SecHack365

Zig×WASM による セキュアなWeb開発を支援する Framework

開発駆動コース 仲山ゼミ
山脇 颯太

公式サイト →

背景・動機

- ➔ 安全なWeb開発の難しさ
  - ◆ グローバル環境でのJS / CSSの干渉
  - ◆ 外部通信時の脆弱性対策
- ➔ 安全な技術はあるが欠点もあり普及が課題
  - WebAssembly (Wasm)
    - サンドボックス環境で実行
    - 外部インターフェイスの制限
  - Web Components
    - JS/CSSをComponent内に閉じる
- ➔ 欠点を改善し統合して開発できるフレームワークがあれば普及させれるのでは！？

zframe

- ➔ Web Frontend Framework
- ➔ Wasm / Web Componentsを活用したWeb開発のエコシステム
- ➔ 部分的に導入可能
  - Wasmはzframe、他はReactなど
  - 外部依存なし（zigコンパイラのみでビルド可能）
  - 純粋なファイルを生成
- ➔ 開発支援CLI付き
- ➔ CSR / SGR 対応
- ➔ ドキュメント・公式サイト付き

```
pub fn index() node.Node {
    const handler = JsHandler{
        .then = .{
            .filename = "convert.js",
            .func = "wasm",
        },
    };
    const loader = Loader{
        .webassembly = WebAssembly.init(
            "hash.wasm",
            handler,
        ),
    };
    const clickHandler = EventListener{
        .target = .change,
        .content = loader,
    };

    return node.createNode(.div).init(.{
        node.createNode(.div).init(.{
            node.createNode(.div).init(.{
                h1.init("Analyze WebAssembly!"),
                node.createNode(.div).init(.{
                    node.createNode(.form).init(.{
                        node.createNode(.input)
                            .init(.{
                            .type = .file,
                            .accept = ".wasm",
                        }).addEventListener(clickHandler)
                    }),
                }),
                wasmAnalysisTable(5.{}),
            }),
        }),
    });
}
```

← Webアプリのコード

← JS読み込み Wasm設定 ハンドラ定義

← DOM描画 ハンドラ追加

↓ビルド結果 Wasm解析 Webアプリ Wasmで動作 ↓

Analyze WebAssembly!

Analysis Result

ID	params	results
0	i32,i32,i32,i32	i32
1	i32,i32	i32
2	i32,i32	
3		i32
4	i32,i32,i32	
5	i32,i32,i32	i32
6	i32,i32,i32,i32	
7	i32,i32,i32,i32,i32,i32	i32

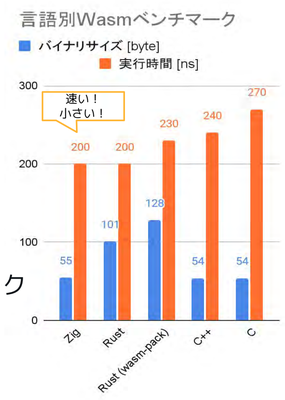
Zig言語

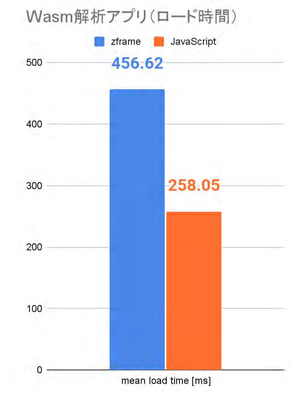
"C but with the problems fixed"
Andrew (Zig作者)

- ➔ ベータ版 (2025/3)
- ➔ Wasm生成に最適
  - ◆ 高速 ∧ 小サイズ
  - ◆ 調査・測定データ →

Yew: Rust製フレームワーク
Wasmを活用する競合
RustはWasmサイズ 大

調査・ベンチマーク





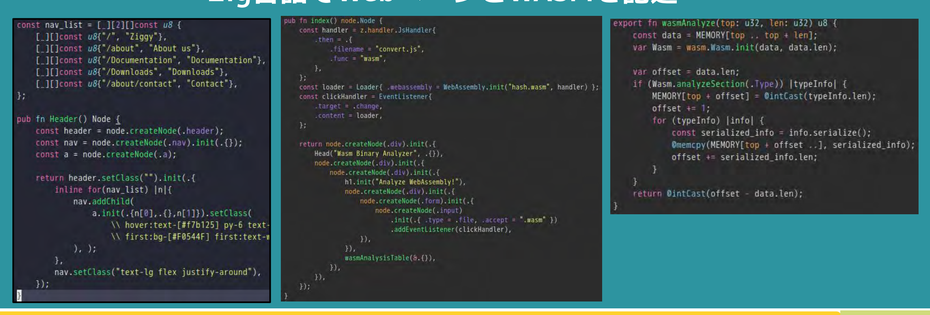
WebAssemblyとは？

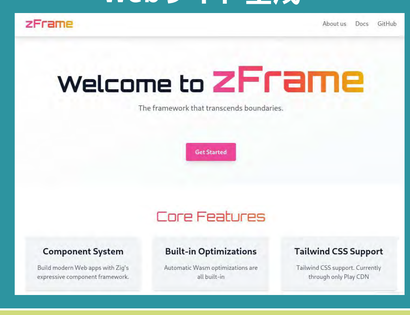
- ✔ 様々な言語で生成されブラウザで高速に動作
- ✔ 既存ソフトウェア資源をWeb開発で活用可能
- ✔ サンドボックスでセキュアに実行可能
- ✔ 外部との通信なしにファイル処理などが可能
- ✗ サイズが大きくロード時間がボトルネック
  - ◆ 2回目以降はキャッシュで改善可能
  - ◆ JavaScriptの方が速いケースもある
- ✗ 開発環境が未整備でハードルが高い
- ✗ Wasm実行のためのJSグルーコードが都度必要

Web Componentsとは？

- ✔ JSでComponentを構築するWeb標準技術
  - フレームワーク非依存 (!)
- ✔ JS/CSSをコンポーネント単位で完全に分離
  - 開発者は自身のコンポーネントに集中
  - グローバル環境汚染の影響なし
- ✗ 登場から早10年、普及していない
  - JSの記述がやや冗長
    - Reactほど洗練されていない
  - ライブラリ/フレームワークほぼなし

プロジェクト生成 → Zig言語でWebページとWASMを記述 → ビルド → Webサイト生成 → 配信





Core Features

コンポーネントシステム

- ➔ メソッドチェーンでDOMを構築
- ➔ 画像・リンクなど組み込みの最適化あり
- ➔ HTML属性などは型付けされている
  ページコンポーネント →

```
pub fn about() node.Node {
    const p = node.createNode(.p);

    return div.init(.{
        title.init("About zframe"),
        p
            .init("under development")
            .setClass("text-xl"),
        p.init("Hold on a second!"),
    });
}
```

Web Components

通常のDOM　Web Components

Markdown　Result

1　3

* { color:red}
通常のDOMに影響なし

グローバルなCSS環境を
コンポーネントに限定

```
const div = node.createNode(.div);
div.init("normal");
```

↑ 通常のDOM定義
↓ Web Componentsの定義

共通のインターフェイス

Web Componentsは
定義から自動生成

ユーザーに差異を隠蔽
気軽に活用できる！

```
const custom = node
    .createNode(.custom)
    .define("web-components");
custom.init("Web Components");
```

Wasmインターフェイス

- ➔ ZigコードをWasmに自動ビルド
- ➔ Wasmバイナリ解析で読み込みに必要な情報を抽出
- ➔ JavaScriptのグルーコードを自動生成

```
WebAssembly.instantiateStreaming(
    fetch('example.wasm'),{env}
).then(test)
```

Wasmをコンパイルするコード

実行可能な関数の情報
メモリの情報
インポートする関数の情報

File System Based Routing

- ➔ src/pagesディレクトリ配下の構造でルーティングを作成
- ➔ ZigとMarkdownを配置するだけでHTMLを自動生成

ディレクトリ構成

```
src/pages
├── Docs
│   └── element.md
├── Docs.md
├── about.zig
└── index.zig
```

組み込み形式のJavaScriptサポート

- ➔ Wasm実行結果を動的に反映させるJSコードをプロジェクトに反映

イベントハンドラなど定義 JS ＋ JS呼び出し/ハンドラ設定 ZIG → ビルド

Markdown対応

- ➔ Markdownで書いたものに共通のレイアウトを適用できる
- ➔ zframeはCMSとしても利用可能

Eco Systems

zframe CLI

- ➔ 開発者体験向上を目的としたツール
- ➔ プロジェクト生成, 自動ビルド, Hot-reload

自動ビルドの様子 →
エラー表示付き

```
BUILD SUCCESS.
既存のブラウザ セッションで開いています。
listening on http://0.0.0.0:3001
press Ctrl-C to quit...
error: Failed to serve client: error.FileNotFound
run
└─ run zframe
   └─ zig build-exe zframe Debug native
src/pages/about.zig:9:12: error: use of undeclared
    return div.init(.{
           ^~~
```

zerver

- ➔ 軽量なZig製プレビュー用Webサーバー
  - ◆ 超軽量Web Socketプロトコル対応（Hot-reload用に最適化）
- ➔ CLI内部で使用

Abelha & markdown-zig

- ➔ nom風パーサコンビネータとMarkdownパーサー
  - ◆ 小さなパーサーを組み合わせて解析
- ➔ MarkdownからHTMLを生成するときに使用

パース例
（カラーコード） →

```
fn hexColor(input: []const u8) !ParseResult([]const u8) {
    const result = try tag("#")(input);
    const res = try separated_list1(
        u8,
        tag(""),
        parseHex,
    )(result.rest);
    return res;
}

test {
    const text = "#1A2B3C";
    const result = try hexColor(text);
    const answer = [_]u8{ 0x1a, 0x2b, 0x3c };
    try std.testing.expectEqualSlices(u8, &answer, result
}
```

VIGIA

- ➔ マルチプラットフォームなファイル監視ライブラリ
  - ◆ ポーリングとinotifyを使用
- ➔ 自動ビルドのための変更検知に使用

変更検知の様子 →

```
var Monitor = try FileMonitor.init(observe_dir);
defer Monitor.deinit();
while (true) {
    if (try Monitor.detectChanges()) {
        const status = execute_command("zig build run")
        if (status == 0) {
            try insertWebSocketConnectionCode(manager);
            try stdout.print("\x1B[1;92mBUILD SUCCESS.\
            try manager.sendData("Reload!");
        }
    }
}
```

Binalyze

- ➔ Wasmバイナリを解析して情報を抽出
- ➔ Wasm最適化、JSグルーコード自動生成で使用

例）バイナリから必要なメモリ量を抽出して最適なJSグルーコードを生成

ドキュメント類

- ➔ 公式サイト・チュートリアル・Examples（日本語・英語・ポル語）

課題・展望

機能の追加・改良

- ➔ Wasm-JS対応のclass生成自動化
- ➔ CSS in Zigをやりたい
- ➔ 動的なページ生成

関連ライブラリの整備

- ➔ UIライブラリ（table, CSS, Chart …）
- ➔ fetchライブラリ（外部データを動的に反映したい）

TOML [Tom's Obvious Minimal Language]

実行速度以外も
競合Yewに勝つ

NICT National Institute of Information and Communications Technology

リポジトリ
https://github.com/yamada031016/zframe

公式サイト
https://yamada031016.github.io/zframe