

# 一、状态管理、Flutter Getx介绍

## 1.1、状态管理

**通俗的讲：**当我们想在多个页面（组件/Widget）之间共享状态（数据），或者一个页面（组件/Widget）中的多个子组件之间共享状态（数据），这个时候我们就可以用Flutter中的状态管理来管理统一的状态（数据），实现不同组件之间的传值和数据共享。

现在Flutter的状态管理方案很多，redux、bloc、state、provider、Getx。

provider是官方提供的状态管理解决方案，主要功能就是状态管理。Getx是第三方的状态管理插件，不仅具有状态管理的功能，还具有路由管理、主题管理、国际化多语言管理、Obx局部更新、网络请求、数据验证等功能，相比其他状态管理插件Getx 简单、功能强大并且高性能。

## 1.2、Flutter Getx介绍

GetX 是 Flutter 上的一个轻量且强大的解决方案，GetX为我们提供了高性能的状态管理、智能的依赖注入和便捷的路由管理。

- GetX 有3个基本原则：
  - **性能：** GetX 专注于性能和最小资源消耗。GetX 打包后的apk占用大小和运行时的内存占用与其他状态管理插件不相上下。
  - **效率：** GetX 的语法非常简捷，并保持了极高的性能，能极大缩短你的开发时长。
  - **结构：** GetX 可以将界面、逻辑、依赖和路由完全解耦，用起来更清爽，逻辑更清晰，代码更容易维护
- **GetX 并不臃肿，却很轻量。**如果你只使用状态管理，只有状态管理模块会被编译，其他没用到的东西都不会被编译到你的代码中。它拥有众多的功能，但这些功能都在独立的容器中，只有在使用后才会启动。
- Getx有一个庞大的生态系统，能够在Android、iOS、Web、Mac、Linux、Windows和你的服务器上运行。通过[Get Server](#)可以在你的后端完全重用你在前端写的代码。

官网：<https://pub.dev/packages/get>

中文文档：<https://github.com/jonataslaw/getx/blob/master/README.zh-cn.md>

# 二、Flutter Getx 中的Dialog 以及改变主题

## 2.1 Getx安装

将 Get 添加到你的 pubspec.yaml 文件中。

```
dependencies:  
  get: ^4.6.5
```

在需要用到的文件中导入，它将被使用。

```
import 'package:get/get.dart';
```

## 2.2 Getx 使用 Dialog

### 一、设置应用程序入口

当我们导入依赖后，在应用程序顶层把 `GetMaterialApp` 作为顶层，如下所示

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  widget build(BuildContext context) {
    return GetMaterialApp(
      title: "GetX",
      home: Scaffold(
        appBar: AppBar(title: Text("GetX Title")),
      ),
    );
  }
}
```

### 二、调用BottomSheet 以及改变主题

我们可以通过 `GetX` 很轻松的调用 `bottomSheet()`，而且无需传入 `context`，下面我给出一个例子，使用 `GetX` 弹出 `bottomSheet` 并很轻松的实现切换主题。

我们可以通过 `Get.bottomSheet()` 来显示 `BottomSheet`，通过 `Get.back()` 实现路由返回，通过 `Get.changeTheme(ThemeData.dark())` 切换皮肤主题，通过 `Get.isDarkMode` 判断主题样式。

```
ElevatedButton(
  onPressed: () {
    Get.bottomSheet(Container(
      color: Get.isDarkMode ? Colors.black12 : Colors.white,
      height: 200,
      child: Column(
        children: [
          ListTile(
            leading: Icon(Icons.wb_sunny_outlined,
              color:
                Get.isDarkMode ? Colors.white : Colors.black),
            title: Text("白天模式",
              style: TextStyle(
                color: Get.isDarkMode
                  ? Colors.white
                  : Colors.black)),
            onTap: () {
```

```
        Get.changeTheme(ThemeData.light());
        Get.back();
    },
),
ListTile(
  leading: Icon(Icons.wb_sunny,
    color:
      Get.isDarkMode ? Colors.white : Colors.black),
  title: Text("黑夜模式",
    style: TextStyle(
      color: Get.isDarkMode
        ? Colors.white
        : Colors.black)),
  onTap: () {
    Get.changeTheme(ThemeData.dark());
    Get.back();
  },
),
],
),
));
},
child: const Text("Show BottomSheet"))
```

BottomSheet属性和说明

字段	属性	描述
bottomsheet	Widget	弹出的Widget组件
backgroundColor	Color	bottomsheet的背景颜色
elevation	double	bottomsheet的阴影
persistent	bool	是否添加到路由中
shape	ShapeBorder	边框形状，一般用于圆角效果
clipBehavior	Clip	裁剪的方式
barrierColor	Color	弹出层的背景颜色
ignoreSafeArea	bool	是否忽略安全适配
isScrollControlled	bool	是否支持全屏弹出，默认false
useRootNavigator	bool	是否使用根导航
isDismissible	bool	点击背景是否可关闭，默认ture
enableDrag	bool	是否可以拖动关闭，默认true
settings	RouteSettings	路由设置
enterBottomSheetDuration	Duration	bottomsheet进入时的动画时间

字段	属性	描述
exitBottomSheetDuration	Duration	bottomsheet退出时的动画时间

### 三、调用snackbar

Snackbar 和我们前面讲的toast有点相似， 如果想在应用程序中触发某些特定的事件后， 需要弹出快捷消息， 那么使用 Snackbar 则是最佳的选择。

我们可以通过 Get.snackbar() 来显示 snackbar ， 如下所示

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  widget build(BuildContext context) {
    return GetMaterialApp(
      title: "GetX",
      home: Scaffold(
        appBar: AppBar(
          title: Text("GetX Title"),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              ElevatedButton(
                onPressed: () {
                  Get.snackbar("Snackbar 标题", "欢迎使用Snackbar");
                },
                child: Text("显示 Snackbar"))
            ],
          ),
        ),
      ),
    );
  }
}
```

如果您运行了代码，那么恭喜你，你已经会用 Getx 来展示 snackbar 了。你将得到下面的结果

#### Snackbar属性和说明

字段	属性	描述
title	String	弹出的标题文字
message	String	弹出的消息文字
colorText	Color	title和message的文字颜色
duration	Duration	Snackbar弹出的持续时间（默认3秒）
instantInit	bool	当false可以把snackbar 放在 initState，默认true
snackPosition	SnackPosition	弹出时的位置，有两个选项【TOP，BOTTOM】默认TOP
titleText	Widget	弹出标题的组件，设置该属性会导致 title属性失效
messageText	Widget	弹出消息的组件，设置该属性会导致 messageText属性失效
icon	Widget	弹出时图标，显示在title和message的左侧
shouldIconPulse	bool	弹出时图标是否闪烁，默认false
maxWidth	double	Snackbar最大的宽度
margin	EdgeInsets	Snackbar外边距，默认zero
padding	EdgeInsets	Snackbar内边距，默认 EdgeInsets.all(16)
borderRadius	double	边框圆角大小，默认15
borderColor	Color	边框的颜色，必须设置borderWidth，否则无效果
borderWidth	double	边框的线条宽度
backgroundColor	Color	Snackbar背景颜色，默认 Colors.grey.withOpacity(0.2)
leftBarIndicatorColor	Color	左侧指示器的颜色
boxShadows	List	Snackbar阴影颜色
backgroundGradient	Gradient	背景的线性颜色
mainButton	TextButton	主要按钮，一般显示发送、确认按钮
onTap	OnTap	点击Snackbar事件回调
isDismissible	bool	是否开启Snackbar手势关闭，可配合 dismissDirection使用
showProgressIndicator	bool	是否显示进度条指示器，默认false
dismissDirection	SnackDismissDirection	Snackbar关闭的方向
progressIndicatorController	AnimationController	进度条指示器的动画控制器
progressIndicatorBackgroundColor	Color	进度条指示器的背景颜色
progressIndicatorValueColor	Animation	进度条指示器的背景颜色，Animation
snackStyle	SnackStyle	Snackbar是否会附加到屏幕边缘

字段	属性	描述
forwardAnimationCurve	Curve	Snackbar弹出的动画，默认Curves.easeOutCirc
reverseAnimationCurve	Curve	Snackbar消失的动画，默认Curves.easeOutCirc
animationDuration	Duration	Snackbar弹出和小时的动画时长，默认1秒
barBlur	double	Snackbar背景的模糊度
overlayBlur	double	弹出时的毛玻璃效果值，默认0
snackbarStatus	SnackbarStatusCallback	Snackbar弹出或消失时的事件回调（即将打开、已打开、即将关闭、已关闭）
overlayColor	Color	弹出时的毛玻璃的背景颜色
userInputForm	Form	用户输入表单

四、调用defaultDialog

```
ElevatedButton(  
  onPressed: () {  
    Get.defaultDialog(  
      title: "提示",  
      middleText: "您确定退出登录? ",  
      confirm: ElevatedButton(  
        onPressed: () {  
          print("确定");  
          Get.back();  
        },  
        child: const Text("确定")),  
      cancel: ElevatedButton(  
        onPressed: () {  
          print("取消");  
          Get.back();  
        },  
        child: const Text("取消"))),  
  },  
  child: const Text("显示默认的Dialog"))
```

Dialog属性和说明

字段	属性	描述
title	String	弹出的标题，默认 (Alert)
titlePadding	EdgeInsetsGeometry	标题的内边距，默认 (EdgeInsets.all(8))
titleStyle	TextStyle	标题的样式
middleText	String	中间内容区域显示的文字
middleTextStyle	TextStyle	中间内容区域显示的文字样式
content	Widget	弹出的内容，该值设置后middleText将无效
contentPadding	EdgeInsetsGeometry	内容的内边距，默认 (EdgeInsets.all(8))
onConfirm	VoidCallback	确认按钮回调
onCancel	VoidCallback	取消按钮回调
onCustom	VoidCallback	自定义按钮回调
cancelTextColor	Color	取消按钮文字的颜色
confirmTextColor	Color	确认按钮文字的颜色
textConfirm	String	确认按钮的文字
textCancel	String	取消按钮的文字
textCustom	String	自定义按钮的文字
confirm	Widget	确认按钮的组件
cancel	Widget	取消按钮的组件
custom	Widget	自定义按钮的组件
backgroundColor	Color	弹出框的背景颜色
barrierDismissible	bool	是否可以通过点击背景关闭弹窗
buttonColor	Color	按钮的文字颜色，根据按钮类型来设定不同的位置
radius	double	弹出框的圆角大小，默认20
actions	List	增加额外的子组件
onWillPop	WillPopCallback	拦截关闭之前做一些操作
navigatorKey	GlobalKey	用于打开对话框的key

## 三、Flutter Getx 路由管理

GetX 为我们封装了 `Navigation`，无需 `context` 可进行跳转，使用 `GetX` 进行路由跳转非常的简单，只需要调用 `Get.to()` 即可进行路由跳转，`GetX` 路由跳转简化了**跳转动画设置**、**动画时长定义**、**动画曲线设置**。

### 3.1 Get.to()实现普通路由跳转

#### 一、设置应用程序入口

当我们导入依赖后，在应用程序顶层把 `GetMaterialApp` 作为顶层，如下所示

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: "GetX",
      home: Scaffold(
        appBar: AppBar(title: Text("GetX Title")),
      ),
    );
  }
}
```

#### 二、调用to方法切换路由

```
import './Home.dart';
```

```
ElevatedButton(
  onPressed: () async {
    Get.to(Home());
  },
  child: Text("跳转到首页")
)
```

#### 二、调用Get.toNamed()跳转到命名路由



以前

```
Navigator.pushNamed(context, "/login");
```

使用GetX后

```
Get.toNamed("/login");
```

```
Get.toNamed("/shop", arguments: {  
    "id": 20  
});
```

### 三、Get.back(); 返回到上一级页面

以前

```
Navigator.of(context).pop();
```

使用GetX后

```
Get.back();
```

### 四、Get.offAll(); 返回到根

以前

```
Navigator.of(context).pushAndRemoveUntil(  
    MaterialPageRoute(builder: (BuildContext context) {  
        return const Tabs(index: 4);  
    })  
    , (route) => false);
```

使用GetX后

```
Get.offAll( const Tabs(index: 4));
```

### 五、Get.off(NextScreen());

进入下一个页面，但没有返回上一个页面的选项（用于闪屏页，登录页面等）。

```
Get.off(NextScreen());
```

## 六、Flutter Getx 配置路由以及动画

### 一、defaultTransition可以配置默认动画

```
GetMaterialApp(  
  debugShowCheckedModeBanner: false,  
  title: 'Flutter Demo',  
  theme: ThemeData(  
    primarySwatch: Colors.blue,  
    appBarTheme: const AppBarTheme(  
      centerTitle: true,  
    )),  
  initialRoute: "/",  
  defaultTransition: Transition.rightToLeftWithFade,  
  getPages: [  
  
  ],  
);
```

### 二、GetPage 可以配置动态路由

```
import 'package:flutter/material.dart';  
  
import 'package:get/get.dart';  
import './pages/tabs.dart';  
import './pages/shop.dart';  
import './pages/user/login.dart';  
import './pages/user/registerFirst.dart';  
import './pages/user/registerSecond.dart';  
import './pages/user/registerThird.dart';  
  
void main() {  
  runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  const MyApp({Key? key}) : super(key: key);  
  
  // This widget is the root of your application.  
  @override  
  Widget build(BuildContext context) {  
    return GetMaterialApp(  
      debugShowCheckedModeBanner: false,  
      title: 'Flutter Demo',  
      theme: ThemeData(  
        primarySwatch: Colors.blue,  
        appBarTheme: const AppBarTheme(  
          centerTitle: true,  
        )),  
      initialRoute: "/",  
      defaultTransition: Transition.rightToLeftWithFade,  
      getPages: [  

```

```
        GetPage(name: "/", page: () => const Tabs()),
        GetPage(name: "/login", page: () => const LoginPage()),
        GetPage(
            name: "/registerFirst",
            page: () => const RegisterFirstPage(),
            transition: Transition.rightToLeft),
        GetPage(
            name: "/registerSecond", page: () => const RegisterSecondPage()),
        GetPage(name: "/registerThird", page: () => const RegisterThirdPage()),
        GetPage(name: "/shop", page: () => const ShopPage()),
    ],
);
}
```

### 三、Getx 路由跳转传值以及接受数据

路由配置

```
getPages: [
    ...
    GetPage(name: "/shop", page: () => const ShopPage()),
    ...
],
```

跳转传值

```
Get.toNamed("/shop", arguments: {
    "id": 20
});
```

接受数据

```
print(Get.arguments);
```

## 七、Flutter Getx 中间件配置

更多详情参考: <https://github.com/jonataslaw/getx/blob/master/README.zh-cn.md#redirect>

新建shopMiddleware.dart

```
import 'package:flutter/cupertino.dart';
import 'package:get/get.dart';
class ShopMiddleware extends GetMiddleware {
  @override
  // 优先级越低越先执行
  int? get priority => -1;
  @override
  RouteSettings redirect(String ? route){
    return const RouteSettings(name: '/login');
  }
}
```

### GetPage配置路由

```
return GetMaterialApp(
  ...
  initialRoute: "/",
  defaultTransition: Transition.rightToLeftWithFade,
  getPages: [
    GetPage(name: "/", page: () => const Tabs()),
    ...
    GetPage(
      name: "/shop",
      page: () => const ShopPage(),
      middlewares: [ShopMiddleware()]),
  ],
);
```

## 四、Flutter Getx 状态管理

### 4.1、状态管理

目前，Flutter有几种状态管理器。但是，它们中的大多数都涉及到使用ChangeNotifier来更新widget，这对于中大型应用的性能来说是一个很糟糕的方法。你可以在Flutter的官方文档中看到，[ChangeNotifier应该使用1个或最多2个监听器](#)，这使得它们实际上无法用于任何中等或大型应用。

Get 并不是比任何其他状态管理器更好或更差，而是说你应该分析这些要点以及下面的要点来选择只用Get，还是与其他状态管理器结合使用。

Get不是其他状态管理器的敌人，因为Get是一个微框架，而不仅仅是一个状态管理器，既可以单独使用，也可以与其他状态管理器结合使用。

Get有两个不同的状态管理器：响应式状态管理器、简单的状态管理器。

### 4.2、响应式状态管理器

响应式编程可能会让很多人感到陌生，因为它很复杂，但是GetX将响应式编程变得非常简单。

- 你不需要创建StreamControllers.
- 你不需要为每个变量创建一个StreamBuilder.
- 你不需要为每个状态创建一个类。

- 你不需要为一个初始值创建一个get。

使用 Get 的响应式编程就像使用 setState 一样简单。

让我们想象一下，你有一个名称变量，并且希望每次你改变它时，所有使用它的小组件都会自动刷新。

### 4.2.1、响应式状态管理器-计数器

让我们想象一下，你有一个名称变量，并且希望每次你改变它时，所有使用它的小组件都会自动刷新。

这就是你的计数变量。

```
int _counter = 0;
```

要想让它变得可观察，你只需要在它的末尾加上".obs"。

```
RxInt _counter = 0.obs;
```

而在UI中，当你想显示该值并在值变化时更新页面，只需这样做。

```
Obx(() => Text("_counter"));
```

这就是全部，就这么简单。

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key});
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  RxInt _counter = 0.obs;
  @override
```

```
widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      title: const Text("Gex 计数器"),  
    ),  
    body: Center(  
      child: Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: <Widget>[  
          const Text(  
            'You have pushed the button this many times:',  
          ),  
          Obx(() => Text("$_counter"))  
        ],  
      ),  
    ),  
    floatingActionButton: FloatingActionButton(  
      onPressed: () {  
        _counter++;  
      },  
      tooltip: 'Increment',  
      child: const Icon(Icons.add),  
    ), // This trailing comma makes auto-formatting nicer for build methods.  
  );  
}
```

## 4.2.2、声明一个响应式变量三种方式

### 第一种 使用 Rx{Type}

```
final name = RxString('');  
  
final isLoggedIn = RxBool(false);  
  
final count = RxInt(0);  
  
final balance = RxDouble(0.0);  
  
final items = RxList<String>([]);  
  
final myMap = RxMap<String, int>({});
```

### 第二种是使用 Rx, 规定泛型 Rx。

```
final name = Rx<String>('');  
  
final isLoggedIn = Rx<Bool>(false);  
  
final count = Rx<Int>(0);
```

```
final balance = Rx<Double>(0.0);

final number = Rx<Num>(0)

final items = Rx<List<String>>([]);

final myMap = Rx<Map<String, int>>({});

自定义类 - 可以是任何类

final user = Rx<User>();
```

**第三种** 更实用、更简单、更可取的方法，只需添加 .obs 作为value的属性。(推荐)

```
final name = ''.obs;

final isLoggedIn = false.obs;

final count = 0.obs;

final balance = 0.0.obs;

final number = 0.obs;

final items = <String>[].obs;

final myMap = <String, int>{}.obs;

自定义类 - 可以是任何类

final user = User().obs;
```

### 4.2.3、 监听自定义类数据的变化

#### 第一步：应用程序入口设置

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  widget build(BuildContext context) {
    return GetMaterialApp(
      title: "GetX",
      home: MyHomePage(),
    );
  }
}
```

## 第二步：创建Person 类

```
import 'package:get/get.dart';

class Person {
  // rx 变量
  RxString name = "Jimi".obs;
  RxInt age = 18.obs;
}
```

## 第三步：获取类属性值以及改变值

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key});
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {

  var person = Person();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Getx Obx"),
      ),
      body: Center(
        child: Column(
```



```
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Obx(() => Text(
            "我的名字是 ${person.name.value}",
            style: const TextStyle(color: Colors.red, fontSize: 30),
          )),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: (){
        person.name.value = person.name.value.toUpperCase();
      },
      tooltip: 'Increment',
      child: const Icon(Icons.add),
    ), // This trailing comma makes auto-formatting nicer for build methods.
  );
}
```

## 4.3、Flutter Getx 简单的状态管理(依赖管理) GetxController

### 4.3.1、Getx 依赖管理简介

Get有一个简单而强大的依赖管理器，它允许你只用1行代码就能检索到与你的Bloc或Controller相同的类，无需Provider context，无需inheritedWidget。

```
Controller controller = Get.put(Controller());
// 而不是 Controller controller = Controller();
```

想象一下，你已经浏览了无数条路由，现在你需要拿到一个被遗留在控制器中的数据，那你需要一个状态管理器与Provider或Get\_it一起使用来拿到它，对吗？用Get则不然，Get会自动为你的控制器找到你想要的的数据，而你甚至不需要任何额外的依赖关系。

```
Controller controller = Get.find();
//是的，它看起来像魔术，Get会找到你的控制器，并将其提供给你。你可以实例化100万个控制器，Get总会给你正确的控制器。
```

### 4.3.2、多页面之间的数据共享

Flutter默认创建的 "计数器" 项目有100多行 (含注释)，为了展示Get的强大功能，我将使用 GetX 重写一个 "计数器 Plus版"，实现：

- 每次点击都能改变状态
- 在不同页面之间切换
- 在不同页面之间共享状态
- 将业务逻辑与界面分离

## 第一步：应用程序入口设置

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

import 'package:get/get.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      ....
    );
  }
}
```

## 第二步：新建CountController

```
import 'package:get/get.dart';

class CountController extends GetxController{
  var count = 0.obs;

  void inc(){
    count++;
    update();
  }
  void dec(){
    count--;
    update();
  }
}
```

## 第三步：Home.dart执行inc方法

```
import 'package:flutter/material.dart';
import '../controller/count.dart';
import 'package:get/get.dart';
class HomePage extends StatefulWidget {
  const HomePage({super.key});
  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {

  CountController countController = Get.put(CountController());

  @override
  widget build(BuildContext context) {
    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Obx(()=>Text("${countController.count}",style:
Theme.of(context).textTheme.headline1)),
          ElevatedButton(onPressed: (){
            countController.inc();
          }, child: const Text("数值+1"))
        ],
      ),
    );
  }
}
```

## 第四步：Category.dart执行dec方法

你可以让Get找到一个正在被其他页面使用的Controller，并将它返回给你。

```
final countController= Get.find<CountController>();
```

或者

```
final CountController countController = Get.find();
```

```
import 'package:flutter/material.dart';
import '../controller/count.dart';
import 'package:get/get.dart';

class CategoryPage extends StatefulWidget {
  const CategoryPage({super.key});

  @override
  State<CategoryPage> createState() => _CategoryPageState();
}

class _CategoryPageState extends State<CategoryPage> {
  final CountController countController = Get.find();
```

```
@override
widget build(BuildContext context) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Obx(() => Text("${countController.count}",
          style: Theme.of(context).textTheme.headline1)),
        ElevatedButton(
          onPressed: () {
            countController.dec();
          },
          child: const Text("数值-1"))
      ],
    ),
  );
}
```

### 4.3.3、GetXController 绑定数据的几种方法

#### 方法1:

```
CountController countController = Get.put(CountController());
或者
final CountController countController = Get.find();
```

```
Obx(()=>Text("${countController.count}",style:
Theme.of(context).textTheme.headline1)),
```

#### 方法2:

只是绑定数据无需调用 `Get.put(CountController());`

```
GetX<CountController>(
  init: CountController(),
  builder: (controller) {
    return Text(
      "${controller.count}",
      style: const TextStyle(color: Colors.green, fontSize: 30),
    );
  },
),
```

#### 方法3:

```
CountController countController = Get.put(CountController());  
或者  
final CountController countController = Get.find();
```

```
GetBuilder<CountController>(  
    init: countController,  
    builder: (controller) {  
        return Text(  
            "${controller.count}",  
            style: const TextStyle(color: Colors.green, fontSize: 30),  
        );  
    },  
)
```

## 4.4、GetX Binding

**需求:**所有页面都要使用状态管理

在我们使用 `GetX` 状态管理器的时候，往往每次都是用需要手动实例化一个控制器，这样的话基本页面都需要实例化一次，这样就太麻烦了，而 `Binding` 能解决上述问题，可以在项目初始化时把所有需要进行状态管理的控制器进行统一初始化，接下来看代码演示：

在前面的文章中，我们经常使用 `Get.put(MyController())` 来进行控制器实例的创建，这样我们就算不使用控制器实例也会被创建，其实 `GetX` 还提供很多创建实例的方法，可根据不同的业务来进行创建，接下来我们简单介绍一下几个最常用的

- **Get.put():** 不使用控制器实例也会被创建
- **Get.lazyPut():** 懒加载方式创建实例，只有在使用时才创建
- `Get.putAsync():` `Get.put()` 的异步版版本
- `Get.create():` 每次使用都会创建一个新的实例

### 第一步：声明需要进行的绑定控制器类

```
import 'package:get/get.dart';  
  
class BindingHomeController extends GetxController {  
    var count = 0.obs;  
    void increment() {  
        count++;  
    }  
}
```

```
import 'package:get/get.dart';

class BindingMyController extends GetxController {
  var count = 0.obs;
  void increment() {
    count++;
  }
}
```

```
import 'package:get/get.dart';

class AllControllerBinding implements Bindings {

  @override
  void dependencies() {
    // TODO: implement dependencies
    Get.lazyPut<BindingMyController>(() => BindingMyController());
    Get.lazyPut<BindingHomeController>(() => BindingHomeController());
  }
}
```

## 第二步：在项目启动时进行初始化绑定

```
return GetMaterialApp(
  title: "GetX",
  initialBinding: AllControllerBinding(),
  home: GetXBindingExample(),
);
```

## 第三步：在页面中使用状态管理器

```
Obx(() => Text(
  "计数器的值为 ${Get.find<BindingMyController>().count}",
  style: TextStyle(color: Colors.red, fontSize: 30),
)),

SizedBox(height: 20),

ElevatedButton(
  onPressed: () {
    Get.find<BindingMyController>().increment();
  },
  child: Text("点击加1")
),
```

# 五、Flutter GetX 自定义语言包 国际化配置

在我们使用系统自带 `MaterialApp` 来实现国际化配置，需要进行很多配置，而且还需要手动去依赖第三方组件，而使用 `GetX` 来实现国际化配置，你只需要一行代码即可实现切换，接下来我们看一下具体实现。

## 5.2、定义一个语言包

```
import 'package:get/get.dart';

class Messages extends Translations {
  @override
  Map<String, Map<String, String>> get keys => {
    'zh_CN': {
      'hello': '你好 世界',
    },
    'de_DE': {
      'hello': 'Hallo welt',
    }
  };
}
```

## 5.2、应用程序入口配置

- **translations:** 国际化配置文件
- **locale:** 设置默认语言，不设置的话为系统当前语言
- **fallbackLocale:** 添加一个回调语言选项，以备上面指定的语言翻译不存在

```
return GetMaterialApp(
  translations: Messages(), // 你的翻译
  locale:const Locale('zh', 'CN'), // 将会按照此处指定的语言翻译
  fallbackLocale:const Locale('en', 'US'), // 添加一个回调语言选项，以备上面指定的语言翻译不存在
);
```

## 5.3、调用语言包

只要将 `.tr` 追加到指定的键上，就会使用 `Get.locale` 和 `Get.fallbackLocale` 的当前值进行翻译。

```
Text('title'.tr);
```

## 5.4、改变语言

调用 `Get.updateLocale(locale)` 来更新语言环境。然后翻译会自动使用新的 `locale`。

更新后所有页面生效

```
var locale = Locale('en', 'US');
Get.updateLocale(locale);
```

## 5.5、完整代码

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import './language/message.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: 'Flutter Demo',
      translations: Messages(), // 你的翻译
      locale: const Locale('zh', 'CN'), // 设置默认语言
      fallbackLocale: const Locale('en', 'US'), // 添加一个回调语言选项，以备上面指定的语言翻译不存在
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key});

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Title'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('title'.tr),
            const SizedBox(),
            Text('hello'.tr),
            ElevatedButton(
              onPressed: () {
                var locale = const Locale('zh', 'CN');
                Get.updateLocale(locale);
              },
              child: const Text("切换到中文")),
          ],
        ),
      ),
    );
  }
}
```



```
const SizedBox(  
  height: 20,  
)  
,  
ElevatedButton(  
  onPressed: () {  
    var locale = const Locale('en', 'US');  
    Get.updateLocale(locale);  
  },  
  child: const Text("切换到英文")),  
],  
)  
,  
)  
);  
}
```

## 六、Flutter GetX GetConnect(了解)

GetConnect可以便捷的通过http或websockets进行前后台通信。

你能轻松的通过extend GetConnect就能使用GET/POST/PUT/DELETE/SOCKET方法与你的Rest API或websockets通信。

### 6.1、默认配置

你能轻松的通过extend GetConnect就能使用GET/POST/PUT/DELETE/SOCKET方法与你的Rest API或websockets通信。

```
class UserProvider extends GetConnect {  
  // Get request  
  Future<Response> getUser(int id) => get('http://youapi/users/$id');  
  // Post request  
  Future<Response> postUser(Map data) => post('http://youapi/users', body:  
data);  
  // Post request with File  
  Future<Response<CasesModel>> postCases(List<int> image) {  
    final form = FormData({  
      'file': MultipartFile(image, filename: 'avatar.png'),  
      'otherFile': MultipartFile(image, filename: 'cover.png'),  
    });  
    return post('http://youapi/users/upload', form);  
  }  
  
  GetSocket userMessages() {  
    return socket('https://yourapi/users/socket');  
  }  
}
```

## 6.2、自定义配置

GetConnect具有多种自定义配置。你可以配置base Url，配置响应，配置请求，添加权限验证，甚至是尝试认证的次数，除此之外，还可以定义一个标准的解码器，该解码器将把您的所有请求转换为您的模型，而不需要任何额外的配置。

```
class HomeProvider extends GetConnect {
  @override
  void onInit() {
    // All request will pass to jsonEncode so CasesModel.fromJson()
    httpClient.defaultDecoder = CasesModel.fromJson;
    httpClient.baseUrl = 'https://api.covid19api.com';
    // baseUrl = 'https://api.covid19api.com'; // It define baseUrl to
    // Http and websockets if used with no [httpClient] instance

    // It's will attach 'apikey' property on header from all requests
    httpClient.addRequestModifier((request) {
      request.headers['apikey'] = '12345678';
      return request;
    });

    // Even if the server sends data from the country "Brazil",
    // it will never be displayed to users, because you remove
    // that data from the response, even before the response is delivered
    httpClient.addResponseModifier<CasesModel>((request, response) {
      CasesModel model = response.body;
      if (model.countries.contains('Brazil')) {
        model.countries.remove('Brazil');
      }
    });

    httpClient.addAuthenticator((request) async {
      final response = await get("http://yourapi/token");
      final token = response.body['token'];
      // Set the header
      request.headers['Authorization'] = "$token";
      return request;
    });

    //Authenticator will be called 3 times if HttpStatus is
    //HttpStatus.unauthorized
    httpClient.maxAuthRetries = 3;
  }

  @override
  Future<Response<CasesModel>> getCases(String path) => get(path);
}
```

## 七、Flutter GetX 其他高级API

```
// 给出当前页面的args。
```

```
Get.arguments
```

```
//给出以前的路由名称
```

```
Get.previousRoute
```

```
// 给出要访问的原始路由，例如，rawRoute.isFirst()
```

```
Get.rawRoute
```

```
// 允许从GetObserver访问Rounting API。
```

```
Get.routing
```

```
// 检查 snackbar 是否打开
```

```
Get.isSnackbarOpen
```

```
// 检查 dialog 是否打开
```

```
Get.isDialogOpen
```

```
// 检查 bottomsheet 是否打开
```

```
Get.isBottomSheetOpen
```

```
// 删除一个路由。
```

```
Get.removeRoute()
```

```
//反复返回，直到表达式返回真。
```

```
Get.until()
```

```
// 转到下一条路由，并删除所有之前的路由，直到表达式返回true。
```

```
Get.offUntil()
```

```
// 转到下一个命名的路由，并删除所有之前的路由，直到表达式返回true。
```

```
Get.offNamedUntil()
```

```
//检查应用程序在哪个平台上运行。
```

```
GetPlatform.isAndroid
```

```
GetPlatform.isIOS
```

```
GetPlatform.isMacOS
```

```
GetPlatform.iswindows
```

```
GetPlatform.isLinux
```

```
GetPlatform.isFuchsia
```

```
//检查设备类型
```

```
GetPlatform.isMobile
```

```
GetPlatform.isDesktop
```

```
//所有平台都是独立支持web的！
```

```
//你可以知道你是否在浏览器内运行。
```

```
//在windows、ios、OSX、Android等系统上。
```

```
GetPlatform.isweb
```

```
// 相当于.MediaQuery.of(context).size.height,
```

```
//但不可改变。
```

```
Get.height
```

```
Get.width
```

```
// 提供当前上下文。
```

```
Get.context
```

```
// 在你的代码中的任何地方，在前台提供 snackbar/dialog/bottomsheet 的上下文。
```

```
Get.contextOverlay
```

```
// 注意：以下方法是对上下文的扩展。
```

```
// 因为在你的UI的任何地方都可以访问上下文，你可以在UI代码的任何地方使用它。

// 如果你需要一个可改变的高度/宽度（如桌面或浏览器窗口可以缩放），你将需要使用上下文。
context.width
context.height

// 让您可以定义一半的页面、三分之一的页面等。
// 对响应式应用很有用。
// 参数： dividedBy (double) 可选 - 默认值： 1
// 参数： reducedBy (double) 可选 - 默认值： 0。
context.heightTransformer()
context.widthTransformer()

/// 类似于 MediaQuery.of(context).size。
context.mediaQuerySize()

/// 类似于 MediaQuery.of(context).padding。
context.mediaQueryPadding()

/// 类似于 MediaQuery.of(context).viewPadding。
context.mediaQueryViewPadding()

/// 类似于 MediaQuery.of(context).viewInsets。
context.mediaQueryViewInsets()

/// 类似于 MediaQuery.of(context).orientation;
context.orientation()

///检查设备是否处于横向模式
context.isLandscape()

///检查设备是否处于纵向模式。
context.isPortrait()

///类似于MediaQuery.of(context).devicePixelRatio。
context.devicePixelRatio()

///类似于MediaQuery.of(context).textScaleFactor。
context.textScaleFactor()

///查询设备最短边。
context.mediaQueryShortestSide()

///如果宽度大于800，则为真。
context.showNavbar()

///如果最短边小于600p，则为真。
context.isPhone()

///如果最短边大于600p，则为真。
context.isSmallTablet()

///如果最短边大于720p，则为真。
context.isLargeTablet()

///如果当前设备是平板电脑，则为真
context.isTablet()
```

```
///根据页面大小返回一个值<T>。  
///可以给值为:  
///watch: 如果最短边小于300  
///mobile: 如果最短边小于600  
///tablet: 如果最短边 (shortestSide) 小于1200  
///desktop: 如果宽度大于1200  
context.responsiveValue<T>()
```

## 八、Flutter GetX Get Service

这个类就像一个 `GetxController`，它共享相同的生命周期 `onInit()`、`onReady()`、`onClose()`。但里面没有“逻辑”。它只是通知GetX的依赖注入系统，这个子类**不能**从内存中删除。所以如果你需要在你的应用程序的生命周期内对一个类实例进行绝对的持久化，那么就可以使用 `GetxService`。

所以这对保持你的 "服务" 总是可以被 `Get.find()` 获取到并保持运行是超级有用的。比如

`ApiService`，`StorageService`，`CacheService`。

### 8.1、定义并使用ApiService

#### 第一步、定义一个ApiService

新建 `services/api.dart`

```
import 'package:get/get.dart';  
  
class ApiService extends GetxService {  
  String api() {  
    return "http://jdmall.itying.com/";  
  }  
}
```

#### 第二步、注册服务

```
import './services/api.dart';  
  
void main() async{  
  await initServices();  
  runApp(const MyApp());  
}  
  
Future<void> initServices() async {  
  print("初始化服务");  
  await Get.putAsync(() async => ApiService());  
  print("所有服务启动");  
}
```

#### 第三步、调用服务

```
import './services/api.dart';

Get.find<ApiService>().api()
```

## 8.2、定义并使用StorageService

### 第一步、定义一个StorageService

新建 `services/storage.dart`

```
import 'package:get/get.dart';
import 'package:shared_preferences/shared_preferences.dart';

class StorageService extends GetxService {
  Future<void> getCounter() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    int count = (prefs.getInt("counter") ?? 0) + 1;
    print("count 的值为: $count");
    await prefs.setInt("counter", count);
  }
}
```

### 第二步、注册服务

```
import './services/api.dart';
import './services/storage.dart';

void main() async{
  await initServices();
  runApp(const MyApp());
}

Future<void> initServices() async {
  print("初始化服务");
  await Get.putAsync(() async => ApiService());
  await Get.putAsync(() async => StorageService());
  print("所有服务启动");
}
```

### 第三步、调用服务

```
import './services/storage.dart';

Get.find<StorageService>().getCounter();
```

## 九、GetView介绍 以及 GetxController生命周期

`GetView` 只是对已注册的 `Controller` 有一个名为 `controller` 的getter的 `const Stateless` 的 `Widget`，如果我们只有单个控制器作为依赖项，那我们就可以使用 `GetView`，而不是使用 `StatelessWidget`，并且避免了写 `Get.find()`。

### GetView如何使用

`GetView` 的使用方法非常简单，只是要将你的视图层继承自 `GetView` 并传入需要注册的控制器并 `Get.put()` 即可，我们来看下代码演示

### 9.1、GetView结合GetxController使用

#### 第一步、定义一个CountController

```
import 'package:get/get.dart';

class CountController extends GetxController{
  var count = 0.obs;

  @override
  void onInit() {
    super.onInit();
    print("onInit");
  }

  @override
  void onReady() {
    super.onReady();
    print("onReady");
  }

  @override
  void onClose() {

    print("onClose");
  }
  void inc(){
    count++;
    update(['first_count']);
  }
  void dec(){
    count--;
    update();
  }
}
```

#### 第二步、继承GetView并使用状态管理

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import '../controller/count.dart';

class ShopPage extends GetView<CountController> {
  const ShopPage({super.key});
  @override
  widget build(BuildContext context) {
    //如果第一次使用还需要put
    Get.put(CountController());

    return Scaffold(
      appBar: AppBar(
        title: const Text("shop"),
      ),
      body: Center(
        child: Column(
          children: [
            Obx(()=>Text("${controller.count}")),
            ElevatedButton(onPressed: (){
              controller.inc();
            }, child: const Text("加1"))
          ],
        ),
      ),
    );
  }
}
```

## 9.2、GetView Binding结合GetxController使用

### 第一步、定义一个shopController

```
import 'package:get/get.dart';

class shopController extends GetxController{
  var count = 0.obs;

  @override
  void onInit() {
    super.onInit();
    print("onInit");
  }

  @override
  void onReady() {
    super.onReady();
    print("onReady");
  }

  @override
```



```
void onClose() {  
  
    print("onClose");  
}  
void inc(){  
    count++;  
    update(['first_count']);  
}  
}
```

## 第二步、定义一个shop Binding

```
import 'package:get/get.dart';  
import '../controllers/shop.dart';  
class ShopBinding implements Bindings{  
    @override  
    void dependencies() {  
        // TODO: implement dependencies  
        Get.lazyPut<ShopController>(() => ShopController());  
    }  
  
}
```

## 第三步、路由中绑定Binding

```
import 'package:get/get.dart';  
import '../pages/tabs.dart';  
import '../pages/shop.dart';  
import '../middlewares/shopMiddleware.dart';  
import "../binding/shop.dart";  
class AppPage {  
    static final routes = [  
        GetPage(name: "/", page: () => const Tabs()),  
        GetPage(  
            name: "/shop",  
            page: () => const ShopPage(),  
            binding: ShopBinding(),  
            middlewares: [ShopMiddleware()],  
        ),  
    ];  
}
```

## 第四步、继承GetView并使用状态管理

```
import 'package:flutter/material.dart';  
import 'package:get/get.dart';  
import '../controllers/shop.dart';  
  
class ShopPage extends GetView<ShopController> {  
    const ShopPage({super.key});  
  
    @override  
    widget build(BuildContext context) {
```

```
return Scaffold(  
  appBar: AppBar(  
    title: const Text('Title'),  
  ),  
  body: Center(child: Obx(() {  
    return Text("${controller.counter}");  
  })),  
);  
}
```

## 十、GetX GetUtils

`GetUtils` 是 `getx` 为我们提供一些常用的工具类库，包括**值是否为空**、**是否是数字**、**是否是视频**、**图片**、**音频**、**PPT**、**Word**、**APK**、**邮箱**、**手机号码**、**日期**、**MD5**、**SHA1**等等。

```
import 'package:flutter/material.dart';  
import 'package:get/get.dart';  
void main() {  
  runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  // This widget is the root of your application.  
  @override  
  Widget build(BuildContext context) {  
    return GetMaterialApp(  
      title: 'Flutter Demo',  
      theme: ThemeData(  
        primarySwatch: Colors.blue,  
      ),  
      home: const HomePage(),  
    );  
  }  
}  
  
class HomePage extends StatefulWidget {  
  const HomePage({super.key});  
  
  @override  
  State<HomePage> createState() => _HomePageState();  
}  
  
class _HomePageState extends State<HomePage> {  
  
  var textFieldController = TextEditingController();  
  
  @override
```

```
widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      title: const Text('Title'),  
    ),  
    body: Center(  
      child: Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        crossAxisAlignment: CrossAxisAlignment.center,  
        children: [  
          Padding(  
            padding: const EdgeInsets.all(20),  
            child: TextField(  
              controller: textEditingController,  
            ),  
          ),  
          const SizedBox(height: 10,),  
          Padding(  
            padding: const EdgeInsets.all(10),  
            child: ElevatedButton(  
              child: const Text("判断是否是邮箱"),  
              onPressed: () async {  
                if (GetUtils.isEmail(textEditingController.text)) {  
                  Get.snackbar("正确", "恭喜你，完全正确", backgroundColor:  
Colors.greenAccent);  
                } else {  
                  Get.snackbar(  
                    "邮箱错误",  
                    "请输入正确的邮箱",  
                    backgroundColor: Colors.pink  
                  );  
                }  
              },  
            ),  
          ),  
          Padding(  
            padding: const EdgeInsets.all(10),  
            child: ElevatedButton(  
              child: const Text("判断是否是手机号"),  
              onPressed: () async {  
                if (GetUtils.isPhoneNumber(textEditingController.text)) {  
  
                  Get.snackbar("正确", "恭喜你，完全正确", backgroundColor:  
Colors.greenAccent);  
                } else {  
                  Get.snackbar(  
                    "手机号错误",  
                    "请输入正确的手机号",  
                    backgroundColor: Colors.pink  
                  );  
                }  
              },  
            ),  
          ),  
          Padding(  
            padding: EdgeInsets.all(10),  
            child: ElevatedButton(  
              child: Text("判断是否是IPv4"),  

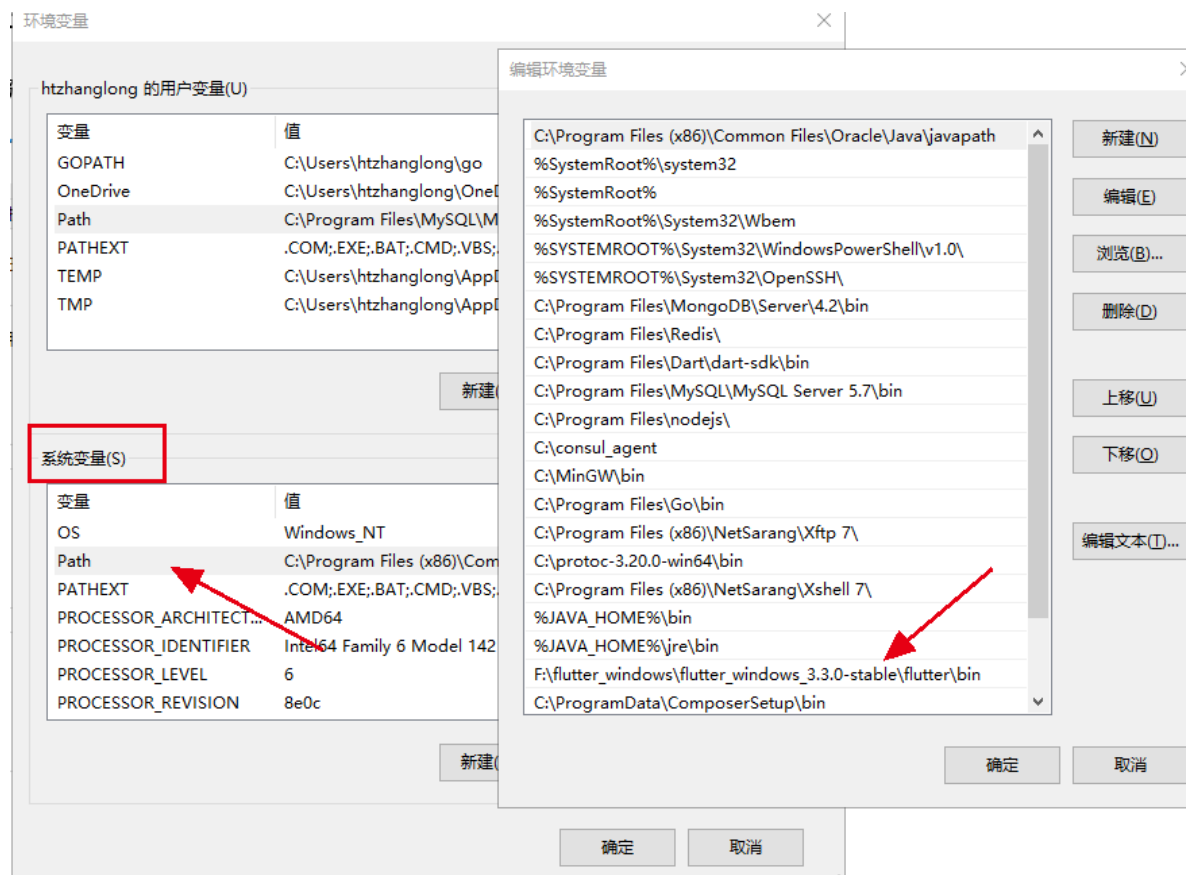
```

## 十一、Flutter get\_cli的使用

下面路径添加到Path环境变量里面

```
F:\flutter_windows\flutter_windows_3.3.0-stable\flutter\.pub-cache\bin
```

```
F:\flutter_windows\flutter_windows_3.0.5\flutter\bin\cache\dart-sdk\bin
```



## 2、mac安装get\_cli

```
pub global activate get_cli
```

```
flutter pub global activate get_cli
```

### 配置环境变量

```
Warning: Pub installs executables into $HOME/.pub-cache/bin, which is not on your path.
```

一般 Mac 的环境变量都是通过根目录的 `.bash_profile` 进行环境变量设置。

```
vim ~/.bash_profile  
vim ~/.zshrc
```

```
export PATH=/Users/aishengwanwu/flutter_mac/flutter/bin:$PATH

export PUB_HOSTED_URL=https://pub.flutter-io.cn

export FLUTTER_STORAGE_BASE_URL=https://storage.flutter-io.cn

....

export PATH="$PATH":"$HOME/.pub-cache/bin"

export PATH=/Users/aishengwanwu/flutter_mac/flutter/bin\cache\dart-sdk\bin:$PATH
```

让配置环境变量生效

```
source ~/.bash_profile
source ~/.zshrc
```

不管是执行 `fvm list` 还是 `fvm --version`，都报错：

```
Can't load kernel binary: Invalid SDK hash.
```

最后的解决方案是执行：

```
dart pub global activate fvm
```