我们先来解决一下上节课获取文章数据时循环引用的错误，回到项目，我们当前的Comment类并没有添加ApiResource注解。我们往下看，在API列表下方定义了一些schema，我们查看一下Post.jsonld的schema。当我们获取文章数据时，数据的格式就是schema中定义的样式，我们查看一下comments值的定义。

这里comments的值，是Comment.jsonld的数组，我们展开Comment.jsonld。当我们获取文章数据时，就会获取当前文章的所有评论数据。当前评论的数据需要展开，在展开的时候用循环引用了当前的文章，这样就形成了循环引用。

我们看错误信息当前循环引用的限制为1，超出限制就会报错。回到项目，我们为Comment类添加API注解，同样的我们为FileManaged类添加注解。回到文档页面刷新，我们看到现在文档中有评论类和FileManaged类的接口。

我们往下看，我们查看Post.jsonld。现在我们看评论属性，评论属性的值变成了字符串数组，所有的字符串都是评论对象的`IRI`。往上看，我们再次获取所有的文章数据，点击Try it out，点击Execute，现在就没有错误了。

我们查看id为20的文章，评论的数据都以`IRI`的形式进行了显示，另外我们可以在文档页面使用接口来生成一篇文章。我们展开post操作的接口，在post接口中有一个Request Body，这里需要上传的就是文章的数据，数据的格式就是这里对应的schema格式，其中评论和作者都可以使用IRI字符串。

我们点击Try it out按钮，这里我们来修改一下标题，标题我们修改为Api Post，摘要我们输入Api Post，正文也输入Api Post。文章的状态，我们可以留空。然后是评论的数据，评论的数据我们留一个空数组。然后是封面图像，我们也留空。然后是文章的作者，文章的作者是IRI字符串，我们可以使用/api/users/1使用admin用户来作为文章的作者。

点击Execute按钮，我们查看响应结果，响应结果201。我们的响应已经成功了，创建了一篇文章。文章的IRI是/api/posts/22。我们可以查看一下博客的管理端，刷新文章列表，我们看到我们已经成功的使用API创建了一篇文章。

我们来查看ApiPlatform的文档，点击左边API组件的文档，我们往下看。这里有序列化过程的文档，我们点击查看，往下看。这里有张图是序列化与反序列化的过程，当我们使用get方法来获取资源时，首先资源对象会执行`normalize`转换为数组，再将数组进行encode来生成对应的格式，这就是序列化的过程。

当我们使用post方法来创建资源对象时，首先会将请求的数据执行decode生成数组，再将数组转换为对象，转换的过程叫做`denormalize`，这个过程就叫做反序列化。

我来搜索`symfony serializer`，查看文档，Api Platform使用了serializer组件。我们可以在`normalize`和`denormalize`的过程中，指定要序列化和反序列化的组用于获取或者更新特定的属性。

往下看，这里有一段示例代码，我们在注解中配置了normalize的组和denormalize的组。这样在进行序列化时，我们只可以获取Book对象的name属性，当我们在进行反序列化的过程中，我们需要添加name属性和author属性的数据，才能对Book数据进行反序列化。

回到项目，打开Post类，在ApiResource注解中。我们添加一行，`normalizationContext`，这里我们设置组groups，使用数组，组的名称我们叫做`post:read`。然后添加`denormalizationContext`，再次添加一个组groups，组的名称叫做`post:write`。

这里出现黄线了，我们需要调整一下顺序，在获取文章数据时，我们可以指定要序列化的属性。在$id属性前我们添加Groups注解。注解的参数这里我们输入`post:read`，然后对于文章的标题，我们也添加这个注解，文章的摘要，正文，然后是文章的封面，文章的评论和文章的作者，我们没有添加组。

回到文档刷新，我们查看Post-post.read schema，这就是根据我们刚刚配置的组自动生成的schema，在schema中就只显示了我们刚刚配置了组的属性。回到项目，我们注释这个get设置，然后添加get操作。

回到浏览器刷新，我们可以通过第三个API来获取文章的数据，点击Try it out，这里参数我们输入22，点击Execute。我们看到我们获取到的文章数据，并没有显示评论和作者信息，回到项目我们再配置反序列化的组，我们需要设置文章的标题，这里输入post:write。

复制一下组名，我们需要设置文章的摘要，设置文章的正文，文章的状态我们不设置，评论也不设置，封面图像我们也暂不设置，文章的作者我们需要设置。添加注解Groups。粘贴组名，回到浏览器刷新。我们查看Post-post.write schema。

如果我们需要生成一篇文章，我们就需要上传title数据、summary、body和作者数据。往上看，我们使用post接口来生成一篇文章，点击Try it out。现在我们就只需要修改这四个数据就可以了，标题输入Hello groups，摘要输入hello，正文输入hello，author输入/api/users/1，使用admin用户作为作者。点击Execute，这里响应201我们文章创建成功了。

回到项目往上看，当前我们在ApiResource的构造方法中，设置了denormalizationContext和normalizationContext。设置了序列化与反序列化的组。如果设置了这两个参数，那么当前的所有的API，在normalize与denormalize的过程中都会使用这两个设置。我们还可以在某个操作中设置组。

比如说在get操作中我们可以添加设置，get操作我们要获取数据，这里输入normalization_context，这里normalization_context要用小写字母加下划线的格式进行编写，不能使用驼峰的格式进行编写。在操作中配置的normalization_context或者denormalization_context，将会覆盖构造方法中的设置。然后指向一个数组，数组的键值我们输入groups，在groups中我们定义一个组的名称，这里使用post:item:get操作。

```
#src/Entity/Post.php

/**
 * @ORM\Entity(repositoryClass=PostRepository::class)
 * @ORM\HasLifecycleCallbacks
 */
#[ApiResource(
    collectionOperations: ['get', 'post'],
    itemOperations: [
        'get' => [
            'controller' => NotFoundAction::class,
            'read' => false,
            'output' => false,
        ],
    ]
            'normalization_context' => ['groups' => ['post:item:get']]
        ]
//        'get' => [
//            'controller' => NotFoundAction::class,
//            'read' => false,
//            'output' => false,
//        ],
    ],
    denormalizationContext: ['groups'=> ['post:write']],
    normalizationContext: ['groups'=> ['post:read']]
```

```
)]
class Post
{
    // ...
}
```

通常我们在操作中定义的组使用这种格式，首先是类小写，然后是item操作或者collection操作，然后是对应的操作名称，这里拼写错了groups。然后我们定义当前操作需要序列化的属性。复制组名，我们要序列化文章的$id粘贴，复制一下。我们要序列化文章的标题，我们要序列化文章的摘要，我们要序列化文章的正文，我们还可以序列化文章的作者。

回到浏览器刷新，我们查看下面的schema，增加了一项我们展开这个schema，如果我们使用Post类单项操作的get操作这个接口。我们将获取到下面的数据，其中文章的作者仍然是个IRI字符串。我们来测试一下，展开这里，点击Try it out，这里我们输入23点击Execute，我们看到成功的获取到了文章的数据。

这就是Api Platform中序列化与反序列化组的使用，我们看到当前的作者显示的是作者的IRI，通常我们在使用API希望获取文章数据时，同时获取作者的数据，在下节课我们讲解数据的嵌套显示。