

## 14 | 大师级程序员的工作秘笈

2019-01-30 郑晔

10x程序员工作法

[进入课程 >](#)



讲述：郑晔

时长 14:01 大小 12.84M



前面我和大家分享了 TDD 的来龙去脉，那些尚未将 TDD 烂熟于胸的同学会分为两个派别。一派是摩拳擦掌，准备动手实践一番；另一派是早就自我修炼过，但实践之路不通。所以，市面上经常会听到有人说，TDD 不实用。

但是 TDD 真的不实用吗？

和任何一门技能一样，TDD 也是需要练习的。更重要的是，你需要打通 TDD 的“任督二脉”，而这关键正是我们这个模块的主题：任务分解。而且，在今天的 content 中，我还将带你领略大师级程序员的工作风范。让我们开始吧！

### TDD 从何而来？

要学最原汁原味的 TDD，莫过于从源头学起。

从前 TDD 只在小圈子里流行，真正让它在行业里广为人知的是 Kent Beck 那本知名的软件工程之作 [《解析极限编程》](#) (Extreme Programming Explained)。这是一本重要的作品，它介绍了一种软件开发方法：[极限编程](#)。

当年他写作之时，许多人都在努力探寻瀑布开发方法之外的软件工程方法，除了极限编程，还有[特征驱动开发](#)、[水晶开发方法](#)等等，正是这些开发方法的探索，才有了后面敏捷方法的诞生。

极限编程对于行业最大的贡献在于，它引入了大量的实践，比如，前面提到过的持续集成、这里提到的 TDD，还有诸如结对编程、现场客户等等。

极限编程之所以叫“极限”，它背后的理念就是把好的实践推向极限。

前面提到持续集成时，我们已经介绍过这个理念，如果集成是好的，我们就尽早集成，推向极限每一次修改都集成，这就是持续集成。

如果开发者测试是好的，我们就尽早测试，推向极限就是先写测试，再根据测试调整代码，这就是测试驱动开发。

如果代码评审是好的，我们就多做评审，推向极限就是随时随地地代码评审，这就是结对编程。

如果客户交流是好的，我们就和客户多交流，推向极限就是客户与开发团队时时刻刻在一起，这就是现场客户。这种极限思维是一种很好的思考问题方式，推荐你也在工作中尝试使用一下。

虽然 TDD 只是《解析极限编程》介绍的诸多实践的一种，它却是与开发人员关系最为密切的一个实践。

随着 TDD 逐渐流行开来，人们对如何做 TDD 也越来越感兴趣，于是，Kent Beck 又专门为 TDD 写了一本书，叫 [《测试驱动开发》](#)。

## 大师级程序员的秘笈

《测试驱动开发》这本书很有意思。如果你只是为了了解 TDD，这本书可能很无聊。Kent Beck 在第一部分只是在写一个功能，写完一段又写一段。

这本书我看过两遍，第一遍觉得平淡无奇，这种代码我也能写。第二遍看懂他的思路时，我几乎是震惊的感觉，因为它完全是在展示 Kent Beck 的工作方式。这也是我把 TDD 放到这个部分来讲的重要原因，Kent Beck 在做的就是任务分解。任务分解，也是这本书的真正价值所在。

当时，我已经工作了很多年，自以为自己在写代码上已经很专业了。看懂 Kent Beck 的思路，我才知道，与他相比，我还不够专业。

Kent Beck 是怎么做的呢？每当遇到一件要做的事，Kent Beck 总会先把它分解成几个小任务，记在一个清单上，然后，才是动手写测试、写代码、重构这样一个小循环。等一个循环完成了，他会划掉已经做完的任务，开始下一个。

一旦在解决问题的过程中遇到任何新的问题，他会把这个要解决的问题记录在清单上，保证问题不会丢失，然后，继续回到自己正在处理的任务上。当他把一个个任务完成的时候，问题就解决完了。

你或许会纳闷，这有什么特别的吗？你不妨回答这样一个问题，你多长时间能够提交一次代码？如果你的答案超过半天，对不起，你的做法步子一定是太大了。你之所以不能小步提交，一定是牵扯了太多相关的部分。

**Kent Beck 的做法清晰而有节奏，每个任务完成之后，代码都是可以提交的。**看上去很简单，但这是大多数程序员做不到的。

只有把任务分解到很小，才有可能做到小步提交。你能把任务分解到很小，其实是证明你已经想清楚了。**而大多数程序员之所以开发效率低，很多时候是没想清楚就动手了。**

我在 ThoughtWorks 工作时，每个人都会有个 Sponsor，类似于工厂里师傅带徒弟的关系。我当时的 Sponsor 是 ThoughtWorks 现任的 CEO 郭晓，他也是写代码出身的。有一次，他给我讲了他和 Wiki 的发明者 Ward Cunningham 一起结对编程的场景。

Ward 每天拿到一个需求，他并不急于写代码，而是和郭晓一起做任务分解，分解到每个任务都很清晰了，才开始动手做。接下来就简单了，一个任务一个任务完成就好了。

当时，郭晓虽然觉得工作节奏很紧张，但思路则是非常清晰的。有时，他也很奇怪，因为在开始工作之前，他会觉得那个问题非常难以解决。结果一路分解下来，每一步都是清晰的，也没遇到什么困难就完成了。

之所以这里要和你讲 Ward Cunningham 的故事，因为他就是当年和 Kent Beck 在同一个圈子里一起探讨进步的人，所以，在解决问题的思路，二人如出一辙。

为什么任务分解对于 TDD 如此重要呢？因为只有当任务拆解得足够小了，你才能知道怎么写测试。

**很多人看了一些 TDD 的练习觉得很简单，但自己动起手来却不知道如何下手。中间就是缺了任务分解的环节。**

任务分解是个好习惯，但想要掌握好它，大量的练习是必须的。我自己也着实花不少时间进行练习，每接到一个任务，我都会先做任务分解，想着怎么把它拆成一步一步可以完成的小任务，之后再动手解决。

## 微操作

随着我在任务分解上练习的增多，我越发理解任务分解的关键在于：**小**。

小到什么程度呢？有时甚至可以小到你可能认为这件事不值得成为一件独立的事。比如升级一个依赖的版本，做一次变量改名。

这样做的好处是什么呢？它保证了我可以随时停下来。

我曾在一本书里读到过关于著名高尔夫球手“老虎”伍兹的故事。高尔夫球手在打球的时候，可能会受到一些外界干扰。一般情况下还好，如果他已经开始挥杆，这时候受到了干扰，一般选手肯定是继续把杆挥下去，但通常的结果是打得不理想。

而伍兹遇到这种情况，他会停下来，重新做挥杆的动作，保证了每一杆动作的标准。

伍兹能停下来，固然是经过了大量的练习，但还有一个关键在于，对于别人而言，挥杆击球是一个动作，必须一气呵成。而对伍兹来说，这个动作是由若干小动作组成的，他只不过是刚好完成了某个小动作，而没有做下一个小动作而已。

换句话说，大家同样都是完成一个原子操作，只不过，伍兹的原子操作比其他人的原子操作小得多。

同样，我们写程序的时候，都不喜欢被打扰，因为一旦被打扰，接续上状态需要很长一段时间，毕竟，我们可不像操作系统那么容易进行上下文切换。

但如果任务足够小，完成一个任务，我们选择可以进入到下一个任务，也可以停下来。这样，即便被打扰，我们也可以很快收尾一个任务，不致于被影响太多。

其实，这种极其微小的原子操作在其他一些领域也有着自己的应用。有一种实践叫微习惯，以常见的健身为例，很多人难以坚持，主要是人们一想到健身，就会想到汗如雨下的健身场景，想想就放弃了。

但如果你一次只做一个俯卧撑呢？对大多数人来说，这就不是很难的一件事，那就先做一个。做完了一个如果你还想做，就接着做，不想做就不做了。

一个俯卧撑？你会说这也叫健身，一个俯卧撑确实是一个很小的动作，重要的是，一个俯卧撑是你坚持完成的，如果每天做 10 个，恐怕这都是大多数人做不到的。我们知道，养成一个习惯，最难的是坚持。**如果你有了一个微习惯，坚持就不难了。**

我曾经在 github 上连续提交代码 1000 天，这是什么概念？差不多三年的时间里，每天我都能坚持写代码，提交代码，这还不算工作上写的代码。

对于大多数人来说，这是不可思议的。但我坚持做到了，不是因为我有多了不起，而是我养成了自己的微习惯。

这个连续提交的基础，就是我自己练习任务分解时，不断地尝试把一件事拆细，这样，我每天都至少能保证完成一小步。当然，如果有时间了，我也会多写一点。正是通过这样的方法，我坚持了 1000 天，也熟练掌握了任务分解的技巧。

**一个经过分解后的任务，需要关注的内容是有限的，我们就可以针对着这个任务，把方方面面的细节想得更加清晰。**很多人写代码之所以漏洞百出，一个重要的原因就是任务粒度太大。

我们作为一个普通人，能考虑问题的规模是有限的，也就很难方方面面都考虑仔细。

## 微操作与分支模型

经过这种练习之后，任务分解也就成了我的本能，不再局限于写程序上。我遇到任何需要解决的问题，脑子里的第一反应一定是，它可以怎么一步一步地完成，确定好分解之后，解决问题就是一步一步做了。

如果不能很好地分解，那说明我还没想清楚，还需要更多信息，或者需要找到更好的解决方案。

一旦你懂得了把任务分解的重要性，甚至通过训练能达到微操作的水准，你就很容易理解一些因为步子太大带来的问题。举一个在开发中常见的问题，代码开发的分支策略。

关于分支策略，行业里有很多不同的做法。有的团队是大家都在一个分支上写代码，有的是每个人拉出一个分支，写完了代码再合并回去。你有没有想过为什么会出现这种差异呢？

行业中的最佳实践是，基于主分支的模型。大家都在同一个分支上进行开发，毕竟拉分支是一个麻烦事，虽然 git 的出现极大地降低了拉分支的成本。

但为什么还有人要拉出一个分支进行开发呢？多半的原因是他写的代码太多了，改动量太大，很难很快地合到开发的主分支上来。

那下一个问题就来了，为什么他会写那么多代码，没错，答案就是步子太大了。

如果你懂得任务分解，每一个分解出来的任务要改动的代码都不会太多，影响都在一个可控的范围内，代码都可以很快地合并到开发的主分支上，也就没有必要拉分支了。

在我的实际工作中，我带的团队基本上都会采用基于主分支的策略。只有在做一些实验的时候，才会拉出一个开发分支来，但它并不是常态。

## 总结时刻

TDD 在很多人眼中是不实用的，一来他们并不理解测试“驱动”开发的含义，但更重要的是，他们很少会做任务分解。而任务分解是做好 TDD 的关键点。只有把任务分解到可以测试的地步，才能够有针对性地写测试。



同样听到任务分解这个说法，不同的人理解依然是不一样的。我把任务分解的结果定义成微操作，它远比大多数人理解得小。我们能将任务分解到多小，就决定了我们原子操作的粒度是多大。软件开发中的许多问题正是由于粒度太大造成的，比如，分支策略。

如果今天的内容你只能记住一件事，那请记住：**将任务拆小，越小越好。**

最后，我想请你分享一下，你身边是否有一些由于任务分解得不够小带来的问题。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。



# 10x 程序员工作法

掌握主动权，忙到点子上

郑晔

火币网首席架构师  
前 ThoughtWorks 首席咨询师  
TGO 鲲鹏会会员



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 13 | 先写测试，就是测试驱动开发吗？

下一篇 15 | 一起练习：手把手带你分解任务

精选留言 (25)

写留言





**Y024**  
2019-02-03

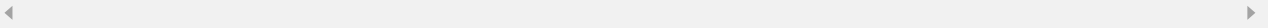
👍 8

Feature toggle (功能开关) 分享两篇文章:  
<https://martinfowler.com/articles/feature-toggles.html>

<https://www.infoq.cn/article/function-switch-realize-better-continuous-implementations>

展开 ▾

作者回复: 多谢补充!



**葛晨**  
2019-01-30

👍 5

很受启发

展开 ▾

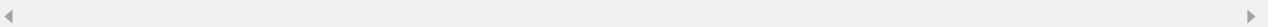


**~~**  
2019-01-30

👍 3

老师好, 关于基于主分支的策略上线流程是怎麼样的, 假如两个人同时在开发两个功能, 并不不断的进行小的粒度提交, 那其中一个人完成功能要上线的话, 另一个人的功能才完成一部分, 待主干中已经有他那没完成的任务的代码了, 是要一块发布吗

作者回复: 这种问题的常见解决方案是Feature Toggle。



**Geek\_fe033...**  
2019-03-22

👍 1

想起某个广告词, 每天一小步, 向上走就是新高度。小步快跑是关键

展开 ▾



**如明如月**  
2019-01-31

👍 1

对任务分解的体会非常深刻, 刚入职的时候任务评估不准。现在想想主要是两个原因: 1、需求梳理的不清晰, 还没清楚地搞明白需求就动手写代码, 导致返工、导致一些“意想不



到”的情况。2、任务分解做的不好，没有将任务分解成非常清晰地可执行的单元，导致有些时候无从下手，而且任务时间评估不准确。

展开 ∨



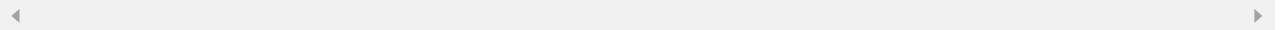
**划时代**

2019-01-31

👍 1

听了老师的音频讲解，有种醍醐灌顶的感觉，很多以前没有想明白的问题，都得到了答案，感谢！

编辑回复: 加油💪



**banyejing...**

2019-05-16

👍

讲任务原子化，拆分的越小，越容易做到

展开 ∨



**kyo**

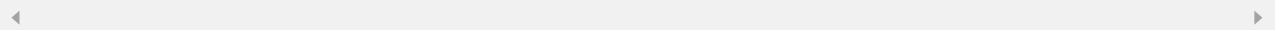
2019-05-09

👍

Feature Toggle 并不算一个好的解决方案. 我们使用的是频繁的分支合并. 每次 master 分支有提交会自动向其他分支合并, 这样有冲突就可以第一时间发现了.

展开 ∨

作者回复: 这取决于你怎么定义“频繁”，如果真的能够每天都提交，分支也可以接受，但如果真能每天都提交，你还会开分支吗？



**邵俊达**

2019-04-18

👍

又发现了几个任务分解都好处，

1. 让自己的工作可以被量化。
2. 可以加强对任务估算的能力。
3. 可以掌控项目的进度。
4. 每完成一组任务就可以提交一个 PR，这时就可以去休息一下。之前是一直坐到「天...

展开 ▾

作者回复: 很高兴看到了你的进步!

◀ ▶



小龙

2019-03-22



看了这么多，忍不住发个言：这是我定过的所有极客专栏里 写的最有诚意、最有价值的一个！

作者回复: 欢迎把它分享给你的朋友!

◀ ▶



红糖白糖

2019-03-10



baby step~

主干分支用着委实不错。但有几个点，一个是在已用线上产品的时候，我们通常会有一条 release 分支。这样当有线上 bug 出现的时候，我们能在 release 分支上快速修复 + 上线。另外一个，在规律性的迭代上线时，难免会遇到有 story 没有完成(开发或者测试)，此...

展开 ▾



苦行僧

2019-03-08



任务足够小的操作也方便切换 这个思路不错

展开 ▾



Wei

2019-03-05



首先感谢作者，这个是我在极客最喜欢的专栏，收益良多；再有一个小建议：在每一个主题模块的小结中，把文章中提到的书籍做一个书单方便读者；期待后面内容越来越精彩！

编辑回复: 感谢你的建议 😊

◀ ▶



王小勃

2019-02-28



打卡

展开 ∨



zhengfc

2019-02-25



TDD、XP、Refactor三本书自从买的时候过了一遍，已经封印了五六年了，看来是时候重新跟着来了，老师不愧是thoughtworks的专家，一脉相承

作者回复: 能看出好东西的好，也是一种能力。



One day

2019-02-21



今天来补课了，看到任务分解和习惯这两方面真的很受启发，很受用，我最近刚好给自己定制了每天20个以上的俯卧撑计划（以前能更多），和一份不太紧凑的学习计划。在老师这里感受很深，我会坚持下去

展开 ∨

作者回复: 欢迎回来，继续坚持！



树根◆...

2019-02-14



昨天刚改了编程习惯，先在notion写出思路、需要用到的知识点，api等，写出各个小任务，然后对应写出关键代码段。最后真正敲代码就花了10来分钟。

重新开始看极客就看到这篇，实践过来读，很认同。

...

展开 ∨

作者回复: 学以致用！

妮可

2019-02-03



公司经常存在有两个需求，其中一个需求因公司需要，如发布会，延迟更新的情况，这种情况似乎无法用任务分解完成。请问老师所在的团队如何解决单分支上线不同步的情况呢？

展开 ▾

作者回复: 两个功能的问题应该用 Feature Toggle 解决。

Geek\_28d7f...

2019-01-31



醍醐灌顶，期待老师对任务分解的实践。这样无论什么工作，都知道如何分解了，越来越感觉这些软技能比技术本身还要重要！感恩老师的分享！

作者回复: 其实，任务分解是硬功夫，需要像写代码一样，不断练习。

ysyou

2019-01-31



觉得任务分解和规划本身就是一个大任务了，想听老师讲下是如何分解任务分解这个任务的，应该给规划分配多少时间，用什么样的标准来定义规划做好了，规划中一定要把每个步骤都想得清楚才能开始吗？

展开 ▾

作者回复: 你把任务分解理解得太大了，在开发中的分解就是写代码前想想的过程，参见下一篇。