继续查看文档，在文档中这里说明，可以注册一些事件监听器或者注册事件订阅器来处理表单的消息，事情的名称格式就是form.present_data这种格式，在FormEvents类中定义了事件名称的常量。

我们继续往下看，这里有个事件监听器的案例，首先使用$formFactory创建了一个FormBuilder，再使用FormBuilder的addEventListener()方法来添加一个事件监听器。第一个参数就是事件的名称，第二个参数是处理事件的回调方法。回调方法有个参数，参数是FormEvent对象实例。

在回调方法中可以用$event->getData()获取当前表单的数据，可以使用getForm()来获取当前表单对象。然后进行一些表单的数据的处理，我们继续往下看。在这个案例中，我们还可以将回调方法定义为类的一个方法来提高代码的可读性。

回到项目，打开FileManagedType，我们在buildForm()方法中，使用$builder对象的addEventListener()来添加表单事件监听器。事件的名称，我们使用FormEvents下的SUBMIT常量，我们在表单事件提交时处理数据，回调方法我们输入function，需要传一个参数。参数是FormEvent类型的对象，我们可以使用FormEvent对象来获取表单的数据。我们使用dd()方法来查看一下这个数据。

回到浏览器，回到详情页，刷新，我们手动的添加一个评论，上传一个图片文件，点击提交。在FileManagedType第18行，Symfony将我们表单上传的文件封装成了UploadedFile对象，我们可以将UploadedFile对象转换为FileManaged对象，再将FileManaged对象设置到表单中。

回到项目，我们打开PostController，复制一下代码，粘贴。现在$data是个UploadedFile对象，添加注释，修改变量名称为$file，出现了另外一个问题。

第23行，$this->getParameter()是Controller提供的一个内部方法，我们回到PostController，在Controller中我们按着command键点击getParameter()方法，在getParameter()方法中，通过容器获取parameter_bag服务类，通过get()方法来获取对应的参数。

在FileManagedType类中，我们可以直接在构造方法中通过依赖注入的方式直接使用ParameterBag对象。参数的类型，输入ParameterBagInterface，按着alt键点击回车初始化一个私有属性，这样我们就可以在FileManagedType类中使用$parameterBag对象了。

这里进行一下修改，注释后面的代码，输入$this->parameterBag->get('base_path')来获取文件的上传目录。回到PostController，我们在表单提交后查看一下表单的数据，第53行我们输入dd($data);。

```
#src/Form/FileManagedType.php

class FileManagedType extends AbstractType
{
    /**
     * @var ParameterBagInterface
     */
    private ParameterBagInterface $parameterBag;

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

    public function buildForm(FormBuilderInterface $builder, array
$options)
```

```
    {
        $builder->addEventListener(FormEvents::SUBMIT, function (FormEvent
$event){
            /**@var UploadedFile $data**/
            $file = $event->getData();
            $originName = $file->getClientOriginalName();
            $fileName = pathinfo(htmlspecialchars($originName),
PATHINFO_FILENAME) . '-' . $file->getFilename() . '.' . $file-
>getClientOriginalExtension();
            $uploadPath = $this->parameterBag->get('base_path');//$this-
>getParameter('base_path');
            $mimeType = $file->getMimeType();
            $filesize = $file->getSize();

            $file->move($uploadPath, $fileName);

            $fileManaged = new FileManaged();
            $fileManaged->setOriginName($originName);
            $fileManaged->setFileName($fileName);
            $fileManaged->setMimeType($mimeType);
            $fileManaged->setPath($uploadPath . '/' . $fileName);
            $fileManaged->setFileSize($filesize);

            $event->setData($fileManaged);
        });
    }

    // ...
}
```

回到浏览器，我们在文章下方的评论框中手动的添加一个评论，上传一个文件000.jpg，点击提交。我们查看files属性，files属性中仍然没有包含文件对象。

我们回到项目，打开CommentType类，我们之前设置files表单行，它的`mapped`配置设置为`false`。这样的话files表单行和我们Comment类的$files属性并没有创建映射关系，我们注释58行代码。

再次提交数据，我们再次查看files属性，files属性中已经有个元素了，元素就是我们刚刚创建的FileManaged对象。回到项目，打开`PostController`，我们注释53行代码，注释之前的文件处理代码，同样的也注释掉CommentController中的代码。

回到浏览器再次提交表单，现在我们的评论就已经成功提交了。我们查看一下数据库中的数据，刚刚我们上传的000.jpg它已经插入了第6条数据，在`comments_files`中搜索，也成功的插入了一条新的数据。

我们的自定义表单类型就已经完成了，回到文章详情页刷新。在评论列表中，我们之前提交的评论图片并没有显示。

在下节课，我们来处理评论列表中图片的显示。