第 1 章

「Twitter の Bot を作ってみる」 by object_1037 (73)

1.1　自己紹介と説明

皆さんこんにちは、73 回生 (中 3) の object_1037 です。部誌を書くのは初めてなのですが、温かい目で見守って頂けると幸いです。

ぼくは今年の文化祭の部誌で Twitter の Bot の作り方について書きました。Bot は結構簡単に作れると聞いたので当時暇だった僕は Bot を作ろうと決めたのですが、これが結構難しく、部誌に書こうと決めたわけです。(能力が低い)

1.2　Bot について

突然ですが皆さん Twitter 使ってますか？ Twitter を使ったことがある人ならわかると思いますが、Twitter には Bot と呼ばれるアカウントが数多く存在します。

Bot とは

Bot ってなんやねんと思うかもしれませんが、

> Bot（ボット）は、robot（ロボット）の短縮形・略称で、転じてコンピュータやインターネット関連の自動化プログラムの一種のこと。(Wikipedia より)

とのことです。要は普通のアカウントと違ってプログラムで動いているアカウントのことを Bot とよぶわけです。

Bot の種類

Bot と一概に言ってもたくさん種類があるのですが、ざっと分けると

- 決められた言葉の中からつぶやくもの。名言 Bot とかは大体これ。
- フォロワーからのリプなどに反応してつぶやくもの。占う系の Bot とか。
- 天気予報 Bot など、どこかのサイトから情報を持ってきてつぶやくもの。

このぐらいに分けられます。

僕は今回決められた時間につぶやく時報 Bot と単語をつぶやく Bot, 天気予報 Bot を作りました。

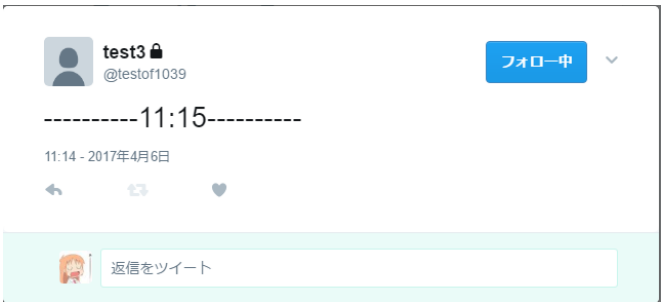

図 1.1 時報 Bot

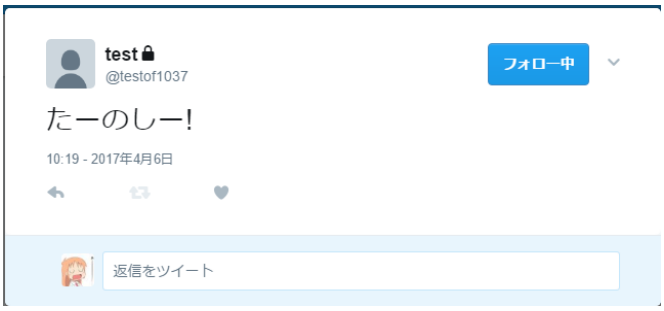

図 1.2 単語をつぶやく Bot

図 1.3 天気予報 Bot

1.3 Bot の作成

それでは実際に Bot を作っていきます。前提環境は

- OS: Windows 10 Home
- Ruby: 2.2.6
- gem: twitter 6.1.0, json 2.0.3, 1.8.1

です。異なるバージョンを使用している場合には一部操作が違う場合があるので注意してください。

作成手順

作成手順は大きく分けて

1. 環境構築
2. Twitter の設定
3. プログラミング
4. 運用

です。それでは作っていきましょう。

環境構築

まず、今回は Ruby を使って Bot を作ったので Ruby のインストールが必要です。Ruby は https://rubyinstaller.org/downloads/ からダウンロードできるのでダウンロード、インストールしておきましょう。[*1]

次に DevKit をインストールします。DevKit は Windows 環境でネイティブな C/C++ 拡張をビルドするためのツールキットです。gem のインストール時に必要になるので、https://rubyinstaller.org/downloads/ からさきほどインストールした Ruby のバージョンに対応した DevKit をダウンロードするようにしましょう。そこでインストール時にこんな画面が出てくるはずです。

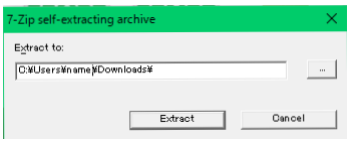

図 1.4 DevKit のインストール

このまま展開すると Downloads フォルダに中身がバラけてしまうので、あらかじめ DevKit のフォルダを作っておいてそこに展開するようにしましょう。

つぎに DevKit の初期設定を行います。コマンドライン（コマンドプロンプト等) を開いてさっき DevKit をインストールしたフォルダに移動してから、

```
$ ruby dk.rb init
```

[*1] Ruby のバージョンは特にこだわりがない限り最新、または一つ前のバージョンがおすすめです。少なくとも 2.0.0 以降のものをインストールするようにしましょう。

と打ち込んでください。つぎにエディターでさっき展開した中にある config.yml というファイルを開きます。するとこんな感じになっているはずです。

List 1.1 config.yml

```
# This configuration file contains the absolute path locations of all
# installed Rubies to be enhanced to work with the DevKit. This config
# file is generated by the 'ruby dk.rb init' step and may be modified
# before running the 'ruby dk.rb install' step. To include any installed
# Rubies that were not automagically discovered, simply add a line below
# the triple hyphens with the absolute path to the Ruby root directory.
#
# Example:
#
# ---
# - C:/ruby19trunk
# - C:/ruby192dev
#
---
- C:\Ruby22-x64
```

ここで一番下の行にあるパスを Ruby をインストールしたフォルダのものに書き換えてください。(C:\Ruby 等)

次にもう一度コマンドプロンプトを開いて

```
$ ruby dk.rb install
```

と打ちます。エラーが出なければ DevKit の設定は完了です。

次は gem をインストールします。コマンドプロンプトを開いて

```
$ gem i twitter json
```

と打ちます。エラーが出なければ環境構築は完了です。

Twitter の設定

次に Twitter 側の設定をしていきます。まず当然ですが Bot を作るには Bot にするためのアカウントが必要なのでアカウントを作ります。

次にそのアカウントでログインしたまま https://apps.twitter.com/ にアクセスして、【Create New App】で新規アプリケーションを作ります。色々聞かれるので適当に埋めましょう。

ちなみに【Name】に入れた名前は TweetDeck 等の他社製クライアントで開くときに【Name】から投稿された、と出ます。なので恥ずかしい名前にしないようにしましょう。(ブーメラン)

図 1.5 こんなかんじ

また、すでに登録してある名前をつけることはできません。今回私は「JAPARIPARK」という名前をつけようとしたのですが、あいにくすでに登録してあったようで、後ろに半角スペースをつけ、「JAPARIPARK 」とすることで解決しました。

また、この名前 (ここでは JAPARIPARK) をクリックすると【Website】にいれたサイトに飛びます。

アプリを作ったら、【Keys and Access Tokens】タブから【Create My Access Token】というところを押して、表示される Consumer Key、Consumer Secret、Access Token、Access Token Secret をそれぞれメモ帳にでもコピペしておきましょう。これで Twitter の設定は完了です。

プログラミング

今回のメインはここです。プログラミングと聞くと難しそうですが、結構かんたんです。

Hello,World!

早速定番の「Hello,World!」とつぶやくプログラムを作ってみましょう。

List 1.2 hello.rb

```
require 'twitter'

client = Twitter::REST::Client.new(
  consumer_key:        "YOUR_CONSUMER_KEY",
  consumer_secret:     "YOUR_CONSUMER_SECRET",
  access_token:        "YOUR_ACCESS_TOKEN",
  access_token_secret: "YOUR_ACCESS_TOKEN_SECRET",
)
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE

client.update "Hello,World!"
```

(プログラム内にある YOUR_CONSUMER_KEY、YOUR_CONSUMER_SECRET、YOUR_ACCESS_TOKEN、YOUR_ACCESS_TOKEN_SECRET にはそれぞれ Twitter の設定のときにのときにコピペしておいた Consumer Key、Consumer Secret、Access Token、Access Token Secret を入れてください。)

これを実行すると「Hello,World!」とつぶやかれるはずです。

図 1.6 Hello,World!

プログラムの説明をすると、まず一行目の

```
require 'twitter'
```

はモジュール (ここでは gem twitter) を呼び出すコードです。

また、3~8 行目の長いのは Twiter のアカウントに OAuth 接続するコードです。ただの変数代入ですから、ここは

```
hoge = Twitter::REST::Client.new(
  consumer_key:        "YOUR_CONSUMER_KEY",
```

```
    consumer_secret:     "YOUR_CONSUMER_SECRET",
    access_token:        "YOUR_ACCESS_TOKEN",
    access_token_secret: "YOUR_ACCESS_TOKEN_SECRET",
  )
```

としておいて、最後のところを

```
hoge.update "Hello,World!"
```

としても動作します。
また、9 行目にある

```
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE
```

これは SSL 証明書を確認しないという意味です。セキュリティ的に危ないですが、**Windows** ではなぜか証明書が更新されていないため、接続するにはこのオプションを付けるか証明書を書き換えるかする必要があります。(Mac OS Sierra ではなくても接続できました。)
そして最後の

```
client.update "Hello,World!"
```

ですが、これは Twitter に投稿するコードです。ほかにもフォローなどいろいろなことができます。[*2]

表 1.1 できること

メソッド名	利用例と意味
.update	client.update "Hello" Hello とつぶやく
.follow	client.follow "Apple" @Apple をフォロー
.unfollow	client.unfollow "Apple" @Apple をフォロー解除
.block	client.block "Apple" @Apple をブロック
.unblock	client.unblock "Apple" @Apple をブロック解除
.user	user = client.user "Apple" @Apple の情報を取得
.id	user.id 上で取得した情報のうちユーザー ID
.screen_name	user.screen_name 上で取得した情報のうちユーザー名
.protected?	user.protected? 上で取得した情報のうち非公開かどうか (true,false)
.tweets_count	user.tweets_count 上で取得した情報のうちツイート数
.followers_count	user.followers_count 上で取得した情報のうちフォロワー数
.friends_coun	user.friends_count 上で取得した情報のうちフォロー数
.home_timelin	client.home_timeline 自分のタイムラインを取得
.user_timelin	client.user_timeline "Apple" @Apple のタイムラインを取得
.mentions_timeline	client.mentions_timeline メンションを取得

コマンドラインからつぶやく

さっきのプログラムを応用することでコマンドライン (コマンドプロンプト等) からつぶやくことができるようになります。いちいちブラウザを開かなくていいので便利です。

[*2] もっと知りたい人は https://github.com/sferik/twitter、https://sites.google.com/site/gyakuhikiruby/home/twitter-gem に載っています。

List 1.3 cmd.rb

```
require 'twitter'
require 'kconv'

client = Twitter::REST::Client.new(
  consumer_key:        "YOUR_CONSUMER_KEY",
  consumer_secret:     "YOUR_CONSUMER_SECRET",
  access_token:        "YOUR_ACCESS_TOKEN",
  access_token_secret: "YOUR_ACCESS_TOKEN_SECRET",
)
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE

tweet = gets.chomp.toutf8
client.update tweet
```

さっきと比べると呼び出しが追加されています。

```
require 'kconv'
```

この kconv というのは文字コードを変更するためのモジュールです。なぜこんなものがいるかというと、Windows の標準文字コードは Shift-JIS なのでコマンドラインから入力した文字列は Shift-JIS でフォーマットされるのに対し Twitter では UTF-8 を使用しているため Twitter 側では文字化けしてしまうからです。(Mac だと標準で UTF-8 を使っているのでこの部分はいりません。)

あと最後らへんに追加されている

```
tweet = gets.chomp.toutf8
```

の `gets.chomp` というのはコマンドラインで受け取った文字列から改行を削除するメソッドで、`.toutf8` と言うのはさっき言った kconv を使って文字コードを UTF-8 に変換するメソッドです。

これを実行してからツイートしたい言葉を入力するとツイートされます。

最後の部分を

```
print "いまどうしてる?>"
tweet = gets.chomp.toutf8
client.update tweet
```

みたいな感じにすると面白いかもしれません。

時報

いよいよ Bot らしい Bot を作っていきます。今回の時報 Bot の仕様は 5 分おきに現在時刻を表示し、かつ昼か夜かわかりやすくするため 6 時から 18 時のあいだと 18 時から 6 時のあいだとでは表示様式を変えるというものです。

List 1.4 chrono5.rb

```
require "twitter"

client = Twitter::REST::Client.new(
  consumer_key:        "YOUR_CONSUMER_KEY",
  consumer_secret:     "YOUR_CONSUMER_SECRET",
  access_token:        "YOUR_ACCESS_TOKEN",
  access_token_secret: "YOUR_ACCESS_TOKEN_SECRET",
)
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE

loop do
  t = Time.now
  hour = t.hour
  min = t.min
  sec = t.sec
  mins = min.to_s
```

```
  if hour < 18 && hour >= 6 && min % 5 == 0 && sec == 0 && mins.length == 1
    client.update "----------#{hour}:0#{min}----------"
    sleep 295
  elsif hour < 18 && hour >= 6 && min % 5 == 0 && sec == 0 && mins.length == 2
    client.update "----------#{hour}:#{min}----------"
    sleep 295
  elsif (hour < 6 || hour >= 18) && min % 5 == 0 && sec == 0 && mins.length == 1
    client.update "■■■■■■■■■■#{hour}:0#{min}■ ■ ■ ■ ■ ■ ■ ■ ■ ■"
    sleep 295
  elsif (hour < 6 || hour >= 18) && min % 5 == 0 && sec == 0 && mins.length == 2
    client.update "■■■■■■■■■■#{hour}:#{min}■ ■ ■ ■ ■ ■ ■ ■ ■ ■"
    sleep 295
  end
end
```

最初と比べて変わったのは loop 内だけです。

loop の最初では変数代入をしています。`Time.now` と言うのは現在時間、`.hour`, `.min`, `.sec` はそれぞれ時、分、秒を取得する Time クラスのメソッドです。

その次は条件分岐です。

```
if hour < 18 && hour >= 6 && min % 5 == 0 && sec == 0 && mins.length == 1
  client.update "----------#{hour}:0#{min}----------"
  sleep 295
elsif hour < 18 && hour >= 6 && min % 5 == 0 && sec == 0 && mins.length == 2
  client.update "----------#{hour}:#{min}----------"
  sleep 295
elsif (hour < 6 || hour >= 18) && min % 5 == 0 && sec == 0 && mins.length == 1
  client.update "■■■■■■■■■■#{hour}:0#{min}■ ■ ■ ■ ■ ■ ■ ■ ■ ■"
  sleep 295
elsif (hour < 6 || hour >= 18) && min % 5 == 0 && sec == 0 && mins.length == 2
  client.update "■■■■■■■■■■#{hour}:#{min}■ ■ ■ ■ ■ ■ ■ ■ ■ ■"
  sleep 295
end
```

1 本目の条件分岐は時間が 6 時から 18 時までの間で分が 5 で割り切れて、一桁 (0 分か 5 分) で秒が 0 のときに傍線と現在時間を表示するということです。一桁と二桁で分けたのは

----------12:5----------

みたいになって気持ち悪いと思ったからです。

また、投稿した後に sleep に入るようにしていますが、これは無限ループによる負荷を減らすために入れています。5 分おきなんだから 300 秒のほうが効率的だと思うかもしれませんが、Twitter の投稿に数秒かかるので 295 秒にしています。ただし動かしてから最初の投稿までの間は sleep がきかず、高負荷がかかるので注意しましょう。

あとの条件分岐もあまり変わりませんね。

これで時報 Bot は完成です。投稿間隔を変えれば n 分おきの時報も作れます。

単語をつぶやく Bot

次は 2 時間おきに決められた単語をつぶやく Bot を作ります。大体の Bot はこんな感じです。

List 1.5 kemono.rb

```
require "twitter"

client = Twitter::REST::Client.new(
  consumer_key:        "YOUR_CONSUMER_KEY",
  consumer_secret:     "YOUR_CONSUMER_SECRET",
  access_token:        "YOUR_ACCESS_TOKEN",
  access_token_secret: "YOUR_ACCESS_TOKEN_SECRET",
)
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE

loop do
```

```
  t = Time.now
  hour = t.hour
  min = t.min
  sec = t.sec
  words = ["わーい!", "たーのしー!", "すっごーい!", "なにあれ?", "なにこれ?",
    "おもしろーい!", "しゃべったあああああああ！ ", "ひどいよー!", "どこここ?", "あーはー!","えっへん!",
    "そんなことないよ!", "あらーいさーん"]
  words.each do |tweet|
    if hour % 2 == 0 && min == 0 && sec == 0
      client.update tweet
      sleep 7195
    end
  end
end
```

配列を作って、その中からツイートする単語を順番に選んでいく感じですね。

今回は words という配列の中に 13 個の単語があり、2 時間ごとにそれを順番に投稿するので 26 時間で一周するようになっています。

ここで注意です。Twitter には投稿制限というものがあり、同一内容のツイートを短時間のうちには連続して投稿できないんですね。制限が解除される時間は、はっきりとはわからない (無責任) のですが、24 時間挟めば確実に引っかからないです。よって単語の候補を増やせばもっと短い間隔で投稿できますし、候補が少なくても間隔を開ければ投稿できます。

今回は each メソッドで順番に取り出していますが、こんな感じでランダムに取り出すこともできます。

```
tweet = words.sample
if hour == 10 && min == 0 && sec == 0
  client.update tweet
  sleep 86395
end
```

ただしランダムにする場合は同じ単語が連続で当たる可能性もあるので上記の理由から投稿間隔を 24 時間以上にしておくのを忘れないようにしましょう。

また、後で詳しく述べますが、さっき作った時報 Bot のように時間の正確性および現在時間の把握などが必要ない今回のような Bot ではプログラム内に投稿の周期を書くより、運用の際に設定したほうが楽です。

天気予報

最後に朝 7 時に大阪の天気を投稿する天気予報 Bot を作ります。天気と言ってもサイトによって予報が違うこともありますが、今回は Livedoor 天気予報の Weather Hacks から天気予報を取ってくることにしました。

List 1.6 weather.rb

```
require "twitter"
require "net/http"
require "uri"
require "json"

client = Twitter::REST::Client.new(
  consumer_key:        "YOUR_CONSUMER_KEY",
  consumer_secret:     "YOUR_CONSUMER_SECRET",
  access_token:        "YOUR_ACCESS_TOKEN",
  access_token_secret: "YOUR_ACCESS_TOKEN_SECRET",
)
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE

loop do
  uri = URI.parse('http://weather.livedoor.com/forecast/webservice/json/v1?city=270000')
  json = Net::HTTP.get(uri)
```

```
    result = JSON.parse(json)
    today_tel = result['forecasts'][0]['telop']
    tomor_tel = result['forecasts'][1]['telop']
    min_tem = result['forecasts'][1]['temperature']['min']['celsius']
    max_tem = result['forecasts'][1]['temperature']['max']['celsius']
    t = Time.now
    hour = t.hour
    min = t.min
    sec = t.sec
    if hour == 7 && min == 0 && sec == 0
      client.update "#{result['title']}\n#{result['link']}\n予報の発表日時： #{result['
publicTime']}\n今日： #{today_tel}\n明日： #{tomor_tel} 最低#{min_tem}度 最高#{max_tem}度"
      sleep 86395
    end
  end
```

まず呼び出しが追加されています。

```
require "net/http"
require "uri"
require "json"
```

いずれも Weather Hacks のサイトから情報を取ってくるためのモジュールです。

また、loop の最初の

```
uri = URI.parse('http://weather.livedoor.com/forecast/webservice/json/v1?city=270000')
```

ですが、この URI から天気情報を取ってくるという意味です。また、URI の最後の `city=270000` の数字に他の値を入れると他の都市の天気情報を持ってこられます。例えば神戸の天気なら `city=280010`、東京なら `city=130010` みたいな感じです。各地域の ID は http://weather.livedoor.com/weather_hacks/webservice に載っています。

あとは JSON で情報を読み込んで、変数に代入して投稿しているだけです。

これでプログラミングは終わり、いよいよ運用に入っていきます。

運用

さっきのプログラミングで一応 Bot そのものは完成です。ところが単純にコマンドプロンプト等で実行していると間違えて終了させてしまったり、またログインしていないと実行されないなどたくさんの問題があります。そこでどうにかして自動で実行させようというわけです。ありがたいことに、Windows にはタスクスケジューラというアプリが標準ではいっていて、それを使うとかんたんに自動実行を設定できます。とはいっても流石に電源が切れていたりするとプログラムを実行できないので、常に電源の入った PC が必要となります。

> ■コラム: 常に電源の入った PC とは
>
> サーバーが理想的ですが、基本的に常に電源が入っていればデスクトップでもノートパソコンでもいいです。充電が切れたり、シャットダウン、再起動、スリープ等しているときに Bot のツイート時刻が被るとその時刻のツイートはされないので注意しましょう。

タスクスケジューラで回す

(注意) ここからは管理者権限でログインしているものとします。

まずタスクスケジューラを起動します。すると右の【操作】というパネルに【タスクの作成...】(【基本タスクの作成...】ではない。) というところがありますね。そこを押して、出てきた画面で【名前】と【説明】を適当に書き、下の方にあるラジオボタンを【ユーザーがログオンしているかどうかにかかわらず実行する】にしておき OK を押します。

次に【トリガー】タブを開き、左下にある【新規】をクリックします。開いたウィンドウで【タスクの開始】を【スタートアップ時】にしてから OK を押します。これで起動したときにプログラムが実行されます。

次に【操作】タブを開き、【新規】をクリックして、【操作】では【プログラムの開始】を選択、【プログラム/スクリプト】は **rubyw.exe** が存在する場所 (C:\Ruby\bin\rubyw.exe 等) を記述します。【引数の追加】はプログラミングのところでかいたプログラムの場所 (C:\Ruby\samples\chrono5.rb 等) を記述して、【開始】はそのプログラムが保存されているフォルダ (C:\Ruby\samples 等) を記述します。

これで OK を押したら Twitter の Bot は完成です。お疲れ様でした。

おまけ

プログラミングのところで時報 Bot のように時間の正確性および現在時間の把握などが必要ない単語をつぶやく Bot、天気予報 Bot などではプログラム内に投稿の周期を書くより、運用の際に設定したほうが楽だ、と言いました。たとえば単語の Bot の場合、

List 1.7 kemono-kai.rb

```
require "twitter"

client = Twitter::REST::Client.new(
  consumer_key:        "YOUR_CONSUMER_KEY",
  consumer_secret:     "YOUR_CONSUMER_SECRET",
  access_token:        "YOUR_ACCESS_TOKEN",
  access_token_secret: "YOUR_ACCESS_TOKEN_SECRET",
)
OpenSSL::SSL::VERIFY_PEER = OpenSSL::SSL::VERIFY_NONE

words = ["わーい!", "たーのしー!", "すっごーい!", "なにあれ?", "なにこれ?",
  "おもしろーい!", "しゃべったあああああ！ ", "ひどいよー!", "どこここ?", "あーはー!","えっへん!",
  "そんなことないよ!", "あらーいさーん"]
client.update words.sample
```

プログラムをこういうふうに書き換えて、さっきの【トリガー】を設定するところで【タスクの開始】を【スケジュールに従う】にして、左にあるラジオボタンを【毎日】にします。

これで一日置きにランダムに単語をつぶやかせることができます。もちろん最初に言ったやり方でやっても何も問題ありません。

1.4 おわりに

これで 2017 年度の部誌「Twitter の Bot を作ってみる」は終わりです。

今回作ったのは本当に最低限の機能しかない Bot です。もっと高機能な Bot も作ることができるのでこの部誌を読んで Bot に興味が湧いてきたという人はぜひチャレンジしてみてください。

ここまで私の拙い文を読んでくださった皆さん、本当にありがとうございました。