**听app**

**编**

**码**

**格**

**式**

**课 程： 软件工程**

**题 目： 幻听**

**基于Flutter框架的集各大音乐平台于一体的音乐APP**

**成 员：邢海粟 31801347 组长**

**黄德煜 31801163 组员**

**章拾瑜 31801335 组员**

|  |  |  |  |
| --- | --- | --- | --- |
| **文件名** | **版本号** | **更改日期** | **更改人** |
| **SE2020-G06-编码格式** | V0.0.1 | 2020.12.21 | 黄德煜 |
| **SE2020-G06-编码格式** |  |  |  |
| **SE2020-G06-编码格式** |  |  |  |
| **SE2020-G06-编码格式** |  |  |  |

## 一. 代码风格

#### 1.大驼峰命名

类、枚举、typedef和类型参数

|  |
| --- |
| class SliderMenu { ... }  class HttpRequest { ... }  typedef Predicate = bool Function<T>(T value); |

包括用于元数据注释的类

|  |
| --- |
| class Foo {  const Foo([arg]);  }  @Foo(anArg)  class A { ... }  @Foo()  class B { ... } |

#### 2.使用小写加下划线来命名库和源文件

|  |
| --- |
| library peg\_parser.source\_scanner;  import 'file\_system.dart';  import 'slider\_menu.dart'; |

不推荐如下写法：

|  |
| --- |
| library pegparser.SourceScanner;  import 'file-system.dart';  import 'SliderMenu.dart'; |

#### 3.使用小写加下划线来命名导入前缀

|  |
| --- |
| import 'dart:math' as math;  import 'package:angular\_components/angular\_components'  as angular\_components;  import 'package:js/js.dart' as js; |

不推荐如下写法：

|  |
| --- |
| import 'dart:math' as Math;  import 'package:angular\_components/angular\_components'  as angularComponents;  import 'package:js/js.dart' as JS; |

#### 4.使用小驼峰法命名其他标识符

|  |
| --- |
| var item;  HttpRequest httpRequest;  void align(bool clearItems) {  // ...  } |

#### 5.优先使用小驼峰法作为常量命名

|  |
| --- |
| const pi = 3.14;  const defaultTimeout = 1000;  final urlScheme = RegExp('^([a-z]+):');  class Dice {  static final numberGenerator = Random();  } |

不推荐如下写法：

|  |
| --- |
| const PI = 3.14;  const DefaultTimeout = 1000;  final URL\_SCHEME = RegExp('^([a-z]+):');    class Dice {  static final NUMBER\_GENERATOR = Random();  } |

#### 6.不使用前缀字母

因为Dart可以告诉您声明的类型、范围、可变性和其他属性，所以没有理由将这些属性编码为标识符名称。

|  |
| --- |
| defaultTimeout |

不推荐如下写法：

|  |
| --- |
| kDefaultTimeout |

### 二.代码整洁

为了代码保持整洁，可读性强，每个“部分”应该用空行分隔。

#### 在其他的dart文件引入之前先引入所需的dart库

|  |
| --- |
| import 'dart:async';  import 'dart:html';  import 'package:bar/bar.dart';  import 'package:foo/foo.dart'; |

#### 在相对的dart文件引入之前先引入在包中的库

|  |
| --- |
| import 'package:bar/bar.dart';  import 'package:foo/foo.dart';  import 'util.dart'; |

#### 第三方包的导入先于其他包

|  |
| --- |
| import 'package:bar/bar.dart';  import 'package:foo/foo.dart';  import 'package:my\_package/util.dart'; |

#### 在所有的dart导入之后，在最后指定导出

|  |
| --- |
| import 'src/error.dart';  import 'src/foo\_bar.dart';  export 'src/error.dart'; |

不推荐如下写法：

|  |
| --- |
| import 'src/error.dart';  export 'src/error.dart';  import 'src/foo\_bar.dart'; |

### 三.所有流控制结构，请使用大括号

这样做可以避免悬浮的else问题

|  |
| --- |
| if (isWeekDay) {  print('Bike to work!');  } else {  print('Go dancing or read a book!');  } |

#### 例外

一个if语句没有else子句，其中整个if语句和then主体都适合一行。在这种情况下，如果你喜欢的话，你可以去掉大括号

|  |
| --- |
| if (arg == null) return defaultValue; |

如果流程体超出了一行需要分划请使用大括号：

|  |
| --- |
| if (overflowChars != other.overflowChars) {  return overflowChars < other.overflowChars;  } |

不推荐如下写法：

|  |
| --- |
| if (overflowChars != other.overflowChars)  return overflowChars < other.overflowChars; |

## 四.注释

### 1.要像句子一样格式化

除非是区分大小写的标识符，否则第一个单词要大写。以句号结尾(或“!”或“?”)。对于所有的注释都是如此：doc注释、内联内容，甚至TODOs。即使是一个句子片段。

|  |
| --- |
| greet(name) {  // Assume we have a valid name.  print('Hi, $name!');  } |

不推荐如下写法：

|  |
| --- |
| greet(name) {  /\* Assume we have a valid name. \*/  print('Hi, $name!');  } |

可以使用块注释(/…/)临时注释掉一段代码，但是所有其他注释都应该使用//

### 2.Doc注释

使用///文档注释来记录成员和类型。

使用doc注释而不是常规注释，可以让dartdoc找到并生成文档。

|  |
| --- |
| /// The number of characters in this chunk when unsplit.  int get length => ... |

### 3.考虑为私有api编写文档注释

Doc注释并不仅仅针对库的公共API的外部使用者。它们还有助于理解从库的其他部分调用的私有成员

#### 3.1用一句话总结开始doc注释

以简短的、以用户为中心的描述开始你的文档注释，以句号结尾。

|  |
| --- |
| /// Deletes the file at [path] from the file system.  void delete(String path) {  ...  } |

不推荐如下写法：

|  |
| --- |
| /// Depending on the state of the file system and the user's permissions,  /// certain operations may or may not be possible. If there is no file at  /// [path] or it can't be accessed, this function throws either [IOError]  /// or [PermissionError], respectively. Otherwise, this deletes the file.  void delete(String path) {  ...  } |

#### 3.2“doc注释”的第一句话分隔成自己的段落

在第一个句子之后添加一个空行，把它分成自己的段落

|  |
| --- |
| /// Deletes the file at [path].  ///  /// Throws an [IOError] if the file could not be found. Throws a  /// [PermissionError] if the file is present but could not be deleted.  void delete(String path) {  ...  } |

### 五. 字符串的使用

#### 1.使用相邻字符串连接字符串文字

如果有两个字符串字面值(不是值，而是实际引用的字面值)，则不需要使用+连接它们。就像在C和c++中，简单地把它们放在一起就能做到。这是创建一个长字符串很好的方法但是不适用于单独一行。

|  |
| --- |
| raiseAlarm(  'ERROR: Parts of the spaceship are on fire. Other '  'parts are overrun by martians. Unclear which are which.'); |

不推荐如下写法:

|  |
| --- |
| raiseAlarm('ERROR: Parts of the spaceship are on fire. Other ' +  'parts are overrun by martians. Unclear which are which.'); |

#### 2.优先使用模板字符串

|  |
| --- |
| 'Hello, $name! You are ${year - birth} years old.'; |

#### 3.在不需要的时候，避免使用花括号

|  |
| --- |
| 'Hi, $name!'  "Wear your wildest $decade's outfit." |

不推荐如下写法：

|  |
| --- |
| 'Hello, ' + name + '! You are ' + (year - birth).toString() + ' y...'; |

不推荐如下写法：

|  |
| --- |
| 'Hi, ${name}!'  "Wear your wildest ${decade}'s outfit." |

### 六.集合

#### 1. 尽可能使用集合字面量

如果要创建一个不可增长的列表，或者其他一些自定义集合类型，那么无论如何，都要使用构造函数。

|  |
| --- |
| var points = [];  var addresses = {};  var lines = <Lines>[]; |

不推荐如下写法：

|  |
| --- |
| var points = List();  var addresses = Map(); |

#### 2.不要使用.length查看集合是否为空

|  |
| --- |
| if (lunchBox.isEmpty) return 'so hungry...';  if (words.isNotEmpty) return words.join(' '); |

不推荐如下写法：

|  |
| --- |
| if (lunchBox.length == 0) return 'so hungry...';  if (!words.isEmpty) return words.join(' '); |

#### 3.考虑使用高阶方法转换序列

如果有一个集合，并且希望从中生成一个新的修改后的集合，那么使用.map()、.where()和Iterable上的其他方便的方法通常更短，也更具有声明性

|  |
| --- |
| var aquaticNames = animals  .where((animal) => animal.isAquatic)  .map((animal) => animal.name); |

#### 4.避免使用带有函数字面量的Iterable.forEach()

在Dart中，如果你想遍历一个序列，惯用的方法是使用循环。

|  |
| --- |
| for (var person in people) {  ...  } |

不推荐如下写法：

|  |
| --- |
| people.forEach((person) {  ...  }); |

#### 5.不要使用List.from()，除非打算更改结果的类型

给定一个迭代，有两种明显的方法可以生成包含相同元素的新列表

|  |
| --- |
| var copy1 = iterable.toList();  var copy2 = List.from(iterable); |

明显的区别是第一个比较短。重要的区别是第一个保留了原始对象的类型参数

|  |
| --- |
| // Creates a List<int>:  var iterable = [1, 2, 3];  // Prints "List<int>":  print(iterable.toList().runtimeType); |

|  |
| --- |
| // Creates a List<int>:  var iterable = [1, 2, 3];  // Prints "List<dynamic>":  print(List.from(iterable).runtimeType); |

### 七.参数的使用

#### 1.使用=将命名参数与其默认值分割开

由于遗留原因，Dart均允许“:”和“=”作为指定参数的默认值分隔符。为了与可选的位置参数保持一致，使用“=”。

|  |
| --- |
| void insert(Object item, {int at = 0}) { ... } |

不推荐如下写法：

|  |
| --- |
| void insert(Object item, {int at: 0}) { ... } |

#### 2.不要使用显式默认值null

如果参数是可选的，但没有给它一个默认值，则语言隐式地使用null作为默认值，因此不需要编写它

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

不推荐如下写法:

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

### 八.变量

#### 1.不要显式地将变量初始化为空

在Dart中，未显式初始化的变量或字段自动被初始化为null。不要多余赋值null

|  |
| --- |
| int \_nextId;    class LazyId {  int \_id;    int get id {  if (\_nextId == null) \_nextId = 0;  if (\_id == null) \_id = \_nextId++;    return \_id;  }  } |

不推荐如下写法：

|  |
| --- |
| int \_nextId = null;    class LazyId {  int \_id = null;    int get id {  if (\_nextId == null) \_nextId = 0;  if (\_id == null) \_id = \_nextId++;    return \_id;  }  } |

#### 2.避免储存你能计算的东西

在设计类时，您通常希望将多个视图公开到相同的底层状态。通常你会看到在构造函数中计算所有视图的代码，然后存储它们:

应该避免的写法：

|  |
| --- |
| class Circle {  num radius;  num area;  num circumference;    Circle(num radius)  : radius = radius,  area = pi \* radius \* radius,  circumference = pi \* 2.0 \* radius;  } |

如上代码问题：

* 浪费内存
* 缓存的问题是无效——如何知道何时缓存过期需要重新计算？

推荐的写法如下：

|  |
| --- |
| class Circle {  num radius;    Circle(this.radius);    num get area => pi \* radius \* radius;  num get circumference => pi \* 2.0 \* radius;  } |

### 九.类成员

#### 1.不要把不必要地将字段包装在getter和setter中

不推荐如下写法：

|  |
| --- |
| class Box {  var \_contents;  get contents => \_contents;  set contents(value) {  \_contents = value;  }  } |

#### 2. 优先使用final字段来创建只读属性

|  |
| --- |
| StatelessWidget |

尤其对于

#### 3. 在不需要的时候不要用this

不推荐如下写法：

|  |
| --- |
| class Box {  var value;    void clear() {  this.update(null);  }    void update(value) {  this.value = value;  }  } |

推荐如下写法：

|  |
| --- |
| class Box {  var value;    void clear() {  update(null);  }    void update(value) {  this.value = value;  }  } |

### 十.构造函数

#### 1.尽可能使用初始化的形式

推荐如下写法：

|  |
| --- |
| class Point {  num x, y;  Point(this.x, this.y);  } |

不推荐如下写法：

|  |
| --- |
| class Point {  num x, y;  Point(num x, num y) {  this.x = x;  this.y = y;  }  } |

#### 2.不要使用new

Dart2使new 关键字可选

推荐写法：

|  |
| --- |
| Widget build(BuildContext context) {  return Row(  children: [  RaisedButton(  child: Text('Increment'),  ),  Text('Click!'),  ],  );  } |

不推荐如下写法：

|  |
| --- |
| Widget build(BuildContext context) {  return new Row(  children: [  new RaisedButton(  child: new Text('Increment'),  ),  new Text('Click!'),  ],  );  } |

### 十一.异步

#### 1.优先使用async/await代替原始的futures

async/await语法提高了可读性，允许你在异步代码中使用所有Dart控制流结构。

|  |
| --- |
| Future<int> countActivePlayers(String teamName) async {  try {  var team = await downloadTeam(teamName);  if (team == null) return 0;    var players = await team.roster;  return players.where((player) => player.isActive).length;  } catch (e) {  log.error(e);  return 0;  }  } |

#### 2.当异步没有任何用处时，不要使用它

如果可以在不改变函数行为的情况下省略异步，那么就这样做。、

|  |
| --- |
| Future afterTwoThings(Future first, Future second) {  return Future.wait([first, second]);  } |

不推荐写法：

|  |
| --- |
| Future afterTwoThings(Future first, Future second) async {  return Future.wait([first, second]);  } |

### 十二.参考资料

1. 《软件工程导论》 出版社：清华大学出版社 作者：张海藩、牟永敏 国际书号：9787302330981
2. Flutter Dart 代码开发规范

https://www.jianshu.com/p/ca2b4136e96f