

架构师

ARCHITECT



热点 | Hot

2017谷歌云大会

推荐文章 | Article

Uber优步分布式追踪技术

淘宝开放平台如何攻克技术难关

专题 | Topic

MySQL JDBC连接配置上的误区

TensorFlow产品部署经验



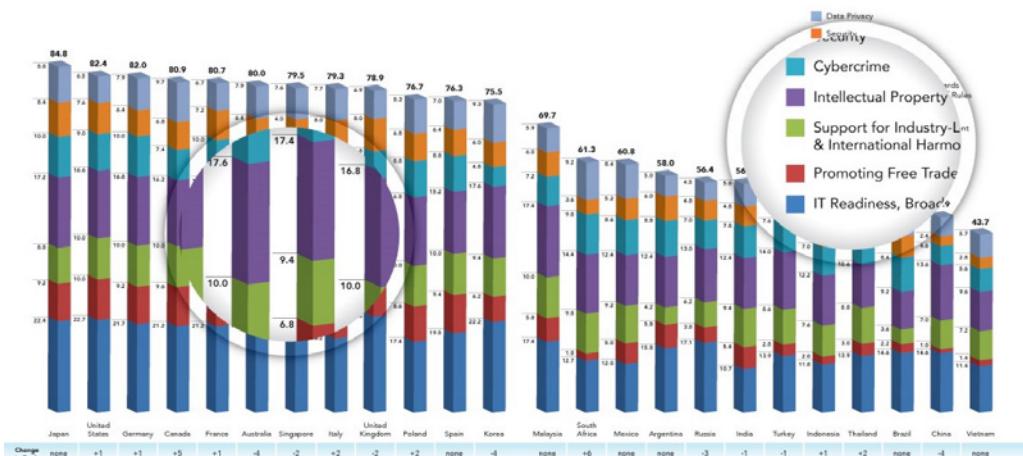
卷首语

Thinking outside the box

穆寰

几周前去日本出差，切身体会到了其发达的实体经济，即使在东京都外 40 公里的近郊，只消走数分钟就可见 24 小时营业的便利店。这是日本电商并没有流行开来的原因之一吧。

那么，电商落后是否就意味着 IT 现代化的发展不好呢？根据 BSA 软件联盟发布的 2016 年全球云计算计分卡，在共计 24 个评分国家中日本出人意料地以最高分获得榜单头名，而互联网经济发达乃至全球领先的中国仅仅位列第 23 名。BSA 解释称日本数据合规完善、宽带光纤布网广用户多；中国 IT 基础设施进步显著但是数据合规方面仍有不足。如此看来，某方面的优势并不意味着赢下全盘，还是不该太早地沾沾自喜。





虽然云计算的概念已经深入人心，底层技术原理相似相通；但是最终在各个国家的具体实施细节和表现层却有很大差异。比如，对于日常生活中手机端的网络即时消息 SaaS，落地产品的设计思路和用户体验不尽然相同：中国是 Wechat，日本、印度尼西亚、台湾和泰国是 Line，韩国则是 Kakaotalk，而其他亚洲国家和世界地区则普遍使用 Facebook、Whatsapp、Viber 等。IT 产业服务最终面向的还是每个用户个体；个体又归属于群体和文化背景，产品也因此反应了这种差异性。

直接面向用户的 IT 产业也具备着服务业的属性。通常而言要做到求同存异，底层核心技术模块可以复用，但是贴近用户的使用层模块则一定要因地制宜，倾听思考用户之所需。跨国 IT 公司必须认识和适应全球化的差异，正如饮食零售连锁巨头们都会根据当地喜好推出特别版口味产品般，AWS、Azure、阿里云的云产品组合在各个国家地区不尽然相同，谷歌搜索主页会有不同版 Doodle，LinkedIn 公司甚至比尔·盖茨个人会开通微信微博账号，Airbnb 费心费力地选取了个中文名字。

除了上述微观层面，地域差异还体现在宏观层面。Gartner 报告指出全球各个地区对 IT 费用投入有所不同。Gartner 研究副主席 John-David

Lovelock 认为 2017 年企业对 IT 支出投入的力度在世界范围内不尽相同。大部分亚太地区、撒哈拉以南非洲和大中华地区增加花费将超 4.5%，而北美和拉丁美洲增加花费将超过 2.5%，但是日本、西欧和东欧的增加花费则不会超过 2%。Gartner 预计 2017 年全球的 IT 支出将到达 3.5 万亿美元，与 2016 年相比将增加 2.7%。以此推断，虽然日本云计算发展条件相对来说最为便利，但是日本云计算全面市场爆发仍然有待时日。

不过，这也并不意味着日本 IT 界毫无生机，目前无人机广为人知的用途是拍照、摄影和运输，而此次日本拜访我看到已经开始尝试用无人机监控环境和检测高楼墙壁异常裂缝。

李开复曾经说过：纸上谈兵的理论创新是无用的，做产品必须与实际相互结合。要做有用的创新。与“实际相互结合”，也同样意味着要考虑地域的不同。并且，技术再新奇再酷炫，也需要找到接地气、真正解决痛点的应用，如此才能长久而不是如昙花一现、烟花一瞬般短暂。愿最终技术人付出的努力都可以完美地响应用户们的需求。

一起和百位技术大牛 讨论业务架构的最佳实践



7月7-8日
深圳华侨城洲际酒店

8 折购票 (限时优惠)
立减1360元



《腾讯监控创新术》

聂鑫
腾讯运维总监



《Mobile Performance at Scale》

Mike Magruder
Architect and Engineering Manager @Facebook



《创业维艰》

丁宁
荔枝FM CTO



《复杂性：架构设计中的挑战》

潘爱民
阿里巴巴安全部首席架构师



《360手机卫士性能提升攻略》

卜云涛
奇虎360手机卫士APM负责人



《Operator + Phabricator》

郭华阳
OperatorCTO



《从程序员到架构师，
从架构师到CTO》

洪强宁
爱因互动创始人&CTO



《京东容器平台与数据中心
基础设施协同发展实践》

鲍永成
京东商城基础平台部技术总监



《创业公司研发团队的迭代》

徐裕建
贝贝网合伙人&研发副总裁



《独角兽成记》

黄伟坚
Mobvista联合创始人



《人工智能开创投资新模式》

蒋龙
通联数据首席科学家
&智能大数据总监



《业务规模驱动技术升级
--京东物流系统高并发
架构演进之路》

者文明
京东运营研发部首席架构师



《从0到1再到100，创业
不同阶段的技术管理思考》

蔡锐涛
有米科技创始合伙人&CTO



《微信Android模块化架构
重构实践》

郭锐
腾讯微信Android功能开发组
高级工程师



《Apache Helix 对于在线
/离线生态系统的决定性作用》

薛君凯
Software Engineer/ Apache
Helix Commiter @LinkedIn



《Move Fast and Break Things:
Engineering at Facebook》

Joel Pobar
Engineering director
@Facebook



《海量地址的狂欢：京东智
能分单平台性能提升之路》

王梓晨
京东运营研发部系统架构师



《B2B创业中产品&技术
&销售之间的平衡之道》

刘小明
广州经传集团CTO



《Facebook的代码开发工具》

黄力菲 (Leo Huang)
Engineering Manager
@ Facebook



《存储世界，不止如此 –
EB级存储引擎背后的技术》

邹方明
腾讯存储业务中心总监



archsummit.com

CONTENTS / 目录

热点 | Hot

2017 谷歌云大会，一口气发布 100+ 消息

推荐文章 | Article

Uber 优步分布式追踪技术再度精进

历经 8 年双 11 流量洗礼，淘宝开放平台如何攻克技术难关？

观点 | Opinion

处理微服务架构的内部架构和外部架构

专题 | Topic

浅析 MySQL JDBC 连接配置上的两个误区

Zendesk 的 TensorFlow 产品部署经验

特别专栏 | Column

人工智能永恒的春天已经到来，你准备好了吗？



架构师 2017 年 4 月刊

本期主编 穆 襄

流程编辑 丁晓昀

发行人 霍泰稳

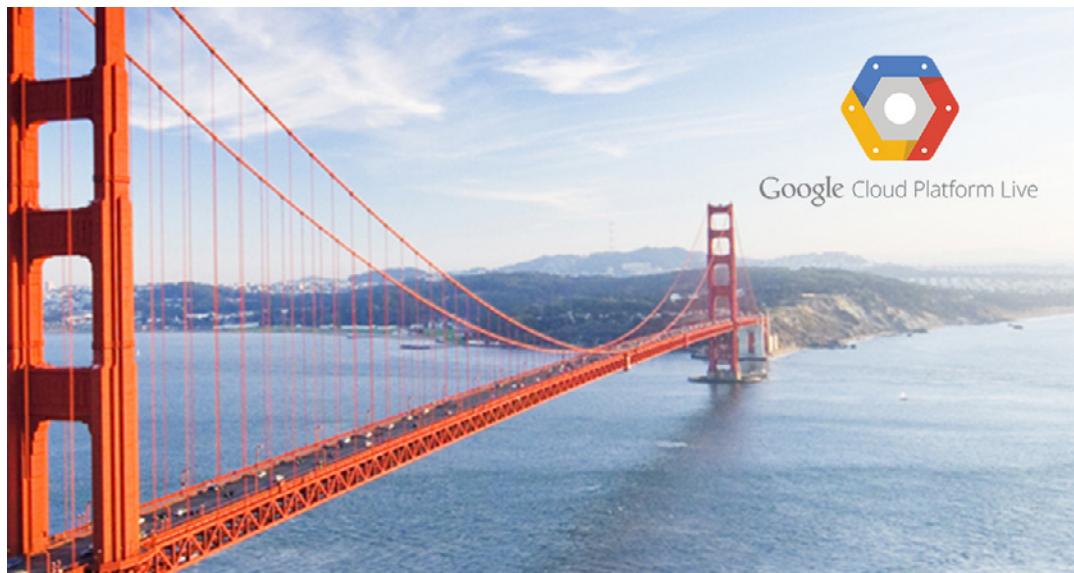
提供反馈 feedback@cn.infoq.com

商务合作 sales@cn.infoq.com

内容合作 editors@cn.infoq.com

2017 谷歌云大会，一口气发布 100+ 消息

作者 麦克周



为期 3 天的[谷歌云 2017 大会](#)(又称Next' 17 大会)已经于旧金山闭幕，超过 1 万名参与者参加了本次大会，包括谷歌的客户、合作方、开发人员、IT 领袖、工程师、媒体、分析家、云端爱好者(也包括持怀疑态度的朋友)等。在这 3 天时间里，超过 200 位分享者进行了主题演讲，另外还有 4 场邀请讲座。

Google Cloud Next '17

Next' 17 大会于 3 月 8 日在旧金山召开，3 月 10 日闭幕(下一次会议时间是 2018 年 6 月 4 日 -6 月 6 日，地点依然是旧金山)，挑选了 200 位专家从战略规划、贡献视角、技术、使用经验等多个方面阐述自己对于云端的想法。

演讲嘉宾众多，包括谷歌 CEO Sundar Pichai、谷歌云资深副总裁 Diane Greene、Alphabet 执行主席 Eric Schmidt、谷歌云首席科学家李飞飞、Linux 基金会执行总裁 Jim Zemlin、谷歌大脑及 TensorFlow 领导 Rajat Monga 等等 200 余位大牛，具体名单详见[这里](#)。

谷歌云通告集合

为了方便读者阅读最新的谷歌云变化情况，我们收集了最近几天时间里谷歌发布的所有通告信息，总计 100 项。

谷歌云并购

Kaggle: [Kaggle](#) 是全球最大的数据科学家和机器学习爱好者交流社区之一。Kaggle 和谷歌云将会继续支持机器学习培训、服务部署，也会继续提供社区存储和查询大数据集的能力。谷歌云通过本次并购，可以帮助自己笼络更多的领域专家、收集更多的有用数据。

AppBridge: 谷歌云收购了总部位于温哥华的 [AppBridge](#)，这家公司帮助用户从 on-prem 文件服务器迁移数据到谷歌的 G Suite 和 Google Drive。谷歌云通过本次收购，可以强化 AppBridge 工具，也可以将该工具纳入自身生态系统。

谷歌云安全

Identity-Aware Proxy (IAP) for Google Cloud Platform(Beta): [Identity-Aware](#) 代理提供了基于风险的应用程序访问方式，替换 VPN 方式。它提供了全方位的安全应用程序访问方式，通过用户、验证组、部署集成的防钓鱼安全密钥等方式限制访问。

Data Loss Prevention (DLP) for Google Cloud Platform (Beta): 数据防丢失 API 提供了扫描超过 40 种敏感数据类型数据的能力，被用作 Gmail 和 Drive 里 [DLP](#) 的一部分。你可以查找并编辑存储在 GCP 里面的敏感数据，支持老的应用程序使用敏感数据传感器，支持使用预定义的探测

器以及自定义工具。

Key Management Service ([KMS](#)) for Google Cloud Platform (GA): 密钥管理服务允许你生成、使用、转换和销毁云端使用的对称加密密钥。

Security Key Enforcement ([SKE](#)) for Google Cloud Platform (GA): 允许你要求安全密钥作为双重验证因子，增强 GCP 应用被访问时的反钓鱼安全。

Vault for Google Drive (GA) : 谷歌 [Vault](#) 是对于 G Suite 的 eDiscovery 和归档方案。Vault 提供了轻松管理 G Suite 数据生命周期和检索的方式，允许在站点内部预览和导出 G Suite 数据。

Google-designed security chip, [Titan](#): 谷歌使用 Titan 构建可以信赖的硬件，使得用户可以在硬件级别安全地识别和验证合法访问。Titan 包括一个随机硬件码数字生成器，在隔离内存环境下执行加密操作，并且具有专用的安全处理器（芯片上）。

谷歌云平台：数据分析

BigQuery Data Transfer Service (Private Beta): [BigQuery](#) 数据传输服务让用户快速地从所有谷歌管理的广告数据集里拿到数据。只需要点击几次鼠标，你就可以预约从 Google Adwords、DoubleClick Campaign Manager、DoubleClick for Publishers and YouTube 的数据导出服务。

Cloud Dataprep (Private Beta): [Cloud Dataprep](#) 是一个全新的数据服务管理，与 Trifacta 协同工作，对于 BigQuery 终端用户提供快速和方便的可视化操作，在不需要使用专门的数据工程师资源的情况下，为分析探索和准备数据。

New Commercial Datasets: 企业家通常需要查看自身业务领域以外的数据（公开的或者商业的）。[Commercial Datasets](#) 提供的数据包括从 Xignite 来的金融市场数据，从 HouseCanary 来的房地产数据，从 Remine 来的房产消息预测信息，从 AccuWeather 来的历史天气数据，以及从 Dow

Jones 来的新闻归档信息，所有这些信息都已经在 BigQuery 里就位。

Python for Google Cloud Dataflow in GA: [Cloud Dataflow](#) 是一个数据处理服务，支持处理批量和流式数据。这次发布之前，对于 Java 开发人员这些特性已经可以使用。本次发布带来了 Python SDK。

Stackdriver Monitoring for Cloud Dataflow ([Beta](#)): 让 Cloud Dataflow 与 Stackdriver Monitoring 进行了集成。

Google [Cloud Datalab](#) in GA: 这是一款交互式数据科学工作流工具，使用标准 SQL、Python、Shell 命令可以轻松地在 Jupyter 笔记本上执行模型迭代、分析数据。

Cloud [Dataproc updates](#): 对于运行 Apache Spark、Flink、Hadoop 管道的管理服务提供了新的支持，支持创建轻量级部署的单节点集群，提供了 GPU 支持。

谷歌云平台：数据库服务

Cloud SQL for PostgreSQL ([Beta](#)): 提供了与针对 MySQL 的云端 SQL一样的功能。

Microsoft [SQL Server Enterprise](#) (GA): 谷歌计算引擎支持 Windows 服务器灾备集群 (WSFC) 和 SQL 服务器灾备能力。

Cloud SQL for [MySQL improvements](#): 32-core 情况下提升到 208GB 内存，提升工作流性能，通过身份和访问管理控制资源中央管理。

Cloud Spanner: 将超级可扩展的 [Spanner](#) 数据库移植到了谷歌云平台，该数据库具备关系型数据库管理方式、全局强一致性特征、跨大洲扩展及灾备能力。

SSD persistent-disk performance improvements: [SSD persistent disks](#) 提升了吞吐量和 IOPs 性能，对于数据库和分析工作流有一定的益处。

Federated query on [Cloud Bigtable](#): 针对要求低延时和高吞吐的大批量分析或者操作流，扩展 BigQuery 的搜索数据范围覆盖 Cloud Bigtable、NoSQL 数据库服务。

谷歌云平台：机器学习服务

Cloud [Machine Learning Engine](#) (GA) : 云端机器学习引擎，为组织提供云端生产环境训练和部署模型能力。

Cloud [Video Intelligence API](#) (Private Beta): 允许开发人员快速地搜索和发现视频内容，只需要提供实物信息，例如“狗”、“花”、“人类”类似的名词，或者“跑步”、“游泳”、“飞行”类似的动词。

Cloud [Vision API](#) (GA): 对于企业和合作伙伴提供了更为多样化的图片分类方式。

Machine learning Advanced Solution Lab ([ASL](#)): 为客户提供专门的设施，直接与谷歌的机器学习专家合作，应用机器学习算法到它们的业务领域。

Cloud [Jobs API](#): 一个功能强大的工作搜索和发现 API。

Machine Learning [Startup Competition](#): 提供了来自 a16z、Greylock Partners、GV、Kleiner Perkins Caufield & Byers 和 Sequoia Capital 等的额外支持。

谷歌云平台：资金和支持

Compute Engine [price cuts](#): 继续价格领导地位，已经对谷歌计算引擎费用降价 8%。

Committed [Use Discounts](#): 客户可以收到最高 57% 的折扣，换取 1 到 3 年的每月支付方式，没有前期费用。

Free trial [extended to 12 months](#): 延长使用期限从 60 天到 12 个月。

Engineering [Support](#): 转变为工程师支持工程师模式，根据你的实际业务，无论你是在部署的哪一个阶段，都可以找到对应工程师支持。

Cloud.google.com/[community site](#): 新推出的专用于分享 GCP 使用经验的一个社区。

谷歌云平台：开发者平台和工具

Google [AppEngine Flex](#) (GA)：对于 App 引擎平台提供了更多的开放性、开发人员选择项、应用程序可移植性等。

Cloud [Functions](#) (Beta)：无服务器环境，用于创建事件驱动应用程序和微服务，让你可以使用代码构建和连接云端服务。

Firebase [integration](#) with GCP (GA)：与 Google Cloud Functions 集成。

Cloud [Container Builder](#)：是一个单一工具，允许用户在 GCP 上构建 Docker 容器，无论部署环境是什么。它是一种快速、可信赖的、一致性的打包软件进入容器工具，是自动化工作流的一部分。

Community [Tutorials](#) (Beta)：任何人都可以提交或者请求一篇关于谷歌云平台的技术文章。

谷歌云平台：基础设施

New data center region: [California](#): 新的 GCP 数据中心，负责美国西海岸和相邻的地理区域。类似于其他的谷歌云区，它将至少包含三个分区特性，受益于谷歌全球化、私有云光钎网络，并提供 GCP 服务的补充。

New data center region: [Montreal](#): 新的 GCP 数据中心，负责加拿大和相邻的地理区域。类似于其他的谷歌云区，它将至少包含三个分区特性，受益于谷歌全球化、私有云光钎网络，并提供 GCP 服务的补充。

New data center region: [Netherlands](#): 新的 GCP 数据中心，负责欧洲西部和相邻的地理区域。类似于其他的谷歌云区，它将至少包含三个分区特性，受益于谷歌全球化、私有云光钎网络，并提供 GCP 服务的补充。

Google Container Engine: [Managed Nodes](#): 新增自动化检测和修复 GKE 节点能力，Google 确保用户集群是可用的和最新版本。

64 [Core machines + more memory](#): 每一个实例可以运行在 64 颗 vCPUs 上，数量级翻了一倍，并且支持最大 416GB 内存。

Internal [Load balancing](#) (GA): 内部负载均衡，允许用户服务内部伸缩，基于内部实例可以访问的私有负载均衡 IP 地址。

Cross-[Project Networking](#) (Beta): 跨越多个谷歌云平台的项目的虚拟网络，提供了通用网络体验，启用简单的多租户部署。

G Suite: 企业协作 & 生产力

Team Drives ([GA for G Suite Business, Education and Enterprise customers](#)): 提供简易且安全的方式管理权限、所有者，以及文件访问。

Drive File Stream ([EAP](#)): 加速从云端下载流式文件到本地计算机的方式，支持笔记本没有充足硬盘空间的情况下直接访问公司数据。

Google Vault for Drive ([GA for G Suite Business, Education and Enterprise customers](#)): 支持管理员控制所有文件及其安全策略，包括员工 Drives 和团队 Drives。

Quick Access in [Team Drives](#) (GA): 基于谷歌机器智能，帮助员工第一时间得到准确的信息。

Hangouts Meet ([GA to existing customers](#)): 基于 Hangouts 的视频会议系统，可以举办不超过 30 人的视频会议，无需账号、插件，也不需要下载程序。

Hangouts Chat ([EAP](#)): 一款基于 Hangouts 的智能通讯 App，可以在跨企业团队之间建立虚拟办公室。

@meet: 基于 Hangouts 平台之上构建的一款[智能机器人](#)。

Gmail Add-ons for G Suite ([Developer Preview](#)): 提供了一种在 Gmail 内部访问 App 或者服务功能的方式。

Edit [Opportunities](#) in Google Sheets: 支持从 Salesforce 同步信息到 excel 表单，并且更新会自动同步至 Salesforce。

[Jamboard](#): 合并了物理世界和数据创新。

Android & Chrome 设备

[Android Kiosk Apps for Chrome](#): 允许用户在Web和安卓应用上管理和部署Chrome数字signage、kiosks。

[Chrome Kiosk Management Free trial](#): 提供免费试用Chromesignage、kiosks部署方式。

Chrome Device Management ([CDM](#)) APIs for Kiosks: 提供了Kiosk的API。

[Chrome Stability API](#): 帮助Kiosk的App开发人员提升应用程序的可用性。

谷歌云客户

[Colgate](#): 在谷歌云和 SAP 之间架设协作桥梁和工具。

[Disney Consumer Products & Interactive \(DCPI\)](#): 提升下一代机器学习的用户体验。

[eBay](#): eBay 目前在 ShopBot 上使用的谷歌云技术包括谷歌容器引擎、机器学习和人工智能

[HSBC](#): 部署云端 DataFlow、BigQuery，以及其他数据服务。

[LUSH](#): 花费少于 6 周时间，从 AWS 迁移全球电子商务网站到 GCP，极大地提升了站点的可用性和稳定性。

[Oden Technologies](#): 从 AWS 上迁移整个平台到 GCP。

[Planet](#): 2 月份已经迁移到 GCP。

[Schlumberger](#): 使用 GCP 的高性能计算、远程虚拟化、快速部署等特性。

[The Home Depot](#): 为了黑色星期五和黑客工具而迁移 HomeDepot.com 到云端。

[Verizon](#): 为超过 15 万名员工部署 G Suite，在维持安全性和约束标准的情况下，在办公环境内更好地协作和弹性工作。

[Alooma](#): Alooma 和 Google Cloud SQL、BigQuery 集成完毕。

[Authorized Training Partner Program](#): 为了帮助企业更快地扩大它们的培训产品，也让其他的培训伙伴加入到谷歌生态系统，谷歌正在推出一个全新的合作伙伴计划，以支持它们特有的产品和需求。

[Check Point](#): 发布针对谷歌云平台的 [Check Point](#) vSEC，与 GCP 集成高安全机制，并且加入谷歌云技术伙伴项目。

[CloudEndure](#): 与 [CloudEndure](#) 合作，为 GCP 客户提供一种免费的自助迁移工具。

[Coursera](#): 与谷歌云平台合作，提供涵盖广阔的谷歌云培训课程。为 GCP 基础课程提供 100% 的折扣。

[DocuSign](#): 与谷歌 Docs 深度集成。

[Egnyte](#): 与谷歌 Docs 加强集成，允许双方共同的客户，在 Egnyte 连接内创建、编辑和存储谷歌 Docs、Sheets 以及 Slides 文件。

[Google Cloud Global Partner Awards](#): 授予 12 家在 2016 年建立了强大的客户关系和解决创新方案上取得重大成功的谷歌云伙伴奖项，包括 Accenture、Pivotal、LumApps、Slack、Looker、Palo Alto Networks、Virtru、SoftBank、DoIT、Snowdrop Solutions、CDW Corporation，以及 SYNEX Corporation。

[iCharts](#): 宣布对几个 GCP 数据库提供额外的支持，对于当前谷歌 BigQuery 用户免费使用 pivot 表格，以及一个全新的产品“iCharts for SaaS”。

[Intel](#): 除了 Skylake 进展以外，Intel 和谷歌云推出了几个技术措施和市场培养计划，涵盖物联网、Kubernetes 和 TensorFlow，包括优化、开发程序和工具。

[Intuit](#): 发布 Gmail Add-Ons，其目的是基于邮件上下文整合自定义 Gmail 工作流。

[Liftigniter](#): 谷歌云启动程序会员，聚焦于个性化机器学习，使用预测分析方式提升网站或者 App 的点击率。

[Locker](#): 发布一系列产品，与谷歌 BigQuery 数据传输服务兼容，旨在

让营销人员提升关键数据分析能力。

[Low interest loans for partners](#): 为了帮助白金会员团队的生长，谷歌宣布将会向符合条件的合作伙伴提供低利息的贷款业务。

[MicroStrategy](#): 宣布与谷歌云SQL针对PostgreSQL和MySQL集成。

[New incentives to accelerate partner growth](#): 对于已经存在的或者新兴项目提供多方面投资。

[Orbitera Test Drives for GCP Partners](#): 允许客户使用软件，生成高质量的数据，这些数据可以被直接发给合作伙伴的销售团队。谷歌为白金会员提供了一年的免试使用套餐。

[Partner specializations](#): 在客户关系、技术影响等众多方面取得成功的合作伙伴，现在可以向谷歌申请授予专家称号。谷歌专家覆盖应用程序开发、数据分析、机器学习，以及基础设施。

[Pivotal](#): GCP宣布Pivotal成为首个CRE技术合作伙伴。

[ProsperWorks](#): 发布Gmail Add-Ons，，其目的是基于邮件上下文整合自定义Gmail工作流。

[Qwiklabs](#): 本次收购将向授权的培训合作伙伴提供各种能力，包括提供动手实验室、谷歌专家开发的综合性课程。

[Rackspace](#): 宣布与谷歌云达成战略伙伴关系。

[Rocket.Chat](#): 增加了一系列与HCP集成的新产品，包括Autotranslate via Translate API。

[Salesforce](#): 发布Gmail Add-Ons，，其目的是基于邮件上下文整合自定义Gmail工作流。

SAP: 战略伙伴关系包括授权[SAP HANA](#)运行在GCP上，新的G Suite集成以及未来在机器学习属性构建上的合作。

Smyte: [Smyte](#)最近从自己运营的Kubernetes移植到了谷歌容器云引擎上（GKE）。

[Veritas](#): 扩大其与谷歌云的合作伙伴关系，提供共有客户360数据管理能力。

[VMware Airwatch](#): 提供企业对于Android的移动管理解决方案，并且持续驱动谷歌设备生态环境符合企业级客户需求。

[Windows Partner Program](#): 与Windows社区的顶级系统集成商合作，帮助GCP客户获得Windows、.Net App、运行在Windows上的服务的所有优点。

[Xplenty](#): 宣布集成两个新的谷歌云服务，它们分别是Google Cloud Spanner、Google Cloud SQL for PostgreSQL。

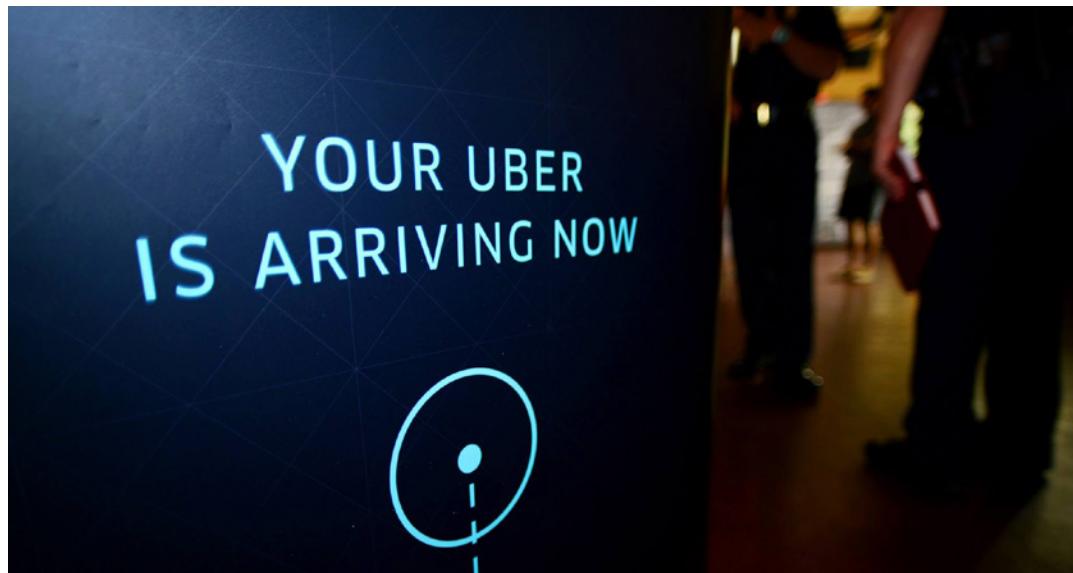
[Zoomdata](#): 宣布支持GCP上的Google Cloud Spanner和PostgreSQL，进一步加强了Zoomdata Smart Connector对于谷歌BigQuery的支持。与谷歌云平台深度集成、优化支持其Cloud Spanner、PostgreSQL、Google BigQuery、以及Cloud DataProc services等服务。

结论

本次大会期间一共发布了超过100项新的并购消息、技术升级、平台扩容、新社区，战略规划之准确、大胆，谷歌的执行速度之快，视野之广阔，都是国内企业应该积极学习的，希望能够在谷歌的引领下，更多的科技企业能够与谷歌并肩。

优步分布式追踪技术再度精进

作者 Yuri Shkuro 译者 大愚若智



对于希望监视复杂的[微服务架构](#)系统的组织，分布式追踪正在快速成为一种不可或缺的工具。Uber 工程团队的开源分布式追踪系统 [Jaeger](#) 自 2016 年起，在公司内部实现了大范围的运用，已经集成于数百个微服务中，目前每秒钟已经可以记录数千条追踪数据。新年伊始，我们想向大家介绍一下这一切是如何实现的，从我们最开始使用现成的解决方案，如 Zipkin，到我们从拉取转换为推送架构的原因，以及 2017 年有关分布式追踪的发展计划。

从整体式到微服务架构

随着 Uber 的业务飞速增长，软件架构的复杂度也与日俱增。大概一年多前，2015 年秋季，我们有大约 500 个微服务，2017 年初这一数量已

增长至超过 2000 个。这样的增幅部分是由于业务该功能的增加，例如面向用户的 [UberEATS](#) 和 [UberRUSH](#) 等功能，以及类似欺诈检测、数据挖掘、地图处理等内部功能的增加。此外随着我们从大规模整体式应用程序向着分布式微服务架构迁移，也造成了复杂度的增加。

迁移到微服务生态总是会遇到独特的挑战。例如丧失对系统的能见度，服务之间开始产生复杂的交互等。[Uber 工程团队](#)很清楚，我们的技术会对大家的生活产生直接影响，系统的可靠性至关重要，但这一切都离不开“可观测性”这一前提。传统的[监视工具](#)，例如度量值和分布式日志依然发挥着自己的作用，但这类工具往往无法提供跨越不同服务的能见度。分布式追踪应运而生。

Uber 最初的追踪系统

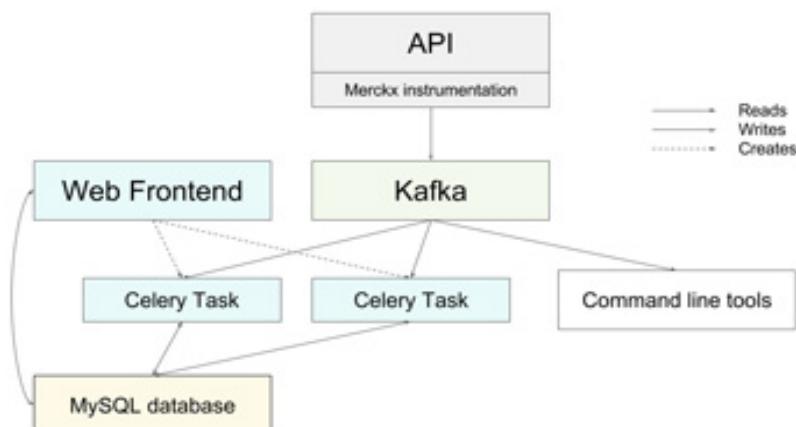
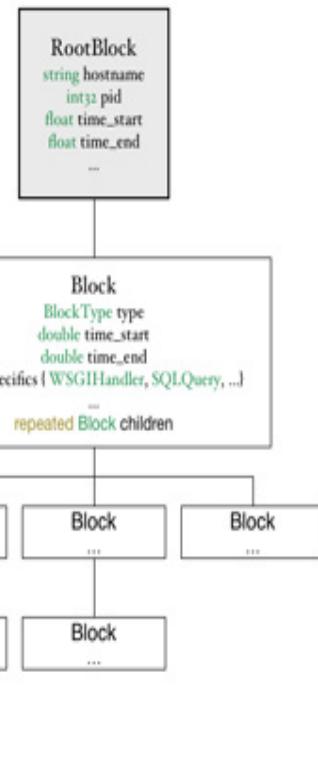
Uber 最初广泛使用的追踪系统叫做 Merckx，这一名称源自全球速度最快的[自行车骑行选手](#)。Merckx 很快就帮助我们了解了有关 Uber 基于 Python 的整体式后端的很多问题。我们可以查询诸如“查找已登录用户的请求，并且请求的处理时间超过 2 秒钟，并且使用了某一数据库来处理，并且事务维持打开状态的时间超过 500ms”这样的问题。所有待查询的数据被组织成树状块，每个块代表某一操作或某个远程调用，这种组织方式类似于[OpenTracing API](#) 中“Span”这个概念。用户可以在[Kafka](#) 中使用命令行工具针对数据流执行即席查询，也可以使用 Web 界面查看预定义的摘要，这些信息均从 API 端点的高级别行为和[Celery](#) 任务中摘要汇总而来。

Merckx 使用了一种类似于树状块的调用图，每个块代表应用程序中的一个操作，例如数据库调用、RPC，甚至库函数，例如解析 JSON。

Merckx 的编排调度可自动应用于使用 Python 编写的一系列基础架构库，包括 HTTP 客户端和服务器、SQL 查询、Redis 调用，甚至 JSON 的序列化。这些编排调度可记录有关每次操作的某些性能度量值和元数据，例如 HTTP 调用的 URL，或数据库调用的 SQL 查询。此外还能记录其他信息，例如数据库事务维持打开状态的时长，访问了哪些数据库 Shard 和副本。

Merckx 架构使用了拉取模式，可从 Kafka 的指令数据中拉取数据流。

Merckx 最大的不足在于其设计主要面向 Uber 使用整体式 API 的年代。Merckx 缺乏分布式上下文传播的概念，虽然可以记录 SQL 查询、[Redis](#) 调用，甚至对其他服务的调用，但无法进一步深入。Merckx 还有另一个有趣的局限：因为 Merckx 数据存储在一个全局线程本地存储中，诸如数据库事务追踪等大量高级功能只能在 uWSGI 下使用。随着 Uber 开始使用 Tornado（一种适用于 Python 服务的异步应用程序框架），线程本地存储无法体现 Tornado 的 IOLoop 中同一个线程内运行的大部分并发请求。我们开始意识到不借助全局变量或全局状态，转为通过某种方式保存请求状态，并进行恰当的传播的重要性。



使用 TChannel 进行追踪

2015 年初，我们开始开发 [TChannel](#)，这是一种适用于 RPC 的网络多路复用和框架协议。该协议的设计目标之一是将类似于 [Dapper](#) 的分布式追踪能力融入协议中，并为其提供最优秀的支持。为了实现这一目标，[TChannel 协议规范](#)将追踪字段直接定义到了二进制格式中。

`spanid:8 parentid:8 traceid:8 traceflags:1`

| 字段 | 类型 | 描述 |
|------------|-------|-------------|
| spanid | int64 | 用于识别当前 span |
| parentid | int64 | 前一个 span |
| traceid | int64 | 负责分配的原始操作方 |
| traceflags | uint8 | 位标志字段 |

追踪字段作为二进制格式的一部分已包含在 TChannel 协议规范中。

除了协议规范，我们还发布了多个开源客户端库，用于以不同语言实现该协议。这些库的设计原则之一是让应用程序需要用到的请求上下文这一概念能够从服务器端点贯穿至下游的调用站点。例如在 [tchannel-go](#) 中，让出站调用使用 [JSON 进行编码](#) 的签名需要通过第一个参数提供上下文：

```
func (c *Client) Call(ctx Context, method string, arg, resp
interface{}) error {..}
```

Tchannel 库使得应用程序开发者在编写自己的代码时始终将分布式上下文传播这一概念铭记于心。

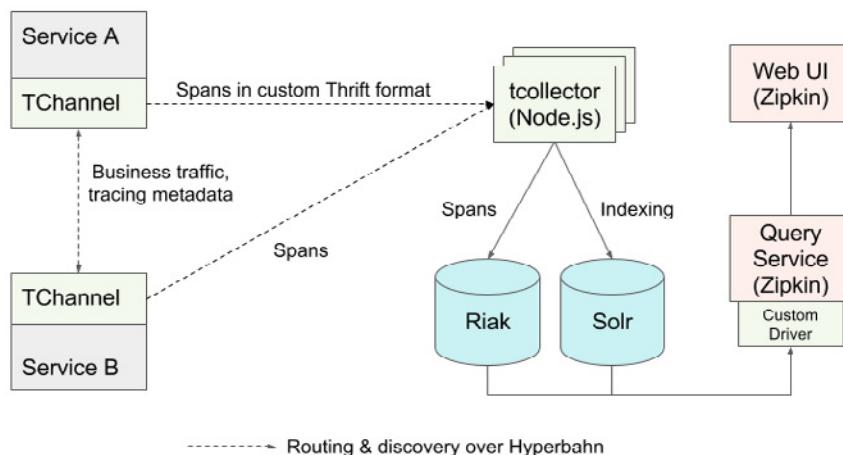
通过将所传输内容以及内存中的上下文对象之间的追踪上下文进行安排，并围绕服务处理程序和出站调用创建追踪 Span，客户端库内建了对分布式追踪的支持。从内部来看，这些 Span 在格式上与 [Zipkin](#) 追踪系统几乎完全相同，也使用了 Zipkin 所定义的注释，例如“cs”（Client Send）和“cr”（Client Receive）。Tchannel 使用追踪报告程序（Reporter）接口将收集到的进程外追踪 Span 发送至追踪系统的后端。该技术自带的库默认包含一个使用 Tchannel 本身和 Hyperbahn 实现的报告程序以及发现和路由层，借此将 Thrift 格式的 Span 发送至收集器群集。

Tchannel 客户端库已经比较近似于我们所需要的分布式追踪系统，

该客户端库提供了下列构建块：

- 追踪上下文的进程间传播以及带内请求
- 通过编排API记录追踪Span
- 追踪上下文的进程内传播
- 将进程外追踪数据报告至追踪后端所需的格式和机制

该系统唯独缺少了追踪后端本身。追踪上下文的传输格式和报表程序使用的默认 Thrift 格式在设计上都可以非常简单直接地将 Tchannel 与 Zipkin 后端集成，然而当时只能通过 [Scribe](#) 将 Span 发送至 Zipkin，而 Zipkin 只支持使用 [Cassandra](#) 格式的数据存储。此外当时我们对这些技术没什么经验，因此我们开发了一套后端原型系统，并结合 Zipkin UI 的一些自定义组件构建了一个完整的追踪系统。



后端原型系统架构：Tchannel 生成的追踪记录推送给自定义收集器、自定义存储，以及开源的 Zipkin UI。

分布式追踪系统在谷歌和 Twitter 等主要技术公司获得的成功意味着这些公司中广泛使用的 RPC 框架、Stubby 和 [Finagle](#) 是行之有效的。

同理，Tchannel 自带的追踪能力也是一个重大的飞跃。我们部署的后端原型系统已经开始从数十种服务中收集追踪信息。随后我们使用 Tchannel 构建了更多服务，但在生产环境中全面推广和广泛使用依然有些困难。该后端原型以及所使用的 Riak/Solr 存储系统无法妥善缩放以

适应 Uber 的流量，同时很多查询功能依然无法与 Zipkin UI 实现足够好的互操作。尽管新构建的服务大量使用了 Tchannel，Uber 依然有大量服务尚未在 RPC 过程中使用 Tchannel，实际上承担核心业务的大部分服务都没有使用 Tchannel。这些服务主要是通过四大编程语言（Node.js、Python、Go 和 Java）实现的，在进程间通信方面使用了多种不同的框架。这种异构的技术环境使得 Uber 在分布式追踪系统的构建方面会面临比谷歌和 Twitter 更严峻的挑战。

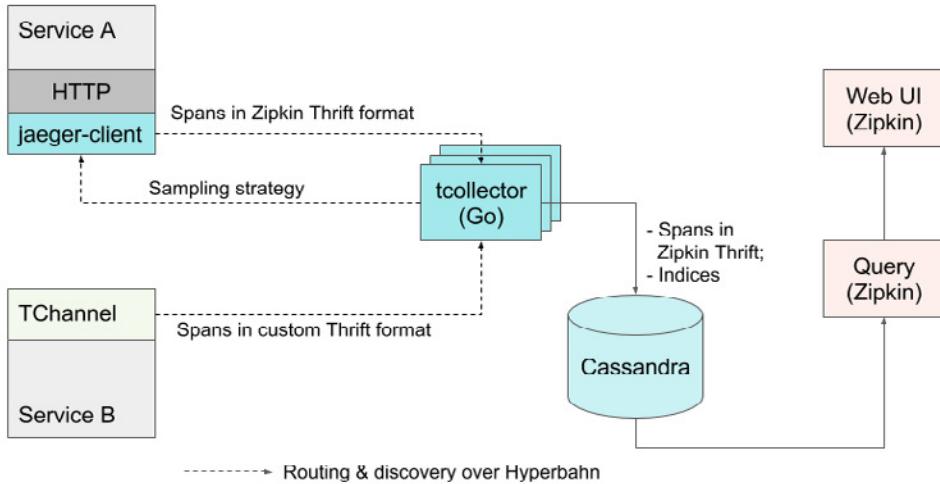
在纽约市构建的 Jaeger

Uber 纽约工程组织始建于 2015 年上半年，主要包含两个团队：基础架构端的 Observability 以及产品（包括 UberEATS 和 UberRUSH）端的 Uber Everything。考虑到分布式追踪实际上是一种形式的生产环境监视，因此更适合交由 Observability 团队负责。

我们组建了分布式追踪团队，该团队由两个工程师组成，目标也有两个：将现有的原型系统转换为一种可以全局运用的生产系统，让分布式追踪功能可以适用并适应 Uber 的微服务。我们还需要为这个项目起一个开发代号。为新事物命名实际上是计算机科学界两大老大难问题之一，我们花了几周时间集思广益，考虑了追踪、探测、捕获等主题，最终决定命名为 Jaeger，在德语中这个词代表猎手或者狩猎过程中的帮手。

纽约团队在 Cassandra 群集方面已经具备运维经验，该数据库直接为 Zipkin 后端提供着支持，因此我们决定弃用基于 Riak/Solr 的原型。为了接受 TChannel 流量并将数据以兼容 Zipkin 的二进制格式存储在 Cassandra 中，我们用 Go 语言重新实现了收集器。这样我们就可以无需改动，直接使用 Zipkin 的 Web 和查询服务，并通过自定义标签获得了原本不具备的追踪记录搜索功能。我们还为每个收集器构建了一套可动态配置的倍增系数（Multiplication factor），借此将入站流量倍增 n 次，这主要是为了通过生产数据对后端系统进行压力测试。

Jaeger 的早期架构依然依赖 Zipkin UI 和 Zipkin 存储格式。



第二个业务需求希望让追踪功能可以适用于未使用 TChannel 进行 RPC 的所有现有服务。随后几个月我们使用 Go、Java、Python 和 Node.js 构建了客户端库，借此未包括 HTTP 服务在内各类服务的编排提供支持。尽管 Zipkin 后端非常著名并且流行，但依然缺乏足够完善的编排能力，尤其是在 Java/Scala 生态系统之外的编排能力。我们考虑过各种开源的编排库，但这些库是由不同的人维护的，无法确保互操作性，并且通常还使用了完全不同的 API，大部分还需要使用 Scribe 或 Kafka 作为报表 Span 的传输机制。因此我们最终决定自行编写库，这样可以通过集成测试更好地保障互操作性，可以支持我们需要的传输机制，更重要的是，可以用不同的语言提供一致的编排 API。我们的所有客户端库从一开始都可支持 OpenTracing API。

在第一版客户端库中，我们还增加了另一个新颖的功能：可以从追踪后端轮询采样策略。当某个服务收到不包含追踪元数据的请求后，所编排的追踪功能通常会为该请求启动一个新的追踪，并生成新的随机追踪 ID。然而大部分生产追踪系统，尤其是与 Uber 的缩放能力有关的系统无法对每个追踪进行“描绘”（Profile）或将其实记录在自己的存储中。这样做会在服务与后端系统之间产生难以招架的大流量，甚至会比服务所处理的实际业务流量大出好几个数量级。我们改为让大部分追踪系统只对小比例的追踪进行采样，并只对采样的追踪进行“描绘”和记录。用于进行

采样决策的算法被我们称之为“采样策略”。采样策略的例子包括：

- 采样一切。主要用于测试用途，但生产环境中使用会造成难以承受的开销！
- 基于概率的采样，按照固定概率对特定追踪进行随机采样。
- 限速采样，每个时间单位对X个追踪进行采样。例如可能会使用漏桶（[Leaky bucket](#)）算法的变体。

大部分兼容 Zipkin 的现有编排库可支持基于概率的采样，但需要在初始化过程中对采样速率进行配置。以我们的规模，这种方式会造成一些严重的问题：

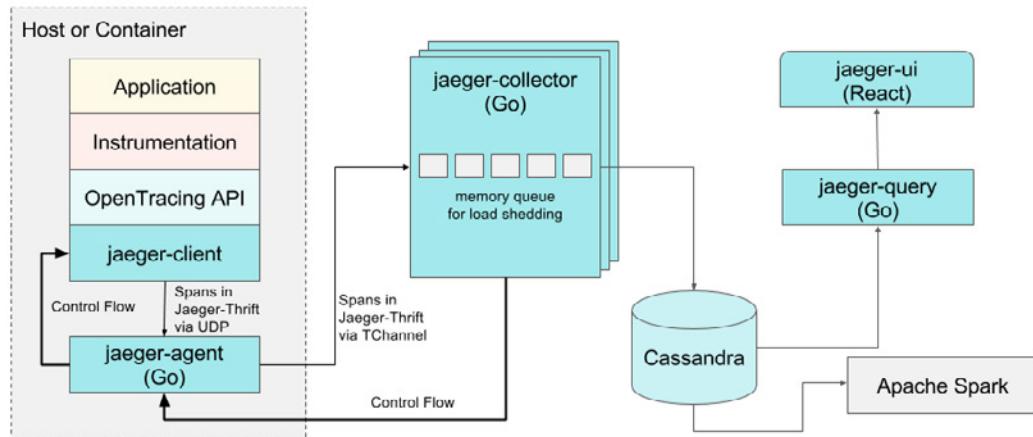
- 每个服务对不同采样速率对追踪后端系统整体流量的影响知之甚少。例如，就算服务本身使用了适度的每秒查询数（QPS）速率，也可能调用扇出（Fanout）因素非常高的其他下游服务，或由于密集编排导致产生大量追踪Span。
- 对于Uber来说，每天不同时段的业务流量有着明显规律，峰值时期乘客更多。固定不变的采样概率对非峰值时刻可能显得过低，但对峰值时刻可能显得过高。

Jaeger 客户端库的轮询功能按照设计可以解决这些问题。通过将有关最恰当采样策略的决策转交给追踪后端系统，服务的开发者不再需要猜测最适合的采样速率。而后端可以按照流量模式的变化动态地调整采样速率。下方的示意图显示了从收集器到客户端库的反馈环路。

第一版客户端库依然使用 TChannel 发送进程外追踪 Span，会将其直接提交给收集器，因此这些库需要依赖 Hyperbahn 进行发现和路由。对于希望在自己的服务中运用追踪能力的工程师，这种依赖性造成了不必要的摩擦，这样的摩擦存在于基础架构层面，以及需要在服务中额外包含的库等方面，进而可能导致[依赖性地域](#)。

为了解决这种问题，我们实现了一种 jaeger-agent 边车（Sidecar）进程，并将其作为基础架构组件，与负责收集度量值的代理一起部署到所有宿主机上。所有与路由和发现有关的依赖项都封装在这个 jaeger-

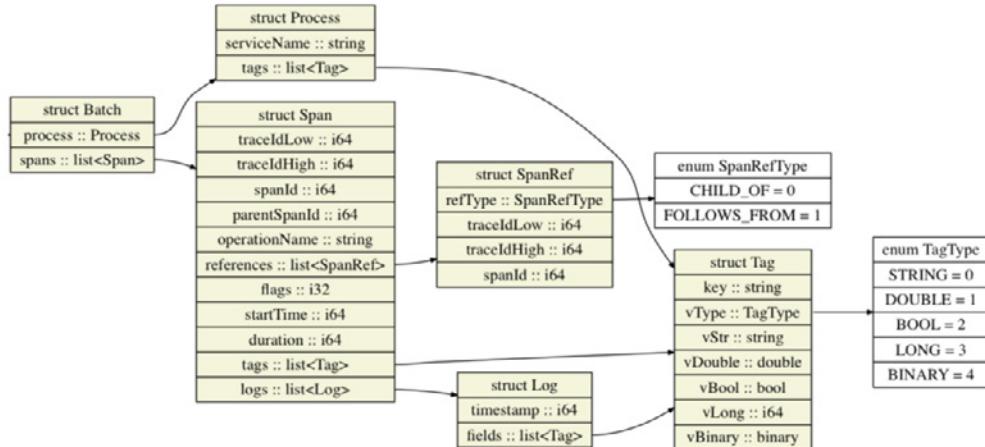
agent 中，此外我们还重新设计了客户端库，可将追踪 Span 报告给本地 UDP 端口，并能轮询本地回环接口上的代理获取采样策略。新的客户端只需要最基本的网络库。架构上的这种变化向着我们先追踪后采样的愿景迈出了一大步，我们可以在代理的内存中对追踪记录进行缓冲。



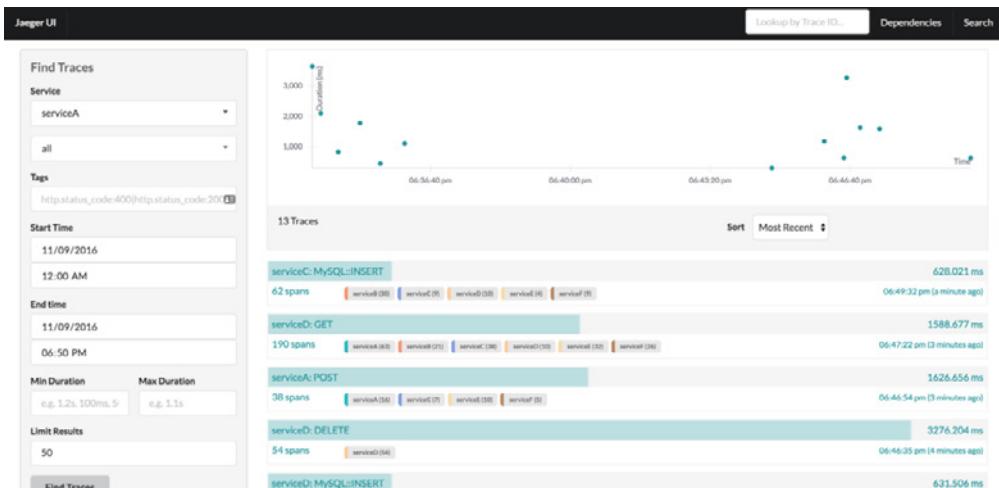
目前的 Jaeger 架构：后端组件使用 Go 语言实现，客户端库使用了四种支持 OpenTracing 标准的语言，一个基于 React 的 Web 前端，以及一个基于 Apache Spark 的后处理和聚合数据管道。

统包式分布式追踪

Zipkin UI 是我们在 Jaeger 中使用的最后一个第三方软件。由于要将 Span 以 Zipkin Thrift 格式存储在 Cassandra 中并与 UI 兼容，这对我们的后端和数据模型产生了一定的限制。尤其是 Zipkin 模型不支持 OpenTracing 标准和我们的客户端库中两个非常重要的功能：键 - 值日志 API，以及用更为通用的有向无环图（Directed acyclic graph）而非 Span 树所代表的追踪。因此我们毅然决定彻底革新后端所用的数据模型，并编写新的 UI。如下图所示，新的数据模型可原生支持键 - 值日志和 Span 的引用，此外还对发送到进程外的数据量进行了优化，避免进程标签在每个 Span 上重复：Jaeger 数据模型可原生支持键 - 值日志和 Span 引用。

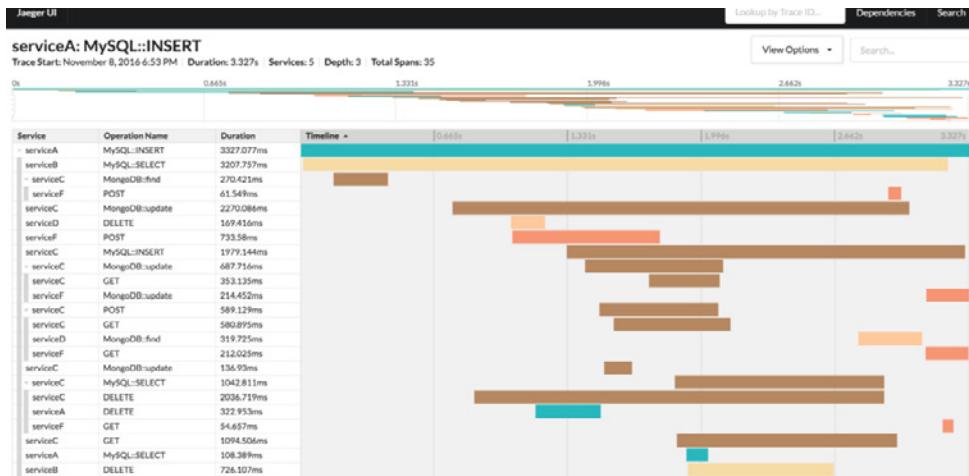


目前我们正在将后端管道全面升级到新的数据模型，以及全新的，更为优化的 Cassandra 架构。为了充分利用新的数据模型，我们还用 Go 语言实现了一个全新的 Jaeger 查询服务，并用 React 实现了一套全新的 Web UI。最初版本的 UI 主要重现了 Zipkin UI 的原有功能，但在设计上更易于通过扩展提供新的功能和组件，并能作为 React 组件嵌入到其他 UI。例如，用户可以选择用多种不同视图对追踪结果进行可视化，例如追踪时段内的直方图，或服务在追踪过程中的累积时间：



Jaeger UI 显示的追踪信息搜索结果。右上角显示的时刻和持续时间散点图用可视化方式呈现了结果，并提供了向下挖掘能力。

另一个例子，可以根据不同用例查看同一条追踪记录。除了使用默认的时序渲染方式，还可以通过其他视图渲染为有向无环图或关键路径图：



Jaeger UI 显示了一条追踪记录的详情。界面顶部是一条追踪记录的迷你地图示意图，借此可在更大规模的追踪记录中进行更轻松的导航。

通过将架构中剩余的 Zipkin 组件替代为 Jaeger 自己的组件，我们将 Jaeger 彻底变为一种统包式的端到端分布式追踪系统。

我们认为编排库是 Jaeger 固有的一部分，这样可以确保与 Jaeger 后端的兼容性，以及通过持续集成测试保障相互之间的互操作性。（Zipkin 生态系统做不到这些。）尤其是跨越所有可支持语言（目前支持 Go、Java、Python 和 Node.js）和可支持的传输方式（目前支持 HTTP 和 TChannel）实现的互操作性会在每个 Pull 请求中测试，并用到了 Uber 工程部门 RPC 团队所开发的 [Crossdock](#) 框架。Jaeger 客户端集成测试的详细信息请参阅 [jaeger-client-go](#) crossdock 代码库。目前所有 Jaeger 客户端库都已[开源](#)：

- Go
- Java
- Node.js
- Python

我们正在将后端和 UI 代码迁移至 GitHub，并计划尽快将 Jaeger 的源代码全部公开。如果你对这个过程感兴趣，可以关注[主代码库](#)。我们欢迎大家为此做贡献，也很乐于看到更多人尝试使用 Jaeger。虽然我们对目前的进展很满意，但 Uber 的分布式追踪工作还有很长的路要走。

历经 8 年双 11 流量洗礼， 淘宝开放平台如何攻克技术难关？

作者 风胜



前言

淘宝开放平台（open.taobao.com）是阿里系统与外部系统通讯的最重要平台，每天承载百亿级的 API 调用，百亿级的消息推送，十亿级的数据同步，经历了 8 年双 11 成倍流量增长的洗礼。本文将为您揭开淘宝开放平台的高性能 API 网关、高可靠消息服务、零漏单数据同步的技术内幕。

1 高性能 API 网关

阿里巴巴内部的数据分布在各个独立的业务系统中，如：商品中心、交易平台、用户中心，各个独立系统间通过 HSF（High-speed Service Framework）进行数据交换。如何将这些数据安全可控的开放给外部商家和 ISV，共建繁荣电商数据生态，在这个背景下 API 网关诞生。

1.1 总体架构

API 网关采用管道设计模式，处理业务、安全、服务路由和调用等逻辑。为了满足双 11 高并发请求（近百万的峰值 QPS）下的应用场景，网关在架构上做了一些针对性的优化：

元数据读取采用富客户端多级缓存架构，并异步刷新缓存过期数据，该架构能支持千万级 QPS 请求，并能良好的控制机房网络拥塞。

同步调用受限于线程数量，而线程资源宝贵，在 API 网关这类高并发应用场景下，一定比例的 API 超时就会让所有调用的 RT 升高，异步化的引入彻底的隔离 API 之间的影响。网关在 Servlet 线程在进行完 API 调用前置校验后，使用 HSF 或 HTTP NIO client 发起远程服务调用，并结束和回收到该线程。待 HSF 或者 HTTP 请求得到响应后，以事件驱动的方式将远程调用响应结果和 API 请求上下文信息，提交到 TOP 工作线程池，由 TOP 工作线程完成后续的数据处理。最后使用 Jetty Continuation 特性唤起请求将响应结果输出给 ISV，实现请求的全异步化处理。线程模型如图 1 所示。

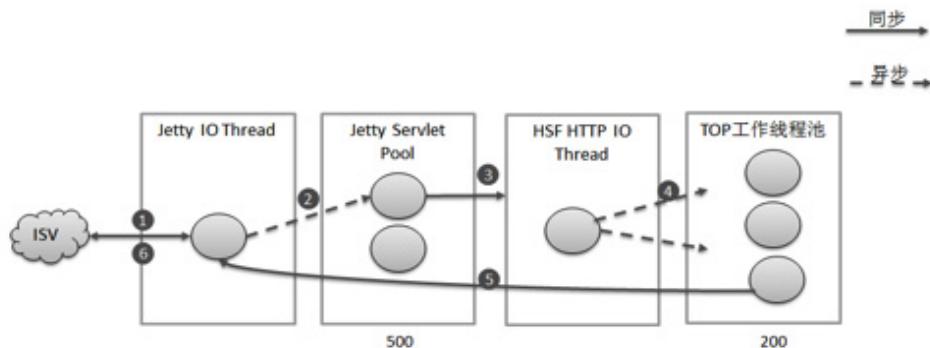


图1 API网关全异步化调用模型

1.2 多级缓存富客户端

在 API 调用链路中会依赖对元数据的获取，比如需要获取 API 的流控信息、字段等级、类目信息、APP 的密钥、IP 白名单、权限包信息，用户授权信息等等。在双 11 场景下，元数据获取 QPS 高达上千万，如何优化

元数据获取的性能是 API 网关的关键点。

千万级 QPS 全部打到 DB 是不可取的，尽管 DB 有做分库分表处理，所以我们在 DB 前面加了一层分布式缓存；然而千万级 QPS 需要近百台缓存服务器，为了节约缓存服务器开销以及减少过多的网络请求，我们在分布式缓存前面加了一层 LRU 规则的本地缓存；为了防止缓存被击穿，我们在本地缓存前面加了一层 BloomFilter。一套基于漏斗模型的元数据读取架构产生（如图 2 所示）。缓存控制中心可以动态推送缓存规则，如数据是否进行缓存、缓存时长、本地缓存大小。为了解决缓存数据过期时在极端情况下可能出现的并发请求问题，网关会容忍拿到过期的元数据（多数情况对数据时效性要求不高），并提交异步任务更新数据信息。

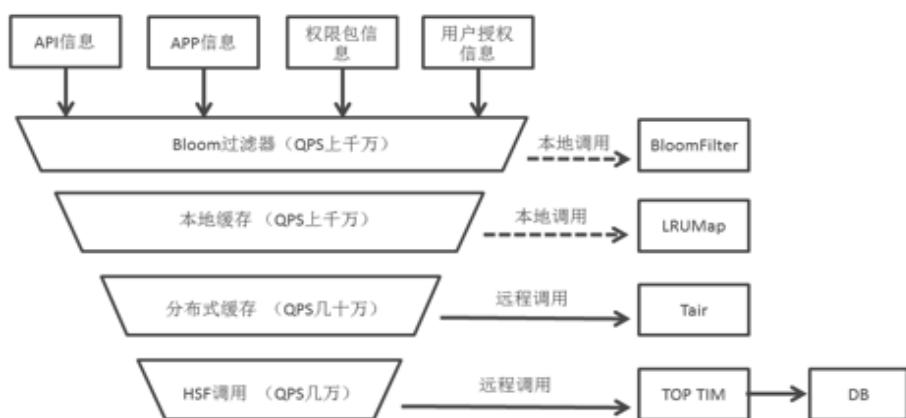


图2 基于漏斗模型的元数据读取

1.3 高性能批量API调用

在双 11 高并发的场景下，对商家和 ISV 的系统同样是一个考验，如何提高 ISV 请求 API 的性能，降低请求 RT 和网络消耗同样是一个重要的事情。在 ISV 开发的系统中通常存在这样的逻辑单元，需要调用多个 API 才能完成某项业务（如图 3），在这种串行调用模式下 RT 较长同时多次调用发送较多重复的报文导致网络消耗过多，在弱网环境下表现更加明显。

API 网关提供批量 API 调用模式（如图 4 所示）缓解 ISV 在调用 RT

过高和网络消耗上的痛点。ISV 发起的批量请求会在 TOP SDK 进行合并，并发送到指定的网关；网关接收到请求后在单线程模式下进行公共逻辑计算，计算通过后将调用安装 API 维度拆分，并分别发起异步化远程调用，至此该线程结束并被回收；每个子 API 的远程请求结果返回时会拿到一个线程进行私有逻辑处理，处理结束时会将处理结果缓存并将完成计数器加一；最后完成处理的线程，会将结果进行排序合并和输出。

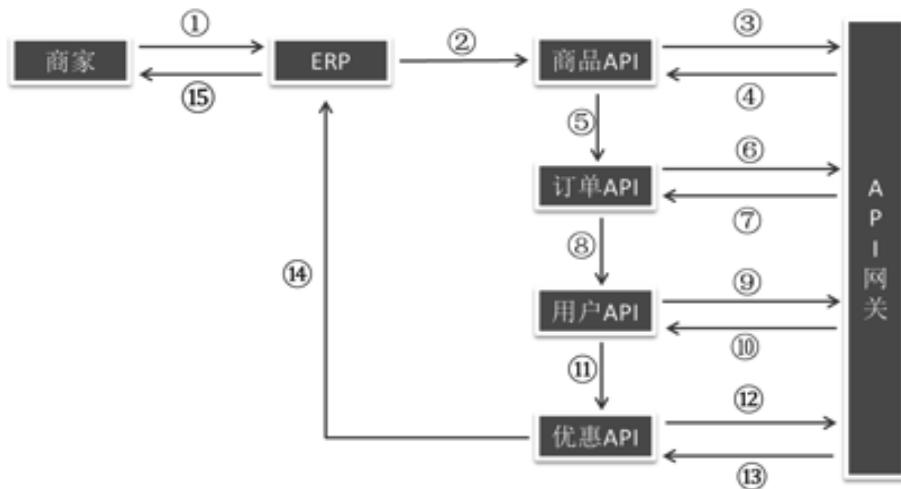


图3 串行API调用处理流程



图4 批量API调用处理流程

1.4 多维度流量控制

TOP API 网关暴露在互联网环境，日调用量达几百亿。特别是在双 11 场景中，API 调用基数大、调用者众多以及各个 API 的服务能力不一致，为了保证各个 API 能够稳定提供服务，不会被暴涨的请求流量击垮，那么

多维度流量控制是 API 网关的一个重要环节。API 网关提供一系列通用的流量控制规则，如 API 每秒流控、API 单日调用量控制、APPKEY 单日调用量控制等。

在双 11 场景中，也会有一些特殊的流量控制场景，比如单个 API 提供的能力有限，例如只能提供 20 万 QPS 的能力而实际的调用需求可能会有 40 万 QPS。在这种场景下怎么去做好流量分配，保证核心业务调用不被限流。TOP API 网关提供了流量分组的策略，比如我们可以把 20 万 QPS 的能力分为 3 个组别，并可以动态去配置和调整每个组别的比例，如：分组 1 占比 50%、如分组 2 占比 40%、分组 3 占比 10%。我们将核心重要的调用放到分组 1，将实时性要求高的调用放到分组 2，将一些实时性要求不高的调用放到分组 3。通过该模式我们能够让一些核心或者实时性要求高的调用能够较高概率通过流量限制获取到相应的数据。同时 TOP API 网关是一个插件化的网关，我们可以编写流控插件并动态部署到网关，在流控插件中我们可以获取到调用上下文信息，通过 Groovy 脚本或简单表达式编写自定义流控规则，以满足双 11 场景中丰富的流控场景。

使用集群流控还是单机流控？单机流控的优势是系统开销较小，但是存在如下短板：

1. 集群单机流量分配不均。
2. 单日流控计数器在某台服务器挂掉或者重启时比较难处理。
3. API QPS 限制小于网关集群机器数量时，单机流控无法配置。基于这些问题，API 网关最开始统一使用集群流控方案，但在双 11 前压测中发现如下一些问题：
4. 单 KEY 热点问题，当单 KEY QPS 超过几十万时，单台缓存服务器 RT 明显增加。
5. 缓存集群 QPS 达到数百万时，服务器投入较高。

针对第一个问题的解法是，将缓存 KEY 进行分片可将请求离散多台缓存服务器。针对第二个问题，API 网关采取了单机 + 集群流控相结合的解决方案，对于高 QPS API 流控采取单机流控方案，服务端使用 Google

ConcurrentLinkedHashMap 缓存计数器，在并发安全的前提下保持了较高的性能，同时能做到 LRU 策略淘汰过期数据。

2 高可靠消息服务

有了 API 网关，服务商可以很方便获取淘系数据，但是如何实时获取数据呢？轮询！数据的实时性依赖于应用轮询间隔时间，这种模式，API 调用效率低且浪费机器资源。基于这样的场景，开放平台推出了消息服务技术，提供一个实时的、可靠的、异步双向数据交换通道，大大提高 API 调用效率。目前，整个系统日均处理百亿级消息，可支撑百万级瞬时流量，如丝般顺滑。

2.1 总体架构

消息系统从部署上分为三个子系统，路由系统、存储系统以及推送系统。消息数据先存储再推送，保证每条消息至少推送一次。写入与推送分离，发送方不同步等待接收方应答，客户端的任何异常不会影响发送方系统的稳定性。系统模块交互如图 5 所示。

路由系统，各个处理模块管道化，扩展性强。系统监听主站的交易、商品、物流等变更事件，针对不同业务进行消息过滤、鉴权、转换、存储、日志打点等。系统运行过程记录各个消息的处理状况，通过日志采集器输出给 JStorm 分析集群处理并记录消息轨迹，做到每条消息有迹可循。

存储系统，主要用于削峰填谷，基于 BitCask 存储结构和内存映射文件，磁盘完全顺序写入，速度极佳。数据读取基于 FileRegion 零拷贝技术，减少内存拷贝消耗，数据读取速度极快。存储系统部署在多个机房，有一定容灾能力。

推送系统，基于 Disputor 构建事件驱动模型，使用 Netty 作为网络层框架，构建海量连接模型，根据连接吞吐量智能控制流量，降低慢连接对系统的压力；使用 WebSocket 构建长连接通道，延时更低；使用对象池技术，有效降低系统 GC 频率；从消息的触发，到拉取，到发送，到确认，

整个过程完全异步，性能极佳。

2.2 选择推送还是拉取

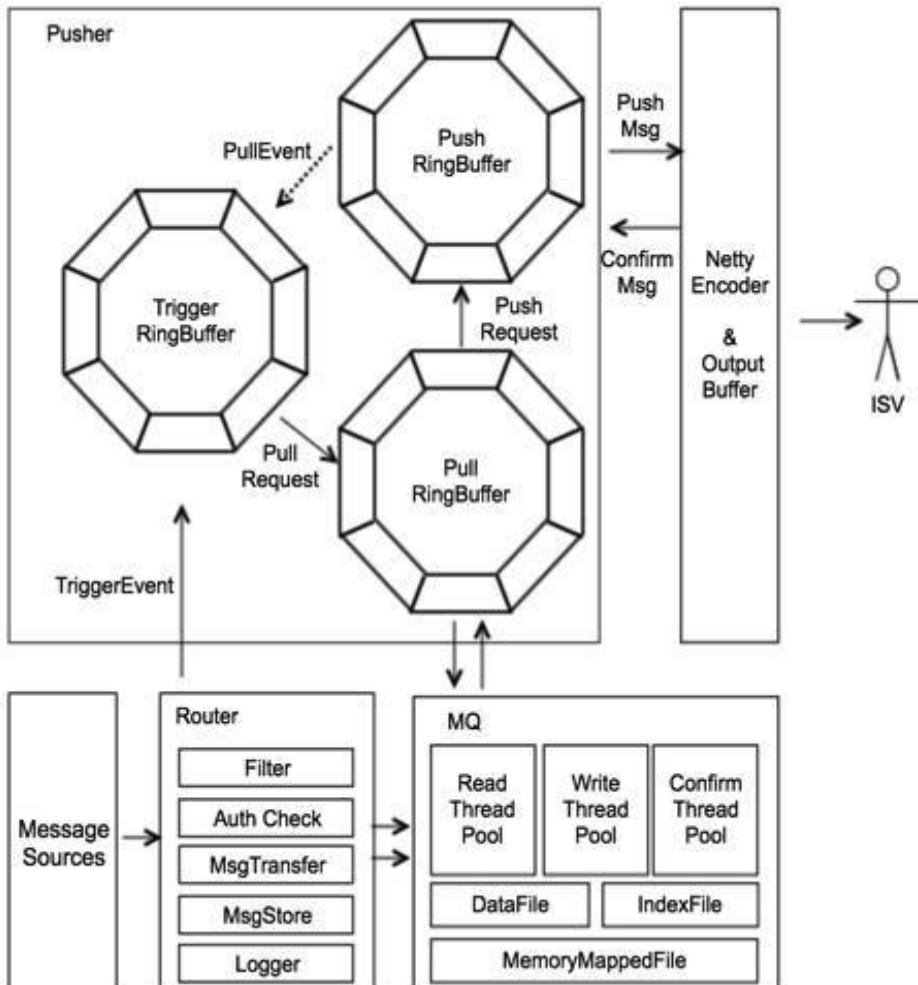


图5 消息服务总体架构

在消息系统中，一般有两种消费模式：服务端推送和客户端拉取。本系统主要面向公网的服务器，采用推送模式，有如下优点：

- 实时性高。从消息的产生到推送，总体平均延时100毫秒，最大不超过200毫秒。
- 服务器压力小。相比于拉取模式，每次推送都有数据，避免空轮询消耗资源。
- 使用简便。使用拉取模式，客户端需要维护消费队列的位置，以

及处理多客户端同时消费的并发问题。而在推送模式中，这些事情全部由服务器完成，客户端仅需要启动SDK监听消息即可，几乎没有使用门槛。

当然，系统也支持客户端拉取，推送系统会将客户端的拉取请求转换为推送请求，直接返回。推送服务器会据此请求推送相应数据到客户端。即拉取异步化，如果客户端没有新产生的数据，不会返回任何数据，减少客户端的网络消耗。

2.3 如何保证低延时推送

在采用推送模式的分布式消息系统中，最核心的指标之一就是推送延时。各个长连接位于不同的推送机器上，那么当消息产生时，该连接所在的机器如何快速感知这个事件？

在本系统中，所有推送机器彼此连接（如图6所示），构成一个通知网，其中任意一台机器感知到消息产生事件后，会迅速通知此消息归属的长连接的推送机器，进而将数据快速推送给客户端。而路由系统每收到一条消息，都会通知下游推送系统。上下游系统协调一致，确保消息一触即达。

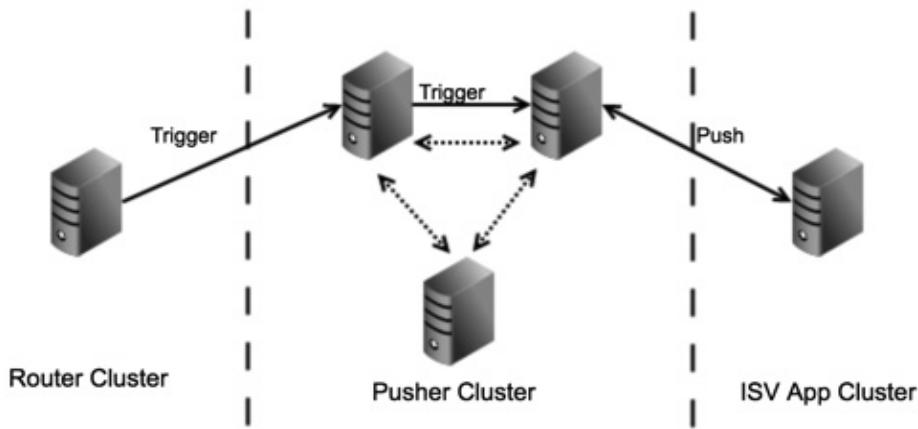


图6 消息事件触发流程

2.4 如何快速确认消息

评估消息系统另外一个核心指标是消息丢失问题。由于面向广大开发

者，因此系统必须兼顾各种各样的网络环境问题，开发者能力问题等。为了保证不丢任何一条消息，针对每条推送的消息，都会开启一个事务，从推送开始，到确认结束，如果超时未确认就会重发这条消息，这就是消息确认。

由于公网环境复杂，消息超时时间注定不能太短，如果是内网环境，5秒足矣，消息事务在内存就能完成。然后在公网环境中，5秒远远不够，因此需要持久化消息事务。在推送量不大的时候，可以使用数据库记录每条消息的发送记录，使用起来也简单方便。但是当每秒推送量在百万级的时候，使用数据库记录的方式就显得捉襟见肘，即便是分库分表也难以承受如此大的流量。

对于消息推送事务数据，有一个明显特征，99%的数据会在几秒内读写各一次，两次操作完成这条数据就失去了意义。在这种场景，使用数据库本身就不合理，就像是在数据库中插入一条几乎不会去读的数据。这样没意义的数据放在数据库中，不仅资源浪费，也造成数据库成为系统瓶颈。

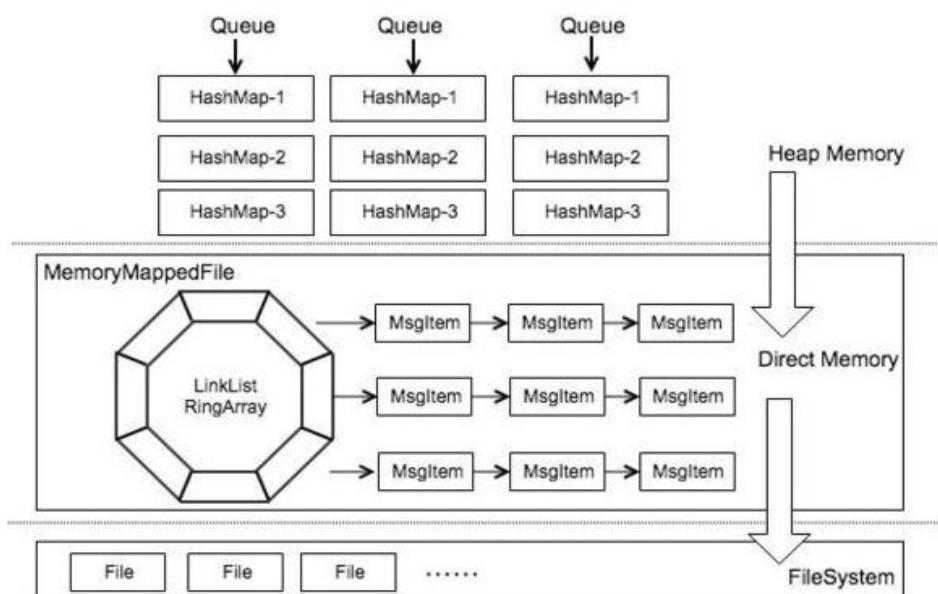


图7 消息确认流程

如图 7 所示，针对这种场景，本系统在存储子系统使用 HeapMemory、

DirectMemory、FileSystem 三级存储结构。为了保护存储系统内存使用情况，HeapMemory 存储最近 10 秒发送记录，其余的数据会异步写入内存映射文件中，并写入磁盘。HeapMemory 基于时间维度划分成三个 HashMap，随着时钟滴答可无锁切换，DirectMemory 基于消息队列和时间维度划分成多个链表，形成链表环，最新数据写入指针头链表，末端指针指向的是已经超时的事务所在链表。这里，基于消息队列维护，可以有效隔离各个队列之间的影响；基于时间分片不仅能控制链表长度，也便于扫描超时的事务。

在这种模式下，95% 的消息事务会在 HeapMemory 内完成，5% 的消息会在 DirectMemory 完成，极少的消息会涉及磁盘读写，绝大部分消息事务均在内存完成，节省大量服务器资源。

3 零漏单数据同步

我们已经有了 API 网关以及可靠的消息服务，但是对外提供服务时，用户在订单数据获取中常常因为经验不足和代码缺陷导致延迟和漏单的现象，于是我们对外提供数据同步的服务。

传统的数据同步技术一般是基于数据库的主备复制完成的。在简单的业务场景下这种方法是可行的，并且已经很多数据库都自带了同步工具。但是在业务复杂度较高或者数据是对外同步的场景下，传统的数据同步工具就很难满足灵活性、安全性的要求了，基于数据的同步技术无法契合复杂的业务场景。

双 11 场景下，数据同步的流量是平常的数十倍，在峰值期间是百倍，而数据同步机器资源不可能逐年成倍增加。保证数据同步写入的平稳的关键在于流量调控及变更合并。

3.1 分布式数据一致性保证

在数据同步服务中，我们使用了消息 + 对账任务双重保障机制，消息保障数据同步的实时性，对账任务保障数据同步一致性。以订单数据同

步为例，订单在创建及变更过程中都会产生该订单的消息，消息中夹带着订单号。接受到该消息后，对短时间内同一订单的消息做合并，数据同步客户端会拿消息中的订单号请求订单详情，然后写入 DB。消息处理过程保证了订单在创建或者发生了任意变更之后都能在极短的延迟下更新到用户的 DB 中。

对账任务调度体系会同步运行。初始化时每个用户都会生成一个或同步任务，每个任务具有自己的唯一 ID。数据同步客户端存活时每 30 秒发出一次心跳数据，针对同一分组任务的机器的心跳信息将会进行汇总排序，排序结果一般使用 IP 顺序。每台客户端在获取需执行的同步任务列表时，将会根据自身机器在存活机器总和 x 中的顺序 y，取得任务 ID % x = y - 1 的任务列表作为当前客户端的执行任务。执行同步任务时，会从订单中心取出在过去一段时间内发生过变更的订单列表及变更时间，并与用户 DB 中的订单进行一一对比，如果发现订单不存在或者与存储的订单变更时间不一致，则对 DB 中的数据进行更新。

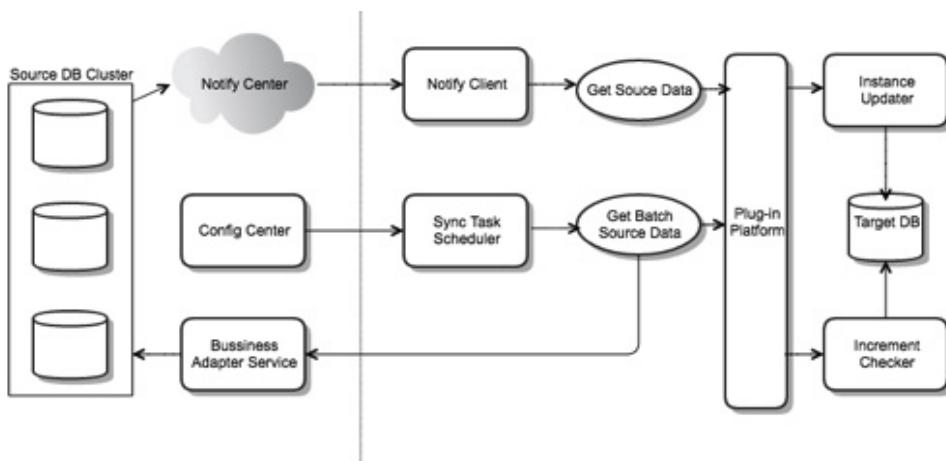


图8 数据同步服务架构

3.2 资源动态调配与隔离

在双 11 场景下如何保证数据同步的高可用，资源调配是重点。最先面临的问题是，如果每台机器都是幂等的对应全体用户，那么光是这些用户身后的 DB 连接数消耗就是很大问题；其次，在淘宝的生态下，卖家用

户存在热点，一个热点卖家的订单量可能会是一个普通卖家的数万倍，如果用户之间直接共享机器资源，那么大流量用户将会占用几乎全部的机器资源，小流量用户的数据同步实效会受到很大的影响。

为了解决以上问题，我们引入了分组隔离。数据同步机器自身是一个超大集群，在此之上，我们将机器和用户进行了逻辑集群的划分，同一逻辑集群的机器只服务同一个逻辑集群的用户。在划分逻辑集群时，我们将热点用户从用户池中取出，划分到一批热点用户专属集群中。分组隔离解决了 DB 连接数的问题，在此场景下固定的用户只会有固定的一批机器为他服务，只需要对这批机器分配连接数即可，而另一个好处是，我们可以进行指定逻辑集群的资源倾斜保障大促场景下重点用户的数据同步体验。

数据同步服务大集群的机器来源于三个机房，在划分逻辑集群时，每个逻辑分组集群都是至少由两个以上机房的机器组成，在单个机房宕机的场景下，逻辑集群还会有存活机器，此时消息和任务都会向存活的机器列表进行重新分配，保证该逻辑集群所服务的用户不受影响。在机器发生宕机或者单个逻辑集群的压力增大时，调度程序将会检测到这一情况并且对冗余及空闲机器再次进行逻辑集群划分，以保证数据同步的正常运行。在集群压力降低或宕机机器恢复一段时间后，调度程序会自动将二次划分的机器回收，或用于其他压力较大的集群。

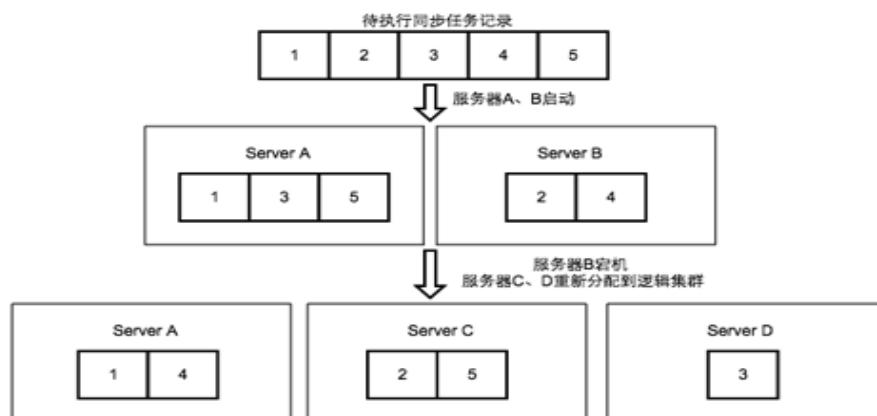


图 9 机器宕机与重分配

3.3 通用数据存储模型

订单上存储的数据结构随着业务的发展也在频繁的发生了变化，进行订单数据的同步，需要在上游结构发生变化时，避免对数据同步服务产生影响，同时兼顾用户的读取需求。对此我们设计了应对结构易变数据的大字段存储模型。在订单数据的存储模型中，我们将订单号、卖家昵称、更新时间等需要被当做查询 / 索引条件的字段抽出独立字段存储，将整个的订单数据结构当成 json 串存入一个大字段中。

| 名称 | 类型 | 是否索引 | 说明 |
|---------------------------|------------|------|-----------|
| <code>tid</code> | NUMBER | Y | 交易ID |
| <code>status</code> | VARCHAR | Y | 交易状态 |
| <code>type</code> | VARCHAR | Y | 交易类型 |
| <code>seller_nick</code> | VARCHAR | Y | 卖家昵称 |
| <code>buyer_nick</code> | VARCHAR | | 买家昵称 |
| <code>created</code> | DATETIME | Y | 交易创建时间 |
| <code>modified</code> | DATETIME | Y | 交易修改时间 |
| <code>jdp_created</code> | DATETIME | Y | 数据同步的创建时间 |
| <code>jdp_modified</code> | DATETIME | Y | 数据同步的修改时间 |
| <code>jdp_hashcode</code> | VARCHAR | | 用作数据校验的字段 |
| <code>jdp_response</code> | MEDIUMTEXT | | 复杂的交易结构 |

图10 订单同步数据存储结构

这样的好处是通过大字段存储做到对上游业务的变化无感知，同时，为了在进行增量数据同步时避免对大字段中的订单详情进行对比，在进行数据同步写入的同时将当前数据的 hashcode 记录存储，这样就将订单数据对比转换成了 hashcode 与 modified 时间对比，提高了更新效率。

3.4 如何降低数据写入开销

在双 11 场景下，数据同步的瓶颈一般不在淘宝内部服务，而在外部用户的 DB 性能上。数据同步是以消息的方式保证实时性。在处理非创建消息的时候，我们会使用直接 update + modified 时间判断的更新方式，替换传统的先 select 进行判断之后再进行 update 的做法。这一优化降低了 90% 的 DB 访问量。

传统写法:

```
SELECT * FROM jdp_tb_trade WHERE tid = #tid#;  
UPDATE jdp_tb_trade SET jdp_response = #jdpResponse#, jdp_modified = now() WHERE tid = #tid#
```

优化写法:

```
UPDATE jdp_tb_trade SET jdp_response = #jdpResponse#, jdp_modified = now() WHERE tid = #tid# AND modified < #modified#
```

订单数据存在明显的时间段分布不均的现象，在白天订单成交量较高，对 DB 的访问量增大，此时不适合做频繁的删除。采用逻辑删除的方式批量更新失效数据，在晚上零点后交易低峰的时候再批量对数据错峰删除，可以有效提升数据同步体验。



全球移动技术大会2017

首届以大前端为主题的技术大会

2017.6.09-10 北京·国际会议中心

重量级嘉宾齐助阵

| | | |
|---|--|--|
| 《移动项目快速持续交付的工程化实践》 林永坚 REA Group Mobile Developer | 《Instagram Direct: 高效可靠的数据端到端传输》 李晨 Instagram iOS 高级工程师 | 《利用CNN实现无需联网的智能图像处理》 李永会 百度 图像搜索客户端工程师 |
| 《QQ移动页面框架优化实践》 陈志兴 腾讯 高级工程师 | 《微信SQLite数据库损坏恢复实践》 何俊伟 微信 Android高级工程师 | 《手机天猫面向业务的界面解决方案-Tangram》 高嘉峻（伯灵） 天猫 无线技术专家 |
| 《豌豆荚的反作弊技术架构与设计》 胡强 阿里 应用分发平台 Android端负责人 | 《H5互动的正确打开方式》 金擎（渚薰） 阿里巴巴 前端专家 | 《ReactNative框架在京东无线端的实践》 沈晨 京东 专家架构师 |
| 《滴滴出行iOS端瘦身实践》 戴铭 滴滴 出行技术专家 | 《移动虚拟化：360分身大师那些事》 王云鹏 奇虎360 分身大师项目技术负责人 | 《携程无线持续交付平台工程实践》 赵辛贵 携程 高级研发经理 |

.....

5月12日前购票享**2880**元 (团购更优惠)



处理微服务架构的内部架构和外部架构

作者 Asanka Abeysinghe 译者 足下



今天的企业里内容范围非常广，包括服务、传统应用程序和数据等等，这由一系列的消费渠道为首，包括桌面、网站和移动应用等。但是，通常情况下，由于没有一个被以适当的方式创建和系统化地管理的集成层，会出现断层，这些消费渠道需要有这样的集成层来使能业务功能。大多数企业正在通过面向服务的架构（Service-Oriented Architecture, SOA）来应对这一挑战。在面向服务的架构中，应用程序组件通过网络上的通信协议向其他组件提供松散耦合的服务。最终，它的意图是让微服务架构可以更灵活和更容易地扩展。虽然没有充分地准备好采用微服务架构，但是这些组织正在规划架构并实现企业应用和服务平台，使他们可以逐步走向微服务架构。

事实上，Gartner 预测到 2017 年为止，超过 20% 的大型机构将会部署完整的微服务，以提高灵活性和可扩展性，事实上她们已经这样做了。微服务架构正日益成为有效地提交新功能的重要途径。它可以解决伴随着创建新服务的同时带来的复杂难题，结合传统应用程序和数据库，并开发 Web 应用程序、移动应用程序或任何基于消费者的应用。

现在，企业们正在朝着 SOA 发展，并且在 SOA 的范围里张开臂膀迎接微服务架构的概念。很有可能，最大的吸引力在于由这些微服务提供的组件化和单一功能，使得迅速部署组件以及按需扩展变得可能。它已经不再只是一个新的概念。

比如在 2011 年，有一个医疗服务平台启动了一个新的策略，就是每当它写了一个新的服务时，它就会相应地为它配备一台新的应用服务器，以支持服务部署。因此，这是一个来自于 DevOps 的实践，它为服务之间创造了一种依赖较少的环境，并确保在进行某些维护操作的时候对其他系统影响可以降到最小。其结果是，这些服务运行在超过了 80 台服务器之上。事实上这是非常基本的，因为那时不像现在这样有适当的 DevOps 工具，相反，他们使用 Shell 脚本和 Maven 类型的工具来构建服务器。

然而微服务是非常重要的，它只是一个大格局的一个小方面。很明显，一个组织不可能仅仅因为使用了微服务就能得到微服务的全部好处。在设计微服务的时候，采用微服务架构以及最佳实践的做法是营造一个鼓励创新的环境，并实现业务能力的快速提交的关键所在。这就是真正的附加价值。

解决实现的挑战

在构建你自己的微服务架构时，大家普遍接受的实践是专注于你如何打磨出单一功能的服务，而不是服务的规模。内部架构通常只能解决微服务自己的实现。外部架构包括必需的平台能力，在开发和部署你的微服务时，会需要这些平台能力来确保可连接性、灵活性和可扩展性。为此，在制作你内部和外部的微服务架构的时候，企业中间件起着关键的作用。

首先，中间件技术应该兼容是对 DevOps 友好的，它包含高性能的功能，并且支持关键服务标准。此外，它必须支持一些设计原则，比如迭代架构并且易于插拔，这反过来将提供快速的应用程序开发与持续发布。最重要的是，一个全面的数据分析层对于设计失败的支持是至关重要的。

在实施微服务架构时，企业常犯的最大错误往往是完全抛弃了已经确立的 SOA 方法，并且用微服务背后的理论来替换它们。这样就会导致架构不完整，并且引入了冗余。聪明一些的做法是把一个微服务架构当成一个分层的系统，它包括类似企业服务总线（Enterprise Service Bus, ESB）的功能来处理所有的与整合相关的功能。这也将作为一个中间层，使变化可以发生在这个层面上，这样它可以应用到所有相关的微服务。换句话说，ESB 或类似的中介引擎通过提供所需的连接性去将历史数据和服务整合到微服务，实现逐步向微服务架构推进。先发布微服务，然后再通过 API 把它提供出去，这种方法对于整合一些基本规则来说也很重要。

内部架构的范围确定和设计

值得注意的是，内部架构需要设计得比较简单，因此它才可以很容易地独立部署，或者单独废弃。一旦出现微服务失败或有更好的服务出现，可废弃性是必需的。无论是这两种情况下的哪一种，都需要可以很容易地废弃相应的微服务。微服务也需要获得部署架构和运行环境的强有力支持，微服务要在这样的运行环境中被构建、部署和执行。因此，它需要足够简单，这样才能独立部署。理想的做法是发布一个相同服务的新版本，加入对缺陷的修复，包括新的功能或者对现有功能的改进，并去除掉废弃的功能。

微服务架构建立的基础框架决定着对一个微服务架构的内部架构的关键需求。吞吐量、延迟和低资源使用率（内存和 CPU）等都是需要考虑的关键需求。一个好的微服务框架通常会建立在轻量级、运行速度快和现代编程模型的基础之上，比如注释元配置，它与核心业务逻辑相独立。此外，它应该有能力来保障微服务的安全，满足必要的行业领先的安全标准，也应同时提供一些指标来监测微服务的行为。

与外部架构相比，内部架构中每一个微服务的实现都比较简单。在搜寻和设计内部架构时，一个好的服务设计要考虑到以下六个因素：

首先，微服务应该有单一的目的和单一的责任，并且服务本身应该作为一个独立的部署单元，可以在生产部署时创建多个实例。

其次，微服务应该有这样的能力，那就是可以采用一个最适合它要发布的功能的架构，并且此架构能够使用适当的技术。

第三，一旦单体服务被细分为微服务，每一项微服务或每套微服务就应该有可以通过 API 提供服务的能力。然而，在内部实现中，该服务可以采用任何合适的技术，实现业务需求，实现各自的业务能力。为此，企业可能要考虑像 Swagger 那样的东西来定义 API 规范或特定微服务的 API 定义，并且微服务能利用这个作为互动的点。这在微服务开发中被称为以 API 为先的方法。

第四，要部署的内容也可能有不同，比如捆绑在基于 Hypervisor 的镜像的独立的可部署单元，或者是容器镜像，这通常是更受欢迎的选择。

第五，企业需要利用分析来细化微服务，同时一旦服务失败要提供恢复手段。为此，企业要整合使用度量指标和监控来支持微服务的演进方法。

第六，即使微服务范式本身可以让企业的微服务有多种实现或多语言实现，对最佳实践和标准的使用对于保持一致性是非常必要的，同时要确保解决方案遵循一般的企业架构原则。这并不是说，多语言的机会就被完全否决了，而是说它们在使用时需要被监管。

用“外部架构”来解决平台能力

一旦内部架构已经建立，架构师就需要关注组成他们微服务架构的外部架构的功能。外部架构的一个关键组成部分是引入企业服务总线（ESB）或类似的中介引擎，它们将会帮忙把历史数据和服务与微服务架构连接起来。中介层也将使企业可以维护自己的标准，同时让别人可以在相应的生态系统里管理他们自己的。

使用服务注册中心可以支持依赖管理、影响分析、微服务和 API 的发

现功能。这也完全可以把服务和 API 的组合流水线化，并把微服务连线到服务经纪人或枢纽。任何微服务架构都应该支持创建 RESTful API，这将有助于在企业开发应用软件时定制资源模型和应用逻辑。

先设计 API，再实现微服务，然后通过 API 将它提供出去，要注意提供出去的是 API 而不是微服务。要坚持这样的原则。各家企業都想要解决的另一个共同需求是微服务的安全问题。在一个典型的单体应用程序中，企业将使用底层存储库或用户存储区来生成来自旧架构的安全层所需要的信息。在微服务架构中，企业可以利用在业界广泛采用的 API 安全标准，比如 OAuth2 和 OpenID Connect 等，去实现对边缘模块的安全层，包括微服务架构中的 API。

在所有这些能力中，最重要的也是真正有助于解决微服务架构复杂性的是使用企业级的底层平台，它可以提供丰富的功能来管理可扩展性、可用性和性能。这是因为把一个单体应用分解成微服务并不意味着一个简化了的环境或服务。可以肯定的是在应用层面，企业本质上是在处理几个微服务，这远比一个单一的复杂的应用更简单。然而，作为一个整体的构建却是相当地艰巨。

事实上，微服务架构的复杂度可能更大，因为我们当需要考虑其他方面时，比如向一个进程发起一次单向调用这不算复杂，但微服务之间是需要相互调用的。这在本质上意味着，系统的复杂性已经变成了所谓的“外部架构”，这通常由 API 网关、服务路由、发现、消息通道和依赖管理等组成。

因为内部架构现在已经极其简单，它只包含用来构建一个微服务架构的基础和运行时，架构师将会发现在微服务架构已经有了一个干净的服务层。我们需要更多地关注外部架构，以解决大家所共同面临的复杂性。如图 1 所示，有一些常见的切实的场景需要解决。

外部架构将需要一个 API 网关，以帮助它对内对外提供业务 API 的能力。通常，大家会使用 API 管理平台来管理这方面的外部架构。这对于那些正在构建 Web 应用程序、移动应用程序和物联网解决方案等的用户来说，

把这些基于微服务架构的服务能力提供给他们是非常必要的。

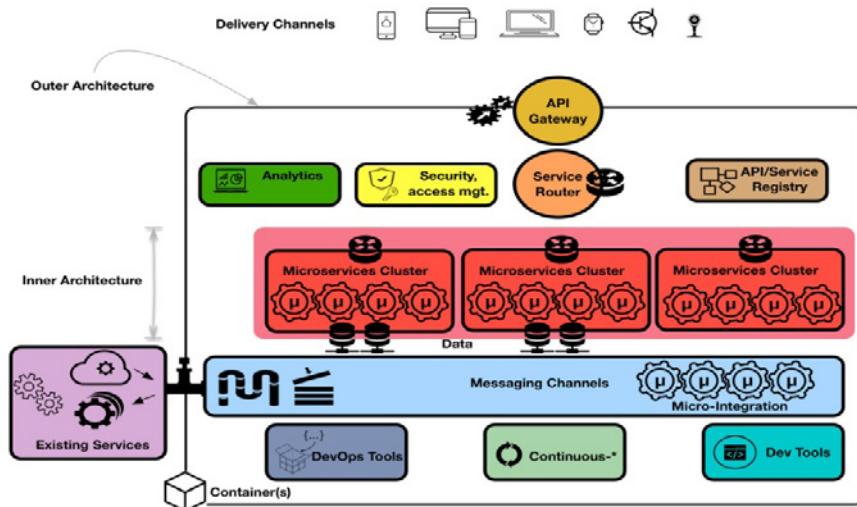


图1

一旦微服务到位，就会出现某种形式的服务路由，通过 API 发过来的业务请求将被路由到相关服务集群或服务。在微服务内部，会有以分担负载为目的的多个实例。因此，有必要进行某种形式的负载均衡。

此外，微服务之间也有相关性。例如，如果微服务 A 对微服务 B 有依赖，它将在运行时调用微服务 B。服务注册中心可以让微服务发现后台服务节点，所以它可以解决这个需求。服务注册中心还将管理 API 和服务依赖关系，以及其他资产，包括策略。

接下来，微服务架构外部架构需要一定的信息传递渠道，这基本上形成了一层，允许微服务内部的交互，并且该层还把微服务架构连接到旧的系统。此外，这一层也有助于建立微服务之间通信（微整合）通道，并且这样的通道应该是轻量级的协议，比如 HTTP、MQTT 等等。

当微服务之间相互通信时，需要有某种形式的身份验证和授权。使用单体的应用程序时这是不必要的，因为有一个直接的过程调用。相比之下使用微服务时，这些就转化成了网络调用。最后，诊断和监控都是需要考虑的关键方面，以找出由每个微服务处理的请求类型。这将有助于企业扩展单个微服务的规模。

回顾微服务架构场景

为了全面正确地看待事情，我们分析一些实际场景，这些场景展示了微服务架构的内部和外部架构如何一起运作。我们将假设组织已经使用微软 Windows Communication Foundation 或 Java JEE/J2EE 服务框架实现了她的服务，并且开发人员还在写新服务代码，使用的是应用了微服务架构原则的微服务框架。

在这种情况下，提供数据和业务能力的现有服务不能被忽视。因此，新的微服务将需要与现有服务平台之间相互通信。在大多数情况下，这些现有的服务将使用框架所坚持的固有标准。例如，旧的服务可能使用服务绑定，比如 HTTP 上的 SOAP、Java Message Service（JMS）或 IBM MQ 等，并使用 Kerberos 或 WS-Security 进行保护。在这个例子中，消息渠道也将在协议转换、信息转换、从旧架构通向新的微服务架构的安全过渡中起重要的作用。

另一个组织需要考虑的方面是在业务增长方面可能造成的对其可扩展性的影响，由于单体应用在这方面是有很明显的局限性的，而微服务架构是可以横向扩展的。在这些明显的限制中，还有可能的错误，因为在一个单片环境里测试新的功能非常繁琐，会导致延迟实现这些变更，成为满足快速提交需求的障碍。另一个挑战将是在所有者缺失的情况下支撑这个单体的代码库，在微服务的情况下，个人或单个功能是可以自己管理的，这些可以在不影响其他功能的情况下根据需要迅速扩展。

总之，虽然微服务对于组织有显著的好处，以逐步淘汰或迭代的方式向前推进微服务架构以确保平稳过渡，这可能是最好的办法。能使微服务架构成为被优先选择的以服务为导向的方案，其关键是明确所有权，以及它可以将故障隔离，从而使这些所有者可以把他们的领域内的服务实现得更加稳定和高效。

浅析 MySQL JDBC 连接配置上的两个误区

作者 丁雪丰



相信使用 MySQL 的同学都配置过它的 JDBC 驱动，多数人会直接从哪里贴一段 URL 过来，然后稍作修改就上去了，对应的连接池配置也是一样的，很少有人会去细想这每一个参数都是什么含义。今天我们就来聊两个比较常见的配置——是否要开启 autoReconnect 和是否缓存 PreparedStatement。

一、autoReconnect=true 真的好用么？

笔者看到过很多 MySQL 的 URL 里都是这样写的，复制过来改改 IP、端口和库名就能用了：

```
jdbc:mysql://xxx.xxx.xxx.xxx:3306/xxx?autoReconnect=true&...
```

从字面上看挺好的，在连接断开后还会自动重连，加之 MySQL 有 8 小时自动断开连接的特性，在断开后连接会重连，多好的功能呀。但是如果你去阅读一下 MySQL Connect/J 开发手册的相关章节，就会看到官方是这么说明的：

The use of this feature is not recommended, because it has side effects related to session state and data consistency when applications don't handle SQLExceptions properly, and is only designed to be used when you are unable to configure your application to handle SQLExceptions resulting from dead and stale connections properly.

简单来说，不推荐开启这个特性，因为有副作用，在没有正确处理 SQLException 时容易造成会话状态和数据一致性的问题。

一般的应用都会使用数据库连接池，那我们的连接池是否正确地处理了抛出的 SQLException 呢？抱着这个疑问，我们来看看阿里的 Druid 连接池是怎么处理的。

首先，通过设置合理的健康检查及连接存活时间能解决大部分问题；其次，它有针对性特定异常的处理逻辑，在 MySqlExceptionSorter 中会对特定返回码、异常类（比如 com.mysql.jdbc.CommunicationsException 和 com.mysql.jdbc.exceptions.jdbc4.CommunicationsException）以及错误消息进行处理，如果是致命错误就把连接抛弃。也就是说，如果用了 Druid，不管是否设置了 autoReconnect，都能保证后续请求的正确处理。JBoss 的连接池实现也有类似的特性。

二、MySQL 是否真的不用打开 PSCache ?

一般在设置连接池时，都会有类似下面的设置：

```
<property name="poolPreparedStatements" value="true" />
<property name="maxPoolPreparedStatementPerConnectionSize"
value="20" />
```

很多文章上都说 PSCache 对使用游标的数据库有巨大的性能提升，但 MySQL 不建议开启，因为它不支持游标。所以很多人在用 MySQL 时，都会将 poolPreparedStatements 设置为 false，就连 Druid 的文档上也是这么写的。

但事实真的是这样么，MySQL 使用 PSCache 真的对性能没有提升么？

先来看看关于游标的问题，其实大部分文章的表述不太准确，现在的 MySQL 在存储过程里是支持游标的，但其他地方的确不支持，具体详见官方手册（MySQL supports cursors inside stored programs.）。但这并不是我们要讨论的关键。

3.1.0 版本后的 JDBC 驱动里有一个参数是 useServerPrepStmts，如果服务器支持的话，会开启服务端 PreparedStatement，默认是 false。官方手册中有如下说明：

Server-side Prepared Statements – Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

也就是说在 MySQL 4.1.0 版本后，3.1.0 以上的驱动会检测到支持服务端 PreparedStatement，并且启用该特性。根据 MySQLTUTORIAL 上的说明，整个过程分为 PREPARE、EXECUTE 和 DEALLOCATE PREPARE 三步。MySQL JDBC 驱动的 Contributor Jess Balint 在 StackOverflow 上做了一个详细的说明，《High-Performance Java Persistence》的作者也专门撰写文章分析了两者的区别。

```
ps=conn.prepareStatement("select ?")
ps.setInt(1, 42)
ps.executeQuery()
ps.setInt(1, 43)
ps.executeQuery()
```

上述代码在使用客户端 PreparedStatement 时，MySQL 日志里看到的是：

```
255 Query select 42
```

```
255 Query select 43
```

如果用的是服务端 PreparedStatement，看到的则是（实际每次执行只会传占位符的值，语句是不传的）：

```
254 Prepare select ?
254 Execute select 42
254 Execute select 43
```

在整个使用过程中，Prepare 只会做一次，在这时服务端会对语句进行解析，后续收到具体值时会优化执行计划。如果同一条语句每次都新建 PreparedStatement，那么每次都会多一回网络交互和语句解析，这显然是可以优化的。

综上所述，现在在使用 MySQL 时（如果版本比较新的话），出于性能考虑，应该在数据库连接池上开启针对 PreparedStatement 的缓存。如果没有使用连接池，或者所用的连接池不支持 PSCache，也可以在 JDBC 连接上设置 cachePrepStmts=true。

事实上，MySQL 的 JDBC 驱动还有不少针对性能的优化，比如设置 useConfigs=maxPerformance（请酌情使用），相当于同时做了如下设置：

```
cachePrepStmts=true
cacheCallableStmts=true
cacheServerConfiguration=true
useLocalSessionState=true
elideSetAutoCommits=true
alwaysSendSetIsolation=false
enableQueryTimeouts=false
```

各位同学，是时候检视一下自己的系统是如何连接 MySQL 的了，时代在发展，有些以前适用的配置也许就不再合适了。

Zendesk 的 TensorFlow 产品部署经验

作者 Wai Chee Yau 译者 尚剑



我们如何开始使用 TensorFlow

在 Zendesk，我们开发了一系列机器学习产品，比如最新的自动应答（Automatic Answers）。它使用机器学习来解释用户提出的问题，并用相应的知识库文章来回应。当用户有问题、投诉或者查询时，他们可以在线提交请求。收到他们的请求后，Automatic Answers 将分析请求，并且通过邮件建议客户阅读可能最有帮助的相关文章。

Automatic Answers 使用一类目前最先进的机器学习算法来识别相关文章，也就是深度学习。我们使用 Google 的开源深度学习库 TensorFlow 来构建这些模型，利用图形处理单元（GPU）来加速这个过程。

Automatic Answers 是我们在 Zendesk 使用 Tensorflow 完成的第一个数据产品。在我们的数据科学家付出无数汗水和心血之后，我们才有了在 Automatic Answers 上效果非常好的 Tensorflow 模型。

但是构建模型只是问题的一部分，我们的下一个挑战是要找到一种方法，使得模型可以在生产环境下服务。模型服务系统将处理大量的业务，所以需要确保为这些模型提供的软件和硬件基础架构是可扩展的、可靠的和容错的，这对我们来说是非常重要的。接下来介绍一下我们在生产环境中配置 TensorFlow 模型的一些经验。

顺便说一下我们的团队——Zendesk 的机器学习数据团队。我们团队包括一群数据科学家、数据工程师、一位产品经理、UX / 产品设计师以及一名测试工程师。

TensorFlow 模型服务

经过数据科学家和数据工程师之间一系列的讨论，我们明确了一些核心需求：

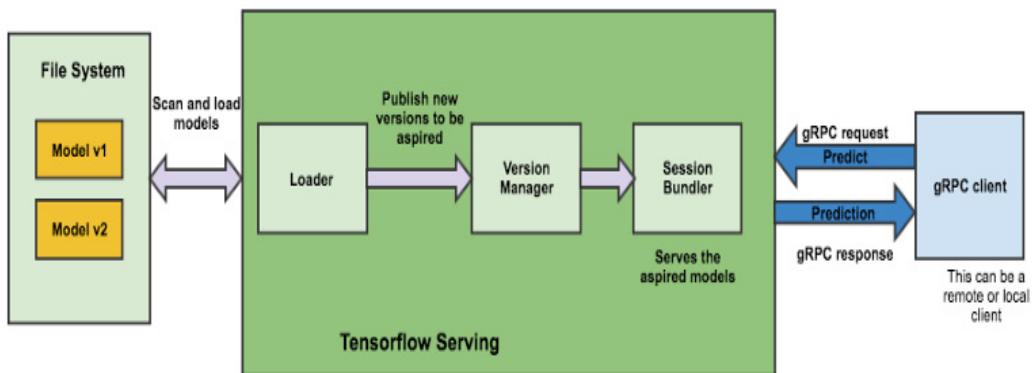
- 预测时的低延迟
- 横向可扩展
- 适合我们的微服务架构
- 可以使用A/B测试不同版本的模型
- 可以与更新版本的TensorFlow兼容
- 支持其他TensorFlow模型，以支持未来的产品

TensorFlow Serving

经过网上的调研之后，Google 的 TensorFlow Serving 成为我们首选的模型服务。TensorFlow Serving 用 C++ 编写，支持机器学习模型服务。开箱即用的 TensorFlow Serving 安装支持：

- TensorFlow模型的服务
- 从本地文件系统扫描和加载TensorFlow模型

TensorFlow Serving 将每个模型视为可服务对象。它定期扫描本地文件系统，根据文件系统的状态和模型版本控制策略来加载和卸载模型。这使得可以在 TensorFlow Serving 继续运行的情况下，通过将导出的模型复制到指定的文件路径，而轻松地热部署经过训练的模型。



根据这篇 Google 博客中报告的基准测试结果，他们每秒记录大约 100000 个查询，其中不包括 TensorFlow 预测处理时间和网络请求时间。

有关 TensorFlow Serving 架构的更多信息，请参阅 TensorFlow Serving 文档。

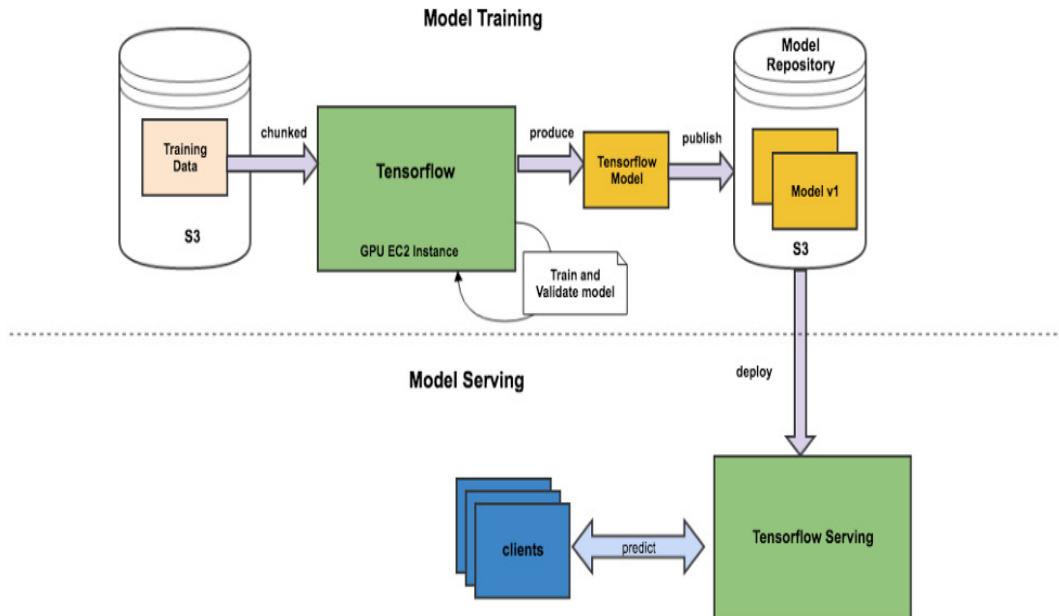
通信协议 (gRPC)

TensorFlow Serving 提供了用于从模型调用预测的 gRPC 接口。gRPC 是一个开源的高性能远程过程调用 (remote procedure call, RPC) 框架，它在 HTTP/2 上运行。与 HTTP/1.1 相比，HTTP/2 包含一些有趣的增强，比如它对请求复用、双向流和通过二进制传输的支持，而不是文本。

默认情况下，gRPC 使用 Protocol Buffers (Protobuf) 作为其信息交换格式。Protocol Buffers 是 Google 的开源项目，用于在高效的二进制格式下序列化结构化数据。它是强类型，这使它不容易出错。数据结构在 .proto 文件中指定，然后可以以各种语言（包括 Python, Java 和 C++）将其编译为 gRPC 请求类。这是我第一次使用 gRPC，我很想知道它与其他 API 架构（如 REST）相比谁性能更好。

模型训练和服务架构

我们决定将深度学习模型的训练和服务分为两个管道。下图是我们的模型训练和服务架构的概述：



模型训练管道

模型训练步骤：

- 我们的训练特征是从Hadoop中提供的数据生成的。
- 生成的训练特征保存在AWS S3中。

然后使用 AWS 中的 GPU 实例和 S3 中的批量训练样本训练 TensorFlow 模型。

一旦模型被构建并验证通过，它将被发布到 S3 中的模型存储库。

模型服务管道

验证的模型在生产中通过将模型从模型库传送到 TensorFlow Serving 实例来提供。

基础结构

我们在 AWS EC2 实例上运行 TensorFlow Serving。Consul 在实例之

前设置，用于服务发现和分发流量。客户端连接从 DNS 查找返回的第一个可用 IP。或者弹性负载平衡可用于更高级的负载平衡。由于 TensorFlow 模型的预测本质上是无状态操作，所以我们可以通过旋转加速更多的 EC2 实例来实现横向可扩展性。

另一个选择是使用 Google Cloud 平台提供的 Cloud ML，它提供 TensorFlow Serving 作为完全托管服务。但是，当我们在大概 2016 年 9 月推出 TensorFlow Serving 时，Cloud ML 服务处于 alpha 阶段，缺少生产使用所需的功能。因此，我们选择在我们自己的 AWS EC2 实例中托管，以实现更精细的粒度控制和可预测的资源容量。

模型服务的实现

下面是我们实现 TensorFlow Serving 部署和运行所采取的步骤：

1. 从源编译TensorFlow Serving

首先，我们需要编译源代码来产生可执行的二进制文件。然后就可以从命令行执行二进制文件来启动服务系统。

假设你已经配置好了 Docker，那么一个好的开端就是使用提供的 Dockerfile 来编译二进制文件。请按照以下步骤：

运行该 gist 中的代码以构建适合编译 TensorFlow Serving 的 docker 容器。

在正在运行的 docker 容器中运行该 gist 中的代码以构建可执行二进制文件。

一旦编译完成，可执行二进制文件将在你的 docker 镜像的以下路径中：/work/serving/bazel-bin/tensorflow_serving/model_servers/tensorflow_model_server。

2. 运行模型服务系统

上一步生成的可执行二进制文件（tensorflow_model_server）可以部署到您的生产实例中。如果您使用 docker 编排框架（如 Kubernetes

或 Elastic Container Service)，您还可以在 docker 容器中运行 TensorFlow Serving。

现在假设 TensorFlow 模型存储在目录 /work/awesome_model_directory 下的生产主机上。你可以在端口 8999 上使用以下命令来运行 TensorFlow Serving 和你的 TensorFlow 模型：

```
<path_to_the_binary>/tensorflow_model_server - port=8999 -  
model_base_path=/work/awesome_model_directory
```

默认情况下，TensorFlow Serving 会每秒扫描模型基本路径，并且可以自定义。此处列出了可作为命令行参数的可选配置。

3. 从服务定义（Service Definitions）生成 Python gRPC 存根

下一步是创建可以在模型服务器上进行预测的 gRPC 客户端。这可以通过编译 .proto 文件中的服务定义，从而生成服务器和客户端存根来实现。.proto 文件在 TensorFlow Serving 源码中的 tensorflow_serving_apis 文件夹中。在 docker 容器中运行以下脚本来编译 .proto 文件。运行提交版本号为 46915c6 的脚本的示例：

```
./compile_ts_serving_proto.sh 46915c6
```

运行该脚本后应该在 tensorflow_serving_apis 目录下生成以下定义文件：

```
model_pb2.py  
predict_pb2.py  
prediction_service_pb2.py
```

你还可以使用 grpc_tools Python 工具包来编译 .proto 文件。

4. 从远程主机调用服务

可以使用编译后的定义来创建一个 python 客户端，用来调用服务器上的 gRPC 调用。比如这个例子用一个同步调用 TensorFlow Serving 的 Python 客户端。

如果您的用例支持异步调用预测，TensorFlow Serving 还支持批处理预测以达到性能优化的目的。要启用此功能，你应该运行 tensorflow_

model_server 同时开启 flag?—enable_batching。这是一个异步客户端的例子。

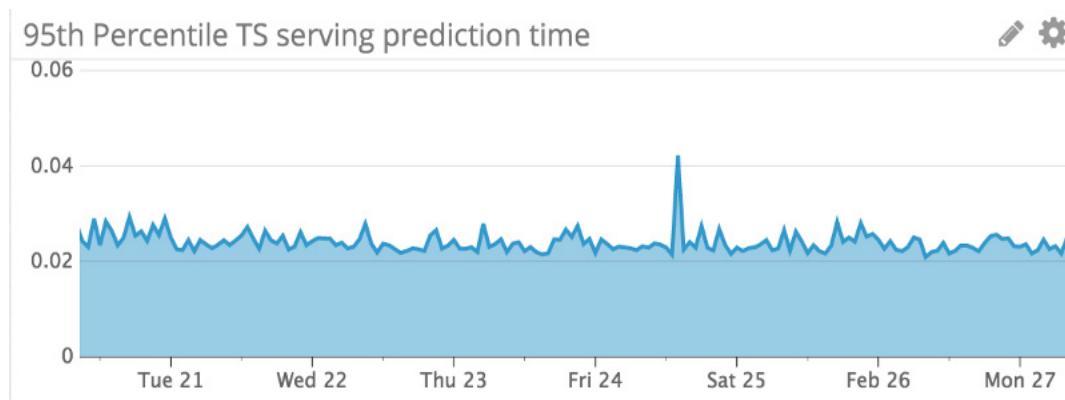
从其他存储加载模型

如果你的模型没有存储在本地系统中应该怎么办？你可能希望 TensorFlow Serving 可以直接从外部存储系统（比如 AWS S3 和 Google Storage）中直接读取。

如果是这种情况，你将需要通过 Custom Source 来拓展 TensorFlow Serving 以使其可以读取这些源。TensorFlow Serving 仅支持从文件系统加载模型。

一些经验

我们在产品中已经使用 TensorFlow Serving 大概半年的时间，我们的使用体验是相当平稳。它具有良好的预测时间延迟。以下是我们的 TensorFlow Serving 实例一周内的预测时间（以秒为单位）的第 95 百分位数的图（约 20 毫秒）。



然而，在生产中使用 TensorFlow Serving 的过程中，我们也有一些经验教训可以跟大家分享。

1. 模型版本化

到目前为止，我们已经在产品中使用了几个不同版本的 TensorFlow

模型，每一个版本都有不同的特性，比如网络结构、训练数据等。正确处理模型的不同版本已经是一个重要的任务。这是因为传递到 TensorFlow Serving 的输入请求通常涉及到多个预处理步骤。这些预处理步骤在不同 TensorFlow 模型版本下是不同的。预处理步骤和模型版本的不匹配可能导致错误的预测。

a. 明确说明你想要的版本

我们发现了一个简单但有用的方法，也就是使用在 model.proto 定义中指定的版本属性，它是可选的（可以编译为 model_pb2.py）。这样可以始终保证你的请求有效负载与预期的版本号匹配。

当你请求某个版本（比如从客户端请求版本 5），如果 TensorFlow Serving 服务器不支持该特定版本，它将返回一个错误消息，提示找不到模型。

b. 服务多个模型版本

TensorFlow Serving 默认的是加载和提供模型的最新版本。

当我们在 2016 年 9 月首次应用 TensorFlow Serving 时，它不支持同时提供多个模型。这意味着在指定时间内它只有一个版本的模型。这对于我们的用例是不够的，因为我们希望服务多个版本的模型以支持不同神经网络架构的 A / B 测试。其中一个选择是在不同的主机或端口上运行多个 TensorFlow Serving 进程，以使每个进程提供不同的模型版本。这样的话就需要：

- 用户应用程序（gRPC 客户端）包含切换逻辑，并且需要知道对于给定的版本需要调用哪个 TensorFlow Serving 实例。这增加了客户端的复杂度，所以不是首选。
- 一个可以将版本号映射到 TensorFlow Serving 不同实例的注册表。

更理想的解决方案是 TensorFlow Serving 可以支持多个版本的模型。

所以我决定使用一个“lab day”的时间来扩展 TensorFlow Serving，使其可以服务多个版本的时间。在 Zendesk，“lab day”就是我们可以每两周有一天的时间来研究我们感兴趣的东西，让它成为能

够提高我们日常生产力的工具，或者一种我们希望学习的新技术。我已经有 8 年多没有使用 C++ 代码了。但是，我对 TensorFlow Serving 代码库的可读性和整洁性印象深刻，这使其易于扩展。支持多个版本的增强功能已经提交，并且已经合并到主代码库中。TensorFlow Serving 维护人员对补丁和功能增强的反馈非常迅速。从最新的主分支，你可以启动 TensorFlow Serving，用 model_version_policy 中附加的 flag 来服务多个模型版本：

```
/work/serving/bazel-bin/tensorflow_serving/model_servers/
tensorflow_model_server
  - port=8999 - model_base_path=/work/awesome_model_directory -
  model_version_policy=ALL VERSIONS
```

值得注意的要点是，服务多个模型版本，需要权衡的是更高的内存占用。所以上述的 flag 运行时，记住删除模型基本路径中的过时模型版本。

2. 活用压缩

当你部署一个新的模型版本的时候，建议在复制到 model_base_path 之前，首先将导出的 TensorFlow 模型文件压缩成单个的压缩文件。Tensorflow Serving 教程中包含了导出训练好的 Tensorflow 模型的步骤。导出的检查点 TensorFlow 模型目录通常具有以下文件夹结构：

包含版本号（比如 0000001）和以下文件的父目录：

- saved_model.pb：序列化模型，包括模型的图形定义，以及模型的元数据（比如签名）。
- variables：保存图形的序列化变量的文件。

压缩导出的模型：

```
tar -cvzf modelv1.tar.gz 0000001
```

为什么需要压缩？

- 压缩后转移和复制更快。
- 如果你将导出的模型文件夹直接复制到 model_base_path 中，复制过程可能需要一段时间，这可能导致导出的模型文件已复制，但相应的元文件尚未复制。如果TensorFlow Serving 开始加载你的

模型，并且无法检测到源文件，那么服务器将无法加载模型，并且会停止尝试再次加载该特定版本。

3. 模型大小很重要

我们使用的 TensorFlow 模型相当大，在 300Mb 到 1.2Gb 之间。我们注意到，在模型大小超过 64Mb 时，尝试提供模型时将出现错误。这是由于 protobuf 消息大小的硬编码 64Mb 限制，如这个 TensorFlow Serving 在 Github 上的问题所述。

最后，我们采用 Github 问题中描述的补丁来更改硬编码的常量值。(这对我们来说还是一个问题。如果你可以找到在不改变硬编码的情况下，允许服务大于 64Mb 的模型的替代方案，请联系我们。)

4. 避免将源移动到你自己的分支下

从实现时开始，我们一直从主分支构建 TensorFlow Serving 源，最新的版本分支（v0.4）在功能和错误修复方面落后于主分支。因此，如果你只通过检查主分支来创建源，一旦新的更改被合并到主分支，你的源也可能改变。为了确保人工制品的可重复构建，我们发现检查特定的提交修订很重要：

- TensorFlow Serving
- TensorFlow (TensorFlow Serving 里的 Git 子模块)

期待未来加入的一些功能增强清单

这里是一些我们比较感兴趣的希望以后 TensorFlow Serving 会提供的功能：

- 健康检查服务方法；
- 一个TensorFlow Serving实例可以支持多种模型类型；
- 直接可用的分布式存储（如AWS S3和Google存储）中的模型加载；
- 直接支持大于64Mb的模型；
- 不依赖于TensorFlow的Python客户端示例。

人工智能永恒的春天已经到来，你准备好了吗？

作者 Andrew Ng，译者 杨旸



译者在印尼Manado拍摄，Goniobranchus tritos，一种体长8厘米的海蛞蝓。

本文基于 Andrew Ng 在斯坦福 MSx 论坛的演讲（Artificial Intelligence is the New Electricity），经演讲人授权，由 InfoQ 中文站总结并分享。

2017 年 2 月，百度首席科学家、Coursera 的联合创始人 Andrew Ng 在斯坦福 MSx 未来论坛上的一个演讲，吸引了全球的眼球。他认为，人工智能（AI）对许多行业带来的变革，如同 100 多年前，美国“触电”一样——电对制造、运输、农业（尤其是冷藏）、医疗等等带来了划时代的变革。

AI 驱动着百度的搜索和广告，调度百度外卖的快递员，选择路线，和预估运送时间。AI 正在彻底改变金融工程，对物流的转变进行了一半，

医疗和自动驾驶刚开始，而前景巨大。和“电”带来的变革一样，很难想象哪个行业不会被 AI 改变。

监督学习

驱动百亿的市场容量的，基本上属于同一种AI：监督学习(Supervised learning)，即用AI来确定A-->B的映射——输入A和响应B的映射。

- 用Email作为输入A，判断是否是垃圾邮件是响应B。
- 用图像作为输入，识别这是一千种物体中的哪种？
- 从声音A到文字B，从英文到法文，或从文字到声音。

软件可以学习这些输入A到响应B的映射——有很多好的工具来让机器学习。比如50,000小时的音频和对应的文本，就能让机器学到如何从音频内容转化为文本内容。通过大量的电邮数据和区分垃圾的标签，也可以很快地训练出一个垃圾邮件过滤器。

现在的AI还很初级——A到B的映射而已，不过已经推动着很大的市场。百度有很好的算法来预测某用户是否会点击某广告。向受众呈现更相关的广告，能为互联网营销和广告公司带来极大的赚钱机会。这可能是AI最赚钱的应用。

在哪些产品里能用到AI？

产品经理常常希望了解AI能实现的，和不能实现的。一个简单的思路是：一般人能在一秒内想出来的事情，现在或很快就可以用AI自动实现。

AI进展最快的领域正是人能做得到的领域。比如自动驾驶。人类能驾驶，所以AI也能驾驶。在医学影像阅片和分析上，人类放射科医生能够阅片，所以AI也很可能在未来几年内做到。

而人类难以做到的事情，比如预测股市变化，AI可能也难。

- 原因1：人类能做的，至少是可行的；
- 原因2：可以利用人类的数据作为培训样本，比如前面提到的输入A和响应B；

- 原因3：人类能提供指导。如果AI对某个放射影像的结论有误，设计者可以向医生请教，医生所做的正确结论的原因是什么？进而对AI进行改善。

在 Andrew Ng 所接触到的 80–90% 的 AI 项目中，都遵循这一规律：在人类能做到的领域，AI 的进展更快。很多项目的发展一旦超越人类水准，发展也会变得缓慢。这也带来一个社会矛盾：如果 AI 和人的水平类似，实质上是跟人类竞争。

AI 的发展趋势

AI 已经出现了几十年了，而近五年发展明显加速，为什么？

当以前的机器学习算法性能上升到一定程度，即使再增加数据样本量（前文谈到的输入 A、响应 B 的 A-B 映射），性能改善也很有限。似乎超过一定样本量之后，再多的数据也对算法不起作用。

而过去几年，主要由于 GPU，我们终于实现了能利用这么巨大的数据集的机器学习软件。将数据输入一个小的神经网络，当超过一定性能后，上升变得平缓。而不断地把数据输入一个很大的神经网络时，即使性能上升没有那么快，也会保持上升趋势，随着数据量的增大，不断提高。

因此，要想获得很好的 AI 性能，需要两样东西：

- 很大的A-B映射的数据集；
- 大的神经网络。现在常用的大型神经网络建立在HPC高性能计算集群上。

现在的大型 AI 团队包括机器学习和高性能计算两组人，才能获得足够计算能力。百度 AI 团队里的这两种人员都专注于各自领域，没有人能两者兼备。

什么是神经网络？有没有可能取代人类大脑？

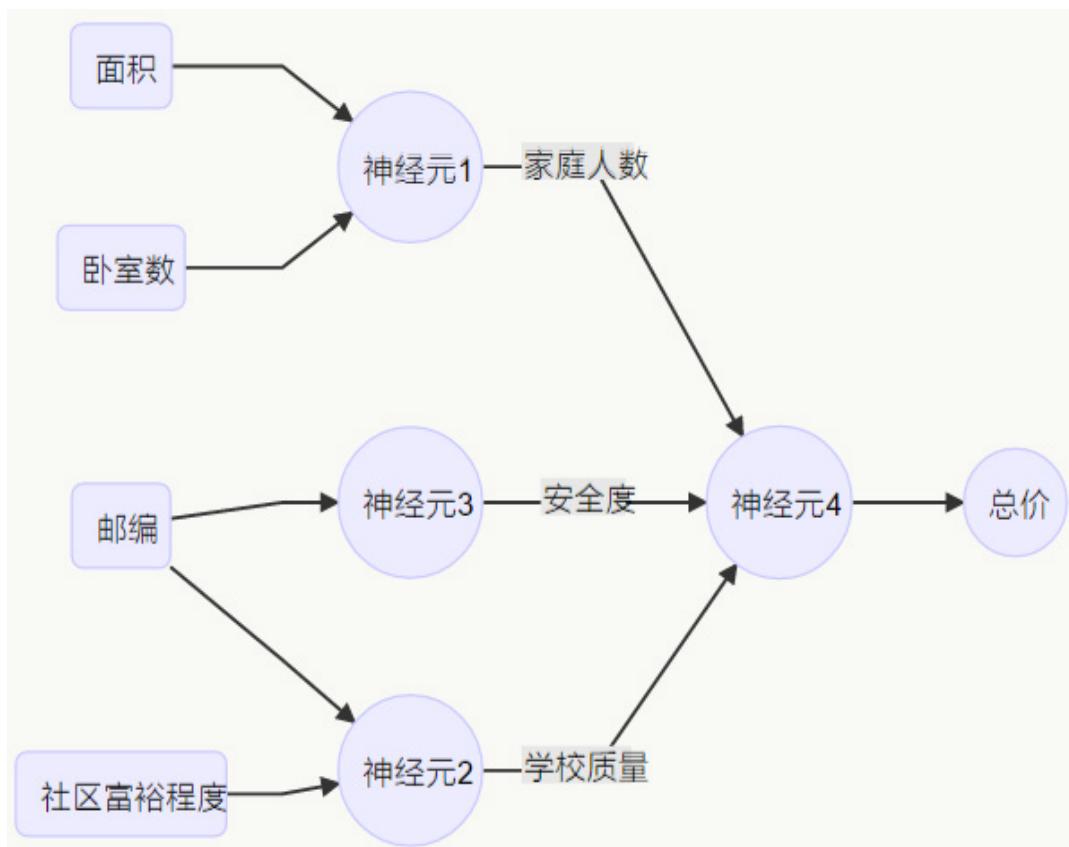
问题是，我们不清楚人脑如何工作，所以很难造出取代人类大脑的神经网络。

什么是神经网络？先看个最简单的神经网络：



如果想输入房屋面积，得到房屋总价，可以用面积 – 总价的一阶近似的线性模型来描述这个神经网络。

或者用更多因素建模，比如通过面积和卧室数，从第一个神经元得到可以支持的家庭人数。再通过所在地址的邮编和社区富裕程度，从第二个神经元得到附近学校的质量。



这就成为一个神经网络。面积、卧室数、邮编、社区富裕程度属于“输

入”集合 A，总价属于“响应”集合 B。

好处在于，当训练这样一个神经网络时，用户无需关心中间因素，诸如家庭人数、安全度、学校质量等，也无需关心每个神经元如何将输入映射到中间结果。只需要给出输入集合 A 和响应集合 B，神经网络将自动形成中间的计算过程和参数。当 A 和 B 的集合足够大，神经网络可以自动算出很多东西。神经网络看上去非常简单，让很多初学者觉得有点失望，但它确实能解决很多问题。关键在于数据量要够大——几万或几十万个样本本身能提供大量的信息，而软件本身只是一小部分。

如何保护 AI 业务？

AI 研究较前沿的团队都比较开放，常常发布研究成果。百度的 AI 研究论文也没有隐藏什么成果——在人脸识别等论文里，都分享了所有的细节。既然很难把算法本身隐藏起来，如何保护 AI 业务？当前稀缺资源有两种，一种是数据，二是人才。获取巨量数据很难，要包括输入 A+ 响应 B。比如语音识别用了 5 万小时的音频来训练，今年准备用 10 万小时，相当于百度 10 年积累的音频。

以人脸识别所用的训练图像数量为例：

- 学术上最常用的基准测试/比赛：1 百万幅；
- 所用图像数最多的计算机视觉对象识别学术论文：1500 万幅；
- 百度用来训练世界上最先进的人脸识别系统：两亿幅！

如果只是 5-10 人的研发团队，很难获得这样规模的数据。百度这样的大企业的经常推出一些新产品不一定是为了营收，而是为了数据，然后通过后续的产品来获得收益。

另一个稀缺资源是人才。AI 的应用需要根据具体业务场景来定制。仅仅下载个开源包，无法解决问题。实际情况下，是否适合用某种垃圾邮件识别或语音识别技术？针对某种场景，机器学习怎么用？所以各个公司都在为数据挖掘争夺 AI 人才，来定制 AI 技术，找到所需要的 A 和 B 各自代表什么，怎么找到这些数据和如何调整算法来适应业务场景。

AI 的良性循环

- 先做出某种产品。比如通过语音识别，以语音实现搜索；
- 然后吸引来很多用户，用户产生数据；
- 再通过机器学习，用数据改善产品。

这就形成了 AI 产品的良性循环。最好的产品能获得最多的用户，带来最多的数据，通过现代机器学习体系，能得到最好的 AI，最终让产品变得更好，周而复始。

百度发布新的产品，会特别考虑怎样推动这样的良性循环，会包括相当先进的产品发布策略，比如按地理区域、细分市场等，来更好地推动这个循环。

这种良性循环的理念很早就有了，只是最近变得更加明显。正如前文所述，当数据超过一定规模后，传统 AI 算法无法明显改善 AI 性能，因此数据多的优势不明显，大公司也很难保护自己的 AI 业务。现在数据越多，AI 性能越好，大公司也更容易保护自己的优势。

AI 炒作的非良性循环

许多人担心 AI 会不会取代或威胁人类。有一小部分研究 AI 的人专门从事对“邪恶 AI”的炒作，以获得投资人或政府机构的投资，来研究“反邪恶 AI”。道高一尺，魔高一丈，又进一步推动对“邪恶 AI”的炒作，从而形成非良性循环，非常不健康。

担心 AI 变得邪恶，类似于担心火星的未来人口过剩。现在看不出 AI 将会怎样走偏，因此也谈不上有针对地研究相应措施。研究本身没有问题，不同的研究是好事，但是对邪恶 AI 的研究占用不恰当的资源，就不应该了。两个人，或者 10 个人来研究邪恶 AI 也许没问题，但是现在投资得太多。

AI 对就业的影响

AI 对就业带来的影响更让人担心。有些 AI 项目确实是瞄准了某些人

类岗位，而从事这些工作的人并不清楚严重性。硅谷创造了大量财富，但也应该对其造成的问题承担责任，比如造成的失业问题。AI 取代人类岗位的现实问题，更应该引起重视，而不是被邪恶 AI 的炒作转移了注意力。

AI 产品管理

AI 是个让人兴奋的领域，同时也存在一些挑战。如何将 AI 融入公司业务？

产品经理的职责是找到用户喜欢的，而工程师的角色是做出可行的产品。两者共同协作，才能做出理想的产品。

AI 是个新生事物，所以技术公司以前的流程和工作方法，不太适用。硅谷的产品经理和工程师的合作已有一套标准流程。比如开发 APP 时，产品经理先画出线框图，比如 logo，按钮，各个板块等，工程师再写出代码来实现。但是 AI 的 APP 无法通过画线框来描述。通过什么形式，把产品经理头脑里对 AI 产品的功能要求明白地分享给工程师呢？

比如开发语音识别系统，实现语音搜索，有很多改善方向。比如：

- 在嘈杂环境下如何改善，比如车里或咖啡馆？
- 仅改善窄带语音信号；
- 对不同口音改善。

百度发现，产品经理通过数据和工程师沟通，是个较好的办法。产品经理负责提供测试数据集给工程师，比如一万个音频和对应的文字，来说明所关心的问题，工程师也能更明白需要解决的问题。如果这些音频里有大量车辆噪音，工程师就知道车辆噪音是问题。如果是混合了几种不同噪声，工程师也能想办法解决。最糟糕的情况是，产品经理提供的测试数据，并不能代表自己想解决的问题，那就出问题了。

同时，新产品设计的流程有很多，比如想设计一个交流型 AI 机器人：

- 人：“我想叫个外卖”；
- AI：“你喜欢哪种类型餐馆？”；
- 人：“川菜”；

- AI：“这些可供选择，xxx,yyy,zzz,...”。

线框图只能显示对话过程，无法描述所需 AI 的复杂程度等。百度的产品经理和工程师会在一起，写五十种对话，

- 人：“请帮我定一个结婚纪念日的餐馆。”
- AI：“你需要订花吗？”

这时候，工程师会问一些更具体的问题，比如每种场景是否都需要继续提配套产品的问题，比如谈到圣诞节时，是否要问对方要不要买圣诞装饰？一起思考，共同讨论需求和技术，很有效。

对 AI 的宣传里，有很多吸引眼球的技术，不过它们未必最有用。如何将吸引眼球的技术和产品、业务相结合？软件产业已经有标准流程，比如代码审查、敏捷开发等，如何组织 AI 的产品工作，有很长的路要走，现在正是考虑这些问题的时候。

短期内，AI 有哪些机会？

语音识别正在起飞

最近准确率已经提高到很有用的程度。4-5 个月之前，斯坦福大学计算机系教授 James Landay、百度、华盛顿大学一起展示了在手机上输入英文和普通话，用语音识别的速度比用手机输入快 3 倍。去年百度的所有语音识别产品年度环比增长大约 100%，现在正是语音识别技术腾飞之时。美国有几个公司做智能语音控制器（Smart Speakers），用语音控制家用设备也会很快推广。相关的操作系统和硬件都会很快发布。

计算机视觉也即将到来

中国的人脸识别发展速度很快。因为中国的手机比笔记本更普及，很多人有手机，而不一定有笔记本。在中国可以仅仅凭手机申请助学贷款。涉及到钱，所以需要先验证身份和很多东西。这加速了人脸识别的发展。通过手机进行人脸识别，作为用生物标识进行身份认证的一种方法，在中国发展很快。

在百度总部，不需要 RFID 卡进行认证，而是直接刷脸进门，Andrew Ng 在 YouTube 上有一段视频。现在人们对人脸识别技术已经足够信任，并在安全要求较高的场景下使用。

百度在语音识别和计算机识别上的资金投入和数据投入巨大，任何小开发团体远远无法相提并论，也不太可能有其他出乎意料的技术突破。

医疗健康的AI应用

Andrew Ng 对 AI 对医疗健康领域带来的影响很看好。很多现在的放射科医生会被 AI 影响到。如果想在放射科一直工作四十年，不是个好的职业计划。

还有很多垂直领域将受到 AI 的影响，比如金融工程和教育。不过短期之内还不太会对教育产生实质性的影响。

永恒的春天

光从监督学习已经看得出 AI 将如何逐渐改变各个行业。其他的 AI 形式，比如无监督学习、强化学习、迁移学习等等，都还在研究阶段，现在的市场规模较小。

有很多行业会经历几个冬天，然后迎来永恒的春天。AI 经历过两个冬天，现在已经进入永恒的春天。就像硅的春天一样，半导体、晶体管、计算周期这些都将和人类一起发展很久。神经网络和深度学习会繁荣很长时间，一百年或许太远，但一些重要应用改变几个大行业的路线图已经很清晰。

AI 确实正在取代人类的一些岗位。当某些岗位被 AI 取代后，我们需要新的教育系统，来帮助失去工作的人获得新的技能。政府应该为这些愿意学习新技能的人，提供基本收入保障，重新成为劳动者的一员。我们需要新的系统和结构，来让帮助社会向新世界进化。虽然会有新类型的工作，但工作岗位的消失也比以前更快。

一些问题

1. 大公司在数据和人才上有巨大优势，那么创业公司的机会在哪里？ 投资者可以关注哪种规模的创新？
 - 在语音识别、人脸识别上，小公司非常难与大公司竞争，除非有意料之外的技术突破。同时，也有很多小垂直领域适合创业公司，比如医疗影像。有一些疾病的病例不多，如果有一千张影像，也许就涵盖了所有所需的数据了，一些垂直领域需要的数据量也不大。
 - 另外，AI的机遇非常多，大公司会放弃很多的小的垂直市场，因为精力有限，大的机会还研究不过来。
2. AI在发明创造上，有哪些进展？
 - 还很早期。AI可以作曲，但这很主观。20年前的技术做出来的曲子有人喜欢，有人不喜欢。有些项目用AI制作图片特效，用特效模仿某画家作品，这些都是小而有趣的领域。 现在还看不到有什么技术路线能发明复杂的系统。
3. 如果摩尔定律不再成立，对AI的扩展性有什么影响？
 - 一些高性能计算公司的硬件路线图显示，摩尔定律在单芯片上不再那么有效，但神经网络、深度学习所需的计算类型在未来几年仍然能很好地扩展。SIMD（单指令多数据）让并行化处理负载非常容易。神经网络很容易并行化，加速计算的空间还很大。
 - AI面对的诸多问题中，许多问题的瓶颈在于数据，也有很多的瓶颈在于计算速度——能便宜地处理数据的速度赶不上获得数据的速度。所以高性能计算的路线图应该包括这方面。
4. 算法是AI里的特殊作料。是否应通过知识产权保护，还是绕过这个问题去设计产品？ 对机器学习的研究者，是否有和AI产品经理-工程师那样类似的流程或良性循环，来实现突破或改善研究流程？

- 知识产权的问题比较难讲。有些公司申请了大量专利，但是是否真能带来实质性的保护？所以我们往往从如何从战略上思考细节，比如让数据保护自己。
 - 研究机构更偏好新鲜、抢眼球的东西，来发表论文。训练新研究者的方法通常是读很多论文。而大家常常忽视重复论文里的试验的重要性。不一定要把精力大量用于发明新东西，而花时间重复别人的发布结果也是很好的培训方法。和培训博士生一样：去学习和理解别人的论文，重复别人的试验，争取获得类似的结果，很快你就能产生自己的想法去推动最新的科技。
5. 对希望从事机器人相关工作的机械工程学生，有哪些和AI、机器人相关的机会比较适合？
- 很多机械工程背景的人，在AI领域很成功。可以上一些计算机/AI课程，和AI领域的老师聊聊。一些垂直领域存在有趣的AI机器人的机会，比如精准农业。Blue River用计算机视觉来区分不同植物，比如不同品种卷心菜，选择留下哪些，除掉哪些，来提高产量。
 - 中国也生产和销售很多社交和伴侣机器人，美国还没起怎么起步。
6. 让AI和人配合起来的前景如何？很多AI应用是基于AI自己，如果采用AI+人的混合方案？比如自动驾驶等？
- 没有统一的规则，应该跟实际情况有关。很多语音识别是为了让人类更高效，比如通过手机。对自动驾驶汽车，可能需要10–15秒来转换控制权，因为很难让容易分神的人快速接手驾驶，很困难。这种情况下，由AI独立控制更安全。所以从使用者角度来讲，人类和AI混合的自动化比较困难。
7. 对在线教育而言，主要问题是动机，人们不愿意花那么多时间来学完整个课程。这是不是最大的挑战？其他还有什么挑战？
- AI对在线教育有帮助。个性化的辅导已经谈论了很长时间，

Coursera用AI推荐个性化的课程，自动打分，在细节上确实有帮助。但在利用AI之前，教育的数字化还有很长的路要走。很多行业都有个规律：先有数据，再有AI，比如医疗，美国电子病历（EHR）的进展很大。随着电子病历的兴起，影像胶片变成数码图片，这些数字化产生了很多数据供AI使用，并产生价值。教育需要先经历数字化，这一阶段还有很多工作要做。

8. 百度如何用AI来管理自己的云上数据中心？比如IT运维管理的例子？
 - 两年前，百度做了个项目，可以提前一天自动检测出硬件故障，特别是硬盘故障。这就可以事先拷贝、热插拔进行预防处理。还可以降低数据中心的用电量，负载均衡等，都是很多小细节的改善。
9. 能否举一些例子说明能通过仔细地建模和规划，用AI解决的复杂问题？对这些问题，人类可能需要进行长时间的思考。
 - 亚马逊是个很好的例子。它知道我浏览过什么，读过什么，比太太更了解。电脑对人们看过什么，点击过什么广告更了解，所以在广告方面做得非常好。对于有些任务，计算机可以处理的信息量远远超过人类，并根据规律建模，进行预测，这方面AI比人做得更好。
 - 将AI融入人类工作的很大一部分，是将一块块的AI部分串成一个大系统。比如为了造自动驾驶汽车，要用相机拍摄的图像，雷达等，组成车前方的一幅图，再由监督学习估算和其他车的距离，以及和行人的距离，这只是两个重要的AI部件，还需要其他的部件来估计5秒后车的位置，行人的方向。还有一个部件来分析，根据行人车辆等不同对象的运动情况，我应该怎么走？然后还需要算方向盘的旋转程度，以此类推。
 - 所以复杂的AI系统有很多小AI部件，工程人员要知道如何将这种超级学习能力融合到更大的系统里，来创造价值。

10. 产品经理和社会学家、律师等如何协调？比如自动驾驶汽车在撞人前，开发者和AI应从驾驶者，还是行人的角度考虑问题？这只是个法律问题，但也有很多类似情况。产品管理者和不同的功能部门的合作时，应该扮演什么角色？
- 这个问题的一个相似版本是“有轨电车”问题，会产生伦理矛盾。一个电车走到岔道口，继续往前会撞死5个人，你可以用扳手将电车扳到另一条轨道，撞死该轨道上的一个人，而你成为凶手，你扳吗？
 - 除了在哲学课里，很少有谁在现实生活里遇到过这个问题，所以，它并不重要。自动驾驶的开发者没去讨论它。实际上，如果谁真正遇到了，可能之前已经犯了其他错误了。自动驾驶处理的问题更实际，和你自己开车一样。比如，对面有个白色的大车，是否能及时刹车？



本期主要内容：谷歌新发布的分布式数据库服务；Yahoo 开源 TensorFlowOnSpark；百亿级微信红包的高并发资金交易系统设计方案；复盘 GC 算法的发展历程及现状



解读2016

许多年后，如果我们回过头来评点，也许 2016 年是非常重要的一个时间节点。



顶尖技术团队访谈录 第八季

本次的《中国顶尖技术团队访谈录》·第八季挑选的六个团队虽然都来自互联网企业，却是风格各异。希望通过这样的记录，能够让一家家品牌背后的技术人员形象更加鲜活，让更多人感受到他们的可爱与坚持。



架构师特刊 用户画像实践

本电子书中几个作者介绍一个公司如何从无到有的搭建用户画像系统，以及其中的技术难点与实际操作中的注意事项，实为用户画像的实操精华之选，推荐各位收藏阅读。