Gleam という選択肢

6/15 関数型まつり 2 日目

こまもか

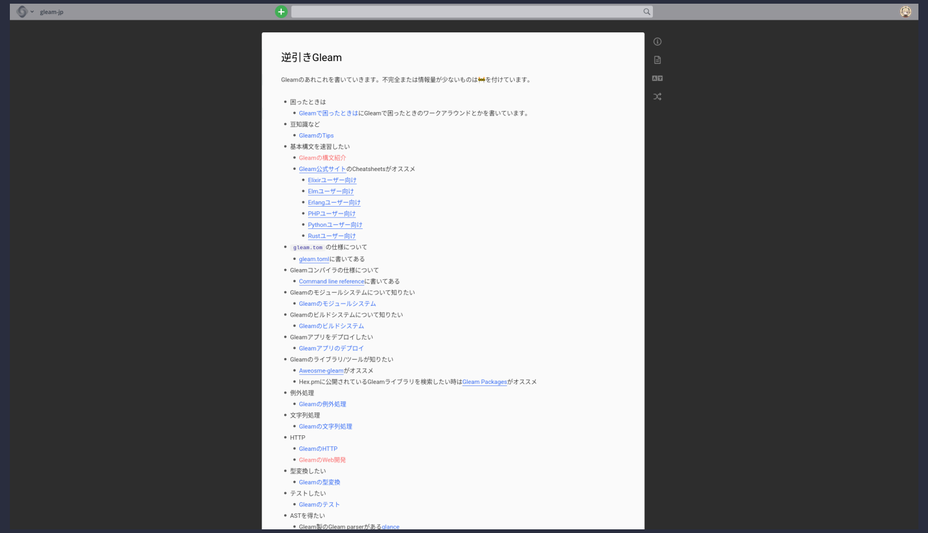
自己紹介

こまもか
Twitter: @Comamoca_

\ Twitter から来ました! /

— 自己紹介

1. 懇親会でこのアイコンを見せると Twitter で見たことあります! と言われる
2. 出身は山梨で、今年就職と同時に上京してきた
3. Gleam をかれこれ 2 年くらい追っている。


gleam-jp
逆引きGleam
Gleamのあれこれを書いていきます。不完全または情報量が少ないものは🚧を付けています。
困ったときは
Gleamで困ったときはにGleamで困ったときのワークアラウンドとかを書いています。
豆知識など
GleamのTips
基本構文を速習したい
Gleamの構文紹介
Gleam公式サイトのCheatsheetsがオススメ
Elixirユーザー向け
Elmユーザー向け
Erlangユーザー向け
PHPユーザー向け
Pythonユーザー向け
Rustユーザー向け
gleam.toml の仕様について
gleam.tomlに書いてある
Gleamコンパイラの仕様について
Command line referenceに書いてある
Gleamのモジュールシステムについて知りたい
Gleamのモジュールシステム
Gleamのビルドシステムについて知りたい
Gleamのビルドシステム
Gleamアプリをデプロイしたい
Gleamアプリのデプロイ
Gleamのライブラリ/ツールが知りたい
Awesome-gleamがオススメ
Hex.pmに公開されているGleamライブラリを検索したい時はGleam Packagesがオススメ
例外処理
Gleamの例外処理
文字列処理
Gleamの文字列処理
HTTP
GleamのHTTP
GleamのWeb開発
型変換したい
Gleamの型変換
テストしたい
Gleamのテスト
ASTを得たい
Gleam製のGleam parserがあるglance

ー 自己紹介

Gleamにマクロは必要なのか？

2024年11月15日

「Gleamにマクロは必要か？」

という議論はGleamコミュニティで度々話題になるテーマで、自分もGleamを好きな人間の一人なのでこれについて色々考えたりしている。

この記事はGleamにとってマクロは必要かどうかを個人的に考察してみた内容を書いてみたもの。いわゆるポエムってやつです。

結論

先に結論を言ってしまうと、**Gleamにマクロは必要ない。またマクロに準ずる方法としてコード生成が良いと考える。**

根拠として、以下が上げられる。

- Gleamの精神性はRustよりGoに近いので、Goで広く用いられているコード生成の方がGleamに向いている。
- async/await, 例外処理、middleware、デコレータは `use` 構文でカバー可能。
- コンパイラの実装や負担が増えるため。
- そもそもGleam自体がある種のマクロである。

Gleamの精神性

Gleamはよくrustに似ていると言われる。Gleamコンパイラ自体Rustで書かれていることもあってか、キーワードなどは非常に似通っている。

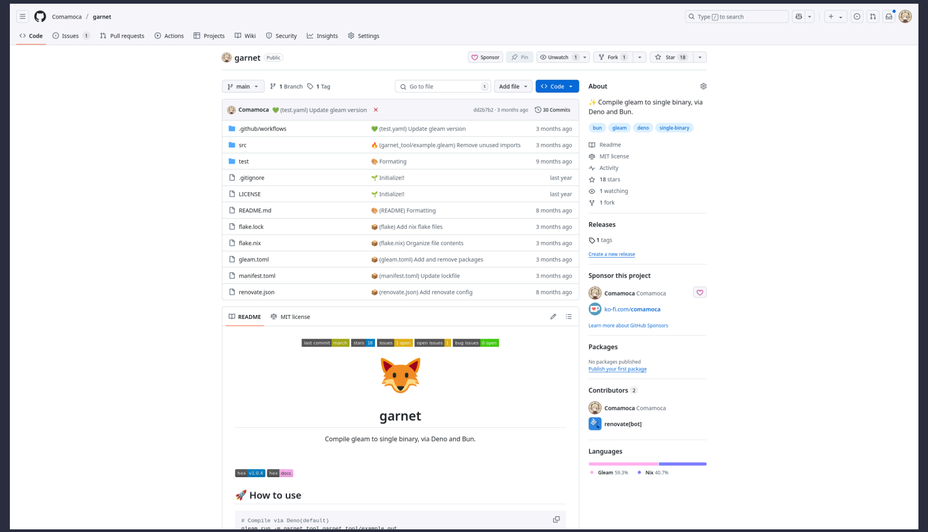
```
// Gleam
fn hello, world!
import gleam/io
```

ー 自己紹介

1. 記事を書いたり

ー 自己紹介

1. 記事を書いたり



— 自己紹介

1. ツールを作ったりしている。
2. このリポジトリは JS Runtime 経由でシングルバイナリを作成するツール
3. esbuild を使ってバンドルしたものをシングルバイナリにしている

注意点

- 基本的に Gleam v1.11.0 を前提にしています。
- 表示の都合上 import などを省略している箇所があります。

Gleam とは

Louis Pilfold 氏が開発している、静的型付けの関数型言語
シンプルな構文と Erlang VM に基づいた並列性が特徴

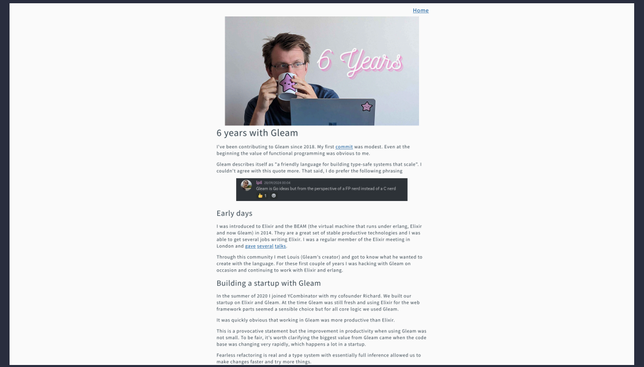
GitHub 19.4K ⭐

— Gleam とは

1. Erlang VM のみならず JS でも動かせる。
2. Louis Pilfold 氏はロンドン在住
3. 日本だと Discord コミュニティのピークタイムが深夜 1 時過ぎぐらいで時差がキツい
4.

シンプル...Go と何が違うの?

Gleam とは



https://crowdhailer.me/2024-10-04/6-years-with-gleam/

— Gleam とは

1. ここでちょっと思想の話になります。
2. シンプルというと Go が思い浮かぶ方もいると思いますが、何が違うのかというとシンプルに対する視点が違います。
3. 6 yearts with gleam っていう記事があるのだけど、そこに作者さんのこういう言葉が紹介されています。

Gleam is Go ideas but from the perspective of a FP nerd instead of a C nerd

— Louis Pilfold

意訳するなら

Gleam は Go の設計思想を取り入れているけど、C オタクの視点じゃなくて FP オタクの視点で解釈した言語だ。

— Louis Pilfold

— 意訳するなら

1. 例えば Go はループを for で行うけれど、Gleam には for はなく再帰で行います。
2. このアプローチはとても関数型的。

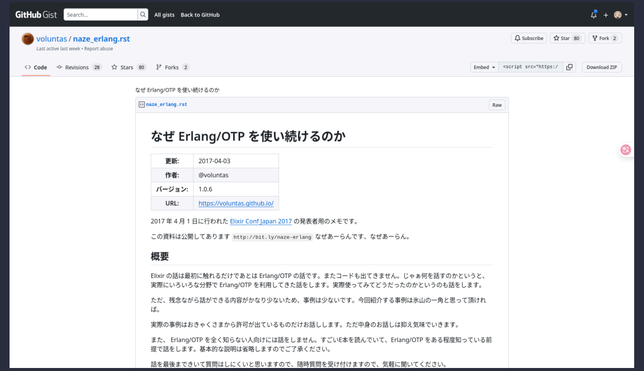
Gleam の特徴

- シンプルな構文
- 関数型言語由来の関数が多いためコードがスッキリする
- エラーメッセージが親切
- Erlang VM / JS Runtime で動く

— Gleam の特徴

1. Gleam は構文がシンプルなので、ある程度プログラミングに慣れている人は半日程度(公式情報)で構文を把握できる。
2. また、`map` や `reduce` 等関数型言語由来の関数が使えるためコードがとてもスッキリするという特徴がある。
3. Gleam はユーザーフレンドリーに重きを置いており、コンパイラのエラーメッセージなどが非常に読みやすくなっている。

Erlnag VM について



https://gleam.run/getting-started/installing/

— Erlnag VM について

1. ここまでは Gleam そのもの考えについて触れてきたけれど、Gleam が動く Erlang VM についてはどうなのか
2. Elixir の発表が多くあり語り尽されているのと、素晴しい文章があるのでこれを紹介するに留める

エラーメッセージの例

```
error: Unknown variable
  ┌─ /src/main.gleam:3:8
  │
3 │   echo prson
  │        ^^^^^ Did you mean person?

The name prson is not in scope here.

warning: Unused variable
  ┌─ /src/main.gleam:2:7
  │
2 │   let person = "Jhon"
  │       ^^^^^^ This variable is never used

Hint: You can ignore it with an underscore: _person.
```

— エラーメッセージの例

1. `person` を `prson` と typo してしまっているのが見て取れる。

LSP

- 型アノテーションの追加
- `import` 文の自動追加
- `case` における不足してるパターンの追加
- パイプ形式への自動変換

— LSP

1. また、LSP によるサポートも充実している。
2. コードアクションの一例としてこれらのコードアクションがある。
3. 最近では JSON のデコーダーを自動生成するアクションなども追加されていて、開発体験がますます向上している。
4. (`gleam/json` ライブラリをプロジェクトに追加しないと発動しない)

Gleam 1.3.0で開発体験が更に向上した

こまもか

Zenn

— LSP

1. 詳しくは Zenn で

構文

- if とか for がない
- コールバックの構文糖(use 構文)
- ブロック構文
- パターンマッチ
- パイプライン演算子

— 構文

1. Elm 経験者いわくサーバーでも動く Elm みたいらしい
2. 基本的な言語機能が少なく標準ライブラリでカバーする方針
3. なので構文を覚えても実用的なプログラムを書くなら標準ライブラリの関数を覚える必要がある

use 構文

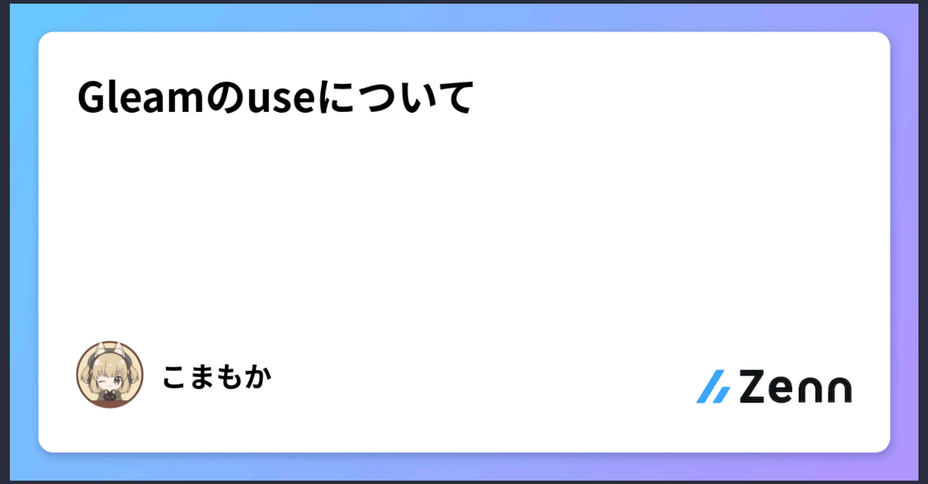
これ一つで

- 例外処理
- 非同期処理
- early return
- middleware

などが表現できる

— use 構文

1. 個人的に一番 Gleam らしいと思う構文



— use 構文

1. 詳しくは Zenn で

例えば

```
let val = True
case True {
  True -> "これは True"
  False -> "これは False"
}
```

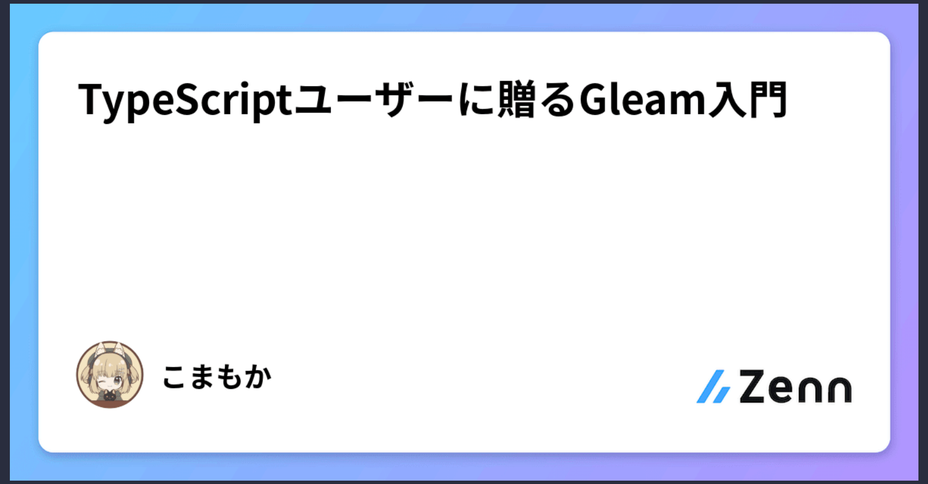
— 例えば

例えば

```gleam
import gleam/list

pub fn main() {
 list.range(0, 10)
 |> list.map(fn (n) { n * 2 })
 |> list.filter(fn (n) {n % 3 == 0})
 |> echo
}
```

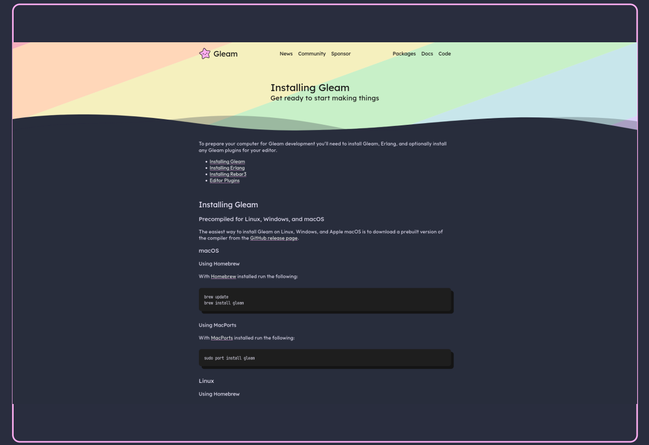

1. パイプライン演算子でデータの処理の流れを直感的に記述できる



— 例えば

1. 詳しくは Zenn で

開発環境



https://gleam.run/getting-started/installing/

— 開発環境

1. 環境構築についてですが、基本的に公式ドキュメントを読めばなんとかなります。

インストール

- brew
- AUR
- apt
- scoop
- Nix

— インストール

1. 主要なパッケージマネージャには大体対応しています。
2. Nix だけはちょっと注意が必要で、nixpkgs にある Gleam は古いのと現状 overlay がないので自分で nix 式を書いて build する必要があります。

拡張機能

- VSCode
- Vim
- Emacs
- Zed

— 拡張機能

1. 拡張機能は公式でこの 4 つがサポートされています。

Gleam のエコシステム

— 拡張機能

Web サーバー

- `gleam/http`
- `mist`
- `wisp`

— Web サーバー

1. 一般的に Gleam 公式の HTTP パッケージをベースに作られているのでライブラリ間で型の互換性が確保されている。
2. web サーバーと合わせて紹介したけれど、web クライアントもこれをベースに開発されている。
3. 型のみを定義することで Erlang VM でも JS Runtime でもサーバーやクライアントを自由に実装できる。
4. mist というのがデファクトな web サーバーになっていて、Gleam で書かれている。
5. wisp は mist をベースに定型的な処理を提供している。

ルーティング

```gleam
import gleam/string_tree
import hello_world/app/web
import wisp.{type Request, type Response}

pub fn handle_request(req: Request) -> Response {
  // ["tasks", "2"]
  case wisp.path_segments(req) {
    [] -> index(req)
    ["hello"] -> greet(req)
    ["tasks", id] -> show_task(req, id)
    _ -> wisp.not_found()
  }
}
```

— ルーティング

1. path_segments っていう関数を使うとリクエストが来た path を文字列のリストに分割してくれる。
2. それをパータンマッチしてルーティングを行う。
3. 先程述べたように、パターンマッチはコンパイラが検証するためこのルーティングもコンパイル時に網羅性が検証される。

ミドルウェア

```gleam
pub fn greet_middleware(req: Request, handler: fn
(Request) -> Response) -> Response {
  io.println("Hello!")
}

pub fn handle_request(req: Request) -> Response {
  use req <- greet_middleware(req)

  case wisp.path_segments(req) {
    [] -> index(req)
    _ -> wisp.not_found()
  }
}
```

— ミドルウェア

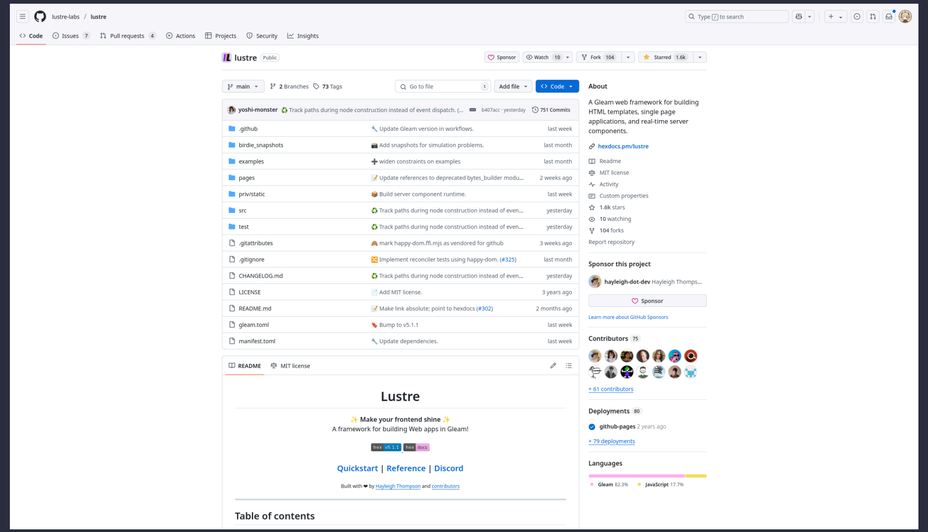
1. リクエストが来たら Hello!を表示するミドルウェア
2. Gleam のハンドラーは Request -> Response という形に抽象化できる
3. ミドルウェアは「Request」と「Request を受け取って Response を返す関数」を受け取って、Response を返す関数 fn (req: Request, next: fn (Request) -> Response) -> Response に抽象化できる。
4. この関数は use を適用できるため、use を複数使って連鎖的にミドルウェアを適用できます。

Lustre

- TEA ベースの Web フレームワーク
- 表示単位が純粋関数なためどこでもレンダリングできる
- CSR, SSR, SSG が可能
- 開発が Lustre dev tools で完結する
- GitHub 1.6K ⭐

— Lustre

1. フロントエンドまわりも色々あるんですが、ボリュームの都合上紹介しきれなさそうなので Lustre に絞って紹介します。
2. Luster は Elm アーキテクチャをベースにした Web フレームワーク。
3. CSR, SSR 両対応でハイドレーションも可能。
4. Gleam のキラーライブライリになりそうだと期待している。
5. 以前は状態を含んだコンポーネントのコストが重かったが、最近になって web components ベースの実装になった影響でかなり軽くなった。

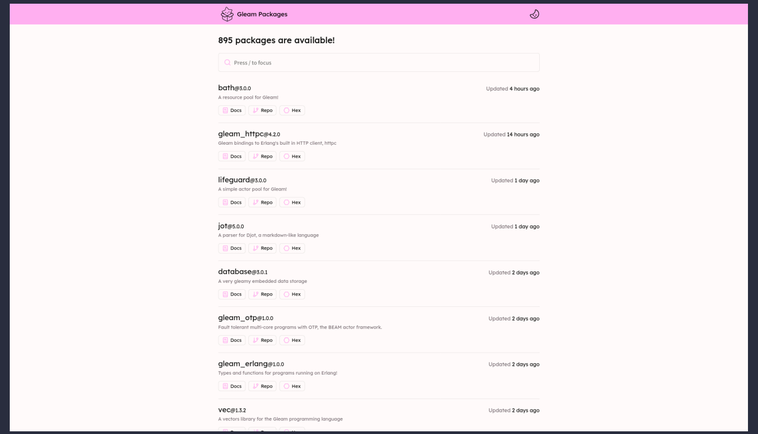

lustre-labs / lustre
Type / to search
Code
Issues 7
Pull requests 4
Actions
Projects
Security
Insights
lustre
Public
Sponsor
Watch 10
Fork 104
Starred 1.6k
main
2 Branches
73 Tags
Go to file
t
Add file
Code
yoshi-monster
Track paths during node construction instead of event dispatch. (...
b407acc · yesterday
751 Commits
.github
Update Gleam version in workflows.
last week
birdie_snapshots
Add snapshots for simulation problems.
last month
examples
widen constraints on examples
last month
pages
Update references to deprecated bytes_builder modu...
2 weeks ago
priv/static
Build server component runtime.
last week
src
Track paths during node construction instead of even...
yesterday
test
Track paths during node construction instead of even...
yesterday
.gitattributes
mark happy-dom.ffi.mjs as vendored for github
3 weeks ago
.gitignore
Implement reconciler tests using happy-dom. (#325)
last month
CHANGELOG.md
Track paths during node construction instead of even...
yesterday
LICENSE
Add MIT license.
3 years ago
README.md
Make link absolute; point to hexdocs (#302)
2 months ago
gleam.toml
Bump to v5.1.1
last week
manifest.toml
Update dependencies.
last week
README
MIT license
Lustre
✨ Make your frontend shine ✨
A framework for building Web apps in Gleam!
Quickstart | Reference | Discord
Built with ❤ by Hayleigh Thompson and contributors
Table of contents
About
A Gleam web framework for building HTML templates, single page applications, and real-time server components.
hexdocs.pm/lustre
Readme
MIT license
Activity
Custom properties
1.6k stars
10 watching
104 forks
Report repository
Sponsor this project
hayleigh-dot-dev Hayleigh Thomps...
Sponsor
Learn more about GitHub Sponsors
Contributors 75
+ 61 contributors
Deployments 80
github-pages 2 years ago
+ 79 deployments
Languages
Gleam 82.3%
JavaScript 17.7%

— Lustre

実例

— Lustre

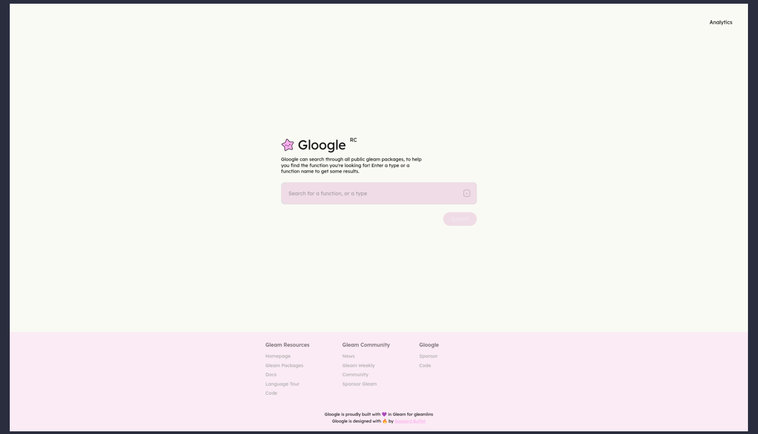
Gleam Packages



— Gleam Packages

1. Gleam のパッケージを検索できるサイト。
2. 内部的には BEAM ファミリー言語向けのパッケージレジストリ hex.pm の API を叩いている。

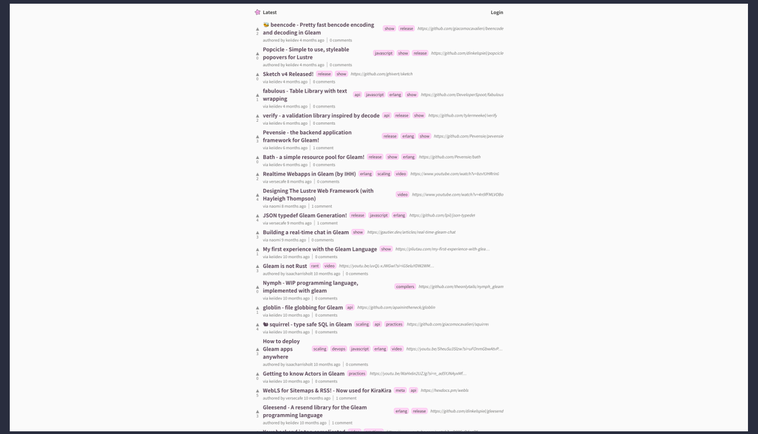
Gloogle



— Gloogle

1. Gleam のパッケージドキュメントを全文検索できるサイト。
2. ここ最近になってパフォーマンスがかなり向上した。

kirakira



— kirakira

1. Gleam で書かれた掲示板。新規登録するには管理人に申請する必要がある。
2. ソースコードが公開されているので、フルスタックなアプリケーションを Gleam で書く際の参考になる。

これからの展望

- 更なる開発支援機能の追加
- コード生成技術の発達
- フルスタックフレームワークの発達
- 新たなコンパイルターゲットの登場

— これからの展望

1. フルスタックアプリケーション FW である pevensie など、エコシステムが成長する余地があるしこれからも成長していくだろうと思う。
2. 個人的な要望としては、現状コード生成するライブラリと解析するライブラリの API が異なるのでこれを統合したい
3. ちゃんとコミュニティ作りたい
4. そろそろ meetup やりたい
5. 認証系のミドルウェアを揃えたい
6. 構文的な不満としてはレコードが冗長な気がするのでスッキリ書けたら嬉しい

寄付について

現在 Louis Pilfold 氏はフルタイムで Gleam を開発しているのですが、残念ながら財政状況は良くないらしいです…
GitHub Sponsors 経由で寄付を行えるので、Gleam を気に入ったらぜひ寄付をお願いします。

— 寄付について

Thank you to all our sponsors! And especially our top sponsor: Lambda.

LAMBDA CLASS

Aaron Gunderson Abel Jimenez ad-ops Adam Brodzinski Adam Johnston Adam Wyluda Adi Iyengar
Adrian Mouat Ajit Krishna Aleksei Gurianov Alembic Alex Houseago Alex Manning Alex Viscreanu
Alexander Koutmos Alexander Stenirud Alexandre Del Vecchio Ameen Radwan Andrea Bueide
AndreHogberg Antharuu Anthony Khong Anthony Maxwell Anthony Scotti Arthur Weagel Arya Irani
Azure Flash Barry Moore Bartek Górny Ben Martin Ben Marx Ben Myles Benjamin Kane
Benjamin Moss bgw Bjarte Aarmo Lund Bjoern Paschen Brad Mehder Brett Cannon Brett Kolodny
Brian Dawn Brian Glusman Bruce Williams Bruno Michel bucsi Cam Ray Cameron Presley
Carlo Munguia Carlos Saltos Chad Selph Charlie Duong Charlie Govea Chew Choon Keat Chris Donnelly
Chris King Chris Lloyd Chris Ohk Chris Rybicki Chris Vincent Christopher David Shirk
Christopher De Vries Christopher Dieringer Christopher Jung Christopher Keele CJ Salem
Clifford Anderson Coder Cole Lawrence Colin Cameron Comet Constantin (Cleo) Winkler
Corentin J. Daigo Shitara Damir Vandic Dan Dan Dresselhaus Dan Strong Danielle Maywood
Danny Arnold Danny Martini David Bernheisel David Cornu Dennis Dang dennistruemper devinalvaro
Diemo Gebhardt DoctorCobweb Donnie Flood Dylan Anthony Dylan Carlson Ed Hinrichsen Edon Gashi
Eileen Noonan eli elke Emma EMR Technical Solutions Endo Shogo Eric Koslow Erik Terpstra
erikareads ErikML erlend-sxeilsson Ernesto Malave Ethan Olpin Evaldo Bratti Evan Johnson evanasse
Fabrizio Damicelli Fede Esteban Felix Fernando Farias Filip Figiel Florian Kraft Francis Hamel
frankwang G-J van Rooyen Gabriel Vincent gamachexx Gavin Panella Geir Arne Hjelle
Georg Hartmann George Georgi Martsenkov gqabbe Giacomo Cavalieri Giovanni Kock Bonetti
Graham Vasquez Grant Everett Guilherme de Maio Guillaume Heu Guillaume Hivert Hammad Javed
Hannes Nevalainen Hannes Schnaitter Hans Raaf Hayleigh Thompson Hazel Bachrach Henning Dahlheim
Henrik Tudborg Henry Warren Heyang Zhou Hubert Małkowski Iain H Ian González Ian M. Jones
Igor Montagner inoas Isaac Isaac Harris-Holt Isaac McQueen István Bozsó Ivar Vong Jacob Lamb
Jake Cleary Jake Wood Jakob Ladegaard Møller James Birtles James MacAulay Jan Pieper
Jan Skriver Sørensen Jean Niklas L'orange Jean-Adrien Ducastaing Jean-Luc Geering Jean-Marc GUERE

ちなみに、寄付を行なうとブログの一番下に名前が載ります。

ー 寄付について