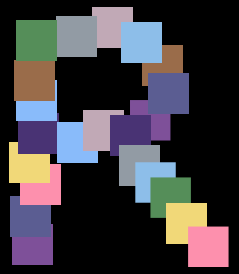
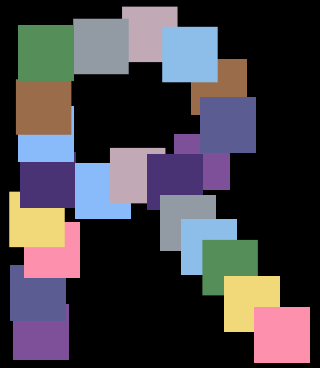
micで
自作use-package!

ROCKTAKEY

2024-08-04



自己紹介 - ROCKTAKEY/ROX



Blog: https://blog.rocktakey.com/
X: @rocktakey
Mastodon: @rocktakey@mstdn.jp

- 読み
  - ろっくてぃきー
  - 長いのでROXでも
- 作ったEmacsのパッケージ:
  - grugru
    - https://github.com/ROCKTAKEY/grugru
  - lsp-latex
    - https://github.com/ROCKTAKEY/lsp-latex
- よく使う言語:
  - Emacs Lisp
  - C++
  - Scheme（主にGNU Guix周り）
- 専攻: 生物、神経科学
- 趣味: プログラミング、数学、物理、自宅サーバ、釣り、ゲーム(スマブラ)

Emacsの設定ファイル

- ほとんどが以下のどれか
  - カスタム変数の設定
    - ユーザによってカスタマイズされることを前提とした変数

```
(customize-set-variable
 ;; '変数
 'package-user-dir
 ;; 値
 (concat "~/.emacs.d/elpa/" emacs-version))

;; 最近のEmacsではこう書いてもよいらしい
(setopt package-user-dir (concat "~/.emacs.d/elpa/" emacs-version))
```

  - キーバインドの設定
    - なんのキーを入力したらなにが起きるかの紐付け

```
(define-key
 ;; キーマップ
 ;; どのモードのときのキーバインドを設定するかを指定
 global-map
 ;; キー (`kbd'を使うと簡単に書ける)
 (kbd "C-x C-b")
 ;; 呼び出されるコマンド
 #'ibuffer)

;; `global-map'(デフォルトのキーマップ)を対象にするときは
;; これでもよい
(global-set-key (kbd "C-x C-b") #'ibuffer)
```

  - Hookの設定
    - なにかを契機に実行される関数群
      - 文字を挿入したとき(post-self-insert-hook)
      - C言語のファイルを開いたとき(c-mode-hook)

```
(add-hook
 ;; フック名
 'c-mode-hook
 ;; 実行される関数
 #'lsp)
```

Emacsの設定ファイル

- ちょっと凝りはじめると
  - コマンドや関数の定義
  - アドバイスによるデフォルトの挙動変更
  - require、autoloadの手動設定
  - Faceの変更
  - etc…

設定ファイルを簡略化するマクロ

use-package

```
(use-package lsp-mode
  ;; `define-key'に展開される
  :bind
  (
   ;; `global-map'の場合は`:map'を省略可能
   ("M-r" . lsp-rename)
   ;; キーマップを指定
   :map lsp-mode-map
   ("M-c" . lsp-execute-code-action))

  ;; `custom-theme-set-variables'に展開される
  ;; `customize-set-variable'の亜種
  :custom
  (lsp-session-file
   ;; `cdr'は`custom-theme-set-variables'によって評価される
   (expand-file-name "etc/.lsp-session-v1" user-emacs-directory))
  (lsp-log-io nil)
  (lsp-log-max nil)

  ;; `eval-after-load'に包まれる
  ;; パーケージロード直後に実行
  :config
  (message "Hello world"))
```

leaf

```
(leaf lsp-mode
  ;; `define-key'に展開される
  :bind
  (
   ;; `global-map'の場合は省略可能
   ("M-c" . lsp-execute-code-action)
   ;; キーマップの指定
   (:lsp-mode-map
    ("M-r" . lsp-rename)))

  ;; `customize-set-variable'に展開される
  :custom
  ;; `cdr'は`backquote'とカンマを使うと
  ;; マクロ展開時に評価される
  `(lsp-session-file
    . ,(expand-file-name "etc/.lsp-session-v1" user-emacs-directory))
  (lsp-log-io . nil)
  (lsp-log-max . nil)

  ;; `eval-after-load'に包まれる
  ;; パーケージロード直後に実行
  :config
  (message "Hello world"))
```

マクロを用いる利点

- 設定がパッケージ毎にまとまって便利
- キーワードによる簡略化
  - 繰り返し同じボイラープレートを書かなくてよい
- 設定の中身を見てrequireしたりautoloadしたりしてくれる

use-package，leafマクロの特徴

カスタマイズ可能

- 同じマクロでも、人によって設定が違う
- 設定の違う複数のマクロを簡単に用意できない

適当に書いても許される

- マクロ側がよしなに解釈
- 使える記法が枯渇している
- 記法を揃えたくてもコケてくれない

キーワード定義がやや煩雑

- 何度も使うものはキーワードに切り出すことが可能
- でもキーワードの追加自体が意外と大変

mic

カスタマイズ不能

- 外付けで新しいマクロを定義するのみ
- 複数の亜種を定義可能

適当に書くことが許されない

- デフォルトではPlistのみを受け付ける
- 外付けで仕様変更は可能

キーワード追加が簡単

- Plistを別のplistに変換する関数を用意するだけ

https://github.com/ROCKTAKEY/mic

使用例: https://github.com/ROCKTAKEY/dotfiles/blob/master/emacs/.emacs.d/init.el

micのコア mic-core

- 内容はたったこれだけ
- Pseudo-plistは許容せず
- これに外付けで
  機能を追加していく

```
(cl-defmacro mic-core (name &key
                            eval
                            eval-after-load
                            eval-after-others
                            eval-after-others-after-load
                            eval-before-all
                            eval-installation)
  "Configuration package named NAME.
It evaluates each element of EVAL-BEFORE-ALL, EVAL, EVAL-AF
In addition, It will evaluate each element of EVAL-AFTER-LO
 EVAL-AFTER-OTHERS-AFTER-LOAD after load of package NAME."
  (declare (indent defun))
  `(prog1 ',name
     ,@eval-before-all
     ,@eval-installation
     ,@(and (or eval-after-load
                eval-after-others-after-load)
            (list
             (append
              (list 'with-eval-after-load `',name)
              eval-after-load
              eval-after-others-after-load)))
     ,@eval
     ,@eval-after-others))
```

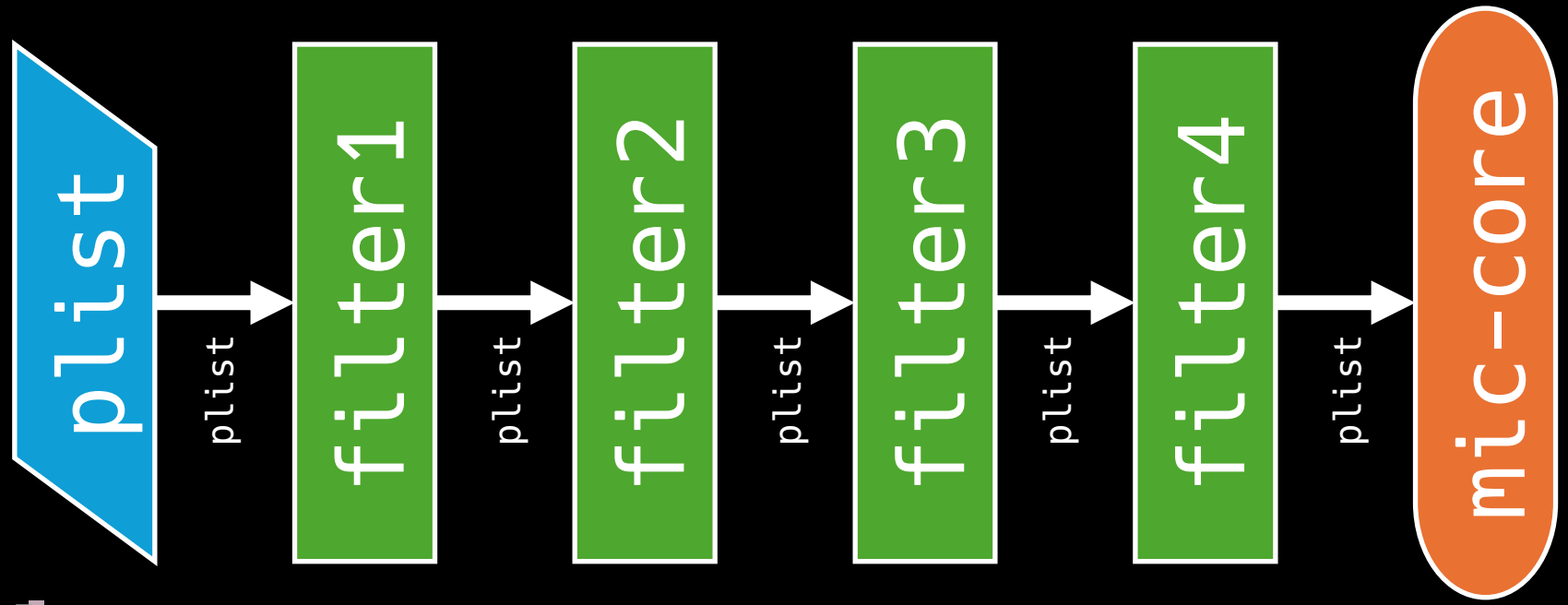
簡易機能追加版 mic

- use-package/leafと同じような機能を簡易的に実装したもの
- キーワードは説明的なので長め
- このまま使うのではなく、自分で機能を外付けすることを推奨

```
(mic lsp-mode
  ;; :bind 相当
  :define-key
  ((global-map
    ;; `cdr' は常に評価される
    ("M-r" . #'lsp-rename)
    ("M-c" . #'lsp-execute-code-action)))
  :custom
  ((lsp-session-file
    ;; `cdr' は常に評価される
    . (expand-file-name
       "etc/.lsp-session-v1" user-emacs-directory))
   (lsp-log-io . nil)
   (lsp-log-max . nil))
  :eval
  ((message "Hello world")))
```

外付け支援ツール mic-defmic

- Filter: plistを別のplistへ変換する
- 例: :customを抜き取ってsetoptへ変換し、:evalに詰め込む



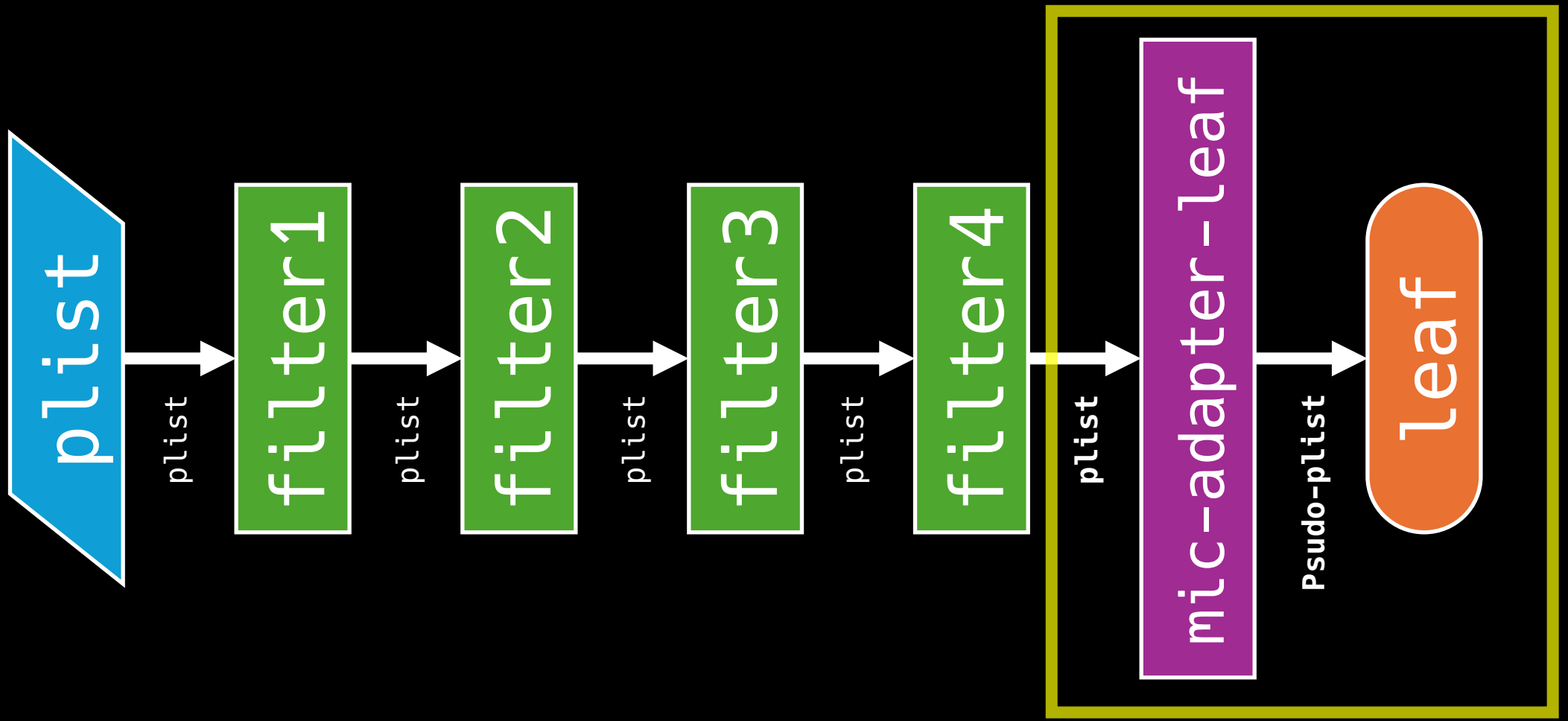

```
(mic-defmic mic mic-core
  :filters
  '(mic-filter-autoload-interactive
    mic-filter-autoload-noninteractive
    mic-filter-auto-mode
    mic-filter-custom
    mic-filter-custom-after-load
    mic-filter-declare-function
    mic-filter-define-key
    mic-filter-define-key-after-load
    mic-filter-define-key-with-feature
    mic-filter-defvar-noninitial
    mic-filter-face
    mic-filter-hook
    mic-filter-package
    mic-filter-require
    mic-filter-require-after
    mic-filter-core-validate))
```

フィルタ定義支援 mic-deffilter

- mic-deffilter-alias
  - キーワードを別のキーワードに載せ換える
- mic-deffilter-const
  - 常になんらかの定数を渡す
- mic-deffilter-t-to-name
  - tが渡されたときにパッケージ名に置き換えてくれる
  - require節がtを渡すだけでよくなったり

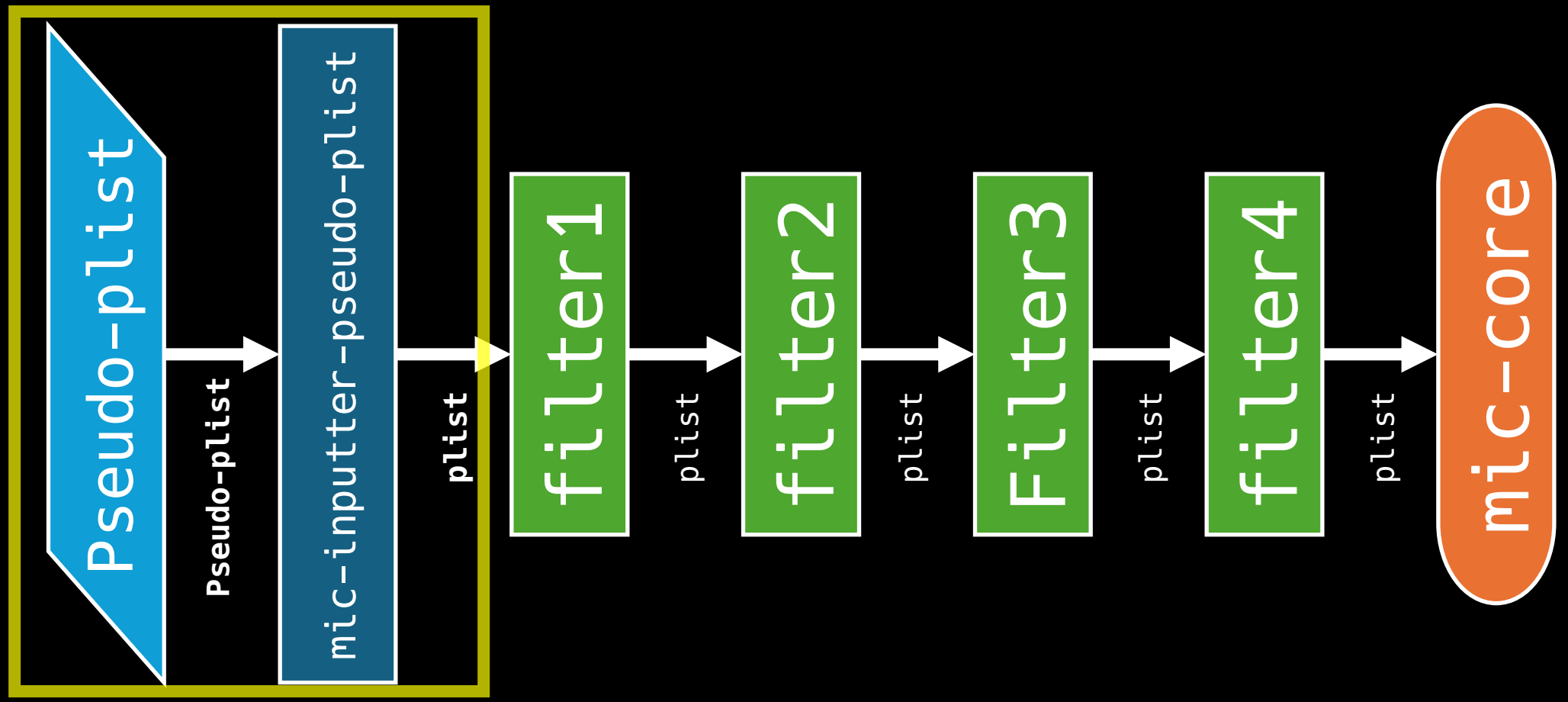
非micマクロへ接続 mic-adapter

- Use-package/leafの機能も使いたいという人向け
- mic-core用の出力をleaf用に変換



Plist以外の入力を受容 mic-inputter

- use-package/leafのような書き方をしたい人向け
- 特定のキーワード入力をplistの1要素に詰め込み



まとめ

- use-package/leafを使うと設定をまとめられて便利
- micのやってくれることは大体同じ
- 重視している点が異なるよ
  - 基本的に自分で拡張することが前提
    - mic-deffilter系統でフィルタ定義
    - mic-defmicで新しいmicを定義
  - インターフェースはいろいろ変えられる
    - mic-adapterでuse-package/leafをバックエンドに
    - mic-inputterで記述方法をuse-package/leaf風に

おわり

フォントはCicaを使わせていただきました