

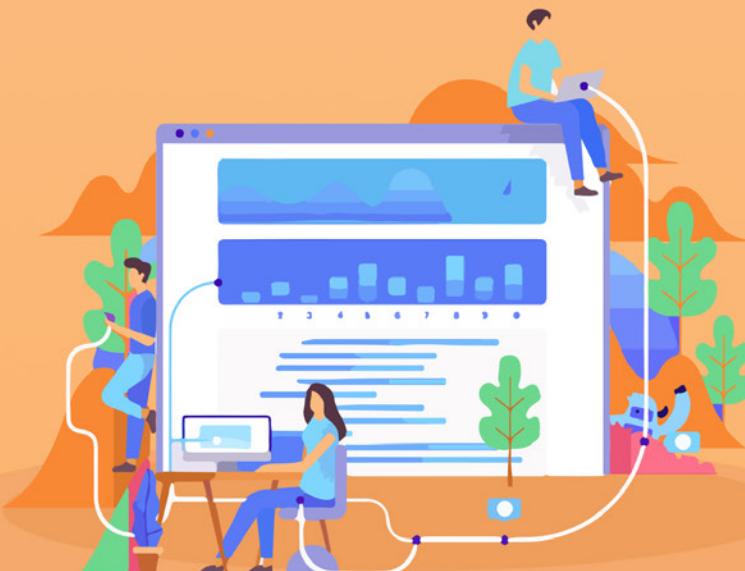
架构师

ARCHITECT

9月刊

热点

Kafka 2.0重磅发布，
新特性独家解读



CONTENTS / 目录

热点 | Hot

集 Python、C++、R 为一体！Julia 1.0 重磅发布

Kafka 2.0 重磅发布，新特性独家解读

推荐文章 | Article

都去炒 AI 和大数据了，落地的事儿谁来做？

如何才能写出好的软件设计文档？

理论派 | Theory

我用 Vue 和 React 构建了相同的应用程序，这是他们的差异

观点 | Opinion

REST 将会过时，而 GraphQL 则会长存

专题 | Topic

人工智能与区块链初探：交集与前瞻

特别专栏 | Column

Istio 在 UAEK 中的实践改造之路



架构师 2018 年 9 月刊

本期主编 陈思

提供反馈 feedback@cn.infoq.com

流程编辑 丁晓昀

商务合作 hezuo@geekbang.org

发行人 霍泰稳

内容合作 editors@cn.infoq.com

卷首语

人工智能其实就是『八卦』

作者 王文广 达观数据副总裁

《诗》，即《诗经》，和《易》，即《易经》可以说是中国文化的两个源头，再加上作为科学的源头的《几何原本》，就像三原色一样衍生出多彩的中国社会。在这个社会里，《易》的影响是非常巨大的。从春秋时期的诸子百家，到唐宋时期的诗词歌赋，以至于现代的仙侠武侠小说，都可以从《易》中找到各种痕迹。更为广泛的是深入大江南北的风水、算命、求神拜佛、婚配嫁娶、升官发财、建筑结构、家具摆放等等南北风俗，无不可以从《易》中找到些许痕迹。就连中国唯二的TOP1学府也从中找了个句子作为自己的校训。

《易》不仅影响中国，也影响了世界。《易》不仅仅在东亚文化圈里影响巨大，也是当代物理学的基础。众所周知，没有量子力学，就没有信息时代的一切，而量子力学的奠基人之一波尔就是受《易》的阴阳相对立统一原理的启发，提出了量子力学的基础原理之一『互补原理』。量子力学的另一个奠基性理论『波粒二象性』也隐隐约约有『阴阳』理论的影子。至于信息时代的基础——电脑，以及当今热火朝天的人工智能，更是《易》这个系统的直接传承。

话说当年莱布尼茨就是看了《易》之后，受到启发，发明了二进制体系，

而二进制体系则是我们这年代所离不开的计算机、手机等基础。也就是说，本质上，计算机和手机就是高阶的八卦。还原一下这里面的逻辑，两仪四象和八卦分别是 1 位、2 位和 3 位二进制系统，而《易》则是 4 位二进制系统 -- 它一共有六十四个卦象。早期计算机是 8 位二进制系统 -- 大家所熟悉的 ASCII 是 7 位（后来扩展为 8 位）的二进制系统。最为大家所熟悉的 WindowsXP 是运行在 32 位二进制 CPU 上的系统，而现在大家所用计算机和手机则大都是 64 位二进制系统了。到此你可以看出其中是如何一脉相承的了吧？

《易》不仅仅是一个系统，它还是一个生态。从中衍生出的应用纷繁复杂，丝毫不逊于前阵子被吹捧的美国第一个超万亿美元市值的苹果公司（世界第一个超万亿美元市值的是中石油，又一次，美国落后于中国）的 iOS 生态。比如八字，算卦，梅花易数，风水，大六壬、奇门遁甲、紫微斗数等等不同门派，它的特点是通过朴素的『统计学』原理，结合『自然语言理解』和『自然语言生成』技术，来实现对未来的预测。后来，这些门派传到西欧和北美，经由同样从《易》衍生出来的计算机一起发展，终究成为了现在的大热门 -- 人工智能。当年由《易》衍生出来的不同门派，现在则成了不同算法派系，比如决策树，随机森林，支持向量机，逻辑回归，神经网络，深度学习等等。这种延绵数千年的传承关系，充分说明了我们祖先的智慧。简单举个例子，比如梅花易数，显然是一个表示学习，它将日常见到的各种场景学习成向量（卦名），然后将特征表示通过已经学习出来的八卦六爻五行模型来预测，这便是人称『神算子』的 AI 模型。

作为读者的你看到这里，肯定觉得这是在胡扯，是歪理邪说，赤裸裸的碰瓷。而我要告诉你的是，这是没有理论自信道路自信制度自信文化自信。不然，看看人家诺贝尔获奖者的『四个自信』。2011 年诺贝尔经济学奖获得者 Thomas J. Sargent 在前几天（大约 8 月 11 日）就说『人工智能首先是一些很华丽的辞藻。人工智能其实只是统计学，只不过用了一个很华丽的辞藻，其实就是统计学。』他不仅说了，还有好多拥趸。要真这么简单的话，那些统计学家怎么在几十年的时间都 没有搞出一堆公

式来搞定围棋的解？至于图像处理、语音识别、以及达观数据所擅长的文本智能处理、机器翻译、推荐系统等等，没见有统计学家搞出个公式来求解，也没见有统计学家来证明某种方法能够搞定这些事情？即使到现在，深度学习这种暗黑系的方法效果已经很好了，也没有统计学家出来证明一下，深度学习为什么有用？更不用说让统计学家来证明或者指导一下，怎样才能更有用。

其实统计学本身的发展，也是在一系列的应用后总结归纳抽象出来的。这个跟几何学或者算数学这种古老的数学发展起来的很不一样。而从另一方面来讲，当下的人工智能是一个综合性的多学科交叉的学科（或者技术）。这里面确实有一部分统计学的理论，但其中更多的还包括诸多其他学科，比如包括芯片、分布式体系和架构、高性能计算、程序设计在内的计算机学科。如果非要强行说 AI 是数学，那也不一定轮的上『统计学』，像数值分析，组合数学，计算理论，离散数学，图论，微积分，微分几何等等学科对人工智能的作用都不见得比统计学少。

到这里，大概你也明白了。我们要做的是，鼓起『四个自信』大声说出：『（计算机科学和）人工智能其实就是易经里的八卦。只不过用了一个很华丽的辞藻，人工智能其实就是八卦。』最后，欢迎大家加入达观数据来一起『八卦』人工智能。

关于作者

王文广达观数据副总裁，在人工智能领域和系统架构设计有十余年工作经验，浙江大学计算机硕士。早期在百度负责 MP3 搜索、语音搜索系统和音频指纹系统核心研发。

曾担任金融 AI 公司 Kavout 首席架构师，将人工智能和文本挖掘技术应用于金融、证券、量化交易等领域，效果得到美国大型基金公司认可。曾负责盛大创新院搜索、推荐、广告等多个项目的架构设计工作，所参与开发的系统具备海量数据的快速处理和高精度的挖掘能力，多次获得嘉奖。

AiCon

全球人工智能与机器学习技术大会

大会聚焦

- 机器学习应用和实践
- 计算机视觉
- NLP和语音技术
- 搜索推荐与算法
- 数据智能驱动业务
- AI+行业案例



6折报名中，立减1440元

联系热线：18514549229

2018年12月20-23 / 北京·国际会议中心



▶ 大咖助阵



颜水成
360人工智能研究院
院长及首席科学家



华先胜
阿里巴巴达摩院机器智能技术实
验室副主任 城市大脑人工智能
技术负责人



裴健
京东集团副总裁
大数据与智能供应链事业部总裁



马维英
今日头条副总裁
人工智能实验室负责人



崔宝秋
小米人工智能与云平台副总裁
首席架构师

▶ 分享嘉宾



薄列峰
京东金融
AI实验室首席科学家



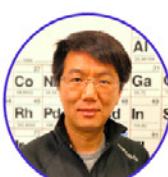
袁进辉 (老师木)
一流科技创始人



张俊林
新浪微博
AI Lab资深算法专家



王兴星
美团点评技术总监
外卖商业技术负责人



陈博兴
阿里巴巴达摩院
资深算法专家



鲍捷
文因互联CEO
人工智能领域知名专家

6折报名中，立减1440元

联系热线：18514549229

2018年12月20-23 / 北京·国际会议中心



集 Python、C++、R 为一体！Julia 1.0 重磅发布

编辑 陈利鑫



近年来，Julia 语言已然成为编程界的新宠，今年 TOIBE8 月份编程语言排行榜上，Julia 已迅速攀升至第 50 名。短短几年，这门由 MITCSAIL 实验室开发的编程语言就变得炙手可热，很大部分是因为这门语言结合了 C 语言的速度、Ruby 的灵活、Python 的通用性，以及其他各种语言的优势于一身，并且具有开源、简单易掌握的特点。

8 日，Julia 正式发布 1.0 版本。Julia 团队表示：“Julia1.0 版本是我们为如饥似渴的程序员构建一种全新语言数十年来工作成果的巅峰。”那么问题来了，Julia 真有这么神？你做好学习一门新编程语言的准备了吗？

为什么你应该学习 Julia ?

从 2012 年到现在，Julia1.0 在编程界已经打出了自己的一片“小天地”。截至发稿前，Julia 在 Github 上已经获得了 12293 颗星星，TOIBE8 月份编程语言排行榜上已迅速攀升至第 50 名。

40	Haskell	0.242%
41	OpenCL	0.226%
42	Scheme	0.206%
43	Kotlin	0.205%
44	Groovy	0.190%
45	Hack	0.188%
46	Bash	0.169%
47	Tcl	0.158%
48	Erlang	0.158%
49	REXX	0.156%
50	Julia	0.156%

Julia 之所以这么受欢迎，这与它解决了工程师们一个“坑爹”问题有关：工程师们为了在数据分析中获得速度和易用性，不得不首先用一种语言编码，然后用另一种语言重写，即很多人口中的“双语言问题”。

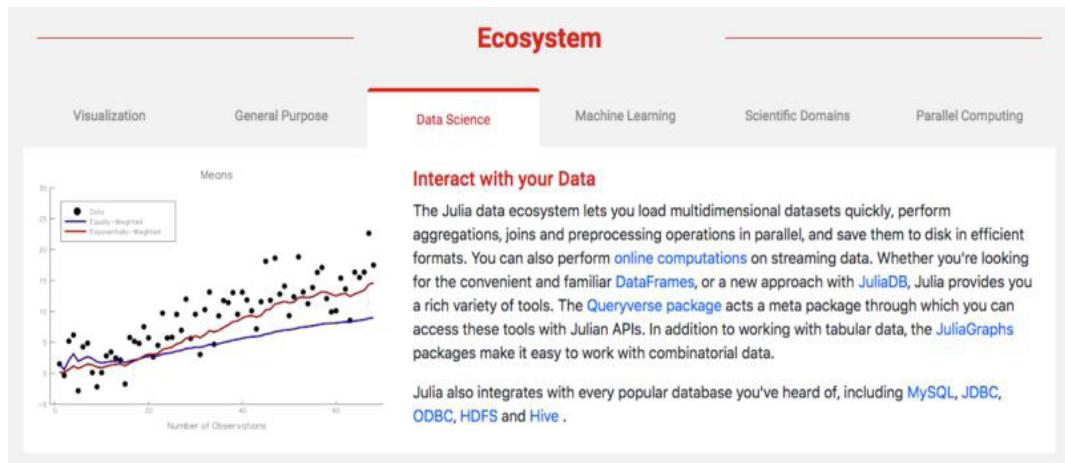
与其他语言相比，Julia 易于使用，大幅减少了需要写的代码行数；并且能够很容易地部署于云容器，有更多的工具包和库，并且结合了多种语言的优势。据 JuliaComputing 的宣传，在七项基础算法的测试中，Julia 比 Python 快 20 倍，比 R 快 100 倍，比 Matlab 快 93 倍。

目前 Julia 的应用范围已经非常广泛了，可以用于天文图像分析、自动驾驶汽车、机器人和 3D 打印机、精准医疗、增强现实、基因组学和风险管理等领域。

两年前，诺贝尔经济学奖得主 Thomas Sargent 和澳大利亚国立大学的经济学教授 John Stachurski，共同建议纽约联邦储备银行把其用于市场走势预测和政策分析的动态随机一般均衡模型 (DSGE) 转到 Julia 语言平台。在项目第一阶段后，他们发现，Julia 把模型运行时间缩短至原先 Matlab

代码的十分之一到四分之三。

除了语言本身的优点，Julia 还拥有非常强大的生态系统，主要应用于数据可视化、通用计算、数据科学、机器学习、科学领域、并行计算六大领域。



Julia 在规模化机器学习领域为深度学习、机器学习和 AI 提供了强大的工具（Flux 和 Knet）。Julia 的数学语法使其成为表达算法的理想方式，支持构建具有自动差异的可训练模型，支持 GPU 加速和处理数 TB 的数据。Julia 丰富的机器学习生态系统还提供监督学习算法（如回归、决策树）、无监督学习算法（如聚类）、贝叶斯网络和马尔可夫链蒙特卡罗包等。

Julia 目前下载量已经达到了 200 万次，Julia 社区开发了超过 1900 多个扩展包。这些扩展包包含各种各样的数学库、数学运算工具和用于通用计算的库。除此之外，Julia 语言还可以轻松使用 Python、R、C/C++ 和 Java 中的库，这极大地扩展了 Julia 语言的使用范围。

所以说，Julia 火起来不是没有原因的，而最新发布的 1.0 版本又添加了很多新功能。

按例，先贴上新版本相关链接：

Julia1.0 试用版链接：<https://julialang.org/downloads/>

GitHub 地址：<https://github.com/JuliaLang/julia>

目前支持 Julia 的平台：

Operating System	Architecture	CI	Binaries	Support Level
Linux 2.6.18+	x86-64 (64-bit)	✓	✓	Official
	i686 (32-bit)	✓	✓	Official
	ARM v7 (32-bit)		✓	Official
	ARM v8 (64-bit)		✓	Official
	PowerPC (64-bit)			Community
	PTX (64-bit)	✓		External
macOS 10.8+	x86-64 (64-bit)	✓	✓	Official
Windows 7+	x86-64 (64-bit)	✓	✓	Official
	i686 (32-bit)	✓	✓	Official
FreeBSD 11.0+	x86-64 (64-bit)	✓		Community

Julia 到底是怎样一门语言？

Julia 首次公开面世时便体现出该社区对语言的一些强烈要求：

我们想要一种拥有自由许可的开源语言。我们想要它拥有 C 的速度与 Ruby 的灵活。它要容易理解，像 Lisp 一样真正地支持宏，但也要有像 Matlab 一样的明显、熟悉的数学符号。它还要像 Python 一样可用于通用编程，像 R 一样易于统计，像 Perl 一样可自然地用于字符串处理，像 Matlab 一样擅长线性代数，像 shell 一样擅长将程序粘合在一起。总之，它既要简单易学，但也要让最严肃的黑客开心。我们既希望它是交互式，也希望它是可编译的。

现在，一个充满活力和蓬勃发展的社区围绕着这种语言成长起来，来自世界各地的人们在追求这一目标的过程中不断地精炼并重塑着 Julia。超过 700 人为 Julia 做出了贡献，还有很多人制作了数以千计的令人惊叹的开源 Julia 软件包。总而言之，我们建立的语言：

快速：Julia 就是为高性能而设计的。Julia 程序通过 LLVM 编译为多

个平台的高效本机代码。

通用：它使用多个调度作为范例，使得它很容易表达众多面向对象和函数编程的模式。它的标准库提供异步 I/O、进程控制、日志记录、概要分析、软件包管理器等。

- 动态：Julia是动态类型的，就像一种脚本语言，并且很好地支持交互式使用。
- 技术：它擅长于数值计算，其语法非常适合数学，支持的数字数据类型众多，并具有开箱即用并行性。Julia的多次调度非常适合定义数字和数组类型的数据类型。
- （可选）键入：Julia具有丰富的描述性数据类型语言，类型声明可用于阐明和巩固程序。
- 可组合：Julia的软件包可以很好地协同工作。单位数量矩阵，货币和颜色数据表都可以进行，并且性能良好。

如果你要从 Julia0.6 或更早版本升级代码，我们建议首先使用过渡版 0.7，其中包括弃用警告帮助指导完成升级。如果你的代码没有警告，则可以更改为 1.0 而无需任何功能更改。已注册的软件包正在使用该过渡版本发布 1.0 兼容的更新。

1.0 更新了哪些功能？

当然，Julia1.0 中最重要的一个新功能是对语言 API 稳定性的承诺：你为Julia1.0 编写的代码可以继续在Julia1.1、1.2 等版本中运行。该语言是“已完善”的，核心语言开发人员和社区都可以放心使用基于此版本的软件包、工具和新功能。

但 Julia1.0 更新的不仅是稳定性，它还引入了一些强大、创新的语言功能。自 0.6 版以来，新发布的一些功能包括：

- 全新的内置软件包管理器性能得以大幅改进，使安装包及其 dependencies 项变得前所未有的简单。它还支持每个项目的包环境，并记录工作应用程序的确切状态，以便与他人和你自己进行

共享。最后，新的设计还引入了对私有包和包存储库的无缝支持。你可以使用与开源软件包生态系统相同的工具来安装和管理私有软件包。JuliaCon上展示了新功能设计的[详细情况](#)。

- Julia有了一个新的规范表示[缺失值](#)。能够表示和处理缺失的数据是统计和数据科学的基础。与Julian的一贯风格相符，这个新的解决方案具有通用性、可组合性和高性能。任何泛型集合类型都可以通过让元素包含missing的预定义值来有效地支持缺失值。在以前的Julia版本中，这种“联合类型”集合的性能会太慢，但编译器的改进现在使得Julia可以跟上其他系统中自定义C或C++缺失数据表示的速度，同时也更加通用和灵活。
- 内置的String类型现在可以安全地保存任意数据。你的程序数小时甚至数天的工作不再会因为一些无效Unicode杂乱字节而失败。保留所有字符串数据，同时标记哪些字符有效或无效，可以使你的应用程序安全方便地处理不可避免具有缺陷的真实数据。
- 语法简单的广播（Broadcasting）已经成为核心语言功能，现在它比以往任何时候功能都更强大。在Julia1.0中，将广播扩展到自定义类型并在GPU和其他矢量化硬件上实现高效优化计算变得更简单，为将来提高性能提升铺平了道路。
- 命名元组是一种新的语言特性，它使得通过名称表示和访问数据变得高效快捷。例如，你可以将一行数据表示为`row = (name="Julia", version=v"1.0.0", releases=8)`，并将版本列作为`row.version`访问，其性能与不甚快捷的`row[2]`相同。
- 点运算符现在可以重载，让类型使用`obj.property`语法来获取和设置结构字段之外的含义。这对于使用Python和Java等基于类的语言更顺畅地进行互操作是个福音。属性访问器重载还允许获取一列数据以匹配命名元组语法的语法：你可以编写`table.version`来访问表的`version`列，就像`row.version`访问单行的`version`字段一样。
- Julia的优化器在很多方面变得比我们在这里提到的更聪明，但有

一些亮点值得一提。优化器现在可以通过函数调用传播常量，可以更好地做到死码消除和静态评估。另外，编译器在避免在长生命周期对象周围分配短期包装器方面也要好得多，这使得程序员可以使用便利的高级抽象而无需降低性能成本。

- 现在使用声明相同的语法调用参数类型构造函数。这消除了语言语法的模糊和令人困惑的地方。
- 迭代协议已经完全重新设计，以便更容易实现多种迭代。现在是一对一定义一个或两个参数方法，而不是定义三个不同泛型函数的方法——start, next, 和done。这通常使得使用具有开始状态的默认值的单个定义可以更方便地定义迭代。更重要的是，一旦发现无法生成值就可以部署迭代器。这些迭代器在I/O、网络和生产者/消费者模式中无处不在；Julia现在可以用简单直接的方式表达这些迭代器。
- 范围规则简化。无论名称的全局绑定是否已存在，引入本地范围的构造现在都是一致的。这消除了先前存在的“软/硬范围”区别，并且意味着现在Julia可以始终静态地确定变量是本地的还是全局的。
- 语言本身非常精简，许多组件被拆分为“标准库”软件包，这些软件包随Julia一起提供但不属于“基础”语言。如果你需要它们，它可以给你方便（不需要安装），但不会被强加给你。在未来，这也将允许标准库独立于Julia本身进行版本控制和升级，从而允许它们以更快的速度发展和改进。
- 我们对Julia的所有API进行了彻底的审查，以提高一致性和可用性。许多模糊的遗留名称和低效的编程模式已被重命名或重构，以更优雅地匹配Julia的功能。这促使使用集合更加一致和连贯，以确保参数排序遵循整个语言的一致标准，并在适当的时候将（现在更快）关键字参数合并到API中。
- 围绕Julia1.0新功能的新外部包正在构建中。例如：

- 正在改进数据处理和操纵生态系统，以利用新的缺失支持
- Cassette.jl提供了一种**强大的机制**，可以将代码转换传递注入Julia的编译器，从而实现事后分析和现有代码的扩展。除了用于分析和调试等程序员的工具之外，甚至可以实现机器学习任务的自动区分。
- 异构体系结构支持得到了极大的改进，并且与Julia编译器的内部结构进一步分离。英特尔KNL只能用Julia工作。NvidiaGPU使用CUDANative.jl软件包进行编程，GoogleTPU的端口正在开发中。

另外，Julia1.0 还有无数其他大大小小的改进。有关更改的完整列表，请参阅文件。

在 2012 年的文章《为什么我们创造Julia》这篇博客文章中，我们写道：它不完整，但现在是 1.0 发布的时候——我们创建的语言叫做 Julia。

现在，我们提前叩响了 1.0 版本发布的扳机，但它发布的时刻已然到来。真诚地为这些年来为这门现代化编程语言做出贡献的人们感到骄傲。

Kafka 2.0 重磅发布，新特性独家解读

作者 王国璋



增强在线可进化性

就我个人而言，这几年来学到的最重要的一课，就是要永远保证一个流式数据平台的在线可进化性（online-evolvable）。

之前我曾经读到 Amazon CTO Werner Vogels 写过的一篇博客，里面就提到这一点，并且有一个精彩的比喻：搭建一个能够在不断产品升级过程中保证永远在线的数据架构，就像是驾驶着一架简单的单螺旋桨飞机起飞，然后在飞行过程中，不断换新零件和添加新引擎，直到最后升级成一架超大的空客飞机，这一切都必须在飞行中同步完成，并且坐在里面的乘客不能有任何感觉，其难度可想而知。

而关于一个流数据平台，对于在线可进化性这一点的需求尤甚：这里我也打一个比方，尽管没有 Vogels 的那样精彩——数据流就像是一个连接城市各个地区的高速公路，高速公路所连接的地方，比如一家超市、一家影院，或者一个居民小区，就如同一个企业里面大大小小的各种应用产品或者数据仓库，超市可以暂时歇业，影院可以暂时关门翻修，居民小区甚至也可以迁出人口夷平重建，就像是产品或者数据仓库都可以短暂下线升级维护。然而高速公路却很难彻底打断重搭，因为随时都有人要上路。更多时候，它只能一边继续完成输送车辆的任务，一边增设车道或者加盖匝道。就像是一个流数据平台本身，因为不会有零流量的时刻，所以所有的维护和升级都需要保证同步在线完成，而且期间最好没有任何用户可感知到的性能弱化或者服务差别。而在云环境下，后者显得更为重要。对于 Kafka 而言，如果一个用户没有一个安全稳定的升级路线的话，那么她就只能停留在最初的那个版本，再不会升级。

因此我们从很早以前开始注意保证在线升级的方便性，在这一次的 2.0.0 版本中，更多相关的属性被加了进来，比如 KIP-268、KIP-279、KIP-283 等等。

KIP-268：简化 Kafka Streams 升级过程

Kafka Streams 利用 Consumer Rebalance 协议里面的元数据字符串编码诸如任务分配、全局查询、版本升级相关的信息。然而，当编码版本本身改变的时候，就需要进行离线升级。比如之前从 0.10.0 版本向更高级的版本升级的时候，用户就需要将所有的 Streams 程序下线，换上新的 Kafka 版本号，然后在全部重启。

KIP-268 利用 version prob 可以使得旧版本的任务分配者告知其他高版本的成员暂时使用旧版本的 Rebalance 元数据编码，这样就可以让用户依然能够通过 rolling bounce 在线升级 Kafka Streams 的版本。而当所有参与的成员全部升级完毕之后，最后一次 rebalance 会自动切换回新版本的元数据编码。

KIP-279：修补多次 Kafka 分区主本迁移时的日志分歧问题

在升级 Kafka 版本或者做定期系统维护的时候，用户往往需要进行连续的多次 Kafka 分区迁移。在这次发布中我们修补了一个在此过程中可能会出现的一个会导致日志分歧发生的边缘情况。具体方案就是将此前版本中已经加入的主本 epoch 信息扩散到 OffsetForLeaderEpochResponse。如此所有主副本就可以清晰知道自己到底处于当前分区备份的哪一个阶段，从而杜绝因为消息不对等而可能导致的日志分歧。

KIP-283：降低信息格式向下转换时的内存消耗

在一个多客户端组群的环境下，客户端与服务器端的版本不匹配是常见现象。早在 0.10.0 版本中，Kafka 已经加入了允许不同版本客户端与服务器交互的功能，即高版本的 Kafka 客户端依然可以与低版本的服务器进行数据传导，反之亦然。然而当低版本的消费者客户端和高版本的服务器进行交互时，服务器有时需要将数据向下转换（format down-conversion）成为低版本客户端可以认知的格式后才能发回给消费者。向下转换有两个缺点：

1. 丢失了 Kafka 数据零拷贝（zero-copy）的性能优势；
2. 向下转换需要额外的大量内存，在极端情况下甚至会导致内存溢出。

前者无法避免，但是后者依然可以改进：在即将发布的 2.0 版本中，我们使用了一种新的基于分块（chunking）的向下转换算法，使得需要同时占据的内存需求大幅缩减。这使得高低版本的客户端与服务器之间的交互变得更加有效。

更多的可监控指标

对于企业级数据平台来说，另一个很重要的要求就是提供各种实时的监控能力。在 LinkedIn 的时候，同事间流传着据传是我们公司传奇人物

David Henke 的一句话：what gets measured gets fixed，充分体现了监测的重要性。

长期以来，Apache Kafka 社区不断地完善各种区块的各种指标，这每一个新添加的指标背后都有一个我们曾经踩过的坑，一段在线调试和修 bug 的痛苦经历。

举一个具体的例子：Kafka 长期以来被诟病添加分区太慢，因此在 1.1.0 版本里面来自六个不同企业的贡献者共同完成了重新设计 Kafka 控制器（Kafka Controller）这个规模巨大的 JIRA。在这个长达九个月的项目里，被谈论很多的一点就是如何增添控制器操作的各种指标。在未来更多的新功能和新属性，比如继续增强 Kafka 的伸缩性，包括多数据中心支持等等，如何能够让用户继续便捷地实时监测这些新增功能的性能，及时发现可疑问题，并且帮助缩短需要的在线调试时间，都将是讨论的重要一环，因为这也是任何一个企业级流数据平台必须要注意到的。

在 2.0.0 版本中，我们进一步加强了 Kafka 的可监控性，包括添加了很多系统静态属性以及动态健康指标，比如 KIP-223、KIP-237、KIP-272 等等。

KIP-223：加入消费者客户端的领先指标

在此前，Kafka 消费者客户端已经加入了对每一个消费分区的延迟指标（lag metrics），定义为当前消费者在分区上的位置与分区末端（log-end-offset）的距离。当此指标变大时，代表消费者逐渐跟不上发布的速度，需要扩容。我们发现，当分区 retention 时间很短而导致消费者跌出可消费范围时（out-of-range），此指标不能完全针对潜在的危险为用户报警。

因此在即将发布的 2.0 版本中，我们加入了另一个“领先”指标（lead metrics），定义为分区首端（log-start-offset）与消费者在分区上的位置距离，当此指标趋近于零时，代表消费者有跌出可消费范围因而丢失数据的危险。值得称赞的是，这个新特性是由国内的社区贡献者完成的。

KIP-237：加入更多 Kafka 控制器的健康指标

在完成了针对增强 Kafka 控制器性能的全面重设计之后，我们认为现在 Kafka 已经可以支持千台机器，百万分区级别的集群。在此之前，这样规模集群的一个阻碍就是 Kafka 控制器本身处理各种 admin 请求，诸如主本迁移、话题扩容、增加副本等等时的时间消耗。在完成了这个重设计之后，我们也相应地加入了更多和新设计相关的健康指标，包括控制器内部请求队列的长度，请求处理速率等等。

更全面的数据安全支持

最后，也是最近一段时间被讨论最多的，就是数据安全的重要性：在云计算大行其道的今天，多租户方案已成为潮流。而许多数据保护条例的实施，比如 GDPR，更是让“保证数据不泄漏”成为一个标准流数据平台的基本要求。这项要求包括从写入，到处理，到导出的一些系列措施，包括认证、访问控制及授权、端到端加密等等。

举个例子，如果一个 Kafka 集群可以被一个企业里面的多个部门所共享，如何控制哪些部门可以读取或写入哪些主题、哪些部门可以创建或删除哪些主题、谁可以看见别人创建的主题、以及与 Kafka 相关的其他服务和客户端，比如应该如何保护 Zookeeper、谁可以直接读写、哪些 Kafka Streams 或者 Kafka Connect 应用程序可以发起哪些管理请求等，都需要提供足够的控制手段。

在 2.0.0 版本里面，我们对此提供了一系列的改进，比如更细粒度的更细粒度的前缀通配符访问控制（KIP-290、KIP-227），支持 SASL/OAUTHBEARER（KIP-255），将委托令牌扩展到管理客户端（KIP-249），等等。

KIP-290、KIP-227：细粒度前缀通配符访问控制

在 2.0 以前，很多 Kafka 自身的访问控制（access control list）机制还

是粗粒度的。比如对“创建话题”这一访问方式的控制，只有“全集群”这一种范围。也就是说，对于任何一个用户来说，我们只能或者给予其在一个集群内创建任何话题的权限——比如说，这个 Kafka 集群的运维或者可靠性工程团队（Devops or SRE），或者不给予任何话题的创建权限。但是在一个多租户环境下，我们很可能会出现需要更细粒度的控制机制。比如一个 Kafka Streams 客户端，除了读取给予的源话题之外，还需要实时创建一下内部用于 data shuffling 和 change logging 的话题，但是给予其一个“全集群”的话题创建权限又太危险。

因而在 2.0 版本中，我们进一步细粒度化了很多权限，比如 KIP-290 就加入了前缀通配符（prefix wildcard）的范围，而 KIP-227 就将这种范围加入到了单个或多个话题创建的权限中。在这个机制下，用户可以被赋予单独一个话题基于其话题名的创建权限（literal topic），也可以被赋予多个话题基于其话题名前缀匹配的创建权限，比如可以创建所有名字开头为“team1-”的话题，等等。

总结

这些总结出来的经验，很多都是在将 Apache Kafka 推广到互联网以外的领域，比如银行金融，制造和零售业内的公司时发现的。我们观察到在各种不同的领域里面，公司内部工程团队的技术倚重和深度、事务逻辑的严谨性要求、对于实时性以及成本控制的权衡偏好都各有不同，因此用户对于“流数据处理”这一名词所代表的需求都或多或少有自己独特的定义，但是以上这些特征却是共同的。因此，我觉得作为一个逐步成为标准流数据平台的开源项目，Kafka 还很“年轻”，还有很多值得我们去探索，未来可期。

关于 2.0.0 版本，我们其实还有很多新的发布特性，比如新的 Kafka Streams Scala API 帮助 Scala 用户更好的利用 Scala typing system 进行编程，Kafka Connect REST extension plugin 对于认证和访问控制的支持等等。关于更多细节，欢迎大家阅读相关公告：<https://www.apache.org/dist/>

[kafka/2.0.0/RELEASE_NOTES.html](#)

作者介绍

王国璋，Apache Kafka PMC，Kafka Streams 作者。分别于复旦大学计算机系和美国康奈尔大学计算机系取得学士和博士学位，主要研究方向为数据库管理和分布式数据系统。现就职于 Confluent，任流数据处理系统架构师和技术负责人。此前曾就职于 LinkedIn 数据架构组任高级工程师，主要负责实时数据处理平台，包括 Apache Kafka 和 Apache Samza 系统的开发与维护。

都去炒 AI 和大数据了，落地的事儿谁来做？

作者 杨雷



摘要

几乎每个企业都期望建立自己的完善的数据体系，但成功的例子并不多。本文希望用一些实践阐述以下几个观点：

- 数据产品应该朴实无华；
- 浮躁的认知会有大麻烦；
- 如何正确认识自己，如何敏捷。

前言

最近读到一篇文章 "SQL 足以解决你的问题，别动不动就是机器学习"，

教我们落地之法，在这个浮躁的世界中，犹如一股清流，实在大快人心。就像皇帝的新衣一样，终于有人说了出来。

有位做供应链数据分析的朋友很开心的说正在创业中，打算在供应链金融方面有一番作为，用神经网络的方法做用户画像，然后进行市场精准营销。作者工科数学博士一枚，每每看到有人探讨这么实际应用的东西，都觉得汗颜（自己不懂）与欣慰（越来越多人参与）并存，以至于给我已经是博导的师姐说，“好好鼓励你的弟子，数学系的春天来了！”

但是，要泼一下冷水，想必每个投身于大数据、人工智能的人士都碰到过某个瓶颈阶段，就是想要更深入了解原理的时候，那些公式算法实在是看不懂啊。每次我只能劝慰说，就当那是个黑盒，你只要知道输入输出，就能得到想要的结果。难道我要告诉实情其实是，最快你得花费半年到一年时间恶补数学知识，才能知道什么时候用模式识别，什么时候用小波分形，什么时候那个东西是动态规划。

这篇文章，继续泼冷水，“如果所有人都去做人工智能了，落地的事情谁来做？”，好比烧饭师傅都去研究自动炒菜机，在“懒人创造新的世界”之前，世界上的人都已经饿死了。认清自己手头要做的事情，比展望未来更关键，至少你能先存活下来。

为什么要数据产品

不论是初创、上升期、转型还是平台期的企业，回答好自己是谁，为谁服务，服务得如何，怎样更好的获利这几个问题，离不开数据。

从产品的角度看数据产品。

- Why? 很明显，企业需要看数据，用户需要看数据。不管是做战略计划、公司愿景、企业架构、运维治理、扩张市场、客户流失、目标营销，甚至做OKR、KSF、KPI、威士忌分析，或者告诉你的老板或下属做得有多成功或，，，多失败，你需要数据，这是你的价值。
- What? When? Who? 这是你的内容（范围和服务），你的视野

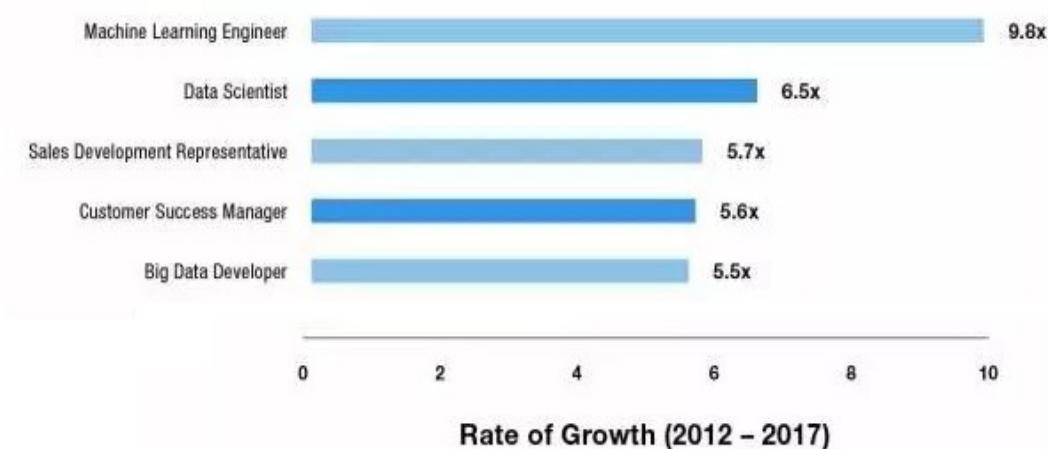
(过程和效率) , 你的上帝 (细分) 。

到底怎样做? 一个笨手笨脚的人 (Klutz) 都告诉你可以这样做。

- KNOW 找准自己的定位, 企业用户尚在起步阶段, 是没有能力去索取更多的数据的。此外, 还需要精通业务流程, 数据流离不开业务流, 不论你是To B还是To C, 把握好用户痛点和需求, 是首要的。
- LIGHT 不用再介绍一次KISS原则了吧。保持轻装上阵吧, 那样就算死, 也死得轻松。
- USE 动手吧, "想总是问题, 做才是答案"。试错是如此的关键, 一个企业是否有这个价值观, 甚至影响了是否最终的成败。后面会提到完美主义者, 是如何总是在关键时候触礁的。
- TELL 告诉你的用户或老板, 这个产品现在该有多糟糕, 虽然你和它已经竭尽全力在工作。奇迹是, 他们总会站在你这边。
- ZANY 莎士比亚造了这个词“滑稽”, 是让我们轻松点, 数据人已经很累了。“数据科学家”, 这个称号显然已经违背了这个原则, 背负了太多。我更倾向于数据分析师, 人人皆可当之。

数据产品的各种 " 圈钱 " 模式

让我们先来看看领英 2017 的一个岗位增长报告, 谁说大数据已死的?



曾几何时，作为数据库管理员或者 java 工程师的你也动心想深入了解一下何为数据科学，何为机器学习，何为大数据？别犹豫，其他人早就开始了（来自领英 2018 的行业报告）。

大 数据 使用 现 状



的人才招聘人员至少“有时”使用大数据

未 来



动辄“大”

一个很有趣的讨论，来自我和一位 BAT 数据分析师：

- “大”代表，首先，很fashion了；
- “大”代表平台很大，数据很多；
- “大”代表业务应用很广，至少传统方式做不了了；
- “大”代表0到1已经平安度过，深挖广种是时候了；
- “大”代表，有很多钱做事，需要你也很“贵”才行。

自然，我们在每一个评价后面，跟了一个“？””。但不管，就像项目竞标最好有个博士牵头一样，“大”代表着，新来的老板很喜欢。

动辄“智能”

同样，新来的老板更喜欢另外一个词“智能”，毋庸置疑的 Top One。作为数学专业出身的我，从来没想到过会有那么多人来问“神经网络”的算法怎样才能实现。他们都，疯了么？还是世上本无路，走的人多了，就有路了。每次我都用这个来安慰自己，这是一条光明之路，需要越来越多的

人前仆后继，不管你扛着的是步枪还是大炮。

- 图像处理，落地了。
- 语音处理，落地了。
- 还有？

我们是如何失败的

失败案例一：零售行业中的零库存

在本世纪初期，新零售流行“一单到底”和“零库存”这两个东西，愿望是美好的。我“不幸”也参与了其中对库存优化的计划中，那是一个零售业的 IT 供应商，为打造这个美好的愿景老板给了我一个艰巨的任务，3个月拿出一个算法实现先进的补货策略。

于是，加班加点，带着一群人搜索学习了各种算法对进货渠道、缺货周期、日销售情况进行了分析，最终开发出一个几千行代码几十个输入变量的程序，准备上马。

这时，老板问了一句，“这算法准么？某便利店商品 A 今天销售 20 件，库存只有 5 件，你算出来要补进 30 件，我排不过来货运啊？而且这两天卖得好是因为天热，过几天下雨咋办？”

最后，老板决定，还是按照老办法，盘点时由店长决定，快断货的时候补一周的货，灵活处理。

失败案例二：仓储行业中的自动化监控

2005 年，作为方案架构师，“有幸”参与了某大型跨国物流集团仓储中心产能监控系统设计。系统要求很简单，监控每个节点的容量、吞吐、以及排队情况，提供优化方案改善效率。

不知道谁头脑一热，前期要做一个非常漂亮的 3D 效果的模拟系统，还能显示每个热点并进行预警。于是乎，一个加大伯克利的博士（现不知所踪），一个清华的博士（现某外资银行做算法），一个人大硕士（现某金融系统分析员），一个交大博士（现某行业产品经理），开始学习

Photoshop 和 AutoCAD。悲惨的一幕随着数据从客户传来而开始，2000 多个线程并发跑，还是 B/S 的 3D 效果，性能可想而知。

被客户拿掉后，大家回顾说，还不如老老实实用 Excel 做几个表格和图形，能反映性能状态，发送问题原因，再研究下优化算法其实并不难。

失败案例三：教育行业中进度控制

这是一个 CRM 体系再造项目，用 Salesforce 替换原有老系统，作者参与的是其中 Business Intelligence 系统的再造，也就是俗称的企业报表系统。背景如下。

- 老板是完美主义者，需要漂亮的结果向投资人证明自己的成功。
- 用户对新产品信心不足，抗拒心很强，并不太配合前期需求调研和后期产品验收。
- 产品经理以及技术经理都是新人，并且有远大的做好事情的抱负。
- 开发人员 80% 都是新人，技术力量参差不齐；老员工属于内向型。

其实，它最终没有失败，只是所有人都累垮了。

- Salesforce 平台作为数据源，初期系统尚在开发中，技术经理考虑不想将来重复工作（rework），决定暂缓启动开发计划。这个决定直接导致中期项目进度确认时一无所有，于是被老板和项目高层标识为危险而责难，而后期用户伸手要数据时，各种没有准备也导致整个项目被推迟上线。分析：敏捷之大忌就是怕重复工作，那是设计分析能力问题，不是延迟工作的借口，谁说数据产品就不能敏捷？
- 到底是完全拷贝原来的系统 KPI，还是重新定义，以及是否要设计全新的前端展示？这个问题从一开始到结束，困扰了整个团队的每个人。老板严要求+产品新人+业务不配合+老员工的惰性，导致举步维艰。最后，一套七零八落的半成品系统展示在用户面

前，正确率和使用率很低。分析：从上往下剥离，老板要求的不一定就是对的（这往往无解），产品和业务必须在目标和方向上达成一致，以及技术决定生产力，这几点缺一不可，要突破却难上加难。

- 需求要考量教育平台学生成绩，一个学生某次考试会有各种技能的不同分数。问题来了，是需要数据准备到最细粒度，还是汇总聚合？爱好完美和细节的技术经理又出现了，一定要到最低粒度。不幸的是，由于项目进度紧迫，出现了各种设计和需求脱节以及数据不一致问题出现，各种会议讨论甚至互相指责随之而来。分析：还是敏捷问题，数据仓库权威 Ralph Kimball 是一个典型的细节专家，他所追求的细节是数据架构设计以及企业数据平台建设的愿景。但是，这个项目是一个典型的CRM系统切换，业务再造是基本目标，这时追求极致的细节变成了不切实际的要求，带来的后果就是本末倒置，所有人疲于其实不那么重要的问题上。

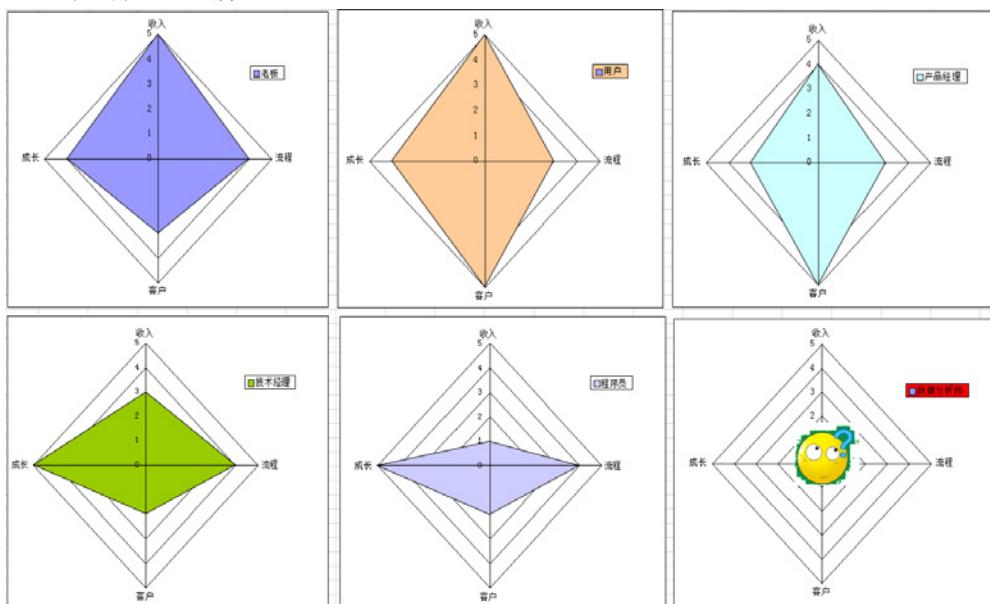
远离斜视

有位猎头顾问对我说，目前大数据分析师的岗位不多，我近乎惊讶的回答到，“怎么会，这个时代，你招人不说和大数据相关，都会觉得不够档次啊。事实总是证明我们是错的，拿开障目的那片叶子，正视真实需求，是多么难能可贵的企业家精神。

科学家是严谨的代名词，而大数据不需要严谨。是这样么？责任不同，视角也应该不同。

- 老板，720度看数据，3-5年规划中打算带着企业到什么样的数据成熟度 -- 这个成熟度一定和企业规模，组织管理，业务流程的成熟度成正比。
- 用户，360度看数据，这里把用户摆到很高的视角，因为他们不是傻子，是最知道怎么看和用数据的人。

- 产品经理，30-180度看数据，首要近视看眼前问题，不然会被用户骂死。也要远视看路线图，不然会被老板下岗。
- 技术经理，60-120度看数据，短期+长期规划，切忌操之过急，切忌漫无目的。
- 前后端程序员，90度看数据，那是你的两大支柱之一（算法+数据），多快好省是你的职责。
- 数据分析师，？？度看数据，你在哪儿，你去哪儿，你是谁，谁是你？想清楚。



图：不同视角看 Score Card

落地是如此的简单：80/20 原则

传统与自动化的纠缠，从古至今一直存在。再一次提及这篇令人爱恨交加的 "SQL 足以解决你的问题，别动不动就是机器学习"，如果传统方式能达到 95% 的精确度，够了么？

当我们在所有的算法中，对于圆周率的使用仅仅是 3.14 就已经足矣，又有多少人知道并在乎 3.1415926 后面的一位是 5 呢？

最后那 5% 的精准度，是红海最后的利润。这是收到最多的一个反驳的论点。但是当我们的企业，有超过 80% 的用户对数据的认知，还停留在填鸭阶段；当我们的运维还相当大程度依赖于半自动化，是不是该多花点心思写个 SQL 之类的。搭建数据产品的过程和企业以及用户的认知息息相关。

- 积累业务数据，重点在采集。Excel，SQL够了。
- 推送信息到用户，重点在快速提供。Excel，SQL够了。
- 自助式体验，重点在提升。Excel，SQL够么？
- 平台，重点在交互。Excel，SQL不够了。

认知的过程是相当漫长的，每一步都要踏踏实实落地，跑之前要学会走。

结束语

有客户问我何为敏捷？我的答复如下，不仅仅只针对数据产品。

- 我竭力面对任何一个需求，可能优先级上会有区别，可能个人能力上会理所不能及，或者让自己无法权衡处理好每一个事情。但我仍然愿意告诉每一个希望我帮助的人，我会竭力帮助他，哪怕其实我答应的实情超出了我的能力。
- 敏捷，本质就是靠近用户，有效沟通，快速迭代产品，不追求完美，要求脚踏实地。做产品就是要满足领导要求，要满足用户需求，而这两者常常会有冲突，就会很心累，这点在很多公司特别突出。这种情况，任何一个有丰富经验的项目管理者或者产品经理，都不能很好的协调。所以，搭建好领导，产品，用户几方之间的相互理解的桥梁，用用户导向的工作方式，尽量让你的方案能落地，尽量让目标达成一致而不是冲突，是每个做数据产品的人士应该牢记的工作原则。

如何才能写出好的软件设计文档？

作者 Angela Zhang 译者 无明



作为一名软件工程师，我花了很多时间在阅读和撰写设计文档上。在磨砺了数百篇文档之后，我发现，优秀的设计文档与项目的成功之间有着密切的联系。

这篇文章将介绍怎样才能写出一份优秀的设计文档。

为什么要写设计文档？

设计文档也就是技术规范，用于描述你打算如何解决问题。

已经有很多文章说明了为什么在开始编码之前要先写设计文档，这里不再赘述，不过我想再补充一句：

设计文档是确保正确完成工作最有用的工具。

设计文档的主要目的是迫使你对设计展开缜密的思考，并收集他人的反馈，以便更好地完成你的工作。人们通常认为，设计文档的目的是让其他人了解某个系统，或者作为参考文档。这些只是设计文档带来的有益的副作用，但绝不是它们的根本目的。

从经验来看，如果你的项目需要一个人月或更长时间，那么就应该编写设计文档。另外，一些小型项目也可以从迷你设计文档中获益。

但不同的工程团队，甚至是同一团队中的工程师，他们编写设计文档的方式存在很大差异。接下来，让我们来谈谈优秀设计文档的内容、风格和流程应该是怎样的。

设计文档应该包含哪些内容？

设计文档描述了问题的解决方案。因为每个问题的性质不一样，所以设计文档的结构也不一样。

下面列出了一个清单，你至少可以考虑在文档中包含这些内容。

标题和人物

设计文档的标题、作者（应该与计划参与项目的人员名单相同）、文档的评审人员（我们将在“流程”部分详细讨论），以及文档的最后更新日期。

摘要

摘要可以帮助公司的每位工程师理解文档的内容，并决定是否需要继续阅读文档的剩余部分。摘要最多可以包含 3 段内容。

背景

描述要解决什么问题、为什么需要这个项目、人们需要哪些知识才可以评估这个项目，以及它与技术战略、产品战略或团队的季度目标之间的关系。

目标和非目标

目标部分应该包括：

描述项目对用户的影响——你的用户可能是另一个工程团队或另一个系统。

指定如何使用指标来度量项目的成功——如果可以链接到指标仪表盘那就更好了。

非目标也同样重要，用于描述项目不会解决哪些问题，让每个人都达成共识。

里程碑

一系列可衡量的检查点，你的 PM 和上司可以借助它们大致了解项目的每个部分将在什么时间完成。如果项目时间超过 1 个月，我建议你将项目分解为面向用户的里程碑。

在设定里程碑时可以使用日历日期，把延期、假期、会议等因素都考虑进去。例如：

- 开始日期：2018年6月7日
- 里程碑1——以暗模式运行新系统MVP：2018年6月28日
- 里程碑2——弃用旧系统：2018年7月4日

结束日期：在新系统中添加新特性 X、Y、Z：2018 年 7 月 14 日

如果其中一些里程碑的 ETA 发生变化，需要在此处添加 [更新]，相关人员可以很容易看到更新的内容。

当前的方案

除了描述当前的实现之外，还应该通过示例来说明用户如何与系统发生交互以及数据是如何流经系统的。

这个时候可以使用用户故事。请记住，你的系统可能拥有不同类型的用户，他们的使用场景也不尽相同。

提议的方案

有人把这部分称为技术架构。这部分也可以通过用户故事来进行具体化，可以包含多个子部分和图表。

先从大处着手，然后填充细节。你要做到即使你在某个荒岛上度假，团队的其他工程师也能按照你的描述来实现解决方案。

其他替代方案

在提出上述解决方案的同时，你还考虑了其他的方案了吗？替代方案

的优点和缺点是什么？你是否考虑使用第三方解决方案——或使用开源解决方案——来解决问题，而不是自己构建解决方案？

监控和警报

人们经常把这部分视为马后炮，甚至直接忽略掉。而在发生问题后，他们就手忙脚乱，却不知道该怎么办，也不知道为什么会发生这些问题。

跨团队影响

这样会增加轮班待命和 DevOps 的负担吗？

它的成本是多少？

它会增加系统延迟吗？

它会暴露系统安全漏洞吗？

它会带来哪些负面后果和副作用？

支持团队该如何与客户沟通？

讨论

这部分可以包含任何你不确定的问题、你想让阅读文档的人一起权衡的有争议的决定、对未来工作的建议，等等。

详细的范围和时间表

这一部分的主要阅读对象是参与该项目的工程师、技术主管和经理。因此，这部分放在文档的最后。

从本质上讲，这是项目计划的实施方式和时间细分，可以包含很多内容。

我倾向于将这一部分视为项目任务跟踪器，因此每当范围估计发生变化时，我都会更新它。但这更像是个人偏好。

怎样才能写好文档

在讲完优秀的设计文档应该包含哪些内容之后，现在让我们来谈谈写作风格。

尽可能保持简单

不要把设计文档写成你曾经读过的学术论文。论文是为了给期刊评论

家留下深刻印象，而编写设计文档是为了描述你的解决方案，并从其他人那里获得反馈。你可以通过以下方式让文档变得更清晰：

- 用词简单
- 使用短句
- 使用符号列表和编号列表
- 提供具体的示例

添加图表

图表便于对几种选项进行比较，而且通常比文本更容易理解。我经常用 Google Drawing 来创建图表。

请记得在截图下方添加一个指向图表源文件的链接，以便在发生变更时可以更新图表。

包含数字

问题的规模通常决定了解决方案的规模。为了更好地帮助评审人员了解情况，请在文档中使用实际的数字，例如数据库行数、用户错误数、延迟，以及这些数据与使用量之间的关系。（还记得大 O 表示法吗？）

加入趣味性

记住，技术规范不是学术论文。另外，人们喜欢阅读有趣的东西，但也不要为了趣味性而偏离了核心思想。

自审

在将设计文档发给其他人评审之前，自己先假装是评审人员。你对这份设计文档有什么疑问？然后在发出文档之前把这些问题解决掉。

假期测试

如果你在独处，无法访问网络，那么团队中的其他人是否可以读懂这份文档，并按照你的意图实现文档中的内容？

设计文档的主要目的不是为了知识共享，但这是一种评估清晰度的好办法，让其他人可以为你提供有用的反馈。

流程

设计文档可以帮你在浪费大量时间实现错误的解决方案或解决错误的

问题之前获得反馈。获得良好反馈是一门艺术，但那是后话了。现在，让我们来讨论下如何为设计文档获取反馈。

首先，参与项目的每个人都应该参与设计过程。即使有很多决定是由技术主管做出的，那也没关系，至少每个人都应该参与讨论，并接收他们的设计。

其次，设计的过程并不一定是盯着白板，在上面画出各种想法。你可以动起手来，建立各种可能的解决方案原型。这与在写好设计文档之前就开始写代码是不一样的。你完全可以随意写一些一次性的代码来验证你的想法。为了确保你的代码只是用于概念验证，原型代码都不能合并到 master 上。

在你了解了如何进行项目之后，请执行以下步骤：

- 请你团队中经验丰富的工程师或技术负责人来评审你的文档。理想情况下，评审人员应该是一个受到尊重并且对问题边缘情况已经很熟悉的人。
- 在有白板的会议室里进行评审。
- 向评审人员描述你正在解决的问题（这是非常重要的一步，不要跳过它！）。
- 然后向评审人员解释你想要怎样实现你的想法，并说服他们这是正确的解决方案。

在开始正式编写设计文档之前完成所有这些步骤，可以让你在投入更多时间并敲定解决方案之前尽快获得反馈。通常情况下，即使实现方式可以保持不变，评审人员仍然会指出一些需要你覆盖到的极端情况，或者指潜在的不明确的地方，并预测你以后可能会遇到的困难。

然后，在你为设计文档写了粗稿之后，让之前的评审人员再次阅读它，并在设计文档的标题和人物部分写下他们的名字，作为对评审人员的鼓励，也表示他们是有责任对自己的评审行为负责的。

在添加评审人员名字时，可以考虑为特定的问题添加不同的评审人员（例如 SRE 和安全工程师）。

在完成评审之后，请将设计文档发给你的团队，以便获得额外的反馈。我建议把这个时间限制在 1 周左右，以避免延期。尽量在一周内解决他们留下的问题。

最后，如果你和评审人员及其他阅读该文档的工程师之间存在争议，我强烈建议你将这些争议放在文档的“讨论”部分。然后，与各方召开会议讨论这些分歧。

如果一个主题超过了 5 条评论，那么进行面对面讨论往往更有效率。请记住，即使无法让每个人都达成共识，你仍然可以拍板。

在最近与 Shrey Banga 谈论此事时，我了解到 Quip 有一个类似的过程，只是除了让团队中经验丰富的工程师或技术负责人来充当评审人员之外，他们还建议让不同团队的工程师评审设计文档。我没有尝试过这个，但可以肯定的是，这样有助于从拥有不同视角的人那里获得反馈，从而进一步提高文档的质量。

在完成上述所有步骤之后，接下来就可以开始实现了！在实现设计的过程中，可以将设计文档视为活动的文档。在对实施方案做出调整时，也一并更新文档，这样你就不需要向所有利益相关者反复解释这些变化。

那么，我们如何评估一份设计文档成功与否？

我的同事 Kent Rakip 对这个问题的回答是：如果能够获得正确的投资回报率（ROI），那么设计文档就是成功的。也就是说，一份成功的设计文档可能会导致：

1. 你花了 5 天时间编写设计文档，迫使你对技术架构的不同部分进行了充分的思考。
2. 你从评审人员那里获得反馈，知道架构中某些部分具有较高的风险。
3. 你决定先解决这些问题，以便降低项目的风险。
4. 3 天后，你发现这些问题要么不可能发生，要么比你原先想象的要困难得多。
5. 你决定停止这个项目去做其他工作。

在本文开头，我们说设计文档的目的是确保正确地完成工作。在上述的示例中，因为有了设计文档，你只花了 8 天时间而不是浪费了几个月才决定要中止项目。在我看来，这似乎也是一个非常成功的结果。

我用 Vue 和 React 构建了相同的应用程序，这是他们的差异

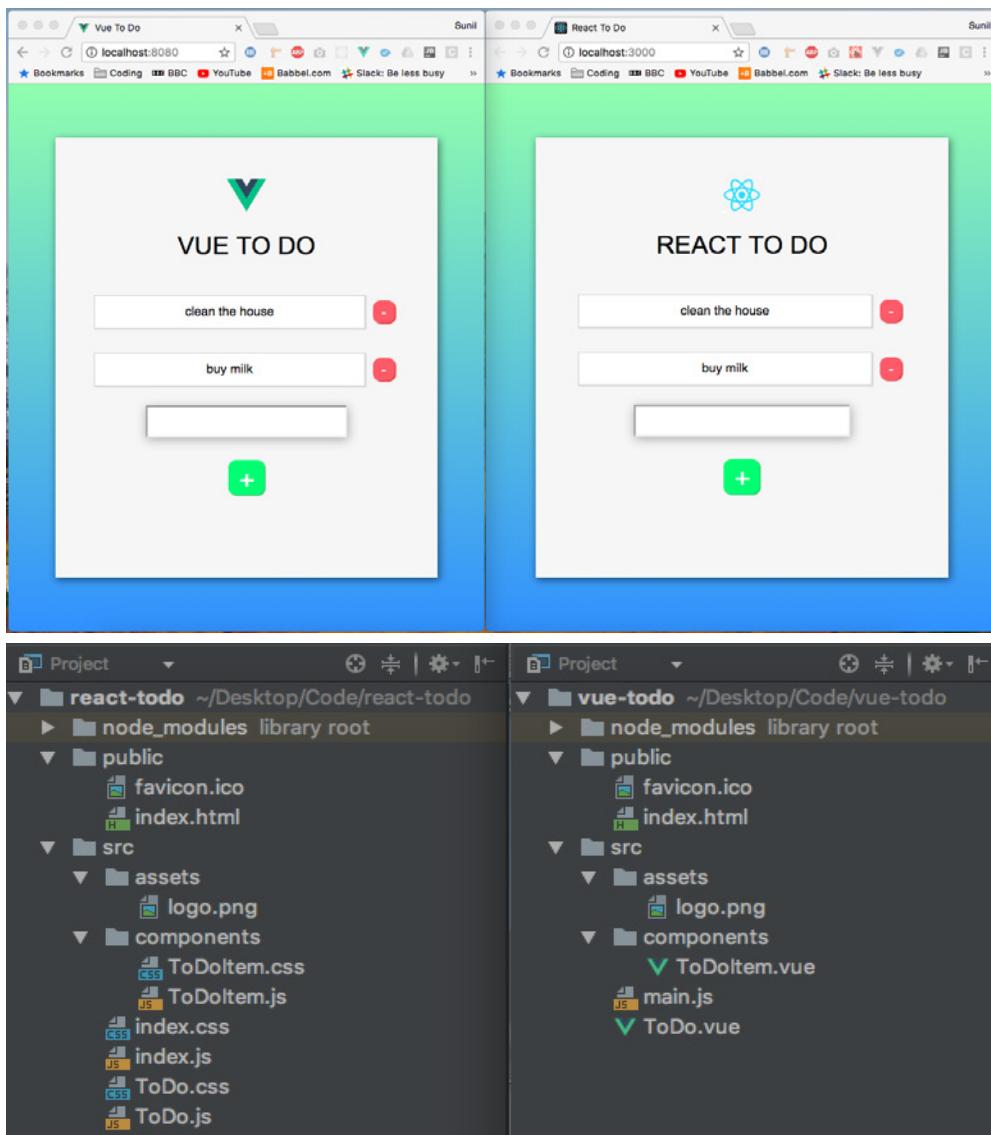
作者 Sunil Sandhu 译者 无明



在工作中使用了 Vue 之后，我已经对它有了相当深入的了解。同时，我也对 React 感到好奇。我阅读了 React 的文档，也看了一些教程视频，虽然它们很棒，但我真正想知道的是 React 与 Vue 有哪些区别。这里所说的区别，并不是指它们是否都具有虚拟 DOM 或者它们如何渲染页面。我真正想要做的是对它们的代码进行并排比较，并搞清楚在使用这两个框架开发应用时究竟有哪些差别。

我决定构建一个标准的待办事项应用程序，用户可以添加和删除待办事项。我分别使用它们默认的 CLI（React 的 `create-react-app` 和 Vue 的 `vue-cli`）来创建这个应用。先让我们看一下这两个应用的外观。

两个应用程序的 CSS 代码几乎完全相同，但代码存放的位置存在差别。



它们的结构也几乎完全相同，唯一的区别是 React 有三个 CSS 文件，而 Vue 则没有。这是因为 React 组件需要一个附带的文件来保存样式，而 Vue 采用包含的方式，将样式声明在组件文件中。

从理论上讲，你可以使用老式的 style.css 文件来保存整个页面的样式，这完全取决于你自己。不管怎样，还是展示一下 .vue 文件中的 CSS 代码长什么样。

看完样式方面的问题，现在让我们深入了解其他细节！

```
<template>
  <div class="ToDoItem" :key=id>
    <p class="ToDoItem-Text">{{todo}}</p>
    <div class="ToDoItem-Delete" @click="deleteItem(todo)"></div>
  </div>
</template>

<script>
  export default {
    name: "to-do-item",
    props: [
      'id', 'todo'
    ],
    methods: {
      deleteItem(todo) {
        this.$parent.$emit('delete', todo)
      }
    }
  }
</script>

<style>
  .ToDoItem {
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .ToDoItem-Text {
    width: 90%;
    background-color: white;
    border: 1px solid lightgrey;
    box-shadow: 1px 1px 1px lightgrey;
    padding: 12px;
    margin-right: 10px;
  }
</style>
```

我们如何改变数据？

我们说“改变数据”，实际上就是指修改已经保存好的数据。比如，如果我们想将一个人的名字从 John 改成 Mark，我们就要“改变数据”。这就是 React 和 Vue 的关键区别之一。Vue 创建了一个数据对象，我们可以自由地更新数据对象，而 React 创建了一个状态对象，要更新状态对象，需要做更多琐碎的工作。下面是 React 的状态对象和 Vue 的数据对象之间的对比。

从图中可以看到，我们传入的是相同的数据，它们只是标记的方式不一样。但它们在如何改变这些数据方面却有很大的区别。

假设我们有一个数据元素 name:'Sunil'。

在 Vue 中，我们通过 this.name 来引用它。我们也可以通过 this.

```

constructor(props) {
  super(props);
  this.state = {
    // this is where the data goes
    list: [
      {
        'todo': 'clean the house'
      },
      {
        'todo': 'buy milk'
      }
    ],
  };
}

data() {
  return {
    list: [
      {
        'todo': 'clean the house'
      },
      {
        'todo': 'buy milk'
      }
    ],
  };
}

```

name='John' 来更新它，这样会把名字改成 John。

在 React 中，我们通过 this.state.name 来引用它。关键的区别在于，我们不能简单地通过 this.state.name='John' 来更新它，因为 React 对此做出了限制。在 React 中，我们需要使用 this.setState({name:'John'}) 的方式来更新数据。

在了解了如何修改数据之后，接下来让我们通过研究如何在待办事项应用中添加新项目来深入了解其他细节。

我们如何创建新待办事项？

React:

```

createNewItem = () => {
  this.setState( ({ list, todo }) => ({
    list: [
      ...list,
      {
        todo
      }
    ],
    todo: ''
  })
};

```

Vue:

```
createNewItem() {  
    this.list.push(  
    {  
        'todo': this.todo  
    }  
);  
    this.todo = '';  
}
```

React 是怎么做到的?

在 React 中，input 有一个叫作 value 的属性。我们通过几个与创建双向绑定相关的函数来自动更新 value。React 通过为 input 附加 onChange 函数来处理双向绑定。

```
<input type="text"  
      value={this.state.todo}  
      onChange={this.handleInput}/>
```

只要 input 的值发生变化，就会执行 handleInput 函数。这个函数会将状态对象中 todo 字段的值改为 input 中的值。这个函数看起来像这样：

```
handleInput = e => {  
    this.setState({  
        todo: e.target.value  
    });  
};
```

现在，只要用户按下页面上的 + 按钮，createNewItem 就会调用 this.setState，并传入一个函数。这个函数有两个参数，第一个是状态对象的 list 数组，第二个是 todo（由 handleInput 函数更新）。然后函数会返回一个新对象，这个对象包含之前的整个 list，然后将 todo 添加到 list 的末尾。

最后，我们将 todo 设置为空字符串，它也会自动更新 input 中的值。

Vue 是怎么做到的?

在 Vue 中，input 有一个叫作 v-model 的属性。我们可以用它来实现

双向绑定。

```
<input type="text" v-model="todo"/>
```

v-model 将 input 绑定到数据对象 ToDoItem 的一个 key 上。在加载页面时，我们将 ToDoItem 设置为空字符串，比如 todo:”。如果 todo 不为空，例如 todo:'add some text here'，那么 input 就会显示这个字符串。我们在 input 中输入的任何文本都会绑定到 todo。这实际上就是双向绑定（input 可以更新数据对象，数据对象也可以更新 input）。

因此，回看之前的 createNewItem() 代码块，我们将 todo 的内容放到 list 数组中，然后将 todo 更新为空字符串。

我们如何删除待办事项？

React:

```
deleteItem = indexToDelete => {
  this.setState(({ list }) => ({
    list: list.filter((ToDo, index) => index !== indexToDelete)
  }));
};
```

React 是怎么做到的？

虽然 deleteItem 函数位于 ToDo.js 中，我仍然可以在 ToDoItem.js 中引用它，就是将 deleteItem() 函数作为 <ToDoItem/> 的 prop 传入：

```
<ToDoItem deleteItem={this.deleteItem.bind(this, key)}>
```

这样可以让子组件访问传入的函数。我们还绑定了 this 和参数 key，传入的函数需要通过 key 来判断要删除哪个 ToDoItem。在 ToDoItem 组件内部，我们执行以下操作：

```
<div className="ToDoItem-Delete" onClick={this.props.deleteItem}>-</div>
```

我使用 this.props.deleteItem 来引用父组件中的函数。

Vue:

```
this.$on('delete', (event) => {
  this.list = this.list.filter(item => item.todo !== event)
})
```

Vue 是怎么做到的？

Vue 的方式稍微有点不同，我们基本上要做三件事。

首先，我们需要在元素上调用函数：

```
<div class="ToDoItem-Delete" @click="deleteItem(todo)">-</div>
```

然后我们必须创建一个 emit 函数作为子组件内部的一个方法（在本例中为 ToDoItem.vue），如下所示：

```
deleteItem(todo) {  
  this.$parent.$emit('delete', todo)  
}
```

然后我们的父函数，也就是 this.\$on('delete') 事件监听器会在它被调用时触发过滤器函数。

简单地说，React 中的子组件可以通过 this.props 访问父函数，而在 Vue 中，必须从子组件中向父组件发送事件，然后父组件需要监听这些事件，并在它被调用时执行函数。

这里值得注意的是，在 Vue 示例中，我也可以直接将 \$emit 部分的内容写在 @click 监听器中，如下所示：

```
<div class="ToDoItem-Delete" @click="this.$parent.$emit('delete',  
todo)">-</div>
```

这样可以减少一些代码，不过也取决于个人偏好。

我们如何传递事件监听器？

React:

简单事件（如点击事件）的事件监听器很简单。以下是我们为添加新待办事项的按钮创建 click 事件的示例：

```
<div className="ToDo-Add" onClick={this.createNewToDoItem}>+</div>
```

非常简单，看起来很像是使用纯 JS 处理内联的 onClick 事件。而在 Vue 中，需要花费更长的时间来设置事件监听器。input 标签需要处理 onKeyPress 事件，如下所示：

```
<input type="text" onKeyPress={this.handleKeyPress}/>
```

只要用户按下了 'enter' 键，这个函数就会触发 createNewToDoItem 函

数，如下所示：

```
handleKeyPress = (e) => {
  if (e.key === 'Enter') {
    this.createNewToDoItem();
  }
};
```

Vue:

在 Vue 中，要实现这个功能非常简单。我们只需要使用 @ 符号和事件监听器的类型。例如，要添加 click 事件监听器，我们可以这样写：

```
<div class="ToDo-Add" @click="createNewToDoItem()">+</div>
```

注意：@click 实际上是写 v-on:click 的简写。在 Vue 中，我们可以将很多东西链接到事件监听器上，例如 .once 可以防止事件监听器被多次触发。在编写用于处理按键特定事件监听器时，还可以使用一些快捷方式。我发现，在 React 中为添加待办事项按钮创建一个事件监听器需要花费更长的时间。而在 Vue 中，我可以简单地写成：

```
<input type="text" v-on:keyup.enter="createNewToDoItem"/>
```

我们如何将数据传给子组件？

React:

在 React 中，当创建子组件时，我们将 props 传给它。

```
<ToDoItem key={key} item={todo} />
```

我们将 todo props 传给了 ToDoItem 组件。从现在开始，我们可以在子组件中通过 this.props 引用它们。因此，要访问 item.todo，我们只需调用 this.props.todo。

Vue:

在 Vue 中，当创建子组件时，我们将 props 传给它。

```
<ToDoItem v-for="item in this.list"
          :todo="item.todo"
          :key="list.indexOf(item)"
          :id="list.indexOf(item)">
</ToDoItem>
```

```
</ToDoItem>
```

然后，我们将它们加入到子组件的 props 数组，如：props:[id,'todo']。然后可以在子组件中通过名字来引用它们，入'id'和'todo'。

我们如何将数据发送回父组件？

React:

我们在调用子组件时将函数作为 prop 传给子组件，然后通过任意方式调用子组件的函数，这将触发位于父组件中的函数。我们可以在“如何删除待办事项”一节中看到整个过程的示例。

Vue:

在我们的子组件中，我们只需写一个函数，让它向父函数发回一个值。在父组件中，我们写了一个函数来监听这个值，然后触发函数调用。我们可以在“如何删除待办事项”一节中看到整个过程的示例。

示例代码链接：

Vue: <https://github.com/sunil-sandhu/vue-todo>

React: <https://github.com/sunil-sandhu/react-todo>

REST 将会过时，而 GraphQL 则会长存

作者 Samer Buna 译者 张卫滨



本文最初发布于 Medium 上 freeCodeCamp 的博客站点，经原作者 Samer Buna 授权由 InfoQ 中文站翻译并分享。

在处理过多年的 REST API 之后，当我第一次学习到 GraphQL 以及它试图要解决的问题时，我禁不住发了一条推文，这条推文的内容恰好就是本文的标题。



Samer Buna
@samerbuna



#REST APIs are now #RestInPeace APIs | Long live #GraphQL

12:30 AM - Sep 18, 2015



67 54 people are talking about this



当然，那个时候，我只是抱着好奇的心态进行了尝试，但现在，我相信当时的戏言却正在变成现实。

请不要误解，我并不是说 GraphQL 将会“杀死”REST 或类似的断定。REST 可能永远都不会消亡，就像 XML 永远也不会灭亡一样。我只是认为 GraphQL 之于 REST，就像 JSON 之于 XML 那样。

本文不会 100% 地鼓吹 GraphQL，这里面会有一个很重要的章节讨论 GraphQL 灵活性的代价。巨大的灵活性会带来巨大的成本。

简而言之：为何要使用 GraphQL？

GraphQL 能够非常漂亮地解决三个重要的问题：

为了得到视图所需的数据，需要进行多轮的网络调用：借助 GraphQL，要获取所有的初始化数据，我们仅需一次到服务器的网络调用。要在 REST API 中达到相同的目的，我们需要引入非结构化的参数和条件，这是很难管理和扩展的。

客户端对服务端的依赖：借助 GraphQL，客户端会使用一种请求语言，该语言：1) 消除了服务器端硬编码数据形式或数量大小的必要性；2) 将客户端与服务端解耦。这意味着我们能够独立于服务器端维护和改善客户端。

糟糕的前端开发体验：借助 GraphQL，开发人员只需使用一种声明式的语言表达用户的界面数据需求即可。他们所描述的是需要什么数据，而不是如何得到这些数据。在 GraphQL 中，UI 所需的数据以及开发人员描述数据的方式之间存在紧密的联系。

本文将会详细阐述 GraphQL 是如何解决这些问题的。

在开始之前，有些人可能还不熟悉 GraphQL，所以我们先给出一个简单的定义。

什么是 GraphQL？

GraphQL 是一门语言。如果我们将 GraphQL 传授给一个软件应用的

话，这个应用能够以声明式的方式与同样使用 GraphQL 的后端数据服务交流任意的数据需求。

孩子能够快速学习一门新的语言，而成人学起来就会更困难一些。与之类似，在一个新应用中从头开始使用 GraphQL 要比将其引入到一个成熟的语言中更容易一些。

要教会一个数据服务使用 GraphQL 语言，我们需要实现一个运行时层，并将其暴露给想要与服务通信的客户端。我们可以将服务器端的这个层视为一个简单的 GraphQL 翻译器，或者是讲 GraphQL 语言的代理，它代表了数据服务。GraphQL 不是一个存储引擎，所以它自己无法成为一个解决方案。这也是为什么我们无法具备一个只使用 GraphQL 语言的服务器，而是要实现一个转换运行时的原因。

这个运行时层，可以使用任何语言编写，它定义了一个基于图的通用模式（schema），该模式能够发布数据服务的能力（capabilities）。使用 GraphQL 语言的客户端应用能够在它的能力范围内查询该模式。这种方式将客户端和服务端进行了解耦，允许它们都能独立地演化和扩展。

GraphQL 可以是查询（读操作），也可以是变更（写操作）。在这两种情况下，请求都是一个简单字符串，该字符串能够被 GraphQL 服务解析、执行并以特定的格式解析数据。在移动和 Web 应用中，流行的响应格式是 JSON。

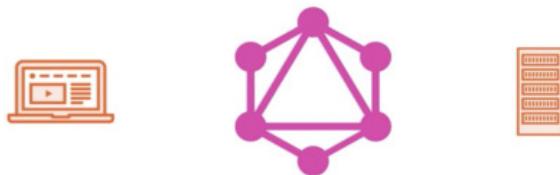
什么是 GraphQL ? (通俗讲解版)

GraphQL 就是关于数据通信的。我们有客户端和服务器端，它们之间都需要进行对话。客户端需要告诉服务器端它需要什么数据，而服务器端要以实际的数据满足客户端的需求。GraphQL 就位于这种通信之间。

你可能会问，我们为什么不能让客户端和服务器端直接通信呢？当然可以。

有多个原因促使我们在客户端和服务器端之间放置一个 GraphQL 层。其中有个原因，可能也是最常见的，那就是效率。客户端通常需要跟服务

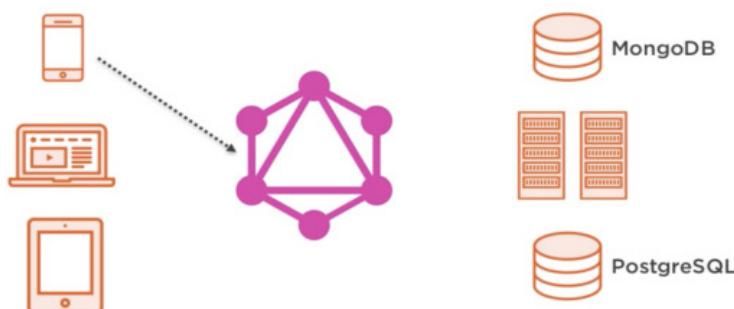
端要求多个资源，而服务端通常只能理解如何响应单个资源。所以，客户端需要发起多轮请求，这样才能收集到它需要的所有数据。



图片来源于作者 Pluralsight 课程的截图：使用 GraphQL 构建可扩展的 API

借助 GraphQL，我们可以将这种多请求的复杂性转移到服务端，让 GraphQL 层来对其进行处理。客户端对 GraphQL 层发起一个请求并且会得到一个响应，该响应中精确包含了客户端所需的内容。

使用 GraphQL 还会有许多收益，比如，另外一个收益就是与多个服务进行通信的时候。如果你有多个客户端要从多个服务请求数据的话，位于中间的 GraphQL 层能够简化和标准化这种通信。尽管这并不是针对 REST API 的（因为它也能很容易地实现），但是 GraphQL 运行时提供了一个结构化和标准化的方式来实现这一点。



图片来源于作者 Pluralsight 课程的截图：使用 GraphQL 构建可扩展的 API

客户端不会与两个不同的数据服务直接交互，我们现在可以让客户端与 GraphQL 层进行通信。然后，GraphQL 层会与两个不同的数据服务进行通信。这样的话，GraphQL 首先能够将客户端进行隔离，这样它们就没有必要使用多种语言进行通信了，同时，GraphQL 还会将一个请求转换为针对不同服务的多个请求，这些不同的服务可能会使用不同的语言编写。

让我们假设有三个不同的人，他们使用不同的语言并且具备不同类型的知识。假设你有一个问题，该问题需要组合这三个人的知识才能给出答案。如果你有一个能够说这三门语言的翻译器，那么为你的问题给出答案就会变得很容易。这其实就是 GraphQL 运行时所做的事情。

计算机还没有足够智能来回答任意的问题（至少目前还不可以），所以它们必须要在某些地方遵循一定的算法。这也是我们需要在 GraphQL 运行时上定义模式的原因，客户端会使用该模式。

基本上来讲，模式就是一个能力文档，它包含了客户端可以请求 GraphQL 层的所有问题的列表。在如何使用模式方面有一定的灵活性，因为我们在这里所讨论的是一个节点图。模式主要体现的是 GraphQL 层所能回答的问题都有哪些限制。

还感到不清楚吗？那我们一针见血地回答 GraphQL 是什么：REST API 的替代品。那么接下来，我们来回答你最可能会提出的一个问题。

那 REST API 有什么问题呢？

REST API 最大的问题在于其多端点的特质。这需要客户端进行多轮请求才能获取到想要的数据。

REST API 通常是端点的集合，其中每个端点代表了一个资源。所以，当客户端需要来自多个资源的数据时，就需要针对 REST API 发起多轮请求，这样才能将客户端所需的数据组合完整。

在 REST API 中，没有客户端请求语言。客户端对服务端返回的数据没有控制权。在这方面，没有语言能够帮助它们实现这一点。更精确地说，

客户端可用的语言非常有限。

例如，用来实现读取 (READ) 的 REST API 一般不外乎如下两种形式：

- GET /ResouceName：获取指定资源的所有记录的列表，或者
- GET /ResourceName/ResourceID：根据ID获取单条记录。

举例来说，客户端无法指定该选择记录中的哪个字段。这些信息位于 REST API 服务本身之中，不管客户端实际需要哪些字段，REST API 服务始终都会返回所有的字段。GraphQL 对该问题的描述术语是过度加载 (over-fetching) 不需要的信息。不管是对于客户端还是对于服务器端，这都是网络和内存资源的一种浪费。

REST API 的另外一个大问题是版本化。如果你需要支持多版本的话，通常意味着要有多个端点。在使用和维护这些端点的时候，这通常会导致更多的问题，而这也可能是服务端出现代码重复的原因所在。

上文所述的 REST API 的问题恰好是 GraphQL 所要致力解决的。上面所述的这些肯定不是 REST API 的所有问题，我也不想过多讨论 REST API 是什么，不是什么。我主要讲的是基于资源的 HTTP 端点 API。这些 API 最终都会变成常规 REST 端点和自定义专门端点的混合品，其中自定义的专门端点大多都是因为性能的原因而制作的。在这种情况下，GraphQL 能够提供好得多的方案。

GraphQL 的魔力是如何实现的呢？

在 GraphQL 背后有着很多理念和设计决策，但是最为重要的包括：

- GraphQL模式是强类型的模式。要创建GraphQL模式，我们需要按照类型来定义字段。这些类型可以是原始类型，也可以是自定义类型，模式中的任何内容都需要一个类型。这种丰富的类型系统允许实现丰富的特性，比如具备内省功能的API，以及为客户端和服务端构建强大的工具；
- GraphQL将数据以Graph的形式来进行表示，而数据很自然的表现形式就是图。如果想要表示任意的数据，那正确的结构就是

图。GraphQL运行时允许我们以图API的方式来表示数据，该API能够匹配数据的自然图形形状；

- GraphQL具有一个声明式的特质来表示数据需求。GraphQL为客户端提供了一种声明式的语言，允许它们描述其数据需求。这种声明式的特质围绕GraphQL语言创建了心智模型，这与我们使用自然语言思考数据需求的方式非常接近，从而使得GraphQL API要比其他替代方案容易得多。

其中，正是由于最后一项理念，我个人认为 GraphQL 将是一个游戏规则的改变者。

这些都是高层级的理念，接下来让我们看一些细节。

为了解决多轮网络调用的问题，GraphQL 将响应服务器变成了只有一个端点。从根本上来讲，GraphQL 将自定义端点的思想发挥到了极致，将整个服务器变成了一个自定义的端点，使其能够应对所有的数据请求。

与这个单端点概念相关的另一个重要理念是富客户端请求语言（rich client request language），这是使用自定义端点所需要的。如果没有客户端请求语言的话，单端点是没有什么用处的。它需要有一种语言来处理自定义的请求并为该请求响应数据。

具备客户端请求语言就意味着客户端将会是可控的。客户端能够确切地请求它们想要的内容，服务器端则能够确切地给出客户端想要的东西。这解决了过度加载的问题。

在版本化方面，GraphQL 有一种非常有趣的做法，能够彻底避免版本化的问题。从根本上来讲，我们可以添加新的字段，而不必移除旧的字段，因为我们有一个图，从而可以通过添加节点来灵活地扩展这个图。所以，我们可以继续保留旧 API 的路径，并引入新的 API，而不必将其标记为新版本。API 只是不断增长而已。

这对于移动端尤为重要，因为我们无法控制它们使用哪个版本的 API。一旦安装之后，移动应用可能会多年一直使用相同版本的旧 API。在 Web 端，我们能够很容易地控制 API 的版本，我们只需推送并使用新

的代码即可。对移动应用来说，这样做就有些困难了。

还不完全相信吗？我们通过一个具体的例子来对 GraphQL 和 REST 进行一对一的比较如何？

RESTful API 与 GraphQL API 的样例

假设我们是开发人员，负责构建一个崭新的用户界面，展现《星球大战》电影及其角色。

我们要构建的第一个 UI 界面很简单：显示每个《星球大战》人物信息的视图。例如，Darth Vader 以及他在哪些电影中出现过。这个视图将会展现人物的姓名、出生年份、星球名称以及他们所出现的电影的名字。

听起来非常简单，但实际上我们在处理三种不同类型的资源：人物（Person）、星球（Planet）以及电影（Film）。这些资源之间的关系很简单，任何人都可以猜测到数据的形状。每个 Person 对象属于一个 Planet 对象，同时每个 Person 对象有一个或多个 Film 对象。

这个 UI 的 JSON 数据可能会如下所示：

```
{
  "data": {
    "person": {
      "name": "Darth Vader",
      "birthYear": "41.9BBY",
      "planet": {
        "name": "Tatooine"
      },
      "films": [
        { "title": "A New Hope" },
        { "title": "The Empire Strikes Back" },
        { "title": "Return of the Jedi" },
        { "title": "Revenge of the Sith" }
      ]
    }
  }
}
```

假设某个数据服务能够为我们提供这种格式的数据，如下是使用 React.js 展现视图的一种可能的方式：

```
// The Container Component:  
<PersonProfile person={data.person} ></PersonProfile>  
  
// The PersonProfile Component:  
Name: {person.name}  
Birth Year: {person.birthYear}  
Planet: {person.planet.name}  
Films: {person.films.map(film => film.title)}
```

这是一个非常简单的样例，《星球大战》的体验可能会为我们带来一些帮助，UI 和数据的关系是非常清晰的。UI 用到了 JSON 数据对象中所有的“key”。

现在，我们看一下如何使用 RESTful API 请求该数据。

我们需要一个人的信息，假设我们知道人员的 ID，暴露该信息的 RESTful API 预期将会是这样的：

`GET - /people/{id}`

这个请求将会为我们提供该人员的姓名、生日和其他信息。一个好的 RESTful API 还会给我们提供该人员的星球 ID 以及这个人员所出现的所有电影的 ID 数组。

该请求的 JSON 响应可能会像如下所示：

```
{  
  "name": "Darth Vader",  
  "birthYear": "41.9BBY",  
  "planetId": 1  
  "filmIds": [1, 2, 3, 6],  
  *** 其他我们并不需要的信息 ***  
}
```

为了读取星球的名称，我们需要调用：

`GET - /planets/1`

随后，为了读取电影的名称，我们还要调用：

```
GET - /films/1
GET - /films/2
GET - /films/3
GET - /films/6
```

从服务器端得到这六个响应之后，我们就可以将它们组合起来以满足视图的数据需求。

为了满足一个简单 UI 的数据需求，我们发起了六轮请求，除此之外，我们在这里的方式是命令式的。我们需要给出如何获取数据以及如何处理数据使其满足视图需求的指令。

如果你想要明白我的真实含义的话，那么你可以自行尝试一下。在 <http://swapi.co/> 站点上，《星球大战》的数据目前有一个 RESTful API。你可以去那里尝试构建我们的人员数据对象。所使用的 key 可能会略有差异，但是 API 端点是相同的。你需要六次 API 调用，除此之外，在这个过程中还会过度加载视图并不需要的信息。

当然，这仅仅是该数据的一种 RESTful API 实现方式而已。我们可能还会有更好的实现方式，让视图编写起来更加容易。例如，如果 API 服务器的实现能够嵌套资源并理解人员和电影之间的关联关系，那么我们通过该 API 来读取电影数据：

```
GET - /people/{id}/films
```

但是，纯粹的 RESTful API 可能并不会实现这些，我们需要请求后端工程师为我们创建这个自定义的端点。这就是 RESTful API 进行扩展的现实：我们只能添加自定义端点来有效满足不断增长的客户端需求。管理这样的自定义端点是非常困难的。

现在，我们再来看一下 GraphQL 的方式。GraphQL 在服务端拥抱了自定义端点的理念，并将其发挥到了极致。服务器只有一个端点，至于通道则无关紧要。如果你通过 HTTP 来实现的话，HTTP 方法也是无关紧要的。我们假设有一个通过 HTTP 暴露的 GraphQL 端点，其地址为 /graphql：

因为想要通过一轮请求就将数据获取到，所以需要有一种方式来向服务器表达完整的数据需求。我们通过一个 GraphQL 查询来实现这一点：

`GET or POST - /graphql?query={...}`

GraphQL 查询只是一个字符串，但是它需要包含我们所需数据的所有片段。此时，声明式的方式就能发挥作用了。

在中文中，会这样描述我们的数据需求：我们需要一个人员的姓名、出生年份、星球的名字以及所有相关电影的名称。在 GraphQL 中，这会翻译为：

```
{
  person(ID: ...) {
    name,
    birthYear,
    planet {
      name
    },
    films {
      title
    }
  }
}
```

再次阅读一下使用中文表达的需求，然后将其与 GraphQL 查询进行对比。你会发现，它们非常接近。现在，对比一下这个 GraphQL 查询和我们开始时所见到的原始 JSON 数据。GraphQL 查询与 JSON 数据的格式完全相同，唯一的差异在于“值（value）”部分。如果我们将其想象为问题和答案的关系的话，所提出的问题就是将答案语句刨除了答案值。

如果答案语句是：距离太阳最近的行星是水星。

对该问题进行表述时，一种非常好的方式就是将相同语句的答案部分刨除掉：距离太阳最近的行星是（什么）？

同样的关系可以用到 GraphQL 查询中。以 JSON 响应为例，我们将其中的“答案”部分（也就是 JSON 中的值）移除掉，最终就能得到一个 GraphQL 查询，它能够非常恰当地表述 JSON 响应所对应的问题。

现在，对比一下 GraphQL 查询和我们为数据所定义的声明式 React UI。GraphQL 查询中的所有内容都用到了 UI 之中，而 UI 中用到的所有

内容也都出现在了 GraphQL 查询中。

这是 GraphQL 非常强大的思想模型。UI 知道它所需要的确切数据，抽取需求相对是非常容易的。生成 GraphQL 是一项非常简单的任务，只需将 UI 所需的数据直接抽取为变量即可。

如果我们将这个模型反过来，它依然非常强大。有一个 GraphQL 查询之后，我们就能知道如何在 UI 中使用它的响应，这是因为查询与响应有着相同的”结构“。我们不需要探查响应就能知道如何使用它，我们甚至不需要任何关于该 API 的文档。它都是内置的。

SwAPI [站点](#) 将《星球大战》的数据托管为 GraphQL API。你可以在这里进行尝试，构建我们的人员数据对象。这里有些小的差异，如下给出了一个官方的查询，我们可以基于该 API 读取视图所需的数据（以 Darth Vader 为例）：

```
{
  person(personID: 4) {
    name,
    birthYear,
    homeworld {
      name
    },
    filmConnection {
      films {
        title
      }
    }
  }
}
```

这个请求给出的响应结构非常类似于我们视图所使用的结构，需要记住的是，我们在一轮请求中就得到了所有的数据。

GraphQL 灵活性的代价

完美的解决方案只可能出现在童话之中。GraphQL 带来了灵活性，同时也有一些值得关注的地方和问题。

GraphQL 所带来的一个非常重要的风险就是资源耗尽攻击（即拒绝服务攻击）。GraphQL 服务器可以通过过于复杂的查询来进行攻击，这种查询将会消耗尽服务器的所有资源。它也非常容易查询深层的嵌套关联关系（用户 -> 好友 -> 好友），或者使用字段别名多次查询相同的字段。资源耗尽攻击并不是 GraphQL 特有的，但是在使用 GraphQL 的时候，我们必须格外小心。

我们能够采取一些措施来缓解这种情况。我们可以在查询之前进行预先的成本分析，并限制人们可以消费的数据量。我们还可以实现超时功能，将消耗过长时间进行解析的请求杀掉。同时，因为 GraphQL 只是一个解析层，我们可以在 GraphQL 层之下，进行速度的限制。

如果我们试图保护的 GraphQL API 端点不是公开的，也就是只用于客户端（Web 或移动）的内部使用，那么可以使用白名单的方式，服务器只能执行预选得到许可的查询。客户端可以使用一个唯一的查询标识符，请求服务器执行预先许可的查询。Facebook 似乎采用了这种方式。

在使用 GraphQL 时，另外一个需要考虑的地方就是认证和授权。在 GraphQL 查询解析之前、之后或解析的过程中，我们需要在这方面进行处理吗？

要回答这个问题，我们可以将 GraphQL 视为自己的后端数据获取逻辑之上的一个 DSL（领域特定语言）。它只是一个分层，我们可以将其放到客户端和实际的数据服务（或多个服务）之间。

我们将认证和授权视为另一个分层。GraphQL 并不会为实际的认证和授权逻辑提供帮助。它也不是做这个的。但是如果你想要将这些分层放到 GraphQL 之后的话，我们可以使用 GraphQL 在客户端和限制逻辑之间传输访问 token。这非常类似于在 RESTful API 认证和授权时，我们所采取的做法。

在客户端数据缓存方面，GraphQL 也面临着更多的挑战。RESTful API 由于其字典（dictionary）的特性，因此更容易进行缓存。对应的地址给出数据，因此我们可以使用这个地址本身作为缓存的 key。

在使用 GraphQL 的时候，我们可以采用类似的基本方法，使用查询的文本作为 key 来缓存它的响应。但是，这种方式是有一定限制的，效率不高，并且会导致数据一致性方面的问题。多个 GraphQL 查询的结果很容易出现重叠，这种基本的缓存机制不能解决重叠的问题。

但是，在这方面有一个很好的解决方案。图查询（Graph Query）意味着图缓存。如果我们将 GraphQL 查询的响应规范化为一个扁平的记录集合，为每条记录提供一个全局唯一的 ID，那么我们就可以缓存这些记录，而不是整个响应。

不过，这并不是一个简单的过程。记录会引用其他的记录，我们将会管理一个循环图。填充和读取缓存需要遍历查询。我们需要编码实现一个分层来处理缓存逻辑。但是，总体而言，这种方式要比基于响应的缓存高效得多。Relay.js 是采用该缓存策略的一个库，它会在内部进行自动管理。

关于 GraphQL，我们最需要关注的问题可能就是所谓的 N+1 SQL 查询。GraphQL 的查询字段被设计为独立的函数，在数据库中为这些字段解析获取数据可能会导致每个字段都需要一个新的数据库请求。

对于简单的 RESTful API 端点，可以通过增强的结构化 SQL 查询来分析、检测和解决 N+1 查询问题。GraphQL 动态解析字段，因此并没有那么简单。幸好，Facebook 正在通过 DataLoader 方案来解决这个问题。

顾名思义，DataLoader 是一个工具，我们可以借助它从数据库中读取数据，并将其提供给 GraphQL 解析函数使用。我们可以使用 DataLoader 读取数据，避免直接使用 SQL 查询从数据库中进行查询，DataLoader 将会作为我们的代理，减少实际发往数据库中的 SQL 查询。

DataLoader 组合使用批处理和缓存来实现这一点。如果相同的客户端请求需要向数据库查询许多内容的话，DataLoader 能够合并这些问题，并从数据库中批量加载问题的答案。DataLoader 还会对答案进行缓存，后续的问题如果请求相同的资源的话，就可以使用缓存了。

人工智能与区块链初探：交集与前瞻

作者 Matt Turck 译者 王强

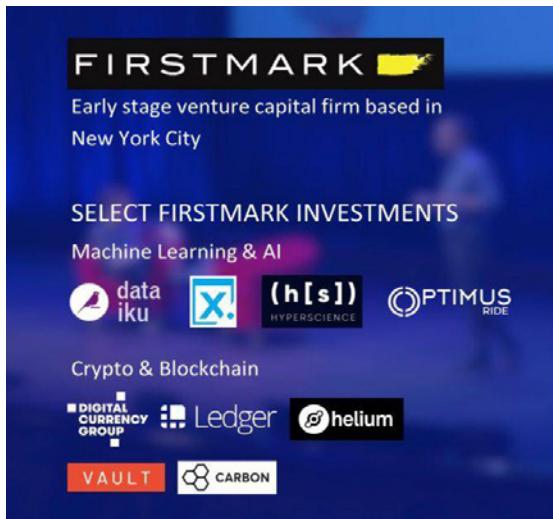


最近在纽约举办的的 Brains and Chains [会议](#)上，我荣幸地受 Rob May 和 Botchain 团队邀请发表了演讲。这次有趣的会议旨在探索人工智能与区块链的交集。

这是一个激动人心并具有挑战性的话题。而我的演讲是希望能做一个宽泛的介绍，并为之后的讨论建立框架：首先探讨为什么这一话题有着重大意义，并介绍这一领域一些有趣的企业所做的工作。

下面的图片都是 PPT 截图，文末有 PPT 的完整链接，其中有一些相关注释。

我是以 VC 风投的视角谈论这一主题的。我的投资机构 FirstMark 最近在人工智能和加密资产 / 区块链领域都很活跃。



MATT TURCK
Managing Director
Twitter: [@mattturck](#)
Blog: [mattturck.com](#)



对这一话题一笑置之当然也是可以理解的。因为无论是人工智能（机器学习）还是区块链，都有明显的实验性，都充斥着炒作和泡沫。人工智能在 2016 到 2017 年炒得最火，区块链则在 2017 到 2018 年最受关注。这两大趋势最后都可能令人失望，它们的交集也可能毫无作为。

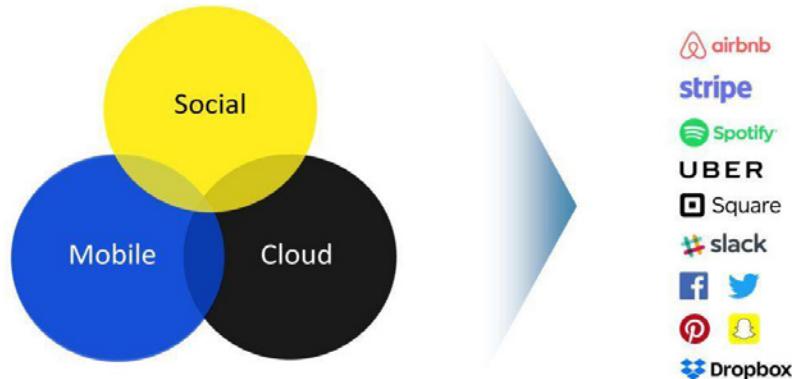
但如果我们回顾计算技术的历史，似乎每 10 到 15 年就会有重大的变革：硅芯片、PC、互联网、Web2.0 等等。

我们当前可能处于现在这波趋势的末端。这波趋势的三大推动力分别是：社交网络、移动计算和云计算。

今天我们熟知的许多巨头都是从这一趋势中崛起的。

当然，这些趋势的宏大前景并不总是那么明显。

Defining Technologies of the Last Decade



But Only Obvious in Hindsight



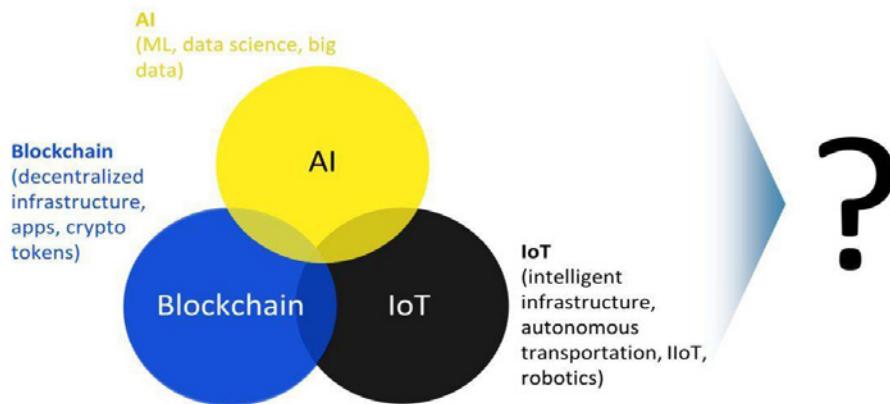
比如，云计算现在可能是公认的大趋势。但如果回到 2008 年，云计算那时还饱受争议，有些人认为它只是“营销噱头”。最后，云计算用了十多年才成为今天的巨型产业。

一开始，新趋势通常看起来是高度实验性的，伴随着过度炒作；但随着时间流逝它们愈加成熟，吸引了更多的资本和人才，逐渐成为新的主导模式。

一如阿马拉定律所言，新技术的影响在短期往往被高估，而在长期被低估。

现在，似乎已经到了新技术范式出现的时候。谁将定义并掀起下一波计算变革的浪潮？

Defining Technologies of the Next Decade

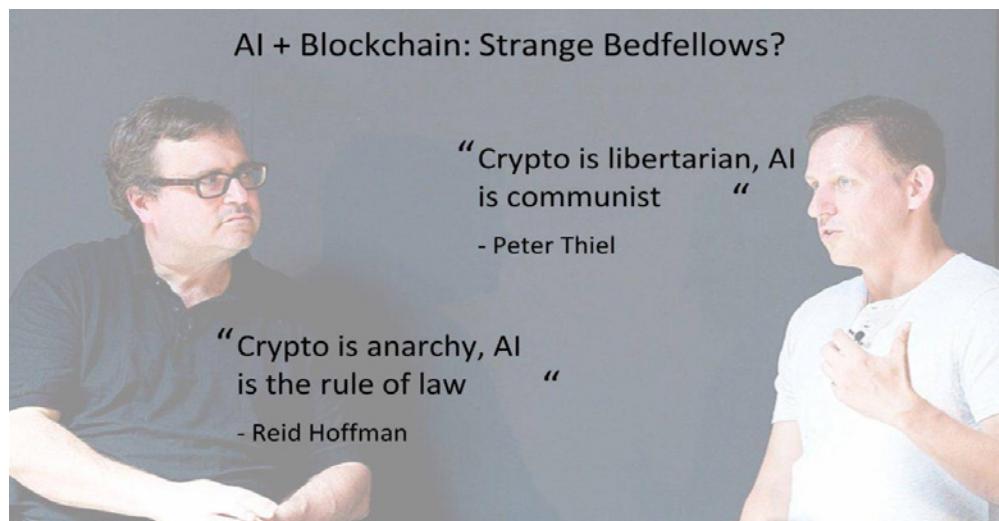


有理由相信，“人工智能、区块链和物联网”正是新时代的“社交网络、移动计算和云计算”。这些趋势仍在其发展初期，但它们的潜在影响难以估量。

这一范式中将产生哪些新的巨头？

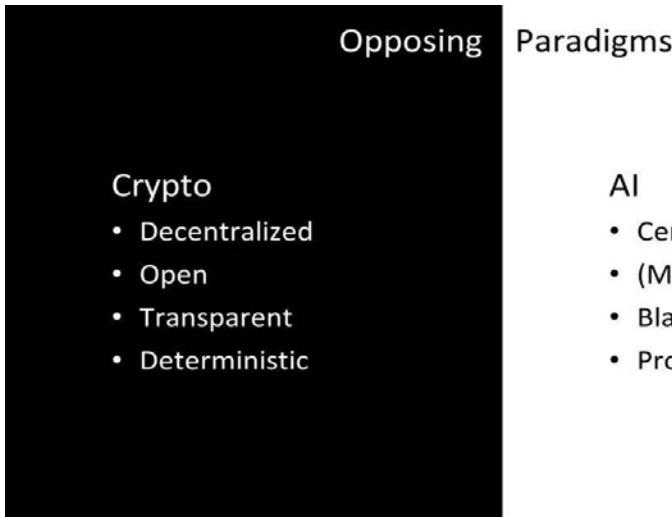
正如社交、移动、云计算相互之间彼此促动一样，上面这三大趋势也有着有趣的重叠部分。之前我在“物联网与区块链的交集”中讲了一个[例子](#)，类似的例子还有很多。

今天，我会主要谈人工智能与区块链的交集。



一个有趣的切入点是，从哲学角度来看人工智能和区块链在很多方面

都是对立的。Peter Thiel 和 Reid Hoffman 在最近一次对话中很好地总结了这一点：

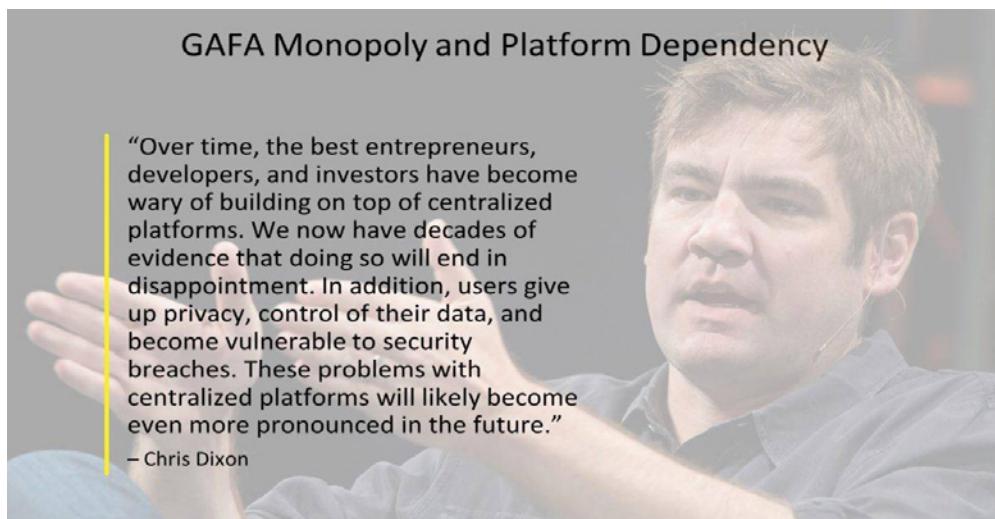


比如说，人工智能是非常中心化的。它由少量公司控制，主要是谷歌、苹果、Facebook 和亚马逊（“GAFA”），以及中国的阿里巴巴、腾讯和百度。尽管一些人工智能研究在学术界是开源的，但这些公司吸引了全球顶尖的人工智能人才；更重要的是，它们拥有规模史无前例的数据来训练人工智能算法。这些数据集为它们带来了巨大的竞争优势，却不对外界任何人开放。



人工智能的中心化为各种形式的滥用大开方便之门。专制国家政府使用计算机视觉和面部识别技术加强监控就是一个例子。

就在过去几个月，美国出现的一系列事件让人们想起了专制国家的类似案例，说明这已经发展为全球性的问题。



除了这类政治问题之外，中心化平台还会打压它们周围出现的新生态。可以读一读 Chris Dixon 的[这篇高论](#)：“为什么去中心化如此重要”。

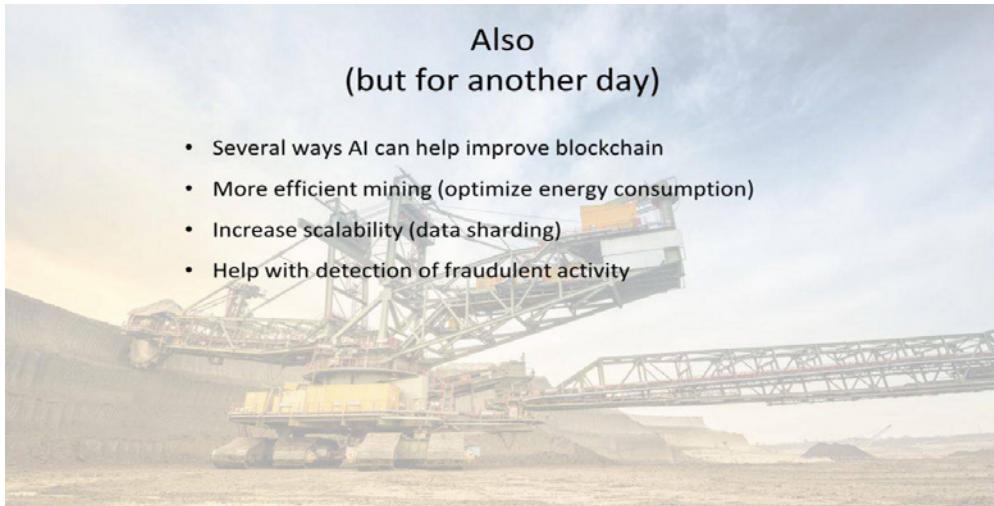


区块链不仅是纯粹的技术解决方案，也是对政治和组织问题的有力回应。

我们前面谈到的很多话题本质上都是政治和组织问题。那么可以用区块链来解决人工智能的这些缺陷吗？

区块链能帮助改进人工智能吗？

这一领域的先驱者一直在探索各种想法，包括使用去中心化的方式创建人工智能、自主机器网络，以及由人工智能管理的全自主组织。



今天我们谈论的是如何用区块链改进人工智能，但应该意识到人工智能也能在很多方面帮助区块链技术，这也是个很有趣的话题，改日再谈。

Decentralized AI Marketplaces: The Concept

What if...

- ... every individual and business could provide their data completely privately and securely in an open data exchange?
- ... AI models could compete with each other to provide the best results?
- ... everyone could be compensated fairly for participating in the above?

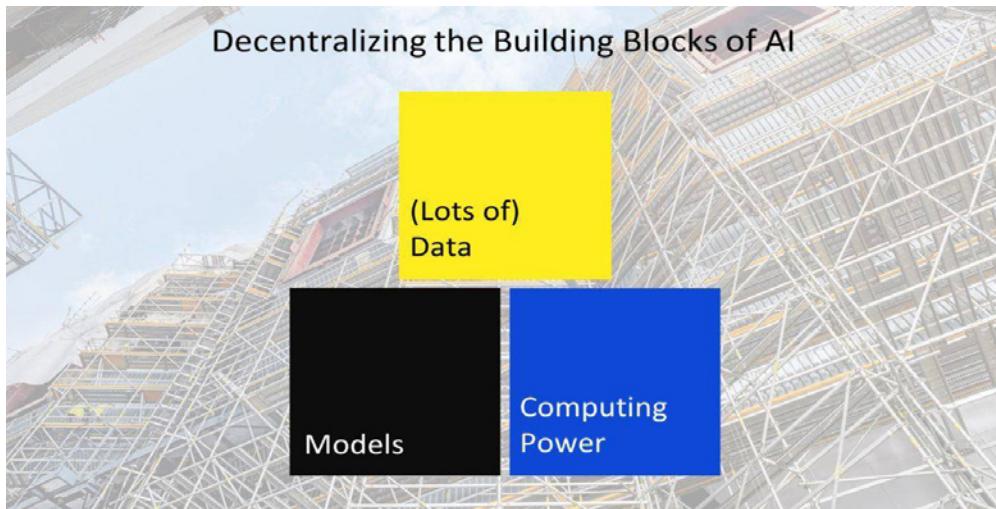
Wouldn't we...

- ... end up with MORE data than is currently available to GAFA and other centralized entities?
- ... also have NEW and BETTER data?
- ... and therefore better models, and better AI?
- ... but also more transparent AI

第一大设想是创造一个去中心化的市场以改进人工智能。

总体思路如下：用物质奖励来激励我们所有人（个体与组织）贡献自己的私有和专业数据。因为数据分享时会完全保证安全性和隐私性（基于去中心化与安全计算技术），我们就会更愿意分享各种隐私数据（开支、健康信息等）。久而久之，这些市场积累的数据会比 GAFA 还要多、质量还更高。基于这些数据，平台以经济利益激励机器学习专家互相竞争，

开发出最高效模型的专家将获得最多的奖励。



要探索如何构建这样一个去中心化市场，我们看看怎样将人工智能的三大组成元素进行去中心化：它们是数据、模型和算力。

接下来我们会展示很多企业的案例，这些企业在融合人工智能与区块链技术方面取得了杰出的成果。这一领域生机勃勃、进展迅速，所以少数案例也无法覆盖所有成功的企业和项目。

还要注意几点：这一行业的许多企业都有着极富野心的计划，要建立各种形式的生态系统。不过这些方案中有很多看上去很相似。这些项目大多尚未发布，所以在尘埃落定之后我们才能看到谁才是真正做事情的。

Wait... isn't the blockchain bad infrastructure for AI?

Among other issues:

- Not scalable
- Heterogenous nodes → hard to arrive at the same machine learning outputs

 **Tuur Demeester**
@TuurDemeester Following

blockchain: the world's worst database

Jimmy Song (宋珥盈) @jimmySong
Why you don't want a blockchain -

1. Storage: Everyone has to store everything
2. Bandwidth: Everything has to be broadcast to everyone...

8:55 PM – 15 Feb 2018

105 Retweets 333 Likes 

36 105 333

首先来看数据。很重要的一点：如果想使用区块链存储大量数据，现

有的区块链数据库远远不够，需要大幅进化才能满足需求。

BigchainDB: Building a Scalable Blockchain Database

Approach:

- Start with an enterprise-grade distributed database
- Engineer-in blockchain characteristics



	Bitcoin Blockchain	Distributed Database	BIGCHAINDB
Immutability	✓		✓
No Central Authority	✓		✓
Assets Over Network	✓		✓
High Throughput		✓	✓
Low Latency	✓		✓
High Capacity	✓		✓
Rich Permissioning	✓		✓
Query Capabilities	✓		✓

上图是来自柏林的 BigChainDB，它构建了一个可扩展的区块链数据库。这张很有趣的表格显示，分布式数据库与区块链技术的功能几乎没有重叠。因此，构建一个真正的数据库级区块链项目是很有挑战性的。

Ocean, Computable Labs: Building Decentralized Data Protocols

- A tokenized service layer to allow data to be shared and sold in a secure manner
- Stores metadata (i.e. who owns what), links to the data itself, and more
- On top of the protocol, there can be numerous data marketplaces and exchanges, all accessing the same data



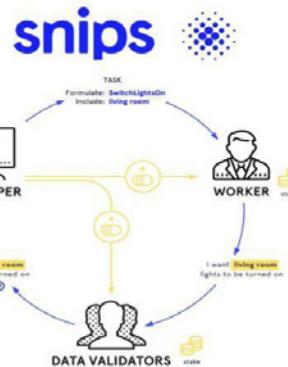
为了帮助数据共享，另一大关键的基础设施组件就是协议。

Ocean Protocol 是这一领域的先行者，有兴趣深入了解的话可以看看它的创始人 Trent McConaghy 所写的，关于区块链和人工智能的[所有内容](#)。Computable Labs 也致力于构建一个数据市场协议，其 CEO Roger Chen 的这篇文章也值得一读。

很多时候你需要创建自己的数据来训练人工智能模型，因为你无法接

Snips: Decentralized Data Generation

- Sometimes the data does not exist! (e.g. consumers have never spoken to their coffee machine before)
- Create “fake” user data by generating thousands of training examples
- Snips is creating a decentralized network incentivized by the upcoming Snips AIR token



入最合适的数据集，或者你用来训练模型的用例太新，根本就没有对应的数据。

来自巴黎的 Snips 正利用加密货币经济激励从业者创造一个网络，来生成合成数据。

Gems, Effect:
Decentralized Mechanical Turk for Data Labeling

- Decentralized, interactive marketplace for micro tasks that require human intelligence
- When workers compete a task, they are paid with a network token




Secure Computing

Goal:
Training models on data while keeping the data private



再来看看人工智能的第二大组成要素：模型。

要让一个去中心化的人工智能市场开始运作，你需要保证个人和公司提供的任何数据都能以完全私密的方式来处理。由此引入了安全计算。

OpenMined: Private Machine Learning

Protects **data owners** using:

- federated learning
- differential privacy



Protects **model owners** using:

- multi-party computation
- homomorphic encryption

OpenMined 项目就是一个很好的例子，它关注的重点是私密机器学习，其运用了各种安全计算技术，包括联合学习（由谷歌提出）和差分隐私（由苹果提出）技术。

Algorithmia: Selling Models

- Developed a protocol for crowdsourcing ML models
- Buyers supply data and offer rewards to incentivize model submissions
- Model validation takes place on a public blockchain
- If model meets requirements, payment is sent via smart contract



Numerai: Creating Competition Among Models

- Building a stock market meta-model through crowdsourcing
- Participants get data and in return submit models
- Incentives are aligned through staking and rewards
- Model validation is centralized but payment takes place via smart contract



DeepBrain Chain: Decentralized Cloud Computing Platform

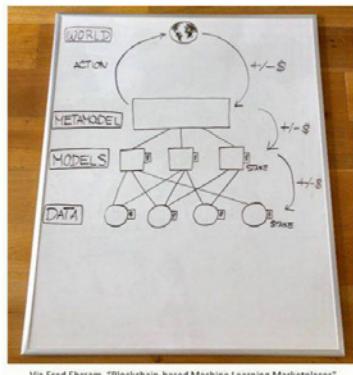
- Vision is to become “The AWS of AI”
- Help AI companies save up to 70% of computing costs
- Leverages idle GPU / FPGA clusters and individual computing units owned by SMEs



接下来看人工智能的第三大要素：算力。最近人工智能领域的许多进展都得益于算力的大幅提升，这既是因为我们更好地利用了现有的硬件，也受益于一些专为人工智能发展的新型高性能硬件（如谷歌 TPU 等）。

DeepBrain Chain 就是一个有趣的项目，致力于共享全世界空闲的计算资源。它的整体思想与 Coronai、Hadron、Golem 或 Hypernet 等项目类似，但 DeepBrain Chain 更关注符合人工智能特定需求的计算资源类型（与相关硬件）。

Putting It All Together: Decentralized Data Marketplaces



- Tokens & Crypto Economics to solve the cold start problem and incentivize participants
- Network effects:
“Multi-sided network effects from users, data providers, and data scientists make the system self-reinforcing.”
— Fred Ehrsam

将上述内容结合在一起，你应该能想象到一个去中心化的人工智能市场，其中人们贡献自己的数据，开发者竞相开发最好的机器学习模型，整个系统就像一个自进化的网络，吸引越来越多的参与者创造更好的人工智能技术。

这里的秘密武器其实是加密货币经济：就是创造一个小型经济生态，

参与者通过代币积累并交换价值。因为这种机制鼓励人们尽早加入网络，于是代币模式解决了曾让很多网络在发展初期头疼的冷启动问题。

上面图中的图表来自 Fred Ehrsam 在 Medium 发表的雄文：基于区块链的机器学习市场。[这篇文章](#)很好地描述了一个去中心化的人工智能市场是如何运作的。

当然，其他市场面临的典型机遇和挑战都适用于本文的讨论主题。去中心化的市场可能是一种创建人工智能的非常新颖的方式，但它产生的内容依旧需要符合产品 / 市场的需求并解决现实问题，这样才能取得商业上的成功。从这个角度来说，垂直因素（行业、基因、财务等）是尤其重要的。

Next Big Idea: Bots / AI networks



- Currently: lots of companies building “AI for X”: Specialized bots
- What happens when we have many bots covering individual tasks? Can they be combined for more ambitious projects?
- Can this be done in an open manner?

现在我们让话题再进一步。我们假设去中心化的市场会让人工智能继续繁荣并加速前进。我们就会为每一种任务都创造出对应的人工智能类型。那么这些人工智能会是怎样的形态，又如何运行？区块链能提供一个有趣的组织模型，帮助这些人工智能自动机器以一种透明的方式协作。

Fetch: Autonomous Economic Agents

- AEAs = autonomous digital entities that can transact independently of human intervention and can work together to construct solutions
- Open economic framework = “the ultimate value exchange dating agency”
- Smart ledger (scalable)
- Example use case: organize complex trip (predict misconnections, dynamically reroute trips, etc.)

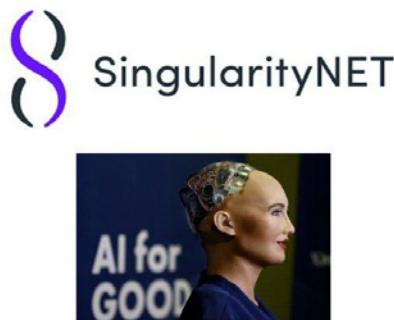


Fetch 就是这样一家[公司](#)，他们正在开发一种网络，用来创建人工智能自动机器，并让它们有序地协作。

自动机器之间协作的一个例子是旅行场景：比如你让一个自动机器买了张机票，如果航班延误了，另一个自动机器会预测转机失误的可能性，再提出一条新的路线，于是第一个自动机器就能改签了。这些都能在后台实时自动完成，完全不需要人力费心费力。

SingularityNET: “The Global AI Network”

- Decentralized marketplace for transacting with AI agents
- AI agents could include neural net tools, machine vision toolkits, etc.
- Uses staking and reputation to optimize discovery of the best AI agents
- AI agents can sub-contract tasks to more specialized AI agents



SingularityNET 是另一个有趣的[例子](#)。这个项目非常复杂、野心勃勃，有很多运动组件。为了展示不同的人工智能如何协作以融合为同一个大脑，他们开发出了索菲亚这个机器人，由 SingularityNET 驱动。这段[演示视频](#)效果惊人（令人想起了电视剧《西部世界》）。

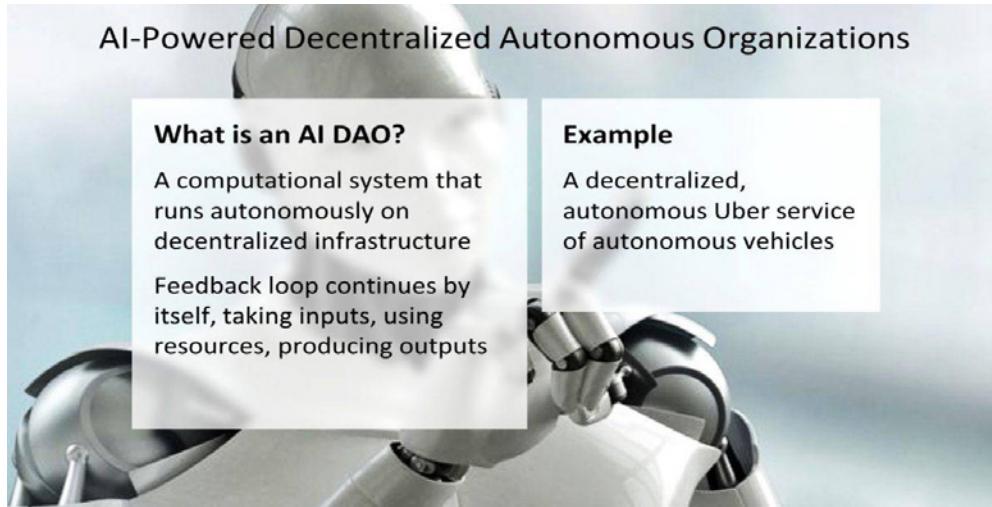
Botchain: Secure Identity System for Autonomous AI Agents

- Bot identity and validation
- Bot audit and compliance
- Control boundaries of autonomy
- Shared marketplace for bot add-ons



如果这个世界要依赖一群自动机器来执行各种任务，就需要一个能让它们保持透明、受控的基础架构。这正是 Botchain 的目标，这家公司由

Rob May 创立，CEO 是 Talla，并由 Brains and Chains 委员会管理。



在自动机器协作基础上再进一步，可以设想整个系统都由人工智能管理、完全自动运作。这也是去中心化自动组织（DAO）的理想。

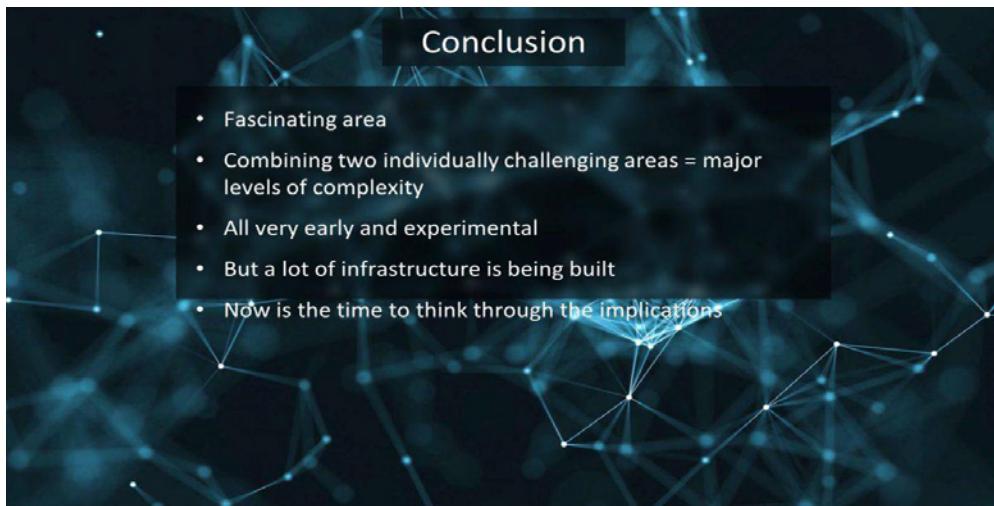
很多人听说过“那个 DAO”，这个投资者主导的风投基金在 2016 年曾被黑客入侵。

而“人工智能 DAO”的理念比 DAO 更进一步。它会是完全由机器来管理的去中心化组织，没有或很少收到人类的干涉。例如，你可以想象未来会出现一个由人工智能管理、完全去中心化的 Uber，运营自动驾驶汽车。它会包含一个巨大的反馈闭环，使系统可以持续学习改善派车算法、提升运客效率并处理所有逻辑，将众多技能和复杂操作融入一个自主运行的平



台中。

但这个人工智能 DAO 也会引发一个令人恐惧的设想：如果这样一个组织完全是去中心化、自动运行的话，它出问题的时候人们不知道怎样让它停下来。这可不像你拔掉电脑电源那么容易……



总结起来，上面很多想法都非常诱人，但也是高度实验性的。我提到的大部分项目都还没有面世，它们能不能实现自己追求的远大目标尚未可知。

同时，很多基础的工作已经做好了，于是人工智能和区块链的结合能够以难以预测的速度创造出非常强大的技术。有些进步可能会导致意料之外的结果，所以现在真应该好好研究一下相应的影响了。

感谢 Multicoin Capital 的 Kyle Samani 和 Computable Labs 的 Roger Chen 为这次演讲提供的反馈意见；感谢 Rob May 邀请我参加会议发表演讲；感谢 FirstMark 的 Demi Obayomi 为演讲内容做的重要贡献。

Istio 在 UAEK 中的实践改造之路

作者 陈绥



为什么需要 ServiceMesh

UCloud App Engine on Kubernetes（后简称“UAEK”）是 UCloud 内部打造的一个基于 Kubernetes 的，具备高可用、跨机房容灾、自动伸缩、立体监控、日志搜集和简便运维等特性计算资源交付平台，旨在利用容器技术提高内部研发运维效率，让开发能将更多的精力投入在业务研发本身，同时，让运维能更从容应对资源伸缩、灰度发布、版本更迭、监控告警等日常工作。

考虑到 Kubernetes 本来就是为自动部署、伸缩和容器化而生，再加上 UCloud UAEK 团队完成 IPv6 组网调研和设计实现后，一个成熟的容器

管理平台很快正式在北京二地域的多个可用区上线了。相比于过去申请管理虚拟机部署应用服务，Kubernetes 确实带来了实实在在的便利，例如方便灵活的自动伸缩以及触手可及的微服务架构，只需简单配置即可实现跨可用区容灾等。

然而，微服务化又为系统架构带来许多新的问题，例如服务发现、监控、灰度控制、过载保护、请求调用追踪等。大家已经习惯自行运维一组 Zookeeper 集群用以实现服务发现和客户端负载均衡，使用 UAEK 后能否免去运维 Zookeeper 的工作？为了监控业务运行状态，大家都需要在代码里加上旁路上报逻辑，使用 UAEK 是否能无侵入零耦合地实现监控上报？

此外，过去很多系统模块间调用缺少熔断保护策略，波峰流量一打就瘫，使用 UAEK 是否能帮助业务方免去大规模改造呢？过去排查问题，尤其是调用耗时环节排查总是费时费力，使用 UAEK 能否为定位瓶颈提供方便的工具？

显然，仅凭一个稳定的 Kubernetes 平台不足以解决这些问题。因此，在 UAEK 立项之初，团队就把 ServiceMesh 作为一个必须实现的目标，任何在 UAEK 上部署的 TCP 后台服务，都能享受到 ServiceMesh 带来的这些特性：

SideCar 模式部署，零侵入，微服务治理代码与业务代码完全解耦；

- 与 Kubernetes 平台融合的服务发现机制和负载均衡调度；
- 提供灵活，实时，无需重启、能根据 7 层业务信息进行流量灰度管理功能；
- 提供统一抽象数据上报 API 层，用于实现监控和访问策略控制；
- 使用分布式请求链路追踪系统，快速追溯 Bug，定位系统性能瓶颈；
- 过载保护机制，能在请求量超过系统设计容量时自动触发熔断；
- 能在服务上线前提供故障模拟注入演习剧本，提前进行故障处理演练。

这样，使用 UAEK 部署应用服务后，即可从小范围按账号灰度上线

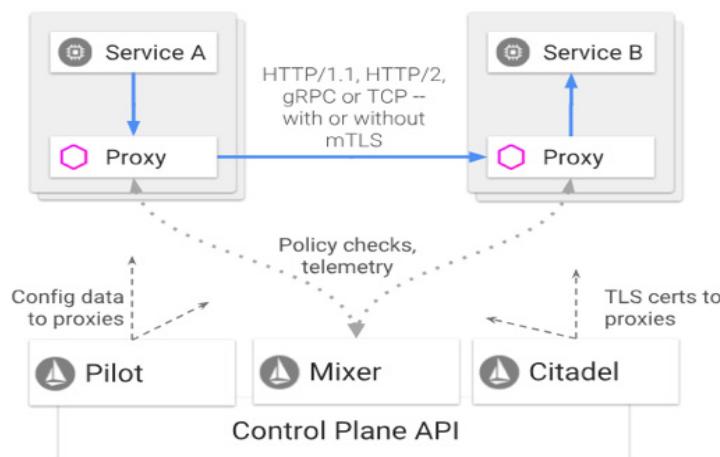
开始，通过陆续地监控观察，轻松掌握版本异常回退、扩大灰度范围、全量发布、过载保护、异常请求定位追踪等信息。

为什么是 Istio ?

关于 ServiceMesh 的实现，我们重点考察了 Istio。通过前期的调研和测试，我们发现 Istio 的几个特性能很好满足 UAEK 的需求：

- 完美支持Kubernetes平台；
- 控制面和数据转发面分离；
- Sidecar部署，掌控所有服务间调用流量，无上限的控制力；
- 使用Envoy作为Sidecar实现，Envoy使用C++11开发，基于事件驱动和多线程机制运行，性能好并发能力强，媲美NGINX；
- 对业务的代码和配置文件零侵入。

配置简单，操作方便，API 完善。



图一：Istio1.0 架构图

整个服务网格分成控制面板和数据面两大部分。数据面指的就是注入到应用 Pod 中的 Envoy 容器，它负责代理调度模块间的所有流量。控制面分为 Pilot, Mixer 和 Citadel 三大模块，具体功能如下：

Pilot 负责向 Kubernetes API 获取并 Watch 整个集群的服务发现信息，

并向 Envoy 下发集群服务发现信息和用户定制的路由规则策略。

Mixer 分为 Policy 和 Telemetry 两个子模块。Policy 用于向 Envoy 提供准入策略控制，黑白名单控制，QPS 流速控制服务；Telemetry 为 Envoy 提供了数据上报和日志搜集服务，以用于监控告警和日志查询。

Citadel 为服务和用户提供认证和鉴权、管理凭据和 RBAC。

此外 Istio 为运维人员提供了一个叫 istioctl 的命令行工具，类似 kubernetes 的 kubectl。运维编写好路由规则 yaml 文件后，使用 istioctl 即可向集群提交路由规则。

Istio 整体工作的原理和流程细节非常复杂，所涉及到的技术栈有一定的深度和广度。这里只概括一下大体过程：

- 运维人员使用 istioctl 或者调用 API 向控制层创建修改路由规则策略。
- Pilot 向 Kube APIServer 获取并 watch 集群服务发现信息。
- 部署应用程序时，Istio 会在 pod 的部署配置中注入 Envoy 容器，Envoy 会通过 iptables nat redirect 劫持代理 pod 中的全部 TCP 流量。
- Envoy 会实时从 Pilot 更新集群的服务发现信息和路由规则策略，并根据这些信息智能调度集群内的流量。
- Envoy 会在每次请求发送前向 Mixer Policy 发送 Check 请求检查该请求是否收策略限制或者配额限制，每次请求接收后会向 Mixer Telemetry 上报本次请求的基本信息，如调用是否成功、返回状态码、耗时数据。
- Citadel 实现了双向 TLS 客户端证书生成与注入，服务端密钥和证书的下发注入，以及 K8S RBAC 访问控制。

Istio 在 UAEK 环境下的改造之路

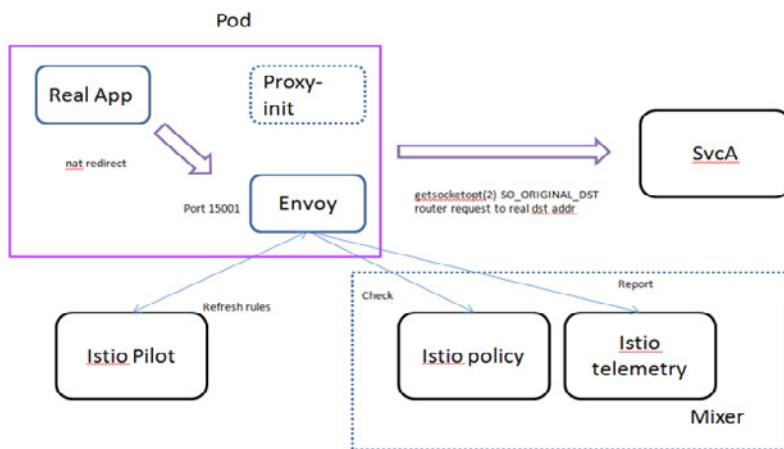
经过上述的调研和与一系列测试，UAEK 团队充分认可 Istio 的设计理念和潜在价值，希望通过利用 Istio 丰富强大的微服务治理功能吸引更多的内部团队将服务迁移到 UAEK 环境中。

然而，事实上，在 UAEK 上接入 Istio 的过程并非一帆风顺。最早开始调研 Istio 的时候，Istio 还在 0.6 版本，功能并不完善，在 UAEK 环境中无法开箱即用。

IPv6问题的解决

我们首先碰到的问题是，UAEK 是一个纯 IPv6 网络环境，而 Istio 对 IPv6 流量的支持并不完备，部分组件甚至无法在 IPv6 环境下部署。

在介绍具体改造案例之前，先了解下 Istio Sidecar 是如何接管业务程序的流量。



如上图所描述，Istio 会向应用 Pod 注入两个容器：proxy-init 容器和 envoy 容器。proxy-init 容器通过初始化 iptables 设置，将所有的 TCP 层流量通过 nat redirect 重定向到 Envoy 监听的 15001 端口。以入流量为例，Envoy 的服务端口接收到被重定向到来的 TCP 连接后，通过 getsockopt(2) 系统调用，使用 SO_ORIGINAL_DST 参数找到该 TCP 连接的真实目的地 IP 地址，并将该请求转发到真实目的 IP。

然而，我们发现在 IPv6 环境下，Envoy 无法劫持 Pod 的流量。通过抓包观察和追溯源码发现，Pod 启动的时候，首先会运行一个 iptables 初始化脚本，完成 pod 内的 nat redirect 配置，将容器内的 TCP 出入流量都

劫持到 Envoy 的监听端口中，但这个初始化脚本没有 ip6tables 的对应操作并且 discard 了所有 IPv6 流量，因此我们修改了初始化脚本，实现了 IPv6 的流量劫持。

一波刚平，一波又起。完成 IPv6 流量劫持后，我们发现所有访问业务服务端口的 TCP 流量都被 Envoy 重置，进入 Envoy 容器中发现 15001 端口并没有开启。追溯 Envoy 和 Pilot 源码发现，Pilot 给 Envoy 下发的 listen 地址为 0:0:0:0:15001，这是个 IPv4 地址，我们需要 Envoy 监听地址的为 [::]:15000，于是继续修改 Pilot 源码。

经过上述努力，应用服务端程序 Pod 终于能成功 Accept 我们发起的 TCP 连接。但很快，我们的请求连接就被服务端关闭，客户端刚连接上就立刻收到 TCP FIN 分节，请求依然失败。通过观察 Envoy 的运行日志，发现 Envoy 接收了 TCP 请求后，无法找到对应的 4 层流量过滤器 (Filter)。

深入跟进源码发现，Envoy 需要通过 getsockopt(2) 系统调用获取被劫持的访问请求的真实目的地址，但在 IPv6 环境下 Envoy 相关的实现存在 bug，如下代码所示。由于缺少判定 socket fd 的类型，getsockopt(2) 传入的参数是 IPv4 环境下的参数，因此 Envoy 无法找到请求的真实目的地址，遂报错并立刻关闭了客户端连接。

```
--  
291     Address::InstanceConstSharedPtr Utility::getOriginalDst(int fd) {  
292     #ifdef SOL_IP  
293         sockaddr_storage orig_addr;  
294         socklen_t addr_len = sizeof(sockaddr_storage);  
295         int status = getsockopt(fd, SOL_IP, SO_ORIGINAL_DST, &orig_addr, &addr_len);  
296  
297         if (status == 0) {  
298             // TODO(mattklein123): IPv6 support. See github issue #1094.  
299             ASSERT(orig_addr.ss_family == AF_INET);  
300             return Address::InstanceConstSharedPtr{  
301                 new Address::Ipv4Instance(reinterpret_cast<sockaddr_in*>(&orig_addr));  
302         } else {  
303             return nullptr;  
304         }  
305     #else
```

发现问题后，UAEK 团队立刻修改 Envoy 源码，完善了

getsockopt(2) 的 SO_ORIGINAL_DST 选项的 IPv6 兼容性，然后将这一修改提交到 Envoy 开源社区，随后被社区合并到当前的 Master 分支中，并在 Istio1.0 的 Envoy 镜像中得到更新使用。

到此为止，Istio SideCar 终于能在 UAEK IPv6 环境下正常调度服务间的访问流量了。

此外，我们还发现 Pilot、Mixer 等模块在处理 IPv6 格式地址时出现数组越界、程序崩溃的情况，并逐一修复之。

性能评估

Istio1.0 发布之前，性能问题一直是业界诟病的焦点。我们首先考察了增加了 Envoy 后，流量多了一层复制，并且请求发起前需要向 Mixer Policy 进行一次 Check 请求，这些因素是否会对业务产生不可接收的延迟。经过大量测试，我们发现在 UAEK 环境下会比不使用 Istio 时增加 5ms 左右的延迟，对内部大部分服务来说，这完全可以接受。

随后，我们重点考察了整个 Istio Mesh 的架构，分析下来结论是，Mixer Policy 和 Mixer Telemetry 很容易成为整个集群的性能短板。由于 Envoy 发起每个请求前都需要对 Policy 服务进行 Check 请求，一方面增加了业务请求本身的延迟，一方面也给作为单点的 Policy 增大了负载压力。我们以 Http1.1 请求作为样本测试，发现当整个网格 QPS 达到 2000-3000 的时候，Policy 就会出现严重的负载瓶颈，导致所有的 Check 请求耗时显著增大，由正常情况下的 2-3ms 增大到 100-150ms，严重加剧了所有业务请求的耗时延迟，这个结果显然是不可接受的。

更严重的是，在 Istio 0.8 以及之前的版本，Policy 是一个有状态的服务。一些功能，如全局的 QPS Ratelimit 配额控制，需要 Policy 单个进程记录整个 Mesh 的实时数据，这意味着 Policy 服务无法通过横向扩容实例来解决性能瓶颈。经过取舍权衡，我们目前关闭了 Policy 服务并裁剪了一些功能，比如 QPS 全局配额限制。

前面提到过，Mixer Telemetry 主要负责向 Envoy 收集每次请求的调用

情况。0.8 版本的 Mixer Telemetry 也存在严重的性能问题。压测中发现，当集群 QPS 达到 2000 以上时，Telemetry 实例的内存使用率会一路狂涨。

经过分析定位，发现 Telemetry 内存上涨的原因是数据通过各种后端 Adapter 消费的速率无法跟上 Envoy 上报的速率，导致未被 Adapter 处理的数据快速积压在内存中。我们随即去除了 Istio 自带的并不实用的 stdio 日志搜集功能，这一问题随即得到极大缓解。幸运的是，随着 Istio1.0 的发布，Telemetry 的内存数据积压问题得到解决，在相同的测试条件下，单个 Telemetry 实例至少能胜任 3.5W QPS 情况下的数据搜集上报。

问题、希望与未来

历经重重问题，一路走来，一个生产环境可用的 ServiceMesh 终于在 UAEK 环境上线了。在这一过程中，也有部门内其他团队受 UAEK 团队影响，开始学习 Istio 的理念并尝试在项目中使用 Istio。然而，目前的现状离我们的初心依然存在差距。

Istio 依然在高速迭代中，无论是 Istio 本身还是 Envoy Proxy，每天都在演进更新。每一次版本更新，带来的都是更为强大的功能，更为简练的 API 定义，同时也带来了更复杂的部署架构。从 0.7.1 到 0.8，全新的路由规则 v1alpha3 与之前的 API 完全不兼容，新的 virtualservice 与原先的 routerule 截然不同，给每位使用者构成了不少麻烦。

如何完全避免升级 Istio 给现网带来负面影响，官方依然没有给出完美平滑的升级方案。此外，从 0.8 到 1.0 虽然各个组件的性能表现有显著提升，但从业内反馈来看，并没令所有人满意，Mixer 的 Check 缓存机制究竟能多大程度缓解 Policy 的性能压力依然需要观察。

值得一提的是，我们发现的不少 bug 同时也在被社区其他开发者发现并逐一解决。令我们开心的是，UAEK 团队不是信息孤岛，我们能感受到 Istio 官方社区正在努力高速迭代，始终在致力于解决广大开发者关心的种种问题，我们提交的 issue 能在数小时内被响应，这些，都让我们坚信，Istio 是一个有潜力的项目，会向 Kubernetes 一样走向成功。

从 UAEK 接入用户的经验来看，用户需要正确地使用好 Istio 离不开前期深入的 Istio 文档学习。UAEK 后续需致力于要简化这一过程，让用户能傻瓜化、界面化、随心所欲地定制自己的路由规则成为我们下一个愿景。

UAEK 团队始终致力于改革 UCloud 内部研发流程，让研发提升效率，让运维不再苦恼，让所有人开心工作。除了继续完善 ServiceMesh 功能，下半年 UAEK 还会开放更多的地域和可用区，提供功能更丰富的控制台，发布自动化的代码管理打包持续集成 (CI/CD) 特性等等，敬请期待！

作者介绍

陈绥，UCloud 资深研发工程师，先后负责监控系统、Serverless 产品、PaaS 平台 ServiceMesh 等开发，有丰富的分布式系统开发经验。



全球软件开发大会

▶ 聚焦

- 互联网高可用架构
- 国际化互联网业务架构
- 工程师个人成长与技术领导力
- 架构设计
- 后移动互联网时代的技术思考与实践
- 大规模基础设施DevOps探索
- 硅谷人工智能
- 人工智能与业务实践
- Java生态与创新
- 大数据系统架构
- 区块链技术与应用
- 前端新趋势
- 深度学习技术与应用
- 微服务架构 & Serverless
- 产品经理必修之用户细分与产品定位

▶ 实践

Facebook / 硅谷公司的互联网计算性能优化经验谈

LinkedIn / 推荐系统：提升用户增长与参与的利器

Uber / 核心Trip Flow容量管理

Confluent /Apache Kafka / 从0.8到2.0：那些年我们踩过的坑

快手 / 如何快速打造高稳定千亿级别对象存储平台

微软 / 集成AI开发平台实践

会议：2018年10月18-20日

培训：2018年10月21-22日

地址：上海·宝华万豪酒店

8折报名中，立减1360

团购享更多优惠，

截至2018年8月19日

大咖助阵



专题：硅谷人工智能
夏磊 / LinkedIn高级工程师，
湾区同学技术沙龙Board Member



专题：Java生态与创新
张建锋 / 永源中间件 共同创始人



专题：互联网高可用架构
吴其敏 / 平安银行
零售网络金融事业部首席架构师



专题：研发效率提升
徐毅 / 华为 技术专家



专题：产品经理必修之用户细分与产品定位
袁店明 / Dell EMC
敏捷与精益创业咨询师

.....

分享嘉宾



Julien Viet
Red Hat
首席软件工程师



庄振运
Facebook
计算机性能高级工程师



邹欣
微软亚洲研究院
首席研发经理



李欣 (Bruce Li)
Paypal
PayPal Risk Infra
Director level architect



Hien Luu
LinkedIn
工程经理



孙彦
Pinterest
视觉搜索团队高级工程师



施磊
Airbnb
技术经理



Jonathan Giles
Microsoft
Senior Cloud Developer Advocate

.....



100+技术专家的实践分享

联系我们：

热线：010-84782011 微信：qcon-0410

智能时代的新运维

CNUTCon

全球运维技术大会

▶ 聚焦12大专题

- AIOps实践与探索
- 自动化运维平台实践
- 监控与分析 (APM)
- 日志处理
- 大规模系统的性能优化
- 智能+时代下的运维破局
- 数据库运维
- CI/CD实践
- 微服务架构与实践
- Kubernetes实战
- 运维新技术
- SRE实践与思考

会议：2018年11月16-17日

培训：2018年11月18-19日

地址：上海 光大会展中心大酒店



8折报名

团购享更多优



联席主席



刘国华

阿里巴巴 研究员



吴其敏

平安银行

零售网络金融事业部首席架构师



涂彦

腾讯 游戏运维总监



李涛

百度 工程效率部负责人

百度平台化委员会秘书长

▶ 部分演讲嘉宾



陈云

百度 智能云事业部
资深研发工程师



不畏

阿里云
视频云运维专家



刘伟

腾讯
技术运营部高级工程师



夏舰波

京东商城
技术架构部资深架构师



顾宇

ThoughtWorks
高级咨询师



谭宇 (茂七)

阿里巴巴
高级技术专家



闫鹏

阿里云
ARMS技术负责人



刘林

搜狗
资深软件工程师



李浩

eBay
主管工程师

中，立减720元

惠，截止2018年10月21日



联系我们：

售票咨询(电话): 13269078023

售票咨询(邮箱): piaowu@geekbang.com



▶ 聚焦

- 高性能业务架构
- 数据基础平台构建技术
- 短视频架构和算法
- 数据工程 & 大数据智能处理
- 区块链技术实践
- 智能高效运维
- 微服务治理
- 数据驱动产品和用户增长
- 金融技术框架
- 大前端技术
- 信息安全与隐私保护
- CTO技术选型思维

▶ 实践

[菜鸟网络 / 菜鸟仓储物流平台架构演进及技术挑战](#)

[Pinterest / Build a dynamic and responsive Pinterest on AWS: a system engineer's perspective](#)

[京东 / 分布式BaaS的设计与实践](#)

[Netflix / Netflix API Gateway that handles 2M requests per second](#)

[LinkedIn / 大规模机器学习在LinkedIn预测模型中的应用](#)

[满帮集团 / 货车帮云原生平台架构设计思路和实践](#)

[阿里巴巴 / 深度学习在智慧餐厅中的应用](#)

会议：2018年12月07-08日

培训：2018年12月09-10日

地址：北京·国际会议中心

7折报名

团购享更多优

▶ 出品人



许家滔

腾讯 微信架构部后台总监



何径舟

百度 自然语言处理部高级经理
技术专家



张磊

阿里巴巴集团

高级技术专家 & 技术顾问



蒋佳涛

京东 供应链研发部技术负责人

▶ 分享嘉宾



Susheel Aroskar

Netflix

Software Engineer



李刚

百度 可用性工程
技术负责人



杨巍威

Hortonworks
YARN/Staff Engineer



长纪

阿里巴巴

高级技术专家



刘波

Pinterest
Engineering Manager



郭平（坤宇）

阿里巴巴

高级技术专家



张雷

LinkedIn

Engineering Manager



刘春伟

京东

区块链技术专家



钟勇

菜鸟网络

高级技术专家

中，立减2040元

优惠，截止2018年9月9日



联系我们：

电话：17326843116 (灰灰)

微信：aschina666