11.1 作为客户端与HTTP服务交互¶

问题¶

你需要通过HTTP协议以客户端的方式访问多种服务。例如，下载数据或者与基于REST的API进行交互。

解决方案¶

对于简单的事情来说，通常使用 `urllib.request` 模块就够了。例如，发送一个简单的HTTP GET请求到远程的服务上，可以这样做：

```
from urllib import request, parse

# Base URL being accessed
url = 'http://httpbin.org/get'

# Dictionary of query parameters (if any)
parms = {
   'name1' : 'value1',
   'name2' : 'value2'
}

# Encode the query string
querystring = parse.urlencode(parms)

# Make a GET request and read the response
u = request.urlopen(url+'?' + querystring)
resp = u.read()
```

如果你需要使用POST方法在请求主体中发送查询参数，可以将参数编码后作为可选参数提供给 `urlopen()` 函数，就像这样：

```
from urllib import request, parse

# Base URL being accessed
url = 'http://httpbin.org/post'

# Dictionary of query parameters (if any)
parms = {
   'name1' : 'value1',
   'name2' : 'value2'
}

# Encode the query string
querystring = parse.urlencode(parms)

# Make a POST request and read the response
u = request.urlopen(url, querystring.encode('ascii'))
resp = u.read()
```

如果你需要在发出的请求中提供一些自定义的HTTP头，例如修改 `user-agent` 字段,可以创建一个包含字段值的字典，并创建一个Request实例然后将其传给 `urlopen()` ，如下：

```
from urllib import request, parse
```

```
...

# Extra headers
headers = {
   'User-agent' : 'none/ofyourbusiness',
   'Spam' : 'Eggs'
}

req = request.Request(url, querystring.encode('ascii'), headers=headers)

# Make a request and read the response
u = request.urlopen(req)
resp = u.read()
```

如果需要交互的服务比上面的例子都要复杂，也许应该去看看 requests 库（https://pypi.python.org/pypi/requests）。例如，下面这个示例采用requests库重新实现了上面的操作：

```
import requests

# Base URL being accessed
url = 'http://httpbin.org/post'

# Dictionary of query parameters (if any)
parms = {
   'name1' : 'value1',
   'name2' : 'value2'
}

# Extra headers
headers = {
   'User-agent' : 'none/ofyourbusiness',
   'Spam' : 'Eggs'
}

resp = requests.post(url, data=parms, headers=headers)

# Decoded text returned by the request
text = resp.text
```

关于requests库，一个值得一提的特性就是它能以多种方式从请求中返回响应结果的内容。从上面的代码来看，`resp.text` 带给我们的是以Unicode解码的响应文本。但是，如果去访问 `resp.content` ，就会得到原始的二进制数据。另一方面，如果访问 `resp.json` ，那么就会得到JSON格式的响应内容。

下面这个示例利用 `requests` 库发起一个HEAD请求，并从响应中提取出一些HTTP头数据的字段：

```
import requests

resp = requests.head('http://www.python.org/index.html')

status = resp.status_code
last_modified = resp.headers['last-modified']
content_type = resp.headers['content-type']
content_length = resp.headers['content-length']
```

下面是一个利用requests通过基本认证登录Pypi的例子：

```
import requests
```

```
resp = requests.get('http://pypi.python.org/pypi?:action=login',
                    auth=('user','password'))
```

下面是一个利用requests将HTTP cookies从一个请求传递到另一个的例子：

```
import requests

# First request
resp1 = requests.get(url)
...

# Second requests with cookies received on first requests
resp2 = requests.get(url, cookies=resp1.cookies)
```

最后但并非最不重要的一个例子是用requests上传内容：

```
import requests
url = 'http://httpbin.org/post'
files = { 'file': ('data.csv', open('data.csv', 'rb')) }

r = requests.post(url, files=files)
```

讨论¶

对于真的很简单HTTP客户端代码，用内置的 `urllib` 模块通常就足够了。但是，如果你要做的不仅仅只是简单的GET或POST请求，那就真的不能再依赖它的功能了。这时候就是第三方模块比如 `requests` 大显身手的时候了。

例如，如果你决定坚持使用标准的程序库而不考虑像 `requests` 这样的第三方库，那么也许就不得不使用底层的 `http.client` 模块来实现自己的代码。比方说，下面的代码展示了如何执行一个HEAD请求：

```
from http.client import HTTPConnection
from urllib import parse

c = HTTPConnection('www.python.org', 80)
c.request('HEAD', '/index.html')
resp = c.getresponse()

print('Status', resp.status)
for name, value in resp.getheaders():
   print(name, value)
```

同样地，如果必须编写涉及代理、认证、cookies以及其他一些细节方面的代码，那么使用 `urllib` 就显得特别别扭和啰嗦。比方说，下面这个示例实现在Python包索引上的认证：

```
import urllib.request

auth = urllib.request.HTTPBasicAuthHandler()
auth.add_password('pypi','http://pypi.python.org','username','password')
opener = urllib.request.build_opener(auth)

r = urllib.request.Request('http://pypi.python.org/pypi?:action=login')
u = opener.open(r)
resp = u.read()

# From here. You can access more pages using opener
```

```
# From here. You can access more pages using opener
...
```

坦白说，所有的这些操作在 `requests` 库中都变得简单的多。

在开发过程中测试HTTP客户端代码常常是很令人沮丧的，因为所有棘手的细节问题都需要考虑（例如cookies、认证、HTTP头、编码方式等）。要完成这些任务，考虑使用httpbin服务（http://httpbin.org）。这个站点会接收发出的请求，然后以JSON的形式将相应信息回传回来。下面是一个交互式的例子：

```
>>> import requests
>>> r = requests.get('http://httpbin.org/get?name=Dave&n=37',
...     headers = { 'User-agent': 'goaway/1.0' })
>>> resp = r.json
>>> resp['headers']
{'User-Agent': 'goaway/1.0', 'Content-Length': '', 'Content-Type': '',
'Accept-Encoding': 'gzip, deflate, compress', 'Connection':
'keep-alive', 'Host': 'httpbin.org', 'Accept': '*/*'}
>>> resp['args']
{'name': 'Dave', 'n': '37'}
>>>
```

在要同一个真正的站点进行交互前，先在 httpbin.org 这样的网站上做实验常常是可取的办法。尤其是当我们面对3次登录失败就会关闭账户这样的风险时尤为有用（不要尝试自己编写HTTP认证客户端来登录你的银行账户）。

尽管本节没有涉及， `request` 库还对许多高级的HTTP客户端协议提供了支持，比如OAuth。 `requests` 模块的文档（http://docs.python-requests.org)质量很高（坦白说比在这短短的一节的篇幅中所提供的任何信息都好），可以参考文档以获得更多地信息。