flutter异常的捕获方式

大宝来巡山

App异常，就是应用代码的异常，通常由未处理应用层其他模块所抛出的异常引起。根据异常代码的执行时序，App异常可以分为两类，即同步异常和异步异常：同步异常可以通过try-catch机制捕获，异步异常则需要采用Future提供的catchError语句捕获

```
//使用try-catch捕获同步异常
try {
  throw StateError('This is a Dart exception.');
}
catch(e) {
  print(e);
}

//使用catchError捕获异步异常
Future.delayed(Duration(seconds: 1))
    .then((e) => throw StateError('This is a Dart exception in Future.'))
    .catchError((e)=>print(e));

//***注意，以下代码无法捕获异步异常***
try {
  Future.delayed(Duration(seconds: 1))
      .then((e) => throw StateError('This is a Dart exception in Future.'))
```

```
}
catch(e) {
  print("This line will never be executed. ");
}
```

需要注意的是，这两种方式是不能混用的。可以看到，在上面的代码中，**我们是无法使用 *try-catch* 去捕获一个异步调用所抛出的异常的。**

同步的try-catch和异步的catchError，为我们提供了直接捕获特定异常的能力，而如果我们想集中管理代码中的所有异常，Flutter也提供了 Zone.runZoned方法。

我们可以给代码执行对象指定一个Zone，在Dart中，Zone表示一个代码执行的环境范围，其概念类似沙盒，不同沙盒之间是互相隔离的。如果我们想要观察沙盒中代码执行出现的异常，沙盒提供了onError回调函数，拦截那些在代码执行对象中的未捕获异常。

在下面的代码中，我们将可能抛出异常的语句放置在了Zone里。可以看到，**在没有使用 *try-catch* 和 *catchError* 的情况下，无论是同步异常还是异步异常，都可以通过 *Zone* 直接捕获到：**

```
runZoned(() {
  //同步抛出异常
  throw StateError('This is a Dart exception.');
}, onError: (dynamic e, StackTrace stack) {
  print('Sync error caught by zone');
```

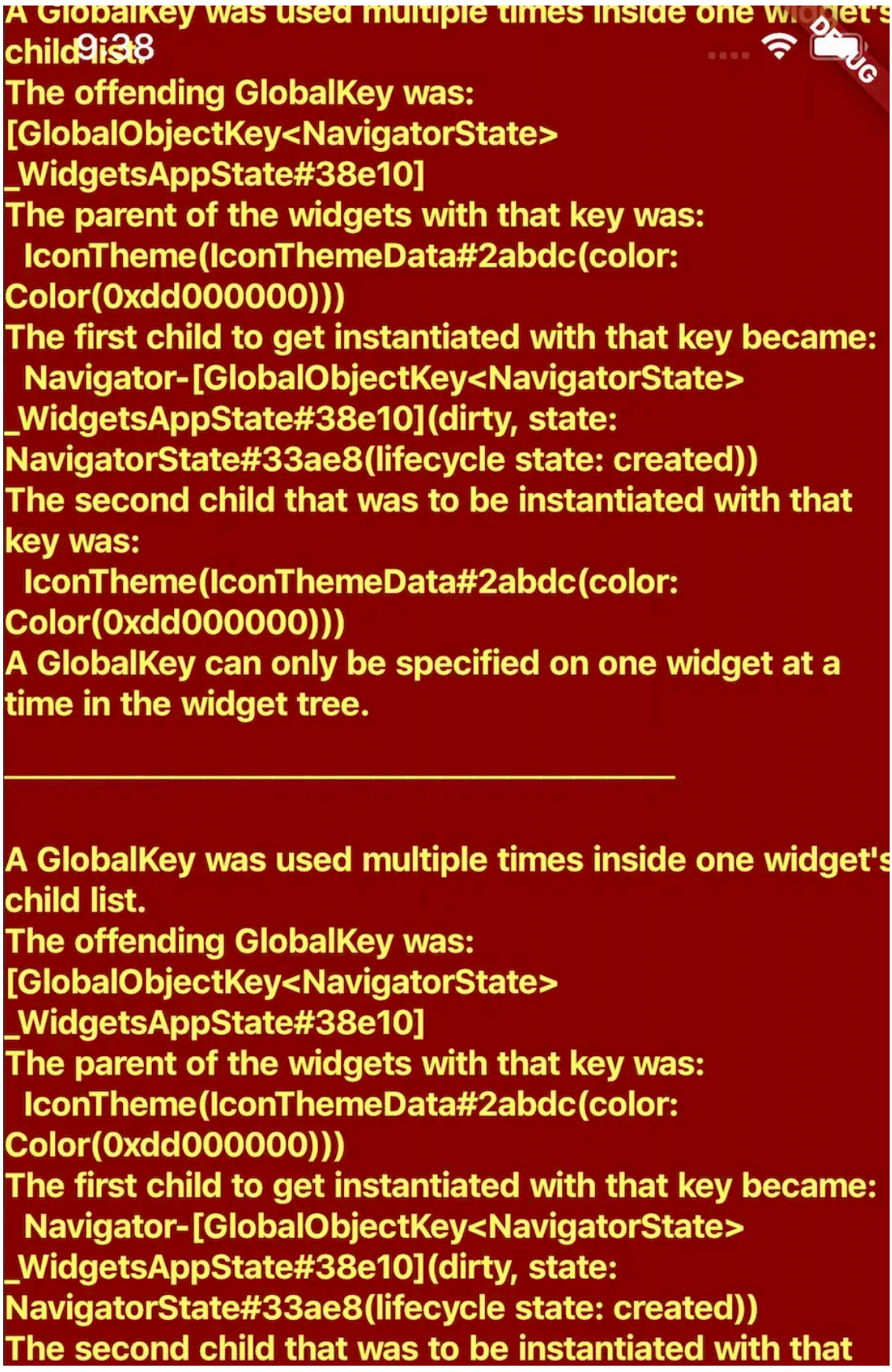
```
});

runZoned(() {
  //异步抛出异常
  Future.delayed(Duration(seconds: 1))
      .then((e) => throw StateError('This is a Dart exception in Future.'));
}, onError: (dynamic e, StackTrace stack) {
  print('Async error aught by zone');
});
```

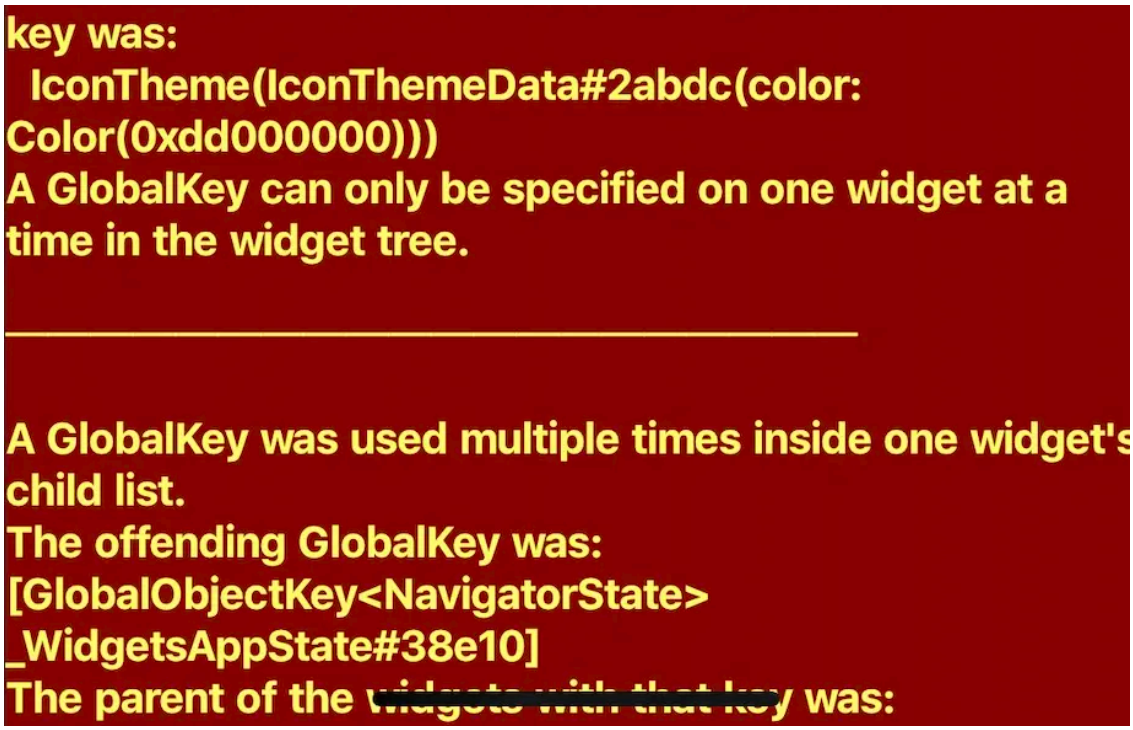
因此，如果我们想要集中捕获Flutter应用中的未处理异常，可以把main函数中的runApp语句也放置在Zone中。这样在检测到代码中运行异常时，我们就能根据获取到的异常上下文信息，进行统一处理了：

```
runZoned<Future<Null>>(() async {
  runApp(MyApp());
}, onError: (error, stackTrace) async {
 //Do sth for error
});
```

Framework异常的捕获方式

Framework异常，就是Flutter框架引发的异常，通常是由应用代码触发了Flutter框架底层的异常判断引起的。比如，当布局不合规范时，Flutter就会自动弹出一个触目惊心的红色错误界面，如下所示：


A GlobalKey was used multiple times inside one widget's
child list.
9:38
DEBUG
The offending GlobalKey was:
[GlobalObjectKey<NavigatorState>
_WidgetsAppState#38e10]
The parent of the widgets with that key was:
IconTheme(IconThemeData#2abdc(color:
Color(0xdd000000)))
The first child to get instantiated with that key became:
Navigator-[GlobalObjectKey<NavigatorState>
_WidgetsAppState#38e10](dirty, state:
NavigatorState#33ae8(lifecycle state: created))
The second child that was to be instantiated with that
key was:
IconTheme(IconThemeData#2abdc(color:
Color(0xdd000000)))
A GlobalKey can only be specified on one widget at a
time in the widget tree.
A GlobalKey was used multiple times inside one widget's
child list.
The offending GlobalKey was:
[GlobalObjectKey<NavigatorState>
_WidgetsAppState#38e10]
The parent of the widgets with that key was:
IconTheme(IconThemeData#2abdc(color:
Color(0xdd000000)))
The first child to get instantiated with that key became:
Navigator-[GlobalObjectKey<NavigatorState>
_WidgetsAppState#38e10](dirty, state:
NavigatorState#33ae8(lifecycle state: created))
The second child that was to be instantiated with that



1699431797925.jpg

这其实是因为，Flutter框架在调用build方法构建页面时进行了try-catch的处理，并提供了一个ErrorWidget，用于在出现异常时进行信息提示：

```
@override
void performRebuild() {
  Widget built;
  try {
    //创建页面
    built = build();
  } catch (e, stack) {
    //使用ErrorWidget创建页面
    built =
ErrorWidget.builder(_debugReportException(ErrorDescription(
"building $this"), e, stack));
    ...
  }
```

```
  ...
}
```

这个页面反馈的信息比较丰富，适合开发期定位问题。但如果让用户看到这样一个页面，就很糟糕了。因此，我们通常会重写 ErrorWidget.builder 方法，将这样的错误提示页面替换成一个更加友好的页面。

下面的代码演示了自定义错误页面的具体方法。在这个例子中，我们直接返回了一个居中的 Text 控件

```
ErrorWidget.builder = (FlutterErrorDetails
flutterErrorDetails){
  return Scaffold(
    body: Center(
      child: Text("Custom Error Widget"),
    )
  );
};
```

比起之前触目惊心的红色错误页面，白色主题的自定义页面看起来稍微友好些了。需要注意的是，ErrorWidget.builder 方法提供了一个参数 details 用于表示当前的错误上下文，为避免用户直接看到错误信息，这里我们并没有将它展示到界面上。但是，我们不能丢弃掉这样的异常信息，需要提供统一的异常处理机制，用于后续分析异常原因。

为了集中处理框架异常，Flutter 提供了 FlutterError 类，这个

类的 onError 属性会在接收到框架异常时执行相应的回调。因此，要实现自定义捕获逻辑，我们只要为它提供一个自定义的错误处理回调即可。

在下面的代码中，我们使用 Zone 提供的 handleUncaughtError 语句，将 Flutter 框架的异常统一转发到当前的 Zone 中，这样我们就可以统一使用 Zone 去处理应用内的所有异常了：

```
FlutterError.onError = (FlutterErrorDetails details) async
{
  //转发至Zone中
  Zone.current.handleUncaughtError(details.exception,
details.stack);
};

runZoned<Future<Null>>(() async {
  runApp(MyApp());
}, onError: (error, stackTrace) async {
 //Do sth for error
});
```

异常上报

到目前为止，我们已经捕获到了应用中所有的未处理异常。但如果只是把这些异常在控制台中打印出来还是没办法解决问题，我们还需要把它们上报到开发者能看到的地方，用于后续分析定位并解决问题。

关于开发者数据上报，目前市面上有很多优秀的第三方SDK服务厂商，比如友盟、Bugly，以及开源的Sentry等，而它们提供的功能和接入流程都是类似的。可以讲异常日志上报给第三方sdk厂商。