

这是创建您的第一个Flutter应用程序的指南。如果您熟悉面向对象和基本编程概念（如变量、循环和条件控制），则可以完成本教程，您无需了解Dart或拥有移动开发的经验。

- [第1步: 创建 Flutter app](#)
- [第2步: 使用外部包\(package\)](#)
- [第3步: 添加一个 有状态的部件 \(Stateful widget \)](#)
- [第4步: 创建一个无限滚动ListView](#)
- [第5步: 添加交互](#)
- [第6步: 导航到新页面](#)
- [第7步：使用主题更改UI](#)
- [做的好!](#)

你将会构建什么？

您将完成一个简单的移动应用程序，功能是：为一个创业公司生成建议的名称。用户可以选择和取消选择的名称、保存（收藏）喜欢的名称。该代码一次生成十个名称，当用户滚动时，会生成一新批名称。用户可以点击导航栏右边的列表图标，以打开到仅列出收藏名称的新页面。

这个 GIF 图展示了最终实现的效果

你会学到什么:

- Flutter应用程序的基本结构.
- 查找和使用packages来扩展功能.
- 使用热重载加快开发周期.
- 如何实现有状态的widget.
- 如何创建一个无限的、延迟加载的列表.
- 如何创建并导航到第二个页面.
- 如何使用主题更改应用程序的外观.

你会用到什么？

您需要安装以下内容:

- Flutter SDK

Flutter SDK包括Flutter的引擎、框架、widgets、工具和Dart SDK。此示例需要v0.1.4或更高版本

- Android Studio IDE

此示例使用的是Android Studio IDE，但您可以使用其他IDE，或者从命令行运行

- Plugin for your IDE

你必须为您的IDE单独安装Flutter 和 Dart插件，我们也提供了 [VS Code](#) 和 [IntelliJ](#) 的插件。

有关如何设置环境的信息，请参阅[Flutter 安装和设置](#)

第1步：创建 Flutter app

创建一个简单的、基于模板的Flutter应用程序，按照[创建您的第一个Flutter应用](#)中的指南的步骤，然后将项目命名为startup_namer（而不是myapp），接下来你将会修改这个应用来完成最终的APP。

在这个示例中，你将主要编辑Dart代码所在的 **lib/main.dart** 文件，

提示：将代码粘贴到应用中时，缩进可能会变形。您可以使用Flutter工具自动修复此问题：

- Android Studio / IntelliJ IDEA: 右键单击Dart代码，然后选择 **Reformat Code with dartfmt**.
- VS Code: 右键单击并选择 **Format Document**.
- Terminal: 运行 `flutter format <filename>`.

1. 替换 lib/main.dart.

删除lib / main.dart中的所有代码，然后替换为下面的代码，它将在屏幕的中心显示“Hello World”。

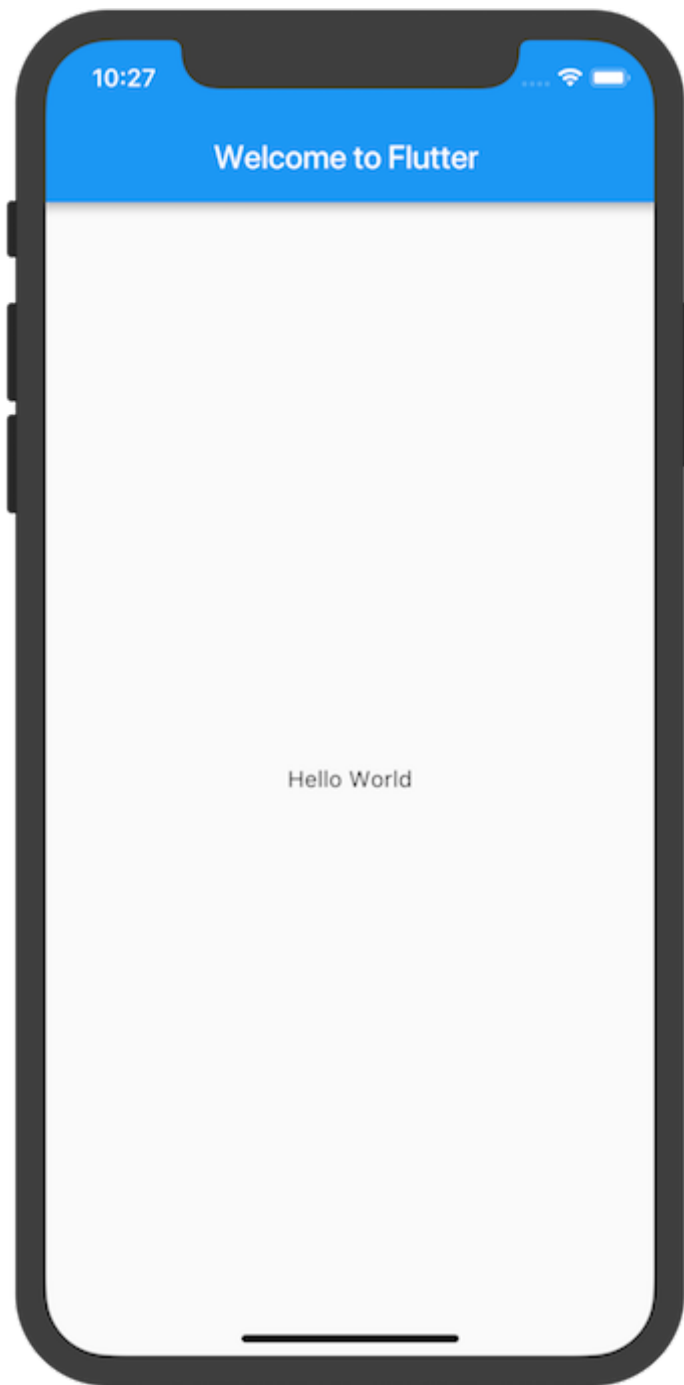
```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
```

```
        title: 'Welcome to Flutter',  
        home: new Scaffold(  
          appBar: new AppBar(  
            title: new Text('Welcome to Flutter'),  
          ),  
          body: new Center(  
            child: new Text('Hello World'),  
          ),  
        ),  
      );  
    }  
  }
```

2. 运行应用程序，你应该看到如下界面.



screenshot of hello world app

分析

- 本示例创建一个Material APP。[Material](#)是一种标准的移动端和web端的视觉设计语言。Flutter提供了一套丰富的Material widgets。
- main函数使用了(\Rightarrow)符号, 这是Dart中单行函数或方法的简写。
- 该应用程序继承了 StatelessWidget, 这将会使应用本身也成为 widget。在Flutter中, 大多数东西都是widget, 包括对齐(alignment)、填充(padding)和布局(layout)

- Scaffold 是 Material library 中提供的一个widget, 它提供了默认的导航栏、标题和包含主屏幕widget树的body属性。widget树可以很复杂。
- widget的主要工作是提供一个build()方法来描述如何根据其他较低级别的widget来显示自己。
- 本示例中的body的widget树中包含了一个Center widget, Center widget又包含一个 Text 子widget。Center widget可以将其子widget树对其到屏幕中心。

第2步：使用外部包(package)

在这一步中，您将开始使用一个名为english_words的开源软件包，其中包含数千个最常用的英文单词以及一些实用功能。

您可以在pub.dartlang.org上找到[english_words](https://pub.dartlang.org/packages/english_words)软件包以及其他许多开源软件包

1. pubspec文件管理Flutter应用程序的assets(资源，如图片、package等)。在pubspec.yaml中，将english_words (3.1.0或更高版本) 添加到依赖项列表，如下面高亮显示的行：

```
dependencies:  
  flutter:  
    sdk: flutter  
  
  cupertino_icons: ^0.1.0  
  english_words: ^3.1.0
```

2. 在Android Studio的编辑器视图中查看pubspec时，单击右上角的 **Packages get**，这会将依赖包安装到您的项目。您可以在控制台中看到以下内容：

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

3. 在 lib/main.dart 中, 引入 english_words, 如高亮显示的行所示:

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

在您输入时，Android Studio会为您提供有关库导入的建议。然后它将呈现灰色的导入字符串，让您知道导入的库尚未使用（到目前为止）

4. 使用 English words 包生成文本来替换字符串“Hello World”。

Tip: “驼峰命名法” (称为 “upper camel case” 或 “Pascal case”), 表示字符串中的每个单词 (包括第一个单词) 都以大写字母开头。所以, “uppercamelcase” 变成 “UpperCamelCase”

进行以下更改, 如高亮部分所示:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final wordPair = new WordPair.random();
    return new MaterialApp(
      title: 'Welcome to Flutter',
      home: new Scaffold(
        appBar: new AppBar(
          title: new Text('Welcome to Flutter'),
        ),
        body: new Center(
          //child: new Text('Hello World'),
          child: new Text(wordPair.asPascalCase),
        ),
      ),
    );
  }
}
```

5. 如果应用程序正在运行, 请使用热重载按钮 (



lightning bolt icon

) 更新正在运行的应用程序。每次单击热重载或保存项目时, 都会在正在运行的应用程序中随机选择不同的单词对。这是因为单词对是在 `build` 方法内部生成的。每次 `MaterialApp` 需要渲染时或者在 Flutter Inspector 中切换平台时 `build` 都会运行。



screenshot at completion of second step

遇到问题？

如果您的应用程序运行不正常，请查找是否有拼写错误。如果需要，使用下面链接中的代码来对比更正。

- [pubspec.yaml](#) (The pubspec.yaml file won't change again.)
 - [lib/main.dart](#)
-

第3步：添加一个 有状态的部件 (Stateful widget)

Stateless widgets 是不可变的, 这意味着它们的属性不能改变 - 所有的值都是最终的. Stateful widgets 持有的状态可能在widget生命周期中发生变化. 实现一个 stateful widget 至少需要两个类:

1. 一个 StatefulWidget类。
2. 一个 State类。 StatefulWidget类本身是不变的，但是 State类在widget生命周期中始终存在。

在这一步中，您将添加一个有状态的widget-RandomWords，它创建其State类 RandomWordsState。State类将最终为widget维护建议的和喜欢的单词对。

1. 添加有状态的 RandomWords widget 到 main.dart。它也可以在MyApp之外的文件的任何位置使用，但是本示例将它放到了文件的底部。RandomWords widget除了创建State类之外几乎没有其他任何东西

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

2. 添加 RandomWordsState 类.该应用程序的大部分代码都在该类中，该类持有RandomWords widget的状态。这个类将保存随着用户滚动而无限增长的生成的单词对，以及喜欢的单词对，用户通过重复点击心形 ♥ 图标来将它们从列表中添加或删除。

你会一步一步地建立这个类。首先，通过添加高亮显示的代码创建一个最小类

```
class RandomWordsState extends State<RandomWords> {  
}
```

3. 在添加状态类后，IDE会提示该类缺少build方法。接下来，您将添加一个基本的build方法，该方法通过将生成单词对的代码从MyApp移动到 RandomWordsState来生成单词对。

将build方法添加到RandomWordState中，如下面高亮代码所示


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

4. 通过下面高亮显示的代码，将生成单词对代的码从MyApp移动到RandomWordsState中

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final wordPair = new WordPair.random(); // 删除此行

    return new MaterialApp(
      title: 'Welcome to Flutter',
      home: new Scaffold(
        appBar: new AppBar(
          title: new Text('Welcome to Flutter'),
        ),
        body: new Center(
          //child: new Text(wordPair.asPascalCase),
          child: new RandomWords(),
        ),
      ),
    );
  }
}
```

重新启动应用程序。如果您尝试热重载，则可能会看到一条警告：

```
Reloading...
Not all changed program elements ran during view reassembly; consider
restarting.
```

这可能是误报，但考虑到重新启动可以确保您的更改在应用界面中生效。

应用程序应该像之前一样运行，每次热重载或保存应用程序时都会显示一个单词对。



screenshot at completion of third step

遇到问题？

如果您的应用程序运行不正常，可以使用下面链接中的代码来对比更正。

- [lib/main.dart](#)

第4步：创建一个无限滚动ListView

在这一步中，您将扩展（继承）RandomWordsState类，以生成并显示单词对列表。当用户滚动时，ListView中显示的列表将无限增长。ListView的builder工厂构造函数允许您按需建立一个懒加载的列表视图。

1. 向RandomWordsState类中添加一个 `_suggestions` 列表以保存建议的单词对。该变量以下划线 (`_`) 开头，在Dart语言中使用下划线前缀标识符，会强制其变成私有的。

另外，添加一个 `biggerFont` 变量来增大字体大小

```
class RandomWordsState extends State<RandomWords> {  
  final _suggestions = <WordPair>[];  
  
  final _biggerFont = const TextStyle(fontSize: 18.0);  
  ...  
}
```

2. 向RandomWordsState类添加一个 `_buildSuggestions()` 函数。此方法构建显示建议单词对的ListView。

ListView类提供了一个builder属性，`itemBuilder` 值是一个匿名回调函数，接受两个参数- BuildContext和行迭代器*i*。迭代器从0开始，每调用一次该函数，*i*就会自增1，对于每个建议的单词对都会执行一次。该模型允许建议的单词对列表在用户滚动时无限增长。

添加如下高亮的行：

```
class RandomWordsState extends State<RandomWords> {  
  ...  
  Widget _buildSuggestions() {  
    return new ListView.builder(  
      padding: const EdgeInsets.all(16.0),  
      // 对于每个建议的单词对都会调用一次itemBuilder，然后将单词对添加到ListTile行中  
      // 在偶数行，该函数会为单词对添加一个ListTile row.  
      // 在奇数行，该行书湖添加一个分割线widget，来分隔相邻的词对。  
      // 注意，在小屏幕上，分割线看起来可能比较吃力。  
      itemBuilder: (context, i) {  
        // 在每一列之前，添加一个1像素高的分隔线widget  
        if (i.isOdd) return new Divider();  
  
        // 语法 “i ~/ 2” 表示i除以2，但返回值是整形（向下取整），比如i为：1, 2, 3, 4, 5  
        // 时，结果为0, 1, 1, 2, 2， 这可以计算出ListView中减去分隔线后的实际单词对数量  
        final index = i ~/ 2;  
        // 如果是建议列表中最后一个单词对  
        if (index >= _suggestions.length) {  
          // ...接着再生成10个单词对，然后添加到建议列表  
          _suggestions.addAll(generateWordPairs().take(10));  
        }  
        return _buildRow(_suggestions[index]);  
      }  
    );  
  }  
}
```

```
}  
}
```

3. 对于每一个单词对，`_buildSuggestions`函数都会调用一次`_buildRow`。这个函数在`ListTile`中显示每个新词对，这使您在下一步中可以生成更漂亮的显示行

在`RandomWordsState`中添加一个`_buildRow`函数：

```
class RandomWordsState extends State<RandomWords> {  
  ...  
  
  Widget _buildRow(WordPair pair) {  
    return new ListTile(  
      title: new Text(  
        pair.asPascalCase,  
        style: _biggerFont,  
      ),  
    );  
  }  
}
```

4. 更新`RandomWordsState`的`build`方法以使用`_buildSuggestions()`，而不是直接调用单词生成库。更改后如下面高亮部分：

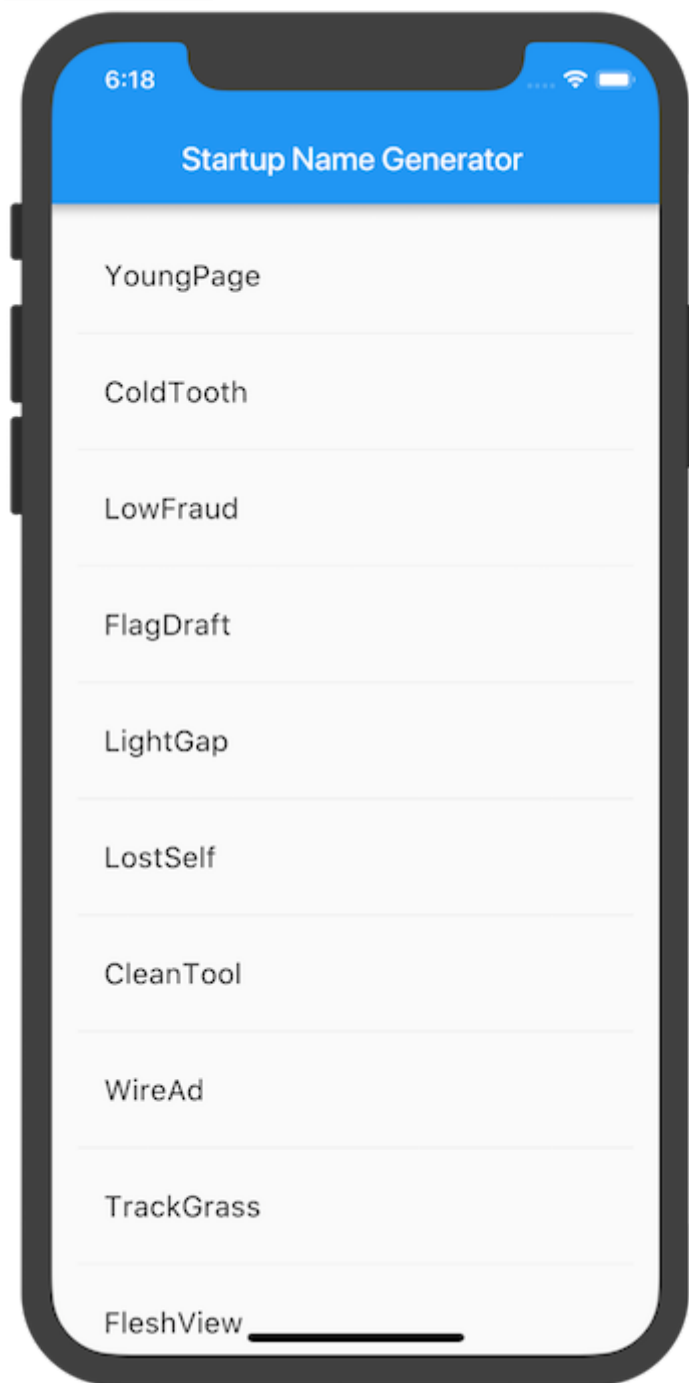
```
class RandomWordsState extends State<RandomWords> {  
  ...  
  
  @override  
  Widget build(BuildContext context) {  
    final wordPair = new WordPair.random(); // 删除这两行  
    return new Text(wordPair.asPascalCase);  
    return new Scaffold (  
      appBar: new AppBar(  
        title: new Text('Startup Name Generator'),  
      ),  
      body: _buildSuggestions(),  
    );  
  }  
  ...  
}
```

5. 更新`MyApp`的`build`方法。从`MyApp`中删除`Scaffold`和`AppBar`实例。这些将由`RandomWordsState`管理，这使得用户在下一步中从一个屏幕导航到另一个屏幕时，可以更轻松地更改导航栏中的路由名称。

用下面高亮部分替换最初的`build`方法：

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Startup Name Generator',
      home: new RandomWords(),
    );
  }
}
```

重新启动应用程序。你应该看到一个单词对列表。尽可能地向下滚动，您将继续看到新的单词对。



screenshot at completion of fourth step

遇到问题？

如果你的应用没有正常运行，你可以使用一下链接中的代码对比更正。

- [lib/main.dart](#)

第5步：添加交互

在这一步中，您将为每一行添加一个可点击的心形 ♥ 图标。当用户点击列表中的条目，切换其“收藏”状态时，将该词对添加到或移除出“收藏夹”。

1. 添加一个 `_saved` `Set`(集合) 到 `RandomWordsState`。这个集合存储用户喜欢（收藏）的单词对。在这里，`Set`比`List`更合适，因为`Set`中不允许重复的值。

```
class RandomWordsState extends State<RandomWords> {  
  final _suggestions = <WordPair>[];  
  
  final _saved = new Set<WordPair>();  
  
  final _biggerFont = const TextStyle(fontSize: 18.0);  
  ...  
}
```

2. 在 `_buildRow` 方法中添加 `alreadySaved` 来检查确保单词对还没有添加到收藏夹中。

```
Widget _buildRow(WordPair pair) {  
  final alreadySaved = _saved.contains(pair);  
  ...  
}
```

3. 同时在 `_buildRow()` 中，添加一个心形 ♥ 图标到 `ListTiles`以启用收藏功能。接下来，你就可以给心形 ♥ 图标添加交互能力了。

添加下面高亮的行：

```
Widget _buildRow(WordPair pair) {  
  final alreadySaved = _saved.contains(pair);  
  return new ListTile(  
    title: new Text(  
      pair.asPascalCase,  
      style: _biggerFont,  
    ),  
    trailing: new Icon(  
      alreadySaved ? Icons.favorite : Icons.favorite_border,  
      color: alreadySaved ? Colors.red : null,  
    ),  
  );  
}
```

```

    ),
  );
}

```

4. 重新启动应用。你现在可以在每一行看到心形♥图标，但它们还没有交互。
5. 在 `_buildRow` 中让心形♥图标变得可以点击。如果单词条目已经添加到收藏夹中，再次点击它将其从收藏夹中删除。当心形♥图标被点击时，函数调用 `setState()` 通知框架状态已经改变。

添加如下高亮的行：

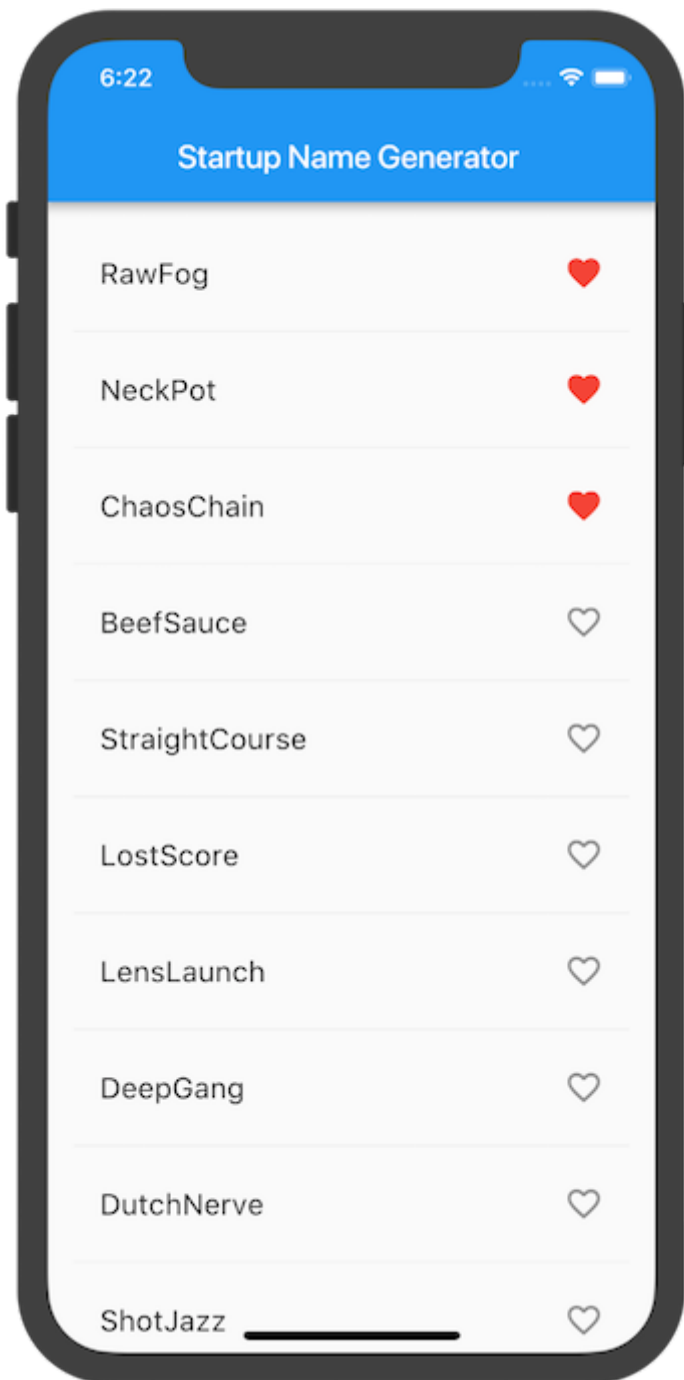
```

Widget _buildRow(WordPair pair) {
  final alreadySaved = _saved.contains(pair);
  return new ListTile(
    title: new Text(
      pair.asPascalCase,
      style: _biggerFont,
    ),
    trailing: new Icon(
      alreadySaved ? Icons.favorite : Icons.favorite_border,
      color: alreadySaved ? Colors.red : null,
    ),
    onTap: () {
      setState(() {
        if (alreadySaved) {
          _saved.remove(pair);
        } else {
          _saved.add(pair);
        }
      });
    },
  );
}

```

提示：在Flutter的响应式风格的框架中，调用 `setState()` 会为State对象触发 `build()` 方法，从而导致对UI的更新

热重载你的应用。你就可以点击任何一行收藏或删除。请注意，点击一行时会生成从心形 ♥ 图标发出的水波动画



screenshot at completion of 5th step

遇到了问题？

如果您的应用没有正常运行，请查看下面链接处的代码，对比更正。

- [lib/main.dart](#)

第6步： 导航到新页面

在这一步中，您将添加一个显示收藏夹内容的新页面（在Flutter中称为路由（route））。您将学习如何在主路由和新路由之间导航（切换页面）。

在Flutter中，导航器管理应用程序的路由栈。将路由推入（push）到导航器的栈中，将会显示更新为该路由页面。从导航器的栈中弹出（pop）路由，将显示返回到前一个路由。

1. 在RandomWordsState的build方法中为AppBar添加一个列表图标。当用户点击列表图标时，包含收藏夹的新路由页面入栈显示。

提示：某些widget属性需要单个widget（child），而其它一些属性，如action，需要一组widgets(children），用方括号[]表示。

将该图标及其相应的操作添加到build方法中：

```
class RandomWordsState extends State<RandomWords> {  
  ...  
  @override  
  Widget build(BuildContext context) {  
    return new Scaffold(  
      appBar: new AppBar(  
        title: new Text('Startup Name Generator'),  
        actions: <Widget>[  
          new IconButton(icon: new Icon(Icons.list), onPressed: _pushSaved),  
        ],  
      ),  
      body: _buildSuggestions(),  
    );  
  }  
  ...  
}
```

2. 向RandomWordsState类添加一个 _pushSaved() 方法.

```
class RandomWordsState extends State<RandomWords> {  
  ...  
  void _pushSaved() {  
  }  
}
```

热重载应用，列表图标将会出现在导航栏中。现在点击它不会有任何反应，因为 _pushSaved 函数还是空的。

3. 当用户点击导航栏中的列表图标时，建立一个路由并将其推入到导航管理器栈中。此操作会切换页面以显示新路由。

新页面的内容在MaterialPageRoute的builder属性中构建，builder是一个匿名函数。

添加Navigator.push调用，这会使路由入栈（以后路由入栈均指推入到导航管理器的栈）

```
void _pushSaved() {  
  Navigator.of(context).push(  
  );  
}
```

4. 添加MaterialPageRoute及其builder。现在，添加生成ListTile行的代码。

ListTile的divideTiles()方法在每个ListTile之间添加1像素的分割线。该divided变量持有最终的列表项。

```
void _pushSaved() {  
  Navigator.of(context).push(  
    new MaterialPageRoute(  
      builder: (context) {  
        final tiles = _saved.map(  
          (pair) {  
            return new ListTile(  
              title: new Text(  
                pair.asPascalCase,  
                style: _biggerFont,  
              ),  
            );  
          },  
        );  
        final divided = ListTile  
          .divideTiles(  
            context: context,  
            tiles: tiles,  
          )  
          .toList();  
      },  
    ),  
  );  
}
```

5. builder返回一个Scaffold，其中包含名为“Saved Suggestions”的新路由的应用栏。新路由的body由包含ListTiles行的ListView组成；每行之间通过一个分隔线分隔。

添加如下高亮的代码：

```
void _pushSaved() {  
  Navigator.of(context).push(  
    new MaterialPageRoute(  
      builder: (context) {
```

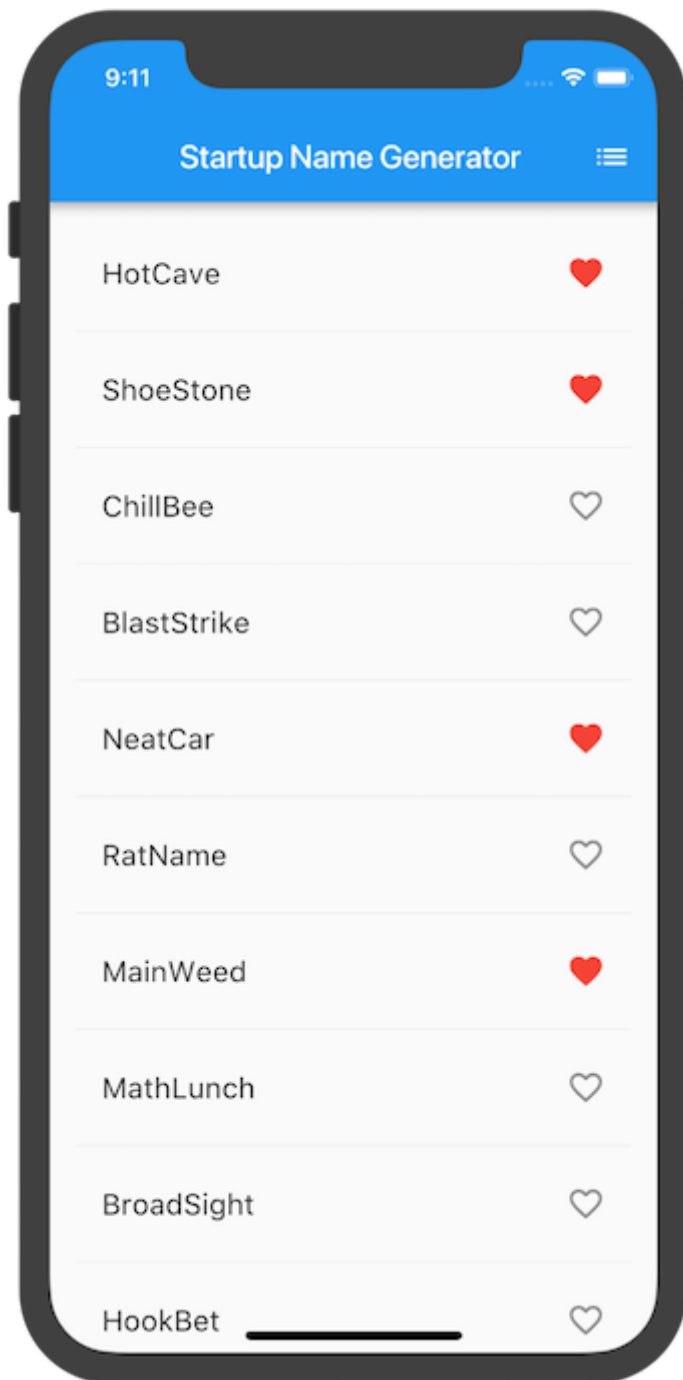
```

        final tiles = _saved.map(
          (pair) {
            return new ListTile(
              title: new Text(
                pair.asPascalCase,
                style: _biggerFont,
              ),
            );
          },
        );
        final divided = ListTile
          .divideTiles(
            context: context,
            tiles: tiles,
          )
          .toList();

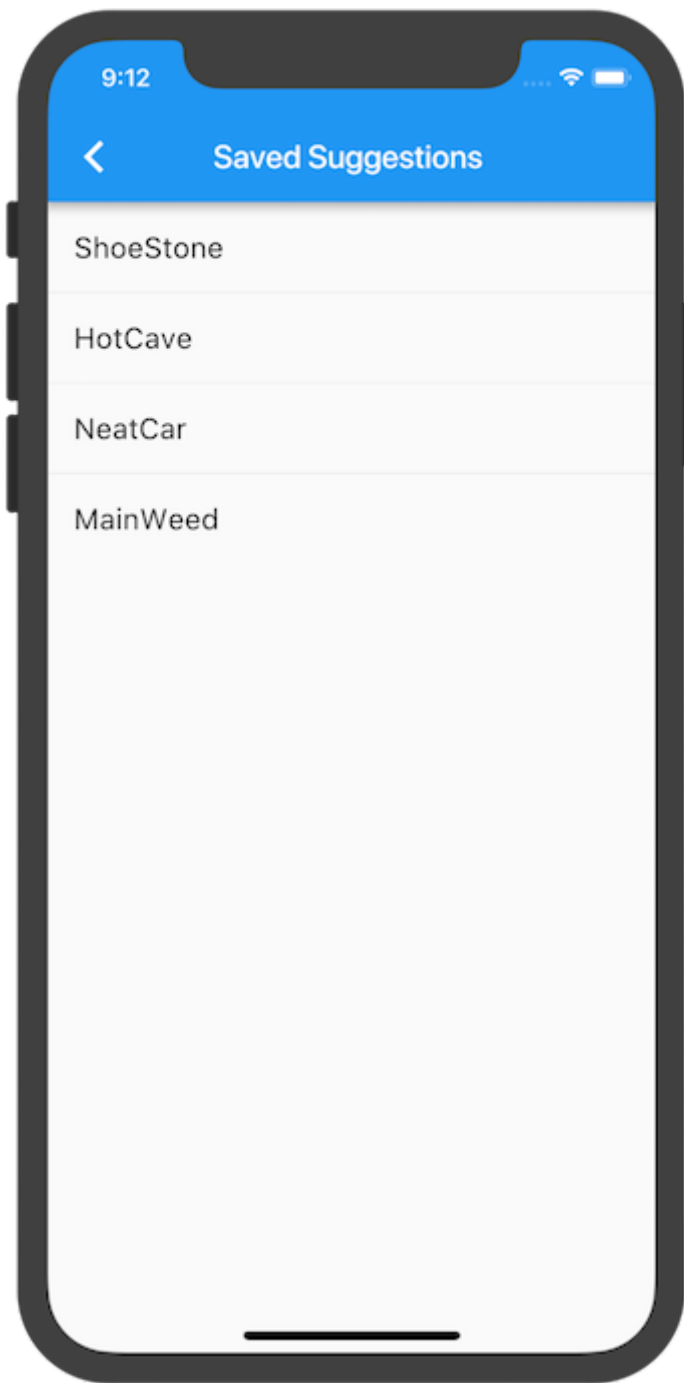
        return new Scaffold(
          appBar: new AppBar(
            title: new Text('Saved Suggestions'),
          ),
          body: new ListView(children: divided),
        );
      },
    ),
  );
}

```

6. 热重载应用程序。收藏一些选项，并点击应用栏中的列表图标，在新路由页面中显示收藏的内容。请注意，导航器会在应用栏中添加一个“返回”按钮。你不必显式实现Navigator.pop。点击后退按钮返回到主页路由。



screenshot at completion of 6th step



second route

遇到了问题？

如果您的应用不能正常工作，请参考下面链接处的代码，对比并更正。

- [lib/main.dart](#)

第7步：使用主题更改UI

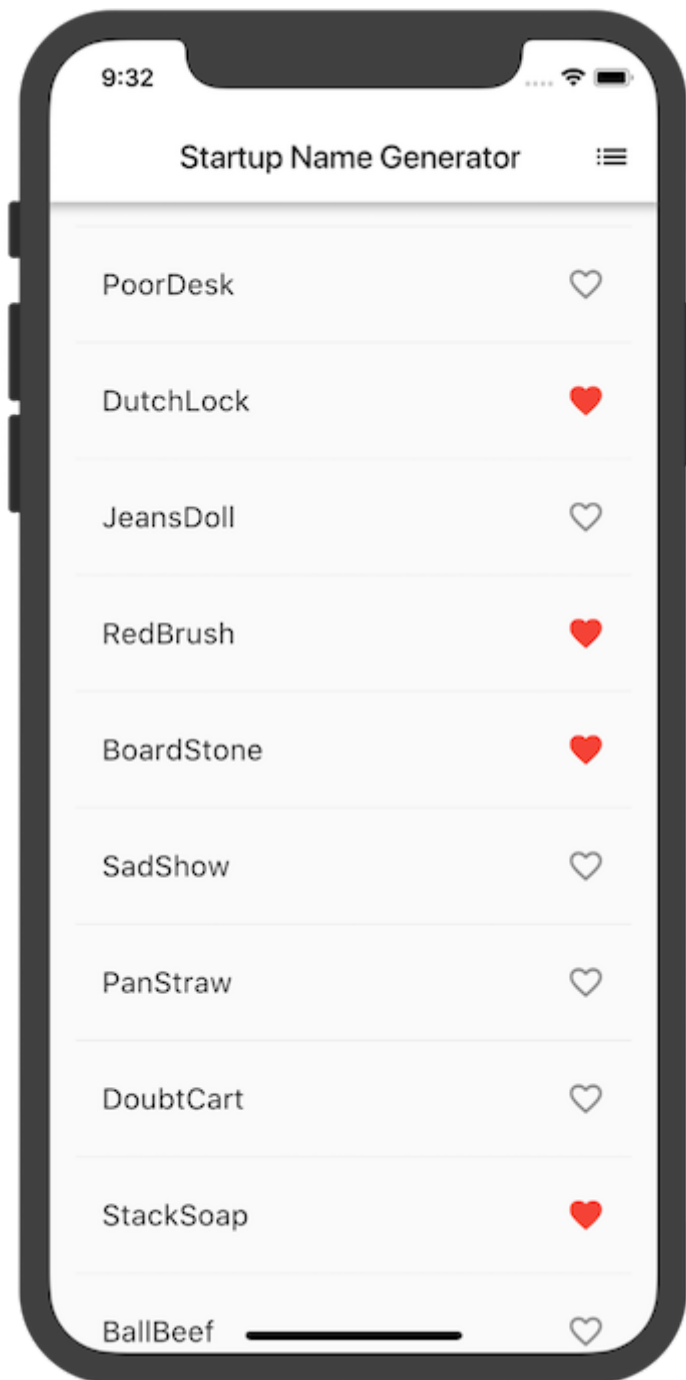
在这最后一步中，您将会使用主题。主题控制您应用程序的外观和风格。您可以使用默认主题，该主题取决于物理设备或模拟器，也可以自定义主题以适应您的品牌。

1. 您可以通过配置ThemeData类轻松更改应用程序的主题。 您的应用程序目前使用默认主题，下面将更改primary color颜色为白色。

通过如下高亮部分代码，将应用程序的主题更改为白色：

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return new MaterialApp(  
      title: 'Startup Name Generator',  
      theme: new ThemeData(  
        primaryColor: Colors.white,  
      ),  
      home: new RandomWords(),  
    );  
  }  
}
```

2. 热重载应用。 请注意，整个背景将会变为白色，包括应用栏。
3. 作为读者的一个练习，使用 [ThemeData](#) 来改变UI的其他方面。 Material library中的 [Colors](#)类提供了许多可以使用的颜色常量， 你可以使用热重载来快速简单地尝试、实验。



screenshot at completion of 7th step

遇到了问题？

如果你遇到了问题，请查看以下链接中应用程序的最终代码。

- [lib/main.dart](#)

做得好！

你已经编写了一个可以在iOS和Android上运行的交互式Flutter应用程序。在这个例子中，你已经做了下面这些事：

- 从头开始创建一个Flutter应用程序.
- 编写 Dart 代码.
- 利用外部的第三方库.
- 使用热重载加快开发周期.
- 实现一个有状态的widget , 为你的应用增加交互.
- 用ListView和ListTiles创建一个延迟加载的无限滚动列表.
- 创建了一个路由并添加了在主路由和新路由之间跳转逻辑
- 了解如何使用主题更改应用UI的外观.