境变量（在 ~/.bash_profile 文件中可以看到），如果你的终端默认 SHELL 不是 bash 的话（用 `echo $SHELL` 查看默认 shell 是啥），加了系统也找不到 conda 命令，比如我的 mac 默认 shell 是 zsh ，需要把下面这行添加到 ~/.zshrc 文件中

```
# added by Anaconda3 5.0.1 installer
export PATH="/Users/你的用户名/anaconda3/bin:$PATH"
```

再检查 conda 命令是否能用

```
conda -V
conda 4.3.30
```

Windows 平台安装的时候请自动勾选加入 PATH 路径，如果安装的时候没有勾选，要手动找到 Anoconda 的安装路径加入到 PATH 变量中，否则一样找不到 conda 命令。



为了使用 conda 安装包的过程中加快速度，可以把镜像地址修改为国内清华大学的镜像：编辑 ~/.condrc，（Windows 是在C:\Users\你的用户名.condrc，如果没有该文件就创建一个），添加内容：

```
channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
ssl_verify: false
show_channel_urls: true
```

常用 conda 命令

包管理

```
# 查看帮助
conda -h
# 查看conda版本
conda --version
# 安装 matplotlib
conda install matplotlib
# 查看已安装的包
conda list
# 包更新
conda update matplotlib
# 删除包
conda remove matplotlib
```

环境管理

```
# 基于python3.6版本创建一个名字为test的pytyhon独立环境
conda create --name test python=3.6
# 激活此环境
activate test
source activate test # linux/mac
# 退出当前环境
deactivate test
# 删除该环境
conda remove -n test --all
# 或者
conda env remove  -n test

# 查看所有安装的python环境
conda info -e
test                  *  D:\Programs\Anaconda3\envs\test
root                     D:\Programs\Anaconda3 (安装 cor
```

其他命令

```
# 更新conda本身
conda update conda
# 更新anaconda 应用
conda update anaconda
# 更新python，假设当前python环境是3.6.1，而最新版本是3.6.2，
conda update python
```

安装完 Anoconda 之后，Jupyter Notebook 也装好了。

Jupyter Notebook 是一个强大的数据分析工具，你可以在上面写代码、运行代码、写文档、列方程式、做数据可视化展示。 正如其名，它就像一个草稿本可以在上面随意地涂写改改画画，画错了还可以擦除重做。

启动jupyter

在命令行直接输入：

```
jupyter notebook
```

Jupyter 启动成功后，在浏览器中会自动打开 notebook 的主界面，新建一个notebook时要点击右上角的 `New` ，选择 Python3 ，这里的 Python3 就是 jupyter 的内核，是安装 Anaconda 的时候的名字为root的默认 python 环境。

新建了 notebook 之后你就可以在单元格里面写代码或者写 markdown 文档，或者基于用 matplotlib 制图。

补充

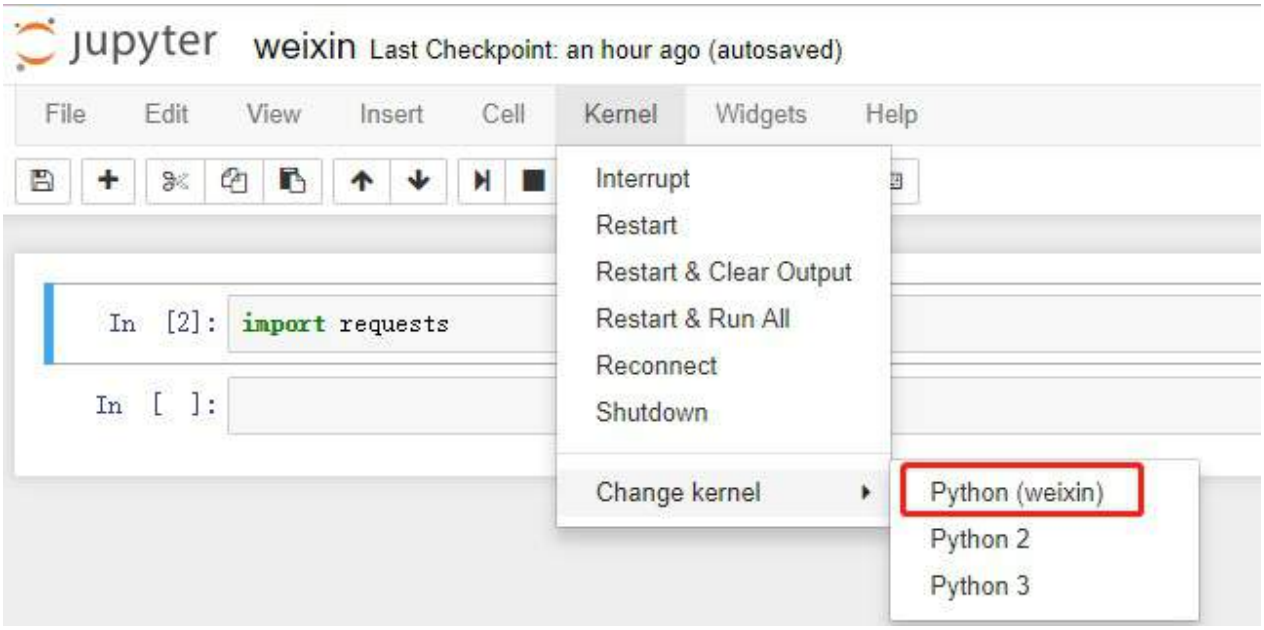
如何查看 jupyter 使用了哪些 kernel

```
 ~ jupyter kernelspec list

Available kernels:
  weixin      /Users/xxx/Library/Jupyter/kernels/weixin
  python3     /Users/xxx/anaconda3/share/jupyter/kernels
```

如何新增 kernel

```
# 创建python环境
conda create -n weixin python=3.6
# 激活
source activate weixin
# 加入到juypter
python -m ipykernel install --user --name weixin --disp
```



新增了 kernel 之后，你可以在不同的 kernel 之间切换运行代码，本质上 kernel 还是 Python 的虚拟环境。