

## 划重点讲关于“任务分解”，你要重点掌握哪些事



你好，我是郑晔，恭喜你，又完成了一个模块的学习。

在这个模块中，我主要讲解的是“任务分解”这个知易行难的工作原则。普通人与高手之间的差异，很大程度上取决于任务分解的粒度大小。但真正理解并应用好“任务分解”的原则并不容易，希望你能勤于练习，将知识内化成为你的能力。

### 重点复习

在这个模块中，我们学习到了一些最佳实践：

- **测试金字塔**
  - 行业中测试组合的最佳实践。
  - 多写单元测试是关键。
- **测试驱动开发**
  - 测试驱动开发的节奏是：红—绿—重构，重构是测试驱动开发区别于测试先行的关键。
  - 有人把测试驱动开发理解成测试驱动设计，它给行业带来的思维改变是，编写可测的代码。
- **艾森豪威尔矩阵 (Eisenhower Matrix)**
  - 将事情按照重要和紧急进行划分。
  - 重要且紧急的事情要立即做。重要但不紧急的事情应该是我们重点投入精力的地方。紧急但不重要的事情，可以委托别人做。不重要不紧急的事情，尽量少做。
- **最小可行产品**
  - “刚刚好”满足客户需求的产品。
  - 在实践中，要用最小的代价找到一条可行的路径。

另外，我还提到了一些可以直接在工作中应用的做法和评判标准：

- 尽量不写 static 方法；

- 主分支开发模型是一种更好的开发分支模型；
- 好的用户故事应该符合 INVEST 原则；
- 估算是一个加深对需求理解的过程，好的估算是以任务分解为基础的；
- 好的测试应该符合 A-TRIP。

我也带你学习了一些重要的思想，帮你更好地改善自己的开发工作：

- 分而治之，是人类解决问题的基本手段；
- 软件变更成本，它会随着时间和开发阶段逐步增加；
- 测试框架把自动化测试作为一种最佳实践引入到开发过程中，使得测试动作可以通过标准化的手段固定下来；
- 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；
- 大师级程序员的工作秘笈是任务分解，分解到可以进行的微操作；
- 按照完整实现一个需求的顺序安排开发任务。

## 实战指南

在“任务分解”的板块，我也将每篇内容浓缩为一句实战指南，现在一起回顾一下。

- 动手做一个工作之前，请先对它进行任务分解。  
[—— 《11 | 向埃隆·马斯克学习任务分解》](#)
- 多写单元测试。  
[—— 《12 | 测试也是程序员的事吗？》](#)
- 我们应该编写可测的代码。  
[—— 《13 | 先写测试，就是测试驱动开发吗？》](#)
- 将任务拆小，越小越好。  
[—— 《14 | 大师级程序员的工作秘笈》](#)
- 按照完整实现一个需求的顺序去安排分解出来的任务。  
[—— 《15 | 一起练习：手把手带你拆任务》](#)
- 要想写好测试，就要写简单的测试。  
[—— 《16 | 为什么你的测试不够好？》](#)
- 想要管理好需求，先把需求拆小。  
[—— 《17 | 程序员也可以“砍”需求吗？》](#)
- 尽量做最重要的事。  
[—— 《18 | 需求管理：太多人给你安排任务，怎么办？》](#)
- 做好产品开发，最可行的方式是采用 MVP。  
[—— 《19 | 如何用最小的代价做产品？》](#)

## 额外收获

在这个部分的最后，针对大家在学习过程中的热门问题，我也进行了回答，希望你懂得：

- 对不了解技术的任务，先要去了解技术，然后再做任务分解；
- 通过一次技术 Spike，学习新技术；

- 丢弃掉在 Spike 过程中开发的原型代码；
- 分清目标与现状，用目标作为方向，指导现状的改变；
- 多个功能并行开发可以考虑使用 Feature Toggle；
- 在遗留系统上做改造可以考虑使用 Branch by Abstraction 。

## ——《答疑解惑 I 如何分解一个你不了解的技术任务？》

### 留言精选

在“任务分解”的模块中，有很多同学非常用心，将自己的学习心得和工作中的经验进行了分享，在此我挑选了一些同学的留言，与你一起学习。

在讲大师级程序员的工作秘笈时，**西西弗与卡夫卡** 同学提到：

最近在做战略拆解，都是一样的道理。战略飘在空中遥不可及，要落地就必须拆解。比如说达成目标有哪几个方面可以努力，各方面都需要做哪些事，这是路径。这些路径里哪些优先级最高，需要配置哪些组织资源。心里有数之后就是制订计划时间表。

另外，**西西弗与卡夫卡** 同学还为Spike给出了一个很生动的解释：

“技术Spike”可以翻译成“技术撩”，就是撩妹的那个撩。试探下，有戏就继续，撩不动就算或者放一段时间再说。

针对分解的粒度问题，**大彬** 同学也分享了自己的心得：

我会的任务分解，不仅可执行，粒度还很细。比如说，我要修复一个rpc接口的bug。我会列出每个代码的修改点，要修改的测试，要增加的测试，合并到哪个分支，修改rpc文档，文档中有哪些点要修改。

每一步都非常容易执行，看起来没多少必要，但我当前的工作环境特别有用：（1）事前思考，不会造成遗漏；（2）任务实施过程中经常被打断，比如，测试有疑问和你讨论、主管找你谈事、紧急会议来了，这种“硬中断”完全打破了节奏，而任务列表，让我清楚知道当前做了多少，该从哪一步继续。

对于单元测试，**树根** 同学提到：

我的想法可以在复杂度高，重要核心的模块先开始写单元测试。特别是公用、底层的，因为这些靠功能测试很难覆盖。单元测试难以推行主要是没有碰到质量的痛点，通常都依靠测试工程师来保证质量。我们之前就遇到过质量崩塌，倒逼着我们去，以保证质量。

**树根** 同学还分享了自己的任务分解实践心得：

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。

我特别佩服国外的工程师写的代码，代码块很小，非常清晰易读。特别记得之前参加infoq会议，听socketio作者的分享，看他现场撸码，思路、代码结构都非常顺畅和清晰。

关于TDD的具体应用，**萧** 同学提到了遇到的问题：

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，需求变更下改动，也不伤筋动骨了，但还是有许多感觉做的不够好的地方。看了这篇文章，补充了对TDD的认知，感受到如果和任务分解结合起来，TDD会有更好的效果，期待后面的文章！

关于“任务分解”的执行问题，[如明如月](#) 同学分享了感悟：

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，导致返工和一些“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。

在讲到为什么很多人的测试不够好这个问题时，[毅](#) 同学提到：

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，我在工作中发现，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)

针对需求的管理问题，[WL](#) 同学提到的点也非常关键：

程序员也应该更积极主动一些，最好能推动事情发展，当这件事情由你推动时，主动权就在你的手里了。

感谢同学们的精彩留言。在下一个模块中，我将为大家分享“沟通反馈”这个原则的具体应用。



## 沟通反馈主题预告

为什么世界和你的理解不一样？

——信息论的视角看沟通反馈

你的代码为谁而写？

——用业务的语言写代码

你总是在开会吗？

——团队的沟通：轻量级沟通

可视化：一种更为直观的沟通方式

——谈可视化沟通的关键点

为什么你们公司总是做不好持续集成？

——持续集成的关键：持续反馈

开发中的问题一再出现，应该怎么办？

——回顾会议：复盘与改善

作为程序员，你也应该了解用户

——用户思维，聆听来自用户的声音

为什么被指责的总是你？

——把事情做在前面：变被动为主动

写文档，做分享，也是一种学习方式

——让自己理顺思路

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。

精选留言