Flutter基础(一)--库的使用

wwp9527

一、Flutter中库的简介

- 在Dart语言中：所有的后缀名为`.dart`文件都是库包。
- 在Flutter工程中，**lib**包为根目录，`package:项目名/`此目录指定的就是**lib**包。

二、库的使用

Flutter中使用库，可以使用以下几种方式

1. `import`导入；
2. `export`重新导出库；
3. `library`声明库；
4. `part`和`part of`关联文件与库；
5. 依赖第三方库。

1、import的使用

1. 格式：`import 库文件的路径.`

```
    import 'package:工程名字/lib下的相对路径';
    import 'lib下的相对路径';
```

2. 当导入库在**lib**包目录下的*lib1.dart*时：
   可以直接写库名：此时默认指向**lib**包内。

```
import 'lib1.dart';
```

   也可以写全路径：

```
import 'package:lib_demo/lib1.dart';
```

3. 当导入库在**lib**目录下自定义包**src**目录下的*lib1.dart*时：要定位到自定义包的位置
   可以直接写库名：此时定位到**lib**包下自定义的**src**目录内。

```
import '/src/lib1.dart';
```

   也可以写全路径：

```
import 'package:lib_demo/src/lib1.dart';
```

2、as 指定库前缀

1. 概述：
   - 当导入的不同库引发的冲突时，使用as指定库前缀用来解决；
   - 当不同库内有相同的类，方法，成员变量，当其被使用时就会发生冲突。
2. **注意事项：**
   1. as指定的库前缀，相当于**C**中的命名空间，使用时用前缀.要是有的对象。
   2. 当**未指定**的前缀的库内有成员（方法，变量，类）和本地文件相同时，此时本地的会覆盖导入的库内的成员，在本地调用时之后执行本地的成员。
3. 使用方法：

```
//在导入路径后 + as +自定义前缀名称
import 'lib下的相对路径' as 自定义名称;
//使用时，以前缀名称开始.调用，相当于c中的命名空间
```

lib1.dart

```
String libName = "lib1";

void getLib(){
    print("我是$libName");
}

class Person{
    String name ;
    int age ;

    void run(){
        print('$name is Running !');
    }
}
```

lib2.dart

```
String libName = "lib2";
```

```
        void getLib(){
            print("我是$libName");
        }

        class Person{
            String name ;
            int age ;

            void run(){
                print('$name is Running !');
            }
        }
```

此时**lib1**与**lib2**内的成员名都相同，都导入使用时，会报`The name 'Person' is defined in the libraries 'lib1.dart' and 'lib2.dart'`这个错误，此时导入要给一个库加一个前缀`as`，以lib1为例：

main.dart

```
import 'lib1.dart' as lib1;//给lib1加个前缀就叫lib1
import 'lib2.dart';

void getLib() {
  print("我是本地成员");
}

main() {
  lib1.getLib();
  //本地方法覆盖掉lib2中的方法
  getLib();

  print('====================');
  //lib1就相当于命名空间，使用时用lib1.要使用的对象
  lib1.Person p1 = new lib1.Person();
  //这里可以看出如果这里libName就是lib2中的成员
  p1.name =libName;
  p1.run();
```

```
  print('====================');
  Person p2 = new Person();
  p2.name = lib1.libName;
  p2.run();
}
//打印结果
我是lib1
我是本地成员
==================
lib2 is Running !
==================
lib1 is Running !
```

3、show和hide操作库的部分

概述：

- show 名称只导入库的这边代码，其余的隐藏（不导入）；
- hide 名称隐藏（不导入）这部分代码，其余的可见（已导入）；
- 当想操作多个成员时，可用,分隔

使用方法：

```
import 'lib下的相对路径' as 自定义名称 show 名称1,名称2,名称3;
//或
import 'lib下的相对路径'  hide 名称1,名称2,名称3;
```

main.dart

```
import 'lib1.dart' as lib1 show Person;//只导入Person类
import 'lib2.dart' hide getLib;//只隐藏getLib()方法

main() {
  //lib1.getLib();  //lib1中的getLib()方法被隐藏
  lib1.Person p1 = new lib1.Person(); //lib1中Person类被导入
  p1.name = libName; //lib2中libName成员未被隐藏 被导
```

```
入了
  p1.run();

  print('==================');
  //getLib(); //lib2中getLib()被隐藏
  Person p2 = new Person(); //lib2中Person未被隐藏
被导入
  //p2.name = lib1.libName; //lib1中libName 成员变量
被隐藏
  p2.name = '张三';
  p2.run();
}
```

4、export重新导入库

1. 概述：
    1. 当需要导入的库过多或者要从新整合库使用时，可以通过重新导入库者，把部分库或全部的库来组合或者重新打包库；
    2. export中也有show和hide操作，但没有as库前缀；
    3. 使用export重新导入多个库时，各个库中有相同名称的成员（类，方法，变量）会发生冲突。
    4. export重新导入的库相当于将库内代码**复制到当前文件中**，但在当前文件并不能使用。因为Dart语言不存在重载，所以会出现冲突。
    5. 部分冲突可以使用hide将冲突的部分隐藏来解决。
2. 使用方法：

lib3.dart

```
String lib3Name = "lib3";
void getLib3(){
    print("我是$lib3Name");
}
class Person3{
    String name ;
    int age ;
}
```

lib4.dart

```
String lib4Name = "lib4";
void getLib4(){
    print("我是$lib4Name");
}
class Person4{
    String name ;
    int age ;
}
```

libs.dart

```
//将要使用的库放在同一个文件下，方便其他文件调用
//export导入的库的内容，当前文件并不能使用
export 'lib3.dart';
export 'lib4.dart' show lib4Name,getLib4;
//lib3和lib4内的成员变量，方法和类的名称不允许相同，否则将会冲突，这也是不能使用as的原因
```

main.dart

```
import 'libs.dart'
```

5、library 声明库

- 使用`library`关键字直接给当前文件标记成库，
- 格式：一般在文件头部`library 自定义名称`
- 声明的库中内容可由下方式导入：
  - 可使用`export`导入的，相当于复制进来，但当前库内引用不到，不过`import`引用当前库时可以使用。
  - 也可使用`part`和`part of`来关联文件与库。`part`的优先级要高于`import`。当前库可以直接应用`part`中的内容。

libs.dart

```
library name;

export 'lib3.dart';
export 'lib4.dart' show lib4Name,getLib4;

```

```
//与lib5.dart文件内的part of配合使用，此时lib5内的代码在本地也可以使用
part 'lib5.dart';
```

lib5.dart

```
part of name; //与name库中part配合使用

String lib5Name = "lib5";
void getLib5(){
    print("我是$lib5Name");
}
class Person5{
    String name ;
    int age ;
}
```

三、依赖第三方库

1. 依赖方式：
   在项目中**pubspec.yaml**文件内的`dependencies:`标签下添加依赖；

```
dependencies:
  flutter:
    sdk: flutter
  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.2
  english_words: any
  #any自动导入与环境兼容的库包，版本号前要有空格，推荐使用any
```

依赖包和flutter头对齐

依赖库名称应该和flutter:的头对齐

- 当添加完依赖后使用`flutter packages get`命令去下载包，或者工具上边`pacages get`按钮；
- 然后去`pubspec.lock`文件中查看导入依赖的兼容版本。

`^version`相当于>= version < big version

10. 依赖方式还分为以下三种：
    1. 常规依赖：`dependencies:`此标签下依赖在调试和发版后都会生效
    2. Dev依赖：`dev_dependencies:`此标签下的依赖均在调试时生效。
    3. 重写依赖：`dependency_overrides:`强制下载依赖包，不管是否兼容，不推荐使用。