如何让你的代码更易于做单测

单测是代码质量的试金石，代码写得好不好，写一下单测就知道了。 写过一段时间代码的人，不管所在的团队有没有做单测，多多少少都听过单测，刚听到这个的时候大多数人的第一反应这个不是很简单吗，测试一下函数的输入输出呗。但是当你一开始做的时候，你就会开始怀疑人生，感觉自己都不会写代码了。测试一个函数，单侧的代码可能都已经是这个函数的好几倍了，各种全局变量需要初始化，好多函数需要打桩，才能使得这个函数执行下去，校验结果也是需要各种打桩。下面一起来看看怎么才能让写单测愉快一点。

怎么衡量易于做单侧

要看一件事情做得好不好，得先有一个评价标准，没有评价标准就没有好坏一说。所以首先我们先来看看如何衡量一段代码是容易做单测的。

> 1. **代码覆盖率**：一般需要在 **85%** 以上
> 2. **单测代码行数与被测代码的比例**：1:1以内可以认为是优秀，1:1到2:1可以认为是良好，超过2:1就需要优化一下设计了

阻碍点&&解决方案

上面了解了衡量好做单测的一些指标，接下来看看有哪些阻碍点以及其对应的解决方案。

单元内过于复杂

```
output_type complicate_code(input_type x, ...)
{
```

```
    {
        if (xxx)
        /*Logic A*/
    }
    {
        if (xxx)
        /*logic B*/
    }
    {
        if (xxx)
        /*logic C*/
    }
    /*Logic */
}

```

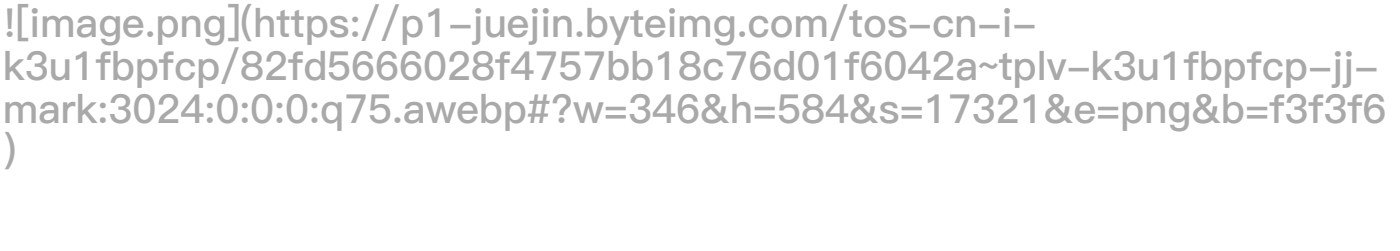
如上代码，函数`complicate_code`里面包括三个业务逻辑`logic A/B/C`。这个会导致几个问题：

> 1. 测试用例过多：比如`A/B/C`各有三种情况，那么就需要`3x3x3=27`个测试用例
> 2. 单测用例难以维护：修改了`logic A`的逻辑，排列组合的情况就又会变化，很多用例都要跟着改。

解决方案

还是上面的代码，我们来看看如何解决。

* `Logic A/B/C`间没有依赖：这种就简单了，只需要把`A/B/C`拆出来成为三个独立的函数，独立测试即可。
* `Logic A/B/C`间有依赖关系：这种我们可以构造一个数据结构将依赖隔开。这里对这个数据结构有一个要求是**好构造**，不然就没有意义了。比如`B`依赖于`A`，这个时候可以构造一个数据结果X，让`A`的输出为`X`，而`B`的输入则为`X`。因为`X`是比较好构造的，所以我们可以很容易地测试`A/B`。

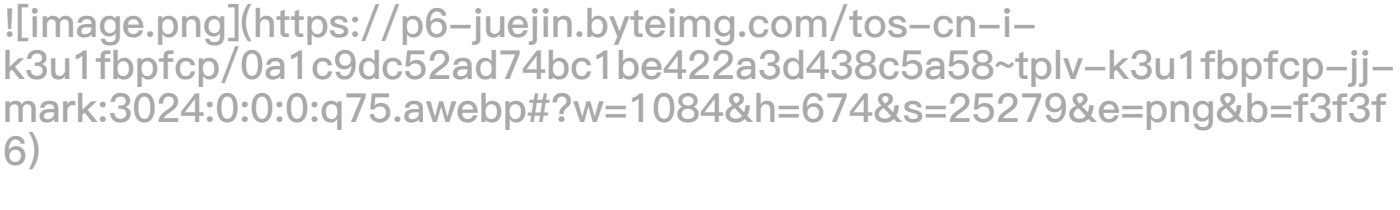


单元间耦合度过高

1. 依赖全局变量

> 全局变量的过度使用，对于单元测试来说简直是灾难。一个函数内部有全局变量，意味着单测的用例你都要去初始化这个全局变量，而且还要测试全局变量的不同情况，因为很多时候可能还依赖全局变量做分支判断。

2. 依赖其他单元，或者说依赖其他对象。

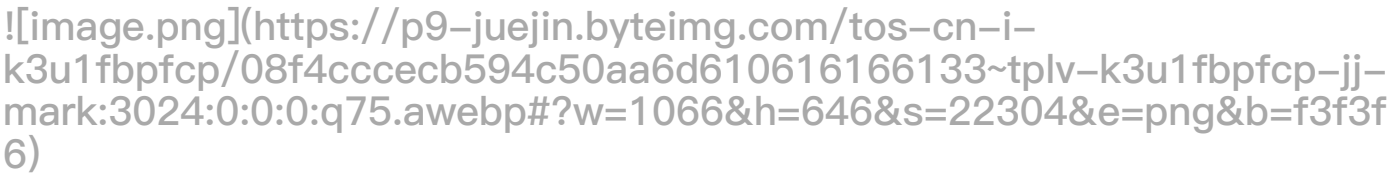


> 如上图所示，
>
>
> 1. 在`UnitA`里面你需要测试`Unit B/C/D`的各种异常返回，而且由于这个是直接调用的，所以这里面还需要打桩。
> 2. `Unit A`里面除了测试自己的业务逻辑以外，如果`B/C/D`的调用关系之间还有依赖，那么你还要测试这个依赖。

解决方案

解决全局变量：简单来说就是去掉全局变量，具体可以通过函数参数的形式注入，**让依赖的全局变量变成函数接口的一部分。**

解决单元依赖：如果调用逻辑不复杂，那么可以通过依赖注入的方式，将`B/C/D`的接口直接注入进去，这样子就可以减少打桩带来的成本，这个也就是**基于接口编程而非实现编程**的设计原则。如果B/C/D的调用逻辑比较复杂，那么可以将单元的业务逻辑和调用逻辑解耦。如下面所示，由门面接口来处理`B/C/D`的调用，处理调用的异常以及`B/C/D`之间的依赖，对单元A只提供一个固定的接口。



单元外依赖过多

很多时候单元测试不好测，难点都在于外部依赖，但是一个程序基本上可以说都是有输入输出的，所以外部依赖可以说是绕不过的问题。外部依赖导致不好测的一个最大原因在于**业务逻辑和外部依赖耦合在一起**了，导致每次测试都只能**集成整个系统**进行测试。

解决这个问题的关键点就在于**解耦外部依赖和内部逻辑**，这个的核心解决方案就是**隔离**。

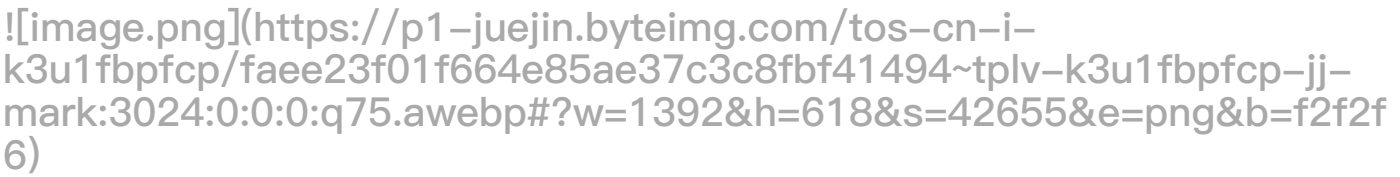
通过隔离我们可以把重点集中在内部逻辑上，让内部逻辑不用集成外部依赖就可以测试。针对外部依赖的使用方式不同可以分成两种：

* **业务逻辑调用外部依赖项。** 这种在开发过程中是最多的。比如一个业务逻辑可能涉及到数据库读写，这个时候就需要去调用数据库的读写接口，比如`mysql`库提供的读写数据库接口。
* **外部依赖调用业务逻辑。** 这种一般是通过注册的方式注入到框架，由第三方框架处理业务的时候调用，比如nginx服务器的插件。

业务逻辑调用外部依赖项

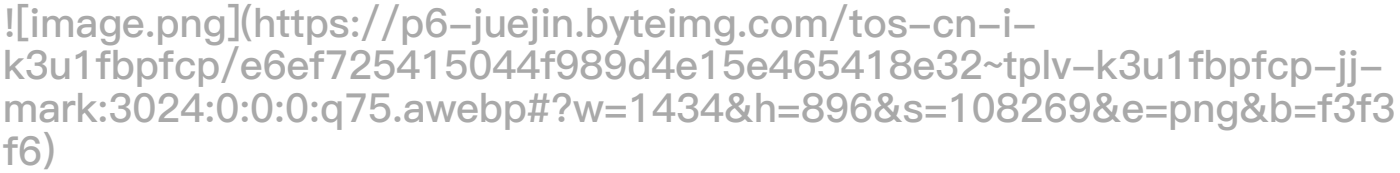
如上图所示业务逻辑依赖于外部依赖项，通过一个抽象的接口把外部依赖的细节封装在接口之后，这样子就可以把**外部依赖的细节都隐藏在抽象接口之后**，业务逻辑就可以做独立测试了。这个对应的其实就是**依赖倒置**的设计原则。

外部依赖调用业务逻辑



外部框架调用我们的业务逻辑，如果直接处理框架的业务数据，带来的问题就是业务逻辑里面全是框架绑定的接口数据，这里我们的做法是实现一个**转发层，** 这个转发层的目的是隔离外部框架。**在这个转发层里面我们需要将外部框架的指令分发到业务逻辑里面**。也就是我们实现业务逻辑的时候是不应该去考虑外部框架的。

总结
--



上面我们可以看到易于做单测的方法用了很多设计原则，设计模式。所以一个拥有较好可测试性的代码，需要有一个好的设计去支撑。反之一个好测试的代码也是一个设计良好的代码，**通过单元测试可以驱动我们去检验设计，优化设计**。

在设计的时候我们就应该考虑可测试性，而不是等到代码写出来再去修改。当然了，一开始做单测的时候是很难做到设计的时候就考虑全，在编码构成中逐步重构也未尝不可。

原文链接: https://juejin.cn/post/7359464203358453786