# 15.4 全局变量及共享状态

应用程序中通常会包含一些贯穿APP生命周期的变量信息，这些信息在APP大多数地方可能都会被用到，比如当前用户信息、Local信息等。在Flutter中我们把需要全局共享的信息氛围两类：全局变量和共享状态。全局变量就是单纯指会贯穿整个APP生命周期的变量，用于单纯的保存一些信息，或者封装一些全局工具和方法的对象。而共享状态则是指哪些需要跨组件或跨路由共享的信息，这些信息通常也是全局变量，而共享状态和全局变量的不同在于前者者发生改变时需要通知所有使用该状态的组件，而后者不需要。为此，我们将全局变量和共享状态分开单独管理。

## 15.4.1 全局变量-Global类

我们在“lib/common”目录下创建一个Global类，它主要管理APP的全局变量，定义如下：

// 提供四套可选主题色  
const \_themes = <MaterialColor>[  
 Colors.blue,  
 Colors.cyan,  
 Colors.teal,  
 Colors.green,  
 Colors.red,  
];  
  
class Global {  
 static SharedPreferences \_prefs;  
 static Profile profile = Profile();  
 // 网络缓存对象  
 static NetCache netCache = NetCache();  
  
 // 可选的主题列表  
 static List<MaterialColor> get themes => \_themes;  
  
 // 是否为release版  
 static bool get isRelease => bool.fromEnvironment("dart.vm.product");  
  
 //初始化全局信息，会在APP启动时执行  
 static Future init() async {  
 \_prefs = await SharedPreferences.getInstance();  
 var \_profile = \_prefs.getString("profile");  
 if (\_profile != null) {  
 try {  
 profile = Profile.fromJson(jsonDecode(\_profile));  
 } catch (e) {  
 print(e);  
 }  
 }  
  
 // 如果没有缓存策略，设置默认缓存策略  
 profile.cache = profile.cache ?? CacheConfig()  
 ..enable = true  
 ..maxAge = 3600  
 ..maxCount = 100;  
  
 //初始化网络请求相关配置  
 Git.init();  
 }  
  
 // 持久化Profile信息  
 static saveProfile() =>  
 \_prefs.setString("profile", jsonEncode(profile.toJson()));  
}

Global类的各个字段的意义都有注释，在此不再赘述，需要注意的是init()需要在App启动时就要执行，所以应用的main方法如下：

void main() => Global.init().then((e) => runApp(MyApp()));

在此，一定要确保Global.init()方法不能抛出异常，否则 runApp(MyApp())根本执行不到。

## 15.4.2 共享状态

有了全局变量，我们还需要考虑如何跨组件共享状态。当然，如果我们将要共享的状态全部用全局变量替代也是可以的，但是这在Flutter开发中并不是一个好主意，因为组件的状态是和UI相关，而在状态改变时我们会期望依赖该状态的UI组件会自动更新，如果使用全局变量，那么我们必须得去手动处理状态变动通知、接收机制以及变量和组件依赖关系。因此，本实例中，我们使用前面介绍过的Provider包来实现跨组件状态共享，因此我们需要定义相关的Provider。在本实例中，需要共享的状态有登录用户信息、APP主题信息、APP语言信息。由于这些信息改变后都要立即通知其它依赖的该信息的Widget更新，所以我们应该使用ChangeNotifierProvider，另外，这些信息改变后都是需要更新Profile信息并进行持久化的。综上所述，我们可以定义一个ProfileChangeNotifier基类，然后让需要共享的Model继承自该类即可，ProfileChangeNotifier定义如下：

class ProfileChangeNotifier extends ChangeNotifier {  
 Profile get \_profile => Global.profile;  
  
 @override  
 void notifyListeners() {  
 Global.saveProfile(); //保存Profile变更  
 super.notifyListeners(); //通知依赖的Widget更新  
 }  
}

### 用户状态

用户状态在登录状态发生变化时更新、通知其依赖项，我们定义如下：

class UserModel extends ProfileChangeNotifier {  
 User get user => \_profile.user;  
  
 // APP是否登录(如果有用户信息，则证明登录过)  
 bool get isLogin => user != null;  
  
 //用户信息发生变化，更新用户信息并通知依赖它的子孙Widgets更新  
 set user(User user) {  
 if (user?.login != \_profile.user?.login) {  
 \_profile.lastLogin = \_profile.user?.login;  
 \_profile.user = user;  
 notifyListeners();  
 }  
 }  
}

### APP主题状态

主题状态在用户更换APP主题时更新、通知其依赖项，定义如下：

class ThemeModel extends ProfileChangeNotifier {  
 // 获取当前主题，如果为设置主题，则默认使用蓝色主题  
 ColorSwatch get theme => Global.themes  
 .firstWhere((e) => e.value == \_profile.theme, orElse: () => Colors.blue);  
  
 // 主题改变后，通知其依赖项，新主题会立即生效  
 set theme(ColorSwatch color) {  
 if (color != theme) {  
 \_profile.theme = color[500].value;  
 notifyListeners();  
 }  
 }  
}

### APP语言状态

当APP语言选为跟随系统（Auto）时，在系通语言改变时，APP语言会更新；当用户在APP中选定了具体语言时（美国英语或中文简体），则APP便会一直使用用户选定的语言，不会再随系统语言而变。语言状态类定义如下：

class LocaleModel extends ProfileChangeNotifier {  
 // 获取当前用户的APP语言配置Locale类，如果为null，则语言跟随系统语言  
 Locale getLocale() {  
 if (\_profile.locale == null) return null;  
 var t = \_profile.locale.split("\_");  
 return Locale(t[0], t[1]);  
 }  
  
 // 获取当前Locale的字符串表示  
 String get locale => \_profile.locale;  
  
 // 用户改变APP语言后，通知依赖项更新，新语言会立即生效  
 set locale(String locale) {  
 if (locale != \_profile.locale) {  
 \_profile.locale = locale;  
 notifyListeners();  
 }  
 }  
}