在学习QueryBuilder之前，我们先来学习一下DQL。我们打开浏览器，搜索doctrine DQL，打开doctrine文档，DQL是doctrine查询语言，它是一个对象查询语言，和Hibernate HQL和Java Persistence的JPQL语言很像。

我们往下看，这有一个很简单的查询示例，EntityManager的createQuery()方法中传入了一段很像SQL语句的字符串，但是字符串中FROM关键字后面跟的并不是表名，它跟的是一个Model类的全类名，类名后面是这个Model类的别名。SELECT查询别名就是查询这个Model类的所有属性，查询条件是u.age大于20，createQuery()方法最终将返回一个Query对象。

Query对象再通过getResult()方法来执行这个查询，最终返回所有查询的结果，查询的结果是Model类的对象集合。这段字符串就是DQL语言，它和SQL语句很像，你可以使用SQL中的很多关键字来编写自己的DQL语言，但是我们手写DQL语言时，难免会出现一些拼写错误。所以doctrine提供了一个QueryBuilder对象 可以通过编程的方式来帮助我们创建DQL语句。

我们往下查看文档，这里说通过EntityManager对象的createQueryBuilder()方法，就可以创建一个QueryBuilder对象，然后再往下QueryBuilder对DQL语言中的关键字进行了一些封装。比如说SELECT关键字，这里可以使用QueryBuilder的select()方法，还有from()方法等等等等。

我们回到项目，在EntityManagerTest类中，我们添加一个测试方法，叫做testQueryBuilder()，不需要返回值，void。

首先我们在数据库中插入一些数据，把上面的插入数据代码复制过来。我们使用EntityManager来创建一个QueryBuilder对，然后使用QueryBuilder对象来查询数据库中所有的Post文章，select()方法参数是个别名，这里我们输入p，它会查询文章的所有属性，from，from()方法第一个参数就是Model类的全类名，这里我们使用Post::class。第二个参数就是别名，我们输入p，我们不添加where查询条件。QueryBuilder最终要返回一个Query对象，然后使用Query对象的getResult()方法来获取所有的数据。

它的返回结果是Post对象的集合，这里我们下断言，断言数据库中现在有4篇文章。我们执行testQueryBuilder()方法，打开底部控制台，输入php bin/phpunit，我们只想执行textQueryBuilder()方法，在后面添加--filter选项，选项值就是方法的名称。执行测试，测试通过了。

我们在96行输入dd()查看一下$posts变量，再次执行测试代码，$posts是个数组，就是我们刚刚插入的4篇文章。删除96行代码，下面我们来修改PostRepository类中的方法，我们取消26行前后的注释。

我们希望能够在PostRepository服务类中有一个方法，能够对文章的发布状态进行过滤，我要修改方法名称为findByStatus()，这个方法有一个参数，这个参数就是我们需要过滤的文章状态。第27行，Repository类的父类为我们提供了一个createQueryBuilder()方法，可以帮助我们快速的创建一个QueryBuiler对象。

我们按着command键进入这个方法，这个方法的内部和我们测试代码中编写的代码类似，它也是通过EntityManager对象来创建一个QueryBuilder对象。然后查询对应的Model类，这里使用andWhere()来进行条件查询，我们查询p.status等于:value，:value就是命名的占位符参数，我们可以在第29行使用setParameter()方法对占位符参数传值，然后第30行使用orderBy()方法对查询到的结果进行排序，第31行可以设置最大的结果数量，我们只查询查询，查询结果的前十条，第32行通过getQuery()来获取到Query对象。最终，第33行获取到结果。

```
class PostRepository extends ServiceEntityRepository
{
    // ...
```

```
    /**
     * @return Post[] Returns an array of Post objects
     */
    public function findByStatus(string $value)
    {
        return $this->createQueryBuilder('p')
            ->andWhere('p.status = :value')
            ->setParameter('value', $value)
            ->orderBy('p.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult();
    }
    // ...
}
```

回到测试代码，我们要使用到PostRepository这个服务类的对象，我们复制前面的代码，然后通过我们刚刚编写的findByStatus()方法来对文章的状态进行过滤。这里我们参数输入published，返回的结果是数组类型，这里我们断言返回的数据只有一条，执行测试代码，测试通过了。

我们再次回到PostRepository类中，我们想对文章的标题进行模糊查询，我们创建一个查询方法`findByTitle()`，需要传递一个参数，参数就是我们要查询的标题关键字。

首先我们要创建一个QueryBuilder对象，来使用$this->createQueryBuilder()，为QueryBuilder对象添加查询条件，我们使用andWhere()，p.title LIKE后面我们输入占位符。再设置占位符的值，这里输入'%'.$keywords.'%'，然后排序方法我们仍然使用升序的方法，最大结果数量我们选择前十条，我们复制30行到33行的代码，在测试代码中我们来测试findByTitle()，关键字我们输入title，输入dd()查看一下返回到的结果。

```
class PostRepository extends ServiceEntityRepository
{
    // ...

    public function findByTitle(string $keywords)
    {
        $qb = $this->createQueryBuilder('p');
        return $qb->andWhere('p.title LIKE :keywords')
            ->setParameter('keywords', '%' . $keywords . '%')
            ->orderBy('p.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult();
    }

    // ...
}
```

再次执行测试代码，测试代码显示值为null，我们查看一下findByTitle()方法，我们忘了将结果返回了，这里输入return，将结果返回，然后再次执行测试代码。这次测试代码显示了4条，就是我们测试代码中插入的4条数据，现在我们修改一下标题的查询条件，关键字这里我们输入02，再次执行测试，这次结果只有1条，就是我们刚刚插入的第2条数据。

我们下个断言，断言返回结果就只有1条，并且断言结果的标题，就是Post title 02，再次执行测试代码，测试通过了。

```
class EntityManagerTest extends KernelTestCase
{
    // ...

    public function testQueryBuilder(): void
    {
        $factory = static::getContainer()->get(PostFactory::class);

        $post1 = $factory->create('Post title 01', 'Post Body 01');
        $post2 = $factory->create('Post title 02', 'Post Body 02', null,
'published');
        $post3 = $factory->create('Post title 03', 'Post Body 03');
        $post4 = $factory->create('Post title 04', 'Post Body 04');
        $this->entitymanager->persist($post1);
        $this->entitymanager->persist($post2);
        $this->entitymanager->persist($post3);
        $this->entitymanager->persist($post4);

        $this->entitymanager->flush();

        $qb = $this->entitymanager->createQueryBuilder();
        $posts = $qb->select('p')->from(Post::class, 'p')
            ->getQuery()->getResult();

        $this->assertCount(4, $posts);

        $postRepo = static::getContainer()->get(PostRepository::class);

        $posts = $postRepo->findByStatus('published');
        $this->assertCount(1, $posts);

        $findByTitlePosts = $postRepo->findByTitle('02');
        $this->assertCount(1, $findByTitlePosts);
        $this->assertSame('Post title 02', $findByTitlePosts[0]-
>getTitle());

        $findByTitlePostsDQL = $postRepo->findByTitleDQL('03');
        $this->assertCount(1, $findByTitlePostsDQL);
        $this->assertSame('Post title 03', $findByTitlePostsDQL[0]-
>getTitle());
    }

    // ...
}
```

我们回看PostRepository的findByTitle()方法，我们在andWhere()中使用了LIKE关键词，我们还有另外一种方式来添加条件查询，我们注释39行代码 返回值仍然是$qb。

QueryBuilder添加andWhere()查询条件，QueryBuilder有一个expr()方法，expr()方法的返回对象，提供了like()方法，like()方法可以执行模糊查询，like()方法第1个参数就是p.title，第2个参数就是我们的占位符名称，再次执行测试代码，测试代码依然通过了。

```
class PostRepository extends ServiceEntityRepository
{
    // ...

    public function findByTitle(string $keywords)
    {
        $qb = $this->createQueryBuilder('p');
        return $qb->andWhere($qb->expr()->like('p.title', ':keywords'))
            ->setParameter('keywords', '%' . $keywords . '%')
            ->orderBy('p.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult();
    }

    // ...
}
```

expr()的like()方法，它对LIKE关键字进行了封装，我们可以使用like()方法(口误)来减少代码拼写时出现的错误。我们按着command键点击getQuery()方法，我们进入QueryBuilder的getQuery()方法，在getQuery()方法中，我们编写的QueryBuilder查询方法最终都将变成DQL语句，然后进行查询。

在下节课，我们将手动编写DQL语句来进行结果的查询。