flutter 关于 abstract 抽象类、mixins 混入、implements 接口实现

CodingMann丶许半仙

前言：

最近看了很多关于标题的文章，忙里偷闲总结一下自己的理解，这篇文章的总结不会很全面的解释各个名词的概念，而是针对我个人觉着大家容易遗漏混淆的点，或者比较重要的点的一个总结。如果对概念都不清晰的朋友可先自行百度了解。

ps：有不对的地方还请大家指正，谢谢。

ps：今天又看到一篇关于这块内容不错的文章- https://www.jianshu.com/p/863c531380cd

- ## abstract 抽象类

1、抽象类不能被实例化，只有继承它的子类可以。

2、抽象类中一般我们把没有方法体的方法称为抽象方法。

3、子类继承抽象类必须实现它的抽象方法。

4、如果把抽象类当做接口实现的话必须得实现抽象类里面定义的所有属性和方法。

- ## mixins 混入

1、mixin用于修饰类，和abstract类似，该类可以拥有成员变量、普通方法、抽象方法，但是不可以实例化。
2、mixin一般用于描述一种具有某种功能的组块，而某一对象可以拥有多个不同功能的组块。

- mixin：定义了组块。
- on：限定了使用mixin组块的宿主必须要继承于某个特定的类；在mixin中可以访问到该特定类的成员和方法。
- with：负责组合组块，而with后面跟的类并不一定需要是mixin的，abstract class和普通类都是可以的，这一点需要注意

混入的一些基本概念与作用的理解可以参考 https://blog.csdn.net/HuberCui/article/details/93468810

下面用一些实际的例子来加深我们的理解：

在我们开发多屏app的时候，我们更倾向于复用一些类的代码去完成一些功能，让代码能够“高复用”，比如：全局错误提示、页面的公共视图部分的复用、响应式编程（Bloc）里面的依赖逻辑等。使用抽象类abstract基本上就能完成这些功能，但是问题来了，如果页面上的公共部分我不想用到所

有页面，只想在特定的页面上用，那该怎么办呢? 比如AppBar, 部分页面不需要公共的，而需要自定义，因为一个class类只能是一个类的子类，而我们需要更灵活的类的组合，这就是我们为什么需要mixin。

ps：本段转自 https://www.jianshu.com/p/7e14ed414dce，这是我觉得很好的一篇关于mixins实践的文章。

1、flutter中的继承是单继承，使用mixins相当于变相的实现了多继承。

2、由于mixins通过代码更好理解它的含义与用法，上代码！

```
mixin到底是怎么用的? 让我们先举一个例子，新建一个abstract class
Person
abstract class Person {
  void think() {
    print("Hmm, I wonder what I can do today");
  }
}

我们可以用extend关键字把这个类作为父类来使用，比如：
class Mike extends Person {} //由于person类里并不是抽象方法，我
们可以不实现或者重写

void main() {
  var mike = Mike();
  mike.think(); // prints: "Hmm, I wonder what I can do
today"
}

但是我们要这么给Mike添加一些其他“新功能”呢? 比如Mike是一位 coder ，
```

他有coder的一些特性，但不是所有人都有，该怎么办呢？ mixin就能解决这个问题。
首先，我们需要创建一个mixin类并添加一我们需要的新的方法：

```
mixin Coder {
  void code() {
    print("Coding intensifies");
  }
}
```

使用关键字with,我们能将这个“新功能”添加给Mike:

```
class Mike extends Person with Coder {}
```

并且，与父类一样，我们可以调用在Coder中创建的所有函数：

```
void main() {
  var mike = Mike();
  mike.code(); // prints: "Coding intensifies"
}
```

现在，每一个使用 mixin coder的类都拥有coder的方法，然而这带来了一个问题：这意味着，如果我们有一个带有子级Squirrel的父类Animal，那么我们也可以拥有一个可以code（）方法的Squirrel！为了防止这种情况，我们可以使用关键字on将mixin的使用“锁定”到一个类以及从该类继承的所有类：

```
mixin Coder on Person{
  void code() {
    print("Coding intensifies");
  }
}
```

这相当于为我们提供了一个强大的工具：现在我们可以覆盖重写在Person类中设置的方法用来添加或扩展其功能。

```
mixin Coder on Person{
  //...

  @override
  void think() {
```

```
    super.think();
    print("I'm going to code today!");
  }
}
```

调用super.think () 可确保我们仍然可以调用Person中定义的代码，上面的代码扩展了Mike类的think()方法将会输出：

```
Hmm, I wonder what I can do today
I'm going to code today!
```

- 接下来用一个开发中可能会经常遇到的情景来加深我们的理解

mixins 与 基础类：一个实际的常用例子

先试着想一下这种情况我们在 flutter app 中应该怎么做：

在 app 中我们有两个页面是这样的：

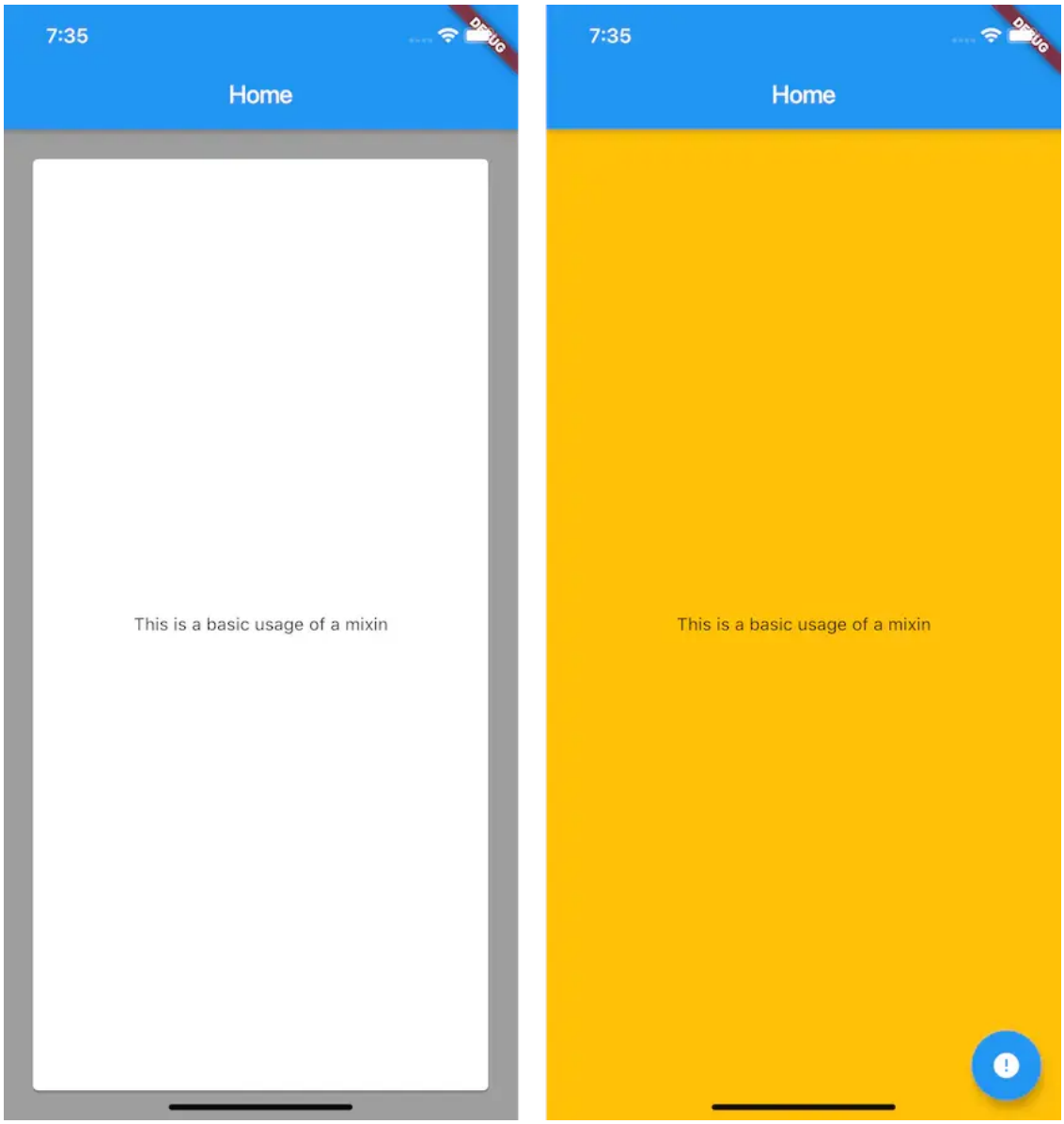

image.png

我们的app有几个屏幕如上面显示的那样。我们想共用每个屏幕的AppBar和background，我们可以使用mixin解决问题。

在这种情况下，我们都定义了屏幕标题，我们将创建一个基类，该基类具有一种提供屏幕名称的方法，该基类称为

BasePage。我们也将仅在 StatefulWidgets 中应用 mixins，因为我们的类将维护并更改其状态。这样，我们创建了两个用于页面的类：BasePage 和 BaseState <BasePage>分别继承 StatefulWidget 和 State <StatefulWidget>。

```
abstract class BasePage extends StatefulWidget {
  BasePage({Key key}) : super(key: key);
}

// TODO: Page为命名泛型 继承 BasePage,    BaseState作为抽象基类，也会被子类继承，所以传入泛型限制参数类型
abstract class BaseState<Page extends BasePage> extends State<Page> {
  String screenName();
}
```

我们现在创建一个自定义 mixin BasicPageMixin，在其中定义页面的背景和标题名称。

```
// TODO: BasicPage 是一个mixin,作用于BaseState和其基类，抽取渲染页面公共部分
mixin BasicPage<Page extends BasePage> on BaseState<Page> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: Text(screenName()),
        ),
        body: Container(
```

```
          child: body(),
          color: Colors.amber,
        ));
  }

  Widget body();
}
```

由于 body（）方法没有实例，因此使用此 mixin 的每个类都必须实现它，以确保我们不会忘记在页面中添加 body（）。

在屏幕上面我们看到了 FloatingActionButton，但是我们不是每个屏幕都需要展示它，该怎么办呢？我们可以声明一个新方法 fab()，默认的渲染输出一个空的 Container。如果继承的子类需要它，那么子类可以通过 @override 重写 fab() 来添加一个 FloatingActionButton。

```
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: Text(screenName()),
        ),
        body: Container(
          child: body(),
          color: Colors.amber,
        ));
        floatingActionButton: fab(),
  }
```

```
  // NOTE: 这里没有在基类中初始化是因为渲染wedget 尽量写在子组件
state中, 更好的状态管理,更好的性能
  // TODO: body()子类必须实现
  Widget body();
  // TODO: fab()子类可选实现
  Widget fab() => Container();
}
```

我们的 mixin 已经创建好了， 我们来通过它新建一个页面：

```
class MyMixinHomePage extends BasePage {
  MyMixinHomePage({Key key}) : super(key: key);
  @override
  _MyMixinHomePageState createState() =>
_MyMixinHomePageState();
}

class _MyMixinHomePageState extends
BaseState<MyMixinHomePage> with BasicPage{
  @override
  String screenName() => "Home";

  @override
  Widget body() {
    return Center(child: Text("This is a basic usage of a
mixin"));
  }
}
```

有了这个，我们现在只需要声明一个 body（）和一个可能在屏幕中使用的 fab（）小部件，从而节省了几十行代码。

very nice !

- ## implements 接口实现

1、implements与 extends最大的不同就是允许后面接上多个普通或者抽象类，当我们使用 B implement A 修饰时，那么 A 中的所有的属性和方法都要在~~A~~ B 中实现，无论它原来是抽象方法还是普通方法。

2、如果我们只想要 A 中的接口定义，而不想要它的实现，那么就试用 implements。

废话少说，上代码！

```
class Implements {

  void base() {
    print('base');
  }

  void log() {
    print('extends');
  }

}

class Log implements Implements {
```

```
  base() {
    print('log#base');
  }

  log() {
    print('log');
  }

}

void main() {
  Log().base();
  Log().log();
}
输出结果:
log#base
log
```