# 使用Intl包

使用[Intl](https://pub.dartlang.org/packages/intl)包我们不仅可以非常轻松的实现国际化，而且也可以将字符串文本分离成单独的文件，方便开发人员和翻译人员分工协作。为了使用[Intl](https://pub.dartlang.org/packages/intl)包我们需要添加两个依赖：

dependencies:  
 #...省略无关项  
 intl: ^0.15.7   
dev\_dependencies:  
 #...省略无关项  
 intl\_translation: ^0.17.2

[intl\_translation](https://pub.dartlang.org/packages/intl_translation) 包主要包含了一些工具，它在开发阶段主要主要的作用是从代码中提取要国际化的字符串到单独的arb文件和根据arb文件生成对应语言的dart代码，而intl包主要是引用和加载intl\_translation生成后的dart代码。下面我们将一步步来说明如何使用：

### 第一步：创建必要目录

首先，在项目根目录下创建一个l10n-arb目录，该目录保存我们接下来通过intl\_translation命令生成的arb文件。一个简单的arb文件内容如下：

{  
 "@@last\_modified": "2018-12-10T15:46:20.897228",  
 "@@locale":"zh\_CH",  
 "title": "Flutter应用",  
 "@title": {  
 "description": "Title for the Demo application",  
 "type": "text",  
 "placeholders": {}  
 }  
}

我们根据“@@locale”字段可以看出这个arb对应的是中文简体的翻译，里面的title字段对应的正是我们应用标题的中文简体翻译。@title字段是对title的一些描述信息。

接下来，我们在lib目录下创建一个l10n的目录，该目录用于保存从arb文件生成的dart代码文件。

### 第二步：实现Localizations和Delegate类

和上一节中的步骤类似，我们仍然要实现Localizations和Delegate类，不同的是，现在我们在实现时要使用intl包的一些方法（有些是动态生成的）。

下面我们在lib/l10n目录下新建一个“localization\_intl.dart”的文件，文件内容如下：

import 'package:flutter/material.dart';  
import 'package:intl/intl.dart';  
import 'messages\_all.dart'; //1  
  
class DemoLocalizations {  
 static Future<DemoLocalizations> load(Locale locale) {  
 final String name = locale.countryCode.isEmpty ? locale.languageCode : locale.toString();  
 final String localeName = Intl.canonicalizedLocale(name);  
 //2  
 return initializeMessages(localeName).then((b) {  
 Intl.defaultLocale = localeName;  
 return new DemoLocalizations();  
 });  
 }  
  
 static DemoLocalizations of(BuildContext context) {  
 return Localizations.of<DemoLocalizations>(context, DemoLocalizations);  
 }  
  
 String get title {  
 return Intl.message(  
 'Flutter APP',  
 name: 'title',  
 desc: 'Title for the Demo application',  
 );  
 }  
}  
  
//Locale代理类  
class DemoLocalizationsDelegate extends LocalizationsDelegate<DemoLocalizations> {  
 const DemoLocalizationsDelegate();  
  
 //是否支持某个Local  
 @override  
 bool isSupported(Locale locale) => ['en', 'zh'].contains(locale.languageCode);  
  
 // Flutter会调用此类加载相应的Locale资源类  
 @override  
 Future<DemoLocalizations> load(Locale locale) {  
 //3  
 return DemoLocalizations.load(locale);  
 }  
  
 // 当Localizations Widget重新build时，是否调用load重新加载Locale资源.  
 @override  
 bool shouldReload(DemoLocalizationsDelegate old) => false;  
}

注意：

* 注释1的“messages\_all.dart”文件是通过[intl\_translation](https://pub.dartlang.org/packages/intl_translation)工具从arb文件生成的代码，所以在第一次运行生成命令之前，此文件不存在。注释2处的initializeMessages()方法和“messages\_all.dart”文件一样，是同时生成的。
* 注释3处和上一节示例代码不同，这里我们直接调用DemoLocalizations.load()即可。

### 第三步：添加需要国际化的属性

现在我们可以在DemoLocalizations类中添加需要国际化的属性或方法，如上面示例代码中的title属性，这时我们就要用到Intl库提供的一些方法，这些方法可以帮我们轻松实现不同语言的一些语法特性，如复数语境，举个例子，比如我们有一个电子邮件列表页，我们需要在顶部显示未读邮件的数量，在未读数量不同事，我们展示的文本可能会不同：

|  |  |
| --- | --- |
| 未读邮件数 | 提示语 |
| 0 | There are no emails left |
| 1 | There is 1 email left |
| n(n>1) | There are n emails left |

我们可以通过Intl.plural(...)来实现：

remainingEmailsMessage(int howMany) => Intl.plural(howMany,  
 zero: 'There are no emails left',  
 one: 'There is $howMany email left',  
 other: 'There are $howMany emails left',  
 name: "remainingEmailsMessage",  
 args: [howMany],  
 desc: "How many emails remain after archiving.",  
 examples: const {'howMany': 42, 'userName': 'Fred'});

可以看到通过Intl.plural方法可以在howMany值不同时输出不同的提示信息。

[Intl](https://pub.dartlang.org/packages/intl)包还有一些其他的方法，读者可以自行查看其文档，本书不在赘述。

### 第四步：生成arb文件

现在我们可以通[intl\_translation](https://pub.dartlang.org/packages/intl_translation)包的工具来提取代码中的字符串到一个arb文件，运行如下命名：

flutter pub pub run intl\_translation:extract\_to\_arb --output-dir=l10n-arb \ lib/l10n/localization\_intl.dart

运行此命令后，会将我们之前通过Intl API标识的属性和字符串提取到“l10n-arb/intl\_messages.arb”文件中，我们看看其内容：

{  
 "@@last\_modified": "2018-12-10T17:37:28.505088",  
 "title": "Flutter APP",  
 "@title": {  
 "description": "Title for the Demo application",  
 "type": "text",  
 "placeholders": {}  
 },  
 "remainingEmailsMessage": "{howMany,plural, =0{There are no emails left}=1{There is {howMany} email left}other{There are {howMany} emails left}}",  
 "@remainingEmailsMessage": {  
 "description": "How many emails remain after archiving.",  
 "type": "text",  
 "placeholders": {  
 "howMany": {  
 "example": 42  
 }  
 }  
 }  
}

这个是默认的Locale资源文件，如果我们现在要支持中文简体，只需要在该文件同级目录创建一个“intl\_zh\_CN.arb”文件，然后将“intl\_messages.arb”的内容拷贝到“intl\_zh\_CN.arb”文件，接下来将英文翻译为中文即可，翻译后的“intl\_zh\_CN.arb”文件内容如下：

{  
 "@@last\_modified": "2018-12-10T15:46:20.897228",  
 "@@locale":"zh\_CN",  
 "title": "Flutter应用",  
 "@title": {  
 "description": "Title for the Demo application",  
 "type": "text",  
 "placeholders": {}  
 },  
 "remainingEmailsMessage": "{howMany,plural, =0{没有未读邮件}=1{有{howMany}封未读邮件}other{有{howMany}封未读邮件}}",  
 "@remainingEmailsMessage": {  
 "description": "How many emails remain after archiving.",  
 "type": "text",  
 "placeholders": {  
 "howMany": {  
 "example": 42  
 }  
 }  
 }  
}

我们必须要翻译title和remainingEmailsMessage字段，description是该字段的说明，通常给翻译人员看，代码中不会用到。

有两点需要说明：

1. 如果某个特定的arb中缺失某个属性，那么应用将会加载默认的arb文件(intl\_messages.arb)中的相应属性，这是Intl的托底策略。
2. 每次运行提取命令时，intl\_messages.arb都会根据代码重新生成，但其他arb文件不会，所以当要添加新的字段或方法时，其他arb文件是增量的，不用担心会覆盖。
3. arb文件是标准的，其格式规范可以自行了解。通常会将arb文件交给翻译人员，当他们完成翻译后，我们再通过下面的步骤根据arb文件生成最终的dart代码。

### 第五步：生成dart代码

最后一步就是根据arb生成dart文件：

flutter pub pub run intl\_translation:generate\_from\_arb --output-dir=lib/l10n --no-use-deferred-loading lib/l10n/localization\_intl.dart l10n-arb/intl\_\*.arb

这句命令在首次运行时会在“lib/l10n”目录下生成多个文件，对应多种Locale，这些代码便是最终要使用的dart代码。

### 总结

至此，我们将使用[Intl](https://pub.dartlang.org/packages/intl)包对APP进行国际化的流程介绍完了，我们可以发现，其中第一步和第二步只在第一次需要，而我们开发时的主要的工作都是在第三步。由于最后两步在第三步完成后每次也都需要，所以我们可以将最后两步放在一个shell脚本里，当我们完成第三步或完成arb文件翻译后只需要分别执行该脚本即可。我们在根目录下创建一个intl.sh的脚本，内容为：

flutter pub pub run intl\_translation:extract\_to\_arb --output-dir=l10n-arb lib/l10n/localization\_intl.dart  
flutter pub pub run intl\_translation:generate\_from\_arb --output-dir=lib/l10n --no-use-deferred-loading lib/l10n/localization\_intl.dart l10n-arb/intl\_\*.arb

然后授予执行权限：

chmod +x intl.sh

执行intl.sh

./intl.sh