B5 Cover Page

(182x257mm or 516x728pt)

Rust と Github Pages で公開する Web アプリ開発

— クラウドにお金を払いたくない人のための開発入門 —

［著］J-IMPACT

技術書典 18（2025 年夏）新刊
2025 年 5 月 31 日　 ver 1.0

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、Ⓡ、Ⓒ などのマークは省略しています。

はじめに

無料で運営し、使われ、成長する Web アプリケーションへ

本書で使用する技術と、その選定理由

♣ **Github Pages**

目次

第 1 章

環境構築と技術解説

1.1 Github Pages

Github Pages（ギットハブ・ページズ）は、Github 上のリポジトリを使って Web サイトを無料で公開できるサービスです。Github のアカウントがあれば誰でも使用できる機能なので、アカウントをお持ちでない場合は Github のサイトでアカウント作成（サインアップ）してください。

[外部リンク] Github
https://github.com/

このような機能を**ホスティングサービス**といいます。一般に、ホスティングの形式には**静的・動的**の 2 種類が存在していますが、Github Pages は静的ホスティングにのみ対応しています。静的ホスティングとは、Web サイトを見にきたクライアント（ユーザ）に対して、常に同じ内容のページを送信するサービスです。それに対し動的ホスティングでは、クライアントからのリクエスト内容に応じて、送信するページの内容を「動的に」生成します。

これを聞いて、静的ホスティングしかできない Github Pages では、Web アプリを運営することはできないのではないか？ と感じた読者も多いと思います。なぜならば、私たちが日頃から利用している Web アプリの多くが、ユーザからの様々なリクエストに対して適切な応答を返すサービスであるからです。

しかし、ホスティングが静的であることと、Web ページの内容が常に固定されていることは同義ではありません。Web ページに**スクリプト**や**バイナリ**を含めることによって、見た目には「動的な」応答を作り出すことができます。

確かに、動的ホスティングサービスを使って作れる Web アプリと比べると、Github Pages が提供できる機能には制限があります。たとえば、ログイン機能はホスティングサーバでの照合処理が必要になるため、Github Pages では実装することができません。しかし、私たちが Web アプリに求める多くの機能は、実は静的ホスティングでも十分に対応が可能なのです。

本書では、静的ホスティングでも提供可能な様々な機能を紹介します。もし、読者の頭に「こういうサービスを気に入ってくれる人がいるのでは？」というアイデアが浮かんで、それが本書で紹介した機能で実現可能なのであれば、Github Pages を使って「サッ」と公開してしまいましょう。そして、実際にユーザに使ってもらってフィードバックを受け、サービスをさらに良いものにする…という流れが実現されたなら、著者としてこれ以上に嬉しいことはありません。

1.2 WebAssembly（WASM）

静的ホスティングで「動的な」機能を実現するためには、**Javascript**（JS）を使用することが多いです。しかし本書では **WebAssembly**（WASM）を使用します。JS がスクリプトであるのに対し、WASM はバイナリであるという違いがあります。

私たちは人間が読める言語を使ってプログラミングをします。この人間語がスクリプトです。これを Web アプリとして実行するためには、インタプリタによって機械語へ翻訳する必要があります。一方、WASM 等のバイナリは機械語への翻訳後（コンパイル済み）のファイルなので、実行時に翻訳する必要がありません。この、実行時に翻訳の手間を省けることに加え、WASM が C/C++ や Rust などの高速な言語をコンパイルしたものであることから、Javascript に比べて実行速度がかなり速くなります。本書では、Web アプリで WASM の機能が生きる例も紹介します。

1.3 Rust

WASM を生成するために、本書では **Rust** を使用します。Rust は安全性・高速性・並列性に優れた開発言語です。安全性とは、悪意を持ったユーザの攻撃対象となるようなバグを作り込みにくいということであり、高速性と並列性はそのままプログラムの軽快な動作に繋がります。このように Web アプリにとって嬉しい特徴を備えていることから、本書では Rust を採用しました。Rust は Web アプリに限らず、オペレーティングシステム（OS）開発や組み込み開発などのコアな業界でも広く採用されており、今後もシェアを大きく伸ばしていくと予想されています。

♣ Rust のインストール

Rust をインストールする手順は、公式サイトを参考にしてください。

[外部リンク] Rust プログラミング言語（公式）
`https://www.rust-lang.org/ja`

インストール作業後にリスト 1.1 のコマンドを実行し、Rust のバージョン情報が表示されたら、インストール手順が正しく完了したことを確認できます。

▼リスト 1.1: Rust のバージョン確認: shell

```
> rustc --version
```

♣ WASM 向け target のインストール

Rust のインストールが完了すると、Rust 言語で記述したコードを**ビルド**（コンパイル）してバイナリを作ることができるようになります。ただし、このバイナリは**どこで動かすか**（実行環境）によって中身が変わります。たとえば、Windows 用・macOS 用・Linux 用では、それぞれ違う形式のバイナリが必要になります。

Rust では、どの環境向けにバイナリを作るかを**ターゲット**によって指定します。このターゲットは 3 つの情報で表現されており、まとめて**ターゲットトリプル**（target triple）といいます。

▼リスト 1.2: ターゲットトリプルの構造

```
<アーキテクチャ（CPUの種類）>-<ベンダー（提供元）>-<OSの種類>
```

Rust は普段、自分が動いている PC 向けのターゲットでバイナリを作ります。しかし、任意のターゲットをインストールすることで、その環境向けのバイナリを生成することができます。この機能を**クロスビルド**（クロスコンパイル）といいます。

WASM 向けのターゲットは `wasm32-unknown-unknown` です。`wasm32` は WebAssembly の 32 ビットアーキテクチャを意味します。ベンダー情報・OS 情報がともに `unknown` であり、特定の提供元や OS に縛られていないことから、WASM は非常に広い環境で動作することがわかります。

本書を読み進めるにあたっては、このターゲットをリスト 1.3 のコマンドでインストールしておいてください。

▼リスト 1.3: WASM 向けターゲットのインストール: shell

```
> rustup target add wasm32-unknown-unknown
```

1.4　Yew

Yew は Rust で Web フロントエンドアプリケーションを作るためのフレームワーク（ライブラリ）です。WASM を生成するためのコードを、安全性に優れた Rust で記述することができます。なお、Rust ではライブラリのことを**クレート**と呼びます。

Yew などのクレートを使用するためには、Rust のプロジェクト内で使うクレートとバージョンを宣言します。具体的な方法は 2 以降で解説します。

1.5 Trunk

Yew を使って記述した Rust のコードから WASM を生成するために、**Trunk** を使用します。リスト 1.4 のコマンドで、Trunk を Rust にインストールしてください。

▼ リスト 1.4: Trunk のインストール: shell

```
cargo install --locked trunk
```

Trunk は Rust から WASM へのビルドを実行するだけでなく、それを HTML ファイルと結合して WASM が動作する Web ページを生成します。また、PC にローカルなサーバ（localhost）を一時的に構築し、実際のページの動作を確認することもできます。

このあとのサンプルコードで、Trunk の動作時にエラーが生じた場合は公式ページ https://trunkrs.dev/の Getting Started を参照してください。たとえば、Apple M1 の環境では追加のコマンド実行が必要になる場合があります。

第 2 章

Hello, Yew!

2.1 最も簡単な Web アプリ

新しい技術の最初の一歩を踏み出すときには、"Hello, World!"を実行することが慣例です。ここでは、「Hello, Yew!」の文字列を表示するだけの Web アプリを作成し、Github Pages で公開するための基本を学びます。この Web アプリのサンプルは、以下の URL で公開されています。

[外部リンク] Hello, Yew!
https://j-impact.github.io/hello-yew/

文字を表示するだけでも立派な Web アプリです。この Web アプリを作るためのコード一式は、以下のリポジトリにまとめています。

[外部リンク] J-IMPACT/hello-yew
https://github.com/J-IMPACT/hello-yew

2.2 作業の進め方

以降の章では、このリポジトリに含まれるファイルの内容を 1 つずつ解説することで、Github Pages で Web アプリを公開するための最小構成を学習します。実際に作業を行いながら本書を読み進める場合には、以下の 2 通りの方法を推奨します。

ただし、どの方法を選択する場合あっても、Github アカウントが作成済みであり、自身の PC で `git` コマンドが使用可能になっていることを前提とします。`git` コマンドが使用できない場合には、Git の公式サイト https://git-scm.com/を参照してインストールしてください。

♣ 空のプロジェクトを作って、必要なファイルを再現する

Rust で空のプロジェクトを作成し、その中に `J-IMPACT/hello-yew` 内のファイルを作成しながらリポジトリを再現します。コードの写経を通して理解度を高めたい読者におすすめの方法です。コードを写し間違えると Web アプリが動作しなくなることがあるため注意が必要ですが、動作に不具合がある場合のバグ修正も含めて、非常に実践的な経験を積むことができると思います。

この方法で作業を進める場合には、シェルやコマンドプロンプトで Rust プロジェクトを作成したいディレクトリ（フォルダ）に移動し、リスト 2.1 のコマンドで、`hello-yew` という名前の空の Rust プロジェクトを作成します。

▼ リスト 2.1: Rust プロジェクトの新規作成: shell

```
> cargo new hello-yew
```

すると `hello-yew` ディレクトリが作成され、その中に Rust で開発を行うための必要最小限のファイルが生成されます。この `hello-yew` ディレクトリを VS Code などのエディタで開き、各ファイルの内容が `J-IMPACT/hello-yew` と同じになるようにコードを修正してください。

[外部リンク] Visual Studio Code（参考）
https://code.visualstudio.com/

いくつかのファイルは新規に作成する必要があります。
ただし、`Cargo.lock` ファイルは無視し、**複製も削除もしないでください**。このファイルは Rust のビルド時に、そのときのクレートの状況に合わせて自動で生成されますが、Web アプリ全体の動作に影響を与えません。

また、`J-IMPACT/hello-yew` に含まれていないからといって、リスト 2.1 の実行直後から存在する `.git` ディレクトリは**絶対に削除しないでください**。このディレクトリは Git ならびに Github でのプロジェクト管理に極めて重要です。
なお、`.git` ディレクトリは隠しフォルダに設定されているため、PC の設定によっては見えない場合があります。

この方法を採用した場合には、新規作成した `hello-yew` ディレクトリと、読者の Github リポ

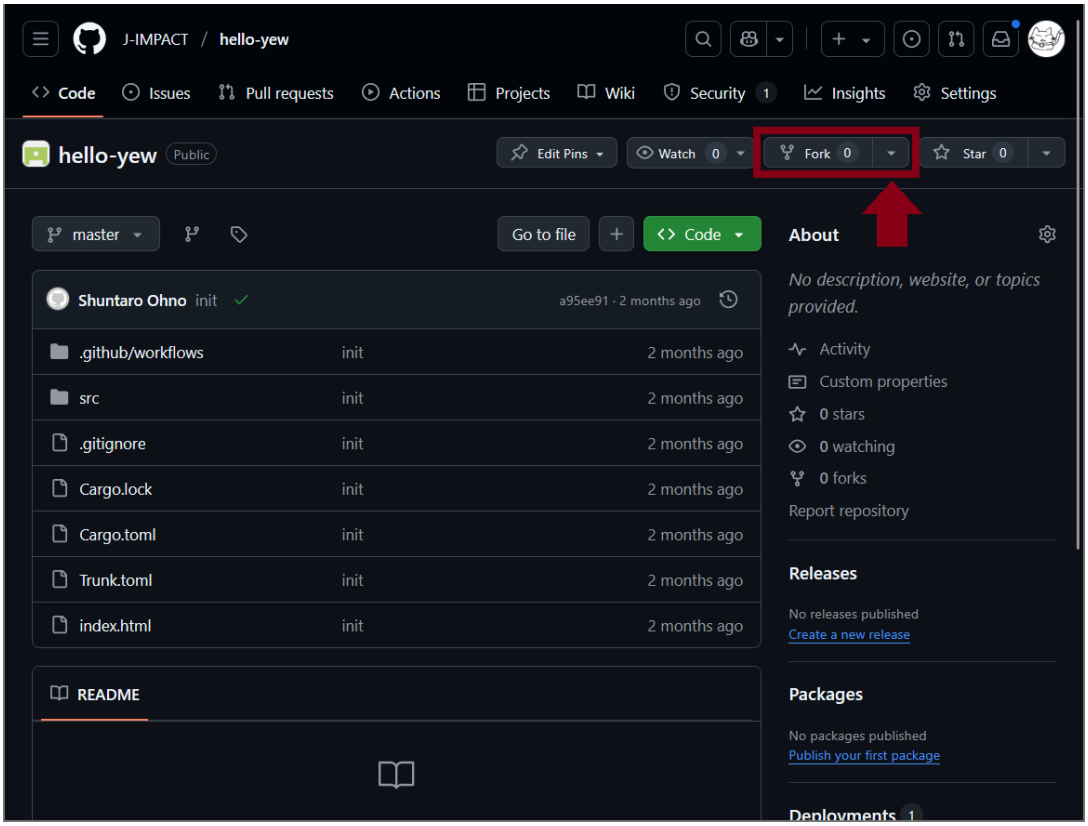
ジトリを結びつける作業が必要です。（作業内容は後ほど解説します）

♣ リポジトリをフォークする

コードを写経することに意味を感じない、またはサッと一通り本書を読み終えてしまいたい読者にはこちらの方法をおすすめします。

Github におけるリポジトリの「**フォーク**」とは、他のユーザが作成したリポジトリを自分のアカウントにコピーすることを意味します。コピー元のリポジトリを「本家」とし、自身のアカウントの管理下に「分家」を作成するイメージです。具体的には、本章の「Hello, Yew!」リポジトリは「J-IMPACT」のアカウントで作成したため `J-IMPACT/hello-yew` という名前になっていますが、これをフォークすることで `[your-user-name]/hello-yew` が作成されます。

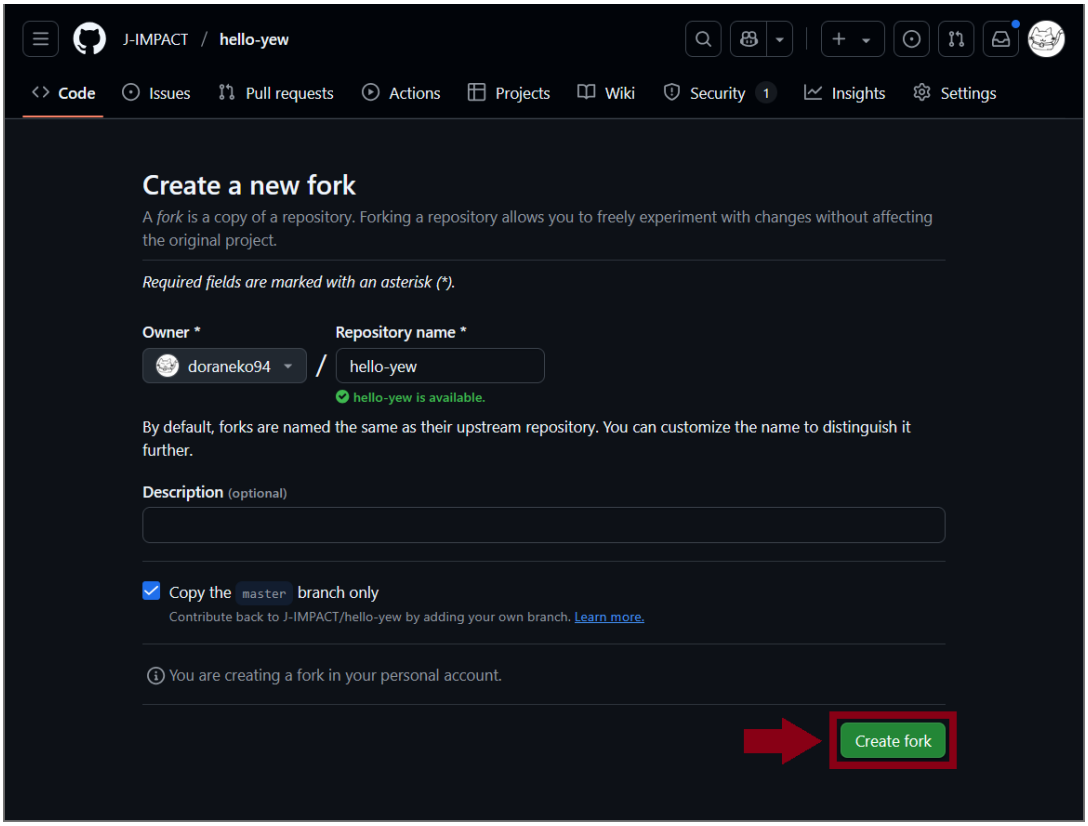
フォークの方法は非常に簡単で、Github にログインした状態で図 2.1 に示した「Fork」ボタンをクリックするだけです（なお、Fork」ボタンをクリックする前に、その右横にある「Star」ボタンも押していただけると著者が泣いて喜びます）。



▲図 2.1:「Fork」ボタンの位置

すると図 2.2 のような画面が表示されるので、「Create fork」ボタンをクリックするとフォークが完了し、あなたのアカウントの管理下に置かれた `[your-user-name]/hello-yew` リポジトリ

のページに遷移します。



▲図 2.2:　「Create fork」ボタンの位置

`J-IMPACT/hello-yew` と `[your-user-name]/hello-yew` の内容は独立しています。よって、分家であるあなたの `hello-yew` の内容を変更しても、本家のリポジトリの内容には影響を与えません。

この `[your-user-name]/hello-yew` を使って作業を行うため、リポジトリの内容を自身の PC にコピーしておきます。シェルやコマンドプロンプトでリポジトリのコピーを作成したいディレクトリ（フォルダ）に移動し、リスト 2.2 のコマンドでリポジトリを**クローン**してください（`[your-user-name]` は、あなたのアカウント名で置き換えてください。カッコ `[]` は削除してください。つまり、この部分はあなたのリポジトリの URL に相当します）。

▼リスト 2.2: リポジトリのクローン: shell

```
> git clone https://github.com/[your-user-name]/hello-yew
```

2.3 Cargo.toml

付録 A
Background

A.1 backsub

式 A.1: EQ

$$a = 0$$

▼表 A.1: TBL

a	b
1	2

キャプションあり

フィボナッチ数列

```
def fib(n)
  return n <= 1 ? n : fib(n-1) + fib(n-2)
end
```

キャプションなし（スペースが広すぎる）

```
def fib(n)
  return n <= 1 ? n : fib(n-1) + fib(n-2)
end
```

▲図 A.1: IMG

あとがき / おわりに

Rust と Github Pages で公開する Web アプリ開発

クラウドにお金を払いたくない人のための開発入門

2025 年 5 月 31 日　ver 1.0 （技術書典 18）

著　者　J-IMPACT

(powered by Re:VIEW Starter)