促进软件开发领域知识与创新的传播

架构师

ARCHITECT

热点 | Hot

如何在云平台构建大规模分布式系统

观点 | Opinion

云计算时代让整个IT业呜咽

推荐文章 | Article

携程移动App架构优化之旅

微信摇一摇背后的技术细节

特别专栏 | Column

在线支付之风控系统架构选型

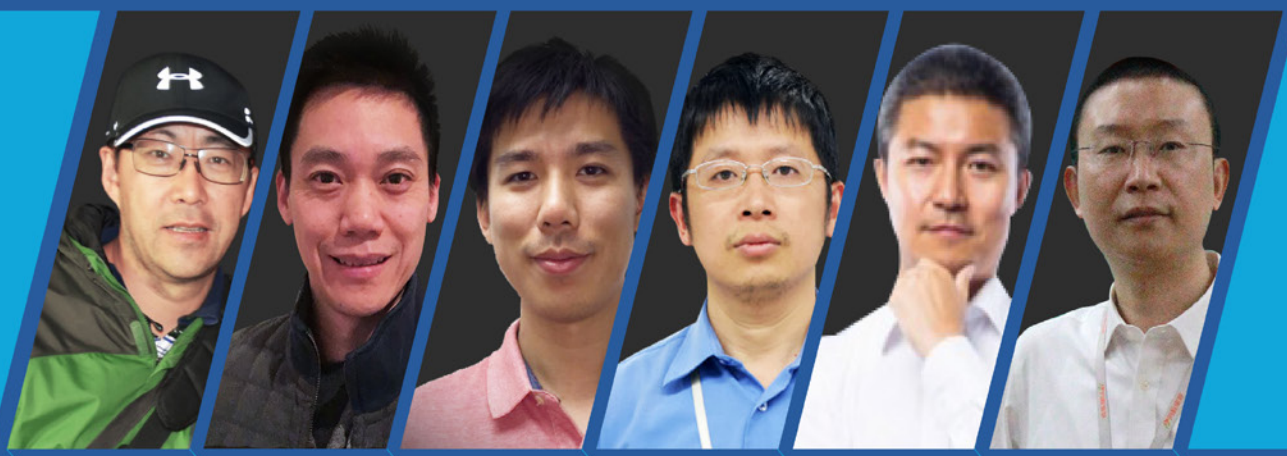
Geekbang 极客邦科技 | InfoQ

2016年5月

ArchSummit
全球架构师峰会 2016

最好的架构师都在这里

2016年7月15-16日　深圳 · 华侨城洲际大酒店



郭东白
阿里巴巴
速卖通
技术部总监

魏凯
Uber高级软件
工程师 /
工程经理

郭晓江
Tech Lead,
Machine Learning
Infrastucture
team, Twitter Inc.

张雪峰
饿了么CTO
EGO会员

吴甘沙
驭势科技联合
创始人CEO

张海龙
同程网CTO
EGO会员

扫描二维码关注ArchSummit技术关注”公众号，定期
参与线上微课堂分享

主办方 Geekbang. 极客邦科技 InfoQ

卷首语：大师都是孤独的吗？

作者 **崔康**

几天前，京剧界梅派代表人物梅葆玖去世，在社会上引起一阵轰动，粉丝众多，显示出大师传承的影响力。

何谓大师？在某一领域开宗立派，引领潮流，对产业具有极大的影响力，在某种程度上是创新颠覆者，又难以被其他人颠覆。

大师并不是曲艺界独享的专利，在科技界同样是人才辈出。比如，陈天桥开创了网络游戏新模式，马云引领了电子商务行业，雷军搅局了移动互联网 ..... 技术人中，也不乏大师。比如 Martin Fowler，Richard Stallman，Linus Torvalds。

如果仔细分析这些业界领袖的影响力，你就会发现他们的背后都是一个庞大的社区和一众狂热的粉丝。这就是影响力的真正所在。不论是梅派京剧、德云社、小米、还是 Linux，甭管你喜不喜欢，他们的品牌都是建立在自身基础和粉丝社群之上，两者互相促进，缺一不可。

有人说，大师都是孤独的，理解其心意者少。某些技术人也借此作为特立独行、不闻窗外事的理由，我不是非常认可这种观点和做法。孤独的大师也有，像梵高这种去世以后才火起来的艺术家，不应该成为现代社会的主流趋势。我有幸接触过不少国内外科技界的大师或者准大师，大多数人给我的印象是平易近人，有感染力、凝聚力，善于与人沟通，宣传和组织能力都不错。

在社群经济的年代，大师和粉丝完整的统一体。对于技术人来说，成为大师，除了职业技能的提升，还有很多重要的功夫要做。

你可以选择成为一个明星大师。

CONTENTS / 目录

热点 | Hot

如何在云平台构建大规模分布式系统

推荐文章 | Article

携程移动 App 架构优化之旅

10 亿红包从天而降，揭秘微信摇一摇背后的技术细节

观点 | Opinion

云计算时代来了，没有狂欢盛宴只有整个 IT 业的呜咽

特别专栏 | Column

在线支付之风控系统架构选型

架构师

2016 年 5 月刊

本期主编　杜小芳

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

亚马逊 aws

Geekbang. 极客邦科技 | InfoQ

InfoQ在线课堂

无服务器架构实践：如何利用 AWS Lambda 创建应用

时间

2016年5月24日（周二） 20:00–21:00

技术 干货
活动大本营
Geekbang
极客邦科技
InfoQ
免费 大会 课程 沙龙

如何在云平台构建大规模分布式系统

作者 王煜

【编者按】在云计算环境中，当用户业务面对流量激增、数据量翻番、访问量指数级攀升的“烦恼”时，如何利用云计算平台的弹性，结合业务自身特点，设计和构建一个高可用、高伸缩性的后端系统架构？本文以真实案例为背景，讲述从简单后端系统到大规模分布式系统的演进之路。

光快速扩容是不够的

很多企业和开发者在开发一款产品时，首要考虑的是产品功能的实现，其后端架构通常都是非常简单直接的。产品在刚上线初期，由于用户访问压力很小，数据量积累也并不大，在短时间内都会运行良好。

然而如今移动互联网的普及和成熟，让一款产品很可能在短时间内聚集大量用户，面对流量激增、数据量翻番、访问量指数级攀升等诸多“烦恼”，这本来是一件好事，可是如果后端系统不能及时扩展，势必会造成响应缓慢，频繁出错甚至拒绝服务的情况。

即便没有上述系统压力突然增大的“烦恼”，产品在不断开发升级的过程中，各种功能模块会变的越来越复杂，如果不能很好的梳理和组织后端架构，系统出错崩溃、不可使用的风险也会越来越大。

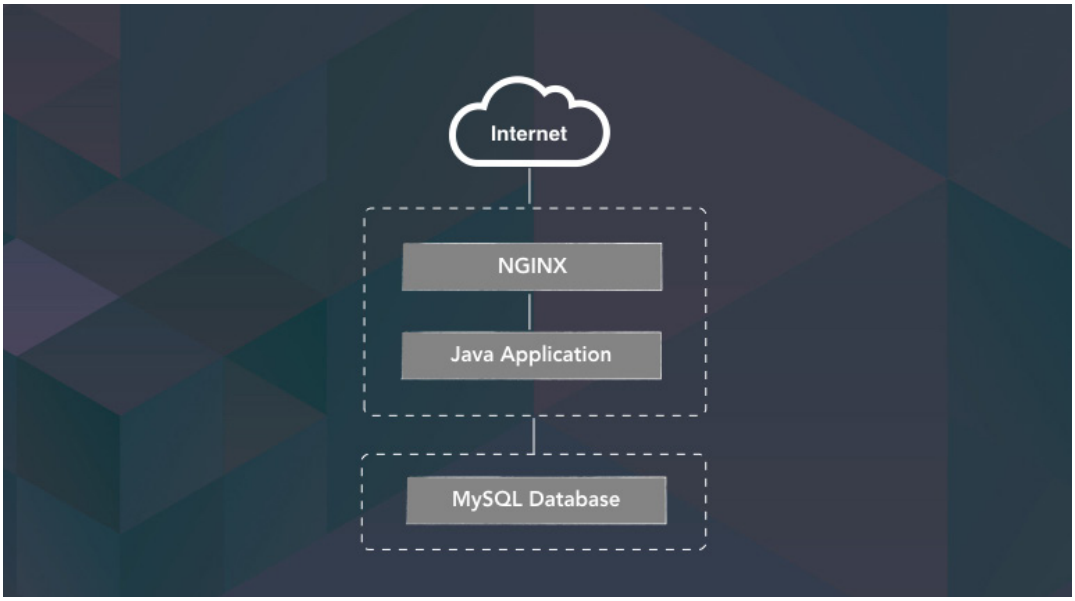


图 1

在没有云计算的时代，物理硬件从采购、上架、插线，到安装、调试、部署，再到真正投入使用，是一个漫长而耗费人力的过程，往往跟不上系统紧急扩容的节奏。而云服务的出现不仅仅让我们节约了使用成本，更重要的是可以利用云计算极度弹性的特点，让企业和开发者根据需求，对系统进行在线快速的扩容。

但仅仅在云服务上快速扩容是不够的，企业也需要在业务层面，关注各个系统组件的可用性和伸缩性。如何利用云计算的优势，结合企业的业务特点构建稳定可靠的分布式系统。

打造高性能、高可用的负载均衡集群

首先我们从一个最简单的后端架构开始（见图 1）。

在云计算环境中，网络架构的组织非常重要，QingCloud 提供了基础网络和 VPC 两种网络，他们的区别在官网用户指南中已经介绍，这里不赘述。推荐企业使用 VPC 来构建自己的网络，将所有主机和系统资源放置在 VPC 网络中，指定内网网段（如 192.168.x.x / 172.16.x.x），主

机可以通过内网地址进行通信，该地址不会变化。

随着主机越来越多，IP 地址不易记忆，为了方便主机间相互识别，可以给每台主机设置内网别名。为方便在控制台管理，给每个资源打上标签，按照标签来组织分类。

接下来我们回到上面那个简单的后端架构。随着访问压力越来越大，单台 Nginx + Java Application 可能不足以应付，你会看到这台主机的 CPU 越来越忙 Application，内存使用越来越多。而且这台主机一旦故障，整个服务都不可用了。

所以我们首先调整这里的结构，增加多台 Nignx + Java Application 同时提供服务，在接入层引入负载均衡器（下文用 LB 这个词代替），使外网请求首先发到 LB 上。LB 的选择有很多，比如提供七层负载能力的 Nginx 和 HAProxy，也有提供四层负载能力的 LVS，安装和配置的方法各有不同。

LB 的引入可以分摊请求压力到后端的多台业务服务器，并且可通过心跳检查，自动隔离后端出现故障的服务器，实现业务层的高可用。但这时 LB 本身也会成为一个单点，当出现故障也会导致全局不可用。所以可以使用 Keeplived 服务为 LB 提供一个副本，在一台出问题的时候可以马上顶上，部署方法网上有很多资料。

有人会说可以通过 DNS 轮询到不同的 IP ，实现 LB 的高可用，但事实上这样不行，因为一旦一台 LB 挂掉，DNS 还会解析到这个 LB，此时即便马上修改 DNS，在 DNS 缓存更新之前（通常要很久），服务也是不可用的。虽然 LB 的原理并不复杂，但是部署配置有很多工作量，而且为了实现 LB 的高可用还要额外做一些事情。

改造后的架构如图 2 所示。

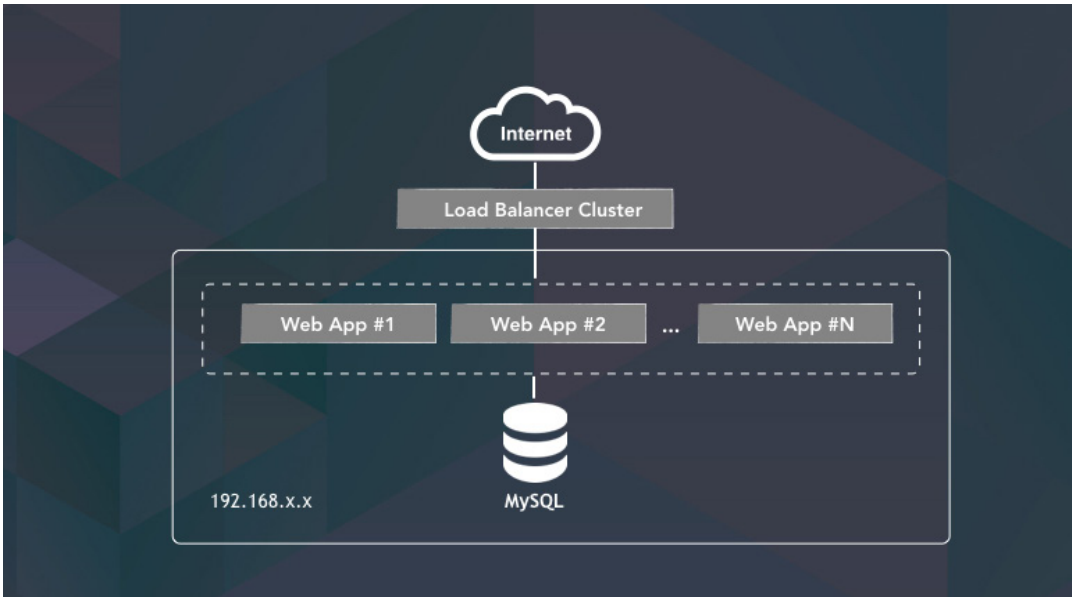


图 2

业务层扩展的问题

接下来我们来思考业务层的扩展问题。

首先要解决如何快速扩充业务服务器。如果业务服务器的运行环境和程序不会频繁更新，可以基于已有的业务服务器制作主机映像，当需要扩容时，直接基于映像创建新的主机，挂接到 LB 后端就可以马上对外服务了。

此时你还可以使用 AutoScaling 功能自动化这一过程，即当到达某种触发条件，如 LB 并发数、响应延迟达到多少后，自动触发主机的扩容。当触发条件不满足时，可以回收资源。

当然如果你的业务服务器的环境或程序需要频繁更新，不适合做成固定模版。此时可以自己搭建自动化部署（如 Puppet Application / Application Ansible）实现业务自动扩容（使用 QingCloud 的开放 API 接口可以轻松实现）。

此外你还需要保证业务服务器是无状态的，因为每次 LB 请求的后端

可能不同，不能假设上一次请求和这一次请求落在同一台业务服务器上。如果服务器需要保存用户访问的 session 信息，可将其下放到缓存或数据库中存储。

随着产品功能越来越丰富，你会发现原有单一的业务项目越来越庞大，各种功能逻辑交织在一起，当一个功能出现故障，可以引发全局不可用。此时你需要考虑将单一的业务项目分拆成多个独立子服务。子服务之间可以基于消息的通信，亦或基于 RPC 的通信方式。

子服务的调用可分为需同步处理和可异步处理两类。你应该尽量异步化所有不需要马上返回结果的请求。对于可异步处理的请求，我们通过引入消息队列，为请求产生的数据做缓冲，请求的接收者（队列消费者）可根据队列中任务的数量做水平扩容。消息队列的选择有很多，例如 Redis, RabbitMQ, ActiveMQ, Kafka，QingCloud 平台上目前已经提供分布式、可分区、多副本的消息队列服务，具有高吞吐量、低延迟等特点，用户可以方便的集成到自己的系统中。

如今数据分析对于企业越来越至关重要，业务服务器在处理请求的过程中，可以将原始数据通过队列，源源不断地导入大数据处理系统。用户可以根据需求方便的创建、使用和扩容 QingCloud 大数据分布式处理平台的 Spark 和 Hadoop 服务。

通过拆分子服务，使得我们有能力在某项子服务发生故障时，尽可能降低对于全局的影响，提高系统整体的可用性。另外，对于处理压力比较大的子服务，我们还可以进行独立的水平扩容。操作方式和前面讲到的业务服务器扩容相似，QingCloud 内网 LB 服务也可以在这里发挥作用。

改造后的架构如图 3 所示。

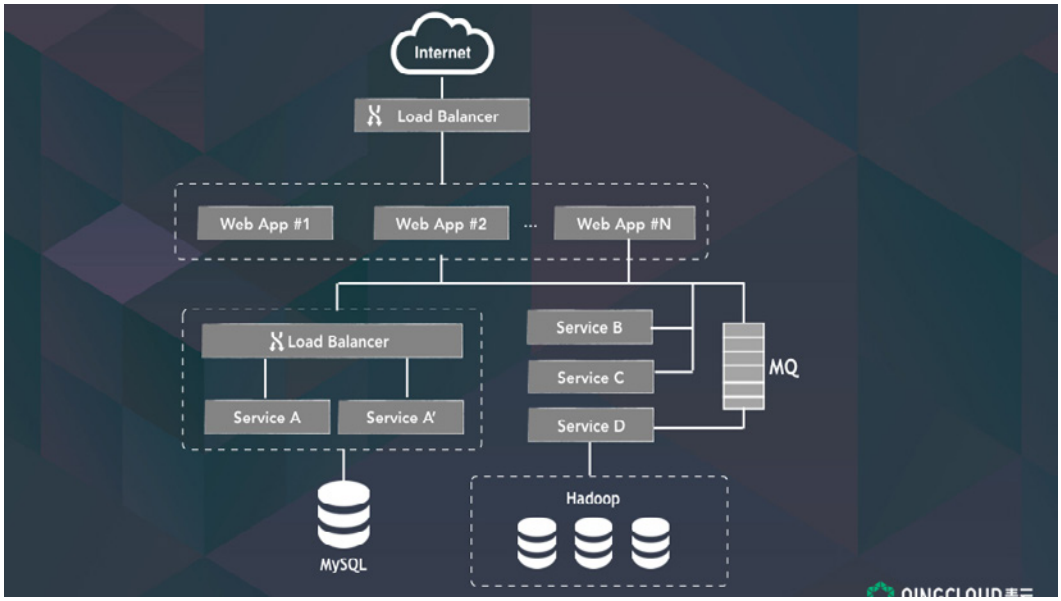


图 3

数据层的扩展问题

随着业务的增长，数据层面临的压力会越来越大，单机数据库已经不足以支撑。对于大多数的业务场景来说，数据的操作都是读多写少，而且读都集中在少部分的热点数据上。

我们首先想到的是引入缓存层来缓解数据库的读压力；如果缓存容量需求比较大，可以构建缓存集群，在上层按照 Consistent Hashing 算法将数据分散到多个节点，后续需要增加新缓存节点时，只有少部分的数据会失效。

接着引入新的数据库种类。Redis 已经成为诸多企业的首选标配，因为其支持丰富的数据类型和数据查询接口，且内存型的数据库天然具有更高的性能。

你可以将业务中关系性要求不高的数据，从 MySQL 转移到 Redis 中，尤其是列表类的数据以及计数统计类的数据。给 MySQL 减负的同时

提高数据的查询性能。如果单台 Redis 节点不能满足你对容量的需求，QingCloud 平台提供了支持多主多从 Redis 3.0 集群服务，一方面可对数据自动分区提高存储容量，另一方面保证了服务的高可用性。

对于 MySQL 的扩展可以分为几个步骤来做。

首先，增加 MySQL slave 节点，在上层将部分读请求分发到 slave 节点上去。由于 slave 同步可能有延时，业务应该能容忍短暂的数据不一致现象。例如你的一个用户修改了年龄属性，其他用户要等一会儿才能看到他的新年龄。QingCloud MySQL 数据库支持一主多从的架构，并且已经在多个从节点之上做好了负载均衡，你可以轻易在界面上操作增加新的从节点来为你分担读压力。

即便有 slave 作为数据副本，你也应该定期对你的数据库进行冷备份，方便当业务出现误操作时，能够回滚或恢复到曾经的某个时间点。在 QingCloud 平台上，备份的过程可以手动执行或者配置为自动任务，在备份过程中对数据库正常使用没有影响。

其次，随着数据的增长，单个数据库不能承载完整的数据集合，并且写操作对于单库的压力越来越明显，你应该考虑分库、分表技术。将比较庞大的数据表拆分出来单独存放，可以给主数据库腾出来一部分空间，分担读写压力。拆分的时候，还可以按照功能逻辑，把相关联的数据表存在一个库里。

第三，当数据库单表非常庞大，对读写都造成瓶颈时，你需要开始考虑水平分表 sharding。这种扩展方式可以同时解决单表容量过大，读压力和写压力很大的问题，但带来的研发和运维难度也会增大，推荐把上述的优化做完以后，最后在有必要的情况下再做这一步。

这里简略说一下水平分表的要点。首先要从数据表的字段中，选择一

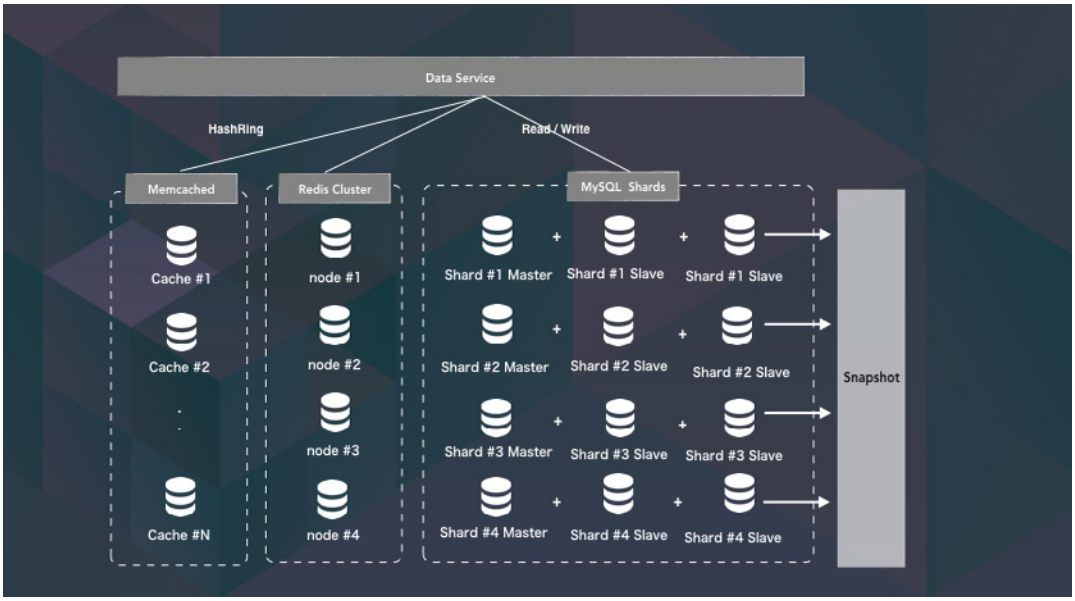


图 4

个合理的分区键（shard key），这个键应该是所有该表查询条件里，最经常用到的字段，这样才会使大部分的查询，能够提前判断应该向哪些特定的分区发送请求，如果查询条件中不带分区键，需要遍历所有的分区，并将结果进行 merge。

有了 shard key 还要设计一种分区算法。比如常见的有按照区间，如 user_id in [0, 100] 在 shard 1，user_id in [101, 200] 在 shard2，还比如按照 Hash 取模等等。设计分区算法的时候要充分考虑业务特点，多从读写操作的角度思考，这样设计能否将 I/O 压力和数据均匀分摊到每个 shard 上去。

最后，还需要考虑数据层的扩展如何对上层透明。比如引入分布式数据库中间件，或者结合业务逻辑把数据库操作做成一个独立的子服务，供其它服务调用。如果不做成子服务，至少在业务代码里有独立的一层来封装对数据库的操作。

至此，数据层的扩展示意图如图 4 所示。

除了上述的结构化数据的存取以外，企业还有存储海量小文件数据（非结构化数据）的需求。单机硬盘、LVM 和 NAS 可以作为临时方案使用，但都无法同时满足无限容量、高性能、高安全性、高可用性的多重需要。而自行搭建分布式存储系统，如 Ceph、GlusterFS、HDFS 适用场景非常有限，且运维和二次开发的成本也非常高。

在 QingCloud 平台上用户可以使用 QingStor 对象存储服务来存储海量的数据文件，服务本身提供了无限容量、高扩展性、高可用性和高安全性的特性。

多活与灾备问题

讲完数据层的扩展技术，最后来谈一下多机房部署和异地容灾的话题。QingCloud 从北京 3 区机房开始，通过自营的骨干网光纤和多路环网技术，使得当机房出现网络故障时对用户无感知，在基础设施上保障了高可用性。但是用户的业务如果能够多机房部署，可以在分摊访问负载的同时加速区域访问，比如加速中国南北方的用户或者海外用户的访问。

如图 5 所示，若是有三个机房，中间是 QingCloud 北京 3 区机房，负责主营业务。左边是 QingCloud 亚太 1 区机房，主要服务亚太和海外的客户。这两个机房都使用了 QingCloud 私有网络（VPC）部署，通过 GRE 或 IPsec 加密隧道在网络上的互联互通。右边是你办公室的物理机房，IT 人员可以在这个环境下进行开发和办公。

在业务上实现异地多活时，通常从易到难有三个阶段。

第一，在备用机房搭建反向代理，用户请求到备用机房，请求直接被转向主机房，如果两机房有专线互联或延时很小，这样部署最为简单。

第二，两个机房同时部署业务服务器和缓存，由于大部分数据请求可

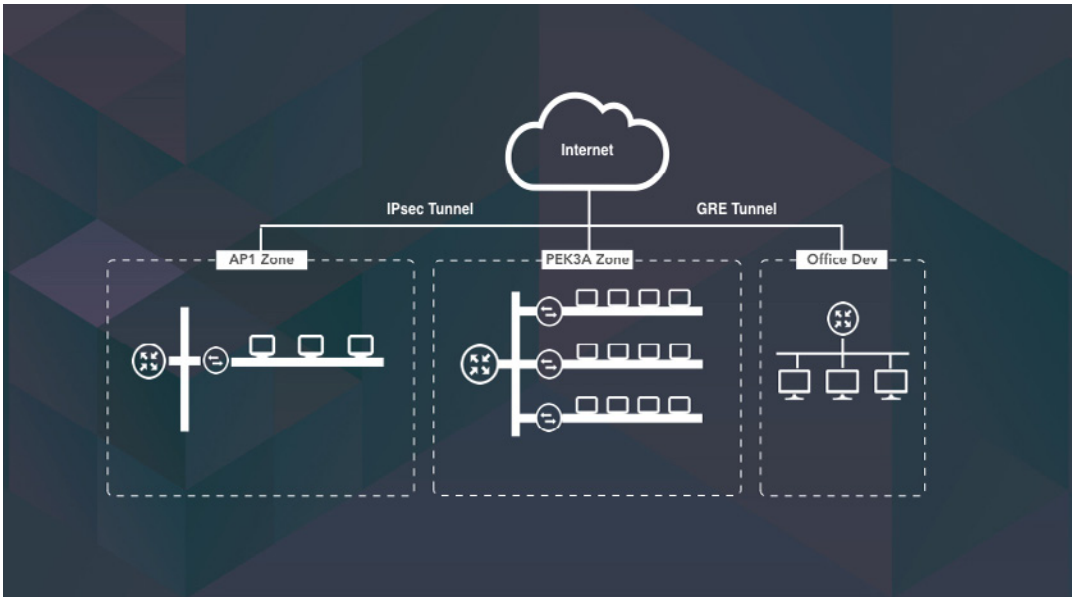


图 5

以从缓存中读取，不用进行跨机房访问。但当缓存失效时，依然要从主机房的数据库去查询。

第三，两机房同时部署全套系统，包括接入层、业务层和数据层。数据层依靠数据库双主或主从技术进行跨机房同步。

总结一下。没有一个所谓经典或完美的架构，只有最适合企业业务的架构，本文主要讲述在最通用的业务场景下，系统在接入层、业务层和数据层的常用扩展方法。企业后端架构的演进过程是一个漫长而艰巨的过程，不可能一蹴而就设计出一个万般周全的系统，但如果设计之初能更多着眼于未来，就可以为进一步优化留出了余地。

王煜，青云 QingCloud 系统研发工程师，原街旁团队创始成员，基础架构负责人，现负责 QingStor 对象存储服务的设计与研发。对 Linux 操作系统、计算机网络、分布式系统、云计算等领域有较深入的研究。九零前，文青程序员，代码诗人，北京土著。通信等行业成功实施并支持多个上百节点 Hadoop 集群。

Geekbang. 极客邦科技 | EGO EXTRA GEEKS' ORGANIZATION NETWORKS

高 端 / 社 交 / 人 脉 / 学 习 / 提 升

CTO 招募令

在EGO，与高端技术人一起 LEVEL UP!

EGO是高端技术人聚集和交流的组织，由InfoQ中国团队运营，旨在打造全球最具影响力的高端技术人学习和成长的平台。

扫描二维码，了解更多EGO相关内容

携程移动 App 架构优化之旅

作者 陈浩然

【编者按】本文为携程移动开发总监陈浩然在 2015 年 10 月份的 ArchSummit 全球架构师峰会上的演讲总结。由于面向受众为架构师，因此不会涉及到很多技术细节。通过本文，你可以了解携程通过哪些手段来优化它的 App 架构的。

“携程旅行 App”作为携程超级 App 产品，是公司全品类旅行产品的核心售卖入口，过去两年为了更好支撑无线业务的快速发展，携程移动 App 在产品和技术架构方面也做了大量的优化。

产品

产品方面，携程 App 从原先的 iPhone、iPad、Android Phone、Android Pad 和 Windows Phone 共五个版本精简为 Universial iOS 和 Universial Android 两个版本，以便于集中研发和市场资源发布新产品。

无线后端服务架构

无线后端服务架构方面，原有的无线架构（见图 1）是所有无线服务

图 1

耦合成一个统一的服务模块，囊括了所有提供给客户端访问的 API。

在统一服务模块功能较少的时候，这套架构完全符合业务功能需求，但当功能迭代加快，问题不断出现：

每个 API 都是 DLL 动态库形式（.net）实现，在某些公共逻辑接口发生变化时，不同时期上线的 API 可能使用了不同版本的逻辑接口定义，会导致运行时出现诡异结果或者进程 Crash，影响稳定性；

每次大版本发布上线，从测试到全面发布完毕，都需要全量回归测试过程，开发和测试进度严重耦合，影响发布效率；

新增的低优先级 API 可能稳定性不好，出现问题时会影响到整个服务进程的稳定性。

除了这种完全耦合的弊端之外，还存在其他诸如缺少负载均衡、越少监控、缺少熔断等影响后端稳定性的问题。于是我们开始尝试使用一种新的无线架构：Gateway（图 2）。

携程基于 Netflix 的开源项目 Zuul 开发了无线 Gateway（曾在 2015

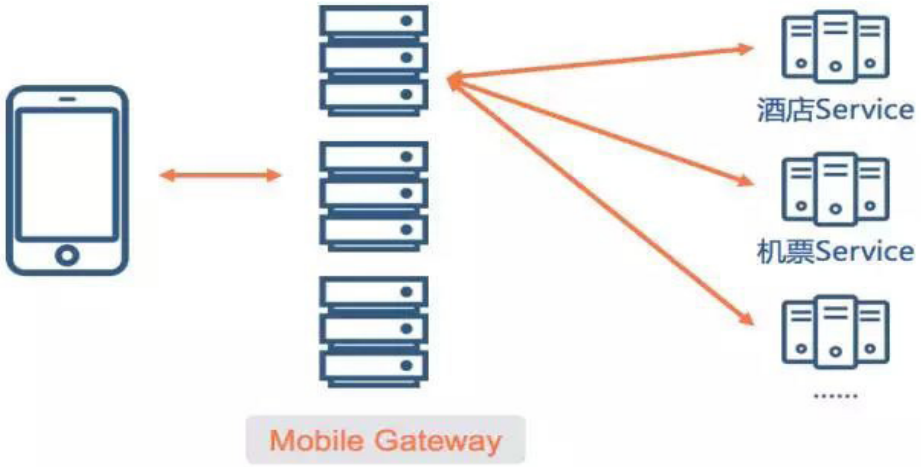


图 2

年 QCon 上海会议中分享）。Gateway 的职能是负责接收来自无线端的所有 API 请求，并将他们路由到正确的目标应用服务器，并且提供限流、隔离、熔断等功能，保证了无线服务的长期稳定运行，拥有的弹性容错机制也减少了日常运维工作。同时该 Gateway 提供了多维度的监控数据，并与报警系统对接，实时监控线上情况，达到运维自动化。

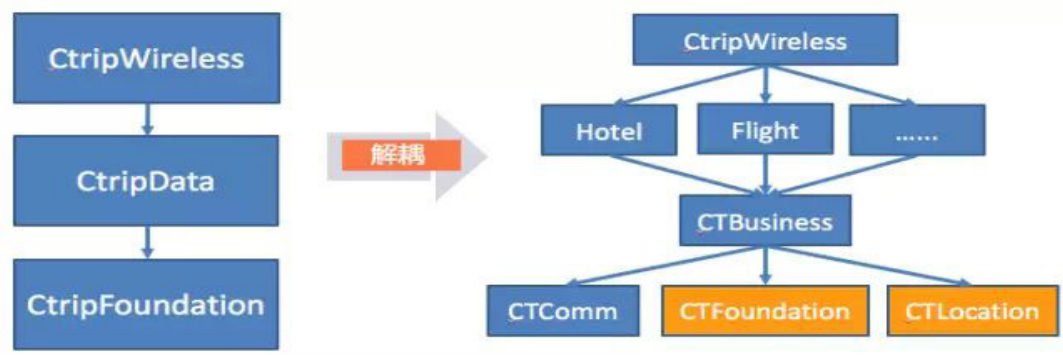
App 工程架构

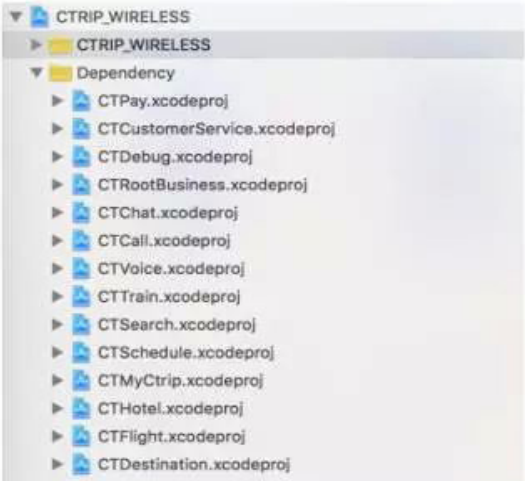
App 工程架构方面，原先的 App 开发还是单工程单构建的方式，各产品间的代码耦合严重，于是进行了 App 工程的解耦（图 3 和图 4）。

解耦后各产品间代码完全独立，相互页面和功能的跳转走数据总线或者 URL 总线方式来实现，打包时的资源问题可以通过 Run Script（iOS）和 Gradle（Android）来定制化解决。

在 App 工程架构解耦后，紧接着是两个问题：

- 开发框架是否满足产品开发需求？
- 性能和质量是否达到用户体验要求？





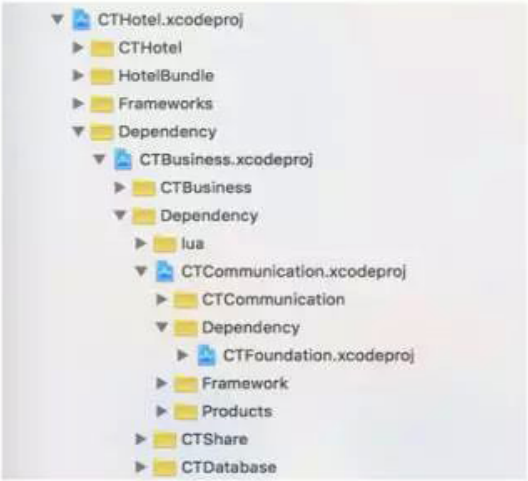


图 3 / 图 4

为了解决第一个问题，我们梳理了现有的 App 功能，将通讯、定位、Hybrid 框架、数据库、登录、分享、基础库等核心功能实现了 SDK 化，也提供给公司其他 App 直接使用；同时将 App 内部的公共业务组件例如地图、日历、城市、图片、通讯录等实现了统一。

要解决第二个问题『App 性能和质量』，首先需要了解 App 性能的现状，即 App 端性能的采集。常规做法有两种：1. 使用第三方性能采集 SDK，例如 OneAPM、听云等第三方工具；2. 自建。携程为了完整掌握用户数据采用了自建的方式：App 通过日志 SDK 采集日志，上传至日志服务端，日志消息经 Kafka 消息队列存入 HDFS（RCFile 格式）分布式文件存储系统，后期的数据查询均基于 Hive 系统来实现。

图 5

App 端的性能数据会在多种纬度（操作系统、App 版本、网络状况、位置）下采集网络（网络服务成功率、平均耗时、耗时分布等）、定位（经纬度成功率、城市定位成功率等）、启动时间、内存流量等各类指标。其中像网络服务性能是对于用户体验至关重要的端到端性能，也是优化的核心依据。

性能数据采集后需要采用简单直观的 Portal 进行展示，携程为无线业务开发了 Web 端和 App 端性能展示 Portal，图 5 是网络性能的截屏，数据会每小时进行更新聚合并展示。

基于完备的性能采集数据，很多 App 性能便可以依据数据进行迭代优化，例如 App 网络服务方面，携程 App 采用了以下多种优化手段：

- 使用 TCP 长连接实现网络服务，减少网络连接时间
- 根据网络状况 2G/3G/4G/WIFI 进行调优参数
- 根据连接 / 读 / 写不同阶段使用重试机制
- 使用 IP 列表避免 DNS 解析失败或者劫持，无需进行 DNS 解析
- 根据网络延迟选择服务端 IP（使用 Ping）

- 使用 ProtocolBuffer+Gzip 减少 Payload

经过这些优化手段，携程 App 的端到端网络服务成功率从最初很差的 95.32% 提升至 99.87%，用户体验得到明显提升。

携程 App 的 Hybrid 框架也是经过多个版本的迭代，已经支持强大的插件功能，已经做到凡是可以用 Native 的组件通通使用 Native 组件来优化 Hybrid 业务的体验。携程 Hybrid 框架在设计之初即采用了离线包功能：Hybrid 业务整体打包在 App 中，节省了用户打开页面时的资源加载时间；同时离线包支持查分增量更新，并通过 7z 压缩方式进一步降低了增量更新包的大小，相对 zip 压缩减少 30% 大小。

Native 的插件化和 HotFix

Native 的插件化和 HotFix 方面，我们在 iOS 端使用开源的解决方案 JSPatch，在 Android 端采用了自研的解决方案 DynamicAPK，可以支持各组件的资源及代码的更新。DynamicAPK 方案无需做任何 activity/fragment/resource 的 proxy 实现，使得原有的业务代码无需修改即可支持，迁移代码很小，同时可以提升 App 启动时间，详情请参考 GitHub。

其他优化都是针对特定的业务场景展开，例如 Android 的海外地图没有好的解决方案，iOS 的地图控件又存在人在国外看国外和人在国外看国内地图数据精度低的问题，携程基于 Google 地图开发了 Hybrid 版地图，用于在特定场景的业务需求；Hybrid 网络性能优化，通过 Hybrid 接口发送 Native 网络服务，可以避免 DNS 劫持并且利用 Native 端的 TCP 长连接；海外网络性能优化，通过 TCP 海外加速产品实现链路优化；图片性能优化方面使用了 WebP 图片格式，可以降低 30-40% 图片大小，图片分片上传等。

移动端技术发展很快，携程也在积极尝试新技术，例如 React

Native（已在账户信息部分页面使用），新的网络协议 SPDYHTTP/2.0，Apple/Huawei/Samsung Watch App 等都做了大量尝试，以期能够提升产品品质。

总之，App 架构的技术优化没有尽头，我们会继续以开发效率、性能、质量、新技术几个纬度不断推进，希望未来可以有更多内容分享给业内同行。

Geekbang. 极客邦科技 | EGO EXTRA GEEKS' ORGANIZATION NETWORKS

扫描二维码，了解峰会嘉宾讲师、议程安排及票务信息

截止5月25日前
购票可享受7折优惠
[双重优惠]两人同行，第二张半价

帮助深具远见卓识的[技术人] TECHNICIAN
审时度势 提升领导力

GTLC
GLOBAL
TECH LEADERSHIP
CONFERENCE

全球技术领导力峰会
BEIJING · CHINA

北京·国家会议中心
2016·08·29~30

10 亿红包从天而降 揭秘微信摇一摇背后的技术细节

作者 张文瑞、王鹏程

【编者按】与传统意义上的红包相比，近两年火起来的“红包”，似乎才是如今春节的一大重头戏。历经上千年时代传承与变迁，春节发红包早已成为历史沉淀的文化习俗，融入了民族的血脉。按照各家公布的数据，除夕全天微信用户红包总发送量达到 80.8 亿个，红包峰值收发量为 40.9 万个 / 秒。春晚直播期间讨论春晚的微博达到 5191 万条，网友互动量达到 1.15 亿，网友抢微博红包的总次数超过 8 亿次。

2016 年春节的三波红包雨——摇红包已经过去一段时间，面对亿级的企业资金以及亿级的红包，回想这其中的过程并不简单，充满各种变数，稍有闪失就可能功亏一篑。让我们一起回顾一下这里的准备过程，看看技术上是如何为这只许成功不许失败的项目保驾护航的。

一 . 除夕活动红包系统

红包系统由三部分组成：信息流、业务流和资金流。这三部分在组织架构上由不同的后台团队完成：信息流——微信后台，业务流——微信支付后台，资金流——财付通后台。

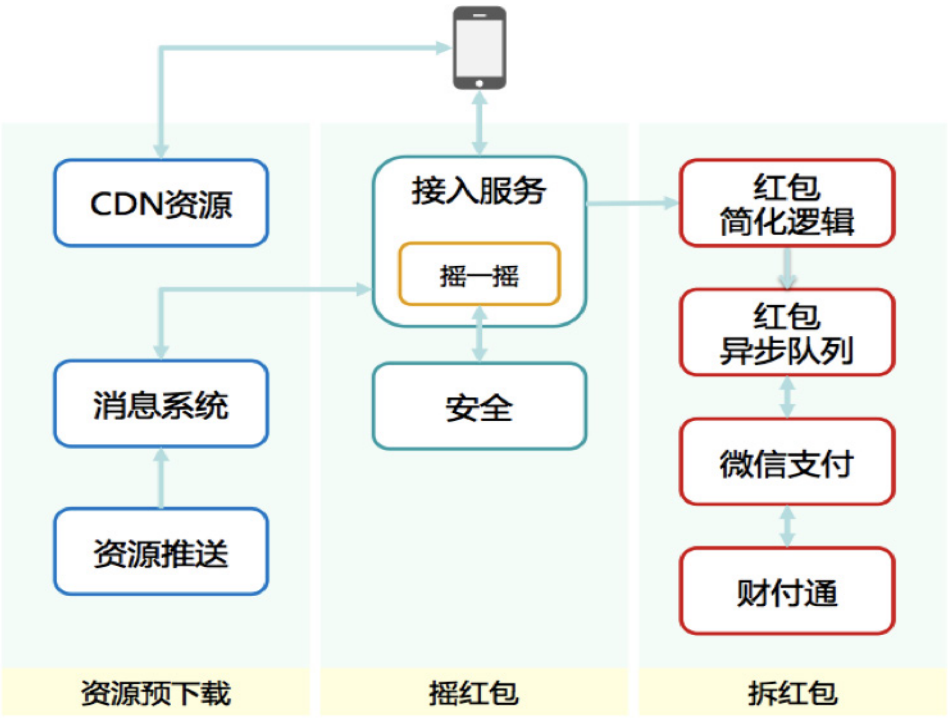


图 1

在平时，红包系统主要处理个人会话中以消息形式发出的红包。其中，信息流主要包括用户操作背后的请求通信和红包消息在不同用户和群中的流转；业务流是用户请求引发的包红包、抢红包和拆红包等的业务逻辑；资金流则是红包背后的资金转账和入账等流程。

红包系统在除夕活动时和平时的实现不大一样。在除夕活动时，除了个人红包外，红包系统还要处理由后台通过摇一摇集中下发的大量企业红包。这里边信息流的实现变化较大。

接下来简单介绍一下 2016 年除夕活动时的红包系统架构，包括三个方面：资源预下载、摇红包、拆红包（见图 1）。

1. 资源预下载

在除夕，用户通过摇一摇参与活动，可以摇到红包或其他活动页，这

些页面需要用到很多图片、视频或 H5 页面等资源。在活动期间，参与用户多，对资源的请求量很大，如果都通过实时在线访问，服务器的网络带宽会面临巨大压力，基本无法支撑；另外，资源的尺寸比较大，下载到手机需要较长时间，用户体验也会很差。因此，我们采用预先下载的方式，在活动开始前几天把资源推送给客户端，客户端在需要使用时直接从本地加载。

2 . 摇 / 拆红包

除夕的摇一摇子系统是专门为活动定制的，按时间轴进行各项活动，这里边最重要、同时也是请求量最大的是摇红包。从需求上看，系统需要完成两个事：用户可以通过摇一摇抢到红包，红包金额可以入到用户的支付账户。在除夕，系统需要在很短时间内将几十亿个红包发放下去，对性能和可用性要求很高。考虑到涉及资金的业务逻辑比较复杂，还有很多数据库事务处理，耗时会比较长，于是我们将抢红包（信息流）和红包的账务逻辑（业务流和资金流）异步化。将前一部分处理流程尽可能设计得轻量，让用户可以很快抢到红包，然后再异步完成剩下的账务逻辑。

那么，抢红包阶段是怎样做到既轻量又可靠呢？

1. 零 RPC 调用

在微信后台系统中，一般情况下客户端发起的请求都是通过接入服务转发给具体的业务服务处理的，会产生 RPC 调用。但对于摇一摇请求，我们将摇一摇逻辑直接嵌入接入服务中，接入服务可以直接处理摇一摇请求，派发红包。

2. 零数据库存储

按一般的系统实现，用户看到的红包在系统中是数据库中的数据记录，

抢红包就是找出可用的红包记录，将该记录标识为属于某个用户。在这种实现里，数据库是系统的瓶颈和主要成本开销。我们在这一过程完全不使用数据库，可以达到几个数量级的性能提升，同时可靠性有了更好的保障。

支付系统将所有需要下发的红包生成红包票据文件；

- 将红包票据文件拆分后放到每一个接入服务实例中；
- 接收到客户端发起摇一摇请求后，接入服务里的摇一摇逻辑拿出一个红包票据，在本地生成一个跟用户绑定的加密票据，下发给客户端；
- 客户端拿加密票据到后台拆红包，后台的红包简化服务通过本地计算即可验证红包，完成抢红包过程。

3. 异步化

用户抢到红包后不会同步进行后续的账务处理，请求会被放入红包异步队列，再通过异步队列转给微信支付后台，由微信支付后台完成后续业务逻辑。

二 . 大规模集群中保证数据一致性

事实上网络分裂很难从根本上避免，我们在设计系统时都是假设网络分裂一定会出现，基于此思考应该采用什么方案保障系统在网络分裂时能正常运作。

我们的方案是在每个数据中心都建设三个独立的数据园区，可以做到在任意一个数据园区出现网络分裂等故障，甚至彻底变成园区孤岛后，另外两个数据园区可以无损承接整个数据中心的请求。

三园区容灾的关键就是数据一致性。我们需要做到在分裂出去的那个数据园区的数据在其他园区有个强一致的副本，这样请求落到其他两个园

区后，可以无损完成服务，此外在故障园区恢复后，数据在所有园区还能自动保持强一致。微信后台实现了基于 Quorum 算法，对数据有强一致性保证的存储系统——KvSvr（这一存储系统另文介绍）。此外还有可以提供三园区强一致保证的可靠异步队列，这次就应用在这个红包系统中。前边提到的部署在接入服务的红包文件实际上也是可以实现三园区容灾的，我们在每台接入服务部署的红包文件都会在其他数据园区有个备份。在某个数据园区故障时，我们可以在其他数据园区发放故障园区的红包。

今年活动红包总量非常大，活动形式也更丰富，我们在以下方面做了优化。

1 . 服务性能

为提升各个服务模块的处理性能，我们通过压测和 Profiler 分析，发现了不少性能瓶颈点，做了大量优化。

2 . 业务支撑能力

支持更加复杂的业务场景，并在客户端和服务器都加入了很多可以后期灵活调整的预埋能力，以更好地服务产品运营。

3. 可用性

不断提升系统可用性是我们一直努力的目标，以下 5 点很好地提高了系统的可用性。

1. 系统容量评估与配额

对系统的容量需要有个准确的评估与验证，并结合业务设计合理的配额方案和降级方案，尽可能保障系统不会过载。例如，我们评估并验证完系统每秒最大拆红包量后，就可以在处理用户摇一摇请求时，限制系统每秒最大发放红包配额，这就间接保证了拆红包量不会超出处理能力。

2. 过载保护

服务如果出现过载了，必须有能力自保，不被压垮，并且不扩散到系统其他的服务。我们在后台的服务框架层面具备通用的过载保护能力：服务如果处理不过来，就按请求的优先级尽快丢掉超出处理能力的请求，保证服务的有效输出；上游调用端在部分服务实例过载时，能自动做负载均衡调整，将请求调整到负载较低的服务实例中；上游调用端发现大部分服务实例都出现过载，也可以主动丢掉部分请求，减轻后端服务器的负担。

3. 减少关键路径

减少核心用户体验所涉及的步骤和模块，集中力量保证关键路径的可用性，从而在整体上提高可用性。我们把活动红包的信息流和业务流进行异步化，就是基于这个考虑。跟用户核心体验相关的抢红包操作，在信息流中的接入服务、红包简化逻辑服务和红包异步队列（入队）这三个服务模块参与下即可完成。这三个服务模块是可以比较容易用较低成本就做到高可用的，可以较好地规避业务流和资金流中几十甚至上百个服务模块可能出现的风险。

4. 监控指标

我们需要对系统的真实负载情况有准确及时的了解，就必须要有一套高效、可靠的监控系统，同时还要有一套有效的监控指标，监控指标不是越多越好，太多了反而会影响判断，必须要有能准确反映问题的几个核心指标。在我们系统里，这些核心指标一般在基础框架集成，根据经验来看，其中一个很有用的指标是服务的最终系统失败。我们把服务的失败分为两类：逻辑失败和系统失败。系统失败一般是服务暂时不可用导致，是可通过重试来自动解决的，如果请求重试若干次仍然为系统失败，就产生最终系统失败。通过最终系统失败通常可以快速定位到异常的服务，及时处置。

5. 人工介入

我们在红包系统内预置了很多配置开关，当自动运作的过载保护无法发挥预期作用时，可以通过人工介入，使用这些保底的手动开关迅速降低负载、恢复服务。

三 . 技术创新

实际上，类似的这种活动用到的技术都是现成的，并不复杂。但为什么大家会觉得很难实现呢？主要是因为规模: 用户和请求的并发规模越大，就越难在系统的成本和可用性上达到平衡，也就是说越难实现一个低运营成本、高服务可用性的系统。

在传统的应对这种有很大规模用户参与的活动的实现方案里，一般会采用在客户端过滤请求，以某种概率（基于时间或互动次数）发到服务器进行处理，这样服务器的压力可以大幅降低。

我们认为还可以做得更好，在这种活动的技术方案上可以有所突破——在保持低成本的前提下，全量处理用户的每次交互。这就大大降低了客户端的实现风险（因为客户端的更新和覆盖周期相对较长）。此外，服务器有了对用户交互有了全面的控制能力和灵活调整的能力。

这些能力对活动的运营是非常可贵的。可以让我们在活动过程中，各种复杂用户场景下，都能做到精细的调整，给了产品运营很大的灵活性，以保证活动效果和用户体验。看看下面两个例子。

我们可以精确控制和调整每次用户交互出现什么结果，曝光哪个赞助商；

活动过程中，有个很难预估的因素——参与人数。说不准参与的用户少了（或互动次数少了），导致红包很久都发不完？或者参与的用户多了

（或互动次数多了），导致需要加快发放速度，更快速发完红包？

于是我们对这个技术方案做了全面的思考和设计，最终实现了现在这个系统，可以用很低的成本实现极高的性能和可用性，在除夕活动中得到了成功的应用。

四 . 服务降级方案

我们对摇一摇 / 朋友圈红包照片在 2016.1.26 的预热活动和 2016.2.7 的正式活动都做了详细的复盘。包括活动过程中各项业务数据是否符合预期，各个模块的表现是否符合预期等，并分析各种不符合预期表现的成因和解决措施。

在红包系统的信息流、业务流和资金流都有很多保障用户核心体验的降级方案。举几个信息流的降级方案的例子。

a） 如果某一个数据园区出现网络分裂等故障，完全不可用了，部署在那里的红包怎么发下去？

红包文件虽然在园区间有冗余存储，但基于性能和可用性考虑，我们并不打算在各园区间维护强一致的红包发放记录，做到记录级的“断点续发”，而是将红包文件按时段进行切分，降级为只做文件级的“断点续发”。在某个园区不可用时，使用降级方案后，故障园区当前发放时段的红包文件不会接着发放，仅保证下一时段的红包能通过其他园区正常发出去。

b）活动过程中，如果用户的交互量超过服务的处理能力了怎么办？

正如前面所述，我们很难准确估计参与用户量及互动次数。就本次活动而言，在系统设计之初，我们评估有 2000 万次 / 秒的峰值请求，并在系统最终实现和部署时预留了一定的余量，提供了评估值 2.5 倍（也就是 5000 万次 / 秒峰值请求）的系统处理能力，除夕当晚服务器处理请求峰

值达到 2500 万次 / 秒，服务实际负载低于 50%。但如果当时用户过多，互动过于火爆，到达后台的请求超过 5000 万次 / 秒，系统就会进入降级模式，客户端可以在服务器的控制下，减少请求，防止服务器过载。

c）红包发放速度过快，后端处理不过来怎么办？

正如前面所述，用户抢红包是在信息流完成的，完成后请求放到红包异步队列再转到业务流。如果领取红包请求量过大，队列会出现积压，红包的入账会出现延时，但不会导致用户请求失败。

类似的降级方案还有很多，每个环节都有若干个不同的降级方案，有些是针对业务专门设计的（如 a 和 b），还有些是使用基础组件 / 服务 / 方案本身就具备的降级能力（如 c）。这些方案都遵循着一个原则：降级时，尽可能保证用户的核心体验。

五．资金与红包准备的难点

除夕活动资金和红包准备的难点总体来说有以下 4 点。

1. 红包的数量

由于招商的不确定性，最终投入微信摇企业红包的资金不可准确预估；

不同的商户其对品牌的曝光量的需求不同，有的要求多曝光，有的要求多关注，红包金额或大或小，数量则或多或少；

从准备到除夕当天，期间各种变化，活动方案变数很多。

红包的数量可能从几亿到几百亿变化，资金与红包的准备需要能够满足需求的巨大变化。

2. 资金的就位

企业各行各业，营销经费的审批流程不尽相同，资金最终支付到位时

间前后差异很大，甚至有些在除夕前一周才会最终敲定支付到账；

某些企业可能在最后阶段停止合作；

某些企业可能在最后阶段调整资金的使用方式。

以上情况会导致资金的到位时间不完全可控，本着尽可能多的资金能够投入到活动中的原则，希望可以尽可能压缩活动的准备时间，让更多的资金能够上车。

3. 资金预算的配置方案（资金剧本）

按设想的活动方案，红包会分为三大类：图片红包、视频红包、品牌 logo 红包。活动方案较去年又有新的变化，特别是 logo 红包的认知度和接受度不确定，logo 红包过度会否导致未领完而浪费资金，不容易确定 logo 红包的资金占比；

除夕当天的活动剧本可能会反复调整优化，红包时段的划分可能修改，不同时段的资金投放量可能变化。

大笔资金、大量的红包、复杂的活动方案、善变的商户要求，都可能反复调整，如果面对百亿量级的红包配置调整时，技术上如何实现缩短准备时间，支持方便的变化。

4. 资金的安全

- 如何防止红包的金额被篡改；
- 如何防止未被领取的红包被入账给用户；
- 如何防止红包金额重复入账给用户；
- 如何防止机器被攻破产生了不存在的红包；
- 如何防止红包被不同用户重复领取；
- 如何在确保资金安全的情况下尽可能保证用户的到账体验（最好是实时或准实时到账）。

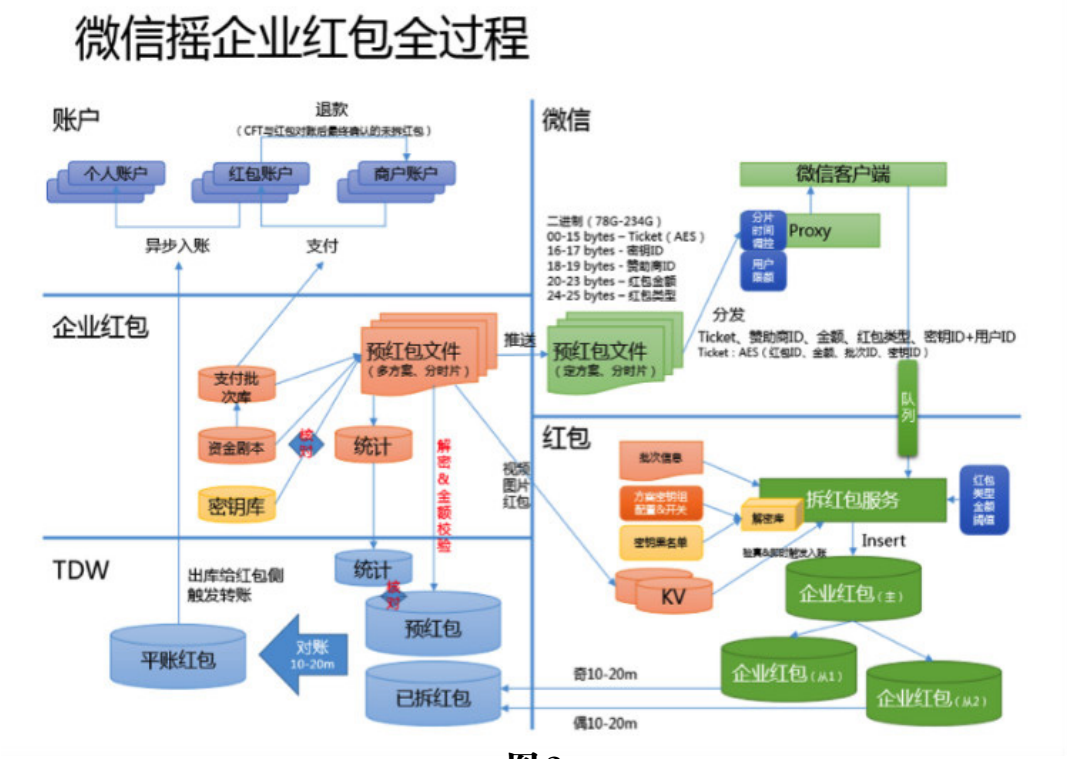


图 2

技术上必须做万无一失的准备，确保资金足够安全，活动顺利完成。

六．微信摇企业红包全过程

如果在除夕当天摇的过程中按前边提到的超级复杂的配置方案即时生成随机红包，这显然是风险齐高逻辑奇复杂的。对待只许成功不许失败的项目，主流程必须极简高效，所以这里全部的资金和红包数量都需要按方案规则提前切分和准备好。

将预生成好的红包数据（预红包数据）进行准确的部署，摇红包的资金和红包准备的整体流程方案有两个选择（见图 2）。

- 方案一：预红包数据提供部署给微信的接入机和写入红包 DB，摇红包过程由红包接入机控制红包的发放，拆红包时修改红包 DB 中的红包数据；

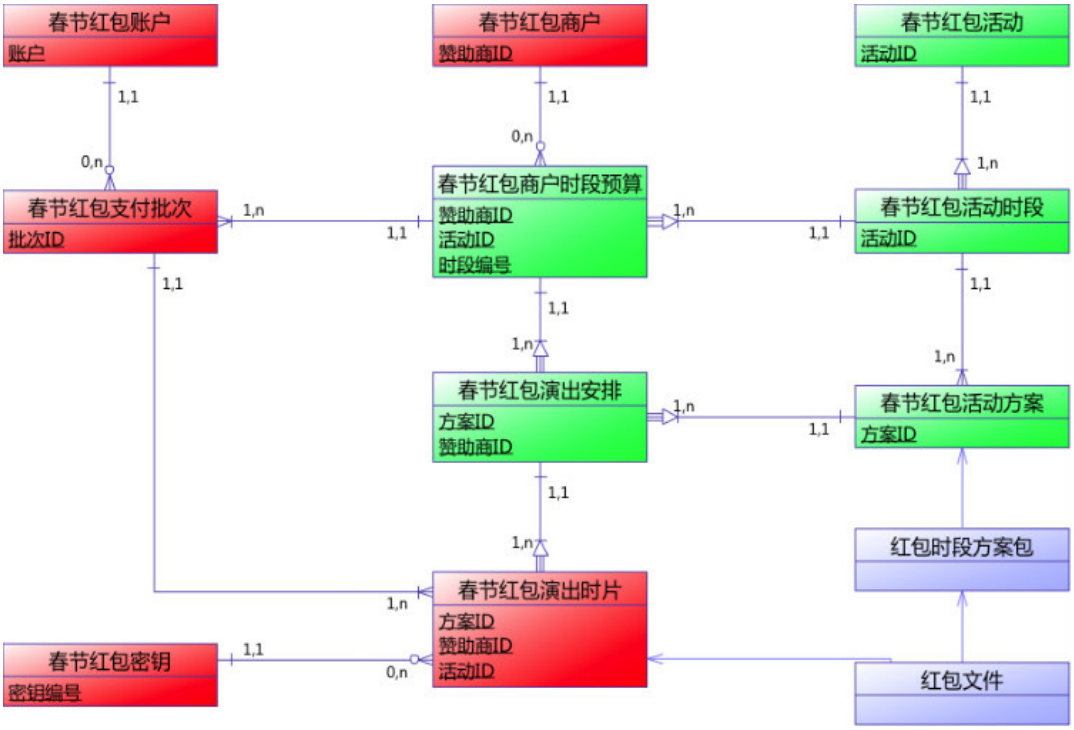


图 3

- 方案二：预红包数据只提供部署给微信接入机，摇红包过程由红包接入机控制红包的发放，拆红包直接 Insert 到红包 DB。

第二个方案减少一次 DB 操作，如果是百亿量级的红包数据，可以极大减少数据导入、对账等活动准备时间，特别是方案需要变更时。

七．充足准备

1. 对资金预算和资金剧本进行合理的建模

首先，面对如此大量的资金和复杂的资金剧本，如何准确高效地管理和控制逻辑呢？要杜绝傻大黑粗，我们需要一个优雅的解决方案——建模。（见图 3）

对资金预算和资金剧本进行合理的建模，让模型涵盖、掌管、控制一切——让工具基于模型进行自动化的处理和校验，这里需要做的就是：

落地该模型的设计，让一切围着模型转；

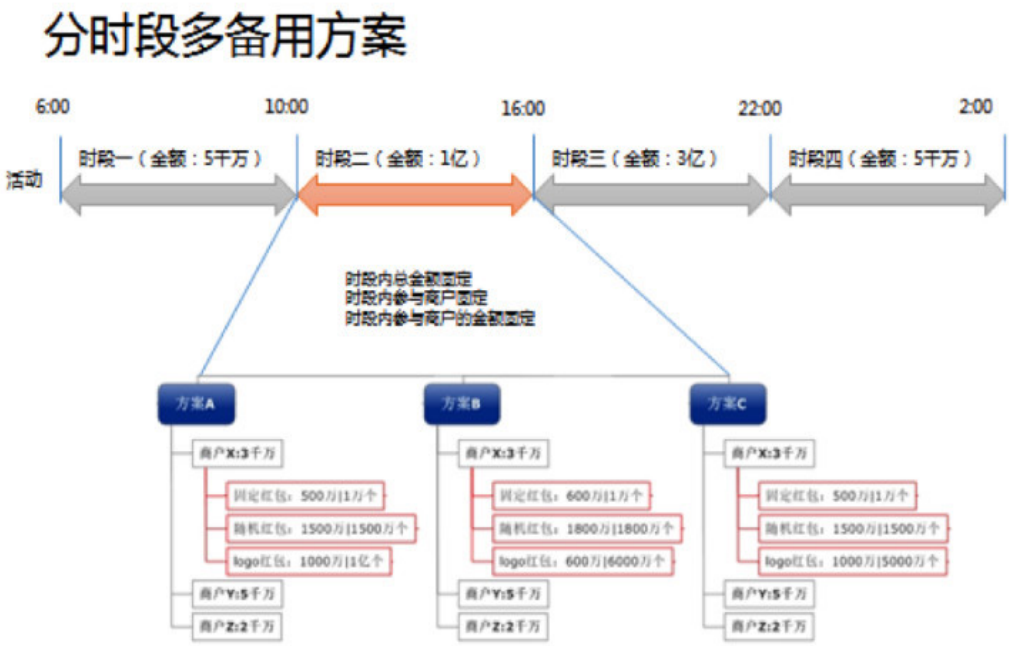


图 4

- 按规定的格式文件导入预算和配置，进行数据和逻辑合理性校验；
- 一切利用工具进行处理，资金的支付、退款、红包的随机生成和多商户随机打散、预红包文件分割、预红包数据的校验，减少人为过程导致的潜在失误；
- 优化红包随机算法和文件处理方法，将红包随机分割和多商户随机打散算法的 n^2 时间复杂度优化 n，压测 30 亿红包的生成时间为 2~3 小时，极大缩减准备时间，增加方案调整的回旋时间，可以让更多的资金上车。

其次，前边提到方案有如此多的变数、调整、变更，如何支持？还是回归模型，建模时就要考虑支持同样的预算多套资金配置方案。（见图 4）

同样的资金预算，同时或依次生成多套预红包数据文件，提前做多手准备，方便方案变更。

再次，如此大量的资金就意味着如此大量的诱惑，会不会出问题呢？

- 方案预红包数据未提前落地 DB，导致拆红包时缺少一次红包数据有效性的检验；
- 预红包数据存放在微信接入机上，存在被攻陷获取或篡改的可能；
- 红包数据在传输的过程中存在系统异常或恶意攻击，导致数据错误特别是金额错误的可能；
- 系统内部可能存在恶意人员直接调用拆红包的接口写入不存在的红包。

墨菲定律要求我们必须重视以上安全隐患，解决方案就是加密——对预红包数据进行加密，加密库和解密库独立维护保证密钥不被泄漏，通过工具生成预红包数据时用密钥进行加密，预红包数据在部署存储和传输的过程中保持加密状态，仅在拆红包的逻辑中编译二进制紧密库进行解密。

同时，鸡蛋也不能放在一个篮子里，为了控制密钥泄漏导致的影响，确保资金风险可控，整个预生成的红包数据可能分别使用了几百到几千个密钥进行加密，一个密钥加密的红包资金量在 20~30 万。解密库还需要能设置密钥 ID 的白名单和黑名单，确保未被使用的密钥不能被利用，确认泄漏的密钥可以进行拦截。

2. 极限压缩

如果是百亿个红包，那么产生预红包数据文件的大小不经过压缩是非常恐怖的，传输部署几百 GB 或几 TB 的数据是很艰苦的，一旦发生调整变更会变得非常艰难，所以需要对数据进行极限的压缩。

- 对于支付单号、商户号、红包账户等信息，由工具导成配置文件，

配置到拆红包逻辑中，加密的红包数据中仅用一个批次 ID 表达；

- 拆分红包 ID，部分分段同样转为 ID，解密库解密后利用配置进行还原；
- 加密部分（Ticket）：红包 ID、金额、批次 ID、密钥 ID，压缩到 16 字节；
- 单条红包记录二进制表达，压缩到 26 字节。

3. 对账

上面所有都做到就安全了么？真的就有人写了一个不存在红包进来会怎样？是否还有其他未考虑到的潜在风险？所以我们需要一个兜底——对账，把一切都要对清楚才放心。

对账后再入账的时效在 30~60 分钟，会造成不好的用户体验。为了提升用户体验，将大额的预红包数据（占比 10% 左右）导入 KV（高速缓存），拆红包时进行即时校验，即时进行转账入账，未命中 KV 的红包则等待对账后异步完成入账。

- 资金配置与资金预算要总分对账；
- 红包数据文件与资金剧本进行总分对账；
- 红包数据进行全局去重验证；
- 红包数据进行解密验证和金额验证；
- 如果密钥泄漏红包金额等被篡改，兜底要进行红包 DB 中已拆红包数据与预红包数据的对账后，才能实际进行转账入账。

小结

未来我们可能还继续会有类似的活动，虽然我们已经有了一个经过实

践的可复用的技术方案，接下来我们希望能够继续优化，并实现一些可复用的模块和服务，可以快速应用到下一次活动或者其他业务场景中。例如，我们在 2015 年春节首次完成除夕活动后，就把其中的资源预下载系统进行了完善和通用化，随后应用在多个业务场景和这次的 2016 年除夕活动中。未来可以有更多类似的系统和服务可以被复用和通用化。

张文瑞，一直从事后台系统设计开发，早期涉足传统行业软件，后投身互联网。作为微信最早的后台开发者之一，见证了微信从零开始到逐渐发展壮大的过程。

王鹏程，微信支付商户系统开发组组长，专家工程师。2008 年加入腾讯，6 年的电商经验，做过 c2c、b2b2c、b2c 及 erp，2 年第三方支付开发经验。

云计算演义（序）：云计算时代来了，没有狂欢盛宴只有整个 IT 业的呜咽

作者 刘黎明

【编者按】InfoQ 中文站新推出《洞见云计算》专栏，精选包括刘黎明个人公众号上的文章，让更多的读者朋友受益，本栏目的内容都经过原作者授权。本文是《云计算演义》系列文章第一篇。

私有云巅峰已过，混合云正在崛起，公有云大战正酣。开源云软件之间争风吃醋与拥抱并存，商业与开源云软件之间攻击与暧昧界限模糊，公有云服务之间实力与谋略火花四溅。云计算时代不可阻挡地来了，但没有想象中的狂欢盛宴，只有整个 IT 业的呜咽。除了公有云三巨头，其他的云计算公司和企业 IT 都要完蛋吗？

这不是一个用来耸人听闻、夸大其词、吸引眼球的标题。相信包括我在内，不少从业者都开始或早已在思索这个问题。我已经思考了一段时间，但始终不愿意公开讨论这个话题。很多从业者和公司其实多少也意识到这个问题是存在的，甚至迫在眉睫，但没有几个人愿意公开承认和讨论。除了在其位谋其事之外，还有就是，确实没有明确的答案。

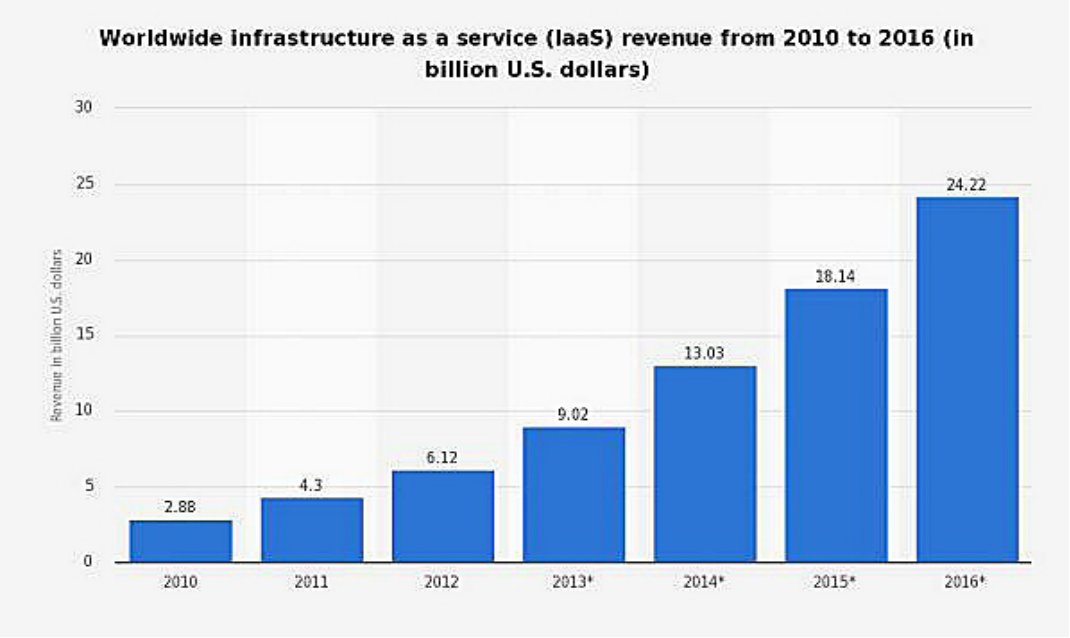


图 1

现在傻瓜也不会拒绝云

Oracle 和它的创始人埃利森，绝对不是傻瓜。他们在 7 年前称云计算为胡言乱语，是废话，是愚蠢。

2015 年 12 月 Oracle 发布的第二季度财报显示收入下滑 6% 为 89 亿美元，而净收入下滑 12% 为 25 亿美元。但是，公司的高层却希望通过强调云计算收入增长 26% 来摆脱这些数字的影响，声称云计算收入达到了出色的……6.49 亿美元。

我相信这回 Oracle 和埃利森是真心拥抱云计算，它们发动了所有的宣传机器来证明 Oracle 云计算将大有作为。有用后发制人理论为 Oracle 打气的，有用企业级客户保有量和忠诚度证明 Oracle 云计算潜力的，以及 Oracle 在亚太或全球招聘 1000 名销售的，有说 Oracle 正在打造秘密武器的。

现在 Oracle 仍然说云计算是个玩具，但仅仅是指亚马逊 AWS。更重

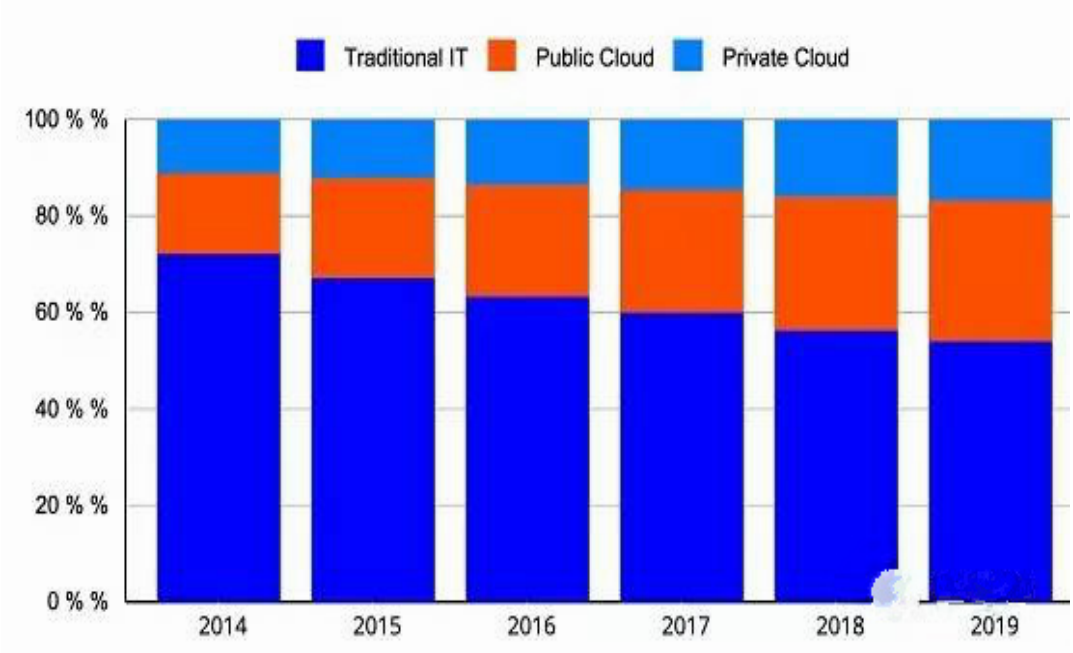


图 2

要的是，心态和行动彻底转变：Oracle 一定要而且正在变成一家云计算公司！

是什么让最固执的聪明人和最具行业统治力的公司，发生了如此巨大的转变？不可逆转的时代趋势。

过去 10 年，即使是狭义的云计算已经从 0 发展到数百亿美元的市场，甚至可能是数千亿美元的市场。图 1 是 Satista 发布的 2010~2016 年 IaaS 市场规模变化图。

未来 5 年，IDC 预计到 2019 年针对云环境的 IT 基础设施开支的年复合增长率为 15.5%，达到 543 亿美元，在企业 IT 基础设施总开支中占比从 2015 年的 32.9% 增长到 46.6%，而同一时期，非云 IT 基础设施开支的年复合增长率为 -1.7%。（见图 2）

Gartner 报告显示，2015 年全球公共云服务市场规模为 1750 亿美元，2016 年全球公共云服务市场规模有望达到 2040 亿美元，相比去年增长

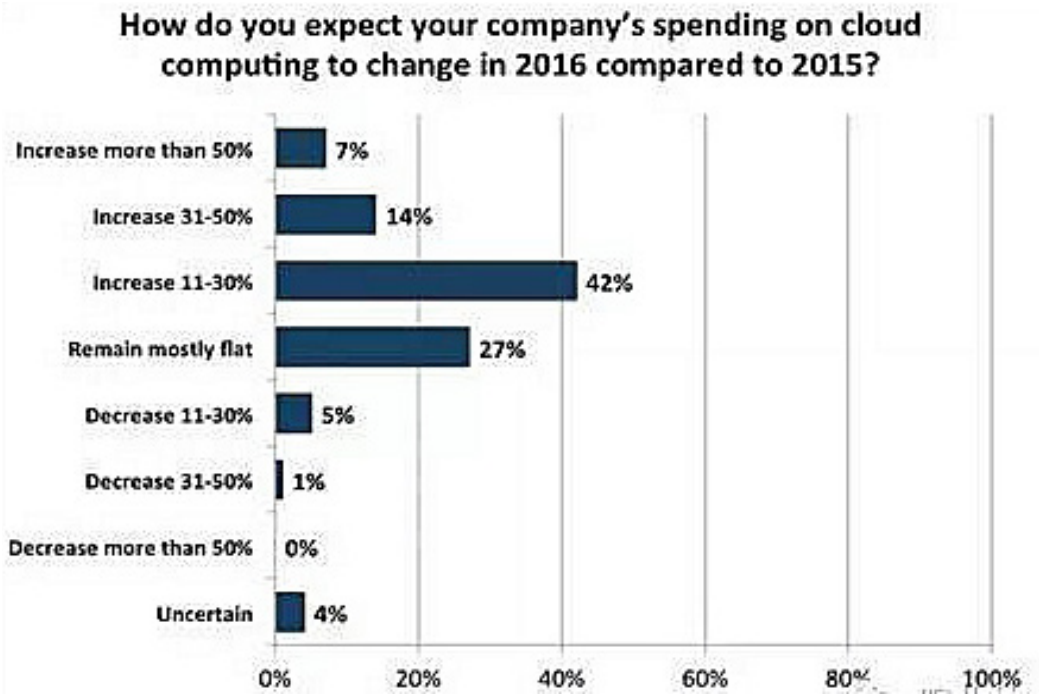


图 3

16.5%。而 2016 年的全球 IT 支出总额将达到 3.54 万亿美元，相比 2015 年的 3.52 万亿美元仅增长 0.6%。2016 年数据中心系统支出将在 2015 年的基础上增长 3% 至 1750 亿美元。

来自调查公司 Clutch 对 300 名大中型企业的 IT 专家的调查显示，90% 的企业将维持或增加云计算的开支。图 3 是调查结果。

云计算市场如此庞大，发展如此迅速，那且不是所有搞云计算公司的狂欢盛宴？搭上这班车，大不了多头前，业务轻松成长？可惜不是。

可惜不是，没有狂欢盛宴，几乎是谷歌微软亚马逊三个人的餐桌。

有人说 2015 年的云计算和亚马逊 AWS 无关，只是所有人想打败亚马逊 AWS。是的，这些人，整个传统 IT 产业，都在呜咽。

除了呜咽还能怎么办

如果你还不承认这是一个问题，那你可能正在井底，或者蒙着眼睛说

图 4

别人看不见你。当然，也可能你跟企业 IT 行业没有关系。

即使如此，你也不能摆脱云计算的影响。你和你的公司不从事企业 IT 软硬件和服务的任何业务，并不意味着你的公司不需要企业 IT。你不用任何 IaaS、PaaS、SaaS 和 XaaS，并不代表你的同事和你的公司不用。

为什么所有人想打败亚马逊 AWS？现在就云计算是块肥肉，但大部分都被 AWS 吃了，我们想吃个渣都困难。图 4 是大型企业级业务公司的营收增长情况。

图 5 是各 IaaS 服务商的市场份额情况。

亚马逊甚至把手伸到了芯片领域。2016 年 1 月 7 日，亚马逊子公司 AnnapurnaLabs 宣布，已开发出一系列名为 Alpine 的芯片，将销售给各大电子产品厂商和数据中心运营商。这标志着亚马逊将会对英特尔统治的数据中心基础设施市场发起挑战。在桌面芯片式微之后，服务器和数据中心芯片是英特尔少有能增长的市场，目前这一市场对 ARM 架构的采用还处于试水阶段，亚马逊至少可以在将来在自家云服务中逐步采用自由芯片，然后拓展外部市场。

如果芯片和 CPU 都自己干了，主板网卡路由器还远吗？这些公有云从

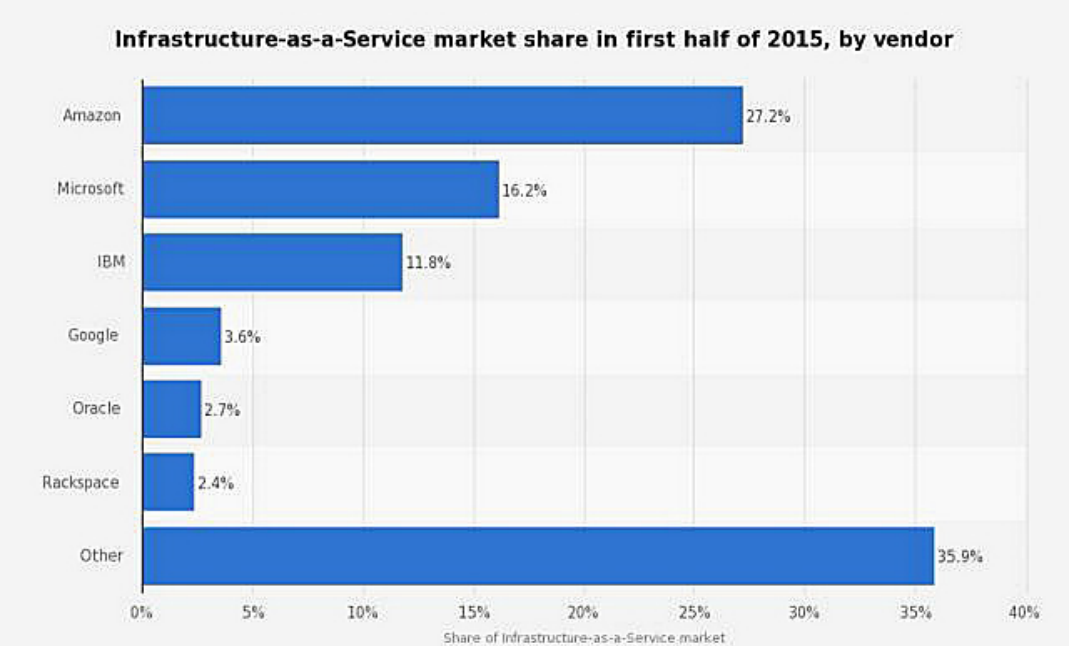


图 5

来没有大量采购外部的 IT 管理软件，难道将来也不准备从传统 IT 设备上哪里采购硬件设备？

数据库？中间件？ SQL 也好，NOSQL 也好，不是开源就是自己开发，公有云不需采购。中间件？早就式微了好吧，再说现在的 PaaS 组件就是新一代的中间件啊。

软件、硬件，公有云都用自家的，企业 IT 软硬件供应商怎么活？哦，还有个数据中心咨询和建设啊，IBM 的长项。以前是，将来未必。

现在流行 All-In Cloud，全压注在公有云上。IT 公司如 Neflix，非 IT 公司如 GE，都开始陆续关闭自己的数据中心。现在热衷于建设数据中心的就是互联网数据中心运营商，以后就剩下公有云服务商，那才几个，而且这些公司未必需要借助第三方建设。这不，微软都在数据中心上创新了，代号为“ProjectNatick”的项目，正在尝试将设备沉到冰冷的海水中解决发热问题，用涡轮或者潮汐能发电解决能源问题。微软研究者们，没有忽视项目对海底环境的影响。对虾在系统附近游动时，它所发出的声

音完全可以淹没容器制造的噪音，在容器上方几英尺的海洋中就没有热量了。注意，是海水？不是千岛湖的湖水。

看上面这些图，亚马逊 AWS 也没有什么大不了嘛，增长快一些，市场份额高一些，但也不是垄断啊？问题是，现在搞 IaaS 和 PaaS 的公司，只有亚马逊和谷歌是新玩家，其它的公司都是传统 IT 玩家，他们是在自己吃自己啊。不是吃老本，是把老业务换成新业务，新业务搞不过谷歌和亚马逊不说，老业务下降得比新业务的增长还要快。客户用一块钱的云服务，将减少 4 块钱的传统 IT 业务。而且，这两个新玩家，还特么打价格战，这是不让别人活的节奏了，难道只能在自残或被别人残之间选择吗？

我们列出这些传统 IT 玩家的市值：Oracle 市值 1499 亿美元，IBM 市值 1247 亿美元，HPE 市值 229 亿美元，DELL 市值约 300 亿美元，EMC 市值 476 亿美元，VMWare 市值 190 亿美元，思科 1162 亿美元，6 家公司合计 5103 亿美元。大约等于微软或谷歌的市值，是亚马逊 AWS 的两倍。但是在其他文章中，我分析，AWS 的市值可能在 8 年内超过所有这些企业 IT 公司之和。

分析师 Trip C howdhry2016 年初预测，传统 IT 公司 2016 年将继续大规模裁员，总计 20 多万人，大约是这样的。

- EMC：总员工 70,000 裁员比例 15%~20% 裁员人数 10,000~14,000。
- VMWare：总员工 17,000；裁员比例 10%~15%，裁员人数 1,700~2,500。
- HP Enterprise(HPE)：总员工 240,000；裁员比例 30%， 裁员人数 72,000。
- IBM (IBM)：总员工 379,000；裁员比例 25%， 裁员人数 95,000。
- Cisco (CSCO)：总员工 72,000；裁员比例 20%， 裁员人数 14,000。
- Juniper (JNPR)：总员工 8,800；裁员比例 15%， 裁员人数 1,300。
- Oracle (ORCL)：总员工 132,000；裁员比例 20%， 裁员人数 26,000。

- Microsoft (MSFT)：总员工 118,000；裁员比例 15%， 裁员人数 18,000。
- Symantec (SYMC)：总员工 19,000；裁员比例 15%， 裁员人数 2,800。
- F5 Networks (FFIV)：总员工 4,500；裁员比例 10%， 裁员人数 450。

企业 IT 市场最大的两个玩家，IBM 已经连续 15 个季度营收下滑，HP 在最近 17 个季度 16 个季度营收下滑。

听到传统企业 IT 公司的呜咽了吗？营收利润下滑，降福利减人员，最大的问题是，所在的市场在萎缩。更大的问题是，新市场被公有云抢了先。这些都是叱咤风云的公司，至少一度，现在，除了继续守住现有产品的市场，也只能暗自呜咽，有钱有志气的，还能出来怒吼一声。

有人说亚马逊 AWS 正在吞噬整个 IT 产业。吞噬不了，会有很多公司一起玩，但仅是在陪 AWS 或三巨头玩而已。就像前几年，搞智能手机和移动互联网的公司，基本上都是陪苹果玩而已。

但是，世界不是线性发展，其他公司更不会坐以待毙。不是自残或被别人残，就是颠覆自己和被被别人颠覆。微软、谷歌、亚马逊 AWS，也逃不过去。

云计算时代，超越和创新的机会还有没有

羡慕嫉妒恨是没有用的。当公有云刚刚兴起的时候，多少专家和公司，指责公有云不可靠、不安全、新瓶装旧酒。当云计算势不可挡时，更多的公司认为是块肥肉，一拥而上，巨资投入，连个泡都没冒，然后铩羽而归。

复制黏贴是没有用的。复制公有云的一个产品，很多公司都能够做到，但不会得到客户，既然是赝品，为什么要拥有你？但现在不仅是一个产品的竞争，而是产品群和生态系统的竞争，又有几个公司能复制的了。但是中国企业可能很难理解这一点，因为所有中国公司都看到，BAT 们复制黏

贴几乎无往不利，但其实都是大公司复制小公司，小公司复制大公司、大公司复制大公司的业务几乎没有成功的。

首先，我们要意识到，传统企业 IT 的高毛利率将不复存在。

其次，云计算的高毛利率也不会存在。

这是时代赋予云计算的使命。IT 基础设施和大部分应用都将成为普及性社会基础服务，将成为日用品，将从企业的核心竞争力变成标准配置。这可能是部分公司和从业人员的不幸，但这是整个信息产业和全社会发展的利好。

我相信有很多公司在内部探讨如何面对云计算的冲击，也有很多创业者在寻求云计算领域的创业机会，还有很多天使和各种投资者在寻找将在云计算浪潮中成长的创业公司和上市公司。

当然，也有不少上了云计算这条船的大企业小公司，在思考当前的云计算业务怎么走。SaaS 这么多年，也不见挣钱。公有云被巨头砸出来的地盘都是巨头的，我们公司没钱砸了怎么办。私有云、混合云，那是直接和传统 IT 巨头直接面对面拼刺刀，传统 IT 巨头都嫌嫌这单越来越少了。

有些我知道答案，有些，我也不知道。我自认为是答案的，也不一定对，也不一定具有可行性。就像今天，我们还在微博上讨论，有些技术是谷歌启动的，但谷歌并没有从中在商业上收益，知道和看好与能做好商业化是两码事，特别是，当你讨论的是技术的时候。

七年前，Xen 还是 IaaS 标配，KVM 还没出世时，我就看好容器，特别是看好容器在私有云中的用途。私有云不应该也不需要，所有场景都采用与公有云同样的虚拟化技术。但即使我把这个观点告诉一个 IT 大佬，或者给我 10 个亿，我也很可能做不到能如 Docker 般推动容器的发展。

再假设，即使我或某人能给这些问题一个正确的答案，不同的人和团

队执行，那结果也是千差万别。为什么？云计算浪潮和所有商业浪潮一样，不是采用一项技术，或参与一个项目，或开发一个产品，或走对了方向，就可以成功的。

你需要坚定的战略，也需要灵活的试错，你需要很牛的核心，也需要互补的团队，还需要钱，等等，商业的成功，没有范式。

虽然目前公有云大战正酣，基本上，三大巨头领先，甚至有二足鼎立之势，绝对领导者也有望数年内确立。但，世界不是线性发展。

公有云第一战已经结束， 基本的计算、存储、网络产品，AWS 大幅领先，微软谷歌差点意思。IBM、Oracle、VMware 还刚进场，迟到了。DigitalOcean 专注于特定产品细分市场，倒也是不错的思路。这一战之后，传统基础设施将成按需服务，日用品。国内巨头，阿里云、腾讯云，还得努把力，入场晚了点，立足中国，再到跟中国贸易密切的国家特别是亚非拉开花，再战发达国家，可能效果好点。

公有云第二战正在如火如荼，数据分析和大数据、移动互联网、物联网、容器。这一战之后，大数据、APP 开发、物联网平台将成按需服务，日用品。谷歌在计算和网络性能上略胜一筹，数据分析和大数据上领先一节，容器上也略占上风。AWS 在 CloudOps 上领先，微软什么都跟得很紧。IBM 在 PaaS、混合云上的 BlueMix、在数据分析上的 Watson 和 MarketPlace、在裸机云上的 SoftLayer 也是有声有色。Oracle，还得看看它有什么秘密武器。

公有云第三战在什么领域，还未可知，也许是 Serverless、CloudOps 或 PaaS。

三大公有云巨头并非没有软肋。

AWS 的母公司亚马逊利润率很低，长久在盈亏平衡线徘徊，而公有云

当前竞争激烈，属于资产和人才密集型行业，如果不能有充足的财力支持，可能功亏一篑。重要的是，亚马逊尽管一年发布数百项更新，但大都是跟随性和防御性的，比如 Lambda，显然受到 Parse 的启发。

亚马逊尽管从 2011 年来，招聘了数百名企业传统 IT 行业的企业销售人才，但如何让这些人在云平台上发挥他们的经验，是个难题。而谷歌，在产品成熟度和对企业客户的支持上，更待改进。微软，最大的问题就是时间，如果不能在新业务上突破，股价将难以维持，然后就是自己，抛弃 windows，拥抱用户所拥抱的。

有人说，打败 AWS 的方法，就是跳过 AWS。AWS 的根基是 IaaS，那就让 IaaS 不再重要。大家一起搞一个标准的统一的 PaaS，而且就基于 AWS 搞，让开发者和运维不再关注 IaaS。这个 PaaS 成功之日，就是 AWS 没落之时，因为这个 PaaS 设计成可以在任何 IaaS 上运行。但是，先不说这样的事情在商业环境里可不可能，AWS 也不允许它发生：自己搞了 ServerLess 服务 Lambda 和运行环境服务 BeanStalk，还开卖 Redhat 的 OpenShit。

这都是梦想啊，大家一起，标准的，统一的，都是梦。就像我三年前所言，中国 IDC 行业的出路，单兵突围困难，那就只能搞中立云平台。当然，中国没有人搞。但是不等于真的没人搞，OnApp 联盟，就搞了这么个东西，目前接入了 43 个国家、113 个城市的主机和 CDN 服务。

机会和创新，一直都在进行，从没停过。以前是，以后同样。

机会和趋势，变化如此之快，甚至令人眼花缭乱，更让人难以取舍。云计算一路走来，系统虚拟化、Xen、KVM、虚拟服务器、容器、Docker、Kubernets、Mesos、对象存储、Ceph、Swift、分布式块设备、NoSql 数据库、内存数据库、Pig、Hadoop 、MapReduce、Hive、Storm、Spark、yarn、数据仓库、商业智能、OpenStack、CloudFoundry、DevOps、Chef、

Puppet、持续交付、OpenFlow 和 Open VSwitch、SDN、Virtual Private Cloud、微服务、ElasticSearch、Solr、Unikernals、ServerLess、私有云、混合云、云经纪，一个个技术、产品、概念，你方唱罢我登场，有一炮而红的，有咸鱼翻身的，有转瞬即逝的，有默默无闻的，最后都归于云计算翼下。

那决定云计算企业发展的，是技术？产品？战略方向？营销？用户体验？企业文化？亦或个人性格？最大的决定因素，当然离不开云计算时代、信息化行业、社会生产力的发展趋势。

在众多云计算有关的技术、项目、方向中，哪些是明日黄花，哪些是过眼烟云，哪些是未来之星，哪些注定是坑，哪些能助你一臂之力？你和你的公司需要有视野、有战略、有思想，然后才能不浪费伙伴们的时间和投资者的金钱，走向勃勃生机。

商业的成功，没有范式。但，可以有经验和教训，可以有战略和方向，提高成功的概率。我们不可以预测未来，但可以察言观色、落叶知秋，顺势而为。如果你有勇气和资源，有想法或思想，你甚至可以创造新的技术、独特的产品，甚至一个新的浪潮和时代。用剑的最高境界，就是心剑合一，创新的最高境界，就是时代和你合一，你创造一个时代，时代造就了你。

说远了，还是来点实际的。

有人说，什么以史为鉴、借古观今，什么洞察秋毫、纵横捭阖，我都不感兴趣，我就爱看看每天有什么新闻，就可以了，足够了，知天下了。是的，我想 99% 的人满足于每天看看突发社会新闻就满足了，看看科技和 IT 新闻都算有追求的了。

但是还有 1% 或者或多或少的一群人，不满足于此。纵览云计算的技术、产品、公司、人物在过去 10 年到今天、今后的发展脉络，探讨云计算的战略、

执行、文化上的实践，大胆地去猜测、预测和创造，帮助自己和公司走得更久更远，成为顶尖的专家和最有价值的公司。

万马奔腾时，有人看到恐惧，有人看到机会。

待续……

刘黎明，云计算布道师，微信公众号：鸣北林（techculture）。2008 年 3 月起从事云计算与 IaaS 产业研究、产品设计、技术研发，现任新网数码信息技术有限公司副总经理，主管研发和云计算事业部，曾担任网银互联副总经理兼云计算事业部总经理、世纪互联云快线产品总监和研发主管、思科系统（中国）研发有限公司软件开发工程师。先后分别主持了国内第一个基于 Xen 和 KVM 的基础设施公有云的研发和产品化工作，并完成重点产业化科研项目——北京市科委“虚拟化管理平台”。湖北随州人，于北京科技大学计算机系统结构专业获工学硕士学位。

扫码关注回复“**Hadoop**”
查看大数据专栏系列文章

在线支付之风控系统架构选型

作者 **康大强**

风控系统介绍

伴随着互联网的发展，游戏、商贸、慈善、博彩、餐饮等各行各业都开始触网。“天下熙熙，皆为利来；天下攘攘，皆为利往”，种类繁多的网络活动直接或间接的都与钱相关，传统的支付不能满足人们快节奏的互联网生活，电子支付应运而生，但电子支付给人们带来方便快捷的同时也给参与支付的各方带来了风险，账号盗用、虚假交易、金融欺诈等事件层出不穷。支付风险自古就存在，在互连网繁荣的今天只是多了些新花样，风控系统就是通过监控交易、渠道、产品、用户，对相关数据进行实时、准实时、定时的分析、挖掘，从而识别交易风险，尽早发现欺诈，采取各种措施降低交易损失。

风控系统功能性需求

实时监控：针对各类支付业务交易风险的事中监测与控制。

- 同步反馈：接受并处理各支付业务平台的支付交易信息请求，并由风控系统将处理结果实时返回业务系统。
- 联动控制：对风控系统识别出的风险信息，进行系统自动化（通过人工预置策略实现）交易风险处理。
- 持续迭代：交易风险特征识别方式可以通过系统自动整理、人工设定参数、外部资源共享等方式进行持续动态更新。
- 统计分析：对于风控系统中存在的各类信息和数据，有关人员可以通

过组合查询和查阅报告等方式，对系统运行情况、交易风险情况、岗位绩效情况等进行监督。

风控系统非功能性需求

- 灵活性：风控规则需要经常新增以及调整，如何降低风控规则新增、修改的成本是风控系统需要解决的一个问题。此外，风控系统需要能够与不同业务系统进行集成，需要和业务系统保持相对的独立，以利于集成。
- 性能：风控系统响应时间需要小于 100ms 同时能够有很大的吞吐量，风控系统降低风险的同时不能影响用户体验，否则就本末倒置了。
- 准确性：控系统根据风控规则会自动冻交易相关资源等，如果准确率过低会引起大面积投诉，损害公司信誉、公众形象使用失去信心，给公司带来巨大的损失，所以风控系统识别风险的准确率必需要有一个下限，同时要注意风险识别的准确率和覆盖率是相互影响的两者要达到一个可以接受的平衡。

风控系统的误区

风控系统的目的是在不影响正常业务的同时把交易风险降低到合理的水平，风控系统并不能消灭风险，所以在建设风控系统时不可盲目追求数据的准确性以及一致性。

风控系统架构

风控系统主要由以下几个部分构成：风控实时引擎、风控准实时引擎、风控定时引擎、惩罚中心、规则中心等，下面分别进行介绍。（见图 1）

1. 风控实时引擎

支付系统会把当前交易部分信息同步传递给风控实时引擎以获取当前

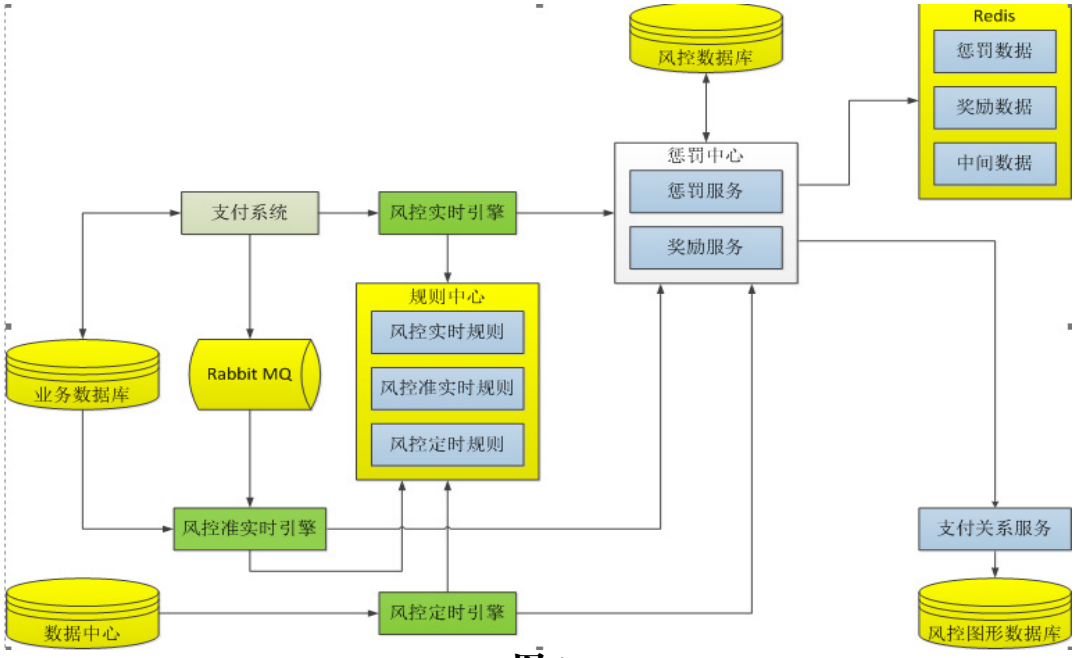


图 1

交易的风险评分和处理建议。风险评分总分为 100，分数越高风险越大，-1 代表风控系统不能对当前交易进行评估风险，一般是指风控系统没能获取到当前交易正面或负面的信息。风控实时引擎最大的挑战是性能，实时引擎对支付交易进行风险评估必须在 100ms 内完成，否则会影响用户体验。

风控实时引擎依据公司积累的风控数据以及第三方风控数据对进行风险评估。公司风控数据主要来自风控准实时引擎和风控定时引擎。

风控实时引擎建立在规则引擎 Drools 基础上，目的是为了提高系统的灵活性和可配置性。

图 2 是风控报告的域模型。

规则引擎优点

风控系统中风控规则变动比较频繁，其它部分相对稳定，通过引入规则引擎可以解耦系统与规则、提高复杂逻辑的可维护性、提高规则的可读性以及可理解性。

2. 风控准实时引擎

交易欺诈行为具备一定的隐蔽性，不是所有风险都能实时的作出正确

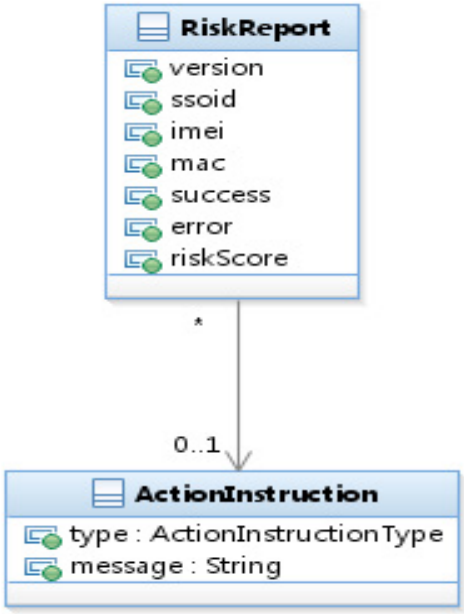


图 2

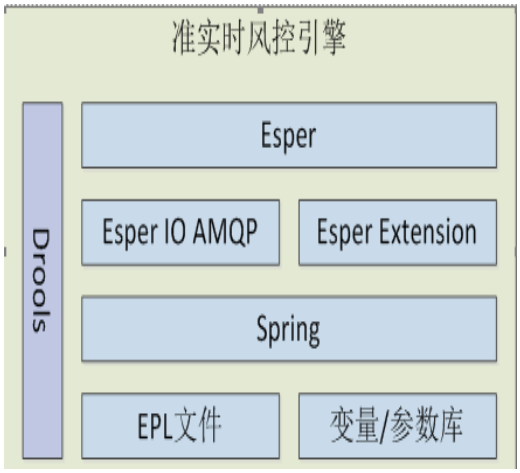


图 3

的评估，某些情况下需要通过对最近一段时间的数据进行分析才能确定欺诈行为，例如：对最近一个月的用户行为进行分析。风控准实时引擎从消息服务器获取交易数据，对交易数据流作异步分析，分析的结果通过风控服务存入风控数据库，供实时风控引擎评估交易风险。风控准实时引擎从发现异常到风控数据生效的时间在 100ms 以内，可以有效防止交易风险扩大。

风控系统初始建设时中风控准实时引擎往往是通过消息监听器消费交易消息，把中间数据存储在 Redis 缓存上，对数据进行多维度分析。由于需要针对每项指标开发相应程序，风控规则开发成本较高、上线周期长，为了解决这个问题以及提高系统的灵活性、可配置性，在风控系统 2.0 版本引入了 CEP 引擎 Esper 以及规则引擎 Drools。

风控准实时引擎需要使用 Drools、Esper、Esper IO AMQP、Esper Extension、Spring 等组件，其中 Esper Extension 是我们对 Esper 的扩展，主要用来提高 Esper 可配置性以及丰富 Esper IO 功能以及弥补 Esper 缺陷。

图 3 是风控准实时引擎架构。

- 什么是 CEP

事件驱动是一种监测、分析信息流从中得出推论的方法。CEP(Complex Event Processing) 也就是复杂事件驱动，是结合多种数据源的数据对信息流进行监测、分析从推理出一些复杂的事件或模式，CEP 的目的是识别出一些有意义的事件，例如：机遇、威胁，并且尽可能快的作出反应。

CEP 引擎已经被一些公司开发出来，用来满足那些需要分析事件并对其作出反应的的需求，下面是一些典型的应用示例：

  - 业务流程管理和自动化（流程监控、商业活动监控、报告异常）
  - 金融（自动化交易、欺诈检测、风险管理）
  - 网络以及应用监控（入侵检测、SLA 监测）
  - 传感器网络应用（读取 RFID、生产线调度与控制）

- CEP 技术选型

下面列出一些 CEP 产品，大家可以根据自己的情况选用。

开源 CEP 产品：

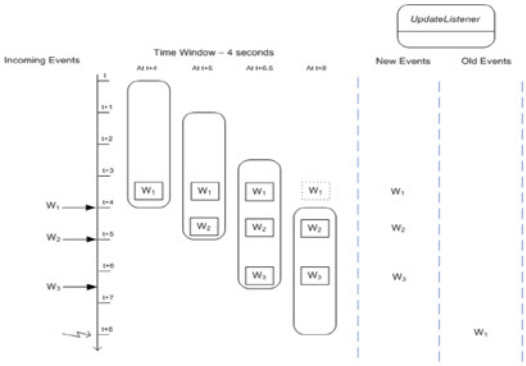
  - JBoss Drools Fusion
  - EsperTech Esper
  - Triceps

商业 CEP 产品：

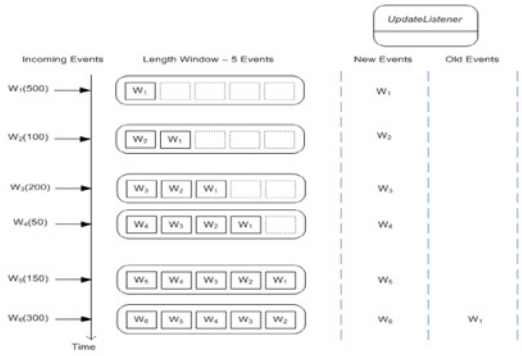
  - EsperTech Esper Enterprise Edition
  - EsperTech EsperHA
  - IBM Operational Decision Manager (IBM ODM)
  - Oracle Stream Explorer platform
  - TIBCO BusinessEvents
  - TIBCO StreamBase

窗口	描述
Length Window	滑动长度窗口，从现在到过去扩展指定数目的元素。
Length Batch Window	翻滚窗口，积累一批事件，当积累了给定数目事件时就会释放这些事件。
Time Window	滑动时间窗口，从现在到过去扩展指定时间间隔。
Time Batch Window	翻滚窗口，积累一批事件，当积累了给定时间事件时就会释放这些事件。
Externally-timed Window	滑动时间窗口，根据表达式给出的时间进行滑动。
Externally-timed Batch Window	翻滚窗口，根据表达式给出的时间进行翻滚。

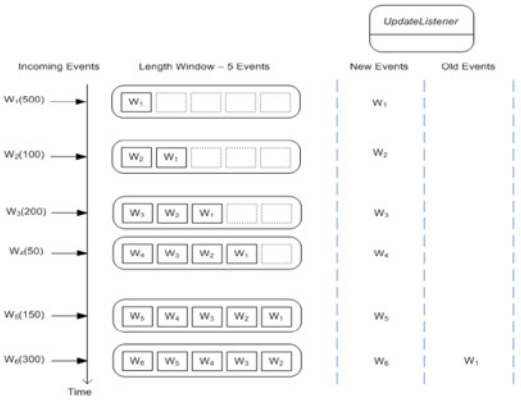
表 1



Time Window



Length Window



Time Batch Window

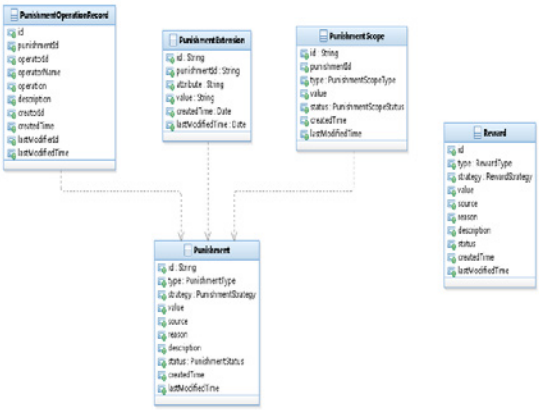


图 4

- Esper 优点

数据窗口机制完善

Esper 目前支持大约 30 种数据窗口，深入理解窗口是应用 Esper 的关键，表 1 列出常用的几种。

EPL 语句的语法与 SQL 相似降低学习成本

名称	描述
Listener Callbacks	应用程序需要为 EPStatement 对象提供 UpdateListener 或 StatementAwareUpdateListener 实现
Subscriber Object	应用程序提供一个 POJO(plain Java object)给 EPStatement 对象，这个 POJO 暴露出一些方法来接收 EPStatement 对象的结果。
Pull API	应用程序向 EPStatement 对象请求结果，然后通过 java.util.Iterator<EventBean>接收结果。如果应用程序不需要实时获取 EPStatement 对象结果，这种方式就可以使用。
On Demand Query	On-demand 查询是不连续的，查询引擎执行查询然后返回结果集给应用程序。On-demand 查询是非常轻量的，Esper 既不会创建 EPStatement 对象也不会在引擎内对这个查询进行追踪。

表 2

事件处理语言（EPL）是 SQL 标准语言并做了扩展，提供了 SELECT、FROM、 WHERE、 GROUP BY、HAVING 和 ORDER BY 等子句。

使用方式灵活

Esper 提供了丰富的 API，可以独立部署也可以集成进任何应用。

支持多种获取结果方式

见表 2。

- Esper 缺点

Esper 统计分析的中间数据全部是存储在内存中，不能跨服务器，只能单机部署，内存有限，存在单点故障，由于全内存操作，系统重启后中间数据就会丢失无法恢复。Esper 的这些缺点风控系统都可以接受，对风控系统没有实质的影响。

3. 风控定时引擎

某些非常隐蔽的交易欺诈通过实时或准实时风控引擎很难发现，这些风险需要通过分析用户跨月或跨年的数据才能识别。定时风控引擎主要用来定时对支付相关等数据进行深度挖掘，建立对应的风控模型，典型应用场景是用户的信用等级模型以及用户行为分析。

定时风控引擎构建在 Hadoop 集群上。

4. 惩罚中心

惩罚中心负责积累风控数据并提供奖励和惩罚的相关服务，风控实时引擎、风控准实时引擎、风控定时引擎会调用惩罚中心的服务查询或保存风控数据。惩罚中心针对不同维度提供多种惩罚策略以及多种奖励策略。风控实时引擎在识别交易风险时根据规则综合考虑奖励以及惩罚相关数据以提高风险识别准确率。

图 4 是处罚中心域模型。

风控系统的建设往往是分阶段实施的，一般分为以下三个阶段：

第一阶段：核心任务：

- 搭建基础架构
- 网上应用接入
- 手机应用接入
- 建立起有效的监控团队和工作机制

第二阶段：提升性任务：

- 交易监控能力
- 风控模型提炼
- 监控团队能力和工作机制

第三阶段：后续发展

- 持续优化
- 新应用接入

团队技能是逐步积累的，对风控系统以及风控模型的理解也需要时日，如果初始建设阶段求大求全会使建设工作增加不确定性，一旦出现技术或业务方向上的偏差会给公司带来巨大的损失。循序渐进逐步建设，在前一阶段的工作产生效益时再去进行下一阶段的工作阻力会小很多，任何时候保持团队对项目的控制力都是相当重要的，这是我们在建设风控系统的过程中的一点感悟。

OPPO OS产品中心

OS产品中心是致力于为OPPO用户提供极致、有格调的移动互联生活方式，通过提供硬件+软件+互联网的一体化服务，形成软、硬并进的半封闭生态平台的战略级事业部。

OS产品中心 目前主要业务包含ColorOS及移动互联网产品两大方向。办公地点位于深圳南山后海中心区。

热招职位： 高级算法工程师、数据分析/挖掘工程师、
数据平台开发工程师、高级PHP开发工程师、资深/高级JAVA开发、
移动互联网高级产品经理、高级游戏运营经理、
移动互联网海外产品运营经理

更多职位信息及投递简历，请关注OPPO招聘 微信公众号

亚马逊 aws

Geekbang. 极客邦科技 | InfoQ

InfoQ在线课堂

无服务器架构实践：如何利用 AWS Lambda 创建应用

时间

2016年5月24日（周二） 20:00-21:00

架构师 月刊 2016年4月

本期主要内容：为了实现一致性，我们从事务方案转移到流处理方案，Airbnb的大数据平台架构，集群调度框架的架构演进之路，如何成为架构师？7个关键的思考、习惯和经验，Jigsaw终于添加到JDK 9中了，在微服务中保证服务的一致性。

云生态专刊 06

《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 第五季

本次的《中国顶尖技术团队访谈录》·第四季挑选的六个团队虽然都来自互联网企业，却是风格各异。希望通过这样的记录，能够让一家家品牌背后的技术人员形象更加鲜活，让更多人感受到他们的可爱与坚持。

架构师特刊 架构漫谈

架构漫谈是由资深架构师王概凯Kevin执笔，InfoQ架构垂直社群策划的系列专栏，专栏以Kevin的架构经验为基础，逐步讨论什么是架构、怎样做好架构、软件架构如何落地、如何写好代码等问题。