flutter_splash_screen

> flutter_splash_screen 0.3.0

用于显示隐藏 flutter 的 splash screen。

1.使用

1-1.Android添加:

// 这个包不行
//import org.devio.flutter.splashscreen.flutter_splash_screen.SplashScreen;

```
package org.devio.flutter.splashscreen.example;
import android.os.Bundle;

+ import org.devio.flutter.splashscreen.SplashScreen;

import io.flutter.app.FlutterActivity;
import io.flutter.plugins.GeneratedPluginRegistrant;

public class MainActivity extends FlutterActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
 +       SplashScreen.show(this, true);// here
        super.onCreate(savedInstanceState);
        GeneratedPluginRegistrant.registerWith(this);
    }
}
```

layout/launch_screen.xml
launch_screen 图片

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:src="@drawable/launch_screen" />
</RelativeLayout>
```

values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="primary_dark">#000000</color>
</resources>
```

values/styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Theme applied to the Android Window while the process is starting when
the OS's Dark Mode setting is off -->
    <style name="LaunchTheme" parent="@android:style/Theme.Light.NoTitleBar">
        <!-- Show a splash screen on the activity. Automatically removed when
             Flutter draws its first frame -->
        <item name="android:windowBackground">@drawable/launch_background</item>

        <!--添加-->
        <!--设置透明背景-->
        <item name="android:windowIsTranslucent">true</item>
    </style>
    <!-- Theme applied to the Android Window as soon as the process has started.
         This theme determines the color of the Android Window while your
         Flutter UI initializes, as well as behind your Flutter UI while its
         running.

         This Theme is only used starting with V2 of Flutter's Android embedding.
-->
    <style name="NormalTheme" parent="@android:style/Theme.Light.NoTitleBar">
        <item name="android:windowBackground">?android:colorBackground</item>
    </style>
</resources>
```

设置沉浸式状态栏

```
//Flutter沉浸式状态栏，Platform.isAndroid 判断是否是Android手机
if (Platform.isAndroid) {
  // setSystemUIOverlayStyle:用来设置状态栏顶部和底部样式
  SystemChrome.setSystemUIOverlayStyle(
      SystemUiOverlayStyle(statusBarColor: Colors.transparent));
}
```

异步UI更新--FutureBuilder

1.控件介绍作用

等数据返回值后，我们重绘页面返回另一个组件。类似于：

```
if(data==null){
    return CircularProgressIndicator();
}else{
    return ListView(...);
}
```

2.构造方法

```
const FutureBuilder({
  Key? key,
  this.future, // 获取数据的方法
  this.initialData, // 初始的默认数据
  required this.builder,
}) : assert(builder != null),
     super(key: key);
```

```
//FutureBuilder控件
new FutureBuilder<String>(
  future: _calculation, // 用户定义的需要异步执行的代码，类型为Future<String>或者null的变量或函数
  //snapshot就是_calculation在时间轴上执行过程的状态快照
  builder: (BuildContext context, AsyncSnapshot<String> snapshot) {
    switch (snapshot.connectionState) {
      //如果_calculation未执行则提示：请点击开始
      case ConnectionState.none: return new Text('Press button to start');
      //如果_calculation正在执行则提示：加载中
      case ConnectionState.waiting: return new Text('Awaiting result...');
      default:    //如果_calculation执行完毕
        if (snapshot.hasError)    //若_calculation执行出现异常
          return new Text('Error: ${snapshot.error}');
        else    //若_calculation执行正常完成
          return new Text('Result: ${snapshot.data}');
    }
  },
)
```

导航返回拦截--WillPopScope

为了避免用户误触返回按钮而导致APP退出，当用户在某一个时间段内点击两次时，才会认为用户是要退出（而非误触）。Flutter中可以通过WillPopScope来实现返回按钮拦截。

```
const WillPopScope({
  Key? key,
  required this.child,
  required this.onWillPop,
})
```

onWillPop是一个回调函数，当用户点击返回按钮时被调用（包括导航返回按钮及Android物理返回按钮）。该回调需要返回一个Future对象，如果返回的Future最终值为false时，则当前路由不出栈(不会返回)；最终值为true时，当前路由出栈退出。我们需要提供这个回调来决定是否退出。

1.示例

为了防止用户误触返回键退出，我们拦截返回事件。当用户在1秒内点击两次返回按钮时，则退出；如果间隔超过1秒则不退出，并重新记时。代码如下：

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

class WillPopScopeTestRoute extends StatefulWidget {
  @override
  WillPopScopeTestRouteState createState() {
    return new WillPopScopeTestRouteState();
  }
}

class WillPopScopeTestRouteState extends State<WillPopScopeTestRoute> {
  DateTime _lastPressedAt; //上次点击时间

  @override
  Widget build(BuildContext context) {
    return new WillPopScope(
        onWillPop: () async {
          if (_lastPressedAt == null ||
              DateTime.now().difference(_lastPressedAt) > Duration(seconds: 1))
{
            //两次点击间隔超过1秒则重新计时
            _lastPressedAt = DateTime.now();
            return false;
          }
          return true;
        },
        child: Container(
          alignment: Alignment.center,
          child: Text("1秒内连续按两次返回键退出"),
        )
    );
  }
}
```