继续重构代码，我们看create()方法，我们希望status参数是个带有默认值的参数。这样我们在使用create()方法时就可以省略掉status参数，我们希望不传递status参数时，它的默认值就是草稿状态。

修改create()方法，回到测试代码，我们删除status参数，再次下个断言，我们断言文章的状态是草稿状态，再次执行测试，测试通过了。我们不想每次在生成文章时都需要手动输入摘要，我们希望我们的摘要是正文内容的前140个字，这样如果我们不传递摘要内容，它会自动截取正文的前140个字符作为摘要内容。

我们首先编写测试代码，创建第二个文章对象，我们断言，文章的摘要就是正文的前140个字符，现在正文的长度不超过140个字符，所以文章的摘要和正文是一致的，执行测试代码。

测试没有通过，在第19行，它至少需要三个参数，现在我们没有传入摘要参数。重构代码，给summary参数传递一个默认值，默认值为null，如果第三个参数summary传递了值，那么Post对象就设置摘要内容。如果没有传递摘要值的话，就需要截取正文的前140个字符作为摘要。

修改代码，我们使用mb_substr()方法来截取正文的前140个字符作为文章的摘要，再次执行代码，这次执行通过了。我们文章编辑的时候，大多都使用了文章编辑器，文章编辑器会在我们的正文中添加一些HTML代码。

我们需要继续重构代码来优化摘要的生成方式，在工厂类中我们添加一个私有方法，叫做sliceBodyToSummary()，它的第一个参数是正文，第二个参数是截取的内容长度，这里使用整数类型参数，我们直接返回mb_substr()，body从第0个字符开始到$length长度，$length我们可以修改为带有默认值的参数。

我们还需要过滤掉body正文中的HTML标签，使用strip_tags()方法，修改create()方法，再次执行测试，测试依旧通过了，我们在正文中添加一些HTML标签，再次测试，测试依旧通过了。这样我们就完成了PostFactory的重构。

```
#文章工厂类
class PostFactory
{
    public function create(string $title, string $body, string $summary =
null, string $status = 'draft'): Post
    {
//        return new Post();
        $post = new Post();
        $post->setTitle($title);
        $post->setBody($body);
        if ($summary) {
            $post->setSummary($summary);
        } else {
            $post->setSummary($this->sliceBodyToSummary($body));
        }
        $post->setStatus($status);

        return $post;
    }

    private function sliceBodyToSummary(string $body, int $length = 140)
    {
        return mb_substr(strip_tags($body), 0, $length);
```

```
    }
}
```

```
#文章工厂测试类
class PostFactoryTest extends TestCase
{
    public function testFactory(): void
    {
        $factory = new PostFactory();
        $post = $factory->create("这是一个标题", "这是正文", "这是摘要");

        $this->assertInstanceOf(Post::class, $post);
        $this->assertSame('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('这是第二个文章正文', $post2->getSummary());
    }
}
```

回看测试代码，我们的工厂类它提供了一个功能方法，在Symfony中提供功能方法的类，我们可以叫做服务类。我们的factory构造函数中也并没有使用其他的服务类，所以我们使用new关键字来创建了一个工厂对象。

如果我们的工厂类，它的构造函数中依赖了其他的服务类，我们在进行单元测试时，我们需要在构造方法中传入其他的服务类对象，这样单元测试就没有意义了。单元测试我们只针对一个类进行测试，如果这个类它依赖了其他的服务类，那么出现了bug，可能是我们的工厂类中的bug，也可能是其他的服务类中出现的bug，这样就失去了单元测试的意义。

phpunit提供了一个替身的概念Mock，在下一节，我们将使用Mock来对工厂类进行单元测试。