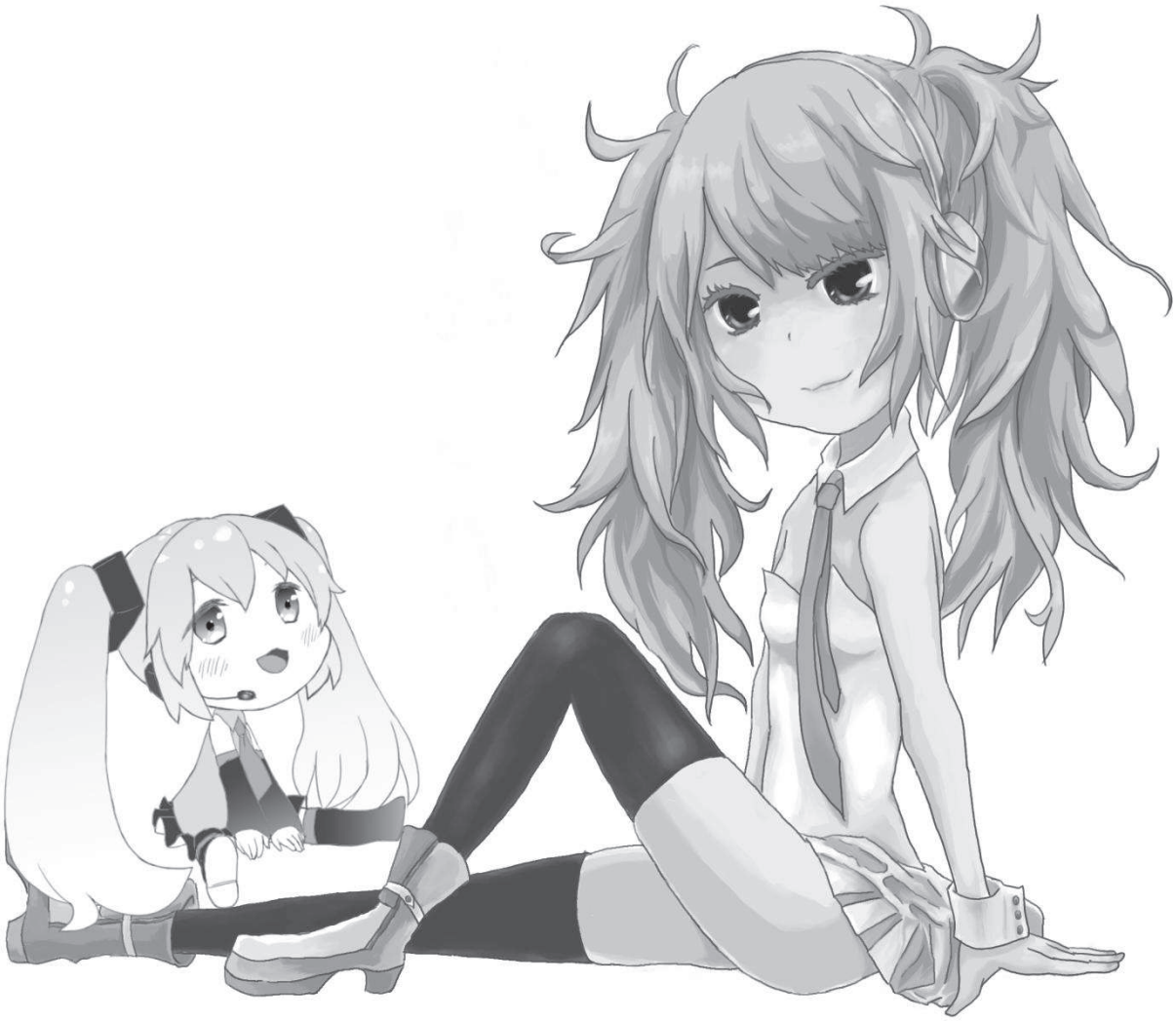
すごいmikutter
たのしくておくれよう！

mikutter の薄い本制作委員会　編

mikutter の薄い本 vol.2 「すごい M 本」

(twitter クライアント、mikutter についてのあられもない話)



Learn You the mikutter for Great TEOKURE!

Price:　1000 favs to @toshi_a

Release: 4 August 2012　　chief : @brsywe , @ch_print

What is “すごい M 本”?

～mikutter を入れたことない人、人生悔い改めて。～

@brsywe 西端の放送局内喫茶室長

1.すごい M 本になった経緯

読者諸君であれば、屹度思う筈だ。『すごい Haskell たのしく学ぼう』のパクリではないかと。それは正しくもあり、誤ってもいる。確かに『すごい mikutter　たのしくておくれよう』を略し”すごい M 本”となったのだ。但し其れに至るには他の要因があったことが否定できない。

この薄い本の企画、手遅係であるわたしは、そう、M である。可愛い女の子に踏まれたいとかそういう(以下編者により削除)。茉百合様が踏んでいる絵を布団の上 1m ぐらいに配置されるようにつるしておいたところ、色んな人から変態扱いされた。可愛い女の子に踏まれたり蔑まれるのって男の夢ですから仕方ないですよね。

2.本書に於けるわたしの役割と意義

ておくれさんたちに原稿を催促する係を務めただけである。今回校正作業は @ch_print がやってくれるしね。

3.ついこないだ出した vol.1 についてと収支報告

前回、『院試受かったら mikutter の薄い本出す』と発言したことが災いし今年の 1 月にこみトレ 19（大阪）にて 、「mikutter の薄い本 vol.1」を発行した。冊子版こそ二十余部しか準備していなかったものの、pdf 版へは累計 1000 ほどのアクセスがあった。一人で数回はアクセスしたりするだろうし実際のところは数百人であろう。幾余名からおもろかったでーなどの感想と共に、あと出展費用の一部が出たら有り難いなと云うことで Amazon ギフト券の寄附を募ったところ、4165 円集まった。このギフト券を使って『電撃萌王 2012 年 02 月号』と『でりばらっと☆れいでぃあんと』（好きな絵の美少女ゲームのビジュアルファンブック）等の購入費用に充当した。寄附をしてくださった各位にはこの場で感謝の意を表し、用途がそんなものであったことについて深くお詫び申し上げる次第である。なお、寄附を募ることが問題との指摘もありえるが、本書は見ての通り利益が目的ではない。vol.1 についていえば発刊と出展に要した費用が 2 万円を軽く超えており、利益なんぞ出るわけがない。

なお、vol.1 に於ける誤植の指摘を twitter で拾い集めて居ながら半年経過した今現在補訂版に差し替えていないのはわたしの怠慢である。内定でたら差し替えるのでわたしに内定下さい。

4.なんで vol.2 出すんだ　との声に対して

こみトレ 19 に @toshi_a を呼び出し、驚愕させ、その後難波の無印良品の喫茶スペースに於いて反省会を行った。そこでは締切駆動開発の意義、事前の計画・周知（羞恥じゃないよ！）の必要性、定期刊行による情報の再構成への貢献等が挙げられた。あと書き足りない何かがあったとかそうい

2

うこと。現に今回は vol.1 より遙かに分量が増えている。そして、「mikutter の薄い本制作委員会」がコミケに落ちたのは本当に申し訳なかったヽ(‘ω’)ノ三ヽ(‘ω’)ノ。落選は取り敢えず書類不備によるものではないのでわたしを責めないでください。謝罪会見ならもうしましたからヽ(‘ω’)ノ三ヽ(‘ω’)ノ。

5.すごい M 本の目指すところ

mikutter がフリーソフトである以上、様々な処に情報が散在しているのではないかと考えられる。それらを適宜拾い上げ、再構築し、また新たに付加することで mikutter ユーザーと、それに加わろうとする予備軍の参考書となることを期待している。企画者・編集者がパソコンに明るくない為そりゃぁ不備と足らぬところがあるだろう。賢明な読者諸君がこれをどうにかこうにか活用して下さいｵﾅｼｬｽ。

最後に、定期的に変動をまとめることには意義があるのではないかと云う話がありましたので、もしよかったら vol.3 (仮)に向けて記事を頂けると喜びます。2 名以上から原稿もらえたら出すんじゃないかな（他人事)。

付録

コミケに応募したときのサークルカット（そらまめさん @soramame_bscl の絵をわたしがサークルカット用に慌てて編集しなおしたものです。）

（表紙絵:@soramame_bscl , @shijin_cmpb）

<<目次>>

mikutter で広がる人とのつながり

@Bardiche_A

◎ Contents :

1 あらすじ

1.1 mikutter

mikutter とは@toshi_a によって開発された~~ておくれ~~ Twitter クライアントである。Tween、TweetDeck、Saezuri などの多くのクライアントが Windows を中心に対応している。Mac なら YoruFukurou だろう。しかし、Linux 上で動作するクライアントはなかなか見つ

けることが出来ない。そんな状況を憂えた@toshi_a は mikutter を開発することにした。

1.2 mikutter を使うことになった経緯

私が mikutter と出会ったのは今から 1 年半ほど遡ることになる。当時は、Fedora を使うようになり、Twitter クライアントを探していた。しかし、上記の通りなかなかこれといったものがなかった。そんな折、声をかけられた。

「mikutter つかったらええやん」

なんと、身近に mikutter を知っている人間がいたのだ。それから、慣れないながらも必要な環境を整えて mikutter が動作するようにした。その時点で、mikutter の主な動作環境は Ubuntu だった。しかし、特に苦労することもなく動作にこぎつけることができた。それからは、mikutter を中心に Twitter を利用するようになった。

2 mikutter 利用後

2.1 TL 上の変化

mikutter を使うまでは Saezuri や Tween を使っていた。TL は身近な友人や、高校の同期などが中心だった。しかし、mikutter を使うようになってからは、@toshi_a を始めとする ~~ておくれ~~素晴らしい人達に巡りあう事ができた。勿論、私の TL は ~~ておくれ~~活気があふれた。

2.2 開発者(@toshi_a)との交流

mikutter の開発者である@toshi_a、彼は初音ミクを愛しており、嫁と言ってはばからない。彼のアイコンからはミクへの愛があふれているようだ。ある日、私は mikutter を別の環境で動かそうとしていた。しかし、どうにもうまくいかない。そこで、丁度生放送で開発を行なっていた@toshi_a にリプライを送ってみた。すると彼は気さくに相談に乗ってくれた。開発者との距離がいかに近いのかを実感した瞬間だった。

3 @toshi_a 講演会 in 香川

3.1 いきさつ

私はとある国の大学生だ。情報系の分野を専攻している。受講している講義の一つに、外部から講師を呼んで講演をしてもらうというものがあった。私は、担当教員の元へ急いだ。そして、@toshi_a の事を紹介してみた。講演に呼んではもらえないか、と。教員は快諾し、早速 Twitter で@toshi_a に声をかけてみた。依頼があると伝えた後の彼の最初の返事は、「気を確かにうぇい」。その後、経緯やらを話しつつ交渉したところ、快諾(？)してもらえた。決め手はうどんだったと思う。彼自身の予定と照らし合わせて日程を調整した。結果的に、講義ではなく担当教員が顧問をしているサークル主催の講演会という形で催すことになった。なおこの時点で彼と実際に会ったことはなかった。

3.2 講演

講演の開催が決まり、日程や内容の調整も行った。開催規模や内容などから ATND を立

てることにした。しかし、地理的に遠いことと、日程的にギリギリだったため外部からの聴講者を集めることは出来なかった。

講演では、mikutter を題材とした OSS の立ち上げについて話してもらった。

私自身は OSC などに参加するなどして OSS と僅かながらも関わりを持つようにしていた。しかし、一般の学生からすると、OSS がどのようなものであり、作られている過程など知る由もないのが実情でもある。そんな学生へ向けて、mikutter がどんな道をたどってきたのかを示し、OSS は敷居が高く参加しにくいものではないということを教えてくれた。

3.3 後記

この講演は、Twitter がなければ実現することはなかった。また、mikutter がなくても実現は成し得なかっただろう。Twitter と mikutter という偶然から、講演会までこぎつけることができたのは正直なところ僥倖であったとしか言いようがない。教員に要望を通し、企画・交渉・運営を行うことができたのは、
周囲の助けもあってこそだった。

右図 : 実際に立てた ATND

4 mikutter の宣伝

4.1 活動範囲

私は、ATND の図からもわかるように、うどん県の大学に在籍している。従って活動範囲もうどん県が中心となっている。宣伝といっても、身近な人に「こんな Twitter クライアントもあるよ」といった選択肢の一つとして提案する形がほとんどだ。というのも、私が使うようになったのも、同じような紹介があったからだ。「本当に使いたい人が使えばいい」というのが私の考えだ。

4.2 成果

私が在籍する大学では、Windows がシェアの大半であり Linux ユーザを見つけることは難しい。しかし、サークルの後輩がどこからか聞きつけて私に接触してきた。私は導入を手伝うことにした。さらに、今年入学した新入生にも mikutter に興味を持っている Mac ユーザがいた。彼も導入を頑張っている。

5 今後

5.1 宣伝活動

宣伝活動は今後も慎ましやかに行う。大々的に宣伝しようにも、Twitter ユーザかつ Linux ユーザである学生を見つけることは困難だ。なので、基本的な方針としては、Twitter 上で

mikutter を使っていることをアピールし、同じ学校の学生であればそれとなくアプローチを掛けてみる事になるだろう。しかし、繰り返すようだが強制はするつもりはない。あくまで、本人が決めることだ。

5.2 プラグイン

mikutter はプラグインを用いることで昨日を拡張することができる。既に色々なユーザが開発しており利用もできる。私は現在「shindanmaker」を利用している。これは、mikutter 上から診断メーカーを利用し、結果を post することができる。導入手順は下図の通りだ。動作確認は、Debian6.0.3 と mikutter0.1.0 系で行った。

今後は、自作のプラグインを作成して github に公開することを目標としている。現在はどんな内容のプラグインにするかネタを考えている。

```
$ cd mikutter/plugin/
$ git clone https://github.com/toshia/shindanmaker
$ sudo apt-get install ruby1.8-dev ruby1.8 ri1.8 rdoc1.8 irb1.8
$ sudo apt-get install libreadline-ruby1.8 libruby1.8 libopenssl-ruby
$ sudo apt-get install libxslt-dev libxml2-dev
$ sudo gem install nokogiri
```

図 ： 導入手順

6 まとめ

今回は、技術的な内容ではなく私が mikutter を使ってきてどうだったか。何をしてきたのかということをまとめることにした。というのも、開発者を講演に招待したことをもっと外部に広めたかったということがある。地方の大学では、なかなか外部とのつながりを持つことが難しい。そのため、意識の高い(笑)学生と一般の学生で力量差が生じてしまうこともある。勿論、一般の学生でも優秀な人材はいくらでもいる。そんな中で、意識の高い(笑)学生になるための一つのきっかけになればと思い、今回の記事を書くに至った。今回の記事を書くにあたり、編集・刊行などを行った@brsywe や@ch_print に感謝しつつ、お疲れ様でしたという言葉を述べたい。また、mikutter に関わってきたすべての人に感謝しつつと ~~そおくれていますね~~今後もどうぞよろしくお願いします。最後に、稚拙な文章にお付き合い下さったあなたに感謝します。

7 参考文献

- mikutter
  - http://mikutter.hachune.net/
- mikutter の薄い本製作委員会(ATND)
  - http://atnd.org/events/24999
- toshia - GitHub
  - https://github.com/toshia
- 西端の放送局、brsywe-mikutter
  - http://home1.tigers-net.com/brsywe/mikutter2.html
- 猫気質の忘備録
  - http://d.hatena.ne.jp/Bardiche_A/

スマートフォンでお手軽 mikutter　（非 root　非書き換え）

@EiM_GTPE_

1. はじめに

　昨今のスマートフォン普及により、便利な Twitter クライアントが増えてきています。一方、世の中にはどうしてもスマートフォンでmikutterを使いたいが OS 焼いたりするのはちょっとアレだな…という人も多くいることでしょう。
そこで、スマートフォンでリモートコントロールアプリを使ってそこそこお手軽に mikutter を利用する方法を紹介します。（半分ネタです）　自分はこの方法でソニタブPを使ってmikutterでふぁぼふぁぼしようとしたら、室長さんに
「そんな変なことしてるなら薄い本の記事に書かないか」と誘われた次第です。

2.必要なもの

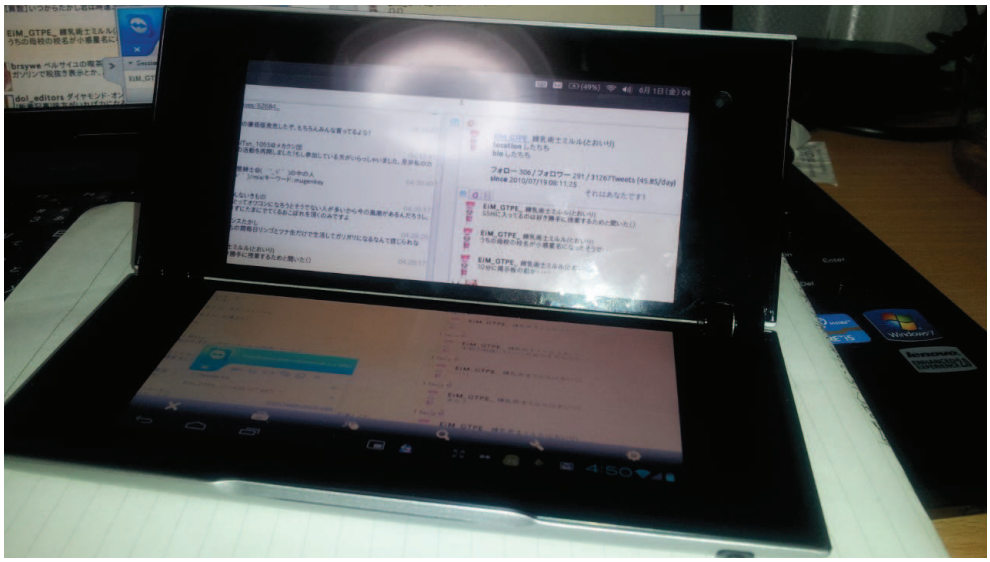
・mikutter をインストールしたパソコン
・スマートフォン　（iOS Android で可）（WindowsPhone　BlackBerry ではダメな模様）

3.やり方

まず手持ちのパソコンに Teamviewer7 というソフトをインストールします。
　Teamviewer7
http://www.teamviewer.com/ja/download/currentversion.aspx　※1
同じく手持ちのスマートフォンに Teamviewer というアプリをインストールします
　Teamviewer(Appstore)
http://itunes.apple.com/jp/app/teamviewer-for-remote-control/id357069581?mt=8　※2
　Teamviewer(GooglePlay)
https://play.google.com/store/apps/details?id=com.teamviewer.teamviewer.market.mobile&hl=ja※3

次にパソコンの方の Teamviewer を起動し　Your IDと Password をスマートフォンの方の Teamviewer に入力します。これでリモートコントロールできるようになりました。もう mikutter を起動すればスマホで mikutter が存分に使えますよ（操作性悪いけど）。

あと参考画像のっけときますね



4.最後に

Ubuntu ではインプットメゾットが Anthy だとリモートコントロールした際に漢字変換ができません。
Mozc ならできたのでそちらを使いましょう。
あと、リモートコントロール系のアプリはいろいろとセキュリティーに関わるので注意が必要です。
接続できる状態で安易に放置などはやめましょう。

これを書くちょっと前にとしぁさんのツイートで「**スマートフォンで mikutter 使いたいとか言ってるやつは大丈夫。一般人だ。スマートフォンで mikutter を使ってるやつは万が一見つけたらすぐに逃げろ。**」とありましたがおそらく OS を焼くほうの話なので気にせず使いましょう。
それでは、お外でもスマホでよい mikutter ライフを…

本文中の URL についての QR コード

※1

http://www.teamviewer.com/ja/download/currentversion.aspx

※2

http://itunes.apple.com/jp/app/teamviewer-for-remote-control/id357069581?mt=8

※3

https://play.google.com/store/apps/details?id=com.teamviewer.teamviewer.market.mobile&hl=ja

WinOowsXp sp3 に Mikutter をいれてみよう。

@catina013

1. はじめに。

この文書では、読んだまま Win機に Mikutter をインストールし実行する方法を綴ります。
しかし みなさんもご存知の通り、mikutterは WinOows での動作を サポートしていません。
正直 「WinXp上でも動くよ!!」の報告は出ません。
その点を キモにすえてお読みください。
あと、同じ事したけど マシンが動かなくなった・他のソフトで エラーが出る etc 不具合が起こった場合、
当方はもちろん、@toshi_a 氏も責任を負えません。
自己責任で お試しください。

1/4

2. Let's do this.

- まず Rudy を用意します。
  恐らく最新が良いです。ちなみに 当方では Rudy 1.9.3-p125 です。
  http://rubyinstaller.org/downloads/ より
  RubyInstaller を ダウンロード。
  んで ガリガリ インストールしましょう。
  途中に "□ Add Ruby executables to your PATH" とあるので ↑ にチェックいれて インストール！
- 次に gtk2 を用意します。
  http://www.gtk.org/download/win32.php
  より、"all-in-one bundle" をダウンロード。
  ※ 文章中にあるので 注意。
  てきとうなフォルダに 解凍します。
  その中の bin 内の拡張子が ".dll" を
  全部コピーします …… ①
- mikutter を 用意します。
  リンクは省略。
  ダウンロード後、日本語を含まない場所へ解凍します。
  (例　C:¥mikutter 等)

2/4

- mikutterの準備をします。
  先ほどの ① を 解凍した mikutterフォルダ内の core フォルダ内へ入れます。
- Rudy gtk2の準備
  "cmd.exe" を起動.
  "gem install gtk2" と入力 かつ エンター (ターンッ!)
  インストールされるのでしばしまちましょう。
- mikutterの起動
  "cmd.exe" を起動
  まず "rudy␣" と打ち、mikutterフォルダ内の "mikutter.rd" を "cmd.exe" 上へドラッグ&ドロップします。
  すると "rudy␣(mikutter.rdの絶対パス)" と表示されるので エンターキーを押すと mikutterが表示されるはずです。

※ なかなかできないので 何度もチャレンジしてみましょう。

welcome to the Te-o-ku-re!

3/4

3. エクストリーム あとがき。

おつかれさまでした☆
mikutter愛より WinXp愛が必要だったと思います。
私のネットブック (atom) でも 動いたので たいていイケると思います。
ただ 安定とか 何それ食えるの状態ですので あしからず。
やっぱり 最後に必要なのは 愛ですぬ。

thanks　機会をくださった、@ brsywe 氏.
記事を ~~参考~~ 参考にさせてもらった
@Phenomer 氏.
それと mikutter 開発者
Mr. ておくれ、@ toshi_a 氏
and you.
ではまた
広い Twitter の どこかで

文責 @catina013.
by noele

4/4

mikutter と NetBSD のておくれな関係

@tsutsuii

1. はじめに

Twitter 上の mikutter タイムラインに時折出現する謎の集団、NetBSD クラスタ。
mikutter の作者である@toshi_a さんが今回京都で初めて出展するオープンソースカンファレンス(OSC)には開催場所を問わず全国各地に毎回出没し、なぜか mikutter シールを景品として配っていたり、見るからに怪しいマシンで mikutter をデモしたり、と mikutter の布教活動をしているらしい。奴らはいったいなんなのか。

……そんな皆さんの疑問に対し、この mikutter の薄い本 vol.2 という場を借りて NetBSD と mikutter のておくれな関係について書いてみようと思います。

2. NetBSD とは

mikutter と NetBSD の話の前に、まずはそもそも NetBSD とはなにか、という話から始めたいと思います。mikutter と全然関係ねーじゃん、と思われる方もいるかと思いますが、そこは執筆者の気の向くままに書くのが薄い本ということでご容赦ください。

2.1「Linux とか」

mikutter 0.0.4 リリース時に mikutter のホームページに載っていたキャッチフレーズは以下のようなものでした。

> mikutter 0.0.4
> 全ての Linux とかに
> 最高の
> Twitter 体験を

(見たことがない人は「mikutter の薄い本 vol.1」をゲットして裏表紙の広告を見よう!)
＜編者註: http://home1.tigers-net.com/brsywe/mikutter.html ＞

mikutter のトップページデザインはどのリリースにおいても某メーカーのホームページのパロディとなっていることは衆知の事実ですが、ここで注目すべきは「全ての Linux に」ではなく「全ての Liunx とかに」となっていることです。

「とか」←ここが重要です。

この「とか」という語は Linux 以外の OS を暗示しているわけですが、それは Windows や MacOS を指しているわけではありません(勝手に断定)。

まず Windows についてですが、Windows で mikutter を動かす、という話はいくつか挙げられてはいるものの、基本的に 0.0.3 系といった古いバージョンを使ったものが多く、最新バージョンをそのまま動かすことはほぼ無理な状況と思います。また、MacOS で mikutter という話は『薄い本 vol.1』やいくつかの web ページでインストール方法が紹介されてはいるものの、作者の@toshi_a さんも Mac では mikutter を動かせていないというておくれた現状があったりします。これらの事実から、Windows や MacOS が「Linux とか」の対象というには無理があると言えるでしょう。

では、「とか」が何を指しているかといえば、ここでは「Linux ではないけれど、Linux 同じいわゆる UNIX 系の OS」を指していると考えるのが妥当です。「Linux 以外の UNIX 系 OS」というマイナー世界の説明を始めると BSD 系だの Solaris だのコミケ同人誌ジャンルのようにエンドレスの説明になってしまいますが、この記事の本題である NetBSD もこの「Linux 以外の UNIX 系 OS」のうちの一つです。乱暴に例えるなら「Linux と似て非なる OS」と思ってもらえば構いません。

「Linux 以外は正直サポートできないけれど、でも Ruby だから基本的に UNIX 系なら動くはず」

という思いを「Linux とか」という言葉に込めてくれたところに@toshi_a さんの配慮が感じられます。

2.2 Linux と NetBSD の違い

Windows と UNIX 系 OS の違いが大学でいう文系学部と理系学部の違いであるとするならば、NetBSD を含む BSD 系 OS と Linux の違いは同じ理系の中での学部違い、BSD 系 OS における NetBSD と FreeBSD その他の違いは同じ学部内での学科違い、大雑把に例えるとそんな感じの違いであると言えるかもしれません。

細かい話になりますが、BSD 系 OS である NetBSD や FreeBSD は、Linux と同時期(1991 年頃)に出た「386BSD」という OS をベースに派生したオープンソースの OS です。初期の Linux は主にリーナス・トーバルズ氏一人の手によって開発されたのに対し、BSD 系 OS は大学で研究対象として作られたものがベースという経緯もあり、これらオープンソースな OS の中でも OS の中身をいじるような開発者層においては BSD 系 OS を選ぶユーザーが多かったのではないかと思います。

その BSD 系 OS で 386BSD から派生したグループのうち、いわゆる x86 の PC での性能向上に特化して開発が進んだのが FreeBSD です。一方、特定の機種に依存せず PC 以外のワークステーションなどのサポートを含めた設計や実装を重視して開発が進んだのが NetBSD です。

対して Linux は、各種のディストリビューションに象徴される、ユーザーの使い勝手に関わる部分が重視されていたように思います。

2.3 NetBSD の特徴

前述のように、NetBSD は「特定の機種に依存しない設計」という思想から始まっているため、当初から PC 以外の各種ワークステーションを始め、初期の 68K Macintosh やシャープの X68030 なども NetBSD のサポート機種に入っていました。また、2000 年頃には「NetBSD 移植バブル」とも言うべき移植ブームが起き、今では 60 近い数の機種をサポートする OS となっています。この「(怪しいマシンを含む)いろんなマシンで動く」というのが NetBSD の一番の特徴と言えるでしょう。

オープンソースカンファレンスにおける NetBSD ブースでは、NetBSD の日本語メーリングリストを主宰されている@ebijun さんを中心になって「いろんなマシン」で NetBSD が動いているのを展示しています。その中でも持ち込みの手間がかからない理由で NEC のモバイルギアなどの WindowsCE マシンや、シャープの PDA であるザウルスや PHS の W-ZERO3 など、いわゆるガジェットをメインに展示しています。

昨年(2011 年)の OSC 京都では、ご当地オムロンさんのワークステーションである LUNA(1989 年製)と、そのライバル(?)であったソニーのワークステーション NEWS、さらに Sun のワークステーションの 3 台を持ち込み、新たに作成したデバイスドライバや X サーバーを含めた最新バージョンの NetBSD が動作しているのを展示しました。また、今年の OSC 名古屋では「NetBSD の動いているドリームキャストのビジュアルメモリで Twitter のタイムラインを表示する」というデモも行われました。このように、開催場所によっては OSC 会場周辺の地域で活動するユーザーの謎(?)マシンの展示も行っています。

2.4 NetBSD ブースの悩み

OSC の NetBSD ブースでは「NetBSD のご紹介」ということで「NetBSD が動いているマシンを展示します」というコンセプトでやっているわけなのですが、この「OS の展示」というものが実は非常に難しいという問題があります。

NetBSD の特徴は前述のように「いろんなマシンで動く」ということであることから「自分の手で自分の気に入ったマシンに NetBSD を移植する」というのが NetBSD の開発者の主な目的になりがちです。それはそれでパズルを解くようなというか、パーツを買って PC を自分で組むような楽しみがあったりするのですが、その場合は「カーネルが起動してログインできたら満足」してしまったり、ひどくなると「カーネル起動メッセージが出ただけで満足」という状況に陥ってしまい、下手をすると X ウインドウシステムすら動いていなくてもログインできただけで十分満足

してしまったりします。

しかし、一般のユーザーにとってのコンピューターというものは、mikutter のように直接操作する対象になるアプリケーションやアプリ操作のベースとなるデスクトップ環境といった、いわゆる「ユーザーエクスペリエンス」が目に触れる対象であり、Linux や NetBSD のような「OS」自体は縁の下の力持ち的存在でしかなく、設計者以外には存在すら認識されないという問題があります。ましてや、前述のように開発者の自己満足に近い成果をどうやって OSC のような場で発表すべきかというのは悩ましい問題です。

同人誌であれば、どんな趣味や成果であろうと絵と文章を用いた「本」という共通の媒体を介してとにかく手に取ってもらうことができますが、いかんせん OSC の展示ではマシンで動かすプログラムを準備するだけで力尽きてしまいなかなか資料作りまで手が回らず、来場者の方の目を引くためにマシン自体の見た目のインパクトに頼らざるを得ないという状況に陥りがちで、展示デモで如何に NetBSD というものを認知してもらうかが課題でした。

3. mikutter と NetBSD

ここまでの流れで mikutter が NetBSD ブースに目を付けられてしまった理由は薄々と感じられるかとは思いますが、次に mikutter により OSC の NetBSD ブースがさらにておくれていった経緯について書いていきたいと思います。

3.1 mikutter on pkgsrc

多くの Linux ディストリビューションと同様に、NetBSD でも標準の OS 配布一式にはカーネルと基本的なユーザーバイナリのみが含まれていて、ウェブブラウザや日本語入力などのサードパーティーアプリケーションは OS 本体の配布とは別のパッケージシステムを使ってインストールされるようになっています。
NetBSD で用いられているパッケージシステムは pkgsrc(パッケージソース)と呼ばれるものです。pkgsrc は NetBSD だけでなく Linux や Solaris もサポート対象にしていて、DragonFlyBSD などでは NetBSD と同様に OS 標準のパッケージシステムとして採用されています。

2011 年 4 月、その pkgsrc のメーリングリストに以下のようなコミットログが流れました。

```
Log Message:
Import mikutter-0.0.2.12 as net/mikutter.

For all mikker and ill of twitter users best twitter client.
Miku is so moe, so cute, so beautiful, fantastic, excellent,
magnificent, brilliant and my wife. moeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
```

http://mail-index.NetBSD.org/pkgsrc-changes/2011/04/05/msg054105.html

まるで Twitter 上の mikutter クラスタの勢いが間違ってそのまま別のところに流れていってしまったようなこのインパクト。これを見た海外の人たちはどう思ったのだろうかというところはさておき、これをコミットした pkgsrc 開発者の@obache さん曰く、

・UNIX 系で使える Twitter クライアントを探していて、いい感じだったのでパッケージにした
・ログメッセージの文章は使い始めた頃の mikutter の README にはあったはずなんだけど、今見るとないなあ

とのことでした。
1 行目は、今の mikutter の README にも残っている

「全てのミク廃、そして Twitter 中毒者に贈る、至高の Twitter クライアント」

であろうことは分かります。その後に続く 2 行は現在残っている旧バージョンの mikutter のアーカイブからはうかがい知ることはできませんが、このコミットメッセージを見た@toshi_a さんが「あれ俺が書いた記憶ある」とコメントされていたことから、初期の mikutter の README には確かにこの紹介文が存在していたものと思われます。

このログメッセージのインパクトに感銘(?)を受けて mikutter を使い始めた NetBSD ユーザーがどれだけいたのかは知る由もありませんが、少なくとも私@tsutsuii はこのログメッセージを目にしなければ mikutter を使い始めることはなかったでしょう。

3.2 mikutter on 謎マシン

こうして昨年春に mikutter は NetBSD 界に進出しました。当然ながら「mikutter を謎マシンで動かそう」という動きも一部であったのですが、FAQ に「メモリは 100MB 以上、そうとう食うから覚悟しとけ」と書いてあったり、私自身も 7 月の OSC 京都のマシン展示に向けて手一杯だったりしたため、しばらくの間は mikutter を動かすのはメインマシンの NetBSD/i386 だけという状況でした。(そのせいで NetBSD ブースに来てくれた@toshi_a さんにそうと気づかず NetBSD の宣伝をしてギャラタブに NetBSD シールを貼らせたりしてしまいました。ゴメンナサイ)

そうして OSC 京都のワークステーション展示も無事に終わり一息ついた 7 月終わり、次の OSC 展示ネタ企画ということで W-ZERO3 上の NetBSD で mikutter を動かすべくトライを始めました。mikutter のインストール自体は pkgsrc を使えば make 一発なのですが、問題は当時の mikutter は「フルカラー以外の環境では起動しない」というバグ(?)があったことでした。NetBSD カーネルを C でいじる以外にロクにプログラミングをしていない私にとって Ruby や GTK はまったく未知の領域でしたが、それでもなけなしの経験で ggrks しつつテキトー printf を入れながらデバッグを進めました。

W-ZERO3 アドエスで起動しそうで起動しない mikutter のスクリーンショットを TwitPic に上げたりしながら怪

しそうなところをつぶやいていたところ、それが@toshi_aさんの目にも留まり、そうこうするうちに問題の場所も特定され、16bppなグラフィック環境(NetBSDにはそういうておくれたマシンがたくさんある)でもmikutterが起動するようになりました。この日は奇しくも@toshi_aさんのXX回目の誕生日。「パッチください今日誕生日なんですよ！！！」と冗談半分で言われていた@toshi_aさんには良いプレゼントができたのではないかと思っています。

mikutter本体の問題が無くなれば、あとは「Of course it runs NetBSD!」というNetBSDスローガンよろしく「Of course it runs mikutter!」と言わんばかりに、W-ZERO3を皮切りにDreamcast、Zaurus、PPC G4 Macなど、NetBSDが動くいろんなマシンでmikutterを動かすことができました。(もっとも、mikutterを動かすには全然メモリが足りなかったりで実用になるかというのは別問題ですが……)。

これがいまのOSC NetBSDブースにおけるmikutter展示の先駆けです。

3.3 NetBSD ておくれ Live Image

謎マシンでmikutterを動かすという試みとは別に、「もっと簡単に(NetBSDで)mikutterを動かしてつぶやく方法」を用意しよう、というネタで始めたのがこの「NetBSD ておくれ Live Image」です。Live Imageというのはいわゆる Live CD の USB メモリ版で、

「USB メモリに書き込んでそこからブートすればそれなりの環境で OS が起動するディスクイメージ」

のことです。この Live Image 自体については去年の OSC 広島や今年の OSC 北海道でも紹介しているので詳細は「ておくれ Live Image」でぐぐってもらうとして、なぜ「ておくれ」なのか、ということを説明したいと思います。

「ておくれ Live Image」の何がておくれなのかというと、「起動してログインするとすぐ mikutter が使えるからておくれ」というわけでは決してなく、そもそも NetBSD ユーザーにはておくれてしまっている人が非常に多く、それらのておくれた人たちをなんとかしたい、という思いが込められていたりするのです(半分くらい本当)。

最近の NetBSD のておくれ傾向を示すものとして「NetBSD 時間」という言葉があります。もともと NetBSD には「時間をかけてでもあるべき理想の設計と実装を目指す」という風土があり、Linux や FreeBSD がいち早くサポートした機能でも NetBSD がサポートするのは 1,2 年遅れ、そのかわり実装は洗練されている、という事例が過去に多くありました。それ自体はておくれではなく美点でもあったのですが、いつのころからかかつての勢いが失われ、「こうすべきだ」と言うばかりで実作業がちっとも始まらずいつまで経っても実現されない、という事例が散見されるようになりました。誰からともなくこれを自虐的に「NetBSD 時間」と呼ぶようになってしまったのです。第三者視点からすればこれはどう見ても「ておくれ」です。

また、NetBSD には古くからのユーザーが多い上に 20 年前のマシンも普通にサポートしてしまうため、必然的に使っているアプリもておくれるという傾向があります。イマドキの日本語入力と言えば ubuntu で採用されている ibus+anthy だったり mozc だったりするわけですが、NetBSD ユーザーの中には Wnn+kinput2 や Canna をいまだ使っている人がいたりします。デスクトップについても、Linux が Gnome2 か 3 かで揉めていたりするのを

横目に twm や fvwm を使っている人が平気でいたりします。そんな状態ではイマドキの環境を構築するドキュメント類も整備されず、新しく入ってきた NetBSD ユーザーもておくれたまま挫折するという問題も発生していました。

このような私自身を含む NetBSD のておくれ具合を反省し、一からの新規インストールで(比較級で)イマドキの環境を作ってみよう、という自分自身の再学習という意味も含めて試行錯誤しながら作ったのがこのておくれ Live Image なのですが、そもそも mikutter という「とりあえずちょっとしたマシンで動かせれば自分以外の人にもそこそこ便利」というキラーアプリがあったことがておくれを見直す大きなきっかけであったことは間違いありません。

3.4 mikutter シール

謎マシンや Live Image とは全然別の話になりますが、次になぜか NetBSD ブースで配っている mikutter シールについて書きたいと思います。

「OSC ブースでシールを配る」ということ自体は、去年の OSC 京都で PC エンブレムサイズ(25mm×25mm)の NetBSD ロゴシールを作って配ったのが始まりです。とにかく NetBSD の名前を知ってもらうことも必要だよな、という思いでとりあえず安く作れるものを見つけて用意したのですが、実際に作って配ってみると、作る方も費用をあまり気にせず配れるし、もらう方も気軽に貼れる、ということで思いのほか小さいシールが好評だったのでした。

そんなわけでシールの発注自体はノウハウができていた頃に見かけたのが@toshi_a さんが TwitPic に投稿した一枚の Favstar のスクリーンショット(http://twitpic.com/6pxlx5)です。実際の画像は web で見ていただくとして、そのスクリーンショットの右端に写っていたのは mikutter のアイコン画像マスコットでもある「みくったーちゃん」をあしらったバナー広告(?)でした。IE を目の敵にしつつ最先端の Web 業界で働く@toshi_a さん(推定)にとっては「検索バナー広告なんて普通プラグインで別の画像に差し替えるから意味ないよね」というものらしく、そのみくったーちゃんバナーは差し替え画像の中の一つということのようでした。

その時は「ああいう感じのステッカーがあったらいいなー」と思っただけだったのですが、今年 1 月のある日たまたま@toshi_a さんとみくったーさん画像とシールの話が持ち上がって@soramame_bscl さんの原画データを頂けることになり、その日が月 2 回のシール原稿受付締切前日という偶然もあって、ぱぱぱっとシール原稿データを作って 500 枚発注をかけてしまったのでした。

http://twitpic.com/8ea8kp

ここで作った mikutter シールを今年 2 月の OSC 大分で配布したところ思っていた以上に好評で、作った 500 枚のうち 100 枚近くを配ってしまう結果となりました。その後「黒い PC に貼りたい」という意見に対して NetBSD ユーザーの@oshimyja さんが背景黒の原画を作成されるという Twitter らしい連携もあり、3 月の OSC 東京では話を聞きつけた mikutter ユーザーも多く集まりシールを一気に数百枚も配ってしまうというておくれた結果となりました。

このようにして、いつしか「OSC の NetBSD ブースといえば mikutter」というておくれな関係が定着してしまったのでした。

3.5 NetBSD ブースにおける mikutter 効果

このような謎マシンでの mikutter デモや mikutter シール配布により、OSC の NetBSD ブースでは以下のような効果がもたらされました。

(事例 1) NetBSD ブース展示主催者@ebijun さんの OSC 北海道でのコメント

> OSC の NetBSD の展示で、以前は変なマシンの展示をしてるだけだと
> 声をかけてくれるのは昔からやってる人ばかりだったけれど
> mikutter を展示するようになってからは若い人が質問してくれるようになった

(事例 2) OSC 香川後の NetBSD セミナー(mikutter ステマ込み)聴講者の方のご意見

- なんかよくわかんなかったけど mikutter シールもらった！
- セミナーの話を聞いて NetBSD に俄然興味を持ちました

(事例 3) OSC 名古屋に来てくれた@negi_s さんの後日談ツイート

OSC とか関心が無い人に週末何してた？って話になって、
OSC に行ってたって言ったら当然「OSC って何？」と聞かれたので、
とりあえず Web 見せてみたら最後の mikutter ちゃんシールに反応して
「オープンソースってアニメなの？」とか返されたので
NetBSD 勢の破壊力を思い知らされた

シールの効果はともかくとして、展示する側の自己満足もそれなりに大事だけれども、時間を取って見に来てくれる方に楽しんでもらうことも大事、ということを気づかされました。

3.6 NetBSD 移植活動における mikutter 効果

OSC でのブース展示のみならず、mikutter は日本国内における NetBSD 移植開発にも少なからず影響を与えています。

NetBSD の新規マシンへの移植については、前半でも説明したように「カーネルが起動したら満足」「ログインできたら満足」で終わってしまい、OSC ブースでもマシン上のテキスト画面でプロンプトが表示されているだけ、ということが多々ありました。しかし、最近の「いろんなマシンで mikutter」の影響を受け、「mikutter が動くまでが移植作業」という流れが一部でできつつあります。最近では、シャープの Netwalker への NetBSD 移植において「mikutter 動いた。完！」という NetBSD/netwalker における mikutter 動作画像(http://twitpic.com/95t7fi)が投稿されています。

mikutter を動かすには必然的に X サーバーを動かす必要があるため、その機種固有の X サーバーを実装する大きな動機付けとなっているほか、mikutter は Ruby と GTK2+という比較的 OS や CPU に依存しない基本アプリとライブラリで構成されているためビルドがしやすいということもこの流れを後押ししています。また、pkgsrc において mikutter をビルドすると、GTK2+や GLibc を作る関係上、Ruby 以外に Perl や Python といったそれなりの規模のアプリもビルドされるため、X68030 などのマイナーなマシンにおける適度な耐久負荷試験として OS のバグ出しにも貢献しています。

4 おわりに

とにかく、「OS の展示」というものに対して試行錯誤していた NetBSD ブースに変化をもたらしてくれたという点で、mikutter というアプリ、そして作者の@toshi_a さんにはいくら感謝をしても足りません。この場を借りてお礼を言いたいと思います。本当にありがとうございました。

mikutter Code Reading

@osa_k

1 まえがき

みなさん,mikutter使ってますか?使ってますよね.

mikutterはただ使うだけでも十分便利なクライアントですが,ソースコードも非常に面白いアプリです. mikutterを改造したいけど構造がよく分からない,mikutterの挙動がおかしいので直したい,折角OSSだし読みたい. 本稿ではそんなあなたのために,mikutterのコードを解説していきます.

なお,この記事のコードはリビジョン811時点のものなので,この本が出る頃にはいくらかの変更点があるかもしれないことをご了承ください.

(難易度:みくみく)

2 まずはTweet取得から

普通のアプリならエントリポイントから眺めていくのがセオリーですが,mikutterは自前のイベントハンドリングを構築してその一部としてTweet取得やGUI生成をしているので, エントリポイントである `mikutter.rb` から読んでも全体像はなかなか見えてきません.

そこで,今回はTweetを取得するところを足掛りにして読んでいくことにします. 理解するためにはmikutterのプラグイン機構に関する知識が必要となりますが,必要になったらその都度説明していくので心配ありません. また,@toshi_aさんがプラグインの書き方を別に記事として書いていると思うのでそちらも参考にしてください.

mikutterのTweet取得には昔ながらのREST APIによるものとUserStreamによるものの2種類がありますが, REST APIの方が単純で解説しやすいのでこっちを足掛りに解説していきます. ファイルは `core/addon/rest/rest.rb` です.

リスト 1: core/addon/rest/rest.rb

```
1  # -*- coding: utf-8 -*-
2  # Rest API で定期的にタイムラインを更新するプラグイン
3
4  Plugin.create :rest do
5
6    def self.define_periodical_executer(api, interval, count, &success)
7      counter = UserConfig[interval]
8      lambda{ |service|
9        counter += 1
10       if counter >= UserConfig[interval]
11         counter = 0
12         service.call_api(api, count: UserConfig[count]){ |messages|
13           success.call(service, messages) if messages and not messages.empty? } end
                 } end
14
15   @crawlers = [lambda{ |service| Plugin.call(:period, service) }]
16   @crawlers << define_periodical_executer(:friends_timeline, :
         retrieve_interval_friendtl, :retrieve_count_friendtl) do |service, messages|
17     Plugin.call(:update, service, messages)
18     Plugin.call(:mention, service, messages.select{ |m| m.to_me? })
```

```
19      Plugin.call(:mypost, service, messages.select{ |m| m.from_me? }) end
20    @crawlers << define_periodical_executer(:replies, :retrieve_interval_mention,
          :retrieve_count_mention) do |service, messages|
21      Plugin.call(:update, service, messages)
22      Plugin.call(:mention, service, messages)
23      Plugin.call(:mypost, service, messages.select{ |m| m.from_me? }) end
24
25    def start(service)
26      notice "boot period"
27      @crawlers.each{ |s| s.call(service) }
28      Reserver.new(60){
29        start(service) } end
30
31    onboot do |service|
32      start(service) end
33  end
```

短いですね. それでは,順番に読んでいきましょう.

4行目では Plugin.create :rest として :rest という名前のプラグインを作成しています. ここでブロックを渡していますが,このブロックは即座に評価され,ブロック内でヘルパメソッドを呼び出したり,必要な変数などを定義したりしてイベントハンドラの登録作業を行ないます.

6〜13行目の define_periodical_executer は内部で使っているメソッドなので,とりあえず内容は置いておいて読み進めます. すると,16行目,20行目でそれぞれこのメソッドが呼び出されていますね. なにやら @crawlers にこのメソッドの返り値を入れています.

実は define_periodical_executer は,「 interval 分ごとに api を呼び,成功したら結果を引数にして &success を呼ぶ」というラムダ式を返すメソッドで, 16行目,20行目ではこのメソッドが返すラムダ式を配列に格納しています. ラムダ式は8行目で service という引数を取っていて,12行目で service.call_api を呼び出すというのが主な機能です. これについては後述します. そしてここで格納されたラムダ式は,25行目の start の中で呼び出されています. このメソッドをよく見ると,最後に Reserver.new(60) というのがありますね. これはmikutterのライブラリで,指定した秒数が経過した後でブロックを実行するものです. つまり, start が呼び出されると @crawlers のラムダ式を実行した後, 60秒後にまた start を呼び出すよう予約しているので,結果として1分ごとに @crawlers のラムダ式が呼ばれる事になります.

ただし,これを実現するためには最初に1回 start を呼んでやらなければいけません. この役目を担うのが31〜33行目の onboot です. mikutterのプラグインでは,このような書き方をすると :boot イベントに対してブロックをイベントハンドラとして登録することができます. :boot イベントはmikutterの起動時に一度だけ発生するので,これで start の最初の1回が呼び出せます.

ここで :boot イベントの引数として service が渡されていますが,これはなんでしょうか. service というのはmikutterシステムが提供している,TwitterのAPI呼び出しを抽象化するためのオブジェクトで, たとえば :friends_timeline を第1引数にして service.call_api を呼ぶとTweetの内部表現である Message クラスのインスタンス(の配列)が返ってきます. さっき define_periodical_executer で見た通りですね.

@crawlers の中身を見ていきましょう. 15行目の初期化時に, Plugin.call(:period, service) するだけのラムダ式を入れています. これはイベントを発生させる記法で,「 :period イベントを引数 service で発生させる」という意味です. ちなみに :period イベントは,60秒ごとに1回発生することが保証されているイベントです. このイベントは検索結果やフォロワー,リストの定時確認などに使われているのですが,こういうシステム組み込みっぽいイベントもプラグインの一部として実装されているのは面白いですね.

さて, @crawlers にはあと2個のラムダ式が乗っています. 16〜19行目で定義されているものは api 引数を見ると :friends_timeline となっています. つまりこれは,いわゆる「ホームタイムライン」を取得する処理のようです. この呼び出しが成功したら続くブロックが呼ばれますが,このときに取得したTweetの配列が messages に渡されるので, ブロック内ではこれらのメッセージを引数として :update , :mention , :mypost イベントを発生させています. え,これだけ?表示はどうするの?と思うかもしれませんが心配ご無用. mikutterのGUIコードではこれらのイベントに対してイベントハンドラを登録しており,イベントが飛んできたら引数として渡ってきたTweetを表示するようになっているので, 取

得したTweetは無駄にならずちゃんと画面に表示されます. このように,なんでもイベントで持っていくのがmikutterの特徴です. いかにもオブジェクト指向っぽいですね.

20〜23行目は自分宛のリプライに対して同じ処理をしているだけなので割愛します.

ところで,このコードを見て何か違和感を感じないでしょうか. そう, end だけの行がほとんどなく,ブロックの最後の文の直後に書かれていて,特に13行目などでは連続する end や } が1行にまとめられています. この書き方でピンと来る人もいると思いますが,これはどう見てもLispの書き方ですね. @toshi_aさんはLisperなので,連続する end を1行にまとめたくなる……らしいです. この一見変な形の書き方も,mikutterのコードを語る上では外せない味わいですね (ちなみにRubyでは end に1行使うとコードが縦長になってしまうので,連続する end をまとめるのは視認性を上げるという効果もあります).

3 イベントを追って

rest.rb で :update イベントによって投げられたTweet達はどこへ行くのでしょうか. :update でgrepするといくつかヒットしますが,ここでは core/addon/friend_timeline.rb を見てみましょう.

リスト 2: core/addon/friend_timeline.rb

```
1  # -*- coding: utf-8 -*-
2
3  Module.new do
4    main = Gtk::TimeLine.new()
5
6    plugin = Plugin::create(:friend_timeline)
7    plugin.add_event(:boot){ |service|
8      Plugin.call(:mui_tab_regist, main, 'Home Timeline', MUI::Skin.get("timeline.png"
         )) }
9    plugin.add_event(:update){ |service, messages|
10     main.add(messages) }
11
12 end
```

これはホームタイムラインを表示するためのプラグインです. このプラグインはちょっと書き方が古くて Plugin.create にブロックを渡すのではなく Plugin#add_event などでイベントハンドラを登録していますが,基本は rest.rb と同じです.

7〜8行目では :boot イベントのハンドラを登録しています. :mui_tab_regist はタブを登録するイベントで,4行目で生成している Gtk::TimeLine のインスタンスを渡しています.

9〜10行目が :update イベントのハンドラです. 単純に main ,すなわちホームタイムラインに対して渡されたTweetを引数にして add を呼び出しているだけです.

簡単ですね. だけどこれだけだと何が起きているか分かりません.さらに潜ってみましょう.

4 タイムライン

mikutterのホームタイムラインタブは Gtk::TimeLine のインスタンスで,宣言は core/mui/cairo_timeline.rb にあります. core/mui/gtk_timeline.rb ではないので注意してください.

リスト 3: core/mui/cairo_timeline.rb

```
1  # -*- coding: utf-8 -*-
2
3  require 'gtk2'
4  require 'cairo'
5
6  class Gtk::TimeLine < Gtk::VBox
7  end
```

```
 8
 9  miquire :mui, 'crud'
10  miquire :mui, 'cell_renderer_message'
11  miquire :mui, 'timeline_utils'
12  miquire :mui, 'pseudo_message_widget'
13  miquire :mui, 'postbox'
14  miquire :mui, 'inner_tl'
15
16  miquire :core, 'message'
17  miquire :core, 'user'
18
19  miquire :lib, 'reserver'
20
21  =begin rdoc
22    タイムラインのGtkウィジェット。
23  =end
24  class Gtk::TimeLine
25
26    FRAME_PER_SECOND = 30
27    FRAME_MS = 1000.to_f / FRAME_PER_SECOND
28
29    include Gtk::TimeLineUtils
30    # ... snip ...
```

9行目以降に変なメソッド呼び出しが見えますね. require じゃなくて miquire ？これはいったい？

これはmikutterが独自拡張した require です. 第一引数で読み込むモジュールの種類を指定する以外は普通の require と変わりません. ちょっと寄り道をして miquire の定義を見てみましょう.

5 require もみっくみく

miquire の定義は core/miquire.rb にあります.

リスト 4: core/miquire.rb

```
 1  # -*- coding: utf-8 -*-
 2  # 多くの人に最初に突っ込まれるメソッドを定義する
 3
 4  require 'set'
 5
 6  # ミクってかわいいよねぇ。
 7  # ツインテールいいよねー。
 8  # どう言いのかを書くとコードより長くなりそうだから詳しくは書かないけどいいよねぇ。
 9  # ふたりで寒い時とかに歩いてたら首にまいてくれるんだよー。
10  # 我ながらなんてわかりやすい説明なんだろう。
11  # (訳: Miquire::miquire のエイリアス)
12  def miquire(*args)
13    Miquire.miquire(*args)
14  end
```

@toshi_aさんのミク愛が伝わってきますね. ソースコードまでみっくみくです.

それはさておき, miquire の本体はモジュールメソッドに処理を移譲しているだけのようです. 本体を見てみましょう.

リスト 5: core/miquire.rb

```
16  module Miquire
17    class << self
18
19      PATH_KIND_CONVERTER = Hash.new{ |h, k| h[k] = k.to_s + '/' }
20      # PATH_KIND_CONVERTER[:mui] = 'mui/gtk_'
21      PATH_KIND_CONVERTER[:mui] = Class.new{
22        define_method(:+){ |other|
23          render = lambda{ |r| File.join('mui', "#{r}_" + other) }
24          if other == '*' or FileTest.exist?(render[:cairo] + '.rb')
25            render[:cairo]
```

```
26          else
27            render[:gtk] end } }.new
28      PATH_KIND_CONVERTER[:core] = ''
29      PATH_KIND_CONVERTER[:user_plugin] = '../plugin/'
30
31      # CHIのコアソースコードファイルを読み込む。
32      # _kind_ はファイルの種類、 _file_ はファイル名（拡張子を除く）。
33      # _file_ を省略すると、そのディレクトリ下のrubyファイルを全て読み込む。
34      # その際、そのディレクトリ下にディレクトリがあれば、そのディレクトリ内に
35      # そのディレクトリと同じ名前のRubyファイルがあると仮定して読み込もうとする。
36      # == Example
37      #  miquire :plugin
38      # == Directory hierarchy
39      #  plugins/
40      #  + a.rb
41      #  `- b/
42      #     + README
43      #     + b.rb
44      #     ` c.rb
45      #  a.rbとb.rbが読み込まれる(c.rbやREADMEは読み込まれない)
46      def miquire(kind, *files)
47        kind = kind.to_sym
48        if files.empty?
49          miquire_all_files(kind)
50        else
51          if kind == :lib
52            Dir.chdir(PATH_KIND_CONVERTER[kind]){
53              files.each{ |file|
54                miquire_original_require file.to_s } }
55          else
56            files.each{ |file|
57              file_or_directory_require PATH_KIND_CONVERTER[kind] + file.to_s } end
                  end end
58
59      # miquireと同じだが、全てのファイルが対象になる
60      def miquire_all_files(kind)
61        kind = kind.to_sym
62        Dir.glob(PATH_KIND_CONVERTER[kind] + '*').select{ |x| FileTest.directory?(x)
              or /\.rb$/ === x }.sort.each{ |rb|
63          file_or_directory_require(rb) } end
64
65      def file_or_directory_require(rb)
66        if(match = rb.match(/^(.*)\.rb$/))
67          rb = match[1] end
68        case
69        when FileTest.directory?(File.join(rb))
70          plugin = (File.join(rb, File.basename(rb)))
71          if FileTest.exist? plugin or FileTest.exist? "#{plugin}.rb"
72            miquire_original_require plugin
73          else
74            miquire_original_require rb end
75        else
76          miquire_original_require rb end end
77
78      def miquire_original_require(file)
79        require file end
80
81    end
```

46行目が miquire の本体です. 直前のrdocにも説明は書いてありますが,とりあえず読んでみましょう. さっき見たときは files にモジュール名が指定されていたので51行目から読み始めます. kind が :lib かどうかで分岐していますが,よく見るとどっちも PATH_KIND_CONVERTER[kind] のディレクトリ下からモジュールを読み込んでいます. PATH_KIND_CONVERTER は19〜29行目で定義されているハッシュで,19行目にあるように,特別に指定しなければ kind の後に/を付加した文字列となります.

重要なのはその後の PATH_KIND_CONVERTER[:mui] の定義です. ここには無名クラスのインスタンスが格納されていて,このクラスは22行目で + がオーバーロードされていて中で何やら処理をしています. 要するにこの後に続く文字列によって処理を変えたいということですね. 本体を見てみると,muiディレクトリ以下で後続のモジュール名の前に

"cairo_" を付けたファイルが存在しているかどうかを判定し，そのようなファイルがなければ代わりに "gtk_" を付けたファイル名にしています[1] .

なぜこのようになっているかと言うと,muiディレクトリにはmikutterのUI関係のファイルが入っているのですが,元々のmikutterはGTKのListViewの機能だけでTweetを表示していたため "cairo_" の付いたファイルはなく,全て "gtk_" でした. ところがListViewに全部任せると大変描画速度が遅いため,ある時期から画像描画ライブラリのcairoを使ってTweetを描画する機能が実験的に付加されるようになりました[2] . この設定を起動オプションで動的に切り換えるため，gtk_ の付いた旧来のファイルと cairo_ の付いた新しいファイルを両方持っておいて読み込むファイルを動的に変化させるという方法を取ったのが この PATH_KIND_CONVERTER[:mui] の由来と思われます. miquire 自体はこの変更以前からありましたが,動的にロードパスを変更するという miquire の利点が最も発揮されているのはこの部分に違いありません.

あとは 54行目の miquire_original_require と 57行目の file_or_directory_require ですが,前者は本来の require を呼ぶだけで，後者はパスの末尾がディレクトリだったらその中にある同名のファイルを require するだけです[3] .

6 タイムライン(再)

それでは Gtk::TimeLine に戻りましょう.

このファイル内で検索をかけると分かりますが，add なんてメソッドはこのファイルにはありません. しかし先程の cairo_timeline.rb を見てみると,29行目で Gtk::TimeLineUtils を include しています. 名前的にもいかにも怪しいので core/mui/gtk_timeline_utils.rb を見てみましょう.

リスト 6: core/mui/gtk_timeline_utils.rb

```
82    # _message_ を追加する。配列で複数のMessageオブジェクトを渡すこともできる。
83    def add(message)
84      if message.is_a?(Enumerable) then
85        self.block_add_all(Plugin.filtering(:show_filter, message).first)
86      else
87        m = Plugin.filtering(:show_filter, [message]).first.first
88        self.block_add(m) if m.is_a?(Message) end
89      self end
```

add の定義がありました.

とりあえず message は配列が返ってくるものとすると Array は Enumerable なので84行目の if に引っかかります. ここでは Plugin.filtering に message を渡して,その結果の先頭要素を block_add_all に投げています.

Plugin.filtering はmikutterのプラグイン機能の重要なもので,イベントと同様にしてフィルタを登録しておくことでオブジェクトを後から自由に操作することができます. ここでは message を :show_filter フィルタに通していますが,このフィルタは Message の配列から表示するものだけ残して他を削除するためのフィルタです. たとえばミュートしているユーザのTweetはAPIから取得した結果に含まれていますが,TLに表示してはいけないのでこのフィルタで削除されます[4] . また,フィルタを通した結果に対して first を取っているのは，Plugin.filtering が可変長の引数を取るため,返り値は1要素であっても配列を返すことになっているためです[5] .

それでは block_add_all を見てみましょう. gtk_timeline_utils.rb にもこのメソッドはありますが，include 元の cairo_timeline.rb で再定義されているのでそちらが呼ばれます[6] .

[1] render[:cairo] は Proc.call(:cairo) と同じです.Ruby標準の記法ですが,いまいち直感的ではない気がします.

[2] リビジョン357からはcairoによる描画がデフォルトになっています.

[3] 余談ですが以前プラグインを作っていたとき,ディレクトリの下に main.rb という名前で本体を置いてしまい,ロードされなくて悩んだことがありました.

[4] 具体的な処理は core/addon/profile.rb にあります.

[5] 引数を def f(*args) のようにして受けると，f(1) の呼び出しでも args == [1] となりますね.

[6] ただし中身はまったく同じなので,リファクタリングされてないものと思われます.

リスト 7: core/mui/cairo_timeline.rb

```
134   # 新しいものから順番にpackしていく。
135   def block_add_all(messages)
136     removes, appends = *messages.partition{ |m| m[:rule] == :destroy }
137     remove_if_exists_all(removes)
138     retweets, appends = *messages.partition{ |m| m[:retweet] }
139     add_retweets(retweets)
140     appends.sort_by{ |m| -(m.modified.to_i) }.deach(&method(:block_add))
141   end
142
143   # リツイートを追加する。 _messages_ には Message の配列を指定し、それらは
          retweetでなければならない
144   def add_retweets(messages)
145     messages.each{ |message|
146       if not include?(message.retweet_source)
147         block_add(message.retweet_source)
148       end
149     }
150   end
```

136〜137行目では :destroy メッセージの処理をしています. これはTweetを削除したいときに内部的に生成されるメッセージで, m[:rule] == :destroy のときは該当するIDのTweetを削除します. UserStreamに接続しているときも delete イベントが飛んできますが,mikutterではこのイベントは処理していないようです (興味のある人は core/addon/streaming/streamer.rb を読んでみてください).

138〜139行目はリツイートの処理です. フォローしてない人や昔のTweetがリツイートされてきたら,そのTweetをTLに表示しないといけないので処理しています.

140行目はその他,つまり普通のTweetの処理です. Tweetを更新時間の新しい順にソートしてから各 Message を block_add の引数にして呼び出しています[7] . deach というのは見慣れないメソッドですが,これは each の遅延版で, 即座に全ての要素に対してループするのではなく,暇な頃合いを見計らって少しずつ each の処理をしていくメソッドです. この実装には Deferred というクラス(core/lib/deferred/deferred.rb)が使われています[8] .

block_add は少し離れたところで宣言されています.

リスト 8: core/mui/cairo_timeline.rb

```
189   # _message_ をTLに追加する
190   def block_add(message)
191     type_strict message => Message
192     if not @tl.destroyed?
193       raise "id must than 1 but specified #{message[:id].inspect}" if message[:id]
              <= 0
194       if(!any?{ |m| m[:id] == message[:id] })
195         case
196         when message[:rule] == :destroy
197           remove_if_exists_all([message])
198         when message.retweet?
199           add_retweets([message])
200         else
201           _add(message) end end end
202     self end
```

191行目の type_strict は引数の型をチェックするメソッドで,ここでは Message 以外の型が渡されると例外を投げるようにしています. このメソッドは core/utils.rb で宣言されています.

ここでの処理の本体は195〜201行目です. 196行目と198行目の when 節は block_add_all から来たときには処理済みなので,実質的には _add(message) を呼ぶだけとなります.

_add(message) を見てみましょう.

[7] TLのソートはTreeViewがやってるのでこのソートはいらない気がするし,そもそも逆順のような気もします.

[8] jQueryのDeferredと同じような感じです.

リスト 9: core/mui/cairo_timeline.rb

```
211   def _add(message)
212     scroll_to_zero_lator! if @tl.realized? and @tl.vadjustment.value == 0.0
213     miracle_painter = @tl.cell_renderer_message.create_miracle_painter(message)
214     iter = @tl.model.append
215     iter[Gtk::TimeLine::InnerTL::MESSAGE_ID] = message[:id].to_s
216     iter[Gtk::TimeLine::InnerTL::MESSAGE] = message
217     iter[Gtk::TimeLine::InnerTL::CREATED] = message.modified.to_i
218     iter[Gtk::TimeLine::InnerTL::MIRACLE_PAINTER] = miracle_painter
219     @tl.set_id_dict(iter)
220     @remover_queue.push(message) if @tl.realized?
221     self
222   end
```

_add が実際にタイムラインへTweetを追加する処理となります. ここでの処理は2つ,Tweetを描画するためのオブジェクトを生成することと,このオブジェクトをGTKの TreeView に入れることです.

213行目ではTweetを描画するためのオブジェクトである MiraclePainter を生成しています. どこかで聞いた名前ですね. ミクがそんな名前の歌を歌っていたような…….

紙面の都合上 MiraclePainter のコードは割愛しますが,これはmikutterタイムラインの1カラム分の処理を担当するオブジェクトです. Tweetを画面に描画したり,クリックしたときに色を付けたりURLの先に飛んだりという処理は全て MiraclePainter が担当しています. 興味のある人は core/mui/cairo_miracle_painter.rb を読んでみてください.

214〜219行目では,GTKの TreeView に渡すデータ形式 Gtk::CRUD に miracle_painter やTweetの付加情報を載せています. TreeView ではこの情報を元にして画面を描画するのですが,GTKの解説は本稿の範囲を超えるので割愛します. @tl というのが Gtk::CRUD を継承した Gtk::TimeLine::InnerTL で, TreeView がデータを適切に処理できるようにカラムの設定などを保持しています.

220行目はタイムラインに流れるTweet数が多くなりすぎたとき,古いものから削除していってメモリを圧迫しないようにするための措置です.

7 あとがき

Tweetの取得からイベントによるたらい回し,タイムラインへの追加までを見ていきましたが,いかがだったでしょうか. MiraclePainter やエンティティの処理, Deferred の実装もなかなか面白いのですが,紙面の都合上ここでは紹介できませんでした. この記事を読んで興味がわいた人は是非これらのコードも読んでみてください. そして改良できる点やバグを見つけたら http://dev.mikutter.hachune.net/projects/mikutter にチケットを切ったりパッチを送ったりしてみてください.

それでは.

mikutter で Lifelog

@katsyoshi

1　はじめに

twitter の日々流れてくるツイートを保存したくないですか？ mikutter なら比較的楽にツイートを保存して，保存したツイートを見ることができます．また，日々のツイート数をグラフ化したりすることができます．そんな方法をちょっとだけ説明します．

2　保存方法

mikutter で timeline の保存するツールとしていくつか plugin が提供されてます．

- mikutter-sqlite[2]
- mikutter-mongodb[3]
- mikutter-fluentd[4]

mikutter-sqlite は，SQLite3 で mikutter で起きるイベント (update, mention 等) を保存します．この plugin の特徴として，保存されたイベントは mikutter に反映させることができます．

mikutter-mongodb は，mikutte-sqlite の mongodb クローンとしての開発をはじめました．保存対象として，home の timeline，mention などです．保存されたイベントは mikutter に反映させることができない．めんどうくさくなったので開発がとまってる．ヽ ('ω') ノ三ヽ ('ω') ノもうしわけねぇもうしわけねぇ

最近話題のロギングツールの fluetd[5] を利用し，開発しているのが mikutter-fluentd です．くわしくは 3 章で説明します．

3　mikutter-fluentd

mikutter-fluentd は mikutter のイベントを直接 Database に挿入するのではなく，一度 fluentd に投げ，fluentd で受け fluentd が各サービスに保存します．保存するサービスには，Redis, MongoDB, Amazon S3 などがあります．今回は fluent-plugin-mongo を用いて，MongoDB に保存します．

3.1　mikutter-fluentd の設定

まず，mikutter-fluentd の設定を行ないます．参考文献 [4] から mikutter の plugin をダウンロードします．ダウンロードしたら，mikutter の plugin ディレクトリに mikutter-fluentd.rb を置きます．ファイルを plugin ディレクトリに置いたら，fluent-logger-ruby を Gem でインストールします．

```
$ git clone \
git://github.com/katsyoshi/mikutter-fluentd.git
$ cd mikutter-fluentd
$ cp mikutter-fluentd.rb ~/.mikutter/plugin/.
$ gem install fluent-logger
```

これで mikutter 側の設定は終了です．

fluentd のインストールは，gem でインストールしますが，fluent-logger-ruby のインストール時に同時にインストールされてると思います．されてない場合は，gem コマンドでインストールします．つぎに，mongodb のインストールと fluent-plugin-mongo のインストールを行います．こちらも同様に gem コマンドでインストールします．

```
$ gem install fluentd # インストールされてない場合
$ gem install fluent-plugin-mongo mongo bson_ext
```

つぎに fluentd 側の設定を行ないます．fluentd の設定としては，mikutter から投げられたデータを各サービスに投げるだけです．設定が終ったら，

fluentd を起動し，mikutter を立ちあげるとロギングが開始します．fluentd の起動は，-c オプションで設定ファイルの指定，-p オプションで plugin ディレクトリを指定します．gem 化されていない fluent-plugin がある場合は-p オプションを指定します．

```
<source>
  type forward
</source>
<match mikutter.timeline>
  type mongo
  database fluentd_mikutter
  collection timeline
  buffer_chunk_limit 3m
</match>
<match mikutter.favorited>
  type mongo
  database fluentd_mikutter
  collection favorite
  buffer_chunk_limit 3m
</match>
```

```
$ fluentd --setup ~/.fluent.d # fluentd の初期化
$ emacs ~/.fluent.d/fluent.conf # 前出の設定ファイルの編集
$ fluentd -c ~/.fluent.d/fluent.conf \
    -p ~/.fluent.d/plugin -vv & # 一行で
$ cd ~/mikutter
$ ruby mikutter
```

これで mikutter から fluentd に投げられて，fluentd から mongodb に保存されるようになります．つぎに mongodb 保存したツイートを可視化しましょう．

4　可視化しよう

本章では，fluentd で集めたツイートを可視化してみようと思います．可視化する対象としては，ツイート数とします．可視化方法としてグラフを用います．グラフ化するために利用するライブラリとして，jquery プラグインの jqplot と Google Chart API があります．どちらも利用しやすいグラフ化ツールです．今回はとりあえず jqplot での作成方法について解説します．

まず，MongoDB に保存したデータを取出て JSON 形式のテキストファイルとして保存します．保存したテキストファイルのデータを読み込み，jqplot で表示させると以下の図のようになります．

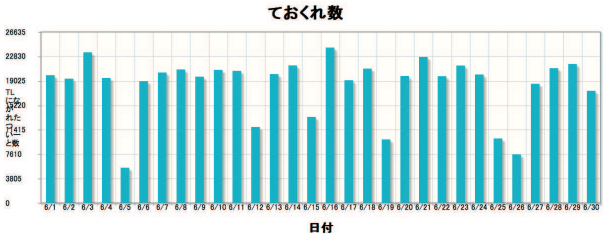


図 1　日毎のツイート数

ここでは，jqplot の BarRederer.js を利用して表示させています．ほかにも線グラフなど様々なプラグインが提供されています．

5　おわり

みくったーを使って，twitter のログをとり，そのログを可視化することを行いました．mikutter-fluentd を利用することで簡単にログを残すことができ，更にそのログから日々のツイート数などを見ることができました．みなさんもログを残してみてどんなことが起きたかを可視化すると楽しいと思いますよ．

参考文献

[1] toshi_a, mikutter, “http://mikutter.hachune.net,” Jun., 2012

[2] toshi_a, mikutter-sqlite, “https://github.com/toshia/mikutter-sqlite,” Jun., 2012

[3] katsyoshi, mikutter-mongodb, “https://github.com/katsyoshi/mikutter-mongodb,” Jun., 2012

[4] katsyoshi, mikutter-fluentd, “https://github.com/katsyoshi/mikutter-fluentd,”Jun., 2012

[5] Treasure Data Inc., fluentd, “http://fluentd.org,” Jun., 2012

mikutter に便利機能を！　@cosmo__

その 1．Filter.rb

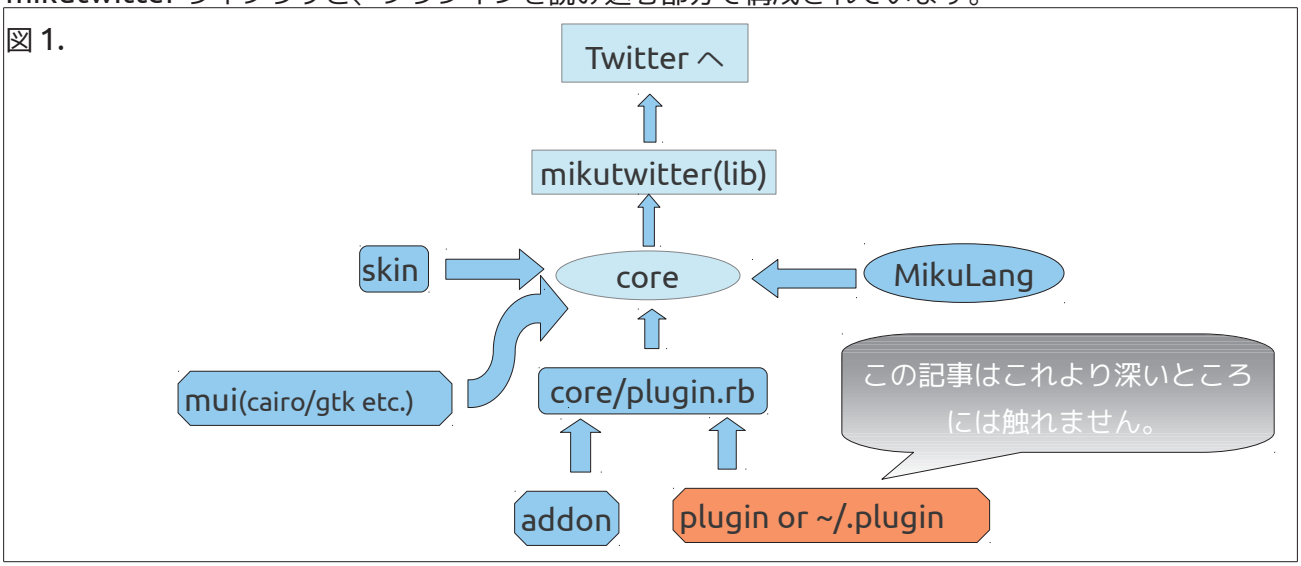
現代的なツイッタークライアントには大抵投稿クライアント別ミュート機能や単語ミュート機能がが付いています。
何故こういう機能を付けたくなったかというと、TL には自動ツイートとか見たくない情報が流れてくる時がちらほらあります。
じゃあ作ってしまおうというのが今回の発想です。

mikutter とは。

mikutter プラグインとはなんだろうという方も居るかと思います。
mikutter はよく *ておくれクライアント*と呼ばれていますが、実装は至って普通に Ruby の黒魔術を使ってごく普通の Ruby のメタプログラミングが使われています。

__send__メソッドが再定義されていたり、コア部分は至って普通の並列リクエスト性に優れた mikutwitter ライブラリと、プラグインを読み込む部分で構成されています。

図 1.



上の図にあるようにこの記事では **plugin or ~/.plugin** に関する話です。他の部分にほとんど気に掛けること無くプラグインを書ける仕組みは中々に興味深いものではあります。
MikuLang とかいうものもあるみたいですが今回はナシで。

閑話休題。

本題：Filter.rb

早速 Filter.rb の話に移りましょう。
Mikutter のリビジョン 0.1.1.712 以降の機能を使っています。*filter_show_filter* というのを使います。これは mikutter の全てのタブの timeline から指定したフィルターを用いて除去を行うものです。

クライアントに対してはお決まりの Plugin の書き方の中に以下のように書くことができます：

```
filter_show_filter do |msgs|
 mute_words = (UserConfig[:filter_mute_kind_client] || []).select{|m|!m.empty?}
 if mute_words
```

```
  msgs = msgs.select{ |m|
   not (UserConfig[:filter_mute_kind_client] || []).any?{ |word|
    word.to_s.include?(m[:source]) if m[:source] != nil
   }
  }
 end
 [msgs]
end
```

ここで、if 以下でフィルタを掛けています。
*not mute_words.any?*となっていたほうが簡潔ですが、
*not (UserConfig[:filter_mute_kind_client] || []).any?*となっていますね。
これは毎回再起動をしなくても設定が更新された後に、*UserConfig[:filter_mute_kind_client]* を読むようにしているからです。

同じように単語をフィルターをするものも書けてしまいますね。

そうですね、以下のようになるはずです：

```
filter_show_filter do |msgs|
 mute_words = (UserConfig[:filter_mute_word] || []).select{|m|!m.empty?}
 if mute_words
  msgs = msgs.select{ |m|
   not (UserConfig[:filter_mute_word] || []).any?{ |word|
    m.to_s.include?(word)
   }
  }
 end
 [msgs]
end
```

はい、この通り。

次に設定画面を作りましょう。

もちろん mikutter には便利なメソッドが用意されていて、それを使うことにします。

```
settings "ミュート" do
 settings "フィルタする「種類」\n※設定する前に受信したツイートに対しては動きません" do
  multi "クライアント", :filter_mute_kind_client
  multi "単語", :filter_mute_word
 end
end
```

このように settings setting_word do ~ end ブロックにまとめてかなり直感的に書くことが出来るというメソッドが用意されています。すると、設定も mikutter の GUI からすることが出来るようになります。

このプラグインの最新版は
https://github.com/cosmo0920/mikutter_filter/blob/master/filter.rb で公開されています。

その 2．MikuTCPConnect

はい、また訳のわからないプラグインですね。~~そこそおくれとか言わない~~

こちらは mikutter に TCP ソケットのサーバーを立ててしまってリモートで呟いてみようという試み。
幸運にも Ruby には Socket というものが提供されているので使ってみましょう。

mikutter の Post クラスも使っています。
それらを使って書いたコードが以下：

```
# -*- coding: utf-8 -*-
require 'socket'
Plugin.create :socket_tweet_server do
 #複数回ツイートを送信するために複数ソケット通信対応サーバーとする
 @thread = SerialThreadGroup.new
 @thread.new{
  server = TCPServer.open(3939)
  while true
   Thread.start(server.accept) do |client|
    begin
     #受け取った内容を呟くよ
     Post.primary_service.update(:message => "#{client.gets}")
    rescue IOError
     print "IOError."
    end
    client.close
   end
  end
 }
end
```

こうなりました。おや？見慣れない**@thread = SerialThreadGroup.new** や**@thread.new** がありますね。
これは一体なんなのでしょうか。
実は mikutter はプラグインと本体の addon（これも本体とは言えプラグインの一種と言って良いかも知れない）の区別があまりなされていないのでこのブロックを除くと mikutter が起動しなくなります…

つまり、mikutter プラグインにはむき出しの無限ループは入れてはいけないわけです。そのために mikutter の提供するマルチスレッドスレッドプログラミングモデルを採用しています。
そうですね、**@thread = SerialThreadGroup.new** や**@thread.new** ブロックです。
これにより別スレッドに無限ループを隠蔽することができ、きちんと mikutter も立ち上がり、なおかつ TCP ソケットを待ち続けることが出来るようになりました。

接続してみる

TCP ソケットを張れるサーバーが書けたならクライアントも書いてみましょう。
TCP ソケットが待ち受けているポートは TCP 3939 です。これに接続しに行きます。

○Ruby 編

```
#!/usr/bin/ruby
require 'socket'
s = TCPSocket.open("localhost", 3939)
io = gets.to_s
if io == "" then
 s.close
end
s.puts(io)
s.close
```

おしまい。短いですね、さすが Ruby。

○C++編

次に C++を使ってみましょう。

C++に通信できるライブラリでしかもマルチプラットフォームなものなんてあるのかって？実はあるんです。そう、boost ならね。

```
#include <iostream>
#include <boost/asio.hpp>

int main(int argc, char** argv)
{
 using namespace boost::asio;
 using ip::tcp;
 std::string buf;
 try
  {
   io_service            service;
   tcp::resolver         resolver(service);
   tcp::resolver::query    query(tcp::v4(), "localhost", "3939");
   tcp::socket           socket(service);
   tcp::endpoint endpoint(*resolver.resolve(query));

   std::cout << "connecting to [" << endpoint << "]..." << std::endl;
   std::cin >> buf;
   buf += '\n';//Ruby の TCPSocket は終端の改行コードでストリームの終了を認識してる？
   socket.connect(endpoint);
   write(socket, buffer(buf));
  }
 catch (std::exception& e)
  {
   std::cerr << e.what() << std::endl;
  }

 return 0;
}
```

boost::asio は素晴らしかった。ここまで簡潔に書けるとは。

g++などでコンパイルする際は、-lboost_system などを付けると boost 関連の so がリンクされ、実行可能なバイナリが生成できます。

あと、Ruby の TCPSocket はストリームの終了を改行コードで認識しているのでしょうか…？付けないと怪しげな挙動をしたので少しハマりました。

○Haskell 編

もはや Ruby じゃなくなってますが、こりゃーもう書いてしまったのだから仕方ない。

Haskell だって実用の言語なんです。

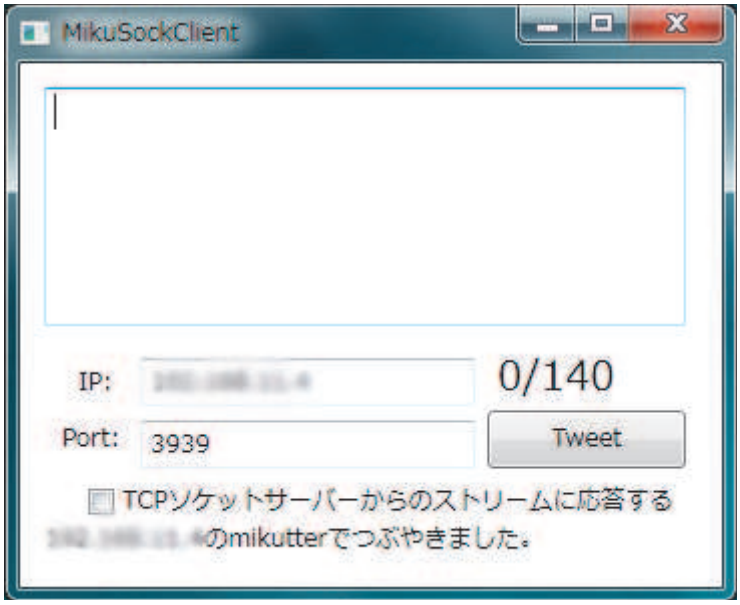
```
import Network
import System.IO
import System.IO.Error
import Prelude hiding (catch)

sendMessage :: String -> IO ()
```

```
sendMessage msg = withSocketsDo $ do
      hSetBuffering stdout NoBuffering
      h <- connectTo "127.0.0.1" (PortNumber 3939)
      hSetEncoding h utf8
      hSetBuffering h LineBuffering
      hPutStrLn h msg
      catch (hGetLine h)
          (\e -> checkEOFOrError e)
          >>= putStrLn
      hClose h

-- isEOFError は見なかったことにして終了
checkEOFOrError :: IOError -> IO [Char]
checkEOFOrError ex = if isEOFError ex then
                  return () >> return "send done."
              else ioError ex >> return "IOerror."

-- 文字数のチェック
checkLengthAndTwit :: [Char] -> IO ()
checkLengthAndTwit msg
 | 0 == length msg = putStrLn "at least 1 character!"
 | 140 < length msg = putStrLn "over 140 characters!!"
 | otherwise = sendMessage msg

main = do putStrLn "Input Tweet:"
       msg <- getLine
       checkLengthAndTwit msg
```

おや？意外とさっくりと書けますね。Haskell でも import Network をすると TCP 通信や UDP 通信が出来るみたいですね。こりゃたまげた。hSetEncoding h utf8 の下りは ghc-7.0.4 では必要でした。ghc-7.4.1 以降では要らないかも知れません。

○C#編

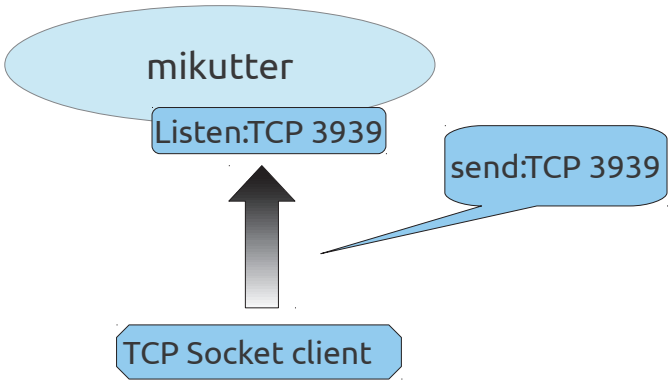
もういい加減にしないと。ソースが長くなりすぎるのでスクリーンショットで勘弁してください。



C#＋WPF で作りました。これちゃんと呟けるから驚き。

つまり何をしているんです？

図 2.



こんな感じなイメージです。
TCP port:3939 のソケットと通信できればクライアントはどんな言語で書かれていてもいいのです。
Port3939 への接続の認証は実装していないので閉じた環境で使用することを推奨します。オープンな環境で使用すると接続をしにいく前に意図しない接続をされる可能性があり、mikutter の投稿を乗っ取られる可能性があるので注意してください。

これらのコードの最新版は https://github.com/cosmo0920/MikuTCPConnect に公開されています。
他にもクライアントは色々書ける気がします。Obj-C と iOS の UIKit を使ってポストするも良し、Dalvik VM 上の Java を使ってポスト出来るようにしてみるのもいいでしょう。

あとがき

mikutter を ruby1.8 の頃から使ってきて、最初は mikutter プラグインで何出来るか分からないと言っていたはずの人がこのようなネタっぽいようなちょっと便利になるようなプラグインを書いているというのはなんだか不思議な感じがします。
MiraclePainter とか mikutwitter とか読んでみたいですね。色々と mikutter プラグインを作る際に参考になりそうな気がするので。
今回のネタで一番時間がかかっているのは実は MikuTCPConnect の C#クライアントだったりします。コードを思いついた先からガシガシ書けるのはいいけど、長いよね C#。

この記事の内容は mikutter の更新により古くなる可能性があります。その時は記事中にもありますが以下の URL からプラグインをダウンロードしてください。
Filter.rb:https://github.com/cosmo0920/mikutter_filter/blob/master/filter.rb
MikuTCPConnect:https://github.com/cosmo0920/MikuTCPConnect
これらは全て個人使用の範囲内では自由に改変することができます。
また、Github 上で Fork し、改変することも Github のライセンス条項により許可されます。

mikutter はプラグインで色んな事ができてしまうので事前にソースを読み、どのような動作が行われるかを把握した上で mikutter に導入したほうがいいのは確かです。

次回またあるなら何かネタを用意しとかないとなぁ。

謝辞

filter.rb は@toshi_a さんの pull request のお陰で再起動不要で設定が反映されるようになりました。感謝。

付録

目の保養になどにお使いください。

たのしい mikutter install

@Phenomer

2012-07-10

1 はじめに

一部界隈で何故か導入が難しいと思われているらしい mikutter。本稿ではその導入方法等を複数解説し、「難しい」というイメージの払拭を試みる。

2 Arch Linux で最新の mikutter を使おう

2.1 概要

最近流行りの Linux ディストリビューション、Arch Linux[1]に mikutter を導入する手順を紹介する。

2.2 AUR の (不) 安定版のパッケージを利用する

Arch Linux には既に Arch User Repository(AUR) に mikutter のパッケージが存在する。AUR は、yaourt[2]を用いると簡単に利用できる。残念ながら yaourt は公式リポジトリには存在しない。そこで、yaourt の配布リポジトリである repo.archlinux.fr を/etc/pacman.conf に追加し、yaourt を利用可能にする。

```
[archlinuxfr]
Server = http://repo.archlinux.fr/$arch
```

yaourt のインストールは、いつも通り pacman を用いる。

```
miku@negi% sudo pacman -Sy
miku@negi% sudo pacman -S yaourt
```

後は mikutter を AUR からビルドしインストールして、mikutter を実行するだけである。build の際にウィザード形式でいろいろ聞かれるが、それとなく答えておけば問題ない。自動的に ruby 等の依存関係も yaourt が解決してくれる。

[1] Arch Linux http://www.archlinux.org/ 現在は mikutter の開発も Arch Linux で行われている。パッケージの更新が非常に早く、ローリングリリースが行われている点が特徴であり、ユーザの環境は常に最先端に保たれる。

[2] pacman のフロントエンド。 AUR を用いたパッケージのビルド等も一括して行える。http://archlinux.fr/yaourt-en

```
miku@negi% yaourt -S mikutter
...
==> Edit PKGBUILD ? [Y/n] ("A" to abort)
==> n
...
==> Continue building mikutter ? [Y/n]
==> y

miku@negi% mikutter
```

2.3 開発リポジトリを利用する

mikutter の最新機能を使いたい、またプラグイン開発等を行いたい場合など、最新の mikutter を使いたくなることが多々あると思われる。(mikutter の trunk 追い掛けユーザの割合はかなり高いというウワサもあったりなかったり)

mikutter の開発は、(一応)Subversion にて行われている。[3] そのため、リポジトリから最新のリビジョンを取得する場合は、subversion が必要になる。また、subversion にてソースを入手する場合、依存関係の解決は行われないため、mikutter に必要なライブラリのインストールも行う。[4]

```
miku@negi% sudo pacman -S subversion ruby-gtk2 libnotify
```

subversion をインストールできれば後は簡単。mikutter のリポジトリをチェックアウトするだけである。

```
miku@negi% sudo pacman -S subversion
miku@negi% svn co svn://mikutter.hachune.net/mikutter/trunk mikutter
miku@negi% ruby mikutter/mikutter.rb
```

リポジトリの更新は、svn の update コマンドを用いる。

```
miku@negi% cd mikutter && svn up
```

2.4 開発バージョンのパッケージを作って利用する

Arch Linux で mikutter を動かしているホストが何台もある場合、いちいち手動で subversion で更新するのは面倒な上に、mikutter.hachune.net への負荷も大きくなる。そこで、mikutter の trunk 版パッケージを作成しプライベートリポジトリに登録することで、それを自宅ネットワーク内のホストにて共有可能にする。

[3] 筆者は mikutter 開発者の toshi_a 氏に「subversion はオワコン、git 使ってるよ」と言われふぇぇした記憶がある。

[4] ruby-gtk2 と libnotify の二つをインストールすれば他の依存関係は自動的に解決される。

2.4.1 Web サーバの作成

Web サーバの構築は、ruby が入っているのであれば webrick が最も手っ取り早い。以下は、/var/repo を http://10.39.39.39:3939/ として公開する Web サーバのワンライナーである。[5]

```
miku@negi% sudo mkdir /var/repo
miku@negi% sudo chown miku:miku /var/repo
miku@negi% ruby -r webrick -e "WEBrick::HTTPServer.new(:Port=>3939,
  :DocumentRoot=>'/var/repo').tap{|s| trap(:INT){s.stop}}.start"
```

NGINX 等の Web サーバを利用したほうがよい場合が多いが、ここでは解説の単純さを優先する。

2.4.2 mikutter-svn パッケージの作成

今回は mikutter の subversion パッケージを作成するため、AUR に公開されている mikutter の PKGBUILD はそのまま利用できない。そこで新たに mikutter-svn の PKGBUILD スクリプトを作成する。作成した PKGBUILD スクリプトは以下のようになった。

[5]:DocumentRoot=>Dir.pwd とすればカレントディレクトリを公開する簡易 HTTP サーバとして使える。shell の alias に登録しておくと何かと便利。

```
# Maintainer: Phenomer <phenomer@g.hachune.net>
pkgname=mikutter-svn
pkgver=20120710
pkgrel=1
pkgdesc="a moest twitter client"
arch=('i686' 'x86_64')
url="http://mikutter.hachune.net/"
license=('GPL v3')
depends=('ruby' 'ruby-gtk2' 'ruby-cairo')
makedepends=('subversion')
optdepends=('libnotify')
conflicts=('mikutter')
provides=('mikutter')
install=
_svntrunk=svn://mikutter.hachune.net/mikutter/trunk
_svnmod=mikutter

build() {
  cd "${srcdir}"
  if [ -d "${_svnmod}/.svn" ]; then
      cd "${_svnmod}" && svn up
  else
      svn co "${_svntrunk}" "${_svnmod}"
  fi
}

package() {
  mkdir -p "$pkgdir/usr/bin"
  mkdir -p "${pkgdir}/usr/lib"
  cp -r "${srcdir}/${_svnmod}" "${pkgdir}/usr/lib/mikutter"
  rm -rf "${pkgdir}/usr/lib/mikutter/.svn"

  cat <<EOF > "$pkgdir/usr/bin/mikutter"
#!/bin/sh
ruby /usr/lib/mikutter/mikutter.rb $0
EOF
  chmod a+x "$pkgdir/usr/bin/mikutter"
}
```

これを用いて makepkg を行えば、最新の mikutter-svn パッケージを作成できる。ビルド前には、subversion での利用時同様に依存関係を事前に解決しておく。

```
miku@negi% sudo pacman -S subversion ruby-gtk2 libnotify
miku@negi% ls
PKGBUILD
miku@negi% makepkg
...
==> Finished making: mikutter-svn 811-1 (2012 年 7 月 9 日 月曜日 02:52:48 JST)
```

2.4.3 プライベートリポジトリの作成とパッケージの追加

プライベートリポジトリの作成とパッケージ追加は、repo-add コマンドを用いて簡単に行うことができる。このコマンドにより、自動的にパッケージのデータベースが作成され、クライアントの pacman がリポジトリの状態を参照可能になる。

```
miku@negi% cp mikutter-svn-811-1-x86_64.pkg.tar.xz /var/repo
miku@negi% cd /var/repo
miku@negi% repo-add mikutter.db.tar.xz \
> mikutter-svn-811-1-x86_64.pkg.tar.xz
==> Adding package 'mikutter-svn-811-1-x86_64.pkg.tar.xz'
  -> Computing checksums...
  -> Creating 'desc' db entry...
  -> Creating 'depends' db entry...
==> Creating updated database file 'mikutter.db.tar.xz'
```

2.4.4 リポジトリを利用

yaourt を導入した際と同様に、/etc/pacman.conf に今回作成したリポジトリを追加する。

```
[mikutter]
Server = http://10.39.39.100:3939/
```

これで pacman にて mikutter-svn を利用可能になった。

```
miku@negi% pacman -Sy
miku@negi% pacman -Si mikutter-svn
Repository     : mikutter
Name           : mikutter-svn
Version        : 811-1
URL            : http://mikutter.hachune.net/
Licenses       : GPL v3
Groups         : None
Provides       : mikutter
Depends On     : ruby  ruby-gtk2  ruby-cairo
Optional Deps  : libnotify
Conflicts With : mikutter
Replaces       : None
Download Size  : 1539.11 KiB
Installed Size : 8188.00 KiB
Packager       : Unknown Packager
Architecture   : x86_64
Build Date     : 2012 年 07 月 09 日 02 時 52 分 44 秒
MD5 Sum        : 1813728ee6e9b1c34731854c371477cc
SHA256 Sum     : 4d9c49b4c1a54f17b1407614b8b6ae62
10e56a20c2143c7d76b6753c8107b252
Signatures     : None
Description    : a moest twitter client
```

2.5 ソースから野良ビルド

mikutter が最新でも、ruby や ruby-gtk2 が古くては面白くない。[6] そこで、ruby と ruby-gtk2 をソースからビルドする手順も紹介しておく。これは Arch Linux だけではなく Debian 等でも、パッケージ名を読み替えつつ同様の手順を踏むことで行える。

[6] Arch Linux では最新の安定版が利用できることが多い。

2.5.1 ruby のビルド

以下が ruby の trunk[7]のビルド手順であるが、早速失敗する。

```
miku@negi% svn co http://svn.ruby-lang.org/repos/ruby/trunk ruby
miku@negi% cd ruby
miku@negi% autoconf
miku@negi% ./configure --prefix=/usr/ruby --enable-shared
miku@negi% make
...
executable host ruby is required. use --with-baseruby option.
```

「trunk の ruby をビルドする際は、ruby が必要」らしい。仕方がないので ruby をインストールしやり直す。

```
miku@negi% sudo pacman -S ruby
miku@negi% ./configure --prefix=/usr/ruby --enable-shared
miku@negi% make
...
make: exec(bison) failed (No such file or directory)
```

今度は bison が実行できないと怒られる。これまた仕方がないのでインストールする。

```
miku@negi% sudo pacman -S bison
miku@negi% make && sudo make install
miku@negi% /usr/miku/bin/ruby -v
ruby 2.0.0dev (2012-07-08 trunk 36337) [x86_64-linux]
```

環境によっては、ruby の拡張ライブラリの利用に必要なライブラリやヘッダファイルが不足している場合がある。今回は、openssl と zlib が必須となるが、その他に gdbm や libyaml、readline、ncursesw、tk 等も必要に応じて導入する。

2.5.2 ruby-gtk2 のビルド

ruby-gtk2 のビルドの前に、pkg-config と gtk2 の導入が必要である。[8]

```
miku@negi% pacman -S pkg-config gtk2
```

また、ruby の pkg-config ライブラリも必要になる。これは rubygems を用いてインストールする。[9]

```
miku@negi% /usr/miku/bin/gem install pkg-config
```

ruby-gtk2 のリポジトリ[10]からチェックアウトし、コンフィグを実行する。

[7] Ruby - レポジトリガイドhttp://www.ruby-lang.org/ja/documentation/repository-guide

[8] Arch Linux なら gtk2、Debian 系なら libgtk2.0-dev である。

[9] pkg-config のリポジトリ (https://github.com/rcairo/pkg-config/) からソースを入手しインストールを試みたが失敗するようなので今回は見送った。

[10] SourceForge.net - Ruby-GNOME2 http://sourceforge.net/scm/?type=svn&group_id=53614

```
miku@negi% svn co \
https://ruby-gnome2.svn.sourceforge.net/svnroot/ruby-gnome2/
ruby-gnome2/trunk ruby-gnome2
miku@negi% cd ruby-gnome2
miku@negi% /usr/miku/bin/ruby extconf.rb
...
-----
Target libraries: glib2, gio2, gdk_pixbuf2, pango, atk, gtk2
Ignored libraries: gstreamer, gtksourceview2, rsvg2, poppler,
goocanvas, vte
-----
Done.
```

いくつかのパッケージの config に失敗するが、ruby-gtk2 としての動作に支障は無いため今回は無視して make。

```
miku@negi% make
...
エラー: expected expression before '/' token
```

が、いきなりエラー多発。なにやら C++ライクなコメント (//) が全て不正扱いされエラーになっている模様。

```
% vi glib2/ext/glib2/rbgobj_valuetypes.c
% vi glib2/ext/glib2/rbgobj_param.c
% vi glib2/ext/glib2/rbgobj_type.c
% vi glib2/ext/glib2/rbglib_keyfile.c
% vi glib2/ext/glib2/glib-enum-types.c
% vi glib2/ext/glib2/rbgobj_signal.c
% vi glib2/ext/glib2/rbgobj_object.c
% vi gdk_pixbuf2/ext/gdk_pixbuf2/rbgdk-pixdata.c
% vi gdk_pixbuf2/ext/gdk_pixbuf2/rbgdk-pixbuf.c
% vi gtk2/ext/gtk2/rbgtkaccellabel.c
% vi gtk2/ext/gtk2/rbgtkwidget.c
% vi gtk2/ext/gtk2/rbgtkcontainer.c
```

その他にも glib.h のインクルードのエラー等、細かな修正が必要な箇所があった。

```
% vi glib2/ext/glib2/rbglib_win32.c
/* #include <glib/gwin32.h> */
#include<glib.h>
% vi glib2/ext/glib2/rbglib_int64.c
/* #include <glib/gtypes.h>*/
#include <glib.h>
```

何十回と make を繰り返し、ようやく全てのビルドが完了した。

```
miku@negi% sudo make install
-----
SUCCEEDED:  glib2 gio2 gdk_pixbuf2 pango atk gtk2
FAILED: NONE
-----
Done.
```

2.5.3 rcairo のビルド

最後にもうひとつ、rcairo が必要である。

```
miku@negi% git clone https://github.com/rcairo/rcairo
miku@negi% cd rcairo
miku@negi% /usr/miku/bin/ruby extconf.rb
miku@negi% make && sudo make install
```

これで、mikutter の動作に必要なライブラリは全て揃った。

2.5.4 動作確認

今回作成した環境では、mikutter は AOuth 認証画面まで正しく行うことができたものの、残念ながら gdk と rcairo のエラーにより、タイムラインの表示を行うことができなかった。ruby の trunk は比較的普通に利用できるが、ruby-gtk2 の trunk を利用するのは、gtk2 との関係もあるためかなかなか難しいようだ。[11]

3 LTSP を用いた mikutter 環境展開システムの構築

3.1 概要

大学の研究室等で mikutter を広めたい、しかし学生は皆 Windows ユーザー。そんな状況を打破するべく、LTSP[12]を用いた mikutter 環境展開システムの構築方法を紹介する。

3.2 クライアント環境

PXE boot に対応したネットワークインターフェースを持ち、Debian wheezy(i386)[13]が正常に動作するコンピュータを想定する。BIOS のブート順の設定にて、PXEboot(ネットワークブート) を最優先にしておく必要がある。

3.3 サーバ環境

物理ネットワークインターフェースが 2 ポート利用できるサーバを準備する。今回は、ネットワークインターフェース (RTL8169、いわゆる蟹) が 3 ポート存在するはつねサーバを利用し、その上に Xen DOMU を作成し Debian wheezy(x86_64) をインストールした。勿論仮想化を行わず直接 Debian をインストールしてもよい。また、サーバは Debian に限らず Ubuntu 等でも同様の手順にて構築できる。[14]

[11]筆者は ruby と mikutter は Subversion にて取得したものを用い、ruby-gtk2 と rcairo は rubygems にてインストールしたものを用いている。ruby-gtk2 と rcairo の gem パッケージ (gtk2, cairo) はディストリビューションに依存しないため、常に最新安定版を利用できる。

[12]LTSP(Linux Terminal Server Project) とは、Linux を用いた容易に利用できるシンクライアントシステムのサポートを目指したプロジェクトである。

[13]Debian http://www.debian.org/

[14]Ubuntu の場合、インストーラの時点で LTSP の設定を行うことが可能である。https://help.ubuntu.com/community/UbuntuLTSP/LTSPQuickInstall

```
name="mikan"
vcpus=2
memory=1024
vif=['mac=00:16:3e:39:13:30,bridge=bridge0',
     'mac=00:16:3e:39:13:31,bridge=bridge1']
disk=['phy:/dev/mapper/hvg0-Mikan,sda,w']
kernel="/home/miku/vm/bzImage"
root='/dev/xvda ro'
```

ネットワークインターフェースは、eth0 が外向け、eth1 が内部向け (クライアント側) になる。また、eth0 には 10.39.39.100/24、eth1 には 192.168.67.1/24 がそれぞれ割り当てられる。

3.4 サーバ環境構築

LTSP サーバ上にて、クライアントが mikutter を起動できる状態になるまでの設定を行う。

3.4.1 ネットワークの設定

初めに、今回の環境に合わせネットワークインターフェースの設定を行う。Debian では/etc/network/interfaces にネットワーク設定を記入する。

```
auto lo eth0 eth1
iface lo inet loopback

iface eth0 inet static
 address 10.39.39.102
 netmask 255.255.255.0
 gateway 10.39.39.1
 dns-domain k.hachune.net
 dns-nameservers 10.39.39.1

iface eth1 inet static
 address 192.168.67.1
 netmask 255.255.255.0
```

networking の restart は deprecated となっているので、一旦 reboot を行う。また、iptables 等の設定は ltsp-server-standalone のインストール時に既に行われる。

3.5 LTSP パッケージの導入

次に LTSP サーバをインストールする。この際、クライアントが利用するデスクトップ環境も同時にインストールを行う。デフォルトは gnome であるが、今回は軽量さを重視して awesome を利用する。

```
miku@mikan% sudo aptitude install ltsp-server-standalone awesome
```

3.5.1 dhcpd の設定

クライアントは PXE boot と NFS root で起動される。まず、/etc/ltsp/dhcpd.conf を編集する。/etc/dhcp/dhcpd.conf ではないので注意が必要である。

```
authoritative;

subnet 192.168.67.0 netmask 255.255.255.0 {
    range 192.168.67.20 192.168.67.250;
    option domain-name "ltsp.k.hachune.net";
    option domain-name-servers 192.168.67.1;
    option broadcast-address 192.168.67.255;
    option routers 192.168.67.1;
    next-server 192.168.67.1;
#    get-lease-hostnames true;
    option subnet-mask 255.255.255.0;
    option root-path "/opt/ltsp/i386";
    if substring( option vendor-class-identifier, 0, 9 ) = "PXEClient" {
        filename "/ltsp/i386/pxelinux.0";
    } else {
        filename "/ltsp/i386/nbi.img";
    }
}
```

また、/etc/default/isc-dhcp-server の設定も必要である。ここでは、DHCPD_CONF と INTERFACES の行を環境に合わせ修正する。

```
DHCPD_CONF=/etc/ltsp/dhcpd.conf
INTERFACES="eth1"
```

設定が完了したら、dhcpd の再起動を行う。

```
miku@mikan% sudo service isc-dhcp-server restart
```

3.5.2 NFS の設定

次に、NFS の設定を行う。今回は/opt/ltsp をクライアントのルートディレクトリとして 192.168.67/24 のネットワークに公開するため、/etc/exports に以下の行を追加する。

```
/opt/ltsp 192.168.67.0/24(ro,no_root_squash,async,no_subtree_check)
```

設定が完了したら、NFS 関連サービスを再起動する。

```
miku@mikan% sudo service nfs-common restart
miku@mikan% sudo service nfs-kernel-server restart
```

3.5.3 tftpd の設定

クライアントの起動に必要なカーネルや初期イメージは、dhcp により指定され tftp により転送される。tftp の設定は、LTSP インストール時に既に行われている為必要ないが、もし修正が必要になった場合は/etc/defaults/tftpd-hpa を参照する。

3.5.4 クライアントブート環境の作成

/opt/ltsp 以下に展開される、クライアントブート環境を作成する。ltsp-build-client コマンドを用いると、指定したアーキテクチャのイメージを自動的に作成できる。クライアントのアーキテクチャに x86_64 を指定することもできるが、ここでは設定ファイルにデフォルトで記入されている i386 を用いる。

```
miku@mikan% sudo ltsp-build-client --arch=i386
```

ディストリビューションを一から構築する為かなり時間がかかる場合があるが、じっくり待つ。

3.5.5 クライアントの起動

クライアント側となる PC の PXE boot 機能を BIOS にて有効にし、eth1 側のネットワークに接続、起動する。PXE boot が一番最初になるよう Boot 順に注意する。

3.5.6 デスクトップ環境と mikutter の設定

クライアント環境 (/opt/ltsp 以下) は、クライアントを動作させるための限られた用途に用いられる。一見クライアント環境はクライアント上で動いているように見えるが、実体はサーバ上にあり、サーバのリソースを各クライアントにて共有する。そのため、クライアントでユーザが利用するソフトウェアパッケージはサーバ側に導入する。

mikutter を利用するには、サーバ側に最低限以下のパッケージを導入する必要がある。

- ruby
- ruby-gtk2
- libnotify-bin

また、以下のパッケージを導入することでより華麗に mikutter を利用できる。

- 日本語フォント (fonts-vlgothic 等)
- 日本語インプットメソッド (ibus-mozc 等)
- subversion (trunk 追い掛け用)
- 仮想端末 (sakura 等)

その他にも、いくつかの設定が必要になるかもしれない。

- locales の再設定
- tzdata の再設定

4 あとがき

本稿ではひたすらに mikutter を利用できる環境の構築について解説を行った。本稿作成の作業を元に、近いうちに mikutter 用の Arch リポジトリを hachune.net にて公開したいと考えている。しかし、諸事情により hachune.net のサーバをリプレースが必要で、少々時間が掛かりそうだ。

5 参考文献

- PKGBUILD - ArchWiki
  https://wiki.archlinux.org/index.php/PKGBUILD
- VCS PKGBUILD Guidelines - ArchWiki
  https://wiki.archlinux.org/index.php/VCS_PKGBUILD_Guidelines
- AUR(en) - mikutter
  http://aur.archlinux.org/packages.php?ID=47755
- LTSP Manual
  https://sourceforge.net/projects/ltsp/files/Docs-Admin-Guide/LTSPManual.pdf

1 プラグイン開発のベストプラクティス

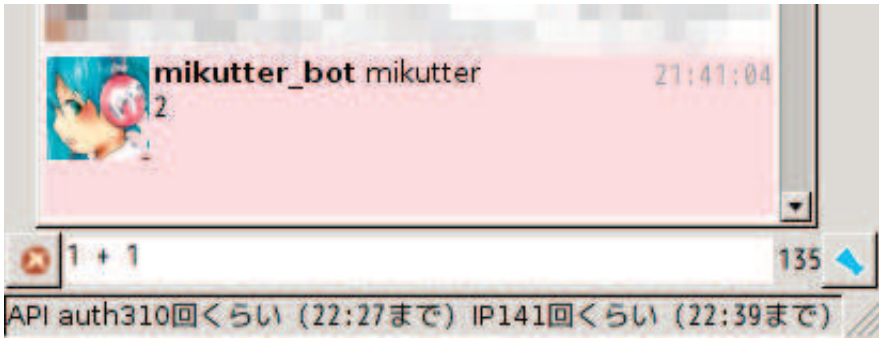
mikutterプラグインを作ってみたいけど、どうやって環境を作るのか。コードさえ書ければいいので、何も言われなくても調べればだいたいみんなできるのですが、今回は開発中であったりドキュメント化されていない等の理由でなかなか見つけづらい開発用の機能やツールをいくつか紹介したいと思います。

1.1 mikutter自体の機能

mikutterには八百万の隠し機能があると言われていますが、開発用の隠し機能もいくつかあります。

1.1.1 Rubyコンソール

Rubyの良いところのひとつとして、インタプリタに気軽にコードを入力して実行できることが挙げられます。しかしそのコードが依存するライブラリがあったら、当然ロードしておかなければいけません。mikutterでそれをするのは無茶でしょう。そこでGUIプラグインは簡素ではありますがRubyコンソールを提供しています。ALT+Xを押してみてください。



図のように、ステータスバーのすぐ上にツイート入力画面のようなものが出てきます。ここにRubyコードを入力すると、その実行結果がアクティビティに表示されます。

簡単なプラグインは、mikutterを再起動せずともここにコピペして投稿ボタンを押せばプラグインをインストールできてしまいます。地味な機能ですが、知っているとmikutterの開発効率が上がることは間違い無いです。

1.1.2 デバッグモード

mikutterプラグインを書いて一番よくあるのは、mikutter自体をクラッシュさせるということです。この時、エラーはコンソールに出力されず、次回起動時にバグ報告をしようとします。この挙動は、プラグインを開発するときには邪魔になるので、以下のようなコマンドでmikutterを起動しましょう

```
$ ruby mikutter.rb --debug
```

--debug オプションをつけて起動すると、mikutterはデバッグモードで起動します。ターミナルには大量のメッセージが表示されていると思います。これはコアに仕込まれている特定のメソッドを呼び出した時に出力されるものです。具体的にデバッグモードではどうい

った違いがあるのかを見ていきたいと思います。

- クラッシュ時にバックトレースを表示する
  現在はクラッシュしたときにバックトレースを表示しませんが、デバッグモードではそれを表示します。

- notice　警告・エラーメッセージを表示
  printデバッグは、よく使われるデバッグ手法の一つです。Rubyでは **p** メソッドなどがよく使われます。mikutterでも当然これを利用することはできますが、mikutter自体が **notice**, **warn**, **error** というメソッドを提供しているので、そちらを使うこともできます。

```
notice "情報"
warn "警告"
error "エラー"
```

  それぞれメソッド名の通り、noticeが一般的な情報、warnは警告、errorはエラーメッセージを表示するために使います。これらの **p** とのちがいは、すべて標準エラー出力にメッセージが出力されること、デバッグモードでないと実際の表示がされないので、異常系には出力を入れたままでリリースすることができるという点です。一方、第一引数を文字列としてそのまま表示してしまうので、文字列に変換できないオブジェクトには向きません。例外として、Exceptionのサブクラスのインスタンスが渡された場合は、バックトレースを表示します。例えば、以下のコードをコンソールに入力して実行してみましょう。

```
begin
  raise "error test"
rescue => e
  error e
end
```

  以下のような出力が得られるはずです。

```
error: (eval):4:in `rescue in post': error test
from (eval):2:in `post'
from {MIKUTTER_DIR}/core/plugin/gui.rb:326:in `eval'
from {MIKUTTER_DIR}/core/plugin/gui.rb:326:in `post'
from {MIKUTTER_DIR}/core/mui/gtk_postbox.rb:142:in `post_it'
from {MIKUTTER_DIR}/core/addon/addon.rb:94:in `call'
from {MIKUTTER_DIR}/core/addon/addon.rb:94:in `block in call_keypress_event'
from {MIKUTTER_DIR}/core/addon/addon.rb:87:in `each'
from {MIKUTTER_DIR}/core/addon/addon.rb:87:in `call_keypress_event'
from {MIKUTTER_DIR}/core/mui/gtk_postbox.rb:70:in `block in widget_post'
from {MIKUTTER_DIR}/core/mui/gtk_extension.rb:37:in `call'
from {MIKUTTER_DIR}/core/mui/gtk_extension.rb:37:in `block in __track'
from mikutter.rb:63:in `call'
from mikutter.rb:63:in `main'
from mikutter.rb:63:in `boot!'
from mikutter.rb:81:in `<main>'
```

1.1.3 pryによるクラッシュ時のオブジェクト参照

またもやクラッシュした時のお話です。クラッシュした時に各変数がどういった値を保持しているか、オブジェクトがどのような状態になっているかを知る必要があります。こういった用途にはデバッガを使うのですが、最終テストがてらTwitterをだらだら楽しみたい時にデバッガを起動してしまうと、動作が遅くなってしまい流れに乗り遅れ、ふぁぼを逃す可能性があります。

- pryとは
  irbのようなインタプリタです。gemでインストールできます。

```
$ gem install pry
$ pry
[1] pry(main)>
```

  pryコマンドでインタプリタが立ち上がります。基本的にはirbと同じように使えるのですが、ここで例えばlsとか入力すると

```
[1] pry(main)> ls
self.methods: include  private  public  to_s
locals: _  _dir_  _ex_  _file_  _in_  _out_  _pry_  version
[2] pry(main)> a = [0, 1, 2]
=> [0, 1, 2]
[3] pry(main)> ls
self.methods: include  private  public  to_s
locals: _  _dir_  _ex_  _file_  _in_  _out_  _pry_  a  version
[4] pry(main)>
```

  と、unixのlsコマンドのように、現在のスコープから見える変数とメソッドが一覧できます。またcdコマンドでどのオブジェクトのスコープでコードを実行するか指定できます。例えば変数aに入ってる配列の中に入ってみましょう。

```
[6] pry(main)> cd a
[7] pry(#<Array>):1> ls
Enumerable#methods: all?  any?  chunk  collect_concat  detect  each_cons  each_entry
each_slice  each_with_index  each_with_object  entries  find  find_all  flat_map  grep
group_by  inject  max  max_by  member?  min  min_by  minmax  minmax_by  none?  one?
partition  reduce  slice_before  sort_by
Array#methods: &  *  +  -  <<  <=>  ==  []  []=  assoc  at  clear  collect  collect!
combination  compact  compact!  concat  count  cycle  delete  delete_at  delete_if
drop  drop_while  each  each_index  empty?  eql?  fetch  fill  find_index  first
flatten  flatten!  frozen?  hash  include?  index  insert  inspect  join  keep_if
last  length  map  map!  pack  permutation  pop  pretty_print  pretty_print_cycle
product  push  rassoc  reject  reject!  repeated_combination  repeated_permutation
replace  reverse  reverse!  reverse_each  rindex  rotate  rotate!  sample  select
select!  shelljoin  shift  shuffle  shuffle!  size  slice  slice!  sort  sort!
sort_by!  take  take_while  to_a  to_ary  to_s  transpose  uniq  uniq!  unshift
values_at  zip  |
self.methods: __binding_impl__
locals: _  _dir_  _ex_  _file_  _in_  _out_  _pry_
[8] pry(#<Array>):1>
```

と、メソッドとそのスコープの変数が見えます。これがmikutterと何の関係があるかというと、実はmikutterはデバッグモード時にクラッシュすると、もしあればクラッシュする前にpryを立ち上げるようになっています。またプラグインからinto_debug_modeメソッドを呼び出せば、いつでもpryが（あれば）起動できます。では、実際に使ってみましょう。以下のようなプラグインを書いたとします。

```
Plugin.create :crash do
  onupdate do |messages|
    messages.each do |m|
      notice m.to_s
    end
  end
end
```

一般情報としてホームタイムラインのツイート内容を印字するプラグインに見えます。実行してみましょう。

```
error: {MIKUTTER_DIR}/core/plugin.rb:219:in `rescue in block in event_wrap': undefined
method `each' for #<MikuTwitter:0x00000003448628>
from {MIKUTTER_DIR}/core/service.rb:180:in `method_missing'
from /home/toshi/.mikutter/plugin/crash.rb:4:in `block (2 levels) in <top (required)>'
from {MIKUTTER_DIR}/core/plugin.rb:217:in `call'
(中略)
from mikutter.rb:81:in `<main>'

From: /home/toshi/Documents/hobby/scripts/mikutter/trunk/core/plugin.rb @ line 214
Plugin.event_wrap:

    214:     def event_wrap(&callback)
    215:       lambda{ |*args|
    216:         begin
    217:           callback.call(*args, &@plugin_exit)
    218:         rescue Exception => e
 => 219:           into_debug_mode(e, binding)
    220:           raise e end } end


[1] pry(Plugin)>
```

クラッシュしてバックトレースが表示されました。一行目にエラーの原因が出ていますが、一体何が悪いんでしょうか。この時、一番下を見るとPryプロンプトが立ち上がってますね。ちょっと様子を見てみましょう。

```
[1] pry(Plugin)> ls
constants: Activity  DirectMessage  ExecuteBox  Executer  GUI  Setting  Streaming
Object.methods: defun  yaml_tag
Plugin.methods: abort_on_exception?  activity  add_event  add_event_filter
add_event_hook  boot_plugin  call  (略)

Plugin#methods: active!  active?  add_event  add_event_filter  add_event_hook  at
create  detach  (略)
```

```
class variables: @@add_event_hook  @@event  @@event_filter  @@eventqueue  @@plugins
locals: _  _dir_  _ex_  _file_  _in_  _out_  _pry_  args  callback  e
[2] pry(Plugin)> self
=> Plugin
[3] pry(Plugin)>
```

ここはPluginクラス自身ですね。残念ながらイベントのコールバックのスコープではないです。実はイベントのコールバックでエラーが起こると必ずここでPryが呼び出されます。コードを見ても明らかな通り、３つの変数が定義されていますね。

変数	意味
args	イベントコールバックの引数(配列)
callback	コールバック自身(Proc)
e	例外オブジェクト

onupdateに渡される引数を見てみます。

```
[4] pry(Plugin)> args
=> [#<Service toshi_a>, [(ツイート)...]]
[5] pry(Plugin)>
```

ああ、onupdateは、第一引数がService、第二引数にツイートが入ってくるんですね。Ctrl-Dを入力してmikutterを一度終了して、onupdateの引数を直して以下のように修正して再起動すると、ターミナルにツイートが次々と表示されると思います。

```
Plugin.create :crash do
  onupdate do |service, messages|
    messages.each do |m|
      notice m.to_s
    end
  end
end
```

Pryは入れておくだけでmikutterから利用できるので、プラグインを開発するなら入れておいて損はないでしょう。

1.1.4 プラグイン開発用のmikutterを作る

mikutterは、できるだけ機能をプラグインとして実装するために、標準機能もほとんどがプラグインでできています。例えば、GUI部分でさえプラグインになっているほどで、コアが実際にやっていることでユーザが意識するようなことは、TwitterAPIの管理くらいなものです。ただし、毎分TwitterAPIをクロールするのはrestプラグイン、UserStreamはstreamingプラグインが担当しており、あくまでAPIのアクセス手段を提供するまでに留められています。mikutterプラグインを開発するとなると、何度もmikutterを起動することになると思いますが、mikutterを毎回起動するのはかなりの負担になりますし、APIのレートリミットに引っかかる場合もあります。しかし、開発中のプラグインとは直接関係がないコアプラグインを全て削除すれば、開発用の軽量な環境を手軽に作ることができま

す。

- コアプラグイン
では、実際にコアプラグインを見てみましょう。プラグインのほとんどは、**/core/addon** の中に入っています。ただし、guiプラグインだけは **/core/plugin** に入っています。これは将来のバージョンで全て*/core/plugin* に統一される予定です。ではaddonディレクトリを見てみましょう。

プラグイン名	効果
activity	アクティビティ
alsa	通知サウンド
api_request_file_cache	TwitterAPIキャッシュ
aspectframe	？？？
bitly	URL短縮サービスbit.lyのURL展開
bugreport	バグ報告
change_account	アカウント変更
contextmenu	主要なmikutterコマンド
directmessage	DM関連
extract	抽出タブ
followingcontrol	フォロイー／フォロワー関連
friend_timeline	ホームTL
image_file_cache	画像ファイルキャッシュ
libnotify	通知をlibnotifyで表示する
list	リスト
mentions	メンション・リプライ
notify	通知
openimg	画像プレビュー
profile	ユーザのプロフィール
proxy	プロキシ
rest	REST API。毎分のTL更新等
search	検索機能
set_input	設定の「入力」
set_view	設定の「表示」
settings	設定タブ
shortcutkey	ショートカットキー機能

smartthread	会話スレッド
streaming	UserStreamとリストStreaming機能
tco	短縮URL、t.co展開

と、実に29個のプラグインが最初から入っています。また、pluginディレクトリには以下のプラグインが入っています。

プラグイン名	効果
core	コア機能。削除すべきではない
gui	ウィンドウを表示

コアプラグインとはいえ、所詮プラグインです。取り外しても動きます。実際にやってみましょう。

- プラグインを削除しよう
/core/addonで rm -rf * したくなるとおもいますが、注意すべきことが一つ有ります。addon.rbというファイルを削除すると一部プラグインが動かなくなります。あと、単純に削除すると、意外と機能が消えるのでプラグイン作成どころではなくなることがほとんどです。失って初めてわかることもあるのです。というわけで、addonディレクトリをリネームし、新しくaddonを作成し、その中にaddon/addon.rbをコピーしてください。これも、現在開発中のバージョンではaddon.rbを削除することになっているので、addon.rbが見当たらない場合は細かいことは考えずに一思いに消してしまって問題ないです。

- 何もないmikutterの世界
では早速起動してみましょう。

ツイート投稿スペースがあるだけの残念なウィンドウが表示されます。ここから俺たちのTwitterクライアントは始まる…。いつか必ずNo1のTwitterクライアントを作って、Twitter社に40億ドルで買収されるんだ。そんな決意に満ち溢れる画面です。起動は当然早くなっているし、APIも多くて一度しかリクエストしません。ただ、これでは本当に何もできないので、削除したプラグインの中から **friend_timeline** と **rest** プラグインを元のaddonディレクトリに戻してもう一度起動してみてください。ホームタイムラインが表示され、数分に一度タイムラインが更新されるという、2008年を彷彿とさせるTwitterクライアントになると思います。

- 自分のプラグインを入れる
  ここまできたらあとは自分で作ったプラグインを **~/.mikutter/plugin** 以下に書いてくだけです。こういった環境を作るメリットは、先に述べたように軽量になるのでテストしやすいということと、関係のないプラグインがないことでバグの特定が容易になるということがあります。このテクニックは、自分が普段使うmikutterから不要なコア機能を削除したい場合にも使えます。

1.2 mikutter-mode

コードを書くにはエディタも大事です。mikutterプラグインはRubyのコードなので手慣れたエディタを使うことができます。しかし最近、mikutterのコードが整理されてきて、エディタにmikutterを書くための支援機能をつけることが可能になって来ました。ここではコアの開発に主に使われているエディタ、Emacsのために書かれたmikutter-modeを簡単に紹介します。

https://github.com/toshia/mikutter-mode

1.2.1 i-menu

i-menuは、一般的なIDEのようなクラスや関数のリストを表示するためのelispです。mikutter-modeは、

- プラグインの定義開始位置
- イベント、フィルタのフック

を、i-menuに表示する機能があります。たくさんのイベントをフックするときは、i-menuから選択してジャンプできるので重宝するでしょう。

1.2.2 yasnippet

yasnippetは、特定のモードの時に特定の単語の後ろで設定されたキーを叩くとスニペットが展開されるというものです。mikutter-modeには、それが有効なときにだけ動作するスニペットがいくつか提供されています。例えば、プラグインディレクトリに空のファイルを作成して以下のようにタイプし、

```
ptl
```

lの後ろで展開キーを押すと

```
Plugin.create : do

end
```

と、プラグインのスケルトンに展開されます。コロンの後ろにカーソルが移動するので、スラッグを入力してタブキーを押すと二行目にカーソルが飛び、展開終了です。こういったスニペットは、大量に提供されていると覚えるほうが大変なので、mikutter-modeに収録されているのはごく僅かです。みなさんもよくあるパターンをスニペット化してみてはいかがでしょうか。

1.2.3 mikutter_mode.rb

最後に、実験的なものではありますが、mikutter_mode.rbの紹介です。mikutter-modeに付属するプラグインで、mikutterプラグインの実行をmikutter本体の再起動無しで実現しようというものです。

- インストール
  mikutter-modeのリポジトリの plugin/ 以下にある mikutter_mode ディレクトリを ~/.mikutter/plugin にコピーします。また、dbus-rubyをgemでインストールしてください。

  ```
  $ gem install dbus-ruby
  ```

- 使ってみる
  まず、mikutter_mode.rbをインストールしたmikutterをデバッグモードで起動します。そして ~/.mikutter/plugin/ に、適当なファイル（ここでは、test.rb）を作成します。正常にmikutter-modeがインストールされているなら、以下の写真のように、モードラインに **Mikutter** と表示されているはずです。

さて、このプラグインは **test** というイベントが発生すると、「テスト」というnoticeメッセージを表示します。実際にやってみるには再起動かプラグインをコンソールにコピペなのですが、Emacsとmikutterにmikutterがインストールされていれば、バッファでC-c C-c とタイプするだけでこのプラグインをインストールすることができます。

```
notice: /home/toshi/.mikutter/plugin/mikutter_mode/mikutter_mode.rb:21:in `block (2
levels) in <class:Server>': ruby code execute:
org.mikutter.eval
notice: /home/toshi/.mikutter/plugin/mikutter_mode/mikutter_mode.rb:23:in `block (2
levels) in <class:Server>': returns:
test
notice: /home/toshi/.mikutter/plugin/mikutter_mode/mikutter_mode.rb:21:in `block (2
levels) in <class:Server>': ruby code execute:
/home/toshi/.mikutter/plugin/test.rb
notice: /home/toshi/.mikutter/plugin/mikutter_mode/mikutter_mode.rb:23:in `block (2
levels) in <class:Server>': returns:
test
```

上記のようなメッセージがターミナルに表示されると思います。次に実際にtestイベントを発生させます。コンソールから以下のコードを実行しましょう。

```
Plugin.call(:test)
```

以下のような出力が得られます。再起動無しにmikutterプラグインをインストールできました。

```
notice: /home/toshi/.mikutter/plugin/test.rb:6:in `block (4 levels) in
<class:Server>': テストです
```

- プラグインの自動再ロード
  test.rb にあった **onunload** について説明しておきます。mikutter-modeを使ってプラグインをロードする時、すでにそのプラグインがロードされている時があります。その場合、一度プラグインが取り外され、新しいプラグインがロードされます。こういう、mikutterの終了以外の要因でプラグインが取り外される時に **onunload** ブロックが呼ばれるのです。ためしにもう一度 C-c C-c とタイプしてみましょう。以

下のように出力されると思います。

```
notice: /home/toshi/.mikutter/plugin/test.rb:9:in `block (4 levels) in
<class:Server>': アンロード
```

onunload の主な使い道は、ThreadやGtk等、プラグインが取り外されても解放されないオブジェクトを解放する、デストラクタ的な処理を書くためです。

- mikutter_mode.rbの今後
  mikutter_modeは最近やっと本格的に開発に着手されつつあるまだ発展途上の機能ですが、Emacs上でEmacs Lispを書くようにmikutterプラグインが書けるようになるポテンシャルを秘めたプラグインです。最新の情報は、上に書いたgithubのページを見てください。現在はEmacsの拡張機能のみ配布していますが、DBusが使えるなら他のエディタ・IDEでも同様のことができるはずです。

2 まとめ

前回、mikutterの薄い本vol.1に「Writing mikutter plugin」を寄稿させていただき、Web上でも公開しました。想定した読者はmikutterにちょっと機能を足したい人でした。この試みはある程度成功し、プラグインを作る人の入門として読まれるようになりました。一方、ある程度プラグインを書くことが日常的になってくると、mikutter固有のデバッグの方法がまとまって公開されていないため、一歩踏み込んだプラグインを書こうとした時に負担になっているケースが散見されました。そのような経緯からプラグイン開発のベストプラクティスを紹介するドキュメントの必要性を感じ、今回このドキュメントを書くに至りました。皆さんがmikutterを自分色に染める一助になれば幸いです。

巻末　と　あとがき　@brsywe

前回 24 ページで今回 60 ページになった mikutter の薄い本。おたのしみ頂けましたでしょうか。　前回は様子見だった人が沢山書いて下さったこともあり、ページ数が一気に増えましたね。

思えば半年前。こみトレで頒布したときには mikutter を全く知らないけれど受け取ったと云う人も居ました。いまごろ twitter 広い海のどこかでておくれているのかしら。

あなたがたのておくれライフの一助となることを期待しつつ。

↑@shijin_cmpb

――次号予告――

原稿提出期限：11 月末日

発行予定：C83（12 月末@東京）　または　こみっくトレジャー21（1 月中旬@大阪）

問合せ先： @brsywe

奥付

発行日：2012年9月7日 初版第2刷/PDF版 OSC TOKYO　(初版第1刷：8月4日 OSC KYOTO)

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･頒布協力：IUJK

ご意見・ご感想はお気軽にどうぞ。twitter ハッシュタグ #すごい M 本

mikutter の薄い本制作委員会ウェブページ（右の QR コードも同じ）
http://home1.tigers-net.com/brsywe/mikutter.html

追補（mikutter の薄い本 vol.2 初版第 1・2 刷/PDF 版共通）

以下の通り誤植がございますので訂正を御願い致します。

【44 ページ　2.5.4】

×AOuth ○OAuth

mikutter の薄い本制作委員会では、Amazon 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。