我们打开底部控制台，输入`symfony console list make`，查看一下所有的make命令。在make命令中有一个`make:validator`，这个命令行可以帮助我们自定义一个验证器。复制命令行，输入`symfony console make:validator`，验证器的名称，我们叫做FileManagedValidator。命令行自动的在src目录下创建了两个类。

我们来查看一下这两个类，FileManaged是个注解，FileManagedValidator是个验证器。我们使用验证器类来对表单行的数据进行验证，我们查看注解类，注解类中只有一个$message属性，属性的内容就是验证出错时会回显的错误信息。这里我们修改一下，我们只允许用户上传图片类型的文件。这里修改为当前行只允许上传图片。

回到Validator类，我们查看一下validate()方法的两个参数。输入`dd($value, $constraint);`。我们打开Comment类，在Comment类的$files属性中，我们添加刚刚创建的注解，FileManaged。回到浏览器，在评论表单中提交一个评论，添加一个文件，点击提交按钮。我们自定义的验证器就已经生效了。

回到FileManagedValidator类，第一个参数就是我们提交的表单行的数据，它是一个ArrayCollection对象。第二个参数是验证器注解类的实例。回到代码，我可以遍历表单提交的数据，然后验证表单提交的数据是不是图片类型。

添加for循环`foreach($value as $file)`，`if`。如果$file是FileManaged类的实例，这里是Entity目录下的FileManaged类，那么我们获取$file对象的mimeType，获取到上传文件的mimeType之后。我们来判断他是不是图片类型的，mimeType字符串中间是用`/`进行分割的，我们获取`/`前面的字符串。

`if $prestr = strstr($mimeType, '/', true);`，第二个参数是`/`，我们要获取/前面的字符串，第三个参数传入true。我们判断字符串是不是image，if如果$prestr==='image'，直接返回退出方法。否则的话就会执行下面的代码，为验证器的消息设置参数，并且将错误消息添加到Violation中，在表单中进行显示。

回到浏览器刷新页面继续，现在我们看错误信息，当前行准许上传图片文件，我们想让错误信息显示在files表单行。我们修改代码，修改Comment类删除这个注解。在CommentType类中，我们为files表单行添加约束，我们验证的是表单行文件上传框，我们需要在`entry_options`中添加设置。输入new，这里使用Validator目录下的FileManaged类。

回到浏览器刷新，再次提交表单，现在提示出错了，我们回到验证器类，在Validator类中我们上传的文件并不是图片，它会执行下面的代码。在下面的代码中会为约束类的$message属性设置value参数，但是我们的消息属性中并没有value参数。我们可以注释掉这行代码。

回到浏览器刷新，现在错误信息就显示在了files表单行前，现在我们来上传一张图片来看看效果。我们选择001这张图片，点击提交，表单还是出错了。回到代码，在CommentType类中，我们将约束条件添加到了FileManagedType类上。所以会对FileManagedType表单行的数据进行验证，并不是对files表单行数据进行验证，在FileManagedValidator类中，我们在循环前查看一下$value值。

回到浏览器刷新，现在$value值是个FileManaged对象，并不是之前的ArrayCollection对象。回到代码，我们需要注释掉循环，这里条件判断，我们来验证一下$value值。如果$value值是FileManaged对象的实例，然后获取$value的getMimeType()。回到浏览器，我们刷新，现在评论就成功的提交了。查看列表，这就是我们刚刚提交的评论。

回到项目，查看Symfony的源码，是学习Symfony的最好方法，我们来查看一下Symfony的File验证器。搜索File，我们查看FileValidator这个类，在这个类中有对文件类型的验证。首先是验证器注解中有$mimeTypes属性，通过遍历注解的$mimeTypes属性来进行判断，这样可以增加代码的灵活性，我们来优化一下代码。

在FileManaged类中，我们来添加一个属性，叫做$mimeTypes。添加构造方法，我们在构造方法中添加一个参数，array $mimeTypes。在构造方法中为$mimeTypes属性设置值。回到验证器类，我们可以循环遍历当前验证器注解对象的$mimeTypes属性 as $allowMimeType。

如果$allowMimeType等于上传的文件的$mimeType，直接退出方法，回到FileValidator类。我们复制192行这段代码，回到FileManagedValidator类粘贴，我们判断如果允许的文件类型，它使用了通配符。那么我们获取/前的字符串，再判断我们当前上传文件类型的字符串是否和通配符前的字符串相同。如果相同的话就返回，否则的话就验证失败。

再回到CommentType类，我们来添加一个构造参数，添加数组，我们允许上传所有的图片，这里输入image/*。回到浏览器新增一条评论，我们上传PDF文件，点击提交，验证仍然出错了。这次我们上传一张图片，点击提交，现在评论提交就成功了。

我们可以再次优化代码，在FileManaged类中，我们来为message属性添加一个参数。使用双大括号，修改一下消息，当前文件，参数名称我们叫作name，不允许上传，只允许上传图片。回到Validator类，在Validator类中我们来设置参数，参数名称是双大括号name，name的值我们使用$value->getOriginName()来设置。

```
#src/Validator/FileManaged.php

/**
 * @Annotation
 */
class FileManaged extends Constraint
{
    public function __construct(array $mimeTypes, $options = null, array
$groups = null, $payload = null)
    {
        $this->mimeTypes = $mimeTypes;
        parent::__construct($options, $groups, $payload);
    }

    public $mimeTypes;
    /*
     * Any public properties become valid options for the annotation.
     * Then, use these in your validator class.
     */
    public $message = '当前文件{{ name }} 不允许上传，只允许上传图片.';
}
```

```
#src/Validator/FileManagedValidator.php

class FileManagedValidator extends ConstraintValidator
{
    public function validate($value, Constraint $constraint)
    {
//        dd($value, $constraint);
        /* @var $constraint \App\Validator\FileManaged */
```

```
        if (null === $value || '' === $value) {
            return;
        }
//dd($value);
//        foreach ($value as $file)
//        {
            if ($value instanceof \App\Entity\FileManaged){
                $mimeType = $value->getMimeType();

                foreach ($constraint->mimeTypes as $allowMimeType){
                    if ($allowMimeType === $mimeType){
                        return;
                    }

                    if ($discrete = strstr($allowMimeType, '/*', true)) {
                        if (strstr($mimeType, '/', true) === $discrete) {
                            return;
                        }
                    }

                }
//                if ($prestr = strstr($mimeType, '/', true)){
//                    if ($prestr === 'image'){
//                        return;
//                    }
//                }
            }
//        }

        // TODO: implement the validation here
        $this->context->buildViolation($constraint->message)
            ->setParameter('{{ name }}', $value->getOriginName())
            ->addViolation();
    }
}
```

回到浏览器，我们再次提交一条评论，这次我仍然选择一个PDF文件，点击提交。现在错误消息就按照我们的预期进行了显示，当用户在提交评论时，他可能在message消息体中添加了一些JS代码，这可能会引起安全问题。另外我们想过滤出评论中的一些敏感字符。

在下节课，我们将开始学习事件系统，我们使用事件系统来解决这个问题。