クイックスタート

それでははじめましょう。 この章では、Requestsの使い方の紹介をします。 ここでは、Requestsが既にインストールされているものとします。 インストールしていない場合は、 [インストールする方法](http://requests-docs-ja.readthedocs.io/en/latest/user/install/#install) の章を見て下さい。

最初に、以下を確認して下さい。:

* Requestsが [インストールされている](http://requests-docs-ja.readthedocs.io/en/latest/user/install/#install)
* Requestsが [最新版のものになっている](http://requests-docs-ja.readthedocs.io/en/latest/community/updates/#updates)

いくつかの簡単なサンプルをやってみましょう。

リクエストの生成

Requestsを使ってリクエストを生成することはとても簡単です。

Requestsモジュールをまずインポートしましょう。:

>>> **import** requests

ではウェブページを取得してみましょう。 今回の例では、GitHubで公開されているタイムラインを取得してみましょう。:

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

現在、 r と呼ばれる **Response** オブジェクトがあります。 このオブジェクトから必要な情報を全て取得することができます。

RequestsのシンプルなAPIは、HTTPリクエストの全てのフォームデータを示していることがわかります。 例えば、以下の例はHTTPのPOSTリクエストをどうやって作成するかを示しています。:

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

いいでしょ? PUT、DELETE、HEAD、OPTIONSなどの他のHTTPリクエストについても本当にとてもシンプルです。:

>>> 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"**)**

これはこれでいいのですが、Requestsができることのほんの少ししか紹介していません。

URLにパラメーターを渡す

しばしばURLのクエリ文字列として、あるデータを追加して送信したい時があるかもしれません。 手動でURLを作成する場合、このデータはクエスション記号の後のURLにキー/バリューのペアで与えて下さい。 例: httpbin.org/get?key=val 。 Requestsは、 params キーワード引数を使って、辞書としてこれらの引数を渡すことができます。 例として、 key1=value1 と key2=value2 を httpbin.org/get に渡したい場合、 以下のコードでできます。

>>> payload = **{**'key1'**:** 'value1'**,** 'key2'**:** 'value2'**}**

>>> r = requests.get**(**"http://httpbin.org/get"**,** params=payload**)**

URLを表示して、URLが正しくエンコードされたか見ることができます。:

>>> **print** r.url

u'http://httpbin.org/get?key2=value2&key1=value1'

レスポンスの内容

サーバーからのレスポンスの内容を見ることができます。 Githubのタイムラインを使って見てみましょう。:

>>> **import** requests

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

>>> r.text

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

Requestsは、サーバーからの内容を自動的にデコードします。 ほとんどのユニコード文字はシームレスにデコードされます。

リクエストを作成した時、RequestsはHTTPヘッダーに基づてレスポンスのエンコーディングについて推測しようとします。 Requestsによって推測されたテキストエンコーディングは、 r.text にアクセスした時に使われます。 Requestsで使われているエンコーディングは調べることができ、 r.encoding プロパティを使って調べたり、変更することができます。:

>>> r.encoding

'utf-8'

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

エンコーディングを変更した場合、 Requestsは r.text を呼び出すたびに r.encodingの新しい値を使います。

Requestsは、必要としているイベントにおいてもカスタムエンコーディングを使います。 オリジナルのエンコーディングを作成したり、 codecs モジュールに登録されているものを使う場合、 単純に r.encoding の値としてコーデックの名前を使うことができ、Requestsはデコードの処理をしてくれます。

バイナリデータのレスポンスの本文

テキスト以外のリクエストに、レスポンスの本文にデータとしてアクセスすることもできます。:

>>> r.content

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

gzip や deflate のようなTransfer-Encodingは自動的にデコードされます。

リクエストによって返されたバイナリデータから画像を作成する例として、以下のようなコードになります。:

>>> **from** PIL **import** Image

>>> **from** StringIO **import** StringIO

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

JSONのレスポンスの本文

JSONデータを扱う場合に、JSONをデコードする機能もあります。:

>>> **import** requests

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

>>> r.json**()**

[{u'repository': {u'open\_issues': 0, u'url': 'https://github.com/...

JSONのデコードに失敗した場合、 r.json は単純に None を返します。

生のレスポンスの本文

サーバーからの生のソケットレスポンスの全てを取得したい稀なケースでは、 r.rawにアクセスできます。 アクセスする場合、リクエストを初期化する際に stream=Trueを設定して下さい。 設定すると以下のようになります。:

>>> 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'

カスタムヘッダー

リクエストにHTTPヘッダーを追加したい場合、 headers パラメーターに dict を渡すだけです。

例えば、前の例のようにコンテントタイプを指定する必要はありません。:

>>> **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**)**

さらに複雑なPOSTリクエスト

一般的にHTMLのフォームのようにエンコードしたデータを送信したい場合、 これをするのは簡単で、 *data* 引数に辞書を渡すだけです。 データの辞書はリクエストを生成する時に自動的にエンコードされます。:

>>> payload = **{**'key1'**:** 'value1'**,** 'key2'**:** 'value2'**}**

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

>>> **print** r.text

{

...

"form": {

"key2": "value2",

"key1": "value1"

},

...

}

エンコードされていないデータを送りたい場合が何度もあると思います。 dict の代わりに string を渡した場合、データは直接送信されます。

例えば、GitHubのAPI v3はJSONエンコードされたPOST/PATCHデータを受け取ります。:

>>> **import** json

>>> url = 'https://api.github.com/some/endpoint'

>>> payload = **{**'some'**:** 'data'**}**

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

マルチパートエンコードされたファイルのPOST

RequestsはMultipartエンコードのファイルをアップロードすることが簡単にできます。:

>>> url = 'http://httpbin.org/post'

>>> files = **{**'file'**:** open**(**'report.xls'**,** 'rb'**)}**

>>> r = requests.post**(**url**,** files=files**)**

>>> r.text

{

...

"files": {

"file": "<censored...binary...data>"

},

...

}

ファイル名を明示的に指定して下さい。:

>>> 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>"

},

...

}

ファイルとして受け取りたい場合に文字列を送信することができます。:

>>> 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"

},

...

}

レスポンスのステータスコード

レスポンスのステータスコードを確認することができます。:

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

>>> r.status\_code

200

Requestsは簡単に参照できるように、組み込みのステータスコードのルックアップオブジェクトがあります。:

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

True

不正なリクエスト(200以外のレスポンス)を作りたい場合は、**Response.raise\_for\_status()** で例外を発生させることができます。:

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

>>> bad\_r.status\_code

404

>>> bad\_r.raise\_for\_status**()**

**Traceback (most recent call last):**

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

**raise** http\_error

requests.exceptions.HTTPError: 404 Client Error

しかし呼び出した時は、 r の status\_code が 200 だったので、 raise\_for\_status()を呼び出します。:

>>> r.raise\_for\_status**()**

None

全て上手くいきました。

レスポンスヘッダー

Pythonの辞書形式で簡単にサーバーのレスポンスヘッダーを見ることができます。:

>>> 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'

}

辞書とはいえ、特別です。辞書はHTTPヘッダーを作成するためだけに作られます。[RFC 2616](http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html) によると、HTTPヘッダーは大文字と小文字を区別しません。

そこで、任意の大文字と小文字を使用してヘッダーにアクセスできます。:

>>> r.headers**[**'Content-Type'**]**

'application/json; charset=utf-8'

>>> r.headers.get**(**'content-type'**)**

'application/json; charset=utf-8'

レスポンスにはないヘッダーを参照すると、値はデフォルトの None になります。:

>>> r.headers**[**'X-Random'**]**

None

クッキー

レスポンスにクッキーが含まれている時、レスポンスに簡単にアクセスして取得することができます。

>>> url = 'http://httpbin.org/cookies/set/requests-is/awesome'

>>> r = requests.get**(**url**)**

>>> r.cookies**[**'requests-is'**]**

'awesome'

サーバーにクッキーを送信するには、 cookies パラメーターを使えます。:

>>> url = 'http://httpbin.org/cookies'

>>> cookies = dict**(**cookies\_are='working'**)**

>>> r = requests.get**(**url**,** cookies=cookies**)**

>>> r.text

'{"cookies": {"cookies\_are": "working"}}'

リダイレクトと履歴

Requestsは、GETやOPTIONSメソッドを使っている時に自動的にリダイレクトを行います。

GitHubは全てのHTTPリクエストをHTTPSにリダイレクトします。 リダイレクト処理をトラッキングするためにレスポンスオブジェクトの history メソッドを使えます。 何が起こるか見てみましょう。:

>>> r = requests.get**(**'http://github.com'**)**

>>> r.url

'https://github.com/'

>>> r.status\_code

200

>>> r.history

[<Response [301]>]

**Response.history** は、 リクエストが完了したときに作られる **Request** オブジェクトがリストとして入っています。 リストはリクエストの古いものから最も新しいものの順に並べ替えられます。

GET、HEAD、OPTIONSを使う場合、 allow\_redirects パラメーターを使うことでリダイレクトの処理を無効にすることができます。:

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

>>> r.status\_code

301

>>> r.history

[]

POST、PUT、PATCHを使う場合、明示的にリダイレクトを有効にすることができます。:

>>> r = requests.post**(**'http://github.com'**,** allow\_redirects=True**)**

>>> r.url

'https://github.com/'

>>> r.history

[<Response [301]>]

Timeouts

timeout パラメーターに秒数を指定すると、指定した秒数の間、Requestsのレスポンスの待機を止めることができます。:

>>> 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)

注意:

timeout は、レスポンスの本文をダウンロードせず接続の処理だけにしか影響しません。

エラーと例外

ネットワークの問題が起こった時(例えば、DNSのエラーやコネクションの切断等)に、 Requestsは **ConnectionError** の例外を発生します。

稀に不正なHTTPレスポンスがあった時に、Requestsは **HTTPError** の例外を発生します。

リクエストがタイムアウトした場合、 **Timeout** の例外を発生します。

リクエストが設定されたリダイレクトの最大数超えた場合、 **TooManyRedirects** の例外を発生します。

全ての例外は、 **requests.exceptions.RequestException** を継承して明示的に発生させます。

danger\_mode オプションにして **HTTPError** の例外をすぐに発生させることや 、safe\_mode オプションで **requests.exceptions.RequestException** でRequestsが捕まえる代表的な例外を取得するためには、 Configuration API Docs を見てください。

準備はできましたか? [advanced](http://requests-docs-ja.readthedocs.io/en/latest/user/advanced/#advanced) の章をチェックして下さい。