第 5 章

编 码 规 范

16min

编码习惯都是因人而异的，并没有所谓的最佳方案。如果你是一个人开发，当然不需要在意这些问题，但是如果你的代码需要展现给别人，或者需要与别人协同开发，编码规范就非常有必要了。

规范主要分为以下几部分：

- 样式规范；
- 文档规范；
- 使用规范。

每个部分都有许多的例子说明，本章将会从项目代码中选取最基本、最典型和发生率较高的一些情况，作为规范说明。

5.1 样式规范

程序代码中到处都是各种标识符，因此取一个一致并且符合规范的名字非常重要。本节将从以下几点来说明样式规范。

（1）类、枚举、类型定义，以及泛型，都需要使用大写开头的驼峰命名法。如下面代码所示，类名 Person、HttpService，以及类型定义 EventChannel 均使用了驼峰命名法。

```
//类名规范
class Person {
    //...
}

//类名规范
class HttpService {
    //...
}

//类型定义规范
typedef EventChannel<T> = bool Function(T value);
```

(2) 在使用元数据的时候,也要使用驼峰命名法。如下面代码中@JsonSerializable 及@JsonKey 遵循了此规范。

```
//类元数据
@JsonSerializable()
class AddressEntity extends Object {

  //变量元数据
  @JsonKey(name: 'id')
  int id;

  //变量元数据
  @JsonKey(name: 'name')
  String name;
}
```

(3) 命名库、包、目录、dart 文件都应该是小写加上下画线。正确的代码写法如下:

```
//命名库
library peg_parser.source_scanner;

//导入 dart 文件
import 'file_system.dart';
import 'slider_menu.dart';
```

错误的命名如下所示,其中库名 pegparser.SourceScanner 中的 S 应该为小写,file-system.dart 文件名中间用下画线表示成 file_system.dart,SliderMenu.dart 文件名的大写应该改为 slider_menu.dart。

```
//命名库
library pegparser.SourceScanner;

//导入 dart 文件
import 'file-system.dart';
import 'SliderMenu.dart';
```

(4) 将引用的库使用 as 转换的名字也应该是小写加下画线。正确的写法如下:

```
//导入库文件 as 命名规范
import 'dart:math' as math;
import 'package:angular_components/angular_components'
    as angular_components;
import 'package:js/js.dart' as js;
```

下面示例中是错误的写法,Math 应为小写,angularComponents 应改为 angular_components。JS 应改为 js。

```
//导入库文件
import 'dart:math' as Math;
import 'package:angular_components/angular_components'
    as angularComponents;
import 'package:js/js.dart' as JS;
```

(5) 变量名、方法、参数名都应采用小写开头的驼峰命名法。正确的代码写法如下：

```
//变量名 item
var item;

//变量名 httpRequest
HttpRequest httpRequest;

//方法名 align 参数名 clearItems
void align(bool clearItems) {
  //...
}

//常量名 pi
const pi = 3.14;
//常量名 defaultTimeout
const defaultTimeout = 1000;
//常量名 urlScheme
final urlScheme = RegExp('^([a-z]+):');

class Dice {
  //静态常量名 numberGenerator
  static final numberGenerator = Random();
}
```

下面代码中常量名使用大写，没有遵循规范。

```
//常量名 PI
const PI = 3.14;
//常量名 DefaultTimeout
const DefaultTimeout = 1000;
//常量名 URL_SCHEME
final URL_SCHEME = RegExp('^([a-z]+):');

class Dice {
  //静态常量名 NUMBER_GENERATOR
  static final NUMBER_GENERATOR = Random();
}
```

(6) 花括号的用法也有一定的规范。只有一个 if 语句且没有 else 的时候，并且在一行内能够很好地展示就可以不用花括号，如下面代码就不需要花括号。

```
if (arg == null) return defaultValue;
```

但是，如果一行内展示比较勉强的话，就需要用花括号了，正确的代码写法如下：

```
if (value != obj.value) {
  return value < obj.value;
}
```

去掉花括号的写法就没有遵循规范，代码如下：

```
if (value != obj.value)
  return value < obj.value;
```

5.2 文档规范

在 Dart 的注释中，推荐使用三个斜杠“///”而不是两个斜杠“//”。至于为什么要这样做，官方表示是由于历史原因及他们觉得这样在某些情况下看起来更方便阅读。下面将从以下几点来说明文档规范。

（1）下面是一种推荐的注释写法。

```
///当图片上传成功后，记录当前上传的图片在服务器中的位置
String imgServerPath;
```

下面的写法不推荐使用。

```
//当图片上传成功后，记录当前上传的图片在服务器中的位置
String imgServerPath;
```

（2）文档注释应该以一句简明的话开头，代码如下：

```
///使用传入的路径[path]删除此文件
void delete(String path) {
  …
}
```

下面的写法显得有些啰唆，建议简化。

```
///需要确定使用的用户授权删除权限
///需要确定磁盘上有此文件
///[path]参数如果不传将会抛出 IO 异常[IOError]
void delete(String path) {
  …
}
```

(3) 如果有多行注释,将注释的第一句与其他内容分隔开来。

```
///使用此路径[path]删除文件
///
///如果文件找不到会抛出一个 IO 异常[IOError];如果没有权限并且文件
///存在会抛出一个[PermissionError]异常
void delete(String path) {
  …
}
```

下面的几行注释代码连在一起写是不推荐使用的。

```
///使用此路径[path]删除文件
///如果文件找不到会抛出一个 IO 异常[IOError];如果没有权限并且
///文件存在会抛出一个[PermissionError]异常
void delete(String path) {
  …
}
```

(4) 在方法的上面通常会写成如下形式,这种方式在 Dart 里是不推荐使用的。需要使用方括号去声明参数、返回值,以及抛出的异常。

```
///使用 name 和 detail 添加一个地址
///
///@param name 用户名称
///@param detail 用户详情
///@returns 返回一个 Address
///@throws ArgumentError 如果没有传递 detail 会抛出此异常
Address addAddress(String name, String detail){
    …
}
```

下面的示例代码中参数 name、detail 和异常 ArgumentError 都用方括号括起来了,推荐使用此方式。

```
///添加一个地址
///
///通过[name]及[detail]参数可以创建一个 Address 对象
///如果没有传递[detail]参数会抛出此异常[ArgumentError]
Address addAddress(String [name], String [detail]){
    …
}
```

5.3 使用规范

在使用过程中也需要遵循一定的规范，例如库的依赖、变量赋值、字符串，以及集合的使用等。

5.3.1 依赖

推荐使用相对路径导入依赖。如果项目结构如下：

```
my_package
 └── lib
    ├── src
    |   └── utils.dart
    └── api.dart
```

想要在 api.dart 中导入 utils.dart，可以按如下方式导入。

```
import 'src/utils.dart';
```

下面的方式不推荐使用，因为一旦 my_package 改名之后，对应的所有需要导入的地方都需要修改，而相对路径就没有这个问题。

```
import 'package:my_package/src/utils.dart';
```

5.3.2 赋值

使用“??”将 null 值做一个转换。在 dart 中“??”操作符表示当一个值为空时会给它赋值“??”后面的数据。

下面的写法是错误的。

```
if (optionalThing?.isEnabled) {
  print("Have enabled thing.");
}
```

当 optionalThing 为空的时候，上面就会有空指针异常了。

这里说明一下，“?.”操作符相当于做了一次判空操作，只有当 optionalThing 不为空的时候才会调用 isEnabled 参数，当 optionalThing 为空时则默认返回 null，用在 if 判断句中自然就不行了。

下面是正确做法，代码如下：

```
//如果为空的时候想返回 false
optionalThing?.isEnabled ?? false;

//如果为空的时候想返回 ture
optionalThing?.isEnabled ?? true;
```

下面的写法是错误的。

```
optionalThing?.isEnabled == true;

optionalThing?.isEnabled == false;
```

5.3.3 字符串

在 Dart 中,不推荐使用加号“+”去连接两个字符串,而使用回车键直接分隔字符串。代码如下:

```
String str = '这个方法是用来请求服务端数据的,'
             '返回的数据格式为 Json';
```

下面代码使用“+”连接字符串不推荐使用。

```
  String str = '这个方法是用来请求服务端数据的,' +
      '返回的数据格式为 Json';
```

在多数情况下推荐使用$variable 或${}来连接字符串与变量值。其中${}里可以放入一个表达式。下面的示例代码就使用了这种规范。

```
String name = '张三';
int age = 20;
int score = 89;
void sayHello(){

  //$name 获取变量值
  String userInfo = '你好, $name 你的年龄是: ${age}.';
  print(userInfo);

  //${}可以加入变量及表达式
  String scoreInfo = '成绩是否及格: ${ score >= 60 ? true : false }.';
  print(scoreInfo);

}
```

5.3.4 集合

Dart 中创建空的可扩展 List 有两种方法：[]和 List()。创建空的 HashMap 有三种方法：{}、Map()和 LinkedHashMap()。

(1) 如果要创建不可扩展的列表或其他一些自定义集合类型，那么务必使用构造函数。尽可能使用简单的字面量创建集合。代码如下：

```
var points = [];
var addresses = {};
```

下面的创建方式不推荐使用。

```
var points = List();
var addresses = Map();
```

(2) 当想要指定类型的时候，推荐如下写法。

```
var points = <Point>[];
var addresses = <String, Address>{};
```

不推荐如下方式。

```
var points = List<Point>();
var addresses = Map<String, Address>();
```

(3) 不要使用.length 的方法去表示一个集合是否为空。使用 isEmpty 或 isNotEmpyt。代码如下：

```
if (list.isEmpty){
    ...
}

if (list.isNotEmpty){
    ...
}
```

(4) 避免使用带有方法字面量的 Iterable.forEach()。forEach()方法在 JavaScript 中被广泛使用，因为内置的 for-in 循环不能达到通常想要的效果。在 Dart 中如果要迭代序列，那么惯用的方法是使用循环。代码如下：

```
for (var person in people) {
  ...
}
```

下面的 forEach 方式不推荐使用。

```
people.forEach((person) {
  ...
});
```

(5) 不要使用 List.from(),除非打算更改结果的类型。有两种方法去获取 iterable,分别是 List.from()和 Iterable.toList()。代码如下:

```
void main(){
  //创建一个 List<int>
  var iterable = [1, 2, 3];

  //输出"List<int>"
  print(iterable.toList().toString());
}
```

下面的示例代码通过 List.from()的方式输出 iterable,但不要这样使用。

```
void main(){
  //创建一个 List<int>
  var iterable = [1, 2, 3];

  //输出"List<int>"
  print(List.from(iterable).toString());
}
```

(6) 使用 whereType()过滤一个集合。下面使用 where 过滤一个集合是错误的用法。

```
//集合
var objects = [1, "a", 2, "b", 3];
//where 过滤
var ints = objects.where((e) => e is int);
```

正确的用法如下:

```
//集合
var objects = [1, "a", 2, "b", 3];
//whereType 过滤
var ints = objects.whereType<int>();
```

5.3.5 参数

方法参数值的设置规范有如下几种情况。

(1) 使用等号“=”给参数设置默认值,代码如下:

```
void insert(Object item, {int at = 0}) {
    //…
}
```

下面代码使用冒号“:”是错误的用法。

```
void insert(Object item, {int at: 0}) {
    //…
}
```

(2) 不要将参数的默认值设置为 null,下面的写法是正确的。

```
void error([String message]) {
  stderr.write(message ?? '\n');
}
```

下面的代码将 message 参数设置为 null 是错误的。

```
void error([String message = null]) {
  stderr.write(message ?? '\n');
}
```

5.3.6 变量

变量的使用便于存储可以计算的值。首先看下面一个求圆的面积的例子。

```
//定义圆类
class Circle {

  //PI 值
  num pi = 3.14;

  //圆的半径
  num _radius;

  //获取圆的半径
  num get radius => _radius;
  //设置圆的半径
  set radius(num value) {
    _radius = value;
    //计算圆的面积
    _recalCulate();
  }

  //圆的面积
```