# 13.2 实现Localizations

前面讲了Material组件库如何支持国际化，本节我们将介绍一下我们自己的UI中如何支持多语言。根据上节所述，我们需要实现两个类：一个Delegate类一个Localizations类，下面我们通过一个实例说明。

### 实现Localizations类

我们已经知道Localizations类中主要实现提供了本地化值，如文本：

//Locale资源类  
class DemoLocalizations {  
 DemoLocalizations(this.isZh);  
 //是否为中文  
 bool isZh = false;  
 //为了使用方便，我们定义一个静态方法  
 static DemoLocalizations of(BuildContext context) {  
 return Localizations.of<DemoLocalizations>(context, DemoLocalizations);  
 }  
 //Locale相关值，title为应用标题  
 String get title {  
 return isZh ? "Flutter应用" : "Flutter APP";  
 }  
 //... 其它的值   
}

DemoLocalizations中会根据当前的语言来返回不同的文本，如title，我们可以将所有需要支持多语言的文本都在此类中定义。DemoLocalizations的实例将会在Delegate类的load方法中创建。

### 实现Delegate类

Delegate类的职责是在Locale改变时加载新的Locale资源，所以它有一个load方法，Delegate类需要继承自LocalizationsDelegate类，实现相应的接口，示例如下：

//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) {  
 print("xxxx$locale");  
 return SynchronousFuture<DemoLocalizations>(  
 DemoLocalizations(locale.languageCode == "zh")  
 );  
 }  
  
 @override  
 bool shouldReload(DemoLocalizationsDelegate old) => false;  
}

shouldReload的返回值决定当Localizations组件重新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，不需要每次在Localizations重新build时都加载，所以返回false即可。可能有些人会担心返回false的话在APP启动后用户再改变系统语言时load方法将不会被调用，所以Locale资源将不会被加载。事实上，每当Locale改变时Flutter都会再调用load方法加载新的Locale，无论shouldReload返回true还是false。

### 最后一步：添加多语言支持

和上一节中介绍的相同，我们现在需要先注册DemoLocalizationsDelegate类，然后再通过DemoLocalizations.of(context)来动态获取当前Locale文本。

只需要在MaterialApp或WidgetsApp的localizationsDelegates列表中添加我们的Delegate实例即可完成注册：

localizationsDelegates: [  
 // 本地化的代理类  
 GlobalMaterialLocalizations.delegate,  
 GlobalWidgetsLocalizations.delegate,  
 // 注册我们的Delegate  
 DemoLocalizationsDelegate()  
],

接下来我们可以在Widget中使用Locale值：

return Scaffold(  
 appBar: AppBar(  
 //使用Locale title   
 title: Text(DemoLocalizations.of(context).title),  
 ),  
 ... //省略无关代码  
 ）

这样，当在美国英语和中文简体之间切换系统语言时，APP的标题将会分别为“Flutter APP”和“Flutter应用”。

### 总结

本节我们通过一个简单的示例说明了Flutter应用国际化的基本过程及原理。但是上面的实例还有一个严重的不足就是我们需要在DemoLocalizations类中获取title时手动的判断当前语言Locale，然后返回合适的文本。试想一下，当我们要支持的语言不是两种而是8种甚至20几种时，如果为每个文本属性都要分别去判断到底是哪种Locale从而获取相应语言的文本将会是一件非常复杂的事。还有，通常情况下翻译人员并不是开发人员，能不能像i18n或i10n标准那样可以将翻译单独保存为一个arb文件交由翻译人员去翻译，翻译好之后开发人员再通过工具将arb文件转为代码。答案是肯定的！我们将在下一节介绍如何通过Dart intl包来实现这些。