

16 | 为什么你的测试不够好?

郑晔 2019-02-04



00:00

讲述: 郑晔

大小: 11.82M

时长: 12:54

1.0x

你好! 我是郑晔。今天是除夕, 我在这里给大家拜年了, 祝大家在新的一年里, 开发越做越顺利!

关于测试, 我们前面讲了很多, 比如: 开发者应该写测试; 要写可测的代码; 要想做好 TDD, 先要做好任务分解, 我还带你进行了实战操作, 完整地分解了一个任务。

但有一个关于测试的重要话题, 我们始终还没聊, 那就是测试应该写成什么样。今天我就来说说怎么把测试写好。

你或许会说，这很简单啊，前面不都讲过了吗？不就是用测试框架写代码吗？其实，理论上来说，还真应该就是这么简单，但现实情况却往往相反。我看到过很多团队在测试上出现过各种各样的问题，比如：

- 测试不稳定，这次能过，下次过不了；
- 有时候是一个测试要测的东西很简单，测试周边的依赖很多，搭建环境就需要很长的时间；
- 这个测试要运行，必须等到另外一个测试运行结束；
-

如果你也在工作中遇到过类似的问题，那你理解的写测试和我理解的写测试可能不是一回事，那问题出在哪呢？

为什么你的测试不够好呢？

主要是因为这些测试不够简单。只有将复杂的测试拆分成简单的测试，测试才有可能做好。

简单的测试

测试为什么要简单呢？有一个很有趣的逻辑，不知道你想没想过，测试的作用是什么？显然，它是用来保证代码的正确性。随之而来的一个问题是，谁来保证测试的正确性？

许多人第一次面对这个问题，可能会一下子懵住，但脑子里很快便会出现一个答案：测试。但是，你看有人给测试写测试吗？肯定没有。因为一旦这么做，这个问题会随即上升，谁来保证那个测试的正确性呢？你总不能无限递归地给测试写测试吧。

既然无法用写程序的方式保证测试的正确性，我们只有一个办法：把测试写简单，简单到一目了然，不需要证明它的正确性。所以，如果你见到哪个测试写得很复杂，它一定不是一个好的测试。

既然说测试应该简单，我们就来看看一个简单的测试应该是什么样子。下面我给出一个简单的例子，你可以看一下。

 复制代码

```
1 @Test
2 void should_extract_HTTP_method_from_HTTP_request() {
3     // 前置准备
4     request = mock(HttpRequest.class);
5     when(request.getMethod()).thenReturn(HttpMethod.GET);
6     HttpMethodExtractor extractor = new HttpMethodExtractor();
7
8     // 执行
9     HttpMethod method = extractor.extract(request);
10
11     // 断言
12     assertThat(method, is(HttpMethod.GET));
13
14     // 清理
15 }
```

这个测试来自我的开源项目 [Moco](#)，我稍做了一点调整，便于理解。这个测试很简单，从一个 HTTP 请求中提取出 HTTP 方法。

我把这段代码分成了四段，分别是**前置准备**、**执行**、**断言**和**清理**，这也是一般测试要具备的四段。

- 这几段的核心是中间的执行部分，它就是测试的目标，但实际上，它往往也是最短小的，一般就是一行代码调用。其他的部分都是围绕它展开的，在这里就是调用 HTTP 方法提取器提取 HTTP 方法。
- 前置准备，就是准备执行部分所需的依赖。比如，一个类所依赖的组件，或是调用方法所需要的参数。在这个测试里面，我们准备了一个 HTTP 请求，设置了它的方法是一个 GET 方法，这里面还用到了之前提到的 Mock 框架，因为完整地设置一个 HTTP 请求很麻烦，而且与这个测试也没什么关系。
- 断言是我们的预期，就是这段代码执行出来怎么算是对的。这里我们判断了提取出来的方法是否是 GET 方法。另外补充一点，断言并不仅仅是 assert，如果你用 Mock 框架的话，用以校验 mock 对象行为的 verify 也是一种断言。
- 清理是一个可能会有部分，如果你的测试用到任何资源，都可以在这里释放掉。不

过，如果你利用好现有的测试基础设施（比如，JUnit 的 Rule），遵循好测试规范的话，很多情况下，这个部分就会省掉了。

怎么样，看着很简单吧，是不是符合我前面所说的不证自明呢？

测试的坏味道

有了对测试结构的了解，我们再来说说常见的测试“坏味道”。

首先是执行部分。不知道你有没有注意到，前面我提到执行部分时用了个说法，一行代码调用。是的，第一个“坏味道”就来自这里。

很多人总想在一个测试里做很多的事情，比如，出现了几个不同方法的调用。请问，你的代码到底是在测试谁呢？

这个测试一旦出错，就需要把所有相关的几个方法都查看一遍，这无疑是增加了工作的复杂度。

也许你会问，那我有好几个方法要测试，该怎么办呢？很简单，多写几个测试就好了。

另一个典型“坏味道”的高发区是在断言上，请记住，**测试一定要有断言**。没有断言的测试，是没有意义的，就像你说自己是世界冠军，总得比个赛吧！

我见过不少人写了不少测试，但测试运行几乎从来就不会错。出于好奇，我打开代码一看，没有断言。

没有断言当然就不会错了，写测试的同事还很委屈地说，测试不好写，而且，他已经验证了这段代码是对的。就像我前面讲过的，测试不好写，往往是设计的问题，应该调整的是设计，而不是在测试这里做妥协。

还有一种常见的“坏味道”：复杂。最典型的场景是，当你看到测试代码里出现各种判断和循环语句，基本上这个测试就有问题了。

举个例子，测试一个函数，你的断言写在一堆 if 语句中，美其名曰，根据条件执行。还是前面提到的那个观点，你怎么保证这个测试函数写的是对的？除非你用调试的手段，否则，你都无法判断你的条件分支是否执行到了。

你或许会疑问，我有一大堆不同的数据要测，不用循环不用判断，我怎么办呢？**你真正应该做的是，多写几个测试，每个测试覆盖一种场景。**

一段旅程 (A-TRIP)

怎么样的测试算是好的测试呢？有人做了一个总结 A-TRIP，这是五个单词的缩写，分别是

- Automatic, 自动化;
- Thorough, 全面的;
- Repeatable, 可重复的;
- Independent, 独立的;
- Professional, 专业的。

下面，我们看看这几个单词分别代表什么意思。

Automatic, 自动化。有了前面关于自动化测试的铺垫，这可能最好理解，就是把测试尽可能交给机器执行，人工参与的部分越少越好。

这也是我们在前面说，测试一定要有断言的原因，因为一个测试只有在有断言的情况下，机器才能自动地判断测试是否成功。

Thorough, 全面，应该尽可能用测试覆盖各种场景。理解这一点有两个角度。一个是在写代码之前，要考虑各种场景：正常的、异常的、各种边界条件；另一个角度是，写完代码之后，我们要看测试是否覆盖了所有的代码和所有的分支，这就是各种测试覆盖率工具发挥作用的场景了。

当然，你想做到全面，并非易事，如果你的团队在补测试，一种办法是让测试覆盖率逐步提

升。

Repeatable, 可重复的。这里面有两个角度：某一个测试反复运行，结果应该是一样的，这说的是，每一个测试本身都不应该依赖于任何不在控制之下的环境。如果有，怎么办，想办法。

比如，如果有外部的依赖，就可以采用模拟服务的手段，我的 [@Moco](#) 就是为了解决外部依赖而生的，它可以模拟外部的 HTTP 服务，让测试变得可控。

有的测试会依赖数据库，那就在执行完测试之后，将数据库环境恢复，像 Spring 的测试框架就提供了测试数据库回滚的能力。如果你的测试反复运行，不能产生相同的结果，要么是代码有问题，要么是测试有问题。

理解可重复性，还有一个角度，一堆测试反复运行，结果应该是一样的。这说明测试和测试之间没有任何依赖，这也是我们接下来要说的测试的另外一个特点。

Independent, 独立的。测试和测试之间不应该有任何依赖，什么叫有依赖？比如，如果测试依赖于外部数据库或是第三方服务，测试 A 在运行时在数据库里写了一些值，测试 B 要用到数据库里的这些值，测试 B 必须在测试 A 之后运行，这就叫有依赖。

我们不能假设测试是按照编写顺序运行的。比如，有时为了加快测试运行速度，我们会将测试并行起来，在这种情况下，顺序是完全无法保证的。如果测试之间有依赖，就有可能出现各种问题。

减少外部依赖可以用 mock，实在要依赖，每个测试自己负责前置准备和后续清理。如果多个测试都有同样的准备和清理呢？那不就是 setup 和 teardown 发挥作用的地方吗？测试基础设施早就为我们做好了准备。

Professional, 专业的。这一点是很多人观念中缺失的，测试代码，也是代码，也要按照代码的标准去维护。这就意味着你的测试代码也要写得清晰，比如：良好的命名，把函数写小，要重构，甚至要抽象出测试的基础库，在 Web 测试中常见的 PageObject 模式，就是这种理念的延伸。

看了这点，你或许会想，你说的东西有点道理，但我的代码那么复杂，测试路径非常多，我怎么能够让自己的测试做到满足这些要求呢？

我必须强调一个之前讲测试驱动开发强调过的观点：**编写可测试的代码**。很多人写不好测试，或者觉得测试难写，关键就在于，你始终是站在写代码的视角，而不是写测试的视角。如果你都不重视测试，不给测试留好空间，测试怎么能做好呢？

总结时刻

测试是一个说起来很简单，但很不容易写好的东西。在实际工作中，很多人都会遇到关于测试的各种各样问题。之所以出现问题，主要是因为这些测试写得太复杂了。测试一旦复杂了，我们就很难保证测试的正确性，何谈用测试保证代码的正确性。

我给你讲了测试的基本结构：前置准备、执行、断言和清理，还介绍了一些常见的测试“坏味道”：做了太多事的测试，没有断言的测试，还有一种看一眼就知道有问题的“坏味道”，测试里有判断语句。

怎么衡量测试是否做好了呢？有一个标准：A-TRIP，这是五个单词的缩写，分别是 Automatic（自动化）、Thorough（全面）、Repeatable（可重复的）、Independent（独立的）和 Professional（专业的）。

如果今天的内容你只能记住一件事，那请记住：**要想写好测试，就要写简单的测试。**

最后，我想请你分享一下，经过最近持续对测试的讲解，你对测试有了哪些与之前不同的理解呢？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。

该试读文章来自付费专栏《10x程序员工作法》，如需阅读全部文章，
请订阅文章所属专栏，新人首单¥19.9

立即订阅



慢慢

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(26)



Kăfkă²⁰²⁰

可重复特别重要，有些开发在本地测和数据库相关应用时，由于前置依赖数据比较多，为了避免测试前写冗长的数据准备代码，所以会预先在数据库中准备好初始数据。每个测试再初始化特定的数据，因为Spring测试框架可以自动回滚，所以在本地是可以重复跑的。但是，放到CI中时，测试就统统没法过了，因为CI的数据库是共用的，没有本地的那份初始化数据集。一种方式是，保持数据库干净，用测试时用初始化脚本准备数据。如果测的场景比较复杂，比如要测多个事务的交互结果，还可以引入Docker，将依赖的数据库及初始化数据做成Docker的image，测试代码就更加简单，并且可以重复运行了，只要CI支持Docker即可

作者回复: 很好的分享!

2019-02-04

💬 3

👍 14



朱国伟

我还是习惯先写代码 在写测试 如 有一个投资机会详情 (opportunities/{id}) 的功能

首先是大的步骤任务拆解

- 查询机会基本信息
- 查询机会参与方 (标的方、投资方) 信息
- 相似推荐 即推荐与该机会类似的机会
- Domain ==> VO

然后是每个大步骤再细分 如

- 查询机会基本信息
 - 调用机会Service getByld() 获取基本信息
 - 调用机会Mappper getByld() 获取基本信息
 - 机会Mapper.xml 写查询sql
 - 机会Mapper单元测试
 - 准备机会数据 setUp/before中
 - 一个有效机会1
 - 一个无效机会2
 - 一个有效机会3
 - 测试getByld(1) 能正常查询 并且页面上需要展示字段都有
 - 测试getByld(2) asertNull()

然后后面就是按部就班的开发了 我是从上往下进行开发的(或者说从始至终进行开发的) 即先开发Service 再开发Mapper

开发完一个打个勾

- - 调用机会Service 获取基本信息

Mapper这一层的测试 直接连到是一个测试库 回滚依赖的是Spring自带的回滚机制

假设实际代码中只需查询 无需插入 那么会专门在该测试类中新建一个额外的Mapper 用于插入测试数据

如

```
class FooMapperTest {  
    @Before  
    public void setUp(){  
        testMapper.insert(1, 1, "name1");  
    }  
}
```

```
interface FooTestMapper{
    @Insert("...")
    void insert(int isvalid, int id, String name, ...)
}
}
```

service controller级别的测试 通过Mock的方式来测试

现在觉得是不是测的有点过于细了 比如 返回的行业 数据库中是逗号分割的字符串 返回给前端是一个行业数组 连VO中的的getIndustries()都测试了

```
assertNull(vo.getIndustries())
asserttArrayEquals(new String[]{"行业一","行业二"}, vo.getIndustries())
```

作者回复: 很赞的分享!

严格地说, 还不够细, 逗号分隔的字符串解析也应该拆出来。

程序员越舍不得在前期花时间, 就越要在后期花时间。

2019-03-09

💬 1

👍 11



行与修

本节课我有以下几点体会:

- 1、从开发者的视角看编码和测试是不分家的, 是可以通过重构形成良性生态圈的, 类似之前课程中的反馈模型和红绿重构模型;
- 2、A-TRIP是个很好的总结和行动指南, 在今后工作中应一以贯之, 把工作做到扎实有成效;
- 3、对文中提到的数据库依赖的问题, 我也说说自己的浅见。我觉得在测试代码中尽量避免与数据库打交道, 测试更关注领域与业务, 往往爆雷更多的是resource和service, 模型的变化往往牵动着表结构的变化, 与其两头兼顾不如多聚焦模型, 我常用的做法是用例配合若干小文件(数据忠实于模型), 保证库操作临门一脚前所有环节都是正确的, 同时方便适应变化。一旦出现异常, 也比较容易定位是否是数据库操作引发的问题。

(此点基于工作中发现项目型程序员大多是先急于把表结构定义出来, 好像不这么做写代码就不踏实)

作者回复: 很好的总结!

2019-02-04

2

8



俊伟

以前我一直觉得先开发完，再写测试。而现在，通过专栏学习让我明白了，要去站在测试的角度去写代码。首先写测试，然后再想办法去实现逻辑。写代码的时候要时刻记住“我的代码应该怎么写才可以通过测试”。

其次测试还要写的尽可能简单，一个测试只测试一个功能。测试还不能依赖外部的环境，测试可以重复运行，而结果要保持一致。测试也是也要符合代码的规范。测试还要确保覆盖所有情况，不能出现无断言的测试。

作者回复: 其实，测试驱动开发才是最好的以终为始案例。

2019-02-11

6

6



蓝士钦

单元测试不好写是因为代码本身耦合度太高不好测试，应该拆分成更小的可测试单元，避免出问题时的一个大方法内靠场景复现人肉debug，拆分耦合的代码本身需要一定的分析设计能力，尽量遵循SOLID原则。

修改某块没有单元测试的旧业务代码时应该提取并补上单元测试，证明自己的修改没有问题。保证后期能够依靠单元测试放心大胆的无脑修改复杂的业务逻辑。每次修改业务都小心翼翼的在头脑中debug运行一次效率是最低的，人是最不可靠的，应该靠单元测试覆盖各种边界条件。

最佳实践Test Pyramid证明研发自身做好单元测试和基于UI的自动化测试相比更加重要，写完代码应该自动验证。

作者回复: 这个理解非常好!

2020-07-19

3

3



钢之镇魂曲

我是游戏服务器开发工程师，我经历过不少公司，但是从来没见过写测试的。不知道是不是游戏有什么

特殊性？还是其他的什么问题？

作者回复: 没有特殊性，不写是一种现象，不是必然。当然，如果你问起，通常会有两类答案，没时间和我特殊。

2019-03-23

1

2



williamcai

原来一直以为开发之后，手动测试一下功能就ok了，原来开发之前把测试写好是多么的难

作者回复: 所谓的难，实际上是练习少。

2019-02-18

1

2



捞鱼的搬砖奇

测试不仅是测试人员的工作。更是开发人员的工作。之前的工作的中自测，常常潜意识的里只会考虑正常的情况，比如输入姓名的input，只会输入不超过三个字符的长度，到测试手冲，会输入一长串，因为程序中没有做长度检查，超过数据库字段长度成都就挂了。后来自己总结，发现测试人员的测试会带着破坏的性质，开发人员总是认为一切操作都是合理的。

看完了文章后，会继续完善之前的总结。把什么场景可能出现什么情况，罗列出来，方便工作中的对照检查。

作者回复: 程序员要学点测试知识，比如，测试等价类的划分，破坏性测试等等，当你开始重视测试了，代码质量才会提高。

2019-02-04

1

2



旭

老师春节快乐~ 开发和测试更像是矛盾的双方，对立但统一。之前做开发感觉测试影响了开发的效率，没事找事；后来接触测试感觉开发太过功利，只为实现而实现，实现不等于可用。矛与盾，同时在手，

或许才能更好的战斗。

作者回复: 所以, 要扩大自己的上下文。:)

2019-02-04



2



漂泊者及其影子

新年快乐, 基于spring的单元测试启动慢, 耗内存, 耗CPU, 怎么解决

作者回复: 涉及到Spring就不是单元测试, 至少是集成测试了, 参见前面的测试金字塔, 多写单元测试。集成测试慢点是可以接受的。

2019-02-04



2



陈斯佳

测试的步骤分为前置准备, 执行断言和清理, 我们要做到a trap, 也就是automatic, thorough, repeatable, independent, professional, 所以想要学好测试, 就要先写简单的测试。

2019-05-17



1



红糖白糖

1. 有断言 -- 就是测试时可以自判断的, 即测试自己知道成功还是失败, 不需要人工去判断。
2. 测试的写法: given -- when -- then
3. 测试的基础设施搭建好了之后, 测试写起来就会很快了。比如常用的使用buildXX(或者是DBUNIT)准备数据、tearDown清理数据。

作者回复: 总结得不错, 现在测试数据的组织还可以使用我的 Object Bot。
<https://github.com/dreamhead/object-bot>

2019-03-10



1



zhengfc

老师您好，如果方法足够简单的话，就可能导致大部分要不要测试的代码是私有方法，会有这问题吗？

作者回复: 首先，无论什么代码，只要是你写的，都应该测；其次，如果你是现在先写代码，后写测试的角度，才会考虑这个问题，先考虑怎么测，就不会问私有代码怎么测了。

2019-02-26



1



秦奋

自己的收获:

- 1.要写断言而不是采用打印结果的方式。这样既方便看出测试的结果，又节省测试的时间。当不能通过时能够及时的停止该测试用例的执行。
- 2.无论是被测代码还是测试代码，都要保证功能单一，不要做多件事情

2020-12-24



Y024

开源的都是技术项目，而大多数人做的都是业务开发，老师啥时候开源个业务的 Moco？

作者回复: 好建议，但有点难。

2020-11-10



mgs2002

老师，我想问下，service里面一个简单的功能比如查询XXX详情也需要单元测试吗
代码如下:

```
public BaseResult getTeamDetail(String teamId) {
```

```
    TeamResponse team = getBaseMapper().selectTeamDetailById(new BigDecimal(teamId));  
    return new BaseResult().success(BaseResultCodeEnum.SUCCESS.getMessage(),team);
```

```
}
```

就是一个简单的SQL查询然后返回，这种需要写测试吗

作者回复: 首先，答案是需要。其次，你这里没有把数据库访问封装起来，如果封装起来的话，就不用暴露mapper了，模拟一个接口方法就好。

2020-07-28



丁丁历险记

没用永恒的银蛋，千万别指望测试解决架构设计的问题。

作者回复: 没有银弹

2019-11-08



学习使人快乐

明白了测试代码的重要性

2019-10-29



hua168

老师，idea测试用JUnit5还是其它比较方便？

作者回复: IDEA 对 JUnit 的4和5支持得都挺好。

2019-03-29



enjoylearning

测试中的3A原则么，还有测试代码中的for循环要如何避免，难道抽出来放到测试辅助类里去？

作者回复: 可以用测试的3A原则来理解。

测试里就不应该有 for 循环, 你为什么要有那么多结果去检查呢? 用少量数据也是可以的, 你需要理解测试等价类的概念。

2019-03-28



號國技醬

打卡

2019-03-15



梦倚栏杆

从数据库或者第三方api查询类内容需要写测试吗? 这种测试怎么写呢?

如果不需要写, 两会发现大量展示类系统不需要写测试了, 感觉怪怪的

作者回复: 重点在于要测什么, 如果是测试别人的东西, 那就成了给别人写测试。如果是测试自己的系统, 可能你需要的是一个 Mock, 模拟对方的行为。这个时候, 可以试试 Moco。

<https://github.com/dreamhead/moco>

2019-03-11



zhengfc

多打了了不字

2019-02-26



九月三秋

让我感觉到写测试是相当重要

2019-02-11



行与修

祝新年快乐!

编辑回复: 新年快乐 🎉

2019-02-04



非鱼

收益了, 新年快乐

2019-02-04

