在上节课，我们创建了PostNormalizer。在获取文章数据时添加了文章的封面图像链接，然后在创建文章资源时自动的为文章设置的文章作者。回到文档页面，我们使用第三个接口来获取id为20的文章，点击Execute。现在我们看到获取到的文章的数据，数据的格式是jsonld格式。但是数据中缺少了`@context`，`@id`和`@type`这三项。也就是数据的`IRI`信息不见了。

回到项目我们查看normalize()方法，在normalize()方法中使用$normalizer，将数据执行了normalize过程，现在的数据中就已经缺少了IRI信息。

我们看构造方法，在构造方法中我们注入的$normalizer对象，它是`ObjectNormalizer`类的实例。我们查看这个类，按着command键点`ObjectNormalizer`。我们看到这个类，它当前在symfony的serializer组件中，并不在我们的Api Platform组件中，所以我们的PostNormal并没有使用Api Platform提供的$normalizer。

我们查看Api Platform源码，打开vendor目录，展开api-platform，展开src目录，我们往下看，我们再展开Serializer文件夹。在Serializer文件夹中，Api Platform提供了normalizer，我们查看ItemNormalizer类。然后再查看父类，在父类中有个normalize()方法，在normalize()方法中对数据的IRI进行了设置，所以我们应该使用Api Platform提供的Normalizer。

打开控制台，我们来查看一下容器中所有的normalizer，输入`symfony console debug:container`，关键字我们输入normalizer。我们当前要将数据格式化为jsonld格式，搜索jsonld，找到两项。我们查看第27个服务类，编号输入27，当我们将数据格式化为jsonld格式时，将会使用Api Platform Jsonld命名空间下的ItemNormalizer。

关闭控制台回到PostNormalizer，如果我们在自定义的Normalizer类中，能够使用到Api Platform提供的Normalizer对象就可以了。我们再来查看Symfony提供的serializer组件。

在vendor目录中找到symfony目录，往下看，我们展开serializer目录。在serializer目录中有个Serializer类，我们打开这个类往上看，在Serializer的构造方法中。首先会对容器中的所有normalizer进行遍历，然后根据normalizer的类型来进行一些额外的设置。

我们在获取数据时，需要对Normalizer对象进行修改，我们可以让自定义的Normalizer类实现`NormalizerAwareInterface`接口。然后Serializer对象会自动的为我们的Normalizer类添加一个$normalizer，添加的$normalizer对象就是我们Serializer类对象本身。

我们继续往下看，当执行序列化时会调用Serializer类的normalize()方法。我们查看normalize()方法，在normalize()方法中，首先Serializer对象会根据数据、格式化的类型和$context数据，会从容器中获取最匹配的$normalizer对象，然后使用$normalize对象对数据执行normalize()。

回到构造方法，当我们让自定义的Normalizer实现`NormalizrAwareInterface`接口之后，我们自定义的Normalizer中就会有当前的Serializer对象。然后我们可以在Serializer执行normalize()之前将$data数据进行修改，然后继续使用Serializer对象执行normalize()方法。我们的数据将会按照预期进行输出。

回到PostNormalizer类，我们来修改一下代码，现在我们让PostNormalizer类只执行`denormalize()`过程。删除NormalizerInterface接口，注释`normalize()`和`supportsNormalization()`方法。

现在我们的PostNormalizer只执行denormalize(口误)过程，我们再新增一个类。在Normalizer文件夹中，我们新增一个PHP类，类名叫做`PostAwareNormalizer`，点击OK，我们让`PostAwareNormalizer`实现`NormalizerAwareInterface`。我们查看`NormalizerAwareInterface`，这只是一个单独的接口，这个接口仅用于对Normalizer类进行标记。

我们还需要让Normalizer类实现NormalizerInterface，这里我们实现另一个接口叫做ContextAwareNormalizerInterface，我们查看这个接口，这个接口继承了NormalizerInterface。

回到PostAwareNormalizer，我们添加接口的方法，setNormalizer()方法将会在Serializer类的构造方法中进行调用，我们可以在Normalizer类中定义一个属性$normalizer，然后在setNormalizer()方法中，我们设置$this->nromalizer = $normalizer;。

我们当前的$normalizer属性就是Serializer对象，这一步我们可以省略。我们使用一个Trait，use NormalizerAwareTrait; 。我们查看Trait代码，Trait代码中定义了一个属性，然后为属性进行了设置，删除重复的代码。

当前的Normalizer对象在执行normalize()方法之前，首先会执行supportsNormalization()方法，来判断当前的数据是否支持normalize，这里我们可以添加一个条件判断，return $data instanceof Post;。

如果当前的数据不是文章类型，那么就不执行normalize()过程，然后在normalize()方法中，我们使用当前的Serializer对象，对当前的数据执行normalize()，$this->normalizer执行normalize()。参数输入$object，$format，$context，然后将normalize()之后的数据进行返回return。

在Serializer对象执行normalize()之前，我们可以对$object数据进行修改，这里我们需要为文章添加封面图像URL，我们先来查看一下当前的$object数据。dd($object);。回到浏览器复制API地址，新建标签页添加后缀.jsonld，当前的$object参数，它是一个文章对象。

回到项目，我们可以在文章类中额外的添加一个属性。打开Post类，我们在类中添加一个私有属性，这个属性不需要配置ORM注解，叫做private $postImageUrl； 。然后为属性添加对应的get和set方法，按着ctrl键加回车，选择$postImageUrl，点击ok，这样就行了。

回到Normalizer类，在normalize()方法前，我们添加一下注释，我们设置一下$object类型，这里输入Post，修改一下代码，$object->setPostImageUrl()。添加构造方法，在构造方法中注入RequestStack对象，生成属性，然后获取当前的Request对象，使用$request对象获取当前的主机地址和协议名称，然后添加文件上传目录/uploads/images/。点$object->getPostImage()获取文件名。

回到浏览器再次访问这个地址，它出现了一个错误，回到项目，这里出现了循环调用。我们可以在类中添加一个私有常量，private const NORMALIZE_ALREADY_CALLED = 'normalize_already_called';，在第一次执行normalize()时，我们为的$context添加了一个设置$context[self::NORMALIZE_ALREADY_CALLED]等于true，在第二次执行normalize()时，我们添加一个条件判断 。if(isset($context[self::NORMALIZE_ALREADY_CALLED]))，直接返回false。

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

class PostAwareNormalizer implements NormalizerAwareInterface,
ContextAwareNormalizerInterface
{
    use NormalizerAwareTrait;

    /**
     * @var RequestStack
     */
    private RequestStack $requestStack;

    private const NORMALIZE_ALREADY_CALLED = 'normalize_already_called';
```

```
    public function __construct(RequestStack $requestStack)
    {
        $this->requestStack = $requestStack;
    }

    public function supportsNormalization($data, string $format = null,
array $context = [])
    {
        if (isset($context[self::NORMALIZE_ALREADY_CALLED])) {
            return false;
        }
        return $data instanceof Post;
    }

    /**
     * @param Post $object
     * @param string|null $format
     * @param array $context
     * @return array|\ArrayObject|bool|float|int|string|null
     * @throws ExceptionInterface
     */
    public function normalize($object, string $format = null, array
$context = [])
    {
        $context[self::NORMALIZE_ALREADY_CALLED] = true;

        $request = $this->requestStack->getCurrentRequest();

        $object->setPostImageUrl($request->getSchemeAndHttpHost() .
'/uploads/images/' . $object->getPostImage());
        return $this->normalizer->normalize($object, $format, $context);
    }
}
```

回到浏览器刷新，现在我们看到数据中IRI信息又回来了，但是数据中没有显示封面图像的URL，回到项目打开Post类，我们需要为$postImageUrl属性设置组，复制97行代码，粘贴。我们添加另外一个组post:read。

回到浏览器刷新，现在我们看到封面图像的URL就出现了，如果我们想要获取json格式的数据，数据将会以json格式进行显示。这样我们自定义Normalizer出现的问题就解决了，在下节课我们来学习Api Platform另一个功能过滤器。