Alibaba Group 阿里巴巴集团 | 阿里技术

2017 阿里技术年度精选（上）

- 阿里前沿技术精华总结
- 覆盖多个热门技术领域
- 10+位技术大牛独家访谈

赠阅

阿里技术

扫一扫二维码图案，关注我吧

「阿里技术」微信公众号

「阿里技术」官方微博

Alibaba Group 阿里巴巴集团 | 阿里技术

2017 阿里技术年度精选（上）

阿里技术

序

你好，我是阿里妹，第一次这么正式地写序，有点紧张。

2017 年，在技术发展的历史上，一定是个特别的一年：柯洁与 AlphaGo 的惊世大战，无人咖啡店开放体验，AI 设计师“鲁班”横空出世、三年投入千亿的达摩院正式成立……

技术前进的脚步，比我们想象中更快。同样在今年，阿里技术公众号，正式迎来了第 35 万位关注者，也有幸发布过数篇 10w+ 乃至百万阅读的文章。

PS：本想借此搞个联谊活动，瞅瞅后台性别比例 9:1，又默默放弃了。

有读者留言告诉我们，每天早上坐公交车上班，看阿里技术推送的文章，已经成为了一种习惯。这让我们倍感不胜荣幸，又更觉肩上责任之重。

我们始终相信，每天一篇干货，效果虽不能立竿见影，但聚沙成塔、滴水成川，你看过的东西，会成为你的一部分，潜移默化影响更多的人。

阿里技术公众号，看似只有阿里妹一个人运营。但在它身后，其实有超过 25000 名阿里工程师的力量。这群可爱的作者里，有刚走出校门的新人，有团队中的中流砥柱资深工程师，有带领数百、上千人的技术管理者。他们在工作之余，抽出宝贵的休息时间，梳理自己对业务、技术、人生的思考和理解。

没有华丽的词藻、繁复的修辞，只有朴质的代码、反复推敲的算法。一篇文章看似只有几千字，但在背后，往往是数十人乃至数百人的团队，长达数月、数年的摸索、跌倒，终于探得的成果。他们把这一路的所得，迫不及待地分享给业界同仁，希望帮助大家少踩坑，少加班。

这次，在全年发布的近 300 篇文章中，我们选出 65 篇，集结成这套《2017 阿里技术年度精选》，分为上、下两册，总计 600 余页。

这套精选集覆盖多个热门技术领域：算法、机器学习、大数据、数据库、中间件、运维、安全、移动开发等，文章内容涉及技术架构、核心算法、解决方案等干货。无论你是计算机相关专业的在校学生、科研机构的研究人员，还是步入社会的 IT 从业人员，相信都能从中受益。

这套书同时收录了十多位阿里技术人的访谈：从工程师到合伙人的多隆，6 年时间影响数亿用户的靖世，入选 MIT2017 年度 TR35 的王刚 & 吴翰清，免试晋升为研究员的钱磊等，将为你展现不一样的技术人生。

它不是一本系统讲述某个领域的书，更像是一本技术杂文选集，内容五花八门。翻开书来，一眼望去，皆是散落在各个技术领域的结晶。你可以把它当作床头书，或是在旅行的路上随手翻翻，充充电。希望这本书，能为你打开一扇窗，去看更大的世界；成为一个小支点，帮你撬动更大的进步。

因为相信，所以看见。这世界依然浮躁，但庆幸始终有人相信，技术让生活更美好。

感谢坚持分享、笔耕不辍的阿里工程师，

感谢所有关注阿里技术的小伙伴，

感谢所有的相遇与陪伴。

希望这套书，能陪你一起回味即将逝去的 2017。

2018，我们一起遇见更好的自己。

最后，阿里妹也期待你的回信，聊聊你的感想与建议：lunalin.lpp@alibaba-inc.com

阿里妹

2017 年 12 月

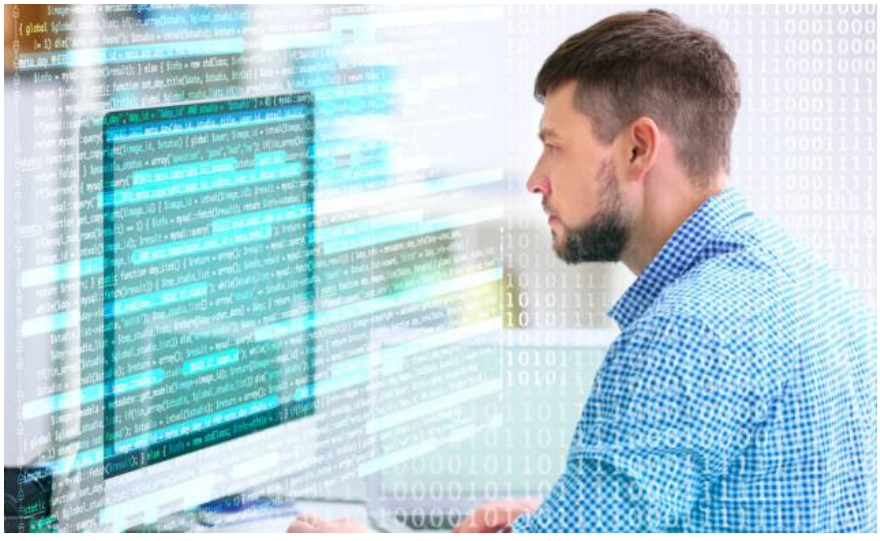
目录

存储 / 数据库

深度解读 | 阿里云新一代关系型数据库 PolarDB

贺军

阿里妹导读：本文通过描述关系型数据库发展的背景以及云计算的时代特征，分享了数据库计算力的螺旋式上升的进化理念，另外结合阿里云 RDS 产品的发展路径，阐述了自主研发的新一代云托管关系型数据库 PolarDB 的产品整体设计思想，对一些关键技术点进行了解读。



关系型数据库

谈到关系型数据库，在这个知识日新月异的 TMT 时代，听起来有些“古董”，这个起源于半个世纪以前的 IT 技术，事实上一直处于现代社会科技的核心，支撑着

当今世界绝大多数的商业科技文明。CPU、操作系统、数据库这三大核心领域，基本上就是 IT 时代的缩影，同时也是一切信息化处理、计算力和智能化的基石。

从 1970 年 E.F.Codd 发表了一篇里程碑论文“A Relational Model of Data for Large Shared Data Banks”，到 80 年代初期支持 SQL 的商用关系型数据库 DB2，Oracle 的面市，以及 90 年代初 SQL-Server 的诞生，都是关系型数据库成功的代表。

时至今日，随着全球互联网的发展，大数据技术的广泛应用，涌现出越来越多的新型数据库，然而关系型数据库仍然占据主导地位。最主要的原因之一就是关系型数据库采用了 SQL 标准，这种高级的非过程化编程接口语言，将计算机科学和易于人类理解认知的数据管理方式完美的衔接在了一起，目前还难以超越。

SQL 语言

SQL(Structured Query Language) 语言是 1974 年由 Boyce 和 Chamberlin 提出的一种介于关系代数与关系演算之间的结构化查询语言，其本质是用一种类似于自然语言的关键字和语法来定义和操作数据，进行可编程的数据存储、查询以及管理。

这种抽象编程接口，将具体的数据问题与数据的存放、查询实现的细节解耦开来，使得商业业务逻辑以及信息管理的计算模式能够被大量复制和应用，解放了生产力，也极大的促进了商业化关系型数据库自身的发展。

从 SQL 后来不断发展和丰富来看，SQL 已经成为关系型数据库语言的标准和王者。到今天这种编程语言还没有更加完美的替代品。

OLTP

1976 年，Jim Gray 发表了名为”Granularity of Locks and Degrees of Consistency in a Shared DataBase”的论文，正式定义了数据库事务的概念和数据一致性的机制。而 OLTP 是关系型数据库涉及事务处理的典型应用，主要是基本的、日常的事务处理，例如银行交易。

事务处理需要遵循 ACID 四个要素来保证数据的正确性，包括原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）。而衡量 OLTP 处理能力的性能指标主要有响应时间、吞吐率等。

开源数据库生态

在我们简要的回顾了关系型数据库的历史、地位和发展阶段后，我们不难看到 Oracle、SQL-Server、DB2 等关系型数据库仍然占据着全球商业数据库的主导地位，虽然曾经耳熟能详的 Informix、Sybase 已经淡出大众的视线。

然而，从 20 世纪 90 年代开始，另一股推崇知识分享、自由开放的软件精神成为趋势和潮流，尤其以 Linux、MySQL、PostgreSQL 等为代表的开源软件，开始以强大的生命力不断发展壮大，释放出巨大的社会进步力量，这些被自由分享的科技红利，孕育和促进了全球互联网高科技公司的飞速发展。

这是整个人类社会的进步，我们要感谢那些开源软件的斗士们，Richard Stallman，Linus Torvalds，Michael Widenius 等。当然，最近几年国内涌现出越来越多积极参与到开源主流社区的中国公司，也在不断地将技术分享回馈给开源世界。

根据 DB-engines 网站的最新统计，不难发现，当把开源数据库 MySQL 和 PostgreSQL 加在一起，开源数据库就已经超越了商业数据库 Oracle，成为世界上最流行的关系型数据库。

330 systems in ranking, August 2017

Rank			DBMS	Database Model	Score		
Aug 2017	Jul 2017	Aug 2016			Aug 2017	Jul 2017	Aug 2016
1.	1.	1.	Oracle	Relational DBMS	1367.88	-7.00	-59.85
2.	2.	2.	MySQL	Relational DBMS	1340.30	-8.81	-16.73
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1225.47	-0.52	+20.43
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	369.76	+0.32	+54.51
5.	5.	↓ 4.	MongoDB	Document store	330.50	-2.27	+12.01
6.	6.	6.	DB2	Relational DBMS	197.47	+6.22	+11.58
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	127.03	+0.90	+2.98
8.	8.	↓ 7.	Cassandra	Wide column store	126.72	+2.60	-3.52
9.	9.	↑ 10.	Redis	Key-value store	121.90	+0.38	+14.57
10.	10.	↑ 11.	Elasticsearch	Search engine	117.65	+1.67	+25.16

云计算当前的阶段

如果说关系型数据库是 IT 时代的产物。那么在互联网时代的云计算，关系型数据库目前正处于一个什么阶段呢？ IT 时代从某种程度上讲，更多是创造了计算力，那么进入互联网时代的云计算，则是专注于用户和计算力的连接，提供无处不在的计算力，这个其实是云计算商业模式的成功所在，可以称之为 1.0 版本。而云计算 2.0 版本，需要在云环境下，重新进化和升级计算力，这种进化体现了社会计算力的整合以及计算资源能效的进步。

为了顺应绿色计算以及共享经济的发展潮流，不仅仅需要云服务器，云数据库，网络互联，硬件芯片等等各个软硬件系统领域的融合以及演进升级，还需要坚持科技以需求为本、服务以用户为根的科技普惠大众的理念来进一步促进计算效率和计算智能的提高。

我们正处在一个蓬勃发展的云计算 2.0 阶段。在这个阶段，关系型数据库在云托管环境逐渐暴露出一些问题，作为在云计算时代的先行者，Amazon 于 2014 年 11 月 12 日的 AWS re:Invent 2014 大会，发布 Aurora 云托管关系型数据库就是为了解决这些问题。这个新一代的数据库的发布，也昭示着云计算时代，传统的 IT 技术核心产品将揭开自我进化的序幕。

而 2017 年 SIGMOD 数据大会，Amazon 发布了论文”Amazon Aurora: Design Considerations for High Throughput Cloud Native Relational Databases”，更加开放的解释了基于云环境的 Cloud-Native 设计的关系型数据库是如何应孕而生的。

为什么阿里云研发 PolarDB？

在我们回顾了关系型数据库以及云计算的背景之后，我们不难发现，云计算 1.0 虽然解决了用户和计算的连接的问题，但是还需要进一步解决在一个共享计算的环境下，传统关系型数据库和公有云服务环境的融合问题。

云计算 1.0 用低廉的成本，灵活快速的部署、弹性和扩展能力，获得了传统 IT 计算上云的转换动力。在低成本享受普惠科技成为常态之后，随着用户业务的增长，

用户新的痛点开始出现，例如，如何从根本上解决用持续低的成本，享受和传统 IT 计算力一样，甚至更好的云服务，成为迫切需要。

这初看起来像伪命题，仔细分析之后，却淋漓尽致的体现了螺旋式上升的哲学思想。就好像在 PC 服务器涌现的时代，PC 服务器首先用低廉的价格提供了和小型服务器接近的计算能力，然后在保持成本和性价比优势的基础上，实现了超越小型服务器的性能优势，直至终结了小型服务器时代，开始了 PC 服务器时代。

所以说云计算时代还远远没有到达鼎盛时期，除非它通过自身进化演变，在不断保持性价比优势的同时，在具有快速灵活弹性的内在属性基础上，拥有超过传统 IT 计算力的能力之后，云计算才会真正进入它所主宰的时代，这只是个时间问题。

也就是说今天不只是阿里云要做这样一款关系型数据库，而是所有的云计算厂商都不可避免的要经历这样一个阶段。那就是云计算时代传统 IT 计算力的重建和进化！只不过 Amazon 走在了最前面，而阿里云紧跟其后，都需要经历这进化到蜕变的过程。

在这个过程中，新一代关系型数据库是关键的里程碑之一。同理，接下来应该有更多更加高级的云服务，比如智能云操作系统出现，来融合为云时代设计的硬件芯片和网络互联等等。

在 IT 时代，传统的计算力（例如用关系型数据库来处理结构化数据等）是服务于系统硬件隔离环境下的多用户使用场景的。而云计算时代是多客户 Self-Service 租用环境，各种计算负载场景更加复杂，在这种计算负载变迁的环境下，如何解决 IT 时代的技术产物和云计算时代应用环境的适配矛盾，正是云计算自我进化的内在推动力。

例如，在公有云环境下，随着用户的增多，以及用户业务和数据的增长，备份、性能、迁移、升级、只读实例、磁盘容量、Binlog 延迟等相关问题渐渐显现出来。这背后大部分原因是由于 I/O 瓶颈（存储和网络）导致，亟须通过技术革新以及新的产品架构解决这个问题。另一方面，从产品形态来讲，阿里云 RDS 目前的产品形态各具优势，在下一节会详细介绍。

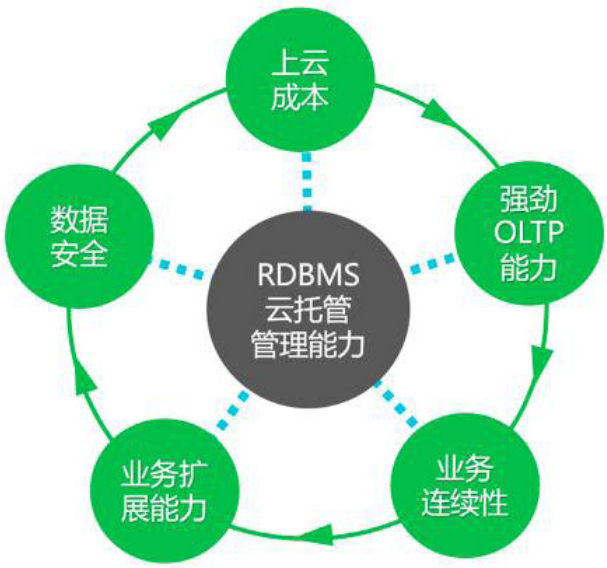
但是从产品架构的发展来看，除去数据库存储引擎的类型之外，对于关系型数据

库，考虑到工程效率以及运维成本，最好有一种通用的产品技术架构能兼顾不同用户场景的需求，而不是针对每一个场景都实现一种对应的技术架构。

在接下来的内容，通过讲述阿里云 RDS 的不同产品形态的特点，我们会更加清晰的了解到，PolarDB 的产品形态正是在吸收了之前几种产品形态的优点而孕育而生的。

PolarDB 的设计思想用户需求和公有云自身发展的选择

作为云托管的关系型数据，除了关系型数据库的核心特征之外。PoalrDB 更多的关注于如何提供满足用户业务需求的云服务，并且通过技术革新，不断进化，在提供更好的数据库计算力的同时，满足用户以下业务需求：上云成本、OLTP 性能、业务连续性、在线业务扩展、数据安全。



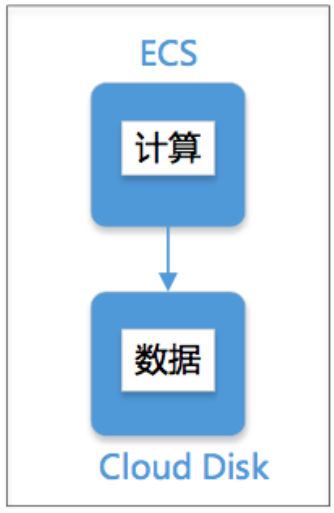
另一方面云计算除了成本优势之外，弹性和可扩展性也是云计算的天然属性。为了用户业务的扩展，更好的 Scale Up 以及故障恢复，计算和存储分离的架构成为云资源环境更好的选择。这一点将在下一节 RDS 产品架构的演进中得到进一步的诠释。

阿里云 RDS 产品架构的演进

如上所述，阿里云 PolarDB 和 Amazon Aurora 数据库进化的方向是一致的，然而进化的路径各有不同。本身来讲，这是由于各自的数据库云服务实现方式不同所

决定的。阿里云 RDS MySQL 有如下几个版本。这些产品形态满足不同的用户业务场景，具有不同的特点，可以进行优势互补。

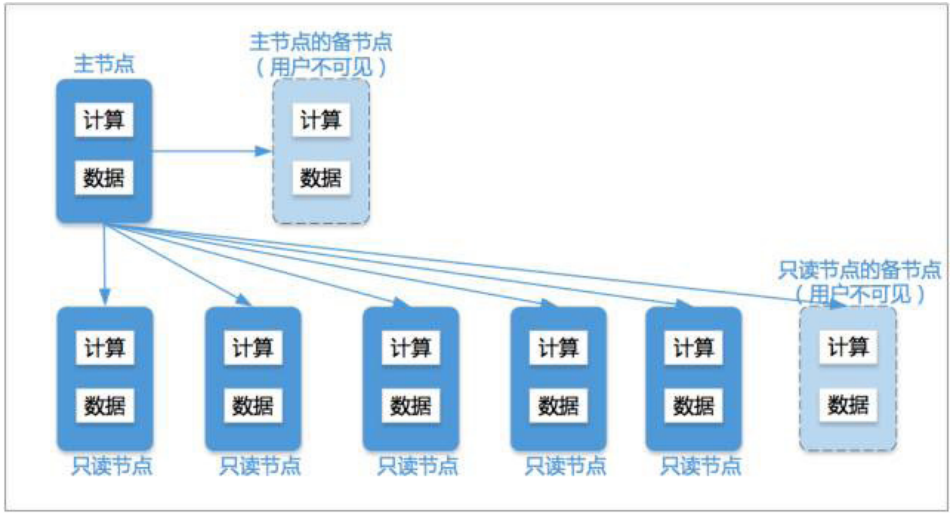
MySQL 基础版



MySQL 基础版采用数据库计算节点和存储节点分离的方式，利用云盘数据本身的可靠性和多副本的特性，同时也利用了 ECS 云服务器虚拟化来提升标准化部署、版本和运维的管理效率，能够满足低端用户不太注重高可用服务的业务场景。

同时这种架构对于数据库的迁移，数据容量的扩容，计算节点的 Scale Up，计算节点故障恢复都有天然的优势，根本原因就是计算和存储的分离。后面也会讲到，PolarDB 也采用了存储和计算分离的设计理念。

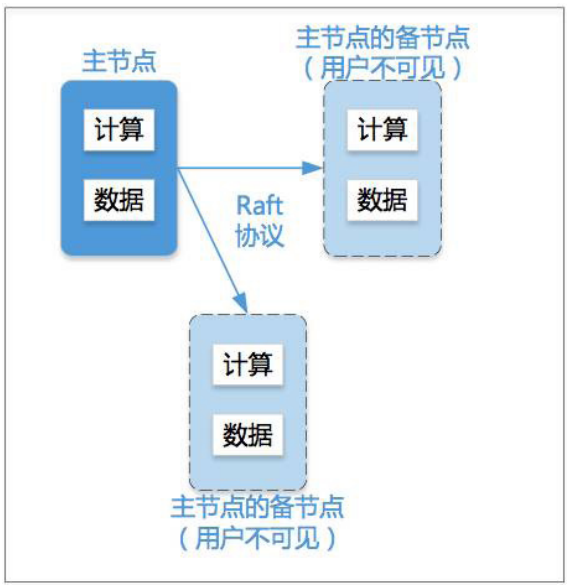
MySQL 高可用版



MySQL 高可用版则是针对企业级用户提供的高可用数据库版本，提供 99.95% 的 SLA 保障。采用 Active-Standby 的高可用架构，主节点和备节点之间通过 MySQL Binlog 进行数据的 Replication。当主节点发生故障，备节点接管服务。

同时还支持多个只读节点，支持负载均衡的数据读写分离的访问方式。采用 Shared-Nothing 架构，计算和数据位于同一个节点，最大程度保障性能的同时又通过数据的多副本带来可靠性。

MySQL 金融版



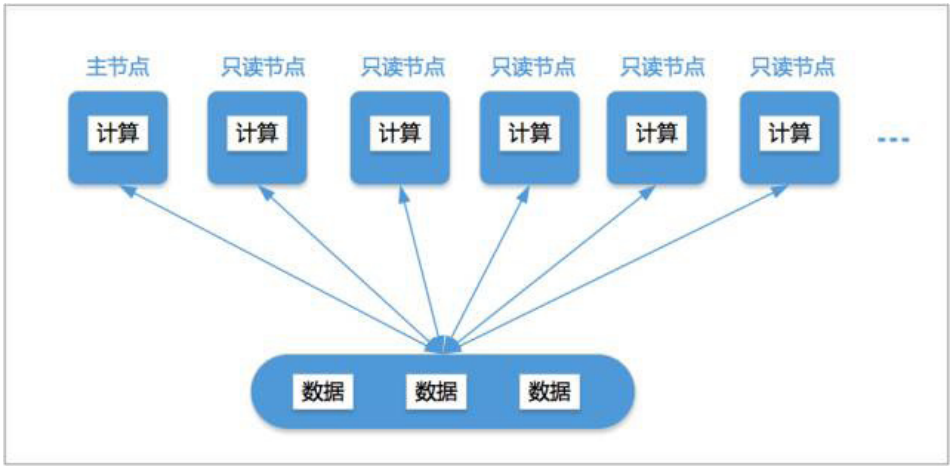
MySQL 金融版可以说是针对金融行业等高端用户设计的高可用、高可靠云服务产品，采用分布式 Raft 协议来保证数据的强一致性，拥有更加优异的故障恢复时间，更加满足数据容灾备份等业务场景的需求。

PolarDB 的进化

PolarDB 采用存储与计算分离的技术架构，同时可以支持更多的只读节点。主节点和只读节点之间是 Active-Active 的 Failover 方式，计算节点资源得到充分利

用，由于使用共享存储，共享同一份数据，进一步降低了用户的使用成本。下一节我们将进一步从细节上描述 PolarDB 的关键特性。

PolarDB 的设计思想有几个大的革新。一是通过重新设计特定的文件系统来存取 Redo log 这种特定的 WAL I/O 数据，二是通过高速网络和高效协议将数据库文件和 Redo log 文件放在共享存储设备上，避免了多次长路径 I/O 的重复操作，相比较 Binlog 这种方式更加巧妙。



另外在 DB Server 设计上，采用 MySQL 完全兼容的思路，完全拥抱开源生态，从 SQL 的编译、性能优化器和执行计划等等都保留了传统关系型数据库的特色。并且针对 Redolog 的 I/O 路径，专门设计了多副本共享存储块设备。

我们知道，分布式数据库一直是数据库领域的热点，具有非常大的实现难度。不管是遵循 CAP 理论，还是 BASE 思想，通用的分布式关系型数据库基本上很难做到技术和商用的完美平衡。与 SQL 标准以及主流数据库兼容，OLTP ACID 事务 100% 支持，99.99% 的高可用，高性能低延迟并发处理能力，弹性 Scale Up，Scale out 可扩展性，备份容灾和低成本迁移等等，能够完美兼顾所有这些特点的商用关系型数据库还没有出现。

阿里云 PolarDB 和 Amazon Aurora 的一个共同设计哲学就是，放弃了通用分布式数据库 OLTP 多路并发写的支持，采用一写多读的架构设计，简化了分布式系

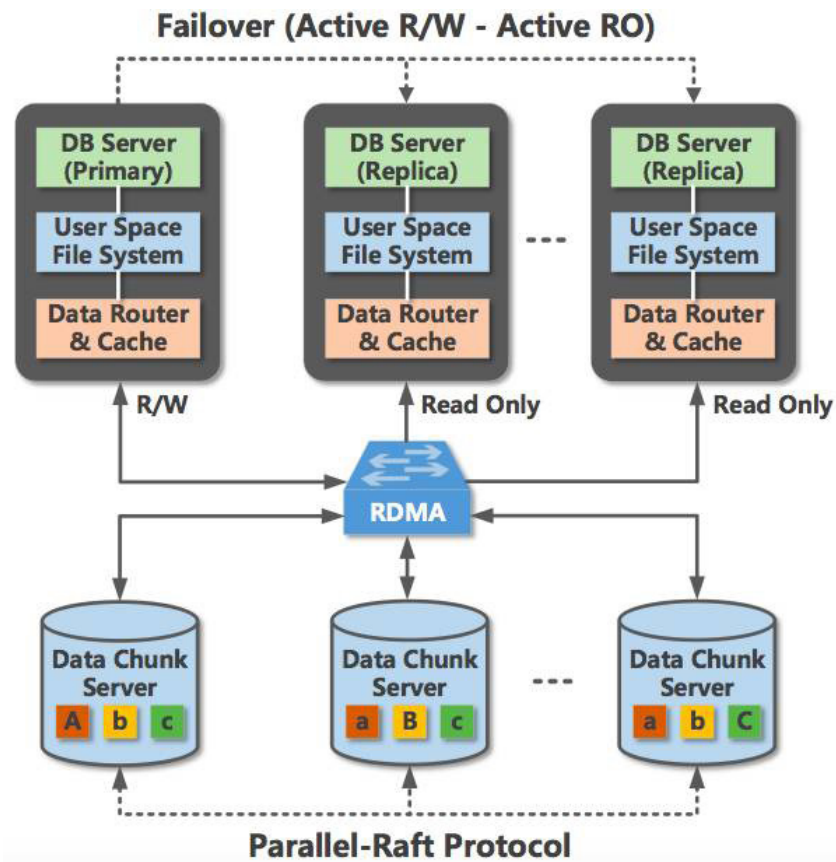
统难以兼顾的理论模型，又能满足绝大多数 OLTP 的应用场景和性能要求。

总之，100% MySQL 的兼容性，加上专用的文件系统和共享存储块设备设计，以及在下文中提到的多项高级技术的应用，使得新一代关系型数据库 PoalrDB 在云时代必将大放异彩。

PolarDB 产品关键技术点剖析

在讲述了阿里云 RDS 的不同产品形态之后，我们再从整体上看一看 PolarDB 的产品架构。下图勾画了 PolarDB 产品的主要模块，包括数据库服务器，文件系统，共享块存储等。

PoalrDB 产品架构



如图所示，PolarDB 产品是一个分布式集群架构的设计。它集众多高级的技术实现于一身，使得数据库 OLTP 处理性能有了质的飞跃。PoalrDB 采用了存储与计算分离的设计理念，满足公有云计算环境下用户业务弹性扩展的刚性需求。数据库计算节点和存储节点之间采用高速网络互联，并通过 RDMA 协议进行数据传输，使得 I/O 性能不在成为瓶颈。

数据库节点采用和 MySQL 完全兼容的设计。主节点和只读节点之间采用 Active-Active 的 Failover 方式，提供 DB 的高可用服务。DB 的数据文件、redolog 等通过 User-Space 用户态文件系统，经过块设备数据管理路由，依靠高速网络和 RDMA 协议传输到远端的 Chunk Server。

同时 DB Server 之间仅需同步 Redo log 相关的元数据信息。Chunk Server 的数据采用多副本确保数据的可靠性，并通过 Parallel-Raft 协议保证数据的一致性。

在描述了 PolarDB 的产品架构之后，我们再分别从分布式架构，数据库高可用，网络协议，存储块设备，文件系统和虚拟化等方面逐一介绍下 PolarDB 使用的关键技术点。

Shared Disk 架构

分布式系统的精髓就在于分分合合，有时候为了并发性能进行数据切分，而有时候为了数据状态的一致性而不得不合，或者由于分布式锁而不得不同步等待。

PolarDB 采用 Shared Disk 架构，其根本原因是上述的计算与存储分离的需要。逻辑上 DB 数据都放在所有 DB server 都能够共享访问的数据 chunk 存储服务器上。而在存储服务内部，实际上数据被切块成 chunk 来达到通过多个服务器并发访问 I/O 的目的。

物理 Replication

我们知道，MySQL Binlog 记录的是 Tuple 行级别的数据变更。而在 InnoDB 引擎层，需要支持事务 ACID，也维持了一份 Redo 日志，存储的是基于文件物理页面的修改。

这样 MySQL 的一个事务处理默认至少需要调用两次 fsync() 进行日志的持久化

操作，这对事务处理的系统响应时间和吞吐性能造成了直接的影响。尽管 MySQL 采用了 Group Commit 的机制来提升高并发下的吞吐量，但并不能完全消除 I/O 瓶颈。

此外，由于单个数据库实例的计算和网络带宽有限，一种典型的做法是通过搭建多个只读实例分担读负载来实现 Scale out。PolarDB 通过将数据库文件以及 Redolog 等存放在共享存储设备上，非常讨巧的解决了只读节点和主节点的数据 Replication 问题。

由于数据共享，只读节点的增加无需再进行数据的完全复制，共用一份全量数据和 Redo log，只需要同步元数据信息，支持基本的 MVCC，保证数据读取的一致性即可。这使得系统在主节点发生故障进行 Failover 时候，切换到只读节点的故障恢复时间能缩短到 30 秒以内。

系统的高可用能力进一步得到增强。而且，只读节点和主节点之间的数据延迟也可以降低到毫秒级别。

从并发的角度来看，使用 Binlog 复制现在只能按照表级别并行复制，而物理复制只按照数据页维度，粒度更细，并行效率更加高。

最后一点，引入 Redolog 来实现 Replication 的好处是，Binlog 是可以关闭来减少对性能的影响，除非需要 Binlog 来用于逻辑上的容灾备份或者数据迁移。

总之，在 I/O 路径中，通常越往底层做，越容易和上层的业务逻辑和状态解耦而降低系统复杂度。而且这种 WAL Redo log 大文件读写的 I/O 方式也非常适用于分布式文件系统的并发机制，为 PolarDB 带来并发读性能的提高。

高速网络下的 RDMA 协议

RDMA 之前是在 HPC 领域被使用多年的技术手段，现在开始被使用到云计算领域，也证实我的一个判断。云计算 2.0 时代，将重建人们对于云计算的认识，云端也能够创造超越传统 IT 技术的计算力，这将是一个越来越严谨的工业实现。

RDMA 通常是需要有支持高速网络连接的网络设备（如交换机，NIC 等），通过特定的编程接口，来和 NIC Driver 进行通讯，然后通常以 Zero-Copy 的技术以达到数据在 NIC 和远端应用内存之间高效率低延迟传递，而不用通过中断 CPU 的方式来进行数据从内核态到应用态的拷贝，极大的降低了性能的抖动，提高了整体系统的

处理能力。

Snapshot 物理备份

Snapshot 是一种流行的基于存储块设备的备份方案。其本质是采用 Copy-On-Write 的机制，通过记录块设备的元数据变化，对于发生写操作的块设备进行写时复制，将写操作内容改动到新复制出的块设备上，来实现恢复到快照时间点的数据的目的。

Snapshot 是一个典型的基于时间以及写负载模型的后置处理机制。也就是说创建 Snapshot 时，并没有备份数据，而是把备份数据的负载均分到创建 Snapshot 之后的实际数据写发生的时间窗口，以此实现备份、恢复的快速响应。PolarDB 提供基于 Snapshot 以及 Redo log 的机制，在按时间点恢复用户数据的功能上，比传统的全量数据结合 Binlog 增量数据的恢复方式更加高效。

Parallel-Raft 算法

谈到分布式数据库的事务一致性，我们很容易想到 2PC（2 Phases Commit），3PC（3 Phases Commit）协议。而论数据状态一致性，我们就不得不提到 Leslie Lamport 发明的 Paxos 协议，在 Paxos 为 Google 等互联网厂商所广泛应用到多个分布式系统实现之后，Paxos 成为了最受关注的数据状态一致性算法之一。可是由于 Paxos 算法论文的理论和实现过于复杂，导致难以被快速应用到工程技术上。

Paxos 解决的问题之一是，在多个机器的集合中，集合中初始状态相同的任何机器能否通过相同的命令序列到达同样相同的状态点，形成一个一致的收敛的状态机。另一个问题是，作为集群中的一员，通过微观的时间串行通讯方式，需要找到一个始终有效的协议，当一个机器的某个数据状态需要改变时，需要和整个集群（包括其他机器）靠通讯和协议达成相同的认知，一起认同这个机器上的某个状态的改变。

基于这两点，基本上就解决了分布式集群架构中，不同角色的机器，达成一致性状态机的问题。也就可以进一步设计出绝大多数分布式系统的框架。

Paxos 可以堪称是 P2P（Peer to Peer）的对等设计，更加抽象和通用，也更难以理解。而 Raft 则是选举出 Leader，再经由 Leader 发起对其他角色进行状态一致性更新的实现，更容易理解。而协议本身的实现流程和 Paxos 有相似之处。

Parallel-Raft 是在 Raft 协议的基础上，针对 PolarDB chunk Server 的 I/O 模型，进行改良的一致性算法。Raft 协议基于 Log 是连续的，log#n 没有提交的话，后面的 Log 不允许提交。而 PolarDB 实现的 Parallel-Raft 允许并行的提交，打破了 Raft 的 log 是连续的假设，提高并发度，通过额外的限制来确保一致性。

Docker

容器虚拟化技术最早的出现是 Linux 内核为了解决进程在操作系统之间，或者在进程运行过程当中做迁移，通过进程和操作系统之间的解耦，来达到进程运行时的上下文和状态能够保存，复制和恢复的目的。可是 LXC 的实现，却促成了曾红极一时的 Docker 的诞生。

从原理上讲，容器虚拟化的实现相对于 KVM 等虚拟化技术而言，更加轻量级。如果用户不需要感知整个操作系统的功能，那么用容器虚拟化技术理论上应该能够获得更好的计算能效比。

其实 LXC 加上 Cgroup 这种进程虚拟和资源隔离的方式已经被使用很多年，在 HPC 领域就常被应用到 MPI 超级任务的 checkpoint 和 restart 恢复上。PolarDB 采用 Docker 环境来运行 DB 计算节点，用更轻量的虚拟化方式，解决了资源的隔离和性能的隔离，也节省了系统资源。

User-Space 文件系统

谈到文件系统，就不得不提一下 IEEE 发明的 POSIX 语义（POSIX.1 已经被 ISO 所接受），就像说到数据库要谈到 SQL 标准。通用分布式文件系统实现的最大挑战就是在完全兼容 POSIX 标准的基础上提供强劲的并发文件读写性能。

可是 POSIX 的兼容势必会牺牲一部分性能来获得对于标准的完全支持，同时系统实现的复杂度也极大的增加。说到底是通用设计和专有设计的取舍和区别，也是易用性和性能之间的平衡，这是个经典问题。

分布式文件系统是 IT 行业最经久不衰的技术，从 HPC 时代，云计算时代，互联网时代，大数据时代一直在推陈出新，其实更严格的说应该是针对不同应用 I/O 场景涌现出很多定制化的实现，再说白点，就是不去支持 POSIX 标准。

这一点上，只能说知难而退，不过只服务于专门的 I/O 场景时，不适用 POSIX 也不是什么问题。这一点，和从 SQL 到 NoSQL 的发展如出一辙。支持 POSIX 的文件系统，需要实现兼容标准的文件读写操作的系统调用接口，这样对于用户而言，就无需修改 POSIX 接口实现的文件操作应用程序。这样一来就要求通过 Linux VFS 层来铆接具体的文件系统内核实现。这也是导致文件系统工程实现难度加大的原因之一。

对于分布式文件系统而言，内核模块还必须和用户态的 Daemon 进行数据交换，以达到数据分片以及通过 Daemon 进程传送到其他机器上的目的。而 User-Space 文件系统提供用户使用的专用 API，不用完全兼容 POSIX 标准，也无需在操作系统内核进行系统调用的 1:1mapping 对接，直接在用户态实现文件系统的元数据管理和数据读写访问支持即可，实现难度大大降低，并且更加有利于分布式系统的进程间通讯。

小结：通过以上的介绍，我们不难发现，PolarDB 采用了从计算虚拟化，高速网络互联，存储块设备，分布式文件系统，数据库物理 Replication 等全方位的技术手段，可以说是众多热点技术的集大成。正式这些关键技术的整合创新，才使得 PolarDB 的性能有了质的飞跃。

阿里在数据库智能优化路上，做了哪些探索与实践?

乔红麟

阿里妹导读：近期，2017 中国应用性能管理大会（简称 APMCon 2017）圆满落幕。阿里巴巴数据库事业部高级技术专家乔红麟发表了题为《数据库智能优化系统的探索与实践》的演讲，现场解读了过去几年阿里巴巴数据库团队在面对数据库规模急速增长以及业务变化越来越快的情况下在智能数据库诊断优化方面的一些探索和实践经验。

以下为演讲实录：

乔红麟：谢谢主持人，大家下午好。今天给大家分享一下我们团队在数据库优化方面做的一些事情。阿里的数据库场景与其他公司可能会有一些不同，所以今天的分享更多是基于阿里的场景和规模所做的一些思考和实践。

先简单介绍一下我自己。我在 2015 年加入阿里，目前负责阿里数据库智能优化产品 CloudDBA 的开发。我今天的分享主要有这几方面：

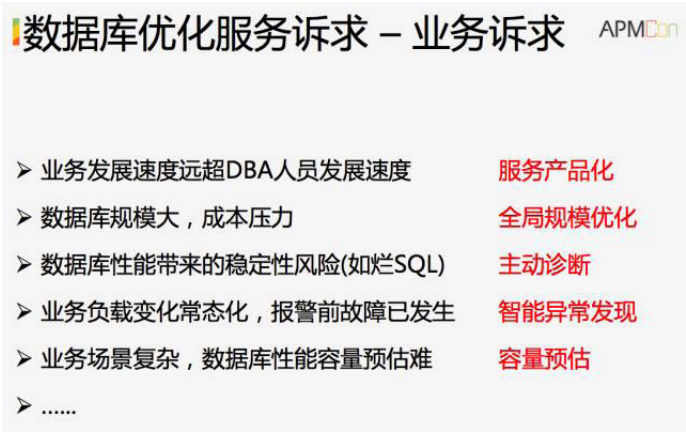
首先讲一下在阿里对数据库优化服务的诉求。我相信大家在数据库性能优化方面都有很多的经验教训，不同公司对优化的具体做法也不太一样。在方式上大部分企业应该还是重人工模式，就是由数据库能力比较强的人，比如 DBA，来解决数据库性能问题。但阿里今天的数据库规模非常大，不管我们有多少 DBA，我们的人员增长速度都无法跟上业务发展的速度，单纯依赖 DBA 已经无法满足业务发展需求。

第二方面讲一下我们的 CloudDBA 是如何做的，里面涉及到哪些技术，希望把这些技术分享给大家。如果大家所在的公司也在做类似的事情，希望能够提供一些参考和帮助。

第三方面大概讲一下目前正在探索的一些事情。现在人工智能技术比较火，数据库相对来说是比较传统的领域。如果我们将机器学习、深度学习这样的技术引入到数据库领域，它到底能做些什么，具体到数据库优化领域又能做什么，这是我们正在探索的一些事情。

一、阿里数据库优化服务诉求

第一部分、业务诉求



首先从整个阿里数据库的角度看一下对于数据库优化服务的业务诉求，这也是我们做这个产品最大的驱动力。

1、服务产品化。阿里业务发展速度远远超过了 DBA 团队发展的速度，单独依靠 DBA 重人工支持模式变得越来越困难，因此我们在几年前开始尝试通过产品来完成 DBA 人工做的一些工作。通过产品解决 DBA 人工服务的扩展性问题，是我们最直接的诉求，希望能把 DBA 人工服务产品化。

2、全局规模优化。站在全局角度来看，数据库规模迅速增大的同时也带来了巨大的成本压力。成本这块怎么理解呢？只要业务有需求，理论上可以通过增加更多的机器来满足业务需求。 但是从另外一个角度来讲，这些机器是不是一定要加，是不是有一些机器可以通过优化节省下来给新的业务服务。当规模非常大的时候，所做的一小点规模化优化，所节省的成本可能都是很可观的。因此我们需要有全局规模优化的能力，仅仅一个数据库实例内部做的优化都是一些局部优化，以全局角度来看是不够的。

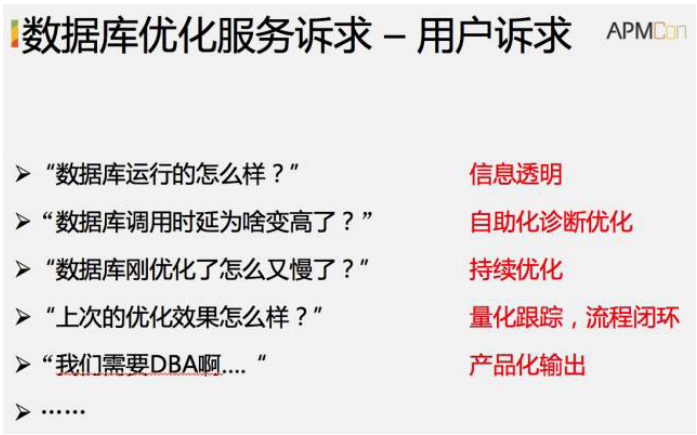
3、主动诊断。从运维的角度来看，阿里同其它公司一样，就是要尽量避免故障的发生。在阿里的业务场景下，大部分业务跟数据库有着非常紧密的关系。数

据库一个微小的抖动，都可能对业务造成非常大的影响，所以如何让数据库更稳定是非常重要的业务诉求。比如一个最常见的情况，有很多线上 SQL 性能是有问题的，这些 SQL 会给业务稳定性带来一定的风险。那么我们能不能通过产品主动对线上有问题的 SQL 进行主动诊断，提前做优化，而不是 SQL 引起故障后才去优化。

4、智能异常发现。线上业务负载不断地变化，业务行为、用户行为也在不断地变化。传统基于阈值设置报警的方式无法可靠、及时地发现数据库故障或者异常。如何可靠地去发现数据库异常，甚至是提前预测到故障的发生并进行及时干预，是有很强的业务需求的，但同时也有非常大的技术挑战，尤其是在阿里这么大数据库规模场景下。

5. 容量预估。还有一些业务诉求是容量预估的需求。比如什么时候需要扩容，如何更精准地对数据库容量做出预估，这些方面后面我会稍微展开一下。

第二部分、用户诉求



另外一部分诉求是使用数据库这些人的诉求，也就是我们的开发人员。每个公司数据库服务方式有所不同。这里我列了一些开发人员经常会问到的一些问题，这些问题背后的诉求让我们思考我们的产品站在开发者的角度，要解决什么问题。在业务发生异常的时候，需要快速定位到整个链路到底哪块出了问题。之前 DB 对于开发者来

说是一个黑盒，不管是信息透明方面，还是大家对数据库领域的知识方面，对于 DB 的了解程度可能都不够，不知道 DB 是什么状态，发生了什么问题。具体来讲用户诉求主要有：

1、信息透明，自助优化。我们期望用户能够自助发现和解决数据库的性能问题，并非发现问题先去找 DBA，这样整个流程会比较长，时间成本也比较高。但做到自助化，首先用户能够全面了解数据库的运行情况。

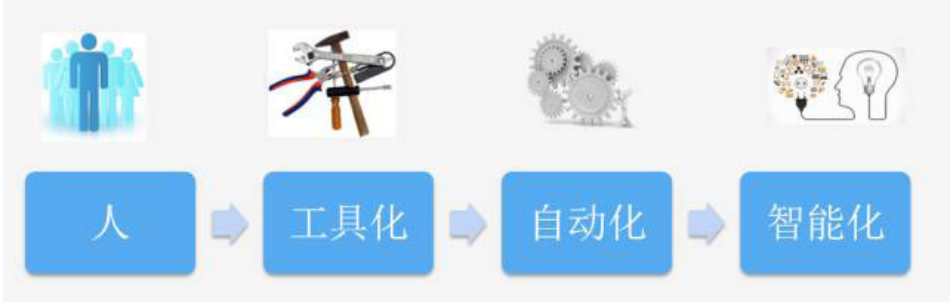
2、持续优化。只要业务在线上运行就会不断的变化，业务负载不断变化、用户行为也会不断变化。所以数据库优化是个持续的过程，并不是今天发现一个问题解决了，以后就不出现问题了。尤其是互联网的应用，持续优化尤其重要。

3、量化跟踪，流程闭环。开发人员经常会问到一个问题，上次帮他做的优化，结果到底怎么样。我们知道并不是每个优化都是实际有效的，因为很多优化方案是基于当时的信息和场景做的一个判断，实际优化结果只有当应用之后才能真正去做评估、做衡量，所以我们要提供量化跟踪和评估的能力。另外，我们期望整个优化流程，从发现问题到最终解决问题在产品内能够闭环，开发人员能够自己完全自助化走完整个流程，而不需要 DBA 的参与。流程闭环也是产品必须具备的能力。

4、输出产品，而不是人。不断有新的业务上线，而我们的 DBA 就这么多人，并且每个人有不同的侧重。对于一些快速发展的业务，在早期我们可能没有 DBA 去做特别支持的，但这些业务的数据库反而是容易出问题的。开发人员如果能够通过产品解决问题，而不是凡事都去找 DBA，解决问题的效率会更高。将我们 DBA 的能力通过产品进行输出，更好去支持我们的业务。

所以今天我们做 CloudDBA 产品 , 是希望我们能够通过产品的方式把 DBA 专家服务提供给业务开发同学，实现 DBA 人力的扩展。同时我们需要具备全局规模优化能力，这是在阿里对数据库优化服务的业务诉求和用户诉求，也是我们做 CloudDBA 这个产品的动机。

二、CloudDBA 关键技术



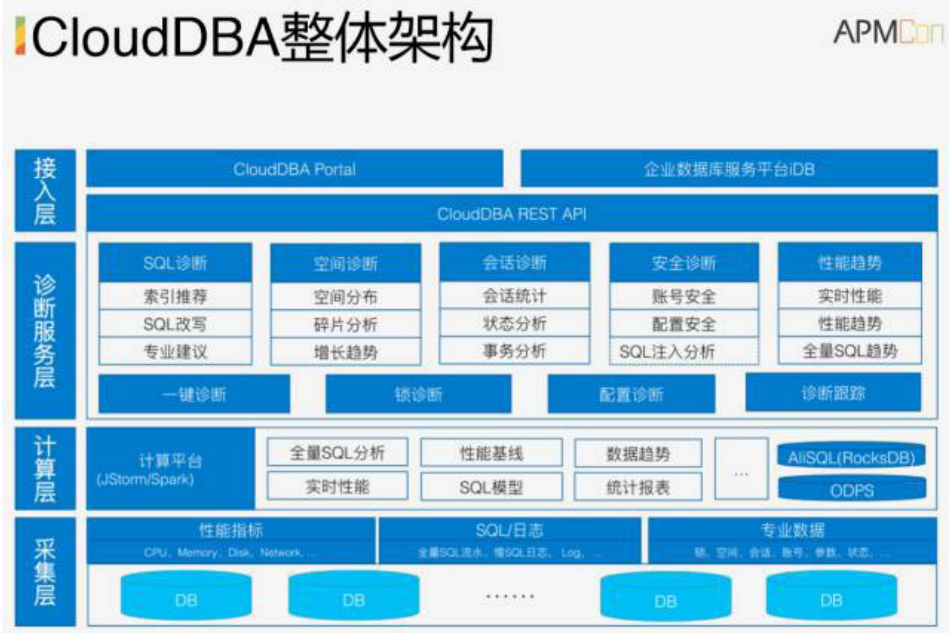
首先简单介绍下 CloudDBA 在阿里的发展历程。早期我们团队有很多非常牛的 DBA，经常是一个人当千军万马的感觉，那个阶段优化更多依赖于 DBA 人工完成。之后我们开发了一些工具，让工具来完成简单的优化操作。几年前我们的理念转变为所有数据库服务都应该由产品来做，所以我们在数据库运维，管理，优化等方面都有相应的产品，开始进入自动化阶段。

未来数据库优化服务会从自动化发展到智能化，这是我的判断。今天仍然有很多问题是解决不了的，比如精确的容量预估，智能的异常发现，故障提前预警等。现在我们有非常多的数据，也有数据加工分析的技术，所以我们开始进行一些探索，通过数据分析和机器学习等技术手段来解决之前解决不了的问题。比如最简单的容量预估，每年都会做预算，做容量预估。至少我现在还没有看到特别多的公司去用很科学的方式，完全基于业务目标以及历史数据的分析来做容量预估。很多时候容量预估是靠拍脑袋决定的，但是今天有了大量的数据和加工数据的技术手段，我们是不是可以做更精准的容量预估。举这个例子来说明一下，未来很多的优化应该向智能化方向去思考，去探索。

CloudDBA 在阿里大概是这样的一个发展历程，我们今天还处于自动化阶段，但同时也有一些智能化的实践。未来我的判断是我们一定向智能化去走，后面会在这方面尝试更多的探索。

说了这么多，那么 CloudDBA 到底是什么？ PPT 上面有一句话，“*CloudDBA 是一个数据库智能优化产品，面向开发人员提供自助化诊断优化服务，致力于成为用*

户身边的数据库专家。”CloudDBA 不是给 DBA 开发的工具，CloudDBA 从一开始我们的用户定义就很明确。我们是面向使用数据库的开发人员提供这种自助化的诊断优化服务，我们的用户不是 DBA，而是真正使用数据库的开发同学。面向 DBA 和面向开发同学对产品来讲是完全不同的概念。比如开发同学没有太多数据库背景知识，我们即使做简单的信息透明，也需要做一些翻译，能够让开发同学理解。用户定义不同，数据的加工、分析以及最终的呈现，都是完全不一样的。



接下来讲一下 CloudDBA 到底能做什么。这是我们简化版的整体架构，涉及的面比较广。从下到上分为四层：

1、最下面是我们的采集层。对所有数据库进行实时的秒级数据采集，包括性能指标，日志数据、SQL 流水，DB 内部的一些信息等等

2、采集完之后数据到达计算层，计算层分两大块。一部分是实时计算，对于 SQL 流水，监控指标等，都会做实时计算和展示。另一部分是离线分析，比如性能基线，读写热点，统计报表等。

3、再往上就是数据库诊断服务层。如果大家做过系统的数据库优化，就清楚数

据库优化会涉及到很多方面。最常见的就是 SQL 优化，SQL 是不是很慢、有没有走到最优路径、SQL 写法是否合理等等。SQL 相关问题是我们开发经常会遇到的。还有其他一些问题，比如说空间，会话，锁，安全，配置等，CloudDBA 能够对 DB 的每一个方面提供相应的专家诊断服务。

4、最上面是接入层，在阿里内部通过企业数据库服务平台 iDB 作为入口向开发同学提供数据库优化服务。

接下来跟大家分享一下我们做这个产品的一些产品设计原则。如果大家也在做类似的产品，希望能够给大家一些参考。

之前我们数据库优化主要是 DBA 来做，但 DBA 人工优化不具备扩展性，CloudDBA 第一个设计原则就是要提供自助化服务，希望整个优化过程只有开发参与，并且整个优化流程能在 CloudDBA 里实现闭环。

由于业务负载会不断地变化，需要对所有线上数据库进行持续的主动诊断，及时发现和解决数据库性能问题。

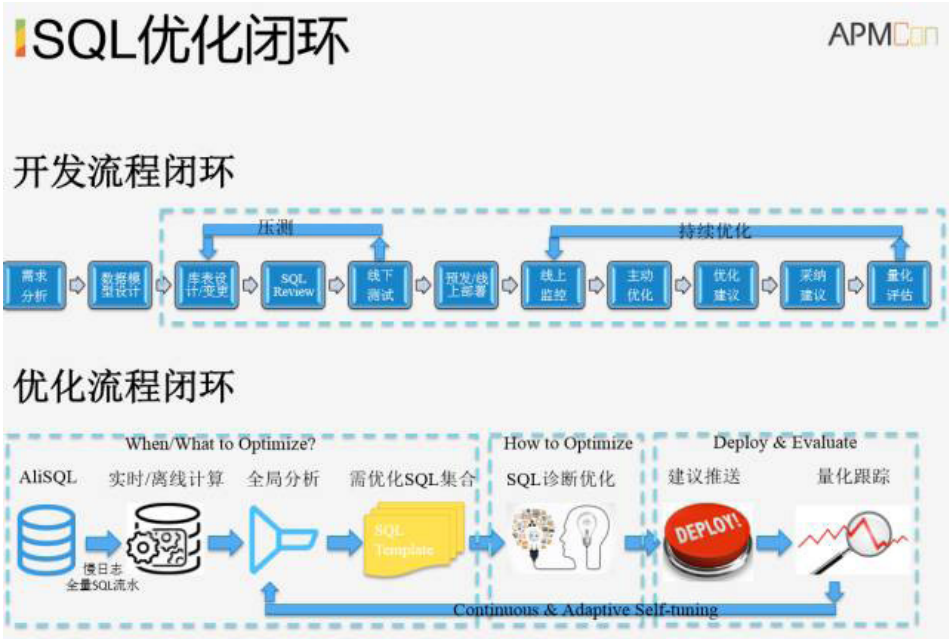
另外这个产品需要有全局的视角，能够从全局角度发现规模优化点，具备规模优化能力，并且能够量化规模优化的收益。

还有最后两点非常重要，首先就是数据驱动。从我个人理解，今天要做这样一个优化产品，首先要有足够的数据，然后用数据分析和挖掘的技术手段，再结合数据库

领域知识，给出更合理的诊断优化建议。智能化是我们对于数据库优化产品未来发展方向的判断，也是我们一直在坚持探索的。

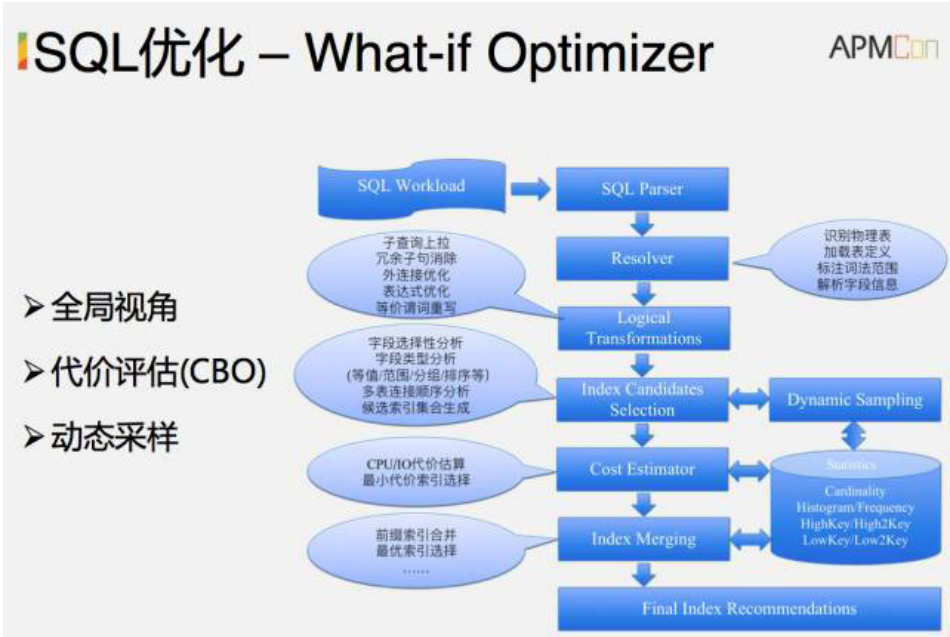
时间关系今天无法全部展开，接下来重点展开几个方面。一个是 CloudDBA 的 SQL 优化怎么做的，还有一个是空间优化，另外就是 CloudDBA 全量 SQL 采集和分析。最后会分享一下我们在智能化方向的探索。

SQL 诊断



先说一下 SQL 优化，不知道大家平时做 SQL 优化时是怎样的流程。大家回想一下，你是怎么发现哪些 SQL 需要优化的？要知道优化什么，为什么要优化它，然后再考虑怎么去优化。还有一个问题是优化完之后效果到底怎么样，是不是真的有效。整个优化过程不管是开发还是 DBA 做，都需要形成一个闭环。

CloudDBA 产品实现了这么一个闭环。第一步决定哪些 SQL 需要优化，第二步是如何优化，第三步是优化后效果如何，要做量化跟踪，确认是不是有效。如果发现没有效果，再次重复这个优化流程，直到问题被解决。



这是 CloudDBA 里 SQL 优化的大概流程。我们实现了一个类似 MySQL 优化器的 What-if optimizer。举个例子说明一下 What-if optimizer 是什么。比如一条 SQL 查询有 10 个可选的访问路径，MySQL 优化器目标是要从这 10 个路径选择访问代价最低的一个路径。而 What-ifoptimizer 要做的事情是如何规划出第 11 条路，让这条路比现有的 10 条路都快。难点在于这条路是不存在的，这个路怎么修，修完之后是不是真的更快，这些是 What-if optimizer 要解决的问题。

比如一个常见的 SQL 优化手段是索引，那建什么样的索引会比当前所有执行路径都好？这是我们的 SQL 优化引擎要解决的问题，也是我们产品比较核心的部分。大家可以看一下这个流程，前面几步跟 MySQL 或者其他优化器类似，但后面的候选索引生成，代价评估，优化建议合并等都不一样。我们的输入是一条 SQL 或者一个 SQL workload，输出是对应的优化建议，比如新建索引，SQL 改写等。

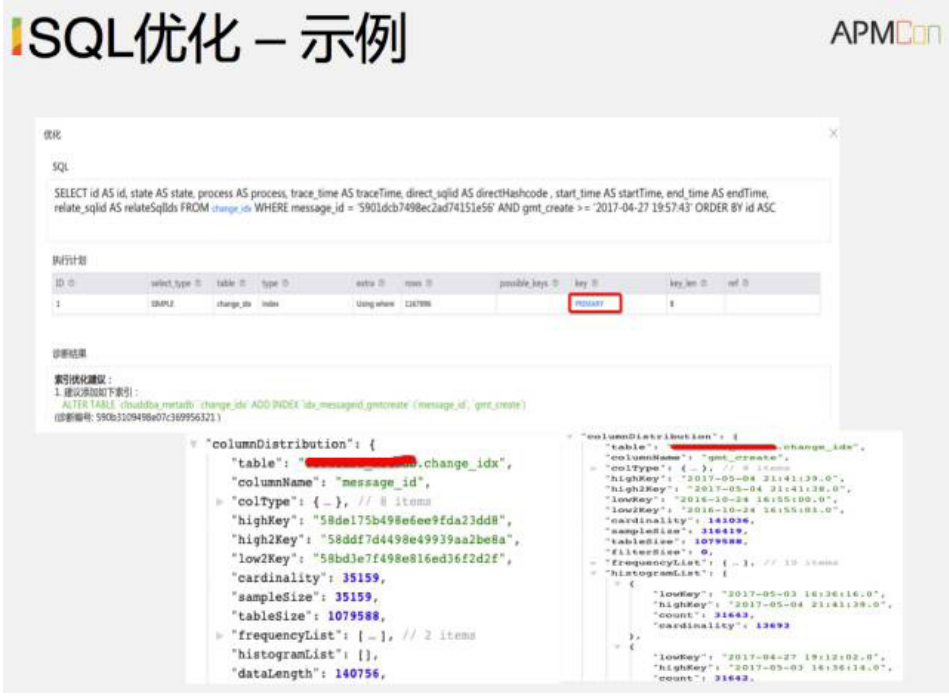
SQL 优化最关键的是要有全面准确的统计信息作为输入，另外就是它不能是规则式的，因为 SQL 的执行路径与数据分布有很大的关系。同样一条 SQL，数据分布不一样，实际执行路径可能会完全不一样。SQL 优化这块有几个关键点需要强调一下：

1、全局视角。如果业务 SQL 非常多，假设有 100 类 SQL(模板化后)，挑选哪些 SQL 来优化是非常关键的。是对这 100 类都做优化，还是选出其中一些重要的 SQL？通常会选择性能有问题的 SQL 优化，但怎么选呢？ CloudDBA 有全量 SQL 性能统计数据，会分析出查询效率低且有优化空间的 SQL Workload 来进行优化。

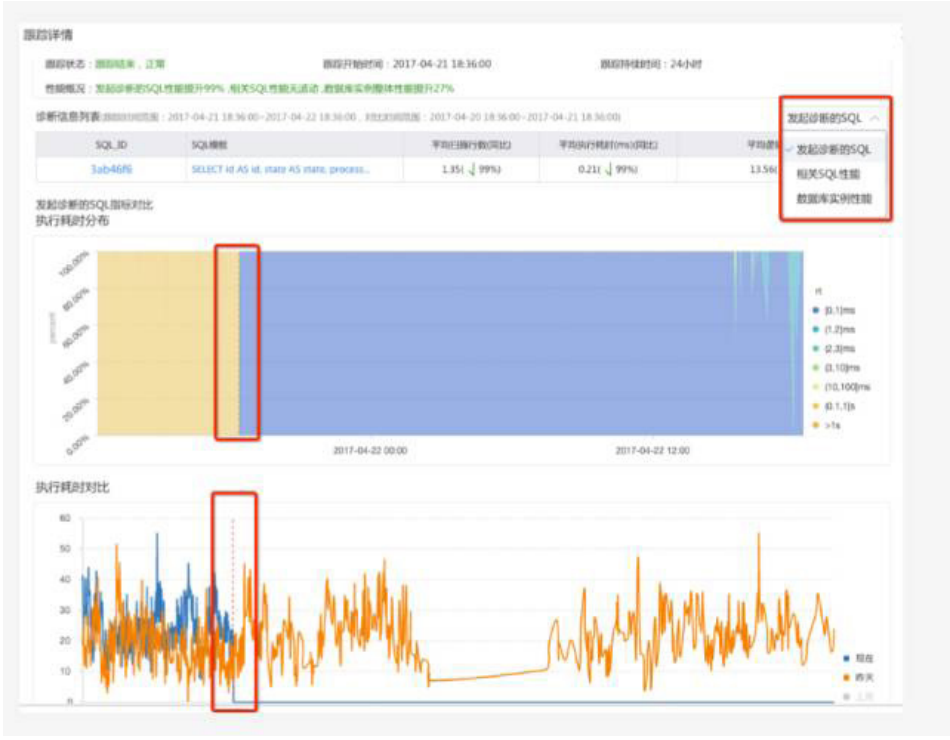
2、代价评估 (Cost-based Optimizer)。比如一条 SQL 可能会有多个索引建议，哪个建议是最优的？候选索引生成阶段确定某列是不是可以放到候选索引里，也需要结合统计信息来评估。这些过程都需要基于代价进行评估，而不是规则。

3、动态采样。优化器在做路径选择时很重要的一个输入是统计信息，对于我们的 What-if optimizer 也一样。我们通过动态采样来获取代价评估所需要的统计信息，包括 Cardinality, Frequency 还有 Histogram 等。数据倾斜比较严重的时候，Histogram 对做出准确的代价评估非常关键。为了更准确的代价评估，所以我们实现了一个动态采样系统。

量化跟踪



CloudDBA 的 SQL 诊断在阿里内部怎么使用呢？我们选择出要优化的 SQL 之后会有一个优化按钮来发起诊断流程。对单条 SQL 来讲，诊断结果包括 SQL 文本，现在的执行计划大概是什么样子，以及我们针对这条 SQL 给出的优化建议。比如这个 SQL 给出一个新建索引的建议。这个建议我们目前直接给到了开发，开发同学会可以应用这个建议，自助去创建这个索引。



那如何判断索引建议是否有效？前面提到优化流程闭环很重要的一环就是量化跟踪。这里有一个例子，是 CloudDBA 推荐的索引被开发采纳后 24 小时的性能量化跟踪情况。这两个框圈出来的时间点是索引生效的时间点，下面两个图表示优化建议被采纳前后的性能对比。会分析这个优化建议对直接优化 SQL，以及该索引可能影响到的 SQL，以及整个实例的性能影响，来量化判定和评估这次这个优化的建议是否有效。

能够发现问题，找出问题根本原因，给出问题解决方案，并且可以应用到线上，

然后完成量化评估，整个优化闭环在产品里做到闭环。只有这样，开发自助优化才能真正实现，我们才能拿到最终的优化结果，中间少任何一环都很麻烦。比如说没有量化评估手段，用户采纳建议的动力就会小很多，因为即使应用了之后，也无法知道到底是不是有效。今天 SQL 诊断在 CloudDBA 可以做到整个流程闭环。

空间优化

接下来简单说一下空间优化。不知道大家平时有没有关注过自己 DB 空间使用情况。过去大多时候数据库空间不够首先考虑就是扩容，加机器。在阿里当然也有这样的扩容需求，但今天我们首先考虑的是优化现有存储空间。比如有的表数据有十列，但索引就建了二三十个，这个是不合理的。有的开发同学根据自己的 SQL 上去就建一个索引，他并没有看有没有建相关的索引，他可能只会建一个对他有用的索引。有些索引自从建了之后从来没被访问过，也有的索引是和其他索引只是索引名字不同但完全重复的，这些都造成了空间的消耗。因此 CloudDBA 在空间优化方面做了非常全面、深入的分析，也花了很大力气去做，因为对阿里今天的数据库规模来讲空间成本非常高。

1. 存储优化。通过数据存储方式的优化来节省存储空间。比如分析出来有些数据表字段占用空间较大且可以做压缩时，我们会建议用户做压缩。对于数据量特别大，但访问量又不是特别高的实例，我们会建议数据库做存储引擎的迁移，从 InnoDB 切换到 RocksDB, 以获取更高的压缩比来节省空间。

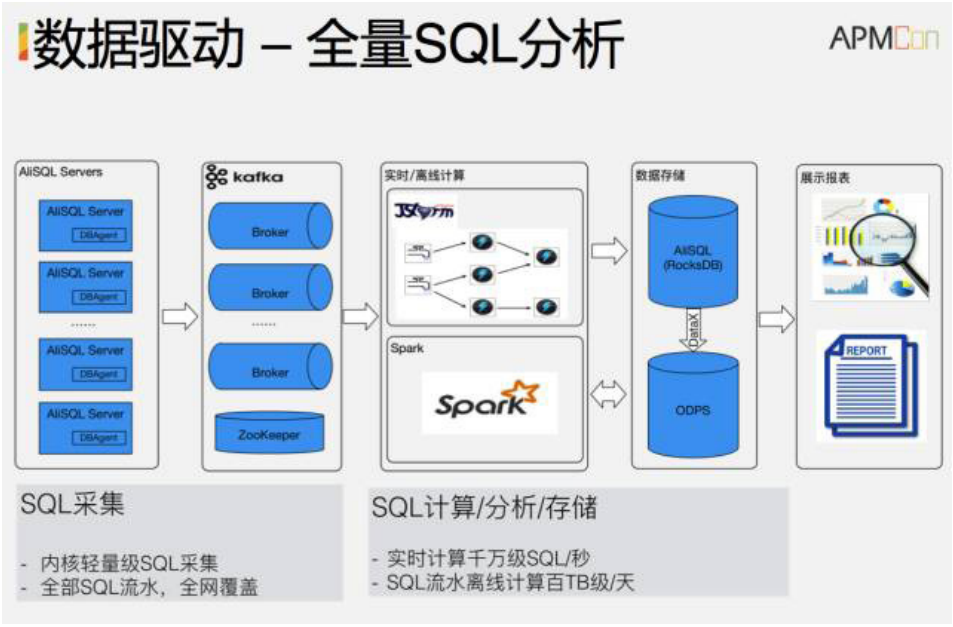
2. 空间回收。通过回收无用数据对象占用空间来优化存储空间，包括对表碎片进行分析和回收，重复索引 / 无用索引删除，无流量表删除等。业务不断变化，业务相关 SQL 也会变化，有些表或者索引在业务变化后可能就再没有人访问了，但还占了非常昂贵的在线存储空间，这些空间都是可以回收的。

3. 数据迁移。把数据库里存放时间过久且不被访问的数据迁移到更低成本的离线存储上。CloudDBA 会分析一张表的数据存储时间分布情况，比如有百分之多少的数据是 3 个月以内，多少数据是 3 个月到 6 个月，多少数据是 6 个月到 1 年等等。开发同学根据自己的业务需求结合数据生命周期来判断哪些数据可以做迁移或者清理。

CloudDBA 会对线上所有数据库空间进行主动诊断，并且提供一键优化功能，开发同学可以自助化完成数据库的空间优化。

数据驱动

前面提到 CloudDBA 产品的设计原则，有一个很重要的核心理念就是数据驱动。数据驱动的前提先得有数据。阿里的数据库规模下数据采集处理还是很有挑战的，因为规模太大了。今天我们所有数据都是秒级采集，分析和展现。过去几年我们花了很多精力建设了一个可靠的数据通道，对采集上来的数据进行实时分析和离线分析。因为我们坚信未来 CloudDBA 产品一定向数据化、智能化的方向走，虽然数据通道和计算花了很多的精力，但是产生的数据价值很高。



数据驱动这块今天重点说一下全量 SQL 分析。如果要对数据库性能进行诊断，单纯看慢 SQL 是不够的。有些业务对 latency 很敏感，尽管 SQL 没有达到慢 SQL 标准，比如 MySQL 里默认的 1 秒，但业务已经感觉到明显的异常，因此需要对数据库实例上的全部 SQL 进行分析，了解当前实例正在执行哪些 SQL，性能如何。

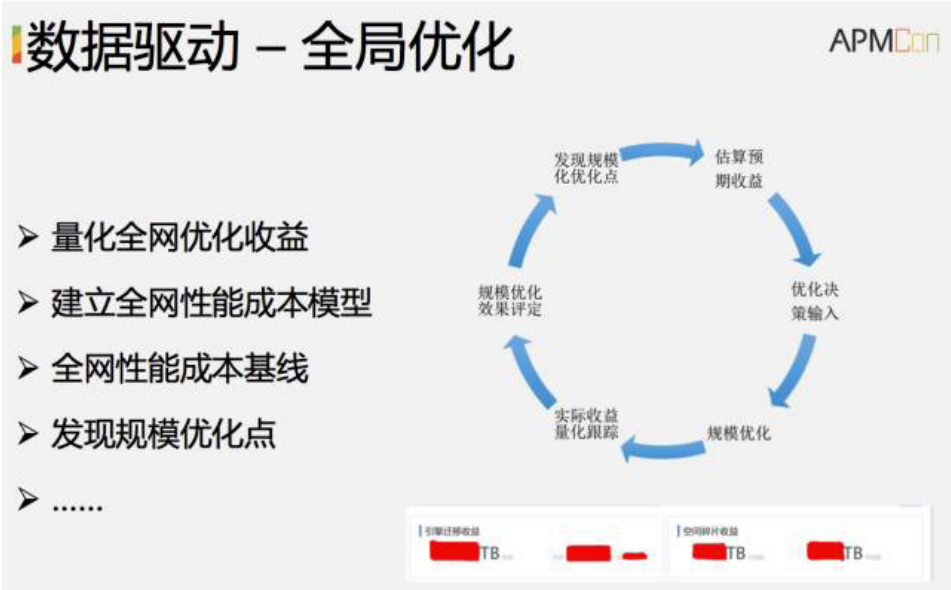
CloudDBA 对全网数据库实例上全部 SQL 进行实时采集、分析和展现，并且基于这些数据来驱动整个诊断优化流程。

先看一下全量 SQL 分析的数据通道。我们的 AliSQL 内核实现了非常轻量级 SQL 流水采集，包括 SQL 来源，SQL 文本，执行时间，锁等待时间，扫描行数 / 返回行数、逻辑 / 物理读等信息。这些信息会实时地上报到 Kafka, 然后利用 JStorm 对全量 SQL 进行各种维度的实时计算和分析。另外，对实时计算完的数据我们还会进行很多离线分析。目前全量 SQL 流水采集在阿里基本上全网覆盖，实时计算数据量达到千万级 SQL/ 秒，离线计算数据量达到百 TB/ 天。

这里列出了一些我们基于全量 SQL 分析进一步做的一些事情。首先用户通过 CloudDBA 可以实时查看当前实例正在执行的全部 SQL 信息以及性能趋势，这些信息能够更全面了解业务 SQL 的健康情况，比如对执行耗时占比较高但执行效率较低，执行频繁但索引缺失，执行性能有明显波动以及新增 SQL 等进行了识别。对于有分库分表的业务，我们能够识别是否有读写热点。对线上的优化操作可以更好地进行性能优化度量。还有从安全的角度，可以进行全面的 SQL 审计，还有我们也进行一些

实际业务模型的分析。

再简单说一下基于数据驱动在全局优化方面做的一些事情。



前面讲业务诉求提到了需要有全局规模优化的能力，全局优化方面我们也构造了一个闭环。基于海量数据分析，从“发现规模优化点”，“估算预期收益”，到提供全局“优化决策输入”，开始“规模优化”，然后进行“实际收益量化跟踪”，以及最后的“规模优化效果评定”，整个流程在 CloudDBA 内部形成闭环。比如我们的全网空间优化，基于上就是通过这个闭环流程完成，同时 CloudDBA 会给出空间规模优化的实际收益。

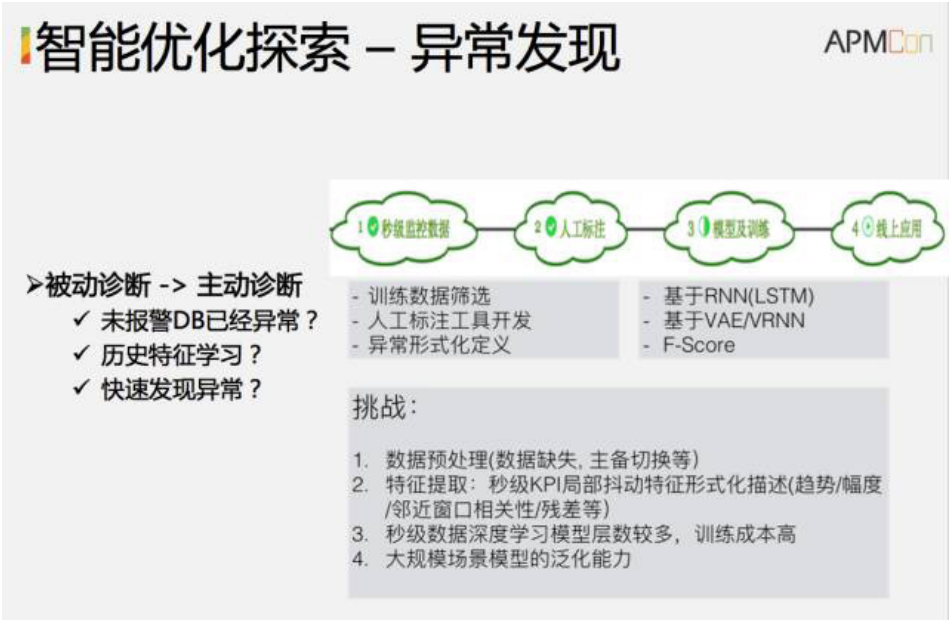
全局优化方面，我们期望能够真正做到数据驱动，包括建立全网性能成本模型，建立性能成本基线等，能够基于数据分析发现收益较大的规模优化点。这方面我们还在持续地做更多的事情。

三、数据库智能优化探索

最后分享一些我们在智能优化方向的一些探索。利用数据分析和机器学习的一些技术，我们尝试去解决数据库领域之前较难解决的一些问题。这块有很大的想像空间，目前我们主要是在以下几个方面进行探索：

1. 异常检测 / 关联分析。上午我听了一个分享讲在其他领域的异常发现 / 关联分

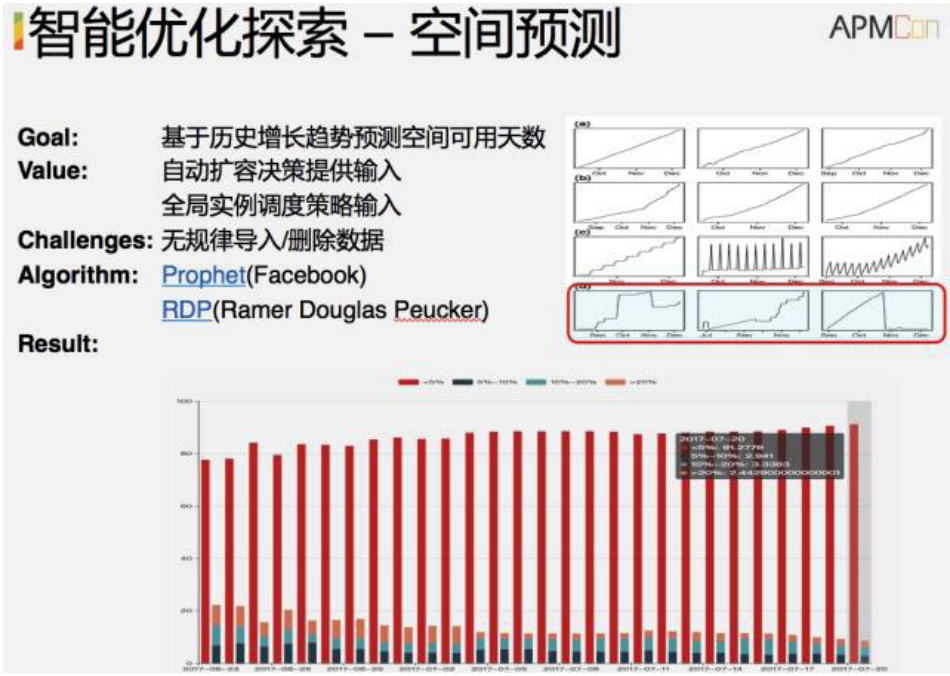
析，异常检测不仅是数据库领域，在其他运维领域也有需求，是一个共性的问题。运维同学做监控，异常现在怎么发现？通常的做法是依赖报警，但报警的阈值怎么定？定高了到时候故障发生了报警还没出来，定低了收到一堆报警，都是误报。能不能做到智能的异常发现？比如说某个数据库实例跟之前的历史行为有比较大的不同，但并没有触发报警，我如何能第一时间知道？因为发现的越晚，线上的损失就会越大，在阿里的场景这个是有直接的业务诉求。及时发现异常，快速定位原因，减少业务影响。



昨天清华的裴老师也讲了，异常发现对于分钟级的数据，目前一些现有算法是适用的，稍微做一些加工就会有比较明显的效果。但是我们的数据都是秒级的，对我们来说分钟级太长了。一个问题如果在分钟级别发现，损失可能已经非常大了。基于秒级数据的异常发现，今天业界的大部分算法效果都非常差，因为秒级数据抖动非常频繁，怎么在这种抖动中区分出来是真正的异常还是正常的抖动，这个挑战比较大。这里我也列举了一些其他方面的挑战，目前对于这些问题我们有一些初步的突破，有时间再分享这块的细节。

2. 主动预警。异常发现都是事后的，因为异常已经发生了，损失也已经造成了，快速的异常发现是尽量减少损失。但我们希望做到更进一步，从被动诊断发展为主动诊断。在数据库故障真正发生之前，根据历史行为特征来对即将发生的异常进行提前预警。因为很多数据库故障发生前已经有一些特征是异常的了。这个之前基于传统阈值报警的方式是没有办法发现的。如果可以提前做预警，就有机会能更早的介入，避免故障的发生。

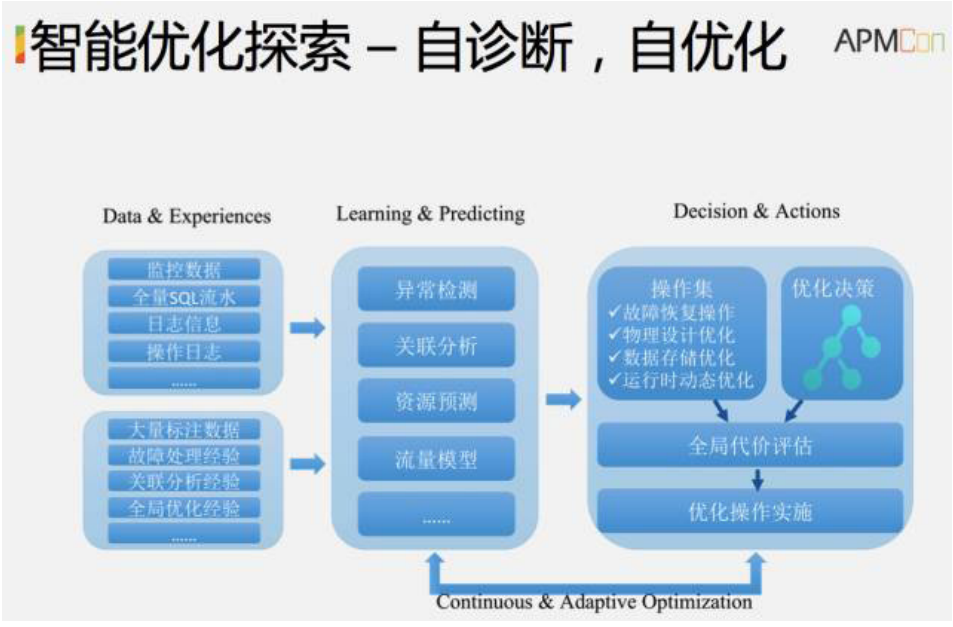
3. 容量预估。我们每年都有双 11，双 11 之前都要对数据库系统做容量预估。如果拍脑袋加 1 万台机器，但可能根本用不了这么多，那就造成大量的浪费。怎么能够更靠谱的做出容量预估，给出更合理的预算是很重要的。另外一方面，数据库实例什么时候要进行扩容，之前比较难预测的事情。我们最近在这方面做了一些尝试，举一个数据库空间增长预测的例子。



我们基于空间增长历史数据分析来对线上实例空间增长趋势进行预测，预测的结果作为实例自动扩容的输入。数据库实例什么时候需要扩容，CloudDBA 会把这个

信息透明出去，CloudDBA 也可以基于这个预测对实例进行自动扩容。根据我们目前的预测结果，对线上 >90% 的实例空间增长预测误差都可以做到 <5%，我们还在不断地优化算法把这个数字做到更好。

4. 自诊断，自优化。这是未来 CloudDBA 智能化阶段要具备的能力。整个系统可以基于大量的基础数据，通过标注把人的经验注入然后利用一些模型或者算法进行训练，分析出目前的性能问题，自动从优化操作集选择最合适的优化方案，在全局代价评估后自动应用到线上。整个过程是自动的，而且可以根据线上优化结果反馈反复进行优化。这块的探索我们还刚刚开始。



最后简单总结一下今天分享的内容。首先分享了在阿里我们为什么要做 CloudDBA，背后的业务诉求和用户诉求是什么。其次分享了 CloudDBA 的一些关键技术，时间原因对 SQL 优化、空间优化以及全量 SQL 分析进行了展开。最后跟大家简单分享了一些我们在智能优化方向的一些思考和实际探索。智能化方向是 CloudDBA 的未来，这块我们还在不断地尝试，也期望有兴趣的同学能够加入我们一起去探索。

如何降低 90%Java 垃圾回收时间? 以阿里 HBase 的 GC 优化实践为例

那珂

阿里妹导读：GC 一直是 Java 应用中讨论的一个热门话题，尤其在像 HBase 这样的大型在线存储系统中，大堆下 (百 GB) 的 GC 停顿延迟产生的在线实时影响，成为内核和应用开发者的一大痛点。

过去的一年里，我们准备在 Ali-HBase 上突破这个被普遍认知的痛点，为此进行了深度分析及全面创新的工作，获得了一些比较好的效果。以蚂蚁风控场景为例，HBase 的线上 young GC 时间从 120ms 减少到 15ms，结合阿里巴巴 JDK 团队提供的利器——AliGC，进一步在实验室压测环境做到了 5ms。本文主要介绍我们过去在这方面的一些工作和技术思想。

背景

JVM 的 GC 机制对开发者屏蔽了内存管理的细节，提高了开发效率。说起 GC，很多人的第一反应可能是 JVM 长时间停顿或者 FGC 导致进程卡死不可服务的情况。但就 HBase 这样的大数据存储服务而言，JVM 带来的 GC 挑战相当复杂和艰难。原因有三 :

1. 内存规模巨大。线上 HBase 进程多数为 96G 大堆，今年新机型已经上线部分 160G 以上的堆配置

2. 对象状态复杂。HBase 服务器内部会维护大量的读写 cache，达到数十 GB 的规模。HBase 以表格的形式提供有序的服务数据，数据以一定的结构组织起来，这些数据结构产生了过亿级别的对象和引用

3. young GC 频率高。访问压力越大，young 区的内存消耗越快，部分繁忙的集群可以达到每秒 1~2 次 youngGC，大的 young 区可以减少 GC 频率，但是会带来更大的 young GC 停顿，损害业务的实时性需求。

思路

1. HBase 作为一个存储系统，使用了大量的内存作为写 buffer 和读 cache，比如 96G 的大堆（4G young + 92G old）下，写 buffer+ 读 cache 会占用 70% 以上的内存（约 70G），本身堆内的内存水位会控制在 85%，而剩余的占用内存就只有在 10G 以内了。所以，如果我们能在应用层面**自管理**好这 70G+ 的**内存**，那么对于 JVM 而言，百 G 大堆的 GC 压力就会等价于 10G 小堆的 GC 压力，并且未来面对更大的堆也不会恶化膨胀。 在这个解决思路下，我们线上的 young GC 时间获得了从 120ms 到 15ms 的优化效果。

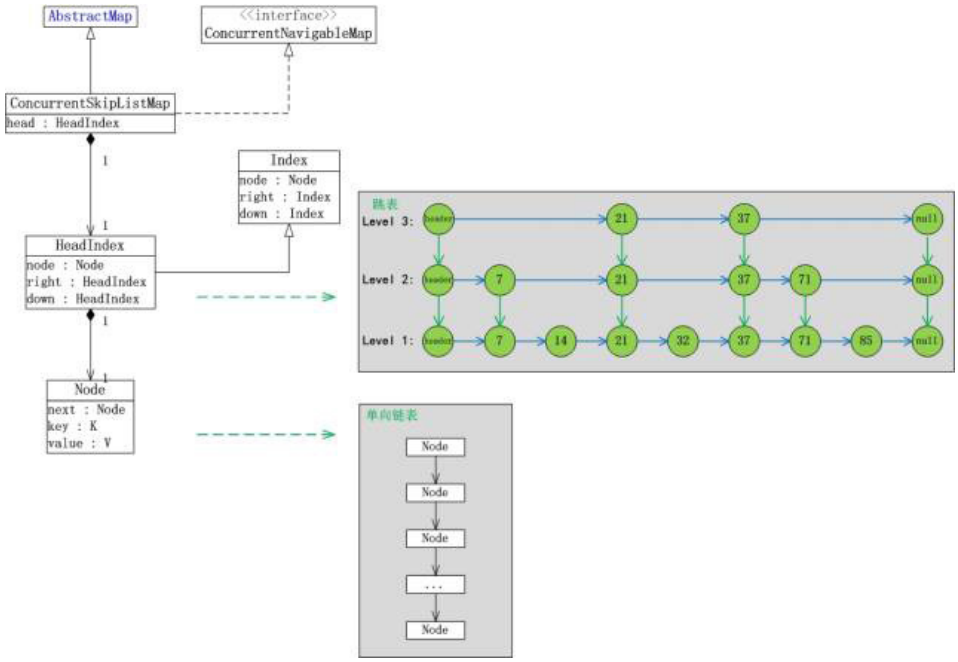
2. 在一个高吞吐的数据密集型服务系统中，大量的临时对象被频繁创建与回收，如何能够针对性管理这些临时对象的分配与回收，AliJDK 团队研发了一种新的基于租户的 GC 算法—AliGC。集团 HBase 基于这个新的 AliGC 算法进行改造，我们在实验室中压测的 young GC 时间从 15ms 减少到 5ms，这是一个未曾期望的极致效果。

下面将逐一介绍 Ali-HBase 版本 GC 优化所使用的关键技术。

消灭一亿个对象：更快更省的 CCSMap

目前 HBase 使用的存储模型是 LSMTree 模型，写入的数据会在内存中暂存到一定规模后再 dump 到磁盘上形成文件。

下面我们将其简称为写缓存。写缓存是可查询的，这就要求数据在内存中有序。为了提高并发读写效率，并达成数据有序且支持 seek&scan 的基本要求，SkipList 是使用得比较广泛的数据结构。



我们以 JDK 自带的 ConcurrentSkipListMap 为例子进行分析，它有下面三个问题：

1. 内部对象繁多。每存储一个元素，平均需要 4 个对象 (index+node+key+value，平均层高为 1)

2. 新插入的对象在 young 区，老对象在 old 区。当不断插入元素时，内部的引用关系会频繁发生变化，无论是 ParNew 算法的 CardTable 标记，还是 G1 算法的 RSet 标记，都有可能触发 old 区扫描。

3. 业务写入的 KeyValue 元素并不是规整长度的，当它晋升到 old 区时，可能产生大量的内存碎片。

问题 1 使得 young 区 GC 的对象扫描成本很高，young GC 时晋升对象更多。问题 2 使得 young GC 时需要扫描的 old 区域会扩大。问题 3 使得内存碎片化导致的 FGC 概率升高。当写入的元素较小时，问题会变得更加严重。我们曾对线上的 RegionServer 进程进行统计，活跃 Objects 有 1 亿 2 千万之多！

分析完当前 young GC 的最大敌人后，一个大胆的想法就产生了，既然写缓存

的分配，访问，销毁，回收都是由我们来管理的，如果让 JVM“看不到”写缓存，我们自己来管理写缓存的生命周期，GC 问题自然也就迎刃而解了。

说起让 JVM“看不到”，可能很多人想到的是 off-heap 的解决方案，但是这对写缓存来说没那么简单，因为即使把 KeyValue 放到 offheap，也无法避免问题 1 和问题 2。而 1 和 2 也是 young GC 的最大困扰。

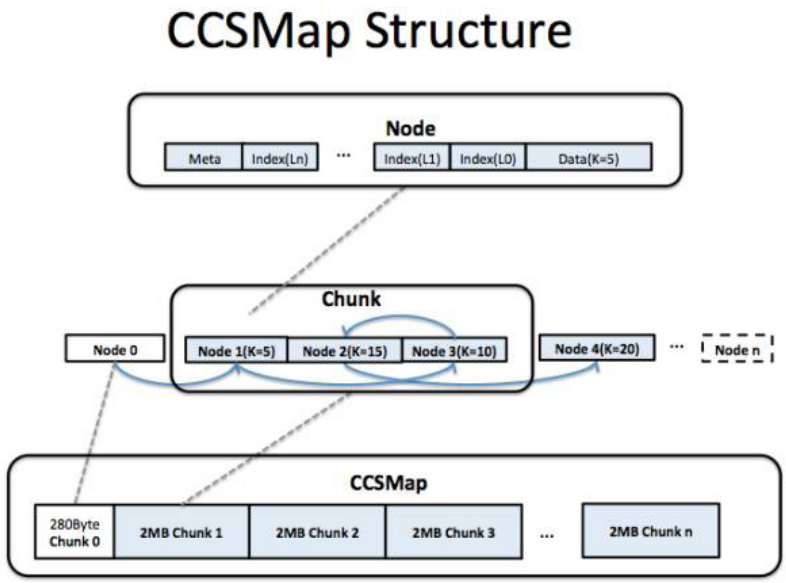
问题现在被转化成了：**如何不使用 JVM 对象来构建一个有序的支持并发访问的 Map**。

当然我们也不能接受性能损失，因为写入 Map 的速度和 HBase 的写吞吐息息相关。

需求再次强化：**如何不使用对象来构建一个有序的支持并发访问的 Map，且不能有性能损失**。

为了达成这个目标，我们设计了这样一个数据结构：

- 它使用连续的内存（堆内 or 堆外），我们通过代码控制内部结构而不是依赖于 JVM 的对象机制
- 在逻辑上也是一个 SkipList，支持无锁的并发写入和查询
- 控制指针和数据都存放在连续内存中

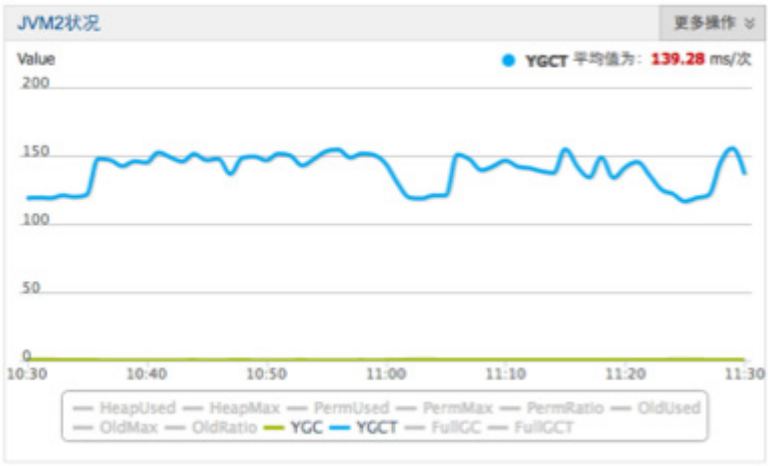


上图所展示的即是 CCSMap(CompactedConcurrentSkipListMap) 的内存结构。 我们以大块的内存段 (Chunk) 的方式申请写缓存内存。每个 Chunk 包含多个 Node，每个 Node 对应一个元素。新插入的元素永远放在已使用内存的末尾。Node 内部复杂的结构，存放了 Index/Next/Key/Value 等维护信息和数据。新插入的元素需要拷贝到 Node 结构中。当 HBase 发生写缓存 dump 时，整个 CCSMap 的所有 Chunk 都会被回收。当元素被删除时，我们只是逻辑上把元素从链表里 " 踢走 "，不会把元素实际从内存中收回 (当然做实际回收也是有方法，就 HBase 而言没有那个必要)。

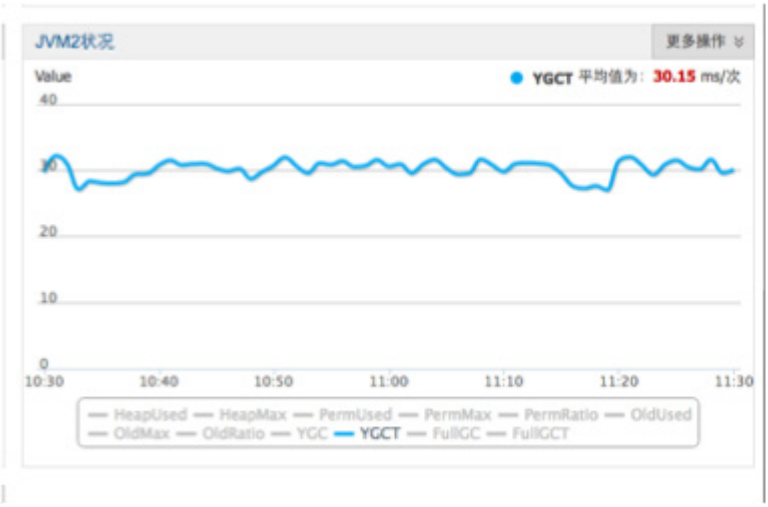
插入 **KeyValue** 数据时虽然多了一遍拷贝，但是就绝大多数情况而言，拷贝反而会更快。因为从 CCSMap 的结构来看，一个 Map 中的元素的控制节点和 KeyValue 在内存上是邻近的，利用 CPU 缓存的效率更高，seek 会更快。对于 SkipList 来说，写速度其实是 bound 在 seek 速度上的，实际拷贝产生的 overhead 远不如 seek 的开销。根据我们的测试，CCSMap 和 JDK 自带的 ConcurrentSkipListMap 相比，50Byte 长度 KV 的测试中，读写吞吐提升了 20~30%。

由于没有了 JVM 对象，每个 JVM 对象至少占用 16Byte 空间也可以被节省掉 (8byte 为标记预留，8byte 为类型指针)。还是以 50Byte 长度 KeyValue 为例，CCSMap 和 JDK 自带的 ConcurrentSkipListMap 相比，内存占用减少了 40%。

CCSMap 在生产中上线后，实际优化效果：young GC 从 120ms+ 减少到了 30ms



优化前



优化后

使用了 CCSMap 后，原来的 1 亿 2 千万个存活对象被缩减到了千万级别以内，大大减轻了 GC 压力。由于紧致的内存排布，**写入吞吐能力也得到了 30% 的提升**。

永不晋升的 Cache：BucketCache

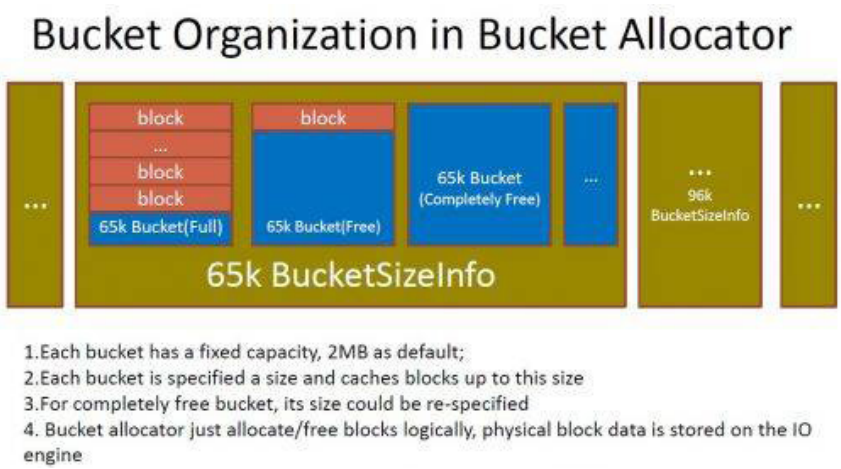
HBase 以 Block 的方式组织磁盘上的数据。一个典型的 HBase Block 大小在 16K~64K 之间。HBase 内部会维护 BlockCache 来减少磁盘的 I/O。BlockCache 和写缓存一样，不符合 GC 算法理论里的分代假说，天生就是对 GC 算法不友好的——既不稍纵即逝，也不永久存活。

一段 Block 数据从磁盘被 load 到 JVM 内存中，生命周期从分钟到月不等，绝大部分 Block 都会进入 old 区，只有 Major GC 时才会让它被 JVM 回收。它的麻烦主要体现在：

1. HBase Block 的大小不是固定的，且相对较大，内存容易碎片化

2. 在 ParNew 算法上，晋升麻烦。**麻烦不是体现在拷贝代价上，而是因为尺寸较大，寻找合适的空间存放 HBase Block 的代价较高**。

读缓存优化的思路则是，向 JVM 申请一块永不归还的内存作为 BlockCache，我们自己对内存进行固定大小的分段，当 Block 加载到内存中时，我们将 Block 拷贝到分好段的区间内，并标记为已使用。当这个 Block 不被需要时，我们会标记该区间为可用，可以重新存放新的 Block，这就是 BucketCache。关于 BucketCache 中的内存空间分配与回收（这一块的设计与研发在多年前已完成），详细可以参考：http://zjushch.iteye.com/blog/1751387



很多基于堆外内存的 RPC 框架，也会自己管理堆外内存的分配和回收，一般通过显式释放的方式进行内存回收。但是对 HBase 来说，却有一些困难。我们将 Block 对象视为需要自管理的内存片段。Block 可能被多个任务引用，要解决 Block 的回收问题，最简单的方式是将 Block 对每个任务 copy 到栈上 (copy 的 block 一般不会晋升到 old 区)，转交给 JVM 管理就可以。

实际上，我们之前一直使用的是这种方法，实现简单，JVM 背书，安全可靠。但这是有损耗的内存管理方式，为了解决 GC 问题，引入了每次请求的拷贝代价。由于拷贝到栈上需要支付额外的 cpu 拷贝成本和 young 区内存分配成本，在 cpu 和总线越来越珍贵的今天，这个代价显得高昂。

于是我们转而考虑使用引用计数的方式管理内存，HBase 上遇到的主要难点是 :

1. HBase 内部会有多个任务引用同一个 Block

2. 同一个任务内可能有多个变量引用同一个 Block。引用者可能是栈上临时变量，也可能是堆上对象域。

3. Block 上的处理逻辑相对复杂，Block 会在多个函数和对象之间以参数、返回值、域赋值的方式传递。

4. Block 可能是受我们管理的，也可能是不受我们管理的 (某些 Block 需要手动释放，某些不需要)。

5. Block 可能被转换为 Block 的子类型。

这几点综合起来，对如何写出正确的代码是一个挑战。但在 C++ 上，使用智能指针来管理对象生命周期是很自然的事情，为什么到了 Java 里会有困难呢?

Java 中变量的赋值，在用户代码的层面上，只会产生引用赋值的行为，而 C++ 中的变量赋值可以利用对象的构造器和析构器来干很多事情，智能指针即基于此实现 (当然 C++ 的构造器和析构器使用不当也会引发很多问题，各有优劣，这里不讨论)

于是我们参考了 C++ 的智能指针，设计了一个 Block 引用管理和回收的框架 ShrableHolder 来抹平 coding 中各种 if else 的困难。它有以下的范式 :

1. ShrableHolder 可以管理有引用计数的对象，也可以管理非引用计数的对象

2. ShrableHolder 在被重新赋值时，释放之前的对象。如果是受管理的对象，

引用计数减 1，如果不是，则无变化。

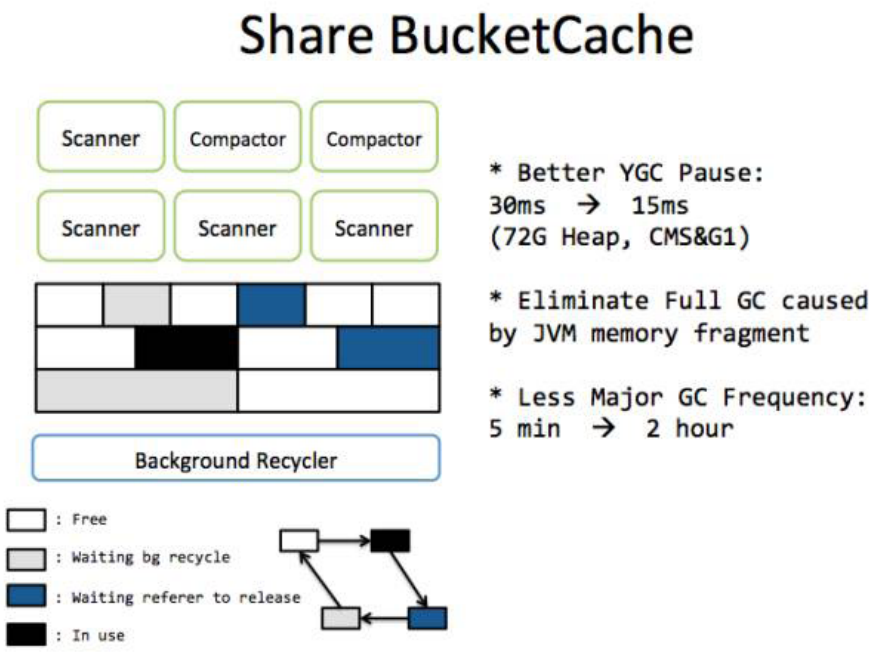
3. ShrableHolder 在任务结束或者代码段结束时，必须被调用 reset

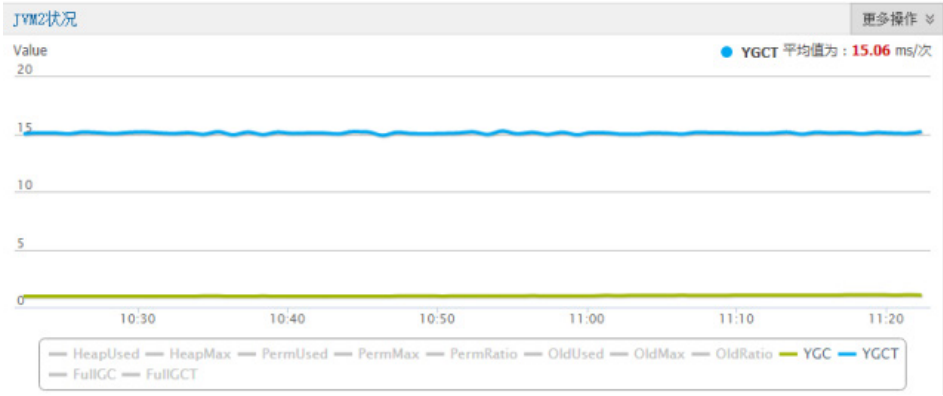
4. ShrableHolder 不可直接赋值。必须调用 ShrableHolder 提供的方法进行内容的传递

5. 因为 ShrableHolder 不可直接赋值，需要传递包含生命周期语义的 Block 到函数中时，ShrableHolder 不能作为函数的参数。

根据这个范式写出来的代码，原来的代码逻辑改动很少，不会引入 if else。虽然看上去仍然有一些复杂度，所幸的是，受此影响的区间还是局限于非常局部的下层，对 HBase 而言还是可以接受的。为了保险起见，避免内存泄漏，我们在这套框架里加入了探测机制，探测长时间不活动的引用，发现之后会强制标记为删除。

将 BucketCache 应用之后，减少了 BlockCache 的晋升开销，减少了 young GC 时间：





（CCSMap+BucketCache 优化后的效果）

追求极致：AliGC

经过以上两个大的优化之后，蚂蚁风控生产环境的 young GC 时间已经缩减到 15ms。由于 ParNew+CMS 算法在这个尺度上再做优化已经很困难了，我们转而投向 AliGC 的怀抱。AliGC 在 G1 算法的基础上做了深度改进，内存自管理的大堆 HBase 和 AliGC 产生了很好的化学反应。

AliGC 是阿里巴巴 JVM 团队基于 G1 算法，面向大堆（LargeHeap）应用场景，优化的 GC 算法的统称。这里主要介绍下多租户 GC。

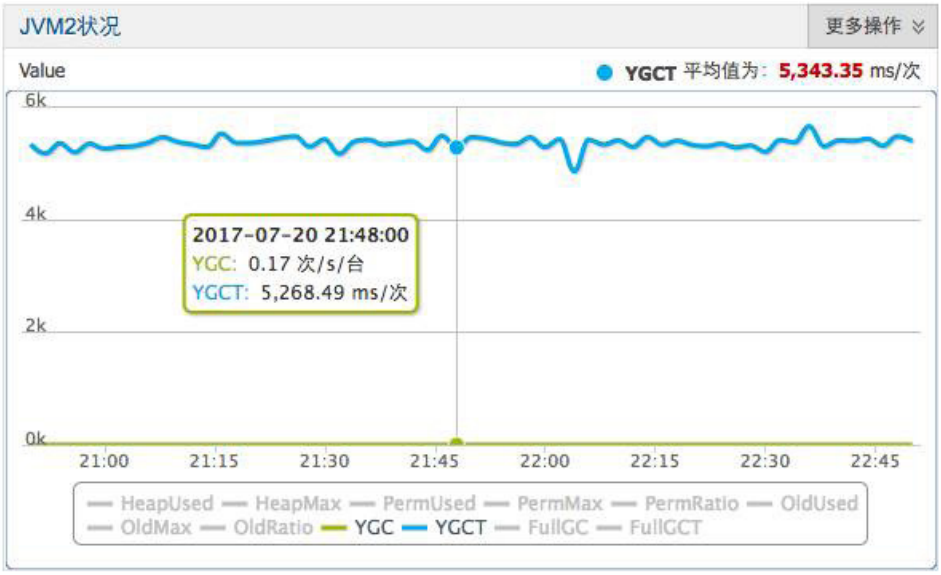
多租户 GC 包含的三层核心逻辑：1）在 JavaHeap 上，对象的分配按照租户隔离，不同的租户使用不同的 Heap 区域；2）允许 GC 以更小的代价发生在租户粒度，而不仅仅是应用的全局；3）允许上层应用根据业务需求对租户灵活映射。

AliGC 将内存 Region 划分为了多个租户，每个租户内独立触发 GC。在个基础上，我们将内存分为普通租户和中等生命周期租户。中等生命周期对象指的是，既不稍纵即逝，也不永久存在的对象。由于经过以上两个大幅优化，现在堆中等生命周期对象数量和内存占用已经很少了。但是中等生命周期对象在生成时会被 old 区对象引用，每次 young GC 都需要扫描 RSet，现在仍然是 young GC 的耗时大头。

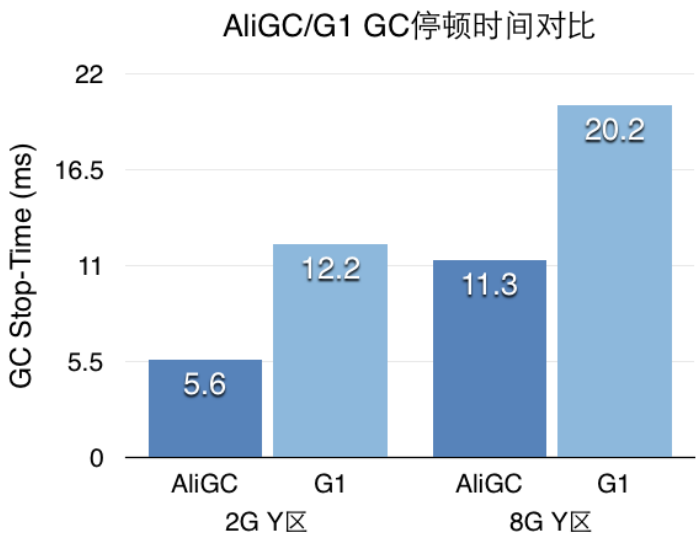
借助于 AJDK 团队的 ObjectTrace 功能，我们找出中等生命周期对象中最 " 大头 " 的部分，将这些对象在生成时直接分配到中等生命周期租户的 old 区，避免

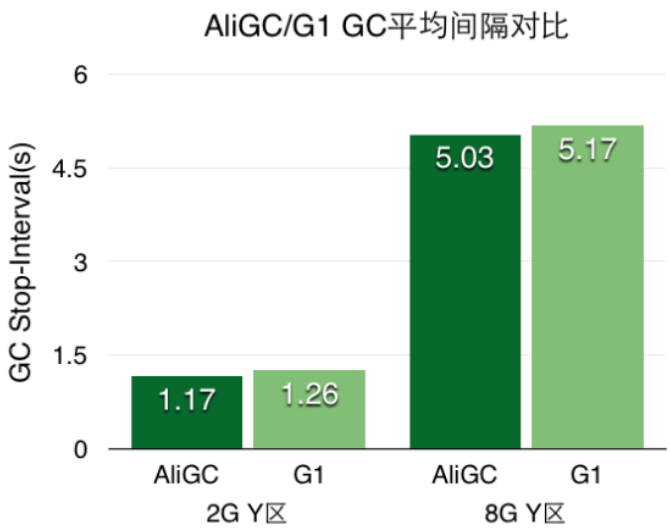
RSet 标记。而普通租户则以正常的方式进行内存分配。

普通租户 GC 频率很高，但是由于晋升的对象少，跨代引用少，Young 区的 GC 时间得到了很好的控制。在实验室场景仿真环境中，我们将 young GC 优化到了 5ms。



（AliGC 优化后的效果，单位问题，此处为 us）





云端使用

阿里 HBase 目前已经在阿里云提供商业化服务，任何有需求的用户都可以在阿里云端使用深入改进的、一站式的 HBase 服务。云 HBase 版本与自建 HBase 相比在运维、可靠性、性能、稳定性、安全、成本等方面均有很多的改进，更多内容欢迎大家关注 https://www.aliyun.com/product/hbase

写在最后

如果你对大数据存储、分布式数据库、HBase 等感兴趣，欢迎加入我们，一起做最好的大数据在线存储，联系方式：tianwu.sch@alibaba-inc.com；也欢迎一起交流问题，一起学习新技术。

如何打造千万级 Feed 流系统？阿里数据库技术解读

德歌 & 窦贤明

2017 年的双十一又一次刷新了记录，交易创建峰值 32.5 万笔 / 秒、支付峰值 25.6 万笔 / 秒。而这样的交易和支付等记录，都会形成实时订单 Feed 数据流，汇入数据运营平台的主动服务系统中去。数据运营平台的主动服务，根据这些合并后的数据，实时的进行分析，进行实时的舆情展示，实时的找出需要主动服务的对象等，实现一个智能化的服务运营平台。

通过 RDS PostgreSQL 和 HybridDB for PGSQL 实时分析方案：

- 承受住了每秒几十万笔的写入吞吐并做数据清洗，是交易的数倍
- 实现分钟级延迟的实时分析，5 张十亿级表关联秒级响应
- 实时发现交易异常，提升淘宝的用户体验

业务背景

一个电商业务通常会涉及商家、门店、物流、用户、支付渠道、贷款渠道、商品、平台、小二、广告商、厂家、分销商、店主、店员、监管员、税务、质检等等角色，这些对象的活动会产生大量的浏览、订单、投诉、退款、纠纷等业务数据。而任何一笔业务，都会涉及很多不同的业务系统。

在这些业务系统中，为了定位问题、运营需要、分析需要或者其他需求，会在系统中设置埋点，记录用户的行为在业务系统中产生的日志，也叫 FEED 日志。比如订单系统、在业务系统中环环相扣，从购物车、下单、付款、发货，收货（还有纠纷、退款等等），一笔订单通常会产生若干相关联的记录。每个环节产生的属性可能是不一样的，有可能有新的属性产生，也有可能变更已有的属性值。

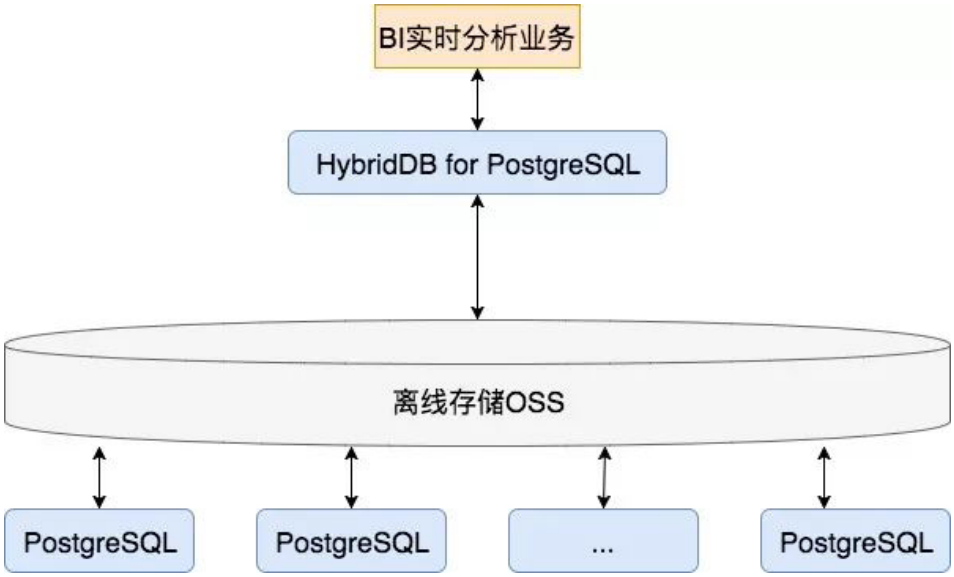
为了便于分析，通常有必要将订单在整个过程中产生的若干记录（若干属性），合并成一条记录（订单大宽表）。数据运营平台的主动服务，根据这些合并后的数据，实时的进行分析，进行实时的舆情展示，实时的找出需要主动服务的对象等，实现一个智能化的服务运营平台。

难点

除了实时性的要求以外，在写入的过程中，还有数据的切换、合并和清理等动作。做过数据库或数据分析的会知道：单独要做到每秒数十万笔吞吐的写入、切换、合并和清理并不算特别困难；单独要做到 TB 级数据的毫秒级分析也不算困难。但要做到实时写入的同时提供分钟级延迟的毫秒级实时分析，并做合理的调度就没那么容易了。

方案

为支撑这样的业务需求，采用的方案图示如下：



其中：

- RDS PostgreSQL 是阿里云基于开源关系型数据库 PostgreSQL 开发的云上版本
- HybridDB for PostgreSQL 是 MPP 架构的分布式分析型数据库，在多表关联、复杂查询、实时统计、圈人等诸多方面性能卓越，并支持 JSON、GIS、HLL 估值等多种独特的功能特性
- OSS，是阿里云推出的海量、安全、低成本、高可靠的云存储服务，此处用作数据的离线存储
- 最关键的，是实现 RDS PostgreSQL 和 HybridDB for PostgreSQL 对离线存储 OSS 的透明化访问能力

在该方案中，多个 PostgreSQL 接受业务的写入，在每个 RDS PostgreSQL 中完成数据的清洗，然后以操作外部表（类似堆表）的方式，将清洗完的数据写入弹性存储 OSS；而在写入完成后，HybridDB for PostgreSQL 也以操作外部表（类似堆表）的方式，从 OSS 中将数据并行加载到 HybridDB 中。在 HybridDB 中，实现几十、几百 TB 级数据的毫秒级查询。

在 PostgreSQL 中，创建一个外部表：

```
# 创建插件，每个库执行一次
create extension oss_fdw;

# 创建 server，每个OSS bucket创建一个
CREATE SERVER ossserver FOREIGN DATA WRAPPER oss_fdw OPTIONS
     (host 'oss-cn-hangzhou-zmf.aliyuncs.com' , id 'xxx', key 'xxx',bucket 'mybucket');

# 创建 oss 外部表，每个需要操作的OSS对象对应一张表
CREATE FOREIGN TABLE ossexample
    (date text, time text, volume int)
     SERVER ossserver
     OPTIONS ( filepath 'osstest/example.csv', delimiter ',' ,
        format 'csv', encoding 'utf8', PARSE_ERRORS '100');
```

这样即创建了映射到 OSS 对象的表，通过对 ossexample 的读写即是对 OSS 的读写。在数据写入”local_tbl”中后，执行以下 SQL：

```
  select date, time, volume) from local_tbl where date > '2013-09-30';
insert into ossexample
# 数据写入OSS
```

表”local_tbl”中满足过滤条件的数据，即会写入 OSS 对应的对象”osstest/example.csv”中。

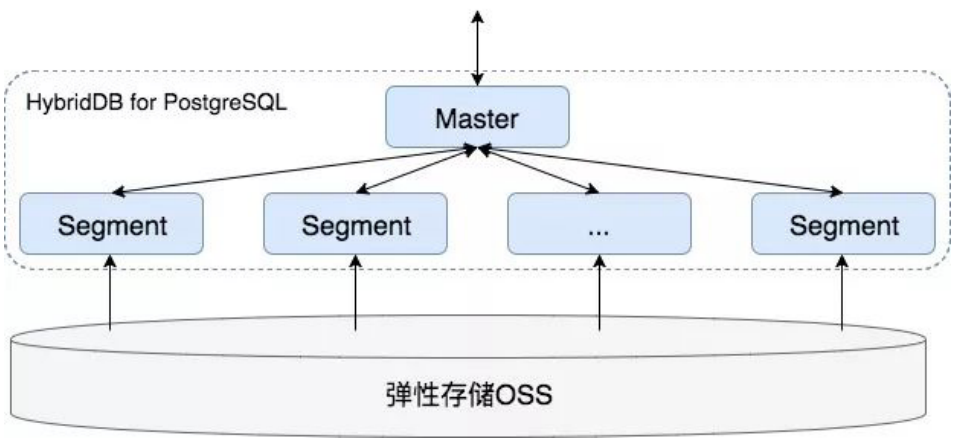
在 HybridDB for PostgreSQL 也用与此类似的方式读写 OSS。整个过程，用户看到的只是一条条 SQL。如下：

```
# 创建外部表，用于导出数据到oss
create WRITABLE external table ossexample_exp
        (date text, time text, volume int)
        location('oss://oss-cn-hangzhou.aliyuncs.com
        prefix=osstest/exp/outfromhdb id=XXX
        key=XXX bucket=testbucket') FORMAT 'csv'
        DISTRIBUTED BY (date);

# 创建堆表，数据就装载到这张表中
create table example
        (date text, time text, volume int)
         DISTRIBUTED BY (date);

# 数据并行的从 ossexample 装载到 example 中
insert into example select * from ossexample;
```

该 INSERT 语句的执行，即会将”osstest/exp/outfromhdb”文件中的数据，并行写入到表”example”中。其原理如下：



HybridDB 是分布式数据库，一个 HybridDB for PostgreSQL 集群中，有一个 Master 和多个 Segment，Segment 的个数可以横向扩充。Segment 负责存储、分析数据，Master 则是主入口接受查询请求并分发。

通过每个 Segment 并行从 OSS 上读取数据，整个集群可以达到相当高的吞吐

能力，且这个能力随 Segment 个数而线性增加。

方案优势

上面的方案初看起来并不复杂，却解决了下面几个问题：

1. 性能

融合了 PostgreSQL 超强的并发写入性能与 HybridDB 卓越的分析性能。

单个 RDS PostgreSQL 甚至可以支撑到百万级的写入；而写入 PostgreSQL 后批量加载到 HybridDB，使得 PostgreSQL 与 HybridDB 无缝衔接，利用 MPP 卓越的分析性能做到实时的毫秒级查询。

2. 数据的搬运与清洗

在传统的分析领域，数据的搬运往往是比较重、且性能较差的一环，导致 TP 和 AP 距离较远，只能采用截然不同的方式和节奏。而如果是异构数据库的搬运，则痛苦指数再上台阶。

如果这些，都可以通过 SQL 来操作，数据的清洗和搬运最终都只是 SQL 的定义与执行，岂不美哉?

在上图中，RDS PostgreSQL 和 HybridDB for PostgreSQL 都有直接读写 OSS 的能力，可以很容易地的串联起来。假以合理的调度和封装，可以以较低的成本实现原本需要很多工作量的功能。

3. 冷热数据的统一

而借操作离线存储的能力，可以将冷数据放在 OSS，热数据放在 PostgreSQL 或者 HybridDB for PostgreSQL，可以通过 SQL 以相同的处理方式实现对冷热数据的统一处理。

4. 动态调整资源

云生态的好处之一就是动态与弹性。RDS PostgreSQL 的资源可以随时动态调整，而不影响任何的可用性，相当于给飞机在空中加油；而对 HybridDB 的扩容与缩容，则是秒级切换即可完成。OSS 本身的弹性，也允许客户放多少的数据都可以。

因此，带来了如下几点优势：

1. 相比于传统的数据分析方案，以 SQL 为统一的方式进行数据的管理，减少异构

2. 资源动态调度，降低成本

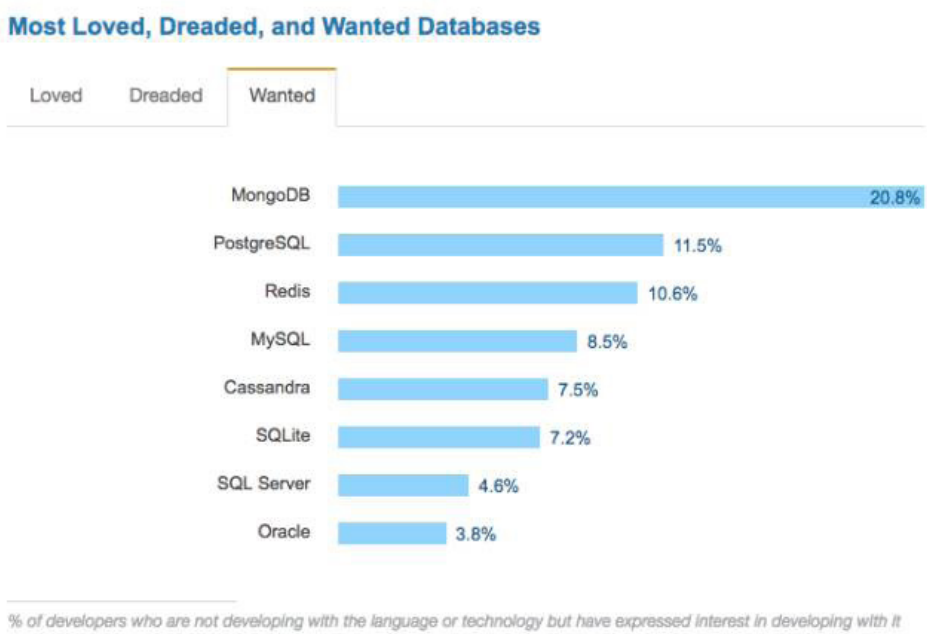
3. 冷热数据界限模糊，直接互相访问

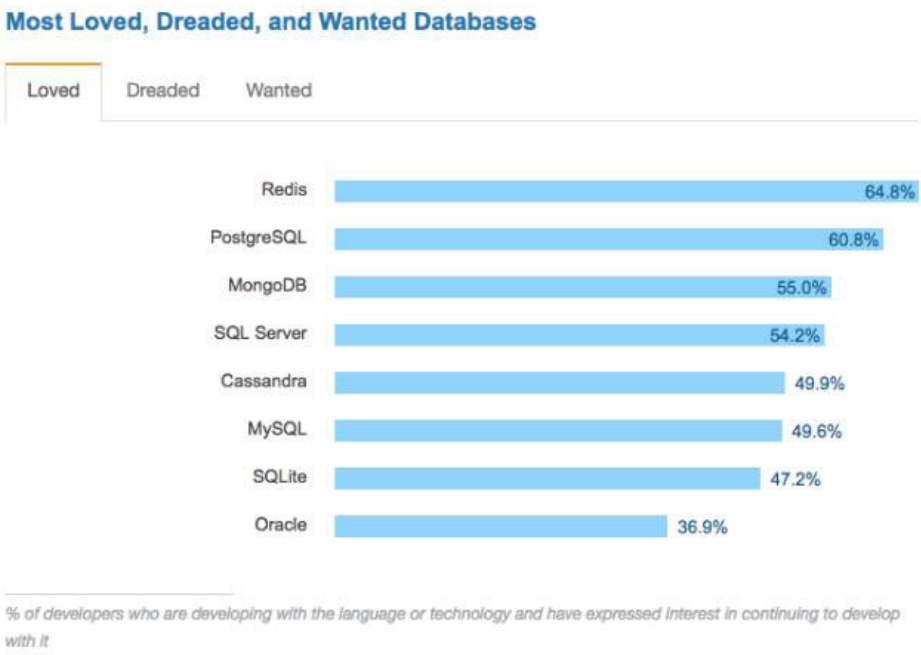
4.TP、AP 一体化

5.RDS PostgreSQL 的个数没有限制；HybridDB 集群的数量没有限制

阿里云云数据库 PostgreSQL

阿里云云数据库 PostgreSQL，基于号称“Most Advanced”的开源关系型数据库。在 StackOverflow 2017 开发者调查中，PostgreSQL 可以说是“年度统计中开发者最爱和最想要的关系型数据库”。





PostgreSQL 的优势有以下几点：

稳定

PostgreSQL 的代码质量是被很多人认可的，经常会有人笑称 PG 的开发者都是处女座。基本上，PG 的一个大版本发布，经过三两个小版本就可以上生产，这是值得为人称道的一个地方。从 PostgreSQL 漂亮的 commit log 就可见一斑。

而得益于 PostgreSQL 的多进程架构，一个连接的异常并不影响主进程和其他连接，从而带来不错的稳定性。

性能

我们内部有些性能上的数据，TPCC 的性能测试显示 PostgreSQL 的性能与商业数据库基本在同一个层面上，个别场景下性能甚至更好。

丰富

PostgreSQL 的丰富性是最值得诉说的地方。因为太丰富了，以至于不知道该如何突出重点。这里只列举几个认为比较有意思的几点（查询、类型、功能）。

功能的丰富

且不说 HASH\Merge\NestLoop JOIN，还有递归、树形（connect by）、窗口、rollup\cube\grouping sets、物化视图、SQL 标准等，还有各种全文检索、规则表达式、模糊查询、相似度等。在这些之外，最重要的是 PostgreSQL 强大的基于成本的优化器，结合并行执行（并行扫瞄、并行 JOIN 等）和多种成本因子，带来各种各样丰富灵活高效的查询支持。另外还有各种索引的类型，如 btree, hash, gist, sp-gist, gin, brin , bloom , rum 索引等。你甚至可以为自己定义的类型定制特定的索引和索引扫瞄。

PostgreSQL 有一个无与伦比的特性——插件。其利用内核代码中的 Hook，可以让你在不修改数据库内核代码的情况下，自主添加任意功能，如 PostGIS、JSON、基因等，都是在插件中做了很多的自定义而又不影响任何内核代码从而满足丰富多样的需求。而 PostgreSQL 的插件，不计其数。

FDW 机制更让你可以在同一个 PostgreSQL 中像操作本地表一样访问其他数据源，如 Hadoop、MySQL、Oracle、Mongo 等，且不会占用 PG 的过多资源。比如我们团队开发的 OSS_FDW 就用于实现对 OSS 的读写。

类型的丰富

如高精度 numeric, 浮点 , 自增序列，货币，字节流，时间，日期，时间戳，布尔，枚举，平面几何，立体几何，多维几何，地球，PostGIS，网络，比特流，全文检索，UUID，XML，JSON，数组，复合类型，域类型，范围，树类型，化学类型，基因序列，FDW, 大对象 , 图像等。

PS: 这里的数组，可以让用户像操作 JAVA 中的数组一样操作数据库中的数据，如 item[0][1] 即表示二维数组中的一个元素，而 item 可以作为表的一个字段。

或者，如果以上不够满足，你可以自定义自己的类型（create type），并且可以针对这些类型进行运算符重载，比如实现 IP 类型的加减乘除（其操作定义依赖于具体实现，意思是：你想让 IP 的加法是什么样子就是什么样子）。

查询的丰富

至于其他的，举个简单的例子，PostgreSQL 的 DDL（如加减字段）是可以在事务中完成的（PS: PostgreSQL 是 Catalog-Driven 的，DDL 的修改基本可以理解为一条记录的修改）。这一点，相信做业务的同学会有体会。

而在开源版本的基础上，阿里云云数据库 PostgreSQL 增加了 HA、无缝扩缩容、自动备份、恢复与无感知切换、离线存储透明访问、诊断与优化等诸多功能，解除使用上的后顾之忧。

阿里云 HybridDB for PostgreSQL

HybridDB for PostgreSQL 是 MPP 架构的分布式分析型数据库，基于开源 Greenplum，在多表关联、复杂查询、实时统计、圈人等诸多方面性能卓越。在此基础上，阿里云 HybridDB for PostgreSQL 提供 JSON、GIS、HLL 估值、备份恢复、异常自动化修复等多种独特的功能特性；并在 METASCAN 等方面做了诸多性能优化，相比开源版本有质的提升。

阿里云 HybridDB for PostgreSQL 有以下特点：

- 实时分析

  支持 SQL 语法进行分布式 GIS 地理信息数据类型实时分析，协助物联网、互联网实现 LBS 位置服务统计；支持 SQL 语法进行分布式 JSON、XML、模糊字符串等数据实时分析，助金融、政企行业实现报文数据处理及模糊文本统计。

- 稳定可靠

  支持分布式 ACID 数据一致性，实现跨节点事务一致，所有数据双节点同步冗余，SLA 保障 99.9% 可用性；分布式部署，计算单元、服务器、机柜三重防

护，提高重要数据基础设施保障。

- 简单易用

  丰富的 OLAP SQL 语法及函数支持，众多 Oracle 函数支持，业界流行的 BI 软件可直接联机使用；可与云数据库 RDS(PostgreSQL/PPAS) 实现数据通讯，实现 OLTP+OLAP(HTAP) 混合事务分析解决方案。

  支持分布式的 SQL OLAP 统计及窗口函数，支持分布式 PL/pgSQL 存储过程、触发器，实现数据库端分布式计算过程开发。

  符合国际 OpenGIS 标准的地理数据混合分析，通过单条 SQL 即可从海量数据中进行地理信息的分析，如：人流量、面积统计、行踪等分析。

- 性能卓越

  支持行列混合存储，列存性能在 OLAP 分析时相比行存储可达 100 倍性能提升；支持高性能 OSS 并行数据导入，避免单通道导入的性能瓶颈。

  基于分布式大规模并行处理，随计算单元的添加线性扩展存储及计算能力，充分发挥每个计算单元的 OLAP 计算效能。

- 灵活扩展

  按需进行计算单元，CPU、内存、存储空间的等比扩展，OLAP 性能平滑上升致数百 TB；支持透明的 OSS 数据操作，非在线分析的冷数据可灵活转存到 OSS 对象存储，数据存储容量无限扩展。

  通过 MySQL 数据库可以通过 mysql2pgsql 进行高性能数据导入，同时业界流行的 ETL 工具均可支持以 HybridDB 为目标的 ETL 数据导入。

  可将存储于 OSS 中的格式化文件作为数据源，通过外部表模式进行实时操作，使用标准 SQL 语法实现数据查询。

  支持数据从 PostgreSQL/PPAS 透明流入，持续增量无需编程处理，简化维护工作，数据入库后可再进行高性能内部数据建模及数据清洗。

- 安全

  IP 白名单配置，最多支持配置 1000 个允许连接 RDS 实例的服务器 IP 地址，从访问源进行直接的风险控制。

DDOS 防护，在网络入口实时监测，当发现超大流量攻击时，对源 IP 进行清洗，清洗无效情况下可以直接拉进黑洞。

总结

利用阿里云的云生态，RDS PostgreSQL、HybridDB for PostgreSQL 等一系列云服务，帮助企业打造智能的企业数据 BI 平台，HybridDB for PostgreSQL 也企业大数据实时分析运算和存储的核心引擎。实现企业在云端从在线业务、到数据实时分析的业务数据闭环。

阿里下一代数据库技术：把数据库装入容器不再是神话

张瑞

张瑞，阿里集团数据库技术团队负责人，阿里巴巴研究员，Oracle ACE。双十一数据库技术总负责人，曾两次担任双十一技术保障总负责人。自 2005 年加入阿里巴巴以来，一直主导整个阿里数据库技术的不断革新。

近日，在京举行的 2017 中国数据库技术大会上，来自阿里巴巴集团研究员张瑞发表了题为《面向未来的数据库体系架构的思考》的主题演讲。主要介绍了阿里数据库技术团队正在建设阿里下一代数据库技术体系的想法和经验，希望能够把阿里的成果、踩过的坑以及面向未来思考介绍给与会者，为中国数据库技术的发展出一份力。

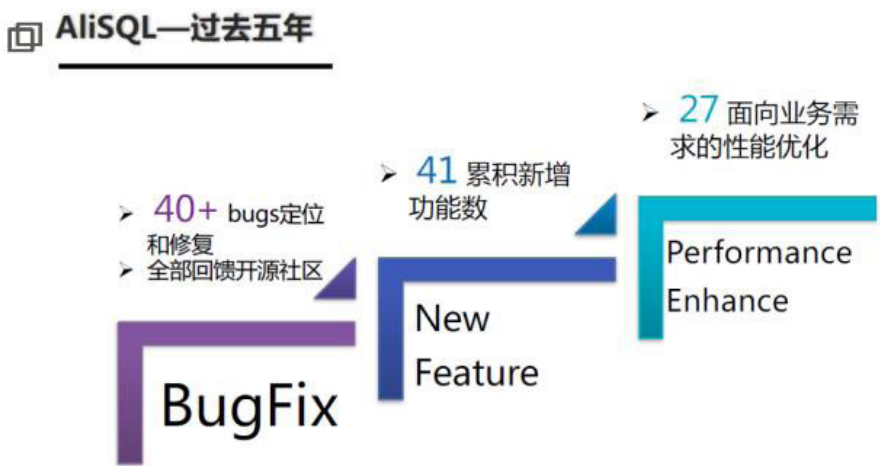
演讲全文：

我先介绍一下我自己，我 2005 年加入阿里一直在做数据库方面的工作，今天这

个主题是我最近在思考阿里巴巴下一代数据库体系方面的一些想法，在这里分享给大家，希望能够抛砖引玉。大家如果能够在我今天分享后，结合自己面对的实际场景，得到一些体会，有点想法的话，我今天分享的目的就达到了。

今天我会讲以下几方面内容：首先讲一下我们在内核上的一点创新、数据库怎么实现弹性调度、关于智能化的思考、最后是曾经踩过的坑和看到未来的方向。

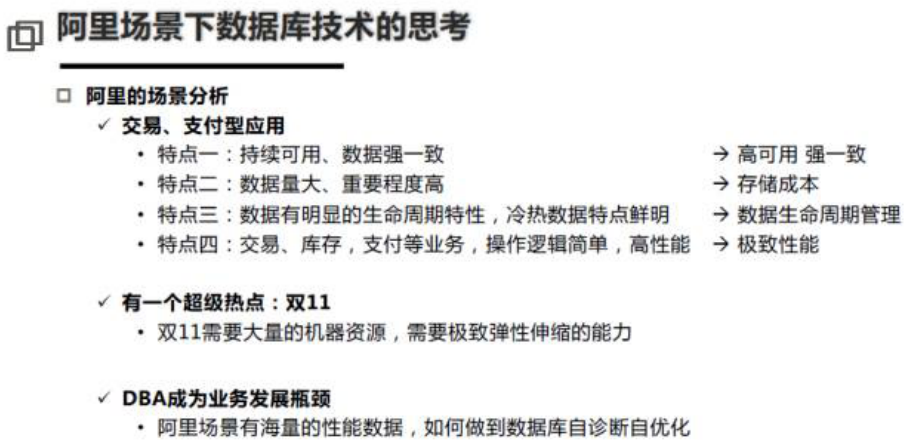
阿里场景下数据库所面临的问题



首先说一下，阿里巴巴最早一代使用的数据库技术是 Oracle，后面大家也知道一件事情就是去 IOE，去 IOE 过程中我们迈向了使用开源数据库的时代，这个时代今天已经过去，这个过程大概持续了五六年，整个阿里巴巴有一个大家都知道的开源 MYSQL 分支 --AliSQL，我们在上面做了大量的改进，所以我这里列了一下在 AliSQL 上的一些改进，但今天我实际上并不想讲这个，我想讲一下面向未来的下一代数据库技术、数据库架构会往哪个方向走。

我觉得是这样的，因为今天的阿里巴巴毕竟是一个技术的公司，所以很多时候我们会看比如说 Google 或者是一些互联网的大的公司，他们在技术上创新点来自于哪里？来自于问题。就是说今天在座的各位和我是一样的，你所面对场景下的问题是什么、你看问题深度如何决定了你今天创造的创新有多大。

所以今天我们重新看一下阿里面临的问题是什么，相信在座的各位一定也有这样的想法，阿里所面临的问题不一定是你们的问题，但我想说今天通过阿里面临的问题，以及我们看到这些问题后所做的事情，期待能够给大家带来参考，希望大家也能够看到自己所面临的问题是什么，你将如何思考。



可以看到其实阿里巴巴的应用和 Facebook、Google 的还是有很大区别的，我们也找他们做了交流，发现跟他们的业务场景真的不一样，首先我们的主要应用是交易型的，这些应用会有些什么要求，你会看到有这些点（见图片），下面主要讲一下我们的思考。

今天数据的高可用和强一致是非常重要的，数据不一致带来的问题是非常非常巨大的，大家也用淘宝，也是阿里巴巴一些服务的用户，数据不一致带来的问题，每一个用户、甚至我的父母都会关注这些事情。

第二，今天存储成本是非常高的，所有的数据中心已经在用 SSD，但数据的存储成本依然是一个大型企业面临的一个非常大的问题，这都是实实在在钱的问题。

另外刚才也提到了，数据都是有生命周期的，那么数据尤其是交易数据是有非常明显的冷和热的状态，大家一定很少看自己一年前在淘宝的购买记录，但是当下的购买记录会去看，那系统就需要经常会去读它、更新它。

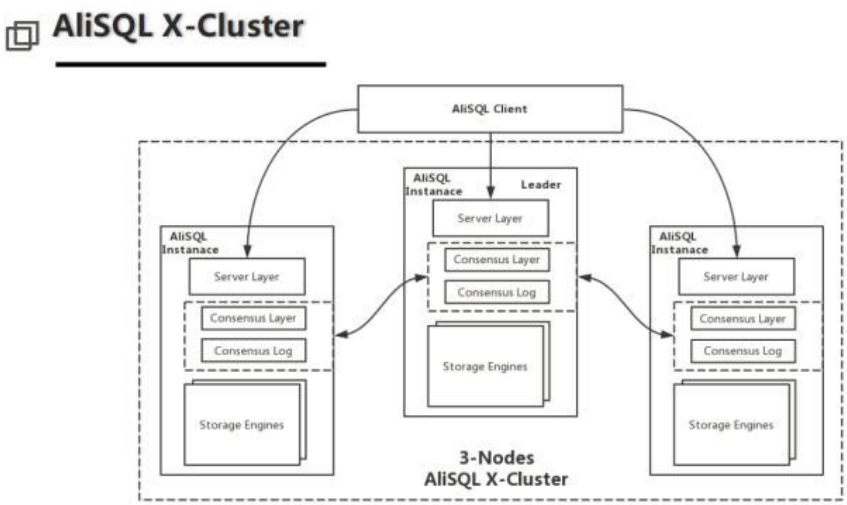
还有一个特点是今天阿里的业务还是相对简单的，比如我们要在 OLTP 性能上

做到极致性。还有一个阿里巴巴特有的点就是双十一，双十一本质上是什么，本质上就是制造了一个技术上非常大的热点效应。这对我们提出什么样的需求呢？需求就是一个极致弹性的能力，数据库实际上在这个方向是非常欠缺的，数据库怎么样去做到弹性伸缩是非常难的事情。

最后我想说说 DBA，今天在座的很多人可能都是 DBA，我想说一下阿里在智能化这个方向上得到的思考是什么样的，我们有海量的数据，我们也有很多经验很丰富的 DBA，但这些 DBA 怎么样去完成下一步的转型、怎么样不成为业务的瓶颈？数据库怎么样做到自诊断、自优化。这是我们看到的问题，最后我也会来分享一下我在这方面的思考。

阿里在数据库内核方向上的思考

我先讲一下我们在数据库内核上的思考。首先我很尊敬做国产数据库的厂商，凡是在内核上改进的人都知道，其实每个功能都是要一行行代码写出来都是非常不容易的，我想表达对国产数据库厂商包括这些技术人的尊敬。今天我要讲的内容是我第一次在国内的会议上来讲，首先我会讲一下 AliSQL X-Cluster。X-Cluster 是在 AliSQL 上做的一个三节点集群，这是我们在引入了 Paxos 一致性协议，保证 MySQL 变成一个集群，并且这个集群具有数据强一致、面向异地部署，且能够容忍网络高延迟等一系列特性。



今天很多数据库都会和 Paxos 联系在一起，比如大家都知道的 Google 的 Spanser 数据库，但是以前大家没有特别想过数据库和 Paxos 会有什么关系，其实以前确实没有什么关系的，但是今天的数据库在几个地方是需要用到 Paxos 协议的，第一个我们需要用 Paxos 来选举，尤其在高可用的场景下需要唯一地选举出一个节点作为主节点，这就需要用到 Paxos；第二是用 Paxos 协议来保证数据库在没有共享存储的前提下数据的强一致，就是数据怎么样在多个节点间保证是强一致，且保证高可用。

所以说现在的数据库架构设计上 Paxos 的应用非常广泛，今天外面很多展商包括 Goolge Spanser 也都在用 Paxos 协议和数据库结合在一起来做。所以 AliSQL 的三节点集群也是一样，就是利用 Paxos 协议变成一个数据强一致的集群。下面我大概解释一下 Paxos 协议在数据库里的作用是什么。

本质上来说 Paxos 也是现在通用的技术，大家都是搞数据库的，简单来说，Paxos 协议用在我们数据库里面，就是一个事务组的提交在一个节点并落地后，必须在多个节点同时落地，也就是说原来写入只需要写入一个节点上，但是现在需要跨网络写到另外一个节点上，这个节点可能是异地的，也可能是全球的另外一个城市，中间需要经过非常漫长的网络时延，这时候需要有一些核心技术。

我们的目标是什么？首先没有办法抗拒物理时延，过去在数据库上的操作只要提

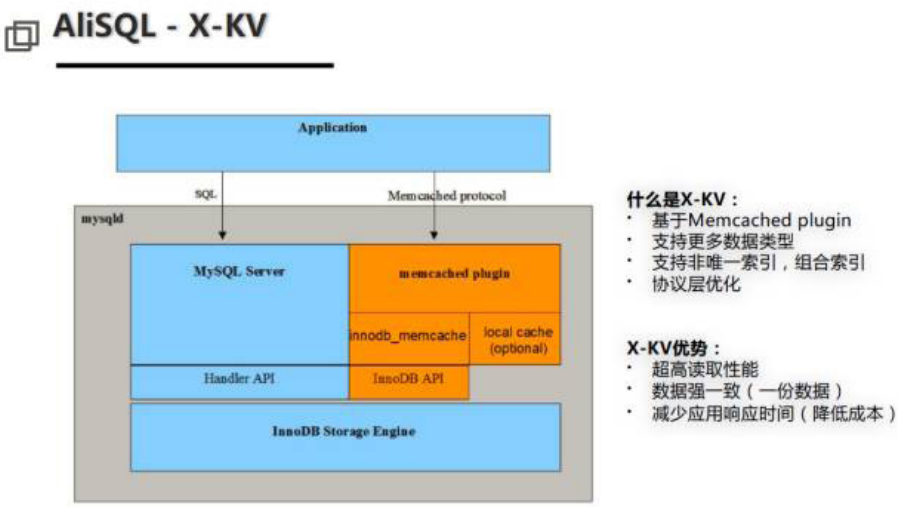
交到本地，但现在数据库全球部署、异地部署，甚至跨网络，这个时延特性是没有办法克服的，但是在这种情况下我们能做到什么？在时延增长情况下尽可能保证吞吐不下降，原来做多少 QPS、TPS，这一点是可以保证的，只要工程做的好是可以保证的，但是时延一定会提升。

这也是大家经常在 Goolgle Spanser 论文里看到的“我的时延很高”的描述，在这种时延很高的情况下，怎么样写一个好的应用来保证可用、高吞吐，这是另外一个话题。大家很长一段时间里已经习惯一个概念，那就是数据库一定是时延很低的，时延高就会导致应用出问题，实际上这个问题要花另外一个篇幅去讲，那就是应用程序必须要去适应这种时延高的数据库系统。当然用了 Batching 和 Pipelining 技术，本质上是通用的工程优化，让跨网络多复本同步变的高效，但是时延一定会增加。

实际上大家知道数据库要做三副本或者三节点，本质上就是为了实现数据强一致，而且大家都在这个方向上做努力，比如 Oracle 前一段时间推出的 Group replication，也是三节点技术，X-Cluster 和它的区别是，我们一开始的目标就是跨城市，最开始设计的时候就认为这个节点一定要跨非常远的距离来部署的，设计之初提出这个目标造成我们设计上、工程实践上、包括最终的性能上有比较大的差异。

这里我们也做了一些 X-Cluster 和 Oracle 的 Group replication 的对比，同城的环境下我们要比他们好一些；在异地场景下这个差异就更大了，因为我们本来设计的时候就是面向异地的场景。可能大家也知道，阿里一直在讲异地多活的概念，就是 IDC 之间做异地多活怎么样能够做到，所以最开始我们设计的就是面向异地的场景做的。

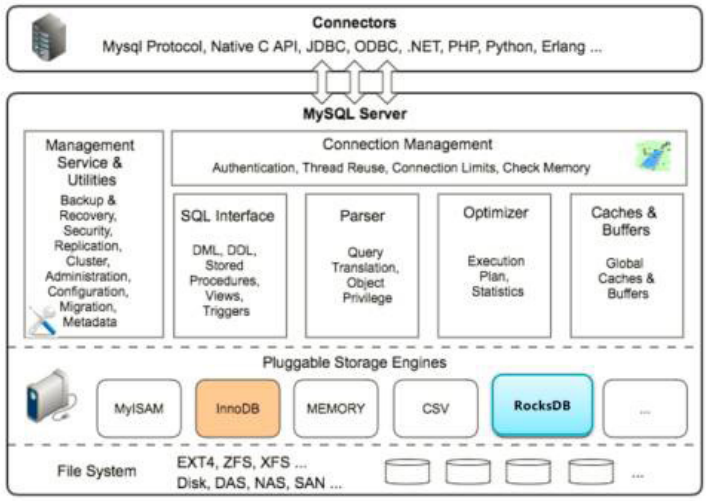
这是一个典型的 X-Cluster 在异地多活的场景下怎么做的架构图，这是一个典型的 3 城市 4 份数据 5 份日志架构，如果要简化且考虑数据存储成本的话，实际上可以做到 3 份数据 5 份日志，这样的话就可以保证城市级、机房机、包括单机任何的故障都可以避免，并且是零数据丢失的，在今天我们可以这么做，且能保证数据是零丢失、强一致的。在任何一个点上的数据至少会被写到另一个城市的数据中心的数据库里面，这是我们 X-Cluster 设计之初的目标，这也是一个典型的异地多活的架构。



再讲一个小的，但是非常实用的创新点，可能大家都比较感兴趣，这就是 X-KV。这里还要说一下，我们所有的下一代技术组件都是以 X 开头的。这个 X-KV 是基于原来 MYSQL 的 Memcached plugin 做的改进，做到非常高的性能，大家可能都了解 MySQL 的 Memcached plugin，可以通过 Memcached plugin 的接口直接访问 InnoDB buffer 里的数据，读的性能可以做到非常高，这对于大家来说，或者对于所谓的架构师，或者设计的过程中意义在什么地方呢?

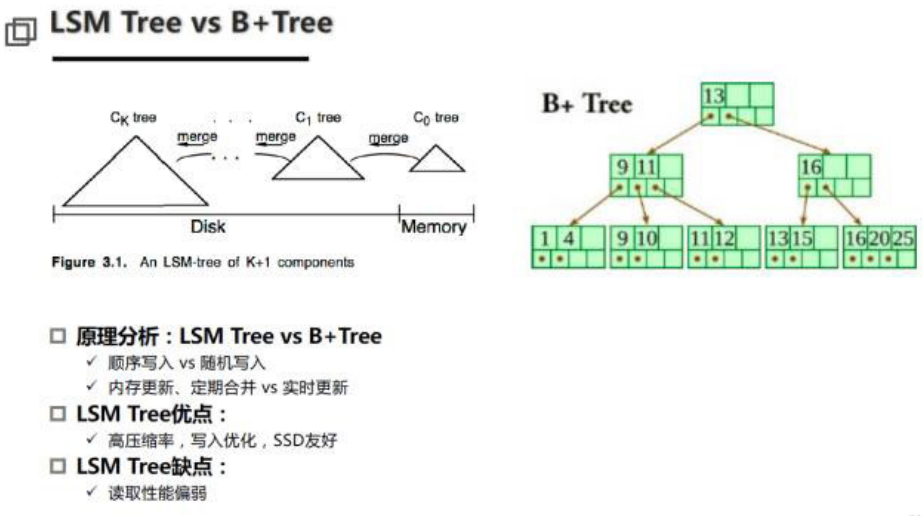
那就是很多场景下不需要缓存了，因为数据库 + 缓存结构基本上是所有业务通用的场景，但是缓存的问题在于缓存和数据库里的数据永远是不一致的，需要一个同步或者失效机制来做这个事情。使用 X-KV 后读的问题基本上就能解决掉。这是因为一份数据只要通过这个接口访问就基本上做到和原来访问缓存差不多的能力，或者说在大部分情况下其实就不需要缓存了。

AliSQL多引擎架构



第二是说它降低了应用的响应时间，原来用 SQL 访问的话响应时间会比较高，我们在这上面做了一些改进，本来 Memcached plugin 插件有一些支持数据的类型限制，包括对一些索引类型支持不太好，所以我们做了改进，这个大家都可以用的，如果用这个方式的话基本上很多缓存系统是不需要的。

第三个事情我想讲一下怎么样解决冷热数据分离的，我们天然地利用了 MySQL 的框架，这里就直接拿了 MySQL 的大图来展示，大家可以看到 MySQL 本质上来说就是上面有个 Client，中间有个 Server，底下有个存储层，在存储层里面可以有各种各样的引擎，所以通过不同的引擎可以实现不同的特性。大家今天最常用的就是 InnoDB 引擎，每个存储引擎的特性本质上是由其结构造成的。比如 InnoDB 采用 B+ Tree 的结构，它带来的特性就是相对来说读和写都比较均衡，因为发展了这么多年确实是比较成熟的。



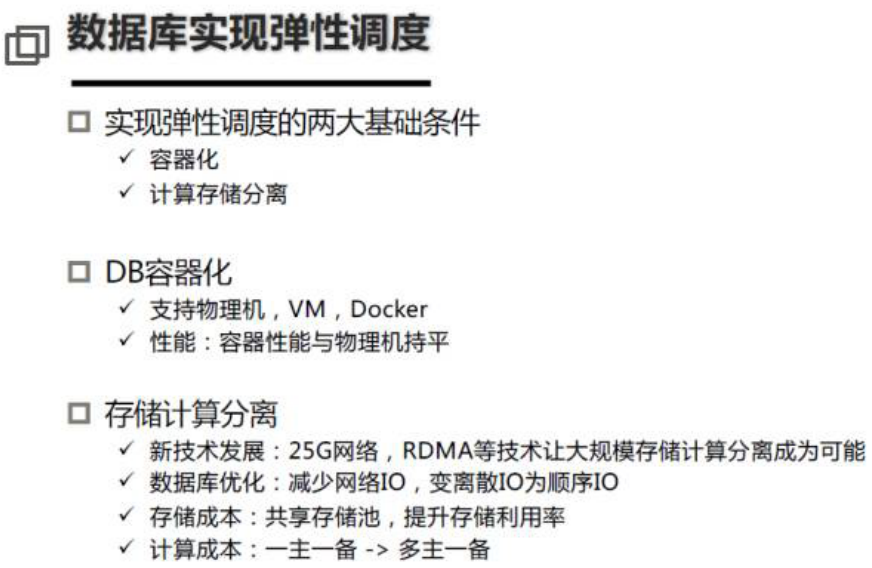
比如我们现在选择 RocksDB，这是因为我们和 Facebook 在 RocksDB 上有一些合作，就是把它引入到 MySQL 上面，它本质的结构是 LSM Tree，这个结构就带来的好处包括写入比较友好、压缩的特性好等。把它引入进来对我们的改革还不仅仅是引入了一个结构，而是今天我们用这两种引擎解决我们今天数据分离的问题。我们也跟 Facebook 有过一些交流，RocksDB 今天并没有那么稳定、那么好，但是作为 InnoDB 存储引擎的补充的话，是非常有效的。

尤其在稳定数据库的背景下，用户今天怎么样才能对自己的数据的冷热没有太多的感觉，因为大家可能也知道，你们以前也有一些数据的分离，但是对应用方来说，需要把数据从某个存储倒到某个存储里，然后再删掉；或者动不动 DBA 去找业务开发方说你的存储空间不够了，占用很多空间，能不能删一些数据或者把这些数据导入到成本更低的存储引擎里。我们经常这么干，这里说的直白一点，我相信大家都这么干过。

但是用这种双引擎结构的话，RocksDB 压缩率高的特性，特别是 OLTP 行存储的场景下，能够给我们带来比较大的收益。所以我们可以把这两个引擎在 MySQL 特性下面把它结合起来，并且可以利用到比较廉价的架构，尤其是 LSM Tree 这种架构，他对廉价的存储介质是比较友好的，因为他的写都是顺序写的。这就是我们今天在数据库内核上面的一些思考。

数据库为什么要实现弹性调度

第二部分，我想说一下数据库弹性调度这个事，大家都知道阿里双十一，双十一对我们来说最大的挑战就是应用程序可能已经很容易去做弹性调度，包括上云、弹性扩容和缩容，但是数据库确实比较难，我们也在这上面也探索了一段时间，今天会把我们的思考分享给大家。



我之前听好多人说数据库容器化是个伪命题，为什么要做容器化，为什么要把数据库放到容器里呢？第二也有一些新技术，比如刚才的分享嘉宾也提到的，把存储放在远端通过网络访问是可能的。但是我们从正向来思考，先别想数据库弹性调度可能不可能，数据库如果要实现弹性调度，它的前提是什么？

我们先去想数据库要像应用一样非常简单的弹性调度，那么数据库要做到什么？我觉得有两大前提是必须要做的：1、它必须要放到一个容器里；2、计算和存储必须分离。因为如果计算和存储本质上不分离的话，数据库基本上没有办法弹性调度。大家知道计算资源是很容易被移动的，但是存储资源基本上很难在短时间内移动它，所以做弹性是非常非常困难的。所以这是两大基础条件。

在我们的场景下如果你也碰到这种问题的话，那就不是伪命题，我觉得这个东西合不合理，更多时候不是技术有没有正确性，而是在你的场景下是否需要它，所以今天我们就做了两件事情，第一是把它放到容器里面，我们目前物理机，VM 和 Docker 都在支持，有一层会把容器的复杂性屏蔽掉，数据库一定要放到容器里。应用程序放到容器里很多时候是为了部署，但是我们把数据库放容器里就是为了做调度，因为数据库本身没有特别多的发布，不需要像应用一样做频繁发布。做了容器化之后，数据库在一个物理机上可以和其他的容器做混部。

我们做 DBA 的都有一些传统的观点，比如数据库服务器上肯定不能跑应用，数据库肯定是不能用容器的。不知道在座的各位，每当有人或者你的老板问你这个问题的时候，你是不是从来都是马上回绝他说“数据库肯定不能这么做”，但是今天你也许可以告诉你的老板可以试一试。

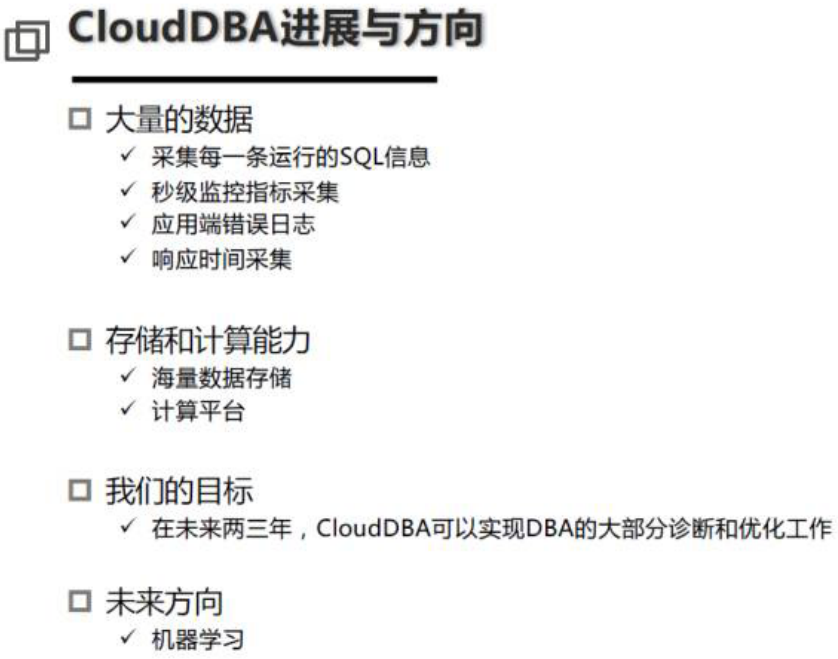
存储计算分离，最早做数据库的时候，存储和计算其实就是分离的，用一个 Oracle 的数据库，用一个 SAN 网络，底下接一个存储，存储和计算本身就是分离的，中间用 SAN 网络连起来。然后演进到用 Local 的盘，用 SSD 盘，用 PC 做服务器。，那未来重新要回到存储和计算分离的结构下，今天的网络技术的发展，不说专有网络，就说通用的 25G 网络，还有 RDMA 和 SPDK 等新技术的使用，让我们具备了存储计算分离的能力，让数据库存储计算分离的条件已经具备。

今天在数据库上已经看到大量优化的特性可以减少 IO，可以把离散的 IO 变成顺序的 IO，可以对下层的存储做的很友好。从存储成本上来说，共享存储会极大的降低成本，是因为存储碎片会被极大地压缩，因为原来每个机器上都空闲 30%、50% 的空间，其他的机器是很难利用到的，当你今天把这些碎片变成一个 Pool 的时候是有很大收益的。

还有数据库未来如果采用存储和计算分离的话，就会打破目前主流的数据库一主一备的架构，这个架构至少有一半的计算资源是被完全浪费的，不管你的备库是否用来做报表或者其他的应用，但是基本是浪费的。如果可以做到共享存储，那这将是一个巨大的收益。这是我们在调度上的思考，明天分会场上也会有一个阿里同学就这个主题给大家做容器和存储资源上的细节介绍，我今天只讲了一个大概。

DBA 未来的工作内容是什么？

最后讲一下 DBA 的事情，刚才也在说，我这里说从自动化走向智能化，我们内部讲从自助化走向智能化，不知道大家是不是受到一个困扰，业务发展的速度远远大于 DBA 人数的增长，如果你没有后面的这些我可以不讲了，但是如果你有，你可以听一下我们在这方面的思考，我们也碰到同样的问题，DBA 要怎么样的发展，自动化的下一步应该做什么，很多人说 DBA 是不是会被淘汰掉，至少我们想清楚了这些问题之后，阿里的 DBA 不纠结这个事情，所以我今天跟大家分享一下这个思考。



首先我们今天做了一个事情，我们放弃了原来的思路，原来的思路是什么呢？最早的时候我们每个上线的 SQL 都需要 DBA 看一下；第二个阶段，我们做了一个系统，在每个 SQL 上线之前系统都要预估一下它的性能好不好，如果好才上线。所有我们今天觉得最大的变化和思考是什么？所有基于单条语句的优化都是没有特别多意义的，因为只有基于大的数据和计算，才有可能变成一个智能化的东西，否则都是基于规则的。

基于规则的系统是很难有特别长久的生命力，因为有永远写不完的规则。我们也曾经做过这样的尝试，一些 SQL 进来的时候，系统要对它进行一些判断，最后发现永远写不完的规则。所以后来我们就找到了另外一个方向，我相信今天在座的所有人，你所在的公司不论大小都都有一个监控系统，我们就从这个监控系统出发，怎么样把一个监控系统变成一个智能的优化引擎，我们在这里也不说是大脑，就说是引擎好了。这个引擎会做什么?

首先来说，我们已经放弃掉基于单条 SQL 的优化，因为没有意义，DBA 也没有审阅单条 SQL，系统去看单条 SQL 的意义也不大。今天我们的第一个场景是说大量的数据，大量的数据是什么? 我们就从我们的监控系统出发，提出了第一个目标，把每一条运行的 SQL 采集下来，不是采样，是每一条。在规模比较大的系统来说对存储来说是个巨大的压力，因为这样会产生大量的副产品。

就像 Facebook 在做监控产品时产生的时序数据库一样，今天我们产生的副产品也是在时序数据库方面带来压力，这个具体的我今天先不展开。我们采集每一条 SQL 的运行情况，因为我们在内核里做了改进，可以把每条 SQL 的来源、路径、以及它在数据库里所有的信息全部采集下来。把监控指标压到秒级，所有监控项的指标必须最小达到秒级，这是我们现有的技术能够做到的。

另外，我们把应用端日志和数据库结合在一起。原来做数据库的时候，应用方吼一嗓子说“数据库有没有问题啊”DBA 说没有问题。但是从应用那端看，其实看到数据库有很多问题，包括应用报错，包括响应时间，把应用端报错也要和数据库结合在一起，尤其是应用里面报数据库的错误，以及这一整条链路。

响应时间，只有应用端的响应时间才是真正意义上可以衡量一个数据库是不是好的指标，而不是数据库本身怎么样，什么 Load 低啊，CPU 利用率多少。当把这些数据全部采集下来之后，这些大量的时序数据我们叫做副产品，这对我们整个链路产生了一个巨大的压力。我们做整个监控系统平台的同学觉得日子要活不下去了，因为原来的存储系统支撑不了、分析系统支撑不了、原来的平台计算不出来。所以先从这个目标考虑，基于链路做了巨大的改进，包括怎么样实现廉价存储、怎么样实时分析，这是存储和计算的要求。

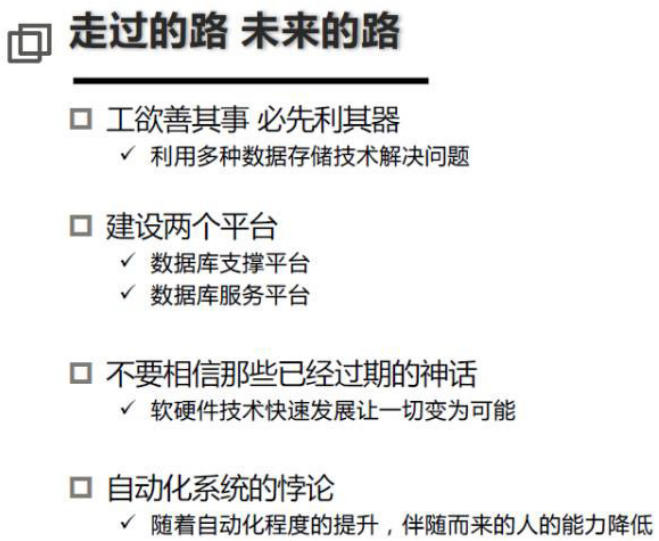
我们今天这个目标是在阿里内部明确提的，我们希望两到三年内希望大部分把 DBA 的工作替换掉，我不知道两到三年能不能做到，我希望能做到。其实今天 DBA 是这样的，DBA 的工作本质上分为两类，第一类就是运维，但运维本质上来说是比较好解决的，不管是用云，小公司用云全搞定，大公司基本上都有一些自动化运维的系统。

但是最难解决的就是刚才我说的诊断和优化。我也了解过很多公司，比如说 Google、facebook，我说你们为什么没有 DBA 呢？他们说我们没有 DBA，没有像国内这种特别传统的针对诊断和性能优化的 DBA，这种职责很少。所以这个东西希望能够做到。

最后我们有了数据、有了计算，我们觉得未来的方向可能就是现在比较火的机器学习，这个主题明天也有一个阿里同学会来分享，机器学习这里我就不多讲了，因为我觉得我们也在入门，所以没有什么值得讲的，但是我们觉得这个设计挺有戏的，你只要积累足够的数据和计算的话这个事情还是挺有戏的。

我们对数据库未来的其他思考

最后一页 PPT 我用大白话讲一下我对整个数据库体系的一些理解。



今天在一个公司里边没有一个存储或者是数据库可以解决所有问题，今天越来越多的趋势看到，数据存储的多样性是必然会存在的，因为行存有行存的优势、列存有列存的优势、做计算有计算的优势、做分析有做分析的优势、做 OLTP 有 OLTP 的优势，不要指望，或者很难指望一个系统干所有的事情很，这个话我说了可能不太好，但是确实比较难，但是我们看到的一点是什么？就是每个技术或产品在生产中干一件事情可以干到最好，你就用它做的最好的那件事解你的问题就好了。

这就回到之前的问题，我们也走过一些弯路，数据存储类型越来越多，今天用这个明天用那个，怎么办？我们的运维没法搞定，这个支持很痛苦。

所以今天我们建议建立两个平台：1、建立一个支撑的平台，这个支撑的平台尽可能把下层存储的复杂性屏蔽掉，对上层提供统一的接口和服务；2、建立一个服务的平台，明确面向研发的平台，研发人员可以直接通过这个平台来用数据库的服务。我看到很多公司把运维平台和 DBA 开发的平台混在一起，但阿里的思路是，支撑平台和服务平台是两个分层的平台，支撑平台在下面，上层服务平台为所有的开发人员服务，开发人员上了这个平台就能看到我用了什么数据库，性能怎么样，在上面可以做什么事情，这样就可以大量节省 DBA 的人力。

我们内部有句开玩笑的话叫“不节省人力的平台、不节省成本的技术都是耍流氓”，这句话怎么讲？就是说我们的自动化系统，尤其是大公司越建越多，最后的结果就是人没有能力了，我不知道大家有没有这个问题，这就是我最后讲的一点，自动化系统的悖论。每个公司每个人今天你们在做自动化系统的过程中有没有发生一件事情？反正在阿里是发生了，就是人的能力弱化了。

这个自动化系统的悖论是我们无意中看到的，在讲飞机的自动驾驶的时候，因为自动驾驶做的足够好，当出现紧急问题的时候，飞机驾驶员反而没有足够的能力去处理紧急的情况，这就是自动化系统的悖论。

可以对比看一下，我们今天做了很多自动化系统，结果人只会点系统，系统一卡壳就完蛋，很多次生故障都是出现在系统卡壳，卡壳人搞不定，怎么办？这是今天要去想的问题，在这个过程中今天所有带团队的或者今天在这个体系的人都要思考的问题，我们也在直面这个问题，让人的能力和系统的能力能够结合在一起，这是另外一

个话题，我今天不能给出答案，但是要特别重视这些问题。

不要相信那些已经过期的神话，数据库存储和计算是可以分离的，数据库也是可以放在容器里的，但你真的要去看一下，原来那些神话或者是那个背后它的问题到底是什么，其实现在可能都已经有解法了，所以在座的各位，当你的老板、CTO 或者什么人来问你“能不能做到这样?”我希望你能告诉他“我能!”

我们内部有一句话原来我们的 DBA 哪里看过一篇文章，说 DBA 的概念是什么? 我印象特别深，有一个开发的同学在底下的回复是说“DBA 就是一群永远在说不的人”就是不能这样不能那样，我们今天我觉得未来我们变成一群永远可以说“yes”，说“可以”的人，谢谢!

接下时序数据存储的挑战书，阿里 HiTSDB 诞生了

钟宇

近日，2017 中国数据库技术大会在京召开，来自阿里巴巴中间件团队高级技术专家钟宇（花名悠你）在数据存储和加速技术专场分享了题为《时间序列数据的存储挑战》的演讲，主要介绍了时序数据的由来，时序数据处理和存储的挑战，以及目前业界的通用做法。在案例展示部分，他结合阿里内部业务场景和时序数据的特点，讲述阿里时序数据处理和存储所面临的问题以及解决问题的过程，以及不断应对挑战慢慢形成 HiTSDB 的过程。

演讲全文：

钟宇：大家好，我叫钟宇，花名悠你（Uni），来自阿里巴巴中间件（Aliware）团队。首先我大概给大家介绍一下今天的分享内容，共有五个方面，首先跟大家介绍一下什么叫做时间序列，时序数据这个领域是一个比较专的领域，但同时它的范围又很广，因为我们平时遇到的数据里头有相当大的一部分都是时序数据，但是它又相对来说比较专，因为它不像我们一般的数据库什么都可以做，所以时序数据这个领域还是蛮特别的。

接下来我会说一说时序数据的存储和分析的一些方案，因为处理时序数据有很多的方案可以用各种各样的东西来做。给大家举一个例子，我高中的时候刚开始学计算机，我当时特别不能理解，为什么世界上会有数据库这样的东西，因为有个东西叫excel，它和数据库一样是一个表格，我只要把数据库放上去，在里面搜索一个数据比数据库还快，所以我想为什么要有数据库。

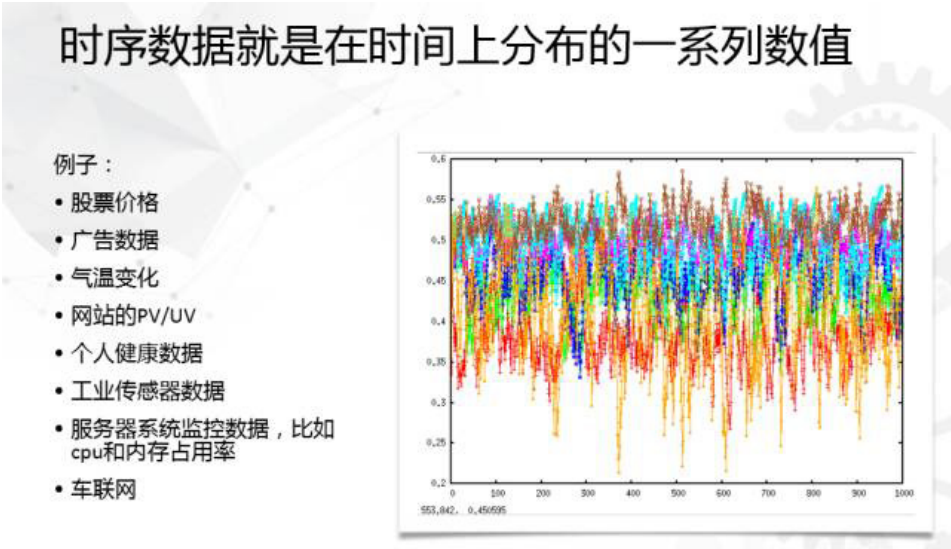
等到用的东西多了以后会发现，数据的存储和分析，excel 是一种方法，数据库也是一种方法，针对的数据量和应用场景是不一样的。时序数据其实也是一样的，就是说我们可以有很多种不同的方式来存储和分析时序数据，就好比是分析一般的表格，用 excel 可以，用 MYSQL 也可以，完全是针对不同的场景做出的不同选择。

接下来我会介绍一下时序数据库，因为时序数据库是时序数据存储和分析的一个非常重要的工具。我们考察过很多时序数据库，包括 influxDB 和 OpenTSDB），我们对此做出了改进来适应阿里的特殊应用场景，接下来我会着重介绍一下阿里对时序数据库的一些改进。

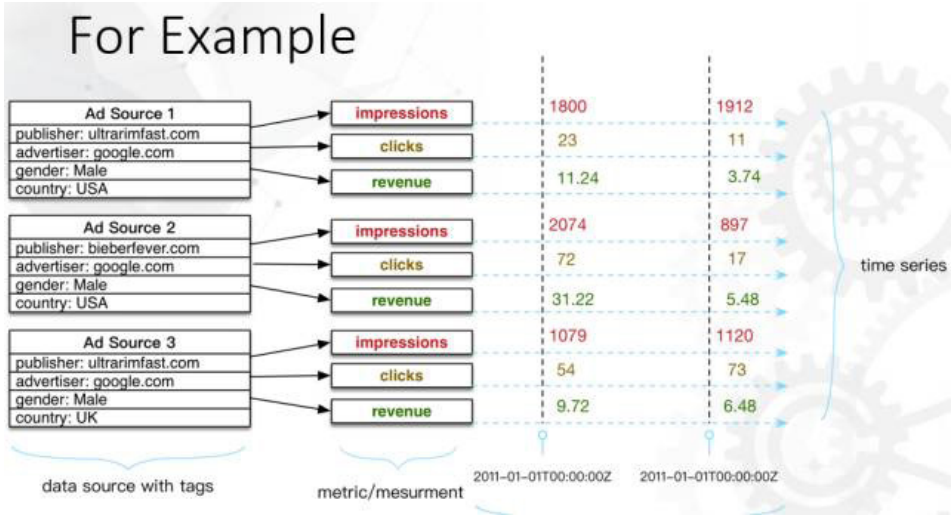
目前来看时序数据是比较年轻的领域，没有什么标准，整个行业发展也比较稚嫩，有很多方面并不完善，所以最后看一下，它还有哪些方面的发展是需要我们来继续的。

时序数据特征及常用应用场景

先跟大家聊一下什么叫时序数据。可能大家平时了解的不太多，这个东西很简单，就是时间上分布的一系列数值，关键字是数值，我们一般认为的时序数据是什么时间发生了什么事情，但是在时序数据这个领域里定义的时序数据全都是跟数值有关的。也就是说如果只是一个带有时间戳的一条数据并不能叫做时序数据。举个例子，比如像我早上 8 点半上楼吃了个饭这条记录，相当于一个日志，这个本身不构成一个时序数据，但是如果某个餐厅早上点半同时有 50 个人在那里吃饭，这个 50 加上餐厅的信息再加这个时间点就构成了一个时序数据。



更明显的例子比如说股票价格，每一个时刻每一支股票都有一个交易的价格，这种其实是时序数据。还有很经典的应用是广告数据，广告数据中很多像 PV、UV 一类的东西，这些都是在某一个时间点上一个数值型的东西。然后就是一些工业和科研上的东西，气温变化、工业上的传感器的数据都是这样的。



最近还有一个比较火的概念，大家会想到物联网，物联网的数据大部分都是时序数据，所以从这个角度来说，时序数据可能占我们平时需要处理的数据量中相当大的一部分。

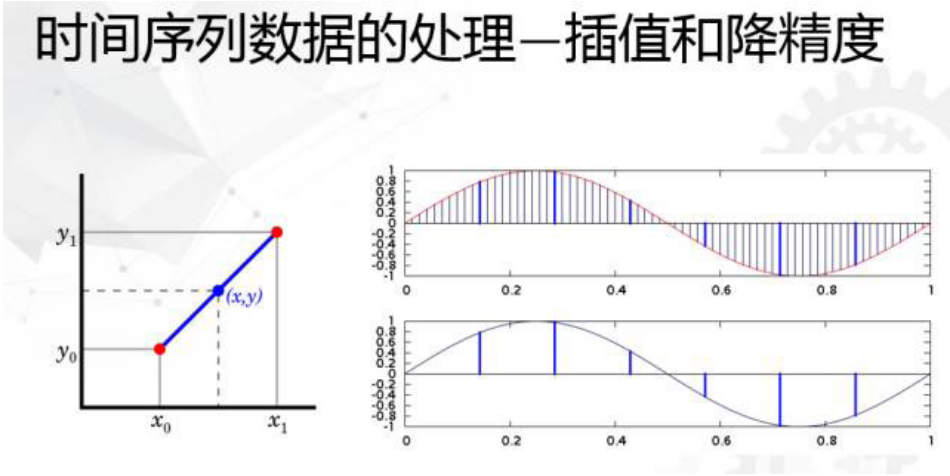
举个广告的例子，我们会发现一个时序数据除了有一个时间点以外，它还有一些别的部分，从头开始说，首先是要有一个东西来标注数据源，举个例子做广告，广告有很多的来源，在这个图表里大家会看到是 AD source1、2、3，这 3 个是不同的广告源，为了区分不同的广告源在上面打了一些标签，1 这个网站实际上是面向 google 发布的，面向的群体是男性，这是一个广告源，这些标签就标识了唯一的数据源，每一个数据源它都有很多的指标，我们把它叫一个指标或者一个测量。

我们在这个广告源上测量了很多次，比如说会测量它被查了多少次、会测量它被点击多少次、会测量它产生了多少收入，我们每个广告源都做了三次的测量，这样每个广告源就产生了三个时间序列，我们叫 time series，每个广告源有三个时间序列，在这个时间序列上在每一秒钟都会对它进行一次测量，所以每次测量会产生一个时间序列的值，我们把它叫做一个点，基本上时序数据的样子就是这样的。

然后再来看建模，实际上通用的建模方式有两种，其中的一种是单值。实际上我们是针对不同的东西来建模的，多值的模型是针对数据源建模，我们每一行数据针对的是一个数据源，它的三个被测量的指标在列上，所以每一个数据源，数据的来源在

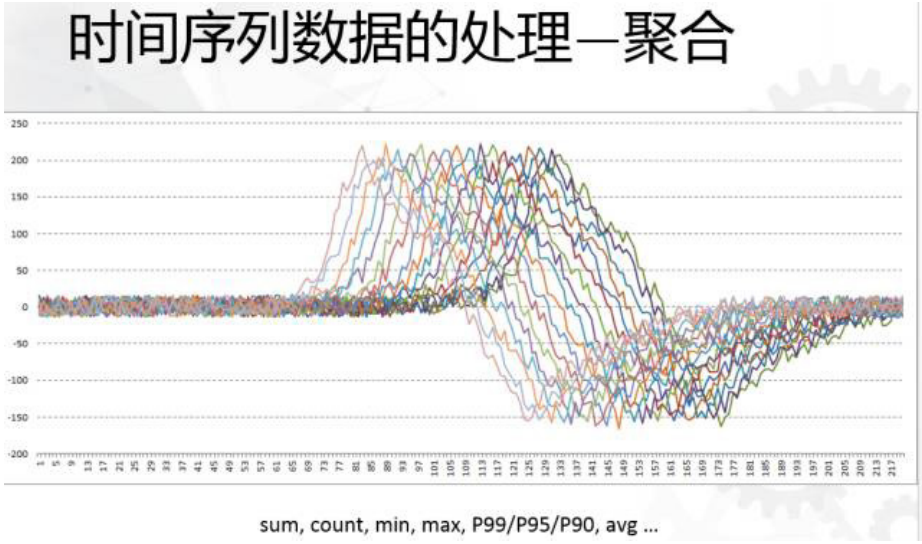
每一个时间点上都有一行，这就是多值的模型。

还有一种模型是单值的模型，单值的模型我们是把它测量的精确到时间序列上，也就在时间序列的每个时间点上只有一个值，所以是个单值，也就是说对于多值模型来说它每一行数据对应的是一个数据源，对于单值模型来说它对应的是一个时间序列，实际上多值模型对应的是一个数据源在一个时间点上就会产生一行数据，而在单值模型里一个数据源上面的每一个指标会产生一行数据。



下面牵扯到时间序列处理的一些比较特别的东西，这在我们的传统数据库里有可能是没有的，叫插值和降精度，刚才我们已经看到，时间序列会分布在一些时间线上，数据源和测量指标确定了的话，时间序列是随着时间轴往后分布的，实际上它的采样在一个典型的场景里是固定时间间隔的，它中间一些点做处理会牵扯到插值和降精处理。比如说中间丢失了一个点，比较简单的方法是中间插一个值，常用的方法是线性插值，就是在时间轴上画一个直线中间的点就插出来了。

另一个叫降精度，例如我们有个按秒采样的时间序列，显示时间范围是一年的数据，为了便于查看，需要把时间精度降到一天。比如我们只选这一天中的最大值或者最小值或者平均值，作为这一天的气温，也就是最高气温，最低气温和平均气温的概念。用算法或者把时序数据转换成精度比较低的时间序列以便于观察和理解它，这是在传统数据库里没有的一种方式。



这个东西就跟传统的数据库有点像但是聚合方式不一样，比如这里有很多时间线，实际上时序数据的聚合是在时间线的维度上的，而不是按点的，我们是在处理平时处理的空间聚合的话，一般是把很多数据点按照一个个聚合起来，而实际数据处理的时候一般会把它抽象的点连成线就是刚才看的时间序列，每个数据源在一个测量值上会产生一行时间线，加上时间序列，如果是根据某一个维度上的测量的话，在同一维度就能调成线就把时间序列处理出来了。

基本上插值，降精度，聚合就是时序数据处理的最常用的方式。

我们再看看特点，如果我们只是做刚才那些东西的话其实会很简单，但是再结合了时间序列数据的特点以后，有一些简单的事情就会变的很复杂。首先第一是时间序列会持续的产生大量的数据，持续的产生什么意思呢？因为我们往往对时间序列来说是定时采样功能，比如我们说气温的波动，每秒测量一次，一天是 86400 秒，如果是我们做系统监控，或者像气温这样的科学仪器持续的调数据的话，24 小时都要用，平均每一个仪器仪表在一个时间点上产生一个数据点，一个仪表就产生 86400 个数据，如果把全国各个县都布一个采样点，那一天数据就上亿了，实际上大家作为气象采样来说每一个县对应一个温度传感器显然有点不够的，可能我们是每一个街道甚至每个小区都有这样的传感器，那么这个数据加起来实际上是一个非常惊人的数字。

还有另外一个东西是数据产生率是平稳的，没有明显的波峰波谷。全国布满了都是传感器，这些传感器只要是不坏的是持续传数据的，所以我们无法期望它会像我们的交易系统一样每年双十一有一个明显的峰值，平时就没什么事了，很少有这样的情况，这个特性对我们做优化来说有优势也有劣势，优势就是我不需要考虑太多的削峰填谷，不需要为突发的大流量准备太多的资源。但不利的地方是，我们有些算法是很难在里头腾出时间来做一些整理的工作的。举个例子，如果把数据存在一个像 LSM Tree 或者 SSTable 一样的东西的话，当 compaction 发生，希望把两个块合并起来的时候，这时候在持续数据这个场景下上面我们写数据可以是很高很高的数据来写，如果底下的 IO 太厉害的话就会产生雪崩的效应导致写入延迟或者失败，需要做一些限流的处理。

还有一个特点是近期数据的关注度更高。时间越久远的数据就越少被访问，甚至有时候就不再需要。

看到这里我们会发现实际上时序数据我们可以用一个单表把它建立起来，我们通过在每一行数据上加很多标签来把这行给挑出来。所以我们展示的时候往往就是对标签做聚合计算。

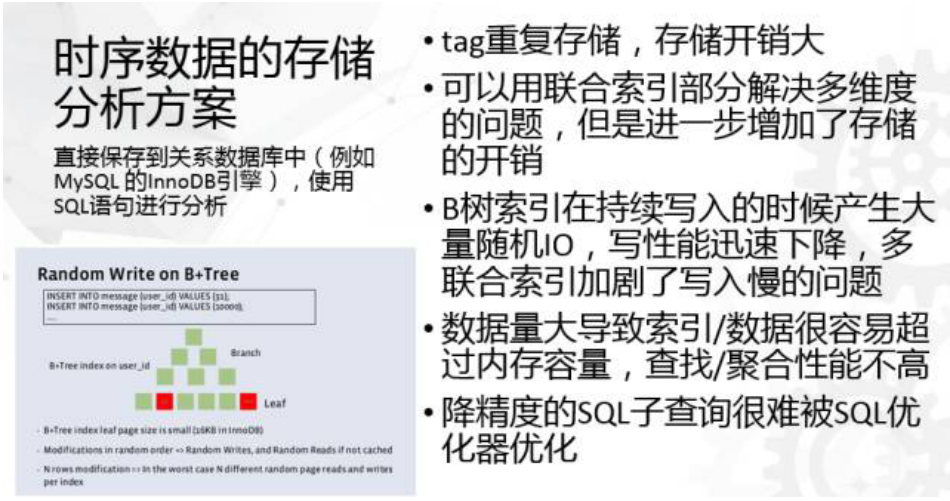
阿里时序数据场景及解决方案演进历程

首先举个例子，阿里巴巴内部有一个系统叫鹰眼，这个系统实际上是做一个机器的，简单来说阿里内部有很多服务器组成的一系列很大很大的集群，因为机器太多了，所以永远会有失败的时候，要么是程序失败要么是鹰眼失败，所以会有一个系统把这些所有的机器应用还有链路这些接入起来，时序数据也是其中的一部分，因为它需要用这些时序数据来监控整个系统的状况，其中包括里头的服务，所有服务器的指标，比如说每个服务器的 CPU 利用率，内存使用率，还包括服务器上所有应用的指标，比如说对应用 QPS 精确到每一个 KPI，每一秒被调用了多少次等等。这些东西会产生很多的时序数据，写入的峰值是 570 万点 / 秒，平均写出来 300 万，虽然说时序数据一般来说会比较平的，但是还是会有些波动。

因为在阿里内部系统请求太多，会产生服务降级的情况，服务降级也就是说当一个集群发现它的服务能力不足以支撑的时候，它会选择另外一种消耗更少的方式，比如原来是用一秒钟采样的，如果发生服务降级了是 5 秒钟采样或者 1 分钟采样，会集中在分钟的边界发送，导致在那一小段时间内有一个峰值。因为管理的机器很多，所以会产生很多的指标。

比如说对于服务器监控会有处理器、会有 IO，对于应用的监控有 QPS，不同的指标加起来有上万个。更大的指标是因为有很多应用，每个服务器上面可能有几十个应用，所以加起来几千万个时间序列，每个时间序列平均有 5 个维度，一个时间序列把几万台机器挑出来，举个例子说，我们部署的机房，在这个机房里头内部的 IP 系统上，机房、哪个机架、IP 是多少、在上面哪个应用以及这个应用里哪个具体的指标，是 QPS 还是 UV 之类的那些东西，大概平均 5 个维度就能把一个时间序列挑出来，所以其实总的维度数量并不是很多，但是每个维度里头它的离散度很大，比如光是 IP 这个离散度就有几十万。因为这个鹰眼系统是内部的系统，使用者是内部的管理人和程序员，每秒大概聚合几百次，这个场景是我们内部比较完整的时间序列的场景。

接下来会谈一谈我们自己怎么样在这样的场景里做存储分析。

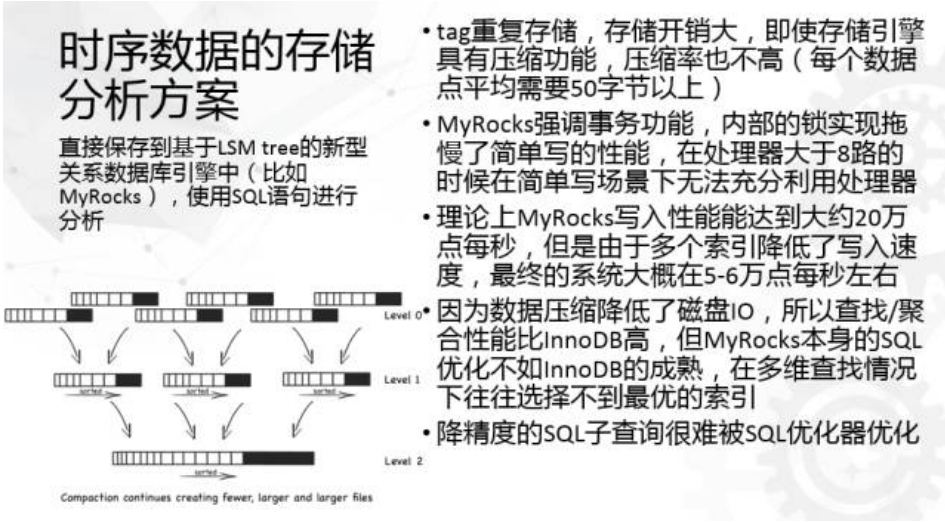


我们内部项目在做时序数据的存储时，最早思考的是把它保存在关系数据库里，这是一个很通常的做法，但是后来发现这个事情可能不太可行，因为大家都知道 InnoDB 的写入性能是很有限的，我们在一个内部测试大概在 24 台机器上，因为我们优化很好，而且是存储设备是 SSD 硬盘，写一秒钟持续写成达到两万左右，和刚才说需要的 570 万还差的很多，如果我们达到这个存储量级大概需要 300 台，其实出现这种情况有原因的，首先一个很大的问题是说，B 树的索引，索引是一个 B 树，这个 B 树有很大的开销，虽然我们可以通过一些办法优化，但是因为我们为了优化时序数据是一个多维数据，我们为了优化所有排列组合的产品，所以我们是很多多列的索引，这些索引每次在写的时候每个都需要更新，所以就会导致很多的 IO。

还有一个更糟糕的问题，我们发现存储空间增长的非常快，就算每秒 300 万的数据，每个数据加起来要 240 字节以上，300 多万 ×200 个字节，也就是说我们一秒钟一个 G，这样的话即使很多机器也会被这些数据塞满了，而这还没算上索引。类似于上述提到的时序数据的存储，如果用这个方案的话只能在一些很小的场景上使用，比如接入几十台机器，上面有很少的应用可能一秒钟只会写个几百条数据这样子，这样的数据量可以用 innoDB 来做。

另外我们还发现，刚才提到时序数据很重要的是降精度，降精度其实特别难优化，因为降精度是在时间序列维度上做的，首先要把时间序列维度拿出来然后在中间

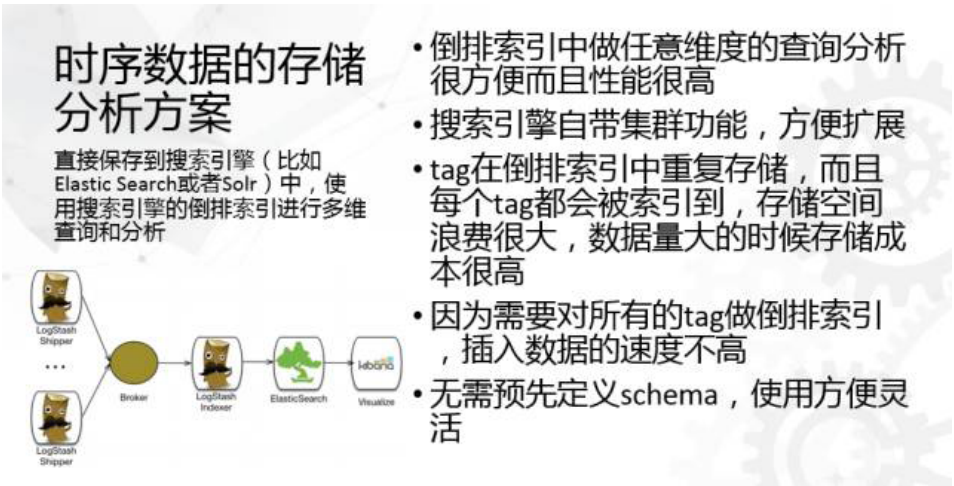
插值，而实际上 SQL 是不知道这件事情的，SQL 是按点来操作的。所以如果要做降精度的话需要用一个值查询把那条时间序列单挑出来，插好值之后才能做时间序列之间的聚合，这意味着我们的服务和 SQL 服务器之间的吞吐量非常非常之大的，相当于 SQL 只是一个数据通道需要把所有值都拉出来运算一遍。很难把这个东西放到 SQL 服务器去。所以我们发现把时序数据放在关系数据库上的方案不可行，就考虑了另一个方案。



一开始最大的问题是写的慢，那我们就找个写的快的东西来处理好了，写的快的东西是什么呢？就类似于 SStable 那种，就不用那种随机操作的方式，就用追加的方式来写，实际上这个东西在写上面是能工作的，因为我们测试了 Google 的 LevelDB 和 MyRocks，我们发现 Rocks 的性能比较强，所以我们在 Rocks 上测试了一下，大概能达到 20 万点 / 秒，而且这是因为 MyRocks 写入性能的优化不够，它在 CPU 的核数多于 8 核的时候可以共用 CPU，线程所用比较保守，阿里内部有一个改造，把它改造成写入性更高的东西。即使是这样性能也不是太高，但是勉强够了，因为能达到 20 万点 / 秒，不需要用 300 台机器了，用 30 台机器就可以搞定。

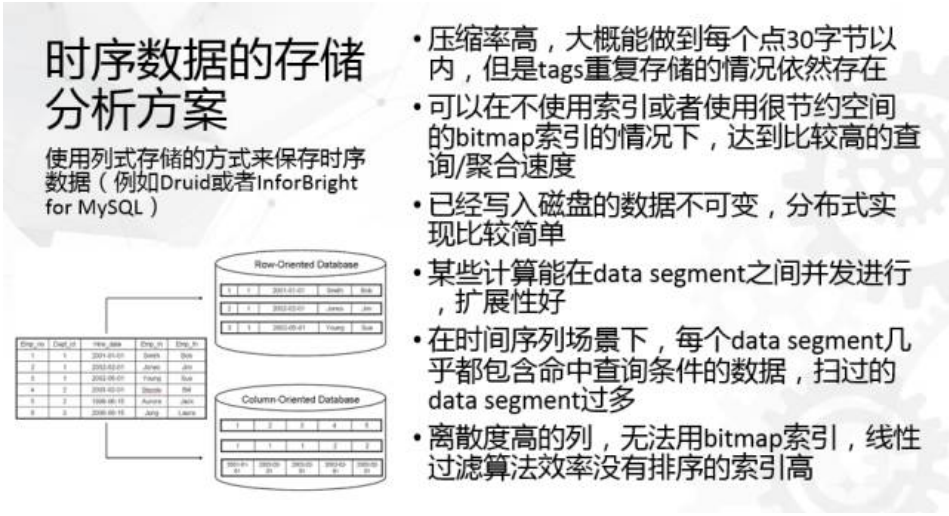
但是我们很快发现其实没有那么乐观，因为又回到刚才那个问题的，我们需要建很多个索引来保证多维这个事情，如果我们需要建很多个索引的话意味着我们每次写

实际上要去更新一个索引，比如为了保证多维查询的性能需要建 4–5 个索引，等于写入数据调到原来的 1/4。所以这个东西也不太可行，而且它有一个问题是 tag 重复存储其实没有解决，我们压缩完以后平均 5 个点还需要 50 个字节，实际上一个点真正的数据只有 8 个字节，加上时间就是 16 个字节，差别还是蛮大的。



下面介绍的是在业界经常被人用的方案 --Elastic search，这个东西是比较有好处的，数据量不大的时候，会针对每个纬度来做索引，能很快的把时间点摘取出来。它的倒排索引很大，但是这个方案特别流行因为对于很多公司规模小、客观业务规模小的，这个东西会非常有戏，因为它很快，而且有整个开源社区的支持。

我们后来还试了用列式存储的方式保存时序数据，这是特别有诱惑力的一个方案，因为列式存储第一压缩率会比行式高很多，因为它把相似的数据都放在一起了，而且它有一点特别适合时序数据的是因为写入磁盘的数据是不可变的，时序数据恰好不太需要修改。但是后来我们发现使用了以后踩了个坑，Druid 或者 inforbright 那种方案处理某些时序场景合适，但是处理我们那个时序场景不太合适，因为列式存储是把导入的数据累积到一定程度，才会打一个包把它固定到磁盘上的，但是时序数据如果长时间的查询的话这意味着要查该时间段内每一个包。



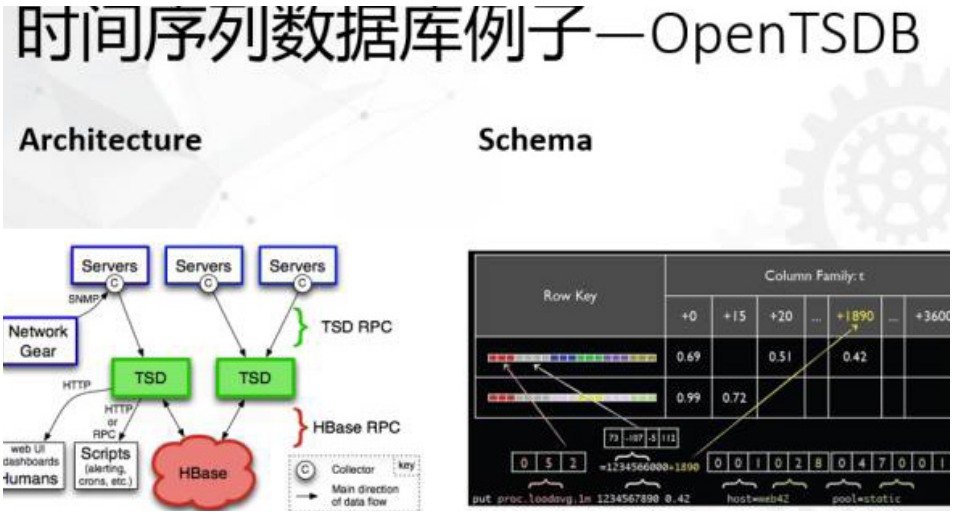
因为所有维度的数据在每个包里都会存在，比如要按机架的维度来看，我们积累了一万个数据文件，每一个数据文件里头都有非常大的，可能会出现同一个机架的两三行数据。这就很糟糕，实际上列式存储的筛选数据文件机制没有生效，没办法迅速的把一部分数据文件剔除掉。

接下来我们还考虑了这样一个工具，流引擎，其实流引擎是一个非常好的东西，

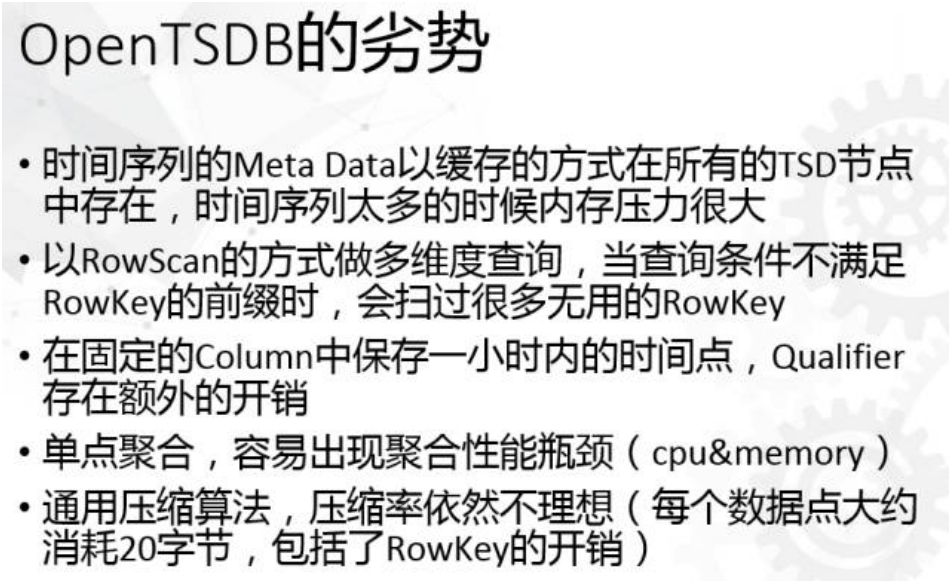
它同时解决了多维计算，反正你能想象的东西它基本上都能解决，除了存储问题，流引擎不是数据存储引擎只是计算引擎，如果事先你的应用没有对你需要查的数据做很好的规划的话，只能事后再补充一个新的拓扑再计算，但是新拓扑无法立即获得数据，比如上一个新的拓扑，是按 24 小时精度的，一次要看 24 小时，那么 24 小时后才能拿到这个数据，在某些场景下面还是很希望把数据保存下来察看的，但是流引擎做不到这一点。流引擎能做到的是能以很高的效率来处理数据，但是提前要把需求预先定义好，没有办法实时计算。

HiTSDB 的由来

所以我们后来还是转到了使用一个专门的时间序列数据库的方案上，业界往往把 MongoDB 之类的东西也算成时序数据，其实那些东西是比较通用的。专业的时序数据库可能是 InfluxDB，openTSDB 这样的，这两个东西最大的区别是它能把时间线提取出来，它的思路相当于是事先做一系列的标签、先找到时间序列，在这个时间序列上再找到对应的点。时序数据是按照一个时间长度分片来存在一起，所以这样的好处就是它存储的压缩率会比较高，而且是搜索的时候不需要搜索每一行了，搜索的比较少。



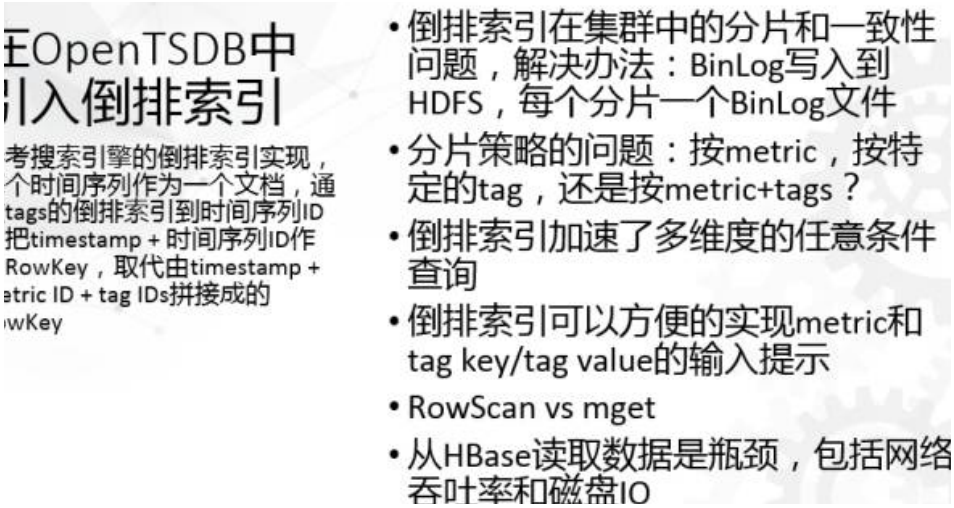
最后我们选了 openTSDB，可以保证它把一个小时的数据放到一个行里，这样压缩率比较高，能做到每个数据 20 字节左右。



但是 openTSDB 用久了发现它有很多劣势：首先，它其实是一个无状态的节点，所以它的 Meta DATA 实际上在所有节点上都是全量的，所以它占用的内存会很大；

其次，时间线数量很多的时候，Rowscan 方式做维度查询，这跟列存数据库有点像，但是当查询条件不满足 rowkey 的前缀时，它的磁盘 IO 还是有点太多的；第三，在固定的 Column 中保存一小时的时间点，这个问题是，大家知道它的 qualifier 存在额外的开销；第四，还有个更大的问题是 openTSDB 是单点聚合，也说不管你有多少节点，实际的计算，每次计算都放在一个节点上。

所以我们后来做了很多改进，其中第一个是先引入了倒排索引，我们发现用倒排索引的方式能更快的把时间线挑出来，搜索引擎的问题在于它挑的不是时间线，挑的是所有的数据，所以索引就会很大的倒排，如果我们把这个索引指定在时间线上，时间线是几千万的量级，对于倒排索引来说是很轻松的事情，所以我们引入了倒排索引。



但是引入了倒排索引以后还有很多没有解决的问题，而且有很多新问题，比如说一旦引入倒排索引以后，我们为了保证让它能按一致的方式工作，我们其实做了分片，我们用了很多办法，比如 binlog 写入到 HDFS，保证集群的可用性及数据的可靠性。每个分片一个 binlog 文件还有分片策略的问题，现在简单的说是按照 metric 加特定的 tag，等于是用了一个结合的方式。

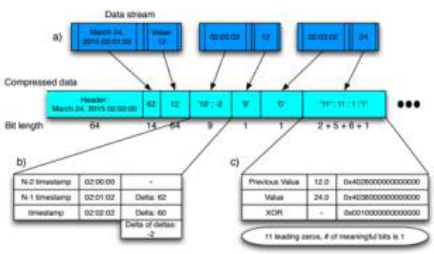
OpenTSDB中的预降精度功能

- 数据老化 vs 预降精度
- 预降精度的级别和额外空间开销
- 预降精度和实时降精度结合
- 平均值带来的问题
- 精确计算 vs 概略计算，在预降精度数据上统计P99
- 时间窗口和数据修改

之后我们再继续往下做优化，虽然引入了倒排索引，但性能有所提升有限，因为HBase mget 的一些性能限制。中间还发现一个问题，是 openTSDB 的降精度，它保存的永远是原始数据，所以我们又做了预先降精度的功能，把预先降精度的数据用一个东西算好存进去，这样比如我看一小时的时候，预存有半小时了，这样放大只有2 而不是原来的 3600，这样改动虽然比较小，但是性能提升很大。

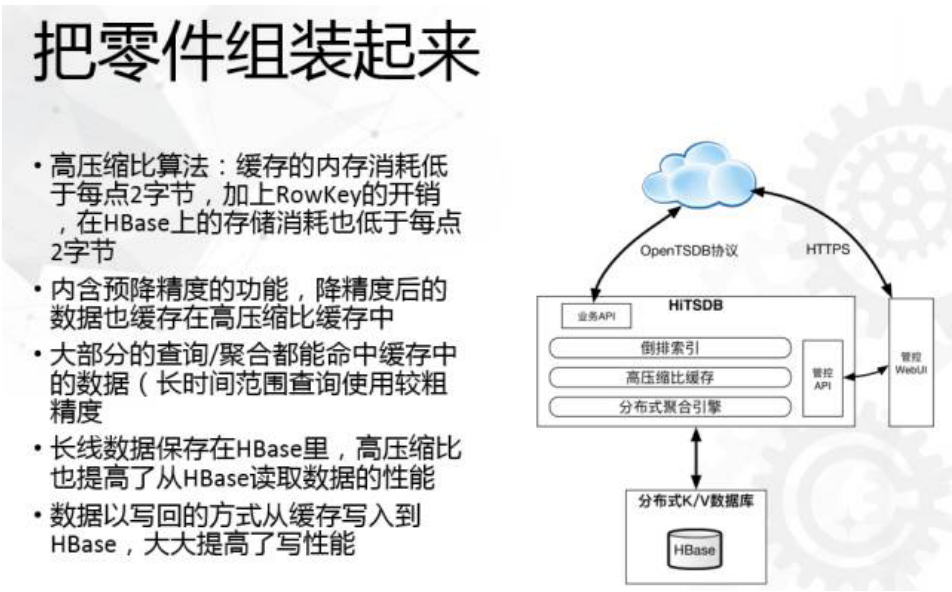
Facebook Gorilla带来的新思路

- 高压缩比算法：平均每个时间点压缩到1.37字节
- timestamp采用delta-delta压缩，value采用二进制xor压缩
- 高压缩比使得最近一段时间（若干小时）的数据可以完全缓存在内存里，查询的时候避免了HBase的mget操作
- 解压缩速度很快，而且降精度可以在解压的过程中同时处理，减少内存的开销



回到刚才那个问题，FaceBook 有个东西叫 Gorilla，它是个高压缩比的算法，它最厉害的是可以把一个时间点压缩到 1.37 字节，在我们的测试中基本上可以做到 2 字节以内，这么高压缩比有什么好处？意味着我们可以把最近的数据，也就是说我们用倒排索引找到的一系列时间线 ID 以后，大部分只要去一个内存的表里头把对应的压缩好的数据块找出来就行了，因为一个时间点压缩到不到两个字节的话，意味着哪怕是每秒 300 多万的点也就 5、6 兆一秒就解决了，通过 256G 内存的机器，这个东西其实提高非常大（不需要去 HBase 做 mget 操作）。

这个和倒排索引结合起来，其实就满足了我们的读写了，但是这个东西带来了一个更大的问题，是写穿还是写回的问题，如果你是一个写穿的方式的话，写性能是并没有提高的，也就是说我们并不能这样写，如果我们要做的更激进一点写回系统，一次性写到后面的存储，这样分片和高可用方案就做的很复杂。最后我们还是搞了点小技巧，把共享文件系统上的 binlog 写入 LDFS，即使这样还是很复杂。



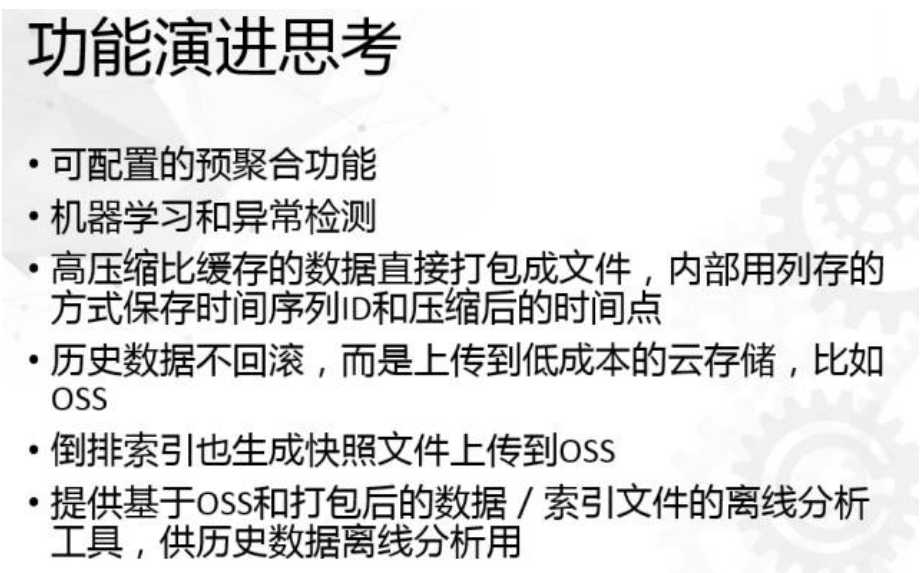
大概再提一下，因为我们做了高可用的工作，但是不管怎么样在时序数据库这个领域里头它并不能保证 ACID 的，传统数据库是有 ACID 的保证，实际上时序数据库里只能保证一条数据写过来至少被处理一次，尤其是 batch 写的方式，比如客户一次

性提交 300 条数据过来，第 150 条失败了，实际上前面 149 条已经写入数据库了，这个事情在我们这儿是被允许的，前面的 149 等于是处理了两遍，这样的话至少每条数据会被处理一次。

然后我们还引入了一个分布式聚合引擎，解决了 openTSDB 的那个单点的问题，然后把零件组装起来就形成了最后的一个产品，我们叫做 HiTSDB，它协议上跟 openTSDB 是兼容的，但是内部已经被我们改的面目全非了。它的主要核心的功能就是倒排索引，缓存还有分布式聚合，最核心的是倒排和缓存，有了倒排和缓存我们就能以很高的速度来处理一个典型的，在最近时间内典型的一个时间序列的查询。

功能演进的思考及不足

因为时序数据刚才跟流引擎对比了一下，如果用流处理的话在写入之前就把所有的东西写进去了，它的缺点是不具备灵活性，优点是很多场景下速度非常快，而时序数据的想法是做后计算，把所有的原始数据都写进去，想怎么算怎么算，想怎么查怎么查，但是对于一些特别大的查询、又特别固定的查询，反复的计算是没有必要的，所以下一步会把一些可以配置的预聚合功能放到数据库里，实际上当你往外查的时候某些东西是已经算好的，你只要把它查出来就好了。



我们还有一些想法是把历史数据的文件存在云存储上，这样我们可以做长线离线分析，这都是我们考虑的事情。

实际上我们还是有很多场景不能很好的应付的，这个我们在内部也发现了一些：

1. 发散时间序列，跟我们的搜索团队有过合作的项目，做离线分析的会把他们的指标数据放在压缩上，离线分布会产生几个小时的数据，时序数据会膨胀为几十几百亿，这个东西目前我们是不能很好的应付的。

2. 还有一个是事件驱动 vs 定时采样，我刚才说的都是定时采样的，对于高压缩采样是固定 20 分钟切一片然后把它压缩起来放进去，但是如果是事件驱动的 20 分钟可能没有几个点，有时候 20 分钟也可能非常多的点，这样对于压缩很不均衡；

3. 还有一个目前解决不了的是高频采样，我们现在的采样精度最多支持到秒，时间精度是支持到毫秒，但采样精度只支持到秒；

4. 还有一个问题是，虽然时序数据是单表，但是很多用户希望和现在的 SQL 数据表互操作，目前这些问题还是都没有支持的，所以未来我们会考虑做成一个存储引擎。

5. 还有一个是 group by+topN 的优化；

6. 结合事件驱动和定时采样，考虑引进一些列存的思路解决数据驱动的模型，考虑双引擎（一个处理事件驱动一个处理定时采样）。

再说一个比较炫酷一点的事情是硬件加速，最近阿里在搞关于硬件加速的东西，其中有些类似于 FPGA，因为 FPGA 会用一个流式的方式工作，FPGA 下面会带一个万兆或者 40GE 的网卡，特别适合时间序列场景的类似流架构的方式，所以我们考虑采用 FPGA 的方式构建下一步硬件加速体系，如果这样的话我们有可能做到在一个板卡上做限速，也就是说数据就是以 40G 的速度流进来，一个固定的数据速率流出。最后稍微聊一下云部署的事情，这些东西最后会上云，提供公共服务。

最后谢谢一下我们团队——阿里中间件时间序列团队，我们的口号是 For Your Time。不浪费大家的时间了。谢谢！

运维

超全总结 | 阿里如何应对电商故障？神秘演练细节曝光

中亭

近日，在 QCon 北京 2017 大会上，来自阿里巴巴中间件团队的技术专家周洋（花名中亭）发表了题为《阿里电商故障治理和故障演练实践》专题演讲。在会后官方组织的评选中，本次演讲的内容得到了一致好评，中亭获选为本次大会的明星讲师。此次演讲整体上分享了从 2011 年至今，阿里巴巴电商平台遇到的诸多有代表性的故障，以及在此过程中积累的非常多的高可用保障经验和解决方案。

演讲内容包括两大部分：第一部分会从分布式系统经典依赖故障出发，剖析故障成因，介绍治理方案和技术演进；第二部分从宏观视角探讨构建一套”防故障”设施的重要性，并探讨如何设计一套故障演练系统，介绍故障演练的基本原则和经验总结。

演讲全文：

大家好，今天来的人不少，可见对于故障耿耿于怀的人，不止我自己。今天分享的内容主要还是围绕故障治理有关。众所周知，故障治理本身就是一个比较大的话题，几乎涉及到运维、研发、故障运行管理的全部岗位，奇葩一点的故障还可能涉及到运营和产品经理。聊到故障的苦与泪，相信 45 分钟绝对连开头都没讲完。今天的分享，主要还是回归故障发生的本质，故障原因角度切入。看是否有一些方法论和通用性的手段可以沉淀出来。希望可以对大家有所帮助。

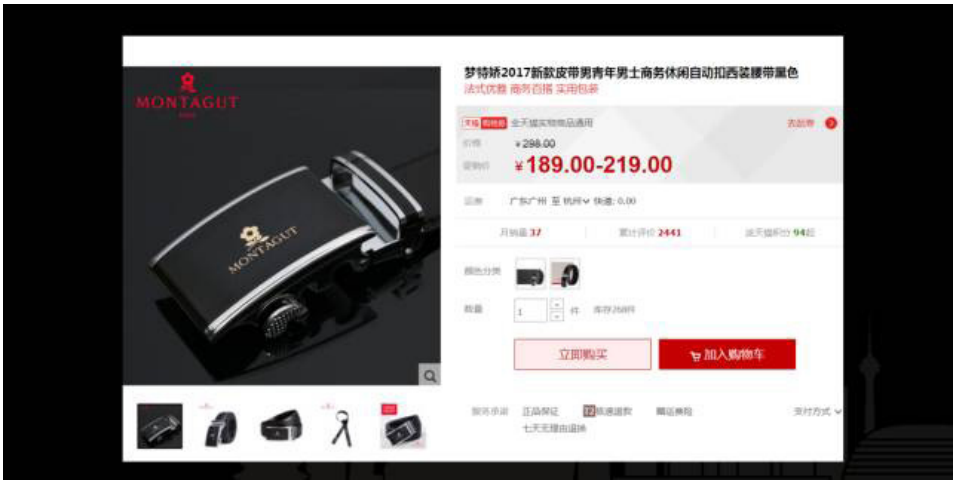
今天演讲的主要包括两个部分：第一部分会从分布式系统经典依赖故障出发，剖析故障成因，介绍治理方案和技术演进；第二部分从宏观视角探讨构建一套”防故障”设施的重要性，并探讨如何设计一套故障演练系统，介绍故障演练的基本原则和最佳经验。考虑演讲时间和内容关联度，之前报给大会提纲中的”通过灰度发布提前发现故障”的章节被隐去，有兴趣的同学可以会后找我单独交流。

首先介绍一下我自己，姓名周洋，花名中亭。2011 年加入阿里接触稳定性技术领域，开始做一些稳定性产品的研发，同时也会承担一些架构演进的推进工作，比如 HTTPS 改造，电商交易链路升配等。2015 年开始搞双 11 大促，做为共享事业部的大促负责人，保障了双 11 的稳定。也获得双 11 老 A 也就是双 11 特种兵的称号。

共享事业部对于在座各位可能比较陌生。如果我换一个说法，商品、交易、会员、优惠、评价、中间件，大家应该就都知道了，这是双 11 当天最具挑战的链条之一。 右边是中间件核心作战室成员，在过了双 11 业务高峰后的一张合影。2016 年至今，工作的重点在常态稳定性的确定性方面，今天的分享也是主要围绕这部分内容。

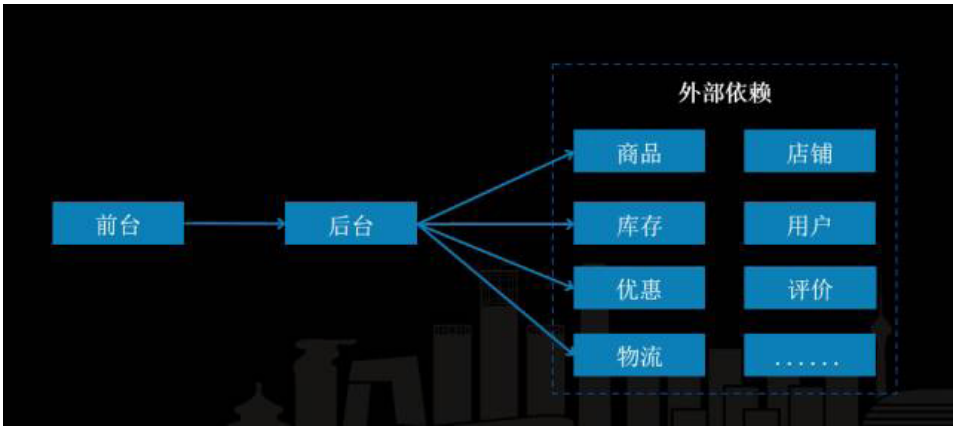
分布式系统常见依赖故障治理及技术演进

首先抛一个问题，什么情况下你会认为淘宝网挂了？ 我相信关注这个问题的人很多，不过能给出确切答案的人并不多。因为这个看似简单的问题，真要回答起来好像也不是那么容易。今天的分享，我先试着给大家回答一下这个问题。



让我们从一张“简单”的页面说起。这张页面叫做商品详情页，相信在座的各位对这张页面不会陌生，因为对于大部分人来讲，这张页面是他们在淘宝完成一笔订单的第一步。而商品详情页的使命就是把商品的信息没有保留的展示给大家，引起大家

的兴趣，引导大家完成购买或是收藏。从信息展示的角度来讲，商品详情页确实是一张非常简单的页面。

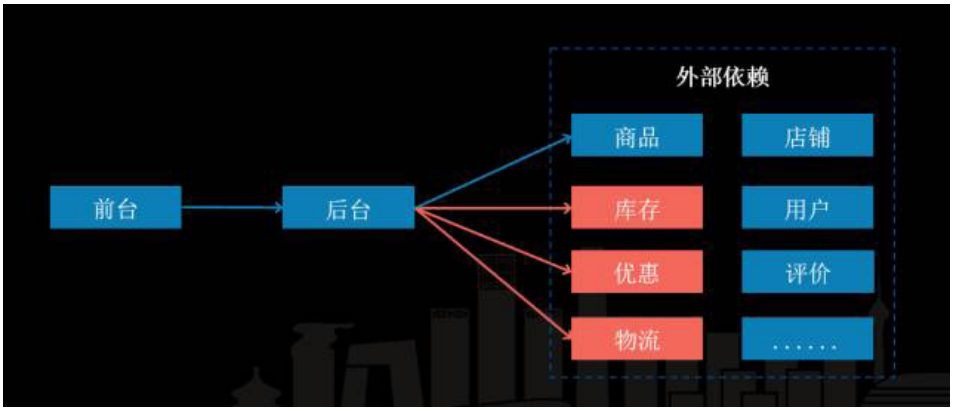


我们再来看一下商品详情页应用的后台架构。商品详情页是阿里最早实现静态化应用之一。那些与浏览者无关信息，比如商品标题、图片信息、销售属性组合等信息均直接进入缓存，其他和用户相关的，如优惠、库存、物流、服务等动态信息则通过异步调用方式填充至静态化后的页面框架内。为了在一张页面展示足够多可供决策信息，撩起用户的购买欲望，详情后台必须去依赖非常多的服务应用，聚合足够多的信息。 少则几十，多则成百。从这个角度来讲，商品详情页面又是阿里依赖最复杂的应用之一。

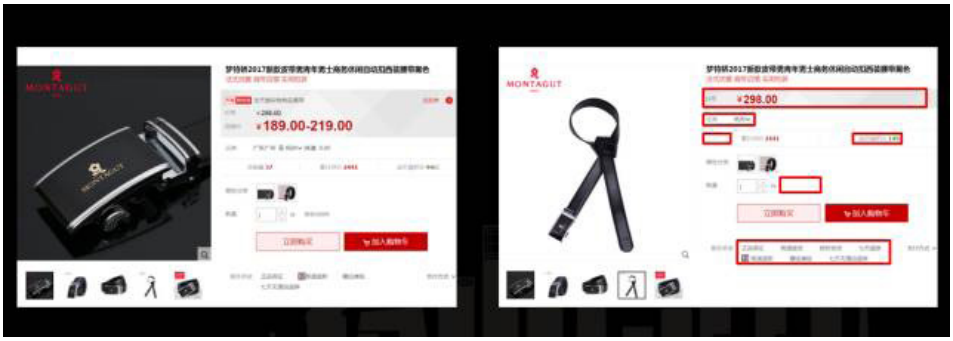
互联网业务的一个主要特点是，业务迭代非常快，每天有新需求，每周都有新发布，每年都有大重构，每一次变化都有可能导致状况的发生。越是贴近用户的系统，受下游服务影响越大。那么我们不仅好奇，对于详情这个阿里最复杂的应用，下游发生一些状况时，系统会变成怎样？我们通过两个实验来观察一下：

实验一：假设后端的优惠、库存、物流发生故障，我们来观察一下商品详情页的表现。

乍一看，好像没什么问题。只是觉得页面清爽了一些。或许在这个信息过暴的时代，看着这么清新脱俗的页面，还有一点点暗爽。



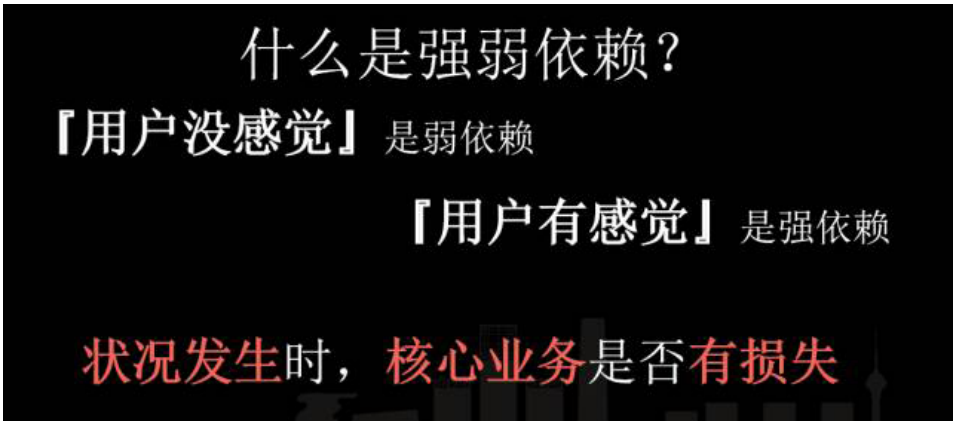
在现场做了两个调查，观察大家对实验一的反映。调查 1 是请认为详情页故障了的同学请举手。结果是现场没有人举手（也可能是现场氛围还比较冷）；调查 2 是请大家来找茬，前后两个详情页有多少处不同？这次有一个妹子说出了正确的答案（同时也向妹子赠送了电子工业出版社出版的讲述阿里双 11 技术演进的《尽在双 11》书籍）。



没有对比就没有伤害，一共有 6 处不同。从功能角度，这铁定是一个故障页面。不过从用户体验和业务角度讲，少了这些信息也不影响商品购买，不影响核心用户体验。好像又是没故障？有点纠结，对吧？ 您先纠结一会儿，我们来进行第二个实验。

实验二：当商品详情的”商品”出了问题，商品详情会怎样?

详情还是那个详情，只不过是商品详情变成了错误详情。第一张页面：很抱歉，你查看的商品找不到了。有可能是你访问的方式不对，比如 URL 上面少了一些参数，也可能是后台真的出问题，对于用户还算是比较温柔的一种方式。第二张页面：很可能就是网站真的出问题了。比较可能的原因是后台没有合理的处理超时导致前端异常。不过说实话，这个页面我也非常少的见到。如果真的出现了，那基本就是一次非常严重的事故。



通过上面的两个实验，相信大家应该对于我们今天要介绍的一个概念”强弱依赖”有一些模糊的感觉了。 从感性的角度来讲，就是当下游依赖服务出现问题时，当前系统会受到一些影响，让用户有感觉的是强依赖，没感觉的是弱依赖。不过这种定义不够严谨，因为总不能说没有用户访问时，就不算故障吧。所以也需要从理性角度

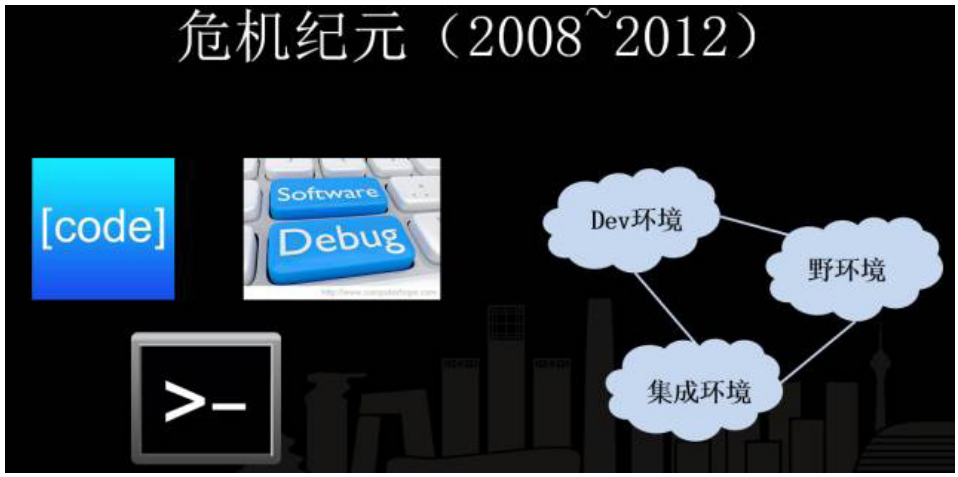
定义一下：首先应该发生状况，其次应该是核心业务，最后是否带来损失。不影响核心业务流程，不影响系统可用性的依赖都可以叫做弱依赖，反之就是强依赖。

终于解释清楚什么是强弱依赖，那么做好强弱依赖治理到底有什么意义？抛开依赖模型来看强弱，意义不大。严谨的依赖模型应该包括关系、流量、强弱三个组成部分。依赖关系定义依赖的方向，我依赖谁，谁依赖我。流量定义着每个应用、服务、方法调用的次数，强弱则定义着依赖的松紧程度。依赖治理就是通过科学的手段持续稳定地拿到关系、流量、强弱的数据。强弱依赖主要可以被应用到下面的场景：

- 系统改造验收：对于分布式系统，至少应该做到运行态中不会因为我依赖的系统出现故障，而引起当前应用出现可用性的问题，比如进程挂掉，频繁FullGC，负载飙高等，何时何地都具备快速止血的能力。
- 限流降级参考：对于弱依赖，一般都要配置限流或是自动降级策略，比起通过拍脑袋或是经验值来设定，倒不如通过实际的故障测试来进行微调，比如对于下游出现超时情况，就可以通过实验得出基于线程池限流到底要填写多少数值。
- 应用启动顺序：理想情况下，应用启动更应该做到 0 强依赖启动。不过有一些情况无法做到。因此应用启动的依赖顺序也需要实时关注。特别是新 IDC、机房建站时，那个蜘蛛网一样的依赖关系，最好是通过系统方式获得。

- 故障根源定位：后台系统的故障，往往通过上一层的业务故障表现出来。故障处理讲究的是争分夺秒，良好的强弱依赖，对于系统自动化诊断有非常大的助力作用。
- 依赖容量评估：正常调用链路下系统容量需要评估，当某个弱依赖挂掉时，整体的容量是否有变化。

说完背景，终于可以聊一下强弱依赖的技术实现。在阿里，强弱依赖的技术演进整体上分了 3 个阶段，每个阶段的方案的诞生都有其独特的时代背景和业务难点。现在回头看来，也可以看到当时技术的局限性和突破。

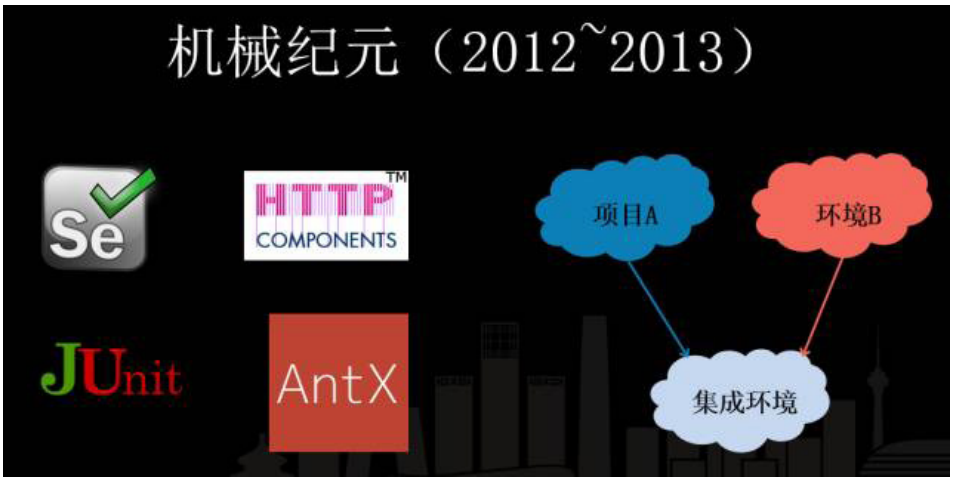


熟悉淘宝技术发展史的同学都知道，2008 年阿里刚刚完成一个代号为五彩石的项目，完成从巨石系统向服务化系统的改造。业务和开发模式上有了较大的发展，不过网状的依赖关系也带来了非常多的问题。这个纪元的主要特点是：故障频发，技术思路和方法都是以结果为导向，糙一点、结果精度差一点可以忍受。

模拟依赖故障技术上有三招，改代码 + 发布，远程 Debug+ 重启，登陆机器去执行一些 shell 命令操作。 好处是灵活随意，可以一定程度达到效果；坏处是成本高，影响环境稳定，你测试的时候其他人处于无法工作状态，影响效率。此外，这个阶段，因为分布式链路追踪技术还没起步，所以模拟依赖故障时，经常会漏掉一些主机或某些服务。故障的粒度也比较粗，对于一些 Linux 的命令，都是主机级别的。

阿里内部有一套日常环境，主要做上线前的集成测试。为了尽量减少对环境的影响。我们通过修改服务版本的方式，形成一个独立的测试环境。记得 11 年下半年，我开始做第一版的时候，我搭了淘宝 12 个核心应用的日常环境，踩坑无数，纯体力活，也算前无古人，后无来者了。

通过这套环境跑了几次结果，发给核心的业务 TL，大家很兴奋，貌似找到一条治理的路子。不过很快暴露了新问题，比如环境的运维归属问题，开发机器的干扰问题，以及对于业务的了解程度和测试粒度问题，所以在很长一段时间测试的范围都局限在交易核心链路。



第二个阶段的核心就是提效，从第一个阶段的痛点入手，解决人的成本和环境的问题。这个阶段之后，基本可以摆脱手工方式，效率上有大幅度提升。

这个阶段也引入了一些测试技术，其中用的比较多的是 Selenium，通过这种技术可以提前录制用户行为并转化为测试脚本，并且每一个步骤都可以截图记录，方便问题复查。

在这个时期，阿里中间件的技术有一定发展，分布式追踪技术出现，可以把用户访问的链条串联起来，排查问题的效率有了一定提升。同时所有的中间件，如 Nginx、消息、分布式服务调用、分布式数据库、软负载和配置中心等都做了改造，支持用户流量的标记、追踪和路由控制。基于上述这些技术进展，环境的问题就有非

常大的突破。

在内部我们称为叫二套环境。它的核心原理是在基础环境之上，动态区分出一些小环境，他们分别是某个业务的子集。项目之间彼此独立，不会互相调用，只有当依赖的服务不在时，才会去访问基础环境的服务。数据库和缓存是公用的。

在这个阶段，我们不必再去修改代码的服务版本，每次发布后，代码的版本等能够自动化的保持一致，运维成本有所降低，野服务干扰的情况也有所缓解，人的介入非常的少。不过还是有一些问题亟待解决：首先，二套环境的路由策略是和用户绑定的，也就是说需要提前去做一些配置；其次，域名上也有一些限制，加了 second 等前缀，测试路径中 URL 等复用率低的问题没有完全解决；第三，测试的粒度仍然很粗，独占机器，规模化推广时，机器成本和用例运行的成本还是很高；第四，故障场景缺失，只存在于基础环境的服务没法模拟的故障，如：数据库故障，缓存故障等。

2014 年的时候，我们的思维方式有了比较大的突破。我们不再纠结于环境和外部手段的改进，而是回归到强弱依赖关注最核心的部分。那就是业务影响和系统设计。能否实现一种只与代码设计和业务相关，而与外部环境无关的方案呢?

这期间有两个关键思路或是推论：

推论 1：我们要的是下游依赖出现故障现象，不必真的是下游服务提供方出现故障。只要消费方感觉下游出现故障即可。从这个思路来讲，商品详情如果要做强弱依

赖测试，只要自己玩就 OK，不需要去折腾下游依赖的几十个应用。

推论 2：我们之所以需要单独搭建环境，为的就是控制故障的影响范围。那么我们可以换一下思路，就是我们只影响要发生故障的请求，其他的业务流量都放过。是不是就可以达到目的。本质上是一种对业务流量的筛查能力。

有了上面的思路，第一问题就是如何拦截用户的请求？拦截用户请求，让用户改造成本最低，没有什么地方比中间件更适合了。每个通用的远程调用接口，都是可以做文章的点，并且中间件之上的业务系统不用做任何改造。

下一个问题就是故障规则和业务识别，我们曾考虑在用户请求的入口就打上标记，置入故障规则，不过发现对于 post 请求，异步 js 请求，定时任务等都有比较大的改造成本，且有安全隐患。 所以就增加了一个服务端，直接下发故障规则到依赖插件上。故障插件通过对流量的调用拦截 + 业务识别，唯一确定影响哪一个请求，然后通过故障规则判断是注入异常还是超时，从而达到模拟故障的效果。 因为插件可扩展的设计，所以我们默认是可以同时注入多种故障场景的，同时插件也会把影响到请求的详细信息异步上报给服务端做分析。

理论上通过上述的方案，在业务流量输入方面，我们没有任何要求。无论是人的自发测试行为，还是机器的测试行为，都没有任何限制。只不过为最大限度复用已有的测试用例积累，提高自动化程度，我们设计了一套用例注解，封装了和强弱依赖服务端的通信。利用 Junit 生命周期的特点，完成故障规则的下发和清除。任何一个测试用例，20 秒之内改造成一个强弱依赖的测试用例。在结果输出方面，会详细的展示一次强弱依赖检测的过程，以及测试用例涉及到的链路的依赖变化。到此阶段，强弱依赖真正达到了一个相对里程碑的版本，2014 年开始，强弱依赖也作为双 11 必做的一个横向项目。

下面是强弱依赖注解和依赖系统的示例：

```
@Test
@AppDisasterScenes(appName = "[redacted]", priority = Priority.P1, strongDep = true)
@DisasterScenesExtensions({
    @DisasterScenesExtension(targetType = TargetType.HSF_APP,
            targetName = "[redacted]",
            delay = 5000,
            disasterType = DisasterType.DELAY),
    @DisasterScenesExtension(targetType = TargetType.TAIR_GROUP, group = "[redacted]"),
    @DisasterScenesExtension(targetType = TargetType.HSF_INTERFACE, interfaceName = "com.taobao.item.service[redacted]"),
    @DisasterScenesExtension(targetType = TargetType.HSF_INTERFACE, interfaceName = "com.taobao.item.service[redacted]"),
    @DisasterScenesExtension(
            targetType = TargetType.HSF_INTERFACE_METHOD,
            interfaceName = "com.taobao.item.service[redacted]",
            version = "1.0[redacted]",
            methodName = "query[redacted]",
            parameterTypes = {"java.util.List", "long.class"},
            exception = "java.lang.Exception")
})
public void testMix() {
    //test case
}
```

总的来说，整个强弱依赖技术演进历史，就是对数据准确性，稳定性，成本、效率的不懈追求，并在这几者之间达成一个动态平衡。

故障演练的基本原则和最佳系统设计实践

众所周知，2017 年不大太平，业界出现了很多大故障。

2017 年 3 月 1 日，弗吉尼亚州数据中心出现故障，亚马逊 S3 服务出现了较高的错误率，直接影响到成千上万个在线服务；2017 年 1 月 31 日，GibLab 同学线上数据库变更时，遇到突发了一个情况。因为操作失误，导致整个生产数据库被误删除，丢失 6 个小时的数据；

2017 年 2 月份国内的一家经常被用来测试网络连通性的友商也出现了故障，工信部迅速关注，并紧急约谈了相关公司。同时下发紧急通知要求 BAT 等各重点互联网企业吸取教训，业界一片哗然。

这时候，有一家公司显得特别淡定，那就是 Netflix。 Netflix 是一家服务全球的

在线影片租赁提供商，他的核心业务完全架设在 AWS 上面。据新闻揭露，Netflix 在亚马逊故障时可以很快的恢复正常，因为他们内部有一个”防故障”的基础设施。听起来，好像是我们需要的东西。

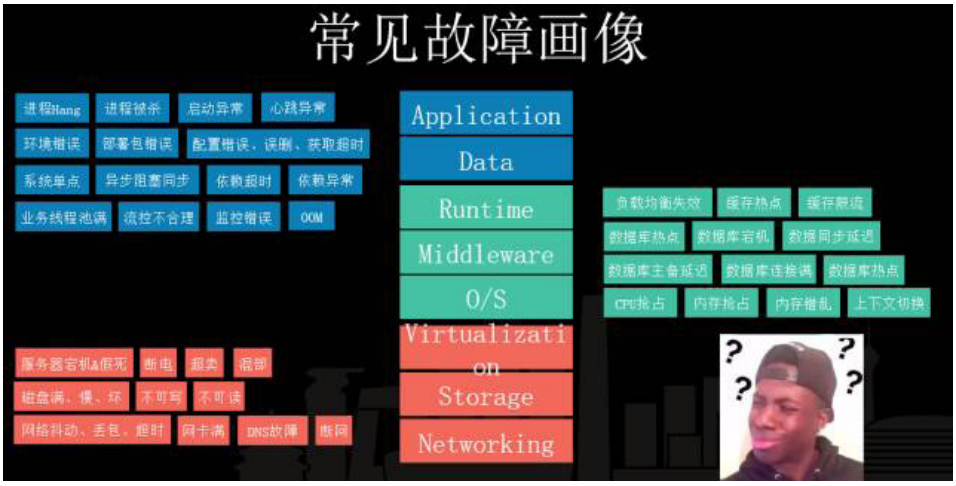
深入调查之后，发现防故障基础设施背后是一个猴子军团。

早在 2012 年，Netflix 就发布了 Chaos Monkey。用来在随机杀死实例，据官方数据指出，到目前累计杀死 65,000 个节点。他们的测试策略也比较有趣：在工作时间在生产和测试环境运行，目标测试系统的健壮性，训练后备人员，让恢复更简洁、快速、自动；Latency Monkey 的作用就是让某台机器的请求或返回变慢，观察系统的表现；Chaos Gorilla 的能力是搞挂一个机房，宏观验证业务容灾和恢复的能力。Netflix 发布猴子军团的原因是因为，他们很早就吃过云故障的亏，所以本能是认为云设施是不可靠的，必须在通过演练来验证软件层面的容灾。

古代有个哲学家说过”没有人曾经两次踏进同一条河流”，因为无论是这条河还是这个人都已不同。故障也是类似的，故障发生的时间地点，影响流量，与故障打交道的人都没法完全相同。从这个角度看，故障治理本身是一个伪命题，都是在解决过去某一个时刻的问题。不过从程序员视角（我习惯叫上帝视角），任何故障的原因都是可被定位的，避免相同原因重复引发故障，是每个程序员应该执着追求的目标。电商历史上遇到了非常多有代表性的故障，为了不让故障重复发生，阿里内部也打造了一套”防故障”的基础设施。

2015 年 5 月 27 日，因为光纤中断的问题，支付宝大规模宕机事故，公司内部得出一个结论：任何基础设施、生产系统、任何流程都可能出现问题，没有经过重大灾难验证的容灾设施都是耍流氓。 启动了代号为虎虎虎的生产突袭项目，用来验证异地多活的质量。

2012 年，完成交易的同城双活后，就启动了同城容灾演练，也叫断网演练。验证核心系统的同城一个机房挂掉的情况下，是否还可以正常工作。



2011 年，开始做强弱依赖的治理和建设，希望提前发现因为依赖问题导致的系

统故障，系统的代号是 EOS（出处是古希腊神话中的黎明女神，语意是能够把纷乱的依赖关系梳理清楚）。

可以看到，这三大件和 Netflix 的猴子军团从功能上基本上是对标的。那是不是就应该没有故障，安枕无忧了呢？ 答案铁定不是。

理想很丰满，现实很骨感。阿里巴巴因为其多元化的业务场景和日益复杂的技术架构，会遇到各式各样的故障，故障治理的难度相比流媒体服务故障治理，难度是也增量了几个台阶。前面介绍过的强弱依赖和容灾演练只能覆盖到部分故障。如果对故障整体做初步画像，故障整体可以分为 IaaS 层、PaaS 层、SaaS 层的故障，每一层都可能有很多故障触发原因和表现。那么对于这么多种类繁杂的故障，内心一定是憷逼的。我们要如何重现，进而避免这么多繁杂的故障呢?

熟悉三体的同学应该听说过”降维攻击”这个词，不妨让我们把维度降低一下，换一个视角看故障：

- 任何故障，一定是硬件如 IaaS 层，软件如 PaaS 或 SaaS 的故障。 并且有个规律，硬件故障的现象，一定可以在软件故障现象上有所体现。
- 故障一定隶属于单机或是分布式系统之一，分布式故障包含单机故障。
- 对于单机或同机型的故障，以系统为视角，故障可能是当前进程内的故障，比如：如 FullGC，CPU 飙高；进程外的故障，比如其他进程突然抢占了内存，导致当前系统异常等。
- 同时，还可能有一类故障，可能是人为失误，或流程失当导致，这部分我们今天不做重点讨论。

任何故障都可以套入到这个故障模型中。有了这个模型，我们就可以开始来设计模拟故障的演练系统。

所以在内部，我们起了一个代号叫做”大圣归来”的项目，项目名叫做”故障演练”，承载的产品叫做 MonkeyKing。MonkeyKing 是中国美猴王的意思，看重的是孙悟空高强的本领（火眼精金、七十二变）和极具反叛的精神来，希望用一种创新的思路来保证稳定性。

我们的系统实现，也是围绕前文讨论的故障模型来设计的：

- 在客户机器部署 OS 层的故障插件，用来模拟硬件层的故障和单机进程外的故障。
- 对于应用进程内的故障，提供插拔式的故障插件，也可以用户按照我们的故障 API 做自己的实现。
- 对于分布式故障，则通过服务端按照 IP 来控制故障的范围。
- 对于一些因为各种原因无法触及的应用，比如数据库。我们提供了一个故障三方实现的标准，供故障服务接入。

通过上面的方式，基本上就把技术型故障的模型就 cover 全了。

在去年的双 11 中，故障演练的应用场景主要应用在图中的几个场景。 按照业务流量、压测流量的峰值可以划为 4 个象限。具体案例如下：

- 预案有效性：过去的预案测试的时候，线上没有问题，所以就算测试结果符合预期，也有可能是有意外但是现象被掩藏了。
- 监控报警：报警的有无、提示消息是否准确、报警实效是 5 分钟还是半小时、收报警的人是否转岗、手机是否欠费等，都是可以 check 的点。
- 故障复现：故障的后续 Action 是否真的有效，完成质量如何，只有真实重现和验证，才能完成闭环。发生过的故障也应该时常拉出来练练，看是否有劣化趋势。
- 架构容灾测试：主备切换、负载均衡，流量调度等为了容灾而存在的手段的时效和效果，容灾手段本身健壮性如何。
- 参数调优：限流的策略调优、报警的阈值、超时值设置等。
- 故障模型训练：有针对性的制造一些故障，给做故障定位的系统制造数据。
- 故障突袭、联合演练：通过蓝军、红军的方式锻炼队伍，以战养兵，提升 DevOps 能力。

故障演练宣言：把故障以场景化的方式沉淀，以可控成本在线上模拟故障，让系统和工程师平时有更多实战机会，加速系统、工具、流程、人员的进步。

关于未来：

故障演练的后续工作主要会关注在以下方向：演练常态化、故障标类化、演练智能化。用常态化的演练驱动稳定性进步，而不是大促前进行补习；丰富更多的故障场景，定义好最小故障场景和处理手段；基于架构和业务分析的智能化演练，沉淀行业故障演练解决方案。

如何高效排查系统故障？一分钱引发的系统设计“踩坑”案例

阿里巴巴集团成长集编委会

阿里妹导读：阿里巴巴的电商业务十分复杂，一方面是市场多样化，业务多样化，另外是消费者，商家的影响面非常广，任何一个小故障都可能引发一些社会问题，所以阿里对产品的质量，对服务的连续性有严格的要求。阿里技术人员在日常的研发运维过程中，积累了丰富的实战经验。今天，阿里妹将为大家分享一个关于故障，排查，分析和改进的真实案例。他山之石可以攻玉，希望对广大开发和运维工程师带来帮助。

背景说明

某日，做产品 X 的开发接到客户公司电话，说是对账出了 1 分钱的差错，无法

处理。本着“客户第一”的宗旨，开发立马上线查看情况。查完发现，按照产品 X 当日的年化收益率，正常情况下用户在转入 57 元后一共收益 3 分钱，合计是 57.03 元。但是该客户当日却有一笔消费 57.04 元，导致客户公司系统对多出的 1 分钱处理不了。再进一步分析，发现用户收益结转时多了 1 分钱的收益，并且已消费……

也就是说，本来用户只有 3 分钱收益，结果多发了 1 分钱给他，也就给公司造成 1 分钱的损失！用户在产品 X 里当天收益本应该是 0.03 元，怎么会变成 0.04 元呢？多出的 1 分钱收益从哪里来的呢？

数据库记录分析

带着上面的一系列疑问，开发人员首先排查了产品 X 收益的数据库记录。通过查询数据库发现，该用户收益结转在同一天内存在 2 笔交易记录。交易记录 1 创建时间为 8:00:23，记录 2 创建时间为 8:00:29，交易记录 1 和 2 的最后修改时间均为 8:00:29，如图 4-1 所示。

图 4-1　用户当日收益结转数据库记录分析

正常情况下产品 X 收益每天只会结转一次，而这个用户当日有两笔收益结转记录。开发人员怀疑，很可能是出现了并发问题。

继续跟踪第一笔“TXID a”的记录，开发确认线上日志存在超时情况，失败原因是数据库链接数已满，线程等待提交。

分布式锁超时时间是 5s，第一笔记录从创建到修改提交经历了 6s，由此可见是在分布式锁失效之后，获得了数据库链接，进行提交成功。

有了以上三个排查思路后，我们可以开始逆推整个过程。

过程逆推

根据数据库记录逆推当时的运行情况，如图 4-2 所示。

（1）由于数据库连接数被占满，流水 1 创建的事务处于等待提交状态。

（2）系统 A 发现交易失败，重试次数不满 8 次的，立即发起重试，触发生成流水 2 的请求。

（3）5s 以内数据均被分布式锁拦截，无法提交。

（4）经过 5s 后，系统 B 的分布式锁失效，此时事务仍在等待未提交。

（5）6s 时，流水 2 成功越过数据库查询幂等校验发起事务，此时流水 1 拿到数据库连接，流水 1 和 2 两个事务同时提交。

（6）由于数据库未做唯一索引，且支付受理模块打穿下层幂等原则，生成 2 个 TXID，导致两事务同时提交成功。

（7）收益结转重复记账，用户多了一笔收入。

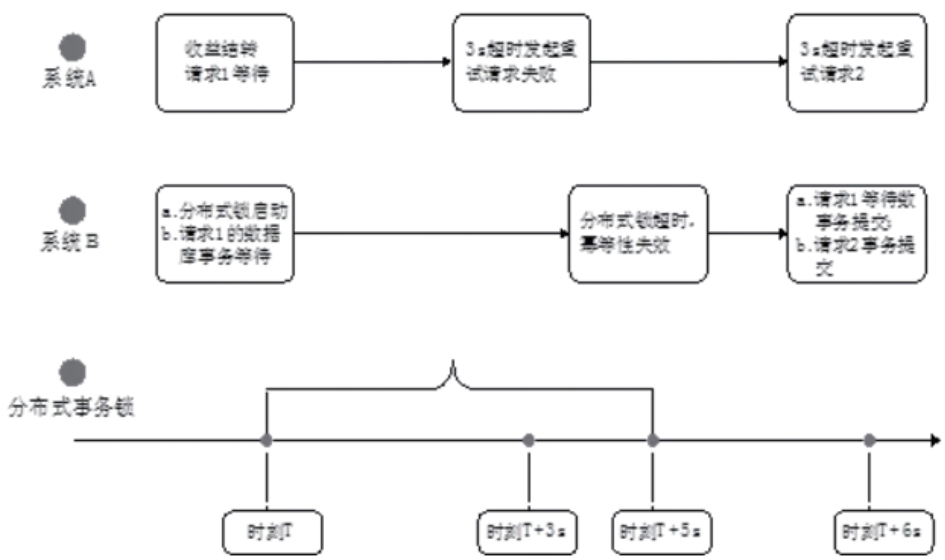


图 4-2　数据库分布式锁超时并发控制失效

深入分析

完成了整个问题的过程逆推后，开发人员进一步分析，发现问题真正的原因还是在系统设计上。如图 4-3 所示，系统 A 的事务允许一定时间的等待，而上层业务的重试时间又比这个等待的时间要短。这就存在一个问题：系统 A 的事务还在等待中，业务就又发起了重试。如果是在这个应用场景下（可能业务上对重试要求更高一些），那么对幂等控制的要求就更高了。而仅仅通过一个分布式锁来控制，如果分布式锁的超时时间设置的比事务允许等待的时间短，那么在锁失效之后就一定会同时提交两笔请求。

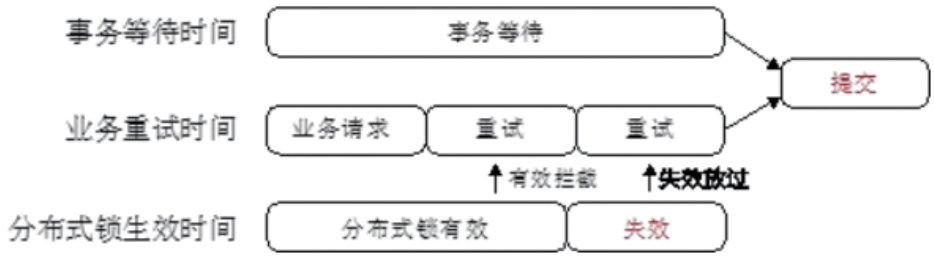


图 4-3　分布式锁超时并发控制时间轴

继续对整个过程抽象化，开发人员得出一个结论：分布式锁在以下条件同时满足的情况下并发控制会被打穿。

（1）上层业务系统层面有重试机制。

（2）业务请求存在一定时间之后提交成功的情况，例如本例中第一次请求在事务等待 6s 后获得了数据库链接，提交数据库成功。

（3）下游系统缺乏其他有效的幂等控制手段。

思考

了解了问题的来龙去脉后，接下来要怎么解决这类问题呢？我们想了以下几个方案。

（1）调整 B 系统上的 tr 和分布式锁超时时间，tr 超时调整为 10s，分布式锁超时调整为 30s。

（2）防止做收益结转产生并发控制幂等，调整了收益结转流水号的生成规则：前 8 位取 X 收益结转传入的交易号的前 8 位，第 10 位系统版本设置为“9”，最后 8 位 seq 取交易号的最后 8 位，降低问题出现几率。

方案一：调整超时时间

调整超时时间后，业务重试时间与分布式锁有效时间的分布时间轴如图 4-4 所示，即在事务允许等待后提交成功的时间之外，再进行重试，另外分布式锁在整个阶段均有效，防止提交。

图 4-4　分布式锁超时并发控制时间轴

方案一验证有效。

方案二：增加幂等控制（推荐）

如图 4-5 所示，单纯靠分布式锁不是控制并发幂等的方式，最稳妥的方式还是在提交记录的时候通过数据库严格控制幂等。确保不论如何设置超时时间，都不会出现幂等控制的问题。

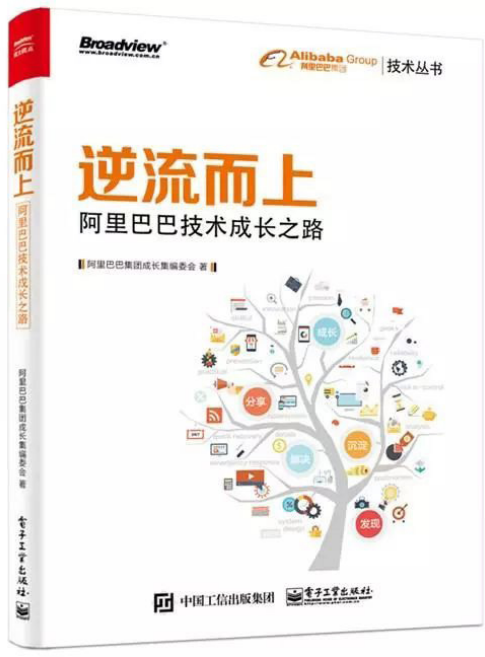
图 4-5　分布式锁超时并发控制时间轴

方案二验证有效。

小结

资金安全无小事，而幂等控制又是资金安全中的重中之重。回顾本文案例，从问题分析定位，到整个逻辑的梳理清洗，其中涉及了三个时间轴的相互作用，再加上事务、分布式锁、重试等，整个问题发生的逻辑还是比较复杂的。因此，在系统并发幂等控制设计中，单纯的分布式锁并不具备严格控制并发幂等的作用，建议在系统设计

时，将第三方唯一性的幂等控制作为幂等控制的兜底方案，控制好这道幂等防线，这样不论业务如何设计，就万变不离其宗了。



作者：阿里巴巴集团成长集编委会

本案例选取自《逆流而上：阿里巴巴技术成长之路》。该书通过分享阿里中间件、数据库、云计算、大数据等各个领域发生的典型“踩坑”案例，帮助大家快速提升自我及团队协作，学习到宝贵的处理经验及实践方案，为互联网生产系统的稳定共同努力。有兴趣的童鞋可以在天猫、淘宝搜索、购买此书。

阿里毕玄：智能时代，运维工程师在谈什么？

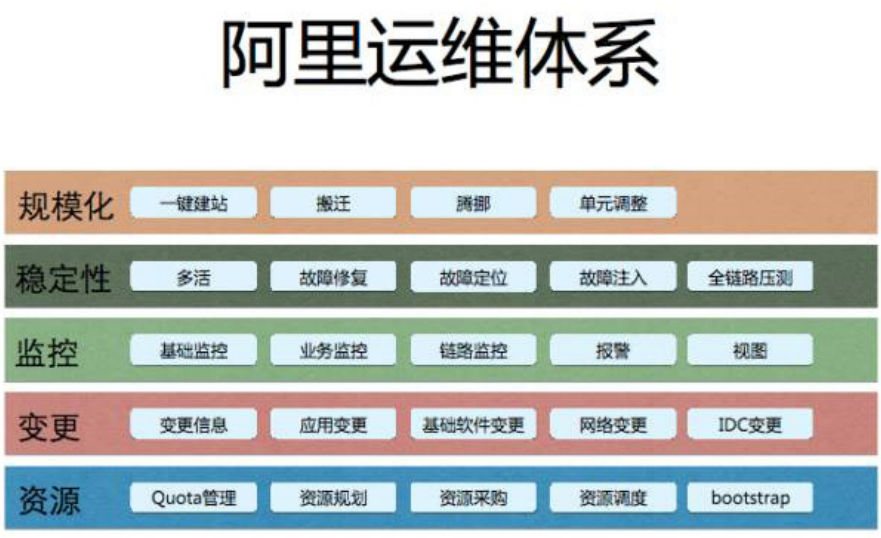
毕玄

阿里妹导读：智能化运维的终极目标，就是将运维人员从繁琐的工作中解放出来，提高整体运维效率，降低运维成本，实现业务系统的高可用性。

目前业界真正的智能化运维的落地实践其实并不多，大多还是停留在自动化甚至人工化阶段，然而智能化运维是大势所趋。阿里又是如何应对呢？下面请看来自阿里巴巴研发效能团队负责人、阿里研究员毕玄的演讲《智能时代的新运维》。

阿里的运维体系承载着怎样的责任？

阿里的运维体系介绍



阿里的运维团队，主要覆盖五个层面。

一．资源的规划与支付是运维的基石

整个运维团队需要负责资源的规划、资源的交付。

Quota 管理：比如我们会跟业务团队做一些预算的管理，对于每个业务团队首先需要有预算。只要你有预算，运维团队一定会把资源交给你，没有预算一切免谈。

规划：比如阿里每年的双十一交易，业务团队要给出下一年的交易额将做到多少，至于背后需要增加多少的机器量，业务团队根本不关心。所以需要运维团队来做从业务需求到资源的转化和规划，这对于公司来讲非常重要，因为意味着最终我在基础设施上要投多少钱，还有节奏的控制。

采购：当规模大了以后，怎么样合理规划资源的数量和交付节奏是非常重要的，比如 5 月份采购这批机器和 6 月份采购这批机器，是完全不同的概念。还需要资源的采购，比如 SSD 采购紧张，供应量不够。通常大公司会有更多的渠道获得更好的供应量，小公司就会很困难。怎么做好供应链控制是非常重要的。

资源调度：对于资源团队来讲，调度也很重要，我们交出去的机器是怎么样的交法，怎么保证可用性、稳定性，Bootstrap 等，每个业务都有自己的规划，按照业务需求怎么把整个业务环境全部交给业务方。阿里目前就遇到了很大的挑战，比如在国际化的扩张上，我们可能这个月需要在这里建个点，下个月需要在另一个地方建个点，怎么快速的完成整个资源，不仅仅是机器资源的交付，还有软件资源的交付，是非常重要的。我们现在在扩展东南亚的业务，怎么样在东南亚快速的完成整个软件资源的交付，对于我们的竞争是非常重要的。

二．变更是运维不可避开的坑

对于运维团队来讲，变更也是经常要做的部分，变更信息的收拢，做应用层面的变更，基础网络的 IDC 等等。

三．监控预测潜在的故障

监控对于阿里来讲主要分为基础、业务、链路，在监控的基础上要去做一些报警等。

四. 稳定性是不少企业追求的目标

稳定性这个概念我们以前认为针对的是大公司，因为它可能会影响到大众的生活，会比较敏感。但是现在新型的互联网公司，如外卖，ofo、摩拜等，它的稳定性要求比以前很多创业型公司更高，因为它有在那个点必须能用，如果不能用，对用户会有直接的影响。所以稳定性可能在整个运维行业会得到越来越高的重视，但是对于很多中小型公司，稳定性的投入相当大的。

五. 一键建站让规模化有力保障

像阿里在稳定性上主要会去做多活体系的建设，然后故障的修复、故障定位，然后还有一套全链路的压测。规模化是很多运维团队很痛苦的事情，可能今年机器在这个机房，明年你的基础设施团队可能告诉你，这个机房不够用了，我们要换个机房。反正在阿里巴巴，很多的运维人员都说了，我们每年的工作中有一项不用写的工作就是搬迁。虽然基础设施团队会承诺说三年内不会再搬，可是到了明年他会跟你说，由于某些原因我们还是再搬一下，搬完之后三年不会让你再搬。但是从我们过去发展的三年，每年都在搬。未来我们确实相信阿里巴巴，可能在未来搬迁会相对更少一点，

我们认为不能让搬迁成为阿里巴巴运维团队的核心竞争力。

我们在规模化层面做了很多事情，比如说我们做了一键建站，对于阿里来讲，我们对机器资源的交付时间，要求会越来越高。比如说双十一，是提前一个月交付资源还是提前两个月还是提前三个月，对我们来讲付出的钱是完全不一样，而且可能相差非常大。

所以，技术层面能不能更好的把这个时间缩短，是非常重要的。所以一键建站的重要目的就是这个，每年双十一我们都会拓展出非常多个站点，通过一键建站快速完成整个过程。搬迁就是我说的，反正我们每年都要搬，那我们应该把搬迁这套系统做得更好。还有腾挪，阿里很多时候因为需要做一些业务资源的复用，最好是有一个机柜，这个时候怎么更好完成挪的过程也是很麻烦。

我们还需要做一些单元的调整，因为对阿里的交易系统来讲是有单元的概念的，我们怎么更好的控制一个单元内机器的比率是非常重要的。一个单元的机器数可能是比较固定的，那如果比率搭配不好，就意味着瓶颈点会非常明显。

以上，正是阿里巴巴的运维团队所覆盖的五个领域。整个运维体系的演进过程，差不多都是从最早的脚本到工具到自动化，到未来的智能化。

从工具化到自动化过关斩将

从工具化到自动化这个层面，过程并没有那么的容易，以及对整个行业来讲，目前更多的工作仍然是在探寻自动化，怎么样让自动化真正的被实现得更好。

这个行业的发展跟其他传统的软件，标准的软件研发行业，我觉得很不一样。比如说阿里从工具化到自动化这个过程中，我们认为工具化，其实挑战相对小，即使传统的运维人员也很容易写一些工具，比如用 Python 去写更多的工具体系。但是如果你的工具最重要变成能够到自动化这个阶段，就意味着对工具的要求会越来越高，比如说工具的质量，如果你写出来的工具经常有问题，规模一大就扛不住，这个时候对于大家来讲慢慢会越来越失去信任感。最后会很难完成这个过程。

运维团队转型研发团队组织能力是最大的壁垒

阿里过去走这条路的过程中，我们觉得最大的挑战是组织的能力问题。运维团队怎么样更好的完成朝研发团队的转型，这个过程对于很多运维团队来讲都是巨大的挑战。对于一个组织来讲怎么完成这个过程也是非常重要的。

我想很多团队都有这个感受，工具研发的团队跟做运维操作的团队之间，很容易产生一些冲突等等。所以阿里巴巴在走这个过程的时候，思考的核心就是怎么让一个运维团队真正从组织能力上，演变成我们所需要的更好的团队。

阿里在走这条路的时候，走了四个过程。这个过程阿里在不断的摸索，最终到现在为止我们认为阿里的方式相对来讲还是不错的。我们最早跟大部分公司一样，有一个专职的工具研发团队和一个专职的运维团队。工具研发团队做工具，做出来给运维团队用。这个过程中容易出现的最明显的问题就是工具做完了，运维团队说这个工具太难用了，不符合需求。要么就是运维团队执行的过程中，经常出问题，出问题还要找工具研发团队来帮忙查问题在哪里。本来运维几行脚本全部能搞定的问题，结果还要依赖工具团队。慢慢这个局面越来越难突破，很难改变。

所以阿里后来做了一个尝试，既然两个团队很难做很好的结合，那有一种方式是工具研发团队做完工具以后，比如说做了一个发布，做完这个功能以后，这个运维工作就彻底交给工具研发团队，不让运维团队做了，运维团队就可以做一些别的事情。

这个模式看起来就是逐步接管的模式，让工具研发团队逐步解耦。

这个做了一段时间，碰到的最大问题还是组织能力问题。对于运维工具来讲，质量怎么做到很高，运维好像很容易做的样子，但是实际上运维工具相当难做，它的复杂度比在线业务更大，就是它不是逻辑上的复杂，更多的是环境层面的复杂。因为比如会涉及网络涉及服务器涉及机房等等，这跟业务完全不一样。所以做了一段时间之后，我们觉得这还是一个问题。

将工具的研发和运维融为一体突破组织能力问题

后面我们做完这轮之后又开始做另外一个方向的尝试，让工具的研发团队和运维团队做一个融合。所谓的融合就是把很多工具研发的人分派给运维团队，到运维团队去做。我们期望通过工具研发的人带动整个运维团队转变成研发型团队。这是我们的思路。

阿里巴巴在走前面这三步的时候，大概花了近一年半左右，意味着这其中我们大概做了三轮组织结构调整。因为我们认为这些都是要有组织层面的保障才能被实现的。

DevOps 是如何真正落地的

去年 6 月，我们做了一个最大的组织结构调整，把日常的运维工作交给研发做，研发自己会把日常的运维工作都做掉。但并不是说所有运维工作，现在仍然有一个做运维的团队，这个运维团队相对来讲更不一样，跟以前有非常大的不同。

我们认为这是 DevOps 真正的被彻底的执行。因为这个好处是，日常的运维工作交给了研发，运维团队转变成研发团队这个过程非常困难，其实不完全是能力上的差距，更大的原因是，运维团队要承担非常多的日常杂活，尤其像集团性的公司，不管是阿里、腾讯、百度都一样，集团性的公司多数支撑的 BU 都是无数个。你一个人支撑二十个 BU 一个 BU 里面一天有一个人找你，你一天就不用干别的活了，你一天就在跟他们不断的聊天，做操作，嘴里又叫着这个团队要升级，要做组织升级，要转变成研发团队，实际上就是逼别人走向了一条死路。

所以我们认为，谷歌的做法，谷歌在 SRE 那本书提到的是，会强制留 50% 的时间给研发团队做研发工作。这个说实话，在大多数公司很难执行这个政策，除非运维团队跟研发团队有非常强的话语权。但这个很难。所以阿里的做法我认为更为彻底，阿里告诉研发团队，以后日常运维的工作不要找运维团队，自己干。这可能粗暴了一点，在运维体系还没有准备得很好的情况下做了这个事情，所以后面相对来讲也导致了问题，比如说运维工具四处建设、重复建设等等现象。

但是从组织层面上来讲，我们很欣慰的看到，在做完这轮组织调整过后的一年后，运维团队的大多数人更多的时间是投入在研发工作上，而不是投入在日常的杂事上。 我们看到了一个团队的能力，在经过这一轮的调整得到了非常好的升级。而这对于组织来讲是最大的利好。所以我们认为，这种模式是阿里现在最为推崇也最为看好的一个方向，这样整个运维团队将专注在我刚才讲的五个部分的系统层面的研发以及建设上，而不是杂活上。这是阿里从工具化到自动化，最主要是这样的一个过程。

成功率是衡量自动化运维的关键指标

对于自动化来讲最重要的问题是成功率，比如我们看所有的运维操作中，我们最关心的指标是成功率。比如一个运维系统里面的功能，在一个星期内，比如说会用几十万

次，我们只关注成功率能不能做到 4 个 9 以上，否则算一下工单数就懂了，这个运维团队得有多少人支持这件事情，这些人又没有时间去干研发的活，又要投入大量的精力做支持性的工作。所以我们在成功率上要做到非常高的保障，运维系统我们以前看过是面临最大的挑战，我以前的背景全部是做在线业务型的系统，比如淘宝的交易等等。

后来我们发现运维系统有个最大的不同在于，运维系统对于成功率的追求比在线业务型系统更高一些。在线业务型系统，比如说我在访问后面一个地方有问题的时候，我们会选择尽快把这个过程失败掉，而不是把时间不断的拖长以及不断的试错。在线系统会更加快的把错误往外抛。但是对于运维系统来讲如果也这样做，就意味着这个成功率非常难保障。所以运维系统要有更好的思考，怎么保障一次运维操作，这背后可能有几十个系统，而且多数是无数的团队写的，阿里以前碰到的情况就是无数个系统，质量层次不起，什么都有。怎么保证在这么复杂的环境下，保证对外的，对用户层面这个成功率可以做到很高的。这是一个很大的问题。

规模带来的挑战也是不容小觑

随着规模的不断增长，所有开源类型的运维类的系统，在规模化，当你的机器规模等等其他规模上升到一个程度以后，通常来讲都会面临非常巨大的挑战。阿里巴巴所有的这种类型的系统，我们论证都是自己做是比较靠谱。最大的原因是规模，规模上去以后会遇到很多问题。像代码托管、代码编译什么的，以前认为不会有太大的问题，事实证明规模上来以后这些里面全都是问题。我们也要投入非常大的精力去做规模方面的解决。

所以我觉得，阿里从以前的工具化走向更加自动化的过程中，我们探讨的核心问题就是能不能有一个非常好的组织去完成这个过程。能让运维的团队更加转型向 DevOps 这样的方向。所以我们一直说，我们一直很纠结运维团队到底应该叫什么名字，我们一致认为，运维研发团队，我们觉得不大对，你的主要的活其实是干研发而不是运维。但是叫研发运维又有点奇怪。后来阿里巴巴基本上是叫研发团队。因为我们认为运维的研发团队和在线业务的研发团队没有本质区别，都是做研发的，只是一个在解决运维领域的业务问题。刚才讲的五个层次，运维领域的业务问题，也是业

务，没有什么区别。在线业务，比如解决交易的问题，解决其他问题，这是完全一样的。两个研发团队没有本质区别。

所以这个过程，阿里经过过去这一年的组织调整以后，我们看到整个自动化层面，阿里有了很好的进展，但是离我们的期望还要更加努力继续往前演进。

阿里巴巴在智能化领域的探寻之路

现在智能化这个话题特别火热，就像我们说，AI 这个名字兴起的时候，我们忽然发现，阿里巴巴所有的业务都讲 AI+ 自己的业务，被所有人狂批一通。我们要想清楚，具不具备 AI 化的前提，可能前提都不具备就不断探讨这个名字。因为业界在不断的炒热非常多的名词，让大家去跟随。

自动化是智能化的前提

对于我们来讲，我们认为，比如说就像我对这个团队，我自己的团队讲的一样，我认为智能化最重要的前提是，一是自动化。如果你的系统还没有完成自动化的过程，我认为就不要去做智能化，你还在前面的阶段。智能化非常多的要求都是自动化，如果不够自动化，意味着后边看起来做了一个很好的智能化的算法等等，告诉别人我能给你很大的帮助，结果发现前面自动化过程还没有做完全。

一个最典型的 case，阿里巴巴以前一直在讲，我们认为资源的搭配上，其实可以做得更好。比如说你半夜流量比较小，白天流量比较大，你能不能更好的做一些弹性，把资源释放出来去干点别的，然后白天再把它补起来。这从算法层面上并没有那么复杂，从算法层面做到一个简单的提升是很容易做的。

所以，当时我们就有很多团队做了一个东西，可以做到这一点。结果等到落地的时候发现，业务不能自动伸缩。如果你想，比如说有些机器上面负载特别高，有些机器特别低，我们希望负载能拉得更均衡，在线业务更加稳定化，做一个算法，比如说背包，更好的去做组合，结果就是这个东西做完了，给出了建议说最好这个应用调到那台机器，那台应用调到这台机器。给完之后业务团队看了一眼，我们不干，因为干这些工作全部要手工干，你还每天给我建议，更不要干了，每天就来调机器了。

所以首先你要想明白你的前提，自动化，具不具备自动化的能力，不具备的话没有必要在这方面做过多的投入。

数据结构化是智能化的源动力

目前 AI 领域基本是靠暴力，暴力破解，未来可能有别的方向，但是目前的 AI 基本上是靠大量数据的积累去寻找一个东西出来，所以它一定需要有大量的数据积累，数据包括非常多的东西，对于运维来讲，可能基础层面的数据，机器的数据，运维变更的数据，上面还有一些场景化的数据，比如你解决故障，有没有更好的结构化的收集数据，这是非常重要的。数据这个层面比较难做的在于，在最开始阶段，多数公司的运维数据都是不够结构化的，结构化不会做得那么好，当然会有结构化，但是结构化的因素不会足够好。

就像阿里巴巴在讲，我们在电商领域 AI 化，我们最大的优势就是不断对外部讲，我们拥有的是结构化的商品数据，其他公司最多从我们这里扒结构化的商品数据。你扒过去之后还要自己分析，并且做商品结构的调整，这非常困难。但是阿里巴巴自己天然，所有人都会帮你把结构做得非常好。所以对运维来讲也是一样，如果你想在智能化上有更多的突破，数据怎么更好的做结构化，是一个非常大的挑战。你很难想清楚。这两个地方是我觉得首先要想清楚的。

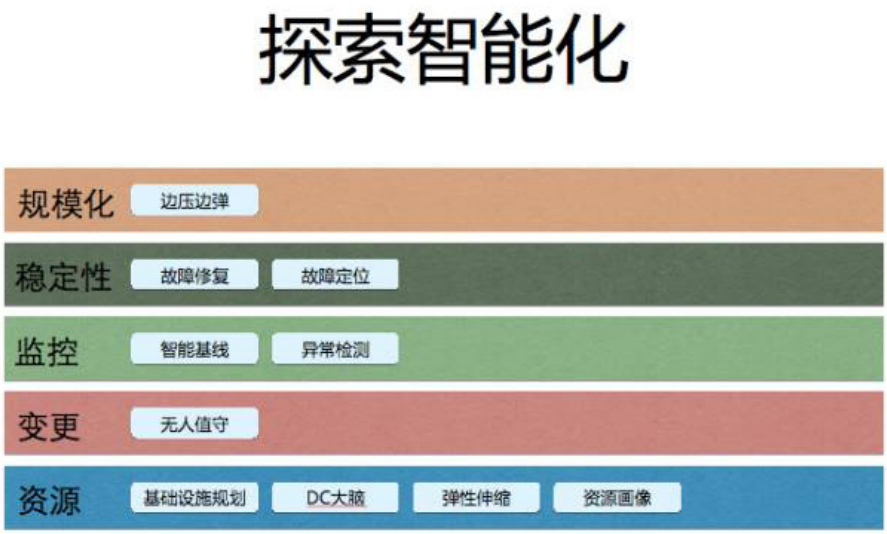
智能化最适合的运维场景

从目前来看，对于运维场景来讲，智能化特别适合解决的问题就两种，对于所有行业好像都差不多，第一是规模，第二是复杂。规模就意味着，我有很多的机器，在很多机器中我要寻找出一个机器的问题，这对于，因为规模太大了，这时候对于用传统的方式，将非常难解决这个问题。或者你要投入非常大的人力等等，有点得不偿失。规模上来以后怎么更好的解决规模的问题，智能化会带来一些帮助。

第二是复杂，比如说你的应用从原来的一个应用变成了几千个、上万个、几十万个，这时候你要寻找出其中哪个应用的问题，将是非常复杂的问题。所以复杂度的问题是人类用人脑非常难推演的，但是机器相对来讲是更容易做的。这是阿里有些团队希望尝试智能化的方向，通常我们会看是不是在前面的这些前提条件上都具备。如果都具备了，那可以去探索一下。所以我讲，阿里其实目前处于整个智能化运维的探索阶段，而不是全面展开阶段。

阿里巴巴智能化运维五步走

简单讲一下我们在各个领域目前在智能化这个领域，在运维这五个领域，对于我们讲，智能化我们看到的一些可能性，包括我们正在做的事情。



一. 资源的重点是成本

1. 基础设施选型

对于资源这一块，整个公司层面最为关注的问题，就是成本。你交付的资源具不具备最低的成本，这个智能化确实可以给非常大的帮助。比如第一点，怎么更好的规划这家公司机型、网络和整个数据中心，这为什么要用智能化的手段在于，一个数据中心的选址来自非常多的因素，除了政府层面的政策因素之外，还有很多其他因素需要考虑，比如说气候等等各种各样的因素，都需要在这个阶段去考虑。

你需要通过大量数据的积累来分析，比如在中国，在海外，到底有那些地方是对你的业务发展策略来讲最适合的，是在哪里，这要确定一个范围，在一个范围基础上是进一步的人的建立。对于网络、机型来讲，目前我们认为最可以做的在于，可能因为阿里的模式跟有些公司不一样，阿里更多的机器都来自同一个部门，基本上是同一个部门在教阿里巴巴所有的机器。这就有巨大的好处了，因为都在一个团队。比如阿里巴巴在去年开始建设统一的调度系统，更大的好处就来了，因为大家所有的资源都来自同一个地方，这个地方就收集了整个阿里巴巴的所有的资源需求、数据，数据全部在它手上。

如果你结合这个数据，以及它实际的运行情况，更好的就可以去推导，比如说对于阿里巴巴来讲最合适的机型是什么，这个阿里大概在去年就开始做尝试。在去年以前所有的过程，阿里巴巴，比如说明年我的服务器的机型，所谓机型，这里讲的机型的含义主要是比率问题，不是选择下一代什么样的 CPU，那是硬件发展决定的。 但是比率因素，以前我们更多的是人脑拍，人肉智能。人肉智能在一定阶段是更加高阶的，过了那个阶段之后人就比不过机器了。团队说我们明年要买的机型里面的配置大概是这样的，人算了一下，就这样吧，就可以拍掉。去年开始我们引入了一套系统，这套系统会分析所有的数据以及钱，最重要的是钱，然后分析一下整个过程，推演对我们来说最合算的是什么。所以适合的机型到底是什么。

如果有一套非常好的推演的系统，来推演你的机型、网络、IDC 未来应该怎么规划，这对于成本领域将会产生巨大的帮助。比如说网络，现在的发展，万兆，25G、45G、100G，你认为对于你的公司来讲最合适的是什么？多数公司八成就是人脑一

拍就决定了，但是事实上可能不是这样。

2. DC 大脑，让控制更加智能化

DC 大脑，这个现在比较火，这个领域现在非常火爆，火爆的主要原因有可能是因为去年谷歌的一篇文章，谷歌去年发表了一篇文章，里面有一个消息透露了一下，他们通过更好的智能化，去控制整个机房的智能等等。比如说控制空调的出口，就是那个风向往哪边吹，控制这个，然后为谷歌节省了非常多的钱，非常可观。所以对于很多数据中心团队来讲，现在都在研究这个领域。因为这个领域实在太省钱了。

我们后来类比了一下，我们说其实大多数人，可能你很难感觉数据中心，但是你最容易感觉的是另外一个地方，你的办公室。比如说我们以前说，阿里巴巴一到夏天的时候，办公室实在是太冷了，比外面冷多了。如果能够更好的控制温度，对于我们来讲就会有巨大的帮助，对公司来讲可能会更加省钱。所以怎么样做好这个非常重要。

3. 弹性伸缩最大的前提是实现自动化

弹性伸缩，这是无数运维团队都想做的事情，研发团队说，业务团队说，我要一

百台机器，你也不好反驳他，最后上线了一百台，你发现他用十台就够了。但是你也很难跟他纠结这个问题，好像无数的运维团队都在尝试弹性伸缩。但是我说了，弹性伸缩最大的前提就是自动化，如果没有自动化也没有什么意义。

4. 资源画像让资源更好搭配

资源怎么更好的搭配，阿里巴巴在尝试做资源的画像。对于所有的在线业务来讲，它的趋势比较好预测，多数在线业务，只有少数的在线业务不大好预测。多数在线业务是一个模式，如果预测得非常好，让资源有合理的搭配，对于这家公司的资源将会产生巨大的帮助。

二．可以下降 30% 由变更引起的故障

在变更这个领域我们觉得首先是效率问题。阿里巴巴现在大概有几万的研发人员，我们又把运维这个工作交给研发了，那怎么让研发在这个过程中，把变更这件事情做得更有效率和更没有感觉，是阿里巴巴现在追求的一个重点。这个重点我们认为，智能化是可以发挥巨大的帮助的。上面讲的第一个案例是讲的文件分发过程当中的智能的流控。比如一次发布要一个小时，那意味着多数研发是需要去盯一个小时的，他虽然不一定要一直看着，但是到发完之后是要去看一下，这挺耗精力的。

另外一个方向是现在业界很火的无人值守，怎么做到在发布过程中，对于研发来讲最好是无感，我制定了在某天发，只要测试通过了我就可以自动完成这个过程，有问题稍微控制一下就好了，没有问题就当这件事情没发生。这对于有众多研发团队，或者当然，如果你有运维团队在做这件事情，对运维团队来讲就更有帮助了，意味着运维很多人可能就去掉了一大块活。

所以，变更这个领域，我们最希望做的是朝这个方向去发展。目前来看阿里巴巴的尝试，我们可以看到变更引发的故障比率是最高的，目前已经铺的这个领域中，可以下降 30% 因为变更引起的故障，拦截主要是用来拦截问题。

三．监控 AI 化

智能报警

这个领域现在是 AI 进入运维行业中最火的领域，所有公司都在做。第一个是阿里在做的，阿里也不例外，我们也同样在做。第一个是智能，大家比如说做运维的都

知道，你写完了一个业务，要配监控报警的阈值的，比如说 CPU 到多少应该报警，然后响应时间到多少应该报警。阿里在尝试的一个方向是让你不要去配，阿里根据分析来决定什么情况下需要报警，这对于研发来讲有巨大的帮助。

异常检测直接影响到效率

第二点是异常检测，这是很多公司都在做的。异常检测之所以要做，最大的原因就是因为效率，如果不做，其实也 ok，但是要投入非常大的人力。比如说交易跌了，那到底是，比如对于我们来讲，交易跌了，只要跌了就需要分析到底什么因素。而这个因素很有可能，最后你发现根本跟我们没关系，可能是外部原因，国家节日等等，各种各样的因素造成的。尤其是小规模的业务，比如我们的海外业务，波动非常大，如果一波动就认为是问题，这对于整个公司的效率来讲是巨大的影响。

所以我们认为，如果异常检测做得非常好，对我们的效率会有非常大的帮助。这张图是通常来讲，做异常检测，运维的数据都是时序化，根据时序有各种各样的算法，上面列了业界常用的算法。最左上角的算法是阿里巴巴自己研究的算法，从我们目前的测试情况来看，我们可以看到阿里巴巴自己研究的算法的准确率等等，得比业界高非常多。细节我不讲了，最重要的原因是这个东西马上会在某个会议上发表一篇论文，大家以后会看到。

四．稳定性是以效率为原则

故障修复要精准且快速

稳定性对我们来讲最重要的是效率问题。第一个是故障的修复，故障出现在越大的公司越大的规模越复杂的业务场景中，出现是不可避免的，一定会出现，关键是出现之后怎么尽快把故障修复掉。故障修复这个领域，阿里巴巴尝试了非常多的方案，也尝试了很多年。很多的案例都是，这个过程需要慢慢的积累，原因在于信任感地当故障出现的时候，我们都说公司的很多团队都处于高度紧张的状态，这个时候有一套系统抛出了，现在多数这种系统都是抛出三个决定，给你三个建议，然后你来选。有时候经验丰富的处理故障的人一看，你抛出的三个建议都不靠谱。当十个故障中，有八次，不用八次，如果有个四五次都是这样的，以后所有人都不会看这套系统了，太

不靠谱了，还不如人来判断。这个系统难度非常高，需要整个公司坚定地朝这个方向走，并且更好的积累很多的数据。

故障修复，阿里现在只尝试了一些非常简单的案例，对于阿里来讲，比如一个机房出故障，因为整个阿里巴巴交易体系的架构是支持多点的，对于我们来讲如果在某种情况下，我们判断一个机房出故障，我们可以自动的做一些流量的切换等等。但阿里现在也认为，智能化在稳定性，尤其故障修复这种动作上，还是要非常小心，万一没事切出了问题，这影响更大。

用智能化做好故障定位

我们以前一直都认为定位这个问题不是个大问题，如果我能快速修复，定位，你慢慢定好了，定个两天我也无所谓。但是现在阿里特别重视的原因在于，故障定位损耗了我们非常多的人力，耗费了我们非常大的团队力量。所以我们认为需要有更智能化的方法，把故障定位出来，以助研发团队更专注投入在其他事情上。比如现在故障一出来，研发查了半天，一看，跟它都没有什么关系。所以就浪费了很多，这张图是我们现在在做的一套系统，从一个异常，那里标一二三四五，当有一个异常出来之后，第一步发现，第二步不断的分析，一直定位到最后到底是哪个地方出了问题，我们的目标是最后尽可能定位到代码层面的问题，或者是网络或者是基础设施等等。

五．边压边弹做好规模化运维

目前对阿里来讲最重要的问题还是效率问题。比如说我们在每年准备双十一容量的时候，很多人都知道阿里有全链路压测，一个最重要的目的就是调整容量，怎么把一个机房的容量调整成比率是最合适的，比如说 A 应用可能是瓶颈，但是事实上如果搭配得好，A 应用就不再是瓶颈。所以怎么样让一个固定机器数下做一个最好的搭配，我们以前是压一轮调整一下，再压一轮再调整一下，这非常耗费一堆人通宵的精力。我们认为这个过程需要提升，现在改成非常简单的模式，流量过来以后不断的自动调整容量比例，我们会有一个所谓边压边弹，一边压测一边调整比例。相信很多运维同学都干过这个事情，因为业务方给你一个指标，你是要算的，而且很难算的很精准。边压边弹意味着你不需要算得很精准，粗略算一个数就可以了，后面靠这套系统

自动给你调平衡。

未来运维领域需要突破的防线

无人化让梦想照进现实

我认为现在运维这个领域中最大的挑战仍然是，能不能真正的走向无人化，整个过程中是完全没有人的。

从目前来看，要做到无人化最重要的是质量问题，质量做得不够好是没有办法无人化的。另外如果出问题了能不能自动修复等等，所以我们认为无人化对运维领域是最大的挑战，能不能把这个落地变成现实，奠定了智能化的基础。如果说智能化所有的动作要人介入，那基本就不用做了。

智能化带来效率上的质变

在智能化这一点上，第一点是有效性的问题，如果这个智能表现得比人的智力还差一些，这个慢慢就没有人相信这个东西了。所以怎么样把有效性提升上来，另外最重要的是要看到智能化给运维领域带来效率上的质变。智能化投入非常大，要做大量的收集做大量的分析。所以最好带来的是质变而不只是量变，如果只是量变可能投入都收不回来。对于所有公司而言，更少的人更低的成本是非常重要的。人最好投入在一些更重要的研发等等事情上。

中间件

阿里 SRE 体系如何支撑 24 小时峰值压力、220+ 个国家“剁手党”？

子伟

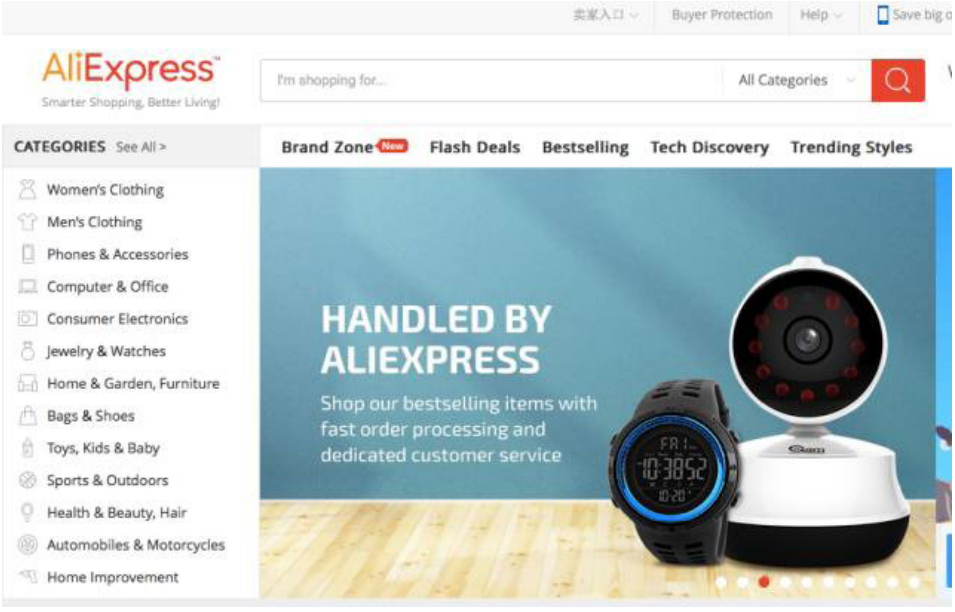
阿里妹导读：淘宝点亮了全中国，Aliexpress 点亮了全球，在近百个国家的购物类 app 排名第一。但 AE 国际只有 1-2 个物流，峰值压力一度导致多个国家的银行系统、物流系统瘫痪，可以想象，作为 Aliexpress 的 SRE 压力多大。

究竟阿里工程师是如何解决这一难题？今天我们通过 AliExpress SRE 负责人周志伟的分享，揭开这个谜题。

阿里巴巴高级技术专家、AliExpress SRE 负责人周志伟

Aliexpress 是阿里巴巴集团跨境及国际消费业务，国内大家都知道淘宝，但是走出国门，知道 Aliexpress 的外国人非常多了。AE 在 Alexa 全球排名前 50，淘宝排名 11，可以想象 Aliexpress 的流量当前已经非常庞大。此外，AE 在近百个国家的购物类 app 排名第一。如果有去俄罗斯的旅游朋友可以问问当地出租车司机、餐馆服务员、或者当地路人 Aliexpress，相信大多都有在上面购物的经历。

目前 Aliexpress 有过成交记录的国家覆盖 220+，大家都知道双 11，淘宝点亮了全中国，而 Aliexpress 点亮了全球。但 AE 国际只有 1-2 个物流，峰值压力一度导致多个国家的银行系统、物流系统瘫痪，爆仓已经不仅仅只有中国才会发生，可以想象，作为 Aliexpress 的 SRE 压力多大。



AliExpress（全球速卖通）是阿里巴巴旗下面向全球市场打造的在线交易平台，被广大卖家称为“国际版淘宝”

Aliexpress SRE

SRE 在 Aliexpress 的定义仅仅指与可用性相关，当它指一种技术方面时，是指原来的稳定性的概念；当它用来指团队时，是指各技术团队负责稳定性的同学组成的

虚拟团队。在 Aliexpress，SRE 是由横向的虚拟团队组成，每个业务团队一主一备保障整个 Aliexpress 的稳定性，只有这样才能最高效和最快速的发现问题根因和解决问题。稳定性是一切一切的基础，所以这个虚拟组织也是得到了众多的资源和 KPI 基础保障。

阿里国际化 SRE 的挑战

时差让每时每刻都是高峰期

在全球化的前提下，SRE 的挑战是非常巨大的，它的难题和挑战并非淘宝所经历过的，在这些方面我们也并没有太多的参考和借鉴。首先，Aliexpress 的用户群体分布来自全球 238 个国家，不同种族不同肤色。这不是最关键的，因为用户分布不同国家和不同时区，对于 Aliexpress 来说其实没有真正意义的低峰期，每个国家的高峰时间都不一样，一波接着一波，对我们的产品可用性提出了更高的要求所有国家*7*24。

网络复杂但容不得半点延迟

在中国，我们的网络在三大运营商的扶持下可以说是非常不错的，虽然偶尔有些抖动，起码我们很清楚也比较容易获得原因或者作出一些预案。但对于全球这么多国家来说，运营商非常复杂，带来的全球互联互通问题也非常复杂，这么多国家，花点

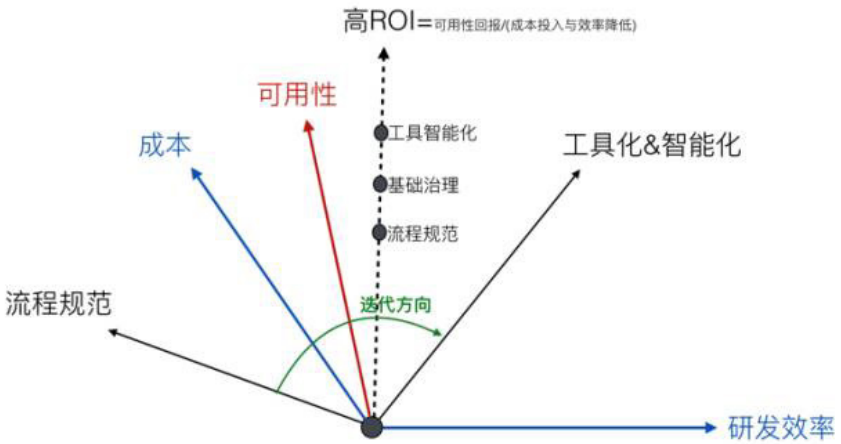
精力知道每个国家哪家运营商好，应该不难，但是要把各个国家串起来，互联互动这个问题就复杂了。

比如东南亚国家访问中国服务器和美国服务器，哪个会更快？从物理距离看应该国内会更快对吧？但是实际并非如此，东南亚访问国内，大部分都是绕行美国再连国内，这是多么奇葩的网路链路，但事实如此，就因为运营商接入美国再到中国更加便宜比直接接入中国便宜。这给我们全球化增加很多困难，我们需要做更多的事情去解决这类难题。

也许大家会说绕就绕呗，就这么用，可中国到美国来回耗时在 130ms 左右，还是在网络非常好的情况下，接近光速的速度，这个延迟看起来不起眼，我们做个对比，一般服务请求数据库基本在 5~10ms 左右会得到返回，如果有类似缓存机制就更快了，有多少服务能扛得住因为距离带来的 130ms 延迟。这对网站稳定性又提出了技术的挑战。国际形势下想获得用户的信息反馈，并不像国内那么容易，需要我们采取更多的手段去主动获取。

Aliexpress SRE 之路

在 Aliexpress，我们要提升可用性，需要考虑成本以及研发效率，在刚开始组建 SRE 团队的时候，没有任何基础，又想提升可用性，我们需要分析从何下手。这个图有点像力学，一方使劲，会造成另一方的倒退，如何寻找平衡，获得最高的回报率。



我们可以看到，可用性的追求是会降低研发效率的，可用性的追求是会增加研发和技术成本的，通过流程规范的建设是可以提升可用性的，但是会极大降低研发效率，通过工具化和智能化实现可用性，对效率提升有帮助，也对成本节省有帮助。

制定规范提高可用性

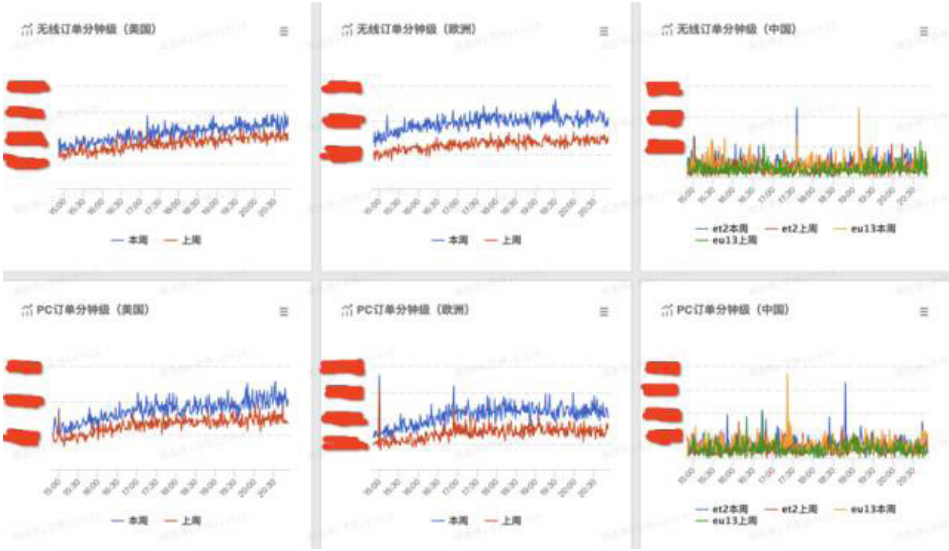
Aliexpress 的 SRE 初期，我们希望能快速的提升可用性，选择了成本最低，也是最容易先拿到结果的方式，制定流程规范，但是他给研发效率会带来降低，规范会有很多的制约，发布需要 review，核心应用改一行代码也需要多机房的观察监控，整个过程耗时比较久。

其实规范现行，虽然很土，不够酷，但非常见效，不得不说对于一线研发同学来说，规范不仅仅是对他的约束，通过稳定性规范的考试，让一线研发知道非常多的流程细则以及为什么需要这么做，以及其中风险是什么，更让一线工程师对生产环境有更多的了解。devops 的角色也得到很多认知上的提升，我们对历史的故障也是做过分析的，会发现有很大比例的问题都是对于生产环境的陌生，不小心或者不知道该怎么做而产生了问题。我们全球化多机房有很多地方需要注意，对线上的陌生一定会带来问题，比如多机房数据同步，没有做好任务消费的幂等性处理，一定会造成数据的一致性的问题，这是架构规约的一部分，也是 SRE 稳定性的范畴。 此外，我们坚持每个半年会进行一次稳定性考试，让大家对规约，线上环境有知识的迭代更新，对生产环境的操作时刻保持敬畏之心。

Aliexpress SRE 基础治理

对于 SRE 来说，最想做到的就是线上发生的一切都在掌控之中，即使出现问题，我们能通过有效的手段快速恢复，这也是 Aliexpress SRE 的核心。我们的治理也是从这条核心思想出发的，首先要做到这点，不可或缺的是对整个站点有全面的监控，出现问题我们都能快速发现，那就是监控。监控建设是有成本投入的，需要业务系统追加日志输出，根据需要绘制出我们想要的核心大盘（交易、流量、登录等待），如果有下跌，可以进入下一级分级是由哪些渠道造成的，帮助快速发现问题，同时我们也会追加分机房的大盘，这个后面会描述为什么需要这样的分类，有何目的。

一开始做这个事情的时候并没有那么系统的来做，而是各个团队分别输出日志，然后手工配置监控大盘，去年年初我们开始推广微服务 Springboot，结合 Springboot 定制一个 starter 专门做日志的标准输出，采集所需日志的同时提升研发效率，标准化的日志对于后续的大数据分析来说非常有利，这也是长远考虑的一步，为日后智能化做的铺垫。



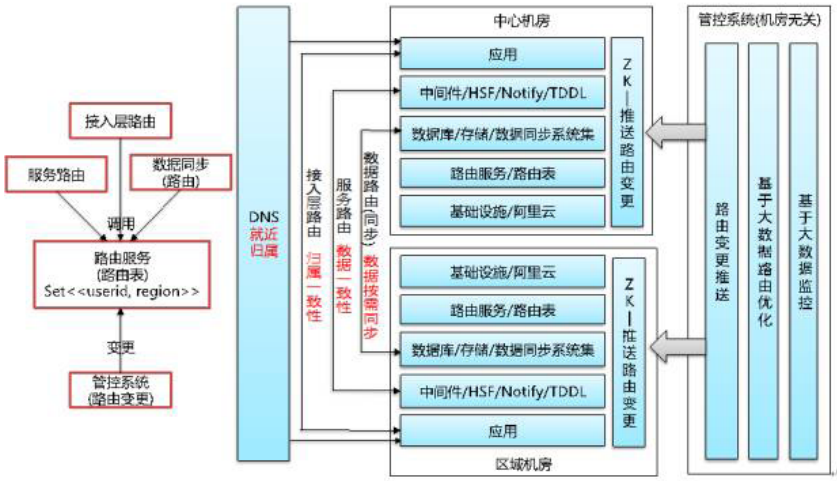
前面提到，对于 SRE 来说，希望自己有掌控权，监控的完善只能做到可见，没有掌控能力，所以我们还有几件事情，让 SRE 有掌控能力。快速恢复能力，俗称“容灾”，在 Aliexpress 容灾是一个体系，分了很多层，应对不同问题而定，这也是全球化所需。

之所以这么做，是有背景的，在前面提到 Aliexpress SRE 面临的挑战，全球网络质量问题，对于 Aliexpress 来说是不会轻易去切换 DNS，原因主要有 2 个：

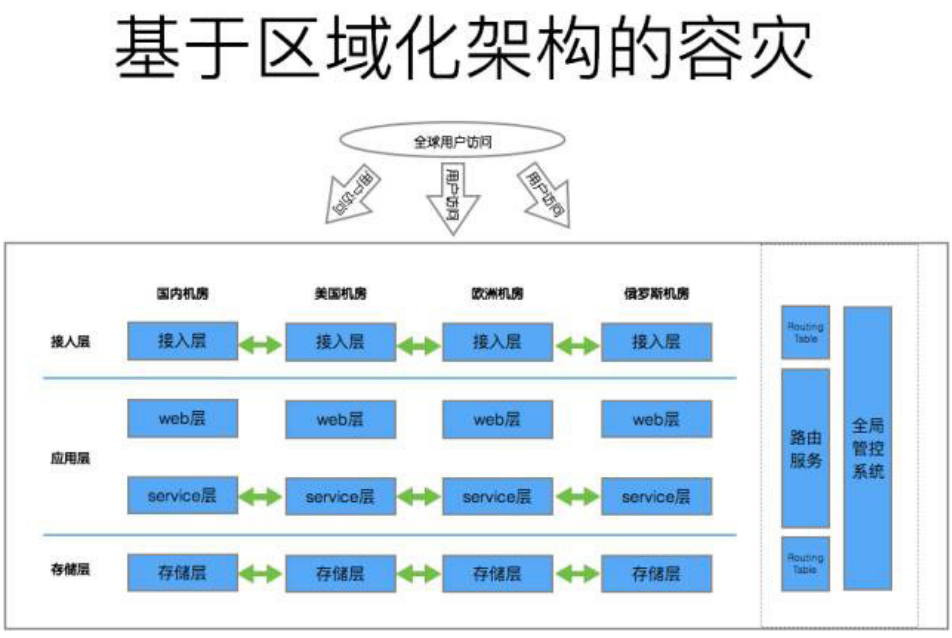
1. 全球化架构会针对用户归属进行路由，接入层的改变并不会使其在后续链路发生变化 2.DNS 的切换会带来性能损耗，更何况我们有很多 cdn 策略，切换回源带来的性能损耗大概在 8~15%，这个损失太心疼了

打造全球化架构

全球化架构，可以简单的描述为我们通过管控全球的用户，通过大数据的计算，配合 DNS 就近最优解析，将用户分别归属到不同的区域 IDC，让全球用户获得最优的购物体验。基于这套逻辑通过严格的版本控制利用 ZK 上报确保全球多个 IDC 的用户路由表保持一致，接入层、服务层、数据层包括数据同步加载用户路由表，进行区域的修正路由，确保归属用户的操作都在一个区域完成，以达到全球数据一致性。这是一套完整的全球化解决方案。



基于这套架构，SRE 的容灾也变的更加丰富，当某个变更导致 web 层发生问题，比如英文站搜索页面出现问题也许是样式也许是页面处理逻辑，基于我们的规范严格按照分机房发布策略，至少有一个机房是可用的，可以通过容灾到没有污染的区域，而其他层的逻辑都不发生改变。



当服务层发生问题，同样可以将用户从 A 区域切换到 B 区域，而在网络接入层不发生任何变化。这一层的容灾更加细腻，支持分流观察，按比例分流。当然发生重大问题，可以将整个区域 failover，切换到灾备区域这些容灾都是秒级生效，并且有数据保护停写机制。

建立保障机制 GTR

前面说到的都是基于机房级别的快速恢复，在全球化背景下，应对全球互联互通问题，我们也做了一套保障机制，GTR（global traffic routing service）。

简单介绍下这个图的含义：红点代表我们在全球有多个 POP 点，也就是网络接入点，五角星代表我们全球的 IDC，思路是我们采集全球用户访问我们机房的信息，比如某国用户通过访问不同的 pop 点然后到我们的 IDC，POP 点都会汇入阿里骨干网，可以认为更加稳定，类似动态加速技术。

通过国家对应 pop 点对应 IDC 的访问响应时间来判断，当一条线路发生问题时，我们可以将这个国家或者叫区域的用户访问切换到其他网络线路，这个是某个国家访问美国机房的数据，通过德国接入点进入美国机房和直接从美国接入美国机房的时间差不多。

以上是对 SRE 监控、容灾的介绍，在此之外，我还是要分享下我们应对重大问题时，确保能快速定位恢复，这套应战平台对于收集作战人员经验起了很好的作用。作战成员都是各个产品线的专家，他们的经验在平台得以沉淀也为我们国际 SRE 智能化有很大的帮助。

Aliexpress SRE 成立以来成果也比较明显，故障数明显下降，一线研发对线上环境以及自己身为 devops 角色的转型都比较成功，在这过程中，成功恢复多次重大线上故障，这也证实了我们平时的演练非常有效和重要。同时也为阿里集团国际化构建基础技术。

在 SRE 的体系运作下，朝着一个良性的循环运作，稳定性规范 – 分区域变更 – 分区域监控 – 分区域容灾 – 常态化演练。持续的优化工具、沉淀数据、培养研发素养，为未来 SRE 智能化做好准备。在这个体系下，我们会持续优化和丰富我们的自动化工具，丰富我们数据，优化我们的基础治理，往智能化的方向发展。谢谢大家。

Aliexpress 诚邀有国际化背景的技术人员加入，直达邮箱：zhiwei.zhouzw@alibaba-inc.com

史上最复杂业务场景，逼出阿里高可用三大法宝

游骥 & 子矜

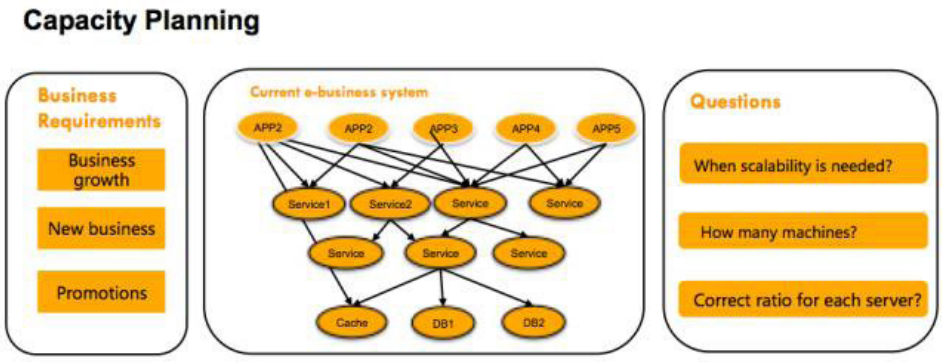
SREcon 是由计算机科学领域知名机构 USENIX 主办，聚焦网站可靠性、系统工程、以及复杂分布式系统相关的运维行业技术盛会，今年 SREcon17 大会 Asia/Australia 站于当地时间 5 月 22 日 -24 日在新加坡举行。阿里中间件（Aliware）团队高级技术专家张军（花名游骥）和林佳梁（花名子矜），受邀在本次大会上给现场听众分享了阿里巴巴容量规划和全链路压测方面的技术进展。

容量规划的由来

阿里巴巴有着非常丰富的业务形态，每种业务都由一系列不同的业务系统来提供服务，每个业务系统都分布式地部署在不同的机器上。随着业务的发展，特别是在大促营销等活动场景下（比如双 11），需要为每个业务系统准备多少机器对于阿里巴巴技术团队来说是一大难题。

“容量规划”正是为解决这个难题而诞生，容量规划的目的在于让每一个业务系统能够清晰地知道：什么时候应该加机器、什么时候应该减机器？双 11 等大促场景

需要准备多少机器，既能保障系统稳定性、又能节约成本?



Capacity planning is the process of determining the capacity needed by a complexity distributed system to meet the workload with guarantee on certain level of performance

容量规划四步走

在双 11 等大促场景的准备过程当中，容量规划一般分为四个阶段：

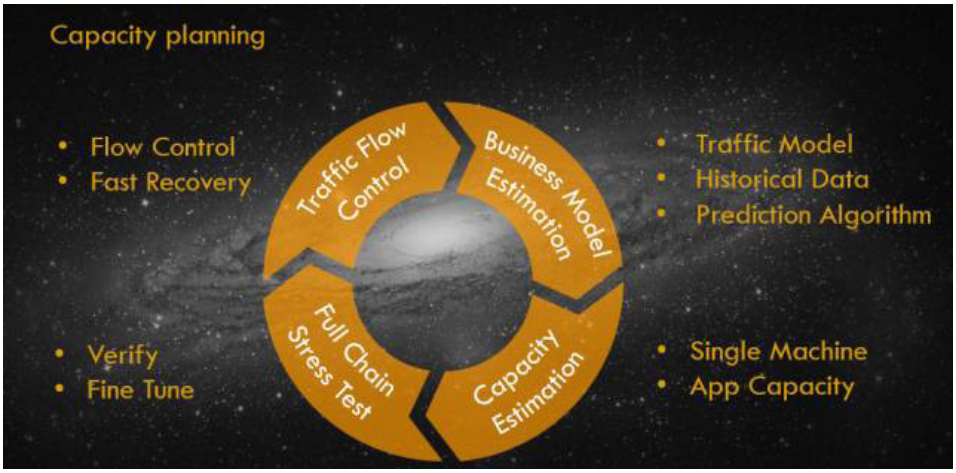
第一个阶段为业务流量预估阶段，通过历史数据分析未来某一个时间点业务的访问量会有多大；

第二个阶段为系统容量评估阶段，初步计算每一个系统需要分配多少机器；

第三个阶段为容量的精调阶段，通过全链路压测来模拟大促时刻的用户行为，在验证站点能力的同时对整个站点的容量水位进行精细调整；

第四个阶段为流量控制阶段，对系统配置限流阈值等系统保护措施，防止实际的业务流量超过预估业务流量的情况下，系统无法提供正常服务。

在第一个阶段当中，通过合适的预测算法和丰富的历史数据，通常能够比较准确的预估业务的访问量。即使在第一阶段预估的业务访问量跟实际的存在误差，通过第四阶段的流量控制也能够确保站点始终处于良好的服务状态。做完业务访问量的预估之后，容量规划进入第二阶段，为系统进行容量的初步评估。如何通过精准的容量评估，用最小的成本来支撑好预估的业务量是这个阶段的核心问题。



要计算一个系统需要多少台机器，除了需要知道未来的业务调用量之外，还有一个更重要的变量，就是单台机器的服务能力。获取单台机器的服务能力在阿里巴巴是通过单机压测的方式来获取。在阿里巴巴，为了精准地获取到单台机器的服务能力，压力测试都是直接在生产环境进行，这有两个非常重要的原因：单机压测既需要保证环境的真实性，又要保证流量的真实性。否则获取到的单台机器服务能力值将会有比较大的误差，影响到整个容量规划的准确性。

生产环境进行单台机器压力测试的方式主要分为 4 种：

1. 模拟请求，通过对生产环境的一台机器发起模拟请求调用来达到压力测试的目的；

2. 复制请求，通过将一台机器的请求复制多份发送到指定的压测机器；

3. 请求转发，将分布式环境中多台机器的请求转发到一台机器上；

4. 调整负载均衡，修改负载均衡设备的权重，让压测的机器分配更多的请求。

模拟请求的实现比较简单，也有非常多的开源或者商业工具可以来做请求模拟，比如 apache ab、webbench、httpload、jmeter、loadrunner。通场情况下，新系统上线或者访问量不大的系统采用这种方式来进行单机压测。模拟请求的缺点在于，模拟请求和真实业务请求之间存在的差异，会对压力测试的结构造成影响。模拟请求的另一个缺点在于写请求的处理比较麻烦，因为写请求可能会对业务数据造成污

染，这个污染要么接受、要么需要做特殊的处理（比如将压测产生的数据进行隔离）。

为了使得压测的请求跟真实的业务请求更加接近，在压测请求的来源方式上，我们尝试从真实的业务流量进行录制和回放，采用请求复制的方式来进行压力测试。请求复制的方式比请求模拟请求方式的准确性更高，因为业务的请求更加真实了。

从不足上来看，请求复制同样也面临着处理写请求脏数据的问题，此外复制的请求必须要将响应拦截下来，所以被压测的这台机器需要单独提供，且不能提供正常的服务。请求复制的压力测试方式，主要用于系统调用量比较小的场景。

对于系统调用量比较大的场景，我们有更好的处理办法。其中的一种做法我们称为请求的引流转发，阿里巴巴的系统基本上都是分布式的，通过将多台机器的请求转发到一台机器上，让一台机器承受更大的流量，从而达到压力测试的目的。

请求的引流转发方式不仅压测结果非常精准、不会产生脏数据、而且操作起来也非常方便快捷，在阿里巴巴也是用的非常广泛的一种单机压测方式。当然，这种压测方式也有一个前提条件就是系统的调用量需要足够大，如果系统的调用量非常小，即使把所有的流量都引到一台机器，还是无法压测到瓶颈。

与请求引流转发的方式类似，最后一种压测方式同样是让分布式环境下的某一台机器分配更多的请求。不同的地方在于采用的方式是通过去调整负载均衡设备的权重。调整负载均衡方式活的的压测结果非常准确、并且不会产生脏数据。前提条件也需要分布式系统的调用量足够大。

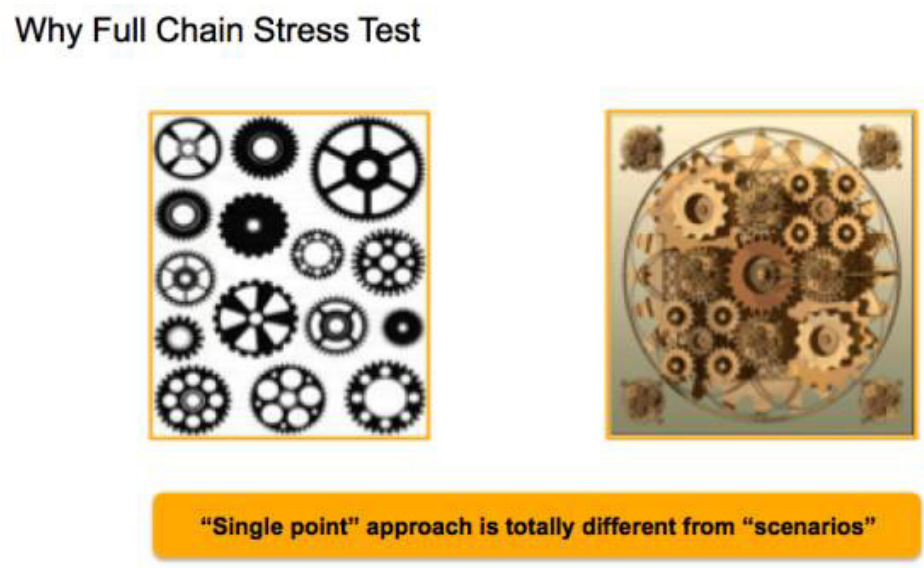
在阿里巴巴，单机压测有一个专门的压测平台。压测平台在前面介绍的 4 种压测方式基础上，构件了一套自动化的压测系统。在这个系统上，可以配置定时任务定期对系统进行压测，也可以在任意想压测的时间点手动触发一次压测。

在进行压测的同时，实时探测压测机器的系统负载，一旦系统负载达到预设的阈值即立刻停止压测，同时输出一份压测报告。因为是在生产环境进行压测，我们必须非常小心，保障压测过程不影响到正常的业务。在单机压测平台上，每个月将进行 5000 次以上的压测，系统发布或者大的变更都将通过单机压测来验证性能是否有变化，通过单机压测获取的单机服务能力值也是容量规划一个非常重要的参考依据。

有了预估的业务访问量，也知道了系统单台机器的服务能力，粗略的要计算需要

多少台机器就非常简单了。最小机器数 = 预估的业务访问量 / 单机能力。通常情况下，我们会预留少量的 buffer 来防止评估的误差和意外情况。

为什么需要全链路压测?



进行到这一步，我们已经完成了系统容量的粗略评估，然而做到这一步是不是就够了呢？过去的教训曾经狠狠地给我们上了一课。

我们对每一个系统都做好了粗略的容量计算，以为一切都会比较顺利了，可是真实场景并非如此，当双 11 的零点到来的时候，许多系统的运行情况比我们想象的要更坏。原因在于真实的业务场景下，每个系统的压力都比较大，而系统之间是有相互依赖关系的，单机压测没有考虑到依赖环节压力都比较大的情况，会引入一个不确定的误差。这就好比，我们要生产一个仪表，每一个零件都经过了严密的测试，最终把零件组装成一个仪表，仪表的工作状态会是什么样的并不清楚。

事实上我们也有过血的教训。在 2012 年的双 11 零点，我们一个系统的数据库的网卡被打满了，从而导致部分用户无法正常购物，尽快当时我们做了非常充分的准备，但还有一些事情是我们没考虑到的。

需要怎么样才能解决这个问题？在 2013 年的双 11 备战过程当中，在很长一段时间内这都是我们面临的一个难题。在中国，学生通常都会有期末考试，为了在期末考试中取得比较好的成绩，老师通常会让学生们在考试前先做几套模拟题。

双 11 对我们的系统来说就是一年一度的期末考试，所以我们冒出了这么一个想法：“如果能让双 11 提前发生，让系统提前经历双 11 的模拟考验，这个问题就解决了”。通过对双 11 零点的用户行为进行一次高仿真的模拟，验证整个站点的容量、性能和瓶颈点，同时验证之前进行的容量评估是否合理，不合理的地方再进行适当的微调。

我们为此研发了一套新的压测平台—“全链路压测”。双 11 的模拟可不是一件简单的事情，上亿的用户在阿里巴巴平台上挑选、购买好几百万种不同类型的商品，场景的复杂性非常高。有三个最主要的难点需要解决：

1、用于的请求量非常大，在双 11 零点，每秒的用户请求数超过 1000w；

2、模拟的场景要跟双 11 零点尽可能的贴近，如果模拟的场景跟双 11 零点差距太大，将不具备实际的参考价值，而双 11 零点的业务场景非常复杂；

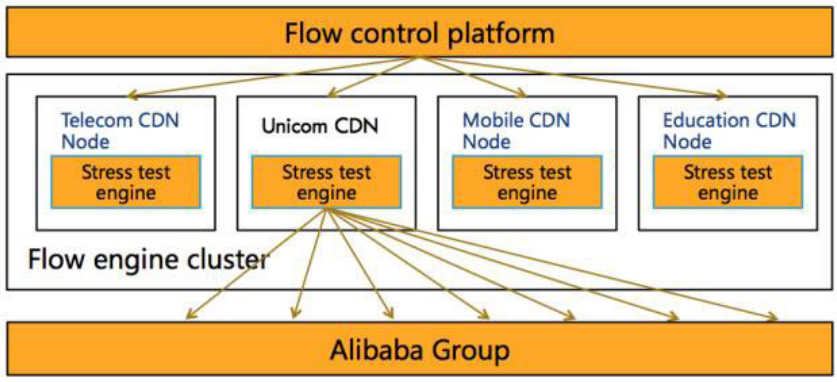
3、我们需要在生产环节去模拟双 11，如何去做到模拟的用户请求不对正常的业务和数据造成影响。

为了能够发出每秒 1000w 以上的用户请求，全链路压测构件了一套能够发出超大规模用户请求的流量平台。流量平台由一个控制节点和上千个 worker 节点组成，每一个 worker 节点上都部署了我们自己研发的压测引擎。

压测引擎除了需要支持阿里巴巴业务的请求协议，还需要具备非常好的性能，要不然 1000w 的用户请求，我们将无法提供足够多的 worker 节点。上千个压测引擎彼此配合、紧密合作，我们能像控制一台机器一样控制整个压测集群，随心所欲的发出 100w ／ s 或者 1000w ／ s 的用户请求。

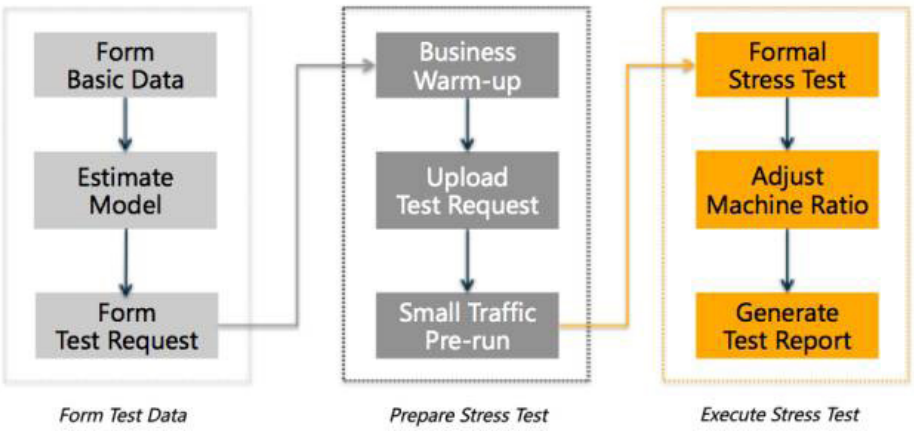
1000w + ／ s 的用户请求量不仅要能够发送出来，而且还需要跟双 11 的用户行为尽可能的接近，而双 11 是一个非常复杂的业务场景。为了使得模拟能够更加真实，我们做了非常多的工作。首先，我们从生产环境提取一份跟双 11 同等数量级的基础数据（包含：买家、卖家、店铺、商品、优惠等等），做好筛选和敏感字段的脱

敏，作为全链路压测的基础数据。然后基于这些基础数据，结合前几年的历史数据，通过相应的预测算法，得到今年双 11 的业务模型。



双 11 的业务模型包含 100 多个业务因子，比如：买家数量、买家种类、卖家数量、卖家种类、商品数量、商品种类，pc 和无线的占比，购物车里的商品数量，每一种业务类型的访问量级等等）。有了业务模型之后，再根据业务模型构造相应的压测请求，最终将压测请求上传到压测引擎。

全链路压测直接在生产环境进行双 11 的模拟，在前面的单机压测方式中也有提到，对于模拟请求的方式，需要考虑脏数据的处理方式。全链路压测的所有数据都在生产环境做了数据隔离，包含存储、缓存、消息、日志等一系列的状态数据。在压测请求上会打上特殊的标记，这个标记会随着请求的依赖调用一直传递下去，任何需要对外写数据的地方都会根据这个标记的判断写到隔离的区域，我们把这个区域叫做影子区域。全链路压测对粗略的容量评估起到了精调的作用，使双 11 零点的各种不确定性变的更加确定。



我们在 2013 年双 11 前夕的全链路压测过程当中共发现了 700 多个系统问题，2014、2015、2016 同样也发现了好几百个问题。这些问题如果没有在全链路压测的过程当中被发现，很有可能会在双 11 零点的真实业务场景当中暴露出来，将造成严重的可用性影响。

意外的甜蜜，超限后的流量控制如何做?

前面章节我们讨论的都是”容量规划”，我们知道容量规划是基于一套精密的业务模型，而这个业务模型是根据历年来的大促数据，以及复杂的预测模型推算出来的。然而，不论这个模型多么强壮，它始终是一个预测。这就意味着我们存在着预测和现实流量有误差。

这个并不仅仅是一个担心，这个发生过非常多次。

最近的一个例子是在 16 年的双 11，我们为某一个重要的场景预备了足以应付 16.2 万每秒的峰值，然而那天的峰值实际上到达了 20 万每秒，超过我们准备能力将近 13%，你可能觉得这只会对峰值产生影响，这些额外的 2W 请求马上就会被消耗掉，但并不是你想的这样。

当一台机器超负荷运转的时候，这台处理请求的时间会变长。这会给用户带来不好的体验，用户会试图重复提交请求，这无形中又给系统带来了更多的请求压力。随着请求堆积的越来越多，系统性能会逐渐下降甚至无法响应新的请求。

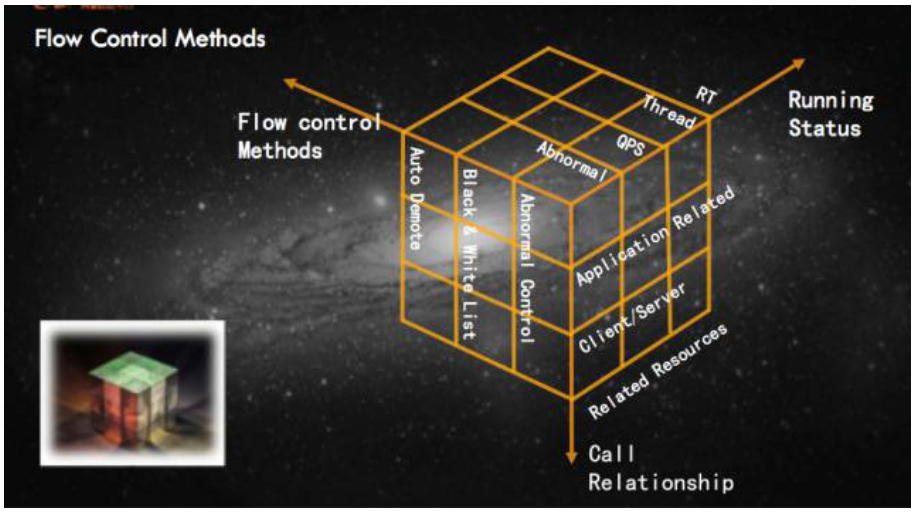
当一台机器挂掉以后，负载均衡会把请求重定向到另外的机器上去，这又无形中给别的机器带来了更多的任务，而这些机器也处于一个饱和的状态，很快也会像第一台机器一样，也无法响应新的请求。

就这样，在很短的时间之内，越来越多的机器会停止响应，最终导致整个集群都无法响应。这就使我们常常说的“雪崩效应”。一旦“雪崩”发生，就很难停止。我们必须有一个有效的机制，来监控和控制进入的流量，来防止灾难的发生。

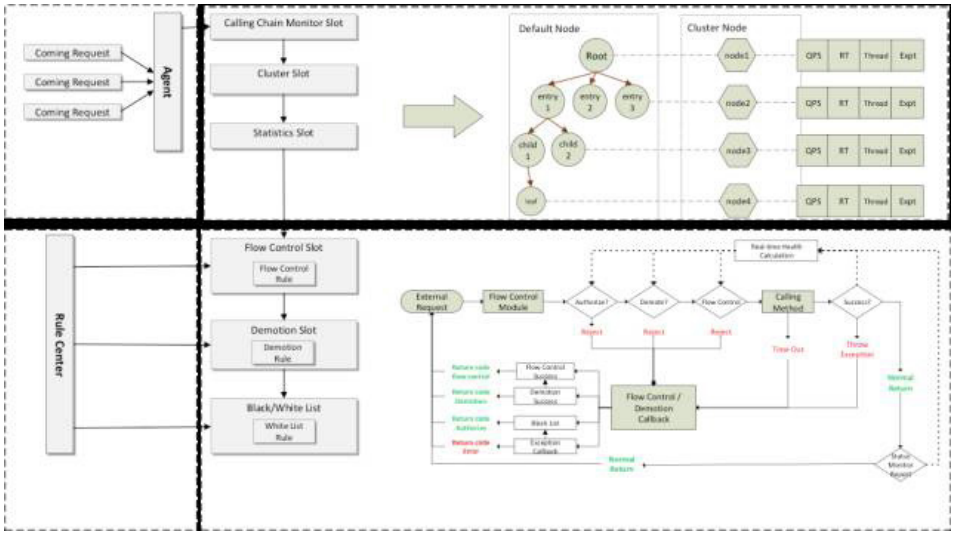
然而，流控并不仅仅用于流量高峰，它在很多的场景都可能用的到。比如在一个业务的链路上，有一个下游系统出现了问题，响应时间变得很长。这个问题在链路上会被放大，甚至导致整个链路不可用。这意味着流控也需要可以根据响应时间来控制系统的健康，当一个应用响应的时间超过阈值，我们可以认为这个应用不可控，应该迅速将它降级。

除了流控的激发原因之外，流控也可以灵活的定义流控的方式。不同的业务场景，可以采取不同的流控方式。比如说，对于有的应用，我们可以简单的丢弃这个请求，有的应用，则需要对下游应用进行降级，甚至直接加入黑名单。而有的应用，则需要把这些多余的请求排队，等到高峰期过后，系统没有那么忙碌之后，再逐步消耗这些流量。

所以，我们最终的流控框架可以从三个纬度着手，运行状况，调用关系，流控方式。应用可以灵活的根据自己的需求，任意组合。



下面这个是我们流控的架构图：



第一步，我们在程序入口给所有的方法都进行埋点；

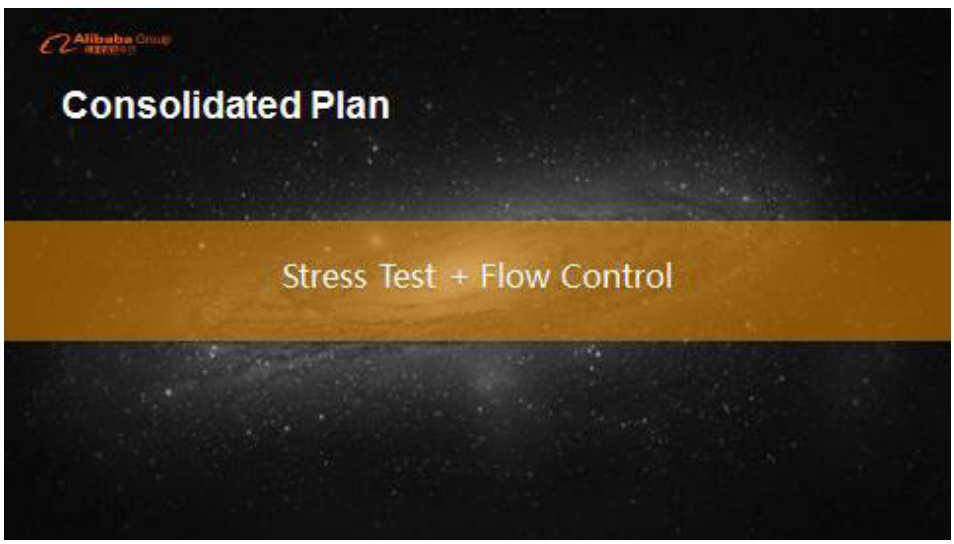
第二步，我们把这些埋点方法的运行状态，调用关系统计记录下来；

第三步，我们通过从预设好的规则中心接收规则，来根据第二步中统计到的系统状态进行控制。

然而，当系统发生流控的时候，系统虽然是安全的，但是它始在一个“受损”状态下运行。所以我们也在问题排除之后，解除流量控制。用我们上面的场景作为例子。 一个链路上的一个下游应用出现了问题，导致响应时间变长，从而导致上游应用的系统负载过高。过了一会儿之后，这个下游应用恢复了，响应时间大大缩短。然而这个时候，上游应用的负载并不能马上恢复，因为进来的请求已经堆积了一段时间了。

这就意味着，如果我们采用传统的方式，用系统负载来判断是否应该恢复流控，那么即使问题已经修复，系统地负载仍然处于一个比较高的状态。这样就会导致系统恢复慢。既要迅速恢复，同时也要系统稳定。最后我们采取的方式是，让 rt,load, 允许通过的 qps 达到动态平衡。

让我们来看一下最后取得的效果。用了新的算法之后，我们可以看到系统稳定在一定的范围之内，同时当问题机器恢复之后，流量也能够很快的恢复。



从近几年双 11 零点的业务稳定性上来看，全链路压测是一个明显的分水岭，在全链路压测之后整个站点的稳定性明显好于全链路压测之前。全链路压测已经成为阿里巴巴大促备战的必要环节，无论是双 11 大促、双 12 大促，还是平时一些比较小的促销活动，每一次活动之前都会进行好几轮的全链路压测来对系统进行一次全方位的模拟验证，提前暴露各个环节的问题。全链路压测的诞生使得阿里大促备战的系统

稳定性有了质的提升，被誉为大促备战的核武器。

除了全链路压测来验证我们的容量规划的正确性以外，流量控制的策略在我们的大促技术规划时也很重要，限流框架通过自由组合运行状态，调用链路，限流措施的灵活组合，覆盖了多种业务场景。同时，通过动态平衡，可以做到快恢复，最低的减低对用户使用体验的冲击。流量控制和流量压测两者结合，让我们的系统稳定健康地渡过各种极限业务场景。

此外，基于阿里在双 11 大促上的多年的系统高可用保障经验，全链路压测服务 6 月份即将在阿里云上线（在原有云产品 PTS 的基础上进行全方位升级），通过模拟海量用户的大流量场景，全方位验证站点各个环节的可用性。压测平台具备千万／秒的用户流量构造能力；从全国各地的 CDN 节点发起请求，最真实地模拟用户行为；采用直接压测生产环境的方式，精准探测站点的服务能力和性能瓶颈；压测流量与正常流量隔离、不对线上正常的业务和数据造成污染。欢迎大家关注和试用！

纯干货 | 从淘宝到云端的高可用架构演进

阿里技术

近日在 Qcon 开发者大会北京站上，来自阿里巴巴商家事业部技术专家沐剑在专场分享了题为《高可用实践：从淘宝到上云的差异》的演讲，主要介绍了其近几年在阿里电商平台及阿里云上的高可用设计的经验，分为两个部分：第一部分主要包括传统的淘宝店铺稳定性体系的建设及相关的基础链路设计、缓存和容灾方案的设计及部署；第二部分主要包括目前公有云高可用设计的主要思路、经典故障及应对措施等。

演讲全文：

沐剑

大家好，我今天分享的题目是《高可用实践：从淘宝到上云的差异》，取这个标题是因为会涉及到两个方面内容，一方面以淘宝为例子，传统的 IDC 的时候，我们稳定性是怎么做的，另外在云计算背景下，有很多创业公司是基于阿里云这样的公有云基础设施做研发，在公有云的环境下怎么做好我们系统的高可用。

长期做稳定性的人会有一些职业病，记得去年冬天有个周末，我要寄快递，穿着

睡衣在门口填快递单，那时候我家里养了一只猫，因为怕猫跑出去，就把门关上了。寄完快递口袋一掏发现自己没带钥匙，冷静了 3 秒钟，打车到公司刚巧碰到同事，看我穿着睡衣来公司了问我什么情况，我说家里钥匙忘带被锁在门外面了，不过没事，我还有一把 backup 钥匙放在公司。生活中很多时候都需要有一个 backup。

我的花名叫沐剑，2011 年加入淘宝做评价系统，2012-2015 年在店铺平台，负责店铺的前台浏览系统和后台的 RPC 服务，以及一些性能优化、双 11 保障的事情。到了 2015 年开始到了 TAE 团队，开始负责云端架构及整体高可用方案，TAE 的升级版 EWS 现在也在聚石塔上面帮大量 ISV 和创业公司解决运维部署、自动化监控和性能分析等等问题。去年我是作为阿里商家事业部双 11 作战项目研发的 PM。2017 年我开始接手商家营销团队。在阿里五六年的经验，其实就做了几件事，比如连续五年参加了双十一的核心备战，然后像去 IOE、异地多活，全链路压测、安全混合云、容器服务等项目参与设计和实施。

首先我会从淘宝店铺角度分享，以前在店铺是怎么样做双 11 保障的，后面是一些公有云相关的内容。这是一个淘宝的店铺系统，这套系统是一个非常典型的高并发的浏览系统，在前几年的双 11 峰值有 20 万次的 Web 页面请求，平均一个页面对应了 20 次的 RPC 调用，这个时候对于整个系统的集合来说每秒的 QPS 是 400 万次，这中间就会涉及到缓存、数据库以及其它二方的 RPC 调用，对于这样的系统来说，在性能、稳定性和体验间要做一个平衡，既不能纯用太多的机器死扛这个访问量，又要保障用户的体验。

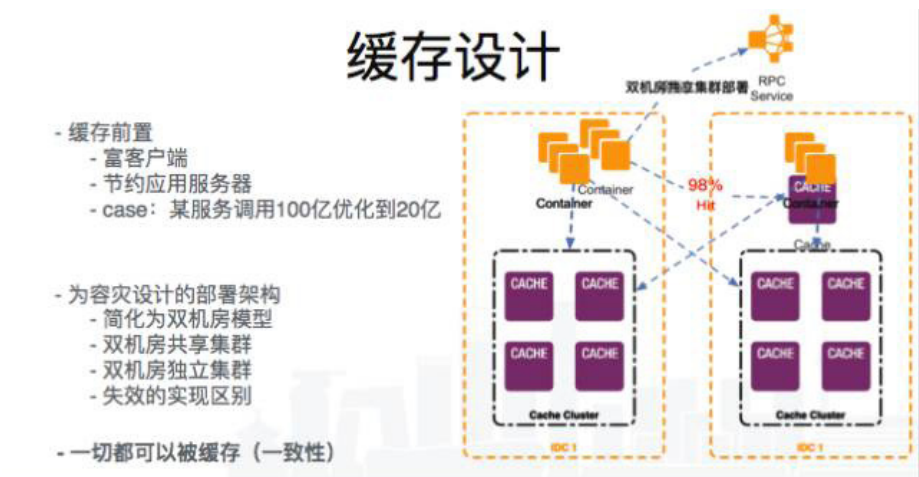
从请求链路来说，首先 DNS 把 CDN 的 VIP 解析出来，分布在全球不同的区域，CDN 回源到接入层分别经过 4 层和 7 层的负载均衡，近几年会发现 CDN 这个行业早已不仅仅局限做 CSS/JS 等静态资源的缓存，也承担了一些动态加速和收敛的特性，所以我们是通过 CDN 做域名收敛，收敛后会把这个流量发到统一接入层，然后到应用集群，后面再经过应用存储、Cache 这些服务。

当我们在做稳定性的时候，会考虑性能和稳定性之间是什么关系，很多人认为这两者是冲突的，比如我要保障一个东西的性能非常高的时候，要牺牲掉很多别的东西，可能你要引入一个非常新的框架或者基础设施来提升性能，但它的稳定性可能是不那么成熟的，但是从容量规划的角度看，只有这套系统性能足够好，才能承担像双 11 那样的大访问量。

店铺也是一套经历了很多年的系统，在应用层上的优化基本上已经做到极致了，我们就转变思路，在操作系统层能不能做一些优化，这里借助了一个比较好的工具 perf，在操作系统层面告诉你系统调用的开销是集中在哪里，从 perf 上就可以定位到有一个百分比，可以看到是比如数组分配还是 GC 产生了大量的开销。 最初我们发现是异常带来的开销，就会看为什么这个系统的异常会导致 20% 以上的 CPU 开销，最后用 BTrace 跟了一下异常的构造函数，发现是我们依赖的开源的三方包里通过异常做控制流，每一次它处理结束的时候，就抛一个 EOFException 出来，这个就导

致了非常大的开销，我们就把开源包替换掉了。 当你依赖一些底层的东西的时候，如果对原理不太了解会给你带来一些意料之外的事情。JVM 里是有一个常量池存储字符串常量的地方，就是一个哈希表，如果说这个表的大小不足够大，就会从哈希查询变成链表查询，性能就会特别低。

再谈一个 warm up 的问题，当我们应用刚刚启动的时候，还没有把字节码编译成 native code，延迟非常高，用户就得到一个有损的服务。我们现在在内部的 JVM 做一个功能，会采集线上系统的调用，把热点方法收集下来做分析，在应用把真实流量挂上去之前，已经预先把所有的热点方法编译成 native code 保证这个性能。开源界也有其他的方案，比如 Azul 的 Zing 有个 ReadyNow，IBM 的 J9 有个 AOT，也是做类似的事情。另外这里我放了一个 Github 的链接，这个 agent 能够让你在 perf 界面里直接看 Java Method 的开销。



谈到缓存，Cache 里有一些小技巧，在做双十一备战时发现一个店铺的基础服务平时日常就每天有 100 亿的调用量，当时是几十台机器估了一下可能要成倍增长，成本是非常高的，怎么解决这个问题，当时写了个富客户端，让业务方先去查我们分布式 Cache，如果命中就直接返回来，如果不命中再走我们的服务端查。这种情况下，只要你能够保证命中率足够高，比如 98% 的命中率，就意味着只有 2% 是需要后端服务器承担剩下的请求，用非常少的服务器去承担非常大的流量，这是成本和性

能间的权衡。

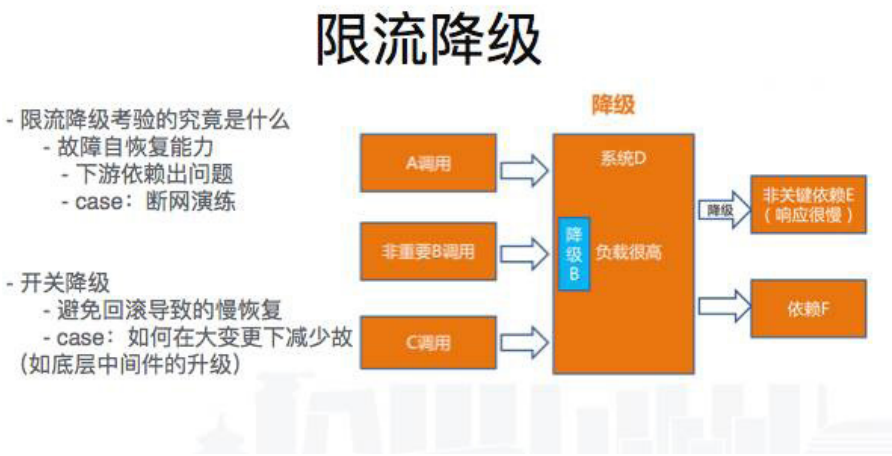
在缓存方面，我们很少会关心缓存的高可用是怎么部署的，它是一个偏运维的内容，我把缓存的部署简化成一个双机房的模型，因为它在高可用里是最简单的场景。对于缓存来说有两种经典部署模式，第一种叫共享集群部署，在 IDC 里我的应用是分机房部署的，Cache 集群也是分机房部署，对于应用服务器来说，两边的 Cache 对他来说逻辑上是一个集群，会往 IDC 1 的 Cache 写一半过去，往 IDC 2 也写一半过去，这种部署的好处在于，机房间网络断掉的时候，有一半的数据是在缓存的，保证一半的数据能够命中，不会直接死掉，另外对成本上相对比较友好，没有浪费任何一个 Cache 的节点，这个 Cache 本身是复用的。但是也正如刚才说的问题，如果中间断掉了，有一半的命中率是有损的，所以就诞生了另外的一个部署模式，就是独立部署，不管你哪个机房挂掉，命中率是基本不变的，两边同时保持了 98% 的命中率，但是它是成本不友好的，两边要同时部署，同时承担副本的作用，并且失效时，要同时失效另外一个 IDC 2，这样才保证一致性。

在缓存上，我认为一切东西都可以被缓存的，通常我们认为缓存跟实际数据库里存在的东西可能是不一样的，有几毫秒的延迟或者怎么样，所以我们设计一个系统的时候，对一致性要求非常高的时候，会倾向于不用缓存，用数据库扛这个流量。但以 MySQL 为例，InnoDB 里有一个很重要的缓存 Buffer Pool，对于一个数据库，在冷库的情况下用一堆 SQL 去查它，和慢慢预热完再查它的时候效果是不一样的，这个是我们当初在做异地多活时面临的一个问题，例如我已经有一个机房，希望建立一个新单元去承担这个机房的流量，当我建完这个单元，把所有的应用都部署好了后，把流量切 50% 过来会怎么样，假设这两个单元的机器数一样，这个单元会挂，因为这个数据库是冷的，缓存是空的，不能承担之前那个单元数据库所能承担的 QPS。

现在业界有很多叫 API 网关或者 CDN，他们在边缘节点也做了一层短暂的 Cache，可能只 Cache 50 或者 100 毫秒，但是当你系统受到攻击的时候可以拯救你后端的应用系统，攻击引发的命中率通常比较高，有这 50 毫秒的缓存，可能后端只有几百个 QPS 过来，那个流量你是可以承受的。

在高可用里两个非常经典的做法是限流和降级，在阿里双 11，有一位老兵说过

一句话，他说当双 11 到来的时候，任何一个系统都可能出问题，你要做的是对你的上游限流，对你的下游限流。怎么理解，当上流的流量超过你的能力的时候就要限流，当下游比如 DBA 告诉你数据库压力很大了，那就对下游限流，只要保证住这个限流，你基本不会挂，每个系统都做到这个的时候，整个系统都是可用的。当流量超出你掌控的时候，这个做法可以让你成为这个暴风下的幸存者。



对限流降级的思考，第一限流降级考验的是什么问题，我认为本质上考验的是故障自恢复能力，在平时工作中会遇到机房断网或者停电，每半个月都会做断网演练，不告诉你发生什么，就把这个网切断，看你的应用 O 不 OK，一般是在晚上两三点，接到很多的机房报警，这个时候看你的架构设计的是否足够可用，如果足够可用就没问题，不会造成什么影响，继续睡觉，如果设计不好，就得爬起来立即处理。

而开关降级最大的作用，比如我们发现一些线上的问题，第一反映是赶紧回滚，但是当你的系统很大的时候，特别像 Java 这种，一个系统启动要启动几分钟，你的回滚完成，20 分钟都过去了，这个过程对用户来说都是有损的，而开关可以在一瞬间把所有的逻辑切到老的。这个是避免回滚时间导致的问题。开关有的时候能救命，如果没有这个开关的话，避免问题放大就只能回滚，所以开关是一个很大的价值所在。

另外一点非常重要的是，在设计一个技术方案的时候，就会把容灾的设计融入到方案里。比如在设计技术方案的时候，在最后一章单独有一个容灾设计，这个节点里任何服务挂掉的时候，你要保持什么样的方式保持这个服务是可用的。

在容灾设计时有几点必须考虑，比如我引了一个新 jar 包或者调了一个新的 RPC 的服务、引入了分布式的存储，以前没用过也不知道它稳不稳定，第一想法是它肯定会挂，它挂了我们怎么做，我们当时在做前台系统的异步化的时候，因为 Redis 支持 map 的数据结构，所以我们就是用 Redis 的 hmget 从这个 map 里拿出部分的 key 减少网卡的流量，但即使这个挂掉了，我们还会走老的 Cache，只不过网卡流量会大一些，但是对用户的服务是无损的，所以这里要考虑如果它挂了怎么做降级，有什么样的恢复流程。

另外是发布计划，在新系统上线时就会关注这些问题，比如这次有没有做数据迁移，比如以前我是 8 个库不够用了我拆到 16 个库或者 32 个库，中间一定是有数据迁移的，涉及到数据迁移一定要有一套对账系统保证这个数据是新数据和老数据是对得平的，不然一定有问题，因为我们是做交易相关的，订单、金额绝对不能出问题。

另外是你的发布顺序是不是有依赖，如果出了问题的时候，谁要先回滚，这里是取决于技术设计。另外是否要通过客服公告的方式告诉外部用户说有 5 分钟的不可用，如果真的有用户打电话有疑问客服同学可以向用户解释。

在高可用这个领域做久了会有一种直觉，这个直觉很重要，来源于你的经验转换成这种直觉，但是对于一个成熟的团队来说，需要把这种直觉转化为产品或工具。有很多牛人他们的技能都只能叫手艺，你需要把这种手艺转换成产品和工具。

2015 年我去做云产品，这里给大家分享下我们是怎么样帮客户包括我们的系统在云上是做高可用的。

首先看两个经典故障案例，第一个是 Gitlab 生产数据库删了，它恢复了很久，Snapshot 等全都没有生效，做了五六层的备份也都没有什么用。这个事情说明第一我们的故障要定期演练，比如中间件在做的线上故障演练，你说你的系统可用性好，我把这个主库断了，虚拟机挂掉几台试试，做这些演练就可以知道你这个容灾体系是不是可靠的，如果没有这个演练的话，当真正的故障发生时你才会发现这个东西是不 OK 的。

另外一个很典型的问题，Gitlab 对备份的原理是不够了解的，比如当时用的 PostgreSQL 的一个版本，当时是有问题的，没有验证，开发人员对这个又不是特别了解的情况下就会出现这个问题，这就是为什么要去了解你的依赖以及你依赖的依赖。去年我们做压测，有个应用一边压测一边在优化做发布，发现第一批发的起不来了，就只是改了一两行代码加日志，他就去看什么原因，最后发现依赖的某个 jar 包依赖一个配置，而这个配置在压测中被降级了，一个 jar 包就把应用启动卡住了。如

果在双十一当天或者在平时业务高峰期的时候发现这个问题是来不及修复的。所以这个时候，我们就要求，依赖的二方 jar 包必须看一下里面是怎么实现的，依赖哪些东西。

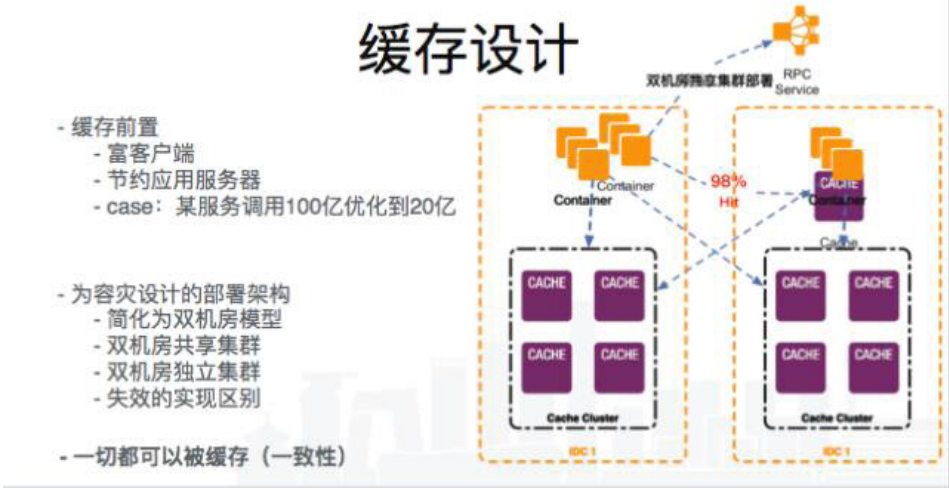
反过来说，别人依赖我的客户端就意味着他不仅依赖着我的服务还依赖着我的缓存，这个缓存出了问题对他也有影响，我们每年双十一前有一个强弱依赖梳理，不仅要梳理自己应用里面的，还有依赖的所有东西都梳理出来，中间任何一个节点挂掉了你应该怎么办，需要给一个明确答复。第二个故障案例是今年发生的，AWS S3 敲错了一个命令把基础核心服务下线了，有一个对象索引服务和位置服务系统被 offline，后来也做了一些改进，每次敲的命令有一个静默期，让你有个反悔的机会，线上有个最小的资源保证服务。

这个给我们带来的启示是什么，云服务本身也是会发生故障的，比如买了云数据库，我们没有办法假设它是 100% 可用的，当它出现问题我们怎么办，是给云厂商提工单说什么时候能恢复，还是我自己能够有一个容灾的方案解决这个问题。从 2015 年开始，我们越来越多地发现，对架构可用性最大的威胁是什么？在市政施工里一定几率就会莫名其妙搞出光缆被挖断等故障，我们不得不考虑，当云服务本身出现问题我们该怎么办。

所以我们需要有一套面向云的高可用架构。在很早以前有厂商提出类似 SDN 的一个概念，叫 SDI——软件定义基础设施，过去我们发现只有大厂可以做这个事情，设计一套很复杂的管理系统帮他实现，这里放一个路由器，这边放一台虚拟机，可以通过软件的控制流去控制这个东西。但是在云的时代，资源变得很容易获得。以阿里云为例子，可以用 API 随时创建新的虚拟机，新的负载均衡，或者是新的存储，都可以通过 API 随时创建随时销毁，这对于你的基础设施的灵活度非常有好处。

以前有的人会觉得，性能问题或者容量问题应该通过性能优化的方式解决，通过一些黑科技方式解决，加机器会觉得很 low。但我觉得一个问题如果能简单用加机器来解决是很不容易的，意味着对你的整个架构的水平扩展性要求非常高，而且解决效率很高，加机器就解决了，而对一些中心化的系统来说就比较麻烦，加机器都加不了，可能要把机器关掉升配置再重新拉起来。所以我们说，在公有云上面，在资源如

此容易获得的情况下要充分利用这个特性，要做一个能够做水平扩展的架构。



那么第一步要做什么，前两年很火的容器、微服务，本质上都是解决了是无状态的应用怎么做自动化的扩容这个问题。右边这个图上面，上面是一个负载均衡，中间是一个前端的服务，后端是一个无状态的后端服务，底层是 MQ、对象存储、数据库这些东西，如果我们能够把前端和后端的无状态服务第一步先容器化，就可以做到当流量过来的时候，只要后端的存储没有问题，整套架构就是能够水平扩展的。

从去年公开的报道和故障来看，很多人有个误会是说云厂商的机器应该是不会挂的，我买了一台云厂商的虚拟机应该是随时可用的，即使不可用云厂商也要帮我解决热迁移的问题，热迁移在业界是很复杂的问题，不光涉及到磁盘存储的迁移，也涉及到内存是要做迁移的，可能还要用 RDMA。并且对于传统的 IDC 来说，不管物理机还是虚拟机都是有可能挂的，对云也不例外。当我们在使用公有云服务的时候，都是会挂的，这是个心理准备。不光是机器，包括负载均衡是不是也有可能挂，下面的消息队列或者数据库是不是也有可能会挂，当你基于任何东西都可能会挂的前提设计一个系统的时候，才能真正做到这个系统在任何情况下都不会受底层云服务的故障影响。

而对于不同的云服务来说是有不同的容灾策略。比如一台虚拟机挂了，通常来说负载均衡不管是 4 层还是 7 层都会做健康检查，挂了健康检查不通自动会把流量切断。 如果我的负载均衡挂了怎么办，如果 DNS 有健康检查那就方便了，如果没有的话可能就要设计一个旁路系统解决这个问题，发现这个已经不通了，就自动把它从 DNS 上摘掉。

不管是云服务发生故障还是自己应用发生故障有个大原则是如何最快速解决问题，就是一个字，切。为什么要做异地多活，为什么要把流量往各个地方引，切流量是解决问题最快的，把坏流量切到好的地方马上就解决了，如果你要等定位问题解决问题再上线客户就流失掉了。对淘宝来说也是一样，当某一个单元低于我们认为的可用性的时候，我们会把这个单元的流量引到另外一个可用的单元，当然前提是那个单元的容量是足够的。

弹性是不是万能的？所有的云服务都是弹性的，弹性其实不是万能的，容量规划仍然是有必要的，不然就没必要做双十一备战了。这里有一个你需要付出的代价，弹性的过程往往是需要时间的，那么容量规划在这个环节中起到的作用就很重要，当真的逼不得已的时候，我要扩容了，怎么保证我扩完容之前系统不雪崩？就是要结合之前的限流，尽可能保障每个客户得到他应有的服务，但是也要保障系统的稳定性。

Region 和 Availability Zone 这两个，当实践的时候，购买虚拟机、负载均衡或者数据库，一定要选择多可能区的服务，比如阿里云买 SLB 是可选可用区的，有个

主可用区和副可用区，如果一边挂了可以切换到另外一边，RDS 也是一样的。

这幅图是一套典型基于公有云的一套架构，不叫异地多活，应该叫跨区域设计。左右两个大框是两个城市，左边是北京，右边是上海，每个里面又有不同的机房可用区去承担这个流量，假如北京的挂掉了，就切，原来是在 A 可用区，就切到 B 可用区，这时候 A 可用区没有流量进来，通过切换的方式能够把这个服务快速恢复。下面画了一个跨区域复制，我们在异地多活项目里，涉及到了跨城市跨数据中心的复制，比如我的北京是提供写服务的，上海要提供读服务，就要通过这种方式同步数据过去。

最后，我们在招聘，招一些志同道合的小伙伴。我看到很多人做招聘，他们都在说我们想要什么样的人，需要你具备什么样的技能，我的想法可能不太一样，我会先思考你到我们团队能得到什么，我能给你什么东西。那么第一可以参与到阿里最前线的作战经验，今年 2017 年双十一大促可以跟我们一起做，第二我们团队内部有非常好的氛围，可以听到来自阿里巴巴各领域的专家像鲁肃、毕玄等大师的分享，第三可以飞速成长，一对一师兄带领，快速度过适应期。新零售新技术，会创造出下一个时代的商业文明，对于技术人员来说我们不光有技术的思维，也要有业务和商业的思维来培养自己。

最后是我的邮箱，如果大家想交流可以发邮件给我：mujian.wcc@alibaba-inc.com

破解世界性技术难题！GTS 让分布式事务简单高效

阿里技术

近日，2017 云栖大会 · 深圳峰会如期举行，多项阿里云新产品对外发布。在企业级互联网架构分会场，来自阿里中间件（Aliware）的技术专家及合作伙伴，为现场参会嘉宾带来最新的传统 IT 架构到企业级互联网架构跨越式升级、实现互联网转型的产品及解决方案。其中高级技术专家姜宇在分享中带来的 Aliware 新产品—全局事务服务（Global Transaction Service ，简称 GTS），在分布式事务处理上带来的高性能和技术创新令到场参会的各路技术专家眼前一亮。

Aliware 新成员—全局事务服务 GTS 技术分享现场

分布式事务背景

OLTP 领域中很多业务场景都会面临事务一致性的需求，传统业务系统常以单体应用形式存在，只需借助特有数据访问技术和框架，结合关系型数据库自带的事务管理机制来实现事务一致性的要求。而目前大型互联网应用和平台往往是由一系列分布式系统构建而成，平台和技术架构也是流派纷呈。

尤其是微服务架构盛行的今天，一个看似简单的功能，内部可能需要调用多个“服务”并操作多个数据库或分片来实现，单一技术手段和解决方案已无法满足这些复杂应用场景。因此，分布式系统架构中分布式事务是一个绕不过去的挑战。什么是分布式事务？简单的说，就是一次大操作由不同小操作组成，这些小操作分布在不同服务器上，分布式事务需要保证这些小操作要么全部成功，要么全部失败。

本质上来说，分布式事务就是为了保证不同数据库或消息系统的数据一致性。

分布式事务三大难题：一致性、高性能和易用性

分布式系统的事务一致性本身是一个技术难题，没有一种简单完美的方案能够应对所有场景，很难兼顾事务一致性，高性能与易用性。三者缺一，则适用场景大大受限，实用价值不高。

首先是一致性：要求在各种异常情况下保证数据是强一致的。目前最常见的一致性解决方案是最终一致性方案，通常是结合消息中间件实现，在互联网企业中广泛使用。最终一致性实现方案比较复杂，开发、运维成本高，并且与强一致相比，业务上是受很多限制的。

其次是高性能：目前基于 XA 协议的两阶段提交是最常见的分布式事务解决方案，但 XA 类产品的典型不足是性能低下，这对于互联网大并发需求下的多数企业是无法接受的。国外具有几十年历史和技术沉淀的基于 XA 模型的商用分布式事务产品，在相同软硬件条件下，开启分布式事务后吞吐经常有数量级的下降。

第三是易用性：为了满足一致性和高性能要求，出现了一些特定场景下的分布式事务方案，但通常会限制用户用法，对业务侵入性强，无法做到简单易用，带来更多开发成本。

世界级应用场景，催生世界级分布式事务解决方案

早期的阿里巴巴集团随着业务高速发展，内部不断涌现各种典型的分布式事务需求，比如阿里内部广泛使用的 TDDL 分库分表所带来的分库间数据不一致问题，HSF 服务化后所带来的服务链路上数据不一致问题等。在这个过程中，各业务技术团队利用现有中间件技术手段实现分布式事务处理，但这些手段都较为复杂，工作量大，对应用侵入严重，有些适用场景还有限制。

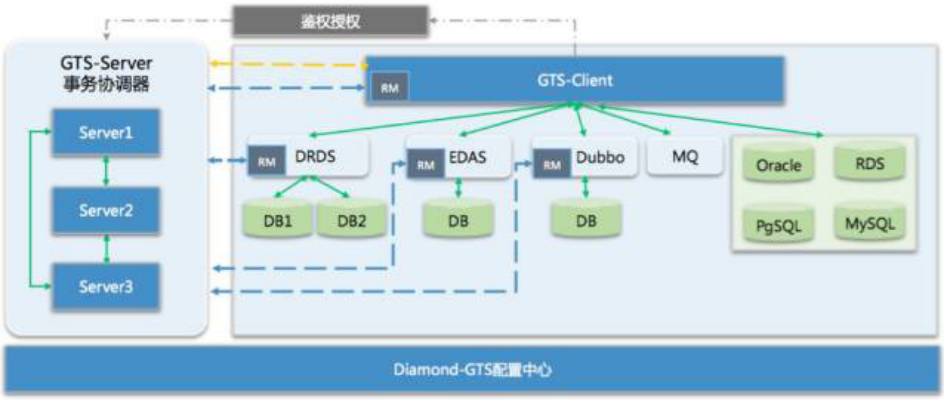
2014 年 5 月开始，阿里中间件（Aliware）内部命名为 TXC 的分布式事务中间件开始研发，同年 10 月 1.0 版本发布，分布式事务功能已经具备，但性能还有局限，只适合于吞吐量较小的场景；2015 年 12 月，TXC 2.0 版本发布，相比 1.0 版本性能提升 10 倍以上，在阿里内部多条业务线得到部署。

通过部署 TXC，应用只需极少的代码改造和配置，即可享受分布式事务带来的便利。TXC 作为阿里内部为解决分布式数据强一致性问题而研发的分布式事务中间件，彻底解决了分布式事务数据一致性的问题，简单易用，先后在淘宝，菜鸟，淘票票和村淘等多个业务的核心系统上得到部署和验证。

顺应云时代潮流，GTS 应运而生

从 2016 年年中开始，在阿里内部一直接受锤炼的分布式事务中间件 TXC 在

2.0 版本后，随着阿里中间件上云热潮，开始通过专有云输出，并得到了市场极大认可，适用场景得到进一步拓展，全面涵盖电商、物流、金融、零售、政企、游戏、文娱等领域。2017 年 2 月，TXC 2.0 通过阿里云对外公测，外部改名为全局事务服务（Global Transaction Service，简称 GTS）。



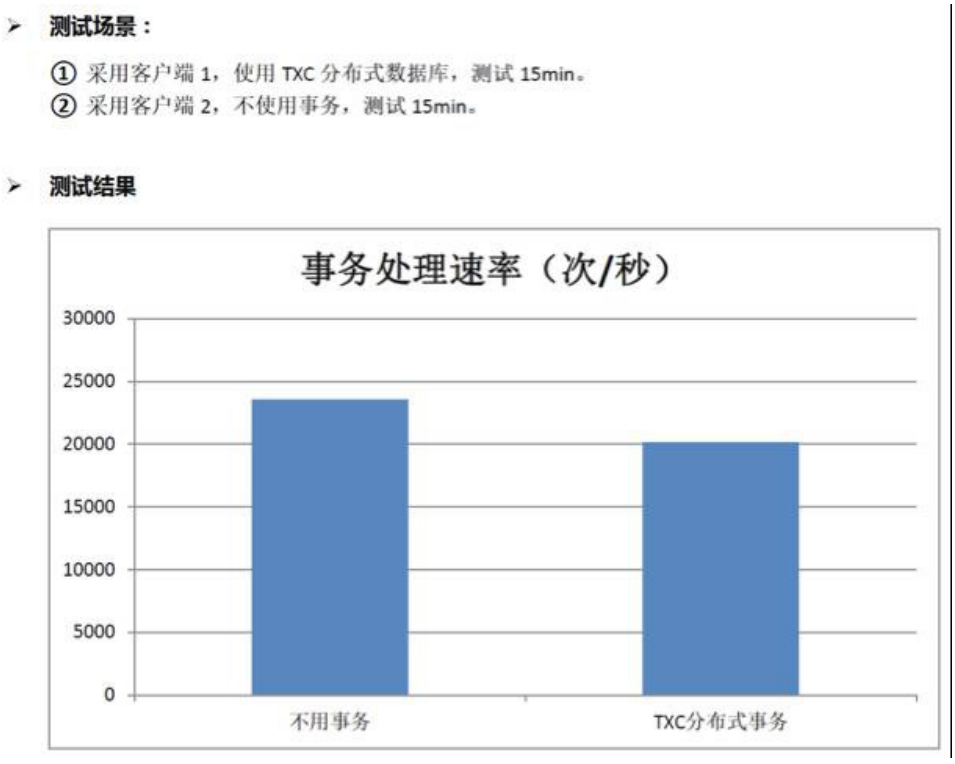
GTS 总体架构图

在整体架构方面，GTS 由三个组件组成：客户端（GTS-Client），资源管理器（RM），事务协调器（GTS-Server）。客户端与事务协调器间，资源管理器与事务协调器间都是通过 GTS 分布式事务协议进行通信。客户端负责界定事务边界，开启 / 提交 / 回滚全局事务，资源管理器负责管理资源，支持的资源包括：DRDS，Oracle，MySQL，RDS，PostgreSQL，H2，MQ，后续计划根据实际业务需求支持更多类型资源。事务协调器，也就是 GTS 服务器，是分布式事务处理的大脑，负责协调整个事务过程。GTS 事务通过 RPC 框架和消息中间件进行事务传递，把整个业务调用链路或者消息链路串成一个分布式事务，极大简化应用开发。

在高可用方面，GTS 支持同城容灾与两地三中心容灾，可保证各种异常情况下的数据一致。在易用性方面，GTS 对业务无侵入，真正做到业务与事务分离，开发者可以集中精力于业务本身。在技术创新方面，GTS 也走在了行业前沿。项目负责人阿里高级技术专家姜宇（花名于皋）拥有 13 项分布式事务的核心技术专利，研发团队的技术专家张松树也有 3 篇专利。通过大量的专利技术，精妙的算法，与精巧的分

布式事务私有协议，GTS 取得了超强的性能。

另外，在部分严苛的行业应用场景，比如金融用户的资管项目分布式事务场景下，GTS 也经历了严格的测试，按照用户要求顺利完成功能性、稳定性和性能测试。下图是一个典型性能测试场景数据，从实测数据可以看出，开启 GTS（TXC）分布式事务后性能下降不明显。目前 GTS 已经在资金业务上有实际应用，线上大量真实数据验证了 GTS 的高效可靠。



GTS 典型性能测试场景数据

性能优异，业务场景广泛

作为新一代企业级分布式事务服务产品，全局事务服务 GTS 兼顾了事务一致性，高性能与易用性。在满足事务 ACID 的前提下，普通配置的单服务器就可以达到 15000TPS 以上的超强性能（两个小时内完成 1 亿多笔业务），3 台 8 核 16G 内存虚

机组成的服务器集群可以支撑 1 万 TPS 以上的分布式事务，与同类产品相比，性能优势明显。另外简单易用对业务无侵入，为广大企业大幅降低开发成本，业务场景非常广泛：

1、跨多分库的分布式数据库事务场景：关系型数据库普遍支持事务，能够满足事务内的 SQL 要么全部成功、要么全部失败。但客户从单机数据库往分布式数据库迁移的情况下，原有的一个事务往往会被拆分为多个分库上的事务。由于网络的不可靠性，容易出现部分分库上成功，部分分库上失败的情况。GTS 结合 DRDS 可彻底解决了这一问题。

2、跨多数据库的事务场景：复杂的业务系统经常会使用多个数据库，甚至多种类型的数据库，比如企业中 Oracle，MySQL 和其他关系型数据库并存的情况时有发生。业务同时操作多个数据库的情况下，一旦发生先提交的事务成功、后提交的事务失败，就很难解决。GTS 支持各种常见关系型数据库，并提供多数据库间的事务保证。

3、跨数据库系统、消息系统的事务场景：消息系统被广泛地用于系统间解耦，一般先执行一段业务逻辑，执行成功会向消息系统发送一条消息，用于通知或触发下游业务。这个场景下，如果业务逻辑执行成功、消息发送失败，则业务不完整；如果先发送消息，但执行业务逻辑失败，同样存在问题。GTS 提供了针对消息系统以及常见关系型数据库的操作入口，保证数据库操作和发送消息要么同时成功、要么同时失败。

4、跨服务的事务场景：随着业务复杂度提升，大多企业会对业务进行服务化改造。可能存在服务一操作 MySQL 和 DRDS，服务二操作 Oracle，要求两个服务操作要么同时成功、要么同时失败，否则会造成业务数据的不一致。GTS 可以很方便地进行跨多个服务的分布式事务。

依托阿里中间件 (Aliware)，打造世界一流企业级互联网架构平台

据 GTS 项目负责人姜宇介绍，“GTS 作为一款高性能、高可靠、接入简单的分布式事务中间件产品，可与 DRDS、RDS、Oracle、MySQL、PostgreSQL、H2

等数据源，EDAS、Dubbo 及多种私有 RPC 框架，MQ 消息队列等中间件产品配合使用，可轻松实现分布式数据库事务、多库事务、消息事务、服务链路级事务及各种组合。策略丰富，易用性和性能兼顾，将真正完善阿里云中间件产品线。”

GTS（TXC）的研发依托于阿里中间件 (Aliware) 团队，中间件技术部是阿里巴巴集团生态系统的技术基石，为集团各大业务群提供可靠、高效、易扩展的技术基础服务；并在此基础上打造世界一流的中间件产品、高可用架构基础设施和企业级互联网架构平台，为全球企业和客户提供服务。

更多 Aliware GTS 产品服务和技术细节，请访问官网：

https://www.aliyun.com/aliware/txc/

如何打造支撑百万用户的分布式代码托管平台?

屹恒

前言：过去一年中，阿里巴巴集团 GitLab 请求量增长 4 倍，项目数增长 130%，用户数增长 56%，在这样的增速下，系统调用的正确率却从 99.5% 提升到了 99.99% 以上!

阿里巴巴集团 GitLab 的架构如何一步步炼成，成为支撑百万规模的用户体量? 下面让我们一起来共同学习。

一、背景介绍

毋庸置疑，代码是 DevOps 流程的起点，是所有研发流程的基础；代码托管为代码“保驾护航”，确保代码的安全性、可用性，同时提供围绕代码的一些基础服务，如 MR、Issue 等等。

阿里巴巴集团 GitLab 是基于 GitLab 社区版 8.3 版本开发，目前支撑全集团数万规模的研发团队，累计创建数十万项目，日请求量千万级别，存储 TB 级别，早已超过了 GitLab 社区版承诺的单机上限能力，且增长速度迅猛。

面对这种情况，顺理成章的做法就是——扩容。然而非常不幸，GitLab 的设计没有遵守 Heroku 推崇的“The Twelve-Factor App”的第四条：“把后端服务当作附加资源”（即对应用程序而言，不管是数据库、消息队列还是缓存等，都应该是附加资源，通过一个 url 或是其他存储在配置中的服务定位来获取数据；部署应可以按需加载或卸载资源），具体体现是：

- Git 仓库数据存储于服务器本地的文件系统之上
- GitLab 所依赖的三个重要组件：libgit2、git、grit 也都是直接操作在文件系统上 GitLab

所以 GitLab 社区版是基于“单机”模式设计，当存储容量和单机负载出现瓶颈时，难以扩容!

二、面对挑战

2015 年初，阿里巴巴集团 GitLab 的单机负载开始呈现居高不下的情况，当时的应对方案是同步所有仓库信息到多台机器，将请求合理分配到几台机器上面从而降低单机的负载。然而这个方法治标不治本：

- 系统运行过程中的数据同步消耗资源和时间，不可能无限制扩充机器
- 这种方案暂时缓解了单机负载的问题，但对于单机存储上限的问题束手无策

2015 年中，团队正式启动了第一次改造尝试，当时的思路是去掉对本地文件系统的依赖，使用网络共享存储。

然而由于本地缓存等问题的限制，网络共享存储的方案在性能上出现较明显性能问题，且大都为基于 C/C++ 的底层改动，改造成本出现不收敛情况。而当时集团 GitLab 服务器在高峰期 CPU 屡屡突破 **95%** 甚至更高的警戒值，而高负载也导致了错误请求占比居高不下，无论是对上游应用的稳定性还是对用户体验都提出了严峻挑战。

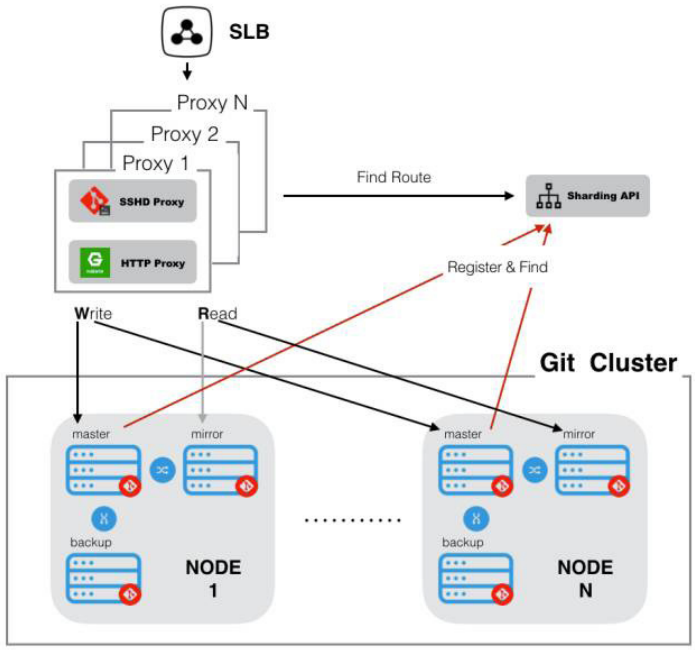
2016 年 8 月起新一轮改造开始。

三、改造方案

既然共享存储的方案暂时行不通（后续如果网络存储的读写性能有质的提升，未尝不是好的方式），首先明确了唯有分布式或者切片才能解决问题的基本思路。

我们注意到，GitLab 一个仓库的特征性名称是”namespace_path/repo_path”，而且几乎每个请求的 URL 中都包含着个部分（或者包含 ID 信息）。那么我们可以通过这个名称作分片的依据，将不同名称的仓库路由到不同的机器上面，同时将对于该仓库的相关请求也路由到对应机器上，这样服务就可以做到水平扩展。

下面通过一幅图介绍一下目前集团 GitLab 在单个机房内的架构。



3.1 各个组件的功能

- Sharding-Proxy-Api 用于记录仓库与目标机器之间的对应关系，可以看作切片的大脑
- Proxy 负责对请求做统一处理，通过 Sharding-Proxy-Api 获取信息，从而将请求路由到正确的目标机器
- Git Cluster 由多组节点构成，每组节点内有三台机器，分别为 master，mirror 和 backup。其中 master 主要负责处理写（POST/PUT/DELETE）请求，mirror 主要负责读（GET）请求，backup 作为该节点内的热备机器

说明

- master 在处理完写请求后，会同步更新此次变更到 mirror 和 backup 机器，以确保读请求的正确性和热备机器的数据准确
- 之所以没有采用双 master 的模式，是不想造成在脏数据情况下，由于双向同步而造成的相互覆盖

3.2 保证方案可用

1. 如何确保切片信息准确

Sharding-Proxy-Api 基于 martini 架构开发，实时接收来自 GitLab 的通知以动态更新仓库信息，确保在 namespace 或 project 增删改，以及 namespace_path 变更、仓库 transfer 等情况下数据的准确性。

这样的场景下，等于每次请求多了一次甚至多次与 Sharding-Proxy-Api 的交互，最初我们曾担心会影响性能。事实上，由于逻辑较为简单以及 golang 在高并发下的出色表现，目前 Sharding-Proxy-Api 的 rt 大约在 5ms 以内。

2. 如何做到切片合理性

海量数据的情况下，完全可以根据 namespace_path 的首字母等作为切片依据进行哈希，然而由于某些名称的特殊性，导致存在热点库的情况（即某些 namespace 存储量巨大或者相应请求量巨大），为此我们为存储量和请求量分配相应的权重，根据加权后的结果进行了分片。目前来看，三个节点在负载和存储资源占用等方面都比较均衡。

3. 如何处理需要跨切片的请求

GitLab 除了对单 namespace 及 project 的操作外，还有很多跨 namespace 及 project 的操作，比如 transfer project，fork project 以及跨 project 的 merge request 等，而我们无法保证这些操作所需的 namespace 或 project 信息都存储在同一台机器上。

为此，我们修改了这些场景下的 GitLab 代码，当请求落到一台机器同时需要另一台机器上的某个 namespace 或 project 信息时，采用 ssh 或者 http 请求的方式来获取这些信息。

最终的目标是采用 rpc 调用的方式来满足这些场景，目前已经在逐步开展。

3.3 提升性能

1.ssh 协议的替换

目前阿里巴巴集团 GitLab 提供 ssh 协议和 http 协议两种方式，供用户对代码库进行 fetch 和 push 操作。其中，原生的 ssh 协议是基于操作系统的 sshd 服务实现，

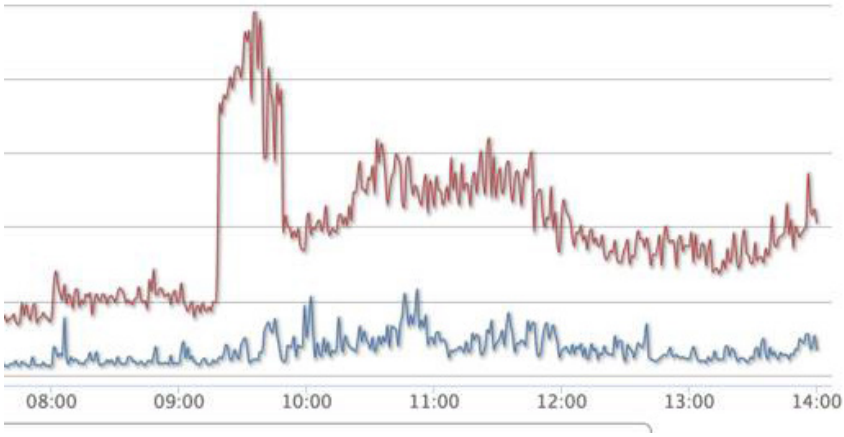
在 GitLab 高并发 ssh 请求的场景下，出现了部分 bug，由此产生的问题是：

- ssh 协议登陆服务器变慢
- 用户通过 ssh 协议 fetch 和 push 代码时速度变慢

为此，我们采用 golang 重写了基于 ssh 协议的代码数据传输功能，分别部署在 proxy 机器以及各组节点的 GitLab 服务器上。由此带来的好处有：

- 机器负载明显降低
- 消除上述 bug
- 在 ssh 服务发生问题的情况下，仍旧可以通过 ssh 登陆（使用原生）服务器，以及重启 sshd 服务不会对服务器本身造成影响

下图是 proxy 机器采用新 sshd 服务后的 cpu 使用情况：



2. 个别请求的优化和重写

对于一些请求量较大的请求，例如鉴权、通过 ssh key 获取用户信息等接口，我们目前是通过文本转 md5，加索引等方式进行性能优化，后期我们希望通过 golang 或 java 进行重写。

3.4 确保数据安全

1. 一主多备

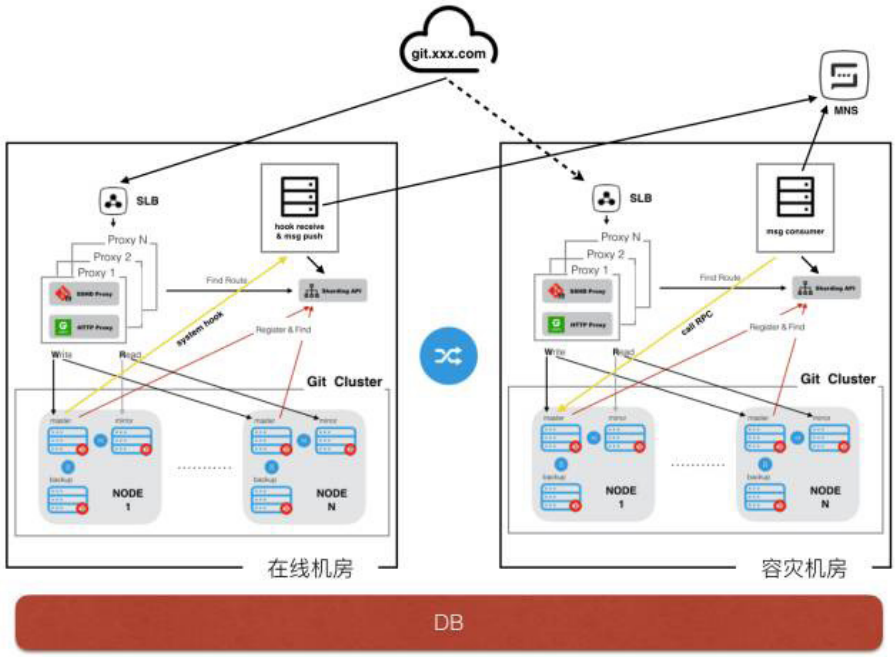
如上面提到的，目前阿里集团 GitLab 的每组分片节点包含有三台机器，也就是

相对应的仓库数据一备三，即使某一台甚至两台机器发生磁盘不可恢复的故障，我们仍旧有办法找回数据。

2. 跨机房备份

刚刚过去的三月份，我们完成了阿里巴巴集团 GitLab 的跨机房切换演习，模拟机房故障时的应对策略。演习过程顺利，在故障发生 1min 内接到报警，人工干预（DNS 切换）后 5min 内完成机房间流量切换。

多机房容灾的架构如下图所示：



保证准实时的仓库数据同步是机房切换的基础，我们的思路按照实际需求，由容灾机房机器主动发起数据同步流程，基本步骤是：

- 利用 GitLab 的 system hook，在所有变更仓库信息的情景下发出消息（包含事件类型及时间包含数据等）
- 同机房内部署有 hook 接收服务，在接收到 hook 请求后对数据进行格式化处理，并向阿里云 MNS（Message Notify Service）的相关主题发送消息

- 容灾机房内，部署有消息消费服务，会订阅相关的 MNS 主题以实时获取 online 机房发送到主题内的消息，获取消息后调用部署在容灾机房 GitLab 节点机上的 rpc 服务，主动发起并实现数据同步的逻辑

hook 接收、消息消费以及 GitLab 节点机上的 rpc 服务，均由 golang 实现。其中 rpc 服务基于 grpc-go，采用 protobuf 来序列化数据。

通过一段时间的运行和演习，已经确定了方案切实可行，在数据校验相关监控的配合下，容灾机房可以准实时同步到 online 机房的数据，且确保 99.9% 至 99.99% 的数据一致性。

3.5 如何提升系统的可用性

1. 日志巡检

面对集团 GitLab 每天产生的大量日志，我们使用阿里自研的监控工具进行日志监控，对系统产生的 5xx 请求进行采集和报警，然后定期去排查其中比较集中的错误。经过一段时间的排查和处理，5xx 错误日志大致可以分为：

- 分布式改造带来的跨分片操作的 bug
- GitLab 本身的 bug
- 高并发情况下带来的系统偶发性故障
- 数据库相关的错误等

2. 服务器监控

无论系统多少健壮，完备的监控是确保系统平稳运行的基础，既可以防患于未然，也可以在问题出现时及早发现，并尽可能减小对用户的影响。目前阿里巴巴集团 GitLab 的监控主要有：

- 机器 cpu，内存、负载等基本指标的监控
- ssh、ping 等网络检测信息（用于判断是否宕机，后将详述）
- 服务器各个端口的健康检查（80，90xx，70xx 等等）
- 异步消息队列长度的监控
- 数据库连接的检查
- 错误请求的监控

• Sharding-Proxy-Api 与 GitLab 的数据一致性校验

很自豪的一点是，我们经常可以根据报警信息，先于用户发现问题。

3. 单机故障的自动切换

虽然监控足够完备，但谁也不能保证服务器永远不宕机，因此我们在同一组节点中保有一台 backup 机器以应对服务器故障。会有专门的 client 定期通过 API 轮询监控平台提供的机器监控信息，当确认机器宕机时（ssh 和 ping 同时不通，且持续时间超过 2min）自动触发机器角色的互换，包括 master 与 backup 互换，mirror 与 backup 互换等。

通过演习，我们已经具备了单机故障时 5min 内全自动实现机器切换的能力。

4. 机房故障时的切换，即前述的跨机房切换。

3.6 单元化部署

单元化架构是从并行计算领域发展而来。在分布式服务设计领域，一个单元（Cell）就是满足某个分区所有业务操作的自包含的安装。而一个分区（Shard），则是整体数据集的一个子集，如果你用尾号来划分用户，那同样尾号的那部分用户就可以认为是一个分区。单元化就是将一个服务设计改造让其符合单元特征的过程。

为了实现单元化的目标，我们在最初设计时就往这方面考虑。比如跨机房备份中，消息消费应用需要调用 Sharding-Proxy-Api 获取 rpc 服务的地址时，尽可能做到数据在单机房内闭环。这样在满足单元化要求的同时，也可以在机房故障时，尽量不影响已进入队列的消息在消费时出现数据断流。

现在阿里巴巴集团 GitLab 在架构上已经基本具备了单元化部署的能力，这样的情况下，无论是后续与阿里云合作对外提供服务，还是当收购海外公司需要单独搭建新服务时，都不会遇到问题。

四、未来的改进

4.1 偶发的 cache 大量释放

由于 GitLab 有大量的 IO 操作，使得系统占用 cache 的数值巨大，也正是因为 cache，系统的性能得到保证。然而成也 cache 败也 cache，为了确保系统不会发

生 OOM，我们设定了 vm.min_free_kbytes，当 cache 占用过多且需要继续申请大片内存时，会触发 cache 的释放，势必会影响释放瞬间请求处理能力（通过日志分析得到，此瞬间的处理能力仅为 cache 存在时的 1/2 左右），直接后果是该瞬间的请求堵塞，甚至出现部分 502。

为此我们咨询了系统部的同学，根据他们的建议修改了部分内核参数（目前仍没有根治），后续会尝试升级内核的方式，也希望遇到过类似问题或对这方面问题有研究的同学，把你的秘籍传给我们。

4.2 自动化运维

目前集团 GitLab 的发布主要靠手，在分布式架构下，机器势必越来越多，全自动化的发布、扩容机制，是我们需要完善的地方。

4.3 rpc 方案的最终落地

如前所述，只有最终实现了全局的 rpc 替换，才能将 web 服务所消耗的资源与 Git 本身消耗的资源进行分离，阿里巴巴集团 GitLab 的分布式改造才能算最终结束。

五、结语

监控及日志数据对比显示，过去一年中阿里巴巴集团 GitLab 请求量增长 4 倍，项目数增长 130%，用户数增长 56%，在这样的增速下，系统调用的正确率却从 99.5% 提升到了 99.99% 以上，这些数字印证了我们方案的可行性和可用性。

接下来的时间里，小伙伴们会为继续代码服务的创新而努力。“高扩展、分布式、响应式、大文件、按需下载、流控、安全、数据化、单元化”，有些我们做到了，有些是接下来努力的方向。

很自豪，今天的我们终于可以有底气地承诺：现在阿里巴巴集团 GitLab 的架构，已经足够支撑百万规模的用户体量，在满足集团业务发展的前提下，会逐步通过阿里云 code 为更多云上开发者提供服务，共同打造云上的协同研发生态！

大牛观点

达摩院：阿里巴巴的科技雄心

阿里技术

10 月 11 日，以“飞天 · 智能”为主题 2017 杭州 · 云栖大会在浙江杭州云栖小镇开幕，阿里巴巴董事局主席马云在开幕式上演讲。（本文来源：中国新闻周刊）

四名科学家同时盖上印章后，两名“武者”展开长卷，淡淡的水墨之间，“达摩院”三个大字跃然纸上。

在武侠世界里，“达摩院”代表着武林绝学和至尊，这是阿里巴巴董事局主席马云为新成立的研究院取的名字。就在 2017 云栖大会前两周，阿里巴巴集团首席人力官童文红给马云打电话，讨论研究院如何命名，马云灵光一闪，“达摩院”三字脱口而出。

而在此前长达半年的酝酿中，阿里巴巴内部都将这个新机构称做“阿里巴巴全球创新研究院”。

“为什么一定要研究院、实验室这样的说法，为什么不能创造一个自己的名字，我觉得达摩院就很好。”马云在给童文红的电话中说，他甚至连英文名都想好了，就是拼音 DAMO。

正如在名称上独辟蹊径一样，马云希望达摩院能走出自己的模式，“我们会学习 IBM，学习微软，学习贝尔实验室，学习在过去人类历史科技发展过程中取得的巨大的经验和教训，但我们必须走出自己的路。”

按照阿里巴巴在云栖大会上的说法，“达摩院”是探索人类科技未来的实验室，阿里巴巴将在研发投入 1000 亿元，用于涵盖基础科学和颠覆式技术创新的研究。

在阿里巴巴董事局主席马云看来，这是 18 岁的阿里巴巴应有的担当精神。他将“达摩院”视为阿里巴巴将留给世界最好的东西之一。有一天即使阿里巴巴不在了，希望“达摩院”还能继续存在。为了做到这一点，“达摩院”必须做到商业与科技、市场与研究的完美结合。

“向技术要红利！”

在十年前，马云可不是这样想的。那时候的他，曾经坚决反对公司有任何研究室、实验室，因为在他看来，当时阿里巴巴还是一个初创公司，在还没有立足之前就考虑研发是大灾难。

在“达摩院”筹备组成员、阿里巴巴技术战略部总监刘湘雯的印象中，早在 2008 年，阿里巴巴就已经从战略层面考虑，从一家电商公司变成一家数据公司。尽管有这样一个愿景，大家却并不知道如何去做。当时，阿里巴巴的平台沉淀了很多数据，怎样去发挥数据的价值，从技术上怎么做，引发了阿里巴巴高层一系列的思考。

最终，马云选择了相信云计算，成立了阿里云计算有限公司。虽然没有被叫做“研究院”，但在刘湘雯看来，这是阿里巴巴第一次从战略上向科技进行转移。至此，阿里巴巴全面进入云计算，对自身的定位也从一家电商公司变成一家数据驱动的公司。

2014 年，阿里巴巴成立了 iDST（Institute of Data Science&Technologies），这是阿里巴巴集团专注于底层数据技术研究的机构。此前，马云已经把下一个时代命名为 DT 时代，也就是数据科技时代。而 iDST 以机器学习、深度学习技术为依托，打造图像视频、语音交互、自然语言理解、智能决策等人工智能核心技术，为阿里巴巴集团的电商、金融、物流、社交、娱乐等业务提供强大的技术后盾。这些 AI 技术通过阿里云平台对外形成产品化的输出。

用刘湘雯的话说，“达摩院”的成立是一个水到渠成的过程，离不开“母体”阿里巴巴的发展。一个公司只有当业务发展到一定的阶段，有足够复杂的场景，足够多的业务体量，才会有足够多的科技难题出现，才能支撑一群科学家在里面做事情，才会产生一家机构，因此，“达摩院”的出现是阿里巴巴发展的必然。

“如果说阿里云让我们拥有了计算的能力，那么 iDST 则更多的是提供算法的能

力。我们集中建设了这样一批能力，加上本身具有非常丰富的场景跟数据，然后才提出了向更纵深去发展。”刘湘雯解释道。

刘湘雯第一次听到马云谈到关于建立“达摩院”的设想是在今年 3 月，阿里巴巴内部召开了首次技术大会，会上马云分享了他的科技愿景。马云认为，此前 18 年，阿里巴巴的商业场景推动了技术升级，面向未来 20 年，核心技术升级才能推动商业模式创新，必须建立起 NASA 这样的机构。

“必须向技术要红利!”这句话，阿里巴巴首席技术官张建锋在会上重复了多次。而早在 2016 年云栖大会上，马云就提出过“五新”战略，即新零售、新金融、新制造、新技术和新能源。截至 2017 年 3 月，新零售已经在落实，新金融正在布局，“已经到半路了”，接下来“必须组建阿里的新技术”。

在这次技术大会上，马云动员全球两万多名科学家和工程师投身“新技术战略”，并启动“NASA”计划，“面向机器学习、芯片、物联网、操作系统、生物识别这些核心技术，我们将组建崭新的团队，建立新的机制和方法，全力以赴。”马云强调，“以前我们的技术跟着业务走，是‘兵工厂模式’，但手榴弹造得再好，也造不出导弹来。

阿里巴巴必须思考建立导弹的机制，成立新技术研发体系，聚焦核心领域的研究。”

阿里巴巴有着巨大的野心——未来 20 年，阿里巴巴要构建世界第五大经济体，服务全球 20 亿消费者，创造 1 亿就业机会，帮助 1000 万家企业盈利。因此，“NASA”计划的目标也是面向未来 20 年，其产品或服务能够覆盖到 20 亿人。

“NASA”计划

据刘湘雯介绍，“NASA”计划的 2 万多人，不仅是研究人员，也包括工程技术人员。近 3 年来，阿里巴巴人才数量年均增长 40%以上，目前拥有 2.5 万名工程师和科学家，500 多位博士。在 36 位合作人中，有 9 位拥有工程师背景。

同时，阿里巴巴也面向全球网罗顶尖科技人才。今年 3 月，人工智能专家、前南洋理工大学教授王刚加入阿里人工智能实验室。6 月 26 日，亚马逊最顶级的华人科学家任小枫加盟了阿里，出任 iDST 副院长。

9 月 11 日，量子技术领域的重量级人物施尧耘加入阿里巴巴，担任阿里云量子技术首席科学家，负责组建并领导阿里云量子计算实验室，同时，施尧耘也在之江实验室担任副主任，该实验室是由浙江省政府、浙江大学、阿里巴巴集团出资成立的混合所有制新型研发机构，9 月 6 日正式挂牌成立。

按照“NASA”计划，如果 2 万多人才全涌到阿里巴巴所在地杭州，似乎也并不现实，在人才聚集地建立海外实验室成为实现“NASA”计划的更好方式。就在云栖大会当天，阿里巴巴首席技术官张建锋透露，“达摩院”已经开始在全球各地组建前沿科技研究中心，包括亚洲达摩院、美洲达摩院、欧洲达摩院，并在北京、杭州、新加坡、以色列、圣马特奥、贝尔维尤、莫斯科等地设立不同研究方向的实验室，初期计划引入 100 名顶尖科学家和研究人员。

张建锋表示，选择在何地建实验室主要有两个原则，一是根据当地的人才状况，比如以色列的安全做得很好，美国的一些大数据算法人才比较好等；二是跟业务有一些关系，比如新加坡本身是有产业基础的，更有利于科技成果在当地转化。

据刘湘雯介绍，现在杭州、北京两地的实验室已经在建设中，美洲、欧洲等实验室的人员陆续到位，已经开始做一些事情。新加坡在加快团队建设的速度，可能很短

的时间就能到位。俄罗斯和以色列的实验室还处于筹备阶段。

“NASA”计划已见雏形，但究竟研究哪些领域并没有确定。10 月 10 日，就在云栖大会前一天，13 位顶级科学家造访阿里巴巴，并与马云举办了一场座谈。

在这 13 位科学家中，包括中国唯一的图灵奖获得者姚期智院士、中国量子力学第一人潘建伟院士、定义了“计算思维”的哥伦比亚大学教授周以真、全球人脸识别技术“开拓者”和“探路者”汤晓鸥教授等。这些科学家研究领域不同，但都参与了“达摩院”的出谋划策。

数年前，潘建伟曾向阿里巴巴提出，成立一个中科院跟阿里巴巴联合量子计算的实验室。今年 7 月 30 日，中国科学院－阿里巴巴量子计算实验室正式成立，实验室将结合阿里云在经典计算算法、架构和云计算方面的技术优势，以及中科院在量子计算和模拟、量子人工智能等方面的优势，颠覆摩尔定律，探索超越经典计算机的下一代超快计算技术。

“当时我说可能 15 年之内都不会有产出，也不会有回报。没想到阿里巴巴很快参与进来合作。”潘建伟感慨地说。

这也是“达摩院”三大组成部分之一，即自主研究中心、与高校和研究机构建立的联合实验室（研究中心）和全球开放研究项目。

与具有科研优势、地缘优势的著名高校联合建立科研基地是阿里学术合作的主要方式之一。继去年 10 月成立清华大学－蚂蚁金服金融科技联合实验室，今年 1 月成立 UC Berkeley RISE 实验室之后，“NASA”计划启动以来，5 月成立了阿里巴巴－浙江大学前沿技术联合研究中心，阿里巴巴不断探索新的合作模式，汇集诸多技术领域内全球最优秀的学术人才，共同打造高效率的科技创新链条和一体化的创新体系。

学术合作的另一种方式是发布全球开放研究项目，将阿里巴巴遇到的工程和技术挑战和各个实验室里最强的学术大脑进行碰撞，进而实现工业界与学术界科技能力的融合。在此次云栖大会上，公布了“阿里巴巴创新研究计划（AIR）”2017 全球课题评选结果，在 13 个国家和地区的 99 个高校与科研机构（国内 54 个，海外 45 个）提交申请的 234 份科研项目提案中，有 40 余个优秀项目最终入选。

AIR 是阿里巴巴集团探索科技创新设立的首个全球性科研项目，聚焦技术驱动未来，致力于推进计算机科学领域基础性、前瞻性、突破性的研究，以校企深度合作的方式引领重大科技创新的实践应用，构建技术生态。以此搭建学术界、工业界的合作平台，联合双方优势共同促进前沿技术的发展。

“一家公司要做长远的科研非常不容易。世界上很少有公司能够做到。阿里巴巴能够有此决心，不只是做跟阿里巴巴商业相关的东西，非常高瞻远瞩。”姚期智对达摩院的雄心表示赞赏。

来势汹汹

“达摩院”在成立之初，便显出凶猛的势头。在马云的演讲中，已经提出“必须要超越英特尔，必须超越微软，必须超越 IBM”，而首批公布的学术委员会更是“星光熠熠”，十人中有三位中国两院院士、五位美国科学院院士，包括世界人工智能泰斗 Michael I. Jordan、分布式计算大家李凯、人类基因组计划负责人 George M. Church 等。

这样的登场，使得达摩院从一开始便赚足了眼球。

“科学研究是有其自身规律的，需要大量的资源、资金和人才，而研究的周期和结果更是无法预测，‘达摩院’才刚起步，现在谈发展如何还为时尚早。”一位某知名企业研究机构的工作人员表示。

“达摩院”首批公布的 13 个研究领域，包括量子计算、机器学习、基础算法、网络安全、视觉计算、自然语言处理、下一代人机交互、芯片技术、传感器技术、嵌入式系统等，涵盖机器智能、智联网、金融科技等多个产业领域。

“这是一个综合决策的过程！”刘湘雯表示，从今 3 月开始，就在确定研究领域，不光有科研人员，还有从事产业研究的人员，以及公司管理层。

刘湘雯表示，这些领域基本上会基于整个科技发展的规律，一方面是阿里巴巴自身的业务诉求，另外一方面，虽然没有看清楚它对业务有怎样的影响，但从大趋势来看，有可能是颠覆性的，比如量子计算机。

然而，一些基础性或颠覆性的学科可能投入大，而回报慢，作为一家企业所属的

研究院，必须考虑不同类别学科的合理组合。因此要放回产业成熟度的链条上，有一些可能三五年能见到成果，有一些可能需要十年以上。“但究竟是怎样一个比例，目前没法给出一个确切的数字。”刘湘雯说。

在中国产学研合作促进会秘书长王建华看来，“达摩院”的建立值得鼓励，要创新一种模式，总需要有人先去探索、实践，“达摩院”正是这样一种探索。

另一个冲击眼球的是阿里巴巴的人才战略，10 月 16 日，“达摩院”宣布，微软亚洲研究院首席研究员聂再清博士、谷歌 Tango 和 DayDream 项目技术主管李名杨博士，入职阿里人工智能实验室。

不难发现，“NASA”计划实施以来，加入阿里和“达摩院”的顶尖人才，除了来自高校和科研院所，有相当一部分来自于知名企业研究院，加上马云对几家研究机构的公开“宣战”，很难不让人联想到“挖墙脚”一词。

“达摩院”的成立起到了一种“鲶鱼效应”，相当于整个科研生态里出了一个新物种，一定会打破暂时的平衡，也一定会有人才的流动。但即使达摩院从某处吸引了一个人才，造成该处暂时的空缺，肯定会从另一处补进一个人才，使整个系统领域在某个阶段会快速地进行重新定位，重新达成一种平衡，并且这种平衡往往会比原来更健康。

按照马云对“达摩院”的定位，即“Research for solving the problem with profit and fun（为解决问题研究并带来利润和快乐）”，刘湘雯认为，“达摩院”招揽的人才除了在专业上彼此认可，在时间地点上恰好也适合这些客观条件之外，从软性条件上来说，双方对于愿景的驱动这件事情要有高度的契合。

经营之道

近年来，阿里巴巴不断加大技术上的投入。财报显示，阿里巴巴 2017 财年技术投入为 170 亿元，居中国互联网公司之首。

此次“达摩院”的成立，阿里巴巴宣布将在 3 年内投入 1000 亿元，相当于每年 330 多亿元，几乎是 2017 财年技术投入的一倍。但在马云眼里，这笔钱只是给“达摩院”的创业基金，实验室绝不能等资金，要有挣钱意识，才能活下去。“我希望不

仅仅靠论文活下来，90%以上研究的东西，不能只在实验室里面，必须在市场上。只有这样，这个实验室才能走得长。”马云说。

如何挣钱？作为“达摩院”首任院长，张建锋对此回应道，达摩院作为一个依托大数据而建立的新型的研究机构，需要一个产业的支撑。而阿里巴巴拥有诸多业务，有金融科技，有电子商务，有物流等等，都是对研究非常重要的支持。同时，依托于阿里云平台，达摩院可以连接更多的应用场景给客户，使他们通过研究院和阿里云这个平台，能够去做智慧城市，做工业大脑，做医疗大脑，连接更多的行业。“我认为这是现在达摩院最大的价值。”

刘湘雯认为，阿里巴巴企业的本质决定了“达摩院”在做科技成果转化上是有天然优势的。

在她看来，一方面，阿里巴巴有丰富的业务场景，都会来到“达摩院”里找他们能用的东西，而“达摩院”通过设立技术开放日，也可以向业务团队去介绍他们的东西。

另一方面，更大的优势就是阿里云。“阿里云从‘达摩院’这头看，是一个放大器，从客户那头看是一个漏斗。”刘湘雯说。

在旷视研究院院长孙剑看来，企业研究院分为两种，一种是研究的内容和企业没有太大关系，主要只是起到“保险”的作用，确保公司将来在大的方向上不要走错；另一种是研究和企业，和当前产品能够有效结合，为公司赚钱。

“其实如果看过去的这些研究院，当一个公司快速发展的时候，或者公司的赚钱是非常没有问题的时候，这个研究院是会蓬勃发展的。但是公司的经济有问题的话，研究院是第一个会被考虑裁减的，因此在企业快速发展，在形势大好的时候，最需要拿出资源投入对未来的投资。”孙剑表示。

从这一角度，阿里巴巴三年 1000 亿元的投入，显然能给“达摩院”一个相对宽松的科研环境，但如何运用这笔资金，也是大家关注的焦点。

刘湘雯表示，达摩院实行的是院长负责制，资金大部分可能会花在人才上，因为对于一个研究院来说，最主要的就是人才，此外，还将用于购买必要的科研设施。

接下来，“达摩院”既会建自己的研究院，也会建联合的实验室，还会向学术界开放，资金也会朝这三个方向分配。

阿里巴巴 CTO 张建锋：下一波创新机会，重点关注这三个领域

以科技创新世界

2017 杭州 · 云栖大会首日，阿里巴巴集团 CTO 张建锋宣布成立达摩院，将在全球各地建立实验室，并引入更多高校教授参与其中，未来三年投入 1000 亿元进行基础科学研发。

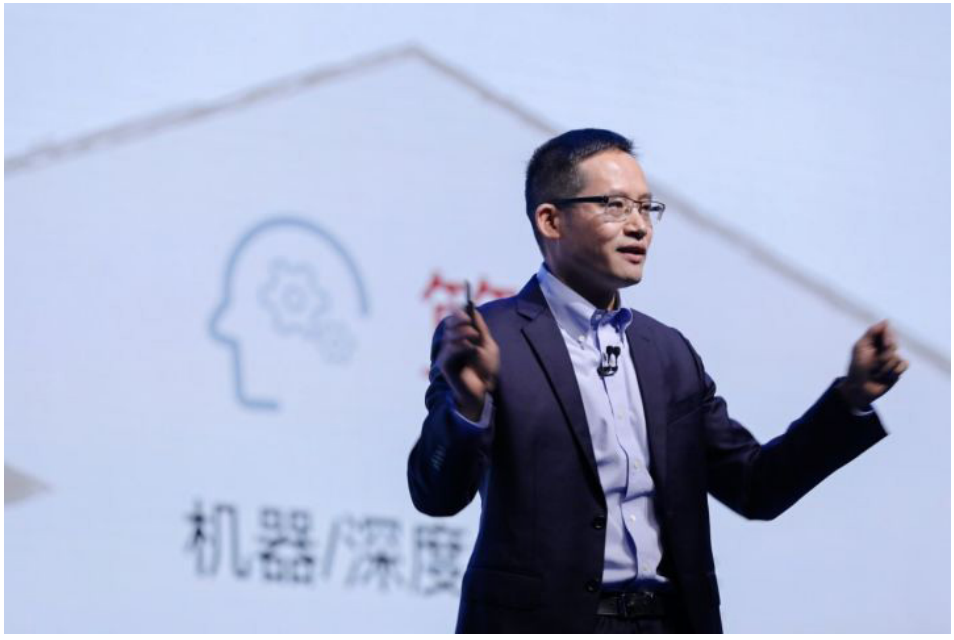
以下为演讲全文：

大家上午好，我想借这个机会，把阿里巴巴技术的一些想法和大家做一个分享。一年前马老师提出了“五新”战略，其中像“新零售”，我们看到盒马、无人咖啡店，得到了非常快速的发展。今天想首先跟大家分享一下“新技术”，以及“新技术”是怎么跟阿里巴巴的未来结合起来的。

互联网的第三次波浪潮：智联网、人机自然交互、机器智能

阿里巴巴是一家成长于互联网时代、扎根于互联网时代的一家公司，我们对技术有着非常深刻的体会跟理解。

互联网时代第一波浪潮，是把计算机从一个单一的工具变成了一个平台，它把信息都连接在一起了，比较有代表性的一些企业，比如作为搜索的像 Google，它把全世界散落在单点的数据、信息都连成一个平台，这是它带来最大的价值。



第二波我认为是移动互联网的发展。移动互联网把信息分享、传递变得更加自然，这一波里面，我觉得最大的贡献就是今天我们在讲的，像社交、应用等等。阿里巴巴在这一波浪潮中做了一件跟大家想象中不一样的事情——我们做电子商务不仅提供一个货架，而是通过互联网这个平台，把消费者跟生产者连接在一起，把品牌跟消费者连接在一起。这个连接其实跟其他人说的做零售、做电子商务是有一个非常本质的区别的——这个连接是双向的，通过这个连接可以诞生出、创造出无数新的可能，而不仅仅是说我通过电子商务、从事互联网来提升零售效率。所以到今天为止，我们走出了一条非常独特的道路。

今天，PC 的销量已经连续在下降，萎缩得比较厉害了。现在去电子市场，纯粹卖一台 PC，基本上没有什么生存空间了；以手机为代表的无线互联网，手机从 2016 年开始，基本上稳定在四亿的出货量，在中国，也没有新的增量。手机操作界面已经决定了，这个界面只有一个屏幕，这个屏幕里面可以放的东西是有限的，现在的超级 APP 基本上占据了手机最主要的入口来源。

下一波机会来自于什么地方，我们思考后觉得有三个领域值得关注：

第一个，是智联网。因为现在还有这么多的设备、这么多的物体没有被连接。以 IoT 为代表的智联网应该是接下来最需要解决的一个问题。这上面我们也做了非常多的尝试，我们做的城市大脑，希望把城市里面所有的物体连接起来，小到井盖、电线杆，再到马路、到红绿灯，都能够通过物联网连接起来，但我们认为光连接是不够的，因为连接只是把所有的人、物聚在一起，我们还需要去感知，还需要去处理数据，最终我们还要实时做出决策，去控制被连接的主体，这才是有价值的智联网。

第二个，新一代人机自然交互。今天我们有了很多交互手段，包括现在非常热门的自动驾驶。自动驾驶目前要解决的主要是一个人机交互的问题。开车一定要拿一

个方向盘吗，可能没有这个必要；控制空调就一定要拿摇控器吗，可能也没有这个必要。因为我们可以有更自然的方式，可能是语音，可能是其他的。以苹果手机为代表，它从原先的键盘式操作，升级到屏幕触摸式的操作，但它只是在一个范围之内的升级。我们希望能够把整个人机交互，从家里的一切应用到驾驶，都有全面的升级。

还有一个，就是机器智能。马老师非常强调我们做的是机器智能。为什么要说我们做的是机器智能，机器智能跟人工智能到底有什么区别?

我的理解，今天我们很多东西之所以这样做，是因为以前人类就是这么做的——以前的做法都是要人来控制，所以我今天不想让人来控制了，我要机器来控制，所以要模仿人类来控制。举个例子，现在人工智能里面最热门的是做图像识别，我们在交通上也好，在城市管理上也好，装了无数的摄像头，因为我们拍了这么多的照片，现在我们人看不过来了，所以需要机器来看，所以机器又要模仿人的所有思考方法，重新认识这个图片。但是我们有没有想过，假如这个照片就是用机器来看的，那为什么一定要拍成现在这个照片的样子，它直接可以是机器认识的就可以了，机器可能不一定要 4K、8K、高清、彩色，可能是从另外一个角度去理解这个世界。 王坚博士举过一个例子，人的东西一定是最好的吗，狗的嗅觉比人更好，你用机器来模仿，会做得更好吗?

所以，我们要做的是，把机器变得有智能，而且变成独立的智能，这个智能应该是机器的能力决定的，而不是人类的能力决定的。这也是为什么，我们今天一定要用机器智能这个概念，重新定义我们真正要的智能是怎么样的一个智能。

平台化、实时化的数据是未来世界的血液

我们今天要做这么多事情，要解决这么多连接的问题，不可避免的会产生大量数据。这个世界一定会被数字化的，我们对此深信不疑，因为只有数字化之后，才有自动化的可能，才有智能化的可能。九年前，阿里巴巴第一次提出阿里巴巴是一家大数据公司，数据是能源，但我今天想说的是，数据不仅是能源，如果机器智能、智联网，包括人机自然交互组成一个人体的话，数据就是血液，没有这个血液，所有上面的一切都没有创新的能量来源。数据，我们认为它远远不止于这个资源，它是组成所

有未来一切的血液。这是我们怎么来看待未来这个世界一个非常重要的出发点。

今后的数据有两个特点非常重要：

一个是实时性。数据一定要非常实时。以前一个产品要推广，做广告。三个月之后，厂家才知道这个广告做得好不好，这个效果好不好，消费者买不买单，这个时候才能去组织生产、组织安排。现在我们这些数字化的广告，每一分钟都知道我这个效果怎么样。

第二个，数据一定要平台化，一定要融合贯通。阿里巴巴有三件事情是统一的，其中最重要的一件事情就是数据的统一，我们统一定义、清洗、处理。我举个例子，我们跟小黄车它们合作，把小黄车给联网了，我们知道每一个车的运行轨迹，我们也知道它的密度。知道这个小区到哪个小区，或者哪个小区到哪个地方，骑共享单车的人是不是特别多。这个数据拿到之后，一方面可以改进小黄车的运营效率，这个数据如果被公交公司知道了，公交公司可以优化它的公交线路，现在没有这些数据，公交公司说今天班车在开，我一直往前开好了。所以数据一定要平台化，它只有融会贯通之后，才能产生新的生产力，才能有新的创造力。

互联网公司跟传统公司有什么不一样，以前我们都讲互联网思维，互联网思维是一个什么样的思维？对于阿里巴巴来讲，我们觉得互联网思维，第一就是一个数据思维——你必须要有数据，你才能做出一些合理的决策。传统公司的 CEO 跟互联网公司的 CEO 有很大的不一样，传统公司的 CEO，他做一个决定，他想知道这个决定正确还是错误，可能要验证很久。在互联网公司，逍遥子可能跟我们讨论，这个页面按钮应该是红色还是蓝色，为什么做这个决定，他有这个数据，他知道改了之后，这个数据有变化了，他敢于做这种决定。我觉得这就是互联网公司跟传统公司非常大的不一样。

我们有一个不成文的规定：我们开会，我跟他们讲，第一，你有数据说数据；没有数据，那就说案例；没有案例，就说观点。都没有，那就不要说了，说了也没用。数据是第一位的，有数据，你就跟 CEO 一样有这个 Power，这是互联网思维里面非常重要的一个维度。

汇聚全球智慧，以科技创新世界的阿里巴巴达摩院

今天我们要做这么多的东西，智联网、人机自然交互、机器智能等等等等，我们后面还有非常多的问题要解决。这些问题包括我们的计算能力、计算平台、算法，自然语言的处理、理解，安全，还有更底层的芯片，更底层的操作系统。因为今天对于阿里巴巴这家公司来说，你已经不可能从市面上买到商用的一些产品来支撑我们未来发展需要的技术。所以我们必须要自己去做更深层次、更高维度的研发。

科学是什么，科学是用来发现规律、掌握规律的；技术是什么，技术是来利用这个规律的；而工程是来实现这个规律的。阿里巴巴这么多年来，通过双 11 积累了非常强的工程技术能力。我们今天把双 11 这一天的技术保障称为“互联网的超级工程”。很多超级工程，比如造世界第一的高楼大桥。而阿里巴巴的双 11 技术支撑这套体系，要支撑那么大规模的业务，解决无数的技术问题，它就是一个“超级工程”。但今天我们想更进一步，我们觉得光解决工程技术问题不够，我们还想掌握规律、发现规律，这是我们真正能够引领未来、真正能够定义未来的核心要素。

今天，在这里，我们正式宣布成立阿里巴巴的全球研究院。因为我们需要有更多的人才，一起参与，一起来改变这个世界。我们这个研究院有一个独一无二的名字叫

做阿里巴巴达摩院。

我们计划在三年之内，对新技术投资超过 1000 亿人民币，我们想要在技术上面，真正做一些原创性、根本性的探索。这么多钱干什么，我们想吸引全球一流的人才，我们也始终认为人才是真正的生产力。在阿里巴巴达摩院，不是叫你来做苦行僧的，是叫你来做骑士的，你们是新一代的骑士，你们不是壮士，科学工作者必须得到应得的尊重与荣誉，这就是阿里巴巴达摩院。

阿里巴巴有这么多的技术、这么多的平台，我们还有一个非常重要的思想，我们不仅去探索未来，不仅服务好我们自己的业务，我们还想通过阿里云这个平台去赋能所有创业者。因为我们是这么想的，所以我们八年前就这么做了——我们做了云计算。我们有这个 Believe，我们相信这个事情一定会发生，我们才做这个事情，我们并不是像其它云计算公司一样，因为我要转型升级了，是因为这个东西非常流行。我做云计算，我们真的是因为坚信。

整个达摩院由三个部分组成：

第一部分，我们在全球各地建自己的实验室，这是阿里巴巴集团自己投资的。我们在以色列、新加坡、莫斯科、西雅图跟圣马特奥都建立了自己的研究机构。在数据智能、智联网、大数据处理等等方面，做一些前沿性的基础性研究，并且能够快速把这些研究成果变成我们业务上可以用的一些东西，也可以通过阿里云这个平台，变成所有人可以使用的一个技术基础设施。

第二部分，我们是跟高校建立联合研究所。我们跟浙江大学联合成立的前沿技

术研究中心运行得非常好，有很多教授、博士在这个平台上工作。为什么吸引他们在这个平台上工作，因为我们有非常大的计算装置，我们有非常多的业务场景。我们采用非常与众不同的方法，别人可能是这样，我有一个项目建好了，然后交给别人来招投标，交给浙大，你来做。我们不是这样——今天这个时代，发现一个问题，跟解决一个问题的难度是一样的。我们在定义未来的世界，发现问题对我们来说也是很大的挑战。我们请他们进来，我们一起来看到底有什么问题，用你们的眼光来看有什么问题，我们一起来解决。我们今天跟浙大、跟伯克利、跟清华大学等都成立了联合实验室，一起来做这个事情。

第三部分，是我们的产学研平台。这个平台非常有意思，我们把要解决的非常多的问题做成一个列表，发给全球的所有高校、机构。高校、机构的教授、学者，对他们感兴趣的研究方向做一个匹配，然后来写他的 Proposal，我们看这个 Proposal 跟我们是否匹配。我们现在有四十多个项目正在开始启动做，而且这个教授、机构，绝大部分来自于海外，国内很多高校也参加了。

最终我们这个达摩院会是三部分：我们自己会建实验室，跟高校做联合实验室，通过产学研平台这个项目，让更多的教授、机构能够参与进来。最终我们希望以科技来创新这个世界，来改变这个世界，这是我们达摩院的愿景。

王坚博士专访 | 揭开国家 AI 创新平台“城市大脑”的神秘面纱

阿里技术

阿里妹导读：王坚博士，一手打造了阿里云。在过去两年，他几乎将所有的时间和精力都投入到“城市大脑”的打造上。近日，首批国家人工智能开放创新平台名单公布，阿里云 ET 城市大脑成功入选。王坚博士对城市大脑有着清楚的定位：它会是未来城市，乃至于整体人类社会的关键基础建设。

互联网作为基础设施的城市大脑

一向善于以深入浅出比喻说明新生概念的王坚，用“电”来说明城市大脑未来所扮演的角色。

王坚说，城市是人类历史上最伟大的发明，但过去几百年上千年，这项伟大的发

明，随着人口的增加、城市面积的扩充、生活形态的改变，已经越来越复杂，有太多的问题已经不能再用过去的方法解决，既有的基础建设，只会让城市的问题越来越严重，而无法真正解决。

就如同许多人过去期待互联网可以解决许多人类生活的问题一样，对于当前城市遭遇的问题，不论是交通、治安、生活环境等等，也有许多人期待并努力通过互联网、AI、甚至是区块链技术，解决这些目前看似无解的问题。

但在王坚看来，既有单一技术或应用就如同电灯泡的发明一样，只发明了灯泡，但没有建立起电网，也是无用的，而以未来人类世界的发展需求来看，城市大脑扮演的就是当初电网的角色，即在全新世界中扮演关键基础建设的角色。

相较于过去，王坚认为现在会是发展城市大脑最好的时点，因为关键的要素都已成熟到位，其中三大关键要素包括互联网、数据、云计算。通过这三样准备就绪的要素，城市大脑将得以成熟运作，进而创造出全新的智能。而这三项技术，也同样是当前包括人工智能、机器智能得以落地应用的基础要素。

王坚甚至认为，以现在的眼光，特别是资本市场的眼光来看，其实是很难想像城

市大脑作为一个基础建设可以激发出全新发明项目，就像当年有了电网之后，人类只看到电灯炮点亮了黑夜，让人类的活动不再被局限于日出日落，而可以有更自由的时间用以生活工作，由此所产生的价值就很难计算，而若以实体发明的角度来看，在电网刚出现时，也很难想像往后会有电冰箱、空调等等应用电力的发明出现，但以今天的眼光来看，电冰箱、空调等技术的发明，对于整体人类生活产生的巨大改变，却也是当初没有人能够预想的。

对于 AI、区块链等技术，王坚则称，这都会是城市大脑这项基础建设运作的一环。许多人谈 AI，但熟悉王坚的人都知道，他并不喜欢用“人工智能”这个词，因为这并不贴近真正意义，与其说“人工智能”，更应该说是“机器智能”（Machine Intelligence)——因为，现在人类之所以有机会解决过去不能解决的问题，是因为机器运作的能力已经达到一定的水准，再加上互联网与数据资源到位，让现在的机器可以为人类解决更多人类不能做到的问题，或者是提升运作的效率。

王坚说：“城市大脑就是要告诉我们，人不能解决的问题，机器可以帮我们解决。”

以阿里巴巴城市大脑第一个落地的城市杭州来看，在导入城市大脑之后，机器智能增加了整个杭州的警力，整体交通状况也明显改善，这其中重点不在于让机器智能取代人力，而是让机器可以做需要投入更多人才能做的事，但原本的人力去做的话只能做人才能做的事，所以，重点不在于取代，也不是让机器做人会做的事，而是让机器做人做不到、或者不需要做的事。

“很多人现在谈人工智能，讲的是所谓‘能听会说’，但这是有问题的。人能做的事情，如果让一台机器去做都是一个玩笑，真正的应该是让机器做人类做不到的事情，人类真正伟大的发明，都是发明人类做不到的事情，为什么阿里把城市大脑说是登月计划，因为那时没有任何人知道我们要去月球上做什么，有什么技术可以做什么。更有意思的是，到了月球之后，人类并没有就留在月球，而是再回到地球，但这个过程至少留下了科技，后来一直影响人类文明的发展。”

他说：“城市大脑这件事用登月计划来形容，除了交通只是解决大家堵的问题，也因为城市大脑可以有更多发明，150 年前人类发明地铁，在未来大家要重新思考，总有一天所有城市不是用修地铁解决交通问题，而是修一个城市大脑来解决交通问题。”

图丨 ET 城市大脑项目从 2016 年自杭州市萧山区开始启动，并逐步扩大推展至衢州、乌镇和苏州等地，也已与澳门特区政府签订战略合作协议

至于区块链技术，王坚认为，城市大脑运作的关键在于有许多的数据，这些数据是由城市中不同功能环境的设施（Facility）运作沉淀而来，例如在车站设施就会有进出吞吐的人数数量，出站之后的方向路径等等，而这些都是支持城市大脑运作的重要资源，但不可否认的是，要确保数据的使用效果，首要确保数据不能被篡改，否则一旦让数据进行城市大脑开始运作，所产生的结果就会非常复杂混乱，所以，区块链就会发挥重要的功能。

另外，王坚认为，在城市大脑运作的过程中，数据的使用是重点，但就数据资源的本质来看，数据要能创造价值就要进行不同形式的交换交流，而区块链的导入就能够完备数据资源进行交换创造价值的空间。

而王坚也强调，区块链对于未来整体人类世界影响，不于单一应用的形式改变，例如现在最常被提及的金融支付等等，而在于区块链技术的出现，会改变过去许多的价值体系设定，例如，区块链确实可能对支付宝产生影响，但这影响不是出现在支付宝要不要使用区块链的技术，而在于支付宝原本是基于现在整体人类社会体系对于“钱”这个价值体系的设定，但区块链技术在未来却是有可能重新改写“钱”这个既有价值体系，甚至于创造一个全新的价值体系，这样影响就会非常明显。

对于城市大脑，王坚不只是在过去几年积极走访全世界多个城市，亲手规画城市大脑平台功能应用，他更深信，这会是未来城市，社会的关键基础建设，他也深信这会是未来城市、社会的关键基础建设，因为城市太需要一个大脑，才能够让智能真正展现，也才能够城市释放出更多的资源，进而让人类生活的更好。

王坚强调，经过先前互联网所带给中国的大量创新，走到今天，在中国的创新已经面临一个回避不了的问题，也就是我们面对我们生活环境所发生的问题，却可能没有能力处理，而城市大脑其实就是这个背景下的产物，从雾霾到交通的问题。

“城市大脑最重要的想法，就是在修马路、修地铁以外，还有没有其他方法解决城市的问题？进而让城市出现一个翻天复地的变化？”

对于这个问题，王坚自己给了一个答案，而这个答案，或许正如阿里云入选成为第一批国家新一代人工智能开发创新平台的自我期望。

王坚：“城市大脑最基础的东西，就是我们人类要做的事，中国必须在这里扮演角色，放眼世界，没有人能够解决我们的问题，所以，我们自己解决！”

如何区分传统公司和互联网公司

11 月下旬，王坚在台湾大学演讲，在此次演讲中，王坚进一步提出许多对于过去互联网发展的反思，也提出对于未来发展的看法：

互联网发展到今天出现很多大企业，最兴奋的不是诞生很多大企业，而是给未来的人创造很多机会，这是这本书最关键的事（《在线》（being online）是王坚在 2016 年写的书）。书名为什么取 being online？这不是一个很热门的词汇。当时出这本书有我跟马云的一个小段子，在大陆出版时的书封面设计是一个小格子，马云说：“这是上一世纪的书的封面，谈的是下一个世界的事情。”

大家都在讨论 Facebook 是美国互联网意义上的最后一家大公司，但这句话不

是终结者的意思，而是一个时代的开始，是在线 being online 时代的开始，这是过去很多东西打了很好的基础，这些基础是什么?

《连线》杂志曾写过一篇文章：The Web Is Dead. Long Live the Internet（web 已死，Internet 永生），这期间经历很多，包括移动互联网、IoT，所以有人开始问互联网是不是到了篮球比赛的下半场? 大家感觉有不少挑战，当人工智能出来英国《金融时报》曾经写道：互联网已死，人工智能时代的开始。

我想讲的是，互联网处在整个时代的“开始的开始的开始”，不管是 IoT 或现在讲 AI，其实我们都没有离开过互联网。互联网已经不单属于电机工程的人，也不单属于 Facebook 、阿里巴巴，而是变成了每一个人、每一家企业的，变成了一种基础建设，所有每家公司都有巨大的机会。

Google 实际上是最早相信互联网是信息传播基础设施的公司，阿里巴巴不是一家电商公司，是一家相信互联网是未来商业基础设施的公司，这是真正的互联网带来的（变化），过去大部分的公司都不会以互联网为基础设施。你的公司是不是这样的公司? 可以想一下，假如今天没有互联网，你的收入会不会受影响，如果不会，或是你是最少受影响的，那你的公司就不是以互联网为基础的公司。

为什么是以互联网为基础设施? 你去杭州看看，虽然还有游民要饭，但他们地上也要放一个二维码，这是很典型以互联网为基础设施，还有一次我看到杭州小偷写几

句话，他写 2017 年都没偷到什么现金，如果今年再偷不到现金，他就不干了，小偷也受互联网影响了，今天因为有支付宝，不只是到超市，小摊贩也可以，这是真正基础设施产生影响的地方。但今天在美国，大部分老百姓还是用支票在付水电费，这就是一个巨大的基础设施的差异。

如果互联网在亚洲变成一个真正的基础设施，就会创造出如同制造业在 20 世纪初期为美国创造的巨大机会，当时他们制造业发达，车子改变了美国所有的城市、工业形态，尽管互联网发明起源于美国，但如果亚洲实现了以互联网为基础，那我们在这一个时代就会有领先的机会，就像汽车是欧洲发明的，却在美国造成巨大影响，真正改变了这个国家，而我们正处于这一个时代。

有了基础设施后，很多东西会改变，比如说时空点改变了，整个世界就是你的，这是缩减地域化。互联网时代带来巨大的财富叫做 data，互联网把个人电脑连起来，移动互联网是把手机连起来。Google 是什么公司？是一家搜索公司，但它其实是把数据做为生产资料的公司，Google 把数据变成产品，那就是搜索，它把全世界网页当成资源来用。

在人类历史发展上，只要有基础设施就会有数据。人最重要的基础设施就是路，人走过去会留下脚印，这就是数据，搜集起来一定是信息，自己沉淀下来才叫数据。人走过去有脚印，不是修路的工人会去搜集脚印，而是你的脚印留在路上，就会沉淀在路上，当警察要查案的时候才会去搜集脚印，以此作为破案的数据。而互联网上沉淀数据的规模将超越人类历史上所有的基础设施。

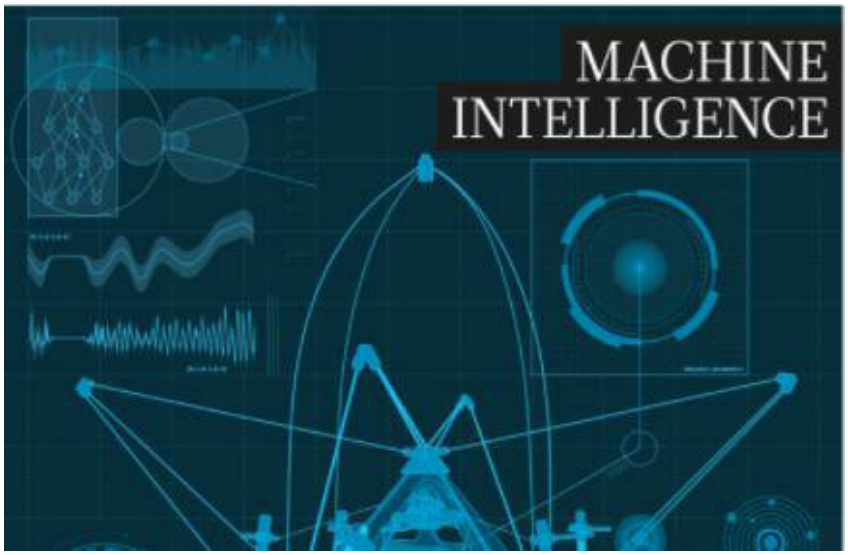
Google 做了一件简单的事，过去网页都在你电脑的硬盘上，Google 就是 being online，如果所有东西只是被数字化而没有上传到互联网，价值就会被大打折扣。但这其中有一个挑战——因为你留下的脚印太多，最后警察要依靠他能找到的脚印破案，那怎么从数据找到你要的东西，这就要靠第三个关键因素：计算。

为什么有云计算？自过去 30 年，你把一个盒子抱回家就拥有计算，现在你个人所需要的计算能力已经远远超出你抱回去的盒子，所以我们需要新的方式，今天你获取计算的方法就是互联网，这就是云计算。从数据里获得有价值的东西就要有计算能力。

互联网公司跟传统软件公司最大的差别就是对数据的看法，所有软件公司都觉得点鼠标是没有价值的，Google 发现鼠标点一下可能没什么价值，但你点进去做什么，它利用计算能力，能把商业价值算出来、猜出来的时候，这就是广告，计算能让数据产生价值，把沙子变成金子。

数据，不叫信息。人类很容易理解信息，但基本上不理解数据。现在大家谈大数据都有一个问题：所有人都努力把数据变成信息。也就是说，千万不要试图把人不能理解的东西硬要变人可理解的，而是把人不能理解的东西，变成可以应用的价值，而这就引出了现在谈的人工智能。

人工智能，我一直觉得最确切的说法应该是人造智能、人造智慧。



先前发明 AI 这个词的人，他们自己也不太相信 AI 能和人类一样，当时计算机的计算能力不好，所以如果一台计算机能够做一点人做的事情，大家就很兴奋。但今天不一样，计算机可做的远远超过人类可以做到的事，所以应该叫做 Machine Intelligence。

大家担心 AI，例如马斯克等人，这其实有点太担心。人最了不起的地方是永远知道自己的智能是不够的，所以我们常常要请教别人，我们第一天就有这种胸怀。

举个例子来说，我们会训练狗，让狗用鼻子去闻一下，是因为我们知道自己的鼻子不够灵，但我们不会因为这样就失去信心。而同样的，机器可以做人做不到的事情，但这些本来就不是人该做的。机器也可以模仿人做一些事情。

今天，你离不开互联网、数据和计算，这三者结合创造了 Machine Intelligence，也就是机器智能的机会。

电被造出来时我们只有一种电器，叫做灯泡，后来就出来了一堆电器，比如洗衣机、冰箱等等，所以，AI 是互联网变成基础设施之后出现的第一个电器、第一个灯泡，有人说 AI 是人类最后的发明，真是太悲观了。现在只是开始而已。

这三个任何一个东西用好了，就是巨大的机会，这是一个最好的时代，结果是什么？这个社会会因为互联网基础设施而让万物在线。

有了电之后，我们有了“晚上”，我们多了一半时间出来，进而有了更多夜晚的经济，现在人工智能出来了，我们的世界也将多出另外一半，人工智能创造了另一半世界，这个世界会比之前大很多，所以，现在是最好的时代，所谓“最坏的时代”这句话并不存在。

Q：大国的网络公司垄断了大国的生态系统，也在全世界范围内实现垄断，这样的差距未来可能缩小？

A：这件事我很乐观。历史上始终有资源不对称、不确定性等问题，互联网发展到今天，提供了让各路英雄涌现的更大的机会，要出现像 Nokia 这样大公司的机会比以前还大得多，因为互联网让大家的距离大大缩短，第二点，同时也更重要的是，把计算变成公众服务很关键，过去只有 IBM 有超级计算机，但现在每个人都可以做，让每个人可以平等，所以基础设施很重要。这不是大小决定而是基础设施好坏决定的。

Q：你刚才说现在的大数据被还原成信息，能进一步解释一下吗？

A：例如美国大选，大家就去做民调，用统计的方式处理一下，这是传统的处理信息的方法。今天不一样，要预测民意，你就把推特的数据也处理，但这里面什么数据都有，你不是要设计谁当总统的表，而是要做一个谁是总统的模型（model）。这个 Model 把所有讯息看一遍，最后得出结论，这就是数据。这么多数据人无法理解，所以我们希望从中提炼出信息，现有模型告诉我们是什么就好，今天没有必要把数据变成信息，只要用数据把模型改一遍，之后改模型就好。

好的模型怎么建立？这我没有答案，至少对年轻的创业者来说是巨大的机会。上个时代有两个东西，软件跟硬件，二者的结合产生了巨大价值，现在来看，数据资源和算法就相当于当年的软体跟硬体，所以现在创业者同样有很巨大的机会。

Q：怎么看苹果？

A：我很难评价，它的每个业务都很复杂。为什么 iPhone 很成功，因为苹果抓住了一个机遇，我觉得苹果是第一家把互联网当作基础设施的公司。iPhone 第一代发布时，乔布斯根本没有提 APP Store，他只提到 Browser，手机可以跟 PC 一样。所以，当时他请了三个人站台，请了 Yahoo 杨致远，他现在不见了，另一个是美国运营商，后来被买走了，最后一家是 Google，现在变成了苹果的对手。当时乔布斯就讲手机对于互联网的意义，这跟现在苹果说的完全不同，苹果的一大挑战就是它的创新少了。

Q：阿里入股高鑫，看中的是线下数据应用？

A：我们需要区分两个事情，一是阿里是把数据用得最好的公司之一，第二件就是，每家公司都会有数据的问题，包括制造业，只是大家对数据的理解不一样。但这不是从阿里开始的，在传统的商业时代，沃尔玛的数据也用得比别人好，否则成本怎么会表现这么好，所以这不是新的问题。阿里是一家坚信互联网是商业基础的公司，目前来说，大家的挑战是没用好数据，而不是有没有数据。

Q：怎么看待 IBM 把量子计算加入到云服务中？阿里有类似的计划吗？

A：量子计算的实现非常重要，我经常说，如果说 cloud computing 存在一个最大的挑战或者说被取代的可能，我认为会是量子计算。量子计算可能把全世界装进去，至少是现在的全世界，它可以满足的计算需求会是现在几百万倍，这是颠覆性的技术，大家并不知道在五到十年后，人们对计算的需求会是现在的多少倍。

Q：目前 AI 的发展是否存在瓶颈或是大家没注意到的问题？

A：人工智能的问题上我要再强调一下，美国把那么多的钱力和最顶尖的人才投入到 APP 开发中，这是付出了代价的，其他领域上的创新会受到影响。如果在 AI 部分，我们把太多资源压在一些 30、40 年前的问题的话，例如语音识别、人脸识别、模仿人类，这会是极大的浪费，就是有问题的。AI 应该是互联网的一部分，以后 AI 的专长应该是人不能做的事情。

Q：云计算是否有成熟的标准？

A：我也不知道，但有几个要素很重要，第一个是，目前全世界计算量还不是靠云计算就能完成的，整个百分比来看还是很小的。当初我说，希望阿里能贡献全世界 70% 的计算，这不是市占 70%，这是一个成熟的标准。第二，从经济的角度，我们常常用电量来评估一个地方经济的好坏，所以我觉得还是和计算量有关系。最后一个重要的标志，如果真正的云计算起来的话，大家就不会再关注云计算本身了，就像我们不会关心电厂有多大一样。现在我们还处在非常初级的阶段，因为大部分的 CPU 还不在 Cloud Computing 上。

Q：阿里云与中科院宣布合作发布量子计算云平台，可以和我们分享下其中与中科院合作的进展吗？

A：阿里跟中科院合作是一个长期的承诺，合作将长达 15 年。量子计算机现在好像应了一句老话：我们低估了它的长期发展，高估了它的短期发展。量子计算跟量子计算机是两件事，短期内量子计算机还是很不成熟的，其中有巨大的挑战，长远来看，我们低估了可能的影响力，因为量子计算的规模已经是超越了人类知识、现有的

理解，所以大家很难想象其可能带来的影响。

Q：可否更进一步谈谈城市大脑?

A：做城市大脑是我们发现城市太复杂了，这个问题没人可以避免，现在的城市完全没有被优化，不像上网买东西，其服务已经得到持续改进。我们通过城市本身的活动来优化城市的运营就要依靠数据资源，例如碳排放量。城市大脑不是一个项目，而像是电网 power grid 一样的基础设施，城市大脑是最复杂的人工智能问题。

本文来源：DeepTech 深科技

华先胜：视觉智能应用成功的关键因素有哪些？

阿里技术

阿里妹导读：人工智能历史上的三次黄金时代是什么？这次有何不同？视觉智能应用成功的关键因素有哪些？本文通过众多的成功实例和遍地黄金的视觉计算应用机会，对这些问题进行探讨，并试图讨论云上视觉智能的终局。

注：本文整理自阿里 iDST 科学家华先胜在全球人工智能技术大会上的演讲。

今天和大家报告的主要是近两年在阿里云上做的视觉智能方面的工作和一些思考。

首先看一下人工智能的三次“春天”。

第一次是在 20 世纪 50 年代，人工智能的概念首次提出，大家觉得人工智能在 20 年之内会改变世界，所有的工作都会被人工智能颠覆。但是后来很遗憾，10 年以后发现不行，大家很失望。

第二次是 80 年代，神经网络的提出，BP 算法的提出，以及专家系统的初步结果，大家又很高兴，人工智能又要改变世界，取代很多人的工作，但是后来证明还是

不行，人工智能又一次进入了低谷。

第三次就是今天，这次是不是真的春天呢？昨天有一个论坛也在探讨这个问题。这次有一些不一样，有很多不同的观点，有人认为深度学习取得了很大的突破，计算能力大大提升，数据更多，网络带宽也大大增加。还有一个很重要的原因，我们已经看到一些结果，虽然这些结果离真正的智能还差很远，但是在一些领域已经取得了非常不错的结果，不管是只有 PR 效应的还是真正在产业界的应用，都有一些可喜的结果。

云上的大数据视觉智能

人工智能技术将会改变哪些行业？我们先从视觉的角度看一看，视觉智能可以从云上做，也可以从端上做，我们今天就从云上来看。我们看看现在发生了什么样的事情，其实有的是发生了很多年的事情。

大家看这些图，左上角是交通的监控场景，右边和左下是治安和教育的场景，最后一个是直播。直播是主动的，前面三个是被动的。这些大量的数据，其价值有没有被充分发掘出来，这是一个很大的问题。

例如，在全世界有数以亿计的摄像头，中国占了一多半，每年有几千万的摄像

头被采购，中国一个一级城市里就有几十万的摄像头。大家可能也注意到一些，这些摄像头的数据到底是怎么被利用的，大家开车可能被处罚过，还有交警的控制中心经常要巡检查看，公安局里出了什么案件也需要调录像查看。仅有这些吗？投入了这么多，这些视频的价值怎么才能充分被挖掘出来，这是一个很大的问题。

再看个人的图像和视频数据，这个量也挺大，和我们每个人切身相关。我们每到一个好的地方、有好的风景，自己看没看没有关系，一定要让相机“看”一下。另外还有各行各业的数据，比如无人机的数据、工业的数据、医疗的数据，以及体育、娱乐、新闻等等。这些大量的数据，在技术往前发展了一大步的今天，它们的价值能不能充分挖掘出来?

我们处理这样的数据，就是一个视觉大数据的问题。它的特点是显而易见，第一就是数据量非常大。视觉数据量最大的地方就在城市里面。有一些电视台有 100 万小时的数据，已经很多了，后来想一想，如果一个城市里有 10 万个摄像头，跑 10 个小时就是 100 万小时。第二是很多应用有实时性的要求。例如，交通红绿灯配时的自适应优化，就需要实时进行分析，实时做出决策。

第三点就是数据的复杂度非常高，各种情况下的数据都有，各种应用的数据都有，数据的干净程度和质量都有很大的不同，需要完成的任务、开发的智能也都是不一样的，这就对算法的普适性提出了很高的要求。

视觉智能的五要素和现状

我们首先回顾一下现在的技术和数据等各方面是不是准备好了。

第一方面，从算法的角度来看，准确率是我们首先关注的目标。我们经常看到这个公司又刷新了一个公测集的记录，包括我们自己最近也刷了一个车辆检测的记录。这是不是说明视觉智能已经很厉害、已经超过人了？在现实的应用当中往往是非常残酷的，公测集上的结果往往只是一个开始，在实际应用中还需要很多非常繁重的工作，才能使得我们的算法在一个行业里做到可用。

其次，从覆盖率上来讲，这个问题就更大了，在座的各位可能很多都是学生，我们在写论文时很少有人关注覆盖率这个问题。覆盖率是什么意思？如果从识别的角度

来讲，就是识别的范围足够大。这个问题很有意思，例如，ImageNet 中 1000 类物体场景的识别，我们拿到真正的应用场景里去看，是远远不够的；或者说，实际应用场景感兴趣的常常不是这些类别，也就是说这些还没有覆盖到用户需要的地方。你要覆盖全世界是非常难的事情，但是不见得是不能做的事情。

几年前我在微软还尝试做过百万标签识别的问题，这个准确率当然很难做得高，但是在一些场景下也是可以用的，例如搜索。覆盖率在视觉搜索中的体现，例如，能搜衣服，不能搜鞋子不行，不能搜其他东西也不行。用户的使用体验往往与覆盖率有非常大的关系。

第二方面，计算效率。效率决定了这个事情可不可能发生，比如我们要处理城市几十万的摄像头，需要花几十亿就完蛋了，这不是成本的问题，是这个事情可不可能发生的问题。从计算的角度来讲，不仅仅是计算的效率，还有计算的平台，尤其是当你处理大量数据时，不是一两台机器，而是百台、千台、万台时，就需要处理系统和流程的问题，比如说容错、流程的控制等，这就需要一个大的计算平台来支撑。

从计算来讲，效率是非常重要的，包括平台的效率、计算节点的效率。例如，一台计算机放多张 GPU 卡，这些卡如何充分利用起来。还有算法本身运行效率的问题。 刚才我忘了说一句，关于算法的一个结论：我们确实有很大的进展，但是还有很长的路要走。对于算法而言，只有把计算的效率发挥到极致，算法的优势才能发挥到极致。

第三方面，数据。这也是争论最多的问题，昨天也有一个论坛讨论数据的问题。大家经常发现数据的威力有时会超过算法，当然如果只是学生作为借口，做不好算法说是数据的问题，那是另外一回事。在昨天的论坛上也一直讨论数据和深度学习算法的问题，实际上数据的使用有两个方面的问题，这个还是一直没有说清楚。

数据的作用到底在哪里？我觉得很多时候大家只是关注了数据对算法研发的作用，但是这只是其中一个作用；而数据对智能本身是另外一种作用，而且是很重要的作用。没有数据，就没有从数据产生的智能。至于没有大量数据是不是就没有深度学习算法，这个还可以商量，也许少量的数据也是可以的，但是作为智能，尤其是强人工智能的话，如果没有大量数据恐怕是不可能的。

所以，数据是有两个维度的作用在里面，数据本身是算法研发的原料，同时数据又是产生智能的原料，这是数据的两个作用。数据本身也有很多的困难，数据量大的时候，包括采集、传输、接入、融合和存储等各方面都不是简单的事情。还有非技术方面的困难，尤其是数据的开放，其实在中国这件事情已经比西方国家好得多了。 在中国，大家对数据开放没有那么纠结，这也是人工智能在中国获得更快发展的一个很重要的原因。

第四个方面，刚才讲了人工智能风声水起，视觉计算遍地开花，但是，花开了，能不能得到结果？就是你做的事情是不是个正确的事情，是不是真的事情。有时候看起来是个真事情，其实是个伪课题、伪需求。昨天也有人提到伪需求，我们在实际当中确实是会碰到的。客户有时提出的需求，仔细想一想可能就是伪需求，也就是说不是一个能够带来真正价值的需求。

无论你带来的价值是节省了人力、降低了成本，还是提高了安全性等等，这些都是要非常明确的。如果这些不明确，你就没有一个商业的模型和应用，没有明确的商业应用，没有持久的商业应用，这个 AI 也就不能持久。

总结一下，一共五点（有一点没有直接讲）：算法是安身立命之本；计算平台保证算法能大规模处理大量数据，也是计算效率的问题；数据，一方面是算法研发的原料，也是产生智能的原料；用户这个要素刚才没有单独分析，但它与商业模式和数据是非常相关的。商业上，有大量的用户使用，或者说用户少，使用的频率比较高也是 OK 的，而用户本身也能产生数据。例如，搜索引擎就是利用了大量用户的数据，每个人对搜索引擎都是有贡献的。商业刚才讲了，合适的商业模式，保证你做的是正确的事情，不是虚假需求。

视觉智能实例：拍立淘

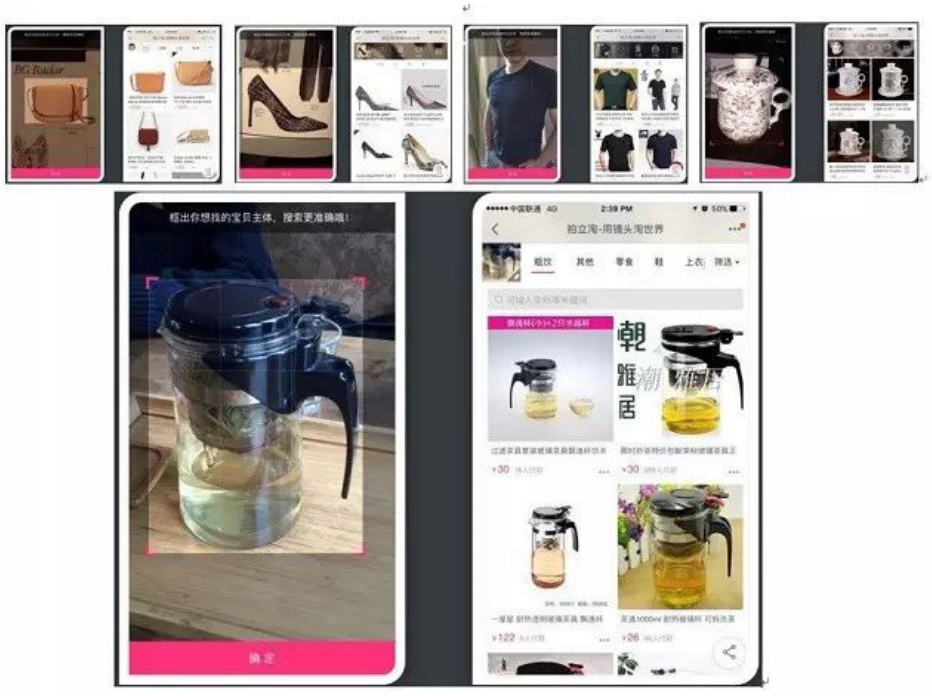
下面讲几个例子，有的是已经做好的，有的是正在做的。

首先看基于图像的商品搜索。我们今天讲的是视觉的搜索，是通过拍照的方式搜索商品。淘宝上有一个功能就是拍照搜索，叫做“拍立淘”。它要解决的问题就是文字之外的搜索入口，是无法用简单文字描述的搜索需求，是种简单直接的搜索方式。

如果这个应用每天的用户和交易量在千万级别的话，还是很有价值的。

这里关键的技术包括商品识别、商品检测、和商品描述。首先，用户拍了商品照片后，要做出精准的商品类型判断，不然后面就全错了；然后要知道这个商品在图像中的位置，再用一个深度学习网络做特征提取；后面还有检索、排序、搜索质量判断，以及结果呈现。这里的几乎每一步都是用深度学习来完成的。

我们来看几个例子。这是同一个包，但其实图像是不一样；这是一只鞋子，虽然我们没有找到同款，但找到了非常相像的款式；这是一件圆领衫，没有什么显著的特征，比较难做，但也是找到了很像的衣服；这个杯子是一次开会的时候看到的，你要用文字搜就说不清楚了，但用图像找到同款却易如反掌。



还有个例子，是和朋友喝茶的时候，看到这个泡茶杯太好了，我之前没有见过；杯子上面有一个红色的按钮，就是水倒下去后，水是在上面泡着茶叶，觉得泡的浓度差不多了，就可以按这个红色的按钮，茶水就流下去了。我想买，但不知道这个杯子

叫什么。好在我们有拍立淘，一拍就知道，这种杯子叫做飘逸杯，淘宝上有很多可以选择。

视觉智能实例：城市之眼

视觉之眼，是城市的眼睛。我们要处理的是城市的摄像头，不管是交通、安防、城管，还是个人的，这些摄像头的数据，我们思考怎样把它的价值挖掘出来。里面涉及到的技术仍然是视觉数据的检测、识别、系统、搜索、挖掘等。

这个例子是交通视频的分析，对车辆的检测、车辆的跟踪、车辆的属性，就是将路面上发生的事情了解个底朝天。过去做交通优化的时候有两个信息源，第一个是地感线圈；但线圈数据不知道这个车的属性、车类型、车多长，这个车到哪里去了，这个信息不全。第二个数据，是 GPS 的数据；但一般只有少数人开启 GPS，所以是采样数据。视频数据不同，是“眼见为实”，摄像头见到的才是真实完整的数据，所以这个数据是不可替代的。

这个例子是另外一种摄像头，高点的摄像头，虽然细节看不清楚，但是数数可以数得出来，而且，你任意画一个区域就知道关于这个区域物体的移动情况。比如说经过多少辆车、大概的类型是什么；有的地方不让停车，你可以画个区域不让停，一旦有车停了就报警。

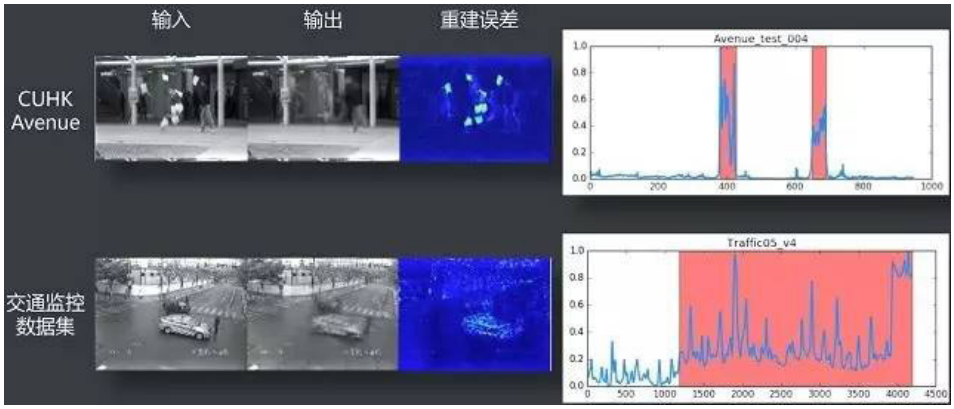
这些技术也没有什么特别的地方，也有很多人做类似的工作。但是有一件事情不同，就是如果处理大量这样的数据，几万、几十万这样的数据，你需要在一个平台上进行实时处理，这就不是一个简单的事情，而且处理的效率要足够高，这是很关键的事情。我们有离线和实时两套处理系统，大规模离线视觉分析，这个是阿里的一套系统，对实时性要求不高的大量视频数据，离线比较容易处理。实时的原理也差不多，只不过有延时方面的要求。

系统实现上，还有时间上的和空间上的实时协同。比如说，对一个路口的交通灯进行管控，你要看这四个路口，还要看旁边几个路口，你在实时分析的时候还需要把空间多路信息进行融合。时间和空间的协同问题，是由平台来支撑，而不是算法，这样我们做算法的人员就可以集中在算法的设计和优化上。

还有搜索的功能，刚才讲了电商的搜索，这个量级不小，但是还有一个量更大的就是城市的数据。城市的数据量太大了，里面有车、有人。人是非常难的事情，人脸相对容易，而看不清人脸的人就非常难；车相对容易一点，我们要学习它的结构化特征和它的非结构化特征，也就是用一个向量表示的视觉特征。

这里我稍微岔开来讲两个关于视觉数据的特别的例子，其实也是城市视觉识别技术的例子，但又是在数据的量上和我们直观的感受并不太一致的例子。第一个是车牌。 数据这件事情是非常有意思的，刚才讲了大数据，但是刚才讲的数据一个是研发算法的原料，第二个是人工智能的原料。对于算法研发而言，往往需要大量的标注数据，但有时这样的数据并不容易获取，或者获取的成本比较高。例如车牌的识别，车牌看起来数据量很大，但双层黄车牌的量就要小很多。

有一种思路就是自动生成一些车牌作为车牌识别的训练数据，这两幅图就是例子，是算法生成的以假乱真的车牌。这个车牌产生以后，对识别的准确率有显著性的提升。还有些场景，数据的获取更可怜，比如事故，但是你有大量正常的样本，一样可以用来作数据的模型，把它作为异常检测的问题来做就可以了。这上面是公开测试级上的结果，视频中间有人撒了一点纸，这个异常的检测响应是非常明显的；下面的这个例子是车辆的刮蹭，是个真实场景，难度就大多了。



从搜索的角度来讲，我们把整个城市的数据如果都收集起来，放到一个大数据里，建好索引，大家脑补一下，将会对城市的交通优化等应用产生什么样的影响。如

果我们再进一步挖掘数据的价值，有很多应用场景可以考虑……

视觉智能实例：视觉诊断

第三个是视觉诊断，包括诊断人和诊断机器。诊断人比较好说，就是医疗图像分析，现在也是很热的题目。当然它比其他的方向慢了半拍，一方面由于数据收集的困难；另一方面是需要很强的专业知识。机器诊断是还没有开发的方向，它的问题有点像前面提到的异常检测的问题，有发生概率很低、正例样本很少，以及正例样本差异性大三个特点。

举个例子，1 万个样本，只有 10 个有问题是你要找出来的。但是你找不准那 10 个，只能说找出 100 个，那 10 个就在 100 个里面。这时你的召回率是 100%，而准确率很低，只有 10%。但是，这有没有用？我们算算省了多少人力，省了 99%，因为你只需要看 100 个就行了。哪怕只有 1% 的准确率，只要召回率足够，也省了 90% 的人力。

所以这类问题的目标不一样，衡量的标准也是不一样的，省人力是非常重要的指标。其实这里面涉及到各行各业的视觉问题，凡是过去需要人眼来看的，是不是都可以用视觉的方法来解决。从这个角度来讲，就是遍地黄金，很多地方都可以挖到黄金，不见得出来一个视觉创业公司就一定要去做人脸识别。

视觉智能实例：视觉广告

前面三个是偏分析、搜索的，第四个方向——视觉广告，是合成的方向。视觉广告是将视觉数据变现的最直接方法，特别是对于娱乐的数据、个人的数据、新闻、电视电影等这些数据。这些数据怎么发挥更多价值，除了人看以外，广告是很重要的方法。但是增加广告后用户的观看体验就很差，大家如果看过网上的视频，应该有深切的体会。那广告是不是可以做的更好一点？我们看几个例子。

例如，可不可以把广告放在场景里，作为场景的一部分？当然，这个已经有人工在做这样的事情，但是人工做不了大量的内容。如果可以自动化，就可以用到大量的视频中。像下面这个例子，把视频中电视机的屏幕部分换成广告视频。这样的广告既

不耽误观看者欣赏视频的内容，也不占用观看者的时间，但实际上它已经潜移默化地影响了你。

云上视觉智能生态

阿里云上的视觉技术有一个统一的名字——阿里云眼，是阿里云大数据平台的智能视觉中心，这是它的总图。回到一开始提出的问题，人工智能将会改变什么行业，答案就是智能将进入各行各业，Intelligence Everywhere 势不可挡。

但是，人工智能的从业者也是很容易翻船的，因为你需要这五个要素齐备。还有一种选择，就是你可以加入到一个生态里。终于回到今天讲的主题上来了——打造云上视觉计算的生态。不仅仅是视觉，其他智能也是一样。在云上可以搭一个舞台，这个舞台不仅仅是大公司在玩，小公司也可以玩，个人也可以玩。不管是哪个层次的智能，基础 API、功能模块和解决方案都可以。

这个舞台上还有一些基本的道具可以使用，例如搜索引擎、机器学习平台、大规模视觉计算等，还有最基本的计算和存储，这些东西都可以利用起来，大家都可以在这个平台上玩。其实，整个云上的智能也不是一两个公司可以完成的，各行各业的需求量非常大，需要很多人一起努力，把这个生态一起繁荣起来。

道哥自述：为什么弹性安全网络将诞生最大的人工智能？

道哥的黑板

阿里妹导读：前阵子，阿里科学家王刚、吴翰清同时入选 MIT2017 年度 TR35 开创中国互联网企业先河一文刷爆了朋友圈，阿里巴巴人工智能实验室首席科学家王刚、阿里云首席安全科学家吴翰清同时入选 MIT2017。这是自该奖项创立 18 年以来，第一次中国公司里同时有 2 人入选榜单。今天，阿里妹分享一篇来自吴翰清（也就是大家熟悉的道哥、小黑）的文章，让我们一起走进道哥的弹性安全网络世界。

前些天得知自己入选了 MIT 的 TR35，非常开心。我想这是中国安全技术在国际上被认可的一次证明。但这个荣誉不仅属于我一个人，更属于我团队中所有为此做出过努力和贡献的人，也属于那些敢于和我们一起尝试最新技术的客户们，因为新技术在诞生之初往往是生涩的，但缺少了孵化过程中的磨难，我们永远见不到美丽绽放的那天。我也非常感谢王坚博士、弓峰敏博士、华先胜老师、Dawn Song 教授能够成为我的 TR35 推荐人，感谢你们对我所从事的工作的认可。

自从参加工作以来，我一直执着于将中国技术推向全球，我认为中国有着最好的安全技术和最好的人，只是缺乏了让他们成长的土壤和展示的舞台。所以我也希望这次 MIT 对我个人的认可，能够成为一次鼓励中国安全产业的优秀人才和优秀技术成果走向世界的契机。长期以来，我们享受了很多开源技术的红利，但中国技术对世界互联网发展的贡献却非常微薄。我认为这中间有语言的障碍，有文化的障碍，但没有能力的障碍。现在是时候让我们去跨越这些障碍，去解决全球互联网发展过程中遇到的那些问题了。只有中国本土的优秀人才成长起来，中国才会变得更加强大。

回顾我十多年的工作生涯，期间从事和研究过非常多的技术工作，但我认为唯有「弹性安全网络」的研究是最独特的。「弹性安全网络」不是对现有技术的一种应用，它是真正的发明了一项此前所没有的技术，提出了一种全新的方法，采用了一个全新的角度来看待现有世界。也因此它能跳出现有的技术框架，带来一些突破性的惊喜。这些惊喜，往往连创造者都没有办法在一开始就想清楚。正如从比特币中抽象出了区块链技术一样，最早我们构建的产品「游戏盾」是用来防御超大流量 DDoS 攻击，最后抽象出来的「弹性安全网络」技术，却让我们看到了构建下一代互联网的可能性。

简单来说，弹性安全网络是将 DDoS 防御前置到网络边缘处。但是，未来真正要做的事情是通过端到端的连接，通过风险控制技术，重新构建一个干净的、安全的互联网。

前些天《麻省理工学院技术评论》的记者对我做了一次采访，我完整的阐述了一次关于弹性安全网络的构想。我把这次采访的录音放在这里，分享给所有对这项技术感兴趣的人，并附上整理后的文字稿（但依然强烈推荐听录音原文）。未来我希望有更多人参与到对「弹性安全网络」的建设中来。

为什么要做弹性安全网络

互联网的流量就像流淌在管道里的水，但互联网发展到今天，流量里已经掺杂了太多的东西，变得不再纯粹和健康了。比如说，这些流量里面包含了很多攻击请求，也有很多恶意爬虫请求和一些欺诈行为的请求。

理想状况下，我们希望未来的流量是干净、健康的，希望把所有的网络攻击前置到整个网络的边缘处。就是说进入这张网络的时候，流量本身就是干净的。这就是 clear traffic 的概念。

为了实现这个想法，我们遇到了很多的困难。我们在思考，需要用一个什么样的架构去实现它。刚巧这个时候，我们有一些客户尝试用快速切换的思路来对抗 DDoS 攻击。这给了我灵感。最终，我把两个东西结合起来，产生了做弹性安全网络的想法。

什么是弹性安全网络

弹性安全网络真正想要去做的，是替换掉整个互联网最核心的心脏，替换掉 DNS，从而让网络变得有弹性，能够快速调度资源，形成一个全新的网络架构。

事实上，DNS 诞生在互联网早期，是互联网 1.0 时代的产物，是一个开放的协议。到今天，也没有一个独立的运营商来运营整个互联网的 DNS Server。它分散在各家不同的运营商。全球可能有上百家运营商，都在提供自己的 DNS 服务。运营商跟运营商之间的打通，是通过标准的 DNS 协议进行数据交换。

这也是为什么这么多年 DNS 协议都没办法进步的原因，过于碎片化。

目前，DNS 有三个显著问题。第一个，是 DNS 完全解析的时间过长，这是整个 DNS 使用中遇到的一个非常大的痛点。

比如，对于一个大型网站，要把用户的所有流量指向一个新地址。把 DNS 的解析修改之后，可能需要花两到三天时间，流量才会百分之百的切到新地址去，不会在旧地址上还有残余流量。

为什么需要两到三天时间？原因是有很多运营商的 DNS 递归解析服务器，都需要更新自己的数据。而有的运营商还有自己的省级运营商，甚至更下面的地市级的

DNS 的递归解析。过于碎片化，使得难于进行统一的数据管理，这是今天现实存在的问题。

第二个问题是今天 DNS Server 软件中的解析数遇到了瓶颈，没有办法一个名字解析到几千个、甚至上万个，甚至未来十几万个不同地址。一个名字可能最多也就解析到十几个或几十个地址就不能再扩大了。这种瓶颈限制了我们的一些能力拓展。

第三个就是，原本可以基于 DNS 去实现的一些安全机制，比如风险控制，并没有建立起来。其实也比较好理解，在互联网 1.0 时代并没有如今天这般强大的数据能力和计算能力。

今天，我们要解决这些问题。在整个弹性安全网络的架构下面，我们在构思下一代的互联网应该是什么形态？答案就是通过可靠的快速调度技术把互联网心脏重构掉。

首先，就是它的快速解析的能力，一定要非常实时以及干净。其次，就是它本身支持的调度能力，要能达到上万的这个级别，规模特别的重要，就是一个名字能够解析到上万个地址、甚至是十几万个地址。

我们以防御 DDoS 攻击为切入点，进行尝试。过去防御 DDoS 攻击时，必须要做的是储备单点大带宽。因为 IP 是变不了的（在中国的网络环境下由于政策原因暂不考虑 anycast 的方案）。所以在 DNS 架构下，就是去硬抗这个 IP 遇到的流量攻击。比如说 300G 的流量打过来，必须要有 300G 的带宽在这里，才能够扛得住。如果只有 100G 的带宽，那整个机房就被堵死了，甚至可能会影响到运营商的网络稳定。

这是在过去攻防对抗的思路，就是你攻击打过来多少，我就必须要有多少带宽储备在这儿。这比的是资源，比的是单纯的带宽储备。

我们现在的思路是，你攻击这个 IP，我马上就把这个 IP 拿掉，不要这个 IP 了，然后启用一个新的地址，并告诉所有客户，你来访问新地址。

当然，这时候攻击者会跟随，但是攻击者跟随是有成本的。一般，攻击者跟随到一个新地址，需要大概 10 多分钟。

在这个 10 分钟里，通过数据分析的方式，我们可以分析出攻击者到底是谁，把

好人和坏人分离出来，阻止坏人的流量，并同时放干净的流量继续访问，这就是整个弹性安全网络的核心思想。

如何实现弹性安全网络

弹性安全网络的实现，是通过快速完成上万个地址的调度，从根本上改变过去需要在单点储备大带宽的一种防御方式能力。

就是，你不需要在单点储备大带宽了，你需要的更多的地址，更强的数据分析能力。

要知道，单点储备大带宽的价格非常贵。改用这种方式之后，DDoS 防御成本可以下降两到三个数量级，因为不需要再单点储备大带宽。

做完这个之后，我们就发现，其实这个事情，最重要的不是多了一种对抗 DDoS 攻击的方法，而是改变了 DNS 本身，这是本质的东西。所以，我们是用一种新技术去解决了一个老问题。

弹性安全网络将诞生最大的人工智能

沿着弹性安全网络的思路，我们希望通过风险控制来管理整个互联网的资源。

未来，弹性安全网络将重新定义互联网的入口。通过为每一个访问者建立“足迹库”，分析他是好人还是坏人的概率。一旦判断这次访问请求可能是有风险的，则可以随即让他访问不到这个资源。

所以，未来最大的人工智能应该是诞生在弹性安全网络，因为整个互联网的资源都被管理起来了，而且是基于每一个访问者的行为沉淀，来判断风险。

相当于想要进入这个封闭的网络，每个访客要先过安检。只有通过安检才能访问到这个资源。而且，访客所有的历史行为会被积累下来，为未来的风险判断做储备。而今天互联网的心脏 -- DNS，由于其开放性和碎片性，已经失去了将所有访问数据统一汇聚后进行分析的可能性。

在一个自成闭环的体系里面，由一家基础设施的提供商，去运营整个网络心脏的这种解析服务。然后也基于这种解析服务，它能够对整个网内的所有访客进行智能分

析，最终就能够实现这张网内的所有访客的请求，都是在风险控制之下的，从而构建一个全新的互联网。

弹性安全网络的未来

今天，一些阿里云上的游戏客户，就是通过弹性安全网络的技术，来调度他们所有的游戏资源，同时对所有玩家进行风险控制的。

弹性安全网络自成闭环。也就是说，这些使用弹性安全网络的游戏，已经从我们现在的互联网，也就是今天以 DNS 为支撑的这个互联网里，消失掉了。

一个玩家，通过 DNS，是访问不到弹性安全网络这张网里的所有资源的。未来我们要做的事情就是，不断地去扩大这张网，直到网内可调度的资源覆盖整个互联网的资源。

目前来看，主要机会就是在 IoT 和移动互联网，因为这两者实际上是没有 DNS 的需求的。过去，之所以需要 DNS，是因为有一个浏览器，浏览器里面有一个地址栏，这个东西必须通过输入一个好记的地址，才能访问到资源。

在移动互联网时代，今天手机不需要浏览器，而是直接打开一个 App。那这个 App 访问的是什么东西，它不一定需要 DNS 来解析。

这是我们看到今天这个技术有可能走下去的一个非常重要的原因。

延伸出来，在 IoT 时代，也是不需要有一个浏览器去访问你所需要访问的服务和资源的。

所以这是我看到，这张网在未来有可能升级今天整个互联网最重要的一个原因。

阿里将开放弹性安全网络技术能力

未来，阿里会开放弹性安全网络的技术。

类似 DNS，弹性安全网络本身也不涉及任何访问资源，它只是知道你今天到这个地方来了。就像，一个人今天到某个国家去，需要入关和出关，是一个道理。

事实上，在很多关键领域，弹性安全网络非常有价值。

比如，各个国家政府，或者大型企事业单位的专网或内网。如果它是以 DNS 为

核心的话，那这是一个暴露在整张网内的弱点。因为 DNS 是一个公开的服务。一旦 DNS 这个单点被瘫痪掉，整张网可能就没法工作了，所以这是非常大的风险。

所以，弹性安全网络技术，不是为某一个客户设计的，它是为整个互联网设计的。

阿里开源

重磅！阿里巴巴正式开源全球化 OpenMessaging 和 ApsaraCache 项目

阿里巴巴

10 月 14 日，在 2017 杭州 · 云栖大会之开源技术峰会上，阿里巴巴正式发布了全球化 OpenMessaging 和 ApsaraCache 两个开源项目，并宣布与 GitHub、Hashicorp 两家公司成为技术合作伙伴。此前，阿里巴巴捐赠开源的 RocketMQ 已被 Apache 基金会接纳为全球顶级项目，此番动作体现阿里巴巴在全球开源业界的引领地位。

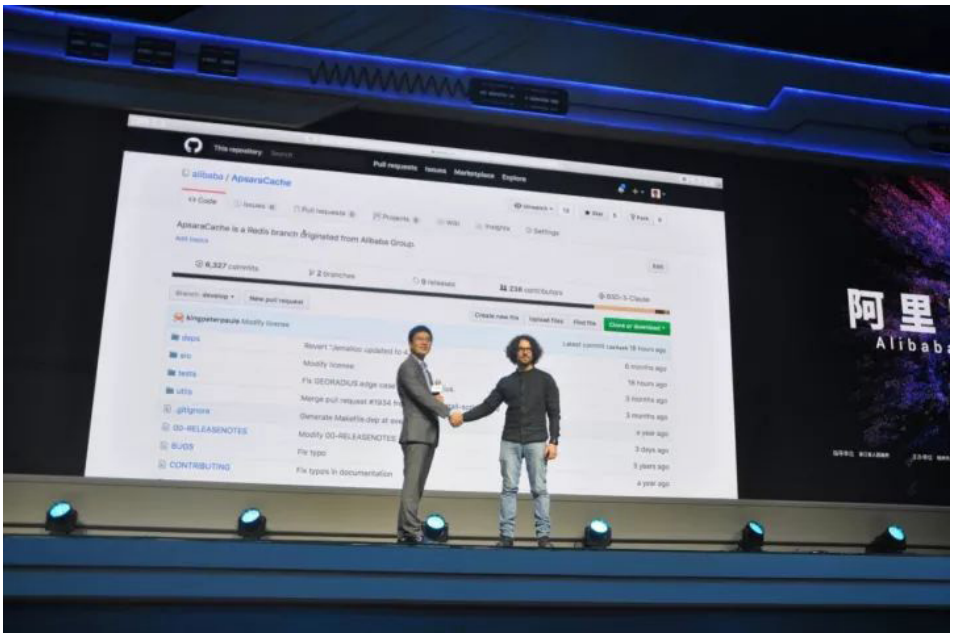
阿里巴巴集团 CTO 张建锋

“开源和阿里巴巴都根植于互联网，有了互联网技术平台之后，开源和商业将在未来相当长的时间内保持平衡的发展。”阿里巴巴集团 CTO 张建锋表示。

据悉，OpenMessaging 项目由阿里巴巴发起，与雅虎、滴滴出行、Streamlio 公司共同参与创立的分布式消息中间件、流处理领域的应用开发标准，目前已正式入驻 Linux 基金会，这也是国内首个在全球范围内发起的分布式消息领域国际标准。

该标准可以不受编程语言限制，能满足企业对扩展性、伸缩性、隔离和安全的要求，可提供大规模的工业级支持，支持标准参照点的添加与标准化测试，开放接口便于对其他不同标准的接入，适用于金融、电商、物联网、工业互联网等行业。

“OpenMessaging 希望成为全球化、无国界、无公司边界，面向云和大数据，多行业领域的一站式方案标准，这也是阿里巴巴第一次在国际社区进行的主导和探索。”项目负责人蒋江伟表示。



阿里云数据库负责人余锋与 Reids 创始人 Salvatore 共同宣布 ApsaraCache 在 Github 上正式开放下载

同时，在云栖大会现场，阿里云数据库负责人余锋与 Redis 创始人 Salvatore 共同宣布 ApsaraCache 在 Github 上正式开放下载。ApsaraCache 是阿里云数据库 Redis 版的分支，适用于更大的数据规模和更多的应用场景。Salvatore 表示，“ApsaraCache 项目开源是一件非常好的事情，将能够吸引全世界更多 Redis 核心专家参与，进一步提升产品的稳定性和可用性。”

峰会上，阿里巴巴还宣布与全球知名代码托管平台 GitHub 和主流 DevOps 技术公司 Hashicorp 达成技术合作关系。阿里云用户不仅可以直接在阿里云上使用 GitHub 企业版，还可以将 GitHub 企业版与 CI/CD 环节联动，同时还可通过 Hashicorp 开源的基础设施编排软件，全面打通云上 DevOps 流程。

拥有五千万软件项目 GitHub 是全球最大的开源代码托管平台，超过 2400 万名用户在 GitHub 上托管、分享代码，为全球创造了数千亿的价值。 Hashicorp 则致力于针对大规模服务化软件开发与部署环节的 DevOps 化支撑。

Mysql 之父、MariaDB 创始人 Michael Widenius，已经连续三年参加云栖大会

Mysql 之父、MariaDB 创始人 Michael Widenius 已经连续三年参加云栖大会，

年过 50 的他依然奋斗在代码第一线，Widenius 表示：“很多 MariaDB 的运用源自我们的开发者，维基百科用的就是 MariaDB，我们也从阿里巴巴中获得了很多开源的支持和贡献，确保能给大家提供功能丰富的数据库产品。”

近年来，阿里巴巴在技术领域投入不断加强，拥抱开源也由来已久，积极加入了包括自由软件基金会、Apache 软件基金会和 Linux 基金会在内的多家国际知名开源组织。目前，阿里巴巴开源和维护的开源项目超过 150 个，涵盖中间件、开发框架、数据库和各种工具类软件。

在开源中国公布的“2016 年度最受欢迎中国开源软件评选 TOP20”榜单中，阿里巴巴独占 4 席。其中 Weex、Ant Design、Dubbo、Fastjson 在 GitHub 上 Star 已经破万，Alibaba 在 GitHub 上 Star 数超过 170,000，组织排名前十。

ApsaraCache 开源地址：

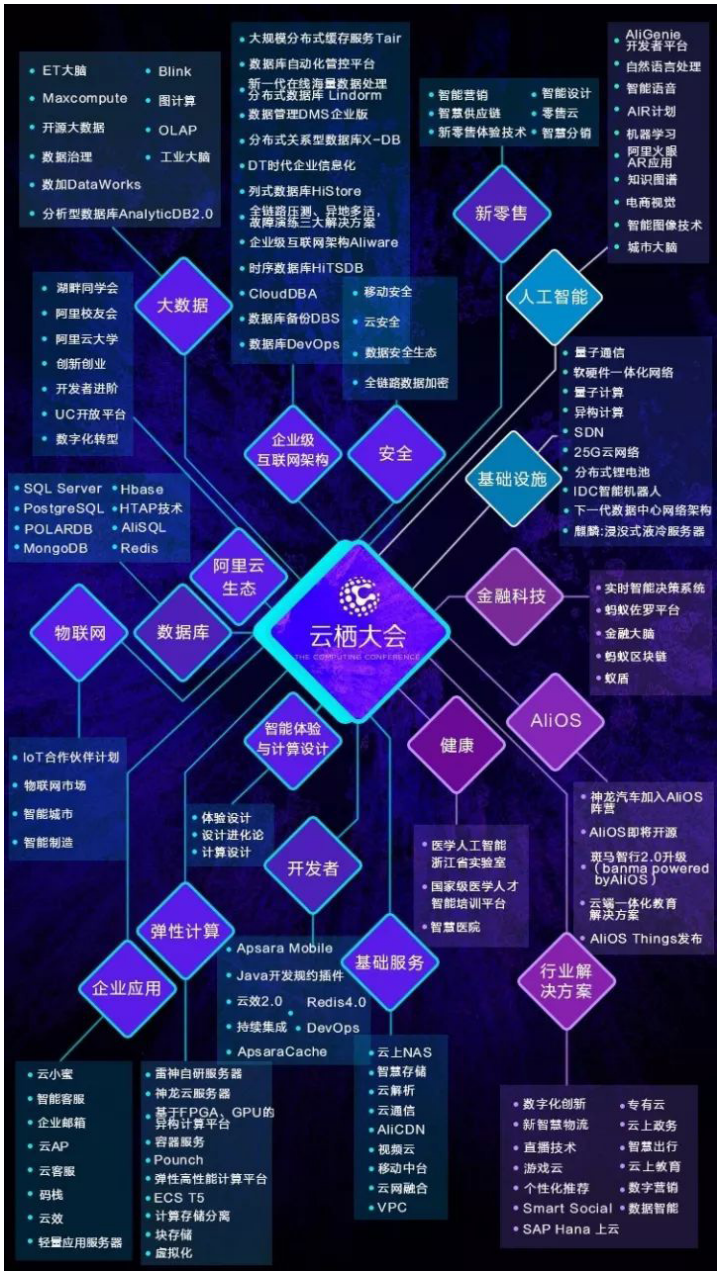
https://github.com/alibaba/ApsaraCache

OpenMessaging 官网：

http://openmessaging.cloud/

四天的云栖大会已经落下帷幕，但是技术前进的脚步从未停止。

近期，阿里妹会把本次云栖大会 140+ 专场的技术干货精华，陆续整理放出。点开放大下面的技术重点一览图，你最期待哪一个主题？不妨在评论区告诉阿里妹吧！


云栖大会
THE COMPUTING CONFERENCE
大数据
ET大脑
Blink
Maxcompute
图计算
开源大数据
OLAP
数据治理
工业大脑
数加DataWorks
分析型数据库AnalyticDB2.0
大规模分布式缓存服务Tair
数据库自动化管控平台
新一代在线海量数据处理分布式数据库 Lindorm
数据管理DMS企业版
分布式关系型数据库X-DB
DT时代企业信息化
列式数据库HiStore
全链路压测、异地多活，故障演练三大解决方案
企业级互联网架构Aliware
时序数据库HiTSDB
CloudDBA
数据库备份DBS
数据库DevOps
新零售
智能营销
智能设计
智慧供应链
零售云
新零售体验技术
智慧分销
人工智能
AliGenie开发者平台
自然语言处理
智能语音
AIR计划
机器学习
阿里火眼AR应用
知识图谱
电商视觉
智能图像技术
城市大脑
安全
移动安全
云安全
数据安全生态
全链路数据加密
企业级互联网架构
基础设施
量子通信
软硬件一体化网络
量子计算
异构计算
SDN
25G云网络
分布式锂电池
IDC智能机器人
下一代数据中心网络架构
麒麟:浸没式液冷服务器
金融科技
实时智能决策系统
蚂蚁佐罗平台
金融大脑
蚂蚁区块链
蚁盾
AliOS
神龙汽车加入AliOS阵营
AliOS即将开源
斑马智行2.0升级（banma powered byAliOS）
云端一体化教育解决方案
AliOS Things发布
健康
医学人工智能浙江省实验室
国家级医学人才智能培训平台
智慧医院
行业解决方案
数字化创新
专有云
新智慧物流
云上政务
直播技术
智慧出行
游戏云
云上教育
个性化推荐
数字营销
Smart Social
数据智能
SAP Hana 上云
基础服务
Apsara Mobile
Java开发规约插件
云效2.0
Redis4.0
持续集成
DevOps
ApsaraCache
云上NAS
智慧存储
云解析
云通信
AliCDN
视频云
移动中台
云网融合
VPC
开发者
体验设计
设计进化论
计算设计
智能体验与计算设计
弹性计算
雷神自研服务器
神龙云服务器
基于FPGA、GPU的异构计算平台
容器服务
Pounch
弹性高性能计算平台
ECS T5
计算存储分离
块存储
虚拟化
企业应用
云小蜜
智能客服
企业邮箱
云AP
云客服
码栈
云效
轻量应用服务器
物联网
IoT合作伙伴计划
物联网市场
智能城市
智能制造
数据库
SQL Server
Hbase
PostgreSQL
HTAP技术
POLARDB
AliSQL
MongoDB
Redis
阿里云生态
湖畔同学会
阿里校友会
阿里云大学
创新创业
开发者进阶
UC开放平台
数字化转型

史无前例开放！阿里内部集群管理系统 Sigma 混布数据

林轩

互联网普及的 20 年来，尤其是近 10 年移动互联网、互联网 + 的浪潮，使互联网技术渗透到各行各业，渗透到人们生活的方方面面，这带来了互联网服务规模和数据规模的大幅增长。日益增长的服务规模和数据规模带来数据中心的急剧膨胀。在大规模的数据中心中，传统的运维方式已经不能满足规模化的需求，于是基于自动化调度的集群管理系统纷纷涌现。

这些系统往往有一个共同的目标，就是提高数据中心的机器利用率。在庞大的数据中心服务器规模下，平均利用率每提高一点，就会带来非常可观的成本节约。这一点我们可以通过一个简单的计算来感受一下。假设数据中心有 N 台服务器，利用率从 R1 提高到 R2，能节约多少台机器？不考虑其他实际制约因素的情况下，假设能节约

X 台，那么我们有理想的公式：

N*R1 = (N−X)*R2

=> X*R2 = N*R2 － N*R1

=> X = N*(R2−R1)/R2

如果我们有 10 万台服务器，利用率从 28% 提升到 40%，那么代入上述公式有：

N= 100000(台),

R1 = 28%,

R2 = 40%

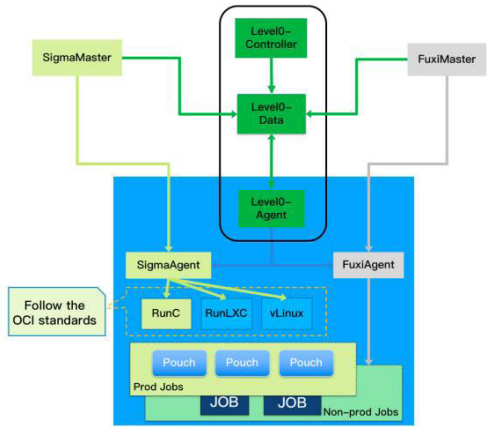
X=100000* (40−28)/40 = 30000(台)

也就是说 10 万台服务器，利用率从 28% 提升到 40%，就能节省出 3 万台机器。假设一台机器的成本为 2 万元，那么节约的成本就有 6 个亿。

但是遗憾的是，根据盖特纳和麦肯锡前几年的调研数据，全球的服务器利用率并不高，只有 6% 到 12%。即使通过虚拟化技术优化，利用率还是只有 7% － 17%；这正是传统运维和粗放的资源使用模式带来的最大问题。调度系统的主要目标就是解决这个问题。

通过资源的精细化调度，以及虚拟化的手段，比如 Virtual Machine 或容器技术，让不同服务共享资源，堆叠高密部署，可以有效的提升资源利用率。但是这种模式对在线业务的应用上存在瓶颈。因为在线业务间的资源共享，高密部署会带来各个层面的资源使用竞争，从而增加在线服务的延迟，尤其是长尾请求的延迟。

对于在线业务来说，延迟的增加往往立刻反应到用户的流失和收入的下降，这是在线业务无法接受的。而近年来随着大数据的普及，对实时性要求并不高的批量离线作业规模越来越大，在资源使用上，逐渐和在线业务的体量相当，甚至超过了在线业务。于是很自然想到，将离线业务和在线业务混合部署在一起运行会怎样？能否在牺牲一些离线作业延迟的情况下，充分利用机器资源，又不影响在线的响应时间？



阿里巴巴从 15 年开始做了这个尝试。在这之前，阿里内部针对离线和在线场景，分别各有一套调度系统：从 10 年开始建设的基于进程的离线资源调度系统 Fuxi（伏羲），和从 11 年开始建设的基于 Pouch 容器的在线资源调度系统 Sigma。 从 15 年开始，我们尝试将延迟不敏感的批量离线计算任务和延迟敏感的在线服务部署到同一批机器上运行，让在线服务用不完的资源充分被离线使用以提高机器的整体利用率。

这个方案经过 2 年多的试验论证、架构调整和资源隔离优化，目前已经走向大规模生产，并已服务于电商核心应用和大数据计算服务 ODPS 业务。混布之后在线机器的平均资源利用率从之前的 10% 左右提高到了现在的 40% 以上，并且同时保证了在线服务的 SLO 目标。

我们了解到，近年来解决资源调度和集群管理领域特定问题的学术研究也在蓬勃发展。但是考虑到学术研究和实际真实的生产环境还是存在很大差异。首先是用于学术研究的机器规模都相对较小，可能无法暴露出实际生产规模的问题；其次是学术研究中所用的数据往往不是实际生产环境产生的，可能会对研究的准确性和全面性产生影响。

因此我们希望将这个阿里内部核心混布集群的数据开放出来，供学术界研究。希望学术界能在有一定规模的真实生产环境数据中，寻找到资源调度和集群管理更好的模式和方法，能够指导优化实际生产场景，将机器利用率和服务质量提高到一个更高

的水平。我们一期先开放 1000 台服务器 12 个小时的数据。

数据格式描述和数据下载链接放在了 github 工程中，欢迎查阅：https://github.com/alibaba/clusterdata

有任何问题和建议可以通过邮件反馈给我们：

alibaba-clusterdata@list.alibaba-inc.com

深度分析 | 阿里开源容器技术 Pouch

孙宏亮

阿里妹导读：近日，阿里正式开源了基于 Apache 2.0 协议的容器技术 Pouch。Pouch 是一款轻量级的容器技术，拥有快速高效、可移植性高、资源占用少等特性，主要帮助阿里更快的做到内部业务的交付，同时提高超大规模下数据中心的物理资源利用率。开源之后，Pouch 成为一项普惠技术，人人都可以在 GitHub 上获取，GitHub 项目地址：

https://github.com/alibaba/pouch

时至今日，全球范围内，容器技术在大多数企业中落地，已然成为一种共识。如何做好容器的技术选型，如何让容器技术可控，相信是每一个企业必须考虑的问题。Pouch 无疑使得容器生态再添利器，在全球巨头垄断的容器开源生态中，为中国技术赢得了一块阵地。

Pouch 技术现状

此次开源 Pouch，相信行业中很多专家都会对阿里目前的容器技术感兴趣。到底阿里玩容器是一个侠之大者，还是后起之秀呢？以过去看未来，技术领域尤其如此，技术的沉淀与积累，大致可以看清一家公司的技术实力。

Pouch 演进

追溯 Pouch 的历史，我们会发现 Pouch 起源于 2011 年。当时，Linux 内核之上的 namespace、cgroup 等技术开始成熟，LXC 等工具也在同时期诞生不久。阿

里巴巴作为一家技术公司，即基于 LXC 研发了容器技术 t4，并在当时以产品形态给集团内部提供服务。此举被视为阿里对容器技术的第一次探索，也为阿里的容器技术积淀了最初的经验。随着时间的推移，两年后，Docker 横空出世，其镜像技术层面，极大程度上解决了困扰行业多年的“软件封装”问题。镜像技术流行开来后，阿里没有理由不去融合这个给行业带来巨大价值的技术。于是，在 2015 年，t4 在自身容器技术的基础上，逐渐吸收社区中的 Docker 镜像技术，慢慢演变，打磨为 Pouch。

带有镜像创新的容器技术，似一阵飓风，所到之处，国内外无不叫好，阿里巴巴不外如是。2015 年末始，阿里巴巴集团内部在基础设施层面也在悄然发生变化。原因很多，其中最简单的一条，相信大家也不难理解，阿里巴巴体量的互联网公司，背后必定有巨大的数据中心在支撑，业务的爆炸式增长，必定导致基础设施需求的增长，也就造成基础设施成本的大幅提高。容器的轻量级与资源消耗低，加上镜像的快速分发，迅速让阿里巴巴下定决心，在容器技术领域加大投入，帮助数据中心全面升级。

阿里容器规模

经过两年多的投入，阿里容器技术 Pouch 已经在集团基础技术中，扮演着极其重要的角色。2017 年双 11，巨额交易 1682 亿背后，Pouch 在“超级工程”中做到了：

- 100% 的在线业务 Pouch 化
- 容器规模达到百万级

回到阿里集团内部，Pouch 的日常服务已经覆盖绝大部分的事业部，覆盖的业务场景包括：电商、广告、搜索等；覆盖技术栈包括：电商应用、数据库、大数据、流计算等；覆盖编程语言：Java、C++、NodeJS 等。

Pouch 技术优势

阿里巴巴容器技术如此之广的应用范围，对行业来说实属一大幸事，因为阿里已经用事实说明：容器技术已经在大规模生产环境下得到验证。然而，由于 Pouch 源自阿里，而非社区，因此在容器效果、技术实现等方面，两套体系存在差异。换言

之，Pouch 存在不少独有的技术优势。

隔离性强

隔离性是企业走云化之路过程中，无法回避的一个技术难题。隔离性强，意味着技术具备了商用的初步条件；反之则几乎没有可能在业务线上铺开。哪怕是阿里巴巴这样的技术公司，实践容器技术伊始，安全问题都无法幸免。众所周知，行业中的容器方案大多基于 Linux 内核提供的 cgroup 和 namespace 来实现隔离，然后这样的轻量级方案存在弊端：

- 容器间，容器与宿主间，共享同一个内核；
- 内核实现的隔离资源，维度不足。

面对如此的内核现状，阿里巴巴采取了三个方面的工作，来解决容器的安全问题：

- 用户态增强容器的隔离维度，比如网络带宽、磁盘使用量等；
- 给内核提交 patch，修复容器的资源可见性问题，cgroup 方面的 bug；
- 实现基于 Hypervisor 的容器，通过创建新内核来实现容器隔离。

容器安全的研究，在行业中将会持续相当长时间。而阿里在开源 Pouch 中，将在原有的安全基础上，继续融合 lxcfs 等特性与社区共享。同时阿里巴巴也在计划开源“阿里内核”，将多年来阿里对 Linux 内核的增强回馈行业。

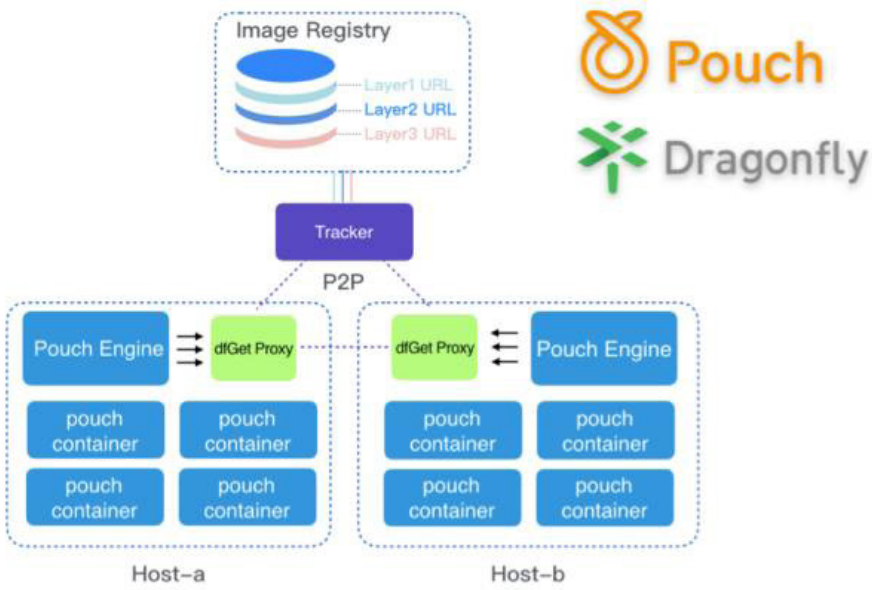
P2P 镜像分发

随着阿里业务爆炸式增长，以及 2015 年之后容器技术的迅速普及，阿里容器镜像的分发也同时成为亟待解决的问题。虽然，容器镜像已经帮助企业在应用文件复用等方面，相较传统方法做了很多优化，但是在数以万计的集群规模下，分发效率依然令人抓狂。举一个简单例子：如果数据中心中有 10000 台物理节点，每个节点同时向镜像仓库发起镜像下载，镜像仓库所在机器的网络压力，CPU 压力可想而知。

基于以上场景，阿里巴巴镜像分发工具“蜻蜓”应运而生。蜻蜓是基于智能 P2P 技术的通用文件分发系统。解决了大规模文件分发场景下分发耗时、成功率低、带宽浪费等难题。大幅提升发布部署、数据预热、大规模容器镜像分发等业务能力。目前，“蜻蜓”和 Pouch 同时开源，项目地址为：

https://github.com/alibaba/dragonfly

Pouch 与蜻蜓的使用架构图如下：



富容器技术

阿里巴巴集团内部囊括了各式各样的业务场景，几乎每种场景都对 Pouch 有着自己的要求。如果使用外界“单容器单进程”的方案，在业务部门推行容器化存在令人难以置信的阻力。阿里巴巴内部，基础技术起着巨大的支撑作用，需要每时每刻都更好的支撑业务的运行。当业务运行时，技术几乎很难做到让业务去做改变，反过来适配自己。因此，一种对应用开发、应用运维都没有侵入性的容器技术，才有可能大规模的迅速铺开。否则的话，容器化过程中，一方面得不到业务方的支持，另一方面也需要投入大量人力帮助业务方，非标准化的实现业务运维。

阿里深谙此道，内部的 Pouch 技术可以说对业务没有任何的侵入性，也正是因为这一点在集团内部做到 100% 容器化。这样的容器技术，被无数阿里人称为“富容器”。

“富容器”技术的实现，主要是为了在 Linux 内核上创建一个与虚拟机体验完全一致的容器。如此一来，比一般容器要功能强大，内部有完整的 init 进程，以及业务应用需要的任何服务，当然这也印证了 Pouch 为什么可以做到对应用没有“侵入性”。技术的实现过程中，Pouch 需要将容器的执行入口定义为 systemd，而在内核态，Pouch 引入了 cgroup namespace 这一最新的内核 patch，满足 systemd 在富容器模式的隔离性。从企业运维流程来看，富容器同样优势明显。它可以在应用的 Entrypoint 启动之前做一些事情，比如统一要做一些安全相关的事情，运维相关的 agent 拉起。这些需要统一做的事情，倘若放到用户的启动脚本，或镜像中就对用户的应用诞生了侵入性，而富容器可以透明的处理掉这些事情。

内核兼容性

容器技术的井喷式发展，使得不少走在技术前沿的企业享受到技术红利。然后，“长尾效应”也注定技术演进存在漫长周期。Pouch 的发展也在规模化进程中遇到相同问题。

但凡规模达到一定量，“摩尔定律”决定了数据中心会存有遗留资源，如何利用与处理这些物理资源，是一个大问题。阿里集团内部也是如此，不管是不同型号的机器，还是从 2.6.32 到 3.10+ 的 Linux 内核，异构现象依然存在。倘若要使所有应用运行 Pouch 之中，Pouch 就必须支持所有内核版本，而现有的容器技术支持的 Linux 内核都在 3.10 以上。不过技术层面万幸的是，对 2.6.32 等老版本内核而言，namespace 的支持仅仅缺失 user namespace；其他 namespace 以及常用的 cgroup 子系统均存在；但是 /proc/self/ns 等用来记录 namespace 的辅助文件当时还不存在，setns 等系统调用也需要在高版本内核中才能支持。而阿里的技术策略是，通过一些其他的方法，来绕过某些系统调用，实现老版本内核的容器支持。

当然，从另一个角度而言，富容器技术也很大程度上，对老版本内核上的其他运维系统、监控系统、用户使用习惯等实现了适配，保障 Pouch 在内核兼容性方面的高可用性。

因此综合来看，在 Pouch 的技术优势之上，我们不难发现适用 Pouch 的应用场景：传统 IT 架构的的迅速容器化，企业大规模业务的部署，安全隔离要求高稳定

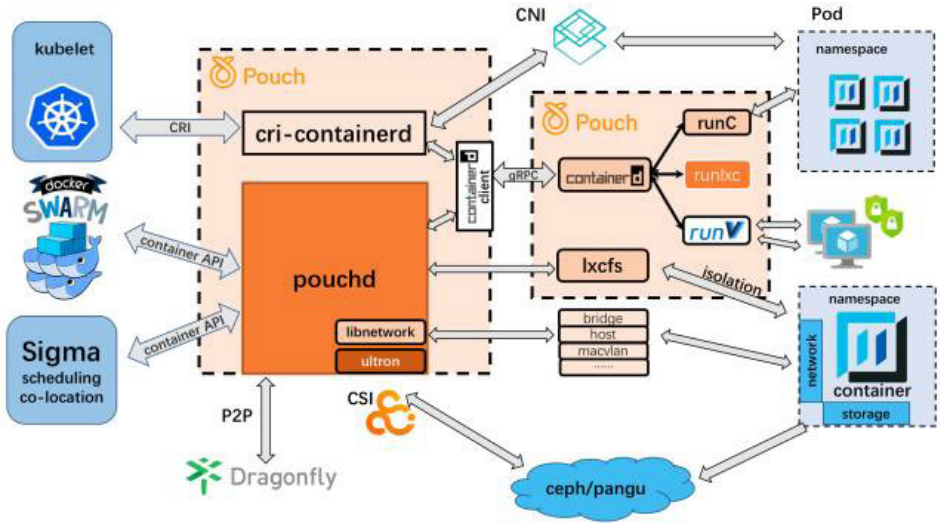
性要求高的金融场景等。

Pouch 架构

凭借差异化的技术优势，Pouch 在阿里巴巴大规模的应用场景下已经得到很好的验证。然而，不得不说的是：目前阿里巴巴内部的 Pouch 与当前开源版本依然存在一定的差异。

虽然优势明显，但是如果把内部的 Pouch 直接开源，这几乎是一件不可能的事。多年的发展，内部 Pouch 在服务业务的同时，存在与阿里内部基础设施、业务场景耦合的情况。耦合的内容，对于行业来说通用性并不强，同时涉及一些其他问题。因此，Pouch 开源过程中，第一要务即解耦内部依赖，把最核心的、对社区同样有巨大价值的部分开源出来。同时，阿里希望在开源的最开始，即与社区站在一起，共建 Pouch 的开源社区。随后，以开源版本的 Pouch 逐渐替换阿里巴巴集团内部的 Pouch，最终达成 Pouch 内外一致的目标。当然，在这过程中，内部 Pouch 的解耦工作，以及插件化演进工作同样重要。而在 Pouch 的开源计划中，明年 3 月底会是一个重要的时间点，彼时 Pouch 的 1.0 版本将发布。

从计划开源的第一刻开始，Pouch 在生态中的架构图就设计如下：



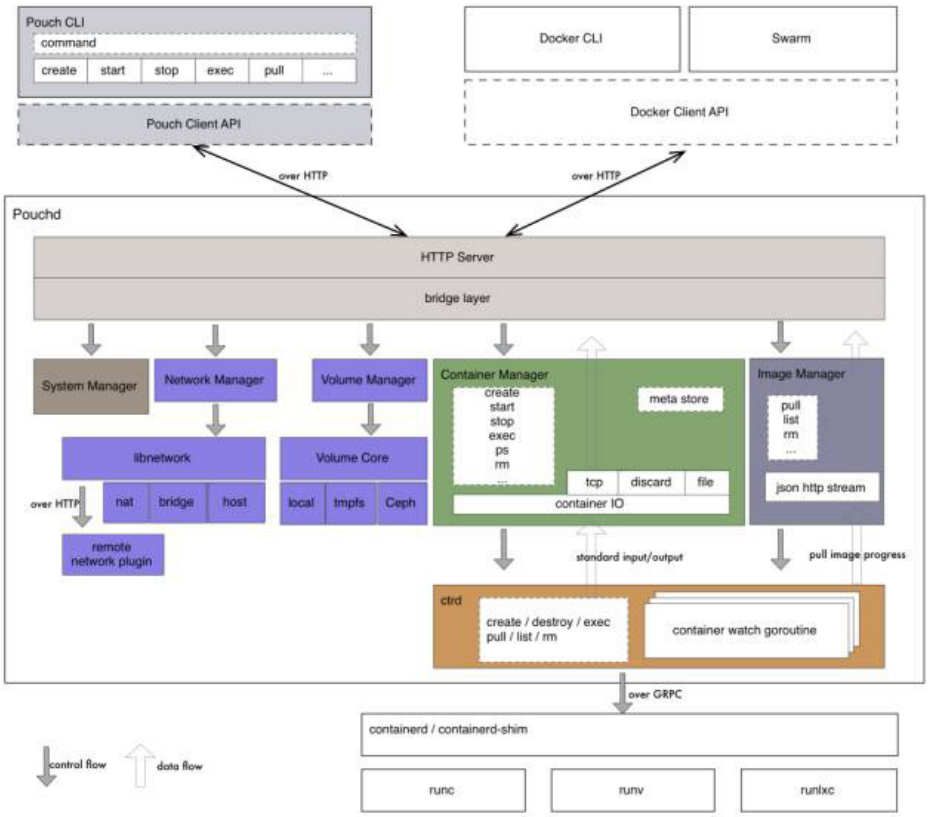
Pouch 的生态架构可以从两个方面来看：第一，如何对接容器编排系统；第二，如何加强容器运行时。

容器编排系统的支持，是 Pouch 开源计划的重要板块。因此，设计之初，Pouch 就希望自身可以原生支持 Kubernetes 等编排系统。为实现这点，Pouch 在行业中率先支持 container 1.0.0。目前行业中的容器方案 containerd 主要停留在 0.2.3 版本，新版本的安全等功能还无法使用，而 Pouch 已经是第一个吃螃蟹的人。当前 Docker 依然是 Kubernetes 体系中较火的容器引擎方案，而 Kubernetes 在 runtime 层面的战略计划为使用 cri-containerd 降低自身与商业产品的耦合度，而走兼容社区方案的道路，比如 cri-containerd 以及 containerd 社区版。另外，需要额外提及的是，内部的 Pouch 是阿里巴巴调度系统 Sigma 的重要组成部分，同时支撑着“混部”工程的实现。Pouch 开源路线中，同样会以支持“混部”为目标。未来，Sigma 的调度（scheduling）以及混部（co-location）能力有望服务行业。

生态方面，Pouch 立足开放；容器运行时方面，Pouch 主张“丰富”与“安全”。runC 的支持，可谓顺其自然。runV 的支持，则表现出了和生态的差异性。虽然 docker 默认支持 runV，然而在 docker 的 API 中并非做到对“容器”与“虚拟机”的兼容，从而 Docker 并非是一个统一的管理入口。而据我们所知，现有企业中仍有众多存量虚拟机的场景，因此，在迎接容器时代时，如何通过统一的运维入口，同时管理容器和虚拟机，势必会是“虚拟机迈向容器”这个变迁过渡期中，企业最为关心的方案之一。Pouch 的开源形态，很好的覆盖了这一场景。runlxc 是阿里巴巴自研的 lxc 容器运行时，Pouch 对其的支持同时也意味着 runlxc 会在不久后开源，覆盖企业内部拥有大量低版本 Linux 内核的场景。

Pouch 对接生态的架构如下，而 Pouch 内部自身的架构可参考下图：

和传统的容器引擎方案相似，Pouch 也呈现出 C/S 的软件架构。命令行 CLI 层面，可以同时支持 Pouch CLI 以及 Docker CLI。对接容器 runtime，Pouch 内部通过 container client 通过 gRPC 调用 containerd。Pouch Daemon 的内部采取组件化的设计理念，抽离出相应的 System Manager、Container Manager、Image Manager、Network Manager、Volume Manager 提供统一化的对象管理方案。



写在最后

如今 Pouch 的开源，意味着阿里积累的容器技术将走出阿里，面向行业。而 Pouch 的技术优势，决定了其自身会以一个差异化的容器解决方案，供用户选择。企业在走云化之路，拥抱云原生（Cloud Native）时，Pouch 致力于成为一款强有力的软件，帮助企业的数字化转型做到最稳定的支持。

Pouch 目前已经在 GitHub 上开源，欢迎任何形式的开源参与。GitHub 地址为：

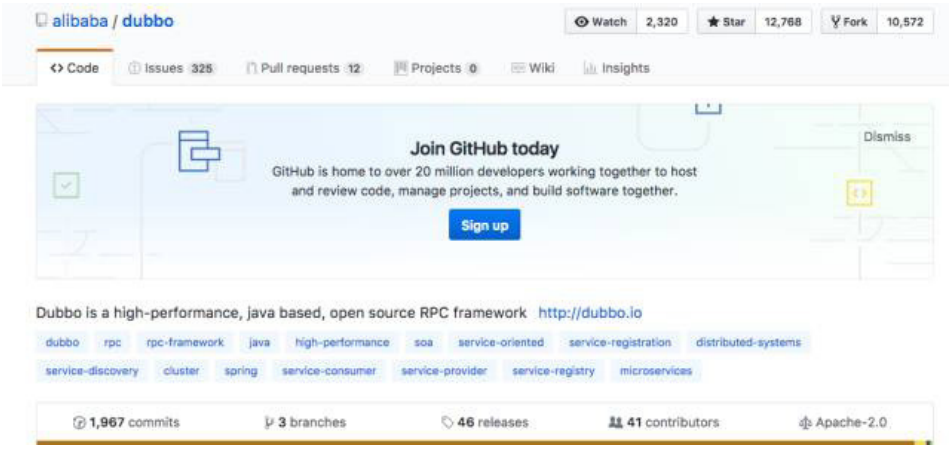
https://github.com/alibaba/pouch

分布式服务框架 Dubbo 疯狂更新

拥抱开源

阿里妹导读：最近，开源社区发生了一件大事——使用最广的开源服务框架之一 Dubbo 低调重启维护，并且 3 个月连续发布了 3 个维护版本。这 3 个维护版本不仅解决了社区关心的一系列问题和需求，还让整个社区的活跃度得到了大幅提升。

Dubbo 启动维护后，阿里中间件（Aliware）组建了由专职人员和 RPC 技术专家组成的虚拟维护团队。通过这篇文章，Dubbo 的虚拟维护团队将和大家分享一些 Dubbo 启动维护的历程、取得的成绩以及后续的规划，具体包括 Dubbo 社区的建设情况、当前的版本维护主线、近期 roadmap 及后续计划等。



Dubbo 是阿里巴巴于 2012 年开源的分布式服务治理框架，目前已是国内影响力最大、使用最广泛的开源服务框架之一，在 Github 上的 fork、start 数均已破万。

在过去几年，Dubbo 开源社区虽然一直有陆续维护，但是由于 Dubbo 用户群体庞大，基础维护根本无法完全满足社区的旺盛需求。随着整个阿里中间件内部技术的迅速发展，如今不仅能够保证集团及客户的系统高效运行，还能抽调更多精力将技术赋能给全社会。开源就是阿里巴巴集团在技术层面赋能的重要领域。

目前，阿里集团正以更高的姿态、更开放的态度拥抱开源。RocketMQ 已被 Apache 社区接纳为顶级项目，OpenMessaging、ApsaraCache 等全球化的开源项目也于云栖大会正式公布，Dubbo 就是在这样的背景下被列入重点维护开源项目。

我们一起总结下 Dubbo 项目的进展、维护后整个社区的变化以及包括后续版本的 roadmap 等，同时也分享一些我们对 Dubbo 期待和想法。

一、社区建设概况

Dubbo 启动维护后我们组建了由专职人员和 RPC 技术专家组成的虚拟维护团队，首先组织专人对官网和使用文档进行了重新整理，后续又以社区反馈为主线发布了 2.5.5 等维护版本。

已发布的内容

- [官网](http://dubbo.io) 发布新版
- 文档重新整理后发布到 [gitbook](http://dubbo.gitbooks.io)，对于 gitbook.io 国内不稳定的问题，计划于下个迭代予以解决
- 09 月 12 日 2.5.5 版本发布
- 10 月 12 日 2.5.6 版本发布
- 11 月 02 日 2.5.7 版本发布

关于三个版本包含的具体内容会在下一节详细介绍，发布时间上基本维持了一月一版本的节奏，有灵活加快的趋势，近期我们仍会保持这种节奏；**发版内容将以维护升级为主基调，遵循以下思路：**

- 优先解决社区内被反复提及的框架缺陷、吸纳开发者贡献的 Pull Request
- 优先支持社区呼声较高的新需求、新特性
- 逐步完善测试、OPS、性能指标等周边基础设施，推动项目管理标准化
- 主动优化或提供一些必要的功能支持

二、已发布版本回顾

本节回顾一下已经发布的 3 个版本的主要内容，详细版本发布记录可通过

Github 追踪。发版内容也体现了当前的维护思路：**发版内容以维护为主，优先解决社区关注度较高问题**

1. 2.5.5 版本：维护后的第一个版本，包括依赖升级和 issue 修复

- 升级了依赖包版本
- 以问题反馈频率和影响面排定优先级，优先解决了几个反馈最多、影响较大的一些缺陷，包括优雅停机、异步调用等

依赖	当前版本	目标版本	影响点
spring	3.2.16.RELEASE	4.3.10.RELEASE	schema配置解析；Http RPC协议
zookeeper	3.3.3	3.4.9	常用注册中心
zkclient	0.1	0.10	zookeeper客户端工具
curator	1.1.16	2.12.0	zookeeper客户端工具
commons-logging	1.1.1	1.2	日志实现集成
hessian	4.0.6	4.0.38	hessian RPC协议
jedis	2.1.0	2.9.0	redis注册中心；缓存RPC
httpclient	4.1.2	4.5.3	hessian等用http连接池
validator	1.0.0	1.1.0.Final	java validation规范
cxf	2.6.1	3.0.14	webservice
jcache	0.4	1.0.0	jcache规范

2. 2.5.6 版本：优先级较高的几个 issue 修复，吸纳社区的优秀 PullRequest

- 通过跟踪 PR、issue 反馈，修复了一些框架缺陷
- 新增了 [Netty4 通信模块](https://dubbo.gitbooks.io/dubbo-user-book/content/demos/netty4.html)、[线程堆栈 dump 特性](https://dubbo.gitbooks.io/dubbo-user-book/content/demos/dump.html)

3. 2.5.7 版本：阶段性完成了社区累积 issue 的处理，同时开始满足社区反映的新需求

- 解决注册中心缓存、监控阻塞 rpc 链路、泛化调用解析等 issue
- 满足社区诉求

- 开放注册 / 监听 ip、port 的配置，以支持 docker 等隔离网络环境部署，[参见示例](https://github.com/dubbo/dubbo-docker-sample)；
- 完善注解配置形式，提供 spring-boot 配置形式支持；

三、近期 Roadmap 与规划

2.5.7 版本后，关注度高的一些 issue 基本都已得到解决，其他一些疑似问题或优先级相对较低的 issue 我们也会开始着手处理，另外我们会投入一定的精力开发新功能及优化代码结构。

近 2~3 个版本，我们计划提供以下内容的支持：

社区的反馈与需求	新功能 新特性	代码优化
解决社区使用过程中遇到的问题或框架缺陷 吸纳社区贡献的新见解、新特性 解决文档在gitbook访问不稳定的问题 提供一些官方的基准性能测试数据 dubbo-admin等官方docker镜像	提供服务延迟暴露、优雅停机API接口 支持RESTFUL风格服务调用 提供netty http的支持 集成高性能序列化协议	路由功能优化 消费端异步功能优化； 提供端异步调用支持 注册中心推送通知异步、合并处理改造

这些内容也在我们近期的候选需求列表中：

- 重构动态配置模块，动态配置和注册中心分离，集成流行的开源分布式配置管理框架
- 服务元数据注册与注册中心分离，丰富元数据内容
- 适配流行的 consul etcd 等注册中心方案
- 考虑提供 opentrace, oauth2, metrics, health,gateway 等部分服务化基础组建的支持

- 服务治理平台 OPS 重做，除代码、UI 重构外，期望能提供更强的服务测试、健康检查、服务动态治理等特性
- Dubbo 模块化，各个模块可单独打包、单独依赖
- 集群熔断和自动故障检测能力

想了解当前版本的具体内容规划及开发进度，可关注 [github milestone](https://github.com/alibaba/dubbo/milestones) 查看详情、反馈建议。

如果您有兴趣，也可以积极参与到 Github issue 问题追踪、gitter 问题讨论中，帮助社区的使用者。**我们正积极吸纳社区活跃的贡献者（代码或问题解答者）加入 Dubbo 组织，共同推动 Dubbo 的进步。**作为一个项目而言，Dubbo 在项目管理及开源社区运营上还有很多不足，我们也会努力向一些更优秀的开源项目靠拢，方便大家能更好的参与到项目建设中。

Dubbo 是阿里巴巴公司开源的一个高性能服务框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案，使得应用可通过高性能 RPC 实现服务的输出和输入功能，和 Spring 框架无缝集成。

Dubbo 包含远程通讯、集群容错和自动发现三个核心部分。提供透明化的远程方法调用，实现像调用本地方法一样调用远程方法，只需简单配置，没有任何 API 侵入。同时具备软负载均衡及容错机制，可在内网替代 F5 等硬件负载均衡器，降低成本，减少单点。可以实现服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的 IP 地址，并且能够平滑添加或删除服务提供者。

技术人生

北大博士在阿里：因为期待，你需要更出色！

星罡

阿里妹导读：施晓罡，花名星罡。阿里巴巴算法工程师，2016 届阿里星。今天阿里妹为大家采访了这位阿里星同学，来看看作为一名阿里星，从校园到企业，从学术界到工业界，都经历了什么！

2016 年，星罡在北京大学毕业并获得了博士学位，研究课题为《动态数据流上的实时迭代计算》。

博士期间，他是一位名副其实的学霸，作为一名学生，在学术上取得了令人瞩目的成绩，曾在包括 SIGMOD 和 TODS 在内的顶级国际期刊和会议上发表多篇学术

论文，并获得校长奖学金、五四奖学金等诸多荣誉和奖励，其中校长奖学金是北大的最高奖学金。

2016 年毕业加入阿里之后，参与了实时计算系统 Blink 的开发工作，负责计算状态的存储、备份和恢复等功能的研发。在 2016 年双 11 期间，Blink 为搜索、推荐和广告等关键业务提供了稳定、高效和可靠的服务。Blink 相关工作同时也在开源社区取得了较大影响。通过将部分工作贡献给社区，他已经在今年 4 月成为 Apache 顶级项目 Flink 的 Committer。

谈渊源：源自游戏，钟情算法

提问：大概从什么时候开始接触到计算机编程方面的东西？又是因为什么喜欢上了计算机？

星罡：第一次接触是小学，那个时候有一个很流行的东西叫裕兴学习机，里面可以学一些简单的 dos、命令等编程的东西。我印象最深的是学习机里面有一个实现超级马里奥的程序，然后我就照着那个教程把马里奥绘出来，虽然也不懂代码的翻译，就照着教程把代码一行行敲进去，这样我就可以控制马里奥到处移动。

这应该是我人生写的第一个程序，虽然很简单，但是写完之后很开心。从那个时候开始一直到上高中我对计算机的理解基本上就等于游戏，因为我特别爱玩游戏，就喜欢上了编程。

提问：那计算机有这么多领域，为什么后来选择了算法呢?

星罡：选算法有很多偶然的原因，最早我在实验室的主要工作是在分布式环境下的数据存储和查询。那个时候互联网技术随着 web2.0 的发展，用户产生的数据有了一个指数级的增加，所以数据的存储和查询是一个很大的问题。但到了后来计算机的技术的发展使得很多这方面的问题被解决了，同时大家对于数据价值的追求越来越高，非常渴望从数据之中寻找到那些潜在的联系，并运用到实际应用中。

所以我在读博的时候有一个很重要的研究课题，就是如何从搜集到的大数据中高效的提取数据价值，如何高效的运行丰富复杂的数据挖掘算法和计算机算法。这个问题非常有挑战，因为我既需要对算法非常了解，还需要能提供一个支持这些算法的平台。所以我就特别希望能够深入了解这些算法，考虑如何把这些算法通过一个真实的系统让大数据运行起来。

提问：在校期间取得了哪些学术上的成绩，哪些对你来说意义非凡?

星罡：博士期间我的课题是《实时数据流上的迭代计算》，这个问题主要是想解决在实时动态的数据流上怎么去高效的编写和运行基于迭代的数据挖掘算法，主要的工作是两块，一个是如何提供一个声明语言，就是一种很简单的高级语言，能够允许用户简单的编写他们的迭代程序。

另外的一个工作就是考虑在分布式环境下在实时数据流上怎么样去跑这些迭代的程序。这些工作在当时非常有创新性，也非常有意思，在 SIGMOD 和 TODS 这些国际的顶级会议和期刊上也发表了一些相关的 paper。另外，这些成果也帮助我也获得了学校的校长奖学金（北大校长奖学金为北大最高规格的奖学金，每年全校仅有 100 个名额）。

谈选择：研究与应用并重

提问：那么在毕业前都有哪些选择？为什么最终选择了阿里？

星罡：找工作的时候我面了不少公司，也拿了挺多的 offer，我当时希望找一个能够继续我博士期间研究方向并在实际中能够发挥作用的工作，所以最早我犹豫要不要去微软或者 IBM 这些公司的研究院，但是接触之后感觉那边的工作可能更偏向于学术研究，在实际应用里并没有想象中那么大。之后我面了很多互联网公司，滴滴、hulu、阿里等等，比较了业务情况、技术氛围等等，发现阿里这边的工作和我研究方向非常匹配。

另外阿里的数据场景非常吸引人，因为阿里的流量非常庞大，另外我觉得阿里在业务上已经非常成功了，有足够的资源在技术上做探索，通过技术创新来获得优势。通过和面世官的交流，我知道实时计算是阿里技术发展的一个重要方向，希望通过实时计算在搜集和广告这些关键业务中，提高用户体验获得更高的收益。所以在阿里我的研究和我的技术可能有更大的发挥空间，所以就果断的过来了。

提问：那么加入阿里的初心是什么呢？想达成什么？

星罡：上大学的时候我开始接触阿里，那个时候经常会上淘宝买东西，感觉淘宝上的宝贝非常多，存量也非常大，那时候我就特别好奇淘宝的系统是怎么去搞定这么庞大的数据量和访问量的。后来随着我研究的深入和对阿里的接触，我更希望在阿里自己能够搭建一个在世界上有影响力的实时计算平台，自己也能够成为一个理论和实践兼具，成为领域内的专家。

谈工作：伴随初心，在压力中成长

提问：从学生到工作的转变，这个过程中有没有遇到过什么问题或者不适应的地方？

星罡：之前在读博士的时候一般我们的工作都自己安排，自己去寻找一些有意思的课题，然后追逐最前沿的科研问题。但是工作之后大部分的工作通常是由业务提出需求，所以在开始的时候总感觉自己的价值没有发挥出来，然后日常的工作节奏也不一样，因为以前读博士的时候可能更关注于自己，每天的时间也由自己安排。

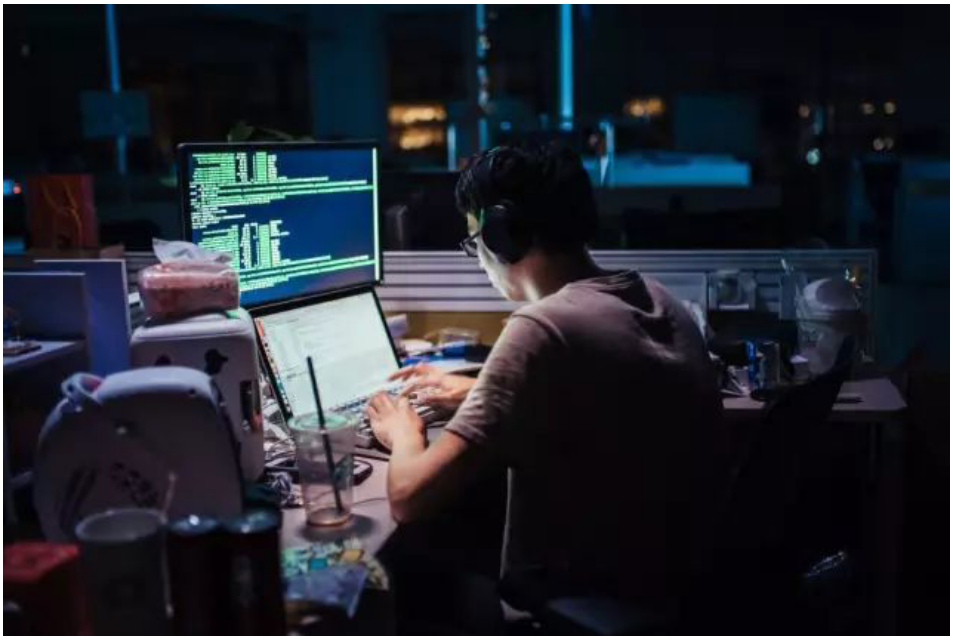
但是现在可能会经常需要和其他团队交流，每天的时间也比较碎片，利用率不会很高。所以刚开始的时候很难适应这样的节奏，也很难适应这样的工作环境，开始的时候工作效率会比较低，需要一个转化的过程。

提问：在工作中，阿里星的光环对你来说意味着什么？

星罡：作为阿里星，会受到来自不同方面的瞩目，可能有更大的舞台或者更灵活的空间给你，也有更多机会和那些大牛去探讨问题，但是可能对我来说意味着一个非常大的压力，因为阿里星其实公司和同事对你有期待，所以说你需要更出色，在工作中要更加努力，然后在个人发展中需要从更多的角度去思考一些问题，尽自己所能帮助团队提高。

提问：到阿里之后做的这些事情你觉得做得最好的是什么？

星罡：在阿里我主要还是做实时计算系统相关的事情，我们基于 Apache 开源社区的一个计算系统 Flink 搭建自己的计算系统，名字叫 Blink，我主要负责计算状态 state 相关的工作，我对 state 接口做了大量的改动，丰富了很多类型，并且优化了状态备份的实现，还提供了很多工具来提高用户程序开发的效率以及用户程序的性能。



另外我们对可靠性做了很多工作，比如前面提到的增量备份来减少每次备份的数据量，提高备份的效率，另外还做了一些在恢复时我们也通过本地复用来使得任务在发生异常时能够很快的恢复。这些工作在实际业务中都起到了很大的作用，在科研社

区也得到了非常好的认可。我们通过把这些工作贡献给社区，我也成长为了 Flink 项目的一个 Committer，这对我来说是在阿里收获的一个非常好的荣誉。

提问：有没有得到你预期的收获，感觉最大的收获和成长是什么？

星罡：有的。一个明显的收获就是我对阿里这样的大公司的业务有了非常深入的了解，然后在技术上就是实际能力有明显的提高，丰富了用户场景，使得我有了非常丰富的调试经验，对峰值环境下的容错、恢复这些都有了非常深刻的认识，在这方面的技术也有了长足的进步。

另外很关键的收获就是深刻理解的平台化和服务化的重要性，阿里丰富的业务不仅要求我们开发满足现有需求的产品，还需要我们能够进一步的分析需求、分解抽象，设计出一个灵活高效的计算平台，来驱动业务进一步的迭代。在阿里丰富的团队合作使得对大项目的需求安排和项目规划有了很好的认识。了解到了如何在一个大项目中规划自己的进度以及开发的能力。

提问：作为一名阿里星，在你眼中，你觉得阿里的哪些东西对你的成长来说是非常有价值的？

星罡：第一个我觉得就是鼓励创新和允许探索的企业文化，这让我们在技术上有很多的大胆的尝试。然后丰富的数据以及业务场景使得我们能够从中获取到丰富的问题，同时能让自己的技术有一个实践和落地的舞台，看到自己的技术在实际中发挥作用，对于自己来说通常是一个很振奋人心的事情。另外在阿里有很多非常优秀的同事，我的身边就有很多在领域内属于顶级的同事，从他们身上获得了很多帮助，对我在技术上的进步有巨大的影响力。

提问：你当初的博士的同学现在都在做什么？和他们相比你觉得现在的你有什么不同？

星罡：有一部分同学在学术界做一些科研研究，还有更多的同学和我一样在互联网公司做业界的东西。还在做研究的那些博士同学们可能会更加关注科研的热点，然后寻求一些新奇大胆的想法。和他们相比我感觉我更加关注技术和业务的结合，考虑这些技术在实际中的可行性以及在真实场景下的表现，想法也和以前有了很大的不同。

给建议：保持初心，长远打算

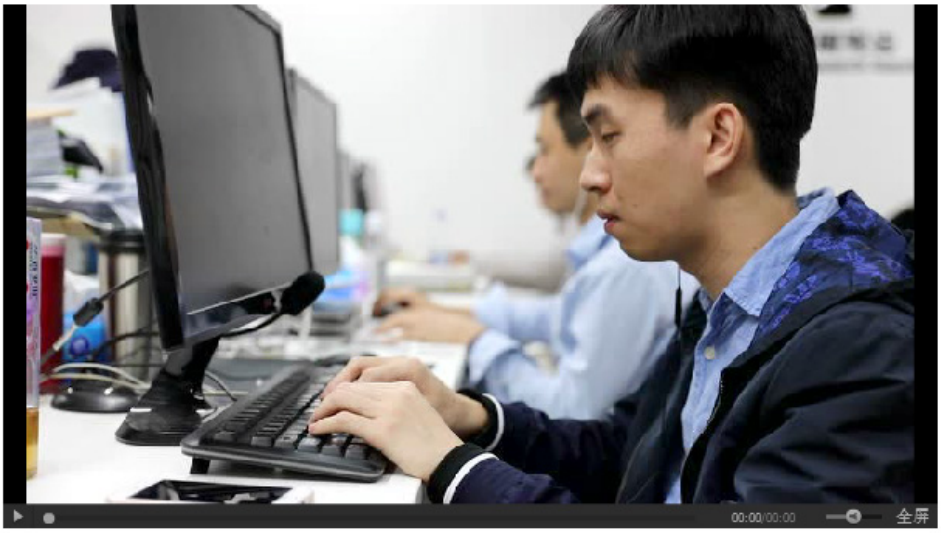
提问：对于即将毕业的师弟师妹们你有没有什么建议？

星罡：我觉得最重要的还是首先要打好自己的基本功，要有过硬的技术能力，然后在找工作的时候一定要想清楚自己想要的东西是什么，保持一颗初心，在选择工作的时候一定要考虑到长远的打算，不要急于眼前的一些利益。

对着黑屏，背代码编程，他的终极目标是让自己失业

阿里味儿

阿里妹导读：明天是 9 月 5 日，一年一度的国际慈善日。阿里妹想带大家认识一位盲人工程师，他的名字叫蔡勇斌，一直努力让自己失业。



4 分钟视频阿斌的故事

蔡勇斌，是深圳信息无障碍协会的一名盲人程序员，负责对互联网产品进行适用于视觉障碍者的改造。他靠“背”代码来编程。

6 岁时，阿斌因为一场意外失明，在特殊教育学校，和大部分视觉障碍同学接受按摩等“手艺活儿”不同，阿斌对计算机产生了浓厚的兴趣。盲人使用计算机，长久以来都必须依靠并不成熟的读屏软件，有一次阿斌不小心把哥哥电脑中的系统文件删除了，电脑开不了机。

重装系统，成了阿斌程序员道路的第一步。回忆起最初装系统的经历时，阿斌说他只能依靠听电脑光驱的转速不同发出的声音，来区别安装系统的进度。一次次的重装系统，就是在光驱的声音中完成。

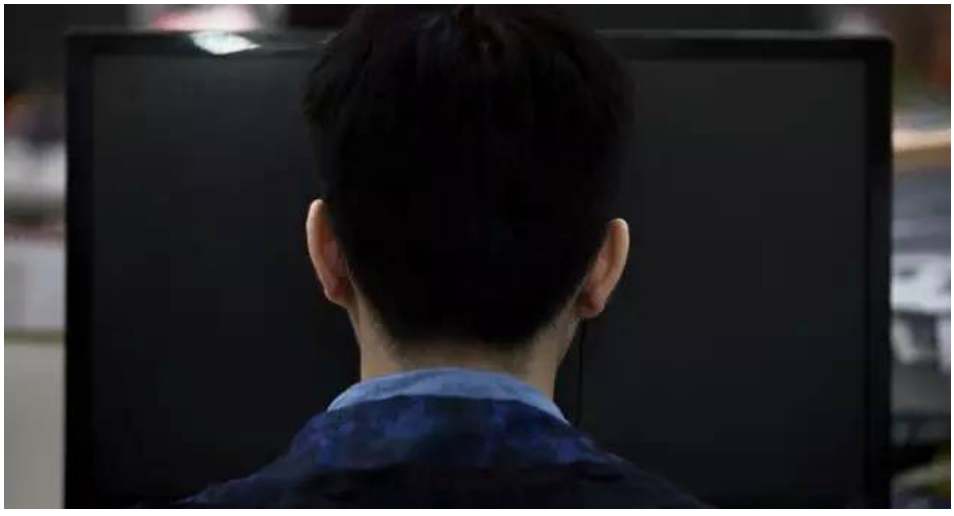
熟悉了计算机的基本操作以后，阿斌在学校里拉着几个同学半逼半求着老师教他们 C 语言，但学校的老师其实也不会，只好自己一边学一边教。

因为没法看到屏幕上的一行行代码，阿斌很多时候只能用死记硬背的笨办法来学习编程。阿斌略显自豪地回忆说，他曾经将数百行代码背下来，就为了在脑子里不断地检查、修改、订正。

百行代码大概是多少个字符？阿斌笑了笑说，也就几万个吧。

盲人程序员的日常

盲人程序员的电脑屏幕是不用打开的，靠键盘和耳机完成工作。每写百行代码，阿斌需要背上万个字符。



耳朵是他们接触外界的窗口

耳朵是阿斌接触外界的窗口，他希望通过互联网可以走的更远。如今，阿斌已经在这个领域成为独一无二的专家，并加入了深圳信息无障碍协会。协会与阿里巴巴、百度等国内知名互联网企业开展了长期合作，致力于对常用互联网产品的改造，为中国 1300 万视觉障碍者创造无障碍使用条件。

手机淘宝、天猫、钉钉等产品都在阿斌和团队优化的产品名录中。谈到这些工作，阿斌说：“视觉障碍者和明眼人的需求都是一样的，我们也要购物、交流、学习。从根本上来说做信息无障碍优化是希望更加平等，可以同样享受到科技带来的红利。在生活和工作上，我们都希望能与常人实现平等。”

深圳信息无障碍协会

午休时间，阿斌和同事还在讨论修改程序，手机淘宝、天猫、钉钉等各个常用软件都在改造计划之中。

阿斌座椅上放着一只可爱的小熊，很多关于信息无障碍化的工作就是在这间办公室完成的。

阿斌和他的团队目前正在进行无障碍化的钉钉，经过与阿里巴巴钉钉团队的工程师合作，已初步通过测试，开始在全国多个视觉障碍学校试点。在浙江省视觉障碍学校，阿斌希望自己改造的通讯软件，能够把师生教学、学生之间的讨论、乃至教务课程组织，都搬到网上。

浙江省盲人学校的学生正在学习使用无障碍版钉钉来交流和学习。通过钉钉无障碍项目，学校将教务和学生管理都实现了互联网化。

“实现工作上的平等，更能实现生活上的平等。盲人同样可以从事复杂的工作，也同样需要进行组织管理和工作管理的软件”，阿斌说，这是他现在专注于对钉钉进行无障碍改造的原因。

“视觉障碍者的世界是不需要灯的”，阿斌用略显诗意的话解释。按他的说法，常人是从纸上到网上，而盲人所缺失的纸这个环节，反倒应该成为把生活和工作搬到网上的优势。

阿斌说，他的目标是最终让自己失业：“我们的工作就是为视觉障碍者进入互联网时代搭建基础，信息无障碍一旦实现，互联网的力量就可以让我们和常人实现生活和工作上的真正平等，我的工作也就不需要继续了。”

阿斌去以前就读的盲校看望小师妹，十年前她还是小孩。

几个从事信息无障碍化工作的小伙伴关系都非常好，他们一般都会结伴出行。

下班回家的路，阿斌已经很熟悉，可以凭经验带同伴走回去。

阿斌一度疯狂迷恋学习编程，曾从早上六点学到晚上八点，做梦都在编程。

我们已经很难想象离开网络该如何生活，对于视觉障碍者来说，无障碍化程序能够帮助他们打开一个全新的世界。阿斌和他的伙伴们希望通过互联网无障碍改造，让他们平等学习、工作和生活。

这是一个光明的世界。

免试晋升为研究员，他在阿里十年经历了什么？

阿里味儿

在上海工作 8 年后，身为部门经理的钱磊，管理着一家 ERP 公司的百十来号员工，“再往上爬就是老板和他儿子了……从这个领域的技术角度来讲算是做到了顶。”05 年，钱磊就开始关注一家名字奇怪，做事也奇怪的公司。

要不要折腾一下？ 2008 年 5 月的一天钱磊对新婚的妻子说，想去杭州发展，那里有个公司叫阿里巴巴……

阿里技术人钱磊

侧面像马总，正面比马总帅

——他的内网标签之一

01 不扒层皮，你怎么知道自己是谁?

告别上海的家和妻子，钱磊拖着行李箱独自来了杭州。

豪情万丈地入职，不料压根没有“蜜月期”……“刚来的那两周，时时刻刻觉得自己是不是要背着包回上海了……”钱磊并不回避刚入职时的囧境。

“不舒服，压力大。在原来的公司里，我就是拍板的人，来了以后发现其它人都挺厉害的。端着架子放不下，总想着自己好歹也是在业内干了八九年的人，在这里被刚毕业一两年的工程师挑战……”

像他这样外地来杭的人，一般入职后就开始着手找房子，钱磊则是在宾馆里住了整整两周。

“那段经历对我来说还是蛮宝贵的，不是有句阿里土话嘛，当你觉得不舒服的时候，就是成长的时候。一个人四平八稳时，很难获得成长。”

02 主管搭了半条命，我学到了两个字“担当”

“硬”着陆半个月后，扒了层“皮”的钱磊又被主管砸了块“石头”。

他们要用技术的手段彻底解决业务问题——为阿里巴巴最早的“产品后台”B2B 的 CRM 做系统重构。这就好比开着汽车换引擎，百年老宅动地基……

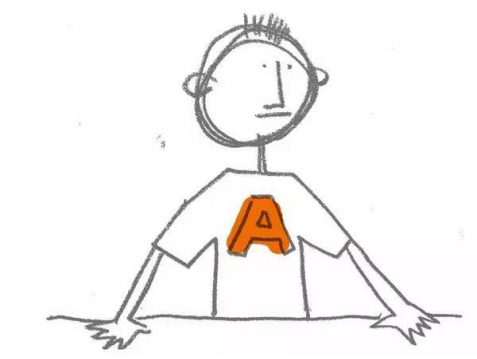
“最重要的一个责任落在了我身上，研发框架。当时业务给我们的时间只有两个月。我用了两周时间搭框架、做论证。对于一个入职不久的新人来讲，挑战非常大。”

这种打仗的感觉，让钱磊渐渐忘记了自己是谁，业务在奔跑，系统延迟一天上线，影响就越大。每天几百人依靠这个系统工作。

“……我的主管被业务方总经理请去喝了两个小时的茶，他是把半条命都搭上了。做不好，没有退路。为了系统上线，当时还停了两天业务，对公司来说每一分钟都是损失。”

系统上线后虽然又经历了多次调试，但这是一次**技术对业务的引领**，而不是以往那样跟在后面打配合。因为框架的通用性还不错。直到现在，这套 CRM 还在使用。

把业务风险和技术风险搭在一起，并不是一个最好的选择，但却是当时唯一的选择。“我从主管身上学到了两个字‘**担当**’，这是一个管理者的气质。”



阿里管理者气质：担当

03 离开场景和业务的修炼都是伪命题

接手用户权限中心的技术团队后，钱磊开始实打实地接触客户。在这几年里，他从偏技术的架构，转为偏业务的架构。“离业务太远，做出来的东西不是场景驱动，后续会有较多的问题。**架构师别把自己修炼成仙了，给公司留下的却是与业务无关的垃圾**。”

当时 B2B 的 ICBU 遇到了用户权益打包的问题，底层需要一个产品化的解决方案，“原来每年销售做完规划以后，技术同学都要算，会花掉多少人力去做一个新的产品配置。每次都靠研发同学冲到代码里面写一大堆。我们做了权益包功能后，只花几个小时就能配置好了。权益的产品化解决了 ICBU 的一个大问题。这又是一次技术引领业务的实例。”

阿里土话：业绩永远都是附属品

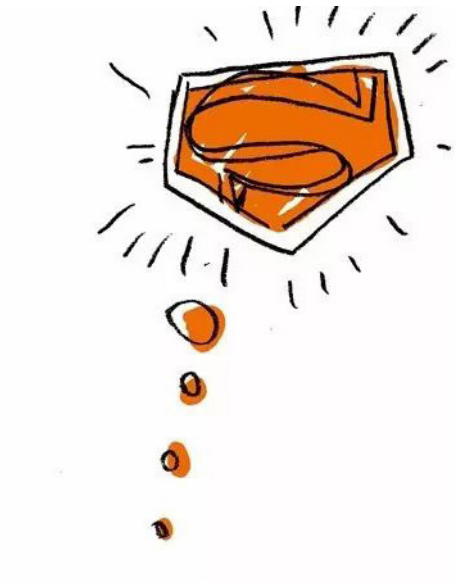
04 越想证明自己死得越快

两年前，钱磊调到共享平台，做会员和安全。他们团队从技术和产品端入手，解决了移动端与 PC 端会员整体链路兼容的问题。降低了一半与帐户相关的客服电话量。“现在帐户相关的客服量在 7% 左右，我们的目标是三年后降到 0。”

从中间件到信息平台，再到共享平台，钱磊接的摊子越来越大，2015 年，入职的第 7 年，他获得了一次提名晋升 P10（研究员）的机会。

“面试的时候，自我感觉还不错，毕竟这几年也做了几件拿得出手的东西……”一个月后，钱磊却得知了自己晋升失败的消息。

“心里肯定有波动啊。”提起两年前的那次晋升面试，钱磊坦白：“我当时还请了两天假，出去放空了一下。想明白了几件事，第一，这几年我在阿里巴巴学到的东西是不是已经远远超出过去？第二，公司给予的平台和空间是不是足够我成长？第三，我为业务创造最大化的价值了吗？当我连续问了自己几个问题后，一下子就看透了。**证明自己没什么大意思，给业务和用户创造价值才是最踏实的。越想证明自己的人，死得越快。**”



阿里土话：总是想要证明自己时，就没有了投入工作的心态

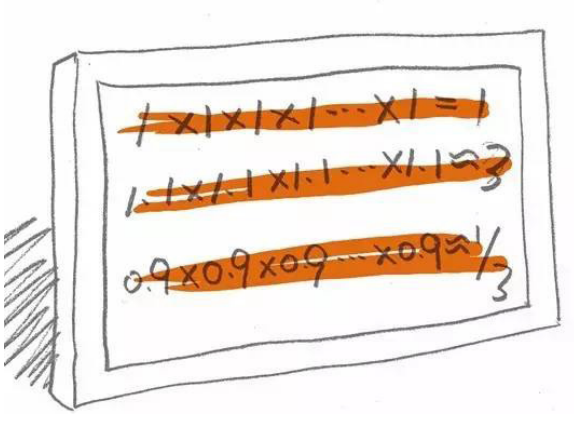
05 善于总结才能获得加速度

钱磊觉得自己这九年来做得最正确的事，**就是坚持总结和反思**。“我几乎每个月都会给自己留出做总结的时候，这个月自己和团队做了什么事，还有哪些提升空间。一定要**趁热打铁记录下来**，反思能让我保持头脑的清晰。

我觉得一个不善于总结的人，就是在吃老本，吃惯性，吃你的智商和知识。**反思就是获得加速度的那个点**。这是工作多年来，让我最受益的一个习惯。”

06 技术人千万别陷在技术的圈子里

钱磊在自己的内网签名里写着“知行合一”四个字。身为技术男，在业务相关的书之外，他独爱王阳明和南怀瑾，“王阳明的知行合一，是我的座右铭，倾听内心的声音，力量强大。南怀瑾的《庄子》我读了好几遍，佩服他为人的格局。我觉得技术同学特别需要打开思路，千万不要局限在技术这个圈子里。**要有自己丰富的阅历，去看很多东西，触类旁通**，在某些时候形成链接，对你的技术思考、业务思考都有帮助。”



阿里土话：成长是自己的事

07 好奇心与三个 Why

带了七八年团队，最多的时候钱磊一年要参加 50 多场面试，有招聘新人的，也

有晋升的。他选人用人会特别看重对方的心态：

“我觉得心态非常重要，我们面试人的时候，一定要去看他的**进取心和好奇心。我觉得这两者是面向未来的能力。**

我会问三个 Why：**为什么这样做？这样做背后有什么问题和思考？你的业务价值支撑点是什么？**我对技术同学的要求是：至少你自己做的事儿，如果到‘P7’这个级别，一定是问不倒的，**一定经得起我问三个‘为什么’。**如果你经不起问的话，我认为你做这个事要么是需求驱动，要么你就是一个吃瓜群众。我非常在意一个人的独立思考能力。”

终极三问，why,how,who

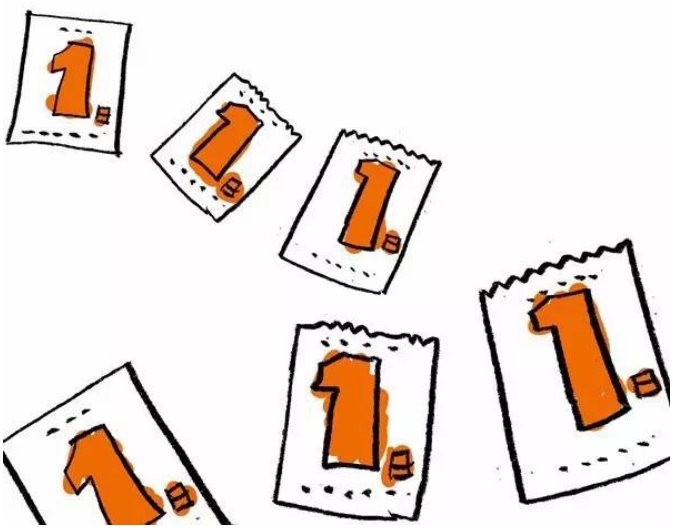
08 一个计较 KPI 的主管

肯定会带出一个计较 KPI 的团队

钱磊说自己的管理之道就是四个字“**言传身教**”，“首先，要做好自己，不要对别人要求很高，对自己要求很低。再有，你是业务驱动、客户价值驱动，还是 KPI 驱动？一个天天计较自己 KPI 的人，肯定带不出一个不计较 KPI 的团队。

我们团队的 KPI 很简单，就是回归到业务场景。我的 KPI 已经连续几年都是以

客服量为衡量点。你这个产品怎么样，上了多少功能，都不是最主要的因素，关键是你最终解决了客户的什么问题。客户用脚来投票。我会带着团队的 D（业务负责人）一起 review 团队的 KPI。



阿里土话：刚工作的几年比谁更踏实，再过几年比谁更激情

09 人生没有白走的路

2017 年 7 月，钱磊通过“绿色通道”（免试）晋升为 P10（研究员），既在意料之中，又在意料之外。尽管不做要求，他还是准备了一个简单的 PPT，那是写给自己的 9 年总结。

钱磊的办公桌面一如典型的技术男那样简单。桌上摆着他 5 岁女儿的照片。问起他印象最深刻的一个瞬间，他不加思索地回答：“2015 年双 11，连续熬了两个通宵后，11 月 13 日的凌晨，我走到园区一号楼取车，眼前的夜空很美，身后是灯火通明的楼宇。就是一种很幸福的感觉。我，在这里，和大家在一起。我们和这家公司，挺伟大的……”

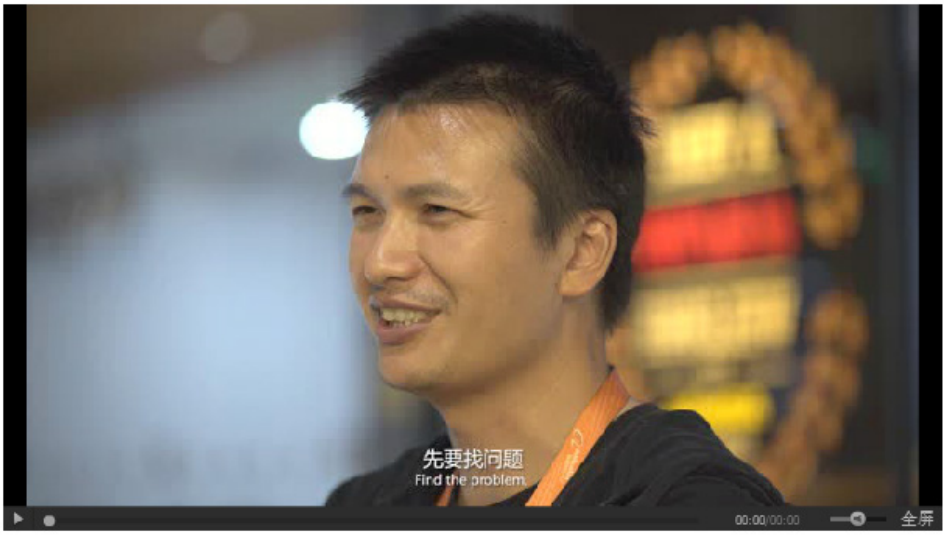
这就是一个阿里技术男的成长史。哪有那么多成长快乐，每一步都不是白走的路。你怎样选择和对待人生，人生就怎么样待你。

多隆：从工程师到合伙人 | 阿里技术人纪录片

孝杨

你简单，世界就会跟着简单。

秋意深浓之际，我们一起走进阿里合伙人多隆的技术世界：



阿里技术人纪录片系列 | HELLO 多隆

1 ◇◆◇

“去了，还是写代码的吗?”

2003 年初春，杭州空气多了一丝暖意。10 分钟前，还在电脑前写代码的小伙子，被叫到马老师办公室，接到一份全英文的合同。从小看到英语就发怵的他，直接抛出了最关心的问题。

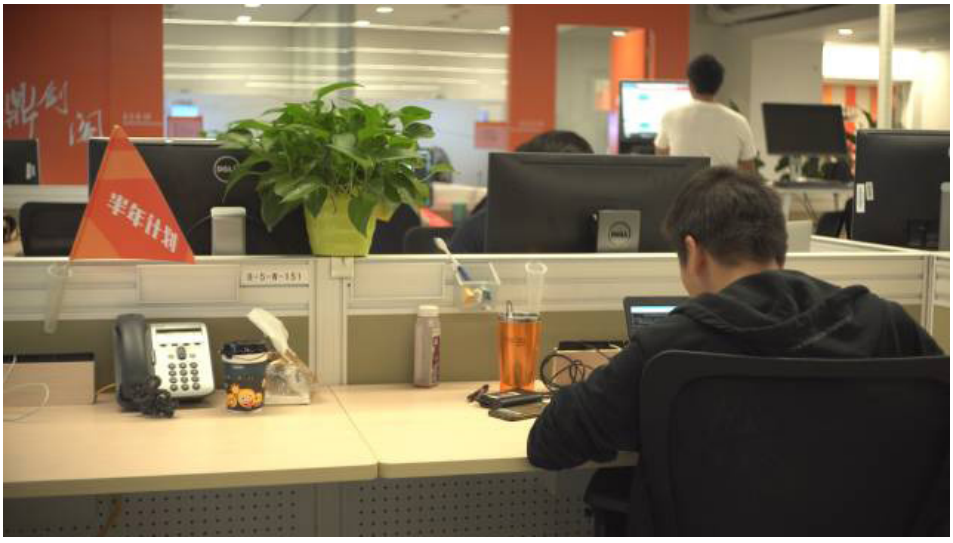
得到肯定的回复后，他毫不犹豫签下自己的名字：蔡景现。

蔡景现，花名多隆，淘宝的第一代程序员。

那时的多隆，还不知道文件里的项目叫淘宝；更不知道，他即将参与的项目，将

改变中国、乃至全球互联网的格局，影响千千万万的小微企业和消费者。

对他来说，只要能写代码，哪里都好。



写代码的多隆背影

2 ◇◆◇

2003 年 4 月 10 日，淘宝初创团队搬到了湖畔花园。为了尽快出活，多隆和另外两位工程师，花了几千块钱，买了一个拍卖类网站的源码，在此基础加了会员、论坛两个功能。一个月后，淘宝的第一个版本上线了。

多隆回忆道，“刚开始的时候，我们每个人都要泡在论坛，客户有什么问题我们都会及时的反应。小宝（淘宝初创团队成员之一）几乎每时每刻都在盯着论坛，有什么问题他立即就说‘请稍等一下，我找总管帮你解决。’然后他会直接找我，我就立即改了，可能就几分钟时间。那时候我们的程序很简单，用 PHP+MySQL 这种结构开发的，响应非常快。后来时间久了，小宝就建议我取多隆这个名字，名字就这么来了。”

直到 2007 年，多隆一个人维护着整个淘宝的搜索引擎，而这还不是他全部的工作。

湖畔花园位于杭州西湖边。在 150 平方米的 4 居室里，诞生了阿里巴巴、淘宝网

当时办公室里放着一堆开着的服务器，吹出比七月烈阳下更热的风：因为限电，空调基本上只能看。

在现任阿里集团 CTO 行癫回忆里，坐在角落的多隆是一个奇怪的人，他总能以很快的速度解决一些别人看起来奇形怪状的问题，哪怕他以前从未接触过。他日复一日年复一年地坐在电脑前忙个不停，一坐就是七年。除了当时晚上下班与多隆一起骑车回家外，几乎没见他怎么闲下来过。

很偶然的机会，行癫听说 NETAPP 不太稳定。当时他恰好在看一个 JAVA 的分布式文件系统，便让多隆研究研究。看了一阵之后，多隆说还不如自己写一个，这次连行癫都表示有点怀疑。不过，行癫相信既然他说行，就一定能行。

没过多久，多隆便完成了原型，三个月后就提供了一个能够运行的产品。而这一切，基本上是他一个人利用平时闲暇时间完成的。这个系统就是现在 TFS，淘宝的文件系统。它成功地解决了大量小尺寸文件分布式存储的可靠性与读写的性能问题。如今，仍然有以 T 字开头的产品运行在淘宝的生产环境中。

淘宝创业团队合影

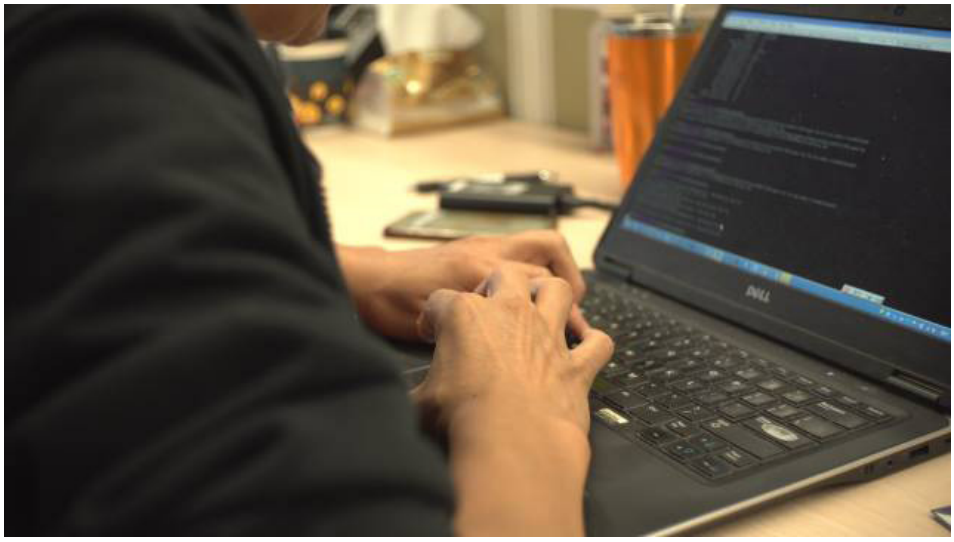
回忆起湖畔花园的日子，多隆说的最多的词，是“开心”。那时候，公司在办公点旁边租了房子，他们经常十二点下班，晚上直接睡在公司宿舍里，过着两点一线的生活。后来，大伙觉得应该加强锻炼，就自娱自乐，学起了倒立。多隆坦诚，“倒立的话，其实刚开始我也很怕。因为一下子上去，万一掉下来，脖子扭了都有的。”

后来他们又发明了新的玩法：叠罗汉，一个一个叠起来，可能会架起十几个人，叠成一排，一群人玩得不亦乐乎。

3 ◇◆◇

看到多隆本人时，你很难将他与阿里合伙人的身份联系在一起。留着板寸头，背着黑色双肩电脑包，从背后看，就像一名理工学院的大学生。

他的桌子更是简单：笔记本、书包、手机、某活动主办方送的保温杯。“本来位子上还放了点书，前几天都搬回家了。”多隆不好意思地笑着解释道。



因为常年使用，键盘上的字母有些掉色了

即使成为阿里合伙人，多隆还是日复一日，年复一年，沉浸在代码的世界中。阿里巴巴西溪园区，有个沿着西溪湿地而建的小花园。员工们闲暇时就会去散散步，看看鱼。但多隆几乎不怎么去转，每天去最近的食堂吃完饭，就回座位敲电脑，如此反复。

阿里食堂，多隆在等对面的小伙伴吃完饭

多隆不擅交际，也不玩社交网络，一般很难在公众场合见到他。但是在阿里内部，却流传着许多关于他的传说。

阿里系统软件事业部研究员毕玄说，“牛 P 很多，但能被称为‘神’的只有多隆一个。他在解决故障方面的能力更是无人能及，在淘宝的故障解决历史上有 N 多的案例。”

阿里中间件研究员小邪随手举了两个例子，“五彩石项目多隆完成了商城搜索的 dump 逻辑，当时如果没有多隆，整个项目需要延后 2 周；还有一次，淘宝 session 框架调用 session_tair 故障，一堆人（应该有 10 人 +）一起排查问题，从 6:00 排查到第二天的 6:00，最后多隆查到了问题。”

同样是阿里合伙人、天猫技术部副总裁范禹，之前在淘宝有个习惯，碰到搞不定的技术问题，就去找多隆。

阿里内网里多隆的个人页面，童鞋们打得最多的标签是“神”“大牛中的大牛”

虽然被很多人视为神，但多隆由衷地觉得自己是一个凡人。他做的最多的就是默默坐在工位上，对着屏幕上的黑框，写代码、解决问题。“就这样搞好了，不知道怎么搞得”，这是多隆经常说的话。

多隆生性内向，不大说话，更多的时候是埋头干活。但是与多隆共事过的阿里人，对他总是丝毫不吝赞美之词。这位从农村出来的工程师，用自己的技术和真诚，赢得了大家的认可与尊重。

4 ◇◆◇

1991 年 9 月，15 岁的多隆进入苍南中学。他开始接触了所谓的“电脑”，其实就是类似小霸王一样的学习机。有次，他看到了一本关于 Basic 编程语言的书，从此就入了迷，开始用它做一些加加减减，或者做 1 到 9 的乘法表、口诀表。

腼腆、害羞是高中林尚游老师对他最深刻的印象。“平时不爱说话，在班级里属于默默无闻型的。但是会经常带着问题来找老师，有时候还会问得脸红。”当时只要学校组织数学竞赛，多隆就会参加，而且每次都能拿奖。与此相反，碰到语文、英语，多隆只能举白旗投降。

1994 年，多隆考上杭州大学。当时高中教育并不像现在这样普及，农村出来的孩子能够一直读到高中甚至大学是件了不得的事情。因为计算机专业太过热门，多隆被调剂到生物科学专业，但这并不妨碍他对技术的热爱。

大学四年，他基本上整天泡在图书馆，机房，还会跑到老师的办公室，把他们的机器拆开搞来搞去，经常被老师骂，然后又自己给乖乖地装回去。

5 ◇◆◇

多隆与阿里童鞋交流技术问题

阿里技术：加入阿里的这十几年当中，有没有特别难过、特别低沉的时候？

多隆：这比较少，因为我这个人没有太多想法的，有什么做什么，看起来也比较傻一点，这样的话挺开心的。我觉得不要知道太多，只要把工作做好就可以了。

阿里技术：如何处理自己内心的焦躁？

多隆：不行的。我静不下来的。

阿里技术：为什么你可以写这么多年的代码，不觉得疲惫？

多隆：我的想法很简单，就是说一定要找到自己感兴趣的事情做。给大家分析的话，我真的很头痛。比如说你不感兴趣的话，可能早上一过来，就在想什么时候下班，怎么还没有下班，这个日子是很难熬的。

我坐火车经常一坐就是七个小时，真的感觉太长了。一到上班的时候，早上 9 点过来，一直到晚上 6 点，我都不知道时间怎么一下子就过去了。因为你有事做，不会觉得很辛苦。所以说真的需要热爱这份工作，要不然你会觉得怎么老是加班。

其实说真的，很多情况下工作跟生活真的是分不开。很多时候工作就是你生活中的一部分了。我只要在电脑前面，坐在那里不动的话，都没事。因为现在年纪大了，可能有时候脖子有点酸。但是如果没有电脑的话，我就不知道做什么。

阿里技术：和团队一起合作，你感触最多的是什么？

多隆：需要有担当精神。不管是谁的问题，我一般都是先去把它看一下，把自己当作问题的终结者。不管谁的问题我都会尽量解它，当然不一定每个都能解得出来。

不要太计较得失。从 B2B 过来，一直到淘宝，其实我什么事情都做。老板觉得有什么事，都会找你去做，这个就是机会。如果这一次他叫你，你不做，下一次肯定不找你做了，就这么简单。

在你做的过程中，自己会学到原来根本不懂的东西。现在技术更新换代是非常快的，我们不懂的东西只会越来越多。只有不停的做，不停去选择，我们才不至于落后。

阿里技术：如何能够像你一样，成为一位大牛，或者说提升自己的技术水平？

多隆：在我看来的话，没有所谓的大神、大牛，真的都是从做项目开始。我刚开始的时候其实什么都不懂的，比如 2000 年进阿里的时候，我连 JAVA 都不懂。当你在工作中遇到问题了，就去找资料，然后去把它弄懂、弄会。只要肯花时间和力气，那你自然而然就会了。

周末我送小孩去少年宫，自己也会带着电脑去看看资料或者写写代码。很多情况下真的没有捷径，就是看你肯不肯花时间，就是这样。

要学会总结。比如，原来经常做一些重复劳动的工作，那你是不是可以做一个工具出来，让自己从这种重复劳动的工作中解放出来。

发现问题，解决问题，不要绕开问题的本身。工程师对于代码，一定要“精益求精”，不论是性能，还是简洁优雅，都要认真打磨自己的作品。

41 岁阿里工程师：35 岁转管理，真的是必经之路吗？

墨玦

阿里妹导读：墨玦，阿里巴巴 iDST 高级技术专家。博士毕业于北京邮电大学，计算机应用专业，目前主要从事语音技术工程化方面的研发。回顾在阿里的三年时光，他感慨良多，写下了这篇总结，与大家共勉。

今年的程序员节，也恰恰是我在阿里工作满 3 年的时候，借此机会盘点一下自己近 3 年来的工作，也为自己后续发展把把关。个人的眼界和思考总是有限的，特别是对于研究和技术领域来说，知道得越多，其实就会知道自己有多无知，从而对未知心生敬畏，并因未知的广阔而兴奋。

我是 1976 年生人，属龙，今年 41 岁，所以可以算是老程序员了，15 年前我读研的时候，就被一起创业的小伙伴称为老何了。我对写代码确实喜欢，大概在 96 年，

大三的时候拿到了高级程序员证书，算是一桩可以拿来吹的事。

博士毕业工作以来，最大的乐趣就是学习和深入思考。所以，从来不以工作过程中项目或者业务的简单或者复杂而困惑。对自身的发展，我一直有一个明确的指导方针：一步一个脚印，提升自己解决问题的能力，不给自己设限。我大概在 10 年前面试一个 40 岁的大叔的时候，就认真地思考过，结论是：我喜欢写代码，我会为此坚持一辈子。

一、既然这个项目这么重要，我们就干吧

回归正题，总结一下在阿里最近 3 年的工作。前两年，我主要在御膳房数据引擎团队做猪头小队长，聊两个重点经历的项目。

第一个就是 5k+，一个通用大数据平台。刚去一个月就赶上这个集合京杭两地的大项目，确实蛮幸运的。在这个项目中印象深刻的有几个地方吧，一个是立项的时候参与方案的讨论，因为涉及京杭两地、跨部门、跨团队的沟通，各个团队的老大难免在一起相爱相杀，我们一帮小弟在旁边参与讨论。一直相杀到凌晨的时候，在自由发

言的阶段，我实在忍不住，跳出来说：既然这个项目这么重要，我觉得我们就干吧。真的是仗着自己在创业公司积累的锐气跳出来说这一句。我觉得大不了就是拼啦。

我说完就意识到这可能给自己的老大带来了巨大的麻烦。很幸运我老大也早就扯烦了，站出来承担责任，一时间各个老大分别出人出枪，一时间群情激奋。最后的责权分配在 20 分钟内就完成了，甚至一位老阿里都哭了。

感谢那一晚上的感觉，也感谢阿里给我一帮很棒的队友和老大。永远记得，后面 996 两个月，在京杭两地互换出差的过程中，我负责两个小模块的项目管理，我发挥自己解决问题快的能力，哪里有窟窿我就去哪里堵，当然，整个团队的人都非常强大也非常努力。在最后项目结束评奖的时候，我拿了个最佳救火队员奖，我真喜欢这个奖。

虽然我也是项目的 PMO 之一，但是除了打酱油，更多的是观摩和学习阿里的项目管理和组织协调。这个项目真的很难，做的过程中经历了各种妥协，项目完工后我们又断断续续还了一年的技术债，但是当时那种拼搏自己和燃烧自己成就 BIG ONE 的感觉，再难重现。后来也跟一些其他公司的同学沟通关于数据平台构建的事情，发现我们真的走得很远。因为是写自己的感受，就不表扬其他同学了，要不然写一本书都可以啦。

二、这不是我一个人的工作，只是努力使它变得完美

第二个项目也带有我自己的强烈特色，我们一直被业务压得很紧。但是对于引擎层研发来说，团队成员也有自己的诉求，而且，系统要逐步完善和改进。在 5K+ 项目完成后，我们依赖的一个重要模块开始频繁出现问题，随着团队间沟通的深入，我们发现对方团队的不稳定和发展方向不确定导致这个模块未来风险非常高。

我们首先想到的是部署一套新的作为过渡。在过渡阶段，为了完成业务的同时来做这件事，我把团队两位同学的一部分业务工作承接过来，腾出人力开始做这件事。两位同学远赴杭州，出差一个月，把平台基本接过来，保障了我们业务的平稳运行。

随后，我们调研后果断抛弃了这个模块的原有实现，调集团队的技术力量重新规划设计新的模块，除了替换，更重要的是为了发展。这一步走出后发现后面很多东西

都活了，数据服务开始作为一个重要的点慢慢从整个平台浮现出来，与数据团队产生更深度的互动，进而随着原数据服务的不稳定，催生了新的数据服务平台。

这不是我一个人的工作，我只是努力使它变得完美。当初万分纠结，每一步都步步惊心，现在相信每个参与这个项目的同学，心里都是美好的回忆。而且，我们不仅通过这个项目成就了自己，更成就了兄弟团队，成就了御膳房的发展。说大了。

三、技术挑战是猎物，是机会，是战功

在这三年里，我不认为自己遇到了很大的技术挑战，很多事情提前想到，组织技术专家提前讨论，当和团队在一起的时候，技术挑战是猎物，是机会，是战功。

举一个简单的例子来说明我做一个项目的过程，例如：我们要实现一个限流的服务，就是允许一个租户的 QPS 最高多少。首先需要界定问题的边界，包括：要不要考虑网络层攻击（可能被其他模块处理掉）、未来一段时间业务的规模，系统稳定性、架构扩展性等。

这些问题确定后，会有一系列的技术方案成为技术选型，那么如何判断采用什么

技术方案，当时不是最新最酷的最好的，要考虑将来部署环境，上下游环境。最重要的这是一个分布式需求。简单来说，N 台服务器共同维护一个 QPS 值，这后面的分布式理论，朴素来说就是 CAP，在 CAP 三者不能同时满足的情况下，应该降低那个并保证业务的目标。

为此，我们参考分布式的 BASE 模型，降低了一致性的需求，采用分区和主体配额池结合的思路解决了大租户的流量控制，而针对长尾租户采用了另一套控制来保证精确限流。

同时考虑第三方模块和非关键模块挂掉或者降级中的应急预案，以及模块部分宕机或者机房断电导致的服务不可用，以及某些服务的单点问题等而设计完整的稳定性方案。当然，最后还要考虑扩展性方案，如果需求规模突然变大，但是整体是有边界的。

总结起来，整个项目要有明确的目标和阶段以及对应的关键指标，做到可观测、可评估、可扩展、可恢复、以及容易交付给其他人继续研发和维护。

我不认为自己做得很好，但是当逐步完善一个系统时，真的蛮快乐的。

四、我不认为做到 35 岁转管理是必要的

现在细想起来，在我的成长道路上，给我提供技术指导的大牛真的很少，更多的时候，我更喜欢向任何一个我遇到的人学习，我学习的目的不是超越别人，而是超越自己。

转岗到 iDST 团队后，我坐在 iDST 老大的旁边，有幸耳濡目染研发和管理达人的工作，受益匪浅。我一直觉得，程序员没有人能教会，要的是自己的钻研和用心的学习。包括我原来御膳房团队的同学和现在 iDST 的同学，在合作和交流中，总能发现那些令你眼前一亮的优点和闪光点，这如何不欣喜?

另外，我觉得保持持续的思考和以开放的态度与其他同学沟通非常重要，互通有无。我觉得我目前最大的技术优势可能还是在工程领域，主要是服务器后端研发以及数据平台建设这边，主要是思考和经验比较多。我会在理论和实践两个方面继续增强自己的能力。与此同时，我一直在储备自己在人工智能方面的知识。得益于我博士论文期间在信息检索方向的深入思考，我很早就发现了这个能让我着迷的领域。

这里稍微聊两句人工智能领域，虽然一些人说这里面也就是所谓的调参、特征

工程、训练深度网络等。这些真的是门外汉才会这么说。真正里面需要的是理性的思维，解决这种没有明确的路径可寻的问题需要非常深入的思考、尝试，经历无数次失败可能有一点小收获，然后还要把这种小收获用最精准的数学语言阐述出来，为后续的研发铺平道路。

目前我刚刚读完 NLP 领域的一本综述——《统计自然语言处理》，正在重新构建自己的概率论和统计学的知识体系，尽量做到基本的概念信手拈来，然后找一个小的领域进行深入的思考和尝试。对于我来说，找一个点建立一个模型，改改参数出一篇论文的诱惑不大，我希望能够研究前人的研究成果，在理论深度有一定的突破。

在这个领域，那些谈 35 岁就转管理的程序员可能根本就无法明白。当遇到一个可以持续投入精力去钻研，并且越钻研觉得越难的事情的时候，对于我来说，何其幸运。更何况，我不认为做到 35 岁转管理是必要的。管理并不好做，很多人以为管理就是分配工作，技术人员都是心高气傲之辈，能力低的根本领导不了，只能领导能力更低的。而且真正的管理和写代码一样，也是一门学问，一门理论与实践相结合，需要边探索边实践的学问。人家让你领导，是把自己的发展托付到你的手里，所以是更重的责任。

from 朋友虞老板拍摄，当我们吟诵春风不度玉门关的时候，玉门关其实是这个小土坑了。所以要辩证思考那些流传的话，比如 35 岁前必须转管理。

我主要利用上下班路上的时间来做机器学习的钻研，每天大概 3 个钟头左右，上班时间主要还是写业务代码，我热爱写码，调通一个功能的感觉真爽！

举个例子，《统计自然语言处理》，我用了将近一年的时间，在上班路上读完，在读到机器翻译模型的时候，深深的迷醉于那 5 个 IBM 工程师发明的模型。这些模型发明的过程，思考的过程，是我学习的对象。

最后想说的是，在阿里，一样经历繁华，经历迷茫，经历失落甚至冷落，最重要的是守住自己的技术之心，与大家共勉。

PS：如果你熟悉语音方面的算法，希望加入一个潜力无限的团队，我在阿里等你。我的邮箱：zhaowei.hzw@alibaba-inc.com

技术变化那么快，程序员如何做到不被淘汰？

空融

阿里妹导读：写了这么多年的代码，你是否曾经有过这样的迷茫和困惑——技术发展日新月异，奋力追赶的我们，究竟是技术的主人还是技术的奴隶？今天，我们邀请到了蚂蚁金服的技术专家空融，一起来聊聊技术人的软件世界观。

在浩大的软件世界里，作为一名普通程序员，显得十分渺小，甚至会感到迷茫。我们内心崇拜技术，却也对日新月异的技术抱有深深的恐惧。有时候我会思考难道在技术领域内不断紧跟新潮，不断提升技能就是我的价值所在？那么我是技术的主人还是技术的奴隶？

人之所以迷茫往往是找不到工作生活的重心，感受不到工作或生活的价值。那么什么是价值呢？说的大一点就是我改变了世界，说的小一点就是我的所作所为改善了某些问题。如果不清楚自己的行为、目标、价值三者的关系，那么又何来重心？又如

何能分得清重要性与优先级呢?

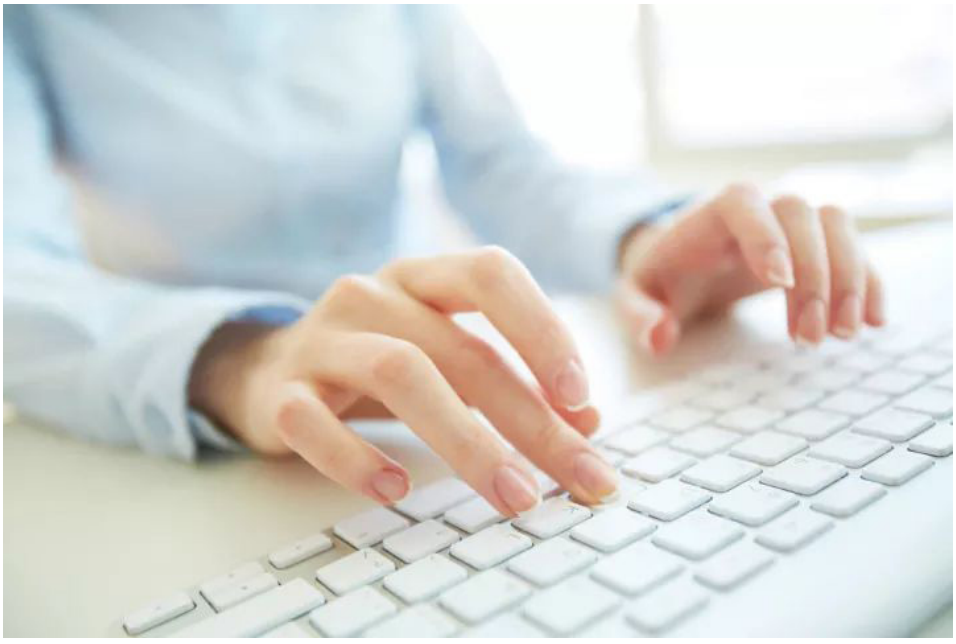
程序员的迷茫不仅仅是面对技术繁杂的无力感，更重要的是因为长期埋没于软件世界的浩大的分工体系中，无法看清从业务到软件架构的价值链条，无法清楚定位自己在分工体系的位置，处理不好自身与技术、业务的关系所致。

很多程序员打心底不喜欢业务，这一点我曾经也经历过，我更宁愿从事框架工具、技术组件研究的相关事情。我有个朋友经常吐槽我说：”你们天天加班加点写了那么多代码，然后呢？有改变什么吗？还不是写出了一堆垃圾。”仔细想想很多时候业务在我们脑海中存留的只是逻辑和流程，我们丢失的是对业务场景的感受，对用户痛点的体会，对业务发展的思考。这些都是与价值紧密相关的部分。我们很自然的用战术的勤快掩盖战略的懒惰！那么这样的后果就是我们把自己限死在流水线的工位上，阉割了自己能够发现业务价值的能力，而过多关注新技术对职场竞争力的价值。这也就是我们面对繁杂技术，而产生技术学习焦虑症的根本原因。

业务、技术与软件系统的价值链

那么什么是业务呢？就是指某种有目的的工作或工作项目，业务的目的就是解决人类社会与吃喝住行息息相关的领域问题，包括物质的需求和精神的需求，使开展业务活动的主体和受众都能得到利益。通俗的讲业务就是用户的痛点，是业务提供方（比如公司）的盈利点。而技术则是解决问题的工具和手段。比如为了解决用户随时随地购物的业务问题时，程序员利用 web 技术构建电子商务 App，而当需求升级为帮助用户快速选购商品时，程序员会利用数据算法等技术手段构建推荐引擎。 技术如果脱离了业务，那么技术应用就无法很好的落地，技术的研究也将失去场景和方向。而业务脱离了技术，那么业务的开展就变得极其昂贵和低效。

所以回过头来我们想想自己没日没夜写了那么多的代码从而构建起来的软件系统，它的价值何在呢？说白了就是为了解决业务问题，所以当你所从事的工作内容并不能为解决业务问题带来多大帮助的时候，你应该要及时做出调整。那么软件系统又是如何体现它自身的价值呢？在我看来有如下几个方面的体现：



业务领域与功能：比如支付宝立足支付领域而推出的转账、收款功能等，比如人工智能自动驾驶系统等。

服务能力：这就好比火车站购票窗口，评判它的服务能力的标准就是它能够同时处理多少用户的购票业务，能不能在指定时间内完成购票业务，能不能 7*8 小时持续工作。对应到软件系统领域，则表现为以下三个方面：

- 系统正确性（程序能够正确表述业务流程，没有 Bug）
- 可用性（可以 7 * 24 小时 * 365 不间歇工作）
- 大规模（高并发，高吞吐量）

互联网公司正是借助大规模的软件系统承载着繁多的业务功能，使其拥有巨大的服务能力并借助互联网技术突破了空间限制，高效低廉解决了业务问题，创造了丰厚的利润，这是人肉所不可比拟的。

理解了这一层面的概念，你就可以清楚这个价值链条：公司依靠软件系统提供业务服务而创造价值，程序员则是通过构建并持续演进软件系统服务能力以及业务功能以支撑公司业务发展从而创造价值。

有了这个价值链条，我们就可以反思自己的工作学习对软件系统的服务能力提升起到了多大的推动作用？可以反思自己的工作学习是否切实在解决领域的业务问题，还是只是做一些意义不大的重复性工作。

前两天面试了一个候选人，他的工作是从事票务系统开发，他说自己在研究 linux 内核与汇编语言，我就问他 linux 内核和汇编语言的学习对你的工作产生了哪些帮助？能否举一个例子？他哑口无言，我内心就觉得这样一个热爱学习的好苗子正迷茫找不到重心，正在做一件浪费精力的事情。正确的学习方式应该是将学习与具体业务场景结合起来，和公司通过软件系统开展业务服务而创造价值，程序员通过提升软件系统服务能力创造价值这一链条串接起来，从对这些价值产生帮助的程度去思考优先级。学习本身没有错，错的往往就是那颗初心。

现在你再来看高并发分布式相关的知识，你会发现并不是因为这些知识比较高深、比较时髦，很多公司有需求才值得学习，而是他们对价值链条有着实实在在的贡献。

价值驱动的架构

一谈到软件系统，人们免不了想起架构这件事来。之所以此处去谈及架构是因为每一个程序员本质都是软件架构体系中的一分子，我们可能深埋于体系流水线之中，感受不到位置和价值。但如果站在架构这一高度去看这些问题则将会非常透彻。那么架构究竟是什么？和上述的价值链又有什么关系呢？

什么是架构？

在我看来软件架构就是将人员、技术等资源组织起来以解决业务问题，支撑业务增长的一种活动。可能比较抽象，我想我们可以从架构师的一些具体工作任务来理解这句话含义：

组织业务：架构师通过探索和研究业务领域的知识，构建自身看待业务的”世界观”。他会基于这种认识拆分业务生命周期，确立业务边界，构建出了一套解决特定业务问题的领域模型，并且确认模型之间、领域之间的关系与协作方式，完成了对业务领域内的要素的组织工作。

组织技术：为了能在计算机世界中运作人类社会的业务模型，架构师需要选用计算机世界中合适的框架、中间件、编程语言、网络协议等技术工具依据之前设计方案组织起来形成一套软件系统方案，在我看来软件系统就像是一种技术组织，即技术组件、技术手段依据某种逻辑被组织起来了，这些技术工具被确定了职责，有了明确分工，并以实现业务功能为目标集合在了一起。比如 RPC 框架或消息队列被用于内部系统之间的通信服务就如同信使一般，而数据库则负责记录结果，它更像是一名书记员。

组织人员：为了能够实现利用软件系统解决业务问题的目标，架构师还需要关注软件系统的构建过程，他以实现软件系统为号召，从公司组织中聚集一批软件工程师，并将这些人员按不同工种、不同职责、不同系统进行组织，确定这些人员之间的协作方式，并关注这个组织系统是否运作良好比如沟通是否顺畅、产出是否达到要求、能否按时间完成等。

组织全局，对外输出：架构师的首要目标是解决业务问题，推动业务增长。所以他非常关心软件的运行状况。因为只有在软件系统运行起来后，才能对外提供服务，

才能在用户访问的过程中，解决业务问题。架构师需要关注运行过程中产生的数据比如业务成功率，系统运行资源占用数据、用户反馈信息、业务增长情况等，这些信息将会帮助架构师制定下一步架构目标和方向。

所以软件架构不仅仅只是选用什么框架、选用什么技术组件这么简单。它贯穿了对人的组织、对技术的组织、对业务的组织，并将这三种组织以解决业务问题这一目标有机的结合在了一起。

很多面试的候选人在被问及他所开发的系统采用什么架构的问题时，只会罗列出一些技术组件、技术框架等技术要素，这样看来其根本没有理清架构的深层含义。也有一些架构师只专注对底层技术的研究，以为打造一个卓越的系统是非常牛逼的事情，可是他忽略了软件系统的价值是以解决业务问题的能力、支撑业务增长的能力为衡量标准，所以最后生产出了很多对组织，对业务没有帮助的系统。

成本与收益

正如之前所说软件系统只有在运行的时候才能创造价值，也就是说软件系统能否

7*24 小时 * 365 天稳定的工作关系到公司的收益水平。所以开发团队对生产环境的发布总是小心翼翼，对解决生产环境的问题总是加班加点。而软件系统的成本则体现在软件构建过程，这时候我们就能理解那些工程技术如项目管理、敏捷开发、单元测试、持续集成、持续构建，版本管理等的价值了，他们有的是保证软件系统正确性，有的是为了降低沟通成本，有的是为了提升开发效率等但总的来说就是为了降低软件的构建成本。所以在提升系统服务能力，创造更多业务收益的同时，降低构建成本也是一种提升收益的有效手段。

作为一名软件工程师而言，我们往往处在软件构建过程体系中的某个环节，我们可以基于成本与收益的关系去思考自己每一项技能的价值，学习新的有价值的技能，甚至在工作中基于成本与收益的考量选择合适的技术。比如在逻辑不大发生变化的地方，没有必要去做过多的设计，应用各种花俏的设计模式等浪费时间。这样我们才能成为技术的主人。

架构目标需要适应业务的发展

架构的目标就是为了支撑业务增长，就是提升软件系统的服务能力。可是话虽说如此，但真实却要做很多取舍。比如对初创团队而言，其产品是否解决业务问题这一设想还没得到确认，就立即去构造一个高性能、高可用的分布式系统，这样的架构目标远超出业务发展的需求，最后的结果就是浪费大量人力物力，却得不到任何起色。架构师需要审时度势，仔细衡量正确性、大规模、可用性三者的关系，比如今年业务蓬勃发展日均订单 300 万，基于对未来的可能预测，明年可能有 3000 万的订单，那么架构师应该要着重考虑大规模和可用性。而且每一点提升的程度，也需要架构师衡量把握，比如可用性要达到 2 个 9 还是 3 个 9。

回顾自己以往的工作很多时候就是因为没有确立架构目标导致浪费了组织很多资源，比如在之前的创业团队中，由于本人有一定的代码洁癖，经常会花费很多时间和同事计较代码质量，这样本可以更快上线的功能却需要被延迟，当时过度追求正确性的行为是与创业团队快速验证想法的业务需求不匹配的。

另外一点比较深刻的案例则是在本人担任一个技术团队负责人的时候，在一次述

职报告的时候，leader 问我对接下来团队工作有什么计划？我当时说了一堆什么改进代码质量，每天晨会，任务透明化，建立迭代机制等等，然后就被各种批驳一通。当时团队基本以外包人员为主，人员水平较差，开发出来的金融系统也是千疮百孔而这条业务线最重要的业务价值则是按计划实现潜在投资方的需求，争取拉到投资。所以不久 leader 就召集测试架构的相关人员与我这边一同梳理对核心功能的测试工作，将研发、测试、上线的流程自动化。

当时并不理解这样做核心价值是什么。但回过头来看这样的工作方式恰好符合了业务发展的需求，即确保系统是符合设计需求的，保证系统达到可接受的正确性，为后续能过快速前进打下基础，最重要的是为企业降低了构建成本。所以程序员想要工作出业绩，必须认清楚系统背后的业务价值，按价值去梳理工作优先级，而不是像我一般过度纠结细节，追求技术理想化。

成也分工，败也分工

正如在程序员的迷茫那一章节提到的：程序员的迷茫因为长期埋没于软件世界的浩大的分工体系中，无法看清从业务到软件架构的价值链条，无法清楚定位自己在分工体系的位置，处理不好自身与技术、业务的关系所致，所以在这里我想谈谈分工。架构师为了使软件系统更好的服务业务，必然将软件系统生命周期进行拆分，比如分出开发生命周期、测试生命周期、用户访问生命周期、软件运维生命周期，并根据不同的生命周期划分出不同的职责与角色。

比如开发人员负责开发周期负责完成软件研发，测试人员负责对开发人员交付的成果进行测试等，于是就形成了分工。一旦分工形成，每一个分工组织都会有自己的价值追求，架构师关注的顶层的价值即软件系统能否支撑业务增长被分工的形式打碎到各个组织中。分工是有其价值的，他使得复杂昂贵的任务可以被简单、并行、可替换的流水线方式解决。但久而久之，价值碎片化的问题就出现了，比如测试人员只关注找出更多问题，开发人员只关注快速开发更多的系统，运维人员只关注保障系统稳定。

三者之间常常都只站在自己的立场去要求对方怎么做，没有人再关注整体价值，产生诸多矛盾增加软件实施成本。而身处流水线中的一员，又因为困扰于重复性工作，迷茫于工作的意义，甚至感觉自己做为了人的创意与灵感都被扼杀了。所以我的朋友吐槽我说你写了那么多代码然后并没有怎么样是非常有道理的，那是因为我只关注着做为流水工人的价值要求，看不到生态链最顶端的价值。

我们仔细想想那些团队领导，精英领袖哪一个不是为着更广大的价值所负责，比如项目经理只需要关心自身项目的商业价值，而公司 CEO 则关心公司范畴内所有业务的总体商业价值。所以关注的价值越大且职位也就越高。这些高层领导者们把控着整体的价值链条，及时纠正底层分工组织的价值目标与整体价值目标出现偏差的问题。

从价值出发 – 找寻学习与工作的新思路

迷茫能引发思考，架构则塑造了视野，而价值则是我们之所以存活，之所以工作的逻辑起点。基于这样一种价值思维，对我们的学习和工作又可以有哪些改启示呢?

明确自身的业务相关主体：找出你工作的协作关系网内的业务方和客户方，这样你就可以从客户方中找到离你最近的业务价值点，从你的业务方中挖掘更多的资源。甚至你可以按这个思路顺着网络向上或向下挖掘价值链条，整合更多的上下游资源以实现更大的价值。

向前一步，为更大的价值负责：不要因为自己是开发人员就不去关注软件运维，不要因为只是测试就不关注软件开发，因为你关注的越多你越能看清全局的价值目

标。如果只关注一亩三分地，那么注定这辈子只能困守在这一亩三分地里，成为一名流水线上焦虑至死的码农。试着转变思维，从架构师的角度思考价值问题，看看能否将技术贯穿到业务、到用户、到最终的价值去。之前我的朋友说过要把产品经理踢到运营位置去，把程序员踢到产品经理位置去，这样才是正确做事方式。这句话也是类似的意思，向前一步才能懂得怎么做的更好。

像架构师一样思考，用价值找寻重心：人的迷茫是因为找不到重心，而价值的意义在于引导我们思考做哪些事情才能实现价值，先做哪些事情会比后做哪些事情更能创造收益。像架构师那样全局性思考，把遇到问题进行拆分，把学习到的事物串联起来，努力构成完整的价值链条。

学会连接，构建体系：前几天看到一篇文章对今日头条的产品形态极尽批判之词，指责它的智能算法将人类封死在自己的喜好之中，将人类社会进一步碎片化。这似乎很有道理，有趣的是互联网将我们连接至广袤的世界，却也把我们封闭在独属于自己的小世界里。依旧是我的那位朋友，他说他的最大价值在于连接，将不同的人连接在一起，有趣的事情可能就会即将发生。

或许算法的天性就是顺从与迎合，但人最终想理解这个世界还是需要依靠自身的行动与不同人之间建立联系，这也是一种摆脱流水线限制的有效方式。另外，我们自身也是某种事物连接的产物，比如架构师，他是业务、技术、管理连接在一起的一种产物。所以我们应当树立自身的知识体系以吸收融合新知识，将孤立的概念连接起来，形成自身的价值链条。比如这篇文章将我从事技术开发经验、与对架构的理解以及自身过往经历结合起来，这也是一种内在的体系梳理。

作者简介：空融，网名“D 调的暖冬”。现就职蚂蚁金服，从事支付宝身份认证相关领域的技术开发。

在技术的世界里，你又有哪些感悟和体会呢？欢迎在留言区与我们一起交流互动。

如何成为一名顶尖的阿里架构师?

无叶

在技术圈，架构师一方面是已经被说烂的职务，另一方面也是让人困扰的职位，行业发展到现在似乎人人都是架构师，各种架构图绚丽多彩漫天飞舞，同时永远有人在抱怨架构太烂、坑太多。

那么到底什么是架构师？如果有一天把你丢到架构师的位置上你会怎么做？ 做什么呢？今天，阿里国际技术事业部的无叶，与大家坐一起，聊一聊。

一、两种架构师

工作五年以上的童鞋，或多或少都会有这样的经历：在小团队或者项目中承担非明确的架构师职责，我们做项目或者产品的关键设计和实施；负责产品基础设施；引入新的理念，框架；解决团队中的复杂问题；在团队成员中享有较高的声誉；被老板

认为是团队的关键人物。

如果有一天我们决定（或者其他原因）去做一个专职架构师，那么这两者会有什么区别呢？是否只是之前的方式的延续就足够？

我把第一种状态称之为“兼职架构师”，因为处于这种状态下的同学大部分的时候担当开发人员、PM 的角色，只有在小部分时间承担了架构师的部分角色。做的绝大部分事情是自己可控的，自己做架构自己做实施或者在小团队中推行。而后一种“专职架构师”则面临的是：他们不负责具体的业务系统，而又对所有的系统负责，他们也很少直接负责项目，但是职责却要求他们必须对项目要有提前把控，他们面对的是更大的团队，更大的问题域。

当然每一个人对是否应该存在“专职架构师或团队”都有自己的想法，从阿里的历史来看单独的架构团队也是分分合合。在这里不去探讨，我们关心的是如果有，可以怎么做。

二、专职架构师的职责

首先要弄清楚的是专职架构师的职责到底是什么?

微软对架构师有一个分类：企业架构师 EA(Enterprise Architect)、基础结构架构师 IA(Infrastructure Architect)、特定技术架构 TSA(Technology-Specific Architect) 和解决方案架构师 SA (Solution Architect)。这个分类是按照架构师专注的领域不同而划分。

在阿里除了 EA 之外的领域大家可能会同时涉及到，只是不同的时期偏重点不一样。比如前面说的“兼职架构师”可能偏重 SA？专职架构师偏向 IA+TSA。另外一个角度专职架构师更多考虑问题域和相应的系统架构，而“兼职架构师”更多的是产品的系统架构，具体来说我认为专职架构师重要的职责如下：

职责一：全局的技术规划

架构师第一个最重要的职责是技术规划，架构师最重要的产出是架构，架构就是蓝图，就是阿里常说的一张图。画蓝图就是做“全局的技术规划”，这张图上有什么？ 没有什么？什么时候有？什么时候没有？当你尝试去画图的时候一连串的问题拷打着你。对于一个架构师的心力和体力都是很大的考验。只有这张图非常清晰明确才能指引整个团队在同一个时间向同一个方向前进。

为了这张图你必须和业务紧密沟通，你必须有对应的技术深度和广度，在选型上有自己的方法论，敢于做出判断和决策。

另外一个重点是全局。全局我的理解是全面 + 格局，全面就是你的技术规划包含各个方面的，在所有的领域都有明确的指引，所以这张图本质是一系列的图的集合；格局上不要只关注短期利益，更多关注长期利益。不止关注团队利益，更多从公司角度出发，只有这样长期才能为团队带来更多的成长。

职责二：统一的方法 & 规范 & 机制

架构师第二个重要的职责，我们不仅仅要提供蓝图，还要提供配套的方法论 & 规范 & 机制来保障有序进行。蓝图确保整个团队在同一个时间向同一个方向前进。规范确保前进是有序的。为了有序，你必须拆解你的图，纵向、横向、功能内聚等等纬

度拆解到权责清晰对等。这是一项相对复杂且繁琐的过程。

职责三：完备的基础构建

除了蓝图确保整个团队在同一个时间向同一个方向前进、规范确保前进的有序的、我们还需要提供强大的武器库，基础构建的完备程度决定你的团队装备是小米 + 步枪，还是飞机 + 大炮。完备的基础构建是否全部作为实际架构的职责，可以因情况而定，比如是否有实体的架构组。但是架构对此应当负责。

职责四：落地的规划才是架构

如果规划不能落地就是传说中的 PPT 架构师，我甚至觉得这是对专职架构师最大的挑战，前面的几个职责更加偏向硬实力，而这一个更多的是软实力的体现。专职的架构师如果不去关注落地的话慢慢就会架空，变成 PPT 架构师，那差不多就 game over 了。

三、专职架构师的权利

正如前面说到对架构师最大的挑战是落地层面，实际上“完备的基础构建”已经涉及到落地层面的事情，但是和完备的基础构建不同的是整体架构的落地涉及到方方面面，面临是更多影响因素：和业务的关系、组织结构、权责定义等等。

所以有人从“架构师的权利和职责”的角度出发推论谁合适做架构师。得出的结论是一个组织的领导者。因为只有他才能调动、协调组织。也有人认为架构师既不能完全负责技术团队，也不能完全游离在技术团队之外。因为负责容易屁股决定脑袋，游离就只能靠个人声望值吃饭了。

如正架构分类中 EA 的存在，很多领导者也确实身体力行的践行架构师的职责，然而精力终有限。实际上更多是平衡的过程。当然最高境界是影响力。

四、专职架构师的考核

针对前面的职责怎么考核？或者怎么设定自己目标？虽然说在不同的团队阶段，不同外在环境，不同的权责情况下不一样，但是在结果导向的背景下落地肯定是架构师重要的考核指标之一。

考核一：全局的技术规划

相比其他几项这一项是最重要又最难评价的，技术规划的好坏、全面性、前瞻性都是定性的描述，如何指引我们做出一个理性的评价呢？回归到本质上“技术规划”只是一个指路灯，团队中每一个人能不能看到“指路灯”就到达目的地是指路灯价值的体现。所以无论是唯价值论还是唯口碑论衡量的其实是同一个东西。

考核二：统一的方法 & 规范 & 机制

这一项的考核就相对容易多了，无论是业界还是每一个架构师本身都有自己的一套方法，所以只需关注这些东西对应的产出。

考核三：完备的基础构建

我认为在大公司，大部分重量级的基础构建已经是非常完备，对于架构师来说更难的不是从 0 到 1，而是克制、边界和从 1 到 2 的过程。对于架构师也好、技术团队

也好“从 0 到 1”总是充满了吸引力，加上技术人的特征，大公司技术史上永远不缺少重复的轮子，创建这些轮子成就了一代一代的同学，拆除这些轮子再成就了一代的同学，所以克制尤为重要；有了克制跨团队的合作就尤为重要，对应的有两个点一是清晰边界，二是共建。

考核四：落地的规划才是架构

虽然说落地是非常不控的事情，但是考核却很容易的：做到就是做到、没有就是没有、质量好就是质量好，标准非常清晰。过程中只需要紧跟拆解的事情结合实际的组织和业务情况做出决策。

五、实施的一些想法

对现阶段团队的情况来说，我认为第一是建立“架构语言”，有了语言才有沟通协作的基础，所谓的“架构语言”并不是什么新的东西，而是产品的业务架构，用例和领域模型；研发的应用架构，组件和时序图；运维的部署架构等等。

第二是建立“认同体”，无论是通过技术能力、知识传递、领域组织等各种方式逐渐形成“认同体”，且在其中形成架构体系对应的人员体系。

第三永远做服务者，架构师对应的客户是团队的每一个成员，必须始终保持客户第一的心态。架构师存在的目的是成就研发团队每一个同学，我们提供必要的平台、服务和空间，然后彼此成就。

最后借用一句话：从无到有的是架构；从表到里的是抽象；从粗到细的是设计。大家对架构师有哪些看法，也欢迎在留言区留言，我们一起交流讨论。

PS：阿里国际技术事业部 – 无线技术部求贤若渴。欢迎苹果、安卓、服务端童鞋加入我们。

浙大博士来阿里，为了打造地球最强的安全策略战队！

阿里味儿

甲第，80 后，标准的山东汉子，浙大博士。

目前在阿里巴巴安全部带着一百多号人的一个团队。

听说他有个标签“资深吵架专家”，背后有什么故事呢?

01，学霸的青春也迷茫

从浙大数据挖掘专业毕业后，甲第“阴差阳错”跑了一年的 sales，这段日子对于一个技术男来说，简直是“生不如死”。

一年后，加入阿里，进入国际站风控团队，成为一名数据产品经理，算是如鱼得水。

可好景不长，之后几个团队合并，随之而来的各种想法、理念的剧烈碰撞让甲第感到很迷茫。

“那段时间很痛苦，很纠结，很郁闷。”甲第连续用了三个形容词。“明明觉得自己做的事儿很有价值，但是其他团队不认可，于是拼命去吵、去争论。”在这样吵吵闹闹的过程中，他被人贴上了“资深吵架专家”的标签。

如此迷茫痛苦，人的本能反应大概就是逃。

甲第也不例外，再这么下去，不就是浪费时间、浪费生命吗？他想到了转岗。大概有个十天半个月的时间，他每天凌晨一两点在小区楼下绕圈圈，走？不走？走？不走？

故事到这里，应该出现个转折人物了。

没错，甲第特别感谢当时的主管说了一番话，让他醍醐灌顶：“我们每个人最后都会选择离开，离开团队或者公司，但换做我要离开的话，**不应该以这种低着头的姿态离开，而应该抬着头离开**。”

任何一个铁骨铮铮的汉子听到这样的话，想必都会血槽加满，重新加入战斗。

02，唯有热爱才能抵挡危机

调到安全部的业务风险防控团队后，甲第体重从原先的 84 公斤掉到了 72 公斤。橙子忙问他瘦身秘诀是什么，他一脸苦笑说：“哪用得着什么减肥啊，我这是压力大，晚上睡不着。”

在阿里，你不行，总有别人比你行，每天都有危机感。

要关注无时无刻不存在的业务风险，同时你还得想未来，想未来的策略体系、攻防模式，不能光靠加人来解决层出不穷的新问题；还有行业、生态、监管、标准、顶层设计的问题，以及组织升级、梯队管理、人才培养……

甲第说，当我们老的时候，抱着孙子在摇椅上回忆当年时，肯定不会记得哪年在阿里晋升了，哪年发股票了，但一定会记得，哪件事儿是爷爷当年在阿里干成的。

“**只有从心里热爱，睡一觉起来还是会愿意满怀希望地去拼。**要是你不热爱的话，趁早离开，别耽误公司和你自己的时间。我觉得这么好的一个平台，就是期望我们给行业、给社会带去一点变化。”

“那现在还会想着离开这里，去别的团队，或者离开阿里吗?”

“不，我觉得自己还是非常适合安全工作的。今天，它还没有完成伟大变革和使命，我肯定不会离开……”嘿，简直是停不下来的节奏?

甲第和团队合影

03，人通透了才不纠结

甲第说，对于刚入行的年轻人来说，别怕纠结和痛苦，这是好事儿。**一个人的成长高度和速度是和痛苦压力成正比的**。

一定要找到自己热爱的工作，不适合或不喜欢就换，别死扛。因为热爱，所以不会去计较得失，反而能专注地坚持做下去，比如 1 万小时学习理论，笃定地把它当做事业来干。

勇敢接受时间和事情的磨练，这会让人变得通透。

不计较不比较，不纠结不拧巴。

一旦通透了，做什么事儿就酣畅淋漓了。

最后送上阿里妹对学霸甲第童鞋的一手采访！

阿里妹：能否和大家介绍下，你与团队在业务安全这块的突破与创新?

甲第：我们的 team 比较新，而且一直处于打仗的状态。在业务安全历史上，主要有下面几个：

1、第一次在业务安全攻防中设立技术岗，通过数据技术解决复杂多变的攻防问题；

2、第一次基于“人”防控而不是仅仅基于账号；

3、第一次从端 – 网络 – 应用 – 业务全链路打通搭建攻防策略体系，而不是单点防御；

4、第一次建设精品业务安全能力，使得安全成为业务的核心竞争力，而不仅仅是防和控，如社区反垃圾、营销反作弊和反黄牛能力。

阿里妹：你与团队的下一个目标是什么?

甲第：成为这个星球上最强的业务安全策略 team，没有之一！

1、让安全成为业务的核心竞争力，安全是业务的安全，业务是安全的业务，能够帮助业务而不仅仅是管控；

2、开放业务安全能力，改变互联网业务安全现状，净化网络空间，通过策略、产品、技术和算法的合力，赋能互联网的中小企业和用户，远离风险。

阿里妹：作为一位名副其实的学霸，你认为阿里最吸引人的地方是什么?

甲第：太多了，无法一一举例。我就说几点吧。

人：阿里聚集了全球互联网行业最精英的人才，每天和他们在一起工作，能够不断成长；

平台：平台太大了，当站在这么大的舞台上，你不好好做点事感觉特别对不起自己和这个平台；

机会：不断做事的机会，不断犯错但还能重来的机会，不断成事的机会；

味道：阿里不但教会我如何成事，如何在逆境中不放弃，如何思辩如何创新，如何欣赏他人，更教会了我如何做人，一个纯粹的人。

阿里妹：你们部门还缺人吗？期待什么样的人加入？

甲第：

缺！缺！缺！

数据！数据！数据！

重要的事说三遍！！！

所有统计学、数据挖掘等数据专业人才，我们统统欢迎！来吧，一起战一场，打造这个星球最强的策略团队：D

有兴趣的童鞋可以直接发简历至：jiaxin.jx@alibaba-inc.com

本文部分内容来源：阿里味儿

从清华到阿里，他只用 6 年时间，影响了数亿用户

靖世

“阿里技术直播”，是专为技术人量身制作的视频直播节目，旨在分享行业前沿趋势、技术干货和技术人生。今天为大家送上阿里资深算法专家靖世的精彩直播内容。

阿里资深算法专家靖世直播实录

大家好，我名字叫盖坤，在阿里花名叫靖世。之前在清华大学读的本科跟博士，专业是机器学习跟人工智能。毕业之后一直在阿里巴巴做广告算法，现在在阿里妈妈负责竞争展示广告技术，做的工作包括广告算法，广告算法里面包括匹配算法、预估模型、排序算法，也包括广告工程部分。还有其它相关跟人工智能相关的部分，包括机器学习平台，包括计算机视觉里面有图象识别等等，也包括 NLP 的一些技术。

今天我的分享有三个部分的内容，希望对大家有用。

1. 从找工作的时候选择阿里巴巴，到工作六年来一路走过来的历程跟体会。

2. 做 AI 背后的想法，在 AI 上取得的一些成果和背后的思考。

3. 最后会建议一下大家在面试阿里之前应该准备哪些内容。

缘起阿里：研究与应用有了完美交叉点

先说一下跟阿里巴巴的一个开始。其实我在学生的时候，是做机器学习和计算机视觉，也发了一些国际的会议跟期刊的论文。在毕业的时候，我有一个，或者说在学生的时候我有一个愿望就是希望自己做的东西不但是以论文的形式分享给别人，而且希望能够对于这个世界产生更大的作用。所以当时就希望能够去找一个研究跟应用结合比较好的一个地方。刚开始的目标主要是集中在一些外企的研究院上，当时也聊了一些研究院。

跟阿里巴巴缘分的开始也是一个机缘巧合，之前清华同学经常上的论坛叫水木社区，当时叫水木清华，我在那里的 AI 版上发现有一个人发了帖子说阿里巴巴要做大规模的机器学习系统，这里面有很多挑战。我当时也是晚上很晚的时候回了一封私信，当时也没多想，而且在学生的时候，说实话有晚上玩游戏的习惯，然后回了之后就玩游戏到两三点，睡得比较晚。结果第二天一早八九点就被阿里巴巴的 HR 叫醒去面试，这是跟阿里的缘分的开始。

拿到阿里的 Offer 之后，其实也是思考了很多，纠结了很久，最后选择阿里巴巴，是因为觉得阿里巴巴其实在中国零售的业务上，也是蓬勃发展，业务前景非常好。第二个做机器学习可能最看中的就是数据，阿里巴巴有一个围绕零售的消费者的，从逛到买，再到买之后的后续行为的一个完整的类目数据。

这个类目数据上用人工智能的方法可以做很多事情，有很多可能性，所以会觉得人工智能在阿里巴巴会特别有空间。还有最后一个点，当时也是跟阿里的同学聊了很多，会觉得阿里里面不管团队还是各方面，其实也都说对研究会比较看中，也会在应用之余鼓励做研究的事情，包括鼓励大家去出一些研究的成果。所以当时会觉得就是：

1. 业务的应用前景非常的广阔；

2. 研究跟应用可以做一些结合。

当时基于这两个考虑选择了阿里巴巴。

我分享一下在阿里巴巴一路走过来的经历，包括其中涉及到一些做的事情，希望给大家一些参考，最后再做个阶段性的总结，想做得更成功的话，应该具备哪些特质。

阿里 6 年经历：从沉寂半年到连续提升 2 个 10%、每天影响上亿用户

我进阿里巴巴之后，其实前半年的感觉就是自己什么都不会，然后其实进了实际的业务，实际的数据，虽然之前学了很多的机器学习相关的知识，然后发现这些知识可能跟业务跟实际的数据还不能很好的结合在一起。所以前半年其实是一个沉入到业务，沉入到数据，而且是一个相对来讲比较平淡跟寂寞的时间。但后来我会发现，其实必须得沉入到实际的业务，实际的数据里面才有可能做出一些不一样的东西，这个过程是必须经历的，这是很多在实际工作人的一个体会。

但是这半年说实话，虽然可能没有做出惊天动地的事情，但是这里面有一件事情，其实一直在思考。其实在我加入阿里巴巴的时候，当时是机器学习在广告，在 CTR 预估系统里面开始大规模的使用，然后国内的各个公司也开始建立团队研发这一块。这一块也可以认为是这几年国内的主流业界的公司在机器学习去投入大量的资

源的一个开始。

因为大家可能也都知道，整个互联网行业里面，现在收入最大的两个板块：一个是广告，一个是游戏。广告应该是比游戏还要靠前，而且不是所有的公司，包括大公司都做游戏的，大部分公司的支柱收入其实是广告。对于各个公司这么重要的一个业务，背后其实点击率预估是对广告主，浏览者，平台收入，做好了的话是一个三赢的事情，所以各个公司都投入非常多的资源来做这一块。

所以这一块是机器学习可能第一次非常大规模，而且在实际的大业务中起非常核心作用的一个契机，这也是近年来的一个起点。机器学习在后面，在更多的业务里面去发挥更大的作用，这是后面的事了。

在点击率预估里面，当时有一个经典的做法就是叫大规模的特征加上一个简单的线性模型、逻辑回归的一个做法，这个做法简单说一下怎么理解呢？就是其实那时候大规模特征是一种叫 ID 特征的一种形式。比如，我们假设说现在中国有 13 亿人，我们有 13 亿用户。我们一定要用一个向量，有 Sample 有 Label 的概念，用一个向量，用样本来表示它。这个样本里面特征怎么办呢？我有 13 亿用户，我就用 13 亿的系数向量来表示这个用户，然后 13 亿维，这个样本对应哪个用户就是在哪一个维度上标记为 1，其它维度都为 1，是一个非常大规模的一个稀疏的一个表示。

其实用户可以这么表示，商品可以这么表示，所有的信息基本上都可以 One hot 的编码来表示。所以，其实连续的一些统计值或者连续的一些值也可以做一些离散化，把它变成哪个区段的，继续用 One hot 来表示，我们会把大量的信息用 One hot 编码或者用 ID 特征表示方法来把它给变成规模非常庞大的特征，特征维度也特别大，这可能是没接触过工业界实际的，比如说 CTR 预估系统的同学可能之前不太知道的一个概念，就是为什么工业界需要这么大量的特征。

这样的特征用简单的逻辑回归来做的话，其中有一个问题，当然这里面挑战也很大，有两个挑战：

1. 样本量特别大，特征维度特别大。

样本量特别大什么意思？比如说点击率预估，那我们的 Sample，就是用户的历史行为，如果用户看了没点，这就是 Label 是 0，就是负样本；如果看了又点了，这

个样本就是一个正样本。其实你每天看大量的东西，大量的用户在网站上浏览，所以我们有非常多的样本。第二个特征维度特别大。所以对逻辑回归来讲的话，一个挑战就是如何能支撑这么大的样本和这么高的维度，这是第一个挑战。然后这里面会涉及到大量的机器学习的并行算法相关的东西。

2. 正则化，怎么理解?

就是逻辑回归一般来讲会跟一个 L1 范数一起来使用，它其实背后有两个目的，第一个目的就是这么大量的特征的化很容易发生过拟合，所以我要用一种方法，在机器学习里面是非常经典的方法，正则化的方法去来抑制过拟合，让这个模型能够在未来的使用的场合里面表现比较好。过拟合的意思就是我只记住了那些已见过的东西，但是对未来新的东西其实预测能力并不好。

第二个其实就是这个特征库特别大的时候，我们在离线做训练的时候也许能够承受这么大规模的特征，但是我们在在线预测的时候，我们希望性能尽量的好。所以我们希望训练的时候做一些特征选择，这么多的特征是不是全需要? 很有可能我们其实只需要其中的一小部分。能不能让模型在训练的时候自动做这件事，这就是正则化的一个方法。那加上正则化的方法也是会给整个优化问题带来一些挑战，这是逻辑回归的两个挑战跟难点。

除了这两个挑战和难点解决了很大一方面的问题之外，逻辑回归还有一个问题我倒是一直在思考，这样的特征体系用线性模型够不够，这个问题其实当前来讲在深度学习这么如火如荼的今天来讲大家都不会产生疑问，非线性一定会做得更好，大家可能都这么想。但当时其实还是有挺大争议，很多人会有一个观念，包括一些论文里其实也把这个结论会明确的写出来，就是特征维度比较高的时候，线性模型就够了。

当时在几年前，谷歌还算是在某些方面是国内公司的技术领导者，某些方面的技术大家都是向谷歌看齐。当时谷歌也是用大规模的特征加上一个线性模型，所以你要做非线性模型很多人就会挑战为什么非线性模型有用，线性模型是不是就够了。现在这个观念好像并不是那么的统一，大家都觉得深度学习都用上了，那非线性更强了，非线性一定有用。

但当时其实要打破这个思维定势，其实相当于在挑战公司内外、业界、学术界的权威概念。当时我一直在想够不够，其实我们知道，其中有一个做法很说明问题，就是在使用逻辑回归的时候，我们的特征即原始的特征，就是 One hot 编码或者 ID 特征的维度很大，但我还是需要做特征加工、特征工程。比如说我们的目标的目标跟两个相关，可能要对这两种 ID 做一个笛卡尔积，做笛卡尔积是特征非常爆炸的过程。

比如说一亿维的用户特征，一亿维的宝贝特征，我们要做他们的关系的话，就是笛卡尔积会的话就是 1 亿 ×1 亿种可能，一下子就爆到 1 亿亿的维度了。我们在实际工作里面很多公司做过这样的事情，我们用算法工程师来做特征的组合、特征的加工，然后尤其是笛卡尔积等等的工作。这里面非常的繁杂，而且两两特征可以组合，三种特征是不是可以组合，有没有其它的加工方式。所以这里面有大量的繁杂的工作。

这其实是一个很好的例子，就是线性模型并不够，如果线性模型够的话为什么还要做特征加工的工作，用人工来补足呢，其实这是当时业界的一个经典做法，一个经典的思维定势，内部的自我思维的一个矛盾。然后这底下其实我在想的就是如何能够省去这些繁杂的人工处理的工程，而且人工处理对特征加工一定是有限的，我们其实能不能用现在做 AI，做智能的方法，去抽取更精确的信息，做更好的预测，这其实可能是一个做 AI 的人，内心在真正追求的东西。

所以我们其实思考的是能不能够去做一个更强力的人工智能的模型，准确来说是机器学习的模型来代替原来这种重工程的简单的线性模型。两个目的：一个省去人工繁杂的加工动作；第二个就是能够去达到更好的效果。另外一个目的，如果我们做智能的工作可以达到一个端到端的学习，像现在深度学习一样，可以让很多事情更自动化起来。

这个事情其实前半年一直在我脑海里去思考，虽然前半年没做出什么特别惊天动地的事情，然后在日常的项目投入跟业务理解里面，我一直在想这个问题的答案。后面找到一种方法。就是在大规模特征，而且大规模的样本上做非线性的学习，而且还能拿到效果。

下面简单介绍一下后面采取一种方法就是叫分片线性的方法，准确来说是整个高维空间里面，如果维度特别高的话，把空间分成很多不同的区域，每个区域里面有一个自己独立的一个线性模型。这样的话整个空间变成分段线性的模型，如果是二维比较好理解，原来是一条线，现在变成分段折线，折线足够多可以去二维上逼近非常复杂的曲线，可以去逼近任意复杂的一个函数。

其实背后这种思想还是蛮简单直接的，我们会有两个挑战：

1. 我们如何能够把空间划分跟最后的每个划分里面分段线性，一起用机器学习的方法去一起把这东西全部学到，然后通过数据的方法学到；

2. 我们去面对的数据规模特别大，特征规模也特别大，是不是能发展出一个非常有效的方法做这件事。有一个叫混合逻辑回归的方法，在阿里妈妈内部真正使用的，用分片使用的 Softmax 函数，最后每个区域内做分类线性的逻辑回归这样的一个组合。实际上当时我应该是研发了差不多有十余种不同的模型，有各种各样的分片的方法，分段的方法，包括有一起并行去学的，也有像 GBBT 之类的是一个片段一个片段去学，一个片段一个片段续贯学的很多不同的算法。

这背后其实遇见了大量的问题，比如说模型到底 Work 不 Work，模型如果 Work 的话我们复杂的函数一般对应一个优化问题，优化问题不是能寻找一个很好的解，包括其实这个模型也会限制它的算法，这个算法能不能很快的收敛。这些其实都遇见大量的问题。最后我们其实现在留下的，看起来虽然非常直接简单，也是当时试

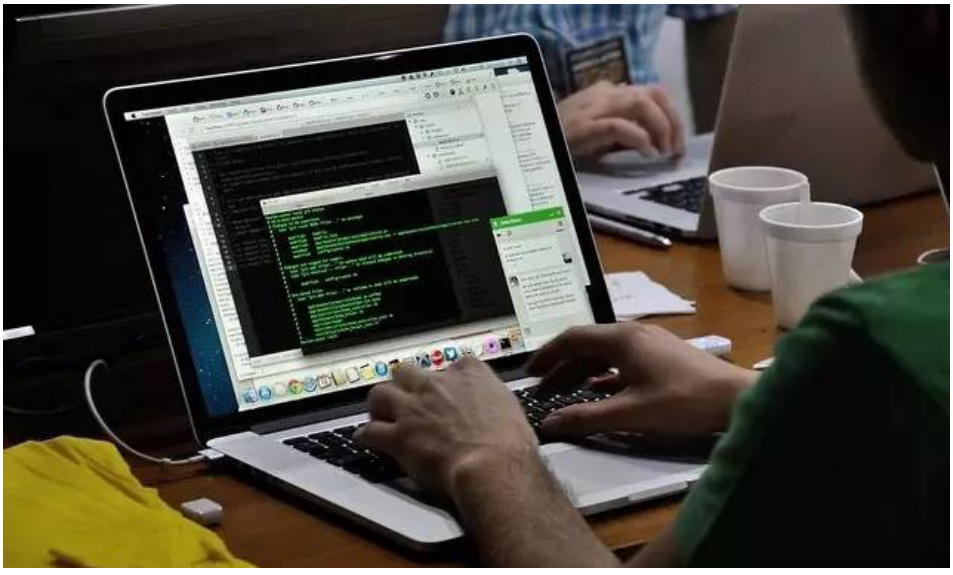
验了大量的方法之后，最后留下了几种可行的方法中的一种。背后有非常多的失败的案例，这个可能是其它人不太知道的背后的故事。

做完这件事之后，其实基本上解决了之前说的一些问题，一个是非线性学习能够让模型自动的智能化的在数据里面去挖掘人挖掘不出来的知识，节省人工的劳力，来真正形成现在说的一个智能化的方法，用一个非常强力的人工智能的方法去挖掘出一个很好的效果。

这件事做完之后，第一期在业务上的效果应该大概在入职已经是大半年了，当时在上线的业务线上，广告的收入，包括点击率提升 10% 以上。而且分两期，每期都是 10% 以上。当时具体的数字比这还高不少，但是具体数字现在记得不准确了。当时就觉得第一次觉得特别激动，终于就是把自己的知识去应用到实际场所里面，不仅提升了用户的体验，平台收入也有一个大幅的提升。自己做的东西，每天能影响上亿用户。这感觉真的是用自己的专业知识做到了一件能够更影响这个社会，更影响用户的事情，非常有成就感。这是加入阿里的想分享的第一个事。

深度学习现状与规律总结

下面我大概说一下深度学习。深度学习怎么看？我分享一下我的观点。



深度学习现在特别火。这里面其实我们去抽取几个规律，深度学习有这样的性质：

1. 深度学习其实把原来模型跟算法耦合的一件事结耦了，深度学习背后的比如说叫梯度传播算法，基本上是 SGD 的方法，SGD 基础上会有一些加上动量，自适应动量之类的一些算法。然后这些算法其实变成了一些标准化的算法，你架什么模型都是在这些算法里面去选择，你可以很大概率上可以不用去特别深入这些算法，或者创建一套算法。这样结耦的模式跟原来做机器学习的模型其实是概念完全不一样。

如果真的对机器学习特别了解的，或者学过相关专业的同学会知道，SVM 其实从模型到算法是整个体系化的方法，逻辑回归也是一体化的方法，GBDT 也是一个一体化的方法。当然它的整体性会非常好，但是它做这件事的代价特别重，必须是专业能力非常强的人，从模型理解、数据理解到整个算法设计都需要非常强的专业知识，才能够去做这件事。这个在生产力上来讲是被束缚了。深度学习其实是把模型跟算法结耦，算法标准化，模型可以任意的改变跟搭建，极大释放生产力，能够让更多的同学参与进来，而且去尝试更多更复杂不同的模型。

2. 深度学习这个方法有点像搭积木，我其实有很多模块，其实各个公司，各个学术机构还在持续不断得根据实际的问题跟数据去创造新的模块。

很多场合下，其实大部分的工作是把这些模块自由的根据自己的业务问题组合起来。跟原来不一样，原来是没有模块，你要建个楼，整个楼都是自己建，现在给你很多墙，很多楼板，你用这个楼板搭起来就可以了。这个其实在我们实际的基础模块称之为组件化，它可以组件化、标准化的把原来想像不到的复杂体系更简单搭建起来。

这两个特点，一个结耦，一个模型本身的组件化，在深度学习框架下能够让我们的工程师搭建原来完全无法想像的复杂模型，解决原来解决不了，或者是效果达不到我们预期的一些问题。这是深度学习背后的一些机会，我认为是对生产力的一个极大的促进点。

所以在这个点下，在深度学习下，阿里巴巴也展开了很多的研究跟应用。这里面我说一下现在我们在深度学习的一个思路。深度学习说起来很简单，就是 Deep，非常深的一个神经网络。那这里面，会有一个问题，就是深层次，就是全部，或者说我

们只要深就够了吗？那这里面，我再举一个思考，就是业界比如说图象识别，图象识别其实它有两个要素。第一个要素是一些现在大家比较公认的，或者是一些基础网络结构，像单层的 CNN，像 Pooling 等等的结构，尤其是以 CNN，就是卷积神经网络为代表，卷积神经网络在图象里面基本上占了主导型的应用。大家都是在这种结构的基础上才搭建深层的网络。

如果深层网络够了，为什么不用全链接搭建这样的一个深层网络呢？这是第一个例子。后面其实还有其它例子，比如在时序，在 NLP 等等里面会有很多这种结构，像时序上的 RNN，或者是 LSTM 这样一个稍微复杂一些，可能在很多领域更有效果的一些模型。

这里面有两个要素：第一个要素是跟数据比较匹配的一个网络结构。为什么说跟数据匹配呢？以 CNN 为例子，CNN 是局部的卷积窗，这个局部卷积窗代表了一个参数，这个卷积窗在移动的时候参数是共享的。它其实代表了图象的性质叫平行不变性，就是一个人脸或者狗的图象，在图象的左上角，它跑到右下角还是一个狗，它就叫平行不变性，就是物体识别不会因为这一块区域整体移动之后就会发生显著的变化。另一个物体识别也是局部的，这一个窗口位置如果是个脸全包括进去了，那这个窗口的信息就够了，这是一个领域性。其实像在图像领域或者已经成功的深度学习比较成功的领域，网络结构跟它的问题特性以及数据特性是非常匹配的。

我们现在其实在关注的是在互联网的，如何基于互联网的用户数据，比如刚才这种 ID 号编码的大规模数据上，我们如何能够去找到一种网络结构，能够去适应这样的数据特性，然后我们再用深层次的方法来加强这个结构的泛化能力或者拟合能力来达到更好的效果，这是我们在深度学习上的一个思考以及现在做的一件事。

这里面有什么特性？举两个例子：

1. 我们的推荐，背后有一个很重要的数据，就是用户的行为，用户历史上浏览过什么，购买过什么，或者你在哪些页面上停留的时间更长，这个对我们分析你未来对什么更感兴趣，是非常有帮助的。第一个概念就是结构化，这背后有一个非常复杂的结构化数据。比如说用户在淘宝上的行为，你点了一些宝贝，那除了点这个宝贝 ID 本身之外，宝贝是什么之外，其实这个宝贝后面还有相关的数据，比如宝贝有图象，

你也看到这个图象，会跟图象发生了一些思考跟交互，你才决定点不点，买不买，跟商家交互不交互。

再比如说宝贝有标题，再比如你点宝贝的，历史上点宝贝的时间也有关系。再比如说宝贝详情页有很多介绍，你还会看它的评论。这里面大家发现它都跟某一个宝贝相关，所以这里是一个高度的结构化的数据，不同源的数据其实是内部有很强的关联性，异构的数据是关联在一起的。如果我们去设计一个比较好的深度学习网络结构，去用好这些异构化的数据，这是一个很大的一个挑战跟问题。

2. 另一个是时序，用户在我们的电商环境下，其实他会可能会浏览很多的宝贝，他可能看一些手机，同时给女朋友买一个包包，在行为序列上我们能够做什么样的分析以及设计什么样的网络结构能够更好更精准榨取用户真正感兴趣的东西，这对深度学习来讲也是一个挑战。

这两个例子其实就是说明了对一个实际的，比如电商里面的业务问题来讲，你能利用到的东西其实可能比一个学术界给你的数据集要广泛的多，这里面你可以想各种办法，你可以产出更多的创造性来解决这个问题。其实我们前一段时间公开了一个工作叫：用户多兴趣分布模型，我们会在用户的整个序列上做一些网络结构去更合理榨取用户的兴趣信息。我们也对外发表了这个成果，在 arXiv 上已经挂出来，网上应该也有一些讲解，大家对细节感兴趣可以看这个模型。

这是我对深度学习的看法，以及我们在做的一些事情。

技术的三个进阶过程，阿里在哪里?

最后谈一下我对技术的一个理解。我会把做技术这件事分三个阶段：

1. 应用，提升业务效果，更好的解决问题的阶段。这个是大家做的最多的，不管是研究还是实际的业务。就是看下业界大概怎么做的，不同的公司怎么做的，哪些公司更先进，我们是不是能把这些先进的做法吸取借鉴过来，运用在自己的业务上，而且做的更好，这是第一个阶段，这也是研究工作的基础，把所有现在最前沿的方法摸清楚。

2. 技术上做一些创新，我们不管是对技术的理解，还是对问题的理解，更深一

步之后，我们如何能够发挥创造性，提出一些更新更好的解法，把这个问题解决得更好。这个我们称之为技术创新的阶段。

3. 我理解的可能就是更大的一个层面，就是用技术去驱动一些整个因果链跟业务链条的变化的阶段。这是什么意思呢？其实不管解什么问题，我们背后都是有因果链的，因为什么所以什么，用户会怎样，商家会怎样，平台会怎样，等等一系列的因果链。

这个因果链建立的过程是在当时的技术条件下会排除掉很多可能性，因为当初很多技术决定了某些事情结果达不到，所以我们会排除很多可能，在这样的过滤下建立因果链，这是已有的业务。那这里面为什么阿里巴巴对技术会做这么大投入，就是因为技术的变革，其实会导致这里面我们原来排除掉的因果可能在未来成为可能性，整个因果链的思维过程会跟原来不一样，整个业务的形态也会发生不一样，或者催生完全不一样新的业务。这就是为什么各大公司都在技术上加大投入这么重视的原因。

阿里巴巴其实在第一个阶段已经做得非常多，我们的业务体量也非常好。第二个阶段是现在在技术创新上在着力做的一件事情。大家可以看到近几年比如说机器学习跟数据挖掘等相关的领域，这种国际顶级会议的论文其实阿里巴巴也发表了很多，我们目前来讲在第二阶段已经开始做出很多很漂亮的结果。

我们的理想是在第三个阶段布局，我们希望在技术、行业变革里面也贡献自己的一份力量，去催生更多的这种因为因果约束关系的变化造成的整个业务因果链的变化，思维的变化，催生出一些新的业务形态，这是我们布局的事情，这是阿里巴巴目前的阶段。

阿里期待什么样的技术人？

我是阿里技术岗的面试官，也是带团队的技术 Leader，作为这两个角色的话，我怎么看，有哪些特质的人是觉得团队更需要，或者公司更需要的人。其实阿里巴巴有很好的两个总结：有一个总结叫“非凡人平常心做非凡事”。

非凡人什么意思呢？我们希望这个人的特质非常好的，首先比较聪明，对事情有韧性、有坚持、有驱动力，能够去更好的解决问题，然后甚至能够创造性的解决问

题。 非凡人我们会概括几个特质：聪明、皮实、乐观、自省。乐观不能遇到问题就特别伤心，啥都干不了了。聪明的话就是得有一定的专业素养，甚至或者说你得有足够的潜力，即使你不太会的，能够在实际的问题，或者通过自己的学习迅速的提高自己。皮实就是我们做任何一件事都不是一帆风顺的。

举一个例子，阿里的云战略，阿里其实有一个著名的人是王坚博士，王坚博士在做阿里云战略的时候，应该说还是承受很大的压力的。关注业界的人，都知道那时候相信云这件事情的公司并不多。大家非议特别大，王博士那时候肯定承受了很大压。其实真正一路走过来其实阿里巴巴的云现在在国内来讲，应该是阶段性最大的，而且在持续投入更多的资源，希望做得更好。

目前来讲，大家基本上都已经相信了云这件事。这一路走来，其实我觉得我特别钦佩王坚博士的一点就是他在各种压力下，甚至也有很多人可能对王博士本身对云技术的理解之类的也会提出一些挑战。其实王坚博士以他对云的这种坚信，坚持，真的把这件事情做成。不管可能有其他大牛觉得怎么样，但是真正能把这件事做成的，可能除了王博士还真不一定有。因为这个承受的压力非常大，真的是内心的坚持，内心对这件事的坚信才能把一件很困难的事做成。

所以我们需要有这种皮实的精神，遇见困难之后就打退堂鼓，那一件伟大的事一定不是你的，任何伟大的事，如果现在还没被人做出来，一定都是不那么容易的。自省就是说在整个做事的过程里面需要反复的去反思自己以及吸收外界的这种反馈来思考怎么把这件事做的更好以及自己初心对不对，自己做的方向是不是在自己初心上面。所以这是对人的要求。

第二个就是大家进入工作阶段要保持平常心，其实从学术界或者从一个环境切换到一个新环境，面临一些可能非常复杂的数据和业务。这里面可能就需要用一种平常心，用一种坚持力，要耐得住寂寞，能够沉下去，把这些事情摸透，把数据能够认识清楚。这是第一个阶段，如果特别心高气傲，永远觉得很多事好像不愿意去做，可能永远沉不下去，然后也很难真的做出一些特别伟大的事情。

最后就是一个建议，在平常心的同时，要有一颗追求卓越的心。首先要沉下去，沉下去之后，对很多现状可能要去思考怎么才是一个改变它的更好的方式。怎样对业

务，对技术是一个更好的未来。然后一直去想一些有挑战性的问题，有价值的问题，这颗追求卓越的心一定要有。非凡人、平常心、做非凡事，这就是我们怎么看，我们需要什么样的人。以及聪明、乐观、皮实、自省，都是非常好的总结。

如何在面试时更好展现自己?

每年其实也接触过很多找工作的同学，下面会给一些建议。

首先就是你的简历有哪些亮点能够抓住 HR 跟技术主管的眼球，比如学校背景大家肯定会看的，但是更抓眼球的是真正能够在更大范围内去证明自己能力的一些事情。 这里面包括比如说高质量的学术论文，比如说在一些国际的比如数据比赛的名次或者奖项。再比如你是不是有一些代码受到了大家的认可，例如说最经典的就是一些开源项目，参与的方式等等，这些东西会特别抓住 HR 跟面试官的眼球。

如果没有的话也没关系，我们还要记住一些面试技巧。

第一个的话就是准备阶段，准备阶段的话，有一些基础的，比如编程能力，有一些同学后面用高级语言做一些研究的事情可能会把底层的编程能力，比如说本科时候学的一些东西稍微忘掉了，实际工作很多需要自己操刀做的，这个东西还是在面试之前稍微捡一下。举几个例子，第一个基础的编程能力，第二个是最基本的比如说数据结构的知识，比如你面试的时候链表一问都不知道，这可能还是会让面试官很失望的。第三个如果面试算法岗位，除了刚才说的机器学习或者视觉或者运筹学等专业知识之外，一些基本的算法，最好也准备一下。

这里面其实推荐一下《算法导论》之类的业界比较经典的书。哪怕你只花两小时，或者一下午稍微复习一下前一两章基础的概念，这对面试来讲都是非常有好处的。基础能力在研究生、博士生阶段还真是有可能丢的时间比较久，建议大家可能花一两个小时或者一下午温习一下就会好多。

第二个就是专业能力，这是日常的，之前的经历，研究项目，在学生时代做的研究工作，包括实习的工作都可能很说明问题。

最后讲下面试技巧。在面试过程里面，面试过程里面，其实大家都可能会担心自己有些缺点，我觉得这并不是最关键的，最关键的就是你认为自己的亮点在面试过程

里面真正展示给面试官。比如整场面试下来，面试官没有问到你觉得自己最牛最厉害的地方，正常面试完了，可能这方面还是会有问题的。

其实面试官会去看这个人潜力，去从各方面来看这个人未来是不是能够有更好的发展，他会抱着这样一个心态来跟你聊的。所以你也可以试图用一些比较好的语言方式去引导一下面试官，引导到你觉得自己比较亮眼，比较强的地方，跟面试官在这上面开展一些讨论跟聊天，这样对展示自己非常有帮助。不怕有缺点，但怕没有把你的亮点说出来。

我的分享就到这里，谢谢大家。

25 岁 Java 工程师如何转型学习人工智能?

阿里技术

“大牛我要问”栏目推出一段时间后，阿里妹收到不少童鞋的来信，其中以职业发展、技术成长的困惑居多。

今天阿里妹选择了一个颇具有代表性的问题：关于目前大热的 AI 入门学习，希望能帮助有同样问题的童鞋解惑指路。

来信问题：25 岁 Java 工程师如何转型学习 AI？

我是一名 25 岁的 Java 开发工程师。本科学习的专业是信息与计算科学（数学专业），因为对计算机方面感兴趣，之后培训学习了 Java，所以现在从事 Java 开发。目前就是在电商公司开发一些系统。

我对人工智能非常感兴趣，对数学的兴趣也从未减弱。人工智能设计的学习材料很多，像我这样的状况，如果想要转型以后从事这方面的工作，具体应该学习些什么?

阿里技术童鞋“以均”回信：

首先，我想聊聊为何深度学习最近这么火。

外行所见的是 2016 年 AlphaGo 4 比 1 战胜李世石，掀起了一波 AI 热潮，DeepMind 背后所用的深度学习一时间火得不得了。其实在内行看来，AlphaGo 对阵李世石的结果是毫无悬念的，真正的突破在几年前就发生了。

2012 年，Gefferey Hinton 的学生 Alex 使用一个特别构造的深度神经网络（后来就叫 AlexNet），在图像识别的专业比赛 ImageNet 中，得到了远超之前最好成绩的结果，那个时候，整个人工智能领域就已经明白，深度学习的革命已经到来了。

果然，之后深度学习在包括语音识别，图像理解，机器翻译等传统的人工智能领域都超越了原先各自领域效果最好的方法。从 2015 年起，工业界内一些嗅觉灵敏的人士也意识到，一场革命或已到来。

关于基本概念的学习

机器学习与深度学习

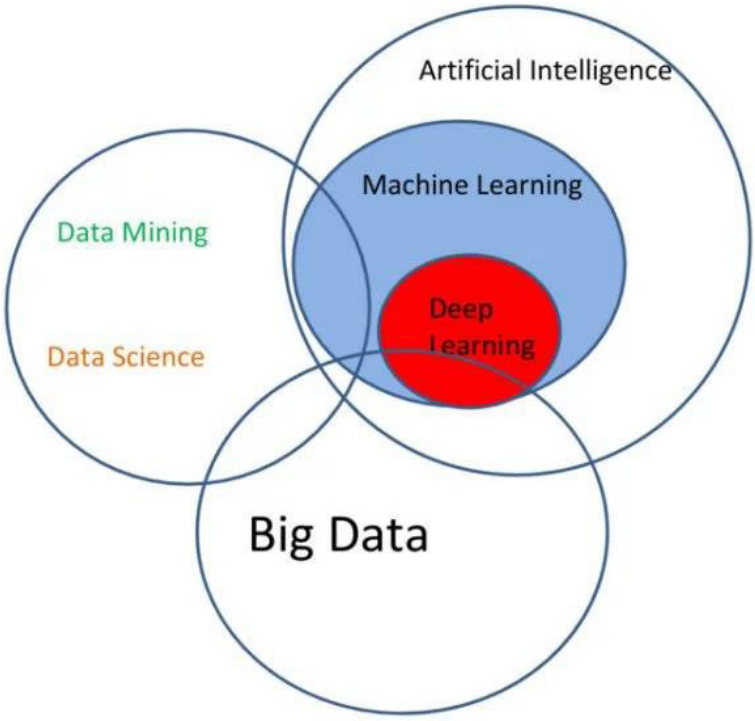
深度学习是机器学习中的一种技术，机器学习包含深度学习。机器学习还包含其他非深度学习的技术，比如支持向量机，决策树，随机森林，以及关于“学习”的一些基本理论，比如，同样都能描述已知数据的两个不同模型，参数更少的那个对未知数据的预测能力更好（奥卡姆剃刀原理）。

深度学习是一类特定的机器学习技术，主要是深度神经网络学习，在之前经典的多层神经网络的基础上，将网络的层数加深，并辅以更复杂的结构，在有极大量的数据用于训练的情况下，在很多领域得到了比其他方法更好的结果。

机器学习与大数据

大数据：机器学习的基础，但在多数语境下，更侧重于统计学习方法。

机器学习，深度学习，数据挖掘，大数据的关系可以用下图表示



系统学习资料

深度学习火起来之后，网上关于深度学习的资料很多。但是其质量参差不齐。我从 2013 年开始就关注深度学习，见证了它从一个小圈子的领先技术到一个大众所追捧的热门技术的过程，也看了很多资料。我认为一个高质量的学习资料可以帮助你真正的理解深度学习的本质，并且更好地掌握这项技术，用于实践。

以下是我所推荐的学习资料：

首先是视频课程。

Yaser Abu-Mostafa

加州理工的 Yaser Abu-Mostafa 教授出品的机器学习网络课程，非常系统地讲解了机器学习背后的原理，以及主要的技术。讲解非常深入浅出，让你不光理解机器学习有哪些技术，还能理解它们背后的思想，为什么要提出这项技术，机器学习的一些通用性问题的解决方法（比如用正则化方法解决过拟合）。强烈推荐。

课程名称：Machine Learning Course – CS 156

视频地址：

https://www.youtube.com/watch?v=mbyG85GZ0PI&list=PLD63A-284B7615313A

Geoffrey Hinton

深度学习最重要的研究者。也是他和另外几个人（Yann LeCun，Yoshua Bengio 等）在神经网络被人工智能业界打入冷宫，进入低谷期的时候仍然不放弃研究，最终取得突破，才有了现在的深度学习热潮。

他在 Coursera 上有一门深度学习的课程，其权威性自不待言，但是课程制作的质量以及易于理解的程度，实际上比不上前面 Yaser Mostafa 的。当然，因为其实力，课程的干货还是非常多的。

课程名称：Neural Networks for Machine Learning

课程地址：https://www.coursera.org/learn/neural-networks

UdaCity

Google 工程师出品的一个偏重实践的深度学习课程。讲解非常简明扼要，并且注重和实践相结合。推荐。

课程名称：深度学习

课程地址：https://cn.udacity.com/course/deep-learning--ud730

小象学院

国内小象学院出品的一个深度学习课程，理论与实践并重。由纽约城市大学的博士李伟主讲，优点是包含了很多业内最新的主流技术的讲解。值得一看。

课程名称：深度学习（第四期）

课程地址：http://www.chinahadoop.cn/classroom/45/courses

推荐阅读书目

《Deep Learning the Book》—— 这本书是前面提到的大牛 Yoshua Begio 的博士生 Goodfellow 写的。Goodfellow 是生成式对抗网络的提出者，生成式对抗网络被 Yann LeCun 认为是近年最激动人心的深度学习技术想法。这本书比较系统，专业，偏重理论，兼顾实践，是系统学习深度学习不可多得的好教材。

英文版：http://deeplearningthebook.com

目前 Github 上已经有人翻译出了中文版：

https://github.com/exacity/deeplearningbook-chinese

推荐学习路径

不同的人有不同的需求，有些人希望掌握好理论基础，然后进行实践，有些人希望能够快速上手，马上做点东西，有些人希望理论与实践兼顾。下面推荐几条学习路径，照顾到不同的需求。大家可以根据自己的特点进行选择。

Hard way

Yaser -> Geoffrey Hinton -> UdaCity -> Good Fellow

特点：理论扎实，步步为营。最完整的学习路径，也是最“难”的。

推荐指数：4 星

Good way

Yaser -> UdaCity -> 小象学院 -> Good Fellow

特点：理论扎实，紧跟潮流，兼顾实战，最后系统梳理。比较平衡的学习路径。

推荐指数：5 星

"Fast" way

UdaCity -> Good Fellow

特点：快速上手，然后完善理论。

推荐指数：4 星

"码农" way

UdaCity

特点：快速上手，注重实践。

推荐指数：3 星

阿里科学家王刚、吴翰清同时入选 MIT2017 年度 TR35 开创中国互联网企业先河

了不起的

8 月 16 日，《麻省理工学院科技评论》（MIT Technology Review）杂志揭晓 2017 年全球青年科技创新人才榜评选结果，阿里巴巴人工智能实验室首席科学家王刚、阿里云首席安全科学家吴翰清脱颖而出，获此殊荣。

自该奖项创立 18 年以来，这是第一次中国公司里同时有 2 人入选榜单。

科技领域对 TR35 并不陌生。该奖项创立于 1999 年，这是一个针对 35 岁以下青年科技才俊，为找出最有可能改变世界的牛人而设立的奖项。每年，该杂志会在全球 IT（计算机、通信、网络）和生物医药、商业等领域内，从影响力、创新力、进取力、未来潜力、沟通力五个维度进行评估，挖掘学术界和工业界的 35 位科技创新精英。

“这些年来，我们所选择的这些青年才俊的创新成果和公司一直深刻地影响着人类进步的方向，”《MIT Technology Review》总编辑兼发行人 Jason Pontin 表示。

从以往的上榜者来看，Google 联合创始人拉里 · 佩奇（2002 年）和谢尔盖 · 布林（2002 年），Linux 之父林纳斯 · 托瓦兹（1999 年），Facebook 创始人马克 · 扎克伯格（2007 年），Yahoo 创始人杨致远（1999 年），Apple 设计总监乔纳夫 · 伊森（1999 年）等，都曾是该荣誉的获得者。

正因如此，TR35 的门槛极高。据统计，从 1999 年到 2016 年，共评选出 820 名 TR35，中国入选者有 17 人。今年，阿里巴巴有两位科学家同时入选，创下中国科技企业的先河。

获奖者王刚是前新加坡南洋理工大学终身教授，今年 3 月加入阿里巴巴 AI Labs，负责机器学习、计算机视觉和自然语言理解的研发工作。2016 年，他因在深度神经网络设计上的卓越贡献，成为 MIT 评选出的 10 名亚洲区 35 岁以下青年科技创新人才得主之一。在加入阿里巴巴之后，因推动了 AI 商业化再次入选了 MIT 全球青年科技创新人才榜。今年 7 月阿里 AI labs 发布的人工智能消费级产品“天猫精灵 X1”就运用了王刚的多项研究成果。

获奖者吴翰清是中国互联网安全领域入选 TR35 的第一人。2005 年加入阿里，参与创建了阿里巴巴、淘宝、支付宝、阿里云的应用安全体系。23 岁成为阿里巴巴最年轻的高级技术专家，是阿里安全从无到有、从有到强的亲历者。

吴翰清和王刚的入选，也意味着中国科技力量正获得全球科技行业的认可，背后则是对科技的坚持与战略投入。

今年 3 月，阿里巴巴推出 NASA 计划，面向未来 20 年组建强大的独立研发机构，为服务近 20 亿人的新经济体储备核心科技。为实现该目标，一方面由金榕、华先胜、任小枫等技术领军人物，组建 iDST、AI Labs 等研究机构；另一方面发布首个全球性科研项目“AIR”计划，推进计算机科学领域基础性、前瞻性、突破性的研究，构建技术生态。据第三方报道统计，2015–2017 年，阿里巴巴的 AI 人才增幅高达 325%。

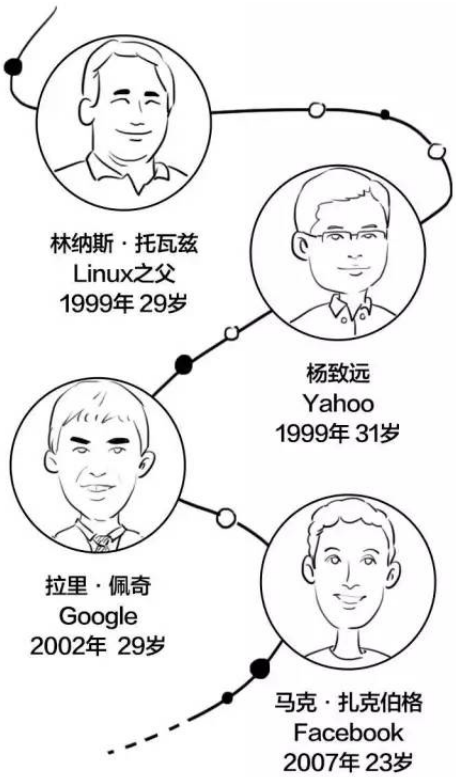
最后我们用一张图，全面感受这个振奋人心的消息！

8月16日，《麻省理工学院科技评论》（MIT Technology Review）杂志揭晓2017年全球青年科技创新人才榜（TR35）评选结果，阿里巴巴集团人工智能实验室首席科学家王刚、阿里云首席安全科学家吴翰清脱颖而出，获此殊荣！

TR35是什么?

TR35是《麻省理工科技评论》为表彰青年创新人物而设立的评选制度，每年在全球评选出35位35岁以下的青年科技才俊。上榜者皆是学术界和工业界的科技创新精英。

TR35所挖掘的新人及其项目都极具创新性，不少都成为风云人物，他们也正在改变着我们的世界。曾经的获奖者有如：



TR35的门槛极高，从1999年–2016年，共评选出820名TR35，中国入选者仅17人。今年，阿里巴巴有两位科学家同时入选，创下中国互联网企业的先例。

王刚
和他的深度神经网络

前新加坡南洋理工大学终身教授，今年3月加入阿里巴巴AI Labs，负责机器学习、计算机视觉和自然语言理解的研发工作。

2016年，他因在深度神经网络设计上的卓越贡献，成为MIT评选出的10名亚洲区35岁以下青年科技创新人才得主之一。加入阿里巴巴之后，因推动了AI商业化再次入选了MIT全球青年科技创新人才榜。

王刚教授的深度神经网络是AI商业化背后的技术引擎。神经网络是AI的“大脑”，强大的“大脑”能让人工智能更聪明。他的研究是让这个“大脑”具备更强的学习能力，让它从“应试教育”进化到“自主学习”！

深度神经网络这个“大脑”，是能够让机器在海量数据中学习现实世界的规律的系统。这个系统的高明之处在于它拥有自动适应的能力，能够分辨数据中细微的差别，从而大大提高学习效果！

目前这个方法已经在很多重要的机器学习问题上取得了更好的效果，今年7月阿里AI labs发布的“天猫精灵X1”就运用了这项研究成果。

吴翰清

和他的弹性安全网络

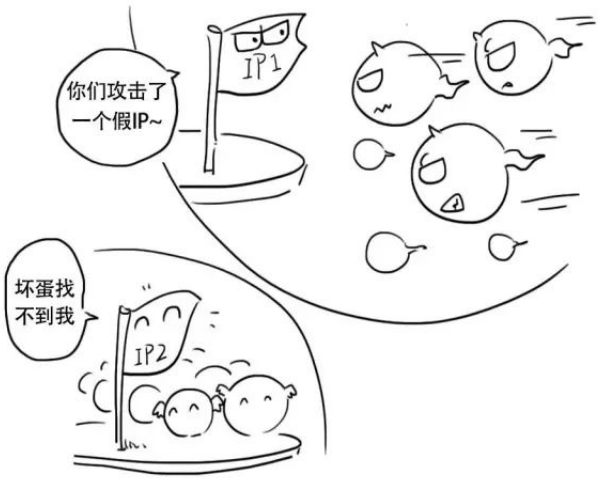
人称道哥、刺总，2000年开始研究安全技术，2005年加入阿里巴巴，是阿里安全的早期建设者，目前负责阿里云云盾。他长期活跃在中国的安全社区，在行业中极具影响力，《白帽子讲Web安全》、“道哥的黑板报”相信大家都听说过。

道哥未来真正想要实现的是整个互联网的重塑，建设一个只有安全流量的自成闭环的互联网。

在互联网庞大的流量中，有很多不安全的流量，它们就像坏人进行着攻击、欺诈，我们要做的是把他们牵制在世界的边缘。当流量被允许进入我们地盘的时候，他们本身就是干净的，我们把它叫做弹性安全网络。

如何做到这点?

以DDoS攻击为切入点，当流量攻击一个IP，我们就马上干掉这个IP，然后启用新的IP。这时候攻击者会跟随，但是跟随是有成本的，一方面它会有大概10多分钟的延时，另一方面我们能够通过数据分析辨认攻击者，给它一个无用的IP地址。这是整个弹性安全网络的核心思想。

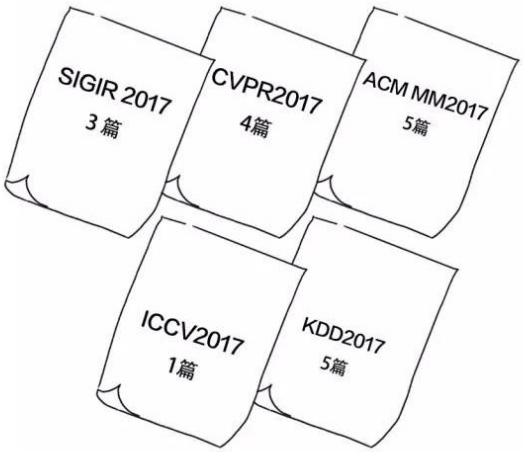


所以要做到这一点，常被人忽视的东西，那就是DNS。我们真正想要去做的，是干掉整个互联网最核心的心脏-DNS，我们要把它变成自建调度的一套网络架构。

从“互联网+”的商业创新引领者转向科技创新引领者

阿里的科技成果

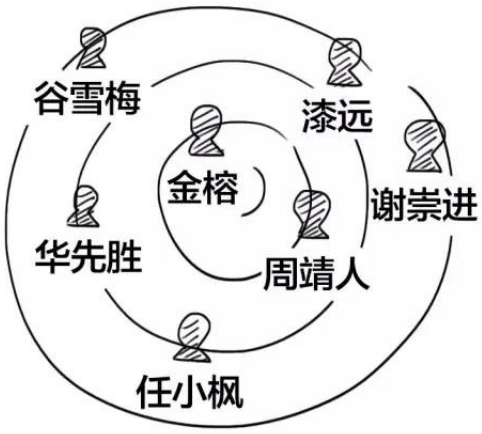
诸多论文入选全球顶级会议.



阿里的科技产品

阿里的科技人才

阿里在AI、计算机视觉、光通信、云计算、大数据等技术领域，吸引了大量人才和全球技术领军人物。



为实现NASA计划，为服务近20亿人的新经济体储备核心科技，阿里一方面组建iDST、AI Labs等研究机构；另一方面发布首个全球性科研项目“AIR”计划，推进计算机科学领域基础性、前瞻性、突破性的研究，构建技术生态。

吴翰清、王刚入选TR35，是阿里NASA计划的小荷一角，更是全球科技行业对中国科技力量的认可！

从来往到钉钉，从技术 Leader 到产品负责人，陶钧到底经历了什么？

阿里味儿

难受，那是因为你正在爬坡，正在成长

你有没有想过，如果你现在做的工作，能看到你 10 年后的样子，你愿意吗？

领着固定薪水，做着熟练到形成肌肉记忆的事情。

有一天，在舒适区里呆得太久，想跳出来，发现已经没有了斗志。

平庸的人生，就是从接受自己的平庸那一天开始的。

如果以上的情景，**有一丢丢刺痛你**，这篇**阿里技术大神陶钧**的分享，值得你看一看。

阿里技术大牛陶钧

“一旦安稳下来，我就会很焦虑”

你可能听过一句话，消停点吧，都三十多岁了，还折腾啥。

2010 年，我刚好 30 岁，从干了七年的前东家离职，到了淘宝。

前东家是一家外企，很多手机巨头都是我们的客户，也一度是行业的翘楚。但在这家公司的最后一两年，**我能感受到自己的成长速度在下降，更多的是经验驱动工作，而不是新的思考和成长驱动**，而且我能感觉到，外面的世界正在发生变化，最终促使我离开。

一开始，我在手机淘宝负责客户端 App 开发团队，虽然那个时候只有几万 DAU，但正好赶上行业发展的风口，用三年的时间，就见证了手淘 DAU 从 5W 到百万，到千万的改变，而我的团队也扩大到了 70 人。

2013 年，我又开始有点不安分了，不管我有多忙，我的思想都停留在了舒适区。这样的状态久了，我会发现自己的进步又开始慢下来了，就会开始焦虑。

在 2013 年底，阿里巴巴 All-in 无线之际，我来到了来往，“全新的挑战，创业的激情”是我选择这里的初心。

“创业是要自己找答案的”

但说实话，我没有想到会有这么困难，会有那么多挑战。

从最开始的快速增长，到 6 个月后，来往业务没有找到突破口，数据开始缓慢的持续的下跌。我们一直努力去做新的场景，**平均每 3 周就会拿出一个新的社交场景，给我们的用户去使用，大部分都没有取得成功**。

在当时，这对于我们这个年轻的团队而言打击是很大的，士气低落，有点懵逼，有点迷茫……

每天开车到园区，都觉得无法面对，恨不得调转车头回家。走在园区，遇到熟人有人关心我：“你们明年肯定 3.25，你还不快转岗?”……

团队里有人要走，或转岗，或离职….. 我不想放弃，但也很低迷。

那时候又陷入到一个死局里面，老大们说上线一个什么功能，我们就上线什么功能，以为是抓到了救命稻草，但并没有改变什么。

直到我遇到一件事情，2014 年 6 月，当时有一个前辈大牛，在外面创业，找我去聊。一见面他就问我，你们来往现在怎么样啊？我说了一下我的迷茫和困扰。

（陶钧喜欢徒步，大脑宁静片刻，想清楚自己有什么，要什么，放弃什么）

大牛说，“每周都有几个人跑来我这倒苦水，都跟你一样。**你们也就是温室里的花，就是矫情。你的状态只是在等待马总或者什么人，再给你们一个答案，指一个**

方向，然后你们可以继续享受所谓创业的激情和快感。小伙子，创业是要自己找答案的”。

坦白说，这句话点醒了我，的确，创业最重要的就是坚持，不确定性和失败的风险是常态。

一个天天在外面风吹雨晒的人，不会为明天房子可能会漏水而烦恼。

答案已经找到，难受是因为假装在创业，回归本心，坚定了就好。

“要转岗的，我一个都没有留”

一回来，我就开始稳定团队和找新突破了。这时，我又有一个有意思的发现：**要转岗的人的和要离职去创业的人想法完全不同。**

转岗的人更强调他不看好来往的前景，而对于要去的部门，理由是业务更有前途，会有更多发展，更多股票。

要离职出去创业的人说，**虽然来往失败了，我还年轻，我还能搏一把。**

后来，对于要转岗的人我基本不留，浪费时间，能力再强也不留。

但对于想出去创业的人，我会尽一切办法挽留，这些人才是来往真正需要的人。**人在逆境的时候，心气是最重要的。**

（钉钉管理层团队去拜访老河沟，大家一起爬山拉练，在山上吃面）

“不撞南墙不回头，就算撞了也不回”

有时候，当你坚定了内心，上帝也会给你打开一扇窗。

2014 年，无招带领团队在湖畔花园开启了钉钉产品。那个时候，钉钉还只是一个项目，有一个内部代号，叫工作圈。

因为无招需要支援，我每天白天在西溪园区处理来往和技术的工作，晚上去湖畔花园钉钉团队工作。

但就是那时候，**我开始慢慢觉得这里有点不一样了。无招带领钉钉的小团队，白天和客户共创，晚上回到湖畔花园，团队成员无论开发还是产品都兴奋的参与讨论，然后快速实现。**

（阿里有将创业团队“扔”去湖畔花园的传统，每一天都充实给力）

我喜欢这种火热的氛围，战斗的气息。最后，到钉钉正式发布前，我基本上天天都泡在湖畔花园。钉钉 1.0 发布取得了非常好的口碑。

我和小伙伴的干劲都挺高，**每天在 AppStore 上用户对我们的评价，做得好就奖励自己一个鸡腿，做得不好就赶紧补上。**后来，整个 App 运行流畅，在电量，流量，速度，稳定等方面都有了巨大的进步。

而对我而言，更重要的是，我们一起造就了一支优秀的创业团队。**就算是从万众瞩目的 CEO 项目到业务跌入低谷，核心成员都没有放弃。**

不撞南墙不回头，就算撞了也不回。

在创业团队里重新创业

2015 年 5 月，钉钉搬进了龙章大厦，办公室楼下是一棵名为九爪龙章的古树，马路对面是西溪湿地。古树枝繁叶茂，钉钉也进入快速发展的阶段，我在钉钉负责钉钉客户端 & 前端技术团队，同时也负责钉钉开放平台业务。

好不容易业务走上正轨，无招有一天突然又找我，**让我去做智能办公业务，而且是一个人去。**

我加入智能办公团队时，这个团队只有 3 个同学。**相当于重新在钉钉内部组建一个新的团队，**首先人就要自己去招，客户要自己去共创。这也意味着我将从一个技术

leader 角色转变成一个产品业务负责人（Product Owner，PO），**为钉钉智能办公业务结果负责。**

“PO 真的是我要的发展路线吗？我保留 Tech Leader 职位，同时做 PO，这算是给自己留一条后路吗？”，这些问题我不止一次问过自己，最后还是回归本心，关键是怎么做？ PO 对我的挑战是摆在眼前的，先做好它，不辱使命。

所以，**我主动提出不做 Tech Leader 了，就做一个 PO，也不用带人；做不好也没有退路了。**这一点，在做智能办公 PO 时，给了我巨大的动力，**每天抱着不安全感就是最大的安全感，**这激发着我，也激发着智能办公团队的所有小伙伴。

Stay Hungry，Stay Foolish. But Be Crazy

做钉钉智能办公 PO 是我成长最快的阶段，现在智能办公已经成为钉钉上中小企业最受欢迎的应用。

我们钉钉有一句话：“**Stay Hungry，Stay Foolish. But Be Crazy**”，前面是 Jobs 的名言。我们对这句话有自己的解读，**保持谦卑的心，才能静下心来去和我们的客户一起，理解他们的工作方式，做出他们需要的好产品。**

（钉钉的口头禅是，我们还在创业）

而钉钉人的内心狂热，让我们互相感受到力量，这在我们不断突破前进的路上，至关重要。**面对中国 4300 万中小企业正在进入移动智能办公时代的大潮，我们现在还在路上。**

最后说说创业，我们说自己在创业，我的很多朋友也在创业。创业和人在哪里，关系不大。**现在有两种人，一种是在互联网创业的人，还有一种是在互联网工作的人。**

我们之所以在钉钉这么疯狂，就是因为我们在创业。创业不是喊出来的，也不是不拿工资就叫创业，也不是拿了很多股权就算创业。

创业是看你做的事情有多大挑战？你和你的团队取得了什么突破？我们所做的事情对客户有多少价值，对社会有什么价值？你是不是绝对的坚定？你是不是异常的优秀？

我的感觉是，因为人面对挑战和压力时，会本能的让你看到机会，看到往前走的路；难受的时候不放弃，能让你坚持下来，抓住机会。**难受，那是因为你正在爬坡，正在成长。**

有些人的人生，一直在浑浑噩噩中度过。等待变化和挑战真的来临时，就抱怨为什么要倒霉的是我，我究竟做错了什么。

其实，有时候，你没有做错啥，你错在啥都没做。

“成长是自己的事情，坚持学习。

舒适的时候寻求突破，难受的时候不要放弃，因为人面对挑战和压力时，会本能的让你看到机会，看到往前走的路。难受的时候不放弃，能让你坚持下来，抓住机会。难受，那是因为你正在爬坡，正在成长。

不要一专多能，专业的同时尽量保证自己有足够的技术视野，找到新的浪潮。”——陶钧

阿里妹想问大家一句话：

你敢想象，你 10 年后的生活吗？

你想象着那一天，你嘴角会有微笑吗？你充满期待吗？

陶钧的团队目前在招人，没有什么限制，有创业之心，想做点改变的人都欢迎。

本文来源：阿里味儿（ID：aliweierV）

阿里技术

扫一扫二维码图案，关注我吧

「阿里技术」微信公众号

「阿里技术」官方微博