2.3 学习型心态

所谓学习型心态，指的是：有主动学习的意识，时刻以学习的眼光和心态来看待发生在自己身上的事情。

举个例子，作为开发人员，你做完了一个功能，展示给领导看，结果他说你做出来的东西根本不是他们想要的，简直一无是处。这个时候，你会怎么办？

一个选择是怼回去，大家吵一架，然后不欢而散，甚至因此和领导不睦，最终觉得领导太没水平，此地无法发展，离职。

另一个选择是坐下来分析：

- 哪里不对？该怎样调整才符合需求？
- 这个“不对”是在何时被引入的？
- 以后如何避免开发过程中的走样？
- 沟通机制是否存在问题？如何改进？

后面这个选择就是积极主动的选择，是面向解决问题的，是学习型心态的开发者会采取的方式。他们会考虑自己怎样去改进、怎样能达成目标、通过这件事自己能收获什么。

作为软件开发人员，其实有很多提升自己的机会和方式，而最终你能不能提升，除了要看你是否有内在动力，还要看你是否拥有学习型心态。

2.4 技术精进之道

做了各种铺垫，我们终于可以开始介绍精进的方法了。

1. 对标管理法

在专业领域成长的一般模型如图 2-2 所示。

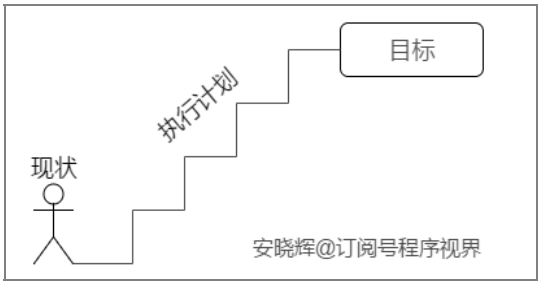

图 2-2

模型中有三个要素：

- 现状
- 目标
- 执行计划

每个人都可以评估自己的现状，自己在做什么、用什么技术、技术达到了什么程度、拿多少薪水、是什么职级、是否被领导认可、与人协作是否顺畅……有很多维度，静下心来思考一下，在纸上列一列，就能得出自己当下的状态。

而目标则很可能随着旧目标的达成而消失，或者随着日复一日的编码、Debug、交付而褪色，或者随着每个月的薪水蒸发掉。一旦我们失去目标，就会陷入迷茫，被动工作，进而慢慢失去竞争力。

所以，要想日有寸进，必须要在日常的开发工作中找到努力的目标。这非常关键——很多人就是因为没有目标而放任自己随波逐流、被动工作，最终变得庸常而被组织淘汰。

因此我们引入原本用于企业的对标管理法，帮助自己在日常工作中找到贴合自己的目标。一旦我们找到目标，对比现状，就可以找到差距和前进方向，有了方向，就可以制定计划，稳步前进，获得提升。

图 2-3 是实践对标管理法指导个人成长的基本过程。

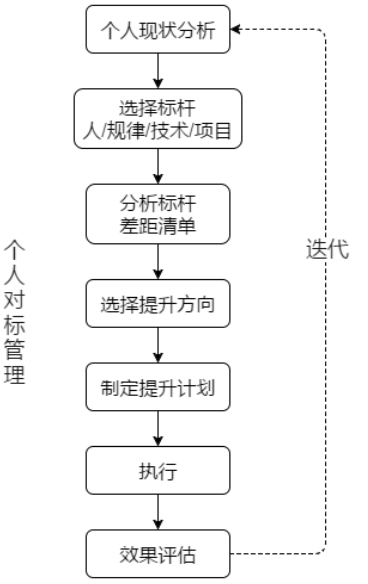

图 2-3

以下解释来自百度百科：

对标管理，由美国施乐公司于 1979 年首创，均将其视为现代西方发达国家企业管理活动中支持企业不断改进和获得竞争优势的最重要的管理方式之一，西方管理学界将对标管理与企业再造、战略联盟一起并称为 20 世纪 90 年代三大管理方法。

对标管理是指企业以行业内或行业外的一流企业作为标杆，从各个方面与标杆企业进行比较、分析、判断，通过学习他人的先进经验来改善自身的不足，从而赶超标杆企业，不断追求优秀业绩的良性循环过程。

所谓“对标”就是对比标杆找差距。推行对标管理，就是要把企业的目光紧紧盯住业界最好水平，明确自身与业界最佳的差距，从而指明工作的总体方向。

在针对个人运用对标管理法时，可以从 4 个方面来寻找标杆：

- 优秀的人
- 一般性规律

- 技术本身的知识层次
- 项目指标

接下来我们就从这 4 个方面展开，看看怎么寻找我们的目标。

2．从优秀者身上找目标

我们身边一定有人在某方面做得比自己好，比如：

- 张三的设计文档写得结构合理、条理清晰；
- 李四的 UML 图表画得准确；
- 王五对 ES6 标准掌握得好；
- 赵六擅长做代码管理；
- 钱七的架构设计能力超群，对产品的架构如数家珍；
- 毛八每天上班前都会列出要完成的三件事，每天下班时都会总结；
- 胡九学习新技术特别快，总是在项目组中担任技术预研角色。

……

别人做得好的方面，都可能是我们努力的方向。我们要用善于发现的眼睛，找到身边人的突出之处。

在向优秀者对标时，下面的问题清单可以帮助我们有序、系统地分析标杆：

- 他在什么事情上做得突出？是怎么做到的？
- 他有哪些知识、技能是我不具备的？
- 他有哪些提升效率的工具？
- 他有哪些好的工作习惯？

举个例子。

袁大每天都能准时下班，工作还完成得很好。你对这一点很感兴趣，就暗中观察他是如何做事的，发现他过一段时间就会翻看一下纸质笔记本，或者用笔在

本子上记录点什么，还有，每天下班的时候，他都会在本子上写点东西。

于是你跟他聊天，发现他每天下班前都会在笔记本上记录今天完成了什么、遇到了什么问题、明天做什么。还了解到他每天都会早到半个小时左右，利用这段时间规划一天的工作。

后来你明白了，袁大培养了一个“早规划晚回顾”的工作习惯，通过这个习惯，保证每天都有几件重要的事可做，每天都有目标、有节奏，这样就可以不慌不忙地工作。

于是你就会思考：袁大的习惯是否可以成为我的习惯？

这个时候，你就找到了一个提升的方向：培养每日完成三件事的习惯。一旦你养成这个习惯，习惯的力量就会帮助你集腋成裘，完成从量变到质变的过程。

再举个例子。

你发现组里的袁二，排查 Bug 特别厉害，像一休哥一样，点点头沉思一下，就可以说出问题所在。即便是别人的代码引入的 Bug，他也可以很快找到原因——只需要翻翻代码，和对方聊几句。

为什么袁二这么牛？

你向他请教，发现他做到了以下几点：

- 对业务特别熟悉，非常清楚某个业务到底是什么，用户在软件上怎么使用这个业务。
- 对业务逻辑和代码的映射关系特别熟。
- 爱看代码，所有人的代码他都看。

好了，正好你总是被 Bug 困扰，往往一个 Bug 能让你愁烦一个星期，这下是不是有努力方向了？

3．一般性规律

所谓一般性规律，指的是那些通用的、可以指导我们在什么时候做什么事情的规律。

举个例子，舒伯的生涯发展阶段理论就是一般性规律，男大当婚女大当嫁也是一般性规律。

对于开发者来讲，要关注专业能力成长的一般性规律，即技术成长三阶段。如图 2-4 所示。

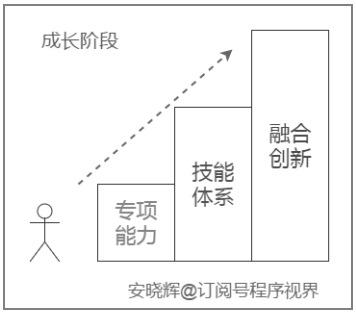

图 2-4

在技术领域内的成长，基本上都会经历三个阶段：

- 第 1 阶段，专项能力的提升，这是初级阶段，你为了做事情，必须先具备某些基础能力，比如你要学会 Python、Visual Studio、Vue.js、TensorFlow 或者 MyBatis 等。
- 第 2 阶段，技能体系的构建，这是中级阶段，你拥有了一组技能，围绕某个方向构建了自己的知识图谱，能够用自己的方式来解决问题。比如在 C++这个方向上，你用 C++、Qt、OpenGL、Libevent、FFmpeg、WebRTC 等组成了自己的知识图谱，可以胜任流媒体方面的产品开发。
- 第 3 阶段，融合创新，这是高手阶段，你具有了丰富的实践经验，具备了 T 型知识结构，形成了自己的思维框架和解决问题的框架，能够融合不同领域的知识，组合各种资源，创造性地解决各种问题。此时你跳出了具体的技术束缚，站在了更高的层面，用底层认知和思维来指导你的工作。

对开发者来讲，若拥有一年左右经验，多数人处在第一个阶段——专项能力提升的阶段，熟悉某种编程语言，可以完成别人安排的一个小模块的开发。

若拥有三年及以上的经验，就应该进入第二个阶段了。当你在某个技术方向上构建了技能体系后，就可以完成相对复杂的工作，可以独立做一些事情，甚至可以辅导初级开发者来完成工作了。这个时候，你往往已经是团队里富有生产力的成员了。

若有五年以上开发经验，应该进入融合创新阶段，能够独当一面，可以独立完成特定项目的评估、设计、技术方案选择等事情。此时你往往是团队里的技术领袖或者技术管理者，具有比较大的影响力。

假如一个开发者干上八年十年，还到不了第 3 个阶段，可能就需要考虑通过其他方式来提升自己的竞争力，保住自己在团队中的位置。

这个模型更适合应用开发人员，对于做基础研究（比如音频算法、图像处理算法等）的开发者，在第 3 阶段，可能在他所在领域内钻得更深，成为专家。

我们了解了技术成长的三个阶段，就可以结合自己的工作情况，判断自己当下处于哪个阶段，该做什么事情了。

比如你做了 2 年 PHP 开发，可能你处于从第 1 阶段向第 2 阶段转型的过程中，此时提升的方向，就可以考虑和 PHP 相关的技术栈，比如了解 HTTP 服务器是如何和 PHP 整合在一起的，再比如了解数据库、操作系统，这样你就可能会定下掌握 LAMP（Linux/Apache/MySQL/PHP）或者 LNMP（Linux/Nginx/MySQL/PHP）技术栈的目标。

4．技术本身的知识层次

一门编程语言、一个技术框架，其本身的知识层次也会有深浅，在学习时，也存在先后顺序和一般性规律。从这个角度讲，技术本身的深浅层次也可以用于个人对标管理。

一般来讲，学习一门技术时，有三个阶段：

- 第 1 阶段，基础开发，了解 API，基于 API 开发应用。
- 第 2 阶段，熟悉内核及原理，主要是了解框架的设计原理，阅读源码，洞悉内在机理。

- 第 3 阶段，优化框架，主要是针对框架已有功能的不足进行完善、优化，或者使用框架提供的机制扩展框架功能，或者对框架进行定制，让它适合特定情境。

我比较熟悉 Qt，Qt 这个应用开发框架，三阶段的划分可能是图 2-5 所示这样的。

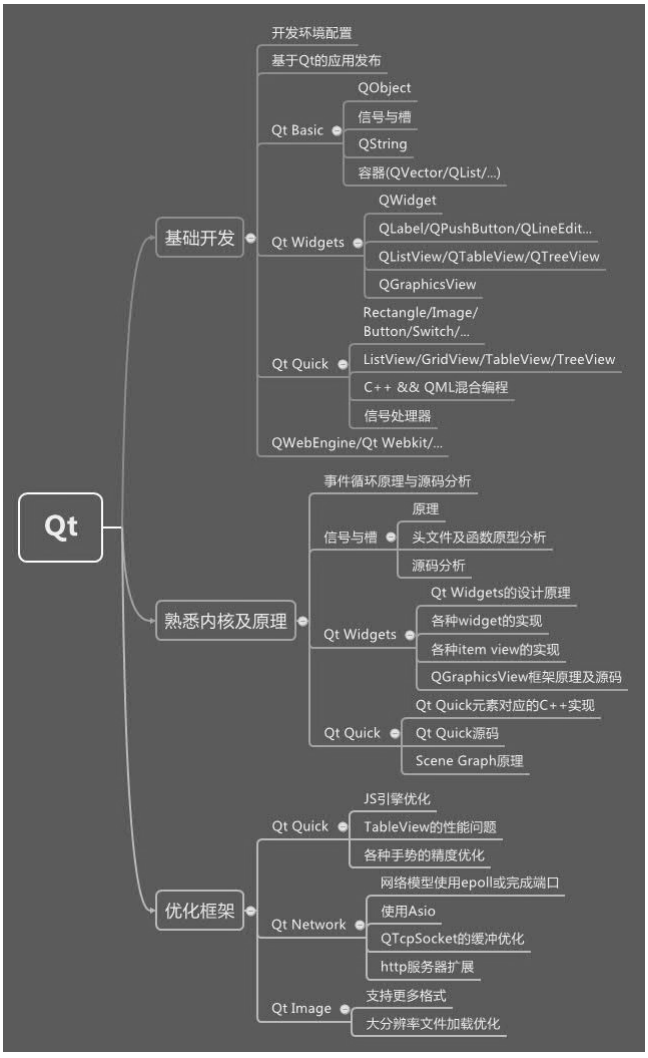

图 2-5

多数技术框架，通过分析，都可以划分出类似上面的知识层次和学习阶段。

以此作为对标的标杆，就可以弄明白每个阶段应该达到什么程度，还可以定位自己处在哪个阶段，当前阶段的任务有没有完成，接下来该学什么。

5．项目指标

开发者的工作往往是由一个又一个项目串起来的，每个项目都会有预期结果，都会界定怎么样才算是完成，然后会有一系列的指标用于衡量项目做得怎么样，比如 Bug 率、延期时间、并发用户数、持续运行时间、单元测试覆盖率、安全性等。

我们在做项目时，就可以用这些指标来要求自己，这样每个项目都有目标，都可以制定一些策略，帮助自己来实现这些目标。

很多开发者其实不太关心交付时间、Bug 率、冒烟测试通过率、并发用户支持、内存占用、CPU 占用、电池消耗等问题，往往是做完了，能跑，觉得就可以了。

以这样的态度来应付开发任务，其实损失最大的是自己，因为你白白失去了锻炼和提升的大好机会。

如果以项目指标来要求自己，把项目指标分解到开发工作中，并且在开发过程中贯彻执行，收获一定比被动完成任务多得多。

举个简单的例子，如果你用 Java 开发一个电商类的 Android App，内存占用就应当是你关心的一个指标，否则你的应用就会经常出现 OOM 错误，严重损害用户体验，导致用户大量卸载，最终影响产品的市场。

如果你把内存占用作为重点考虑的指标，一定会考虑如何使用图片预缩放、重用、解码格式、缓存等策略来优化内存占用，你甚至会自己设计一个图片缓存池或者特殊的 ListView 来专门处理用户快速浏览商品时巨大的内存消耗。

你有了这样的考虑，做出来的 APP 肯定比你从未考虑过内存占用问题而做出来的稳定得多。

6．拿来即用的自我提升方法

前面介绍了如何从 4 个方面寻找目标提升自己，只要你遵循那些策略，付出一些努力，就可以找到适合自己的提升策略。

下面给出一些经过验证切实有效的提升策略，你可以拿来直接用。

- **尝试用同一技术的不同模块或 API 来实现**，能让你更了解所用技术。
- **看看你正在用的技术，想想你处在三个层次的哪一层**，找到继续提升的空间，去学习、实践。持续这么做，能让你从泛泛的了解、基础的使用，进阶到熟悉、精通。
- **了解和当前所用技术相关的技术**，可以拓展你的知识图谱。
- **尝试用不同的技术来实现**，能加深对问题的理解，也能淬炼新的技术。
- **看看别人用的技术点、技术栈**，尝试了解，能拓宽你的视野。
- **看看同一项目内他人的设计和代码**，有助于理解整个项目。
- **尝试新的设计**，能加深对问题的理解，更能锻炼自己的架构和设计能力。
- **看看整个项目的需求、设计文档**。不要局限于自己负责的模块，这样可以提升全局观和系统观。
- **迭代式重构老代码**，迭代式重构可以解决时间不够用的问题。
- **阅读优秀源码**，看到好的，思考好在哪里，琢磨自己怎么做到，这样你就会日有寸进，终至千里。
- **参与开源项目**，参与开源项目比阅读开源代码的要求高得多，你要能够理解已有的代码，找到你可以做贡献的地方（issue、feature 等），你的代码要符合该项目的规范，还会被项目 owner 或其他成员 Review，这些都是非常大的挑战，能让你快速成长。
- **写作技术博客**，有利于写作、逻辑思考、讲授、设计等能力的提升，也有利于系统化你的知识。最好的学习方式是输出。
- **讲给别人听**，锻炼讲授、演讲、沟通、归纳总结、逻辑思考等能力，对知识的内化与系统化也很有帮助。当你能够把一个知识点讲到别人也能听明白时，你就是真明白了。