Flutter实现国际化

原创 coderwhy coderwhy

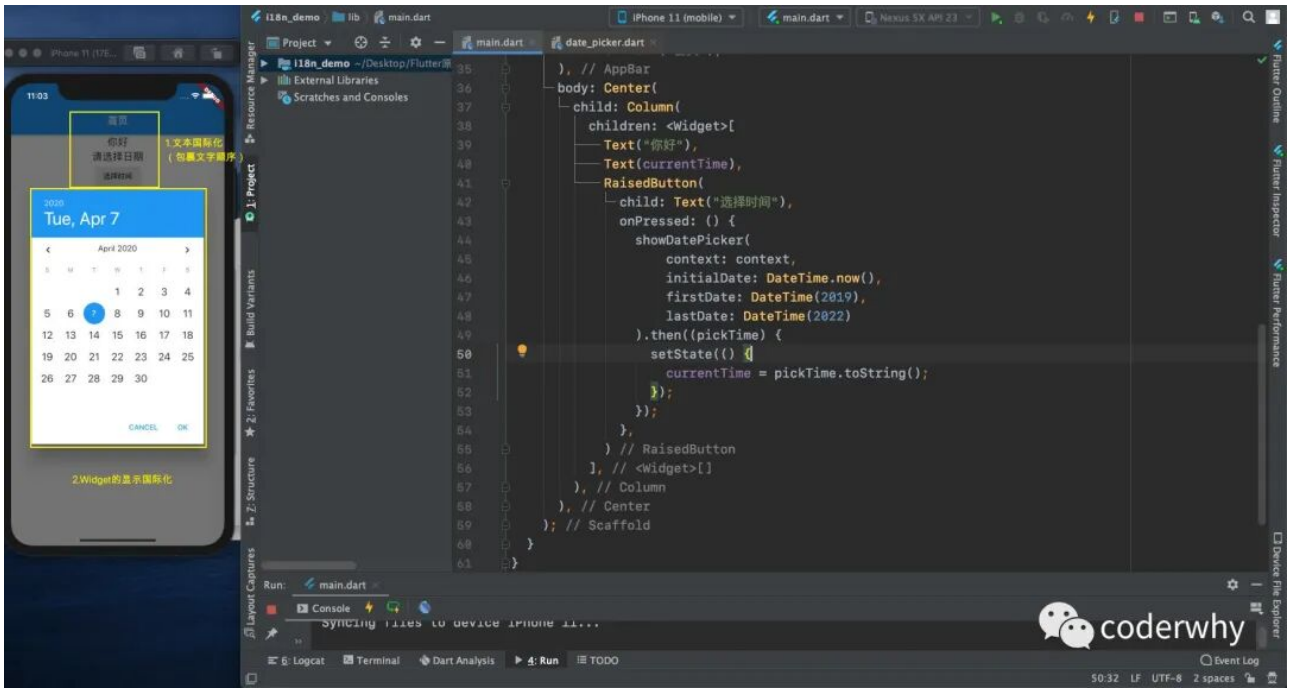
一. 国际化的认识

开发一个App，如果我们的App需要面向不同的语种（比如中文、英文、繁体等），那么我们需要对齐进行国际化开发。

国际化的英文称呼： **internationalization** （简称为 **i18n**，取前后两个字母，18表示中间省略字母的个数）

App国际化开发主要包括：文本国际化（包括文本的顺序），Widget显示的国际化：

- 比如我们下面开发的这个App
- 某些文本在英文环境下应该显示为英文；
- 某些Widget在中文环境下，应该显示中文（比如弹出的时间选择器）；



国际化

二. 国际化的适配

2.1. Widget的国际化

Flutter给我们提供的Widget默认情况下就是支持国际化，但是在没有进行特别的设置之前，它们无论在什么环境都是以 **英文** 的方式显示的。

如果想要添加其他语言，你的应用必须指定额外的 MaterialApp 属性并且添加一个单独的 package，叫做 `flutter_localizations` 。

- 截至到 2020 年 2 月份，这个 package 已经支持大约 77 种语言。

2.1.1. pubspec添加依赖

想要使用 flutter_localizations 的话，我们需要在 `pubspec.yaml` 文件中添加它作为依赖：

```
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
```

2.1.2. 设置MaterialApp

- 在localizationsDelegates中指定哪些Widget需要进行国际化
  - 用于生产 **本地化值** 集合的工厂
  - 我们这里指定了Material、Widgets、Cupertino都使用国际化
- supportedLocales指定要支持哪些国际化
  - 我们这里指定中文和英文（也可以指定国家编码）

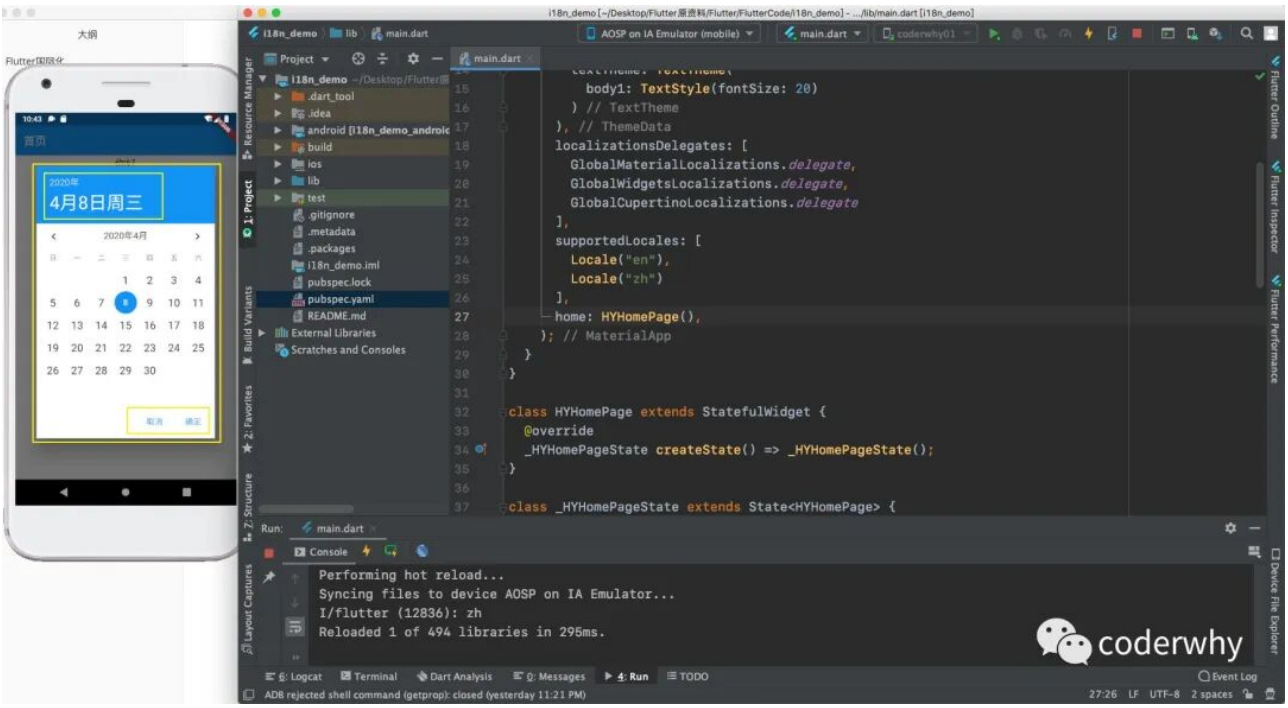
```
MaterialApp(
  localizationsDelegates: [
    GlobalMaterialLocalizations.delegate, // 指定本地化的字符串和一些其他的值
    GlobalCupertinoLocalizations.delegate, // 对应的Cupertino风格
    GlobalWidgetsLocalizations.delegate // 指定默认的文本排列方向, 由左到右或由右到左
  ],
  supportedLocales: [
    Locale("en"),
    Locale("zh")
  ],
)
```

注意：如果要指定语言代码、文字代码和国家代码，可以进行如下指定方式：

```
// Full Chinese support for CN, TW, and HK
supportedLocales: [
  const Locale.fromSubtags(languageCode: 'zh'), // generic Chinese 'zh'
  const Locale.fromSubtags(languageCode: 'zh', scriptCode: 'Hans'), // generic simplified Chinese 'zh_Hans'
  const Locale.fromSubtags(languageCode: 'zh', scriptCode: 'Hant'), // generic traditional Chinese 'zh_Hant'
  const Locale.fromSubtags(languageCode: 'zh', scriptCode: 'Hans', countryCode: 'CN'), // 'zh_Hans_CN'
  const Locale.fromSubtags(languageCode: 'zh', scriptCode: 'Hant', countryCode: 'TW'), // 'zh_Hant_TW'
  const Locale.fromSubtags(languageCode: 'zh', scriptCode: 'Hant', countryCode: 'HK'), // 'zh_Hant_HK'
],
```

2.1.3. 查看Widget结果

设置完成后，我们在Android上将语言切换为中文，查看结果：



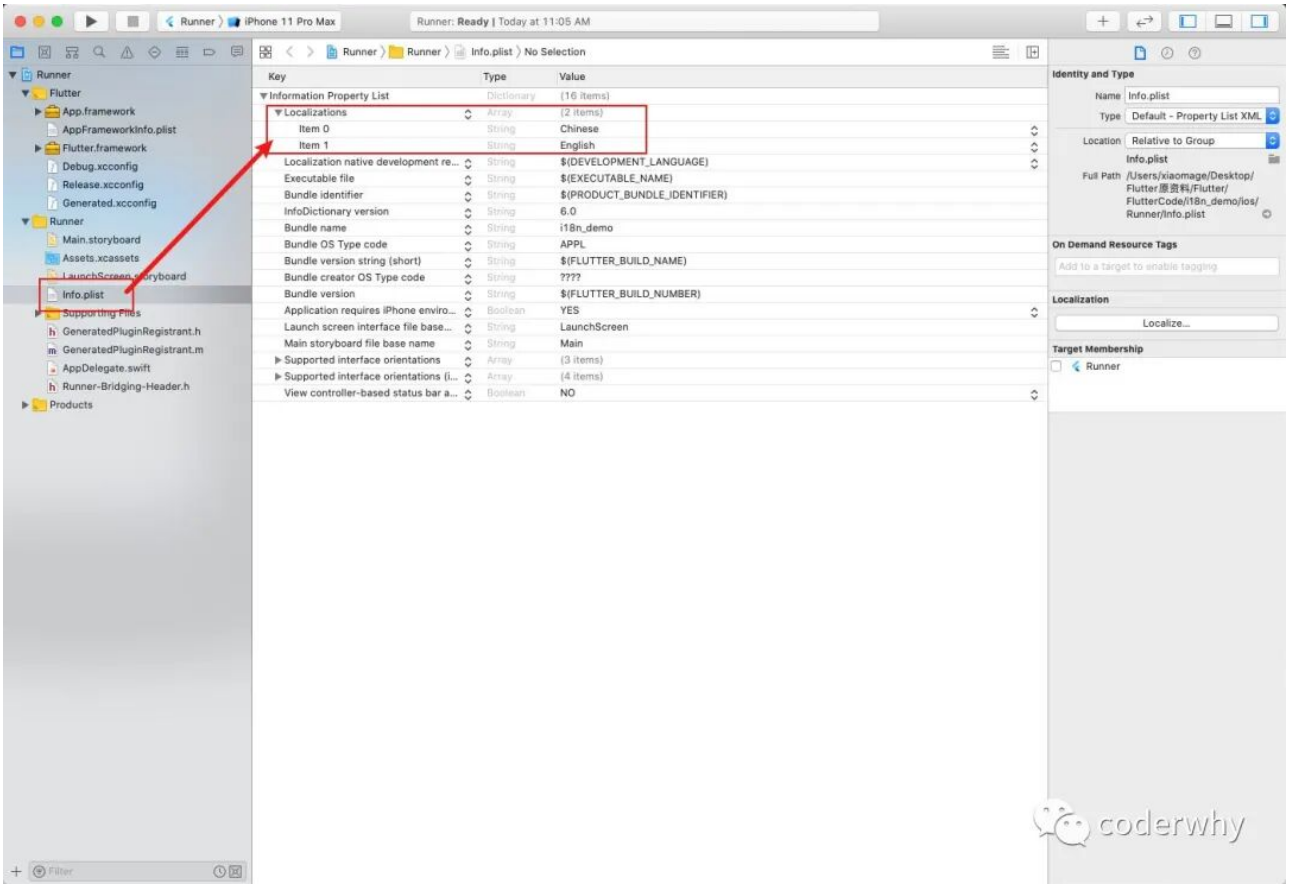
Android运行效果

但是对于iOS，将语言切换为中文，依然显示是英文的Widget

- 这是因为iOS定义了一些应用的元数据，其中包括支持的语言环境；
- 我们必须将其对应的元数据中支持的语言添加进去；
- 元数据的设置在iOS项目中对应的info.plist文件中；

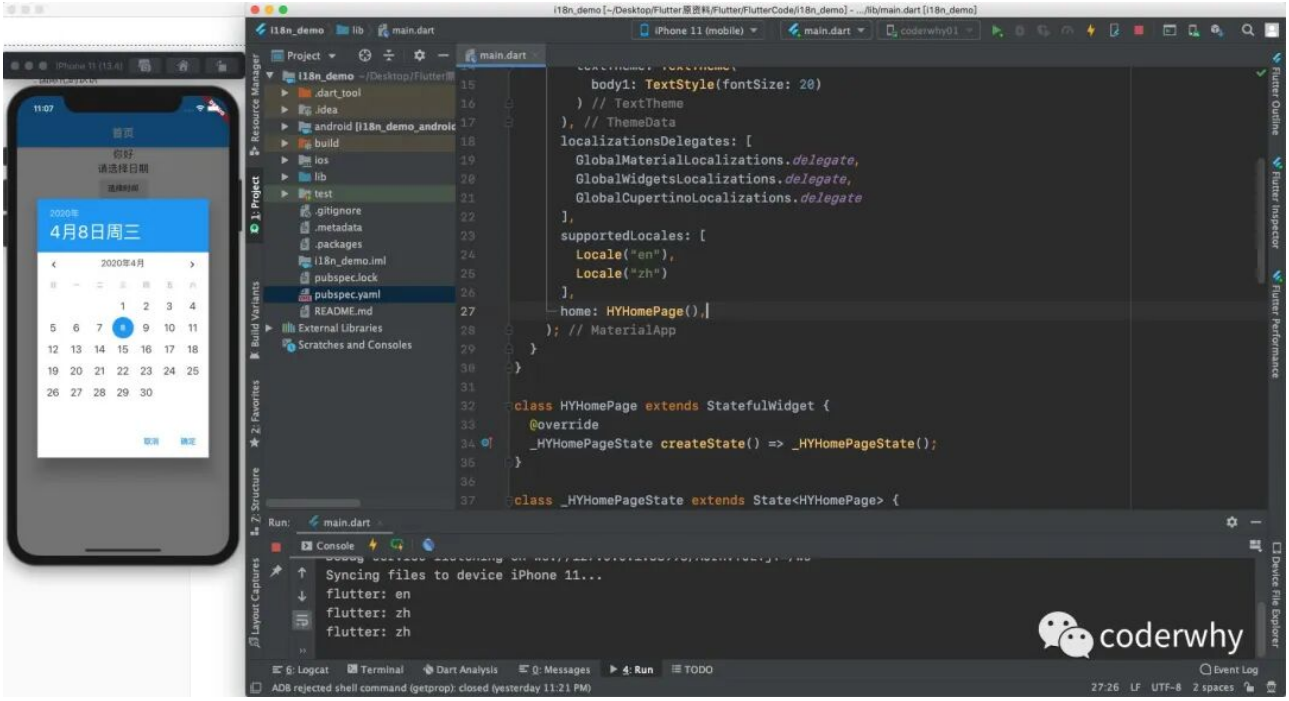
修改iOS的info.plist文件配置：

- 选择 **Information Property List** 项；
- 从 *Editor* 菜单中选择 *Add Item* ， 然后从弹出菜单中选择 **Localizations** ；
- 为array添加一项选择 *Add Item* ， 选择Chinese；



Xcode中配置info.plist文件

配置完成后，卸载之前的app，重新安装：



iOS运行效果

2.2. 其它文本国际化

App中除了有默认的Widget，我们也希望对自己的文本进行国际化，如何做到呢?

2.2.1. 创建本地化类

该类用于定义我们需要进行本地化的字符串等信息：

- 1.我们需要一个构造器，并且传入一个Locale对象（后续会使用到）
- 2.定义一个Map，其中存放我们不同语言对应的显示文本
- 3.定义它们对应的getter方法，根据语言环境返回不同的结果

```
import 'package:flutter/material.dart';

class HYLocalizations {
  final Locale locale;

  HYLocalizations(this.locale);

  static Map<String, Map<String, String>> _localizedValues = {
    "en": {
      "title": "home",
      "greet": "hello~",
      "picktime": "Pick a Time"
    },
    "zh": {
      "title": "首页",
      "greet": "你好~",
      "picktime": "选择一个时间"
    }
  };

  String get title {
    return _localizedValues[locale.languageCode]["title"];
  }

  String get greet {
    return _localizedValues[locale.languageCode]["greet"];
  }

  String get pickTime {
    return _localizedValues[locale.languageCode]["picktime"];
  }
}
```

2.2.2. 自定义Delegate

上面的类定义好后，我们在什么位置或者说如何对它进行初始化呢?

- 答案是我们可以像Flutter Widget中的国际化方式一样对它们进行初始化；
- 也就是我们也定义一个对象的Delegate类，并且将其传入localizationsDelegates中；
- Delegate的作用就是当 `Locale发生改变` 时，调用对应的 `load方法` ，重新加载新的 `Locale资源` ；

HYLocalizationsDelegate需要继承自LocalizationsDelegate，并且有三个方法必须重写：

- isSupported：用于当前环境的Locale，是否在我们支持的语言范围
- shouldReload：当Localizations Widget重新build时，是否调用load方法重新加载Locale资源
  - 一般情况下，Locale资源只应该在Locale切换时加载一次，不需要每次Localizations重新build时都加载一遍；
  - 所以一般情况下返回false即可；
- load方法：当Locale发生改变时（语言环境），加载对应的HYLocalizations资源
  - 这个方法返回的是一个Future，因为有可能是异步加载的；
  - 但是我们这里是直接定义的一个Map，因此可以直接返回一个同步的Future（SynchronousFuture）

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/foundation.dart';
import 'package:i18n_demo/i18n/localizations.dart';

class HYLocalizationsDelegate extends LocalizationsDelegate<HYLocalizations> {
  @override
  bool isSupported(Locale locale) {
    return ["en", "zh"].contains(locale.languageCode);
  }

  @override
  bool shouldReload(LocalizationsDelegate<HYLocalizations> old) {
    return false;
  }

  @override
  Future<HYLocalizations> load(Locale locale) {
    return SynchronousFuture(HYLocalizations(locale));
  }

  static HYLocalizationsDelegate delegate = HYLocalizationsDelegate();
}
```

2.2.3. 使用本地化类

接着我们可以在代码中使用HYLocalization类。

- 我们可以通过Localizations.of(context, HYLocalizations)获取到HYLocalizations对象

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(Localizations.of(context, HYLocalizations).title),
      ),
      body: Center(
        child: Column(
          children: <Widget>[
            Text(Localizations.of(context, HYLocalizations).greet),
            RaisedButton(
              child: Text(Localizations.of(context, HYLocalizations).pickTime),
              onPressed: () {
                showDatePicker(
                    context: context,
                    initialDate: DateTime.now(),
                    firstDate: DateTime(2019),
                    lastDate: DateTime(2022)
                ).then((pickTime) {
                });
              },
            )
          ],
        ),
      ),
    );
  }
```

当然，我们可以对Localizations.of(context, HYLocalizations)进行一个优化

- 给HYLocalizations定义一个of的静态方法

```
class HYLocalizations {
  static HYLocalizations of(BuildContext context) {
    return Localizations.of(context, HYLocalizations);
  }
}
```

接下来我们就可以通过下面的方式来使用了（其它地方也是一样）：

```
appBar: AppBar(
  title: Text(HYLocalizations.of(context).title),
)
```

2.2.4. 异步加载数据

假如我们的数据是异步加载的，比如来自Json文件或者服务器，应该如何处理呢?

这里我们可以修改HYLocalizations的数据加载：

```
  static Map<String, Map<String, String>> _localizedValues = {};

  Future<bool> loadJson() async {
    // 1.加载json文件
    String jsonString = await rootBundle.loadString("assets/json/i18n.json");

    // 2.转成map类型
    final Map<String, dynamic> map = json.decode(jsonString);

    // 3.注意：这里是将Map<String, dynamic>转成Map<String, Map<String, String>>类型
    _localizedValues = map.map((key, value) {
      return MapEntry(key, value.cast<String, String>());
    });
    return true;
  }
```

在HYLocalizationsDelegate中使用异步进行加载：

```
  @override
  Future<HYLocalizations> load(Locale locale) async {
    final localization = HYLocalizations(locale);
    await localization.loadJson();
    return localization;
  }
```

三. 国际化的工具

3.1. 认识arb文件

目前我们已经可以通过加载对应的json文件来进行本地化了。

但是还有另外一个问题，我们在进行国际化的过程中，下面的代码依然需要根据json文件 手动编写 ：

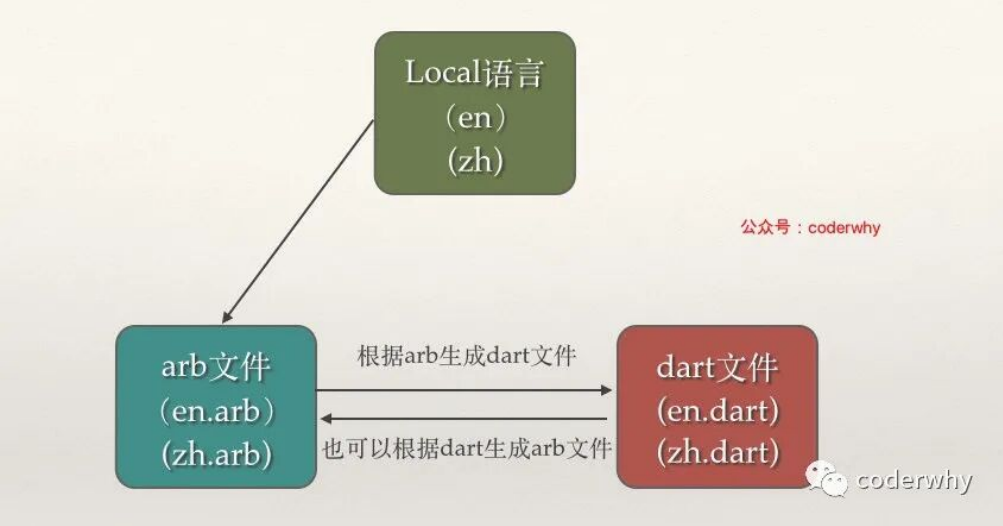
```
  String get title {
    return _localizedValues[locale.languageCode]["title"];
  }

  String get greet {
    return _localizedValues[locale.languageCode]["greet"];
  }

  String get pickTime {
    return _localizedValues[locale.languageCode]["picktime"];
  }
```

有没有一种更好的方式，让我们可以快速在本地化文件-dart代码文件直接来转换呢？答案就是arb文件

- arb文件全称 **Application Resource Bundle** ，表示应用资源包，目前已经得到Google的支持；
- 其本质就是一个json文件，但是可以根据该文件转成对应的语言环境；
- arb的说明文档：https://github.com/google/app-resource-bundle/wiki/ApplicationResourceBundleSpecification



dart和arb转换

3.2. intl package

官方文档推荐可以使用intl package来进行arb和dart文件之间的转换（通过终端指令）

- https://flutter.dev/docs/development/accessibility-and-localization/internationalization#appendix-using-the-dart-intl-tools

需要在在pubspec.yaml中添加其相关的依赖，具体步骤这里不再详细给出，可以参考官方文档

3.3. 使用IDE插件

在之前有一个比较好用的Android Studio的插件：Flutter i18n

- 但是这个插件已经很久不再维护了，所以不再推荐给大家使用

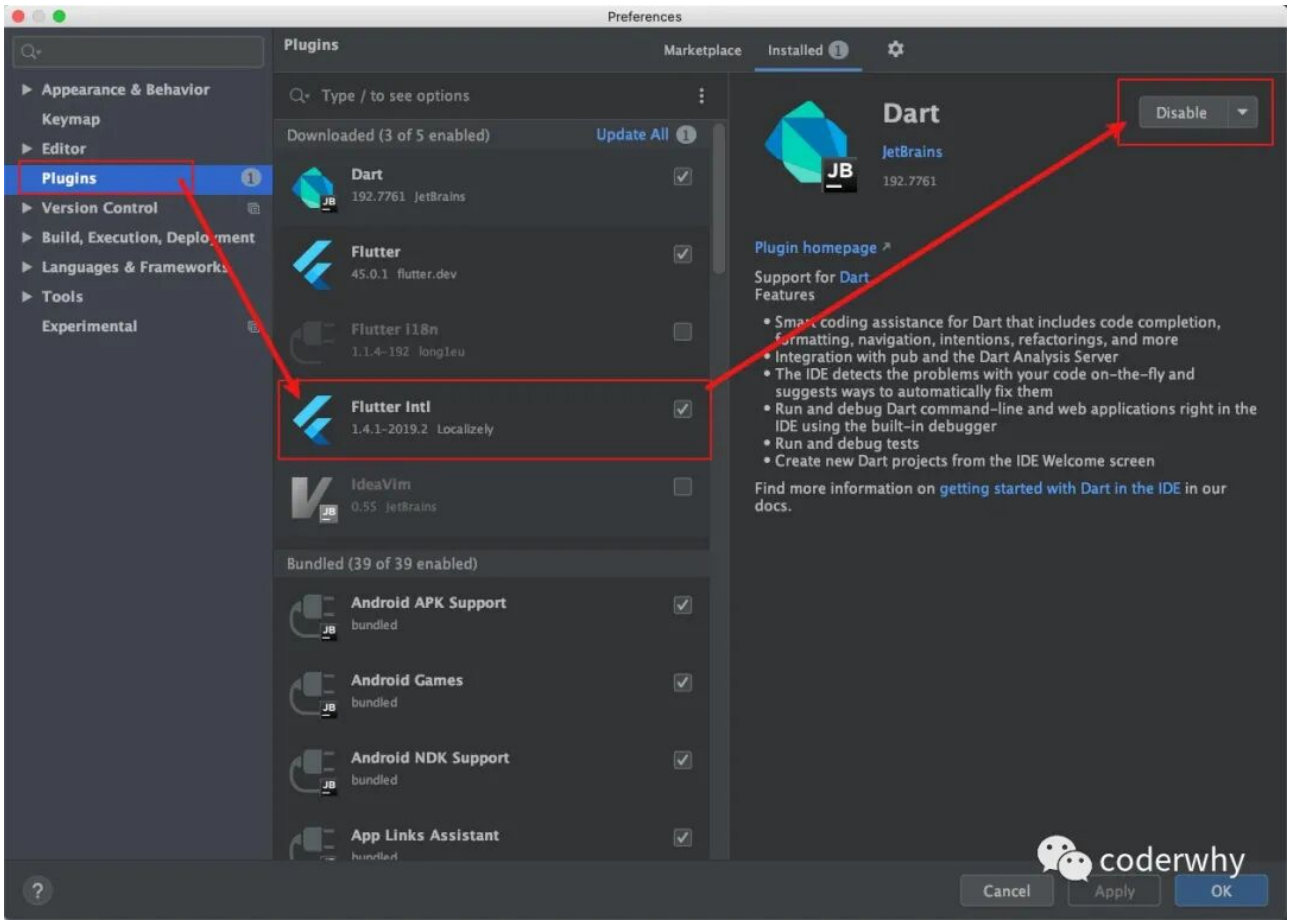
目前我们可以使用另外一个插件：Flutter Intl

- 该插件更新维护频率很高，并且广受好评；
- 另外，在Android Studio和VSCode中都是支持的

我们这里以Android Studio为例，讲解其使用过程：

3.3.1. 安装插件

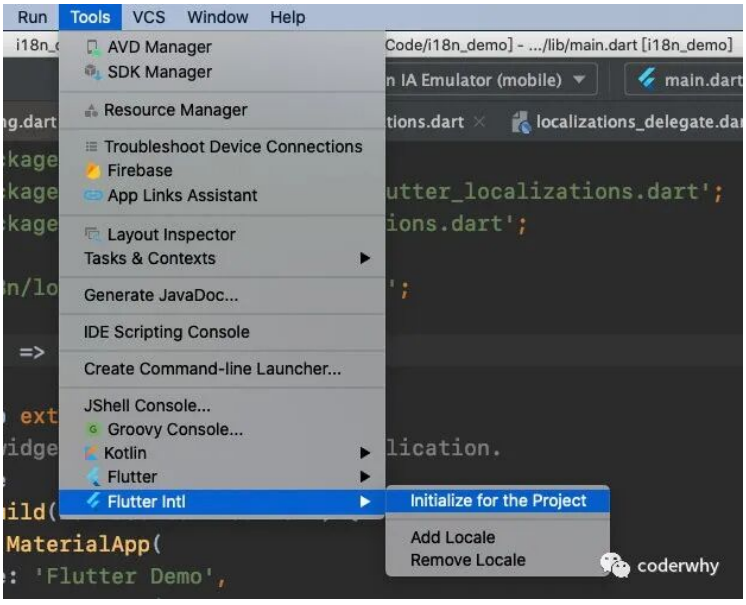
在Android Studio的Plugins中安装插件：



插件安装

3.3.2. 初始化intl

选择工具栏Tools - Flutter Intl - **Initialize for the Project**



初始化intl

完成上面的操作之后会自动生成如下文件目录：

- generated是自动生成的dart代码
- l10n是对应的arb文件目录

目录结构

3.3.3. 使用intl

在localizationsDelegates中配置生成的class，名字是S

- 1.添加对应的delegate
- 2.supportedLocales使用S.delegate.supportedLocales

```
localizationsDelegates: [
  GlobalMaterialLocalizations.delegate,
  GlobalWidgetsLocalizations.delegate,
  GlobalCupertinoLocalizations.delegate,
  HYLocalizationsDelegate.delegate,
  S.delegate
],
supportedLocales: S.delegate.supportedLocales,
```

因为我们目前还没有对应的本地化字符串，所以需要在intl_en.arb文件中编写：

- 编写后ctrl（command） + s保存即可

```
{
  "title": "home",
  "greet": "hello~",
  "picktime": "Pick a time"
}
```

在代码中使用即可

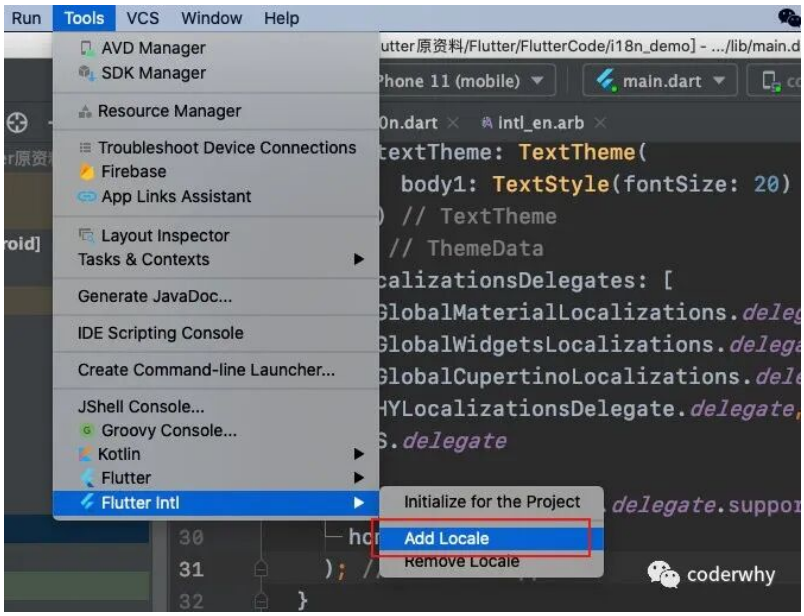
- 按照如下格式：S.of(context).title

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(S.of(context).title),
      ),
      body: Center(
        child: Column(
          children: <Widget>[
            Text(S.of(context).greet),
            RaisedButton(
              child: Text(S.of(context).picktime),
              onPressed: () {
                showDatePicker(
                    context: context,
                    initialDate: DateTime.now(),
                    firstDate: DateTime(2019),
                    lastDate: DateTime(2022)
                ).then((pickTime) {
                });
              },
            )
          ],
        ),
      ),
    );
  }
```

3.3.4. 添加中文

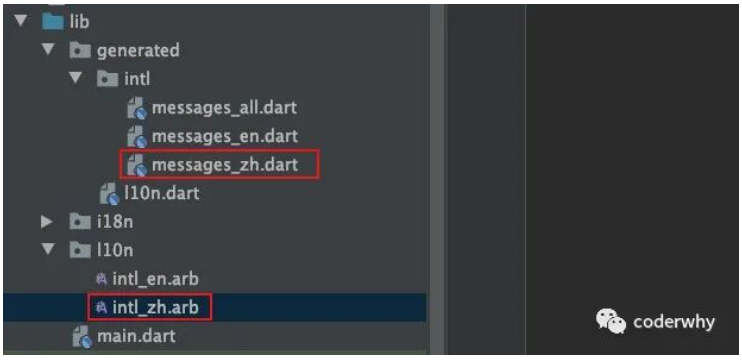
如果希望添加中文支持：add local

- 在弹出框中输入zh即可



添加中文

我们会发现，会生成对应的intl_zh.arb和messages_zh.dart文件

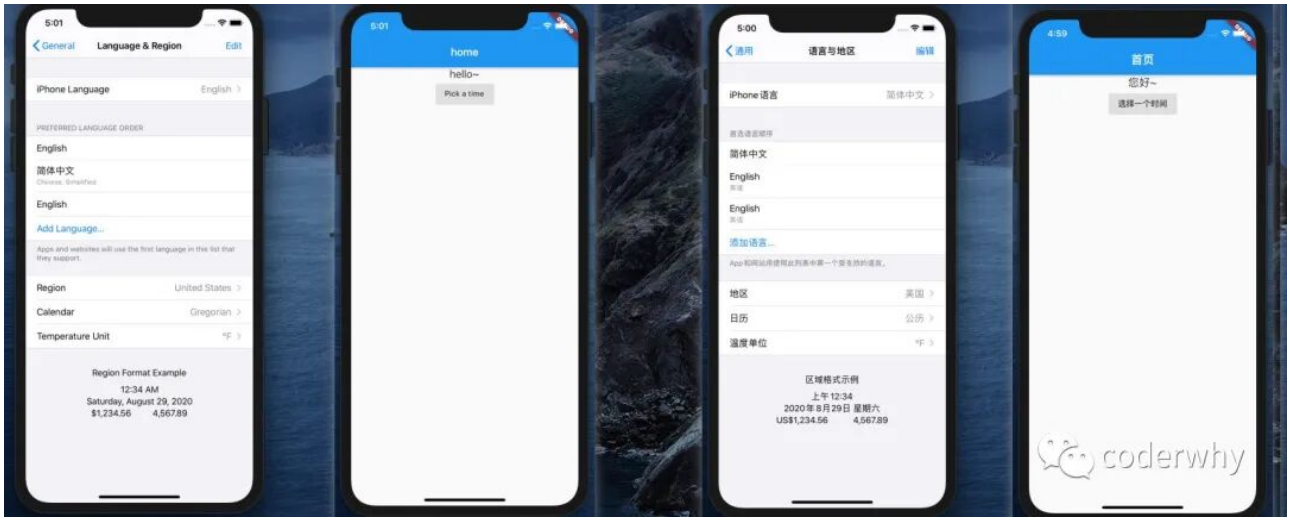


目录结构

编写intl_zh.arb文件：

```
{
  "title": "首页",
  "greet": "您好~",
  "picktime": "选择一个时间"
}
```

查看界面，会根据当前语言显示对应的语言文本



显示结果

3.4. arb其它语法

如果我们希望在使用本地化的过程中传递一些参数：

- 比如hello kobe或hello james
- 比如你好啊，李银河 或 你好啊，王小波

修改对应的arb文件：

- {name}：表示传递的参数

```
{
  "title": "home",
  "greet": "hello~",
  "picktime": "Pick a time",
  "sayHello": "hello {name}"
}
```

在使用时，传入对应的参数即可：

```
Text(S.of(context).sayHello("李银河")),
```

arb还有更多的语法，大家可以在之后慢慢学习和发掘~

> 备注：所有内容首发于公众号，之后除了Flutter也会更新其他技术文章，TypeScript、React、Node、uniapp、mpvue、数据结构与算法等等，也会更新一些自己的学习心得等，欢迎大家关注

coderwhy

微信扫描二维码，关注我的公众号

coderwhy

公众号