13 | 先写测试，就是测试驱动开发吗？

郑晔 2019-01-28

你好，我是郑晔。

在上一讲中，我向你说明了为什么程序员应该写测试，今天我准备与你讨论一下程序员应该在什么阶段写测试。

或许你会说，写测试不就是先写代码，然后写测试吗？没错，这是一个符合直觉的答案。但是，这个行业里确实有人探索了一些不同的做法。接下来，我们就将进入不那么直觉的部分。

既然自动化测试是程序员应该做的事，那是不是可以做得更极致一些，在写代码之前就把测试先写好呢？

有人确实这么做了，于是，形成了一种先写测试，后写代码的实践，这个实践的名字是什么呢？它就是测试先行开发（Test First Development）。

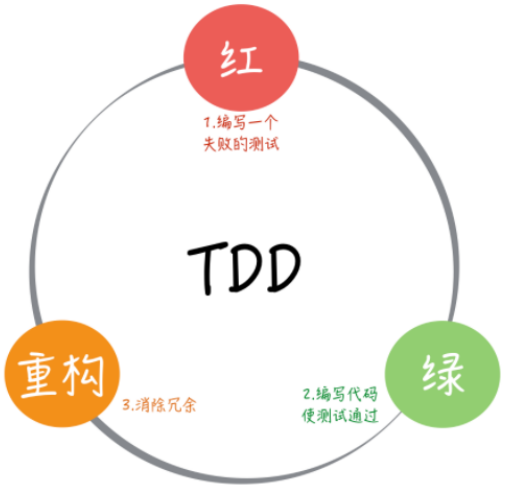
我知道，当我问出这个问题的时候，一个名字已经在很多人的脑海里呼之欲出了，那就是**测试驱动开发（Test Driven Development）**，也就是大名鼎鼎的 **TDD**，TDD 正是我们今天内容的重点。

在很多人看来，TDD 就是先写测试后写代码。在此我必须澄清一下，这个理解是错的。先写测试，后写代码的实践指的是测试先行开发，而非测试驱动开发。

下一个问题随之而来，测试驱动开发到底是什么呢？测试驱动开发和测试先行开发只差了一个词：驱动。只有理解了什么是驱动，才能理解了测试驱动开发。要理解驱动，先来看看这两种做法的差异。

测试驱动开发

学习 TDD 的第一步，是要记住 TDD 的节奏：“红 - 绿 - 重构”。



红，表示写了一个新的测试，测试还没有通过的状态；绿，表示写了功能代码，测试通过的状态；而重构，就是再完成基本功能之后，调整代码的过程。

这里说到的“红和绿”，源自单元测试框架，测试不过的时候展示为红色，通过则是绿色。这在单元测试框架形成之初便已经约定俗成，各个不同语言的后代也将它继承了下来。

我们前面说过，让单元测试框架流行起来的是 JUnit，它的作者之一是 Kent Beck。同样，也是 Kent Beck 将 TDD 从一个小众圈子带到了大众视野。

考虑到 Kent Beck 是单元测试框架和 TDD 共同的贡献者，你就不难理解为什么 TDD 的节奏叫“红 - 绿 - 重构”了。

测试先行开发和测试驱动开发在第一步和第二步是一样的，先写测试，然后写代码完成功能。二者的差别在于，测试驱动开发并没有就此打住，它还有一个更重要的环节：**重构（refactoring）。**

也就是说，在功能完成而且测试跑通之后，我们还会再次回到代码上，处理一下代码上写得不好的地方，或是新增代码与旧有代码的重复。因为我们第二步“绿”的关注点，只在于让测试通过。

测试先行开发和测试驱动开发的差异就在重构上。

很多人通过了测试就认为大功告成，其实，这是忽略了新增代码可能带来的“坏味道（Code Smell）”。

如果你真的理解重构，你就知道，它就是一个消除代码坏味道的过程。一旦你有了测试，你就可以大胆地重构了，因为任何修改错误，测试会替你捕获到。

在测试驱动开发中，重构与测试是相辅相成的：没有测试，你只能是提心吊胆地重构；没有重构，代码的混乱程度是逐步增加的，测试也会变得越来越不好写。

因为重构和测试的互相配合，它会驱动着你把代码写得越来越好。这是对“驱动”一词最粗

浅的理解。

测试驱动设计

接下来，我们再来进一步理解“驱动”：**由测试驱动代码的编写。**

许多人抗拒测试有两个主要原因：第一，测试需要“额外”的工作量。这里我特意把额外加上引号，因为，你也许本能上认为，测试是额外的工作，但实际上，测试也应该是程序员工作的一部分，这在上一篇文章中我已经讲过。

第二，很多人会觉得代码太多不好测。之所以这些人认为代码不好测，其中暗含了一个假设：代码已经写好了，然后，再写测试来测它。

如果我们把思路反过来，我有一个测试，怎么写代码能通过它。一旦你先思考测试，设计思路就完全变了：**我的代码怎么写才是能测试的，也就是说，我们要编写具有可测试性的代码。**用这个角度，测试是不是就变得简单了呢？

这么说还是有些抽象，我们举个写代码中最常见的问题：static 方法。

很多人写代码的时候喜欢使用 static 方法，因为用着省事，随便在哪段代码里面，直接引用这个 static 方法就可以。可是，一旦当你写测试的时候，你就会发现一个问题，如果你的代码里直接调用一个 static 方法，这段代码几乎是没法测的。尤其是这个 static 方法里面有一些业务逻辑，根据不同业务场景返回各种值。为什么会这样？

我们想想，常见的测试手法应该是什么样的？如果我们在做的是单元测试，那测试的目标应该就是一个单元，在这个面向对象作为基础设施流行的时代，这个单元大多是一个类。测试一个类，尤其是一个业务类，一般会涉及到一些与之交互的类。

比如，常见的 REST 服务三层架构中，资源层要访问服务层，而在服务层要访问数据层。编写服务层代码时，因为要依赖数据层。所以，测试服务层通常的做法是，做一个假的数据层对象，这样即便数据层对象还没有编写，依然能够把服务层写完测好。

在之前的“蛮荒时代”，我们通常会写一个假的类，模拟被依赖那个类，因为它是假的，我

们会让它返回固定的值，使用这样的类创建出来的对象，我们一般称之为 Stub 对象。

这种“造假”的方案之所以可行，一个关键点在于，这个假对象和原有对象应该有相同的接口，遵循同样的契约。从设计上讲，这叫符合 Liskov 替换法则。这不是我们今天讨论的重点，就不进一步展开了。

因为这种“造假”的方案实在很常见，所以，有人做了框架支持它，就是常用的 Mock 框架。使用 Mock 对象，我们可以模拟出被依赖对象的各种行为，返回不同的值，抛出异常等等。

它之所以没有用原来 Stub 这个名字，是因为这样的 Mock 对象往往有一个更强大的能力：验证这个 Mock 对象在方法调用过程中的使用情况，比如调用了几次。

我们回到 static 的讨论上，你会发现 Mock 对象的做法面对 static 时行不通了。因为它跳出了对象体系，static 方法是没法继承的，也就是说，没法用一系列面向对象的手法处理它。你没有办法使用 Mock 对象，也就不好设置对应的方法返回值。

要想让这个方法返回相应的值，你必须打开这个 static 方法，了解它的实现细节，精心地按照里面的路径，小心翼翼地设置对应的参数，才有可能让它给出一个你预期的结果。

更糟糕的是，因为这个方法是别人维护的，有一天他心血来潮修改了其中的实现，你小心翼翼设置的参数就崩溃了。而要重新进行设置的话，你只能把代码重读一遍。

如此一来，你的工作就退回到原始的状态。更重要的是，它并不是你应该关注的重点，这也不会增加你的 KPI。显然，你跑偏了。

讨论到这里你已经知道了 static 方法对测试而言，并不友好。所以，如果你要想让你的代码更可测，**一个好的解决方案是尽量不写 static 方法。**

这就是“从测试看待代码，而引起的代码设计转变”的一个典型例子。

关于 static 方法，我再补充几点。static 方法从本质上说，是一种全局方法，static 变量就

是一种全局变量。我们都知道，全局方法也好，全局变量也罢，都是我们要在程序中努力消除的。一旦放任 static 的使用，就会出现和全局变量类似的效果，你的程序崩溃了，因为别人在另外的地方修改了代码，代码变得脆弱无比。

static 是一个方便但邪恶的东西。所以，要限制它的使用。除非你的 static 方法是不涉及任何状态而且行为简单，比如，判断字符串是否为空。否则，不要写 static 方法。你看出来了，这样的 static 方法更适合做库函数。所以，我们日常写应用时，能不用尽量不用。

前面关于 static 方法是否可以 Mock 的讨论有些绝对，市面上确实有某些框架是可以 Mock static 方法的，但我不建议使用这种特性，因为它不是一种普遍适用的解决方案，只是某些特定语言特定框架才有。

更重要的是，正如前面所说，它会在设计上将你引到一条不归路上。

如果你在自己的代码遇到第三方的 static 方法怎么办，很简单，将第三方代码包装一下，让你的业务代码面对的都是你自己的封装就好了。

以我对大多数人编程习惯的认知，上面这个说法是违反许多人编程直觉的，但如果你从代码是否可测的角度分析，你就会得到这样的结论。

先测试后写代码的方式，会让你看待代码的角度完全改变，甚至要调整你的设计，才能够更好地去测试。所以，很多懂 TDD 的人会把 TDD 解释为测试驱动设计（Test Driven Design）。

还有一个典型的场景，从测试考虑会改变的设计，那就是依赖注入（Dependency Injection）。

不过，因为 Spring 这类 DI 容器的流行，现在的代码大多都写成了符合依赖注入风格的代码。原始的做法是直接 new 一个对象，这是符合直觉的做法。但是，你也可以根据上面的思路，自己推演一下，从 new 一个对象到依赖注入的转变。

有了编写可测试代码的思路，即便你不做 TDD，依然对你改善软件设计有着至关重要的作

用。所以，**写代码之前，请先想想怎么测。**

即便我做了调整，是不是所有的代码就都能测试了呢？不尽然。从我个人的经验上看，不能测试的代码往往是与第三方相关的代码，比如访问数据库的代码，或是访问第三方服务之类的。但不能测试的代码已经非常有限了。我们将它们隔离在一个小角落就好了。

至此，我们已经从理念上讲了怎样做好 TDD。有的人可能已经跃跃欲试了，但更多的人会用自己所谓的“经验”告诉你，TDD 并不是那么好做的。

怎么做好 TDD 呢？下一讲，我会给你继续讲解，而且，我们“任务分解大戏”这个时候才开始真正拉开大幕！

总结时刻

一些优秀的程序员不仅仅在写测试，还在探索写测试的实践。有人尝试着先写测试，于是，有了一种实践叫测试先行开发。还有人更进一步，一边写测试，一边调整代码，这叫做测试驱动开发，也就是 TDD。

从步骤上看，关键差别就在，TDD 在测试通过之后，要回到代码上，消除代码的坏味道。

测试驱动开发已经是行业中的优秀实践，学习测试驱动开发的第一步是，记住测试驱动开发的节奏：红——绿——重构。把测试放在前面，还带来了视角的转变，要编写可测的代码，为此，我们甚至需要调整设计，所以，有人也把 TDD 称为测试驱动设计。

如果今天的内容你只能记住一件事，那请记住：**我们应该编写可测的代码。**

最后，我想请你分享一下，你对测试驱动开发的理解是怎样的呢？学习过这篇内容之后，你又发现了哪些与你之前理解不尽相同的地方呢？欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。

律责任。

胡瑶

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Ctrl + Enter 发表　　0/2000字　　提交留言

精选留言(33)

邵俊达

最近在一个新项目中尝试使用了 TDD 有测试保驾护航是真的爽。事情是这样的，今天经过讨论要把一个模型替换掉，刚听到这个消息的时候我是崩溃的，心想这要是出 bug 怎么办，到转念一想我测试覆盖率 88% 应该还好，动手改完跑起测试，果然不过，但是只是几个叫小问题，再次运行测试 绿灯！我的天，此时内心别提多么舒爽，这要是没有测试我今晚应该不用睡了，谢谢老师。最近也开始先分解任务再小步提交，这么做下来有一种很踏实的感觉，同事 review 代码也舒服很多。

作者回复: 恭喜你，进步了！

2019-04-17　　13

行与修

测试驱动不但可以写出精炼的代码，还能养成良好的编程习惯和设计思维， 相辅相成。
团队达到这种状态还真是不易，架子搭好了有人觉得没发挥空间，要是放开了代码又会五花八门难以测试。莫名有种担心，会不会为了测试而测试在代码里混搭着workaround 呢?

作者回复: 测试应该是什么样子，后面即将呈现，敬请期待!

2019-01-30 7

彩色的沙漠

阅读之后有了基本的认知，还需要阅读这方面的相关书籍和实践

作者回复: 更重要的是练习

2019-01-30 5

Sudouble

之前在其他领域的里也有介绍负反馈相关单位内容，各个领域之间还是有很多互通之处。软件或者其他系统一直处于熵增的状态，需要持续的保养和维护。不得不感慨大道至简啊

作者回复: 一通百通

2019-02-11 4

又双叒叕是一年啊

好感动哭了

编辑回复: 这位同学，你是认真的吗😁

2019-01-28 3

Kǎfkā²⁰²⁰

印象最深的几次TDD。
1. 是个CS应用，从服务端拉取数据后，根据不同状态，客户端执行不同逻辑。采用的方法是，将服务端的响应值记录，然后在测试代码里回放，不依赖服务端。修bug时，每个bug就是一个测试，测试代码里直接回放记录的服务端响应。好处是，回归非常快，而且不依赖服务端
2. 上家公司要做HA软件开发，情况很复杂，手工做测试代价很高。正好赶上docker兴起，于是就写了很多“暴力”代码（比如直接kill服务、删除服务等）测试各种场景，只留少数必须用物理机测试的场景交给人工

> 作者回复: 第一个像验收测试，第二个像暴力测试。

2019-01-28 3

sam

学完这篇，真正了解了TDD，以前一直凭直觉理解TDD是完善测试驱动开发，现在发现最根本是驱动代码设计。

> 作者回复: 这回理解是对的。

2020-07-04 1

escray

之前在学习 TDD 的时候，确实忽略了“驱动”的概念，而且也有很多时候，根本驱动不起来。

从测试先行开发，通过重构，成为测试驱动开发，进而成为测试驱动设计。

以前似乎也忽略了重构，只注意“红-绿”的节奏，而把重构当做了测试驱动开发之外的一个动作，其实重构，或者说是小规模的重构，应该是 TDD 的应有之意。

这个任务分解的模块，其实更多的讲的是 TDD，准备把《测试驱动开发》也一并读一下。

作者回复: 《测试驱动开发》值得细细品味。

2020-06-09 1

红糖白糖

TDD，和任务分解可以说是相辅相成。
在写测试的时候，一个一个的case其实在对任务的分解，考虑每个case所要达到的目标，输入、输出，以及case与case之间的衔接。写测试的时候，我们是站在一个consumer的角度来的，考虑的是这个case的输入和输出。首先，这对于设计能力有一定的要求，其次，按照这种方式写出来的代码可用性更高。因为我们的起点是consumer 而不是provider

作者回复: 关于设计，欢迎加入《软件设计之美》。

2019-03-10 1

楼下小黑哥

以前理解的测试驱动开发，我写完测试、代码完成就结束了。看完文章，增加对重构理解。有了测试的依托，改动代码的结果也能从测试结果中看出。
目前对于 TDD 还是处于理解状态，不知道如何真正的在项目工程中使用。因为项目工程往往还有很多其他调用，如rpc，数据库服务，第三方服务，不知道在这个过程如何处理。期待老师的之后文章讲解

作者回复: 终于在综合运用模块答疑这个问题。

2019-01-31 1

liu

以前以为测试驱动开发只是先写测试，再开发就完了。原来还有重构。想想也对。代码第一步完成功能表达，但未经重构的代码必然存在坏味道。重构，才能使代码机构清晰，便于理解阅读

作者回复: 重构拯救代码!

2019-01-28 1 1

pyhhou

感谢老师指点，之前没听过 TDD，但是知道 Unit Test 很重要，要随着写代码一起写，也没想过原来可以先写 test，后实现 code，这样 test 很自然地时刻存在在开发的每一个阶段。在 TDD 里面 “红——绿——重构” 中，看完老师这边文章对其中的 “红” 和 “绿” 都能理解，因为就是写 code，保证 code 可测并且能够通过所有 test，但是对这里的 “重构” 还是比较地困惑，“重构” 很重要，那么这里有没有什么方向性或者方法可言呢，因为仅仅看 “重构” 的定义，貌似和 test 没有直接的联系，test 只是保证能够更好的 “重构”，文章中只说了 “重构” 是为了消除 “冗余”，消除代码的 “坏味道”，那什么是 “冗余”，什么是 “坏味道”，具体该怎么定义 “重构” 需要解决的问题呢？还有 “重构” 怎样才能更好地执行呢？谢谢老师

作者回复: 你可以看一下 Martin Fowler 的《重构》（https://book.douban.com/subject/4262627/），2018年第二版的英文版也已经出版了，其中文版的译者还是第一版的译者熊节。

2019-01-25 1

Better me

TDD：红-绿-重构流程
TDD中的重构节点确实很有必要，可以避免代码的坏味道
但这同时对开发提出更高的要求，要求我们具备时刻重构的意识
而重构代码后又可以通过测试来进行验证
不仅保证功能代码运行质量，同时保证功能代码设计质量

2020-11-29

Y024

原来理解的是测试先行开发（有时基于代码检查会做些被动的重构），离真正的测试驱动开发少了重要的主动重构环节。

作者回复: 失之毫厘谬以千里

2020-11-10

昨日的天使

老师我有一个问题：如果一个人重构运用的很熟练，那么他肯定能慢慢地变得能写出一些不需要重构的或者很少需要重构的代码。如果把这个问题想得极端一点，一个人的代码，如果写得不需要重构，或者代码想要重构得更好很难，那么这些不需要重构或者很难变得更好的代码，会不会成为代码维护者的负担呢？ 这里涉及到一个问题：有最好的代码吗？有不需要重构的代码吗？请老师解惑，谢谢啦。

作者回复: 其实不会，即使我现在已经对重构这些东西比较熟悉了，我还是会先按照一个简单的版本实现，然后，再去重构。

代码没有最好的代码，因为需求一直在变。需求一进来，再好的代码也要重新适应。

2020-11-07

Alexis何春光

请问为什么“一旦放任 static 的使用，就会出现和全局变量类似的效果，你的程序崩溃了，因为别人在另外的地方修改了代码，代码变得脆弱无比。”？ 一般来说，别人修改代码，应该不改变输入和输出，所以不会受到影响。而如果改变了输入和输出，就算是实例方法也会break吧?

作者回复: 如果代码可以保证是纯函数，当然不会有问题，但static的出现常常会破坏函数的纯粹性。你修改这个static的值，还不知道它在哪用到了，就可能带来很多问题。

2020-05-10

时代先锋

测试驱动开发，新颖的理论

2020-03-25

wesleydeng

对于开发者自测，有什么比较好的书籍推荐么？现在对于不同层次的测试感觉还是没有一个章法，有点乱，怎样可以提高自测的效率？

> 作者回复: 可以看看Kent Beck的《测试驱动开发》，还有一本《测试驱动的面向对象软件开发》（Growing Object-Oriented Software, Guided by Tests）也可以看看。

2020-03-25

我能走多远

测试驱动开发的精髓是第三步重构。最近也是做一个业务测试。发现最初的修改只是通过代码中加特殊的判断去规避了这个问题。在影响性测试中发现，正常的处理逻辑完全满足了这种业务场景。只是我们在当前场景中少置为了一个标记。导致下游不识别。在正常业务流程中增加这个标示完全解决问题。这就是重构的精髓

> 作者回复: 这好像是缺少了集成测试吧！

2019-12-21

丁丁历险记

可测的代码往往符合类的单一原则。

> 作者回复: 虽然并不直接相关，但一般情况下，是对的。

2019-11-07

陈斯佳

老师，运维写脚本也需要考虑测试吗？

作者回复: 我还真见过有人给运维脚本写测试。

2019-07-24

姚琪琳

感觉TDD的时候很容易就写成了冰淇淋蛋卷的测试结构，因为不太可能去TDD某个类里的某个方法。而且感觉冰淇淋蛋卷对重构更友好，毕竟测试是端到端的，不在乎内部如何实现。

2019-07-23

lu4nx

有时别人写的文档比较模糊时，我会直接看测试用例去了解接口怎么用。

2019-06-08 1

小一

PowerMock框架是可以支持mock static类和方法的

作者回复: 但我不建议使用，因为它会把设计引导向一个错误的方向。如果是应对遗留代码，勉强可用。

2019-06-03

陈斯佳

TDD的节奏是：红-绿-重构，先编写一个失败的测试，然后编写一个代码通过测试，最后消除冗余，消除代码的坏味道。写代码前先想想怎么测试，我们必须保证编写可测试的代码。

2019-05-15

enjoylearning

今天又被作者引入了测试先行的开发理念，一直觉得Tdd不一定要先写测试，完成功能代码再写测试也可以啊，用单元测试去验证写的逻辑，发现测试失败了，回去看代码实现，绿了再回去看代码坏味道。

> 作者回复: TDD 的关键其实在于设计。

2019-03-28

lyning

TDD 还需要事先分析和任务分解，不然就变成为了 TDD 而 TDD 了

> 作者回复: 开发就应该先做分析和分解，和TDD无关。

2019-03-17

姜浩远

测试清单可以和DoD联系起来么?

> 作者回复: 从实践的层面，这是两回事，如果你非要找二者的共性，也是有的。

2019-02-06 1

ZackZzzzzz

老师如果有机会的话，谈一谈对不同层面测试的理解吧。

我们现在的后端代码库大概有三种层面的测试
1.单元测试 - 对某个类的测试
2.系统测试 - 测试service之间的interaction
3. 我们还有一个container test, 测试方法是mock所有的external dependency数据库也好，其他的服务也好，这样能保证负责的业务逻辑从头到尾的结果

老师你对测试的理解是怎么样的？什么应该被测试，大概投入多少比例

作者回复: 其实，测试金字塔已经说了测试比例，越是底层的测试应该越多，只有尽可能多的单元测试才能有接近100%的覆盖率。

2019-01-29

九月三秋

实践实践

2019-01-29

又双叒叕是一年啊

挺好挺好

2019-01-28

chaoqiang

“作者回复: 你可以看一下 Martin Fowler 的《重构》（https://book.douban.com/subject/4262627/），2018年第二版的英文版也已经出版了，其中文版的译者还是第一版的译者熊节。”：重构第二版这本书期待很久了，但这本书好像目前国内购买途径还比较少，Amazon 上价格特别贵。想咨询下老师，怎么看待技术书（尤其是英文版）普遍价格高昂的现象呢，另外，针对新书有什么性价比高的途径购买呢

作者回复: 相对程序员的收入而言，书的价格其实并不贵，尤其是国内的书。其实行业中少有特别需要追的书，经典毕竟是少数，大部分经典书国内都有了。所以，如果你特别介意英文书的价格，那就等一段时间，国内就有了。如果时间是关键因素，价格就不重要了。

2019-01-28

萧

不久前第一次接触TDD时为它的思想而惊叹，感觉它能极大的提升编码效率，编码后期的大量重构，还能保障代码质量。后面自己在写代码的时候也注意使用它的思想，但说实话，理解是一回事，用起来就不是那么回事了，很多的东西还不是太熟练，前期说实话比较耗时间，有些拖进度。由于也毕业不久，经验上有些欠缺，还不太熟练，有些测试还不知道怎么写。现在写多了一点，感受到的是代码质量上的提高，bug比起以前少了，需求变更下也改动也不伤筋动骨了，但还是有许多感觉做的不够好的。看了这篇文章，给了一个补充TDD的认知，感受到如果和任务分解结合起来会有更好的效果，期待后面的文章！

作者回复: 大幕即将拉开，敬请期待！

2019-01-28