TODO:

1. **单线程模型不会因为未捕获的异常而崩溃?**
2. **我们可以创建第三个的任务队列吗?**
3. **加入队列的方式 Future.eventtask(...) ?**
4. **Zone： 这里，我有一个大胆的想法。类似于事件流是不是可以通过这种形式来处理?**

只有先了解Dart的代码执行流程，我们才能知道在什么地方去捕获异常。

Dart的单线程模型

2.6.1 Dart 单线程模型

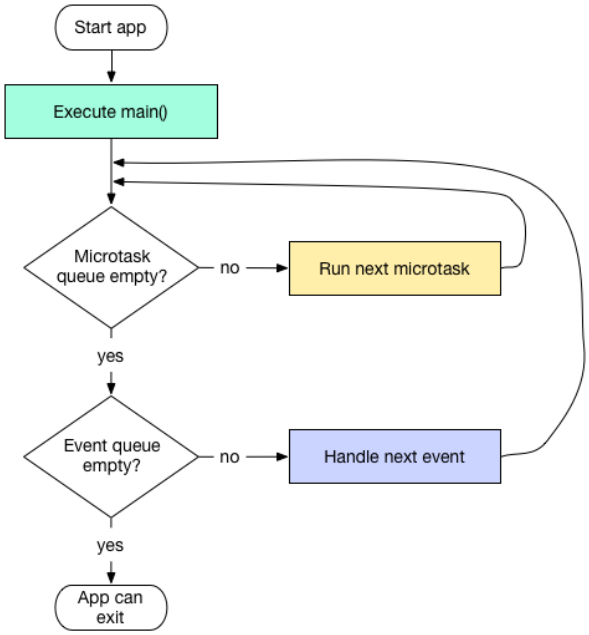
当我们使用Java或者 OC 时，如果发生异常且没有被捕获，程序将被终止。
但是Dart或者 JS不会有这个问题。
书中给出的理由是 Java 和OC 是多线程模式的编程语言，任意一个线程发生error，且没被捕获的情况下，就会导致整个进程退出。
但是Dart 和 JS并不会这样。原因是 Dart 和 JS 是单线程模型。

但为什么单线程模型不会因为未捕获的异常而崩溃，目前并不是很理解...
在事件循环中，当某个任务发生异常并没有被捕获时，程序并不会退出，而直接导致的结果是后续代码不会被执行。

即一个任务重的异常，不会影响其他任务的执行。

即跳过未执行的代码，直接开始执行下一个任务。

但为什么Java/OC 不行呢？这和单线程/多线程有什么关系呢？



两个任务队列相关知识点：

1. Dart 在单线程中是以消息循环机制来运行的，其中包含两个任务队列。
    a. 一个是“微任务队列” microtask queue.
    b. 一个是 “事件队列” event queue.
2. 启动时机
    a. 入口函数 main() 执行完之后，消息循环机制就启动了。
3. 功能
    b. microTask queue 通常处理来源于 Dart内部的事件，任务非常小。
    c. eventTask queue 处理 IO、 计时、点击、绘制事件等。
    d. 通常，我们会向 eventTask queue中添加任务，
4. 优先级
    e. microTask queue 要优先 eventQueue的消息队列。
    f. 这也是为什么 microTask queue 的任务量要小的原因。
5. 加入队列的方式
    g. Future.microtask(...)
    h. Future.eventtask(...) ?? 这个暂时不确定。

那么，问题来了，我们可以创建第三个的任务队列吗？

Flutter 框架异常捕获

try/catch/finally

注意：
上面说的Dart中，并没有帮我们捕获异常，如果发生异常了， 会跳过当前任务，继续执行下一个任务。
如果我们的Flutter 代码发生异常了，eg:我们开启了一个不存在的页面/组件。我的理解是，当前开启的事件出错了，当前事件剩下的代码就不会执行了。由该事件后续引发的其他事件发生改变，

Flutter 其实已经帮我们在很多关键地方做了异常捕获。eg:我们常见的红色报错页面，就是Flutter帮我们catch住了异常，并将异常信息作为单独页面的形式展示了出来。

那Flutter是在什么地方帮我们加上的呢?

是在我们执行Build方法的时候，添加的异常捕获。

```
void performBuild(){
    ...
    try {
        build = build();
    } catch (e, stack) {
        ErrorWidget.builder(_debugReportException('building $this'), e ,stack));
    }
    ...
}
```

之后，经过一系列追踪发现最终处理的位置：

```
_debugReportException
-> FlutterError.reportError(details);
-> FlutterError.onError(details);

# 我们在 main()方法的入口处，定义一个 reportError(details)的方法，在该方法中就能够完成日志的
void main(){
    FlutterError.onError = (FlutterErrorDetails details) {
        reportError(details);
    };
    ...
```

```
}
```

其他异常捕获与日志收集：

书中写到，Flutter中有些异常，Flutter是没有替我们捕获的，eg: 空对象方法异常、Flutter中的异常。
但是我在实操的时候，发现如果调用下面的方法，同样会跳转至 flutter error页面。

```
var temp = '';
temp.someMethod();
# 跳转至 flutter error页面。 这不是对该error 进行捕获了吗?
```

Dart 中，异常分为两类，同步异常，异步异常。
同步异常可以被 try/catch 捕获。 异步异常是不可以的。

```
try {
    Future.delayed(Duration(seconds:1)).then((e) => Future.error('xxx'));
}catch (e) {
    # 这里是捕获不了的。
    print(e)
}
```

我们可以通过Zone方法来捕获所有未能捕获的异常。
他可以类比成一个沙箱，不同沙箱之间是隔离的，沙箱可以捕获、拦截或者修改一些代码。
这里，我有一个大胆的想法。类似于事件流是不是可以通过这种形式来处理?

```
R runZoned<R>(R body(), {
    # 这里是map, key-value的形式。
    Map zoneValues,
    # 可以处理打印日志，可以处理和后面 onError 参数一样的错误处理。
    ZoneSpecification zoneSpecification,
    # 错误日志处理
    Function onError,
}
)
```

综上，我们捕获、处理和上报错误日志的代码如下，直接贴上书中的代码:

```
void collectLog(String line){
    ... //收集日志
```

```
}
void reportErrorAndLog(FlutterErrorDetails details){
    ... //上报错误和日志逻辑
}

FlutterErrorDetails makeDetails(Object obj, StackTrace stack){
    ...// 构建错误信息
}

void main() {
  FlutterError.onError = (FlutterErrorDetails details) {
    reportErrorAndLog(details);
  };
  runZoned(
    () => runApp(MyApp()),
    zoneSpecification: ZoneSpecification(
      print: (Zone self, ZoneDelegate parent, Zone zone, String line) {
        collectLog(line); // 收集日志
      },
    ),
    onError: (Object obj, StackTrace stack) {
      var details = makeDetails(obj, stack);
      reportErrorAndLog(details);
    },
  );
}
```