mikutterの薄い本 vol.3

人類(じんるい)は手遅(ておく)れました

Release: 31 December 2012
chief:@brsywe , @ch_print

（PDF 版　画像付録）

上：トートバッグデザイン

下：IC カードステッカーデザイン

mikutter 徹底入門

編者　@brsywe
西端の放送局内喫茶室長
（執筆開始日：平成 24 年 12 月 12 日）

そもそも　とは

なんでこの本を手に取った？

What is “mikutter の薄い本”

こうしてわたしが、mikutter の薄い本の冒頭に阿呆なことを書くのも三回目となった。先ず、mikutter の薄い本とはなんたるか、概論について整理し考察したい。

vol.1(平成 24 年 1 月刊行)
「mikutter の歴史、将来、mikutter への意見、ておくれと mikutter のカンケイ、としぁがアへる漫画載せ、mikutter のておくれなハッテンに資する のが本来の目的だったような。」

vol.2 (同年 8 月刊行)
「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。」

――我ながら尤もらしいことを述べていた。
かくいうわたしも、Twitterを初めて2年、mikutterを使い始めて1年半である。この薄い本が世に出回るときにはmikutterは3歳である。みくったーちゃんは18歳みたいだけどね！！（ @soramame_bscl　の記事を参照されたい。）　誰か3歳児で、園児服着たみくったーちゃんをぺろぺろする薄い本書いておくれ。vol.4に載せたい。
ておくれたちが提出期限を定められててんやわんやしながら記事を書き、けれど間に合わずに已むなく締切を延長し、そのうちで集められたものを載せる。それこそがmikutterの薄い本なのだっ！！！（ｷﾏｯﾀｧ

コラム――小さい記事

1. 「mikutter とわたし」作文コンクール。あなたが mikutter を使い始めたきっかけは？　と云う題で募集していた。ここで最優秀賞の発表。

   @MORIMOTOA

   「その前にみくったーのせいでツイッターはじめるはめになったことから」

   最高にておい。最優秀賞に選ばれた方には賞品の 100 ふぁぼを御送りいたします。

2. おくれアイコン職人として名高い @catina013 にインタビューしてきた。

   ――dropbox 上で。

Q1.自己紹介をしてくだしあ。

一応はじめまして。

mikutter の薄い本 Vol.2 にも投稿させていただきました　めびうす(永遠の年男)です。おくれアイコンを 15 分で製作できる程度の能力です。Linux とか　もっというとプログラムとはまったく関係ない WindowsXP で、これまで 10 年弱過ごしてきた人間です。一応 Ubuntu マシンも持ってますが性能が良くないので埃被ってますね。

Twitter 暦は全然長く無い新参者ですね。ネット暦自体はソコソコ長いんですけどね。

Q2.ておくれを知ったきっかけを。

たしか…知らない間に室長氏にフォローされていて(恐らく大隊長あたり繋がり)フォローを返したのが最初の気がします。そのあと室長氏の RT 出てくる各人をフォローしていって、そのうちとしぁ神を知りフォローして　ておくれに触れた感じだったと思います。

Q3.初めて作ったておくれアイコンは？

はじめてのておくれアイコンは、ペンギンさんのておくれアイコン一覧から改変して”としぁへ顔”ってアイコン作ったのが最初だった気がします。線画から全部自作したのはえりつぃん氏のパクリアイコンを作ったときですね。ちょっと特殊なアイコンだったので、初めのって云っていいのかわかりませんが。その後　数点作り”ておキュアシリーズ”を作る過程で、作業の効率化から今のておくれアイコンのベースが完成しました。

Q4.依頼のあったておくれアイコンを作り始めた経緯は？　いままでにつくったておくれアイコンの数は？

室長氏より「はぴねす！の小日向すもも」のアイコンを作ってくれと依頼されたのが最初です。今まで東方・プリ○ュアシリーズだったので初エロゲ原作のアイコンでもあります。11 月 12 日現在で Ver 違いも含めて 90 個製作しています。自分で集計したんですが自分が一番驚いてます。…何時間費やしてるんだ俺

Q5.あなたにとっての「ておくれ」とは？

「ておくれ」とは、

・○○したから「ておくれ」だ

とか

・あの人は○○使ってるから「ておくれ」だ

なんて簡単なくくりが出来ないものだと思います。

上手く云えないですが、「ておくれ」には「ておくれ」のオーラがあって本物の「ておくれ」に触れたとき、心の奥底から『あぁ…この人ておくれだ』と感じるのでその感覚に近いものが「ておくれ」だと思います。少なくとも自称するものじゃ無いのは断言できます。

まさかわたしのせいで @catina013 がておくれていようとは。どうもわたしも何人かのておくれ感染源になっている。わたしはておくれではないが、感染源となっていることについて十分に反省しなければならない。

今号の紹介

さて、本題に入ろう。今回 vol.3 の副題は『人類は手遅れました』となった。どっかのアニメでみたようなタイトルだな、おい。

生粋のておくれ複数人に依頼するなりマグカップ与えて脅すなりして発行出来る十分な分量の記事が集まった。短くだが紹介したい。

＜目次＞　気が向いたら執筆者などをﾌｫﾛｰしてみてもいいとおもうの。

イラストを寄稿してくださった人の紹介

表紙:@soramame_bscl(左) , @shijin_cmpb(ちびみくったーちゃん)

本稿:@iori4213 (左), @shijin_cmpb

43 頁:@shijin_cmpb

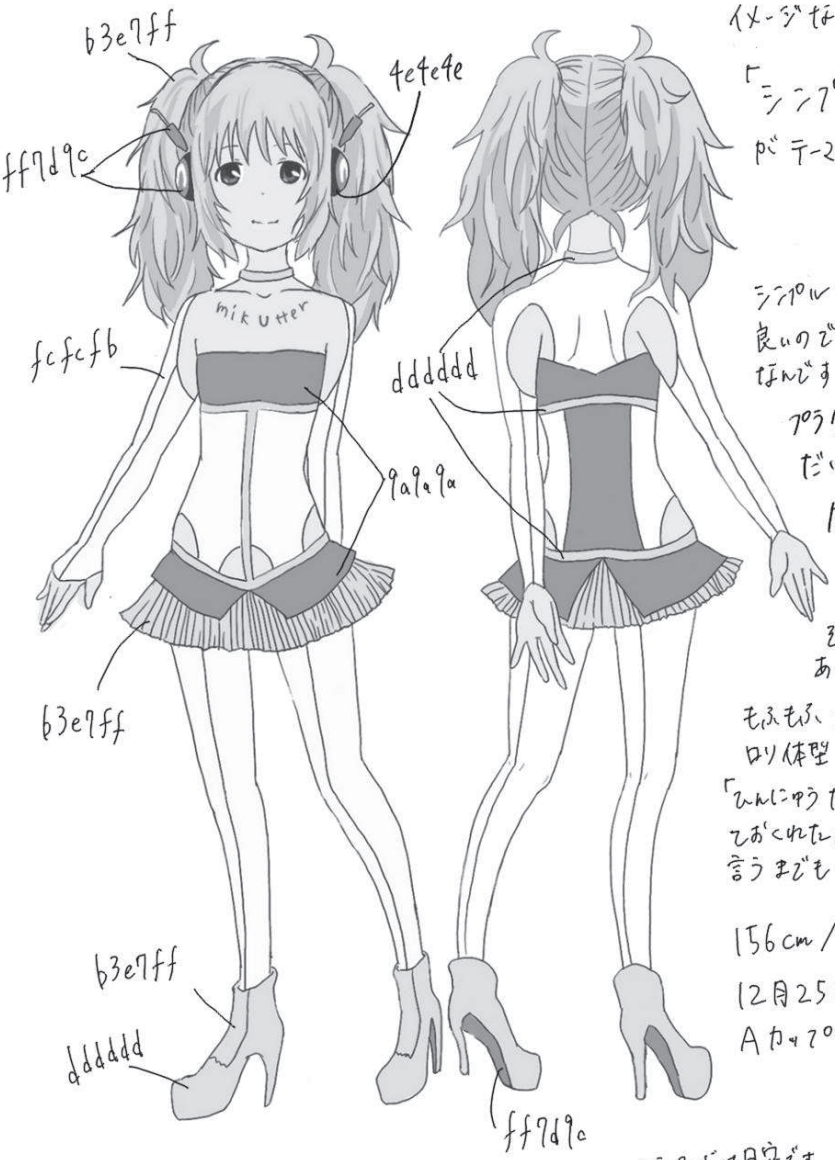
あとがき:@iori4213

それでは、ておくれの世界に気持ち良く旅立っておくれ。

Have a Nice TEOKURE!!

みくったーちゃん とは

「初音ミクちゃうん？」「描きたいけど"デザインよくわからない」…といった声を聞いていたので、この機会に設定集的なものでうすい本に参加させてもらうことになりました。@soraname_bscl（そらまめ）です。自分がもっと描き甲斐ある、カッコイイ…サイバーアイドル？というか、スマイルプリキュアで言うところの「プリンセスフォーム」的な位置づけでデザインしなおしてみました。



mikutterの擬人化のようなイメージなので、

「シンプル」

がテーマです。

シンプルを追求するなら全裸で良いのですし、服なんてただの飾りなんです…。

プラグインを装備してアナタだけのみくったーちゃんをカスタマイズ！　など夢と妄想がふくらみます。

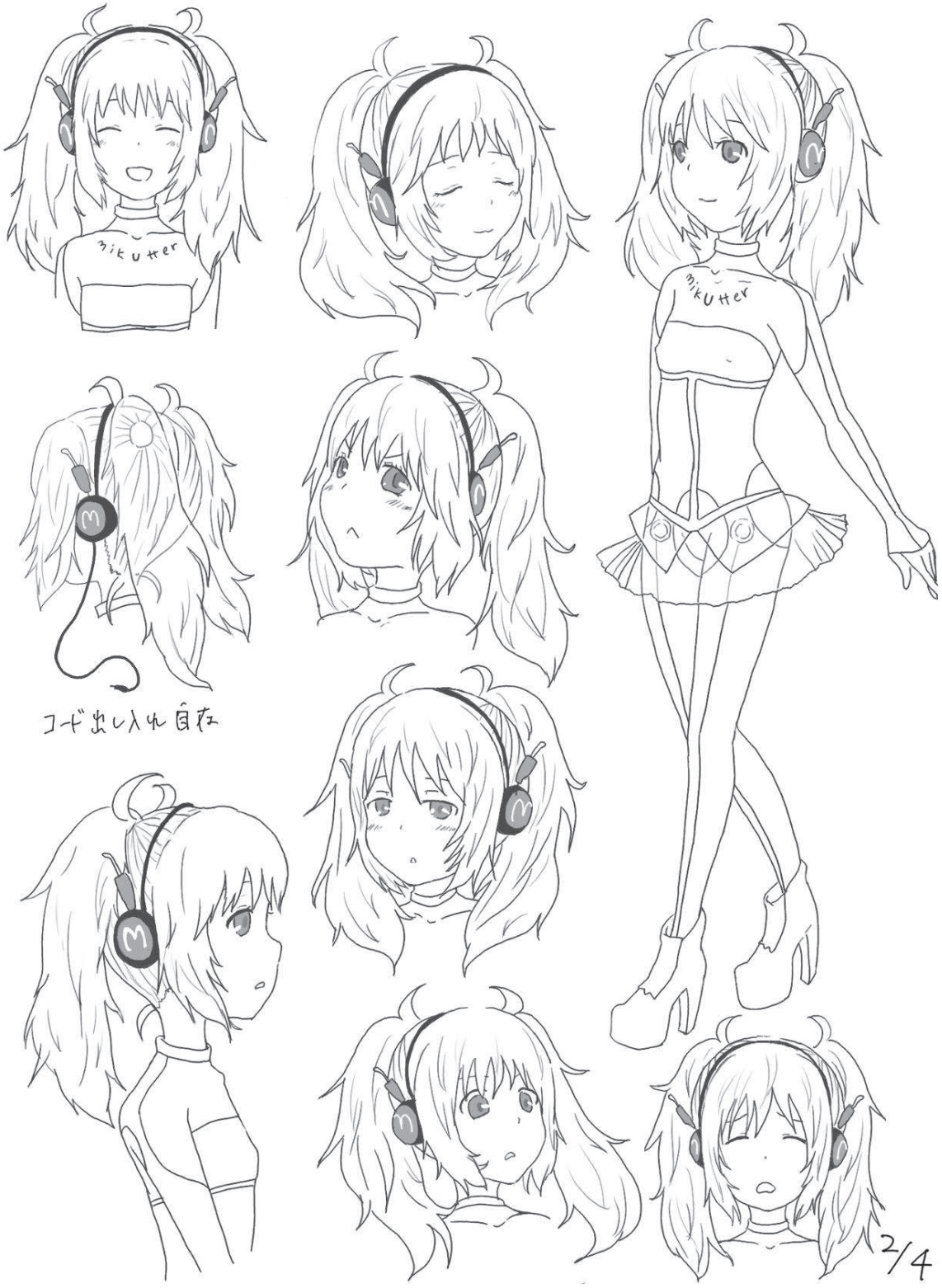
そんなわけで色んな服を着せてあげたいですね！

もふもふ、フンフンしたツインテにロリ体型ではないのにえんにゅう。「えんにゅうだいすきー」という、としあきのひぐれた発言が基となっているのは言うまでもない。

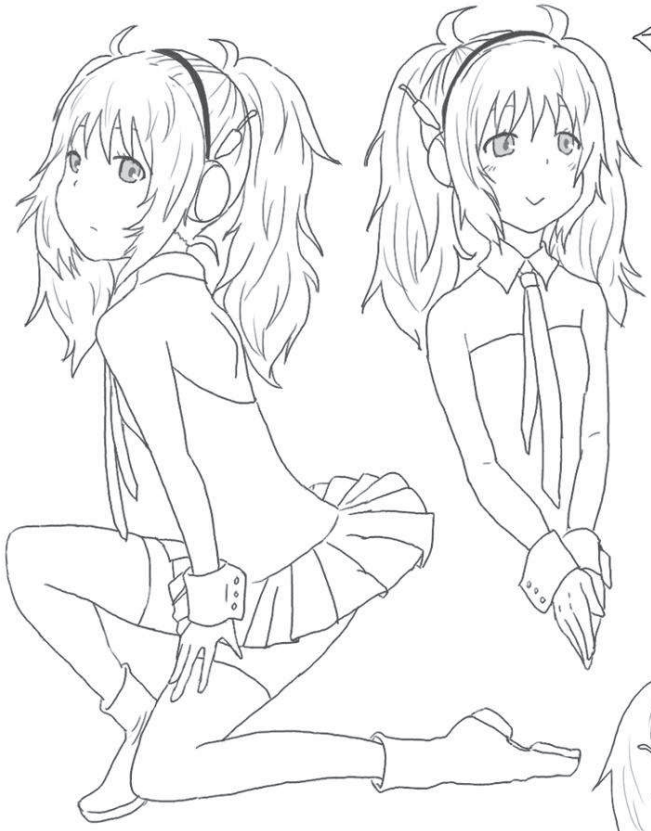
156cm／47kg／18歳
12月25日生まれ
Aカップ

カラーコードは目安です

1／4


miku Her
miku Her
コード出し入れ自在
2/4


キャストオフ!
すると、ちょっぴりエッチ
miku
miku
miku
miku
腹全開です
ふくほぼ爆撃に耐える表情
3/4



初期のデザインというか
スタンダードな みくったーちゃん。
白ワンピに ピンクの細いタイ
お好みで黒ニーソックス
シンプルすぎるけど割と気に入ってるので、
プリキュアで言えば変身前の制服姿
みたいな 位置づけに しておきたい。

mikutter 0.2 記念の おめかしver.
髪を巻いたりしてみた
デートとかしそうな イメージの服？

やっつけ感あふるる
ヘッドホンの図

まめちしき
みくったーちゃんの
好きな食べ物は
ツナバターしょう油パスタ

おすしやな
さんこす

@brsywe
@toshi_a

AND You!!!

@soramame_bscl が
お送りしました

4/4

mikutter グッズを作ろう！

@tsutsuii

1. はじめに

世の中に~~そおくれな~~Twitter クライアントは数あれど、薄い本が vol.3 まで刊行されてしまったり、シールやマグカップなどのキャラクターグッズが製作されて各種のイベントで配布されたりしてしまうのは、今のところ mikutter くらいではないかと思います（Krile の薄い本という話もあるようですが……）。

そういうグッズを作っていると「mikutter ユーザーの~~そおくれ~~愛ぱねぇ」などと言われてしまうのですが、それはもちろん mikutter というソフトの持つ魅力とみくったーさん絵師こと@soramame_bscl さんの絵のかわいらしさ、そして作者である@toshi_a さんの~~そおくれ~~人徳の為せる業です。とはいえ、グッズの製作自体はちょっとした慣れと思い切りがあれば思ったより簡単に手配することができます。

本記事では、実際に配布したいくつかの mikutter グッズについて、どんな感じで~~そおくれたのか~~作っていったのかを紹介してみたいと思います。

2. 企画編

今まで作った mikutter グッズは「こういうのが作りたい！」から始まっていて、「何を作ろうか」というところから考えたことはないのですが、実際に作ってみたシール（ステッカー）、マグカップ、タペストリの 3 つについてそれぞれグッズ作成におけるメリットとデメリットを挙げてみます。

2.1 シール（ステッカー）

mikutter の薄い本 vol.2 の記事でも書いていますが、mikutter シールを作ってみようと思ったのは@toshi_a さんが Twitpic に投稿したスクリーンショット中に貼られていた mikutter バナー広告(?)を見たのがきっかけです。実際のみくったーさんバナー広告画像は Twitpic http://twitpic.com/6pxlx5 で見ることができます。

写真 1 初代みくったーさんシール

グッズとしてのシールの利点としては

・単価が安い（1 枚 10～35 円程度）。なので気兼ねなくばんばん配布できる。

・かさばらない。なので配布でもほとんどの人に受け取ってもらえる。郵送も簡単。

・ノート PC やスマホに貼ってくれる人が多い。なので宣伝効果もそれなりにある。

といった点が上げられます。逆に欠点としては

・安く作ろうとすると 1 ロットの数量が多いので、不良在庫を抱えるリスクがある。

くらいでしょうか。自分用に少量を作る場合は、市販のインクジェットプリンタ用ラベル作成シートで自作するという手もあります。

2.2 マグカップ

mikutter マグカップを作ったのもわりと突発的で、mikutter 0.1.0 リリースでろくろみくったーさんがデビューして TL で mikutter マグカップというキーワードが飛び交ったときに、たまたま自分が部屋で使っていたカップが欠けてしまったので、これまた勢いだけで発注して@toshi_a さんと@soramame_bscl さんに宅配テロしてしまったのでした。詳しくは@okano_t さんによるまとめ http://togetter.com/li/284852 がありますのでそちらを参照してください。

グッズとしてのマグカップの利点としては、

・思ったよりは安い（1 個 700～800 円+送料）。それに対して現物の見た目のインパクトが高い。

・普段から使うものなので、手に入れた後の満足感も結構持続する。

欠点としては

・安いとはいえ、さすがにシールのように気兼ねなく無償配布できる値段ではない。

・かさばる上に割れ物なので持ち込みや発送が大変。頒布の送料や梱包の手間もそれなりにかかる。

といった点が挙げられます。

写真 2 ろくろみくったーさんマグカップ

2.3 タペストリ

TL では mikutter タペストリというのは特に話題に挙がっていませんが、今年 8 月のオープンソースカンファレンス 2012 京都（OSC 京都）に@toshi_a さんが mikutter のブースを出すということで、その目印にということで作ったものです。

写真 3 みくったーさんタペストリ@OSC 京都 mikutter ブース

グッズとしては大量に作るものでもないのですが、利点としては

・サイズの割には安く作れる（60cm × 60cm で 2000 円程度）。値段の割にサイズで圧倒できる。

・イベントでは「名前は知っているけど顔を知らない」という人も多いので、会場ブースの良い目印になる。

といったところです。欠点としては、みくったーさんタペストリではなく同様に作った NetBSD ロゴのタペストリで

・空港の手荷物検査で「これは何ですか？」と問われ、筒の中身を広げて見せる羽目になってしまった

という、各地の OSC で NetBSD ブースを出展している@ebijun さんの事例が挙げられます。

2.4 その他のグッズ

シールやマグカップのほかには、@okano_t さんが mikutter タオルを作成されています。タオルの場合は法人需要が多いと思われるため、最小発注数量が多い、色数の制約が多い（1 色あたりいくらという価格設定）といった課題があることが予想されますが、詳しい情報は@okano_t さんのレポートを待ちたいと思います。

写真 4 @okano_t さんによる mikutter タオル

みくったーさん抱き枕 などという意見も出たことがありますが、調べてみると昨今それなりの需要はあるのかオリジナル抱き枕を製作可能な業者もそこそこ見つかります。しかし、想定ターゲット層が偏っている(?)ことから単価がかなり高い(カバーだけでも1つ 5000 円以上)のと、その用途と形状からして専用の原画を起こさないといけない(トリミングするわけには行かないのでふさわしい(?)ポーズを描いてもらう必要がある)ので、それなりにハードルは高いと思われます。

3. デザイン編

作るブツが決まったら次はどんな絵にするかを考えるわけですが、ここではデザインセンス的な話は抜きにして画像作成時のデータの仕様や作成上の注意点について挙げたいと思います。

3.1 原画

mikutter グッズを作るとなったらみくったーさんを含む何らかの原画が必要になるわけですが、自分自身で絵が描ける人は別として、突発的に思いつきと勢いでグッズを作る場合は既存の絵を流用することになると思います。

図 1 @soramame_bscl さんのみくったーちゃんたんぶらー

ここで注意が必要なのが、印刷用原稿データとしての画像は通常の Web 用の画像と比べるとかなり高解像度の絵、つまり巨大なサイズの絵が必要になるということです。最近はやりの Retina ディスプレーや IGZO 液晶は例外として、たいていの PC 画像は 96dpi 程度の液晶画面を前提に描かれていますが、印刷用原稿としては 300～400dpi 程度の画像データが要求されます。96dpi 前提で描かれた絵をそのまま印刷データとして出してしまうと、むりやり拡大されてガタガタもしくはボケボケの絵になってしまいます。

具体的な例を挙げると、mikutter シールのサイズは 35×100mm で、これを 96dpi で換算した場合は 132×377 ドットでバナー広告程度のサイズですが、印刷用解像度として 350dpi で換算すると 482×1378 ドットとなり、フル HD 液晶で縦がはみ出すようなサイズの絵が必要になります。

使用する原画が 350dpi 換算で必要なサイズより小さい場合はなんらかのツールで拡大することになりますが、その場合は GIMP などの一般的な画像レタッチツールの拡大機能より、拡大操作に特化したツールを探した方がきれいにできるようです。私の場合は SmillaEnlarger というツールを使いましたが、グッズ作成はたいてい単発作業なのでその都度適当に検索して使いやすそうなものを選べばよいかと思います。

拡大ツールを使う場合も、原画のサイズがあまりにも小さいときれいに拡大することが難しくなります。目安として、キャラクターの輪郭線をそのまま拡大して見た場合に1ドットの単位でブロックが見えてしまうようなサイズだとツールを使っても厳しいです。そのような場合は、ターゲット原稿のサイズを再考する、別の原画を探す、原画描きさんにお願いして大きなサイズの絵をもらう、等々の対応が必要です。

3.2 mikutter ロゴ

mikutter 公式サイトで使われている mikutter ロゴのフォントは Maniackers Design (http://mksd.jp/) さんの“Pico-Alphabet White”というフォントです。Twitter 公式のロゴフォントとは似ているようで e の字体が微妙に違っていたりしますが、これはもともと Twitter 公式が当初採用した Pico-Alphabet フォントを後に独自にアレンジを加えて変更したためで、mikutter はオリジナルの Pico-Alphabet White のフォント をそのまま使っています。Pico-Alphabet のフォントは http://www2.wind.ne.jp/maniackers/pico.html からダウンロードできます。

図 2 mikutter ロゴ

フォントの利用に関しては「非営利団体でのグッズ作成、小さな規模の同人誌での使用などに関しては、自由に使って下さって結構です」とありますので、mikutter 界隈の配布程度であれば問題にならないと思います。

文字の色については公式ロゴでは u 以外のピンクの文字は #FF009A、u の水色は #52DBFF が使われています。ただ、印刷の場合は CMYK インクの原色であるマゼンタ(#EC008C)とシアン(#00FFFF)を使ったほうがきれいに見えるかもしれません(未確認)。

4. データ作成および発注編

4.1 画像データ形式

原稿画像のファイル形式は発注先業者の指定によりますが、たいていはイラストレーター形式や jpeg、png、eps 形式などでのデータ入稿が要求されます。今までのグッズ発注では、GIMP で画像データを作成し、eps 形式や png 形式にエクスポートして原稿データとして提出しています。

GIMP でデータを作成する場合は、「画像」→「印刷サイズ」で出てくるメニューウインドウで解像度と印刷サイズを目的のサイズに合わせておき、画像エクスポートの際にも解像度情報を保持するようにしておくと、データ受付側も余計な変換作業が発生せず間違いも少なくなると思われます。GIMP そのものの操作については世間の Web ページの方が詳しいため割愛します。

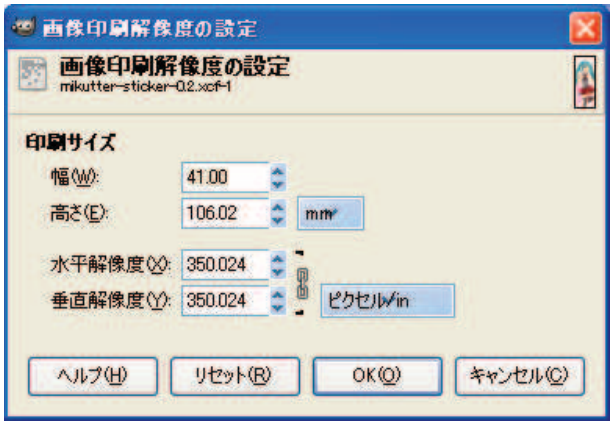


図 3 GIMP の印刷サイズと解像度の設定

印刷データ受け付けではいずれの業者で CMYK 形式(印刷の三原色)でのデータ作成を指定されますが、GIMP は標準では CMYK 形式に対応していないこともあり、いずれのグッズ発注でも RGB 形式のままの原稿データを送っています。RGB 形式のデータを CMYK 形式に変換するとやや赤みが強くなるようですが、今まで作ったグッズでは特に気になったことはありません。

ただし、元画像の明るさやコントラストについては、実際に紙に印刷してみて薄すぎたり濃すぎたりする場合はガンマ値などのパラメータをある程度調整してやる必要があります。

4.2 シールの発注まで

■発注先

mikutter シール作成では毎回 Studio-Happiness さん (http://www.studio-happiness.com/sticker/) を使っています。選択のポイントは他の業者と比較してかなり安かったこと(25×25mm の PC エンブレムサイズなら 500 枚で 5250 円)と、薄い本 vol.2 でも書いたように OSC での NetBSD ブース配布用にロゴシールを作った実績があったから、というものです。

安い理由は多数の原稿を集めて一枚のシートに印刷する付け合わせ印刷を採用しているためですが、その代わりに 1 ロット 500 枚単位でしか発注できない、月に 2 回(15 日と月末)しか受け付けをしていない、納期も 15 日ほど必要という欠点があります。タイミングが悪いと、思い立っても発注できないとか目的のイベントに納期が間に合わないという事態も発生しうるので、締切を見据えて計画を立てるか、場合によっては別のシール業者を開拓する必要もあるかもしれません。

■サイズ

データ作成の前にまずはシールのサイズを決める必要があります。自分の作りたいと思ったシールのイメージにもよりますが、サイズは費用に直結するのでサイズを先に決めてそれに合う絵を選択するという順序もアリですただ、あまり大きすぎると高価な上に貼れるブツが限られ、小さすぎるとデザインやロゴが難しくなります。

初代みくったーさんシールの 35×100mm というサイズは、元々のイメージがバナー広告であったこと、幅 35mm が価格表で一番安かったこと、縦 100mm がみくったーさんのポーズにうまく合いそうだったこと等々で、GIMP 上でデザインとレイアウトを考えながら決めたものです。

低価格優先であれば 25×25mm の PC エンブレムサイズでデザインを工夫することになります。また、同じく @okano_t さんが作成された緑枠の丸みくったーさんシールは 80mmφ サイズです。

写真 5 @okano_t さんによるエンブレムサイズ mikutter シール

■データ作成

データ形式はイラストレーター形式や jpeg、png、eps 形式が指定されていますが、私の場合は GIMP から eps 形式でエクスポートしたデータを正として、確認用として png でエクスポートしたデータも添付して送付しました。

Studio-Happiness さんの「データ作成上の注意」のページにもありますがシール原稿作成の際にはいくつか注意点があります。

・シール外周の裁断ズレ

シールはシート印刷後の裁断で 1mm 程度ずれてカットされることがあるため、シールの端にロゴなどを置く場合は多少切れても問題ないように端から 2mm 程度離しておく必要があります。シールのサイズが小さい場合は周囲 2mm の領域の割合が馬鹿にならないので、ある程度はずれることも覚悟して 1mm 程度の余白で多少切れても違和感のないデザインを考える、といったことも必要になります。

・シール外側の塗り足し

裁断がずれると当然外側にもずれるため、シール端まで一杯に絵を置く場合はシール外形より 3mm 程度外側まで絵のデータを用意しておく必要があります。35×100mm のシールであれば両側 3mm を足した 41×106mm 分（350dpi 換算では 565×1461 ドット）の画像データを用意する必要があります。このとき、塗り足しを含むサイズで画像データを作るとそれだけでも見た目の印象が変わってしまうので、外周 3mm を目隠ししたデータもあわせて見ながら完成形をイメージする必要があります。

■mikutter ロゴ

シールのサイズはそれほど大きくないので、みくったーさんの配置とロゴのサイズのバランスをどうするかというのもなかなか難しく、作成者のセンスが問われます。とはいえ製作には勢いも重要なので、あまり悩みすぎずにいろいろ試してえいやで決めてしまうのが良いのではないかと思います。

■サンプル作成

ある程度の数量を印刷する場合は発注前に原寸でデータを印刷して実際のサイズに裁断してみてイメージを確認しておくべきです。画面上のイメージと印刷したイメージ、さらに実際の寸法に裁断した後のイメージはそれぞれかなり違います。特に、印刷の色合い（くすんだ色になっていないか）と文字のサイズ（離れた状態で見て読めるか）の 2 点は確認しておくべきです。

写真 6 新しいみくったーさんシールの塗り足しあり原稿データと印刷サンプル

■発注

発注はメールで依頼することになります。メールで納品先、注文内容（種類、形状、サイズ、枚数）、原稿データ（zip で固めておくのが無難）、データの説明（画像形式、作成 OS、解像度、塗り足しやロゴ配置などカット部の扱いなど）を送れば 1～2 日後に受付確認のメールが来ます。

なお、従来は原稿確認用画像データを送ってくれていたのですが、11 月に発注した際には受付確認のみで確認画像の送付がありませんでした。規定解像度で塗り足しまで含めた原稿データを送ったので確認するまでもないということなのか、合理化のために確認がなくなったのかは不明です。

■支払いと発送

支払いはゆうパックによる代引きのみですが、発注時に依頼すれば銀行振り込みでも受け付けてくれるようです クレジットカードは対応していません。

4.3 マグカップの発注まで

■発注先

いままで mikutter マグカップは 3 回ほど発注していますが、いずれもマグ屋.com さん（http://mugya.com/）を使っています。マグ屋.com さんの利点としては 単価が割りと安い（1 個でも 730 円）、1 個からでも発注可能、そして何よりも「印刷原稿データが複数あっても発注総数で単価が決まる」というところで、みくったーさんの白/黒/ろくろの各バージョンを 3 個ずつ、という発注も気軽にできます。

■カップのサイズと原稿サイズ

オリジナルマグカップは汎用の白無地のマグカップに専用の機械で焼付け印刷する方式なので、どの業者でもカップの大きさと印刷範囲は同じようなサイズになっています。マグ屋 .com さんではカップサイズが 82mm 径 × 高さ 95mm、印刷範囲は縦 75mm × 周囲 190mm です。より広い印刷範囲を売りにする業者もありますが、このサイズでも特に不足はないと思います。

75 × 190mm で 350dpi のデータを作成すると 2618x1033 ドットの絵になります。データ形式はシールと同様にフォトショップ、イラストレーター形式、png, jpg 等々になりますが、私は毎回 png で発注しています。

■データ作成

マグカップの場合はシールと違い円柱面の上に絵が印刷されるため、画面上の絵と実際の仕上がりの印象がかなり異なります。そのため、発注前に原寸で紙に原稿を印刷して同じようなサイズのカップに巻きつけてみて、絵の見え方を確認することが必須です。特に、円柱面上ではやや絵が細く見えるため不自然にならないか、横幅について正面視でどの範囲まで見えるか、といった点を確認する必要があります。だいたい 90mm 程度の幅であれば正面視で全体が見えますが、ロゴの場合は多少長くても見栄えは悪くありません。

写真 7 マグカップ原稿確認とマグカップ現物

シールと違って外周の塗り足しなどは考慮する必要はありませんが、カップの白地の上に印刷範囲分だけ絵が印刷されるので、上下左右のトリミング部分が不自然にならないか等も一応見ておく必要があります。

マグカップの場合、絵の仕上がりがやや赤みが強くなるほか、シールと違いぼやけた感じの印刷になるので、いきなり大量発注するのはリスクがあります。マグカップは 1 個からでも発注できるので、大量生産の場合はまずいくつか試作してから量産するという作戦でいくべきかもしれません。

■発注

マグ屋.com さんの場合、発注は原稿データを含め Web ページ上の注文フォームから入力することになります。マグカップの画像サイズであればそれほど巨大にはなりませんし、複数のデータを送る場合も原稿ファイル 5 つ分までフォームが用意されていますので、ブラウザからアップロードするのに特に難しいことはありません。

■支払いと発送

支払いはクレジットカード、銀行振り込み、代引きそれぞれが利用できます。発送は佐川で届きますが、梱包は特に緩衝材などは入っておらず、マグカップが個別のダンボール箱に入っているだけなので、佐川の担当者によっては破損が怖いような気もします。うちに来る佐川担当の人はすでに顔なじみで取り扱いも丁寧なので問題があったことはありませんが、届いたらすぐに全数状態を確認すべきかもしれません。

4.4 タペストリ発注まで

■発注先

タペストリ作成では Web で IMAKU さん（http://www.imaku.jp/）というところを使いました。ポイントは値段が安かったことですが、どちらかというと Google 検索結果からホームページを見て見積もりを試してみたらあまりに安かったので作ってみようという気になった、というのが実態です。他の業者との比較はあまりしていませんが、納期も短く Web 上の発注システムも良くできているので、そこそこお勧めだと思います。

■データ作成

タペストリの場合は大量に作るものではなく一品ものなのでわりと気楽に作れますが、ブツのサイズが大きい分原稿データも巨大になるという点がやや異なります。

ろくろみくった一さんタペストリの場合、サイズを 60×50cm としたのでそのまま 350dpi の画像でデータを作り、結果 8269×5512 という巨大サイズの画像を原稿データになりましたが、特に問題なくそのまま受け付けてもらえました。通常はここまで巨大なサイズの原画を用意することは無理なので、データ作成時は前述の拡大ツールで元絵を拡大することになりますが、遠目には細かいドットは全然わかりませんのでそれほど神経質になる必要はありません。文字やロゴデータについてもアウトラインフォントを使えば特に難しいところはありません。

写真 8 タペストリ用フル HD 解像度みくった一さん原稿データ作成中の図

なお、NetBSD ロゴマークのタペストリを作った際はベクター形式のデータが eps 形式で用意されていたので、そのようなデータを使えば手動での拡大操作は必要ありません。その場合、画像の実サイズについては注文時の備考欄などで指定しておく必要があります。

■発注

IMAKU さんの場合も、発注は原稿データを含め Web ページ上の注文フォームから入力することになります。上述の 60×50cm のタペストリの 8269×5512 ドット原稿だと eps 形式で約 28MB になりましたが、50MB 程度までは Web アップロードできるようなので、よほどの巨大なものを作らない限り問題はないと思います。

発注後の原稿データ確認も Web ページ上で行うことになります。サイズや見栄えを確認してから「これで OK」のボタンを押した時点で GO がかかるので、前述のシールやマグカップと比べると安心して発注することができます。納期はかなり早く、3 日程度ですぐに届きます。

タペストリにはロープも付属しますが、実際にブースで使う場合はロープではうまく固定ができないので、ホームセンターや 100 円ショップで売っている 50～70cm 程度の伸縮つっぱり棒を使うと持ち運び含めて便利です。

■支払いと発送

支払いは銀行振り込みか代引きになります。クレジットカードは対応していません。現物は卒業式の筒というか、キャップ付きのいわゆるサランラップの筒で送られてくるので、イベント会場への参戦もそのまま流用できます。

5. 終わりに

すべての始まりである mikutter 作者@toshi_a さんが OSC 京都で mikutter ブース出展および mikutter セミナー発表をされた後、自身のブログで「mikutter を作り始めたときに決めたこと」として以下のようなことを書かれていました。

- **楽しくなくなったらやめる。**
- **自分のために。**

これが何を示しているかといえば、作者の@toshi_a さん自身の

『俺はこういう Twitter クライアントが作りたいんだ！』

という思いが mikutter 開発の原動力となっているということ、そして

『mikutter というソフトを開発し使っている @toshi_a さん自身が一番楽しいと感じている』

ということではないかと思います。そして、これらの思いこそが、mikutter が多くのユーザーを ~~でおくれさせている~~ 魅了し続けている要因ではないかと思います。

グッズの作成も同様で、自分自身が「こういうものが欲しい！」「作っていて楽しい！」という思い、そしてノリと勢いさえあれば、結果としてほかの人たちにも喜んでもらえるものができあがると思います。グッズでもプログラミングでも絵画でも「これだ！」という思いが形になったときの楽しさは格別なので、皆さんもそれぞれの「思い」を ~~でおくれさせて~~カタチにしてみてください。

写真 9 としぁさん作の ShootingStar T シャツ

Armadillo ではじめる mikutter ライフ

@arc680

1　はじめに

1.1　mikutter との出会い

Twilog によると少なくとも 2011 年の冬ぐらいにインストールしたと思われる。詳しいことは、mikutter Advent Calendar で書いてると思うよ（適当）。ただ言えるのは、その当時はとしぁさんのこととか知らなかったし、Windows で動かそうとしていたということかな。少なくともておくれではなかった。としぁさんも「ほらみろ、mikutter 使ってても<u>ておくれ</u>てないやつはいくらでもいるんだ。今はもう<u>ておくれ</u>だがな…」って言ってたし。

1.2　Armadillo 500FX との出会い

今回使う"Armadillo（アルマジロ）500FX"は、ARM コア搭載組み込みプラットフォーム（公式ページより）の 1 つである。もともと別の研究で使われており、最近出番のない状況であった。ちなみにスペックは表 1 のとおりである。

表 1 スペック

CPU	ARM11 532MHz
RAM	128MB
ROM	32MB
SSD	1GB
インターフェース	USB×2、LAN、microSD、タッチパネル液晶

研究室でやったちょっとした仕事で使いたいって指導教員に言ったことから始まった。建前としてのアレはほとんどやってない。許し<u>ておくれ</u>。大体のことは Armadillo（以後，端末）を使わずにやったしもうゴールしてもいいよね。

いや、本題に入ろう。

2 Debian のインストールまでのあれこれ

2.1 端末の設定

PC と端末はシリアルで通信する。そのため、シリアルクロスケーブルで接続し，シリアル通信に対応したソフトウェア（例：TeraTerm など）が必要である。PC にシリアルインターフェースがない場合は、USB-シリアル変換ケーブルも必要である。端末の初期設定とかそこら辺は省略する。設定って言っても、固定 IP アドレスにしたい場合はネットワーク設定するぐらいだが。

2.2 Debian のダウンロード

端末で公式で配布している Debian をダウンロードする。そして展開して所定の場所へ移動する。

```
# mke2fs -j /dev/sda1
# mount /dev/sda1 /mnt/
# mkdir tmp
# mount -t ramfs ramfs tmp
# cd tmp
# for N in 1 2 3 4 5 a500fx; do
> wget http://download.atmark-techno.com/armadillo-500-fx/debian/
debian-lenny-armel-${N}.tgz;
> gzip -cd debian-lenny-armel-${N}.tgz | (cd /mnt; tar xf -);
> sync;
> rm -f debian-lenny-armel-${N}.tgz;
> done
# cd
# umount tmp
# rmdir tmp
# umount /mnt
```

そのあとに、カーネルをビルドする必要がある。実際には、配布されている開発環境（VMWare）上でビルドする。バージョンに関しては，適宜最新のものにあわせていただきたい.

```
$ wget http://armadillo.atmark-techno.com/files/downloads/
kernel-source/linux-2.6.26-at/linux-2.6.26-at13.tar.gz
$ tar xzvf linux-2.6.26-at13.tar.gz
$ cd linux-2.6.26-at13
$ make armadillo500fx_dev_defconfig
$ make menuconfig
Linux Kernel Configuration
  Kernel Features  --->
    [*] Use the ARM EABI to compile the kernel
    ↑チェックを入れる
    [ ] Allow old ABI binaries to run with this kernel (EXPERIMENTAL)
    ↑チェックを外す
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
  && gzip -c arch/arm/boot/Image > linux.bin.gz
```

しかし、そのまますんなりソースコードからビルドするだけでは上手く行かなかった。

make 時になんか専用の gcc などが無いって言われる。足りないものをシンボリックリンクで置き換えてやる方向で問題ないかと思われる（ググって出てきたところではちょっと違った書き方であったが、エラー文にあわせている。あとバージョンで変わるような気もする)。改めてビルドをする。

そして、できたカーネルを USB メモリに突っ込んで端末に書き込む。次のコマンドで linux.bin.gz を USB メモリに突っ込んで書き換える。

```
# netflash -knusr /dev/flash/kernel /mnt/linux.bin.gz
```

保守モードに切り替えて設定変更する。

```
hermit> setenv console=ttymxc0 root=/dev/sda1 rootwait
        usb-storage.delay_use=0
```

モードをもとに戻して再起動を行う。これで Debian が起動するはず（図 1)。

図 1 Debian が起動

3 GUI 環境との戦い

いくら mikutter 用サーバにするとはいえ、この端末から直接投稿できたら素敵やん？と考えたので GUI 環境も整えることに。どちらにせよ他にもいろいろ入れるものがあったので apt-get する。そんなサーバねーよって言われた。しかたがないので、apt 先を最新のものにして再度 apt-get する（下記参照)。

```
deb http://ftp.jp.debian.org/debian/ squeeze main
deb-src http://ftp.jp.debian.org/debian/ squeeze main
deb http://security.debian.org/ squeeze/updates main
deb-src http://security.debian.org/ squeeze/updates main
```

じゃあ上手く行ったのでここからインストールしていく。まずは Xfbdev と jwm 入れた。

```
# Xfbdev &
# jwm &
```

起動したが、タッチパネルとマウスがデバイスとしてちゃんと認識されている（？）のに、動かない (図 2)。公式どおりに設定ファイルを作ったのに。今回は違う方法で mikutter を使うため放置する。

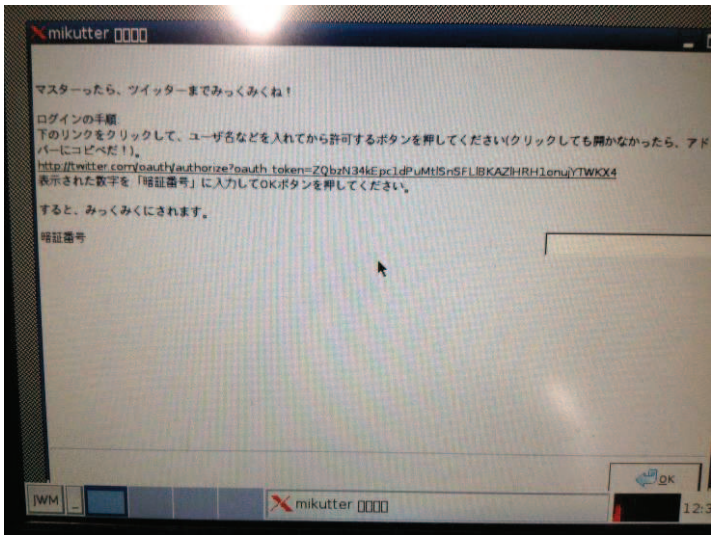


図 2 タッチパネルとか使えないから何もできない図

4 mikutter のインストール

まずは mikutter を動かすのに必要なものを入れる。

```
$ sudo apt-get install ruby1.9.1-dev libgtk2.0-dev
$ sudo gem1.9.1 install gtk2
```

あとはいつも通りに svn とかで mikutter を入れる感じ

```
$ sudo apt-get install subversion
$ svn co svn://toshia.dip.jp/mikutter/trunk
```

5 「みくらうど」でみっくみく

違う方法で mikutter を動かす、なんて偉そうなことを言った。しかし、単純に「みくらうど」を作ってそこで動かすだけである。Windows の場合は、Xming をインストールする。これは画面の指示に従っていけば問題ない。~~Mac の場合は、普通に動かせばいいだろう。~~また、Putty の SSH の設定で XForwarding するように設定しておく（図 3)。もちろん Debian に SSH で接続できるように設定もしておくこと。

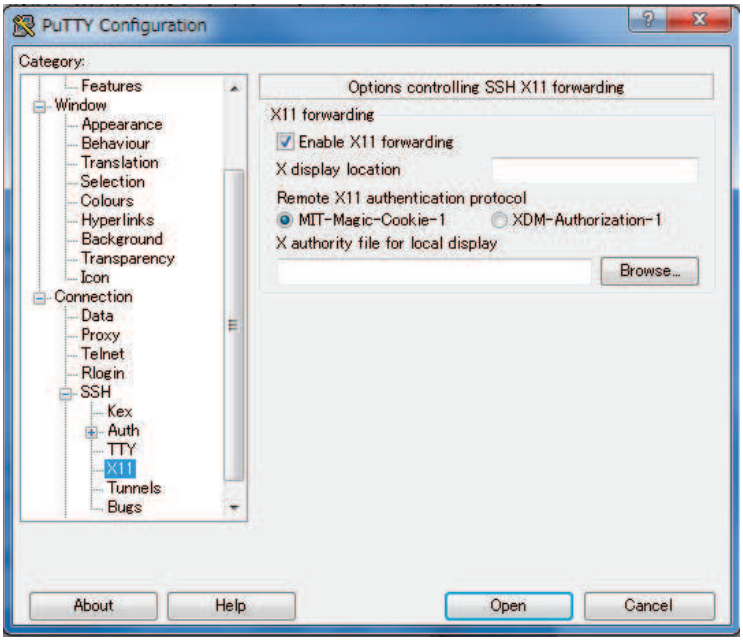


図 3 Putty の設定

Putty でログインする（図 4)。そして、Xfbdev を実行し、mikutter を起動する（図 5)。

```
$ sudo Xfbdev &
$ ibus-daemon -d -x
$ ruby1.9.1 mikutter/trunk/mikutter.rb
```

図 4 Putty を使って Armadillo にアクセス

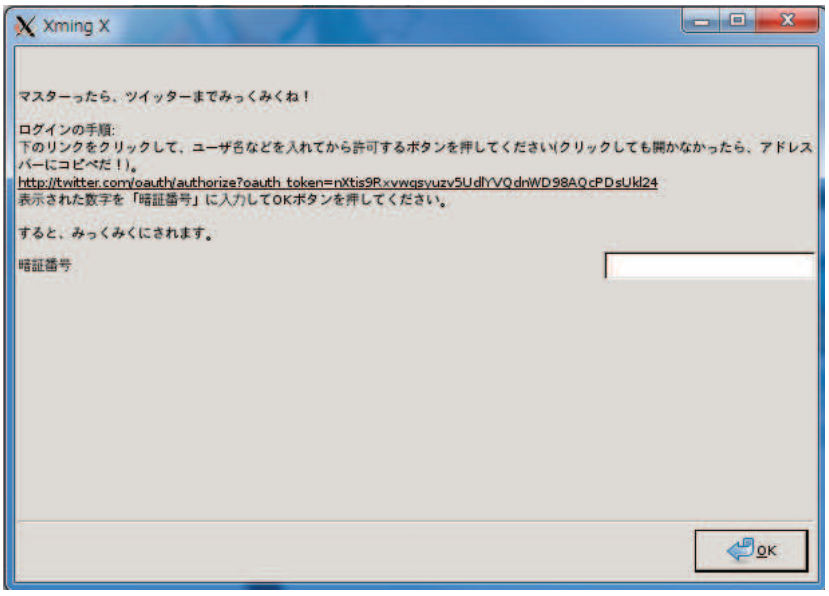


図 5 Windows 上に現れる mikutter

いつもどおりに認証して番号を入力して、「OK」をクリックする。すると図 6 みたいに TL が表示される。あとはツイートするだけ。

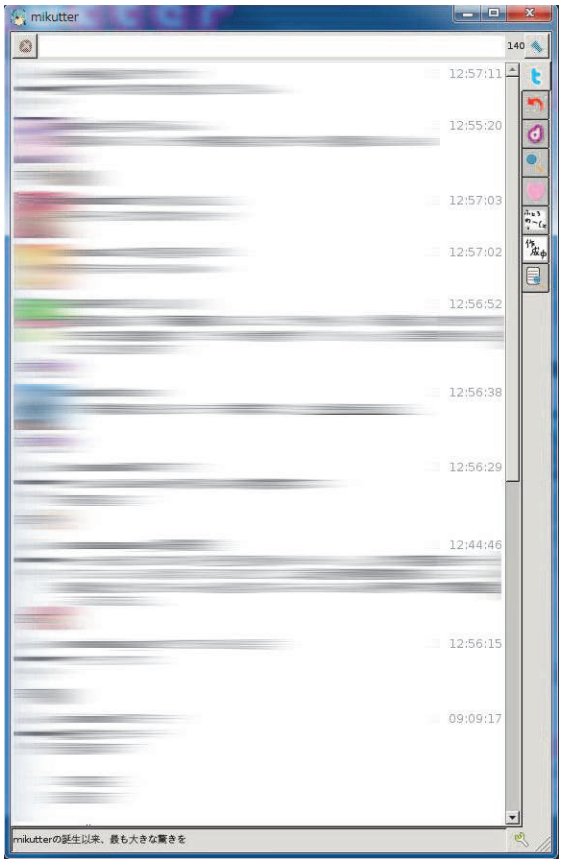


図 6 TL を表示

6 おわりに

とりあえずは投稿できたので目的は達成できたと言えるでのはないだろうか。今後は、日本語入力できるようにするのと、メモリまわりがアレで~~、だんだん遅延が激しくなって落ちるのを~~どうにもならないだろうが，どうにかしたいと思ってたりする。とは言え、こんなスペック（クロック数 532MHz，メモリ 128MB）でも mikutter が動くことが確認できたので、この端末よりもスペックが上なら少々古い PC でもなんとかなる（はず）。

正直な話、このネタはぺんぎんさんの「みくらうど」に関する記事がなかったら存在してなかった可能性が大きいだろう。この場を借りてお礼申し上げます。ありがとうございます。

特に大したネタがあるわけでもないのにこんなことをしていいのだろうかと思いつつ、この記事を書いた。もうしわけねぇヽ(‘ω’)ノ三ヽ(‘ω’)ノもうしわけねぇ

【参考】

- Armadillo-500 FX | 組み込み Linux の Armadillo サイト
  http://armadillo.atmark-techno.com/products/a500fx
- Armadillo-500 FX 液晶モデルで Debian lenny を使う | 組み込み Linux の Armadillo サイト
  http://armadillo.atmark-techno.com/howto/a500fx-debian-lenny-x
- Xming X Server for Windows - Official Website
  http://www.straightrunning.com/XmingNotes/
- Debian squeeze に mikutter をインストールしたよ - もぐてっく
  http://moguno.hatenablog.jp/entry/20120320/p1
- mikutter クラウド「みくらうど」をつくろう！ - ぺんぎんさんのめもちょう
  http://d.hatena.ne.jp/penguin2716/20120615/1339720703

mikutter プラグインで学ぶはじめての Ruby

@penguin2716

1. はじめに

1.1. 対象とする読者

この文章は，あなたが Ruby 以外の言語をある程度書けることを想定しています．Ruby や mikutter の環境構築等に関しては基本的に説明していません．また，mikutter が導入済みであることを想定しています．まだ導入していない方は，ぜひググりながら導入してください．既に mikutter プラグインを書いたことがある人も，楽しく読んでもらえると嬉しいです．

1.2. この文章の目的

「はじめての Ruby」とタイトルに入っていますが，**Ruby の入門に関しては Ruby の入門本を読んだほうが良い**です．Ruby が書けるようになる保証はしませんが，今までに Ruby を書いたことのない人も，ある程度 mikutter プラグインは書けるように考慮しています．

1.3. メタキーの記述方法について

Ctrl+a で全選択，**Alt+Tab** でウィンドウの切り替えといったショートカットキーがありますが，この文章ではこれらのキーを **C-a** や **M-tab** といったように記述します．お使いのキーボードによっては Windows キーがないかもしれませんが，その場合は適宜読み替えて下さい．

キーの組み合わせ	記述例
Ctrl + a	C-a
Ctrl + Enter	C-RET
Alt + x	M-x
Windows + r	s-r

2. mikutter の Ruby コンソール

mikutter には Ruby コンソールが付属しています．適当にタイムラインを選択して，M-x するとコンソール画面が開き，Ruby のコードを入力すると mikutter で実行することができます．mikutter は Ruby で記述されているので，mikutter 内にあるメソッドを呼べばツイートしたり，特定の相手をふぁぼったり，bot を作ったりできます．

2.1. とりあえず使ってみる

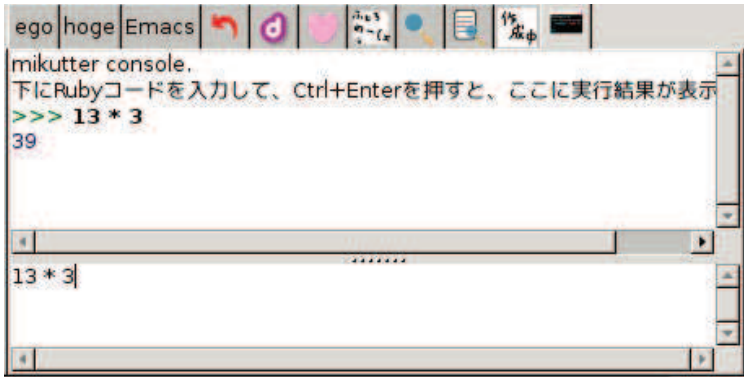
mikutter を起動し，適当にタイムラインを選択して M-x しましょう．

```
mikutter console
下に Ruby コードを入力して、Ctrl+Enter を押すと、ここに実行結果が表示されます
```

といったメッセージが表示されたタブが開きますね．そのメッセージの下には Ruby コードの入力領域があります．ここに，次のコードを入力して C-RET してみましょう．

```
13 * 3
```

上の実行結果に 39 と表示されましたね．ここに Ruby のコードを貼って C-RET すると，そのコードが評価されて実行結果が出力されます．



コンソールでは mikutter 内のメソッドを呼ぶことができるため，次のようなコードを入力して C-RET すると"Hello, mikutter console!" とツイートされます．

```
Service.primary.post :message => "Hello, mikutter console!"
```

2.2. 基本的な FizzBuzz ツイート

スクリプト言語を触ったことがある方なら，フィーリングでなんとなく遊べると思います．条件分岐とループがあれば基本的なプログラムは記述できると思うので，ここでは簡単なコードを記述してみましょう．

```
Thread.new {
 10.times do |x|
   if x % 2 == 0 and x % 3 == 0
     Service.primary.post :message => "#{x}: FizzBuzz"
```

```
  elsif x % 2 == 0
    Service.primary.post :message => "#{x}: Fizz"
  elsif x % 3 == 0
    Service.primary.post :message => "#{x}: Buzz"
  else
    Service.primary.post :message => "#{x}"
  end
  sleep 1
 end
}
```

このコードを実行すると，1 秒毎に FizzBuzz なツイートがされます．

まず，Thread.new { … } のブロックは，ブロック内のコードを別のスレッドとして実行します．このブロック部分の記述がないと mikutter の動作が 10 秒間停止するので，sleep を利用する場合は Thread を利用したほうが良いと思います．

10.times do |x| … end のブロックは，変数 x に 0 から 9 までの 10 個の数字を順に代入しながらブロック内のコードを実行していきます．{…}と do … end は基本的に同じなので，好みによって書き換えることが可能です．

if ~ elsif ~ else ~ end もキーワードが若干異なるかもしれませんが， 慣れ親しんだ言語と動作は同じものです． Service.primary.post :message = "foobar"でツイートします．好みによって，Service.primary.post(:message = "foobar")のように括弧を記述できます．ツイートする文字列の中に#{}のような記述をすると，その中は Ruby のコードとして評価されます．

sleep は引数として秒数を取ります．Twitter ではリクエストを送信した順にツイートされるとは限らないので，1 秒待って順番通りにツイートされるように調整しています．

3. mikutter-mode 導入のすすめ

mikutter-mode を導入すると，プラグインの開発が*かなり*簡単になります．Emacs に mikutter-mode を導入してもっとプラグインの開発を加速しましょう！mikutter-mode を導入すると，次のようなことが可能になります．

- 起動中の mikutter に動的にプラグインをインストールする
- 起動中の mikutter にインストールされているプラグインを動的に書き換える
- 起動中の mikutter で Emacs で編集中の Ruby のコードを実行する

※Emacs ユーザの方以外はこの章を読み飛ばすか，Emacs の導入を検討して下さい．

3.1. mikutter-mode 導入手順

基本的には以下の手順に沿って実行すると導入できると思います．mikutter-mode では

コマンドを dbus 経由で転送しているので，dbus を先に起動しておく必要があります．

1. dbus をインストールして起動します
2. ~/.emacs.d/とかで次のコマンドを実行します

```
git clone git://github.com/toshia/mikutter-mode.git
```

3. お好きなエディタで~/.emacs とかを開きます
4. 次の行を追記します

```
(add-to-list 'load-path "~/.emacs.d/mikutter-mode/")
(require 'mikutter)
```

5. Emacs を再起動します
6. mikutter/plugin/の中に~/.emacs.d/mikutter_mode/mikutter_mode へのシンボリックリンクを作成して mikutter を再起動

これで mikutter-mode の導入ができました！！

3.2. mikutter-mode の基本的な使い方

まずは mikutter を起動します．次に Emacs で適当なファイルを開きます．何でも構いませんが，今回は*scratch*を使います．*scratch*バッファに移動したら，次のようにモードを変更します．

```
M-x ruby-mode
M-x mikutter-mode
```

モードが設定できたら，次のコードを挿入してください．

```
Service.primary.post :message => "Hello, mikutter-mode!"
```

入力できたら，C-c C-c します．すると，起動中の mikutter で"Hello, mikutter-mode"とツイートされます．

このように，編集中のバッファのモードを ruby-mode/mikutter-mode にしてコードを書いて C-c C-c すると，そのバッファ内のコードが mikutter に転送され，実行されます．

4. mikutter のプラグインを作る

他の言語と同じように，Ruby にも様々な構文やメソッドが用意されています．ここの章では，適当なプラグインを作りながら Ruby の構文を勉強しましょう！「Ruby が書けるようになる」ことよりも「プラグインを作って（ふぁぼって）楽しむ」ことを目標にすると早くスラスラ書けるようになると思います．

4.1. 流れてくる「てぉくれ」を全部ふぁぼるプラグイン

「てぉくれ」って素敵な響きですよね．mikutter ユーザなら誰しもがそう思うでしょう．そう，**TL に流れてくるすべての『てぉくれ』をふぁぼりたい**と思うのも自然なことです．そこで，mikutter から離れていても，「てぉくれ」を逃すことなくふぁぼってくれるプラグインを作ってみましょう！

```
# -*- coding: utf-8 -*-
Plugin.create :fav_teokure do
 on_appear do |ms|
   ms.each do |m|
     if m.body =~ /てぉくれ/ and m.user != Service.primary.user
       m.favorite(true)
     end
   end
 end
end
```

短いですね！'#'から始まる行はコメントです．mikutter プラグインは，`Plugin.create` で始めます．`:fav_teokure` はプラグイン名です．`on_appear{ |ms| … }` では，タイムライン，検索結果，リスト等に出現したツイートの配列が ms に格納されています．ブロック内では，ms に対して処理を行います．

`ms.each { |m| … }` では，ツイートの配列 `ms` から個別のツイート `m` を取り出しながら処理を行います．`m` や `ms` はただの識別子なので，別に他の文字列でもかまいません．

`m.body` は対象のツイート本文の文字列を返すメソッドです．`"文字列" =~ /正規表現/` の形式で文字列中の正規表現にマッチする箇所が返ります．

Ruby の if 構文は and/or で条件を構成できます．`m.user` はツイートの投稿主を，`Service.primary.user` は自分を表します．ここでは，"てぉくれ"という文字列を含む他のユーザのツイートをふぁぼるように定義しています．

`m.favorite(true)`では，該当するツイートをふぁぼっています．ちなみに，ここで `m.favorite(false)`とするとあんふぁぼになります．`m.favorite` の引数はデフォルト値が `true` なので，`m.favorite` だけでも動作します．

`on_`*`eventname`* の形式で様々なイベントをフックできます．利用可能なイベントに関しては，mikutter/core/plugin.rb を覗いてください．

4.2. ツイートに色をつける

鍵垢ユーザのツイート背景を灰色にすれば，誰が鍵垢なのか一目瞭然ですね！mikutter は標準では鍵垢を区別して表示する機能はありませんので，プラグインを作りましょう！

```
Plugin.create :gray_protected_users do
 filter_message_background_color do | mp, array |
```

```
   if mp.to_message.user[:protected] == true
     array = [0xcccc, 0xcccc, 0xcccc]
   end
   [mp, array]
 end
end
```

短いですね！！filter_*filter*の形式でフィルタを実装することができます．利用可能なフィルタに関しては，mikutter/core/plugin.rb を参照して下さい．

```
=== message_background_color(Gdk::MiraclePainter miracle_painter, Array color)
_miracle_painter_ のツイートの背景色を変更する。 _color_ は現在の色又は nil。
_color_ は、0-65535 までの RGB 値でを含む三要素の配列([65535, 65535, 65535] 等)。
```

このようなコメントがあるので，message_backgroundcolor というフィルタの引数は Gtk::MiraclePainter のインスタンスと色の配列だとわかります．mp.to_message でツイートのオブジェクトが返ってくるので，on_update で取得した際と同じように使用できます．

フィルタプラグインを作成する場合，引数として受け取った値（上記のコードでは|mp,array|の部分）を返り値として設定します．Ruby ではメソッドの最後で評価された値が戻り値として設定されるので，処理の末尾に[mp,array]と記述しています．

4.3. コマンドを登録する

mikutter でショートカットキーを設定する場合，次のように command メソッドを呼び出します．あるツイートの Favstar を参照したい場合，次のようなコードとなります．

```
Plugin.create :open_favstar_of_tweet do

 command(:open_favstar_of_tweet,
         name: 'この Tweet の Favstar を見る',
         condition: Plugin::Command[:HasMessage],
         visible: true,
         role: :timeline) do |opt|
     opt.messages.each do |m|
     Gtk::openurl("http://favstar.fm/users/#{m.user.idname}/status/#{m.id}")
   end
 end

end
```

condition に関しては mikutter/core/plugin/command/command.rb を参照して下さい．Gtk::openurl で URL をブラウザ等で開くことができるので，メッセージのユーザと ID を利用して URL を生成します．

4.4. ふぁぼられ数をカウントする

誰でも一度は数えてみたいふぁぼられ数．mikutter でも数えることができます．今回は

mikutterが起動してから自分が何ふぁぼされたかをカウントしてみます．

```
Plugin.create :favorited_count do
 @global_favedcount = 0
 @devils = {}

 def global_favedcount
   @global_favedcount
 end

 def devils
   @devils
 end

 on_favorite do |service, user, message|

   if user != Service.primary.user
     if @devils[user.to_s] == nil
       @devils[user.to_s] = 0
     end
     @devils[user.to_s] += 1
     @global_favedcount += 1
   end

   if @global_favedcount % 100 == 0 and @global_favedcount > 0
     top = @devils.sort_by{|key, value| -value}
     tweet = "mikutterを起動してから#{@global_favedcount}ふぁぼされましたヾ(㋮⌒ー⌒㋮)
ノ"
     if top.size >= 3
       tweet += "ふぁぼ魔Top3は，㋮#{top[0][0]}(#{top[0][1]}ふぁぼ)，㋮#{top[1][0]}
(#{top[1][1]}ふぁぼ)，㋮#{top[2][0]}(#{top[2][1]}ふぁぼ)です！"
     end
     Service.primary.post :message => tweet
   end

 end

end
```

@から始まる変数はインスタンス変数，@@から始まる変数はクラス変数です．

```
@global_favedcount = 0
@devils = {}
```

=> 変数の初期化を行っています．{}は空のハッシュです．ハッシュは文字列を引数としてデータを保存します．詳しくはおググり下さい．

```
def ...
```

=> 各変数にアクセスするためにメソッドを用意しています．このメソッドがあると，mikutterコンソールから

```
Plugin.create(:favorited_count).global_favedcount
```
のように叩くことで，現在のふぁぼられ数がわかります．自分のふぁぼりを除外していますが，条件を変更すれば自分のふぁぼ数をカウントすることもできます．

on_favorite では，ふぁぼられた際に全体のふぁぼられ数とふぁぼったユーザのふぁぼ数をインクリメントしています．ふぁぼられが 100 の倍数になる毎にふぁぼられ状況をツイートするようにしてみました．ふぁぼ魔にどれだけ自分がふぁぼられているかをアピールすることで，さらなるふぁぼの嵐を呼び込むことができるでしょう．

5. mikutter のソースを漁る

mikutter プラグインを作る場合，mikutter のソースを漁れば使えそうな部分が出てくることが多いので，

```
$ grep -Rin favorite
```
とか叩けばそれっぽい部分が出てきます．同様に

```
$ grep -Rn 'みくちゃんぺろぺろ'
```
とか叩けば，そのようなソースコードが出力されると思います．面白そうなクエリで検索をかけて Twitter に晒してみたりするのも面白いと思います．

プラグインの作成に役立ちそうなソースは，

- mikutter/core/message.rb
- mikutter/core/plugin.rb
- mikutter/core/plugin/*

あたりにあるので，参照してみてください．

6. おわりに

最初の章くらいでしか Ruby のことを書いてない気がしますが，なんとなくプラグインは作れたでしょうか．mikutter のソースを探検するとたくさんのプラグインが出てくるので，ソースの一部をパクってきて改造したり，組み合わせるだけでもかなり柔軟なプラグインを作ることができるようになります．ぜひソースを grep して楽しくておくれなプラグインを量産してください．くだらないものほど良いです！！やる気がでます！！！

さて，mikutter のプラグインを作ることで，自分好みにカスタマイズできることが分かって頂けたのではないかと思います．Ruby は他のプログラミング言語と同様に大体のことは実現できます．自分好みの mikutter にして，ておくれライフを満喫してください．

最後になりましたが，執筆中に研究室に忙殺され，原稿の締切を 24 時間以上ておくれてしまいました．室長さんごめんなさい．この場を借りてお詫び致します．次回は研究室に忙殺されないように気をつけますね．

mikutter フレームワーク

@toshi_a , 2012 年 12 月吉日

1. はじめに

mikutter プラグインが機能を自然に追加できるのは、プラグイン機能が後付けではなく、最初からそう設計されていたからです。 イベントやフィルタといった mikutter の核を少しずつ見ながら、mikutter それ自体がプラグインを連携させ一つのアプリケーションたらしめる「環境」を構築するフレームワークであるということを紹介したいと思います。 私が今まで「mikutter の薄い本」に寄稿したものはいずれも実用的なものでしたが、今回は知らなくても書ける内部処理をちょっと覗いてみるという、ややトリビアルな内容です。

2. 並列実行と非同期処理

多くの UI ツールキットがそうであるように、mikutter もシングルスレッドモデルを採用しています。 UI を操作するような処理はすべてメインスレッド（＝UI スレッド）で実行され、他の重い処理は必要に応じて別のスレッドで行います。 mikutter の場合は、Ruby が現在複数のスレッドを同時に実行することができないので、マルチスレッドにするメリットが薄く、 メインスレッド以外のスレッドで処理をすることは稀です。 しかし、プラグインをチューニングしていくと、どうしてもマルチスレッド処理がしたい場合があります。そういう時は、Ruby の Thread オブジェクトを使う前に、以下のクラスを使えないか検討してみましょう。

2.1. SerialThread

SerialThread は、コンストラクタに渡されたブロックを、メインスレッドではないスレッドで実行します。最も単純な使い方は Thread クラスと大差ありません。

```
SerialThread.new {
  # 別のスレッドで実行される
}
```

複数のインスタンスを作った時に変わった挙動をします。SerialThread が二つ以上生成されると、先に作られたものから順番に実行されます。 他のスレッドとは並列に動作しますが、SerialThread 同士は直列に実行されるのです。例えば以下のコードを実行すると

```
SerialThread.new {
  p "a"
}
SerialThread.new {
  p "b"
}
```

必ず a の出力が終わったあとに b が出力されます。

むやみにスレッドを増加させると、それ自体が負荷の原因になることもあります。定期的な DB への保存など、あまり優先順位が高くない処理をメインスレッドをブロックせずに実行したい場合は、 SerialThread を使うと良いと思います。

2.2. SerialThreadGroup

実は先に紹介した SerialThread は、 **mikutter/core/serialthread.rb** で以下のように定義されています。

```
SerialThread = SerialThreadGroup.new
```

実は SerialThread はクラスではなく、単なる SerialThreadGroup の一インスタンスに過ぎないのです。 SerialThreadGroup のインスタンスは、自前で作成することができます。それの new メソッドを呼べば、SerialThread とは別に、直列実行されるブロック郡を作成することができます。 **SerialThreadGroup#max_threads=** を使えば、内部で並列に実行するスレッド数を設定することでセマフォのようなことができます。

```
group = SerialThreadGroup.new
group.max_threads = 2
group.new{ open("http://twitter.com/", &:read) }
group.new{ open("http://facebook.com/", &:read) }
group.new{ open("http://mixi.jp/", &:read) }
```

以上のコードは、Twitter と Facebook と mixi に同時にアクセスしています。しかし、**group** に束縛されている SerialThreadGroup は 2 つまでしか同時に実行できないことになっているので、 まず Twitter と Facebook にアクセスし、どちらかが終わったら mixi にもアクセスします。HTTP コネクションを同時に 2 つまでしか作成しないようになるわけです。 デフォルトでは一つずつですが、2 以上にすると順番の保証がなくなることに注意してください。

2.3. Reserver

特定の時間や、一定時間後にブロックを処理したいことがあります。そういう時は Thread と sleep を使うのではなく、Reserver を使うと若干簡潔に書くことができます。

```
Reserver.new(5){ p "5 秒後に実行" }
Reserver.new("2013/03/09 0:00:00"){ p "ミクの日" }
Reserver.new(Time.now + 5){ p "5 秒後に実行" }
```

Reserver の第一引数に、数値を指定するとその秒数経過してから、文字列や Time オブジェクトを渡すとその時刻にブロックを実行します。 用途が限定的な機能ですが、何個タイマーを登録してもスレッドを一つしか生成しないというメリットがあります。 ただし、ブロックが実行される時に新しいスレッドを作成してそこで実行するので、注意が必要です。

2.4. Delayer

Delayer は、単にあとで処理を実行するためのクラスです。最も単純な使い方は Thread に似ています。

```
Delayer.new{
  # あとで
}
```

上記のようなコードを書くと、コメント部分はメインスレッド上の他の処理が終わってから実行されます。 また、これは必ずメインスレッドで実行されるので、サブスレッドの処理の過程で UI を操作する必要が出てきた時などに、単に処理をメインスレッドに飛ばすためにも使えます。そのために、 **Delayer.new** はスレッドセーフになっています。

```
Reserver.new(5) { # 5秒後に
  Delayer.new{
    # UI等を操作
  }
}
```

Gtkの操作は必ずメインスレッドで行う必要があるので上記のような書き方をする必要があります。ただし、0.2から実装されたUI DSLは任意のスレッドから操作しても問題ありません。とはいえ、スレッドを使うことが稀ならば、Delayerを直接使うことも稀でしょう。しかし、イベントなどもDelayerを使って実装されているので、かなり大きな要素です。

3. イベント

プラグインは、ある種類のイベントが起こった時に呼び出されるブロックを好きなだけ登録できます。

```
Plugin.create :test do
  # 定義
  on_test do |arg|
    # testイベントが発生した時
  end
end

# 呼び出し
Plugin.call(:test, 1)
```

プラグインDSLの中で、 **on** なんとか と書けば、なんとかイベントが発生した時にブロックが実行されます。また、上の例のように on のあとにアンダーバーを入れることもできます。 on_test ブロックの引数は、イベントが呼ばれた時の引数です。イベントは **Plugin.call** で発生させることができます。例では引数 **arg** には 1 が渡されるでしょう。 mikutter自体のソースコードから、 **Plugin.call** の実装を見てみましょう。

```
# mikutter/core/plugin.rb
  def call(event_name, *args)
    Delayer.new{
      filtered = filtering(event_name, *args)
```

```
    plugin_callback_loop(@@event, event_name, :proc, *filtered) if filtered } end
```

plugin_callback_loop はプラグインのコールバック呼び出しですね。ここで重要なのはメソッドの中身がすべて **Delayer.new** でラップされていることです。 **Plugin.call** は Delayer にブロックを登録してすぐさま処理を戻します。そして、Delayer ブロックが実行されたら中身（イベント呼び出し）が実行されます。 このことがわかればイベントについてほとんど理解したようなものです。Delayer ブロックの中ですべてのコールバックが呼ばれるということは、 イベントのコールバックはすべてメインスレッドで実行されるということですし、 **Plugin.call** 自体はスレッドセーフなので、どのスレッドからでもイベントを発生させることができます。

4. フィルタ

フィルタには主に二つの使い方があります。イベントの引数を改変するという使い方と、単なる値の加工です。

4.1. イベントの引数を改変する

```
Plugin.create :test do
  filter_mention do |service, messages|
    [service, messages.reject{ |msg| msg.to_s.include?("ておくれ") }]
  end
end
```

上記のコードは、メンションのうち、「ておくれ」という単語が入ったツイートを全て除去します。私はておくれではないのでね。

　戻り値に注目してください。reject メソッドを使ってツイートを幾つか削除していますが、それ以外はそのまま引数が全部入った配列を返しています。 これがフィルタという名前の理由で、mention という「イベント」が発生した時に、そのイベントに本来渡された引数を受け取り、加工して実際のイベントに渡すという役割を持っています。 つまり、mention イベントが発生したら、実際に mention を待ち受けているコールバックは、本来のイベントの引数ではなく、フィルタの戻り値を受け取ることになるのです。
mention イベントは、「自分」へのメンションが全て流れてくるイベントです。mikutter のリプライタブは単純にこのイベントに流れてきたツイートをすべて表示するので、 ここで削除してしまえばリプライタブから消えるというわけです。

4.1.1. 実装を見てみる

さて、ここで前節「イベント」で引用した **Plugin.call** の実装をもう一度見てみましょう。3 行目で **Plugin.filtering** というメソッドを呼び出しています。これはフィルタの呼び出しです。 確かに、イベントと同じ名前のフィルタを呼び出していますね。そして、**Plugin.call** 自体に渡された引数ではなく、filtering の戻り値(＝フィルタの戻り値)を引数として採用しています。

イベントをキャンセル ここで注目されたいのはフィルタ結果が真かどうかを確認しているということです。もし偽を返せばイベントの発生をそもそもなかったことにできます。とはいってもフィルタが nil や false を返してはいけません。イベントの発生をなかったことにしたい場合は **Plugin.filter_cancel!** を使いましょう。 めったに使うこともありませんが、具体的な使用例は plugin.rb に定義されているフィルタにあります。

```
filter_favorite do |service, user, message|
  Plugin.filter_cancel! if favorites[user[:id]].include? message[:id]
  favorites[user[:id]] << message[:id]
  [service, user, message]
end
```

これは、2 回以上同じ人が同じツイートをふぁぼってきたら **Plugin.filter_cancel!** を呼んで、ふぁぼをなかったことにするという処理です。 Twitter の不具合で複数回同じイベントが送られてくるというのは稀によくあるので、このような処理が入っています(この原稿を書いてるたった今、別の Twitter クライアントで発生しました。mikutter なら切り捨てられたことでしょう)。

4.2. 値を加工する

もうひとつの使い方は、直接フィルタを呼ぶというものです。通常は先に述べた通りイベントの引数を加工したり、そもそも実行をキャンセルするために使いますが、フィルタだけを直接呼ぶこともできます。具体的には、どこでもいいので以下のように書けば **show_filter** というフィルタが呼べます。

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

show_filter は mikutter 内部で暗黙的に予約されているフィルタで、引数として Message の配列を受け取り、UI 上に表示してはいけないものを取り除いて返します。mikutter 標準機能では、ユーザをミュートにすると **show_filter** で切り落とされるようになります。 実際に使ってるところも見てみましょう。

```
# mikutter/core/mui/gtk_timeline_utils.rb
module Gtk::TimeLineUtils
  def add(message)
    if message.is_a?(Enumerable) then
      self.block_add_all(Plugin.filtering(:show_filter, message).first)
    else
      m = Plugin.filtering(:show_filter, [message]).first.first
      self.block_add(m) if m.is_a?(Message) end
    self end
end
```

タイムラインに Message を入れる処理です。5 行目と 7 行目がその処理です。分かれているのは、このメソッドが Message 以外に Enumerable を渡されたら、 すべての要素を追加するという処理があるからです。Message か Enumerable かで呼び出す内部関数を分けていますが、その前に **show_filter** が入っています。 これによってミュートされたユーザのツイートが一切表示されないようになっています。 いちいち戻り値の first メソッドを読んでいることに注意してください。フィルタの戻り値は与えた引数を加工した配列なので、 実際に必要な要素をこうやって取り出してやる必要があります。また、このフィルタは通知プラグインなど、 すべてのツイートを表示しうるプラグインでも利用されてるということも付け加えておきます。例えばあなたが全く別の種類の通知プラグインを作った時、 **show_filter** を使うことができます。またあなたが新しい種類のミュートプラグインを作った場合も、 **show_filter** に新しいコールバックを追加すべきです。 そうすれば、別々の人が開発したプラグイン同士が、フィルタを通じてうまく連携することが出来るでしょう。

4.2.1. フィルタのコールバックが呼ばれるスレッドは不定

イベントは必ずメインスレッドで呼ばれていましたが、上記の例を見ても分かる通り、フィルタは **Plugin.filtering** が呼び出されたスレッドで実行されます。 イベントのフィルタは今のところ必ずメインスレッドで呼ばれていますが、それにしたってフィルタだけを呼び出すことは可能なので、一応そのつもりで書きましょう。

5. まとめ

mikutter の様々な構成要素のなかで、お世話になる頻度が高い機能をとりあげて紹介しました。 この他にもプラグインを支援する機能がたくさんあり、そのすべてがプラグインに対してひらけているということは開発してる人ならよくご存知だと思います。それはプラグインデベロッパに自由を与えるためではなく、単にアプリケーションとしての mikutter が、mikutter というフレームワークの中で動くプラグインの集合体に過ぎないからです。 ほかにもいろんな要素があって、それら全てを詳しく紹介するには至りませんでしたが、少しでも mikutter についての理解の助けとなったなら幸いです。

【編者註】
としぁ氏は、
OSC KYOTO(2012 年 8 月)の所謂「ておくれ演説」にて、

「mikutter は環境！！」　と発言している。

お礼

今一瞬でも ShootingStar の機能が思い浮かんだやつは　@haru067 にふぁぼ爆撃して中指飛ばした上で ShootingStar の薄い本でも書いてろ。

vol.1 からこの執筆時点までで、Amazon ギフト券による支援が累計 23838 円も受けている。平素より格別のご高配を賜り、感謝の尽きぬ次第である。

金銭面での支援者、執筆者各位、読者、印刷所に伝えきれない感謝をふぁぼという星に託して―― *@brsywe*

――校正者 @ch_print 後書き――

あ、またでおくれた！！！

（誤植等あれば Twitter にて @ch_print までご指摘ください。）

――次号予告――

原稿提出期限：6 月末日

発行予定：OSC KYOTO　または　C84（通るといいなｱﾍｱﾍ

問合せ先： @brsywe

奥付

発行日：2012 年 12 月 31 日　（コミックマーケット 83）

版：PDF 版：2013 年 1 月 23 日

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

頒布協力：らぼちっく；げーと

ご意見・ご感想はお気軽にどうぞ。

twitter ハッシュタグ #人類は手遅れました

mikutter の薄い本制作委員会ウェブページ

http://home1.tigers-net.com/brsywe/mikutter.html

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。