我们来修改Post类，复制一下get操作的组名。往下看，我们想在get操作中获取文章的封面的图像数据。粘贴组名，回到浏览器刷新文档，我们使用get API，然后获取id为20的文章，这里输入20点击Execute。我们查看文章的数据，现在文章的封面图像只是图像的文件名。我们希望能够在返回数据中添加封面图像的url地址。查看Api Platform文档，我们可以在normalize过程中修改数据。

回到项目，打开控制台。搜索`symfony console list make`，这里有个`make:normalizer`命令行，我们复制命令行，输入`symfony console`粘贴命令行，我们来创建一个Normalizer类，类名我们叫做PostNormalizer。

打开PostNormalizer类，PostNormalizer类中有个normalize()方法，normalize()方法中将我们的文章对象进行规范化，最后的$data数据是个数组。我们可以在数组中添加封面图像的url地址，我们先来查看一下$data数据。

这里我们输入`dd($data);`，回到浏览器，我们复制请求地址，打开新的标签页粘贴，后缀我们输入.jsonld，访问。现在就是显示了我们的$data数据，$data数据就是我们当前的文章数据，我们可以修改$data数据，在$data数据中增加封面图像的url地址。

回到项目，输入$data，为数组添加一项post_image_url等于。$data。复制封面图像的名称，然后再添加前缀，首先我们要获取当前的主机地址，我们可以在构造方法中注入一个对象`RequestStack`，alt加回车初始化属性。我们可以用$requestStack对象来获取当前的请求对象`$request = $this->requestStack->getCurrentRequest();`，然后我们使用$request对象的`getSchemaAndHttpHost()`，来获取当前的主机地址 然后我们添加文件的上传目录地址，这里我就使用硬编码了`/uploads/images/`。

回到浏览器刷新，现在我们看到在数据中增加了一项`post_image_url`，url地址就是我们当前封面图像的地址，这就是自定义Normalizer在normalize过程中对数据的修改。如果我们想在denormalize过程中对数据进行修改，我们可以让Normalizer实现DenormalizerInterface这个接口。

回到项目，我们让PostNormalizer额外实现一个接口，`DenormalizerInterface`。实现接口中的方法，点击OK，在`supportsDenomalization()`方法中，我们对当前提交的数据进行检查，如果支持反序列化的话，就执行`denormalize()`方法，我们使用构造方法中的$normalizer对象来检查$data数据，`return $this->normalizer->supportsDenormalization()`。传入三个参数$data，$type，$format。然后在denormalize()方法中就可以对提交的数据进行denormalize。`$date = $this->normalizer->denormalize($data, $type, $format, $context);`我们查看一下denormalize之后$data数据`dd($data);`。

回到Post类，我们删除作者属性前post:write这个组，我们希望在创建文章资源时不用设置作者数据，让文章数据在执行denormalize()时，为文章对象设置作者。

回到浏览器刷新文档，我们查看post接口，现在只需要提交三个数据，我们复制这三个数据，我们使用postman来发送这个请求，这里使用post方法请求地址输入`127.0.0.1:8000/api/posts`。在Body这里我们添加请求的body，粘贴。

这里使用json格式，标题我们输入hello，摘要hello，正文hello，发送请求，点击preview。现在我们看到，我们提交的数据经过denormalize()之后，转换为了Post对象，我们可以为Post对象来设置文章的作者。

回到项目，添加一个条件判断`if( $date instancesof Post )`，这里我们添加`$data->setAuthor()`，我们现在没有作者对象，这里输个字符串，暂时代替一下，然后我们直接返回$data。

我们可以在构造方法中注入UserRepository对象，alt键加回车初始化属性，我们来获取admin用户，`$admin = $this->userRepository->findOneBy(['username'=> 'admin']);`。然后我们设置文章对象的

作者为$admin，我们再次提交数据，点击send。现在我们看到我们的文章成功的提交了。

```
#src/Serializer/Normalizer/PostNormalizer.php

class PostNormalizer implements NormalizerInterface,
DenormalizerInterface, CacheableSupportsMethodInterface
{
    private $normalizer;
    /**
     * @var RequestStack
     */
    private RequestStack $requestStack;
    /**
     * @var UserRepository
     */
    private UserRepository $userRepository;

    public function __construct(ObjectNormalizer $normalizer, RequestStack
$requestStack, UserRepository $userRepository)
    {
        $this->normalizer = $normalizer;
        $this->requestStack = $requestStack;
        $this->userRepository = $userRepository;
    }

    public function normalize($object, $format = null, array $context =
[]): array
    {
        $data = $this->normalizer->normalize($object, $format, $context);

        $request = $this->requestStack->getCurrentRequest();
        // Here: add, edit, or delete some data
        $data['post_image_url'] = $request->getSchemeAndHttpHost()
.'/uploads/images/'. $data['postImage'];
        return $data;
    }

    public function supportsNormalization($data, $format = null): bool
    {
        return $data instanceof \App\Entity\Post;
    }

    public function hasCacheableSupportsMethod(): bool
    {
        return true;
    }

    public function denormalize($data, string $type, string $format =
null, array $context = [])
    {
        $data = $this->normalizer->denormalize($data, $type, $format,
$context);
        if ($data instanceof Post)
```

```
        {
            $admin = $this->userRepository->findOneBy(['username'=>
'admin']);
            $data->setAuthor($admin);
        }
        return $data;
    }

    public function supportsDenormalization($data, string $type, string
$format = null)
    {
        return $this->normalizer->supportsDenormalization($data, $type,
$format);
    }
}
```

我们打开博客的管理端刷新列表，我们新增的文章，它的作者就是admin，我们可以通过这种方式在创建文章资源时自动的为文章设置作者。通常我们在使用API时都会为用户生成一个token，我们可以使用token来获取当前使用接口的用户，然后设置当前文章的作者。

这就是normalize和denormalize的过程，在下节课我们来修复自定义Normalizer时出现的问题。