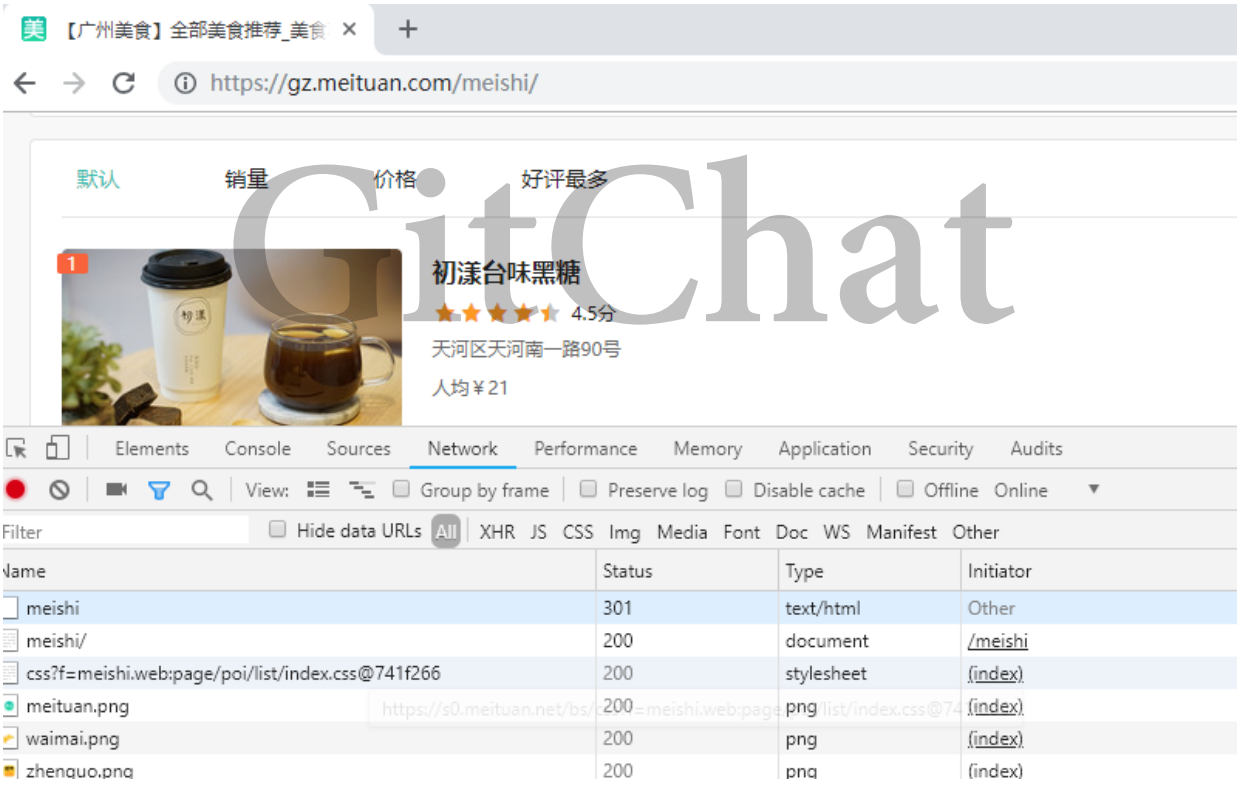
实战 Python 网络爬虫：美团美食商家信息和用户评论

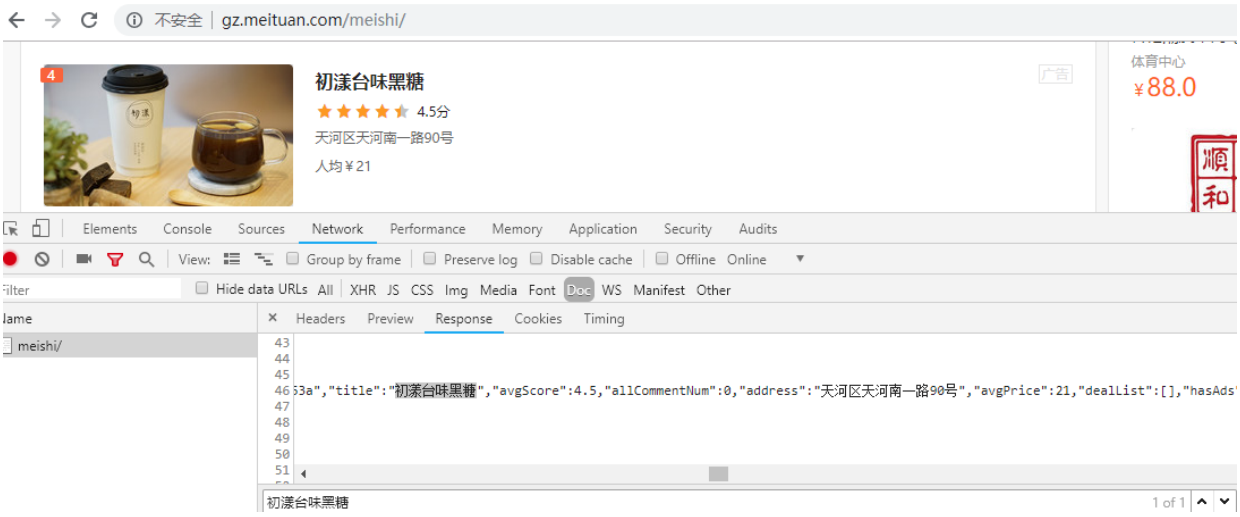
一、网站分析及项目设计

美食是人类的毕生追求，说到美食，我们总会想起美团美食，面对类型众多的商家，应如何选择优质的商家，使消费最大合理化。在本 Chat 里，将讲述如何爬取美团商家信息。

废话不多说，我们直接在浏览器打开美团美食的网址，然后打开谷歌的开发者工具，并刷新网页，重新捕捉请求资源，如图所示：



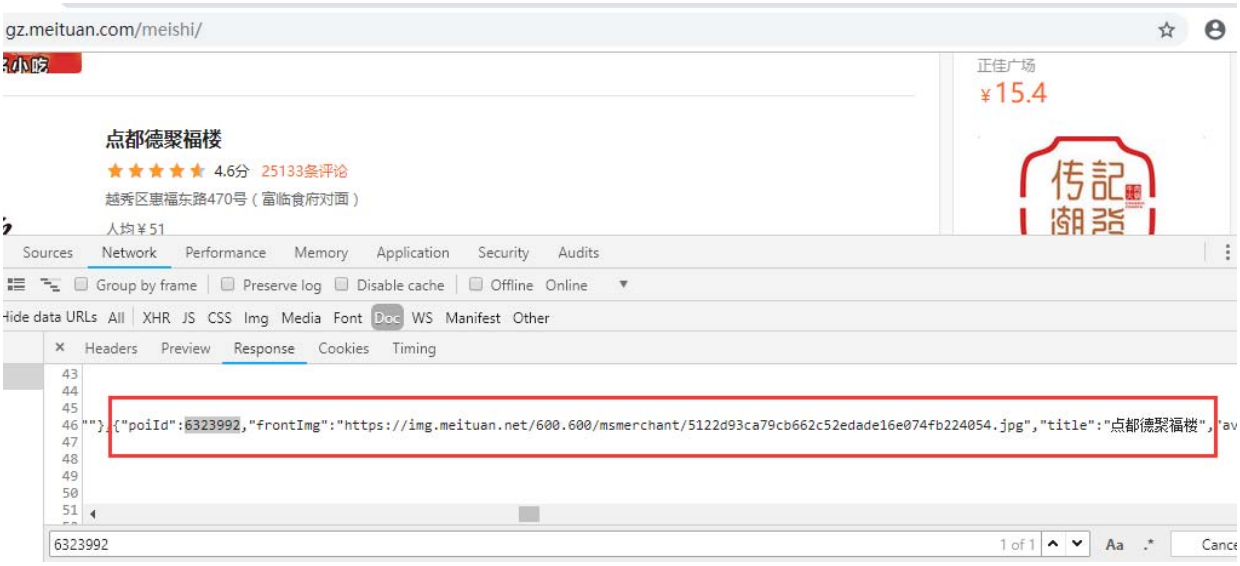
根据店名在 Network 选项卡的各个分类标签下查找数据所在的 HTML 源码位置，在每个请求信息的 Response 下使用 Ctrl+F 快速查找店名（初漾台味黑糖），最终在 Doc 标签下找到相关信息，如图所示：



从图上的信息可以看到，地址栏的 gz 代表区域“广州”，全国的美食商家是以城市进行划分的。而商家的数据是以 JSON 格式表示，网页上显示的信息都可以在此找到。在这个网页中，我们是要查找这个商家的 URL 地址，从而进入商家详细页。

但从美团美食的首页中，我们能获取的信息就这么多，因此，我们先访问店家详细页，发现商家详细页的 URL 地址带有一串数字。不同的商家，数字内容都不一样，如图所示：

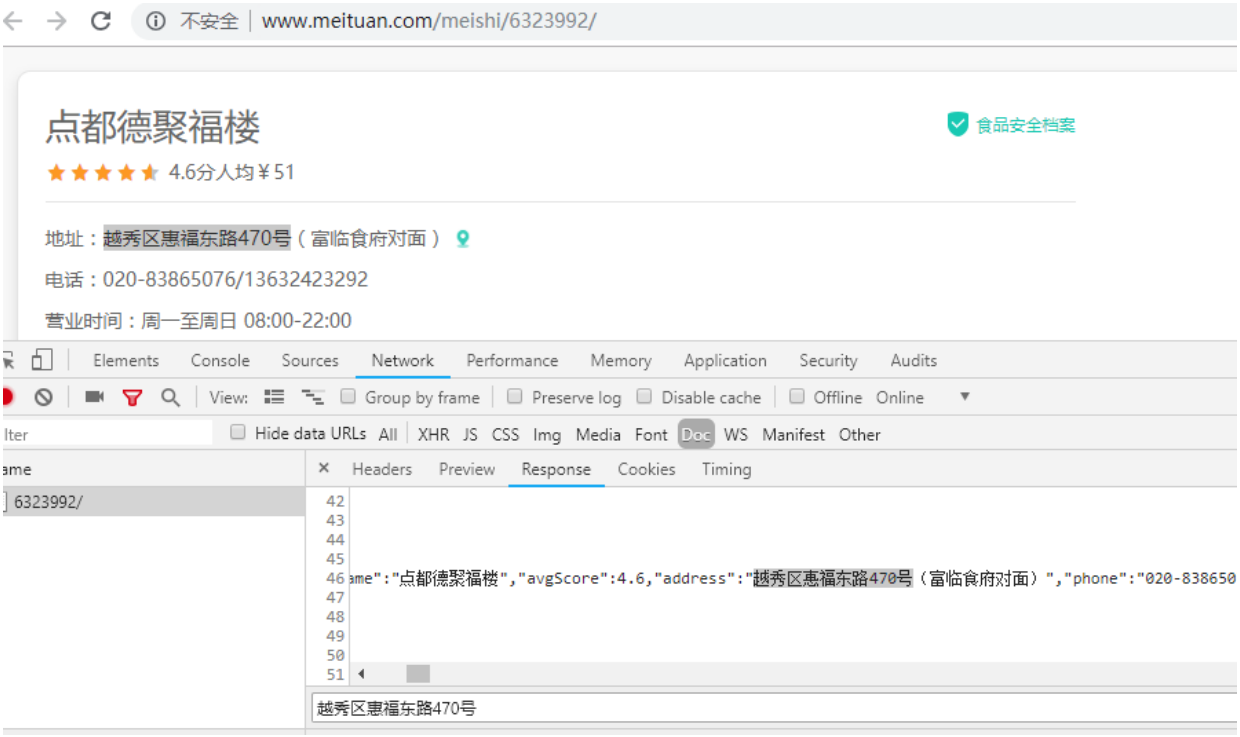
通过对比发现，每个商家详细页的 URL 地址只有末端的数字串是不相同的，这应该是美团给商家标记的 id，我们取其中一个商家 id 回到美团首页查找，发现可找到相关信息，如图所示：



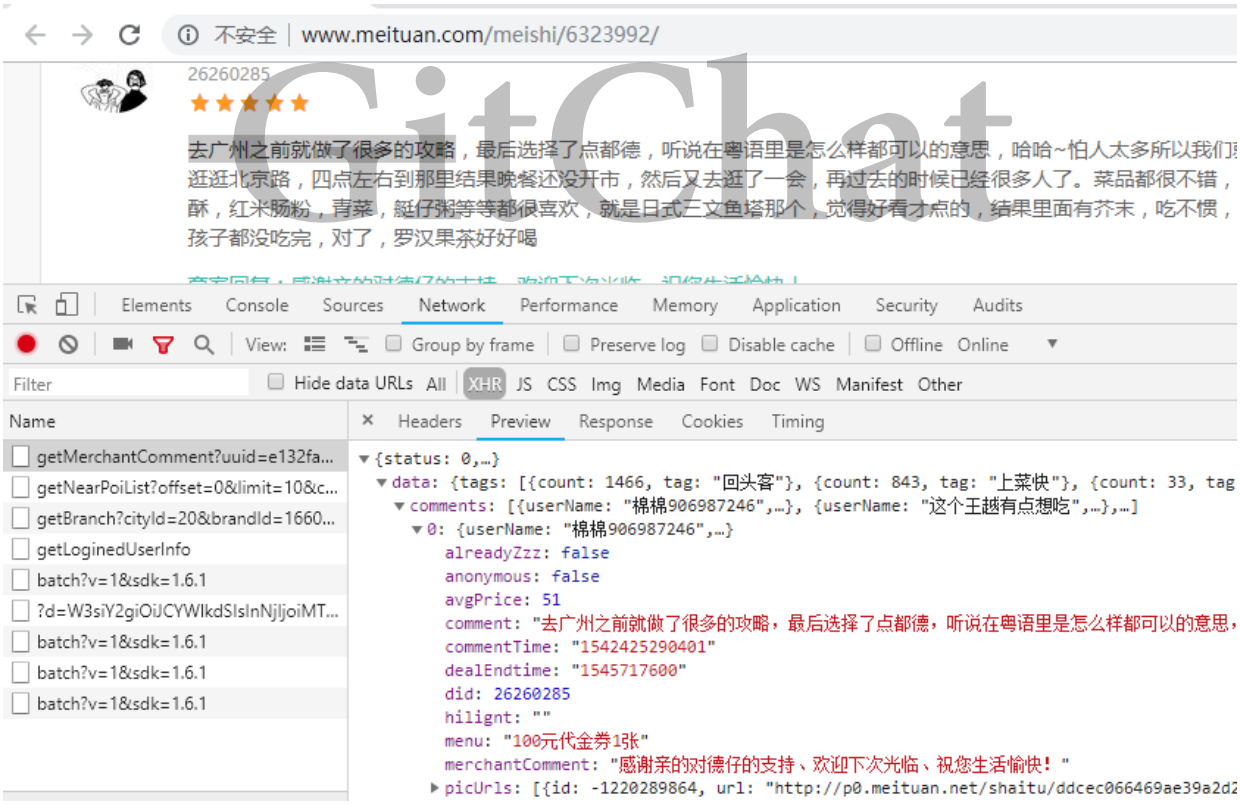
根据上述分析，我们可以在美团美食首页里获取商家 id，通过 id 来构建商家详细页的 URL 地址。

得到了商家详细页的 URL 地址后，下一步是在商家详细页里进行数据爬取。数据爬取分为两部分：商家信息和顾客评论，如图所示：

首先，我们找出商家信息所在的请求信息，在开发者工具的 Network 选项卡的 doc 标签下找到相关信息，商家信息是在 doc 标签下找到，并且也是以 JSON 格式表示，如图所示：



接着是分析顾客评论所在的请求信息，最终在 XHR 标签下找到相关的请求信息，如图所示：



综合上述，我们需要从三个请求信息里获取数据，三个请求信息的说明如下：

- 美团美食的首页地址，获取每个商家的 id
- 商家详细页地址，获取商家信息
- 顾客评论的 AJAX 接口，获取顾客评论信息

目前只是简单分析了三个请求信息，每个请求的请求参数和请求头会在功能实现的过程中详细讲解。

根据现有的分析，我们创建项目文件夹 meituan，项目文件分别有 Insql.py、meishi.conf 和 meishi.py，文件说明如下：

- Insql.py 是将数据入库处理，由 ORM 框架 SQLAlchemy 实现
- meishi.conf 设置区域信息，如广州（gz）或北京（bj）等，从而控制爬虫的爬取方向
- meishi.py 实现爬虫功能

二、爬取所有商家信息

简单分析网页后，接下来我们先实现所有商家的信息爬取。由于商家详细页只需要商家 id 即可，因此爬取所有商家信息只需爬取商家 id 即可。

从美团美食的首页得知，其 URL 地址的“gz”代表广州。首页的底部设有分页功能，当点击第二页的时候，URL 末端新增下级目录 pn2，第三页的下级目录为 pn3，以此类推，新增的下级目录代表分页的页数。当遍历 32 次即可得到当前城市所有美食商家信息，如图所示：

根据 URL 的变化规律和商家信息的 HTML 源码结构，所有商家信息的爬取功能定义为函数 get_all()，函数参数 city_list 代表各个城市信息并以字符串表示，如 gz、bj 等，函数代码如图所示：

```
def get_all(city_list):
    url = 'http://%s.meituan.com/meishi/pn%s/'
    for city in city_list.split(','):
        # 每个城市只显示32页的美食信息，因此循环32次
        for i in range(32):
            headers = {
                'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36'
                              '(KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36',
                'Upgrade-Insecure-Requests': '1',
                'Host': '%s.meituan.com' % (city),
                'Referer': 'http://%s.meituan.com/meishi/' % (city)}
            r = requests.get(url % (city, i + 1), headers=headers)
            # 获取每页所有的美食商家的id，uuid是唯一的，不同电脑访问会生成不同的uuid
            find_poiId = re.findall('"poiId":(\d+.*?),', r.text)
            find_uuid = re.findall('"uuid":"(.*?)",', r.text)
            # 调用函数get_info()，将每页所有的商家ID
            get_info(find_poiId, city, find_uuid)
            print(find_poiId)
```

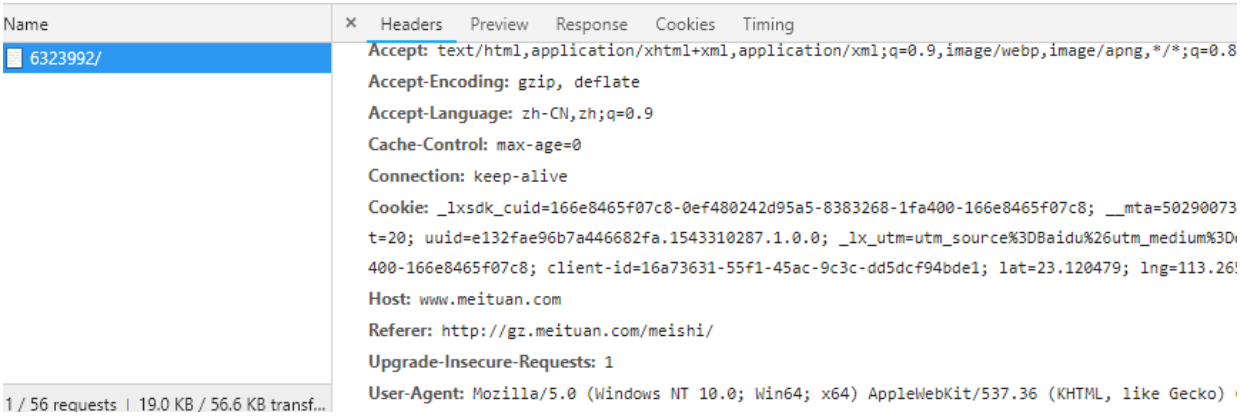
函数 get_all() 所实现的功能说明如下：

- 首先将参数 city_list 以英文逗号进行截取，得到各个城市的拼音缩写。
- 然后遍历各个城市，分别爬取各个城市的美食信息，每个城市只显示 32 页的美食信息，因此需要遍历 32 次。
- 每次遍历都会对当前分页发送 HTTP 请求，请求头设有 Upgrade-Insecure-Requests、Host 和 Referer 属性，这些属性最好写入请求头，这样可以避开反爬虫检测。特别是 Host 属性，因为 URL 的域名设有城市信息，如 gz.meituan.com，而 Host 属性是为 URL 指定相应的域名，使其一一对应。
- 从当前请求中获取响应内容，并用正则表达式提取当前分页所有的商家 id（即 find_poiId）以及访客信息 find_uuid。
- 调用函数 get_info()，将爬取的数据作为函数参数传入。函数 get_info() 是进入商家详细页，爬取商家的基本信息。

三、分别爬取每个商家的信息和用户评论信息

在函数 get_all() 里，我们调用了函数 get_info()，它是进入访问商家详细页的，主要爬取商家的基本信息。商家详情页的 URL 地址为 http://www.meituan.com/meishi/%s/，其中 %s 代表商家 id。

注意：如果对商家详细页发送 HTTP 请求，这里涉及了一个反爬虫机制——Cookies 的使用，我们查看该请求的请求头内容。如图所示：



商家详细页的请求头与一般的请求头并无太大差异，按照以往的开发模式，首先构架 URL 地址，然后对 URL 发送请求，最后从请求里获取响应内容并提取目标数据。按照该思路，商家的基本信息爬取功能如图所示：

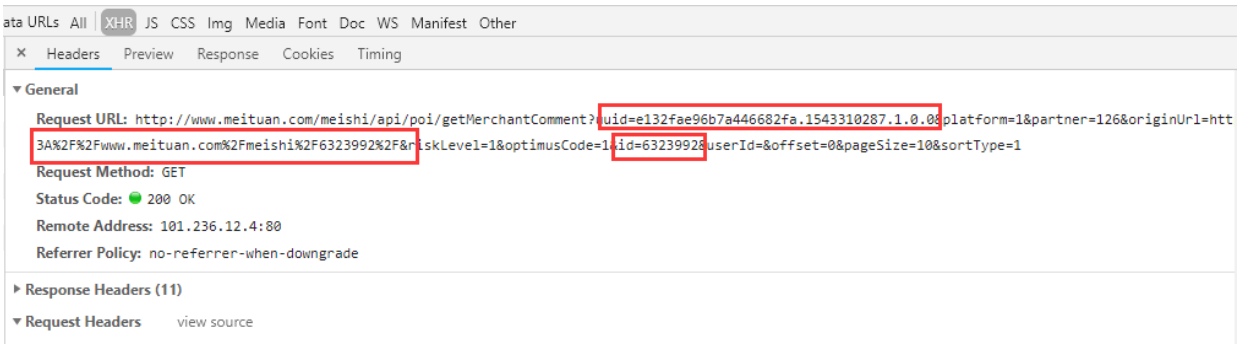
```
def get_info(find_poiId, city, find_uuid):
    for b in find_poiId:
        url = 'http://www.meituan.com/meishi/%s/' %(b)
        headers={
            'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36'
                          '(KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36',
            'Host': 'www.meituan.com',
            'Upgrade-Insecure-Requests':'1',
            }
        r = requests.get(url, headers=headers)
        # 数据清洗
        if r:
            if 'detailInfo' in r.text:
                shop_info = {}
                get_name = r.text.split('detailInfo')[1].split('address')[0]
                shop_info['shop_city'] = city
                shop_info['shop_id'] = b
                value = re.findall('"name":"(.*?)",', get_name)
                shop_info['shop_name'] = value[0] if value else ''
                value = re.findall('"address":"(.*?)",',r.text)
                shop_info['shop_address'] = value[0] if value else ''
                value = re.findall('"phone":"(.*?)",',r.text)
                shop_info['shop_phone'] = value[0] if value else ''
                value = re.findall('"openTime":"(.*?)",',r.text)
                shop_info['shop_openTime'] = value[0] if value else ''
                value = re.findall('"avgScore":(.*?),', r.text)
                shop_info['shop_avgScore'] = value[0] if value else ''
                value = re.findall('"avgPrice":(.*?),', r.text)
                shop_info['shop_avgPrice'] = value[0] if value else ''
                shop_db(shop_info)
                get_comment(find_uuid[0], b)
        time.sleep(3)
```


当运行程序的时候，程序是没有提取到商家信息了，这说明该请求的响应内容不是商家详细页的网页内容，肯定遇到反爬虫检测。在请求头里，除了尚未加入 Cookies 之外，其余属性已添加，因此，我们尝试加入 Cookies，发现可以提取到商家信息。

但是只使用一个 Cookies 也会中断爬取过程，原因在于访问频繁。为了降低访问频繁，引入 Cookies 池，将代码的请求部分进行修改，如下所示：

```
# 获取每间商家信息
def get_info(find_poiId, city, find_uuid):
    r = ''
    for b in find_poiId:
        url = 'http://www.meituan.com/meishi/%s/' %(b)
        headers={
            'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; Win64; x64)'
                          'AppleWebKit/537.36 (KHTML, like Gecko)'
                          'Chrome/64.0.3282.140 Safari/537.36',
            'Host': 'www.meituan.com',
            'Upgrade-Insecure-Requests':'1',
            }
        for n in range(11):
            # 该请求需要Cookies，否则服务器会重复跳转30次，这是反爬虫机制之一
            # 每次请求随机在Cookies列表抽取其中一条，防止一条Cookies重复使用导致服务器禁封
            number = random.randint(0,11)
            cookieStr = cookieList[number]
            cookies = {}
            for i in cookieStr.split(';'):
                cookies[i.split('=')[0]] = i.split('=')[1]
            try:
                r = requests.get(url, headers=headers, cookies=cookies)
                break
            except:pass
        if r:
            if 'detailInfo' in r.text:
```


从函数 get_info() 里可到，它调用了函数 get_comment()，并将商家 ID 和 find_uuid 分别传入，find_uuid 是从函数 get_all()提取出来的数据，这两个函数参数都是构建顾客评论的 AJAX 接口的请求参数，如图所示：



- 请求参数 uuid 是函数参数 find_uuid
- 请求参数 originUrl 是商家详细页的 URL 地址
- 请求参数 id 是商家 id

因此，函数 get_comment() 的代码如图所示：

```
# 获取顾客评论，参数uuid作为请求参数，参数id是商家的id，是find_poiId里面有一个元素
def get_comment(uuid, id):
    # 获取评论的总页数
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 '
                      '(KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36',
    }
    url = 'http://www.meituan.com/meishi/api/poi/getMerchantComment?uuid=%s&platform=1&partner=126&' \
          'originUrl=http://www.meituan.com/meishi/%s/&riskLevel=1&optimusCode=1&id=%s&userId=&' \
          'offset=%s&pageSize=10&sortType=1'
    r = requests.get(url %(uuid,id,id,'0'), headers=headers)
    total = r.json()['data']['total']
    # 循环评论的每一页，获取每条数据
    for i in range(math.ceil(int(total)/10)):
        offset = str(10*i)
        r = requests.get(url % (uuid, id, id, offset), headers=headers)
        # 获取每条数据并入库处理
        for c in r.json()['data']['comments']:
            comment_dict = {}
            comment_dict['shop_id'] = id
            comment_dict['userId'] = c.get('userId', '')
            comment_dict['reviewId'] = c.get('reviewId', '')
            comment_dict['userName'] = c.get('userName', '')
            comment_dict['userScore'] = str(int(c.get('star', ''))/10)
            comment_dict['comment'] = c.get('comment', '')
            comment_dict['commentTime'] = time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(int(c.get('commentTime', '0'))/1000))
            comment_dict['merchantComment'] = c.get('merchantComment', '')
            comment_db(comment_dict)
```

四、ORM 框架实现数据持久化存储

爬虫的核心功能大致已实现，接着讲解数据存储方面。数据存储以 MySQL 为例，存储过程使用 ORM 框架 SQLAlchemy，这样可实现数据持久化，它的优点此处不一一讲述。

首先在 Insql.py 里导入 ORM 框架 SQLAlchemy，并创建相关对象，通过 SQLAlchemy 连接本地 MySQL 数据库。数据库编码设为 UTF8mb4，因为评论信息里可能出现手机表情这类特殊内容，这些特殊内容是超出 UTF-8 的编码范围。代码如下：

```
import time
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# 使用utf8mb4编码连接MySql
engine = create_engine('mysql+pymysql://root:1234@localhost/spiderdb?charset=utf8mb4')
DBSession = sessionmaker(bind=engine)
SQLsession = DBSession()
Base = declarative_base()
```

将商家信息和顾客评论信息分别存储在数据表 meituan_shop 和 meituan_comment。数据表之间存在一对多的数据关系，一个商家会有多条顾客评论，映射类的定义如下：

```python
# 商家数据表
class shop(Base):
    __tablename__ = 'meituan_shop'
    id = Column(Integer(), primary_key=True)
    shop_id = Column(String(100), comment='商家ID')
    shop_name = Column(String(300), comment='商家名称')
    shop_address = Column(String(500), comment='商家地址')
    shop_phone = Column(String(100), comment='商家电话')
    shop_openTime = Column(String(500), comment='营业时间')
    shop_avgScore = Column(String(100), comment='评分')
    shop_avgPrice = Column(String(100), comment='人均价格')
    shop_city = Column(String(100), comment='所在城市')
    log_date = Column(String(100), comment='记录日期')
# 顾客评论
class comment(Base):
    __tablename__ = 'meituan_comment'
    id = Column(Integer(), primary_key=True)
    shop_id = Column(String(100), comment='商家ID')
    reviewId = Column(String(100), comment='评论ID')
    userId = Column(String(100), comment='用户ID')
    userName = Column(String(100), comment='用户名')
    userScore = Column(String(100), comment='用户评分')
    comment = Column(String(3000), comment='评论内容')
    commentTime = Column(String(100), comment='评论时间')
    merchantComment = Column(String(3000), comment='商家回复')
    log_date = Column(String(100), comment='记录日期')
```

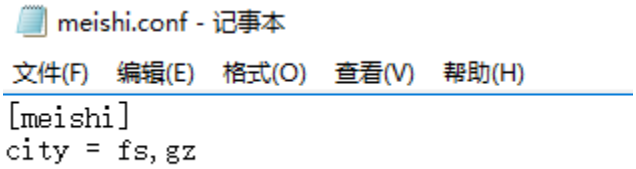

上述只是定义映射类，数据存储的功能尚未实现。数据存储由函数 shop_db() 和函数 comment_db() 实现，两者会对待存储的数据进行判断，如果数据已存在数据库，则进行更新处理，反之新增一条数据。代码如下：

```python
# 写入商家信息
def shop_db(info_dict):
    temp_id = info_dict['shop_id']
    # 判断是否已存在记录
    info = SQLsession.query(shop).filter_by(shop_id=temp_id).first()
    if info:
        info.shop_id = info_dict.get('shop_id', '')
        info.shop_name = info_dict.get('shop_name', '')
        info.shop_address = info_dict.get('shop_address', '')
        info.shop_phone = info_dict.get('shop_phone', '')
        info.shop_openTime = info_dict.get('shop_openTime', '')
        info.shop_avgScore = info_dict.get('shop_avgScore', '')
        info.shop_avgPrice = info_dict.get('shop_avgPrice', '')
        info.shop_city = info_dict.get('shop_city', '')
        info.log_date = time.strftime('%Y-%m-%d', time.localtime(time.time()))
    else:
        inset_data = shop(
            shop_id=info_dict.get('shop_id', ''),
            shop_name=info_dict.get('shop_name', ''),
            shop_address=info_dict.get('shop_address', ''),
            shop_phone=info_dict.get('shop_phone', ''),
            shop_openTime=info_dict.get('shop_openTime', ''),
            shop_avgScore=info_dict.get('shop_avgScore', ''),
            shop_avgPrice=info_dict.get('shop_avgPrice', ''),
            shop_city=info_dict.get('shop_city', ''),
            log_date=time.strftime('%Y-%m-%d', time.localtime(time.time()))
        )
        SQLsession.add(inset_data)
    SQLsession.commit()
```

```python
# 写入顾客评论信息
def comment_db(info_dict):
    temp_id = info_dict['reviewId']
    # 判断是否已存在记录
    info = SQLsession.query(comment).filter_by(reviewId=temp_id).first()
    if info:
        info.shop_id = info_dict.get('shop_id', '')
        info.userId = info_dict.get('userId', '')
        info.userName = info_dict.get('userName', '')
        info.userScore = info_dict.get('userScore', '')
        info.comment = info_dict.get('comment', '')
        info.commentTime = info_dict.get('commentTime', '')
        info.merchantComment = info_dict.get('merchantComment', '')
        info.log_date = time.strftime('%Y-%m-%d', time.localtime(time.time()))
    else:
        inset_data = comment(
            shop_id=info_dict.get('shop_id', ''),
            reviewId=info_dict.get('reviewId', ''),
            userId=info_dict.get('userId', ''),
            userName=info_dict.get('userName', ''),
            userScore=info_dict.get('userScore', ''),
            comment=info_dict.get('comment', ''),
            commentTime=info_dict.get('commentTime', ''),
            merchantComment=info_dict.get('merchantComment', ''),
            log_date=time.strftime('%Y-%m-%d', time.localtime(time.time()))
        )
        SQLsession.add(inset_data)
    SQLsession.commit()
```


五、设置配置文件，动态控制爬取方向

配置文件给爬虫程序 meishi.py 读取，用于控制爬虫的爬取方向，比如爬取北京、上海等城市的美食信息。将配置文件命名为 meishi.conf，配置信息如下：



配置文件只设置配置属性 city，属性值是将每个城市编号以英文逗号的形式拼接起来，城市编号是首个字母拼音开头组成的。函数 get_all() 的函数参数 city_list 就是配置属性 city。

在爬虫文件 meishi.py 里，文件运行函数 __main__ 主要读取配置文件的配置属性 city，并调用函数 get_all()，代码如下：

```
if __name__ == '__main__':
    # 读取配置文件
    cf = configparser.ConfigParser()
    cf.read("meishi.conf")
    city_list = str(cf.get('meishi', 'city'))
    get_all(city_list)
```

六、基于请求头的反爬虫机制：根据商家信息动态设置请求头

从函数 get_all() 的请求头可以看到，请求头属性 Host 和 Referer 都带有变量 city，这是一种最为常见的反爬虫机制。

属性 Host 是设置域名信息，特别 URL 地址以地域划分，这种情况下，属性 Host 与 URL 地址是相互对应的，如 URL 为 https://gz.meituan.com/meishi/，属性 Host 应 gz.meituan.com。假如请求头的 Host 对不上 URL 的域名，很可能无法正确地响应内容。

属性 Referer 表示当前的网页的上一级链接地址，这个没用固定的属性值，一部分网站会对 Referer 进行检测（一些资源网站的防盗链就是检测 Referer）。一般情况下，只需加入 Referer 属性即可避免这种反爬虫检测，但个人建议 Referer 属性值尽量动态变化，若属性值固定不变，请求次数过多时，还是很容易给网址的反爬虫机制检测出来。

七、基于 Cookies 的发爬虫机制：利用浏览器构建 Cookies 池

在函数 get_info() 可以看到，函数是爬取商家信息，对商家详细页发送 HTTP 请求，除了设置请求头之外，还需要设置 Cookies，否则无法获取目标数据。Cookies 的获取方法有两种：

- 从浏览器开发者工具中获取，并写入代码里。
- 通过 Selenium 等自动化工具模拟用户打开浏览器，再从浏览器获取并传递到代码。

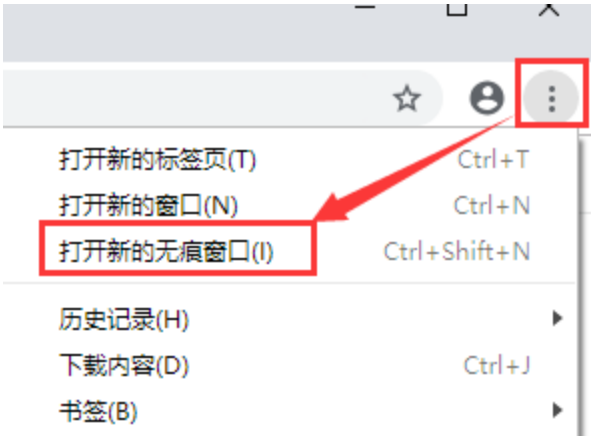
在商家详细页里，虽然加入 Cookies 访问能正常获取数据，但项目上线或长时间运行的时候，就会发现程序爬取几十条商家信息后就无法再爬取，这是长期使用一个 Cookies 并且访问过频而触发的反爬虫机制。

这种带 Cookies 并且又不能访问过频的请求只能构建 Cookies 池，每次访问从 Cookies 池抽取其中一条 Cookies 进行访问，不管发送请求的频率多高，只要不是长期使用同一个

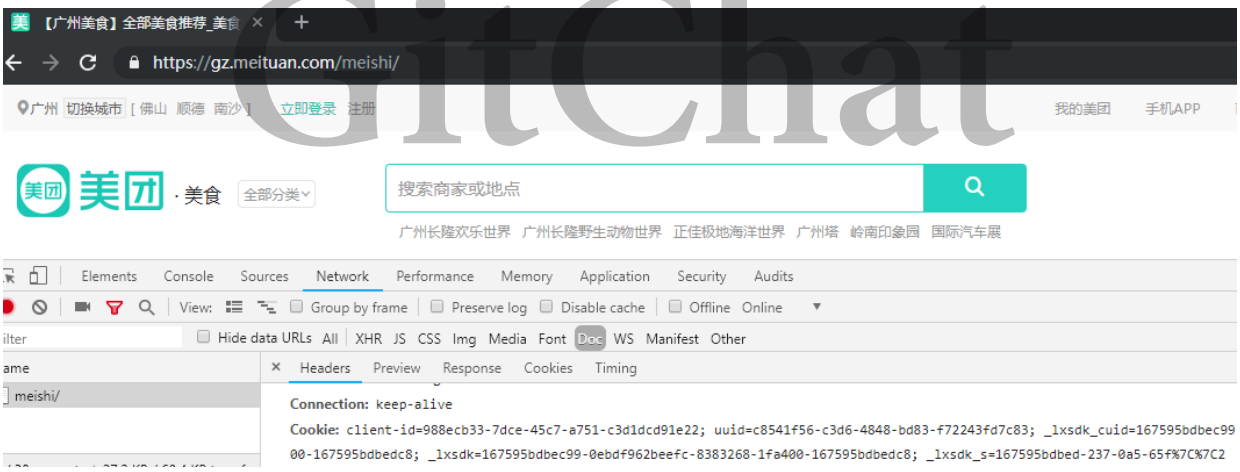
Cookies 就能避开这种反爬虫检测。

那么问题来了，现在我们只有一个浏览器、一台电脑，也就意味着只有一个外网 IP 地址，如何构建 Cookies 池？一般情况下，一台电脑在一个网站只会有一个 Cookies 信息，若要使用同一台电脑生成多个不同的 Cookies，我们需要借助谷歌的无痕模式。

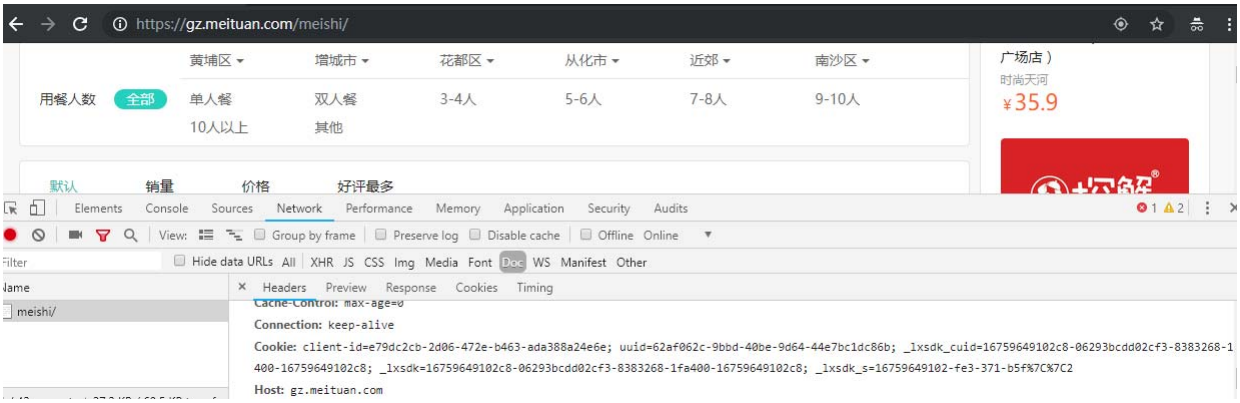
首先打开新的谷歌浏览器，点击右上方“自定义及控制”按钮，选中点击“打开新的无痕窗口”，如图所示：



在无痕模式下输入美团美食网站 https://gz.meituan.com/meishi/，并在开发者工具获取 Cookies 信息，如图所示：



在第二次获取新的 Cookies 之前，需要将谷歌浏览器全部关闭，然后重新打开新的无痕窗口，访问美团美食网站获取新的 Cookies，如图所示：



通过两图对比，可以发现无痕模式下的 Cookies 是各不相同，因此使用这种方式可以构建 Cookies 池。

八、分布式爬虫的扩展说明

分布式策略考虑的因素有网站服务器负载量、网速快慢、硬件配置和数据库最大连接量。

> 举个例子，爬取某个网站 1000 万数据，从数据量分析，当然进程和线程越多，爬取的速度越快。但往往忽略了网站服务器的并发量。假设设定 10 个进程，每个进程 200 条线程，每秒并发量为 200×10=2000。若网站服务器并发量远远低于该并发量，在请求网站的时候，就会出现卡死的情况，导致请求超时（即使对超时做了相应处理），无形之中增加等待时间。除此之外，进程和线程越多，对运行程序的系统的压力越大，若涉及数据入库，还要考虑并发数是否超出数据库连接数。

从本项目的函数可以看到，函数 get_all() 里实现两个循环，最外层循环是遍历城市列表，爬取多个城市的美食商家；最内层循环是爬取美食商家列表，共 32 页。

我们可以将外层循环设置多进程执行，每个城市单独使用一个进程；内层循环是固定循环 32 次，每次循环由一个线程执行。

总得来说，每个城市的美食信息以一个进程表示，在每个进程里创建 32 条线程，分别爬取当前城市每一页的美食商家信息和顾客评论。

九、总结

整个爬虫项目由三个文件组成，文件所实现的功能说明如下：

- Insql.py 是将数据入库处理，由 ORM 框架 SQLAlchemy 实现
- meishi.conf 设置区域信息，如广州（gz）或北京（bj）等，从而控制爬虫的爬取方向
- meishi.py 实现爬虫功能

爬虫文件一共定义了三个功能函数 get_all()、get_info() 和 get_comment()，函数所实现的功能说明如下：

- get_all() 是遍历各个城市的所有美食列表，从美食列表里提取商家 id 和相关数据，调用函数 get_info()。
- get_info() 是根据商家 id 构建商家详细页，并在商家详细页爬取商家信息，最后调用函数 get_comment() 和入库函数 shop_db()。
- get_comment() 是根据商家信息去访问 AJAX 接口，从中爬取顾客评论，并调用入库函数 get_comment()。

最后，由于本人水平有限，如有问题以及建议，请留下您宝贵的意见，谢谢！！

项目源码

> GitHub下载地址