**1.1. 数字类型**

对于数值来说，我们也不用关心它是否有符号，以及数据的宽度和精度等问题。只要记着整数用int，浮点数用double就行了。

不过，要说明一下的是Dart中的int和double可表示的范围并不是固定的，它取决于运行Dart的平台。

// 1.整数类型int

int age = 18;

int hexAge = 0x12;

print(age);

print(hexAge);

// 2.浮点类型double

double height = 1.88;

print(height);

字符串和数字之间的转化:

// 字符串和数字转化

// 1.字符串转数字

var one = int.parse('111');

var two = double.parse('12.22');

print('${one} ${one.runtimeType}'); // 111 int

print('${two} ${two.runtimeType}'); // 12.22 double

// 2.数字转字符串

var num1 = 123;

var num2 = 123.456;

var num1Str = num1.toString();

var num2Str = num2.toString();

var num2StrD = num2.toStringAsFixed(2); // 保留两位小数

print('${num1Str} ${num1Str.runtimeType}'); // 123 String

print('${num2Str} ${num2Str.runtimeType}'); // 123.456 String

print('${num2StrD} ${num2StrD.runtimeType}'); // 123.46 String

**1.2. 布尔类型**

布尔类型中,Dart提供了一个bool的类型, 取值为true和false

// 布尔类型

var isFlag = true;

print('$isFlag ${isFlag.runtimeType}');

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。

var message = 'Hello Dart';

// 错误的写法

if (message) {

print(message)

}

**1.3. 字符串类型**

Dart字符串是UTF-16编码单元的序列。您可以使用单引号或双引号创建一个字符串:

// 1.定义字符串的方式

var s1 = 'Hello World';

var s2 = "Hello Dart";

var s3 = 'Hello\'Fullter';

var s4 = "Hello'Fullter";

可以使用三个单引号或者双引号表示多行字符串:

// 2.表示多行字符串的方式

var message1 = '''

哈哈哈

呵呵呵

嘿嘿嘿''';

字符串和其他变量或表达式拼接: 使用${expression}, 如果表达式是一个标识符, 那么{}可以省略

// 3.拼接其他变量

var name = 'coderwhy';

var age = 18;

var height = 1.88;

print('my name is ${name}, age is $age, height is $height');

**1.4. 集合类型**

**1.4.1. 集合类型的定义**

对于集合类型，Dart则内置了最常用的三种：List / Set / Map。

其中，List可以这样来定义：

// List定义

// 1.使用类型推导定义

var letters = ['a', 'b', 'c', 'd'];

print('$letters ${letters.runtimeType}');

// 2.明确指定类型

List<int> numbers = [1, 2, 3, 4];

print('$numbers ${numbers.runtimeType}');

其中，set可以这样来定义：

* 其实，也就是把[]换成{}就好了。
* Set和List最大的两个不同就是：Set是无序的，并且元素是不重复的。

// Set的定义

// 1.使用类型推导定义

var lettersSet = {'a', 'b', 'c', 'd'};

print('$lettersSet ${lettersSet.runtimeType}');

// 2.明确指定类型

Set<int> numbersSet = {1, 2, 3, 4};

print('$numbersSet ${numbersSet.runtimeType}');

最后，Map是我们常说的字典类型，它的定义是这样的：

// Map的定义

// 1.使用类型推导定义

var infoMap1 = {'name': 'why', 'age': 18};

print('$infoMap1 ${infoMap1.runtimeType}');

// 2.明确指定类型

Map<String, Object> infoMap2 = {'height': 1.88, 'address': '北京市'};

print('$infoMap2 ${infoMap2.runtimeType}');

**1.4.2. 集合的常见操作**

了解了这三个集合的定义方式之后，我们来看一些最基础的公共操作

第一类，是所有集合都支持的获取长度的属性length：

// 获取集合的长度

print(letters.length);

print(lettersSet.length);

print(infoMap1.length);

第二类, 是添加/删除/包含操作

* 并且，对List来说，由于元素是有序的，它还提供了一个删除指定索引位置上元素的方法

// 添加/删除/包含元素

numbers.add(5);

numbersSet.add(5);

print('$numbers $numbersSet');

numbers.remove(1);

numbersSet.remove(1);

print('$numbers $numbersSet');

print(numbers.contains(2));

print(numbersSet.contains(2));

// List根据index删除元素

numbers.removeAt(3);

print('$numbers');

第三类，是Map的操作

* 由于它有key和value，因此无论是读取值，还是操作，都要明确是基于key的，还是基于value的，或者是基于key/value对的。

// Map的操作

// 1.根据key获取value

print(infoMap1['name']); // why

// 2.获取所有的entries

print('${infoMap1.entries} ${infoMap1.entries.runtimeType}'); // (MapEntry(name: why), MapEntry(age: 18)) MappedIterable<String, MapEntry<String, Object>>

// 3.获取所有的keys

print('${infoMap1.keys} ${infoMap1.keys.runtimeType}'); // (name, age) \_CompactIterable<String>

// 4.获取所有的values

print('${infoMap1.values} ${infoMap1.values.runtimeType}'); // (why, 18) \_CompactIterable<Object>

// 5.判断是否包含某个key或者value

print('${infoMap1.containsKey('age')} ${infoMap1.containsValue(18)}'); // true true

// 6.根据key删除元素

infoMap1.remove('age');

print('${infoMap1}'); // {name: why}