Flutter 基礎 4 - Dart Null Aware Operators

參考資料 + 翻轉教室輔助影片

Flutter 基礎與實作 4 - Dart Null Aware Operators

APP 程式設計實務 (2021 Fall)

https://youtu.be/qh0t-Hx9S2Q

Home

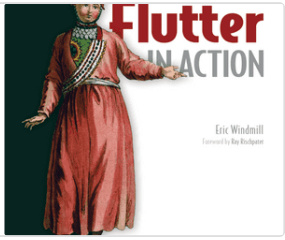
Tutorials about Dart and Flutter.

https://flutterbyexample.com/

Null Aware Operators

The following is an excerpt from the book Flutter in Action . Null-

https://flutterbyexample.com/lesson/null-aware-operators



課程設計

Dart 基礎程式設計 三：Null Aware Operators

Null safety principles

> https://dart.dev/null-safety

Dart null safety support is based on the following three core design principles:

- `Non-nullable by default` Unless you explicitly tell Dart that a variable can be null, it's considered non-nullable. This default was chosen after research found that non-null was by far the most common choice in APIs.
- `Incrementally adoptable` You choose *what* to migrate to null safety, and *when*. You can migrate incrementally, `mixing null-safe and non-null-safe` code in the same project. We provide tools to help you with the migration.
- `Fully sound` Dart's null safety is sound, which enables compiler optimizations. If the type system determines that something `isn't null`, then that thing `can never be null`. Once you migrate your whole project and its dependencies to null safety, you reap the full benefits of soundness — not only `fewer bugs`, but `smaller binaries` and `faster execution`.

Null Aware Operators

A standard null check

```
import 'dart:math';

String? getName() {
  final List names = <String?>['Joe', 'Kiven', 'Young', null];

  return names[Random().nextInt(names.length)];
}

void main() {
  String? s1 = getName();
  if (s1 != null) {
    print(s1.length);
  }
}
```

The ?. operator

```
import 'dart:math';

String? getName() {
  final List names = <String?>['Joe', 'Kiven', 'Young', null];
  return names[Random().nextInt(names.length)];
}

void main() {
  String? s1 = getName();
  print(s1?.length);
}
```

If user is indeed `null`, then your program will assign userAge to `null`, but it The ?. operator won't throw an error, and everything will be fine.

The ?? operator

```
import 'dart:math';

String? getName() {
  final List names = <String?>['Joe', 'Kiven', 'Young', null];
  return names[Random().nextInt(names.length)];
}

void main() {
  String s1 = getName() ?? '';
  print(s1.length);
}
```

The ??= operator

```
void main() {
  int? x = null;
  x ??= 88;
  int? y = 0;
  y ??= 88;
  print("x = $x  y = $y");
}
```

Console

```
x = 88  y = 0
```

```
import 'dart:math';

String? getName() {
  final List names = <String?>['Joe', 'Kiven', 'Young', null];
  return names[Random().nextInt(names.length)];
}

void main() {
  String? s1 = getName();
  s1 ??= '';
  print(s1.length);
}
```

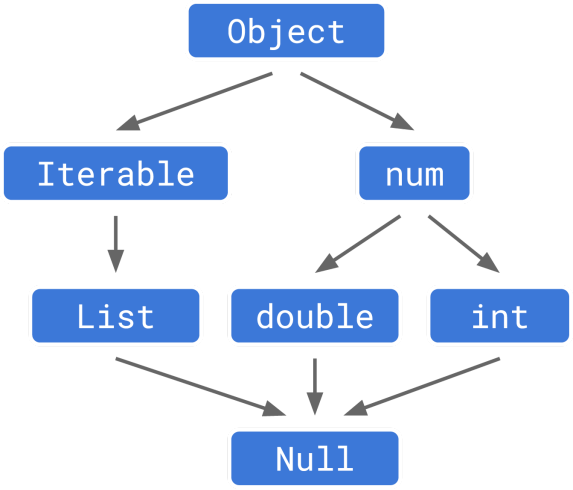
Sound null safety

> https://dart.dev/null-safety

When you opt into null safety, types in your code are `non-nullable by default`, meaning that variables can't contain `null` *unless you say they can.* With null safety, your `runtime` null-dereference errors turn into `edit-time` analysis errors.

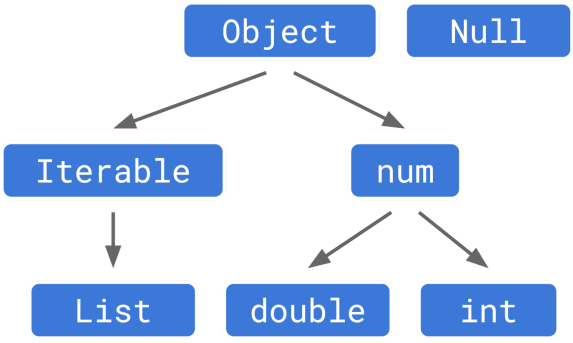
```
import 'dart:math';

int? foo() {
  if (Random().nextBool()) {
    return Random().nextInt(100);
  } else {
    return null;
  }
}

void main() {
  int? a = 3;
  print(a);
  a = foo();
  print(a);
  print(foo());
}
```

Nullability in the type system



Understanding null safety | Dart

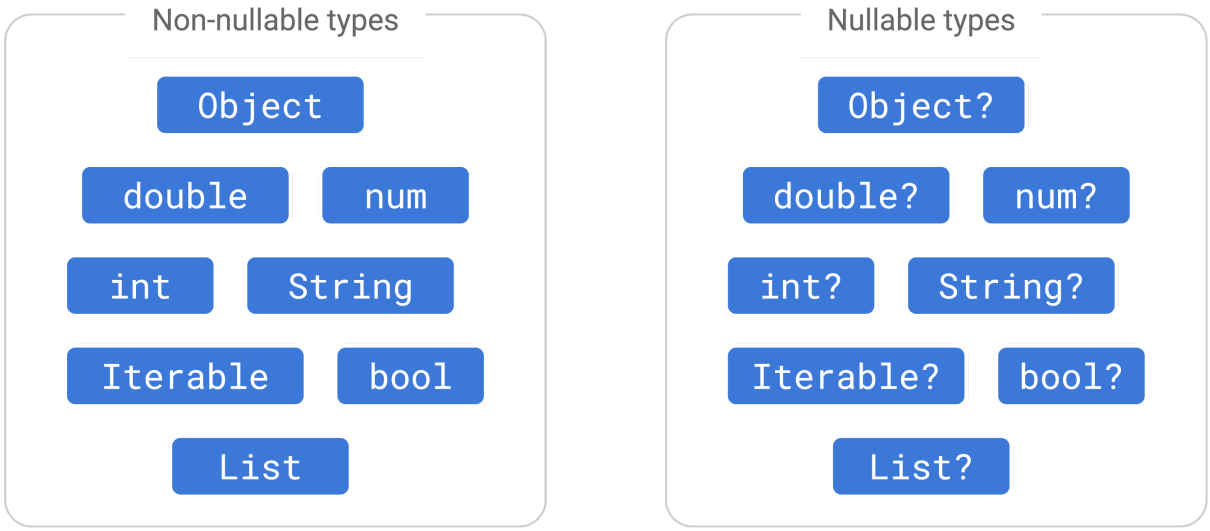
Non-nullable and nullable types



Understanding null safety | Dart

```
import 'dart:math';

// Hypothetical unsound null safety:
void foo(String? nullableStr) {
  if (nullableStr != null) {
    print(nullableStr.length);
  }
}

// Using null safety
void goo(String? nullableStr) {
  print(nullableStr?.length);
}

main() {
  foo(Random().nextBool()?"Hello":null);
```

```
  goo(Random().nextBool()?"Hello":null);
}
```



Understanding null safety | Dart

`?`, `?.`, `??`, `??=` Altogether

```
import 'dart:math';

int? foo() {
  return Random().nextBool() ? Random().nextInt(6) : null;
}

void main() {
  print(foo()?.isEven);
  print(foo() ?? 88);
  int? x = foo();
  x ??= 99;
  print(x);
}
```

更詳細的操作說明，請參閱『翻轉教室輔助影片』...


FUTURE