Flutter中Map、List数组的常用方法

ailinghao

Dart语言是Flutter开发的必备语言，官方地址如下：Dart：https://dart.cn/

使用dart工具来运行这些常用的方法，工具：https://dartpad.cn

在Dart中，Map用来存储对像类型的数据，List与Set用来存储数组类型的数据。

Dart中的Map用来保存key-value键值对的数据集合

Map的创建实例如下：

```
// 创建一个Map实例，默认实现是LinkedHashMap。

Map()
// 创建一个LinkedHashMap实例，包含other中的所有键值对。
Map.from(Map other)
// 创建一个Map实例，其中Key和Value由iterable的元素计算得到。
Map.fromIterable(Iterable iterable, {K key(element), V value(element)})
 // 将指定的keys和values关联，创建一个Map实例。
Map.fromIterables(Iterable<K> keys, Iterable<V> values)
 // 使用默认实现LinkedHashMap创建一个严格的Map。
Map.identity()
 // 创建一个不可修改、基于哈希值的Map，包含other所有的项
Map.unmodifiable(Map other)
```

建有一个有初始值的Map，代码如下：

```
  // 根据一个Map创建一个新的Map，插入顺序进行排列
  var dic1 = new Map.from({'name': '张三'});
```

```
  print(dic1);  // {name: 张三}
  // 根据List创建Map, 插入顺序进行排列
  List<int> list = [1, 2, 3];
  // 使用默认方式, key和value都是数组对应的元素
  var dic2 = new Map.fromIterable(list);
  print(dic2);  // {1: 1, 2: 2, 3: 3}
  // 设置key和value的值
  var dic3 = new Map.fromIterable(list, key: (item)
=> item.toString(), value: (item) => item * item);
  print(dic3);  // {1: 1, 2: 4, 3: 9}
  // 创建一个不可修改、基于哈希值的Map
  var dic6 = new Map.unmodifiable({'name': 张三});
  print(dic6); // {name: 张三}
```

根据List数据来创建Map,代码如下：

```
  // 两个数组映射一个字典, 插入顺序进行排列
  List<String> keys = ['name', 'age'];

  var values = [张三, 20];
  // 如果有相同的key值, 后面的值会覆盖前面的值
  var dic4 = new Map.fromIterables(keys, values);

  print(dic4);  // {name: 张三, age: 20}
var words = <String>['fee', 'fi', 'fo', 'fum'];
var map = words.asMap();  // {0: fee, 1: fi, 2: fo,
3: fum}
map.keys.toList(); // [0, 1, 2, 3]
```

set

Set中的元素是不可重复的,用{}声明Set，并用,分割元素。

Set集合可直接对String、int、double类型去重。

```
var set1 = {1,2,3,4,5,6}; // 声明一个 set 并赋初始元素
var set2 = <Int>{}; // 声明一个空 set
var set3 = new Set(); // 声明一个空 set
var set4 = Set(); // 声明一个空 set, new 关键词可有可无
Set set=new Set();
set.add(1);
```

```
set.add(2);
set.add(1);
set.add(1);
set.add(3);
Set set=new Set();

set.add('a');
set.add('b');
set.add('');
set.add('c');
set.add('a');
set.add('b');
print(s);   // {a, b,,c}
```

List数组的常用方法

定义固定长度数组

```
void main() {  var list = List(2);
print('$list'); // [null, null]}
```

定义混合类型数组

```
void main() {
 var list = List<dynamic>();
  list.add('我是文本');
  list.add(0.66);
 print(list); // [我是文本, 0.66]}
```

判断数组内是否有满足条件的元素- any

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];// 数组中是否有大于3的元素
  print(list.any((v) => v > 3)); // true
// 数组中是否有大于7的元素
  print(list.any((v) => v > 7)); // false
}
```

判断数组所有元素是否都满足设定条件 - every

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];
// 数组中所有元素是否都大于0
  print(list.every((v) => v > 0)); // true
```

```
// 数组中所有元素是否都大于5
  print(list.every((v) => v > 5)); // false
}
```

将数组用指定字符拼接成字符串 – join

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];
  // 将数组转换为用英文逗号拼接的字符串
  print(list.join(','));  // 1,2,2,3,4,5,6,6

}
```

将字符串以指定字符分割成数组–split

```
void main() {
  String str = '1,2,2,3,4,5,6,6';
  print(str.split(','));  // [1,2,2,3,4,5,6,6]
}
```

数组去重 – toSet

```
void main() {
var list = [1, 2, 2, 3, 4, 5, 6, 6];
print(list.toSet());  // {1, 2, 3, 4, 5, 6}
}
```

数组取指定个数的数组– take

```
void main() {
  List latest = [1, 2, 3, 4, 5, 6, 7, 8, 9, 2, 3,
4];
  List list = latest.take(5).toList();
  print('value: $latest  list:$list');
}
打印: value: [1, 2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4]
list:[1, 2, 3, 4, 5]
```

按指定条件返回 – map

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];  // 将list所
```

```
有元素加1并返回数组
  var v = list.map((e) {
    return e + 1;
}).toList();
 print(v); //[2, 3, 3, 4, 5, 6, 7, 7]
}
```

数组遍历 – for\for in\forEach

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];
 //for
  for (var i = 0; i < list.length; i++) {
    print("for:$i");
}
  //for in
  for (var item in list) {
    print("for in:$item");
}
  //forEach
  list.forEach((element) {
    print("forEach:$element");
  });
//while+iterator迭代器遍历，类似Java中的iteator
while(list.iterator.moveNext()) {
  //获取对应的值
  var value = list.iterator.current;
 print("for:$value");
}
}
```

累加 – reduce

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];// 将每次返回值
作为value循环执行。最终返回最后一次执行值
  var count = list.reduce((value, element) {
    print('value: $value - element: $element');
    /**    每次的执行结果
    value: 1 - element: 2
    value: 3 - element: 2
   value: 5 - element: 3
```

```
   value: 8 - element: 4
   value: 12 - element: 5
   value: 17 - element: 6
    value: 23 - element: 6

 */

   return value + element;
 });
 print('count: $count'); // count: 29
}
```

排序 – sort

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];
// a - b 为升序,  b - a为降序
  list.sort((a, b) {    return b - a;  });
  print(list); //[6, 6, 5, 4, 3, 2, 2, 1]
}
```

获取满足条件的元素 – where

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];
  // 获取所有大于3的元素
  print(list.where((v) => v > 3).toList()); //[4,
5, 6, 6]
}
```

获取满足条件的第一个元素 – firstWhere

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];  // 获取最后
一个大于3的元素
  print(list.firstWhere((v) => v > 3)); // 4
  // 如果未查找到所制定条件的元素，进入orElse参数
 list.firstWhere((v) => v > 6, orElse: () {
    print(888);
```

```
  });
}
```

获取满足条件的最后一个元素 - lastWhere （与firstWhere同理，第一个与最后一个的区别）

删除满足条件的元素 - removeWhere

```
  /// final numbers = <String>['one', 'two', 'three', 'four'];
  /// numbers.removeWhere((item) => item.length == 3);
  /// print(numbers); // [three, four]
```

从指定位置开始，获取满足条件的第一个元素的索引 - indexWhere

获取满足条件的最后一个元素的索引(倒叙查询) - lastIndexWhere（与indexWhere同理，第一个与最后一个的区别）

从指定位置开始，获取指定值的索引 - indexOf

从指定位置开始，倒叙获取指定值的索引 - lastIndexOf（与indexOf同理，第一次与最后一次的区别）

批量添加 - addAll或者 扩展操作符（...）和 空感知扩展操作符（...?）

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 6, 6];
  var list2 = [0, 20, 40];
  list.addAll(list2);
  print(list); //[1, 2, 2, 3, 4, 5, 6, 6, 0, 20, 40]
}
```

```
//或使用扩展操作符，结果是一样的
```

```
void main() {
  var list2 = [0, 20, 40];
  var list = [1, 2, 2, 3, 4, 5, 6, 6, ...?list2];
  print(list); //[1, 2, 2, 3, 4, 5, 6, 6, 0, 20,
```

```
40]
}
```

获取倒序迭代器 – reversed.注意:翻转过后的数组，要用toList方法，才能成为一个新的数组

```
void main() {
   var list = [1, 2, 2, 3, 4, 5, 6, 6];
  print(list.reversed); // (6, 6, 5, 4, 3, 2, 2, 1)
print(list.reversed.toList()); // [6, 6, 5, 4, 3,
2, 2, 1]
}
```

生成值列表：List.generate

```
List<E>.generate(
int length,
E generator(int index),
{bool growable: true}
)
```

例子:

```
Row(
                  children: List.generate(
                    10,
                        (index) => Container(
                      width: 200,
                      height: 200,
                      margin: EdgeInsets.all(16),
                      color: Colors.grey[300],
                      child: Center(
                        child: Text(
                          'Inner $index',
                          style:
TextStyle(fontSize: 24),
                        ),
                      ),
                    ),
                  ),
                )
```

List.from 和 .of 之间以及 Map.from 和 .of 之间的区别

from 和 of 方法的重要区别是后者有类型注解，前者有类型注解不要.由于 Dart 泛型被具体化并且 Dart 2 是强类型的，因此这是确保 List/Map 正确构造的关键

```
var foo = new List.from(<int>[1, 2, 3]); // List<dynamic>
var bar = new List.of(<int>[1, 2, 3]); // List<int>
```

有条件的向列表中插入内容

```
var list = [
    'a',
    'b',
    if (isAdd) 'c'
];//isAdd 为 true 则 list 中包含'c'，否则就不包含
var list = [1,2,3];
var list2 = [
    '0',
    for (var i in list) '$i'
];// list2 中包含 0,1,2,3
```

常用属性：

1.add 增加
2.addAll 拼接数组
3.indexOf 查找 传入具体值
4.remove 删除 传入具体值
5.removeAt 删除 传入索引值
6.fillRange 修改
7.insert(index,value) 指定位置插入
8.insertAll(index,value) 指定位置插入数组
9.toList 其他类型转换为List类型
10.first 获取数组第一个元素　last最后一个