漫谈中小企业研发技术栈

--- 中小企业技术领导者的自我修养

此为试读版，完整版请扫码付费阅读

这远远不是终点，而不过漫漫旅途中的一个小驿站。驻足于过去和未来交织的路口，我记录下那些曾经的思虑，也有幸能和你们分享，然后收拾行囊，继续奔向更遥远的未来！

2019-01-08 雨夜

目录

1. 前言

1.1 缘起

2015 年的秋天，我通过猎头收到了两份回南宁工作的邀请，一份是去一个已小有规模的公司当技术总监，另一份是去一家智慧农业公司当高级软件工程师。第一份工作的内容跟云计算相关，而之前的三年我也是一直从事云计算基础平台方面的工作，因此看起来和个人经历、能力比较匹配，另一份工作邀请的公司目前还没有软件人员，但是有扩大队伍的需求，主要做大型农场的智能灌溉系统。

在从深圳离职之前，我的目标是再用几年，使自己成为云计算领域名副其实的专家，之前的三年，我在云计算领域付出了非常多的时间和精力，通读了云计算方面的各种图书资料，搭建过各种虚拟化云平台。计算、存储、网络是云计算的三个主要构成部分，每一部分我都曾经有较深入研究，延续这样的技术路线似乎也是合理的。但是聘请我当技术总监的这家公司，他们的研发团队在北京，南宁团队主要还是负责维护方面的工作，和自己的想法有点差距。而又转念一想，云计算主要还是巨头们玩的东西，大部分公司只需要了解怎么用好云计算资源就好了，而进入第二家公司，我有机会从无到有去构建一个软件部门的研发体系，从无到有去设计一个包含前后端的应用系统架构，而我已有的云平台方面的技术经验，则会让自己知道怎么更好地去使用云平台的资源。因此，我选择了第二份工作，实现了从系统平台架构到应用架构的转型，就像三年前，我从深耕了五年的 Windows

客户端开发转向 Linux 服务端开发运维一样，也算是一个较新的开始。

然而由于人手不够，加上团队成本等方面的控制和高管对软件开发进度的要求，一些基础的环境都还没搭建起来，就开始进行编码了。而一般我们说留到后面再弄的东西，往往都是再也不会弄了。这样做从短期看起来，似乎研发进度有保障，但是从长远来看，效率上会差挺多。没有持续集成、持续部署环境，只是写了个简单的 Hook 脚本，当开发人员上传网站代码到 SVN 时，这些代码会被自动同步到本地网站的根目录。对于编译型的项目，因为只有一两个项目，也是每次修改完代码，手动去执行 make 命令进行编译。

两年后，因为公司搬迁离家太远，我离开了这家智慧农业公司，去做了一阵智慧停车方面的项目。在做停车项目期间，接触到一些不大不小的公司，了解到他们都还在用一些老旧的系统和架构，在软件架构设计上非常落后，高层领导虽想找一些高手来重构或者重新设计软件系统，但是苦于高手难找而且往往太贵。这让我看到了一些机会，我想如果将自己在平台和应用方面的架构经验沉淀下来，成为一套套方案提供给这些中小型企业，这样既能帮助他们解决技术上的问题，又能实现自己的技术理想，将自己所知转化为产品，同时获取报酬，岂不美哉。在这种想法的驱动下，我和几个志同道合的朋友共同创建了优笛瑞博，这个名字来自“UTLab”的音译，全称为“Unite Talent Laboratory”。

随着这本小书的成书，优笛瑞博的研发体系也在逐步构建并不断完善。本书不会教你使用命令行一步一步搭建某个系统，不会带你进

入技术细节，而是通过阐述我个人的所思所想，给你一些方向上的指引和资料推荐，因此，它更适合那些具有一定开发经验、有志在技术架构方向有更进一步发展并成为中小企业技术负责人的程序员、架构师等人士阅读。而对我自己而言，也是给自己多年来汗水付出的一个交代。好了，地图就在你面前，路需要你自己来走完，请开启你的愉快旅程吧！

2. 研发体系

2.1 研发体系定义

我所理解的研发体系包括研发规范、标准、流程，规范即研发过程中需要遵守的规矩和准则，比如项目构建方式、文件组织方式、文档撰写方式，标准即做到怎样的程度能满足一般性需求，流程即需要以怎样的步骤去进行研发才能顺利完成项目。将规范、标准、流程糅合在一起进行工程化，再将工程化后的流程进行自动化。自动化并不包括在研发体系的概念里面，不过自动化水平某种程度上体现了一个公司的生产力水平，不容忽视。

2.2 生产闭环系统

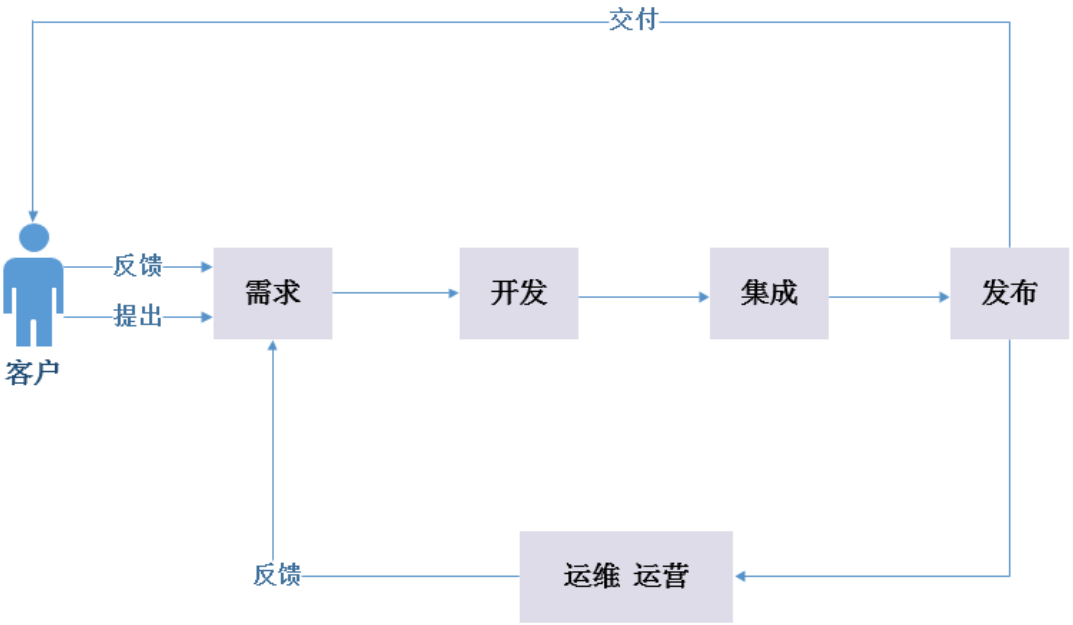
你可能听说过闭环系统，也可能并不清楚其实际含义，不知道怎么个闭环法，只貌似觉得这种说法很高大上。下面以工厂的流水线为

例来阐述我们对闭环系统的理解。

当我们构建了一个自动化的工程环境之后，这个环境就像工厂里的流水线，从获取原材料到各个环节的加工步骤都串联在了一起，生产者按照这样的流水线各司其职，到达最后一个环节就会生产出完整的产品来交付给用户。而用户的反馈数据或运维系统的反馈可以说是一种新的需求，这些需求就类似于新的原材料又进入到生产线的原材料获取环节，最后又生产出了新的或者改进后的产品，形成生产线的一个闭环系统。当然在这个过程中，这个闭环系统本身也在不断的改进以适应新产品或新方案的进入。

为什么要闭环系统呢，你可能已经看出来了，其实是为了效率和反馈，这两样东西是促使企业进步的重要因素。没有效率意味很容易被淘汰，没有反馈意味着难以修正自身存在的问题，因为根本不知道问题在哪里。

因此，我们的目标是要将研发体系打造成一条类似工厂流水线的生产线闭环系统，在软件研发中，这个闭环系统大致如下图所示：



2.3 研发体系成效

研发体系的具体成效也就是研发目标，我们认为目标包括研发效率、研发过程的可管理性和风险可控性，但是不包括你那一个亿的小目标。

研发效率也能体现一个企业的生产力水平，而研发过程如果可管理性不高，其实很难有很高的研发效率，因此，可管理性是高效率的必要条件。风险可控性是研发管理中需要特别注意的，因为风险不可控的话，一旦出现风险企业可能就失去造血或换血能力了。

3. 技术方向选择

虽然这一章叫“技术方向选择”，但我在这里也是没有办法给各位某种具体的技术选择的，因为具体到每个公司每个项目，情况千差万别。因此，这一章我将企业研发中可能遇到的相关技术方向先做一个罗列，然后阐述我个人在遇到这些问题时的思考方式，供您参考。

3.1 是否云化

百度 CEO 李彦宏曾经在互联网峰会上说云计算是新瓶装旧酒，后来估计他发现不是这么回事，才不得不赶紧在百度云的研发上加大投入以追赶其他巨头。而马云认为云计算的发展状况与阿里集团存亡相关，必须要做好，因此阿里云早早发力，目前两家云厂商的市场份额也相差甚远。如果说十年前很多人还在质疑云计算的话，今天几乎可以说大家已经全面拥抱云计算了。就本质而言，云计算提供了一种新型的计算资源交付的模式。过去我们在物理机上安装虚拟机来模拟多台电脑，云计算也类似这么做的，只是它在一个大的集群里面去创建数以万计的虚拟机或者容器，然后通过网络交付给用户，用户只要在有网络的地方可以随时随地使用计算资源。按需使用、自助使用、无处不在的网络接入、弹性计算，这些都是云计算的显著特征。云计算典型的分层是 IAAS，PAAS 和 SAAS 三层，这些相关概念的解释网上汗牛充栋，而且云计算目前已经发展了十来个年头，顺哥觉得自己没有必要在这里对基础概念再做解释，想深入了解的同学自行觅粮吧。

那么在公司自己搭建服务器和使用云平台产品的选择时，我们一般认为中小型企业基本不需要考虑自建的问题，因为自建成本往往会更高、周期更长，成本主要包括服务器成本、搭建时间、运维成本、安全质量保证等等，这些都不适合人、财、物都不充裕的中小企业去折腾，一般的中小企业应该专注于业务创新和应用产品的研发，把计算、存储、网络、安全等底层基础设施交给专业的云厂商更合适，当然，如果你的目标是建设一个提供基础设施的云平台，那另当别论了。

3.2 微服务化

微服务架构是近几年最为流行的一种架构模式，据说是从 SOA 衍生而来的，很多大牛级的人甚至调侃说自己区分不了这两种架构模式。有人开玩笑说，现在出去面试时，说不懂微服务架构，你自己都有点不好意思。IT 界大名鼎鼎的 Martin Fowler 和 James Lewis 大叔在个人博客上对微服务做了如下这么一个定义：

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized

management of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler

理解他们老人家的话之后，我们总结一下，微服务架构大致有以下几个特点：

1) 小，每个服务都很小；

2) 独，每个服务运行于独立的进程中，独立部署；

3) 轻，服务间通讯一般使用轻量级协议，如 HTTP；

4) 集，服务集中管理；

5) 多，服务可用多种语言编写；

那企业为什么采用微服务架构呢？这要从微服务的优势说起。微服务因为以上几个特征，使得这种架构具备很好的可扩展性，并且每个服务符合高内聚低耦合的设计原则，服务间通过轻量级协议如 HTTP 进行交互，使得这些服务可以使用不同的编程语言编写，只要接口遵循 HTTP 规范就好了。当某个服务出现问题时，只需要修改该服务的代码然后重新部署，其他服务不用重新部署，避免了单体服务牵一发动全身的缺点。

虽然如此，企业一上来就采用微服务，可能并不是最合适的，如

果业务规模和团队规模都比较小，采用微服务架构并不见得有什么好处，而且一般每个微服务部署在独立的 Docker 容器，这样就会使容器数量大大增加，每个容器的业务逻辑又比较小，白白给部署增添负担并耗费更多的内存和 CPU 资源。因此建议在业务量还不大，比如只需要一两台服务器承载业务时，可以先不考虑将服务分得太细，但是在设计时可以先考虑日后业务量上去时如何进行微服务划分以及采用怎样的部署方案。

说到微服务，就要提一下 Spring Cloud。Spring Cloud 是基于 Spring Boot 的微服务开发框架，目前已经成为微服务事实上的标准，相对于国内出自阿里的服务治理框架 Dubbo，Spring Cloud 集成了更多组件，提供从服务治理到服务运维、监控等多个方面的解决方案。如果使用 Dubbo，可能需要自己去集成第三方组件，而使用 Spring Cloud，这些组件社区已经帮我们集成好了，知道怎么去使用就可以。近两年的 Service Mesh 大有取代 Spring Cloud 之意，但是目前成熟度还有待提高，比较有名的如 Istio，下一节中我们会讲到。如果项目采用 Java，那么微服务架构建议选择 Spring Cloud 或 Dubbo，其他语言也有一些开源的微服务框架，比如 Python 方面有 Nameko，本人曾经对这个框架的源码进行过剖析放到技术博客上，Go 社区有 Go-micro 等等不胜枚举，有兴趣了解和学习更多的童鞋，我建议你关注 GitHub 上的项目，在这里：awesome-microservices，里面列出了基于各种编程语言的在微服务架构方面的一些基础组件。

3.3 网格服务

网格服务也叫 Service Mesh，是近两年兴起的一种架构模式，大有取代现有的微服务架构之势。而实际上在我们看来，网格服务是对现有微服务架构的补充和加强，将微服务系统中一些基础又普遍性的功能特性标准化，然后集成到微服务系统的基础设施中去，变成微服务架构基础设施的一部分。比如 Kubernetes，可以看作是微服务应用架构的一种基础设施，网格服务将一些功能标准化后，可以随着 Kubernetes 一起部署，也就是只要部署了 Kubernetes 环境，这些标准化组件就已经在运行了，用户编写的微服务应用相应地会去掉这些已经标准化在 Kubernetes 中的功能，减轻微服务应用开发人员的工作量。这就好比 TCP/IP 标准确立前，应用开发人员如果进行网络编程，那么需要根据 TCP/IP 协议去处理数据，而当这些协议标准化后，对该协议的实现也成为操作系统或者基础库的一部分后，编写网络应用时你只需要调用这个库的接口去获取数据，而不需要自己去解析协议本身，解放了程序员并大大提高生产率了有木有。

在网格服务方面，Istio 名声比较大并且目前发展比较好，其背后有 IBM 和 Google 等这样的大厂背书也是原因之一。顺哥在这里只是列出这些概念，更多内容大家还是得去阅读官方文档，也可以去看 QCon 大会上关于这方面的讲座，PPT 都可以免费下载。国内在 Service Mesh 方面有尝试的有华为、新浪微博、网易等等，并且这些公司的相关技术人员都在各种大会上进行了分享，作为准公司技术负责人的你，需要具备自己去检索资料的能力，难道不是吗？我仿佛看到你在

点头了。

3.4 云原生应用

近几年云原生这个概念也炒得比较火热，云原生应用的意思就是这些应用天生就是为云环境准备的，这些应用利用了云计算平台的某些特征，充分发挥云平台的技术优势，成为云计算平台的原住民一般，少了水土不服的潜在隐患。我们为这些云原生应用采用的架构模式就叫云原生应用架构，这种架构也是基于云平台的，是因为云的出现而产生或者为了适应云基础设施而从其它架构模式演化过来的。

云原生应用架构的特性，包括比如微服务、十二因素应用程序、自助基础设施、基于 API 的协作、抗压性。关于云原生应用架构的更多内容，大家可以去看顺哥给的这些链接，都已经写得很好了，因此没有必要在这里再长篇累牍浪费彼此的时间。关于云原生架构，Pivotal 的技术产品经理 Matt Stine 曾经写过一本书，英文名全称是：《Migrating to Cloud-Native Application Architectures》，中文翻译版可以看这里：迁移到云原生应用架构。

在是否云化那一节中，我就说过目前云化的趋势不可逆反，因此建议大家在进行应用架构设计和开发时，多从云原生应用架构需要具备哪些特性的角度去考虑问题。

3.5 大数据技术

说到大数据，就不能不提 Hadoop，Hadoop 是雅虎发起的开源项目，已经成为离线大数据分析的重要项目，它的灵感源自于 Google 三驾马车之二的 GFS 和 MapReduce 论文。Hadoop 是一套用于在由通用硬件构建的大型集群上运行应用程序的框架，它实现了 Map/Reduce 编程范式，计算任务会被分割成小块运行在不同的节点上，它还提供了一款分布式文件系统（HDFS），类似于 Google 的 GFS。由于 Hadoop 处理的数据是存放在基于磁盘的文件系统上，使得它用来分析数据时效率不是那么的高，于是后来又出现了基于内存的 Spark，以及流式计算 Storm，因为基于内存，这些大数据开源框架能够更高效实时的分析处理数据。

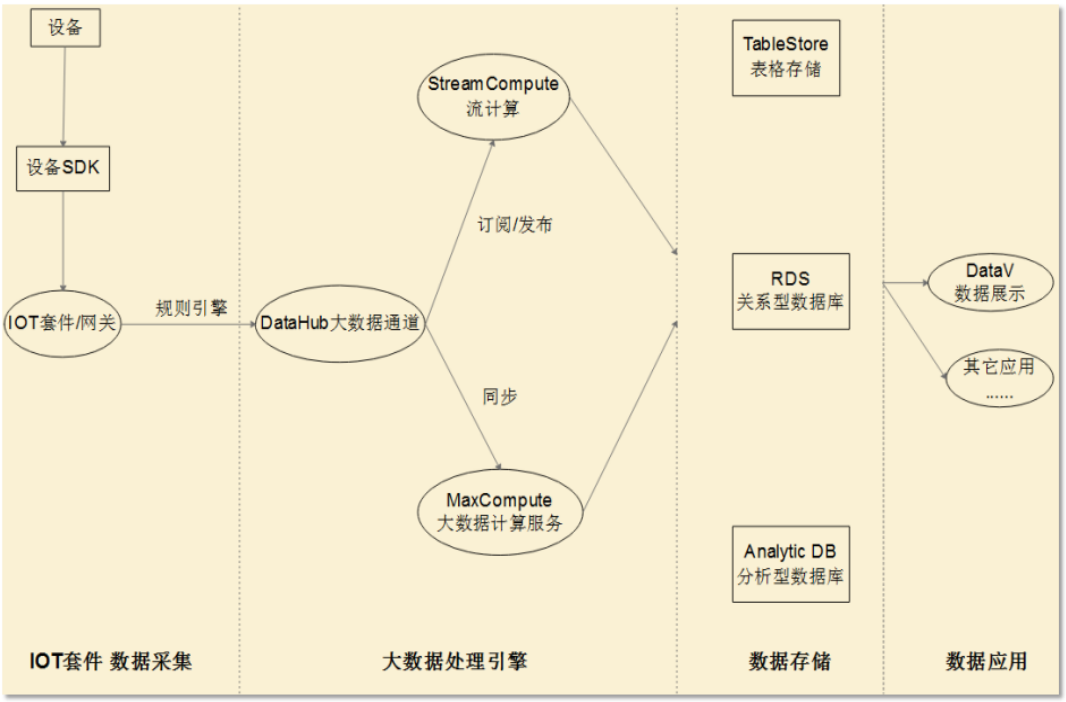
对于一个中小企业来说，有的研发团队只有几号人，老板让你搞时髦的大数据，这时候你该怎么办呢。去研究使用 Hadoop 还是 Spark 还是 Storm，然后跟老板汇报说要买多少台服务器来搭建大数据系统，需要多少人进行系统运维等等吗？这样的话，估计老板听了你的预算，还是会愁眉不展，没想到搞大数据那么费钱啊。是的，顺哥认为一般的中小企业根本没有能力组建大数据团队，除非是那种技术驱动型的公司，或者创业方向就是大数据产品的，普通的中小企业尤其是传统的企业想进行互联网+转型的，一般不具备自建大数据团队的技术和资金实力。

关于大数据，有一个广为流传的幽默段子是这么说的：Big data is like teenage sex: everyone talks about it, nobody really knows how to do

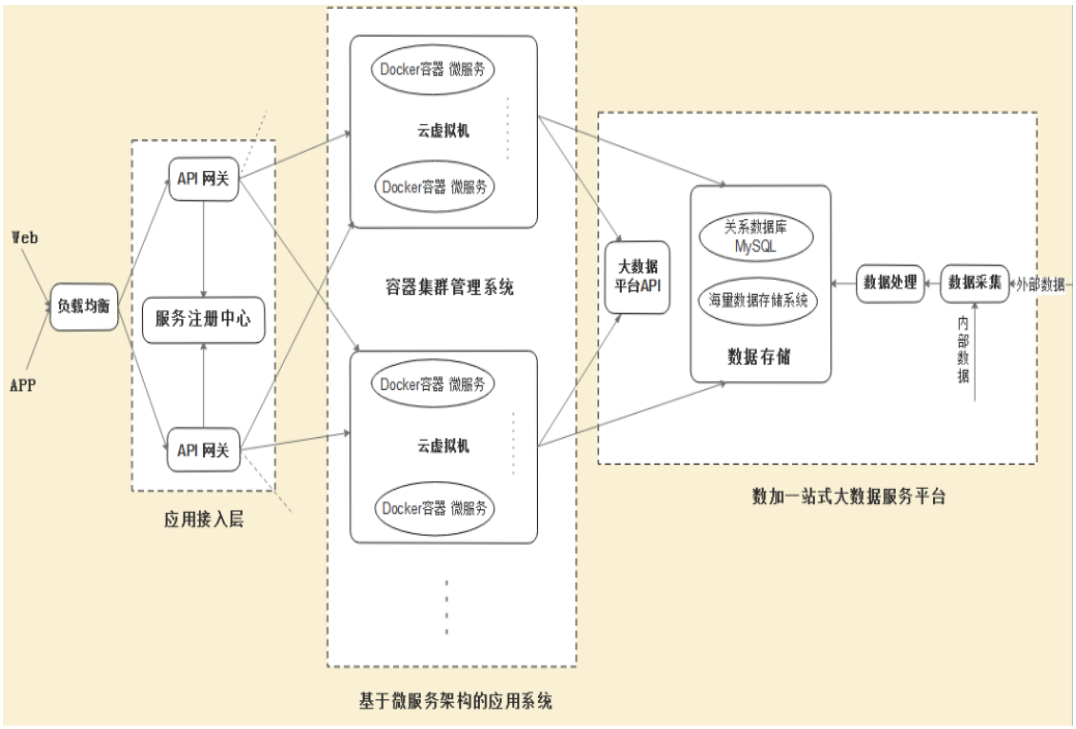
it, everyone thinks everyone else is doing it, so everyone claims they are doing it.【大数据就像青少年性爱：每个人都在谈论它，但没人知道怎么做，每个人都以为别人正在做，因此所有人都声称自己也在做】

实际情况也差不多如此，因此，顺哥建议，作为中小企业的技术负责人，你这时候应该将目光转向大的云计算平台，这些云平台一般都有大数据组件和解决方案，只需要半个小时，你就能购买到一个可用于大数据分析的计算和存储集群，而这些成本我相信不会比你自建的成本高。因此我们个人认为，一般的中小企业，能充分利用云平台提供的各种产品组件构建自己的研发环境，是非常关键的。

顺哥之前所就职的公司基本都是使用阿里云的产品，阿里云的产品线丰富是一个原因，还有就是大平台的品牌可信度一般会比较高，因此阿里云也成为很多公司采购云产品时的首选。下面是本人几年前在使用阿里云产品进行物联网项目的架构设计时画的一张图，仅供参考。图中的各种组件都是直接从阿里云购买，只需要你懂得如何去搭配使用，而要懂得如何搭配使用，进一步需要你先对每个组件的功能和使用场景有一个大概的了解。



从上面这张图看，如果我们搭建的是一个物联网平台，那么可以利用阿里云的这些组件快速搭建数据平台，包括数据采集、分析、存储和展示，当然，更进一步的分析可能需要开发人员去编写应用服务来进行，也就是对应到上图中右边的“其它应用”所表示的内容。那么如果把这部分展开，我们还可以得到下面这样一张图，是基于微服务架构的一个应用系统，这个应用系统将对数据进行处理，然后通过大数据 API 或者直接写入等方式将数据分析结果存储到大数据存储系统，这个存储系统就是阿里云一站式大数据处理平台数加，之前也叫作 ODPS(Open Data Processing Service)，即开放数据处理服务。



这张架构图仅供你参考，先不做过多解释。

3.6 人工智能

人工智能(AI, Artificial Intelligence)在未来将成为生产力，彻底地改变这个世界，虽然这项技术几十年前就有人研究，但随着这些年深度学习领域的迅猛发展，AI 已经走进了普通人的生活。各种人脸识别、语音识别设备也层出不穷，在这里不得不提的就是 Google 研发的 AlphaGo、微软小冰、百度大脑、阿里云 ET 等等，是这个领域非常有代表性的产品。

目前来看，AI 已经不是什么炒作了，而是实实在在地改变一些东西，所以我认为，无论是现在还是在遥远的未来(唱起来呀)，AI 会深

刻地改变这个世界是毫无疑问的。有人担心人工智能会取代人类，会威胁人类，而在顺哥看来也许是杞人忧天吧，人类有必要研发一个东西来打败自己吗？当然，极端的情况下，就是掌握这个领域精尖技术的一部分人是坏人，他们不计后果地使用这些技术，那有可能会给人来带来灾难，但人类可以未雨绸缪，建立一个安全稳健的社会架构，防止这种情况的发生，把它扼死在襁褓中，这也是必须的。你想想，AI 的目的是什么呢？应该是为人类服务的吧，而不是取代人类。

本人在人工智能方面没有研究，但也零星学过一些，从初学者的角度给想入门的同学推荐一些学习资料吧。首先是本人博客的文章里边的一些链接，这些文章相对通俗易懂，在这里：机器学习资料，还有就是大名鼎鼎的吴恩达的课程，虽然本人没有看过，但如果你去网上找资料的话，相信很多人都会给你推荐他的课程，关于他的经历网上也有很多，爱八卦的请自行前往搜索。GitHub 上也有相当多的学习资料，随便搜一下就能找到，搜索资料是程序员的必备技能，善于搜索资料使你能够更快地成长。

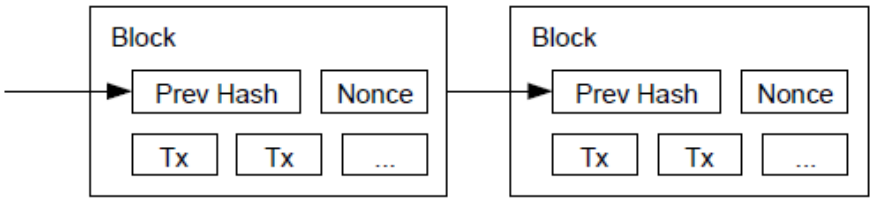
很多同学看到哪里火热就想往那个方向涌，因此会问到，既然人工智能这么有前途，我是不是应该赶紧去学习呢？在顺哥看来，人工智能的学习需要比较好的数学基础，人工智能是一个专业性非常强的领域，一般来说，这方面的研究留给那些专注于这方面的公司，比如商汤科技、旷视科技等等，大部分的从业者，只需要了解这些技术发展到大致哪个阶段，然后会将这些技术集成运用到自己公司的产品里面就够了。当然，如果你意志坚定，对人工智能技术又特别感兴趣，

那你去研究吧，或者加入这方面的独角兽公司都可以，像你这种骨骼清奇的人士，做什么别人拦都拦不住的，当然你也是对的。如果只是听说这方面火，能赚钱就跟风去学习，你还是先歇歇，然后该干嘛干嘛去吧。

顺便推荐一下李开复老师的书，叫《人工智能》，而且开复老师还出了一系列视频，叫《想象视频》，本人都没来得及看。

3.7 区块链

区块链的数据以区块的形式存在而得名，有人调侃说它是史上最慢的分布式数据库，但区块链技术在这两年发展得如火如荼，著名的以太坊就是区块链应用的开发平台，根据 CSDN 的调查，基于以太坊开发的人员数量在迅速增加。大家熟悉的比特币就是区块链技术的一种应用，比特币区块的存储形式如下图：



上面图中，每个 Block 就是一个区块，存放的是交易数据相关信息比如时间戳，并且有个指针指向了前一个区块，其实这个指针存放的是前一个区块的哈希值，通过加密算法来生成。如果有人修改了某

个区块的值，那么其哈希值也会被改变，指向它的区块就要更新哈希值，这样会引起连锁反应，使得后面所有的区块信息都需要修改。又因为有基于工作量证明的共识机制，超过 50%的算力都同意修改一个区块，这个区块的修改才能成功，因此区块链的不可篡改性就是因为篡改成本极高而得名，因为你要掌握 50%以上的算力，在这种分布式网络上掌握 50%的算力是极为困难的。

综上，区块链技术的特点是去中心化、不可篡改，分布式保证了去中心化，数字签名保证不能伪造交易数据，工作量证明保证区块的信息不被篡改，最终保证区块链每一条交易数据都是可信的。需要更深入了解的，建议去读一下比特币的设计白皮书。

八卦时间到了。据说著名的天使投资人徐小平曾在微信群要求他所投资的公司全面拥抱区块链，要毫不犹豫地。在他看来区块链是一种技术革命，会彻底改变这个世界的生产关系。后来都怎么样了，我们不得而知。不过如果你现在看到有人要发币了，建议你提高警惕。技术是中立的，区块链是无辜的，只是很多人利用这种技术和噱头来圈钱，等玩不下去了，留给用户只剩一个公告：再见，我们跑路了！

区块链技术目前还没有到需要选择哪种的时候，不过基于以太坊开发的人数应该是最多的，可以了解一下以太坊的开发生态。对于真正想学习区块链技术的同学，顺哥在这里又不厌其烦的推荐，推荐你看看 GitHub 这个系列：《awesome-blockchain-cn》。还有陈皓在极客时间专栏里面关于区块链的几篇文章，他发表了自己对区块链技术及其相关行业的诸多个人看法，看完你可能也会觉得他的思想还是相当

深邃的。

4. 技术架构设计

设计一个中大型后台系统的技术架构时，这个技术架构需要具备怎样的一些特质，可能会用到哪些云产品或组件，这一章我将根据自己的经验和所思所想，逐步给你大部分的答案，不能保证面面俱到，错漏之处也在所难免，仅供参考，还望海涵。

4.1 可扩展架构

可扩展性对现今的软件系统来说，怎么强调都不过分，因为如果系统不能扩展，那意味着业务量上来时可能随时面临重新整一套系统的风险，而当你弄出来一套新系统时，原有的客户早已经离你而去了。从广义上来说，可扩展包括两个维度，分别为纵向扩展和横向扩展，但我们一般强调更多的往往是横向扩展。纵向扩展很简单，比如一台服务器内存不够用了，我们再买几块内存条插上去，使单个服务器能承受更多的业务负载。横向扩展是原来的服务器不需要添加硬件部件比如内存条，而是添加一台服务器，由一台服务器变成两台，再有两台变成三台等等。

那为什么更应该强调横向扩展呢？聪明的你估计已经想到了，因为纵向扩展很快就遇到瓶颈了，你不可能一直给旧机器添加内存条

吧，况且服务器的内存槽也有限制，而添加新的服务器，只要购买新服务器并且分布式集群软件能够支持这么多台服务器就行。目前我们说的貌似都是硬件方面的横向扩展，而软件方面的横向扩展是跟系统架构和应用架构有关。也就是你添加新服务器了，分布式集群软件能够将这些服务器联合起来提供物理资源，公司开发人员编写的业务应用也能充分利用集群的资源为客户提供业务服务。这么说来，硬件、分布式集群软件、应用服务这三个因素都具有可横向扩展性，整个系统的可扩展性才有可能被称之为高可扩展性。

问题又来了，这三个因素又是如何才能扩展呢？硬件就不用说了吧，一个字：买，但是硬件的增加又需要分布式集群软件的支持，集群软件呢，需要公司自己研发或者鼓捣开源的东西，使得集群软件能够支持硬件设备的不断增加，比如阿里云的飞天集群，很早以前据说是突破 5000 台服务器，因此集群软件也不是可以支撑硬件的无限扩展的。最后到应用服务，因为底层环境已经可以扩展了，应用运行的模式必须也能随着集群资源的增多而横向扩展，比如一个应用原来启动一个容器来运行，现在可以启动多个容器来运行多个副本，然后还要保证各个端口使用没有冲突，整个系统还是一个完整的可对外正常提供服务的系统。对于非云计算厂商的中小企业来说，一般主要考虑应用层的扩展问题，而前面两个维度的扩展一般云厂商都已经解决了，只是需要我们去了解如何使用底层资源以及使用过程中有哪些限制。

对可扩展应用架构来说，还有一个非常重要的思想，就是尽量将

所有应用设计成无状态的，有状态的内容都存储到数据库或者其它第三方存储空间，这样在迁移应用时才不会引发状态不一致的问题。典型的例子就是用户 session 问题，如果 session 保存在内存中，用户第一次登录时被被负载均衡器转发请求到某台服务器，session 就被保存在这台服务器的内存中，后面访问时可能就被转发到其它服务器上，这时候从内存就获取不到用户 session 了，就会出现将用户重定向到登录界面的诡异现象。在这里，session 就是状态数据，应该存储到数据库或者其它共享存储空间中。当然，我们也可以设置用户亲和性，比如 IP Hash 方式，将同一个用户的请求始终转发到同一台服务器，这种方法属于负载均衡器的一项功能，这里就不展开讨论了。

4.2 高可用架构

高可用架构要求服务不中断或者中断频率和时长都在一个很小的范围内，因此就涉及到可用性的度量。比如我们说一个星期 7 天，总时间按秒算的话是 24*7*60*60=604800 秒，在这么一个时间段内，如果服务只发生一次中断，并且中断时间为 10 秒以内，那么我们可能会把这个架构称为高可用架构。当然这里只是举个例子，业界更常用的衡量方法是 SLA(Service Level Agreement)，即服务等级协议，一般以百分数表示，按全年来算。很多服务商都声称自己的服务 SLA 为多少个 9，比如五个 9，那么可用性为 99.999%，这样全年时间乘以可用性百分比，即可得到该厂商承诺的一年内服务可持续提供的时间

长度。我们可以算一下在五个 9 情况下，一年时间内服务中断时间为 24*365*60*(1-99.999%)=5.256 分钟，如果一年内服务中断时间到了 6 分钟以上，我们就可以说改服务厂商违反了 SLA 协议。

那么如何进行高可用性架构设计呢？我们应该马上想到避免单点故障(Single Point Of Failure)，具备高可用性的系统中不允许单点存在，每个组件都应该有冗余，因此一般一个组件会被部署到多个节点上，节点之间通过心跳(HeartBeat)之类的方法来感知对方，整个系统是一个分布式集群系统。冗余具体来说比如物理服务器有冗余、磁盘存储有冗余、应用服务有冗余，这样在某个组件崩溃或者失效时，另外的服务可以继续提供服务，当然如果冗余部分也崩溃时，那服务就无法提供的了，比如机房的大片断电。我们说的冗余其实也是相对的，绝对的冗余不存在，世界这么大什么情况都有可能发生对不对，这也是我们上面提到的 SLA 存在的部分原因吧，因此高可用性架构是在系统中消除了单点，每个组件都有冗余的情况下的尽力而为的系统和应用架构。

还有就是系统容量的设计，其实也就是系统的可扩展性，因此这么来看的话，高可用性的系统一般来说也是可扩展的，当请求激增达到容量上限或接近某个值时，可以自动进行扩展，这样才能保证系统的高可用性。

这里还要介绍一下分布式领域的 CAP 理论，CAP 即 Consistency(一致性)、Availability(可用性)、Partition tolerance(分区可容忍性)。CAP 理论认为，在分布式系统中，因为网络故障不可避免，系统要么选择

可用性，要么选择一致性。如果你选择了一致性，因为网络故障已经分区了，那么系统不可用，如果你选择可用性，那么也因为网络故障导致分区已经发生，继续使用系统的话会导致数据无法同步而出现不一致的状况。CAP 理论经常被人误解为要么选择 C 要么选择 A，实际上这是在网络故障出现的时候，如果网络正常，CA 可以同时得到满足。在实际中我们应该根据业务需求来选择系统是偏向 C 还是偏向 A，然后在出现相应的状况时根据我们的选择给用户返回合理的结果。

4.3 高并发架构

高并发架构在顺哥看来跟高性能架构差不太多，只是强调了并发数，即这种架构需要在请求数量非常高的情况下依然坚挺。那么在考虑高并发架构时，同时要考虑可扩展性，因为高并发系统往往面临着随时扩展的要求。

为了应对高并发请求，在架构时应该思考每个请求在整个系统中的调用链路，尽量将请求拦截在上游，将数据存放在离用户最近的地方(比如浏览器缓存)。比如我们常见的秒杀系统是典型的高并发架构，那么在一件商品库存量只有几百的情况下，将几万个请求下放到后台数据库是没有必要而且是对计算资源的极大浪费。这时候我们可以把前端的几百个请求放到队列(消息队列是系统解耦的利器)里面，然后另外一个消息处理系统对这些请求进行后续处理(也就是异步处理)，其它多余的几万个请求因为已经没有商品库存了，没有必要放进队

列，直接返回秒杀失败。

在进行了以上处理以后，如果系统还是无法处理这么多请求，那么可能是数据库的读写瓶颈，可以进行数据库设计上的优化，可以加入缓存系统，如 memcached 或者 redis，或者确实到了需要添加服务器资源的时候了，也就是可扩展架构需要做的事情，因此，高并发架构往往首先是一个可扩展架构。当然，实际情况可能不是这么简单，这里只举个例子，各位同学在遇到问题时要根据具体情况去分析瓶颈所在，寻找恰当的应对策略。

4.4 高性能架构

在聊高性能架构之前，首先我们得明白什么是高性能。在我们看来，高性能主要体现在这两个方面：

1) 吞吐量大，单位时间(一般是每秒)内可以处理的请求数；

2) 系统延迟低，一个请求从发起请求到获取回复数据的时间足够短；

为了更形象一点，我们就拿 API 接口为例吧，比如一个远程接口 getUserName，满足同时十万个调用请求，并且全部在 0.5 秒内完成数据返回，我们就说这个接口的性能达到了一定的水准，具体是否高性能要看各个公司对自己业务的要求可能有不同的看法。

上面的吞吐量和系统延迟，在服务器资源一定的情况下会相互制约，因此在我们看来，高性能架构必须在某个维度上达到一定高度时，另一个维度同时保持在较高的水准。简单来说吧，如果吞吐量每秒一百万，系统延迟达到一分钟，那么客户端在请求一分钟之后才有数据返回，一般来说这种系统不会有正常人使用，这样看来这个高吞吐量并没有什么卵用。同样的道理，如果系统延迟在 0.1 秒，但吞吐量不过百，也许能满足特定场景需求，比如请求数很小的情况，但我们一般不会说这是一个高性能架构的系统，因为一旦请求量上来，系统不是挂掉就是延迟增加了。因此，需要以上两个因素同时达到一定高度才能称之为高性能。

前面说的其实只是服务器端接口的高性能，作为一个系统来说，我们还要包括前端的内容，因此高性能架构需要从前端到后端进行整体设计，在后端接口能快速响应前端请求的基础上，还要进行更精细的数据传输设计，比如返回怎样的数据信息在满足前端数据需求的同时可以减少流量消耗，怎样的数据结构有利于前端快速解析，因为这些都会影响到前端的用户体验。前端是用户直接触及的部分，也就是给到用户的最终产品，因此良好的用户体验非常有必要，而后端对普通用户来说，是无法直接感知的。

4.5 数据库架构

很难想象现今哪个系统没有用到数据库，因此数据库的重要性无

需多言。在数据库架构设计过程中，一般从以下几个维度去考虑。

1) 可用性，也就是高可用的问题，需要冗余机制；

2) 可扩展性，数据量暴增时，如何扩展数据库以保证满足需求；

3) 一致性，当使用分布式数据库时，主从数据库的同步方式是怎样的；

4) 性能，在前面几个维度都考虑好的情况下，数据库的读写性能如何，比如可能需要读写分离等方案来满足性能需求；

5) 灾备方案，如何进行数据备份和恢复，极其重要，数据可能是一个公司的命根子，是生产资料；

架构做好了，就需要设计数据结构，也就是数据库表及其之间的关联。可以说一个系统的数据结构反应了这个系统的本质属性，因此设计一个能真实反应系统中各种实体及其之间关系的数据结构是应用系统健壮的关键因素之一。业务应用都是基于这些数据结构进行各种操作，因此需要数据结构的设计者对业务流程有非常清晰的理解，然后进行抽象，将这些实体映射到系统的数据结构中去。

数据结构设计好了，还要考虑索引的问题，索引的用处不需我多说了，具体怎么使用索引，网上也有不少资料，这里本人推荐极客时间上丁奇老师的 MySQL 专栏，还有数据库大牛何登成的技术博客上相关文章。

4.6 安全架构

试读版只能看到这里哦.....

5. 后记

5.1 缘续

凡事有始有终会更妥当些，既然有前言，何不来个后记。正如前言里说的，这本小书不会带你进入技术细节，而是从较为宏观的技术角度和方向阐述个人的理解和观点。从目标读者的定位来看，我想有志于成为中小企业技术领导者的你，此刻也不需要我来做技术细节上的分享和指导，对吧？当然，我今后的技术文章中，不排除有对某些开源项目进行源码剖析的，要看我有没有动力去写了(你的鼓励，我的动力)！

这本小书其实比较适合在架构方面有些经验的技术人员，对于行业新人，你可能看完了还是觉得云里雾里，但你不必焦虑，这是正常现象，很多事情需要你自己去经历才能体会深刻。李嘉诚曾经说过这么一句话：“世界上最浪费时间的事就是给年轻人讲经验，讲一万句不如你自己摔一跤，眼泪教你做人，后悔帮你成长，疼痛才是最好的老师。人生该走的弯路，其实一米都少不了”。那么，对于新人来说，本书对你的意义又是什么呢？我认为它能让你提前知道了在到达你

目标的路上，大致都会有哪些坑等着你，让你有目的地踩，踩得坦然，踩得有效率。

如果你看完了，觉得值，我很欣慰，同时欢迎你推荐给身边需要的朋友；如果觉得不值呢，你就算是我的赞助商，慷慨赞助我一中杯星巴克咖啡吧，也好让我今后漫漫长夜中码字时每每想到你的赞举，心底里多涌起一丝温情，谢谢。无论如何，欢迎扫描下面的二维码加我微信或者关注公众号并加入社群，你可以私信或者在“技术人成长”社群里给我提些问题或建议，我争取拿出时间给你解惑，同时帮助我自己进步，后会有期，朋友们！

微信号

【技术人成长】 公众号