促进软件开发领域知识与创新的传播

架构师
ARCHITECT

热点 | Hot

Facebook开源跨平台
前端布局引擎Yoga

推荐文章 | Article

深入浅出Paxos算法协议
Redis client/server交互流程

观点 | Opinion

我们为什么选择Vue.js
而不是React

Geekbang 极客邦科技 | InfoQ

2017年1月

卷首语

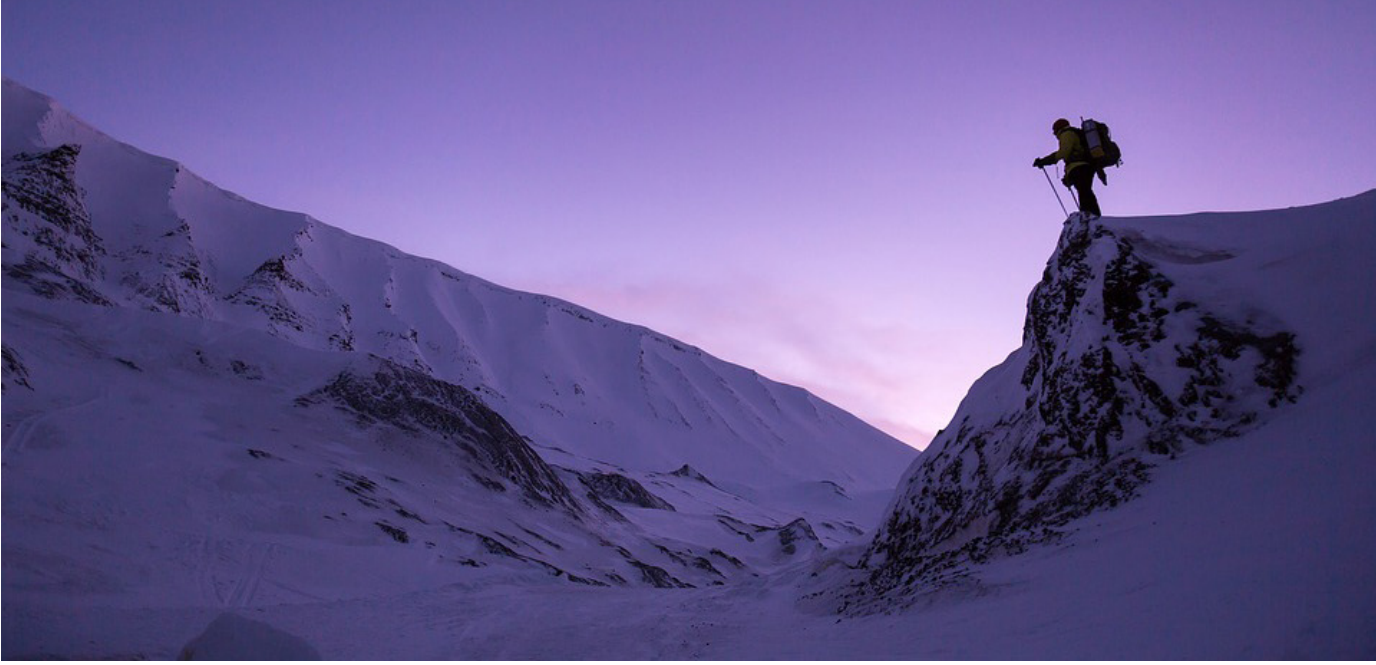
既然必须穿越地狱，那就走下去吧

霍泰稳 极客邦科技创始人兼 CEO

在每月我们给新员工准备的“CEO 私房话”时，总有同学会问到，“Kevin,作为一个公司的老板,每天那么多的事情,面临那么多的烦心事儿,你是如何一直保持高昂的状态的？”。刚开始被问到这个问题的时候，还有些懵，不知道如何回答，因为一直以来是这样啊，兵来将挡水来土掩，有问题就解决，没有为什么啊。被问的多了，也会经常去思考一下背后的原因。

直到最近看特斯拉创始人的传记《硅谷钢铁侠：埃隆·马斯克的冒险人生》，才有些恍然大悟的感觉，原来很多做事情的人是被一种称之为“使命感”的东西在推动的。比如对于马斯克，作者万斯评论他是个肩负使命的人，而且始终如一。他每天想的就是如何改变人类的命运，想着怎么能够把人类变成跨星球的物种，这样当地球毁灭时，还有其他的星球可以安住。所以，他在将 Paypal 卖给 eBay 后，开始做太空发射公司 SpaceX，做电动汽车特斯拉，做清洁能源太阳城。现在又开始在太空中打造高速卫星网络，按照他的话说，“全球通信网络对火星而言是很重要的，我认为必须做这件事，而且我没发现其他人在做。”

对于我和我的团队来说，过去十年间，我们坚持做 InfoQ，团队从 0



到现在 100 人，服务的人群也从 0 到上千万。其背后的使命虽然没有马斯克那么伟大，但确实是一直想把全球优质的学习资源整合起来，帮助我们的技术人和企业成长。因为这个使命，才在 InfoQ 创建一年后，制作了现在大家喜欢的《架构师》杂志，第二年引入了现在被公认为 No.1 的技术会议 QCon 全球软件开发大会，随后持续创建了针对架构师、移动开发人群、运维人群等垂直社区的各类高端会议等。

直到 2016 年，InfoQ 中国团队正式使用新的品牌“极客邦科技”，完成对 InfoQ 大中华区业务的收购，并推出了针对技术人的教育平台斯达克学院（StuQ）和针对高端技术领导者的社交网络 EGO，才基本构建了一个技术人学习和成长的生态圈。

做技术社区不是一件容易的事情，过去十年间，多少社区起来，又有多少社区倒下。有些甚至在还没有广为人知的时候，就在创始人的一声叹息中关门大吉。其原因还是因为技术人群是个小众人群，看上去薪资很高，有些高大上，但是对于大多数投资人来说，它体量太小，不能在短时间让自己的资金增值。另外一个原因是技术社区的门槛比较高，要求从业人员的综合素质要高，既要对技术有些了解，又熟知编辑知识，沟通协作能力

还要好。但是，要知道具备这样素质的人都是每个企业争夺的对象，很多大公司动辄以双倍甚至三倍的价格吸引技术社区的从业者。

但也有好消息，那就是技术被越来越重视，阿里巴巴创始人马云在去年世界互联网大会上就提到说，“未来 30 年一定不会只是‘互联网公司’的天下，未来 30 年是‘用好互联网技术’的公司，是‘用好互联网技术’的国家的天下，是‘用好互联网技术’的年轻人的天下。”。从当前云计算技术、人工智能技术、大数据技术的发展也大概能看出端倪，如果现在一个快速发展的企业中，没有研发团队作支撑，那真是处处捉襟见肘；如果没有数据做支撑，不能及时了解到用户的个性化需求并提供相应的服务，可能很快在市场的竞争中就失去优势。

所以，有些事情还是要做，而且要坚持去做，只要你认为有价值，自己也乐在其中。引用英国首相丘吉尔的一句话：“既然必须穿越地狱，那就走下去吧。”《架构师》新年第一期，用这句话和万千技术人共勉，与各位技术社区从业者共勉，祝大家新年愉快，在自己认为正确的路上坚持走下去。

QCon
全球软件开发大会

主办方 Geekbang. 极客邦科技 InfoQ

海量技术干货汇集
与大咖讲师 0 距离

《Maglev: A Fast and Reliable Software Network Load Balancer》

Cheng Yi

Google 高级工程师，网络负载均衡技术 Maglev 主要作者

《深入理解 Apache Beam》

Amit Sela

PayPal 架构师，Apache Beam 贡献者，PPMC 成员

《Mesos，数据中心操作系统的核心》

俞捷

Mesosphere 架构师，Apache Mesos 贡献者、PMC 成员

《聊聊开发工具的云端化》

赵扶摇

Lambda Lab 联合创始人，前 Google 工程师

《走进音视频测试体系》

罗必达

腾讯音视频实验室质量平台组组长

《趣店（前趣分期）风控业务那些事》

黄浩天

趣店集团（前趣分期）架构师 & 风控技术负责人

《人人车供应链系统技术架构演进》

路胜华

人人车业务平台部架构师 & 供应链技术负责人

《大数据与流处理》

王峰（莫问）

阿里搜索事业部资深技术专家，数据基础设施团队负责人

《技术管理的思考和实践——技术团队如何边打仗边成长》

姜华阳

美团点评销售和运营系统技术负责人

《精益创新组合决策——更科学地决策你的投资》

姚安峰

ThoughtWorks 国内咨询首席顾问

《智能运维里的时间序列：预测、异常检测和根源分析》

赵宇辰

AppDynamics 首席数据科学家

《高速发展业务的架构应对实践》

陈霖

百度外卖基础架构部资深架构师

《OCTO：千亿规模下的服务治理挑战与实践》

张熙

美团点评基础架构中心高级技术专家

《百度外卖从 IDC 到云端服务迁移历程》

赵晓燕

百度外卖研发中心运维部运维经理

《产品与运营分离的产品架构——网易教育的实践》

孙志岗

网易教育事业部战略总监

《菜鸟末端业务技术架构治理实践》

章天锋（大通）

阿里巴巴资深技术专家

《腾讯 GaiaStack 容器技术深度探索》

洪志国

腾讯高级工程师

《基于 Mesos 搭建 PaaS 平台你可能需要修的路》

杨成伟

爱奇艺助理研究员

大会官网：www.qconbeijing.com
议题提交：qcon@cn.infoq.com
扫码关注大会官网，获取更多大会信息

购票咨询请联系售票经理Hanna
联系电话：010-64738142
电子邮箱：hanna@infoq.com

8折 优惠报名中，截至2017年3月12日
团购享受更多优惠

[北京站]
2017年4月16-18日
北京·国家会议中心

CONTENTS / 目录

架构师

2017 年 1 月刊

本期主编　朱昊冰

流程编辑　丁晓昀

发行人　霍泰稳

提供反馈　feedback@cn.infoq.com

商务合作　sales@cn.infoq.com

内容合作　editors@cn.infoq.com

Facebook 开源跨平台前端布局引擎 Yoga

作者 薛命灯

随着这几年前端技术的崛起，作为前端 UI 骨架的布局系统也在其中占据了越来越重要的位置。不管是在移动端、桌面端还是 Web 端，特别是不同设备的屏幕大小和分辨率千变万化，如何构建良好的布局系统以便应付这些变化已经变得越来越重要。

目前，各个平台都有自己的一套解决方案。iOS 平台有自动布局系统，Android 有容器布局系统，而 Web 端有基于 CSS 的布局系统。多种布局系统共存所带来的弊端是很明显的，平台间的共享变得很困难，而每个平台都需要专人来开发维护，增加了开发成本。

Facebook 在这个问题上没有少下功夫。首先，Facebook 在 React Native 里引入了一种跨平台的基于 CSS 的布局系统，它实现了 Flexbox 规范。基于这个布局系统，不同团队终于可以走到一起，一起解决缺陷，改进性能，让这个系统更加地贴合 Flexbox 规范。

随着这个系统的不断完善，Facebook 决定对它进行重启发布，并取名 Yoga。虽然目前还不知道为什么会给它取名 Yoga，但从字面理解——瑜伽——我们很自然地联想起柔韧、舒展、变化等名词，这个跟布局系

统的跨平台特性似乎不谋而合。借助 Yoga，开发人员不仅可以在 React Native 里，还能在各个平台上快速地构建 UI 布局。

Yoga 是基于 C 实现的。之所以选择 C，首先当然是从性能方面考虑的。基于 C 实现的 Yoga 比之前 Java 实现在性能上提升了 33%。其次，使用 C 实现可以更容易地跟其它平台集成。到目前为止，Yoga 已经有以下几个平台的绑定：Java（Android）、Objective-C（UIKit）、C#（.NET）。而且已经有很多项目在使用 Yoga，比如 React Native、Components for Android、Oculus，等等。

不同于其它的一些布局框架，比如 bootstrap 的栅格系统或 Masonry，它们要么不够强大，要么不支持跨平台。Yoga 遵循了 Flexbox 规范，同时又将布局元素抽象成 Node，为各个不同平台暴露出一组标准的接口，这样不同的平台只需实现这些接口就可以了。

当然，Facebook 不会就此止步。作为一款跨平台的布局引擎，自然需要各个平台的开发人员一起努力来促进它的发展，所以 Facebook 把 Yoga 开源了。目前微软已经成为 Yoga 的贡献者之一，他们不仅修复缺陷，还为 Yoga 带来新的特性。

除了完全遵循 Flexbox 规范，Facebook 还计划在未来为 Yoga 加入更多特性，这些特性将超出 Flexbox 的范畴。

Yoga 的源码托管在 GitHub 上，有兴趣开发人员可以在上面进行反馈。

腾讯大数据宣布开源第三代高性能计算平台 Angel：支持十亿维度

作者 杜小芳

12 月 18 日，深圳 - 腾讯大数据宣布推出面向机器学习的第三代高性能计算平台——Angel，并预计于 2017 年一季度开放其源代码，鼓励业界工程师、学者和技术人员大规模学习使用，激发机器学习领域的更多创新应用与良好生态发展。

InfoQ 采访了腾讯大数据负责人蒋杰，本文根据采访稿件以及姚星和蒋杰在腾讯大数据技术峰会暨 KDD China 技术峰会上的演讲内容整理而来。

研发背景

腾讯公司是一家消息平台 + 数字内容的公司，本质上也是一家大数据公司，每天产生数千亿的收发消息，超过 10 亿的分享图片，高峰期间百亿的收发红包。每天产生的看新闻、听音乐、看视频的流量峰值高达数十 T。这么大的数据量，处理和使用上，首先业务上存在三大痛点：

- 第一，需要具备T/P级的数据处理能力，几十亿、百亿级的数据量，基本上30分钟就要能算出来。

- 第二，成本需低，可以使用很普通的PC Server，就能达到以前小型机一样的效果；
- 第三，容灾方面，原来只要有机器宕机，业务的数据肯定就有影响，各种报表、数据查询，都会受到影响。

其次是需要融合所有产品平台的数据的能力。“以前的各产品的数据都是分散在各自的 DB 里面的，是一个个数据孤岛，现在，需要以用户为中心，建成了十亿用户量级、每个用户万维特征的用户画像体系。以前的用户画像，只有十几个维度主要就是用户的一些基础属性，比如年龄、性别、地域等，构建一次要耗费很多天，数据都是按月更新”。

另外就是需要解决速度和效率方面的问题，以前的数据平台“数据是离线的，任务计算是离线的，实时性差”。

“所以，我们必须要建设一个能支持超大规模数据集的一套系统，能满足 billion 级别的维度的数据训练，而且，这个系统必须能满足我们现网应用需求的一个工业级的系统。它能解决 big data，以及 big model 的需求，它既能做数据并行，也能做模型并行。”

经过 7 年的不断发展，历经了三代大数据平台：第一代 TDW（腾讯分布式数据仓库），到基于 Spark 融合 Storm 的第二代实时计算架构，到现在形成了第三代的平台，核心为 Angel 的高性能计算平台。

Angel 项目在 2014 年开始准备，15 年初正式启动，刚启动只有 4 个人，后来逐步壮大。项目跟北京大学和香港科技大学合作，一共有 6 个博士生加入到腾讯大数据开发团队。目前在系统、算法、配套生态等方面开发的人员，测试和运维，以及产品策划及运维，团队超过 30 人。

Angel 平台是使用 Java 和 Scala 混合开发的机器学习框架，用户可以像用 Spark，MapReduce 一样，用它来完成机器学习的模型训练。

Angel 采用参数服务器架构，支持十亿级别维度的模型训练。采用了多种业界最新技术和腾讯自主研发技术，如 SSP（Stale synchronous Parallel）、异步分布式 SGD、多线程参数共享模式 HogWild、网络带宽流量调度算法、计算和网络请求流水化、参数更新索引和训练数据预处理方案等。

这些技术使 Angel 性能大幅提高，达到常见开源系统 Spark 的数倍到数十倍，能在千万到十亿级的特征维度条件下运行。

自今年初在腾讯内部上线以来，Angel 已应用于腾讯视频、腾讯社交广告及用户画像挖掘等精准推荐业务。未来还将不断拓展应用场景，目标是支持腾讯等企业级大规模机器学习任务。

腾讯为何要选择自研？

首先需要一个满足十亿级维度的工业级的机器学习平台，蒋杰表示当时有两种思路：一个是基于第二代平台的基础上做演进，解决大规模参数交换的问题。另外一个，就是新建设一个高性能的计算框架。

当时有研究业内比较流行的几个产品：GraphLab，主要做图模型，容错差；Google 的 Distbelief，还没开源；还有 CMU Eric Xing 的 Petuum，当时很火，不过它更多是一个实验室的产品，易用性和稳定性达不到要求。

“其实在第二代，我们已经尝试自研，我们消息中间件，不论是高性能的，还是高可靠的版本，都是我们自研的。他们经历了腾讯亿万流量的考验，这也给了我们在自研方面很大的信心”。

第三代高性能计算平台

“同时，我们第三代的平台，还需要支持 GPU 深度学习，支持文本、

语音、图像等非结构化的数据”。

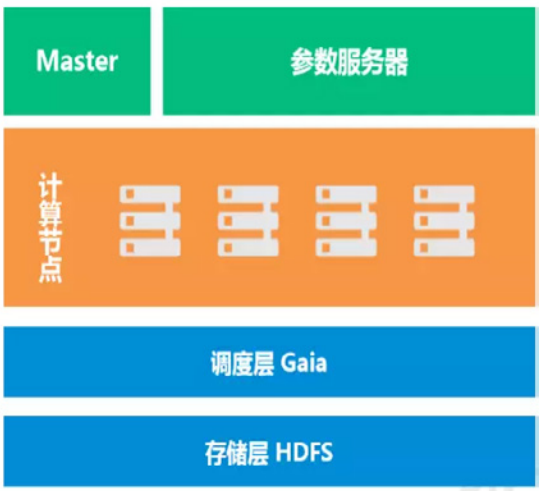
Angel 的整体架构

Angel 是基于参数服务器的一个架构，整体架构上参考了谷歌的 DistBelief。Angel 在运算中支持 BSP、SSP、ASP 三种计算模型，其中 SSP 是由卡耐基梅隆大学 EricXing 在 Petuum 项目中验证的计算模型，能在机器学习的这种特定运算场景下提升缩短收敛时间。Angel 支持数据并行及模型并行。

在网络上有原创的尝试，使用了港科大杨强教授的团队做的诸葛弩来做网络调度，

ParameterServer 优先服务较慢的 Worker，当模型较大时，能明显降低等待时间，任务总体耗时下降 5%~15%。

另外，Angel 整体是跑在 Gaia



Angel架构图

（Yarn）平台上面的。

主要的模块有 3 个：

Master：主控节点，负责资源申请和分配，以及任务的管理。

ParameterServer：包含多个节点，可对参数进行横向扩展，解决参数汇总更新的单点瓶颈，支持 BSP,SSP,ASP 等多种计算模型，随着一个任务的启动而生成，任务结束而销毁，负责在该任务训练过程中的参数的更新和存储。

WorkerGroup：一个 WG 包含多个 Worker,WG 内部实现模型并行，WG 之间实现数据并行，独立进程运行于 Yarn 的 Container 中。

Angel 已经支持了 20 多种不同算法，包括 SGD、ADMM 优化算法等，我们也开放比较简易的编程接口，用户也可以比较方便的编写自定义的算法，实现高效的 ps 模型。并提供了高效的向量及矩阵运算库（稀疏 / 稠密），方便了用户自由选择数据、参数的表达形式。在优化算法方面，Angel 已实现了 SGD、ADMM，并支持 Latent DirichletAllocation (LDA)、MatrixFactorization (MF)、LogisticRegression (LR) 、Support Vector Machine(SVM) 等。

Angel 的优势包括几点：

- 能高效支持超大规模（十亿）维度的数据训练；
- 同样数据量下，比Spark、Petuum等其他的计算平台性能更好；
- 有丰富的算法库及计算函数库，友好的编程接口，让用户像使用MR、Spark一样编程；
- 丰富的配套生态，既有一体化的运营及开发门户，又能支持深度学习、图计算等等其他类型的机器学习框架，让用户在一个平台能开发多种类型的应用。

Angel 做过哪些优化?

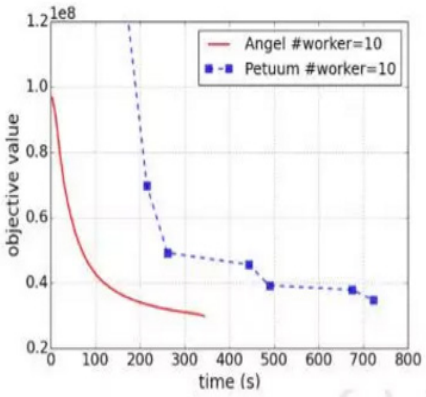
Angel 是基于参数服务器的一个架构，与其他平台相比，在性能上很多优化。首先，我们能支持 BSP、SSP、ASP 三种不同计算和参数更新模式，其次，我们支持模型并行，参数模型可以比较灵活进行切分。第三，我们有个服务补偿的机制，参数服务器优先服务较慢的节点，根据我们的测试结果，当模型较大时，能明显降低等待时间，任务总体耗时下降 5%~15%。最后，我们在参数更新的性能方面，做了很多优化，比如对稀疏矩阵的 0 参数以及已收敛参数进行过滤，我们根据参数的不同数值类型进行不同算法的压缩，最大限度减少网络负载，我们还优化了参与获取与计算的顺序，边获取参数变计算，这样就能节省 20-40% 的计算时间。

我们除了在性能方面进行深入的优化，在系统易用性上我们也做了很多改进。第一，我们提供很丰富的机器学习算法库，以及数学运算算法库；第二，我们提供很友好的高度抽象的编程接口，能跟 Spark、MR 对接，开发人员能像用 MR、Spark 一样编程；第三，我们提供了一体化的拖拽式的开发及运营门户，用户不需要编程或只需要很少的开发量就能完成算法训练；第四，我们内置数据切分、数据计算和模型划分的自动方案及参数自适应配置等功能，并屏蔽底层系统细节，用户可以很方便进行数据预处理；最后一点，Angel 还能支持多种高纬度机器学习的场景，比如支持 Spark 的 MLLib，支持 Graph 图计算、还支持深度学习如 Torch 和 TensorFlow 等业界主流的机器学习框架，提供计算加速。

Angel 的性能项目测试结果

同等数据量下的性能测试。Angel 跟其他平台相比，比如 Petuum，和 spark 等，在同等量级下的测试结果，Angel 的性能要优于其他平台。比

如用 Netflix 的数据跑的 SGD 算法，结果可以看下图中的对比。



- 比较算法：矩阵分解
- 数据集：Netflix
- 优化算法：SGD
- Worker数目：10
- Worker内存：10G

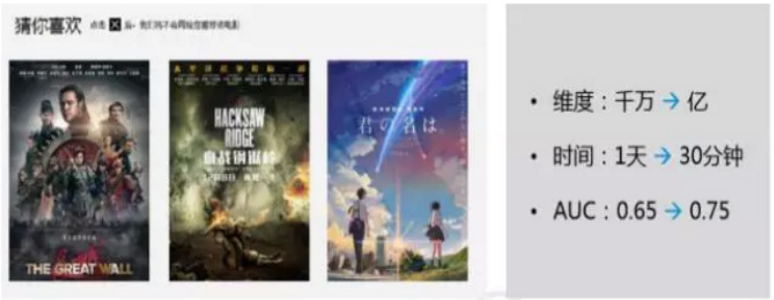
超大规模数据的训练测试。目前 Angel 支持了很多腾讯内部的现网业务。举两个例子，比如，在构建用户画像方面，以前都是基于 Hadoop 和 Spark 来做，跑一次模型要 1 天甚至几天，话题只有 1k；而在 Angel 上，20 多亿文档、几百万个词，3000 亿的 token，1 个小时就跑完了。以前 Spark 能跑的，现在 Angel 快几十倍；以前 Spark 跑不了的，Angel 也能轻松跑出来。

- 同等量级，Angel比Spark提升**12倍**，并能支持**超大规模**的场景
- 模型更新：天→小时
- 数据集：几百G→几T（20亿文档、500万词，3000亿token）

大规模数据集的训练能力。例如腾讯视频的点击预测，同等数据量下，Angel 的性能是 Spark 的 44 倍以上。用了 Angel 以后，维度从千万扩展到亿，训练时间从天缩短到半小时，而准确度也有很大的提升。

为什么开源？

Angel 不仅仅是一个只做并行计算的平台，它更是一个生态，我们



视频点击预测（LR）

围绕 Angel，建立了一个小生态圈，它支持 Spark 之上的 MLLib，支持上亿的维度的训练；我们也支持更复杂的图计算模型；同时支持 Caffe、TensorFlow、Torch 等深度学习框架，实现这些框架的多机多卡的应用场景。

Angel的生态圈

腾讯大数据平台来自开源的社区，受益于开源的社区中，所以我们自然而然地希望回馈社区。开源，让开放者和开发者都能受益，创造一个共建共赢的生态圈。在这里，开发者能节约学习和操作的时间，提升开发效率，去花时间想更好的创意，而开放者能受益于社区的力量，更快完善项目，构建一个更好的生态圈。

我们目前希望能丰富 Angel 配套生态圈，进一步降低用户使用门槛，

促进更多开发人员，包括学校与企业，参与共建 Angel 开源社区。而通过推动 Angel 的发展，最终能让更多用户能快速、轻松地建立有大规模计算能力的平台。

我们一直都向社区做贡献，开放了很多源代码，培养了几个项目的 committer，这种开放的脚步不会停止。

小结

腾讯公司通过 18 年的发展今天已经成为了世界级的互联网公司。“在技术上，我们过去更加关注的是工程技术，也就是海量性能处理能力、海量数据存储能力、工程架构分布容灾能力。未来腾讯必将发展成为一家引领科技的互联网公司，我们将在大数据、核心算法等技术领域上进行积极的投入和布局，和合作伙伴共同推动互联网产业的发展。”

受访嘉宾

蒋杰，国内知名大数据专家，CCF 大数据专家委员会委员，ACM 数据挖掘中国分会委员，拥有超过 12 年以上从业经验，在数据库、分布式架构、高性能计算、机器学习等方面积累了丰富经验。目前是腾讯首席数据专家、数据平台部总经理，全面负责腾讯数据业务。

前端组件化开发方案及其在 React Native 中的运用

作者 刘先宁

随着 SPA，前后端分离的技术架构在业界越来越流行，前端（注：本文中的前端泛指所有的用户可接触的界面，包括桌面，移动端）需要管理的内容，承担的职责也越来越多。再加上移动互联网的火爆，及其带动的 Mobile First 风潮，各大公司也开始在前端投入更多的资源。这一切，使得业界对前端开发方案的思考上多了很多，以 React 框架为代表推动的组件化开发方案就是目前业界比较认可的方案，本文将和大家一起探讨一下组件化开发方案能给我们带来什么，以及如何在 React Native 项目的运用组件化开发方案。

一、为什么要采用组件化开发方案?

在讲怎么做之前，需要先看看为什么前端要采用组件化开发方案，作为一名程序员和咨询师，我清楚地知道凡是抛开问题谈方案都是耍流氓。那么在面对随着业务规模的增加，更多的业务功能推向前端，以及随之而来的开发团队扩张时，前端开发会遇到些什么样的问题呢？

1. 前端开发面临的问题

1. 资源冗余：页面变得越来越多，页面的交互变得越来越复杂。在这种情况下，有些团队成员会根据功能写自己的CSS、JS，这会产生大量的新的CSS或JS文件，而这些文件中可能出现大量的重复逻辑；有些团队成员则会重用别人的逻辑，但是由于逻辑拆分的粒度差异，可能会为了依赖某个JS中的一个函数，需要加载整个模块，或者为了使用某个CSS中的部分样式依赖整个CSS文件，这导致了大量的资源冗余。
2. 依赖关系不直观：当修改一个JS函数，或者某个CSS属性时，很多时候只能靠人力全局搜索来判断影响范围，这种做法不但慢，而且很容易出错。
3. 项目的灵活性和可维护性差：因为项目中的交叉依赖太多，当出现技术方案变化时，无法做到渐进式的、有节奏地替换掉老的代码，只能一次性替换掉所有老代码，这极大地提升了技术方案升级的成本和风险。
4. 新人进组上手难度大：新人进入项目后，需要了解整个项目的背景、技术栈等，才能或者说才敢开始工作。这在小项目中也许不是问题，但是在大型项目中，尤其是人员流动比较频繁的项目，则会对项目进度产生非常大的影响。
5. 团队协同度不高：用户流程上页面间的依赖（比方说一个页面强依赖前一个页面的工作结果），以及技术方案上的一些相互依赖（比方说某个文件只能由某个团队修改）会导致无法发挥一个团队的全部效能，部分成员会出现等待空窗期，浪费团队效率。
6. 测试难度大：整个项目中的逻辑拆分不清晰，过多且杂乱的相互

依赖都显著拉升了自动化测试的难度。

7. 沟通反馈慢：业务的要求，UX的设计都需要等到开发人员写完代码，整个项目编译部署后才能看到实际的效果，这个反馈周期太长，并且未来的任何一个小修改又需要重复这一整个流程。

2.组件化开发带来的好处

组件化开发的核心是“业务的归业务，组件的归组件”。即组件是一个个独立存在的模块，它需要具备如下的特征：

- 职责单一而清晰：开发人员可以很容易了解该组件提供的能力。
- 资源高内聚： 组件资源内部高内聚，组件资源完全由自身加载控制。
- 作用域独立：内部结构密封，不与全局或其他组件产生影响。
- 接口规范化：组件接口有统一规范。
- 可相互组合：组装整合成复杂组件，高阶组件等。

独立清晰的生命周期管理：组件的加载、渲染、更新必须有清晰的、可控的路径。

而业务就是通过组合这一堆组件完成User Journey。下一节中，会详细描述采用组件化开发方案的团队是如何运作的。

在项目中分清楚组件和业务的关系，把系统的构建架构在组件化思想上可以：

1. 降低整个系统的耦合度：在保持接口不变的情况下，我们可以把当前组件替换成不同的组件实现业务功能升级，比如把一个搜索框，换成一个日历组件。
2. 提高可维护性：由于每个组件的职责单一，在系统中更容易被复用，所以对某个职责的修改只需要修改一处，就可获得系统的整

体升级。独立的，小的组件代码的更易理解，维护起来也更容易。

3. 降低上手难度：新成员只需要理解接口和职责即可开发组件代码，在不断的开发过程中再进一步理解和学习项目知识。另外，由于代码的影响范围仅限于组件内部，对项目的风险控制也非常有帮助，不会因为一次修改导致雪崩效应，影响整个团队的工作。
4. 提升团队协同开发效率：通过对组件的拆分粒度控制来合理分配团队成员任务，让团队中每个人都能发挥所长，维护对应的组件，最大化团队开发效率。
5. 便于自动化测试：由于组件除了接口外，完全是自治王国，甚至概念上，可以把组件当成一个函数，输入对应着输出，这让自动化测试变得简单。
6. 更容易的自文档化：在组件之上，可以采用Living Style Guide的方式为项目的所有UI组件建立一个‘活’的文档，这个文档还可以成为业务，开发，UX之间的沟通桥梁。这是对‘代码即文档’的另一种诠释，巧妙的解决了程序员不爱写文档的问题。
7. 方便调试：由于整个系统是通过组件组合起来的，在出现问题的时候，可以用排除法直接移除组件，或者根据报错的组件快速定位问题。另外，Living Style Guide除了作为沟通工具，还可以作为调试工具，帮助开发者调试UI组件。

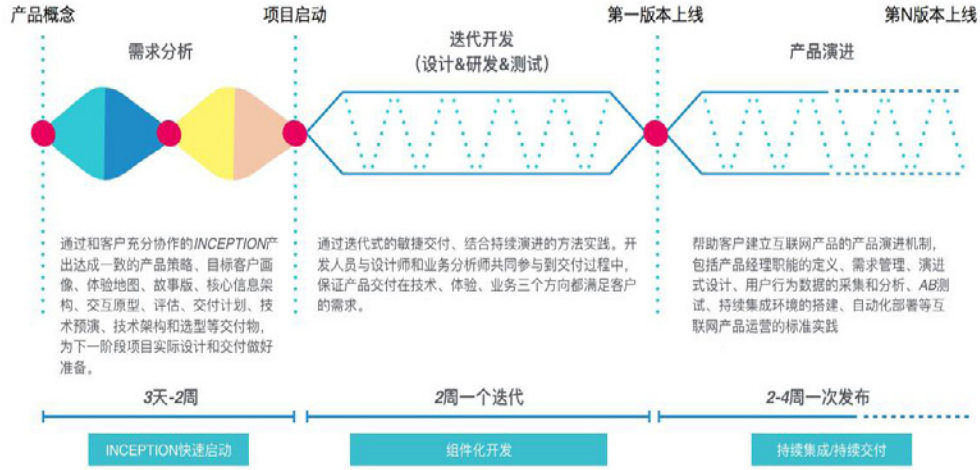
二、组件化开发方案下，团队如何运作?

前面大致讲了下组件化开发可以给项目带来的好处，接下来聊一聊采

用组件化开发方案的团队是应该如何运作？

在 ThoughtWorks，我们把一个项目的生命周期分为如下几个阶段：

组件化开发方案主要关注的是在迭代开发阶段的对团队效率的提升。它主要从以下几个方面提升了开发效率：



1. 以架构层的组件复用降低工作量

在大型应用的后端开发中，为了分工、复用和可维护性，在架构层面将应用抽象为多个相对独立的模块的思想和方法都已经非常成熟和深入人心了。但是在前端开发中，模块化的思想还是比较传统，开发者还是只有在需考虑复用时才会将某一部分做成组件，再加上当开发人员专注在不同界面开发上时，对于该界面上哪些部分可以重用缺乏关注，导致在多个界面上重复开发相同的 UI 功能，这不仅拉升了整个项目的工作量，还增加了项目后续的修改和维护成本。

在组件化开发方案下，团队在交付开始阶段就需要从架构层面对应用的 UI 进行模块化，团队会一起把需求分析阶段产生的原型中的每一个 UI 页面抽象为一颗组件树，UI 页面自己本身上也是一个组件。如下图。

通过上面的抽象之后，我们会发现大量的组件可以在多个 UI 界面上复用，而考虑到在前端项目中，构建各个 UI 界面占了 80% 以上的工作量，

这样的抽象显著降低了项目的工作量，同时对后续的修改和维护也会大有裨益。

在这样的架构模式下，团队的运作方式就需要相应的发生改变：

(1) 工程化方面的支持，从目录结构的划分上对开发人员进行组件化思维的强调，区分基础组件，业务组件，页面组件的位置，职责，以及相互之间的依赖关系。

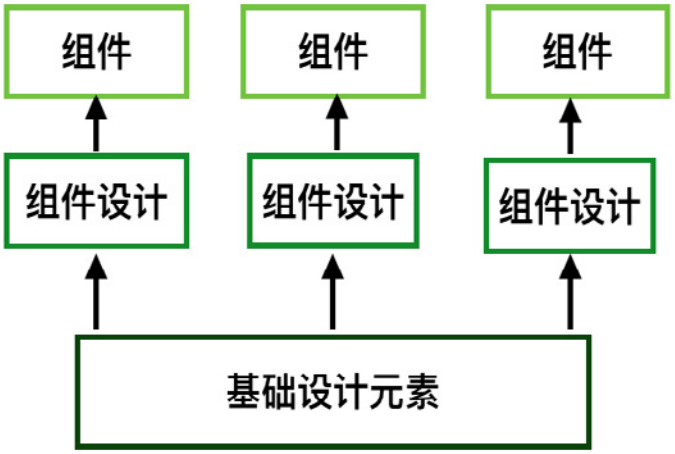
(2) 工作优先级的安排，在敏捷团队中，我们强调的是交付业务价值。而业务是由页面组件串联而成，在组件化的架构模式下，必然是先完成组件开发，再串联业务。所以在做迭代计划时，需要对团队开发组件的任务和串联业务的任务做一个清晰的优先级安排，以保证团队对业务价值的交付节奏。

2.以组件的规范性保障项目设计的统一性

在前端开发中，因为 CSS 的灵活性，对于相同的 UI 要求（比如：布局上的靠右边框 5 个像素），就可能有上十种的 CSS 写法，开发人员的背景，经历的不同，很可能会选择不同的实现方法；甚至还有一些不成熟的项目，存在需求方直接给一个 PDF 文件的用户流程图界面，不给 PSD 的情况，所有的设计元素需要开发人员从图片中抓取，这更是会使得项目的样式写的五花八门。因为同样的 UI 设计在项目中存在多种写法，会导致很多问题，第一就是设计上可能存在不一致的情况；第二是 UI 设计发生修改时，出现需要多种修改方案的成本，甚至出现漏改某个样式导致 bug 的问题。

在组件化开发方案下，项目设计的统一性被上拉到组件层，由组件的

统一性来保障。其实本来所有的业务 UI 设计就是组件为单位的，设计师不会说我要“黄色”，他们说得是我要“黄色的按钮……”。是开发者在实现过程中把 UI 设计下放到 CSS 样式上的，相比一个个，一组组的 CSS 属性，组件的整体性和可理解性都会更高。再加上组件的资源高内聚特性，在组件上对样式进行调整也会变得容易，其影响范围也更可控。



在组件化开发方案下，为了保证 UI 设计的一致性，团队的运作需要：

定义基础设计元素，包括色号、字体、字号等，由 UX 决定所有的基础设计元素。

所有具体的 UI 组件设计必须通过这些基础设计元素组合而成，如果当前的基础设计元素不能满足需求，则需要和 UX 一起讨论增加基础设计元素。

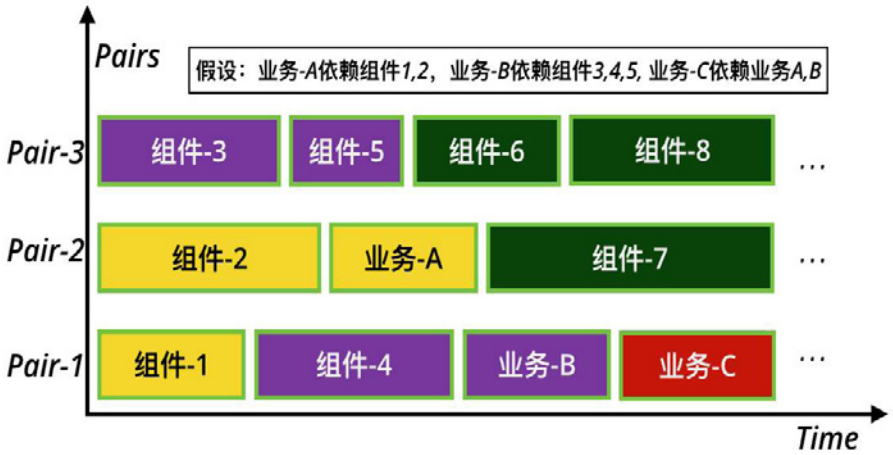
UI 组件的验收需要 UX 参与。

3. 以组件的独立性和自治性提升团队协同效率

在前端开发时，存在一个典型的场景就是某个功能界面，距离启动界面有多个层级，按照传统开发方式，需要按照页面一页一页的开发，当前一个页面开发未完成时，无法开始下一个页面的开发，导致团队工作的并发度不够。另外，在团队中，开发人员的能力各有所长，而页面依赖降低

了整个项目在任务安排上的灵活性，让我们无法按照团队成员的经验，强项来合理安排工作。这两项对团队协同度的影响最终会拉低团队的整体效率。

在组件化开发方案下，强调业务任务和组件任务的分离和协同。组件任务具有很强的独立性和自治性，即在接口定义清楚的情况下，完全可以抛开上下文进行开发。这类任务对外无任何依赖，再加上组件的职责单一性，其功能也很容易被开发者理解。所以在安排任务上，组件任务可以非常灵活。而业务任务只需关注自己依赖的组件是否已经完成，一旦完成就马上进入 Ready For Dev 状态，以最高优先级等待下一位开发人员选取。



在组件化开发方案下，为了提升团队协同效率，团队的运作需要：

1. 把业务任务和组件任务拆开，组件的归组件，业务的归业务。
2. 使用Jira，Mingle等团队管理工具管理好业务任务对组件任务的依赖，让团队可以容易地了解到每个业务价值的实现需要的完成的任务。
3. Tech Lead需要加深对团队每个成员的了解，清楚的知道他们各自的强项，作为安排任务时的参考。
4. 业务优先原则，一旦业务任务依赖的所有组件任务完成，业务任

务马上进入最高优先级，团队以交付业务价值为最高优先级。

5. 组件任务先于业务任务完成，未纳入业务流程前，团队需要Living Style Guide之类的工具帮助验收组件任务。

4. 以组件的Living Style Guide平台降低团队沟通成本

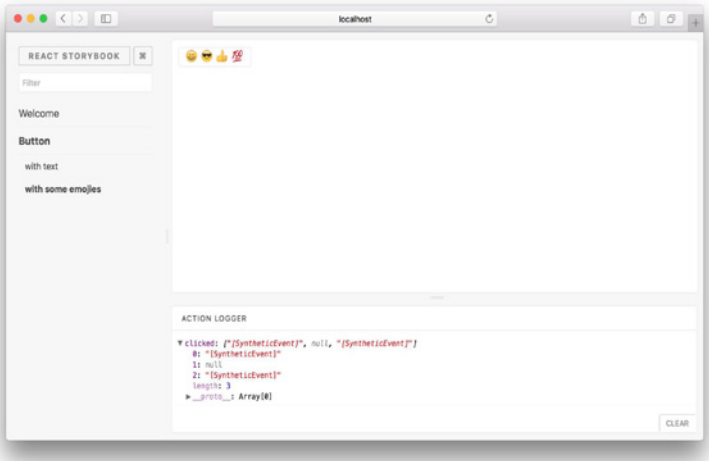
在前端开发时，经常存在这样的沟通场景：

- 开发人员和UX验证页面设计时，因为一些细微的差异对UI进行反复的小修改；
- 开发人员和业务人员验证界面流程时，因为一些特别的需求对UI进行反复的小修改；
- 开发人员想复用另一个组件，寻找该组件的开发人员了解该组件的设计和职责；
- 开发人员和QA一起验证某个公用组件改动对多个界面上的影响。

当这样的沟通出现在上一小节的提到的场景，即组件出现在距离启动界面有多个层级的界面时，按照传统开发方式，UX 和开发需要多次点击，有时甚至需要输入一些数据，最后才能到达想要的功能界面。没有或者无法搭建一个直观的平台满足这些需求，就会导致每一次的沟通改动就伴随着一次重复走的，很长的路径。使得团队的沟通成本激增，极大的降低了开发效率。

在组件化开发方案下， 因为组件的独立性，构建 Living Style Guide 平台变得非常简单，目前社区已经有了很多工具支持构建 Living Style Guide 平台（比如 getstorybook），开发人员把组件以 Demo 的形式添加到 Living Style Guide 平台就行了，然后所有与 UI 组件的相关的沟通都以该平台为中心进行，因为开发对组件的修改会马上体现在平台上，再加上平台对组件的组织形式让所有人都可以很直接的访问到任何需要的

组件，这样，UX 和业务人员有任何要求，开发人员都可以快速修改，共同在平台上验证，这种“所见即所得”的沟通方式节省去了大量的沟通成本。此外，该平台自带组件文档功能，团队成员可以从该平台上看到所有组件的 UI，接口，降低了人员变动导致的组件上下文知识缺失，同时也降低了开发者之间对于组件的沟通需求。



想要获得这些好处，团队的运作需要：

1. 项目初期就搭建好Living Style Guide平台。
2. 开发人员在完成组件之后必须添加Demo到平台，甚至根据该组件需要适应的场景，多添加几个Demo。这样一眼就可以看出不同场景下，该组件的样子。
3. UX，业务人员通过平台验收组件，甚至可以在平台通过修改组件 Props，探索性的测试在一些极端场景下组件的反应。

5. 对需求分析阶段的诉求和产品演进阶段的帮助

虽然需求分析阶段和产品演进阶段不是组件化开发关注的重点，但是组件化开发的实施效果却和这两个阶段有关系，组件化方案需要需求分析阶段能够给出清晰的 Domain 数据结构，基础设计元素和界面原型，它们是组件化开发的基础。而对于产品演进阶段，组件化开发提供的两个重要

特性则大大降低了产品演进的风险：

低耦合的架构，让开发者清楚的知道自己的修改影响范围，降低演进风险。开发团队只需要根据新需求完成新的组件，或者替换掉已有组件就可以完成产品演进。

Living Style Guide 的自文档能力，让你能够很容易的获得现有组件代码的信息，降低人员流动产生的上下文缺失对产品演进的风险。

三、组件化开发方案在 React Native 项目中的实施

前面已经详细讨论了为什么和如何做组件化开发方案，接下来，就以一个 React Native 项目为例，从代码级别看看组件化方案的实施。

1. 定义基础设计元素

在前面我们已经提到过，需求分析阶段需要产出基本的设计元素，在前端开发人员开始写代码之前需要把这部分基础设计元素添加到代码中。在 React Native 中，所有的 CSS 属性都被封装到了 JS 代码中，所以在 React Native 项目开发中，不再需要 LESS，SCSS 之类的动态样式语言，而且你可以使用 JS 语言的一切特性来帮助你组合样式，所以我们可以创建一个 theme.js 存放所有的基础设计元素，如果基础设计元素很多，也可以拆分位多个文件存放。

```
import { StyleSheet } from 'react-native';  module.exports =
StyleSheet.create({
  colors: {...},
  fonts: {...},
  layouts: {...},
  borders: {...},
  container: {...},
  });
```

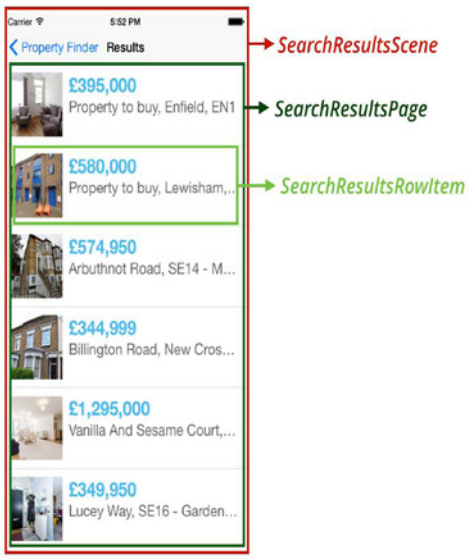
然后，在写具体 UI 组件的 styles，只需要引入该文件，按照 JS 的规

则复用这些样式属性即可。

2.拆分组件树之Component，Page，Scene

在实现业务流程前，需要对项目的原型UI进行分解和分类，在React Native项目中，我把UI组件分为了四种类型。

- Shared Component：基础组件，Button，Label之类的大部分其它组件都会用到的基础组件。
- Feature Component：业务组件，对应到某个业务流程的子组件，但其不对应路由，他们通过各种组合形成了Pag组件。
- Page：与路由对应的Container组件，主要功能就是组合子组件，所有Page组件最好名字都以Page结尾，便于区分。
- Scene：应用状态和UI之间的连接器，严格意义上它不算UI组件，主要作用就是把应用的状态和Page组件绑定上，所有的Scene组件以Scene后缀结尾。



Component和Page组件都是Pure Component，只接收props，然后展示UI，响应事件。Component的Props由Page组件传递给它，Page组件的Props则是由Scene组件绑定过去。下面我们就以如下的这个页面为例来看看这几类组件各自的职责范围：

（1）searchResultRowItem.js

```
export default function (rowData) {
  const {title, price_formatted, img_url, rowID, onPress} = rowData;
```

```
  const price = price_formatted.split(' ')[0];
  return (
    <TouchableHighlight
      onPress={() => onPress(rowID)}
      testID={'property-' + rowID}
      underlayColor='#dddddd'>
      <View>
        <View style={styles.rowContainer}>
            <Image style={styles.thumb} source={{ uri: img_url
}}/>
          <View style={styles.textContainer}>
            <Text style={styles.price}>{price}</Text>
                      <Text style={styles.title}
numberOfLines={1}>{title}</Text>
          </View>
        </View>
        <View style={styles.separator }/>
      </View>
    </TouchableHighlight>
  );}
```

（2）SearchResultsPage.js

```
import SearchResultRowItem from '../components/
searchResultRowItem';export default class SearchResultsPage
extends Component {

  constructor(props) {
    super(props);
    const dataSource = new ListView.DataSource({rowHasChanged:
(r1, r2) => r1.guid !== r2.guid});
    this.state = {
        dataSource: dataSource.cloneWithRows(this.props.
properties),
      onRowPress: this.props.rowPressed,
    };
```

```
  }

  renderRow(rowProps, sectionID, rowID) {
     return <SearchResultRowItem {...rowProps} rowID={rowID}
onPress={this.state.onRowPress} />;
  }

  render() {
    return (
      <ListView
        style={atomicStyles.container}
        dataSource={this.state.dataSource}
        renderRow={this.renderRow.bind(this)} />
    );
  }}
```

(3)SearchResultsScene.js

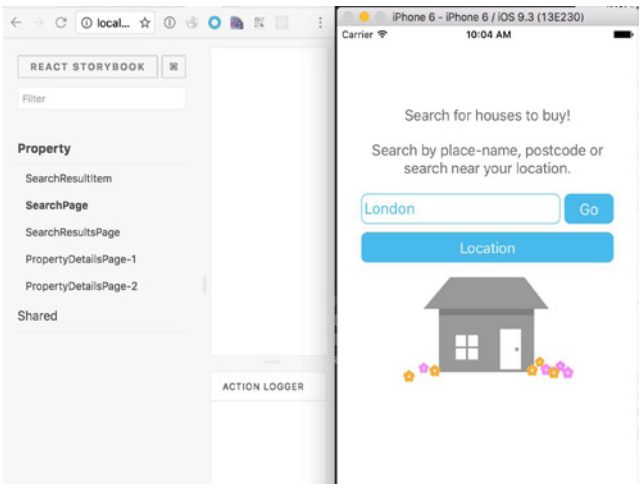
```
import SearchResults from '../components/
searchResultsPage';function mapStateToProps(state) {
  const {propertyReducer} = state;
  const {searchReducer:{properties}} = propertyReducer;
  return {
    properties,
  };}function mapDispatchToProps(dispatch) {
  return {
    rowPressed: (propertyIndex) => {
      dispatch(PropertyActions.selectProperty(propertyIndex));
      RouterActions.PropertyDetails();
    }
  };}module.exports = connect(
  mapStateToProps,
  mapDispatchToProps,)(SearchResults);
```

3.Living Style Guide

目前社区上，最好的支持 React Native 的 Living Style Guide 工

具是 getstorybook，关于如何使用 getstorybook 搭建 React Native 的 Living Style Guide 平台可以参见官方文档（https://github.com/kadirahq/react-native-storybook）或者我的博客（http://www.jianshu.com/p/36cbd8393288）。

搭建好 Living Style Guide 平台后，就可以看到如下的界面：



接下来的工作就是不断在往该平台添加 UI 组件的 Demo。向 storybook 中添加 Demo 非常简单，下面就是一个关于 SearchPage 的 Demo：

```
import React from 'react';import {storiesOf, action} from '@kadira/react-native-storybook';import SearchPage from '../../../../src/property/components/searchPage';storiesOf('Property', module)
  .add('SearchPage', () => (
      <SearchPage request={{place_name:"London"}} isLoading={false} search={action('Search called')}/>
  ));
```

从上面的代码可以看出，只需要简单的三步就可以完成一个 UI 组件的 Demo：

- import要做Demo的UI组件。
- storiesOf定义了一个组件目录。
- add添加Demo。

在构建项目的 storybook 时，一些可以帮助我们更有效的开发 Demo 小 Tips：

1. 尽可能的把目录结构与源代码结构保持一致。
2. 一个UI组件对应一个Demo文件，保持Demo代码的独立性和灵活性，可以为一个组件添加多个Demo，这样一眼就可以看到多个场景下的Demo状态。
3. Demo命名以UI组件名加上Demo缀。
4. 在组件参数复杂的场景下，可以单独提供一个fakeData的目录用于存放重用的UI组件Props数据。

4.一个完整的业务开发流程

在完成了上面三个步骤后，一个完整的 React Native 业务开发流程可简单分为如下几步：

1. 使用基础设计元素构建基础组件，通过Living Style Guide验收。
2. 使用基础组件组合业务组件，通过Living Style Guide验收。
3. 使用业务组件组合Page组件，通过Living Style Guide验收。
4. 使用Scene把Page组件的和应用的状态关联起来。
5. 使用Router把多个Scene串联起来，完成业务流程。

四、总结

随着前后端分离架构成为主流，越来越多的业务逻辑被推向前端，再加上用户对于体验的更高要求，前端的复杂性在一步一步的拔高。对前端复杂性的管理就显得越来越重要了。经过前端的各种框架，工具的推动，在前端工程化实践方面我们已经迈进了很多。而组件化开发就是笔者觉得其中比较好的一个方向，因为它不仅关注了当前的项目交付，还指导了团队的运作，帮助了后期的演进，甚至在程序员最讨厌的写文档的方面也给出了一个巧妙的解法。希望对该方法感兴趣的同学一起研究，改进。

深入浅出 Redis client/server 交互流程

作者 赵景波

综述

最近笔者阅读并研究 Redis 源码，在 Redis 客户端与服务器端交互这个内容点上，需要参考网上一些文章，但是遗憾的是发现大部分文章都断断续续的非系统性的，不能给读者此交互流程的整体把握。所以这里我尝试，站在源码的角度，将 Redis client/server 交互流程尽可能简单地展现给大家，同时也站在 DBA 的角度给出一些日常工作中注意事项。

Redis client/server 交互步骤分为以下 6 个步骤：

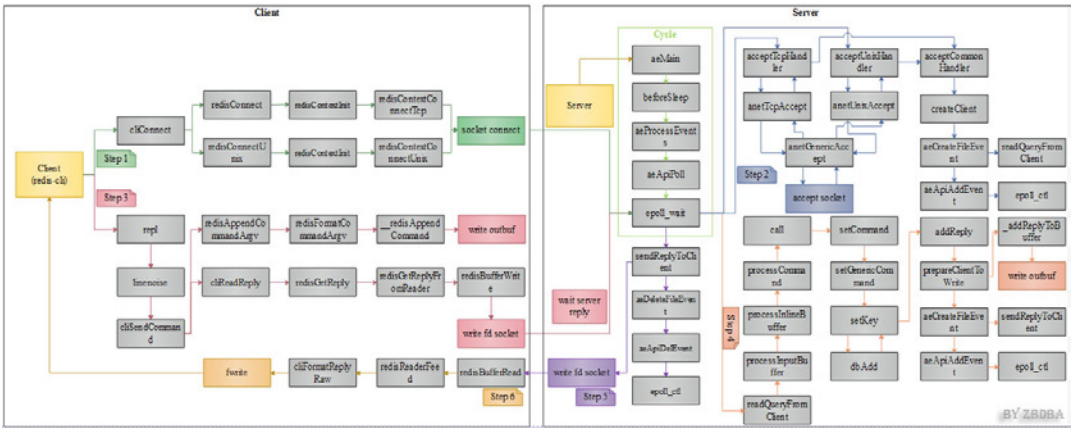
1. Client 发起socket 连接
2. Server 接受socket连接
3. 客户端 开始写入
4. Server 端接收写入
5. Server 返回写入结果
6. Client收到返回结果

注：为使文章尽可能简洁，这里只讨论客户端命令写入的过程，不讨论客户端命令读取的流程。

在进一步阅读和了解互动流程之前，请大家确保已经熟练掌握了 Linux Socket 建立流程和 epoll I/O 多路复用技术两个技术点，这对文章内容的理解至关重要。

交互的整体流程

在介绍 6 个步骤之前，首先看一下 Redis client/server 交互流程整体的程序执行流程图：



上图中 6 个步骤分别用不同的颜色箭头表示，并且最终结果也用相对应的颜色标识。

首先看看绿色框里面的循环执行的方法，最末是 epoll_wait 方法，即等待事件产生的方法。然后再看第 2、4、5 步骤的末尾都有 epoll_ctl 方法，即 epoll 事件注册函数。关于 epoll 的相关技术解析请参看文末一段。

在这里的循环还有个 beforeSleep 方法，其实它跟我们这次讨论的话题没有太大的关系。但是还是想给大家介绍一下。

beforeSleep 方法主要做以下几件事：

- 执行一次快速的主动过期检查，检查是否有过期的key
- 当有客户端阻塞时，向所有从库发送ACK请求
- unblock 在同步复制时候被阻塞的客户端
- 尝试执行之前被阻塞客户端的命令
- 将AOF缓冲区的内容写入到AOF文件中

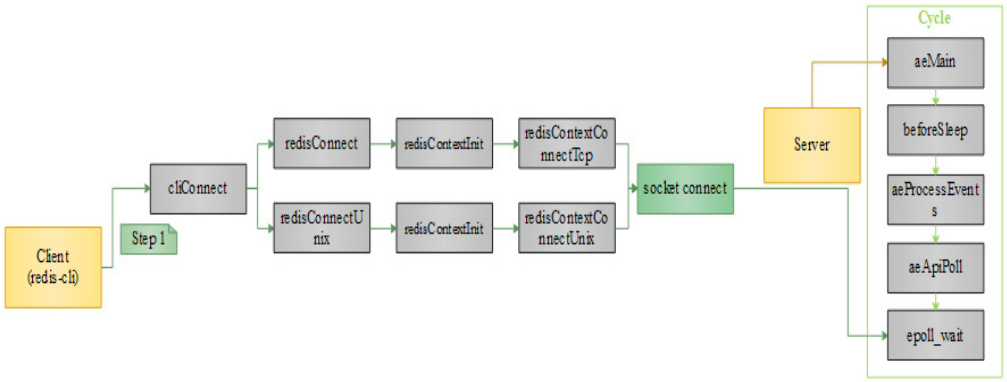
如果是集群，将会根据需要执行故障迁移、更新节点状态、保存 node.conf 配置文件。

如此，redis 整个事件管理器机制就比较清楚了。接下来进一步探讨并理解事件是如何触发并创建。

交互的六大步骤

下面正式开始介绍 redis client/server 交互的 6 大步骤

一、Client 发起socket 连接



这里以 redis-cli 客户端为例，当执行以下语句时：

```
[root@zbdba redis-3.0]# ./src/redis-cli -p 6379 -h 127.0.0.1
127.0.0.1:6379>
```

客户端会做如下操作：

1、获取客户端参数，如端口、ip 地址、dbnum、socket 等

也就是我们执行 ./src/redis-cli --help 中列出的参数

2、根据用户指定参数确定客户端处于哪种模式

目前共有：

```
Latency mode/Slave mode/Get RDB mode/Pipe mode/Find
big keys/Stat mode/Scan mode/Intrinsic latency mode
```

以上 8 种模式

例如：stat 模式

```
[root@zbdba redis-3.0]# ./src/redis-cli -p 6379 -h 127.0.0.1
--stat
------- data ------ -------------------- load --------------
----- - child -
keys          mem         clients blocked requests
connections
1         817.18K  2       0       1 (+0)             2
1         817.18K  2       0       2 (+1)             2
1         817.18K  2       0       3 (+1)             2
1         817.18K  2       0       4 (+1)             2
1         817.18K  2       0       5 (+1)             2
1         817.18K  2       0       6 (+1)             2
```

我们这里没有指定，就是默认的模式。

3、进入上图中 step1 的 cliConnect 方法，cliConnect 主要包含 redisConnect、redisConnectUnix 方法。这两个方法分别用于 TCP Socket 连接以及 Unix Socket 连接，Unix Socket 用于同一主机进程间的通信。我们上面是采用的 TCP Socket 连接方式也就是我们平常生产环境常用的方式，这里不讨论 Unix Socket 连接方式，如果要使用 Unix Socket 连接方式，需要配置 unixsocket 参数，并且按照下面方式进行连接：

```
[root@zbdba redis-3.0]# ./src/redis-cli -s /tmp/redis.sock
```

```
redis /tmp/redis.sock>
```

4、进入 redisContextInit 方法，redisContextInit 方法用于创建一个 Context 结构体保存在内存中，如下：

```
/* Context for a connection to Redis */
typedef struct redisContext {
    int err; /* Error flags, 0 when there is no error */
     char errstr[128]; /* String representation of error when
applicable */
    int fd;
    int flags;
    char *obuf; /* Write buffer */
    redisReader *reader; /* Protocol reader */
} redisContext;
```

主要用于保存客户端的一些东西，最重要的就是 write buffer 和 redisReader，write buffer 用于保存客户端的写入，redisReader 用于保存协议解析器的一些状态。

5、进入 redisContextConnectTcp 方法，开始获取 IP 地址和端口用于建立连接，主要方法如下：

```
s = socket(p->ai_family,p->ai_socktype,p->ai_protocol
connect(s,p->ai_addr,p->ai_addrlen)
```

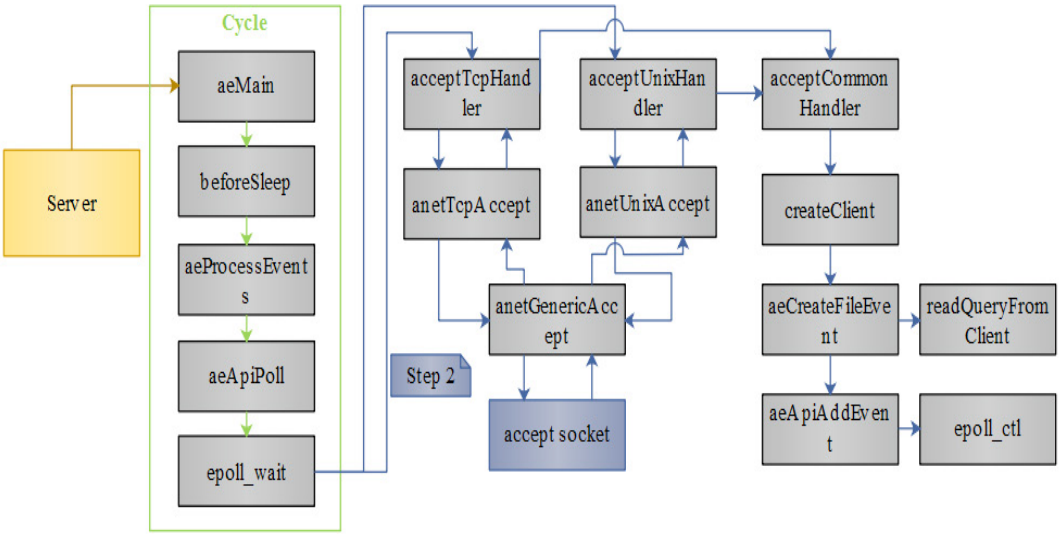
到此客户端向服务端发起建立 socket 连接，并且等待服务器端响应。

当然 cliConnect 方法中还会调用 cliAuth 方法用于权限验证、cliSelect 用于 db 选择，这里不着重讨论。

二、Server 接受socket连接

服务器接收客户端的请求首先是从 epoll_wait 取出相关的事件，然后进入上图中 step2 中的方法，执行 acceptTcpHandler 或者 acceptUnixHandler 方法，那么这两个方法对应的事件是在什么时候注册的呢？他们是在服务器端初始化的时候创建。下面看看服务器端在初始化

的时候与 socket 相关的地方



1、打开 TCP 监听端口

```
    if (server.port != 0 &&
            listenToPort(server.port,server.ipfd,&server.ipfd_
count) == REDIS_ERR)
        exit(1);
```

2、打开 Unix 本地端口

```
  if (server.unixsocket != NULL) {
        unlink(server.unixsocket); /* don't care if this fails
*/
           server.sofd = anetUnixServer(server.neterr,server.
unixsocket,
            server.unixsocketperm, server.tcp_backlog);
        if (server.sofd == ANET_ERR) {
                redisLog(REDIS_WARNING, "Opening socket: %s",
server.neterr);
            exit(1);
        }
        anetNonBlock(NULL,server.sofd);
    }
```

3、为 TCP 连接关联连接应答处理器 (accept)

```
for (j = 0; j < server.ipfd_count; j++) {
          if (aeCreateFileEvent(server.el, server.ipfd[j], AE_
READABLE,
             acceptTcpHandler,NULL) == AE_ERR)
             {
                 redisPanic(
                      "Unrecoverable error creating server.ipfd
file event.");
             }
    }
```

4、为 Unix Socket 关联应答处理器

```
if (server.sofd > 0 && aeCreateFileEvent
     (server.el,server.sofd,AE_READABLE,
        acceptUnixHandler,NULL) == AE_ERR)
redisPanic("Unrecoverable error creating server.sofd file
event.");
```

在 1/2 步骤涉及到的方法中是 Linux Socket 的常规操作，获取 IP 地址，端口。最终通过 socket、bind、listen 方法建立起 Socket 监听。也就是上图中 acceptTcpHandler 和 acceptUnixHandler 下面对应的方法。

在 3/4 步骤涉及到的方法中采用 aeCreateFileEvent 方法创建相关的连接应答处理器，在客户端请求连接的时候触发。

所以现在整个 socket 连接建立流程就比较清楚了，如下：

- 服务器初始化建立socket监听
- 服务器初始化创建相关连接应答处理器,通过epoll_ctl注册事件
- 客户端初始化创建socket connect 请求
- 服务器接受到请求，用epoll_wait方法取出事件

服务器执行事件中的方法 (acceptTcpHandler/acceptUnixHandler) 并接受 socket 连接

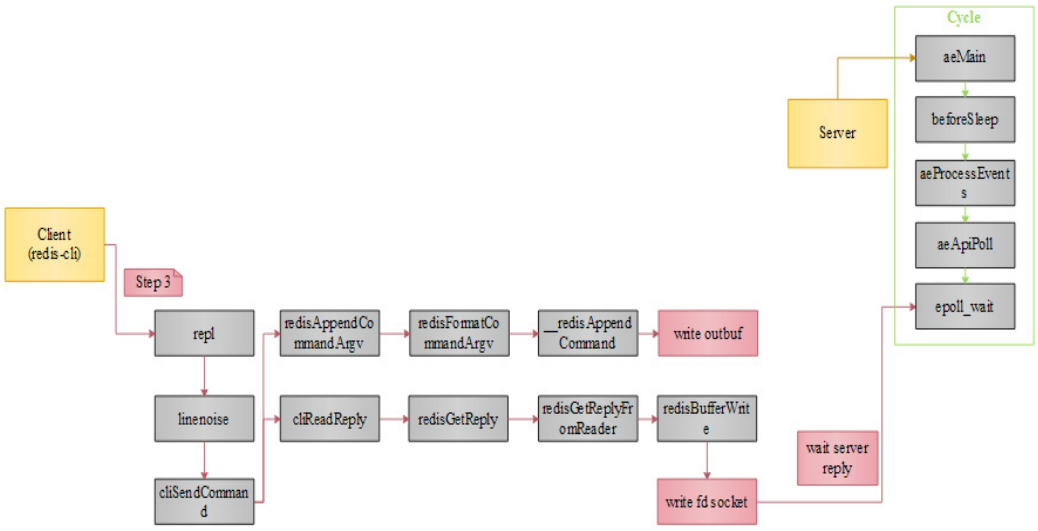
至此客户端和服务器端的 socket 连接已经建立，但是此时服务器端还继续做了 2 件事：

采用 createClient 方法在服务器端为客户端创建一个 client，因为 I/O 复用所以需要为每个客户端维持一个状态。这里的 client 也在内存中分配了一块区域，用于保存它的一些信息，如套接字描述符、默认数据库、查询缓冲区、命令参数、认证状态、回复缓冲区等。这里提醒一下 DBA 同学关于 client-output-buffer-limit 设置，设置不恰当将会引起客户端中断。

采用 aeCreateFileEvent 方法在服务器端创建一个文件读事件并且绑定 readQueryFromClient 方法。

可以从图中得知，aeCreateFileEvent 调用 aeApiAddEvent 方法最终通过 epoll_ctl 方法进行注册事件。

三、客户端开始写入



客户端在与服务器端建立好 socket 连接之后，开始执行上图中 step3 的 repl 方法。从图中可知 repl 方法接受输入输出主要是采用 linenoise 插件。当然这是针对 redis-cli 客户端哦。linenoise 是一款

优秀的命令行编辑库，被广泛的运用在各种 DB 上，如 Redis、MongoDB，这里不详细讨论。客户端写入流程分为以下几步：

1、linenoise 等待接受用户输入

2、linenoise 将用户输入内容传入 cliSendCommand 方法，cliSendCommand 方法会判断命令是否为特殊命令，如：

```
help
info
cluster nodes
cluster info
client list
shutdown
monitor
subscribe
psubscribe
sync
psync
```

客户端会根据以上命令设置对应的输出格式以及客户端的模式，因为这里我们是普通写入，所以不会涉及到以上的情况。

3、cliSendCommand 方法会调用 redisAppendCommandArgv 方法，redisAppendCommandArgv 方法会调用 redisFormatCommandArgv 和 __redisAppendCommand 方法

redisFormatCommandArgv 方法用于将客户端输入的内容格式化成 redis 协议：

例如：

```
set zbdba jingbo
*3\r\n$3\r\n set\r\n $5\r\n zbdba\r\n $6\r\n jingbo
```

__redisAppendCommand 方法用于将命令写入到 outbuf 中

接着客户端进入下一个流程，将 outbuf 内容写入到套接字描述符上

并传输到服务器端。

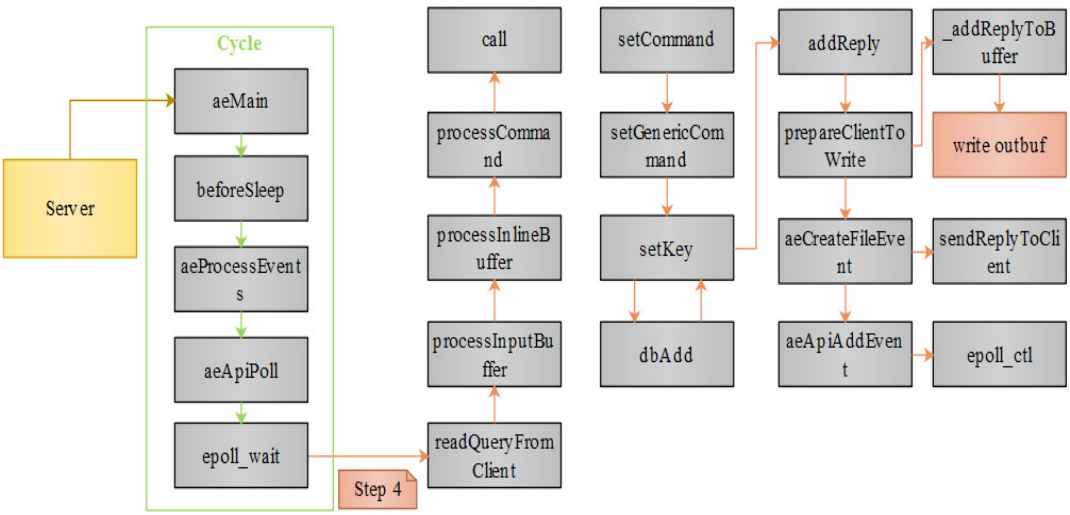
4、进入 redisGetReply 方法，该方法下主要有 redisGetReplyFromReader 和 redisBufferWrite 方法，redisGetReplyFromReader 主要用于读取挂起的回复，redisBufferWrite 方法用于将当前 outbuf 中的内容写入到套接字描述符中，并传输内容。

主要方法如下：

```
nwritten = write(c->fd,c->obuf,sdslen(c->obuf));
```

此时客户端等待服务器端接收写入。

四、Server 端接收写入



服务器端依然在进行事件循环，在客户端发来内容的时候触发，对应的文件读取事件。这就是之前创建 socket 连接的时候建立的事件，该事件绑定的方法是 readQueryFromClient 。此时进入 step4 的 readQueryFromClient 方法。

readQueryFromClient 方法用于读取客户端的发送的内容。它的执行步骤如下：

1、在 readQueryFromClient 方法中从服务器端套接字描述符中读取

客户端的内容到服务器端初始化 client 的查询缓冲中，主要方法如下：

```
nread = read(fd, c->querybuf+qblen, readlen);
```

2、交给 processInputBuffer 处理，processInputBuffer 主要包含两个方法，processInlineBuffer 和 processCommand。processInlineBuffer 方法用于采用 redis 协议解析客户端内容并生成对应的命令并传给 processCommand 方法，processCommand 方法则用于执行该命令

3、processCommand 方法会以下操作：

- 处理是否为quit命令。
- 对命令语法及参数会进行检查。
- 这里如果采取认证也会检查认证信息。
- 如果Redis为集群模式，这里将进行hash计算key所属slot并进行转向操作。
- 如果设置最大内存，那么检查内存是否超过限制，如果超过限制会根据相应的内存策略删除符合条件的键来释放内存。
- 如果这是一个主服务器，并且这个服务器之前执行bgsave发生了错误，那么不执行命令。
- 如果min-slaves-to-write开启，如果没有足够多的从服务器将不会执行命令；注：所以DBA在此的设置非常重要，建议不是特殊场景不要设置。
- 如果这个服务器是一个只读从库的话，拒绝写入命令。
- 在订阅于发布模式的上下文中，只能执行订阅和退订相关的命令。
- 当这个服务器是从库，master_link down 并且slave-serve-

stale-data 为 no 只允许info 和slaveof命令。

- 如果服务器正在载入数据到数据库，那么只执行带有REDIS_CMD_LOADING标识的命令。
- lua脚本超时，只允许执行限定的操作，比如shutdown、script kill 等。

4、最后进入 call 方法。

call 方法会调用 setCommand，因为这里我们执行的 set zbdba jingbo，set 命令对应 setCommand 方法，redis 服务器端在开始初始化的时候就会初始化命令表，命令表如下：

```
struct redisCommand redisCommandTable[] = {
    {"get",getCommand,2,"r",0,NULL,1,1,1,0,0},
    {"set",setCommand,-3,"wm",0,NULL,1,1,1,0,0},
    {"setnx",setnxCommand,3,"wm",0,NULL,1,1,1,0,0},
    {"setex",setexCommand,4,"wm",0,NULL,1,1,1,0,0},
    {"psetex",psetexCommand,4,"wm",0,NULL,1,1,1,0,0},
    {"append",appendCommand,3,"wm",0,NULL,1,1,1,0,0},
    {"strlen",strlenCommand,2,"r",0,NULL,1,1,1,0,0},
    {"del",delCommand,-2,"w",0,NULL,1,-1,1,0,0},
    {"exists",existsCommand,2,"r",0,NULL,1,1,1,0,0},
    {"setbit",setbitCommand,4,"wm",0,NULL,1,1,1,0,0},
    ....
}
```

所以如果是其他的命令会调用其他相对应的方法。call 方法还会做一些事件，比如发送命令到从库、发送命令到 aof、计算命令执行的时间。

5、setCommand 方法，setCommand 方法会调用 setGenericCommand 方法，该方法首先会判断该 key 是否已经过期，最后调用 setKey 方法。

这里需要说明一点的是，通过以上的分析。redis 的 key 过期包括主动检测以及被动监测。

主动监测：

- 在beforeSleep方法中执行key快速过期检查，检查模式为ACTIVE_EXPIRE_CYCLE_FAST。周期为每个事件执行完成时间到下一次事件循环开始。
- 在serverCron方法中执行key过期检查，这是key过期检查主要的地方，检查模式为ACTIVE_EXPIRE_CYCLE_SLOW，serverCron方法执行周期为1秒钟执行server.hz 次，hz默认为10，所以约100ms执行一次。hz设置越大过期键删除就越精准，但是cpu使用率会越高，这里我们线上redis采用的默认值。redis主要是在这个方法里删除大部分的过期键。

被动监测

- 使用内存超过最大内存被迫根据相应的内存策略删除符合条件的key。
- 在key写入之前进行被动检查，检查key是否过期，过期就进行删除。
- 还有一种不友好的方式，就是randomkey命令，该命令随机从redis获取键，每次获取到键的时候会检查该键是否过期。

以上主要是让运维的同学更加清楚 redis 的 key 过期删除机制。

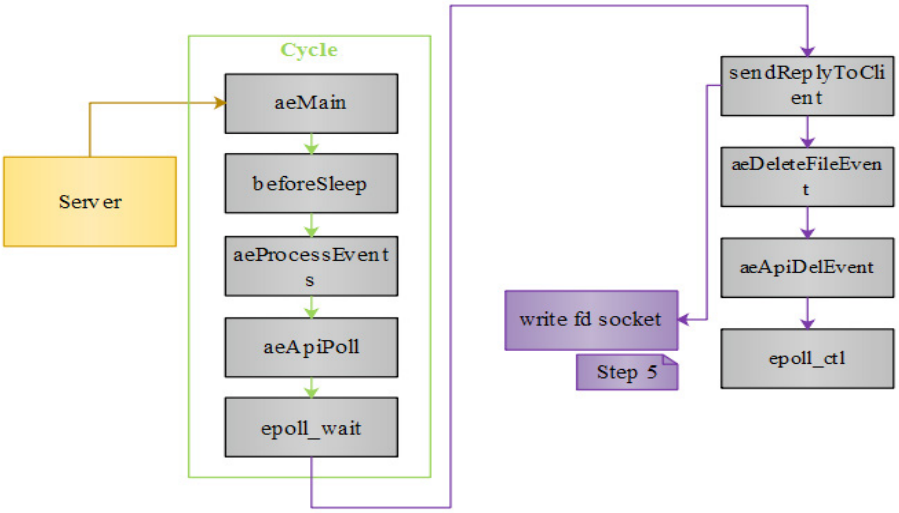
6、进入 setKey 方法，setKey 方法最终会调用 dbAdd 方法，其实最终就是将该键值对存入服务器端维护的一个字典中，该字典是在服务器初始化的时候创建，用于存储服务器的相关信息，其中包括各种数据类型的键值存储。完成了写入方法时候，此时服务器端会给客户端返回结果。

7、进入 prepareClientToWrite 方法然后通过调用 _addReplyToBuffer 方法将返回结果写入到 outbuf 中（客户端连接时创建

的 client）

8、通过 aeCreateFileEvent 方法注册文件写事件并绑定 sendReplyToClient 方法

五、Server 返回写入结果



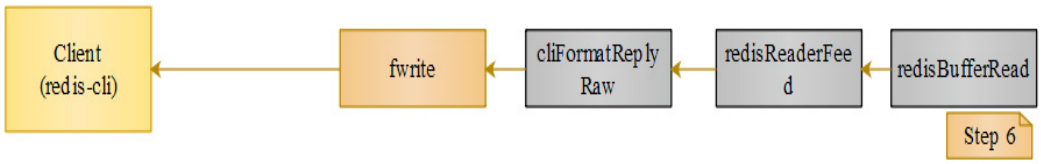
此时按照惯例，aeMain 主函数循环，监测到新注册的事件，调用 sendReplyToClient 方法。sendReplyToClient 方法主要包含两个操作：

1、将 outbuf 内容写入到套接字描述符并传输到客户端，主要方法如下：

```
nwritten = write(fd, c->buf+c->sentlen, c->bufpos-c->sentlen);
```

2、aeDeleteFileEvent 用于删除 文件写事件

六、Client收到返回结果



客户端接收到服务器端的返回调用 redisBufferRead 方法，该方法主要用于从 socket 中读取数据。主要方法如下：

```
nread = read(c->fd,buf,sizeof(buf));
```

并且将读取的数据交由 redisReaderFeed 方法，该方法主要用于将数据交给回复解析器处理，也就是 cliFormatReplyRaw，该方法将回复内容格式化。最终通过

```
fwrite(out,sdslen(out),1,stdout);
```

方法返回给客户端并打印展示给用户。

至此整个写入流程完成。以上还有很多细节没有说到，感兴趣的朋友可以自行阅读源码。

结语

在深入了解一个 DB 的时候，我的第一步就是去理解它执行一条命令执行的整个流程，这样就能对它整个运行流程较为熟悉，接着我们可以去深入各个细节的部分，比如 Redis 的相关数据结构、持久化以及高可用相关的东西。写这篇文章的初衷就是希望我们更加轻松的走好这第一步。这里还需要提醒的是，在我们进行 Redis 源码阅读的时候最关键的是需要灵活的使用 GDB 调试工具，它能帮我们更好的去理顺相关执行步骤，从而让我们更加容易理解其实现原理。

附录：两个相关重要知识点

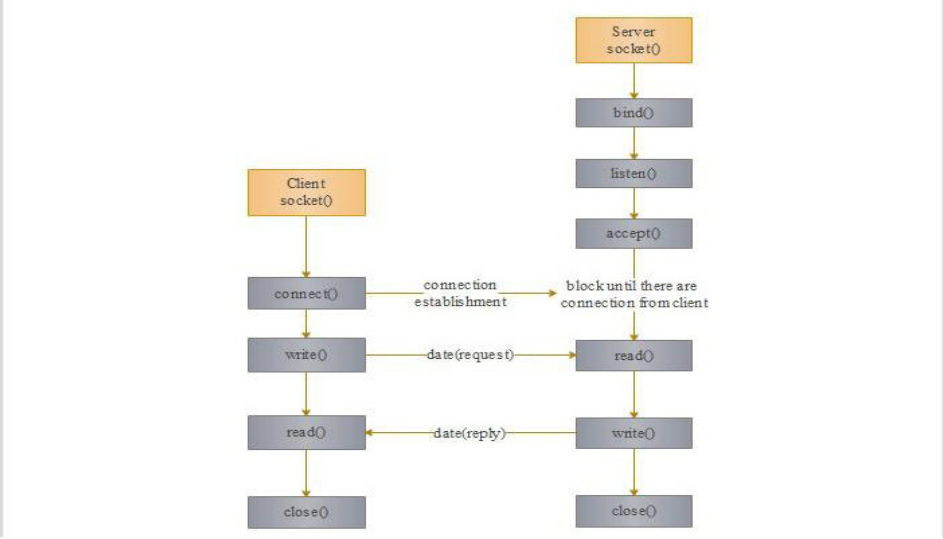
1、Linux Socket 建立流程

Linux socket 建立过程如上图所示。在 Linux 编程时，无论是操作文件还是网络操作时都是通过文件描述符来进行读写的，但是他们有一点区别，这里我们不具体讨论，我们将网络操作时就称为套接字描述符。大家可以自行用 c 写一个简单的 demo，这里就不详细说明了。

这里列出几个重要的方法：

```
int socket(int family,int type,int protocol);
```

```
int connect(int sockfd,const struct sockaddr *
servaddr,socklen_taddrlen);
int bind(int sockfd,const struct sockaddr * myaddr,socklen_
taddrlen);
int listen(int sockfd,int backlog);
int accept(int sockfd,struct sockaddr *cliaddr,socklen_t *
addrlen);
```



Redis client/server 也是基于 linux socket 连接进行交互，并且最终调用以上方法绑定 IP，监听端口最终与客户端建立连接。

2、epoll I/O 多路复用技术

这里重点介绍一下 epoll，因为 Redis 事件管理器核心实现基本依赖于它。首先来看 epoll 是什么，它能做什么？

epoll 是在 Linux 2.6 内核中引进的，是一种强大的 I/O 多路复用技术，上面我们已经说到在进行网络操作的时候是通过文件描述符来进行读写的，那么平常我们就是一个进程操作一个文件描述符。然而 epoll 可以通过一个文件描述符管理多个文件描述符，并且不阻塞 I/O。这使得我们单进程可以操作多个文件描述符，这就是 redis 在高并发性能还如此强大的原因之一。

下面简单介绍 epoll 主要的三个方法：

1. int epoll_create(int size) //创建一个epoll句柄用于监听文件描述符FD，size用于告诉内核这个监听的数目一共有多大。该epoll句柄创建后在操作系统层面只会占用一个fd值，但是它可以监听size+1 个文件描述符。
2. int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event)//epoll事件注册函数。
3. int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout)//等待事件的产生。

Redis 的事件管理器主要是基于 epoll 机制，先采用 epoll_ctl 方法 注册事件，然后再使用 epoll_wait 方法取出已经注册的事件。

我们知道 redis 支持多种平台，那么 redis 在这方面是如何兼容其他平台的呢？ Redis 会根据操作系统的类型选择对应的 IO 多路复用实现。

```
#ifdef HAVE_EVPORT
#include "ae_evport.c"
#else
    #ifdef HAVE_EPOLL
    #include "ae_epoll.c"
    #else
        #ifdef HAVE_KQUEUE
        #include "ae_kqueue.c"
        #else
        #include "ae_select.c"
        #endif
    #endif
#endif
ae_evport.c sun solaris
ae_poll.c linux
ae_select.c unix/linux epoll是select的加强版
ae_kqueue BSD/Apple
```

以上只是简单的介绍，大家需要详细了解了 epoll 机制才能更好的理解后面的东西。

微信 PaxosStore：深入浅出 Paxos 算法协议

作者 郑建军

引言

早在 1990 年，Leslie Lamport（即 LaTeX 中的"La"，微软研究院科学家，获得 2013 年图灵奖）向 ACM Transactions on Computer Systems (TOCS) 提交了关于 Paxos 算法的论文 The Part-Time Parliament。几位审阅人表示，虽然论文没什么特别的用处，但还是有点意思，只是要把 Paxos 相关的故事背景全部删掉。Leslie Lamport 心高气傲，觉得审阅人没有丝毫的幽默感，于是撤回文章不再发表。直到 1998 年，用户开始支持 Paxos，Leslie Lamport 重新发表文章，但相比 1990 年的版本，文章没有太大的修改，所以还是不好理解。于是在 2001 年，为了通俗性，Leslie Lamport 简化文章发表了 Paxos Made Simple，这次文中没有一个公式。

但事实如何？大家不妨读一读 Paxos Made Simple。Leslie Lamport

在文中渐进式地、从零开始推导出了 Paxos 协议，中间用数学归纳法进行了证明。可能是因为表述顺序的问题，导致这篇文章似乎还是不好理解。

于是，基于此背景，本文根据 Paxos Made Simple，重新描述 Paxos 协议，提供两种证明方法，给出常见的理解误区。期望读者通过阅读本文，再结合 Paxos Made Simple，就可以深入理解基本的 Paxos 协议理论。

基本概念

- Proposal Value：提议的值；
- Proposal Number：提议编号，要求提议编号不能冲突；
- Proposal：提议 = 提议的值 + 提议编号；
- Proposer：提议发起者；
- Acceptor：提议接受者；
- Learner：提议学习者。

注意，提议跟提议的值是有区别的，后面会具体说明。协议中 Proposer 有两个行为，一个是向 Acceptor 发 Prepare 请求，另一个是向 Acceptor 发 Accept 请求；Acceptor 则根据协议规则，对 Proposer 的请求作出应答；最后 Learner 可以根据 Acceptor 的状态，学习最终被确定的值。

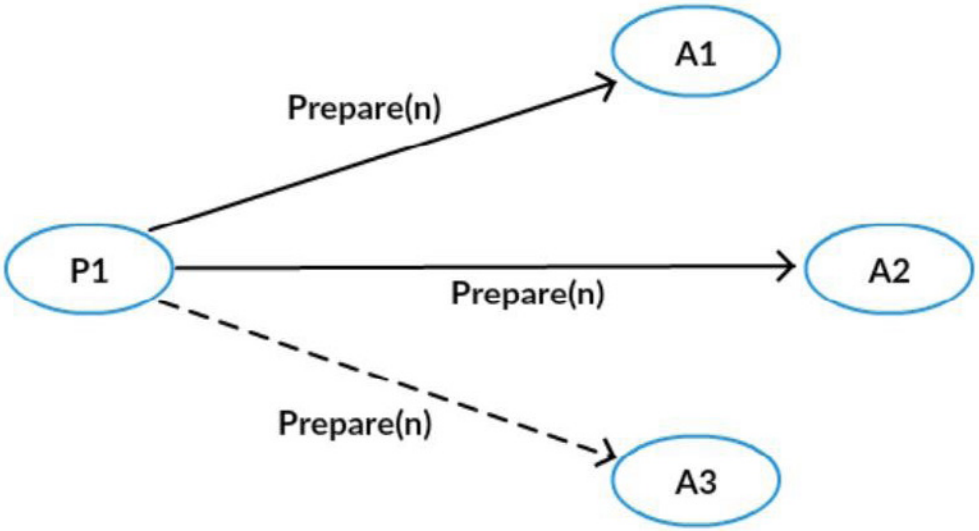
方便讨论，在本文中，记 {n，v} 为提议编号为 n，提议的值为 v 的提议，记 (m，{n，v}) 为承诺了 Prepare（m）请求，并接受了提议 {n，v}。

协议过程

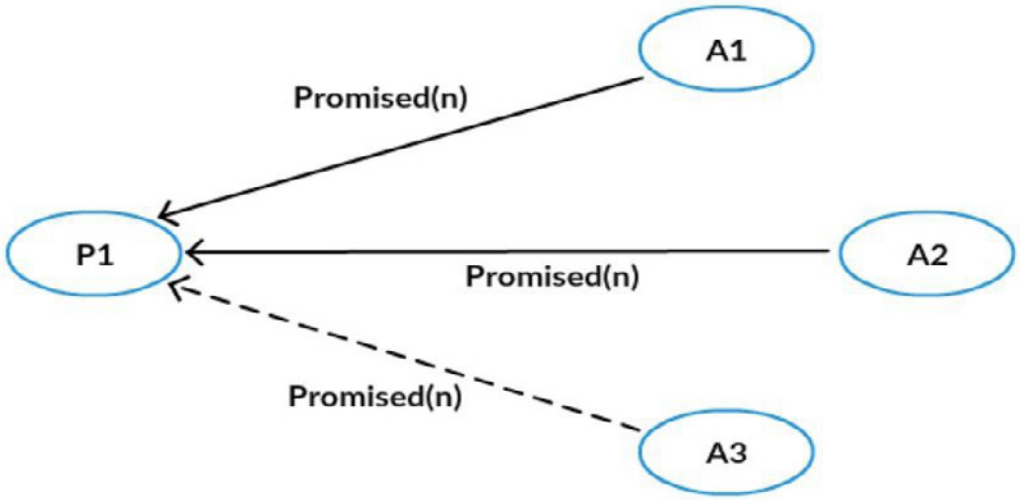
第一阶段 A

Proposer 选择一个提议编号 n，向所有的 Acceptor 广播 Prepare（n）请求。

第一阶段 B

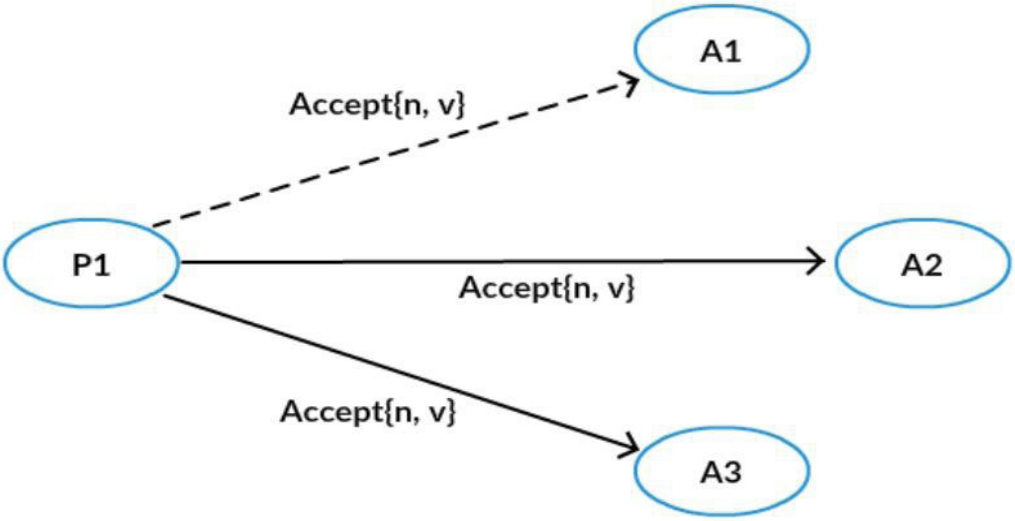


Acceptor 接收到 Prepare（n）请求，若提议编号 n 比之前接收的 Prepare 请求都要大，则承诺将不会接收提议编号比 n 小的提议，并且带上之前 Accept 的提议中编号小于 n 的最大的提议，否则不予理会。



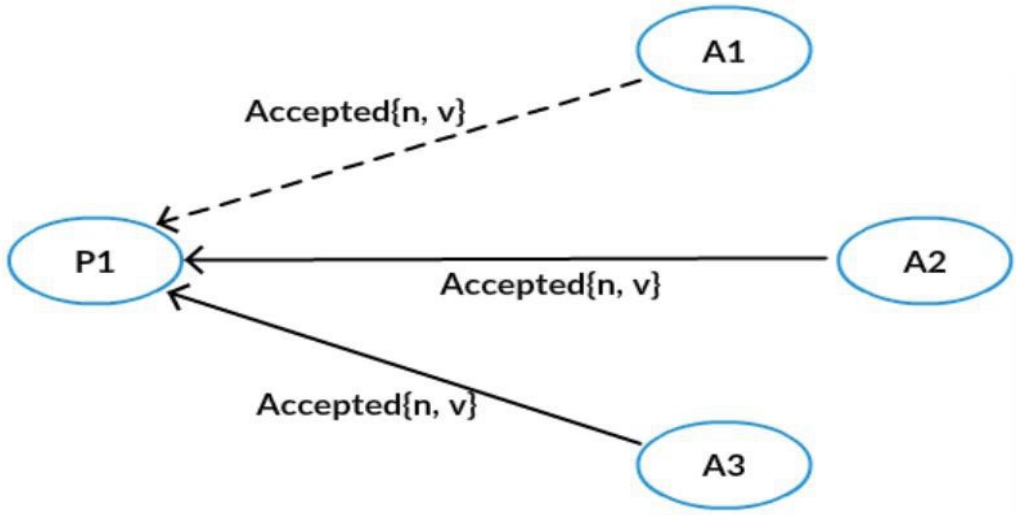
第二阶段 A

Proposer 得到了多数 Acceptor 的承诺后，如果没有发现有一个 Acceptor 接受过一个值，那么向所有的 Acceptor 发起自己的值和提议编号 n，否则，从所有接受过的值中选择对应的提议编号最大的，作为提议的值，提议编号仍然为 n。



第二阶段 B

Acceptor 接收到提议后，如果该提议编号不违反自己做过的承诺，则接受该提议。



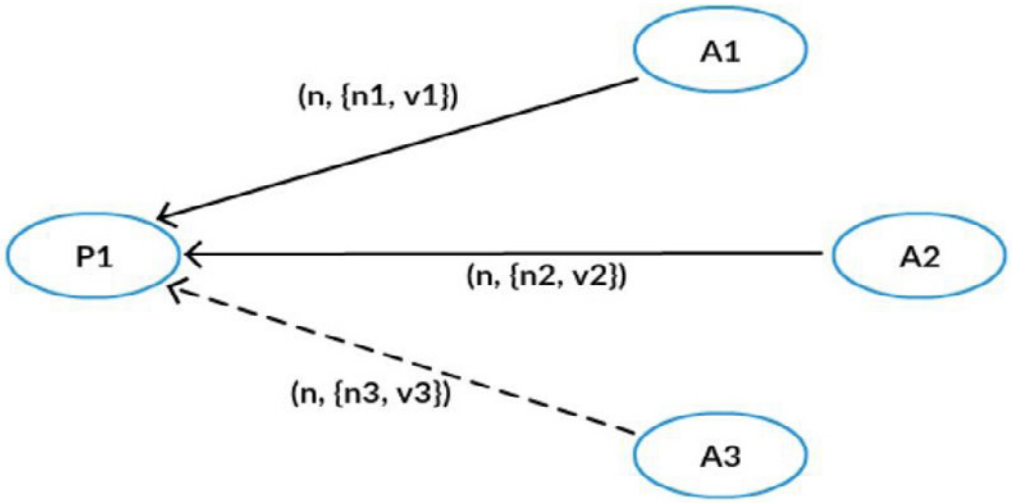
需要注意的是，Proposer 发出 Prepare(n)请求后，得到多数派的应答，然后可以随便再选择一个多数派广播 Accept 请求，而不一定要将 Accept 请求发给有应答的 Acceptor，这是常见的 Paxos 理解误区。

小结

上面的图例中，P1 广播了 Prepare 请求，但是给 A3 的丢失，不过

A1、A2 成功返回了，即该 Prepare 请求得到多数派的应答，然后它可以广播 Accept 请求，但是给 A1 的丢了，不过 A2，A3 成功接受了这个提议。因为这个提议被多数派（A2，A3 形成多数派）接受，我们称被多数派接受的提议对应的值被 Chosen。

三个 Acceptor 之前都没有接受过 Accept 请求，所以不用返回接受过的提议，但是如果接受过提议，则根据第一阶段 B，要带上之前 Accept 的提议中编号小于 n 的最大的提议。



Proposer 广播 Prepare 请求之后，收到了 A1 和 A2 的应答，应答中携带了它们之前接受过的 {n1, v1} 和 {n2, v2}，Proposer 则根据 n1，n2 的大小关系，选择较大的那个提议对应的值，比如 n1 > n2，那么就选择 v1 作为提议的值，最后它向 Acceptor 广播提议 {n, v1}。

Paxos 协议最终解决什么问题

当一个提议被多数派接受后，这个提议对应的值被 Chosen（选定），一旦有一个值被 Chosen，那么只要按照协议的规则继续交互，后续被 Chosen 的值都是同一个值，也就是这个 Chosen 值的一致性问题。

协议证明

上文就是基本 Paxos 协议的全部内容，其实是一个非常确定的数学问题。下面用数学语言表达，进而用严谨的数学语言加以证明。

Paxos 原命题

如果一个提议{n0,v0}被大多数 Acceptor 接受，那么不存在提议{n1,v1}被大多数 Acceptor 接受，其中 n0 < n1，v0 != v1。

Paxos 原命题加强

如果一个提议{n0，v0}被大多数 Acceptor 接受，那么不存在 Acceptor 接受提议{n1，v1}，其中 n0 < n1，v0 != v1。

Paxos 原命题进一步加强

如果一个提议{n0，v0}被大多数 Acceptor 接受，那么不存在 Proposer 发出提议{n1，v1}，其中 n0 < n1，v0 != v1。

如果“Paxos 原命题进一步加强”成立，那么“Paxos 原命题”显然成立。下面我们通过证明“Paxos 原命题进一步加强”，从而证明“Paxos 原命题”。论文中是使用数学归纳法进行证明的，这里用比较紧凑的语言重新表述证明过程。

归纳法证明

假设，提议{m，v}（简称提议 m）被多数派接受，那么提议 m 到 n（如果存在）对应的值都为 v，其中 n 不小于 m。

这里对 n 进行归纳假设，当 n = m 时，结论显然成立。

设 n = k 时结论成立，即如果提议{m，v}被多数派接受，

那么提议 m 到 k 对应的值都为 v，其中 k 不小于 m。

当 n = k+1 时，若提议 k+1 不存在，那么结论成立。

若提议 k+1 存在，对应的值为 v1，

因为提议 m 已经被多数派接受，又 k+1 的 Prepare 被多数派承诺并返回结果。

基于两个多数派必有交集，易知提议 k+1 的第一阶段 B 有带提议回来。

那么 v1 是从返回的提议中选出来的，不妨设这个值是选自提议 {t，v1} 。

根据第二阶段 B，因为 t 是返回的提议中编号最大，所以 t >= m。

又由第一阶段 A，知道 t < n。所以根据假设 t 对应的值为 v。

即有 v1 = v。所以由 n = k 结论成立，可以推出 n = k+1 成立。

于是对于任意的提议编号不小于 m 的提议 n，对应的值都为 v。

所以命题成立。

反证法证明

假设存在，不妨设 n1 是满足条件的最小提议编号。

即存在提议 {n1，v1}，其中 n0 < n1，v0 != v1。----------（A）

那么提议 n0，n0+1，n0+2，...，n1-1 对应的值为 v0。----------（B）

由于存在提议 {n1，v1}，则说明大多数 Acceptor 已经接收 n1 的 Prepare，并承诺将不会接受提议编号比 n1 小的提议。

又因为 {n0，v0} 被大多数 Acceptor 接受，

所以存在一个 Acceptor 既对 n1 的 Prepare 进行了承诺，又接受了提议 n0。

由协议的第二阶段 B 知，这个 Acceptor 先接受了 {n0，v0} 。

所以发出 {n1，v1} 提议的 Proposer 会从大多数的 Acceptor 返回中得知，

至少某个编号不小于 n0 而且值为 v0 的提议已经被接受。------（C）

由协议的第二阶段 A 知，

该 Proposer 会从已经被接受的值中选择一个提议编号最大的，作为提议的值。

由（C）知该提议编号不小于 n0，由协议第二阶段 B 知，该提议编号小于 n1，

于是由（B）知 v1 == v0，与（A）矛盾。

所以命题成立。

通过上面的证明过程，我们反过来回味一下协议中的细节。

- 为什么要被多数派接受？因为两个多数派之间必有交集，所以Paxos协议一般是2F+1个Acceptor，然后允许最多F个Acceptor停机，而保证协议依然能够正常进行，最终得到一个确定的值。
- 为什么需要做一个承诺？可以保证第二阶段A中Proposer的选择不会受到未来变化的干扰。另外，对于一个Acceptor而言，这个承诺决定了它回应提议编号较大的Prepare请求，和接受提议编号较小的Accept请求的先后顺序。
- 为什么第二阶段A要从返回的协议中选择一个编号最大的？这样选出来的提议编号一定不小于已经被多数派接受的提议编号，进而可以根据假设得到该提议编号对应的值是Chosen的那个值。

原文的第一阶段 B

Acceptor 接收到 Prepare（n）请求，若提议编号 n 比之前接收的 Prepare 请求都要大，则承诺将不会接收提议编号比 n 小的提议，并且带上之前 Accept 的提议中编号最大的提议，否则不予理会。

相对上面的表达少了“比 n 小的”，通过邮件向 Leslie Lamport 请教了这个问题，他表示接受一个提议，包含回应了一个 Prepare 请求。这

个有点隐晦，但也完全合理，有了这个条件，上面的证明也就通顺了。就是说 Acceptor 接受过的提议的编号总是不大于承诺过的提议编号，于是可以将这个“比 n 小的”去掉，在实际工程实践中我们往往只保存接受过的提议中编号最大的，以及承诺过的 Prepare 请求编号最大的。

Leslie Lamport 也表示在去掉“比 n 小的”的情况下，就算接受一个提议不包含回应一个 Prepare 请求，最终结论也是对的，因为前者明显可以推导出后者，去掉反而把条件加强了。

假如返回的提议中有编号大于 n 的，比如 {m，v}，那么肯定存在多数派承诺拒绝小于 m 的 Accept 请求，所以提议 {n，v} 不可能被多数派接受。

学习过程

如果一个提议被多数 Acceptor 接受，则这个提议对应的值被选定。

一个简单直接的学习方法就是，获取所有 Acceptor 接受过的提议，然后看哪个提议被多数的 Acceptor 接受，那么该提议对应的值就是被选定的。

另外，也可以把 Learner 看作一个 Proposer，根据协议流程，发起一个正常的提议，然后看这个提议是否被多数 Acceptor 接受。

这里强调“一个提议被多数 Acceptor 接受”，而不是“一个值被多数 Acceptor”接受，如果是后者会有什么问题？

A	B	C
(1, {1, v1})	(1, {})	
	(3, {})	(3, {3, v3}
(4, {4, v1})	(4, {})	
	(5, {5, v3})	(5, {3, v3})
(7, {7, v1})		(7, {7, v1})

提议 {3，v3}，{5，v3} 分别被 B、C 接受，即有 v3 被多数派接受，但不能说明 v3 被选定（Chosen），只有提议 {7，v1} 被多数派（A 和 C 组成）接受，我们才能说 v1 被选定，而这个选定的值随着协议继续进行不会改变。

总结

“与其预测未来，不如限制未来”，这应该是 Paxos 协议的核心思想。如果你在阅读 Paxos 的这篇论文时感到困惑，不妨找到“限制”的段落回味一番。Paxos 协议本身是比较简单的，如何将 Paxos 协议工程化，才是真正的难题。

目前在微信核心存储 PaxosStore 中，每分钟调用 Paxos 协议过程数十亿次量级，而《微信 PaxosStore 内存云揭秘：十亿 Paxos/ 分钟的挑战》一文， 则对内存云子系统做了展开。

后续我们将发表更多的实践方案，包括万亿级别的海量闪存存储，支持单表亿行的 NewSQL 解决方案，以及有别于业界的开源实现，PaxosStore 架构方案基于非租约的 Paxos 实现等内容。

作者介绍

郑建军，微信工程师，目前负责微信基础存储服务，致力于强一致、高可用的大规模分布式存储系统的设计与研发。

我们为什么选择 Vue.js 而不是 React

作者 Anton Sidashin ，译者 薛命灯

最近，Qwintry 开发团队把很多项目都迁移至 Vue.js，包括所有遗留的项目和新开始的项目：

- 遗留的Drupal系统（qwintry.com）
- 新的qwintry.com分支，该分支是对旧有项目的彻底重写
- 基于Yii 2的B2B系统（logistics.qwintry.com）
- 其他大大小小的内部和外部项目（大部分使用PHP和Node.js作为后端）

Qwintry 在全球有差不多五十万用户，我们在美国和德国各有一个仓库，而且我们是美国最大的货物转运公司，业务主要面向东欧和中东地区。

简单地说，我们帮助上述地区的人们购买美国网上商店的商品，并帮他们节省跨国运费。我们使用自己的 IT 系统和物流系统为他们提供高质量的转运服务，而且费用是非常实惠的。

我们的系统有很多代码，大部分是 PHP 和 JS 代码。

我们分别使用 React、Vue.js 和 Angular 2 构建了一个用户计算器作为对比实验，经过比较之后，我们最终决定采用 Vue.js。

React.js 之我见

React 的出现震惊了 JS 世界，几乎成为 JS 开发的首选框架。

我使用 React 构建了一些单页应用和动态控件，我也玩过 React Native（iOS）和 Redux。我认为，对于面向状态的应用来说，React 再合适不过了，而且 React 为我们提供了真正的面向函数编程。React Native 在很大程度上改变了原生应用的开发。

对我来说，React 的不足之处在于：

纯净、不可变性和解决问题的意识形态

不要误解，我其实很感激 React 所带给我们的纯净的函数编码方式和简洁的渲染手法，在实际应用中，这些都算得上好东西。我想说的是其它方面的东西。

如果你的公司里有千人开发团队，而刚好你决定要为 PHP 里的静态类型开发自己的语法，又或者你正从 Haskell 转向 JS，那么一定程度的严格和纯净是非常有用的。不过大部分公司不会有那么大规模的开发团队，也不会有 Facebook 那样的宏大目标。下面我会更详细地解释这一点。

JSX 糟糕透了

我知道，它“不过是具有特殊语法的 javascript 罢了”。我们的前端设计人员为了做出漂亮的表单，把表单元素放置在 div 里面，他们根本不关心什么纯净或 ES6。设计 React 组件仍然需要耗费大量的时间，因为 JSX 的可读性太差。还有一个不好的地方，就是你无法在 HTML 代码里使用普通的 IF 语句。React 的忠实用户会告诉你说，有了三元运算，就不

需要再使用条件判断了。不过我向你们保证，当你再次阅读或编辑这些代码时，你会发现它们仍然是 HTML 和 JS 的混合体，尽管它们可以被编译成纯粹的 JS。

```
<ul>
        {items.map(item =>
          <li key={item.id}>{item.name}</li>
        )}
</ul>
```

很多开发者认为这种严格的限制可以帮助我们写出更加模块化的代码，因为我们必须把代码块放到工具函数里，并在 render() 方法里调用，就像这个人建议的那样：http://stackoverflow.com/a/38231866/1132016。

JSX 甚至让我们不得不把 15 行的 HTML 代码分成 3 个组件，每个组件里包含 5 行代码。

不要认为这种做法会让你成为更好的开发人员，你只是不得不这么做而已。

而事实其实是这样的：如果你正在开发一个相对复杂的组件，而你并不打算明天就把它发布到 GitHub 上，那么上述的方式只会拖你的后腿，特别是在你要解决真实的业务问题时。不过不要误会，我并不是说拆分成小组件就一定不好。

你当然清楚通过拆分可以提升代码的可管理性和可重用性。但前提是，只有当业务逻辑实体可以被放到一个单独的组件里时才要这么做，而不是每写一个三元操作代码就要进行拆分！每次在创建新组件时都会让你和你的意识流付出一定的代价，因为你需要从业务思维（在你记住当前组件状态时，需要增加一些 HTML 代码让它运行起来）转换到“管理思维”——你需要为这个组件创建单独的文件，考虑如何给新组件添加属性，并把它

们跟组件状态映射起来，还要考虑如何把回调函数传递进去，等等。

你被迫使用这种过度且不成熟的组件模块化方式来编写代码，从而降低了编码速度，而从中得到的模块化可能并非你所需要的。在我看来，不成熟的模块化跟不成熟的优化没有什么两样。

对于我和我的团队来说，代码的可读性是非常重要的，不过是否能够从编码中获得乐趣也很重要。为了实现一个简单的计算器控件而去创建六个组件，这样的事情一点也不有趣。大多数情况下，这样做也不利于维护、修改或控件检修，因为你要在很多文件和函数间跳来跳去，逐个检查每一个 HTML 小代码块。再次强调，我并不是在建议使用单体，我只是建议在日常开发当中使用组件来替代微组件。这是常识性问题。

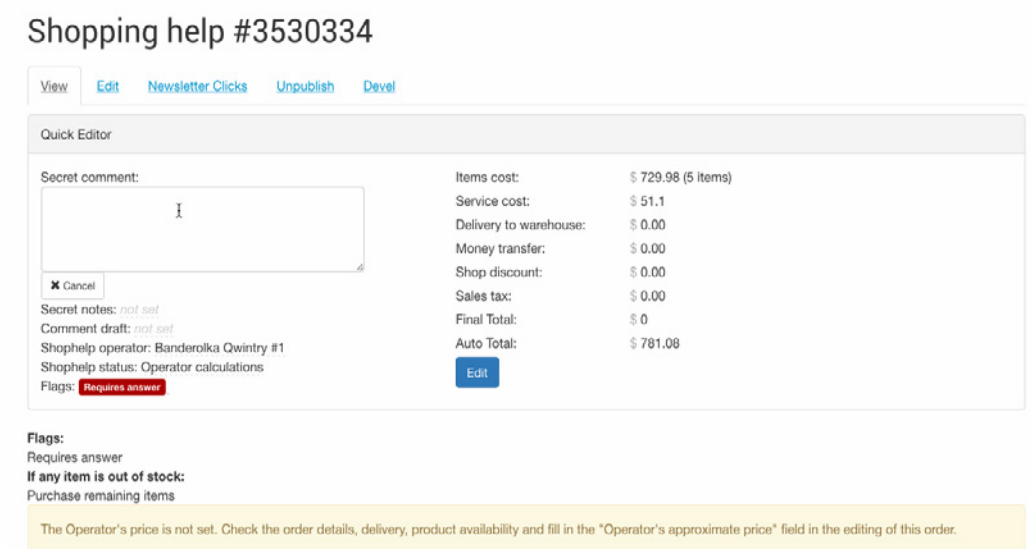
在 React 里使用表单和 Redux 会让你忙得停不下来

还记得吗，React 的设计在于它的纯净以及干净的单向数据流。这就是为什么 LinkedStateMixin 不受待见，你需要为 10 个输入创建 10 个函数，而 80% 这样的函数只包含了一行 this.setState() 代码，或者一次 redux 调用（或许你还需要为每个输入创建一个常量）。如果只要在脑子里想想就能自动生成这些代码的话，或许还是可以接受的，但现在还没有哪个 IDE 可以提供这样的功能。

为什么要敲这么多的代码呢？因为双向绑定被认为是不安全的，特别是在大型应用里。我可以肯定地说，双向数据流的代码可读性确实不是很好，而 Angular 1 的双向绑定更是糟糕透顶。不过，这些还算不上大问题。

最近我用 Vue.js 为 Drupal 网站开发了一组快速编辑器组件。

由于一些众所周知的原因，我不能把代码分享出来，不过我可以说使用 Vue 真的很有趣，而且代码可读性很好。而且我可以肯定地说，在



React 里开发这种控件，为每个输入创建一个单独的函数，我一定会感到很痛苦。

Redux 看起来更像是啰嗦的代名词。开发人员抱怨 Mobx 把 React 变成了 Angular，就因为它使用了双向绑定——可以参见之前讲到的第一点。似乎很多聪明人只是让他们的代码看起来更纯净，但是并没有完成更多的事情（不过如果你没有截止期限或许问题不大）。

过度的工具绑定

React 有点乱糟糟的。如果离开了一大堆 npm 包和 ES5 编译器，要做出 React 应用简直是寸步难行。基于 React 官方基础包开发的一个简单应用在 node_modules 目录下包含了大约 75MB 的 JS 代码。

这不算什么大问题，它更像是 JS 世界的事情，不过使用 React 仍然只会增加我们的挫折感。

Angular 1：太多的自由有时候不是好事

相比 React 所强调的所谓 JS 纯净性和代码可读性，Angular 1 算得上一款优秀的前端框架。Angular 1 可以帮助我们快速进入开发，在代码

的头一千行，我们会感到很有趣，但在那之后，代码开始变得糟糕起来。你会迷失在指令和作用域里，而且层间的双向数据流会变得像蛋糕上的樱桃一样，那些新来的开发人员甚至都不想去触碰一下，因为它们太难以管理了。

为什么会这样？

Angular.js 是在 2009 年出现的，那个时候前端世界还很单纯，甚至没有人会想到状态问题。不过我们不能抱怨这些人，他们只是想创造出一个框架，并成为 Backbone 的竞争对手，他们赋予它一些新的概念，并且可以少敲一些代码。

Angular 2

我们只是创建了 hello world 应用，它就生成了很多相关文件。你需要使用 Typescript 和编译器才能开始工作。这些对于我来说已经够了……在开始真正的开发工作之前，我还是觉得需要敲的代码太多了。在我看来，Angular 2 团队只是试图构建一个可以完美击败 React 的框架，而不是试图为一般的用户解决业务上的问题。或许是我想错了，或许我会改变想法。不过我还没有太多使用 Angular 2 的经验，我们只是构建了一个计算器演示应用作为内部的评估。Vue.js 网站上有一个很精彩的关于 Angular 2 和 Vue.js 之间的比较，这两者之间共享了很多设计上的概念。

Vue.js

简而言之，Vue.js 是一个我等待了很久的框架（我以后会讨论 Vue.js 2，相对第一个 Vue 版本，它做了很多改进，目前所说的是第一个稳定版本）。对于我来说，它优雅而简洁，并把注意力集中在解决问题上。Vue.js 是继 2007 年 jQuery 出现之后 JS 世界最大的一次改变。

如果你看过 Vue.js 的热度图，你会发现不止我一个人这么认为：http://www.timqian.com/star-history/#vuejs/vue&facebook/react。

Vue.js 是 2016 年发展最快的 JS 框架之一，而且我认为它的崛起并不是因为粉丝的过度追捧，也不是因为某个大公司的权威推动。

Laravel 把 Vue.js 加入到它的核心组件，这算得上是一件大事。

Vue.js 的优势

Vue.js 在可读性、可维护性和趣味性之间做到了很好的平衡。Vue.js 处在 React 和 Angular 1 之间，而且如果你有仔细看 Vue 的指南，就会发现 Vue.js 从其它框架借鉴了很多设计理念。

Vue.js 从 React 那里借鉴了组件化、prop、单向数据流、性能、虚拟渲染，并意识到状态管理的重要性。

Vue.js 从 Angular 那里借鉴了模板，并赋予了更好的语法，以及双向数据绑定（在单个组件里）。

从我们团队使用 Vue.js 的情况来看，Vue.js 使用起来很简单。它不强制使用某种编译器，所以你完全可以在遗留代码里使用 Vue，并对之前乱糟糟的 jQuery 代码进行改造。

恰到好处的神奇

Vue.js 可以很好地与 HTML 和 JS 一起协作。你可以开发出非常复杂的模板，而不会影响你对业务的专注，而且这些模板一般都具有很好的可读性。当模板膨胀到很大的时候，说明你在业务实现方面已经取得进展，这个时候你或许想把模板拆分成更小的组件。相比项目启动之初，此时你对应用的整体“映像”会有更好的把握。

从我的经验来看，这个跟在 React 里不太一样：Vue.js 帮我节省了

很多时间。在 React 里，在一开始就要把组件拆分成微组件和微函数，否则你会很容易迷失在乱糟糟的代码里。在 React 里，你需要花很多时间在一次又一次的整理 prop 和重构微组件（这些组件可能永远都不会被重用）上面，因为如果不这么做，在修改应用逻辑时就看不清方向。

在 Vue 里面使用表单是件轻而易举的事情。这个时候双向绑定就会派上用场。就算是在复杂的场景里也不会出现问题，不过 watcher 乍一看会让人想起 Angular 1。在你拆分组件的时候，会经常用到单向数据流和回调传递。

如果你需要用到编译器的一些特性、lint、PostCSS 和 ES6，你会如愿以偿。在 Vue.js 2 里，Vue 的扩展特性将会成为开发公共组件的默认方式。顺便提一下，开箱即用的组件 CSS 看起来是个好东西，它们可以减少对 CSS 层级命名和 BEM 的依赖。

Vue.js 的核心具有简单有效的状态和 prop 管理机制，它的 data() 和 props() 方法在实际当中可以有效地工作。通过 Vuex 可以实现更好的关注点分离（在我看来它跟 React 里的 Mobx 有点类似，都包含了部分可变状态）。

我认为大部分 Vue.js 场景都不需要 Vuex 提供的状态管理，不过多一个选择总不是坏事。

Vue.js 的不足

最大的一个问题：模板的运行时错误描述不够直观，这个跟 Angular 1 有点类似。Vue.js 为 JS 代码提供了很多有用的警告信息，例如当你试图改变 prop 或不恰当地使用 data() 方法时，它会给出警告。这也是从 React 借鉴过来的比较好的方面。但对模板的运行时错误处理仍然是 Vue

的一个弱项，它的异常堆栈信息总是指向 Vue.js 的内部方法，不够直观。

这个框架还很年轻，还没有稳定的社区组件。大部分组件是为 Vue.js 1 创建的，对于新手来说有时候难以区分它们的版本。不过你可以在不使用其它第三方库的前提下在 Vue 里面完成很多事情，你可能需要一些 ajax 库（如果你不关心同构应用，可以考虑 vue-resource）或者 vue-router，这在一定程度上平衡了 Vue 在组件方面存在的不足。

社区软件包里的代码有很多中文注释，这一点也不奇怪，因为 Vue.js 在中国很流行（它的作者就是个中国人）。

Vue.js 是由一个人维护的项目，这个也算不上大问题，不过还是要考虑其它一些因素。尤雨溪是 Vue 的作者，他曾经在 Google 和 Meteor 工作，在那之后他创建了 Vue。Laravel 也曾经是一个单人项目，不过后来也很成功，但谁知道呢……

在 Drupal 中使用 Vue.js

首先声明一下，我们不打算在 Qwintry 里使用 Drupal 8，因为我们正在转向使用更快跟简单的 PHP 和 Node.js 架构，况且我们的遗留代码是基于 Drupal 7。

因为我们的遗留系统 qwintry.com 是基于 Drupal 开发的，所以有必要在 Drupal 里对 Vue.js 进行测试。对于遗留代码，我并不引以为豪，不过它们能够正常运行，并为我们带来价值，所以我们尊重它们，并对它们进行改进，还增加了很多新的特性。以下列出了我们在 Vue 和 Drupal 里所做的事情：

就地编辑复杂的订单节点对象。这个功能包括为客户生成票据以及快速对商品进行编辑。它要求提供基本的 JSON API 来加载和保存节点数据，

没有什么特别的，就是一些菜单回调函数。

为我们正在使用的 SaaS 软件系统提供了两个基于 REST 的仪表盘，有了这两个仪表盘，我们的客服就不需要登录到不同的站点去检查客户相关信息，他们可以直接从仪表盘上看到这些。

我知道还有很多后端开发人员被困在 Drupal 7 的 Ajax 系统里，仍然停留在 2010 年的水平。

我知道，当你试图使用 Drupal 的核心特性来构建多步 Ajax 交互表单时，事情会变得多么复杂，代码会变得多么的难以维护。是的，罪魁祸首就是 ctools_wizard_multistep_form() 和 Ajax render！

Drupal 的开发人员同时面临着构建现代 UI 的挑战，但现代 JS 框架的复杂性又让他们望而却步。我在一年前就是这样的。我可以告诉你，现在正是使用 Vue.js 来改善 UI 的最佳时机，把 Vue.js 的代码库放到 /sites/all/libraries 里，通过 drupal_add_js 把它们添加到模板里，然后开始改造吧。你会惊奇地发现，在客户端（包括表单）使用 Vue，会让 hook_menu 里的 JSON 回调变得相当易于维护。

在 Yii 2 里使用 Vue.js

一个有趣的事实：Yii 是由一个叫作薛强的中国人创建的，所以 Yii+Vue 技术栈的发音并不难，它们是中国式的技术栈 :)

对于新版的 Qwintry.com（还没有公开发布），我们选择 Yii 2，我们相信它是最好也是最快的 PHP 框架之一。它当然不如 Laravel 流行，但我们用得很开心（不过我们也研究过 Laravel，他们确实做得不错）。

我们将逐渐减少 Yii 2 和 PHP 生成的 HTML 代码量，而在 REST 后端生成越来越多的 JSON，这些 JSON 是为 VueJS 客户端生成的。对于我们的

Active Record 模型，我们遵循的是 API 先行原则。

我们非常重视 API，这也是为什么我们花了很多时间完善 API 文档，虽然它们只是在内部使用。

使用 PHP 7 和最新的 MySQL 数据库，Yii 2 JSON 后端的响应速度几乎跟 Node.js 没有什么区别（差别也就是 15 到 20 毫秒），这对我们来说足够了，更何况它比 Drupal 要快上 10 到 20 倍。这是一直以来 PHP 跟其它第三发库最好的整合方式，足以保证我们手头代码的稳定。

总的来说，Yii 2 和 Vue.js 的结合所带来的响应速度是很快的，而且我们也很高兴基于这两个框架开发代码。

我们还在内部的很多项目里使用了 Vue.js。

结论

我们在三个月的时间里使用 Vue.js 为不同的项目开发了很多代码，结果也很令人满意。三个月对于后端来说也许算不上什么，但在 JS 世界里，它举足轻重。我们将会关注后续的进展。如果尤雨溪走对了方向的话，我期待着 Vue 会在 16 到 24 个月之后会变成主要的 JS 框架，至少对于小型的前端团队来说是这样的。不过我认为 React 仍然会是 2017 的主要 JS 框架，特别是如果 React Native 能够以之前的速度改进并成熟起来。

StuQ 斯达克学院

知乎 LIVE

斯达克学院
【StuQ公开课】
2017全新上线

联手知乎LIVE与众专家
畅聊技术直播轮番听

「StuQ公开课」为斯达克学院联合知乎Live，在2017推出的程序员技能提升直播栏目，公开课将充分挖掘极客邦各业务线（InfoQ、EGO、StuQ）优质技术专家资源，与知乎Live平台深度合作，为技术人提供学习指导、成长规划、技术实战干货等职业发展必备软、硬技能分享。

每周将邀请不少于两位专家大咖，围绕热点技术、实战案例、职场软技能培养、职业成长规划等涵盖多技术领域技术内容的主题分享。

每次分享时长将在1小时至1个半小时间。直播开始前+过程中，可随时以文字方式提问，专家会针对具体问题做出回答或给出指导意见。

1月你将听到……

- 1月12日 谢孟军 《开源如何影响程序员？》
- 1月13日 安晓辉 《程序员的精进：你适合做开发吗？》
- 1月17日 列旭松 《深入PHP扩展开发》

每次live直播后的内容，会异步整理发布到StuQ公众号。
详细了解直播专家个人简介及直播大纲，识别二维码关注「StuQ」公众号回复，还有源源不断的live更新静候聆听。

www.stuq.org

扫码了解更多

InfoQ 中文站 2017迷你书

架构师 月刊 2016年12月

本期主要内容：开源搜索引擎 Elasticsearch 5.0 版本正式发布；大数据框架对比：Hadoop、Storm、Samza、Spark 和 Flink；奇谈怪论：从容器想到去 IOE、去库存和独角兽；Web 不是未来会赢，而是已经赢了；亿级用户 PC 主站的 PHP7 升级实践

云生态专刊 09

《云生态专刊》是 InfoQ 为大家推出的一个新产品，目标是“打造中国最优质的云生态媒体”。

顶尖技术团队访谈录 第七季

本次的《中国顶尖技术团队访谈录》·第七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

架构师特刊 Apache Kylin实践

值此 Apache Kylin 开源两年之际，InfoQ 将之前发表的 Kylin 相关的文章集结成册，向社会发行电子书。