架构师

ARCHITECT

6月刊

推荐文章

为什么高级程序员写的代码
都是傻瓜式的?

Geekbang. 极客邦科技 | InfoQ

CONTENTS / 目录

架构师

2019 年 6 月刊

本期主编 田晓旭

流程编辑 丁晓昀

发行人 霍泰稳

提供反馈 feedback@geekbang.com

商务合作 hezuo@geekbang.org

内容合作 editors@geekbang.com

卷首语
人工智能的成功取决于数据

作者 MongoDB 中文社区主席 唐建法

最近，清华大学教授、中国科学院张钹院士在接受经济观察报的时候提到：“基于深度学习的人工智能在技术上已经触及天花板， AI 奇迹短期难以再现。”这无疑为许许多多仍然以 AI 为主要业务亮点的创业公司敲响了警钟。

当下，人工智能已经成为一个流行语。从语音识别到自动客服，从人脸图像识别到自动驾驶，不可否认，AI 已经实实在在出现在我们的生活中了。AI 创业公司更是热情甚高，动辄即称 AI 驱动，就连世界级大牌公司 Oracle 也不例外，前段时间的 Oracle Autonomous Database，按照 Larry 的说法，通过 AI 和 ML 技术，做到了“Totally Automated，Self-Driving”（全部自动，自运行），不再需要人类来管理或者调优。我觉得要么是 Larry 在骗我，要么最近作为数据库顾问加入 Oracle 的朋友在骗我——如果 Larry 说实话的话，为什么 Oracle 还要继续大量招聘为客户做数据库维护性能调优的顾问？

像大数据一样，对人工智能的炒作也导致了一种趋势，即每个供应商都声称在技术、解决方案或产品中利用了它，都说要改变、替代人类，从而造成了一个良莠不齐、极度混淆的技术怪圈。

事实上，就像大数据技术不能解决所有的企业数据问题一样，AI 也

不能用来解决所有问题。如果想了解 AI 最应该用在何处，并且最容易成功，我们必须先了解 AI 的真正含义。

AI 或机器学习是指一组广泛的算法，如果训练得当，它们可以解决一组特定的问题。把机器学习算法集成到产品中其实是微不足道的——有大量公开的算法可以用，但有效地使用数据来训练算法并执行任务却并非那么简单。事实上，一个 AI 项目从开始施行到初有成效，多达 80% 的精力都是用在数据准备上的。

当有大量丰富的数据可用时，AI 效果是最佳的。数据量越大，覆盖维度越多，算法学习和调整预测分析的速度就越快。根据行业预测，在 2018 年，人工智能的最大限制——高质量数据，将变得更加明显。成功的机器学习取决于大型和广泛的数据集，以及对这些数据的有效管理。

谁能够获得最优质最全面的原始数据，谁掌握最成熟的数据处理与加工技术，才是未来真正 AI 项目的核心能力。所以，对很多不明真相的技术人来说，与其盲目的追捧 AI，不如实实在在地掌握精通数据处理的技术，这才是一条不会受到泡沫破裂而影响的务实之路。

主办方 Geekbang. 极客邦科技 InfoQ

ArchSummit 全球架构师峰会

会议：07.12-13 | 培训：07.14-15

地址：深圳·大中华喜来登酒店

干货日程抢先看

时间	议题	讲师
7月12日	唯品会自研服务化基础体系建设之路	薛珂 唯品会 / 基础架构负责人
	百度春晚极限压力场景下的运维解决方案	陈曦洋 百度 / 运维部 资深运维工程师
	支付宝舆情监控智能化应用实践	刘凡（梵鸿） 蚂蚁金服 / 技术专家
	更小和更大--美团智能调度演进之路	王圣尧 美团 / 算法专家
	下一代金融基础设施的建立	黄连金 中国移动通信 / 研究员联合会 区块链专委会首席安全专家
7月13日	淘宝应用架构升级--反应式架构的探索与实践	许泽彬 阿里巴巴 / 淘宝技术专家
	大数据创业公司 ToB 技术实践：挑战与破局之道	代其锋 百分点 / 资深架构师
	大数据自助平台的思考与建设	成峰 Grab / Data Engineering Lead
	如何弥合 Spark Datasets 和 DataFrames 之间的性能差距？	蔡東邦(DB Tsai) Apple / Staff Software Engineer & Apache Spark PMC
	去哪儿网跨端小程序开发实践谈	钟钦成（司徒正美） 去哪儿 / 架构师
	Lazada 技术选型和团队搭建的思考和实践	陈思淼（云动） 阿里巴巴 / 资深技术专家，前Lazada CTO
	阿里巴巴中台技术架构实践与思考	谢纯良 阿里云/中间件架构总监
	Shopee 数据事件中心的设计和实现	林锋 Shopee / Engineering & Technology，技术平台团队负责人
	腾讯NOW直播前端工程效率体系实践	程柳锋 腾讯 / 高级工程师、IVWEB团队成员
	细说Tech Leader在开发团队的核心职责	孔凡勇（云狄） 阿里巴巴 / 高级技术专家

联/系/我/们

电话：17326843116(微信同号)

扫描二维即可获得大会详细议程>>

谷歌中止与华为业务往来，华为自研手机 OS 也要一夜“转正”？

作者 赵钰莹 ，张晓楠

一连数天，华为海思事件持续发酵。今天根据路透社最新报道，谷歌已经暂停与华为的业务，这意味着华为只能使用开源版本的安卓系统，无法访问来自谷歌的专有应用程序和服务。晚些时候，彭博社报道称，英特尔、高通、赛灵思和博通在内的多家芯片制造商告知其员工停止对华为供货。谷歌禁运，安卓告急。随着手机操作系统的至暗时刻到来，华为的 B 计划在哪里？前路漫漫，是否还有更多的困难在等着华为？

事件回溯

根据路透社的最新报道，谷歌已经暂停与华为的业务。知情人士表示：“这意味着华为只能使用安卓的公开版本，无法访问来自谷歌的专有应用程序和服务”，包括但不限于 Play Store、Gmail 和 YouTube。此举可能会阻碍华为在中国境外的智能手机业务，预计对中国市场的影响微乎其微。因为大多数谷歌移动应用程序在中国被禁止，国内竞争对手可提供替代应用程序，华为的第二大市场——欧洲业务可能受到重创。

路透社称，谷歌发言人认为，公司“正在遵从相关指令并研判影响”，但拒绝提供相关细节。今年 3 月份，华为轮值董事长徐直军（Eric Xu）在接受路透社采访时称：“无论发生什么，安卓社区都没有阻止任何公司访问其开源许可证的任何合法权利”。

从芯片到操作系统，华为是否还有更多的 B 计划可以亮出？

华为手机操作系统研发史

曾经，谷歌与华为的合作关系颇为紧密。作为全球第一大移动操作系统，安卓的卡顿问题一直未能得到解决，华为决定使用 F2FS（Flash Friendly File System）替换原生的文件系统。

根据 CBG 软件部总裁王成录的描述：“2018 年，谷歌将 F2FS 吸收到安卓原生版本中，所有安卓厂商因此受益，对安卓生态是很有价值的贡献。至今，谷歌的自研手机 Pixel 3 也使用了 F2FS 文件系统。”

此外，华为工程师曾被发现在谷歌开源社区中提交了基于荣耀 Play 的 Fuchsia 测试代码，添加了对麒麟 970 平台的支持，已经能够将设备引导到 Fuchsia 的内核 Zircon，这被看作是谷歌 Fuchsia OS 的首发平台。

此时，谷歌深陷与 Oracle 的 Java 专利纠纷并遭遇了欧盟的反复审查，华为荣耀 Play 对 Fuchsia OS 的支持至少在一定程度上说明华为作为全球排名靠前的手机大厂对 Fuchsia OS 的开放态度。然而，好景不长，一纸禁令斩断华为与谷歌的过往与未来，华为在手机操作系统层面的 B 计划备受关注。

过去几年，华为在手机操作系统的硬件和软件层面均有所准备，即便是与谷歌一直处于“友好合作”的状态，余承东在接受采访的时候也表示：

> 我们已经准备好了自己的操作系统，一旦发生了我们不能够再使用这些（来自 Google 和微软的）操作系统的情况，我们就会做好启动 B 计划的准备。

众所周知，华为内部一直在研发麒麟 OS，虽后来有媒体报道称这款麒麟 OS 的真名是“鸿蒙”，并已对 Linux 进行了大量优化（已开源），用于华为手机中（安全部分），但这一报道的真实性难以探究，本文对此不做过多探讨。

一直以来，麒麟 OS 被认为是华为在手机操作系统层面的备用方案，对外曝光的信息不多，但仍然是舆论关注的焦点。此前，任正非曾说过：“如果说其他操作系统都给华为一个平等权利，那我们的操作系统是不需要的，为什么不可以用别人的优势呢？”。

此前国际市场研究公司 CCS Insight 曾预测：华为将在 2022 年发布移动操作系统是真实存在的。面对如今的这一局面，不少用户猜测：华为的麒麟 OS 系统恐怕要提前亮相。

虽然华为的麒麟 OS 可能已经做好准备，但是它的软件生态圈近乎没有，毕竟麒麟 OS 出现的时机不是智能机的普及初期，与其他各大厂商的软件兼容可能也存在磨合期。现在的麒麟 OS 只能作为备选方案，是华为战略意义上的备用手机操作系统。

在软件层面，王成录曾在心声社区发表题为《让软件成为华为手机硬实力》的文章，文中提及了不少华为对安卓系统软件层面的改造，比如前文提到的 F2FS 文件系统，该方案最初于 2016 年被搭载在运行 EMUI5.0 的 Mate 9 中，华为给这个解决安卓卡顿的方案取了一个简单易懂的名字，叫做“天生快一生快”，同时承诺消费者“18 个月不卡顿”。

为了解决手游对手机图形处理能力的较高要求，华为 GPU Turbo 全球联合研发团队推出 GPU Turbo ，提升游戏性能的同时降低功耗。在该技术的发布会现场，搭载 GPU Turbo 的荣耀 Play 在帧率、抖动率、掉帧、耗电等硬指标上都略胜友商手机。

三年多以前的 EMUI，基于海思 Kirin、高通、MTK 芯片的主干各不相同。从 EMUI5.0 版本立项开始，交付采用了全解决方案运作模式，将用户交互、OS（操作系统）、海思、通信协议、安卓原生多个模块纳入统一规划，同源设计、同源开发、同源测试。此外，内部多个团队经过长时间的研发和讨论，EMUI 最终实现了一个清晰可解耦的架构，让“抽屉式”替换相应的安卓组件成为可能。

在前不久的 P30 系列国行发布会上，华为宣布方舟编译器，通过架构级优化，显著提升性能，尤其是全程执行机器码，高效运行应用，彻底解决安卓应用“边解释边执行”造成的低效率。华为方面表示，方舟编译器可让系统操作流畅度提升 24％，系统响应速度提升 44％，第三方应用重新编译后流畅度可提升 60％ ！

曾经的未雨绸缪让华为在硬件层面拥有麒麟 OS，软件层面拥有 EMUI、方舟编译器等系列优化，这是华为多年的研发成果积累。如今，检验期逐渐临近，华为的手机终端未来值得期待。

手机操作系统的前车之鉴

对于华为手机操作系统的 B 计划是否成功，现在还不是下定论的时候。但是跟芯片一样，操作系统本就不是一个谁都能成功的领域。在 Android 和 iOS 之外，想要在”夹缝中生存“的手机操作系统并不少，比如塞班

（Symbian），黑莓（BlackBerry），英特尔与诺基亚合作的 Meego，三星 Tizen，微软 Windows Phone，Palm 的 WebOS 等等，只是大多以失败告终，这也是华为的前车之鉴吧。

Windows Phone 是微软于 2010 年 10 月 21 日正式发布的一款手机操作系统，伴随 2017 年微软宣布停止对 Windows Phone8.1 移动操作系统的支持，标志着 Windows Phone 的时代彻底终结，虽然此时依然有数以百万计的设备运行着 Windows Phone 系统。

三星 Tizen 系统自诞生起就命运多舛，在安卓呈现强势势头的 2012 年对外发布。为了怕影响销量，三星甚至都不敢在自己的旗舰机上尝试。不过虽然在手机领域以失败告终，但是 Tizen 可是全球最大的智能电视操作系统。

诺基亚手机的流行，让 Symbian 成为不少人第一次用到的智能手机系统。然而也正因为诺基亚缺乏软件生态建设能力，Symbian 最终被诺基亚抛弃，并在 2013 年初正式退出历史舞台。

除了一些备受关注的操作系统之外，可能还有这么一个不太受关注的操作系统 Firefox OS，它虽然是个失败的操作系统，但是其希望借助开放网络的力量颠覆应用生态的做法可圈可点。

在 Firefox OS 之后，当包括 Ubuntu Touch 和 Windows 10 Mobile 在内的多个“另类”智能手机平台也逐渐淡出人们的视线、Jolla 一直在为 Sailfish OS 苦苦挣扎后，又出现了一个新进者：KaiOS。这个新进者瞄向的是功能手机，目标是让功能手机变得更智能，目前全球已经有 8000 多万台设备在运行 KaiOS。虽然智能手机仍然是大多数人的未来，但 KaiOS 已经证明了功能手机也是可以引导大量用户加入互联网的，特别是如果你能够以低于 10 美元的价格把这些手机卖给消费者。

结束语

一纸禁令虽无法阻止华为继续使用开源安卓技术，但还是为国产手机操作系统的研发敲响了警钟。目前，国内具备完全自主研发能力的厂商十分有限，无论是硬件层面还是软件层面，大部分都是基于开源代码进行改进，但要知道即便是最为广泛使用的 Apache 产品，其基金会依旧在其官网表示：

> 美国的出口法律和法规适用于我们（Apache）的发行版，并且随着产品和技术再出口到不同的地方依旧保持有效。

Apache 方面对禁运的相关描述为：“除非经美国政府正式授权，否则 ASF 软件或技术不得直接或间接出口 / 再出口到受美国禁运或贸易制裁的任何目的地。”虽目前开源部分尚未受到影响，但国产自研技术需要更多“科技自立”的中国科技公司站出来并有所作为。如今，高通、英特尔、赛灵思在内的芯片制造商对其员工表示拒绝向华为供货，从芯片到操作系统，华为面临的压力愈来愈大，如若技术难以自立，何谈企业自立？

产业互联网起风了：这一次，腾讯又站上了风口

作者 小智，张晓楠

2019 年 5 月 21 日，腾讯全球数字生态大会在云南昆明举办。从去年 9 月底腾讯宣布成立云与智慧产业事业群（CSIG）至今，今天的腾讯全球数字生态大会是腾讯 CSIG 半年多来交出的第一张重要成绩单。互联网的下半场是产业互联网，也是下一个风口所在。继移动互联网红利以后，腾讯会引领又一个潮流吗？

产业互联网，起风了

2019 年 5 月 21 日，腾讯全球数字生态大会在彩云之南召开。云南省人民政府副省长陈舜在开场致辞上把自己有关”风“的观点饶有趣味的比喻为“521”定律：

> 风很重要，风口很重要，不知道什么时候风就来了。
>
> 如果互联网的上半场过去了，没你戏，下半场产业互联网的风，来了没有？

腾讯总裁刘炽平说：产业互联网不仅是风，还是热风，越吹越热。

过去这些年，随着互联网技术的发达普及，人类的生活方式从根本上被改变了。尤其是移动互联网的兴起，造就了大批应运而生的新兴独角兽公司，也让老牌的科技公司焕发了生机，移动互联网的大航海时代蔚为壮观。

腾讯是其中的一个缩影。在腾讯成立至今的时间里，中国网民人数从百万人增加到 11 亿人。腾讯公司也从几百 K 文件大小的 OICQ 起步，成长为拥有 10 亿级别月活产品、市值万亿元的巨型企业。

- “今天我们面临一个新问题，即从消费互联网到产业互联网的转变。”
- “产业互联网将为实体经济高质量发展提供历史机遇和技术条件，对实体经济产生全方位、深层次、革命性的影响。”
- “在此过程中，互联网公司要作为实体产业的数字化助手，帮助实体产业在各自的赛道上成长为世界冠军。”

2018 年以来，马化腾在多个场合发表过关于产业互联网的演讲。当移动互联网红利消失殆尽的观点渐趋主流时，马化腾指出：资本对于互联网产业的追逐是有周期性的，产业互联网的春天才刚开始。中国的创新红利还在，而且潜力巨大。

在今天的大会上，刘炽平提到：现代社会，数字生态与物理生态日渐趋同。从泛互联网生态到数字生态到实体产业、线下社会的结合，是大势所趋。未来互联网将不会是一个产业，而是各行各业的核心之一。

在这其中，刘炽平特别强调：“腾讯是帮助者，而不是颠覆者。”

对于腾讯 To B 业务来说，CSIG 是一个窗口，让腾讯各业务通过 CSIG 来帮助客户实现数字化转型和升级。不过在腾讯云与智慧产业事业群（CSIG）总裁汤道生看来，”企业数字化升级不是为了造概念，而是为了解决实际问题。“他认为：“产业互联网是研发、生产、组装、流通、服务全周期的概念，只有各个环节都完成数字化改造，打通整条价值链，才能实现产业的进化。”

近几年的政府报告里频繁提到过一些性质相近的词汇与言论：

- 2019 年，“打造工业互联网平台，拓展‘智能 +’，为制造业转型升级赋能。”；
- 2018 年，数字中国；
- 2017 年，数字经济；
- 2015 年，互联网 +。

提法各异，目标相同：希望在全球新一轮科技与产业革命中，抓住“信息技术（IT）”这个最大的变量，推动各个产业进行“数字化、网络化和智能化”的转型升级。

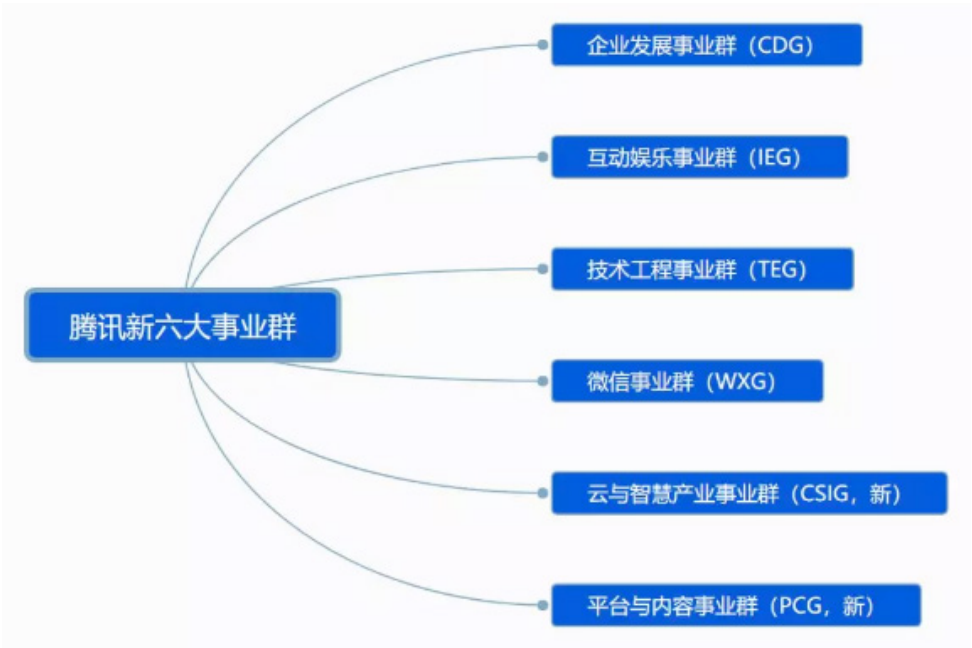
事实上，产业互联网虽然是个崭新的概念，但工业企业、互联网企业已经在这其中做出了不少的探索与创新。联想到近期的中美关系、华为困

境，不难看出转型到底有多难。此时腾讯的转型目标已经清晰明了：扎根消费互联网，拥抱产业互联网，连接一切。

产业互联网，会让腾讯保持领先，成为互联网下半场的领头羊吗？拭目以待。

CSIG，成果几何？

2018 年 9 月底，腾讯 6 年来首次调整架构，七大事业群变为六个，首次新增云与智慧产业事业群（CSIG）和平台与内容事业群（PCG）。



成立 CSIG 可视作腾讯向 B 端转型的开始，而首次从组织架构层面把云计算拎出来放在关键位置，不难看出腾讯云在未来腾讯业务发展中的重要地位与权重提升。

在过去的 20 年里，腾讯一直是一个 To C 能力极强，To B 能力稍弱的互联网巨头。靠着微信、QQ 的社交产品，依靠游戏行业的收入，腾讯的财报一直十分出彩。但对互联网巨头而言，To B or To C 一直不是选择题，答案从来都是我全都要，区别只是怎么做罢了。

对腾讯来说，云业务的重要性在近年来愈加凸显。根据腾讯 2019 年第一季度财报，金融与云业务持续增长，占比 25%，成为腾讯第二大收入来源。考虑到腾讯云计算布局较晚，这份成绩单可以称得上是优秀。

作为腾讯 To B 事业的窗口，在会上汤道生表示：CSIG 在过去的半年多时间里已经联结了众多 500 强客户，五大行中有四大行已经选择使用腾讯云。腾讯没有 To B 基因？ CSIG 不同意这个观点。

开源，有态度还要有成绩

在腾讯副总裁邱跃鹏看来，云的发展要跨过三个坎：规模效应的坎、产品价值的坎、产业升级的坎。在云的规模上，目前腾讯云全网服务器总数量达到 100 万台，是中国首家达到此规模的云计算厂商；峰值带宽达到 100T，也是中国第一家达到这一峰值带宽的云计算厂商。

在开发者方面，”云开发“小程序是传统开发者生态的升级，目前开发者小程序账号已经达到 16 万个。

开源能力也是对云计算厂商能力的一个重要考量：

- 凭借向 KVM 贡献的 patch 数，腾讯云连续两年登上 KVM 开源贡献排行榜，成为国内贡献度最高的公有云厂商；
- 腾讯云运行着国内云厂商中规模最大的容器集群，这一容器集群兼容云原生开源项目 API，可以大幅降低用户上云、使用云的门槛；
- 腾讯云在 TBDS、EMR、Sparkling 以及 Ti 的基础上，集成和优化了诸多开源技术，全面兼容开源 API，已经成为业界最开放的 AI 大数据平台。

这里还有一组腾讯开源的整体数字：截至 6 月 19 日，腾讯已在 Github 上发布 73 个开源项目，包含微信、腾讯云、腾讯游戏、腾讯 AI、腾讯安全等相关领域，腾讯发起的开源项目累计在 Github 获得 221435 Star 数。诸如腾讯云 T stack、微信开源系列（如 WeUI）、TARS 等，都是腾讯开源的模范案例。官网及项目详情地址。

没有 CTO，还有技术委员会

2014 年，腾讯 CTO 张志东退休后，CTO 一职空缺至今。

张志东任 CTO 时，一手打造了 QQ 的技术架构，用户数从百万级飙升到亿级时，架构仍旧适用。但在其退休后的 5 年时间里，腾讯 CTO 一职一直处于空缺状态。

坊间对于腾讯技术建设的诟病由来已久。从《腾讯当下的技术建设是否落后于同体量的公司？》到《张志东归位鹅厂如何》，折射出来的是腾

讯缺少技术上的领军人物，直接导致技术没有足够的话语权、日渐式微。研发体系落后、技术领袖出走、研究人才收集癖、企业 IT 权力过大、不投资技术，这些都是腾讯在技术上曾经遭到的批评与指责。

去年腾讯宣布调整组织架构时，一条腾讯宣布成立技术委员会的消息引起了我们的注意。2019 年 1 月 4 日，腾讯技术委员会正式成立。在 CTO 空缺多年后，这个技术委员会的重要性不言而喻。事实上，从牵头人、参与者的身份来看，腾讯更是向外界释放了一个鲜明的信号：互联网的下半场，腾讯希望以技术说话！

腾讯技术委员会由技术工程事业群（TEG）总裁卢山、云与智慧产业事业群（CSIG）总裁汤道生两名腾讯总办成员牵头，几大事业群的技术负责人悉数进入技术委员会决策圈。技术委员会同时下设「开源协同」和「自研上云」项目组，发力内部代码的开源和协同，并推动业务在云上全面整合。

汤道生在专访中也曾表示，他负责的云与智慧产业事业群与卢山负责的技术工程事业群在技术委员会里有非常紧密的合作关系。

> 我们会加大力度推动技术共享，而且未来资源的分配将基于腾讯云来展开，以服务模式去做研发，是我们未来去推动的一个重要的点。腾讯云将作为一个窗口，我们把腾讯很多业务能力给到 B 端，涉及到的团队非常多。

在成立技术委员会以后，腾讯内部更多“自上而下”地推动多个技术团队一起去构建共用的技术模块，更多去做开源的贡献，更积极地优化上云，也把更多内部组件与框架开放给云上的客户。除此之外，还有研发工具的优化等等。汤道生谈到过去腾讯对安全特别敏感，为了保护 IP、网络的安全，出过一些加强安全防护的措施，导致研发人员在研发环境里面会多出一些步骤、多一些操作的门槛。所以技术委员会也在推动怎么让研发更有效率，把便捷性跟安全都照顾到。

成立技术委员会，对于历史积累的技术问题是一件好事，但未必能取得立竿见影的效果。事实上，这个组织的成立，更像是激发企业文化中的工程师文化传统。以技术为导向的企业文化，更容易为开发人员争取到相关的资源和权益。过去腾讯在这个方面并不够重视，接下来产业互联网的战略重心中，核心驱动力是技术，研发人员的重要性不言而喻。

针对这种变化，卢山就曾对内表示，希望今后的新员工会觉得，在腾

讯做开发是很幸福的东西，能学到很多东西，能看到很多优秀的代码。“人们不仅谈论硅谷文化，也谈论我们中国开发人员的代码文化。”

而在公有云上开发，也同样对腾讯员工的职业生涯意义深远：“我们希望员工在腾讯这几年所学习的技能同样在外面有用的，而不是只在腾讯有用。”汤道生说。

结束语

腾讯一直在期望成为一家受人尊重的科技公司，最近更是修改了自己的愿景和使命：科技向善。什么样的企业是受人尊重的企业呢？在中国，达到这样的标准并不容易。但达到受人尊重的一个前提是：尊重他人。调整企业组织架构，成立更高层面的技术委员会，就是认识到技术的重要性，并真正尊重技术的开始。这样的腾讯，也会收获来自腾讯内部、外部的开发人员的尊重。

波澜壮阔的互联网下半场已经来临，在产业互联网的大潮里，腾讯和腾讯的开发人员们能做出怎样的成绩呢？真是充满期待啊。

阿里将 Transformer 用于淘宝电商推荐，效果优于 DIN 和谷歌 WDL

作者 吴少杰

基于深度学习的方法已经广泛用于工业推荐系统（RSs）。以前的工作通常采用嵌入（Embedding）和 MLP 范式：原始特征嵌入到低维向量中，然后将其输入 MLP 以获得最终的推荐结果。 然而，这些工作中的大多数只是连接不同的特征，忽略了用户行为的连续性。近日，阿里巴巴搜索推荐事业部发布了一项新研究，首次使用强大的 Transformer 模型捕获用户行为序列的序列信号，供电子商务场景的推荐系统使用。该模型已经部署在淘宝线上，实验结果表明，与两个基准线对比，在线点击率（CTR）均有显著提高。本文是 AI 前线第 79 篇论文导读，我们将对这项研究工作进行详细解读。

论文作者系阿里巴巴 Qiwei Chen、Huan Zhao、Wei Li、Pipei Huang、Wenwu Ou，由 AI 前线编译整理

论文原文链接：https://arxiv.org/pdf/1905.06874.pdf

介绍

推荐系统（RSs）已经在工业界流行了十来年，在过去五年里，基于深度学习的方法在工业界得到了广泛应用，例如，Google 的 wide & deep 模型和 Airbnb 的《Real-time personalization using embeddings for search ranking》。在阿里电商平台，RSs 已经成为 GMV 和收入的关键引擎，并在丰富的电子商务场景中部署各种基于深度学习的推荐方法。RSs 在阿里巴巴分为两个阶段：匹配（match）和排名（rank）。在匹配阶段，根据用户和商品的交互，一些相似的商品被选择出来作为候选集，然后学习一个 fine-tuned 预测模型，来预测用户点击给定候选商品集的概率。

在本文中，我们关注的是阿里巴巴淘宝的排名（rank）阶段。在阿里

电商平台有数百万候选商品，我们需要根据用户的历史行为，预测他 / 她点击给定候选商品的概率。在深度学习时代，嵌入和 MLP 已经成为工业 RSs 的标准范式：大量的原始特征嵌入到低维空间中作为向量，然后输入到全连接层，即多层感知机（MLP），以预测用户是否会点击某个商品。其代表工作是 Google 的 wide&deep 网络（WDL）和阿里巴巴的深度兴趣网络（DIN）。

在淘宝，我们基于 WDL 网络构建 rank 模型，其中各种特征都使用 Embedding 和 MLP 范式，比如，商品的类别和品牌特征、商品的统计特征、用户画像特征。尽管这一框架取得了成功，从本质上讲，它远不能令人满意，因为它在实践中忽略了一种非常重要的信号，即用户行为序列背后的序列信号，即用户按顺序点击商品。实际上，该顺序对于预测用户的未来点击非常重要。例如，用户在淘宝买了一部 iPhone 后，往往会点击手机外壳，或者在买了一条裤子后试图找到合适的鞋子。从这个意义上来说，在淘宝排名阶段部署一个预测模型时，不考虑这个因素是有问题的。在 WDL 中，它仅仅是连接所有特征，而没有捕获用户行为序列之间的顺序信息。DIN 提出使用注意力机制来捕获候选项与用户先前点击商品之间的相似性，但未考虑用户行为序列背后的序列性质。

因此，在这项工作中，为了解决 WDL 和 DIN 面临的上述问题，我们尝试将淘宝上的用户行为序列的顺序信号整合到 RS 中。受自然语言处理（NLP）中机器翻译任务 Transformer 大获成功的启发，通过考虑嵌入阶段的顺序信息，我们运用了 self-attention 机制，在用户行为序列中，为每个商品学习一个更好的表征。然后，将它们输入 MLP，以预测用户对候选商品的反馈。

Transformer 的主要优点是，它能更好地捕获句子中单词之间的依赖性，通过 self-attention 机制，直观地说，用户行为序列中 item 之间的“依赖关系”可以通过 Transformer 抽取。因此，我们提出了在淘宝电商推荐中的用户行为序列 Transformer（BST）。离线实验和在线 A/B 测试表明，BST 与现有方法相比有明显优势。目前 BST 已经部署在淘宝推荐的 rank 阶段，每天为数亿消费者提供推荐服务。

本文第 2 节将详细阐述 BST 的体系结构，第 3 节介绍包括离线和在线测试的实验结果。相关工作在第 4 节中进行了回顾，最后是我们对这项工作的总结。

架构

在 rank 阶段，我们将推荐任务建模为点击率（CTR）预测问题，定义如下：给定用户的行为序列 S(u) = {v1,v2, ⋯,vn } 被用户 u 点击，我们需要学习一个函数 F 来预测用户点击 vt 的概率，其中 vt 是其中的一个候选 item。其他特征包括用户画像、上下文、item 和交叉特征。

我们在 WDL 之上构建 BST，总体架构如图 1 所示。从图 1 中，我们可以看到它遵循流行的 Embedding&MLP 范式，其中，先前的点击 item 和相关的特征首先嵌入到低维向量中，然后再输入到 MLP。BST 和 WDL 之间的关键区别在于我们添加了 Transformer 层，通过捕获底层的顺序信号来学习更好地表征用户点击的 item。在下面的部分中，我们自下而上地介绍了 BST 的关键组件：Embedding 层、Transformer 层和 MLP。

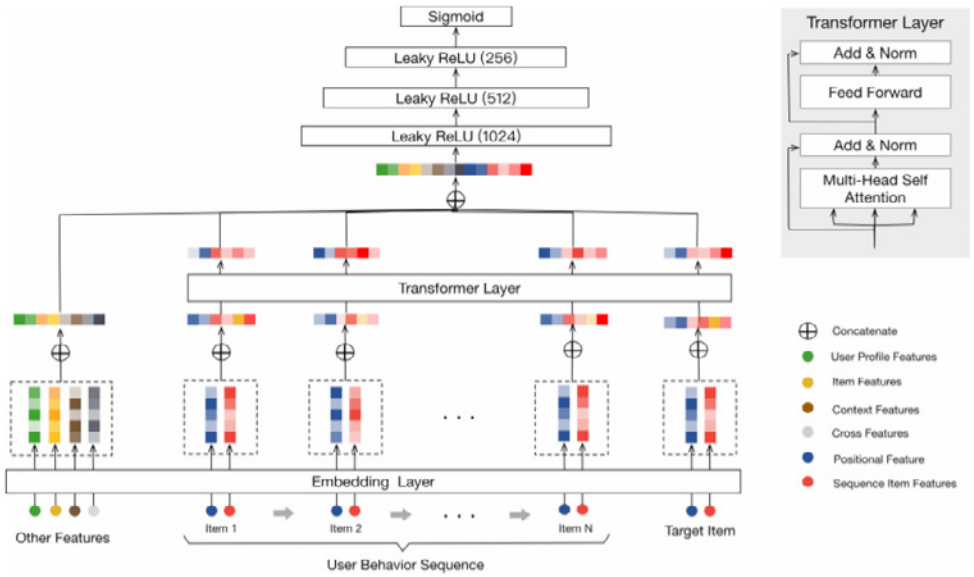


图 1 BST 的总体架构

BST 将用户的行为序列作为输入，包括目标 item 和其他特征。它首先将这些输入特征嵌入为低维向量。为了更好地捕获行为序列中 item 之间的关系，Transformer 层用于学习序列中每个 item 的更深层次的表示。然后通过连接其他特征的嵌入和 Transformer 层的输出，三层的 MLP 用于学习隐藏特征的交互作用，Sigmoid 函数用于生成最终的输出。

注：“位置特征”被纳入了“序列特征”。

2.1 Embedding 层

第一个组件是 Embedding 层，它将所有输入特征嵌入到固定大小的低维向量中。在我们的场景中，有各种各样的特征，比如，用户画像特征、item 特征、上下文特征以及各种不同的组合特征。由于这项工作的重点是用 Transformer 建模行为序列，为了简单起见，我们将所有这些特征表示为“其他特征”，并在表 1 中给出一些示例。如前面图 1 所示，我们将左侧的“其他特征”连接起来，并将它们嵌入到低维向量中。对于这些特征，我们创建了一个嵌入矩阵 Wo ∈ R(|D |×do)，其中 do 是维度大小。

Table 1: The “Other Features“ shown in left side of Figure 1. We use much more features in practice, and show a number of effective ones for simplicity.

User	Item	Context	Cross
gender	category_id	match_type	age * item_id
age	shop_id	display position	os * item_id
city	tag	page No.	gender * category_id
...	...	...	...

此外，我们还获得了行为序列中每个项目的嵌入，包括目标 item。如前面图 1 所示，我们使用两种类型的特征来表征一个 item，“序列 item 特征”（红色部分）和“位置特征”（深蓝色）。其中，“序列 item 特征”包括 item_id 和 category_id。

请注意，一个 item 往往有数百个特征，但是，在行为序列中选择全部来表征这个 item 太昂贵了。正如我们之前的工作《Billion-scale commodity embedding for e-commerce recommendation in alibaba》介绍，item_id 和 category_id 对于性能来说已经足够好了。

在嵌入用户行为序列中，我们选择这两个作为稀疏特征来表征每个 item。“位置特征”对应于下面的“位置嵌入”。然后，对于每个 item，我们将序列特征与位置特征相结合，生成嵌入矩阵 Wv ∈ R(|V |×dv)，其中，dv 为嵌入的维度大小，|V | 为 item 的数量。我们使用 ei ∈ R(dv) 来表征给定行为序列中第 i 个 item 的嵌入。

位置嵌入：在《Attention is all you need》论文中，作者提出了一种位置嵌入来捕获句子中的顺序信息。同样，顺序也存在于用户的行为序列中。

因此，我们添加“位置”作为 bottom layer 中每个 item 的输入特征，然后将其投射为低维向量。注意，item vi 的位置值计算为 pos(vi)=t(vt)-t(vi)，其中，t(vt) 表示推荐的时刻，t(vi) 表示用户点击 item vi 时的时间戳。我们采用这种方法是因为在我们的场景中，它优于《Attention is all you need》论文中使用的 sin 和 cos 函数。

2.2 Transformer 层

Transformer 层通过捕获与行为序列中其他 item 的关系，为每个 item 学习更深入的表征。

Self-attention layer

scaled 点积 attention 在论文《Attention is all you need》定义如下：

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}, \quad (1)$$

其中，Q 表示查询，K 表示键，V 表示值。在我们的场景中，self-attention 操作将 item 的嵌入作为输入，并通过线性投影将它们转换为三个矩阵，并将它们输入到 attention 层。跟论文《Attention is all you need》一样，我们使用 multi-head attention：

$$\mathbf{S} = \text{MH}(\mathbf{E}) = \text{Concat}(head_1, head_2, \cdots, head_h)\mathbf{W}^H, \quad (2)$$

$$head_i = \text{Attention}(\mathbf{E}\mathbf{W}^Q, \mathbf{E}\mathbf{W}^K, \mathbf{E}\mathbf{W}^V), \quad (3)$$

其中，投影矩阵 W(Q)、W(K)、W(V) ∈ R(d×d)，E 是嵌入所有 item 的矩阵。h 是 head 的数量。

Point-wise Feed-Forward Network

在论文《Attention is all you need》基础上，我们添加了 point-wise Feed-Forward Network（FFN），以进一步增强模型的非线性能力，定义如下：

$$\mathbf{F} = FFN(\mathbf{S}).$$

为了避免过拟合，并从层次上学习有意义的特征，我们在 self-attention 和 FFN 中都使用了 dropout 和 LeakyReLU。

self-attention 和 FFN 层的整体输出如下：

$$\mathbf{S}' = \text{LayerNorm}(\mathbf{S} + \text{Dropout}(MH(\mathbf{S})), \tag{5}$$

$$\mathbf{F} = \text{LayerNorm}\left(\mathbf{S}' + \text{Dropout}(\text{LeakyReLU}(\mathbf{S}'\mathbf{W}^{(1)} + b^{(1)})\mathbf{W}^{(2)} + b^{(2)})\right), \tag{6}$$

其中 ,W(1)、b(1)、 W(2)、 b(2) 都是可学习的参数，和 LayerNorm 层是标准的归一化层。

堆叠 self-attention 块

在 self-attention 之后，它聚合了之前所有 item 的嵌入。为了进一步建模 item 序列基础上的复杂关系，我们将自构建块堆叠起来，第 b 个块的定义如下：

$$\mathbf{S}^b = SA(F^{(b-1)}), \tag{7}$$

$$\mathbf{F}^b = FFN(\mathbf{S}^b), \forall i \in 1, 2, \cdots, n. \tag{8}$$

在实践中，我们在实验中观察到与 b=2,3 相比 b=1 获得了更好的性能（见下文表 4）。为了效率，我们没有尝试更大的 b，这部分工作将在接下来进一步研究。

2.3 MLP 层和损失函数

通过连接其他特征的嵌入和应用于目标 item 的 Transformer 层的输出，我们使用三个完全连接的层来进一步学习稠密特征之间的交叉，这是在工业界的标准实践。

为了预测用户是否点击目标 item vt，我们将其建模为一个二元分类问题，使用 sigmoid 函数作为输出单元。为了训练这个模型，我们使用了交叉熵（cross-entropy）损失函数：

$$\mathcal{L} = -\frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \left(y \log p(x) + (1-y) \log(1-p(x))\right), \tag{9}$$

其中，D 代表所有样本，y ∈ {0, 1} 为标签表示用户是否点击了某个 item，p(x) 是经过 sigmoid 单元之后的网络输出的概率值，表示样本 x 被

点击的预测概率。

实验

3.1 设置

数据集

数据集是根据淘宝 App 的日志构建的。我们根据用户 8 天内的行为构建了一个离线数据集。我们使用前七天作为训练集，最后一天作为测试集。数据集的统计数据如表 2 所示。我们可以看到数据集非常大并且稀疏。

Table 2: Statistics of the constructed Taobao dataset.

Dataset	#Users	#Items	#Samples
Taobao	298,349,235	12,166,060	47,556,271,927

基准线

为了说明 BST 的有效性，我们将其与两个模型进行比较：WDL 和 DIN。此外，我们还通过将顺序信息整合到 WDL 中创建一个基准，称为 WDL(+Seq)，它平均聚合以前点击的 item 的嵌入。我们的框架是建立在 WDL 之上的，通过添加 Transformer 的顺序建模，而 DIN 被提出是由于用 attention 机制捕获目标 item 和先前点击的 item 之间的相识性。

评价指标

对于离线结果，我们使用 AUC 评分用于评估不同模型的性能。对于在线 A/B 测试，我们使用 CTR 和 average RT 评估所有的模型。RT 是响应时间的缩写，表示给定查询生成推荐结果的耗时。我们使用平均 RT 作为度量标准来评估不同在线生产环境下的效率。

配置

我们的模型是用 Python 2.7 和 TensorFlow 1.4 实现的，并选择“Adagrad”作为优化器。此外，我们在表 3 中给出了模型参数的详细信息。

3.2 结果分析

结果如表 4 所示，从中我们可以看到 BST 相对基准线的优势。

Table 3: The configuration of BST, and the meaning of the parameters can be inferred from their names.

Configuration of BST.			
embedding size	4 ~64	batch size	256
head number	8	dropout	0.2
sequence length	20	#epochs	1
transformer block	1	queue capacity	1024
MLP Shape	1024 * 512 * 256	learning rate	0.01

Table 4: Offline AUCs and online CTR gains of different methods. Online CTR gain is relative to the control group.

Methods	Offline AUC	Online CTR Gain	Average RT(ms)
WDL	0.7734	-	13
WDL(+Seq)	0.7846	+3.03%	14
DIN	0.7866	+4.55%	16
BST($b = 1$)	**0.7894**	**+7.57%**	20
BST($b = 2$)	0.7885	-	-
BST($b = 3$)	0.7823	-	-

具体来说，离线实验的 AUC 由 0.7734(WDL) 和 0.7866(DIN) 提高到 0.7894(BST)。当比较 WDL 和 WDL(+Seq) 时，我们可以看到，以简单的平均方式整合顺序信息的有效性。这意味着在 self-attention 的帮助下，BST 提供了一种强大的能力来捕获用户行为序列的顺序信号。请注意，从我们的实践经验来看，即使是线下 AUC 的微小收益也会导致在线 CTR 的巨大收益。在 WDL 中，Google 的研究人员也报告了类似的现象。

此外，在效率方面，BST 的平均 RT 接近 WDL 和 DIN 的平均 RT，这保证了在现实推荐场景中大规模部署像 Transformer 这样的复杂模型的可行性。

最后，我们还展示了堆叠 self-attention 层的影响。从表 4 可以看出，b=1 得到最佳的离线 AUC。这可能是由于用户行为序列中的顺序依赖性不如机器翻译任务中句子复杂，因此，少量的块数量，就足以获得良好的性能。类似的观察报告见《Self-attentive sequential recommendation》论文。

因此，我们选择 b=1 在生产环境中部署 BST，只报告了表 4 中 b=1 的在线 CTR 的收益。

相关工作

自从 WDL 提出以来，业界研究同仁提出了一系列以深度学习为基础的方法，如 Deepfm、Xdeepfm、Deep&Cross 网络等。但是，所有这些之前的工作都集中在神经网络的特征组合和不同架构，忽略了实际推荐场景中用户行为序列的顺序性。2017 年，阿里妈妈的精准定向检索及基础算法团队提出了深度兴趣网络 DIN，通过注意力机制来处理用户的行为序列。

我们的模型和 DIN 的关键区别在于，我们提出使用 Transformer 来学习用户行为序列中每个 item 的更深层的表征，而 DIN 试图捕获之前点击的 item 和目标 item 之间的不同相似性。换句话说，我们的 Transformer 模型更适合捕获顺序信号。在《Self-attentive sequential recommendation》论文和《Sequential Recommendation with Bidirectional Encoder Representations from Transformer》论文中，提出 Transformer 来解决顺序推荐问题，同时在 CTR 预测方面，体系结构方面不同于我们的模型。

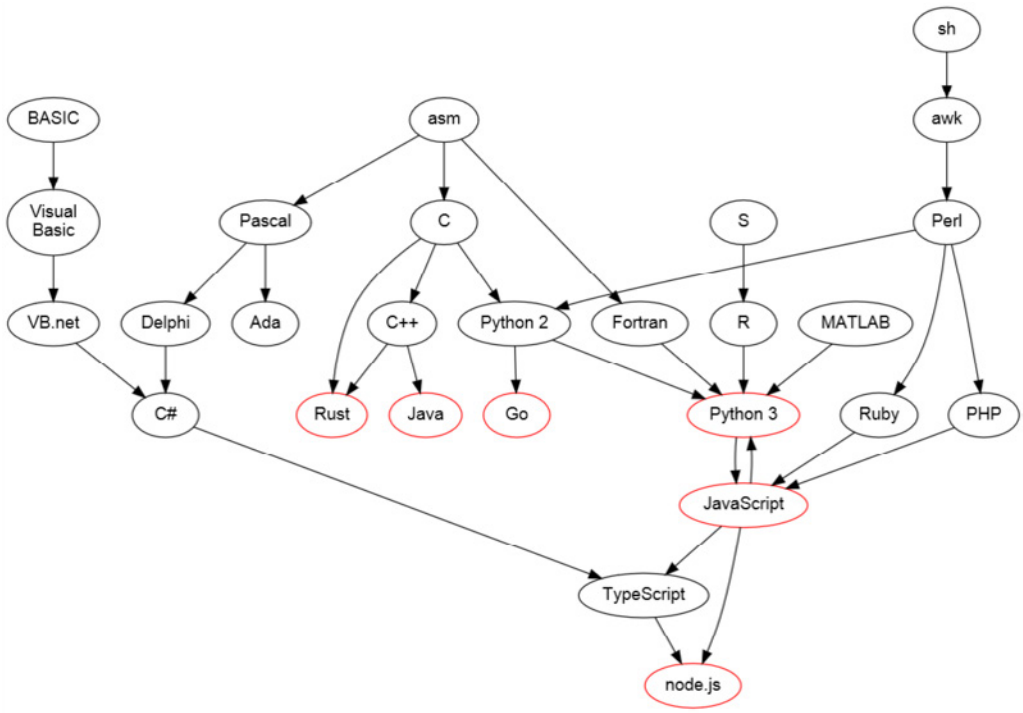
结论

在本文中，我们介绍了如何将 Transformer 应用到淘宝推荐中的技术细节。利用强大的序列关系捕获能力，通过大量的实验证明了该 Transformer 在用户行为序列建模中的优越性。此外，我们还介绍了在淘宝生产环境中部署该模型的细节，目前 BST 已经在淘宝线上为中国数亿用户提供推荐服务。

一张主流编程语言的变迁图，讲清程序员迁移模式

作者 apenwarr 译者 方彦

我绘制了一个主流编程语言的变迁图，用以表示程序员在不同语言之间的切换路径。



关于编程语言，还有很多类似的图可以表示它们相互之间的演进。不过我并不想从语言设计者角度来说明这个问题，而是想从程序员本身来看待语言演变。虽然两者间有些接近，但并不完全相同。

从该图可以看出，如果开始使用的是编程语言 A，下一个最有可能切换过去的是哪种语言。这种推测不是非常科学。不过如果你需要精确的科学，就不会在这里阅读这篇文章了，对吗？也许这张流程图对我来说，能

揭示更多的内容。

声明：在此处，不考虑程序员最喜欢的是什么语言。人们可以在任意两个语言之间切换，也可以学习很多种语言、然后选择最适合工作的一种语言。本文中的观点有一定的倾向性。

在接下来的篇幅里，我所阐述的均为个人观点。为了不影响读者阅读，就不再一一做出声明了。

程序员迁移模式

我想强调下最普遍的“终极节点”。在这些节点上，人们在他们所处的维度找不到更好的可替代编程语言。这些终极节点包括：Rust、 Java、Go、 Python 3、JavaScript 和 node.js（node.js 作为一种特殊的 JavaScript，在这里特别指出）。

几年前，我认为 C 也是一个终极节点。可能现在也还可以这样认为，因为有大量的重要项目（如 OS 内核）仍使用了 C，而且可以认为它无可替代。不过有迹象表明 C 其实是可以替代的。我最喜欢的例子就是有趣的空指针。Linux 内核有个编译器带来的致命弱点，即 NULL 值“不可能”出现，因此没有对函数进行空指针检查。C 也是一团糟，其规格里有几个新编程语言所没有的致命错误。也许某天这些错误能被修复。

让我们回退几步。如果从顶部开始，根据人们进入编程的不同规格，可以看到四个主干：

- “低级”编程，包括 asm 和 C。
- “商业型”或“学习型”编程，从 BASIC 开始。
- 计算类 / 科技类编程，如 Fortran，MATLAB 和 R。
- 脚本 / 胶水编程，如 shell 和 perl。

（我们也会谈到“数据库查询语言”，比如 SQL。它是一枝独秀，所有替代它的尝试都以失败告终。数据库语言从上世纪六十年代开始就停滞不前了。甚至到现在，其关键字仍使用大写，因为（他们认为）这样能更好的理解代码。）

（我还忽略了 HTML 和 CSS。他们是真正的语言，不过现在大家都开始学习这两种语言，图上无法用相应的箭头来标识。Lisp 也没有考虑在内，因为它一直没有流行过，虽然有一小部分人一直希望它能流行起来。

我还建议添加第五个分类，“配置编辑”）

（该图也忽略了 Haskell。它可以在旁边用一个独立的框来表示，和其他语言之间没有出入的箭头，不过这没关系。Haskell 是个自嘲式的大笑话，除非涉及到 Monads，它不再使用 I/O 概念。）

不管怎么样，让我们回到上世纪九十年代。假设那时编程世界很简单，(1) 初级程序员使用 C，asm 或者 Turbo Pascal，(2) 商业程序员使用 VB，(3) 数值计算人员使用 Fortran，R 或 MATLAB，(4) 胶水编程者使用 sh 或 perl。

那时，编程语言就是这么严格划分的。画该图时，我才意识到这一点。显然，我们不会用 perl 来写操作系统内核，不会用 MATLAB 来写胶水程序，不会用 VB 来写大型矩阵相乘算法。

现在则变化很大。选择什么样的语言已经不再像过去那样明确了。

语言的变化主要是风格的变化

我们先来看树起点 asm（汇编语言）。用 Asm 来写程序是相当困难的。不过即使到现在，它仍是写某些程序最好的方式（如电脑启动后的最初几个指令，或是中断处理的入口代码）。不管是在 App Store 里还是手机上的 JIT 里，每个编译语言最终都会将代码编译成汇编或机器语言。

基于 asm，出现了两个分支：C 类型分支和 Pacal 类型分支。（Algol 出现的更早，不过此处忽略掉。宣称自己是 Algol 程序员的人并不多。Algol 对其他语言影响很大。）

Pascal 风格分支语言的特点是有 "begin…end"。C 风格语言的特点则是有括号。当然，C 影响了很多的编程语言设计，这点在图中没有体现。因为我们现在讨论的是程序员，而不是语言设计人员。

首先来看看 C。很奇怪，一旦人们开始使用 C，就习惯用它来处理各种情况。不管实现优劣与否，它是为数不多的能合理实现所有四类编程问题的语言之一。这四类都有些难度（除了低级编程，它正是 C 擅长的领域），不过 C 都能搞定，速度也还可以。

如果你是个 C 程序员，接下来会使用那种语言呢？这取决于用它来做什么。

显然，C++ 是一个选择。虽然其名字与语法和 C 很像，但它其实和

C 风格迥异。除了 BeOS，其他操作系统内核不会使用 C++。在极具潜力的 Rust 使用前，操作系统基本都使用 C 编写。

但是商业（“大型程序”）和数值计算（“快速程序”）领域的人员喜欢 C++。说喜欢不一定准确，但他们别无选择，只能使用 C++。

对于胶水程序，很多人会直接从 C（或 C++）转到 Python 2。我最近也这样做过。和怪异的 perl 不同，Python 2 类似 C 语言风格，其语法更简单。C 程序员很容易理解 Python C 模块（并可以编写新的 Python 模块）。从 Python 里调用 C 函数比其他语言更简单。如果在 Java 里调用，就需要处理非引用计数的垃圾回收问题。Python 的“os”模块提供了 C 系统调用及该调用能工作的环境。程序员可以访问 C 语言中的错误码并设置相应信号处理程序。唯一的问题就是 Python 有些慢。不过只把它作为胶水语言，则可以不考虑 Python 的慢速。速度慢时，可以写 C 模块或调用 C 的库或子程序。

另外，Java 面世后，很多 C 和 C++ 商业软件的程序员非常快地切换到 Java。C++ 编译时间长，头文件繁多，可移植性差，有释放后重用的错误问题。因此，虽然 Java 运行的很慢（和 Python 不同的是，Java 宣称“理论上运行很快”），人们还是更愿意使用 Java。

我记得有篇文章讲过，Go 的设计者最开始认为 Go 可以和 Java 或 C++ 媲美，但实际没有做到。Java 就像知名酒店，或是门洛帕克（Menlo Park），一旦入住就不想离店。同时，程序员没有从 C++ 切换到 Java 主要是因为：a)Java 速度比 C++ 慢，b）Java 仍有垃圾回收的经典问题。

Go 在之前已经切换到 Python 2 的胶水程序人员中流行起来。事实证明 Python 的慢速是其痛点所在。计算机复杂度急剧增加，Python 胶水程序规模也越来越大。相较其优势，动态类型带来的麻烦更多，因此人们开始使用预编译二进制。Python 2 占用很多内存，因此 Go 做了 RAM 改进，避免了从 C++ 迁移到 Go 带来的问题。Go 的难度和 Python 差不多，但它运行更快，占用 RAM 更少。

我们现在称 Go 是一种“系统”语言，因为提起胶水程序，我们更多的是想到 perl 和 ruby，不过它们的作用是一样的。（试试告诉一个 C 语言内核开发者，Go 是“系统”语言，看看他们的反应）Go 是粘合剂，可以把各个组件组合到一起成为一个系统。

Hejlsberg 因素

我们接下来看 Visual Basic 和 Pascal 分支。人们有不同的想法：明显正确的（“我为什么会使用与 C 或 Java 一样让人痛苦的语言呢？”），或明显错误的（“可视化的…Basic？开玩笑吧？”）。二十世纪八十年代和九十年代，一些人仍认为编程应该让新手可以方便使用，因此在个人电脑上预装了免费的编程语言，大部分都是 BASIC。

另一方面，大学教授编程时，则避开了 BASIC（“如果学生前期使用过 BASIC，就不能对学生很好的进行编程授课”），也没有选择 C。他们更倾向于 Pascal，认为 Pascal 易于学习。就像 Algol 的学术论文里提到的一样，而且它的语法适合教学，能让每个学生都能听懂。因此就有了学术分支和个人电脑分支，不过它们有个共同点，那就是都和 C 不像。

基于 PC（DOS）的 BASIC 演变为基于 Windows 的 Visual Basic，这可能是 JavaScript 出现前使用最多、最受欢迎的编程语言。（现在，它仍是在 Excel 中使用的“宏”语言。目前有很多 Excel 的程序员，虽然他们并不认为自己是程序员。）

同时，Pascal 也在努力往 PC 转。因为 Turbo Pascal 的出现，它变得流行起来，并一度成为最快的编译器。在速度上，Pascal 的确没有夸张。甚至有一些 C 程序员也喜欢用 Pascal，不是因为喜欢其类 C 的语法，而是因为它的速度很快。（Turbo C 也可以，但速度还不够。它比其他 C 编译器都快。）（大学里，Pascal 学术性越来越强，后来演变成 Modula 和 Ada。如果不是美国军方在其高可靠系统中采用了 Ada 语言，这个分支早该终结了。现在我们可以忽略 Ada。）

那时还有两个“商业”开发分支：BASIC 和 Pascal 分支。Windows 问世后，出现了 Visual Basic。基于 DOS 的 Turbo Pascal 有点过时，基于 WIndows 的 Turbo Pascal 也并不出众。为了竞争，Turbo Pascal 的设计者 Anders Hejlsberg 创建了 Delphi。Delphi 和 Visual Basic 一样，有可视化的编程环境，但它基于 Turbo Pascal 语言，也极少出现找不到或不匹配实时动态链接库的烦人问题。

Delphi 很好，但它不属于 Microsoft。掺杂商业因素后，局面变得有些困难。在一系列出人意料的事件之后，Hejlsberg 离开了 Microsoft，但仍继续 C# 的开发，发布了 Microsoft .NET 平台，并包含 Visual Basic.Net（这是个很可怕的产品）。据称 C# 统一了两个分支。

不幸的是如前所述，VB.NET 很可怕。它和 Visual Basic 几乎没有共同点，更像是 C++ 的一个慢速版本，披了件有点非典型 Basic 的语法外衣，还带着一个更糟的 UI 设计工具。C# 也不是 Delphi。不过这几种语言都销声匿迹了，Microsoft 尽力推动了这一点。（除了 Microsoft Office，它到现在仍在使用最开始的 Visual Basic 语法，称为 "Visual Basic for Applications"，即 VBA。比起 .NET，它使用的更广泛，更受用户喜欢。）

我不清楚怎样才能叫做一名 Visual Basic 程序员。微软致力于让他们改用 VB.NET，但大多数人并不愿意。我想在图中画一条“他们实际的选择”的箭头，不过老实说我也不知道应该指向哪里。也许他们成为了 web 开发者，或者编写了 Excel 的宏。

从现在看，如果写基于微软主推的基于 .NET 平台的 Windows 软件，是件很有趣的事。可能使用的语言都会深受 Hejlsberg 的影响。Hejlsberg 的语言在反击之前被微软和 Visual Basic 所遏制，于是 Hejlsberg 转向写 Typescript，这个留待以后讨论。

胶水语言的简要介绍

最初的胶水语言是 Unix shell，它因引入“管道”概念也很著名。“管道”连接简便的工具来完成复杂的工作。

> 啊，就是那些日子
>
> 那些日子一去不复返，Perl 就是献给它们的悼词
>
> —— Rob Pike

事实证明，设计小而简单的工具是困难的，通常我们没有足够的时间来做这个。能够让我们跳过这些轻便工具，致力于编写奇特的、能够粘着很多乱七八糟的小程序的语言变得越来越流行。（它对 shell 语言的缺陷，尤其是与引用和通配符扩展规则相关的缺陷并没有帮助。）

第一个是 awk，它是语法和 C 类似的解析语言，可以用在 shell 的管道上。那时，在一种语言（sh）的一行中里使用另一种微语言（awk）有点奇怪，庆幸的是我们适应了，因为现在的 web 程序都是这样的。（我们略过 csh，它是另一种与 C 语法不兼容的语言，存在不同的致命缺陷，可以被 sh 替代。）

接下来是 Perl。awk 没有足够多的标点符号，从而促成了 Perl 的产生。

（好吧，这只是个玩笑。）

Perl 开始到 Perl 5，越来越受欢迎。现在，Perl 停止改进语法，在 Perl 6 上倾尽全力，从零开始打造。（在图中并没有标出 Perl 6，因为还没有人切换过去。）

这样的配置给在几个方向断层进行“粘合”留下了空间。如果程序员觉得 Perl 的语法差劲，可能会切换到 Python。如果他们认为 Perl 的语法很神奇有效力，只需要一些调整，则可能会切换到 Ruby。如果使用 Perl 来运行 Web 的 CGI 脚本，则可能会保持原样，也可能会转而切换到 PHP。

Ruby 很快成为 Web 服务器支持的语言（进而是 Ruby on Rails）。Python 也同样在演进。

现在有趣的是：整整一代程序员摒弃了命令行方式（这也是胶水语言运行的方式），希望在 Web 端可以做任何事情。从某方面来说，这样更好，比如在一个胶水程序中可以超链接到另一个胶水程序。从另一方面来说，则更糟糕，因为现在所有的 Web 程序都很慢，不能使用脚本，而且安装 Electron 的另一个副本需要 500MB 的 RAM 空间等等。这就引入了 Web 语言这个话题。

Web 语言

图中，集中在 JavaScript 的“胶水”分支有很多的箭头指向，这并不奇怪。JavaScript 最初只使用于前端。当 Node.js 出现后，这种情况完全改变了。现在，只需要学习一种语言来写前后端和命令行工具。JavaScript 最初的设计是将其作为最终的胶水语言，试图融合 HTML、CSS、面向对象编程、面向函数编程、动态语言、JITs 以及其它一切能通过 HTTP 请求得到的东西。

但是这样不太好，因为后向兼容对于 web 的成功至关重要。要保证这一点，就无法修复一些严重错误。1995 年，经过 10 天的设计，JavaScript 发布了。对于 10 天的成果而言，它相当优秀，但同时它也存在一些问题，无法对其进行修复。

这就是图中唯一一个有双向箭头的地方：JavaScript 和 Python 3 之间。我们把它叫做脚本语言的阴阳两面。

大部分出现过的胶水 +Web 语言正在消失，Python 不在其列，至少目前还不会消失。我猜是因为 Python 本身是合理的。使用 JavaScript 编程时间足够长的话，过段段时间后就会变得不大正常。这时为了缓解压力，程

序员有可能会切换到 Python。

同时，如果长时间使用 Python，最后准备编写 Web 应用程序时，前端代码和后端使用完全不同的语言是很烦人的。一个的语法是 ['a','b','c'].join(','),而另一个则变成了','.join(['a','b','c']),这让人完全记不清谁是谁了。

一种语言有 JIT，可以让其一旦运行起来就会速度很快。而另一种则是启动快，运行慢。

一种有合理的命名空间系统，而另一种则没有。

我不清楚从长期看，Python 3 是否能打败 JavaScript。但至少目前看，它不会被击败。

同时，对于编程事实分支从不满意的 Hejlsberg，看到了 JavaScript 的很多问题，引入了 TypeScript。与此同时，微软突然停止了对 Windows 应用程序的大力推进，开始大面积推广 Web 和开源。这意味着 Microsoft 第一次将其开发者推向 Web 语言即 JavaScript。在此基础上，他们有自己的 TypeScript，我觉得这是一种很好的语言。这个分支存在有数十年，开始和其分支融合，可能不久后会消失。

TypeScript 和 JavaScript 比，能胜出吗？这是个有趣的问题，我也不知道。我以前赌 Hejlsberg 能赢，不过我一般容易赌输。

Python 2 和 Python 3 的对比

综上所述，我对 Python 2 和 3 有了结论。它们很相似，但不尽相同。我认为，这是因为他们在整个程序员语言迁移图中所处的位置不同。Python 2 开发者来自 C 和 perl 开发人员，希望编写胶水代码。Web 服务器是后续添加的一个应用场景。我的意思是，Python 2 出现后，web 程序变得流行起来，这并不出人意料。很多 Python 2 的开发者转到 Go 的开发，因为他们想写的某些“系统胶水”代码使用 Go 正合适。

Python 3 的开发者是从不同的语言切换而来的。事实证明，Python 3 问世后，Python 的使用得到很大的发展，不过新加入的人群和以前的人群有所不同。由于带有模块 SciPy 和 Tensorflow，从科学类和数值类处理转过来的新程序员占了其中很大的比例。老实说，在高吞吐量的数值处理中，Python 是一个相当怪异的选择。但不论如何，这些库的存在是我们选择它的一个原因。我猜 Python 的另一个优势则是易于和 C 模块集成。当然，Python 3 本身就是网络编程。

想要理解 Python 2 和 3 的区别，只需看看其不同的字符串类型。Python 2 中，字符串是一组字节，因为操作系统、Unix 管道处理、网络 socket 的处理均以字节为单位。对于系统程序而言，Python 2 是胶水语言，其处理以字节为单位。

在 Python 3 中，字符串是一组 unicode 码。因为人们不擅长 unicode 码的转换，而和网络交互时，都是以 unicode 为基础。做科学数值计算的人不关心字符串，做网络编程的人更关心 unicode，所以 Python 3 使用 unicode。如果要用 Python 3 来编写系统程序，就会一直疲于 unicode 的转换，即使最简单的文件名也需要进行转换。这也正是有其因，必有其果。

相关文档

- Misunderstanding Exceptions (2007)
- You can't make C++ not ugly, but you can't not try (2010)

为什么高级程序员写的代码都是傻瓜式的？

作者 Scott Shippp 译者 核子可乐

Brian Goetz 是 Java 领域的技术大牛，同时也是《Java 并发实践》一书的作者之一。我最喜欢的一句名言就是他讲的。这句话出现在甲骨文公司以《编写傻瓜式代码》为题发表的访谈当中，当时记者问到 Goetz 如何才能编写出性能良好的代码。以下是他给出的睿智回应：

> 一般来说，在 Java 应用程序当中快速编写代码的方法，就是先写点傻瓜式代码——这类代码简单、干净，而且遵循最明确的面向对象原则。

在接下来的发言中，他一直在具体解释：为什么尝试优化代码并试图让代码看起来不那么傻瓜，正是程序员群体中的一种常见错误。而在我看来，这更多是一种新手程序员常犯的错误。

高级开发者的代码

如果大家像我一样，也经历过初窥门径的过程，那么各位应该还记得自己第一次看到高级开发者写出的代码时，心里想的是：“这玩意儿我也写得出来，为什么我就不是高级开发者呢？”

此后，我花了不少时间想写出相类似的代码，最后发现我根本就做不到。

关于“高级开发者”的疑问，并不在于我无法理解代码当中的特征。相反，我一眼就能明白他们写的代码在说些什么，因为这些代码从根本层面上就是在走傻瓜式路线，谁还看不懂啊。但除此之外，似乎还有更多不同。我记得当时我在想，“这“更多的”究竟是什么？又是怎么做到的？”

从那时起，我逐渐学会了编写傻瓜式代码的所有原则与质量保证方法，

包括：YAGNI 原则 (You Ain't Gonna Need It)；单一责任原则；DRY 原则（Don't Repeat Yourself）；单一级别抽象原则；低耦合等等。我自己，也慢慢成了“高级开发者”。（我其实很讨厌「高级开发者」这种说法，因此我始终坚持称自己为「软件工程师」，但这又是另一个故事了。）

我学到的最重要的经验就是：

> 编写傻瓜式代码实际上非常困难，但一旦实现则会带来远超预期的回报。

如何从一里外认出菜鸡程序员

在《重构：改进现有代码设计》当中，Kent Beck 指出：

> 任何傻瓜都能够编写出计算机可以理解的代码，但只有优秀的程序员能够编写出人类可以理解的代码。

如果大家遇到精妙无比的单一代码行，以及模糊的抽象及 / 或语言特征时，其背后几乎总会有一位初级开发者。说实话，后一种情况尤其常见。看到这些代码，我仿佛就看到这位程序员在强调：

> “快来看我！我真的很懂这门编程语言！我在使用默认接口同步本地线程 JavaBean 复制构造函数，并配合自定义通用未检查异常以及跨函数安全强化 JAXB Lombok 代码生成！看我厉不厉害！”

以上场景纯属胡说八道，但相信大家能够明白我的意思。这类代码往往来自那些只关注计算机理解，而不重视人类理解的程序员。

代码的本质，在于向其他人交流并向计算机发出指令；但相较于后者，目前代码的前一项作用显得尤其重要，因为有编译器负责将程序员编写的内容翻译成机器语言。一般来讲，这种转换会分多层实现，例如在将 Java 编译为 ByteCode 时，Java 代码首先由 Java 虚拟机在运行中读取，并最终将内容翻译为 0 和 1 的形式。

不过代码代表的仍然是人类的语言，除了指导计算机之外，必须也能够表达任务当中的谁、什么、何时、哪里、如何以及为什么等问题。再举个例子，即使你的公司被收购了五年多，这些代码也仍然得具有实际意义，并确保之前从未见过这些代码的新团队能够快速打开并进行功能增强 / 错误修复。

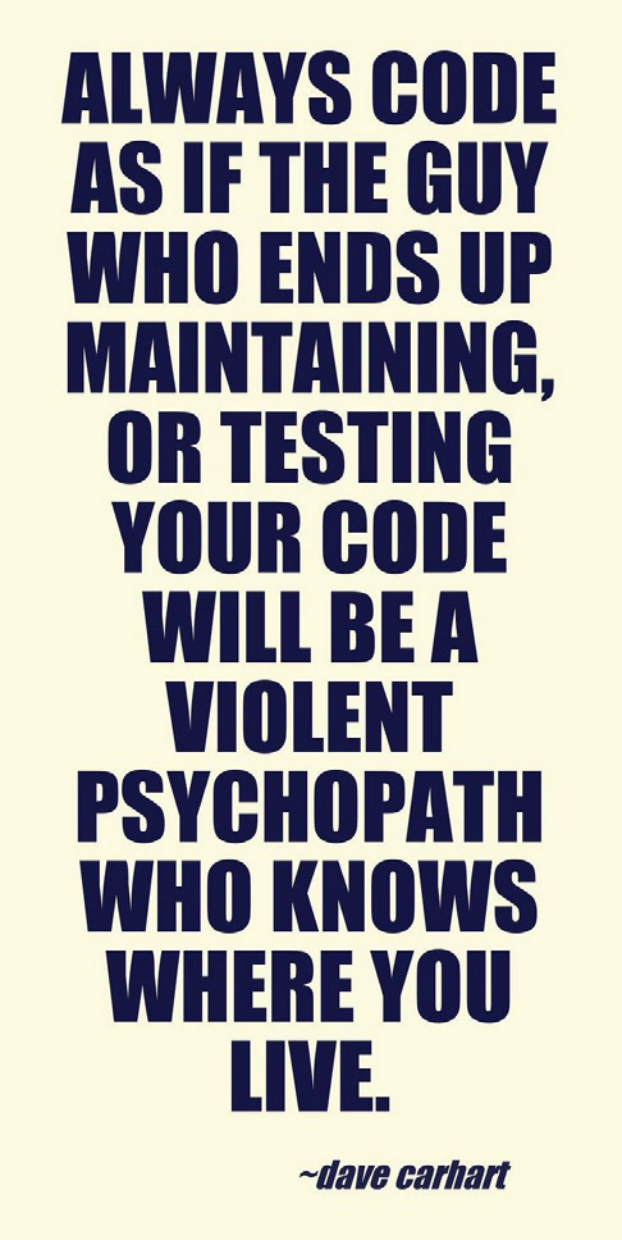
是的，编写傻瓜式代码非常困难。随着时间的推移，我觉得我越来越

喜欢这种编程风格了。当我在代码评论中收到“这代码真干净！”之类的评语时，我真的感到非常欣慰。我知道我能为整个团队、乃至未来的代码维护者们做的最好的事情，就是编写傻瓜式代码。

需知：

> 道路千万条，维护第一条。代码不规范，运维两行泪。

作为结尾，我再列出 Dave Carhart 的一些观点，只为博君一笑：



W3C Web 技术总负责人：拓展 Web 核心能力，W3C 关注哪些技术？

作者 徐川

5 月 6 日，InfoQ 主办的 QCon 2019 全球软件开发大会在北京举行。W3C Web 技术总负责人 Philippe Le Hégaret 在大会上做了《Now and the Future: An Overview of the Web in 2019》的分享，介绍了 Web 技术当前关注的技术方向。

Web 技术在过去的标准制订方面进展比较缓慢，因此在移动互联网时代受到一些质疑，不过这一情况已经在 HTML5 和 ES6 等新标准的推出下有所好转。

我们根据演讲整理如下。

> 声明：本文为整理而成，非直接引用部分不代表 Philippe 先生的观点，文章不代表 W3C 官方说法。

万维网联盟 W3C 于 1994 年由 Tim Berners-Lee 创立，迄今已出台 400 多份 Web 标准，W3C 的宗旨是中立、世界范围的技术协调、为未来技术发展奠定基础，它在全球有 4 个总部，包括位于北京航空航天大学的 W3C 中国。

近年来，W3C 着力拓展新的领域，其中最重要的一项是汽车互联网。

BYTON’s next generation IVI system

W3C 希望顺应机器应用生态的需求，减少独家商业方案带来的碎片化，打造适用于车联网的 Web 技术方案，因此成立了 W3C Auto 工作组，目前重点关注汽车数据 W3C VISS (Vehicle Information Service Specification 汽车信息服务标准)、消息推送、多媒体、地理数据服务 / 导航、支付等方面。

在拓展 Web 核心能力方面，W3C 当前的关注重点包括：

- WebRTC：Web 上原生的视频通话能力；
- Web Assembly：极大提升性能，以及将其它语言生态引入 Web；
- WebPerf：Web 应用的性能管理框架；
- Web Payments：原生的 Web 支付 API 标准；
- Immersive Web：让 Web 原生支持 VR 和 AR；
- WebAuthn：Web 原生验证；
- Service Workers：通过后台任务来支持离线以及提升性能；
- Web Components：自定义可重用的 Web 组件；
- MSE：多媒体子系统和 API；
- Extensive Web：允许开发者扩展浏览器功能。

下面对其中一些重点分别介绍。

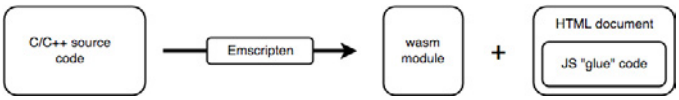
Web 图形处理能力

在利用 GPU 能力处理复杂和 3D 图形时，过去的 Web 标准是 WebGL，但这个标准面对快速发展的软硬件来说还不够，因此 W3C 又推出了 WebGPU 和 WHLSL，前者可进行幕前渲染与幕后图像绘制，以及多任务计算，后者是平台中立的 Shading 语言，可以独立于 Direct3D, Metal, or Vulkan 之上实现，InfoQ 之前做过介绍：

Webkit 推出新的着色语言 whlsl

Web Assembly

Web Assembly 一经问世就引起了广泛的注意，它基于栈的二进制指令结构虚拟机，目标是为高层语言，如 C/C++/Rust 提供编译，为高性能的页面脚本设计。

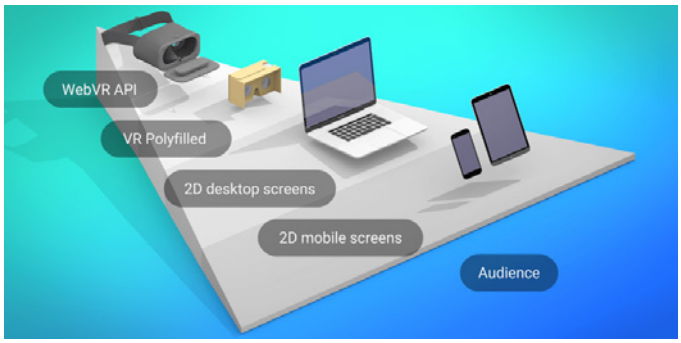


Web Assembly 的一个重要应用场景是游戏，之前已经有厂商演示将 C++ 编写而成的大型 3D 游戏迁移到浏览器上。

W3C 也在考虑更多的为游戏提供支持，包括对输入设备如手柄的支持以及多线程操作的支持。

WebXR

WebXR Device API 旨在为开发者提供用于开发沉浸式应用程序的接口，让他们可以通过这些接口开发出基于 Web 的、舒适的、有吸引力的沉浸式应用程序。

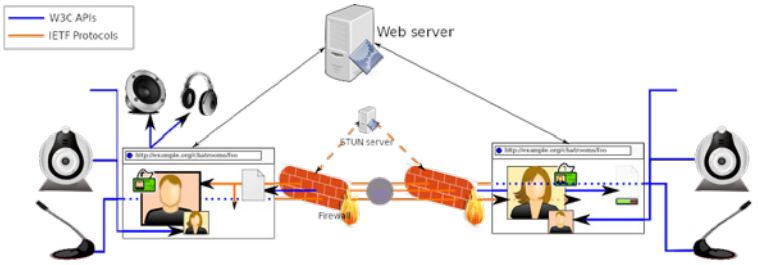


开发者可以通过 WebXR API 来创建 XR 体验。XR 包括了增强现实（AR）、虚拟现实（VR）和最近出现的沉浸式技术。前不久 W3C 刚发布 WebXR 的标准草案，我们也对其进行了介绍：

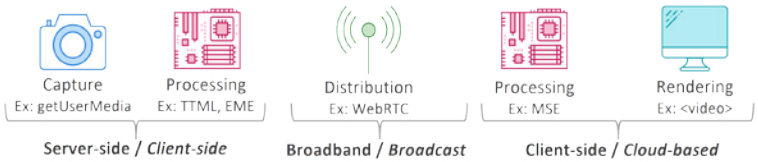
W3C 发布 WebXR 规范草案，用于开发沉浸式 Web 应用程序。

WebRTC 和 MSE

WebRTC 让语音 / 视频通话能力为所有的 Web App 所用，它已经被大多数浏览器支持，计划本年末成为浏览器默认能力。



WebRTC 只是 Web 视频体验其中的一部分，从视频的前端的生产 / 采集，到传输 / 编解码，更完整的图示如下：



MSE 媒体源扩展主要作用在视频流处理这一块，其中一项重要的技术是自适应比特率流 ABR，而 ABR 具体的实现包括 MPEG-DASH，近几年它已经逐渐引起重视，前段时间我们介绍过 bilibili 在这方面的实践：

MPEG-DASH在bilibili的应用与实践

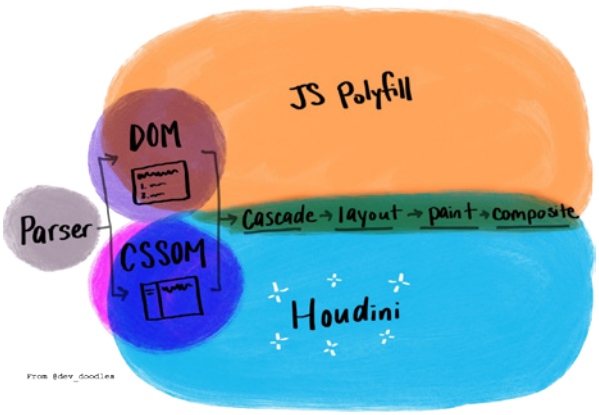
Web 字体（Web Fonts）

Web 字体是扩展 Web UI 展现能力的重要一环，它已经在英文互联网得到了广泛的实践，但由于字符集的问题——中文字体的字符集超过 20000 个字体，动态加载 Web 字体对中文来说不太现实。

但这一情形也即将改善，根据统计发现，网页上使用的中文常用字符少于 2400 个，基于这个事实催生了子字库提案，它可以动态分割字体并通过流的形式仅传输需要的字符的字体。这一提案可能在 2020 年成为现实。

渲染加强

W3C 在渲染加强方面主要包括两项技术：Worklets 和 CSS Houdini，前者是针对渲染线程的独立运行脚本，可以认为是一个简化版本的 Web Workers，让开发者能操作渲染管线中的底层部分，后者通过 Javascript 拓展 CSS 能力，通过 paint API/Layout API，提供比之前的 Canvas/Grid/Flexbox 更强大的表现能力。



Web 与机器学习

AI 是近年来的热门领域，公司和社区都有不少开发者试图将 Web 与 AI 联系起来，从而提供更好的体验。W3C 也成立了机器学习的社区组，并策划 2019 年的全球研讨会。

W3C 也试图提供原生的 Web 神经网络 API，这是为神经网络理论硬件加速设计的底层接口，用例包括：人物识别，语义诊断，情绪分析，图像自动分类等等，目标是可以直接使用预训练的深度神经网络模型。来自 Intel 的 Ningxin Hu 对此已经做出了一份提案。

Web 和网络

5G, QUIC, 边缘计算等正在革新网络拓扑服务。这些新的机遇对 Web 提出了新的技术需求，W3C 举办了 Web5G 研讨会，探讨新的网络形式带来的可能趋势。

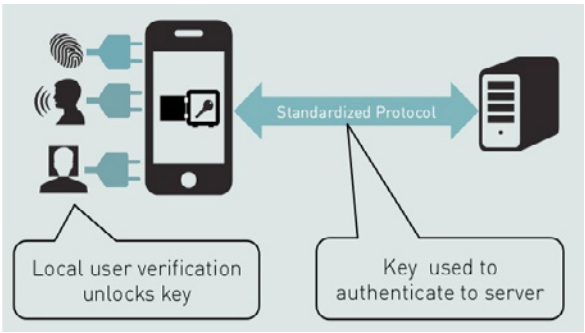
Web 性能

Web 性能方面的研究旨在提升 Web 开发体验以及优化 Web 应用性能，包括监控 / 优化和报告方面的提升，监控包括 Performance Timeline、server, resources、tasks, element, paint； 优 化 包 括 scheduling background tasks、resource loading、visibility；报告包括 beacon, network error 等。

Web 应用安全与 WebAuthn

Web 应用安全方面通过浏览器与后端相互协作的安全策略来保证浏览器安全，已经实现和推进中的技术包括域名策略 CORS, CSP, Secure Context，对混合内容如 HTTP 和 HTTPS 混用等也提升了安全等级，另外

还有新的 FeaturePolicy API 和 Permission API 来防止特性的滥用。



WebAuthn 是为线上快速身份验证 FIDO 2.0 设计的 Web API，它利用如今越来越流行的生物特征识别机制（如指纹 / 面容识别），为每个网页提供了与其域名绑定的独一无二的加密和验证机制。它可以实现本地化的验证数据，如生物识别等，不会离开设备。2019 年 3 月第一版标准正式通过实现。

无障碍访问（Accessibility/a11y）

世界上有超过十亿的残障人士，我们需要确保他们无障碍地在线进行教育，电商，工作，娱乐活动。Web 无障碍访问技术保证残障人士也可以流畅使用 Web，包括听觉，视觉，操作，认知等多个层面。中国正参考 W3C 相关提案简体中文版本进行国家标准的更新。

W3C 十分重视无障碍访问技术，为其提供了多种开发者资源和工具，可访问官方页面获取。

Web 中文兴趣组与小程序研讨会

在本地社区方面，W3C 正在尝试使用熟识的本地语言（中文）讨论技术想法和独特的 Web 需求，目前中文兴趣组策划的小程序 / 快应用研讨会将于近日举行，其主题包括：

- 小程序和快应用解决了哪些 Web 还没有解决的问题，如何解决的；
- 小程序和快应用存在的问题，包括开发中遇到的问题和使用中遇到的问题；
- 不同小程序 / 快应用 API 之间的差异、跨平台开发框架以及下一代移动 Web 应用的标准化方向；
- 新场景的 API (AR and AI)。

人人都是 API 设计师：我对 RESTful API、GraphQL、RPC API 的思考

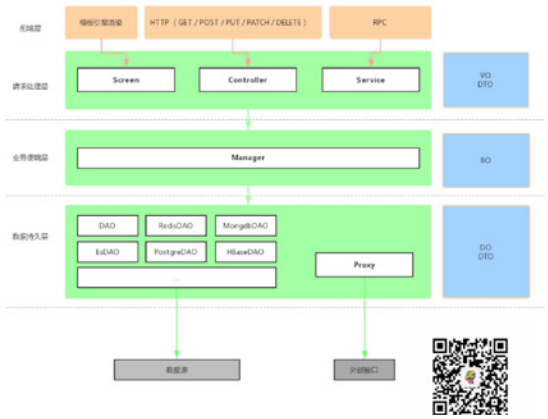
作者 梁桂钊

有一段时间没怎么写文章了，今天提笔写一篇自己对 API 设计的思考。首先，为什么写这个话题呢？其一，我阅读了《阿里研究员谷朴：API 设计最佳实践的思考》一文后受益良多，前两天并转载了这篇文章也引发了广大读者的兴趣，我觉得我应该把我自己的思考整理成文与大家一起分享与碰撞。其二，我觉得我针对这个话题，可以半个小时之内搞定，争取在 1 点前关灯睡觉，哈哈。

现在，我们来一起探讨 API 的设计之道。我会抛出几个观点，欢迎探讨。

一、定义好的规范，已经成功了一大半

通常情况下，规范就是大家约定俗成的标准，如果大家都遵守这套标准，那么自然沟通成本大大降低。例如，大家都希望从阿里的规范上面学习，在自己的业务中也定义几个领域模型：VO、BO、DO、DTO。其中，DO（Data Object）与数据库表结构一一对应，通过 DAO 层向上传输数据源对象。而 DTO（Data Transfer Object）是远程调用对象，它是 RPC 服务提供的领域模型。对于 BO（Business Object），它是业务逻辑层封装业务逻辑的对象，一般情况下，它是聚合了多个数据源的复合对象。那么，VO（View Object）通常是请求处理层传输的对象，它通过 Spring 框架的



转换后，往往是一个 JSON 对象。

事实上，阿里这种复杂的业务中如果不划分清楚 DO、BO、DTO、VO 的领域模型，其内部代码很容易就混乱了，内部的 RPC 在 service 层的基础上又增加了 manager 层，从而实现内部的规范统一化。但是，如果只是单独的域又没有太多外部依赖，那么，完全不要设计这么复杂，除非预期到可能会变得庞大和复杂化。对此，设计过程中因地制宜就显得特别重要了。

另外一个规范的例子是 RESTful API。在 REST 架构风格中，每一个 URI 代表一种资源。因此，URI 是每一个资源的地址的唯一资源定位符。所谓资源，实际上就是一个信息实体，它可以是服务器上的一段文本、一个文件、一张图片、一首歌曲，或者是一种服务。RESTful API 规定了通过 GET、 POST、 PUT、 PATCH、 DELETE 等方式对服务端的资源进行操作。

```
【GET】        /users          #查询用户信息列表
【GET】        /users/1001     #查看某个用户信息
【POST】       /users          #新建用户信息
【PUT】        /users/1001     #更新用户信息(全部字段)
【PATCH】      /users/1001     #更新用户信息(部分字段)
【DELETE】     /users/1001     #删除用户信息
```

事实上，RESTful API 的实现分了四个层级。第一层次（Level 0）的 Web API 服务只是使用 HTTP 作为传输方式。第二层次（Level 1）的 Web API 服务引入了资源的概念。每个资源有对应的标识符和表达。第三层次（Level 2）的 Web API 服务使用不同的 HTTP 方法来进行不同的操作，并且使用 HTTP 状态码来表示不同的结果。第四层次（Level 3）的 Web API 服务使用 HATEOAS。在资源的表达中包含了链接信息。客户端可以根据链接来发现可以执行的动作。通常情况下，伪 RESTful API 都是基于第一层次与第二层次设计的。例如，我们的 Web API 中使用各种动词，例如 get_menu 和 save_menu ，而真正意义上的 RESTful API 需要满足第三层级以上。如果我们遵守了这套规范，我们就很可能就设计出通俗易懂的 API。

注意的是，定义好的规范，我们已经成功了一大半。如果这套规范是业内标准，那么我们可以大胆实践，不要担心别人不会用，只要把业界标准丢给他好好学习一下就可以啦。例如，Spring 已经在 Java 的生态中举

足轻重，如果一个新人不懂 Spring 就有点说不过去了。但是，很多时候因为业务的限制和公司的技术，我们可能使用基于第一层次与第二层次设计的伪 RESTful API，但是它不一定就是落后的，不好的，只要团队内部形成规范，降低大家的学习成本即可。很多时候，我们试图改变团队的习惯去学习一个新的规范，所带来的收益（投入产出比）甚微，那就得不偿失了。

总结一下，定义好的规范的目的在于，降低学习成本，使得 API 尽可能通俗易懂。当然，设计的 API 通俗易懂还有其他方式，例如我们定义的 API 的名字易于理解，API 的实现尽可能通用等。

二、探讨 API 接口的兼容性

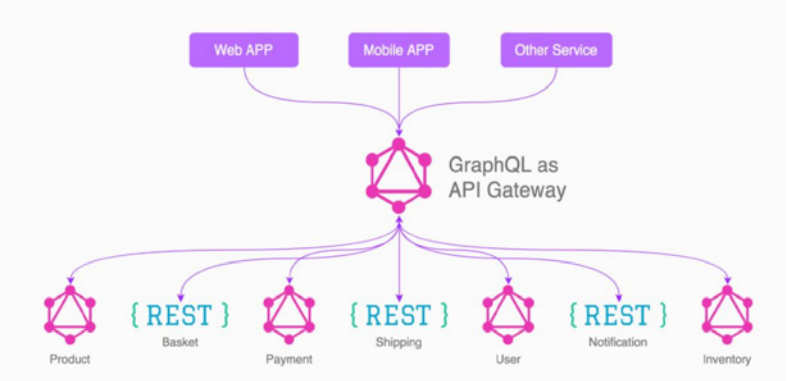
API 接口都是不断演进的。因此，我们需要在一定程度上适应变化。在 RESTful API 中，API 接口应该尽量兼容之前的版本。但是，在实际业务开发场景中，可能随着业务需求的不断迭代，现有的 API 接口无法支持旧版本的适配，此时如果强制升级服务端的 API 接口将导致客户端旧有功能出现故障。实际上，Web 端是部署在服务器，因此它可以很容易为了适配服务端的新的 API 接口进行版本升级，然而像 Android 端、IOS 端、PC 端等其他客户端是运行在用户的机器上，因此当前产品很难做到适配新的服务端的 API 接口，从而出现功能故障，这种情况下，用户必须升级产品到最新的版本才能正常使用。为了解决这个版本不兼容问题，在设计 RESTful API 的一种实用的做法是使用版本号。一般情况下，我们会在 url 中保留版本号，并同时兼容多个版本。

```
【GET】  /v1/users/{user_id}  //版本v1的查询用户列表的API接口
【GET】  /v2/users/{user_id}  //版本v2的查询用户列表的API接口
```

现在，我们可以不改变版本 v1 的查询用户列表的 API 接口的情况下，新增版本 v2 的查询用户列表的 API 接口以满足新的业务需求，此时，客户端的产品的新功能将请求新的服务端的 API 接口地址。虽然服务端会同时兼容多个版本，但是同时维护太多版本对于服务端而言是个不小的负担，因为服务端要维护多套代码。这种情况下，常见的做法不是维护所有的兼容版本，而是只维护最新的几个兼容版本，例如维护最新的三个兼容版本。在一段时间后，当绝大多数用户升级到较新的版本后，废弃一些使用量较少的服务端的老版本 API 接口版本，并要求使用产品的非常旧的版本的用户强制升级。注意的是，“不改变版本 v1 的查询用户列表的 API

接口”主要指的是对于客户端的调用者而言它看起来是没有改变。而实际上，如果业务变化太大，服务端的开发人员需要对旧版本的 API 接口使用适配器模式将请求适配到新的 API 接口上。

有趣的是，GraphQL 提供不同的思路。GraphQL 为了解决服务 API 接口爆炸的问题，以及将多个 HTTP 请求聚合成了一个请求，提出只暴露单个服务 API 接口，并且在单个请求中可以进行多个查询。GraphQL 定义了 API 接口，我们可以在前端更加灵活调用，例如，我们可以根据不同的业务选择并加载需要渲染的字段。因此，服务端提供的全量字段，前端可以按需获取。GraphQL 可以通过增加新类型和基于这些类型的新字段添加新功能，而不会造成兼容性问题。



此外，在使用 RPC API 过程中，我们特别需要注意兼容性问题，二方库不能依赖 parent，此外，本地开发可以使用 SNAPSHOT，而线上环境禁止使用，避免发生变更，导致版本不兼容问题。我们需要为每个接口都应定义版本号，保证后续不兼容的情况下可以升级版本。例如，Dubbo 建议第三位版本号通常表示兼容升级，只有不兼容时才需要变更服务版本。

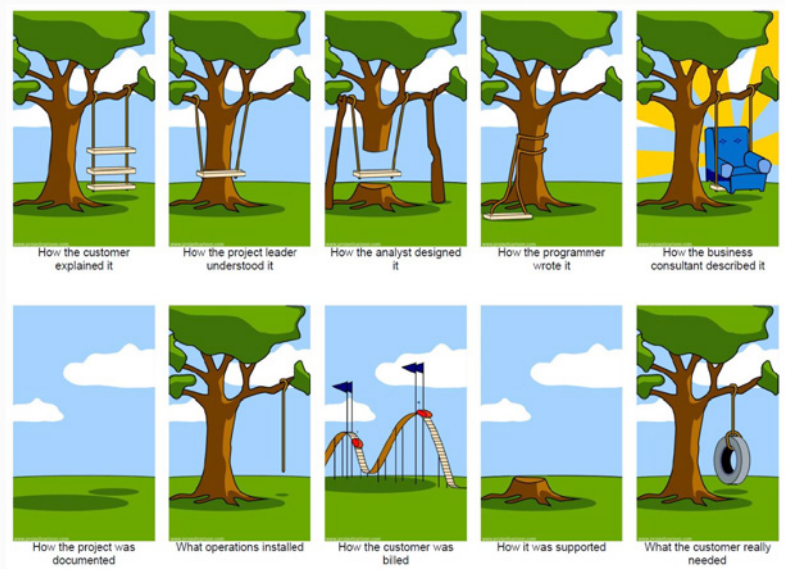
关于规范的案例，我们可以看看 k8s 和 github，其中 k8s 采用了 RESTful API，而 github 部分采用了 GraphQL。

- https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/
- https://developer.github.com/v4/

三、提供清晰的思维模型

所谓思维模型，我的理解是针对问题域抽象模型，对域模型的功能有统一认知，构建某个问题的现实映射，并划分好模型的边界，而域模型的价值之一就是统一思想，明确边界。假设，大家没有清晰的思维模型，那

么也不存在对 API 的统一认知，那么就很可能出现下面图片中的现实问题。



四、以抽象的方式屏蔽业务实现

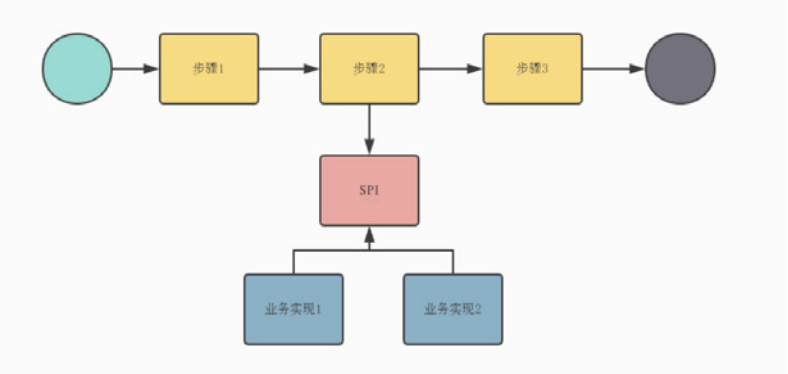
我认为好的 API 接口具有抽象性，因此需要尽可能的屏蔽业务实现。那么，问题来了，我们怎么理解抽象性？对此，我们可以思考 java.sql.Driver 的设计。这里，java.sql.Driver 是一个规范接口，而 com.mysql.jdbc.Driver。

则是 mysql-connector-java-xxx.jar 对这个规范的实现接口。那么，切换成 Oracle 的成本就非常低了。

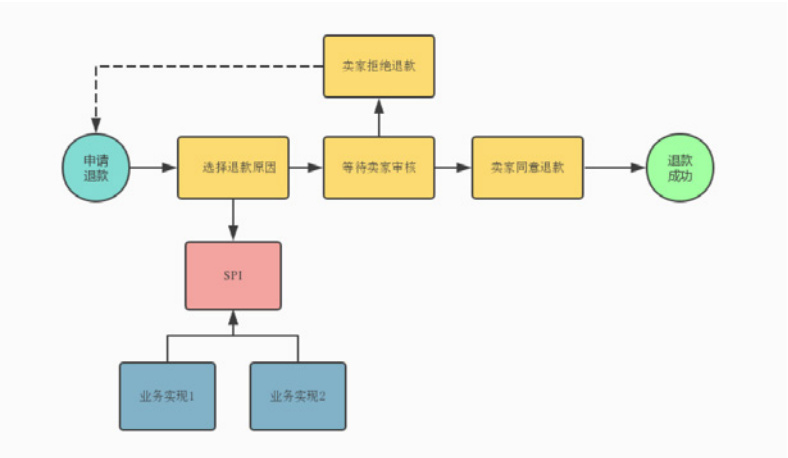
一般情况下，我们会通过 API 对外提供服务。这里，API 提供服务的接口的逻辑是固定的，换句话说，它具有通用性。但是，但我们遇到具有类似的业务逻辑的场景时，即核心的主干逻辑相同，而细节的实现略有不同，那我们该何去何从？很多时候，我们会选择提供多个 API 接口给不同的业务方使用。事实上，我们可以通过 SPI 扩展点来实现的更加优雅。什么是 SPI？ SPI 的英文全称是 Serivce Provider Interface，即服务提供者接口，它是一种动态发现机制，可以在程序执行的过程中去动态的发现某个扩展点的实现类。因此，当 API 被调用时会动态加载并调用 SPI 的特定实现方法。

此时，你是不是联想到了模版方法模式。模板方法模式的核心思想是定义骨架，转移实现，换句话说，它通过定义一个流程的框架，而将一些步骤的具体实现延迟到子类中。事实上，在微服务的落地过程中，这种思

想也给我们提供了非常好的理论基础。



现在，我们来看一个案例：电商业务场景中的未发货仅退款。这种情况在电商业务中非常场景，用户下单付款后由于各种原因可能就申请退款了。此时，因为不涉及退货，所以只需要用户申请退款并填写退款原因，然后让卖家审核退款。那么，由于不同平台的退款原因可能不同，我们可以考虑通过 SPI 扩展点来实现。



此外，我们还经常使用工厂方法 + 策略模式来屏蔽外部的复杂性。例如，我们对外暴露一个 API 接口 getTask(int operation)，那么我们就可以通过工厂方法来创建实例，通过策略方法来定义不同的实现。

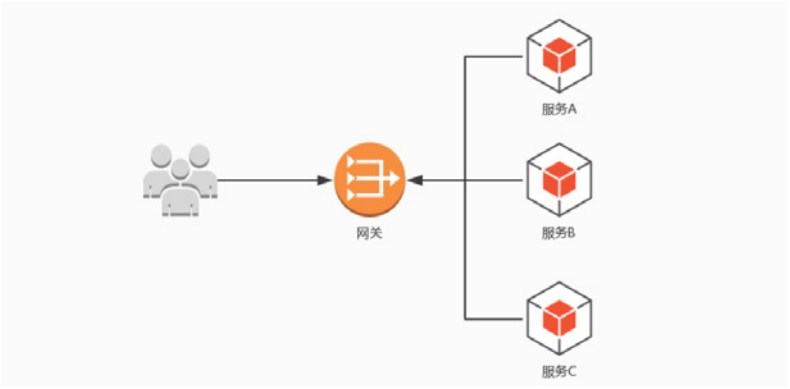
```
@Component
public class TaskManager {
    private static final Logger logger = LoggerFactory.
getLogger(TaskManager.class);

    private static TaskManager instance;
```

```
    public MapInteger, ITask> taskMap = new HashMap<Integer,
ITask>();

    public static TaskManager getInstance() {
        return instance;
    }

    public ITask getTask(int operation) {
        return taskMap.get(operation);
    }

    /**
     *初始化处理过程
     */
    @PostConstruct
    private void init() {
        logger.info("init task manager");
        instance = new TaskManager();
        //单聊消息任务
           instance.taskMap.put(EventEnum.CHAT_REQ.getValue(), new
ChatTask());
        //群聊消息任务
           instance.taskMap.put(EventEnum.GROUP_CHAT_REQ.getValue(),
new GroupChatTask());
        //心跳任务
           instance.taskMap.put(EventEnum.HEART_BEAT_REQ.getValue(),
new HeatBeatTask());

    }
}
```

还有一种屏蔽内部复杂性设计就是外观接口，它是将多个服务的接口进行业务封装与整合并提供一个简单的调用接口给客户端使用。这种设计的好处在于，客户端不再需要知道那么多服务的接口，只需要调用这个外

观接口即可。但是，坏处也是显而易见的，即增加了服务端的业务复杂度，接口性能不高，并且复用性不高。因此，因地制宜，尽可能保证职责单一，而在客户端进行“乐高式”组装。如果存在 SEO 优化的产品，需要被类似于百度这样的搜索引擎收录，可以当首屏的时候，通过服务端渲染生成 HTML，使之让搜索引擎收录，若不是首屏的时候，可以通过客户端调用服务端 RESTful API 接口进行页面渲染。

此外，随着微服务的普及，我们的服务越来越多，许多较小的服务有更多的跨服务调用。因此，微服务体系结构使得这个问题更加普遍。为了解决这个问题，我们可以考虑引入一个“聚合服务”，它是一个组合服务，可以将多个微服务的数据进行组合。这样设计的好处在于，通过一个“聚合服务”将一些信息整合完后再返回给调用方。注意的是，“聚合服务”也可以有自己的缓存和数据库。事实上，聚合服务的思想无处不在，例如 Serverless 架构。我们可以在实践的过程中采用 AWS Lambda 作为 Serverless 服务背后的计算引擎，而 AWS Lambda 是一种函数即服务（Function-as-a-Servcie，FaaS）的计算服务，我们直接编写运行在云上的函数。那么，这个函数可以组装现有能力做服务聚合。



当然，还有很多很好的设计，我也会在陆续在公众号中以续补的方式进行补充与探讨。

五、考虑背后的性能

我们需要考虑入参字段的各种组合导致数据库的性能问题。有的时候，我们可能暴露太多字段给外部组合使用，导致数据库没有相应的索引而发生全表扫描。事实上，这种情况在查询的场景特别常见。因此，我们可以只提供存在索引的字段组合给外部调用，或者在下面的案例中，要求调用

方必填 taskId 和 caseId 来保证我们数据库合理使用索引，进一步保证服务提供方的服务性能。

```
ResultVoid> agree(Long taskId, Long caseId, Configger configger);
```

同时，对于报表操作、批量操作、冷数据查询等 API 应该可以考虑异步能力。

此外，GraphQL 虽然解决将多个 HTTP 请求聚合成了一个请求，但是 schema 会逐层解析方式递归获取全部数据。例如分页查询的统计总条数，原本 1 次可以搞定的查询，演变成了 N + 1 次对数据库查询。此外，如果写得不合理还会导致恶劣的性能问题，因此，我们在设计的过程中特别需要注意。

六、异常响应与错误机制

业内对 RPC API 抛出异常，还是抛出错误码已经有太多的争论。《阿里巴巴 Java 开发手册》建议：跨应用 RPC 调用优先考虑使用 isSuccess() 方法、“错误码”、“错误简短信息”。关于 RPC 方法返回方式使用 Result 方式的理由 : 1）使用抛异常返回方式，调用方如果没有捕获到，就会产生运行时错误。2）如果不加栈信息，只是 new 自定义异常，加入自己的理解的 error message，对于调用端解决问题的帮助不会太多。如果加了栈信息，在频繁调用出错的情况下，数据序列化和传输的性能损耗也是问题。当然，我也支持这个论点的实践拥护者。

```
public ResultXxxDTO> getXxx(String param) {
    try {
        // ...
        return Result.create(xxxDTO);
    } catch (BizException e) {
        log.error("...", e);
            return  Result.createErrorResult(e.getErrorCode(),
e.getErrorInfo(), true);
    }
}
```

在 Web API 设计过程中，我们会使用 ControllerAdvice 统一包装错误信息。而在微服务复杂的链式调用中，我们会比单体架构更难以追踪与定位问题。因此，在设计的时候，需要特别注意。一种比较好的方案是，当

RESTful API 接口出现非 2xx 的 HTTP 错误码响应时，采用全局的异常结构响应信息。其中，code 字段用来表示某类错误的错误码，在微服务中应该加上“{biz_name}/”前缀以便于定位错误发生在哪个业务系统上。我们来看一个案例，假设“用户中心”某个接口没有权限获取资源而出现错误，我们的业务系统可以响应“UC/AUTH_DENIED”，并且通过自动生成的 UUID 值的 request_id 字段，在日志系统中获得错误的详细信息。

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
{
   "code": "INVALID_ARGUMENT",
   "message": "{error message}",
   "cause": "{cause message}",
   "request_id": "01234567-89ab-cdef-0123-456789abcdef",
   "host_id": "{server identity}",
   "server_time": "2014-01-01T12:00:00Z"
}
```

七、思考 API 的幂等性

幂等机制的核心是保证资源唯一性，例如客户端重复提交或服务端的多次重试只会产生一份结果。支付场景、退款场景，涉及金钱的交易不能出现多次扣款等问题。事实上，查询接口用于获取资源，因为它只是查询数据而不会影响到资源的变化，因此不管调用多少次接口，资源都不会改变，所以是它是幂等的。而新增接口是非幂等的，因为调用接口多次，它都将会产生资源的变化。因此，我们需要在出现重复提交时进行幂等处理。那么，如何保证幂等机制呢？事实上，我们有很多实现方案。其中，一种方案就是常见的创建唯一索引。在数据库中针对我们需要约束的资源字段创建唯一索引，可以防止插入重复的数据。但是，遇到分库分表的情况是，唯一索引也就不那么好使了，此时，我们可以先查询一次数据库，然后判断是否约束的资源字段存在重复，没有的重复时再进行插入操作。注意的是，为了避免并发场景，我们可以通过锁机制，例如悲观锁与乐观锁保证数据的唯一性。这里，分布式锁是一种经常使用的方案，它通常情况下是一种悲观锁的实现。但是，很多人经常把悲观锁、乐观锁、分布式锁当作幂等机制的解决方案，这个是不正确的。除此之外，我们还可以引入状态机，通过状态机进行状态的约束以及状态跳转，确保同一个业务的流程化

执行，从而实现数据幂等。事实上，并不是所有的接口都要保证幂等，换句话说，是否需要幂等机制可以通过考量需不需要确保资源唯一性，例如行为日志可以不考虑幂等性。当然，还有一种设计方案是接口不考虑幂等机制，而是在业务实现的时候通过业务层面来保证，例如允许存在多份数据，但是在业务处理的时候获取最新的版本进行处理。

2019极客邦科技会议推荐

5

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12 (月份)

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

Geekbang. 极客邦科技 InfoQ

查看会议详情

扫码关注InfoQ公众号

Geekbang 极客邦科技 | InfoQ