Flutter 基礎 6 - Dart Lists (Basics, Access, Update, Iterate)

參考資料 + 翻轉教室輔助影片

Flutter 基礎與實作 6 - Dart Lists (Basics, Access, Update, Iterate)

APP 程式設計實務 (2021 Fall)

https://youtu.be/CVwgf72cxhc

1. Dart/Flutter List Tutorial with Examples - BezKoder
2. List class - dart:core library - Dart API

課程設計

Fixed-length list

```
void main() {
  List<int> dat1  = List<int>.filled(5,0);
  List<int?> dat2 = List<int?>.filled(5,null);

  dat2[0] = 3;
  dat2[2] = 6;
  dat2[4] = 9;

  print("dat1 = $dat1");
  print("dat2 = $dat2");
}
```

Console

```
dat1 = [0, 0, 0, 0, 0]
dat2 = [3, null, 6, null, 9]
```

```
void main() {
  var dat1 = List<dynamic>.filled(5, null);

  for(var i=0; i<10; i++) {
    dat1[i] = i;
  }
  print("dat1 = $dat1");
}
```

Run

Console

```
Uncaught Error: RangeError (index):
Index out of range: index should be
less than 5: 5
```

```
void main() {
  List<int> dat1 = List<int>.filled(5,0);
  print("dat1 = $dat1");

  for(var i=0; i<10; i++) {
    dat1.add(i*i);
  }

  print("dat1 = $dat1");
}
```

Run

Console

```
dat1 = [0, 0, 0, 0, 0]
Uncaught Error: Unsupported operation: add
```

Shallow Copy!

```
void main() {
  var dat1 = List<dynamic>.filled(5, null);

  dat1[0] = 1;
  dat1[2] = "hello";
  dat1[4] = 4.5;

  var dat2 = dat1;

  print("dat1 = $dat1");
  print("dat2 = $dat2");
  print("-"*36);
  dat1[4] = "world";
  print("dat1 = $dat1");
  print("dat2 = $dat2");
}
```

Console

```
dat1 = [1, null, hello, null, 4.5]
dat2 = [1, null, hello, null, 4.5]
------------------------------------
dat1 = [1, null, hello, null, world]
dat2 = [1, null, hello, null, world]
```

Growable List

```
void main() {
  List<int> dat1 = [];
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");

  for (var i = 0; i < 10; i++) {
    dat1.add(i * i);
  }
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
}
```

Console

```
dat1 = []
dat1.length = 0
dat1 = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
dat1.length = 10
```

```
void main() {
  List<int> dat1  = List<int>.filled(5,0, growable:true);
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
```

```
  dat1.add(1);
  dat1.add(2);
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
}
```

```
Console

dat1 = [0, 0, 0, 0, 0]
dat1.length = 5
dat1 = [0, 0, 0, 0, 0, 1, 2]
dat1.length = 7
```

Initialize List with values

```
void main() {
  List<int> dat1  = [1,2,3];
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
  dat1.add(4);
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
  dat1.remove(2);
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
}
```

```
Console

dat1 = [1, 2, 3]
dat1.length = 3
dat1 = [1, 2, 3, 4]
dat1.length = 4
dat1 = [1, 3, 4]
dat1.length = 3
```

```
1 void main() {
2   List<int> dat1  = List.unmodifiable([1,2,3]);
3   print("dat1 = $dat1");
4   print("dat1.length = ${dat1.length}");
5   dat1[2] = 5;
6 }
```

▶ Run

```
Console

dat1 = [1, 2, 3]
dat1.length = 3
Uncaught Error: Unsupported operation: indexed set
```

```
1 void main() {
2   List<int> dat1  = List.unmodifiable([1,2,3]);
3   print("dat1 = $dat1");
4   print("dat1.length = ${dat1.length}");
5   dat1.add(4);
6 }
```

▶ Run

```
Console

dat1 = [1, 2, 3]
dat1.length = 3
Uncaught Error: Unsupported operation: add
```

Initialize List with `generate` constructor

```
void main() {
  List<int> dat1  = List.generate(5,(index) => index*index);
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
  dat1.addAll([1,2,3]);
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
}
```

```
Console

dat1 = [0, 1, 4, 9, 16]
dat1.length = 5
dat1 = [0, 1, 4, 9, 16, 1, 2, 3]
dat1.length = 8
```

Concatenate Lists

```
void dump(var a, var b, var c, var d, var e) {
  print("a = $a\nb = $b\nc = $c");
  print("d = $d\ne = $e\n");
}

void main() {
```

```
  List<int> a  = List.generate(5,(index) => index);
  List<int> b  = List.generate(5,(index) => index*2);
  List<int> c  = List.generate(5,(index) => index*3);
  List d = [];
  List e = [];
  dump(a,b,c,d,e);

  d = List.from(a);
  d.addAll(b);
  d.addAll(c);
  dump(a,b,c,d,e);

  e = List.from(a)..addAll(b)..addAll(c);
  dump(a,b,c,d,e);
}
```

Console

```
a = [0, 1, 2, 3, 4]
b = [0, 2, 4, 6, 8]
c = [0, 3, 6, 9, 12]
d = []
e = []

a = [0, 1, 2, 3, 4]
b = [0, 2, 4, 6, 8]
c = [0, 3, 6, 9, 12]
d = [0, 1, 2, 3, 4, 0, 2, 4, 6, 8, 0, 3, 6, 9, 12]
e = []

a = [0, 1, 2, 3, 4]
b = [0, 2, 4, 6, 8]
c = [0, 3, 6, 9, 12]
d = [0, 1, 2, 3, 4, 0, 2, 4, 6, 8, 0, 3, 6, 9, 12]
e = [0, 1, 2, 3, 4, 0, 2, 4, 6, 8, 0, 3, 6, 9, 12]
```

Spread Operator (...)

```
void dump(var a, var b, var c, var d, var e) {
  print("a = $a\nb = $b\nc = $c");
  print("d = $d\ne = $e\n");
}

void main() {
  List<int> a  = List.generate(5,(index) => index);
  List<int> b  = List.generate(5,(index) => index*2);
  List<int> c  = List.generate(5,(index) => index*3);
  List d = [...a, ...b, ...c];
  List e = a + b + c;

  dump(a,b,c,d,e);
}
```

Console

```
a = [0, 1, 2, 3, 4]
b = [0, 2, 4, 6, 8]
c = [0, 3, 6, 9, 12]
d = [0, 1, 2, 3, 4, 0, 2, 4, 6, 8, 0, 3, 6, 9, 12]
e = [0, 1, 2, 3, 4, 0, 2, 4, 6, 8, 0, 3, 6, 9, 12]
```

Access items from List

```
void main() {
  List<int> dat1 = List.generate(5, (index) => index);
  print("dat1 = $dat1");
  print("dat1.length = ${dat1.length}");
  print("dat1[2] = ${dat1[2]}\n");

  var slice = dat1.getRange(1, 3).toList();
  print("\ndat1.getRange(1,3).toList() = $slice");

  slice = List.from(dat1.take(3));
  print("\nList.from(dat1.take(3)) = $slice");
}
```

Console

```
dat1 = [0, 1, 2, 3, 4]
dat1.length = 5
dat1[2] = 2


dat1.getRange(1,3).toList() = [1, 2]

List.from(dat1.take(3)) = [0, 1, 2]
```

Remove items from List

```
void main() {
  List<int> dat1 = [7, 54, 80, 1, 93, 64, 65, 28, 37, 2];
  print("dat1 = $dat1\n");
  dat1.removeAt(3);
  print("dat1.removeAt(3)\ndat1 = $dat1\n");
  dat1.remove(80);
  print("dat1.remove(80)\ndat1 = $dat1\n");
  dat1.removeWhere((x) => x%2==1);
```

```
  print("dat1.removeWhere((x) => x%2==1)\ndat1 = $dat1\n");
  dat1.removeRange(1,3);
  print("dat1.removeRange(1,3)\ndat1 = $dat1\n");
  dat1.clear();
  print("dat1.clear()\ndat1 = $dat1");
}
```

Console

```
dat1 = [7, 54, 80, 1, 93, 64, 65, 28, 37, 2]

dat1.removeAt(3)
dat1 = [7, 54, 80, 93, 64, 65, 28, 37, 2]

dat1.remove(80)
dat1 = [7, 54, 93, 64, 65, 28, 37, 2]

dat1.removeWhere((x) => x%2==1)
dat1 = [54, 64, 28, 2]

dat1.removeRange(1,3)
dat1 = [54, 2]

dat1.clear()
dat1 = []
```

Update List item

```
void main() {
  List<int> dat1 = [7, 54, 80, 1, 93, 64, 65, 28, 37, 2];
  print("dat1 = $dat1\n");
  dat1.replaceRange(1,2,[88]);
  print("dat1.replaceRange(1,2,[88])\ndat1 = $dat1\n");
  dat1.replaceRange(2,3,[77,66]);
  print("dat1.replaceRange(2,3,[77,66])\ndat1 = $dat1\n");
  dat1.replaceRange(0,4,[11]);
  print("dat1.replaceRange(0,4,[11])\ndat1 = $dat1");
}
```

Console

```
dat1 = [7, 54, 80, 1, 93, 64, 65, 28, 37, 2]

dat1.replaceRange(1,2,[88])
dat1 = [7, 88, 80, 1, 93, 64, 65, 28, 37, 2]

dat1.replaceRange(2,3,[77,66])
dat1 = [7, 88, 77, 66, 1, 93, 64, 65, 28, 37, 2]

dat1.replaceRange(0,4,[11])
dat1 = [11, 1, 93, 64, 65, 28, 37, 2]
```

Iterate Over A List

```
void main() {
  List<int> dat1 = [7, 54, 80, 1, 93, 64];
  for (var i = 0; i < dat1.length; i++) {
    print(dat1[i]);
  }
}
```

```
void main() {
  List<int> dat1 = [7, 54, 80, 1, 93, 64];
  for(var x in dat1){
    print(x);
  }
}
```

Iterable objects

```
void main() {
  List<int> dat1 = [7, 54, 80, 1, 93, 64];
  dat1.forEach((x) => print(x));
}
```

```
void main() {
  List<int> dat1 = [7, 54, 80, 1, 93, 64];
  dat1.forEach(print);
}
```

style recommended style flutter

AVOID using **forEach** with a function literal.

BAD:

```
people.forEach((person) {
  ...
});
```

GOOD:

```
for (var person in people) {
  ...
}

people.forEach(print);
```

Iterator

```
void main() {
  List<int> dat1 = [7, 54, 80, 1, 93, 64];
  var ite = dat1.iterator;
  while(ite.moveNext()) {
    print(ite.current);
  }
}
```

Documentation

`bool moveNext()`

Advances the iterator to the next element of the iteration.

Should be called before reading `current`. If the call to `moveNext` returns `true`, then `current` will contain the next element of the iteration until `moveNext` is called again. If the call returns `false`, there are no further elements and `current` should not be used any more.

It is safe to call `moveNext` after it has already returned `false`, but it must keep returning `false` and not have any other effect.

更詳細的操作說明，請參閱『翻轉教室輔助影片』...