Emacs as a VM

A Virtual Machine called Emacs

pixiv Inc.
USAMI Kenta

pixiv

お前誰よ

- うさみけんた（@tadsan）/ Zonu.EXE / にゃんだーすわん
- ピクシブ株式会社 pixiv事業本部 エンジニア
  - 最近はピクシブ百科事典(dic.pixiv.net)を開発しています
- Emacs Lisper, PHPer
  - Emacs PHP Modeを開発しています（2017年-）
- TechFeed公認PHPエキスパート

私の立ち位置


ツイート
にゃんだーすわん
@tadsan
これは何度でも申し上げるのですが、軽率にエディタ戦争だと煽ったりVimmerはEmacserはと中傷するひとたちは全員軽蔑してます。
午後2:46 · 2018年9月16日
1 件のリツイート
6 件のいいね

pixiv

Emacs最新情報！

といいたいのですが

わかりやすい
ニュースとしての
変化は多くない

pixiv

Emacs30の開発は着々と進んでいます

AIなど使えるような拡張も登場している

さて

前回のおさらい

Emacs最前線

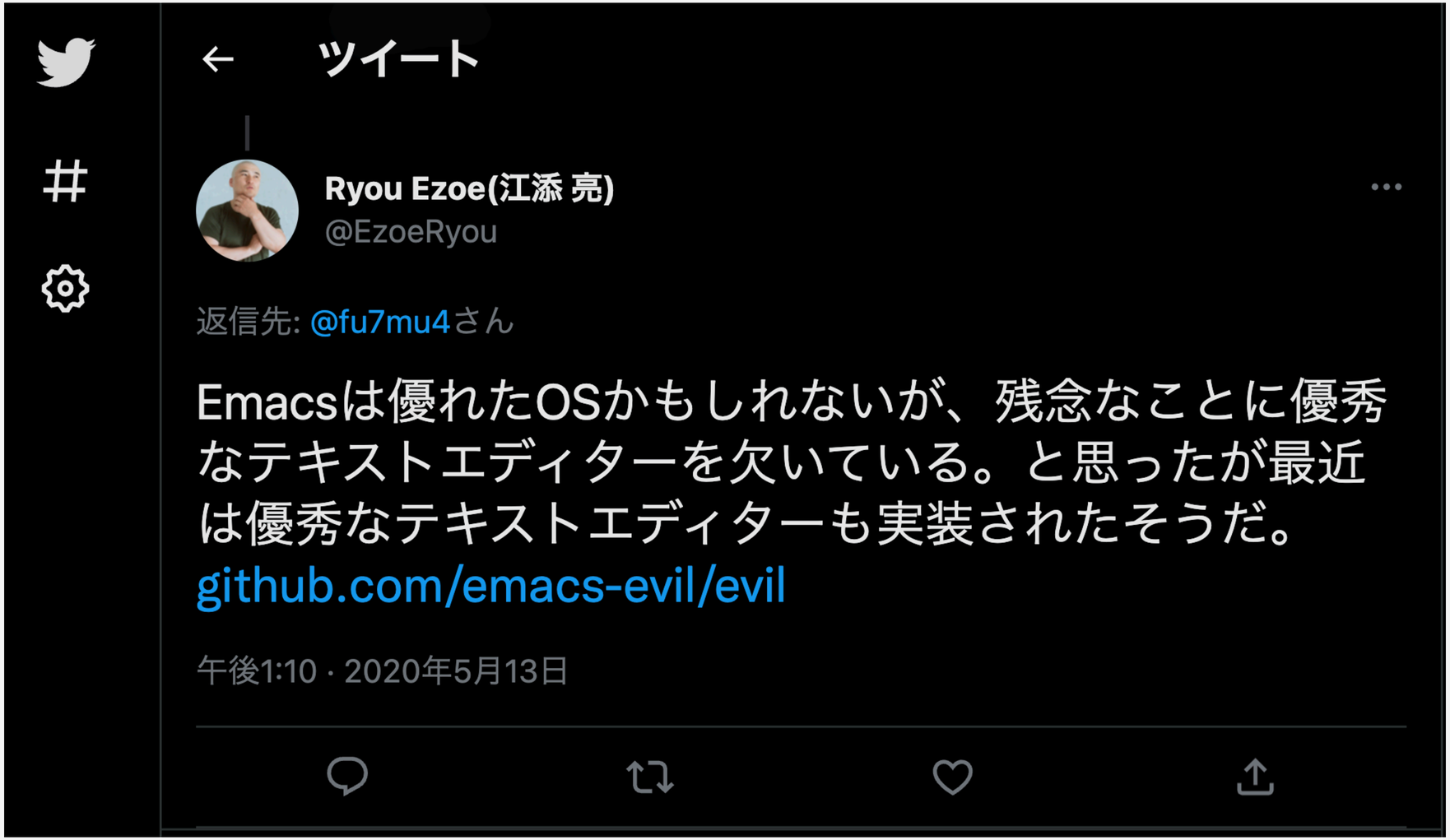
Emacs front line

pixiv Inc.
USAMI Kenta

pixiv

2023-02-01
TechFeed Experts Night #12

Vimはいいぞ！


ツイート
Ryou Ezoe(江添 亮)
@EzoeRyou
返信先: @fu7mu4さん
Emacsは優れたOSかもしれないが、残念なことに優秀なテキストエディターを欠いている。と思ったが最近は優秀なテキストエディターも実装されたそうだ。
github.com/emacs-evil/evil
午後1:10 · 2020年5月13日

pixiv

Emacsの本質とは何か

A modern Lisp machine

ウィキペディア
フリー百科事典

Fafnir127

LISPマシン

15の言語版

ページ　ノート　　閲覧　ソースを編集　履歴表示　☆　その他

出典: フリー百科事典『ウィキペディア（Wikipedia）』

LISPマシンは、LISPを主要なプログラミング言語として効率的に実行することを目的として設計された汎用のコンピュータである。ある意味では、最初の商用シングルユーザーワークステーションと言うこともできる。それほど数量的に大成功を収めたとはいえないが（1988年までに約7000台が出荷された[1]）、その後よく使われることになる様々な技術を商用化する先駆けとなった。例えば、効率的ガベージコレクション、レーザープリンター、ウィンドウシステム、コンピュータマウス、高解像度ビットマップグラフィックス、CHAOSNet（英語版）などのネットワーキングにおける技術革新などである。1980年代にシンボリックス（3600、3640、XL1200、MacIvoryなど）、LMI（Lisp Machines Incorporated、LMI Lambda）、テキサス・インスツルメンツ（Explorer、MicroExplorer）、ゼロックス（InterLisp-D搭載ワークステーション）といった企業がLISPマシンを製造販売した。オペレーティングシステムは Lisp Machine Lisp（英語版） や Interlisp（ゼロックスの場合）で書かれ、後に一部は Common Lisp で書かれた。

歴史 [ソースを編集]

背景 [ソースを編集]

そういう機械が
40年以上前にあった

図1 AI ワークステーション ELIS

現在，ELIS は NTT，沖電気工業などが出資して作った NTT の子会社 NTT インテリジェントテクノロジ (NTT-IT) から約1000万円で販売されている． (この価格は Lisp マシンとしてはかなり安いと思うが，EWS として考えると高い． 今後こういうマシンの真の競争相手は既存の Lisp マシンではなく UNIX ベースの EWS になるはずだから， マルチユーザで AI 向きという付加価値を加味してももう少し安いことが望まれる． もっとも，私は NTT-IT の経営に口をはさむ立場ではないので， これはあくまでも私の個人的願望だ．) というわけで， 商品としての TAO/ELIS と我々 (主に TAO 周辺のソフトウェア担当者) の研究材料としての TAO/ELIS の切り分けが一時ちょっと微妙だったが， 現在いろいろな関係が整理され，我々はまた研究材料としての TAO/ELIS に没頭できる状態になっている． 今後の我々の研究の成果の中にもし「使える」ものがあれば， それを世に出すことのできるパイプが布設されたというわけである．

NTT ソフトウェア研究所の私の同僚たちは， これから TAO/ELIS の上に知能処理のための統合的プログラミング環境 NUE (New Unified Environment) を作り上げる研究を進めようとしている． NUE のキャッチフレーズは，「知能処理ソフトウェアの研究開発の場において研究者や開発技術者の創造性を最大限に発揮させる高性能・高機能のプログラミング環境」となかなか勇ましくて格好いいのだが， いまところ具体的なイメージは定まっていない． NUE (鵺) という名前からしてこれは当然かもしれない． ある人が NUE は NUE's Undefined Environment (cf. GNU's Not Unix) のことではないかと鋭く指摘してくれたが正鵠を突いている．

「Multi-Paradigm Language TAO」より引用
https://www.nue.org/nue/tao/bitao/s001.html

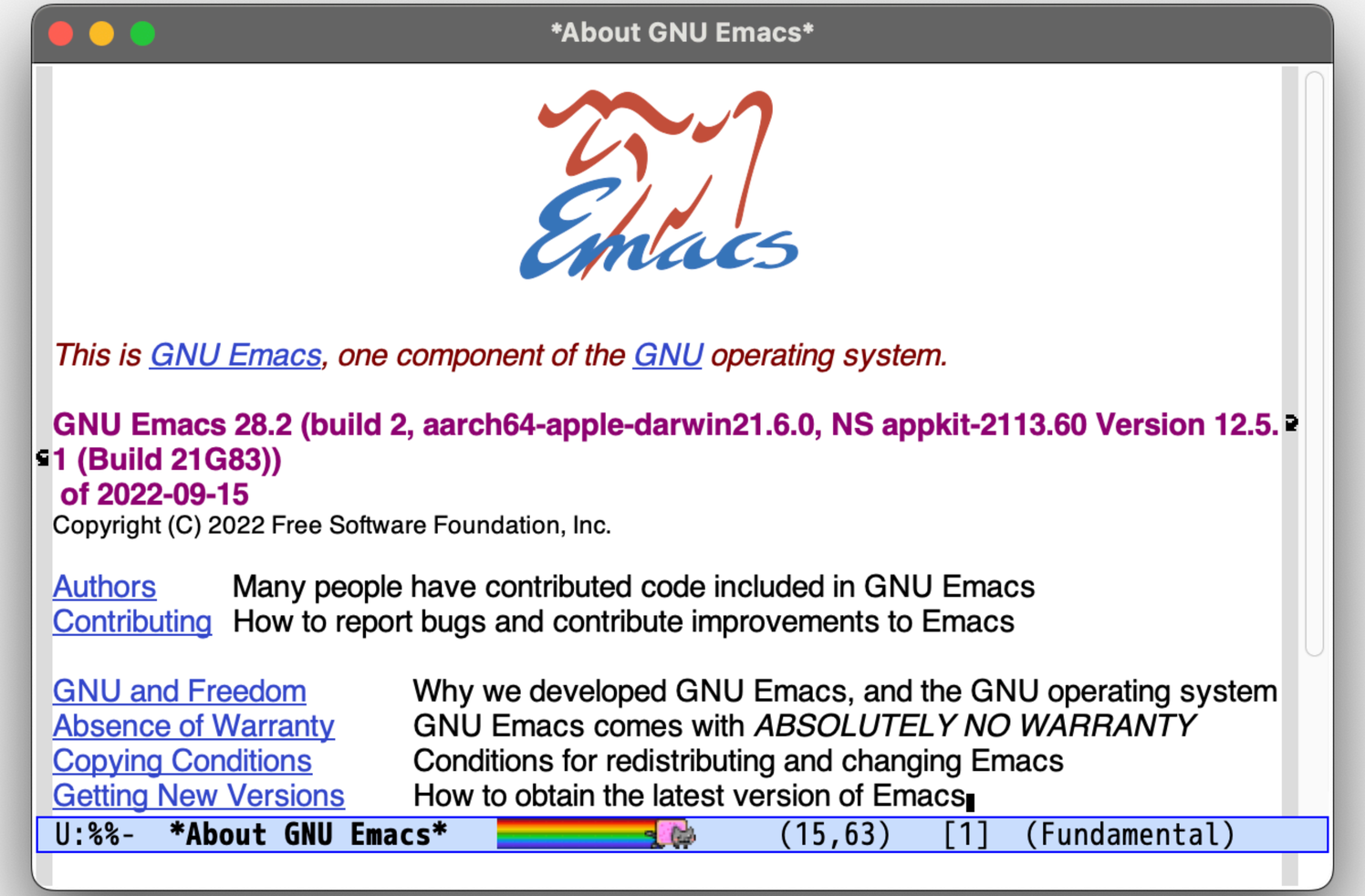
この写真は国産LISPマシンのELISなので本家Emacsの開発経緯とは直接関係ないのだが内蔵されたZENというEmacs風独自エディタの日本語入力メソッドKanzenが後のSKKに繋がりIME史に影響を及ぼしている

みなさんご存じ(ない)通り LISPマシンのような 高級言語専用マシンは 廃れている

だが、Lispは
死滅していなかった！

pixiv

世紀末救世主伝説Emacs



pixiv

スクリプトで拡張できる
テキストエディタの元祖

元祖Emacsは TECOというエディタの 拡張マクロ集 (EditorMACroS)

この人が作りました

ウィキペディア
フリー百科事典

Wikipedia内を検索 検索 Fafnir127

リチャード・ストールマン

106の言語版

ページ ノート 閲覧 ソースを編集 履歴表示 ツール

出典: フリー百科事典『ウィキペディア（Wikipedia）』

「**ストールマン**」はこの項目へ転送されています。この人物に因み命名された小惑星については「ストールマン (小惑星)」をご覧ください。

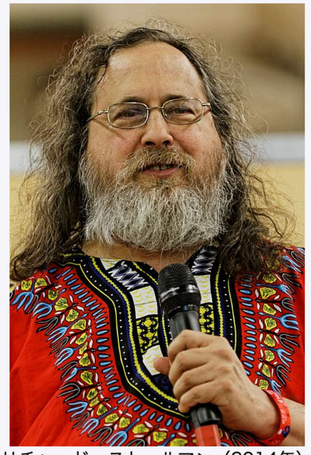
この項目「**リチャード・ストールマン**」は**加筆依頼**に出されており、内容をより充実させるために次の点に関する加筆が求められています。
加筆の要点 - 2019年に発生したFSF代表理事辞任に至った一連の騒動およびその際の発言について
（貼付後は**Wikipedia:加筆依頼のページに依頼内容を記述してください。**記述が無いとタグは除去されます）
（2023年9月）

リチャード・マシュー・ストールマン（**Richard Matthew Stallman**、1953年3月16日 - ）は、アメリカ合衆国のプログラマー、フリーソフトウェア活動家。コピーレフトの強力な推進者として知られ、現在にいたるまでフリーソフトウェア運動において中心的な役割を果たしている。また、プログラマーとしても著名な存在であり、開発者としてその名を連ねるソフトウェアにはEmacsやGCCなどがある。なお、名前の頭文字を取って **RMS** と表記されることもある。

年表 [ソースを編集]

- 1953年 - ダニエル・ストールマンとアリス・リップマンの子としてニューヨークに生まれる[1]。
- 1971年 - ハーバード大学に入学。Math55で好成績を残し[2]、MIT AI研のプログラマとなりハッカーコミュニティに加わる。
- 1974年 - 物理学の学位を取得し[3]、最優等の成績で大学を卒業。これに続いてMITの大学院に入学するが、MIT AI研でプログラマとしての活動を続けるうちに、物理学の研究をやめ、博士号をとることを放棄する。
- 1983年 - GNUプロジェクトを創始。
- 1984年 - GNUプロジェクトに専念するためMITを退職。
- 1985年 - GNU宣言の発表。

リチャード・ストールマン



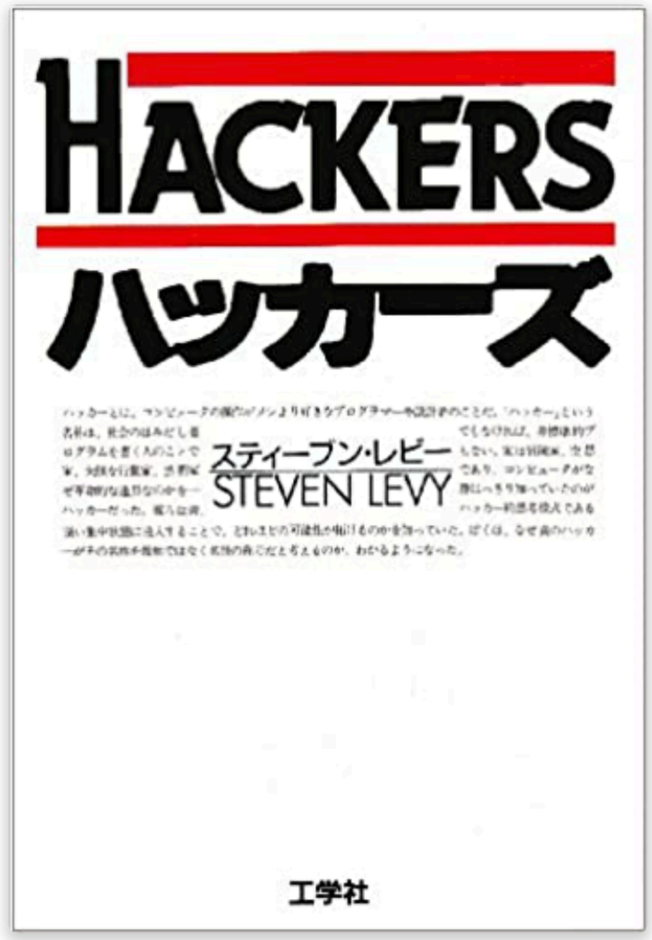
リチャード・ストールマン（2014年）

生誕 1953年3月16日（71歳）
アメリカ合衆国 ニューヨーク

国籍 アメリカ合衆国

pixiv

なんだかんだありまして



ハッカーズ 単行本 – 1987/3/1

スティーブン・レビー (著), 松田 信子 (著), 古橋 芳恵 (著)

★★★★☆ 22個の評価

すべての形式と版を表示

単行本
￥2,750
獲得ポイント: 83pt ✓prime

￥1,253 より 14 中古品
￥2,750 より 13 新品

出版社	発売日
工学社	1987/3/1

リチャードストールマン(RMS)とかいう若者がマサチューセッツ工科大学で自由を求めて戦ったりAI研究所でLispを書いてたりしてたらラボ内がLISPマシンを作ってる企業の派閥で分裂ちゃったのでどうしようもなくなってAIラボを去ってGNUっていう自由なOSを作ろうぜ！という物語があったりするのは後の話

自由なOSを作りたいよ

まずコンパイラと
エディタを作ったよ

そうしてGCCと
Emacsができましたとさ

めでたしめでたし

振り返りここまで

pixiv

改めて

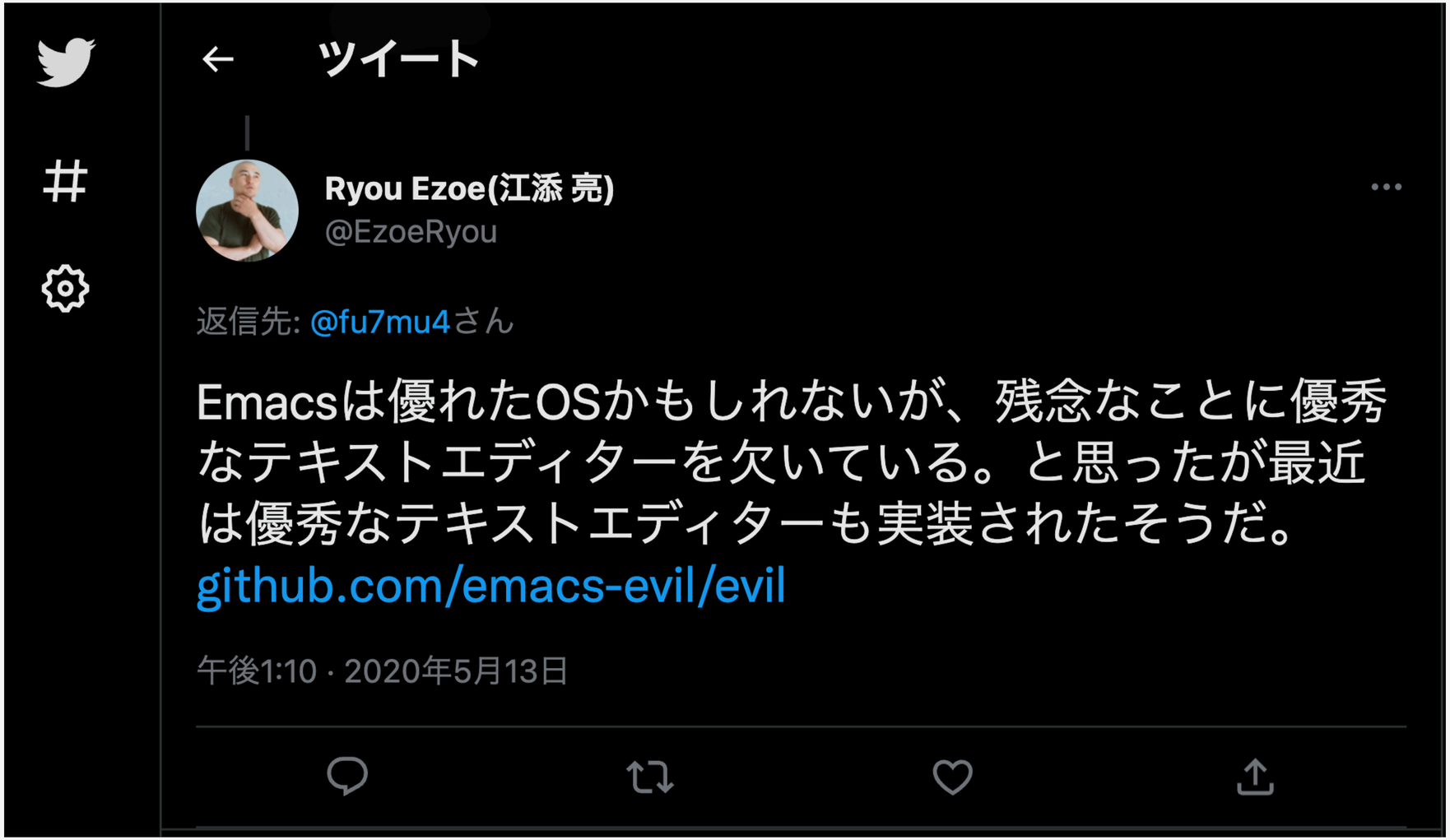
Emacsで何ができるか
ユーザーと非ユーザーで
大きな認識の開きがある

(ある種のジョークとして)

しばしば

EmacsはエディタではないEmacsはエディタではない

と言及される


ツイート
Ryou Ezoe(江添 亮)
@EzoeRyou
返信先: @fu7mu4さん
Emacsは優れたOSかもしれないが、残念なことに優秀なテキストエディターを欠いている。と思ったが最近は優秀なテキストエディターも実装されたそうだ。
github.com/emacs-evil/evil
午後1:10 · 2020年5月13日

pixiv

CI passing | melpa 20230116.1002 | melpa stable 1.14.2 | NonGNU ELPA evil 1.15.0 | docs failing | License GPL v3

Evil is an **e**xtensible **vi** **l**ayer for Emacs. It emulates the main features of Vim, and provides facilities for writing custom extensions. Also see our page on EmacsWiki.

Installation

See the official documentation for installation instructions. We recommend using *package.el*.

As a quickstart, you can add the following code to your Emacs init file.

```
;; Set up package.el to work with MELPA
(require 'package)
(add-to-list 'package-archives
             '("melpa" . "https://melpa.org/packages/"))
(package-initialize)
(package-refresh-contents)
```

Emacsはなんでもできる

メーラーでもあり
RSSリーダーでもあり
EPUBやPDFも読める

EmacsはOS
Emacsは環境

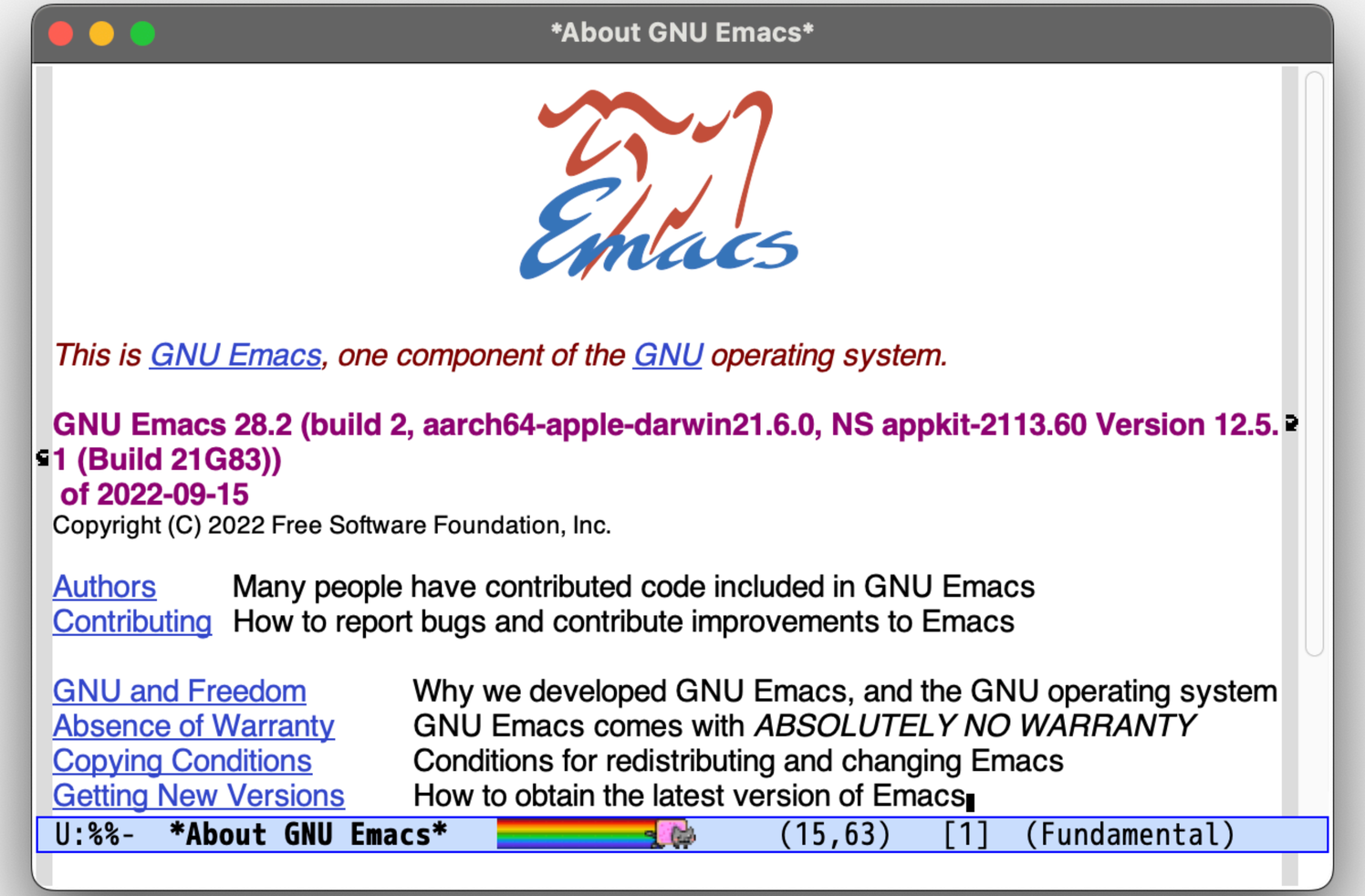
ジョークではなく

Emacsの大きな誤解

「Emacsってターミナルのエディタでしょ？」

EmacsはGUIでも動く

起動時から目立つEmacsロゴ



pixiv

フォントを扱える

README.md

```
# Hakone PSR-15 middleware dispatcher

**Hakone** is a lightweight [PSR-15] middleware dispatcher implementation. It is inspired by [Relay].

## Concept

The PSR-15 is designed to be general purpose middleware. All middleware can intervene in both requests and responses.

Hakone is a "queue-based request handler" described in [PSR-15 Meta Document]. The downside of queue-based is that the more
middleware you have, the more stack traces will pile up.

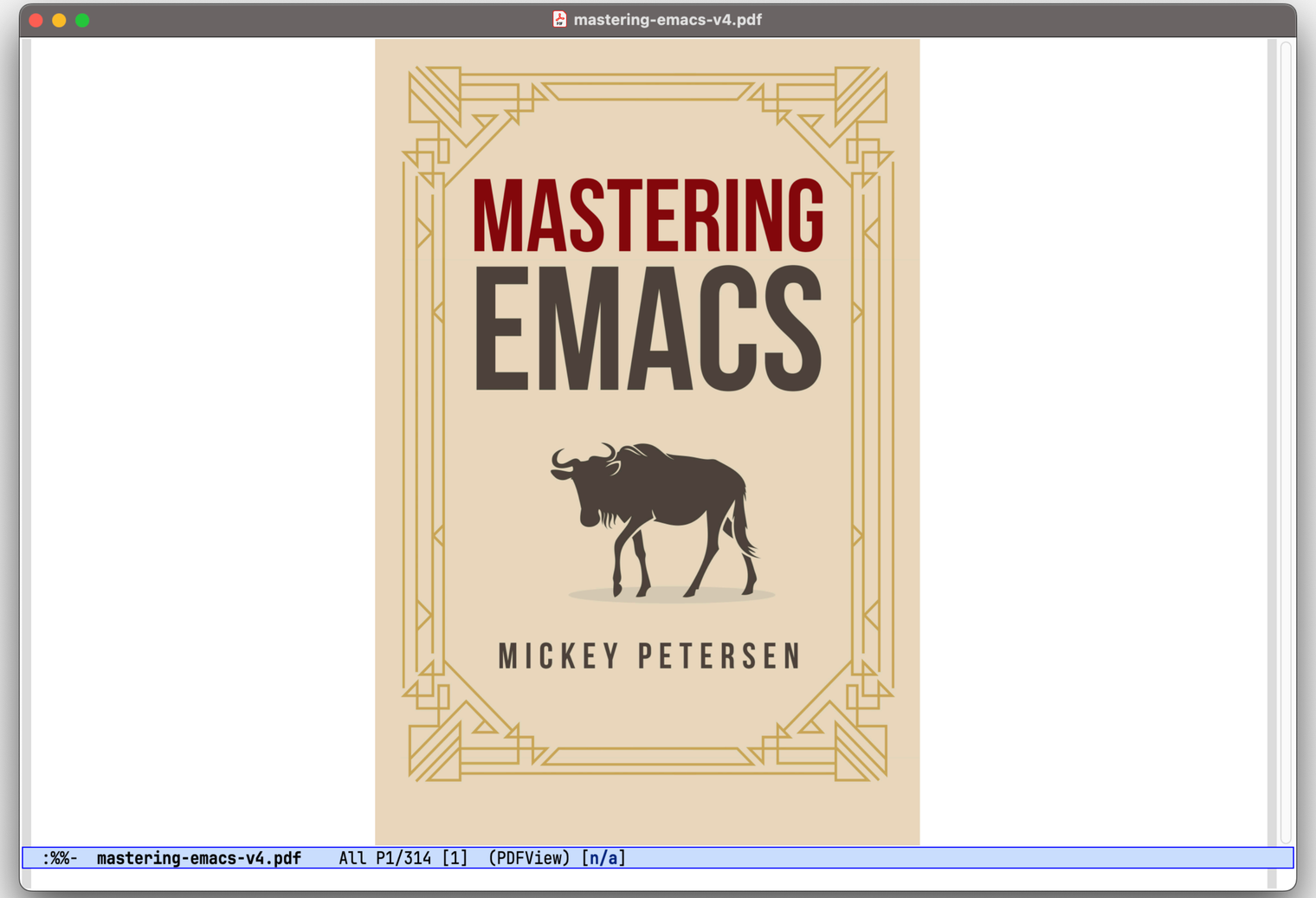
We've written a number of small-duty middlewares, but for many use cases they only do work on either the request or the
response. We decided to classify the types of middleware into three types: **"request interceptor"**, **"general-purpose
middleware"** and **"response decorator"**.

## How to use

```php
// Hakone\relay() returns ResponseHandler
$dispatcher = Hakone\relay([
    'interceptors' => [
        // ...
    ],
    'middlewares' => [
        // ...
    ],
    'handler' => $handler,
    'decorators' => [
        // ...
    ],
]);
U:---  README.md                (5,10)    [1] Git:master  (GFM Wrap) [Concept]
```

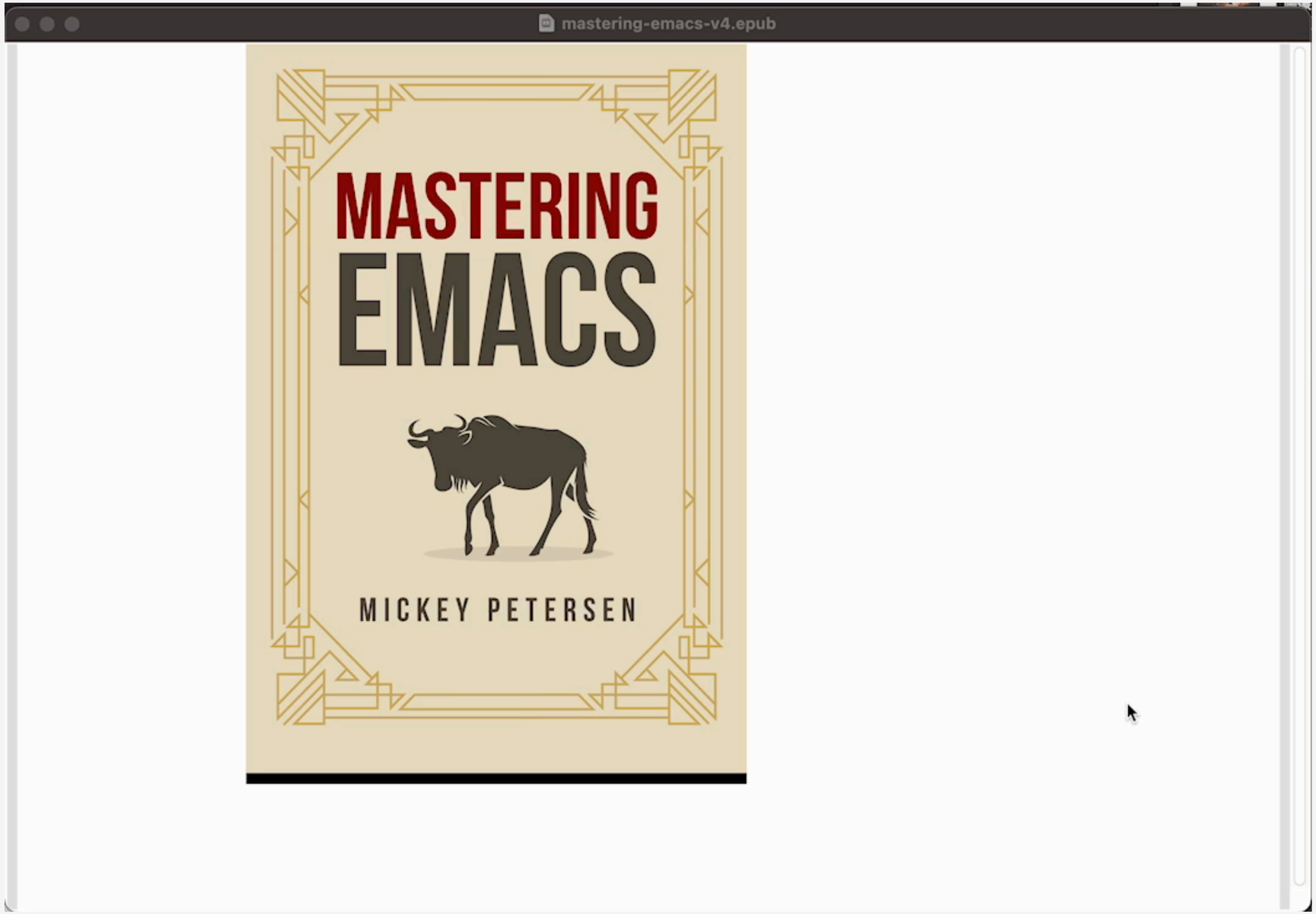

pixiv
```

PDFが開ける



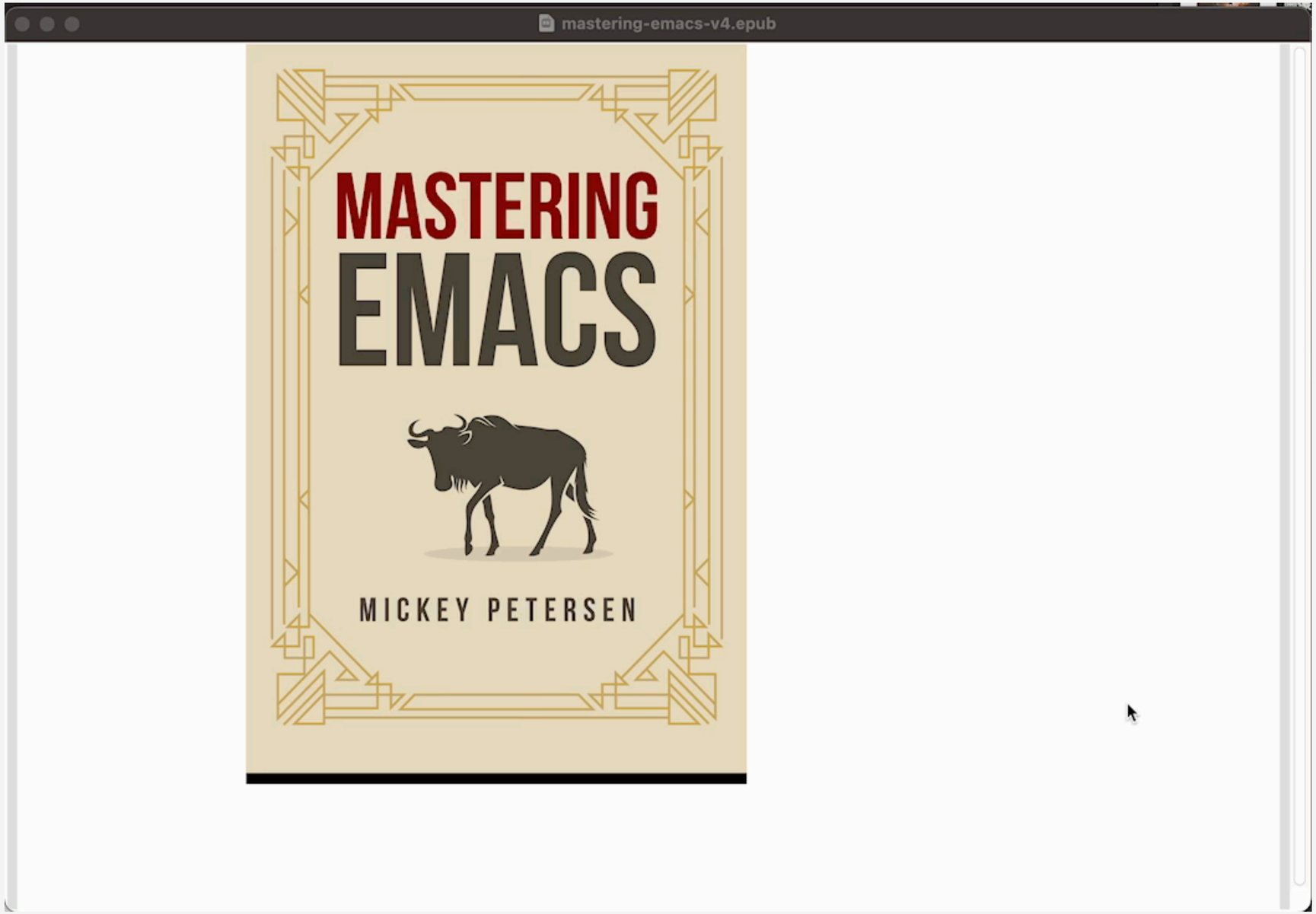
pixiv

EPUBも開ける



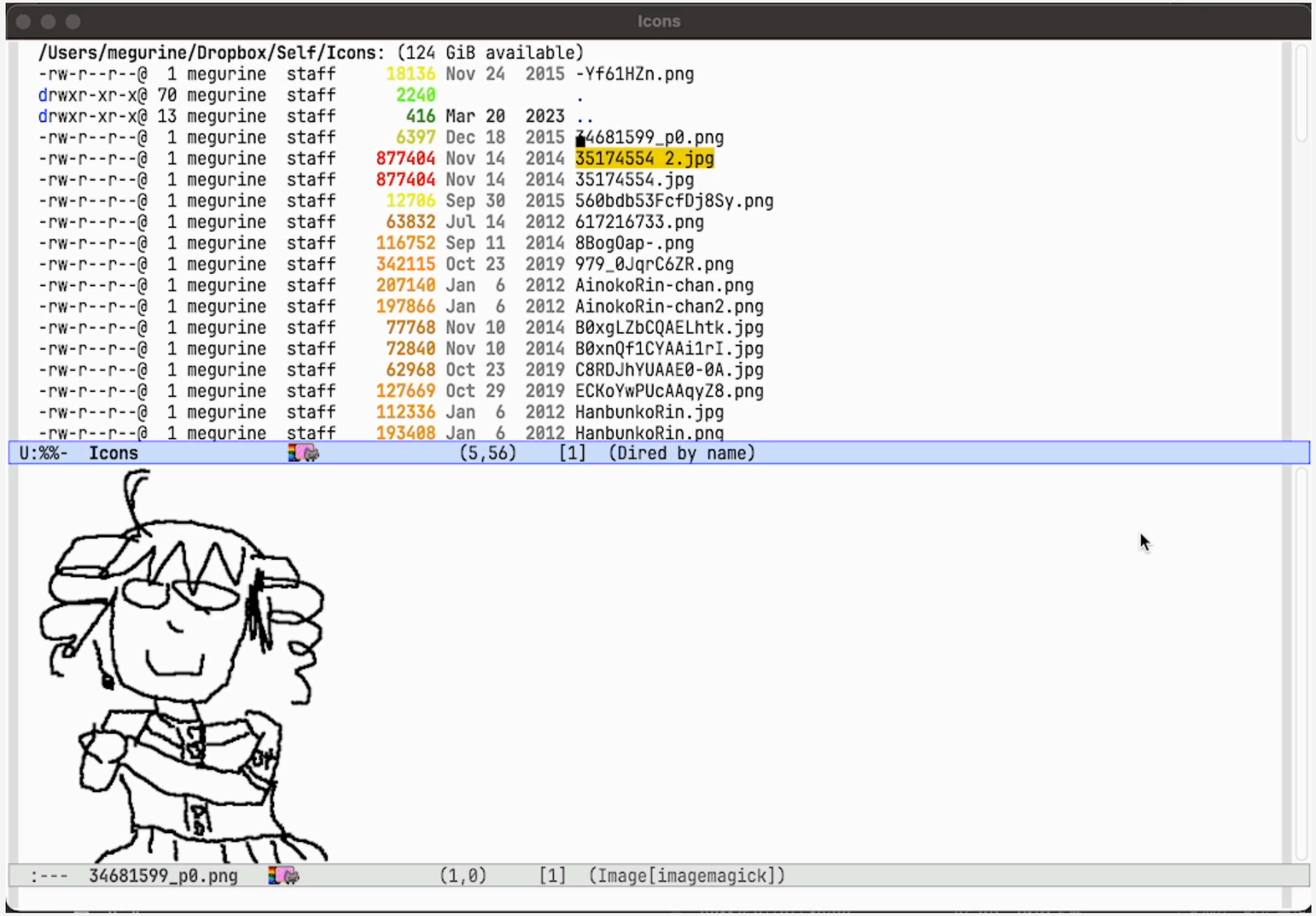
pixiv

EPUBも開ける



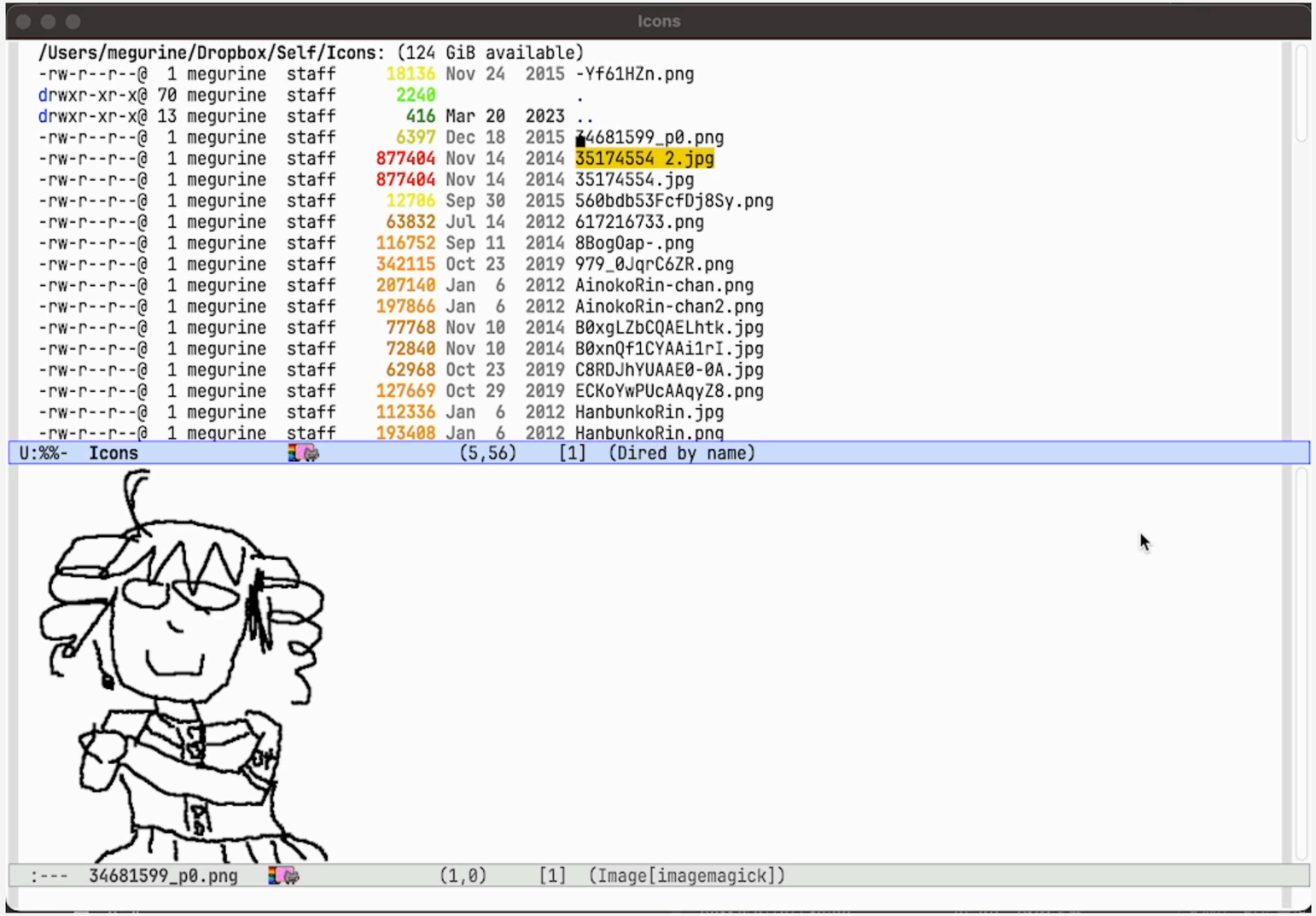
pixiv

ディレクトリを開いて画像見れる

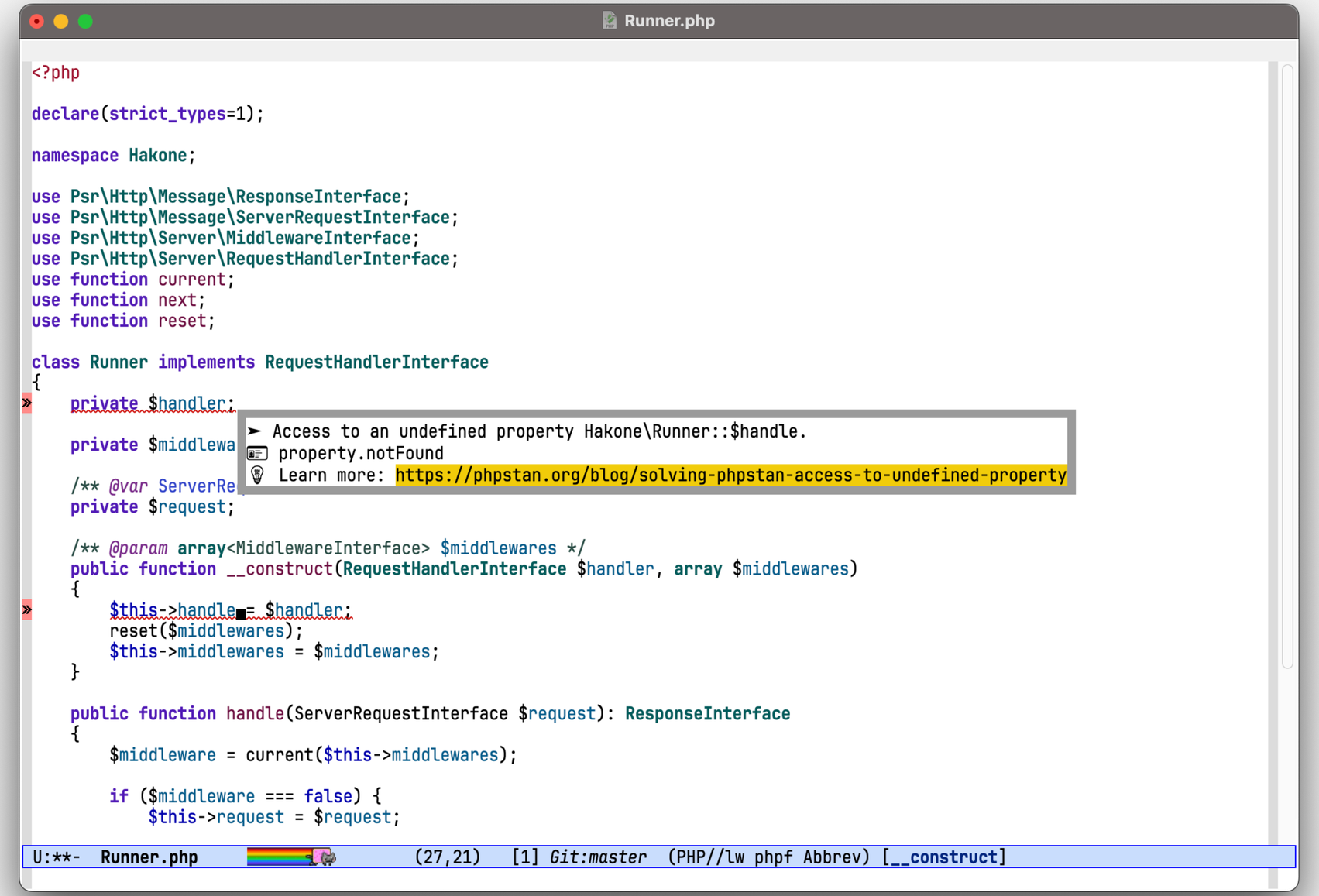


pixiv

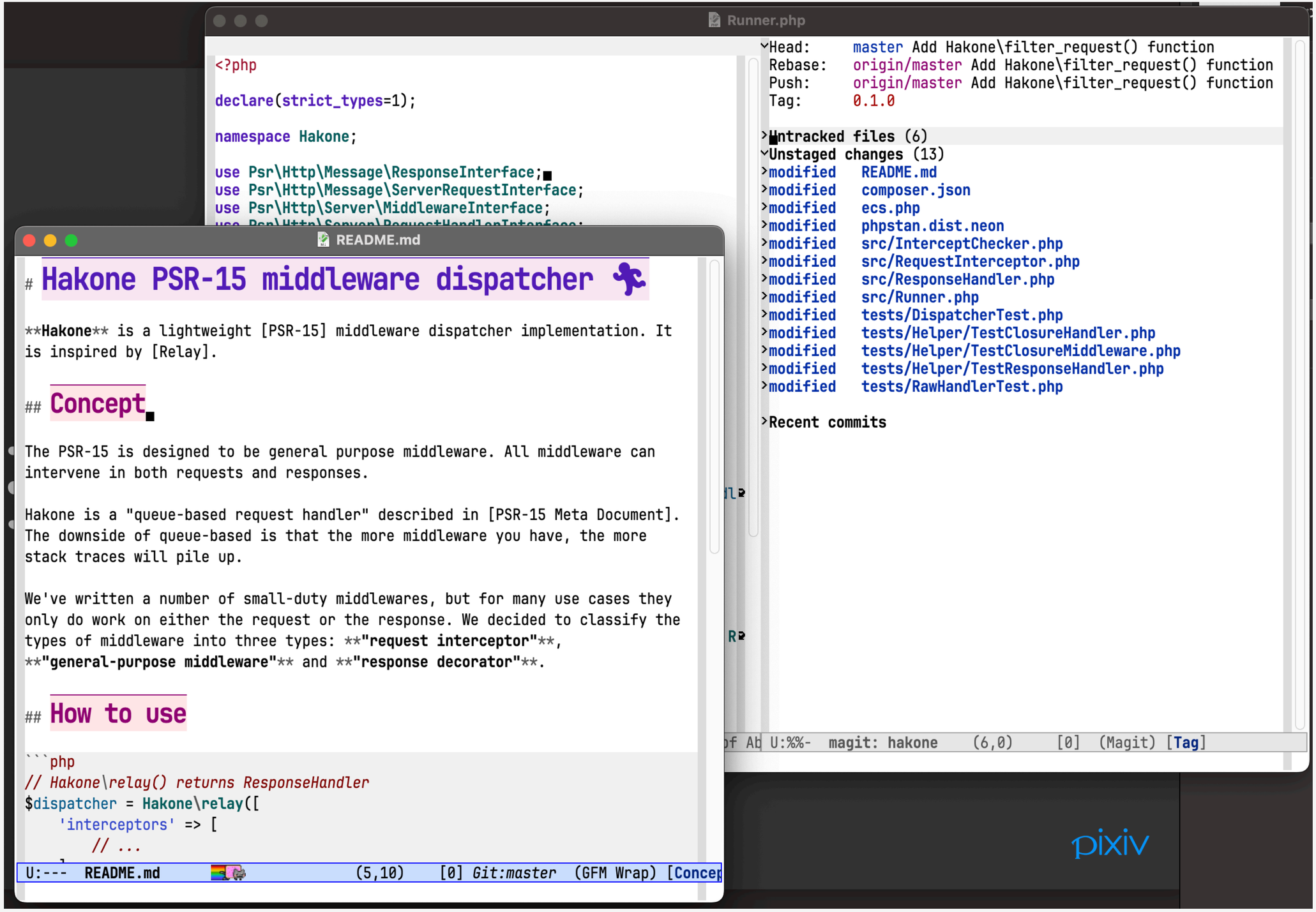
ディレクトリを開いて画像見れる



pixiv


Runner.php
<?php
declare(strict_types=1);
namespace Hakone;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\MiddlewareInterface;
use Psr\Http\Server\RequestHandlerInterface;
use function current;
use function next;
use function reset;
class Runner implements RequestHandlerInterface
{
private $handler;
➤ Access to an undefined property Hakone\Runner::$handle.
property.notFound
Learn more: https://phpstan.org/blog/solving-phpstan-access-to-undefined-property
private $middlewa
/** @var ServerRe
private $request;
/** @param array<MiddlewareInterface> $middlewares */
public function __construct(RequestHandlerInterface $handler, array $middlewares)
{
$this->handle= $handler;
reset($middlewares);
$this->middlewares = $middlewares;
}
public function handle(ServerRequestInterface $request): ResponseInterface
{
$middleware = current($this->middlewares);
if ($middleware === false) {
$this->request = $request;
U:**- Runner.php (27,21) [1] Git:master (PHP//lw phpf Abbrev) [__construct]

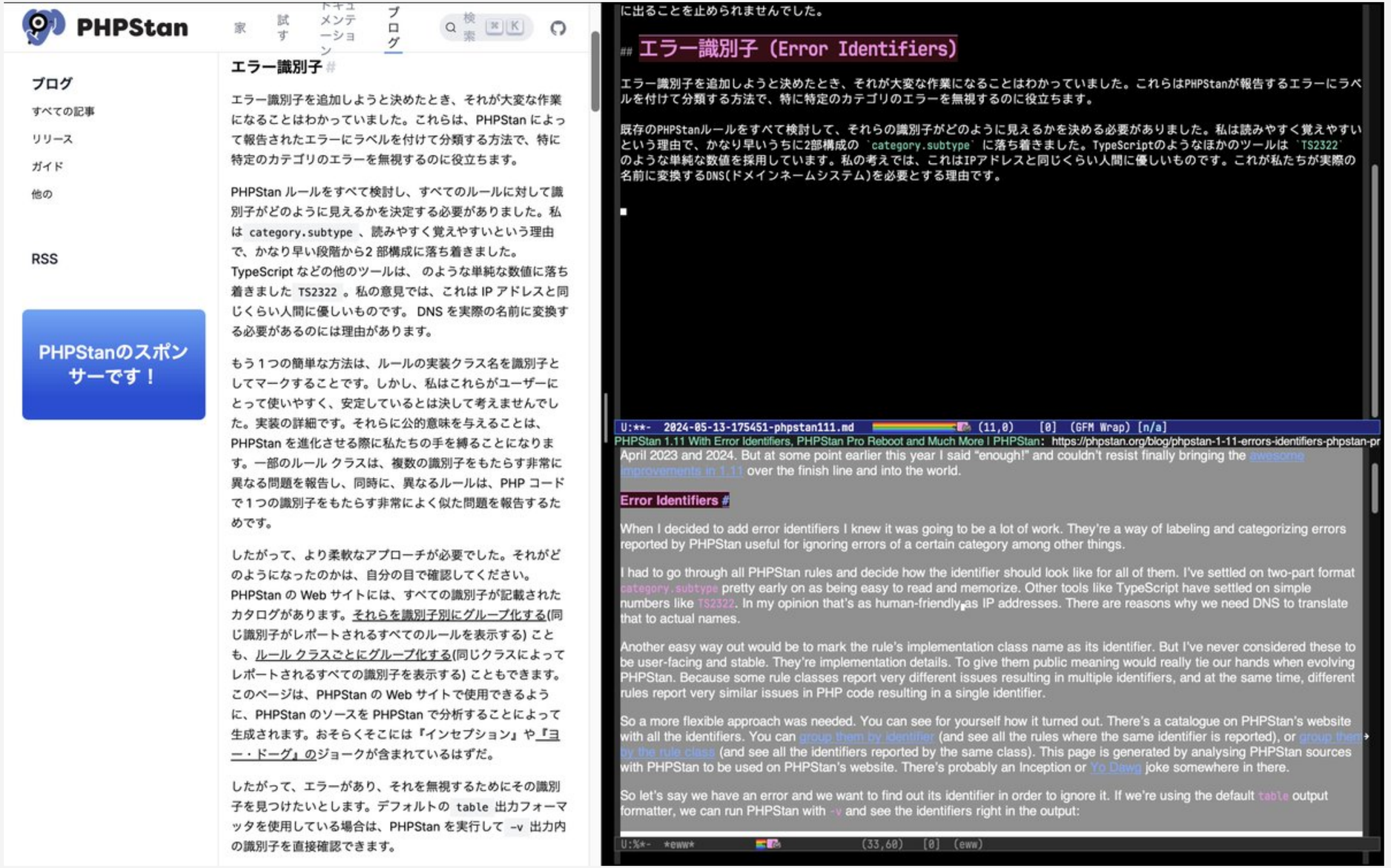
pixiv


Runner.php
<?php
declare(strict_types=1);
namespace Hakone;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\MiddlewareInterface;
Head: master Add Hakone\filter_request() function
Rebase: origin/master Add Hakone\filter_request() function
Push: origin/master Add Hakone\filter_request() function
Tag: 0.1.0
Untracked files (6)
Unstaged changes (13)
modified README.md
modified composer.json
modified ecs.php
modified phpstan.dist.neon
modified src/InterceptChecker.php
modified src/RequestInterceptor.php
modified src/ResponseHandler.php
modified src/Runner.php
modified tests/DispatcherTest.php
modified tests/Helper/TestClosureHandler.php
modified tests/Helper/TestClosureMiddleware.php
modified tests/Helper/TestResponseHandler.php
modified tests/RawHandlerTest.php
Recent commits
U:%%- magit: hakone (6,0) [0] (Magit) [Tag]
README.md
Hakone PSR-15 middleware dispatcher
Hakone is a lightweight [PSR-15] middleware dispatcher implementation. It is inspired by [Relay].
Concept
The PSR-15 is designed to be general purpose middleware. All middleware can intervene in both requests and responses.
Hakone is a "queue-based request handler" described in [PSR-15 Meta Document]. The downside of queue-based is that the more middleware you have, the more stack traces will pile up.
We've written a number of small-duty middlewares, but for many use cases they only do work on either the request or the response. We decided to classify the types of middleware into three types: **"request interceptor"**, **"general-purpose middleware"** and **"response decorator"**.
How to use
```php
// Hakone\relay() returns ResponseHandler
$dispatcher = Hakone\relay([
'interceptors' => [
// ...
U:--- README.md (5,10) [0] Git:master (GFM Wrap) [Concep
pixiv



pixiv
```

ウィンドウ内で
ブラウザも開ける！
(が、macOSで動作しなかった…)


PHPStan
家
試す
ドキュメンテーション
ブログ
検索
ブログ
すべての記事
リリース
ガイド
他の
RSS
PHPStanのスポンサーです！
エラー識別子 #
エラー識別子を追加しようと決めたとき、それが大変な作業になることはわかっていました。これらは、PHPStan によって報告されたエラーにラベルを付けて分類する方法で、特に特定のカテゴリのエラーを無視するのに役立ちます。
PHPStan ルールをすべて検討し、すべてのルールに対して識別子がどのように見えるかを決定する必要がありました。私は category.subtype 、読みやすく覚えやすいという理由で、かなり早い段階から2 部構成に落ち着きました。TypeScript などの他のツールは、 のような単純な数値に落ち着きました TS2322 。私の意見では、これは IP アドレスと同じくらい人間に優しいものです。 DNS を実際の名前に変換する必要があるのには理由があります。
もう 1 つの簡単な方法は、ルールの実装クラス名を識別子としてマークすることです。しかし、私はこれらがユーザーにとって使いやすく、安定しているとは決して考えませんでした。実装の詳細です。それらに公的意味を与えることは、PHPStan を進化させる際に私たちの手を縛ることになります。一部のルール クラスは、複数の識別子をもたらす非常に異なる問題を報告し、同時に、異なるルールは、PHP コードで 1 つの識別子をもたらす非常によく似た問題を報告するためです。
したがって、より柔軟なアプローチが必要でした。それがどのようになったのかは、自分の目で確認してください。PHPStan の Web サイトには、すべての識別子が記載されたカタログがあります。それらを識別子別にグループ化する(同じ識別子がレポートされるすべてのルールを表示する) ことも、ルール クラスごとにグループ化する(同じクラスによってレポートされるすべての識別子を表示する) こともできます。このページは、PHPStan の Web サイトで使用できるように、PHPStan のソースを PHPStan で分析することによって生成されます。おそらくそこには『インセプション』や『ヨー・ドーグ』のジョークが含まれているはずだ。
したがって、エラーがあり、それを無視するためにその識別子を見つけたいとします。デフォルトの table 出力フォーマッタを使用している場合は、PHPStan を実行して -v 出力内の識別子を直接確認できます。
に出ることを止められませんでした。
エラー識別子（Error Identifiers）
エラー識別子を追加しようと決めたとき、それが大変な作業になることはわかっていました。これらはPHPStanが報告するエラーにラベルを付けて分類する方法で、特に特定のカテゴリのエラーを無視するのに役立ちます。
既存のPHPStanルールをすべて検討して、それらの識別子がどのように見えるかを決める必要がありました。私は読みやすく覚えやすいという理由で、かなり早いうちに2部構成の `category.subtype` に落ち着きました。TypeScriptのようなほかのツールは `TS2322` のような単純な数値を採用しています。私の考えでは、これはIPアドレスと同じくらい人間に優しいものです。これが私たちが実際の名前に変換するDNS(ドメインネームシステム)を必要とする理由です。
U:**- 2024-05-13-175451-phpstan111.md (11,0) [0] (GFM Wrap) [n/a]
PHPStan 1.11 With Error Identifiers, PHPStan Pro Reboot and Much More I PHPStan: https://phpstan.org/blog/phpstan-1-11-errors-identifiers-phpstan-pr
April 2023 and 2024. But at some point earlier this year I said "enough!" and couldn't resist finally bringing the awesome improvements in 1.11 over the finish line and into the world.
Error Identifiers #
When I decided to add error identifiers I knew it was going to be a lot of work. They're a way of labeling and categorizing errors reported by PHPStan useful for ignoring errors of a certain category among other things.
I had to go through all PHPStan rules and decide how the identifier should look like for all of them. I've settled on two-part format category.subtype pretty early on as being easy to read and memorize. Other tools like TypeScript have settled on simple numbers like TS2322. In my opinion that's as human-friendly as IP addresses. There are reasons why we need DNS to translate that to actual names.
Another easy way out would be to mark the rule's implementation class name as its identifier. But I've never considered these to be user-facing and stable. They're implementation details. To give them public meaning would really tie our hands when evolving PHPStan. Because some rule classes report very different issues resulting in multiple identifiers, and at the same time, different rules report very similar issues in PHP code resulting in a single identifier.
So a more flexible approach was needed. You can see for yourself how it turned out. There's a catalogue on PHPStan's website with all the identifiers. You can group them by identifier (and see all the rules where the same identifier is reported), or group them by the rule class (and see all the identifiers reported by the same class). This page is generated by analysing PHPStan sources with PHPStan to be used on PHPStan's website. There's probably an Inception or Yo Dawg joke somewhere in there.
So let's say we have an error and we want to find out its identifier in order to ignore it. If we're using the default table output formatter, we can run PHPStan with -v and see the identifiers right in the output:
U:%*- *eww* (33,60) [0] (eww)

pixiv

Emacsは単なる
エディタではなく
アプリケーション実行環境

OSを問わず実行できる
(Windows, GNU/Linux, *BSD, macOS, Haiku, ...)

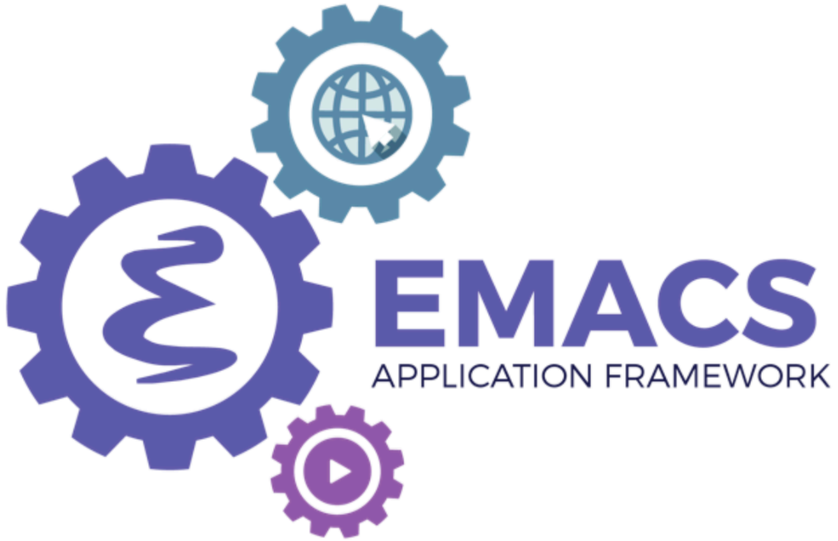
…といいたいが、
外部プロセスやファイル
システムに依存すると
意図通り動かない

WindowsではWSLで動かすこともできる
(WSLgでGUI起動もできる)

問題意識を持って
高速な非同期UIを
実装している人も居る

README.md

English | 简体中文

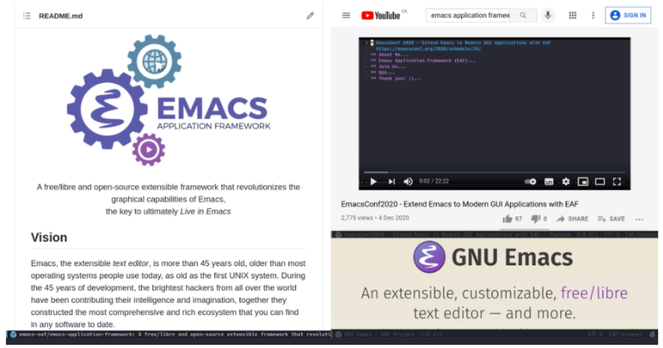


A free/libre and open-source extensible framework that revolutionizes the graphical capabilities of Emacs.
The key to ultimately *Live in Emacs*

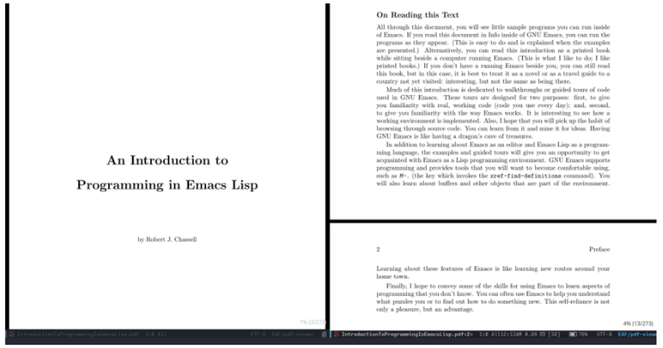
Features

EAF is very extensible. We ship a lot of applications, feel free to choose anything you find interesting to install:

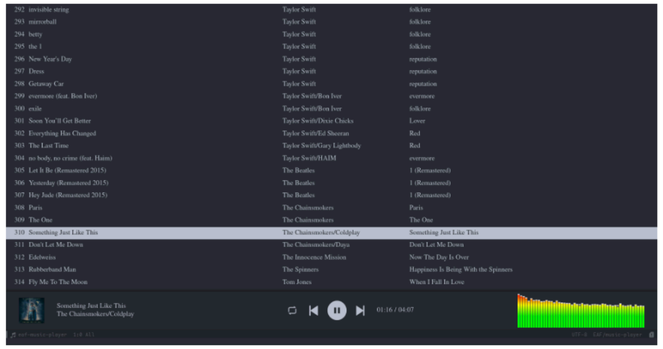
Browser



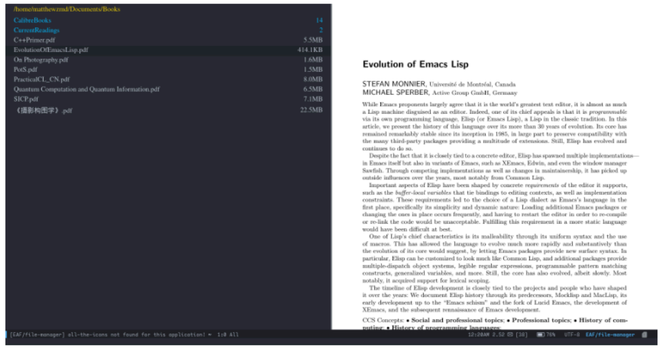
PDF Viewer



Music Player



File Manager



Browser: A modern, customizable and extensible browser in Emacs

pixiv

lsp-bridge

Lsp-bridge's goal is to become the fastest LSP client in Emacs.

Lsp-bridge uses python's threading technology to build caches that bridge Emacs and LSP server. Lsp-bridge will provide smooth completion experience without compromise to slow down emacs' performance.

```
class LspServer:
    def __init__(self, message_queue, project_path, server_info, server_name):
        self.message_queue = message_queue
        self.project_path = project_path
        self.server_info = server_info
        self.initialize_id = generate_request_id()
        self.server_name = server_name
        self.request_dict: Dict[int, Handler] = dict()
        self.root_path = self.project_path

        self.p
        # (x) p                                          Variable
        # parse_document_uri                             Method
        s (x) project_path                               Variable
        s (x) completion_resolve_provider                Variable
        s (x) completion_trigger_characters              Variable
          get_capabilities                               Method
        # get_server_workspace_change_co ...             Method
        s handle_workspace_configuration ...             Method
          lsp_message_dispatcher                         Method
        # (x) rename_prepare_provider                    Variable
        message_emacs("Start LSP server ({}) for {}...".format(self.server_info["name"], self.root_path))

        # Two separate thread (read/write) to communicate with LSP server.
        self.receiver = LspServerReceiver(self.p)
        self.receiver.start()

        self.sender = LspServerSender(self.p)
        self.sender.start()

        # All LSP server response running in ls_message_thread.
        self.ls_message_thread = threading.Thread(target=self.lsp_message_dispatcher)
        self.ls_message_thread.start()

        self.files: Dict[str, "FileAction"] = dict()
```

```
parse_document_uri: (filepath: Unknown, external_file_link: Unknown) ->
(Unknown | str)

If FileAction include external_file_link return by LSP server, such as jdt.
We should use external_file_link, such as uri 'jdt://xxx', otherwise use
filepath as textDocument uri.
```

```
ufsize=DEFAULT_BUFFER_SIZE, stdin=PIPE, stdout=PIPE, stderr=stderr)
```

Emacsは環境！

仮想環境？

図1 AI ワークステーション ELIS

現在，ELIS は NTT，沖電気工業などが出資して作った NTT の子会社 NTT インテリジェントテクノロジ (NTT-IT) から約1000万円で販売されている．(この価格は Lisp マシンとしてはかなり安いと思うが，EWS として考えると高い．今後こういうマシンの真の競争相手は既存の Lisp マシンではなく UNIX ベースの EWS になるはずだから，マルチユーザで AI 向きという付加価値を加味してももう少し安いことが望まれる．もっとも，私は NTT-IT の経営に口をはさむ立場ではないので，これはあくまでも私の個人的願望だ．) というわけで，商品としての TAO/ELIS と我々 (主に TAO 周辺のソフトウェア担当者) の研究材料としての TAO/ELIS の切り分けが一時ちょっと微妙だったが，現在いろいろな関係が整理され，我々はまた研究材料としての TAO/ELIS に没頭できる状態になっている．今後の我々の研究の成果の中にもし「使える」ものがあれば，それを世に出すことのできるパイプが布設されたというわけである．

NTT ソフトウェア研究所の私の同僚たちは，これから TAO/ELIS の上に知能処理のための統合的プログラミング環境 NUE (New Unified Environment) を作り上げる研究を進めようとしている．NUE のキャッチフレーズは，「知能処理ソフトウェアの研究開発の場において研究者や開発技術者の創造性を最大限に発揮させる高性能・高機能のプログラミング環境」となかなか勇ましくて格好いいのだが，いまところ具体的なイメージは定まっていない．NUE (鵺) という名前からしてこれは当然かもしれない．ある人が NUE は NUE's Undefined Environment (cf. GNU's Not Unix) のことではないかと鋭く指摘してくれたが正鵠を突いている．

「Multi-Paradigm Language TAO」より引用
https://www.nue.org/nue/tao/bitao/s001.html

現代に残る最後の(仮想)Lispマシン

言語にはパワーがある

普通のやつらの上を行け ---Beating the Averages---

著者：Paul Graham
Copyright 2001 by Paul Graham

これは、Paul Graham: Beating the Averages を、原著者の許可を得て翻訳・公開するものです。

プロジェクト杉田玄白正式参加テキスト。

<版権表示>
本和訳テキストの複製、変更、再配布は、この版権表示を残す限り、自由に行って結構です。
(「この版権表示」には上の文も含まれます。すなわち、再配布を禁止してはいけません)。
Copyright 2001 by Paul Graham
原文: http://www.paulgraham.com/avg.html
日本語訳：Shiro Kawai (shiro @ acm.org)
<版権表示終り>

「普通のやつらの上を行け ---Beating the Averages---」より引用
http://practical-scheme.net/trans/beating-the-averages-j.html

(*この記事は2001/3/25にケンブリッジで開催されたFranz Developer Symposium で行った講演をベースにしている。2001/5/3版*)

1995年の夏、私は友人のロバート・モリスとともにViawebというベンチャー企業を 立ち上げた。エンドユーザーがオンラインストアを自分で作ることが出来るソフトウェアを書く、 というのが計画だった。当時、このソフトウェアが新しかったのは、 ソフトウェア自身を我々のサーバーで走らせて、 通常のWebページをインタフェースにするという点だった。

もちろんたくさんの人達がこのアイディアを思い付いていたのだろうが、 私の知る限り、Viawebは最初のWebベースアプリケーションだった。 そのアイディアがとても斬新なものに思えたので、私達は自分の会社を そのように名付けたくらいだ：私達のソフトウェアはユーザのデスクトップでは なく、webを通して (via web) 動く、というわけだ。

もうひとつ、私達のソフトウェアがユニークだったのは、それが主にLispと呼ばれる プログラミング言語で書かれていたからだ [注1]。 それまで主として大学や研究所で使われていたLispの、事実上初めての大規模なエンドユーザ アプリケーションだった。 そして、パワーにおいて劣る他の言語を使っている競争相手に対して、 Lispは強力なアドバンテージとなった。

「普通のやつらの上を行け ---Beating the Averages---」より引用
http://practical-scheme.net/trans/beating-the-averages-j.html

「ほげ言語」のパラドックス

Lispの何がそんなに素晴らしいんだ? それに、もしLispがそんなに良いのなら、 どうしてみんなそれを使わないんだ? これは修辞疑問みたいに聞こえるが、きちんとした答えのある質問だ。 Lispは、信者のみが見ることができる魔法の特性があるから素晴らしいんじゃない。 単に、今ある言語の中でもっともパワフルだからだ。 そして、みんながそれを使わない理由は、プログラミング言語とは単なる技術だけでなく、 心の習慣でもあり、それは最も変化の遅いものであるというものだ。 もちろん、この二つの答えはもっとよく説明されなければなるまい。

まず、思いっきり物議を醸しそうな発言から始めよう。 プログラミング言語は、その力において差がある。

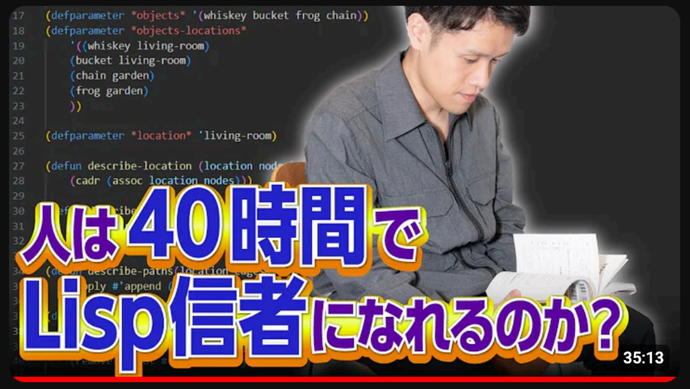
少なくとも、高級言語は機械語より力があるということに反対する人はほとんど居ないだろう。 こんにちではほとんどのプログラマーは、通常、機械語でプログラムを書きたいとは思わない ということに同意するだろう。かわりに高級言語で書いて、 コンパイラにそれを機械語に変換させるべきなのだ。 この考えは既にハードウェアの設計に採り入れられている。 1980年代以降、命令セットは人間のプログラマよりはコンパイラ向けに設計されるようになった。

あなたがプログラムの全てを機械語で書くと言い出したら、誰もがそれは間違いだと言うだろう。 しかし、同じ原理だが見過ごされがちな一般法則がある。もしあなたがいくつかの言語を 選択することができて、他の条件が同じ場合、最も力のある言語以外を選択することは間違いなのだ [注3]。

「普通のやつらの上を行け ---Beating the Averages---」より引用
http://practical-scheme.net/trans/beating-the-averages-j.html

ということを
Yコンビネータ創始者の
ポールグレアムが
言ってる

ゆるコンピュータ科学ラジオ Lisp



【検証】40時間Lispを勉強したら信者になれる？【Lisp1】#118

7.9万 回視聴・3 か月前

ゆるコンピュータ科学ラジオ

「Lisp」シリーズの第1回。「気難しそうで気難しくないLisper」「"3+5"より"+ 5 3"の方が良く見えてくる」「神が作った言語と ...

チャプター数: 8 40時間勉強したらLisp信者になる？ | 熱狂的な信者が多いLisp | 気難しそうで気難しくないLisper ...

Lispはなぜ狂信者を生むのか？ その答えは、奇妙な出自。【ポール・グレアム3】#112

5.5万 回視聴・4 か月前

ゆるコンピュータ科学ラジオ

「ポール・グレアム」の第3回です。「Lispはなぜ狂信者を生むのか」「文法よりも意味を重視した言語」「最後までチョケ ...

チャプター数: 7 自分のうんざりベンチマークとは？ | ポール・グレアムはスーツ嫌い | Lisperたちが狂信するLisp...

大人の勉強は「夢の中」が主戦場。寝ながらマクロを書くといい【Lisp雑談】#120

4.9万 回視聴・2 か月前

ゆるコンピュータ科学ラジオ

「Lisp」の撮り終わり雑談回。「ライバルに勝つための秘策"夢の中で勉強"」「Lispはレゴの基本パーツしか使っていない」「言語 ...

チャプター数: 7 夢のようなLisp勉強用サイト | 大人になって実感する演習問題 | ライバルに勝つための秘策「夢の...

ハッカーになりたければ Emacs

日々、とんは語る。

Posts

自分が関わったSI案件のコードメンテナンスとその価値

いい感じのRemixのディレクトリ構成

QNAP TBS-464でM.2 SSDのNASを作った

家庭の味を支える調味料

VimConf 2023にスポンサー参加しました

マイノリティ自慢

原器としてのHHKB

ブログ再開にあたってブログをメンテナンスした

2021-2022年のお仕事近況報告

インプレスの「いちばんやさしいWeb3の教本」回収判断について

書籍「いちばんやさしいWeb3の教本」は本当に酷い内容だし、Web3界隈の人は一致団結して間違いを指摘して、インプレスは回収して内容修正するべき

風になりたい奴だけがEmacsを使えばいい 2020

2020-09-26T14:58:12+0900
Emacs

先日、Emacsに一生入門できねえ2020という記事を目にした。

確かにEmacsは難しい。まったくもって増田の言う通りだ。うんうんと頷きながら、過去に自分が書いた「風になりたい奴だけが Emacs を使えばいい。」という記事が脳裏に浮かんだ。

10年間の出来事

僕が「風になりたい奴だけがEmacsを使えばいい」と言った記事は2010年9月4日に投稿されていて、あれから実に10年の月日が経過していた。とても懐しい。

振り返ればこの10年間でエディタの世界は大きく変わった。次世代エディタを銘打ったAtomが誕生し、エディタにおける表現の限界をぶち壊した。そして後続で登場したVSCodeが一気にシェアを奪い、一瞬でトップシェアの座に立ってしまった。予想しなかった未来があった。

一方、Emacsはどうなったかと言えば、メジャーバージョンが23から27になった。しかし、起動したてのEmacsはまるで時が止まっているかのように見た目に変化がなく、表面からのぞかせるEmacsの顔は驚くほど何も変化していなかった。

外見に変化はなくても
中身は洗練されている

コンピュータサイエンス
の問題に立ち向かって
最強の環境を作りたい人
はEmacsを

コミュニティは Emacs JPのSlackに (不定期にオンラインミートアップもやってるよ)

pixiv

Technology Roundtable on the future of Asahikawa
for Makers and ENgineers.

上川地方に技術の知見を惜しみなく注ぐ，技術好きのための円卓会議

TechRAMEN 2024 Conference

Date

2024/7/26(Fri) 16:00-20:00
2024/7/27(Sat) 10:00-18:00

Location

旭川市大雪クリスタルホール

pixiv

TechRAMEN 2024 Conference

採択 2024/07/26 17:45〜 チャーシュー - [セッション/ハンズオン] 1F 第一会議室 [一杯] トーク - 20 分 対象地域出身

なぜテキストエディタを極めるのか

うさみけんた tadsan

☆ 9

人類は10種類に分けられます。そう、テキストエディタにこだわるものと、こだわらないものです。

私は20年前に中空知の片田舎でパソコンをｶﾀｶﾀしていた頃からテキストエディタの世界に魅せられてきました。
弘法筆を選ばずという言葉もありますが、木こりは斧を研ぎ、職人は道具にこだわるものです。

このトークでは私がテキストエディタにのめりこんでいった体験から、エディタを極めることで見える景色についてお話しいたします。

- テキストエディタの種類について
- 拡張と外部プログラム連携について
- テキストエディタのプログラミング言語サポートについて
- エディタを使うということ

connpass produced by Be PROUD

イベント検索 ダッシュボード 新着イベント イベント管理 イベント作成

8月 4

東京Emacs勉強会 サマーフェスティバル2024

フォロー参加者

募集内容	**発表する** 無料	先着順 7/7人
	一般参加 無料	先着順 37/40人
出席登録	(イベント開始時間の2時間前から終了時間まで、参加者のみに公開されます)	

イベントの説明

EmacsJP サマーフェスティバル 2024

ポジションペーパー

東京Emacs勉強会では短い時間で最大限に交流するため、参加者の自己紹介に「ポジションペーパー」を利用しています。 イベント開始前に記入しておいてください。

※ 共有される前提の資料なので、公開されて困る個人情報は書かないでください

トーク

東京Emacs勉強会では参加者による発表を歓迎しています。目安は15分前後ですが、長くても短くても構いません。

- 使ってみて便利だった機能／パッケージの紹介
- 自分で作っているパッケージや取り組みの紹介
- Emacsについて困っていること／物申したいこと
- ほかのエディタ／IDEで便利だった機能の紹介

グループ

メンバーです

東京Emacs勉強会

イベント数 8回
メンバー数 147人

開催前

2024/08/04(日)
13:00 ～ 18:00

Googleカレンダー icsファイル

このイベントに参加できます

受付票を見る

※受付や入場方法は主催者の案内に従ってください。

募集期間
2024/06/03(月) 21:54 ～
2024/08/04(日) 18:00

イベントへのお問い合わせ

会場

ピクシブ株式会社
東京都渋谷区千駄ヶ谷4丁目23-5 (JPR千駄ヶ谷ビル5F)

pixiv

発表一覧

発表者	内容(仮)
conao3	Emacsで始めるClojureのススメ
tomoya	ローカルLLM on Emacs
kuuote	VimとEmacsのSKK実装の差から見る思想の違い
gamou-tatsumi	iPadの外部キーボード用SKKの開発にまつわる話（これから作り始める）
tadsan	Evolution of Emacs Lispダイジェスト
grugrut	ChromebookでEmacsを動かす
ROCKTAKEY	micで自作use-package！
ayatakesi	これまでの活動について