My Web ちぇんじろぐ

2023年
part1
ver. 1

2023年1月から2023年5月までに
登場したライブラリ・フレームワーク
サービスを好き勝手紹介する本

はじめに

本書で扱うこと

本書は著者の趣味で、直近にあったWeb関連の出来事をまとめた本になります。以下のカテゴリーを設定してますが、各カテゴリーは独立しているため、初めから読む必要はありません。説明の粒度は疎らですが、個人的に紹介したいものや思い入れのあるものほど詳しめに書いてます。

- AI編
- WebAssembly編
- 開発ツール編
- フレームワーク編
- サービス編
- ライブラリ編
- 番外編

問い合わせ先

誤植や間違い、その他感想などはGitHubまたはTwitter, メールなどにて、ご連絡いただけると助かります。また、本書ではわかりやすさ優先のため、エンドポイントやリソース名をマスクせずに表示してます。それらのリソースは削除済みなので、ご連絡不要です。

GitHub: https://github.com/kusakabe-t

メールアドレス: pilefort2020@gmail.com

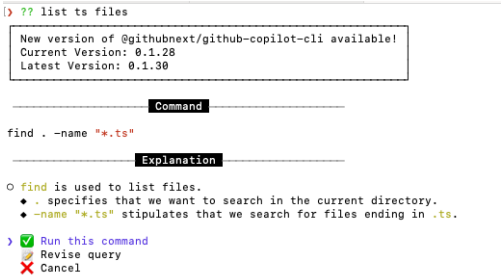
Twitter: @pilefort (https://twitter.com/pilefort)

GitHub Copilot CLI

検証費: 2023年3月22日

GitHub Copilot CLIは、実行したいコマンドを説明するだけで、該当するシェルコマンドやgit, github cliのコマンドを生成するためのツールです。

例えば、tsファイル一覧を取得したい場合は、`?? list all ts files`と入力することで、`find . -name "*.ts"`が提案されます。さらに、Revise queryで`ignore node_modules`を追加すると、`find . -name "*.ts" | grep -v node_modules`が提案されます。このように、シェル上でGitHub Copilotと会話しながら、コマンドを組み立てることができます。

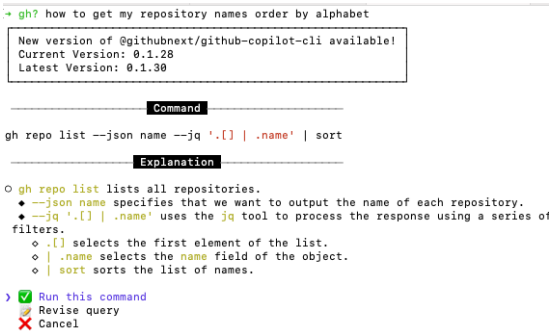


tsファイル一覧を取得するデモ

Revise queryでnode_modules配下のtsファイルを検索から除外するデモ

その他にも、gitコマンドで試したい場合は`git?`で、GitHub CLIの場合は`gh?`で、コマンドを組み立てることができます。また、コマンドを組み立てるだけでなく、各コマンドの意味も教えてくれるため、このコマンドを実行しても大丈夫かどうか判断する材料にもなります。

mainブランチ以外の全てのローカルブランチを削除するデモ



自身のリポジトリをアルファベット順に並べるデモ

DockerのWebAssembly対応

調査日: 2023年1月26日

DockerをWebAssembly化し、軽量で高速な起動が可能になりました。ただし、現状はβ版で本番利用は推奨されてません[1]。

WebAssembly化されたDockerイメージを起動するには、Dockerのruntimeに io.containerd.wasmedge.v1, platformに wasi/wasm32を指定します。

```
docker run -dp 8080:8080 \
  --name=wasm-example \
  --runtime=io.containerd.wasmedge.v1 \
  --platform=wasi/wasm32 \
  michaelirwin244/wasm-example
```

既存のDockerfileのWebAssembly化も以下のコマンドでできます。

```
docker buildx build --platform wasi/wasm32 -t <タグ名> .
```

docker-composeで使う場合は以下のようになり、ビルドや起動方法が変わるだけで、使う側はほぼ何も変わらないです。最近はコンテナを直接デプロイして使う機会が増えてきてるので、AWS Lambdaなどにデプロイする際にWebAssembly化しておくというのは今後の選択肢としてありそうだなと感じてます (WebAssembly化すると、軽量かつ起動が高速になります)。

docker-compose.yml

```
services:
  app:
    image: michaelirwin244/wasm-example
    platform: wasi/wasm32
    runtime: io.containerd.wasmedge.v1
    ports:
      - 8080:8080
```

---

1. https://docs.docker.com/desktop/wasm/↩

WCGI

検証日: 2023年4月30日

WCGI (WebAssembly + CGI) はCGIアプリケーションをWASI (WebAssembly System Interface) にコンパイルし、安全に再利用するための仕組みです[1]。

CGIはリクエストごとに記述されたコードを実行します。これをWebAssembly化した場合は、リクエストごとにサンドボックス化されたプロセスで実行されるため、より安全にコードを実行できるらしいです (詳しくは分かりませんが、既存プロセスを利用した攻撃には効果的に見えます)。

Wasmer 3.2からはWasmer上でPHPとRustのCGIを動かせるようになりました。Rustはcgiクレートをインストールし、wasmにビルドすれば使えます。PHPの方はwasmにビルドする方法がないので、Wasmerが別途提供しているファイルを入れて使います。

例えば、PHPのCGIを動かす場合は以下のファイル構成にします。index.phpに動かしたいコードを記述し、wasmer.tomlでファイルの依存関係を記述します。php-cgi.wasmはWasmerが提供しているPHPのCGIを動かすためのファイルです[2]。

```
app
├── index.php
├── wasmer.toml
└── php-cgi.wasm
```

app/index.php

```php
<? print("Hello, World!"); ?>
```

app/wasmer.toml

```toml
[package]
name = ... (省略)

[[module]]
name = "php"
source = "php-cgi.wasm"
abi = "wasi"

[[command]]
name = "php"
runner = "wcgi"
module = "php"

[command.annotations.wcgi]
dialect = "rfc-3875"

[command.annotations.wasi]
atom = "php"
env = ["DOCUMENT_ROOT=/app", "SCRIPT_FILENAME=/app/index.php"]

[fs]
"app" = "app"
```

---

1. https://wasmer.io/posts/announcing-wcgi↩
2. https://github.com/wasmerio/wcgi-php-template/tree/main↩

ツール編

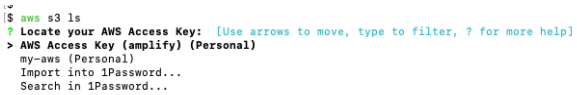
1Password Shell Plugins

調査日: 2023年1月19日

AWS CLIやOpenAI CLI, SSH接続などを利用するには、キー情報をローカルに登録しておく必要があります。AWSの場合は`~/.aws`配下にキー情報が、SSHの場合は`~/.ssh`にキー情報があるなど、ある程度の推測が可能なため、悪意あるライブラリやツールによって、読み取られる危険性があります。また、複数端末で利用したい場合は管理の手間やリスクが高くなります。

1Password Shell Pluginsは1Password上に登録したキー情報を使って、CLIを実行するためのツールです[1]。CLIコマンドを叩くと、1Passwordの認証が要求され、認証を突破すると、1Passwordで登録しているキーを使ってCLIを実行できます。ちなみに、1Passwordへの認証方法はパスワードだけでなく、指紋認証、Apple Watch認証などが使えるらしいです。

1Password Shell Pluginsのデモ



1Password突破後のprofile選択 (AWSの場合)

1Password Service Accounts

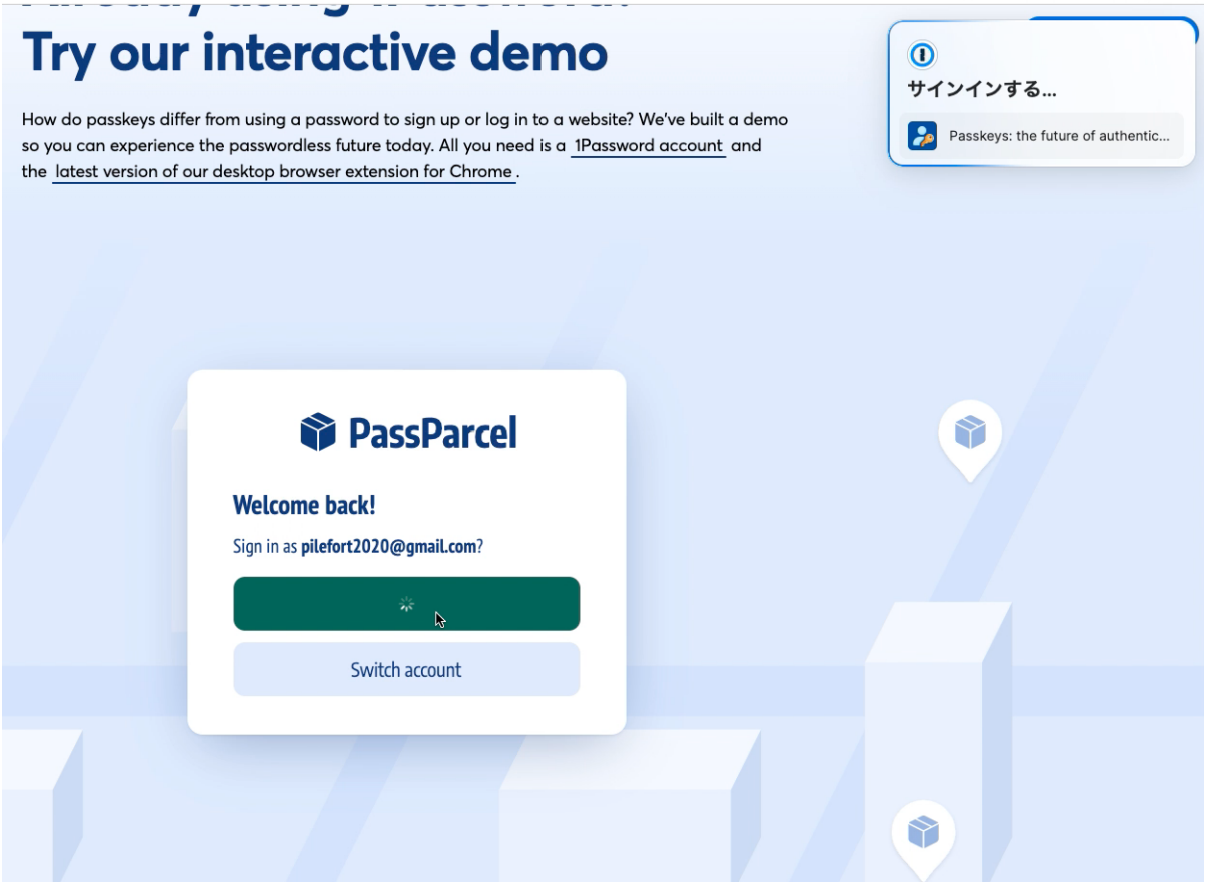
調査日: 2023年5月16日

こちらの機能は実験できてませんが、GitHub ActionsやCircleCIといったCI/CDサービスで、1Passwordが利用できるらしいです[2]。その際に使う1PasswordはService Accountsと呼ばれ、他のユーザーと共有可能なものになるらしいです。まだベータ版のため、機能制限などがありますが、キー情報の分散 (GitHubにAWSキーを登録したり、CircleCIにAWSキーを登録したりなど) を防ぐことができる機能で、大変期待してます。

1Password Passkey対応

調査日: 2023年5月16日

明確な証拠がないのですが、6/6から1Passwordへのログインにpasskeyを導入するという噂[3]もあります。ちなみに、passkeyはスマホの指紋認証やPINコード入力を使って、Webサイトにログインできる仕組みです。passkeyの導入自体はfutureとして紹介されてるので、本当だとありがたいです[4]。ちなみに、passkeyが利用可能なサイトはpasskeys.directoryで確認できます。現状だと、GoogleやPayPal, Microsoft, Docomoなどで利用できます。



1Password Passkey対応のデモ

---

1. https://developer.1password.com/docs/cli/shell-plugins↩
2. https://developer.1password.com/docs/service-accounts/↩
3. https://www.theverge.com/2023/5/16/23725223/1password-passkey-date-password-manager↩
4. https://www.future.1password.com/passkeys↩

Qwik

検証日: 2023年5月7日

QwikはDeno FreshやAstroと同様に、最初にHTMLを送信し、後からJavaScriptが使えるようになるフロントエンドフレームワークです。ただし、Deno Fresh, Astroの場合はコンポーネントが表示されたタイミングでJavaScriptをダウンロードしますが、QwikはJavaScriptを必要とする要素がクリックされるまで、JavaScriptのダウンロードと実行を可能な限り遅らせます[1]。

例えば、Qwikで以下のようなカウンター用コンポーネントを作成したとします (Qwikもjsxが使えます)。

```
// クリックすると、カウントアップするコンポーネント
export const Counter = component$(() => {
  const count = useSignal(0);

  return (
    <button onClick$={() => count.value++}>
      {count.value}
    </button>
  )
});
```

上記のコードはQwikによって以下のように変換されます。

```
const Counter = component(qrl('./chunk-a.js', 'Counter_onMount'));
```

`$`の数だけ (今回の例だと、`component$` と `onClick$`) ファイルが分割されていき、JavaScriptの読み出しが細かく別れます[2]。結果、このカウンターコンポーネントをクリックすると、`chunk-a.js`と`chunk-b.js`だけがダウンロードされ、カウントアップが動くようになります。

chunk-a.js

```
export const Counter_onMount = () => {
  const count = useSignal(0)

  return (
    <button
      onClick$={qrl('./chunk-b.js', 'Counter_onClick', [count])}
    >
      {count.value}
    </button>
  )
};
```

chunk-b.js

```
const Counter_onClick = () => {
  const [count] = useLexicalScope()

  return count.value++
};
```

---

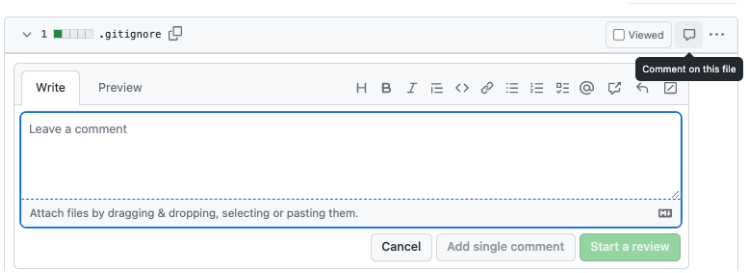
1. https://qwik.builder.io/docs/think-qwik/↩
2. https://qwik.builder.io/docs/advanced/optimizer/↩

サービス編

GitHub

ファイルにPRコメント

画像やバイナリファイルなどに対して、PRコメントを追加できるようになりました[1]。



ファイルにPRコメントをつけるサンプル

Dependabotが作成したPRの自動クローズ

調査日: 2023年4月30日

Dependabotが作成したPRが30日間放置されると、PRが自動クローズされるようになりました[2]。

ブランチ、タグの命名ルール設定

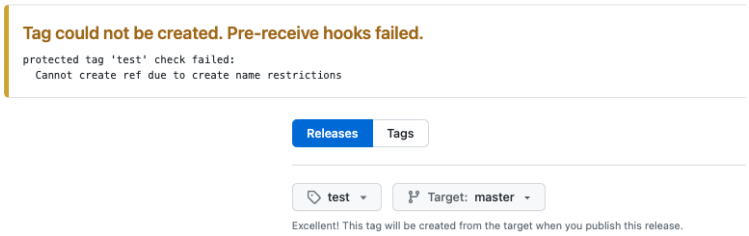
検証日: 2023年4月30日

ブランチ名やタグ名の命名ルールを設定できるようになりました[3]。ルールに反するブランチ名やタグ名を設定すると、以下のようなエラーになります。

また、現状 (2023年4月23日) ユーザー指定はできませんが、チーム単位で特定のブランチ名やタグ名の作成権限の制限もできます。チーム単位とはいえ、タグを切ってデプロイしてる方にはより安全になる機能だと思います。

```
$ git branch
  master
* test
$ git push origin head
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote: error: GH006: Protected branch update failed for refs/heads/test.
remote: error: Cannot create ref due to create name restrictions
To github.com:kusakabe-t/                    git
 ! [remote rejected] head -> test (protected branch hook declined)
error: failed to push some refs to 'github.com:kusakabe-t/                .git'
$
```

ブランチ名の命名ルールに反する場合



タグ名の命名ルールに反する場合

GitHub Actions Importer

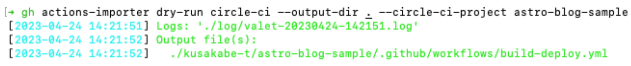
検証日: 2023年4月30日

これはCircleCI, Jenkins, Travis CIなどからGitHub Actionsに移行するためのツールです[4]。

例えば、CircleCIの設定ファイルをGitHub Actionsに変換するときは、CircleCIのアクセストークンとGitHubのアクセストークンを使い、以下のコマンドで変換できます。

```
# アクセストークンの登録
$ gh actions-importer configure

# 変換
$ gh actions-importer dry-run circle-ci --output-dir . \
  --circle-ci-project <CircleCIでのプロジェクト名>
...
```



GitHub Actions Importerのデモ

以下のように、CircleCIのconfig.ymlはgithubの設定ファイルに変換されます。

.circleci/config.yml

```yaml
version: 2.1
jobs:
  lint:
    docker:
      - image: cimg/node:16.15.1
    steps:
      - checkout
      - restore_cache:
          key: dependency-cache-{{ checksum "yarn.lock" }}
      - run:
          name: Install Dependencies
          command: yarn
      - run:
          name: Lint Check
          command: yarn lint
      - save_cache:
          key: dependency-cache-{{ checksum "yarn.lock" }}
          paths:
            - ./node_modules
```

```
workflows:
  build-deploy:
    jobs:
      - lint
```

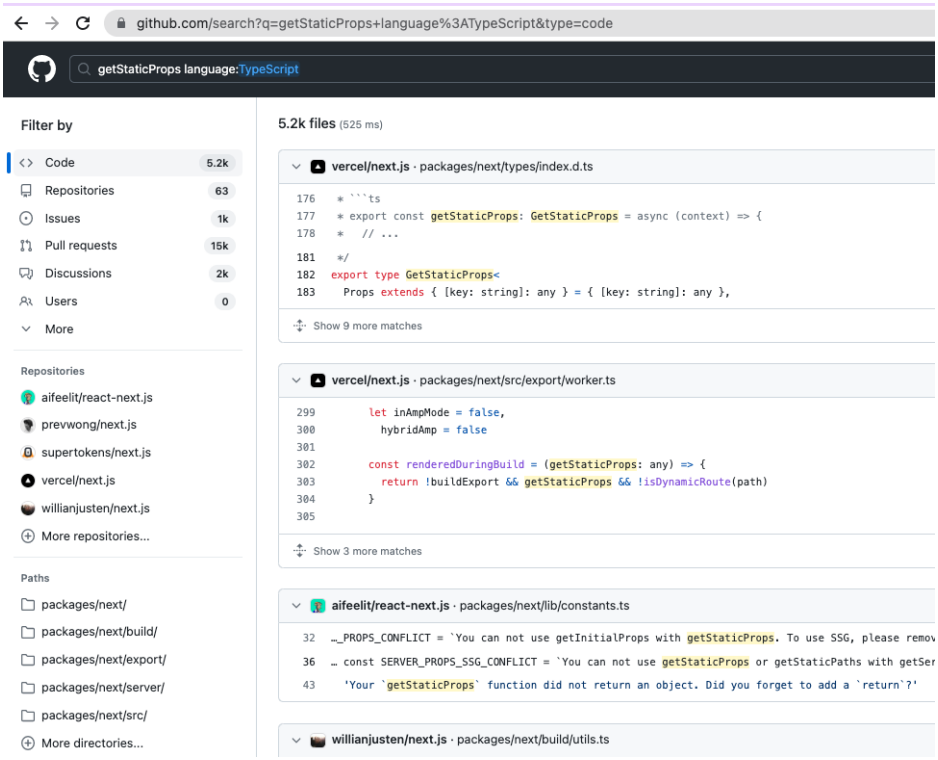
build-deploy.yml

```
name: <CircleCIの組織名>/<CircleCIのプロジェクト名>/build-deploy
on:
  push:
    branches:
    - master
jobs:
  lint:
    runs-on: ubuntu-latest
    container:
      image: node:16.15.1
    steps:
    - uses: actions/checkout@v3.5.0
    - name: restore_cache
      uses: actions/cache@v3.3.1
      with:
        key: dependency-cache-{{ checksum "yarn.lock" }}
        path: "./node_modules"
    - name: Install Dependencies
      run: yarn
    - name: Lint Check
      run: yarn lint
```

GitHub CodeSearch

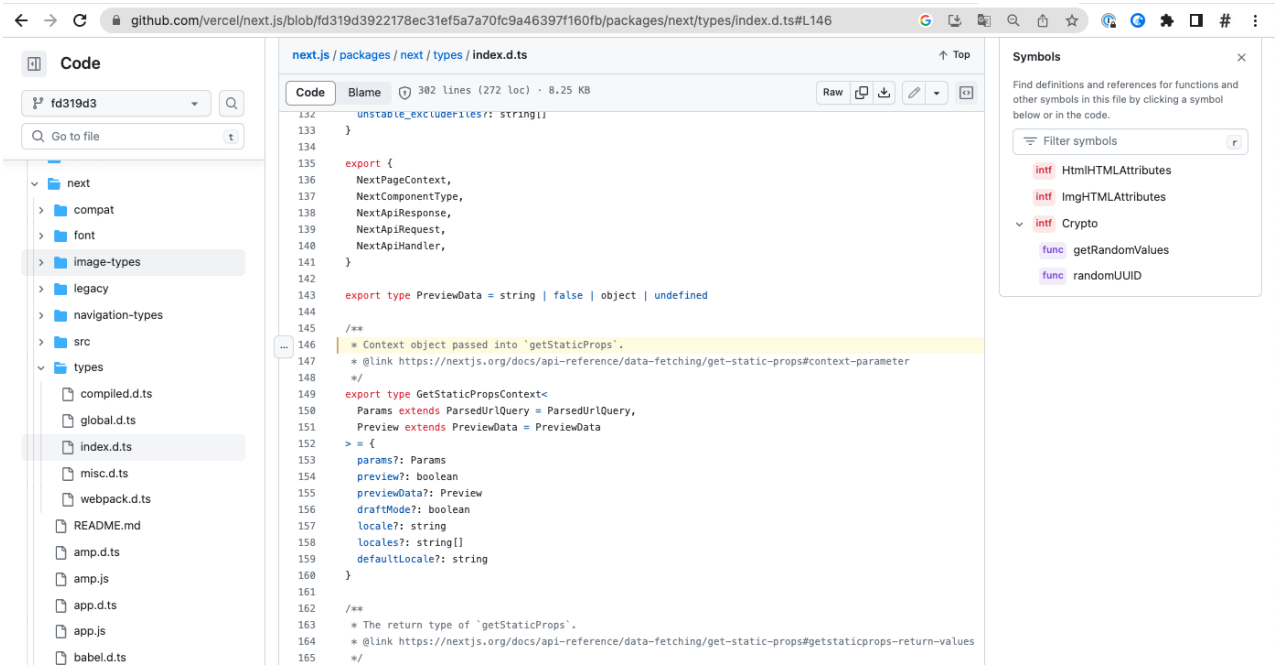
検証日: 2023年3月2日

GitHub CodeSearch (β版) が利用可能になり、コードの検索性能が抜群に向上しました。コード検索する際に、言語や拡張子、リポジトリなどの指定もできるようになりました[5]。



GitHub CodeSearchのデモ

また、ファイル内の表示も型や関数を認識し、サイドバーでワンクリックで選択できるようになりました



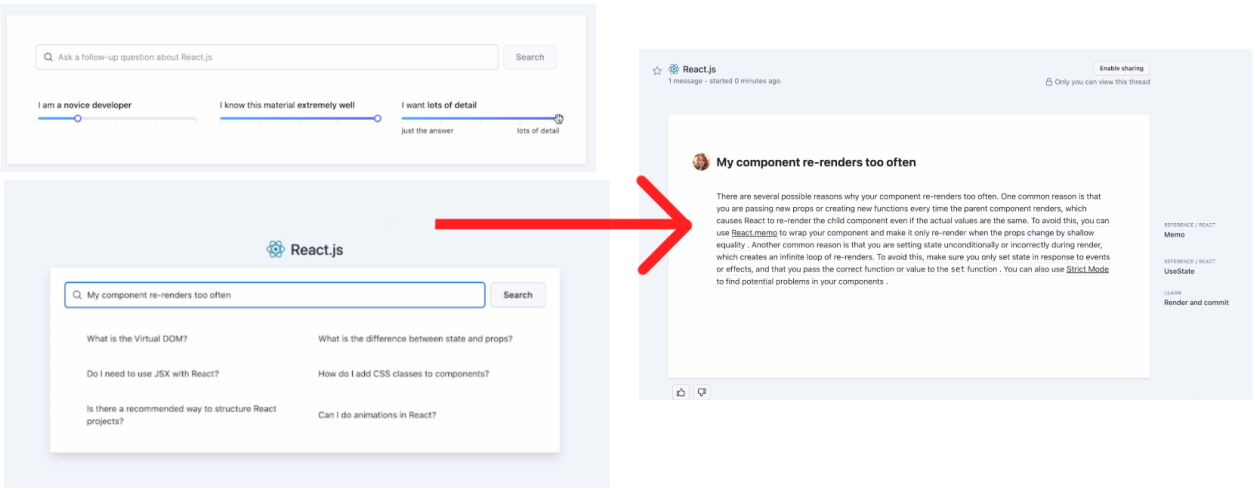
GitHub CodeSearchのデモ2

Copilot for Docs

調査日: 2023年4月30日

これはGitHub上の特定のリポジトリをデータソースとして、GPT-4ベースのアプリを通して、質問や調査ができるサービスです[6]。waiting list待ちのため、詳細は不明ですが、リポジトリ内のissueやdiscussionの内容もデータソースにできれば、特定のライブラリごとのGPTサービスになるので、すごく便利になりそうだと感じてます。

ちなみに検索時は、開発者としての経験、理解度、回答の詳細度を指定でき、以下のように回答してくれるそうです。



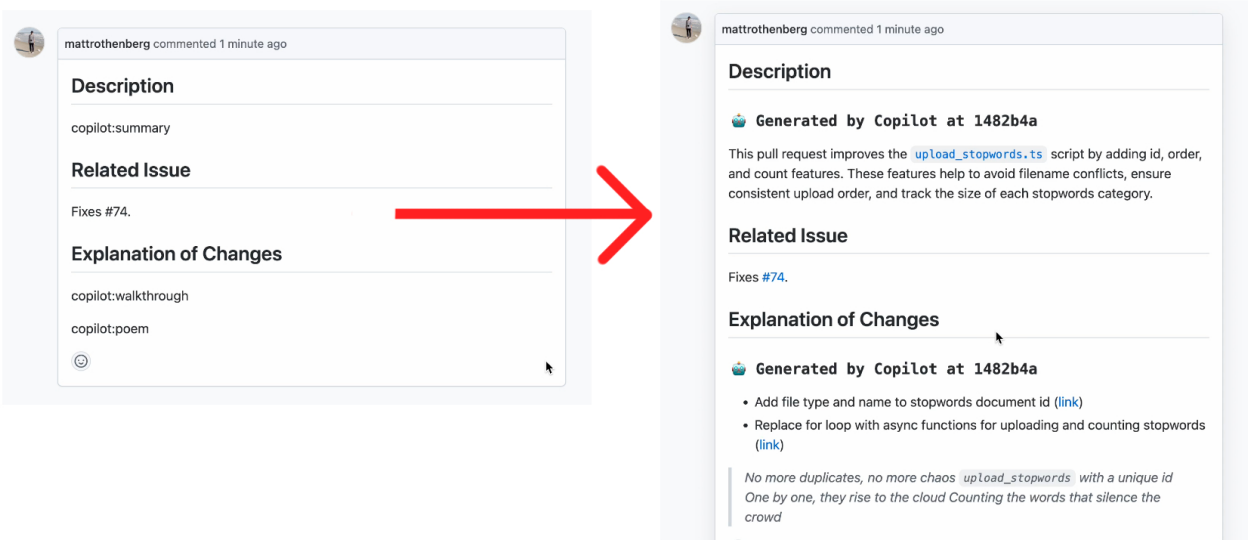
GitHub Copilot for Docs

Copilot for PRs

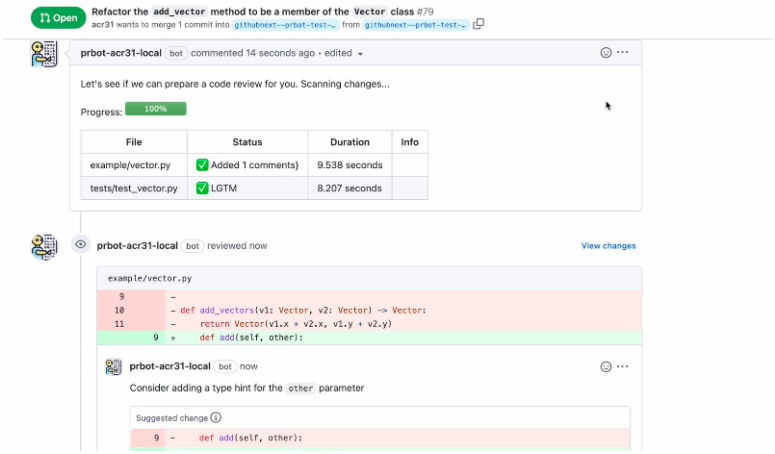
調査日: 2023年4月30日

こちらは、リポジトリにCopilot (GPT-4ベース) を連携し、PRコメントの自動生成やコメント補完、PRレビュー、テストコードの提案などをするための機能です[7]。

以下のように、PRの説明を自動生成したり、PRのレビューなどをしてくれるそうです (公式の説明より引用)。



GitHub CopilotによるPR説明の自動生成



GitHub Copilotによる自動PRレビュー

---

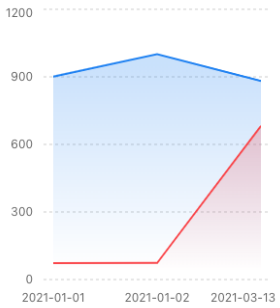
1. https://github.com/community/community/discussions/49954↩
2. https://github.blog/changelog/2023-04-13-automatic-rebases-on-dependabot-pull-requests-stop-after-30-days-of-inactivity/↩
3. https://github.com/github/roadmap/issues/477↩
4. https://github.com/github/roadmap/issues/639↩
5. https://github.com/features/code-search↩
6. https://githubnext.com/projects/copilot-for-docs/↩
7. https://githubnext.com/projects/copilot-for-pull-requests/↩

tremor 2.0

調査日: 2023年5月6日

tremorはダッシュボードを構築するためのReactのコンポーネントライブラリです[1]。面グラフ、棒グラフ、折れ線、テーブル、データバーなど様々なチャートを使えます。

例えば、以下のような面グラフを書く場合は以下のように記述するだけです。簡単な記述で綺麗なグラフを作成できるため、管理画面などの利用で大変便利だと感じます。ちなみに、Tremor 2.0は1.xと比較して、TailwindCSSとの連携が強化されました。



tremorのデモ

```
import { AreaChart } from "@tremor/react";

// グラフに使用するデータ
export const data = [
  {
    date: "2021-01-01",
    Sales: 900.73,
    Customers: 73,
  },
  // ...
];

export default function ChartView() {
  return (
      <AreaChart
        data={data}
        // 横軸に使用するデータ
        index="date"
        // 面グラフとして描画するデータ
        categories={["Sales", "Customers"]}
        // Salesは青でCustomersを赤で表現する
        colors={["blue", "red"]}
        // 凡例の表示
        showLegend={false}
        yAxisWidth={56}
        className="mt-8 w-[300px]"
      />
  );
}
```

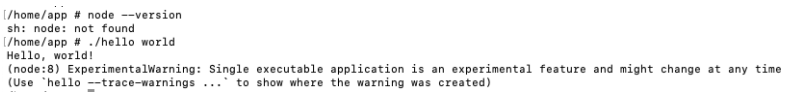
1. https://www.tremor.so/↩

番外編

Node.js Single Executable Applications (experimental)

検証日: 2023年5月6日

Node.js 18.16.0 から実験機能として、Single Executable Application (SEA) を生成できるようになりました[1]。Single Executable Applicationsは `node index.js` で動くようなコードを実行ファイル化し、Node.jsなしで動かせるようにする機能です[2]。



Single Executable Applicationsのデモ

必要なファイルは実行したいファイル (hello.js) と構成ファイル (sea-config.json) だけです。構成ファイルには、どのファイルを実行ファイルにするかを指定します。

index.js

```
console.log(`Hello, ${process.argv[2]}!`);
```

sea-config.json

```
{ "main": "hello.js", "output": "sea-prep.blob" }
```

環境により実行コマンドが異なりますが、以下のコマンドでindex.jsから実行ファイルを生成します。`--experimental-sea-config` でSEA (Single Executable Application) の構成ファイルをsea-config.jsonファイルから生成します。

```
$ node --experimental-sea-config sea-config.json
```

`cp $(command -v node) hello` で現環境に存在するnodeの実行コマンドをファイル化します (node hello.jsとhello hello.jsが同じ結果を示すようになる)。

```
$ cp $(command -v node) hello
```

最後に、postjectコマンドでhelloというファイルにhello.jsの実行コードが挿入され、helloファイルの実行だけで、`node hello.js` と同じ結果が得られるようになります。こちらは、Linux, MacOS, Windowsで生成できますが、OS間の互換性はないため、利用時に注意が必要です (Linuxで生成した実行ファイルはMacOSなどでは実行できない)。

```
$ npx postject hello NODE_SEA_BLOB sea-prep.blob \
    --sentinel-fuse NODE_SEA_FUSE_fce680ab2cc467b6e072b8b5df1996b2
```

Node.js 20ではその他の機能として、Permissionsという概念も追加されました[3]。詳しく紹介しませんが、こちらはNode.jsの実行コードにアクセス権限を付与し、指定されたリソース以外の読み取りを防ぐ機能です。

---

1. https://nodejs.org/en/blog/release/v18.16.0↩
2. https://nodejs.org/api/single-executable-applications.html#single-executable-applications↩
3. https://nodejs.org/dist/v20.0.0/docs/api/permissions.html#process-based-permissions↩