

44 | 测试先行：测试驱动开发(TDD)

2018-10-08 茹炳晟

软件测试52讲

[进入课程 >](#)



讲述：茹炳晟

时长 12:12 大小 5.59M



你好，我是茹炳晟。今天我和你分享的主题是“测试先行：测试驱动开发（TDD）”。

通过上一篇文章，我们已经深入理解了什么是探索式测试，以及如何用探索式测试开展具体的测试。今天我这次分享的目的，就是和你聊聊软件测试领域中的另一个很热门的话题：测试驱动开发，也就是 Test-Driven Development，通常简称为 TDD。

听上去有些迷惑是不是？测试怎么可能驱动开发呢？在传统软件的开发流程中，软件开发人员先开发好功能代码，再针对这些功能设计测试用例、实现测试脚本，以此保证开发的这些功能的正确性和稳定性。那么，TDD 从字面上理解就是要让测试先行，这又是怎么回事呢？

确切地说，TDD 并不是一门技术，而是一种开发理念。它的核心思想，是在开发人员实现功能代码前，先设计好测试用例的代码，然后再根据测试用例的代码编写产品的功能代码，最终目的是让开发前设计的测试用例代码都能够顺利执行通过。

这样对于开发人员来说，他就需要参与到这个功能的完整设计过程中，而不是凭自己想象去开发一个功能。他有一个非常明确的目标，就是要让提前设计的测试用例都可以顺利通过，为此，他先实现测试用例要求的功能，再通过不断修改和完善，让产品代码可以满足测试用例，可以说是“小而美”的开发过程。

所以，从本质上来讲，TDD 并不属于测试技术的范畴。那么，我为什么还要单独用一篇文章和你分享这个主题呢？因为，TDD 中通常会用到很多常见的自动化测试技术，使得测试在整个软件生命周期中的重要性和地位得到了大幅提升。

可以说，TDD 的思想和理念给软件开发流程带来了颠覆性的变化，使得测试工作从原本软件研发生命周期的最后端走向了最前端。也就是说，原本测试工作是软件研发生命周期最后的一个环节，而现在 TDD 相当于把测试提到了需求定义的高度，跑到了软件研发生命周期最前面。

那么，接下来我们就一起看看 TDD 的优势有哪些，以及 TDD 的具体实施过程。

TDD 的优势

TDD 的优势，可以概括为以下五个方面：

1. 保证开发的功能一定是符合实际需求的。

用户需求才应该是软件开发的源头，但在实际的软件开发过程中，往往会在不知情的情况下，或者自己的主观判断下，开发出一个完全没有实际应用场景的功能。而这些没有实际应用场景的功能，却因为产品验证和测试工作介入的时机都在项目后期，所以往往在集成测试中或者产品上线后才会被发现。

比如，开发人员在实现用户注册的功能时，认为需要提供使用手机号注册的功能。但是，这个功能开发完成后，测试人员却告知开发人员这个功能用不上，或者产品上线后才发现这个功能在实际场景中完全不是必须的，因为用户可以使用邮箱注册，然后再通过绑定手机号实现手机号登陆。所以，直接用手机号注册这个功能是不需要的，真正需要的是绑定邮箱和手机号的功能。

试想一下，如果是测试驱动开发，即先根据用户的实际需求编写测试用例，再根据测试用例来完成功能代码，就不会出现这种既浪费时间、精力，又没有必要的功能了。

2. 更加灵活的迭代方式。

传统的需求文档，往往会从比较高的层次去描述功能。开发人员面对这种抽象的需求文档，往往会感觉无从下手。但是，在 TDD 的流程里，需求是以测试用例描述的，非常具体。那么，开发人员拿到这样的需求时，就可以先开发一个很明确的、针对用户某一个小需求的功能代码。

在开发过程中，开发人员可以不断的调试这个功能，通过测试 -> 失败 - 修改 / 重构 -> 测试 -> 成功的过程，使开发的代码符合预期，而不是等所有功能开发完成后，再将一个笨重的产品交给测试人员进行一个长周期的测试，发现缺陷后再整个打回来修改，然后由此又可能会引入新的缺陷。

另外，如果用户需求有变化，我们能够很快地定位到要修改的功能，从而实现快速修改。

3. 保证系统的可扩展性。

为了满足测试先行的灵活迭代方式，我们会要求开发人员设计更松耦合的系统，以保证它的可扩展性和易修改性。这就要求，开发人员在设计系统时，要考虑它的整体架构，搭建系统的骨架，提供规范的接口定义而非具体的功能类。

这样，当用户需求有变化时，或者有新增测试用例时，能够通过设计的接口快速实现新功能，满足新的测试场景。

4. 更好的质量保证。

TDD 要求测试先于开发，也就是说在每次新增功能时，都需要先用测试用例去验证功能是否运行正常，并运行所有的测试来保证整个系统的质量。在这个测试先行的过程中，开发人员会不断调试功能模块、优化设计、重构代码，使其能够满足所有测试场景。所以，很多的代码实现缺陷和系统设计漏洞，都会在这个不断调优的过程中暴露出来。

也就是说，TDD 可以保证更好的产品质量。

5. 测试用例即文档。

因为在 TDD 过程中编写的测试用例，首先一定是贴合用户实际需求的，然后又在开发调试的过程中经过了千锤百炼，即一定是符合系统的业务逻辑的，所以我们直接将测试用例生成需求文档。

这里，直接将测试用例生成需求文档的方法有很多、很简单的方法，比如 JavaDoc。

这样，我们就无须再花费额外的精力，去撰写需求文档了。

你看，TDD 真的是优势多多吧。那么，接下来我们就一起来看看实施 TDD 的具体过程。

测试驱动开发的实施过程

站在全局的角度来看，TDD 的整个过程遵循以下流程：


1. 为需要实现的新功能添加一批测试；
2. 运行所有测试，看看新添加的测试是否失败；
3. 编写实现软件新功能的实现代码；
4. 再次运行所有的测试，看是否有测试失败；
5. 重构代码；
6. 重复以上步骤直到所有测试通过。

接下来，我们就通过一个具体的例子，来看看 TDD 的整个流程吧。

我们现在要实现这么一个功能：用户输入自己的生日，就可以输出还要多少天到下次生日。

根据 TDD 测试先行的原则，我们**首先要做的是设计测试用例**。

测试用例一，用户输入空字符串或者 null：

 复制代码


```
1 @Test
2 // 测试输入空字符串 null 时，是否抛出 "Birthday should not be null or empty" 异常
3 public void birthdayIsNull() {
4     RuntimeException exception = null;
5     try {
```

```

6         BirthdayCaculator.caculate(null);
7     }catch(RuntimeException e) {
8         exception = e;
9     }
10    Assert.assertNotNull(exception);
11    Assert.assertEquals(exception.getMessage(), "Birthday should not be null or empty")
12 }
13
14 @Test
15 // 测试输入空字符串 "" 时, 是否抛出 "Birthday should not be null or empty" 异常
16 public void birthdayIsEmpty() {
17     RuntimeException exception = null;
18     try {
19         BirthdayCaculator.caculate("");
20     }catch(RuntimeException e) {
21         exception = e;
22     }
23     Assert.assertNotNull(exception);
24     Assert.assertEquals(exception.getMessage(), "Birthday should not be null or empty")
25 }

```

根据这个测试用例, 我们可以很容易地写出这部分的 Java 代码:


 复制代码

```

1 public static int caculate(String birthday) {
2     if(birthday == null || birthday.isEmpty()) {
3         throw new RuntimeException("Birthday should not be null or empty");
4     }
5 }

```

测试用例二, 用户输入的生日格式不符合 YYYY-MM-dd 的格式:

 复制代码


```

1 @Test
2 // 测试输入错误的时间格式, 是否抛出 "Birthday format is invalid!" 异常
3 public void birthdayFormatIsInvalid() {
4     RuntimeException exception = null;
5     try {
6         BirthdayCaculator.caculate("Sep 3, 1996");
7     }catch(RuntimeException e) {
8         exception = e;
9     }
10    Assert.assertNotNull(exception);

```

```
11     Assert.assertEquals(exception.getMessage(), "Birthday format is invalid! ");
12 }
```

那么，这部分的 Java 代码实现便要 catch 住 ParseException, 重新自定义错误信息并抛出异常。

 复制代码


```
1 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
2 Calendar birthDate = Calendar.getInstance();
3 try {
4     // 使用 SimpleDateFormat 来格式化输入日期值
5     birthDate.setTime(sdf.parse(birthday));
6 } catch (ParseException e) {
7     throw new RuntimeException("Birthday format is invalid!");
8 }
```

测试用例三，用户输入的生日格式正确，但是今年的生日已经过了，就应该返回离明年的生日还有多少天：

 复制代码

```
1 @Test
2 // 测试用户输入的日期晚于今年生日的情况，判断是否返回离明年的生日有多少天
3 public void thisYearBirthdayPassed() {
4     Calendar birthday = Calendar.getInstance();
5     birthday.add(Calendar.DATE, -1);
6     SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-dd");
7     String date = sdf.format(birthday.getTime());
8     int days = BirthdayCaculator.caculate(date);
9     // 天数不应该出现负数
10    Assert.assertTrue(days > 0);
11 }
```

测试用例四，用户输入的生日格式正确且今年生日还没过，返回的结果应该不大于 365 天：

 复制代码


```
1 @Test
```

```

2 // 测试用户输入的日期早于今年生日的情况，判断返回的天数是否小于 365
3 public void thisYearBirthdayNotPass() {
4     Calendar birthday = Calendar.getInstance();
5     birthday.add(Calendar.DATE, 5);
6     SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-dd");
7     String date = sdf.format(birthday.getTime());
8     int days = BirthdayCaculator.caculate(date);
9     // 天数不应该大于一年的天数，365 天
10    Assert.assertTrue(days < 365);
11 }

```

测试用例五，用户输入的生日格式正确并且是今天，返回的结果应该为 0：


 复制代码

```

1 @Test
2 // 测试用户输入的日期恰好等于今年生日的情况，判断返回的天数是否是 0
3 public void todayIsBirthday() {
4     Calendar birthday = Calendar.getInstance();
5     SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-dd");
6     String date = sdf.format(birthday.getTime());
7     int days = BirthdayCaculator.caculate(date);
8     Assert.assertEquals(days, 0);
9 }

```

综合上述五种测试场景，根据测试用例，我们可以编写完整的功能代码覆盖所有类型的用户输入，完整代码如下：

 复制代码

```

1 public static int caculate(String birthday) {
2     // 首先对输入的日期是否是 null 或者是 "" 进行判断
3     if(birthday == null || birthday.isEmpty()) {
4         throw new RuntimeException("Birthday should not be null or empty");
5     }
6
7     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
8     Calendar today = Calendar.getInstance();
9
10    // 处理输入的日期恰好等于今年生日的情况
11    if(birthday.equals(sdf.format(today.getTime())) {
12        return 0;
13    }
14

```



```
15 // 输入日期格式的有效性检查
16 Calendar birthDate = Calendar.getInstance();
17 try {
18     birthDate.setTime(sdf.parse(birthday));
19 } catch (ParseException e) {
20     throw new RuntimeException("Birthday format is invalid!");
21 }
22 birthDate.set(Calendar.YEAR, today.get(Calendar.YEAR));
23
24 // 实际计算的逻辑
25 int days;
26 if (birthDate.get(Calendar.DAY_OF_YEAR) < today.get(Calendar.DAY_OF_YEAR)) {
27     days = today.getActualMaximum(Calendar.DAY_OF_YEAR) - today.get(Calendar.DAY_OF_
28     days += birthDate.get(Calendar.DAY_OF_YEAR);
29 } else {
30     days = birthDate.get(Calendar.DAY_OF_YEAR) - today.get(Calendar.DAY_OF_YEAR);
31 }
32 return days;
33 }
```

以上场景，每添加一个新的功能点，都会添加一个测试方法；完成新功能点的软件代码后，接着运行当前所有的测试用例，以保证新加的功能代码能够满足现有的测试需求。这就是一个典型的 TDD 过程了。但是，在实际开发场景，肯定会更复杂，你想要用 TDD 思想写出健壮稳定的代码，就需要深入理解 TDD 中的每一步。

首先，需要控制 TDD 测试用例的粒度。如果测试用例并不是最小粒度的单元测试，开发人员就不能不假思索地直接根据测试用例开发功能代码，而应该先把测试用例分解成更小粒度的任务列表，保证每一个任务列表都是一个最小的功能模块。

在开发过程中，要把测试用例当成用户，不断分析他可能会怎样调用这个功能，大到功能的设计是用类还是接口，小到方法的参数类型，都要充分考虑到用户的使用场景。

其次，要注意代码的简洁和高效。随着功能代码的增加，开发人员为了让测试能顺利通过，很可能会简单粗暴地使用复制粘贴来完成某个功能，而这就违背了 TDD 的初衷，本来是为了写出更优雅的代码，结果反而造成了代码冗余混乱。因此，在开发 - 测试循环过程中，我们要不断地检查代码，时刻注意是否有重复代码、以及不需要的功能，将功能代码变得更加高效优雅。

最后，通过重构保证最终交付代码的优雅和简洁。所有功能代码都完成，所有测试都通过之后，我们就要考虑重构了。这里可以考虑类名、方法名甚至变量名命名，是否规范且有意

义，太长的类可以考虑拆分；从系统角度检查是否有重复代码，是否有可以合并的代码，你也可以参考市面上比较权威的关于重构的书完成整个系统的重构和优化。这里我建议你阅读 Martin Fowler 的《重构：改善既有代码的设计》这本书。

总的来说，TDD 有其优于传统开发的特点，但在实际开发过程中，我们应该具体场景具体分析。

比如，最典型的一个场景就是，一个旧系统需要翻新重做，并且针对这个老系统已经有很多不错的测试用例了，这就很适合选用 TDD。

总之，我们可以通过分析当前的时间、人、方式、效果各要素来最终决定是否选用 TDD。另外，需要特别注意的是，选用 TDD 并不是测试人员或者测试部门的事情，而是需要公司层面的流程和体系的配合，也正是这种原因，虽然大家都能看到 TDD 的优势，但是在实际项目中的运用还是比较有限。

总结

今天我和你分享了测试驱动开发核心理念，以及 TDD 的优势。

TDD 的核心思想便是在开发人员实现功能代码前，先设计好测试用例，编写测试代码，然后再针对新增的测试代码来编写产品的功能代码，最终目的是让新增的测试代码能够通过。

相对于传统软件开发流程，TDD 的优势主要包括对需求精准的把控、更灵活的迭代、促使更好的系统设计、更好的交付质量以及轻量级的文档等。

最后，我用“用户输入自己的生日，就可以输出还要多少天到下次生日”作为例子，展示了测试驱动开发的完整流程，希望帮助你对 TDD 有更直观的认识。

思考题

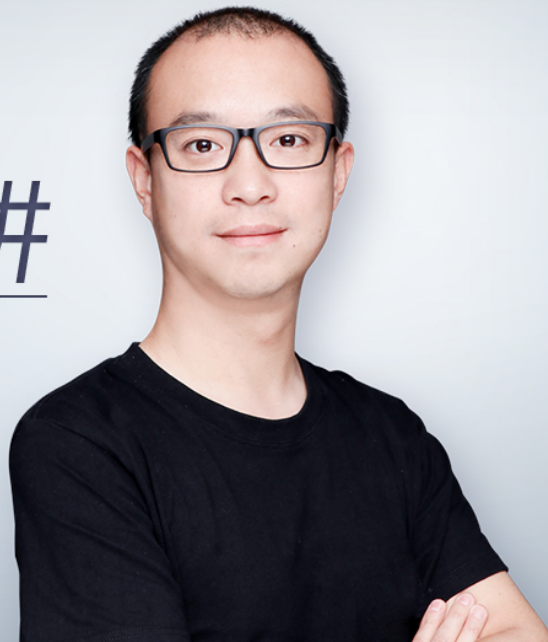
在实际的工程项目中，你实际使用过 TDD 吗？如果有的话，是否可以分享一下你的实践心得？如果没有的话，你是否可以设想一下你会怎么规划和设计一个 TDD 的项目？

感谢你的收听，欢迎你给我留言一起讨论。

软件测试52讲

从小工到专家的实战心法

茹炳晟 eBay中国研发中心
测试基础架构技术主管



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 43 | 发挥人的潜能：探索式测试

下一篇 45 | 打蛇打七寸：精准测试

精选留言 (15)

写留言



叶夏立

2018-10-08

3

tdd怎么样做才能落实到项目中，我觉得这才是核心问题，当然不是所有的项目都适合tdd。不知道茹老师是否能分享一下tdd落地推动的做法？

作者回复：很高的问题，首先就像你说的，不是所有的项目都适合tdd，而且采用tdd对测试人员的要求会很高。我的建议是一些小型的poc项目，或者是功能相对单一的微服务开发是比较适合tdd的。另外，要推动tdd，一定需要改革整个研发的流程，这个往往是十分困难的，也正是这个原因，实际开展tdd的项目也不是很多。



秦浩然
2019-04-09

👍 1

虽然 TDD 并不适合所有项目，但是将 TDD 思想放大到整个开发流程上，我总结了一套开发流程，请大家参考。

所有人员参与需求评审 -> 测试人员编写测试用例 -> 所有人员参与用例评审 -> 开发人员按照测试用例进行编码 -> 开发人员执行用例，进行自测，所有用例通过后 -> 开发人员...
展开 ▾



秦浩然
2019-04-08

👍 1

确实要考虑项目的适用性，如果对于试水项目、用户需求不确定的，就不太合适了。后期需求频繁变更的话，测试的维护成本也是很高的。



我是谁
2019-02-18

👍 1

tdd感觉就是详细的单元测试，那对于测试用例的项目建立，包括持续集成，都是由测试来做。开发人员是不是就不需要写单元测试了，那开发自测用测试人员写的测试用例吗，是拉去测试这边项目吗

展开 ▾



涅槃Ls
2018-10-09

👍 1

打卡44，国庆节后 好好学习

展开 ▾

作者回复: 支持打卡👍



仰望星空
2018-10-08

👍 1

老师讲的很系统，每篇都听，几乎涵盖了测试的方方面面。有一点就是设计安全性方面的测试能否也讲一讲呢

作者回复: 感谢支持，后面马上会有讲渗透测试的文章



口水窝

2019-05-17



对于TDD，我有两点要说。1是TDD对于测试人员的要求较高，至少要会白盒测试，而且测试用例的粒度，是否有遗漏，测试用例代码通过率都有要求，这对于很多企业都觉得测试职业都是点点点的公司来说，根本做不到，而且从领导层面上都没有这个意识去落实的。2是TDD的推动要从领导层，甚至公司技术部的最高层从上而下去推动，这才是最难执行的，这一点也和作者想法一样，所以现实中很多无法实现。

展开 ∨



subona

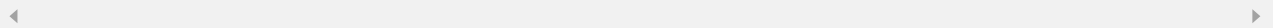
2018-12-24



tdd测试代码都是单元测试了吧

展开 ∨

作者回复: 都是直接面向代码的



小老鼠

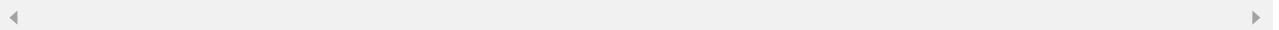
2018-11-29



您是不是把TDD、BDD、ATDD混在一起了 😊

展开 ∨

作者回复: 本质上这三个是不同层面的东西，但是出发点和思路是异曲同工的



郭小菜

2018-11-07



个人认为先写测试代码比较适合单一场景，如果是较为复杂业务场景先去写测试代码是很复杂的，测试代码的数量甚至多余系统代码



木宇寒影

2018-10-29



如果代码能力不高，测试驱动开发可以先从excel用例入手，先保证开发出的功能都是符合要求的



喵呜呀呵嘿...

2018-10-15



测试人员的综合能力强于开发人员，感觉TDD会更好推行也适合使用。相反则不然吧。



~黑凤梨~

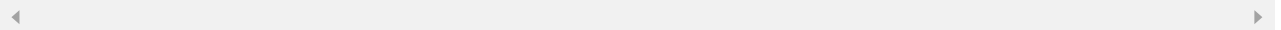
2018-10-09



我们WEB项目也在要求做TDD，并将TDD与现有的CICD（其实只有CD）结合，不知道具体如何来管理TDD的test case那些东西。希望老师指点一下，谢谢。

展开 ∨

作者回复: 严格来说，前端项目不一定适合tdd，更适合的应该是bdd，当然和jenkins之类的工具结合本身的确是个好方法，这种情况下，测试用例也都需要通过代码仓库来集中版本化管理



潘达

2018-10-09



我们WEB项目也在要求做TDD，但是开发做的极其敷衍，后补TDD的test case已经算是好的了。有想法将TDD与现有的CICD（其实只有CD）结合，不知道具体如何来管理TDD的test case那些东西。希望老师指点一下，谢谢。

展开 ∨



伪专家

2018-10-08



没有强的coding能力，不行的

展开 ∨

作者回复: 是的，tdd一定要求有很好的代码能力。



