# mikutterの薄い本 vol.15

## ておくれ！2　Teokure Celebration

# mikutter のスマホ通知

@yuzumone

## 1. はじめに

お久しぶりです，ゆずもねです．前回は「夢の国でもてあそばれ」というタイトルで，舞浜にあるテーマパークのアトラクションの待ち時間などの情報を mikutter で確認できるようにしたプラグインについて寄稿をさせていただきました．またこの薄い本に寄稿できることは最高の喜びです．さて，今回は最近実装したモバイル通知プラグイン mikutter_fcm と Android アプリの mikutter_fcm_for_android について述べていきたいと思います．

## 2. これまでのモバイル端末への通知

これまで私は mikutter_pushover と mikutter_pushbullet の 2 つの通知プラグインを実装し，公開しています．それぞれ名前の通り，Pushover と Pushbullet の 2 つの Push 通知サービスを利用したプラグインです．しかしながら，上記のプラグインは以下のような問題点がありました．

まず, Pushover と Pushbullet ともに受け取れる通知の数に上限が設定されています．Pushover はアプリが有料ということもあり上限数はそれなりですが，Pushbullet の場合 Free だと 1 か月 100 通の通知しか受け取ることができません．これではふぁぼ爆されるようなておくれだと一瞬で上限を超えてしまいます．

次に，実装されたのが World 導入以前ということです．mikutter_pushover，mikutter_pushbullet ともに Twitter のことしか考慮されておらず，かつ通知されるイベントも Mention，Favorite，Retweet の 3 つのイベントに限定されていました．今や mikutter は World 系プラグインによって独自の World が定義でき，すでに Mastodon や Slack プラグインが公開され，多くの人に利用されています．

最後にモバイル端末へ通知したいものは人それぞれに異なる，という点です．上記で述べた通り，これまでの通知プラグインでは Mention，Favorite，Retweet の 3 つのイベントを否応なしに通知します．Mention だけというような柔軟性は一切ありません．また，mikutter の特徴の 1 つであるプラグインの作りやすさから多くのプラグインが作成されています．プラグインでできることが増えれば増えるほど通知できるイベントは増えますが，通知したいイベントは人それぞれです．例えば，舞浜の某テーマパーク好きな人であれば，ショーやパレードの情報が更新されたら mikutter からモバイル端末に通知を送りたい！と思うのは自然なことでしょう．

## 3. これからのモバイル端末への通知

前置きが長くなりました．ここからが本題です．このようにこれからのモバイル通知プラグインに求められている要件は Push 通知上限数のない方法でかつ，ユーザが自ら通知したいイベントを選択し，簡単にモバイル通知が実現できることと考えました．そこで mikutter_fcm ではモバイル端末に通知を送るイベントを提供し，送信された通知を受け取る Android アプリとして mikutter_fcm_for_android を開発しました．名前の通り mikutter_fcm は Google の Firebase Cloud Messaging を利用してモバイル端末へ通知を送信します．Firebase Cloud Messaging は無料でかつ，送信上限数が設定されていません．これを利用することでふぁぼ爆をされるようなておくれでも通知を受け取り続けることが可能となります．また，今回通知を送信するイベントというかたちで実装しました．これによってどのプラグインからでもイベントを呼び出すことで通知を送信することが可能となり，ユーザは自分が通知したいものを好きなプラグインから fcm イベントに渡すだけ良いのです．

さて，ここからはコードを示しながらユーザがどのように fcm イベントを使用できるかを説明していきたいと思います．まず，README にも書いてある mikutter 起動時に通知を送信する例を以下に示します．

```
on_boot do
  data = {
    :title => 'on_boot',
    :body => 'mikutter が起動(^0^)／',
    :url => 'https://mikutter.hachune.net/'
  }
  Plugin.call(:fcm, data)
end
```

Plugin.call(:fcm, data) が fcm イベントを呼び出しています. data のところは Hash もしくは Diva::Model インスタンスを渡すことができます. 注目してほしいのは通知送信のために何かを require する必要はなく，ユーザがすることは fcm イベントを呼び出すだけで良い，ということです．

次に，mikutter_tdr でショーの情報を通知する例を以下に示します．通知を送信するためにまずショーやパレードの情報が更新された呼ばれる tdr_show イベントを定義しました．mikutter_tdr では 5 分毎にショーやアトラクションの情報を取得していますので，tdr_show イベントも 5 分毎に呼ばれてショーやパレードの情報が Array でやってきます．modified には情報が更新された時間が入っているので，コードでは modified が 5 分以内のショー情報を通知しています．Diva::Model インスタンスの場合はそのまま fcm イベントに渡すだけで良しなに通知が送信されます．

```
defevent :tdr_show, priority: :ui_passive, prototype: [Array]
on_tdr_show do |data|
  # 5 分毎に情報が更新されるので on_tdr_show も 5 分毎に呼ばれる
  t = Time.now - 5 * 60
  data.each do |show|
    if show.modified > t
      Plugin.call(:fcm, show)
    end
  end
end
```

mikutter_tdr から通知が送信された様子を以下に示します．図 1 が通知の例，図 2 がアプリ起動時の通知履歴一覧になっています．

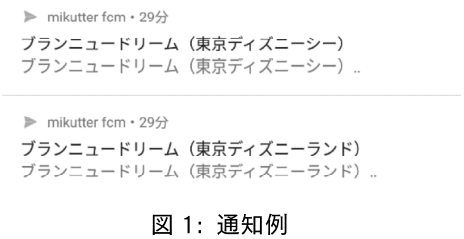


図 1: 通知例


図 2: 通知履歴一覧例

## 4. おわりに

新しい時代のモバイル端末通知プラグイン mikutter_fcm と通知を受け取る Android アプリケーションの mikutter_fcm_for_android を紹介しました．mikutter の特徴はなんといってもプラグインの作りやすさです．プラグイン作成時に mikutter_fcm がお役に立てば幸いです．

## 参考

mikutter_fcm: https://github.com/yuzumone/mikutter_fcm
mikutter_fcm_for_android: https://github.com/yuzumone/mikutter_fcm_for_android

# Diva, World, Spell, Score, and Worldon

金具 @cobodo@mstdn.kanagu.info

## 1. はじめに

はじめまして。金具です。

オープンソースのマイクロブログ実装である、Mastodon[*1]向けの mikutter プラグイン、Worldon[*2]を作ったので、紹介したいと思います。

## 2. Mastodon とは

Mastodon は、Eugen Rochko[*3]氏により開発された、Twitter ライクなマイクロブログ分散プラットフォームです。各サーバのことを「インスタンス」と呼び、実装詳細に依存せずにインスタンス間を接続する、OStatus や ActivityPub[*4]といったプロトコルに対応していて、GNU Social[*5]、Pleroma[*6]、Misskey[*7]等のマイクロブログ実装と相互にメッセージを送信し合うことができます。こうして繋がったインスタンスの集合を、fediverse（連合）といいます。

中でも Mastodon は、Web 上から閲覧するためのデフォルト UI が、TweetDeck 風のモダンなものであることから、何らかの事情で Twitter 以外のプラットフォームを求めたユーザーに人気を博し、比較的多くのユーザーを得ています[*8]。

現在では多数の日本人エンジニアも開発に加わり、活発なアップデートが続けられていま

1 https://joinmastodon.org

2 https://github.com/cobodo/mikutter-worldon

3 @Gargron@mastodon.social

4 https://www.w3.org/TR/activitypub/

5 https://gnu.io/social/

6 https://pleroma.social/

7 https://joinmisskey.github.io/ja/

8 もっとも、前述のプロトコルにより連合を組むので、各ユーザーがどのサービスに所属しているか意識することはほぼありません。

す。 ただ、Mastodon の話は Mastodon でされることが多いためか、Twitter ユーザーにはあまり状況が理解されていないようです。

## 2.1. Twitter との比較

Twitter の tweet にあたるものは、Mastodon では toot（トゥート）と呼び、retweet は boost（ブースト）[*9]と呼びます。 このあたりは用語が違うだけで、 特段意味があるわけではありませんが、この文書では Mastodon の話をするので、Mastodon での用語を使います。

Mastodon は、Twitter にもあるホームタイムライン (ホーム TL； HTL) やリストタイムラインの他に、ローカルタイムライン（ローカル TL； LTL）と連合タイムライン (連合 TL； FTL) を持っているのも特徴です。 ローカル TL は所属するインスタンスに所属するユーザーの公開 toot すべてを、連合 TL はそのインスタンスに所属しないユーザーのものも含めて観測したすべての toot を、それぞれ流します。初期の Twitter にも Public Timeline という全ユーザーの tweet が流れるタイムラインがありましたが、 これは、twitter.com というインスタンスのローカル TL である、という見方もできるでしょう。 ローカル TL と連合 TL を合わせて、公開タイムライン（公開 TL）とも呼びます。

Twitter では、tweet の公開 / 非公開や、NSFW フラグの設定はアカウントごとに行いますが、Mastodon では toot ごとに個別に指定できます。公開度には以下の 4 段階があります。

- Public（公開；誰でも閲覧可＆公開 TL に載る）
- Unlisted（未収載；誰でも閲覧可＆公開 TL には載らない）
- Private（非公開；フォロワーのみ閲覧可）
- Direct（ダイレクト； mention 先の相手にのみ閲覧可）

公開 TL ではこのうち Public のものしか流れてこないため、Twitter と同じように気になるユーザーをフォローして、ホーム TL でやり取りすることが多いようです。

https://friends.nico のように、 ローカル TL でやり取りすることがメインになっているインスタンスも存在します。

toot ごとに注意書きをつけることもできます。この機能を Content Warning(CW) と呼び、

9 C++ の著名ライブラリである Boost と紛らわしいので、私はカタカナで表記するのを好みます。

R-18 だと考えられるテキストであったり、ネタバレであったり、あるいは単に長文であったりと、閲覧に注意を要する場合に警告文を記載することができます。この CW テキストが設定されている場合、WebUI では（NSFW フラグが設定された画像のように）本文が隠された状態で表示され、クリックで内容を確認することができるようになります。

## 2.2. mikutter プラグインから見た Mastodon

mikutter プラグインを書こうとしたときの Mastodon の特徴は、以下のようなものが挙げられると思います。

(1) Twitter 相当の機能

- ふぁぼ[*10]
- ブースト (Retweet)
- マルチアカウントによる投稿
- 公開範囲（Twitter ではアカウント単位だが Mastodon ではトゥート単位）
- 添付画像
- NSFW フラグ（Twitter ではアカウント単位だが Mastodon ではトゥート単位）
- プロフィール
  - アイコン
  - 名前
  - 紹介文
  - ピン留め toot（固定 Tweet）

(2) Twitter には無い独自の機能

- Content Warning
- インスタンス固有のカスタム emoji を含むメッセージ表示

10 ふぁぼは、ふぁぼ。

(3) 任意のドメイン上で稼働しており、特定のドメイン・運営元に依存しないし、できない

- URI が与えられていても、本当に Mastodon インスタンスかどうかは接続してみるまでわからない。
- Twitter における API key の取得も、各インスタンスごとに API 経由で行なう自己申告制である。
- 必要なストリームは各インスタンスに個別かつ同時にコネクションを張り続ける必要がある。

(4) ユーザーが複数のドメインにまたがって複数のアカウントを持つことが珍しくない。

このため、mikutter に搭載された機能のうち、Delayer や Pluggaloid はもちろんのこと、twitter のために用意され、その後汎用化されたような機能はほとんど網羅的に使うことになります。

2017 年 12 月 25 日にリリースされた mikutter 3.6 以降、mikutter における Twitter 機能はプラグインとなりました。これは、Twitter に関する依存なしで、mikutter のほとんどの機能を使える、ということを意味しています。

Worldon 以前にも、Mastodon プラグインは存在しました。mikutter 丼[*11]と、mikutodon[*12]です。しかし、これらのプラグインでは mikutter 3.6 以降の新機能、特に World への対応は行われませんでした。そこで、最初から World 機能に対応し、また過去のプラグインへのリスペクトを示す意味で、World の don、すなわち Worldon と命名しました。

# 3. Worldon の実装

## 3.1. 経過

以下のような過程を経ました。多分。

(1) カスタムモデルを表示する実験をする。

11 https://github.com/moguno/mikutter-don

12 https://github.com/sora0920/mikutodon

(2) Ruby で WebSocket ストリームを受信する実験をする（JSON を文字列としてコンソールに流すだけ）。

(3) ドキュメントの Entities[*13]を見て、それぞれに対応する Diva モデルを全部書く。

(4) ストリームでカスタムモデルを流す。

(5) `world_setting` を作る。

(6) Entity class を作る。

(7) spell を作る。

(8) `MessageMixin` の各メソッドを順次オーバーライドして機能を増やしていく。

(9) 設定画面を作る。

(10) （この頃、うんこでたボタンに採用される）

(11) subparts で toot の公開度を表示し始める。

(12) WebSocket を捨てて ServerSentEvents で受信するように変更する。

(13) ストリームの代わりに REST で定期取得することもできるようにする。

(14) カスタム投稿機能を作る。

(15) Score 対応。

実際には行きつ戻りつといった風に、mikutter のソースを読みながら、 時には toshi_a さんにアドバイスをもらいつつ開発を進めました。ここでは Worldon の開発を進める上で理解したものごとを紹介していこうと思います。

## 3.2. タイムラインの管理

mikutter でタイムラインを作るには、大きく分けて 2 つの方法があります。1 つは `gui` プラグインの `tab` DSL や `timeline` DSL を利用して、 タブやタイムラインのオブジェクトを作り、自力で管理する方法、もう 1 つは `extract` プラグインの抽出タブを利用して、datasource のみを提供する方法です。

サードパーティプラグインを作成する場合には第 3 の道もあります。つまり、他のプラグインが用意したタイムラインオブジェクトや、datasource に勝手に差し込む、 という方法です。

Worldon では datasource と、`update` や `appear` に類似した、`worldon_appear_toots` イ

13 https://docs.joinmastodon.org/api/entities/

ベントを用意する、という方針をとりました。自前のタイムライン管理はせず、`extract` プラグインに任せて、しかもサードパーティプラグインによる拡張が可能なように、このようにしてあります[*14]。

## 3.3. ストリーム

mikutter は、Twitter に対するストリーム受信に対応していますが、汎用的なものではないので、サードパーティプラグインは独自にストリーム処理を行う必要があります。

Mastodon の場合、 すべてのタイムラインはストリームも REST も、 両方提供されており、 しかも通信方式に WebSocket と ServerSentEvents(SSE) の 2 種類が用意されています。Worldon では、まずストリームを実装し、その後 REST によるポーリングにも対応する、という経過をたどりました。

Mastodon に対するストリーム受信を最初に実装した mikutter プラグインは、mikutodon でした。mikutodon では、`eventmachine` gem と `faye-websocket` gem を利用して、WebSocket ストリームを受信していたようでした[*15]。Worldon も、当初はそれを真似て WebSocket で受信していましたが、以下のような問題がありました。

(1) `eventmachine` はイベントの発生とハンドラの呼び出しを行なう gem で、singleton として実装されている（ように見える）ため、複数並立させることができない。
(2) `faye-websocket` 自体が `eventmachine` に依存していて、 複数のストリームを扱うことが余計に困難になっている。
(3) `faye-websocket` の代替として使えるものに `websocket-client-simple` gem があるが、全くメンテナンスされていない[*16]。

(1) が特に致命的で、1 つのインスタンスの 1 つのアカウントを取ってみても、 ホーム TL とローカル TL の 2 つを同時受信したいし、まして複数のアカウントを利用するとなれば、複数のストリーム受信は当然必要な機能です。

しかし、(1) は最も解決しやすい問題でもありました。**「`eventmachine` はイベントの発生とハンドラの呼び出しを行なう gem」** です。 我々はそれを既に持っていますね？そう、

---

14 とはいえ、 執筆時点で API ドキュメントは全く書いていないため、 そういったサードパーティプラグインは存在しないのではないかと思います。

15 恐らく https://qiita.com/rerofumi/items/894cde1f57357d2c8479 を参考にしていたと思われます。

16 単に更新がないだけでなく、issue も pull-req も放置されていました。

pluggaloid gem[*17]です。eventmachine は捨てましょう。

(2)(3) に関しては、 要するに WebSocket の仕様が複雑なのだろう、 と思います。faye-websocket も websocket-client-simple も、websocket gem の wrapper となっているので、 自前で websocket gem を使ってストリーム処理を行うという方法もあるでしょう。その場合、websocket gem に直接依存することになります。

ここまでがわかった段階で、mikutodon が SSE でストリーム処理をする実験をしている話が流れてきました。調べてみると、以下のように WebSocket よりも扱いやすそうな仕様でした[*18]。

(1) WebSocket は HTTP コネクションからアップグレードした別のプロトコルを喋る必要があるが、SSE はただの HTTP コネクション。
(2) WebSocket は双方向のデータ通信で、Mastodon の用途でも ping に対する pong を返す必要があるが、SSE はサーバからクライアントへ一方的にデータを流すだけで済む。

そこで、Pluggaloid ベースで動く汎用の SSE クライアントプラグインを自作しました。それが sse_client プラグインです。これは独立したプラグインとしても使えるように作ってありますが、WebSocket を捨てた時点で外部 gem への依存をなくせたことから、Worldon としては手軽にインストールできる「全部入り」を志向することにして、Worldon に同梱することにしました[*19]。 もし将来的に、 依存関係の記述を元にして他のプラグインを動的にインストールできるような仕組みが mikutter に標準搭載されることになれば、分離することになると思います。

sse_client プラグインは、sse_ プレフィクスを持つイベントを用意しており、sse_stream.rb の中で使っています。 ドキュメントなどは全く書いていないので、 興味のある方は読んでみてください。

## 3.4. Diva モデル

かつての mikutter では Twitter 用の Message クラスしかなかったようですが、Retriever を経て、 現在では Diva::Model が利用できます。 基本的な使い方は、Writing mikutter

17 正確には Delayer も含みます。

18 https://developer.mozilla.org/ja/docs/Server-sent_events/Using_server-sent_events#Event_stream_format

19 sse_client.rb

plugin(以下、Wmp) の Reference[*20]に記載されています。 まだ Retriever で記載されているものもありますが、単に Diva で読み替えれば OK です。

特に、 入れ子の Hash をコンストラクタに渡せば、 そのまま入れ子の Diva::Model にできる機能が便利です。

例えば Mastodon では Status エンティティが account という名前で Account エンティティのオブジェクトを持っていますが、Status エンティティに対応する Plugin::Worldon::Status を Diva::Model を継承して作っておき、 コンストラクタに（入れ子になったままの）Hash オブジェクトを渡してやれば、Account に対応する Plugin::Worldon::Account に自動的に変換されます。

Web API のレスポンスでは、 このように基本的なエンティティを定義しておいて組み立てるような構造になっていることが多いと思うので、mikutter 3.6 以降ではこういったプラグインは格段に作りやすくなっていると思います。

また、Diva::Model::MessageMixin や、Diva::Model::UserMixin を include すると、メッセージとして表示するモデルや、そのメッセージに属するユーザー情報を表すモデルが作れます。 これらは何をしているかというと、mikutter から呼ばれる可能性のあるメソッドをスタブとしてあらかじめ実装してあるものです。 ここにあるメソッドをオーバーライドしていくと、mikutter の機能を使えるようになるわけです。

例として、 返信先のメッセージをプレビュー表示する、Gdk::ReplyViewer を見てみます。これは mikutter 本体の core/mui/cairo_replyviewer.rb にあり、Gdk::SubPartsMessageBase を継承しています。

コンストラクタを見ると、

- message.has_receive_message?
- message.replyto_source_d

の 2 つのメソッドを呼んでいることがわかります。

core/lib/diva_hacks/mixin/message_mixin.rb を見ると、has_receive_message? メソッドが定義されていて、常に false を返すようになっています。

20 https://reference.mikutter.hachune.net/reference/

`replyto_source_d` メソッドは `MessageMixin` には定義されていません。`has_receive_message?` が `false` を返せば、`replyto_source_d` が呼ばれることはないからです。この 2 つのメソッドを定義してやれば、ダックタイピングによってそれが `Gdk::ReplyViewer` によって呼び出され、返信先メッセージが表示されるわけです。

このようにして、カスタムモデルを mikutter 上に表示させ、かつインクリメンタルに機能を追加していけるのです。

## 3.5. World

`world` は mikutter 3.6 で導入されたプラグインです。`world_shifter` や `current_world` 等のプラグインと協調して動作しますが、`world` 自体は、基本的には Diva モデルのシリアライズ・デシリアライズをして、ファイルに保存したり復元したりするだけのものです。

`world` は、mikutter の起動時に所定のディレクトリから暗号化されたファイルを読み出し、シリアライズされた Diva モデルを Ruby のオブジェクトとして復元します。この際、必要なプラグインが入っているかどうかの判定を Diva モデルの slug で行います。つまり、保存されている Diva モデルの slug と同じ slug を持つプラグインがロードされていなければ[*21]、そのプラグインをロードしてから、オブジェクトの復元を行います。このため、World モデルの slug は、必ずそのプラグインと同じ slug を持つ必要があります。

この機能は、アカウント管理を Twitter 非依存にするためのものだと紹介され、サードパーティプラグインでもマルチアカウントが使えるようになる、というものですが、解説記事がまだあまり出ておらず、対応に苦慮されている開発者の方も見かけました。ここでは私の理解が及ぶ範囲において、簡単な説明をします。

World 自体は単なる Diva モデルのシリアライザと考えればいいのですが、`world_setting` DSL が曲者なのだと思います。

`world_setting` について理解するには、まず `settings`(Form) DSL について理解し、次に `dialog` DSL について理解する必要があります。

21 world 系プラグインは、性質上 `world` プラグインに依存するため、`world` プラグインのロード時にはまだロードされていないことになります。

### 3.5.1. settings(Form) DSL

settings DSL は、Wmp でも紹介されています[*22]。また、あひるさんがまとめたリファレンスがあります[*23]。

これは mikutter の設定ダイアログに画面を作成し、それと UserConfig を対応させる仕組みです。改行を含まないテキストを入力させるなら、以下のように書きます。

```
Plugin.create(:sample) do
  settings "サンプル" do
    input('テキストを入力してください', :tekisuto)
  end
end
```

この場合、設定ダイアログのメニューに「サンプル」という項目が出現し、選択すると「テキストを入力してください」というラベルとともにテキストボックスが出現します。テキストボックスになにがしか入力すると、UserConfig[:tekisuto] に入力されたテキストが保存されます。

このようにして、フォーム的なものを簡単に作ることができます。

### 3.5.2. dialog DSL

dialog DSL は、Form DSL を設定とは無関係なダイアログで使う機能です。いきなりサンプルを出します。mikutter コンソールで実行できます。

```
Plugin.create(:sample) do
  dialog "サンプル" do
    input('テキストを入力してください', :tekisuto)
  end.next do |result|
    # result[:tekisuto] に入力されたテキストが入る
    puts result[:tekisuto]
    $stdout.flush
  end
```

22 https://reference.mikutter.hachune.net/basis/2018/02/10/settings.html

23 https://github.com/Na0ki/mikutter_doc/blob/master/settings.adoc

```
end
```

dialog の中身は settings と全く同じであることがわかります。このようにすると、設定ダイアログの代わりに、単独のダイアログが表示され、settings の場合と同様に、ラベルとテキストボックスが表示され、「OK」を押すと入力された値が取れます。ただし、入力が保存される先は、UserConfig ではなく、.next に渡すブロックの引数である result です。

この .next は Delayer::Deferred の next です。「キャンセル」でダイアログを閉じた場合は、fail 扱いとなり、next に渡したブロックは実行されません。

値を受け取る別の方法として、await_input を使う方法があります。

```
Plugin.create(:sample) do
  dialog "サンプル" do
    input('テキストを入力してください', :tekisuto)
    result = await_input
    label "入力は #{result[:tekisuto]} でした"
  end
end
```

この場合、await_input より前までの部分でダイアログが描画されます。「OK」を押すと await_input により入力状態が result に返され、次の画面で入力データを使って別のフォームを組み立てることができます。「OK」の代わりに「キャンセル」を押すと、次の画面には進まず、そのままダイアログが閉じます。

### 3.5.3. world_setting DSL

いよいよ world_setting です。world_setting には、World slug、world の名前、そしてブロックを渡します。そうすると、World モデル（＝ Diva モデル）を作成するための手続きとして、world プラグインに登録されます。

```
Plugin.create(:sample) do
  world_setting(:sample, 'サンプル') do
    input('テキストを入力してください', :tekisuto)
    result = await_input
    Plugin::Sample::World.new(tekisuto: result[:tekisuto])
  end
```

```
end
```

mikutter でアカウントの追加操作を行うと、まず `change_account` プラグインが `dialog` DSL によりダイアログを作成し、登録された World モデルのうちどれを使うか選択させる画面を作ります。そこで選択されたものが `await_input` により取得され、`world_setting` により登録されていた、 その World モデル用のブロックが `instance_eval` されます。 つまり、`change_account` が作成した `dialog` のオブジェクトを `self` としてみなすようにして、ブロックが実行されます。結果的に、2 画面目以降は `world_setting` に渡したブロックの記述により挙動が決まります。

ブロックは最後に World モデルのインスタンスを返す必要があります。ということは、その前に必要な情報を受け取る必要があるので、 必然的に `await_input` が 1 回は行われることになります。 その結果、 特に入力欄のない画面が必ず最後に入ります。`twitter` プラグインや、`worldon` プラグインでは、 認証したアカウントの情報を表示することで、 空白の画面を回避しています。

### 3.5.4. World を使う

登録された World オブジェクトは mikutter 起動のたびに `world` プラグインによってロードされ、`:worlds` フィルタや `:world_current` フィルタ、 あるいは互換性のために用意されている `Service.primary` などを通じて取得できるようになります。

とはいえ、World オブジェクトに何か必須の機能があるわけではありません。Worldon の場合は、 アカウントごとのドメインとアクセストークンを主に保管するようにしてありますが、他のフィールドやメソッドはオマケで、ここに置いておく必然性は存在しません。

World オブジェクトは、 アカウントの保管の他にもうひとつ、Spell の選択で大きな役割を果たします。次節では Spell を見ていきましょう。

## 3.6. Spell

Spell は `spell` プラグインで提供される機能です。`defspell` DSL で定義し、`Plugin.create` に渡すブロック内では通常のメソッドのように呼び出すことができます。Wmp でも紹介されています[*24]。

24 https://reference.mikutter.hachune.net/basis/2017/12/12/spell.html

誤解を恐れずに言えば、Spell は Diva モデルに対するオーバーロード機構を備えたメソッド定義です。`defspell` は第 1 引数に Spell 名、2 番目以降の通常の引数に Diva モデルの slug、その他の非 Diva モデル引数はキーワード引数で渡します。

最もよく定義されるであろう `compose` spell を例にして見てみましょう。実装はスキップするので、定義だけ見てください。まず Twitter プラグインの `compose`（の一部[*25]）がこれです。

```
defspell(:compose, :twitter) do |twitter, body:, to: nil, **options|
  # 宛先なしの投稿
end

defspell(:compose, :twitter, :twitter_tweet) do |twitter, tweet,
    body:, **options|
  # 発言への返信
end
```

次に Worldon の `compose` spell です。

```
defspell(:compose, :worldon) do |world, body:, **opts|
  # 宛先なしの投稿
end

defspell(:compose, :worldon, :worldon_status) do |world, status,
    body:, **opts|
  # 発言への返信
end
```

Twitter プラグインと Worldon を両方ロードした mikutter 上では、これらすべてが `compose` spell として登録されています。呼び出される際に実引数として渡された Diva モデルに応じて、いずれかの spell が選択され、動的にディスパッチされるのです。`compose` spell を呼び出しているのは、`Gtk::PostBox` です。

```
module Gtk
```

25 説明の都合上、`condition` は省いています。また、一部のオーバーロードも省いています。

```
  class PostBox < Gtk::EventBox
    def post_it
      @posting = Plugin[:gtk].compose(
        current_world,
        to_display_only? ? nil : @to.first,
        **compose_options
      ).next{
        destroy
      }.trap{ |err|
        warn err
        end_post
      }
      start_post
    end
  end
end
```

current_world は現在選択中の World を返します。第 2 引数は返信の場合は返信対象のメッセージモデル、そうでなければ nil になります。従って、アカウントの選択状態や返信先のメッセージモデルに応じて、以下のように呼び出し先の Spell が切り替えられます。

- :twitter World を選択中で、返信でない。
  → defspell(:compose, :twitter)
- :twitter World を選択中で、返信先が :twitter_tweet。
  → defspell(:compose, :twitter, :twitter_tweet)
- :twitter World を選択中で、返信先が :worldon_status。
  → 該当なし
- :worldon World を選択中で、返信でない。
  → defspell(:compose, :worldon)
- :worldon World を選択中で、返信先が :worldon_status。
  → defspell(:compose, :worldon, :worldon_status)
- :worldon World を選択中で、返信先が :twitter_tweet。
  → 該当なし

「該当なし」の場合、`Plugin::Spell::SpellNotFoundError` が `raise` されますが、`Delayer::Deferred` で包まれているので、対処が必要なければ無視することもできます。

これは `compose` に限った話ではありませんが、具体的なサービスに対するアクションを Spell として定義する際に、最初の手がかりとなるのが World です。World（として使っている Diva モデル）が決まれば、具体的なサービスと、そのサービス上のアカウントが決定される、という形にしておくと、シンプルになると思います。

サードパーティプラグインで Spell を生やすこともできます。例えば、`:worldon` World を選択中で、`:twitter_tweet` に対して返信する Spell を、以下のような形で書くことができます。

```
Plugin.create(:quoting_toot) do
  defspell(:compose, :worldon, :twitter_tweet) do |world, tweet,
      body:, **options|
    body = "#{body} #{tweet.uri}"
    spell(:compose, world, body: body, **options)
  end
end
```

この場合、返信先の tweet を URL 引用する形でトゥートを投稿します。これが mikutter の GUI 上で、「返信」コマンドを通じて実行できるわけです。

## 3.7. Score

Score は比較的新しい機能で、mikutter 3.7 で実装されました。Wmp にも説明があります[*26]。mikutter 本体における今後の機能拡張にも関わってきそうなこともあり、今後も仕様が変更される可能性が多々あります。

Score は、メッセージモデルの描画時に、ハイパーリンクの設定やカスタム emoji の設定をするための機構です。将来的には文字装飾なども可能になるかもしれません。

26 https://reference.mikutter.hachune.net/basis/2018/04/24/score.html

### 3.7.1. score_filter の形式

サードパーティプラグインからは、Score は score_filter を通じて定義します。

```
filter_score_filter do |model, note, yielder|
  yielder << score
end
```

この score_filter を通じて提案された Score の集合から、「Score 選定アルゴリズム」に基づいて適切なものを選択し、 それに含まれる Plugin::Score::TextNote を、 また score_filter で再帰的に展開し、最終的に 1 つの Score を使用して MiraclePainter が描画を行います。

とはいえ、なかなか思い通りに動かないことが多いのではないかと思います。
score_filter を書く際には、特に、以下の点に注意する必要があります。

#### 3.7.1.1. 特定のモデルに対する Score を定義する

第 1 引数のモデル slug が目的のものであり、かつ、トップレベルの呼び出しであるようにする。

```
filter_score_filter do |model, note, yielder|
  if model.class.slug == :target_model_slug && model == note
    yielder << score
  end
end
```

#### 3.7.1.2. モデルに依存しない汎用の Score を定義する

この場合は逆に、トップレベルの呼び出しを避ける。

```
filter_score_filter do |model, note, yielder|
  if model != note
    yielder << score
  end
end
```

score_filter は、必ずこのどちらかの形式で定義する必要があります。そうでない場合、大抵は予期せぬ結果を招きますが、逆にこれさえ守れば、かなりいろいろなことができます。

これを踏まえて、Worldon における Score について詳しく見てみます。

### 3.7.2. Worldon における Score の構造

Worldon では、score_filter を 1 つだけ用意しています。

```
filter_score_filter do |model, note, yielder|
  next [model, note, yielder] unless model == note
  next [model, note, yielder] unless
    (model.is_a?(pm::Status) || model.is_a?(pm::AccountProfile))

  if model.score.size > 1 || model.score.size == 1 &&
      !model.score[0].is_a?(Plugin::Score::TextNote)
    yielder << model.score
  end
  [model, note, yielder]
end
```

先程説明した通り、トップレベルかつ目的のモデルが対象である場合のみ働くようにしてあります。具体的な Score の構成は model.score に丸投げです。

model.score は Plugin::Worldon::Status のインスタンス変数 @score の attr_accessor です。コンストラクタで Plugin::Worldon::Parser.dictate_score によりあらかじめ用意しておきます。

#### 3.7.2.1. HTML の処理

Mastodon では、トゥートの内容は HTML で配信されます。nokogiri を使って処理してもよいのですが、ある程度パターンが絞られているため、Worldon では正規表現で処理しています。HTML には、本文や WebUI での表示の都合上、長い URL を省略するための  や  要素があったりします。Worldon では WebUI を尊重する形で、invisible に指定された範囲は Score に渡すテキストからは除去したり、ellipsis に指定された範囲は "..." に置き換えたりしています。

他にも、（本文データとは無関係に）添付画像の URL を本文末尾に追記しています。これは `sub-parts-image` プラグインのような機能を MiraclePainter と Score が提供するようになれば、そちらに切り替えるかもしれません。

CW テキストに関しては、単に `"\n----\n"` で本文と結合して、最初から見えている状態にしています。MiraclePainter の将来的な拡張によっては、最初は隠した状態にして、何らかのアクションで切り替えるような実装をするかもしれません。

こういった処理を加えた上で、プレーンテキストは `TextNote` にします。`<a>` 要素は以下の 3 種類に分けられます。

- 最初の文字、もしくは直前の文字が `'#'`
  → ハッシュタグであると見なして、対応する Diva モデル `Tag` にする。
- 中身のテキストが `mentions` に含まれる
  → アカウントであると見なして、対応する Diva モデル `Account` にする。
- それ以外
  →`HyperLinkNote` にする。

Score は Diva モデルのリストですから、Diva モデルである `Tag` や `Account` が直接置けますし、クリックすれば、そのオブジェクトが `:open` イベントに渡されます。`intent` を使って開き方を定義しておけば、mikutter 上で「開く」ことができます。

これら一連の処理が終わった段階で、HTML の除去が完了し、第一段階の Score が作成できます。

#### 3.7.2.2. Emoji の処理

Mastodon の特徴であるカスタム Emoji ですが、これは実は HTML 化されておらず、本文中に `:mikuslime:` のような形でプレーンテキストとして、投稿時の入力がそのままにされています。それとは別に、トゥートを表す `Status` オブジェクトに `emojis` というメンバ変数があって、これは `Emoji` オブジェクトのリストになっています。

もし `emojis` が空でない場合、それぞれの `Emoji` の探索を、HTML の除去が完了した第一段階の Score に含まれるすべての `TextNote` に対して行います。

見つかった emoji を `Emoji` オブジェクトで置き換え、前後のテキストをそれぞれ改めて

`TextNote` で包んで並べたもので、元の `TextNote` を置き換えます。`Plugin::Score::EmojiNote` も用意されていますが、カスタム emoji を表示するのに必ずしもこの Diva モデルを使用する必要はありません。そのモデルが `inline_photo` メソッドで Photo モデルを返すようになっていれば、カスタム emoji が描画できます。

このようにして最終的な Score が用意できます。

# 4. おわりに

Worldon を作成し始めてから、もうすぐ 1 年が経とうとしています。私自身、忘れかけていた部分もありましたが、git log や notestock[*27]のようなサービスを通じてログを漁り、ある程度思い出しながら、これを書くことができました。

振り返ってみて思うのは、やはり先達の存在あってこそだったな、ということです。mikutter 本体はもちろんのこと、mikutter の薄い本シリーズのバックナンバーや、様々なドキュメント、そして mikutter 丼に mikutodon といったプラグインのおかげで、Worldon は成り立っています。

この記事が、いつか誰かの役に立つものであればいいと、願っています。

27 https://notestock.osa-p.net/

あとがき　@brsywe

前号(vol.14)においては、Twitter を想いしのんだところであるので、今回は mikutter を取り戻すべく、今アツいお二方からの記事を掲載した。

としぁおじいちゃんは生理休暇です。

1 月末にははぴねす！2 がでるそうなのでよろしくお願いします。頭の中を軽くして、幸せな気持ちになれることを期待しています。


## ――次号予告と寄稿者募集――

## 原稿提出期限：6 月末

## 頒布予定：コミックマーケット 96

## 問合せ先：@brsywe , @ch_print

最近寄稿者が減少してきているので、まだ記事出したことないけれど！という方も是非どうぞ。

（但しいつも通り内容を精査します）

奥付

発行日：2018 年 12 月 31 日(コミックマーケット 95)：初版第一刷

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

印刷所：ラクスル

連絡先： brsywe　@　hotmail.co.jp

ご意見・ご感想は Amazon ギフト券のメッセージ欄にどうぞ。

mikutter の薄い本制作委員会ウェブページ

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

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