# Twitter クライアントを作ろう

hideo54

# 目次

# 第 I 章

# まえがき

## 1　自己紹介

こんにちは、70 回生の@hideo54 です。いつのまにか中 3 になってしまいました。早いものです。

昨年、中 1(記事公開時は中 2) のとき、当時の部誌で「ぱそこんのなかをのぞいてみよう」的な記事を書いたことがあります。PC パーツの説明記事です。今読み返してみれば、完全に黒歴史ですがw当時の僕は、中 1 前期に C をいじっていたのですがなかなか使いこなせず、ちょうど Mac のメモリを換装しようとしたときに、PC パーツに興味を持ち、あのような記事を書いたのでした。

中 2 の 1 年間では、主に Python を勉強していました。@hiromu1996 に Python を勧められて、なんとなく勉強したのが始まりでした。o 昨年の文化祭後に、Apple の次くらいに Twitter が好きな僕は、tweepy というライブラリの存在を知り、Twitter API をいじったり、競技プログラミングをしたりしながら、Python の練習をすることになりました。おかげで、1 年かけて、すらすらと Python が書けるようになりました。

この記事は、今までそうやって学んだことの集大成的な記事にしようと思います。また、これを読んだ人が、Twitter API や Python での GUI の構成に興味を持ってくれたり、学習の参考にしてくれたら、幸いです。

## 2 注意

この記事では、次のことを前提として話をすすめていきます: 初歩的な Python の知識があること

また、この記事を書くにあたり使用した環境は以下の通りです:

| | |
|---|---|
| OS | Ubuntu 13.04 |
| Python | Python 2.7.4 |
| Twitter API | Twitter API 1.1 |
| PyGTK | PyGTK 2.24.0 |

第 2 章から記事は始まります。すこし長い記事ですが、なるべくわかりやすく説明したつもりですので、お付き合いください。

# 第 II 章

# CUI で Twitter API をいじる

## 1　下準備

### 1a　Twitter Developer になろう

Twitter API をいじるためには、Twitter Developer 登録をして、Application の申請をせねばなりません。Twitter Developer ページ [https://dev.twitter.com] には、普通の Twitter アカウントを使ってログインすることができます。某企業みたいに、毎年 Developer 費用を払う必要はありません。

右上のアイコン部分 (Sign In 表示の場合は、選択してサインイン) にカーソルを当てたら出てくる項目のうち、My Applications を選択します。そして、Application Management ページ [https://apps.twitter.com] に入れたら、"Create New App"を押します。"Application details"欄の"Name", "Description", "Website"を適当に埋めて、"Developer Rules of the Road"(英文) を読み、同意したら"Yes, I agree"をチェックした上で、"Create your Twitter application"を押します。これで Application の作成は完了です。

"Permission"タブを選択してください。初期設定では"Read Only"になっていますが、これだとタイムラインを読むことしかできないので、"Read, Write and Access direct messages"を選択して、"Update settings"を押します。これで、ツイートや、DM への操作をすることができるようになります。

### 1b　tweepy をインストールしよう

easy_install が入ってなかったら

```
sudo apt-get install python-setuptools
```

easy_install が入っていたら/入れたら

```
easy_install tweepy
```

## 2 基礎知識

### 2a API

Twitter は、サードパーティ製のアプリ (Twitter 社が公式に作成したものでないアプリ) がユーザーのアカウントを操作することを公認しています。このとき、サードパーティ製アプリが、ユーザーの権限の一部を借りて、ツイートしたり、誰かをフォローしたりということができるようになります。サードパーティ製アプリの開発者が、より簡単にそのようなことをできるように、関数のようなものが提供されています。これが API です。

### 2b OAuth

よくある Web サービスでは、ユーザー名とパスワードを入力することでログインできます。普通のサービスであれば、それでいいですね。しかし、Twitter は、サードパーティ製のアプリが API を用いてユーザーのアカウントを操作することを公認しています。ユーザーがサードパーティ製アプリを使用するときに、サードパーティ製アプリにユーザー名とパスワードを渡して認証することは、非常に危険です。

そこで、OAuth というしくみが使われます。ユーザーは、サードパーティ製アプリを使用したいとき、そのサードパーティ製アプリに、自分のアカウントを用いてツイートなどの操作を行うことを許可します。許可されたサードパーティ製アプリは、ユーザーの情報を扱うための認証鍵を Twitter 社からもらいます。この鍵はユーザーのパスワードとは何ら関係ないため、ユーザーのアカウントの安全性が保証されています。サードパーティ製アプリは、ユーザーのアカウント情報に触れることはできません。

## 3 Test OAuth で Hello, World!

じゃあさっそく Twitter API をいじってみましょう。

Twitter Developer ページ [https://dev.twitter.com] を開いてください。右上のアイコン部分にカーソルを当てたら出てくる項目のうち、My Applications を選択します。(Sign In 表示の場合は、選択してサインイン) Application Management ページ [https://apps.twitter.com] に入れたら、先ほど作成した Application を選択し、詳細ページに移ります。(今後、このページを「Application 詳細ページ」と呼びますね。) "Test OAuth"ボタンを押し、"Consumer key", "Consumer Secret", "Access token", "Access token secret"をそれぞれメモっておきます。

いよいよ、Python の出番です。

---

```
import tweepy
```

```
consumer_key = '********'
consumer_secret = '********'
access_token = '********'
access_token_secret = '********'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)

api.update_status('Hello, World!')
```

これで、"Hello, World!"とつぶやかれたはずです。説明します。

## 3a 説明

```
import tweepy
```

tweepy ライブラリを import しています。

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
```

auth というのは authorization の略で、認証という意味です。Consumer key と Consumer secret という 2 つのクライアント認証鍵を、tweepy ライブラリに用意されている OAuthHandler というメソッドにつっこんでいます。OAuthHandler() というのはちゃんと登録されたクライアントかを確認するメソッドです。

```
auth.set_access_token(access_token, access_token_secret)
```

クライアントを使用するユーザーは、それぞれ Access token, Access token secret という 2 つのユーザー認証鍵を発行されています。token というのは証拠という意味です。ユーザーは、先ほど OAuth の説明をしたときに述べたとおり、単にユーザー名とパスワードをクライアントに渡して認証するわけではありません。クライアントごとに Access token を発行し、それをクライアントに教えることで、ユーザーのパスワードがクライアント作成者にばれることなく安全に連携できるわけです。

この場合の Access token は、Application 詳細ページで"Test OAuth"ボタンを押して

表示されたものを使っています。これは、Application Owner(Application 登録したときのアカウント) の Access token です。set_access_token() メソッドで、クライアントの認証情報にクライアントを使用するユーザーの Access token を結びつけています。

```
api = tweepy.API(auth)
```

API() メソッドは、認証情報を使って、api の使用権限を得るメソッドです。

```
api.update_status('Hello, World!')
```

update_status() メソッドは、引数にとった文字列をつぶやくメソッドです。

今紹介したのは、もっともよく使われる tweepy のメソッドです。

さて、無事 Hello, World!とつぶやくのに成功することができました。では次に行きましょう。

## 4 みんなで Twitter

先ほどのプログラムは、自分しか使えませんね。"Test OAuth"を押して表示させた自分の Access token を使っているからです。そこで、自分以外のアカウントがクライアントを使用できるようなプログラムを書きます。

認証 URL を発行し、そこでユーザーにアプリ連携をしてもらいます。そうするとTwitter 社のサーバーから認証コードが発行されるので、その認証コードをクライアント側で受け取ります (ユーザーに入力をしてもらいます)。そうすることで、ユーザーごとにAccess token を発行します。

```
#coding: utf-8

import tweepy

consumer_key = '********'
consumer_secret = '********'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.secure = True
url = auth.get_authorization_url()
print 'この URL にアクセスして、アプリ連携してください: ' + url
```

```
code = raw_input('認証コード: ')

auth.get_access_token(code)
access_token = auth.access_token.key
access_token_secret = auth.access_token.secret

auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)

api.update_status('Hello, World!')
```

## 4a 説明

```
auth.secure = True
```

2014 年 1 月 14 日以降、Twitter API の仕様が変更され、HTTPS 通信が必須となりました。tweepy モジュールはデフォルトでは HTTP 通信で行われますので、この文を書くことによって SSL 接続に変更します。

```
url = auth.get_authorization_url()
```

get_authorization_url() メソッドは、先ほど述べた認証コードを発行するための URL を生成するメソッドです。

```
print 'このURLにアクセスして、アプリ連携してください: ' + url
```

ユーザーに、認証 URL へのアクセスを促します。webbrowser モジュールなどを用いて、自動的にブラウザが立ち上がるようにしてもいいかもしれませんね。

```
auth.get_access_token(code)
access_token = auth.access_token.key
access_token_secret = auth.access_token.secret
```

get_access_token() メソッドは、先ほど発行した認証 URL にアクセスしてもらったユーザーに入力してもらったワンタイムパスコードを引数に取ることで、Access to-

ken を取得できるメソッドです。auth.access_token.key, auth.access_token.secret は、auth.access_token が持つプロパティで、それぞれユーザーの認証鍵 (Access token, Access token secret) の値を持ちます。

## 4b　ユーザーの Access token を保存しよう

先ほどのプログラムの問題点が分かりますか？はい、先ほどのプログラムのままでは、ユーザーは利用する度に認証コードを要求され続けます。そのような面倒なことにならないように、ユーザーの Access token を保存することによって、ユーザーが一度認証をすませば、あとはすぐに使えるようなプログラムを書いてみましょう。

```
#coding: utf-8

import tweepy

consumer_key = '********'
consumer_secret = '********'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.secure = True

user_id = raw_input('ユーザー名を入力してください: ')

file = open('oauth.txt', 'a+')

url = auth.get_authorization_url()
print 'このURLにアクセスして、アプリ連携してください: ' + url

code = raw_input('認証コード: ')

auth.secure = True
auth.get_access_token(code)
access_token = auth.access_token.key
access_token_secret = auth.access_token.secret
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)
line = user_id + ' ' + access_token + ' ' + access_token_secret + '\n'
```

```
file.write(line)

file.close()
```

このようにして作成された oauth.txt は、次のような構成になります。

[**ユーザー名**] [Access token] [Access token secret]
[**ユーザー名**] [Access token] [Access token secret]
[**ユーザー名**] [Access token] [Access token secret]
...

これで、ユーザーごとの Access token を保存することができました。
この txt ファイルを用いて、クライアントを 1 度使ったことのあるユーザーが認証をスキップできるようなプログラムが書けるはずです。最初にユーザー名を入力してもらい、txt ファイルをそのユーザー名で検索し、見つかれば認証をスキップ、見つからなければ認証をした後 Access token を保存する、というようなプログラムを書きましょう。
完成形はこうなります:

```
#coding: utf-8

import tweepy

consumer_key = '********'
consumer_secret = '********'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.secure = True

user_id = raw_input(' ユーザー名を入力してください: ')

judge = False

file = open('oauth.txt', 'r')

for line in file:
    parts = line.rstrip('\n').split(' ')
    if parts[0] == user_id:
        judge = True
        access_token = parts[1]
```

```
            access_token_secret = parts[2]
            auth.set_access_token(access_token, access_token_secret)
            break

    file.close()
    file = open('oauth.txt', 'a+')

    if judge == False:
        url = auth.get_authorization_url()
        print 'このURLにアクセスして、アプリ連携してください: ' + url

        code = raw_input('認証コード: ')

        auth.secure = True
        auth.get_access_token(code)
        access_token = auth.access_token.key
        access_token_secret = auth.access_token.secret
        auth.set_access_token(access_token, access_token_secret)

        api = tweepy.API(auth)
        line = user_id + ' ' + access_token + ' ' + access_token_secret + '\n'
        file.write(line)

    file.close()

    api = tweepy.API(auth)

    api.update_status('Hello, World!')
```

## 5 他のメソッド

先ほど、つぶやくメソッドである update_status() を紹介しましたが、もちろん、これだけでは Twitter は楽しめません。普段僕達が Twitter を楽しむには、自分が一方的につぶやくだけでは不十分です。タイムライン (TL) を読み込んで、他の人のツイートを表示し、時にはリツイート (RT) したり、お気に入りに登録 (ふぁぼ) したりします。他にも、面白い人を見つけたらフォローしたり、厄介がいたらブロックしたり、自分のプロフィー

ルを変更したりもします。そのような行動についてもそれぞれ、Twitter 社は api を提供してくれていますし、そのほとんどに tweepy メソッドが用意されています。

第 3 章では、一旦 Tweepy の解説は終了して、PyGTK の使い方の解説をします。

# 第 III 章

# GUI で Twitter

## 1　PyGTK 入門

Python で GUI を構成するためのモジュールはたくさんあります。PyGTK を選んだのは、理解が容易であり、かつ、比較的できることが多いからです。(単純でできることが多い分、すこし長くなってしまいますが...。)

以下がチュートリアル、リファレンスです。

| | |
|---|---|
| Tutorial | http://www.pygtk.org/pygtk2tutorial/ |
| Reference | https://developer.gnome.org/pygtk/stable/ |

すべて英語なのでつらい人にはつらいかもしれません。この章で、必要最低限のことを記しておこうと思います。

### 1a　基本的な書き方

PyGTK では、各 GUI 部品 (Widget) を上から順に定義することで Widget を配置します。また、ボタンやテキストボックスなどのユーザーが選択する Widget においては、ユーザーの選択したときに実行される関数内に、ユーザーの選択を反映するコードを書く必要があります。まず、普通に定義したあとに、connect() メソッドを使います。1 つ目の引数に、ユーザーの行動 (シグナル)、2 つ目の引数に、そのシグナルが発生したときに実行するときの関数名をいれます。そして、シグナルが発生したときに実行する関数を定義しておきます。

…まあ、言葉だけで言ってもアレですよね。今から段階を踏んで、具体例をあげながら基本的なことだけ解説します。

### 1b　ウィンドウ

参考 URL　http://www.pygtk.org/docs/pygtk/class-gtkwindow.html

```
#coding: utf-8
import pygtk
import gtk

class App(object):
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_title('Sample Program')
        self.window.show()

    def main(self):
        gtk.main()


if __name__ == '__main__':
    app = App()
    app.main()
```

実行すると、ウィンドウが作られます。

## 1c　ラベル

参考 URL　http://www.pygtk.org/docs/pygtk/class-gtklabel.html

```
#coding: utf-8
import pygtk
import gtk

class App:
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_title('Sample Program')
        self.window.show()
```

```
        self.label = gtk.Label()
        self.label.set_text('This is Label')

        self.window.add(self.label)
        self.label.show()

    def main(self):
        gtk.main()

if __name__ == '__main__':
    app = App()
    app.main()
```

ここでは、1 つのラベルをウィンドウの上に重ねました。しかし、PyGTK では、1 つのウィンドウには 1 つまでの Widget しか載せられません (!!) 2 つ以上の Widget を配置するために、Box というものに n 個の Widget を載せ、その n 個の Box をウィンドウに載せます。(これを packing といいます。)
では、Box を使って、今のプログラムにボタンを足してみましょう。

## 1d ボタン

参考 URL　http://www.pygtk.org/docs/pygtk/class-gtkbutton.html

```
#coding: utf-8
import pygtk
import gtk

class App:
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_title('Sample Program')
        self.window.show()

        self.label = gtk.Label()
        self.label.set_text('This is Label')
```

```
        self.button = gtk.Button()
        self.button.set_label('This is Button')

        self.box = gtk.VBox()
        self.box.add(self.label)
        self.box.add(self.button)

        self.window.add(self.box)
        self.window.show_all()

    def main(self):
        gtk.main()

if __name__ == '__main__':
    app = App()
    app.main()
```

VBox の V は Vertical であり、縦という意味です。縦並びに Widget を整列させています。もう一つ、HBox という box も用意されています。この H は Horizontal であり、横という意味です。横並びに Widget を整列させます。

このボタン、押しても何も起こりません。そこで、押したらなんか動作するものを作りましょう。

```
#coding: utf-8
import pygtk
import gtk
import webbrowser

class App:
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_title('Sample Program')
        self.window.show()

        self.label = gtk.Label()
```

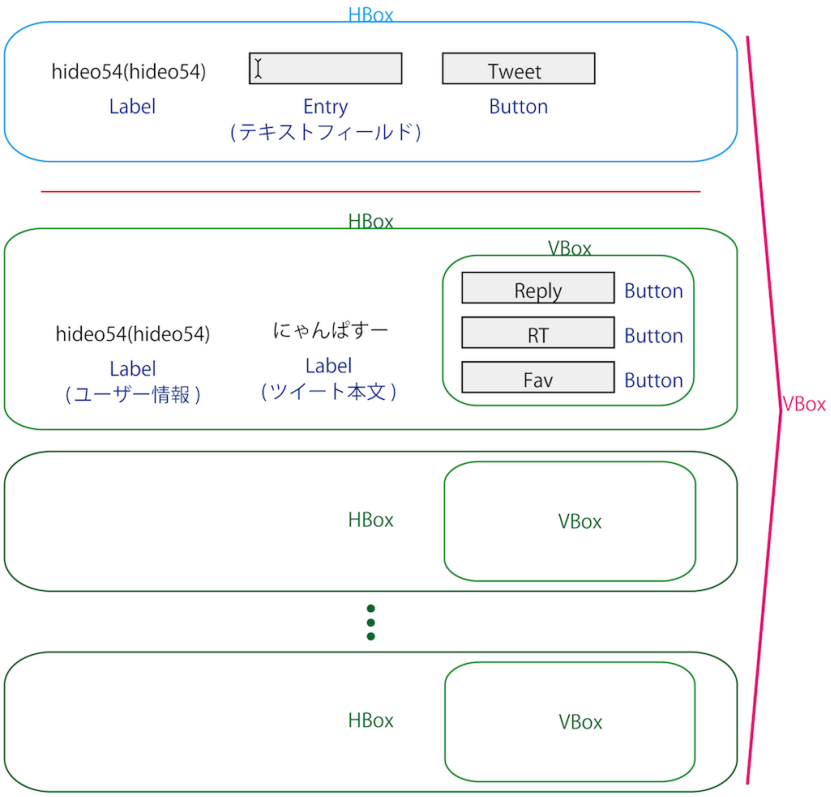
```
        self.label.set_text('This is Label')

        self.button = gtk.Button()
        self.button.set_label('This is Button')
        self.button.connect('clicked', self.open_browser)

        self.box = gtk.HBox()
        self.box.add(self.label)
        self.box.add(self.button)

        self.window.add(self.box)
        self.window.show_all()

    def open_browser(self, widget):
        webbrowser.open('http://hideo54.com')

    def main(self):
        gtk.main()

if __name__ == '__main__':
    app = App()
    app.main()
```

前述した connect() メソッドが出てきましたね。実際にコードを見てみたらわかりますように、第 2 引数に、押したときに呼ばれる関数を指定し、クラス内に関数を作成し、そこに押したときの処理を書いています。第 1 引数に書いてある謎の"clicked"というのは、シグナルという PyGTK の機能の 1 つです。ユーザーが何か行動を起こしたとき、シグナルというものが発生します。ボタンが押されたときに発生するシグナルが、"clicked"です。このコード内では、clicked シグナルが発生したときに、open_browser() 関数を呼ぶ、と指定しています。(conncet() メソッドの名前的には、「第 1 引数のシグナルが出たときに第 2 引数と結びつける」と言った解説のほうが適切なのかもしれませんが、こっちの説明のほうがわかりやすいかなー、と思ってこう説明しました。) 実行すると、ボタンを押したら hideo54.com に飛べるようになったことがわかると思います。

はい、これで PyGTK の基本的な説明は終わりです！はい、これだけわかってたら十分作れます！では、実際に PyGTK で GUI を構成し Twitter クライアントを作っていきましょう。

## 2　設計

どんな Twitter クライアントにするか、設計図を書いてみましょう。ってわけで描いてみた。




文章でやりたいことをまとめると、こんな感じです。

- 最上部には、ツイートの編集らん。
- TL を取得して、5 件ほど Hbox にしてならべる。
- その Hbox の中身は、ツイート元のユーザーとツイートを VBox に入れた Label、Reply ボタンと RT ボタンと fav ボタンを VBox に入れた、の 2 つ。

### 2a　必要なメソッドやプロパティを探す

探し方は 2 章の最後らへんで書いたとおりですね。

### TL 取得

tweepy の公式ドキュメントの API Reference ページ [https://pythonhosted.org/tweepy/html/api.html] を見てみましょう。Timeline methods 項目にそれっぽいものがありますね。

| | |
|---|---|
| api.public_timeline() | (日本語訳) カスタマイズされたアイコンを設定している鍵垢でないユーザーたちの最近 20 のツイートを拾います。 |
| api.home_timeline() | (日本語訳) 認証されたユーザーとそのユーザーがフォローしてる人たちがツイートしたり RT した最近の 20 ツイートを拾います。 |
| api.friends_timeline() | (日本語訳) 認証されたユーザーとそのユーザーがフォローしてる人たちの最近 20 のツイートを拾います。 |

別にどれでもいいけど、もっとも一般的なものは api.home_timeline() ですし、それを使いましょうか。Reference の該当部分を見ると、

Returns: list of class:Status objects" とあります。日本語訳すると、「Status オブジェクトのリストを返します」ってことですね。返されるのはオブジェクトのリストなので、普通に

```
print api.home_timeline()[0]
```

などとしても、

<tweepy.models.Status object at 0x1a56750>

としか返されません。Python に用意されている dir() 関数を使うと、Status オブジェクトの持つプロパティを調べることができます。

```
print dir(api.home_timeline()[0])
```

とすると、

['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__getstate__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_api', 'author', 'contributors', 'coordinates', 'created_at', 'destroy',

```
'entities', 'favorite', 'favorite_count', 'favorited', 'geo', 'id',
'id_str', 'in_reply_to_screen_name', 'in_reply_to_status_id',
'in_reply_to_status_id_str', 'in_reply_to_user_id', 'in_reply_to_user_id_str',
'lang', 'parse', 'parse_list', 'place', 'retweet', 'retweet_count',
'retweeted', 'retweets', 'source', 'source_url', 'text', 'truncated',
'user']
```

が返されます。これが、Status オブジェクトの持つプロパティです。このようにして、オブジェクトの持つプロパティを調べます。たとえば、これらをもとにして、

```
tl = api.home_timeline()
for i in range(len(tel)):
    print tl[i].text
```

とすると、タイムラインのツイートを集めることができます。このようにして、tweepy を活用していきましょう。

### ユーザー名の表示

先ほど示した Status オブジェクトの持つプロパティ一覧の中に、user っていうそれっぽいものがあるので、ためしに表示してみましょう。

```
print api.home_timeline()[0].user
```

すると、

```
<tweepy.models.User object at 0x1a78ab0>
```

が返されました。どうやら、Status オブジェクトが持つ User プロパティは、User オブジェクトを返すようです。そこで、User オブジェクトの持つプロパティを調べる必要がでてきました。

```
print dir(api.home_timeline()[0].user)
```

すると、

```
['__class__', '__delattr__', '__dict__', '__doc__',
'__format__', '__getattribute__', '__getstate__', '__hash__',
'__init__', '__module__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', '_api',
```

'contributors_enabled', 'created_at', 'default_profile', 'default_profile_image', 'description', 'entities', 'favourites_count', 'follow', 'follow_request_sent', 'followers', 'followers_count', 'followers_ids', 'following', 'friends', 'friends_count', 'geo_enabled', 'id', 'id_str', 'is_translation_enabled', 'is_translator', 'lang', 'listed_count', 'lists', 'lists_memberships', 'lists_subscriptions', 'location', 'name', 'notifications', 'parse', 'parse_list', 'profile_background_color', 'profile_background_image_url', 'profile_background_image_url_https', 'profile_background_tile', 'profile_banner_url', 'profile_image_url', 'profile_image_url_https', 'profile_link_color', 'profile_sidebar_border_color', 'profile_sidebar_fill_color', 'profile_text_color', 'profile_use_background_image', 'protected', 'screen_name', 'statuses_count', 'time_zone', 'timeline', 'unfollow', 'url', 'utc_offset', 'verified']

わお、いっぱいあるみたいですね。それっぽいのは…id,id_str,name,screen_name といったものでしょうか。試しに出力してみましょう。

```
tl = api.home_timeline()
print tl[0].user.id
print tl[0].user.id_str
print tl[0].user.name
print tl[0].user.screen_name
```

すると、こう返されます:

932032663
932032663
hideo54@**つらい**
hideo54

よって、id というのは各ユーザーに割り振られているアカウント ID、id_str というのは id の文字列型、name というのは表示名、screen_name というのは@で表されるアカウント名であるとわかります。

### 特定のツイートのリツイート

改めて tweepy の公式ドキュメントの API Reference ページを見てみましょう。Status methods 項目にそれっぽいものがありますね。

| | |
|---|---|
| api.retweet() | (日本語訳) カスタマイズされたアイコンを設定している鍵垢でないユーザーたちの最近 20 のツイートを拾います。 |
| api.retweets() | (日本語訳) あなたのツイートのリツイートの上限 100 個を取得します。 |

今必要なのはどっちですか？はい、単数形のほうですね。ツイートの id というのは何でしょう？ update_status() などといったメソッド名などから察してもわかるように、tweepy は各ツイートのことを status と読んでいます。よって、先ほど Status オブジェクトのプロパティを探るために

```
print dir(api.home_timeline()[0])
```

しましたが、そのときに返された、

['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__getstate__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_api', 'author', 'contributors', 'coordinates', 'created_at', 'destroy', 'entities', 'favorite', 'favorite_count', 'favorited', 'geo', 'id', 'id_str', 'in_reply_to_screen_name', 'in_reply_to_status_id', 'in_reply_to_status_id_str', 'in_reply_to_user_id', 'in_reply_to_user_id_str', 'lang', 'parse', 'parse_list', 'place', 'retweet', 'retweet_count', 'retweeted', 'retweets', 'source', 'source_url', 'text', 'truncated', 'user']

のうちの、id がアヤシイのではないでしょうか。試してみましょう。

```
print api.retweet(api.home_timeline()[0].id)
```

これを実行すると、タイムラインの一番上のツイートがリツイートされたはずです。わーい！

**特定のツイートのふぁぼ**

引き続き tweepy の公式ドキュメントの API Reference ページを見てみましょう。名前からして Favorite 項目ですね。

| | |
|---|---|
| api.favorites() | (日本語訳) 認証されたユーザー (、また引数にユーザーの id をとったときはそのユーザー) の (今までの) ふぁぼ一覧を表示 |
| api.create_favorite() | (日本語訳) 認証されたユーザーのアカウントで、引数にとられた id のツイートをふぁぼる |
| api.destroy_favorite() | (日本語訳) 認証されたユーザーのアカウントで、引数にとられた id のツイートをふぁぼるのをやめる |

今必要なのはどっちですか？はい、create_favorite() のほうですね。ツイートの id の説明はさっきしましたからはぶきますね。

```
print api.retweet(api.home_timeline()[0].id)
```

これを実行すると、タイムラインの一番上のツイートがふぁぼられたはずです。わーい！

### 特定のツイートへの返信

tweepy の公式ドキュメントの API Reference ページ [https://pythonhosted.org/tweepy/html/api.html] 内を"reply"で検索すると、

```
API.update_status(status[, in_reply_to_status_id][, lat][, long][, source][, place_id])
```

これにのみにひっかかります。なんか見覚えあるメソッドですよね。はい、ツイートするときに使ったメソッドです。[] はオプションの (入れても入れなくてもいい) 引数です。in_reply_to_status_id というのは、日本語訳すると「返信するツイートの id」といったところでしょうか。第 1 引数にツイート内容の文字列、第 2 引数に in_reply_to_status_id を指定し返信先のツイートの id をとればいいっぽいですね。じゃあ試しに書いてみましょう。誰にも迷惑をかけないように、自分のツイート [https://twitter.com/hideo54/status/457149806771576832] に返信してみました。

```
api.update_status('.@hideo54 This is test reply', in_reply_to_status_id=457149806771576832)
```

はい、正常に動きましたね！わーい！

**まとめ**

長々と書いてきましたが、要するに、こうやって tweepy を活用していくんだよって話がしたかったのです。このようにしていけば、目的のツールは必ず見つかります。

さて、この節で見つけたものをまとめると、次のようになります。

| | |
|---|---|
| TL のツイート | api.<Status **オブジェクト**>.text |
| ツイートをつぶやいたユーザーの表示名 | api.<Status **オブジェクト**>.user.screen_name |
| ツイートをつぶやいたユーザーのアカウント名 | api.<Status **オブジェクト**>.user.screen_name |
| ツイートをリツイートする | api.retweet(<Status **オブジェクト**>.id) |
| ツイートをふぁぼる | api.create_favorite(<Status **オブジェクト**>.id) |

では、PyGTK と以上の事柄を使って、クライアントを作成していきましょう。

# 3 クライアントの作成

## 3a **トップの** HBox(top_box)

Label<[自分の表示名]([自分のアカウント名])> + TextField + Button<Tweet>

```
#coding: utf-8
import pygtk
import gtk
import tweepy

consumer_key = '********'
consumer_secret = '********'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.secure = True

user_id = raw_input(' ユーザー名を入力してください: ')

judge = False

file = open('oauth.txt', 'r')

for line in file:
    parts = line.rstrip('\n').split(' ')
```

```
    if parts[0] == user_id:
        judge = True
        access_token = parts[1]
        access_token_secret = parts[2]
        auth.set_access_token(access_token, access_token_secret)
        break

file.close()
file = open('oauth.txt', 'a+')

if judge == False:
    try:
        url = auth.get_authorization_url()
        print 'このURLにアクセスして、アプリ連携してください: ' + url
    except tweepy.TweepError:
        print 'エラー'

    code = raw_input('認証コード: ')

    auth.secure = True
    auth.get_access_token(code)
    access_token = auth.access_token.key
    access_token_secret = auth.access_token.secret
    auth.set_access_token(access_token, access_token_secret)

    line = user_id + ' ' + access_token + ' ' + access_token_secret + '\n'
    file.write(line)

file.close()

global api
api = tweepy.API(auth)

class App:
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_title('hideous')
```

```
        self.window.show()

        # 一番上の HBox
        self.top_box = gtk.HBox()

        self.me_label = gtk.Label()
        self.my_name = api.me().name
        self.my_screen_name = api.me().screen_name
        self.me_label.set_text(self.my_name + '(' + self.my_screen_name + ')')

        self.entry = gtk.Entry()
        self.entry.set_max_length(140)

        self.tweet_button = gtk.Button()
        self.tweet_button.set_label('Tweet')
        self.tweet_button.connect('clicked', self.tweet)

        self.top_box.add(self.me_label)
        self.top_box.add(self.entry)
        self.top_box.add(self.tweet_button)

        self.window.add(self.top_box)
        self.window.show_all()


    def tweet(self, widget):
        api.update_status(self.entry.get_text())
        self.entry.set_text('')

    def main(self):
        gtk.main()

if __name__ == '__main__':
    app = App()
    app.main()
```

TextField には、Entry という Widget を使います。この Widget は 1 行のテキストフィールドです。Twitter の字数制限の問題から、set_max_length() メソッドを使って、140 字以上書き込めないようにしています。このテキストフィールドの中の文章は、set_text() メソッドでセットすることができ、get_text() メソッドで取得することができます。このプログラムでは、tweet_button ボタンを押した時に文章を tweet 関数に渡し、つぶやいたあと、set_text() メソッドで、テキストフィールド内を空欄にリセットしています。

## 3b　タイムラインの VBox(tl_box)

Label<[ツイート主の表示名]([ツイート主のアカウント名])> + VBoxButton<Reply> + Button<RT> + Button<fav>

```
#coding: utf-8
import pygtk
import gtk
import tweepy

consumer_key = '********'
consumer_secret = '********'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.secure = True

user_id = raw_input(' ユーザー名を入力してください: ')

judge = False

file = open('oauth.txt', 'r')

for line in file:
    parts = line.rstrip('\n').split(' ')
    if parts[0] == user_id:
        judge = True
        access_token = parts[1]
        access_token_secret = parts[2]
        auth.set_access_token(access_token, access_token_secret)
        break
```

```
file.close()
file = open('oauth.txt', 'a+')

if judge == False:
    try:
        url = auth.get_authorization_url()
        print 'このURLにアクセスして、アプリ連携してください: ' + url
    except tweepy.TweepError:
        print 'エラー'

    code = raw_input('認証コード: ')

    auth.secure = True
    auth.get_access_token(code)
    access_token = auth.access_token.key
    access_token_secret = auth.access_token.secret
    auth.set_access_token(access_token, access_token_secret)

    line = user_id + ' ' + access_token + ' ' + access_token_secret + '\n'
    file.write(line)

file.close()

global api
api = tweepy.API(auth)

global reply_to
reply_to = None

class App:
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_title('hideous')
        self.window.show()

        self.all_box = gtk.VBox()
```

```
# 一番上の HBox
self.me_box = gtk.HBox()

self.me_label = gtk.Label()
self.my_name = api.me().name
self.my_screen_name = api.me().screen_name
self.me_label.set_text(self.my_name + '(' + self.my_screen_name + ')')

self.entry = gtk.Entry()
self.entry.set_max_length(140)

self.tweet_button = gtk.Button()
self.tweet_button.set_label('Tweet')
self.tweet_button.connect('clicked', self.tweet)

self.me_box.add(self.me_label)
self.me_box.add(self.entry)
self.me_box.add(self.tweet_button)

self.all_box.add(self.me_box)

#TL の各ツイートに対する VBox(複数)
self.tl_box = []
self.tl = api.home_timeline(count=5)

self.one_label = []
self.ones_name = []
self.ones_screen_name = []
self.ones_tweet = []
for i in range(5):
    self.one_label.append(gtk.Label())
    self.ones_name.append(self.tl[i].user.name)
    self.ones_screen_name.append(self.tl[i].user.screen_name)
    self.one_label[i].set_text(self.ones_name[i] + '(' + self.ones_screen_name[i] + ')')

    self.ones_tweet.append(gtk.Label())
```

```
            self.ones_tweet[i].set_text(self.tl[i].text)

            self.reply_button = gtk.Button()
            self.reply_button.set_label('Reply')
            self.reply_button.connect('clicked', self.reply, self.tl[i].id)

            self.rt_button = gtk.Button()
            self.rt_button.set_label('RT')
            self.rt_button.connect('clicked', self.retweet, self.tl[i].id)

            self.fav_button = gtk.Button()
            self.fav_button.set_label('Fav')
            self.fav_button.connect('clicked', self.fav, self.tl[i].id)

            self.action_box = gtk.VBox()
            self.action_box.add(self.reply_button)
            self.action_box.add(self.rt_button)
            self.action_box.add(self.fav_button)

            self.tl_box.append(gtk.HBox())
            self.tl_box[i].add(self.one_label[i])
            self.tl_box[i].add(self.ones_tweet[i])
            self.tl_box[i].add(self.action_box)
            self.all_box.add(self.tl_box[i])

        self.window.add(self.all_box)
        self.window.show_all()

    def tweet(self, widget):
        self.tweet = self.entry.get_text()
        if reply_to == None:
            api.update_status(self.tweet)
        elif reply_to.user.screen_name in tweet:
            api.update_status(self.tweet in_reply_to_status_id=reply_to)
        else:
            api.update_status(self.tweet)
        self.entry.set_text('')
```

```
    def reply(self, widget, tweet_id):
        self.entry.set_text('@' + tweet_id.user.screen_name)
        reply_to = tweet_id

    def retweet(self, widget, tweet_id):
        api.retweet(tweet_id)

    def fav(self, widget, tweet_id):
        api.create_favorite(tweet_id)

    def main(self):
        gtk.main()

if __name__ == '__main__':
    app = App()
    app.main()
```

ツイート主の情報のラベル (one_label) と、ツイート (ones_tweet)、Reply ボタン、RT ボタン、Fav ボタンを縦に並べた VBox(action_box) を横並びにした HBox(tl_box) を 5 個作り、配列にしてまとめています。

以上で Twitter クライアントが完成しました。

# 第 IV 章

# あとがき

## 1　最後に

以上で僕の本年度の記事は終わりです。こんなに長い駄文を読破してくださった方はいったいどのくらいいらっしゃるのでしょうか (汗

僕が tweepy をいじるようになった当初、Python も学習途中でまともにいじれないような状態でしたので、ぐぐっても、いまいちよくわからなかったりしました。

この記事を書いている途中で、もし去年の僕みたいな境遇の人が、この記事を見つけて、読んでくれて、理解に大きく役立ったなら、どんなに嬉しいだろうかと考えていました。記事では、去年の僕のような人でもわかるように、なるべくやさしく書きました。この記事を読んだ方で、これでもわかりにくいと思い、困っている方は、いつでも以下の連絡先に連絡してください。

あと、もうひとつ。

実は、今僕がこのあとがきを書いているのは、4/21 です。はい、締め切り超えてしまいました。今年は部内でも全体的に文化祭の準備に関して遅れが出ていて、同級生もみんな記事を書くのをリタイアしてしまいました。そして僕も、部誌をがんばって書き上げようか諦めようか、迷っていました。そんななか、僕を応援してくれた方がいました。そのおかげでがんばって駄文ではありますが無事書き上げることができました。すごく感謝しています。また、先輩方、こんなに書き上げるのが遅くなってしまって申し訳ありませんでした。待ってくれてありがとうございました。これからはもっと精進します。

読んでくださってありがとうございました。

## 2　連絡先

E-Mail　hideo54.pub@gmail.com
Web　hideo54.com
Twitter　@hideo54