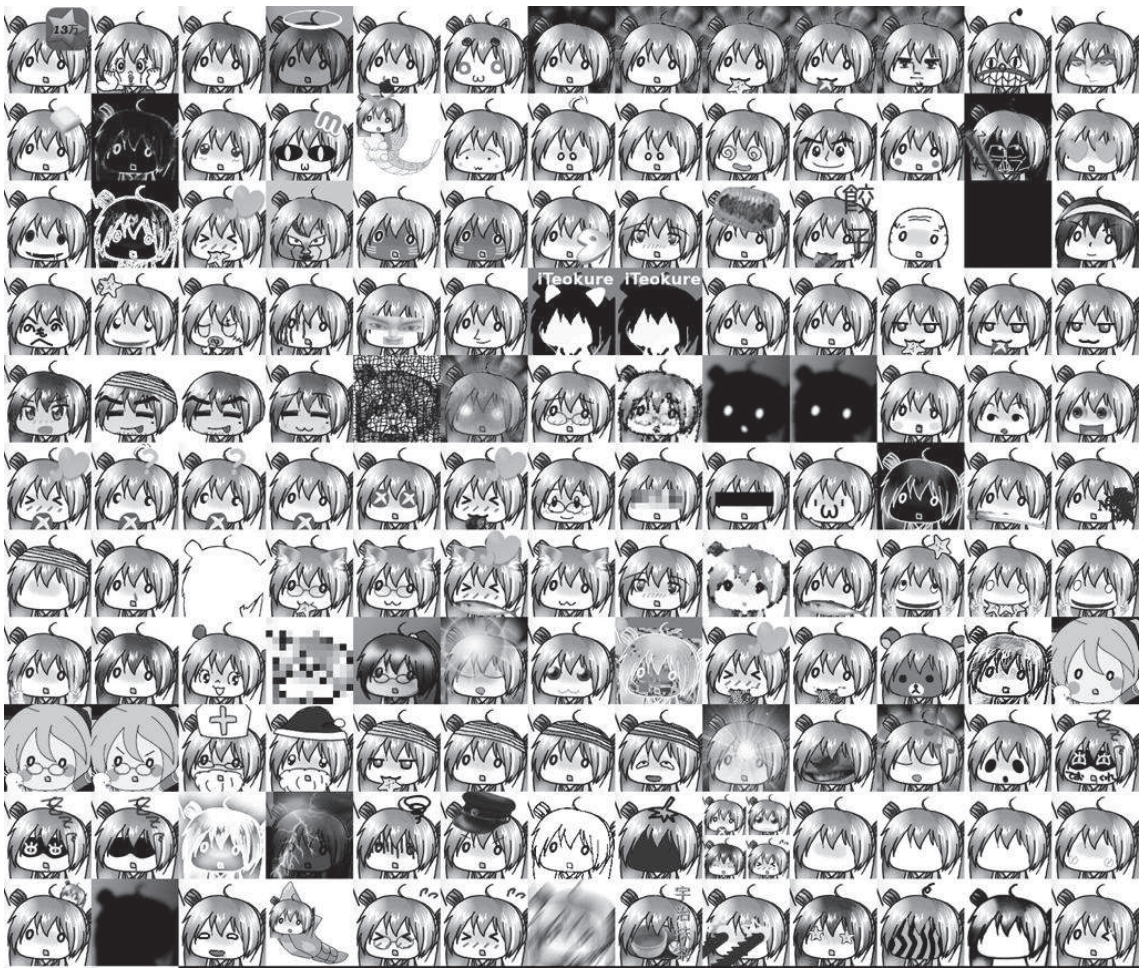


mikutterの薄い本

mikutterの薄い本

vol.1

mikutter の薄い本制作委員会 編

みっくみくな twitter クライアント：mikutter

mikutter インストール方法、mikutter プラグイン制作方法　等

***Rateing:　Teokurated*（ておくれ・ておくれ候補生以外購入・閲覧禁止）**

企業「Twitter クライアントは活用してる？」
↓
学生「はい」
↓
企業「じゃぁ名前書いて」
↓
学生「mikutter」
↓
企業「ﾋﾟｬｱｱｱｱｱｱｱｱｱｱｱｱｱｱｗｗｗｗｗｗ」

@toshi_a , 平成 23 年 11 月 30 日

この「薄い本」の目的

・mikutter の歴史、将来、mikutter への意見、ておくれと mikutter のカンケイ、としぁがアへる漫画載せ、mikutter のておくれなハッテンに資する　　のが本来の目的だったような。

mikutter の薄い本制作委員会　編　「mikutter の薄い本 vol.1」の概要

1.mikutter とは

mikutter とは　みっくみくな twitter クライアントである。そう、その表現でいいだろう。
世の中には星の数ほどの twitter クライアントがあり、その twitter クライアントの一つに過ぎない。では何故その mikutter にこれほど迄傾倒し、薄い本を出してしまうに至ったのか。其れを考えてみよう。

・みっくみくであること
前頁最後の URL を見て頂けば分かるが、どこにみっくみく要素があるのかちっともわからない。faq には**「ミク要素がありません」「愚か者め…知らず知らずのうちに精神が蝕まれていることにも気づかないとは…。」**とあるほど。なるほど、何も分からない。
なんと、実際に使ってみないと分からないみっくみくクライアント。
ふぁぼ(Favorite:お気に入りに登録すること)られたことで通知がでるばかりでなく「ふぁぼっ♪」とミクが喋る。ReTweet、Reply でも同じように
そう、あの初音ミクが、可愛らしく通知してくれるのだ！
初音ミクにより導かれたておくれ達が、「知らず知らずのうちに」よりておくれて、傾倒するほかない状態に陥るのである。

・Linux で使えること
Linux なんか使うのと云うのは御尤もな意見である。実際、文系には殆ど居ない。然し Linux を使っている人であれば twitter クライアントさえ探すことに苦労するほどに無い。インストールについては後述する。
Linux で使えるばかりでなく、いまでは様々なディストリビューション、OS へと移植されている。なるほどユーザーが増える訳だ。
文系に多いと考えられる Windows については次のページにちょこっと書いておく。

・クライアント製作者が面白い
これが最も珍しいことである。 twitter で @toshi_a (以下、としぁさんと呼ぶ)をフォローすれば一目瞭然。彼が面白すぎて涙が出る。mikutter ユーザーは mikutter 制作者であるとしぁさんを弄ることで楽しめる。こんなサーヴィスを提供するとしぁさん、非常にておくれだ。

2.mikutter と私とておくれ

わたしが mikutter に接してしまった原因は知り合いの知り合いの知り合いの知り合いがとしぁさんだったことだ。
間に Gentoo 遣いとかが居たような（汗
当時 twitter クライアントを使っていなかった私だが、としぁさんが面白くて仕方無かった（？）。段々と彼の人柄に魅かれ、何時の間にか彼と夜を共にする仲にハッテン・・・。平成 24 年をとしぁさんと迎えてしまった。居酒屋で。
mikutter ユーザーには独特の雰囲気がある。まず、みな口を揃えて言うのは「ておくれている」。”ておくれ”とはなんなのか。
これについてはとしぁさんの発言をもとに考える必要がある。
以前某人（現在当該アカウントは抹消され他のアカウントに移っている）が、としぁさんに「ておくれ」と云ったことが端緒であるとされる。以降、としぁさんの固有名詞が「ておくれ」となり、また彼の作ったクライアント”mikutter”のユーザーもまた、ておくれであるとされる。辞書をひいておくれ。そこに意味が載っているだろう。
日常的に「あぁ、ておくれだな」と思われる行動をし、tweet をする。ただこの一点なのかもしれない。
mikutter ユーザー以外にもておくれは居るし、mikutter ユーザーでもておくれでない人間も居る。ただ何かしらておくれと惹き合う人が「ておくれ」なのだろう。この冊子も事前に計画を練り十分な”量”と”質”を提供するつもりだった。然し現実には如何だろう。まともな記事を書いたのが 2 名（彼等には頭が上がらない。今度カラダで報酬払います）である。今度冊子を出すような機会があればもっとまともなものに仕上げたいと思う。是非とも次に期待し**ておくれ**。

memo:mikutter を Windows に導入するには

mikutter　windows あたりでググればそれなりに方法を示したものが出てくる筈だが、取り敢えず、(パソコン周りの知識が全くなかった)筆者のやっている方法を簡単にメモ。まわりにこんなことやっている人も居ないらしい。

1. Windows に VMware Player を install
2. Ubuntu の iso イメージを download
3. VMware Player にて Ubuntu を用いて仮想マシンを立てる
4. mikutter を download
5. あとは mikutter の ReadMe を読めばいいと思うよ。

How to Install Mikutter?

@katsyoshi

1　はじめに

この文書では，mikutter[1] をインストールし実行する方法について書いてあります．ここでは，Linux(2.2), FreeBSD(2.3), Mac(2.4) におけるインストール方法について述べます．

2　インストール

インストール方法についてある程度まで実行できるようにします．

2.1　必須アプリケーション

mikutter を動かすためには，以下のアプリケーションが必要となります [2].

- ruby-1.9.2 以上
- x11
- gtk2
- Open SSL
- Subversion

これらのアプリケーションは mikutter をインストールする前にインストールします．mikutter を動かすために必要ではないが，subversion をあげておく．この文書で mikutter は，基本的に trunk 版を推奨する．

Ruby はパッケージからでもインストールできますが，提供されてる Ruby のパッケージが 1.8 系となっているディストリビューションも多いので，ここでは，RVM(Ruby Version Manager) について 2.1.1 で述べます．

2.1.1　RVM:Ruby Version Manager

mikutter をインストールする前に Ruby の実行環境を RVM(Ruby Version Manager)[3] を用いて構築します．Linux の一部ディストリビューション，FreeBSD では必要ありませんが，ここでは，一応説明します．RVM の公式サイトを参考に RVM をインストールし，Ruby の最新版をインストールします．

```
$ bash -s stable < <(curl -s \
https://raw.github.com/wayneeseguin/rvm/master/binscripts/rvm-installer)
$ rvm install 1.9.3 --default
```

RVM をインストールするには，curl, gcc などが必要になりますので RVM の公式サイトを参考にインストールしてください．

2.2 Linux

まず，Linux でのインストール方法について説明します．Linux では，Ubuntu Linux(2.2.1) と Gentoo Linux(2.2.2) について説明します．

2.2.1 Ubuntu Linux

Ubuntu では，README の通りにいけますが，ここでは，RVM を用いてインストールします．そのまえにまず，必要なアプリケーションをインストールします．

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get build-dep ruby
$ sudo apt-get install libgtk2.0-dev
```

必要なアプリケーションをインストールする前に OS を最新の状態にします．つぎに，Ruby をビルドするために必要なアプリケーションをインストールします．そして，libgtk2.0-dev をインストールします．これは，gem から ruby-gtk2 をインストールするために必要です．

RVM をインストールしてください (2.1.1)．RVM をインストール，ruby-1.9.3 のインストールが完了したら，つぎに gem から必要なライブラリをインストールします．

```
$ gem install gtk2
```

これで必要なものがインストールできました．

2.2.2 Gentoo Linux

Gentoo では，すでにパッケージが用意されてますので，コマンドを入力するとインストールできます．

```
$ emerge mikutter
```

でインストール終了です．

2.3 FreeBSD

FreeBSD も Gentoo(2.2.2) と同様に ports が提供されていますので，簡単にインストールできます．

```
$ cd /usr/ports/net-im/mikutter
# make install
```

でインストール終了です．

2.4 Mac

Mac では，Macports か Homebrew を用いて必要なライブラリをインストールします．ここでは，Homebrew と RVM を用いた方法について述べます．

まず，Homebrew から必要なライブラリをインストールします．

```
$ brew install gtk subversion
$ gem install gtk2
```

準備はこれで終わりですが，日本語入力用に MacUIM をインストールする必要 2.4.1 があります．

2.4.1 MacUIM

MacUIM[4] は，Mac の X11 で日本語入力するためのアプリケーションです．MacUIM のインストールは，公式サイトから最新版の dmg をダウンロードします．

公式サイトからダウンロードし，インストールが完了すると，つぎに MacUIM の設定を行ないます．MacUIM を X11 でも使えるよう [5] にまず，X11 の/usr/X11/lib/X11/xinit/xinitrc を~/.xinitrc へコピーします．コピーしたら，つぎの一行を追加します．

```
/Library/Frameworks/UIM.framework/Versions/Current/bin/uim-xim &
```

これは，uim をバックグラウンドで起動します．つぎに，.MacOS ディレクトリを作成し，以下の内容で environment.plist を作成します．

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>LANG</key>
<string>ja_JP.UTF-8</string>
<key>GTK_IM_MODULE</key>
<string>uim</string>
<key>XMODIFIERS</key>
<string>@im=uim</string>
</dict>
</plist>
```

最後に X11 起動時に UIM が起動するように/usr/X11/lib/X11/xinit/xinitrc.d/20-uim.sh を作成します．

```
IM=/Library/Frameworks/UIM.framework/Versions/Current/bin/uim-xim
[ -x $IM ] && $IM &
```

これで Mac での日本語入力が可能となります．

2.5 mikutter のダウンロード

Gentoo Linux や FreeBSD を利用していない場合は，以下の SVN を用いて最新版 (開発版) をダウンロードします．

```
$ svn clone svn://toshia.dip.jp/mikutter/trunc mikutter
```

3　実行

実行は，インストールした方法により変わってきます．Gentoo Linux や FreeBSD では，コマンドラインに mikutter と入力するだけで実行できます．RVM を用いた場合では，

```
$ cd mikutter
$ ruby mikutter.rb
```

と入力することで実行することができます．

4　まとめ

mikutter のインストール方法について，簡単ですが述べてきました．mikutter を動作させるために，ruby などの必要なアプリケーションのインストール，RVM を用いた Ruby のインストールについて説明をし，OS ごとのインストール方法について書きました．

つたない説明で分かりずらいと思ますので，Twitter で@katsyoshi にリプライを投げてくださると反応すると思います．

参考文献

[1] mikutter: http://mikutter.hachune.net

[2] README: https://github.com/katsyoshi/mikutter

[3] RVM: http://beginrescueend.com/

[4] MacUIM: http://code.google.com/p/macuim/

[5] 猫にルビー X11 上の日本語入力環境を構築する: http://catruby.blog83.fc2.com/blog-entry-10.html

Writing mikutter plugin

@toshi_a

1　はじめに

Ruby で mikutter を拡張する方法についてです。

mikutter はサードパーティの「プラグイン」を追加して、あとから機能を追加することができます。

本文章では、四つの事例を取り上げて、プラグインを作るための大まかな方法を説明します。

サンプルコードは、 https://github.com/toshia/writing-mikutter-plugin にこのドキュメントの最新版があるので、そこから取得してください。

1.1　プラグインの動かし方

mikutter のプラグインをインストールするには、 ~/.mikutter/plugin の直下に拡張子 **.rb** で終わるファイルを置くか、プラグイン名のディレクトリを作成し、その中に **プラグイン名.rb** という名前のファイルを設置する方法があります。後者の方法を使うと、git などでプラグインごとにバージョン管理が容易になりますし、プラグイン自体を複数のファイルに分割することができます。

尚、 ~/.mikutter/plugin は、最初は存在しないので、以下のようにして作成します。

```
$ mkdir ~/.mikutter/plugin
```

2　特定の時間にツイートする

このセクションでは、特定の時間に定型文をツイートする、bot のようなプラグインを作成します。

2.1　コード例

```
# -*- coding: utf-8 -*-
# say "よるほー"
# よるほーの例です。あくまで Reserver のデモンストレーション用に作りました。
# なので実戦投入はしないこと。よるほーは自分でタイミングを合わせることにこそ意義があるのです。
```

```
Plugin.create :yoruho do
  def main
    Reserver.new(nextyrhtime){
      say_yoruho(Post.primary_service)
      sleep 1
      main } end

  # 次回のよるほー時間を取得
  def nextyrhtime
    now = Time.new
    result = Time.local(now.year, now.month, now.day, 0, 0)
    while result < now
      result += 86400 end
    result end

  # よるほーとつぶやく
  def say_yoruho(service)
    service.update(:message => 'ておほー') end

  main
end
```

2.2 解説

ほぼすべてのプラグインは、以下のテンプレートから作り始めることになります。

```
Plugin.create :プラグイン名 do
  (プラグインの定義)
end
```

Plugin.create は、プラグインをコアに登録します。また、ブロック内でプラグインを実装するための様々なメソッドを提供しています。この中で変数や関数を定義すれば外のスコープも汚さないので、基本的にはこのブロックの中だけでプラグインは完結させるべきです。

2.2.1 メソッドの定義

ブロックの中は、すぐに Plugin のインスタンスの中で評価されます。def でメソッドを定義して、メソッドの内外で呼ぶことができます。例では main と next_yrh_time と say_yoruho を定義していますね。

2.2.2 文字列を投稿する

- Post オブジェクト

  自動投稿を実現するためには、Post クラスのインスタンスを得ます。Post クラスは、Twitter API のラッパで、ログインしている Twitter アカウントの数だけインスタンスがあります。なので、プラグインがこのクラスを new することはありません。

  これを書いている現在では、mikutter は 1 つのアカウントでしかサインアップできないということになっているので、以下のように書けば、Post クラスを得ることができると覚えておいてください。

  ```
  Post.primary_service
  ```

  mikutter はそのうち複垢対応しますが、そうなったらこのメソッドは、現在アクティブな (選択されている) アカウントの Post を返します。

- update

  Post#update で、ツイートが投稿されます。

  ```
  Post.primary_service.update(:message => "ツイートの本文")
  ```

  ツイートの投稿は非同期で行われるので、処理がここで止まるということはありません。

2.2.3 Reserver 予約実行

特定の時間に処理を実行したいことがあります。今回の場合だと、毎日 0:00:00 に、よるほーとつぶやくコードを実行したいわけです。そういう時は、mikutter が提供している Reserver を使います。

String, Time, Integer のいずれかの値を渡します。

String の場合は Time.parse() された値が使われ、Time を渡すと、その時刻になったら実行します。過去の時刻を渡すとすぐに実行されます。

```
Reserver.new("10:00"){ 朝 10 時に実行 }
```

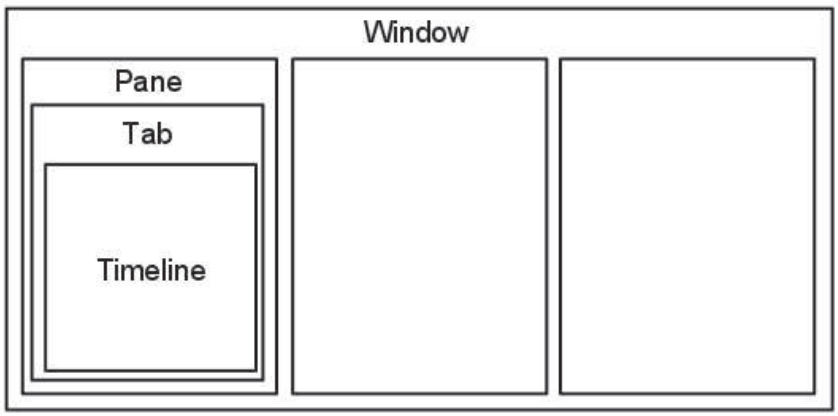
Integer の場合は、その秒数待ってから実行します。

```
Reserver.new(30){ 30 秒後に実行 }
Reserver.new(HYDE){ 156 秒後に実行 }
```

今のところ、Reserver のブロックは、メインスレッド以外のスレッドで実行されます。非同期処理によるバグには気をつけてください。

3　特殊なタイムライン

現在の mikutter の UI は、以下のような構成です。



ウィンドウが一つ有り、その中にユーザは任意の個数のペインを作ります。ペインの中には 1 つ以上のタブがあり、タブの中身はプラグイン次第ですが、基本的にはタイムラインが入っているはずです。

このセクションでは、特殊な抽出条件を持ったタイムラインを作成します。タイムラインを用いたあらゆるプラグインに応用できる知識です。

3.1　コード例

```
# -*- coding: utf-8 -*-
# mikutter についてフォロイーが言及したら、しばらく TL に入れる

Plugin.create :gossip_detector do

  EXPIRE = 300                  # 5 min

  main = Gtk::TimeLine.new()
```

```
  gossip_users = {}

  # TL タブの登録
  onboot do |service|
    Plugin.call(:mui_tab_regist, main, 'Gossip Detector', MUI::Skin.get("timeline.png"))
  end

  # ツイートの受信、振り分け
  onupdate do |service, messages|
    main.add message.select { |m|
      if m.to_s =~ /mikutter|みくった/
        gossip_users[m.user] = Time.now + EXPIRE
        true
      else
        gossip_users.has_key?(m.user) and gossip_users[m.user] > m[:created]
      end
    }
  end

end
```

3.2 解説

このプラグインは、タイムラインをひとつ作って、フォロイーが mikutter について何か言及したら、そのツイートをそのタイムラインにも表示します。これだけだと、現在 mikutter に標準で入っている extract plugin と同じですが、このプラグインはその人のツイートを、マッチしたツイートから 5 分間の間は、無条件にタイムラインに入れます。

最初に mikutter の名前を出して、あとでツイートを小分けにして mikutter について言及しているかもしれないからです。そんなことないかもしれませんが、こんな変なフィルタも書けるんだよというデモなので勘弁して下さい。

3.2.1 イベントの待受

ユーザや Twitter から何かしらの入力があれば、mikutter 内ではイベントが発生します。ほとんどのプラグインは、このイベントを受け取って何かを実行することで、目的を達成します。Plugin.create のブロック内に、以下のように書けば、イベント **event** が発生したときに **処理** が実行されます。その際、引数は **params** にセットされます。

```
onevent do |params|
```

```
  # 処理
end
```

onevent は、 **on_event** と書いても **event** を待ち受けます。イベント名が長いなど、つなげて書くと読みづらい場合は、間にアンダーバーを入れるパターンを使ってもいいかもしれません。引数は、イベントによって異なります。 mikutter プラグインの開発にとってイベントはとても重要なものですが、数が多いので、このあと出てくるぶんについても逐一解説はしません。各イベントについて知りたい場合は、RDoc のイベント・フィルタリファレンス http://mikutter.hachune.net/rdoc/Plugin.html を参照してください。ただし、これが mikutter のすべてのイベントを網羅しているわけではありません。プラグインが新しい種類のイベントを定義することができるからです。

3.2.2 タイムライン

タイムラインをタブに登録するには、タイムラインウィジェットを作成し、それを GUI プラグインに渡してやる必要があります。

- タイムラインウィジェット

  タイムラインウィジェットを作成するのは簡単です。以下のように書きます。

  ```
  timeline = Gtk::TimeLine.new
  ```

  これで、表示するタイムラインの準備が出来ました。

- イベントを発生させる

  タイムラインウィジェットは、確保したあと GUI プラグインに送らないと表示されません。GUI プラグインにウィジェットを送るためには、**mui_tab_regist** イベントを発生させてやる必要があります。

  GUI プラグインは、 **mui_tab_regist** イベントを監視していて、イベントが発生したら引数のタイムラインウィジェットをペインに追加するようになっています。

  ```
  timeline = Gtk::TimeLine.new
  Plugin.call(:mui_tab_regist, timeline, 'Title', MUI::Skin.get("timeline.png"))
  ```

  Plugin.call() は、イベントを発生させるためのメソッドです。第一引数のイベントを発生させ、第二引数以降は、イベントの引数です。このメソッドは即座に返ってきて、実際のイベントの起動は、あとで処理の手が空いたときに行われます。

今回の範囲からは逸脱しますが、イベントを発生させる時、待ち受けるブロックを登録するときには、そのイベントが存在するかどうかは確認しません。つまり、プラグインが好きにイベントを作成することができる、ということです。この仕組を用いて、あなたの作成したプラグイン同士がイベントを使って通信することができますし、本来は別のプラグイン同士が通信する内容を受け取ることも出来ます。今回の **mui_tab_regist** は、GUI プラグインが独自に提供しているイベントです。

  - タブのアイコン

    例の中で使っている MUI::Skin.get() について、一応解説しておきます。
    これは、タイムラインのアイコンのパスを返します。省略可能ですが、このように指定しておけば、ホームタイムラインと同じアイコンが表示されます。スキン機能が実装されたら、現在のスキンでのアイコンの絶対パスを返すようになりますが、現在は実装されていないので、デフォルトスキンの絶対パスを返しています。

- タイムラインにツイートを追加する

作成したタイムラインを表示することは出来ました。次に、ツイートをタイムラインに追加する方法です。これは簡単です。

```
timeline.add(Message...)
```

Gtk::TimeLine のインスタンスに Message のインスタンスか、それが入った配列を渡せば追加されます。

Message については、結構複雑なので後述します。今回の例では、Message#user を使って、ツイートしたユーザを取得しているのと、Message#[] の **:created** (投稿日時) を取得しています。

この情報を元に、mikutter の言及があれば今後 5 分間だけツイートを抽出するフィルタを実現できるわけです。少し実装を変えれば、他のツイッタークライアントではできないような複雑なフィルタを比較的簡単に実装することができるでしょう。

4　スマートなミュート機能

mikutter 標準のミュート機能は、すべてのタイムラインからツイートを非表示にします。このセクションでは、イベントが発生する前にその内容を編集・削除するためのイベントフィルタ機能、またそれを単体で使って、プラグイン間通信を実現する方法について解説します。

4.1 コード例

今回は、特定の単語の入ったツイートを非表示にします。

```
# -*- coding: utf-8 -*-

Plugin.create(:mute_word) do

  exclude_words = ["政治","原発"] # フィルタする単語

  filter_show_filter do |msgs|
    msgs = msgs.select{ |m| not exclude_words.any?{ |word| m.to_s.include?(word) } }  end
    [msgs]
  end

end
```

4.2 解説

4.2.1 イベントフィルタ

- イベントの引数を編集する

  ツイートをプラグインに配送するためにイベントが使われることは前のセクションで触れました。今回は、イベントの発生時にそれを捕まえて、引数を書き換えることができる「イベントフィルタ」を使用します。

  フィルタは、Plugin.create のブロック内で以下のように宣言します。

  ```
  filter_update do |service, msgs|
    msgs = msgs.select{ |m| not m.to_s.include?("政治") }
    [service, msgs]
  end
  ```

  イベントが発生すると、イベントと同じ名前のフィルタが登録されていないか確認して、登録されていたら先にフィルタを実行し、その戻り値を新たに引数リストにします。なので、普通のイベントと同じように引数を受け取り、加工した結果を返すことで、イベントの引数を変更できます。

  この場合は、update イベントで配送されるツイートをすべて確認して、「政治」「原発」というキーワードが含まれたツイートを削除します。三行目が戻り値ですが、ここに注目してください。

引数を配列で返しています。こうやって、加工した値を返してやることで、実際にイベントが発生する時には、該当するツイートが除外されるようになるわけです。

一つ注意しなければいけないのは、戻り値の配列は、必ず引数の数と一致しなければいけないということです。なぜなら、イベントの引数の数は、今のところ常に固定だからです。

- イベントフィルタを使ったプラグイン間の通信

イベントを使えば、別のプラグインに情報を伝えることができるというのは前に紹介したとおりです。

しかし、イベントは受け取りたい方がイベントを待ち受けて、発信されるのを待つ必要があります。実際には受け取り側のリクエストに即座に答えるような通信をしたいことのほうが多いと思われますが、こういった用途にもイベントフィルタを使用します。

まず、イベントフィルタは、イベントが発生していなくても、単独で呼び出すことができます。

```
Plugin.filtering(:update, Post.primary_service, [Instance of Messages...])
```

この戻り値は、[Post, [Message...]] のような、イベントフィルタの戻り値そのままです。

次に、イベントフィルタは、イベントが未定義でも作成できます。

```
filter_plus do |num|
  [num + 1]
end
```

つまり、直接呼び出されることだけを想定したフィルタを勝手に定義してしまえば、リクエスト-レスポンス型の通信が実現できるのです。

  - mikutter から完全に排除するには

    実は、update イベントのフィルタでは、ホームタイムラインから表示されなく成るだけで、他のタイムラインからはフィルタリングできていません。この実装のほうがいい場合もあるでしょうが、mikutter 上にそもそも表示されないようにしたい場合もあると思います。

    そのためには、タイムラインが表示前にフィルタを呼び出していれば、そこで削除することができますね。当然、mikutter はミュートを実現するために、表示する寸前に以下のようなコードでフィルタを呼び出しています。

    ```
    Plugin.filtering(:show_filter, message)
    ```

    つまり、前に書いた update フィルタは以下のように書き換えれば、すべてのタイムラインに影響するようになります。

```
filter_show_filter do |msgs|
  msgs = msgs.select{ |m| not m.to_s.include?("政治") }
  [msgs]
end
```

show_filter は、notify プラグインからも使用されていて、TL に表示されないツイートはポップアップ通知や効果音も鳴らさないようになっています。こんなふうに、他のプラグインが他の用途のためのフィルタを使用したり、コールバックを登録することもできます。イベントとフィルタをうまく使えば、プラグインそれ自身に拡張性を持たせることができます。

5 STOT 形式でコピー

STOT(ShareTwitterOnTumblr) は、ツイートを Tumblr に投稿するためのフォーマットの一つです。様々なアプリケーションで使われていることもあり、この形式でツイートをコピーできると何かと便利です。このセクションでは、

- ユーザが選択して呼び出せる任意の機能 (mikutter コマンド) の追加方法
- ツイートの情報 (Message) とユーザ情報 (User) のオブジェクトの取扱方法

について解説します。

5.1 コード例

```
# -*- coding: utf-8 -*-

Plugin.create(:copy_as_stot) do

  filter_command do |menu|
    menu[:copy_as_stot] = {
      :slug => :copy_as_stot,
      :name => 'STOT 形式でコピー',
      :condition => lambda{ |opt|
        true
      },
      :exec => lambda{ |opt|
        message = opt.message
        screen_name = message.user[:idname]
        Gtk::Clipboard.copy("#{screen_name}: #{message.to_s} [http://twitter.com/#!/
#{screen_name}/status/#{message.id}]")
```

```
      },
      :visible => true,
      :role => :message }
    [menu]
  end

end
```

5.2 解説

5.2.1 STOT について

まずはじめに、STOT の一例を示します。

```
    toshi_a: 会社でふぁぼるのたのしい [http://twitter.com/#!/toshi_a/status/151947400691449856]
```

こんなふうに、ユーザの screen_name と本文、URL(ツイートの ID) の 3 つの要素が含まれています。要は、この 3 つの要素だけ取得して、クリップボードに合成したテキストを送れば終わりです。

5.2.2 mikutter コマンド

この機能の呼び出し方法ですが、

- ツイートを右クリックしたときに出るメニュー
- ツイートを選択した状態で設定されたショートカットキーを押したときに実行される

の 2 つ方法で呼び出せるようにします。mikutter を使っていれば気づくかと思いますが、右クリックメニューで選択できる項目はすべてショートカットキーに割り当てることができます。これは、どちらも登録された「mikutter コマンド」を参照しているからで、ここに一つ機能を追加するようにしておけば、ユーザは右クリックメニューから使ったり、ショートカットキーに割り当てたり、好きに使うことができます。

サンプルコードでいうと、menu という連想配列に入れる内容がポイントです。以下のキーを持ったハッシュを入れます。

1. **slug** コマンドスラッグ。一意なコマンド名。Symbol。
2. **name** 表示名。コンテキストメニューに表示される内容。String。
3. **description** この機能の説明など。省略可能。

4. **icon** アイコンがあれば。今のところ使われない。Gdk::Pixbuf とか String(ファイル名) を指定しましょう。

5. **condition** 実行条件。これの値と===で引数（後述）が比較されます。引数にもよりますが、Proc、Regexp、String などを指定できます。

6. **exec** 実行される本体。:condition と同じ引数を受け取るが、:condition が false になった場合はそもそもこれは呼ばれない。

7. **visible** コンテキストメニューに表示するかどうかのフラグ。false なら表示されない。これは例えば、「ひとつ上のつぶやきを選択」のように、右クリックメニューに表示する必要のないものに使う。

8. **role** コマンドを実行できる環境。たとえば、つぶやきを右クリックすると、ここに:message が指定されたもののうち、:condition が true なものだけが実行又はコンテキストメニューに表示される。指定できるもののバリエーションは後述。

:condition や **:exec** の「引数」というのは、 **:role** に何を設定したかによって変わります。

1. **message** つぶやきにフォーカスがあるとき。以下のような構造体を引数に呼び出されます。

```
Struct.new(
  :event          # Gdk::Event or nil,
  :message        # Message,
  :timeline       # Gtk::TimeLine,
  :miraclepainter # Gdk::MiraclePainter
)
```

2. **messages message** とほぼ同じだが、ツイートの複数選択に対応しています。ツイートの数だけ構造体を含んだ配列が引数として渡されます。

3. **message_select message** の時で、なおかつテキストが選択されているとき。引数も同じ。

4. **timeline** タイムラインで右クリックされたとき。基本的に **message** と同じタイミングだけど、引数は Gtk::TimeLine しか受け取らないという点がちがう。

5. **postbox** つぶやき入力欄。今のところ、ショートカットキーにしか対応していない。引数として Gtk::PostBox を受け取ります。

こういうふうに、わりと簡単に mikutter コマンドを新しく追加できます。mikutter コマンドの話については、mikutter 開発日記の mikutter コマンド http://mikutter.blogspot.com/2011/05/mikutter.html でも触れています。

5.2.3 Message

次に、前のセクションでも出てきた Message について触れます。御存知の通り mikutter 上で一つのツイートを表すオブジェクトです。Message はプラグインから作ることはまずありません。現在のところ、一つのツイートにつき必ず一つだけ Message のインスタンスが生成されることが保証されています。実はキャッシュ機能や他のツイートを探索する機能も持っているのですが、今回は圧倒的に使用頻度が高いと思われる、ツイートの内容を取得するところに焦点を絞って紹介します。

- 用意されているキー

  次のキーで、ツイートの情報の取得はできます。例えば、 **m** に Message のインスタンスが格納されているとしたら、 **m[:id]** で、ツイートの ID が取得できます。

  また、いくつかの情報はインスタンスメソッドでも取得できます。キーを指定する方法なら正直に今持っている値を返しますが、メソッドを呼んだ場合はできるだけ正確に値を返そうとします。

  例えば、 **m[:reply_to]** は、返信先のツイートが読み込まれていない場合はツイート ID を数字で返す可能性がありますが、**m.receive_message** は、そういう場合、データベースのキャッシュや、場合によっては Twitter サーバに API リクエストを発行して返します。

キー	意味	他の方法
id	ツイートの ID(数字)	m.to_i
entity	エンティティ情報 (後述)	m.links, m.entity
message	ツイートの本文	m.body
user	ツイートを投稿したユーザ (User)	m.to_user
receiver	このツイートを宛てられたユーザ (User)	m.receiver
replyto	このツイートの返信先の Message	m.receive_message
retweet	これがリツイートなら元の Message	
source	投稿したクライアント情報	
geo	ジオタグ	
created	投稿時刻	
modified	更新時刻	

  詳しいメソッドの一覧は、http://mikutter.hachune.net/rdoc/Message.html を参照してください。

5.2.4 User

User クラスと Message クラスはともに Retriever のサブクラスで、機能はほとんど同じです。ただし、当然持っている情報のキーに違いがあります。

- 用意されているキー

キー	意味	他の方法
id	ユーザの ID(数字)	u.to_i
idname	スクリーンネーム	u.idname, u.to_s
name	名前	
location	(プロフィールの) 現在地	
detail	プロフィール本文 (bio)	
profile_image_url	アイコン画像の URL	
url	Web ページ（プロフィールで設定された）	
protected	非公開ユーザ (鍵垢) なら真	
followers_count	フォロワーの数	
statuses_count	ツイート数	
friends_count	フォロイー (フォローしている人) の数	

専用メソッドはほとんど用意されていませんが、例えばこのユーザが「自分」かどうかを判定する **is_me?** のような便利なメソッドがいくつかあります。詳細は、http://mikutter.hachune.net/rdoc/User.html を参照してください。

6 あとがき

mikutter は、ほとんどすべての機能がプラグインで実装されていることからも分かる通り、プラグインで柔軟に機能が追加できるようになっています。今回は、4 つの例をとおして、簡単なプラグインを実装する方法を紹介しました。これらを組み合わせれば、ある程度いろんなプラグインを作成できるでしょう。

今まで書いてきた内容も、現在では古くなっている可能性があります。日々進化しつづける mikutter ですが、時にはどうしても後方互換性が取れないような変更をしなければいけない時もあるからです。また、今まで冗長だった書き方も、スマートな方法が提供される場合もあります。最後に、最新の情報へのポインタを幾つか示しておきます。

RDoc http://mikutter.hachune.net/rdoc/index.html を見れば、trunk で、どういったクラスやメソッドが使用できるか確認することができます。完璧ではありませんが、主要なものや最近書かれたものについては、ドキュメントが存在します。

Redmine http://dev.mikutter.hachune.net/ を参照すれば、最近どんな変更が加わったのかを知ることができます。

mikutter 開発日記 http://mikutter.blogspot.com/ は、ある程度まとまった情報を取得するのに便利です。

最後に、「Writing mikutter plugin」の最新版は、https://github.com/toshia/writing-mikutter-plugin に公開されています。最新バージョンに対応していますし、冒頭にも書いたとおり、サンプルコードを取得することも出来ます。

あとがき
話をしよう、あれは今から 36 万・・・いや 4 箇月前だったか。
まぁいい。私にとってはつい昨日の出来事だが、君たちにとっては多分…今日の出来事だ。
彼には 72 通りの名前があるから、なんて呼べばいいのか。
確か最初に会ったときは、…ておくれ。そうあいつは最初からておくれていた。
私の言うとおりにしておけばな、まぁいいやつだったよ

なんか技術書みたいになったから mikutter ユーザーのみなさんはよろこんでくださるかしら。

ほんじゃまた。twitter かどこかでおあいしましょう。

奥付

発行日： 2012 年 1 月 15 日　初版(pdf 版)
発　行：mikutter の薄い本制作委員会
発行者：@brsywe 西端の放送局内喫茶室長
連絡先： brsywe@hotmail.co.jp

ご意見、ご感想はお気軽にどうぞ。

Price：時価

広告

http://mikutter.hachune.net/