促进软件开发领域知识与创新的传播

架构师
ARCHITECT

热点 | Hot

谷歌Fuchsia：一个新的操作系统

推荐文章 | Article

10x程序员应该具备哪些素质

阿里业务监控平台架构演进

观点 | Opinion

2017年会是Serverless爆发之年吗？

专题 | Topic

深度学习框架：2016年的大盘点

Geekbang 极客邦科技 | InfoQ

2017年5月

卷首语

杨赛

2009 年，一位信奉自由至上主义的女青年在初识软件技术的世界之后，毫无悬念的变成了一只开源理念的脑残粉。把各种 Linux 发行版挨个装来卸去，在 SourceForge 和 Launchpad 上东张西望，没事儿再刷刷 Linus 和 RMS 这些大神们的八卦，如此这般。

她相信创新的前沿就在那里。

在 2012 年，OpenStack 项目初露头角，一度被誉为“第二个 Linux”。在开源的世界，这几乎是一个项目所能得到的最高赞誉；于是，开源脑残粉自然而然的开始关注 OpenStack。在这样一个新生项目，你不断看到活生生的人在加入、讨论、以及尝试做些新的事情。开源脑残粉这次走的近了一些，深了一些。

结果在两年之后，开源脑残粉成功转型成了一只 OpenStack 黑、开源怀疑论者和 AWS 脑残粉。

因为她看到创新的前沿正在 AWS 的那堆机器上面狂奔，OpenStack 拖着庞大的身躯在后面喘着粗气追赶，而这是开源理念所解释不了的。

时间转瞬到了 2017 年，此时的这位开源怀疑论者正在红帽峰会的会

场里四处溜达。红帽一直以来都是开源死忠们的阵营，至今初心不改，坚决相信未来是属于开源的。他们已经知道用 OpenStack 是造不出来第二个 AWS，这的确令人沮丧；但是他们拍拍身上的土，转而找到了另一个开源的方式，得以把 AWS 上面那些狂奔的创新们交付给自己的客户。

开源怀疑论者看看红帽，看看 AWS，再看看红帽的客户们——那些传统大企业们正在做的数字转型，忽然意识到了一件事。

1. 开源的共享协作模式的确能够加速创新。
2. 但重复性的人工劳动却拖了创新的后腿（资源有限的情况下尤其如此）。
3. 自动化为创新带来的价值，说不定远在开源模式之上（在机器总数远超人类总数的情况下尤其如此）。

AWS 虽然谈不上拥抱开源，但要说拥抱自动化，世间无人能及。他们不断的将一切属于人类的经验转移到了机器系统内部，变成了属于机器的共同经验，所以才能够放心的将所有属于过去的交给机器，自己则没有后顾之忧的向未来前行吧。

CONTENTS / 目录

架构师

2017 年 5 月刊

本期主编 徐 川

流程编辑 丁晓昀

发行人 霍泰稳

提供反馈 feedback@cn.infoq.com

商务合作 sales@cn.infoq.com

内容合作 editors@cn.infoq.com

谷歌的 Fuchsia：一个新的操作系统

作者 Nur Hussein 译者 孙薇

引言

谷歌开发的新开源 OS Fuchsia 引发了很大的关注，但此前由于信息不足，我们只能对这个系统的具体用途、背景信息以及架构做些猜测。本文对这个系统进行了较为详细的介绍，并给出了可安装的平台和安装方式，有兴趣的开发者可以按部就班进行尝试。本文翻译已获得原文作者 Nur Hussein 和英文网站的授权。

Fuchsia 是谷歌开发的一个操作系统，这是一个差不多从头研发的新系统。2016 年 8 月，Fuchsia 的开发新闻在技术新闻网站上引发了轰动，不过很多细节依旧成谜。Fuchsia 是一个开源项目，无论开发工作还是相关的文档都还在持续完善中，除了该项目为开源之外，谷歌并未透露更多

信息，包括其真正用途。根据零散分布在网上的文档、源码等信息，我们可以推测：Fuchsia 是一个专为 PC、平板电脑及高端手机所开发的一套完整的操作系统。

在源码库中可以下载到 Fuchsia 的源码及所有组件，如果诸位喜欢探索实验性质的操作系统，可以尝试一下这个系统——非常有趣。Fuchsia 包含一个内核，其顶层是使用者空间（User Space）组件，负责提供 Library 及工具。源数据库中，在 Fuchsia 目录的下面有许多子项目，主要都是协助开发者创建应用的资料库及工具项目。Fuchsia 大部分使用三条款 BSD 许可证（3-clause BSD license）授权，不过其内核是基于另一个项目 Little Kernel（LK）建立的，而后者使用了 MIT 许可证，因此其内核的授权协议是混合型的。Fuchsia 中包含的第三方软件都是根据其各自的开源许可证进行授权的。

Magenta

Fuchsia 的核心是 Magenta microkernel，它负责管理硬件，并为系统的使用者空间组件提供抽象层，类似于 Linux 对 GNU 等项目的支持。而 Magenta 建立的基础——Little Kernel 则是 Fuchasic 的开发者 Travis Geiselbrecht 在加入谷歌前所研发的一个项目。LK 的目标是要建立一个能够运行在有限资源的微型嵌入式系统（类似于 FreeRTOS 或 ThreadX）之上的小型内核，而 Magenta 则是针对更复杂些的硬件——想要允许的话，需要至少 64 bit 的 CPU，配以一块内存管理单元。因此，Magenta 在 LK 的有限功能上进行了扩展，它使用了 LK 的“内部构造”，包含线程、互斥锁、计时器、事件（信号）、等待队列、信号及虚拟内存管理器（VMM）。在 Magenta 中，LK 原本使用的 VMM 已经有了实质提升。

Magenta 的一个关键设计在于运用了 capability，这是一个计算机科学的抽象概念，封装了访问某个对象的权限。这一概念是 1966 年由 Dennis 和 Van Horn 首次提出的（点击这里查看原 PDF），指的是一个不可伪造的数据结构，在操作系统中担任控制访问权限的基元。在 Magenta

中，开发人员使用了 capability 的模型，以定义某个进程与内核、与其他进程的互动方式。

具体实现中，Magenta 采用了名为 handle 的构造，只要有进程请求创建内核对象，就会自动生成 handle，用于处理与该内核对象的“会话”，几乎所有系统调用都需要通过 handle 来实现。handle 包含与其相关的权限，也就是说它们定义了在使用时允许哪些操作。此外，handle 可以通过进程复制或转移。可授予 handle 的权限包括读写相关核心对象的权限，只要是虚拟内存对象即可，无论能否被映射为可执行对象。对于沙盒中的特殊进程来说，handle 非常有用。因为经过调整，handle 可以设置为仅允许对系统的某个子集可见、可访问。

由于我们将内存作为可通过核心对象访问的资源，各个进程可通过 handle 来获得对内存的应用。在 Fuchsia 中创建进程，代表着一个负责创建的进程（比如 shell）必须手动为子进程完成创建虚拟内存对象的工作——这一点与传统不同：在传统的 Unix 类内核，比如 Linux 中，内核会完成大部分的虚拟内存设置，自动处理进程。Magenta 的虚拟内存对象可通过多种方式来映射内存，而且在进程执行中灵活性很高。甚至可以想象完全不对内存做映射的场景，在此场景下，通过类似文件描述符之类的 handle，也可以对系统进行读写。尽管这种设置允许任何类型的创造性使用，这也意味着想要运行进程，需要通过使用者空间的环境完成许多的基础架构工作。

由于 Magenta 是按照微内核来设计的，操作系统的大多主要功能组件也是当作使用者空间进程来运行的，其中包含驱动程序、网络堆栈以及文件系统。最初从 IwIP 引导的网络堆栈最终被 Fuchsia 团队所编写的用户网络堆栈所取代。网络堆栈是一个中间性应用，简于使用者空间的网络驱动，以及需要网络服务的应用之间，通过网络堆栈，系统提供了一个 BSD socket API。

默认的 Fuchsia 文件系统被称为 minfs，也是从零开发的。驱动管理器创建了根文件系统内存，为其它安装在下面的文件系统提供虚拟文件系

统层（VFS）。然而，由于文件系统是作为使用者空间的服务器来运行的，需通过服务器的相应协议来访问。文件系统中的各个实例都包含后台运行的服务器，以处理所有的数据访问。借助使用者空间的 C library，对于只进行调用以开关、读写文件的用户程序来说，协议都是透明的。

Fuchsia 的图形驱动程序也作为使用者空间的服务存在，按照逻辑分为系统驱动以及应用驱动。两者之间促进沟通的软件层被称为 Magma，是一个提供合成与缓冲区共享的框架。绘图堆栈的某部分即 Escher，这是一个基于物理的渲染器，通过 Vulkan 来提供渲染 API。

完整的 POSIX 兼容并非 Fuchsia 项目的目标，足够的 POSIX 兼容性可通过 C library 来提供，C 库也是 musl 项目连接 Fuchsia 的端口。这样一来，将 Linux 程序移植到 Fuchsia 系统会比较方便，但本身运行在 Linux 平台上的复杂程序自然需要更多的工作。

实践尝试

可按照本文档中的指南获取自己的 Fuchsia 安装程序并运行，方法十分简单。该脚本设置了一个 QEMU 实例，大家可自行尝试。它可以在模拟 x86-64 系统的 q35 或“标准 PC“的机器上，以及模拟 ARM-64 系统的 virt 或 Qemu ARM-64 虚拟机上运行，也可以在真机上运行。可按照指南来安装，针对机型包括配有 Intel Skylake 的 Acer Switch 12 寸笔记本、Broadwell 超迷你电脑 NUC（next unit of computing）以及 Raspberry Pi 3。就目前来说，虽然通过兼容外设也能在类似的硬件上运行，但对这三款机器的物理硬件支持也十分有限。

目前，针对 Fuchsia 系统编写应用的支持还不充分，还需要很长时间的开发和文档构建。据了解，谷歌的 Dart 编程语言在该系统中有着广泛应用，而基于 Dart、针对 iOS 和 Android 的移动应用框架 Flutter 的 SDK 就被移植到了这个系统中，似乎是创建图形应用程序的主要方式之一。负责窗口绘制以及用户输入的 compositor 叫做 Mozart，基本上相当于 Linux 上的 Wayland/weston。

当引导 Fuchsia OS 进入图形模式时，我们可以看到在带有选项卡的图形环境中包含了五个 dash shell。第一个标签展示的是系统消息，后面三个是可以在 Fuchsia 环境中启动应用程序的 shell。最后一个标签是 Magenta shell，里面的内容更为基础简单，由于缺少 Fuchsia 环境，无法通过它来运行图形应用。标签可以通过 Alt-Tab 来切换。

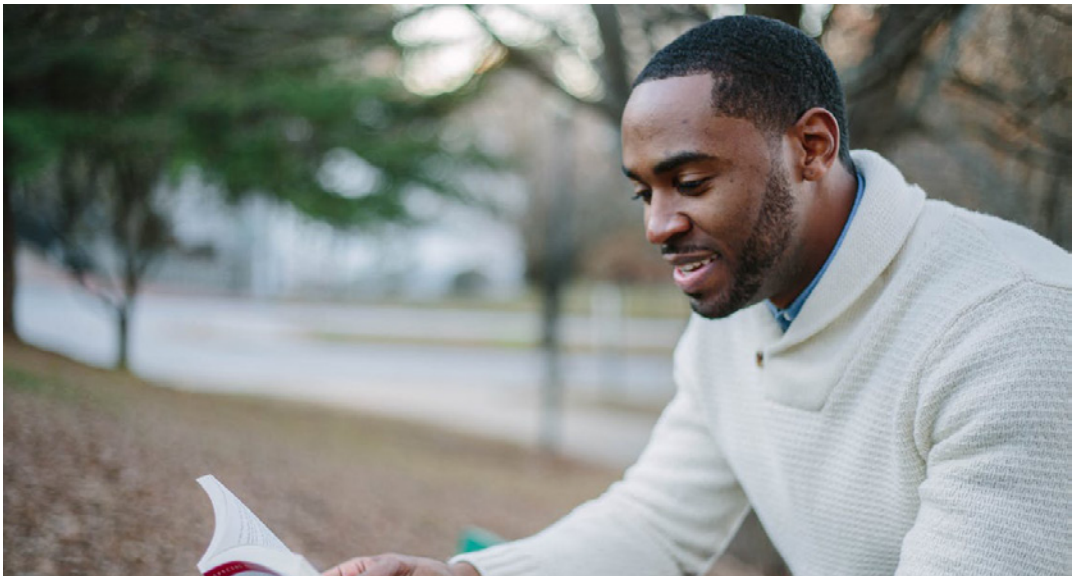
目前这套操作系统能够运行的程序并不多，用户组件的大多内容仍在积极地开发中。可以运行的案例应用包括一些命令行程序，比如经典的 Unix fortune，还有一个能在屏幕上绘制旋转方块的图形应用，叫做 spinning_square_view。尽管目前来说内容确实有限，不过敬请持续关注 Fuchsia 的资源库，开发者正在积极更新，以便让这个系统功能更加丰富。相信很快就有更多功能可以尝试。

结论

看到有新的操作系统出现，并且还需要很多开发工作才能逐渐完善，并投入使用总是颇有趣味。Fuchsia 还不完善，但目前看来发展的方向是正确的。由于谷歌在这个项目中投入了许多资源，目前针对 Magenta 以及其它 Fuchsia 组件的开发都非常活跃。对大众来说，所有承诺的功能都是能够实现的。不过目前信息还不够，想要了解项目的确切方向也有些困难。敬请大家关注这个新的开源项目，相信很快就有更多内容填充进来。

Redis 之父：10x 程序员应该具备哪些素质

作者 Antirez 译者 薛命灯



Fred Brooks（《人月神话》的作者）最早在他的论文“没有银弹——软件工程的本质和偶然性（No Silver Bullet - Essence and Accidents of Software Engineering）”中提出了“10x 程序员”的概念。技术社区对于这个概念呈现出两级分化的观点。Redis 之父 Salvatore Sanfilippo（antirez）列出了 9 种特质，他认为，如果一个程序员同时具备了这 9 种特质，那么就可以说他是一个 10x 程序员。以下内容已获得 antirez 的翻译授权，查看英文原文 The mythical 10x programmer。

一个 10x 程序员，在相同条件下，可以完成十倍于普通程序员的工作。这里所说的“普通程序员”，是指那些能够胜任自己工作的程序员，只是他们不具备 10x 程序员的神奇能力。普通程序员代表了这个领域所有专业程序员的平均水准。

对于是否存在 10x 程序员这种“神兽”，编程社区的观点呈现出两级分化：有人认为根本不存在所谓的 10x 程序员，有人则认为不仅存在 10x 程序员，如果找对了门路，甚至能找到 100x 的程序员。

如果说编程是一项“线性”的工作，那么很明显，10x 程序员是不可能存在的。一个跑步运动员怎么可能跑得比另一个快上十倍？在相同时间内，一个建筑工人建造的东西怎么可能是其他人的十倍？不过，编程是一项很特别的设计工作。虽然程序员可能不会参与程序的架构设计，但程序的实现仍然涉及到一些设计工作，而这部分需要程序员去完成。

如果说程序的设计和实现不是线性的，那么在我看来，经验、编码能力、知识和去伪存真的能力就不仅仅是线性的优势，它们在编程过程中相互交织，成倍地发挥效能。当然，如果程序员能够同时胜任设计和实现工作，那么这种现象就尤为明显。任务越是具有目标导向性，程序员就越是能够以更少的付出达成相同的目标，从而体现 10x 程序员的潜在能力。如果手头的工作很死板，而且限定了可使用的工具和实现方式，那么 10x 程序员事半功倍的能力就会大打折扣。不过，在不改变大前提的情况下，程序员仍然可能通过局部的设计优化来改进工作，包括在项目的某些部分不按常理出牌。所以，他们可以少付出很多却能达成几乎相同的目标。

在我的二十年程序员生涯中，我与其他程序员一起工作，作为同事，或者由我指导他们达成目标，为 Redis 和其他项目贡献代码补丁。在工作过程中，我仔细观察他们。与此同时，有很多人说我是一个高能的程序员。不过我不认为自己是一个工作狂，我只是编码速度比较快而已。

下面列出了我认为可以用于区分程序员生产力高低的重要特质。

编程裸技能：完成子任务

从处理编程子任务可以看出一个程序员的短板和长处，比如实现一个函数或者一个算法。但从我的经验来看，擅于应用基本的编程技能来高效完成任务的程序员并非如人们所想得那样普遍存在。有时候，团队里有些不是很称职的程序员，他们甚至不知道该怎么写一个简单的排序算法，但

比起那些看似称职却缺乏实战经验的程序员，这些不称职的程序员却能完成更多的工作。

经验：模式匹配

我认为，经验就是一系列解决方案，它们已经被证实可以用于处理一些重复性的任务。经验老道的程序员知道该如何处理各种子任务，这样不但省掉了很多设计工作，而且避免了很多设计错误，而设计错误是简洁性最大的敌人。

专注：实际时间和假设时间

花在编码上的时间不仅要看数量，也要看质量。造成注意力不集中的因素既有内部的，也有外部的。内部的因素包括拖延、对手头的项目不感兴趣（一个人总是做不好自己不喜欢的事情）、缺乏练习、缺乏睡眠。外部因素包括频繁的会议、不固定的工作环境、同事的打扰，等等。集中注意力和避免被打扰对于提高编程效率来说是至关重要的。有时候，为了集中注意力，需要采取一些极端的手段。例如，我会时不时地查看邮件，但大部分邮件先不做回复。

设计权衡：用 5% 换取 90%

项目的非根本性目标在很大程度上导致了设计的复杂性，或者导致无法达成其他更重要的目标，因为根本性功能和非根本性功能在设计上存在竞争关系。如果意识不到这点，复杂性就会随之而来。实现全面的设计并不是件轻而易举的事，付出与回报之间不能通过简单的比例来衡量。对于设计者来说，意识到这一点是很重要的。如果项目要最大化产出，那么就要把精力集中在重要的事情上，并在合理的时间内完成。例如，在设计 Disque（一个分布式消息队列）时，我发现对消息进行排序之后，项目的其他方面就会得到实质性的改进：可用性、查询和客户端交互、简洁性和性能。

简洁性

简洁性是成败之间最为明显的分界点。理解复杂性的产生过程有助于理解什么是简洁性。我认为，不愿意做出设计权衡和设计错误的累积是导致复杂性的两个主要因素。

在设计过程中，每次走错一条道，就离最优的方案越来越远。一个初始的设计错误，如果没能被纠正过来，那么可能导致一条道走到黑，最终得到的是一个复杂的系统，而不是对原先系统的重新设计。项目会因此变得更加复杂和低效。

程序员可以在脑子里进行“概念验证”，从大量简单的设计想法中选择可行性最高且最直接的方案，从而达成简洁性。在后续的改进工作中，个人的经验和设计能力开始发挥作用，为子任务找到更加明智的解决方案。

不过，如果系统复杂性不可避免，那么在放弃挣扎之前也要尽量想办法降低系统复杂性，甚至尝试采取完全相反的设计。

完美主义（为了偏袒设计而放弃生产力）

完美主义可以分为两种：一种是追求程序极致性能的工程文化，另一种是个人特质。不管是哪一种完美主义，它们都会对程序员实现快速交付造成阻碍。完美主义和对外部评判的恐惧会导致设计上的偏袒，程序员根据主观的心理因素和无关紧要的衡量参数做出设计决策，却忽略了健壮性、简洁性和及时交付。

知识：理论有益

在处理复杂任务时，具备一些理论方面的知识会对设计产生重要影响，比如数据结构方面的知识、了解计算能力的局限性和一些重要的算法。我们没有必要成为无所不知的超级专家，但至少要知道一些问题的潜在解决方案。例如，在给一个给定流统计单一元素的个数时，我们可以在设计上做出权衡（接受一定程度的错误），并结合概率集合的基数估计

（cardinality estimation）算法，避免设计出复杂、缓慢、低内存效能的解决方案。

底层：理解机器原理

程序的很多问题都是源于对计算机工作原理的误解，即使是使用高级语言开发的程序也不外乎如此。这种情况可能导致一个项目需要重新设计和实现，因为项目所使用的工具和算法出现了根本性的错误。精通 C 语言，知道 CPU 的工作原理，了解系统内核的行为以及系统调用的实现原理，做到这几点可以挽救你于危难之中。

调试技能

查找和解决 bug 经常会占用我们大量的时间。查找引起 bug 的问题根源，在合理的步骤内修复 bug，以简单的方式编写包含较少 bug 的代码，对于程序员来说，做到这几点可以显著提升效率。

一个程序员如果具备了上述几点特质，那么他们的产出将会有 10 倍的提升，对此我一点也不感到惊讶。综合这些特质，从一个可行的模型开始，实现更简单更好的设计。我认为简洁性就是一种“投机取巧的编程”。简而言之，就是在开发的每个阶段选择性地实现一些功能，以最小化的付出为用户带来最大化的影响。

阿里 Goldeneye 业务监控平台之架构演进，如何实时处理 100T+/ 天的日志量

作者 马国强 许琦

一、黄金眼业务背景

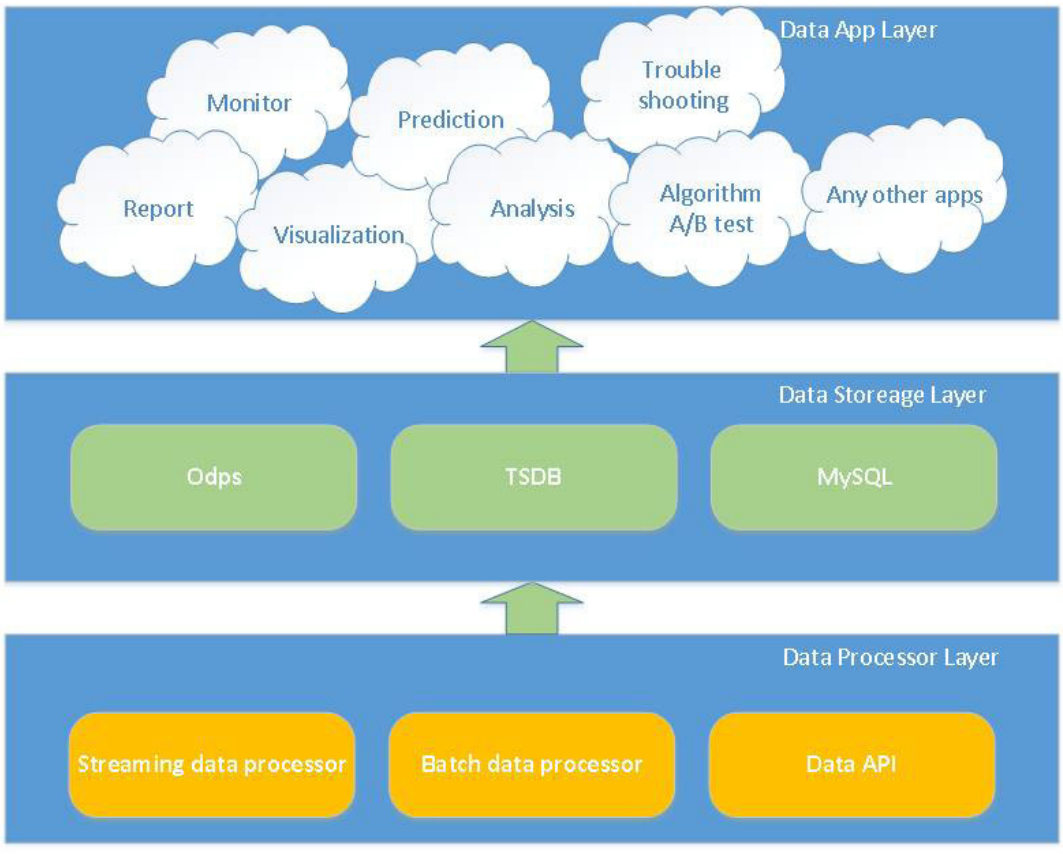
淘宝天猫等业务线上每天会存在的超大流量，这些流量就是阿里宝贵的数据资源。阿里妈妈线上每天会产生超大规模的流量，这些流量数据就是阿里宝贵的资源。在大流量上任何的服务器波动，故障都会造成巨大的经济损失。然而故障总是会出现，如何快速的定位问题，及早发送报警给开发同学，是解决这类问题高效、快速的途径。黄金眼业务需求也是为了解决这个问题从而孕育而生，黄金眼业务方面大致经历了以下几个发展阶段：

- 第一阶段：黄金眼只服务于单个业务线，主要针对大盘核心数据进行监控，监控的粒度从原始的小时级别提升到5分钟级。实时计

算取代离线计算成为监控报警的核心计算框架。

- 第二阶段：黄金眼开始在阿里妈妈崭露头角，逐步覆盖内部核心业务线，并得到一直好评。此阶段针对多个核心业务的大盘数据、细粒度数据进行监控，同时也逐步深入到引擎内部的指标进行故障定位预警，取得了很好的效果。
- 第三阶段：黄金眼在阿里妈妈内部覆盖90%以上的监控需求，同时将外部BU的需求也逐步接入到系统中。面对每个场景纷繁的需求，更细粒度的监控维度，黄金眼也逐步走向自动化，智能化的道路。

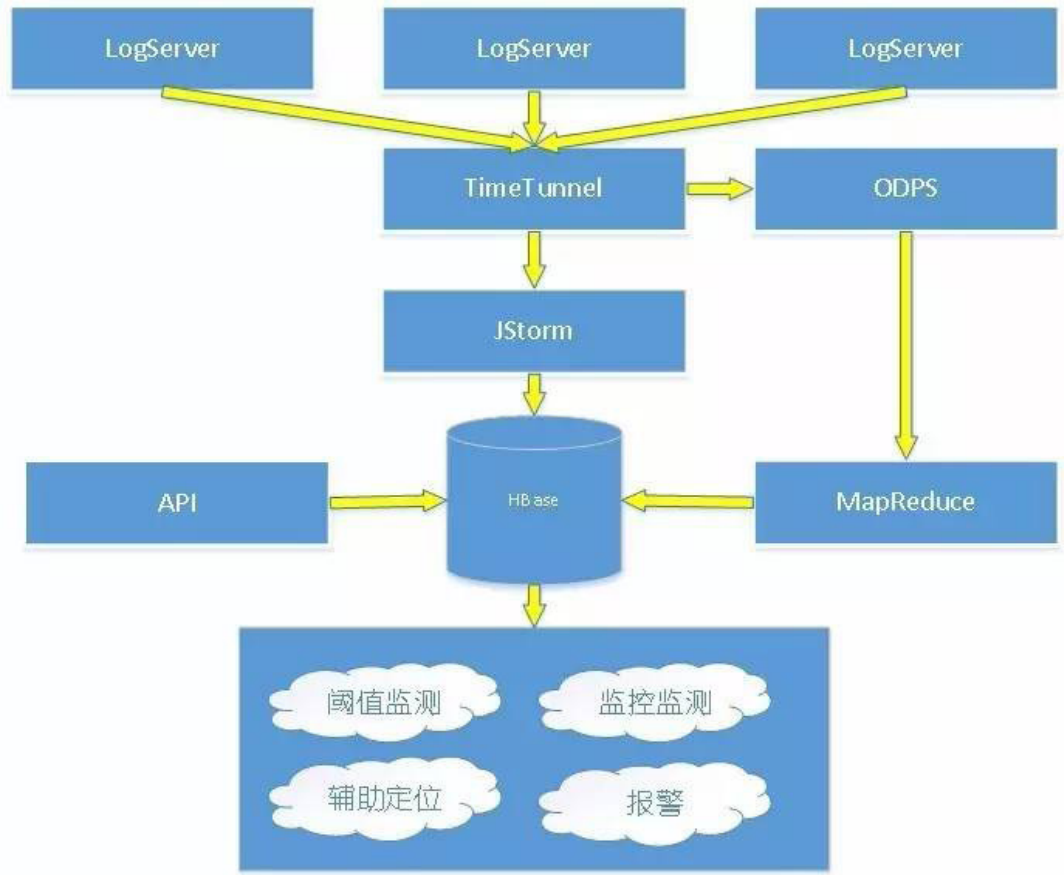
目前的黄金眼覆盖监控报警、数据可视化、预测服务、故障定位、数据分析、算法 A/B Test 等功能。在黄金眼的核心计算框架上不断提高内部性能，深挖各种数据模型，逐步完善黄金眼的技术深度和应用广度。



二、黄金眼架构演进

黄金眼为阿里妈妈业务保驾护航已久，这里以最初的版本为起点来阐述黄金眼架构的变迁。架构和业务是均衡发展的，业务的规模驱动着架构的演进，架构的变迁能够为业务灵活扩展提供稳定的支撑。

下文中所讲的三种架构版本的演进过程，是我们保持业务发展、系统性能和工作效率之间平衡的过程。反推这个过程，新的架构也适合之前的业务规模和场景，然而衡量一个架构的优劣不是兼容了历史，而是能预见未来。所以，我们在保持业务和技术均衡的基础上，结合对未来业务潜在的需求的思考，才产生了每一次架构的改进和升级。这些是黄金眼每位成员的集体智慧，供大家参考学习。



- 应用：构建在整个黄金眼之上，对外部用户提供各种有针对性的

服务，包括监控报警服务，预测服务，故障定位服务，算法A/B test服务等。

- 实时计算：黄金眼中95%以上的业务线指标数据都是通过实时计算得到的，需要监控报警的数据更是全部由实时计算storm平台解析，处理，汇聚之后写入分布式存储当中，供给前台展现报警。
- 存储： HBase集群在整个黄金眼中是集数据存储、展现的数据中心，所有的实时计算结果数据、阈值数据、其他时间序列的数据都要汇总到HBase存储当中
- 离线计算：离线计算部分主要处理的是离线的数据表，用于每天的日报，周报，反作弊数据统计，也是之前比较传统的统计报警模式。离线计算一般是以天为单位，可以优化到小时级别，但仍然存在报警晚等问题。

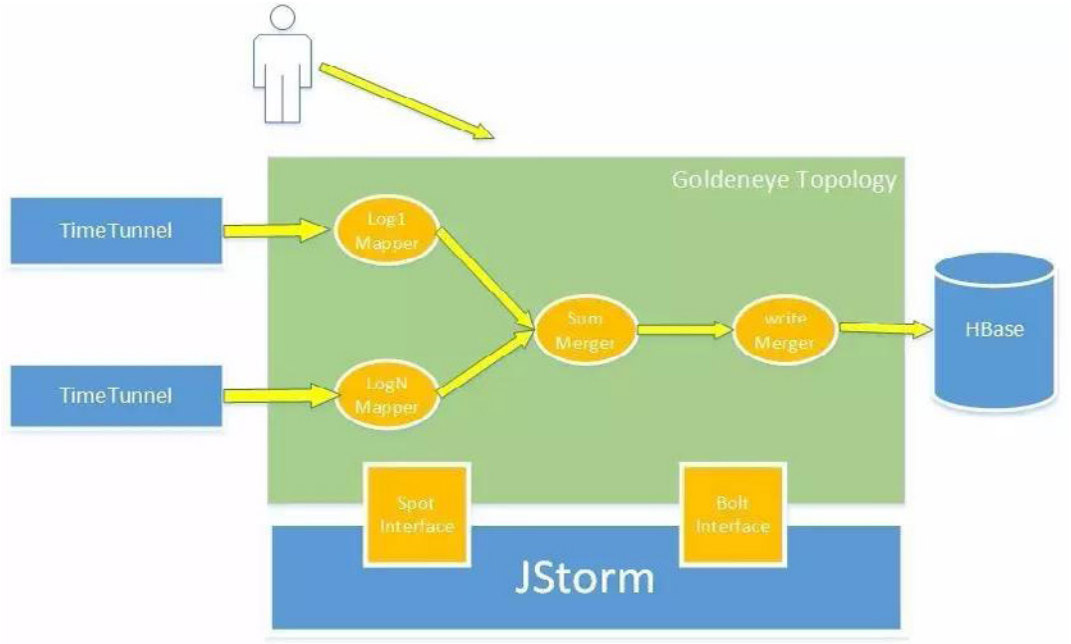
1、Goldeneye架构 V1版本——storm模式

这一阶段的业务特点就是野蛮生长，业务每天都会迸发出新的玩法，线上流量也在逐步突破历史最高。当线上出现问题需要排除查的时候，对几十亿级别的流量的线上系统故障维持 1 小时，损失是不可估量的。而业务还大部分依赖于离线计算，即使优化到小时级别，也很难减少损失。因此我们改变之前依靠离线报表进行监控报警的方式，引入 storm 实时计算引擎开发我们的监控报警任务。根据原始的 storm 语义，结合当时的大盘数据 sum 的需求，我们设计了第一版黄金眼实时计算的架构。对于每一个独立的业务线都定制一个独立的 storm 任务，每个实时任务的框架图如下所示。

Spout 阶段

该阶段主要和消息中间件进行通信，读取对应的实时日志消息，维护读取的时间戳保证数据的 exactly once。阿里内部使用的 Time Tunnel 作为实时的中间件，功能类似开源的 kafka。根据大盘监控的统计需求，组合对应的 key（例如广告位 + 时间戳进行组合）、value（例如 pv 数量）

信息，发送给下游的 bolt。



Count Bolt 阶段

使用 filed grouping 方式，相同的 key 在同一个 bolt 中处理。在 count bolt 内部，先进行初步的内存聚合，然后根据批量 batch 的条目数或者 batch 时间触发 emit 发送给下一级的 write bolt。

Write bolt 阶段

write bolt 阶段有两种模式，根绝数据量的情况选择 Put 模式或 increment 模式。put 模式采用先 get 在 put 的方式对数据进行更新，increment 模式直接发送给 HBase，让 HBase 内部的 region server 进行累计。

通过这一版本的改造，我们的计算周期从小时级别缩短到秒级别，报警周期缩短到 5 分钟报警。在报警策略方面，我们也摆脱了人工拍定阈值的方式，采用同环比阈值报警，提高了报警的准确性。

Goldeneye V1 的优势很明显，每个业务线单独开发逻辑，部署任务，业务逻辑清晰。不足也是比较明显，需要开发同学对 storm 编程比较熟悉。进行双 11、双 12 等大促活动时，需要针对各自的业务进行逻辑的调整，

资源扩容等工作。而且每个业务都需要重复开发外围的 TT/HBase 代码，抽象封装可以大大减少重复代码量。大盘需求得到满足，而更细粒度的监控需求需要定制开发，定制开发难度大、周期长。

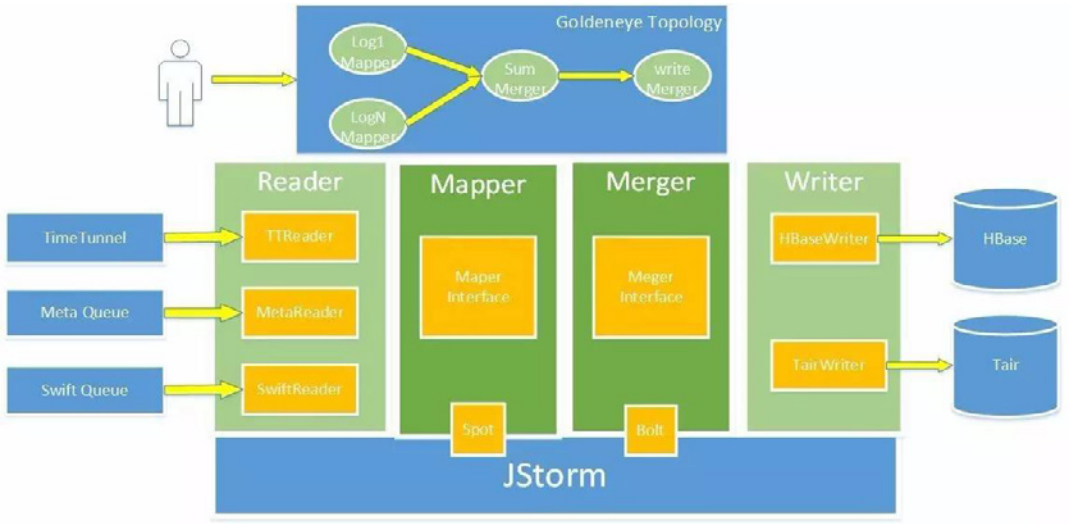
2、Goldeneye架构 V2版本——ARM框架

黄金眼大盘监控的效果很好，于是其他业务线也找到了各自的技术团队，生产了一台给自己业务线使用的 Goldeneye V1。那个时期各业务线终于过上了美好而又和谐的生活。（直到机器老化了，业务线同学觉得自己维护起来很费劲，而且业务整合的时候跨业务线数据不能打通，这就是 V2 架构的背景）。

技术方面，用户重复计算这些场景，对于 storm 的理解不够深入、实时任务性能不够，所以也急需要我们改善用户的编程习惯和编程体验。让用户只关注业务逻辑本身就可以。于是我们开始着手抽象实时计算框架了。我们在 storm 的原语上抽象了 ARM(Alimama Realtime Model) 是广告实时框架的简称，它是包含输入、输出、存储以及实时计算框架的组件集合。

ARM 提供如下组件：

- 输入输出：输入输出包括TT，HDFS，ODPS及MetaQ/Swift等。通过配置实例化组件，API访问。
- 存储：支持对LDB，MDB，HBase的访问，通过配置实例化组件，通



过API访问。

- mapmerge：mapmerge框架抽象出map-merge计算框架，帮助用户实现KV流的实时计算。

Mapper 阶段

用户只需要关心日志解析的代码，无需在关心与外部 TT/HBase 的链接配置等操作。核心代码就是将一条原始的 raw log, 解析成对应的结构。其中一条日志可以输出多条组合向后发送。</KEY,VALUE></KEY,VALUE>

Merger 阶段

用户只需要关心真正的内存 count 逻辑，关心用户代码即可，不需关心外部存储的配置等信息，在需要输出的时候直接调用 write(key,value) 即可。

在 V2 版本用户只需要关心解析包和业务的聚合逻辑，已经不需要在关心外围设备，以及集群相关的配置信息了，这样大大解放了用户的生产力，使得各个业务线可以在 ARM 平台上快速构建实时监控任务。不足之处是每个业务独立的存储结构，每次都需要独立开发一条 web 展示系统，同时考虑到大部分的需求是 Sum，分布式 sum 操作可以进一步抽象，而且我们在实践中又发现了类似 join、TopN 等需要抽象的实时语义，需要进一步的完善。

3、Goldeneye架构 V3版本——ARM算子层

这一阶段的业务特点就是黄金眼在阿里妈妈内部已经覆盖了所有的核心业务线。对于其他业务线的需求，需要给非技术同学提供更易用性的接入，为外部引流部门提供高性能的解决方案，为已经在系统上运行的部门提供快速、可配置的聚合逻辑。

颠覆之前的开发模式，真正走向平台化，在完整兼顾阿里妈妈内部需求的同时给集团内部的其他部门提供平台化服务。ARM 算子层在 ARM 框架 mapper/merger 接口之上，封装了实时 sum、实时 join 等多种计算模型。

Source 阶段

和 goldeneye V2 mapper 的相同之处在于，用户只需要关心具体的解

析逻辑。和 goldeney V2 mapper 不同的是，之前用户会把数据转换为结构，例如 key=a，key=b，key= c，需要对下游发送 3 条数据，而在新版本中用户仅需要发送一条的流 </KEY1,KEY2,KEY3,VALUE></KEY，VALUE>

aggregator 阶段

在这个阶段相比 goldeneye V2 有了本质的提高。用户不需要关心 cache buffer 等代码的配置，只需要在配置文件中书写类 SQL 的配置就可以。ARM 算子层会把这个 SQL 语句转换成对应的执行计划，映射到具体的计算模型上。例如：

```
Select key1,key2, ts, sum(val1),sum(val2)
From input_table
Where key1!=null and key2!=null
Group by key1,key2,ts
```

经过这几个版本的迭代，ARM 框架已经很趋于成熟，在 storm 平台之上，构建一套完整的分布式计算 sum，分布式 join，topn ，unique 等功能。同时随着框架的引进我们也接入了更多的业务，覆盖了阿里妈妈 90% 以上的业务线，同时我们还对外在阿里集团内部和多个部门合作共建，努力推广黄金眼监控系统。我们的优势其实很明显，我们在监控上做到准确无误，用户的接入效率也提高了 10 倍以上，而且用户在后期的监控需求变更上

也非常方便，易于维护。经过算子层的抽象，用户只需要关心解析内容和简单的 SQL 配置就可以完成实时聚合了，这大大提高了使用的易用性，接入效率提高 10 倍以上。同时在这个阶段我们对后台的算子层也进行了深入的优化，在性能生也有很大的提高，峰值流量 QPS 可以覆盖 300W/s，完美的度过了 2016 年双 11 的流量高峰，以及外部媒体流量引入的高峰。技术优化点在接下来详细阐述。

三、黄金眼计算存储技术难点

伴随着阿里妈妈业务的发展，日志数据量也在逐步增加。黄金眼系统实时处理的日均处理日志量超过 100T，峰值的 QPS 逐步提高。同时单条日志中需要解析的核心指标也在增加，使得一条日志会转变为多条需要统计的消息，进一步造成消息量的膨胀。内部对于监控需求也越来越细化，由原来简单的大盘粒度（时间维度）一维聚合统计，变化到多维聚合统计。这进一步促使黄金眼实时计算的计算量激增。面对这么大的输入流量和这么复杂的计算场景，我们在计算存储上做了以下方面的优化。

1、实时计算如何支撑巨大的流量压力

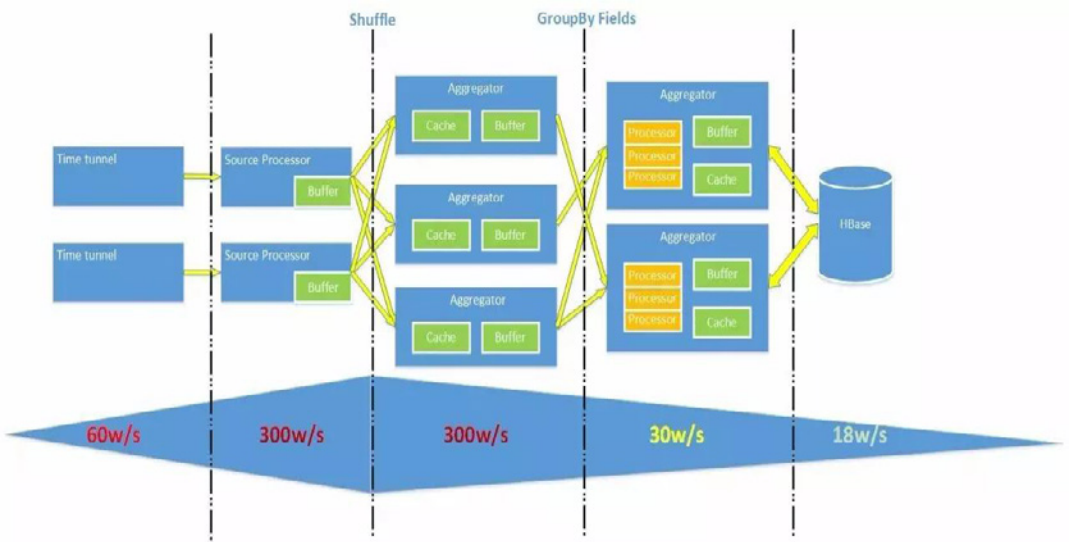
在内部系统中，我们观察到某个任务内部出现 300W/s 的高峰 QPS 的现象，因此我们采取以下几点方式逐步泄洪，将流量高峰平稳度过。

提升拓扑处理效率——流量分层

在离线计算中常常会有数据倾斜的情况，在实时计算中也会出现同样的情况。为了解决数据倾斜，我们在实时计算的 count 阶段采用了了多种防止倾斜的办法。

我们在设计聚合算子是采用了 2 级计算模式： 2 级模式的第一级采用 shuffle 的模式，平均分配到第一级的 shuffle aggregator（SA）。在 shuffle Aggregator（SA）内部中完成对 group by key 的内存聚合。聚合之后的 key 的数量已经降了 n 个数量级。然后再根据 group by 的 key 进行 filed group 分发到 group by Aggregator（GBA）保证 key 的分发是均匀的。

在 group by Aggregator（GBA）内部我们通过多个线程对数据进行读写，同时增加 read cache 和 write batch 进一步提高效率。



这样就解决了数据倾斜的问题，同时通过 key 的数量在内存逐步聚合，保证流量在逐级递减，使得黄金眼系统应对 300W/s 的流量得以实现。

提高后端读写效率——剔除无效读写

在线上系统运行过程中，我们观察到后端存储压力过大，为减少对存储访问压力我们做了一些优化工作。通过观察半结构化的日志数据，我们发现在 source parse 阶段处理输出的日志字段非常稀疏，这与业务访问特点有关。前端采集的日志会包含全部指标，但并不是全部指标都会有值被更新，单条日志中实际需要更新的内容只占一少部分，所以我们在 cache 中增加了对空值和 0 值的判断，去掉了大量无用读写存储的操作。

提升网络传输效率——消息压缩

面对海量的实时日志，我们首先考虑将数据处理的数据量进行了缩减。Storm 是一个基于消息的实时计算引擎，在 spout/bolt 之间传输都需要进行序列化，反序列化。我们采用了 Google 的 ProtoBuffer 方式进行序列化，将一条日志中的需要统计处理的核心字段保留并进行传输。对于核心的数据的字段我们尽量避免 String 类型，而是改为对应的数值类型定义，例如 timestamp=1435051454 如果用 string 存储要 10 个字节，

用 fix32 只用 4 个字节。根据字段的实际值把每个字段都设置了最合适的类型，这么调整一下后，实测 ProtoBuffer 序列化后的大小相对于之前减少了 30% 以上，效果明显。虽然输入的数据没有变化，但是经过瘦身后的消息在传输和计算的过程中节省了大量的网络带宽，为实时计算能力的提高打下了坚实的基础。

2、复杂应用场景如何保持schema通用性

存储系统的易用性将直接影响到上层应用使用，以及数据处理的流程，如何设计通用的存储系统是分布式系统中重要的一个环节，核心的思路是存储 schema 保持清晰简明。底层设计的越通用，上层才能构建出丰富的应用。黄金眼经历了三次 schema 结构的演变，是去繁从简的过程。下面介绍这三次 schema 的变化。

（1）第一版设计：业务驱动

第一版设计是每个业务按月建表，由程序定时触发按月建表，每张表的 schema 大致如下：

TableName：Code_Name_YYYYMMDD	
Metric + DimA:Val+ ..... + DimN:Val + Timestamp	ColumnFamily： f
	Qualifier：
	Value：count bytes

使用过程中发现 metric 存在数据倾斜问题，造成 HBase 的写 region 过程存在热点影响查询性能。这里的优化点是新建表的 region 范围按照上个月的 region 分布来创建，可以缓解写热点造成的影响。

（2）第二版设计：解决热点问题

第二版设计对第一版做了稍微改动，主要是为了解决热点问题，表的 schema 设计如下：

TableName：Code_Name_YYYYMMDD	
Md5（Metric + DimA:Val+ ..... + DimN:Val，2） + Metric + DimA:Val + ... + DimN:Val + Timestamp	ColumnFamily： f
	Qualifier：
	Value：count bytes

稳定运行了一段时间后，我们也在思考如何可以让黄金眼更高效的接入业务，于是我在存储层面重新设计时间序列框架 theia，Theia 是基于 HBase 存储时间序列数据的应用，提供多种 metric 数据采集方式，处理，存储，报警及展示功能的整体解决方案，黄金眼主要使用了 theia 的存储功能。

（3）第三版设计：通用 schema

第三版我们重新梳理了黄金眼的业务场景，以及对存储的访问模式，结合业务特点重新设计了 HBase 的 schema，取消了按月建表，所有业务可以共用同一份表，表存储分为 meta 表和 data 表，表结构大致如下：

meta 表

TableName：Code_Name_Meta	
Namespace + Metric	ColumnFamily：
	Qualifier：
	Value：
Namespace + Dim	ColumnFamily：
	Qualifier：
	Value：

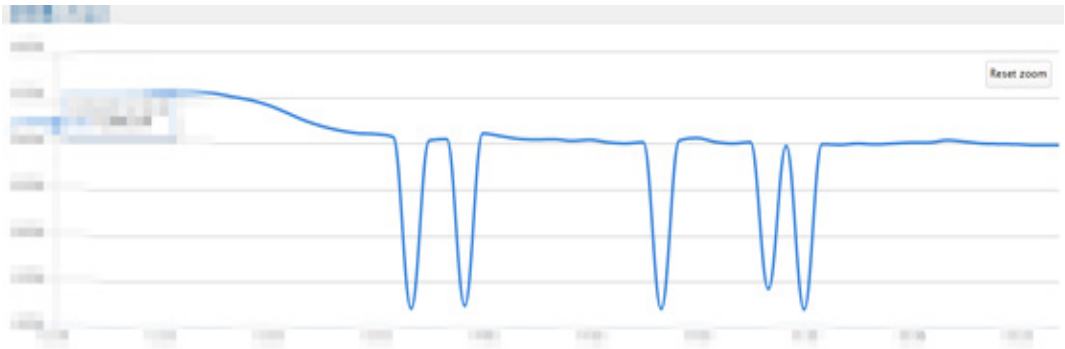
data 表

TableName：Code_Name_Data	
Md5（Namespace + Metric + DimA:Val+ ..... + DimN:Val）+ Timestamp	ColumnFamily： f
	Qualifier：Value 值类型（支持 long，double，ProtoBuffer），Metric Name（可选）
	Value：bytes

由于 theia 支持 long、bytes、pb 格式的 value 类型，所以 qualifier 中会标明 value 类型。第三版的设计参考的 opentsdb 的设计，同时避免了热点问题以及查询性能不稳定等问题。同时所有业务线使用同一份表，通过 namespace 区分业务线，简化了数据处理和访问的复杂度，同时也减少了运维成本。

3、一致性问题填坑——多线程get/put不一致

一致性问题一直是分布式系统中遇到的让人头疼的问题，在黄金眼中最终数据写入存储是通过类似 HBase 的 Increment 操作完成的，但由于使用 HBase 的 Increment 操作性能上较差需要堆大量存储资源，所以经过对比我们最终采用了 get/put 的方式来做累加操作。在黄金眼使用的初期，我们观察到监控的曲线图上曲线经常出现毛刺现象，如下所示：



通过与离线数据进行对比发现不一致情况确实存在。分析原因得知数据经过 mapper 处理过后会 shuffle 到 merger 的不同 writerQueue 中，由于 hash 规则使得相同的 key 被随机分配到了不同的 Writer Queue 中，key 在更新过程中需要从本地缓存或者 HBase 中获取数据的最新状态，导致相同 key 的更新操作会随机出现在多个 writer Queue 中。找到问题原因后我们调整了数据处理方式，保证相同 key 的状态更新由同一个 writer 处理，最终解决了数据不一致的问题。

经过上述的几点优化后，系统的整体性能得到了大幅提升。同时随着系统的稳定运行，我们还在探索新的计算框架，在 storm 原语之上构建 sum/join 等实时算子会带有一定的局限性，我们目前也在调研 Flink 相关的计算平台，统一实时、离线的变成框架，同时进一步提高我们的系统效率，为业务提供更优质的服务。

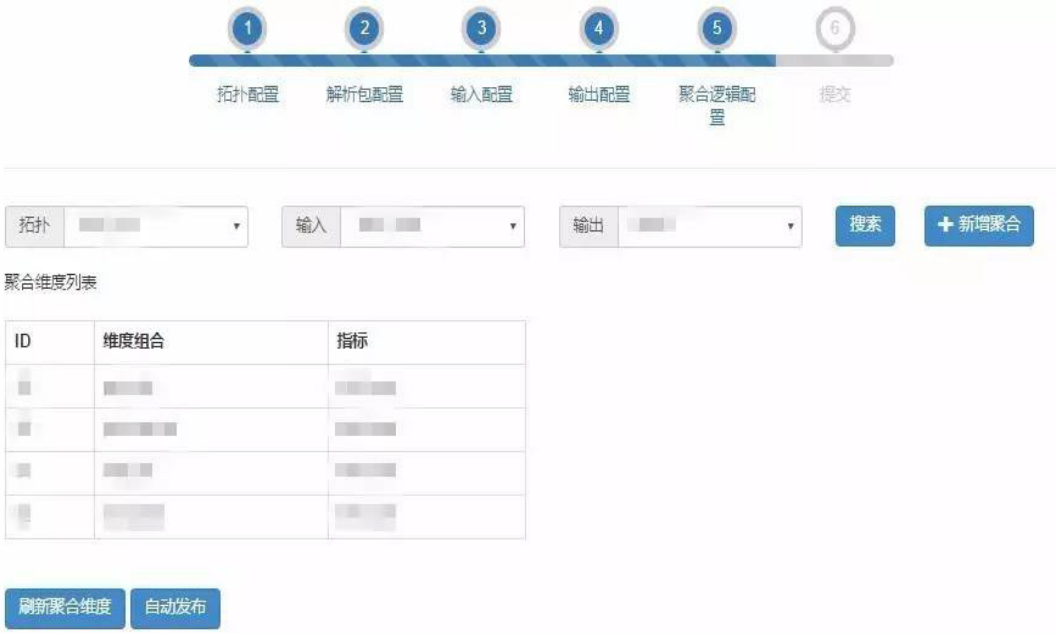
四、未来发展趋势

随着黄金眼系统的逐步完善平台化是大势所趋，我们已经完成了初步的平台化工作，后续还回进一步在自动化，智能化方向深挖。

1、自动化

1) 数据自动接入

未来的数据是自动接入的，每一个核心的实时数据回有一个监控报警的开关，用户只需要打开这个开关就可以自动的配置到黄金眼的监控平台中，节省开发量，用户只需要关心维度即可。尽管在 V3 的状态下用户已经可以只需要关注日志的 parse 和书写对应的 sql 就可以完成对应的实时 sum 了，对开发同学已经很简便了，但对很多运营同学提出了更高的的要求。而且运行同学的监控需求是非常普遍的，经常需要关注某几个 id 下的流量变化，所以如何让他们快速的使用我们的黄金眼系统便成为一个挑战，而且我们的接入效率基本也在开发测试 2-3 天左右，如果更加快速的开发，提高接入效率了，能否将接入效率提高到 1 天以内接入。 我们参考了开源的可视化的做法，后续将会逐步退出基于 web 拖拽方式的自动接入系统，方便所有用户的接入。 目前已经针对 V3 版本实现了 Web 的自动化配置接入（半自动），后续会逐步优化用户的使用习惯，实现完全的自动化接入。



2) 实时任务自动发布

当用户的任务自动配置完成后，只需要点击“自动发布”功能即可发

布到对应的 jstorm 集群上，实现一键的任务起停，彻底改变我们目前依赖 start/stop 脚本手工起停任务的方式。完全的把我们的 PE 同学解放出来。

3) 指标自动展示

用户在自动接入阶段已经配置好了所有的维度，指标等信息，这些信息会被记录到 mysql 中，用于后面自动展示。用户只需要配置对应的中文名称，具体含义，即可展示，做到即配即用。同时对于用户统计的 val1,val2 等指标我们定义为直接指标，而用户可以在系统内部方便的定义间接指标例如 val3=val1+val2,val4=val1/val2，做到灵活可用。

4) 指标自动报警

随着阈值模型的进一步完善，后续会逐步的开发自动报警功能。用户数据自动展示到页面后，只需要拖拽 2 根线选定作为报警的上限和下限，同时选择报警的方式是动态阈值或者是变点检测等预测算法产出阈值即可，后台会根据用户的选择直接配置生成报警规则，做到自动报警目前我们已经走在自动化，平台化的大路上。

2017 年初我们已经初步完成了第一版的设计，用户可以在我们的 web 页面上实现自动配置、半自动发布、自动展示、自动报警等功能，后续还会持续在交互设计上进行优化，为用户提供更好、更准确的监控服务。

2、智能化

1) 智能调整参数

每天的流量高峰，流量低谷会出现在不同时段，基本上测试好一定的配置参数可以很好的应对。但是如果经历季度的变化、出现大促等流量升高等情况都需要我们对拓扑的参数进行调整。所以后续会采集每天的流量 QPS 变化数据，根据 QPS 变化数据的预警值调整拓扑的配置参数，进行流量自适应。

2) 智能算法模型

监控报警，阈值预测是我们黄金眼平台的核心功能，实时计算和存储

系统为它提供了非常好的平台支撑。未来我们会在架构上不断的优化智能算法模型，不断挖掘计算存储的能力，为业务的快速发展保驾护航。

作者介绍

马国强，资深研发工程师，阿里妈妈全景业务监控（黄金眼）核心研发。2012 年中科院计算所硕士毕业，毕业后进入人人网从事广告平台的实时系统、人人网离线计算平台的研发和运维工作。2014 年加入阿里妈妈从事实时计算框架、大规模数据处理等基础架构研发。在分布式计算、实时计算方面有丰富实践经验，目前致力于阿里妈妈全景业务监控（黄金眼）的计算框架设计与优化。

许琦，资深研发工程师，2014 年 4 月加入阿里从妈妈基础服务部门，从事广告数据存储平台研发与性能优化相关工作，2012 年在新浪数据库平台部门负责 NoSQL 数据服务平台研发及存储性能优化。

GMTC

全球移动技术大会2017

首届以大前端为主题的技术大会

2017.6.09–10　北京·国际会议中心

重量级嘉宾齐助阵

《移动项目快速持续交付的工程化实践》

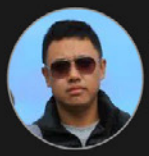
林永坚

REA Group
Mobile Developer

《Instagram Direct：高效可靠的数据端到端传输》

李 晨

Instagram iOS 高级工程师



《利用CNN实现无需联网的智能图像处理》

李永会

百度 图像搜索客户端工程师

《QQ移动页面框架优化实践》

陈志兴

腾讯 高级工程师

《微信SQLite数据库损坏恢复实践》

何俊伟

微信 Android高级工程师

《手机天猫面向业务的界面解决方案-Tangram》

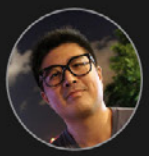
高嘉峻（伯灵）

天猫 无线技术专家

《豌豆荚的反作弊技术架构与设计》

胡 强

阿里 应用分发平台
Android端负责人



《H5互动的正确打开方式》

金 鑫（渚薰）

阿里巴巴 前端专家

《ReactNative框架在京东无线端的实践》

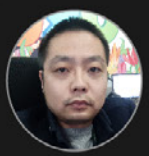
沈 晨

京东 专家架构师

《滴滴出行iOS端瘦身实践》

戴 铭

滴滴 出行技术专家



《移动虚拟化：360分身大师那些事》

王云鹏

奇虎360
分身大师项目技术负责人

《携程无线持续交付平台工程实践》

赵辛贵

携程 高级研发经理

……

5月12日前购票享2880元（团购更优惠）

主办方 Geekbang. 极客邦科技　InfoQ

票务咨询：18618231445 / alfred@infoq.com

2017 年会是 Serverless 爆发之年吗？

作者 麦克周

前言

中小型公司，尤其是互联网行业的创业公司，本身并没有太多的技术人员，如果设计系统时需要考虑诸多的技术问题，例如 Web 应用服务器如何配置、数据库如何配置、消息服务中间件如何搭建等等，那对于他们来说人员成本、系统成本会很高，Serverless 架构的出现，让这种情况可能可以大幅度改善。

初识 Serverless ?

在目前主流云计算 IaaS（Infrastructure-as-a-Service，基础设施即服务）和 PaaS（Platform-as-a-Service，平台即服务）中，开发人员进行业务开发时，仍然需要关心很多和服务器相关的服务端开发工作，比如缓存、消息服务、Web 应用服务器、数据库，以及对服务器进行性能优化，

还需要考虑存储和计算资源，考虑负载均衡和横向扩展能力，考虑服务器容灾稳定性等非专业逻辑的开发。这些服务器的运维和开发知识、经验极大地限制了开发者进行业务开发的效率。设想一下，如果开发者直接租用服务或者开发服务而无须关注如何在服务器中运行部署服务，是否可以极大地提升开发效率和产品质量？这种去服务器而直接使用服务的架构，我们称之为 Serverless 架构（无服务器架构）。

Serverless 架构的问世

其实，最初“无服务器”意在帮助开发者摆脱运行后端应用程序所需的服务器设备的设置和管理工作。这项技术的目标并不是为了实现真正意义上的“无服务器”，而是指由第三方供应商负责后端基础结构的维护，以服务的方式为开发者提供所需功能，例如数据库、消息，以及身份验证等。这种服务基础结构通常可以叫做后端即服务（Backend-as-a-Service，BaaS），或移动后端即服务（MobileBackend-as-a-service，MBaaS）。

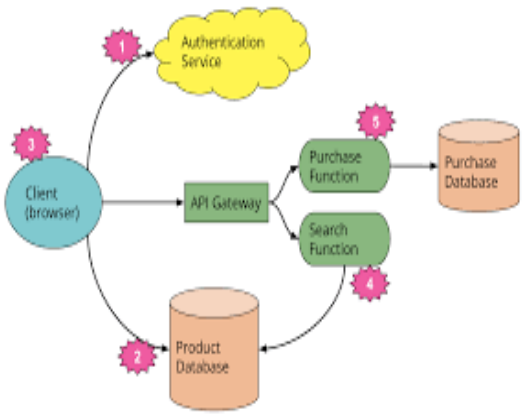
现在，无服务器架构是指大量依赖第三方服务（也叫做后端即服务，即“BaaS”）或暂存容器中运行的自定义代码（函数即服务，即“FaaS”）的应用程序，函数是无服务器架构中抽象语言运行时的最小单位，在这种架构中，我们并不看重运行一个函数需要多少 CPU 或 RAM 或任何其他资源，而是更看重运行函数所需的时间，我们也只为这些函数的运行时间付费。无服务器架构中函数可以多种方式触发，如定期运行函数的定时器、HTTP 请求或某些相关服务中的某个事件。

Serverless 案例

以带有服务功能逻辑的传统面向客户端的三层应用为例（一个典型的电子商务应用网站）。一般来说包含客户端、服务端程序、数据库，服务端用 Java 开发完成，客户端用 JavaScript。

采用这种架构，服务端需要实现诸多系统逻辑，例如认证、页面导航、搜索、交易等都需要在服务端完成。如果采用 Serverless 架构来对该应

用进行改造，则架构如图所示：



Serverless 架构相比于传统面向客户端的三层应用架构，有以下几方面的差异：

- 删除认证逻辑，用第三方BaaS服务替代；
- 使用另外一个BaaS，允许客户端直接访问架构与第三方（例如AWS Dynamo）上面的数句子库。通过这种方式提供给客户更安全的访问数据库模式；
- 前两点中包含着很重要的第三点，也就是以前运行在服务端的逻辑转移到客户端中，例如跟踪用户访问。客户端则慢慢转化为单页面应用。
- 计算敏感或者需要访问大量数据的功能，例如搜索这类应用，我们不需要运行一个专用服务，而是通过FaaS模块，通过API Gateway对HTTP访问提供响应。这样可以使得客户端和服务端都从同一个数据库中读取相关数据。由于原始服务使用Java开发，AWS Lambda（FaaS提供者）支持Java功能，因此可以直接从服务端将代码移植到搜索功能，而不用重写代码。
- 最后，可以将其他功能用另外一个FaaS功能取代，因为安全原因放在服务端还不如在客户端重新实现，当然前端还是API Gateway。

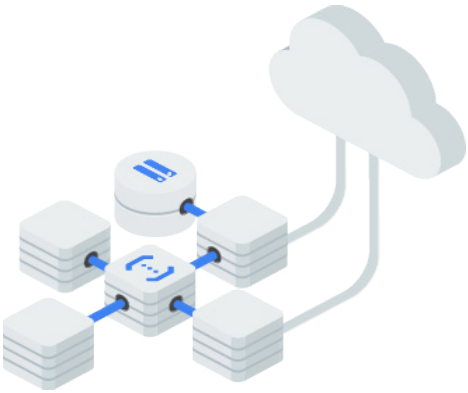
常见的 Serverless 框架介绍

Amazon的Lambda产品

2014年11月14日，AWS发布了AWS Lambda。AWS Lambda是市面上最早，也是最为成熟的Serverless框架之一。该服务最迟支持Node.js，现在也支持Java和Python。它与Alexa Skills Kit（软件开发工具包）紧密集成，亚马逊提供交互式控制台和命令行工具，以便上传和管理代码片段。

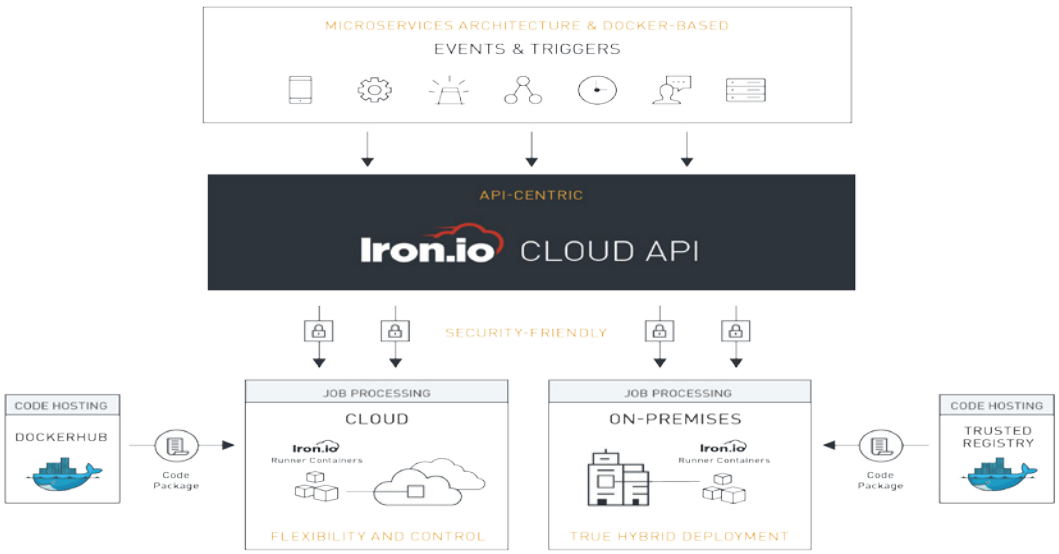
Google Cloud Functions

Google是为服务架构的最前沿公司，除了推动Kubernetes，Google还投资了Cloud Functions，该架构可以在其公共云基础设施上运行。



Iron.io

Iron.io最初是为企业级应用提供微服务。Iron.io是用Go语言编写的，用于处理高并发、高性能计算服务，并已经集成Docker服务，提供一种完整的微服务平台。



IBM OpenWhisk

2016 年 2 月的 InterConnect 大会，IBM 发布了 OpenWhisk，这种事件驱动型开源计算平台可以用来替代 AWS Lambda。OpenWhisk 平台让广大开发人员能够迅速构建微服务，从而可以响应诸多事件，比如鼠标点击或收到来自传感器的数据，并执行代码。事件发生后，代码会自动执行。

Serverless Framework

Serverless Framework 是无服务器应用框架和生态系统，旨在简化开发和部署 AWS Lambda 应用程序的工作。Serverless Framework 作为 Node.js NPM 模块提供，填补了 AWS Lambda 存在的许多缺口。它提供了多个样本模板，可以迅速启动 AWS Lambda 开发。

Azure WebJobs

Azure Web 的应用功能，可以与 Web、API 应用相同的上下文中运行程序或脚本。可以上传并运行可执行文件，例如 cmd、bat、exe、psl 等等。WebJobs 提供 SDK 用于简化针对 Web 作业可以执行的常见任务，例如图像处理、队列处理、RSS 聚合、文件维护，以及发送电子邮件等等。

Serverless 架构原则

按需使用计算服务执行代码

Serverless 架构是 SOA 概念的自然延伸。在 Serverless 架构中，所有自定义代码作为孤立的、独立的、细粒度的函数来编写和执行，这些函数在 AWS Lambda 之类的无状态计算服务中运行。开发人员可以编写函数，执行常见的任务。在比较复杂的情况下，开发人员可以构建更复杂的管道，编排多个函数调用。

编写单一用途的无状态函数

单单负责处理某一项任务的函数很容易测试，并稳定运行。通过以一种松散编排的方式将函数和服务组合起来，能够构建易于理解、易于管理

的复杂后端系统。

为 lambda 等计算服务编写的代码应该以无状态方式进行构建，这样会让无状态功能很强大，让平台得以迅速扩展，处理数量不断变化的请求或者事件。

设计基于推送的、事件驱动的管道

可以构建满足任何用途的服务器架构。系统可以一开始就构建成无服务器，也可以逐步设计现有的单体型应用程序，以便充分发挥这种架构的优势。最灵活、最强大的无服务器设计是事件驱动型的。

构建事件驱动的、基于推送的系统常常有利于降低成本和系统复杂性，但是要注意，并不是任何情况下都是适当的或者容易实现的。

创建更强大的前端

由于 Lambda 的定价基于请求数量、执行时间段以及分配的内存量，所以代码执行需要越快越好。数据签名的令牌让前端可以与不同的服务直接通信。相比之下，传统系统中所有通信经由后端服务器来实现。让前端与服务进行通信有助于减少创建环节、尽快获得所需的资源。

与第三方服务集成

如果第三方服务能提供价值，并减少自定义代码，那么自然它们就很有价值。开发人员可以通过引入第三方服务来减少自己实现各种业务逻辑的需要，可以减少小型公司的开发成本，避免价格、性能、可用性等要素上的劣势。

未来趋势

随着移动和物联网应用蓬勃发展，伴随着面向服务架构（SOA）以及微服务架构（MSA）的盛行，造就了 Serverless 架构平台的迅猛发展。在 Serverless 架构中，开发者无须考虑服务器的问题，计算资源作为服务而不是服务器的概念出现，这样开发者只需要关注面向客户的客户端业务程序开发，后台服务由第三方服务公司完全或者部分提供，开发者调用相

关的服务即可。Serverless 是一种构建和管理基于微服务架构的完整流程，允许我们在服务部署级别而不是服务器部署级别来管理应用部署，甚至可以管理某个具体功能或端口的部署，这就能让开发者快速迭代，更快速地交付软件。

这种新兴的云计算服务交付模式为开发人员和管理人员带了很多好处。它提供了合适的灵活性和控制性级别，因而在 IaaS 和 PaaS 之间找到了一条中间道路。由于服务器端几乎没有什么要管理的，Serverless 架构正在彻底改变软件开发和部署流程，比如推动了 NoOps 模式的发展。

深度学习框架：2016 年的大盘点

作者 刘志勇

引言

刚刚过去的 2016 年，回顾这一年，深度学习无疑是 2016 年最热的词。包括 Google、Amazon、Facebook、Microsoft 等各大巨头都在不遗余力地推进深度学习的研发和应用。

与前几代人工智能不同，应用深度学习能力的人工智能是一项重大突破，机器开始可以模仿人类的神经系统进行有效学习并且进步神速，也就有了最近的神秘高手“Master”横扫中日韩顶级棋手，碾压人类的神话，它不是别人，正是 2016 年战胜李世石的 AlphaGo 的升级版！

深度学习的概念由加拿大多伦多大学教授 Geoffrey Hinton 等人于 2006 年提出，它本质上是一种神经网络算法，算法训练时可以不用人工

干预，因此它也属于一种无监督式机器学习算法。

可以毫不夸张的说，深度学习正在重塑 Google、Facebook、Microsoft 和 Amazon。

BEEVA Labs 数据分析师 Ricardo Guerrero Gomez-Ol 在他的博客上发表了一篇博文，盘点了目前最流行的深度学习框架。他在博文中表示，他写此文的初衷是，他常常听到人们谈论深度学习时，总是问：“我应该从哪里开始呢？”“我听说 TensorFlow 是最流行的，对吧？”“Caffe 很常用，但是我觉得它学起来有点困难。”

因为 Ricardo 所在的 BEEVA 实验室，经常和深度学习的许多库打交道，所以他想分享有趣的发现和感想，帮助那些刚进入深度学习这一迷人世界的人们。

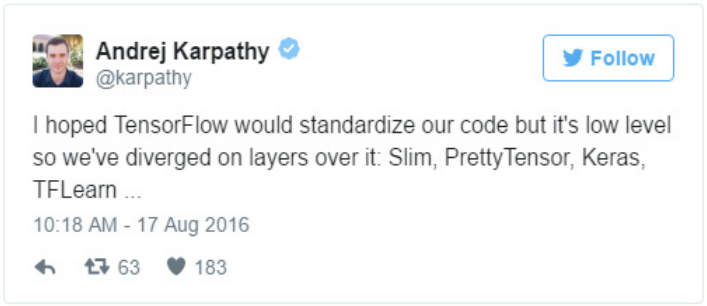
InfoQ 整理、结合了 Ricardo 关于深度学习框架的盘点，写成此文，以飨广大有志于深度学习领域的读者们。

Tensorflow

TensorFlow 是 Google 开源的一款深度学习工具，使用 C++ 语言开发，上层提供 Python API。在开源之后，在工业界和学术界引起了极大的震动，因为 TensorFlow 曾经是著名的 Google Brain 计划中的一部分，Google Brain 项目的成功曾经吸引了众多科学家和研究人员往深度学习这个“坑”里面跳，这也是当今深度学习如此繁荣的重要原因。

TensorFlow 在官网被定义为“用于机器智能的开源软件库”，但 Ricardo 认为下面的定义更为准确：“TensorFlow™是使用数据流图进行数值计算的开源软件库。”此处，TensorFlow 并未归类为“深度学习框架”，而是与 Theano 一起，归到“图编译器（Graph compilers）”的类别。

在 Ricardo 完成 Udacity 的深度学习课程后，他对 TensorFlow 的印象是，它一个非常好的框架，但是它非常底层。用 TensorFlow 的话，有很多代码要编写，你必须重新一遍又一遍的发明轮子。不止他有这样的抱怨，其他人亦如此，比如下图所示：



Andrej Karpathy 在 Twitter 发推文抱怨 TensorFlow。Andrej 是 OpenAI 的科学家，斯坦福大学的计算机科学博士。

几个月前，Ricardo 参加了“Google Experts Summit: TensorFlow, Machine Learning for everyone, with Sergio Guadarrama”。Sergio 是开发 Tensorflow 的一名工程师，在会上，他没有展示 Tensorflow，而是展示了一个运行在 TensorFlow 上的更高层的库 tf.contrib。Ricardo 的印象是，他们内部已经意识到，如果想让更多的人使用 Tensorflow，他们需要通过在更高的抽象层上创建一些层来简化使用。

Tensorflow 支持 Python 和 C ++，允许在 CPU、GPU，甚至支持使用 gRPC 进行水平扩展进行计算分布。

综上所述：Tensorflow 非常好，但你必须知道好在哪里。如果你不想以手动编程和重新发明轮子来完成大部分事情，你可以使用更容易的库。

Theano

Theano是老牌、稳定的库之一。它是深度学习开源工具的鼻祖，由蒙

特利尔理工学院时间开发于2008年并将其开源，框架使用Python语言开发。它是深度学习库的发轫，许多在学术界和工业界有影响力的深度学习框架都构建在Theano之上，并逐步形成了自身的生态系统，这其中就包含了著名的Keras、Lasagne和Blocks。

Theano 是底层库，遵循 Tensorflow 风格。因此不适合深度学习，而更合适数值计算优化。它支持自动函数梯度计算，它有 Python 接口，集成了 Numpy，使得这个库从一开始就成为通用深度学习最常用的库之一。

今天，它依然健壮可用，但事实上，由于没有多 GPU 支持和水平扩展，加之 TensorFlow 的如天花乱坠的大肆宣传之下（它们都是针对同一个领域），结果，Theano 逐渐被世人遗忘了。

Keras

Ricardo 表示他很喜欢 Keras，因为它的句法相当清晰，文档也非常好的（尽管相对较新），它还支持 Ricardo 所熟知的 Python 语言。它如此易用，它的指令、函数和每个模块直接的连接方式都可以直观地了解。

Keras 是一个非常高层的库，工作在 Theano 或 Tensorflow（可配置）之上。此外，Keras 强调极简主义，你可以用寥寥可数的几行代码来构建神经网络。在这里，您可以看到一个 Keras 代码示例，与在 Tensorflow 中实现相同功能所需的代码相比较。

Lasagne

Lasagne是一个工作在Theano之上的库。它的任务是将深度学习算法的复杂计算予以简单地抽象化，并提供一个更友好的Python接口。这是一个老牌的库，长久以来，它是一个具备高扩展性的工具。在Ricardo看来，它的发展速度跟不上Keras。它们适用的领域相同，但是，Keras有更好的、更完善的文档。

Caffe

Caffe是最老的框架之一，比老牌还要老牌。 Caffe 是加州大学伯克利分校视觉与学习中心（Berkeley Vision and Learning Center ，BVLC)贡献出来的一套深度学习工具，使用C/C++开发，上层提供Python API。Caffe同样也在走分布式路线，例如著名的Caffe On Spark项目。

Ricardo 认为，它有非常好的特点，但也有一些小缺点。最初，它不是一个通用的框架，只专注于计算机视觉，但它确实很好。在实验室的实验中，CaffeNet 架构的训练时间在 Caffe 比在 Keras（使用 Theano 后端）少 5 倍。缺点是它缺乏灵活。如果你想引入新的改进，你需要在 C ++ 和 CUDA 编程。如果你要做较小的改进，你可以使用它的 Python 或 Matlab 接口来达到。

另外它的文档很贫乏，你需要大量时间检查代码才能理解它。

它最大缺点之一是安装方式。它有很多大量的依赖包才能解决，Ricardo 曾经安装过两次，他表示这一过程痛苦不堪。

但要注意的是，它并非一无是处。它作为投入生产的计算机视觉系统的工具，是无可争议的领导者。它非常健壮、非常快速。Ricardo 建议使用 Keras 进行实验和测试，然后迁移到 Caffe 进行生产。

DSSTNE

DSSTNE（Deep Scalable Sparse Tensor Network Engine，DSSTNE）是Amazon开源的一个非常酷的框架，由C++语言实现。但它经常被忽视。为什么？因为，撇开其他因素不谈，它并不是为一般用途设计的。DSSTNE 只做一件事，但它做得很好：推荐系统。正如它的官网所言，它不是作为研究用途，也不是用于测试想法，而是为了用于生产的框架。

Ricardo 测试 DSSTNE 后得到的印象是，它是一个非常快的工具，能得到一个非常好的结果（平均精度 mAP 很高）。为了达到这种速度，它使用了 GPU，然而这也是它的一个不利之处：不同于本文提到的其他框架或库，它不允许你在 CPU 和 GPU 切换，这点对一些尝试可能有用，但他们在 DSSTNE 试图这样做的时候，被告知不允许。

Ricardo 认为目前 DSSTNE 不算是一个成熟的项目，它过于像“黑盒子”。为了了解它的工作原理和运行机制，不得不去看它的源代码，发现了很多重要的待办事项（//TODO）。他们还发现，在互联网上没有足够的教程，也没有什么人做相关实验。他的意见是，最好等待 4 个月，看看 DSSTNE 的最新版本。这是一个非常有趣的项目，它只是需要一点成熟的时间。

顺便说一句，DSSTNE 不需要编程技能。与 DSSTNE 的交互都是通过终端中的命令完成的。

Torch

Torch是Facebook和Twitter主推的一个特别知名的深度学习框架，Facebook Reseach和DeepMind所使用的框架，正是Torch（DeepMind被Google收购之后才转向TensorFlow）。出于性能的考虑，它使用了一种比较小众的编程语言Lua，目前在音频、图像及视频处理方面有着大量的应用。

在目前深度学习大部分以 Python 为编程语言的大环境之下，一个以 Lua 为编程语言的框架只有更多的劣势，而不是优势。Ricardo 没有 Lua 的使用经验，他表示，如果他要用 Torch 的话，就必须先学习 Lua 语言才能使用 Torch。就他个人来说，更倾向于熟悉的 Python、Matlab 或者 C++ 来实现。

mxnet

mxnet是支持大多数编程语言的库之一，它支持Python、R、C++、Julia等编程语言。Ricardo觉得使用R语言的人们会特别喜欢mxnet，因为直到现在，在深度学习的编程语言领域中，Python是卫冕之王。

Ricardo 以前并没有过多关注 mxnet，直到 Amazon AWS 宣布将 mxnet 作为其深度学习 AMI 中的参考库时，提到了它巨大的水平扩展能力，他才开始关注。这就是为什么 mxnet 会出现在我们 2017 年的 BEEVA 技术测试名单之中。

Ricardo 表示他对多 GPU 的扩展能力有点怀疑，但仍然很愿意去了解实验更多的细节。但目前还是对 mxnet 的能力抱有怀疑的态度。

DL4J

DL4J，全名是Deep Learning for Java。正如其名，它支持Java。Ricardo说，他之所以能接触到这个库，是因为它的文档。当时，他在寻找限制波尔兹曼机（Restricted Boltzman Machines）、自编码器（Autoencoders），在DL4J找到这两个文档，文档写得很清楚，有理论，也有代码示例。Ricardo表示D4LJ的文档真的是一个艺术品，其他库的文档应该向它学习。

DL4J 背后的公司 Skymind 意识到，虽然在深度学习世界中，Python 是王，但大部分程序员都是 Java 起步的，因此，DL4J 兼容 JVM，也适用于 Java、Clojure 和 Scala。随着 Scala 的潮起潮落，它也被很多有前途的初创公司使用。

Skymind 曾发布过一篇文章“DL4J vs. Torch vs. Theano vs. Caffe vs. TensorFlow”，对这些主流的深度学习框架的优劣势进行了详细的分析比较。

顺便说一句，Skymind 有一个非常活跃的 Twitter 帐户，他们发布新的科学论文、示例和教程。非常推荐去看看。

Cognitive Toolkit

认知工具包（Cognitive Toolkit），就是之前被大家所熟知的缩略名 CNTK，但最近刚更改为现在这个名字，可能利用 Microsoft 认知服务（Microsoft Cognitive services）的影响力。在发布的基准测试中，它似乎是非常强大的工具，支持垂直和水平推移。

到目前为止，认知工具包似乎不太流行。关于这个库，还没有看到有很多相关的博客、网络示例，或者在 Kaggle 里的相关评论。Ricardo 表示这看起来有点奇怪，因为这是一个背靠微软研究的框架，特别强调自己

的推移能力。而且这个研究团队在语音识别上打破了世界纪录并逼近了人类水平。

你可以在他们的项目 Wiki 中的示例，了解到认知工具包在 Python 的语法和 Keras 非常相似。

结论

Ricardo 的结论是，如果你想进入深度学习的领域，必须首先就要学习 Python。尽管这一领域支持其他很多语言，但 Python 是应用最广泛也最简单的一个。至于为什么非 Python 莫属？它运行速度太慢了。因为大多数库都是用符号式语言方法，而不是命令式语言方法。也就是说，并不是逐行执行你的指令，而是根据你给出的所有指令，生成一个计算图（computing graph）。这个图在内部被优化、编译成可执行的 C++ 代码。这样你就可以享受世界上最好的特点：Python 的开发速度和 C++ 的执行速度。

关于深度学习的讨论越来越火爆了。但是人们并不愿意为了算法训练耗费大量时间，因此，多 GPU 支持、多机器的水平扩展甚至硬件定制开始占上上风，你不要考虑 CPU，它的效能远低于 GPU。

Ricardo 建议，如果是初学者，**就用 Keras；如果已经入门，也可以用它**。

深度学习作为 AI 领域的一个重要分支，我们可以预见，随着以后大数据和深度学习技术的不断发展，今后越来越难的问题，将会被深度学习算法成功解决。我们也非常期待深度学习算法可应用于商业产品中，就像过去 10 年中人脸识别器被整合到消费级相机中那样。

CNUTCon2017
全球运维技术大会

2017年9月10日–11日 | 上海·光大会展中心

全球运维技术大会
全新起航

5月27日前购票立减1440元
（团购更优惠）

联系我们

会务咨询
15600448113
molly@infoq.com

商务咨询
13426412029
yolanda@infoq.com

售票咨询
18504256269
hedy.hu@geekbang.org

议题申请
13240810004
bella@infoq.com

主办方 Geekbang. 极客邦科技 | InfoQ

架构师 月刊 2017年4月

本期主要内容：2017 谷歌云大会，一口气发布 100+ 消息；Uber 优步分布式追踪技术再度精进；历经 8 年双 11 流量洗礼，淘宝开放平台如何攻克技术难关？；处理微服务架构的内部架构和外部架构；浅析 MySQL JDBC 连接配置上的两个误区；Zendesk 的 TensorFlow 产品部署经验；人工智能永恒的春天已经到来，你准备好了吗？

解读2016

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

顶尖技术团队访谈录 第八季

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

架构师特刊 用户画像实践

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