*.isBullsh.it

Home » Python » Communicating with RESTful APIs in Python

Communicating with RESTful APIs in Python

Written by Balthazar

28 Jun 2012
From Lyon
Tag: Python
Tag: Api

Table of contents

REST defines a way to design an API with which you can consume its ressources using HTTP methods (GET, POST, etc) over URLs. Interacting with such an API basically comes down to sending HTTP requests.

In this article, we'll see which python modules are available to solve this problem, and which one you should use. We'll test all modules with this simple test case: we would like to create a new Github repository using their RESTful API.

Python and HTTP: too many modules!

One known problem with Python is the abundance of modules enabling HTTP communication:

- urllib2
- httplib
- httplib2
- pycurl
- requests

If you're totally new to Python, this might be confusing, so let me try to clarify things a little bit. First things first: httplib will not solve your problem.

> This module defines classes which implement the client side of the HTTP and HTTPS protocols. It is normally not used directly — the module urllib uses it to handle URLs that use HTTP and HTTPS.

So we're down with 4 modules that could allow us to communicate with a RESTful API (between you and me, that's still way too much and should be standardized).

The Github API

Reading the Github API documentation, it appears that to create a new repository, we need to send a POST request to `https://api.github.com/user/repos`, with some input data (the only mandatory input being `name`, a string), and our credentials.

Let's see how to do such a thing with urllib2, httplib2, pycurl and requests.

urllib2

```
import urllib2, urllib

github_url = 'https://api.github.com/user/repos'

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm() # o_Ô seriously?
password_manager.add_password(None, github_url, 'user', '***')

auth = urllib2.HTTPBasicAuthHandler(password_manager) # create an authentication handler
opener = urllib2.build_opener(auth) # create an opener with the authentication handler
urllib2.install_opener(opener) # install the opener...

request = urllib2.Request(github_url, urllib.urlencode({'name':'Test repo', 'description': '
Some test repository'})) # Manual encoding required
handler = urllib2.urlopen(request)

print handler.read()
```

This code is so ugly, it looks like Java. However, once we've instanciated the url, the password manager, the opener, a request is only 2 lines of code. With urllib2, a request without data will be interpreted as a GET request, and a request with data will be intepreted as a POST one. The other HTTP methods (PUT, PATCH, DELETE, OPTIONS, HEAD) are not supported. Plus, the documentation is horrible…

Clearly, urllib2 is not the way to go.

httplib2

The httplib2 module is a comprehensive HTTP client library that handles caching, keep-alive, compression, redirects and many kinds of authentication.

```
import urllib, httplib2

github_url = 'https://api.github.com/user/repos'

h = httplib2.Http(".cache") # WAT?
h.add_credentials("user", "******", "https://api.github.com")

data = urllib.urlencode({"name":"test"})
```

```
resp, content = h.request(github_url, "POST", data)

print content
```

Ok that seems better. The only weird line is `h = httplib2.Http(".cache")`. Why would I instanciate an `HTTP` object? Shouldn't the `HTTP.request` method be available as a function of the `httplib2` module? Seems like a design flaw to me, but that's not a major problem anyway.

The thing is, I lied to you. The previous examples fail miserably, sending back a 401.

Michael Foord explains the mechanism of Basic Authentication in urllib2 in his great blogpost urllib2 - The Missing Manual: HOWTO Fetch Internet Resources with Python

> When authentication is required, the server sends a header (as well as the 401 error code) requesting authentication. This specifies the authentication scheme and a 'realm'. The header looks like : `Www-authenticate: SCHEME realm="REALM"`. The client should then retry the request with the appropriate name and password for the realm included as a header in the request

I inspected the header Github sends me back, and it appears that there is no trace of `Www-authenticate` in it.

```
'cache-control': '',
'connection': 'keep-alive',
'content-length': '37',
'content-type': 'application/json; charset=utf-8',
'date': 'Sun, 01 Jul 2012 06:29:07 GMT',
'server': 'nginx/1.0.13',
'status': '401',
'x-ratelimit-limit': '5000',
'x-ratelimit-remaining': '4999'
```

That's why, with urllib2 and httplib2, all we get is an error and a 401 status code. The credentials are not sent back after getting a 401 from the server. If we wanted to use one of these modules, we'd then have to write some more code that would intercept the 401 and automatically send back the credentials. We certainly do not want to do that…

pycurl

One other option is pycurl, a python binding of the cURL C library, libcurl.

```
import pycurl, json

github_url = "https://api.github.com/user/repos"
user_pwd = "user:*****"
data = json.dumps({"name": "test_repo", "description": "Some test repo"})

c = pycurl.Curl()
c.setopt(pycurl.URL, github_url)
c.setopt(pycurl.USERPWD, user_pwd)
c.setopt(pycurl.POST, 1)
c.setopt(pycurl.POSTFIELDS, data)
c.perform()
```

This time, the repo is created and we get a `201 - Created` status code. Yay! However, we can see that pycurl is very low-level: it took us 4 lines to configure a simple request. These lines would either have to be repeated for each request, or be bundled into a function. We'd then have to implement some `get`, `post`… functions, which basically comes down to re-inventing the wheel.

Thus, pycurl could be an option, but kind of defeats the purpose Python high-levelness.

requests

Finally, let's see how to create a Github repo with requests.

```
import requests, json

github_url = "https://api.github.com/user/repos"
data = json.dumps({'name':'test', 'description':'some test repo'})
r = requests.post(github_url, data, auth=('user', '*****'))

print r.json
```

Wait, is that it? No `CreateManagerWithExtraLargeName()` method call? No manual credential sending? Well, no. requests was designed to be an HTTP high level API, supporting all HTTP methods, SSL encryption, proxies, redirection, caching, etc.

I find it perfect for communicating with RESTful APIs, and clearly recommend it over the previous 3 modules! Have a look at their documentation and examples if you're still not convinced.

Conclusion

requests is the perfect module if you want to repeatedly interact with a RESTful API. It supports pretty much everything you might need in that case, and its documentation is extremely clear. (I'm looking at you, urllib2…)

Notes

I've synthetised this article into a 5 minutes presentation I intend to give at the EuroPython 2012 Lightning Talks session. You can find it on Youtube.

Comments

19 Comments isBullsh.it Login

Sort by Best Share Favorite

Join the discussion…

Kiran Koduru • 2 years ago

" This code is so ugly, it looks like Java. " LOL

42 | • Reply • Share ›

42 ^ | v • Reply • Share ›

BYK • 3 years ago

We have developed Pyresto just to solve this exact problem.

3 ^ | v • Reply • Share ›

rgmz • 3 years ago

It is not without reason that the tag line of requests reads "HTTP for Humans", clearly it has the most friendly API from the modules you compared.

One thing that might make a difference is performance, have you done any benchmarks?

3 ^ | v • Reply • Share ›

> **Balthazar Rouberol** Mod ➦ rgmz • 3 years ago
>
> I did not, but blk's comment makes perfect sense: pycurl is a python binding to libcurl, which is entirely written in C. You can expect good performances from it.
>
> 1 ^ | v • Reply • Share ›
>
> **blk** ➦ rgmz • 3 years ago
>
> For performance go with pycurl
>
> 1 ^ | v • Reply • Share ›

Mohsen Hassani • 2 years ago

WOW...Thanks! Saved me after two days wondering what REST is!

1 ^ | v • Reply • Share ›

Rattler • 3 months ago

Problem with requests if you try using anything more complicated than user password authentication. It will take a cert and key file, but it can not be password protected...

^ | v • Reply • Share ›

your butt • 9 months ago

youtube link broke

^ | v • Reply • Share ›

Kimmo Brunfeldt • 10 months ago

Nap is also an option. It's also a wrapper around requests, but makes it very convenient to call HTTP APIs. https://github.com/kimmobrunfe...

^ | v • Reply • Share ›

Michael Lippert • a year ago
Just wanted to say great article, and very helpful. I think I need to keep an eye on this blog.
• Reply • Share ›

Pierre Thibault • a year ago
Thank you. Really useful.
• Reply • Share ›

Erik • 2 years ago
This article made me laugh of your funny comments. Thanks for a great overview! :)
• Reply • Share ›

Vladimir Vasilyev • 2 years ago
Great article! I love "google -> blog -> code -> done" workflow. =)
• Reply • Share ›

Brian • 3 years ago
It's a bad sign when the example code doesn't run, and then doesn't work when it's fixed...
• Reply • Share ›

blk • 3 years ago
What about urllib3?
• Reply • Share ›

Peter Scott blk • 2 years ago
Capable and well-designed, but pretty low-level. It's actually used as part of the back-end for requests.
• Reply • Share ›

chassing • 3 years ago
how about http://slumber.in/ ?
• Reply • Share ›

Peter Scott chassing • 2 years ago
That's a handy convenience module around requests. It wouldn't make the code substantially simpler in this example, but if you want to write (say) a high-level wrapper around some REST API, it could make your code nicer. Worth looking into, anyway.
• Reply • Share ›

James Casbon • 3 years ago
How about remoteobjects[1]? Also, I did this in tornado[2]

[1] https://github.com/saymedia/re...
[2] http://casbon.me/connecting-to...

• Reply • Share ›

Subscribe Add Disqus to your site Privacy