```
#本节课代码变更地址
https://github.com/teebbstudios/teebblog/commit/3f1fec8a923df669036d8a0012053575ee1ac8ed
```

首先我们来查看一下Api Platform关于操作的文档，这里我们往下看。当我们为类添加ApiResource注解之后，会自动的生成六个API，其中两个是集合操作，剩下的是单项操作。其中集合操作的get操作是必须有的，它用来检索资源的列表。单项操作的get操作也是必须有的，它用来对一个资源进行检索，另外还用来生成IRI信息。

我们继续往下看，回到项目，虽然我们在代码中只是简单的写了一下操作的名称，并没有进行配置，但是我们是可以为操作进行更多的设置的，这里对操作的方法进行了设置。

如果你要自定义操作，在自定义的操作名称后面可以设置需要的method，继续往下看，我们可以通过path设置来设置API的路径，另外还可以为API的路由参数进行设置。

继续往下看，如果我们要在API前添加一个统一的前缀，我们可以在ApiResource注解中添加`routePrefix`参数，这样在API地址前就会自动的添加这个前缀。

我们继续往下看，这里有个自定义操作`get_weather`，这里设置为`get_weather`使用GET方法，并且自定义了API的路径，还指定了当前的API所使用的Controller类，我们可以参考这个设置来生成一个文件上传的API。

文档的目录，继续往下拉，这里有一篇是关于文件上传的文档。我们查看一下，往下看。首先有个媒体对象类，然后在媒体对象类前指定了post操作的Controller类，然后设置了文档的一些内容往下看，在Controller类中我们需要实现__invoke()方法。

然后获取请求中上传的文件，再将文件保存到服务器上，生成一个MediaObject对象，再将MediaObject对象返回，这样文件上传的API就完成了。当然如果想对API的响应结果进行修改，我们可以自定义Normalizer类。

我们查看FileManaged类的接口，这里生成了六个接口，创建资源的接口是根据我们类中的属性自动设置的，我们需要对post操作进行修改。

回到项目，打开FileManaged类，添加注解构造方法，在构造方法中我们配置`collectionOperations`，数组这里我们输入get，post操作我们需要自定义。这里我们先留空，然后是item操作，item操作我们只保留get操作，我们来修改post操作。

当我们自定义操作时，如果你不对path进行设置，那么当前的操作会自动生成对应的操作路径，这里post操作将会自动的生成/api/file_manageds操作路径。首先我们来设置post操作的controller，它指向一个类我们叫做`ApiFileController::class`，然后我们设置method，这里使用post方法。

查看文档，这里我们也将`deserialize`设置为false，粘贴，下面是`openapi_context`。我们查看当前文档页面的源码，在script标签中有一段数据，这段数据就是我们当前文档的数据，当前的文档数据使用了openapi格式的数据，这些数据就是我们各个API的设置数据。

回到文档，我们复制`openapi_context`设置，回到项目粘贴，在`openapi_context`设置中，我们添加了requestBody，在content中设置了一个表单，表单的格式就是`multipart/form-data`格式。这里有个属性设置，这里file就是我们表单行的name，它的类型是string类型，格式化为binary。

回到浏览器，我们刷新文档页面，展开FileManaged post操作，现在我们看Request Body，它将上传一个file字段，表单的类型就是`multipart/form-data`。点击Try it out 之后，会让我们选择一个文件上传，如果你想学习更多的设置，你需要查看openapi文档。

回到项目我们已经设置好了post操作，现在我们来创建Controller类，打开Controller目录。新增一个类，类名叫做APIFileController，点击OK，Controller类继承于`AbstractController`，在Controller类中，我们要实现`__invoke()`方法。

我们注入Request对象，引入一下Request类，这里引入HttpFoundation下的Request类。修改FileManaged类，这里我们引入一下ApiFileController类。

回到文档，我们查看一下文档中的Controller类，$request对象，通过files属性来获取上传的文件，如果没有找到文件就抛出错误，然后将获取到的文件转换为一个对象，然后返回对象。

我们复制一下代码，粘贴，引入一下Exception类，通过$request对象获取到的$uploadedFile，Symfony已经进行了封装，这里我们直接添加一行注释UploadedFile类型。

首先我们生成一个新的文件名，`$uploadedFile->getClientOriginalName()`加下划线，然后我们生成一个随机的字符串，粘贴，这里使用sha1算法对上传的文件名称进行加密，然后截取加密后的字符串的前八位，最后我们添加文件的后缀`$uploadedFile->getClientOriginalExtension();`。

现在我们获取到了新的文件名称，使用new关键字来创建一个FileManaged对象。`$file = new FileManaged();`，然后我们设置$file的属性。`$file->setMimeType($uploadedFile->getMimeType());`，`$file->setOriginName($uploadedFile->getClientOriginalName());`，`$file->setFileName($newFileName);`，`$file->setFileSize($uploadedFile->getSize());`。然后我们设置文件的路径`$file->setPath();`，这里我就使用硬编码了`'/uploads/images/' . $newFileName`。

然后我们需要使用EntityManager对象将$file对象保存到数据库中，我们通过依赖注入的方式注入EntityManager对象，`$em->persist($file);`，`$em->flush();`。然后我们将上传的文件移动到指定的目录中，`$uploadedFile->move();`。

move()方法的第一个参数，我们要设置项目的绝对路径，我们之前在services.yaml配置文件中绑定了一个参数$projectDir。我们可以直接在`__invoke()`方法中注入这个参数。string $projectDir，这里目录名称我们输入`$projectDir . '/public/uploads/images'`。

我们还需要为move()方法添加第二个参数，就是$newFileName。新的文件名称，然后我们将$file对象返回`return $file;`。

回到浏览器刷新文档，展开post操作，点击Try it out，我们来上传一个文件，选择000.jpg，点击Execute。我们查看返回的结果，成功的返回了文件的数据。回到项目，我们查看文件上传目录，我们的文件成功的上传到了images目录中。

回到浏览器，文件上传后的数据中，我们需要生成文件的URL地址，回到项目。我们需要自定义一个Normalizer类，可以参考前面的课程，我们快速的定义一个Normalizer类。

在Normalizer文件夹中，我们新增一个PHP类，类名叫做`FileAwareNormalizer`，当前Normalizer要实现`NormalizerAwareInterface` 和`ContextAwareNormalizerInterface`。

实现接口的方法，添加方法，我们取消setNormalizer()方法，点击OK。在类中我们添加一个Trait，`use NormalizerAwareTrait;`。然后我们添加一个私有常量`private const`

FILE_NORMALIZER_ALREADY_CALLED。在supportsNormalization()方法中，我们添加条件判断 if(isset($context[self::FILE_NORMALIZER_ALREADY_CALLED]))，返回false return false;，否则的话 我们判断当前的$data数据是不是FileManaged类的一个实例。

在normalize()方法中，我们对$context参数进行设置，然后返回$this->normalizer->normalize($object, $format, $context);。在return之前，我们对$object数据进行设置，打开FileManaged类，在类中我们添加一个属性，private $fileUrl;。添加对应的get和set方法，ctrl+回车。点击OK，添加注释。

我们设置一下$object参数的类型FileManaged，这里我们修改为，$object->setFileUrl()。添加构造方法，注入RequestStack对象，alt加回车生成属性。这里输入$this->requestStack->getCurrentRequest()->getSchemeAndHttpHost()。直接使用FileManaged对象的$path属性getPath()，这样我们自定义的Normalizer类就完成了。

回到浏览器再次执行Execute，现在我们看到返回的数据中就增加了当前文件的URL地址，我们自定义的操作就完成了。

查看Api Platform文档，我们查看左侧的菜单，我们对菜单中的一些章节进行一下简单介绍。这里是Data Providers和Data Persisters Data Providers就是数据提供器，它是用来从数据库中查询数据的。然后是Data Persisters它是用来向数据库中插入数据的，我们可以在数据进行读取或插入的时候对数据进行更改。

然后是Filters我们前面介绍了几个filters，我们还可以自定义Filters。

继续往下看Validation和Security，Api Platform它使用了使用了symfony的validation组件，我们可以在类中添加注解，当我们使用API创建资源时，Api Platform就会对提交的数据进行验证。然后是Security，我们可以在ApiResource注解中对操作添加权限验证。

然后是Pagination，我们可以对集合资源进行检索时，对资源进行分页查询，因为课程篇幅的原因，Api Platform的课程就暂时讲到这里。

后期如果有可能的话，我会补充完Api Platform的所有课程。在下节课，我们将我们的项目进行打包发布到服务器上。