我们打开FileManagedType类，之前我们学习自定义表单类型的时候，我们学习了表单的事件。我们可以在表单提交的事件中对表单的数据进行处理，现在我们想过滤出评论正文中的html代码或者一些敏感字。我们可以在CommentType类中，创建一个监听器来监听表单的提交事件，这是一个方法。

Symfony还提供了事件系统，事件系统一方面可以解耦合代码，另一方面我们可以在一些操作的前后使用事件系统，然后来实现一些其他系统中钩子(hook)的效果。

回到浏览器，我们搜索`symfony event`，我们打开`EventDispatcher`文档，这里有创建和发送事件的文档。我们点击，首先我们要创建一个事件类，事件类可以是个普通的PHP类对象。对象中有一些属性和对应的getters和setters方法，然后使用`EventDispatcher`的`dispatch()`方法发送事件。最后我们可以创建事件订阅器或者监听器来接收事件对事件中的数据进行处理，最后再获取事件中的数据进行使用。

回到代码，我们在src目录下新增一个目录，目录名称叫做`Event`。在目录中我们新增一个类，类名我们叫作`AfterCommentSubmitEvent`。当前事件类是用于在表单提交后发送的事件，我们需要设置一个属性。当评论表单提交后，Symfony会将表单中的数据封装成一个Comment对象，我们将Comment对象添加到事件中，设置一个属性，我们设置一下$comment属性的类型。

我们为$comment属性添加一个get方法，鼠标点击$comment属性，按着alt键点击回车，添加getter方法，在构造函数中我们传入一个$comment对象，这样我们的事件类就创建好了。

```
#src/Event/AfterCommentSubmitEvent.php

class AfterCommentSubmitEvent
{
    private Comment $comment;

    public function __construct(Comment $comment)
    {
        $this->comment = $comment;
    }

    /**
     * @return Comment
     */
    public function getComment(): Comment
    {
        return $this->comment;
    }

}
```

在controller方法中我们来发送这个事件。打开PostController，在show()方法中，在表单提交之后，我们来发送事件。创建一个变量$event， `new AfterCommentSubmitEvent()`，传入参数就是我们表单提交的评论对象。然后我们需要使用`EventDispatcher`对象来发送这个事件，通过依赖注入的方式，我们注入`EventDispatcher`对象，这里类型我们输入`EventDispatcherInterface`，换行，使用$eventDispatcher对象发送事件，我们按着command键点击`dispatch()`方法。`dispatch()`方法的返回值是被监听器或者订阅器修改过的Event对象。

我们回到Controller添加加返回值，我们添加一个注释，我们可以获取事件中已经被修改过的Comment对象，$data等于`$modifiedEvent->getComment()`。然后再使用之前的代码对数据进行保存。我们已经创建了事件类，也已经发送了事件，我们还需要创建监听器来接收事件。在Listener目录中我们新增一个类，类名叫做CommentSubmitListener，我们可以在类中自定义一个方法用来接收事件，方法名称叫做`onCommentSubmit()`。传入一个参数，参数就是我们创建的事件对象，我们查看一下事件对象，输入`dd($event);`。

我们已经编写好了监听器类，但是现在的监听器类只是一个普通的PHP类，我们还需要对监听器类进行设置，将它注册到容器中，用来处理AfterCommentSubmit事件。回到浏览器我们往上查看文档，这里有段注册监听器和订阅器的说明。

我们查看一下代码，我们需要在容器中注册监听器类，添加`kernel.event_listener`标签。并且标签需要指定事件和处理事件的方法，回到项目，我们打开config目录。在services.yaml文件中，我们对Listener类进行配置，粘贴全类名，添加tags标签。tags是个数组，我们换一行添加一个大括号。我们要指定tags的name，这里输入`kernel.event_listener`， 然后设置要处理的事件类，我们复制刚刚创建的事件类类名，粘贴。然后指定处理事件的方法，复制Listener的方法，粘贴。

```
#config/services.yaml

# ...
services:
    # ...
    App\Listener\CommentSubmitListener:
        tags:
            - {name: 'kernel.event_listener', event:
App\Event\AfterCommentSubmitEvent, method: 'onCommentSubmit'}
```

回到浏览器，我们来提交一篇评论，点击提交，现在我们看到在表单提交后发送的事件，并且我们的监听器接收到了事件。我们可以在监听器中对事件的$comment属性进行修改，回到代码，打开CommentSubmitListener类。首先我们获取Comment对象，然后我们对Comment对象的$message属性进行过滤。使用`strip_tags()`进行过滤，并将过滤后的消息设置到$comment对象中。

```
#src/Listener/CommentSubmitListener.php

class CommentSubmitListener
{
    public function onCommentSubmit(AfterCommentSubmitEvent $event)
    {
        $comment = $event->getComment();
        $message = $comment->getMessage();
        $comment->setMessage(strip_tags($message));
    }
}
```

回到浏览器再次访问详情页，我们添加一个新的评论，这次评论消息的正文，我们添加一些html标签，点击提交。我们的评论，已经成功的提交了。我们检查一下评论的源码，我们刚刚提交的评论，它的h1标签已经被过滤掉了。

除了创建事件监听器，我们还可以创建事件订阅器来处理事件。我们查看文档，往下看，事件订阅器它实现了一个接口。在类中，首先它要设置监听的事件，并且指定处理事件的方法，然后定一个方法和我们的监听器写法很像，但是事件订阅器它的代码可复用性会更高一些。

在下节课，我们将使用事件订阅器能过滤评论消息中的一些敏感字。