

# Requests库入门

WS01

---



嵩天

[www.python123.org](http://www.python123.org)

# The Website is the API ...



## Requests

自动爬取HTML页面  
自动网络请求提交

掌握定向网络数据爬取和网页解析的基本能力

# Python网络爬虫与信息提取



04X -Tian



# Requests库的安装



**Requests**  
*http for humans*

 Star 22,840

Requests is an elegant and simple  
HTTP library for Python, built for  
human beings.

## Stay Informed

Receive updates on new releases and  
upcoming projects.

 Follow @kennethreitz

# Requests: HTTP for Humans

Release v2.12.4. ([Installation](#))

<http://www.python-requests.org>

*Requests is the only Non-GMO HTTP library for Python, safe for human consumption.*

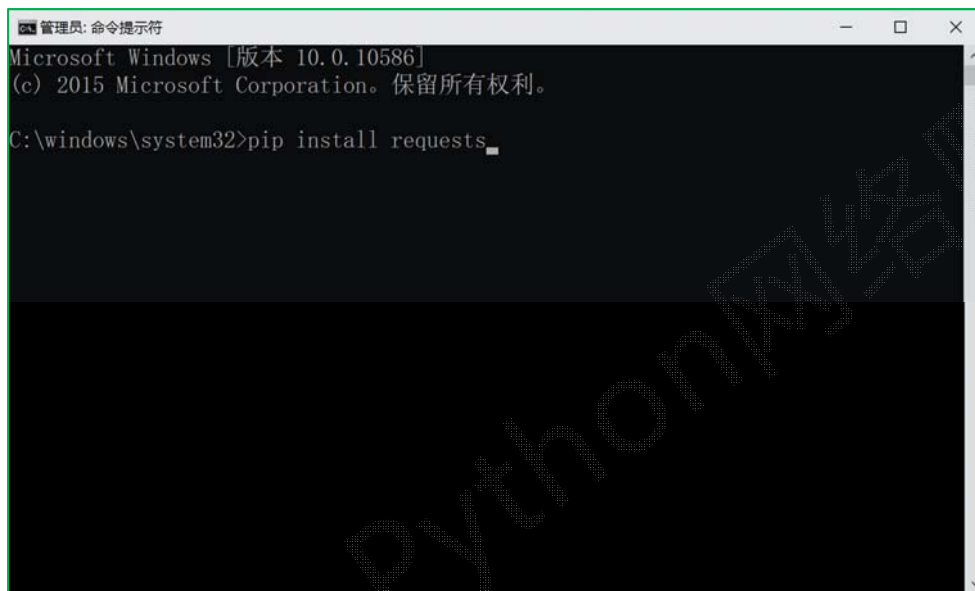
**Warning:** *Recreational use of other HTTP libraries may result in dangerous side-effects, including: security vulnerabilities, verbose code, reinventing the wheel, constantly reading documentation, depression, headaches, or even death.*

*Behold, the power of Requests:*

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type": "User"... '
>>> r.json()
{'private_gists': 419, u'total_private_repos': 77, ...}
```

# Requests库的安装

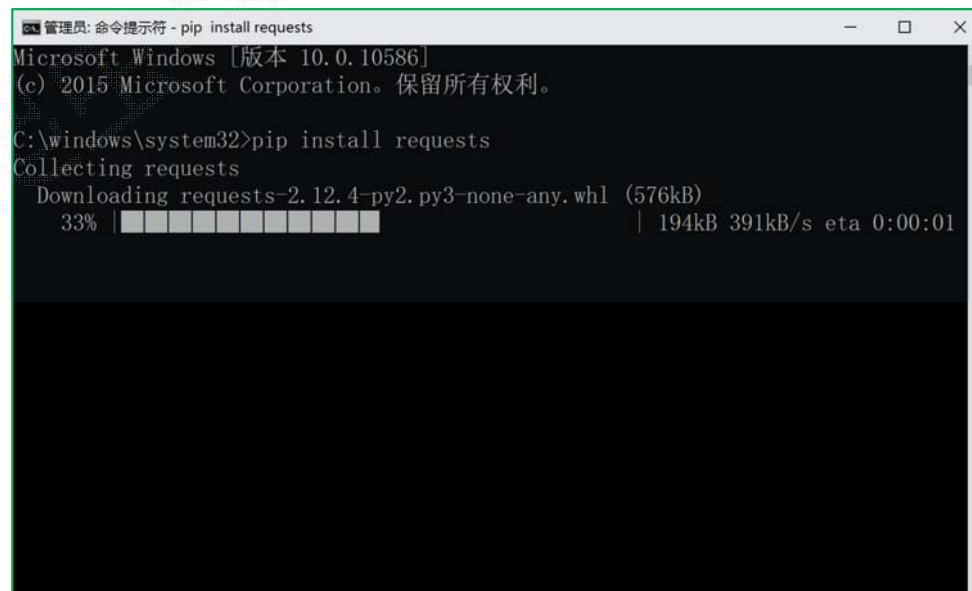
Win平台: “以管理员身份运行” cmd, 执行 `pip install requests`



The screenshot shows a Windows Command Prompt window titled "管理员: 命令提示符". The window content displays the following text:

```
Microsoft Windows [版本 10.0.10586]  
(c) 2015 Microsoft Corporation。保留所有权利。  
  
C:\windows\system32>pip install requests_
```

A large, diagonal watermark reading "Python网络" is visible across the lower half of the image.



```
管理员: 命令提示符 - pip install requests
Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation。保留所有权利。

C:\windows\system32>pip install requests
Collecting requests
  Downloading requests-2.12.4-py2.py3-none-any.whl (576kB)
    33% |██████████████████████| 194kB 391kB/s eta 0:00:01
```

# Requests库的安装小测

```
>>> import requests
>>> r = requests.get("http://www.baidu.com")
>>> print(r.status_code)
200
>>> r.text
'<!DOCTYPE html>\r\n<!--STATUS OK--><html> <head><meta http-equiv
=text/html; charset=utf-8><meta http-equiv=X-UA-Compatible content
t=always name=referrer><link rel=stylesheet type=text/css href=ht
r/www/cache/bdorz/baidu.min.css><title>ç\x99¾å°|ä, \x80ä, \x8bï¼\x8
\x93</title></head> <body link=#0000cc> <div id=wrapper> <div id=
```

# Requests库的7个主要方法

方法	说明
<code>requests.request()</code>	构造一个请求，支撑以下各方法的基础方法
<code>requests.get()</code>	获取HTML网页的主要方法，对应于HTTP的GET
<code>requests.head()</code>	获取HTML网页头信息的方法，对应于HTTP的HEAD
<code>requests.post()</code>	向HTML网页提交POST请求的方法，对应于HTTP的POST
<code>requests.put()</code>	向HTML网页提交PUT请求的方法，对应于HTTP的PUT
<code>requests.patch()</code>	向HTML网页提交局部修改请求，对应于HTTP的PATCH
<code>requests.delete()</code>	向HTML页面提交删除请求，对应于HTTP的DELETE

# Requests库的get()方法



# requests.get()

```
r = requests.get(url)
```

返回一个包含服务器  
资源的Response对象

Response

构造一个向服务器请求  
资源的Request对象

Request

```
requests.get(url, params=None, **kwargs)
```

- **url** : 拟获取页面的url链接
- **params** : url中的额外参数，字典或字节流格式，可选
- **\*\*kwargs**: 12个控制访问的参数

requests.get(url, params=None, \*\*kwargs)

```
def get(url, params=None, **kwargs):
```

```
    """Sends a GET request.
```

```
    :param url: URL for the new :class:`Request` object.
```

```
    :param params: (optional) Dictionary or bytes to be sent in the query string for the
```

```
    :param \*\*kwargs: Optional arguments that ``request`` takes.
```

```
    :return: :class:`Response` object
```

```
    :rtype: requests.Response
```

```
    """
```

```
    kwargs.setdefault('allow_redirects', True)
```

```
    return request('get', url, params=params, **kwargs)
```

# Requests库的2个重要对象

```
r = requests.get(url)
```

Response

Request

Response对象包含爬虫返回的内容

# Response对象

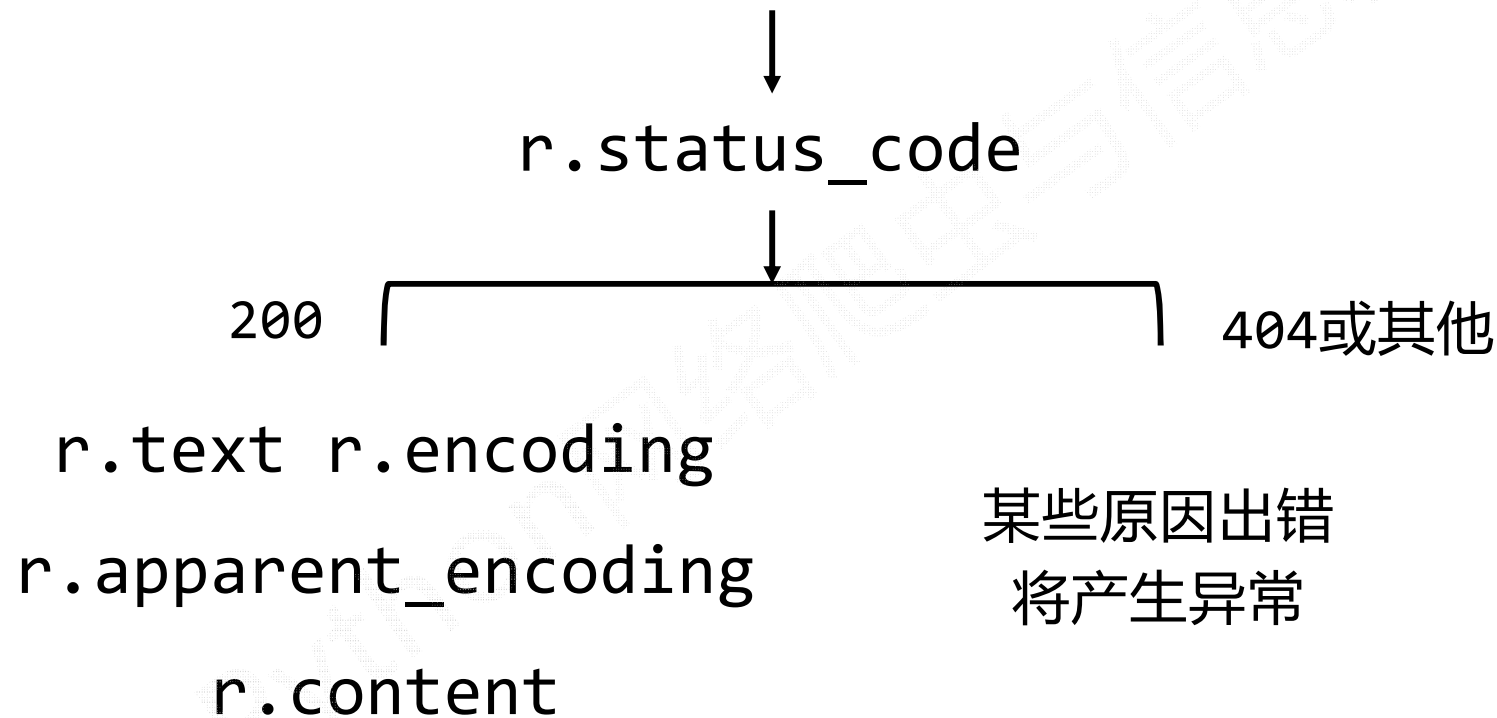
```
>>> import requests
>>> r = requests.get("http://www.baidu.com")
>>> print(r.status_code)
200
>>> type(r)
<class 'requests.models.Response'>
>>> r.headers
{'Cache-Control': 'private, no-cache, no-store, proxy-revalidate,
ection': 'Keep-Alive', 'Transfer-Encoding': 'chunked', 'Server':
```

Response对象包含服务器返回的所有信息，也包含请求的Request信息

# Response对象的属性 (1)

属性	说明
<code>r.status_code</code>	HTTP请求的返回状态，200表示连接成功，404表示失败
<code>r.text</code>	HTTP响应内容的字符串形式，即，url对应的页面内容
<code>r.encoding</code>	从HTTP header中猜测的响应内容编码方式
<code>r.apparent_encoding</code>	从内容中分析出的响应内容编码方式（备选编码方式）
<code>r.content</code>	HTTP响应内容的二进制形式

# Response对象的属性



```
>>> r = requests.get("http://www.baidu.com")
>>> r.status_code
200
>>> r.text
'<!DOCTYPE html>\r\n<!--STATUS OK--><html> <head><meta http-equiv=content-type c
ontent=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge>
<meta content=always name=referrer><link rel=stylesheet type=text/css href=http:
//s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>ç\x99¼å°|ä,\x80ä,\x8bï¼
>>> r.encoding
'ISO-8859-1'
>>> r.apparent_encoding
'utf-8'
>>> r.encoding = "utf-8"
>>> r.text
'<!DOCTYPE html>\r\n<!--STATUS OK--><html> <head><meta http-equiv=content-type c
ontent=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge>
<meta content=always name=referrer><link rel=stylesheet type=text/css href=http:
//s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>百度一下, 你就知道</titl
```



# 理解Response的编码

<code>r.encoding</code>	从HTTP header中猜测的响应内容编码方式
<code>r.apparent_encoding</code>	从内容中分析出的响应内容编码方式（备选编码方式）

`r.encoding`：如果header中不存在charset，则认为编码为ISO-8859-1

`r.text`根据`r.encoding`显示网页内容

`r.apparent_encoding`：根据网页内容分析出的编码方式  
可以看作是`r.encoding`的备选

```
r = requests.get(url)
```

# 爬取网页的通用代码框架

# 理解Requests库的异常

```
r = requests.get(url)
```



Exception

网络连接有风险，异常处理很重要

# 理解Requests库的异常

异常	说明
<code>requests.ConnectionError</code>	网络连接错误异常，如DNS查询失败、拒绝连接等
<code>requests.HTTPError</code>	HTTP错误异常
<code>requests.URLRequired</code>	URL缺失异常
<code>requests.TooManyRedirects</code>	超过最大重定向次数，产生重定向异常
<code>requests.ConnectTimeout</code>	连接远程服务器超时异常
<code>requests.Timeout</code>	请求URL超时，产生超时异常

# 理解Response的异常

<code>r.raise_for_status()</code>	如果不是200，产生异常 <code>requests.HTTPError</code>
-----------------------------------	--

```
r = requests.get(url)
```

`r.raise_for_status()`在方法内部判断`r.status_code`是否等于200，不需要增加额外的if语句，该语句便于利用try-except进行异常处理

# 爬取网页的通用代码框架

```
import requests
```

```
def getHTMLText(url):
```

```
    try:
```

```
        r = requests.get(url, timeout=30)
```

```
        r.raise_for_status() #如果状态不是200, 引发HTTPError异常
```

```
        r.encoding = r.apparent_encoding
```

```
        return r.text
```

```
    except:
```

```
        return "产生异常"
```

```
if __name__ == "__main__":
```

```
    url = "http://www.baidu.com"
```

```
    print(getHTMLText(url))
```

# 爬取网页的通用代码框架

```
if __name__ == "__main__":  
    url = "http://www.baidu.com"  
    print(getHTMLText(url))
```

```
>>>  
<!DOCTYPE html>  
<!--STATUS OK--><html> <head><meta http-equiv=content-ty  
rset=utf-8><meta http-equiv=X-UA-Compatible content=IE=E  
name=referrer><link rel=stylesheet type=text/css href=h  
www/cache/bdorz/baidu.min.css><title>百度一下, 你就知道</t  
=#0000cc> <div id=wrapper> <div id=head> <div class=head
```

```
if __name__ == "__main__":  
    url = "www.baidu.com"  
    print(getHTMLText(url))
```

```
>>>  
产生异常
```





# HTTP协议及Requests库方法

# Requests库的7个主要方法

方法	说明
<code>requests.request()</code>	构造一个请求，支撑以下各方法的基础方法
<code>requests.get()</code>	获取HTML网页的主要方法，对应于HTTP的GET
<code>requests.head()</code>	获取HTML网页头信息的方法，对应于HTTP的HEAD
<code>requests.post()</code>	向HTML网页提交POST请求的方法，对应于HTTP的POST
<code>requests.put()</code>	向HTML网页提交PUT请求的方法，对应于HTTP的PUT
<code>requests.patch()</code>	向HTML网页提交局部修改请求，对应于HTTP的PATCH
<code>requests.delete()</code>	向HTML页面提交删除请求，对应于HTTP的DELETE

# HTTP协议

**HTTP** , Hypertext Transfer Protocol , 超文本传输协议

HTTP是一个基于“请求与响应”模式的、无状态的应用层协议

HTTP协议采用URL作为定位网络资源的标识，URL格式如下：

`http://host[:port][path]`

**host**: 合法的Internet主机域名或IP地址

**port**: 端口号，缺省端口为80

**path**: 请求资源的路径

# HTTP协议

**HTTP** , Hypertext Transfer Protocol , 超文本传输协议

HTTP URL实例 :

`http://www.bit.edu.cn`

`http://220.181.111.188/duty`

HTTP URL的理解 :

URL是通过HTTP协议存取资源的Internet路径 , 一个URL对应一个数据资源

# HTTP协议对资源的操作

方法	说明
GET	请求获取URL位置的资源
HEAD	请求获取URL位置资源的响应消息报告，即获得该资源的头部信息
POST	请求向URL位置的资源后附加新的数据
PUT	请求向URL位置存储一个资源，覆盖原URL位置的资源
PATCH	请求局部更新URL位置的资源，即改变该处资源的部分内容
DELETE	请求删除URL位置存储的资源

# HTTP协议对资源的操作



通过URL和命令管理资源，操作独立无状态，网络通道及服务器成为了黑盒子

# 理解PATCH和PUT的区别

假设URL位置有一组数据UserInfo，包括UserID、UserName等20个字段

需求：用户修改了UserName，其他不变

- 采用PATCH，仅向URL提交UserName的局部更新请求
- 采用PUT，必须将所有20个字段一并提交到URL，未提交字段被删除

PATCH的最主要好处：节省网络带宽

# HTTP协议与Requests库

HTTP协议方法	Requests库方法	功能一致性
GET	<code>requests.get()</code>	一致
HEAD	<code>requests.head()</code>	一致
POST	<code>requests.post()</code>	一致
PUT	<code>requests.put()</code>	一致
PATCH	<code>requests.patch()</code>	一致
DELETE	<code>requests.delete()</code>	一致



# Requests库的head()方法

```
>>> r = requests.head('http://httpbin.org/get')
>>> r.headers
{'Content-Length': '238', 'Access-Control-Allow-Origin': '*', 'Access-
Control-Allow-Credentials': 'true', 'Content-Type':
'application/json', 'Server': 'nginx', 'Connection': 'keep-alive',
'Date': 'Sat, 18 Feb 2017 12:07:44 GMT'}
>>> r.text
''
```

# Requests库的post()方法

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

向URL POST一个字典  
自动编码为form ( 表单 )

# Requests库的post()方法

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

```
>>> print(r.text)
```

```
{ ...
```

```
  "data": "ABC"
```

```
  "form": {},
```

```
}
```

向URL POST一个字符串

自动编码为data

# Requests库的put()方法

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



# Requests库主要方法解析

# Requests库的7个主要方法

方法	说明
<code>requests.request()</code>	构造一个请求，支撑以下各方法的基础方法
<code>requests.get()</code>	获取HTML网页的主要方法，对应于HTTP的GET
<code>requests.head()</code>	获取HTML网页头信息的方法，对应于HTTP的HEAD
<code>requests.post()</code>	向HTML网页提交POST请求的方法，对应于HTTP的POST
<code>requests.put()</code>	向HTML网页提交PUT请求的方法，对应于HTTP的PUT
<code>requests.patch()</code>	向HTML网页提交局部修改请求，对应于HTTP的PATCH
<code>requests.delete()</code>	向HTML页面提交删除请求，对应于HTTP的DELETE

`requests.request(method, url, **kwargs)`

- **method** : 请求方式，对应get/put/post等7种
- **url** : 拟获取页面的url链接
- **\*\*kwargs**: 控制访问的参数，共13个

`requests.request(method, url, **kwargs)`

- **method** : 请求方式

`r = requests.request('GET', url, **kwargs)`

`r = requests.request('HEAD', url, **kwargs)`

`r = requests.request('POST', url, **kwargs)`

`r = requests.request('PUT', url, **kwargs)`

`r = requests.request('PATCH', url, **kwargs)`

`r = requests.request('delete', url, **kwargs)`

`r = requests.request('OPTIONS', url, **kwargs)`



`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数，均为可选项

params : 字典或字节序列，作为参数增加到url中

```
>>> kv = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.request('GET', 'http://python123.io/ws', params=kv)
>>> print(r.url)
http://python123.io/ws?key1=value1&key2=value2
```

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数，均为可选项

params : 字典或字节序列，作为参数增加到url中

data : 字典、字节序列或文件对象，作为Request的内容

```
>>> kv = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.request('POST', 'http://python123.io/ws', data=kv)
>>> body = '主体内容'
>>> r = requests.request('POST', 'http://python123.io/ws', data=body)
```

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数，均为可选项

params : 字典或字节序列，作为参数增加到url中

data : 字典、字节序列或文件对象，作为Request的内容

json : JSON格式的数据，作为Request的内容

```
>>> kv = {'key1': 'value1'}
```

```
>>> r = requests.request('POST', 'http://python123.io/ws', json=kv)
```

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数，均为可选项

params : 字典或字节序列，作为参数增加到url中

data : 字典、字节序列或文件对象，作为Request的内容

json : JSON格式的数据，作为Request的内容

headers : 字典，HTTP定制头

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

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

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数，均为可选项

params : 字典或字节序列，作为参数增加到url中

data : 字典、字节序列或文件对象，作为Request的内容

json : JSON格式的数据，作为Request的内容

headers : 字典，HTTP定制头

cookies : 字典或CookieJar，Request中的cookie

auth : 元组，支持HTTP认证功能

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数 (续)

`files` : 字典类型, 传输文件

```
>>> fs = {'file': open('data.xls', 'rb')}
```

```
>>> r = requests.request('POST', 'http://python123.io/ws', files=fs)
```

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数 (续)

`files` : 字典类型, 传输文件

`timeout` : 设定超时时间, 秒为单位

```
>>> r = requests.request('GET', 'http://www.baidu.com', timeout=10)
```

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数 ( 续 )

`files` : 字典类型, 传输文件

`timeout` : 设定超时时间, 秒为单位

`proxies` : 字典类型, 设定访问代理服务器, 可以增加登录认证

```
>>> pxs = { 'http': 'http://user:pass@10.10.10.1:1234'  
            'https': 'https://10.10.10.1:4321' }
```

```
>>> r = requests.request('GET', 'http://www.baidu.com', proxies=pxs)
```



`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数（续）

`files` : 字典类型，传输文件

`timeout` : 设定超时时间，秒为单位

`proxies` : 字典类型，设定访问代理服务器，可以增加登录认证

`allow_redirects` : True/False，默认为True，重定向开关

`stream` : True/False，默认为True，获取内容立即下载开关

`verify` : True/False，默认为True，认证SSL证书开关

`cert` : 本地SSL证书路径

`requests.request(method, url, **kwargs)`

- **\*\*kwargs**: 控制访问的参数，均为可选项

`params`

`data`

`json`

`headers`

`cookies`

`auth`

`files`

`timeout`

`proxies`

`allow_redirects`

`stream`

`verify`

`cert`

```
requests.get(url, params=None, **kwargs)
```

- **url** : 拟获取页面的url链接
- **params** : url中的额外参数，字典或字节流格式，可选
- **\*\*kwargs**: 12个控制访问的参数

```
requests.head(url, **kwargs)
```

- **url** : 拟获取页面的url链接
- **\*\*kwargs**: 12个控制访问的参数

```
requests.post(url, data=None, json=None, **kwargs)
```

- **url** : 拟更新页面的url链接
- **data** : 字典、字节序列或文件, Request的内容
- **json** : JSON格式的数据, Request的内容
- **\*\*kwargs**: 12个控制访问的参数

```
requests.put(url, data=None, **kwargs)
```

- **url** : 拟更新页面的url链接
- **data** : 字典、字节序列或文件, Request的内容
- **\*\*kwargs**: 12个控制访问的参数

`requests.patch(url, data=None, **kwargs)`

- **url** : 拟更新页面的url链接
- **data** : 字典、字节序列或文件, Request的内容
- **\*\*kwargs**: 12个控制访问的参数

```
requests.delete(url, **kwargs)
```

- **url** : 拟删除页面的url链接
- **\*\*kwargs**: 12个控制访问的参数

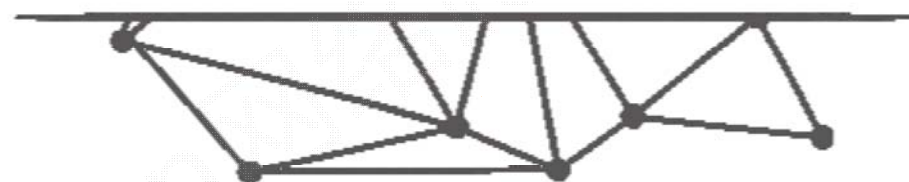


```
requests.get(url, params=None, **kwargs)
```

- **url** : 拟获取页面的url链接
- **params** : url中的额外参数，字典或字节流格式，可选
- **\*\*kwargs**: 12个控制访问的参数



## 单元小结



# Requests库入门

`requests.request()`

`requests.get()`

`requests.head()`

`requests.post()`

`requests.put()`

`requests.patch()`

`requests.delete()`

```
try:
    r = requests.get(url, timeout=30)
    r.raise_for_status()
    r.encoding = r.apparent_encoding
    return r.text
except:
    return "产生异常"
```

爬取网页的通用代码框架