requests官方中文指导文档：https://docs.python-requests.org/zh_CN/latest/index.html

requests库的七个主要方法

requests.request()　构造一个请求，支持以下各种方法

requests.get()　　获取html的主要方法

requests.head()　　获取html头部信息的主要方法
requests.post()　　向html网页提交post请求的方法
requests.put()　　向html网页提交put请求的方法
requests.patch()　　向html提交局部修改的请求
requests.delete()　　向html提交删除请求

requests.get()

这个方法是我们平时最常用的方法之一，通过这个方法我们可以了解到其他的方法，所以我们详细介绍这个方法。

具体参数是：

```
r=requests.get(url,params,**kwargs)
```

url: 需要爬取的网站地址。

params: 翻译过来就是参数， url中的额外参数，字典或者字节流格式，可选。

**kwargs : 12个控制访问的参数

**kwargs

**kwargs有以下的参数，对于requests.get,其第一个参数被提出来了。

- params：字典或字节序列， 作为参数增加到url中,使用这个参数可以把一些键值对以?key1=value1&key2=value2的模式(GET方法）增加到url中

```
例如：
kv = {'key1':' values', 'key2': 'values'}
r = requests.get('http:www.python123.io/ws', params=kv)
```

- data：字典，字节序列或文件对象，重点作为向服务器提供或提交数据(POST方法),作为request的内容，与params不同的是，data提交的数据并不放在url链接里，而是放在url链接对应位置的地方作为数据来存储。它也可以接受一个字符串对象。
- json：json格式的数据， json合适在相关的html，http相关的web开发中非常常见， 也是http最经常使用的数据格式， 他是作为内容部分可以向服务器提交。

```
 例如：
kv = {'key1': 'value1'}
r = requests.post('http://python123.io/ws', json=kv)
```

- headers：字典是http的相关语，对应了向某个url访问时所发起的http的请求头字段， 可以用这个字段来定义http的访问的http头，可以用来模拟任何我们想模拟的浏览器来对url发起访问。

```
例子：
 hd = {'user-agent': 'Chrome/10'}
```

```
r = requests.post('http://python123.io/ws', headers=hd)
```

- cookies：字典或CookieJar，指的是从http中解析cookie
- auth：元组，用来支持http认证功能
- files：字典， 是用来向服务器传输文件时使用的字段。

```
例子：
fs = {'files': open('data.txt', 'rb')}
r = requests.post('http://python123.io/ws', files=fs)
```

- timeout: 用于设定超时时间， 单位为秒，当发起一个get请求时可以设置一个timeout时间， 如果在timeout时间内请求内容没有返回， 将产生一个timeout的异常。
- proxies：字典， 用来设置访问代理服务器。
- allow_redirects: 开关， 表示是否允许对url进行重定向， 默认为True。
- stream: 开关， 指是否对获取内容进行立即下载， 默认为True。
- verify：开关， 用于认证SSL证书， 默认为True。
- cert： 用于设置保存本地SSL证书路径

文件上传

```
import requests
url = "http://httpbin.org/post"
files= {"files":open("test.jpg","rb")}
response = requests.post(url,files=files)
print(response.text)
```

获取cookie

```
import requests
response = requests.get('https://www.baidu.com')
print(response.cookies)
for key,value in response.cookies.items():
 print(key,'=',value)
```

会话保持

cookie的一个作用就是可以用于模拟登陆，做会话维持

```
import requests
session = requests.Session()
session.get('http://httpbin.org/cookies/set/number/12456')
response = session.get('http://httpbin.org/cookies')
```

```
print(response.text)
```

cookie和session区别

cookie数据存放在客户的浏览器上，session数据放在服务器上

cookie不是很安全，别人可以分析存放在本地的cookie并进行cookie欺骗

session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie

返回对象属性

其中response对象有以下属性：

r.status_code	http请求的返回状态，若为200则表示请求成功。
r.text	http响应内容的字符串形式，即返回的页面内容
r.encoding	设置返回对象内容的编码格式
r.apparent_encoding	从内容中分析出的响应内容编码方式
r.content	http响应内容的二进制形式
response.headers	response.headers['content-type'] 获得响应头内容
response.request.headers	获取请求头内容
response.cookies	获取cookie

response.content

response.text返回的是Unicode格式，通常需要转换为utf-8格式，否则就是乱码。

response.content是二进制模式，可以下载视频之类的，如果想看的话需要decode成utf-8格式。

```
import requests

#allow_redirects=False#设置这个属性为False则是不允许重定向，反之可以重定向
response = requests.get("http://www.baidu.com",allow_redirects=False)
#打印请求页面的状态（状态码）
print(type(response.status_code),response.status_code)
#打印请求网址的headers所有信息
print(type(response.headers),response.headers)
#打印请求网址的cookies信息
 print(type(response.cookies),response.cookies)
 #打印请求网址的地址
 print(type(response.url),response.url)
 #打印请求的历史记录（以列表的形式显示）
 print(type(response.history),response.history)
```

获取Json数据

从下面的数据中我们可以得出，如果结果：

1、requests中response.json()方法等同于json.loads（response.text）方法，只有返回数据是json格式才能使用此方法

```
import requests
import json

response = requests.get("http://httpbin.org/get")
print(type(response.text))
print(response.json())
print(json.loads(response.text))
print(type(response.json())
print(response.json()['data'])
```

获取二进制数据

在上面提到了response.content，这样获取的数据是二进制数据，同样的这个方法也可以用于下载图片以及视频资源

内置的状态码

```
100: ('continue',),
101: ('switching_protocols',),
102: ('processing',),
103: ('checkpoint',),
122: ('uri_too_long', 'request_uri_too_long'),
200: ('ok', 'okay', 'all_ok', 'all_okay', 'all_good', '\\o/', '✓'),
201: ('created',),
202: ('accepted',),
203: ('non_authoritative_info', 'non_authoritative_information'),
 204: ('no_content',),
 205: ('reset_content', 'reset'),
 206: ('partial_content', 'partial'),
 207: ('multi_status', 'multiple_status', 'multi_stati',
'multiple_stati'),
 208: ('already_reported',),
 226: ('im_used',),

 # Redirection.
```

```
 300: ('multiple_choices',),
 301: ('moved_permanently', 'moved', '\\o-'),
 302: ('found',),
 303: ('see_other', 'other'),
 304: ('not_modified',),
 305: ('use_proxy',),
 306: ('switch_proxy',),
 307: ('temporary_redirect', 'temporary_moved', 'temporary'),
 308: ('permanent_redirect',
  'resume_incomplete', 'resume',), # These 2 to be removed in 3.0

 # Client Error.
 400: ('bad_request', 'bad'),
 401: ('unauthorized',),
 402: ('payment_required', 'payment'),
 403: ('forbidden',),
 404: ('not_found', '-o-'),
 405: ('method_not_allowed', 'not_allowed'),
 406: ('not_acceptable',),
 407: ('proxy_authentication_required', 'proxy_auth', 'proxy_authenticati
on'),
 408: ('request_timeout', 'timeout'),
 409: ('conflict',),
 410: ('gone',),
 411: ('length_required',),
 412: ('precondition_failed', 'precondition'),
 413: ('request_entity_too_large',),
 414: ('request_uri_too_large',),
 415: ('unsupported_media_type', 'unsupported_media', 'media_type'),
 416: ('requested_range_not_satisfiable', 'requested_range', 'range_not_s
atisfiable'),
 417: ('expectation_failed',),
 418: ('im_a_teapot', 'teapot', 'i_am_a_teapot'),
 421: ('misdirected_request',),
 422: ('unprocessable_entity', 'unprocessable'),
 423: ('locked',),
 424: ('failed_dependency', 'dependency'),
 425: ('unordered_collection', 'unordered'),
 426: ('upgrade_required', 'upgrade'),
 428: ('precondition_required', 'precondition'),
 429: ('too_many_requests', 'too_many'),
```

```
431: ('header_fields_too_large', 'fields_too_large'),
444: ('no_response', 'none'),
449: ('retry_with', 'retry'),
450: ('blocked_by_windows_parental_controls', 'parental_controls'),
451: ('unavailable_for_legal_reasons', 'legal_reasons'),
499: ('client_closed_request',),

# Server Error.
500: ('internal_server_error', 'server_error', '/o\\', '✗'),
501: ('not_implemented',),
502: ('bad_gateway',),
503: ('service_unavailable', 'unavailable'),
504: ('gateway_timeout',),
505: ('http_version_not_supported', 'http_version'),
506: ('variant_also_negotiates',),
507: ('insufficient_storage',),
509: ('bandwidth_limit_exceeded', 'bandwidth'),
510: ('not_extended',),
511: ('network_authentication_required', 'network_auth', 'network_authentication'),
```

requests.head()--只获取响应头

获取html头部信息的主要方法

```
>>> r=requests.head("http://httpbin.org/get")
>>>r.headers
 {'Connection': 'keep-alive', 'Server': 'meinheld/0.6.1', 'Date': 'Mon, 20 Nov 2017 08:08:46 GMT',
  'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Credentials': 'true', 'X-Powered-By': 'Flask',
  'X-Processed-Time': '0.000658988952637', 'Content-Length': '268', 'Via': '1.1 vegur'}
```

requests.post()

向url post一个字典：

```
>>> payload={"key1":"value1","key2":"value2"}
>>> r=requests.post("http://httpbin.org/post",data=payload)
>>> print(r.text)
{
```

```
  "args": {},
  "data": "",
  "files": {},
  "form": {
  "key1": "value1",
   "key2": "value2"
   },
   "headers": {
   "Accept": "*/*",
   "Accept-Encoding": "gzip, deflate",
   "Connection": "close",
   "Content-Length": "23",
   "Content-Type": "application/x-www-form-urlencoded",
   "Host": "httpbin.org",
   "User-Agent": "python-requests/2.18.4"
   },
   "json": null,
   "origin": "218.197.153.150",
   "url": "http://httpbin.org/post"
}
```

向url post 一个字符串，自动编码为data：

```
>>>r=requests.post("http://httpbin.org/post",data='helloworld')
>>>print(r.text)
{
  "args": {},
  "data": "helloworld",
  "files": {},
  "form": {},
  "headers": {
  "Accept": "*/*",
   "Accept-Encoding": "gzip, deflate",
   "Connection": "close",
   "Content-Length": "10",
   "Host": "httpbin.org",
   "User-Agent": "python-requests/2.18.4"
   },
   "json": null,
   "origin": "218.197.153.150",
   "url": "http://httpbin.org/post"
```

```
}
```

向url post一个文件：

```
>>> import requests
>>> files = {'files':open('F:\\python\\test\\test_case\\files.txt','rb')}
>>> r = requests.post('https://httpbin.org/post',files=files)
>>> print(r.text)
{
 "args":{

 },
 "data":"",
  "files":{
  "files":"hello worle!"
  },
  "form":{

  },
  "headers":{
  "Accept":"*/*",
  "Accept-Encoding":"gzip, deflate",
  "Connection":"close",
  "Content-Length":"158",
  "Content-Type":"multipart/form-data; boundary=d2fb307f28aeb57b932d867f80f2f600",
  "Host":"httpbin.org",
  "User-Agent":"python-requests/2.19.1"
  },
  "json":null,
  "origin":"113.65.2.187",
  "url":"https://httpbin.org/post"
}
```

requests.put()

看代码：

```
>>> payload={"key1":"value1","key2":"value2"}
>>> r=requests.put("http://httpbin.org/put",data=payload)
>>> print(r.text)
```

```
{
 "args": {},
 "data": "",
 "files": {},
 "form": {
 "key1": "value1",
  "key2": "value2"
  },
  "headers": {
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "close",
  "Content-Length": "23",
  "Content-Type": "application/x-www-form-urlencoded",
  "Host": "httpbin.org",
  "User-Agent": "python-requests/2.18.4"
  },
  "json": null,
  "origin": "218.197.153.150",
  "url": "http://httpbin.org/put"
}
```


requests.patch()

requests.patch和request.put类似。

两者不同的是：

当我们用patch时仅需要提交需要修改的字段。

而用put时，必须将20个字段一起提交到url，未提交字段将会被删除。

patch的好处是：节省网络带宽。

requests.request()

requests.request()支持其他所有的方法。

requests.request(method，url,**kwargs)

method: “GET”、” HEAD”、” POST”、” PUT”、” PATCH”等等

url: 请求的网址

**kwargs: 控制访问的参数

requests库的异常

注意requests库有时会产生异常，比如网络连接错误、http错误异常、重定向异常、请求url超时异常等等。所以我们需要判断r.status_codes是否是200，在这里我们怎么样去捕捉异常呢?
这里我们可以利用r.raise_for_status() 语句去捕捉异常，该语句在方法内部判断r.status_code是否等于200，如果不等于，则抛出异常。
于是在这里我们有一个爬取网页的通用代码框架:

```
try:
 r=requests.get(url,timeout=30)#请求超时时间为30秒
 r.raise_for_status()#如果状态不是200，则引发异常
 r.encoding=r.apparent_encoding #配置编码
 return r.text
except:
 return "产生异常"
```

关于reqeusts的异常在这里可以看到详细内容： http://www.python-requests.org/en/master/api/#exceptions
所有的异常都是在requests.excepitons中

Exceptions

exception **requests.RequestException**(**args*, ***kwargs*) [source]

There was an ambiguous exception that occurred while handling your request.

exception **requests.ConnectionError**(**args*, ***kwargs*) [source]

A Connection error occurred.

exception **requests.HTTPError**(**args*, ***kwargs*) [source]

An HTTP error occurred.

exception **requests.URLRequired**(**args*, ***kwargs*) [source]

A valid URL is required to make a request.

exception **requests.TooManyRedirects**(**args*, ***kwargs*) [source]

Too many redirects.

exception **requests.ConnectTimeout**(**args*, ***kwargs*) [source]

The request timed out while trying to connect to the remote server.

Requests that produced this error are safe to retry.

exception **requests.ReadTimeout**(**args*, ***kwargs*) [source]

The server did not send any data in the allotted amount of time.

exception **requests.Timeout**(**args*, ***kwargs*) [source]

The request timed out.

Catching this error will catch both **ConnectTimeout** and **ReadTimeout** errors.

从源码我们可以看出
RequestException继承IOError,
HTTPError，ConnectionError,Timeout继承RequestionException，ProxyError，
SSLError继承ConnectionError，
ReadTimeout继承Timeout异常

这里列举了一些常用的异常继承关系，详细的可以看：http://cn.python-requests.org/zh_CN/latest/_modules/requests/exceptions.html#RequestException
通过下面的例子进行简单的演示

```
1 import requests
2 from requests.exceptions import ReadTimeout, ConnectionError, RequestExce
ption
3 try:
4  response = requests.get("http://httpbin.org/get", timeout = 0.5)
5  print(response.status_code)
6 except ReadTimeout:
7  print('Timeout')
8 except ConnectionError:
9  print('Connection error')
```

```
except RequestException:
 print('Error')
```

首先被捕捉的异常是timeout,当把网络断掉的haul就会捕捉到ConnectionError，如果前面异常都没有捕捉到，最后也可以通过RequestExctption捕捉到

3.12 持久连接keep-alive

> requests的keep-alive是基于urllib3，同一会话内的持久连接完全是自动的，同一会话内的所有请求都会自动使用恰当的连接。

也就是说，你无需任何设置，requests会自动实现keep-alive。