一、Dart简介

Flutter使用Dart语言进行开发。

Dart是Google开发的计算机编程语言，可用于移动端、PC、Web、以及服务器开发，是一门全栈语言。

在不同的平台上，使用的Dart的框架也不同,如下：

平台	开发语言	框架
Flutter(Android/iOS/Linux/MacOS/Windows/Web)	Dart	Dart for the Flutter
Web	Dart	Dart for the web
服务器	Dart	Server-side Dart

其中移动端、PC以使用的是Dart for the Flutter

Web使用的是Dart for the web

服务器使用的是Server-side Dart

二、Dart历史

最早在2011年Dart语言就已经发布，而在2019年4月，Dart 2.3 版本发布

三、Dart语言优势

- Dart中所有的东西都是对象，包括数据、函数等，都是继承于Object，默认类型都是null
- Dart既支持JIT（动态编译），也支持AOT（静态编译）
- Dart是强类型语言，但是由于Dart可以推断类型，因此也支持动态类型，例如var、dynamic
- Dart有强大的异步编程能力

四、Dart用法

1、注解

（1）单行注解

//

（2）多行注解

/**

**/

2、外部库导入

import 'XXXXXX';

例如：

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

3、类

在类的定义和j继承上基本和java一致

（1）类的定义

使用class 定义类

如下：

```
class MyApp extends StatelessWidget {
 }
```

（2）类的实例化

在Dart中类的实例化不需要使用new关键字，直接类名（）;即可

例如：

```
MyApp app = MyApp();
```

（3）类的继承

使用extends来继承父类

4、权限修饰

Dart中没有public、protected、private和default关键字，类或者成员变量默认就是public的，如果需要表示私有，则在类名或者变量名前加_，

例如：

```
String _name = "by 小德";
```

这里表示name变量是私有的

5、Dart的数据类型

	含义	使用
int	整数，范围为 -2^63 到 2^63 - 1.	int x = 1;//没有小数点就是int
double	浮点数，64位（双精度）浮点数	double y = 1.1;//有小数点就是浮点数

num	num 是数字类型，既可以表示整数，也可以表示浮点数，具体看赋的值	num x = 1;//num x是整数
		num y = 1.1;//num y是浮点数

String	字符串 Dart字符串采用UTF-16编码 可以使用单引号或双引号来创建字符串	var s1 = 'string'; var s2 = "string";
bool	布尔值	var isTrue = true;
List	List<E> E 表示 List 里的数据类型 用中括号来赋值	List<int> list = [1, 2, 3];
Set	Set<E> E 表示 Set 里的数据类型 用大括号来赋值	Set<String> halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};

Map	Map<K, V>
	K 是 Key 的数据类型,V是 Value 的数据类型
		Map<String,String> gifts = {
		// Key: Value 格式
		'first': 'partridge',

		'second': 'turtledoves', 'fifth': 'golden rings'};
Runes	表示采用 UTF-32 的字符串，用于显示Unicode 因为Dart字符串是UTF-16，因此在Dart中表示32位的Unicode值需要Runes这个特殊语法。	Runes input = new Runes('\u{1f600}'); print(new String.fromCharCodes(input)); 打印出来的是笑脸emoji：😆

(1) 整数

整数只有int一个类型，取值 -2^63 到 2^63 - 1，对应java中的int取值-2^31,2^31-1，long取值为： -2^63 到 2^63 - 1。

即int类型包含了byte、short、int、long四种范畴

(2) 浮点数

浮点数只有double一个类型，相当于java中的float和double

(3) 数字类型

num表示数字类型，可以表示int和double两个类型，例如num x = 1;表示整数，num y = 1.1;表示浮点数

(4) 字符串

String类型表示字符串，使用utf-16编码，可以使用“ ”或者 ‘ ’ 表示

```
var s1 = 'string';
var s2 = "string";
```

(5) 布尔值

bool用来表示布尔值，这里和java不同，java中使用boolean表示布尔值

例如：

```
bool isTrue = true;
```

(6) List数组

List用来表示数组和列表，使用[ ]来赋值

例如：

```
List<int> list = [1,2,3];
```

(7) Map

Map用来表示Key-Value键值对，通过{表示}

```
Map<String,String> strs = {
 'first':'partridge',
 'second':'turtledoves'
}
```

(8) Runes

使用Runes来表示utf-32字符串，常见的例如 表情：

```
Runes input = new Runes('\u{1f600}');
print(new String.fromCharCodes(input));
```

打印出来的是笑脸emoji：😆

6、变量声明

（1）var

使用var声明变量，不需要指定变量的数据类型，Dart会自动推断类型，因此可以使用var来声明任何变量

原因是var只是存储了对象的引用，因此可以定义任何变量

例如：

```
var name = 'test'
var a = 1
```

（2）使用明确的数据类型声明变量

```
String name = '小七';
int count= 0;
```

（3）dynamic

dynamic比较特别，指定的数据类型是可变的，即可以首先指定为string，然后改为int型。例如

```
dynamic e = 'example';
e = 1;
```

（4）Object

Dart中所有的东西都是对象，都是继承于Object，因此可以定义任何变量，然后赋值。同样，object指定的类型也可以改变，这一点类似于dynamic·

一般来说我们使用Object代替dynamic来表示可变类型，但是在Native与Flutter交互时，通常用dynamic来表示从native传回的数据

7、常量声明

（1）final

可以使用final来修饰常量，而在修饰常量时可以省略var，同时声明常量时就要赋值，并且赋值后不能再赋值。

例如：

```
final int a = 1;
```

（2）const

const也可以修饰常量，在修饰类里的变量时需要加static，而在修饰全局变量时则可以省略static。

例如：

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

```
2
3 const demoConst = 'demo'; // 这里不用加 static
4
5 void main() => runApp(MyApp());
6
7 class MyApp extends StatelessWidget {
8
9  static content = 'Dart 语法'; // 这里必须加 static
10  ...
11 }
```

（3）fianl与const的区别

const是编译时常量，编译的时候就初始化了，而final则是当类创建对象时才初始化。

8、函数

Dart中函数也是一个对象，函数的对象类型是function

（1）函数格式

返回类型 函数名（函数参数）{

}

例如：

```
1 bool isVisible(bool visible){
2
3 }
```

（2）函数参数

Dart中的函数参数分为必须参数和可选参数两种

必选参数就是必须填写的参数，而可选参数则是选填的参数。

当一个函数中既有必选又有可选参数时，则必选参数在前，可选参数在后

（3）必选参数

必选参数就是类似于java中一般定义的参数，我们使用时为这些参数赋值

例如：

```
1 bool say(String msg , String from, int clock){
2  print(msg+" from " + from + " at " + clock?.toString());
3  return true;
4 }
```

（4）可选参数

可选参数又分为两种：

- 可选命名参数

使用{ }包起来

- 可选位置参数

使用[ ] 包起来

1> 可选命名参数

可选命名参数的类型是Map，所以使用{ }包起来

传入参数时要传入key - value对

例如：

```
bool say(String msg , {String from, int clock}){
 print(msg+" from " + from + " at " + clock.toString());
 return true;
}
```

使用时：

```
say('Hello Flutter');//✅ 因为 from 和 clock 是可选参数，所以可以不填

say('Hello Flutter',from: 'XiaoMing');//对 用命名参数格式 paramName: value 为 from 赋值
say('Hello Flutter',clock: 11);//✅
say('Hello Flutter',from: 'XiaoMing',clock: 11);//✅
```

2> 可选位置参数

可选位置参数类型是List，使用[ ] 来赋值，赋值时需要和参数一一对应，不能跳过中间的某一个可选参数

例如：

```
bool say(String msg , [String from , int clock]){
 print(msg+" from " + from + " at " + clock.toString());
 return true;
}
```

使用时：

```
say('Hello Flutter');//✅ 因为 from 和 clock 是可选参数，所以可以不填

say('Hello Flutter','XiaoMing',1);//✅ 为可选位置参数赋值，只能一个参数一个参数对应的赋值，所以要全部赋值

say('Hello Flutter','XiaoMing')//✅
say('Hello Flutter',1)//❌ 因为 1 赋值给了 from,但是 from 是String，所以会报错
```

3>可选参数默认值赋值

可选参数可以选择赋默认值，也可以不赋默认值，而赋默认值可以通过 = 来赋值

```
bool say(String msg , {String from = 'empty', int clock = 0}){
print(msg+" from " + from + " at " + clock.toString());
return true;
}
```

9、=>语法

在Dart中可以通过=>来表示Lambda表达式，但是需要注意，这里的=>后只能跟一行代码，这行代码只能是一个表达式。

例如：

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

等价于

```
void main(){
 return runApp(MyApp())
}
```

10、类

(1) 构造方法

类的默人的构造函数是使用类名作为函数名的构造函数

我们使用java的方式类写构造函数，如下：

```
class Point {
 num x, y;

 Point(num x, num y) {
 // There's a better way to do this, stay tuned.
 this.x = x;
 this.y = y;
 }
}
```

而在Dart中我们可以这样声明构造函数，并成员变量进行赋值

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

(2) Widget构造参数采用可选命名参数

由于Widget构造函数有很多参数，为了使用起来清晰，Widget采用可选命名参数

(3) 创建实例

不需要使用new

```
Point point = Point(0,0);
```

（4）使用类的变量和方法

和java类似，使用对象加 . 来引用实例变量和方法

11、操作符

（1）算数运算符

这一点基本和java一致

操作符	含义	例子
+	加	var a = 2 + 3;
-	减	var a = 2 - 3;
-exper	负数	var a = -1;
*	乘	var a = 2 * 3;
/	除，精确除法	var a = 5 / 2; // a的结果为 2.5
~/	整除	var a = 5 ~/ 2; // a的结果为 2
%	取余	var a = 5 % 2; // a的结果为 1
++var		var a = 1; var b = ++a;// b 的结果为 2,a 的结果为 2
var++		var a = 1; var b = a++;// b 的结果为 1,a 的结果为 2
--var		var a = 1; var b = --a;// b 的结果为 0,a 的结果为 0
var--		var a = 1; var b = a--;// b 的结果为 1,a 的结果为 0

（2）相等和大小操作

也和java类似

操作符	含义	例子
==	是否相等	assert(2 == 2);
!=	不等于	assert(2 != 3);
>		

	大于	assert(3 > 2);
<	小于	assert(2 < 3);
>=	大于等于	assert(3 >= 3);
<=	小于等于	assert(3 <= 3);

（3）类型判断符

这一点和java不同，需要特别注意

操作符	含义	例子
as	类型转换	(emp as Person).firstName = 'Bob';

is	判断是否是某个类型,如果是的话，就返回 true	if (emp is Person) {
		// 如果 emp 是 Person 类型
		emp.firstName ='Bob';
		}

is!	判断是否不是某个类型，如果不是的话，就返回 true	if (emp is! Person) {
		// 如果 emp 不是 Person 类型
		}

上面的例子中，如果 emp 是 null 的话，as 的例子就会抛异常，is 和 isn't 的例子会返回 false.

（4）赋值操作符

默认赋值是通过 = 来赋值，例如

```
var a = 1;
```

另外，当需要在某个对象为null时才进行赋值可以使用??=来进行 赋值

```
var a ??= 1;
```

（5）逻辑运算符

其中&&、||都和java一致，但是非的条件不同，dart中使用!expr来表示非

操作符	含义	例子
!expr	反转表达式（将 false 改为 true，反之亦然）	!(2 == 3);// 结果为 true
||	逻辑或	(2 == 2) || (2 == 3);// 结果为 true
&&	逻辑与	（2 == 2）&& (3 == 3);// 结果为 true

（6）按位与移位符

操作符	含义	例子
&	按位与	final value = 0x22;

	对于每一个比特位，只有两个操作数相应的比特位都是1时，结果才为1，否则为0。	final bitmask = 0x0f;
		var result = value &
		bitmask;//结果为 0x02

`|`	按位或，对于每一个比特位，当两个操作数相应的比特位至少有一个1时，结果为1，否则为0。	final value = 0x22;
		final bitmask = 0x0f;
		var result = value

`^`	按位异或，对于每一个比特位，当两个操作数相应的比特位有且只有一个1时，结果为1，否则为0。	final value = 0x22;
		final bitmask = 0x0f;
		var result = value ^
		bitmask;//结果为 0x2d

`~expr`	按位非，反转操作数的比特位，即0变成1，1变成0。	final value = 0x22;
		final bitmask = 0x0f;
		var result = value &
		~bitmask;//结果为 0x20

`<<`	左移	final value = 0x22;
		final bitmask = 0x0f;
		var result = value << 4;//结果为 0x220

`>>`	右移	final bitmask = 0x0f; var result = value >> 4;//结果为 0x02

(7) 条件运算符

1>、condition ? expr1 : expr2

简化if else 的写法，和java类似

```
var visibility = isPublic ? 'public' : 'private';
```

2> expr1 ?? expr2

如果 expr1 为 null，就返回 expr2 的值，否则返回 expr1 的值。

```
String playerName(String name) => name ?? 'Guest';
```

(8) 级联操作符

通过..来进行级联操作，即可以在一个对象上进行一系列的操作

```
querySelector('#confirm') // Get an object.
  ..text = 'Confirm' // Use its members.
  ..classes.add('important')
  ..onClick.listen((e) => window.alert('Confirmed!'));
```

可以看到，这里通过..连续访问该对象上的方法

(9) 其他操作符

操作符	含义	例子
`()`	函数调用	代表函数调用
`[]`	访问列表	引用列表中指定索引处的值
`.`	访问成员变量	访问表达式里的成员变量，例如 `foo.bar`,表示访问 `foo` 表达式里的 `bar` 成员变量

?.	有条件的成员变量访问	很像 .，但是左边的表达式可以为 null，例如 foo?.bar，如果 foo 为 null，则不会抛异常，而是返回 null，如果 foo 不为 null，则可以返回 bar

1> ?.

通过?.来省略多余判空处理，如果为null，则赋默认值，也就是null

```
var yourName = user?.name;
等价于
var yourName;
if(user == null){
 yourName = null;
}else{
 yourName = user.name;
}
```

2> ??

在赋值时，如果发现为null，则为其赋我们设定的值

```
var yourName = name ?? "Bob";
等价于
var yourName;
if(name == null){
 yourName = "Bob";
}else{
 yourName = name;
}
```

3> ??=

expr1 ??= expr2等价于expr1 = expr1 ?? expr2

例如：

```
user ??= User();
等价于
if(user == null) {
 user = User();
}
```