创业

为什么会创业？

离开阿里的时候其实没想过要创业，因为那时候最重要的事情是父亲病危。我爸胰腺有些问题，医生说得做手术。不做手术肯定撑不了多久，做手术的话还有 50% 的概率能够救回来，但是需要观察一年。

当时我就不知所措。人到中年，基本上都会遇到这些问题，中年危机啊。那年我 38 岁，工作出了问题，小孩上学也出了问题，父亲还生病了，**所有的事情堆叠在一起，我的头发很快就白了。**

从阿里出来后，**我就一个人在家里待着，安心陪父亲。没想到的是，这时候，几乎国内所有的云计算公司都来找我，还有一些创业团队，**我就帮他们解决各种各样的技术问题。那段时间，我就感觉技术人的技术能力是被社会认可的，而且大家的付费意愿也很好。

记得当时，一家创业公司找我。他们花钱买流量，但大流量来了后，系统接不住，用户没办法注册，等于投放的钱打了水漂。于是，我帮他们解决这些问题。

他们系统是 PHP 的，我直接在线上帮他重构，没有重构源代码，只是重构架构。然后大概干了两天吧，基本上就可以救火了。我用了一些纯运维的手段，在不改代码的情况下，帮他搞定这种高并发的营销活动。

这一套都搞定之后，**他问我要多少钱，我说两天 5000 块钱，你这没啥技术含量。**他赶忙说可不是，我必须得给你五万块，因为你帮我们解决了大问题。

就这样，我一边在家里待着陪父亲，一边帮其他公司解决问题。印象中有一个月，我大概一口气服务了七八个公司，挣了 50 万。算了下，那段时间，我的平均月薪是我在阿里、亚马逊三到四倍的样子。

我意识到，可以创业了。很明显，市场对技术是有强烈需求的。**因为绝大多数好的技术人员都被大公司垄断了，但中小公司也要发展**，他们的发展也需要技术，所以他们需要像我这样的一些技术人员，能够帮助他们解决一些实实在在的技术问题。

或主动，或被动地，我走上了创业这条路。我希望能用标准化的方式，来解决客户遇到的问题。之前，虽然我以救火队员的方式，手忙脚乱地暂时帮他们渡过了难关，但心里也清楚，那不是长久之计。

我要创业，做一款标准化的产品。

饿了么的技术故事

再讲个故事，那段时间，我也和饿了么合作过。饿了么的电商，是我经历过的所有电商系统中最难做的。

电商系统中，我觉得第一简单的是阿里的电商，因为它没有库存，所有的问题都是软件层面的问题。

第二是亚马逊，**亚马逊有库存，所有的订单都要甩到库存去，但是他们都不是实时的，送货可以隔一天慢慢处理。**它不需要实时，你可以用异步的方式解决很多问题。

但像饿了么这种实时订单类型的，就很难用异步的方式做。因为它要解决的问题是每天中午用户集中下单，想在 30 分钟之内吃到午饭，就需要在几分钟之内快速调动所有的资源来为一个订单服务。

当然， 这不只是技术问题，还是个业务问题，涉及商家、外卖员等等链条。不过，它有一个好的地方，就是区域性特征明显，很适合分库分表。

我接触到饿了么团队的时候，他们系统并不稳定，经常会挂。而且有一个比较有意思的现象是如果美团挂了，那饿了么必挂。或者说是饿了挂了，美团也会跟着挂。

为什么呢？因为没有任何一家公司可以在那一瞬间扛住全网的流量。所以，饿了么就想做异地灾备，**他们想的是既然我系统经常挂，那我就到异地另外一个机房建个灾备系统。**

后来我跟饿了么 CTO 雪峰讲，灾备不行，还是要做多活，而且饿了么天生可以做多活。淘宝是全国人民在任何地方都可以买任何地方的商品，但饿了么是本地消费，上海的在上海买，北京的在北京买。天生就适合做成异地多活。

雪峰一听，觉得这个技术决策太夸张了，步子迈得有些大。我说这事对你们的架构和团队都有极大好处。其实我这人，**经常在客户那边都会给出这种比较大的命题，有的客户可能会觉得我不是来帮他们解决问题的，而是来让他们难堪的**，哈哈哈哈。

对于我提出的异地多活的解决方案，饿了么的团队犹豫了很久。这中间，我一直在努力说服他们。有一次，我没忍住给雪峰打电话，我说不行，你必须得大张旗鼓地干，要不然无解。

你面对的是一艘摇摇晃晃、力有不逮的小船，今天这补一个漏洞，明天那补一个漏洞，你这种一点点的小修小补是不行的，你必须要大来。而且通过这个项目你可以全部把所有架构都梳理一遍。这多好的事。

后来，他们听进去了，启动了饿了么异地多活的重构项目。为了这个项目，所有的需求全部停掉，全部让路于这个项目。

做到最后，饿了么几乎所有系统都上了异地多活。

记得有一天，比较好玩的是，订单峰值一下子上去了，是平时的 1.7 倍、1.8 倍的样子，就想是不是市场在做活动。结果市场说没有做任何活动，大家都懵了，不知道为什么订单量暴增。

过了 20 分钟，就有新闻爆出来，说美团宕机了。**但饿了么北京机房和上海机房是 1:1 的，可以容纳全国所有流量，遇到类似的情况，不会再挂了。**后来，我还和雪峰开玩笑说，要不然咱们这样，先让饿了么宕机，然后所有流量就会到美团那边去，美团肯定撑不住，肯定要宕机。这个时候我们再把饿了么拉起来，撑住全网的流量。哈哈哈，这纯粹是在群里开个玩笑。

关于创业方向的选择

为什么我创业会去做网关。其实要讲清楚网关这些东西，我应该放出一个打印的 PPT 出来，像老罗一样拿出一个 PPT，嗯，现在我放不出来（笑）。

我整理一下思路。其实用户来找我无非就是几个事。第一个是要我做高并发的事，比如营销活动。第二个是想提高上线速度，比如可能需要用灰度发布、实时发布手段，来加快速度。第三个事情是用户量很大，速度又很快，系统必然不稳定，所以需要高可用。

你发布速度快，必须要并行发布，肯定要用微服务。不用微服务的话还是单体，所有人都在排期，不可能快。**我记得阿里的交易线上就不是一个微服务，很多人都在里面写，写相同的代码，一个大单体往上发布，得排期。**阿里那种所谓的“快”其实也是拿人堆，并不是技术上你可以很悠闲的那种快，他是那种很忙碌的快，你通常要加班，如果排不了期就得赶下一期。不可能快的。

所以，所有这些技术都会导向高并发、开发速度要快、并行开发，再加上高可用。最后都会导向只有那么一个类技术，就是云原生、分布式、微服务，还有 Kubernetes。但这些东西是不是能解决用户的问题？并不一定啊。

引导到这些地方，我们再抽象一下，**基本上就 4 块，流量治理、服务治理、资源治理，还有数据治理**，这四块东西会变成你的基础设施。要解决上层的问题，你的基础设施必须要好，就这意思。

就像我怎么让车开到 180 公里每小时，**你先得铺路去，先得把路搞好，就像中国的高铁，并不是车开不快，而是之前铁路不支持这样的速度**，所以基础设施不支持，你就没办法快起来。就像要让飞机起飞，必须要有大量大规模的配套设施，基础设施没跟上，你完全做不到。

现在我看到，服务治理有 Java 在做，比如 Dubbo 或者 Spring Cloud，资源治理是 Kubernetes 在做。**还有两个事情没有人做，一个是流量治理，我就想流量那么重要的事情，为什么没人在做？另外一个是监控，今天的监控就是整体运维的数据，你必须要有。**所以我们公司一开始起来做的就是一个流量网关，再加一个监控。

监控什么意思？我创业的时候，每一家公司都在比谁家的监控指标多，包括饿了么也一样。我记得我在饿了么的时候，有个同学来晋升，他说我们今天饿了么的监控系统可以采集将近 50 万个指标，到明年的时候，我要把它翻成 100 万个指标。

但是，数据太多等于没有信息啊。

有一个观点，**数据是没有用的，只有把数据关联起来才有意义。**数据关联了以后，才叫信息，我们不是做数据，我们是做信息。信息里面找到因果关系，我们才能有知识，比如说因为这个所以那个，这叫知识；有了知识以后，才能导出公式，我们才能通过公式去完成一些事情。

所有做科学实验都是走这条路的，不断地做实验、拿数据，在数据里面把它标注好，关联起来，然后找信息，从信息里面找因果关系，从因果关系里看看能不能推出一些公式。大概就是这么一个逻辑。

所以我觉得，今天所有监控系统都做错了，都在拼命地采集数据，而不思考这些数据到底里面有什么样的联系。

举个例子，我们做监控系统时，要说清楚这个对外的 API，后面连着多少 service？以及这些 service 运行在哪些 resource 上？又使用了哪些数据库、中间件？你如果说不清楚这些关联数据，是没办法做的。

今天的运维，因为是按技能分工，我只关心我的底层资源，不关心你的应用层。这就导致了一个问题。比如说我来问你，我想下线这台机器，它到底影响哪些用户？

我可以告诉你，在今天，中国没有人，没有任何一家公司，可以说清楚这个事情，包括 BAT。国外可以说得清清楚楚，因为数据有关联，我知道你这台机器到底服务着前面哪些 API，这些 API 有多少用户在用？我说的清楚这个事，因为数据是关联着的。

这就是我们公司为什么要走这条路。**一个是拿着流量的调度，另外一个是数据，我觉得这两个事，然后再辅以像 Spring Cloud 服务治理、Kubernetes 资源治理这样的产品，就完全齐活了。**这就是整体的基础设施。

我们公司叫 MegaEase，翻译成中文就是“巨大且容易”（笑）。

MegaEase 的 GitHub 地址：

https://github.com/megaease。

为什么我公司会选择 Go 语言？

我可能又要唠叨一点了哈哈。

首先，做流量网关，其实国内现在已经有一些东西了，一种是偏应用，一种是偏控制面。偏控制面的像 Nginx 或者 HAProxy；偏应用的像 Envoy，还有 Java 的 Spring Cloud 或者 Zuul 。但 Java 那些东西的性能又很糟糕，性能好的就在 C 和 C++ 这边，但我觉得语言门槛太高了，一般人是玩不了的。

因为流量调度网关，一定是会有业务逻辑侵入的。有业务逻辑侵入的话，就必然需要大家可以定制、可以在上层进行开发扩展。所以我们选择技术，首先你要有很高的扩展性，而且技术门槛要低。

如果用 C 语言，我相信能够参与贡献 Commit 的人少之又少，Node.js、PHP、Java、Python 都属于门槛比较低的，Go 也是，但是我们又需要兼顾性能。看了一圈所有，只有 Go 语言和 Rust 语言。

但是你要说为什么没有选 Rust 语言？首先 2016 年 Rust 还不怎么样，这几年非常火我们也在用。我想告诉大家 Rust 语言是一个很好的语言，但是我觉得它的门槛也是非常非常高的，甚至会比 C 和 C++ 还要高，真的。

所以我们觉得，要让大家可以去贡献、去扩展，语言门槛放在这里就会导致整个社区变凉，为什么像 PHP 的社区，像 Java 的社区、 Python 社区、Node.js 这种社区很多很多？就是因为门槛低参与人多，所以基于这些方面的考虑，我们选择了 Go 语言，这是第一个因素。

然后第二个因素，**为什么我们会选择 Go 语言，而不是其他的？**

因为我觉得一个流行技术必须要满足这么几个条件。第一个门槛低。除了门槛低以外还需要有大公司在后面撑着，没有大公司用的话不行，因为大公司用了以后就会在上面贡献更多的特性。

而且大公司用得越多，就一定会有标准化的东西出来，比如 Java、C 和 C++，很多公司都在用，就有一个标准化组织。第三个，社区很重要，必须要有社区。第四个它必须要有杀手级应用。杀手级应用的意思就是它必须要有个成功案例，没有成功案例不行。

所以这 4 个因素：**门槛低、有大公司撑着、好的社区，还有杀手级应用，决定这个技术会不会成功，会不会爆。**

PHP 也有大公司用，比如 Facebook，社区也很好，门槛也低，又有杀手级应用 LAMP，所以 PHP 能爆火。虽然我不是很喜欢这个语言，我非常非常不喜欢。像 Java 语言也是全部都有，门槛也低，大公司不断地投入。Go 语言其实也是这样。

但 Rust 目前来说还没看到。Rust 门槛比较高，社区并不活跃。大公司只有 AWS 说要用。杀手级应用才刚刚开始有苗头，一个是 WebAssembly 就是 Web 上的汇编，一个是它逐步进入内核了。所以现在，Rust 有一点点起势的苗头。**但是它有一个非常致命的缺点，就是门槛太高。**

我很早就总结过：Java 适合做业务层，Go 适合做中间件、Rust 适合做系统层。很多公司业务层也都在用 Go 写，我觉得没啥问题。是这样，业务层 Go 可以写，你拿 Node.js 也能写业务层，我以前都是拿 C 语言写业务层，所以你拿什么东西写都可以。

Go 语言写业务层，我觉得天然有一个好处就是 Go 没有太多乱七八糟的东西，它特别适合写面条代码。**所谓的面条代码就是这个业务流程长啥样就怎么写，不需要任何抽象，这些代码也懒得重用**，我直接写就好。国内的很多 Go 语言开发者，我觉得可能有 80% 都是从 PHP 转过来的，因为太适合写面条代码了。

但是，当你的架构变得越来越大、越来越复杂，里面一定要有一个框架的。比如 MVC 框架。**为什么你发现 Java 代码写不烂，因为它有框架保证。**但 Go 语言代码我看烂的就有一大堆。另外，服务越来越多，你开始要运维，做配置中心、服务治理那些中间件，但这方面，你会发现 Go 语言可用的不是很多，而 Java 那边成熟的一大堆。

所以，早期的项目，你用什么语言都行。**假如说我们把场景分成 0 到 1、1 到 10、10 到 100**，我今天跟大家说， 0 到 1 你爱用什么语言用什么语言，随便你用，Rust、Swift、Kotlin 随便什么，C 语言都行，你要用汇编写业务代码，我觉得都没问题。

但是 1 到 10，你开始要尊重两个东西。第一个你的开发速度要快，第二个你的稳定性、性能这些要保证因为要扩展用户，这个时候，你就必须得选一些工业化的语言。像 PHP 可能就有点受不了，Go 语言或者 Java、Ruby、Python 这些都还可以，因为生产力各方面都还可以。但是如果 10-100 的话，哎呦，我估摸着只剩 Java。所以，关键是你公司在哪个阶段。

有时候我们上网一看，哇，**大量都是 Java 不行、Java 反人类、Java 太啰嗦，但是你可以看看，说的那些人他们可能都在小公司哈哈哈**，因为他的业务场景的确不需要用 Java，他说的是对的。或者是那些独立开发者，都是从 0 到 1，他爱用啥就用啥，自己舒服就好。

但是你看那些大公司，银行、电商、电信，只要是这种有交易型的用户公司，包括哔哩哔哩所谓的自己用 Go，但是他做电商，你看他用啥，还是用 Java。Java，跑不掉的这个事情。

创业几年的反思

我创业也是很坎坷崎岖的。看似很好的一条路，但是一到创业的过程中，就发生了像中美贸易、疫情、资本趋于谨慎等等各种问题，互联网一片惨淡。现在也还有很多各种问题，所以其实都还是挺不容易的。

但是我坚持"条件受限是好事"，让我可以去想一些更重要的东西。比如很多公司可能想怎么样拿投资，现在我在想的是，怎么样能让我们的公司不拿投资，自己成为一个能盈利的公司，能成为一个真正的公司，不是被人包养需要寄养的一个公司。这些是我每天在思考的。

其实大家也可以去思考，**这个世界其实并不完美，你会遇到很多很多让你很无助的事情，**我也经历过工资发不出来，自己掏钱给团队发工资的时候，我一晚上睡不着觉，也经历过。

但是我想说的就是，所有的路都是可以走的，不用担心。但不是纯靠坚持，关键是你一定要去想方法，想更好的、更优的解，这个事很关键，**千万不要使蛮力，没有太大意义。这是我创业学会的第一个道理。**方法会比努力更重要，所以我们不加班，如果团队在加班，我觉得我们肯定方法不对。

我创业学会的第二个道理是关注你的重点，**不追小兔子。能让你分心的事情太多太多了，你不知道你前面那座城池有多高。**因为你从来没见过，一路上都会有很多很多的小兔子来找你。你一看，这个兔子好看，我来追一下，那边又有一只蝴蝶，你就忘了你未来要去哪。

这两个东西我觉得挺重要，这也是我创业这几年来受到最大的两个教训。我觉得跟我们的人生也是很相通的。

我的梦想

我活到 40 岁了，算是看出来，之所以这些大公司能垄断，就是因为垄断了技术人员，垄断技术人员他就会垄断掉这些技术，用技术来压制你。你们觉得有人写程序抢月饼，他把那些抢月饼的人开掉了，其实他在整个社会上跟你抢月饼，跟所有没有技术能力的人抢月饼，你们根本抢不过他。

所以我一定要把这个抢月饼的技术给到大家。我不觉得这个世界就是那一些公司在垄断的，我希望能够把技术惠及到所有人，大家都能抢月饼，就这意思。

所以我们公司的使命就是，**残酷无情地降低技术门槛，让普通的公司可以用到这些高级技术。**不是让你用云计算，我是要让你用到解决方案，让你一行代码不改做秒杀，让你系统再差也可以做秒杀，让你可以做高可用，成本很低地做高可用，成本很低地做灰度发布，就这样子，让你可以去跟大厂去抢月饼，他们每天都在用各种各样的技术手段在抢月饼。

我觉得这些技术并不难。难者不会，会者不难。

技术

技术不是用来写 CRUD 的

低代码会不会颠覆外包公司？这是一个很有意思的话题。过去一段时间，云计算，更多是对企业运维侧工作的变革。特别是 IaaS 和 PaaS，它做的都是技术侧的事情，比如怎么管理计算资源，怎么做好监控。

我相信，**未来云计算肯定是要慢慢向业务渗透的，因为有一部分业务是可以标准化的。**从这个角度看，外包公司可能会被颠覆。很多中小公司并不一定需要一个特别复杂、定制化的产品，它们的需求是可以快速使用。

而快速使用、开箱即用类的产品，有两类。第一，SaaS。像项目管理软件、聊天软件、销售软件这类的产品，企业没有特别多的定制化需求，标准化的产品就够用了。第二，低代码。通过低代码，企业可以搞定一些简单的定制化类的需求。

很多外包公司我觉得还是缺少思考。**一个人，一家公司，如果不思考的话，那颠覆你的往往不是不是这些云计算、低代码这些技术，而是什么都有可能颠覆你。**因为你只是在按部就班地做事，别人让你做啥你就做啥，你没有去想更多。

客户说我要一匹更快的马，你就去训马了，而不是想着做个汽车。这是外包公司最大的问题，他们没有任何的话语权，客户让他干啥他就干啥。而绝大多数客户，绝大多数时候是不知道自己想要什么的。

低代码肯定会替代一部分简单的 CRUD 类的工作，但我觉得我们技术人员本应该去干更高级的事情，而不是纠结这些。我想说的是，技术不是用来写 CRUD 的，技术是用来创新的，这一点大家要理解。

如果你只是把业务流程给数字化、代码化，那即使在大公司里，你本质也是做外包。不要觉得在大公司里就不是做外包，我给你讲，你很有可能是在给那些运营做外包。**要理解，真正的技术是要拿来做创新的，你要去颠覆点什么，要去创造点什么。**

基础知识的意义

我一直强调要学好基础底层技术。两个原因，第一，你得知道原理，知道某个技术它是怎么运作的。不要着急，**有的人可能会说今天学了也没什么用，你还是要坚持学，因为你慢慢会发现，很多东西都是相通的。**

第二，**当遇到一些比较难解的问题时，你学到的这些知识就会派上用场。**这些难解的问题会让你跟别人拉开差距，懂基础和不懂基础的人，他们的思考完全是在两个层面。不懂基础知识的人，他就在那瞎搞，这里试一下，那里试一下。运气好，碰巧解决了，但也不知道怎么回事。而懂基础的人，他可以很快理解大概是怎么回事。

我举个例子，Linux 操作系统有时候我们会感觉写硬盘很慢，但我想说，写硬盘其实不慢，因为它有 Page Cache。**你写硬盘，其实也是写在操作系统的内存，然后内存在一点点换页换到硬盘上。**很多中间件，比如 RocketMQ、Kafka，都会使用到 Page Cache 技术。只要 Page Cache 玩得好，那系统性能也会不错。

还有像 TCP 原理。**有些时候网络连接会闪断。从 a 节点到 b 节点，中间经过了很多设备，你怎么排查故障？**如果你懂原理，那就会容易些。第一，你先看看 TCP 的状态，如果是 time wait 的话，那就是我主动断开连接；如果是 close wait 的话，那就是对方主动断开连接。你看状态就能有一个基本的判断。

第二，如果没有看到连接建立的话，你需要使用抓包的方式，像什么 tcpdump、wireshark 之类，用这些工具，就能很快定位问题。如果你不懂基础知识，那你很可能就不知道怎么使用这些工具。**而没有这些工具，解决那些问题，你估计就得使用蛮力了。**

如何选择技术？

每次直播的时候都有人来问我一个问题：我该不该从 .NET 转 Java、该不该从 Java 转 Go。我不直接回答这个问题。

我不知道这个问题背后的思维方式是啥，就是 .NET 怎么样？或者说 .NET 转 Go 怎么样？我不知道你为什么想问这个问题，你想从我这边得到什么样的答案？想转就转呗。**是不是你需要有个人帮你坚定想法，还是需要有人告诉你一个答案，你才去做这个事情？我觉得你不要把人生的一些决定寄托在我身上，你自己的决定最好自己做。**

但是，我跟大家说一下，我之所以能够做成今天这个样子，**就是我随时都在追逐一些我觉得是主流的技术，它是未来技术。**就像我在阿里追 Docker，惹了 3 个团队。但是我一点都不后悔，为什么呢？因为我今天的很多想法都是因为我去用了这个技术我才会有。

包括我学 Java 也是一样，**我始终都觉得我必须要靠上主流。说得俗一点，就是你必须要去风口，这个风口猪都会飞。**但是，要看那个风口是不是个短暂的风口，千万不要去找短暂的风口，你要去找长风口，风可以吹一辈子的。

因为你是猪，你是没翅膀的，你必须要一直有风吹着你才能飞；否则的话，你飞不起来，因为你没翅膀。**你不是鸟，没风都能飞，绝大多数人跟我一样都是猪。**

所以你需要站在一个天天都有风吹的地方，**比如说你选对了计算机专业，这就是一个风不停地吹的地方。**过去也好，今天也好，未来也好，整个世界都是被一堆程序员驱动，就是这些写代码的人在改变着未来。

所以今天这个风口是没啥问题的，那问题就是你有没有站在这个风口里面的主流的技术上。

比如说，我觉得主流技术就是 Java，你又没去学 Java。你看那么多公司，有多少公司在用这个技术？这个技术可以干什么？我用了它我就不会失业，对吧？Go 语言也上来了，你是不是也要去看一下？满世界都是 Linux，你怎么可能不学呢？所以就这个意思。

现在都是手机，都是移动互联网，所以你也得要去了解一下。包括云原生很火，大数据也很火，这些都需要我们去看一下。

但是像区块链这些东西，你就扪心自问一下，区块链是不是所有公司都在用？没有用就让那些猪在天上再飞一会，指不定哪天风停了呢。但是，今天那些很强劲的风已经刮了十多年、二十年了，你为啥不去呢？

那些只刮了一两天的风，你要小心，因为你没翅膀，飞上去以后，风停了你就会摔得很惨。**那些刮了十几二十年的风，你还没站上去，要赶快站上去。**

如何完成早期的技术积累？

很多人觉得我已经完成了自己的技术积累，已经开始探索马斯洛需求层次理论上层的事情了。是这样子。

很多人知道我的时候，我已经有点名气，我已经去了阿里、亚马逊，反正稍微有点成就。所以**你并不知道我刚开始出来是去了工商银行，他们给我分了套房子。我父母是下岗工人，每个月两个人加起来就 200 块钱工资，我是 600 块钱，我们一家三口 800 块钱。**

但当时，我不要了，全部都不要了，我要出去到上海做外包。

到上海的时候我拿 1500 块钱的工资，租房 900 块钱，然后一顿饭要吃 15 块钱，你们算一算，一天的生活费我只够吃一顿饭，而且我还得自己做饭。公司说出差一天有 30 块钱的出差补贴，而且出差还管饭，所以我必须得出差。一出差，我的收入就好很多。

那时，我无时无刻都在想着，我要去更好的地方。那时候我们封闭开发，周一到周六全部待在开发中心，家都回不去。**即便这样我也要去学习，周日我也要出去面试，去改变自己。**因为那种没有技术含量的事情，我知道再做 100 次，也依旧没有任何未来。

老实说，**我跟那些外包公司极度不和，我跟老板吵架，我甚至跟甲方的行长吵架你们相信。**因为我把事情做完了以后就在那看书，那个行长过来说，大家都在这外包，你为什么要在这看书？我说我看的又不是闲书，我看的是技术书籍，你凭什么不准看？你安排活给我做就好了，你管我看不看书？我就跟他吵架。

后来我选择职业或者工作的时候，我宁可被公司开掉，也要去做有价值的东西，我从一开始的初衷就是这样子。**并不是说今天功成名就了，我才这样选；而是我一直都在这么选，所以我才会有今天的成就，大家要理清楚这里面的因果关系。**你不这么选，你永远不可能成为这个样子，明白吧？

现在我做公司也类似，有些项目我不接就不接了，我要接项目的话，也可以把自己做成一个外包公司，一年也有几个亿，但是有什么意义呢？我设想是十几个亿的事情。

刚才有人跟我说我做的事情太大了，有点不可能。**这就对了，因为绝大多数人都不敢想，他们想一想就觉得这事不可能，就离开了，所以我没有太多的竞争对手，明白吗？**因为绝大多数人都不敢想，他们一想他们觉得我做不到，就傻掉了，他们自己就给自己思维框架里面就设了一座墙，因为连尝试他都不敢尝试。

至少我敢尝试，失败了又能怎么样呢？反正我不会后悔。**我未来老了以后，四五十岁看我这段时间我至少试过失败、至少试了一个牛逼的事情。**

那你们呢，自己的人生你连试都不敢试，想都不敢想。人，最可怕的是不敢想。**不**好意思，讲得稍微有点激动。**

成长

996 没有成长怎么办？

“毕业两年，现在公司里头都是 996，日常的业务需求做不完，感觉自己在这个公司没有什么成长，学习也没有多少时间，是不是应该考虑换工作了。”

对于这个问题，我想问，大家想过没有，你找工作的目的是啥？

任何事情我们都得回到目的上，**如果找工作的目的是为了找碗饭吃，不让自己饿死的话，那 996 这些我觉得都顺其自然就好**，因为这是让你不饿死的一条路。

如果你找工作的目的是为了实现自我价值，**那假使现在的公司不能帮你达到这样的目标，你就应该勇敢寻找能够帮你达到目标的公司。**比如，我以前是在银行里，从甲方跑出来，房子都不要了，我就是为了实现自我价值，所以我选择公司的标准就是这个公司可不可以帮助我实现自我价值。

这是互相成就的一个事，我跟我的团队的员工也是这样说的：**你成就我，我成就你，这是双向的事情。**

所以我觉得换不换工作问题在于，你得问一问自己，你想要什么？如果你想要的是提升自我，那就应该去找能够成就你的公司。那些公司也会问你，你能成就我啥？这就是一个相互交换的过程。当我们不断去索取的时候，我们也需要给予。

我贡献价值，反过来，我希望公司也能给我价值。这是互相交换。大家都是在签合同，是雇佣关系，而不是什么劳动关系，不是说是我为你工作，我就是你的人，不是卖身关系，这是个相互合作的关系。所以，如果不能达到目的，我建议你不要去，公司也需要很好的人。**好的机会从来都是给好的人，不会给不好的人。**

大公司和小公司的权衡

其实无论大公司也好，小公司也罢，只要你用心观察都能学到东西，但你不用心什么都学不到。

大公司最大的问题就是一个萝卜一个坑，你就只能干你那一小个环节，你就是他的零件。所以很多人觉得大公司里面学不到东西，反而在小公司里可以做一个面的事情，能学得更多。其实不是。到大公司里面，我觉得有三个东西是可以学得到的。

第一，组织管理。大公司组织的运作方式、体系化的管理方式，怎么调动那么多人来完成一个事；或者一些思维方式，为什么他要做这个而不是做那个。这是大公司跟小公司不一样的地方，因为大公司有资源，可以去试错，试过很多很多错误。每一个东西都是可以让你思考。**第二，怎么挣钱，大公司怎么挣钱的。**

第三，大公司里面的一些高手是怎么工作的。因为大公司里我觉得应该是有这个世界上最好的人。比如这个世界上最好的运营、最好的产品经理、最好技术人员，都在阿里、亚马逊、谷歌这样的一些公司。所以你进去要去找大公司里面那些聪明的人，跟他学，就像你打羽毛球下象棋一样，你必须要找高手学。就这三个东西，在大公司里面我觉得是最有价值的。

那亚马逊和阿里我学到什么东西？

在亚马逊，我学到的是他组织整个公司是以微服务架构以工程师文化来分工的，**不再以技能分工，是以职责分工，也就是说我让你完成一个事，你得从前端一直干到数据库，干到运维等等**，任何一个环节都不算完，必须全部从头到尾干完。

所以亚马逊拆分组织是竖着切的。比如说这是购物车，这是下单，这是发短信，这是地址，然后有一个团队负责所有事情，**这个团队是 two-pizza team，两张披萨可以喂饱，也就是 10 个人左右。**所以，亚马逊是小团队可以干大事。这是我学到的，相当厉害。**因为以前什么瀑布模型、敏捷，我都觉得有很多问题，直到我看到亚马逊的这种组织方式。**

然后怎么挣钱？亚马逊是用技术挣钱，所以你看他做了世界上第一个推荐系统，也做了世界上第一个云计算平台。

他是这么想的：**首先一件事情，你必须把它抽象化，因为抽象化之后才可以简化，简化后才可以标准化，标准化的事情才能自动化，自动化的事情才能规模化。**

简化—>标准化—>自动化这条线，全部都是技术在干的事，第一次工业革命、第二次工业革命是一样的，只有技术做到了，你才最终可能规模化，之后才有飞轮模型。这是亚马逊的理念。

那在阿里我学到了什么？

第一，怎么组织整个企业。**我觉得阿里用一种政委文化。我虽然不太认可，但是这也是一种组织方式。**另外是一种激情、热情，我觉得阿里巴巴的员工就像打了鸡血似的，可以为公司拼命地加班。当时双十一的技术支持团队，因为要做连续做 48 个小时，我说你们分成两个团队，一个做第一天，另一个做第二天，看行不行？然后他们说不，我们可以干 48 小时不睡觉，大概是这样。

而且我经常在阿里看到员工说我这事做不好了，我就去跳西湖，或者说有军令状对赌的方式，员工跟公司立军令状说我今年我要做 300 亿，如果做不到，我全年白干，奖金升职都不要给我，但是我要是做上去了，每一分钱我都要参与，我要分给我的团队。

我觉得，**阿里让我看到了跟其他公司完全不一样的文化，他们太有热情了，非常非常有热情**，比起外企那种闲着整天想着怎么早下班，是两种不同的文化。这是阿里的组织，我到今天我都在想怎么样把人的这种热情给激励出来。

然后怎么挣钱的？**电商，其实核心是想着怎么样让你更快地做决定，因为你做决定的时间越短，我流量的转化率就会越高**，对吧，这跟我们做技术一样的，响应时间越短，你的吞吐量就会越大。所以我在买一件东西的时候，做决定的时间越短，转化率就会越高。

阿里挣钱其实跟亚马逊有点类似，但是他们走的不是同一条路。

亚马逊走的路叫简单标准化，我为了要让用户更快地做决定，我要把最好的商品呈现在他面前，或者展示出能够影响用户做决定的那些最关键的因素，比如好评率，所以会有一些 rank。

但国内不是。国内要让用户更快做决定，其实要让用户疯狂。国外是认为让用户在理智的情况下，会更快地做决定，国内是觉得用户越不理智，会越快做决定，这是两种不同挣钱方式。

所以我觉得这些东西都只能在大公司里面学，小公司完全不开窍。那如果你一直在小公司去不了大公司，怎么能快速成长？

其实也能成长，**关**键是你有没有跟那些聪明人在一块。因为并不是所有聪明人都会在大公司里，越是聪明的人，可能越不喜欢待在大公司。**因为他待在大公司收益其实很小，他必须得没有约束，不需要去搞什么办公室政治。

所以你的朋友圈会比公司更重要。你到大公司，也是要看你的朋友圈，这是一个非常非常重要的观点，**你跟什么样的人在一起，决定了你怎么样成长；你跟什么样的人在一起，你就会被影响成什么样的人。**所以有的小公司也不一定不能成长，关键是你有没有跟对人？他的格局够不够？

我们公司就非常注重员工成长，每周都要让他们去做分享，我会留出大量的时间让大家去学习，甚至要去写东西。因为我觉得**写作是一种深度思考，在写作的过程中，你才会认真地去想、去总结、去归纳你所学到的知识。**

没有时间怎么办？

这个事情我是这样想的，你得管理时间。有时候向下管理，有时候向上管理。

向下管理就是管理自己的时间。

我们现在有点不好，**有手机随时都会来通知，一会就被打断。如果你的时间总是被打断的话，利用效率是不高的，所以不要被打断。**你说我们公司里面还有什么钉钉群这些东西，就很可怕的事，反正管理好自己时间。一般来说我喜欢专注做事情，在阿里有些时候我也害怕被打断，我就跑到别的办公区里工作，把手机直接静音，然后一个小时以后再出来回复一波。

然后第二个，向上管理。

比如说是这个事情本来就是得一个星期才能做完的，他就要让你两三天做完，你要自己接了，就不要怪别人。该跟他斗争的还是要斗争，但是斗争是要有资本的。没资本的话，就比较麻烦。

对于类似的事情，我的思路是我可以辛苦，我可以加班，我肯定会努力做。欲先取之，必先予之。等到领导对你、公司对你有比较大的依赖的时候，你就可以跟他谈任意的条件，因为他很害怕你跑了。这就是资本。

但沟通的时候，也不要太直。**我在外企里面学到的一个方法叫 Never Say No。他说和用户沟通的时候，你永远不要说不，这事我做不到。你要说 Yes，但永远是有条件的 Yes。**比如这种情况下我能做到，那种情况下我做不到。你给我更多的时间，我能做得到，你给我那么短的时间我做不到。或者说在那么短的时间内我只能交付你半成品，你要不要？

就是我永远都可以给你做出来东西，但是总是有代价的，总是有条件的。为什么要这样呢？别人把球传给你，问你能不能做？我们技术人员比较实在，好像只能回答"能做"还是"不能做"，那你就被 PUA 了，知道吧？你知道你被 PUA 是什么概念？**被 PUA 就说是你进入了别人的环境里面，它营造了一个空间，把你扔进来，按照他的逻辑，就问你，给你一个答案，你能干还是不能干？**那无论你的答案是能还是不能，你都是被 PUA 的。

你要反 PUA 的话，必须得把他拉到我们喜欢的角度来，把这个压力推回去，给他一系列的条件：

Option 1 ：我可以再按时完成，但是质量别想。

Option 2 ：你给我足够的时间，我保质保量全部交付。

Option 3 ：我还是按你这个时间，但是你必须砍掉一些需求。

三个 Option 传回去，就反转了，因为这个逻辑是你的。他这时候就要痛苦地做选择了。到底选 1、2 还是 3？因为每个选择他都要付出一些东西，那就让他去选，不要剥夺他的选择权，**不要硬杠，你把选择权传给他，表面上你是在很好的交流，实际上你在 PUA 他。**

我给大家再举一个例子，当时从阿里出来的时候，因为工作居住证的一些问题，所以我需要阿里给我赔偿，但当时他们就很可怕地说：你是不是要敲诈我？律师团队这么说。我一想我敲诈多大的罪啊？要把我抓到监狱里！因为我要求的那个数是比他要赔偿我的更多一些，所以我说：

第一个我不要钱，一分钱不要。既然你也认这个事的话，你给我写个道歉信，盖上阿里巴巴的公章就好。

第二个是按你说的赔偿，你不用给我道歉，你告诉我这个钱的事由是什么样子，然后盖上盖公章，不然你说我诈骗，我可不敢要了这个钱。也就是说你给我赔偿这个钱，必须说明这个钱是用来干什么的，然后盖上你的公章，不然我以后说不清楚。

第三个是这些都不需要，然后我们就签个协议，按照我的要求给我。他们自己肯定不愿意出具我可以拿出去说的这么一个证明嘛，对吧？

也就是说要反向 PUA，选项给对方，选项永远都说可以干的，但是你要有条件。**小胡同赶猪，我给你了 5 条路选，但是你肯定不会选那 4 条路，你只会选这条路**，因为那 4 条路他自己就去盘算了，就这样子反向 PUA 就是反过来给他更多的选项。

好，这些都是这些我教大家的，不单单是管理时间了，哈哈哈。

怎么才能学得快？

我之前说我学 Go 语言大概一两周时间就能够学会。但很多同学说自己已经是 Java 程序员，要学 Go 语言可能用半年时间都搞不定。

我跟大家讲一个故事。我以前刚去上海的时候买不了书，只能在书店里面看书。我就把书店当成一个图书馆，看完后我会做笔记。我希望大家都有一种写作或者做笔记的习惯。

以学编程语言为例。你学一门编程语言，总是要把它抽象成一棵知识树。

因为刚开始你学各种各样的语言，会发现，这个语言也有这几块东西，那个语言也有那几块东西。比如说，一个语言，它必然会有变量、分支，还有循环，这是它最基本的逻辑结构，所有语言都应该有的。然后更高级一点的特性，它也有一些数据结构，还有数组、字符串，再高级点比如泛型，还有面向对象的多态，还有一些类库等等，这些都必须得要有。没有这些东西的话，这个语言是挺难用。

所以我学一个新的语言，我就会把这个语言照着我的框架来套。这个语言有哪些分支逻辑、哪些声明变量，多线程是怎么玩的、面向对象多态怎么玩的等等。我就照着套，半天或一两天就套完了。这个语言也就学会了。

所以，**首先你有一个大的语言框架，新语言跑不出这个框架范围，然后你把语言特性往框架里面套就好了。**你学得多了，你未来就会越学越快的，因为你会自己总结。这就是学习的一个过程，当你有了一个系统的知识树以后，然后你去填充这个树，把它全部填满了，你自然就学会了。

但是，**我学 Rust 语言就没那么快，虽然 Rust 我也在套，但套完以后，准备上手写点东西的时候，就发现挺复杂。**

我讲一下我学 Rust 遇到的问题。首先，Rust 有所有权、生命周期、借用（借用就是引用）、共享各种各种乱七八糟的概念。

我写代码，要共享内存。**但是 Rust 不支持共享，一个变量赋值给另外一个变量，所有权就转移了。**但是我偏要实现共享，然而共享就要用引用，引用就要玩生命周期，我就去写 Lifetime ，写 Lifetime 又要写多线程。多线程里面必须得用闭包，我就要在闭包里面玩共享变量，闭包里面我就要用 Lifetime……完蛋了。

这个代码，我调了 3 天都没调通，编译我都编译不出来。我到 Stack Overflow 上问了这个问题，两天以后有人告诉我答案了。我看那个语法，哎呀，我都崩溃了，我是绝对不可能自己想得出来的。**所以我想说为什么 Rust 门槛有点高，当我想去干一件事情的时候，我却发现我干不出来。**所以我觉得它不是很成熟，学习门槛太高。

学了就忘怎么办？

很多同学说自己学了就忘，这种情况下，要不要继续坚持。

我觉得你学了就忘，**是因为你在用记忆去学，而不是基于问题去学。**任何一个技术，都是在解决一个实际的问题。你一定要明白，你学这个技术，它到底解决什么样的问题，它是为什么而生的。当你明白了这个“为什么”以后，你才能够记住那些原理。

“为什么”是件很重要的事情。大家可以看我的 CoolShell 上的一篇博客《如何做一个有质量的技术分享》，**你必须要告诉用户为什么，到底要解决什么样的问题。**

Why 这种学习方式，我们叫理性学习；What 这种学习方式，我们叫感性学习。任何事情都是先从感性从是什么开始，因为一开始你也问不出那么多 Why。比如数学是 10 进制，不是 2 进制，不是 8 进制，也不是 16 进制，那为什么一开始就设计成 10 进制，但是时间又是 12 进制？这些东西，你不知道 What 是问不出来的，但你一旦问出这个 Why，没人告诉你，你的老师可能都不知道。

你的学习一定是从感性到理性的，**所以一开始你大概知道有那么一回事，慢慢地看的东西多了，顺理成章你就会想去了解一下为什么。**比如说我学 Windows、Unix、 Linux 这三个系统，里面有很多东西是相似的，有些东西又不太一样的，学多了以后自然会问为什么 C 和 C++ 全都是玩指针， Java 里面为啥没有指针？

如果只学单一的东西，你不会有那么多疑问。你看多了，学多了，开始比较，自然会有疑问，有了疑问就会想去探究背后的 Why。

还有就是你遇到一些实际的问题，在解决这些实际问题的时候，也会去真正地理解这个技术。总之，带着问题去学。

对我影响比较大的书

技术书，对我影响比较大的是《Effective C++》，这本书让我感受到了作者 Scott Meyers 的严谨精神。**他不是把一个技术设计出来就好了，他会想这个事有各种各样的副作用，然后这些副作用怎么不断地求解，这种求真精神让我很受震撼。**所以为什么《Effective C++》是一本全球畅销书。很多后来的书，比如《Effective Java》、《Effective GO》都在模仿它。

书中有一句话，其实我记了一辈子，他说："**美丽的东西都是肤浅的。**"意思就是你看得见那些美好的东西都是表面上的，所以他就是肤浅。你把它剥开了以后，那些东西才是比较深刻的东西，所以千万不要被美所迷惑。哈哈，这也是我教育我们家孩子的，但是我不知道什么时候才能感受到这个东西。

第二本书是让我感觉写代码有一下子能够提高的《Code Complete》。这本书可能有点过时，微软的书，但是内容非常好。装在书包里面还挺重的，可以防身，你遇到有人要抢你，先拿这本书先，就是随身带着一块板砖，可以拿着来打人的。

这本书讲了怎么样把一个代码写好的各种各样的方向，以前从来没有人跟我讲过这些东西，没有训练，所以这是一本让我觉得很爽的一个东西。和这个书更相似的是像《设计模式》这样的书，比如《Beautiful Code》，就是教你怎么写代码，怎么组织代码，《重构》也是。

还有另外一本书，对我影响比较大：《Rework》。它不是技术书籍，但是我被它里面所有的想法，严重洗脑了。

里面讲"条件受限是个好事"，**因为条件受限可以倒逼你抓重点，倒逼你做自动化，倒逼你简化。我突然就想起我以前买不起书，那个条件受限的时候就必须要用更聪明的方式去学习，因为你买不起书，所以你必须要画知识图，全部都是在条件受限。**要是我条件不受限，可能我就不会这么样，我就会去买书，最后书堆着都没法看。

还有一个观点，**就是挠自己的痒处，自己觉得不爽了自己要去改变它。所以我觉得这个世界上很多能创业的人，或者说是能改变世界的人，一定是不能忍的人。**各位，如果你今天什么事都能忍，你想一想还是忍了，我不去抗争了，你一定不会成为这个世界的厉害的人，你一定没有什么利益的噱头。

这个世界是属于那些不能忍的人的，因为你不能忍了你才会想去改变它。知道吗？各位。从你身边的一点点小事开始，**你觉得不能忍了，不能抱怨了，你决定行动起来，改变点什么，这时候，你就会琢磨怎么才能更好。所有的东西都是从不能忍开始的。**

如果你什么都能忍，那对不起你一定是韭菜。我看过的很多好的产品经理，他们都是不能忍，都是各种不能忍。所以他们就可以做得很好。不能忍的人一般标准很高，就是这样子。

35 岁危机

我在一些群里，40 岁程序员的群，然后还有些 35 岁以上程序员的群。我觉得那些群我连一天都不想看，太无聊了，里面就是各种八卦，我不知道为什么这些中年人天天都喜欢聊八卦，今天看见这个社会热点又怎么样，那个热点又怎么样。

我觉得是这样的，不要觉得 35 岁、40 岁会是一个坎。我今天也在写代码，有句话是这么说的："**人不是因为变老了才没有热情，而是因为没有热情才会变老。**"

有的人说我老了，我学不动了。不是的，因为你学不动，你才变老了；不是你老了，你才学不动。**这个世界上有大量这样本末倒置的因果关系的逻辑，如果你学得动，或者你还对这个事情充满憧憬，有想法，愿意去奋斗的话，70 岁你都不老。**

像我今天这个样子，我也不觉得我老。我觉得很多年轻人可能还不如我，他们在 20 多岁、30 多岁，可能就已经放弃了自己的人生，我今天还没有放弃。所以， 35 也好， 40 也好，不用去想这个数字的事。