目录

[参考文章 2](#_Toc9938)

[算是flutter最完整题库了 - 掘金 (juejin.cn) 2](#_Toc20699)

[Dart 2](#_Toc8578)

[● Dart 当中的 「..」表示什么意思？ 2](#_Toc6525)

[● Dart 的作用域是怎么样的？ 2](#_Toc12586)

[● dart是多线程还是单线程执行？ 2](#_Toc29433)

[● 阻塞式调用和非阻塞式调用是怎么样的？ 2](#_Toc15356)

[● 事件循环是什么？ 2](#_Toc31444)

[● dart是值传递还是引用传递？ 2](#_Toc24854)

[● Dart 语言有哪些重要的特性？ 3](#_Toc13468)

[● Dart 语言有哪些重要的概念？ 3](#_Toc6935)

[● Dart线程模型是如何执行的？ 3](#_Toc10419)

[● Dart 是如何实现多任务并行的？ 5](#_Toc20517)

[● await for 如何使用? 5](#_Toc28336)

[● Stream 有哪两种订阅模式？分别是怎么调用的？ 5](#_Toc866)

[● dart中mixin机制是怎么样的？ 6](#_Toc10114)

[● JIT 与 AOT分别是什么？ 6](#_Toc15083)

[● Dart的内存分配与垃圾回收是怎么样的？ 6](#_Toc19966)

[● 使用mixins的条件是什么？ 7](#_Toc1742)

[● mixin 怎么指定异常类型？ 7](#_Toc14458)

[● main future mirotask 的执行顺序是怎样的? 7](#_Toc14482)

[● Future和Isolate有什么区别？ 8](#_Toc15523)

[● Stream 与 Future是什么关系？ 8](#_Toc6231)

[Flutter 8](#_Toc11959)

[● template->vnode->dom 8](#_Toc2962)

# 参考文章

[算是flutter最完整题库了 - 掘金 (juejin.cn)](https://juejin.cn/post/6844904199726039054)

# Dart

* Dart 当中的 「..」表示什么意思？

|  |
| --- |
| 级连操作符  “..” 和 “.” 不同：调用..后返回的相当于是this，而.返回的则是该方法返回的值； |

* Dart 的作用域是怎么样的？

|  |
| --- |
| Dart没有public和private等关键词，默认就是公开的，私有变量使用下划线开头； |

* dart是多线程还是单线程执行？

|  |
| --- |
| 单线程执行，多线程是使用异步来执行的； |

* 阻塞式调用和非阻塞式调用是怎么样的？

|  |
| --- |
| 阻塞：调用结果之前，当前线程会被挂起，调用线程只有在得到结果之后才会继续执行；  非阻塞：调用执行之后，当前线程不会停止运行，只需要过一段时间来检查有没有结果返回即可； |

* 事件循环是什么？

|  |
| --- |
| 将需要处理的一系列事件，放在一个事件队列（Event Queue）中，不断从事件队列中取出事件，并执行需要执行的代码块，直到事件被清空。 |

* dart是值传递还是引用传递？

[Dart是值传递还是引用传递？ - 简书 (jianshu.com)](https://www.jianshu.com/p/3465660c0690)

|  |
| --- |
| dart是值传递。我们每次调用函数，传递过去的都是对象的内存地址，而不是这个对象的复制。 |
| 我们只要记住一点，参数是把内存地址传过去了，如果对这个内存地址上的对象修改，那么其他位置的引用该内存地址的变量值也会修改。千万要记住dart中一切都是对象。 |

* Dart 语言有哪些重要的特性？

|  |
| --- |
| 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） |

* Dart 语言有哪些重要的概念？

|  |
| --- |
| 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 |

* Dart线程模型是如何执行的？

[初略讲解Dart线程模型及异常捕获 - 掘金 (juejin.cn)](https://juejin.cn/post/6844903882301112328)

|  |
| --- |
| 在Java和Objective-C（以下简称OC）中，如果程序发生异常且没有被捕获，那么程序将会被中止，但在Dart或Javascript中则不会，究其原因，这和它们的运行机制有关系。Java和OC都是多线程模型的编程语言，任意一个线程触发异常且没有被捕获时，整个进程就退出了。但Dart和Javascript不会，它们都是单线程模型，运行机制很相似（但有区别），下面我们通过Dart官方提供的一张图来看看Dart大致运行原理：  IMG_256  Dart在单线程中是以消息循环机制来运行的，其中包括两个任务队列，一个是“微任务队列”**microtask queue**，另一个叫做“事件队列”**event queue**。从图中可以发现，微任务队列的执行优先级高于事件队列。  现在我们来介绍一下Dart线程运行过程，如上图中所示，入口函数main()执行完后，消息循环机制便启动了。首先会按照先进先出的顺序逐个执行微任务队列中的任务，当所有微任务队列执行完后便开始执行事件队列中的任务，事件任务执行完毕后再去执行微任务，如此循环往复，生生不息。  在Dart中，所有的外部事件任务都在事件队列中，如IO、计时器和点击以及绘制事件等，而微任务通常来源于Dart内部，并且微任务非常少，之所以如此，是因为微任务队列优先级高，如果微任务太多，执行时间总和就越久，事件队列任务的延迟也就越久，对于GUI应用来说最直观的表现就是比较卡，所以必须得保证微任务队列不会太长。值得注意的是，我们可以通过Future.microtask(...)方法向微任务队列插入一个任务。  在事件循环中，当某个任务发生异常并没有被捕获时，程序并不会退出，而直接导致的结果是当前任务的后续代码就不会被执行了，也就是说一个任务中的异常是不会影响其它任务执行的。 |

* Dart 是如何实现多任务并行的？

|  |
| --- |
| Dart 是单线程的，不存在多线程，那如何进行多任务并行的呢？其实，Dart的多线程和前端的多线程有很多的相似之处。Flutter的多线程主要依赖Dart的并发编程、异步和事件驱动机制。  IMG_258  简单的说，在Dart中，一个Isolate对象其实就是一个isolate执行环境的引用，一般来说我们都是通过当前的isolate去控制其他的isolate完成彼此之间的交互，而当我们想要创建一个新的Isolate可以使用Isolate.spawn方法获取返回的一个新的isolate对象，两个isolate之间使用SendPort相互发送消息，而isolate中也存在了一个与之对应的ReceivePort接受消息用来处理，但是我们需要注意的是，ReceivePort和SendPort在每个isolate都有一对，只有同一个isolate中的ReceivePort才能接受到当前类的SendPort发送的消息并且处理。 |

* 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。  IMG_259 |

* Stream 有哪两种订阅模式？分别是怎么调用的？

|  |
| --- |
| 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。 |

* dart中mixin机制是怎么样的？

|  |
| --- |
| mixin 是Dart 2.1 加入的特性，以前版本通常使用abstract class代替。  简单来说，mixin是为了解决继承方面的问题而引入的机制，Dart为了支持多重继承，引入了mixin关键字，它最大的特殊处在于：mixin定义的类不能有构造方法，这样可以避免继承多个类而产生的父类构造方法冲突。  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。 |

* JIT 与 AOT分别是什么？

|  |
| --- |
| 借助于先进的工具链和编译器，Dart 是少数同时支持 JIT（Just In Time，即时编译）和 AOT（Ahead of Time，运行前编译）的语言之一。  那，到底什么是 JIT 和 AOT 呢？  语言在运行之前通常都需要编译，JIT 和 AOT 则是最常见的两种编译模式。  JIT 在运行时即时编译，在开发周期中使用，可以动态下发和执行代码，开发测试效率高，但运行速度和执行性能则会因为运行时即时编译受到影响。  AOT 即提前编译，可以生成被直接执行的二进制代码，运行速度快、执行性能表现好，但每次执行前都需要提前编译，开发测试效率低。 |

* Dart的内存分配与垃圾回收是怎么样的？

|  |
| --- |
| Dart VM 的内存分配策略比较简单，创建对象时只需要在堆上移动指针，内存增长始终是线性的，省去了查找可用内存的过程。  在 Dart 中，并发是通过 Isolate 实现的。Isolate 是类似于线程但不共享内存，独立运行的 worker。这样的机制，就可以让 Dart 实现无锁的快速分配。  Dart 的垃圾回收，则是采用了多生代算法。新生代在回收内存时采用“半空间”机制，触发垃圾回收时，Dart 会将当前半空间中的“活跃”对象拷贝到备用空间，然后整体释放当前空间的所有内存。回收过程中，Dart 只需要操作少量的“活跃”对象，没有引用的大量“死亡”对象则被忽略，这样的回收机制很适合 Flutter 框架中大量 Widget 销毁重建的场景。 |

* 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。 |

* 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：  IMG_256  on 的是一个接口： 得首先实现这个接口，然后再用mix |

* main future mirotask 的执行顺序是怎样的?

|  |
| --- |
| 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）。 |

* 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，Dart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。 |

* 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。 |

# Flutter

* StatelessWidget

[Flutter系列之Widget生命周期 - 掘金 (juejin.cn)](https://juejin.cn/post/6916164962276605959)

|  |
| --- |
| StatelessWidget  StatelessWidget 派生的组件是无状态组件，无状态组件只是在构建的时候渲染一次，不支持动态变化，即无法通过其他用户操作重绘组件，只能接收传入时的参数进行构建，如下 ：  /// StatelessWidget/// 表示无状态Widget  class StatelessSamplePage extends StatelessWidget {  // 外部传入数据  final String data;  StatelessSamplePage(this.data);  @override  Widget build(BuildContext context) {  return Container(  color: Colors.lightBlue,  child: Text(data),  );  }  }  如上传入的参数只能使用 final 修饰，否则会出现如下警告：  This class (or a class which this class inherits from) is marked as '@immutable', but one or more of its instance fields are not final: StatelessSamplePage.data  提示 Widget 被 @immutable 注解修饰，如下：  @immutableabstract class Widget extends DiagnosticableTree {  此时只能使用 final 来修饰变量，Dart 中被 final 修饰的变量只能被初始化一次，这也符合 StatelessWidget 的无状态特征。 |

* StatefulWidget

|  |
| --- |
| StatefulWidget  StatefulWidget 派生的组件是有状态组件，有状态的组件是支持随着数据变化多次构建 Widget 以完成动态界面的渲染，如果要实现一个实时显示当前时间的界面，显然 StatelessWidget 是不能完成的，只能使用有状态的StatefulWidget 来实现.  实际上，如果只是静态界面 StatelessWidget 和 StatefulWidget 是完全没有区别的，都是能够实现的，唯一区别就是 StatefulWidget 可以通过 setState 方法触发 Widget 的重新构建， State 类就是就是 Stateless -> Stateful 的桥梁。 |

* State生命周期状态

|  |
| --- |
| State生命周期状态  Flutter 生命周期实际上各个组件的生命周期：   * StatelessWidget：无状态组件的生命周期只有 build 构建这个过程； * StatefulWidget：有状态组件的生命周期指的是 State 的生命周期。   Flutter 生命周期实际上就是无状态组件的生命周期，即 State 的生命周期，如下图所示：  IMG_256  如上各个 State 的生命周期状态主要是三个：   * created：指的是 State 的创建状态，当调用 createState 方法之后就处于 created state； * dirty：指的是当调用 setState 等方法数据发生变化，但是 Widget 还未重新构建时的状态； * clean：指的是 Widget 构建后的状态； * defunct：指的是 State.dispose 调用之后的状态，此时对应的 Widget 已被销毁不能够被再次构建。 |

* State生命周期方法

|  |
| --- |
| State生命周期方法  有状态组件的生命周期就是 State 的生命周期，其具体调用过程与 build 触发时机如下图所示：  IMG_256  其生命周期方法具体含义如下：   * createState：StatefulWidget 中用于创建 State； * initState：State 的初始化操作，如变量的初始化等； * didChangeDependencies：initState 调用之后调用，或者使用了 InheritedWidgets 组件会被调用，其中 InheritedWidgets 可用于Flutter 状态管理； * build：用于 Widget 的构建； * deactivate：包含此 State 对象的 Widget 被移除之后调用，若此 Widget 被移除之后未被添加到其他 Widget 树结构中，则会继续调用 dispose 方法； * dispose：该方法调用后释放 Widget 所占资源； * reassemble：用于开发阶段，热重载的时候会被调用，之后会重新构建； * didUpdateWidget：父 Widget 构建的时候子 Widget 的 didUpdateWidget 方法会被调用。 |

# Flutter状态管理

* 状态管理是什么？

|  |
| --- |
| 程序=算法+数据结构  数据是程序的中心。数据结构和算法两个概念间的逻辑关系贯穿了整个程序世界，首先二者表现为不可分割的关系。其实Flutter不就是一个程序吗，那我们面临的最底层的问题还是算法和数据结构，所以我们推导出  Flutter=算法+数据结构  那状态管理是什么？我也用公式来表达一下，如下：  Flutter状态管理=算法+数据结构+UI绑定 |

* 为什么需要状态管理？

|  |
| --- |
| 用于解决状态更新问题，不需要WidgetState 被全局化，保证组件隐私，使得代码可扩展，易维护，可以动态替换UI而不影响算法逻辑，安全可靠，保持数据的稳定伸缩，性能佳，局部优化； |

* 说下状态管理基本分类

|  |
| --- |
| 分为局部管理和全局管理；  局部管理：短暂的状态，这种状态根本不需要做全局处理；  全局管理：即应用状态，非短暂状态，您要在应用程序的许多部分之间共享，以及希望在用户会话之间保持的状态，就是我们所说的应用程序状态（有时也称为共享状态） |

* 状态管理的底层逻辑一般是怎么样的？

|  |
| --- |
| * State：如StatefulWidget、StreamBuilder状态管理方式； * InheritedWidget专门负责Widget树中数据共享的功能型Widget：如Provider、scoped\_model就是基于它开发； * Notification：与InheritedWidget正好相反，InheritedWidget是从上往下传递数据，Notification是从下往上，但两者都在自己的Widget树中传递，无法跨越树传递； * Stream 数据流 ：如Bloc、flutter\_redux、fish\_redux等也都基于它来做实现； |

* 状态管理的使用原则是怎么样的？

|  |
| --- |
| 局部管理优于全局、保持数据安全性、考虑页面重新build带来的影响； |

* 使用成熟状态管理库的弊端有哪些？

|  |
| --- |
| 增加代码复杂性、框架bug修复需要时间等待、不理解框架原理导致使用方式不对，反而带来更多问题、选型错误导致不符合应用要求、与团队风格冲突不适用； |

# Flutter进阶探讨

* ListView流畅度翻倍！！Flutter卡顿分析和通用优化方案

[ListView流畅度翻倍！！Flutter卡顿分析和通用优化方案 - 掘金 (juejin.cn)](https://juejin.cn/post/6940134891606507534)