# requests 高级请求库

1. 基于urllib编写的HTTP请求库
2. 更加具有Python的风格，操作更加人性化
3. 是爬虫中最常用的方法
    1. requests是目前为止企业中应用最广泛的爬虫相关模块
    2. scrapy是一款爬虫框架，功能更加强大，但是目前为止，企业应用不是很广泛
4. 是一个第三方模块
    pip install requests  下载安装
5. 用于发送请求和处理响应
6. 安装：
    1. pip  直接安装
    2. pip  源文件压缩包（whl）

## requests 模块的属性

1. get(url,params=None,**wkargs)
    构建一个request对象，通过url发送get请求，返回一个response对象
    url:http请求路径
    params：get请求需要携带的参数（可以是一个字典，或者bytes类型数据）
    **kwargs：
        1. 关键字参数传值
        2. 可变长参数
        3. 有指定参数（查看底层源码）

2. post(url,data=None,json=None,**kwargs):
    构建一个request对象，通过url发送posts请求，返回一个response对象
    url:http请求路径
    data：post请求需要携带的数据（可以是一个字典，或者bytes类型数据）
    json：需要一个json对象，用于发送json数据
    **kwargs：
        1. 关键字参数传值
        2. 可变长参数
        3. 有指定参数（查看底层源码）

**# 说明：**
post方法和get方法本质都是调用request方法

3. request（method，url，**，kwargs）：
    metohd:请求的方法：是一个字符串‘post’  ‘get’
    url:http请求路径
    **kwargs：
    param：get请求携带的数据
    data:post请求携带的数据
    json：post请求需要携带的json对象
    headers：是一个字典，HTTPheaders
    cookies：是一个CookieJar对象，需要是一个字典
    files:字典，用于传递数据，只能2-4个元素（利用元组）

auth：是一个元组，启用认证
timeout:设置超时时间，以秒为单位
allow_redirects：是否重定向（布尔类型
proxies：是一个字典，设置代理（多个）
verify：是布尔类型，设置TLS服务器证书
cert:设置SSL客户端证书

4. head():
获得请求头
5. put（）：
提交put请求
6. patch（）：
提交局部修改数据
7. delete（）：
提交删除数据

补充：
1. 爬虫：是正规的技术
广而告之我是爬虫
2. 企业爬虫：
以数据为优先
利用伪装技术

```
import requests

# url='http://httpbin.org/get'
# url='http://httpbin.org/post'

# res=requests.get(url,params={'name':'hehe'})
# print(res.text)

# res3=requests.get(url+'?name=hehe&pwd=123')
# print(res3.text)

# res2=requests.post(url,data={'name':'hehe','pwd':'123'})
# print(res2.text)

# res4=requests.request('get',url+'?name=hehe&pwd=123')
# print(res4.text)
# requests.get()


url='https://www.dianping.com/'
headers={
    'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/71.0.3578.98 Safari/537.36'
}
print(requests.get(url,headers=headers).text)
```

- response对象的属性

1. 响应回来的数据会打包成response对象
2. 常用的属性：
    1. encoding：设置响应编码
    2. text: 返回响应内容的str格式的内容（默认使用Unicode编码）
    3. content：返回响应内容的bytes格式的内容
    4. status_code:状态码
    5.apperent_encoding:分析响应的编码，并返回
    6. cookies：返回响应回来的cookies
    7. elapsed：返回耗时
    8. history：响应跳转的历史记录
    9. json：返回对应的json数据
    10. url：返回当前请求的地址
    11. headers：返回请求头的内容

```
import requests
# url='http://www.baidu.com'
# res=requests.get(url)
# # 给响应对象设置编码
# res.encoding='utf-8'
# print(res.text)
# print(requests.get(url).content)
# print(requests.get(url).status_code)
# print(requests.get(url).apparent_encoding)
# res=requests.get(url)
# res.encoding=res.apparent_encoding
# # print(res.cookies)
# # print(res.elapsed)
# print(res.json())

url='http://www.honestcareer.com/hr/dologin'
data={
'type': '1',
'username': '18501279410',
'password': 'hbw123'
}
res=requests.post(url=url,data=data)
print(res.text,type(res.text))
print(res.json(),type(res.json()))
print(res.url)
print(res.headers)
```

---

- 编码的设置

```
# 1. 手动设置编码： 在网页中查找charset
# 2. 通过apparent_encoding  自动查找编码（不可靠）
# 3. chardet模块，用于自动分析编码（reqeusts模块底层应用chardet实现的自动查找）

import requests
res=requests.get('http://www.baidu.com')
# 1. 手动
res.encoding='utf-8'
```

```
# 2. 内置自动
res.encoding=res.apparent_encoding
# 3. 引入chardet
import chardet
result=chardet.detect(res.content)
res.encoding=result['encoding']
# print(result)
print(res.text)
```

- Headers设置

```
url='https://www.dianping.com/'
headers={
    'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/71.0.3578.98 Safari/537.36'
}
print(requests.get(url,headers=headers).text)
```

- cookies设置

```
Cookie存储于客户端
1. cookie是由服务器发送给客户端的特殊信息，而这些信息以文本文件的方式存储在客户端,然后客户端每一次访问
服务器时，都会携带这些特殊信息
2. 访问网站时，用户提交登录信息，然后服务器响应一个网页（JSON），以及cookie信息（响应头）
3. 客户端再次向服务器发送请求时，把cookie信息（包含认证信息）再次发送回服务器
4. 在session出现之前，cookie用于回话跟踪
```

- cookie的分类

```
1. 会话cookie
        1. 如果不设置过期时间，则cookie生命周期为浏览器的会话周期，关闭浏览器，cookie随之死亡
        2. 会话cookie一般存在于内存中
2. 持久cookie
        1. 如果设置了过期时间，则cookie的生命周期为设置的时间所控，关闭浏览器，cookie不消失
        2. 持久化cookie存在于硬盘（以文件的形式存储）
```

```
import requests
# 用于登录
url='http://www.honestcareer.com/hr/dologin'
# 用于访问主页
url2='http://www.honestcareer.com/hr/index'

data={
'type': '1',
'username': '18501279410',
'password': 'hbw123'
}
# 1. 自动从响应中拿到cookie信息
res1=requests.post(url,data=data)
cookies=res1.cookies
#userToken=bdc58323-a842-40de-b0f4-16cabedad7ba
```

```
# 2. 手动传入cookie信息
cookies={
'userToken':'77b2efa6-e6a3-4f0d-825f-4bbf3be6e9b5',
# 'CNZZDATA1260937792':'1549063092-1547708119-null%7C1548042458'
}
res2=requests.get(url2,cookies=cookies)
print(res2.text)
```

---

- 代理设置

```
import requests as r

#
url='http://httpbin.org/ip'
proxies={
    'http':'119.101.118.142:9999'
}
print(r.get(url,proxies=proxies).text)

# # 测试  httpbin.org/ip
# testURL='http://httpbin.org/ip'
# res1=r.get(testURL).text
# print(res1)
# res2=r.get(testURL,proxies=proxies).text
# print(res2)
```

- 超时设置

```
import requests as r

# 在爬取数据时，常常会遇到，url的响应时间过长
# 如果时间过长：1.该链接是坏连接，2. 网速或服务器故障，  这一类连接往往是无效链接（可以直接忽略不要---丢弃）
# 如果超时，会报错，应该try-except  爬虫项目中---一定会有try-except
# try-except使用的时机： 设置超时，发送任何请求，多模板使用时
# try-except的目的：防止任何不必要的异常导致整个程序的终止

while 1:
    url = 'http://www.baidu.com'
    try:
        res = r.get(url, timeout=0.1)  # 超时单位： 秒
        res.encoding='utf-8'
        print(res.text)
    except Exception as e:
        print(e)
```

- 小结：

```
1. requests实现了HTTP请求  中的get函数，post函数
        delete，pu，head
2. 参数：request中常用的请求底层都是调用的requset（）函数
        参数遵照reqeust（）函数的文档说明
3. requests是第三方模块
4. text和content：
        text:str
        content:bytes
```

---

## 补充：Session会话

```
1. session的概念：
        1. session又称之为会话状态，用于恢复当前浏览器相关的信息
        2. http是无状态的，一次请求得到一次响应，该状态结束
        3. session是会话保持状态，一次请求和响应之后，通信不结束
2. session的运行机制：
        1. 服务器session对于每一个客户端：人手一份，用户首次与服务器建立连接时，服务器会创建一个session空间，存储相关信息，并且创建SessionID作为唯一标识别  （区分多个客户端）
        2. sessionID：是一个由24个字符构成的一个ID信息（不内存地址）
        3. 用户每次提交页面，或发生请求，客户端会携带sessionID发送请求到服务器（cookie携带sessionID）
3. sessionID的存储位置：
        1. cookie中：不能使用绝对连接（如果禁止了cookie，sessionID无法传递）
        2. url中：直接拼接到url之后
4. session内容的存储位置：
        1. InProc：web服务器的内存
        2. DBserver：数据库
        3. StateServer：服务进程中
5. session的分类：
        1. 服务器session：
                用于存储session的具体信息
        2. 客户端session：
                用于存储sessionID
6. session不需要任何的配置，可以直接使用
7. session的生命周期：
        根据设置的时间而定
        本质：不是保持了会话状态，而是调用存储在服务器中的session内容，间接的模拟了会话
8. 会话保持的原理
9. session和cookies的关系：
        在浏览器中利用cookie传递sessionID（cookie不存登录信息）
        在服务器中利用session存储了登录信息
10. session的优缺点：
        1. 优点：使用方便，可以传递大量信息
        2. 缺点：占用的服务器资源过大，给服务器压力过大
        总结：能不用，就不用
```

---

- 利用session保持cookie信息

```
import requests as  r
```

```
url='http://httpbin.org/cookies'
cookies={
    'hehe':'123'
}

# 往往爬取任何数据，都要发送请求，发送请求常常会携带一个cookie（登录认证）
# 访问多个请求时，要携带多次cookies，并且多次的cookies值完全一样---代码冗余
# session的cookies绑定，可以避免代码冗余，绑定一次，发送的所有请求不用再次书写

session=r.session()
res1=session.get(url,cookies=cookies)
print(res1.text)
# 将cookies绑定到session中
session.cookies.update(cookies)
res2=session.get(url)
print(res2.text)

# 在绑定了session的cookies的前提下，可以再次手动传入cookie
# 如果手工输入的cookies信息和绑定的cookies信息，键冲突（底层是一个字典），以手动输入为优先（遮蔽）
cookies3={'hehe':'hehehehehe'}
res3=session.get(url,cookies=cookies3)
print(res3.text)

res4=session.get(url)
print(res4.text)
```

- session保持headers信息

```
import requests as r

url='http://httpbin.org/headers'
headers={
    'hehe':'123'
}
s=r.session()
# 绑定headers到session
# 1. 使用update绑定headers
# s.headers.update(headers)
# 2. 直接赋值
s.headers=headers
# 发送请求
res1=s.get(url)
print(res1.text)
```

- with--- session

```
# with open() as f:
    # pass
import requests as r
url='http://httpbin.org/headers'
```

```
s=r.session()
headers={
    'hehe':'123'
}
s.headers=headers
res1=s.get(url)
print(res1.text)

with r.session() as s:
    s.headers=headers
    res2=s.get(url)
    print(res2.text)
# with session的优势
1. 可以将全局变量规划为局部变量更加安全
2. 将相同业务的代码，组织在一起，可读性更强
```

- 小结

```
1. request（post，get）占用的资源较少，安全性较高，但是缺乏持续性
2. session，更加方便（保持了会话），但是数据不安全，资源消耗大
3. 尽量使用request 尽量少使用session，如果需要使用登录状态保持，尽量直接使用cookies
```

---

## 实战---智联招聘

```
1. 跳过登录
2. 关键字：
        1.职位：[python  AI  大数据  爬虫] 记录下标
        2.地区：[北京 成都 西安 南京]  记录下标
3. 进入到列表页
        获取详情页连接
        列表页的分页（动态url）
4. 进入到详情页：
        获取指定的字段：
        职位，薪资，公司名称，招聘基本要求，职位信息，公司简明信息，公司概况
5. 采集数据要求：
        页面分析
        1. 多模板
        2. 采集量80%以上
        3. 可能会有js渲染（看到的结构，和拿到的结构不一定一样）
        4. 基本的反爬虫手段（user-agent，refer）
        5. 如果IP被封禁（做代理---西刺）
6. 异常处理
        一定要写异常（超时设置，请求一般都是有异常处理（except：pass））
7. 数据存储：
        mysql存储
8. 断点设置
        断点要点：页码，数据（职位，地区）
        思路：
            1. 从下标0开始执行关键字查询
            2. 时刻存储下标的数值，存到文件中
            3. 再次查询时，只需从文件中取出，存储的下标和页数
```

4. 继续爬取

问题：数据不一定全部都不重复（数据可以部分重复---去重）