mikutterの薄い本　vol.7

手遅センセーション！

イラスト

@soramame_bscl	白/その他	表紙/ブロマイド
@shijin_cmpb	黒/白/青/緑/赤/その他	表紙のちっこいの/缶バッジ

前書きにかえて、d250g2 にみる Twitter と mikutter

@brsywe

1.　この記事の経緯

本当はTwitpic廃止について書こうと思っていたんですが、この記事を書き始める前にTwitpic が救われてしまったので原稿が消え失せました。

> Twitter は、Twitpic の持っている書類を指差して、続きを促した。
> 最後の紙だと思っていたが、少し小さな紙がもう一枚あった。
> 「Twitter さん、こ、これは…！？」
> 「結婚しよう。」

――mikutter 開発日記
http://mikutter.blogspot.jp/2014/10/mikutter-307.html

2.　Twitpic 騒動

省略する。

3.　d250g2 とは

mikutterユーザーにはおなじみのd250g2。これはなんなのか。

> □　Twitpic 終了に伴う作業
>
> - http://twitpic.com/d250g2 が見れなくなったので、http://d250g2.com/d250g2.jpg を表示するように仕様変更

――mikutter 開発日記、同上。

事の発端は @whywaita　による http://twitpic.com/d250g2　というポストである。「現在の僕の心境です」という言葉とともに森林火災の画像をつけたものだ。~~ヒまっちゅうプロジェクトが炎上している~~

@toshi_a に目をつけられ、あるいはTwitter ユーザーにウケ、d250g2 という語だけで炎上を表すようになっただけである。

https://twitter.com/Akkiesoft/status/409603157153435648
@Akkiesoft の悪乗りにより 2013 年 12 月、d250g2.com を取得している。d250g2.com はただ単に先の画像が貼り付けられているだけである。

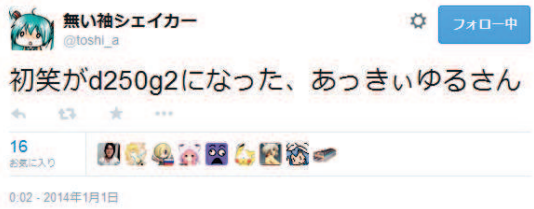
4.　d250g2 のその後

その直後、12 月 31 日のコミックマーケット 85　最終日に次の写真が持ち込まれた。

コミックマーケット 85 といえば vol.5『としぁくんちのておくれ事情』を頒布した回

である。

このとき、彼が持ち込んだ d250g2 写真が 3 箇所のサークル[1]に掲示され、「炎上」したのである。d250g2.com はただ単に森林火災の画像が貼り付けられているだけである。その直後に@toshi_a を悲劇が襲う。



コミケから持ち帰った d250g2 写真をとしぁけて直後、3 人で見て笑う羽目となった。

5. Twitpic と mikutter

Twitpic がサービスの提供をやめると宣言した直後にリリースされた mikutter 3.0.7 に於いて異例ともいえる d250g2.com 対応となっている。これは@toshi_a が d250g2 をいたく気に入っており、Twitpic サービス終了による影響が大きいと判断したためだと考えられる。

その数日後 Twitpic はサービス終了を撤回し、1 項で引用した@toshi_a のホモ小説が現実のものとなったのである。

6. まとめ

今後も d250g2 の活躍を注意深く見守っていきたい。

7. 本題

これは mikutter の薄い本 vol.7『手遅センセーション！』の前書きである。薄い本の中身について簡単に説明しよう。

@tsutsuii「謎マシンと謎 OS でておくれる mikutter」。mikutter と OSC でもよく見かける謎マシン、謎 OS との馴れ初めを描いている。8 頁という超大作。

@moguno「mikutter スキン変更のご提案」。mikutter をよりｵｻﾚに。これでスタバドヤリングもしやくなりますね。

@toshi_a「よりよい mikutter ハックのために」。mikutter3.2 をリリースする前に書かれた mikutter3.2 記事である。わたくしも現時点で 3.2 がリリースされているか知るすべもない。コミケでこの薄い本が出ていくときに 3.2 が出ているかどうかも分からない。

@ soramame_bscl。イラストの提供。表紙絵。@shijin_cmpb イラストの提供。その他の口絵など。マタニティキーホルダーとクリスマス缶バッジも頒布されているはず。

さすがに巻数を重ねて　い　ろ　い　ろとつらい感じが否めない。新規の寄稿者の開拓に努めねばなるまい。

おっともうこんな時間だ。そろそろ寝なければいけないのでここらへんで。また、OSC で会いましょう。　　（了）

[1] こくだランド、らぼちっく；げーと、mikutter の薄い本制作委員会。

謎マシンと謎 OS でておくれる mikutter

@tsutsuii

1. はじめに

mikutter の薄い本 vol.2 に「mikutter と NetBSD のておくれな関係」という記事を寄稿しておくれてしまってからもう 2 年半もの年月が過ぎてしまいました。

当時の記事では、Linux 用クライアントといいつつも「Linux ではないが Linux っぽい OS」のことも考慮してくれている@toshi_a さんに感謝しつつ、私を含む NetBSD 界隈のメンバーが全国各地のオープンソースカンファレンス（OSC）ブースで謎なマシンで mikutter をデモしたり mikutter シールを配ったりしてしまうようになった背景を説明させていただきました。

一方、当該記事の中では「Windows や Mac OS は mikutter のサポート対象ではない」という趣旨のことを書いてしまいましたが、2014 年現在では Windows や Mac OS で当たり前のように mikutter を使っているユーザーが一定数存在します。また、我々 NetBSD 関係者以外にもいわゆる「謎マシン」および「謎デバイス」で mikutter を動かすことに喜びを見出してしまうユーザーが発生しているようです。

本記事では、2014 年末で 5 周年を迎える mikutter が各種の謎 OS・謎マシンで動いている状況を紹介し、それぞれの事例から「なぜ mikutter なのか」という考察を行ってみたいと思います。

2.「Linux とか」

この「Linux とか」という表現については薄い本 vol.2 の記事でも紹介しましたが、vol.2 の記事を執筆した年の 2012 年 3 月にリリースされた mikutter 0.1 に与えられたキャッチフレーズは次のようなものでした。

> **mikutter は**
> **あらゆる Linux とかに対する**
> **最高のソリューションです**

また、上記 mikutter 0.1 の一つ前のバージョンである mikutter 0.0.4 が「開発版」から「不安定版」に昇格した 2011 年 11 月当時に mikutter ホームページ上に表示されていたキャッチフレーズは次のようなものでした。

図 1. mikutter の薄い本 vol.1 裏表紙広告より

この「Linux とか」という表現ですが、私のように古くからインターネットを使っているおっさん層にとっては「UNIX 系 OS」と言ってしまえば済むのに、と思ってしまいがちです。しかし、「Twitter ユーザー」というターゲット層を考えると、「UNIX」という単語自体を知らない層も多数含まれるであろうというのは妥当な判断です。さらに、「Linux」と限定してしまうと、今度はおっさんを含むニッチな領域のユーザーからマサカリが飛んでくることが容易に想像できます。

この「Linux とか」という文言は、ホームページトップに表示するキャッチフレーズに対して各ユーザー層から予想される反応を十分に考慮した、優れた広告表現といえるでしょう。

一方で、「Linux とか」とあえて限定されていたということは、いわゆる「UNIX 系」以外の OS、特に世間の多数派である Windows や Mac OS では解決の難しい問題が存在したということにほかなりません。次項以降では、各 OS で mikutter を動かそうと努力したユーザーの事例を紹介し、実際にどのような問題があったのかを考察してみることにしましょう。

3. mikutter on Windows

Windows で mikutter を動かす方法といえば、前回の mikutter の薄い本 vol.6.0 に掲載された、もぐのさん(@moguno)による「mikutter は Windows でも動きます！マジで。」の記事に詳細が紹介されています。Windows で mikutter で動かすにあたって対処された問題についてはこのもぐのさんの記事に詳細まで書かれていますので、まずは薄い本のバックナンバーをホームページ(※1)等から入手してください。

※1 http://kohinata.sumomo.ne.jp/mikutter.html

図 2. mikutter 3.1 on Windows 7

この記事を書くにあたり私も実際に試して見ましたが、gem のダウンロードやシステムメッセージフォントなどちょっとした問題はあるものの、フリーソフトを zip からインストールしたことがある人ならばそれほど困難なく Windows で mikutter が体験できると思います。

図 3. @toshi_a さん見解ツイート

@toshi_a さんはこのような見解を述べられていますが、mikutter を Windows 上で動かすという試みは案外古くから行われており、最も古い記事は mikutter の開発者の一員でもあるふぇのまーさん（@Phenomer）が mikutter Redmine に書かれた記事(※2)があります。

※2 http://dev.mikutter.hachune.net/boards/2/topics/3

また、mikutter の薄い本 vol.2 にも、上記ふぇのまーさんの記事をベースにした@catina013 さんの記事が掲載されています。これらの記事を読むと、Windows だからといって特殊な手順が必要というというわけではなく、手順としてはあくまでも Ruby と関連 gem のインストールのみです。黒魔術が必要になっているのは、文字コードや改行コードのような既知の OS 固有の問題と、Ruby 本体や GUI としての GTK 関連の Windows 用実装の問題に対する回避策がほとんどであることがわかります。

Ruby も GTK も、本来は Windows を含む各種の OS で動作することを目指して開発されています。しかし、いかんせんそれらを使った Windows 用アプリケーションが少なく、それがテスト不足によるプラットフォーム固有バグといった形で現れているのが現状と言えるでしょう。これは mikutter の実装がどうだという話ではなく、絶対的なアプリの数が少ないのが問題であるということです。つまり、これらの Ruby および GTK 的にマイナーな「変な OS」においても果敢に mikutter というアプリをトライしていくことで、mikutter 以外の開発者も同時に幸せになれる、という構図であるとも言えます。

図 4. ほた (@lindwurm)さんの感想ツイート

余談ですが、mikutter は ruby-GNOME2（Ruby の GTK2 gem）の開発者の間でも主要アプリケーションと認識されているようで(というか、薄い本 vol.6.0 に寄稿されている@cosmo__さんのように mikutter がきっかけで ruby-GNOME2 のコミッタになってしまったという方もいますが)、もぐのさんの記事のベースになっているかずきさん(@kazuki_kaihatu) の「Windows で mikutter を

動かしてみる」の記事中で触れられていた IME 関連のトラブルについて、GTK2 gem の開発者の方から修正リリースの報告メンションが飛んでいたりします。

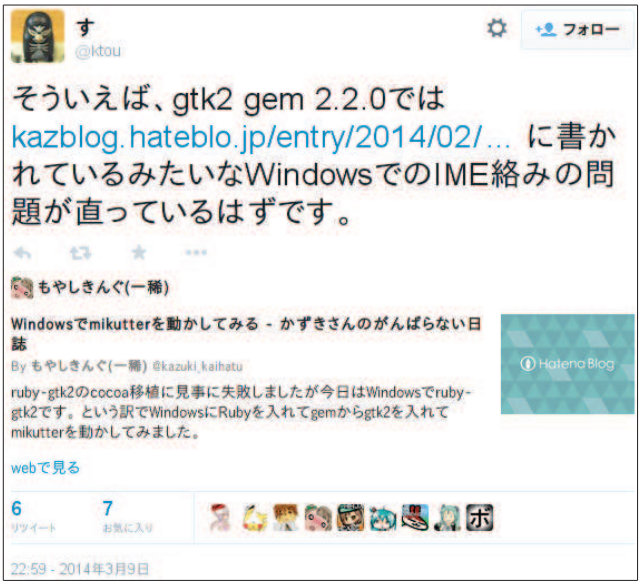


図 5. gtk2 gem 開発者 @ktou さんのツイート

また、ruby-GNOME2 gem が正式に Windows 環境をサポートしたのは ruby-GNOME2 2.0.0 からですが、このとき ruby-GNOME2 が 1.2.6 から 2.0.0 にメジャーバージョンアップして Windows サポートを謳ったのは、mikutter を Windows でトライしていたユーザーからの不具合報告が一つの要因という説[要出典]もあります。

4. mikutter on Mac OS X

「mikutter を Mac で動かす」という課題も、過去に多くの人が挑み、ある人は成功し、ある人は敗北し、また一時は勝利した人も Mac OS のバージョンアップにより再び敗れていった、という歴史が積み重ねられているようです。これをよく表しているのが mikutter の FAQ ページに過去掲載されていた以下のエントリです。

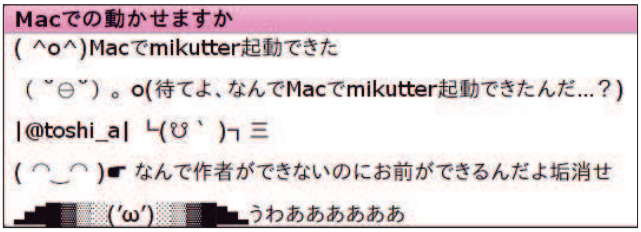


図 6. 過去の FAQ「Mac での動かせますか」(原文ママ)

筆者は Mac を所持しておらず Mac OS についてもほとんど使用したことがないのですが、そもそも Mac OS X は BSD 系 OS の流れを取り込んでいる上に UNIX の正規認定(?)を得ている OS でもあり、前項の Windows と比べれば「Linux とか」により近い存在ではないのかという先入観がありました。

しかし、実際に Mac OS X で mikutter を動かした報告を見ると、Mac OS X には Ruby 自体は標準で入っているものの、GUI 的(つまり GTK2 的)には Windows 以上に困難があることがうかがえました。

Mac OS X へのインストールをトライした記事は古くは mikutter の薄い本 vol.1 に掲載されたかつよしさん(@katsyoshi) さんの記事もありますが、これは mikutter も Mac OS もちょっと古いバージョンです。

現在のバージョンでも通用する手順としては、あっきぃさん(@Akkiesoft) が「mikutter Advent Calendar 2013 Day2」の記事として書かれた Mac OS 10.9 Mavericks へのインストール記事(※3)、そして前述のもぐのさん(@moguno)が書かれた「Mavericks で動いてた mikutter を Yosemite でも動くようにする方法」の記事(※4)が参考になります。

※3 http://akkiesoft.hatenablog.jp/entry/20131202/1385969580
※4 http://moguno.hatenablog.jp/entry/2014/11/23/095157

実際に mikutter が動作したときにはなかなかの感動があるようで、もぐのさんは別のブログ記事(※5)で以下のような感想を書かれています。

> **わっふー！無事マイ Mac で mikutter が動作するようになったぜ。作者にもなし得なかったことを成した背徳感。たまんねえ！！**

※5 http://moguno.hatenablog.jp/entry/2013/01/14/011627

なお、これらの記事のおかげか、現在では作者である @toshi_a さんも無事 Mac OS X で mikutter を動かすことができるようになったらしく、先の FAQ エントリも現在では以下のように更新されています。

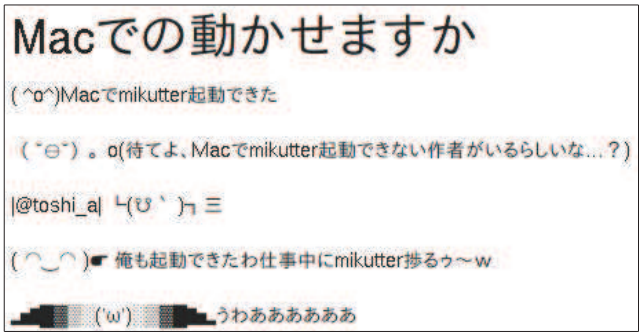


図 7. 現在の FAQ「Mac での動かせますか」(原文ママ)

これらの記事から、mikutter on Mac OS X における問題は以下の 3 点であることがわかります。

① GTK2 GUI 表示（現状 XQuartz が必要）
② GTK2 での日本語入力（現状 MacUIM が必要）
③環境変数（MacOS 的には無いことになってる？）

③の環境変数については OS 実装そのものの設計にかかわる内容であり解決手段はちょっと見えませんが、①②の GTK 関連については MacOS ネイティブな GTK 実装が可能なのかという話かと思います。Windows の場合とは違い、これら Mac OS X 固有の問題に関する Twitter 上のメンションは今のところ見当たらないようですが、各プロジェクトの今後の動きを見守りたいところです。「mikutter に GTK2 以外の GUI を実装する」という壮大な計画も不可能ではありませんが……。

5. mikutter on スマートフォン

OS というくくりからはちょっと外れますが、「スマートフォン上で mikutter を動かす」というネタも過去から何度か報告が挙がっています。

古くは mikutter の薄い本 vol.2 に掲載された @EiM_GTPE_さんの「スマートフォンでお手軽 mikutter」の記事があります。これは、スマホ側に Teamviewer というリモートコントロールアプリを入れ、パソコン（Linux, Windows, Mac OS）側で起動した mikutter をスマホ側に表示させて操作するというものです。記事には「半分ネタです」と書かれてはいますが、root 不要、OS インストールも不要でスマホ上で動く mikutter を手軽に試せる、iOS もサポート、という点でお勧めかと思います。

次にネット上で報告がされたものとして、Nexus を始めとする、いわゆるグローバルモデルの Android タブレットの OS を Linux に入れ替えて（あるいはループバックデバイスを用いて Android 上で Linux カーネルを動かして）、その上で mikutter を動かすというものです。代表的なものとして、先ほど Windows で mikutter の項でもお名前を挙げたかずきさん(@kazuki_kaihatu) のブログ記事「Nexus7(2013)で mikutter を動かしてみた」（※6）があります。

※6 http://kazlog.kazukioishi.net/archives/95

これらの記事のインストールを実践するには、Android の root を取ってさらに Arch や Ubuntu などの OS をインストールする必要がありややハードルが高い上に、mikutter 自身はキーボードとマウス操作用に設計されているためタッチパネルでは各種操作がしづらいという問題があったりします。が、過去に各種謎マシンで mikutter を動かしていた筆者としては
「とにもかくにも mikutter を動かしてみたい！」
という純粋な(?)思いの発露はよく理解できるところです。

そんな中、2014 年の 10 月に「Android アプリ“GIMP Inkscape”上で mikutter を動かす」という話が話題になりました。

図 8. 芝生(@shibafu528)さんの動作報告ツイート

これは、この Android アプリ“GIMP Inkscape”自体が「non root でアプリとして Debian Linux をインストールして、その上で GIMP や Inkscape」を動かすというもので、原理上は先の root 化した端末でのループバックデバイスでの Linux インストールと大きくは変わらないかと思います（筆者不勉強のため詳細が間違っていたらすいません）。

この Android での mikutter インストールについては mikutter Wiki のページ（※7）によりシンプルなアプリの“Debian noroot”を使用した詳細手順が解説されていますので、対応している機種をお持ちの方はトライしてみるのもよいかと思います。

※7 http://yuzuki.hachune.net/wiki/MikutterInstallBattle/Android

この mikutter インストール手順は root 化不要で Google ストアアプリからインストール可能、かつその後は apt-get のコマンドのみで mikutter がインストールできるという点で注目を集め、作者の@toshi_a さんもておくれたユーザーによる mikutter 起動報告までの速さに驚きつつも自身の Nexus5 でトライされたようです。

図 9. @toshi_a さんの感想ツイート

6. mikutter on NetBSD on 謎マシン

今までの項で挙げた OS とは時間が前後しますが、mikutter の薄い本 vol.2 および vol.4 の筆者(@tsutsuii)の記事で紹介しているとおり、私を含む一部の NetBSD ユーザーの間では 2011 年頃から「NetBSD が動作するマシンで mikutter を動かす」という活動(?)を行っており、その成果を全国のオープンソースカンファレンス(OSC)等のイベントでのブースでマシン展示という形でデモしています。

NetBSD には「CPU や新旧を問わず様々なマシンをサポートしている」という特徴があります。その NetBSD を「Ruby と GTK2 さえ動けば起動する」という性格を持つ mikutter と組み合わせることで、**「あらゆるマシンで mikutter を動作させる」**という**「手段(OS)のためには目的(アプリ)を選ばない」**という活動と展示が現在も続いています。

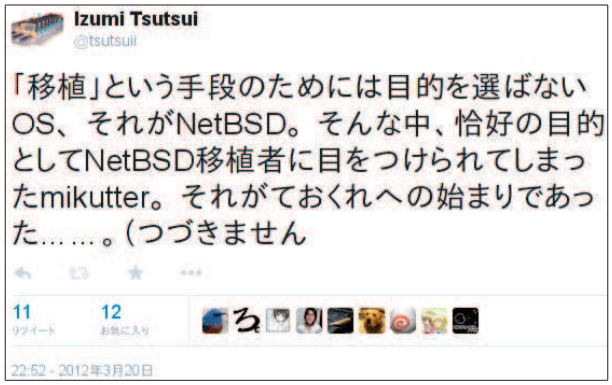


図 10. ておくれた関係

当初筆者が mikutter を動かすターゲットとして選んだのは今は亡き(?)ウィルコムのスマートフォン(??)である W-ZERO3 アドエスでした。当時の mikutter の GUI には 65536 色モードの動作に問題があり、やみくもにデバッグして@toshi_a さんに報告したのが@toshi_a さんとておくれる関わるようになったきっかけです。

その後も Dreamcast、携帯端末 Zaurus、SPARC や Alpha のワークステーション、最近ではラズベリーパイ等のあらゆる NetBSD が動くマシンで mikutter を動かしていますが、NetBSD という OS 固有部分あるいは各マシンに固有な部分に対応するために mikutter を修正しなければならなかったことはほとんどありません。むしろ Ruby 本体や GTK2、それに伴って必要になる glib2 やその他のライブラリ周辺で NetBSD あるいは CPU 関連のバグが見つかることが多く、mikutter がバグ出し活動の原動力となっているというのが現状です。

冒頭の Windows や Mac OS X とは違い「Linux とか」の範疇に含まれる NetBSD ですが、OS や CPU に依存しないどころか「Linux とか」の範疇をも超えて動いてしまう mikutter だからこそ、「ひたすらビルドして正しく動くかどうか」という OS 品質をテストできるわけです。さらには「動いた結果でそのまま展示デモできる」という利点もあり、この活動は当面続くことでしょう。

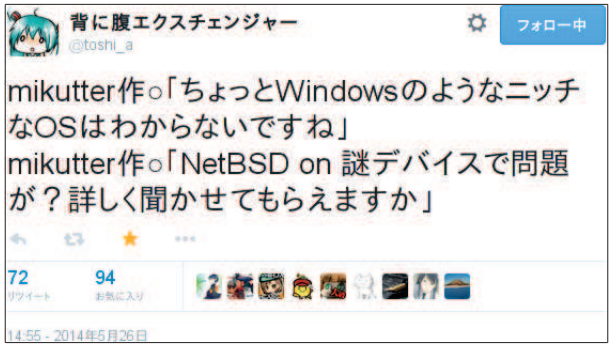


図 11. @toshi_a さんによる NetBSD サポート状況

7. ターミナルモード mikutter on 謎マシン

1 年前の mikutter の薄い本 vol.5 の筆者の記事でも少しだけ触れましたが、NetBSD が動く謎マシン上の mikutter の一つとして、約 23 年前の 1991 年に発売された OMRON 製ワークステーション「LUNA」で動く mikutter、というデモを今年（2014 年）のオープンソースカンファレンス京都の NetBSD ブースで行いました。

図 12 LUNA-II での mikutter デモ@OSC 2014 京都

薄い本の印刷だと見づらいとは思いますが、この mikutter の画面は mikutter といいながらも通常の GTK ウインドウでの mikutter とは異なる見た目をしています。しかし、このデモ画面も mikutter の一つの姿であるのです。

前世紀生まれの LUNA は、CPU は MC68040 25MHz（メガヘルツです）、メモリは 64MB（ギガではありません）という、現在のマシンの 1/100 未満のスペックであり、いまどきのアプリである Ruby を X サーバーとともに動かすことは事実上困難です。それ以前に、X サーバーは GTK が最低限として要求する 65536 色をサポートしていないなど、本来の mikutter の GUI 表示を行うことは不可能です。

一方、Windows および Mac OS の項で説明したとおり、mikutter は Ruby と GTK2 があれば動作します。しかし、mikutter は内部的にはすべての機能がプラグインとして構成されており、「twitter.com とやりとりしてツイートを取得する部分」と「ツイートを GUI 画面上に表示する部分」はそれぞれ独立したプラグインとして実装されています。

Mac OS X の項では「GTK が X サーバーではなく Mac OS ネイティブの描画に対応すれば……」という趣旨のことを書きましたが、それは GTK 担当部分の機能の話でした。その GTK での表示処理以前に「mikutter のツイート表示部分を GTK GUI 以外のものに差し替えてしまえ」という発想がこの「ターミナルモード mikutter」の考え方です。この発想は mikutter の薄い本 vol.4 に掲載された@_osa_k さんの「mikutterd のある生活」で詳しく紹介されていますので、ぜひそちらのバックナンバーもお読みください。

この LUNA-II でのターミナルモード mikutter デモは以下のソフトウェアによって実現されています。

① NetBSD/luna68k のフレームバッファドライバ
② フレームバッファで日本語表示を行う mlterm-fb
③ mikutter のターミナルモードプラグイン mikutterm
④ ③から fork した sixel アイコン対応版 mikutterm
⑤ 画像を sixel 形式テキストに変換する libsixel ツール

聞きなれないソフト名もあるかと思いますが、
① は筆者がメンテ担当である NetBSD 機種固有部分
② は @arakiken さんによりメンテされている端末ソフト
③ は @syusui_s さんによる mikutter プラグイン
④ は @arakiken さんによる③の改造版
⑤ は @kefir_さんによる画像 sixel 処理ライブラリ
（sixel：端末で画像表示をするエスケープシーケンス的プロトコル）
と、①以外は筆者以外の方によって開発されたソフトで、それを組み合わせることによってデモが実現されています。つまり、
(1)mikutter 本体は TL に表示するツイートを取得
(2)mikutterm がそれらのツイートをテキスト表示
(3)mikutterm 改造部がアイコンや添付画像を取得
(4)libsixel が画像を sixel フォーマットでテキストに変換
(5)mlterm-fb がテキストおよび sixel 画像を表示
というわけで、(1)のタイムライン取得で mikutter が本来の仕様で動いていて、それ以外の表示部分は各ソフトが分担して描画しているわけです。ですから、異なる外観ではあったとしても、立派な「mikutter デモ」と言えるでしょう。

実際にこのターミナルモード mikutter with sixel デモが動作している状態では、mikutter 本体の ruby が約 50MB 程度、mlterm-fb が 20MB 程度を使用していました。メモリ 64MB の LUNA-II ではスワップ動作を含めてもかなり限界に近いメモリ使用量で、起動からツイート表示開始まで 10 分以上かかっていたり TL 取得中に User Stream が切れて度々止まったりと、実際の Twitter クライアントとしての実用性は皆無という感じではあります。しかし、「こんなマシンでも mikutter が動きます！」というデモとしては来場者の方に十分アピールできていたのではないかと思います。

8. mikutter の設計思想

前項の「ターミナルモード mikutter」のポイントは、「ツイート取得と画面表示が独立した構造になっている」という部分でした。では、この構成は「たまたま mikutter の構造がそうなっていた」ということなのでしょうか？

答えは否、です。mikutter は開発当初からこのような構成や使い方（＝サービスおよび UI 部分その他を独立して組み合わせ可能）を考慮して設計されていたのです。

2012 年と 2014 年と 2 回にわたり OSC 京都で行われた@toshi_a さんによる mikutter セミナーにおいて、繰り返し挙げられた以下のキーワードがポイントです。

「mikutter は環境！」

2012 年の OSC 京都でこの言葉を聞いてもピンと来なかった方も多いと思います。かく言う筆者もその一人でした。しかし、2013 年末の mikutter の薄い本 vol.5 の @toshi_a さんの記事、そして 2014 年のセミナーを聞くにいたり、そこで@toshi_a さんが mikutter の開発における真の目的としていたものはなんだったのか、そして「環境」とは何を意図していたのかをようやく理解することができました。

薄い本 vol.5 の記事で@toshi_a さんは「mikutter は PC を監視する bot からスタートした」と書かれています。つまり、何かのイベントを監視し、それを別のイベントとしてどこかに通知する、という機能がスタートで、その通知先として最初に選択されたのが Twitter だった、というわけです。この起源を考えれば、イベント元のツイート取得と、出力先の TL 表示が独立しているという構造も当然の結果であると言えるでしょう。

さらに、2014 年の OSC の mikutter セミナーでは以下の「現状」「理想」と題した 2 枚のスライドがありました。

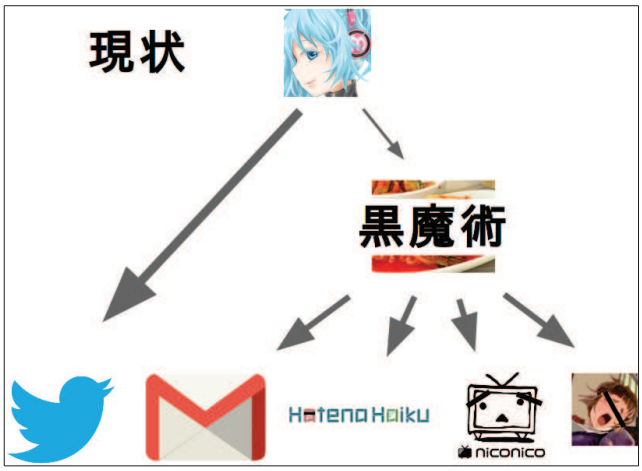


図 13. mikutter「現状」スライド（現状=3.0 以前と思います）

「現状」のスライドは、本来 Twitter クライアントである mikutter に対し、そおくれたユーザーが黒魔術的プラグインを追加することによって Twitter 以外のサービス（Google カレンダー、はてな、ニコ動、大破艦これ等々）を mikutter 上で実現していたことを示しています。

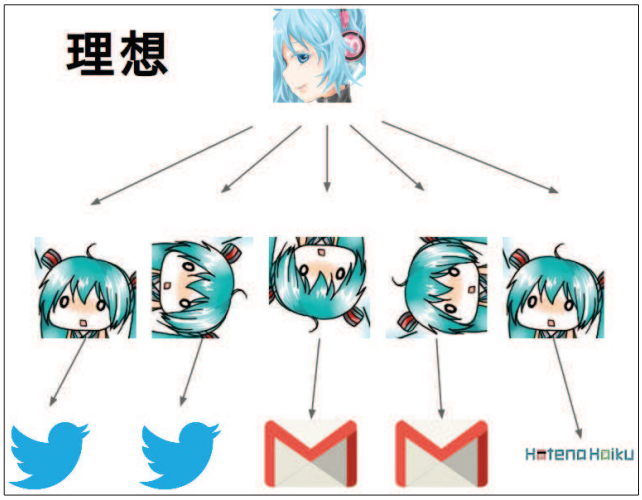


図 14. mikutter「理想」スライド（3.2 以降も含むと思います）

この「理想」のスライドは、mikutter 開発の原点であった「あらゆるイベント」のサポートの形として「マルチサービス」が導入されていることを示しています。「マルチサービス」には各サービスに対する「複数アカウント」の概念も含まれており、「複数のサービスとして複数の Twitter アカウントを処理する」「別のサービスの個別アカウントして Google サービスを処理する」（他のサービスも同様）といった具合に、あらゆるネット上の

イベントを監視して通知する機能を持つこの形態が理想形というわけです。この「理想」こそ、まさに

「mikutter は環境！」

という言葉を体現していると言えるでしょう。

mikutter の薄い本 vol.4 の記事「mikutter はなぜ人をておくらせるのか」において、mikutter の持つ魅力の一つとして「プラグインによる拡張性」というものを挙げました。確かに再ビルドやプルリク等することなく自分の欲しい機能を自分で追加でき、かつ個別に配布も可能というプラグイン構成は mikutter の大きな魅力です。が、ここまでの説明にあるとおり、プラグイン構成は目的ではなく、あくまでも前述の「理想」を実現するために着々と構築された構造だったのです（勝手に断定）。

単なるプラグイン構造であればしくみを用意することは簡単かもしれません。しかし、プログラムの構成や見通しが悪ければ「足したい機能を足したいところに足せない」といった問題が発生し、事実上プラグインの追加が不可能といったことも起こるはずです。

実際の mikutter のプラグインの現状を見ると、その数もさることながら、作者も想定していなかったような機能を実現するプラグインが多数登場しています。このような展開を引き起こしたのは mikutter の開発当初からの設計思想の結果であると言えるでしょう。

9. mikutter の可能性とこれから

前項のような「理想」の一方で、@toshi_a さんは薄い本 vol.5 の記事で以下のような文を書かれていました。

> *“mikutter の進化は常に綱渡りでした。*
> *決して順風満帆ではなく、可能な限り*
> *できることを諦めて、墜落しないように*
> *前に進むことだけを考えていたのです。”*

> *“最初の頃の mikutter は*
> *「全てのことができるように」*
> *という目標を掲げていたにもかかわらず、*
> *驚くほど何もできませんでした。”*

「理想を抱いて溺死しろ」という某ゲーム主人公の有名な(?)セリフとは対照的に、「理想を掲げ、でも実現のために諦めるところは諦め、それでもいつか理想を実現する」という@toshi_a さんの思いが読み取れます。この姿勢を個人のプロジェクトで 5 年もの間続けてきたというのは、とても尊敬に値することだと思うのです。
（理想ばかりで何も実現できていない自分を省みて反省）

筆者は、mikutter 開発に対して常に「自分のために」を標榜していた@toshi_a さんがなぜ mikutter 3.0 でマルチアカウントを導入したのだろう、ということを当初疑問に思っていました。@toshi_a さん自身は複数アカウントをあまり必要としていない上に「複アカふぁぼ爆撃」の発生という危険性もはらんでいたからです。しかし、今年の mikutter セミナーを聞いて「マルチアカウントはマルチサービスという理想に向けての第一歩である」ということをようやく理解することができたのでした。

階層別の見通しのよい mikutter の設計は NetBSD の移植性を重視する思想と共感できる部分です。一方で、妥協すべきところは妥協して実装とリリースを優先する姿勢は私たち NetBSD 関係者に大いに欠けているところでした。これら@toshi_a さんの mikutter に対する取り組み方は大いに見習いたいと思っていました。

私たち NetBSD 関係者のみならず、あらゆる階層のユーザーを~~ておくれさせ~~共感させ、マルチサービスについてもすでに機能拡張プラグインが登場しているというのは、mikutter の「楽しむための素材」としてのポテンシャルの高さを表しています。ユーザーのみなさんも「mikutter は環境」という言葉の本当の意味に着目してもらえれば、と思います。

10. 終わりに

謎 OS と謎マシンの話から始まったのにもかかわらずなぜか最後は一人で勝手に@toshi_a さんと mikutter をヨイショするだけの記事になってしまいました。が、普段から思っていることを書きたいように書くのが薄い本ということで今回もご容赦ください。

昨年の薄い本 vol.5 の記事では「これからは Twitter 本体とともに mikutter は成熟期に入っていく」というようなことを書いてしまいましたが、今後は Twitter という枠をも超えて成長していく mikutter が期待できそうです。2015 年も今まで以上に~~ておくれて~~楽しくわいわいやっている mikutter 界隈をながめていければいいなと思いつつ、締切をすっかり超過してしまったこの記事を締めたいと思います。
（室長さん、どうもすいませんでした！）

mikutterスキン変更機能のご提案

もぐの（@moguno）

1. はじめに

ちす！もぐのです。今回はmikutterへの搭載が検討されている「スキン切り替え機能（仮称）」について書こうと思います。

なお、この記事の内容は2014年12月3日現在の「mikutterやること」のIssue722「機能追加：ユーザスキンの適用」を元に記載しています。そのため、この機能が搭載された暁には仕様がゴロッと変わってたりするかも知れません。

http://dev.mikutter.hachune.net/issues/722

2. プロローグ

2014年10月12日。観測史上初となるmikuter用のアイコンセット「miq」がリリースされました。

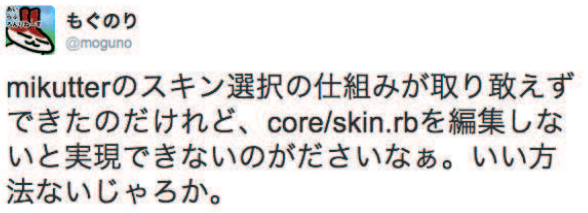
http://ray34g.deviantart.com/art/Miq-487987082
れいずろーあ✿(@rayzlore)さん

miqの圧倒的なオサレ感に完全に魅了された私もぐの。早速アイコン画像を変更するためのプラグインに着手しました。

・・・しかし、このプラグイン開発プロジェクトは見事に頓挫。

mikutterのプラグインの読み込み順序はシステム→サードパーティなので、サードパーティプラグインであるアイコン変更プラグインが動作出来る頃には、システムプラグインによっていくつかのアイコンが描画されてしまうからです。

悔し紛れに誰に向ける訳でもない愚痴ツイートを垂れ流していると・・・



それがとしぁ(@toshi_a)さんの目に留まり、アイコン変更機能はめでたくmikutterの新機能として議論されることになりました。

・・・くくく、作戦通り(*´∀`)

3. 「スキン切り替え機能（仮称）」について

ここで言う「スキン」とは、タブやボタンなどのアイコンのセットの事を指します。

これはmikutterがアイコンファイルの管理をSkinと言うクラスで行っているからです。

Winampの様なウインドウ装飾から部品の見た目まで一新する「スキン」とは意味が異なるので注意して下さい。そう言うのをやりたいのならば、GTK2のテーマについて繰繰れば幸せになれると思います。

3.1 スキンの使い方

スキンのインストールはとてもシンプルです。
$HOME/.mikutter/skinの下にスキン名に相当するサブディレクトリを作って、そこにpng画像をぶち込むだけです。

```
$HOME/
 .mikutter/
   skin/
    miq/ ←スキン名になる
      timeline.png
      reply.png
      activity.png
      ・・・
    miq-darkgray/ ←こっちも
      timeline.png
      reply.png
      activity.png
      ・・・
```

この状態でmikutterの設定画面の「スキン」を表示すると、先ほどインストールしたスキンが選択可能になっているはずです。

試しにmiq-darkgrayを適用した画面が下記となります。シンプル＆クールですね。

3.2 スキンの作り方

スキンの作り方もこれまたお手軽です。次のルールでアイコン画像の詰め合わせを作ればOKです。

(1)スキンのアイコン画像はpng。

(2)アイコンのファイル名は置き換えたいデフォルトアイコンのファイル名と同名にする。
（デフォルトスキンの画像ファイルは
MIKUTTER_DIR/core/skin/data/にあります。）

(3)スキンに含まれていないアイコンは自動的にデフォルトスキンが適用される。

3.3. プラグインからスキンを使う

先述の「miq」には、拙作プラグイン「私、気になります！」用のchitanda.png他、サードパーティプラグイン用のアイコンが幾つか収録されています。

確かにプラグインのアイコンも統一的なデザインに差し替えられると良いですよね。

そこで、サードパーティプラグインもスキンの恩恵が受けられる様に、プラグインDSLにskin()と言うメソッドを追加しました。

skin()は引数にファイル名を渡すと下記の順番でディレクトリを探索して、スキン画像のパスを返却します。

(1)ユーザースキン
$HOME/.mikutter/skin/(選択したスキン名)/

(2)プラグインが用意した画像ファイル
$HOME/.mikutter/plugins/(プラグイン名)/skin/

今までプラグインの画像ファイル置き場に規定はありませんでしたが、スキン機能が搭載後はここが標準になります。

(3)デフォルトスキン
MIKUTTER_DIR/core/skin/data
最終的にはデフォルトスキンが選択されます。

サンプルコードはこんな感じになります。

```
Plugin.create(:skin_test) {
  on_boot { |service|
    image_file = skin("chitanda.png")

    image = Gtk::Image.new(image_file)

    ...
  }
}
```

4. まとめ

・miqかっこいい。
・mikutter楽しい。
・10年前の俺へ。お前今コミケ原稿書いてるで。

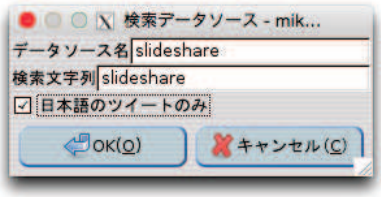
5 うわ、余った。

じゃあSlideShareの日本語スライドをはてなブックマークにかたっぱしから登録する芸やりまーす！

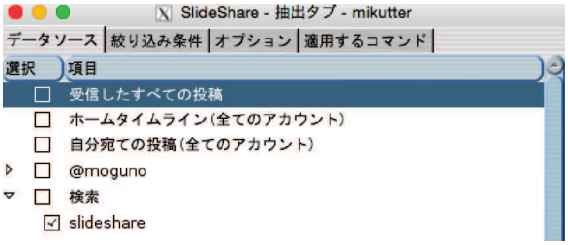
(1)プラグインをインストール
・mikutter-datasource-search
・mikutter-extract-fire-command
・mikutter-hatena-bookmark

(2)はてなブックマークの認証
設定から「はてなブックマーク」を選択して、認証を行ってください。

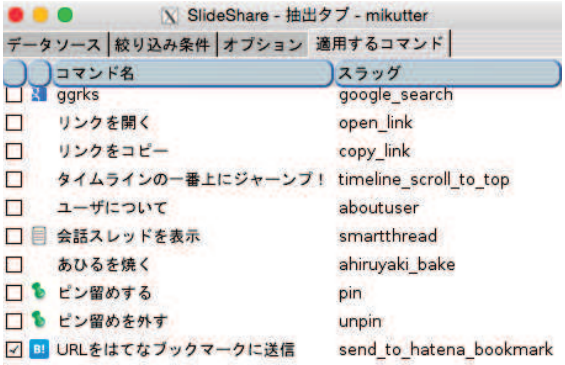
(3)検索データソースの設定
設定から「検索データソース」を選択して、"slideshare"を検索するデータソースを定義します。日本語ツイートのみを対象にするのがポイント。



(4)抽出タブにデータソースを設定
さっき作った抽出タブに、これまたさっき作った検索データベースをアサインします。



(5)抽出タブに適用するコマンドを設定
さっき作った抽出タブに、「URLをはてなブックマークに送信」コマンドをアサインします。



(6)しばらく待ってはてなブックマークを開く

時期にもよりますが、これで1日に数十件のスライドがゲットできます。

以上、でははでは一。

よりよい mikutter ハックのために

@toshi_a

1. ごあいさつ

mikutter は、10 月の 3.1 に続き、満 5 年となる 12 月 25 日には 3.2 をリリースしました。

多くの変更があった中でも、mikutter 3.1 が Ruby 1.9 系のサポートを切り、新たに Ruby 2.0 以降を対象としたことに今回は注目します。実は、mikutter5 年の歴史のなかで、サポート[1] する Ruby のバージョンを切るということは、今回を含めまだ 2 回しかないのです。前半では、サポートする Ruby のバージョンを変更した時の思い出話を通して、Ruby が新しいことは良いことだなあというお話をします。

後半では、そのことがプラグイン開発者にとってどのようなメリットがあるか考えます。mikutter は、Ruby を採用することで、プログラミングの経験が浅い人でも比較的簡単にプラグイン開発を始めることができます。一方 Ruby 自体への理解が浅いままなんとなく動くものが作れてしまうので、Ruby への理解を深める必要性に駆られることがあまりないようです。mikutter プラグインのためだけに Ruby を書くならそれでもいいのですが、ある程度長いことプラグイン開発をしているなら、Ruby 自体の知識を深めるのも悪くないと思います。その取っ掛かりとして、mikutter 3.1 から使える Ruby 2.0 の新機能のなかでも、mikutter プラグイン開発にすぐに威力を発揮しそうな項目をいくつか紹介します。初歩的な内容ですが、Ruby に自信のある人も何か気づくことがあるかもしれません。

1.1. 本稿の見方

event	mikutter コアやプラグインで定義される変数、クラス、イベント等です
Array	Ruby や外部ライブラリ、例示したコードで提供されるクラス等です
コマンド	mikutter 用語です
`code`	サンプルコードなどの一部です

2. Ruby 1.8 から 1.9 へ

2.1. 高速化への挑戦

2011 年、mikutter の開発が満一年になった頃です。はじめ、mikutter が bot として作り始められたのはあまりにも有名な話です。 その bot はプラグイン機構を持っていました。200 か 300 行程度の GUI を表示するプラグインを書いたところ、Twitter クライアントになり、mikutter という名前が与えられました。

しかし、その GUI はあまりにも非効率で、ツイート一つの描画を終えるのにかなりの時間がかかっていました。具体的なデータは残っていませんが、同時に 3 ふぁぼもされたら苦しいくらいには大変だったという記憶があります。その時は 100 人もフォローしていなかったように思うので、それでよかったのです。当然、100 人以上フォローしているよう

1 サポート…？ w

な人は、mikutter を使って、フリーズして全く動かないから、すぐに使用をやめていました。当時の私からすれば、100 人もフォローしているような人は変な人だったのです。

そういう私の考えを変えたのがふぁぼ爆撃に伴うふぁぼのインフレです。詳しくは語りませんが、ふぁぼられすぎて mikutter がまともに動作しなくなったのです。mikutter が使えない。高速化する必要に迫られ、何が遅い原因なのかを調べたところ、二つの問題が浮かび上がってきました。

- (Ruby)Gtk が遅い
- Ruby もっと速くなってほしい

どちらも本当は違っていたのですが…。とにかく、順番に見て行きましょう。

2.2. RubyGtk2 → RubyGtk2 + rcairo

これは Ruby のバージョン上げたことと関係はないのですが、上記のような背景があるので簡単に触れておきます。 今では全く珍しくもないですが、mikutter のタイムラインは、ツイートを一行で表示するようなものと比較すると非常にリッチです。このような UI を実現するのに、mikutter は Gtk の Box ウィジェットの中にいろんなウィジェットを詰め込み無理やり表現していました。

この方法のパフォーマンスは到底使えたものではありませんでした。Gtk に限らず、ウィジェットを頻繁に追加・削除するのは、あまり良い方法ではないように思います。一方で、頻繁なデータの追加・削除に強いと思われるリストビューは、1 行をカラムで区切って、それぞれのカラムに文字か画像を持つことしかできず、mikutter の UI を実現するには機能が足りませんでした[2] 。

ここで、画像が表示できることに注目しました。ツイート一つひとつを 1 枚の画像として扱うことにしたのです。スクロール処理などはリストビューに任せ、後のことは全て画像を動的に描き、それをリストビューにはめる。この方法は私の目論見通り、見違える速度になりました。毎秒 3 ふぁぼされてもフリーズしなくなり、快適な Twitter ライフを取り戻すことができたのです。

2.2.1. 結論

設計が悪かっただけ

2.3. Ruby 1.9 のサポート開始

一方で、Ruby での処理が増えたため、Ruby 自体の処理速度が気になり始めました。そこで登場するのが Ruby 1.9 です。

Ruby 1.9 は、1.8 といくつか非互換な変更がありました。そのせいか、多くのディストリでなかなか Ruby 1.9 へとすんなり移行できていなかったように思います。mikutter もタイミングが難しく渋っていたのですが、パフォーマンスが上がるよ、無料だよ、みんなやってるよなどと Rubyist に勧められて移行しました。

結果として、劇的に速度が上がりました。こちらも随分前のことなので詳しいデータや記憶は残っていませんが、 Ruby 1.9 の対応を開始した mikutter 0.0.3 のリリース記事[3] で

2 Gtk のツリービューは一つのカラムに複数のビューを持たせることもできるが、そういう話をしているのではない

3 http://mikutter.blogspot.jp/2011/04/mikutter-0030.html

は、3 倍速で動くと書いています。rcairo で適切な実装になった代わりに Ruby コードが増え、Ruby 1.9 でそれが速くなる。二つが合わさった mikutter は、もはや別のソフトに見えました。

2.3.1. 結論

Ruby 1.9 は神

2.4. 今の mikutter へ

こういった下積みがあったからこそ、mikutter は 1 ツイートを何秒で処理するかというレベルから、毎秒 10 ツイート[4] でもそれを適切にレンダリングできるまでに成長したのです。

しかし、これは何も私の努力だけではありません。この時得た反省のひとつに、できるだけ新しい物を使っておくべきである、ということがあります。速度最適化に関して、Ruby 開発者の人たちと私では、どちらのほうが良い成果を上げることができるか、容易に想像できることです。これは諦めなどではなく、長いものには巻かれろという名セリフにもあるように、和の心です。

以前動いていたバージョンを切るというのは、今まで mikutter を使っていた人が使えなくなる可能性もあります。一方で、いつまでもサポートするわけにもいきません。いいタイミングのひとつとしては、Ruby プロジェクトがそのバージョンのサポートを停止する時、というのがあります。

今回、Ruby 1.9 を切る理由はこれです。執筆時点では、 Ruby1.9 は 2015/2/23 に終了すると発表されています。一方で Ruby 1.8 は、それより早く mikutter のサポートを終了しています[5]。Ruby 1.9 との互換性を維持した状態でメンテナンスを続けていくことが難しいこと、Ruby 1.9 で mikutter を使ってほしいと思ったことの二つが大きな理由でした。

2.4.1. 結論

和の心、大切

3. Ruby の変化で mikutter プラグインはこう変わる

この節の内容は、Ruby の新しい機能に、mikutter プラグインの書き方がどのように影響を受けるのかを紹介します。したがって、ルビーチョットワカル人には、特に新鮮な内容はないかと思います。 また、mikutter を中心に書いているので、いくつかの事実については書いていないこともあります。最後に参考になる Web ページをいくつか示しておくので、より正確な情報についてはそちらをご覧ください。

mikutter プラグインの中には、古く冗長な書き方をしているものが散見されます。この機会に Ruby 力を上げて、プラグイン開発効率を上げましょう。

---

4 実際秒間何ツイートまで処理できるかは計測してません。ふぁぼのほうは、UserStream から秒間 20 以上落ちてくることがありました。ツイートは時間帯などで流れる量が大体決まってるのでどうでもいいですが、ふぁぼはピーク値が極端なので、気をつける必要があります。

5 2011/10/10 リリース mikutter 0.0.4.534 で Ruby1.8 のサポートを事実上終了している http://mikutter.blogspot.jp/2011/10/mikutter-003533-004533.html

3.1. 1.8 -> 1.9

まず、1.8 から 1.9 での変化をいくつかおさらいします。なぜなら、最近のプラグインでも、未だに Ruby 1.8 のシンタックスを使っているケースが散見されるからです。 これらは動いているから別に良いとも思いますが、一方で知らないと大きな生産性の違いに繋がってくることもあります。

3.1.1. Symbol をキーとしてもつ Hash

この二つの違いがわかりますか？

```
table = {
  :a => 1,
 :b => 2 }
```

```
table = {
  a: 1,
 b: 2 }
```

これは同じ意味です。Ruby 1.9 以降では、キーが Symbol である場合、それは `symbol:` と書けます。JSON に似ていますね。

mikutter では、可能な限り Hash のキーに String を使うような局面では Symbol を使用するように心がけています。mikutter API を使用する時にこれを知っているかどうかは大きな違いです。全てはこういったちょっとしたことの違いです。

3.1.2. Symbol#to_proc

Symbol といえば、配列の各要素に同じメソッドを呼ぶ時に便利なちょっとした変更がありました。例えば *Message* の配列 **messages** に対して、投稿者だけを抜き出してくることを考えましょう。Ruby 1.8 でも(たぶん)動くコードは以下。

```
posters = messages.map{|message| message.user }.uniq
```

これはこれで素直なコードです。Ruby1.9 からは以下のような書き方でも動作します。

```
posters = messages.map(&:user).uniq
```

map の引数に注目してください。 `.map(&:user)` の `&` は、次の内容をブロックとして引数に渡すという意味です。渡すブロックの内容に `:user` などと Symbol を書いた場合は、第一引数に対してシンボル名のメソッドを呼んでその結果を返す Proc を返します。言葉で説明するとややこしいですが、要は上記の二つが同じ意味であるということがわかれば、あらゆるところで使えるテクニックです。とくに *Message* や *User* には、こういったメソッドがたくさん用意されているので、知っていれば使う機会は多いはずです。

3.2. 1.9 -> 2.0

3.2.1. __dir__

Ruby 2.0 から、そのソースコードのあるディレクトリパスを得ることができるようになりました。今まで複数のソースコードからなるプラグインは、それをロードする時に以下のように書くのが習慣でした。

```
require File.join(File.basename(__FILE__), "other.rb")
```

これをこんな風に書けます。

```
require File.join(__dir__, "other.rb")
```

随分 type 量が減りましたね。

3.2.2. マジックコメント

Ruby 1.9 のことを考えたら、コードの冒頭に以下のようなコメントを書いておく必要がありました。これは、そのファイルの文字コードを Ruby インタプリタに宣言するものです。

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

これは必要無くなりました。mikutter プラグインでは UTF-8 を使うことを前提にしています。Ruby 2.0 からは、デフォルトのエンコーディングが ASCII から UTF-8 になったので、わざわざ書く必要がなくなったのです。

とはいえ、私は書いています。理由は、昔から書いていたからです。以下のようにファイルの冒頭に書いておけば、Emacs はファイルがその文字コードで書かれていると判断します。以下は、Ruby にも Emacs にも意味のある書き方です。mikutter のソースコードにはほぼ全てこれが入っていると思います。

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

書かなくても良くなった、というだけなので知らなくても害はないですが、他人に mikutter プラグインのうんちくを垂れる時には知っておかないと恥をかくかもしれません。

3.2.3. Enumerable#lazy

mikutter では、複数の *Message* を特定の条件で絞り込んで何かするというのは頻繁にある処理です。 Enumerable#lazy を使いこなせば、 Enumerable の変換・絞り込みなんかをエレガントに書くことができるでしょう。 簡単な説明と、mikutter でありそうな具体的な例を一つ示したいと思います。

ツイートのいずれかが条件にあうかどうか判別するというケースとしては、mikutter コマンドの実行可否判定があります。ここでは標準コマンドの条件「CanReTweetAny」をリファクタリングしながら、どのような違いがあるか見て行きましょう。まずは元のコードです。

```
# 選択されているツイートのうち、一つでも現在のアカウントでリツイートできるものがあれば真を返す
CanReTweetAny = Condition.new { |opt|
 opt.messages.any? { |message| message.retweetable? and not
message.retweeted_by_me? Service.primary } }
```

あるツイートがリツイートできるかどうかは、以下の条件で判別します。

- *Message#retweetable?* が真を返す
- *message.retweeted_by_me?(現在のサービス)* が偽を返す

このいずれをも満たすツイートが、リツイートできるものです。つまり、リツイート可能で、まだリツイートしていないものということです。

まずは select で書き換えてみます。

```
CanReTweetAny = Condition.new { |opt|
  !opt.messages
    .select(&:retweetable?)
    .reject{ |message| message.retweeted_by_me? Service.primary }
    .any?
}
```

二つの条件に分けて書いています。最初の select で、 *retweetable?* かどうかを判定しています。この書き方は前節で解説しました。 その条件にあう配列を作ったうえで、更に reject メソッドで自分がリツイートしたものを弾いた配列を作成します。 それで、残った配列に真になる値があるということは(any?)、この条件にマッチした *Message* があったということですから、 *CanRetweetAny* としては真になります。

この方法が非効率なことがわかるでしょうか。極端な例ですが、100 ツイート選択され、*retweetable?* なものが前半 50 ツイート、そのうち自分がまだリツイートしていないツイートは最初の一つだったとしましょう。 リファクタリング前なら、最初の一回目のループでこれを検出して真を返すことができました。しかしリファクタリング後では、 100 + 50 = 150 回のループ処理を行うことになり、明らかに無駄です。

こういった処理を効率的にできるのが Enumerable#lazy の使い方の一つです。先ほどのソースは以下のように書き換えましょう。

```
CanReTweetAny = Condition.new { |opt|
  !opt.messages.lazy
    .select(&:retweetable?)
    .reject{ |message| message.retweeted_by_me? Service.primary }
    .any?
}
```

といっても 2 行目に `.lazy` と書き足しただけです。 *retweetable?* が真を返すツイートがあれば、すぐにふたつ目の条件 *retweeted_by_me?* が評価され、これが偽なら、少なくともこの一連の処理は空配列ではないので、即座に真となります。

抽象的な例で、 pry でも確認してみましょう。1 から 100 までの値で、偶数のものだけを切り出してから、偶数のものがあるかどうかを計算します。この時、途中の map によって、偶数として処理された値が出力されます。

```
pry(main)> (1..100).lazy.select(&:even?).map{|x| p x; x}.any?
2
=> true
```

結果が true なのはいいですが、「2」と、最初の偶数しか出力されていませんね。最初の一回で any? の結果が決まったので、それ以上実行されなかったのです。lazy を削ると、2 から 100 までの値が全部表示されてしまいます。

どちらが見やすいかは人それぞれなのですが、mikutter では既に lazy を使い始めています。条件が長くなれば長くなるほどこちらのほうが見やすくなるからです。また、うまくやるには lazy を使うしかないという場面も出てくるでしょう。例えばイベントなどで Enumerable を渡すことができるようなものもありますが、そういったものに Enumerable#lazy を使えば、プラグインをまたいで計算量を減らすことができる可能性があると見ています。これは今までになかった発想です。

このトピックに関してもっと深く知りたい場合は、Rubyist Magazine の「無限リストを map 可能にする Enumerable#lazy」(http://magazine.rubyist.net/?0041-200Special-

lazy)などを見ると良いと思います。

4. これからの mikutter

4.1. キーワード引数がもたらすインターフェイスの変化

キーワード引数は、引数に名前をつけることができる機能です。 これのおかげで、従来では Hash でだましだましやっていたようなことをより簡単にできるようになりました。早速これを利用してみたところ、今までとは随分見た目が異なるものになってきたのでご紹介します。

例として、mikutter 3.2 の *extract* プラグインの新機能、抽出条件の拡張を見てみましょう。これを使えば、抽出タブの条件に、新しい条件を指定できるようになります。

```
defextractcondition(:bio, name: "bio", operator: true, args: 1) do |argument,
message:raise, operator:raise, &compare|
  compare.(message.user[:detail], argument)
end
```

defextractcondition が新たに提供されるプラグイン DSL メソッドです。ブロックには実際にツイートを受信した時に実行される呼び出される条件を指定しています。真を返せばこれにマッチしたことになります。 そしてこの例は、 「投稿者の **bio** に特定の文字列が含まれていること」 という条件を追加するためのコードです。こういう、自分にしか需要がなさそうなものを手軽に実装できるのが mikutter の†良さ†ですね。

早速このコードを、キーワード引数の観点から見ていきます。まずは **defextractcondition** の引数です。

第一引数

: 条件スラッグ。抽出条件の中でユニークなものを与えてください。

name:

: 表示名です。

args:

: 引数の数です。現在は 0 か 1 にのみ対応しています。

operator:

: true を渡すと、演算子コンボボックスが有効になります。デフォルトは false で、 **args** が 0 以外の値の時に有効です。

これは、キーワード引数を受け取る **defextractcondition** の呼び出しの例でした。最後の引数が Hash になっているのと見分けがつきませんね。 一方、一行目の do のあとは、キーワード引数を伴うブロックの定義です。

argument

: 通常の引数は、1 つ目がブロックの第一引数に渡されます。テキストボックスに入力された値なので、今のところ必ず String です。

operator:

: ユーザに選択された演算子を Symbol で受け取ります。 **&compare** だけで不十分な場合、例えば特定の演算子の時だけどうしても追加処理がいる場合に使います。

普通はいらないかと。

message:

対象のツイートを現す Message です。

&compare

演算子に応じて比較するブロックが渡されます。引数を 2 つ取り、二つの値がその演算子で比較した時に真になるかどうかを返します。例では `compare.()` と記述しています。これは `.call()` の省略形です。このシンタックスシュガーは Ruby 1.9 からですね。

これはいままでと大きく見かけが異なります。通常の引数のほかに、 キーワード: デフォルト値 という組が存在します。これがキーワード引数の受け取りです。ブロック内からは キーワード を変数名としてその値にアクセスできます。デフォルト値は省略した時の値です。Ruby 2.1 からは必須にすることも出来ます。つまり今は必須にできません 。なので、省略した時は raise で例外を飛ばすようにしています。

これからの mikutter API では多用していくことになる機能なので押さえておくべきなのは勿論のこと、 mikutter プラグインを実装するときにも便利な機能だと思います。覚えておいて損はないでしょう。

5. まとめ

今回は、あまり Ruby に詳しくない人を想定して書いてみました。これは知らなかったとか、今後はこう書こうとか思ってもらえたら幸いです。

mikutter をカスタマイズするために Ruby の細かい話を知る必要はありません。しかし、たまにはちょっと覗いてみるのも面白いと思います。

＜＜あとがき＞＞

@ch_print いかがでしたでしょうか。よいとしぁけを。

@brsywe 編集は計画的に。

――次号予告と寄稿者募集――

原稿提出期限：6 月 30 日

頒布予定：OSC Kyoto,コミックマーケット 88

なお、mikutter の薄い本別冊は @misodengaku がいいものを提出したときに刊行する予定。

問合せ先：@brsywe , @ch_print

奥付

発行日：2014 年 12 月 30 日(コミックマーケット 87)：冊子初版第一刷

2015 年 2 月 5 日：PDF 版初版

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

印刷・企画・編集協力：IUJK (special thanks: @EiM_GTPE_)

ご意見・ご感想は Amazon ギフト券のメッセージ欄にどうぞ。

mikutter の薄い本制作委員会ウェブページ

http://kohinata.sumomo.ne.jp/mikutter.html

mikutter の薄い本制作委員会では、Amazonギフト券・DMMギフト券による金銭面の支援を受け付けております。

もし、あなたがこの薄い本を読んで、何かしら満足感を得られたなら、送って貰えるとその満足感を誰かと共有できるかも。上のウェブページを御参照ください。