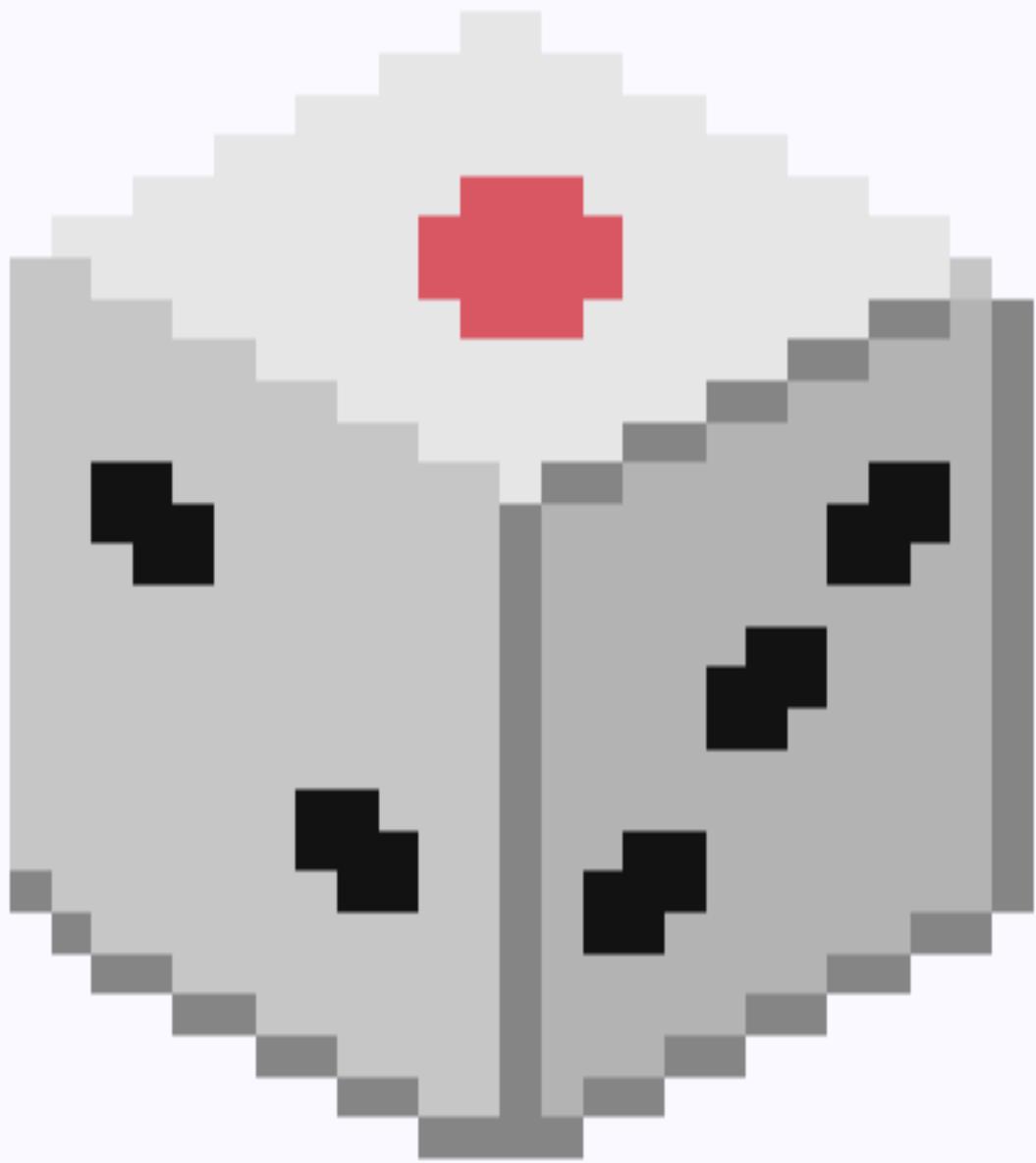


Flutter Web入門

マルチプラットフォーム開発から
WebAssemblyまで

目次

1. Flutterとは
2. Flutter Webとは
3. WebComponentsとFlutter
4. WebAssembly (WASM)とFlutter
5. まとめ



自己紹介

名前: 早坂太吾 / @dicenull

所属: 株式会社jig.jp

Flutterとの関わり: 2020年新卒入社から

Burikaigiとの関わり: 昨年初参加、今年初登壇します！

1. Flutter マルチプラットフォーム

Flutterとは

Google製のUIフレームワーク

言語: Dart

特徴: マルチプラットフォーム対応

1つのプロジェクトでiOS / Android / Web / Desktopに対応

UIの仕組み: すべてが Widget

プラットフォームごとのUI差分なし

パッケージ (pub.dev) が複数のプラットフォームをサポート

Flutterの歴史

2018年: Flutter 1.0 Android / iOS Stable

2021年: Flutter 2.0 Web Stable

2022年: Flutter 3.0 Desktop Stable

2026年現在、主要プラットフォームをカバー

UI = f(state)

$UI = f(state)$

UIは状態 (state) の関数

同じ状態なら、常に同じUIが表示される

状態が変わると、UIが自動的に更新される

予測可能で、デバッグしやすい

宣言的UI (Declarative UI)

「何を」表示するかを記述する

状態が変わると、UIが更新される

```
class _CounterDemo extends StatefulWidget {
  const _CounterDemo();
  @override
  State<_CounterDemo> createState() => _CounterDemoState();
}

class _CounterDemoState extends State<_CounterDemo> {
  int _counter = 0;
  void _incrementCounter() {
    // 状態を更新する
    setState(() {
      _counter++;
    });
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
```

0

+

パッケージ(pub.dev)で拡張できる

便利なパッケージがたくさん

マルチプラットフォームに対応したパッケージが多い

Riverpod × HooksでReactっぽく書ける

状態管理がシンプル

関数型コンポーネント風

React経験者にも馴染みやすい

```
import 'package:flutter/material.dart';
import 'package:flutter_hooks/flutter_hooks.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

// カウンターの状態を管理するProvider
final counterProvider = StateProvider<int>((ref) => 0);

// RiverpodHooksを使ったカウンターアプリ
class RiverpodHooksCounter extends HookConsumerWidget {
  const RiverpodHooksCounter({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    // Providerから状態を取得
    final counter = ref.watch(counterProvider);

    // カウンターを増やす関数
    final increment = () {
      ref.read(counterProvider.notifier).state++;
    };

    return Scaffold(
```

0

+

Flutter Web

マルチプラットフォーム対応なので、Webでも動く

同じコードでiOS / Android / Web / Desktop

プラットフォーム間でUIの一貫性を保てる

他にも、

WebComponentsで既存のJS資産を活用

WebAssembly (WASM) でパフォーマンス向上

Flutter Webのレンダリング方式比較

方式	HTML	CanvasKit	WASM
状態	廃止	現時点で推奨	将来推奨
パフォーマンス	低	高	最高
SEO対応	可	部分的	部分的
起動速度	速い	やや遅い	やや遅い
ファイルサイズ	小さい	大きい (2.4MB)	中程度 (1.9MB)

レンダリング方式の選択

WASM

将来推奨

WASMを試してみて、
問題なければ継続利用

CanvasKit

現時点で推奨

WASMに問題がある場合は
CanvasKitを使用

HTML

廃止

HTMLレンダリングは
廃止されている

Flutter WebとSEO

公式推奨のアプローチ

index.htmlにmetaタグを設定

```
<meta name="description" content="...">  
<meta property="og:title" content="...">
```

別途SEO対応ページを作成、
Flutter Webに誘導

公式ドキュメントより

「Flutter Webは検索エンジンが適切にインデックスするためには必要なものと一致しません。静的またはドキュメントのようなWebコンテンツにはHTMLを使用することを推奨します。また、Flutterで作成したアプリケーション体験と、ランディングページ、マーケティングコンテンツ、ヘルプコンテンツを分離することを推奨します（これらはSEO最適化されたHTMLで作成）。」

<https://docs.flutter.dev/platform-integration/web/faq>

Flutter WebとProgressive Web App (PWA)

デフォルトでPWA対応

何も設定しなくてもPWAとして動作

アプリのようにホーム画面に追加可能

ServiceWorkerキャッシュ戦略

pwa-strategyオプションで変更可能

```
flutter build web --pwa-strategy=offline-first
```

オプション: offline-first, online-first, none

pwa-strategyオプション

offline-first



推奨

オフラインでも動作させたい
場合

online-first

常に最新データが必要な場合

オンライン優先。ネットワー
クを優先的に使用

none

ServiceWorkerが不要な場合

ServiceWorkerを無効化

Flutter Web活用事例

「ふわっち」のモバイルWeb

視聴・配信機能を構築

モバイルアプリのコードを活用

WebComponentsも活用

既存のJS資産をFlutter Webに統合



2. WebComponentsの活用

WebComponents (WC) を使う理由

Flutter Webでできないことを補完する

Flutter側でまだサポートされていないブラウザ固有の技術を使いたい

既存のJS資産（チャットUI、特殊なプレイヤー等）を再利用したい

WCの作り方:

ピュアなHTML/CSS/JSで構築

Angularなどのフレームワークから書き出し

Web Componentsとの連携

Flutter WebでWeb Componentsを埋め込む方法

手順

1. PlatformViewRegistryでWeb Componentを登録
2. HtmlElementViewでWidget Tree内に配置
3. イベントやプロパティで双方向通信

メリット

- FlutterのWidget Tree内に配置可能
- 既存のJS資産を再利用

1. PlatformViewRegistryで登録

Web ComponentをFlutterに登録

```
import 'package:web/web.dart' as web;
import 'package:flutter_web_plugins/
flutter_web_plugins.dart';

void registerWebComponent() {
  PlatformViewRegistry.registerViewFactory(
    'my-web-component',
    (int viewId) {
      final element = web.document.createElement('my-
component');
      element.setAttribute('data-value', '123');
      return element;
  },
);
```

ポイント

- 文字列でIDを指定
- web.documentでDOM操作
- プロパティやイベントリスナーも設定可

2. HtmlElementViewで表示

Widget Tree内に配置

```
import 'package:flutter/material.dart';

class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: SizedBox(
          width: 300,
          height: 200,
          child: HtmlElementView(
            viewType: 'my-web-component',
          ),
        ),
      ),
    );
  }
}
```

ポイント

- 文字列のIDで配置
- 通常のWidgetのように配置
- HTMLでWCを埋め込む

⚠HTMLの埋め込みは高
価な処理

3. 双方向通信

```
import 'package:web/web.dart' as web;
import 'package:flutter_web_plugins/flutter_web_plugins.dart';
import 'package:flutter/material.dart';

// 1. Web Componentを登録
void registerWebComponent() {
  PlatformViewRegistry.registerViewFactory(
    'communication-demo',
    (int viewId) {
      final container = web.HTMLDivElement()
        ..style.display = 'flex'
        ..style.flexDirection = 'column';

      // 表示エリア
      final display = web.HTMLDivElement()
        ..id = 'display'
```



詳細はZennの記事をご覧ください

🤝 Flutter Webからウェブコンポーネントを使う

https://zenn.dev/jigjp_engineer/articles/cc7fbc31d045d2

記事の内容

- Web Componentsの基本概念
- Flutter Webでの実装方法
- 双方向通信の実装例

3. WebAssembly (WASM) の話

WASM対応で変わるレンダリング

Flutter 3.22 以降でWASMビルドが正式対応！

従来のJavaScriptビルド（CanvasKit/SK Wasm）との違い

パフォーマンス向上: より滑らかなアニメーション

バイナリサイズ: 効率的な実行

WASMビルドを試す

ビルド

```
flutter build web --wasm
```

ビルド後のファイル

- main.dart.js: Dartコード (JavaScript)
- main.dart.wasm: Dartコード (WebAssembly)
- flutter.js: Flutterフレームワーク

WASM判定フロー

flutter.js

WASMサポート判定

dart2wasmコンパイル、WasmGCを確認



WASMをサポートしているか



TRUE



FALSE

main.dart.wasm

WASMビルドを読み込み

約1.9MB



main.dart.js

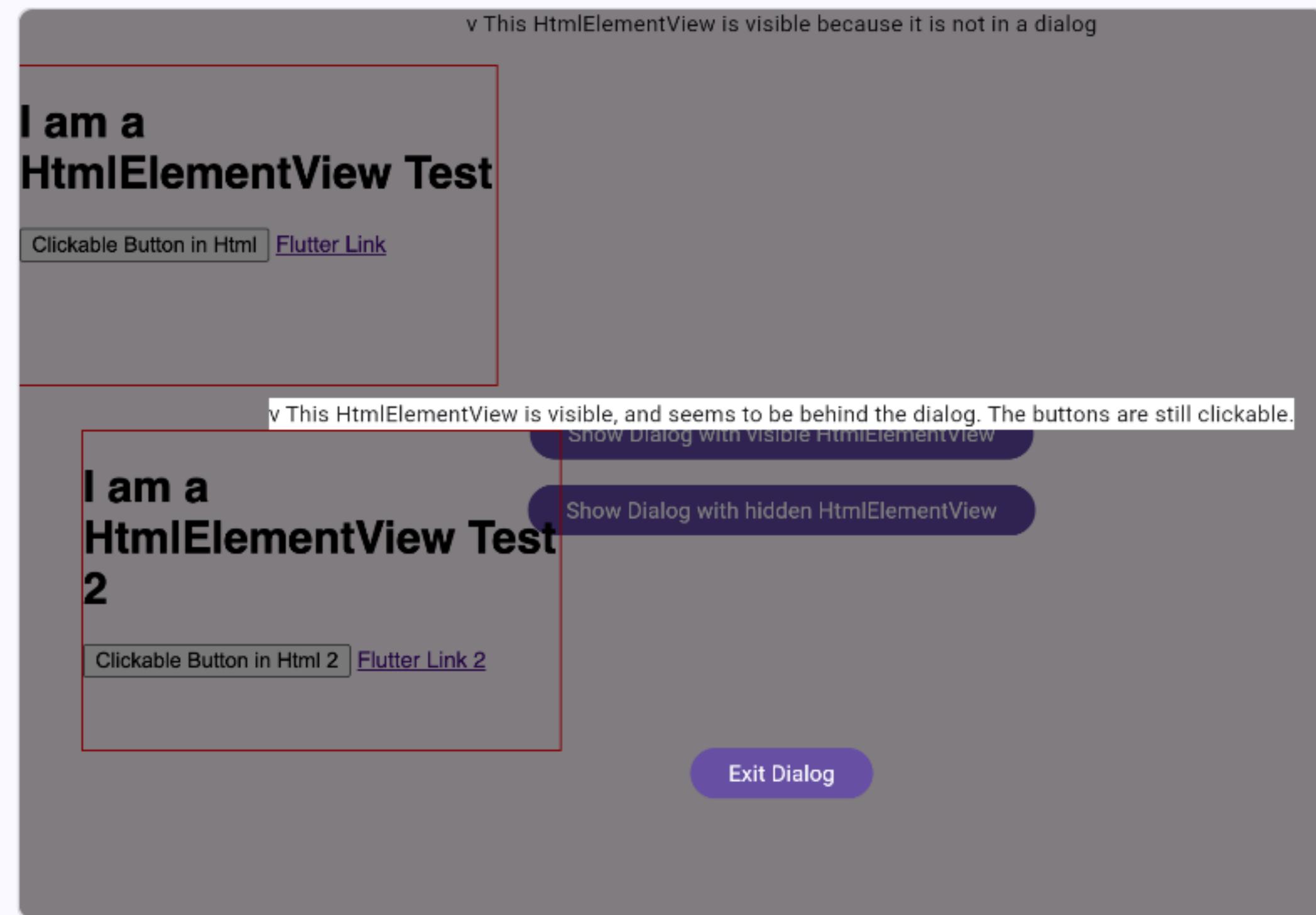
JSビルドを読み込み

約2.4MB

WASMとWebComponentsの併用時の注意

WebComponentsがWidgetの後ろに隠れてしまう

- <https://github.com/flutter/flutter/issues/166357>



学んだことのまとめ

Flutter Web

- マルチプラットフォーム対応でWebでも動く
- 同じコードでiOS / Android / Web / Desktop

WebComponents

- Flutter Webでできないことを補完
- 既存のJS資産を再利用可能

WebAssembly (WASM)

- Flutter 3.22以降で正式対応 WebComponentsと併用は注意



このスライドも
Flutter Webで作りました

flutter_deck パッケージを使用

pub.dev/packages/flutter_deck

ソースコード

github.com/dicenull/flutterweb_burikaigi2026