doctrine内置了一个类可以对Query对象进行分页查询。回到浏览器，我们搜索doctrine paginator，我打开doctrine文档。文档上有段示例代码，我们查看一下示例代码。首先它定义一个DQL语句，然后使用$entityManager->createQuery()方法，创建一个Query对象，然后使用new Paginator()来生成一个Paginator对象。

Paginator的构造函数中我们需要传入一个Query对象，在代码第11行，我们可以通过count()方法来对$paginator的结果进行数量统计，也可以使用for循环来遍历$paginator中所有对象的数据。

回到项目，我们打开首页路由的controller方法。在index()方法中，我们为首页的模板传入了posts参数，posts参数就是使用PostRepository对象，通过findBy()方法获取到符合条件的所有文章。

我们想在首页实现分页功能，那么我们可以为模板文件传入一个Paginator对象，我们可以在PostRepository类中添加一个方法，通过方法来获取Paginator对象。

按着command键点击PostRepository，在代码下方，我们新增一个方法，getPostPaginator()方法。方法的返回值是个Paginator对象，这里我们选择Doctrine下的Paginator，我们使用new关键字来返回一个Paginator对象，return new Paginator();。

Paginator对象构造函数需要传入一个Query对象，Query对象我们可以使用前面课程讲到的，使用EntityManager的createQuery()方法来获取，或者使用createQueryBuilder()方法来创建一个QueryBuilder，再通过QueryBuilder的getQuery()方法获取，输入$this->createQueryBuilder()。

别名我们叫做p，我们进行一下查询，我们需要对首页的文章进行过滤。过滤条件首先是仅允许已发布状态的文章可以显示，我们添加andWhere()，过滤条件我们设置为p.status等于占位符。然后设置占位符参数setParameter()，第一个参数是status，第二个参数是占位符的值。status的值我们可以通过方法的参数来传入，然后我们需要对结果进行一下排序，orderBy()，p.id我们按照降序的方式进行排列。

分页查询的数据，我们要限制每页显示的数量，也要计算每页数据的起始偏移量。我们来限制一下每页显示的数量，setMaxResults()，这里我们传入一个参数。$limit的值，同样的我们使用方法的参数来传入，int类型的$limit，我们设置一个默认值为10。

我们可以通过QueryBuilder的setFirstResult()来设置查询的起始量，这里输入一个参数$offset，$offset的值，同样的使用参数的方式传入，最后我们通过getQuery()方法来获取一个Query对象。我们可以设置status有个默认值，设置为published，方法的单词我们拼写错了，我们修改为Paginator。

```
#src/Repository/PostRepository.php
class PostRepository extends ServiceEntityRepository
{
    public function getPostPaginator(int $offset,  int $limit = 10, string
$status='published'): Paginator
    {
        $query = $this->createQueryBuilder('p')
            ->andWhere('p.status = :status')
            ->setParameter('status', $status)
            ->orderBy('p.id', 'DESC')
            ->setMaxResults($limit)
            ->setFirstResult($offset)
            ->getQuery();

        return new Paginator($query);
```

```
    }
}
```

回到controller方法，我们通过PostRepository的`getPostPaginator()`方法来获取一个Paginator对象，方法的参数，我们现在需要传入一个偏移量，我们可以通过当前页的页码和每页的限制量来计算。

回到浏览器打开博客首页，页码我们可以通过在地址栏中添加query参数的方式来获取。回到项目，注入Request对象，我们通过Request对象的query属性，getInt()方法来获取页码，默认值我们设置为1，如果没有传page参数的话，默认是第1页。定一个变量$page，我们定义一个变量来限制每一页的数量，这里设置为10。

下面我们来计算一下偏移量，偏移量等于当前的页码-1，-1后的结果，再乘以每页的限制量，乘以$limit。在`getPostPaginator()`方法中，我们传入偏移量，注释第29行代码，paginator参数。我们设置为Paginator对象。

回到`index.html.twig`文件中，循环遍历这里我们要修改一下，修改post参数为paginator参数。回到浏览器，打开博客首页，现在首页就显示了十篇文章。我们打开管理端，查看文章列表，我们过滤所有已发布状态的文章，点击Filters，勾选一下，总共有11条数据。

我们回到博客首页，在后面添加page=2，第2页只有1条数据，我们的分页代码已经生效了。但是我们页面底部的分页器按钮并没有办法使用，回到项目我们把当前的页码传入模板中。修改模板代码，在上一页和下一页链接这里，我们修改链接地址，使用path()方法。

路由的名称我们输入`post_index`，我们为路由传递page参数，参数的值就是当前的页码减1，我们需要在按钮的外面添加一个条件判断，如果当前的页码-1大于0，我们显示上一按钮，endif。

在下一页这里，同样的我们使用`path('post_index')`，页码参数，我们这里输入page+1，当然我们要在按钮外面添加一个条件判断`{% if page + 1 < max_page %}`，结束判断。

`max_page`我们计算当前内容一共要显示多少页。回到PostController类中，在$paginator代码下方我们定一个变量$max_page，等于，我们通过Paginator的count()方法获取所有的文章数量，除以每页显示的数量$limit。然后我们再使用ceil()方法，CEIL，来向上取整，我们将$max_page传入模板中。

回到浏览器刷新，在第2页就显示了上一页按钮，第1页没有显示下一页按钮，我们修改一下模板代码，模板的page参数最小就是1。我们需要让它小于等于最大页码，回到浏览器刷新。

我们还需要修改一下样式类，我们删除disabled样式类。刷新，现在按钮就可以使用了，回到controller方法，我们可以优化一下代码，页面显示的数量，这里使用了硬编码。我们可以在conflg的`services.yaml`文件中定义一个参数，再通过Controller的`$this->getParameter()`方法来获取每一页的显示数量，然后将$limit参数传入`getPostPaginator()`方法中。

```
#config/services.yaml

parameters:
    # ...
    page_limit: 5
```

回到浏览器刷新，现在出错了，我们修改一下代码，我们进入getPostPaginator()方法，调整一下$limit参数和$status参数的顺序。

```
#src/Controller/PostController.php

class PostController extends AbstractController
{
    public function index(Request $request, PostRepository
$postRepository): Response
    {
        $page = $request->query->getInt('page', 1);
        $limit = $this->getParameter('page_limit');
        $offset = ($page - 1) * $limit;
        $paginator = $postRepository->getPostPaginator($offset, $limit);
        $max_page = ceil($paginator->count() / $limit);
        return $this->render('post/index.html.twig', [
            'max_page' => $max_page,
            'paginator' => $paginator,
            'page' => $page
//            'posts' => $postRepository->findBy(['status' =>
'published'], ['id' => 'DESC']),
        ]);
    }
}
```

回到浏览器刷新，现在页面数据就可以正常显示了，我们来修改一下页面数量的限制。刷新，在第2页同时出现了上一页和下一页按钮，在第1页出现了下一页按钮，在第3页出现了上一页按钮，现在文章列表的分页就已经完成了。

但是我们编写的代码花费了很长时间，在下节课我们可以使用第三方的包来快速的开发分页功能。