Table of Contents

目录 1.1

概述 1.2

命名规范 1.3

- UpperCamelCase 1.3.1
- lowerCamelCase 1.3.2
- lowercase_with_underscores 1.3.3
- 下划线“_”的使用 1.3.4

代码风格 1.4

自定义 Widget 1.5

Dart 和 Flutter 编码规范及解释

命名规范

Dart 语言中使用的命名风格有三种。

- UpperCamelCase 首字母大写的驼峰命名法。
- lowerCamelCase 首字母小写的驼峰命名法。
- lowercase_with_underscores 下划线连接的小写字母单词。

UpperCamelCase

UpperCamelCase 即所有单词首字母大写，其他字母小写，并且没有连接符号，又称驼峰命名法。

Dart 中使用 UpperCamelCase 的情形有

- 类名 Class
- 枚举类型名 enum
- 类型定义 typedef
- 泛型参数
- Extension 名

举例说明

```
/// 类名
class MyApp extends StatelessWidget {
}
```

```
/// 枚举
enum TextAlign {
  left,
  right,
  center,
  justify,
  start,
  end,
}
```

```
/// 类型定义
typedef VoidCallback = void Function();
```

```
/// 泛型参数
/// 类的泛型参数 K、V
class NameValuePair<K, V> {
  K name;
  V value;
}

/// 函数的泛型参数 T
Future<T> invokeMethod<T>(String method, [ dynamic arguments ]) async {
}
```

```
/// Extension
extension NumberParsing on String {
  int parseInt() {
    return int.parse(this);
  }
  // ···
}
```

lowerCamelCase

lowerCamelCase 即第一个单词首字母小写，之后的单词首字母大写的驼峰命名法。

Dart 中使用 lowerCamelCase 的情形有

- 类成员（字段、方法）
- 变量（全局变量、局部变量）
- 常量/枚举值
- 函数名和函数参数名

举例说明

```
/// 类成员
class Color {

  /// 方法
  Color withAlpha(int a) {
    return Color.fromARGB(a, red, green, blue);
  }

  /// 字段
  final int value;
}

/// 变量
int value;

/// 运行时常量
final int value;

/// 编译时常量
const int value;

enum Week {
  /// 枚举值
  mon,
  tue,
  wed,
  thu,
  fri,
  sat,
  sun,
}

/// 函数名
static bool isNotEmpty(
  String text // 函数参数名
) {
  return text != null && text.length > 0;
}
```

lowercase_with_underscores

lowercase_with_underscores 即所有单词都为小写字母组成，单词之间使用下划线连接。

Dart 中使用 lowercase_with_underscores 的情形有

- 库名
- 包名
- 文件夹名
- 文件名

举例说明

```
library flutter_screenutil;
import 'form_data.dart';
import 'form_data.dart' as form_data;
import 'package:angular_components/angular_components' as angular_components;
```

下划线“_”的使用

在不同的语言中，都可以用下划线连接单词构成变量或常量。除此之外，其含义和用法各不相同。

Dart 中规定以“_”开头的变量和方法的访问权限为私有。

如

```
/// 私有变量
HttpClient _httpClient = new HttpClient();

/// 私有方法
void _printLog(String log) {
}
```

Dart 中没有使用双下划线和以下划线结尾的情形，所以以下命名方式是**不符合规范的**

```
var __num;    ❌
var num_;     ❌
var __num__;  ❌
```

在定义变量和方法的时候，如非必须提供外部访问权限，则优先考虑定义为私有。

代码风格

缩进

Dart 和 Flutter 的代码缩进均为 2 个空格。

避免 var 的使用

应避免使用 var 定义变量。

虽然 Dart 支持类型推导，但代价是更长的编译和执行时长，同时也会降低代码的可读性，在 Dart 和 Flutter 源码以及三方库中极少使用。

函数的定义

Dart 不支持函数重载，要实现类似功能时，应优先考虑使用 **命名可选参数**。

路由（Route）

自定义 Widget

Flutter 自定义 Widget 应优先考虑组合的方式，将基础的 Widget 组装成复杂的 Widget。

如非必须自己管理状态，则优先继承 StatelessWidget，由父 Widget 管理状态。

> 以按钮为例，一般有普通状态和按下状态，这两种状态的切换与父 Widget 无关，完全由自己管理，所以按钮一般继承自 StatefulWidget。
>
> 而 Text 之类的 Widget，其内容如何显示完全由外部决定，这种只需要继承 StatelessWidget 即可。
>
> 再比如 TabBar 这一类的 Widget，切换 Tab 之后，不仅自身状态需要更新，外部的页面也需要切换，这种 Widget 的状态可以自己管理，也可以交给父 Widget 管理，所以继承 StatefulWidget 或 StatelessWidget 都可以。