

38 | 新入职一家公司，怎么快速进入工作状态？

郑晔 2019-04-12



00:00

讲述：郑晔 大小：10.46M

11:24

经过前面几个模块的学习，我们分别领略了各个原则在不同场景下的应用，相信你对于这些原则的理解也上了一个台阶。但实际工作并不会清晰地告诉你，到底该运用哪个原则来解决问题。

所以，在接下来的三讲中，我挑选了程序员职业生涯中三个非常经典的场景，与你一起看看怎么在实际的工作中运用好已经学习到的这些原则。

在综合运用这个模块的第一讲，我们就来谈谈，当你加入一家新公司时，应该怎么做。

IT 行业快速发展，无数的机会涌现了出来，程序员频繁流动是这个行业的一个典型特征。频繁换工作，无论是对公司，还是对个人都是成本很高的一件事。所以，在加入一个新公司时，怎么让自己快速融入，尽快发挥价值，是摆在我们面前的一个重要问题。

以行业标准来看，我换工作的速度是很低的，但因为之前工作的原因，我需要到不同的公司与不同的人合作，每到一个新公司，工作的内容就是全新的，就如同换了一个新工作一般。因为合作周期有限，我不可能像普通员工入职新公司一样，花几个月时间慢慢熟悉，只能在尽可能短的时间内，快速上手，而且还要提出自己的新想法。

那我是怎么做的呢？其实，我就是运用这个专栏里提到的各种方法解决这个问题。下面我就来分享一下具体的做法。

运用思考框架

还记得专栏之初我提出的思考框架吗？我们要问三个问题：

- Where are we? (我们现在在哪?)
- Where are we going? (我们要到哪儿去?)
- How can we get there? (我们如何到达那里?)

先来看第一个问题，如果刚刚加入一家公司，哪怕我们不是一脸懵，也只是对公司业务有一个简单地了解，这是我们的现状。

第二个问题来看看我们的目标。一般来说，我们都是打算在新公司大展身手，但这个答案太宽泛了，我们还需要进一步细化。在这个公司长远的发展是以后的事，我们还是把第一步的目标制定成能够达到上手工作的程度，比如，能够解决日常的项目问题。

那接下来，我们就需要回答第三个问题了，怎么才能够达到这个目标呢？我们需要做一个分解。

你可以回想一下过往的工作经验，要在一个项目上工作起来，先要了解什么呢？很多人的第一反应是技术，我是程序员嘛，当然就是技术优先了。估计大多数人进到项目里，都是一头奔进代码里，然后，从各种细节研究起来。技术肯定是你了解的，但它不应该是第一位的。

技术解决的是“怎么做”的问题，而我们第一个应该了解的问题是“做什么”。一个软件到底在做什么，能够回答这个问题的就是业务。所以，我们排在第一优先级的事情应该是业务。

了解业务和技术都只是让你扮演好你个人的角色，但我们通常都是在一个团队内工作的，所以，还要解决好与其他人协作的问题，这就需要我们了解团队本身是如何运作的。

好，我们已经将大目标做了一个分解，得到了三个小目标：

- 业务；
- 技术；
- 团队运作。

从大图景入手

接下来，我们来针对每一个目标，进一步看看需要了解哪些内容。

业务

首先是业务。这是程序员入手新项目时最容易忽略的点。在这个专栏中，我在不同的模块中都提到了知识结构的重要性，没有结构的知识是零散的。所以，不管做任何项目，都要先从大图景入手。只有了解了大图景，各种知识才能各归其位。

对于一个普通的程序员来说，业务就是这个大图景。

如果你了解了业务，你自己就可以推演出基本的代码结构。但反过来，如果让你看了代码，从中推演出业务，那几乎是不可能的。

事实上，每次了解到一个业务，我都会在脑中过一下，如果是我做这个业务，我会怎么做。这样一来，我就会先在整体上有一个预判，后面再对应到实际的代码上，就不会那么陌生了。

要了解业务，我一般都会请人给我讲一下，这个业务是做什么的，解决什么样的问题，具体的业务流程是什么样子的，等等。

在初期的了解中，我并不会试图弄懂所有的细节，因为我的目标只是建立起一个基本的框架，有了这个初步的了解，后续再有问题，我就知道该从哪里问起了。

理论上，了解业务是每个程序员都该做的事，但事实上，这也常常是出问题的地方。在请别人给我讲解业务的过程中，我发现，很多人是分不清业务和技术的，经常把二者混在一起讲。如果你跟着他的思路走，很容易就会陷入到对细节的讨论中。

所以，了解业务时，一定要打起精神，告诉自己，这个阶段，我要了解的只是业务，千万别给我讲技术。

技术

了解完业务，就该到技术了。这是程序员最喜欢的话题。但即便是了解技术，也要有个顺序，所以，我们先从宏观内容开始。第一个问题就是这个系统的技术栈：Java、JavaScript 还是.NET，这样，我就可以对用到的工具和框架有个大致的预期。

接下来是系统的业务架构，这个系统包含了哪些模块，与哪些外部系统有交互等等。最好能够有一张或几张图将架构展现出来。现实情况是，不少项目并没有现成的图，那就大家一起讨论，在白板上一起画一张出来，之后再慢慢整理。

有了一个初步的体系，接下来，就可以稍微深入一些。

我会选择从外向内的顺序了解起。首先是外部，这里的外部包括两个部分：

- 这个系统对外提供哪些接口，这对应着系统提供的能力；
- 这个系统需要集成哪些外部系统，对应着它需要哪些支持。

一旦涉及到与外部打交道，就涉及到外部接口是什么样子的，比如，是用 REST 接口还是 RPC（Remote Procedure Call，远程方法调用）调用，抑或是通过 MQ（Message queue，消息队列）传递消息。

不要简单地认为所有接口都是你熟悉的，总有一些项目会采用不常见的方式，比如，我曾见过有系统用 FTP 做接口的。

所有这些都相当于信息承载方式，再进一步就是了解具体的信息是什么格式，也就是协议。

今天常见的协议是 JSON 格式，或者是基于某个开源项目的二进制编码，比如：[Protocol Buffers](#)、[Thrift](#) 等等。一些有年头的系统可能会采用那时候流行的协议，比如：XML；有一些系统则采用自己特定领域的协议，比如，通信领域有大量 3GPP 定义的协议。

一般来说，从外部接口这件事就能看出一个项目所处的年代，至少是技术负责人对技术理解的年代。

了解完外部，就该了解内部了。了解内部系统也要从业务入手，对应起来就是，这个系统由哪些模块组成，每个模块承担怎样的职责。如果系统已经是微服务，每个服务就应该是一个独立的模块。

通常这这也是一个发现问题的点，很多系统的模块划分常常是职责不清的，因此会产生严重的依赖问题。在前面的内容中，我多次提到限界上下文，用限界上下文的视角衡量这些模块，通常会发现问题，这些问题可以成为后续工作改进的出发点。

业务之后是技术，对应着我需要了解分层。前面说过，[分层结构反应着系统的抽象](#)。我希望了解一个模块内部分了多少个层，每个层的职责是什么。了解了这些对系统的设计，也就对系统有了一个整体的认识。

设计之后，就到了动手的环节，但还不到写代码的时候。我会先从构建脚本开始，了解项目的常用命令。我预期从版本控制里得到的是一个可以构建成功的脚本，如果不是这样，我就知道哪里需要改进了。

最后才是代码，比如，代码的目录结构、配置文件的位置、模块在源码上的体现等等，这是程序员最熟悉的东西，我就不多说了。作为初步的接触，了解基本的东西就够了，代码是我们后期会投入大量精力的地方，不用太着急。

团队运作

最后，我们还要了解一下团队运作。同样从外部开始，这个团队有哪些外部接口，比如，需求是从哪来的，产品最终会由谁使用，团队需要向谁汇报。如果有外部客户，日常沟通是怎么安排的。

再来就是内部的活动，一方面是定期的活动，比如，站会、回顾会议、周会，这些不同活动的时间安排是怎样的；另一方面是团队的日常活动，比如，是否有每天的代码评审、是否有内部的分享机制等等。

通过了解这些内容，基本上可以大致判断出一个团队的专业程度，也可以知道自己需要帮助的时候，可以找谁帮忙，为自己更好地融入团队打下基础。

你也许会问，了解这么多东西需要很长时间吧？其实不然，因为只需要从整体上有认知，如果有人很清楚团队现状的话，你可以去请教，也许一天就够了，这也是我往往能够快速上手的原因。接下来，就该卷起袖子干活了！

总结时刻

我给你介绍了怎么把前面学到的知识运用在了解一个项目上，按照业务、技术和团队运作三个方面去了解。

大多数程序员习惯的工作方式，往往是从细节入手，很难建立起一个完整的图景，常常是“只见树木不见森林”，而我的方式则是**从大到小、由外而内**，将要了解的内容层层分解，有了大图景之后，很容易知道自己做的事情到底在整体上处于什么样的位置。我把上面的内容总结了成一份供你参考。

| | | |
|------|---------|----------------------|
| 业务 | 业务 | 做什么，解决什么问题，业务流程是什么样的 |
| 技术 | 技术栈 | |
| | 系统的业务架构 | 包含哪些模块、与哪些外部系统有交互 |
| | 外部接口 | 接口方式、承载协议 |
| | 内部模块 | 模块划分、模块职责、分层抽象 |
| | 代码 | 构建脚本、代码结构 |
| 团队运作 | 外部接口 | 需求来源、产品用户等 |
| | 内部运作 | 定期活动、日常活动等 |

附赠一点小技巧：使用“行话”。在交流的过程中，学习一点“行话”。这会让人觉得你懂行，让你很快得到信任，尽早融入团队。

如果今天的内容你只能记住一件事，那请记住：**了解一个项目，从大图景开始。**

最后，我想请你分享一下，你在入职一个新公司遇到过哪些困难呢？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。



©

 一手资源 同步更新 加微信 [ixuexi66](#)

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Ctrl + Enter 发表

0/2000字

提交留言

精选留言(5)



Obug

了解业务这一步是最难的，要么需要有人给讲，一般都没那么多时间，要么有详细的文档，一般文档都是零碎化的。

 1 2019-04-12



Y024

1.我会在权限允许的范围内，时不时的到处翻翻 ftp、内部 wiki 等资源，星星点点构建全貌（业务、技术、团队）；

2.梳理系统数据流。去年很火的电视剧「大江大河」里，宋运辉初入职场的方式就很值得借鉴：先走通全部流程，有个全貌，利用图书馆、师傅等资源再自己动手各个击破并绘制流程图，最终实践检验认知以技术说话融入团队。

「他就每天只要天气晴朗，绕着设备上上下下、里里外外地跑。一个星期下来，全部流程走通；两个星期不到，原理搞通，仪表能读，普通故障能应付；第三星期开始，他可以开出维修单，但得给师父过目；第四星期起，谁有事请假他可以顶上，坐到仪表盘前抄表看动态做操作。师父说他学得很快。

第四星期起，没人可以让他顶替时候，他在仪表室后面支起绘图板。先画出工艺流程图，经现场核对无误，又让师父审核后，开始按部就班地根据液体走向，测绘所有设备的零件图、装配图、管段图等。这工作最先做的时候异常艰难，首先是绘图不熟练，很多小毛病，尤其是遇到非标零件，还得到机修工段测绘，一天有时都绘不成一个小小非标件。如果车间技术档案室有图纸还好，可以对照着翻画，可档案室里的图纸残缺不全，前后混乱，想找资料，先得整理资料。资料室中年女管理员乐得有个懂事的孩子来帮她整理，索性暗暗配把钥匙给宋运辉，要是她下班不在的时候，让宋运辉自己偷偷进来关上门寻找资料。

机修工段的人本来挺烦这个宋运辉，说他一来维修单子多得像雪片，支得他们团团转，有人还趁宋运

辉上班时候冲进控制室指桑骂槐，被寻建祥骂了回去，差点还打起来。但后来集中一段维修高峰后，维修单子又少了下去，上面还表扬跑冒滴漏少很多，一工段和机修工段各加一次月奖，可见设备性能好。再以后遇到维修，他们不能确定要用什么零件，打个内线电话给控制室问宋运辉，一问就清楚。双方关系渐渐变得铁起来。基层有时候很简单，只要拿得出技术，别人就服。」

👍 1 2019-04-12

作者回复: 非常赞的补充！



西西弗与卡夫卡

有朋友正在转型，从乙方商业化产品的交付经理转向新公司的产品经理。原本得心应手的思维方式和习惯，遇到了巨大挑战。以前只需依据已有产品的功能出解决方案，能做就能做，不能实现就是不能实现，到某个时间交付什么功能很明确，考核是以交付签字为准。现在需要面对各方需求，自己想明白用户真正的问题是什么，最终要交付的价值是什么，没有一个实体的谁来签字，只有不断地迭代。

借鉴领域驱动设计，可以采用以下方法。简单描述的话，是一个点、一个圈再加一个箭头线，是不是有点像丘比特？

一个点，指的是用户核心价值。这是最关键的一条，基本上只能靠自己想明白。想，不是闭门造车式的苦思冥想，可以是已有的领域经验，可以从书本中学习，可以是大家的各种吐槽，可以是自己从旁边观察用户的实践，还可以是自己变身为用户的实践。有些人会纠结点想的对不对，迟迟不敢动手。其实一开始想得对不对不是那么重要，关键是要有这点，然后快速到市场上验证，根据反馈再调整。

一个圈，指的是围绕核心价值划出的范围，即领域驱动设计中的限界上下文。产品经理面临的一个现实是，各种人都会给你提需求，只要他们觉得和你有关，还时不时来问什么时候可以实现。需求轰炸之下很容易焦虑，不光自己焦虑，所有的利益相关者都会焦虑。依据核心价值，框出需求范围，在和各方交流过程中可以有一种确定性，减少焦虑，利于行动。大家（不光是研发团队，也包括其他需求方）就能明白，哪些和当前核心价值密切相关我们优先考虑；哪些与核心价值有关但它不在我们的范围内，属于其他团队需要他们协助；哪些有关系，但目前没想清楚价值大不大并且代价可能很高建议先搁置。范围不是一成不变，它随着时间会发生变动，所以我们不要追求固定，只要保证在某个时间段内，大家一致认同即可。

一个箭头，指的是实现路径，箭头指向核心目标（核心价值）。目标（核心价值）和范围描绘的是终极，而从现实到终极还有很多路要走，可能的路径还有很多条。我们需要琢磨怎么走更稳当，怎么走代价比较低，路上关键的里程碑是什么。路径对不对其次，重要的是思考过程，可以把关键点需要交付的价值、需要支持的资源等等梳理清楚。

以上

👍 2019-04-12



捞鱼的搬砖奇

融入同事吧，需要点时间积累

👍 2019-04-12



Zapup

曾有机会负责编写开发团队的“入职须知”，帮助新成员一步步快速进入状态。但一直在“关注点顺序”、“是否还有遗漏”、“哪些信息不急于传递”等问题上纠结。本文是比较全面了！

👍 2019-04-12

