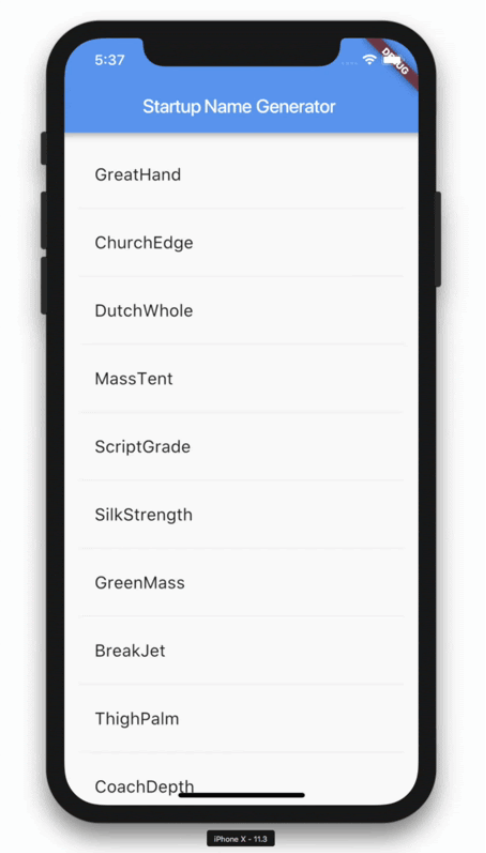
最初のFlutterアプリを書く、パート1

目次



これは、最初のFlutterアプリを作成するためのガイドです。オブジェクト指向コードと、変数、ループ、条件などの基本的なプログラミングの概念に精通している場合は、このチュートリアルを完了することができます。Dart、モバイル、またはWebプログラミングの経験は必要ありません。

このコードラボは、2部構成のコードラボのパート1です。Google Developers Codelabsでパート2を見つけることができます（このコードラボのパート1をコピーすることもできます）

パート1で作成するもの

スタートアップ企業の提案された名前を生成する簡単なアプリを実装します。ユーザーは名前を選択および選択解除して、最適な名前を保存できます。このコードは、一度に10個の名前を遅延生成します。ユーザーがスクロールすると、さらに多くの名前が生成されます。ユーザーがスクロールできる距離に制限はありません。

アニメーションGIFは、パート1の完了時にアプリがどのように機能するかを示しています。

パート1で学ぶこと

- iOS、Android、およびWebで自然に見えるFlutterアプリを作成する方法
- Flutterアプリの基本構造
- 機能を拡張するためのパッケージの検索と使用
- より迅速な開発サイクルのためのホットリロードの使用
- ステートフルウィジェットを実装する方法
- 無限にロードされたリストを作成する方法

このコードラボのパート2で、あなたは、インタラクティビティを追加し、アプリのテーマを変更し、新しい画面に移動する機能を追加します（Flutterでrouteと呼ばれる）

使用するもの

このlab:を完了するには、FlutterSDKとエディターの2つのソフトウェアが必要 です。このコードラボはAndroidStudioを想定していますが、お好みのエディターを使用できます。

このコードラボは、次のデバイスのいずれかを使用して実行できます。

- コンピューターに接続され、開発者モードに設定されている物理デバイス（AndroidまたはiOS）
- iOSのシミュレータ（Xcodeのツールをインストールする必要があります）
- Androidのエミュレータは（AndroidStudioのセットアップが必要です）
- ブラウザ（デバッグにはChromeが必要です）

アプリをコンパイルしてWebで実行する場合は、この機能を有効にする必要があります（現在ベータ版です）。Webサポートを有効にするには、次の手順を使用します。

```
$ flutter channel beta
$ flutter upgrade
$ flutter config --enable-web
```

configコマンドを実行する必要があるのは1回だけです。Webサポートを有効にすると、作成するすべてのFlutterアプリもWeb用にコンパイルされます。IDEのデバイスプルダウンの下、またはを使用したコマンドラインで、Chrome とWebサーバーが一覧表示されているflutter devicesはずです。クロームデバイスは自動的にChromeを起動します。Webサーバは、 ホストアプリはあなたが任意のブラウザからそれをロードできるようにすることを、サーバーを起動します。開発中にChromeデバイスを使用して、DevToolsを使用できるようにし、他のブラウザーでテストする場合はWebサーバーを使用します。詳細については、「Flutter を使用したWebアプリケーションの構築」および「Web上で最初のFlutterアプリを作成する」を参照してください。

ステップ1：初めてのFlutterアプリを作成する

初めてのFlutterアプリ入門の手順を使用して、シンプルなテンプレート化されたFlutterアプリを作成 します。プロジェクト名はflutter_appの代わりにstartup_namerという名前を付けます。

ヒント： IDEのオプションとして「新しいFlutterプロジェクト」が表示されない場合は、FlutterとDart用のプラグインがインストールされているかを確認してください。

基本的に、Dartコードが存在するlib /main.dartを編集します。

1. lib/main.dartの内容を置き換えます。

   lib/main.dartからすべてのコードを削除します。画面中央に「HelloWorld」と表示される次のコードに置き換えます。

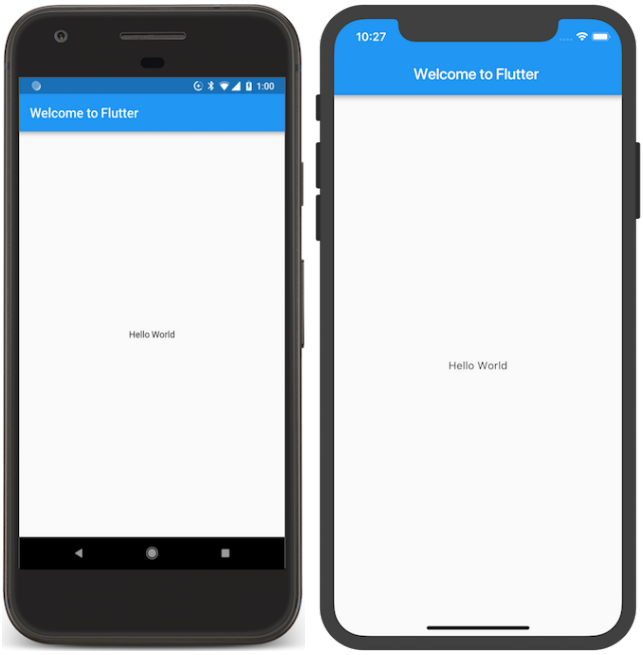
lib / main.dart

```
// Copyright 2018 The Flutter team. All rights reserved.
// Use of this source code is governed by a BSD-style license
that can be
// found in the LICENSE file.

import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

ヒント： アプリにコードを貼り付けると、インデントが歪む可能性があります。これは、次のFlutterツールを使用して修正できます。

- AndroidStudioとIntelliJ IDEA：コードを右クリックし、[dartfmtでコードを再フォーマット]を選択します。
- VS Code：右クリックして[ドキュメントのフォーマット]を選択します。
- ターミナル：flutter format <filename> を実行します。

2. IDEで説明されている方法でアプリを実行します。デバイスに応じて、Android、iOS、またはWeb出力のいずれかが表示されます。



> ヒント： 物理デバイスで初めて実行するときは、ロードに時間がかかることがあります。その後、ホットリロードを使用してすばやく更新できます。 アプリが実行されている場合、保存はホットリロードも実行します。を使用してコンソールから直接アプリを実行する場合は flutter run、Enter　rキーを押してホットリロードを実行します。

観察

- この例では、マテリアルアプリを作成します。 Materialは、モバイルとWebで標準となっているビジュアルデザイン言語です。Flutterは、豊富なマテリアルウィジェットのセットを提供します。ファイルのセクションに uses-material-design: trueエントリを含めることをお勧めします。これにより、事前定義されたアイコンのセットなど、マテリアルのより多くの機能を使用できるようになります。flutterpubspec.yaml
- このmain()方法では、矢印（=>）表記を使用します。1行の関数またはメソッドには矢印表記を使用します。
- アプリは拡張されStatelessWidget、アプリ自体がウィジェットになります。Flutterでは、配置、パディング、レイアウトなど、ほとんどすべてがウィジェットです。
- Scaffoldマテリアルライブラリのウィジェットは、デフォルトのアプリバーと、ホーム画面のウィジェットツリーを保持するbodyプロパティを提供します。ウィジェットのサブツリーは非常に複雑になる可能性があります。
- ウィジェットの主な仕事はbuild()、他の下位レベルのウィジェットの観点からウィジェットを表示する方法を説明するメソッドを提供することです。
- この例の本体はCenter、Text子ウィジェットを含むウィジェットで構成されています。Centerウィジェットは、ウィジェットサブツリーを画面の中央に揃えます。

ステップ2：外部パッケージを使用する

このステップでは、english_wordsというオープンソースパッケージの使用を開始します。このパッケージに は、最もよく使用される数千の英語の単語といくつかのユーティリティ関数が含まれています。

このenglish_wordsパッケージ、および他の多くのオープンソースパッケージは、pub.devにあります。

1. このpubspec.yamlファイルは、Flutterアプリのアセットと依存関係を管理します。pubspec.yamlの依存関係リストにenglish_words（3.1.5以降）を追加します。

{step1_base → step2_use_package}/pubspec.yaml

```
@@ -8,4 +8,5 @@
  dependencies:
    flutter:
      sdk: flutter
    cupertino_icons: ^0.1.2
+   english_words: ^3.1.5
```

2. そのままAndroid Studioのエディタビューでpubspec.yamlを表示している状態で、上部の”Pub get”をクリックします。これにより、パッケージがプロジェクトに取り込まれます。コンソールに次のように表示されます。

```
$ flutter pub get
Running "flutter pub get" in startup_namer...
Process finished with exit code 0
```

Pub getを実行し、また、プロジェクトにプルされたすべてのパッケージとそのバージョン番号のリストを含むpubspec.lockファイルを自動生成します。

3. lib/main.dartに、新しいパッケージをインポートします。

lib/main.dart

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

入力すると、AndroidStudioからライブラリをインポートするための提案が表示されます。インポート文字列が灰色表示され、インポートされたライブラリが（現在のところ）使用されていないことを示します。

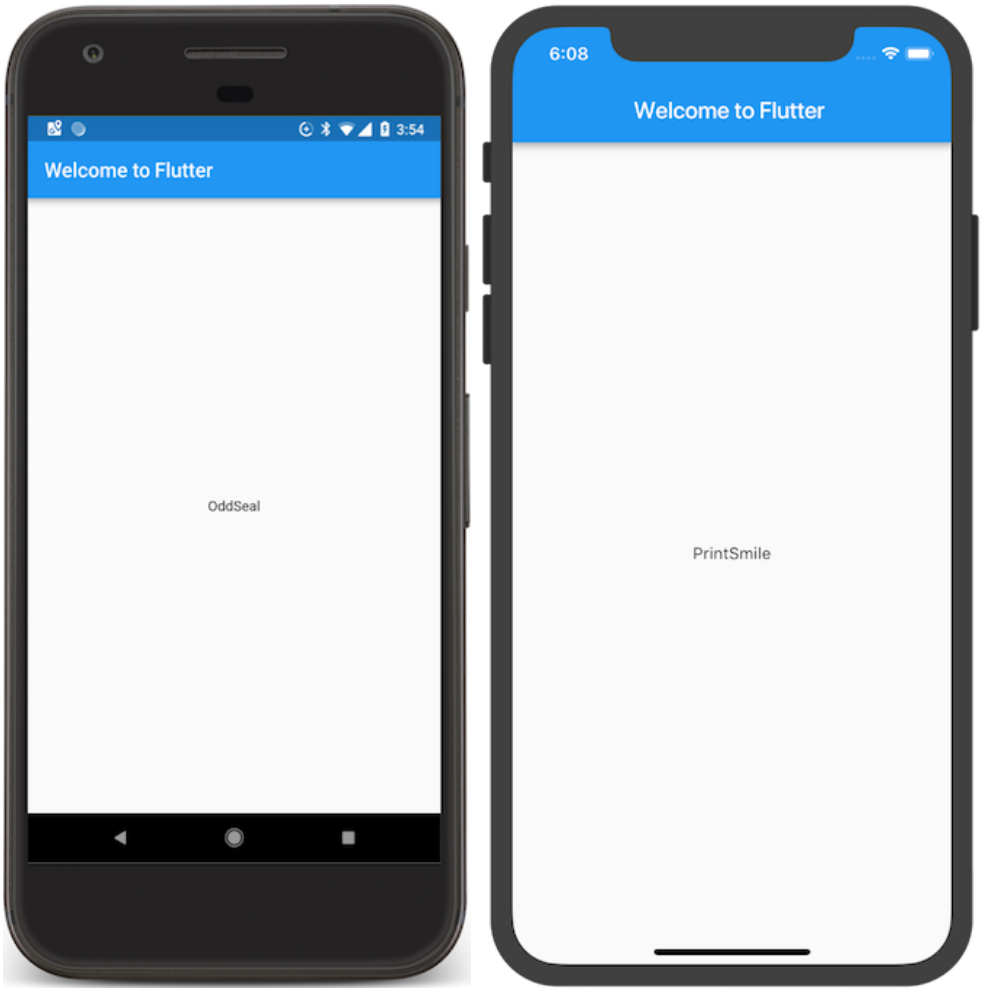
4. “HelloWorld”の代わりに、英語の単語パッケージを使用してテキストを生成します。

{step1_base→step2_use_package}/lib/main.dart

```
@@ -9,6 +10,7 @@
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
+   final wordPair = WordPair.random();
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
@@ -16,7 +18,7 @@
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
-         child: Text('Hello World'),
+         child: Text(wordPair.asPascalCase),
        ),
      ),
    );
```

注：“Pascal case”（アッパーキャメルケースとも呼ばれます）は、最初の単語を含む文字列内の各単語が大文字で始まることを意味します。したがって、”uppercamelcase”という文字列は”UpperCamelCase”と表示されます。

5. アプリが実行中の場合は、ホットリロードで実行中のアプリを更新します。ホットリロードをクリックするかプロジェクトを保存するたびに、実行中のアプリにランダムに選択された異なる2つの英単語が表示されます。これは、単語のペアがビルドメソッド内で生成されるためです。ビルドメソッドは、MaterialAppがレンダリングを必要とするたびに、またはFlutter Inspectorでプラットフォームを切り替えるときに実行されます。



問題？

アプリが正しく実行されていない場合は、タイプミスを探してください。Flutterのデバッグツールのいくつかを試してみたい場合は、デバッグおよびプロファイリングツールのDevToolsスイートを確認してください。必要に応じて、次のリンクのコードを使用して、このチュートリアルに戻ります。

- pubspec.yaml
- lib/main.dart

ステップ3：ステートフルウィジェットを追加する

ステートレスウィジェットのプロパティは変更できません。すべての値がfinalです。

ステートフルウィジェットは、ウィジェットの存続期間中に変更される可能性がある状態を維持します。ステートフルウィジェットを実装するには、少なくとも2つのクラスが必要です。※1 Stateクラスのインスタンスを作成するStatefulWidgetクラス。※2 StatefulWidgetクラスは不変であり、破棄して再生成することができますが、Stateクラスはウィジェットの存続期間にわたって存続します。

このステップでは、ステートフルウィジェットRandomWordsを追加します。これにより、Stateクラス_RandomWordsStateが作成されます。次に、既存のMyAppステートレスウィジェット内でRandomWordsを子として使用します。

1. ステートフルウィジェットの定型コードを作成します。

   lib/main.dartで、すべてのコードの後にカーソルを置き、Returnキーを数回押して、新しい行から開始します。IDEで、”stful”と入力し、Returnキーを押すと、2つのクラスの定型コードが表示され、ステートフルウィジェットの名前を入力するためのカーソルが配置されます。

2. ウィジェットの名前として”RandomWords”を入力します。
   RandomWordsウィジェットは、Stateクラスを作成する以外にほとんど何もしません。

   ステートフルウィジェットの名前として”RandomWords”を入力すると、IDEは付随するStateクラスを自動的に更新し、_RandomWordsStateという名前を付けます。
   デフォルトでは、クラスの名前の前にアンダーバーが付いています。
   識別子の前にアンダースコアを付けると、Dart言語でプライバシーが強化され、オブジェクトのベストプラクティスとして推奨されます。

   IDEはまた、State<RandomWords>を拡張するためにstateクラスを自動的に更新します。これは、 RandomWordsでの使用に特化したジェネリックステートクラスを使用していることを示します。
   アプリのロジックのほとんどはここにあり、RandomWordsウィジェットの状態を維持します。このクラスは、生成された単語ペアのリストを保存します。これは、ユーザーがスクロールすると無限に大きくなり、このラボのパート2では、ユーザーがハートのアイコンを切り替えてリストに追加またはリストから削除すると、お気に入りの単語ペアになります。

両方のクラスは次のようになります。

```
class RandomWords extends StatefulWidget {
  @override
  _RandomWordsState createState() => _RandomWordsState();
}

class _RandomWordsState extends State<RandomWords> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

3. _RandomWordsStateのbuild()メソッドを更新します。

lib/main.dart(_RandomWordsState)

```
class _RandomWordsState extends State<RandomWords> {
    @override
    Widget build(BuildContext context) {
     final wordPair = WordPair.random();
     return Text(wordPair.asPascalCase);
    }
  }
```

4. 下の差分に示す変更を加えて、MyAppから単語生成コードを削除します。

{step2_use_package→step3_stateful_widget} /lib/main.dart

```
@@ -10,7 +10,6 @@
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
-   final wordPair = WordPair.random();
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
@@ -18,8 +17,8 @@
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
-         child: Text(wordPair.asPascalCase),
+         child: RandomWords(),
        ),
      ),
    );
  }
```

5. アプリを再起動します。アプリは以前と同じように動作し、アプリをホットリロードまたは保存するたびに単語のペアが表示されます。

> ヒント： ホットリロードでアプリの再起動が必要になる可能性があるという警告が表示された場合は、アプリを再起動してください。警告は誤検知である可能性がありますが、アプリを再起動すると、変更がアプリのUIに確実に反映されます。

問題？

アプリが正しく実行されていない場合は、タイプミスを探してください。Flutterのデバッグツールのいくつかを試してみたい場合は、デバッグおよびプロファイリングツールのDevToolsスイートを確認してください。必要に応じて、次のリンクのコードを使用して、このチュートリアルに戻ります。

- lib/main.dart

ステップ4：無限にスクロールするListViewを作成する

このステップでは、_RandomWordsStateを展開して、単語の組み合わせのリストを生成して表示します。
ユーザーがスクロールすると、リスト（ListViewウィジェットに表示される）は無限に大きくなります。
ListViewの ビルダーファクトリコンストラクタを使用すると、オンデマンドでリストビューを遅延して作成できます。

1. 提案された単語の組み合わせを保存するために、_suggestionsリストを_RandomWordsStateクラスに追加します。また、フォントサイズを大きくするための_biggerFont変数を追加します。

   lib/main.dart

   ```
   class _RandomWordsState extends State<RandomWords> {
     final _suggestions = <WordPair>[];
     final _biggerFont = TextStyle(fontSize: 18.0);
     // ···
   }
   ```

   次に、_RandomWordsStateクラスに_buildSuggestions()関数を追加します。このメソッドは、提案された単語のペアを表示するListViewを構築します。

   このListViewクラスは、匿名関数として指定されたファクトリビルダーおよびコールバック関数であるビルダープロパティitemBuilderを提供します。
   関数にはBuildContextと行イテレータの2つのパラメータが渡されます。
   イテレータは0から始まり、関数が呼び出されるたびに足されます。提案された単語のペアごとに2回インクリメントします。1回はリストアイテム用、もう1回は仕切り線用です。
   このモデルにより、ユーザーがスクロールしても、提案されたリストが増え続けることができます。

2. _RandomWordsStateクラスに_buildSuggestions()関数を追加します。

   lib/main.dart（_buildSuggestions）

   ```
   Widget _buildSuggestions() {
     return ListView.builder(
         padding: EdgeInsets.all(16.0),
         itemBuilder: /*1*/ (context, i) {
           if (i.isOdd) return Divider(); /*2*/

           final index = i ~/ 2; /*3*/
           if (index >= _suggestions.length) {
             _suggestions.addAll(generateWordPairs().take(10));/*4*/
           }
           return _buildRow(_suggestions[index]);
         });
   }
   ```

/*1*/ itemBuilderコールバックは、提案された単語ペアごとに1回呼び出され、単語をリストアイテムに配置します。偶数行の場合、関数は単語ペアのリストアイテムを追加します。奇数行の場合、この関数は仕切りウィジェットを追加してエントリを視覚的に分離します。小さいデバイスでは、仕切りが見えにくい場合があります。

/*2*/ ListViewの各行の前に、高さ1ピクセルの仕切りウィジェットを追加します。

/*3*/ “i ~/ 2”はiを2で除算し、商を返します。
例：iが1なら0、2,3なら1、4,5なら2になります。これにより、ListViewの単語ペアの数から仕切りウィジェットを差し引いた数が算出されます。

/*4*/ 利用可能な単語の組み合わせの最後に達した場合は、さらに10個を生成して、提案リストに追加します。

_buildSuggestions()関数は、単語ペアごとに_buildRow()を1回呼び出します。この関数は、リストアイテムに新しいペアをそれぞれに表示します。これにより、次のステップで行をより魅力的にすることができます。

3. _RandomWordsStateに_buildRow()関数を追加します。

lib/main.dart（_buildRow）

```
Widget _buildRow(WordPair pair) {
  return ListTile(
    title: Text(
      pair.asPascalCase,
      style: _biggerFont,
    ),
  );
}
```

4. _RandomWordsStateクラスでは、単語生成ライブラリを直接呼び出すのではなく、_buildSuggestions()を使用するようにbuild()にメソッドを更新します。（Scaffoldは、基本的なマテリアルデザインのビジュアルレイアウトを実装しています。）メソッド本体を強調表示されたコードに置き換えます。

lib/main.dart (build)

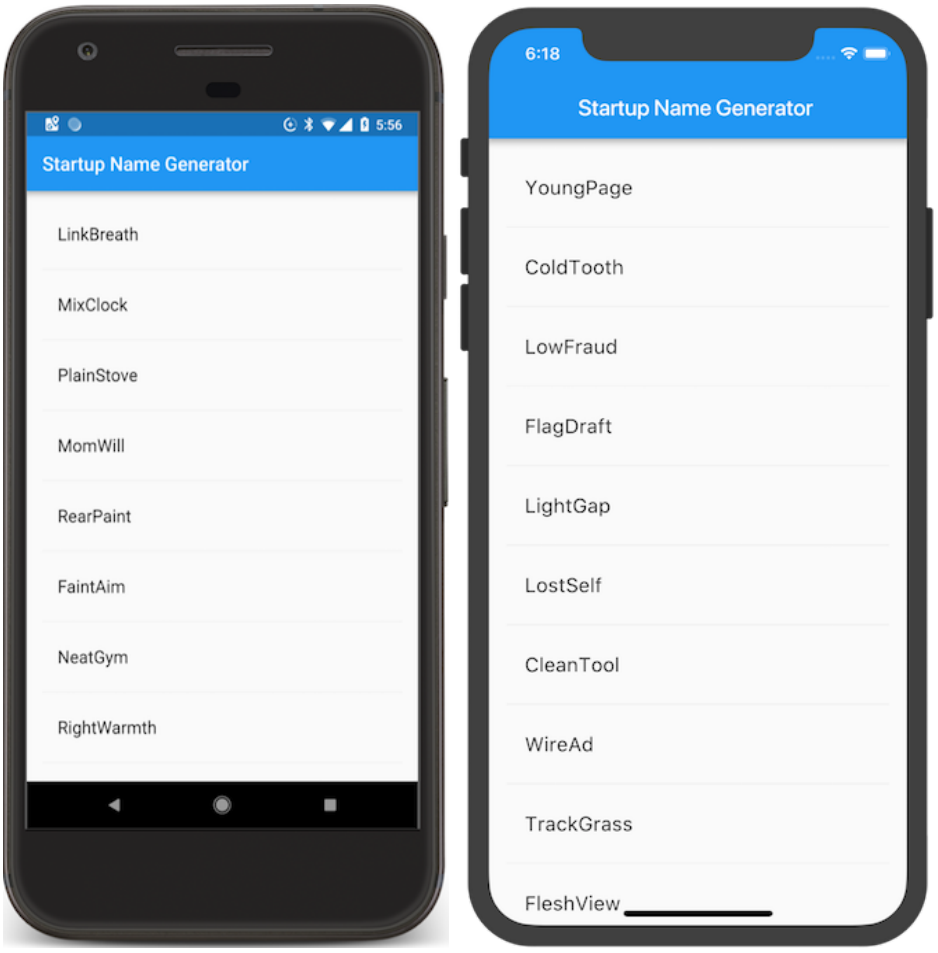
```
@override
Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
     title: Text('Startup Name Generator'),
   ),
   body: _buildSuggestions(),
  );
}
```

5. MyAppクラスで、タイトルを変更し、ホームをRandomWordsウィジェットに変更して、build()メソッドを更新します。

{step3_stateful_widget→step4_infinite_list} /lib/main.dart

```
@@ -10,15 +10,8 @@
  class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
      return MaterialApp(
-       title: 'Welcome to Flutter',
-       home: Scaffold(
+       title: 'Startup Name Generator',
+       home: RandomWords(),
-         appBar: AppBar(
-           title: Text('Welcome to Flutter'),
-         ),
-         body: Center(
-           child: RandomWords(),
-         ),
-       ),
      );
    }
```

6. アプリを再起動します。どこまでスクロールしても、単語の組み合わせのリストが表示されます。



問題？

アプリが正しく実行されていない場合は、タイプミスを探してください。Flutterのデバッグツールのいくつかを試してみたい場合は、デバッグおよびプロファイリングツールのDevToolsスイートを確認してください。必要に応じて、次のリンクのコードを使用して、このチュートリアルに戻ります。

- lib/main.dart

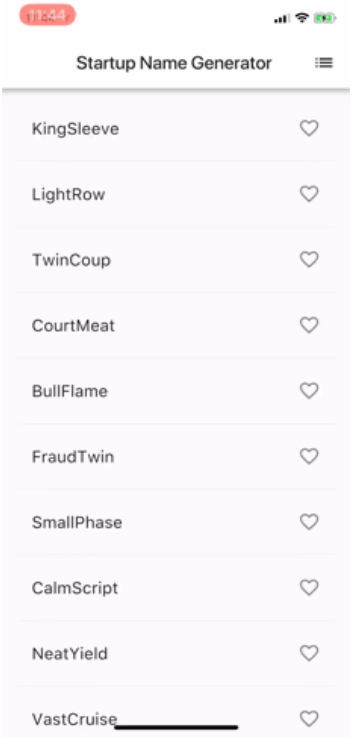
プロファイルまたはリリースの実行

> 重要：デバッグとホットリロードを有効にしてアプリのパフォーマンスをテストしないでください。

ここまで、アプリをデバッグモードで実行してきました。
デバッグモードは、ホットリロードやステップデバッグなどの便利な開発者機能とと引き換えにパフォーマンスを犠牲にします。
ですので、デバッグモードでパフォーマンスが低下し、アニメーションがぎくしゃくするのは当然のことです。
パフォーマンスを分析したり、アプリをリリースする準備ができたら、Flutterの「プロファイル」または「リリース」ビルドモードを使用することをお勧めします。詳細については、 Flutter's build modesを参照してください。

> 重要： アプリのパッケージサイズが気になる場合は、Measuring your app's sizeをご覧ください。

次のステップ



おめでとう！

iOSとAndroidの両方で実行されるインタラクティブなFlutterアプリを作成しました。このコードラボでは、次のことを行います。

- Flutterアプリをゼロから作成しました。
- Dartコードで書きました。
- 外部のサードパーティライブラリを活用しました。
- 開発サイクルを高速化するためにホットリロードを使用しました。
- ステートフルウィジェットを実装しました。
- 遅延ロードされた無限スクロールリストを作成しました。

このアプリを拡張する場合は、Google Developers Codelabsサイトのpart 2に 進み、次の機能を追加します。

- クリック可能なハートアイコンを追加してインタラクティブ機能を実装し、お気に入りのペアリングを保存します。
- 保存したお気に入りを含む新しい画面を追加して、新しいルートへのナビゲーションを実装します。
- テーマの色を変更して、真っ白なアプリを作成します。