我们希望在测试代码中可以操作数据库，Symfony的orm组件，为我们提供了一个服务类EntityManager。我们双击shift键，搜索EntityManager，如果你没有搜索到EntityManager的话，你需要勾选这个选项。

我们打开ORM框架下的EntityManager，EntityManager提供了一些Entity对象的操作方法，来操作数据库中Entity对象的数据。比如说我们想将Post对象保存到数据库中，可以使用persist()方法，persist()方法它的参数是一个Entity对象。如果你想删除一个对象对应的数据库表中的数据，你可以使用``emove()方法，最后使用flush()`方法将数据库的操作进行提交。

我们之前课程讲过如何使用仿件对象来对服务类进行测试，但是现在我们要真正的操作数据库，仿件对象并不能操作数据库，我们只能预测仿件对象的行为。这样我们就需要将服务类集成到我们的测试代码中，来真正的调用服务类的方法实现数据库的操作，这就是集成测试。

在编写测试代码之前，我们将之前写的单元测试代码调整一下目录，在tests目录中新建一个文件夹叫UnitTest，将我们之前写的单元测试代码移动到UnitTest目录中。修改测试代码的命名空间，再次运行单元测试命令，测试依旧通过了，现在我们需要创建一个集成测试。

打开底部控制台，输入symfony console make:test，我们之前讲过KernelTestCase就是用于集成测试的。我们可以在测试代码中使用Symfony的服务类，我们输入KernelTestCase，我们在tests目录中添加一个目录，用来保存集成测试的代码IntegrationTest。

测试类的类名前，我们添加目录，叫做EntityManagerTest。这样就在我们对应的目录下添加了一个测试类，我们打开测试类，我们看到集成测试的测试类，它继承于KernelTestCase。在示例代码中首先集成测试会启动内核，我们前面课程讲过，Symfony是个MVC框架，它实现了PSR标准规范的容器接口。Symfony项目的所有服务类最后都将存在于容器中。

我们查看一下bootKernel()它的代码，按着command键鼠标点击bootKernel()方法，在bootKernel()方法第80行，它首先确认内核没有启动，然后创建一个Kernel，然后Kernel开始启动，我们查看boot()方法。

按着command点击boot()，它进到了KernelInterface接口中，我们查看http-kernel这个文件夹，文件夹中有个Kernel类，点开Kernel类，在Kernel类中有个boot()方法。

我们查看boot()方法，在boot()方法中，第124行如果容器为空，它将会进入preBoot()方法。在preBoot()方法中初始化我们安装的所有bundle，并且初始化容器。

再回看boot()方法，第128行循环遍历我们安装的所有bundle，并且将bundle中的所有服务类设置到容器中，再启动bundle，所以在Kernel启动时，它会事先读取所有的bundle，并且准备容器，将所有的服务类添加到容器中，这时我们就可以通过容器对象，来获取容器中对应的服务类了。

第14行有段注释的代码，通过$container的get()方法可以获取到对应的服务类，我们需要在测试代码中获取到EntityManager服务类。EntityManager服务类，它是EntityManagerInterface一个实现。Symfony提供了一个命令行可以查看容器中的所有可用的服务。 我们打开底部控制台，输入symfony console debug:container，回车，这就列出了容器中的所有服务，左边是服务的别名(ID)，右边是服务对应的类名，我们搜索一下EntityManager按command+f搜索。回车。

我们找到了一个EntityManager类名，左边是服务的别名，但是别名后面添加一个abstract，它是一个抽象的。在上一行。我们找到了doctrine.orm.entity_manager这个服务类的别名(ID)。在右边，这是doctrine.orm.default_entity_manager的别名。我们将使用doctrine.orm.entity_manager这个别名(ID)，来获取到EntityManager服务类对象。

修改14行代码，get()方法的参数，id就是服务的别名，这里修改变量名称，我们断言$entitymanager对象他是EntityManagerInterface的一个实例。再修改测试的方法名称，修改为testEntityManager()。

我们看到这一行$container变量已经被弃用了，$container变量是KernelTestCase提供的一个静态变量，它弃用的说明是从Symfony5.3版本开始使用static::getContainer()方法来代替。我们修改代码。

打开底部控制台测试代码，php bin/phpunit测试通过了。

在前面的课程，我们编写了PostFactory作为服务类，那么我们编写的PostFactory服务类有没有自动添加到$container容器中呢?

我们打开config目录下的services.yaml配置文件。在services配置下，第14行，我们看这个注释。注释说在src下所有的类都被用作服务类，然后第16、17行，这个配置就是说在src目录下的所有类都可以称为服务类。但是第18行排除了一些目录，比如说src下的Entity目录中的类就不是服务类，Kernel.php也不是服务类，我们使用debug:container命令行。查看一下我们编写的PostFactory服务类有没有添加到容器中。

打开底部控制台，输入symfony console debug:container，在命令行后面添加PostFactory，会检索容器中PostFactory关键字，找到相关的服务类。回车，我们找到了这个服务类，服务类的ID就是我们的PostFactory类名。复制，回到测试代码。我们要获取到容器中的PostFactory对象。然后用PostFactory对象创建几篇文章，最后使用EntityManager，把我们的文章存入到数据库中。

编写代码，get服务的ID就是我们的类名，这里可以使用PostFactory::class，我们断言一下$factory对象就是PostFactory的一个实例，执行测试代码，出错了。它提示我们没有找到PostFactory服务类对象，错误的原因是当我们容器进行编译的时候，这个服务类可能被移除了，或者被内联了。我们可以让这个服务类变为公开的，或者通过依赖注入的方式来使用这个服务类对象。

这该怎么理解呢?

我们再次查看容器中的PostFactory服务类，我们的PostFactory有个public选项，它现在是私有的，不是公开的。在src目录下，我们编写的所有服务类，默认它都是私有的。私有的服务类它不能通过容器的方式来获取到服务类对象，但是它可以通过使用依赖注入的方式，使用服务类对象。依赖注入的方式就是在我们的action方法中直接获取到这个服务类对象。

我们可以修改services.yaml配置文件，在services配置下的_defaults配置下，我们设置public为true，这样在src目录下所有的服务类都是公开的 可以通过容器来直接获取。

```
#config/services.yaml
services:
    # default configuration for services in *this* file
    _defaults:
        autowire: true      # Automatically injects dependencies in your
services.
        autoconfigure: true # Automatically registers your services as
commands, event subscribers, etc.
        public: true
```

再次执行单元测试，这次执行通过了。如果我们不想公开src下的所有服务类，我们可以单独配置服务类的public配置，注释这行代码。在services配置段下添加服务类的配置，我们的服务ID就是我们的类名，拷贝，

我们设置PostFactory这个服务ID它的public属性为true。这样就单独的对PostFactory服务类设置为公开状态。

```
#config/services.yaml
services:

    App\Factory\PostFactory:
        public: true
```

我们再次执行单元测试代码，测试依旧通过了。

下面我们使用PostFactory对象来快速的创建几篇文章，然后使用EntityManager将我们的文章保存到数据库中。$factory->create()，title，第二个参数正文。我们使用EntityManager的`persist()`方法，将我们创建的文章对象保存到数据库中。在`persist()`方法之后，我们需要提交数据库操作，使用`flush()`方法。

测试这段代码，我们看文章对象有没有保存到数据库中，打开数据库客户端，我们打开测试数据库。在post表中，我们查看所有的行，我们刚刚创建的文章成功保存到了数据库中。

我们现在进行的是集成测试代码的编写，每次我们执行persist()方法之后，我们还需要手动的查看一下数据库的结果，这就不自动化了。

在下一节课，我们将读取数据库来验证我们的数据是否已经插入成功了。