

32 | 适配国际化，除了多语言我们还需要注意什么？

2019-09-10 陈航

Flutter核心技术与实战

[进入课程 >](#)



讲述：陈航

时长 11:24 大小 10.46M



你好，我是陈航。今天，我们来聊聊 Flutter 应用的国际化。

借助于 App Store 与 Google Play，我们能够把应用发布到全世界的任何一个应用商店里。应用的（潜在）使用者可能来自于不同国家、说着不同的语言。如果我们想为全世界的使用者提供统一而标准的体验，那么首先就需要让 App 能够支持多种语言。而这一过程，一般被称为“国际化”。

提起国际化，你可能会认为这等同于翻译 App 内所有用户可见的文本。其实，这个观点不够精确。**更为准确地描述国际化的工作职责，应该是“涉及语言及地区差异的适配改造过程”。**

比如，如果我们要显示金额，同样的面值，在中国会显示为 ¥ 100，而在美国则会显示为 \$100；又比如，App 的引导图，在中国我们可能会选用长城作为背景，而在美国我们则可能会选择金门大桥作为背景。

因此，对一款 App 做国际化的具体过程，除了翻译文案之外，还需要将货币单位和背景图等资源也设计成可根据不同地区自适应的变量。这也就意味着，我们在设计 App 架构时，需要提前将语言与地区的差异部分独立出来。

其实，这也是在 Flutter 中进行国际化的整体思路，即语言差异配置抽取 + 国际化代码生成。而在语言差异配置抽取的过程中，文案、货币单位，以及背景图资源的处理，其实并没有本质区别。所以在今天的分享中，我会以多语言文案为主，为你讲述在 Flutter 中如何实现语言与地区差异的独立化，相信在学习完这部分的知识之后，对于其他类型的语言差异你也能够轻松搞定国际化了。

Flutter i18n

在 Flutter 中，国际化的语言和地区的差异性配置，是应用程序代码的一部分。如果要在 Flutter 中实现文本的国际化，我们需要执行以下几步：

首先，实现一个 `LocalizationsDelegate`（即翻译代理），并将所有需要翻译的文案全部声明为它的属性；

然后，依次为需要支持的语言地区进行手动翻译适配；

最后，在应用程序 `MaterialApp` 初始化时，将这个代理类设置为应用程序的翻译回调。

如果我们中途想要新增或者删除某个语系或者文案，都需要修改程序代码。

看到这里你会发现，如果我们想要使用官方提供的国际化方案来设计 App 架构，不仅工作量大、繁琐，而且极易出错。所以，要开始 Flutter 应用的国际化道路，我们不如把官方的解决方案扔到一边，直接从 **Android Studio 中的 Flutter i18n 插件开始学习**。这个插件在其内部提供了不同语言地区的配置封装，能够帮助我们自动地从翻译稿生成 Dart 代码。

为了安装 Flutter i18n 插件，我们需要打开 Android Studio 的 Preference 选项，在左边的 tab 中，切换到 Plugins 选项，搜索这个插件，点击 install 即可。安装完成之后再重启 Android Studio，这个插件就可以使用了。

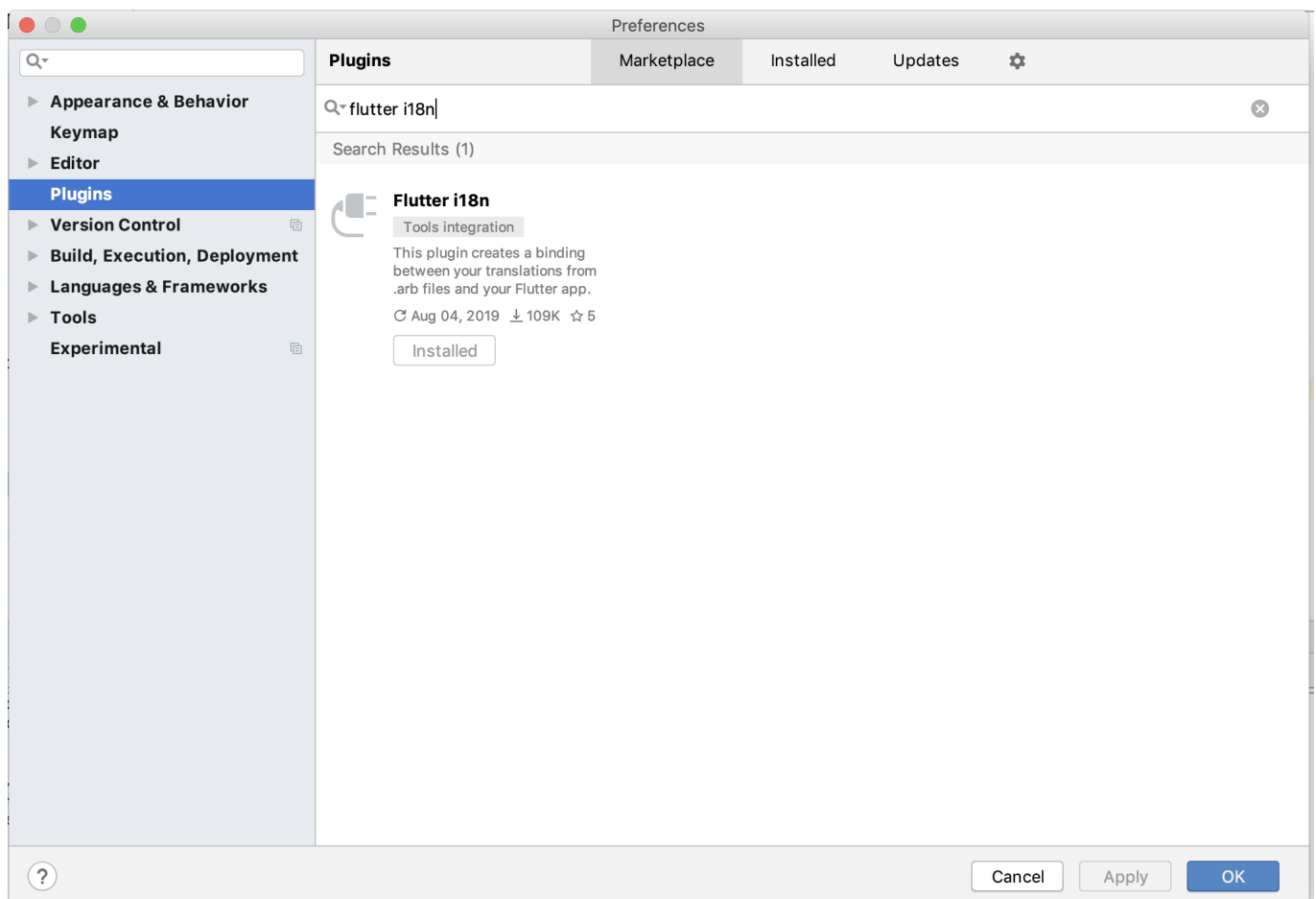



图 1 Flutter i18n 插件安装

Flutter i18n 依赖 flutter_localizations 插件包，所以我们还需要在 pubspec.yaml 文件里，声明对它的依赖，否则程序会报错：

 复制代码

```
1 dependencies:
2   flutter_localizations:
3     sdk: flutter
```

这时，我们会发现在 res 文件夹下，多了一个 values/strings_en.arb 的文件。

arb 文件是 JSON 格式的配置，用来存放文案标识符和文案翻译的键值对。所以，我们只要修改了 res/values 下的 arb 文件，i18n 插件就会自动帮我们生成对应的代码。

strings_en 文件，则是系统默认的英文资源配置。为了支持中文，我们还需要在 values 目录下再增加一个 strings_zh.arb 文件：

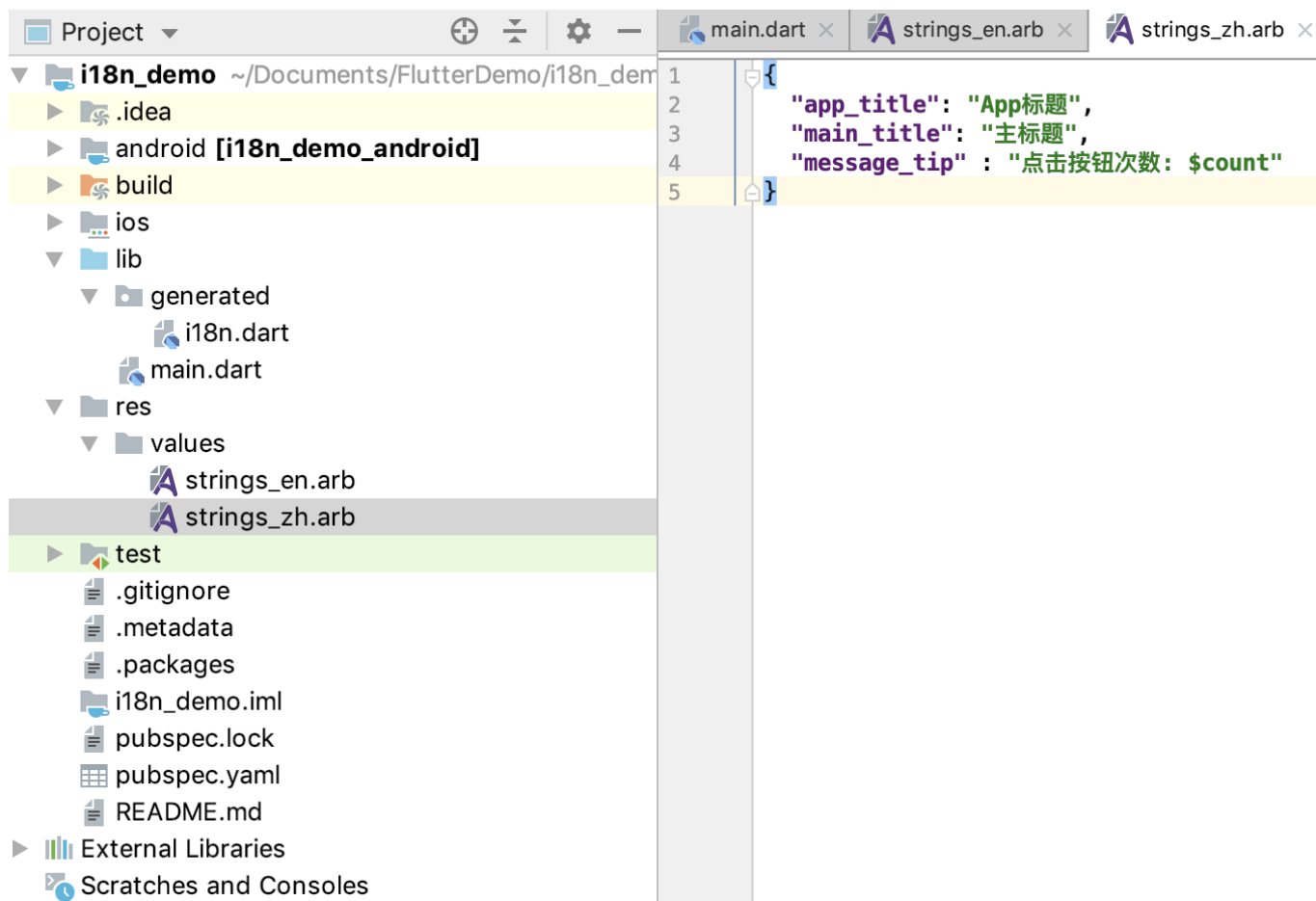


图 2 arb 文件格式

试着修改一下 strings_zh.arb 文件，可以看到，Flutter i18n 插件为我们自动生成了 generated/i18n.dart。这个类中不仅以资源标识符属性的方式提供了静态文案的翻译映射，对于通过参数来实现动态文案的 message_tip 标识符，也自动生成了一个同名内联函数：

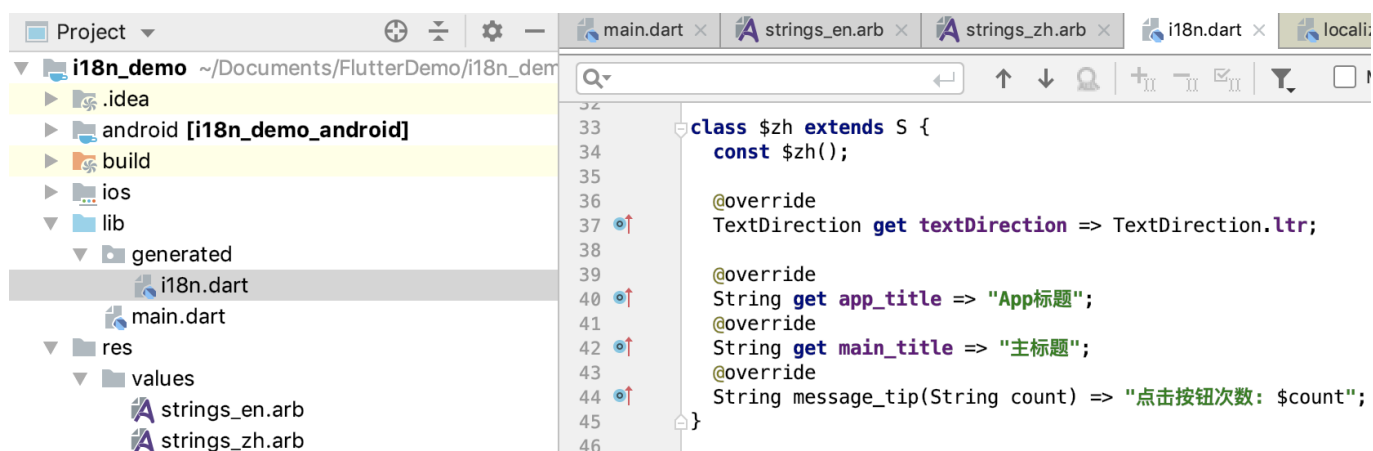


图 3 Flutter i18n 插件自动生成代码

我们把 strings_en.arb 继续补全，提供英文版的文案。需要注意的是，i18n.dart 是由插件自动生成的，每次 arb 文件有新的变更都会自动更新，所以切忌手动编辑这个文件。

接下来，我们以 **Flutter 官方的工程模板，即计数器 demo** 来演示如何在 Flutter 中实现国际化。


在下面的代码中，我们在应用程序的入口，即 MaterialApp 初始化时，为其设置了支持国际化的两个重要参数，即 localizationsDelegates 与 supportedLocales。前者为应用的翻译回调，而后者则为应用所支持的语言地区属性。

S.delegate 是 Flutter i18n 插件自动生成的类，包含了所支持的语言地区属性，以及对应的文案翻译映射。理论上，通过这个类就可以完全实现应用的国际化，但为什么我们在配置应用程序的翻译回调时，除了它之外，还加入了 GlobalMaterialLocalizations.delegate 与 GlobalWidgetsLocalizations.delegate 这两个回调呢？

这是因为 Flutter 提供的 Widget，其本身已经支持了国际化，所以我们没必要再翻译一遍，直接用官方的就可以了，而这两个类则就是官方所提供的翻译回调。事实上，我们刚才在 pubspec.yaml 文件中声明的 flutter_localizations 插件包，就是 Flutter 提供的翻译套装，而这两个类就是套装中的著名成员。

在完成了应用程序的国际化配置之后，我们就可以在程序中通过 S.of(context)，直接获取 arb 文件中翻译的文案了。

不过需要注意的是，**提取翻译文案的代码需要在能获取到翻译上下文的前提下才能生效，也就是说只能针对 MaterialApp 的子 Widget 生效。**因此，在这种配置方式下，我们是无法对 MaterialApp 的 title 属性进行国际化配置的。不过，好在 MaterialApp 提供了一个回调方法 onGenerateTitle，来提供翻译上下文，因此我们可以通过它，实现 title 文案的国际化：

 复制代码

```
1 // 应用程序入口
2 class MyApp extends StatelessWidget {
3   @override
4   Widget build(BuildContext context) {
5     return MaterialApp(
6       localizationsDelegates: const [
7         S.delegate, // 应用程序的翻译回调
8         GlobalMaterialLocalizations.delegate, // Material 组件的翻译回调
```




```

9      GlobalWidgetsLocalizations.delegate,// 普通 Widget 的翻译回调
10    ],
11    supportedLocales: S.delegate.supportedLocales,// 支持语系
12    //title 的国际化回调
13    onGenerateTitle: (context){
14      return S.of(context).app_title;
15    },
16    home: MyHomePage(),
17  );
18 }
19 }

```

应用的主界面文案的国际化，则相对简单得多了，直接通过 `S.of(context)` 方法就可以拿到 `arb` 声明的翻译文案了：

 复制代码

```

1 Widget build(BuildContext context) {
2   return Scaffold(
3     // 获取 appBar title 的翻译文案
4     appBar: AppBar(
5       title: Text(S.of(context).main_title),
6     ),
7     body: Center(
8       // 传入 _counter 参数，获取计数器动态文案
9       child: Text(
10        S.of(context).message_tip(_counter.toString())
11      )
12    ),
13    floatingActionButton: FloatingActionButton(
14      onPressed: _incrementCounter,// 点击回调
15      tooltip: 'Increment',
16      child: Icon(Icons.add),
17    ),
18  );
19 }

```

在 Android 手机上，分别切换英文和中文系统，可以看到，计数器应用已经正确地处理了多语言的情况。



10:28

DEBUG

My Main Title

You have pushed the button many times: 0

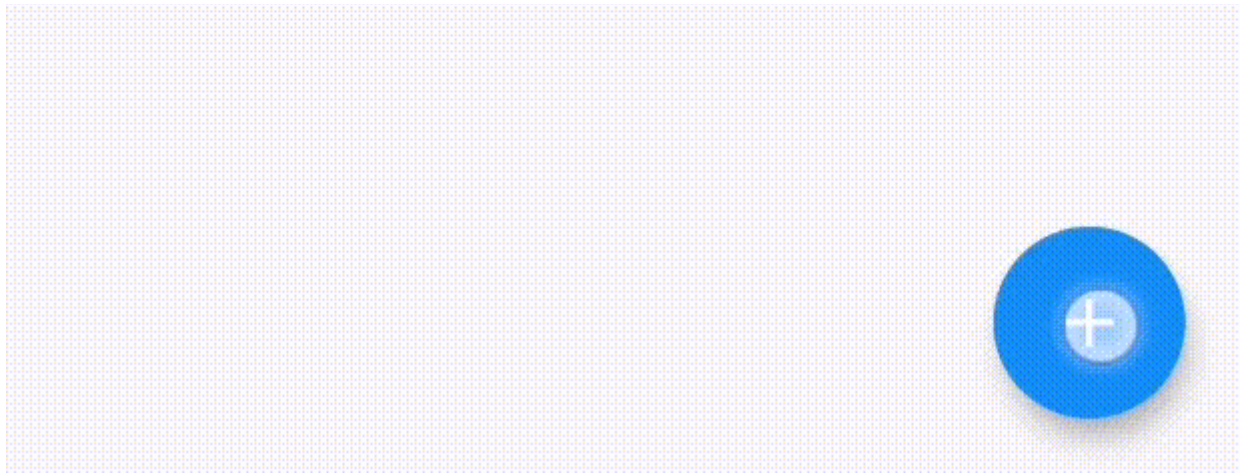


图 4 计数器示例 (Android 英文系统)



29

DEBUG

主标题

点击按钮次数: 0

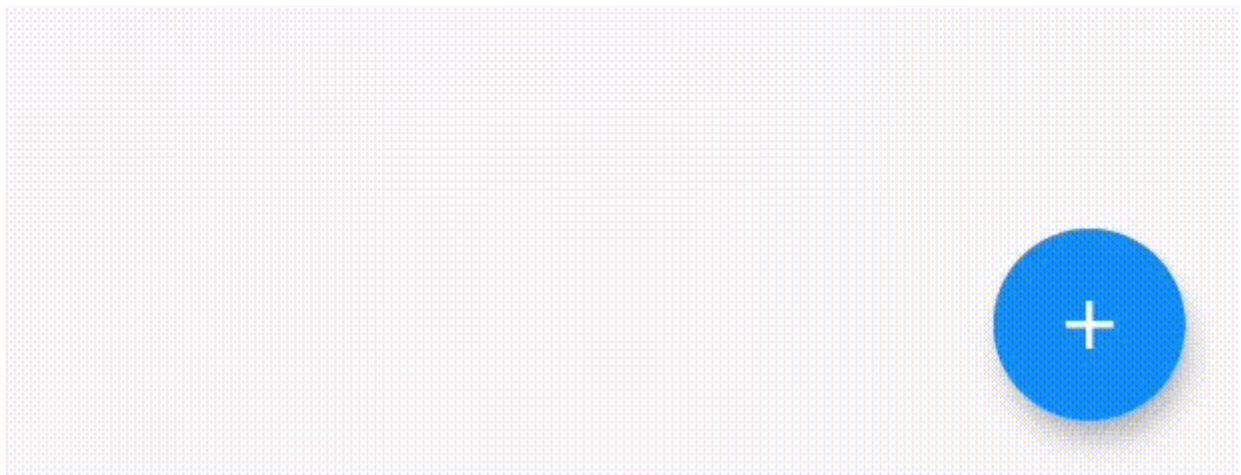


图 5 计数器示例 (Android 中文系统)

由于 iOS 应用程序有一套自建的语言环境管理机制，默认是英文。为了让 iOS 应用正确地支持国际化，我们还需要在原生的 iOS 工程中进行额外的配置。我们打开 iOS 原生工程，切换到工程面板。在 Localization 这一项配置中，我们看到 iOS 工程已经默认支持了英文，所以还需要点击 “+” 按钮，新增中文：

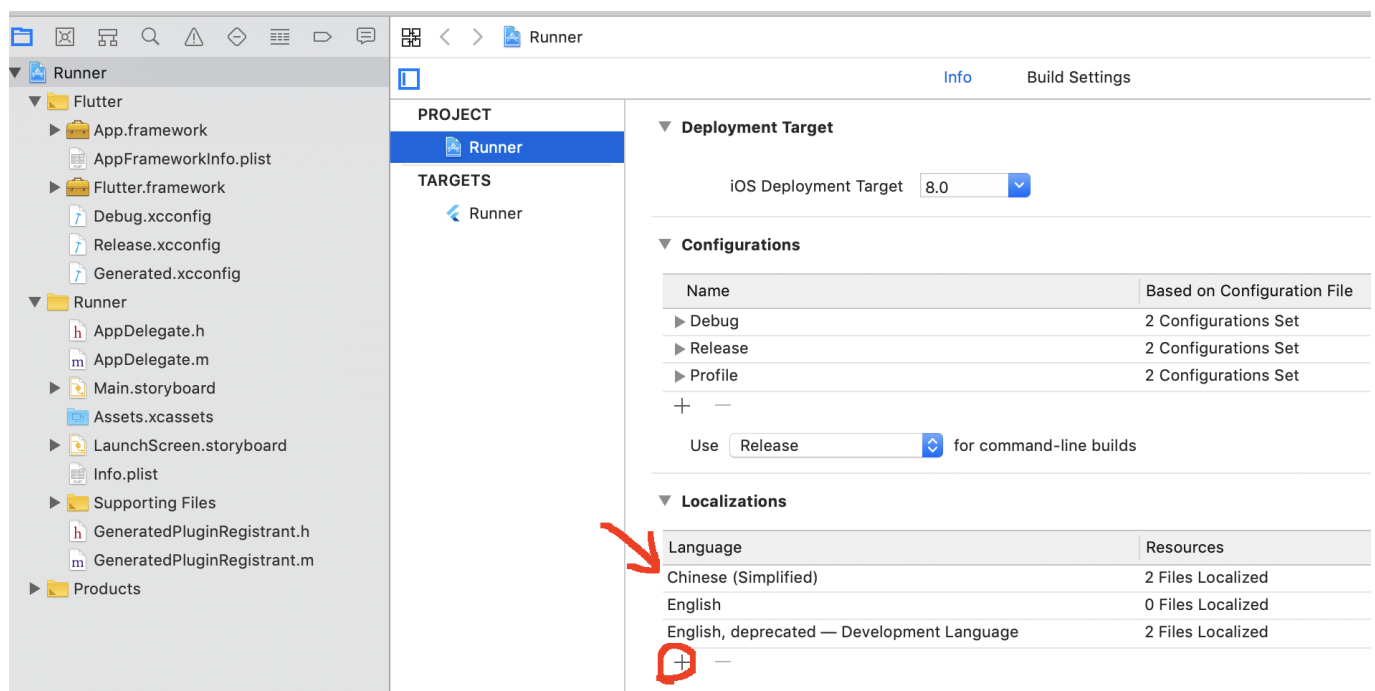


图 6 iOS 工程中文配置

完成 iOS 的工程配置后，我们回到 Flutter 工程，选择 iOS 手机运行程序。可以看到，计数器的 iOS 版本也可以正确地支持国际化了。

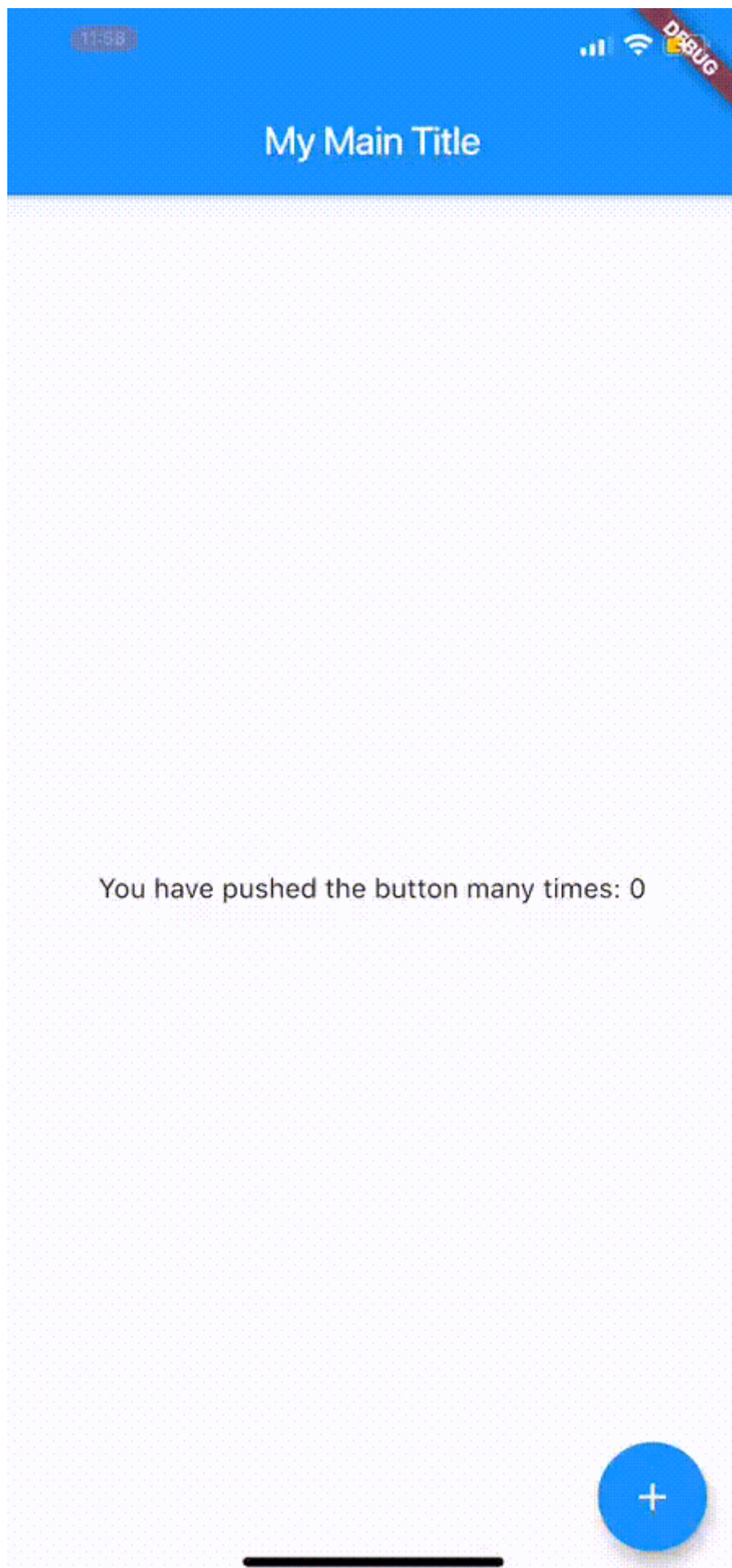


图 7 计数器示例 (iOS 英文系统)



图 8 计数器示例 (iOS 中文系统)


原生工程配置

上面介绍的国际化方案，其实都是在 Flutter 应用内实现的。而在 Flutter 框架运行之前，我们是无法访问这些国际化文案的。

Flutter 需要原生环境才能运行，但有些文案，比如应用的名称，我们需要在 Flutter 框架运行之前就为它提供多个语言版本（比如英文版本为 computer，中文版本为计数器），这时就需要在对应的原生工程中完成相应的国际化配置了。

我们先去 Android 工程下进行应用名称的配置。

首先，在 Android 工程中，应用名称是在 AndroidManifest.xml 文件中 application 的 android:label 属性声明的，所以我们需要将其修改为字符串资源中的一个引用，让其能够根据语言地区自动选择合适的文案：

 复制代码

```
1 <manifest ... >
2     ...
3     <!-- 设置应用名称 -->
4         <application
5             ...
6             android:label="@string/title"
7             ...
8         >
9         </application>
10 </manifest>
```

然后，我们还需要在 android/app/src/main/res 文件夹中，为要支持的语言创建字符串 strings.xml 文件。这里由于默认文件是英文的，所以我们只需要为中文创建一个文件即可。字符串资源的文件目录结构，如下图所示：

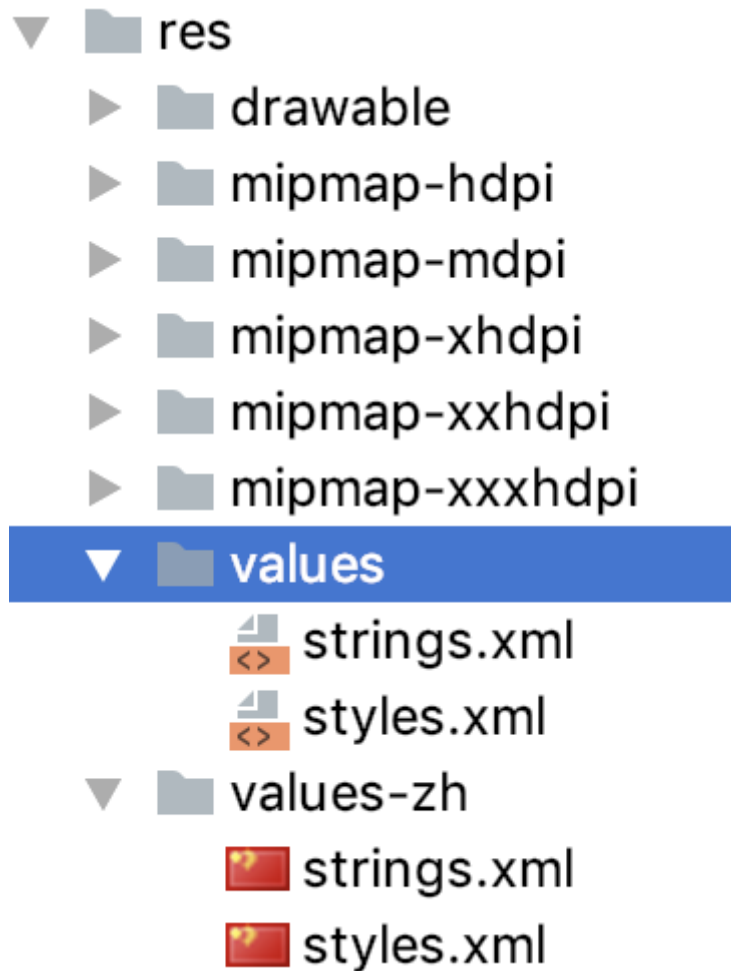



图 9 strings.xml 文件目录结构

values 与 values-zh 文件夹下的 strings.xml 内容如下所示：

 复制代码

```
1 <!-- 英文（默认）字符串资源 -->
2 <?xml version="1.0" encoding="utf-8"?>
3 <resources>
4     <string name="title">Computer</string>
5 </resources>
6
7
8 <!-- 中文字符串资源 -->
9 <?xml version="1.0" encoding="utf-8"?>
10 <resources>
11     <string name="title"> 计数器 </string>
12 </resources>
```


完成 Android 应用标题的工程配置后，我们回到 Flutter 工程，选择 Android 手机运行程序，可以看到，计数器的 Android 应用标题也可以正确地支持国际化了。

接下来，我们再看iOS 工程下如何实现应用名称的配置。

与 Android 工程类似，iOS 工程中的应用名称是在 Info.plist 文件的 Bundle name 属性声明的，所以我們也需要將其修改為字符串資源中的一個引用，使其能夠根據語言地區自動選擇文案：

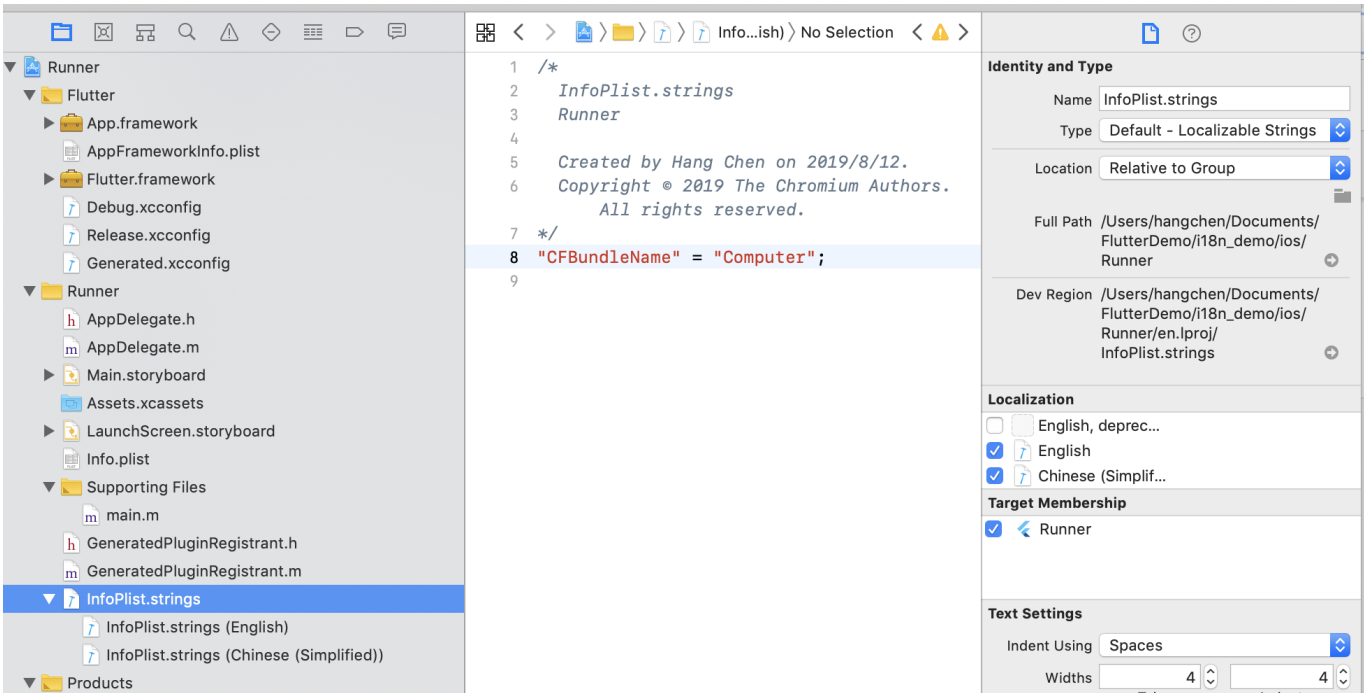


图 10 iOS 工程应用名称配置

由于应用名称默认是不可配置的，所以工程并没有提供英文或者中文的可配置项，这些都需要通过新建与字符串引用对应的资源文件去搞定的。

我们右键单击 Runner 文件夹，然后选择 New File 来添加名为 InfoPlist.strings 的字符串资源文件，并在工程面板的最右侧文件检查器中的 Localization 选项中，添加英文和中文两种语言。InfoPlist.strings 的英文版和中文版内容如下所示：

复制代码

```
1 // 英文版
2 "CFBundleName" = "Computer";
3
4 // 中文版
5 "CFBundleName" = " 计数器 ";
```

至此，我们也完成了 iOS 应用标题的工程配置。我们回到 Flutter 工程，选择 iOS 手机运行程序，发现计数器的 iOS 应用标题也支持国际化了。

总结

好了，今天的分享就到这里。我们来总结下核心知识点吧。

在今天的分享中，我与你介绍了 Flutter 应用国际化的解决方案，即在代码中实现一个 `LocalizationsDelegate`，在这个类中将所有需要翻译的文案全部声明为它的属性，然后依次进行手动翻译适配，最后将这个代理类设置为应用程序的翻译回调。

而为了简化手动翻译到代码转换的过程，我们通常会使用多个 `arb` 文件存储文案在不同语言地区的映射关系，并使用 `Flutter i18n` 插件来实现代码的自动转换。

国际化的核心就是语言差异配置抽取。在原生 Android 和 iOS 系统中进行国际化适配，我们只需为需要国际化的资源（比如，字符串文本、图片、布局等）提供不同的文件夹目录，就可以在应用层代码访问国际化资源时，自动根据语言地区进行适配。

但，Flutter 的国际化能力就相对原始很多，不同语言和地区的国际化资源既没有存放在单独的 `xml` 或者 `JSON` 上，也没有存放在不同的语言和地区文件夹中。幸好有 `Flutter i18n` 插件的帮助，否则为一个应用提供国际化的支持将会是件极其繁琐的事情。

我把今天分享所涉及到的知识点打包到了 [GitHub](#) 中，你可以下载下来，反复运行几次，加深理解与记忆。

思考题

最后，我给你留下一道思考题吧。

在 Flutter 中，如何实现图片类资源的国际化呢？

欢迎你在评论区给我留言分享你的观点，我会在下一篇文章中等待你！感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。

Flutter 核心技术与实战

来自 Google 的高性能跨平台开发框架

陈航

美团点评高级技术专家



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 31 | 如何实现原生推送能力？

下一篇 33 | 如何适配不同分辨率的手机屏幕？

精选留言 (2)

写留言



~ Sweet ~

2019-09-10

教师节快乐

展开 ∨

作者回复: 谢谢



YJ

2019-09-10

教师节到了，老师辛苦了！

展开 ∨

作者回复: 谢谢

