我们搜索symfony mail，查看第一个文档。要使用Symfony发送邮件，首先我们需要安装mailer组件。复制命令行，回到项目，粘贴。

```
composer require symfony/mailer
```

mailer组件安装完成之后，我们查看一下flex组件对系统的更改。首先是.env文件，我们查看这个文件，在.env文件下新增了一个配置MAILER_DSN。通过MAILER_DSN，我们可以配置邮件发送方的地址，然后我们查看配置文件。在配置文件中新增了mailer.yaml配置文件，yaml配置文件使用了.env文件中的环境变量。

回到浏览器，我们继续查看文档，这里提示我们需要配置邮件的发送地址。我们继续往下看，这里有段示例代码。在controller方法中，我们使用new Email()方法来创建一个email对象。然后使用from()方法指定邮件的发送地址，使用to()方法来指定邮件的接收地址。然后可以使用subject()方法，指定邮件的标题，使用text()或者html()来指定邮件的内容。最后我们使用$mailer对象的send()方法来发送邮件。

我们现在是开发环境，在开发时，我们不想发送一封真正的邮件。我们可以搜索mailcatcher，mailcatcher库可以在本机提供一个邮件发送的服务器和接收服务器。我这里已经安装过了mailcatcher，如果没有安装的话，你可以使用gem install命令行，来安装这个库。最后我们需要将mailcatcher库添加到环境变量中。

```
gem install mailcatcher
mailcatcher
Go to http://127.0.0.1:1080/
Send mail through smtp://127.0.0.1:1025
```

打开控制台，我这里就直接启用mailcatcher了，现在邮件的发送地址是127.0.0.1:1025端口。1080端口可以用来查看我们当前接收到的邮件，查看一下，现在还没有任何邮件。我们来修改.env文件。在.env文件中，我们来指定MAILER_DSN，这里输入smtp://127.0.0.1:1025端口。

在services.yaml配置文件中，我们可以定义邮件的发送地址和接收地址。首先我们来定义邮件的发送地址，send_mail: admin@teebblog.com。然后我们来定义编辑的邮件地址editor_email:，这里输入editor@teebblog.com。然后我们定义checker用户的邮件的地址。

打开PostCrudController，我们查看PostCrudController类的父类，在父类中有个new()方法，当我们点击添加按钮时，会调用new()方法。在new()方法中对新增的文章进行保存，这里有个表单提交的处理，在文章保存之前发送一个持久化的事件，然后在保存之后发送一个AfterEntityPersistedEvent事件。我们可以处理这个事件。

打开控制台，输入symfony console make:subscriber，Subscriber类的名称，我们叫做SendEmailSubscriber，我们需要订阅的事件名称需要复制一下，拷贝，粘贴。

打开我们新增的订阅器类，在新增的订阅器类中，我们首先来获取文章对象。$post = $event->getEntityInstance();，我们添加一个条件判断，if($post instanceof Post)，我们创建一个Email对象，$email等于new Email()，引入Email类。这里我们使用mime组件下的Email类。然后我们设置发送地址from()，发送地址我们需要通过ParameterBag对象来从配置文件中进行获取。

添加构造方法，在构造方法中我们注入ParameterBagInterface，创建一个属性，我们使用$parameterBag对象来获取发送地址，get()，粘贴send_email，然后我们指定接收(口误)地址，同样的我们使用$this->parameterBag->get('editor_email')。再获取checker_email。然后我们指定邮件的主题subject()，主题我们叫做有新的文章发布了，请检查。我们将文章的标题添加进来。使用$post对象getTitle()。然后我们再指定邮件的正文，text()，我们复制一下标题。最后结束。

这样我们就创建好了Email对象，我们还需要通过依赖注入的方式注入Mailer服务类，MailerInterface。初始化属性，使用$this->mailer对象 send()发送$email。

```
#src/EventSubscriber/SendEmailSubscrberSubscriber.php

class SendEmailSubscrberSubscriber implements EventSubscriberInterface
{
    /**
     * @var ParameterBagInterface
     */
    private ParameterBagInterface $parameterBag;
    /**
     * @var MailerInterface
     */
    private MailerInterface $mailer;

    public function __construct(ParameterBagInterface $parameterBag,
MailerInterface $mailer)
    {
        $this->parameterBag = $parameterBag;
        $this->mailer = $mailer;
    }

    public function onAfterEntityPersistedEvent(AfterEntityPersistedEvent
$event)
    {
        $post = $event->getEntityInstance();
        if ($post instanceof  Post)
        {
            $email = (new Email())
                ->from($this->parameterBag->get('send_email'))
                ->to($this->parameterBag->get('editor_email'), $this-
>parameterBag->get('checker_email'))
                ->subject('有新的文章<'.$post->getTitle().'>发布了，请检查。')
                ->text('有新的文章<'.$post->getTitle().'>发布了，请检查。');
sleep(10);
            $this->mailer->send($email);
        }
    }

    public static function getSubscribedEvents()
    {
        return [
            AfterEntityPersistedEvent::class =>
'onAfterEntityPersistedEvent',
        ];
```

```
        }
    }
```

回到浏览器，我们打开管理端，新增一篇文章，点击Hello world，点击添加按钮，我们看到mailcatcher接收到了一封邮件。发送者是admin，接收者是editor和checker用户，然后主题是新的文章已发布了请检查。

现在我们用的是本机的邮件服务器在发送邮件，所以不考虑网络情况。大多数是我们都会使用第三方的邮件平台发送邮件，有时候因为网络的延迟或者平台的问题，邮件就会发送失败。

回到项目，我们现在使用了订阅器来发送邮件 (Symfony的事件系统代码是同步的)，当出现网络延迟时，我们当前的页面就会发生卡顿，在发送邮件之前我们添加一段代码，来模拟网络延迟。`sleep()`这里我们暂停10秒钟。回到管理端，我们再次创建一篇文章，点击添加。我们看到现在页面一直在旋转，在10秒钟之后成功的发送了第二封邮件。

通常发送邮件发送短信等等这些比较耗时的网络任务，我们需要使用异步的方式来进行发送，这样就不会影响我们页面的流程了，在下节课我们来学习消息队列。