继续修改测试代码，来删除第16行，在第15行，我们期待$factory的create()方法使用这三个参数，可以创建一个Post对象，在创建之前我们可以预测一下它的行为。

在`create()`方法前一行添加代码，$factory现在是个仿件对象，我们期待`create()`方法可以执行一次，create()方法的参数就是下面这三个参数 最终他将返回一个Post对象。

我们有标题，有正文，有摘要，我们会手动创建一个Post对象。设置Post对象的标题，设置Post对象的摘要，设置Post对象的正文，设置Post对象的发布状态，因为我们没有传递status参数，所以它默认就是草稿状态。

再看22行代码，我们期待`create()`方法可以执行一次，create()方法的三个参数是这三个，然后它将返回一个Post对象。在26行，仿件对象执行create()方法。在28行我们添加一个断言，我们断言26行返回的Post对象就是刚刚我们创建的$postObj这个对象。注释32行、33行代码，执行单元测试，测试通过了。

```
class PostFactoryTest extends TestCase
{
    public function testFactory(): void
    {
//        $factory = new PostFactory();
        $factory = $this->createMock(PostFactory::class);

        $postObj = new Post();
        $postObj->setTitle('这是一个标题');
        $postObj->setSummary("这是摘要");
        $postObj->setBody('这是正文');
        $postObj->setStatus(['draft']);

        $factory->expects($this->once())->method('create')
           ->with("这是一个标题", "这是正文", "这是摘要")
           ->willReturn($postObj);

        $post = $factory->create("这是一个标题", "这是正文", "这是摘要");

        $this->assertInstanceOf(Post::class, $post);
        $this->assertSame($postObj, $post);
        $this->assertArrayHasKey('draft', $post->getStatus());
    }
}
```

我们取消32行、33行行前的注释，现在`create()`方法将会执行两次，两次的参数不同。我们注释(口误)22行到24行的代码，我们重新预测$factory仿件对象的行为。$factory对象的create()方法将执行两次，两次create()方法参数不一致，第一次是这三个参数，第二次是两个参数，所以这一次我们将使用`withConsecutive()`这个方法传递参数。我们期待他返回两个对象，第一次返回的对象就是我们创建过的$postObj对象，第二次返回的对象，我们需要另外创建一个对象。

在行前我们手动创建对象，设置对象的标题，设置对象的正文，设置对象的摘要，设置对象的发布状态，现在我们在第42行先下一个断言，我们断言第二次创建的$post2对象和刚刚我们手动写的$postObj2对象是一致的。

```
class PostFactoryTest extends TestCase
{
    public function testFactory(): void
    {
//        $factory = new PostFactory();
        $factory = $this->createMock(PostFactory::class);

        $postObj = new Post();
        $postObj->setTitle('这是一个标题');
        $postObj->setSummary("这是摘要");
        $postObj->setBody('这是正文');
        $postObj->setStatus(['draft']);

//        $factory->expects($this->once())->method('create')
//            ->with("这是一个标题", "这是正文", "这是摘要")
//            ->willReturn($postObj);
        $postObj2 = new Post();
        $postObj2->setTitle('这是第二个文章标题');
        $postObj2->setBody('<h1>这是第二个文章正文</h1>');
        $postObj2->setSummary('这是第二个文章正文');
        $postObj2->setStatus(['draft']);

        $factory->expects($this->exactly(2))->method('create')
            ->withConsecutive(["这是一个标题", "这是正文", "这是摘要"], ['这是第
二个文章标题', '<h1>这是第二个文章正文</h1>'])
            ->willReturn($postObj, $postObj2);

        $post = $factory->create("这是一个标题", "这是正文", "这是摘要");

        $this->assertInstanceOf(Post::class, $post);
        $this->assertSame($postObj, $post);
        $this->assertArrayHasKey('draft', $post->getStatus());

        $post2 = $factory->create('这是第二个文章标题', '<h1>这是第二个文章正文
</h1>');
        $this->assertSame($postObj2, $post2);
        $this->assertSame('这是第二个文章正文', $post2->getSummary());
    }
}
```

再次执行单元测试，测试通过了，我们没有使用new关键字来创建服务类对象，我们使用仿件对象，通过预测仿件对象的行为以及返回结果来进行单元测试，这就是服务类的单元测试。

在下一节。我们将使用orm框架的命令行，为我们的Post类和Comment类在数据库中生成对应的表。