我们打开PostTest测试代码，我们需要使用Post对象时，都会使用new关键字来生成一个Post对象。随着博客系统的使用，生成的文章会越来越多，我们不想每次都使用new关键字来生成Post对象。我们需要使用一个工厂方法来帮助我们生成Post对象。这样我们在管理端创建文章时，一旦提交表单，就可以在action方法中获取表单的各个参数，再使用工厂方法来协助我们生成一个Post对象。

我们使用测试驱动开发的流程来进行开发，使用命令行`symfony console make:test`来生成一个单元测试，选择`TestCase`。类名称我们叫做`PostFactoryTest`。

打开测试代码，方法的名称我们叫做`testFactory`，我们希望创建一个工厂类，工厂类提供一个create()方法，直接帮我们生成一个Post对象。首先创建一个工厂实例，$factory有一个create()方法，create()方法的返回值是一个Post对象。然后我们断言一下，断言$post对象是一个Post类的实例。

执行这个测试，测试失败了，我们并没有创建`PostFactory`类。在src目录中，我们创建一个目录叫Factory。添加一个PHP类，叫做`PostFactory`，在类中有一个`create()`方法，它的返回对象类型是`Post`类型，它直接返回一个`new Post()`。

修改测试代码，引入`PostFactory`类，再次进行测试，这次测试通过了。回看我们的`PostFactory`类，这是一个很简单的方法，我们希望在`create()`方法中创建一个完整的Post对象，它包含标题、正文、摘要等等信息。

我们重构代码，添加代码$post变量等于`new Post()`，然后添加Post对象的标题，添加Post正文，添加摘要。文章的状态我们设置为草稿状态，最后我们返回Post对象，再次进行测试，测试通过了。

这样写的话有个很大的问题，每次我们使用`create()`方法时，它都会创建一个Post对象，所有的Post对象，它的标题、正文都是一样的，我们需要再次重构代码，为create()方法添加一些参数。

在create()方法中添加一个string类型的参数，叫title，再添加一个string类型的body，再添加一个string类型的摘要，再添加一个string类型的文章状态。这样我们就可以使用传参的方式，每次生成不同的文章，再次执行代码，代码错误了，我们还需要修改测试代码，为create()方法添加标题，摘要等等参数。

再次执行测试，这次执行通过了，这就是测试系统开发的一整套流程。

在下一节，我们将继续优化PostFactory类。