Vadaski LV3
2018年10月01日 阅读 5753

关注

【译】Dart | 什么是Mixin

This article is from Medium written by Romain Rastel, Thank you Romain for allowing me translate your awesome article into Chinese!

这篇文章来自Romain Rastel撰写的Medium，感谢Romain允许我将你精彩的文章翻译成中文！

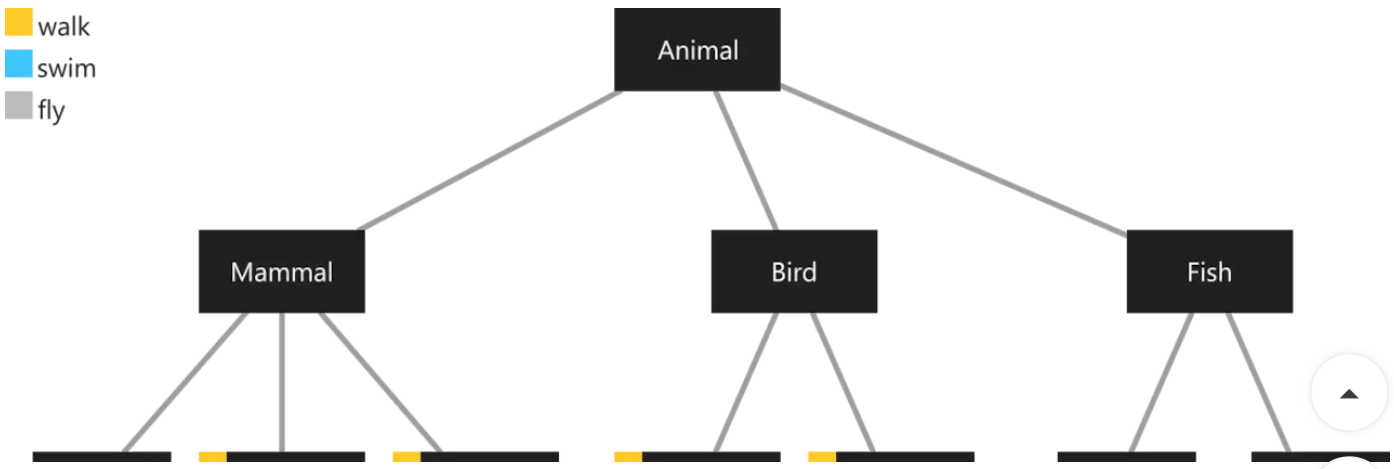
原文链接:medium.com/flutter-com...

当我开始学习Dart时，mixins对我来说是一个新的的概念。 我从C#转过来，Mixin这个概念是不存在的（据我所知，至少在C＃8.0之前不存在）。 起初，我发现这个概念有点难以理解，直到现在我才意识到它有多么强大。

免责声明： Mixins在Dart 2中不断发展。本文一些内容未来可能将会不再适用。

🤔为什么我们需要Mixin

我们先来看看下面的类继承图：



我们这里有一个名为Animal的超类，它有三个子类（Mammal，Bird和Fish）。在底部，我们有具体的一些子类。 小方块代表行为。例如，蓝色方块表示具有此行为的类的实例可以swim。

有些动物有共同的行为：猫和鸽子都可以行走，但是猫不能飞（除了Nyan Cat😀）。 这些行为与此分类正交，因此我们无法在超类中实现这些行为。

如果一个类可以拥有多个超类，那就很容易办到了。我们可以创建另外三个类：Walker，Swimmer，Flyer。在那之后，我们只需从Walker类继承Dove和Cat。但在Dart中，每个类（除了Object类）都只有一个超类。

我们可以实现它，而不是继承自Walker类，因为它是一个接口，但我们必须在多个类中实现行为，因此它并不是一个好的解决方案。

我们需要一种在多个类层次结构中重用类的代码的方法。 Mixin就能够办到这一点！

'Mixins are a way of reusing a class's code in multiple class hierarchies. — dartlang.org'

限制

mixin功能有一些限制（来自dartlang.org）：

- 在Dart 1.12或更低版本使用Mixin时必须继承至Object，并且不能调用super（）方法。
- **SuperMixin**：Dart 1.13或更高版本支持继承至Object以外的类并使用Mixin，而且可以调用super.method（）。这项支持仅在DartVM中默认开启，并且在标志后面的Analyzer中才能够使用。更具体地说，它位于命令行分析器中的--supermixin标志之后。它也可以在分析服务器中，在客户端可配置选项后面。Dart2js和dartdevc不支持SuperMixin。
- 在Dart 2.1中，mixins预计会有更少的限制。例如，Flutter支持mixins调用super（）并从Object以外的类扩展，但是这些在出现在所有Dart SDK中之前，语法有可能会发生变化。

语法

我们明白了mixins为什么如此有用，下面让我们看看如何创建和使用它们。

Mixins通过普通的类声明隐式定义：

dart

```
class Walker {
  void walk() {
    print("I'm walking");
  }
}
```

如果我们不想让我们创建的mixin被实例化或扩展，我们可以像这样定义它：

dart

```
abstract class Walker {
  // This class is intended to be used as a mixin, and should not be
  // extended directly.
  factory Walker._() => null;

  void walk() {
    print("I'm walking");
  }
}
```

要使用mixin的话，你需要使用with关键字，后跟一个或多个mixin的名称：

dart

```
class Cat extends Mammal with Walker {}

class Dove extends Bird with Walker, Flyer {}
```

我在Cat类上定义了Walker mixin，它允许我们调用walk方法而不是fly方法（在Flyer中定义）。

dart

```
main(List<String> arguments) {
  Cat cat = Cat();
  Dove dove = Dove();

  // A cat can walk.
  cat.walk();

  // A dove can walk and fly.
  dove.walk();
  dove.fly();

  // A normal cat cannot fly.
```

详情

我告诉过你我发现这个概念有点难以理解，但是到目前为止它看上去并不那么难是吗?

哈哈。

那么，你能告诉我们以下程序的输出是什么吗😳?

dart

```dart
class A {
  String getMessage() => 'A';
}

class B {
  String getMessage() => 'B';
}

class P {
  String getMessage() => 'P';
}

class AB extends P with A, B {}

class BA extends P with B, A {}

void main() {
  String result = '';

  AB ab = AB();
  result += ab.getMessage();

  BA ba = BA();
  result += ba.getMessage();

  print(result);
}
```

AB和BA类都使用A和B mixins继承至P类，但顺序不同。 所有的A（3个），B和P类都有一个名为getMessage的方法。

那么你认为，结果是什么?

- A. It does not compile
- B. BA
- C. AB
- D. BAAB
- E. ABBA

...

当当当~答案是B！这个程序将打印BA。

我想你在猜测mixins的声明顺序非常重要。

Why？它是如何工作的?

线性化

当您将mixin混入类中时，请记住下面这句话：

'Dart中的Mixins通过创建一个新类来实现，该类将mixin的实现层叠在一个超类之上以创建一个新类，它不是“在超类中”，而是在超类的“顶部”，因此如何解决查找问题不会产生歧义。

— Lasse R. H. Nielsen on StackOverflow.'

实际上，这段代码

```dart
class AB extends P with A, B {}

class BA extends P with B, A {}
```

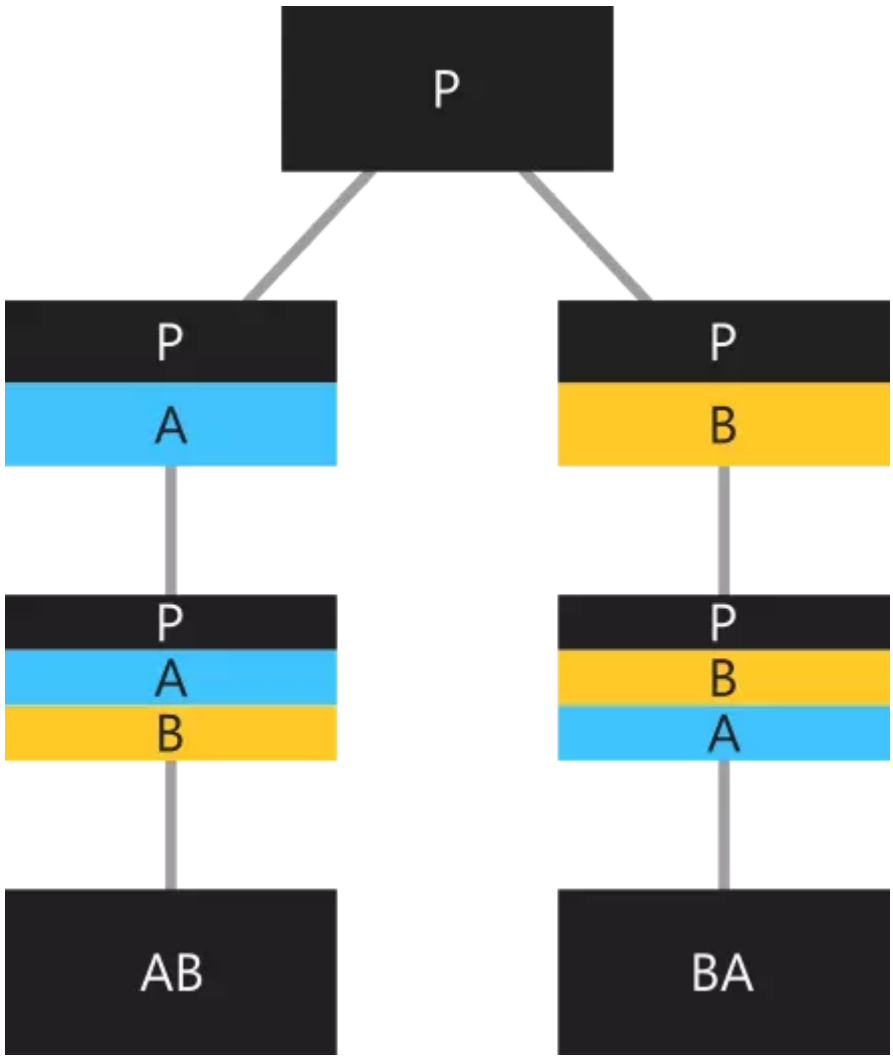
在语义上等同于

```dart
class PA = P with A;
class PAB = PA with B;
```

```
class PB = P with B;
class PBA = PB with A;

class BA extends PBA {}
```

最终的继承关系可以用下图表示：



在AB和P之间创建新类，这些新类是超类P与A类和B类之间的混合类。

正如你所看到的那样，我们并没有使用多重继承！

- Mixins不是一种在经典意义上获得多重继承的方法。
- Mixins是一种抽象和重用一系列操作和状态的方法。
- 它类似于扩展类所获得的重用，但它与单继承兼容，因为它是线性的。

声明mixins的顺序代表了从最高级到最高级的继承链，这件事非常重要，你需要记住。

类型

mixin应用程序实例的类型是什么?

通常，它是其超类的子类型，也是mixin名称本身表示的类的子类型，即原始类的类型。 — dartlang.org

所以这意味着这个程序

```dart
class A {
  String getMessage() => 'A';
}

class B {
  String getMessage() => 'B';
}

class P {
  String getMessage() => 'P';
}

class AB extends P with A, B {}

class BA extends P with B, A {}

void main() {
  AB ab = AB();
  print(ab is P);
  print(ab is A);
  print(ab is B);

  BA ba = BA();
  print(ba is P);
  print(ba is A);
  print(ba is B);
}
```

将在控制台中打印六行true。

Lasse R. H. Nielsen给了我一个很棒的解释：

由于每个mixin应用程序都创建一个新类，它还会创建一个新接口（因为所有Dart类也定义了接口）。 如上所述，新类扩展了超类并包含了mixin类成员的副本，但它也实现了mixin类接口。

在大多数情况下，无法引用mixin-application类或其接口。

Super with Mixin的类只是类的匿名超类，声明类似C类使用Mixin {}扩展Super。 如果你将一个mixin应用程序命名为类CSuper = Super with Mixin {}，那么你可以参考mixin应用程序类及其接口，它将是Super和Mixin的子类型。

— Lasse R. H. Nielsen

😳什么时候应该使用mixins?

当我们想要在不共享相同类层次结构的多个类之间共享行为时，或者在超类中实现此类行为没有意义时，Mixins非常有用。

通常情况下是序列化（例如，查看jaguar_serializer）或持久化。 但是你也可以使用mixins来提供一些实用功能（比如Flutter中的RenderSliverHelpers）。

花点时间玩这个功能，我相信你会找到新的用例😉。 不要局限于无状态mixins，你绝对可以存储变量并使用它们！

📈Mixins的规范正在发展

如果你对Dart语言的演变感兴趣，你应该知道它的规范将在Dart 2.1中发展，他们会喜欢你的反馈。 有关详细信息，请阅读此内容。

为了让您了解未来的一些趋势，请考虑以下源代码：

```dart
abstract class Super {
  void method() {
    print("Super");
```

```
class MySuper implements Super {
  void method() {
    print("MySuper");
  }
}

mixin Mixin on Super {
  void method() {
    super.method();
    print("Sub");
  }
}

class Client extends MySuper with Mixin {}

void main() {
  Client().method();
}
```

第13行到第18行的mixin声明表示Super上的超类约束。 这意味着为了将这个mixin用在这里，这个类必须继承或实现Super，因为mixin使用了Super提供的功能。

这个程序的输出是：

```
MySuper
Sub
```

如果你想知道为什么，请回忆mixins是如何线性化的：

第15行调用super.method（）实际上调用了第8行声明的方法。

完整的 Animal example

你可以在下面找到我用它介绍mixins的完整示例：

```dart
abstract class Animal {}

abstract class Mammal extends Animal {}

abstract class Bird extends Animal {}

abstract class Fish extends Animal {}

abstract class Walker {
  // This class is intended to be used as a mixin, and should not be
  // extended directly.
  factory Walker._() => null;

  void walk() {
    print("I'm walking");
```

```
abstract class Swimmer {
  // This class is intended to be used as a mixin, and should not be
  // extended directly.
  factory Swimmer._() => null;

  void swim() {
    print("I'm swimming");
  }
}

abstract class Flyer {
  // This class is intended to be used as a mixin, and should not be
  // extended directly.
  factory Flyer._() => null;

  void fly() {
    print("I'm flying");
  }
}

class Dolphin extends Mammal with Swimmer {}

class Bat extends Mammal with Walker, Flyer {}

class Cat extends Mammal with Walker {}

class Dove extends Bird with Walker, Flyer {}

class Duck extends Bird with Walker, Swimmer, Flyer {}

class Shark extends Fish with Swimmer {}

class FlyingFish extends Fish with Swimmer, Flyer {}
```

我们可以在下面看到这些mixin是如何线性化的：

总结

我们看到mixins是一个强大的概念，允许您跨多个类层次结构重用代码。

Flutter经常使用到这个功能，我觉得更好地理解它非常重要，这就是为什么我跟你分享我的理解。

感谢Jeroen Meijer的校对。

如果您对Mixin十分感兴趣，欢迎在下方与我留言，或者联系我，我们一起讨论！😃

> **关注下面的标签，发现更多相似文章**

Flutter