我们取消.env文件中doctrine传输队列前的注释，再打开messenger.yaml配置文件。在transports配置下，我们启用async传输队列，async传输队列使用的就是.env文件中的doctrine队列。注释第10行代码，我们将SendEmailMessage消息发送到async队列中。

```
#config/packages/messenger.yaml

framework:
    messenger:
        # Uncomment this (and the failed transport below) to send failed
messages to this transport for later handling.
        # failure_transport: failed

        transports:
            # https://symfony.com/doc/current/messenger.html#transport-
configuration
            async: '%env(MESSENGER_TRANSPORT_DSN)%'
            # failed: 'doctrine://default?queue_name=failed'
            # sync: 'sync://'

        routing:
            # Route your messages to the transports
            'App\Message\SendEmailMessage': async
```

回到浏览器，我们打开管理端添加一篇文章，点击文章列表。点击新增文章，文章的名称叫做Hello world abc，点击添加按钮。现在我们看到点击添加之后，页面就快速的进行了跳转，并没有再等待10秒钟。我们查看数据库，打开数据库客户端，刷新数据库表，自动添加的一个消息队列表。

我们查看表中的数据，在表中添加了一个消息，我们查看body的内容。在body中保存的就是我们SendEmailMessage对象，它的属性是postId，id为315。

回到浏览器，我们查看mailcatcher，仍然是之前的三封邮件，我们没有接收到新的邮件。回到项目，打开控制台，输入`symfony console`，查看所有的命令行。messenger组件在命令行中新增了三个命令行。我们可以使用`messenger:consume`命令行来对队列中的消息进行消费，也就是进行处理。

复制命令行，输入`symfony console`粘贴`messenger:consume`，现在messenger组件会从async队列中获取消息，并且进行消费。回到浏览器，我们的邮件成功的进行了发送。查看数据库，刷新messages表，我们表中的消息，被消费之后也自动的进行了删除。这就是异步消息的发布流程。

我们看命令行，命令行中提示我们可以使用-vv选项，来查看消息队列处理的日志。按着ctrl+c退出命令行，重新执行命令行 -vv。回到浏览器，我们再次新增一篇文章，hello 123，点击添加。

我们看到consume命令行，首先获取到了队列中的消息，然后调用Handler类的`__invoke()`方法，我们看到第二条日志显示之前，等待了10秒钟。然后显示我们的消息已经被Handler类进行了处理，最后向传输队列发送了一个确认消息，队列再将数据库中的数据进行了删除。

我们的mailcatcher，也收到了第五封邮件。在下节课，我们来了解一下当网络出现错误导致邮件发送失败时，消息队列的处理流程。