# [Python Requests快速入门](http://blog.csdn.net/iloveyin/article/details/21444613)

# 快速上手

迫不及待了吗？本页内容为如何入门Requests提供了很好的指引。其假设你已经安装了Requests。如果还没有， 去 [安装](http://cn.python-requests.org/en/latest/user/install.html" \l "install" \t "http://blog.csdn.net/iloveyin/article/details/_blank) 一节看看吧。

首先，确认一下：

* Requests [已安装](http://cn.python-requests.org/en/latest/user/install.html" \l "install" \t "http://blog.csdn.net/iloveyin/article/details/_blank)
* Requests是 [最新的](http://cn.python-requests.org/en/latest/community/updates.html" \l "updates" \t "http://blog.csdn.net/iloveyin/article/details/_blank)

让我们从一些简单的示例开始吧。

## 发送请求

使用Requests发送网络请求非常简单。

一开始要导入Requests模块:

>>> **import** requests

然后，尝试获取某个网页。本例子中，我们来获取Github的公共时间线

>>> r = requests.get**(**'https://github.com/timeline.json'**)**

现在，我们有一个名为 r 的 **Response** 对象。可以从这个对象中获取所有我们想要的信息。

Requests简便的API意味着所有HTTP请求类型都是显而易见的。例如，你可以这样发送一个HTTP POST请求:

>>> r = requests.post**(**"http://httpbin.org/post"**)**

漂亮，对吧？那么其他HTTP请求类型：PUT， DELETE， HEAD以及OPTIONS又是如何的呢？都是一样的简单:

>>> r = requests.put**(**"http://httpbin.org/put"**)**>>> r = requests.delete**(**"http://httpbin.org/delete"**)**>>> r = requests.head**(**"http://httpbin.org/get"**)**>>> r = requests.options**(**"http://httpbin.org/get"**)**

都很不错吧，但这也仅是Requests的冰山一角呢。

## 为URL传递参数

你也许经常想为URL的查询字符串(query string)传递某种数据。如果你是手工构建URL，那么数据会以键/值 对的形式置于URL中，跟在一个问号的后面。例如，httpbin.org/get?key=val 。 Requests允许你使用 params 关键字参数，以一个字典来提供这些参数。举例来说，如果你想传递 key1=value1 和 key2=value2 到 httpbin.org/get ，那么你可以使用如下代码:

>>> payload = **{**'key1'**:** 'value1'**,** 'key2'**:** 'value2'**}**>>> r = requests.get**(**"http://httpbin.org/get"**,** params=payload**)**

通过打印输出该URL，你能看到URL已被正确编码:

>>> **print** r.urlu'http://httpbin.org/get?key2=value2&key1=value1'

## 响应内容

我们能读取服务器响应的内容。再次以Github时间线为例:

>>> **import** requests>>> r = requests.get**(**'https://github.com/timeline.json'**)**>>> r.text'[{"repository":{"open\_issues":0,"url":"https://github.com/...

Requests会自动解码来自服务器的内容。大多数unicode字符集都能被无缝地解码。

请求发出后，Requests会基于HTTP头部对响应的编码作出有根据的推测。当你访问r.text 之时，Requests会使用其推测的文本编码。你可以找出Requests使用了什么编码，并且能够使用 r.encoding 属性来改变它:

>>> r.encoding'utf-8'>>> r.encoding = 'ISO-8859-1'

如果你改变了编码，每当你访问 r.text ，Request都将会使用 r.encoding 的新值。

在你需要的情况下，Requests也可以使用定制的编码。如果你创建了自己的编码，并使用codecs 模块进行注册，你就可以轻松地使用这个解码器名称作为 r.encoding 的值， 然后由Requests来为你处理编码。

## 二进制响应内容

你也能以字节的方式访问请求响应体，对于非文本请求:

>>> r.contentb'[{"repository":{"open\_issues":0,"url":"https://github.com/...

Requests会自动为你解码 gzip 和 deflate 传输编码的响应数据。

例如，以请求返回的二进制数据创建一张图片，你可以使用如下代码:

>>> **from** PIL **import** Image>>> **from** StringIO **import** StringIO>>> i = Image.open**(**StringIO**(**r.content**))**

## JSON响应内容

Requests中也有一个内置的JSON解码器，助你处理JSON数据:

>>> **import** requests>>> r = requests.get**(**'https://github.com/timeline.json'**)**>>> r.json**()**[{u'repository': {u'open\_issues': 0, u'url': 'https://github.com/...

如果JSON解码失败， r.json 就会抛出一个异常。

## 原始响应内容

在罕见的情况下你可能想获取来自服务器的原始套接字响应，那么你可以访问 r.raw 。 如果你确实想这么干，那请你确保在初始请求中设置了 stream=True 。具体的你可以这么做:

>>> r = requests.get**(**'https://github.com/timeline.json'**,** stream=True**)**>>> r.raw<requests.packages.urllib3.response.HTTPResponse object at 0x101194810>>>> r.raw.read**(**10**)**'\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\x03'

## 定制请求头

如果你想为请求添加HTTP头部，只要简单地传递一个 dict 给 headers 参数就可以了。

例如，在前一个示例中我们没有指定content-type:

>>> **import** json>>> url = 'https://api.github.com/some/endpoint'>>> payload = **{**'some'**:** 'data'**}**>>> headers = **{**'content-type'**:** 'application/json'**}**

>>> r = requests.post**(**url**,** data=json.dumps**(**payload**),** headers=headers**)**

## 更加复杂的POST请求

通常，你想要发送一些编码为表单形式的数据—非常像一个HTML表单。 要实现这个，只需简单地传递一个字典给 data 参数。你的数据字典 在发出请求时会自动编码为表单形式:

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

很多时候你想要发送的数据并非编码为表单形式的。如果你传递一个 string 而不是一个dict ，那么数据会被直接发布出去。

例如，Github API v3接受编码为JSON的POST/PATCH数据:

>>> **import** json>>> url = 'https://api.github.com/some/endpoint'>>> payload = **{**'some'**:** 'data'**}**

>>> r = requests.post**(**url**,** data=json.dumps**(**payload**))**

## POST一个多部分编码(Multipart-Encoded)的文件

Requests使得上传多部分编码文件变得很简单:

>>> url = 'http://httpbin.org/post'>>> files = **{**'file'**:** open**(**'report.xls'**,** 'rb'**)}**

>>> r = requests.post**(**url**,** files=files**)**>>> r.text{ ... "files": { "file": "<censored...binary...data>" }, ...}

你可以显式地设置文件名:

>>> url = 'http://httpbin.org/post'>>> files = **{**'file'**:** **(**'report.xls'**,** open**(**'report.xls'**,** 'rb'**))}**

>>> r = requests.post**(**url**,** files=files**)**>>> r.text{ ... "files": { "file": "<censored...binary...data>" }, ...}

如果你想，你也可以发送作为文件来接收的字符串:

>>> url = 'http://httpbin.org/post'>>> files = **{**'file'**:** **(**'report.csv'**,** 'some,data,to,send\nanother,row,to,send\n'**)}**

>>> r = requests.post**(**url**,** files=files**)**>>> r.text{ ... "files": { "file": "some,data,to,send\\nanother,row,to,send\\n" }, ...}

## 响应状态码

我们可以检测响应状态码:

>>> r = requests.get**(**'http://httpbin.org/get'**)**>>> r.status\_code200

为方便引用，Requests还附带了一个内置的状态码查询对象:

>>> r.status\_code == requests.codes.okTrue

如果发送了一个失败请求(非200响应)，我们可以通过 **Response.raise\_for\_status()** 来抛出异常:

>>> bad\_r = requests.get**(**'http://httpbin.org/status/404'**)**>>> bad\_r.status\_code404

>>> bad\_r.raise\_for\_status**()Traceback (most recent call last):**

File "requests/models.py", line 832, in raise\_for\_status

**raise** http\_errorrequests.exceptions.HTTPError: 404 Client Error

但是，由于我们的例子中 r 的 status\_code 是 200 ，当我们调用 raise\_for\_status() 时，得到的是:

>>> r.raise\_for\_status**()**None

一切都挺和谐哈。

## 响应头

我们可以查看以一个**[Python](http://lib.csdn.net/base/11" \o "Python知识库" \t "http://blog.csdn.net/iloveyin/article/details/_blank)**字典形式展示的服务器响应头:

>>> r.headers{ 'status': '200 OK', 'content-encoding': 'gzip', 'transfer-encoding': 'chunked', 'connection': 'close', 'server': 'nginx/1.0.4', 'x-runtime': '148ms', 'etag': '"e1ca502697e5c9317743dc078f67693f"', 'content-type': 'application/json; charset=utf-8'}

但是这个字典比较特殊：它是仅为HTTP头部而生的。根据 [RFC 2616](http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html" \t "http://blog.csdn.net/iloveyin/article/details/_blank) ， HTTP头部是大小写不敏感的。

因此，我们可以使用任意大写形式来访问这些响应头字段:

>>> r.headers**[**'Content-Type'**]**'application/json; charset=utf-8'

>>> r.headers.get**(**'content-type'**)**'application/json; charset=utf-8'

如果某个响应头字段不存在，那么它的默认值为 None

>>> r.headers**[**'X-Random'**]**None

## Cookies

如果某个响应中包含一些Cookie，你可以快速访问它们:

>>> url = 'http://example.com/some/cookie/setting/url'>>> r = requests.get**(**url**)**

>>> r.cookies**[**'example\_cookie\_name'**]**'example\_cookie\_value'

要想发送你的cookies到服务器，可以使用 cookies 参数:

>>> url = 'http://httpbin.org/cookies'>>> cookies = dict**(**cookies\_are='working'**)**

>>> r = requests.get**(**url**,** cookies=cookies**)**>>> r.text'{"cookies": {"cookies\_are": "working"}}'

## 重定向与请求历史

使用GET或OPTIONS时，Requests会自动处理位置重定向。

Github将所有的HTTP请求重定向到HTTPS。可以使用响应对象的 history 方法来追踪重定向。 我们来看看Github做了什么:

>>> r = requests.get**(**'http://github.com'**)**>>> r.url'https://github.com/'>>> r.status\_code200>>> r.history[<Response [301]>]

**Response.history** 是一个:class:Request 对象的列表，为了完成请求而创建了这些对象。这个对象列表按照从最老到最近的请求进行排序。

如果你使用的是GET或OPTIONS，那么你可以通过 allow\_redirects 参数禁用重定向处理:

>>> r = requests.get**(**'http://github.com'**,** allow\_redirects=False**)**>>> r.status\_code301>>> r.history[]

如果你使用的是POST，PUT，PATCH，DELETE或HEAD，你也可以启用重定向:

>>> r = requests.post**(**'http://github.com'**,** allow\_redirects=True**)**>>> r.url'https://github.com/'>>> r.history[<Response [301]>]

## 超时

你可以告诉requests在经过以 timeout 参数设定的秒数时间之后停止等待响应:

>>> requests.get**(**'http://github.com'**,** timeout=0.001**)Traceback (most recent call last):**

File "<stdin>", line 1, in <module>requests.exceptions.Timeout: HTTPConnectionPool(host='github.com', port=80): Request timed out. (timeout=0.001)

注:

timeout 仅对连接过程有效，与响应体的下载无关。

## 错误与异常

遇到网络问题（如：DNS查询失败、拒绝连接等）时，Requests会抛出一个**ConnectionError** 异常。

遇到罕见的无效HTTP响应时，Requests则会抛出一个 **HTTPError** 异常。

若请求超时，则抛出一个 **Timeout** 异常。

若请求超过了设定的最大重定向次数，则会抛出一个 **TooManyRedirects** 异常。

所有Requests显式抛出的异常都继承自 **requests.exceptions.RequestException** 。

requests是**[Python](http://lib.csdn.net/base/11" \o "Python知识库" \t "http://blog.csdn.net/alpha5/article/details/_blank)**的一个HTTP客户端库，跟urllib，urllib2类似，那为什么要用requests而不用urllib2呢？官方文档中是这样说明的：

python的标准库urllib2提供了大部分需要的HTTP功能，但是API太逆天了，一个简单的功能就需要一大堆代码。

我也看了下requests的文档，确实很简单，适合我这种懒人。下面就是一些简单指南。

插播个好消息！刚看到requests有了中文翻译版，建议英文不好的看看，内容也比我的博客好多了，具体链接是：http://cn.python-requests.org/en/latest/(不过是v1.1.0版，另抱歉，之前贴错链接了)。

## 1. 安装

安装很简单，我是win系统，就在[这里](http://docs.python-requests.org/en/latest/user/install/" \l "install" \t "http://blog.csdn.net/alpha5/article/details/_blank)下载了安装包（网页中download the zipball处链接），然后$ python setup.py install就装好了。  
当然，有easy\_install或pip的朋友可以直接使用：easy\_install requests或者pip install requests来安装。  
至于linux用户，[这个页面](http://docs.python-requests.org/en/latest/user/install.html" \l "install" \t "http://blog.csdn.net/alpha5/article/details/_blank)还有其他安装方法。  
测试：在IDLE中输入import requests，如果没提示错误，那说明已经安装成功了！

## 2. 小试牛刀

>>>**import** **requests**>>> r = requests.get('http://www.zhidaow.com') *# 发送请求*>>> r.status\_code *# 返回码* 200>>> r.headers['content-type'] *# 返回头部信息*'text/html; charset=utf8'>>> r.encoding *# 编码信息*'utf-8'>>> r.text *#内容部分（PS，由于编码问题，建议这里使用r.content）*u'<!DOCTYPE html>**\n**<html xmlns="http://www.w3.org/1999/xhtml"...'...

是不是很简单？比urllib2和urllib简单直观的多？！那请接着看快速指南吧。

## 3. 快速指南

### 3.1 发送请求

发送请求很简单的，首先要导入requests模块：

>>>**import** **requests**

接下来让我们获取一个网页，例如我个人博客的首页：

>>>r = requests.get('http://www.zhidaow.com')

接下来，我们就可以使用这个r的各种方法和函数了。  
另外，HTTP请求还有很多类型，比如POST,PUT,DELETE,HEAD,OPTIONS。也都可以用同样的方式实现：

>>> r = requests.post("http://httpbin.org/post")>>> r = requests.put("http://httpbin.org/put")>>> r = requests.delete("http://httpbin.org/delete")>>> r = requests.head("http://httpbin.org/get")>>> r = requests.options("http://httpbin.org/get")

因为目前我还没用到这些，所以没有深入研究。

### 3.2 在URLs中传递参数

有时候我们需要在URL中传递参数，比如在采集百度搜索结果时，我们wd参数（搜索词）和rn参数（搜素结果数量），你可以手工组成URL，requests也提供了一种看起来很NB的方法：

>>> payload = {'wd': '张亚楠', 'rn': '100'}>>> r = requests.get("http://www.baidu.com/s", params=payload)>>> **print** r.urlu'http://www.baidu.com/s?rn=100&wd=*%E*5%BC%A0*%E*4%BA%9A*%E*6%A5%A0'

上面wd=的乱码就是“张亚楠”的转码形式。（好像参数按照首字母进行了排序。）

### 3.3 获取响应内容

可以通过r.text来获取网页的内容。

>>> r = requests.get('https://www.zhidaow.com')>>> r.textu'<!DOCTYPE html>**\n**<html xmlns="http://www.w3.org/1999/xhtml"...'

文档里说，requests会自动将内容转码。大多数unicode字体都会无缝转码。但我在cygwin下使用时老是出现UnicodeEncodeError错误，郁闷。倒是在python的IDLE中完全正常。  
另外，还可以通过r.content来获取页面内容。

>>> r = requests.get('https://www.zhidaow.com')>>> r.contentb'<!DOCTYPE html>**\n**<html xmlns="http://www.w3.org/1999/xhtml"...'

文档中说r.content是以字节的方式去显示，所以在IDLE中以b开头。但我在cygwin中用起来并没有，下载网页正好。所以就替代了urllib2的urllib2.urlopen(url).read()功能。（基本上是我用的最多的一个功能。）

### 3.4 获取网页编码

可以使用r.encoding来获取网页编码。

>>> r = requests.get('http://www.zhidaow.com')>>> r.encoding'utf-8'

当你发送请求时，requests会根据HTTP头部来猜测网页编码，当你使用r.text时，requests就会使用这个编码。当然你还可以修改requests的编码形式。

>>> r = requests.get('http://www.zhidaow.com')>>> r.encoding'utf-8'>>>r.encoding = 'ISO-8859-1'

像上面的例子，对encoding修改后就直接会用修改后的编码去获取网页内容。

### 3.5 json

像urllib和urllib2，如果用到json，就要引入新模块，如json和simplejson，但在requests中已经有了内置的函数，r.json()。就拿查询IP的API来说：

>>>r = requests.get('http://ip.taobao.com/service/getIpInfo.php?ip=122.88.60.28')>>>r.json()['data']['country']'中国'

### 3.6 网页状态码

我们可以用r.status\_code来检查网页的状态码。

>>>r = requests.get('http://www.mengtiankong.com')>>>r.status\_code200>>>r = requests.get('http://www.mengtiankong.com/123123/')>>>r.status\_code404>>>r = requests.get('http://www.baidu.com/link?url=QeTRFOS7TuUQRppa0wlTJJr6FfIYI1DJprJukx4Qy0XnsDO\_s9baoO8u1wvjxgqN')>>>r.urlu'http://www.zhidaow.com/>>>r.status\_code200

前两个例子很正常，能正常打开的返回200，不能正常打开的返回404。但第三个就有点奇怪了，那个是百度搜索结果中的302跳转地址，但状态码显示是200，接下来我用了一招让他原形毕露：

>>>r.history(<Response [302]>,)

这里能看出他是使用了302跳转。也许有人认为这样可以通过判断和正则来获取跳转的状态码了，其实还有个更简单的方法：

>>>r = requests.get('http://www.baidu.com/link?url=QeTRFOS7TuUQRppa0wlTJJr6FfIYI1DJprJukx4Qy0XnsDO\_s9baoO8u1wvjxgqN', allow\_redirects = False)>>>r.status\_code302

只要加上一个参数allow\_redirects，禁止了跳转，就直接出现跳转的状态码了，好用吧？我也利用这个在最后一掌做了个简单的获取网页状态码的小应用，原理就是这个。

### 3.7 响应头内容

可以通过r.headers来获取响应头内容。

>>>r = requests.get('http://www.zhidaow.com')>>> r.headers{

'content-encoding': 'gzip',

'transfer-encoding': 'chunked',

'content-type': 'text/html; charset=utf-8';

...}

可以看到是以字典的形式返回了全部内容，我们也可以访问部分内容。

>>> r.headers['Content-Type']'text/html; charset=utf-8'

>>> r.headers.get('content-type')'text/html; charset=utf-8'

### 3.8 设置超时时间

我们可以通过timeout属性设置超时时间，一旦超过这个时间还没获得响应内容，就会提示错误。

>>> requests.get('http://github.com', timeout=0.001)Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>requests.exceptions.Timeout: HTTPConnectionPool(host='github.com', port=80): Request timed out. (timeout=0.001)

### 3.9 代理访问

采集时为避免被封IP，经常会使用代理。requests也有相应的proxies属性。

**import** **requests**

proxies = {

"http": "http://10.10.1.10:3128",

"https": "http://10.10.1.10:1080",}

requests.get("http://www.zhidaow.com", proxies=proxies)

如果代理需要账户和密码，则需这样：

proxies = {

"http": "http://user:pass@10.10.1.10:3128/",}

### 3.10 请求头内容

请求头内容可以用r.request.headers来获取。

>>> r.request.headers{'Accept-Encoding': 'identity, deflate, compress, gzip','Accept': '\*/\*', 'User-Agent': 'python-requests/1.2.3 CPython/2.7.3 Windows/XP'}

### 3.11 自定义请求头部

伪装请求头部是采集时经常用的，我们可以用这个方法来隐藏：

r = requests.get('http://www.zhidaow.com')**print** r.request.headers['User-Agent']*#python-requests/1.2.3 CPython/2.7.3 Windows/XP*

headers = {'User-Agent': 'alexkh'}r = requests.get('http://www.zhidaow.com', headers = headers)**print** r.request.headers['User-Agent']*#alexkh*

### 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。

## 4. 简单应用

### 4.1 获取网页返回码

**def** get\_status(url):

r = requests.get(url, allow\_redirects = False)

**return** r.status\_code

**print** get\_status('http://www.zhidaow.com') *#200***print** get\_status('http://www.zhidaow.com/hi404/')*#404***print** get\_status('http://mengtiankong.com')*#301***print** get\_status('http://www.baidu.com/link?url=QeTRFOS7TuUQRppa0wlTJJr6FfIYI1DJprJukx4Qy0XnsDO\_s9baoO8u1wvjxgqN')*#302***print** get\_status('http://www.huiya56.com/com8.intre.asp?46981.html')*#500*

## 后记

1、官方文档  
requests的具体安装过程请看：http://docs.python-requests.org/en/latest/user/install.html#install  
requests的官方指南文档：http://docs.python-requests.org/en/latest/user/quickstart.html  
requests的高级指南文档：http://docs.python-requests.org/en/latest/user/advanced.html#advanced  
2、本文内容部分翻译自官方文档，部分自己归纳。  
3、大多数用的IDLE格式，累死了，下次直接用编辑器格式，这样更符合我的习惯。  
4、还是那句话，有问题留言或email。  
5、图注：requests官方文档上的一只老鳖。

转载出处：http://www.zhidaow.com/post/python-requests-install-and-brief-introduction