Python request模块

一.Request模块基本使用

1.1 发送Http请求s

In [4]:
```python
import requests
r = requests.get('https://api.github.com/events')
```

In [5]:
```python
r = requests.post('http://httpbin.org/post', data = {'key':'value'})
```

In [6]:
```python
r = requests.put('http://httpbin.org/put', data = {'key':'value'})
r = requests.delete('http://httpbin.org/delete')
r = requests.head('http://httpbin.org/get')
r = requests.options('http://httpbin.org/get')
```

r是一个Response对象，我们可以从这个对象中获取所有我们想要的信息．在请求url时可以指定超时信息，也就是在指定的时间内url没有做出响应，那么就终止该http请求，停止等待响应．

In [ ]:
```python
requests.get('http://github.com', timeout=0.001)
```

若在指定的超时时间内，url没有做出响应则报TimeoutError错误．

1.2 传递url参数

在某些通过url进行数据查询的需求中，数据会以键值对的形式出现在url中．产生形如http://172.16.70.233/index?a=1&b=2&c=3这样的url， Requests模块允许使用params关键字参数用一个字符串字典来提供这些参数．

示例代码：

In [8]:
```python
payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.get("http://httpbin.org/get", params=payload)
r.url
```

Out[8]:
```
'http://httpbin.org/get?key1=value1&key2=value2'
```

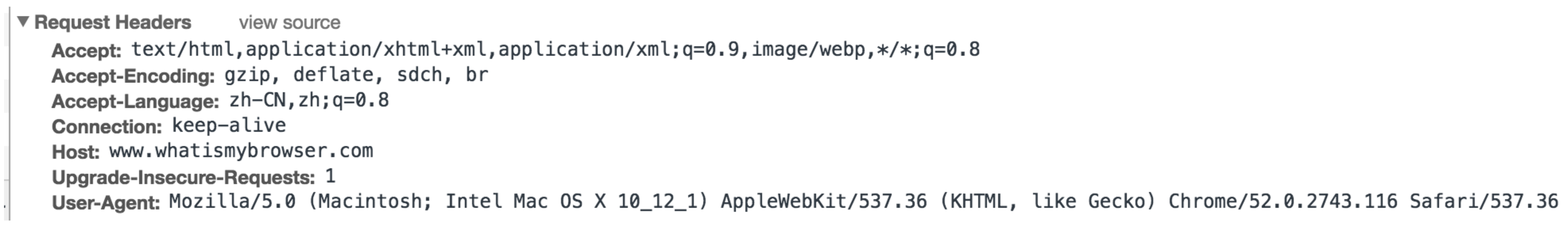
In [9]:
```python
payload = {'key1': 'value1', 'key2': ['value2', 'value3']}

r = requests.get('http://httpbin.org/get', params=payload)
print(r.url)
```

```
http://httpbin.org/get?key1=value1&key2=value2&key2=value3 (http://httpbin.org/get?key1=value1&key2=value2&key2=value3)
```

1.3 定制请求头

在去爬取WEB服务器内容时，为了让代码发出的HTTP请求更像是人类用户，我们需要为发出的HTTP请求附加请求头部信息．HTTP的请求头是每次向网络服务器发送请求的时候传递的一组属性和配置信息，HTTP定义了十几种古怪的请求头信息，不过大多数都不常用，下面的为常用的请求头部信息



Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: zh-CN,zh;q=0.8
Connection: keep-alive
Host: www.whatismybrowser.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36

请求头可以通过 urllib2 或 request 模块自己定义 虽然网站可能会对HTTP请求头的每个属性进行检查 但是最重要的可能被检查的参数信息是 User-Agent 所以无论做什么爬虫项目 建议都要修改User-Agent属性将其设置成不容易引起怀疑的内容 如果是一些警觉性比较高的网站 可能还会检查 比如 Accept-Lanuage 这些属性 请将这些属性妥善配置

In [15]:
```python
url = "https://www.whatismybrowser.com/"
headers = {'user-agent': "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142 Safari/537.36"}
r = requests.get(url, headers=headers)
```

1.4 通过POST请求传参

In [16]:
```python
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",
    "Content-Length": "23",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.22.0"
  },
  "json": null,
  "origin": "218.17.112.191, 218.17.112.191",
  "url": "https://httpbin.org/post"
}
```

In [18]:
```python
url = 'https://api.github.com/some/endpoint'
payload = {'some': 'data'}
r = requests.post(url, json=payload)
r.text
```

Out[18]:
```
'{"message":"Not Found","documentation_url":"https://developer.github.com/v3"}'
```

二.响应内容

2.1 r.text 会自动解码来自服务器的内容，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．

In [10]:
```python
r = requests.get('https://api.github.com/events')
r.encoding
```

Out[10]:
```
'utf-8'
```

2.2 Requests中也有一个内置的JSON解码器用来处理JSON数据

In [ ]:
```
r = requests.get('https://api.github.com/events')
r.json()
```

如果JSON解码失败，r.json()就会抛出一个异常；例如响应内容是401(Unauthorized)，尝试访问r.json()将会抛出ValueError: No JSON object could be decoded 异常. 成功调用r.json()并**不**意味着响应的成功，有的服务器会在失败的响应中包含一个JSON对象(比如 HTTP 500 的错误细节)这种JSON会被解码返回. 要检查请求是否成功，请使用r.raise_for_status()或者检查r.status_code是否和你的期望相同.

2.3 二进制响应内容

对于非文本请求，能以字节的方式访问请求响应体，有以下例子.

In [11]:
```
import requests
r = requests.get("https://www.qq.com")
f = open('123.html', 'wb')
f.write(r.content)
```

Out[11]:
```
244345
```

2.4 响应状态码

In [12]:
```
r.status_code
```

Out[12]:
```
200
```

2.5 响应头

In [13]:
```
r.headers
```

Out[13]:
```
{'Server': 'nginx', 'Date': 'Fri, 02 Aug 2019 03:19:11 GMT', 'Content-Type': 'text/html; charset=GB2312', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive',
'Vary': 'Accept-Encoding, Accept-Encoding, Accept-Encoding', 'Expires': 'Fri, 02 Aug 2019 03:20:11 GMT', 'Cache-Control': 'max-age=60', 'Content-Encoding': 'gzip',
'X-Cache': 'HIT from shenzhen.qq.com'}
```

In [14]:
```
r.headers["Server"]
```

Out[14]:
```
'nginx'
```

三. Cookie

3.1 获取网站响应中的cookie信息

In [31]:
```
url = 'http://example.com/some/cookie/setting/url'
r = requests.get(url)
r.cookies
```

Out[31]:
```
<RequestsCookieJar[]>
```

3.2 将本地Cookie信息发送给服务器

Cookie的返回对象为RequestsCookieJar，它的行为和字典类似，但接口更为完整，适合跨域名跨路径使用. 你还可以把Cookie Jar传到Requests中:

In [24]:
```
jar = requests.cookies.RequestsCookieJar()
jar.set('tasty_cookie', 'yum', domain='httpbin.org', path='/cookies')
jar.set('gross_cookie', 'blech', domain='httpbin.org', path='/elsewhere')
url = 'http://httpbin.org/cookies'
r = requests.get(url, cookies=jar)
r.text
```

Out[24]:
```
'{\n  "cookies": {\n    "tasty_cookie": "yum"\n  }\n}\n'
```

四.Session会话对象

由于HTTP请求是无状态的，会话对象让你能够跨请求保持某些参数，它也会在同一个Session实例发出的所有请求之间保持cookie.

示例代码:

In [35]:
```
import requests
s = requests.Session()
data = {"um_account": "admin", "um_passwd": "P@ssw0rd"}
url = "http://172.16.70.233:8899/api/auth/login"
r = s.post(url, data=data)
print(r.json())
print(r.cookies)
url = "http://172.16.70.233:8899/api/cmdb/cmdb_host_information"
r = s.get(url)
print(r.text)
```

```
{'code': 200, 'message': '登录成功!'}
<RequestsCookieJar[<Cookie login=true for 172.16.70.233/>, <Cookie sessionid=whic80fviz1gbf8lo05zalqyuaotsleo for 172.16.70.233/>]>
{"code": 200, "message": [{"id": 47, "source_addr": "172.16.70.221", "stat_time": 1563873756, "host_user": "root", "host_port": 22, "ssh_conn": 1, "createor": "admi
n"}, {"id": 45, "source_addr": "172.16.70.222", "stat_time": 1563871164, "host_user": "root", "host_port": 22, "ssh_conn": 0, "createor": "admin"}]}
```

In [ ]: