自洽的程序员

关于程序员职场和生活的思考

辣条加辣

Table of contents

1. 自洽的程序员 4
   1.1 简介 4
   1.2 关于作者
   1.3 公众号: 辣条加辣
2. 前言 5
   2.1 为什么要写这本书 5
   2.2 本书适合谁阅读 5
   2.3 如何阅读本书 5
   2.4 评论
   2.5 关于作者
   2.6 公众号: 辣条加辣
3. 第一章：建立你的工作哲学 6
   3.1 第一章 建立你的工作哲学 6
   3.2 什么是工作哲学 6
   3.3 1.1 第一性原理思考工作 7
   3.4 1.2 工作挣扎是常态 13
   3.5 1.3 工作承载不了太多意义，但也不要陷入工作虚无主义 17
4. 第二章 工作方法论 22
   4.1 第二章 工作方法论 22
   4.2 2.1 心态开放，你的职场第一课 23
   4.3 2.2 下场去做，你的职场第二课 28
   4.4 2.3 寻求帮助是项高级技能 32
   4.5 2.4 害怕直面冲突，怎样才能支棱起来 37
   4.6 2.5 如何面对职场PUA 41
   4.7 2.6 内耗，是从拒绝沟通开始的 45
   4.8 2.7 不喜欢编程，还要坚守程序员这个岗位吗 48
   4.9 2.8 工作倦怠了吗，试试三叶草模型 52
   4.10 2.9 天天用AI，以后自己不会写代码怎么办 55
   4.11 2.10 职场中如何做选择 59
   4.12 2.11 职场中最实用的三个思维模型 63
   4.13 不会吹牛逼，述职/晋升总是吃亏怎么办？ 69
   4.14 聊聊职场程序员那些真正重要但却经常被忽略的事 71
5. 第三章 工作中的人际关系 73
   5.1 第三章 职场中的人际关系 73
   5.2 3.1 领导一对一（1on1）聊什么 74

5.3 3.2 领导让我提意见，我该怎么提 78
5.4 3.3 同事太优秀怎么办？ 81
5.5 3.4 同事是傻逼，我有厌蠢症，实在受不了了,想杀人 84
5.6 3.5 卷王太凶残，我要不要'反卷'？ 87
5.7 3.6 我没有晋升不重要，但是他的晋升让我不爽 90
5.8 社恐人混职场是真难啊 92
6. 第四章 工作与家庭 95
6.1 第四章 工作与家庭 95
6.2 4.1 好的伴侣可以帮你消化工作上的负面情绪 97
6.3 4.2 维系亲密关系最重要的一点：接纳胜过改变 102
6.4 情绪稳定是个伪命题 107
6.5 伴侣老是抱怨，我该怎么办 109
6.6 婆媳关系真是男人的噩梦啊 111
7. 第五章 只工作不上班 114
7.1 第五章 只工作不上班 114
7.2 5.1 我所接触到的程序员副业 115
7.3 5.2 Gap 无罪：寻找属于你自己的 Pathless Path 119
7.4 其实，没工作，也能活的很好 122
8. 结语：成为一个自洽的程序员 124
8.1 关于作者 125
8.2 关注公众号：辣条加辣 125
8.3 评论
8.4 关于作者
8.5 公众号: 辣条加辣

1. 自洽的程序员

1.1 简介

1.1.1 这不是一本技术书

首先，这**不是**一本程序员的技术书籍，整本书不会提及任何一个技术词汇，这也不是一本教你如何规划职业生涯，如何在职场走个更远的书，虽然我相信大部分内容确实有助于在职场的发展。

但这本书的真正用意是想解决工作过程中碰到的**焦虑、倦怠、迷茫、抑郁**等情绪，聚焦于解决具体问题，通过改变认知将我们从负面情绪的泥淖中走出来，做到更坦然，真诚的面对自己的内心，成为一个自洽的程序员。

总而言之，这不是一本成功学的书，它不会教你如何赢，笔者本身也不是一个世俗意义上成功的人，而是一本帮你**梳理情绪，转变心境**的书。

1.1.2 关于情绪

为什么要从情绪这个角度出发呢？笔者相信，这个世界就是你自己内心的投射，而情绪，是一个人自洽程度的晴雨表。

我想再花点篇幅聊一下情绪：

情绪并非是理智的对立面，他在人类进化几万年的时间里起到了非常重要的作用，但是在强调价值理性的现代工业社会里，情绪被大家当成了害群之马，仿佛有情绪就是不成熟，这种认知是很偏狭的。

即便是负面情绪，也并不全是负面的，他也有其正面积极的意义，负面情绪是**清醒剂**，他让你认清现实，提醒你是时候接受现实，或者做出改变。

不过现实中，当我们有负面情绪时，往往陷入其中忽略了客观现实，关于这点无需自责，这是人性的弱点，每个人都会有这样的脆弱时刻，这时候，需要有一些外界的善意提醒或者建议，帮助我们理清负面情绪背后的逻辑，我希望，这本书在这个方向上能能发挥一点点这样的作用。

1.1.3 笔者的简单声明

因为个人的局限性，这本书难免有其偏颇狭隘的地方，同时，笔者本身是一个**乐观的悲观主义者**，认为这个世界上有太多我们没法真正认清，没法改变的事情，个人力量在整个时代面前是非常渺小的。我们能做的就是尽量保持清醒，调整自己的心态，把幸福的定义从别人的期望里转到自己的内心感受上。

就像那本著名的祷告词一样：上帝，请赐予我平静，去接受我无法改变的，给予我勇气，去改变我能改变的，赐我智慧，分辨两者的区别。

因为笔者是程序员，所以这本基于作者经验的书起名叫做《自洽的程序员》，但我相信书里面提到的很多问题都是有共性的，绝大部分打工人都可以从书里面找到共鸣。

2. 前言

2.1 为什么要写这本书

作为一名工作了**八年**的程序员，我经历了： - 国企 - 大厂 - 外企

在这些不同的环境中，每次转换都带来新的历练： - 新的项目 - 新的人际关系 - 新的企业文化

当我因为工作困境陷入情绪泥泞时，我很难找到一本能真正契合我的情绪并提出建设性意见的书籍。

2.1.1 现有书籍的局限

职场类书籍的问题：

市面上关于工作的书很大一部分都是职场成功人士在做到高层后输出的职场进阶指南，字里行间充满了高屋建瓴的爹味说教，即便有些道理，也很难从中共情。

经典书籍的局限：

比如《高效能人士的七个习惯》这类书的问题在于他年纪太大了，并没有那么适合这个时代。这类书的屁股还是坐在资产阶级统治者的位置上，强调如何高效的生产，鲜少顾及人的复杂性与多样性。

2.1.2 寻找解决之道

好在笔者从一些**心理学、社科、哲学**书籍里找到了一些慰藉以及解决之道，在踩了一次次坑，一次次自我认知的迭代之后，我也逐渐找到了适合自己的自洽的工作哲学。

这本书分享的是一个**普通人**在工作中遇到的一些普通的烦心事，以及如何解决了这些烦心事，没有宏大叙事，没有大段的说教。就是在解决这些烦心事的过程中，有一些思考和感悟，写下来，分享出去，就这么简单。

本书的写成借助了AI的力量，感谢AI 的协助。

2.2 本书适合谁阅读

虽然笔者是程序员，但书中提到的很多问题都具有**普遍性**，绝大部分打工人都可以从书里面找到共鸣。

2.3 如何阅读本书

2.3.1 带着问题读

如果你是带着具体问题读这本书，那最好不过了： - 通过副标题找到感兴趣的内容 - 这是最有效率的获取信息的方法 - 如果问题得到解决，你可以： - 合上本书，去喝杯咖啡，享受生活 - 或继续探索其他章节

2.3.2 随性阅读

如果你是出于好奇翻开这本书： - **不建议** 按顺序一页页读下去 - **建议**： 1. 打开目录 2. 找到最感兴趣的副标题 3. 从感兴趣的章节开始读 4. 继续寻找下一个感兴趣的主题 5. 随时可以合上书本，去做其他事情

2.4 🏷 评论

3. 第一章：建立你的工作哲学

3.1 第一章 建立你的工作哲学

3.2 什么是工作哲学

工作哲学，本质上是一个人对工作的基本认知和价值判断体系。它不是空洞的说教，而是建立在个人经验和思考基础上的，关于"为什么工作"以及"如何工作"的系统性思考。

3.2.1 工作哲学可以由上到下分为三个维度

认知维度

工作哲学首先体现在对工作本质的理解上：

- 工作在我们人生中的定位，我们如何通过工作跟社会链接
- 对工作与生活关系的认识

价值维度

其次是对工作价值的判断：

- 什么是我们值得追求的工作目标，薪资？能力？影响力？
- 如何平衡个人价值和公司价值，尤其是双方冲突的时候如何平衡
- 在工作中应该坚持什么，可以妥协什么

方法维度

最后是将认知和价值转化为行动指南：

- 如何处理工作中的具体问题
- 如何做出职业选择和决策
- 如何平衡工作中的各种关系

3.2.2 为什么需要工作哲学

在当今快速变化的职场环境中，想清楚自己的工作哲学，或者起码不断的思考这些问题，对于个人来说意义重大，因为我们很容易陷入到工作的细节以及繁琐中，而忘记了自己出发的目的，忘记了工作的本质和意义。

工作哲学并不是一成不变的，一个刚入职的员工和一个工作了10年的员工，他们的工作哲学肯定是不同的，随着我们不断的成长，我们的工作哲学也应该不断开放和进化，它会随着我们的经历和成长不断调整和完善，但核心价值观则应保持相对稳定。这种平衡能够帮助我们在职业生涯中既保持定力，又不失灵活性。

3.2.3 评论

3.3 1.1 第一性原理思考工作

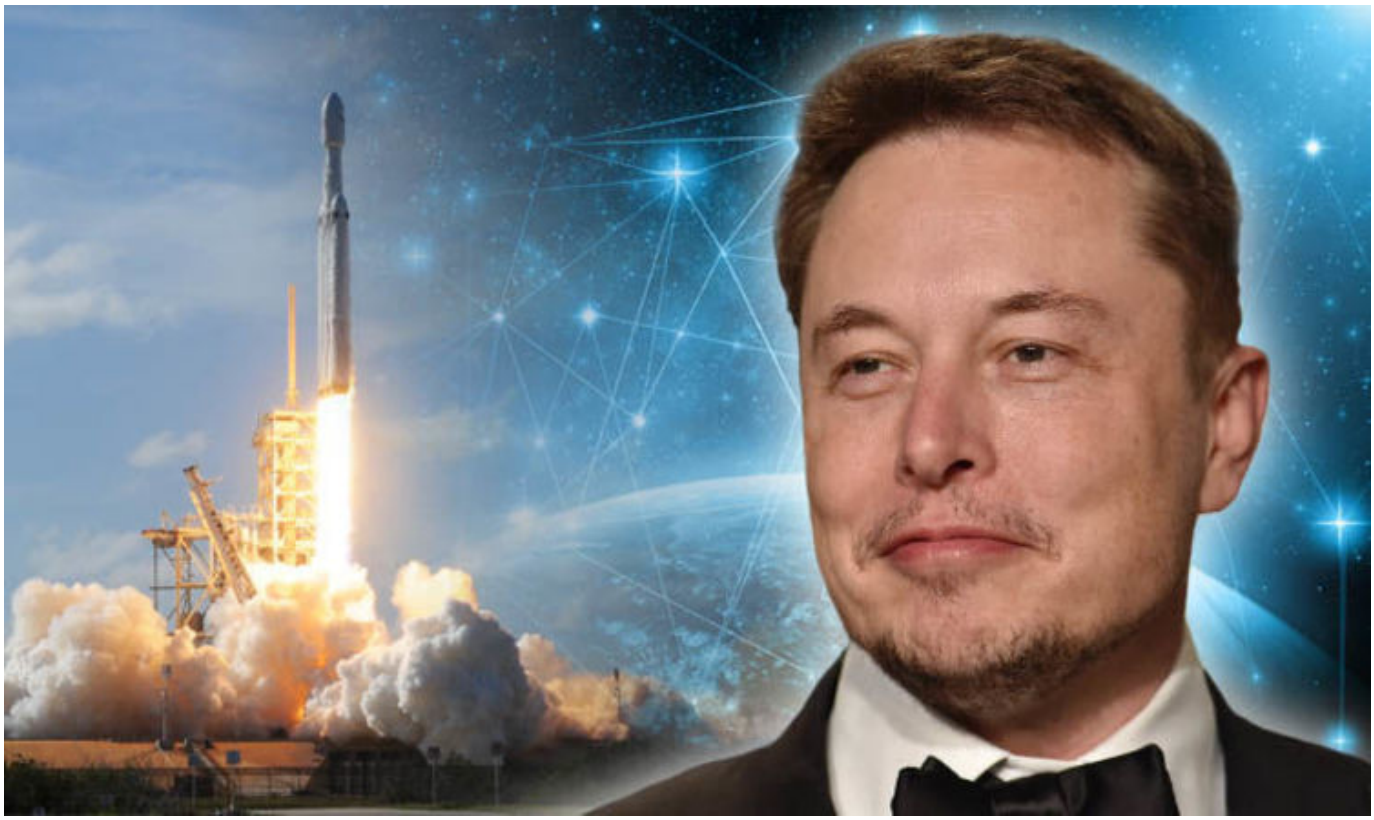
3.3.1 什么是第一性原理思考

第一性原理这个词儿，最早是亚里士多德提出来的。不过要不是马斯克天天挂在嘴边，这词儿可能现在还躺在哲学书的角落里吃灰呢。

说白了，第一性原理就是：不人云亦云，不轻信二手结论，而是从最基本的事实出发，重新思考问题。

来个栗子🌰

马斯克想造火箭的故事可能你们都听腻了，但这真的是个绝佳的例子：



当所有人都在说"火箭太贵了，造不起"的时候，马斯克在想啥？ - "等等，火箭到底是啥玩意儿？" - "造个火箭要多少铝合金、多少燃料？" - "这些原材料一共才多少钱？" - "为啥组装起来就贵了这么多？"

这就像我们写代码，与其复制Stack Overflow上的答案，不如想想这段代码到底要解决什么问题，从零开始写会是什么样。

3.3.2 为什么要用第一性原理思考工作

在职场里，我们经常被各种"你应该..."给包围了：

- "你应该去大厂"（BAT是程序员的终极梦想？）
- "你应该转管理"（技术大牛就该带团队？）
- "你应该卷起来"（不卷就会被优化？）
- "你应该35岁前达到P8"（为啥不是38岁？）
- "你应该像隔壁老王一样努力"（老王也想像你一样清闲...）

这些"应该"是从哪来的？

1. 社会压力

- 爸妈的期望："你看隔壁家小明..."
- 同学的压力："他们都在大厂..."
- 社会舆论："35岁危机..."

2. 行业惯性

- "前端必须会React"
- "后端必须会分布式"
- "全栈工程师才有前途"

3. 个人认知局限

- "大家都这么做，我也这样吧"
- "按老方法来准没错"
- "不确定的事情最可怕"

但是，用第一性原理思考的话，最基本的问题其实是：**"我为什么要上班，为什么要写代码？"**

3.3.3 工作认知的演进

初入职场：简单粗暴

刚开始工作时，想法特别纯粹：

- 赚钱，养活自己
- 学技术，长经验
- 独立，不啃老
- 证明自己，我行的！

真实案例

小辣条（没错，就是我）刚毕业时：

- "月薪过万就满足了"
- "有免费零食的公司就是好公司"
- "能学到技术就行"
- "领导夸我代码写得好，开心！"

那会儿想法简单，就想着能糊口就行，这没啥不好，都是必经之路。

工作三五年后：开始怀疑人生

工作几年后，你可能会发现，代码写得越多，问题越多：

- "我到底喜欢写代码吗？还是只是因为工资还不错？"
- "为啥我天天加班改Bug，隔壁老王天天摸鱼还升职了？"
- "这工作到底是我想要的，还是别人眼中的'好工作'？"
- "35岁危机是真的假的？要不要转管理？"
- "要不要跳槽？要不要创业？要不要躺平？"

典型困惑

- "工资是涨了，但感觉越来越菜了"
- "技术越学越深，但好像离产品越来越远"
- "工作稳定，但无聊得想打瞌睡"
- "收入可观，但头发越来越少"

这时候我们开始关注一些更深层次的问题：

- "我还能卷几年？"
- "要不要转行？"
- "要不要考个公务员？"
- "要不要回老家开个串串香？"

更成熟的阶段：开始看透

经过多年摸爬滚打，很多人会达到一个更通透的状态：

- 不再焦虑要不要转管理（反正都是坑）
- 不再纠结要不要进大厂（大厂也裁员）
- 找到了自己的节奏（摸鱼和卷，都是人生的一部分）
- 建立了自己的判断标准（老板开心不是最重要的，自己开心才是）

真实案例

辣条（还是我）十年工作感悟：

- 从BAT离职后选择了小公司（钱少事少，生活质量高）
- 拒绝了几个管理岗位（我还是喜欢写代码）
- 有时间陪家人了（再也不用和老婆解释为什么要加班）
- 开始做副业（加密货币搞起来）
- 心态更佛系了（项目延期？延就延吧，天还没塌）

3.3.4 用第一性原理重新思考工作

让我们把所有的条条框框都扔掉，重新想想：工作到底是个啥玩意儿？

1. 工作是价值交换

就像API调用：

- **Request**：
- 时间（每天8小时，加班另算）
- 技能（CRUD boy的自我修养）
- 创意（产品经理的需求该怎么实现）
- 体力（连续调试8小时的专注力）
- **Response**：
- 工资（房贷车贷的解药）
- 经验（从Bug中学习）
- 人脉（同事，未来的创业伙伴？）
- 成就感（这个Bug终于改完了！）

2. 工作是成长的游戏

- 技能树不断升级
- 认知水平不断提升
- 思维方式不断进化
- 社交能力不断提高

就像玩RPG游戏，工作就是主线任务，但别忘了还有支线任务（副业）和休闲任务（生活）。

3. 工作是人生的一部分

- 不是全部（还有老婆孩子热炕头）
- 需要平衡（头发和工资不可兼得）
- 要有边界（下班就是下班，工作群设置免打扰）

3.3.5 建立自己的工作观

1. 摆脱"应该"的束缚

- 不是非要进大厂（小公司也能过得很滋润）
- 不是非要当领导（技术专家也很香）
- 找到自己的节奏（有人喜欢冲刺，有人喜欢马拉松）

2. 建立健康的工作边界

- 该摸鱼时摸鱼
- 该努力时努力
- 该休息时休息

3. 保持进化和迭代

- 定期反思和总结（就像代码要重构）
- 及时调整方向（需求变了就要改方案）
- 保持开放和学习（新框架要学，新语言要懂）

3.3.6 实践建议

1. **定期和自己对话**
2. 每月反思：这个月摸鱼摸得值得吗？
3. 记录心情，看见自己的情绪：今天改Bug改得想跳楼了吗？
4. 复盘得失：这个项目坑在哪里？
5. **建立评估框架**
6. 工作是否开心？
7. 技术是否进步？
8. 钱是否够花？
9. **及时做出调整**
10. 不爽就换（总有更适合的坑）
11. 相信直觉（心累就该走了）
12. 大胆尝试（最差也就是回去继续写CRUD）

3.3.7 最后的几句话

用第一性原理思考工作，不是为了否定现有的一切，而是帮助我们：

- 看清本质（工作就是交换）
- 建立标准（开心最重要）
- 做出选择（人生苦短，及时止损）

笔者还想强调的是，你对工作的认知，会随着年龄和阅历不断变化，这很正常。关键是要经常问问自己："我为什么要工作？" 只有时不时的思考下这个问题，才能在代码的细节以及工作的繁琐中偶尔抬起头来，看清现阶段的自己真正想要的是什么。

毕竟，人生苦短，代码要甜。

3.3.8 评论

3.4 1.2 工作挣扎是常态

3.4.1 你今天挣扎了吗？

"今天的需求改了三遍..." "这个bug改了一周还没解决..." "产品经理又改需求了..." "领导说要周末加班..."

如果你也经常发出这样的感叹，别担心，你不是一个人。每个程序员都在挣扎，只是有的人挣扎得比较优雅，有的人挣扎得比较狼狈而已。

程序员的日常挣扎

程序员日常

技术挣扎

- "这个框架文档写得跟天书一样..."
- "这代码是哪个祖宗写的？注释呢？"
- "为啥我本地运行得好好的，一上线就炸了？"
- "这bug到底是从哪来的？"

业务挣扎

- "产品经理，我们能好好说话吗？"
- "这需求真的有人用吗？"
- "又改需求？要不你来写代码？"
- "这个功能真的要这周上线？"

团队挣扎

- "代码评审怎么又被挑刺了？"
- "为啥我的代码总是被说不规范？"
- "老王的代码我看不懂啊..."
- "新来的同事水平有点差，带起来好累..."

3.4.2 为什么会挣扎？

1. 技术发展太快

- 昨天的最佳实践，今天就成了反面教材
- 刚学会Vue2，Vue3就出来了
- 刚搞懂Redux，Mobx又火了
- 刚入门Docker，k8s又来了

就像你刚买的iPhone 13，iPhone 14就发布了，这种感觉，懂的都懂...

2. 业务需求太飘

产品经理的日常语录：

- "这个功能很简单，下班前能改完吧？"
- "客户说要改一下，就改个小需求"
- "这个需求很急，能不能今天就上线？"

每次听到"简单"这个词，内心都会咯噔一下。因为经验告诉我们，产品经理说的"简单"，往往意味着通宵。

3. 人际关系太复杂

- 产品经理想要天马行空
- 测试想要零缺陷
- 运营想要快速上线
- 老板想要降本增效
- 我只想安静地写代码

这就像在玩一个多人在线游戏，每个人都想当C位，但最后背锅的往往是程序员...

3.4.3 挣扎的本质是什么？

其实仔细想想，工作中的挣扎无非是这么几种情况：

1. 能力与要求不匹配

- 项目要求：精通分布式架构
- 现实情况：熟练CRUD
- 最终结果：天天加班还写不完

2. 期望与现实不匹配

- 期望：心无旁骛写优雅的代码，改变世界
- 现实：天天改bug，跟各个方向对需求
- 结果：每天怀疑人生

3. 付出与回报不匹配

- 付出：每天工作12小时
- 回报：工资涨幅不及物价
- 收获：头发越来越少

3.4.4 如何优雅地挣扎？

1. 调整心态

- 把Bug当成送分题
- 把加班当成充电
- 把改需求当成锻炼
- 把背锅当成历练

（好吧，我知道这听起来很像"精神胜利法"，但是真的有用！）

2. 提升能力，挣扎的越厉害，成长越快

- 每天学习一个新技能
- 遇到问题先自己解决
- 做好复盘和总结
- 建立知识体系

3. 建立护城河

- 技术上：
- 至少一个领域要精通
- 至少一个方向要深入
- 至少一个特长要突出
- 软实力上：
- 学会和产品经理谈判
- 学会和测试讲道理
- 学会和领导说不
- 学会和同事互帮互助

3.4.5 挣扎中的成长

1. 技术成长

- 从"这bug怎么改？"到"为什么会有这个bug？"
- 从"这代码怎么写？"到"这代码该怎么设计？"
- 从"复制粘贴"到"看源码找答案"

2. 思维成长

- 从"完成任务"到"解决问题"
- 从"写代码"到"写方案"
- 从"改bug"到"防bug"

3. 职业成长

- 从"被动接需求"到"主动提方案"
- 从"只管写代码"到"参与决策"
- 从"单打独斗"到"团队协作"

3.4.6 最后的几句话

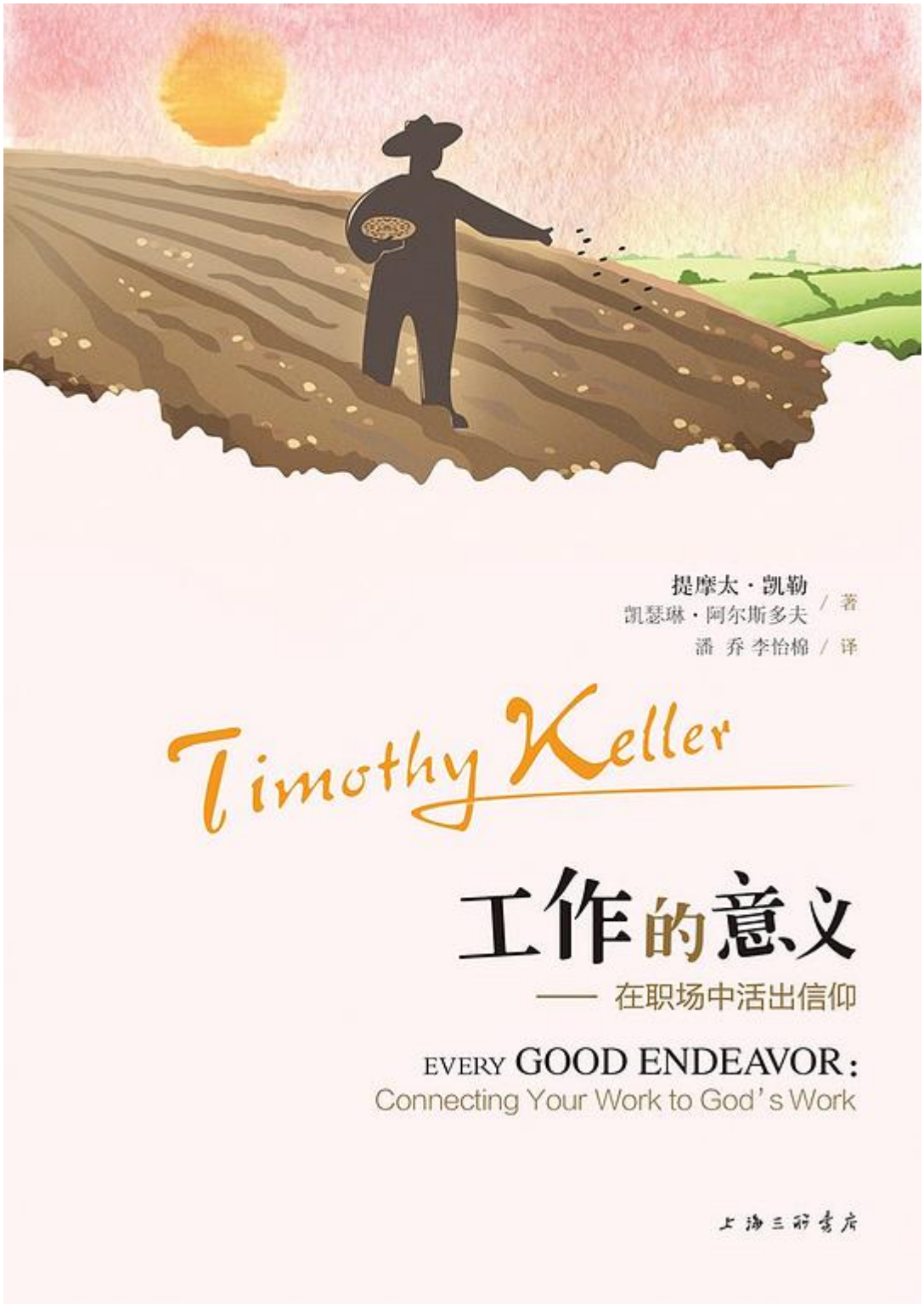
工作中的挣扎就像人生的必修课：不是你的错，但要你来解决。不是你能控制的，但要你来负责。不是你想要的，但要你来面对。

与其抱怨挣扎，不如把挣扎当成成长的机会。就像重构代码一样，挣扎的过程也是重构自己的过程。

最后的最后，挣扎是常态，但快乐也可以是！你无法控制工作的挣扎现实，但可以控制自己面对现实的心态。

3.4.7 评论

3.5 1.3 工作承载不了太多意义，但也不要陷入工作虚无主义



3.5.1 关于工作的两个极端

最近在程序员群里，经常看到两种极端的声音：

极端1：工作就是生活的全部

- "不加班怎么能进步？"
- "不卷怎么能成功？"
- "工作就是生活的意义啊！"
- "多干活才能有未来！"

这帮人把工作当成了人生的全部，仿佛不工作就活不下去了。每天加班到深夜，周末也在想工作，朋友圈全是技术文章...

极端2：工作就是浪费生命

- "工作不就是为了赚钱吗？"
- "反正都是给老板打工"
- "上班就是浪费生命"
- "能摸鱼就摸鱼"

这些人觉得工作毫无意义，上班就是在消耗生命，除了赚钱没有任何其他意义，恨不得立马辞职去流浪。

3.5.2 为什么会有这两种极端？

1. 社会压力

- 房贷车贷的压力
- 35岁危机的焦虑
- 同龄人的对比
- 父母的期望

2. 互联网的放大效应

- 成功学的洗脑
- 躺平学的诱惑
- 各种极端观点的传播
- 戾气的积累

3. 个人认知的局限，当然也是当下主流的工作伦理

- 把工作等同于生活
- 把工作努力等同于最高道德
- 把收入等同于价值

- 把职位等同于能力
- 把忙碌等同于进步

3.5.3 工作到底承载了什么？

1. 最基本的：生存需求

- 温饱（这是最基本的）
- 房子（安身之所）
- 车子（代步工具）
- 医疗保险（以防万一）

2. 进阶的：成长需求

- 技能提升
- 视野拓展
- 人脉积累
- 经验沉淀

3. 更高层的：自我实现

- 成就感
- 价值认同
- 社会贡献
- 个人影响力

3.5.4 工作的真相是什么？

1. 工作既不是全部，也不是虚无

- 它是生活的一部分，但不是全部
- 它有它的价值，但不是唯一的价值
- 它值得认真对待，但不要太较真
- 它需要投入，但不是无限投入

2. 工作是一种平衡

- 在理想和现实之间
- 在付出和收获之间
- 在个人和团队之间
- 在工作和生活之间

3. 工作是一个过程

- 不是终点，而是旅程
- 不是结果，而是经历
- 不是负担，而是选择
- 不是束缚，而是机会

3.5.5 如何找到平衡？

1. 给工作一个合适的位置

- 不要把它看得太重
- 也不要把它看得太轻
- 找到自己舒服的节奏
- 保持健康的距离

2. 建立多元的生活

- 工作之外要有爱好
- 职场之外要有朋友
- 技术之外要有兴趣
- 收入之外要有追求

3. 保持清醒的认知

- 知道自己要什么
- 明白自己在做什么
- 清楚自己能做什么
- 了解自己该做什么

3.5.6 一些具体建议

1. 工作时间

- 正常下班就走
- 周末尽量不加班
- 休假要好好休
- 摸鱼要有度

2. 工作态度

- 该认真时认真
- 该放松时放松
- 该拒绝时拒绝
- 该妥协时妥协

3. 工作方法

- 提高效率，而不是延长时间
- 追求质量，而不是堆砌数量
- 注重成长，而不是纯粹付出
- 讲求方法，而不是蛮干

3.5.7 最后的几句话

工作就像人生的一个维度：它很重要，但不是唯一。它需要投入，但要有度。它值得认真，但别太执着。

工作是为了更好的生活，而不是让生活成为工作的附属品。

我们的目标是：认真工作，快乐生活。既不做工作的奴隶，也不做生活的逃兵。

最后，送大家一句话： 工作和生活就像代码和注释，缺一不可，但要比例适当。

3.5.8 评论

4. 第二章 工作方法论

4.1 第二章 工作方法论

在第一章中，我们建立了对工作的基本认知：接纳工作中的挣扎，理性看待工作的意义。但认知的提升并不能自动转化为问题的解决，我们还需要一套切实可行的方法论来指导具体实践。

4.1.1 为什么需要方法论

工作中的困扰往往表现为两个层面：

- **情绪层面**：焦虑、压力、倦怠、困惑
- **问题层面**：技术难题、项目瓶颈、团队协作、人际关系

这两个层面是相互影响的：情绪影响问题解决的效率，问题的堆积又加剧情绪的波动。

因此，我们需要一套系统的方法论来识别和处理负面情绪，提升问题解决能力，建立良性循环。

4.1.2 本章内容

在这一章中，我们将探讨：

- 如何保持开放的心态，接纳新的可能
- 如何在实践中提升认知，解决问题
- 如何建立情绪管理机制
- 如何提高工作效能

这些方法论不是空洞的理论，而是笔者源于自己实践的经验总结，也就是一个个坑踩过来的。他们有其共性的地方，也有其局限性，请自行斟酌。同时，方法论不是万能钥匙，而是一套思维工具，千万别生搬硬套，关键在于灵活运用，找到最适合自己的方式。

4.1.3 评论

4.2 2.1 心态开放，你的职场第一课

4.2.1 为什么心态开放这么重要？

还记得第一次接触新框架时的感觉吗？面对满屏的新概念，你可能像我一样，恨不得立刻关掉编辑器，躲回自己熟悉的"舒适窝"。这就像一只从来没见过大海的青蛙，突然被扔进了太平洋 —— 慌得不行。

但你知道吗？正是这种"慌得不行"的感觉，往往预示着我们遇到了成长的机会。

固定型思维 vs 成长型思维

说到思维模式，不得不提心理学家德韦克提出的这两种类型：

固定型思维的小伙伴们是这样的： - "这个新框架太难了，我学不会的" - "我就是个后端程序员，前端的事情碰都不想碰" - "我写了十年Java，改不了这个习惯了" - "新技术？等别人踩完坑再说吧"

听起来是不是特别耳熟？这就像一只固执的鸵鸟，遇到挑战就把头埋在沙子里，心想："看不见就不存在"。

而成长型思维的小伙伴则是这样的： - "虽然看不懂，但是好像挺有意思的" - "多学一样技能，多一条路" - "试试看呗，大不了就是报错" - "踩坑也是一种经验啊"

这就像一只好奇的猫咪，看到什么新东西都想去拨弄两下，说不定就发现了新大陆。

为什么说心态开放是第一课？

想象一下，如果你是一个杯子：

- 固定型思维就像一个已经装满水的杯子，新的东西都装不进去
- 成长型思维则像一个空杯子，随时准备接纳新的知识

在职场中，技术更新得比你手机系统还快。今天你精通的技术，明天可能就变成"传统艺能"了。如果不保持开放的心态：

- 新技术学不进
- 新方法用不上
- 新机会抓不住
- 新发展看不到

就像那句老话："`SELECT * FROM world WHERE 心态 = '开放'`"——好吧，这是我编的，但你懂我意思。

4.2.2 如何培养开放的心态？

1. 拥抱"不懂"

记得我刚转行做程序员时，遇到不懂的东西特别害怕被人发现，问问题之前要先憋上半天，搜索一堆资料，生怕被人说"连这都不知道"。

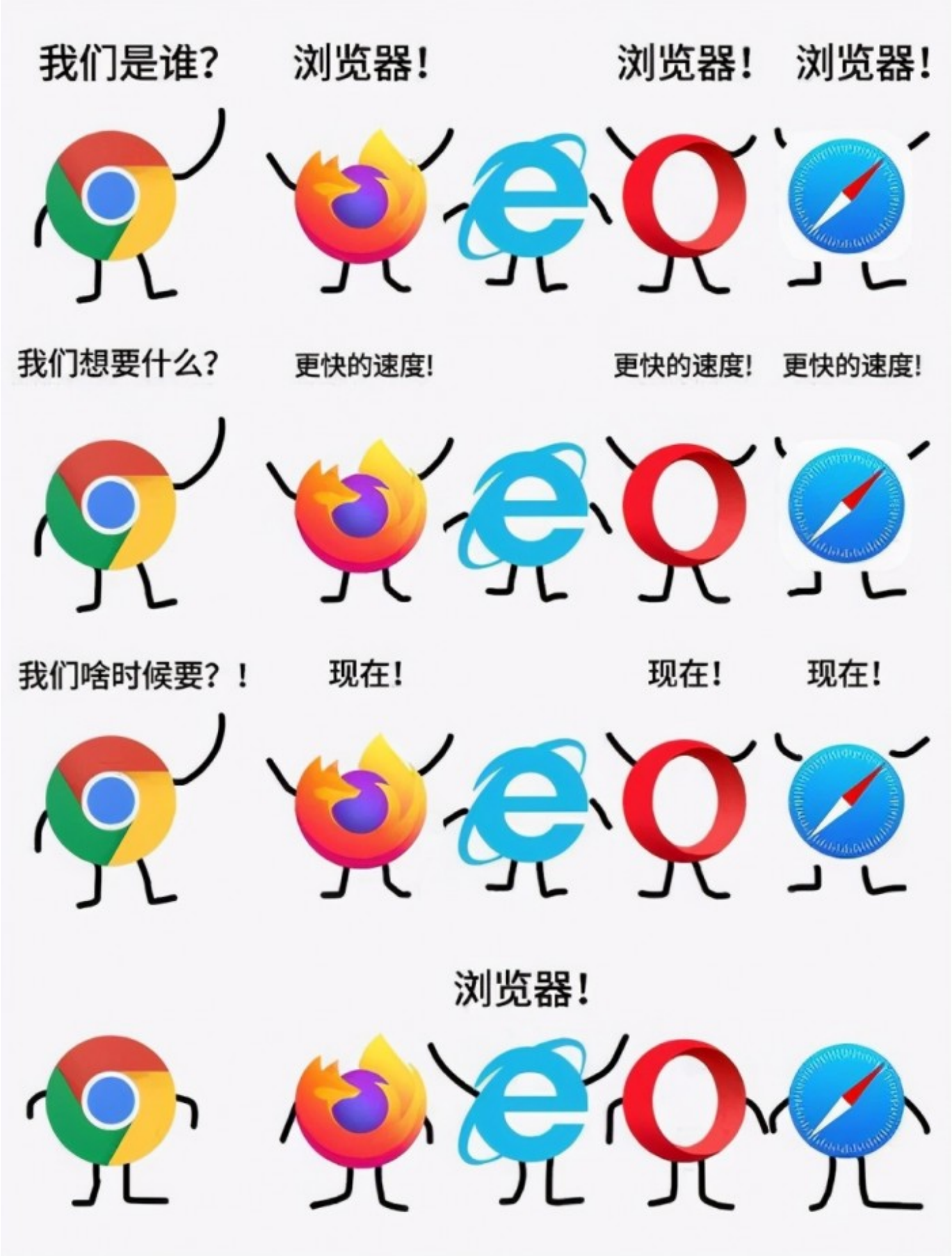
现在想想，这就像公共场合突然想放屁一样，憋得自己难受，还要装作若无其事一样不让人看出来。其实，承认"不懂"一点都不丢人，丢人的是：不懂装懂（这是最容易被戳穿的），不懂不学（这是最快掉队的），不懂不问（这是最耽误事的）。

2. 学会"归零"

每隔一段时间，我们都需要给自己"格式化"一下：

- 清空已有的经验偏见
- 重新审视工作方式
- 思考是否有更好的解决方案

就像我们的电脑一样，时不时需要重启一下，清清缓存，更新更新系统。不然，你可能还在用IE浏览器，坚持认为这是最好的选择。



3. 保持好奇心

好奇心就像是我们的"技能点"，需要不断投入：

- 看到新技术，先想想"这个有意思"
- 遇到新方法，试着问问"为什么"
- 碰到新工具，琢磨着"怎么用"

记得有个同事说过："我不是天才，但我有一颗八卦的心。"没错，职场进步有时候就靠这种"八卦"精神。

4. 建立"试错"机制

很多人不敢尝试新东西，是因为害怕犯错。但是，你想啊：

- 测试环境不就是用来犯错的吗？
- Code Review不就是用来发现问题的吗？
- 版本控制不就是用来后悔的吗？

把"试错"当作学习的一部分，就像打游戏一样，死几次才能熟悉地图，这很正常。

4.2.3 开放心态的实践建议

1. 从小事开始

不要一上来就要啃最难的骨头，可以：

- 今天试用一个新的IDE插件
- 明天学习一个新的快捷键
- 后天尝试一个新的调试方法

就像玩游戏要从新手村开始一样，一步步来。

2. 找到学习伙伴

- 和同事组成学习小组
- 参加技术社区
- 关注技术大牛的博客

有个伙伴一起学习，就不会觉得自己是一个人在战斗，同时，小伙伴还会给你带来新的视角和思路，甚至于，起到一定的监督作用。

3. 建立反馈循环

- 定期总结学习收获
- 及时记录遇到的问题
- 分享自己的心得体会

就像写代码要有单元测试一样，学习也需要及时的反馈。这个是非常重要的，因为只有反馈，才能让我们知道自己的学习效果，才能让我们知道自己的学习方向是否正确，也更能让我们有动力去继续学习。

4.2.4 结语

心态开放不是让你变成一个没有主见的"随风草"，而是要成为一个愿意尝试的探索者，乐于学习的追求者，勇于改变的行动者。

就像一个优秀的程序，既要有稳定的核心逻辑，也要有足够的扩展性。保持开放的心态，就是给自己的"程序"预留了足够的接口，随时准备迎接新的更新和升级。

毕竟，在这个快速发展的互联网时代，唯一不变的就是变化本身。与其抗拒，不如拥抱。让我们带着开放的心态，开始职场的新征程吧！

4.2.5 评论

4.3 2.2 下场去做，你的职场第二课

4.3.1 从一个真实的故事开始

记得我刚开始学React的时候，整整看了两周的教程和文档。脑子里天天想着组件生命周期、虚拟DOM、状态管理...但是迟迟不敢动手写第一行代码。

直到有一天，leader把一个小需求甩给我： "这个页面很简单，你用React重构一下。"

我："啊？可是我还没完全理解React的设计理念..." leader："先做，边做边学。"

就这样，我被迫上手了。结果发现很多觉得复杂的概念，用起来其实很简单，很多觉得简单的东西，做起来还真有点绕，书上看懂的东西，实操时经常卡壳，但是！真的做了之后，很多疑惑都自然解开了。

4.3.2 为什么我们总是迟迟不动手？

1. 完美主义的陷阱

- "再多看几个教程..."
- "等我把设计模式全部搞懂..."
- "让我再研究下最佳实践..."
- "这个方案还不够完美..."

2. 对失败的恐惧

- "万一写得不好被同事笑话..."
- "要是项目做砸了怎么办..."
- "这个技术栈我不熟悉，不敢碰..."
- "万一出了线上问题..."

3. 舒适区的诱惑

- "现有的方案虽然不完美，但是能用..."
- "学新技术太累了，还是用老技术吧..."
- "这个功能用jQuery也能实现..."
- "重构风险太大，能跑就行..."

4.3.3 不动手的后果是什么？

1. 技术停滞

- 知识始终停留在理论层面
- 实战经验得不到积累
- 问题解决能力无法提升
- 技术广度和深度都受限

2. 焦虑加剧

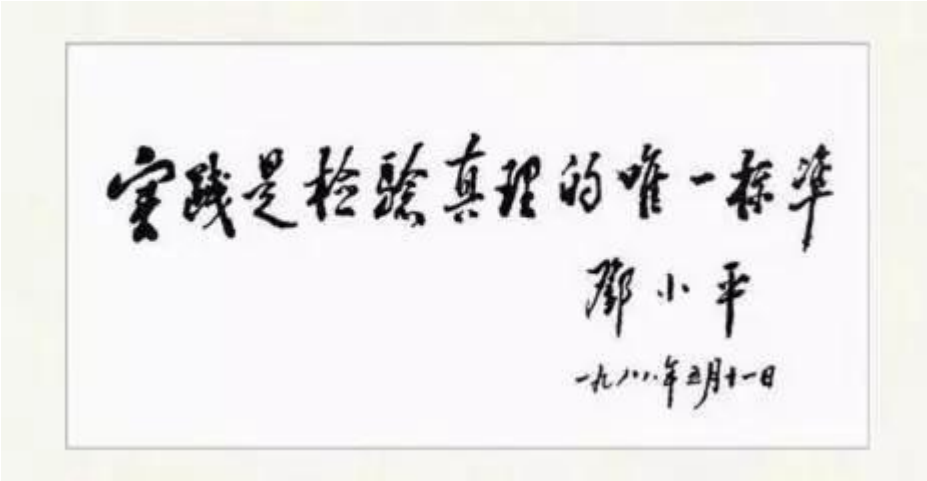
- 看着别人进步，自己原地踏步
- 害怕承担责任，导致机会越来越少

- 对新技术的恐惧越来越深
- 职业发展陷入瓶颈

3. 能力退化

- 执行力逐渐下降
- 解决问题的勇气越来越小
- 面对挑战时习惯性退缩
- 创新能力逐渐丧失

4.3.4 为什么要"下场去做"？



1. 实践是最好的学习

- 动手实操比看十篇教程有用
- 真实项目中的一个bug能学到很多东西
- 生产环境的问题才是最好的老师
- 实战中的教训记得最深

2. 反馈是最好的老师

- 代码写得对不对，跑一下就知道
- 方案可不可行，实践了才清楚
- 性能好不好，上线了就明白
- 用户喜不喜欢，发布了才知道

3. 失败是最好的经验

- 每个bug都是成长的机会
- 每次报错都是学习的契机
- 每个坑都值得记录和总结
- 每次失败都是宝贵的经验

4.3.5 如何开始行动？

1. 从小需求开始

- 先重构一个小组件
- 先优化一个小功能
- 先解决一个小bug
- 先改进一个小细节

2. 给自己设定期限

- 本周必须提交一个PR
- 这个月必须上线一个功能
- 这季度必须掌握一个新技术
- 今年必须完成一个完整项目

3. 建立反馈机制

- 及时code review
- 定期复盘总结
- 主动收集反馈
- 持续改进优化

4.3.6 一些实用建议

1. 关于学习新技术

- 先跑通官方demo
- 再写个小项目
- 然后在非核心项目中试水
- 最后才在重要项目中使用

2. 关于重构老代码

- 先写测试用例
- 再小范围重构
- 然后逐步扩大范围
- 最后整体优化

3. 关于解决问题

- 先本地复现
- 再分析原因
- 然后验证方案
- 最后总结经验

4.3.7 结语

程序员的成长没有捷径：理论再扎实，不实践也是空的；方案再完美，不执行也是零；想法再好，不落地也是假的。

就像写代码一样，再多的设计模式，不如一个能用的功能；再多的架构图，不如一个能跑的程序；再多的技术预演，不如一次真实的上线。

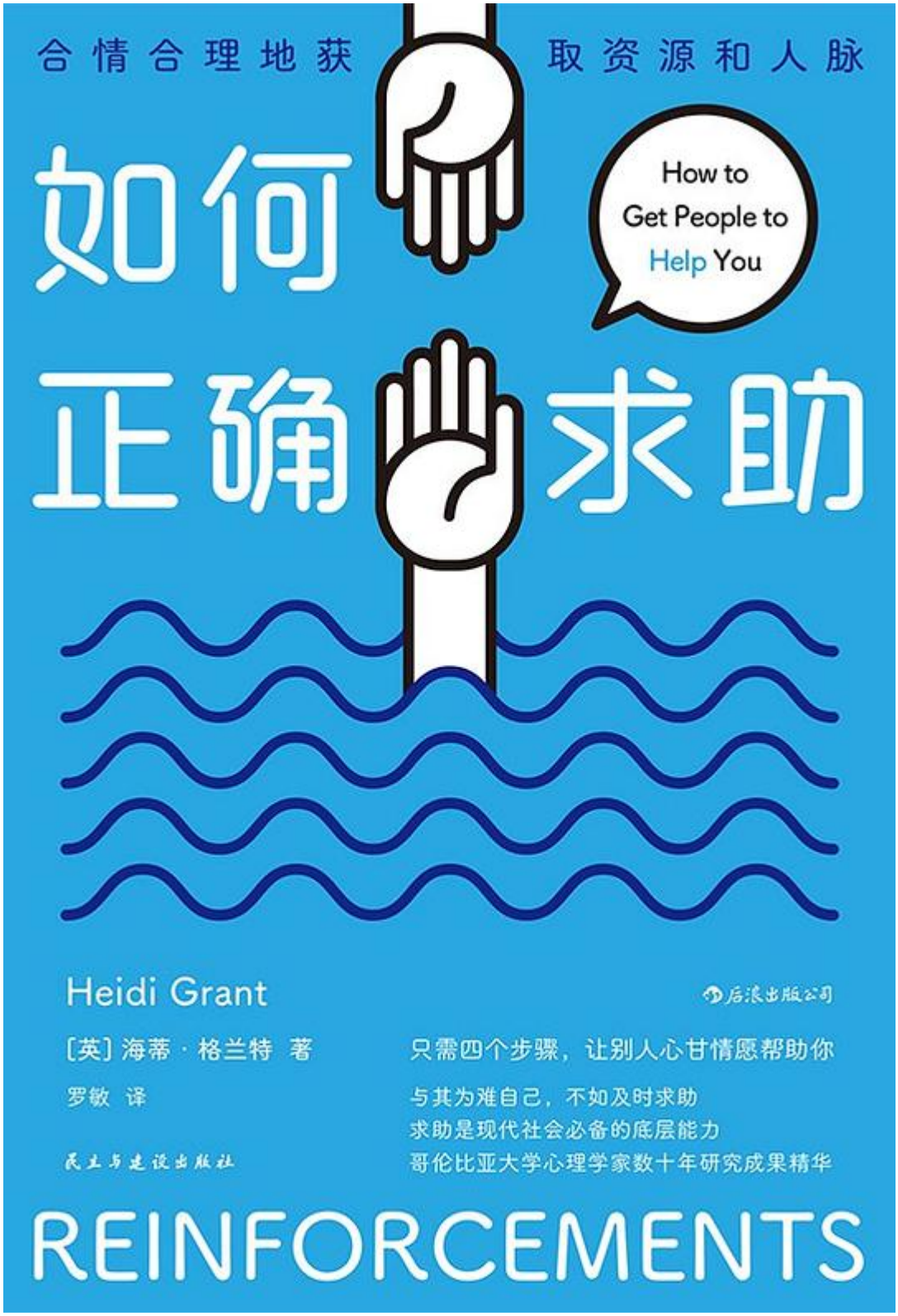
所以，与其在原地焦虑，不如：

- 放下完美主义的包袱
- 克服对失败的恐惧
- 走出舒适区的束缚
- 勇敢地下场去做

毕竟，代码是写出来的，不是想出来的。

4.3.8 评论

4.4 2.3 寻求帮助是项高级技能



4.4.1 从一个尴尬的故事说起

"那个...老王啊，这个报错你知道怎么解决吗？" "你自己有谷歌过吗？" "呃...还没..." "....."

相信每个程序员都经历过这种尴尬：问题没调研就去问同事，结果被嫌弃了。但反过来也有另一种情况：

"这bug我已经调了一周了，实在搞不定..." "你怎么不早说啊？这个问题我上周刚处理过！"

是不是很眼熟？其实这两种情况都说明了一个问题：我们不会寻求帮助，或者说，不会正确地寻求帮助。

4.4.2 为什么我们不敢寻求帮助？

1. 面子问题

- "问这么简单的问题会不会显得我很菜？"
- "都工作这么久了，这都不会，多丢人啊..."
- "万一被同事看不起怎么办..."
- "领导会不会觉得我能力不行..."

2. 错误认知

- "自己的问题应该自己解决"
- "优秀的程序员应该什么都会"
- "问别人就是无能的表现"
- "独立解决问题才是真本事"

3. 性格因素

- 内向不善于交流
- 不想麻烦别人
- 害怕被拒绝
- 社交恐惧症

4.4.3 不会寻求帮助的后果

1. 时间成本

- 一个有经验的同事5分钟能解决的问题
- 你可能要花一整天去摸索
- 项目进度被拖延
- 工作效率严重下降

2. 心理负担

- 越卡越焦虑
- 越焦虑越卡
- 自信心受挫
- 工作热情下降

3. 团队影响

- 本可以共享的经验没有共享
- 本可以避免的坑没有避免
- 团队协作效率低下

- 重复踩同样的坑

4.4.4 什么时候该寻求帮助？

1. 该自己解决的时候

- 基础语法问题
- 简单的配置问题
- 常见的报错信息
- 有明确错误提示的问题

2. 该求助的时候

- 尝试过多种方案都不行
- 搜索了很多资料没头绪
- 卡了较长时间没进展
- 涉及到历史遗留问题
- 需要业务相关的上下文

4.4.5 如何正确寻求帮助？

1. 求助前的准备

- 把问题描述清楚
- 什么情况下出现的
- 已经试过哪些方案
- 当前卡在哪一步
- 准备相关信息
- 错误日志
- 环境信息
- 复现步骤
- 相关代码片段

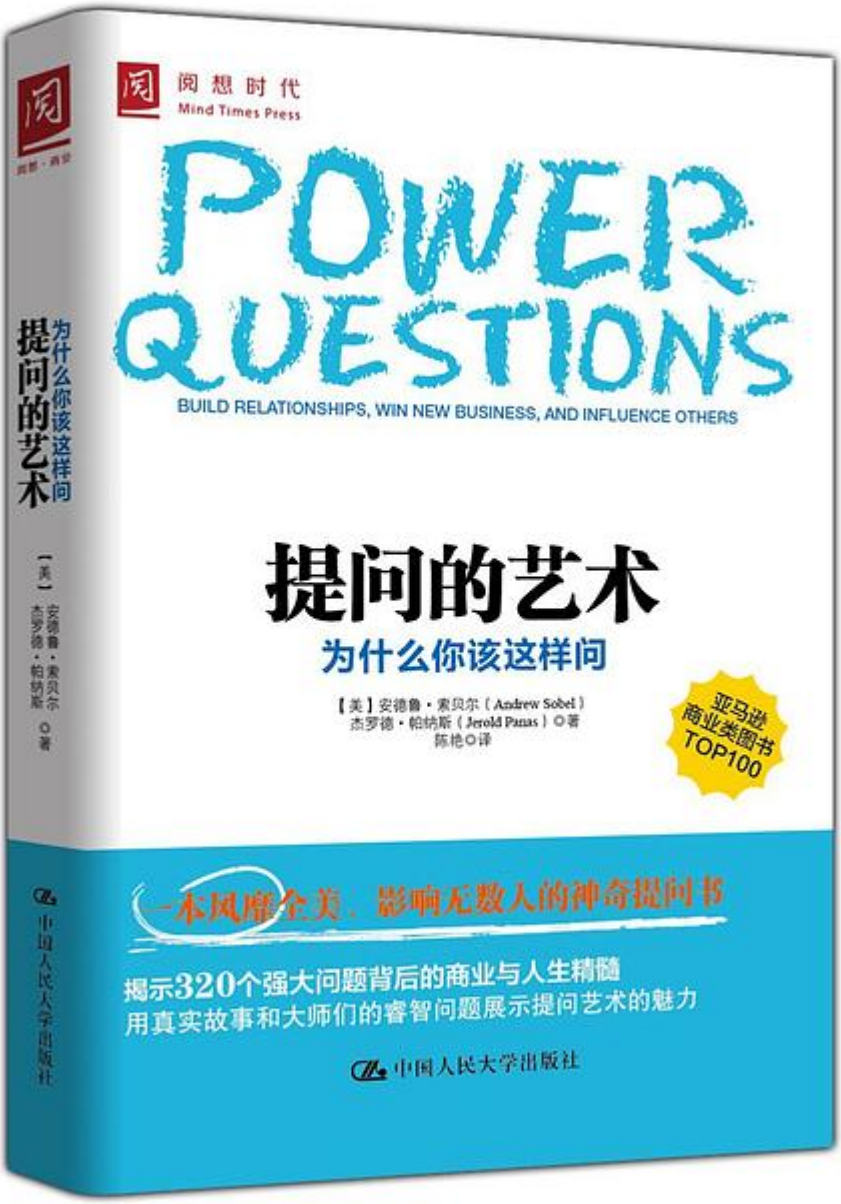
2. 选择合适的对象

- 了解这个领域的同事
- 做过类似项目的前辈
- 有相关经验的朋友
- 特定技术社区的专家

3. 选择合适的时机

- 不要在别人最忙的时候
- 不要在快下班的时候
- 不要在对方在开会时
- 最好提前预约时间

4.4.6 提问的艺术



1. 好的提问方式

- "我在实现XX功能时遇到了问题..."
- "我已经尝试了A、B、C方案，但都不行..."
- "我觉得可能是XX原因，你觉得呢？"
- "能否帮我看看这个思路对不对？"

2. 糟糕的提问方式

- "这个怎么做啊？"
- "为什么我的代码不行？"
- "帮我看看哪错了"
- "这个bug怎么解决？"

3. 提问时的注意事项

- 表达要清晰具体
- 态度要谦虚诚恳
- 要尊重对方时间
- 记得总结和感谢

4.4.7 建立良性循环

1. 及时记录和总结

- 把解决方案记录下来
- 总结问题的原因
- 整理相关的知识点
- 分享经验给其他同事

2. 主动回馈他人

- 帮助遇到类似问题的同事
- 分享自己的经验教训
- 参与技术讨论和分享
- 贡献团队的知识库

3. 建立学习体系

- 收集常见问题
- 整理解决方案
- 建立知识体系
- 形成经验沉淀

4.4.8 最后的话

在程序员这个职业里，寻求帮助不是能力不足的表现，更不是逃避责任的借口，而是一种提高效率的方法，解决问题的手段。

就像代码要讲究复用一样，经验也是可以复用的，知识也是可以共享的，成长也是可以互助的。

会寻求帮助的程序员，才是真正的高手。 不是因为他什么都会，而是因为他知道如何更快地解决问题。

4.4.9 评论

4.5 2.4 害怕直面冲突，怎样才能支棱起来

害怕冲突

"老王，你这代码写得也太烂了吧？" "啥？我这代码怎么了？" "你这变量命名，这代码结构，这是人能看懂的吗？" "我寻思挺好啊，能跑就行呗..." "能跑就行？！后面谁维护你知道吗？"

场面一度很尴尬。

代码评审现场翻车，相信不少同学都经历过。但更多时候，我们的反应是：

- 默默点个"收到"，然后继续摸鱼
- 心里暗骂对方太较真，但嘴上说"好的我改"
- 改完立马找产品理论："这需求本来就不合理！"
- 在工作群怼不过，转手就把对方拉黑

说实话，谁不是从怂包子开始的呢？我自己刚工作那会儿，那叫一个怂。产品经理改需求，默默接受；测试提bug，默默修改；leader说要加班，默默加班...直到有一天，我实在忍不住了。

4.5.1 为啥我们这么怂？

传统文化教我们做"老好人"

从小到大，我们都被教育要"和为贵"。打小学起：

- "要和同学好好相处啊"
- "忍一忍就没事了"
- "吃亏是福" 这些话听得耳朵都起茧子了。

到了职场，这种思维更甚：

- "大家都是同事嘛"
- "和气生财"
- "多一事不如少一事"

结果呢？憋出一堆职场"老好人"。

害怕得罪人

这个真不能怪我们怂，实在是：

- 得罪测试，你的bug就别想过了
- 得罪产品，下次需求改到你怀疑人生
- 得罪leader，你的绩效就悬了
- 得罪同事，代码评审能挑毛病挑到天亮

更要命的是，现在都讲究"团队协作"。得罪一个，可能得罪一群。谁还不想混口饭吃了？

技术底气不足

说实话，很多时候我们不敢怼，是因为：

- 代码写得确实不够好
- 技术深度确实不够
- 方案确实有漏洞
- 经验确实不足

就很尴尬，明明知道对方说的不全对，但又说不出所以然来。

4.5.2 怂着怂着，就出事了

技术债越堆越多

- 今天妥协用了个烂方案
- 明天将就写了个烂代码
- 后天将就改了个烂需求 最后？整个项目烂得像坨浆糊。

我之前就遇到过，一个"临时方案"用了两年，最后重构的时候，连碰都不敢碰，生怕整个系统崩溃。

背锅侠本侠

- 测试说有bug，默默改
- 产品说要改需求，默默改
- 运营说要加功能，默默改
- 领导说要优化，默默改

结果呢？但凡出点问题： "这块代码是谁改的？" "哦，是小王啊，每次都是他改的..."

职场混成透明人

久而久之：

- 技术讨论不敢发言
- 方案评审不敢反对
- 有想法也不敢说
- 有建议也憋着

最后在团队里混成了隐形人，存在感只剩下每天打卡。

4.5.3 冲突其实没那么可怕

职场冲突很正常

就像写代码一样：

- 不同的人有不同的代码风格
- 不同的团队有不同的技术栈
- 不同的项目有不同的要求

有分歧很正常，没分歧才不正常。

把冲突当成机会

- 代码评审被怼，是提高代码质量的机会
- 方案被质疑，是完善设计的机会
- 观点不一致，是思维碰撞的机会
- 需求有分歧，是深入理解业务的机会

4.5.4 如何硬气起来？

先把技术能力搞上去

说白了，底气不足就是因为实力不足。

要学会：

- 写代码先想想为什么要这么写
- 接需求先想想有什么坑
- 做方案先调研同行都怎么做
- 有空多看看源码，别光是CRUD

学会正确表达

以前我的表达方式： "这个...可能...会不会有问题..."

现在的表达方式： "这个方案我觉得有三个问题： 1. 性能可能会有瓶颈 2. 扩展性不太好 3. 维护成本会很高 我建议我们可以..."

把对抗变成合作

与其对抗：

- "你这需求改得也太频繁了！"
- "你这代码写得也太烂了！"
- "你这测试也太细了！"

不如：

- "要不我们先确定核心需求？"
- "我们一起看看代码怎么优化？"
- "测试案例是不是可以优先级排序？"

4.5.5 最后说两句

职场冲突就像代码里的bug，遇到很正常，解决很重要，回避不可取，处理要及时。

重要的不是避免冲突，而是学会处理冲突。

记住：

- 把话说出来总比憋在心里强
- 把问题摆在台面上总比暗地里较劲强
- 把分歧当面讨论总比背后吐槽强

最后，祝大家都能成为职场上的"硬汉"，不再害怕冲突。

4.5.6 🆖 评论

4.6 2.5 如何面对职场PUA

"小王啊，你看看隔壁组的小张，人家周末都在加班呢..." "就这点代码量，你要写到什么时候？" "你这个年纪的时候，我早就是高级工程师了..." "不加班？你是不是对公司没有归属感啊？"

听着耳熟吗？这不是普通的说教，这是赤裸裸的职场PUA。

4.6.1 什么是职场PUA？

简单说，就是用各种"看似合理"的方式，打击你的自信，控制你的行为。

常见的PUA话术

产品经理版：

- "其他开发都说这个需求很简单啊"
- "就改个小需求，怎么这么久还没好？"
- "你看某某某，人家从来不说做不到"

技术领导版：

- "这么简单的bug，你怎么会犯？"
- "代码写成这样，你是怎么通过面试的？"
- "你的代码水平好像没什么进步啊"

HR版：

- "你看看同期的小李，人家都升职了"
- "你觉得就你这表现，年终奖能拿多少？"
- "现在外面行情不好，你要好好珍惜这个机会"

4.6.2 为啥会被PUA？

1. 江湖地位太低

- 刚毕业没经验
- 技术深度不够
- 资历尚浅
- 没有话语权

就像我刚工作那会儿，改个代码还要被喷十遍，写个需求要改八百遍，天天被说"这都不会？"。

2. 不懂套路

- 以为多做就能出头
- 以为忍让就能平安
- 以为付出就有回报
- 以为老实就不会挨欺负

结果呢？越老实越被欺负，越忍让越得寸进尺。

3. 不敢反抗

- "万一得罪领导怎么办？"
- "得罪HR会不会被穿小鞋？"
- "现在找工作不容易啊..."
- "忍忍就过去了吧..."

4.6.3 PUA的后果有多严重？

1. 身心俱疲

- 工作没激情
- 睡觉睡不好
- 上班心慌慌
- 看到某些人就胃疼

2. 职业发展受阻

- 自信心被打击
- 创造力被压制
- 主动性被消磨
- 成长空间被限制

3. 越来越卷

今天：

- "周末加个班吧" 明天：
- "这个月多做点吧" 后天：
- "你看看人家..."

4.6.4 如何应对职场PUA？

1. 擦亮眼睛，识别PUA

正常的批评：

- "这段代码可以优化一下，我们一起看看"
- "这个bug下次注意下，我来教你排查思路"
- "最近进度有点慢，是不是遇到什么困难？"

PUA式批评：

- "就这么简单的代码都写不好？"
- "这种低级bug都会犯，你是怎么想的？"
- "你这个效率，怎么做程序员的？"

2. 建立自我防护

- 记录工作内容
- 每天做了什么
- 解决了什么问题
- 贡献了什么价值
- 留存证据
- 保存聊天记录
- 保存邮件往来
- 记录关键会议内容
- 设立边界
- 工作时间要有度
- 加班要有补偿
- 职责要有边界

3. 学会正面回应

PUA：这么简单的需求，你怎么做这么久？ 回应：

- "这个需求涉及到A、B、C三个模块的改动，我评估需要3天时间"
- "我这边已经完成了70%，还有哪些地方你觉得可以加快？"
- "如果你觉得时间太长，我们可以一起看看有什么可以优化的地方"

PUA：你看看人家小张，周末都在加班... 回应：

- "我更注重效率，周内我都会提前规划好工作"
- "我的工作量和产出都达到了要求，有问题吗？"
- "我们是按产出评估，还是按加班时间评估？"

4. 准备后路

- 保持技术更新
- 扩展人脉网络
- 关注市场机会

4.6.5 一些特别提醒

1. 不要期待PUA者改变

他们不会改变，因为PUA对他们来说是有效的管理工具，他们可能也是被PUA出来的，他们觉得这样做没问题。

2. 保护好自己

自己的身体健康最重要，重视自己的心理健康，规划自己的职业发展，该走的时候要走。

3. 警惕自己变成PUA者

- 当了领导不要学这套
- 带新人要以理服人
- 评审代码要就事论事
- 工作沟通要讲道理

4.6.6 结语

最后的叮嘱：

- 工作只是一份工作
- 公司只是一家公司
- 领导只是一个领导
- 你的人生不该被PUA毁掉

职场PUA就像代码里的死循环：发现得早跳出来还来得及，发现得晚可能整个系统都崩溃。

4.6.7 评论

4.7 2.6 内耗，是从拒绝沟通开始的

"凭什么又是我改bug？" "天天加班到这么晚，他们怎么都走了？" "这个需求改了八百遍了..." "我的付出怎么没人看到？"

你是不是也经常这样，一个人生闷气？ 憋着一肚子情绪，但就是不说。 别人该干嘛干嘛，你自己越想越气。

4.7.1 一个人的内耗现场

早上：

- 看到测试提了十个bug，默默打开电脑
- 产品经理又改需求了，默默接受
- 同事代码又出问题了，默默帮忙改
- 自己的任务堆积如山，默默加班

晚上：

- 躺在床上开始翻来覆去
- 越想越气，越气越想
- 第二天顶着黑眼圈继续
- 周而复始，循环往复

4.7.2 为什么我们喜欢憋着？

1. 觉得说了也没用

- "反正说了他也不会改"
- "说出来显得我很计较"
- "忍忍就过去了吧"

- "大家都这样，我说什么呢"

2. 不知道怎么说

- 怕说得太直接伤害关系
- 怕说得不够专业没人信
- 怕说出来自己承担不起后果
- 怕说了之后更加尴尬

3. 习惯性忍让

- "我脾气好，我能忍"
- "得饶人处且饶人"
- "大不了我多干点"
- "反正我都习惯了"

4.7.3 不说出来的代价

1. 情绪滚雪球

今天： "这需求改得也太随便了..." 明天： "又是这样，真是受够了！" 后天： "我干脆离职算了！"

2. 别人还蒙在鼓里

- 你气得要死，别人还在快乐摸鱼
- 你熬夜改bug，别人还在追剧
- 你怨气冲天，别人还觉得你挺好说话
- 你快崩溃了，别人觉得一切正常

3. 问题永远解决不了

- 你不说，别人不知道
- 别人不知道，问题依旧
- 问题依旧，你更生气
- 你更生气，更不想说
- 更不想说，问题更多...

4.7.4 如何走出自我内耗？

1. 承认自己的情绪

- 生气是正常的
- 委屈是应该的
- 不爽要表达
- 难受要说出来

2. 学会表达

不好的表达： "你们天天改需求烦不烦啊！" "谁写的代码这么烂！"

好的表达： "这个需求改动比较大，能不能先确定下具体方案？" "这段代码可能需要优化，我们一起看看？"

3. 寻找合适的方式

- 对于一些敏感问题，约个1v1沟通
- 对于一些公开问题，在周会上提出来
- 对于一些复杂问题，写个文档说明问题
- 对于一些情绪问题，找个合适的人倾诉

4.7.5 一些小建议

遇到问题先问问自己，我为什么不开心？我期望的是什么？怎么说对方才能听进去？有什么更好的解决方案？

然后：深呼吸，整理思路，选个合适的时机，好好说出来。

4.7.6 最后说点啥

作为一个成熟的程序员，我们表面上云淡风轻地说着"没事"，内心却像一个即将超载的服务器；以为自己的不爽写在了脸上，结果大家都忙着赶自己的需求；看似风平浪静地坐在工位上，其实心里已经演完了十八集职场大戏。

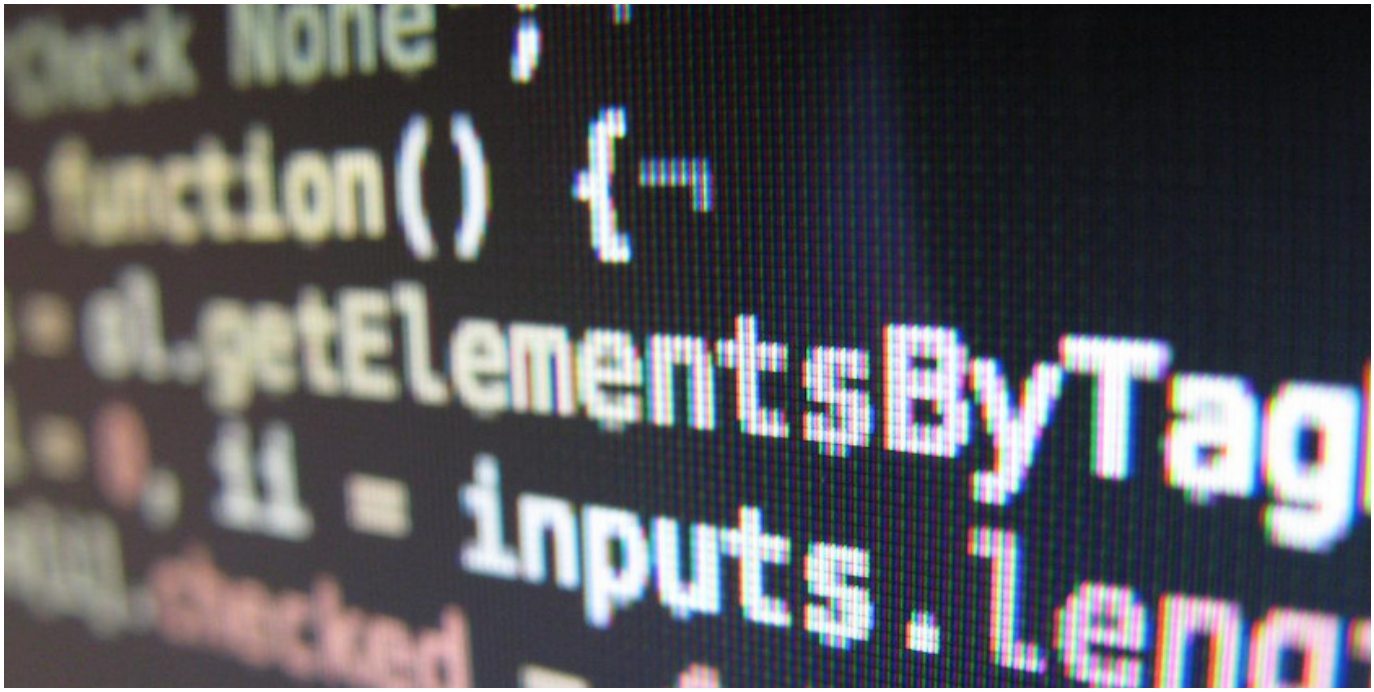
这样，真的好吗？

与其内耗自己，不如责怪他人，哈哈，开玩笑的，与其一个人在工位上生闷气，不如约个奶茶，聊聊天？

（诶，说完感觉舒服多了，今天的代码都写得动了...）

4.7.7 评论

4.8 2.7 不喜欢编程，还要坚守程序员这个岗位吗



"我好像不喜欢编程了..." "每天写代码好痛苦..." "看到IDE就头疼..." "要不要转行呢？"

最近在程序员群里，经常看到这样的声音。 但是等等，真的是不喜欢编程吗？

4.8.1 先问问自己：真的是不喜欢编程？

还记得当年为什么选择做程序员吗：

- 第一次写出Hello World的兴奋
- 第一次解决bug的成就感
- 第一次上线项目的激动
- 第一次被用户夸赞的自豪

那个时候，我们不也是在写代码吗？

4.8.2 让我们挖深一点

1. 是不喜欢编程，还是不喜欢：

- 天天改不完的bug
- 永远改不完的需求
- 说不清的加班
- 解释不完的进度

就像我一个朋友说的： "我喜欢写代码，但我讨厌天天被追着问进度" "我喜欢编程，但我受够了半夜三更的线上问题"

2. 是不喜欢编程，还是因为：

- 写了三年CRUD没成长

- 技术栈还停留在入职时
- 天天被业务压得喘不过气
- 从来没有获得过正面反馈

一个同事跟我吐槽： "代码写得好不好没人关心，bug修得快不快所有人都盯着" "系统不出事，没人会觉得你有多重要，系统出事了，所有人都会骂你"

3. 是不喜欢编程，还是因为：

- 团队氛围太压抑
- 技术氛围太浮躁
- 同事关系太复杂
- 领导风格太操蛋

4.8.3 为什么会失去对编程的热情？

1. 没有正反馈

记得有个同事说： "我优化了整个系统的性能， 领导说：这不是应该的吗？

我加班改了一个小bug， 领导说：这么简单的问题，至于加班？"

日复一日，谁还会热情得起来？

2. 看不到成长

- 业务代码写了一万行
- 技术深度还是那么深
- 职级还在原地踏步
- 薪资涨幅赶不上物价

3. 迷失在日常中

- 早上改bug
- 中午改bug
- 下午改bug
- 晚上还在改bug

每天像个修电工， 哪里漏电补哪里。

4.8.4 如何找回对编程的热情？

1. 重新定义你和编程的关系

不要把编程仅仅当作：

- 完成任务的工具
- 赚钱的手段
- 应付老板的工作

试着把它当作：

- 解决问题的能力
- 创造价值的技能
- 实现想法的途径

2. 主动制造正反馈

- 写个自己感兴趣的小项目
- 参与开源社区
- 记录技术博客
- 分享技术心得

3. 找到成长的方向

与其抱怨：

- 天天写CRUD没技术含量
- 业务代码没什么营养
- 工作内容太单调

不如：

- 从CRUD中抽取通用组件
- 在业务中挖掘技术价值
- 把重复的工作自动化

4. 创造属于自己的空间

- 每天留点时间学习
- 每周抽空看看技术文章
- 每月完成一个小目标
- 每季度总结一下成长

4.8.5 那么，要不要继续做程序员？

其实，选择权一直在你手上：

如果你发现：

- 真的对编程没有一点兴趣
- 写代码让你痛不欲生
- 看到电脑就想摔键盘
- 对技术完全提不起劲

那么，转行也未尝不可。

但如果你发现：

- 解决问题时还会有成就感
- 写出好代码时还会有快感
- 学到新技术时还会有兴奋感
- 只是被一些外部因素影响了

那么，也许你需要的只是换个环境，换个方向，或者换个心态。

4.8.6 最后的碎碎念

其实程序员这个职业有人把它当成事业，有人把它当成工作，有人把它当成跳板，有人把它当成爱好。

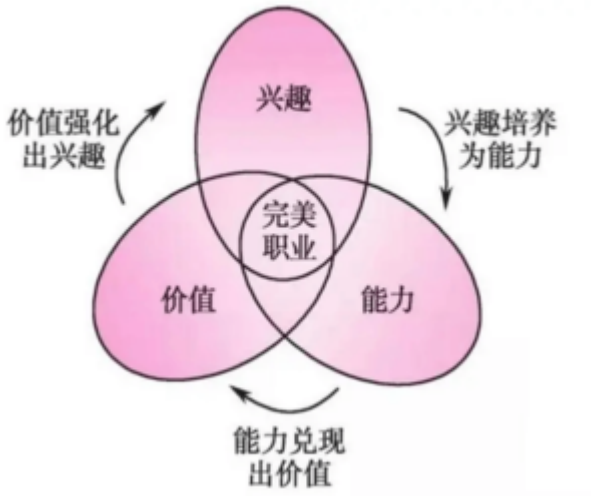
没有对错，只有选择。

重要的不是你选择继续还是放弃， 而是你要搞清楚， 到底是不喜欢编程本身， 还是不喜欢编程之外的那些东西。

（写完这篇文章，我好像又找到了一点写代码的动力...）

4.8.7 评论

4.9 2.8 工作倦怠了吗，试试三叶草模型



"最近上班好累，上班如上坟.." "不是身体累，就是...提不起劲" "天天就想摸鱼，对工作提不起一点兴趣"

如果你也有这种感觉，不妨用三叶草模型来给自己做个诊断。

4.9.1 什么是三叶草模型？

三叶草模型是一个职业生涯规划模型，它把工作动力分成三片叶子：

- 兴趣叶：对工作的热情和喜爱程度
- 能力叶：解决问题的技术和才干
- 价值叶：工作带来的回报和意义

这三片叶子相互影响，缺一不可：

- 有兴趣，学习能力才会提升
- 有能力，才能创造更多价值
- 有价值，才能持续保持兴趣

4.9.2 三种倦怠类型

1. 厌倦型（兴趣叶枯萎）

表现：

- 天天盼着下班
- 看到代码就烦
- 对什么都提不起劲
- 工作完全是为了完成任务

就像我一个同事说的： "以前看到新技术就兴奋，现在看到新框架就头大" "记得刚工作那会儿，周末还会自己写代码，现在连IDE都不想打开"

2. 焦虑型（能力叶不足）

表现：

- 总觉得跟不上节奏
- 害怕接到新需求
- 看不懂同事的代码
- 技术分享时坐立难安

一个典型场景："产品经理说这个很简单，可我看了半天也没头绪..." "同事两天就能写完的功能，我要写一周..."

3. 失落型（价值叶缺失）

表现：

- 付出没有回报
- 努力得不到认可
- 看不到职业发展
- 觉得自己是工具人

常见的抱怨："我优化了整个系统性能，领导说'这不是应该的吗？'" "加班做完的方案，第二天被一句'需求变了'否定"

4.9.3 如何治愈倦怠？

1. 厌倦型的治疗方案

第一步：找回兴趣的源头 - 回忆当初为什么选择编程 - 列出曾经让你兴奋的技术点 - 想想最有成就感的项目

第二步：重新培养兴趣 - 尝试一个自己感兴趣的side project - 研究一下新技术或者开源项目 - 和志同道合的同事一起做点东西

第三步：转化兴趣 - 把枯燥的工作游戏化 - 在日常工作中设立小目标 - 给自己设定技术挑战

2. 焦虑型的治疗方案

第一步：正视能力差距 - 列出具体的技能短板 - 设定合理的学习目标 - 制定可执行的计划

第二步：系统提升 - 每天抽固定时间学习 - 参与有挑战性的项目 - 向高手请教和学习

第三步：发挥优势 - 专注于自己擅长的领域 - 把已有技能做到极致 - 找到适合自己的位置

3. 失落型的治疗方案

第一步：重新定义价值 - 不只看外在回报 - 关注个人成长 - 寻找工作的其他意义

第二步：创造价值 - 主动承担重要任务 - 解决团队痛点问题 - 提升工作的可见度

第三步：寻找平台 - 选择更适合的团队 - 找到欣赏自己的领导 - 寻找价值观一致的环境

4.9.4 一些特别提醒

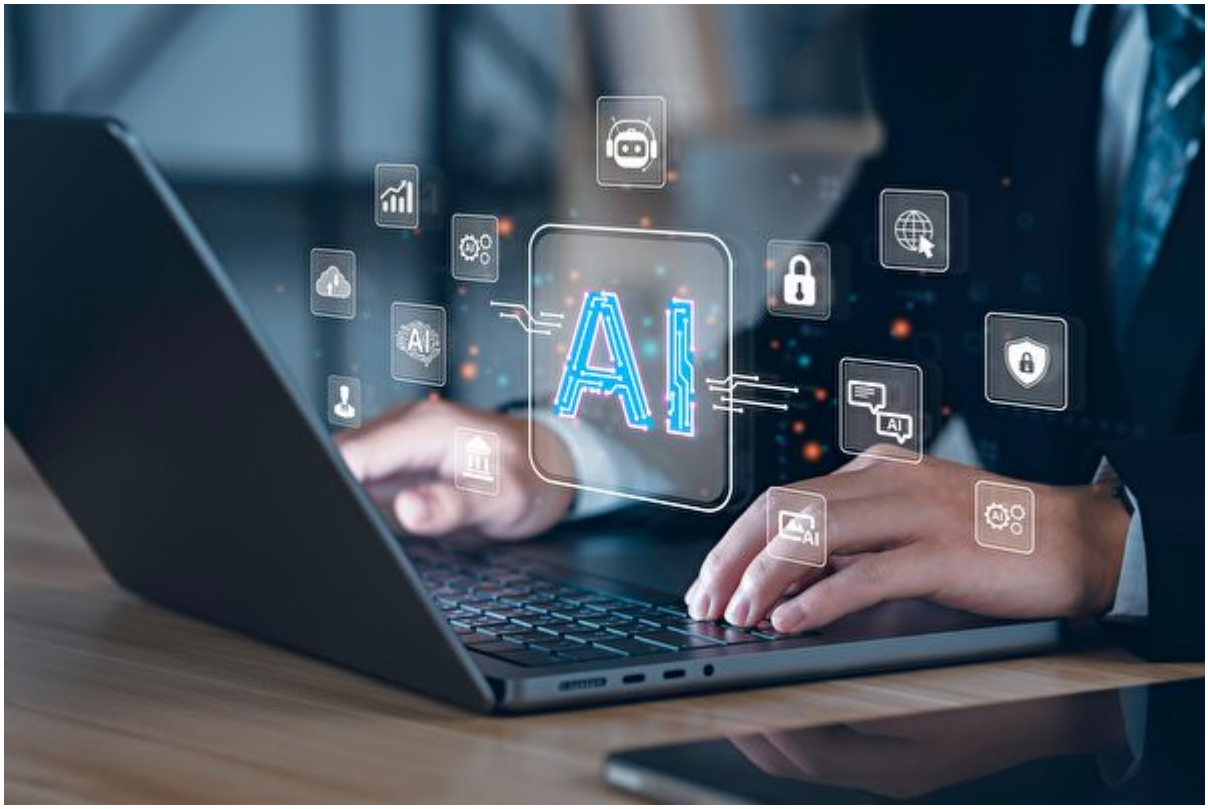
1. 三片叶子是相互影响的，找回兴趣，能力自然会提升，提升能力，价值自然显现，获得价值，兴趣会更浓
2. 治愈倦怠需要时间，不要期待立竿见影，给自己一个缓冲期，保持耐心和信心。
3. 记住选择权在你，可以换个团队，可以转换方向，可以寻找新机会。

就像调试代码一样：先定位问题（哪片叶子出问题了），再分析原因（为什么会这样），最后解决问题（对症下药）。

程序员嘛，修bug的时候都那么执着， 修复自己的倦怠，也要这么认真才行。

4.9.5 评论

4.10 2.9 天天用AI，以后自己不会写代码怎么办



"用AI写代码，会不会显得我很菜？" "天天用AI，以后自己不会写代码怎么办？" "依赖AI工具，会不会显得我不专业？"

最近在程序员群里，有时会看到这样的焦虑。

且慢，让我们先问个问题： 当年从汇编转到高级语言时，前辈们是不是也这么焦虑？ 当年从手写SQL转到ORM时，大家是不是也这么担心？

4.10.1 我们在担心什么？

1. "不会写代码了"

真的吗？ - 用了IDE的自动补全，你忘记怎么写for循环了吗？ - 用了Lombok，你忘记怎么写getter/setter了吗？ - 用了Spring Boot，你忘记怎么写XML配置了吗？

（好吧，最后一个确实有点忘了...）

2. "显得我很菜"

等等，你的竞争力真的在于：

- 手写一个快速排序？
- 背诵设计模式的定义？
- 记住所有API的参数？

还是在于：

- 理解业务需求的能力
- 解决实际问题的思路
- 架构设计的水平
- 技术选型的判断

3. "不够专业"

那么问题来了：

- 用搜索引擎查资料算不算专业？
- 用Stack Overflow找答案算不算专业？
- 用GitHub Copilot写代码算不算专业？

专业的定义是：

- 解决问题的效率
- 输出结果的质量
- 维护代码的可靠性

4.10.2 AI是工具，不是敌人

1. AI的优势

- 帮你写重复的模板代码
- 帮你找到常见的解决方案
- 帮你优化代码结构
- 帮你提高编码效率

就像：

- IDE帮你自动补全
- Git帮你管理代码
- Maven帮你管理依赖
- Docker帮你部署服务

2. 你的优势

- 理解业务场景
- 设计系统架构
- 权衡技术选型
- 把控代码质量
- 解决复杂问题

AI写不了的是：

- 产品经理的脑回路
- 老板的临时需求
- 客户的特殊要求
- 遗留系统的坑

4.10.3 如何正确使用AI？

1. 用AI提高效率

- 生成样板代码
- 编写单元测试
- 重构老代码
- 优化代码结构

解放双手， 专注于真正需要思考的问题。

2. 用AI学习成长

- 让AI解释复杂代码
- 让AI提供最佳实践
- 让AI推荐设计方案
- 让AI分析性能问题

站在巨人的肩膀上， 看得更远，学得更快。

3. 用AI激发创意

- 讨论技术方案
- 探索解决思路
- 寻找新的可能
- 突破思维定式

AI不是你的替代品， 而是你的助手和老师。

4.10.4 一些建议

1. 建立正确认知

不要把AI当作写代码的黑盒，解决问题的万能钥匙，或者逃避思考的借口，而要把AI当作提高效率的工具，辅助思考的伙伴，激发创意的助手。

2. 掌握使用技巧

学会提出好问题，学会验证AI的答案，学会结合实际场景，学会举一反三。

3. 保持学习心态

理解AI给出的方案，思考背后的原理，总结经验教训，持续提升自己。

4.10.5 最后说点啥

其实，程序员最核心的能力是：

- 解决问题的思维
- 学习成长的能力
- 工程化的素养
- 技术的洞察力

代码只是表达这些能力的载体， 工具只是实现这些能力的手段。

与其纠结要不要用AI， 不如想想怎么用好AI。

就像当年： "不用IDE，手写代码才是真本事！" "不用框架，从零开始才是硬核！" "不用云服务，自己搭建才够专业！"

现在想想，还挺逗的。

4.10.6 评论

4.11 2.10 职场中如何做选择

"要不要跳槽？" "要不要转管理？" "要不要接这个项目？" "要不要去创业公司？"

职场就是一个不断做选择的过程。 但很多时候，我们纠结得像在debug一个随机bug。

4.11.1 选择困难症患者日常

1. 跳槽选择困难

- 现在公司虽然不完美，但是很稳定
- 新公司给的多，但是不知道坑多不多
- 现在同事关系不错，换了环境要重新适应
- 要不...再等等？

2. 方向选择困难

- 继续做技术还是转管理？
- 专精一个方向还是广泛发展？
- Java转Go好还是转Python好？
- 要不...都学学？

3. 项目选择困难

- 老项目虽然烂，但是熟悉
- 新项目虽然好，但是风险大
- 这个有挑战，但是可能会很累
- 要不...随便吧？

4.11.2 为什么选择这么难？

1. 信息不对称

- 新公司看起来很好，但实际呢？
- 新技术很火，但会不会昙花一现？
- 新项目很酷，但坑有多深？

就像买房： 看起来装修不错， 但你永远不知道邻居半夜会不会放音乐。

2. 结果不确定

- 跳槽可能升职加薪，也可能处处碰壁
- 转管理可能平步青云，也可能两头不讨好
- 接新项目可能大放异彩，也可能翻车现场

3. 机会成本

- 选A就意味着放弃B
- 选这个就要错过那个
- 现在不选以后可能没机会了

4.11.3 两个实用的决策思路

1. "Hell Yeah" or "No"

遇到选择时，问问自己： "这个机会让我兴奋到想大喊'Hell Yeah!'吗？"

如果你的回答是："还可以吧"，"还不错诶"，"蛮好的"，"值得考虑"，

对不起，这些都不是"Hell Yeah!" 那就应该是"No"。

2. 未来的故事法则

再问问自己： "这会是一个值得在饭局上开心分享的故事吗？"

好故事的样子：

- "我带领团队重构了整个系统架构..."
- "我们在一个月内把性能提升了10倍..."
- "我从零开始组建了公司的AI团队..."

无聊故事的样子：

- "我在那混了三年，每天按时下班..."
- "工作还行，工资还可以..."
- "就那样呗，还能咋样..."

4.11.4 如何利用这两个思路做选择？

1. 跳槽决策

不要问：

- 新公司工资高多少？
- 新公司福利好不好？
- 新公司名气大不大？

要问：

- 想到去这家公司，我兴奋吗？
- 这份工作会让我有好故事讲吗？
- 三年后，我会为这个选择自豪吗？

2. 技术方向

不要问：

- 这个技术火不火？
- 这个方向赚不赚钱？
- 学习曲线陡不陡？

要问：

- 这个技术让我热血沸腾吗？
- 这个方向能让我有成就感吗？
- 这会是我职业生涯的精彩一笔吗？

3. 项目选择

不要问：

- 这个项目简单不简单？
- 这个项目有没有风险？
- 这个项目能不能按时完成？

要问：

- 这个项目够挑战吗？
- 这个项目能让我成长吗？
- 这个项目值得我投入吗？

4.11.5 一些特别提醒

1. "Hell Yeah"不是盲目乐观

不是不考虑风险，不是不考虑现实，而是在充分评估后的由衷兴奋。

2. "好故事"不是吹牛

不是追求表面光鲜，不是为了炫耀，而是真实的成长和收获。

3. 找不到"Hell Yeah"的选择？

可能是视野要再开阔些，可能是能力要再提升些，可能是方向要再调整些。

4.11.6 最后的碎碎念

职场选择就像写代码：不要只看表面的需求，不要只顾眼前的实现，要考虑长期的可维护性，要思考未来的扩展性。

最后的最后： 你的职业生涯是一个个选择堆积而成的故事， 要么激动人心到值得说"Hell Yeah!" 要么干脆就不要写进去。

4.11.7 评论

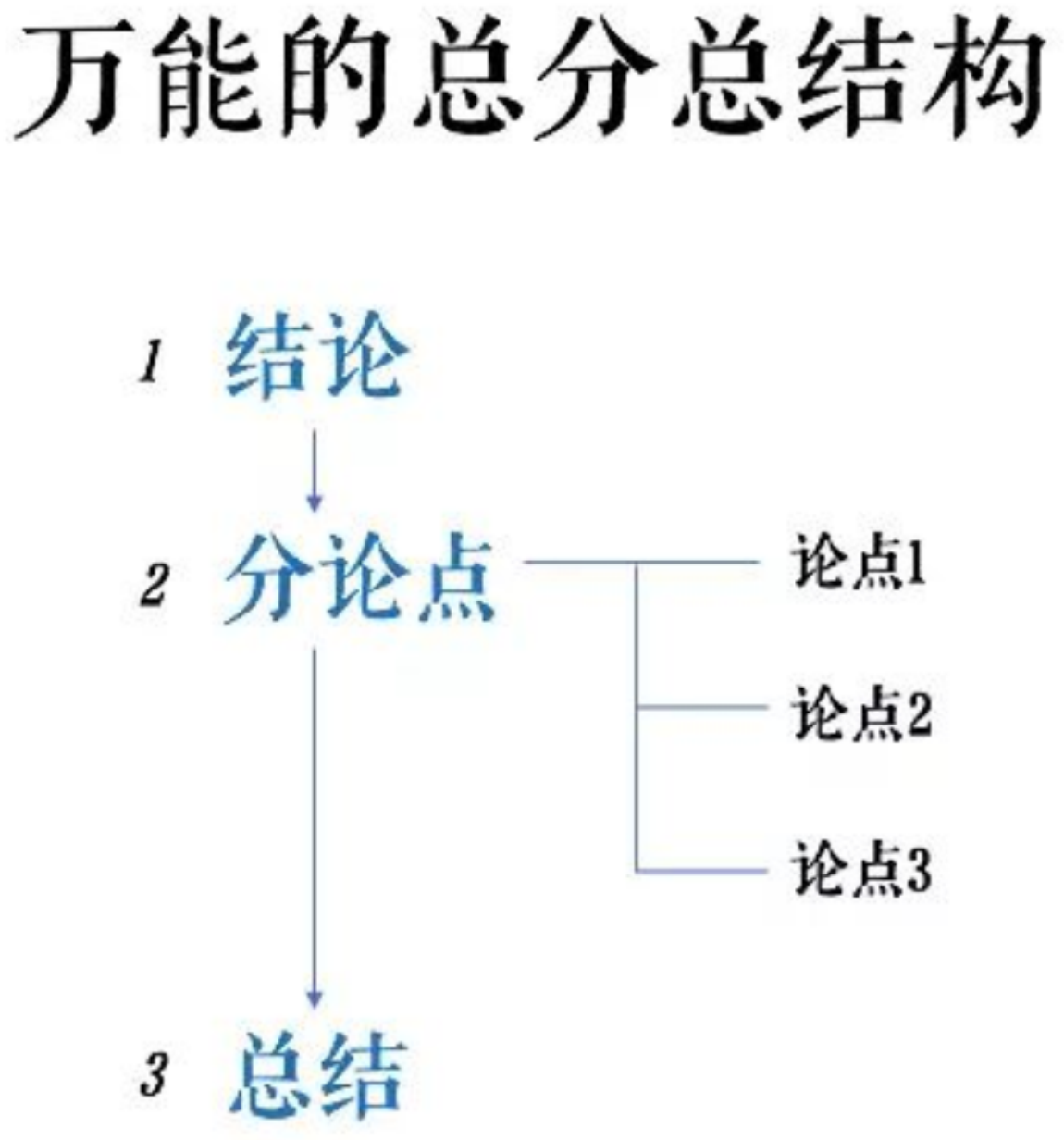
4.12 2.11 职场中最实用的三个思维模型

"这个需求怎么说清楚？" "这堆任务怎么排优先级？" "述职报告怎么写？"

每次遇到这些问题，是不是感觉脑子有点乱？ 今天我们就聊三个特别实用的思维模型。

不是那种看完就忘的"高大上"框架， 而是真的能用、常常用、越用越顺手的工具。

4.12.1 一、总分总模型：让表达更清晰



什么是总分总？

简单说就是：先说结论（总），再说细节（分），最后重申（总）。

就像写周报："本周完成了订单系统优化（总）具体包括修复了三个性能问题，优化了两个接口，新增了监控（分）整体性能提升了30%（总）"

实战场景

1. 写周报

差的版本："这周修了三个bug，开发了两个功能，参加了四个会议..."（领导看完：所以重点是什么？）

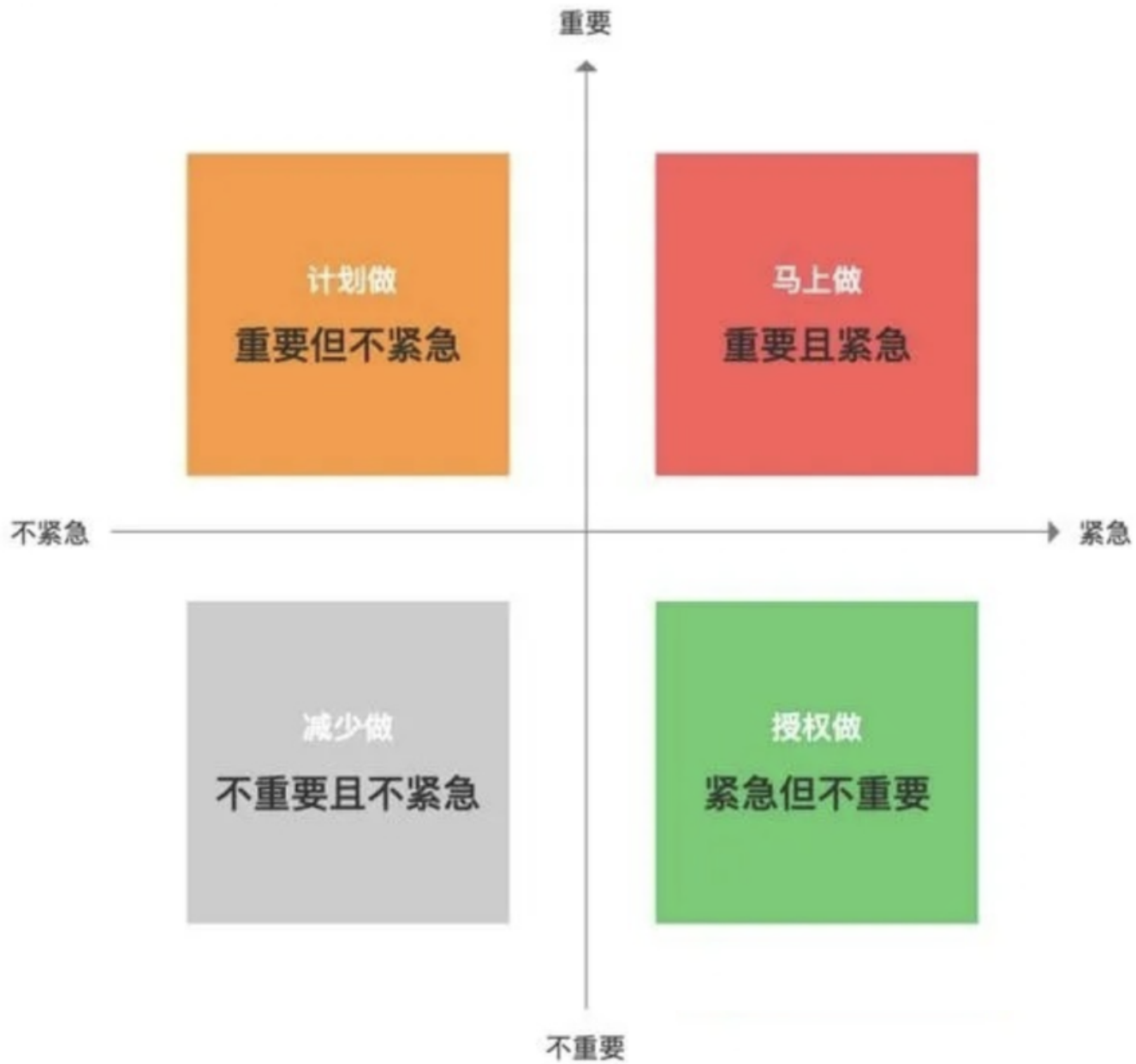
好的版本："这周主要完成了订单系统的性能优化（总）- 修复了三个性能瓶颈 - 优化了两个核心接口 - 系统响应提升了30% 整体达到了预期的优化目标（总）"

2. 技术方案汇报

差的版本："我们要用这个技术，那个框架，还要用什么什么..."（听众：所以你要做什么？）

好的版本："这个方案主要解决订单峰值问题（总）- 优化系统架构 - 增加缓存机制 - 改进数据处理 通过这些措施，系统可以支撑双十一峰值（总）"

4.12.2 二、优先级四象限：让工作更高效



什么是优先级四象限？

把任务按照"重要"和"紧急"分成四类：

- 重要且紧急（第一象限）
- 重要不紧急（第二象限）
- 紧急不重要（第三象限）
- 不紧急不重要（第四象限）

实战场景

1. 处理各种任务

重要且紧急：

- 线上故障
- 重大bug
- 老板要的报告

重要不紧急：

- 系统重构
- 技术升级
- 团队建设

紧急不重要：

- 各种会议
- 日常报表
- 临时帮忙

不紧急不重要：

- 茶水间八卦
- 日常闲聊
- 整理工位

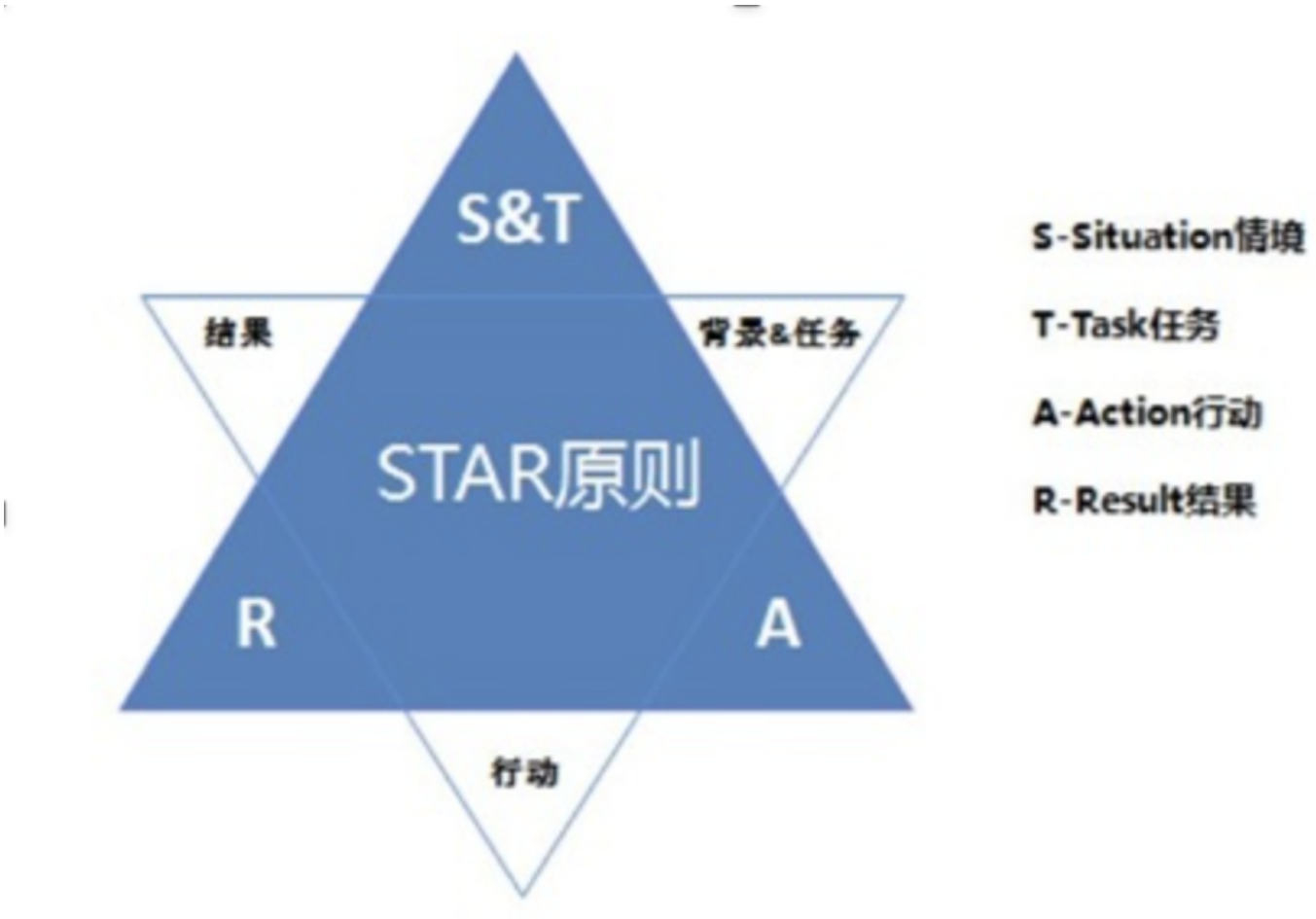
2. 排期安排

早上来到公司：

- 产品经理说有个紧急需求
- 测试说发现个严重bug
- 运维说要升级系统
- 同事说要帮看个问题

按四象限排序： 1. 先处理严重bug（重要且紧急） 2. 再看紧急需求（评估重要性） 3. 安排系统升级（重要不紧急） 4. 最后帮看问题（不紧急不重要）

4.12.3 三、STAR模型：让经历更有说服力



什么是STAR？

- Situation（情境）：什么背景
- Task（任务）：做什么事
- Action（行动）：怎么做的
- Result（结果）：达到什么效果

实战场景

1. 写述职报告

差的版本："我负责了订单系统的开发维护，完成了性能优化..."

好的版本： "Situation：订单系统在双十一期间频繁超时 Task：需要优化系统性能，确保双十一稳定性 Action：

- 梳理系统瓶颈
- 优化数据库查询
- 引入缓存机制
- 添加监控告警 Result：
- 系统响应时间降低70%
- 双十一零故障
- 经验在团队内推广"

2. 面试介绍项目

差的版本： "我做过订单系统，用了各种技术框架..."

好的版本： "S：公司准备双十一大促 T：负责订单系统的性能优化 A：通过以下步骤：

- 性能压测定位问题
- 优化系统架构
- 增加缓存机制
- 改进数据处理 R：系统最终支撑了双十一百万订单"

4.12.4 这三个模型的配合使用

1. 写文档时

- 用总分总组织整体结构
- 用四象限突出重点内容
- 用STAR描述关键实践

2. 解决问题时

- 用总分总梳理思路
- 用四象限确定优先级
- 用STAR总结复盘

3. 汇报工作时

- 用总分总组织语言
- 用四象限展示重点
- 用STAR讲述成果

4.12.5 最后的建议

1. 少即是多

- 与其记100个用不上的模型
- 不如精通3个天天都用的工具

2. 多实践

- 不要只是记住
- 要在日常工作中刻意练习

- 直到形成肌肉记忆

3. 灵活运用

- 模型是工具不是教条
- 根据场景灵活调整
- 找到最适合自己的方式

记住： 思维模型就像我们常用的工具， 不是放在工具箱里就有用， 而是要经常拿出来用， 用着用着就顺手了。

（这篇文章就是用总分总写的，你发现了吗？）

4.12.6 评论

4.13 不会吹牛逼，述职/晋升总是吃亏怎么办？

在职场中，很多人都有这样的困扰：明明做了很多事情，但是一到述职和晋升的时候，就不知道该怎么表达，总觉得说出来就是在吹牛。其实，这往往不是能力的问题，而是表达方式的问题。

4.13.1 🤔 为什么我们不擅长"吹牛"

很多程序员都有这样的心理： - "我做的事情不够出彩，说出来怕别人笑话" - "我只是完成了分内的工作而已，有什么好说的" - "说多了显得在炫耀，说少了又显得没做事" - "别人说起工作头头是道，我说起来就觉得很平淡"

其实，这种心理很常见，尤其在程序员群体中。我们习惯于用代码说话，习惯于把事情做好就行，不习惯"包装"自己。但问题是，在现代职场中，**展示自己的工作成果，本身就是工作能力的一部分。**这方面，印度三哥就是我们值得学习的榜样。 笔者还记得在前司的时候，有一次公司内部的述职会，我们都是几页PPT就糊弄了，印度三哥默默拿出了自己剪辑的述职视频，还有各种特效，把我们都看呆了。

4.13.2 💡 这不是"吹牛"，而是"讲故事"

首先，我们要转变一个观念：**合理展示自己的工作成果，不是在吹牛，而是在帮助他人了解你的价值。**

1. 从"我做了什么"到"我解决了什么问题"

❌ 不好的表达： "我优化了数据库查询性能"

✅ 好的表达： "发现用户反馈系统响应慢，通过分析发现是数据库查询效率低下。优化后，接口响应时间从 3s 降到 300ms，用户体验显著提升，差评率下降 30%"

2. 用数据说话

不要觉得数据是在炫耀，恰恰相反，数据是最客观的表达方式： - 代码覆盖率提升了多少 - 系统性能提升了多少 - 节省了多少人力成本 - 减少了多少线上故障

3. 突出你的思考过程

很多时候，我们不仅要说结果，更要说思考过程： - 为什么选择这个解决方案 - 考虑过哪些备选方案 - 权衡了哪些利弊 - 学到了什么经验教训

4.13.3 📝 如何准备述职报告

1. 建立工作日志习惯

最怕的不是记不住，而是根本就没记录

平时就要养成记录的习惯： - 每周五花 10 分钟总结本周完成了什么 - 遇到重要节点及时记录（比如解决了一个疑难 bug） - 记录下数据变化（性能提升、用户反馈等）

2. STAR 法则

讲述一个工作事例时，可以遵循 我们在之前文章提到过的STAR 法则： - **Situation**：什么场景下 - **Task**：需要解决什么问题 - **Action**：采取了什么行动 - **Result**：取得了什么成果

3. 突出你的独特价值

每个人都有自己的特点，要懂得发挥自己的优势： - 如果你觉得自己技术吊，就多说技术难点攻克 - 如果你觉得自己沟通能力强，就多说项目推进 - 如果你觉得自己产品思维好，就多说业务价值

4.13.4 述职/晋升中的加分项

1. **主动性**
2. 不是被动完成任务，而是主动发现问题
3. 不是等待分配，而是主动承担责任
4. **影响力**
5. 带动了团队成员一起进步
6. 经验分享被其他团队采纳
7. 解决方案变成最佳实践
8. **成长性**
9. 展示你的学习能力
10. 说明你如何突破自己的舒适区
11. 分享你对未来的规划

4.13.5 改变从现在开始

记住，合理展示自己不是虚荣，而是职场中必要的软技能。就像代码需要注释一样，你的工作也需要合适的表达。

从以下几个方面开始改变： 1. 养成记录的习惯，让数据说话 2. 学会用 STAR 法则组织语言 3. 多听优秀同事是如何表达的 4. 在日常工作中就开始练习表达

最后，请记住：**真诚永远是最好的包装**。不要为了晋升刻意去"包装"自己，而是学会更好地展示真实的自己。因为真诚的表达，才能赢得真正的尊重。

4.13.6 评论

4.14 聊聊职场程序员那些真正重要但却经常被忽略的事

大半夜的，窗外霓虹闪烁，我端着一个大茶瓷缸子，看着显示器上那一行行代码，突然想起今天看到的脉脉上的一个帖子：一个技术大牛，写代码贼溜，可就是混不上去。这让我想起高盛那四句话，忍不住想唠叨两句。

4.14.1 那些年，我们都看错的事

记得刚入行那会儿，我跟大多数人想的一样：技术牛逼就能横着走。那时候为了学新技术，经常熬到凌晨三四点，第二天顶着两个黑眼圈去上班，就觉得自己特别牛。

结果呢？干了几年才发现，技术好的人多了去了，可是能往上走的，往往不是技术最强的那个。

高盛挑人的时候特别看重四点：

1. 软实力比专业知识重要

2. 与人沟通的能力重要

3. 解决问题的思路比知道答案重要

4. 知道自己边界，学会求助很重要

这几点，我也是工作几年才真正整明白的。

4.14.2 软实力：那些看不见的竞争力

说起软实力，很多人第一反应就是：这不就是拍马屁吗？

我先给你讲个真事。

几年前我还在某大厂的时候，我们组来了个年轻人，技术一般，但这孩子特别有意思。每次开会他都会提前到，帮大家准备投影；有同事遇到问题，他二话不说就去帮忙；最绝的是，他还会主动收集大家工作中遇到的痛点，整理成文档分享出来。

半年后，他就成了我们组最受欢迎的人。再过一年，他就被提拔成了小组长。

有人说这是"向上管理"，我觉得这话说得片面了。软实力根本不是什么花花肠子，按人话说它就是： - 靠谱：说到做到 - 担当：出了问题不甩锅 - 同理心：能设身处地为他人着想 - 学习能力：不固步自守 - 情商：懂得照顾他人感受

这些东西，说起来容易做起来难。因为它们都是需要长期坚持的习惯，不是说学两天就能掌握的。

4.14.3 沟通：职场的必修课

有一说一，我是吃过沟通不畅的亏的。

记得有次做一个跨部门项目，我这边写好了接口，测试也都通过了，但就是对接不上。找对方部门的人聊，说话吞吞吐吐的，就是不说问题出在哪。后来我直接杀到人家工位上，一聊才知道，人家觉得我们的接口设计有问题，但又不好意思直说。

从那以后，我就琢磨出来一套跨部门协作的招：

1. 先处好关系再谈事
2. 约人吃个饭
3. 一起抽根烟
4. 聊聊工作之外的事
5. 把对方当盟友，不要当对手
6. 有问题直接说，不藏着掖着
7. 多问问对方的想法
8. 一起找解决方案
9. 沟通要过度，不要嫌烦
10. 重要的事情说三遍
11. 开会记录要发出来
12. 口头约定要有书面确认

4.14.4 解决问题：方法论比答案重要

现在的面试，都喜欢问八股文。但在真正的职场中，背完答案就完事了？我看未必。

举个栗子：我们组前段时间遇到个性能问题，系统总是莫名其妙地卡顿。新来的小伙子一顿操作，各种工具上来就是干。忙活了一天，问题没解决，人倒是累得够呛。

最后是老王出手，他先画了个架构图，又抓了几个关键指标，最后定位到是某个服务的连接池配置有问题。整个过程，他花的时间反而比那个小伙子少。

这就是思路的重要性： - 先理清楚问题的边界 - 找到可能的突破点 - 验证假设 - 总结经验教训

这种解决问题的思路，才是真正的核心竞争力。

4.14.5 知道自己的能力边界，懂得求助，避免蛮干，很重要

关于这一点，我之前就犯过类似的错误。

那时候刚进入职场没多长时间，我手上分配了一个比较紧急，对我又有点挑战性的任务，我认为是一个表现自己的好机会，就加班加点的去做， 虽然最终勉强完成了任务，但是效果不是很好。

后来，我跟我们领导聊起这件事，她就直截了当的说，你为什么不向我求助呢，如果你当时问我一下，可能咱们两个一下午的时间，就把这个事情做完了，而不是你周末加班加点的去自己琢磨。之后你自己还为免费加班觉得委屈，我们还觉得你做的不好。

我自己之后反思了一下，确实，当时我的心态就是，这个任务我一定要自己独立完成，不能让我的同事看不起我。这种心态其实还是典型的学生思维。

每个人都有自己覆盖不到的知识面，要承认自己的能力边界，求助别人，并不是一件需要为此感到难堪的事情，其实，这也是为什么公司需要这么多人的原因，就是需要彼此的能力互补。

知道自己的能力边界，懂得求助，避免蛮干，很重要。

4.14.6 评论

5. 第三章 工作中的人际关系

5.1 第三章 职场中的人际关系

在职场中，人际关系就像是一个看不见但却无处不在的"操作系统"，它在默默影响着我们的工作效率、职业发展和个人成长。很多人觉得"能力才是硬道理"，但现实往往是：能力决定你能做什么，而人际关系决定你能走多远。

5.1.1 为什么人际关系这么重要？

首先，没有人是孤岛。在现代企业中，几乎所有工作都需要协作才能完成。一个项目从需求到上线，需要产品经理的规划、设计师的设计、开发的实现、测试的验证、运维的部署，缺一不可。好的人际关系能让这个过程更顺畅，遇到问题时更容易得到支持和理解。

其次，机会往往来自人际网络。很多好的机会，比如重要项目、晋升机会、职位空缺，往往在正式发布之前就通过非正式渠道传播了。良好的人际关系能让你更早获取这些信息，获得更多选择的机会。

再次，个人成长需要他人的帮助。无论是技术指导、经验分享，还是职业建议，都需要他人的支持。一个好的人际网络，就像给自己配备了一个强大的"智囊团"。

但需要注意的是，好的职场人际关系不是靠巴结、奉承或者刻意讨好来维系的。真正持久的职场关系，是建立在专业、真诚和互相尊重的基础上的。它需要我们：

- 在工作中展现专业能力
- 在协作时互相理解和支持
- 在交往中保持真诚和诚信
- 在合作中创造共赢价值

这一章，我们将深入探讨职场中的人际关系：如何建立、如何维护、如何在工作中更好地发挥人际关系的价值，让我们的职业发展之路走得更稳、更远。

在职场中，能力或许能让你赢得比赛，但好的人际关系却能让你赢得整个赛季。

5.1.2 评论

5.2 3.1 领导一对一（1on1）聊什么

"下周一对一，又不知道聊什么..." "每次都是领导问，我就答，感觉好被动" "一对一好尴尬，就像相亲..."

很多人都有这种困扰。 其实1on1是个难得的机会， 就看你会不会用。

5.2.1 为什么要重视1on1？

1. 这是你的专属时间

- 不是日常晨会
- 不是项目周会
- 不是绩效面谈
- 而是属于你的30分钟

2. 这是双向沟通的机会

- 不是领导训话
- 不是工作汇报
- 不是被动答问
- 而是平等的对话

3. 这是建立信任的时刻

- 不是表面客套

- 不是应付了事
- 不是互相防备
- 而是真诚交流

5.2.2 聊什么？

1. 工作进展

不要只说："项目都按计划进行..." "没什么特别的问题..."

要说：

- 遇到了什么挑战
- 如何解决的
- 学到了什么
- 还需要什么支持

2. 个人成长

主动分享：

- 最近在学什么
- 遇到什么瓶颈
- 有什么困惑
- 未来想往哪个方向发展

3. 团队建设

可以聊：

- 团队氛围如何
- 和同事协作情况
- 流程上有什么建议
- 团队还缺什么

4. 对公司的想法

可以谈：

- 对公司战略的理解
- 对产品的建议
- 对流程的改进想法
- 对团队发展的建议

5.2.3 怎么聊？

1. 会前准备

- 列个提纲
- 记录关键点
- 准备具体例子
- 想好你的诉求

比如： "最近在做性能优化，遇到几个难点，想和您讨论下..." "对未来的职业发展有些困惑，想听听您的建议..."

2. 会中技巧

- 先说重要的
- 用数据说话
- 多提建设性意见
- 注意倾听反馈

3. 会后跟进

- 记录关键点
- 落实行动项
- 跟进解决方案
- 下次汇报进展

5.2.4 几个常见场景

1. 遇到困难时

不要说： "这个问题好难，搞不定..."

要说： "我遇到这个问题， 已经尝试了这些方案， 但效果不理想， 想听听您的建议..."

2. 有想法建议时

不要说： "我觉得我们团队应该怎样怎样..."

要说： "我观察到这个问题， 分析原因可能是... 建议我们可以... 您觉得这个思路如何？"

3. 谈成长规划时

不要说： "我想往高级工程师发展..."

要说： "根据团队需要和个人兴趣， 我想在系统架构方面深入， 已经在学习相关知识， 希望能得到一些实践机会..."

5.2.5 注意事项

1. 要诚恳不要抱怨

- 多谈解决方案
- 少说牢骚
- 保持建设性
- 态度要积极

2. 要具体不要空泛

- 多举实例
- 少说空话
- 有数据支撑
- 重结果导向

3. 要主动不要被动

- 提前准备话题
- 主动分享想法
- 争取有效反馈
- 及时跟进行动

5.2.6 关于1on1的一些特别提醒

1. 不是吐槽大会

不要一味抱怨，不要传播负能量，不要说同事坏话，要保持专业性。

2. 不是表演时间

不要过分包装，不要避重就轻，不要夸大成绩，要保持真诚。

3. 不是单向汇报

不要只说成绩，不要只报进度，不要只答不问，要互动交流。

5.2.7 最后的建议

把1on1当作：自我提升的机会，建立信任的渠道，解决问题的平台，职业发展的助推器。

好的1on1不在于时间长短，而在于沟通的质量；不在于说了多少，而在于解决了什么。

5.2.8 评论

5.3 3.2 领导让我提意见，我该怎么提

"小王，你觉得咱们团队有什么问题不？"

这种时候，脑子里第一反应是不是： "完了完了，这是道送命题啊..." "要说没问题吧，显得我没见识，" "要说有问题吧，万一说错了怎么办？" "要不...先说'挺好的'保命要紧？"

别慌，这其实是个好机会， 就像产品经理难得让你提需求一样， 机会难得，且(pian)行且珍惜。

5.3.1 从哪些角度切入？

1. 团队士气

就是看看大家"精气神"：

- 最近大家是不是像霜打的茄子
- 周五下班冲刺速度是不是创新高
- 团建时候是不是都在看手机
- 每天的群聊是不是只剩工作汇报

（要是发现群里连表情包都没人发了，那问题可就大了）

2. 工作效率

观察下是不是有这种情况：

- 改个bug要三个部门盖章
- 一个需求开了五个会还没定
- 线上环境比测试环境还不稳定
- 每天光开会就能开饱

（如果发现自己一天的日历像彩虹一样绚丽，那就要考虑提提建议了）

3. 技术发展

看看团队是不是有这些征兆：

- 系统像补丁贴补丁
- 新技术永远在计划中
- 代码像考古现场
- 文档比代码还老

（如果发现自己天天在写"历史遗留代码"，那真得说点什么了）

4. 人才培养

注意这些现象：

- 新人培训全靠"自己悟"
- 技术分享变成了没几个人参与的"鸡汤分享"
- 晋升通道像迷宫一样
- 老员工都在准备简历

（如果发现新人入职三个月还在问"git怎么用"，那可能真要考虑培训体系的问题了）

5. 业务发展

关注这些信号：

- 产品方向像风向标
- 竞品都出3.0了我们还1.5
- 创新项目永远在PPT里
- 用户反馈像石沉大海

（如果发现竞品的产品经理都换了一茬了，我们的需求还在讨论，那就得说道说道了）

5.3.2 怎么说才不会翻车？

1. 做好功课

不要像背书一样照本宣科， 也不要像说相声一样东扯西扯。 准备点数据，举几个例子， 让领导知道你是真懂行，不是在瞎操心。

2. 态度要诚恳

不是来吐槽的， 不是来表现的， 是真心想把事情搞好， 是真的对团队有感情。

3. 建议要实在

不要说"我觉得应该全面提升一下"这种空话， 也不要说"咱们把技术栈全换了吧"这种大话。 说具体的，说能做的， 让领导听完能点头，而不是摇头。

5.3.3 最后的叮嘱

提意见不是找茬，是帮助团队发现问题、解决问题。

就像给代码提PR：不是为了显示自己多厉害，而是为了让代码变得更好。

5.3.4 评论

5.4 3.3 同事太优秀怎么办？

"隔壁小王又升职了..." "老李的项目又被表扬了..." "新来的应届生居然比我还厉害..."

每次看到这些消息， 心里就像被扎了一刀。 觉得自己怎么这么没用， 产生了严重的自我怀疑。

5.4.1 先说点实在的

1. 你不是一个人

- 每个人都会有这种感受
- 那些看起来很优秀的人也会
- 甚至他们可能也在羡慕你
- 这是再正常不过的情绪

2. 这其实是好事

- 说明你还在乎
- 说明你有上进心
- 说明你没有躺平
- 这种压力可以转化为动力

3. 换个角度想

- 有优秀的同事是你的福气
- 这是学习的好机会
- 这是提升的好环境
- 这比在一个混日子的团队强

5.4.2 如何调整心态？

1. 停止无意义的比较

记住：

- 你看到的是别人的highlight
- 而你只看到自己的behind the scenes
- 你不知道他们付出了什么
- 也不知道他们经历了什么

就像社交媒体一样： 别人都在发朋友圈晒成功， 谁会发自己熬夜改bug的照片？

2. 找到自己的节奏

每个人都有自己的路：

- 有人是短跑选手，早早就冲出去了
- 有人是马拉松选手，慢慢在追赶
- 有人是跨栏选手，一步一个台阶
- 重要的是找到适合自己的节奏

3. 转化压力为动力

与其羡慕别人：

- 不如向他们学习
- 不如请教他们经验
- 不如和他们合作
- 不如把他们当成自己的标杆

5.4.3 具体怎么做？

1. 给自己定个小目标

不要想着一下子赶超别人：

- 先学会他们的一个技能
- 先完成一个小项目
- 先提升一个方面
- 一步一步来

2. 主动靠近优秀的人

- 和他们一起吃午饭
- 请教技术问题
- 参与他们的项目
- 多交流多学习

3. 关注自己的进步

- 记录每天的收获

- 总结每周的进步
- 积累每月的成长
- 看见自己的变化

5.4.4 特别提醒

1. 永远不要因为这个伤害自己

工作只是生活的一部分，职场成就不代表人生价值，每个人都有自己的精彩，保持健康最重要。

2. 需要帮助时及时求助

和家人朋友倾诉，和信任的同事交流，寻求心理咨询，不要一个人扛着。

3. 给自己一些奖励

完成一个目标就犒劳自己，学会庆祝小进步，享受成长的过程，善待自己很重要。

5.4.5 最后的话

人生不是竞赛，是一场属于自己的旅程。

优秀的同事就像路标，指引你可以走多远，但不是要你变成他们，而是成为更好的自己。

（今天又看到老王升职了，不过这次我决定请他吃饭，好好请教一下经验...）

5.4.6 评论

5.5 3.4 同事是傻逼，我有厌蠢症，实在受不了了,想杀人

"这么简单的bug，他怎么改了三天还没改好？" "这个需求都讲了八百遍了，他怎么还不懂？" "我的天呐，这代码是人写的吗？"

如果你经常这样吐槽， 那么恭喜你， 你可能也是个"厌蠢症"患者，这个时候，就是考验你“向下兼容”的能力了。

5.5.1 先冷静一下

1. 每个人都是别人眼中的"傻子"

- 产品经理觉得程序员不懂需求
- 程序员觉得测试不懂开发
- 测试觉得运维不懂测试
- 运维觉得所有人都不懂运维（好像哪里不对...）

2. "聪明"是个相对概念

- 你觉得他笨，
- 可能别人觉得你更笨
- 就像你写的代码，

- 说不定被别人在群里吐槽呢（想想还有点小紧张）

3. 每个人都有自己的长处

- 他虽然代码写得慢，但bug少
- 他虽然理解需求慢，但做得稳
- 他虽然技术一般，但人缘好
- 他虽然不太聪明，但很努力（好像也没那么糟？）

5.5.2 如何应对？

1. 换个角度想

- 也许他是新手
- 也许他是转行的
- 也许他正在学习
- 也许他有特殊困难（谁还没个新手的时候？）

2. 调整心态

- 与其生气，不如帮助
- 与其抱怨，不如指导
- 与其嫌弃，不如包容
- 与其逃避，不如沟通（好像有点道理）

3. 具体行动

- 写详细的文档
- 画清晰的流程图
- 做耐心的解释
- 给出具体的例子（感觉自己变得更专业了）

5.5.3 特别提醒

1. 不要人身攻击

- 不要说"你怎么这么笨"
- 不要说"这都不懂"
- 不要说"谁招的你"
- 不要说"你是不是脑子有问题"（说出来显得自己更low）

2. 不要过度包揽

- 不是所有人都需要你救
- 不是所有事都要你管
- 给他成长的空间
- 给自己喘息的机会（累死自己可不值得）

3. 该放手时放手

如果真的无法沟通：

- 和领导反馈
- 调整分工方式
- 换个协作方式
- 实在不行就换组（及时止损很重要）

5.5.4 最后的话

其实， 所谓的"厌蠢症"， 往往是我们对他人的不理解， 和对自己的过度自信。

记住：每个人都在自己的节奏里成长，你对别人的包容，也是对自己的善待。

（诶，好像最近那个"笨蛋"同事进步不少...）

PS: 如果你真的想"杀"了他， 建议： 1. 用耐心"杀"死他的无知 2. 用帮助"杀"死他的迷茫 3. 用专业"杀"死他的混乱 4. 实在不行，"杀"死自己的偏见

5.5.5 评论

5.6 3.5 卷王太凶残，我要不要'反卷'？

"他早上6点就来公司了..." "他周末还在写代码..." "他又在群里发学习笔记了..." "他连厕所都带着笔记本..."

每当看到这些， 你是不是也想把键盘砸他脸上？（冷静，这样会把键盘弄脏的）

5.6.1 让我们先看看卷王的日常

观察卷王，不是为了模仿，而是为了找到更聪明的方式

1. 早卷

- 早上6点到公司
- 7点开始写代码
- 8点发朋友圈打卡
- 9点在群里分享心得（我8点才起床...）

2. 午卷

- 午饭带便当边吃边学
- 用纸巾擦手继续敲代码
- 休息时间刷技术视频
- 上厕所都带着Kindle（我中午只想睡觉...）

3. 晚卷

- 晚上9点还在写代码
- 10点开始写技术博客
- 11点还在群里讨论问题
- 12点发朋友圈总结今天（我7点就想回家...）

5.6.2 🔥 面对卷王，我们有什么选择？

1. 跟着一起卷？

优点：

- 能力提升快
- 可能升职加薪
- 简历好看

缺点：

- 头发掉得快
- 黑眼圈明显
- 可能猝死（好像有点不值得？）

2. 躺平不卷？

优点：

- 生活质量好
- 压力小
- 头发浓密

缺点：

- 可能被淘汰
- 可能没竞争力
- 可能加薪困难（这也不是办法啊）

3. 🧠 智慧反卷！

重点来了，这才是正确的打开方式：

A) 提升效率而不是时间

核心：用2小时完成别人4小时的工作 - 善用工具和自动化 - 专注高价值的事情（工作要智商，不要时长）

B) 建立个人壁垒

核心：成为领域专家，而不是万金油 - 找到自己的专业领域 - 深耕某个技术方向 - 打造不可替代性

C) 差异化竞争

核心：比成果，不比时间 - 不要和卷王比加班 - 要和他比成果 - 要和他比思路 - 要和他比效率

5.6.3 🎯 具体怎么操作？

1. 工作效率篇

关键词：`自动化` `工具` `流程` - 写自动化脚本 - 整理代码模板 - 建立知识库 - 优化工作流程

2. 专业能力篇

关键词：`深度` `积累` `方法论` - 聚焦核心技术 - 研究系统设计 - 积累实战经验 - 形成方法论

3. 生活质量篇

关键词：`平衡` `健康` `可持续` - 规律作息 - 适度运动 - 保持兴趣爱好 - 维系社交生活

5.6.4 特别提醒

1. 不要被表象迷惑

重要的是真实产出，而不是表面功夫 - 早到不代表效率高 - 加班不等于产出好 - 学习时间长不等于进步快

2. 保持自己的节奏

人生不是百米冲刺，是马拉松 - 每个人都有自己的路 - 不要被别人的焦虑影响 - 找到适合自己的方式

3. 该卷的时候还是要卷

关键是有度，有效率 - 关键项目冲刺 - 重要技术攻关 - 核心业务突破

5.6.5 最后的话

记住三点： 1. 卷王会一直存在 2. 但你不必成为卷王 3. 也不用怕卷王

最重要的是：

与其担心被卷死，不如提升自己的"含金量"。 用脑子工作，而不是用时间堆积。

（诶，好像最近那个卷王...头发掉得有点厉害...）

5.6.6 评论

5.7 3.6 我没有晋升不重要，但是他的晋升让我不爽

"听说小王要升P7了..." "啥？就他？开玩笑吧！" "那货代码写得还没我好呢！" "他进来的时候我都已经干了两年了！"

5.7.1 🤔 为什么这么不爽？

1. 常见的心理活动

其实我们都经历过这些想法 - "凭什么是他不是我？" - "我技术明明比他好" - "他就会讨好领导" - "这个世界太不公平了"（承认吧，你一定也这么想过）

2. 真实的心理根源

不是嫉妒他的晋升，而是质疑自己的价值 - 自我怀疑 - 成就感缺失 - 职业焦虑 - 价值认同感动摇

5.7.2 🔄 换个角度想想

1. 为什么是他？

技术只是晋升的一个维度 - 也许他更懂业务 - 也许他更会带团队 - 也许他解决问题的方式更高效 - 也许他确实付出了你没看到的努力

2. 我们经常忽略的事实

职场不只是比技术 - 技术好 ≠ 一定要晋升 - 晋升 ≠ 能力的全部证明 - 职级 ≠ 个人价值 - 当前位置 ≠ 未来成就

5.7.3 🛠 如何调整心态？

1. 正视自己的情绪

情绪是正常的，但不要被情绪控制 - 承认自己的不爽 - 允许自己难过一下 - 但不要停留在负面情绪里 - 把情绪转化为动力

2. 复盘自己的状态

关键词：`自省` `成长` `改变` - 最近的工作重心是什么 - 在哪些方面有提升 - 哪些地方需要加强 - 晋升需要的能力是否具备

3. 制定行动计划

与其羡慕别人，不如提升自己 - 明确自己的目标 - 找到差距在哪里 - 制定具体计划 - 付诸实际行动

5.7.4 具体怎么做？

1. 技术维度

- 深耕技术领域
- 解决关键问题
- 输出技术方案
- 分享技术心得

2. 业务维度

- 理解业务价值
- 主动承担责任
- 推动业务增长
- 建立业务影响力

3. 软实力维度

- 提升沟通能力
- 加强团队协作
- 培养领导力
- 锻炼项目管理能力

5.7.5 特别提醒

1. 不要做这些事

这些行为只会让情况变得更糟 - 在背后说同事坏话 - 消极怠工 - 散布负能量 - 过度对比抱怨

2. 该做的事

把注意力放在建设性的行动上 - 和领导做职业规划 - 向优秀同事学习 - 补齐自己的短板 - 积累更多实战经验

5.7.6 最后的话

人的常见错误归因方式 高估自己的努力，低估别人的努力。 高估自己的能力，低估自己的运气。 高估自己的能力，低估事情的难度。

与其纠结他人的晋升， 不如专注自己的成长。 也许当你不再执着于晋升的时候， 晋升反而会主动来找你。

（诶，好像最近那个升职的同事天天加班，黑眼圈都快掉到下巴了...）

PS：如果还是很不爽，建议： - 打开 IDE 写会代码 - 去健身房举举铁 - 约上三五好友吃顿饭 - 人生啊，远比职级更精彩

5.7.7 评论

5.8 社恐人混职场是真难啊

作为一个社恐程序员，每天去上班就像是去打仗。站会发言时紧张得像第一次表白，组内讨论时安静得像隔壁组的背景板，领导找谈话时心跳快得像在跑马拉松...但是，且慢，让我们聊聊这个让无数程序员头疼的问题。

5.8.1 🏢 社恐人的日常修罗场

你说社恐程序员的日常是什么样的？让我给你还原一下现场：

早上到公司，看到电梯里有同事，果断选择等下一班。进了办公室，发现最喜欢的角落位置被占了，整个人都不好了。站会的时候，轮到自己发言，明明昨天写了一个超牛的功能，但是嘴里只能蹦出来"我...我昨天改了个bug..."

中午吃饭，看到一群同事叽叽喳喳地在食堂热聊，自己抱着饭盒默默地躲到了楼梯间，一边吃一边刷手机。这时候最怕的就是有同事推门进来，那场面比谍战片还要紧张。

下午开组内讨论会，脑子里明明有一堆绝妙的想法，但就是说不出口。看着同事们激烈讨论，自己就像个透明人，默默地点头，仿佛在演默片。这种感觉，大概就和相亲时被父母逼着说"介绍一下自己"是一个级别的。

5.8.2 🤔 其实社恐也有社恐的优势

很多人只看到了社恐的劣势，其实社恐的特质也带来了一些独特的优势：

1. 专注力更强

- 不爱社交 = 更少的打扰
- 独处时间多 = 更深入的思考
- 不爱说话 = 更多的倾听和观察

2. 更细腻的思维

- 会提前思考各种可能性
- 对细节更敏感
- 能察觉他人容易忽略的问题

3. 更可靠的执行力

- 答应的事情一定会做到
- 不会随意承诺
- 做事更加严谨

4. 更强的同理心

- 容易理解他人的处境
- 不会轻易打断别人
- 更愿意帮助同样处境的人

5.8.3 🎯 社恐人的职场生存指南

1. 利用文字代替口头表达

- 提前准备会议发言稿

- 多用邮件和文档沟通
- 善用异步沟通工具
- 把想法写成文档再分享

2. 找到自己的"安全区"

- 选择合适的工位（比如靠窗或者角落）
- 戴上耳机创造私人空间
- 找到能让自己放松的休息区
- 和关系好的同事建立"联盟"

3. 把社恐转化为优势

- 利用独处时间提升技能
- 把细腻用在代码review上
- 用文档完整地表达想法
- 做好会前准备工作

4. 循序渐进地突破

- 从小范围会议开始练习发言
- 先和熟悉的同事多交流
- 参与线上讨论培养信心
- 给自己设定小目标

5.8.4 改变从小事做起

1. **站会发言的小技巧**
2. 提前写好要说的三点
3. 按照"昨天做了什么-今天要做什么-有什么问题"的顺序
4. 语速放慢，不要急着说完
5. **与同事交流的策略**
6. 从工作相关的话题开始
7. 多问问题，让对方多说
8. 记住一些通用的开场白，比如说"我最近在看一本书，里面提到...","我最近在学一门新技能，感觉很有意思","我最近在看一个电影，里面有..."
9. **应对社交压力的方法**
10. 给自己准备"逃生通道"
11. 事先了解活动流程
12. 找到"盟友"结伴参加

5.8.5 写在最后

社恐并不是缺陷，而是一种特质。在职场中，不是所有岗位都需要你像销售一样能说会道。很多时候，**社恐人的细心、专注和可靠**，反而是非常难得的优势。

重要的是找到适合自己的相处方式： - 不擅长即兴发言，就提前准备 - 不擅长面对面沟通，就多用文字 - 不擅长大场合，就从小范围开始

最后，请记住：**这个社会需要各种性格的人。**社恐不是问题，问题是你是否找到了合适的方式来展现自己的价值。

5.8.6 评论

6. 第四章 工作与家庭

6.1 第四章 工作与家庭

工作与家庭从来都不是非黑即白的选择题，而是一道需要用心经营的辩证题。

6.1.1 工作与家庭：相辅相成的关系

1. 打破对立思维

生活中，我们常常听到这样的说法：

- "工作和家庭只能二选一"
- "要么成功，要么幸福"
- "付出时间给工作，就是亏欠了家庭"

但实际上，这种非此即彼的思维方式本身就值得商榷。

2. 良性循环的真相

事实上，我们常常发现：

- 工作上靠谱的人，在家庭中往往也很靠谱
- 能把工作做好的人，经营家庭也往往有章法
- 职场上情商高的人，家庭关系通常也处理得不错

这不是巧合，而是能力的迁移和品质的统一。

6.1.2 为什么会相互促进？

1. 能力的迁移

- 工作中培养的沟通能力，同样适用于家庭交流
- 职场中锻炼的情绪管理，有助于家庭和谐
- 项目管理的方法，可以用于家庭生活的规划
- 解决问题的思维，能助力家庭难题的处理

2. 品质的统一

- 靠谱的人，在哪里都靠谱
- 负责任的态度，不会因场合而改变
- 善于合作的特质，在工作和家庭中都很重要
- 同理心和包容心，是双向的

6.1.3 如何实现双赢？

1. 建立正确认知

- 工作和家庭不是零和博弈
- 两者的关系是相互滋养而非相互消耗
- 平衡不是时间的均分，而是价值的整合

2. 寻找协同效应

- 把工作中学到的好方法用于家庭
- 将家庭的温暖带入工作的状态
- 让两个领域相互借鉴、共同提升

3. 智慧的分配之道

- 关注质量胜于数量
- 注重效率而非时长
- 把握关键时刻的存在
- 善用碎片时间的价值

6.1.4 本章要点

在接下来的内容中，我们将具体探讨： 1. 如何让伴侣成为工作的支持而非负担 2. 维系亲密关系的核心要素 3. 处理工作以及家庭中情绪的智慧方式

让我们一起探索如何让工作和家庭真正实现相辅相成、互相成就。

6.1.5 评论

6.2 4.1 好的伴侣可以帮你消化工作上的负面情绪

"今天不谈工作！" "回家就忘记公司的事！" "工作的事不要带回家！"

这些话听起来很有道理， 但真的合适吗？

6.2.1 🉑 为什么要和伴侣分享工作情绪？

1. 压抑不等于解决

情绪就像气球，憋得太久总会爆炸

- 刻意回避反而加重压力
- 负面情绪会在潜意识中积累
- 可能导致更严重的心理问题
- 影响工作和生活的质量
- 负面情绪需要适当宣泄
- 找到合适的倾诉对象
- 选择恰当的表达方式
- 保持情绪的健康流动
- 伴侣是最好的倾诉对象
- 最了解你的生活状态
- 最容易产生情感共鸣
- 最愿意倾听你的心声

2. 伴侣的独特价值

为什么伴侣比闺蜜/基友更适合倾诉？

- 更了解你的性格特点
- 知道你的处事方式
- 理解你的情绪变化
- 懂得如何安慰你
- 更深度的情感联结
- 共同的生活目标
- 相互的责任担当
- 长期的情感投入
- 更安全的倾诉环境
- 不用担心信息泄露
- 不用顾虑面子问题
- 可以完全地展示脆弱

6.2.2 怎样和伴侣分享工作情绪？

1. 选择合适的时机

时机比内容更重要

- 不要在这些时候分享：
- 刚到家的疲惫时刻
- 对方工作压力大时
- 正在享受美食时
- 准备休息入睡时
- 推荐的分享时机：
- 晚饭后的散步时光
- 周末早晨的咖啡时间
- 假期出游的轻松时刻
- 双方都有精力倾听的场合

2. 注意分享的方式

态度决定效果

- 语气的选择
- 平和而不激动
- 理性而不情绪化
- 讨论而不是抱怨
- 分享而不是发泄
- 内容的把控
- 重点突出
- 条理清晰
- 适可而止
- 互动的技巧
- 注意对方的反应
- 给对方表达的机会
- 接纳不同的观点
- 感谢对方的倾听

6.2.3 如何开启工作话题？

1. 选择合适的开场白

不恰当的方式：

- "今天那个产品经理又犯傻了！"
- "我们老板真是太过分了！"
- "我受够这个工作了！"
- "你知道今天发生了什么吗？气死我了！"

推荐的方式：

- "亲爱的，能听我说说今天遇到的事吗？"
- "最近工作上有点困惑，想听听你的想法。"
- "你在工作中遇到过类似的情况吗？"
- "能和你分享个工作中的小故事吗？"

2. 不同场景的沟通话术

遇到工作挫折时："今天项目上遇到点困难，能听我说说吗？我想理清楚自己的思路。"

面对职场压力时："最近工作压力有点大，能陪我散散步聊聊天吗？感觉和你说说话会好很多。"

经历职场冲突时："遇到个让我很困扰的情况，想听听你的看法，你在工作中遇到过类似的事吗？"

3. 营造轻松的分享氛围

- 晚饭后："今天遇到个有意思的事，我们边走边聊？"
- 周末早晨："一起喝杯咖啡，听我说个小故事？"
- 休息时光："能借用你几分钟，帮我参考个问题吗？"

6.2.4 如何建立长期的分享机制？

1. 固定的分享时刻

- 建立例行的交流习惯
- 创造专属的分享场景
- 保持适度的分享频率
- 注意互动的质量

2. 健康的互动模式

- 双向的信息流动
- 平等的对话关系
- 积极的反馈机制
- 持续的情感投入

3. 良性的成长循环

- 共同进步
- 互相支持
- 价值认同
- 感情升温

6.2.5 特别提醒

1. 保持边界感

分享不等于全盘托出

- 注意保护公司机密
- 不涉及具体业务数据
- 不透露商业机密
- 不谈论具体人名
- 重点是情绪而不是细节
- 把握分享的度
- 不要变成抱怨大会
- 不要沉浸在负能量中
- 不要期待对方解决所有问题
- 保持适度的分享量

2. 关注正向反馈

不只分享困难，也要分享快乐

- 分享工作中的成就
- 项目完成的喜悦
- 能力提升的欣慰
- 同事认可的自豪
- 工作进展的满意
- 讨论职业发展
- 分享职业规划
- 探讨发展方向
- 憧憬美好未来
- 共同规划人生

6.2.6 最后的话

记住：

好的伴侣关系， 不是形式上的两个人住在一起， 而是灵魂上的相互理解和支持。

工作是生活的重要部分， 而不是需要隐藏的秘密。 适度分享不仅能帮助消化负面情绪， 更能增进彼此的理解和信任。

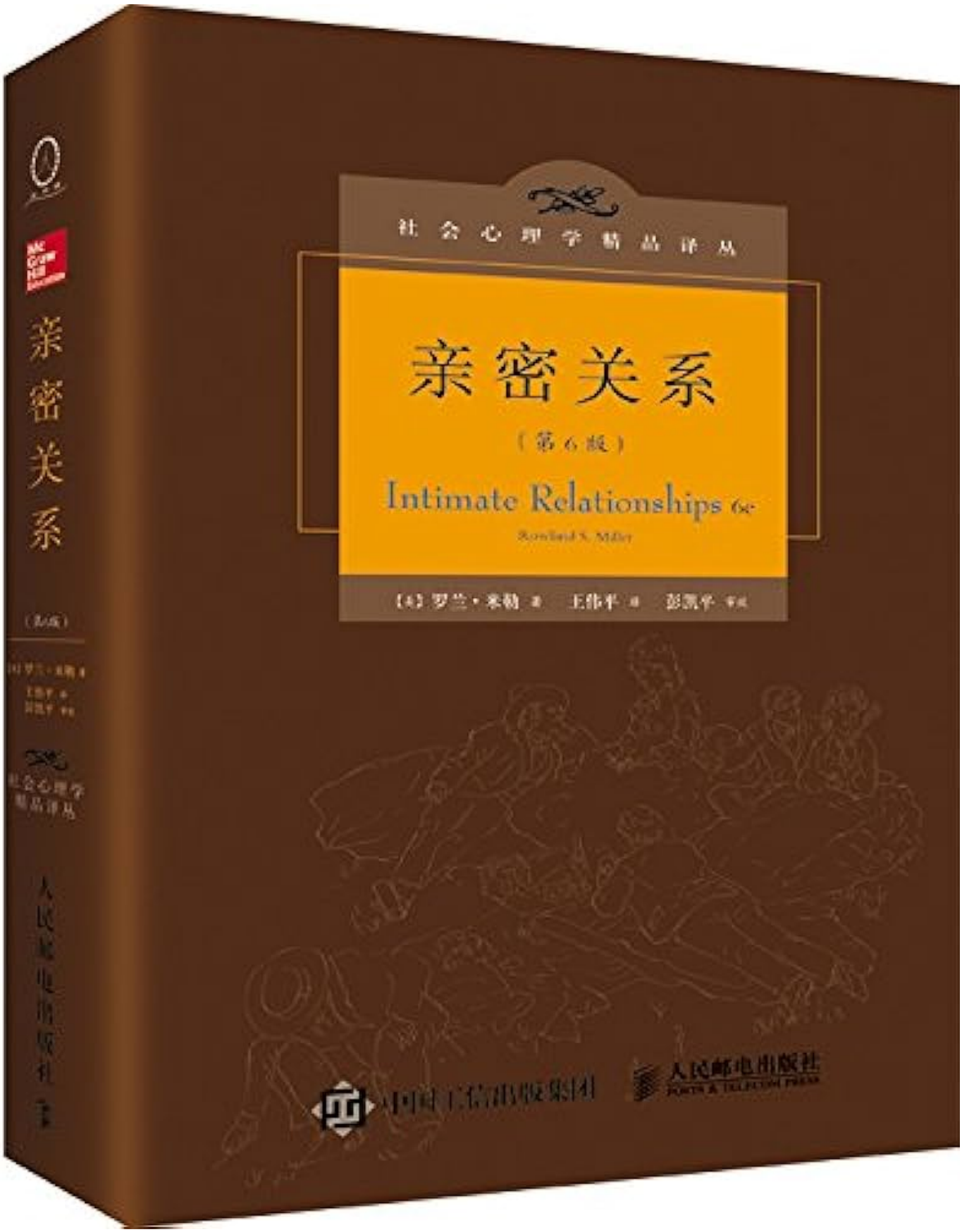
分享工作中的酸甜苦辣， 也是让感情升温的催化剂。 在互相理解和支持中， 两个人都能获得成长。

（今天又被产品经理改需求了，回家得跟对象好好吐槽...啊不是，是"理性探讨"一下～）

PS：给还没有伴侣的打工人： - 找个好伴侣真的很重要 - 不是因为需要人分担房贷 - 而是需要一个懂你的人 - 在你累的时候递上一杯温水 - 说一句："来，给我讲讲。"

6.2.7 评论

6.3 4.2 维系亲密关系最重要的一点：接纳胜过改变



"你为什么不能像他/她一样..." "你要是能改改这个习惯就好了..." "你什么时候才能懂我的心意..."

等一等， 维系一段亲密关系最重要的， 不是轰轰烈烈的爱， 不是无限量的付出， 而是——**接纳对方本来的样子。**

6.3.1 🉑 为什么是接纳？

1. 爱是如你所是，而非如我所愿

真正的爱不是把对方变成自己想要的样子

- 接纳意味着：
- 允许对方保持本真
- 尊重个体的差异
- 给予成长的空间
- 包容不完美
- 强求改变往往导致：
- 压抑真实的自我
- 产生抵触情绪
- 累积负面感受
- 关系渐行渐远

2. 稳定关系的密码

长久的感情建立在相互理解和包容之上

🈲 健康关系的特征：

- 很少的攻击
- 很少的对抗
- 很少的强人所难
- 很多的理解和接纳

🈵 不健康关系的特征：

- 频繁的批评指责
- 持续的争吵对抗
- 试图改变对方
- 过度的期待要求

6.3.2 🉑 如何做到真正的接纳？

1. 允许不完美

完美的关系不存在，但快乐的关系存在

- 接受对方的小缺点：
- 偶尔的健忘
- 某些小习惯
- 性格的特点
- 处事的方式
- 明白一个道理：
- 没有十全十美的人
- 差异造就独特
- 不完美才真实
- 包容创造和谐

2. 给予成长的空间

爱是成全，而不是束缚

🉐 应该这样：

- 支持对方的兴趣爱好
- 理解对方的职业选择
- 尊重对方的社交圈
- 给予独立思考的空间

🈲 避免这样：

- 过度干预对方生活
- 强制改变对方习惯
- 限制对方的社交
- 要求对方事事顺从

6.3.3 🈸 具体的行动指南

1. 日常相处中的接纳

从小事做起，在细节中体现

🈺 这样做：

- "你喜欢就好，我支持你"
- "这是你的选择，我理解"
- "需要帮忙随时说"
- "你有自己的想法很好"

🚫 避免说：

- "你怎么总是这样..."
- "你什么时候才能改改..."
- "你必须要听我的..."
- "要是你能像谁谁谁..."

2. 面对分歧时的态度

不同不代表错误，理解胜过改变

✔️ 建设性的回应：

- "我们想法不同，这很正常"
- "让我试着理解你的观点"
- "也许我们可以各自保留意见"
- "重要的是互相尊重"

6.3.4 ⚠️ 特别提醒

1. 接纳的边界

- 不是放纵伤害行为
- 不是默许原则问题
- 不是忽视重要分歧
- 不是牺牲自我价值

2. 需要警惕的情况

- 单方面的付出
- 过度的忍让
- 原则性的让步
- 价值观的冲突

6.3.5 🔚 最后的话

稳定的关系，不在于爱得多热烈，而在于伤得多少。

真正的爱，是让对方做自己，而不是变成你想要的样子。

每一次的包容，都是给感情加分。每一次的接纳，都是给关系加温。

（诶，好像该去陪对象打游戏了...虽然我不太懂，但看着也挺开心的~ ）

PS：给正在相处的你： - 爱不是改变对方 - 而是欣赏差异 - 包容不完美 - 给予自由 - 相信时间

6.3.6 🔚 评论

6.4 情绪稳定是个伪命题

"情绪稳定"这个词，听起来很美好，但真的存在吗？

6.4.1 为什么说情绪稳定是伪命题？

情绪波动是人类的天性，这是几千年进化的结果。喜怒哀乐，那是老祖宗刻在DNA里的代码，想改？不好意思，这个bug改不了，因为上帝这个产品经理说这是feature不是bug。

这些情绪反应都深深植根于我们的生理机制中，是大脑、内分泌系统和神经系统共同作用的结果。追求"零情绪波动"，就像我们追求头发永不掉落一样不切实际。相反，压抑情绪可能会带来更多问题：身体的压力激素升高，免疫系统受损，心理负担加重，甚至影响人际关系。

6.4.2 我们真正该关注什么？

情绪调节能力

重点不是没有情绪，而是如何管理情绪。健康的情绪管理包括觉察情绪的存在，接纳情绪的产生，理解情绪的原因，以及找到合适的调节方式。很多人误以为"情绪稳定"就是不能有负面情绪，用"你要情绪稳定"来要求自己，这恰恰是不健康的。

建立情绪韧性

情绪韧性就像程序员的发际线，虽然每天都在后退，但只要还在战斗就永远有希望。就像橡皮筋一样，可以被压缩，但终能回弹。这包括我们承受压力的能力，从低谷恢复的速度，适应变化的弹性，以及自我调节的方法。这也是传说中的反脆弱。

6.4.3 如何与情绪和谐相处？

首先要接纳情绪的存在。情绪本身没有好坏，它们都是生命给我们的信号。当我们感到愤怒时，也许是边界被侵犯；当我们感到焦虑时，可能是对未来的担忧；当我们感到沮丧时，或许是某些需求没有被满足。重点是要找到情绪背后的原因并解决，而不是压抑情绪。

其次是提升情绪管理能力。这不是一蹴而就的事情，需要我们持续观察和练习。可以尝试一些具体的方法：深呼吸和冥想可以帮助我们平静下来，运动和户外活动能让身心放松，写日记能帮助我们梳理思绪，与信任的人倾诉也是很好的方式。

6.4.4 给自己合理的期待

不要期待自己永远平静如水，毕竟我们是程序员不是神仙（虽然有时候都会被当成万能的）。重要的是在波动时能够觉察，在起伏时懂得调节，在低谷时知道自救，在恢复时总结经验。

当然，如果发现情绪持续低落，波动极其剧烈，影响到日常生活，或者难以自行调节时，及时寻求专业帮助也是明智的选择。

6.4.5 最后的话

情绪稳定不是没有波动，而是在波动中保持掌控；不是压抑真实感受，而是学会与情绪共处。就像大海，表面有波澜，但深处依然沉稳。

（写完这篇文章，突然觉得，接纳情绪的存在，本身就是一种智慧...）

6.4.6 评论

6.5 伴侣老是抱怨，我该怎么办

每个人都希望下班回到家能得到温暖和理解，但现实往往是：刚进门就听到一连串的抱怨。工作已经够累了，回家还要面对这些，该怎么办？

6.5.1 为什么会有这么多抱怨？

在谈如何应对之前，我们先要理解：为什么会有这么多抱怨？

1. 抱怨的本质

抱怨背后往往隐藏着更深层的诉求： - "今天又加班" = 我很想你/我觉得被忽视了 - "工资太低" = 我担心我们的未来 - "你都不陪我" = 我需要更多的关注和陪伴 - "你总是玩手机" = 我想要更多的互动

2. 抱怨的积累

很多抱怨不是突然产生的，而是长期积累的结果： - 小事情没有及时沟通 - 情绪没有得到及时疏导 - 期望没有被及时调整 - 问题没有被真正解决

6.5.2 换个角度看抱怨

其实，抱怨也不全是坏事：

1. **抱怨是一种信任**
2. 敢对你抱怨，说明还在乎这段关系
3. 愿意表达不满，总比憋在心里强
4. 抱怨是在寻求改变，而不是放弃
5. **抱怨是沟通的开始**
6. 至少ta愿意和你说话
7. 这是一个了解对方需求的机会
8. 可以借此深入交流

6.5.3 如何应对伴侣的抱怨

1. 倾听的艺术

- 不要急着打断或反驳
- 用身体语言表示你在认真听
- 适时地复述对方的话，确保理解准确
- 不要边听边玩手机

2. 情绪先行，解决其次

- 先认可对方的感受："我理解你会这么想"
- 表达共情："换做是我可能也会很烦"
- 等情绪平复后再讨论解决方案
- 不要一上来就讲道理

3. 寻找根源

抱怨往往只是表象，要找到真正的原因： - 是工作压力太大？ - 是对未来感到焦虑？ - 是感情需要被关注？ - 是生活缺乏期待？

4. 共同制定计划

不要停留在抱怨本身，而是要： - 一起讨论可行的解决方案 - 设定清晰的改进目标 - 约定检查的时间点 - 互相监督和鼓励

6.5.4 预防胜于应对

1. 建立日常沟通机制

- 固定时间交流今天的经历
- 每周进行一次深度对话
- 重要决定一起商量
- 及时分享喜怒哀乐

2. 创造美好时刻

- 一起规划周末活动
- 制造小惊喜
- 保持适度的仪式感
- 创造共同的回忆

3. 维护个人边界

- 保留各自的独处时间
- 尊重对方的兴趣爱好
- 给彼此成长的空间
- 不要过度依赖或控制

6.5.5 写在最后

没有人喜欢一直抱怨，如果伴侣经常抱怨，与其抱怨"ta总是抱怨"，不如： - 主动去了解背后的原因 - 真诚地面对问题 - 一起寻找解决方案

最重要的是：**不要把抱怨当作攻击，而要把它看作改善关系的机会。**

因为在亲密关系中，重要的不是谁对谁错，而是我们能否一起把事情变得更好。就像调试代码一样，找到bug不是结束，修复它才是目的。

6.5.6 评论

6.6 婆媳关系真是男人的噩梦啊

作为程序员，我们擅长解决各种复杂的技术问题。但面对婆媳关系这个千古难题时，却常常感觉比解 Bug 还要头疼。今天，让我们用程序员的思维来看看这个问题。

6.6.1 为什么婆媳关系这么难处理？

1. 系统冲突

就像两个不同的操作系统要共享同一个资源： - 都想要儿子/老公的关注和时间 - 都认为自己对他最好 - 都觉得自己的方式是对的 - 都在争夺家庭"管理员权限"

2. 角色转换的不适应

就像系统升级带来的兼容性问题： - 婆婆从"独占"儿子到"共享"的转变 - 媳妇从"独立个体"到"家庭成员"的转变 - 儿子从"儿子"到"丈夫"的角色切换 - 家庭权力结构的重新分配

3. 代际差异

这就像新旧版本的冲突： - 生活方式的差异 - 教育理念的差异 - 价值观的差异 - 沟通方式的差异

6.6.2 核心原则：以小家庭为重

在处理婆媳关系时，要记住一个核心原则：**以核心家庭（你和妻子）为重。**就像系统架构一样，必须有明确的优先级：

1. 为什么要以小家庭为重？

- 这是你未来生活的主要系统
- 妻子是你最亲密的伙伴
- 家庭和睦是孩子成长的基础
- 只有小家稳定，才能更好地照顾大家

2. 如何平衡和妈妈的关系？

- 保持经常性的关心和联系
- 在重要决定上征询意见
- 适时表达感谢和孝心
- 让她感受到被尊重和重视

6.6.3 具体的处理策略

1. 对待妻子（重中之重）

- 在公开场合要明确支持妻子
- 家庭重大决定以妻子意见为主
- 及时和妻子沟通，不让她觉得被忽视
- 在妻子受委屈时要及时表态

2. 对待妈妈（柔性处理）

- 私下多和妈妈沟通
- 让妈妈感受到你的关心没有减少
- 适度满足妈妈的合理需求
- 委婉表达而不是直接对抗

3. 处理冲突时

- 先安抚妻子情绪，获得她的理解和支持
- 再私下和妈妈沟通，解释现代家庭的相处之道
- 寻找双方都能接受的解决方案
- 建立长期的沟通机制

6.6.4 日常相处的智慧

1. 与妻子的默契

- 达成对待父母的共识
- 互相理解和支持
- 有矛盾先私下沟通
- 在孩子教育等重大问题上统一战线

2. 与妈妈的沟通

- 多表达感谢和理解
- 让她感受到被需要
- 创造和孙辈的互动机会
- 适度保持距离，但不疏远

3. 建立长期机制

- 固定的探访频率
- 清晰的界限感
- 独立的生活空间
- 合理的期望管理

6.6.5 写在最后

处理婆媳关系，最重要的是要认清一个事实：**妻子是你的人生伴侣，是和你一起构建未来的人。**就像系统架构一样，主系统的稳定性是首要考虑的因素。

同时，我们也要善待父母这个"上一代系统"，毕竟他们付出了一生的心血，而且他们也在努力适应角色的转变，同样需要被理解和尊重。

最终的目标是：让核心系统（小家庭）稳定运行，同时保持与"上一代系统"（父母）的良好兼容。这确实比处理分布式系统还要复杂，但这是每个男人都必须面对和解决的课题。

修复婆媳关系的最好方式，就是先搞定自己的伴侣，再平衡与父母的关系，摒着宁可委屈自己，也要大家开心的原则。毕竟，一个幸福的小家，才是让父母真正安心的最好方式。

哎，虽然笔者小嘴巴巴的说了这么多，但是真到了事上，笔者也是满脑袋包啊。

婆媳关系，真的是男人的噩梦啊！！！

6.6.6 评论

7. 第五章 只工作不上班

7.1 第五章 只工作不上班

在这个快速变化的时代，工作的定义正在被重新书写。 AI 的发展让"只工作不上班"从理想照进了现实。

7.1.1 工作方式的革新

曾经，我们习惯了朝九晚五的上班模式，仿佛这就是工作的全部定义。但现在，技术的进步给了我们更多的选择 —— **我们可以只工作，不上班。**

这不是在说我们可以不工作，而是工作的方式可以**更灵活，更智能。**

7.1.2 AI：你的得力助手

想想看，当我们有了 AI 这样高效的助手，很多过去需要团队才能完成的工作，现在一个人也能搞定：

AI 工具箱：

现如今，写文案你可以用 ChatGPT，做设计你可以用 Midjourney，写代码你可以用 Copilot/Cursor。

7.1.3 一人公司的崛起

一人公司正在成为一种新的趋势。AI 正在成为我们最得力的助手： - 不会喊累 - 不用休假 - 不会和你抢年终奖（开个玩笑）

这让我们能够以更低的成本，更高的效率来工作。

7.1.4 新的挑战

当然，选择只工作不上班，并不意味着生活会变得更轻松。相反，这需要：

⚠ **更高的要求：** - 更强的自律能力 - 更强的学习能力 - 更强的洞察业务需求能力 - 更好的时间管理 - 更大的责任担当

就像从单纯前端开发转向了全栈开发，还是一个身兼运营，产品，技术的全栈开发，**自由度更大，责任也更大。**

7.1.5 本章探讨内容

在这一章中，我们会深入探讨：

核心内容： 1. 程序员的副业探索 2. 对工作 Gap 的看法 3. 如何找到属于自己的 pathless path

因为笔者目前也是在这条路上探索，所以这一章节会持续更新自己的探索和思考，欢迎关注。

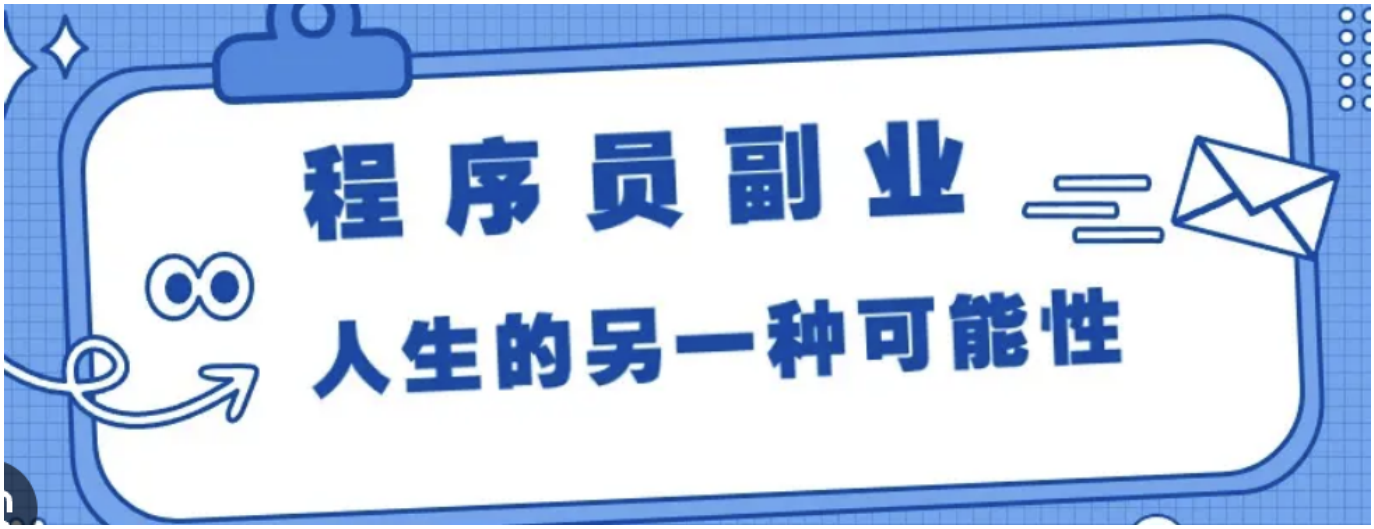
7.1.6 重要提醒

笔者想强调： 选择只工作不上班，不是为了逃避，不是为了躺平，而是为了以更适合自己的方式建立跟这个社会的链接。

也许有一天，当我们回顾2025年，会发现这正是一个时代的转折点。

7.1.7 评论

7.2 5.1 我所接触到的程序员副业



最近经常被问到："程序员适合做什么副业？" 与其泛泛而谈，不如聊聊我这些年真实接触到的案例。

7.2.1 为什么程序员特别适合搞副业？

说实话，程序员可能是最适合搞副业的群体之一。想想看：我们有编程技能，有互联网思维，还有一台电脑就能开始干活。这简直就是副业的最佳选手啊！（虽然我们经常加班，但这是另一个故事了...）

不过有一点特别重要：**副业要在不影响主业的前提下进行。**就像写代码一样，先实现一个最小可用版本（MVP），验证可行性后再逐步迭代。千万别一上来就想着全职做副业，步子迈大了，那样容易扯着蛋。

本文主要聚焦于一些笔者见过的主流的副业，还有一些野路子因为没有普适性就不分享了。

7.2.2 我所见过的程序员副业

1. 接外包项目

这可能是最直接的变现方式了。我有个同事就经常在一些平台接单：

- Upwork（国外项目，报酬不错，但需要英语能力，而且近两年国内用户越来越难接到单子）
- 程序员客栈（国内项目，竞争较大）
- 码市（适合个人接单）

优势：

- 直接利用现有技能
- 收入可观
- 积累项目经验

需要注意：

- 时间估算要留余地
- 合同和收款要谨慎
- 不要接超出能力的单子（我见过有人因为高估自己能力，结果一个项目拖了半年，赔了时间还没赚到钱...）

2. 做技术自媒体

这条路我也在尝试。关键是要找到好的选题，有"网感"。

成功案例：

- 一位做Java的朋友，专注写"面试八股文"系列，月入过万
- 另一位做算法的，把LeetCode题目讲解得特别生动，粉丝量很大

重点关注：

- 选题要有差异化
- 保持更新频率
- 注意知识产权问题
- 培养自己的写作风格

3. 技术咨询服务

这是个很有意思的方向。我发现很多程序员在闲鱼上做：

- 简历修改
- 面试辅导
- 职业规划咨询
- 技术选型建议

其实很多人不缺技术，缺的是前前人的经验和建议，尤其是碰到一些职场的重要选择时，这些经验往往能起到关键作用，这也是这个市场需求存在的原因。 做这个要有很强的沟通能力，同时需要积累口碑，有足够的耐心，才能慢慢积累客户。

4. 独立开发者

这条路有点难，但很有意思。我见过两种类型：

- 做小程序：门槛低，变现快，但竞争激烈
- 开发独立App：投入大，风险高，但一旦成功，收入可观

身边有朋友开发了一个时间管理小程序，虽然用户不多，但每个月能有个几千块的被动收入。关键是找到一个刚需但还没被巨头垄断的细分领域。

做独立开发成功的案例就太多了，比如开发 胃之书/极简时钟 的作者，已经全职做独立开发者了。

5. AI工具出海

这是最近特别火的方向。身边几个朋友在做，我也在学习做相关的尝试：

- 用ChatGPT API开发各种工具
- 针对海外市场的AI应用
- 垂直领域的AI助手

不过要注意：这个领域变化太快，要做好持续学习和迭代的准备。

6. 运营付费社群

这也是近年来程序员副业的一个新趋势。做社群不是简单地建个群收钱，而是要能持续提供价值。

我有个朋友就在运营一个"前端进阶"社群：

- 每周组织算法打卡
- 定期做简历优化
- 分享面试经验
- 组织线上 workshop

他的经验是：**"真正的社群运营，是把自己的经验变成别人的捷径。不过最难的是持续输出，很多社群三分钟热度后就凉了。"**

要关注的重点：

- 找准目标群体
- 设计清晰的收费模式
- 保证内容的持续输出
- 建立良性互动机制
- 控制社群规模和氛围

7. 出版技术书籍

这是一条周期较长但很有成就感的路。写书不是为了发财，更重要的事把自己的知识体系整理清楚。

出书的几种方式：

- 和传统出版社合作
- 在电子书平台发布
- 自出版（比如 GitBook）

经验分享：

- 写作周期通常在6-12个月
- 版税收入其实不高
- 但对个人品牌很有帮助
- 能倒逼自己深入学习

有趣的是，我一个写书的朋友告诉我：**"写技术书最大的收获不是稿费，而是写作过程中发现自己的很多知识盲点。而且写完书后，面试时的底气都足了不少。"**

不过他也提醒说：**"千万别以为技术好就能写好书，技术写作和技术本身是两种完全不同的能力。"**

7.2.3 一些实用建议

1. **从小目标开始** 别一上来就想着做个"中国版XX"，先解决一个小问题，做出一个最小原型。
2. **注意时间管理** 我的经验是：先用周末时间试水，看看是否真的感兴趣，再考虑加大投入。
3. **选择适合自己的方向** 不要盲目跟风。就像我一个同事说的：**"与其做一个自己不懂的热门项目，不如做一个自己擅长的冷门项目。"**
4. **做好长期准备** 副业跟写代码一样，不会一蹴而就，需要持续迭代优化。

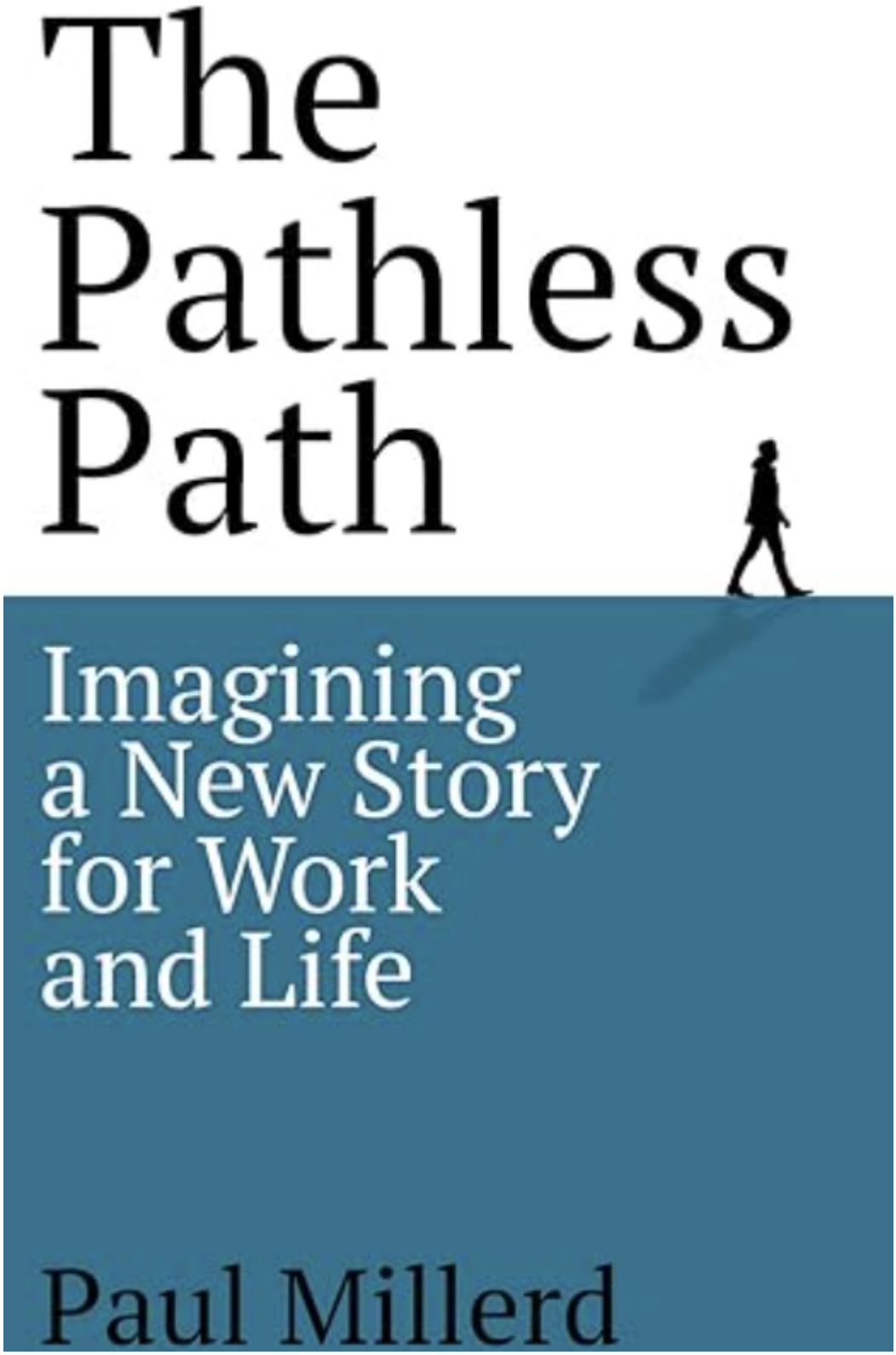
7.2.4 写在最后

本文列举了一些副业的可能性，但这些都只是一个方向，一个引子。真正的价值不在于看完这篇文章，而在于你愿意迈出第一步，把手弄脏，去真正的尝试与探索。

副业的意义，不仅仅在于赚钱。当你没有公司的光环加持，没有现成的资源可用，一切都要从零开始时，你才会真正面对市场，接触社会。这个过程中的每一次尝试，每一次碰壁，都是真实的成长。

7.2.5 评论

7.3 5.2 Gap 无罪：寻找属于你自己的 Pathless Path



笔者按部就班的工作了8年，终于，在35岁前夕被裁的那一天，我问自己："这真的是我想要的生活吗？"

关于作者的更多感触，可以看我之前发的视频：《35岁，我被裁了》

7.3.1 为什么我们需要 Gap？

说实话，在中国谈 gap，总会被贴上"不求上进"的标签。但其实，gap 不是放弃，而是为了更好地前进。就像代码里需要断点调试一样，人生偶尔也需要暂停下来，看看自己到底在哪里。

工作久了，很多人会发现一个有趣的现象：明明工资在涨，技能在增长，但内心的倦怠感却越来越强。这不是你的问题，而是工作异化的结果。我们把自己变成了一台高效的编码机器，忘记了自己还是个需要思考和成长的人。

7.3.2 Default Path vs Pathless Path

默认路径（Default Path）

这是我们最熟悉的路： - 好好学习 - 找份稳定工作 - 努力晋升 - 买房结婚 - 养老退休

这条路没有错，但问题是：这真的是你想要的吗？还是仅仅因为"大家都这样"？

无径之径（Pathless Path）

这是一条需要自己探索的路。没有现成的地图，没有标准答案，但可能更适合你。

我有个朋友，工作第六年时决定 gap 了三个月。他说：**"刚开始特别慌，感觉自己浪费时间。但慢慢地，我开始思考什么是真正重要的。"**

后来他没有回到原来的大厂，而是选择了一家小公司，工资低了不少，但他能主导产品方向，贡献感更强。**"虽然收入少了，但我终于找回了写代码的乐趣。"**

7.3.3 为什么要 Gap？

1. 找回自我

当我们停下来，才能听到内心真实的声音。很多时候，我们忙得连自己想要什么都忘了。

2. 跳出思维定式

就像重构代码一样，有时候需要完全推翻重来，才能写出更好的实现。人生也是如此。

3. 充电升级

这不是在浪费时间，而是在给自己一次版本升级的机会。（程序员永远喜欢用版本号类比人生）

7.3.4 如何优雅地 Gap？

1. **做好准备金储备** 至少要有6个月的生活费。毕竟重启人生不是请假，需要足够的缓冲时间。
2. **设定探索方向** gap 不是无所事事，而是有目的的探索。可以：
3. 学习新技能
4. 环球旅行
5. 尝试创业
6. 写写技术博客
7. 研究自己真正感兴趣的领域
8. **保持开放心态** 也许最后你会回到原来的轨道，也许会找到全新的方向，这都是正常的。重要的是这段探索的过程。

7.3.5 关于焦虑

说实话，gap 期间最大的敌人是焦虑。看着前同事们继续在职场上升，而自己似乎在"浪费时间"，这种感觉不好受。

但要记住：**人生不是竞赛，是探索。**

就像一个资深程序员说的（其实就是臭不要脸的我）：**"与其在一条注定会后悔的路上一直走下去，不如勇敢地按下暂停键，去寻找真正属于自己的路。"**

7.3.6 写在最后

Gap 不是逃避，而是为了更好地前进。 就像代码重构一样，看似是在推倒重来，实际是在为更好的架构打基础。

记住，选择 gap 的人不是失败者，而是有勇气探索自己人生的勇者。

7.3.7 评论

7.4 其实，没工作，也能活的很好

"为什么我们总觉得失去工作就是天塌了？" 也许是时候重新思考：工作对我们来说，究竟意味着什么？

7.4.1 工作中的异化

很多程序员，包括我在内，都经历过这样的困惑：为什么在公司工作总觉得别扭？

就像一个圆形的积木，硬要被塞进方形的孔里：

- 公司要求你 9 点到岗，哪怕你是夜猫子
- 产品让你写一堆不认可的代码
- 领导要你参加毫无意义的会议
- KPI 迫使你做不想做的项目

这就是马克思说的"异化"：**我们在工作中逐渐失去了自我，变成了公司机器上的一个零件。**

7.4.2 困在自己的思维里

我们总觉得：

- 没有工作就没有收入
- 没有收入就无法生存
- 无法生存就会活不下去

但是，真的是这样吗？

我们是不是把"工作"这个概念想得太狭隘了？ 我们是不是被困在了自己给自己设定的牢笼里？

7.4.3 打开思维的牢笼

其实，工作的本质是：**用你的能力为这个世界创造价值，然后获得回报。**

而实现这个目标的方式有很多种：

- 🧑‍💻 接单做自由职业者
- 🏪 经营自己的小生意
- 📱 开发独立产品
- 🎓 做知识付费
- 🎨 发展创意副业

7.4.4 真实的例子

我认识一位程序员，他辞职后开始做独立开发：

"刚开始确实害怕，但后来发现，当你不用再为公司的KPI发愁，不用强迫自己融入不喜欢的环境，反而能做出更好的产品。"

现在他的小产品每月有稳定收入，虽然比不上大厂工资，但他说：**"生活质量反而更好了，因为我终于能按照自己的节奏来生活，这种对生活的掌控感，是以前上班时无法体会到的。"**

7.4.5 如何开始改变

1. **认识自我** 你真正热爱什么？你的核心能力是什么？你想要什么样的生活？
2. **降低生活成本** 真正必需的开支是多少？哪些是可以优化的？如何建立应急储备？
3. **尝试新的可能** 先从副业开始，积累一些经验，建立自己的节奏

7.4.6 重要的提醒

⚠ **在开始之前：**

确保有足够的储蓄，做好心理准备，给自己足够的缓冲期，保持开放和学习的心态

7.4.7 写在最后

生活不是非黑即白， 工作也不是非公司不可。

重要的是找到属于自己的方式， 用自己的方式和这个世界建立连接。

既然我们是个圆形，为什么不去这个社会中主动找寻需要圆形的地方呢？ 也许，当我们不再把自己局限在"必须找到工作"的思维里， 会发现人生其实有更多的可能性。

7.4.8 📝 评论

8. 结语：成为一个自洽的程序员

最近和一个程序员老友聊天，他说："写了十年代码，我终于明白，最难的不是技术，而是如何保持一个自洽的心态，才能走得更好更远。"

这句话让我想起了很多。在这个行业里，我们见过太多故事：有人在大厂累倒，有人在创业路上迷失，有人在技术与管理的转折点徘徊，也有人在35岁的门槛前焦虑。似乎每个程序员都在寻找一个答案：如何在这条路上走得更好？

其实答案可能一直都在，只是我们看得太远，反而忽略了最近的风景。

编程教会我们逻辑，但生活不只是逻辑。当我们把目光从代码中移开，会发现生活还有更多的可能性。就像一个优秀的系统，不仅要有强大的功能，更要有坚实的基础架构。而一个自洽的程序员，也应该在技术和生活之间找到平衡。

记得刚入行时，总觉得写代码就是全部，拼命地学习新框架，追逐最新技术，生怕落后一步就会被淘汰。但慢慢地，我发现那些在这个行业走得最远的人，往往不是技术最强的，而是最懂得平衡的。他们既能钻研技术，也能享受生活；既能在工作中全力以赴，也能为自己保留成长的空间。

在我踏入社会的过程中，我遇到了很多程序员，听到了很多故事。有人选择了管理岗，有人坚持在技术路上深耕，有人去创业了，也有人选择了"只工作不上班"的生活方式。他们的选择不同，但都找到了属于自己的节奏。这让我明白，所谓"自洽"，不是要符合某个标准，而是要找到适合自己的方式。

就像代码里没有银弹，人生也没有标准答案。重要的是，我们能否在这个快速变化的时代，既保持技术的专业性，又不失生活的温度；既能跟上时代的节奏，又能守住内心的平静。

也许，这就是一个自洽的程序员：

- 他们明白技术很重要，但技术不是全部
- 他们追求卓越，但不盲目追逐
- 他们在代码中寻找答案，也在生活中寻找意义

写到这里，我想起了那个经典的"Hello, World!"。每个程序员都是从这里开始的，但不应该在这里结束。愿我们都能在编程的世界里找到快乐，在生活的长河中寻得自我，最终成为一个真正自洽的程序员。

8.1 关于作者

🈶 2013-2016 毕业于北大软件与微电子学院

🈶 2016-2024 编码八年，混迹于国企，大厂，外企

🈶 2025-至今 独立开发/一人公司/只工作不上班（其实就是社会闲散人员）

8.2 关注公众号：辣条加辣

更多职场、个人成长、程序员副业的探索和思考

—— 写于2025年1月12日

8.3 🈯 评论