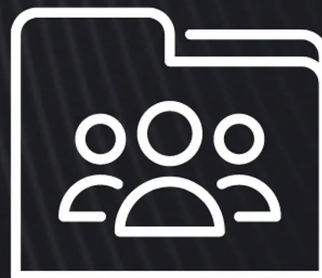


有道 | 技术团队

玩转TypeScript 工具类型（上）



@稀土掘金技术社区

@稀土掘金技术社区

联系我们：有道技术团队助手：ydttech01 / 邮箱：ydttech@rd.netease.com

在 18 年 Flutter 发布正式版 1.0 版本以来，有道 Luna 团队保持持续的关注，在不少业务上进行大量的尝试，Flutter 本身统一 Skia 引擎带来的跨平台特性和一致的体验，AOT 下高性能，JIT 下热重载带来提高开发效率等特性，都让人们保持极大的热情和持续的投入，其生态社区也在快速增长。

从实际表现上来看，整个技术栈设计很好。上层 Flutter Framework 引入 Widget/LayerTree 等概念自己实现了界面描述框架，下层Flutter Engine 把 LayerTree 用 OpenGL 渲染成用户界面。

长期来看，用 Flutter 来替代 Native，实现双端代码统一，节约人力开发，也是我们持续探索的方向。

一、前言

1.1 词典业务尝试

我们使用Flutter在有道词典去年的3月份、7月份分别上线了单词本和听力模考业务，现在是Flutter 1.12版本以下是业务展示



词典笔单词本

8词

 我的单词本

25词

商务英语词汇

2824词



得的

1词

哦哦

1词

医学

30词

@稀土掘金技术社区

单词本

19:56



模考题库

K.O.听力

「地点场景题」，两步搞定so easy!

立即掌握



全部

四级题库

高中题库

2021年新高考卷英语听力

22529人完成



山东省2021届高三开学质检英语听力

691人完成



2021年6月四级第一套英语听力

4338人完成



2021年6月四级第二套英语听力

673人完成



2021年广东省中考英语听力

2095人完成

中考 | 5分钟学会「推理判断」练习一

@稀土掘金技术社区

听力模考

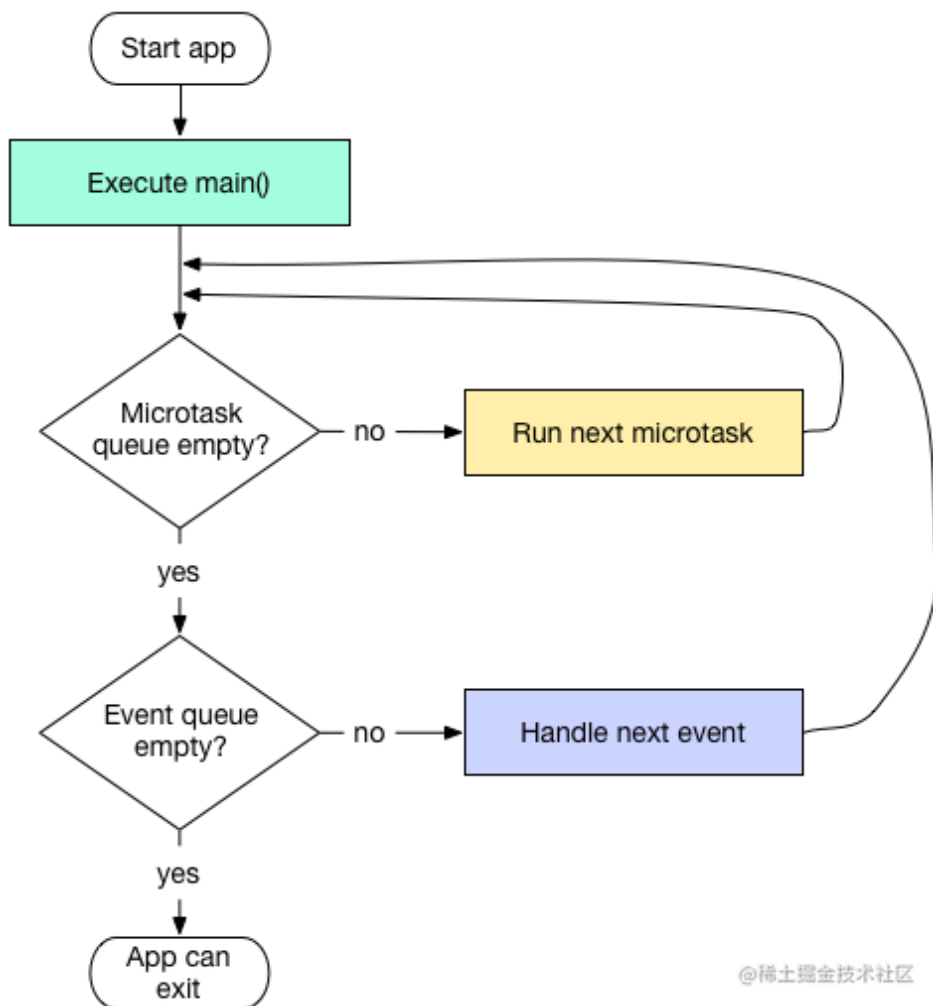
我们在较为独立的新业务上进行大胆尝试，新技术难免会有问题，但是还是要勇于尝试。

二、Flutter基础简介

2.1 Dart

- Dart单线程模型

Dart 和 JavaScript 都是单线程模型，运行机制很相似，Dart 官方提供了一张运行原理图：



Dart 在单线程中是以消息循环机制来运行的，其中包含两个任务队列，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。

其中event queue：负责处理I/O事件、绘制事件、手势事件、接收其他isolate消息等外部事件；
microtask queue：可以自己向isolate内部添加事件，事件的优先级比event queue高。

事件队列模型过程：

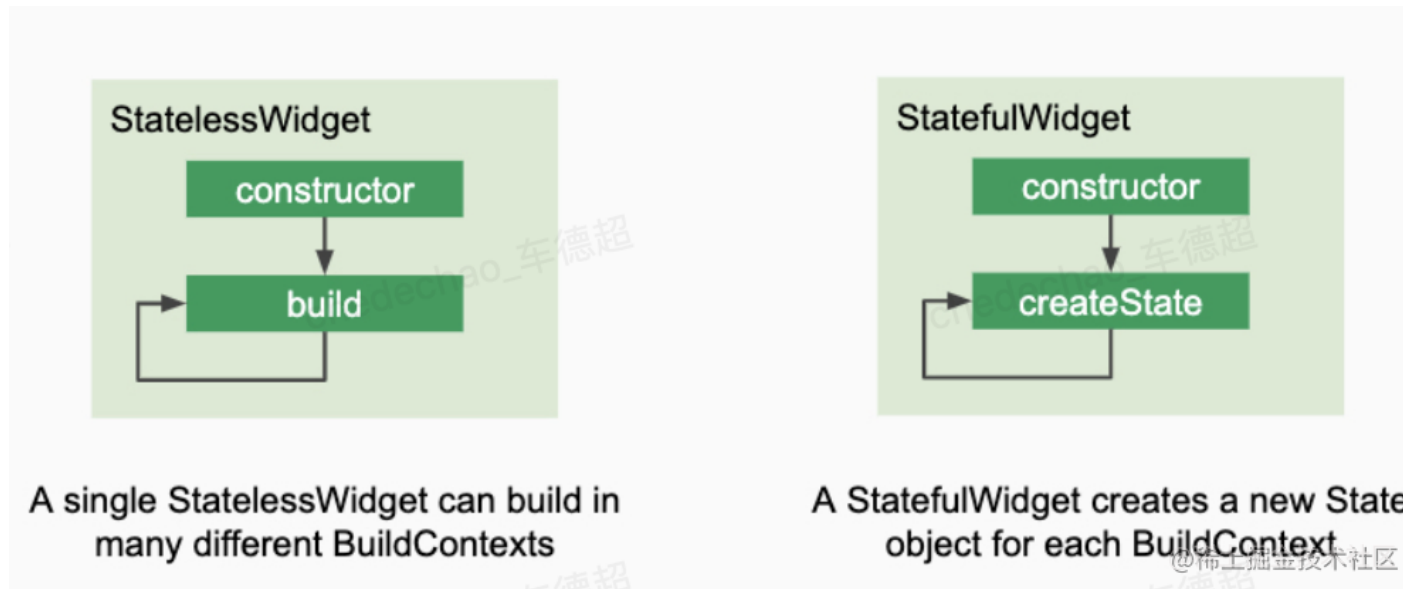
1. 先检查 MicroTask 队列是否为空，非空则先执行 MicroTask 队列中的MicroTask
2. 一个 MicroTask 执行完后，检查有没有下一个 MicroTask ，直到 MicroTask 队列为空，才去执行 Event 队列
3. 在 Evnet 队列取出一个事件处理完后，再次返回第一步，去检查 MicroTask 队列是否为空

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。

异常捕获上传至统计崩溃平台也是应用这个模型，后面会讲到。

2.2 Flutter Widget

介绍完 Dart，我们再看下 Flutter Widget。在 Flutter 中一切皆为 Widget，通过使用 Widget 可以实现页面**整体布局、文本展示、图片展示、手势操作、事件响应**等。



2.2.1 StatelessWidget

StatelessWidget 是一个没有状态的 widget ——没有要管理的内部状态。它通过构建一系列其他小部件来更加具体地描述用户界面，从而描述用户界面的一部分。当我们的页面不依赖 Widget 对象本身中的配置信息以及BuildContext 时，就可以用到无状态组件。例如当我们只需要显示一段文字时。实际上 Icon、Divider、Dialog、Text 等都是 StatelessWidget 的子类。

2.2.2 StatefulWidget

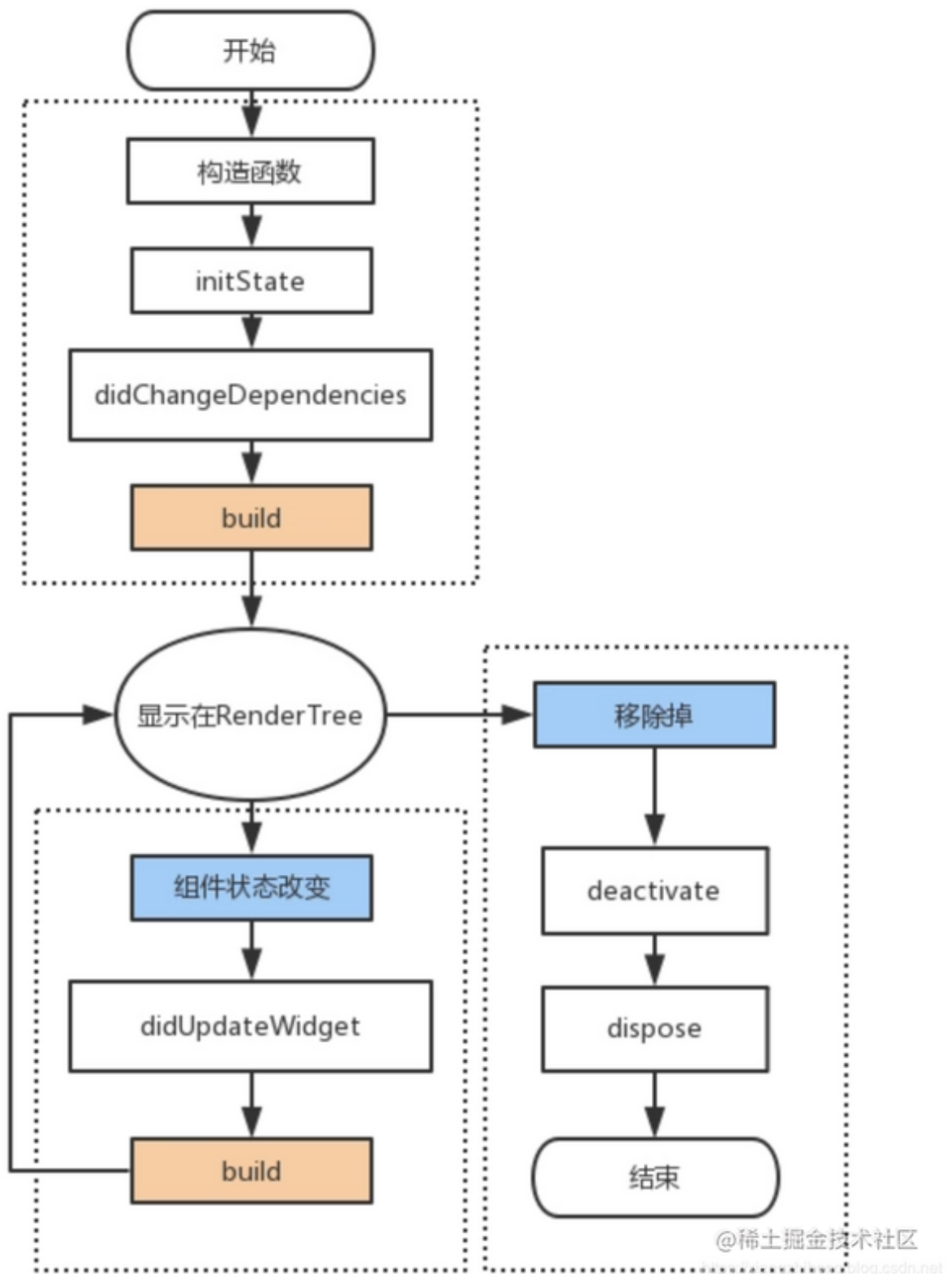
StatefulWidget 是可变状态的 widget。使用 setState 方法管理 StatefulWidget 的状态的改变。调用 setState 通知 Flutter 框架某个状态发生了变化，Flutter 会重新运行 build 方法，应用程序变可以显示最新的状态。

状态是在构建 widget 的时候，widget 可以同步读取的信息，而这些状态会发生变化。要确保在状态改变的时候即使通知 widget 进行动态更改，就需要用到 StatefulWidget。例如一个计数器，我们点击按钮就要让数字加一。在 Flutter 中，Checkbox、FadeImage 等都是有状态组件。

StatefulWidget的生命周期大致可分为三个阶段：

- **初始化**：插入渲染树，这一阶段涉及的生命周期函数主要有 createState、initState、didChangeDependencies 和 build。
- **运行中**：在渲染树中存在，这一阶段涉及的生命周期函数主要有 didUpdateWidget 和 build。
- **销毁**：从渲染树中移除，此阶段涉及的生命周期函数主要有 deactivate 和 dispose。

具体的声明周期调用过程如下：



2.3 StatefulWidget 和 StatelessWidget 的实用场景

在 Flutter 中，组件和页面数据变化是通过 State 驱动的，对于有交互的页面或组件可以继承 StatefulWidget，静态组件或页面可以继承 StatelessWidget。StatelessWidget 没有内部状态，Icon、IconButton 和 Text 都是无状态 widget, 他们都是 StatelessWidget 的子类。StatefulWidget 是

动态的. 用户可以和其交互或者可以随时间改变 (也许是数据改变导致的UI更新)。Checkbox、Radio、Slider、InkWell、Form、TextField 都是 StatefulWidget, 他们都是 StatefulWidget 的子类。

使用 StatefulWidget 还是 StatelessWidget 的**判断依据**：

- 如果用户与widget交互，widget 会发生变化，那么它就是有状态的。
- widget 的状态（state）是一些可以更改的值, 如一个 slider 滑动条的当前值或 checkbox 是否被选中。
- widget的状态保存在一个State对象中, 它和widget的布局显示分离。
- 当widget状态改变时, 调用 setState(), 告诉框架去重绘widget。

三、混合开发 - 整体框架

开发之初我们考虑两个问题：

- 场景1：a,b 两个业务线，都要在 flutter 工程里面开发业务？
- 场景2：M app 有 flutter 工程，这个时候我们 N app 里的 flutter 工程要嵌入到 M app 里我们怎么办？

起初我们希望生成多个产物进行嵌入，通过 Flutter 的线下会议探讨发现这个思路是比较后期的事情，但是也得到了另一个思路将我们的业务进行“**下沉**”，下沉到同一个工程里面进行业务区分，引入组件化的概念进行实践；

3.1 支持多团队开发

Flutter 工程中，通常有以下4种工程类型，下面分别简单概述下：

1. Flutter Application：标准的 Flutter App 工程，包含标准的 Dart 层与 Native 平台层

2. Flutter Module：Flutter 组件工程，仅包含 Dart 层实现，Native 平台层子工程为通过 Flutter 自动生成的隐藏工程

3. Flutter Plugin：Flutter 平台插件工程，包含 Dart 层与 Native 平台层的实现

4. Flutter Package: Flutter 纯 Dart 插件工程，仅包含 Dart 层的实现，往往定义一些公共 Widget

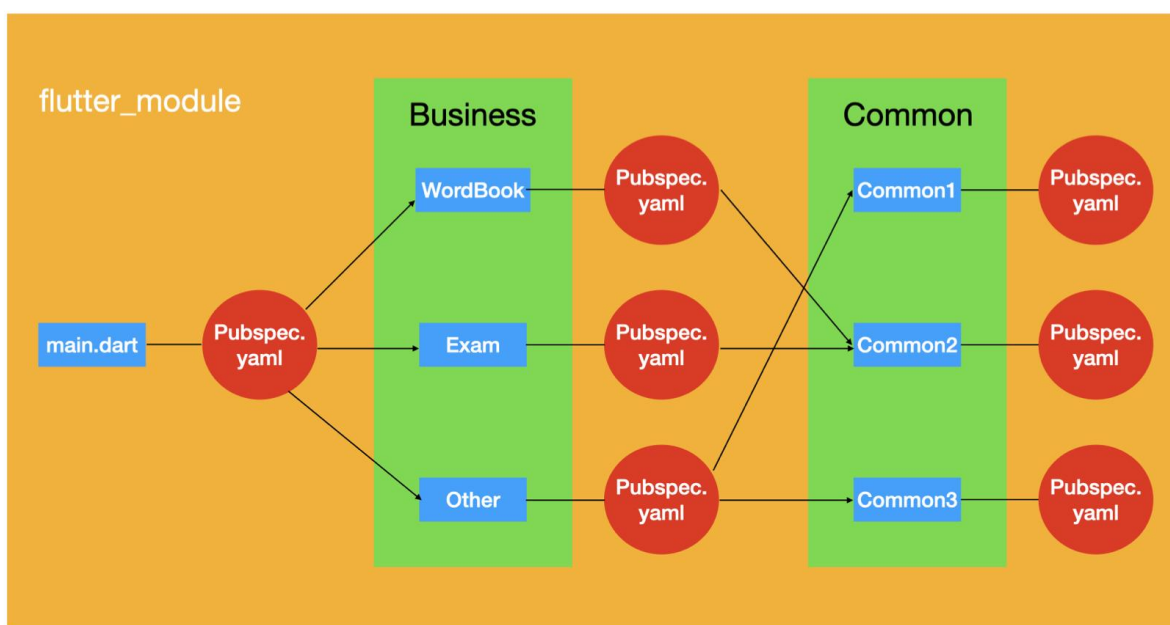
Flutter 工程之间的依赖管理是通过 Pub 来管理的，依赖的产物是直接源码依赖，这种依赖方式和 IOS 中的Pod有点像，都可以进行依赖库版本号的区间限定与 Git 远程依赖 path 等，其中具体声明依赖是在 pubspec.yaml 文件中，其中的依赖编写是基于 YAML 语法，YAML 是一个专门用来编写文件配置的语言，下面是依赖示例：

```
vibration:
  git:
    url: https://github.com/YoudaoMobile/flutter_vibration.git
    ref: 'task/youdao'
```

```
flutter_jsbridge_builder:
  path: ../../Common/flutter_jsbridge_builder
```

所以，通过 **Flutter Plugin / Flutter Package + Pub** 达到解耦的目的

词典Flutter组件化架构图



@稀土掘金技术社区

以 **Flutter Plugin / Flutter Package**为模块开发，原则上我们将工程分为壳工程,业务组件,基础组件；依赖关系为壳工程->业务组件->基础组件，不能依赖倒置，同层之间不能相互引用。

3.2 基础组件沉淀

通过以组件化形式进行开发，通过各个团队业务的不断迭代，逐步沉淀出一套CommonUI的基础Widget组件，方便其他业务和团队扩展使用。

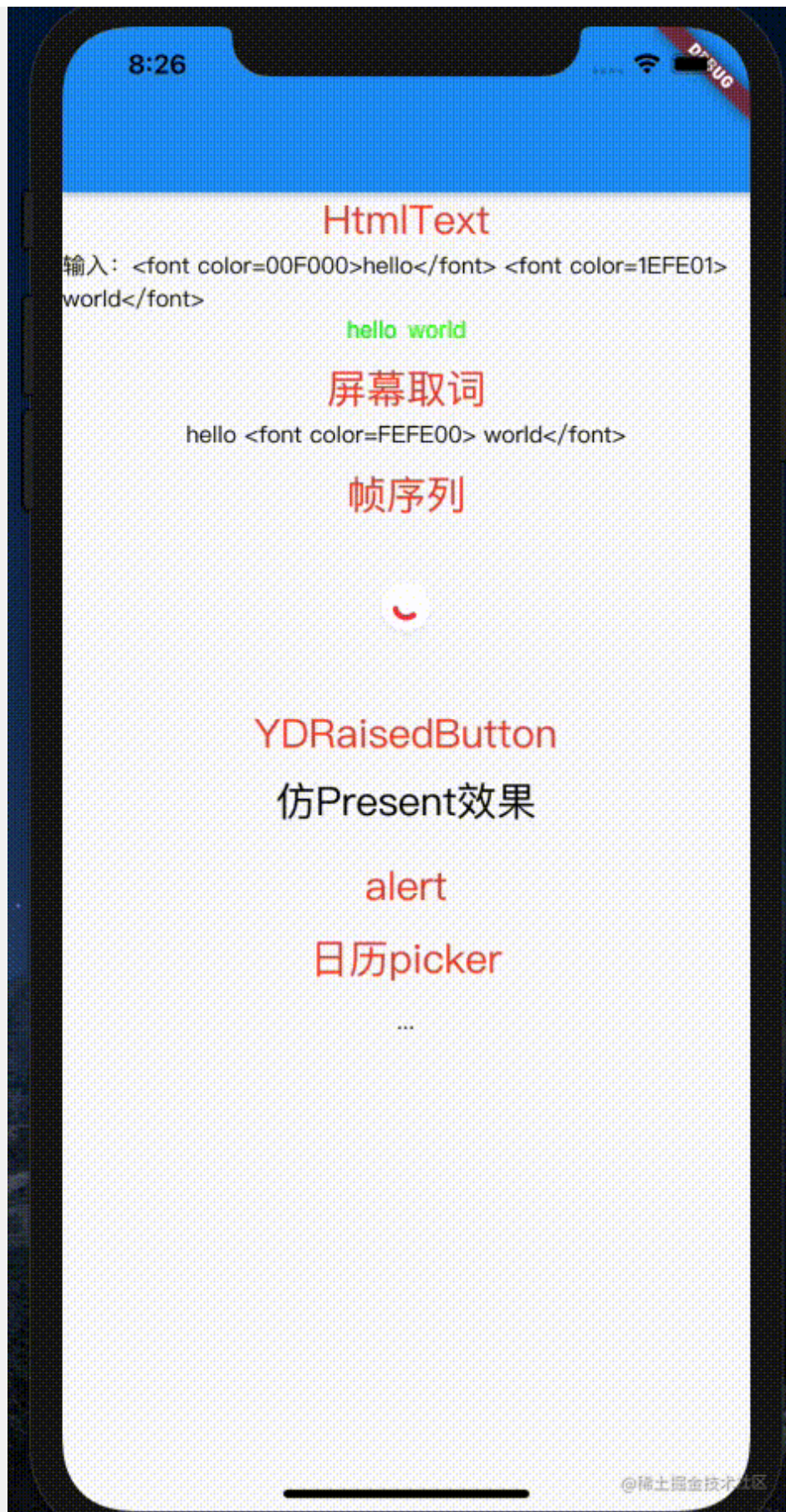
- **button**工具：YDLabelButton，YDRaisedButton
- **字体**工具：YDFontWeight，YDText，YDHtmlText
- **遮罩**工具：YDMaskView
- **loading**工具：YDDefaultLoadingView，YDLoadingView
- **圆角**工具：CornerDecoration
- **模态弹出**工具：YDCupertinoModalPopupRoute
- **点击弹窗**工具：YDCoordinateTap
- **帧序列动画**工具：YDSimpleFrameAnilmage
- ...

YDRaisedButton，YDLabelButton是我们统一遵守有道UI准则自定义的一套button内容

YDText集成了英文屏幕取词，以及解决中日韩同时展示在界面字体展示异常的问题

YDHtmlText我们集成了基于html标签展示进行深层定制，来实现富文本的效果

YDSimpleFrameAnilmage 帧动画播放组件，解决了单纯的图片第一次循环播放会闪烁等问题的播放动画组件 YDCupertinoModalPopupRoute 仿照新的ios模态弹出的效果，支持随手滑动消失的交互方式



3.3 元编程

我们在开发过程发现我们的bridge的内容大多数是相同的，只不过是形参，函数名不同罢了，所以我们打算引入source_gen，来生成bridge层的代码，这样也带来两个好处，一是防止手误，带来的不必要的bug，二是将代码统一

source_gen主要提处理dart源码，可以通过注解生成代码。

大致的流程是通过 source_gen 一个 _Builder ，_Builder 需要生成器 Generator ，之后通过 Generator 去生成代码。

总结一下，在 Flutter 中应用注解以及生成代码仅需一下几个步骤：

1.依赖

```
dev_dependencies:
  source_gen: ^0.9.0
```

2.创建注解

```
class JSBridgeModule {
  final String moduleName;
  final List<String> enumTypeName;
  const JSBridgeModule({this.moduleName : "app", this.enumTypeName : const []});
}
```

3.创建生成器

```
class JSBridgeImplGenerator
  extends GeneratorForAnnotation<JSBridgeModule> {
  JSBridgeImplGenerator() {}
  @override
```



```

Iterable<String> generateForAnnotatedElement (
    Element element, ConstantReader annotation, BuildStep buildStep) {
    if (element is! ClassElement) {
        final name = element.name;
        throw InvalidGenerationSourceError('Generator cannot target `$name`.',
        return _generate(classElement, moduleName, enumTypeName: checkEnumTypeName);
    }
}

```

4.创建Builder

```

Builder getJSBridgeImpGeneratorBuilder(BuilderOptions options) {
    return SharedPartBuilder();}

```

5.编写配置文件

在项目根目录创建 build.yaml 文件，配置各项参数

```

builders:
  JSBridgeImpGeneratorBuilder:
    import: "package:flutter_jsbridge_builder/builder.dart"
  builder_factories: ["getJSBridgeImpGeneratorBuilder"]
  build_extensions: {".dart": ["flutter_jsbridge_builder.g.part"]}
  auto_apply: dependents
  build_to: cache
  applies_builders: ["source_gen|combining_builder"]

```

这样就为我们输出一份模板代码提供了实现的可能

5.4 资源管理

众所周知，flutter图片等资源管理方面，还是处在手动管理阶段，费时费力，所以推荐一款网易严选团队开发的flr插件，flr配合通过AndroidStudio插件，将用于帮助Flutter开发者在修改项目资源后，可

以自动为资源添加声明到 `pubspec.yaml` 以及生成集中在一起的资源路径文件，Flutter开发者可以在代码中通过资源ID函数的方式应用资源。

通过建立起一个自动化的服务来监听和管理资源变化，之后将变化的资源同步到 `pubspec.yaml` 和对应的资源文件当中，也支持文本，字体资源，后续我们也和flr的团队支持黑暗模式的计划。

地址： github.com/Fly-Mix/flr...

3.5 异常捕获上传

在 flutter 简介里面我们介绍了 dart 的线程模型

事件队列模型过程：

1. 先检查 MicroTask 队列是否为空，非空则先执行 MicroTask 队列中的 MicroTask
2. 一个 MicroTask 执行完后，检查有没有下一个 MicroTask，直到 MicroTask 队列为空，才去执行 Event 队列
3. 在 Event 队列取出一个事件处理完后，再次返回第一步，去检查 MicroTask 队列是否为空

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。

Flutter 框架为我们在很多关键的方法进行了**异常捕获**。在发生异常时，错误是通过 `FlutterError.reportError` 方法上报的，其中 `onError` 是 `FlutterError` 的一个静态属性，我们重写 `onError` 就可以捕获异常了；但是还有一些异步异常是需要我们通过 **Zone** 方法来捕获的，整理代码如下：

```
FlutterError.onError = (FlutterErrorDetails details) {
  crashReporter(details);
};
runZoned(() => runApp(YDApplication()),
  onError: (Object obj, StackTrace stack) {
    crashReporterOnZone(obj, stack);
  }
);
```

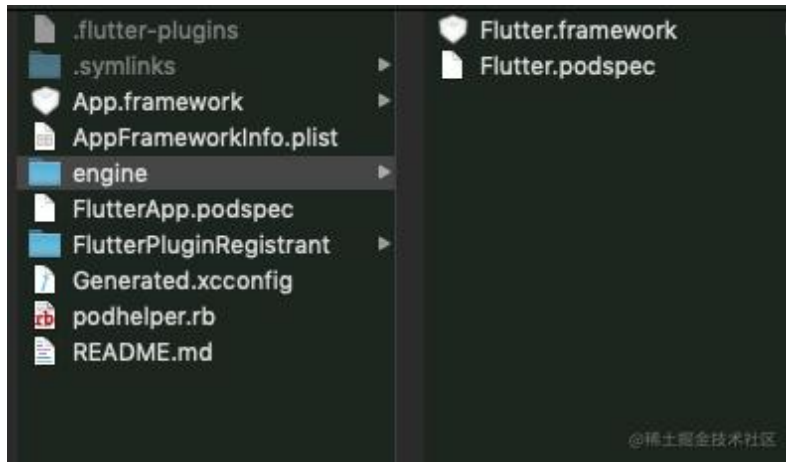
@稀土掘金技术社区

四、产物介绍

4.1 介绍

1.12是个分水岭，在这之前安卓打包方式有所不同，并且iOS官方也提供一些命令也来支持打不同的包

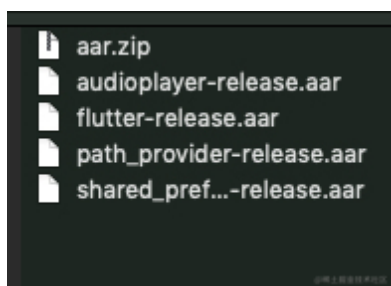
- iOS产物组成



通过 flutter build ios --release 来得到产物，然后 flutter-plugins 里面记录各种 plugin 的位置 copy 过来，放在 .symlinks 文件夹下

app.framework:代码数据段+图片 **flutter.framework**:engine+channel+... **FlutterPluginRegistrant**:源码，一些flutter自身的bridge **podhelper.rb**:通过flutter-plugins里的bridge列表循环的将bridge填到pod中，在宿主工程通过pod引入

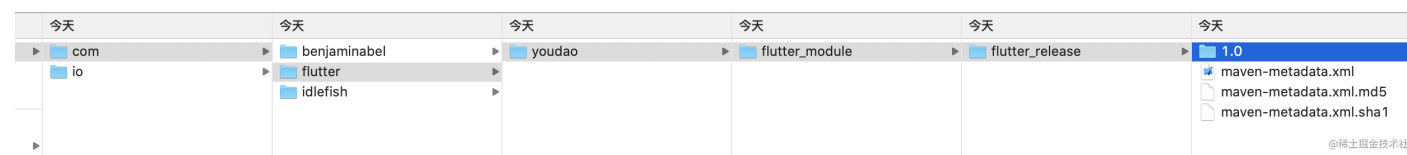
- android产物组成 flutter1.0



进入 flutter 工程的 .android 文件夹执行 ./gradlew assembleRelease 就会打出一个 flutter-release.aar 的包，但是还有 path_provider,share_preference,audioplayer 等官方插件我们也需要 copy出来，这里

我们发现 flutter 工程目录下面有.flutter-plugins 这个文件，这个文件记录着你当前 flutter所使用的官方插件的文件位置，我们通过 shell 读取文件位置，找到对应的 aar 集中到一起。

- ** android flutter 1.12以后**



flutter 1.12打包执行flutter build aar --no-debug --no-profile来得到.

4.2 打包问题汇总

在flutter1.0版本进入android文件夹执行./gradlew assembleRelease会得到aar产物，但是此时的aar 嵌入进去run起来会报错，错误信息是缺少一个.dat文件，根据官方的issue和对源码的思考，讨论结果是里assets下面少一个flutter_assets文件内容，事实上在io/flutter.jar可以看到，但是flutter还是会去assets文件夹下去找，导致嵌入Android失败。解决办法，从apk里copy一份flutter_asset放到aar里

在flutter 1.12版本官方提供aar产物命令，但是工程中引入官方库（shared_preferences）的时候会执行命令失败，原因是第三方会带上macos和web的package，但是这个package不带android文件的内容，解决办法：通过修改官方sdk对其android文件夹进行兼容。

4.3 打包机问题汇总

在打包机配置完flutter环境，需要在Jenkins的节点配置将flutter path添加到PATH当中，否则flutter 命令执行失败，以及ios 打包flutter build ios --release会因为code sign没有权限的问题失败，尽量用 flutter build ios --release --no-codesign来得到环境

五、遇到的问题

5.1 ios端存在的问题

5.1.1 混合栈 boost出现的问题

首先感谢咸鱼团队，提供了混合栈的一种方案，我们从flutter1.9升级到1.12过程中，遇到不少的问题和麻烦。

1.生命周期多次回调

在1.9的版本中，ContainerLifecycle.Appear 方法会回调两次，导致依赖生命周期操作重复，在 ios 这边是在 viewdidappear 的时候会发通过 channel 发 didShowPageContainer 的消息，调用 nativeContainerDidShow，然后在 TransitionBuilder 的方法再去调用一次。

onPageStart 然后再去调用 nativeContainerDidShow，就会导致两次触发，android 也是在 onAppear 的方法上重复上述的操作。

解决办法就是去掉其中一个。

2.升级1.12之后，切前后台的crash问题

这个问题版本有很多，得考虑业务场景，我们这里是先模态出一个NavigationViewController，然后在这个NavigationViewController基础上进行push和pop操作，然后我们在全局提供一个回到模态之前ViewController的操作。在全局回退的过程中，我们清掉了native的栈，然后在native的任意vc，切前后台后crash。但是在1.9版本时并没有发现此类问题。

crash的原因是在1.12的版本中 FlutterEngine自身加了surfaceUpdated的操作，当你整个退出后没有正确的处理，导致FlutterEngine认为你的页面上还存在着Flutter页面，进行刷新创建工作，就crash了。当然这个是我们这个业务场景总结出的crash的原因，据说还有其他版本crash问题，欢迎其他朋友补充。

解决办法是在全局回退的过程中循环调用close方法将栈里的vc退出。

5.1.2 多语言显示异常

当界面同时显示在韩语/日语 与中文时，界面展示异常

官方issue: [github.com/flutter/flu...](https://github.com/flutter/flutter/issues/25284)

解决方式有三种:

1. 增加字体 ttf ，全局指定改字体显示。
2. TextStyle 属性指定

```
fontFamilyFallback: ["PingFang SC", "Heiti SC"]
```

可以封装成一个widget

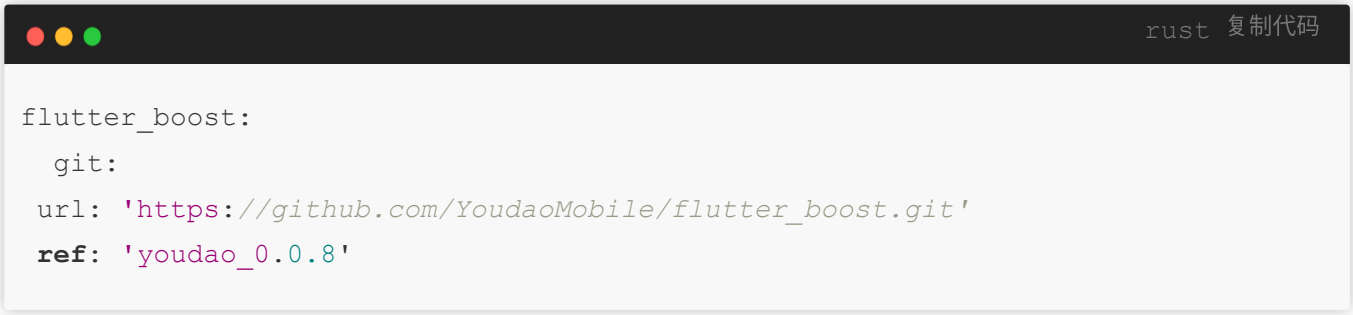
3. 修改主题下所有 ThemeData 的 fontFamilyFallback

```
getThemeData() {  
  var themeData = ThemeData(  
    primarySwatch: primarySwatch  
  );  
  
  var result = themeData.copyWith(  
    textTheme: confirmTextTheme(themeData.textTheme),  
    accentTextTheme: confirmTextTheme(themeData.accentTextTheme),  
    primaryTextTheme: confirmTextTheme(themeData.primaryTextTheme),  
  );  
  return result;  
}  
  
/// 处理 ios 上, 同页面出现韩文和简体中文, 导致的显示字体异常  
confirmTextTheme(TextTheme textTheme) {  
  getCopyTextStyle(TextStyle textStyle) {  
    return textStyle.copyWith(fontFamilyFallback: ["PingFang SC", "Heiti SC"]);  
  }  
  
  return textTheme.copyWith(  
    display4: getCopyTextStyle(textTheme.display4),  
    display3: getCopyTextStyle(textTheme.display3),  
    display2: getCopyTextStyle(textTheme.display2),  
    display1: getCopyTextStyle(textTheme.display1),  
  );  
}
```

```
headline: getCopyTextStyle(textTheme.headline),
title: getCopyTextStyle(textTheme.title),
subhead: getCopyTextStyle(textTheme.subhead),
body2: getCopyTextStyle(textTheme.body2),
body1: getCopyTextStyle(textTheme.body1),
caption: getCopyTextStyle(textTheme.caption),
button: getCopyTextStyle(textTheme.button),
subtitle: getCopyTextStyle(textTheme.subtitle),
overline: getCopyTextStyle(textTheme.overline),
);
}
```

5.2 双端存在的问题

flutter pub get失败。Flutter 项目在引用第三库时，在pub会选择使用 git 引用，如：



```
flutter_boost:
  git:
    url: 'https://github.com/YoudaoMobile/flutter_boost.git'
    ref: 'youdao_0.0.8'
```

会报 pub get fail 的问题

在下载包的过程中出现问题，下次再拉包的时候，在.pub_cache内的 git 可能是空目录，导致 flutter packages get 的时候异常。

所以你需要清除掉 .pub_cache 内的 git 的异常目录或者执行flutter cache repair ，之后重新执行 flutter packages get 。

5.3 channel 通信

Flutter定义了三种不同类型的 Channel，它们分别是

- **BasicMessageChannel**：用于传递字符串和半结构化的信息。
-

- **MethodChannel**: 用于传递方法调用 (method invocation) 。
-
- **EventChannel**: 用于数据流 (event streams) 的通信。

其中channel有个很重要的变量codec；Codec官方定义了两种Codec：MessageCodec和MethodCodec

其中MessageCodec有4种不同的种类：BinaryCodec；StringCodec；JSONMessageCodec；StandardMessageCodec

起初我们使用 MethodChannel 来建立通信，但是使用过程中遇到大内存的传递耗时很长的问题，我们通过一系列的实验和官方文档的指引，当需要传递大内存数据块时，使用 BasicMessageChannel 以及 BinaryCodec 可以解决问题。

以下是**实验内容**：

实验机型：iphone 7 ios 13.7系统

实验数据：开发者可以自行模拟1M-2M左右的数据进行测试，由于涉及到真实数据 这里就不放出来了。

实验结果：

BinaryCodec

实验次数	1	2	3	4	5
传输时间 (s)	0.112	0.003	0.006	0.005	0.004

JSONMessageCodec

实验次数	1	2	3	4	5
传输时间 (s)	0.162	0.121	0.149	0.162	0.163

StringCodec

实验次数	1	2	3	4	5
传输时间 (s)	0.021	0.064	0.035	0.033	0.037

StandardMessageCodec

实验次数	1	2	3	4	5
传输时间 (s)	0.075	0.034	0.052	0.061	0.053

从实验结果上看，传输效率最优的是BinaryCodec，当然选择什么样的code根据项目的需求来定才是最合理的

BinaryCodec>StringCodec>StandardMessageCodec>JSONMessageCodec

5.4 长列表优化

5.4.1 列表内容项长短不一

当我们实现类似上面的页面，item 高度不一的思路肯定是类似以下的代码，但是当我们达到一定的数量级的情况下，发现内存占用的十分严重，导致有些需求就实现不了，比如说支持滚动条快速定位;究其原因CustomScrollView 初始化就加载了很多的 widget。

```
js 复制代码

List<Widget> list = [];
for (int i = 0; i < 10000; i++) {
  list.add(SliverToBoxAdapter(child: Container(height:30,color: Colors.red,)),);
  list.add(SliverFixedExtentList(delegate: SliverChildBuilderDelegate((context, index) {
    return Container(child: Text(index.toString()),);
  },childCount: 50), itemExtent: 50)
  );
}
CustomScrollView(slivers: list,);
```

但是假如都是同样的 itemExtent，滚动效率，内存表现都是良好的；因为事先告诉好高度，而不是依赖 widget自身的 layout 计算效率就高了很多， 比如说以下的代码：

```
js 复制代码

List<Widget> list = [];

list.add(SliverFixedExtentList(delegate: SliverChildBuilderDelegate((context, index) {
  return Container(child: Text(index.toString()),);
},childCount: 20000), itemExtent: 50)
);
CustomScrollView(slivers: list,);
```

如何两者兼得呢？

我们决定自定义SliverFixedExtentList, SliverFixedExtentList返回的RenderObject是RenderSliverFixedExtentBoxAdaptor，我们将RenderSliverFixedExtentBoxAdaptor重新设计下。

RenderSliverFixedExtentBoxAdaptor原先设计的思路是通过scrolloffset除以itemExtent计算出当前的index（itemExtent是写死的所以直接除），SliverConstraints可以拿到他的remainingCacheExtent也就是cacheExtent加上滚动可见区域，也就可以拿到lastIndex，在滚动的过程中不断的释放和创建。我们改写的思路如下：

- 1.SliverFixedExtentList在createRenderObject和updateRenderObject的时候将每个元素的位置重新计算缓存。
- 2.然后重写performLayout方法，通过全局的first和last索引拿到元素的位置和scrolloffset进行比较，得到新的first和last索引，不断的调整。
- 3.将计算好的约束布局传入子布局。

大致的思路就是这样，接口层面我们设计成这个样子，以下是调用示例：

```
js 复制代码

YDSliverFixedExtentList(
  delegate: SliverChildBuilderDelegate((context, int index) {
    if (index > wordList.length - 1){
      return Container(color: Colors.transparent,);
    }

    YDWBListBaseModel model = wordList[index];
    if (model is YDWBListHeaderModel) {
      return buildSusWidget(model.title);
    } else if (model is YDWBListItemModel) {
      return buildListSingleItem(index, wordList[index], onMoveTap, onDeleteTap);
    } else {
      return Container();
    }
  },
  childCount: wordList.length + 1,
),
itemHeightDelegate: (index){
  if (index > wordList.length - 1){
    return 60;
  }

  var model = wordList[index];
  if (model is YDWBListHeaderModel) {
    return kItemHeaderHeight;
  }
}
```

```

} else if (model is YDWBListItemModel) {
    return kItemHeight;
} else {
    return 60;
}
},
itemIndexDelegate: (startIndex, endIndex){
    firstIndex = startIndex;
    lastIndex = endIndex;
},

```

5.4.2 如何获取当前展示列表索引

ios 开发都知道，我们的 TableView 是有代理来知道我们当前页面展示的 Cell 的索引，但是在 Flutter 里我们怎么办呢？

- **思路1：**给每个 item 加上 GlobalKey，然后放在 model 中，然后滚动的过程中利用去找循环遍历 model 的 GlobalKey，通过 GlobalKey 找到对应的 RenderObject，RenderObject 存在着位置坐标等信息，通过此信息可以比较计算出 key 对应的 RenderObject 是否展示界面

代码如下：

```

double y=model.key.currentContext.findRenderObject().getTransformTo(null).getTra
double height=model.key.currentContext.findRenderObject().paintBounds.size.height

```

然后找到对应绑定的model

- **思路2：**如果是实时获取展示的索引可能上述思路不太合适，可能每次都需要在存放 model 里的数组去找，当然也可以在思路1的基础上进行算法优化，暂存当前展示的 index，来做下次起始寻找的index，减少循环次数。

不过，接下来介绍的是，另一种办法，改写 SliverChildBuilderDelegate，在 SliverChildBuilderDelegate里面的didFinishLayout里会返回它的firstIndex和lastIndex，但是要注意此时返回的是加了cacheExtent的firstIndex，所以可能比实际展示的要小，所以可以结合思路一进行精确定位

```

class MySliverChildBuilderDelegate extends SliverChildBuilderDelegate {
  final int markIndex;
  MySliverChildBuilderDelegate(
    this.markIndex,
    Widget Function(BuildContext, int) builder, {
      int childCount,
      bool addAutomaticKeepAlives = true,
      bool addRepaintBoundaries = true,
    }) : super(builder,
      childCount: childCount,
      addAutomaticKeepAlives: addAutomaticKeepAlives,
      addRepaintBoundaries: addRepaintBoundaries,
    );

  @override
  void didFinishLayout(int firstIndex, int lastIndex) {
    debugPrint('pre' + 'didFinishLayout firstIndex: $firstIndex, lastIndex: $lastIndex');
    if (firstIndex == 0 && lastIndex == 0) {
      return;
    }
    YDBaseListEvent.notifyIndexChange({"firstIndex": markIndex + firstIndex, "lastIndex": markIndex + lastIndex});
    debugPrint(mark + 'didFinishLayout firstIndex: $firstIndex, lastIndex: $lastIndex');
  }
}

```

- **思路3：** 可以参考长列表优化，将SliverFixedExtentList改写暴露对应返回index的接口

六、后续计划

我们后续计划升级到 flutter2.0，但是目前来看 2.0 还存在问题，如果 2.0 真的彻底的解决多引擎复用的问题，我们也会尝试去除 boost 的管理机制，根据 flutter.cn/posts/flutt...，在多个引擎复用的视频章节，于潇分析了多引擎复用的内存增长的问题，主要在

- 线程

- GPU资源
- Skia Context
- 字形
- Dart Isolate

这五部分，每起一个 engine 之后就会起 3 个新的操作系统的线程，每个线程都是有成本的尤其是在 ios 上，在 2.0 版本上都合并在一起了；另一部分 GPU 资源, Skia Context 就包含了 opengl 的 context, metal context, metal buffer, shader program, skia program, 为了提高使用启动将 GPU 资源和 Skia Context 的内容做共享；字形的大小都会有缓冲的，假如不加以利用的话也造成一定的浪费；dart Isolate 事实上每次创建 engine 都会重新创建一个，2.0 版本也做了一个共享。

结果也是比较客观，优化的效果比较明显，10 次的启动不升反降，40M 变成了 35M。有兴趣的可以试下，github.com/flutter/sam...，但是对于 flutter 团队来说 2.0 版本只是解决了内存问题，还存在其他的问题，主要是以下几方面：

- 只支持 AOT，不支持 debug
- ios IOSurface 卸载，fluttervc 没有解除但是又被覆盖了的话，它的 metal layer 没有释放 IOSurface，会影响混合栈的场景，目前的办法是覆盖的时候需要手动把 flutterview 去掉
- 不支持数据共享
- 不支持内存共享
- platform View 不支持
- 只支持一个 snapshot

七、结束语

我们在单词本和听力等模块进行 flutter 落地的探索，在前期实践过程中，碰到了很多问题，但总体来说还处于可控的状态；前期把各种困难都解决后，后面业务再此基础上进行开发会顺畅很多，效率会提升很多，这个也是 flutter 期望带给我们的一次开发，多端运行。但是另一方面希望开发者们在落地过程中，更为慎重些，多多实践，提前发现提前解决，毕竟存在处理不好的情况，还需要推动官方或者生态提供更好的解决办法。

未来期望 flutter 以及社区在平台一致性以及混合栈，内存，键盘，音视频等具体问题上持续发力，我们也会进一步的探索 flutter 在业务上更多实现的可能。感谢观看。

以上内容仅代表个人观点，如果内容或者实验数据存在疑问和问题，欢迎大家批评指正，一起学习，一起成长。

本内容仅代表个人观点，不代表网易，仅供内部分享传播，不允许以任何形式外泄，否则追究法律责任。

分类：

前端

标签：

前端

Flutter

相关课程



TypeScript 全面进阶指南

林不渡

3204购买

¥49.9



Webpack5 核心原理与应用实践

范文杰

2840购买

¥59.9

热门评论

snowKing

1年前

请教一下如何解决业务模块之间的依赖，以及各个业务模块如何进行独立的开发和测试呢？比如通常的业务模块中都要依赖登录授权业务模块，而又不能相互引用，如何处理呢

1 2

chedechao333

1年前

1.第一个问题：请教一下如何解决业务模块之间的依赖，以及各个业务模块如何进行独立的开发和测试呢？

首先理解Flutter4种工程类型，其中package为纯dart工程，plugin为dart+native的代码的工程，各个业务方包装成package或plugin，然后将你们的工程进行分层分别为壳工程 - 业务组件 - 基础组件，其中依赖关系为同层之间不能相互依赖，上层可以依赖下