Dart 基本语法及使用

DaZenD 关注

0.123 2021.10.15 17:16:19 字数 1,927 阅读 1,118

碎语，有android、iOS、前端、小程序、RN基础，所以，以下记录仅为个人理解记录，没必要的就忽略的

dart

一. dart 环境

需求：演练dart语法，直接运行.dart代码。。

```
$ dart xxx.dart
```

需要有dart环境支持

> 1 flutter中带的有sdk

安装flutter的时候，把dart的环境变量配置上即可

> 2 专门下载dart的sdk

https://dart.cn/get-dart

按文档安装dart sdk

1.1. vscode 演练dart

code runner 插件。

使用：右键-运行

- 也可以使用网页工具练习dart代码：https://dartpad.dartlang.org

二. 变量

跟ios像，优点是有类型推到，比如：var

dynamic 不建议使用

> 公私有性

dart中没有public，private，protected等，，

- 对象中，变量_field加下划线，就表示私有了。。只能当前文件访问

> 特殊 const

虽然js，oc里有，但是用法不一样，这个dart的还挺有意思

```
final a = const Person();
final b = const Person();
print(identical(a, b)); // true

final m = Person();
final n = Person();
print(identical(m, n)); // false
```

> late 关键字

https://blog.csdn.net/weixin_44239910/article/details/118196797

- 声明变量的时候，需要初始化，使用late关键字：显式声明一个非空的变量，但不初始化。。这样有风险
- 延迟初始化变量：有点像懒加载，用到的时候才会初始化

```
late String temperature = _readThermometer();
```

> dynamic

任意类型，是一个明确类型，比如

```
Map<String, dynamic> map

//这样是可以的
String str = map[''];
this.height = double.parse(map['height'].toString());
```

- 如果map中value类型是Object，那么就不能直接那样取，，dart没有 (String)obj这样的类型强转，，所以，可以使用其他方法
  - 使用dynamic，表示任意类型
  - 或者使用map['key'] as String 这种方式，，
  - int.parse(string)
  - int.toString()

注意：使用as语法，你要提前知道类型，如果类型不匹配，就报错。

所以，所使用的方法的类型匹配的前提下，可以转换类型的。。

三. 数据类型

> 布尔

注意: Dart中不能判断非0即真, 或者非空即真
Dart的类型安全性意味着您不能使用if(非booleanvalue)或assert(非booleanvalue)之类的代码。

> 字符串

三引号，可以换行的字符串。打印结果也是换行的

var str = """
ab
bc
cd
""";

字符串拼接：对象可以省略{}，表达式则不能省略{}

如： `print('my name is ${name}, age is $age, height is ${xiaoming.height}');`

> 集合类型

list，set，map 普通，没啥特别的

四. 函数

基本定义：

```
返回值 函数的名称(参数列表) {
  函数体
  return 返回值
}
```

- dart中没有关键字表示公私有性，使用'_'下划线标记私有。如：_method() 就是私有的。。
- dart中没有函数重载，也就是不允许有同名函数（同类中）

4.1. 函数的可选参数

> 位置可选参数

注意形参：要有默认值

```
void method1(String name, [int age = 1, int? h]) {
  print('name is $name, age is $age, h is $h');
}

调用：

method1('name', 2);

结果：

name is name, age is 2, h is null
```

> 命名可选参数

```
void method2(String name, {int? age, String? add}) {
  print('name is $name, age is $age, h is $add');
}

调用：
```

```
6
7  method1('name', 33);
8
9  结果:
10
11 name is name, age is 33, h is null
12
```

> 注意:

- 只有可选参数能有默认值

4.2. 函数是一等公民

```
1  void method3(Function func) {
2
3  }
```

> 匿名函数

```
1  method3(() {
2  });
```

> 箭头函数

```
1  method3(() => print("xxx"));
```

只有一行代码的时候，可以使用箭头函数

注意：其实不太对，实践：

```
1  method3(() => {
2    print("xxx"),
3    print('object')
4  });
5
6  这样是可以的，只是要注意箭头函数体内不能有分号
```

> 函数形参

可以使用功能typedefine提出来函数定义

五. 运算符

> ??=

```
1  var name = null;
2  name ??= 'jjjj';
```

变量没有值，才执行??=

> ??

```
var s = null;
var ss = s ?? 'ssss';
```

?? 前没有值则用??后面的值

5.1. 级联运算符

```
final p2 = Person()
              ..name = "why"
              ..run()
              ..eat()
              ..swim();
```

特定语法，记住就行

六. 类和对象

默认继承 Object..跟 android 一样

6.1. 构造方法

6.1.1. 命名构造函数

```
class Person {
  String name = '';
  int age = 0;
  double height = 1.8;

  Person(this.name, this.age);
  Person.externalWithHieght(this.name, this.age, this.height);
  Person.fromMap(Map<String, dynamic> map) {
    this.name = map['name'];
    this.age = map['age'];
    this.height = map['height'];
  }
}

调用:

Person person = Person.externalWithHieght('name', 19, 1.9);
person = Person.fromMap({'name':'leixing', 'age':19, 'height':1.99});
print(person.toString());
```

- 由于dart中函数是不能重载的，也就是不能有同名函数，所以，多构造函数可以通过.实现，如上。。
- 打印对象：结果是Instance of 'Person'。。所以，类中需要重写toString方法的。这个java一样

6.1.2. 类的初始化列表

```
class Person1 {
  late final String name;
  final int age;

  //other field 其他参数可选列表，用逗号隔开
  Person1(this.name, {int? age, other field}) : this.age = age ?? 10, other field {

  }

  //这里也可以实现外部传值给final的age，，但是注意：requered表示外部初始Person1的时候必须传age，就
  Person1(this.name, {required this.age}) {

  }

  //也可以这样，，但是注意：局限性是命名可选参数里不能写表达式，比如三目运算符
```

```
  Person1(this.name, {this.age = 10}) {

  }

  Person1.withName(this.name) : age = 2 {

  }
}
```

针对age这种变量，如果在声明的时候指定初始值，那么外部就没办法再初始化的时候后指定值了。。

初始化列表： 可以使用这种技术，初始化列表就支持在运行的时候赋值final变量

解析： `{int? age}` 是命名可选参数，后面的 `:xxx` 是初始化列表，这里是可以赋值的。。这样就解决了在方法体中对final变量赋值的动态需求

6.1.3. 重定向构造方法

```
class Person2 {
  String name;
  final int age;

  //重定向构造方法，用冒号，后面用this，表示前面已经初始化过了，后面可以使用this对象进行参数扩展
  //注意：重定向构造方法不能传可选参数。这是语法规定。可选参数只能供外部调用。内部调用只能调用最全的构
  //注意：形参要单纯，不能进行初始化，比如下面：只能 String name，而不能this.name。也就是说，不能先
  //The redirecting constructor can't have a field initializer.
  //重定向构造方法，很像开发java的时候，那种入参较多，构造方法较多的情况。。
  Person2(String name): this._internal(name);
  Person2._internal(this.name, {int?age}): this.age = age ?? 1;

  Person2._optionalAge(this.name, {this.age = 2});

  Person2.withName(String name): this._withNameAndAge(name, 1);
  Person2._withNameAndAge(this.name, this.age);

  @override
  String toString() {
    // TODO: implement toString
    return 'age is $age';
  }
}
```

6.1.4. 常量构造方法

```
main(List<String> args) {
  var p1 = const Person('why');
  var p2 = const Person('why');
  print(identical(p1, p2)); // true
}

class Person {
  final String name;

  const Person(this.name);
}

class Person3 {
  final String name;
  final int age;

  //注意：加了const，变量就必须是final的
  const Person3(String name): this._withAge(name, 1);
  const Person3._withAge(this.name, this.age);
}

const person3 = Person3._withAge('name', 222);
const person31 = Person3._withAge('name', 222);
print(identical(person3, person31));// true
```

常量构造函数，语法就是这么规定的，变量前加final，构造函数前加const

6.1.5. 工厂构造方法

```
main(List<String> args) {
  var p1 = Person('why');
  var p2 = Person('why');
  print(identical(p1, p2)); // true
}

class Person {
  String name;

  static final Map<String, Person> _cache = <String, Person>{};

  factory Person(String name) {
    if (_cache.containsKey(name)) {
      return _cache[name];
    } else {
      final p = Person._internal(name);
      _cache[name] = p;
      return p;
    }
  }

  Person._internal(this.name);
}
```

- 工厂构造函数最大的特点：可以手动返回一个对象。。必须用factory 修饰构造函数

6.1.6. setter和getter

```
class Person {
  String name = '';

  set setName(String name) {
    this.name = name;
  }

  set setName1(String name) => this.name = name;

  String get getName {
    return name;
  }
  //this写不写都可以，反正就是一个name变量，没有局部变量
  String get getName1 => this.name;
}
```

6.2. 类的继承

- 关键字 extend
- 单继承
- 跟重定向不同，继承，构造函数可以先初始化一部分，然后调用super中对应参数的构造函数

```
Person(String name, int age) : name=name, super(age);
```

6.3. 抽象类

- 关键字，和规则 跟其他语言一样
- 特殊点：

> 抽象类如何实现实例化

- 提供工厂构造方法

- erternal 关键字 可以将方法的声明和实现进行分离，@patch注解 进行方法的实现，可以让实际实例化代码实现不同的方法。跟多态差不多

6.4. 隐式接口

Dart中的接口比较特殊, 没有一个专门的关键字来声明接口.

默认情况下，定义的每个类都相当于默认也声明了一个接口，可以由其他的类来实现(因为Dart不支持多继承)

- 关键字：implements

6.5. Mixin混入

在通过implements实现某个类时，类中所有的方法都必须被重新实现(无论这个类原来是否已经实现过该方法)。

但是某些情况下，一个类可能希望直接复用之前类的原有实现方案，怎么做呢?

- 使用继承吗？但是Dart只支持单继承，那么意味着你只能复用一个类的实现。
  Dart提供了另外一种方案: Mixin混入的方式
- 除了可以通过class定义类之外，也可以通过mixin关键字来定义一个类。
  只是通过mixin定义的类用于被其他类混入使用，通过with关键字来进行混入。
- 如果方法重名：自己 > 混入 > 继承 > 接口

6.6. 类成员和方法

跟普通语言一致

七. 枚举类型

可读性

类型安全

八. 泛型

同java

九. 库的使用

默认：一个dart文件就是一个库文件

> 使用系统的库

核心库不需要导入：'dart:core'

非核心库，用到的会自动导入：'dart:math'

> 封装库

- as关键字给库起别名
- 默认是导入库的所有内容：
  - show：导入指定的内容
  - hide：隐藏某内容，导入除之之外的所有内容
- 导入多个库的时候，使用export，也就是把需要导入的库放到一个dart文件中统一导入

```
dart_util.dart:

export 'math_util.dart'
export 'date_util.dart'

引用:
import 'dart_util.dart'
```

- 通过'_'来区分公有和私有。变量和函数 都适用

> 三方库

三方库网站

1、创建 pubspec.yaml文件 //这个文件像podfile和gradle

2、依赖库

```
name: 一般是你工程名
description: A new Flutter project.
dependencies:
  http: 版本 // 去三方库网站 - 库 - installing - dependencies
```

3、加载依赖

cd到yaml文件路径：pub get

新生成的三个文件不用管：oubspec.lock .dart_tool .packages

4、引用库

import 'package:xxx'

三方库网站 - 库 - readme - using

DaZenD
总资产137 共写了8.2W字 获得160个赞 共56个粉丝

关注