# 安装

## 基础环境变量配置

1. 将以下变量添加到用户变量上

变量名：PUB\_HOSTED\_URL

变量值：https://pub.flutter-io.cn

变量名：FLUTTER\_STORAGE\_BASE\_URL

变量值：<https://storage.flutter-io.cn>

1. 安装GIT
2. 克隆flutter.git到本地

git clone -b beta <https://github.com/flutter/flutter.git>

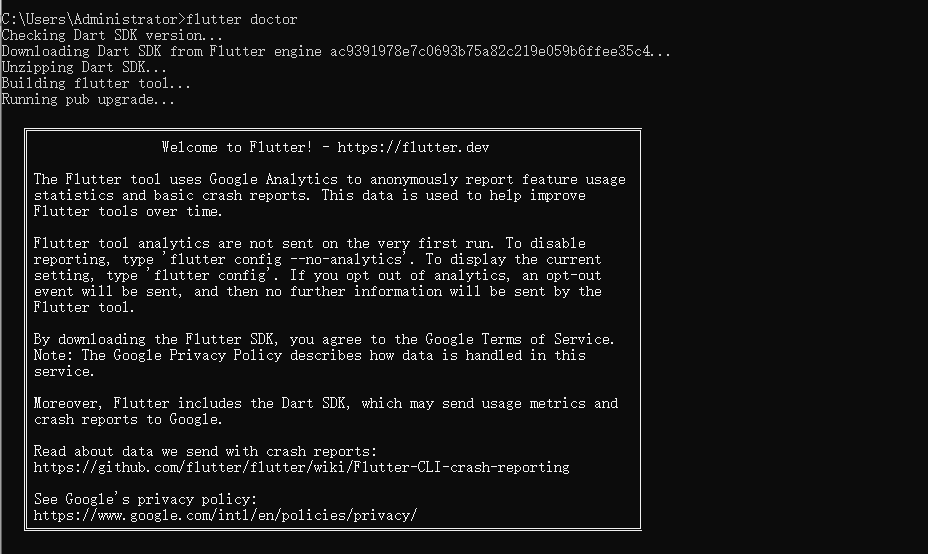
1. 将flutter加入用户变量path

把flutter复制到指定位置，加入用户变量path 例如：F:\myapp\flutter\bin

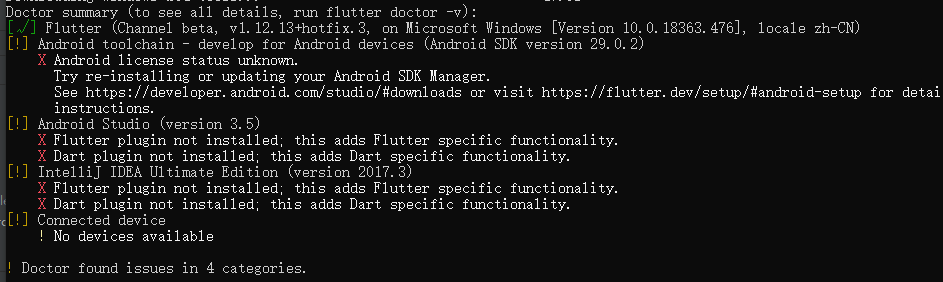
1. 运行flutter doctor

打开CMD 运行flutter doctor 来验证是否已经正确地设置

第一次界面



后边这个指令会快速查看是否有什么插件没安装

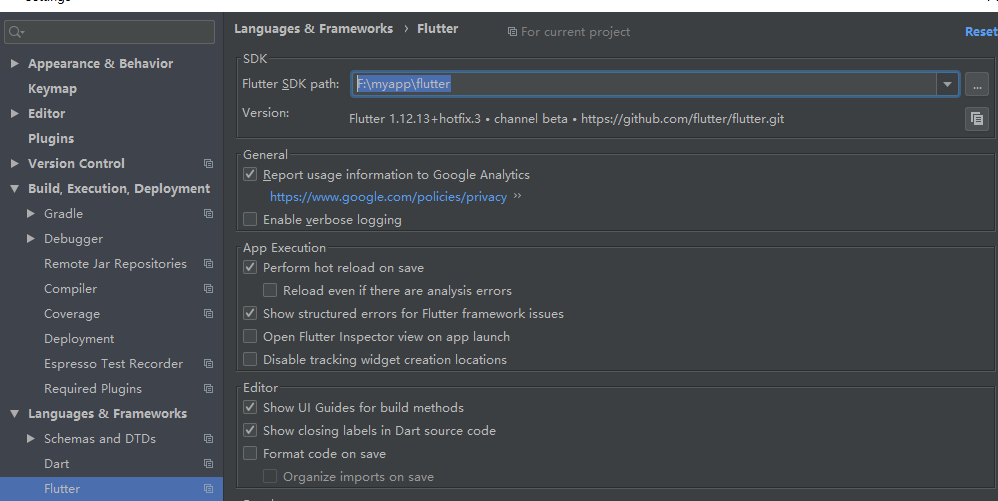


## Android studio配置

1. 导入Dart、Flutter插件

Setting-plugins 搜索对应名字并安装、重启

1. Flutter配置路径



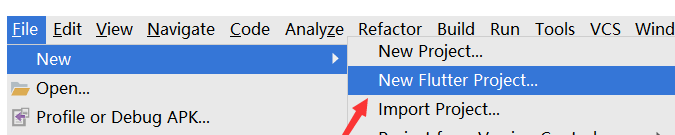
选择刚才git clone 的文件夹，例如：F:\myapp\flutter 成功后会有版本提示

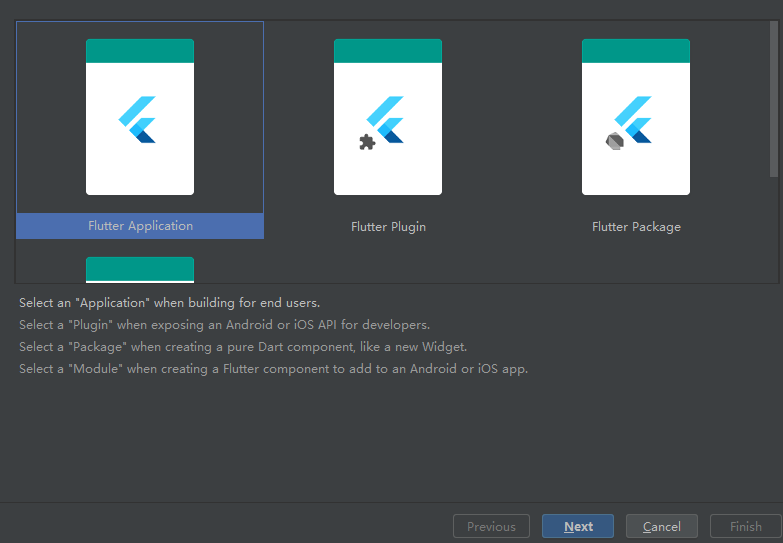
1. Dart配置路径

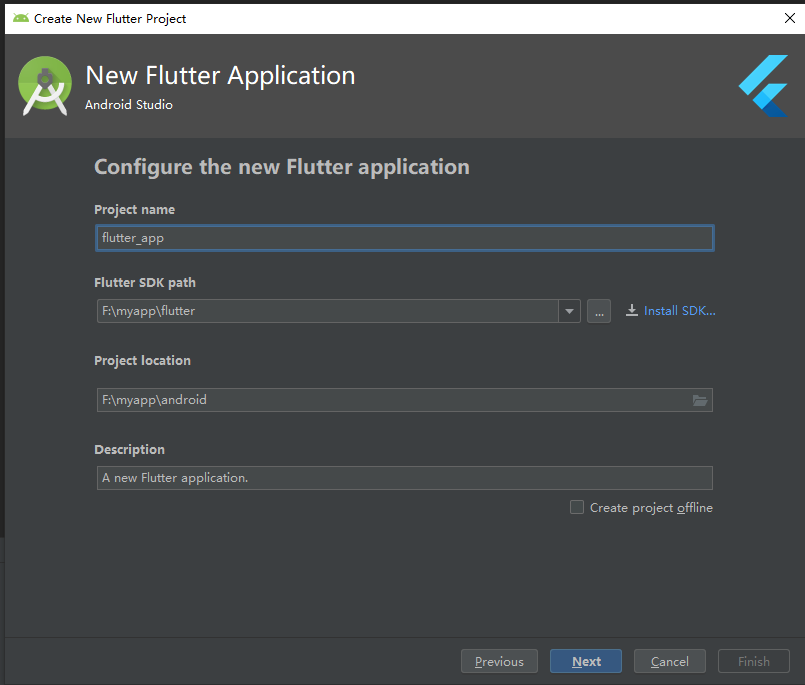
选了Flutter的路径后，Dart的路径会自动匹配，如果没有，手动操作下



1. 创建应用

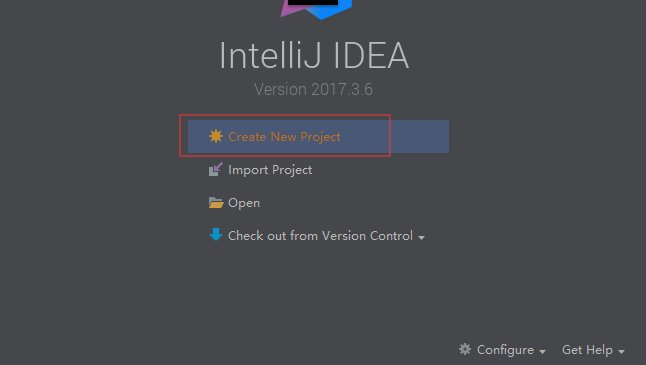




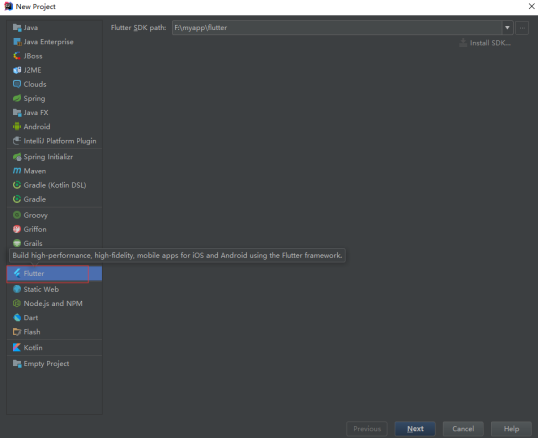


## IntelliJ IDEA配置 //AVD需配置

1. 导入Dart、Flutter插件,此步同上
2. Flutter配置路径

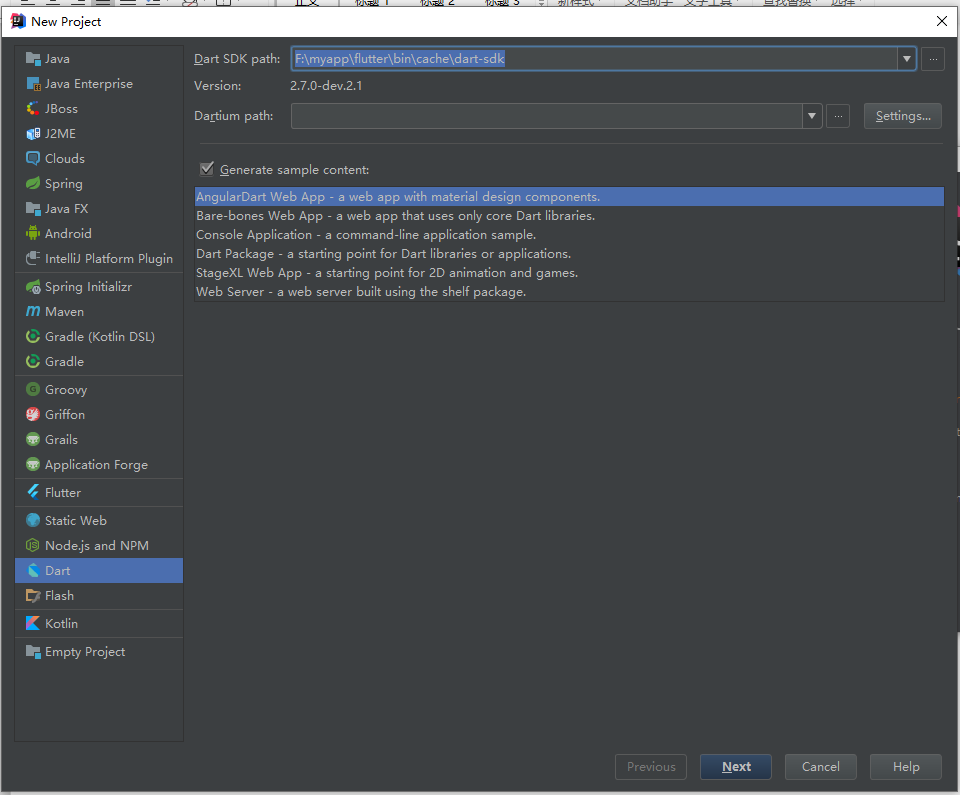


一般Flutter的路径会自动配上，不对手动操作下



1. Dart配置路径

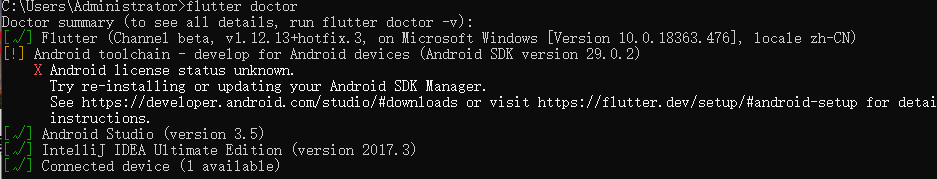
Dart需手动配置F:\myapp\flutter\bin\cache\dart-sdk



1. 新建项目

## Android license status unknown 解决方案

这个问题可以不解决，后边感觉没有影响



## VScode插件配置

auto rename tag 输入开始标签后，自动添加结束标签

Awesome Flutter Snippets stlss 生成 主要结构

Color Info 移上去可以看到颜色的各种表示方法

Color Picker 颜色选取

dart-import

flutter-stylizer

GitLens 查看git的修改人，修改时间

## 导入新项目流程

导入一个新项目

flutter upgrade 因为flutter版本会不同，所以先更新flutter版本

flutter packages get 安装插件

flutter doctor 查看各种是否配置完成

flutter doctor --android-licenses 如果licenses不对，执行这个，这个需要配置JAVA\_HOME,执行后让选择就Y

flutter run 运行，但有的时候有多个设备

To be filled by O E M • 192.168.50.251:5555 • android-x64 • Android 8.1.0 (API 27)

Android SDK built for x86 • emulator-5554 • android-x86 • Android 8.0.0 (API 26) (emulator)

红色的为设备名

flutter run -d emulator-5554

## 修改服务下载路径

### 升级回滚清除

Your flutter checkout has local changes that would be erased by upgrading. If you want to keep these changes, it is recommended that you stash them via "git stash" or else commit the changes to a local branch. If it is okay to remove local changes, then re-run this command with --force.

大致意思：您的flutter检测有局部变化，将升级回滚清除。如果要保留这些更改，建议您通过“git stash”存储它们，或者将更改提交到本地分支。如果可以的话要删除本地更改，请使用--force重新运行此命令。

初次安装Flutter时都是按照文档一步一步来的，不清楚这个局部变化 怎么来的。这次更新使用指令flutter upgrade --force

flutter upgrade --force

如果是下载下来的安装包，第一次升级有大概率出现这种

### 修改服务路径

项目中的 build.gradle

两处

//google()  
//jcenter()  
maven { url 'https://maven.aliyun.com/repository/google' }  
maven { url 'https://maven.aliyun.com/repository/jcenter' }  
maven { url 'http://maven.aliyun.com/nexus/content/groups/public' }

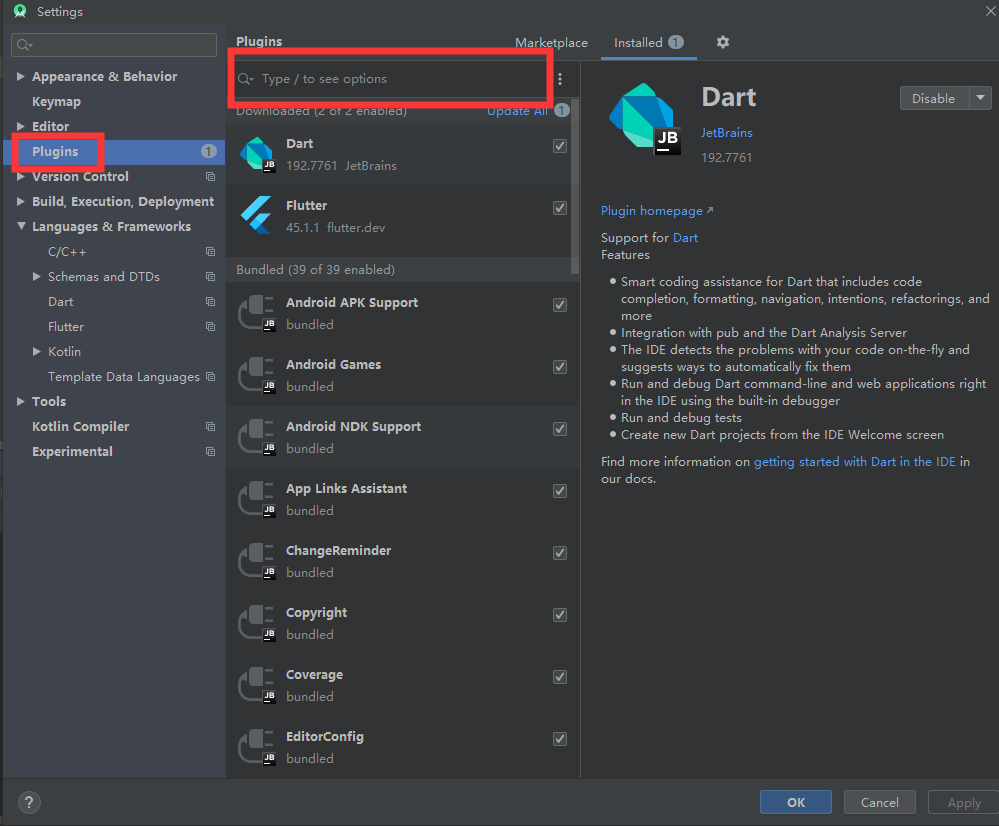
Fluttersdk Flutter -> packages -> flutter\_tools -> gradle  
找到 flutter.gradle 文件

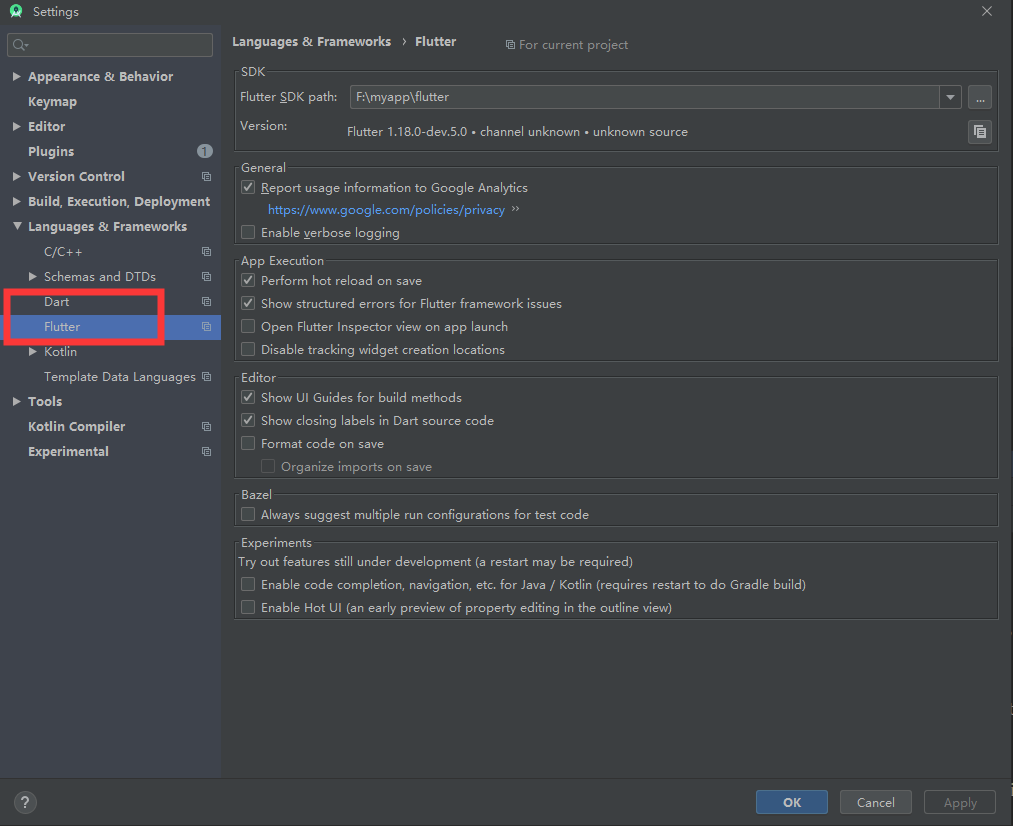
//google()  
//jcenter()  
maven { url 'https://maven.aliyun.com/repository/google' }  
maven { url 'https://maven.aliyun.com/repository/jcenter' }  
maven { url 'http://maven.aliyun.com/nexus/content/groups/public' }

### 修改 android Studio 中Flutter/Drat配置

Flie - Setings

第一次要下载插件





## 基本命令

| **Flutter 常用命令** | **说明** |
| --- | --- |
| flutter | 列出所有的命令 |
| flutter help | 查看具体命令的帮助信息 |
| flutter doctor | 查看是否还需要安装其它依赖 |
| flutter doctor -v | 查看详细信息 |
| flutter channel | 查看 Flutter SDK 所有分支 |
| flutter channel stable | 切换分支 |
| flutter upgrade | 升级 Flutter SDK（此命令会同时更新 Flutter SDK 和你的 Flutter 项目依赖包） |
| flutter packages get | 获取项目所有的依赖包（只更新项目依赖包，不包括 Flutter SDK） |
| flutter packages upgrade | 获取项目所有依赖包的最新版本（只更新项目依赖包，不包括 Flutter SDK） |
| flutter analyze | 分析项目代码 |
| flutter build apk |  |
| flutter build ios |  |

# Dark语法

<https://www.kancloud.cn/marswill/dark2_document/728986>

## 基础特性

### 定义一个函数

printInteger(int aNumber) {

print('The number is $aNumber.'); // Print to console.

}

### 执行程序入口

main() {

var number = 42; // Declare and initialize a variable.

printInteger(number); // Call a function.

}

### 一个字符串字面量

$variableName 或者 ${expression}

### 一种声明变量而不指定其类型的方法。

var

## 变量与常量

### var 变量存储引用

var name = 'Bob';

### Object变量：指定对象

这个种变量有length，可以先把它赋值int，又赋值string，这样都不会出错

Object name = 'Bob';

### dynamic 变量动态类型

这个种变量没有length，相当于更弱类型的Object

dynamic name = 'Bob';

### const常量

编译阶段就已经存在，相当于更小范围的final

const name = 'Bob';

### final常量，包括实例

什么时候用，什么时候初始化

final name = 'Bob';

## 数据类型

### 数字

分为 int 和 double

转换方法

// String -> int

var one = int.parse('1');assert(one == 1);

// String -> double

var onePointOne = double.parse('1.1');assert(onePointOne == 1.1);

// int -> String

String oneAsString = 1.toString();assert(oneAsString == '1');

// double -> String

### 字符串：String

### 布尔：bool

### 数组：List

List list = [1, 2, 3, 4];  
list.add(100); //向数组中加一个值  
list.addAll([7, 8, 9]); //向数组加一个数组  
print(list);  
print(list.length); // 数组的长度  
print(list.first); // 数组的第一个值  
print(list.last); // 数组的最后一个值  
print(list[4]); //数组的第4个元素

### 对象：Maps

//一种定义方式 和js一样  
Map obj = {'x':1,'y':2,'z':3};  
//另一种定义方式  
Map obj2=new Map();  
obj2['x']=1;  
obj2['y']=2;  
print(obj.containsKey('x')); //是否包含 x 这个key值  
obj.remove('y'); //删除 key为y 的一组对象  
print(obj);

### 字符:Runes //todo

### 符号:Symbols //todo

## Function

### 有无返回值

Dark省去了java的private 直接void | String | int 等数据类型+方法名，void是无返回值

String getString(){  
 return '你好';  
}

### 函数带参数

String getStringInfo(int userId){  
 return '你好'+userId.toString();  
}

### 函数带可选参数

int addAge(int age1,[int age2]){  
 return age1 - (age2!=null ? age2 : 0);  
}

使用方法 addAge(23,2)

### 函数参数有默认值

int addAge(int age1,[int age2]){  
 return age1 - (age2!=null ? age2 : 0);  
}

使用方法 addAge2(age1: 23,age2: 3)

### 匿名函数

var list=[1,2,3,4,5];  
list.forEach((str){  
 print(str);  
});

数组.fotEach(分解)

分解:(){} ()可任意定义一个变量，在{}中，刚才那个变量就变成了每一项

## 类与继承

类名一般首字母大写

### 类的实例化

var p= new Person();

### 构造函数

构造函数实际上是建一个跟类同名的方法

class Persons{

int age;

String name;

Persons(int age, String name){

this.age=age;

this.name=name;

}

void sayhello(){

print('我叫'+ this.name+',我今年'+this.age.toString()+'岁了');

}

}

### 继承extends

如果继承的父类有构造函数，继承了他就要实现他的构造器

class Work extends Persons{  
 Work(int age, String name) : super(age, name);  
}

如果除继承外还需要实现其它自己的参数，可在构造器中添加

class Work extends Persons{  
 String sex;  
 Work(int age, String name,String sex) : super(age, name){  
 this.sex=sex;  
 }  
}

// 实现

// var s=new Work(12, '李四', '女');

### 继承之@override

重写父类的方法 在方法名前加 @override ，也可不加，但最好申明，便于后边阅读代码

class Work extends Persons{  
 String sex;  
 Work(int age, String name,String sex) : super(age, name){  
 this.sex=sex;  
 }  
 @override  
 void sayhello(){  
 print('work类，我叫'+ this.name+',我今年'+this.age.toString()+'岁了');  
 }  
}

## Mixin与抽象类

### Mixin继承多个类

这里实际关键词是with ，不是minin，并且这些类不能有构造函数

class Work{  
 void sayhello(){  
 print('work类');  
 }  
}  
  
class Life{  
 void sayhello(){  
 print('Life类');  
 }  
}  
  
class Persons with Work, Life{  
 int age;  
 String name;  
 Persons(int age, String name){  
 this.age=age;  
 this.name=name;  
 }  
 void sayhello(){  
 print('我叫'+ this.name+',我今年'+this.age.toString()+'岁了');  
 }  
}

3个类中都有 sayhello，后边执行这个方法的时候，

类自身具备的方法>继承的第二个类的方法>继承的第一个类的方法

相当于后边把前边的覆盖了

### 抽象类（定规则）

概念：我有一个想法，但不知道怎么实现，后边如果继承了这个类，就必须实现这些方法

this.name=name;  
 }  
 void sayhello(){  
 print('我叫'+ this.name+',我今年'+this.age.toString()+'岁了');  
 }  
}

抽象类的两个特点

1. 可以在with中使用
2. 不能够被实例化（它只是一个规则，并不是被人用的）

## 使用库

### improt引入

import 'package:flutter\_app/pkg/config.dart';  
import 'pkg/config.dart';

import 'package:http/http.dart' as http;

### as重命名与deferred as 延迟加载

as

import 'pkg/config.dart' as co;

后边使用

String aa=co.sayHello("站械", 18);  
print(aa);  
备注 方法不能直接co.sayHello  
var p = new co.Persons();  
p.haveBaby();

deferred as

import 'pkg/config.dart' deferred as co;

后边使用前 加载

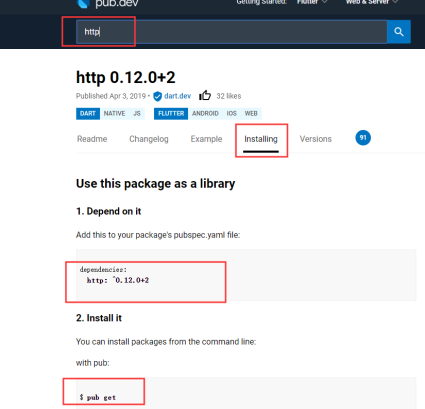
co.loadLibrary();

### 从pub网站上拿依赖包

相当于js的npm，java的maven

网址：<https://pub.dev>

配置文件：pubspec.yaml





安了pub用pub，一般用 flutter pub get

引用：

import 'package:http/http.dart';

使用：



### 调取Dart自身的包

import 'dart:math';

## 异步处理

### 异步函数Futures

Future.delayed(new Duration(seconds: 5),(){  
 print('过5秒后执行该方法');  
});

### async和await

有async 的话，已经申明是异步方法

里边再有await 开头的方法，申明是异步方法里需要同步执行的

void aa() {  
print('11');  
Future.delayed(new Duration(seconds: 5),(){  
 print('过5秒后执行该方法');  
});  
print('12');  
}

输出

11

12

过5秒后执行该方法

void aa() async{  
print('11');  
 await Future.delayed(new Duration(seconds: 5),(){  
 print('过5秒后执行该方法');  
});  
print('12');  
}

11

过5秒后执行该方法

12

### Future.wait

Future.wait([  
Future.delayed(new Duration(seconds: 3),(){ print('11');}),  
Future.delayed(new Duration(seconds: 3),(){ print('22');}),  
Future.delayed(new Duration(seconds: 3),(){ print('33');}),  
]).then((List results){  
 print('执行完了');  
});

## 异常处理

Finally

要确保在抛出异常时运行某些业务代码，请使用finally子句。如果没有catch子句匹配异常，则在finally子句运行后传播异常:

try {

breedMoreLlamas();

} catch (e) {

print('Error: $e'); // Handle the exception first.

} finally {

cleanLlamaStalls(); // Then clean up.

}

## 特殊语法

### ??

String

变量 ?? '变量为null 或者 只定义变量未赋值的时候显示',

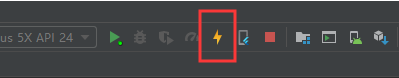
}

'${变量 ?? '--'}'

# 基本方法

## 使用热部署

Ctrl+s 或



## 最简结构

import 'package:flutter/material.dart';  
  
void main() => runApp(MyApp());  
  
class MyApp extends StatelessWidget{  
 Widget build(BuildContext context){  
 return new MaterialApp(  
 title: '整体标题',  
 home: new Scaffold(  
 appBar: AppBar(  
 title: Text('标题'),  
 ),  
 body: Text('内容'),  
 )  
 );  
 }  
}

## 常规结构及分析

每个main.dart分为4部分

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

void main() => runApp(主类()); //启动载入方法

class MyApp extends StatelessWidget {……}

class MyHomePage extends StatefulWidget {……}

class \_MyHomePageState extends State<MyHomePage> {……}

继承StatelessWidget，这个类中主要是设置APP名称，主题，执行方法类MyHomePage(title: 'Flutter Demo Home Page')

class MyApp extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Flutter Demo',  
 theme: ThemeData(  
 primarySwatch: Colors.blue,  
 ),  
 home: MyHomePage(title: 'Flutter Demo Home Page'),  
 );  
 }  
}

继承StatefulWidget ，这个类除title以外，可以固定写法

class MyHomePage extends StatefulWidget {  
 MyHomePage({Key key, this.title}) : super(key: key);  
 final String title;  
  
 @override  
 \_MyHomePageState createState() => \_MyHomePageState();  
}

继承State ，这个类是界面的主要实现类，继承State的关键是为了能使用setState方法来重新渲染页面

class \_MyHomePageState extends State<MyHomePage> {  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text('标题'),  
 ),  
 body: Text('内容'),  
 );  
 }  
}

不执行setState方法，虽然变量的值变了，但是不会渲染到页面上

void \_incrementCounter() {  
 setState(() {  
 \_counter++;  
 });  
}

## 进阶结构及控件封装

在flutter中，一切结构，如Scaffold、MaterialApp、Container、DropdownButton等等均可封装。

实例名为：iStructure.dart

实现类一般就是在类名前边加下划线

以下实例封装了一个MyDropdownButton 的类，并且联动着State类，能够实现数据的实时更新

class MyDropdownButton extends StatefulWidget {  
 Object parms;  
 MyDropdownButton (parms) {  
 this.parms=parms;  
 }  
 @override  
 State<StatefulWidget> createState() {  
 return \_MyDropdownButton(parms);  
 }  
}  
class \_MyDropdownButton extends State<MyDropdownButton> {  
 Object parms;  
 \_MyDropdownButton(parms) {  
 this.parms =parms;  
 }  
 List getCityList() {  
 List<DropdownMenuItem> cityList =new List();  
 cityList.add(DropdownMenuItem(child: Text('昆明'),value: '5301'));  
 cityList.add(DropdownMenuItem(child: Text('玉溪'),value: '5302'));  
 cityList.add(DropdownMenuItem(child: Text('曲靖'),value: '5303'));  
 return cityList;  
 }  
 var selectedCity;  
 @override  
 Widget build(BuildContext context) {  
 return DropdownButton(  
 items: getCityList(),  
 value: selectedCity,  
 hint: Text(this.parms),  
 onChanged: (val){  
 setState(() {  
 this.selectedCity = val;  
 });  
 },  
 );  
 }  
}

}

调用的时候，直接调取这个类就实现了，里边的所有内容，除了标题，其它如列表内容，默认值等等均可写成参数传进去

Container(  
 width: 150,  
 child: MyDropdownButton('fdsfs'),  
)

## 颜色的设置

Colors.red[600],

Colors.red,

Color.fromRGBO(255, 79, 2, 1), //#ff4f02

color: const Color(0xffff4f02))

[0-1]是16进制 0x

[2-3]是不透明度 ff

[4-9]是颜色值

## EdgeInsets

这个单位用的很多，在margin 和 padding 中都有使用

EdgeInsets.all(5)

EdgeInsets.only(top: 20)

## 页面的重新渲染

实现原理：

当实际参数变化之后，得通过State类的setState方法来重新渲染页面

import 'package:flutter/material.dart';  
import 'package:english\_words/english\_words.dart';  
void main() => runApp(new MyHomePage());  
class MyApp2 extends State<MyHomePage> {  
 int \_count = 0;  
 @override  
 Widget build(BuildContext context) {  
 return new MaterialApp(  
 title: 'Welcome to Flutter',  
 home: new Scaffold(  
 appBar: AppBar(  
 title: new Text('标题') ,  
 ),  
 body: Center(  
 child: new Text('你点了$\_count次啊'),  
 ),  
 floatingActionButton: FloatingActionButton(  
 tooltip: '标题',  
 child: new Icon(Icons.menu),  
 onPressed: () => setState(() => \_count++),  
 ),  
 )  
 );  
 }  
}  
class MyHomePage extends StatefulWidget {  
 @override  
 MyApp2 createState() => MyApp2();  
}

## Flutter的生命周期

实现原理：

假设我们从A页面跳转到B页面， 那么A，B页面的生命周期会是怎样的呢？

B页面进入初始化状态，依次执行4个函数：构造函数 > initState > didChangeDependencies > Widget build , 此时页面加载完成,进入运行态。

此时A页面依次执行deactivate > build函数。注意 此时A页面并未卸载。

然后我们假设B页面只有一个按钮，点击B页面中的按钮，改变按钮的文字，会执行widget的build方法 ，(理论上也应该执行didUpdateWidget，但我这里没有)。

这时，我们点击返回键从B页面返回到A页面。

A页面重新显示，B页面开始卸载。

那么A先执行deactivate > build ， 然后B页面依次执行：deactivate > dispose 。

此时A页面进入运行态，B页面移除。

|  |  |  |
| --- | --- | --- |
| **名称** | **状态** | **调用次数** |
| initState | 插入渲染树时调用，只调用一次 | 1 |
| didChangeDependencies | state依赖的对象发生变化时调用 | N |
| didUpdateWidget | 组件状态改变时候调用，可能会调用多次 | N |
| build | 构建Widget时调用 | 1 |
| deactivate | 当移除渲染树的时候调用 | 1 |
| dispose | 组件即将销毁时调用 | 1 |

有这些状态，一般也是在实现类当中State

class NewsDetailState extends State<NewsDetailPage> {

int text = 1;

NewsDetailState() {

print('构造函数');

}

@override

void initState() {

print('init state');

super.initState();

}

@override

void didChangeDependencies() {

print('didChangeDependencies');

super.didChangeDependencies();

}

@override

Widget build(BuildContext context) {

print('widget build');

return Scaffold(

body: Center(

child: \_loading(),

),

appBar: AppBar(

title: Text('咨询详情'),

),

);

}

@override

void didUpdateWidget(NewsDetailPage oldWidget) {

print('组件状态改变：didUpdateWidget');

super.didUpdateWidget(oldWidget);

}

@override

void deactivate() {

print('移除时：deactivate');

super.deactivate();

}

@override

void dispose() {

print('移除时：dispose');

super.dispose();

}

//预加载布局

Widget \_loading() {

return Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

CircularProgressIndicator(

strokeWidth: 1.0,

),

Container(

child: Text("正在加载"),

margin: EdgeInsets.only(top: 10.0),

)

],

);

}

}

## 打开一个新页面

void openPage(BuildContext context) {  
 Navigator.push(context, MaterialPageRoute(  
 builder: (BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Next page'),  
 ),  
 body: const Center(  
 child: Text(  
 'This is the next page',  
 style: TextStyle(fontSize: 24),  
 ),  
 ),  
 );  
 },  
 ));  
}

## 打包，去除Debug

MaterialApp下debugShowCheckedModeBanner设为false

return MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: Indexful(),  
);

# 基础插件

## 文本和字体

相当于html里的label和span

### Text 单一格式的文本

多个Text会自动换行

new Text(  
 'Hello, How are you?',  
 textAlign: TextAlign.center,  
 overflow: TextOverflow.ellipsis,  
 style: TextStyle(fontWeight: FontWeight.bold,color: Colors.indigo,fontSize: 30),  
),

**[textAlign]** TextAlign.center,文本对齐 center Left right end start justify

**[maxLines → int]** 2,文字最多显示几行

**[overflow]** TextOverflow.ellipsis,文字过多显示…… 类似于css的ellipsis

### RichText 一个富文本Text

多个TextSpan不会自动换行

RichText的子内容即为TextSpan

RichText加在Scaffold的body里

换行为 +'\r\n'

body: new RichText(  
 text: TextSpan(  
 text: 'He432342llo ',  
 //style: DefaultTextStyle.of(context).style,  
 style: TextStyle(fontWeight: FontWeight.bold,color: Colors.red),  
 children: <TextSpan>[  
 TextSpan(text: 'bold'+'\r\n', style: TextStyle(fontWeight: FontWeight.bold,color: Colors.blue)),  
 TextSpan(text: ' worldddd!'),  
 ],  
 ),  
)

### DefaultTextStyle 文字样式

const styles=TextStyle() 可以定义，后边引用

## 按钮和按钮组

实例名为：iButton.dart

### RaisedButton

一个凸起的材质矩形按钮

**[[textColor](https://api.flutter.dev/flutter/material/MaterialButton/textColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html) ]** 文本的颜色

**[[shape](https://api.flutter.dev/flutter/material/MaterialButton/shape.html) → [ShapeBorder](https://api.flutter.dev/flutter/painting/ShapeBorder-class.html) ]** 按钮的的形状。shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10.0)),

**[[minWidth](https://api.flutter.dev/flutter/material/MaterialButton/minWidth.html) → [double](https://api.flutter.dev/flutter/dart-core/double-class.html) ]** 按钮将占据的最小水平范围。

**[[key](https://api.flutter.dev/flutter/widgets/Widget/key.html) → [Key](https://api.flutter.dev/flutter/foundation/Key-class.html) ]** 控制一个小部件如何替换树中的另一个小部件

**[[hoverColor](https://api.flutter.dev/flutter/material/MaterialButton/hoverColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html) ]** 当指针悬停在 按钮[材质上](https://api.flutter.dev/flutter/material/Material-class.html)时，其填充颜色。

**[[height](https://api.flutter.dev/flutter/material/MaterialButton/height.html) → [double](https://api.flutter.dev/flutter/dart-core/double-class.html) ]** 高

**[[focusColor](https://api.flutter.dev/flutter/material/MaterialButton/focusColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html) ]** 当 按钮具有输入焦点时，其“ [材质](https://api.flutter.dev/flutter/material/Material-class.html) ”的填充颜色。

**[[enableFeedback](https://api.flutter.dev/flutter/material/MaterialButton/enableFeedback.html) → [bool](https://api.flutter.dev/flutter/dart-core/bool-class.html) ]** 检测到的手势是否应提供声音和/或触觉反馈。

**[[enabled](https://api.flutter.dev/flutter/material/MaterialButton/enabled.html) → [bool](https://api.flutter.dev/flutter/dart-core/bool-class.html) ]** 该按钮是启用还是禁用

**[[disabledTextColor](https://api.flutter.dev/flutter/material/MaterialButton/disabledTextColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html)]** 禁用按钮时用于此按钮文本的颜色。

**[[disabledColor](https://api.flutter.dev/flutter/material/MaterialButton/disabledColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html) ]** 禁用按钮时按钮的填充颜色

**[highlightColor → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html) ]** 高亮的颜色，就是鼠标点上去的颜色

**[[onPressed](https://api.flutter.dev/flutter/material/IconButton/onPressed.html) → [VoidCallback](https://api.flutter.dev/flutter/dart-ui/VoidCallback.html) ]** 轻击或激活按钮时调用的回调。类似于click事件，无长按事件

RaisedButton(  
 onPressed: () {},  
 child: const Text('Enabled Button', style: TextStyle(fontSize: 20)),  
),

### FloatingActionButton

一个圆形图标按钮，它悬停在内容之上，以展示应用程序中的主要动作。

**[floatingActionButton ]** 样子、事件等

floatingActionButton: FloatingActionButton(  
 onPressed: () => setState(() => \_count++),  
 tooltip: 'Increment Counter',  
 child: const Icon(Icons.add),  
),

**[floatingActionButtonLocation ]** 位置 centerFloat（中下） centerDocked （中下） miniStartTop（左上） startTop （左上）endTop （右上）等

floatingActionButtonLocation: FloatingActionButtonLocation.startTop,

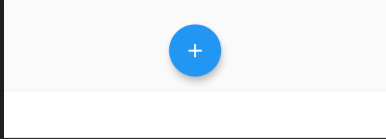
配合bottomNavigationBar，可实现嵌入效果

bottomNavigationBar: BottomAppBar(  
 shape: const CircularNotchedRectangle(),  
 child: Container(height: 50.0,),  
),

centerDocked:



centerFloat:



### FlatButton

一个扁平的Material按钮

这类按钮的api和RaisedButton的一样

FlatButton(  
 color: Color(0xffff4f02),  
 textColor: Colors.white,  
 disabledColor: Colors.grey,  
 disabledTextColor: Colors.black,  
 padding: EdgeInsets.all(8.0),  
 splashColor: Colors.blueAccent,  
 onPressed: () {},  
 child: Text("我是一个按钮"),  
)

### OutlineButton

一个带外边框的Material按钮

这类按钮的api和RaisedButton的一样

在borderSide中设置边框颜色

OutlineButton(  
 onPressed: () {},  
 highlightColor: Colors.red,  
 shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10.0)),  
 borderSide: BorderSide(  
 color: Color(0xffff4f02),  
 ) ,  
 child: Text("我是一个按钮"),  
),

### IconButton(Icon)

一个Material图标按钮，点击时会有水波动画

这类按钮的api和RaisedButton的一样

IconButton(  
 icon: Icon(Icons.volume\_up),  
 tooltip: 'Increase volume by 10',  
 onPressed: () {},  
),

### PopupMenuButton

当菜单隐藏式，点击或调用onSelected时显示一个弹出式菜单列表

先在class外定义一组枚举

enum WhyFarther { list0, list1, list2, list3 }

new PopupMenuButton(

onSelected: (WhyFarther result) { setState(() { print(result); }); },  
 itemBuilder: (BuildContext context) =>  
 <PopupMenuEntry<WhyFarther>>[  
 const PopupMenuItem<WhyFarther>(  
 value: WhyFarther.list0,  
 child: Text('我是第一项'),  
 ),  
 const PopupMenuItem<WhyFarther>(  
 value: WhyFarther.list1,  
 child: IconButton(  
 icon: Icon(Icons.menu),  
 ),  
 ),  
 ])  
],

这个方法不完善，要页面会渲染 加上 SetState

### ButtonBar //todo

水平排列的按钮组

## 输入框和选择框

实例名为：iInput.dart

### TextField+键盘

decoration方式控制了大部分样式，详细API：

<https://api.flutter.dev/flutter/material/InputDecoration-class.html>

TextField(  
 obscureText: false, // 是否隐藏文本，例如密码类型就需要隐藏  
 controller: \_controller,  
 autofocus: true,  
 cursorColor: Color(0xffff4f02), // 光标颜色  
 enabled: true, // 不启用  
 minLines: 3,

maxLines: 4, // 这两个要一起用，不然会报错  
 maxLength:10, //文本字段中允许的最大字符数 且会有个小计数器下右下脚  
 textAlign: TextAlign.left, // 文本应如何水平对齐  
 enableInteractiveSelection: true, //默认true 如果为false，则不会长按此TextField将选择文本并显示剪切/复制/粘贴菜单，而点击将移动文本插入符号  
 style: TextStyle(  
 fontSize: 22.0, // 文字大小  
 color: Colors.indigo, // 文字的颜色  
// backgroundColor: Colors.amber, // 文字背景 一般不用  
 ),  
 decoration:InputDecoration(  
 filled: true, // 这个设为true 前景色才生效  
 hintText: 'Username', // 选中这个框，且这个框里无任何文字时显示的内容  
 prefixIcon: Icon(Icons.search),  
 hasFloatingPlaceholder: true,  
 fillColor: Color(0xfff0f0f0),  
 counterText: 'sssss',  
 labelText: '用户22名', // 边框上的中文  
 labelStyle: TextStyle(  
 color: Colors.indigo,  
 ),  
 border: OutlineInputBorder(  
 borderRadius: BorderRadius.circular(15),  
 borderSide: BorderSide(color: Color(0xffff4f02))  
 ),  
 focusedBorder: OutlineInputBorder( //选中的边框颜色及圆角  
 borderSide: BorderSide(color: Colors.cyanAccent),  
 borderRadius: BorderRadius.circular(5.7),  
 ),  
 enabledBorder: UnderlineInputBorder( //默认的边框颜色及圆角  
 borderSide: BorderSide(color: Colors.teal),  
 borderRadius: BorderRadius.circular(25.7),  
 ),  
 ),  
  
),  
TextField(  
 obscureText: false,  
 autofocus: true,  
 decoration:InputDecoration(  
 border: OutlineInputBorder(),  
 labelText: '密码', // 边框上的中文  
 ),  
  
)

键盘放在TextField里的第一层子集

keyboardType: TextInputType.text,

TextField(

inputFormatters: [

//只允许输入字母

WhitelistingTextInputFormatter(RegExp("[a-zA-Z]")),

],

),

TextField(

//只允许输入数字

inputFormatters: [WhitelistingTextInputFormatter.digitsOnly],

),

TextField(

inputFormatters: [

//只允许输入小数

WhitelistingTextInputFormatter(RegExp("[0-9.]")),

],

）

,

TextField(

inputFormatters: [

//限制小数位数

\_MyNumberTextInputFormatter(digit:5),

],

）

// 限制小数位数

class \_MyNumberTextInputFormatter extends TextInputFormatter {

static const defaultDouble = 0.001;

///允许的小数位数，-1代表不限制位数

int digit;

\_MyNumberTextInputFormatter({this.digit=-1});

static double strToFloat(String str, [double defaultValue = defaultDouble]) {

try {

return double.parse(str);

} catch (e) {

return defaultValue;

}

}

///获取目前的小数位数

static int getValueDigit(String value){

if(value.contains(".")){

return value.split(".")[1].length;

}else{

return -1;

}

}

@override

TextEditingValue formatEditUpdate(TextEditingValue oldValue, TextEditingValue newValue) {

String value = newValue.text;

int selectionIndex = newValue.selection.end;

if (value == ".") {

value = "0.";

selectionIndex++;

} else if (value != ""

&& value != defaultDouble.toString()

&& strToFloat(value, defaultDouble) == defaultDouble

||getValueDigit(value)>digit) {

value = oldValue.text;

selectionIndex = oldValue.selection.end;

}

return new TextEditingValue(

text: value,

selection: new TextSelection.collapsed(offset: selectionIndex),

);

}

}

### Form+TextFormField

实例名为：iForm.dart

Form里的TextField叫 TextFormField 写在Form中最大的好处就是有数据效验，通过validator实现

class iForm extends StatelessWidget {  
 GlobalKey \_formKey = new GlobalKey<FormState>();  
 TextEditingController \_username = new TextEditingController();  
 TextEditingController \_password = new TextEditingController();  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'fds',  
 home: Scaffold(  
 appBar: AppBar(  
 title: Text('fs'),  
 ),  
 body: Column(  
 children: <Widget>[  
 Form(  
 key: \_formKey,  
 autovalidate: true,  
 child: Column(  
 children: <Widget>[  
 TextFormField( //在Form里使用的 TextField 叫 TextFormField  
 decoration: InputDecoration(  
 labelText: "用户名",  
 hintText: "请输入用户名",  
 icon: Icon(Icons.access\_alarms),  
 ),  
 controller: \_username,  
 validator: (val){ // 针对当前输入框的效验  
 return val.trim().length>0 ? null : "请输入用户名";  
 },  
 ),  
 TextFormField( //在Form里使用的 TextField 叫 TextFormField  
 controller: \_password,  
 decoration: InputDecoration(  
 labelText: "密码",  
 hintText: "请输入密码",  
 icon: Icon(Icons.access\_alarms),  
 ),  
 validator: (val){ // 针对当前输入框的效验  
 return val.trim().length>0 ? null : "请输入密码2"; //提示文字  
 },  
 ),  
 RaisedButton(  
 child: Text('提交'),  
 onPressed: (){  
 if((\_formKey.currentState as FormState).validate()){  
 print("提交数据给后台"+ \_username.text+ \_password.text);  
 }  
 },  
 )  
 ],  
 ),  
 )  
 ],  
  
 )  
 )  
  
 );  
 }  
}

### Checkbox

Checkbox 靠value 控制选中或者未选中，改变状态需要写onChanged，没有组的概念

常用属性

**[activeColor → Color]** 激活时的颜色

**[onChanged → ValueChanged]** 改变时触发

**[tristate → bool]** 如果为 true，那么复选框的值可以是 true，false 或 null

**[value → bool]** 复选框的值

Checkbox(  
 value: check,  
 onChanged: (bool value) {  
 setState(() {  
 check = !check;  
 });  
 },  
),

### CheckboxListTile （一点就整行选中）

CheckboxListTile(  
 secondary: const Icon(Icons.menu),  
 title: const Text('硬件加速'),  
 value: check,  
 onChanged: (bool value) {  
 setState(() {  
 check = !check;  
 });  
 },  
),

### Radio（手机上一般用Switch,略）

### Switch

**[activeColor → Color]** 激活时原点的颜色

**[activeThumbImage → ImageProvider ]** 原点还支持图片，激活时的效果。

**[activeTrackColor → Color]** 如激活时横条的颜色

**[inactiveThumbColor → Color]** 非激活时原点的颜色。

**[inactiveThumbImage → ImageProvider]** 非激活原点的图片效果

**[inactiveTrackColor → Color]** 非激活时横条的颜色

**[onChanged → ValueChanged]** 改变时触发

**[value → bool]** 切换按钮的值

new Switch(  
 value: check,  
 activeColor: Colors.blue, // 激活时原点颜色  
 onChanged: (bool val) {  
 this.setState(() {  
 check = !check;  
 });  
 },  
)

### Slider

**[activeColor → Color]** 激活时的颜色

**[divisions → int ]** 离散部分的数量

**[inactiveColor → Color]** 滑块轨道的非活动部分的颜色

**[label → String]** 滑块处于活动状态时显示在滑块上方的标签

**[max → double]** 用户可以选择的最大值

**[min → double]** 用户可以选择的最小值

**[onChanged → ValueChanged]** 改变时触发

**[onChangeEnd → ValueChanged]** 改变后触发

**[onChangeStart → ValueChanged]** 改变前触发

**[value → double]** 滑块的值

new Slider(  
 value: slider,  
 max: 100.0,  
 min: 0.0,  
 activeColor: Colors.blue,  
 onChanged: (double val) {  
 this.setState(() {  
 slider = val;  
 });  
 },  
),

### Date & Time Pickers

实例名为：iDate.dart

在State类中，创建一个时间和一个方法

DateTime \_dateTime = DateTime.now();  
 void showDefaultYearPicker(BuildContext context) async {  
 final DateTime dateTime = await showDatePicker(  
 context: context,  
 //定义控件打开时默认选择日期  
 initialDate: this.\_dateTime,  
 //定义控件最早可以选择的日期  
 firstDate: DateTime(2018, 1),  
 //定义控件最晚可以选择的日期  
 lastDate: DateTime(2022, 1),  
 locale: Locale('zh'),  
 builder: (BuildContext context, Widget child) {  
 return child;  
 },  
 );  
 if (dateTime != null && dateTime != \_dateTime) {  
  
  
 setState(() {  
 this.\_dateTime=dateTime;  
  
// Duration duration = dateTime.difference(now);  
// print(duration.inDays.toString());  
  
 });  
  
 }  
 }

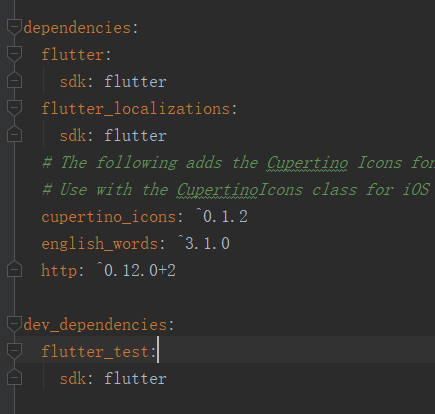
,

在界面中执行那个方法

RaisedButton(  
 child: Text(''+this.\_dateTime.toString()),  
 onPressed: (){  
 this.showDefaultYearPicker(context);  
 },  
)

原版是英文的，还需要实现中文

1. pubspec.yaml中安装flutter\_localizations，注意缩进，Packages get

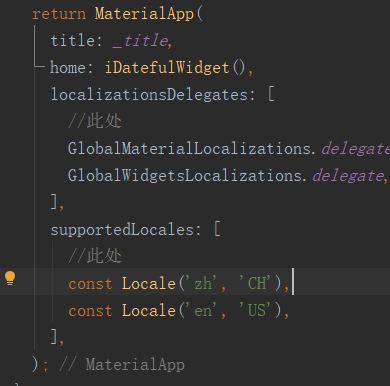


flutter\_localizations:  
 sdk: flutter

1. 在页面中引入

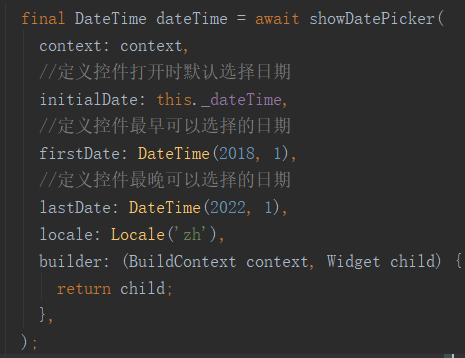
import 'package:flutter\_localizations/flutter\_localizations.dart';

1. 在MaterialApp实例中加入



localizationsDelegates: [  
 //此处  
 GlobalMaterialLocalizations.delegate,  
 GlobalWidgetsLocalizations.delegate,  
],  
supportedLocales: [  
 //此处  
 const Locale('zh', 'CH'),  
 const Locale('en', 'US'),  
],

1. 在showDatePicker方法中使用中文 locale: Locale('zh'),



## 容器集

### Padding

margin和 padding 都是 EdgeInsets. 方法来实现值

Padding(  
//padding: EdgeInsets.all(50),  
 padding: EdgeInsets.fromLTRB(10, 20, 30, 40),  
 child: Container(  
 color: Colors.blue,  
 ),  
),

### 限制类容器

这类容器只有两个SizedBox 和 ConstrainedBox

SizedBox 只可以设置子容器具体的宽高

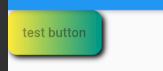
ConstrainedBox 只可以设置子容器的宽高的范围

SizedBox(  
 width: 150,  
 height: 150,  
 child: Container(  
 color: Colors.teal,  
 ),  
),

ConstrainedBox(  
 //填满剩余空间  
// constraints: BoxConstraints.expand(),  
 // 具体的范围  
// constraints: BoxConstraints(  
// minWidth: 100,  
// maxWidth: 200,  
// minHeight: 100,  
// maxHeight: 200  
// ),  
 // 某一个方向填满  
 constraints: BoxConstraints(  
 minWidth: double.infinity, // 横向填满  
 minHeight: 100,  
 maxHeight: 200  
 ),  
 child: Container(color: Colors.lightGreen,),  
 ),

### 装饰类容器

实例名为：iDecoratedBox.dart



DecoratedBox容器相当于装饰child,在child的背景上做各类效果

DecoratedBox(  
 decoration: BoxDecoration(  
 borderRadius: BorderRadius.all(Radius.circular(10.0)), // 圆角  
 gradient: LinearGradient(colors: [Colors.yellow,Colors.teal]), // 渐变色  
 boxShadow: [ //投影  
 BoxShadow(  
 color: Colors.black,  
 offset: Offset(3,3),  
 blurRadius: 4.0  
 )  
 ],  
 ),  
 child: FlatButton(  
 onPressed: null,  
 child: Text('test button'),  
 ),  
),

### 变换transform

实例名为：iTransform.dart

通过Transform可实现 拉伸 放大 旋转 平移

DecoratedBox(  
 decoration: BoxDecoration(  
 borderRadius: BorderRadius.all(Radius.circular(10.0)), // 圆角  
 gradient: LinearGradient(colors: [Colors.yellow,Colors.teal]), // 渐变色  
 boxShadow: [ //投影  
 BoxShadow(  
 color: Colors.black,  
 offset: Offset(3,3),  
 blurRadius: 4.0  
 )  
 ],  
 ),  
 child: FlatButton(  
 onPressed: null,  
 child: Text('test button'),  
 ),  
),

## 布局与列表

### Container

相当于class为Container的div

Center(

child: Container(

margin: const EdgeInsets.all(10.0),

color: Colors.amber[600],

width: 48.0,

height: 48.0,

),

)

**[margin]** const EdgeInsets.all(10.0), 外边距为10

**[padding]** const EdgeInsets.all(10.0), 内边距为10

**[color]** Colors.red, 色后边可以加[]，值越大颜色越深

**[width]** 内一个整数或者小数，屏幕最宽可能为410

**[height]** 内一个整数或者小数，屏幕最高可能为600

**[transform]** Matrix4.rotationZ(0.1),屏幕最高可能为600

**[alignment]** Alignment.center,内容对齐 center topLeft centerLeft bottomLeft ……

**[child][Text]** child: Text('Hello World'),Text不能直接用于Container

### 线性布局Row和Column

**[width]** 内一个整数或者小数，屏幕最宽可能为410

**[height]** 内一个整数或者小数，屏幕最高可能为600

**[color]** Colors.red, 色后边可以加[]，值越大颜色越深

**[crossAxisAlignment]** CrossAxisAlignment.start 对齐方式 因为ROw和Column都适用，所有值为： start 上边 center 默认，中间 end 底部

### 弹性布局Flex

实例名为：iFlex.dart

弹性布局的标签为Flex,Flex有子属性direction这个属性来决定方向

**[direction]** Axis.horizontal 方向 Axis.horizontal横向 Axis.vertical竖向

Flex里边有Expanded，这个决定参与等分的子元素，Container不参与等分

有Expanded标签，里边就必有flex属性

Flex(  
 direction: Axis.horizontal, //方向 Axis.horizontal横向 Axis.vertical竖向  
 children: <Widget>[  
 Container( // Container不参数等分，剩余宽度按比例分配  
 width: 50,  
 height: 300,  
 color: Colors.black,  
 ),  
 Expanded( //Flex下的Expanded必定参与等分，有Expanded标签，里边就必有flex属性  
 flex: 1,  
 child: Container(  
 height: 300,  
 color: Colors.red,  
 )  
 ),  
 Expanded(  
 flex: 1, //  
 child: Container(  
 height: 300,  
 color: Colors.blue,  
 )  
 ),  
 Expanded(  
 flex: 1,  
 child: Container(  
 height: 300,  
 color: Colors.yellow,  
 )  
 )  
 ],  
),

### 定位Stack

实例名为：iStack.dart

Stack无法直接拥有宽高，一般通过SizedBox修饰子组件的模式来控制Stack

SizedBox( //Stack无法直接拥有宽高，一般通过SizedBox修饰子组件的模式来控制Stack  
 width:400,  
 height: 400,  
 child: Stack(  
 alignment: Alignment.topRight, // 对齐方式，如果子元素没设left这些，可以通过这个来统一设定  
 children: <Widget>[  
 Positioned(  
 left:15, //同时有左右， 宽度不生效  
 top: 30,  
 child: Container(  
 width: 50,  
 height: 50,  
 color: Colors.red,  
 ),  
 ),  
 Positioned(  
 right:15, //同时有左右， 宽度不生效  
 bottom: 30,  
 child: Container(  
 width: 350,  
 height: 350,  
 color: Colors.blue,  
 ),  
 )  
 ],  
 ),  
),

### 流式布局Wrap

实例名为：iWrap.dart

流式布局可以理解为横向的，超过手机宽度自动换行的布局

**[spacing → int ]** 水平方向间距

**[runSpacing → int ]** 垂直方向间距

**[alignment]** WrapAlignment.center, // 中间对齐

Wrap(  
 spacing: 25, // 水平方向间距  
 runSpacing: 10, //垂直方向间距  
 alignment: WrapAlignment.center, // 中间对齐  
 children: <Widget>[  
 Container(  
 width: 150,  
 height: 100,  
 color: Colors.red,  
 ),  
 Container(  
 width: 150,  
 height: 100,  
 color: Colors.blue,  
 ),  
 Container(  
 width: 150,  
 height: 100,  
 color: Colors.yellow,  
 ),  
 ],  
),

### ListView

实例名为：和iListView.dart

ListView 里边可以套 ListTile Container Row 等，也可以套ListView

但是 ListView 本身 不能设高度， 这个情况下可以先套一层 Container ，设了高度，就可以使用ListView 了

一般情况下，是在ListView 中加入 Container 里边再加入ListView.builder

不过个人感觉，直接使用ListView 的子元素 ListTile 表头 效果也很好

ListView(  
 physics: const AlwaysScrollableScrollPhysics(),  
 padding: new EdgeInsets.all(5.0),  
 children: <Widget>[  
 new ListTile(  
 leading: new Icon(Icons.info\_outline),  
 title: Text("关于"),  
 trailing: new Icon(Icons.chevron\_right),  
 onTap: () {  
 print('关于');  
 },  
 ),

ListView的子内容里

Container(  
 height: 400,  
 child: ListView.builder(  
 itemCount: 20,  
 itemExtent: 50,  
 itemBuilder: (BuildContext context, int index ) {  
 return Text('列表内容$index');  
 }),  
),

### ListView之controller

定义

ScrollController \_scrollController = new ScrollController();

销毁

@override  
void dispose() {  
 super.dispose();  
 \_scrollController.dispose();  
}

添加监听

@override  
void initState() {  
 super.initState();  
 \_scrollController = new ScrollController();  
 // 监听滚动  
 \_scrollController.addListener(() {

// 这个值为滚动到的位置  
 var tempShowTop = \_scrollController.offset;   
 });  
}

在ListView中加controller

new ListView(  
 scrollDirection: Axis.vertical,  
 controller: \_scrollController,

)

跳转到底部，无动画

setState(() {  
 \_scrollController.jumpTo(\_scrollController.position.maxScrollExtent);  
});

跳转到指定位置，有动画，时间为1s

setState(() {  
 \_scrollController.animateTo(500.0,  
 duration: new Duration(seconds: 1), curve: Curves.ease);

});

### Scrollbar

适用范围：一般是只有一个元素，而且还特别长

Scrollbar(  
 child: SingleChildScrollView(  
 child: Container(  
 height:3000,  
 color: Colors.blueGrey,  
 ),  
 ),  
)

### CustomScrollView

实例名为：iScrollView.dart

new CustomScrollView(  
 shrinkWrap: true,  
 slivers: <Widget>[  
 new SliverPadding(  
 padding: const EdgeInsets.all(20.0),  
 sliver: new SliverList(  
 delegate: new SliverChildListDelegate(  
 <Widget>[  
 const Text('A'),  
 const Text('B'),  
 const Text('C'),  
 const Text('D'),  
 ],  
 ),  
 ),  
 ),  
 ],  
)

### GridView

实例名为：iGirdView.dart

GridView.count

GridView.count(  
 primary: false,  
 padding: const EdgeInsets.all(20), //GridView内边距  
 crossAxisSpacing: 10, //水平子Widget之间间距  
 mainAxisSpacing: 10, //垂直子Widget之间间距  
 crossAxisCount: 4, // //一行的Widget数量  
 childAspectRatio: 2.0, //子Widget宽高比例  
 children: <Widget>[ //子Widget列表  
 Container(  
 padding: const EdgeInsets.all(8),  
 child: const Text('He\'d have you all unravel at the'),  
 color: Colors.teal[100],  
 ),  
 Container(  
 padding: const EdgeInsets.all(8),  
 child: const Text('Heed not the rabble'),  
 color: Colors.teal[200],  
 ),  
 Container(  
 padding: const EdgeInsets.all(8),  
 child: const Text('Sound of screams but the'),  
 color: Colors.teal[300],  
 ),  
 Container(  
 padding: const EdgeInsets.all(8),  
 child: const Text('Who scream'),  
 color: Colors.teal[400],  
 ),  
 Container(  
 padding: const EdgeInsets.all(8),  
 child: const Text('Revolution is coming...'),  
 color: Colors.teal[500],  
 ),  
 ],  
)

还有方法

GridView.custom

GridView.builder

### RefreshIndicator（下拉刷新）

实例名为：iRefreshIndicator.dart

### SafeArea（安全区域）

使用SafeArea能很好的解决刘海，不规则屏幕的显示问题

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(widget.title),

),

body: SafeArea( // 加入SafeArea

child: ListView.builder(itemBuilder: (context, index) {

return Padding(

padding: EdgeInsets.only(left: 10, bottom: 18),

child: Text(

'Data',

style: TextStyle(fontSize: 18),

),

);

}),

),

);

}

### Stepper（步骤条）

new Stepper(

currentStep: 0, // <-- 激活的下标

steps: <Step>[

new Step(

title: new Text('第一步'),

content: new Text('第一步内容'),

state: StepState.complete,

isActive: true,

subtitle: new Text('第一步小标题'),

),

new Step(

title: new Text('第二步'),

content: new Text('第二步内容'),

),

new Step(

title: new Text('第三步'),

content: new Text('第三步内容'),

),

],)



Stepper 有以下常用属性：

currentStep → int - 步骤位置。

onStepCancel → VoidCallback - 点击 cancel 时触发。

onStepContinue → VoidCallback - 点击 continue 时触发。

onStepTapped → ValueChanged - 状态改变时触发。

steps → List - 步骤内容。

type → StepperType - 摆放方向，默认是垂直的，可以设置为水平的。

### List转为列表在页面上显示的几种方法

定义一个List

ListView运行时, 报错Vertical viewport was given unbounded height

ListView(

shrinkWrap: true,

children...

)

方法一

在 children: <Widget>[], 的组件中 把<Widget>[]替换为以下代码

listData.map((\_item) =>  
 // 展示的组件  
 ).toList(),

)

方法二

在Widget build(BuildContext context) 新建变量并赋值

List<Widget> list = listData.map((\_item) =>  
 // 展示的组件  
).toList();

在 children: <Widget>[], 的组件中 把<Widget>[]替换为以下代码

附加： 获取索引值，用到asMap，但是这种方法后边只有索引值了，不推荐

listData.asMap().keys.map((item)=>Container(  
 child:Text("${'索引值'+item.toString()}"),  
)).toList()

方法三 推荐

ListView.builder

这种方法自带index

new ListView.builder(  
 shrinkWrap: true,  
 physics: NeverScrollableScrollPhysics(), // 当外层有滚动插件，这里要禁止滚动  
 itemCount: listData.length,  
 itemBuilder: (BuildContext context,int index){  
 var \_item = listData[index];  
 return Container(  
 height: 80,  
 child: Text('${\_item['name']} $index'),  
 );  
 },  
),

## 结构和导航MaterialApp

一切结构的父结构

import 'package:flutter/material.dart';  
  
void main() => runApp(MyApp());  
  
class MyApp extends StatelessWidget{  
 Widget build(BuildContext context){  
 return new MaterialApp(  
 title: '整体标题',  
 home: new Scaffold(  
 appBar: AppBar(  
 title: Text('标题'),  
 ),  
 body: Text('内容'),  
 )  
 );  
 }  
}

### Scaffold

**[[appBar](https://api.flutter.dev/flutter/material/Scaffold/appBar.html) → [PreferredSizeWidget](https://api.flutter.dev/flutter/widgets/PreferredSizeWidget-class.html)]** 顶部的应用栏

**[[backgroundColor](https://api.flutter.dev/flutter/material/Scaffold/backgroundColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html)]** 背景色

**[[body](https://api.flutter.dev/flutter/material/Scaffold/body.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html) ]** 主体

**[[bottomNavigationBar](https://api.flutter.dev/flutter/material/Scaffold/bottomNavigationBar.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)]** 底部导航栏

**[[bottomSheet](https://api.flutter.dev/flutter/material/Scaffold/bottomSheet.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)]** 要显示的永久性底部工作表

**[[drawer](https://api.flutter.dev/flutter/material/Scaffold/drawer.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)]** 显示在[身体](https://api.flutter.dev/flutter/material/Scaffold/body.html)侧面的面板，通常隐藏在移动设备上。从左向右（[TextDirection.ltr](https://api.flutter.dev/flutter/dart-ui/TextDirection-class.html)）或从右向左（[TextDirection.rtl](https://api.flutter.dev/flutter/dart-ui/TextDirection-class.html)）[刷入](https://api.flutter.dev/flutter/dart-ui/TextDirection-class.html)[[](https://api.flutter.dev/flutter/material/Scaffold/drawer.html)

**[[drawerDragStartBehavior](https://api.flutter.dev/flutter/material/Scaffold/drawerDragStartBehavior.html) → [DragStartBehavior](https://api.flutter.dev/flutter/gestures/DragStartBehavior-class.html)]** 确定处理拖动开始行为的方式

**[[drawerEdgeDragWidth](https://api.flutter.dev/flutter/material/Scaffold/drawerEdgeDragWidth.html) → [double](https://api.flutter.dev/flutter/dart-core/double-class.html)]** 确定处理拖动开始行为的方式

**[[drawerScrimColor](https://api.flutter.dev/flutter/material/Scaffold/drawerScrimColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html)]** 打开抽屉时遮盖主要内容的稀松布使用的颜色

**[[endDrawer](https://api.flutter.dev/flutter/material/Scaffold/endDrawer.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)]** 显示在[身体](https://api.flutter.dev/flutter/material/Scaffold/body.html)侧面的面板，通常隐藏在移动设备上。从右向左（[TextDirection.ltr](https://api.flutter.dev/flutter/dart-ui/TextDirection-class.html)）或从左向右（[TextDirection.rtl](https://api.flutter.dev/flutter/dart-ui/TextDirection-class.html)）[轻扫](https://api.flutter.dev/flutter/dart-ui/TextDirection-class.html)

**[[extendBody](https://api.flutter.dev/flutter/material/Scaffold/extendBody.html) → [bool](https://api.flutter.dev/flutter/dart-core/bool-class.html)]** 如果为true，并且 指定了[bottomNavigationBar](https://api.flutter.dev/flutter/material/Scaffold/bottomNavigationBar.html)或[persistentFooterButtons](https://api.flutter.dev/flutter/material/Scaffold/persistentFooterButtons.html)，则[主体](https://api.flutter.dev/flutter/material/Scaffold/body.html)将扩展到脚手架的底部，而不是仅扩展到[bottomNavigationBar](https://api.flutter.dev/flutter/material/Scaffold/bottomNavigationBar.html) 或[persistentFooterButtons](https://api.flutter.dev/flutter/material/Scaffold/persistentFooterButtons.html)的顶部

**[[floatingActionButton](https://api.flutter.dev/flutter/material/Scaffold/floatingActionButton.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)]** 浮动按钮

**[[floatingActionButtonAnimator](https://api.flutter.dev/flutter/material/Scaffold/floatingActionButtonAnimator.html) → [FloatingActionButtonAnimator](https://api.flutter.dev/flutter/material/FloatingActionButtonAnimator-class.html)]** 动画的移动

**[[floatingActionButtonLocation](https://api.flutter.dev/flutter/material/Scaffold/floatingActionButtonLocation.html) → [FloatingActionButtonLocation](https://api.flutter.dev/flutter/material/FloatingActionButtonLocation-class.html)]** 浮动按钮的位置

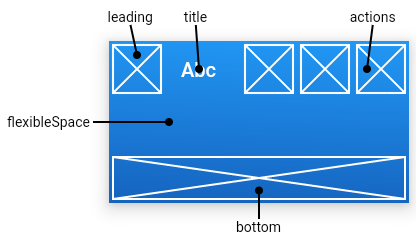
**[[persistentFooterButtons](https://api.flutter.dev/flutter/material/Scaffold/persistentFooterButtons.html) → [List](https://api.flutter.dev/flutter/dart-core/List-class.html)<[Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)>]** 一组按钮显示在支架的底部

**[[primary](https://api.flutter.dev/flutter/material/Scaffold/primary.html) → [bool](https://api.flutter.dev/flutter/dart-core/bool-class.html)]** 此支架是否显示在屏幕顶部

**[[resizeToAvoidBottomInset](https://api.flutter.dev/flutter/material/Scaffold/resizeToAvoidBottomInset.html) → [bool](https://api.flutter.dev/flutter/dart-core/bool-class.html)]** 如果为true，则[主体](https://api.flutter.dev/flutter/material/Scaffold/body.html)和脚手架的浮动小部件应自行调整大小，以避免屏幕键盘的高度由环境[MediaQuery](https://api.flutter.dev/flutter/widgets/MediaQuery-class.html)的[MediaQueryData.viewInsets](https://api.flutter.dev/flutter/widgets/MediaQueryData/viewInsets.html) bottom属性定义

### Appbar

页面的头部



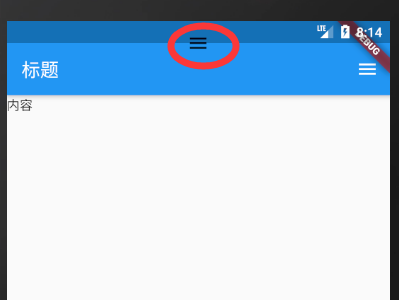
这里的bottom也是头部的bottom

appBar: new AppBar(  
 centerTitle: false, // 默认 false true 为屏幕正中间 false 为除去图标之后的正中间  
 backgroundColor: Color.fromRGBO(255, 79, 2, 1), // 背景色  
 primary: true, //此应用栏是否显示在屏幕顶部 默认true true不在屏幕最顶部，false 和时间、电量、信号那些在一排  
 title: new Text('Welcome to Flut3232ter'),  
 leading: IconButton(  
 icon: const Icon(Icons.menu),  
 tooltip: 'Next page',  
 onPressed: () {  
 },  
 ),  
 actions:<Widget>[  
 IconButton(  
 icon: const Icon(Icons.add\_alert),  
 tooltip: 'Show Snackbar',  
 onPressed: () {  
  
 },  
 ),  
 IconButton(  
 icon: const Icon(Icons.navigate\_next),  
 tooltip: 'Next page',  
 onPressed: () {  
 },  
 ),  
 ] ,  
),

**[[actions](https://api.flutter.dev/flutter/material/AppBar/actions.html) → [List](https://api.flutter.dev/flutter/dart-core/List-class.html)<[Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)>]** 右上角的位置，此处为list

**[[leading](https://api.flutter.dev/flutter/material/AppBar/leading.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)]** 该左上角的位置

**[[flexibleSpace](https://api.flutter.dev/flutter/material/AppBar/flexibleSpace.html) → [Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html)]** 该小部件堆叠在工具栏和选项卡栏的后面。它的高度将与应用栏的整体高度相同。



**[[bottom](https://api.flutter.dev/flutter/material/AppBar/bottom.html) → [PreferredSizeWidget](https://api.flutter.dev/flutter/widgets/PreferredSizeWidget-class.html)]** 该小部件显示在应用程序栏的底部

**[centerTitle]**true为屏幕正中间 false为除去图标之后的正中间, 默认 false

**[primary]** 此应用栏是否显示在屏幕顶部 true:不在屏幕最顶部，false:和时间、电量、信号那些在一排，默认true

**[backgroundColor]** Color.fromRGBO(255, 79, 2, 1)， 背景色

**[title]** new Text('Welcome to Flut3232ter')， 该页的标题

### BottomNavigationBar

实例名为：iBottomNavigationBar.dart

实现滑动效果需要跟PageView结合

**[[backgroundColor](https://api.flutter.dev/flutter/material/BottomNavigationBar/backgroundColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html)]** 背景颜色

**[[currentIndex](https://api.flutter.dev/flutter/material/BottomNavigationBar/currentIndex.html) → [int](https://api.flutter.dev/flutter/dart-core/int-class.html)]** 默认选中第几项

**[[selectedFontSize](https://api.flutter.dev/flutter/material/BottomNavigationBar/selectedFontSize.html) → [double](https://api.flutter.dev/flutter/dart-core/double-class.html)]** 选中标签时 的字体大小

**[[selectedItemColor](https://api.flutter.dev/flutter/material/BottomNavigationBar/selectedItemColor.html) → [Color](https://api.flutter.dev/flutter/dart-ui/Color-class.html)]** 选中标签时 的颜色

**[[showUnselectedLabels](https://api.flutter.dev/flutter/material/BottomNavigationBar/showUnselectedLabels.html) → [bool](https://api.flutter.dev/flutter/dart-core/bool-class.html)]** 显示未选中项的文字 在多于3个的时候，这个值会默认变成false

onTap为单击方法，方法中会自动带一个int参数

import 'package:flutter/material.dart';  
  
  
class iBottomNavigationBar extends StatelessWidget {  
 static const String \_title = 'Flutter Code Sample';  
  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: \_title,  
 home: iBottomNavigationBarfulWidget(),  
 );  
 }  
}  
  
class iBottomNavigationBarfulWidget extends StatefulWidget {  
 iBottomNavigationBarfulWidget({Key key}) : super(key: key);  
  
 @override  
 \_iBottomNavigationBarfulWidgetState createState() => \_iBottomNavigationBarfulWidgetState();  
}  
  
class \_iBottomNavigationBarfulWidgetState extends State<iBottomNavigationBarfulWidget> {  
 int \_selectedIndex = 1;  
 static const TextStyle optionStyle =  
 TextStyle(fontSize: 30, fontWeight: FontWeight.bold);  
 static const List<Widget> \_widgetOptions = <Widget>[  
 Text(  
 'Index 0: Home',  
 style: optionStyle,  
 ),  
 Text(  
 'Index 1: Business',  
 style: optionStyle,  
 ),  
 Text(  
 'Index 2: School',  
 style: optionStyle,  
 ),  
 Text(  
 'Index 2: School',  
 style: optionStyle,  
 ),  
 ];  
  
 void \_onItemTapped(int index) {  
 print(index);  
 setState(() {  
 \_selectedIndex = index;  
 });  
 }  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('BottomNavigationBar Sample'),  
 ),  
 body: Center(  
 child: \_widgetOptions.elementAt(\_selectedIndex),  
 ),  
 bottomNavigationBar: BottomNavigationBar(  
 items: const <BottomNavigationBarItem>[  
 BottomNavigationBarItem(  
 icon: Icon(Icons.home),  
 title: Text('Home'),  
 ),  
 BottomNavigationBarItem(  
 icon: Icon(Icons.business),  
 title: Text('Business'),  
 ),  
 BottomNavigationBarItem(  
 icon: Icon(Icons.school),  
 title: Text('School'),  
 ),  
 BottomNavigationBarItem(  
 icon: Icon(Icons.school),  
 title: Text('School'),  
 ),  
 ],  
 backgroundColor: Color(0xffcccccc),  
 currentIndex: \_selectedIndex, // 默认选中哪个  
 selectedItemColor: Color(0xffff4f02),  
// fixedColor: Colors.deepPurple,

type: BottomNavigationBarType.fixed, //每行一样大，默认不是这种  
 onTap: \_onItemTapped,  
 selectedFontSize: 30, //选中文字的大小  
 unselectedItemColor: Colors.teal , // 未选中的项的颜色  
 showSelectedLabels: true, //显示选中项的文字  
 showUnselectedLabels: true, //显示未选中项的文字 在多于3个的时候，这个值会默认变成false  
  
 ),  
 );  
 }  
}

新建页面，链接到指定的页面

用StatelessWidget 类继承了之后新建4个页面，并引入，

在实现类中，（这里有数据的变化，必定要有实现类State），新建List

List<StatelessWidget> pageList =[  
 PageA(),  
 PageB(),  
 PageC(),  
 PageD()  
];

后边在body位置使用即可

body: this.pageList[\_selectedIndex],

### TabBar 和TabBarView

实例名为：iTabBar.dart

TabBar 不只可以在AppBar中使用

isScrollable: true, //是否可滚动  
indicatorColor: Colors.red, // 选中的那个横条的颜色  
indicatorWeight: 10, // 选中的那个横条的高  
indicatorPadding: const EdgeInsets.all(2), // 底部指示器的Padding  
indicatorSize: TabBarIndicatorSize.tab, //选中那个横条的宽是随文字还是随图片  
labelColor: Colors.deepPurple, // 文字的颜色 当设了 unselectedLabelColor,就是选中的颜色  
labelStyle: TextStyle(fontSize: 18), // 文字的样式  
unselectedLabelColor: Colors.teal, //未选中的颜色  
unselectedLabelStyle: TextStyle(fontSize: 12), //未选中的样式

import 'package:flutter/material.dart';  
  
class iTabBar extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return new MaterialApp(  
 home: new DefaultTabController(  
 length: choices.length,  
 child: new Scaffold(  
 appBar: new AppBar(  
 title: const Text('Tabbed AppBar'),  
 bottom: new TabBar(  
 isScrollable: true, //是否可滚动  
 indicatorColor: Colors.red, // 选中的那个横条的颜色  
 indicatorWeight: 10, // 选中的那个横条的高  
 indicatorPadding: const EdgeInsets.all(2), // 底部指示器的Padding  
 indicatorSize: TabBarIndicatorSize.tab, //选中那个横条的宽是随文字还是随图片  
 labelColor: Colors.deepPurple, // 文字的颜色 当设了 unselectedLabelColor,就是选中的颜色  
 labelStyle: TextStyle(fontSize: 18), // 文字的样式  
 unselectedLabelColor: Colors.teal, //未选中的颜色  
 unselectedLabelStyle: TextStyle(fontSize: 12), //未选中的样式  
 tabs: choices.map((Choice choice) {  
 return new Tab(  
 text: choice.title,  
 icon: new Icon(choice.icon),  
 );  
 }).toList(),  
 ),  
 ),  
 body: new TabBarView(  
 children: choices.map((Choice choice) {  
 return new Padding(  
 padding: const EdgeInsets.all(16.0),  
 child: new ChoiceCard(choice: choice),  
 );  
 }).toList(),  
 ),  
 ),  
 ),  
 );  
 }  
}  
  
class Choice {  
 const Choice({ this.title, this.icon });  
 final String title;  
 final IconData icon;  
}  
  
const List<Choice> choices = const <Choice>[  
 const Choice(title: 'CAR', icon: Icons.directions\_car),  
 const Choice(title: 'BICYCLE', icon: Icons.directions\_bike),  
 const Choice(title: 'BOAT', icon: Icons.directions\_boat),  
 const Choice(title: 'BUS', icon: Icons.directions\_bus),  
 const Choice(title: 'TRAIN', icon: Icons.directions\_railway),  
 const Choice(title: 'WALK', icon: Icons.directions\_walk),  
 const Choice(title: 'TRAIN', icon: Icons.directions\_railway),  
 const Choice(title: 'WALK', icon: Icons.directions\_walk),  
];  
  
class ChoiceCard extends StatelessWidget {  
 const ChoiceCard({ Key key, this.choice }) : super(key: key);  
  
 final Choice choice;  
  
 @override  
 Widget build(BuildContext context) {  
 final TextStyle textStyle = Theme.of(context).textTheme.display1;  
 return new Card(  
 color: Colors.white,  
 child: new Center(  
 child: new Column(  
 mainAxisSize: MainAxisSize.min,  
 crossAxisAlignment: CrossAxisAlignment.center,  
 children: <Widget>[  
 new Icon(choice.icon, size: 128.0, color: textStyle.color),  
 new Text(choice.title, style: textStyle),  
 ],  
 ),  
 ),  
 );  
 }  
}

### Drawer(侧栏)

实例名为：和iBottomNavigationBar.dart 在一起

在Scaffold下

drawerEdgeDragWidth: 0.0, // 禁止手势滑出drawer

Drawer(  
 child: ListView(  
 //ListView padding 不为空的时候，Drawer顶部的状态栏就不会有灰色背景  
 padding: EdgeInsets.zero,  
 children: <Widget>[  
 UserAccountsDrawerHeader(  
 //头像  
 currentAccountPicture: GestureDetector(  
 //圆形头像  
 child: CircleAvatar(  
 backgroundColor: Colors.cyan,  
 child: Icon(  
 Icons.menu,  
 size: 60,  
 color: Colors.white,  
 ),  
 ),  
 onTap: () {  
 print('点击了头像');  
 },  
 ),  
 //其他头像  
 otherAccountsPictures: <Widget>[  
 Icon(  
 Icons.stars,  
 color: Colors.white,  
 ),  
 ],  
 //用户名  
 accountName: Text(  
 "这是名称",  
 style: TextStyle(fontWeight: FontWeight.bold, fontSize: 22),  
 ),  
 //用户邮箱  
 accountEmail: Text("这是邮箱"),  
 onDetailsPressed: () {  
 print('详细信息');  
 },  
  
 ),  
 ///功能列表  
 ListTile(  
 leading: new Icon(Icons.favorite\_border),  
 title: Text("收藏"),  
 trailing: new Icon(Icons.chevron\_right),  
 onTap: () {  
 print('收藏');  
 },  
 ),  
 ListTile(  
 leading: new Icon(Icons.info\_outline),  
 title: Text("关于"),  
 trailing: new Icon(Icons.chevron\_right),  
 onTap: () {  
 print('关于');  
 },  
 ),  
 ListTile(  
 leading: new Icon(Icons.share),  
 title: Text("分享"),  
 trailing: new Icon(Icons.chevron\_right),  
 onTap: () {  
 print('分享');  
 },  
 ),  
  
 Divider(), //分割线  
 ListTile(  
 leading: new Icon(Icons.block,color: Colors.red,),  
 title: Text("退出",style: TextStyle(color: Colors.red),),  
 onTap: () {  
 print('退出');  
 //关闭drawer  
 Navigator.of(context).pop();  
 },  
 ),  
 ],  
 ),  
),

上边的方法并没有些打开的按钮的位置，图标，点击事件等，意味着有Drawer就会自动生成一个左上脚，meun的符号，如果不要它的，要这样写

GlobalKey<ScaffoldState> \_globalKey =new GlobalKey();  
  
@override  
Widget build(BuildContext context) {  
 return Scaffold(  
 key: \_globalKey,  
 appBar: AppBar(  
 leading: IconButton(  
 icon: const Icon(Icons.save),  
 tooltip: 'Next page',  
 onPressed: () {  
 \_globalKey.currentState.openDrawer();  
 },  
 ),  
 title: const Text('BottomNavigationBar Sample'),  
 ),

…………

### MaterialApp(路由的设定)

MaterialApp(  
 routes: <String, WidgetBuilder>{  
 '/': (BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Home Route'),  
 ),  
 );  
 },  
 '/about': (BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('About Route'),  
 ),  
 );  
 }  
 },  
)

## 对话框

### toast

引入类iToast.dark

import 'package:flutter/cupertino.dart';  
import 'package:flutter/material.dart';  
  
//Toast 显示位置控制  
enum ToastPostion {  
 top,  
 center,  
 bottom,  
}  
  
class Toast {  
 // toast靠它加到屏幕上  
 static OverlayEntry \_overlayEntry;  
 // toast是否正在showing  
 static bool \_showing = false;  
 // 开启一个新toast的当前时间，用于对比是否已经展示了足够时间  
 static DateTime \_startedTime;  
 // 提示内容  
 static String \_msg;  
 // toast显示时间  
 static int \_showTime;  
 // 背景颜色  
 static Color \_bgColor;  
 // 文本颜色  
 static Color \_textColor;  
 // 文字大小  
 static double \_textSize;  
 // 显示位置  
 static ToastPostion \_toastPosition;  
 // 左右边距  
 static double \_pdHorizontal;  
 // 上下边距  
 static double \_pdVertical;  
 static void toast(  
 BuildContext context, {  
 //显示的文本  
 String msg,  
 //显示的时间 单位毫秒  
 int showTime = 1000,  
 //显示的背景  
 Color bgColor = Colors.black,  
 //显示的文本颜色  
 Color textColor = Colors.white,  
 //显示的文字大小  
 double textSize = 14.0,  
 //显示的位置  
 ToastPostion position = ToastPostion.center,  
 //文字水平方向的内边距  
 double pdHorizontal = 20.0,  
 //文字垂直方向的内边距  
 double pdVertical = 10.0,  
 }) async {  
 assert(msg != null);  
 \_msg = msg;  
 \_startedTime = DateTime.now();  
 \_showTime = showTime;  
 \_bgColor = bgColor;  
 \_textColor = textColor;  
 \_textSize = textSize;  
 \_toastPosition = position;  
 \_pdHorizontal = pdHorizontal;  
 \_pdVertical = pdVertical;  
 //获取OverlayState  
 OverlayState overlayState = Overlay.of(context);  
 \_showing = true;  
 if (\_overlayEntry == null) {  
 //OverlayEntry负责构建布局  
 //通过OverlayEntry将构建的布局插入到整个布局的最上层  
 \_overlayEntry = OverlayEntry(  
 builder: (BuildContext context) => Positioned(  
 //top值，可以改变这个值来改变toast在屏幕中的位置  
 top: buildToastPosition(context),  
 child: Container(  
 alignment: Alignment.center,  
 width: MediaQuery.of(context).size.width,  
 child: Padding(  
 padding: EdgeInsets.symmetric(horizontal: 40.0),  
 child: AnimatedOpacity(  
 opacity: \_showing ? 1.0 : 0.0, //目标透明度  
 duration: \_showing  
 ? Duration(milliseconds: 100)  
 : Duration(milliseconds: 400),  
 child: \_buildToastWidget(),  
 ),  
 )),  
 ));  
 //插入到整个布局的最上层  
 overlayState.insert(\_overlayEntry);  
 } else {  
 //重新绘制UI，类似setState  
 \_overlayEntry.markNeedsBuild();  
 }  
 // 等待时间  
 await Future.delayed(Duration(milliseconds: \_showTime));  
 //2秒后 到底消失不消失  
 if (DateTime.now().difference(\_startedTime).inMilliseconds >= \_showTime) {  
 \_showing = false;  
 \_overlayEntry.markNeedsBuild();  
 await Future.delayed(Duration(milliseconds: 400));  
 \_overlayEntry.remove();  
 \_overlayEntry = null;  
 }  
 }  
  
 //toast绘制  
 static \_buildToastWidget() {  
 return Center(  
 child: Card(  
 color: \_bgColor,  
 child: Padding(  
 padding: EdgeInsets.symmetric(  
 horizontal: \_pdHorizontal, vertical: \_pdVertical),  
 child: Text(  
 \_msg,  
 style: TextStyle(  
 fontSize: \_textSize,  
 color: \_textColor,  
 ),  
 ),  
 ),  
 ),  
 );  
 }  
  
// 设置toast位置  
 static buildToastPosition(context) {  
 var backResult;  
 if (\_toastPosition == ToastPostion.top) {  
 backResult = MediaQuery.of(context).size.height \* 1 / 4;  
 } else if (\_toastPosition == ToastPostion.center) {  
 backResult = MediaQuery.of(context).size.height \* 2 / 5;  
 } else {  
 backResult = MediaQuery.of(context).size.height \* 3 / 4;  
 }  
 return backResult;  
 }  
}

使用

//默认是显示在中间的

Toast.toast(context,msg: "中间显示的 ");

Toast.toast(context,msg: "中间显示的 ",position: ToastPostion.center);

Toast.toast(context,msg: "顶部显示的 Toast $\_count",position: ToastPostion.top);

Toast.toast(context,msg: "底部显示的 Toast $\_count",position: ToastPostion.bottom);

1. SimpleDialog
2. AlertDialog
3. BottomSheet
4. ExpansionPanel
5. SnackBar

## 信息展示

### Image

实例名为：和iImages.dart

1. 图片的几种加载方式

Image：通过ImageProvider来加载图片

Image.asset：用来加载本地资源图片

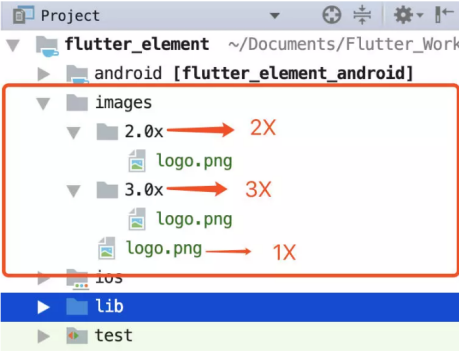
Image.file：用来加载本地（File文件）图片

Image.network：用来加载网络图片

Image.memory：用来加载Uint8List资源（byte数组）图片

1. 本地图片如何配置

加载一个本地资源图片，和 Android 一样，有多种分辨率的图片可供选择，但是沿袭的是 iOS 的图片风格，分为 1x，2x，3x，具体做法是在项目的根目录下创建两个文件夹，如下图所示：



1. API

**[width & height]** 用来指定显示图片区域的宽高（并非图片的宽高）

**[repeat]** 用来设置图片重复显示（repeat-x水平重复，repeat-y垂直重复，repeat两个方向都重复，no-repeat默认情况不重复）

**[centerSlice ]** 设置图片内部拉伸，相当于在图片内部设置了一个.9图，但是需要注意的是，要在显示图片的大小大于原图的情况下才可以使用这个属性，要不然会报错

**[fit]**

BoxFit.contain 全图居中显示但不充满，显示原比例

BoxFit.cover 图片可能拉伸，也可能裁剪，但是充满容器

BoxFit.fill 全图显示且填充满，图片可能会拉伸

BoxFit.fitHeight 图片可能拉伸，可能裁剪，高度充满

BoxFit.fitWidth 图片可能拉伸，可能裁剪，宽度充满

BoxFit.scaleDown 效果和contain差不多， 但是只能缩小图片，不能放大图片

1. 默认加载一个本地图片，等网络图片加载完了再加载

new FadeInImage.assetNetwork(  
 placeholder: 'images/loading.png',  
 image: 'http://n.sinaimg.cn/sports/2\_img/upload/cf0d0fdd/107/w1024h683/20181128/pKtl-hphsupx4744393.jpg',  
 width: 120,  
 fit: BoxFit.fitWidth,  
 height: 120.0  
),

1. 圆

new CircleAvatar( //圆  
 backgroundImage: AssetImage('images/321.png'),  
 radius: 50.0,  
),

1. 椭圆

new ClipOval( //椭圆  
 child: Image.asset('images/321.png',scale: 3,)  
),

1. 圆角

使用裁剪来实现图片圆角：

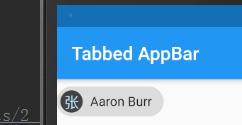
new ClipRRect(  
 child: Image.asset(  
 'images/321.png',  
 scale: 3,  
 fit: BoxFit.cover,  
 ),  
 borderRadius: BorderRadius.only(  
 topLeft: Radius.circular(10),  
 topRight: Radius.circular(10),  
 ),  
),

使用边框来实现图片圆角

new Container(  
 width: 60,  
 height: 60,  
 decoration: BoxDecoration(  
 shape: BoxShape.rectangle,  
 borderRadius: BorderRadius.circular(10.0),  
 image: DecorationImage(  
 image: AssetImage('images/321.png'),  
 fit: BoxFit.cover),  
 ),  
)

### Icon //todo

### Chip



Chip(  
 avatar: CircleAvatar(  
 backgroundColor: Colors.grey.shade800,  
 child: Text('张'),  
 ),  
 label: Text('Aaron Burr'),  
)

### Tooltip

长按显示提示文字

Tooltip(  
 message: "删除",  
 preferBelow:false, //设置false就是在上面弹  
 child: Icon(  
 Icons.delete,  
 size: 50.0,  
 )  
),

### DataTable

### Card

### LinearProgressIndicator

## 交互

### GestureDetector手势

实例名为：iGestureDetector.dart

用来监听各种事件，即使组件本身不提供，也可以外套这个进行添加事件

GestureDetector(  
 child: Container(color: Color(0xffff0000),),  
 onTapDown: (\_) => debugPrint("onTapDown"),  
 onTapUp: (\_) => debugPrint("onTapUp"),  
 onTap: () => debugPrint("onTap"),  
 onTapCancel: () => debugPrint("onTapCancel"),  
 onDoubleTap: () => debugPrint("onDoubleTap"),  
 onLongPress: () => debugPrint("onLongPress"),  
 onLongPressUp: () => debugPrint("onLongPressUp"),  
  
 onVerticalDragDown: (\_) => debugPrint("onVerticalDragDown"),  
 onVerticalDragStart: (\_) => debugPrint("onVerticalDragStart"),  
 onVerticalDragUpdate: (\_) => debugPrint("onVerticalDragUpdate"),  
 onVerticalDragEnd: (\_) => debugPrint("onVerticalDragEnd"),  
 onVerticalDragCancel: () => debugPrint("onVerticalDragCancel"),  
  
// onHorizontalDragDown: (\_) => debugPrint("onHorizontalDragDown"),  
// onHorizontalDragStart: (\_) => debugPrint("onHorizontalDragStart"),  
// onHorizontalDragUpdate: (\_) => debugPrint("onHorizontalDragUpdate"),  
// onHorizontalDragEnd: (\_) => debugPrint("onHorizontalDragEnd"),  
// onHorizontalDragCancel: () => debugPrint("onHorizontalDragCancel"),  
  
// onPanDown: (\_) => debugPrint("onPanDown"),  
// onPanStart: (\_) => debugPrint("onPanStart"),  
// onPanUpdate: (\_) => debugPrint("onPanUpdate"),  
// onPanEnd: (\_) => debugPrint("onPanEnd"),  
// onPanCancel: () => debugPrint("onPanCancel"),  
  
 onScaleStart: (\_) => debugPrint("onScaleStart"),  
 onScaleUpdate: (\_) => debugPrint("onScaleUpdate"), // 双指放大  
 onScaleEnd: (\_) => debugPrint("onScaleEnd"),  
 ),

### Listener监听

flutter没有阻止冒泡的功能

Listener可以加在Container等一系列容器的外边

Listener(  
 onPointerDown: (e){  
 print('down');  
 },  
 onPointerUp: (e){  
 print('up');  
 },  
 onPointerMove: (e){  
 print('move');  
 print(e);  
 },  
 onPointerCancel: (e){  
 print('cancel');  
 },  
 child: Container(  
 width: 200,  
 height: 200,  
 color: Colors.red,  
 ),  
),

### Http请求-参照插件Dio

### 调原生方法//todo

## 其它

### MediaQuery

通常情况下，不会直接将MediaQuery当作一个控件，而是使用MediaQuery.of获取当前设备的信息，用法如下：

var data = MediaQuery.of(context);

| **属性** | **说明** |
| --- | --- |
| size | 逻辑像素，并不是物理像素，类似于Android中的dp，逻辑像素会在不同大小的手机上显示的大小基本一样，物理像素 = size\*devicePixelRatio。 |
| devicePixelRatio | 单位逻辑像素的物理像素数量，即设备像素比。 |
| textScaleFactor | 单位逻辑像素字体像素数，如果设置为1.5则比指定的字体大50%。 |
| platformBrightness | 当前设备的亮度模式，比如在Android Pie手机上进入省电模式，所有的App将会使用深色（dark）模式绘制。 |
| viewInsets | 被系统遮挡的部分，通常指键盘，弹出键盘，viewInsets.bottom表示键盘的高度。 |
| padding | 被系统遮挡的部分，通常指“刘海屏”或者系统状态栏。 |
| viewPadding | 被系统遮挡的部分，通常指“刘海屏”或者系统状态栏，此值独立于padding和viewInsets，它们的值从MediaQuery控件边界的边缘开始测量。在移动设备上，通常是全屏。 |
| systemGestureInsets | 显示屏边缘上系统“消耗”的区域输入事件，并阻止将这些事件传递给应用。比如在Android Q手势滑动用于页面导航（ios也一样），比如左滑退出当前页面。 |
| physicalDepth | 设备的最大深度，类似于三维空间的Z轴。 |
| alwaysUse24HourFormat | 是否是24小时制。 |
| accessibleNavigation | 用户是否使用诸如TalkBack或VoiceOver之类的辅助功能与应用程序进行交互，用于帮助视力有障碍的人进行使用。 |
| invertColors | 是否支持颜色反转。 |
| highContrast | 用户是否要求前景与背景之间的对比度高， iOS上，方法是通过“设置”->“辅助功能”->“增加对比度”。 此标志仅在运行iOS 13的iOS设备上更新或以上。 |
| disableAnimations | 平台是否要求尽可能禁用或减少动画。 |
| boldText | 平台是否要求使用粗体。 |
| orientation | 是横屏还是竖屏。 |

### List操作之fold

var allTotal = \_list.fold(0,(pre,item) => pre + num.parse(item['quantity'].toString()) \* num.parse(item['unit\_price'].toString()));  
print("test fold ====> $allTotal");

## 导航与路由Navigator&Router

实例名为：iNavigator.dart

### 带router的配置，完了跳转pushNamed

使用pushNamed方法跳转 缺点是不能传参

先写公共配置类，在routes中可以把所有类配置好

class iNavigatorRoute extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return new MaterialApp(  
 title: 'Flutter Demos',  
 routes: {  
 '/': (BuildContext context) => new iNavigator1(),  
 '/lo2': (BuildContext context) => new iNavigator2()  
 },  
 );  
 }  
}

跳转：Navigator.of(context).pushNamed('/lo2');

### 可传参的路由Navigator.push

使用Navigator.push跳转并传参

还是一样，先写公共配置类 这个类不定义routes，但定义home还展示默认首页

class iNavigator extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return new MaterialApp(  
 title: 'Navigation',  
 home: new FirstScreen(),  
 );  
 }  
}

接收参数的类，在类中定义同名类的方法，并传参（构造函数）

String article;  
SecondScreen(this.article); //

跳转页面并传参

Navigator.push(  
 context,  
 new MaterialPageRoute(builder: (context) => new SecondScreen('张三')),  
);

### 返回，返回带参数

返回：Navigator.pop(context);

返回带参数

Navigator.pop(context,”这是来自第二个页面的值”);

直接定义返回结果可获得返回值

onPressed: () async {  
 String result = await Navigator.push(  
 context,  
 new MaterialPageRoute(builder: (context) => new SecondScreen('张三')),  
 );  
 print(result);  
 },

### 清除返回列表中的几个队列指定返回到那个页面

本例中我们的跳转顺序是Screen1—>Screen2—>Screen3—>Screen4

当从Screen3跳转到Screen4的时候我们希望将Screen2，Screen3从栈里面移除掉，这样在Screen4点击返回就能直接回到Screen1

使用Navigator.pushAndRemoveUtil()或者Navigator.pushNamedAndRemoveUntil()实现，这个方法有两个必传参数newRoute和predicate，第一个参数表示将要加入栈中的页面，第二个参数表示栈中要保留的页面底线，意思就是在predicate和newRoute之间的页面都会被移除栈

Navigator.of(context).pushAndRemoveUntil(

MaterialPageRoute(builder: (context) => Screen4()),

ModalRoute.withName('/screen1'));

或者

Navigator.of(context).pushNamedAndRemoveUntil(

'/screen4', ModalRoute.withName('/screen1'));

### pushReplacement 清除返回列表中的所有队列

假设我们在做一个登录功能，在登录成功后需要跳转到一个新的页面并且销毁当前登录页，这时候就可以用pushReplacement来实现，从字面上的意思看push很好理解，把一个新页面压入栈中嘛，replacement到底是替代那个页面呢？答案是当前页，因为被替换的对象是固定的，所以该方法的必传参数只有被push的路由。

打开Screen4页面，并销毁当前页

Navigator.of(context).pushReplacementNamed('/screen4');

或者

Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context)=>Screen3()));

### popUntil 一直退出直到某一个页面

Navigator.of(context).popUntil(ModalRoute.withName('/screen1'));

### popAndPushNamed 开启一个新的页面并且销毁之前的页

这个方法和pushReplacement很相近，都是开启一个新的页面并且销毁之前的页面，只是在逻辑上的执行顺序不一样，popAndPushNamed是退出当前页面并且将新的页面放到它原来的位置上，所以在视觉效果上是先退出再进入

Navigator.of(context).popAndPushNamed('/screen3');

### 跳转的特效

创建特效果类

class CustomRoute extends PageRouteBuilder{  
 final Widget widget;  
  
 CustomRoute(this.widget)  
 :super(  
 // 设置过度时间  
 transitionDuration:Duration(seconds: 1),  
 // 构造器  
 pageBuilder:(  
 // 上下文和动画  
 BuildContext context,  
 Animation<double> animaton1,  
 Animation<double> animaton2,  
 ){  
 return widget;  
 },  
 transitionsBuilder:(  
 BuildContext context,  
 Animation<double> animaton1,  
 Animation<double> animaton2,  
 Widget child,  
 ){  
  
 // 需要什么效果把注释打开就行了  
 // 渐变效果  
// return FadeTransition(  
// // 从0开始到1  
// opacity: Tween(begin: 0.0,end: 1.0)  
// .animate(CurvedAnimation(  
// // 传入设置的动画  
// parent: animaton1,  
// // 设置效果，快进漫出 这里有很多内置的效果  
// curve: Curves.fastOutSlowIn,  
// )),  
// child: child,  
// );  
  
 // 缩放动画效果  
 // return ScaleTransition(  
 // scale: Tween(begin: 0.0,end: 1.0).animate(CurvedAnimation(  
 // parent: animaton1,  
 // curve: Curves.fastOutSlowIn  
 // )),  
 // child: child,  
 // );  
  
 // 旋转加缩放动画效果  
 // return RotationTransition(  
 // turns: Tween(begin: 0.0,end: 1.0)  
 // .animate(CurvedAnimation(  
 // parent: animaton1,  
 // curve: Curves.fastOutSlowIn,  
 // )),  
 // child: ScaleTransition(  
 // scale: Tween(begin: 0.0,end: 1.0)  
 // .animate(CurvedAnimation(  
 // parent: animaton1,  
 // curve: Curves.fastOutSlowIn  
 // )),  
 // child: child,  
 // ),  
 // );  
  
// 左右滑动动画效果  
 return SlideTransition(  
 position: Tween<Offset>(  
 // 设置滑动的 X , Y 轴  
 begin: Offset(-1.0, 0.0),  
 end: Offset(0.0,0.0)  
 ).animate(CurvedAnimation(  
 parent: animaton1,  
 curve: Curves.fastOutSlowIn  
 )),  
 child: child,  
 );  
 }  
 );  
}

实现

Navigator.of(context).push(CustomRoute(SecondScreen('张三')));

### 路由插件fluro

# 功能实例

## X

# 插件

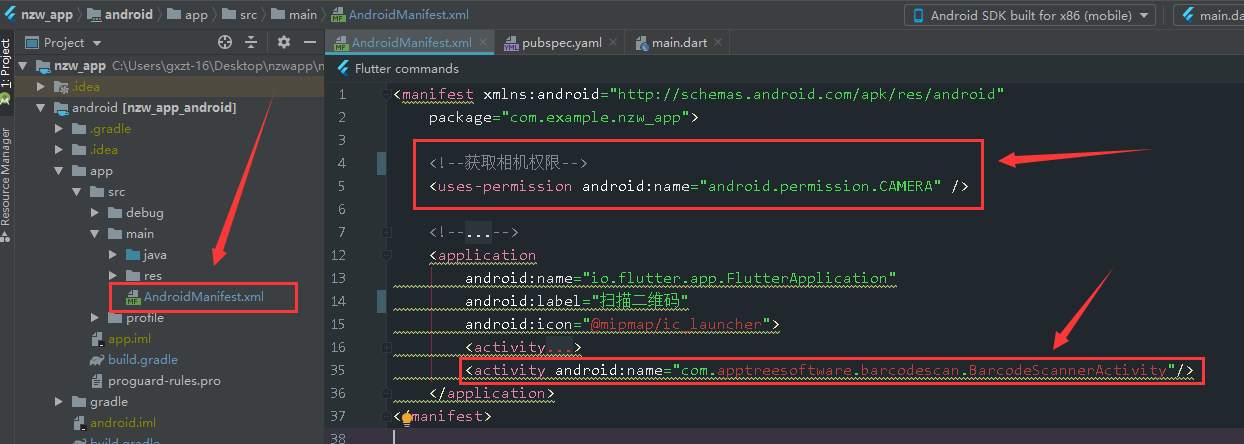
## [barcode\_scan](https://pub.dev/packages/image_picker)

1. 添加插件

dependencies: ... barcode\_scan: ^1.0.0

1. Android配置

<!--获取相机权限--> <uses-permission android:name="android.permission.CAMERA" /> <activity android:name="com.apptreesoftware.barcodescan.BarcodeScannerActivity"/>



1. 编辑项目级build.gradle文件

buildscript {

    ext.kotlin\_version = '1.2.31'

    ...

    dependencies {

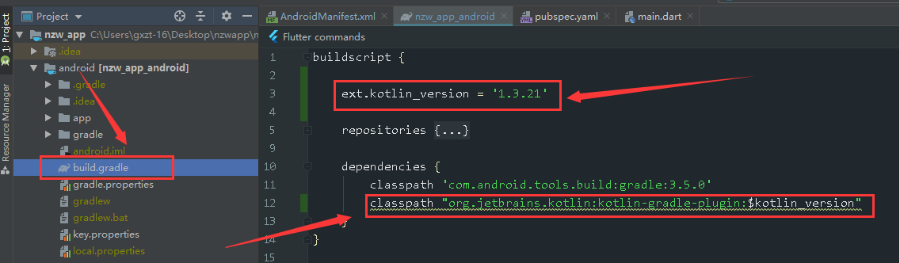
        ...

        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin\_version"

    }

}

...



1. 编辑应用级build.gradle文件

apply plugin: 'kotlin-android'

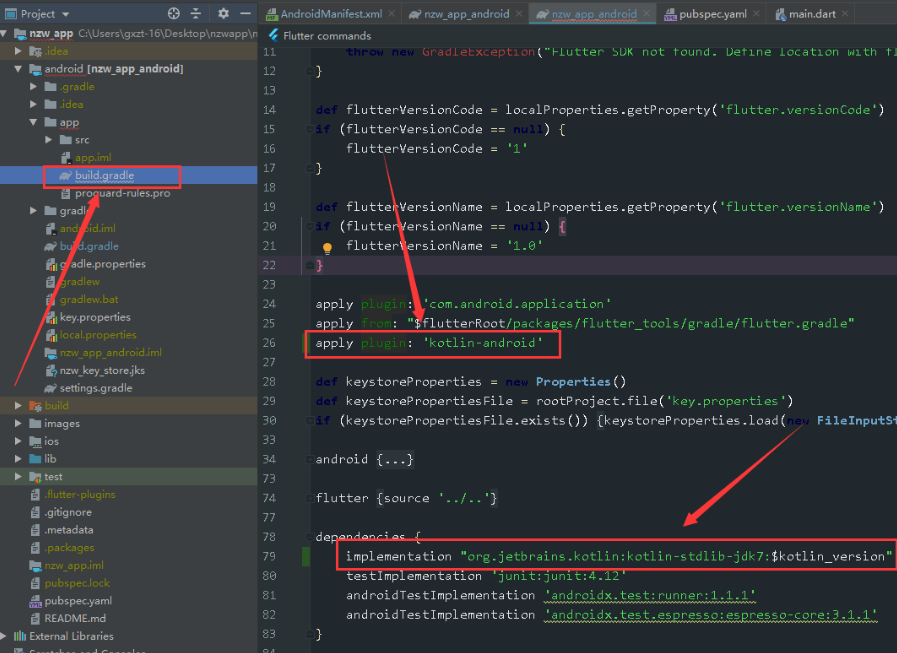
...

dependencies {

    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin\_version"

    ...

}



1. 扫码实现

import 'package:barcode\_scan/barcode\_scan.dart';

## [qrscan（扫码推荐）](https://pub.dev/packages/image_picker)

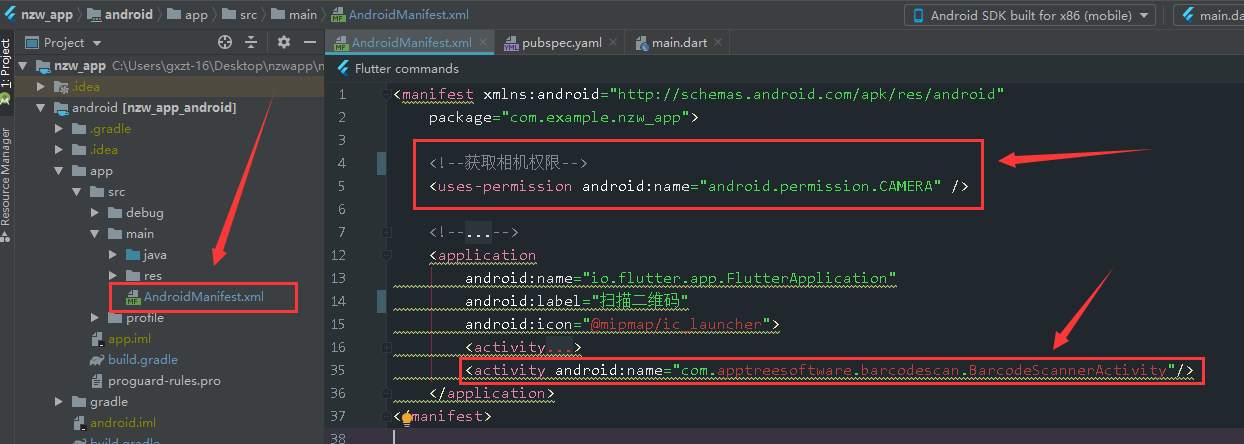
示例名：iQRScan.dart

1. 添加插件

qrscan: ^0.2.17

1. Android配置

<uses-permission android:name="android.permission.CAMERA" /><uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/><uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"/>



1. 使用

String cameraScanResult = await scanner.scan();

Future scan() async {  
 try {  
 String barcode = await Qrscan.scan();  
 print(barcode);  
 setState(() => this.barcode = barcode);  
 } catch (e) {  
 if (e.code == Qrscan.CameraAccessDenied) {  
 setState(() {  
 this.barcode = 'The user did not grant the camera permission!';  
 });  
 } else {  
 setState(() => this.barcode = 'Unknown error: $e');  
 }  
 }   
}

## [url\_launcher](https://pub.dev/packages/image_picker)

示例名：iUrlLauncher.dart

跳转网页或者打电话

1. 添加插件

url\_launcher: 5.4.1

1. 引入

import 'package:url\_launcher/url\_launcher.dart';

1. 使用

\_launchURL() async {  
 const url = 'tel:13648711629';  
 if (await canLaunch(url)) {  
 await launch(url);  
 } else {  
 throw 'Could not launch $url';  
 }  
}

## local\_auth

指纹识别

可以实现，但目前还有bug 报错 no\_fragment\_activity

## [image\_picker](https://pub.dev/packages/image_picker)

Flutter插件，用于从Android和iOS图像库中选择图像，并使用相机拍摄新照片。

实例名为：iImagePicker.dart

包地址：

<https://pub.dev/packages/image_picker>[r](https://pub.dev/packages/flutter_swiper)

安装：

dependencies:

image\_picker 0.6.3 // 低版本有bug

引入

import 'package:image\_picker/image\_picker.dart';

拍照

var image = await ImagePicker.pickImage(source: ImageSource.camera);

相册

var image = await ImagePicker.pickImage(source: ImageSource.gallery);

完整示例

class \_iImagePickerfulState extends State<iImagePickerful> {  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text('432'),  
 ),  
 body: SingleChildScrollView(  
 child: Column(  
 children: <Widget>[  
 \_ImageView(\_imgPath),  
 RaisedButton(  
 onPressed: \_takePhoto,  
 child: Text("拍照"),  
 ),  
 RaisedButton(  
 onPressed: \_openGallery,  
 child: Text("选择照片"),  
 ),  
 ],  
 ),  
 )  
  
 );  
 }  
  
 /\*图片控件\*/  
 Widget \_ImageView(imgPath) {  
 if (imgPath == null) {  
 return Center(  
 child: Text("请选择图片或拍照"),  
 );  
 } else {  
 return Image.file(  
 imgPath,  
 );  
 }  
 }  
 var \_imgPath;  
 /\*拍照\*/  
 \_takePhoto() async {  
 var image = await ImagePicker.pickImage(source: ImageSource.camera);  
 print(image);  
 setState(() {  
 \_imgPath = image;  
 });  
 }  
  
 /\*相册\*/  
 \_openGallery() async {  
 var image = await ImagePicker.pickImage(source: ImageSource.gallery);  
 setState(() {  
 \_imgPath = image;  
 });  
 }  
  
  
}

## [google\_maps\_flutter](https://pub.dev/packages/google_maps_flutter) //todo

## [flutter\_swiper](https://pub.dev/packages/flutter_swiper)

实例名为：iFlutter\_swiper.dart

包地址：

<https://pub.dev/packages/flutter_swiper>

安装：

dependencies:

flutter\_swiper: 1.1.6

引入

import 'package:flutter\_swiper/flutter\_swiper.dart';

使用

new Swiper(  
 itemBuilder: (BuildContext context,int index){  
 return new Image.network("http://via.placeholder.com/350x150",fit: BoxFit.fill,);  
 },  
 itemCount: 3,  
 pagination: new SwiperPagination(),  
 control: new SwiperControl(),  
),

备注：Swiper需要在State类中，父级需要有MaterialApp的实现，可以加Container来控制宽高

数字模式

Swiper(  
 itemBuilder: (BuildContext context, int index) {  
 return Image.network(  
 \_imageUrls[index],  
 fit: BoxFit.cover,  
 );  
 },  
 itemCount: \_imageUrls.length,  
 pagination: SwiperPagination(  
 alignment: Alignment.bottomRight,  
 margin: EdgeInsets.only(bottom: 10.0, right: 10.0),  
 builder: FractionPaginationBuilder(  
 color: Colors.black54,  
 )  
// builder: SwiperPagination.fraction,  
 ),  
 //小圆点  
 autoplay: false,  
 loop: false,  
 onTap: (int index) {},  
 ),

有飞入效果

Swiper(  
 itemBuilder: (BuildContext context, int index) {  
 return Image.network(  
 \_mdrUrls[index],  
 fit: BoxFit.cover,  
 );  
 },  
 itemCount: \_mdrUrls.length,  
 pagination: SwiperPagination(),  
layout: SwiperLayout.CUSTOM,  
customLayoutOption: new CustomLayoutOption(  
 startIndex: -1, stateCount: 3)  
 .addRotate(  
 [-45.0 / 180, 0.0, 45.0 / 180]).addTranslate([  
 new Offset(-370.0, -40.0),  
 new Offset(0.0, 0.0),  
 new Offset(370.0, -40.0)  
]),  
itemWidth: 150.0,  
itemHeight: 170.0,  
 //小圆点  
 autoplay: false,  
 loop: false,  
 onTap: (int index) {},  
 )

## [firebase\_storage](https://pub.dev/packages/firebase_storage) //todo

## [location](https://pub.dev/packages/location) //todo

## cool\_ui //todo

自定义键盘

<https://pub.flutter-io.cn/packages/cool_ui>

## Dio (HTTP请求)

实例名为：iHttp.dart

使用说明：<https://blog.csdn.net/yechaoa/article/details/90234708>

安装：

dependencies:

dio: 3.x #latest version

引入

import 'package:dio/dio.dart';

使用

void getHttp() async {  
 Response response;  
 Dio dio = new Dio();  
 response = await dio.get("https://llyapi.laiyunyou.cn/NKOrderManage/NK\_Station/GetNKStartStation");  
 print(response.data.resultdata.toString());  
}

请求头设置

BaseOptions options = new BaseOptions(

//请求基地址,可以包含子路径

baseUrl: "http://www.google.com",

//连接服务器超时时间，单位是毫秒.

connectTimeout: 10000,

//响应流上前后两次接受到数据的间隔，单位为毫秒。

receiveTimeout: 5000,

//Http请求头.

headers: {

//do something

"version": "1.0.0"

},

//请求的Content-Type，默认值是[ContentType.json]. 也可以用ContentType.parse("application/x-www-form-urlencoded")

contentType: ContentType.json,

//表示期望以那种格式(方式)接受响应数据。接受4种类型 `json`, `stream`, `plain`, `bytes`. 默认值是 `json`,

responseType: ResponseType.json,

);

dio = new Dio(options);

onPressed: () async{  
Response response;  
Dio dio = new Dio();  
response = await dio.get('https://llyapi.laiyunyou.cn/NKOrderManage/NK\_Station/GetNKStartStation', queryParameters: {});  
//dio.post("",data: {});  
//dio.download("下载地址", "存到本地的位置");  
var jsonRes = json.decode(response.toString());  
print(response);  
print(jsonRes['resultdata'].length.toString());  
},

请求完并更新到ListView

import 'dart:convert';  
import 'package:flutter/material.dart';  
import 'package:dio/dio.dart';  
  
class iHttp extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'iHttp',  
 home: iHttpful(),  
 );  
 }  
}  
  
  
class iHttpful extends StatefulWidget {  
 @override  
 \_iHttpfulState createState() => \_iHttpfulState();  
}  
  
class \_iHttpfulState extends State<iHttpful> {  
 @override  
 void initState() {  
 print('init state');  
 getHttp();  
 super.initState();  
 }  
 void getHttp() async {  
 Response response;  
 Dio dio = new Dio();  
 response = await dio.get('https://llyapi.laiyunyou.cn/NKOrderManage/NK\_Station/GetNKStartStation', queryParameters: {});  
 //dio.post("",data: {});  
 //dio.download("下载地址", "存到本地的位置");  
 var jsonRes = json.decode(response.toString());  
 print(jsonRes['resultdata'].length.toString());  
 List items = jsonRes["resultdata"];  
 items.forEach((f) => {  
 print(f["OrganizeName"])  
 });  
 setState(() {  
 \_entries =items;  
 });  
 }  
 List \_entries = [];  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text('iHttp'),  
 ),  
 body:  
 ListView(  
 children: \_entries.map((item) => Text(item['OrganizeName'])).toList(),  
 ),  
 );  
 }  
}

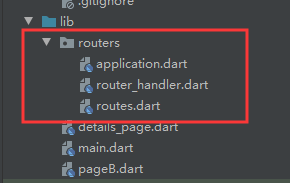
子插件，用于解决cookies的问题

dio\_cookie\_manager 1.0.0

## fluro (路由插件)

引入

fluro: ^1.5.1



建立routers文件夹

【**建立文件rotuer\_handler.dart**】

import 'package:flutter/material.dart';  
import 'package:fluro/fluro.dart';  
import '../details\_page.dart';  
  
  
Handler detailsHander =Handler(  
 handlerFunc: (BuildContext context,Map<String,List<String>> params){  
 return DetailsPage(systemType: 313);  
 }  
);

import引入对应文件

return flutter绘制类（stful）

类里边传的参数注意了，这里是参数名 : 值

【**建立文件routes.dart**】

import 'package:flutter/material.dart';  
import 'package:fluro/fluro.dart';  
import 'router\_handler.dart';  
  
class Routes { //配置类  
 static String root = '/'; //根目录  
 static String detailsPage = '/detafdsafdsil'; //详情页面  
 //静态方法  
 static void configureRoutes(Router router){//路由配置  
 //找不到路由  
 router.notFoundHandler = new Handler(  
 handlerFunc: (BuildContext context,Map<String,List<String>> params){  
 print('ERROR====>没有这个路由');  
 }  
 );  
 //整体配置  
 router.define(detailsPage, handler: detailsHander);  
 }  
}

detailsPage 原则上这个的值可以乱设，因为后边也是调这个变量

【**建立文件application.dart**】

import 'package:fluro/fluro.dart';

class Application{

static Router router;

}

固定写法

【**把路由注册/注入到顶层main.dart**】

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

//-------------------主要代码start

final router = Router(); //路由初始化

Routes.configureRoutes(router);

Application.router = router;

//-------------------主要代码end

return Container(

child: MaterialApp(

title:'百姓生活+',

debugShowCheckedModeBanner: false,

//----------------主要代码start onGenerateRoute: Application.router.generator, //路由静态化

//----------------主要代码end theme: ThemeData(

),

home:IndexPage()

),

);

}

}

【**事件传值**】

Application.router.navigateTo(context,Routes.detailsPage+"?systemType=1");

【**接收值**】

StatefulWidget 类中

final int systemType;  
DetailsPage({Key key, this.systemType}) : super(key: key);

State 中widget.变量名

传中文

编码解码方法

class FluroConvertUtils {  
 // fluro 传递中文参数前，先转换，fluro 不支持中文传递  
 static String fluroCnParamsEncode(String originalCn) {  
 return jsonEncode(Utf8Encoder().convert(originalCn));  
 }  
  
 // fluro 传递后取出参数，解析  
 static String fluroCnParamsDecode(String encodeCn) {  
 var list = List<int>();  
  
 // 字符串解码  
 jsonDecode(encodeCn).forEach(list.add);  
 String value = Utf8Decoder().convert(list);  
 return value;  
 }

}

navigateTo传的时候编码 fluroCnParamsEncode

Application.router.navigateTo(  
 context,  
 Routes.productDetails +  
 '?&vid=$vid&productType=$productType&productInnerSn=$productInnerSn&source=$source&vendorInnerSn=$vendorInnerSn&productName= ${FluroConvertUtils.fluroCnParamsEncode(productName)}&vendorName=${FluroConvertUtils.fluroCnParamsEncode(vendorName)}');  
}

解码 fluroCnParamsDecode

Handler productDetailPageHandler = Handler(  
 handlerFunc: (BuildContext context, Map<String, List<String>> params) {  
 return ProductDetailsPage(  
 vid: FluroConvertUtils.string2int(params['vid'].first),  
 productType: FluroConvertUtils.string2int(params['productType'].first),  
 productInnerSn:  
 FluroConvertUtils.string2int(params['productInnerSn'].first),  
 source: FluroConvertUtils.string2int(params['source'].first),  
 vendorInnerSn: FluroConvertUtils.string2int(params['vendorInnerSn'].first),  
 productName:  
 FluroConvertUtils.fluroCnParamsDecode(params['productName'].first),  
 vendorName:  
 FluroConvertUtils.fluroCnParamsDecode(params['vendorName'].first),  
 );  
});

【**传值给页面B，且由B返回数据**】

传过去的时候

Application.router  
 .navigateTo(  
 context,  
 Routes.buyerOrderDetail +  
 '?orderType=' + \_item['order\_type'].toString() +  
 '&partAId=' + \_item['partA'].toString() +  
 '&partBId=' + \_item['partB'].toString() +  
 '&poSoInnerSn=' + \_item['po\_so\_inner\_sn'].toString() +  
 '&pageType=' + \_pageType)  
 .then((detailsMap) {  
 if (detailsMap != null) {  
 Map<String, dynamic> \_curDetailsMap = detailsMap;  
 var a = \_curDetailsMap['status'];  
 var c = \_rows;  
 for(int i = 0;i<\_rows.length;i++){  
 if(\_rows[i]['po\_sn'].toString() == \_curDetailsMap['po\_sn'].toString()){  
 setState(() {  
 \_rows[i]['status'] = \_curDetailsMap['status'];  
 });  
 }  
 }  
 }  
});

返回的时候

Navigator.pop(context, {  
 'status': \_status,  
 'po\_sn': \_infoData['po\_sn'].toString()  
});

## [flutter\_easyrefresh](https://www.cnblogs.com/xuelongqy/p/10142664.html) +国际化

安装：

dependencies:

flutter\_easyrefresh: ^2.0.9 # 下拉刷新上拉加载

flutter\_localizations:  
 sdk: flutter # 国际化

引入

import 'package:photo\_view/photo\_view.dart';

默认最简单的使用方式：

// 方式一

EasyRefresh(

child: ListView(),

onRefresh: () async{

....

},

onLoad: () async {

....

},

)

国际化

引入easy\_refresh\_i18n.dart

@override  
 Widget build(BuildContext context) {  
 return BotToastInit(  
 child: new MaterialApp(  
 //指定简体中文,  
 localizationsDelegates: [  
 S.delegate,  
 GlobalEasyRefreshLocalizations.delegate,  
 GlobalCupertinoLocalizations.delegate,  
 GlobalMaterialLocalizations.delegate,  
 GlobalWidgetsLocalizations.delegate  
 ],  
 //指定简体中文,  
 supportedLocales: S.delegate.supportedLocales,  
 title: '整体标题',  
 home: new Scaffold(  
 appBar: AppBar(  
 title: Text('标题'),  
 ),  
 body: GestureDetector(  
 child: EasyRefresh(  
 child: new ListView.builder(  
 shrinkWrap: true,  
 physics: NeverScrollableScrollPhysics(),  
 // 当外层有滚动插件，这里要禁止滚动  
 itemCount: listData.length,  
 itemBuilder: (BuildContext context, int index) {  
 var \_item = listData[index];  
 return Container(  
 height: 80,  
 child: Text('${\_item['name']} $index'),  
 );  
 },  
 ),  
 onRefresh: () async {},  
 onLoad: () async {},  
 )))),  
 );  
 }

## flutter\_screenutil

## Photo View图片预览/相册

 使用说明：<https://blog.csdn.net/yechaoa/article/details/90234708>

安装：

dependencies:

photo\_view: 0.9.2 # 图片预览

引入

import 'package:photo\_view/photo\_view.dart';

默认最简单的使用方式：

@overrideWidget build(BuildContext context) {

return Container(

child: PhotoView(

imageProvider: AssetImage("assets/large-image.jpg"),

)

);}

单个文件 预览

import 'package:flutter/material.dart';  
import 'package:photo\_view/photo\_view.dart';  
  
class PhotoViewSimpleScreen extends StatelessWidget{  
 const PhotoViewSimpleScreen({  
 this.imageProvider,//图片  
 this.loadingChild,//加载时的widget  
 this.backgroundDecoration,//背景修饰  
 this.minScale,//最大缩放倍数  
 this.maxScale,//最小缩放倍数  
 this.heroTag,//hero动画tagid  
 });  
 final dynamic imageProvider;  
 final Widget loadingChild;  
 final Decoration backgroundDecoration;  
 final dynamic minScale;  
 final dynamic maxScale;  
 final String heroTag;  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 body: Container(  
 constraints: BoxConstraints.expand(  
 height: MediaQuery.of(context).size.height,  
 ),  
 child: Stack(  
 children: <Widget>[  
 Positioned(  
 top: 0,  
 left: 0,  
 bottom: 0,  
 right: 0,  
 child: PhotoView(  
 imageProvider: imageProvider,  
 loadingChild: loadingChild,  
 backgroundDecoration: backgroundDecoration,  
 minScale: minScale,  
 maxScale: maxScale,  
 heroAttributes: PhotoViewHeroAttributes(tag: heroTag),  
 enableRotation: true,  
 ),  
 ),  
 Positioned(//右上角关闭按钮  
 right: 10,  
 top: MediaQuery.of(context).padding.top,  
 child: IconButton(  
 icon: Icon(Icons.close,size: 30,color: Colors.white,),  
 onPressed: (){  
 Navigator.of(context).pop();  
 },  
 ),  
 )  
 ],  
 ),  
 ),  
 );  
 }  
  
}

跳转

Navigator.of(context).push(new FadeRoute(page: PhotoViewSimpleScreen(  
imageProvider:NetworkImage('https://yrtdev.bjyixiu.com/v1/image/get/FpbfVFa6xuUakf8gvVoqFO4QJHp5'),  
 heroTag: 'simple',  
)));

多个文件 预览

import 'package:flutter/cupertino.dart';  
import 'package:flutter/material.dart';  
import 'package:photo\_view/photo\_view.dart';  
import 'package:photo\_view/photo\_view\_gallery.dart';  
  
import 'colors.dart';  
  
class Album extends StatefulWidget {  
 List images = [];  
 int index = 0;  
 String heroTag;  
 PageController controller;  
  
 Album(  
 {Key key,  
 @required this.images,  
 this.index,  
 this.controller,  
 this.heroTag})  
 : super(key: key) {  
 controller = PageController(initialPage: index);  
 }  
  
 @override  
 \_AlbumState createState() => \_AlbumState();  
}  
  
class \_AlbumState extends State<Album> {  
 int currentIndex = 0;  
  
 @override  
 void initState() {  
 // TODO: implement initState  
 super.initState();  
 currentIndex = widget.index;  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 body: Container(  
 color: Color(ColorConfig.black),  
 child: Stack(  
 children: <Widget>[  
 Positioned(  
 top: 0,  
 left: 0,  
 bottom: 100,  
 right: 0,  
 child: Container(  
 child: PhotoViewGallery.builder(  
 scrollPhysics: const BouncingScrollPhysics(),  
 builder: (BuildContext context, int index) {  
 return PhotoViewGalleryPageOptions(  
 imageProvider: NetworkImage(widget.images[currentIndex]),  
 heroAttributes: widget.heroTag.isNotEmpty  
 ? PhotoViewHeroAttributes(tag: widget.heroTag)  
 : null,  
 );  
 },  
 itemCount: widget.images.length,  
 loadingChild: Container(),  
 backgroundDecoration: null,  
 pageController: widget.controller,  
 enableRotation: true,  
 onPageChanged: (index) {  
 setState(() {  
 currentIndex = index;  
 });  
 },  
 )),  
 ),  
 Positioned(  
 left: 0,  
 bottom: 0,  
 right: 0,  
 child: new Container(  
 color: Color(ColorConfig.white),  
 height: 100,  
 child: new CustomScrollView(  
 shrinkWrap: true,  
 scrollDirection: Axis.horizontal,  
 slivers: <Widget>[  
 new SliverPadding(  
 padding: const EdgeInsets.all(0.0),  
 sliver: new SliverList(  
 delegate: new SliverChildListDelegate(  
 \_imageView(widget.images),  
 ),  
 ),  
 ),  
 ],  
 )),  
 ),  
 Positioned(  
 //图片index显示  
 top: MediaQuery.of(context).padding.top + 15,  
 width: MediaQuery.of(context).size.width,  
 child: Center(  
 child: Text("${currentIndex + 1}/${widget.images.length}",  
 style: TextStyle(color: Colors.white, fontSize: 16)),  
 ),  
 ),  
 Positioned(  
 //右上角关闭按钮  
 right: 10,  
 top: MediaQuery.of(context).padding.top,  
 child: IconButton(  
 icon: Icon(  
 Icons.close,  
 size: 30,  
 color: Colors.white,  
 ),  
 onPressed: () {  
 Navigator.of(context).pop();  
 },  
 ),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
  
 List<Widget> \_imageView(imgList) {  
 return List.generate(imgList.length, (int Index) {  
 var \_imgListItems = imgList[Index];  
 return new Container(  
 width: 75,  
 height: 100,  
 decoration: BoxDecoration(  
 color: Color(ColorConfig.tf5f5f5),  
 image: DecorationImage(  
 image: NetworkImage(imgList[Index]), fit: BoxFit.cover),  
 ),  
 margin: const EdgeInsets.all(2.0),  
 child: FlatButton(  
 onPressed: () {  
 setState(() {  
 currentIndex = Index;  
 });  
 },  
 color: Colors.transparent,  
 ),  
 );  
 });  
 }  
}  
  
class FadeRoute extends PageRouteBuilder {  
 final Widget page;  
  
 FadeRoute({this.page})  
 : super(  
 pageBuilder: (  
 BuildContext context,  
 Animation<double> animation,  
 Animation<double> secondaryAnimation,  
 ) =>  
 page,  
 transitionsBuilder: (  
 BuildContext context,  
 Animation<double> animation,  
 Animation<double> secondaryAnimation,  
 Widget child,  
 ) =>  
 FadeTransition(  
 opacity: animation,  
 child: child,  
 ),  
 );  
}

跳转

List imgs =[  
'https://yrtdev.bjyixiu.com/v1/image/get/FpbfVFa6xuUakf8gvVoqFO4QJHp5',  
'https://yrtdev.bjyixiu.com/v1/image/get/FsXht0iY7ySlhsEXRBndluN0yhYz',  
'https://yrtdev.bjyixiu.com/v1/image/get/FivhCDnnWZ3c-lRvv-2mIbDMeh6t',  
'https://yrtdev.bjyixiu.com/v1/image/get/Fie8VMAWohyDJGghM5vS48bmXx5S'];  
  
Navigator.of(context).push(new FadeRoute(page: Album(  
 images:imgs,//传入图片list  
 index: 2,//传入当前点击的图片的index  
 heroTag: imgs[2],//传入当前点击的图片的hero tag （可选）  
)));

## flutter\_slidable列表左右滑动带出按钮选项

安装：

dependencies:

flutter\_slidable: "^0.5.4"

引入

import 'package:flutter\_slidable/flutter\_slidable.dart';

最简单使用格式

import 'package:flutter/material.dart';  
import 'package:flutter\_slidable/flutter\_slidable.dart';  
  
void main() => runApp(MyApp());  
  
class MyApp extends StatefulWidget {  
 @override  
 \_MyAppState createState() => \_MyAppState();  
}  
  
class \_MyAppState extends State<MyApp> {  
 @override  
 Widget build(BuildContext context) {  
 return new MaterialApp(  
 title: '整体标题',  
 home: new Scaffold(  
 appBar: AppBar(  
 title: Text('标题'),  
 ),  
 body: ListView(  
 children: <Widget>[  
 Slidable(  
 actionPane: SlidableDrawerActionPane(),  
 actionExtentRatio: 0.25,  
 child: Container(  
 color: Colors.white,  
 child: ListTile(  
 leading: CircleAvatar(  
 backgroundColor: Colors.indigoAccent,  
 child: Text('3'),  
 foregroundColor: Colors.white,  
 ),  
 title: Text('Tile n°3'),  
 subtitle: Text('SlidableDrawerDelegate'),  
 ),  
 ),  
 actions: <Widget>[  
 IconSlideAction(  
 caption: 'Archive',  
 color: Colors.blue,  
 icon: Icons.archive,  
 onTap: () => \_showSnackBar('Archive'),  
 ),  
 IconSlideAction(  
 caption: 'Share',  
 color: Colors.indigo,  
 icon: Icons.share,  
 onTap: () => \_showSnackBar('Share'),  
 ),  
 ],  
 secondaryActions: <Widget>[  
 IconSlideAction(  
 caption: 'More',  
 color: Colors.black45,  
 icon: Icons.more\_horiz,  
 onTap: () => \_showSnackBar('More'),  
 ),  
 IconSlideAction(  
 caption: 'Delete',  
 color: Colors.red,  
 icon: Icons.delete,  
 onTap: () => \_showSnackBar('Delete'),  
 ),  
 ],  
 ),  
 ],  
 )  
 )  
 );  
 }  
 \_showSnackBar(code) {  
 print(code);  
 }  
}

如何打开一个其他就关闭

1.定义SlidableController

SlidableController \_slidableController = new SlidableController();

2.在Slidable标签下的controller中设置  
Slidable(  
 controller: \_slidableController,

...

)

如何点击按钮不合起来（默认是合起来）

secondaryActions: <Widget>[  
 IconSlideAction(  
 caption: 'Delete',  
 color: Colors.red,  
 icon: Icons.delete,  
 closeOnTap: false, // 这个值设为false  
 onTap: (){

设置全局变量

GlobalKey key1=GlobalKey();

加入key

Slidable(  
 child: ListItem(  
 key: key1,  
 title: '阻止自动关闭',  
 )),

// 1秒后关闭

Future.delayed(Duration(seconds: 1),(){

Slidable.of(key1.currentContext).close();

});

// 代码关闭

Slidable.of(key1.currentContext).close();

..执行的方法  
 },  
 ),  
],

dismissal

GlobalKey key1=GlobalKey();

Slidable(  
 key: Key('123456'), // 这是关键  
 actionPane: SlidableStrechActionPane(),  
 actionExtentRatio: 0.25,  
 child: ListItem(  
 key: key2,  
 title: '滑动或点击删除记录',  
 ),  
 dismissal: SlidableDismissal(  
 child: SlidableDrawerDismissal(),  
 onWillDismiss: (actionType) {  
 return showDialog<bool>(  
 context: context,  
 builder: (context) {  
 return AlertDialog(  
 title: Text('Delete'),  
 content: Text('Item will be deleted'),  
 actions: <Widget>[  
 FlatButton(  
 child: Text('Cancel'),  
 onPressed: () => Navigator.of(context).pop(false),  
 ),  
 FlatButton(  
 child: Text('Ok'),  
 onPressed: () => Navigator.of(context).pop(true),  
 ),  
 ],  
 );  
 },  
 );  
 },  
 onDismissed: (actionType) {  
 print(actionType);  
 },  
 ),  
 secondaryActions: <Widget>[//右侧按钮列表  
 IconSlideAction(  
 caption: 'Delete',  
 color: Colors.red,  
 icon: Icons.delete,  
 closeOnTap: false,  
 onTap: (){  
 Slidable.of(key2.currentContext).dismiss(); // 这是关键  
 },  
 ),  
 ],  
),

## bot\_toast模态框

安装：

dependencies:

bot\_toast: ^2.2.0 # 模态框

引入

import 'package:bot\_toast/bot\_toast.dart';

最简单使用格式

Widget build(BuildContext context){

//1.使用BotToastInit直接包裹MaterialApp   
 return BotToastInit(  
 child: new MaterialApp(  
 navigatorObservers: [BotToastNavigatorObserver()], //2.注册路由观察者  
 title: '整体标题',  
 home: new Scaffold(  
 appBar: AppBar(  
 title: Text('标题'),  
 ),  
 body: FlatButton(  
 child: Text("f点我"),  
 onPressed: (){  
 BotToast.showText(text:"xxxx");  
 },  
 ),  
 )  
 )  
 );

弹出简单通知Toast

BotToast.showText(text:"xxxx");

弹出简单通知Toast

BotToast.showSimpleNotification(title: "init");

弹出一个加载动画

BotToast.showLoading();

关闭所有动画

BotToast.closeAllLoading();

所有参数地址：<https://github.com/MMMzq/bot_toast/blob/master/API.md>

提示文字的复杂写法

BotToast.showText(  
 text:"xxxx",  
 clickClose: true, // 是否在点击屏幕触发事件时自动关闭该Toast false 点了关闭  
 duration:Duration(seconds: 3), // 设为 null 永不关闭  
 backgroundColor:Colors.black54, // 总背景颜色  
 contentColor:Colors.red, // 弹框背景  
 borderRadius:BorderRadius.all(Radius.circular(5)),// 圆角  
 textStyle:TextStyle(fontSize: 30, color: Colors.white), // 文字颜色及大小  
);

通知的复杂写法

BotToast.showNotification(  
 duration: null,  
 enableSlideOff: true,  
 title: (cancel) {  
 return Text(  
 'title',  
 style: TextStyle(color: Colors.red, fontSize: 18),  
 );  
 },  
 leading: (cancel) {  
 return Container(  
 width: 30,  
 height: 30,  
 color: Colors.blue,  
 );  
 },  
 trailing: (cancel) {  
 return FlatButton(  
 child: Text('关闭'),  
 onPressed: () {  
 BotToast.cleanAll();  
 },  
 );  
 },  
 align: Alignment(0, -0.99)

);

showAttachedWidget 实现 点击按钮有下拉

import 'package:flutter/material.dart';  
import 'package:bot\_toast/bot\_toast.dart';  
  
void main() => runApp(MyApp());  
  
class MyApp extends StatefulWidget {  
 @override  
 \_MyAppState createState() => \_MyAppState();  
}  
  
class \_MyAppState extends State<MyApp> {  
 List listData = [  
 {  
 'key': 1,  
 'name': '待验货订单',  
 'color': 0xffF5A623,  
 'active': true,  
 'icon': 0xe6ca  
 },  
 {  
 'key': 2,  
 'name': '已验货订单',  
 'color': 0xff1677FC,  
 'active': false,  
 'icon': 0xe66e  
 },  
 {  
 'key': -1,  
 'name': '全部验货订单',  
 'color': 0xff11E3B4,  
 'active': false,  
 'icon': 0xe66d  
 },  
 ];  
  
 @override  
 Widget build(BuildContext context) {  
 return BotToastInit(  
 child: new MaterialApp(  
 navigatorObservers: [BotToastNavigatorObserver()], //2.注册路由观察者  
 title: '整体标题',  
 home: new Scaffold(  
 appBar: AppBar(  
 title: Text('标题'),  
 ),  
 body: GestureDetector(  
 child: Text('点我'),  
 onTapDown: (TapDownDetails point) {  
 showAttachedWidget(target: point.globalPosition);  
 },  
 ),  
 )));  
 }  
  
 /// 下拉筛选按钮  
 CancelFunc showAttachedWidget({BuildContext context, Offset target}) {  
 return BotToast.showAttachedWidget(  
 target: Offset(target.dx, 100),  
 verticalOffset: 30,

//animationDuration:Duration(milliseconds: 2),  
 animationDuration: Duration(seconds: 3), // 出现时间  
 animationReverseDuration: Duration(seconds: 3), // 消失时间  
 attachedBuilder: (CancelFunc cancelFunc) => (Stack(  
 overflow: Overflow.visible,  
 children: <Widget>[  
 Positioned(  
 top: 5,  
 right: 0,  
 child: SingleChildScrollView(  
 child: Card(  
 color: Colors.red,  
 elevation: 0,  
 child: Container(  
 color: Colors.blue,  
 width: 300,  
 margin: EdgeInsets.symmetric(  
 horizontal: 30.0,  
 vertical: 10.0,  
 ),  
 child: showAttachedWidgetItem(cancelFunc),  
 ),  
 ),  
 ),  
 ),  
 Positioned(  
 top: 0,  
 right: 130,  
 child: Container(  
 width: 10,  
 height: 10,  
 color: Colors.red,  
 ),  
 ),  
 ],  
 )));  
 }  
  
 // 下拉筛选列表子项  
 Widget showAttachedWidgetItem(cancelFunc) {  
 List<Widget> tiles = [];  
 Widget content;  
 for (var \_item in listData) {  
 tiles.add(  
 Column(  
 children: <Widget>[  
 InkWell(  
 onTap: () {  
 cancelFunc();  
 cancelFunc?.call();  
 for (var \_obj in listData) {  
 \_obj['active'] = false;  
 }  
 \_item['active'] = true;  
 setState(() {  
 listData = listData;  
 //\_status = \_item['key'];  
 });  
 },  
 child: Container(  
 height: 90,  
 decoration: BoxDecoration(  
 border: Border(  
 bottom: BorderSide(  
 color:  
 \_item['key'] == -1 ? Colors.white : Colors.blue,  
 width: 1))),  
 child: Row(  
 crossAxisAlignment: CrossAxisAlignment.center,  
 children: <Widget>[  
 Container(  
 margin: EdgeInsets.only(  
 left: 0,  
 right: 0,  
 ),  
 child: Text('图标')),  
 Container(  
 width: 240,  
 child: Text(  
 \_item['name'],  
 textAlign: TextAlign.left,  
 style: TextStyle(  
 color: Colors.black26,  
 ),  
 ),  
 ),  
 Container(  
 child: Icon(  
 \_item['active']  
 ? IconData(  
 0xe671,  
 fontFamily: 'myIconFonts',  
 )  
 : null,  
 color: Colors.black26,  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 ],  
 ),  
 );  
 }  
 content = new Column(children: tiles);  
 return content;  
 }  
}

showAnimationWidget实现 点击按钮有下拉

CancelFunc showAnimationWidget({BuildContext context, Offset target}) {  
 return BotToast.showAnimationWidget(  
 wrapToastAnimation: (controller, cancelFunc, widget) {  
 return SpinningContainer(  
 controller: controller,  
 );  
 },  
 //animationDuration:Duration(milliseconds: 2),  
 animationDuration: Duration(seconds: 3), // 出现时间  
 animationReverseDuration: Duration(seconds: 3), // 消失时间  
 toastBuilder: (CancelFunc cancelFunc) => (Stack(  
 overflow: Overflow.visible,  
 children: <Widget>[  
 Positioned(  
 top: 5,  
 right: 0,  
 child: SingleChildScrollView(  
 child: Card(  
 color: Colors.red,  
 elevation: 0,  
 child: Container(  
 color: Colors.blue,  
 width: 300,  
 margin: EdgeInsets.symmetric(  
 horizontal: 30.0,  
 vertical: 10.0,  
 ),  
 child: showAttachedWidgetItem(cancelFunc),  
 ),  
 ),  
 ),  
 ),  
 Positioned(  
 top: 0,  
 right: 130,  
 child: Container(  
 width: 10,  
 height: 10,  
 color: Colors.red,  
 ),  
 ),  
 ],  
 )));  
}

需添加动画实现类

class SpinningContainer extends AnimatedWidget {  
 const SpinningContainer({Key key, AnimationController controller})  
 : super(key: key, listenable: controller);  
 Animation<double> get \_progress => listenable;  
 @override  
 Widget build(BuildContext context) {  
  
 return Transform.rotate(  
 angle: \_progress.value \* 2.0 ,  
 child: Container(width: 200.0, height: 200.0, color: Colors.green),  
 );  
 }  
}

## common\_utils常用工具

包含日期，正则，倒计时，定时任务，时间轴等工具类。

<https://pub.dev/packages/common_utils>

安装：

如果您是纯Dart项目，可以直接引用本库。

dependencies:

common\_utils: ^1.1.3 # 常用工具类库。

如果您是纯Dart项目，可以直接引用本库。

dependencies:

common\_utils: ^1.1.3 # 常用工具类库。

引入

import 'package:common\_utils/common\_utils.dart';

import 'package:flustars/flustars.dart';

## flutter\_screenutil 屏幕适配方案

<https://pub.dev/packages/flutter_screenutil>

安装：

dependencies:

flutter\_screenutil: ^0.6.1 # 屏幕适配

引入

import 'package:flutter\_screenutil/flutter\_screenutil.dart';

使用方法

ScreenUtil().setWidth(540) (sdk>=2.6 : 540.w) //根据屏幕宽度适配尺寸

ScreenUtil().setHeight(200) (sdk>=2.6 : 200.h) //根据屏幕高度适配尺寸

ScreenUtil().setSp(24) (sdk>=2.6 : 24.sp) //适配字体

ScreenUtil().setSp(24, allowFontScalingSelf: true) (sdk>=2.6 : 24.ssp) //适配字体(根据系统的“字体大小”辅助选项来进行缩放)

ScreenUtil.pixelRatio //设备的像素密度

ScreenUtil.screenWidth //设备宽度

ScreenUtil.screenHeight //设备高度

ScreenUtil.bottomBarHeight //底部安全区距离，适用于全面屏下面有按键的

ScreenUtil.statusBarHeight //状态栏高度 刘海屏会更高 单位px

ScreenUtil.textScaleFactor //系统字体缩放比例

ScreenUtil().scaleWidth // 实际宽度的dp与设计稿px的比例

ScreenUtil().scaleHeight // 实际高度的dp与设计稿px的比例

老版本方法

// 建议在第一次 build 的时候就进行初始化，这样接下来就可以放心使用了。

@override

Widget build(BuildContext context) {

// 方式一：默认设置宽度1080px，高度1920px

ScreenUtil.instance = ScreenUtil.getInstance()..init(context);

// 方式二：设置宽度750px，高度1334px

ScreenUtil.instance = ScreenUtil(width: 750, height: 1334)..init(context);

// 方式三：设置宽度750px，高度1334px，根据系统字体进行缩放

ScreenUtil.instance = ScreenUtil(width: 750, height: 1334, allowFontScaling: true)..init(context);

...

}

新版本方法

ScreenUtil.init(context);

ScreenUtil.init(context, width: 750, height: 1334);

ScreenUtil.init(context, width: 750, height: 1334, allowFontScaling: true);

具体实现

这里有个大坑，需要先新建一个StatelessWidget 且里边有 MaterialApp

后边的StatefulWidget 的 State 中再去执行初始化方法，后边就可以使用了

import 'package:flutter/material.dart';  
import 'package:flutter\_screenutil/flutter\_screenutil.dart';  
  
class MyApp extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
  
 return MaterialApp(  
 debugShowCheckedModeBanner: false,  
 title: 'Flutter\_ScreenUtil',  
 theme: ThemeData(  
 primarySwatch: Colors.blue,  
 ),  
 home: MyHomePage(),  
 );  
 }  
}  
  
class MyHomePage extends StatefulWidget {  
 MyHomePage({Key key, this.title}) : super(key: key);  
 final String title;  
 @override  
 \_MyHomePageState createState() => \_MyHomePageState();  
}  
  
class \_MyHomePageState extends State<MyHomePage> {  
 @override  
 Widget build(BuildContext context) {  
 //设置适配尺寸 (填入设计稿中设备的屏幕尺寸) 此处假如设计稿是按iPhone6的尺寸设计的(iPhone6 750\*1334)  
 ScreenUtil.init(context, width: 750, height: 1334, allowFontScaling: false);  
 return Container(  
 child: Column(  
 children: <Widget>[  
 Container(  
 color: Colors.red,  
 width: ScreenUtil().setWidth(80), // 这里改成80就可以看出原来的值  
 child: Text('1'),  
 )  
 ],  
 ),  
 );  
 }  
}