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。我们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。

传递 URL 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.url)
http://httpbin.org/get?key2=value2&key1=value1
```

注意字典里值为 `None` 的键都不会被添加到 URL 的查询字符串里。

你还可以将一个列表作为值传入：

```
>>> payload = {'key1': 'value1', 'key2': ['value2', 'value3']}

>>> r = requests.get('http://httpbin.org/get', params=payload)
>>> print(r.url)
http://httpbin.org/get?key1=value1&key2=value2&key2=value3
```

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
u'[{"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。你可能希望在使用特殊逻辑计算出文本的编码的情况下来修改编码。比如 HTTP 和 XML 自身可以指定编码。这样的话，你应该使用 `r.content` 来

找到编码，然后设置 `r.encoding` 为相应的编码。这样就能使用正确的编码解析 `r.text` 了。
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.content
b'[{"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 io import BytesIO

>>> i = Image.open(BytesIO(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。例如，相应内容是 401 (Unauthorized)，尝试访问 `r.json` 将会抛出 `ValueError: No JSON object could be decoded` 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'
```

但一般情况下，你应该以下面的模式将文本流保存到文件：

```
with open(filename, 'wb') as fd:
    for chunk in r.iter_content(chunk_size):
        fd.write(chunk)
```

使用 `Response.iter_content` 将会处理大量你直接使用 `Response.raw` 不得不处理的。 当流下载时，上面是优先推荐的获取内容方式。

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：

```
>>> url = 'https://api.github.com/some/endpoint'
>>> headers = {'user-agent': 'my-app/0.0.1'}

>>> r = requests.get(url, headers=headers)
```

注意：定制 header 的优先级低于某些特定的信息源，例如：

- 如果在 `.netrc` 中设置了用户认证信息，使用 *headers=* 设置的授权就不会生效。而如果设置了`auth=` 参数，``.netrc`` 的设置就无效了。
- 如果被重定向到别的主机，授权 header 就会被删除。

- 代理授权 header 会被 URL 中提供的代理身份覆盖掉。
- 在我们能判断内容长度的情况下，header 的 Content-Length 会被改写。

更进一步讲，Requests 不会基于定制 header 的具体情况改变自己的行为。只不过在最后的请求中，所有的 header 信息都会被传递进去。

注意：所有的 header 值必须是 `string`、bytestring 或者 unicode。尽管传递 unicode header 也是允许的，但不建议这样做。