# **[[Flutter] 一些面试可能会问基础知识](https://www.cnblogs.com/yangyxd/p/10449614.html)**

# **1. Flutter 是什么？**

Flutter是谷歌的移动UI框架，可以快速在iOS和Android上构建高质量的原生用户界面。 Flutter可以与现有的代码一起工作。在全世界，Flutter正在被越来越多的开发者和组织使用，并且Flutter是完全免费、开源的。

# **2. Flutter 特性有哪些？**

快速开发（毫秒级热重载）

* 绚丽UI（内建漂亮的质感设计Material Design和Cupertino Widget和丰富平滑的动画效果和平台感知）
* 响应式(Reactive，用强大而灵活的API解决2D、动画、手势、效果等难题)
* 原生访问功能
* 堪比原生性能

# **3. Flutter 和 Dart的关系是什么？**

Flutter是一个使用Dart语言开发的跨平台移动UI框架，通过自建绘制引擎，能高性能、高保真地进行移动开发。Dart囊括了多数编程语言的优点，它更符合Flutter构建界面的方式。

# **4. Dart 语言的特性？**

* Productive（生产力高，Dart的语法清晰明了，工具简单但功能强大）
* Fast（执行速度快，Dart提供提前优化编译，以在移动设备和Web上获得可预测的高性能和快速启动。）
* Portable（易于移植，Dart可编译成ARM和X86代码，这样Dart移动应用程序可以在iOS、Android和其他地方运行）
* Approachable（容易上手，充分吸收了高级语言特性，如果你已经知道C++，C语言，或者Java，你可以在短短几天内用Dart来开发）
* Reactive（响应式编程）

# **5. Dart的一些重要概念？**

* 在Dart中，一切都是对象，所有的对象都是继承自Object
* Dart是强类型语言，但可以用var或 dynamic来声明一个变量，Dart会自动推断其数据类型,dynamic类似c#
* 没有赋初值的变量都会有默认值null
* Dart支持顶层方法，如main方法，可以在方法内部创建方法
* Dart支持顶层变量，也支持类变量或对象变量
* Dart没有public protected private等关键字，如果某个变量以下划线（\_）开头，代表这个变量在库中是私有的

# **6. dart是值传递还是引用传递？**

dart是引用传递的。

# **7. Widget 和 element 和 RenderObject 之间的关系？**

* Widget是用户界面的一部分,并且是不可变的。
* Element是在树中特定位置Widget的实例。
* RenderObject是渲染树中的一个对象，它的层次结构是渲染库的核心。

Widget会被inflate（填充）到Element，并由Element管理底层渲染树。Widget并不会直接管理状态及渲染,而是通过State这个对象来管理状态。Flutter创建Element的可见树，相对于Widget来说，是可变的，通常界面开发中，我们不用直接操作Element,而是由框架层实现内部逻辑。就如一个UI视图树中，可能包含有多个TextWidget(Widget被使用多次)，但是放在内部视图树的视角，这些TextWidget都是填充到一个个独立的Element中。Element会持有renderObject和widget的实例。记住，Widget 只是一个配置，RenderObject 负责管理布局、绘制等操作。

在第一次创建 Widget 的时候，会对应创建一个 Element， 然后将该元素插入树中。如果之后 Widget 发生了变化，则将其与旧的 Widget 进行比较，并且相应地更新 Element。重要的是，Element 不会被重建，只是更新而已。

# **8. mixin extends implement 之间的关系?**

继承（关键字 extends）、混入 mixins （关键字 with）、接口实现（关键字 implements）。这三者可以同时存在，前后顺序是extends -> mixins -> implements。

Flutter中的继承是单继承，子类重写超类的方法要用@Override，子类调用超类的方法要用super。

在Flutter中，Mixins是一种在多个类层次结构中复用类代码的方法。mixins的对象是类，mixins绝不是继承，也不是接口，而是一种全新的特性，可以mixins多个类，mixins的使用需要满足一定条件。

# **9. 使用mixins的条件是什么？**

因为mixins使用的条件，随着Dart版本一直在变，这里讲的是Dart2.1中使用mixins的条件：

mixins类只能继承自object
mixins类不能有构造函数
一个类可以mixins多个mixins类
可以mixins多个类，不破坏Flutter的单继承

# **10. mixin 怎么指定异常类型？**

on关键字可用于指定异常类型。 on只能用于被mixins标记的类，例如mixins X on A，意思是要mixins X的话，得先接口实现或者继承A。这里A可以是类，也可以是接口，但是在mixins的时候用法有区别.

on 一个类：



class A {

 void a(){

 print("a");

 }

}

mixin X on A{

 void x(){

 print("x");

 }

}

class mixinsX extends A with X{

}



on 的是一个接口： 得首先实现这个接口，然后再用mix



class A {

 void a(){

 print("a");

 }

}

mixin X on A{

 void x(){

 print("x");

 }

}

class implA implements A{

 @override

 void a() {}

}

class mixinsX2 extends implA with X{

}



# **11. Flutter main future mirotask 的执行顺序?**

普通代码都是同步执行的，结束后会开始检查microtask中是否有任务，若有则执行，执行完继续检查microtask，直到microtask列队为空。最后会去执行event队列（future）。

# **12. Future和Isolate有什么区别？**

future是异步编程，调用本身立即返回，并在稍后的某个时候执行完成时再获得返回结果。在普通代码中可以使用await 等待一个异步调用结束。

isolate是并发编程，Dartm有并发时的共享状态，所有Dart代码都在isolate中运行，包括最初的main()。每个isolate都有它自己的堆内存，意味着其中所有内存数据，包括全局数据，都仅对该isolate可见，它们之间的通信只能通过传递消息的机制完成，消息则通过端口(port)收发。isolate只是一个概念，具体取决于如何实现，比如在Dart VM中一个isolate可能会是一个线程，在Web中可能会是一个Web Worker。

# **13. Stream 与 Future是什么关系？**

Stream 和 Future 是 Dart 异步处理的核心 API。Future 表示稍后获得的一个数据，所有异步的操作的返回值都用 Future 来表示。但是 Future 只能表示一次异步获得的数据。而 Stream 表示多次异步获得的数据。比如界面上的按钮可能会被用户点击多次，所以按钮上的点击事件（onClick）就是一个 Stream 。简单地说，Future将返回一个值，而Stream将返回多次值。Dart 中统一使用 Stream 处理异步事件流。Stream 和一般的集合类似，都是一组数据，只不过一个是异步推送，一个是同步拉取。

# **14. Stream 两种订阅模式？**

Stream有两种订阅模式：**单订阅(single)**和 **多订阅（broadcast）**。单订阅就是只能有一个订阅者，而广播是可以有多个订阅者。这就有点类似于消息服务（Message Service）的处理模式。单订阅类似于点对点，在订阅者出现之前会持有数据，在订阅者出现之后就才转交给它。而广播类似于发布订阅模式，可以同时有多个订阅者，当有数据时就会传递给所有的订阅者，而不管当前是否已有订阅者存在。
Stream 默认处于单订阅模式，所以同一个 stream 上的 listen 和其它大多数方法只能调用一次，调用第二次就会报错。但 Stream 可以通过 transform() 方法（返回另一个 Stream）进行连续调用。通过 Stream.asBroadcastStream() 可以将一个单订阅模式的 Stream 转换成一个多订阅模式的 Stream，isBroadcast 属性可以判断当前 Stream 所处的模式。

# **15. await for 如何使用?**

await for是不断获取stream流中的数据，然后执行循环体中的操作。它一般用在直到stream什么时候完成，并且必须等待传递完成之后才能使用，不然就会一直阻塞。



Stream<String> stream = new Stream<String>.fromIterable(['不开心', '面试', '没', '过']);

main() async{

 print('上午被开水烫了脚');

 await for(String s in stream){

 print(s);

 }

 print('晚上还没吃饭');

}



# **16. Flutter中的Widget、State、Context 的核心概念？是为了解决什么问题？**

**Widget**: 在Flutter中，几乎所有东西都是Widget。将一个Widget想象为一个可视化的组件（或与应用可视化方面交互的组件），当你需要构建与布局直接或间接相关的任何内容时，你正在使用Widget。

**Widget树**: Widget以树结构进行组织。包含其他Widget的widget被称为父Widget(或widget容器)。包含在父widget中的widget被称为子Widget。

**Context**: 仅仅是已创建的所有Widget树结构中的某个Widget的位置引用。简而言之，将context作为widget树的一部分，其中context所对应的widget被添加到此树中。一个context只从属于一个widget，它和widget一样是链接在一起的，并且会形成一个context树。

**State**: 定义了StatefulWidget实例的行为，它包含了用于”**交互/干预**“Widget信息的行为和布局。应用于State的任何更改都会强制重建Widget。

这些状态的引入，主要是为了解决多个部件之间的交互和部件自身状态的维护。

# **17. Widget的两种类型是什么？**

**StatelessWidget**: 一旦创建就不关心任何变化，在下次构建之前都不会改变。它们除了依赖于自身的配置信息（在父节点构建时提供）外不再依赖于任何其他信息。比如典型的Text、Row、Column、Container等，都是StatelessWidget。它的生命周期相当简单：初始化、通过build()渲染。

**StatefulWidget**: 在生命周期内，该类Widget所持有的数据可能会发生变化，这样的数据被称为**State**，这些拥有动态内部数据的Widget被称为StatefulWidget。比如复选框、Button等。State会与Context相关联，并且此关联是永久性的，State对象将永远不会改变其Context，即使可以在树结构周围移动，也仍将与该context相关联。当**state与context关联时**，state被视为**已挂载**。StatefulWidget由两部分组成，在初始化时必须要在createState()时初始化一个与之相关的State对象。

# **18. State 对象的初始化流程？**

**initState**() : 一旦State对象被创建，initState方法是第一个（构造函数之后）被调用的方法。可通过重写来执行额外的初始化，如初始化动画、控制器等。重写该方法时，应该首先调用super.initState()。在initState中，无法真正使用context，因为框架还没有完全将其与state关联。initState在该State对象的生命周期内将不会再次调用。

**didChangeDependencies**(): 这是第二个被调用的方法。在这一阶段，context已经可用。如果你的Widget链接到了一个InheritedWidget并且/或者你需要初始化一些listeners（基于context），通常会重写该方法。

**build(BuildContext context)**: 此方法在didChangeDependencies()、didUpdateWidget()之后被调用。每次State对象更新（或当InheritedWidget有新的通知时）都会调用该方法！我们一般都在build中来编写真正的功能代码。为了强制重建，可以在需要的时候调用setState((){...})方法。

**dispose()**: 此方法在Widget被废弃时调用。可重写该方法来执行一些清理操作（如解除listeners），并在此之后立即调用super.dispose()。

# **19. Widget 唯一标识Key有那几种？**

在flutter中，每个widget都是被唯一标识的。这个唯一标识在build或rendering阶段由框架定义。该标识对应于可选的Key参数，如果省略，Flutter将会自动生成一个。

在flutter中，主要有4种类型的Key：GlobalKey（确保生成的Key在整个应用中唯一，是很昂贵的，允许element在树周围移动或变更父节点而不会丢失状态）、LocalKey、UniqueKey、ObjectKey。

# **20. 什么是Navigator? MaterialApp做了什么？**

Navigator是在Flutter中负责管理维护页面堆栈的导航器。MaterialApp在需要的时候，会自动为我们创建Navigator。Navigator.of(context)，会使用context来向上遍历Element树，找到MaterialApp提供的\_NavigatorState再调用其push/pop方法完成导航操作。