

# 架构师

ARCHITECT



## 热点 | Hot

WiFi爆惊天漏洞！

Docker官方将支持Kubernetes

## 推荐文章 | Article

微博大V老师木的机器学习水平怎么样

智能音箱背后有何门道？

## 观点 | Opinion

做技术选型时，要注意些什么？

微信小程序的技术，也许你想错了



# CONTENTS / 目录

## 热点 | Hot

WiFi 爆惊天漏洞！KRACK 可攻陷所有 WiFi 网络

Docker 官方将支持 Kubernetes，容器编排大战宣告结束

## 推荐文章 | Article

微博技术大 V 老师木的机器学习水平怎么样？

智能音箱背后有何门道？

## 观点 | Opinion

做技术选型时，要注意些什么？

王跃：关于微信小程序的技术，也许你想错了

## 理论派 | Theory

体系化认识 RPC



## 架构师 2017 年 11 月刊

本期主编 谢然

提供反馈 feedback@cn.infoq.com

流程编辑 丁晓昀

商务合作 sales@cn.infoq.com

发行人 霍泰稳

内容合作 editors@cn.infoq.com

# 助力人工智能落地

2018.01.13 – 01.14 · 北京国际会议中心

近年来，获得投资界助力的AI市场发展迅猛，人工智能正在渗透到各行各业。过了“尝鲜期”，大多数企业开始发力AI落地：

- ① 如何用机器学习实现2亿月活跃用户？
- ② 如何基于人工智能为用户精准推荐他们最喜欢的商品？
- ③ 如何通过深度学习网络结构使准确度提升 11% ？
- ④ 如何优化风控模型来解决智能金融带来的安全问题？
- ⑤ 如何通过机器学习等 AI 技术来提高运营效率？
- ⑥ 如何解决VR直播中高码率带来的“三高”问题？

为了帮助企业摆脱“落地难”的困扰，InfoQ中国为大家梳理了整个AI产业生态链，并瞄准全球顶尖AI落地案例策划了AICon全球人工智能技术大会。大会将精选30+国内外AI技术专家共享他们的落地痛点及填坑经验。

## 演讲嘉宾



颜水成  
360人工智能研究院  
院长及首席科学家



洪亮劼  
Etsy  
数据科学主管



张浩  
饿了么  
技术副总裁



尹大朏  
摩拜单车  
首席科学家



裴少芳  
iTutorGroup  
大数据部总监



张瑞  
知乎  
机器学习团队负责人



杨骥  
国美在线  
大数据中心副总监



胡时伟  
第四范式  
首席架构师



胡南炜  
微博  
机器学习计算和  
服务平台负责人



吴甘沙  
驭势科技  
联合创始人&CEO



陈伟  
搜狗  
语音交互技术中心  
研发总监



张重阳  
微信小程序  
商业技术高级研究员

8折

限时购票，立减720元

团 购 享 受 更 多 优 惠



# 卷首语

## 写给程序员的话

作者 Rancher Labs CEO 梁胜

纵观当今各行各业，我们可能很难再找到一个像程序员这样的人生了。在云计算，移动互联网，以及人工智能这样的新技术的发展浪潮不断催生出新的商业机会的今天，IT 行业对程序员似乎有无止无尽的需求！作为一个程序员，想到职业规划，一方面我们觉得有太多的机会，另一方面我们也会感到迷茫，甚至畏惧！如果你在一个大公司工作，如何跟上新技术发展的步伐？如何避免在同事得到升职时，自己陷入死胡同，无成长空间？如果你已经决定自主创业，你如何在成百上千的创业竞争对手中脱颖而出？这些竞争对手中的很多人或许拥有比你更多的资金，更有经验。在新技术不断颠覆的今天，我们能否对未来的 10 年，20 年乃至 30 年的职业发展做出一些计划，让我们面对未来的行业发展游刃有余呢？

说编程已经不是一个好的职业了，因为程序员只能吃青春饭。在我看来，这种观点大错特错。实际上，绝对没有任何其它职业比编程序更能体现个人创造力和技能的价值。程序员像艺术家。编程的确是一个辛苦的劳动过程，但不重复。世上不会有两个人写出完全一样的代码。正如一个

艺术家一样，程序员能够持续很长的职业生涯。我就认识很多程序员，他们虽然已经四五十岁以上仍然保持高效。程序员的薪酬比艺术家好得多，因为编程的工作为社会创造了直接的经济价值。

说到职业发展，很多人认为要成功只要找对机会。对这类人来说，职业规划就像买彩票一样。他们不断从一个项目换到另一个项目，从一个公司跳槽到另一个公司并乐此不疲。他们不关心自己在做什么，只关心是否站对了队。事实上，这些人并不能找到财富自由，而是得到一份有污点的简历和败坏的声誉。而在我们的 IT 行业，一个人的声誉决定了一切！在整个职业生涯中，声誉的积累与技能和财富的积累同样重要。当你有良好的声誉时，机会自然会来找上门来。

因此，仅仅关注新的技术发展和新的商业机会是不够的。无论你在手上执行什么工作任务，你都应该在努力交付一流工作结果，以此来积累你的声誉。风险投资家 Ann Miura-Ko 简单地将这一做法概括为“成为世界一流”。在 Ann 的思想中，“世界一流”适用于任何一项工作任务，从简单的用复印机复印一份文件到编写代码，做一个 PPT 演讲，或大到做出一个产品。据我观察，这世界上平庸的东西太多。如果你能够尽力把每一件小事都做得最好，长期积累下去，您将成为最耀眼的明星。

许多人认为编程仅仅是一个重复性的工作。但实际上作画，做音乐也一样都是重复性的工作。是什么让一幅画或一段曲成为世界一流的呢？无疑是激情、创造力以及对完美的追求。而事实上伟大的程序员能在最普通的任务中找到激情把程序写得最好。在 Google 诞生之前，网页搜索的问题被业界公认为是毫无意义而且已解决完毕的问题。Eric Yuan 曾经在 Webex 任工程副总裁，而他坚信 Webex 发明的网络会议系统仍然可以改进，继而他创立了当下全球最成功的 Zoom 视频会议服务。像乔布斯和马斯克这样的传奇人物，他们的成功也以激情，创造力和追求完美做为基础。对我们普通人来说，我们可以从每天的工作做起，从小事做起：

1. 当你在做一个设计或实现，即使只是某一个简单的功能点，你能不能把这个功能做成全世界最好的？不要在乎事情小，越小越容

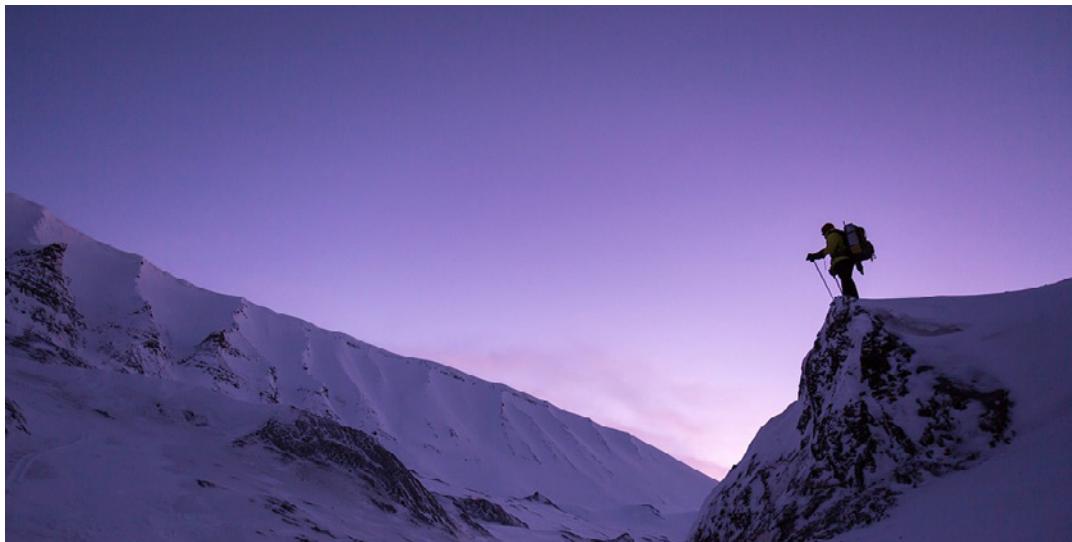
易冒尖！

2. 你多半不能一次就把设计做得最好。就像绘画和音乐一样，你要反复尝试，不断提高，总有一天会被认可！
3. 开源软件是一个向全世界展示你自己的工作成果的机会。在弄懂你为什么喜欢某些开源项目后，你是否也可以创造一个有名的开源项目？
4. 如果你有机会要做一个PPT演讲，千万不要照搬照抄别人的PPT。尽量去理解你的听众的喜好，然后从头开始自己写一个PPT。你的目标应该是：如何把这个PPT做成全世界最好的？
5. 如果有机会再做同一个演讲，你能不能把它做得更好一点？绝对不要把一样的PPT讲2次，每次都要有提高！

在大机会上门之前，你能为自己的职业规划最好的准备就是把你每天从事的本职工作做得尽善尽美。如果保持这种工作态度，你的努力终将得到你整个团队和整个公司的认可。如果你是从事一个开源项目的话，那你将会得到全世界的认可！以这种态度工作，你将会找到脱颖而出的成就感，你会迅速变的更优秀，更快得到升职。如果你哪天决定自主创业，你的声誉能让你很快地得到投资，很快地吸引人才。最重要的是，你会明白怎样才能创建一家世界一流的公司！

# WiFi 爆惊天漏洞！ KRACK 可攻陷所有 WiFi 网络

作者 Mathy Vanhoef 译者 运河凭



## 内容简介

我们发现 WPA2 当中存在一项严重安全漏洞。WPA2 为目前使用范围最广的 Wi-Fi 网络保护协议。身处攻击目标周边的恶意人士能够利用密钥重装攻击 (Key Reinstallation Attacks, KRACK) 利用此类安全漏洞。

具体来讲，攻击者能够使用这种新型攻击方法读取此前被认为是安全加密的信息，进而窃取各类敏感信息，其中包括信用卡号码、密码、聊天信息、电子邮件以及图片等等。

此类攻击指向全部现代受保护 Wi-Fi 网络。根据具体网络配置的不同，攻击者还能够进一步实现注入或者操纵数据。举例来说，攻击者能够将勒索软件或者其他恶意软件注入至目标网站当中。

这类安全缺陷存在于 Wi-Fi 标准本身，而非特定某些产品或者实现方案中。因此，即使是得到正确部署的 WPA2 同样有可能受到影响。为了预防攻击，用户必须在安全更新发布时立即对受影响产品进行修复。

需要注意的是，如果您的设备支持 Wi-Fi，则很有可能会受到影响。通过我们的初步研究，已经发现 Android、Linux、苹果、Windows、OpenBSD、联发科以及 Linksys 等厂商的产品都会受到某些变种攻击的影响。

与特定产品相关的更多细节信息，请参阅 CERT/CC 数据库或者联系您的产品供应商。

此项攻击研究将于本届计算机与通信安全（CSS）大会以及黑帽欧洲大会上正式公布。

## 演示

作为 POC，我们对一部 Android 智能手机执行了一次 KRACK。在本次演示中，攻击者有能力对受害者传输的全部数据进行解密。对于攻击方而言，这一攻击方式易于实现，因为我们的密钥重装攻击对于 Linux 以及 Android 6.0 或者更高版本系统拥有强大的破坏性。

这主要是由于 Android 以及 Linux 会在攻击者的引导下（重新）安装一条全零加密密钥（详见后文）。在攻击其他设备时，虽然解密全部数据包难度极大，但攻击者仍然有能力解密相当一部分数据包。

无论如何，以下演示将强调攻击者在针对受保护 Wi-Fi 网络执行密钥重装攻击时所能够获得的信息类型：

### [观看视频](#)

我们的攻击并不仅限于恢复登录凭证（即电子邮件地址以及密码）。总体而言，受害主机所传输的任何数据或者信息都能够进行解密。另外，根据具体使用设备以及网络设置的不同，攻击者亦有可能解密被发送至受害者处的数据（例如网站内容）。

尽管网站或者应用可能利用 HTTPS 作为额外的保护层，但我们需要

提出警告，这类额外保护手段在相当一部分情况下仍有可能被绕过。举例来说，此前即有案例证明 HTTPS 会在非浏览器软件、苹果 iOS 与 OS X、Android 应用、银行应用甚至是 VPN 应用当中被绕过。

## 细节信息

我们的主要攻击是针对 WPA2 协议的四次握手。这一握手活动会在客户端希望添加到受保护 Wi-Fi 网络当中时执行，并被用于确认客户端与接入点皆拥有正确的凭证（例如网络的预共享密码）。

与此同时，四次握手还会协商一个新的加密密钥，此密钥将被用于对所有后续流量进行加密。目前，所有具备现代保护机制的 Wi-Fi 网络皆使用四次握手机制，这意味着此类网络都将受到我们发现的攻击手段的某些变种的影响。

举例来说，此类攻击可能将黑手指向个人与企业 Wi-Fi 网络、较为陈旧的 WPA 与最新的 WPA2 标准，甚至攻击仅使用 AES 的网络。我们对 WPA2 采取的所有攻击手段皆应用到一种新型技术，即密钥重新安装攻击（简称 KRACK）：

## 密钥重装攻击：高级描述

在密钥重装攻击当中，攻击者会诱导受害者重新安装已经被使用过的密钥。具体实现方法是操纵并重播密码握手消息。当受害者重新安装密钥时，增量发送分组号（即随机数）以及接收分组号（即重播计数器）等相关参数将被重置为初始值。

从本质上来讲，为了保证安全性，每条密钥只能安装并使用一次。遗憾的是，我们发现 WPA2 协议当中并不包含这一强制要求。通过操纵加密握手过程，我们将能够在实践当中利用这一致命缺陷。

## 密钥重装攻击：针对四次握手的具体示例

正如相关研究论文当中指出的，密钥重装攻击背后的思路可以总结如

下。当某客户端加入一个网络时，会执行四次握手协议以协商获取一个新的加密密钥。

它将在接收到四次握手中的第 3 个消息后安装此密钥。一旦该密钥安装完成，它将被用于配合加密协议对普通数据帧进行加密。然而，由于消息内容可能丢失或者丢包，因此如果接入点（AP）没有收到适当的响应作为确认，则将重新发送消息 3。

每当接收到这条消息时，客户端都会重新安装同样的加密密钥，并因此重置增量发送分组号（随机数）且接收加密协议所使用的重播计算器。在演示中可以看到，攻击者能够通过收集并重播四次握手中的消息 3 来重演上述重传操作。

如此一来，即可强制重复使用随机数，意味着该加密协议即遭攻击影响——具体包括重播数据包、加密数据包以及 / 或者伪造数据包。这一技术亦可被用于攻击组密钥、PeerKey、TDLS 以及快速 BSS 切换握手。

## 实际影响

在我们看来，影响最为广泛且实际危害最大的攻击方式正是针对四次握手的密钥重装攻击。之所以得出这样的结论，主要基于我们观察到的两大结果。首先，在我们自己的研究中，我们发现它会导致大多数客户端遭受影响。

第二，攻击者可以利用此种攻击方式解密客户端发送的数据包，从而拦截密码或者 Cookie 等敏感信息。分组解密同样存在可能性，这是因为密钥重装攻击会导致传输随机数（有时亦被称为分组号或者初始化向量）被重置为零。

如此一来，同一个加密密钥可能使用以往已经使用过的随机数值。这反过来会导致全部 WPA2 加密协议皆在加密数据包时重复使用密钥流。此后，该密钥流可被用于利用同一随机数进行消息解密。

当不存在已知内容时，数据包解密很难实现，不过事实证明某些特定情况下攻击者仍可达成这一目标（例如仍可解密英文文本）。实际上，找

出包含书籍内容的数据包并不是什么难题，因此可以假定任意受影响数据包皆可能因此遭到解密。

这种解密数据包的能力会被用于解密 TCP SYN 数据包。这意味着攻击者将能够获取一条连接当中的 TCP 序列号，同时劫持 TCP 连接。在这种情况下，即使使用 WPA2 保护协议，攻击者仍然能够针对开放 Wi-Fi 网络执行一类常见的攻击手段：向未加密 HTTP 连接当中注入恶意数据。举例来说，攻击者可以利用这种方式将勒索软件或者恶意软件注入至受害者访问的网站当中。

如果受害者使用的是 WPA-TKIP 或者 GCMP 加密协议，而非 AES-CCMP，更将面临灾难性的影响。在使用此类加密协议时，随机数复用不仅允许攻击者进行解密，甚至允许其对数据包进行伪造以及注入。

另外，由于 GCMP 在双工通信的两侧中皆使用同样的认证密钥，那么一旦因随机数复用而引发密钥恢复，则影响将极为巨大。需要注意的是，GCMP 支持目前被广泛称为 Wireless Gigabit（简称 WiGig），且预计将在未来几年内逐步得到普及。

哪个方向的数据包可实现解密（乃至伪造），具体取决于受到攻击的握手过程。简单来讲，当攻击四次握手时，我们可以对由客户端发送的数据包进行解密（及伪造）。当攻击快速 BSS 切换（简称 FT）握手时，我们可以对客户端接收到的数据包进行解密（及伪造）。最后，我们的大多数攻击方式亦可实现单播、广播与多播。欲了解更多细节信息，请参阅我们研究论文中的第六章内容。

需要强调的是，我们的攻击无法恢复 Wi-Fi 网络自身使用的密码，也无法恢复在四次握手期间进行协商的（任意部分）新加密密钥。

## Android 与 Linux

我们的攻击对于 wpa\_supplicant 2.4 以及更高版本具有极大威胁——这是一套在 Linux 当中得到广泛使用的 Wi-Fi 客户端。在这里，该客户端将安装一个全零加密密钥——而非重新安装真正的密钥。

这项安全漏洞似乎是由 Wi-Fi 标准当中的一条注释所造成，其提到会在首次进行安装之后，将加密密钥从内存中清除。现在当客户端接收到四次握手所重播的消息 3 时，其会重新安装已经被清除的加密密钥，但实际安装的却是一条全零密钥。

由于 Android 使用 wpa\_supplicant，因此 Android 6.0 以及更高版本亦存在此项安全缺陷。这使得攻击者能够轻松拦截并操纵往来于这些 Linux 与 Android 设备之间的网络流量。需要注意的是，目前有 41% 的 Android 设备会受到我们攻击手段的某类变种的影响。

## 相关 CVE 码

以下常见漏洞与披露（简称 CVE）码，用于追踪哪些产品会受到此次密钥重装攻击中特定安装手段的影响：

- CVE-2017-13077：在四次握手当中重新安装成对加密密钥（PTK-TK）。
- CVE-2017-13078：在四次握手当中重新安装组密钥（GTK）。
- CVE-2017-13079：在四次握手当中重新安装完整性组密钥（IGTK）。
- CVE-2017-13080：在组密钥握手当中重新安装组密钥（GTK）。
- CVE-2017-13081：在组密钥握手当中重新安装完整性组密钥（IGTK）。
- CVE-2017-13082：接收一条重发快速 BSS 切换（简称 FT）重新关联请求，并在处理过程中对成对加密密钥（PTK-TK）进行重新安装。
- CVE-2017-13084：在 PeerKey 握手当中重新安装 STK 密钥。
- CVE-2017-13086：在 TDLS 握手当中重新安装隧道直连设置（简称 TDLS）PeerKey（TPK）密钥。
- CVE-2017-13087：当处理无线网络管理（简称 WNM）睡眠模式响应帧时，重新安装组密钥（GTK）。

- CVE-2017-13088：当处理处理无线网络管理（简称 WNM）睡眠模式响应帧时，重新安装完整性组密钥（IGTK）。

需要注意的是，每个 CVE 码皆代表着密钥重装攻击的一种特定实例。这意味着每条 CVE ID 都描述了特定的协议漏洞，且每一条 CVE ID 都可能影响到多家供应商。您亦可点击此处参阅 CERT/CC 的 VU#228519 安全漏洞说明以了解更多已知受影响产品的细节信息。

## 论文

我们针对此类攻击行为所编撰的论文题为《密钥重装攻击：WPA2 中的强制随机数复用》，其将于 2017 年 11 月 1 日星期三召开的计算机与通信安全（简称 CCS）大会上正式发表。

尽管这篇文章目前已经正式公布，但我们于 2017 年 5 月 19 日即将其提交以进行审查。在此之后，我们仅对内容作出了细微修改。因此，文中的调查结果已经完成数月之久。

与此同时，我们还发现了更为轻松易行的技术手段，用以在四次握手阶段执行密钥重装攻击。复用这种新型攻击技术，我们无需对四次握手中的消息 3 进行加密重传即可达成目标。具体来讲，这意味着对 MacOS 与 OpenBSD 的攻击方式在难度上已经远低于论文当中提到的水平。

我们还希望对以下附录及勘误内容作出强调说明：

### **附录：wpa\_supplicant v2.6 与 Android 6.0+**

Linux 的 wpa\_supplicant v2.6 同样会受到在四次握手期间安装全零加密密钥攻击的影响。这一问题最初由 John A. Van Boxtel 所发现。如此一来，全部高于 6.0 的 Android 版本皆将受到影响，即其会在攻击者的诱导下安装一条全零加密密钥。这种新型攻击手段为注入一条伪造的消息 1，其中使用与原始消息 1 相同的随机数，而后将重发消息 3 转发至受害者处。

### **附录：其他易受攻击的握手协议**

在论文中提到的初步研究之后，我们还发现 TDLS 握手与 WNM 睡眠模式响应帧同样会受到密钥重新安装攻击的影响。

## 内容勘误

在用于描述攻击第三阶段的图九当中，由攻击者发送给认证方的帧应该为 ReassoReq，而非 ReassoResp。

## 工具

我们已经制作了脚本来检测四次握手、组密钥握手或者快速 BSS 切换握手的具体实现是否会受到密钥重装攻击的影响。我们目前正在对脚本中的使用说明进行整理，并将在结束后立即进行发布。

我们还制作了一份概念验证脚本，其可在特定 Android 以及 Linux 设备之上实现全零密钥（重新）安装。我们在演示视频当中使用的也正是这套脚本。当各方已经对相关设备进行合理更新（而我们也对发布代码库进行筹备）之后，这套脚本即将与各位见面。

需要强调的是，我们的概念验证脚本的实际可靠性取决于受害者环境与真实网络间的近似程度。如果受害者环境与真实网络非常相似，则脚本可能失败——这是因为尽管会被迫使用网络上的其他 Wi-Fi 信道，受害者仍将始终直接与真实网络保持通信。

## 常见问题

### 我们现在是否需要 WPA3？

不，幸运的是我们可以通过向下兼容的方式进行修复。这意味着安装补丁的客户端仍将与未经修复的接入点通信，反之亦然。换句话说，安装过补丁的客户端或者接入点将发送与以往完全相同的握手消息，且时间点亦完全一致。然而，安装更新将确保一条密钥仅被安装一次，从而避免受到攻击影响。因此再次强调，一旦安装更新发布，请立即为您的设备安装。

## **我是否应该修改 Wi-Fi 密码？**

变更您 Wi-Fi 网络的密码并不能避免（或者缓解）此类攻击。因此，您不需要对 WiFi 网络的密码进行更新。相反，您应当确保全部设备皆进行更新，并应更新您的路由器固件。在路由器更新完毕后，您应选择性地变更 WiFi 密码以作为额外预防手段。

## **我正在使用纯 AES WPA2 模式。这是否仍然面临安全风险？**

是的，这一网络配置同样存在安全隐患。我们的攻击手段同时针对 WPA1 与 WPA2，会影响到个人及企业网络，且将波及您所使用的任何加密套件（包括 WPA-TKIP、AES-CCMP 以及 GCMP）。因此每位用户都应更新相关设备以预防这类攻击！

## **您在网站当中使用了“我们”这一词汇。“我们”指的是谁？**

我之所以使用“我们”，是因为论文当中也使用了同样的表述。具体来讲，所有工作皆是由我一人完成，即 Mathy Vanhoef。我优秀的上级以荣誉作者的身份被纳入研究论文，并为我的工作提供了出色的指导。但所有具体工作皆由我个人进行。因此，学术论文作者名单并不体现具体分工情况。

## **我的设备是否会受到影响？**

有可能。任何使用 WiFi 的设备都有可能面临这一安全风险。请联系您的供应商以了解更多细节信息。

## **如果我的路由器没有提供安全更新，该怎么办？**

我们的主要攻击手段面向四次握手，且不会利用接入点——而是主要指向客户端。因此您的路由器可能并不需要进行安装更新。我们强烈建议您与供应商联系以获取更多细节信息。总体来讲，您可以通过禁用客户端功能（例如用于中继器模式等）并禁用 802.11r（快速漫游）以减轻针对路由器与接入点的攻击风险。对于普通家庭用户，您应当优先更新各类客户端，例如笔记本电脑以及智能手机。

## **您是如何发现这些安全漏洞的？**

在筹备另一篇论文的终版时，我反复检查了关于 OpenBSD 执行四次

握手时的一些说明。从某种意义上讲，我当时的心态比较放松，因为当时的任务只剩下完善论文，而非抓紧调整代码。但在这一过程中，我再次审查了已经阅读了不知多少次的代码，以避免存在某些纰漏。

就在这时，一条对于 `toic_set_key` 的调用引起了我的注意。此项函数会在处理四次握手中的消息 3 时被调用，其负责向驱动程序安装成对密钥。在面对该行代码时，我想到“哈，我很好奇如果该函数被调用两次，会引发怎样的结果。”当时，我（正确地）猜测到两次调用可能会重置与该密钥相关联的随机数。

而且由于消息 3 可由接入点进行重发，因此确实可能出现两次调用的情况。“最后在这里作出备注。这一函数确实有可能被调用两次。但让我们先把这篇论文完成……”数周之后，前一篇论文已经彻底完成，其他杂事也忙得差不多了，接下来我开始更为深入地考量这个想法。而接下来的事，相信大家都知道了。

### **四次握手在数学层面上被证明是安全的，您的攻击为何能够实现？**

简单来讲，四次握手的正式实现方式并不能确保密钥仅被安装一次。相反，其仅能确保协商密钥始终处于保密状态，且握手消息不可被伪造。

要更具体地解答这个问题，则需要引用研究论文中的相关表述：我们的攻击并不会破坏四次握手在正式分析当中得到证明的安全属性。具体而言，这些证据表明协商加密密钥始终处于私有状态，且客户端与接入点身份亦可得到确认。我们的攻击并不会泄露加密密钥。

另外，虽然使用 TKIP 或者 GCMP 能够伪造正常数据帧，但攻击者仍无法伪造握手信息，因此无法在握手期间冒充客户端或者接入点。不过问题在于，证明本身并不会对密钥安装机制进行建模。换句话来讲，正式模型并没有定义应该于何时安装协商密钥。在实际使用当中，这意味着相同的密钥可进行多次安装，从而重置加密协议（例如 WPA-TKIP 或者 AES-CCMP）当中使用的随机数与重播计数器。

### **论文中的某些攻击方式似乎非常困难**

我们已经采取后续完善工作，希望让我们的攻击手段（特别是针对

MacOS 以及 OpenBSD 的攻击) 更为普遍且易于执行。因此, 尽管我们同意文章中提到的一部分攻击方法缺乏现实意义, 但请相信我, 密钥重新安装攻击在实践当中确实有可能遭到利用。

### **是否已经有人开始对这一漏洞加以实际利用?**

我们无法确定这一漏洞是否已经被他人(或者正在被他人)所利用。具体来讲, 关键是重装攻击实际上能够自发发生, 且其中并不涉及真正的攻击者! 可能的场景为: 背景干扰因素使得握手过程中的最后一条信息发生丢失, 从而导致前一条信息重发。当处理这条重发信息时, 密钥被重新安装, 并致使随机数如真实攻击般被重复使用。

### **我是否应该利用使用 WEP, 直到我的设备完成补丁安装?**

不! 请继续使用 WPA2。

### **Wi-Fi 标准是否会进行更新以解决这个问题?**

似乎有协议规定, Wi-Fi 标准应当进行更新以有效避免受到我们攻击手段的影响。这些更新可能将以向下兼容方式发布, 从而覆盖其他早期 WPA2 实现方案。不过最终 Wi-Fi 标准是否或者将如何更新, 仍然有待时间给出答案。

### **Wi-Fi 联盟是否也在解决这些安全漏洞?**

这里要为各位不熟悉 Wi-Fi 的朋友们讲解一下: Wi-Fi 联盟是一个负责证明 Wi-Fi 设备符合某些互操作性标准的组织。除此之外, 其还负责确保来自不同供应商的 Wi-Fi 产品能够顺利地协同工作。

Wi-Fi 联盟已经有计划帮助解决已发现的各类 WPA2 安全漏洞。总结来讲, 该组织将:

- 要求在全球认证实验室网络当中测试此项安全漏洞。
- 提供一款安全漏洞检测工具以供各 WiFi 联盟成员使用(这款工具以我们的检测工具为基础, 用于检测目标设备是否易受到一些已知的密钥重装攻击手段的影响)。
- 向各设备供应商广泛发布与此项安全漏洞相关的细节信息, 包括补救措施。此外, 鼓励各供应商与其解决方案供应商合作, 从而

快速整合任何必要的修复补丁。

- 向用户强调重要性，确保用户已经安装由设备制造商提供的安全更新。

### **您为何在演示视频中使用 match.com 作为示例？**

用户会在 match.com 这类网站上共享大量个人信息。因此本示例的目的在于强调攻击者能够获取的各类敏感信息，同时希望示例受众能够更清楚地意识到这一攻击手段带来的（个人）影响。我们还希望这一示例能够帮助人们了解约会网站可能收集的用户信息类型。

### **如何避免此类 bug 的出现？**

我们需要对协议实现方案作出更为严格的审查。这亦要求来自学术界的帮助与协同支持！在其他研究员的协助下，我们希望组织起更多研讨活动以改进并验证各类安全协议实现方案的正确性。

### **为何选择 krackattacks.com 这样一个域名？**

首先，我得强调 KRACK 是个拼接词汇，分别来自 key reinstallation attack 这几个部分。不过为了凑个韵脚，最终就选择了这样一个网站名称。

### **您有没有因为这个 bug 获得赏金？**

我还没有申请任何 bug 赏金，当然就没有拿到过了。

### **与其他针对 WPA2 的攻击相比，这种攻击方式有何特点？**

这是有史以来第一种不需要依靠密码猜测的 WPA2 协议攻击手段。事实上，其他针对 WPA2 网络的攻击方法主要针对的是与之相关的其他技术，例如 Wi-Fi 受保护设置（简称 WPS），或者 WPA-TKIP 等较为陈旧的标准。换句话来说，目前还没有哪种攻击方法专门针对四次握手或者 WPA2 协议当中的密码套件。相比之下，我们的密钥重新安装攻击专门针对四次握手（以及其他握手），这更加强调了 WPA2 协议本身所存在的安全隐患。

### **其他协议是否也会受到密钥重装攻击的影响？**

我们预计其他协议的某些实现方案也可能受到类似攻击的影响。因此，最好是能够将这类攻击纳入安全协议实现方案的审计工作当中。不过我们

认为其他协议标准应该不会受到同一攻击手法的影响（或者我们至少希望不会）。然而，对其他协议开展针对性审计仍然很有必要！

### **您是否制作了分辨率更高的 Logo 版本？**

是的。这里要向帮助我制作 Logo 的朋友表示衷心感谢！

### **您是什么时候向供应商通知这一安全漏洞的？**

我们在 2017 年 7 月 14 日左右向已经进行过实际测试的产品相关供应商发出了通知。在与这些供应商进行联络之后，我们意识到我们所发现的问题拥有多么可怕的广泛影响（直到那时，我才真正相信这是一项协议缺陷，而非错误实现方法）。到这里，我们决定由 CERT/CC 协助披露这些安全漏洞。反过来，CERT/CC 又于 2017 年 8 月 28 日向各供应商发出了广泛通告。

### **为什么 OpenBSD 会在通告之前悄悄发布修复补丁？**

OpenBSD 早在 2017 年 7 月 15 日就已经得到了安全漏洞通知，这早于 CERT/CC 得到消息的时间。很快，Theo de Raadt 就作出了回复并要求尽快作出相关反应：“在开放世界，如果有人编写了 diff 并要求在一个月内给出回应，这显然会令人非常沮丧。”需要强调的是，我已经为 OpenBSD 编写了一份建议 diff，并表示将在同年 8 月底进行初步公开披露。这时我作出了让步，允许他们悄悄对漏洞作出修复。事后看来，这不是个明智的决定，因为可能会有人通过研究他们发布的补丁发现这一安全漏洞。为了避免今后出现这一问题，OpenBSD 现在表示能够接受发出时间与公开披露时间之间间隔更短的漏洞通知。

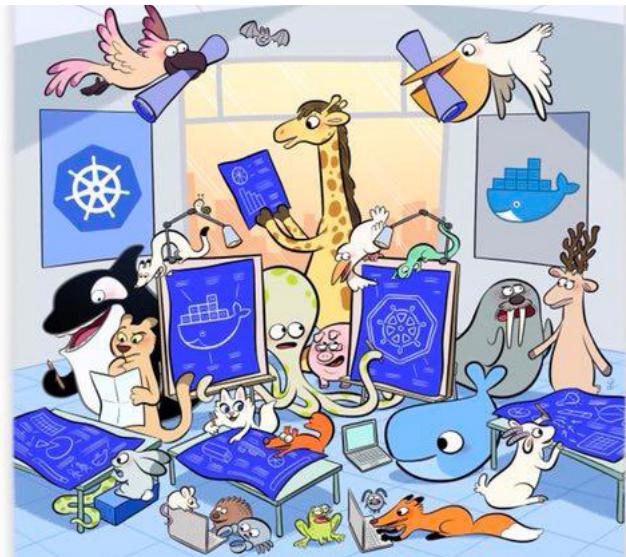
### **那么您是否还期待着找到更多其他 Wi-Fi 漏洞？**

“我认为这一切才刚刚开始。”——《光环 1》，士官长

# Docker 官方将支持 Kubernetes，容器编排大战宣告结束

作者 谢然，雨多田光

WE ARE  
ONE BIG  
COMMUNITY



10 月 17 日，Docker 在丹麦哥本哈根举行的 DockerCon 大会上宣布，将扩大其 Docker 平台并选择积极拥抱容器编排对手 Kubernetes。这意味着 Docker 客户及开发人员将可以选择同时使用 Kubernetes 与 DockerSwarm 进行容器工作负载的编排。

Docker 的创始人 Solomon Hykes 在大会上介绍，对于即将推出的 Docker 平台新版本，开发人员将能够在其工作站中的 Kubernetes 上直接进行生产应用程序的构建与测试。而运营人员则能够从 Docker 企业版中获得各种帮助，具体包括多租户安全保护，镜像扫描以及基于角色的访问控制等，同时配合 Kubernetes 或者 Swarm 在生产环境中实现应用运行。



Solomon Hykes 在大会上表示，今后在选择容器集群管理技术时现有的 Docker 开发人员不必学习新的 Kubernetes 工具，下一个版本的 Docker 将内置完整的 Kubernetes 发行版本，开发人员将能够一直使用 Docker 工具。

Docker 的理念为“Build, Ship and Run Any App, Anywhere”，通过容器和镜像的特性让 DevOps 变得容易，但 Docker 的前景，更在于支持分布式、服务化设计，实现一系列可独立开发、独立部署和独立扩展的服务组合，以保证业务的灵活性和稳定性。

Docker 容器被称为容器运行时的事实标准，而在容器编排上，Kubernetes、Mesos 和来自 Docker 官方的 DockerSwarm 一直以来处于竞争状态，但来自 Google 公司的 Kubernetes 以其高效、简便、高水平的可移植性等优势占领了绝大部分市场，而如今 Docker 官方宣布将拥抱这样一位竞争对手，看起来 Kubernetes 俨然赢得了编排框架市场的胜利。

## 专家观点

TalkingData 大数据及云计算工程师 [宋净超](#) 对此为读者带来了一番解读：

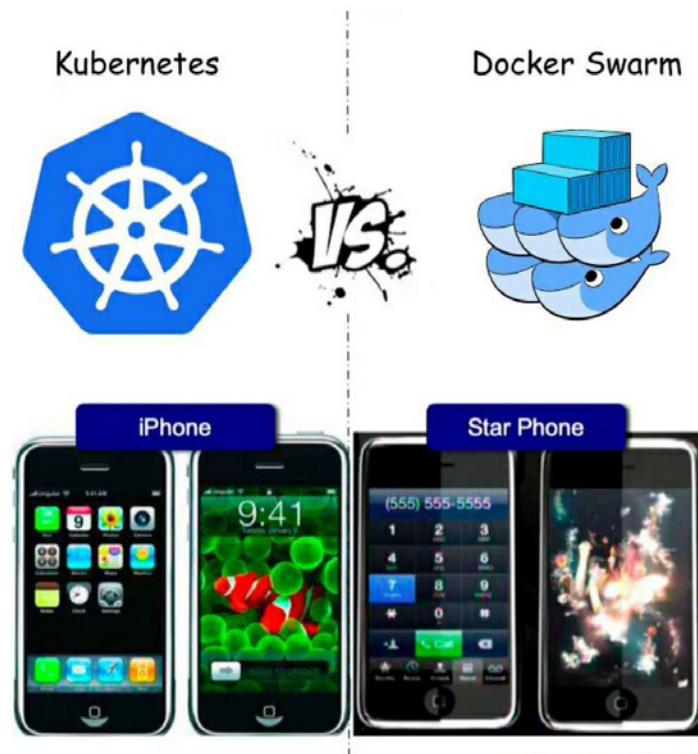
我觉得 Kubernetes 的眼光不止于容器编排，Docker 作为 Cloud

Native 生态中的最基础的 Runtime，之后可能会被其他的 Container Engine 替代，到时候 Docker 也只不过是 Runtime 的一个选择而已。

而 Kubernetes 所在的 CNCF 是为了解决企业上云的系列问题，从 Runtime 到部署、监控、分布式追踪、网络等等，我认为接下来它自己也会去构建一个相应的云原生生态。那到时候是否又是另一场大战，我们拭目以待。

针对这个事件，通俗一点来讲，Docker 相当于一部功能机，可以满足用户的一般需求。但是随着移动互联网的到来，大家需要更多的功能，原来的功能机已经无法满足需求了，这时候智能机，比如 iPhone 开始爆发了，这就是 Kubernetes 问世的一个环境。

但是功能机也不甘示弱，他们有了 MTK，可以低成本去做功能啊！可以低成本满足用户 80% 的需求。这也就是指这一次 Docker 官方说的，它将去支持 Kubernetes。但就是那 20% 的非功能性需求决定了用户体验。而 Kubernetes 正是完全有着占有用户这 20% 需求的能力，所以一直以来它作为容器编排的实际市场领跑者。



再从一个具体的点上去做个比喻就是，Docker 这个机子它本来只能支持单卡模式，但是现在它支持双卡了，它将自己本来的主卡，也就是 DockerSwarm 放到了副卡的位置上去了，而现在的主卡很明显就是 Kubernetes。但是你看 iPhone，这业界超一流的标准，它只支持单卡模式，那么未来 Docker 这部功能机，它如果想要向着这种一流去做，它现在的这个双卡战略会走得远吗？将来是否会直接将 DockerSwarm 这一副卡给去掉？

这可能是一个稍显不恰当的比喻，并不是贬低 Docker 而抬升 Kubernetes，毕竟先有了 Docker 生态，有了容器化之后才有了 Kubernetes，这里只是为了让读者切身感受，帮助理解。

# 技术大 V 老师木：软件平台是深度学习计算力突破的关键

作者 吴少杰 Tina 陈思



在知乎上，“老师木的机器学习水平怎么样”的问题，被浏览了 3.7 万次。虽然关注者众，却不少评论他线下实际“为人低调”。

同时有人称他是“微博大 V 老师木”，没错，作为一个技术人员，他的微博粉丝有 5.8 万。老师木说这个影响力“还太小”。

有人关心他为什么做得好好的要从微软亚洲研究院离职，更有人关心老师木为啥要创业。

老师木，真名袁进辉。读书时成绩一直优异，本科后保送清华大学直博生，师从人工智能领域张钹院士。期间多篇论文在国际顶级会议上发表，在竞争激烈的国际技术评测（TRECVID）中连续多年名列第一。博士后出

站后，于 2011 年入职网易有道。2012 年作为早期成员加入 360 搜索创业团队，一年之后，产品上线成为国内市场份额第二的搜索引擎。2013 年，加入微软亚洲研究院（MSRA），主要从事大规模机器学习平台的研发工作。

从博士后到第一次创业，是从学术研究人员转型成为工程师；进微软亚洲研究院，则又重回到了学术道路。在 MSRA 期间，专注于研发大规模机器学习平台，以出色的科研和工程综合能力，发明了世界上最快主题模型算法 LightLDA 及分布式训练系统：只用几十台服务器就能完成之前需要数千服务器才能完成的训练任务。“LightLDA 的确是迄今为止，我做出来最有影响力的工作。人常说，评价一个学者水平高低不是看成果多少，而是看他能到达的最高水平，可以说这项研究让我跻身于世界一流研究人员的行列”。

MSRA 被称作中国 IT 届的黄埔军校，精英荟萃，并且老师木的成就也开始受到各界的认可，但是他却出人意料的放弃了 MSRA 的优厚工作，走上了创业的路途，更是参与到深度学习框架这种战略级产品竞争中。众所周知，很多大公司都出有自己的深度学习框架，Google 的 TensorFlow，微软 CNTK，Amazon 的 MxNet，Facebook 的 Caffe2 等，并且都在努力的建立生态。以老师木的视角，他是如何看待这些框架？我们从深度学习技术和框架、LightLDA 两大方向和老师木进行了一次深度访谈。

## 关于创业

### InfoQ：为什么你放弃 MSRA 的优厚工作去创业？

老师木：创业者一定有一个或大或小的愿景，或者说使命感，未来的世界应该是什么样的，怎么努力促使愿景实现。我的愿景是：人工智能技术赋能各行各业，推动人们工作效率和生活质量更高，把人类从机器擅长的工作中解放出来，让人类去做更需要创造力的事。在这种使命驱动下，首先选择做什么事最有利于这个愿景实现，其次选择做事的形式。要选能突破自我，能最大化创造价值的事和形式。

## InfoQ：怎么看待人工智能的市场潜力？

**老师木：**首先，互联网行业已经充分验证了数据驱动的业务模式。其次，互联网行业之外的存量业务有显著的人工智能技术红利可吃，或者刚刚尝到人工智能技术的甜头，或者是尚未开垦的处女地，仅仅把人工智能技术引入已有业务，就能获得竞争优势，甚至带来质的飞跃。最后，人工智能技术革命会催生一些新的产业，譬如自动驾驶，精准医疗等。据此，有人认为这次由深度学习引发的大潮可能是第三次工业革命。

## InfoQ：深度学习在业界有哪些靠谱应用？

**老师木：**每个高商业价值的互联网应用背后都有深度学习的身影，搜索引擎，广告，推荐引擎，用户画像，社交媒体，共享经济等等。人类智能可概括为感知，决策和控制三方面，有监督深度学习方法最先在感知类型的任务（图像视频，语音，语言的理解）中取得成功，譬如安防，医学影像，色情信息过滤，语音助手，机器翻译等都已经商用落地。强化学习在决策和控制方面也取得很多成果，主要是机器人自动控制，自动驾驶，处在快速发展中。

## InfoQ：深度学习在技术上存在什么瓶颈？最可能在哪里获得突破？

**老师木：**先分别说有哪些关键问题。在算法和理论方面，目前有监督学习应用最成功，各行各业积累了大量的无标签的数据，怎么利用上无标签或弱标签的数据？深度学习在感知（Perception）类型的任务上非常成功，怎么与认知（Cognition）方法（符号推理）结合形成最终决策？在理论上如何理解深度学习这么惊人的效果，怎么在理论指导下设计模型，而不是靠 ad-hoc 经验试？在计算效率方面，服务器端主要考虑扩展性，怎么能让一批高吞吐协处理器协同解决一个大型任务时总体利用率最高，在终端上则主要是考虑低功耗实现，能否同时实现易用性和高效性。在应用方面，主要是在一些高商业价值的问题上能否从技术上打通达到可用程度，AlphaGo 非常成功，但商业价值还不明确，在杀手级应用如自动驾驶，精准医疗，自动化交易等方向上取得成功，更值得期待。

理论和算法研究上的突破通常可遇不可求，更难预测，而且是否真的

突破最终也要落实到实际应用中去评判。在计算力和应用上的突破确定性更高一些。我们是瞄准了计算力这个方向的商机，一会儿可以深入探讨下这方面的问题。某些垂直应用如自动驾驶方向聚集了大量资金和人才，这方面的突破希望也很大。

### InfoQ：为什么计算力会成为深度学习的一个突破方向？

**老师木：**首先，计算力是极其关键的一项支撑技术。最近发生的人工智能革命通常被认为是三驾马车驱动，数据，算法和计算力。与上世纪九十年代相比，深度学习在算法原理上并无二致，在数据和计算力方面进步更大，各行各业积累了大量的优质数据，GPU 作为新的计算手段引爆了此次深度学习的热潮。

其次，计算力方面还有现成的红利可吃，相同的算法，如果能用上更多的数据，或者用更大规模的模型，通常能带来效果的显著提升，能不能做的更大取决于计算力的水平。

再次，算法和原理的研究进展依赖于计算能力，好的计算力平台可以提高算法和原理研究的迭代速度，一天能实验一个新想法就比一星期才能实验一个新想法快的多。有些理论问题本身是一个大规模计算问题，譬如神经网络结构的自动学习等价于在一个超大规模假设空间的搜索问题，没有强大计算力的支持就只能停留在玩具数据上。深度学习是受生物神经网络启发而设计出来的，现在人工神经网络的规模还远远小于人脑神经网络的规模，人脑有上千亿神经元细胞，每个神经元平均有成千上万的连接。

最后，如何在低功耗约束下完成高通量的计算也是制约了深度学习在更多终端上应用的一大因素。

### InfoQ：计算力具有什么样的商业价值？

**老师木：**一方面，计算力的商业价值体现在它是数据驱动型公司的大部头营业支出（硬件采购，人力成本等）。数据驱动型业务的完整链条包括数据收集，预处理，深度分析和在线预测，无论是私有部署还是上公有云，建设高扩展性的基础设施等支撑技术，都是一笔不可忽视的开销。另一方面，计算力也是数据驱动型公司获得竞争优势的关键，人工智能可提

高公司业务效率，而计算力又可提高人工智能的效率。目前，围绕着计算力已经出现了诸多成功的商业模式，譬如公有云，面向私有部署的商业技术服务，深度学习加速器（GPU，DPU）等。

### InfoQ：计算力在技术上有哪些瓶颈？

**老师木：**从硬件看，我们现在使用的都是冯诺依曼结构的计算机，它的主要特点是计算单元和存储单元分离，主要瓶颈表现在摩尔定律（Moore's law）的失效和内存墙（Memory wall）问题上。克服摩尔定律的主要途径是增加中央处理器上集成的核心（core）数量，从单核，多核发展到现在众核架构（GPU，Intel Xeon Phi），但芯片的面积及功耗限制了人们不可能在一个处理器上集成无穷无尽个核心。内存墙的问题是指内存性能的提升速度还赶不上CPU性能的提升速度，访存带宽常常限制了CPU性能的发挥。纯从硬件角度解决这些瓶颈问题，一方面要靠硬件制造工艺本身的发展，另一方面可能要靠新型的计算机体系结构来解决，譬如计算和存储一体化的非冯诺依曼结构计算机。除了高通量的计算，在电池技术没有大的突破的前提下，终端应用场景（物联网，边缘计算）里低功耗也是计算力的一项重要指标。当前，深度学习专用硬件创业如火如荼，有可能会被忽视的一点是：对突破计算力瓶颈，软件至少和硬件一样关键。

### InfoQ：为什么软件会成为计算力突破的关键？

**老师木：**计算力的基础设施要满足上层用户对易用性，高效率，扩展性的综合需求，仅有硬件是不够的。一方面，数据科学家和算法研究员不像系统研发工程师那样深刻立刻硬件的工作机理，不擅长开发释放硬件计算潜能的软件，对数据科学家最友好的界面是声明式编程，他们只需要告诉计算力平台他们想做什么，具体怎样算的快要由软件工具链来解决。另一方面，尽管单个众核架构的协处理设备（如GPU）吞吐率已远超CPU，但出于芯片面积 / 功耗等物理限制，任何一个单独的设备都无法足够大到处理工业级规模的数据和模型，仍需由多个高速互联的设备协同才能完成大规模任务。出于灵活性需求，设备之间的依赖必定由软件定义和管理，软件怎样协调硬件才能提高硬件利用率和释放硬件潜能极具挑战，至关重

要。在相关领域，软件定义硬件已是大势所趋：上层软件决定底层硬件的发展方向，底层硬件要取得成功离不开完善的上层软件生态。

### InfoQ：业界已经有很多软件平台，为什么要再打造一个？

**老师木：**用户选择众多，但仍有重要需求未被满足，深度学习框架技术演化仍未收敛。深度学习框架一定会出现 Hadoop 那样具有市场支配地位的产品，也就是所谓的事事实工业标准，而现在还没有任何一个软件平台达到这种地位。工业标准级的平台不仅要解决眼前的需求，更要面向未来。现在的确有一些知名的软件平台，但业界还有相当一部分重要需求没有被满足。比如，现有技术方案对于单设备或多设备数据并行这种简单场景的支持已经非常优秀，但在模型更大或者神经网络拓扑更复杂时，通用框架的易用性和效率都大打折扣，有这种需求的工业级应用只好去用定制的 HPC 方案（譬如百度的 DeepSpeech）。问题的根源是，设备之间互联带宽远低于设备内访存带宽，这是和传统 CPU 上内存墙（Memory Wall）类似的难题。我们团队经过艰苦卓绝的努力，探索一条走向通用解决方案的技术路径。沿这个思路开发的软件平台，有望既享受软件的灵活和便利，又享有专用硬件的高效性。我们坚信，通用的解决方案是深度学习平台技术收敛的方向，只有这种通用的解决方案才是深度学习平台的最终形态。

### InfoQ：能说说你们产品的主要技术特点是什么吗？

**老师木：**深度神经网络和人脑信息处理本质数据流计算，信号的传播即计算，然而当前主流的底层硬件都是冯诺依曼结构。纯硬件实现的数据流计算机还不现实，现在必须依赖深度学习软件平台来完成这样一个翻译或者映射的过程：从数据流表达式到冯诺依曼结构上的指令序列。软件平台最终价值体现在易用性和高效性。易用性，要支持用户能够使用最自然的表达方式来描述各种神经网络计算的需求；高效性，对所支持的任何一种上层需求，都能基于通用硬件资源表现出专用硬件的那种效率。我们的产品开创了一种和现有深度学习框架截然不同的技术路线，细节上表现出来静态编译，全链路异步，去中心化，流式计算等特点，我们认为这是深度学习基础架构实现易用和高效的必由之路，是深度学习框架技术收敛的方

向。

### InfoQ：长江后浪推前浪，这样一个先进的技术架构生命力会有多久？

**老师木：**首先，我们可以探讨一下深度学习的范式还有多久生命力，毕竟技术架构应需求而生。可以从这几方面看：从数据流计算模型是生物体采用的信息处理机制，是人工智能的效仿对象；人工神经网络已经在多个领域取得成功，而且深度学习本质上还是统计学习理论，利用算法在数据中挖掘统计规律性，这种学习机制的本质不会变化；深度学习算法便于利用并行硬件的威力，算法和硬件的天作之合，还看不出取代它的必要。其次，从计算机体系结构及硬件演化方向上看，软硬件结合的数据流计算机代表着突破摩尔定律和内存墙限制的方向。

### InfoQ：是不是只有大公司才需要这样的基础设施？

**老师木：**并不是。目力所及，这样的基础设施已经不是大公司的独享的专利，拥有数十台服务器的中小企业，大学研究院所比比皆是。数据驱动是一种先进的生产力，所有行业最终都会变成数据驱动，每个行业的每个公司的数据都在积累，每个公司对数据分析的需求都在进化，从浅层的分析到深度分析，这个大趋势呼之欲出不可逆转。十年前，会有多少公司需要 Hadoop，现今几乎所有的公司都要用到 Hadoop。历史一再证明，无论计算能力发展到多强大，应用总能把它用满。多年以前，有人还觉得 640K 内存对于任何人来说都足够了，今天 64G 的内存都开始捉襟见肘，一辆自动驾驶测试车每天收集的数据达数 TB 之多。从来不是强大的计算力有没有用的问题，而是计算力够不够用的问题。

### InfoQ：深度学习框架竞争很激烈，而且看上去都是业界巨头在玩。

**老师木：**是的。一个深度学习框架一旦像 Hadoop 那样成为事实工业标准，就占据了人工智能各种关键应用的入口，对各类垂直应用，基于私有部署的技术服务，公有云上的 AI 即服务业务，甚至底层专用硬件市场都有举足轻重的影响。它的角色就像互联网时代的浏览器，移动互联网时代的安卓操作系统一样，是战略级产品，业界巨头谁都不想让给他人也就不奇怪了。目前，大公司出品的比较知名的框架有 Google

的 TensorFlow，微软 CNTK，Amazon 的 MxNet，Facebook 的 Caffe2，PyTorch，国内百度的 PaddlePaddle 等。

### InfoQ：为什么用创业的方式做这样一件事？

**老师木：**这种事既有技术攻关上的挑战，也有资源组织上的挑战。这就需要科研院所那种人才密集度，又需要公司的组织支持。我既有在大公司工作的经历，也有两次创业的经历，个人理解，创业是社会资源组织和分配的一种优秀机制，能最大化这项事业的成功率。首先，创业是社会鼓励创新和承担风险的一种资源分配形式，有潜力的创业团队能得到所需要的资源（资金和人才），同时有高度灵活的机制，在大公司，未必是最适合做这项事业的人来承担这样的项目。其次，一项充满挑战的事业需要具有聪明才智的人以持久的热情投入其中，创业公司那种公平合理的利益分配机制才能最大激发成员的主观能动性，为业界做出实质贡献的人也应该得到回报。

### InfoQ：创业公司做这样一件事看上去很不可思议。

**老师木：**有很多大公司加入这场竞争，说明存在真实的需求，而且市场容量足够大，看上去创业公司做这样的产品非常难，实际上大公司做也是同样地难。深度学习框架的用户是开发者 (developer)，也就是说的 To developer，要把这样一件产品做成功，被业界广为采用，关键看两点：

首先，这种深度学习框架是技术密集型产品，一定要做到最广泛的满足实际需求，而且在某些方面要有不可替代的优势，有突出的长板。

其次，要形成生态，具有完善的社区支持，做到没有明显的短板。一个组织只要具备实现这两点目标的要素，就有机会，而不在于那是小公司，还是大公司。

事实上，在开源软件范围竞争还是非常公平的，原来名不见经传的人开发出的软件的确好用就能火，大公司开发出的软件质量不行也没人用，最终靠产品质量说话。现在，创业公司聚集了业界最优秀的一批人，聪明，更重要的是有野心（进取心）。当然，对创业公司来说，不仅要取得产品的成功，还要取得商业上的成功，让所有参与这项事业的人拿到现实的回

报，公司自身也获得更充足的资金支持投入再生产，做出更优秀的产品。大公司在开源产品的商业化上更从容一些。个人观点，很多大公司与你竞争不可怕，更可怕的是面对很多创业公司的竞争。最终结果取决于产品质量。

**InfoQ：如何取得商业上的成功？只有好的技术也可能赚不了钱。**

**老师木：**取得商业上的成功是创业公司的最终追求，我们也一样。我的理解，这涉及两个“价值”问题。

第一，我们在做的事是否为用户创造了价值，我坚决信奉 create value, money follows;

第二个是回归商业价值，在为用户创造价值的前提下，我们需要探索出一条双赢的利益分配机制，把用户转化成客户。

现阶段，我们聚焦在解决第一个问题，打造出解决用户需求和痛点的产品：深度学习平台，不贪大求全，只追求把整个链条中的那最关键一环打造到极致。这是我们这个团队在人工智能大潮中参与顶端竞争的切入点，在我们眼里是那个撬动地球的杠杆支点。从为用户创造价值这个角度切入能最大化实现商业目标的成功率，而且有可能把我们推举到比其它选项要高的多的高度。微软，谷歌，英伟达，甲骨文，华为这样伟大的公司都是因为有了创新的产品才形成了伟大的商业公司。我们对商业模式的各种选项都持 open 态度，不排斥和高商业价值的垂直场景结合。

**InfoQ：您们的深度学习平台第一版预计什么时候公测？需要从哪些方面准备？**

**老师木：**系统主体开发已经完成，目前处在内测阶段，计划年底时开源。开源之前需要从以下方面做充分准备：第一，产品功能完整性，要支持主流的深度学习模型，譬如 CNN/RNN/LSTM，支持图像，语音和语言经典应用；第二，验证高效性，在业界公认的大规模评测中表现出效率优势，给出具体技术指标，如在多大规模上跑到什么水平的加速比，设备利用率等等；第三，打磨易用性，和上下游工具，和已有深度学习框架的兼容性，以及文档建设等等。我们团队先从技术方面打好一个底子，当用户想为这个项

目做贡献时，可以更容易加入进来。

**InfoQ：您们研发深度学习平台会兼容哪些芯片？支持什么操作系统，支持 Linux, Windows, Android 和 iOS 吗？**

**老师木：**目前我们聚焦在服务端的训练场景，在这种场景下，GPU 是最经济的选择，所以目前只支持纯 CPU 或 CPU+GPU 的异构集群，如果未来硬件市场发生变化，我们也可以支持其它芯片。服务器上主要操作系统是 Linux 和 Windows，所以目前只支持这两种。终端的应用场景主要是在线推理（inference），我们团队目前没有投入。

## 关于 LightLDA

**InfoQ：LightLDA 是您的代表作之一么？能给大家介绍下这个项目的一些情况么？**

**老师木：**LightLDA 的确是迄今为止我做出来最有影响力的工作。人常说，评价一个学者水平高低不是看成果多少，而是看他能到达的最高水平，可以说这项研究让我跻身于世界一流研究人员的行列。首先，算法结果是一流的，LightLDA 是当时业界最快的训练 Latent Dirichlet Allocation (LDA) 主题模型的算法，它把单个词采样降低到  $O(1)$  复杂度。其次，系统实现是一流的，我们仅用数十台服务器，完成之前成千上万台服务器才能做的事。LightLDA 和许多其它优秀科研成果一样，是集体努力的结晶。那个时候，CMU 的邢波教授 (Eric Xing) 在 MSRA 任顾问，微软团队和他领衔的 Petuum 团队合作达成此项成果，论文发表在 WWW 2015，系统代码在 Github 开源，也成功应用于微软搜索广告和情景广告产品中。

主题模型特别是 LDA 是广告系统和推荐系统中的关键组件，据说“Google AdSense 背后广告相关性计算的头号秘密武器 Google Rephile”就是一个巨大规模的主题模型。大约三四年前，微软很多产品想用类似的技术，然而并没有大规模主题模型的训练系统。有一天，主管这个领域的副院长马维英（现今日头条副总裁）和我讨论时，说起这件事，产品

部门经常问他的团队有没有这样的解决方案，问我愿不愿意干。恰好那时邢波教授也开始做 MSRA 的顾问，邢教授的团队在这方面有很积累，微软正好可以和他在 CMU 的团队合作研发大规模主题模型训练技术，双方一拍即合。当时，从公开渠道能了解到，为解决工业级需求，训练数据可能涵盖数亿个文档，每个文档包含十几到数百个词，为了覆盖长尾词和长尾语义，词典可能包含数十万到百万个单词，主题个数远超业界发表论文的数字（仅数百个主题），达到万，十万，甚至百万，最先进的解决方案需要数千台服务器运行数天才能得到结果。我们当时立下的 flag 是，相对于业界最好解决方案，做到各个维度上都有数量级的超越（服务器数量必须是数十台，我们那时拿不到数千台这么奢侈的硬件支持，数据规模做到数十亿 Bing 索引的主流网页，词典和主题数至少做到十万级别）。稍微推算一下，就可以知道，即使是当时最先进的算法 SparseLDA，在给定的硬件环境中训练这样规模的模型需要半年到一年的时间。再加上身处研究部门，一没有可供使用的集群，二没有工程师团队的支持，微软这边全时投入的只有我和实习生高飞，这个目标看上去是 mission impossible。我当时的想法是，最低目标要做出来一个能满足产品部门需求可用的主题模型，能不能做出打破纪录，就看运气了。

### InfoQ：请问大规模训练 LDA 模型的瓶颈是什么？

**老师木：**训练 LDA 的算法可以分成两类，一类是变分贝叶斯法，一类是 Gibbs 采样算法。前者计算过程和中间表示都是稠密的，分布式实现时通信量较大，后者是稀疏计算，通信量小，一般大规模主题模型都基于 Gibbs 采样算法实现。使用 Gibbs 采样算法时，算法复杂度和系统实现两方面都有困难。假设有 100 亿文档，平均每个文档有 100 个词，一共有 10000 亿个词，训练过程迭代 100 次，那就需要对 10000 亿个词扫描 100 遍。标准的 Collapsed Gibbs 采样算法处理一个词的计算复杂度与模型的主题数量有关，假设要训练包含 10 万个主题的模型，那么每个词就包含 10 万次计算，主频为 2GHz 的 CPU 核心每秒能处理 1000 个词，这样估算一下下来，假设使用一个单线程程序来做这件事，共需要 1000 亿秒，

也就是 100 万天。使用 10000 个 CPU 核心的分布式集群去训练，假设线性扩展性，也需要 100 天之久。假如每个词的采样效率能提高 100 倍，那么使用 10000 个 CPU 核心的集群去训练这个模型就只需要 1 天。前人已经提出了 Gibbs 采样算法的多种改进，譬如 SparseLDA, AliasLDA，但这些算法的单个词的计算复杂度仍与模型的主题数量相关，与“创造奇迹”仍有距离。另外，实践上，算法中总有一些步骤是无法并行化，受制于阿姆达尔法则，分布式系统很难做到线性加速比，所需要的时间会比上述预估的时间更长。

### InfoQ：LightLDA 设计之处，面临了哪些挑战？

**老师木：**我们 LightLDA 团队资源匮乏（计算资源，工程师资源），同时在算法和系统实现上都挑战极大。我个人认为最大挑战在信心方面：我们能不能做到？在此之前，有多位知名科学家和资深工程师在训练大规模 LDA 的问题上耕耘已久，他们已经把算法和系统实现推进到相当的高度，即使采用当时最先进的技术，仍不可能实现我们的目标。必须做出显著超越前人的奇迹技术突破才有可能实现目标。我和学生都是第一次从事大规模机器学习的项目，名不见经传，何德何能，能比另外一些特别牛逼的人物做的还要好？

首先是算法上的突破。我在重现和把玩 SparseLDA 和 AliasLDA 时，被可遇不可求的灵感眷顾：解耦 Gibbs 采样中与词自身相关的因素和词所在文档上下文的因素这两个因子，能做到单个词采样复杂度与主题个数无关。马维英院长第一次听我介绍完这个想法和初步实现结果后说 *too good to be true*，的确，谁能想到这样一个小小的 insight，竟然能把单个词采样复杂度降到  $O(1)$ ，理论上使得达成那个宏伟的目标成为可能。这个灵感来的偶然又必然，机遇偏爱有准备的人。我动手能力比较突出，很快就重现了 SparseLDA 和当时刚刚在 KDD 上发表并获得最佳论文奖的 AliasLDA 算法，同时理论功底又比较扎实，很快就深刻理解了它们的关键所在。我不断把玩这两个算法，在直觉和理论分析指引下做一些改动，然后观察是否有效，终于在一次改动后发现计算效率陡升，让人怀疑是不

是出现了有益处的 bug，再三推敲后终于确认，这是一个有深刻内涵的新发现。这又一次印证了我从清华数学系林元烈老师那学到的一个诀窍：熟能生巧。他的随机课程巨难无比，我刚开始怎么都入不了门，和很多自认佼佼者的同学一样竟然期中考试不及格。林老师说了一番这样的道理：他认识很多大牛数学家，即使是像他们那么聪明的人，在掌握一些艰深的数学科目时，也是通过做特别多习题才能悟道。我就硬着头皮做了很多习题，有的证明看不懂，甚至都背下来了，也是突然一瞬就知道了随机过程怎么回事。每次遇到困难，在说放弃之前再坚持一会儿结果就会不同。

找到理论上性质很好的算法，只是万里长征第一步。怎么高效地用程序实现，特别是在分布式环境下接近线性加速，包含了一系列的技术挑战，任何一个环节掉链子，所有努力都会化成泡影。做这类事的特点就是，兵来将挡，水来土掩，在你不知道前人这些技巧时，你要自己发明出来，但在系统领域极大概率是这个发明已经在经典文献中被提出过了。我们解决了两个突出的难题，超大规模模型的内存瓶颈和通信瓶颈。100 万的词典和 100 万个主题，模型之大，前所未有，意味着需要若干 TB 的内存，如何存储和支持快速访问也极其严峻。在分布式环境下，如何有效掩盖通信开销又不损失模型精度，也是当时面临的一个主要难题。我的学生高飞在工程实现方面特别给力，交给他的事情总能又快又好的做完。事后回顾这段经历，他说，这段日子是他最愉快的经历之一，偶尔会感到绝望，总发现我在前面仍激情满满的坚持，他深感佩服。我的领导马维英和刘铁岩研究员则克服重重困难，为这个项目提供资源支持和高屋建瓴的指导。同时，我们和 CMU Petuum 团队，Eric Xing, David Dai, Jinliang Wei, Qirong Ho，尽管身处太平洋两岸，但几乎每天都有邮件讨论，每周都有好几次电话会议，遇到技术难题大家凑在一次分析，提出不成熟的好点子又立刻能得到挑战，共鸣和支持，缺少任何一个人，结果都不是大家看到的样子，这就是一个优秀团队的魅力所在。

没有前面技术突破，绝不可能达到目标。仅仅有前面的算法突破，没有执行成功，这项研究也就是一个微不足道的 trick，绝不可能产生后来

的影响。

### InfoQ：LightLDA 如何借助 DMTK 框架做并行化？LightLDA 有哪些优点？

**老师木：**这里可能有一个小小的误解。在 Github 上发布时，LightLDA 是作为 Distributed Machine Learning Toolkit (DMTK) 的一个组件发布的，但实际上 LightLDA 最初是使用 Petuum 的参数服务器实现并行化。在 LightLDA 论文发表后，微软酝酿和发布了 DMTK 项目，这时候把 LightLDA 作为 DMTK 的一个主要应用集成进去了。LightLDA 的优点就不多说了，主要是快，扩展性好，用少得多的硬件资源就可以解决规模大的多的问题。我来说一下开源版本的缺憾吧。首先，理论上单个词采样复杂度是  $O(1)$ ，在工程实现上，因为随机访存造成 cache miss 太多的原因，没有完全发挥算法的优势，不久以后，清华大学朱军和陈文光教授的课题组做了一些新的创新，提出了 WarpLDA，重排训练数据的访问顺序，大大减少 cache miss，才真正发挥了这类  $O(1)$  复杂度算法的威力；其次，LightLDA 开源的代码并没有包含数据预处理和在线预测这一整套工具链，使得用户必须自己去开发和踩坑；最后，有一些较高级的特性虽然在内部版本实现了，却并未在开源代码中发布，譬如能搞定长尾语义的非对称先验的 LDA 等。我们也没有把单线程版本发布出来，方便同行做纯粹地算法比较。

### InfoQ：通过 LightLDA 项目，得到了什么启发？

**老师木：**第一，的确存在不可替代的技术，平凡的创新和破坏式创新的效果不可同日而语，后者往往有四两拨千斤的效果。

第二，要敢于迎接挑战，承担风险，个人理解，这相对于平凡而稳妥的道路更划算，做一件挑战但有风险的事，可能需要付出于平常事 3 倍的努力，但可能获得做 10 件平常事才会有的回报。

第三，无论是科学研究，产品研发，还是商业竞争是智商，意志，情商等综合素质的全面比拼，不仅要有不可替代的优势，在其它任何方面还不能有短板。

第四，机会总是眷顾有准备的人，有所准备才能抓住稍纵即逝的机会。总结，LightLDA 让我体验了做成一件有影响的事所需要的所有困难，我好像对看上去很难的事不会感到畏惧。

## 关于人工智能从业

**InfoQ：人工智能前景良好，那么从业者能发挥什么角色？**

**老师木：**有三种类型的技术可做：

1. 研究机器学习算法或原理，解答怎么做（How）或为什么这么做（Why）的问题，譬如研究怎么训练深度学习模型，什么样的神经网络结构效果最好，为什么深度学习要比其它机器学习方法效果好等等，简略称为原理问题；
2. 机器学习的基础设施，什么样的软硬件设计能使得机器学习算法计算更快，能用上更多数据，或者使模型规模更大，例如研发深度学习软件框架，或深度学习专用硬件等等，可归结为计算力问题；
3. 如何应用机器学习技术（算法和计算力）解决工作和生活中的实际问题，譬如互联网广告系统设计，推荐系统，游戏博弈（如 AlphaGo），自动驾驶等等，可归类为应用问题。

**InfoQ：从事哪种类型的工作更有竞争优势？**

**老师木：**这三种类型的工作我恰好都做过，应该说哪个都很有用武之地，哪一个方向能做到顶尖水平都不易，做好了都能赢者通吃，全栈则更有优势。当然，这些工作也存在一些具体的差别。理论问题，进入门槛较高，工作岗位不太多，一般是兴趣驱动，看天赋和运气，这方面的突破，影响范围广，惠及全行业，从创业看，难以形成独立的商业模式，一般是在大学或企业研究院开展。计算力问题，影响力能到达全行业，通常是业界巨头和精干的创业团队的强项，岗位不太多，门槛也比较高，但主要看后天努力，一般是努力总有结果，创业上有可能形成独立的商业模式。应用类型的问题，业界需求最大，进入门槛低一些，确定性高，离商业近，

周期短，见效快，影响力一般受限于特定领域。统计上，少数人从事理论和计算力类型的工作，大部分人从事应用驱动的工作。现在的开源软件和公开课非常普及，为有志于在此方向上有所造诣的同行提供了前所未有的良好条件。

最后的话：“遍地黄金的日子过去了，低垂的果子已经没了”，技术创新主导的时代必将来临，让我们以“像鹰一样的眼光，像狼一样的精神，像熊一样的胆量，像豹一样的速度”，去抓住属于技术人的机遇。

# 月活超美国人口十分之一，各大科技巨头纷纷布局，智能音箱背后有何门道？

作者 蔡芳芳



未来生活一定是智能的，科幻小说里曾设想过的场景正一步步变成现实。在这个不可逆转的技术发展趋势里，智能音箱扮演的会是怎样的角色？现阶段的智能音箱及其背后的核心技术发展到了哪一步？

## 写在前面

在中国人还在为哪款手机更好而撕逼不休的时候，智能音箱已经悄悄深入美国人的生活。

2017 年 7 月亚马逊 Premium 会员日，它卖出了超过去年同期 7 倍的 Echo 系列智能音箱。而在 2016 年下半年，Echo 总共已经卖出超过

700 万台。市场调查公司 eMarketer 近日公布的智能音箱市场研究报告预测，美国今年智能音箱的月活跃用户将达到 3560 万人。这一数字已经超过美国总人口的十分之一。

智能音箱以及它所代表的趋势，已经不容忽视。智能音箱到底值不值得用？智能音箱的未来又会如何？本文将带你一起观察国内外智能音箱市场的火热现状，解析背后的技术，并给出我们对于趋势的分析和判断。

## 何以起风波？

根据 Gartner 预测，到 2018 年将会有 30% 的人机交互通过自然语言会话完成<sup>[1]</sup>。而基于远场的自然语音交互恰恰就是催生出智能音箱市场的重要需求。

以一次普通的听音乐和看视频为例：

现在用手机听音乐，首先要先解锁手机屏幕，打开某一个音乐 APP，搜索你想听的那首歌，然后点击播放；如果变成语音交互会怎么样呢？你只需要说：播放陈奕迅的好久不见。

如果这个时候恰好是周日晚上 10 点，你想看最新一期极限挑战。如果是现在，你需要在手机上切换到视频播放软件，或是打开电脑输入视频网站地址，然后搜索极限挑战第三季，最后选择最新一期播放；如果换成语音交互，你只需要说：播放极限挑战第三季最新一期。

类似以上场景，语音交互很多时候执行效率都明显高于 GUI 交互。业界普遍认为，智能语音交互会成为未来人机交互的新方式，一如当年乔布斯在 iPhone 上用触屏打败传统手机键盘，语音交互也可能会颠覆图形界面交互。而智能音箱已经成为智能语音交互的重要载体。

2014 年 11 月，亚马逊推出基于语音交互的智能音箱 Echo，拉开智能音箱市场大幕。2016 年 Echo 的销售数据一路高涨，一举突破 500 万台，亚马逊获得先发优势，在美国市场积累了大量用户。

亚马逊率先尝试并大获成功，证明了以智能音箱作为智能语音交互载体和智能家居入口的可行性和正确性。各家纷纷入局智能音箱市场，既是

## 智能音箱初代产品发布时间表

2014年11月	Amazon Echo
2015年5月	京东叮咚
2016年5月	Rokid外星人
2016年11月	Google Home
2017年5月	微软Invoke
2017年6月	苹果HomePod
2017年6月	小雅AI音箱
2017年7月	天猫精灵
2017年7月	小米AI音箱
2017年9月	Tichome
.....	.....

响应智能语音交互时代的召唤，也是不甘亚马逊独占用户和市场红利。

根据 CIRP、RBC Capital Market 数据，自 2014 年 11 月发售以来，包括 Echo、入门级 EchoDot 和便携式 Tap 在内的亚马逊智能音箱，已累计销售超过千万台，销售额达到 8 至 10 亿美元。

而根据市场调查公司 eMarketer 近日公布的智能音箱市场研究报告预测，美国今年智能音箱的月活跃用户将达到 3560 万人，比去年增长 128.9%，其中亚马逊的 Echo 将达到 70.6% 的市场占比，远远领先于第二名 Google Home 的 23.8% 以及联想等其他品牌。今年每月至少使用一次这些语音助手的美国人将达到 6050 万人。这一数字已经超过 1/4 的智能手机用户，并且接近 1/5 的美国人。<sup>[2]</sup>

再看国内的智能音箱市场：

根据《科大讯飞股份有限公司 2016 年年度报告》<sup>[3]</sup>，叮咚智能音箱在 2016 年的总销量为 10 万台。基于线上淘宝（包含天猫）销售数据的跟踪调查，智能音箱品类的整体月销量还不到 2 万台。<sup>[4]</sup>

与国外智能音箱庞大的用户群体相比，国内智能音箱市场似乎“小”到不值一提，但产品数量之多却毫不逊色。

## 乱花渐欲迷人眼：智能音箱产品介绍

智能音箱在传统音箱的基础上增加了一些更“聪明”的功能，主要体现在以下几个方面：

1. 通常内置无线射频芯片或射频模块，可以通过 WiFi 接入互联网。
2. 支持语音交互，无需动手就能控制音箱，一般也支持少量按键操作。
3. 接入丰富的音频内容，如各家音乐提供商的曲库、有声读物等。
4. 提供丰富的互联网服务，如外卖、打车、购物、充话费等，满足日常生活多种多样的场景需求。
5. 实现对各种智能家居设备的控制，使用户能够通过与音箱对话来操控家电产品，成为智能家居控制的核心。

目前市场上的智能音箱产品主流为无屏幕的智能家居助手类音箱，以语音交互技术为核心，旨在成为智能家居的控制中心，亚马逊的 Echo、京东的叮咚、阿里的天猫精灵等都属于这一类。

## 智能音箱代表大比拼之海外党

海外党以亚马逊、Google、苹果和微软这四款智能音箱为代表，这四款智能音箱功能并无太大区别，均支持个人生活助手、智能家居控制等主要功能，都采用了各家自研的语音助手。

产品	Echo	Google Home	HomePod	Invoke
公司	亚马逊	谷歌	苹果	微软
语音助手	Alexa	Google Assistant	Siri	Cortana
代表内容	Kindle、Spotify	YouTube	Apple Music	自选
特色	购物	和TV联动	主推音质	支持Skype

亚马逊进入市场较早并致力于打造开放的 Alexa 开发平台，现在 Alexa 几乎无所不能，成为亚马逊的优势之一。

Google Home 的优势是信息检索和会话聊天的能力。今年四月份 Google Assitant 还添加了一项新功能，能够识别出谁在说话并相应地做出个性化的回应，最多支持六个人的不同声音。谷歌本身拥有完善的内容和应用生态系统，Google Home 已经和部分自家应用以及不少第三方应用打通。通过 Google Home 可以查看日程安排，播放 Google Play Music 和 YouTube Music 里喜爱的音乐，点播 YouTube 或 Netflix 视频并在电视上播放（需配合 Chromecast）。但目前还有很多重要应用如 Gmail、Voice 和 Docs，Google Home 尚无法支持。

苹果和微软的这两款音箱都已经发布但还没有正式上市。HomePod 主打音乐和音质；微软的 Invoke 一开始则以支持 Skype 互联网电话作为亮点，但奈何竞争对手产品更新太快，未等 Invoke 推出，亚马逊的 Echo 和 Google Home 均已经支持拨打电话，虽然还存在一些隐私问题留

待讨论。

2017 年 8 月底，微软与亚马逊达成合作，以更好地整合他们的语音助手“Cortana”和“Alexa”，想必还有一个未明说的原因是为了更好地抗衡谷歌语音助手。

## 智能音箱代表大比拼之本土派



从左至右分别为：天猫精灵、叮咚二代、小米 AI 音箱

产品	叮咚	Pebble	天猫精灵	小雅
公司	京东	Rokid	阿里巴巴	喜马拉雅
语音助手	科大讯飞	若琪	AliGenie	猎户星空
代表内容	百度音乐	网易云音乐	虾米音乐	自家FM
特色	购物	内置电池	购物	情感陪伴

上图选择叮咚、天猫精灵、Rokid 月石和小雅音箱进行对比。目前国内智能音箱产品同质化也比较严重，前三款智能音箱的主要功能依然没有太大差异，而小雅智能音箱主打内容服务，并不支持智能家居控制。

在语音助手方面，Rokid 月石采用了自主研发的语音助手，而叮咚、小雅分别采用了科大讯飞、猎户星空的语音技术方案，天猫精灵的语音助手则集成了思必驰和阿里自研方案。若琪最突出的一点不同是它的唤醒词只有两个音节，而目前市面上其他智能音箱产品的唤醒词大多为三个音节或更多。9 月新发布的叮咚二代支持自定义唤醒词，但实际使用效果有待验证。

## 智能音箱背后的核心技术

智能音箱的核心需求和一切操作的前提是语音交互，因此语音交互技术自然成为其核心技术。当然它的背后还连接着一棵郁郁葱葱的人工智能“技能树”，受限于文章篇幅和笔者能力，本章仅重点介绍语音交互技术。

以下图为例，当我们调戏天猫精灵时，跟它进行一次简单对话的语音交互流程包含哪几步？

### 语音识别

第一步是语音识别（ASR）。智能音箱所使用的语音识别技术与手机端的语音助手有所不同，叫做 远场拾音，指的是我们能够在超过 5 米以上的距离跟设备进行自然语音对话。

有了远场拾音之后，人们可以在家里任意角落、轻松地跟智能设备交流。虽然苹果 siri、谷歌 Google Now、微软 Cortana 等语音助手很早就实现了语音识别，但都是近场语音，使用时需要拿出手机、启动助手、靠近讲话等步骤，与远场拾音相比在体验上有很大的差距。

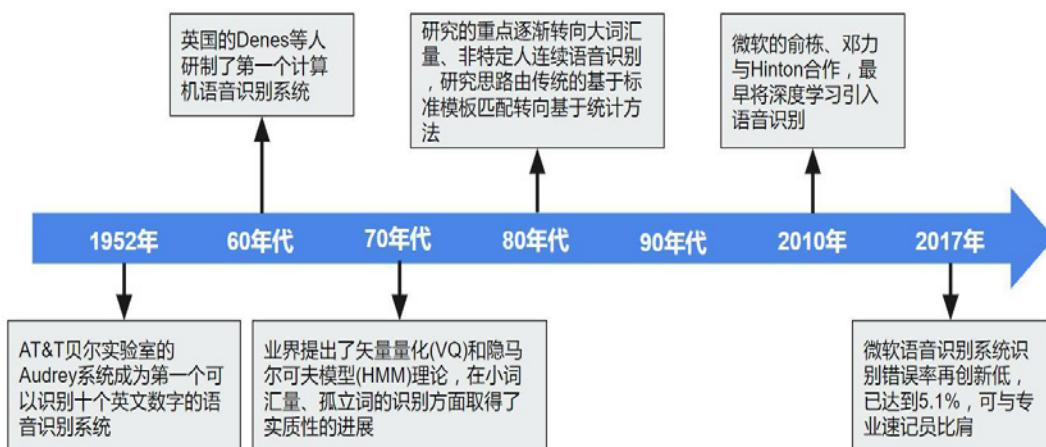
要实现相对理想的远场拾音效果，



降噪是重要的一环。目前常见的做法是利用算法与硬件相结合实现更好的降噪效果。硬件部分，通常麦克风越多，越有助于收集到来自不同方向的声音，从而更容易在噪音环境中识别出有用信息，达到更好的远场交互效果。现在大多数厂商都采用了 6 个以上麦克风组成的麦克风阵列技术，只有 Google Home 通过算法 + 仅仅 2 枚麦克风就实现了还不错的远场拾音效果。

语音识别还需要配置激活词，通过激活词“开启”语音交互功能（就像开机按钮一样），从技术上来说，激活词越短则体验越好、技术难度越高，但同时误激活概率也随之变高。

### 语音识别技术的局限性



### 语音识别技术历史进程

语音识别技术的目标是将人类语音中的词汇内容转换为计算机可读的输入。自 2009 年以来，借助机器学习领域深度学习研究的发展以及大数据语料的积累，语音识别技术得到突飞猛进的发展，语音识别准确率大幅提升。<sup>[5]</sup>

今年 8 月 20 日，微软语音识别系统再次取得重大突破，错误率降低至 5.1%，大幅刷新原先记录，并在语音识别行业树立了新的里程碑<sup>[6]</sup>。

可惜，这些突破更多是针对在安静的室内并近距离靠近麦克风的场合。在噪音或者远场识别环境下，错误率仍居高不下；面对口音、方言，识别

率也还有待提升。

## 自然语言理解

第二步是自然语言理解（NLU），指的是对自然语言的内容和意图的深层把握。通俗地讲，就是在一些话题上，智能设备能够理解人讲的话，或者能把人类的语言理解成机器的语言。目前智能设备只能做到浅层的“理解”，例如把转化成文字后的两句话“给萧敬腾打电话”和“打电话给萧敬腾”理解成同样的操作。

第三步是自然语言生成（NLG），这一步和第二步相反，就是把机器的语言转换成人类的语言。

第二步和第三步从广义上来说也可以合称为 自然语言处理（NLP）。

### 自然语言理解技术的局限性

自然语言理解属于业界难题，也是人工智能的终极目标之一。

现在的自然语言系统一般使用的是基于统计的方法。所谓统计方法，主要指分析单词的统计量作为“特征”，将它们输入到计算模型里，算出一个结果，最终输出成词句。

目前自然语言理解尚处于浅层语义分析阶段，大致包含词法分析、句法分析、语义分析这三个层面。机器对句子的理解还只能做到语义角色标注，如标出句中的句子成分和主被动关系等。当前的研究方法大多是同一套路，即通过语料标注、建立模型、训练模型、使用模型，令自然语言系统做到简单的模型式“理解”。即使是当下最火的深度神经网络，也只是在模式识别这个手段上更加高明一点，仍然无法达到理解语言的程度。自然语言理解研究主要集中的一些特定领域，研究跨领域的通用语言理解为时尚早。

如今为大家所熟知的自然语言处理系统，比如苹果 Siri、微软小冰、讯飞听见等，其实都没有真正的“理解”自然语言本身，大多是基于文本相似度的匹配，更高级的则应用了知识图谱。

## 语音合成

最后一步是语音合成（TTS），也就是将文字转换成声音播放出来，并尽可能地模仿人类自然说话的语音语调，给人以真人之间交谈的感觉。

语音合成技术发展到今天已有 200 多年的历史，但自计算机技术发展起来以后才有了长足的发展。近些年，一种新的基于数据库的语音合成方法得到了更广泛的应用。

随着技术演进，语音合成的复杂度、自然度和音质都已取得不错的成绩，目前研究重点在于提高合成音的表现力（如语气和情感等）以及多语种的语言合成。

## 其他语音交互技术

以上仅仅是最简单的一次对话会涉及到的核心技术，如果进行更复杂的对话或者根据用户给智能音箱指派的不同指令，还会涉及更多（以下技术可能存在交叉）：

- 高级语音技术：声纹识别、情感能识别、多轮会话、场景感知、个性化对话等
- 大数据相关技术：搜索、推荐、知识问答、知识图谱、开放式聊天等
- 其他：可扩展语义技能

其中 声纹识别 技术赋予智能音箱的能力是让设备记忆并识别使用者的身份，在此之上可以扩展更多购物、安防、个性化对话等方面的应用；多轮会话 就是让智能音箱能够在一段比较多来回的会话中自动记住上下文，用户不需要重复说唤醒词，就能对智能音箱提出问题并进行追问，真正做到接近于与人沟通的语音交互体验，多轮会话同样属于语音技术领域研究的难点，其主要建立在语音识别、合成以及自然语言理解等技术基础之上，目前自然度和准确度有待提高；情感能识别 指的是设备能够从声音中听出你现在的情绪，是生气、伤心还是高兴，然后做出相应的个性化回应。搜索和推荐 很好理解，比如你总是放某一类歌曲，下一次你让智能音箱随机给你放首歌，它就能选对你可能喜欢的歌曲。



对话管理是实现多轮对话系统的核心，包括：

- 对话状态追踪(DST)：更新对话状态，记录到目前为止用户所有的聊天记录和系统行为
- 对话决策(DialogPolicy)：依据DST对话状态进行决策并输出系统行为，即决定下一步反馈或调用等行为

## 多轮会话

至于 可扩展语义技能，是指第三方开发者可以在语音开放平台上为语音助手添加新的技能，丰富语音助手的功能。

## 智能音箱功能使用现状与消费者调查

智能音箱背后虽然有许多“高大上”的技术，但这些技术本身还在不断地发展和完善。作为消费者，更关注的是技术交付后的实际使用效果。

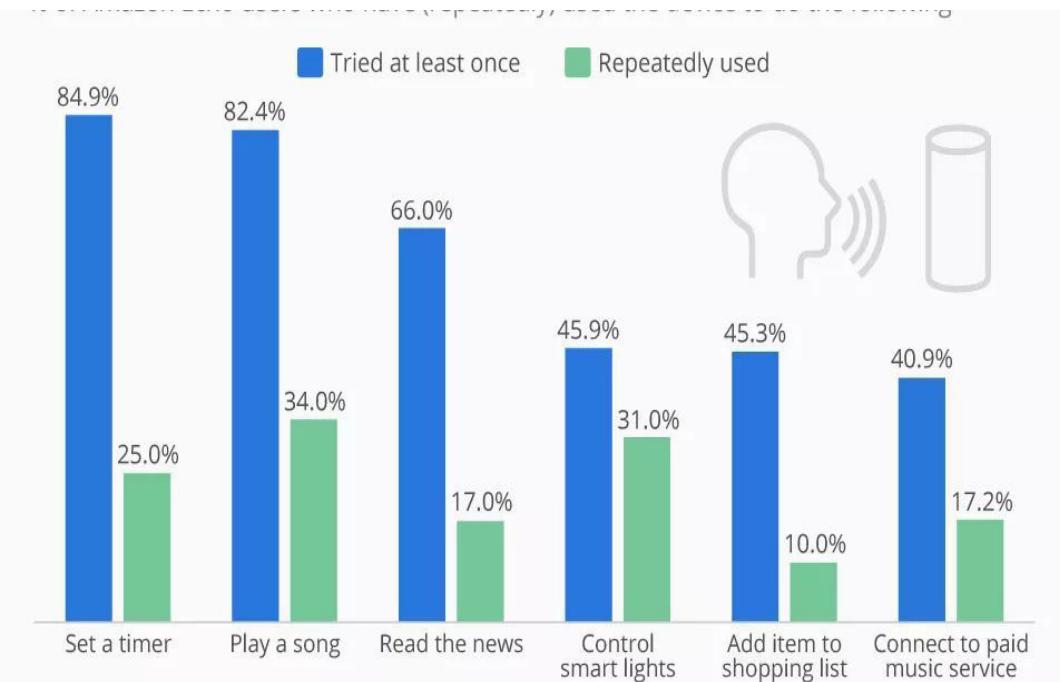
### 智能音箱功能使用现状

截至 9 月，Alexa 已经拥有超过 2w 项技能，Google Assistant 拥有的技能大约不到 600 个，而在这其中真正好用的有多少呢？

据国外的研究机构 2016 年进行的调研显示，Echo 使用最多的功能分别是音乐播放、控制智能灯泡、设置闹钟；用户至少尝试过一次的功能中，排在前三则是设置闹钟（85%）、音乐播放（82%）、新闻播报（66%）。而一直作为 Echo 宣传重点的“Uber 打车”服务，体验比例仅为 6.3%。

再看今年针对美国所有智能音箱使用者的调研结果，最常使用的功能还是诸如普通问答、播放音乐、播报新闻、播报天气、设置闹钟这类比较简单的功能。

易观的一份产业报告称，国内智能音箱使用者最常用的功能是点歌。



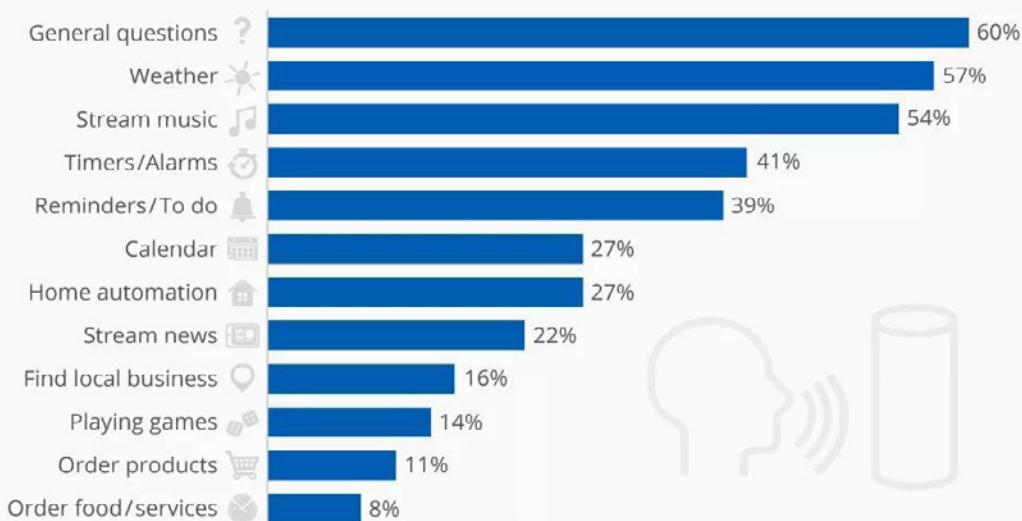
Based on a survey of 180 Amazon Echo users conducted in May 2016

@StatistaCharts Sources: Experian, Creative Strategies

statista

## What Are Smart Speakers Used For?

% of smart speaker owners in the U.S. who use the device to do the following



Base: U.S. households equipped with smart speakers in Q1 2017  
@StatistaCharts Source: comScore

statista

2016 年 Echo 用户调查报告（来自 statista.com）

2017 年美国智能音箱用户调查报告（来自 statista.com）

由此可见，虽然从理论上看，语音交互是更高效的交互方式，但由于现实生活场景复杂且语音交互技术尚未成熟，当前智能音箱的交互体验还无法代替原有的交互逻辑。

## 听听“消费者”怎么说

为了了解消费者对智能音箱的真实看法，笔者找到一些朋友聊了几句。

十几位朋友中只有两位买了智能音箱，其他人大多没了解或者关注得不多，也有人认为现在的智能家电功能不太好用，所以短期内不打算买，当然也有考虑价格因素的。

智能音箱和普通的有啥区别？

你看那些走在前面的人，要么是发烧友，要么是有钱人。

听说过，感觉不是我需要的，好像也没什么花样可玩，就没管过

智能家居现在的程度还是很不智能吧……你光看看那个扫地机器人

感觉现在的智能家居都太虚了，小东西除了灯泡和插座，我真没想到什么实用的。

如果有能根据我的口味比较准确地推荐音乐的功能，不过这点网易云音乐现在已经做得不错了；另外如果能精细化地记住我听音乐的习惯，并能有此衍生出一些功能，估计也能吸引到我

不过我可能会更看重它作为音响的功能

## 聊天记录节选

朋友 W 是科技产品达人，有什么新玩意都会第一时间买来把玩。这次毫不意外地得知他去年就已经买了“叮咚”智能音箱。他家里还有 BroadLink 的智能遥控器用来控制空调，尴尬的是，这个与京东合作的智能遥控器不属于京东微联，所以叮咚并不能控制它去调节空调温度。他反馈最常用的功能是控制开关（京东的智能插座）和听歌，其他还有定闹钟、听喜马拉雅、查天气（但用的不多），其他功能基本没用。并表示短时间内没钱买其他牌子了，但会继续关注。

iOS 圈大咖 Z 今年购置了 Google Home（使用时需要英文 + 架

梯子），并烧钱购置了一些配套的智能家居设备，目前他家里能够配合 Google Home 使用的有 Chromecast 投射、Sony 音箱以及飞利浦的 Hue 灯。最常用的功能是控制智能家居、听歌、放雨声。他表示 Google Home 带来了很好的使用体验，非常智能而且音质很好（当然还是比 Bose Soundlink 要差）；缺点是软件配置使用体验比较差，相关配套智能家居设备很少，而能买到的智能家居设备也比较烧钱。后续他还会考虑购买小米 AI 音箱或 HomePod，未来他的家里可能会有三个智能音箱，他认为小米硬件很全应该会很方便。

Z 对智能音箱未来的发展非常看好：“智能音箱我认为是手机之外的新战场，IoT 的入口，而且有很强的配套购买带动作用，也是 AI 的最好载体。虽然不确定商业模式最终会怎么样，但我认为他会改善人们生活的体验，并创造极大的粘性，渗透到你的生活中，将各种服务、设备变成你的一部分，他在 IoT 上比手机更方便，手机已经像我们的器官一样，为我们提供对外界的眼、耳朵，而 Home 提供的体验更近一步，他把你的家和你连接在一起，你只需要动嘴就可以控制家里的一切，这是信息化和人类结合的重要一步。”

## 厂商布局智能音箱市场，哪家棋高一着？

互联网大公司	以技术、平台和用户数据为支撑，快速整合产业资源，构建智能音箱生态系统	亚马逊 百度 苹果 阿里巴巴 谷歌 京东 微软 腾讯 小米
技术提供商	深挖语音技术垂直领域，提供语音交互技术和智能家居系统解决方案，关注技术与具体应用场景结合	科大讯飞 思必驰 猎户星空 欧瑞博 出门问问 云知声 Rokid 米唐科技 Mobvoi SoundAI
传统音箱制造商	在传统音箱硬件制造上已有积累，音质有保证，倾向与技术和服务提供商合作	哈曼·卡顿 漫步者 JBL Sonos
内容提供商	海量音频内容版权优势	喜马拉雅 酷狗音乐
IT公司	硬件产业链优势	联想 LG

## 百家争鸣，各凭本事

互联网大公司、技术提供商、内容提供商、传统音箱厂商布局智能音箱市场大多以自身优势为切入点。

其中内容厂商的版权优势在巨头公司（如腾讯、阿里）面前其实并不明显。

### 人人争当语音交互时代的安卓，智能音箱不是唯一战场

很多人将语音交互系统比作安卓，而语义技能则被比作安卓应用商店，第三方语义技能是否丰富在一定程度上会影响该智能音箱产品是否能占据竞争优势。

厂商	语音助手	语音OS	开放平台
亚马逊	Alexa	Fire OS	Alexa Skills Kit
谷歌	Google Assistant	Linux-based	Actions on Google
苹果	Siri	/	暂未开放
微软	Cortana	Cortana	Cortana Skills Kit
百度	度秘	Duer OS	小度智能设备开放平台 小度技能开放平台
阿里	AliGenie	AliGenie	AliGenie开发者平台
京东	叮咚	AIUI	DingDong开放平台
小米	小爱同学	/	水滴平台
出门问问	问问VPA	Android-based	AI开放平台
Rokid	若琪	/	Rokid技能商店

百度没有推出自己的智能音箱，而是对 DuerOS 寄予厚望，目标是要打造一个基于语音交互的全新开放平台，向合作伙伴输出 AI 技术能力，将自己的语音系统部署到越来越多的硬件产品中，他们想做“人工智能时代的安卓”。与天猫精灵发布的一天，百度在“百度 AI 开发者大会”上宣布，自己的语音助手 DuerOS 将作为智能语音生态链的基础存在。

但是打这个主意的又何止百度一家？

对各大厂商而言，目前语音开放平台（包括语音交互系统和语义技能）已经成为标配，“有”不再是优势，而“没有”却可能成为巨大的劣势。与此同时，语音交互系统的战火早已燃烧到了智能音箱以外的战场，智能家居硬件、耳机、手机、车载系统、机器人等处处可见各家语音交互系统的身影。

截至 2017 年初，Google Assistant 覆盖智能设备已经超过 1 亿台，并且即将登陆 iPhone，其工程副总裁称“我们的最终目标在于，以后人们可以在任何设备上与 Google Assistant 对话，而它能够尽可能地为你做任何事情。”；而 Alexa 同样以可怕的速度渗透整个电子市场，据不完全统计，截至 2017 年 9 月份已经有近 4 万多种硬件接入了 Alexa。

相较之下，国内厂商的语音开放平台和语音技能商店大多刚推出不久，且开放程度不一，接入的第三方硬件偏少，未来语义技能开发情况尚不明朗。在中文语音交互市场，科大讯飞（市场份额超 70%）和百度（市场份额低于科大讯飞，但拥有更全面的 AI 技术）目前优势较明显。

反观移动端操作系统的另一位霸主苹果，2011 年率先推出语音助手 Siri，颠覆了用户使用手机的交互体验，并引领了一波手机端语音助手的潮流。但从那之后，Siri 除了偶尔被调戏，似乎并不太实用，如今面对层出不穷的语音开放平台，只能运行在封闭的 iOS 内的 Siri 显得有些沉默。

## 看似 FreeStyle 的智能家居布局，却成了智能音箱混战的快车道

与大部分厂商先推出智能音箱、再推动智能家居设备接入的路线不同，

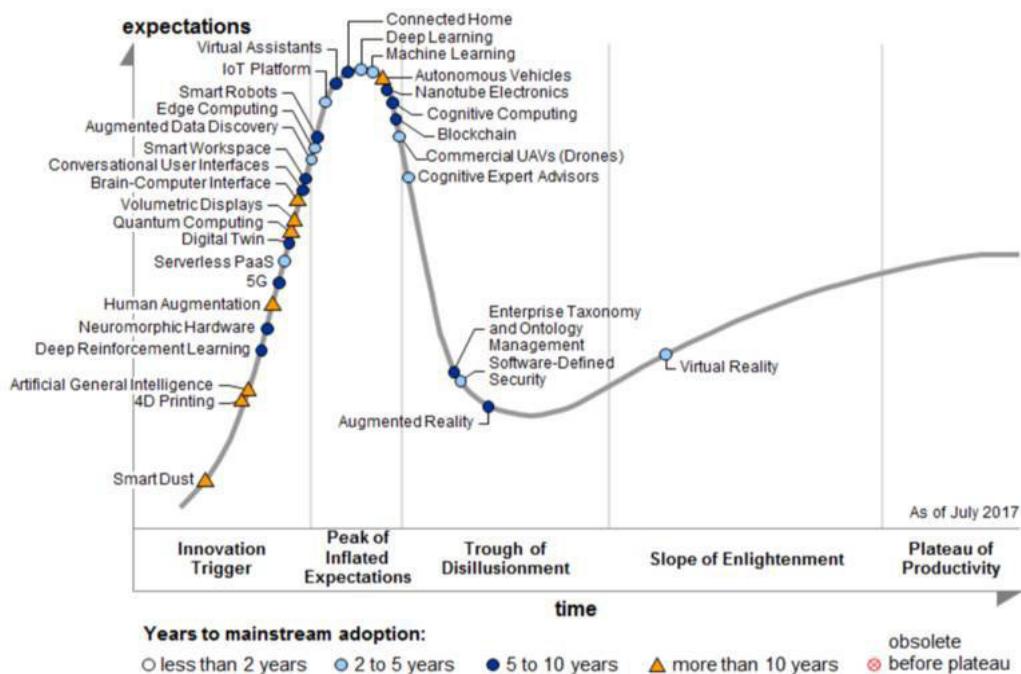
小米很早就开始打造智能家居产品，如今推出智能音箱更像是顺势而为。

很多米粉将米家及其一系列家居产品趣称为“小米全家桶”，这侧面说明了小米这几年积累的智能硬件资源之丰富。虽然米家推出的时间不算早，但依附于小米生态链，整合了小米之前一系列智能产品和几十家生态链公司的智能家居产品，形成了一套相对比较完整的智能家居系统。截至今年 5 月 31 日，基于小米 MIOT 平台的联网设备总量已经突破 6000 万台。当年很多人不理解小米为什么做智能家居产品，如今智能音箱市场火爆，大家都抢着做智能家居入口，在所有人都还在忙着对接更多智能硬件设备时，小米布下的局已经先行启动。

## 智能音箱的未来是什么？

**语音交互势在必行，但是……**

根据 Gartner2017 年最新版技术成熟度曲线图，目前



Note: PaaS = platform as a service; UAVs = unmanned aerial vehicles

Source: Gartner (July 2017)

Conversational User Interface（对话式用户界面）正从科技诞生的促动期步入过高期望的峰值，距离成为主流应用还有 5-10 年。Gartner 报告中将 CUI 列为 2017 年的十大科技趋势之一，报告<sup>[7]</sup>称“随着科技变得能够读懂人心，对话系统将带来下一代信息技术转型。企业架构和技术创新领导者当前必须充分利用可行的应用案例，同时探索未来会话系统存在的机会。”

## CUI 目前最主要的载体是手机、音箱和耳机

对厂商而言，布局智能音箱其实是在为了借此形成入口、输出服务，同时掌控语音交互背后的用户和数据。智能音箱只是当下最适合的载体之一，未来家中所有的电子设备可能都会搭载语音交互模块，届时你将能与电视、冰箱等设备直接对话（P. S. 这不是科幻片预告，在小米电视、美的智能冰箱等设备上已有不少落地案例）。

对于用户而言，语音交互确实更符合人类本能，如果能通过语音交互的统一入口、免去单独到每一个应用上获取对应服务的麻烦，一句话就能便捷高效地完成众多操作，用户自然没有不用的道理。但是现阶段的智能音箱真的能做到“解放双手”吗？

## 技术尚未成熟，谈入口为时尚早

抛开厂商设定，智能音箱本质上就是一款基于语音进行人机交互的智能硬件。播放音乐是传统音箱的主要（几乎是唯一）功能，但对于智能音箱来说，音质只是附加选项，用户更看重人机交互的体验，以及交互背后所能支持和兼容的服务数量与质量。人机交互体验、线上互联网服务和线下智能家居系列产品三者缺其一，智能音箱的入口目标就难以达成。而语音交互技术正是人机交互体验的关键掣肘。

从技术现状和实际产品效果来看，语音交互技术还需要完善，最为关键的自然语言理解尚有众多难关等待突破，各家公司都在艰难地往前探索。智能音箱的实际使用效果必然受限于技术。现在已经发布或者上市并且叫得出名字的智能音箱产品暂时还没有谁甩谁一条街的情况，使用中普遍存

在“动口不如动手”的尴尬场面。

总有做智能音箱的厂商说“用户还没养成语音交互的习惯”、“用户还没做好准备”，用户才是真躺枪，人家倒是想养成习惯，可是你先给整个好用点的语音交互呗？

若要说语音交互存在泡沫，那泡沫主要也是源于各大厂商对语音交互技术成果的盲目夸大，比如家家语音识别准确率都达到 97% 以上（一切不提前置条件和测试数据集光说语音识别率都是要流氓）。在自然语言理解出现重大突破之前，解决噪音问题、提升远场语音识别率才是当务之急。

## 智能音箱还缺什么

智能音箱还缺大屏参与互动。研究表明，在人的感知系统中，视觉所获取的信息占 60% 以上，听觉获取的信息占 20% 左右；而人在沟通中表达的信息 55% 来自肢体语言信息，38% 来自声音信息。

虽然阿里凭借购物场景的优势，为天猫精灵搭载了声纹购功能，但实际上网络购物是典型的离不开屏幕的应用场景，几十秒就能看完的商品描述和评论，智能音箱可能需要几分钟才能念完，更何况眼见为实耳听为虚，不看图片光听几句商品介绍就敢下单的人有几何？语音上场，屏幕却不会消失，融合语音、视觉和肢体动作的交互方式或许更可能成为下一个时代的主宰。

Google IO 2017 上简单演示了使用 Google Home 唤醒 Android TV 并展示信息，智能音箱与智能电视的深度集成可能会成为新的趋势。

除此之外，还需要将语音助手形象化，我们对着空气、对着一个圆柱体说话太傻，需要一个能给予视觉或表情反馈的存在，我们才愿意与语音助手有更多的交流。

## 智能音箱的中国问题：智能音箱能在中国重演安卓的盛况吗？

我的答案：不能。语音助手强烈依赖云端，需要厂商提供服务，而 Android 的核心代码 AOSP 都在本地，可以构建分支。很难想象国内的智能音箱最终都使用同一家公司提供的语音助手。

那么中国的智能音箱市场会变成什么样？

智能音箱是硬件、软件平台、云服务的合体，需要在这三方面都有强大的实力才能做好，目前中国符合这个条件的并不多。创业公司如果使用第三方的语音助手服务，核心技术 操于人手，注定做不大。所以和目前共享模式的利用创业公司进行代理人战争不同，智能音箱需要巨头亲自下场。

巨头有各自的护城河，几乎每家都有自己的音乐和语音内容产品，而智能音箱和这些业务是可以相互促进的，因此只要智能音箱业务没有严重亏损，巨头就不会轻易言弃。所以中国未来智能音箱的市场很可能是在一场混战之后，形成几家割据的局面。

目前的问题是巨头已有布局，但没有人愿意教育市场。

那么问题来了，对于智能音箱，你怎么看？

## 附录

1. <https://www.gartner.com/doc/3021226/market-trends-voice-ui-consumer>
2. <https://www.emarketer.com/Article/Alexa-Say-What-Voice-Enabled-Speaker-Usage-Grow-Nearly-130-This-Year/1015812>
3. <http://www.cninfo.com.cn/finalpage/2017-03-21/1203181704.PDF>
4. <https://www.gelonghui.com/p/132229.html>
5. <http://news.sciencenet.cn/sbhtmlnews/2014/8/291008.shtml>
6. <https://mp.weixin.qq.com/s/lc5uf6vZ1FDWSVWkVHEDKg>
7. <https://www.gartner.com/doc/3645325/top--strategic-technology-trends>

# 聚焦

- 大前端技术与管理
- 新一代 DevOps
- 人工智能与业务应用
- 大数据平台架构
- 架构升级与优化
- 金融应用架构
- 内容分发与精准推荐
- 国际化架构设计
- 互联网产品与创业
- 深入机器学习
- 业务架构
- 工程师文化与团队建设
- 大型游戏架构
- 流媒体架构
- 数据库架构
- 微服务架构

# 分享

## 京东硅谷研究院 | 自动深度语法分析与自然语言应用

如何结合知识图谱和大数据舆情挖掘，展示深度语法分析的原理和威力

## 腾讯 | 黑产对抗与千万在线后台服务演进

核心模块如何不断优化重构，如何应对层出不穷的黑色产业模式与内容

## 百度 | 数据仓库开源架构解读与应用

数百TB乃至PB级别的数据量，如何达到毫秒/秒级分析

## 饿了么 | 移动性能可视化之路

如何实现85%页面秒开，包体积减小20%+

## 58转转 | C2C电商平台推荐系统架构演进

多策略推荐系统架构如何设计以及实时化升级

2017.12.08–09 大会演讲

2017.12.10–11 大会培训

北京 · 国际会议中心



## — 联席主席



崔宝秋  
小米 首席架构师,  
云平台负责人



赵宇辰  
销售易 技术VP  
首席科学家



刘海锋  
京东商城  
首席架构师



张木喜  
musical.ly  
高级技术副总裁



谭待  
百度  
主任架构师

.....

## — 出品人与分享嘉宾



侯兆炜  
Tech Lead  
Manager  
@Facebook



吴惠君  
Data Platform  
/Engineer  
@Twitter



李维 (博士)  
京东硅谷研究院  
主任研究员



何登成  
阿里巴巴  
资深技术专家



黄斯亮  
腾讯音乐  
后台技术总监



姜宁  
华为  
技术专家



牟宇航  
百度  
大数据部技术经理



胡彪  
饿了么  
移动技术总监

.....



### 知名互联网公司系统架构图

如果在使用过程中遇到任何问题, 可联系大会主办方, 欢迎咨询!

微信: aschina666

电话: 15201647919

# 谈谈技术选型的注意事项

作者 周明耀



## 写在前面

对于一名热爱技术的工程师来说，很容易出现非常热衷于使用新技术的情况，记得有一次和一位做平台应用的同事闲聊，他问我最近在搞什么，我说在研究 Hadoop，正在用 MapReduce 处理海量图片的智能分析，他一脸羡慕：“能搞新技术，真好！”。

作为一名工程师，我可以理解大家的心情，我们都是热爱尝试新技术、抛弃过时技术的人。但是首先得明确，到底技术是不是过时的，还是仅仅是认为它过时了。这篇文章我想谈谈我对技术选型的理解。

这篇文章不仅仅是写给工程师，更多是写给技术团队负责人（大多数也是从工程师升职上去的，起初思维和工程师差距不大），因为你们具体负责技术选型的方向、方法、过程、结论明确。

## 技术选型的注意事项

先来看看软件开发领域的变化，变化实在是太快了。在 JavaScript 里，几乎每天都有新框架诞生。Node.js（关键词：事件编程），React 编程，Meteor.js（关键词：共享状态），前端 MVC，React.js…… 你可以随便举例。软件工程领域里新概念也层出不穷：领域驱动开发，六边形架构理论，DCI 架构（数据 - 场景 - 交互）。

洛克希德·马丁公司的著名飞机设计师凯利·约翰逊所提出的 KISS 原则，指出架构设计能简单绝不复杂，坚决砍掉任何华而不实的设计，不要因为 3 年后可能怎样甚至是一些现实中根本无法出现的场景，加入到当下的架构设计中，导致系统无比复杂。有时候看似引入的是一个很简单很容易解决的问题，可能在具体的执行过程中带来一系列不必要的麻烦。技术选型其实遇到的问题和系统架构设计类似，也容易出现人为因素导致的偏差，进而出现和系统架构过度设计类似的麻烦。

对于技术选型，有以下几个建议。

## 选择你最熟悉的技术

记得看过一篇文章，里面提到一个新项目最好不要使用超过 30% 的新技术，我觉得这有一定道理，因为对于你完全不知道的技术，你不可能控制使用过程中出现的风险。我在技术管理中的向下管理里提起过，任何一位技术 Leader，如果你不能得到下属的技术尊重，你必将受到惩罚。

也不能说完全不能使用新技术，前几天和朋友聊天，他提到了另外一位总监下属有几个人转岗了，都是技术牛人，最主要的原因是这位总监坚决排斥新技术，坚持自己熟悉的十年前的框架和编写代码规范。他对于一个新技术的天然不信任，在技术接受程度还不够高，并且认为公司内没

有人能吃透这个技术的情况下，不愿意让自己的业务做第一个吃螃蟹的人，这种做法不能说完全错误，至少对于他自己来说很稳健，但是却压制了一些有追求人的内心。

谨慎是个美德，不过如果在一个非常追求速度的业务里，这可能也意味着过于保守，会延误时机。

那我们应该怎样做到选择技术呢？我认为，在选择技术时有两个大原则。第一，要取其长避其短；第二，要关注技术的发展前景。每种技术都是有它特定的适用场景，开发者经常犯的错误就是盲目追新，当一个新语言、框架、工具出现后，特别是开发者自己学会了这种新技术后，就会有种“拿着锤子找钉子”的感觉，将新技术滥用于各种项目。

记住，技术选型是稳定压倒一切。

## 选择拥有强大社区支撑的开源技术

没有人喜欢“alone in the dark”的感觉，同样，也很少有工程师喜欢孤独地面对代码缺陷。我们之所以喜欢在 Apache 上挑选合适的新框架尝试使用，是因为 Apache 始终保持运作着强大的社区，每天都有很多新建的框架，也设计了一整套生命周期管理标准，让一个项目能够从孵化项目逐渐一步步地走向顶级项目。除了像 Apache 这样的社区，我们也可以评估是否存在一些商业公司提供针对该技术或者框架的有偿支撑，一般来说，有公司愿意围绕该技术布局，也能说明确实存在使用空间。例如 Apache Cassandra，目前就有 Datastax 和 LastPickle 两家公司对它提供技术指导和有偿辅助软件支撑。

其实看一项技术活不活跃，只要去 StackOverflow 这样的网站看看提问的人多不多就知道了。

## 确保技术前进步伐

选择一个技术的最低标准是，技术的生命周期必须显著长于项目的生命周期。

为什么需要确保所选择的技术不断前进？因为这个世界是发展的，科技发展更是非常得快速，你可以看看，所有的成功的科技公司都是因为跑在了别人前面，而不是慢悠悠的工作态度，这就是科技界的残酷，也正是因为 FaceBook 办公室里贴着：“要么做到最好，要么死亡”。

技术的前进不仅仅取决于它本身，而是和大环境发展、上下游用户也密切相关。比如 AI，60 年代其实就已经提出了相应概念，为什么直到今年才进入发展元年？因为芯片的计算效率、数据样本规模没有达到要求。而 Functional Language 为什么这么多年一直默默无闻，而从前几年开始逐渐盛行？因为机器学习来了，AI 来了，它们有了用武之地。

总的来说，你需要使用你所选择的软件技术，快速地实现应用程序的构建。记住一句话：好的技术栈永远跑在用户需求前面。

## 学会从业务端开始思考

技术选型必须贴着业务来选择，不同业务阶段会有不同的选型方式。处于初创期的业务，选型的基准是灵活。只要一个技术够用并且开发效率足够高，那么就可以选择它。初创的业务往往带有风险性和不确定性，朝令夕改、反复试错是常态，技术必须适应业务的节奏，然后才是其他方面。等业务进入稳定期，选型的基准是可靠。技术始终是业务的基石，当业务稳定了技术不稳，那就会成为业务的一块短板，就必须修正。当业务进入维护期，选型的基准是妥协。代码永远有变乱的趋势，一般经过一两年就有必要对代码来一次大一点的重构。在这种时候，必须得正视各种遗留代码的迁移成本，如果改变技术选型会带来遗留代码重写，这背后带来的代价业务无法承受，那么我们就不得不考虑在现有技术选型之上做一些小修小补或者螺旋式上升的重构。

正因为技术选型和业务相关，我们能够观察到一些很明显的现象：新技术往往被早期创业团队或大公司的新兴业务使用；中大型公司的核心业务则更倾向于用一些稳定了几年的技术；一个公司如果长期使用一种技术，就会倾向于一直使用下去，甚至连版本都不更新的使用下去。这现象背后

都是有道理的。

回到我们的主题，学会从业务端思考。首先我们需要充分地理解业务，理解用户需求，理解当下需要解决的首要问题，以及可能的风险有哪些，再将目标进行分解，进行具体的技术选型、模型设计、架构设计。

举个例子。假设我们需要解决的核心问题是并发，则可以通过各种缓存手段（本地缓存、分布式缓存），来提高查询的吞吐，这样虽然会一定程度上需要在数据一致性上做出牺牲，由强一致性变为最终一致性，但是，如果数据一致性不是核心需要解决的问题，那么，此问题的优先级则可以先放一放，反过来如果核心问题变为数据的一致性，如交易系统，那么再怎么强调数据的一致性都不为过，由于分布式环境下为了应对高并发的写入以及海量数据的存储，通常需要对关系型数据库进行分库分表扩展，这也给数据一致性带来了很大的挑战，原本的单库事务的强一致性保障，在这个时候升级为跨库的分布式事务，而通过二阶段或者三阶段提交所保障的分布式事务，由于分布式事务管理器与资源管理器之间的多次网络通信成本，吞吐及效率上很难满足高并发场景下的要求，而这实际上对于交易系统来说，又是一个很难回避的问题，因此，大家又想出很多的招来解决这个问题，通过可靠消息系统来保障不失为一种方式，变同步为异步，但是，又引入新的问题，消息系统为保证不丢消息，则很难保证消息的顺序性以及是否重复投递，这样作为消息的接收方，则需要保障消息处理的幂等性，以及对消息去重。

## 先验证，后使用

对于未经验证的新技术、新理念的引入一定要慎重，一定要在全方位的验证过后，再大规模的使用。新技术、新理念的出现，自然有它的诱惑，慎重并不代表保守，技术总是在不断前进，拥抱变化本身没有问题，但是引入不成熟的技术看似能带来短期的收益，但是它的风险或者是后期的成本可能远远大于收益。

## 重视经验

技术选型是个很需要经验的活，得有大量的信息积累和输入，再根据具体现实情况输出一个结果。我们在选型的时候最忌讳的是临时抱佛脚、用网上收集一些碎片知识来决策，这是非常危险的，我们得确保自己所有思考都是基于以前的事实，还要弄清楚这些事实背后的假设，这都需要让知识内化形成经验。

经验的本质是什么，有什么方法能够确定自己的经验增长了，而不是不断在重复一些很熟悉的东西。我现在的结论是，经验等于知识索引的完备程度。

我们一生中会积累很多的知识，如果把我们的大脑比作数据库的话，那我们一定有一部分脑存储贡献给了内容的索引，它能帮助我们将关联知识更快的取出来，并且辅助决策。经验增长等同于我们知识索引的增长，意味着我们能轻易的调动更多的关联知识来做更全面的决策。

要想建立好这个知识索引，我们得保持技术敏感性和广度，也就是要做到持续的信息输入、内化，并发现信息之间的关联性，建立索引，记下来。说起来容易，做起来还是挺有难度的。

首先难在信息输入量大，忘记了怎么办。我们的大脑不是磁盘，不常用的知识就会忘记，忘记了就跟没看过是一回事。我的经验是一定要对知识进行压缩，记住的是最关键的细节，并且反复的去回味这个细节。

## 我的实际案例

去年我做了一次对于分布式数据库的选型工作。我们为什么要做这次选型？因为存在明确的需求，我们需要解决大规模高并发数据存储，单次数据不大，但是存储频率、读取频率都很高，并且要确保不丢失数据，这样的需求对于关系型数据库来说，出现了性能瓶颈。

我对于技术选型有自己的一套方法论，我知道，我不可能什么技术都懂，所以我会按照自己的这套方法论来具体执行，避免出现选型误差。我

的步骤是：“列出需求”–“细分需求”–“明确搜索方向”–“网络搜索”–“明确评判标准”–“分头执行”–“汇总材料”–“初步选择”–“进一步调研”–“会议评审” – “做出决定”。这些步骤太多，需求我已经介绍了，这里具体再讲讲我这一次是如何进入下一步选型的，也就是“初步选择” – “进一步调研”之间的过程。

我通过网络搜索（进入 Google，搜索 Distributed Database、NoSQL Database 等关键词），我找到了如下这些国内外专家推荐的分布式数据库，他们的基本描述如下所示：

- HyperTable：一个开源、高性能、可伸缩的数据库，它采用与 Google 的 BigTable 相似的模型。该数据库数据按主键在物理上排序，适用于数据分析领域，采用 C++ 编写，可以运行在 HDFS 上面。该数据库受到 GPLv3 协议约束，考虑到它和 HBase 从系统架构上来说很相似，但是协议约束较多，所以放弃调研，转而调研 HBase。
- HBase：即 Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，采用主/从架构设计，利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。它是 Google BigTable 的开源实现。
- VoltDB：一个内存数据库，提供了 NoSQL 数据库的可伸缩性和传统关系型数据库系统的 ACID 一致性，支持单节点 53000 TPS/s。该数据库受到 GPLv3 协议约束。VoltDB 有两个版本，一个开源社区版本和一个付费企业版本。付费企业版本除包含了所有开源社区版的功能，还有些其他特点，诸如计算机集群管理控制台、系统性能仪表盘、数据库宕机恢复、在线数据库 Schema 修改、在线数据库节点重新加入、JDBC 和 OLAP 导出支持、命令日志。
- 由于该框架开源社区不活跃，主导者更加希望使用付费版本，所以决定放弃它，转而调研类似的 Redis。
- CloudData：一个结构化数据库，没有中文资料，从系统架构、功能上分析，类似于 MongoDB。

- Gridool：一种基于MapReduce原理设计的网格计算引擎，不支持数据存储，所以放弃。
- Ddb-query-optimizer：找不到资料，放弃。
- Cages：基于ZooKeeper实现数据协调/同步，不仅性能数据分布式存储，放弃。
- Redis：一个开源的基于键值对和存储系统，具备高性能特征。支持主从复制（master-slave replication），并且具有非常快速的非阻塞首先同步（non-blocking first synchronization）、网络断开自动重连等功能。同时Redis还具有其他一些特征，其中包括简单的check-and-set机制、pub/sub和配置设置等，以便使得Redis能够表现得更像缓存（Cache）。绝大部分主流编程语言都有官方推荐的客户端。
- MongoDB：一个开源的C++编写的面向集合且模式自由的文档型数据库，是NoSQL中功能最丰富、最像关系型数据库的产品。核心优势：灵活文档模型+高可用复制集+可扩展分片集群；功能特点：二级索引、地理位置索引、aggregate、map-reduce、GridFS支持文件存储。不足之处：不支持事务，仅支持简单left join。
- Spanner：Google的可扩展的、多版本的、全球分布式的同步复制方式数据库。Spanner是第一个支持全球规模的分布式数据、外部一致性分布式事务的分布式数据库。它是一个在遍布全球范围的数据中心内部通过多套Paxos状态机器共享数据的数据库。复制被用于全局可用性和地理位置；客户在副本之间自动切换。当数据量或者服务器数量发生变化时，Spanner在机器之间自动共享数据，并且Spanner在机器之间自动迁移数据（甚至在数据中心之间），用以负载均衡和响应失败。Spanner被设计为在几百万台机器之上横向扩展，这些扩展穿过了数百个数据中心和万亿行数据。功能很强大，可以没有开源。
- ElasticSearch：一个基于Lucene的搜索服务器。它提供了一个

分布式多用户能力的全文搜索引擎，基于Restful Web接口。

ElasticSearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是当前流行的企业级搜索引擎。

最终通过这些技术之间的互相相似度对比，并且我们设定了一些规则，例如开源协议的约束，这一点其实逐渐开始真正起到约束了，看看FaceBook针对Reactor的专利约束给大家造成的麻烦，你就懂了。最终，我选择了Cassandra、MongoDB、Redis、MySQL、HBase等几款进入下一步深入调研。

## 写在后面

我们进行技术选型，有的团队会根据社交媒体上的讨论来决定选择哪种架构，有的团队会跟风走，哪个热门就选哪个，这些都不是正确的方式，我们应该按照方法论执行。此外，我们作为团队管理者，一边要督促自己不断学习新技术，自己能够上手使用，也要结合实际团队情况，规划新技术的预研、落地步骤，让团队成员既能享受到稳定技术的红利，也能不断地尝试新事物，让大家能够看到未来，不担心自己逐渐落后于行业的发展，更能提升对于公司的归属感。做到这些，真不容易，加油，诸位。

## 作者介绍

**周明耀**，2004年毕业于浙江大学，工学硕士。13年软件研发经验，近10年技术团队管理经验，4年分布式计算、大数据技术经验。出版书籍包括《大话Java性能优化》、《深入理解JVM&G1 GC》、《技术领导力—码农如何才能带团队》。个人微信号michael\_tec，个人公众号“麦克叔叔每晚10点说”。

# 王跃：关于微信小程序的技术，也许你想错了

作者 徐川



2017年1月9日，微信小程序正式发布。在近一年里，小程序一直在坚定的向前走。它的理念和模式受到广泛认可，也被其他人所模仿。

在微信小程序尚在内测之时，外界对它所采用的技术就有很多猜测，正式发布的小程序解答了人们的一些疑惑，但有些问题官方并未正式对外公开说过。InfoQ对微信小程序相关项目负责人王跃进行了采访，了解了一些开发者关心的问题。

## 受访嘉宾介绍

王跃（微信号：springwang），微信小程序相关项目负责人，拥有

10+ 年前端开发经验，曾就职于搜狐和新浪，2013 年加入腾讯，负责互娱游戏营销系统，道聚城等多个项目前端架构和开发。对小程序底层架构原理有深入的研究和理解，并且有腾讯多款小程序开发实战经验。

**InfoQ：王老师好，您在负责小程序前端之前，做过哪些事情？**

**王跃：**在微信小程序项目之前，我负责过腾讯互娱游戏高级营销系统的前端架构和开发，它承载腾讯几百款游戏业务的日常营销活动，另外还有腾讯道聚城前端架构和开发，覆盖像王者荣耀，LOL，CF 游戏道具的交易，在腾讯之前我还负责过搜狐白社会 SNS 的前端核心框架和模块开发，新浪微博的前端开发工作。

**InfoQ：当时小程序还没发布时，坊间传说小程序使用了类似 RN 的技术，发布后人们发现它还是运行在 WebView 里的，不知道实际情况如何？**

**王跃：**从技术实现的层面来说，不管是小程序，还是 RN，或者 Weex，都有共同点，比如 JS 和 Native 的通讯机制，比如 JS 直接调用原生组件的渲染，如在 iOS 平台，小程序和 RN 都采用 JavaScriptCore 来执行 JS。但是小程序和 RN 设计初衷和应对的场景不一样，我们知道小程序的场景主要是在当前实际物理场景用户可以即扫即用，用完即走，整个交互都是非常轻量级的，不涉及特别复杂的交互逻辑，所以在设计上考虑尽量简单，首先是系统底层框架简单，其次开发者开发简单，再次用户使用简单，所以小程序大部分的 UI 组件还是 H5 的渲染方式，而不是像 RN 设计成 Native 的 UI 组件。

当然小程序本身为了解决部分组件性能的问题也采用了 Native 的方式，所以方案上的选项主要是基于实际场景考虑，不是纯技术上的考量。

另外准确的说小程序不仅仅运行在 Webview 里，需要区分不同的部分，这个在我的分享里会有详细的解释。

**InfoQ：在 Android 上小程序是运行在 X5 引擎上的，X5 团队有为小程序做一些特别的优化，或者添加特性吗？**

**王跃：**微信 Android 版的浏览服务用的确实是我们腾讯浏览器团队提

供的 X5 引擎，在性能方面小程序和 X5 团队之间一直有保持沟通和协调，双方都尽可能设法优化并持续提升用户体验。

**InfoQ：刚发布时有人发现小程序的一些代码和 Vue 的有点像，而单向数据流又让人联想到 React，在当初开发小程序核心框架的时候有哪些思考？**

**王跃：**这个跟问题 2 类似，首先小程序和 Vue，React 本质上还是不一样的，小程序是需要特定的 Native 层支持，同时底层功能也更强大，而 Vue 和 React 运行在通用的 WebView 之上，不需要特定 Native 支持，但大家为什么觉得会有些类似呢，主要是指在数据绑定，事件绑定等部分的实现上会有一些类似，当然这几种技术没有好坏，主要还是看我们是解决什么场景下的什么问题。

**InfoQ：iOS 和 Android 平台的小程序有一些区别，比如 Android 上可以把小程序图标放到主屏，还有人发现微信小程序在 Android 下有单独的进程，小程序是不是对 Android 进行过更多 Native 化的探索？**

**王跃：**Android 可以放到主屏幕而 iOS 不行这个主要是 OS 层面的限制，至于 Android 下的运行方式，主要是通过单独的 Activity 来承载视图，设置为单独的进程主要是为了保证小程序的运行内存，跟 Native 化没有直接的联系。

前面问题也提到了，小程序本身已经有好几个组件是使用 Native 方式实现的，主要目的还是为了保证小程序的执行效率，达到更好的用户体验，Native 的组件也不是针对 Android 一个平台，Android 和 iOS 都有做，后续是否会有更多的 Native 化的实现，还是看实际组件在采用 Web 实现时是否符合我们对用户体验的标准。

**InfoQ：前段时间有人发现小程序出了自己的脚本格式 WXS，它是小程序新的 DSL 吗？**

**王跃：**目前，WXS 对于小程序开发不是必须的，它的主要目的是为了增强 WXML 的数据处理能力而新引入一种技术实现，其实际解析的语言规范还是 JS，并没有引入新的语法，仅仅对 JS 做了上层的封装和限制，所

以学习上基本没什么成本，大致了解下开发文档马上就能上手，这里 WXS 跟 DSL 也没太大关系，没有可对比性。

**InfoQ：小程序和 PWA 可以说代表着移动 Web 的两条不同的发展路线，从旁人的眼光来看，小程序更加务实，但人们也期待小程序更加开放一些。在这方面您是怎么看的？**

**王跃：**这里我说下个人的想法，不代表官方意见，任何一种模式都是为了在特定环境下解决特定问题而设计的，所以 PWA 有它的应用场景，而小程序有小程序的应用场景，两种模式都有其优势和限制，这两种模式的差异其实跟我们现在的 Web 和 Native 很像，Web 提供相对常用和通用的功能（大部分功能和基本使用体验），而个性定制（更流程复杂的功能和交互体验）可以充分发挥当前平台的能力，我个人觉得这两种模式都会一直存在，关键是看能否为用户提供价值，不过，未来这两种模式一定会有越来越多的融合，就像 web 和 Native 的融合产生了 Hybird 模式一样，想象一下，未来一定会有一种新的模式，可以像 PWA 一样具有更通用的运行场景（提供核心功能），同时又可以根据当前的运行环境接入定制化的高级能力，实现 Write Once，Run Anywhere 的美好愿景。

# 体系化认识 RPC

作者 张旭



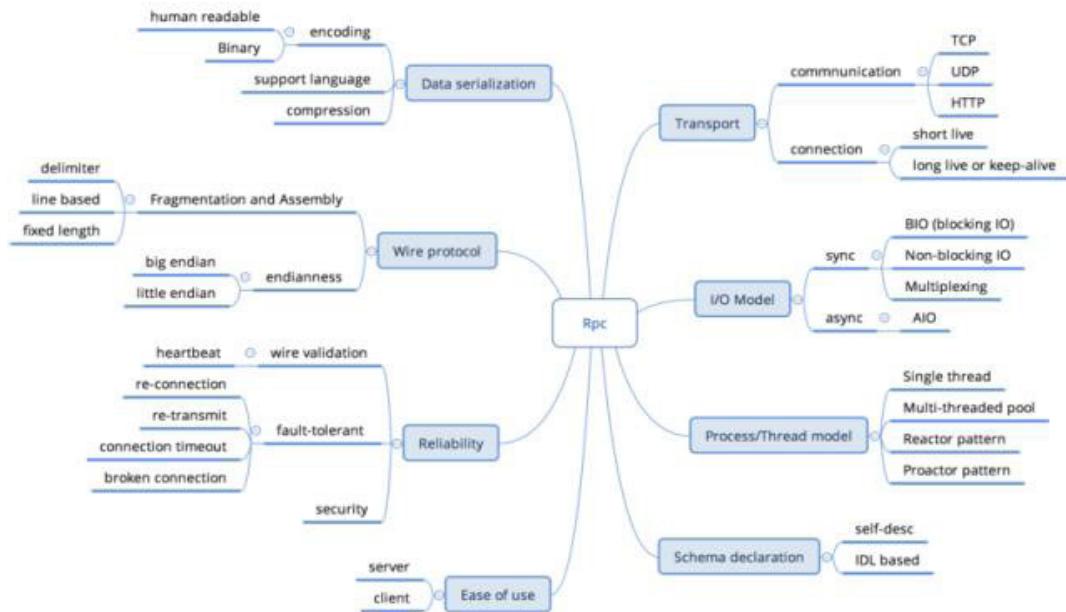
RPC (Remote Procedure Call)，即远程过程调用，是一个分布式系统间通信的必备技术，本文体系性地介绍了 RPC 包含的核心概念和技术，希望读者读完文章，一提到 RPC，脑中不是零碎的知识，而是具体的一个脑图般的体系。本文并不会深入到每一个主题剖析，只做提纲挈领的介绍。

RPC 最核心要解决的问题就是在分布式系统间，如何执行另外一个地址空间上的函数、方法，就仿佛在本地调用一样，个人总结的 RPC 最核心的概念和技术包括如下，如图所示。

下面依次展开每个部分。

## 传输 (Transport)

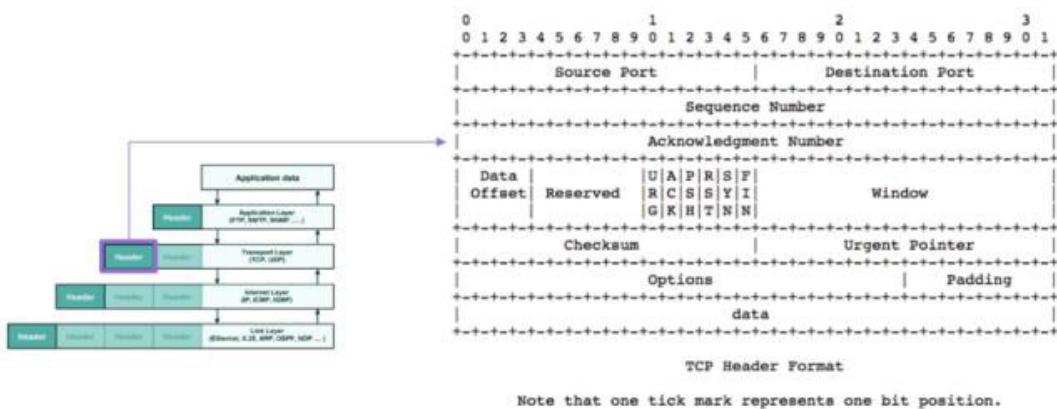
TCP 协议是 RPC 的 基石，一般来说通信是建立在 TCP 协议之上的，



而且 RPC 往往需要可靠的通信，因此不采用 UDP。

这里重申下 TCP 的关键词：面向连接的，全双工，可靠传输（按序、不重、不丢、容错），流量控制（滑动窗口）。

另外，要理解 RPC 中的嵌套 header+body，协议栈每一层都包含了下一层协议的全部数据，只不过包了一个头而已，如下图所示的 TCP segment 包含了应用层的数据，套了一个头而已。



那么 RPC 传输的 message 也就是 TCP body 中的数据，这个 message 也同样可以包含 header+body。body 也经常叫做 payload。

TCP 就是可靠地把数据在不同的地址空间上搬运，例如在传统的阻塞

I/O 模型中，当有数据过来的时候，操作系统内核把数据从 I/O 中读出来存放在 kernel space，然后内核就通知 user space 可以拷贝走数据，用以腾出空间，让 TCP 滑动窗口向前移动，接收更多的数据。

TCP 协议栈存在端口的概念，端口是进程获取数据的渠道。

## I/O 模型 (I/O Model)

做一个高性能 /scalable 的 RPC，需要能够满足：

- 第一，服务端尽可能多的处理并发请求
- 第二，同时尽可能短的处理完毕。

CPU 和 I/O 之间天然存在着差异，网络传输的延时不可控，最简单的模型下，如果有线程或者进程在调用 I/O，I/O 没响应时，CPU 只能选择挂起，线程或者进程也被 I/O 阻塞住。

而 CPU 资源宝贵，要让 CPU 在该忙碌的时候尽量忙碌起来，而不需要频繁地挂起、唤醒做切换，同时很多宝贵的线程和进程占用系统资源也在做无用功。

Socket I/O 可以看做是二者之间的桥梁，如何更好地协调二者，去满足前面说的两点要求，有一些模式 (pattern) 是可以应用的。

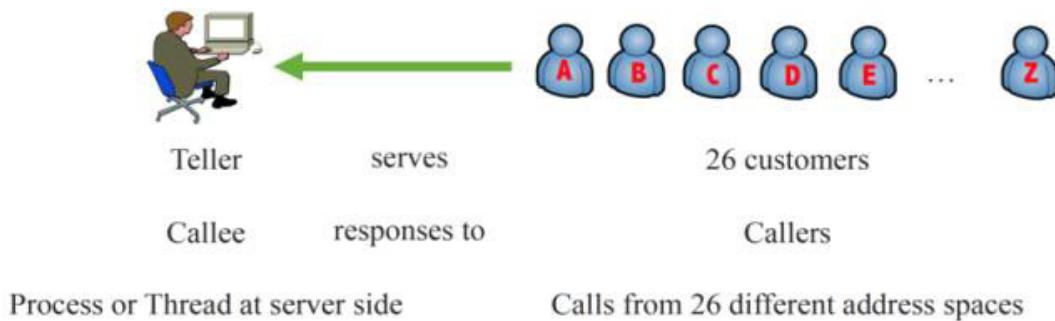
RPC 框架可选择的 I/O 模型严格意义上 5 种，这里不讨论基于信号驱动 的 I/O (Signal Driven I/O)。这几种模型在《UNIX 网络编程》中就有提到了，它们分别是：

1. 传统的阻塞 I/O (Blocking I/O)
2. 非阻塞 I/O (Non-blocking I/O)
3. I/O 多路复用 (I/O multiplexing)
4. 异步 I/O (Asynchronous I/O)

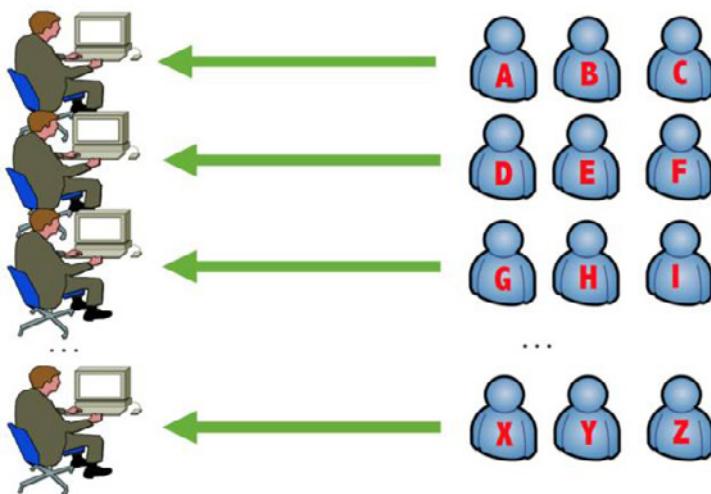
这里不细说每种 I/O 模型。这里举一个形象的例子，读者就可以领会这四种 I/O 的区别，就用 银行办业务 这个生活的场景描述。

下图是使用 传统的阻塞 I/O 模型。一个柜员服务所有客户，可见当客户填写单据的时候也就是发生网络 I/O 的时候，柜员，也就是宝贵的

线程或者进程就会被阻塞，白白浪费了 CPU 资源，无法服务后面的请求。



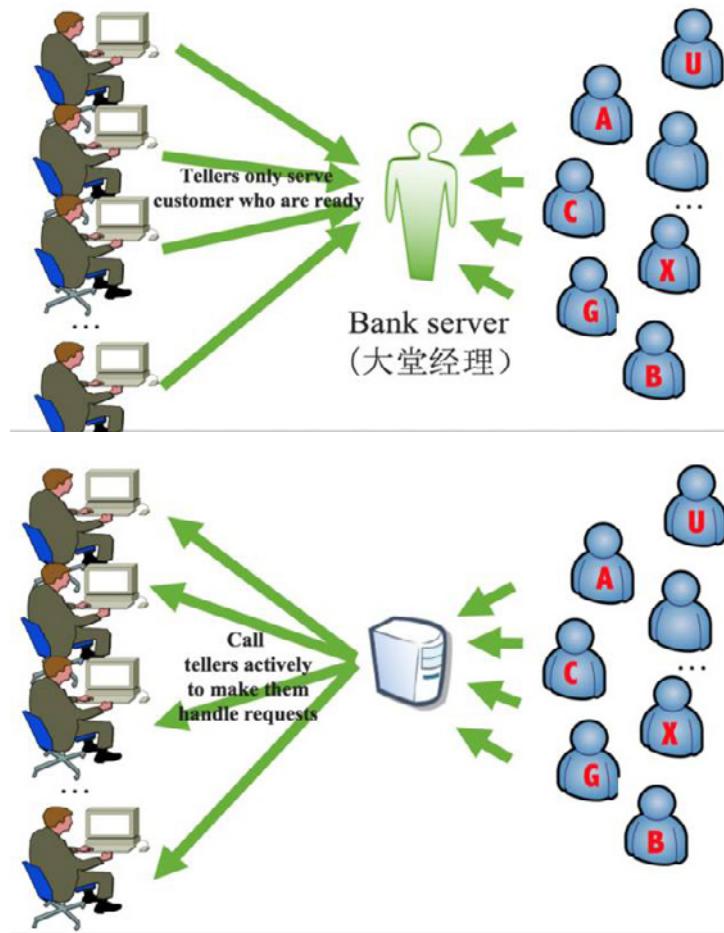
下图是上一个的进化版，如果一个柜员不够，那么就 并发处理，对应采用线程池或者多进程方案，一个客户对应一个柜员，这明显加大了并发度，在并发不高的情况下性能够用，但是仍然存在柜员被 I/O 阻塞的可能。



下图是 I/O 多路复用，存在一个大堂经理，相当于代理，它来负责所有的客户，只有当客户写好单据后，才把客户分配一个柜员处理，可以想象柜员不用阻塞在 I/O 读写上，这样柜员效率会非常高，这也就是 I/O 多路复用的精髓。

下图是 异步 I/O，完全不存在大堂经理，银行有一个天然的“高级的分配机器”，柜员注册自己负责的业务类型，例如 I/O 可读，那么由这个“高级的机器”负责 I/O 读，当可读时候，通过 回调机制，把客户

已经填写完毕的单据主动交给柜员，回调其函数完成操作。



重点说下高性能，并且工业界普遍使用的方案，也就是后两种。

## I/O 多路复用

基于内核，建立在 epoll 或者 kqueue 上实现，I/O 多路复用最大的优势是用户可以在一个线程内同时处理多个 Socket 的 I/O 请求。用户可以订阅事件，包括文件描述符或者 I/O 可读、可写、可连接事件等。

通过一个线程监听全部的 TCP 连接，有任何事件发生就通知用户态处理即可，这么做的目的就是 假设 I/O 是慢的，CPU 是快的，那么要让用户态尽可能的忙碌起来去，也就是最大化 CPU 利用率，避免传统的 I/O 阻塞。

## 异步 I/O

这里重点说下同步 I/O 和异步 I/O，理论上前三种模型都叫做同步 I/O，同步是指用户线程发起 I/O 请求后需要等待或者轮询内核 I/O 完成后再继续，而异步是指用户线程发起 I/O 请求直接退出，当内核 I/O 操作完成后会通知用户线程来调用其回调函数。

## 进程 / 线程模型 ( Thread/Process Model )

进程 / 线程模型往往和 I/O 模型有联系，当 Socket I/O 可以很高效的工作时候，真正的业务逻辑如何利用 CPU 更快地处理请求，也是有 pattern 可寻的。这里主要说 Scalable I/O 一般是如何做的，它的 I/O 需要经历 5 个环节：

Read -> Decode -> Compute -> Encode -> Send

使用传统的阻塞 I/O + 线程池的方案 (Multitasks) 会遇 C10k 问题。

[https://en.wikipedia.org/wiki/C10k\\_problem](https://en.wikipedia.org/wiki/C10k_problem)

但是业界有很多实现都是这个方式，比如 Java web 容器 Tomcat/Jetty 的默认配置就采用这个方案，可以工作得很好。

但是从 I/O 模型可以看出 I/O Blocking is killer to performance，它会让工作线程卡在 I/O 上，而一个系统内部可使用的线程数量是有限的（本文暂时不谈协程、纤程的概念），所以才有了 I/O 多路复用和异步 I/O。

I/O 多路复用往往对应 Reactor 模式，异步 I/O 往往对应 Proactor。

Reactor 一般使用 epoll+ 事件驱动 的经典模式，通过 分治 的手段，把耗时的网络连接、安全认证、编码等工作交给专门的线程池或者进程去完成，然后再去调用真正的核心业务逻辑层，这在 \*nix 系统中被广泛使用。

著名的 Redis、Nginx、Node.js 的 Socket I/O 都用的这个，而

Java 的 NIO 框架 Netty 也是，Spark 2.0 RPC 所依赖的同样采用了 Reactor 模式。

Proactor 在 \*nix 中没有很好的实现，但是在 Windows 上大放异彩（例如 IOCP 模型）。

关于 Reactor 可以参考 Doug Lea 的 [PPT](#)。

以及这篇 [paper](#)。关于 Proactor 可以参考这篇 [paper](#)。

说个具体的例子，Thrift 作为一个融合了序列化 +RPC 的框架，提供了很多种 Server 的构建选项，从名称中就可以看出他们使用哪种 I/O 和线程模型。

Thrift server type	Description	IO type
<b>TSimpleServer</b>	A single-threaded server using std blocking io. $\otimes$	Blocking IO
<b>TThreadPoolServer</b>	A multi-threaded server using std blocking io.	Blocking IO
<b>TNonblockingServer</b>	A multi-threaded server using non-blocking io, TFrameTransport must be used with this server.	IO Multiplexing Reactor - Single thread
<b>THsHaServer</b>	HsHa = Half-sync Half async. IO multiplexing. Reactor based.	IO Multiplexing Reactor – Worker thread pool
<b>TThreadedSelectorServer</b>	Multi-Reactor based server with two work thread pools, mainReactor for accept, subReactor for read/write.	IO Multiplexing Multi-reactor

## Schema 和序列化 ( Schema & Data Serialization )

当 I/O 完成后，数据可以由程序处理，那么如何识别这些二进制的数据，是下一步要做的。序列化和反序列化，是做对象到二进制数据的转换，程序是可以理解对象的，对象一般含有 schema 或者结构，基于这些语义来做特定的业务逻辑处理。

考察一个序列化框架一般会关注以下几点：

- Encoding format。是 human readable 还是 binary。
- Schema declaration。也叫作契约声明，基于 IDL，比如

Protocol Buffers/Thrift，还是自描述的，比如 JSON、XML。另外还需要看是否是强类型的。

- 语言平台的中立性。比如 Java 的 Native Serialization 就只能自己玩，而 Protocol Buffers 可以跨各种语言和平台。
- 新老契约的兼容性。比如 IDL 加了一个字段，老数据是否还可以反序列化成功。
- 和压缩算法的契合度。跑 benchmark 和实际应用都会结合各种压缩算法，例如 gzip、snappy。
- 性能。这是最重要的，序列化、反序列化的时间，序列化后数据的字节大小是考察重点。

序列化方式非常多，常见的有 Protocol Buffers，Avro，Thrift，XML，JSON，MessagePack，Kyro，Hessian，Protostuff，Java Native Serialize，FST。

下面详细展开 Protocol Buffers（简称 PB），看看为什么作为工业界用得最多的高性能序列化类库，好在哪里。

首先去官网查看它的 [Encoding format](#)。

紧凑高效 是 PB 的特点，使用字段的序号作为标识，而不是包名类名（Java 的 Native Serialization 序列化后数据大就在于什么都一股脑放进去），使用 varint 和 zigzag 对整型做特殊处理。

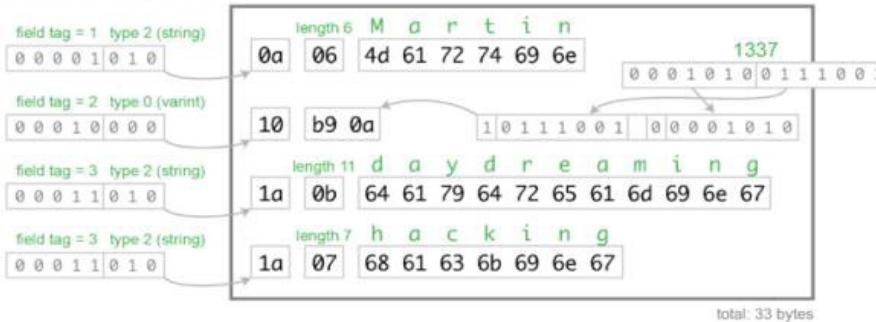
PB 可以跨各种语言，但是前提是使用 IDL 编写描述文件，然后 codegen 工具生成各种语言的代码。

举个例子，有个 Person 对象，包含内容如下图所示，经过 PB 序列化后只有 33 个字节，可以对比 XML、JSON 或者 Java 的 Native Serialization 都会大非常多，而且序列化、反序列化的速度也不会很好。记住这个数据，后面 demo 的时候会有用。

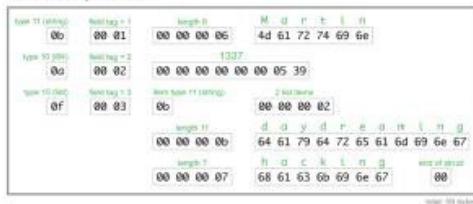
再举个例子，使用 Thrift 做同样的序列化，采用 Binary Protocol 和 Compact Protocol 的大小是不一样的，但是 Compact Protocol 和 PB 虽然序列化的编码不一样，但是同样是非常高效的。

Person.json	Person.proto
<pre>{   "userName": "Martin",   "favouriteNumber": 1337,   "interests": ["daydreaming",   "hacking"] }</pre>	<pre>message Person {   required string user_name      = 1;   optional int64 favourite_number = 2;   repeated string interests      = 3; }</pre>

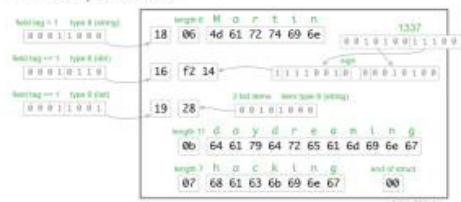
### Protocol Buffers



Thrift BinaryProtocol



Thrift CompactProtocol



这里给一个 Uber 做的序列化框架比较。

可以看出 Protocol Buffers 和 Thrift 都是名列前茅的，但是这些 benchmark 看看就好，知道个大概，没必要细究，因为样本数据、测试环境、版本等都可能会影响结果。

## 协议结构 (Wire Protocol)

Socket 范畴里讨论的包叫做 Frame、Packet、Segment 都没错，但是一般把这些分别映射为数据链路层、IP 层和 TCP 层的数据包，应用层的暂时没有，所以下文不必计较包怎么翻译。

协议结构，英文叫做 wire protocol 或者 wire format。TCP 只是 binary stream 通道，是 binary 数据的可靠搬用工，它不懂 RPC 里面包装的是什么。而在一个通道上传输 message，势必涉及 message 的识别。

举个例子，正如下图中的例子，ABC+DEF+GHI 分 3 个 message，也就是分 3 个 Frame 发送出去，而接收端分四次收到 4 个 Frame。

Socket I/O 的工作完成得很好，可靠地传输过去，这是 TCP 协议保证的，但是接收到的是 4 个 Frame，不是原本发送的 3 个 message 对应的 3 个 Frame。



这种情况叫做发生了 TCP 粘包和半包 现象，AB、H、I 的情况叫做半包，CDEFG 的情况叫做粘包。虽然顺序是对的，但是分组完全和之前对应不上。

这时候应用层如何做语义级别的 message 识别是个问题，只有做好了协议的结构，才能把一整个数据片段做序列化或者反序列化处理。

一般采用的方式有三种：

- 方式 1：分隔符。
- 方式 2：换行符。比如 memcache 由客户端发送的命令使用的是文本行\r\nn 做为 mesage 的分隔符，组织成一个有意义的 message。



- 方式 3：固定长度。RPC 经常采用这种方式，使用 header+payload 的方式。

比如 HTTP 协议，建立在 TCP 之上最广泛使用的 RPC，HTTP 头中肯定有一个 body length 告知应用层如何去读懂一个 message，做 HTTP 包的识别。

在 HTTP/2 协议中，详细见 [Hypertext Transfer Protocol Version](#)



## 2 (HTTP/2)。

虽然精简了很多，加入了流的概念，但是 header+payload 的方式是绝对不能变的。

### 4.1. Frame Format

All frames begin with a fixed 9-octet header followed by a variable-length payload.

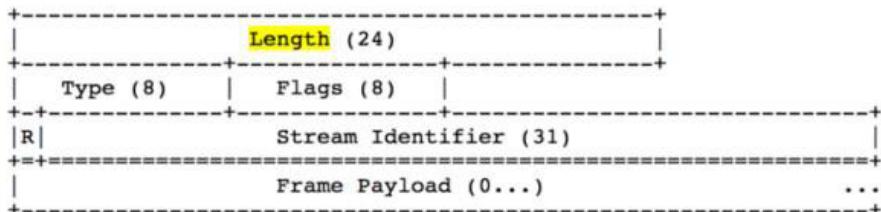
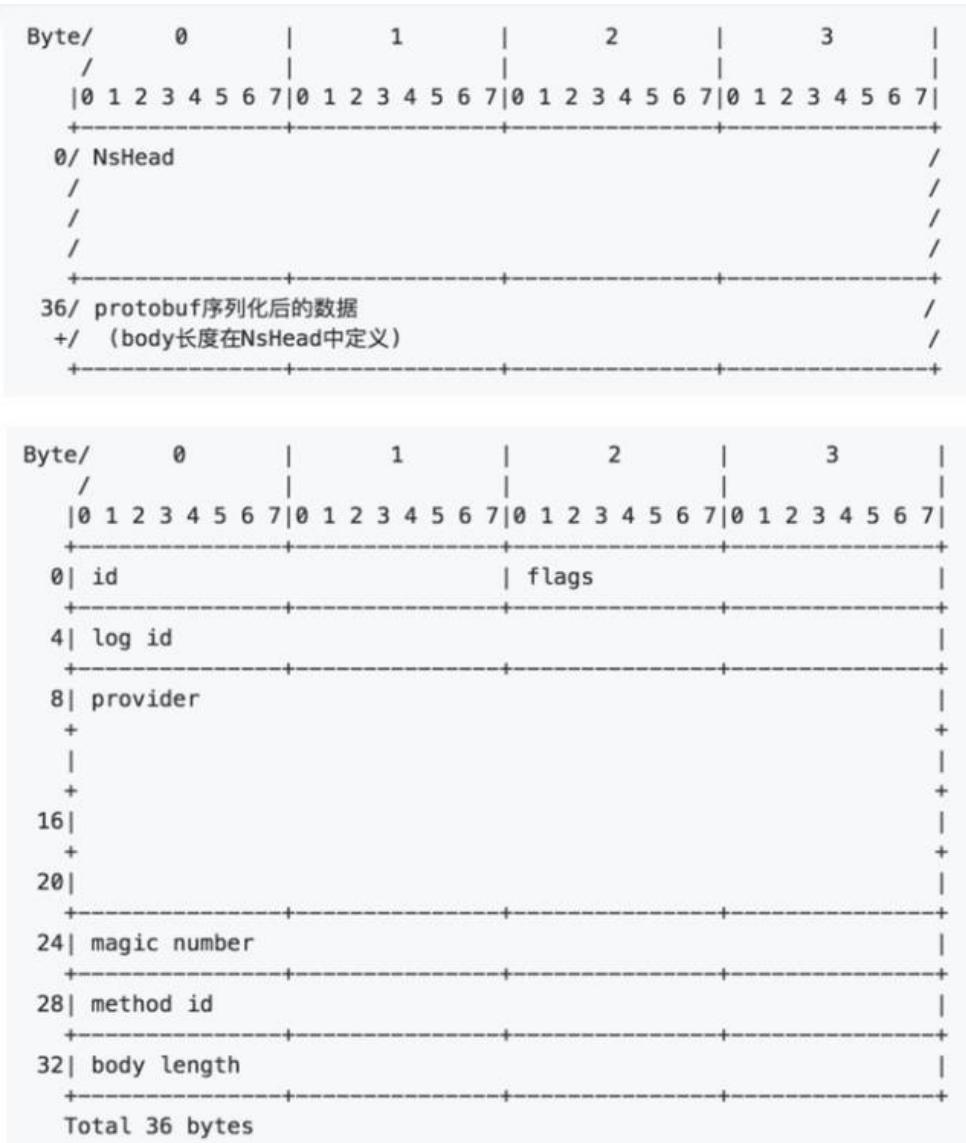


Figure 1: Frame Layout

下面展示的是作者自研的一个 RPC 框架，可以在 github 上找到这个工程。

neoremind/[navi-pbrpc](#):

可以看出它的协议栈 header+payload 方式的，header 固定 36 个



字节长度，最后 4 个字节是 body length，也就是 payload length，可以使用大尾端或者小尾端编码。

## 可靠性 (Reliability)

RPC 框架不光要处理 Network I/O、序列化、协议栈。还有很多不确定性问题要处理，这里的不确定性就是由 网络的不可靠 带来的麻烦。

例如如何保持长连接心跳？网络闪断怎么办？重连、重传？连接超时？这些都非常的细碎和麻烦，所以说开发好一个稳定的 RPC 类库是一

个非常系统和细心的工程。

但是好在工业界有一群人就致力于提供平台似的解决方案，例如 Java 中的 Netty，它是一个强大的异步、事件驱动的网络 I/O 库，使用 I/O 多路复用的模型，做好了上述的麻烦处理。

它是面向对象设计模式的集大成者，使用方只需要会使用 Netty 的各种类，进行扩展、组合、插拔，就可以完成一个高性能、可靠的 RPC 框架。

著名的 gRPC Java 版本、Twitter 的 Finagle 框架、阿里巴巴的 Dubbo、新浪微博的 Motan、Spark 2.0 RPC 的网络层（可以参考 kraps-rpc：<https://github.com/neoremind/kraps-rpc>）都采用了这个类库。

## 易用性 ( Ease of use )

RPC 是需要让上层写业务逻辑来实现功能的，如何优雅地启停一个 server，注入 endpoint，客户端怎么连，重试调用，超时控制，同步异步调用，SDK 是否需要交换等等，都决定了基于 RPC 构建服务，甚至 SOA 的工程效率与生产力高低。这里不做展开，看各种 RPC 的文档就知道他们的易用性如何了。

## 工业界的 RPC 框架一览

### 国内

- Dubbo。来自阿里巴巴 <http://dubbo.I/O/>
- Motan。新浪微博自用 <https://github.com/weibocom/motan>
- Dubbox。当当基于dubbo的 <https://github.com/dangdangdotcom/dubbox>
- rpcx。基于Golang的 <https://github.com/smallnest/rpcx>
- Navi&Navi-pbrpc。作者开源的 <https://github.com/neoremind/navi https://github.com/neoremind/navi-pbrpc>

## 国外

- Thrift from facebook <https://thrift.apache.org>
- Avro from hadoop <https://avro.apache.org>
- Finagle by twitter <https://twitter.github.io/finagle>
- gRPC by Google <http://www.grpc.io> ([Google inside use Stuppy](#))
- Hessian from cuacho <http://hessian.caucho.com>
- Coral Service inside amazon (not open sourced)

上述列出来的都是现在互联网企业常用的解决方案，暂时不考虑传统的 SOAP, XML-RPC 等。这些是有网络资料的，实际上很多公司内部都会针对自己的业务场景，以及和公司内的平台相融合（比如监控平台等），自研一套框架，但是殊途同归，都逃不掉刚刚上面所列举的 RPC 的要考慮的各个部分。

## Demo 展示

为了使读者更好地理解上面所述的各个章节，下面做一个简单例子分析。使用 neoremind/navi-pbrpc: <https://github.com/neoremind/navi-pbrpc> 来做 demo，使用 Java 语言来开发。

假设要开发一个服务端和客户端，服务端提供一个请求响应接口，请求是 user\_id，响应是一个 user 的数据结构对象。

首先定义一个 IDL，使用 PB 来做 Schema 声明，IDL 描述如下，第一个 Request 是请求，第二个 Person 是响应的对象结构。

```
message Request {
    required int32 user_id = 1;
}

message Person {
    required string user_name = 1;
    optional int64 favourite_numbers = 2;
    repeated string interests = 3;
}
```

然后使用 codegen 生成对应的代码，例如生成了 PersonPB.Request 和 PersonPB.Person 两个 class。

server 端需要开发请求响应接口，API 是 PersonPB.Person doSmth(PersonPB.Request req)，实现如下，包含一个 Interface 和一个实现 class。

```
public interface PersonService {
    PersonPB.Person doSmth(PersonPB.Request req);
}

public class PersonServiceImpl implements PersonService {
    @Override
    public PersonPB.Person doSmth(PersonPB.Request req) {
        return PersonPB.Person.newBuilder()
            .setUserName("Martin")
            .setFavouriteNumbers(1337l)
            .addAllInterests(Lists.newArrayList("daydreaming", "hacking"))
            .build();
    }
}
```

server 返回的是一个 Person 对象，里面的内容主要就是上面讲到的 PB 例子里面的。

启动 server。在 8098 端口开启服务，客户端需要靠 id=100 这个标识来路由到这个服务。

```
public static void main(String[] args) {
    PbrpcServer server = new PbrpcServer(8098);
    server.register(100, new PersonServiceImpl());
    server.start();
}
```

至此，服务端开发完毕，可以看出使用一个完善的 RPC 框架，只需要定义好 Schema 和业务逻辑就可以发布一个 RPC，而 I/O model、线程模型、序列化 / 反序列化、协议结构均由框架服务。

navi-pbrpc 底层使用 Netty，在 Linux 下会使用 epoll 做 I/O 多路复用，线程模型默认采用 Reactor 模式，序列化和反序列化使用 PB，协议结构见上文部分介绍的，是一个标准的 header+payload 结构。

下面开发一个 client，调用刚刚开发的 RPC。

client 端代码实现如下。首先构造 PbrpcClient，然后构造 PersonPB.Request，也就是请求，设置好 user\_id，构造 PbrpcMsg 作为 TCP 层传输的数据 payload，这就是协议结构中的 body 部分。

通过 asyncTransport 进行通信，返回一个 Future 句柄，通过 Future.get 阻塞获取结果并且打印。

```
PbrpcClient client =
PbrpcClientFactory.buildShortLiveConnection("127.0.0.1", 8098,
60000);

PersonPB.Request req =
PersonPB.Request.newBuilder().setUserId(127).build();
byte[] data = req.toByteArray();

PbrpcMsg msg = new PbrpcMsg();
msg.setServiceId(100);
msg.setProvider("ap");
msg.setData(data);

CallFuture<PersonPB.Person> future =
client.asyncTransport(PersonPB.Person.class, msg);

PersonPB.Person res = future.get();
System.out.println(res);
```

至此，可以看出作为一个 RPC client 易用性是很简单的，同时可靠性，例如重试等会由 navi-pbrpc 框架负责完成，用户只需要聚焦到真正的业务逻辑即可。

下面继续深入到 binary stream 级别观察，使用嗅探工具来看看 TCP 包。一般使用 wireshark 或者 tcpdump。

客户端的一次请求调用如下图所示，第一个包就是 TCP 三次握手的 SYN 包。

根据 TCP 头协议，可看出来。

- ff 15 = 65301 是客户端的端口
- 1f a2 = 8098 是服务端的端口
- header 的长度 44 字节是 20 字节头 +20 字节 option+padding

No.	Time	A. Source	Destination	Protocol	Length	Info
1	0.000000	10.28.16.68	18.128.45.124	TCP	28	65381 -> 8098 [SYN] Seq=1 Win=65535 Len=468 Win=8 Tsvl=0x84357251 Tsec=r0 SACK_PERM=1
2	0.054601	10.28.16.68	18.128.45.124	TCP	78	65381 -> 8098 [SYN] Seq=1 Win=65535 Len=468 Win=8 Tsvl=0x84357251 Tsec=r0 SACK_PERM=1
3	0.105572	10.28.16.68	18.128.45.124	TCP	66	65381 -> 8098 [ACK] Seq=1 Ack=1 Win=65535 Len=8 Tsvl=0x84357457 Tsec=r0 SACK_PERM=1
4	0.219888	10.28.16.68	18.128.45.124	TCP	384	65381 -> 8098 [PSH, ACK] Seq=3 Ack=1 Win=133528 Len=18 Tsvl=0x84357408 Tsec=r0 SACK_PERM=1
5	0.443205	10.28.16.68	18.128.45.124	TCP	66	8098 -> 65381 [ACK] Seq=1 Ack=39 Win=29856 Len=8 Tsvl=0x84357408 Tsec=r0 SACK_PERM=1
6	0.463421	10.28.16.68	18.128.45.124	TCP	135	8098 -> 65381 [PSH, ACK] Seq=1 Ack=39 Win=29856 Len=69 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1
7	0.463496	10.28.16.68	18.128.45.124	TCP	66	65381 -> 8098 [ACK] Seq=39 Ack=79 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1
8	0.487349	10.28.16.68	18.128.45.124	TCP	66	65381 -> 8098 [FIN, ACK] Seq=39 Ack=79 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1
9	0.502752	10.28.16.68	18.128.45.124	TCP	66	8098 -> 65381 [ACK] Seq=39 Ack=79 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1
-	0.602878	10.28.16.68	18.128.45.124	TCP	66	65381 -> 8098 [ACK] Seq=48 Ack=71 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1

0000	00 10 db ff 10 02 c4 b3	01 cf 15 39 08 00 45 00
0010	00 40 a5 e9 40 00 40 06	41 7b 0a 14 10 44 0a 80
0020	2e 7c ff 15 1f a2 07 bc	78 2f 00 00 00 00 b0 02
0030	ff ff fa 54 00 00 02 04	05 b4 01 03 03 03 01 01
0040	08 0a 0c 2e 3e 85 00 00	00 00 04 02 00 00



- TCP Three-way handshake SYN segment
  - Client port is 65301 (0xff15), server port is 8098 (0x1fa2)
  - 44 bytes TCP header = 20 bytes + 24 options & padding
- 构成的。

三次握手成功后，下面客户端发起了 RPC 请求，如下图所示。

No.	Time	A. Source	Destination	Protocol	Length	Info
1	0.000000	10.28.16.68	18.128.45.124	TCP	78	65381 -> 8098 [SYN] Seq=1 Win=65535 Len=468 Win=8 Tsvl=0x84357251 Tsec=r0 SACK_PERM=1
2	0.054601	10.28.16.68	18.128.45.124	TCP	78	8098 -> 65381 [SYN] Seq=1 Win=29856 Len=8 HShd=531953719 Tsec=r0 SACK_PERM=1
3	0.105572	10.28.16.68	18.128.45.124	TCP	66	65381 -> 8098 [ACK] Seq=1 Ack=39 Win=133528 Len=8 Tsvl=0x84357457 Tsec=r0 SACK_PERM=1
4	0.219888	10.28.16.68	18.128.45.124	TCP	384	65381 -> 8098 [PSH, ACK] Seq=3 Ack=39 Win=133528 Len=18 Tsvl=0x84357408 Tsec=r0 SACK_PERM=1
5	0.443205	10.28.16.68	18.128.45.124	TCP	66	8098 -> 65381 [ACK] Seq=39 Ack=79 Win=131448 Len=8 Tsvl=0x84357408 Tsec=r0 SACK_PERM=1
6	0.463421	10.28.16.68	18.128.45.124	TCP	135	8098 -> 65381 [PSH, ACK] Seq=39 Ack=79 Win=29856 Len=69 Tsvl=0x84357408 Tsec=r0 SACK_PERM=1
7	0.463496	10.28.16.68	18.128.45.124	TCP	66	65381 -> 8098 [ACK] Seq=39 Ack=79 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1
8	0.487349	10.28.16.68	18.128.45.124	TCP	66	8098 -> 65381 [FIN, ACK] Seq=39 Ack=79 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1
9	0.502752	10.28.16.68	18.128.45.124	TCP	66	65381 -> 8098 [ACK] Seq=48 Ack=71 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1
-	0.602878	10.28.16.68	18.128.45.124	TCP	66	8098 -> 65381 [ACK] Seq=48 Ack=71 Win=131448 Len=8 Tsvl=0x84357394 Tsec=r0 SACK_PERM=1

0000	00 10 db ff 10 02 c4 b3	01 cf 15 39 08 00 45 00	..... .9..E.
0010	00 5a c7 77 40 00 40 06	1f d3 0a 14 10 44 0a 80	.Z.w@. @. ....D..
0020	2e 7c ff 15 1f a2 07 bc	78 30 8b 40 9d 16 80 18	. . .... x0. @. ....
0030	40 38 4d 4f 00 00 01 01	08 0a 0c 2e 3f 72 1f b4	@M0. .... .?r..
0040	f8 37 00 00 00 00 f9 3a	9c 72 61 70 00 00 00 00	.7....: .rap....
0050	00 00 00 00 00 00 00 00	00 00 94 93 70 fb 64 00	.....: p.d.
0060	00 00 02 00 00 00 00 08 7f		.....

可以看出 TCP 包含了一个 message，由 navi-pbrpc 的协议栈规定的 header+payload 构成，

继续深入分析 message 中的内容，如下图所示：

0000	00 10 db ff 10 02 c4 b3	01 cf 15 39 08 00 45 00	Data = header + payload
0010	00 5a c7 77 40 00 40 06	1f d3 0a 14 10 44 0a 80	Byte/ 0 1 2 3 4 5 6 7   0 1 2 3 4 5 6 7   0 1 2 3 4 5 6 7   0 1 2 3 4 5 6 7
0020	2e 7c ff 15 1f a2 07 bc	78 30 8b 40 9d 16 80 18	R/  WinSize   /   Flags   /
0030	40 38 4d 4f 00 00 01 01	08 0a 0c 2e 3f 72 1f b4	/   provider   /
0040	f8 37 00 00 00 00 f9 3a	9c 72 61 70 00 00 00 00	/   tagId   /
0050	00 00 00 00 00 00 00 00	00 00 94 93 70 fb 64 00	/   body.length   /
0060	00 00 02 00 00 00 00 08 7f		/   body.data   /

#### Header protocol

0000	00 10 db ff 10 02 c4 b3	01 cf 15 39 08 00 45 00	Header protocol
0010	00 5a c7 77 40 00 40 06	1f d3 0a 14 10 44 0a 80	Byte/ 0 1 2 3 4 5 6 7   0 1 2 3 4 5 6 7   0 1 2 3 4 5 6 7   0 1 2 3 4 5 6 7
0020	2e 7c ff 15 1f a2 07 bc	78 30 8b 40 9d 16 80 18	R/  provider   /
0030	40 38 4d 4f 00 00 01 01	08 0a 0c 2e 3f 72 1f b4	/   tagId   /
0040	f8 37 00 00 00 00 f9 3a	9c 72 61 70 00 00 00 00	/   magic number   /
0050	00 00 00 00 00 00 00 00	00 00 94 93 70 fb 64 00	/   method.id   /
0060	00 00 02 00 00 00 00 08 7f		/   body.length   /

- Data = header + payload (body)
- Note I set “ap” = 0x6170 in hex as provider
- Body length is 2 as little endian long
- Body data is 0x087f. Can you decode the protobuf message? ☺
- User Id is 127 (0x7f)

```
PersonPB.Request req = PersonPB.Request.newBuilder().setUserId(127).build();
```

```
byte[] data = req.toByteArray();
```

```
PbrpcMsg msg = new PbrpcMsg();
```

```
msg.setServiceId(100);
```

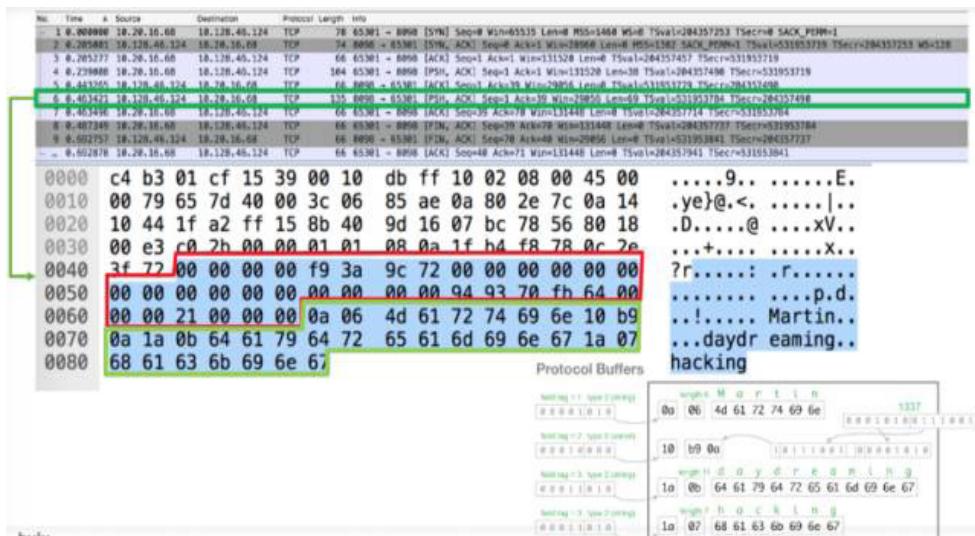
```
msg.setProvider("ap");
```

```
msg.setData(data);
```

其中

- $61\ 70 = \text{ap}$  是头中的 provider 标识
- body length 是 2，注意 navi-pbrpc 采用了小尾端。
- payload 是 08 7f，08 在 PB 中理解为第一个属性，是 varint 整型，7f 表示传输的是 127 这个整型。

服务端响应 RPC 请求，还是由 navi-pbrpc 的协议栈规定的 header+payload 构成，可以看出 body 就是 PB 例子里面的二进制数据。



最后，客户端退出，四次分手结束。

## 总结

本文系统性地介绍了 RPC 包含的核心概念和技术，带着读者从一个实际的例子去映射理解。很多东西都是蜻蜓点水，每一个关键字都能成为一个很大的话题，希望这个提纲挈领的介绍可以让读者在大脑里面有一个系统的体系去看待 RPC。

欢迎访问作者的[博客](#)。

## 作者介绍

**张旭**，目前工作在 Hulu，从事 Big data 领域的研发工作，曾经在

百度 ECOM 和程序化广告部从事系统架构工作，热爱开源，在 github 贡献多个开源软件，id:neoremind，关注大数据、Web 后端技术、广告系统技术以及致力于编写高质量的代码。



华为云

上 | 云 | 狂 | 欢 | 节

# 极速云服务器

低至

1.04

元  
/ 天



数据库·存储·云硬盘

限时半价 安全产品限时免费



## EGO会员招募季第三季正式开启

EGO旨在组建全球最具影响力的技术领导者社交网络  
联结杰出的技术领导者学习和成长



申请加入  
扫码联系E小欧



## 架构师 月刊 2017年10月

本期主要内容：Java 9 正式发布，新特性解读；百度正式开源其 RPC 框架 brpc；Kafka 数据可靠性深度解读；MySQL 到底能不能放到 Docker 里跑？虚拟座谈会：聊聊 AIOps 的终极价值；软件测试技术的未来；gRPC 客户端创建和调用原理解析。



## 架构师特刊 大前端

本期主要内容：当我们在谈大前端的时候，我们谈的是什么；如何落地和管理一个“大前端”团队？



## 顶尖技术团队访谈录 第九季

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。