**引言**

古人云：“活到老，学到老。”互联网算是最辛苦的行业之一，“加班”对工程师来说已是“家常便饭”，同时互联网技术又日新月异，很多工程师都疲于应付，叫苦不堪。以至于长期以来流传一个很广的误解：35岁是程序员工作的终点。

如何在繁忙的工作中做好技术积累，构建个人核心竞争力，相信是很多工程师同行都在思考的问题。本文是我自己的一些总结，试图从三个方面来解答：

* 第一部分阐述了一些学习的原则。任何时候，遵循一些经过检验的原则，都是影响效率的重要因素，正确的方法是成功的秘诀。
* 提升工作和学习效率的另一个重要因素是释惑和良好心态。第二部分分析了我在工作中碰到和看到的一些典型困惑。
* 成为优秀的架构师是大部分初中级工程师的阶段性目标。第三部分剖析架构师的能力模型，让大家对目标所需能力有一个比较清晰的认知。

**如何学习**

在繁忙的工作中，持之以恒、不断学习和进步是一件艰巨的任务，需要坚强的毅力和坚定的决心。如果方法不得当，更是事倍功半。幸好我们的古人和现在哲人已经总结了很多优秀的学习方法论，这里汇总了一些重要原则。遵循这些方法必会对大家的工作学习大有裨益。

**贵在坚持**

有报道指出，过去几十年的知识量超过之前人类几千年的知识量总和。而计算机领域绝对是当代知识更新最快的领域之一，因此，工程师必须要接受这样一个现实，现在所掌握的深厚知识体系很快就会被淘汰。要想在计算机领域持续做优秀架构师，就必须不停的学习，掌握最新技术。总之，学不可以已。

所谓“冰冻三尺，非一日之寒，水滴石穿，非一日之功”，通往架构师的道路漫长而又艰巨，轻易放弃，则所有付出瞬间付之东流。要想成为优秀的架构师，贵在坚持！

虽然知识更新很快，但是基础理论的变化却非常缓慢。这就是“道”和“象”关系，纵是世间万象，道却万变不离其宗。对于那些非常基础的理论知识，我们需要经常复习，也就是“学而时习之”。

**重视实践**

古人云：“纸上得来终觉浅，绝知此事要躬行。” 学习领域有所谓721模型：个人的成长70%来自于岗位实践，20%来自向他人学习，10%来自于培训。虽然这种理论存在争议，但对于工程师们来说，按照实践、学习和培训的方式进行重要性排序，大致是不错的。所以重视实践，在实践中成长是最重要的学习原则。

人类的认知有两种：感性认知和理性认知。这两种认知互相不可替代性。实践很大程度来自于感性学习，看书更像是理性学习。以学开汽车做例子，很难想象什么人能够仅仅通过学习书本知识就会开汽车。

书本知识主要是传道——讲述抽象原型，而对其具体应用场景的讲述往往含糊其辞，对抽象原型之间的关系也是浅尝辄止。采用同样精确的语言去描述应用场景和关联关系将会失去重点，让人摸不着头脑。所以，仅仅通过看书来获得成长就像是用一条腿走路。

重视实践，充分运用感性认知潜能，在项目中磨炼自己，才是正确的学习之道。在实践中，在某些关键动作上刻意练习，也会取得事半功倍的效果。

**重视交流**

牛顿说：“如果说我看得比别人远一些，那是因为我站在巨人的肩膀上。”我们需要从别人身上学习。从老师、领导、同事、下属甚至对手身上学习，是快速成长的重要手段。

向老师和领导学习已经是人们生活习惯的一部分了。但是从同事甚至对手那里学习也很重要，因为这些人和我们自身更相似。所以要多多观察，取其所长，弃其所短。对于团队的小兄弟和下属，也要“不耻下问”。

此外，在项目中积极参与具体方案讨论也非常重要。参与者先验感知了相关背景，并且讨论的观点和建议也是综合了发言者多种知识和技能。所以，讨论让参与者能够非常全面，立体地理解书本知识。同时，和高手讨论，他们的观点就会像修剪机剪树枝一样，快速的剪掉自己知识领域里面的疑惑点。

**重视总结和输出**

工程师在实践中会掌握大量细节，但是，即使掌握了所有细节，却没有深刻的总结和思考，也会陷入到“学而不思则罔”的境地。成长的“量变”来自于对细节的逐渐深入地把控，而真正的“质变”来自于对“道”的更深层次的理解。

将经验输出，接受别人的检验是高层次的总结。这种输出不仅帮助了别人，对自身更是大有裨益。总结的方式有很多，包括组织分享，撰写技术文章等等。当然“日三省吾身”也是不错的总结方式。总之，多多总结，多多分享，善莫大焉！

解答别人的问题也是个人成长的重要手段。有时候，某个问题自己本来不太懂，但是在给别人讲解的时候却豁然开朗。所以，“诲人不倦”利人惠己。

**重视规划**

凡事预则立，不预则废。对于漫长的学习生涯而言，好的计划是成功的一半。

**长期规划**

长期规划的实施需要毅力和决心，但是做正确的长期规划还需要高瞻远瞩的眼界、超级敏感的神经和中大奖的运气。对于大部分人来说，长期规划定主要是“定方向”。但遵循如下原则能够减少犯方向性错误的概率：

* 远离日暮西山的行业。
* 做自己感兴趣的事情。
* 做有积累的事情。
* 一边走一边看，切勿一条道走到黑。

**短期规划**

良好的短期规划应该在生活、成长、绩效和晋升之间取得平衡。大部分公司都会制定一个考核周期——少则一个月，多则一年。所以不妨以考核周期作为短期学习规划周期。本质上，规划是一个多目标优化问题，它有一系列的理论方案，这里不一一细说。基于相关理论，我给出一个简单易行的方案：

* 确定目标优先级。比如：成长、生活、绩效。
* 确定每个目标的下限。从优化理论的角度来看，这被称为约束。比如绩效必须在一般以上，之前已经规划好的旅行不能更改，必须读完《Effective Java》等等。
* 优先为下限目标分配足够的资源。比如，事先规划好的旅行需要10天，这10天就必须预算出去。
* 按照各主目标的顺序依次分配资源。比如，最终分配给学习的时间是10天。
* 在给定的学习预算下，制定学习目标，要激进。然后给出执行方案。比如，学习目标是掌握基本的统计学知识，并成为Java专家。具体方案为：完成《Effective Java》、《Java Performance》、《Design Pattern》、《Head First Statistics》四本书的阅读。
* 对规划中的各学习任务按目标优先级进行排序，并最先启动优先级最高的任务。比如，最高优先级是掌握统计理论，那么就要先看《Head First Statistics》。

对于该方案，要注意以下几点：

* 最低目标必须能够轻松达成的目标，否则，从优化理论的角度来讲，该命题无解。比如，类似“半年内完成晋级两次、绩效全部S、从菜鸟成为Java专家”就不太合适作为最低目标。总之，要区分理想和梦想。
* 主要目标规划必须具备一定的挑战性，需要规划出不可能完成的目标。过度规划本质上是一种贪婪算法，目的是目标价值最大化。因为一切皆有变数，如果其他目标能够提前完成，就不妨利用这些时间去完成更多的学习目标。总之，前途必须光明，道路必须坎坷。
* 各目标之间不一定共享资源，规划不一定互有冲突。

此外，短期规划还可以从如下几个方面进行优化：

* 学习计划最好能结合工作计划，理论联系实际结合，快速学以致用。比如，本季度规划去做一些数据分析工作，那么不妨把学习目标设置为学习统计知识。
* 要灵活对待规划的目标和具体执行步骤，需要避免“郑人买履”式的笑话。面临新的挑战和变化，规划需要不断地调整。

**那些令人纠结的困惑**

人生是一场马拉松，在漫长的征途中，难免有很多困惑。困惑就像枷锁，使我们步履蹒跚，困惑就像死锁，让我们停滞不前。

接下来我将总结自己在工作中碰到和看到的一些典型困惑。这些困惑或者长期困扰作者本人，或者困扰我身边的同事和朋友。当这些困惑被释然之后，大家都感觉如重获释，为下一阶段的征程提供满满的正能量。人生就像一场旅途，不必在乎目的地，在乎的，应该是沿途的风景，以及看风景的心情。良好的心态是技术之旅最好的伴侣。期望通过这个解惑之旅，让大家拥有一个愉快的心情去感受漫长的学习旅途。

**学无止境吗**

必须要承认一个残酷的现实：人的生命是有限的，知识却是无限的。用有限的生命去学习无限的知识是不可能完成的任务。一想到此，有些工程师不免产生一些悲观情绪。如果方法得当并且足够勤奋，悲伤大可不必。

虽然，人类的整体知识体系一直在扩张。但是就很多重要的工程细分领域，基础理论并不高深。计算机的很多重要领域，工程师有能力在有限时间内抓住核心要害。

比如，密码学被认为是门非常高深的学科，但是一大类密码技术的基础是数论中一个非常简单的理论——素因数分解：给出两个素数，很容易算出它们的积，然而反过来给定两个素数的积，分解的计算量却非常惊人。

“一致性”算得上是计算机领域里面最经典的难题，它是所有分布式系统的基础，从多核多CPU到多线程，从跨机器到跨机房，无所不在，几乎所有的计算机从业人员都在解决这个问题，但是Paxos给出了一个很优雅的解决方案。

权限管理是很多工程师的噩梦，但如果你能搞定“Attribute Based Access Control(ABAC)”和“Role-Based Access Control(RBAC)”，也能达到相当高度。

另外，技术学习是一场对抗赛，虽然学无止境，超越大部分对手就是一种胜利。所以，以正确的学习方式，长时间投入就会形成核心竞争力。

**没有绝对高明的技术，只有真正的高手**

致力于在技术上有所成就的工程师，都梦想有朝一日成为技术高手。但技术高手的标准却存在很大的争议。这是一个有着悠久历史的误解：以某种技术的掌握作为技术高手的评判标准。我经常碰到这样一些情景：因为掌握了某些技术，比如Spring、Kafka、Elasticsearch等，一些工程师就自封为高手。有些工程师非常仰慕别的团队，原因竟是那个团队使用了某种技术。

这种误解的产生有几个原因：首先，技多不压身，技术自然是掌握的越多越好，掌握很多技术的人自然不是菜鸟。其次，在互联网时代来临之前，信息获取是非常昂贵的事情。这就导致一项技能的掌握可以给个人甚至整个公司带来优势地位。互联网时代，各种框架的出现以及开源的普及快速淘汰或者降低了很多技能的价值，同时降低了很多技术的学习门槛。所以，在当前，掌握某项技能知识只能是一个短期目标。怀揣某些技能就沾沾自喜的人需要记住：骄傲使人退步。

所谓麻雀虽小，五脏俱全。如果让你来做造物主，设计麻雀和设计大象的复杂度并没有明显区别。一个看起来很小的业务需求，为了达到极致，所需要的技术和能力是非常综合和高深的。真正的高手不是拿着所掌握的技术去卡客户需求，而是倾听客户的需求，给出精益求精的方案。完成客户的需求是一场擂台赛，真正的高手，是会见招拆招的。

**不做项目就无法成长吗**

在项目中学习是最快的成长方式之一，很多工程师非常享受这个过程。但是一年到头都做项目，你可能是在一家外包公司。对于一个做产品的公司，如果年头到年尾都在做项目，要不然就是在初步创业阶段，要不然就是做了大量失败的项目，总之不算是特别理想的状态。正常情况，在项目之间都会有一些非项目时间。在这段时间，有些同学会产生迷茫，成长很慢。

项目真的是越多越好吗？答案显然是否定的。重复的项目不会给工程师们带来新的成长。不停的做项目，从而缺乏学习新知识的时间，会导致“做而不学则殆”。真正让工程师出类拔萃的是项目的深度，而不是不停地做项目。所以，在项目之间的空档期，工程师们应该珍惜难得的喘息之机，深入思考，把项目做深，做精。

如何提高项目的深度呢？一般而言，任何项目都有一个目标，当项目完成后，目标就算基本达成了。但是，客户真的满意了吗？系统的可用性、可靠性、可扩展性、可维护性已经做到极致了吗？这几个问题的答案永远是否定的。所以，任何一个有价值的项目，都可以一直深挖。深挖项目，深度思考还可以锻炼工程师的创造力。期望不停地做项目的人，就像一个致力于训练更多千里马的人是发明不出汽车的。锻炼创造力也不是一蹴而就的事情，需要长时间地思考。总之，工程师们应该总是觉得时间不够用，毕竟时间是最宝贵的资源。

**职责真的很小吗**

很多时候，一个工程师所负责系统的数量和团队规模与其“江湖地位”正相关。但是，江湖地位与技术成长没有必然关联。提升技术能力的关键是项目深度以及客户的挑剔程度。项目越多，在单个项目中投入的时间就越少，容易陷入肤浅。特别需要避免的是“ 在其位不谋其政”的情况。团队越大，在管理方面需要投入的精力就越多。在管理技巧不成熟，技术眼界不够高的前提强行负责大团队，可能会导致个人疲于应付，团队毫无建树。最终“ 一将无能，累死三军”，效果可能适得其反。

从技术发展的角度来说，技术管理者应该关注自己所能把控的活跃项目的数量，并致力于提高活跃项目的影响力和技术深度。团队人数要与个人管理能力、规划能力和需求把控能力相适应。一份工作让多个人来干，每个人的成长都受限。每个人都做简单重复的工作，对技术成长没有任何好处。团队管理和项目管理需要循序渐进，忌“拔苗助长”。

**一定要当老大吗**

有一些工程师的人生理想是做团队里的技术老大，这当然是一个值得称赞的理想。可是，如果整个团队技术能力一般，发展潜力一般，而你是技术最强者，这与其说是幸运，不如说是悲哀。这种场景被称之为“武大郎开店”。 团队里的技术顶尖高手不是不能做，但为了能够持续成长，需要满足如下几个条件：

* 首先你得是行业里面的顶尖专家了——实在很难找到比你更强的人了！
* 其次，你经常需要承担对你自己的能力有挑战的任务，但同时你拥有一批聪明能干的队友。虽然你的技术能力最高，但是在你不熟悉的领域，你的队友能够进行探索并扩展整个团队的知识。
* 最后，你必须要敏而好学，不耻下问。

否则，加入更强的技术团队或许是更好的选择，最少不是什么值得骄傲的事情。

**平台化的传说**

平台化算得上是“高大上”的代名词了，很多工程师挤破头就为了和“平台化”沾点边。然而和其他业务需求相比，平台化需求并没有本质上的区别。无论是平台化需求还是普通业务需求，它的价值都来自于客户价值。不同点如下：

* 很多平台化需求的客户来自于技术团队，普通需求的客户来自于业务方。
* 产品经理不同。普通业务需求来自于产品经理，平台化需求的产品经理可能就是工程师自己。长期被产品经理“压迫”的工程师们，在平台化上终于找到“翻身农奴把歌唱”的感觉。
* 很多平台化的关注点是接入能力和可扩展性，而普通业务的关注点更多。

归根结底，平台化就是一种普通需求。在实施平台化之前，一定要避免下面两个误区：

* 平台化绝对不是诸如“统一”、“全面”之类形容词的堆砌。是否需要平台化，应该综合考虑：客户数量，为客户解决的问题，以及客户价值是否值得平台化的投入。
* 平台化不是你做平台，让客户来服务你。一些平台化设计者的规划设计里面，把大量的平台接入工作、脏活累活交给了客户，然后自己专注于所谓“最高大上”的功能。恰恰相反，平台化应该是客户什么都不做，所有的脏活累活都由平台方来做。本质上讲，平台化的价值来自于技术深度。真正体现技术深度的恰恰是设计者能够很轻松的把所有的脏活累活搞定。

所以平台化的最佳实践是：投入最少的资源，解决最多的问题。平台解决一切，客户坐享其成。

**搞基础技术就一定很牛吗**

经常听到同学们表达对基础技术部同学的敬仰之情，而对搞业务技术的同学表现出很轻视，认为存储、消息队列、服务治理框架（比如美团点评内部使用的OCTO）、Hadoop等才能被称为真正的技术。事实并非如此，更基础的并不一定更高深。

比如下面这个流传很久的段子：越高级的语言就越没有技术含量。但真是这样吗，就拿Java和C来说，这是完全不同的两种语言，所需要的技能完全不同。C或许跟操作系统更加接近一点，和CPU、内存打交道的机会更多一点。但是为了用好Java，程序员在面向对象、设计模式、框架技术方面必须要非常精通。Java工程师转到C方向确实不容易，但作者也见过很多转到Java语言的C工程师水土不服。

基础技术和业务应用技术必然会有不同的关注点，没有高低之分。之所以产生这种误解，有两个原因：

* 基础技术相对成熟，有比较完整的体系，这给人一个高大上的感觉。业务应用技术相对来说，由于每个团队使用的不一样，所以成熟度参差不齐，影响力没有那么大。
* 基础技术的门槛相对来说高一点，考虑到影响面，对可靠性、可用性等有比较高的最低要求。但是门槛高不代表技术含量高，另外成熟技术相对来说在创新方面会受到很大的约束。但是最先进的技术都来自活跃的创新。

对比下来，业务技术和基础技术各有千秋。但真正的高手关注的是解决问题，所有的技术都是技能而已。

**可行性调研的那些坑**

工作中开展可行性调研时有发生。做可行性调研要避免如下情况：

* 把可行性调研做成不可行性调研。这真的非常糟糕。不可行性的结论往往是：因为这样或者那样的原因，所以不可行。
* 避免“老鼠给猫挂铃铛”式的高风险可行性方案。“天下大事必作于细”，可行性调研一定要细致入微，避免粗枝大叶。
* 避免调研时间过长。如果发现调研进展进入到指数级复杂度，也就是每前进一步需要之前两倍的时间投入，就应该果断的停止调研。

可行性调研的结论应该是收益与成本的折衷，格式一般如下：

* 首先明确预期的结果，并按照高中低收益进行分级。
* 阐述达成每种预期结果需要采取的措施和方案。
* 给出实施各方案需要付出的成本。

**工程师天生不善沟通吗**

实际工作中，沟通所导致的问题层出不穷。工程师有不少是比较内向的，总是被贴上“不善沟通”的标签。实际上，沟通能力是工程师最重要的能力之一，良好的沟通是高效工作学习的基础，也是通过学习可以掌握的。下面我按工程师的语言说说沟通方面的经验。

第一类常见的问题是沟通的可靠性。从可靠性的角度来讲，沟通分为TCP模式和UDP模式。TCP模式的形象表述是：我知道你知道。UDP模式的形象表述是：希望你知道。TCP模式当然比较可靠，不过成本比较高，UDP模式成本低，但是不可靠。在沟通可靠性方面，常见错误有如下两种：

* 经常听到的这样的争论。一方说：“我已经告诉他了”，另一方说：“我不知道这个事情呀”。把UDP模式被当作TCP模式来使用容易产生扯皮。
* 过度沟通。有些同学对沟通的可靠性产生了过度焦虑，不断的重复讨论已有结论问题。把TCP模式当成UDP来使用，效率会比较低。

第二类沟通问题是时效性问题。从时效性讲，沟通分为：同步模式和异步模式。同步沟通形象地说就是：你现在给我听好了。异步沟通的形象表述是：记得给我做好了。在沟通时效性方面，有如下两种常见错误：

* 已经出现线上事故，紧急万分。大家你一言，我一语，感觉事故可能和某几个人有关，但是也不能完全确定，所以没有通知相关人员。最终，一个普通的事故变成了严重事故。对于紧急的事情，必须要同步沟通。
* 半夜三点你正在熟睡，或者周末正在逛街，接到一个电话：“现在有个需求，能否立刻帮忙做完。”这会非常令人郁闷，因为那并不是紧急的事情。不是所有的需求都需要立刻解决。

有效沟通的一个重要原则是提前沟通。沟通本质是信息交流和处理，可以把被沟通对象形象地比喻成串行信息处理的CPU。提前沟通，意味着将处理请求尽早放入处理队列里面。下面的例子让很多工程师深恶痛绝：一个需求策划了1个月，产品设计了2周。当开发工程是第一次听说该需求的时候，发现开发的时间是2天。工程师据理力争，加班加点1周搞定。最后的结论是工程师非常不给力，不配合。就像工程师讨厌类似需求一样。要协调一个大项目，希望获得别人的配合，也需要尽早沟通。

有效沟通的另外一个重点是“不要跑题”。很多看起来很接近的问题，本质上是完全不同的问题。比如：一个会议的主题是“如何实施一个方案”，有人却可能提出“是否应该实施该方案”。 “如何实施”和“是否应该实施”是完全不同的两个问题，很多看起来相关的问题实际上跑题很远。“跑题”是导致无效沟通的重要原因。

良好沟通的奥秘在于能掌握TCP模式和UDP模式精髓，正确判断问题的紧急性，尽量提前沟通，避免跑题。

**带人之道**

有些初为导师的工程师由于担心毕业生的能力太弱，安排任务时候谆谆教诲，最后感觉还是有所顾虑，干脆自己写代码。同样的事情发生在很多刚刚管理小团队的工程师身上。最终的结果他们：写完所有的代码，让下属无代码可写。“ 事必躬亲”当然非常糟糕，最终的往往是团队的整体绩效不高，团队成员的成长很慢，而自己却很累。

古人说：“用人不疑，疑人不用。”这句话并非“放之四海而皆准”。在古代，受限于通信技术，反馈延迟显著，而且信息在传递过程中有大量噪音，变形严重。在这种情况下，如果根据短期内收集的少量变形的信息做快速决断，容易陷于草率。在公司里，这句话用于选人环节更为恰当，应该改为：录用不疑，疑人不录。

考虑到招聘成本，就算是在录用层面，有时候也无法做到。作为一个小团队的管理者，能够快速准确的获取团队成员的各种反馈信息，完全不需要“用人不疑，疑人不用”。用人的真正理论基础来自于“探索和利用”(Exploration and Exploitation )。不能因为下属能做什么就只让他做什么，更不能因为下属一次失败就不给机会。

根据经典的“探索和利用”(Exploration and Exploitation )理论，良好的用人方式应该如下：

* 首选选择相信，在面临失败后，收缩信任度。
* 查找失败的原因，提供改进意见，提升下属的能力。
* 总是给下属机会，在恰当地时机给下属更高的挑战。 总之，苍天大树来自一颗小种子，要相信成长的力量。

**效率、效率、效率**

经常看到有些同学给自己的绩效评分是100分——满分，原因是在过去一段时间太辛苦了，但最终的绩效却一般般。天道酬勤不错，但是天道更酬巧。工程师们都学过数据结构，不同算法的时间复杂度的差距，仅仅通过更长的工作时间是难以弥补的。为了提升工作学习效率，我们需要注意以下几点：

* 主要关注效率提升。很多时候，与效率提升所带来的收益相比，延长时间所带来的成果往往不值得一提。
* 要有清晰的结果导向思维。功劳和苦劳不是一回事。
* 做正确的事情，而不仅仅正确地做事情。这是一个被不断提起的话题，但是错误每天都上演。为了在规定的时间内完成一个大项目，总是要有所取舍。如果没有重点，均匀发力，容易事倍功半。如果“南辕北辙”，更是可悲可叹。

**架构师能力模型**

前面我们已经讲完了原则和一些困惑，那么工程师到底应该怎么提升自己呢？

成为优秀的架构师是大部分初中级工程师的阶段性目标。优秀的架构师往往具备七种核心能力：编程能力、调试能力、编译部署能力、性能优化能力、业务架构能力、在线运维能力、项目管理能力和规划能力。

这几种能力之间的关系大概如下图。编程能力、调试能力和编译部署能力属于最基础的能力。不能精通掌握这三种能力，很难在性能优化能力和业务架构能力方面有所成就。具备了一定的性能优化能力和业务架构能力之后，才能在线运维能力和项目管理能力方面表现优越。团队管理能力是最高能力，它对项目管理能力的依赖度更大。



**编程能力**

对工程师而言，编程是最基础的能力，必备技能。其本质是一个翻译能力，将业务需求翻译成机器能懂的语言。

提升编程能力的书籍有很多。精通面向对象和设计模式是高效编程的基础。初级工程师应该多写代码、多看代码。找高手做Code Review，也是提升编程水平的捷径。

**调试能力**

程序代码是系统的静态形式，调试的目的是通过查看程序的运行时状态来验证和优化系统。本质上讲，工程师们通过不断调试可以持续强化其通过静态代码去预测运行状态的能力。所以调试能力也是工程师编程能力提升的关键手段。很早之前有个传说：“调试能力有多强，编程能力就有多强。”不过现在很多编辑器的功能很强大，调试能力的门槛已经大大降低。

调试能力是项目能否按时、高质量提交的关键。即使一个稍具复杂度的项目，大部分工程师也无法一次性准确无误的完成。大项目都是通过不断地调试进行优化和纠错的。所以调试能力是不可或缺的能力。

多写程序，解决Bug，多请教高手是提升调试能力的重要手段。

**编译部署能力**

编译并在线上部署运行程序是系统上线的最后一个环节。随着SOA架构的普及以及业务复杂度的增加，大部分系统只是一个完整业务的一个环节，因此，本地编译和运行并不能完全模拟系统在线运行。为了快速验证所编写程序的正确性，编译并在线上部署就成了必要环节。所以编译部署能力是一个必备技能。

让盘根错节的众多子系统运行起来是个不小的挑战。得益于SOA架构的普及以及大量编译、部署工具的发展，编译部署的门槛已经大大降低。基于应用层进行开发的公司，已经很少有“编译工程师”的角色了。但是对于初级工程师而言，编译部署仍然不是一个轻松的事情。

**性能优化能力**

衡量一个系统成功的一个重要指标是使用量。随着使用量的增加和业务复杂度的增加，大部分系统最终都会碰到性能问题。 性能优化能力是一个综合能力。因为：

* 影响系统性能的因素众多，包括：数据结构、操作系统、虚拟机、CPU、存储、网络等。为了对系统性能进行调优，架构师需要掌握所有相关的技术。
* 精通性能优化意味着深刻理解可用性、可靠性、一致性、可维护性、可扩展性等的本质。
* 性能优化与业务强耦合，最终所采取的手段是往往折衷的结果。所以，性能优化要深谙妥协的艺术。

可以说，性能优化能力是工程师们成长过程中各种技能开始融会贯通的一个标志。这方面可以参考之前的博客文章“常见性能优化策略的总结”。市场上还有很多与性能优化相关的书籍，大家可以参考。多多阅读开源框架中关于性能优化方面的文档和代码也不失为好的提升手段。动手解决线上性能问题也是提升性能优化能力的关键。如果有机会，跟着高手学习，分析性能优化解决方案案例（我们技术博客之前也发表了很多这方面的文章），也是快速提升性能优化能力的手段。

**在线运维能力**

如果说性能优化能力体现的是架构师的静态思考能力，在线运维能力考验的就是动态反应能力。残酷的现实是，无论程序多么完美，Bug永远存在。与此同时，职位越高、责任越大，很多架构师需要负责非常重要的在线系统。对于线上故障，如果不能提前预防以及快速解决，损失可能不堪设想，所以在线运维能力是优秀架构师的必备技能。

为了对线上故障进行快速处理，标准化的监控、上报、升级，以及基本应对机制当然很重要。通过所观察到的现象，快速定位、缓解以及解决相关症状也相当关键。这要求架构师对故障系统的业务、技术具备通盘解读能力。解决线上故障的架构师就好比一个在参加比赛F1的车手。赛车手必须要了解自身、赛车、对手、同伴、天气、场地等所有因素，快速决策，不断调整。架构师必须要了解所有技术细节、业务细节、处理规范、同伴等众多因素，快速决断，迅速调整。

在线运维本质上是一个强化学习的过程。很多能力都可以通过看书、查资料来完成，但在线运维能力往往需要大量的实践来提升。

**业务架构能力**

工程师抱怨产品经理的故事屡见不鲜，抱怨最多的主要原因来自于需求的频繁变更。需求变更主要有两个来源：第一个原因是市场改变或战略调整，第二个原因是伪需求。对于第一个原因，无论是工程师还是产品经理，都只能无奈的接受。优秀的架构师应该具备减少第二种原因所导致的需求变更的概率。

伪需求的产生有两个原因：

第一个原因是需求传递变形。从信息论的角度来讲，任何沟通都是一个编码和解码的过程。典型的需求从需求方到产品经理，最终到开发工程师，最少需要经历三次编码和解码过程。而信息的每一次传递都存在一些损失并带来一些噪音，这导致有些时候开发出来的产品完全对不上需求。此外，需求方和产品经理在需求可行性、系统可靠性，开发成本控制方面的把控比较弱，也会导致需求变形。

第二个原因就是需求方完全没有想好自己的需求。

优秀的架构师应该具备辨别真伪需求的能力。应该花时间去了解客户的真实业务场景，具备较强的业务抽象能力，洞悉客户的真实需求。系统的真正实施方是工程师，在明确客户真实需求后，高明的架构师应该具备准确判断项目对可行性、可靠性、可用性等方面的要求，并能具备成本意识。最后，由于需求与在线系统的紧耦合关系，掌握在线系统的各种细节也是成功的业务架构的关键。随着级别的提升，工程师所面对的需求会越来越抽象。承接抽象需求，提供抽象架构是架构师走向卓越的必经之途。

市场上有一些关于如何成为架构师的书，大家可以参考。但是架构能力的提升，实践可能是更重要的方式。业务架构师应该关注客户的痛点而不是PRD文档，应该深入关注真实业务。掌握现存系统的大量技术和业务细节也是业务架构师的必备知识。

**项目管理能力**

作为工业时代的产物，分工合作融入在互联网项目基因里面。架构师也需要负责几个重大项目才能给自己正名。以架构师角色去管理项目，业务架构能力当然是必备技能。此外，人员管理和成本控制意识也非常重要。

项目管理还意味着要有一个大心脏。重大项目涉及技术攻关、人员变动、需求更改等众多可变因素。面临各种变化，还要在确保目标顺利达成，需要较强的抗压能力。

人员管理需要注意的方面包括：知人善用，优化关系，简化沟通，坚持真理。

* 知人善用意味着架构师需要了解每个参与者的硬技能和软素质。同时，关注团队成员在项目过程中的表现，按能分配。
* 优化关系意味着管理团队的情绪，毕竟项目的核心是团队，有士气的团队才能高效达成目标。
* 简化沟通意味着快速决策，该妥协的时候妥协，权责分明。
* 坚持真理意味着顶住压力，在原则性问题上绝不退步。

成本控制意味着对项目进行精细化管理，需要遵循如下几个原则：

* 以终为始、确定里程碑。为了达成目标，所有的计划必须以终为始来制定。将大项目分解成几个小阶段，控制每个阶段的里程碑可以大大降低项目失败的风险。
* 把控关键路径和关键项目。按照关键路径管理理论（CPM）的要求，架构师需要确定每个子项目的关键路径，确定其最早和最晚启动时间。同时，架构师需要关注那些可能会导致项目整体延期的关键节点，并集中力量攻破。
* 掌控团队成员的张弛度。大项目持续时间会比较长，也包含不同工种。项目实施是一个不断变化的动态过程，在这个过程中不是整个周期都很紧张，不是所有的工种都一样忙。优秀的架构师必须要具备精细阅读整体项目以及快速反应和实时调整的能力。这不仅仅可以大大降低项目成本，还可以提高产出质量和团队满意度。总体来说，“前紧后松”是项目管理的一个重要原则。

项目管理方面的书籍很多。但是，提高业务架构能力同样重要。积极参与大项目并观察别人管理项目的方式也是非常重要的提升手段。

**团队管理能力**

不想做CTO的工程师不是一个好的架构师。走向技术管理应该是工程师的一个主流职业规划。团队管理的一个核心能力就是规划能力，这包括项目规划和人员规划。良好的规划需要遵循如下原则：

* 规划是利益的博弈。良好的规划上面对得起老板，中间对得起自己，下面对得起团队。在三者利益者寻找平衡点，实现多方共赢考验着管理者的智慧和精细拿捏的能力。
* 任何规划都比没有规划好。没有规划的团队就是没头的苍蝇，不符合所有人的利益。
* 规划不是本本主义。市场在变，团队在变，规划也不应该一成不变。
* 客户至上的是项目规划的出发点。
* 就人员规划而言，规划需要考量团队成员的能力、绩效、成长等多方面的因素。

市场上有很多规划管理方面的书籍，值得阅读。最优化理论虽然是技术书籍，但它是规划的理论基础，所以不妨多看看翻阅一下。从自我规划开始，多多学习别人的规划也是规划能力提升的重要手段。

**总结**

因为受邀去做一个关于“一边工作，一边学习”的分享，作者花了一段时间去思考和汇总学习方法论，接着每天不断地采集谣言并尝试解惑，再根据个人经验绘制出优秀架构师的能力模型，最后汇集成文。

文章系统性地阐述了学习原则、分析了常见困惑，并制定明确学习目标，期望对工程师们的工作学习有所帮助。需要申明的是，文章内容挂一漏万，所谓的架构师能力模型也是作者的个人观点。欢迎大家在评论中分享自己在学习成长方面的心得。