

# 不一样的技术创新

-阿里巴巴2016双11背后的技术





# 不一样的技术创新

--阿里巴巴 2016 双 11 背后的技术

阿里巴巴双 11 技术团队 著

## 序

## 第一章 基础设施

- 1.1 万亿交易量级下的秒级监控
- 1.2 双 11 背后基础设施软硬结合实践创新
- 1.3 阿里视频云 ApsaraVideo 是怎样让 4000 万人同时狂欢的

## 第二章 存储

- 2.1 永不停止的脚步——数据库优化之路
- 2.2 阿里云 ApsaraDB--双 11 商家后台数据库的基石

## 第三章 中间件

- 3.1 万亿级数据洪峰下的分布式消息引擎

## 第四章 电商云化

- 4.1 17.5W 秒级交易峰值下的混合云弹性架构之路
- 4.2 集团 AliDocker 化双 11 总结

## 第五章 业务架构

- 5.1 内容+：打造不一样的双 11
- 5.2 双 11 交易核心链路的故事
- 5.3 千亿访问量下的开放平台技术揭秘
- 5.4 智慧供应链
- 5.5 菜鸟双 11 “十亿级包裹” 之战

## 第六章 大数据

6.1 双 11 数据大屏背后的实时计算处理

6.2 双 11 背后的大规模数据处理

6.3 突破传统，4k 大屏的沉浸式体验

## 第七章 人工智能

7.1 基于深度强化学习与自适应在线学习的搜索和推荐算法研究

7.2 颠覆传统的电商智能助理-阿里小蜜技术揭秘

7.3 深度学习与自然语言处理在智能语音客服中的应用

7.4 数据赋能商家背后的黑科技

7.5 探索基于强化学习的智能推荐之路

7.6 DNN 在搜索场景中的应用

## 第八章 交互技术

8.1 VR 电商购物

8.2 淘宝直播在双 11 的互动实践

8.3 2016 双 11 前端突破

8.4 Weex 双 11 会场大规模应用的秒开实战和稳定性保障

8.5 双 11 晚会背后的技术

## 序

2016 天猫双 11 全球狂欢节活动圆满落幕，来自全球的消费者一共创造了 1207 亿的成交额，在这个过程中整体系统平稳运行，用户购物、支付体验流畅，物流包裹也井然有序地送到消费者手中，这背后是阿里领先的交易、支付、物流系统，以及强大的计算平台、海量数据和智能算法的共同支撑。在双 11 零点开始的半个小时里，超过 6 千万的用户使用手机来同时参与了这次活动，在零点的流量高峰，创造了每秒交易峰值 17.5 万笔，每秒支付峰值 12 万笔的新纪录，而 2009 年的第一次双 11，交易峰值仅为 400 笔每秒，支付峰值仅为 200 笔每秒，八年增长数百倍。阿里的交易系统在高峰期每分钟需要处理超过一千万的订单，而复杂的交易系统由用户、库存、订单、优惠、无线接入、安全风控、中间件等几十个核心的子系统组成，每一笔订单都需要经过所有这些系统进行协同处理，并且需要保证不出现包括价格、库存等在内的任何计算错误。同时，阿里是开放的电商平台，在交易系统中产生的每一笔订单，都需要及时、准确地同步到数以万商家的 ERP 系统中进行更进一步的处理，我们通过为商家提供弹性扩展的电商云服务能力，确保了商家 ERP 系统在双 11 系统高峰能正常运转。菜鸟的物流系统在双 11 当天共产生了 6.57 亿个包裹的物流订单和流转信息，得益于菜鸟物流系统的电子面单二段码智能生成和匹配技术，菜鸟物流系统和各个物流公司的信息系统进行物流节点调度流转的高效协作，全国物流网络运行平稳。今年双 11 不少消费者发现，天猫和淘宝越来越懂自己了，这是我们把大数据和人工智能更广泛地应用到了消费者购物链路的场景。在交易外，我们今年还推出了 VR 购物、双 11 晚会的互动等很多新的用户交互和互动形式，进一步丰富了用户的新体验。

双 11 是阿里技术团队的大练兵，双 11 后每个技术团队都会进行总结复盘，我们从中汇总了八个技术领域的经验总结，形成本书，希望能让各位读者全面地了解双 11 背后阿里技术；同时，双 11 也是未来新零售时代的技术准备和大考，我们要做好面向未来和新零售时代的技术升级准备，期待本书能让更多各领域的技术人员和我们一起，为全社会建设互联网时代的商业基础设施。

范禹

2016 年 12 月 16 日

# 第一章 基础设施

# 1.1 万亿交易量级下的秒级监控

作者：郁松、章邯、程超、癫行

## 前言

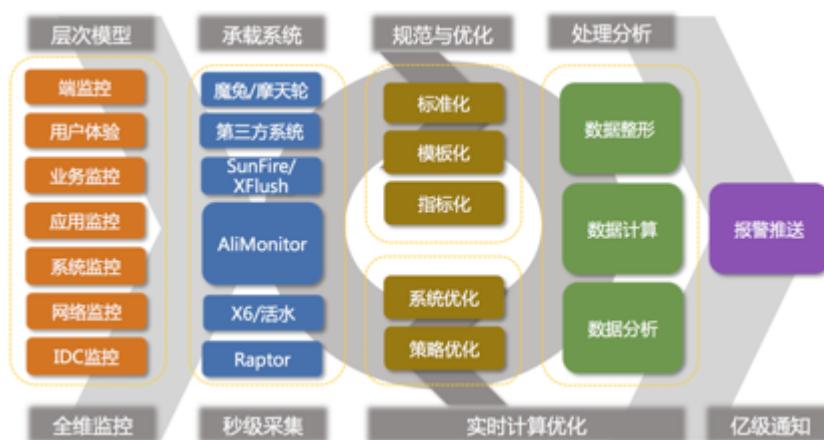
2016 财年，阿里巴巴电商交易额 ( GMV ) 突破 3 万亿元人民币，成为全球最大网上经济体，这背后是基础架构事业群构筑的坚强基石。

在 2016 年双 11 全球购物狂欢节中，天猫全天交易额 1207 亿元，前 30 分钟每秒交易峰值 17.5 万笔，每秒支付峰值 12 万笔。承载这些秒级数据背后的监控产品是如何实现的呢？接下来本文将从阿里监控体系、监控产品、监控技术架构及实现分别进行详细讲述。

## 1 阿里监控体系

阿里有众多监控产品，且各产品分工明确，百花齐放。

整个阿里监控体系如下图：



集团层面的监控，以平台为主，全部为阿里自主研发（除引入了第三方基调、博睿等外部检测系统，用于各地 CDN 用户体验监控），这些监控平台覆盖了阿里集团 80% 的监控需求。

此外，每个事业群均根据自身特性自主研发了多套监控系统，以满足自身特定业务场景的监控需求，如广告的 GoldenEye、菜鸟的棱镜、阿里云的天基、蚂蚁的金融云（基于 XFlush）中间件的 EagleEye 等，这些监控系统均有各自的使用场景。

阿里的监控规模早已达到了千万量级的监控项，PB 级的监控数据，亿级的报警通知，基于数据挖掘、机器学习等技术的智能化监控将会越来越重要。

阿里全球运行指挥中心（GOC）基于历史监控数据，通过异常检测、异常标注、挖掘训练、机器学习、故障模拟等方式，进行业务故障的自动化定位，并赋能监控中心 7\*24 小时专业监控值班人员，使阿里集团具备第一时间发现业务指标异常，并快速进行应急响应、故障恢复的能力，将故障对线上业务的影响降到最低。

接下来将详细讲述本文的主角 承载阿里核心业务监控的 SunFire 监控平台。

## 2 监控技术实现

### 2.1 监控产品简介

SunFire 是一套海量日志实时分析解决方案，以日志、REST 接口、Shell 脚本等作为数据采集来源，提供设备、应用、业务等各种视角的监控能力，从而帮您快速发现问题、定位问题、分析问题、解决问题，为线上系统可用率提供有效保障。

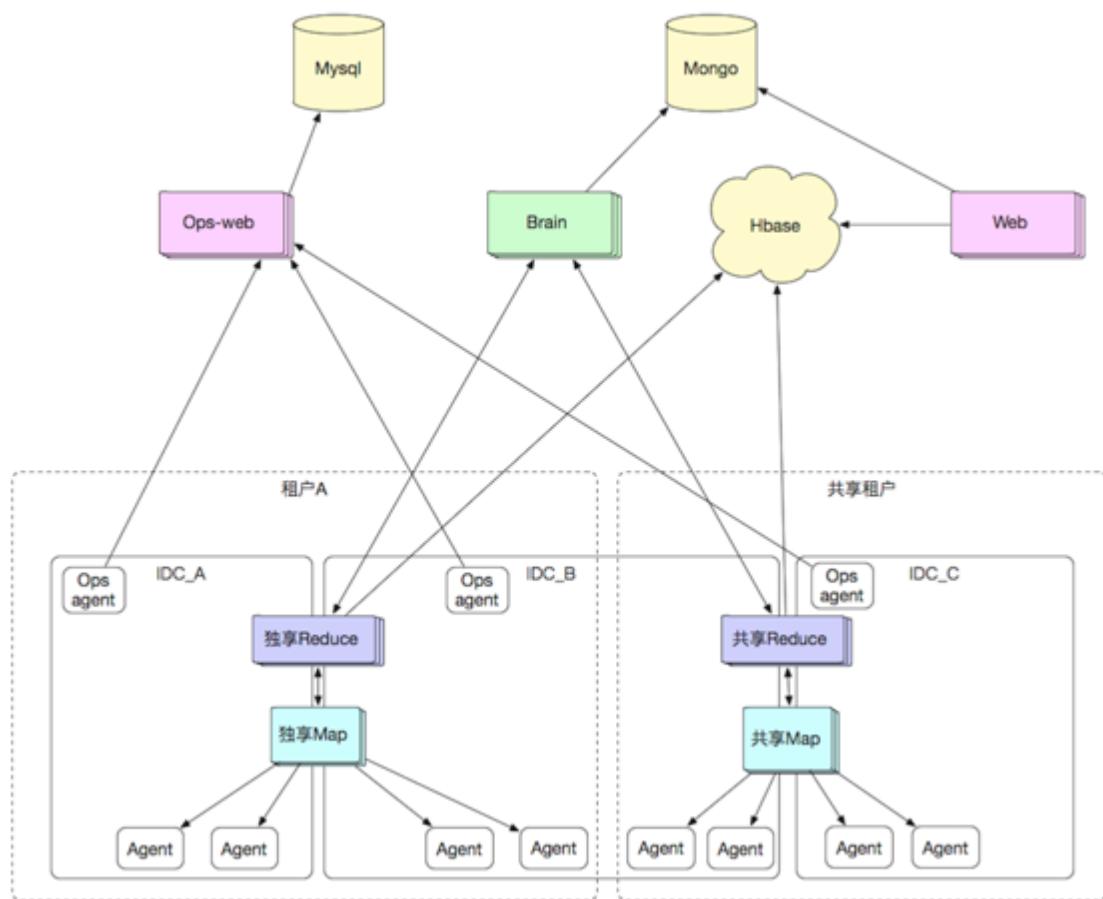
SunFire 利用文件传输、流式计算、分布式文件存储、数据可视化、数据建模等技术，提供实时、智能、可定制、多视角、全方位的监控体系。其主要优势有：

- 全方位实时监控：提供设备、应用、业务等各种视角的监控能力，关键指标秒级、普通指标分钟级，高可靠、高时效、低延迟。
- 灵活的报警规则：可根据业务特征、时间段、重要程度等维度设置报警规则，实现不误报、不漏报。

- 管理简单：分钟级万台设备的监控部署能力，故障自动恢复，集群可伸缩
- 自定义便捷配置：丰富的自定义产品配置功能，便捷、高效的完成产品配置、报警配置。
- 可视化：丰富的可视化 Dashboard，帮助您定制个性化的监控大盘。
- 低资源占用：在完成大量监控数据可靠传输的同时，保证对宿主机的 CPU、内存等资源极低占用率。

## 2.2 监控架构

Sunfire 技术架构如下：



## 2.3 监控组件介绍

针对架构图中的各个组件，其中最关键的为采集（Agent）计算（Map、Reduce）组件，接下来将对这两个组件进行详细介绍。

## 2.3.1 采集

Agent 负责所有监控数据的原始采集，它以 Agent 形式部署在应用系统上，负责原始日志的采集、系统命令的执行。

### 2.3.1.1 定位

日志原始数据的采集，按周期查询日志的服务，且日志查询要低耗、智能。Agent 上不执行计算逻辑。

### 2.3.1.2 特性

#### 低耗

采集日志，不可避免要考虑日志压缩的问题，通常做日志压缩则意味着它必须做两件事情：一是磁盘日志文件的内容要读到应用程序态；二是要执行压缩算法。

这两个过程就是 CPU 消耗的来源。但是它必须做压缩，因为日志需要从多个机房传输到集中的机房。跨机房传输占用的带宽不容小觑，必须压缩才能维持运转。所以低耗的第一个要素，就是避免跨机房传输。SunFire 达到此目标的方式是运行时组件自包含在机房内部，需要全量数据时才从各机房查询合并。

网上搜索 zero-copy，会知道文件传输其实是可以不经过用户态的，可以在 linux 的核心态用类似 DMA 的思想，实现极低 CPU 占用的文件传输。SunFire 的 Agent 当然不能放过这个利好，对它的充分利用是 Agent 低耗的根本原因。以前这部分传输代码是用 c 语言编写的 sendfile 逻辑，集成到 java 工程里，后来被直接改造为了 java 实现。

最后，在下方的计算平台中会提到，要求 Agent 的日志查询服务具备“按周期查询日志”的能力。这是目前 Agent 工程里最大的难题，我们都用过 RAF

( RandomAccessFile ) , 你给它一个游标 , 指定 offset , 再给它一个长度 , 指定读取的文件 size , 它可以很低耗的扒出文件里的这部分内容。然而问题在于 : 周期 ≠ offset 。从周期转换为 offset 是一个痛苦的过程。

在流式计算里一般不会遇到这个问题 , 因为在流式架构里 , Agent 是水龙头 , 主动权掌握在 Agent 手里 , 它可以从 0 开始 push 文件内容 , push 到哪里就做一个标记 , 下次从标记继续往后 push , 不断重复。这个标记就是 offset , 所以流式不会有任何问题。

而计算平台周期任务驱动架构里 , pull 的方式就无法提供 offset , 只能提供 Term ( 周期 , 比如 2015-11-11 00:00 分 ) 。 Agent 解决此问题的方式算是简单粗暴 , 那就是二分查找法。而且日志还有一个天然优势 , 它是连续性的。所以按照对二分查找法稍加优化 , 就能达到 “ 越猜越准 ” 的效果 ( 因为区间在缩小 , 区间越小 , 它里面的日志分布就越平均 ) 。

于是 Agent 代码里的 LogFinder 组件撑起了这个职责 利用上述两个利好 , 实现了一个把 CPU 控制在 5% 以下的算法 , 目前能够维持运转。其中 CPU 的消耗不用多说 , 肯定是来自于猜的过程 , 因为每一次猜测 , 都意味着要从日志的某个 offset 拉出一小段内容来核实 , 会导致文件内容进入用户态并解析。这部分算法依然有很大的提升空间。

## 日志滚动

做过 Agent 的同学肯定都被日志滚动困扰过 , 各种各样的滚动姿势都需要支持。 SunFire 的 pull 方式当然也会遇到这个问题 , 于是我们简单粗暴的穷举出了某次 pull 可能会遇到的所有场景 , 比如

- 正常猜到了 offset
- 整个日志都猜不到 offset
- 上次猜到了 offset , 但是下次再来的时候发现不对劲 ( 比如滚动了 )
- 等等等等

这段逻辑代码穷举的分支之多 , 在一开始谁都没有想到。不过仔细分析了很多次 , 发现每个分支都必不可少。

## 查询接口

Agent 提供的查询服务分为 first query 和 ordinary query 两种。做这个区分的原因是：一个周期的查询请求只有第一次需要猜 offset，之后只需要顺序下移即可。而且计算组件里有大量的和 Agent 查询接口互相配合的逻辑，比如一个周期拉到什么位置上算是确定结束？一次 ordinary query 得到的日志里如果末尾是截断的（只有一半）该如何处理…… 这些逻辑虽然缜密，但十分繁琐，甚至让人望而却步。但现状如此，这些实现逻辑保障了 SunFire 的高一致性，不用担心数据不全、报警不准，随便怎么重启计算组件，随便怎么重启 Agent。但这些优势的背后，是值得深思的代码复杂度。

## 路径扫描

为了让用户配置简单便捷，SunFire 提供给用户选择日志的方式不是手写，而是像 windows 的文件夹一样可以浏览线上系统的日志目录和文件，让他双击一个心仪的文件来完成配置。但这种便捷带来的问题就是路径里若有变量就会出问题。所以 Agent 做了一个简单的 dir 扫描功能。Agent 能从应用目录往下扫描，找到同深度文件夹下“合适”的目标日志。

## 2.3.2 计算

由 Map、Reduce 组成计算平台，负责所有采集内容的加工计算，具备故障自动恢复能力及弹性伸缩能力。

### 2.3.2.1 定位

计算平台一直以来都是发展最快、改造最多的领域，因为它是很多需求的直接生产者，也是性能压力的直接承担者。因此，在经过多年的反思后，最终走向了一条插件化、周期驱动、自协调、异步化的道路。

### 2.3.2.2 特性

#### 纯异步

原来的 SunFire 计算系统里，线程池繁复，从一个线程池处理完还会丢到下

一个线程池里；为了避免并发 bug，加锁也很多。这其中最大的问题有两个：CPU 密集型的逻辑和 I/O 密集型混合。

对于第一点，只要发生混合，无论你怎么调整线程池参数，都会导致各式各样的问题。线程调的多，会导致某些时刻多线程抢占 CPU，load 飙高；线程调的少，会导致某些时刻所有线程都进入阻塞等待，堆积如山的活儿没人干。

对于第二点，最典型的例子就是日志包合并。比如一台 Map 上的一个日志计算任务，它要收集 10 个 Agent 的日志，那肯定是并发去收集的，10 个日志包陆续（同时）到达，到达之后各自解析，解析完了 data 要进行 merge。这个 merge 过程如果涉及到互斥区（比如嵌套 Map 的填充），就必须加锁，否则 bug 满天飞。

但其实我们重新编排一下任务就能杜绝所有的锁。比如上面的例子，我们能否让这个日志计算任务的 10 个 Agent 的子任务，全部在同一个线程里做？这当然是可行的，只要回答两个问题就行：

1) 如果串行，那 10 个 I/O 动作（拉日志包）怎么办？串行不就浪费 cpu 浪费时间吗？

2) 把它们都放到一个线程里，那我怎么发挥多核机器的性能？

第一个问题，答案就是异步 I/O。只要肯花时间，所有的 I/O 都可以用 NIO 来实现，无锁，事件监听，不会涉及阻塞等待。即使串行也不会浪费 cpu。

第二个问题，就是一个大局观问题了。现实中我们面临的场景往往是用户配置了 100 个产品，每个产品都会拆解为子任务发送到每台 Map，而一台 Map 只有 4 个核。所以，你让一个核负责 25 个任务已经足够榨干机器性能了，没必要追求更细粒度子任务并发。

因此，计算平台付出了很大的精力，做了协程框架。

我们用 akka 作为协程框架，有了协程框架后再也不用关注线程池调度等问题了，于是我们可以轻松的设计协程角色，实现 CPU 密集型和 I/O 密集型的分离、或者为了无锁而做任务编排。接下来，尽量用 NIO 覆盖所有的 I/O 场景，杜绝 I/O 密集型逻辑，让所有的协程都是纯跑 CPU。按照这种方式，计算平台已经基本能够榨干机器的性能。

## 周期驱动

所谓周期驱动型任务调度，说白了就是 Map/Reduce。Brain 被选举出来之后，定时捞出用户的配置，转换为计算作业模型，生成一个周期（比如某分钟的）

的任务，我们称之为拓扑(Topology)，拓扑也很形象的表现出 Map/Reduce 多层计算结构的特征。所有任务所需的信息，都保存在 topology 对象中，包括计算插件、输入输出插件逻辑、Map 有几个、每个 Map 负责哪些输入等等。这些信息虽然很多，但其实来源可以简单理解为两个：一是用户的配置；二是运维元数据。拓扑被安装到一台 Reduce 机器（A）上。A 上的 Reduce 任务判断当前集群里有多少台 Map 机器，就产生多少个任务（每个任务被平均分配一批 Agent），这些任务被安装到每台机器上 Map。被安装的 Map 任务其实就是一个协程，它负责了一批 Agent，于是它就将每个 Agent 的拉取任务再安装为一个协程。至此，安装过程结束。Agent 拉取任务协程（也称之为 input 输入协程，因为它是数据输入源）在周期到点后，开始执行，拉取日志，解析日志，将结果交予 Map 协程，Map 协程在得到了所有 Agent 的输入结果并 merge 完成后，将 merge 结果回报到 Reduce 协程（这是一个远程协程消息，跨机器）；Reduce 协程得到了所有 Map 协程的汇报结果后，数据到齐，写入到 Hbase 存储，结束。

上述过程非常简单，不高大也不上，但经过多年大促的考验，其实非常的务实。能解决问题的架构，就是好的架构，能简单，为何要把事情做得复杂呢？

这种架构里，有一个非常重要的特性：任务是按周期隔离的。也就是说，同一个配置，它的 2015-11-11 00:00 分的任务和 2015-11-11 00:01 分的任务，是两个任务，没有任何关系，各自驱动，各自执行。理想情况下，我们可以做出结论：一旦某个周期的任务结束了，它得到的数据必然是准确的（只要每个 Agent 都正常响应了）。所以采用了这种架构之后，SunFire 很少再遇到数据不准的问题，当出现业务故障的时候我们都可以断定监控数据是准确的，甚至秒级都可以断定是准确的，因为秒级也就是 5 秒为周期的任务，和分钟级没有本质区别，只是周期范围不同而已。能获得这个能力当然也要归功于 Agent 的“按周期查询日志”的能力。

## 任务重试

在上节描述的 Brain->Reduce->Map 的任务安装流程里，我们对每一个上游赋予一个职责：监督下游。当机器发生重启或宕机，会丢失一大批协程角色。每一种角色的丢失，都需要重试恢复。监督主要通过监听 Terminated 事件实现，Terminated 事件会在下游挂掉（不论是该协程挂掉还是所在的机器挂掉或是断网等）的时候发送给上游。由于拓扑是提前生成好且具备完备的描述信息，因此

每个角色都可以根据拓扑的信息来重新生成下游任务完成重试。

若 Brain 丢失，则 Brain 会再次选主，Brain 读取最后生成到的任务周期，再继续生成任务

若 Reduce 丢失，每个任务在 Brain 上都有一个 TopologySupervisor 角色，来监听 Reduce 协程的 Terminated 事件来执行重试动作

若 Map 丢失，Reduce 本身也监听了所有 Map 的 Terminated 事件来执行重试动作

为确保万无一失，若 Reduce 没有在规定时间内返回完成事件给 Brain，Brain 同样会根据一定规则重试这个任务。

过程依然非常简单，而且从理论上是可证的，无论怎么重启宕机，都可以确保数据不丢，只不过可能会稍有延迟（因为部分任务要重新做）。

## 输入共享

在用户实际使用 SunFire 的过程中，常常会有这样的情况：用户配了几个不同的配置，其计算逻辑可能是不同的，比如有的是单纯计算行数，有的计算平均值，有的需要正则匹配出日志中的内容，但这几个配置可能都用的同一份日志，那么一定希望几个配置共享同一份拉取的日志内容。否则重复拉取日志会造成极大的资源消耗。

那么我们就必须实现输入共享，输入共享的实现比较复杂，主要依赖两点：

其一是依赖安装流，因为拓扑是提前安装的，因此在安装到真正开始拉取日志的这段时间内，我们希望能够通过拓扑信息判断出需要共享的输入源，构建出输入源和对应 Map 的映射关系。

其二是依赖 Map 节点和 Agent 之间的一致性哈希，保证 Brain 在生成任务时，同一个机器上的日志，永远是分配相同的一个 Map 节点去拉取的（除非它对应的 Map 挂了）。

站在 Map 节点的视角来看，在各个任务的 Reduce 拿着拓扑来注册的时候，我拿出输入源（对日志监控来说通常可以用一个 IP 地址和一个日志路径来描述）和 Map 之间的关系，暂存下来，每当一个新的 Reduce 来注册 Map，我都判断这个 Map 所需的输入源是否存在了，如果有，那就给这个输入源增加一个上游，等到这个输入源的周期到了，那就触发拉取，不再共享了。

### 2.3.3 其他组件

**存储**：负责所有计算结果的持久化存储，可以无限伸缩，且查询历史数据保持和查询实时数据相同的低延迟。Sunfire 原始数据存储使用的是阿里集团的 Hbase 产品（HBase : Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统），用户配置存储使用的是 MongoDB。

**展示**：负责提供用户交互，让用户通过简洁的建模过程来打造个性化的监控产品。基于插件化、组件化的构建方式，用户可以快速增加新类型的监控产品。

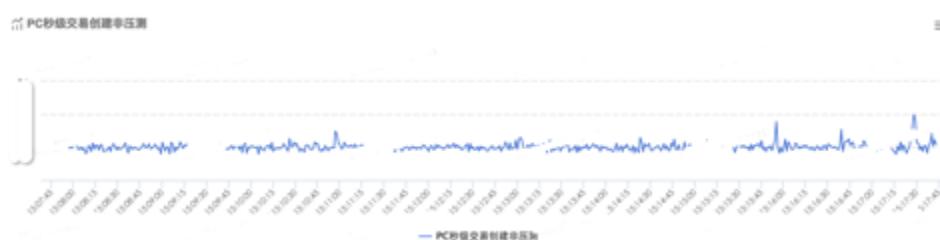
**自我管控**：即 OPS-Agent、Ops-web 组件，负责海量 Agent 的自动化安装监测，并且承担了整个监控平台各个角色的状态检测、一键安装、故障恢复、容量监测等职责。

## 2.4 秒级监控产品化

目前 SunFire 已将秒级监控能力产品化，将秒级监控能力赋能给研发运维，用户可自行根据实际业务监控需求进行配置，操作简单便捷，非常灵活。



效果图如下：



# 1.2 双 11 背后基础设施软硬结合实践创新

作者：希有

## 前言

阿里巴巴作为全球领先的互联网综合业务平台，其遍布全球的数据中心中海量 IT 硬件设备（服务器和网络）承载了世界上最全的业务体系。一方面各种互联网服务种类之广、应用类型之多、对硬件能力需求之差异，另一方面连续 7 年的双 11 狂欢节这类世界级的玩法，对于整个集团整体业务、产品、技术体系的挑战虽不绝后，但一定空前。由此要求与之一体两面的基础设施技术能力必须要解决世界级的问题，建设世界级的架构，掌控核心的关键部件技术。

在广义数据中心总体拥有成本（TCO）模型分析下，其中 IT 硬件设备通常是资本费用（Capex）和运营费用（Opex）投入方面最重要的组成部分。受驱动于特征各异的应用需求（云计算和大数据、电商、普惠金融服务、智能物流平台、生活综合服务），服务器硬件设备在核心部件、系统架构、能效成本方面的技术挑战一直是阿里基础设施竞争力构建方面“节流”环节的重中之重。

在快速发展变化的技术领域，阿里巴巴如何不断变通调整策略适应新的挑战，制定相应的硬件发展策略，通过源源不断的技术驱动力确保为业务提供质量稳定、技术先进、高性价比的解决方案，本文将从 x86 核心部件、闪存部件、系统架构等方面实践创新进行分享。

# 1 从产业结构及价值链说起

广义的数据中心产业生态是一个年需求接近千亿美金的生态，其中以大型互联网行业为代表的的大型数据中心其核心生态圈本质上是 Intel 一手打造、经营的 x86 生态圈。下图以粗线条勾勒了服务器产业典型的价值分层情况。



通过上图产业价值结构的简要分析可以得出我们的判断：

- 1 ) 传统行业链在生态演进过程中已经形成相对稳定的角色分工和附加值模型；
- 2 ) 随着大型互联网行业需求差异化演进，价值链末端中玩家已经出现纵向或者横向的转型和产品延伸(例如 ODM 的 OEM 化, 以及 OEM 的 ODM 能力加持)，价值链高端的玩家则表现出核心产品技术与节奏的差异化(例如 Flash 颗粒技术已经在技术路径和产品化节奏上差异显著)，以及端到端解决方案封装化(例如 WD 收购 Sandisk (后者收购 FusionIO) 后成为颗粒、闪存部件产品、硬盘、存储系统的解决方案提供商)；
- 3 ) 阿里巴巴基于最基础的“开源节流”的逻辑，在节流领域选择白盒化的方式对产业链和系统产品进行解构。即明确工程节奏策略为“从整到零”→“从零到整”。

进一步解析来看，“从整到零”与“从零到整”贯穿始终的驱动初心围绕着

客户价值（服务产品化、高效稳定、解决方案竞争力）和技术价值的长期竞争力构筑（资源安全、技术安全、成本竞争力），并基于此来指导技术架构方向思辨、技术路径取舍、技术产品规划。结合不同技术领域产业发展状态和阿里在不同技术领域的技术储备现状，我们需要对于硬件策略进行不断的思考、不断地调整。基于硬件团队过去三年的实践及思考，我们总结为三句话：“人无我有”、“人有我优”、“人优我快”。简单讲就是：

**人无我有**：第一个想到、实现、用起来；以及构建技术壁垒，模仿及复制成本高。

**人有我优**：用得最好，业务价值最显著，技术红利充分挖掘，具备比较优势。

**人优我快**：敏捷迭代、合理取舍、不求完美、最快落地并拿到产业红利。

正如周知的 RFC 1925 中提到的 "Good, Fast, Cheap: Pick any two (you can't have all three)"，我们需要辩证的理解 3 条（人 x 我 y）之间的关系亦是如此，取舍无时无刻不存在于其中。与此同时 3 条（人 x 我 y）本质上暗示了竞争力构筑的背后是沿着战略方向长期、持续、且不断反思与调整策略的研发投资。

## 2 软硬件结合的思考和实践

### 2.1 定制处理器

x86 生态的庄家兼操盘手英特尔始于 2006 年的 Tick-Tock\* 处理器发展模式像钟摆一样严格且精准的发生了十年，尤其确立了其在数据中心领域 x86 通用处理器的绝对垄断地位。（注\*，Tick 指半导体工艺（semiconductor process）升级年，Tock 指处理器核心架构（core micro architecture）升级年，即处理器的核心架构和制程工艺隔年升级，两年实现新架构和新工艺的更新迭代）。而十年后的当下，硅基半导体制程工艺逐渐逼近其瓶颈，曾经严谨的 tick-tock 钟摆停止，原计划 17 年 Skylake 一代之后的 CannonLake 取消，英特尔将延长每一代半导体制程生命周期为 3 年，改为 P-A-O 模式：即制程升级（Process）→ 架构升级（Architecture）→ 优化升级（Optimization）；各大媒体的忧心忡忡都在重复一个观点：摩尔定律举步维艰、越来越难以继。而阿里巴巴作为重度和深度用户，我们无意评说英特尔在工艺和制程方面遇到的挑战，我们更多回到基本盘从性能、能效、成本几个直接界面问题入手。即代与代之间 core 同质化（通常个位数性百分比能提升）、performance/watt（能效比）持续下降、

以及通用处理器大量存在的“水份”（例如普适业务场景 vs 阿里业务场景，各种 margin 等）。

限于篇幅我们在此不详细展开阐述完整的前因后果，简要提炼几条逻辑主线。

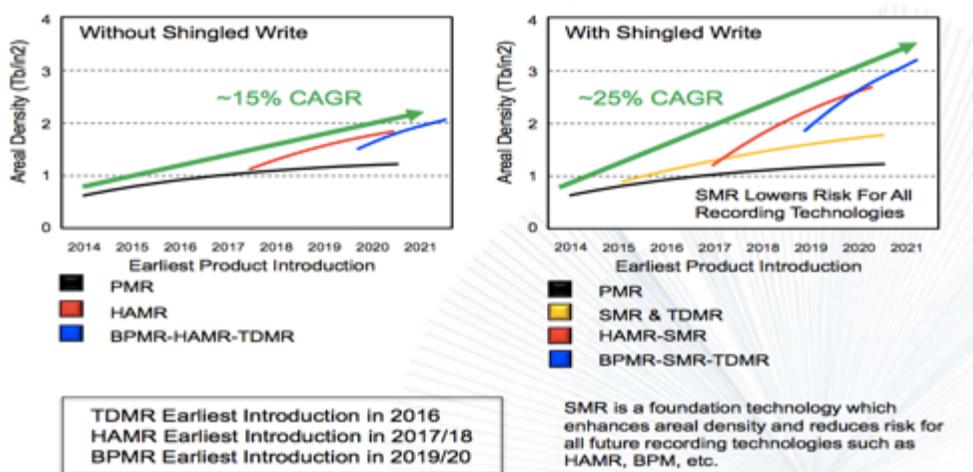
1. 集团 TCO 模型收益为导向硬件策略所要求的单机 scale-up 关键在于处理器性能、能效；
2. 结合阿里业务多样性的综合性能评估体系，创新性定义并重建 speccpu 测试指标及权重因子。即以阿里价值评估体系反向定价，而非 Intel 主导的普适评估体系普适定价；
3. 历时一年与 Intel 团队深入合作，实现业务级优化，以及性能功耗比 vs 上架密度 vs TCO 收益最大化机型置换比等多维度之间相对最优解。

该项目帮助阿里有效沉淀了 x86 处理器定制从 0 到 1 的宝贵经验，产出了 Broadwell 处理器性能功耗比 Top3 的高频高核心 AliCPU E5-2682 V4，工程节奏上实现了亚洲第一款定制处理器大规模应用于生产，严格同步 Intel 官方发布节奏，AliCPU 稳定承载了 2016 双 11 大促，成为保障奇迹背后的关键力量。

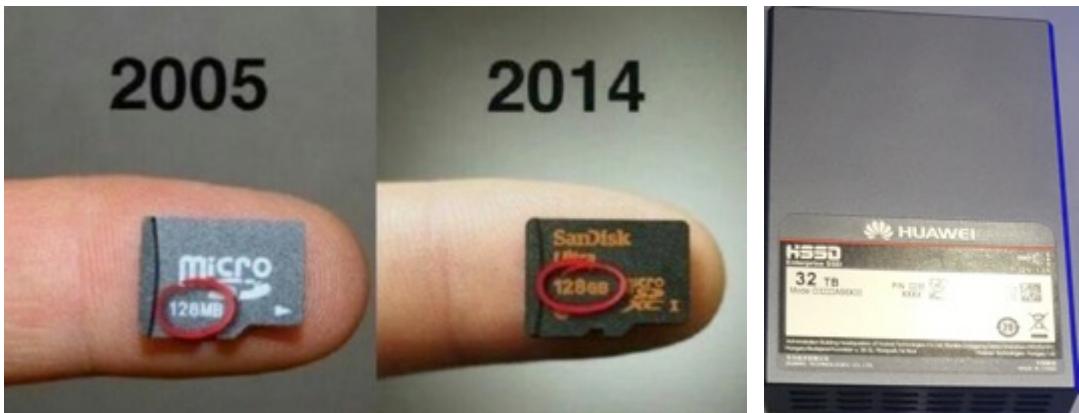


## 2.2 定制 SSD

先看两张来自互联网的图片，感受一下。



注：历时 50 年，预计 2018 年单 HDD 盘容量可达约 20~30TB ( 3~5 年 )，受限于机械硬盘结构 单盘 IOPS 上限大约在 250~300 接口吞吐不超过 300MB/s.

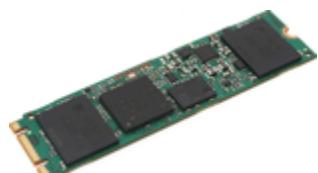


注：历时 10 年，闪存固态盘容量增长了超过 30 万倍，而 2017 年部分闪存盘容量可达到惊人的 60TB，部分已量产 PCIe 接口闪存盘更是突破了单盘百万 IOPS 的能力。

众所周知，闪存（Flash）介质的能力在过去 5 年中以超摩尔定律的速度飞速发展，无论是性能、容量、抑或寿命，从而在基础设施硬件层已经越来越成为变革架构释放新存储介质性能、系统能效比提升、以及整体 TCO 优化的战略机会点。阿里巴巴早在 3 年即开始了闪存介质的相关技术规划和布局。其中今年双 11 大促如丝般顺滑的用户体验就有 AliFlash（阿里自研 SSD）的给力表现，从规模应用承载大促的实践而言，阿里巴巴基础设层对于 Flash 介质的自主掌控力已经完成了从整到零拆解，首次实践了从零到整的重构的阶段。

我们同样从 know-why、know-what、know-how 几个层面来简要总结自研 SSD 之路。Know-why 层面从“如果不做如何保障”技术安全、供应安全、成本竞争力来辩证思考。

1. 核心技术、架构、源码自主掌控
2. 关键颗粒直采、生产质量及供应链管控
3. 软硬件打通、提升业务能力性价比和能效比、降低系统 TCO、以及支持最新技术



M.2



SATA 2.5 寸



NVMe U.2 盘



NVMe PCIe 卡



PCIe add-in 卡 ( host-based )

Know-what 层面我们规划了多层次的工程目标点和落地节奏，目前从 M.2、SATA 2.5 寸、NVMe U.2、以及 PCIe add-in Card 多种规格形态均已完成。其中 PCIe Card 形态 Aliflash 采用 Host-based 架构，即 SSD 的关键逻辑（垃圾回收，FTL 映射等）由 Host 主机通过驱动直接在内存里进行管理，因此只要修改 Host 端的 AliFlash 驱动即可进行软硬件联合调优，具有编译、验证、debug 方便的优点。接下来我们将以 host-based 架构 AliFlash 在软硬结合调优的 know-how 具体案例进行提炼分享。

- 1 容量 vs 寿命，够用就好。 CDN 二级 Cache 容量优化：CDN 的二级 Cache 需求大容量、低延迟、廉价的 SSD 方案。由于 CDN 整体业务压力较低，写入量较小，因此可以调整 AliFlash 的 OP，释放更多空间到用户区域。AliFlash 的 Nand 裸容量为 8.6TB 左右，标准用户空间为 6.4TB，通过调整 CDN 的 AliFlash 容量到 7.8~8TB，既满足 CDN 的性能、寿命需求，又显著提升了容量，降低了单位 GB 成本。
- 2 结合业务特征取舍，场景优化。 阿里妈妈 Tair-ldb 延迟优化：阿里妈妈从 C7 服务器升级到 H41 服务器（三块 AliFlash 卡），希望能

在读写混合的场景下，读的延时能有很好的保证（500MB/s 写入的同时，读延时 5ms 的超时率控制在万分之一以内）。由于业务的吞吐量要求不高，因此可以调优 AliFlash 的 IO 落盘路径，降低一定的吞吐量，用来提升延迟的 QoS。AliFlash 的 IO 落盘路径有两条，直接写 Nand 和 SRAM buffer write（延迟较低）。通过关闭直接写 Nand，强制所有 IO 写 SRAM buffer write，能使业务读 IO 超过 4ms 的比例仅为万分之 0.27，远远低于万分之一，同时也满足业务的吞吐量需求，此项调优超出业务预期。

- 3 软硬件协同 提升业务能力/性价比。集团 DB 原子写优化：MySQL 默认使用双写（double write）来保证数据的一致性，此特性会造成两次 IO，带来性能和容量的额外开销。通过开启 AliFlash 原子写功能，从硬件层面保证了业务写入 innodb 的 page 时，不会产生跨页写入，保障数据的一致性。由于关闭了双写，在高 IO 压力下，对于逻辑卷（LVM），响应时间降低了 8% 左右。对于裸盘，性能提高约 5% 左右，RT 上降低了 18%，约 0.3ms。另外，关闭双写降低了 30%~45% 的业务写入量，能显著提升 AliFlash 的使用寿命。

除了上述典型场景，AliFlash host-based 整体技术白盒化的掌控还能 enable debug 模式的个性化需求，极大提升了业务延迟分析、运维稳定性定位、寿命预警健康管理等产品化的配套能力。限于篇幅，该领域还有大量值得专题研讨的内容，恕不一一罗列。

在后续的规划中，AliFlash V2 家族将继续在新介质适配、核心控制器自主掌控、结合业务场景深度挖掘软硬协同提效方面保持高质量、技术先进性、配套完备的产品化交付物。

## 2.3 高效能存储解决方案

传统存储领域有一个非著名行业规则称为克莱德法则，它类似摩尔定律，可表述为每 12~18 个月同一价格的硬盘存储容量将会翻一番。与此类似在存储消费行业也流行着一种趋势判断，认为存储价格下降一半，则存储需求增加一倍。实际上无论是摩尔定律还是克莱德法则，其内生关联在于技术发展驱动了计算和

存储资源的单位成本持续降低，而云计算与大数据技术则持续推动了用户获取、使用、甚至二次开发这些资源的成本持续降低。

2015 年马老师最早提出 DT 时代研判，阿里巴巴是世界上最早研判数据将作为未来关键生产资料的商业公司，即云计算和大数据作为集团战略聚焦方向之一。前文提到与之一体两面的基础设施建设与竞争力构建上，基础设施团队必然将相对应的高效能存储硬件解决方案作为战略项目进行推进。貔貅（项目代号）产品化方案则是软硬结合优化高密度存储解决方案的典型，下文将从存储模型和系统架构设计两个维度来阐述。

勾勒存储系统的边界来看， $\text{TCO } \text{\textyen}/\text{单位存储容量}$  和  $\text{TCO } \text{\textyen}/\text{IOPS}$  吞吐是其中关键指标。细分存储系统数据访问频度的不同层次（例如热、温、冷、冰），在满足一定存储性能 SLA（服务等级承诺，service level agreement）前提下，关键设计目标聚焦在最优的  $\text{TCO } \text{\textyen}/\text{单位存储容量}$ 。

我们以 HDD 目标介质为例进行模型分析。首先约定边界条件，1) IDC 标准机架物理空间 40U，电力容量 8kw。2) TCO 简化为硬件采购成本 Capex + 分摊至每套系统的 Opex，后者包含 IDC 机架租金/kw/月，每网络端口成本/port 等费用。

其次定义三个维度系数，分别衡量物理密度、功耗利用率、有效成本系数几个指标。

- 存储密度系数 SDE (Storage Density Effectiveness)：硬盘数/单位 U 空间；例如 12x HDD/U 指每单位 U 空间内可容纳 12 片 HDD 的密度度量，越高则表示空间利用率越好
- 有效存储功耗系数 SPE (Storage Power Effectiveness)：有效存储介质的功耗/机架付费功耗；例如 0.7 指 8kw 中 70% 用于存储介质 HDD 的 operation 功耗消耗，越高则付费功耗利用率越好，理想模型趋于接近 1
- 有效存储成本系数 SCE (Storage Cost Effectiveness)：有效存储介质成本/总系统成本；例如 0.85 指系统成本每一元钱中，85% 用于采购有效存储介质 HDD，越高成本占比越优，理想模型趋于接近 1。

至此基于该模型我们简要进行如下量化对比，标准 2U 存储机型 vs 貔貅不同配置、以及 vs 理想模型之间差异。这几个抽象的系数本质上代表了系统架构设计在空间密度、能效比、适配业务特征进行资源能力定制几个维度之间进行求解相对最优解的过程。

	标准 2U	貔貅 1+1	貔貅 1+3	热插拔理想 模型 推演
硬盘数	12	38	78	108
单套功率	X	Y	Z	Z+
上架台数 @ 8kw (基于业务峰值负载功耗，确定上架密度)	20	15	9	7
总硬盘数	240	570	702	756
总部署 U 数	40U*	30U	36U	35U
有效存储密度系数 SDE in 40U ( Higher is Better )	6	14.25	17.55	18.9
有效存储功耗系数 SPE in 8kw ( Higher is Better )	0.26	0.61	0.75	0.80
有效存储成本系数 SCE in 整机 ( Higher is Better )	X	Y	Z	Z+

注：敏感数据以 X|Y|Z|Z+ 替代。

注\*：40U 已经达到标准机架的物理空间极限。即使 8kw 功率配额尚未用完，也无法提升上架密度。

当然实际系统架构设计还需要考虑满足业务需求 SLA 的适当计算能力、网络能力、内存容量等方面的配比，兼顾系统可运维性、部署供应颗粒度等某个维度的边界条件，而这些局部边界条件将会反向修正系统架构设计时在具体实现可行性上的 tradeoff。

下图是基于 Alirack 整机架一体化设计的貔貅节点计算机头与存储机尾俯视图。通过深度结合业务场景的取舍和优化 单位存储容量 TCO 优化比例高于 50%；模块化以及高密度的系统实现，叠加整机架一体式集成交付，极大提高了部署以及运维的效率；节点内部支持闪存介质与硬盘介质的硬件 tiering，便于为业务软件兼顾性能与成本提供灵活性。



貔貅机头



貔貅存储机尾 ( JBOD )

随着貔貅架构的逐步扩大规模应用、新介质技术的发展，新软硬件架构优化的持续演进；例如计算与存储分离架构演进中，存储集群的资源池化、存储能力服务化、存储侧专有计算能力嵌入化等课题仍然需要软硬件团队更深入的研讨，持续优化模型并拉通硬件模型和业务模型一体化思考，持续演进并不断提升业务体验和产品竞争力。

## 2.4 下一代通用底盘自研

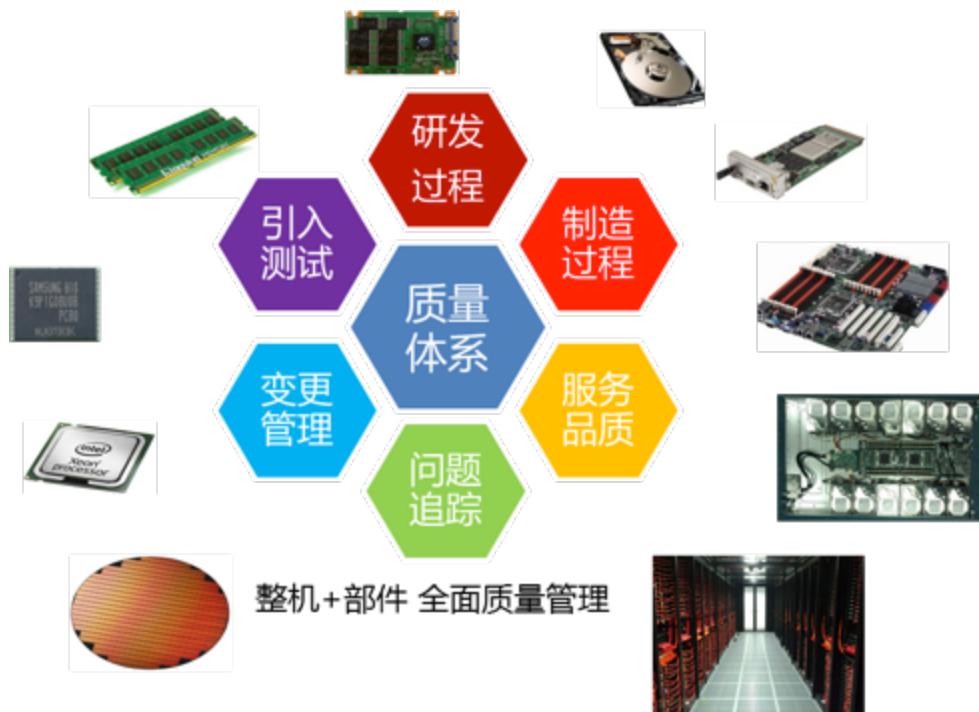
基础设施团队 4 年前开始启动“车同轨”、“书同文”的标准化工作，对应到服务器硬件领域则首当其冲为机型的收敛与标准化。从硬件基础设施的资源属性、成本属性、技术属性几个层面来看，2012 年第一期的标准化工作主要针对资源单元的标准化（系统规格、部件规格、运维界面的 feature 等），旨在提高资源准入、运维、流转等效率。而标准化本身是一项持续的活动，在不同的发展阶段也将有不同战略重点，因此具体的内涵和工作抓手也会不同。而本节提到的下一代（即 Intel Purley 平台，预计 17 年 7 月发布）平台通用底盘所处时期，我们将之定位在通过技术白盒化来确保质量和成本的白盒化，项目代号（雷神，Aliserver）。

呼应文章开篇提到的 RFC 1925 所表述 "Good, Fast, Cheap: Pick any two (you can't have all three)"，从标准化 vs 质量 vs 成本的维度辩证思考来看，前者是手段，后者是杠杆，中间是结果。雷神项目旨在打造完全自主知识产权的通用服务器底盘，实现三大目标：

- 1 ) Purley 平台最极致的配置灵活度以覆盖 90% 以上机型配置
- 2 ) 底盘目标成本管理
- 3 ) 同步 Intel 官方发布 Purley 平台即支撑业务规模上线。

而要达成上述三大目标则需要以端到端全流程质量管理活动标准化掌控为

前提，下图尝试提炼整个硬件体系全面质量管理活动的端到端全流程和关键环节。创新、成本、质量是硬件领域永恒的根本，雷神项目承载的 Purley 平台一代硬件质量大盘 将会在第九次双 11 中以实际担当诠释基础架构事业群“坚若磐石，精益求精”的组织使命。



## 总结

硬件系统涉及领域众多，相互依赖度复杂，研发及测试周期长，投入资源与产出效果存在显著滞后性。这些客观工程规律决定了基础设施层竞争力构建是一项长期、艰苦、技术密集的过程。同时面对技术快速发展的外部产业环境和集团业务与时俱进的内生需求迭代，主观上要求基础设施技术团队要沉淀出终身学习、不断反思、不断求变的意识定位和心态。

笃定客户价值和长期技术竞争力构建的初心，阿里巴巴基础设施技术团队承诺不断挑战自我、勇于创新、持续交付便捷高效的硬件基础设施解决方案。

# 1.3 阿里视频云 ApsaraVideo 是怎样让 4000 万人同时狂欢的？

作者：蔡华

## 前言

在今年的双 11 中，双 11 天猫狂欢夜的直播成为一大亮点。

根据官方披露数据，直播总观看人数超 4257 万，同时观看人数峰值达 529 万，在云端实现了高计算复杂度的 H.265 实时转码和窄带高清技术。其实不光是双 11，直播已经成为了 2016 年互联网最火爆的话题。除了内容的大规模涌现，背后其实是计算、存储、带宽的升级和成本的下降。而 ApsaraVideo 的出现，让直播的技术门槛更是迅速下降，可以使企业快速的上线自己的视频业务。

笔者作为参与双 11 视频云的工程师，希望通过本文和大家分享阿里云的 ApsaraVideo 如何为双 11 这个场景快速创建这种大规模直播场景的经验。

在传统的视频业务中，我们通常要面临这些技术痛点：

1. 一般需要部署上传服务、缓存服务、存储服务器、视频编转码服务、调度服务。
2. 硬件需要准备 IDC 机房、CDN 节点等一系列的硬件和设施。
3. 对于初创团队或者个人来讲，很难逾越这种技术和硬件设置带来的障碍。而最大的坑是耗时耗力，而且很难获得弹性。

在双11中，我们用阿里云ApsaraVideo获得了这些优势：

1. 快速上线：基于阿里云视频服务提供的客户端SDK和服务端openAPI，用户可以最快几天内上线一个大规模的视频业务。
2. 技术成熟稳定：基于阿里云视频服务多年的技术积累和沉淀，可以输出高质量视频编解码服务和稳定可靠的CDN分发服务。
3. 节省：ApsaraVideo服务秉承了云计算的特点，是按使用量付费的，比传统的自建IDC和构建CDN网络节省大量的人力和物力。

那么，如何用阿里ApsaraVideo搭建出视频服务？通常来讲，一个视频业务会分成3个重要的部分，我们以直播业务为例说明一下：第一，客户端上的功能，包括直播视频的采集、编码、视频流网络推送和播放，还可能会包括美颜、弹幕、连麦互动等高级功能。第二，业务服务端的功能，包括转码、截图、水印、直播间管理、直播流状态显示、直播的录制转点播、内容审核、BOSS系统等。第三，CDN网络部分，包括直播域名管理、域名加速、带宽流量扩展资源监控等等功能。

而阿里云视频服务在这3个部分，分别提供了客户端SDK，不但包含上述直播业务端上所需的功能，还做了很多的弱网环境的推流优化和首屏秒开等功能。直播openAPI，提供了开放的API接口，可以用来控制转码、截图、水印、管理直播间、观察直播流状态、配置直播录制等功能。对CDN网络的管理，提供了增删直播域名、分发加速配置和优化、带宽流量监控等功能。所以用户可以使用阿里云视频服务提供的SDK和openAPI快速方便地搭建自己的业务。

## 如何使用阿里云视频服务搭建一个视频业务

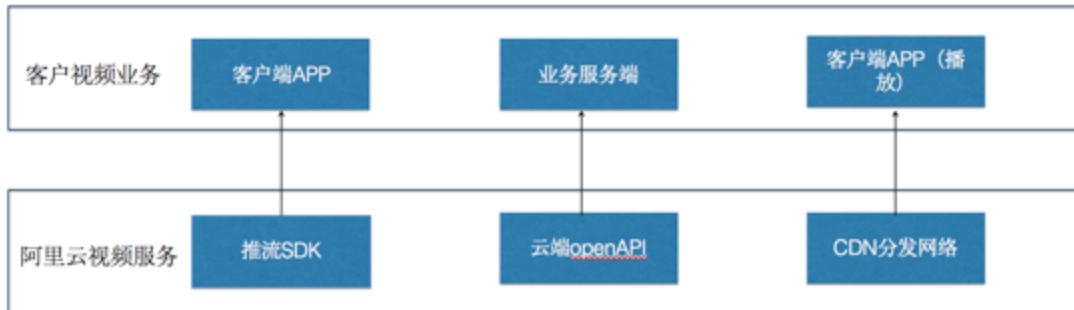




图 1. 阿里云直播端到端的解决方案

今年阿里巴巴组织的双 11 晚会，集结了大量的本年度当红明星加入，而且本次晚会在手机淘宝客户端、天猫客户端、优酷土豆客户端这三个有巨大活跃用户的流量入口进行直播。其背后所要承担的视频直播并发量可想而知。

在本次晚会直播中，我们采用了如下直播优化方案：

1. 直播推流端使用 H.264 码流推直播流到云端，服务端转码服务把 H.264 转成 H.265。
2. 然后经过 CDN 分发到各业务方，客户端播放器需要根据一定的标记来支持 H.265 播放。
3. 在视频云服务端对标准协议进行了扩展，来支持区分 H.264 码流还是 H.265 码流。
4. 增强了服务端转码服务模块来支持对 H.265 的实时转码。
5. 在网络线路以及直播中心都采用了各自的主备方案以及应急措施来保证晚会在各种突发状况下都可以顺利的直播。这个直播据实际演练的测算，在不降低清晰度的情况下。可以节省大概 30% 的带宽成本。
6. 提供内容的甄别的全套解决方案：直播云端服务除了标配直播必须的转码、录制、截图功能外，还增加了智能鉴黄、人脸识别、语音识别等人工智能相关的功能。
7. 随时动态扩展所需要的带宽资源，并且如果带宽峰值变化太大，还可以借助于阿里云在视频领域多年积累的经验，进行系统动态优化。

双 11 晚会的直播优化方案只是阿里云直播服务在这一年多以来做过上百场直播优化方案中的一个，随着直播在今年的爆发，我们的客户随时都会面临流量高峰的来临，使用直播云服务除了计算、存储、带宽资源可以动态扩展之外，阿

里云资深的直播云架构师也在每天帮助我们的客户优化直播系统的架构，以系统化的能力帮助客户度过一个又一个直播流量的高峰。

作为视频云服务，除了接入方便、对网络的压力可以从容的处理以外，对视频编解码的深入研究更是 ApsaraVideo 产品的核心技术，阿里云的视频技术专家在码率更小、视频更清晰的方向上已经进行了数年的研究和积累。

我们对观影体验一直以来都有着执着的追求，独家推出画质重生服务，集成了阿里巴巴与各大高校、研究所在视频领域多年合作的研究成果，让使用阿里云客户的视频观感得到质的飞越。经过画质重生处理，数十年前的怀旧老片也能提供高清画质播放，常规摄像机拍摄的 30 帧/秒视频影像也能提供 60 帧/秒的极致平滑观感，多次压缩造成的马赛克、移动拍摄造成的画面抖动，都可以被去除或缓解，大大改善观影体验。以下是我们推出的广电级的视频处理产品：

## 1. 高帧率视频重制

目前人们观看的大部分影视节目每秒都会刷新 24-30 帧画面，这样的帧率能够满足基本的观影需要，但在快速运动的场景中，30 帧/秒的影片播放起来存在可感知的顿挫感。随着视频行业的主流分辨率越来越高，普通帧率视频在播放时的顿挫感也越来越明显。在现下流行的 4K 电视上，60 帧/秒的刷新率已经成为了观影的基本需求。然而，受限于拍摄设备的性能，除了少量大制作的影片（例如《霍比特人》）采用了 48-60 帧/秒的拍摄技术，大部分影视节目源的帧率依然在 30 帧/秒以下，难以满足超高清视频观看的需求。因此，阿里云推出高帧率视频重制服务，无论是用户节目库里存放多年的经典剧集，还是最新拍摄的节目内容，任何普通帧率的片源都可以经过插帧算法，被重制为 60 帧/秒甚至 120 帧/秒的高帧率视频。借助这项服务，任何视频服务商都有能力提供极致平滑的高帧率视频观看体验。

## 2. 2K 转 4K 重制

眼下，4K 电视机已经逐渐成为家用电视的主流，4K 大屏内容确实能够在视觉感官上给观众带来巨大的冲击。然而由于目前片源的匮乏，购买 4K 电视机的

观众更多的还是在观看 1080p 及以下的视频内容。因此业内兴起了 2K 转 4K 服务，利用视频处理算法来将 1080p 影片重制为 4K 版本。目前大部分 2K 转 4K 的功能都主要依赖经典的超分辨率算法，因此市面上的伪 4K 片源的效果都大同小异，带有明显的人工制作痕迹——例如油画效果等等。阿里云推出的 2K 转 4K 重制服务，除了引入学术界最新的超分辨率算法外，还借鉴了阿里云在视频质量评估上的研究成果，经过影视行业的多年打磨，重制效果在业界首屈一指。



### 3. 片源修复

今天 1080p、4K 清晰度的影片已经逐渐成为主流，但很多怀旧老片虽然仍有庞大的受众群，但由于分辨率低、介质老化等因素，画质模糊且充满噪声。另一方面，即使是近年来的 UGC 内容，由于在互联网平台上过度压缩，含有大量的马赛克和毛刺，让观众难以接受。因此，阿里云针对这类受损片源推出了画质修复服务，通过深度学习网络，去除马赛克和噪声，恢复出抹掉的线条，让互联网上的低清片源重获收藏价值，让怀旧老片焕发新的生命力。

ApsaraVideo 产品完全秉承了云计算产品的所有特点，“按量付费”“共享经济”等，使视频这种表达方式不再只是影视行业的专属，使用视频云服务，普通的企业和个人也可以很容易拥有视频这种更直观的表达方式。

## 第二章 存储

## 2.1 永不停止的脚步——数据库 优化之路

作者：佳毅

### 前言

2016 年双 11 已经顺利落下帷幕，在千亿电商流量的冲击下，集团数据库整体表现完美。完美表现的背后，隐藏着数据库团队对技术的执着追求。这是一个什么样的团队，他们究竟做了什么，是什么支持着双 11 这一全民狂欢的数字一次次突破？笔者以一个亲历者的角度来给大家揭开双 11 背后，阿里巴巴数据库团队的神秘面纱。

### 1 我们是谁

---

1. 我们是阿里巴巴集团基础架构事业群-数据库技术团队，我们团队维护着阿里巴巴集团、蚂蚁金服、菜鸟网络所有的数据库，是整个大阿里系，坚强的存储中台。

2. 我们是阿里巴巴集团神秘的 MySQL 分支——AliSQL 的开发者，传说中的 AliSQL 拥有者远超原生 MySQL 的性能，并且有着为阿里巴巴电商、金融业务场景量身定制的优化。

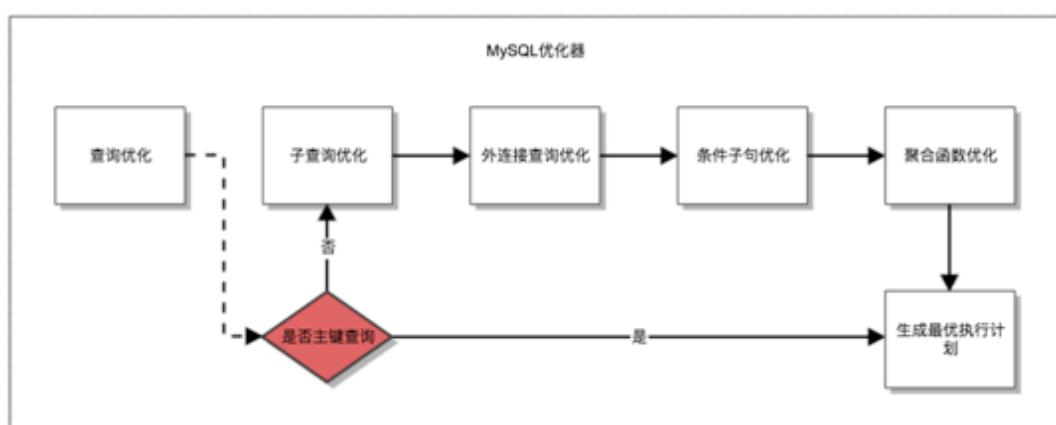
3. 我们为整个阿里巴巴集团的异地多活提供底层基础架构——DRC。DRC 提供超远距离（中美超过 10,000 公里），超低延时（中美同步延时小于 500ms）的数据库同步订阅服务。

4. 我们致力于将阿里巴巴的优秀技术和沉淀，通过云产品的形式提供给广大公有云的用户，如 DTS——数据传输，DMS——数据管理，CloudDBA——用户身边的数据库专家。

## 2 交易查询优化

在 2014 年双 11 时，交易的 AliSQL 秒级查询量已经到了一个天量。虽然当天交易系统一切平稳，但交易的数据库负责人深知这数字背后隐藏的风险。

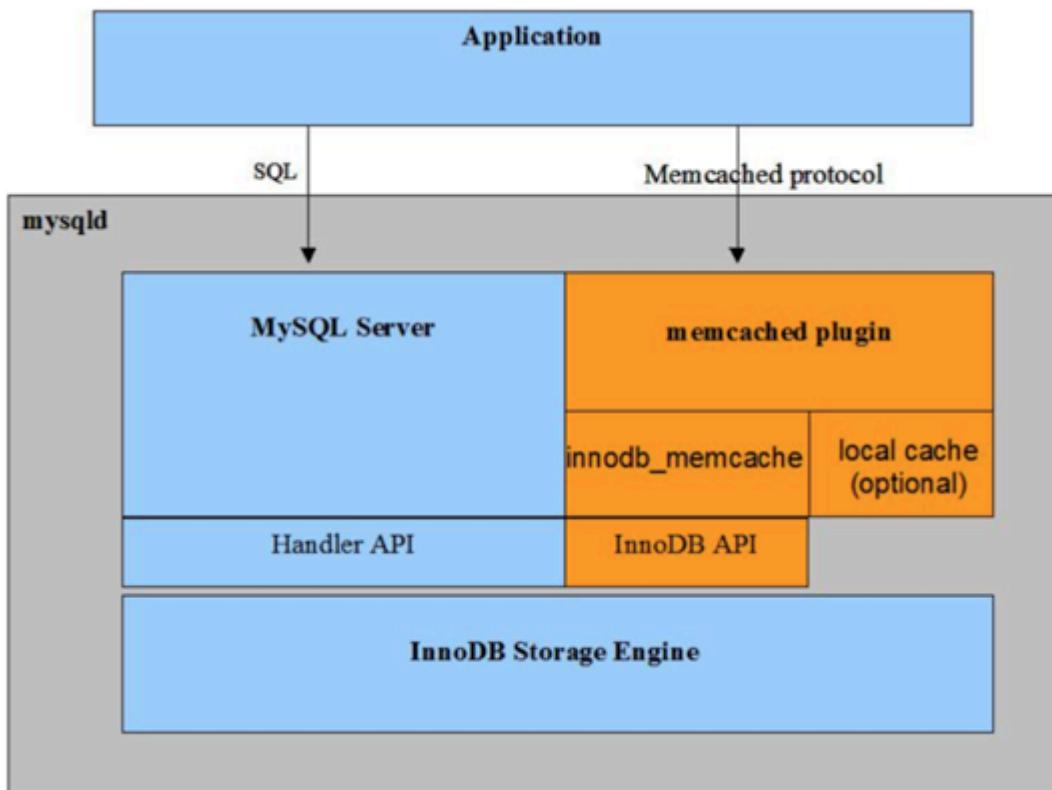
为了保障交易系统下一年的稳定，数据库团队在 2015 年花了大量的时间，深入了解业务特点，分析每一次查询 SQL，最终产出在优化器中注入 PK\_Access 的优化方案。



当 2015 年双 11 来临时，交易的秒级查询量再次达到一个峰值，这一切都在掌握之中。PK\_Access 优化方案让交易库的查询能力提升了 27%，RT 下降 48%。

光鲜数据的背后，不只是一个数据库优化方案的结果，更多的是交易业务系统一堆人花了大量的时间梳理接口并确定流控阈值的结果。这里有这么一个数据，2015年交易系统的对外限流预案一共有125个。回想起交易同学的一句话，“如果明年还需要这样，我肯定疯了”。业务同学的话让我们反思，系统能力的提升不能建立在业务缺失的基础上。

再次回到 pk\_access 的方案，交易的查询在目前的条件下，已经做到了最优，server 层的 sql\_parse 是瓶颈点，但是在 sql 的方式下，没有办法提高更多了。那么能不能将 sql\_parse 也省掉呢？这个时候我们想到了被人遗忘的 MySQL 插件，InnoDB memcached plugin。



由于插件本身的使用场景偏少，且自身的功能缺失，一直并未广泛使用。换句话说，这是一块未被开发的沃土。从原理上证明这是一个有效方案之后，数据库团队马上集中精力，说干就干。最终修复 bug 15 个，新功能开发 6 个，才满足了交易的基本查询。测试中发现，在完整交易场景下，memcached plugin 和 SQL 的对比结果来看，memcached plugin qps 可以达到 42 万，SQL 才能达到 12 万，qps 提升了接近 4 倍。在单 SQL 场景下，交易单条查询接口的 rt 下降约 30%。在相同流量的引流测试中，通过 memcached plugin 查询，应用

load 4.8 , cpu 45% ; 通过 sql 查询 , 应用 load 13 , cpu 72%。通过各个维度的比较 , memcached plugin 带来的收益远远超出每个人的预期。

就在刚刚落幕的 2016 年双 11 , 交易系统秒级查询量突破千万 , 再次刷新秒级查询量 , 也是数据库团队第一个秒级破千万的系统。这一次的增长可谓是 “ 爆炸式 ” 的。

从今年交易数据库运行情况来看 :

- 数据库不扩容 , 整体压力翻翻 , 并且仍有一定的余量。
- 业务降级预案大幅减少 , 做到对业务无损。
- 交易应用服务器扩容大幅减少。业务访问数据库从 SQL 接口切换到 Memcached 接口之后 , 还带来的一个显著优化就是应用服务器的 CPU 开销大幅降低 , 这也导致最终需要扩容的应用服务器数量大幅减少。

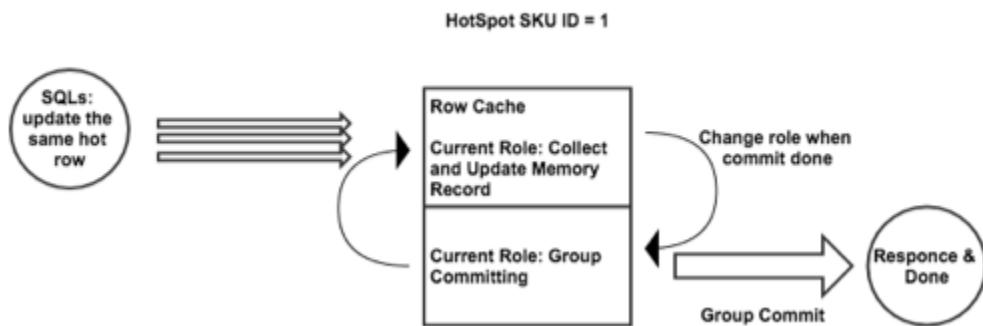
### 3 库存中心扣减能力提升

热点库存 , 相信无论是有多年双 11 备战经验的技术同学 , 还是资深的剁手党 , 对这个名词都不会感到陌生。因为每年双 11 , 总有那么些爆款商品特别热销 , 大家的抢购热情异常高涨 , 这几年双 11 , 小米 / 华为手机、优衣库衣服等 , 都曾经加入过这个热点商品的家族之中。其实 , 热点商品是一直存在的 , 但是热点商品带来的问题 , 我们是 2012 年底 2013 年初才真正意识到的。

从那时起 , 2013 、 2014 、 2015 、 2016 , 每一年我们在 AliSQL 内核层面针对此问题都有优化 , 优化单 Key ( 单商品 ) 的秒级扣减能力。但是 , 只有今年 , 我们将此问题做到了技术上的极致 , 未来可以预见的几年内 , 库存热点问题 , 将从双 11 备战中消失。

2013 年 , 2014 年和 2015 年 , 库存中心随着 AliSQL 版本升级和硬件升级 , 轻轻松松渡过了双 11 的技术大考 , AliSQL 的单行扣减能力 , 从原生 MySQL 的每秒 500 笔 , 提升到了 2015 年的每秒 5000 笔。

轻松并不等于放松，阿里人绝对不会满足现状，如何才能继续提升单行扣减性能？答案是批处理！我们认真思考了 2013 年做的优化，它们都有一个共同点，就是在串行的基础上进行的，而且已经做到极致。于是我们想到了是否可以通过批处理来提高吞吐。2016 年经过半年的闭关开发测试，“史诗级怪兽”补丁 hotspot 横空出世；



通过添加两条流水线轮流提交，并且使用 group commit，在双 11 来临之前的一次测试，我们成功的将单行热点扣减能力提升到了 10 万，有 20 倍以上的提升，并且可实现自动的热点识别，不影响非热点商品的正常扣减。在跟库存应用完成对接之后，几次全链路压测，AliSQL 的热点处理能力甚至远高于下游的处理能力。

2016 年双 11，数据库使用新版本的补丁平稳运行。按照双 11 实际的表现来看，未来几年内，库存热点问题将彻底告别双 11 的历史舞台。

## 4 数据库大中台全面升级

数据库最重要的是数据和 SQL，数据表示结果，SQL 表示过程，两者缺一不可。

在 2011 年初，数据库第一代监控体系 tianji 成型，其中包含了基于定量收集网络包的 SQL 采集工具 MyAWR，标志着第一代数据库 SQL 采集体系正式形成。这种方式是通过 TCPDUMP 抓取 1 万个网络包，然后将抓取到的网络包解

析成 SQL 语句，最后算出单位时间内的 SQL 数，反推出某时间段内的整体 SQL 情况，用公式表达为：

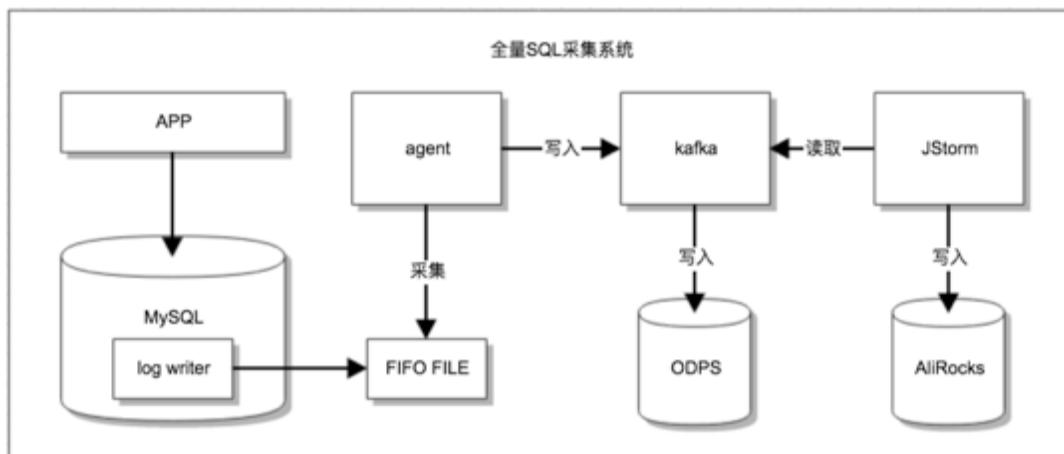
$$\text{每分钟 SQL 数} = \frac{\text{每万个网络包的 SQL 数} * 60 \text{ 秒}}{\text{每万个网络包的耗时}}$$

这种方式存在一个非常大的弊端，就是准确度的问题。But better than nothing。这种方式一直持续到 2013 年。

2014 年至 2015 年 数据库团队自研 MySQL 插件 DAM( DataBase Activity Monitor )，用于记录数据库的执行事件，从安全方面来对数据库异常活动进行监测和审计。但后面由于资源和使用的问题，DAM 项目失败，继续沿用着 MyAWR。

2016 年，倔强的我们再次提出全量 SQL 采集的目标，并且要求双 11 当天也能采集，采集内容不仅只是 SQL 文本，还需要了每一条 SQL 的运行时间、扫描行数等关键信息。换句话描述，就是要求数据库以 INFO 级别打印数据库日志。这是一次史无前例的挑战。设想一下，当一个数据库在 1 万/秒的请求压力下，还要 1 万/秒的输出日志，这无论对磁盘和数据库本身都是非常苛刻的要求。

这一次团队内高度合作，有团队保证高性能的全量 SQL 输出，有团队保证高效的全量采集，有团队保证海量存储，有团队保证精准分析和使用。最终完成了第二代全量 SQL 采集系统。

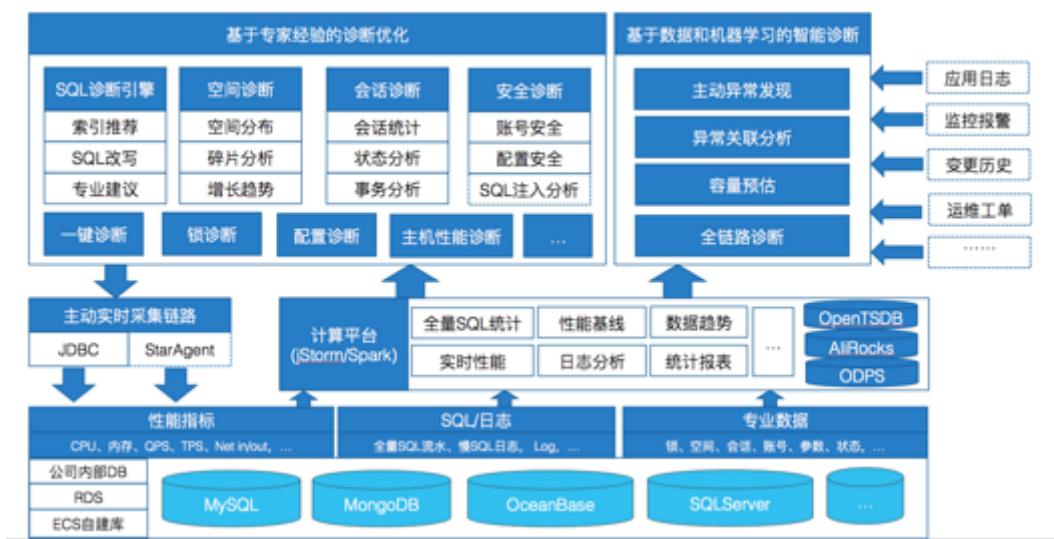


在 ODPS 上，我们可以进行全面的 SQL 分析计算，如 10 分钟内有多少用户连续下单等问题。在 AliRocks 上，我们可以精准的给出 SQL 直方图，如 TOP SQL 问题。

2016 年双 11 当天，我们打开了集团交易核心集群的全量 SQL 采集功能。当天全量 SQL 输出对数据库仅有 5% 不到的影响，并且实时计算峰值处理速度为每秒千万级别，平均处理速度每秒百万级别，全天实时性能计算延迟平均在 100ms 以下。全量 SQL 流水目前都存入 SLS/ODPS 中，实时性能计算数据存储在 AliRocks 中。这是我们第一次真正获取到了双 11 100% 的 SQL 信息，为今后留下了一笔宝贵的财富。

今天，基于全量采集的 SQL 信息，我们的 CloudDBA 不仅将全部 SQL 流水存储做离线分析，双 11 当天我们还通过实时计算分析出 SQL 实时性能数据。有了这些基础数据只是开始，后面我们会基于这些基础数据借助数据挖掘和机器学习等手段，帮助用户更多维度地理解业务数据趋势，提供主动的异常发现/诊断，资源预估等更丰富的数据库服务。帮助用户不仅能够通过 CloudDBA 自助地诊断和优化自己的数据库，还能够通过 CloudDBA 更好地理解业务如何使用数据库以及如何用好数据库。

## CloudDBA - 用户身边的数据库专家



同样在双 11 负责传输海量数据的中台另一重要基础设施，就是 DRC ( Data Replication Center )。DRC 负责解析增量数据库日志，分发给大数据、搜索和应用，同时负责异地多活数据同步。随着实时数据计算业务的增长，DRC 任

务数增长迅速，今年 DRC 集群架构做了重大突破，能够单集群支撑数万个数据同步进程，对外自动化接入 API 响应时间减半，通过 akka 框架使得集群容灾速度得到极大提升，通过合并一个 Region（地区）的集群和自动负载均衡降低整体运营成本，以更低的成本和更高的稳定性支撑了更大的双 11 峰值压力。

在性能上，DRC 解析核心做了很大的技术优化，同时在同步链路上通过 bucket 冲突算法优化和热点合并技术提升了峰值同步能力，在分发链路通过连接池化技术，使得单个解析存储进程可以同时分发给上千个下游用户，对业务快速接入和低成本共享数据通道的能力打下坚实的基础。DRC 支撑了异地多活下，多个异地机房间的数据亚秒级同步，在双 11 这样的峰值压力下，依然保障了 500ms 以内的数据库端到端同步延时。

2016 年 DRC 除了在性能和规模支撑能力得到提升，我们还在数据质量上精益求精，今年 DRC 全面开启了行校验做数据同步，支持数据加密传输。对于上游数据源的变化做到了自动化联动，包括数据库的迁移和拆分，使得订阅实时数据的下游业务完全无需感知。DRC 在队列存储上开放了完整的 SQL 查询能力，使得下游用户可以通过 SQL 很快查到 1 条记录，在 DRC 通道里是否存在，快速定位数据质量问题。同时这个能力支持了部分用户做数据对账和灵活轻量的增量报表，拓展了 DRC 的服务能力。可以预见，DRC 在未来将给业务带来更多超越预期的价值。

## 总结

以上几个例子仅仅只是阿里巴巴数据库团队在今年双 11 中众多优化中的几个，同样还有很多优秀的内容无法一一为大家道来。正如开篇提到的一样，数据库团队是一个有追求的团队，面对技术，我们追求极致，面对未知，我们勇于挑战。

2016 年双 11 已经过去，但是数据库团队已经放下了过去取得种种成绩，准备重新出发了，2017 年 AliSQL5.7、AliSQL X-Cluster（基于三副本 Paxos/Raft 协议的 AliSQL）、AliRocks（基于 RocksDB 存储引擎的 AliSQL 分支）、Ceph On AliSQL（存储计算分离）、DBPaas，CloudDBA 等等每一件事都已经在路上，而且样样极具挑战。

海到无边天作岸，山登绝顶我为峰，如果正在阅读的你热衷新事物，追求极致，敢于挑战，那么就不要犹豫，赶快加入我们。

## 2.2 阿里云 ApsaraDB——双 11 商家后台数据库的基石

作者：玄慚

### 前言

2016 年天猫双 11 购物狂欢节已经完美落下帷幕，千亿成交的背后，作为整个天猫商家后台数据库的基石，阿里云 ApsaraDB 是如何保障在零点洪峰来临时候稳定、安全和顺畅？如此庞大規模的数据库实例集群又是怎样一步步成长起来的？

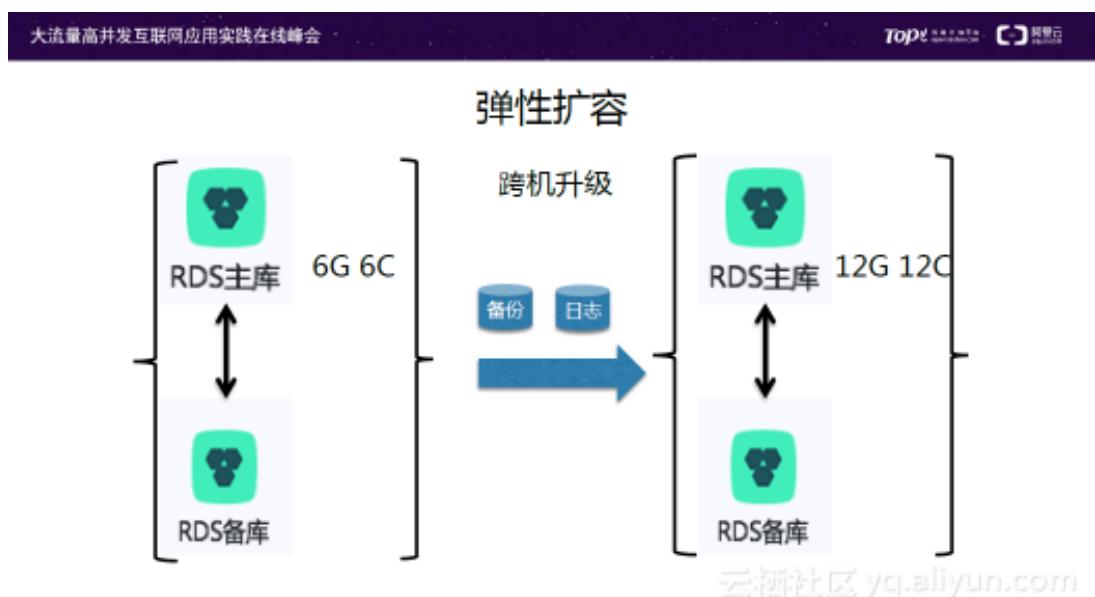
ApsaraDB 团队核心老司机玄慚，为你带来，双 11 是这样用云的姿势....

### 1 弹性扩容



多数用户在双 11 到来之前都会进行弹性扩容，常见的弹性扩容分为两类：本机升降级和跨机升降级。例如现在有一个 6G/6C 的 RDS 数据库想要升级到 12G/12C，如果本机资源足够，则可以在本机完成升级，无需迁移到其他机器上。云数据库默认是主备架构，本机升级时资源系统首先判断升级是否可以在本机完成，工作原理如上图所示：首先升级 RDS 备库；然后重启备库；之后进行主备切换，再修改重启原主库。

将本地升级变成一次主备切换，进而避免了重启主库的操作。这里需要注意的坑是：如果主备有延迟，那么主备切换不会进行，升级任务也会被 block。



另一种弹性扩容的方式是：跨机升降级。当本机资源不足以支撑升级所需要的资源的时候，需要将实例分配到另外一台机器上。所以跨机升级需要使用数据库最近一次的备份和日志实时同步到新的主机上，保证新实例和旧实例的数据是完全一致的。

而这里需要注意的坑是：如果历史备份集较大或原主库压力较大时，会导致跨机迁移时间较长。



## 弹性扩容最佳实践

- 1 **为什么有时候升级需要很长时间？**  
可能发生了跨机迁移，迁移时间受限于数据库大小以及系统压力
- 2 **可用区迁移、数据库版本升级为什么耗时较长？**  
这两者迁移都会发生跨机迁移
- 3 **空间升级为什么非常快？**  
空间升级不用重启迁移数据库
- 4 **选择弹性扩容的时间**  
建议在业务低高峰期，最近一次备份任务完成后进行升级

云栖社区 [yq.aliyun.com](http://yq.aliyun.com)

弹性扩容最佳实践可以总结为以下四点：

1. 如果升级很长时间也没有完成，可能发生了跨机迁移或者主备存在延迟。
2. 可用区迁移、数据库版本升级耗时通常较长，是因为两者迁移都会发生跨机迁移。
3. 空间升级非常快，这是因为空间升级无需重启、迁移数据库，对业务也不会造成影响。
4. 弹性扩容时间的选择，建议在业务低高峰期进行弹性扩容。

## 2 访问链路



### 访问链路



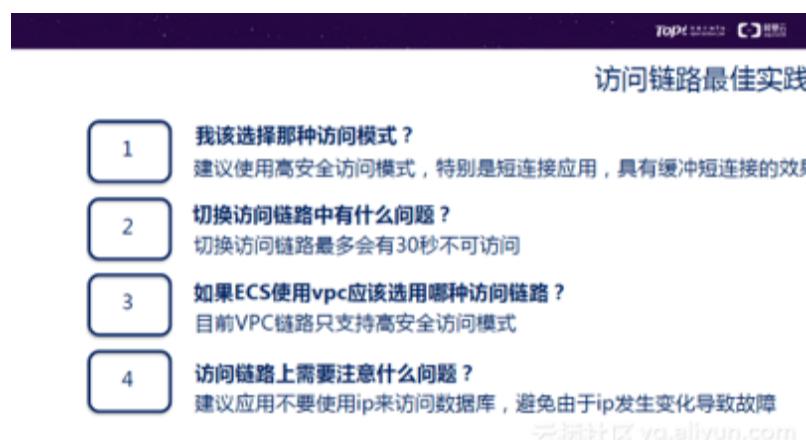
**高安全访问链路：**  
防止 90% 的连接闪断和 SQL 拦截的能力  
支持内外网地址同时访问  
对短连接应用有防护作用  
增加 5% 左右的响应时间

云栖社区 [yq.aliyun.com](http://yq.aliyun.com)

在云数据库中，访问链路分为两种模式：高安全访问链路和标准访问链路。在图上流程图的右侧，RDS 在数据库的前面增加了一层代理层，所有请求在代理层都被解析，在解析过程中添加了 SQL 拦截规则，进而可以防止 SQL 注入的攻击。此外，高安全访问链路可以防止 90% 的连接闪断；并支持内外网地址同时访问；对短连接应用有缓冲防护作用。需要注意的是高安全访问链路较标准链路增加了 5% 左右的响应时间。



标准访问链路与高安全访问链路相比，缺少了代理层，进而失去了连接保持、SQL 拦截以及内外网同时访问的能力；但相对于高安全访问链路响应时间会减少。

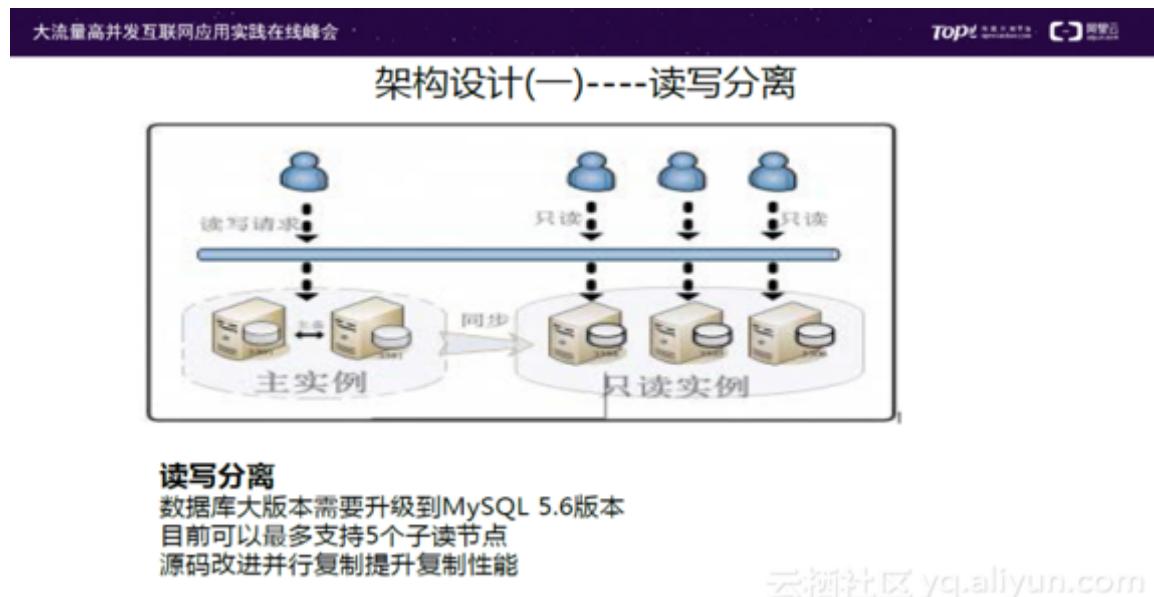


因此，访问链路最佳实践可以总结为以下几点：

1. 建议使用高安全访问模式，特别是短连接应用，高安全访问模式具有缓冲短连接对数据库冲击的效果。
2. 在标准访问链路切换到高安全访问链路时，切换过程最多会有 30 秒不可访问。
3. 如果 ECS 使用 VPC，那么数据库只能选择高安全访问链路。
4. 访问链路上需要注意应用不要使用 IP 来访问数据库，避免由于 IP 变化导致故障。

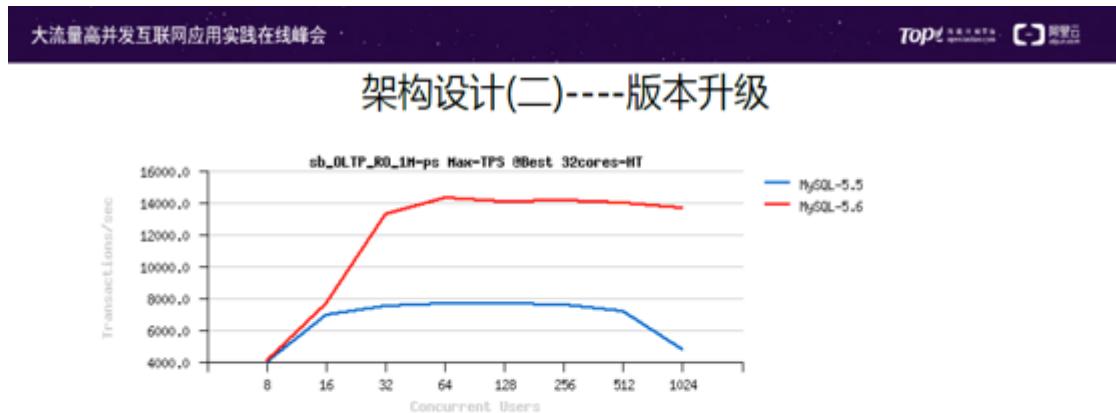
## 3 架构设计

架构设计就像我们修建一幢坚固的房子一样，需要有整体的布局设计，同时在细节上材料的选择以及施工质量的保障也同样重要。在历年的双 11 中，由于业务流量的突增，那些平时没有暴露出来的问题往往在这个时候爆发出来，所以我们要把数据库这块地基打好，细节上做好，架构设计就需要我们在这些上下功夫。



读写分离是常见的架构设计手段。RDS 支持只读节点，主库主要承担写和实时性要求高操作，一些复杂的分析计算业务操作最好不要放在主库上执行，而是选择放在只读节点运算。使用读写分离架构时，首先数据库版本需要升级到 MySQL5.6 版本；同时目前 RDS 最多只支持五个只读节点。在读写分离时，延

时是我们必须关注的重点，目前 RDS 上通过源码改进并行复制 提升复制性能，降低了主库与备库之间数据同步的延迟。

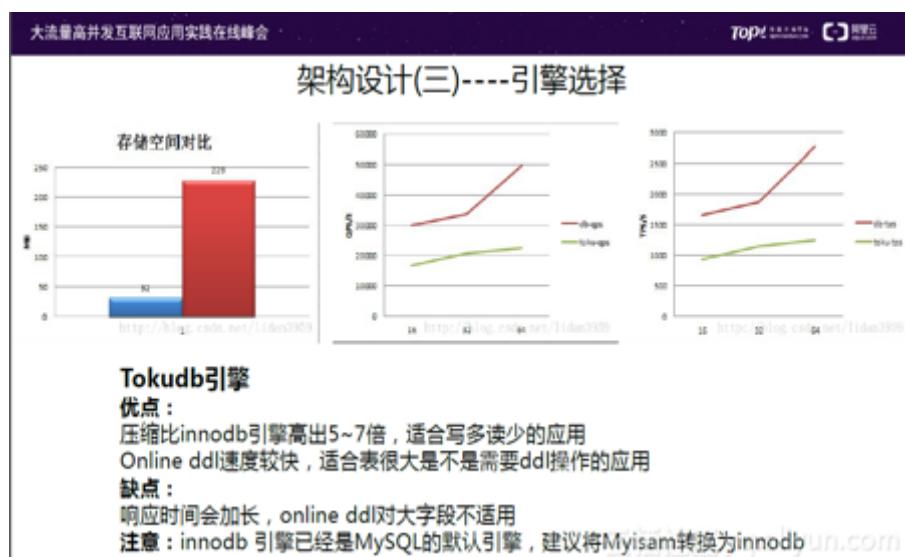


### 升级版本到5.6

5.6版本支持建只读实例，应用可以做读写分离  
支持在线添加字段，索引和重建数据表，应用不在被阻塞  
性能和稳定上较低版本有显著提升

云栖社区 [yq.aliyun.com](http://yq.aliyun.com)

正如上图的压测结果显示，5.6 版本较 5.5 版本，在性能上有很大的提升。目前，RDS 只有 5.6 版本支持只读实例，应用可以做读写分离；支持在线添加字段、索引和重建数据表，应用不再被阻塞。



引擎选择是数据库设计中很基础的一点，这里重点介绍下 Tokudb 引擎。日志型应用的特性是：写操作很高、读操作相对较少。Tokudb 引擎压缩比

Innodb 引擎高出 5~7 倍 ,适合写多读少的应用 ;同时 ,Tokudb 引擎 online ddl 速度较快 ,适合表很大需要经常 DDL 操作的应用。同样 , Tokudb 引擎缺点也十分明显 ,它会增加响应时间 ;同时 online ddl 对大字段不适用。

这里需要注意一点的是 :在 5.5 版本以后 innodb 引擎已经是 MySQL 的默认引擎 ,建议将 Myisam 引擎转换为 innodb 引擎 ,会避免很多问题的发生。

**架构设计(三)----字段设计**

**案例一：大字段**

```
CREATE TABLE `test` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `col1` blob,
  `col2` blob,
  `col3` blob,
  `status` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `ind_user_id` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin

update test set status=1 where id=100;
```

**最佳实践：**

数据库的更新写入压力过大 : update , insert , delete , 导致binlog日志急增  
使用大字段 : varchar(8000),text,blob,clob(sqlserver/mysql)  
将大字段拆分出主表或者存入到其他存储系统中

云栖社区 [yq.aliyun.com](http://yq.aliyun.com)

对于大字段 ,数据库的更新写入压力过大 , update、insert、delete 会导致 binlog 日志急剧增加 ,导致实例磁盘报警。因此在数据库设计时 ,要注意规避大字段引起的问题。常见的大字段有 varchar ( 8000 ) 、 text、blob、clob ( sqlserver/mysql ) , 使用时建议将大字段拆分出主表或者存入到其他存储系统中。

## 架构设计(三)----字段设计

### 案例二：字段类型

```
CREATE TABLE `test` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `ind_user_id` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin

select * from test where user_id=100;
```

### 最佳实践：

发生隐式转换，会导致索引无效，可以使用通过执行计划查看是否使用到索引

常见的隐士转换：字段定义为字符，而传入条件为数字

在设计开发阶段就要避免数据库字段定义与应用程序参数定义出现不一致

云栖社区 yq.aliyun.com

字段类型也是常见的问题之一。如上图所示案例中表的 user\_ID 是 varchar ( 64 )，但访问 SQL 传入的是数值类型，这就会导致隐式转换发生，进而导致索引无效，可以使用 explain 查看是否使用到索引。因此，在设计开发阶段，就要避免数据库字段定义与应用程序参数定义不一致的情况。

## 架构设计(三)----字段设计

### 案例三：字段大小

```
CREATE TABLE `test` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `status` int(11) DEFAULT NULL,
  `user_name` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `col1` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `ind_user_id` (`user_id`),
  KEY `ind_user_name` (`user_name`(255))
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin

select * from test where user_name='xuanan';
select * from test where user_name='xuanan' order by col1;
```

### 最佳实践：

字段长度超过索引允许的最大长度会导致索引字段被截断

过长的字段定义会消耗大量的排序内存以及临时表空间

云栖社区 yq.aliyun.com

字段大小同样会对数据库性能造成影响。字段长度超过索引允许的最大长度会导致索引字段被截断；同时，过长的字段定义会消耗大量的排序内存以及临时表空间。

## 架构设计(四)----索引设计

### 索引设计思路 ( 单条SQL的创建索引思路 )

```
select person_role_id from moive where movie_id=1000 and role_id=1 order by nr_role desc;
```

1-评估出参与运算的结果集范围 :

```
select person_role_id from moivewhere movie_id=1000 and role_id=1 order by nr_role desc;
```

索引 : alter table movie add index ind\_movie(movie\_id,role\_id);

2-参与排序的字段

```
select person_role_id from moive where movie_id=1000 and role_id=1 order by nr_role desc;
```

索引 : alter table movie add index ind\_movie(movie\_id,role\_id,nr\_role);

3-覆盖索引

```
select person_role_id from moive where movie_id=1000 and role_id=1 order by nr_role desc;
```

索引 : alter table movie add index ind\_movie (movie\_id,role\_id,nr\_role, person\_role\_id);

索引设计也是大家经常犯错的一个点，在历年双 11 保障中，索引出现的问题最多。这里，重点讲解单条 SQL 的创建索引思路：

```
select person_role_id from moive where movie_id=1000 and role_id=1  
order by nr_role desc;
```

对于这条 SQL 语句，首先需要评估参与运算的结果集范围，在该语句中创建 movie\_id 和 role\_id 的组合索引；第二步，考虑参与排序字段，在该语句中，排序用到的是 nr\_role，因此需要将其添加到索引中。大部分情况下，经过前两步，就已经完成了索引的创建。

有时候，还需要考虑第三步：覆盖索引，在索引中添加需要查找的字段，无需回表，以期达到优化目的，在该例中将 person\_role\_id 添加到索引中。

常见的索引误区包括：

- 对 SQL 语句的每个查询条件字段建立一个单列索引，MySQL 只能使用其一个索引；
- 对 SQL 语句的所有查询字段建立组合索引，导致索引远大于数据，同时性能低下；
- 小表不建立索引。

[if !supportLists]

## 4 高可用配置



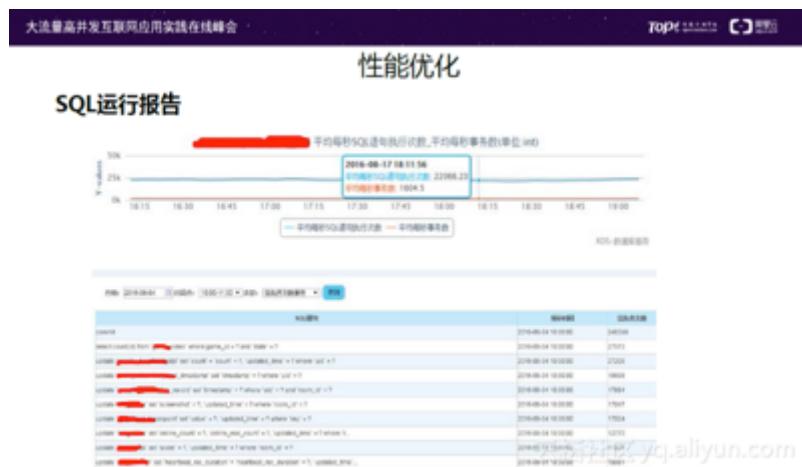
RDS 本身是一个主备的高可用架构，当主库 Down 后，会快速切换到备库。在高可用架构中很重要的一点是数据同步，保障主备数据一致不丢失。常见的高可用配置包括：

- **单可用区**：主备都在统一一个可用区内，可以实现主备之间的快速切换；
- **Binlog 同步**：采取异步和半同步的方式保障主备的数据一致；
- **Binlog 刷写**：根据应用特点设置安全模式或者高性能模式；
- **事务提交**：默认采用最高安全模式。

[if !supportLists]

此外，为了保障服务高可用，也可以采用多可用区配置，即主备在不同可用区，此时，应用同样需要多可用区部署。此时需要注意 Binlog 在主备的同步模式，通常这种情况下开启半同步模式跨可用区访问，可能导致写入性能下降。另外，还有一种跨数据中心的灾备方案，在历年的双 11 中，已经有很多用户实施过这样的方案，你可以选择在两个不同的数据中心部署数据库和应用，比如在杭州和上海两个地区部署，两个数据中心的数据同步采用 DTS，以保证一个数据中心挂掉后，另外一个数据中心能够接管起来。

## 5 性能优化



当性能问题出现时，例如上图所示数据库的 QPS 高达 2W+，这时候如何进行优化？首先我们需要明确导致 QPS 过高的原因，可以查看 SQL 运行报告，对一段时间内的 SQL 语句进行归类排序，这样就知道了数据库中是那些 SQL 导致 QPS 提升的，然后针对这些 SQL 进行分析，对应地给出解决方案，判断调用是否合理，是否添加缓存等。性能优化中，慢日志也是需要重点关注的点。通过查看慢日志运行报告，分析这些慢 SQL 产生的原因：是否缺少索引、字段设计存在问题等等，在双 11 之前优化掉，避免双 11 高峰来临的时候引发雪崩。

## 6 参数优化



在 RDS 中，大部分参数是已经经过调优的，因此很多参数是不需要再去调整的。但是用户可以根据应用场景的不同选择合适的参数，这里重点看下 RDS 新增的四个参数优化：

- `rds_max_tmp_disk_space` 控制 MySQL 能够使用的临时文件的大小，适用于一个 SQL 就消耗掉整个数据库的磁盘空间；
- `tokudb_buffer_pool_ratio`：控制 TokuDB 引擎能够使用的 buffer 内存大小，适用于选择了 tokudb 作为存储引擎的场景；
- `max_statement_time`：控制单个 SQL 语句的最长执行时间，适用于控制数据库中的慢 SQL 数量；
- `rds_threads_running_high_watermark`：控制 MySQL 并发的查询数目，常用于秒杀场景的业务；

[if !supportLists]

除此以外，阿里云的异地灾备的产品化也非常值得分享。从 2015 年起，RDS 为天猫的商家后台数据库提供了异地灾备的功能。当主机房出现较大的负载压力、断网、断电等极端情况，RDS 可将商家的后台系统在 30 分钟内切换至灾备机房继续运行，以保障总体可靠性，进一步确保平台大型品牌商家双 11 期间后台系统安全、稳定。

## 7 未来，走出去传承最佳实践和保障经验

还记得 2012 年一家天猫服务商和我说：“今年遇到了一群靠谱的人，在加上靠谱的技术，才能够一起做靠谱的事情。”这句话一直激励着我。我们也相信，能够真正地帮到商家，是对这次参与双 11 所有人的最大回报。

从上云肩挑背扛到在线迁移，让上云不再成为难事；

从资源手工离散和下线到自动化扩容和收容，让资源真正流动起来；

将诊断经验沉淀为自动化诊断工具，让诊断不再成为难事。

一幕又一幕，我们始终坚信只有把双 11 的经验和能力产品化和工具化，利用双 11 这样极具挑战的项目不断锤炼我们的产品，才是真正长远发展之计。

# 第三章 中间件

# 3.1 万亿级数据洪峰下的 分布式消息引擎

作者：冯嘉、誓嘉、尘央、牟羽

## 前言

通过简单回顾阿里中间件(Aliware)消息引擎的发展史，本文开篇于双 11 消息引擎面临的低延迟挑战，通过经典的应用场景阐述可能会面临的问题 - 响应慢，雪崩，用户体验差，继而交易下跌。为了应对这些不可控的洪峰数据，中间件团队通过大量研究和实践，推出了低延迟高可用解决方案，在分布式存储领域具有一定的普适性。在此基础上，通过对现有有限资源的规划，又推出了分级的容量保障策略，通过限流、降级，甚至熔断技术，能够有效保障重点业务的高吞吐，成功的支撑集团包括海外业务平缓舒畅地度过双 11 高峰。与此同时，在一些对高可靠、高可用要求极为苛刻的场景下，中间件团队又重点推出了基于多副本机制的高可用解决方案，能够动态识别机器宕机、机房断网等灾难场景，自动实现主备切换。整个切换过程对用户透明，运维开发人员无需干预，极大地提升消息存储的可靠性以及整个集群的高可用性。

## 1 消息引擎家族史

阿里中间件消息引擎发展到今日，前前后后经历了三代演进。第一代，推模式，数据存储采用关系型数据库。在这种模式下，消息具有很低的延迟特性，尤其在阿里淘宝这种高频交易场景中，具有非常广泛地应用。第二代，拉模式，自研的专有消息存储。能够媲美 Kafka 的吞吐性能，但考虑到淘宝的应用场景，尤其是其交易链路等高可靠场景，消息引擎并没有一味的追求吞吐，而是将稳定可靠放在首位。因为采用了长连接拉模式，在消息的实时方面丝毫不逊推模式。

在前两代经历了数年线上堪比工况的洗礼后，中间件团队于 2011 年研发了以拉模式为主，兼有推模式的高性能、低延迟消息引擎 RocketMQ。并在 2012 年进行了开源，经历了 6 年双 11 核心交易链路检验，愈久弥坚。目前已经捐赠给阿帕奇基金会（ASF），有望成为继 ActiveMQ，Kafka 之后，Apache 社区第三个重量级分布式消息引擎。时至今日，RocketMQ 很好的服务了阿里集团大大小小上千个应用，在双 11 当天，更有不可思议的万亿级消息流转，为集团大中台的稳定发挥了举足轻重的作用。



## 2 低延迟可用性探索

疾风吹征帆，倏尔向空没。千里在俄顷，三江坐超忽。 —孟浩然

### 2.1 低延迟与可用性

随着 Java 语言生态的完善，JVM 性能的提高，C 和 C++ 已经不再是低延迟场景唯一的选择。本章节重点介绍 RocketMQ 在低延迟可用性方面的一些探索。

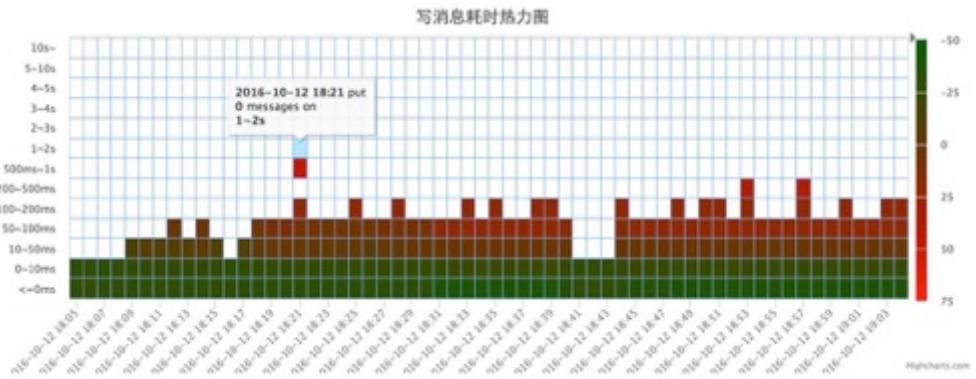
应用程序的性能度量标准一般从吞吐量和延迟两方面考量。吞吐量是指程序在一段时间内能处理的请求数量。延迟是指端到端的响应时间。低延迟在不同的环境下有不同的定义，比如在聊天应用中低延迟可以定义为 200ms 内，在交易系统中定义为 10ms 内。相对于吞吐量，延迟会受到很多因素的影响，如 CPU、网络、内存、操作系统等。

根据 Little's law，当延迟变高时，驻留在分布式系统中的请求会剧增，导致某些节点不可用，不可用的状态甚至会扩散至其它节点，造成整个系统的服务能力丧失，这种场景又俗称雪崩。所以打造低延迟的应用程序，对提升整个分布式系统可用性有很大的裨益。

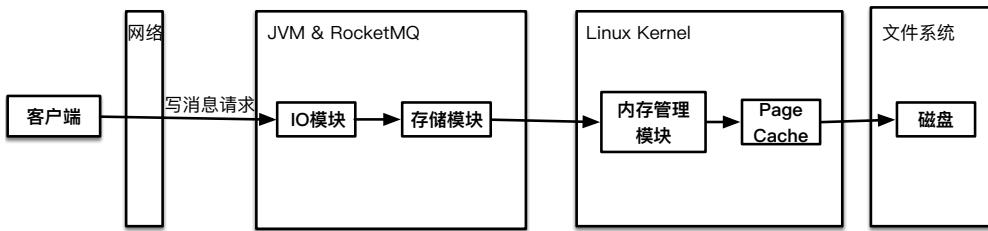
## 2.2 低延迟探索之路

RocketMQ 作为一款消息引擎，最大的作用是异步解耦和削峰填谷。一方面，分布式应用会利用 RocketMQ 来进行异步解耦，应用程序可以自如地扩容和缩容。另一方面，当洪峰数据来临时，大量的消息需要堆积到 RocketMQ 中，后端程序可以根据自己的消费速度来进行数据的读取。所以保证 RocketMQ 写消息链路的低延迟至关重要。

在今年双 11 期间，天猫发布了红包火山的新玩法。该游戏对延迟非常敏感，只能容忍 50ms 内的延迟，在压测初期 RocketMQ 写消息出现了大量 50~500ms 的延迟，导致了在红包喷发的高峰出现大量的失败，严重影响前端业务。下图为压测红包集群在压测时写消息延迟热力图统计。



作为一款纯 Java 语言开发的消息引擎，RocketMQ 自主研发的存储组件，依赖 Page Cache 进行加速和堆积，意味着它的性能会受到 JVM、GC、内核、Linux 内存管理机制、文件 IO 等因素的影响。如下图所示，一条消息从客户端发送出，到最终落盘持久化，每个环节都有产生延迟的风险。通过对线上数据的观察，RocketMQ 写消息链路存在偶发的高达数秒的延迟。



### 2.2.1 JVM 停顿

JVM ( Java 虚拟机 ) 在运行过程中会产生很多停顿，常见的有 GC、JIT、取消偏向锁 ( RevokeBias )、RedefineClasses ( AOP ) 等。对应用程序影响最大的则是 GC 停顿。RocketMQ 尽量避免 Full GC，但 Minor GC 带来的停顿是难以避免的。针对 GC 调优是一个很伽利略的问题，需要通过大量的测试来帮助应用程序调整 GC 参数，比如可以通过调整堆大小，GC 的时机，优化数据结构等手段进行调优。

对于其它 JVM 停顿，可以通过`-XX:+PrintGCAApplicationStoppedTime` 将 JVM 停顿时间输出到 GC 日志中。通过`-XX:+PrintSafepointStatistics -XX:PrintSafepointStatisticsCount=1` 输出具体的停顿原因，并进行针对性的优化。比如在 RocketMQ 中发现取 RevokeBias 产生了大量的停顿，通过`-XX:-UseBiasedLocking` 关闭了偏向锁特性。

另外，GC 日志的输出会发生文件 IO，有时候也会造成不必要的停顿，可以将 GC 日志输出到 tmpfs ( 内存文件系统 ) 中，但 tmpfs 会消耗内存，为了避免内存被浪费可以使用`-XX:+UseGLogFileRotation` 滚动 GC 日志。

除了 GC 日志会产生文件 IO，JVM 会将 jstat 命令需要的一些统计数据输出到/tmp(hsperfdata)目录下，可通过`-XX:+PerfDisableSharedMem` 关闭该特性，并使用 JMX 来代替 jstat。

### 2.2.2 锁——同步的“利”器

作为一种临界区的保护机制，锁被广泛用于多线程应用程序的开发中。但锁是一把双刃剑，过多或不正确的使用锁会导致多线程应用的性能下降。

Java 中的锁默认采用的是非公平锁，加锁时不考虑排队问题，直接尝试获取锁，若获取失败自动进行排队。非公平锁会导致线程等待时间过长，延迟变高。

倘若采取公平锁，又会对应用带来较大性能损失。

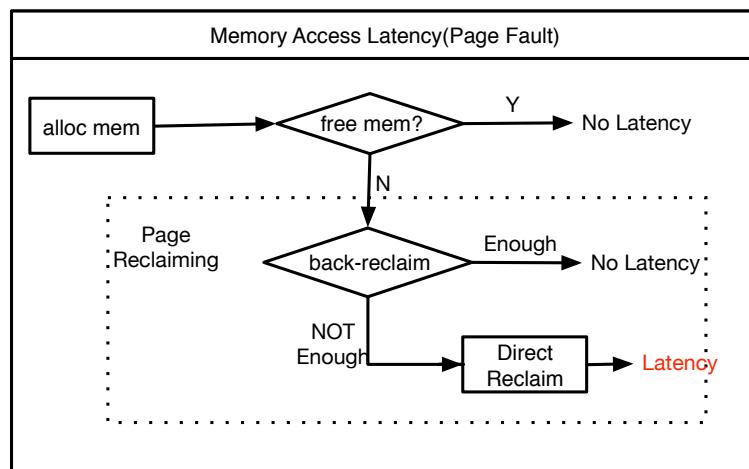
另一方面，同步会引起上下文切换，这会带来一定的开销。上下文切换一般是微秒级，但当线程数过多，竞争压力大时，会产生数十毫秒级别的开销。可通过 LockSupport.park 来模拟产生上下文切换进行测试。

为了避免锁带来的延迟，利用 CAS 原语将 RocketMQ 核心链路无锁化，在降低延迟的同时显著提高吞吐量。

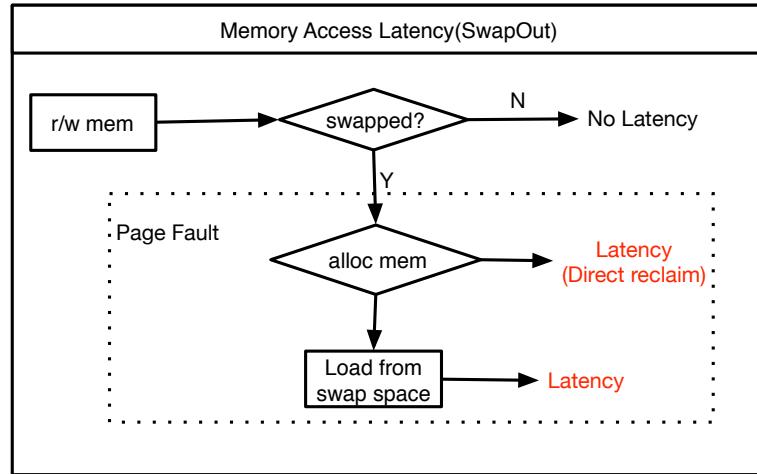
### 2.2.3 内存——没那么快

受限于 Linux 的内存管理机制，应用程序访问内存时有时候会产生高延迟。Linux 中内存主要有匿名内存和 Page Cache 两种。

Linux 会用尽可能多的内存来做缓存，大多数情形下，服务器可用内存都较少。可用内存较少时，应用程序申请或者访问新的内存页会引发内存回收，当后台内存回收的速度不及分配内存的速度时，会进入直接回收( Direct Reclaim )，应用程序会自旋等待内存回收完毕，产生巨大的延迟，如下图所示。



另一方面，内核也会回收匿名内存页，匿名内存页被换出后下一次访问会产生文件 IO，导致延迟，如下图所示。

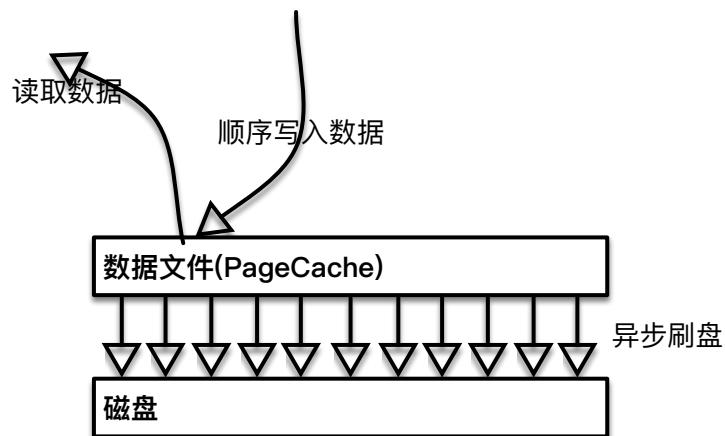


上述两种情况产生的延迟可以通过内核参数(`vm.extra_free_kbytes` 和 `vm.swappiness`)调优加以避免。

Linux 对内存的管理一般是以页为单位，一页一般为 4k 大小，当在同一页内存上产生读写竞争时，会产生延迟，对于这种情况，需要应用程序自行协调内存的访问加以避免。

## 2.2.4 Page Cache——利与弊

Page Cache 是文件的缓存，用于加速对文件的读写，它为 RocketMQ 提供了更强大的堆积能力。RocketMQ 将数据文件映射到内存中，写消息的时候首先写入 Page Cache，并通过异步刷盘的模式将消息持久化（同时也支持同步刷盘），消息可以直接从 Page Cache 中读取，这也是业界分布式存储产品通常采用的模式，如下图所示：



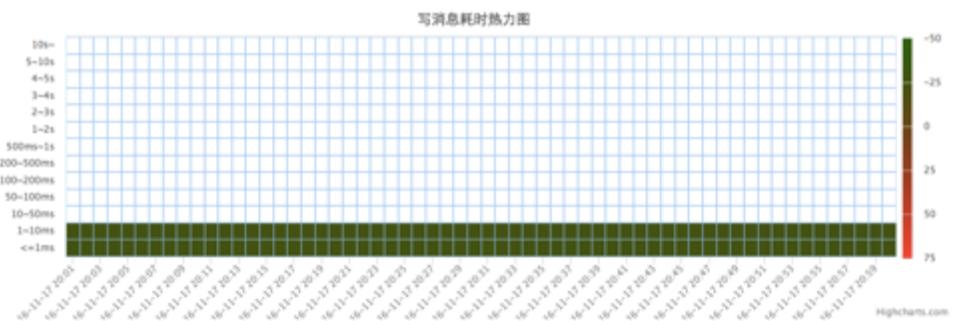
该模式大多数情况读写速度都比较迅速，但当遇到操作系统进行脏页面回写，

内存回收，内存换入换出等情形时，会产生较大的读写延迟，造成存储引擎偶发的高延迟。

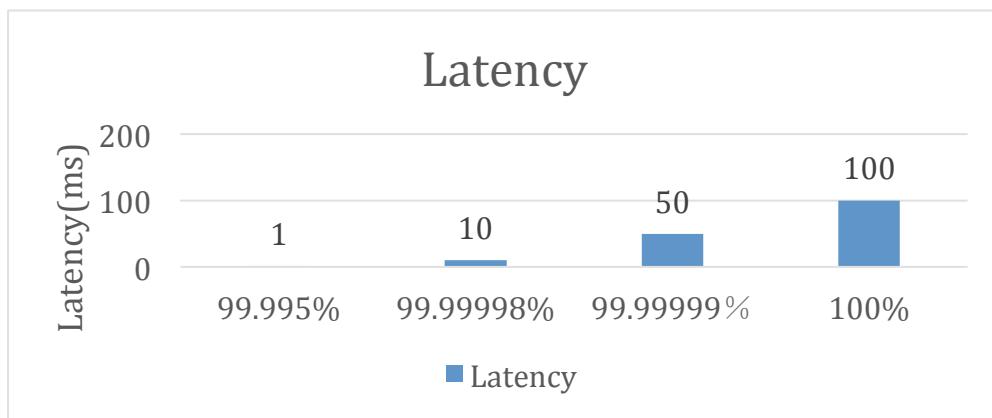
针对这种现象，RocketMQ 采用了多种优化技术，比如内存预分配，文件预热，mlock 系统调用，读写分离等，来保证利用 Page Cache 优点的同时，消除其带来的延迟。

## 2.3 优化成果

RocketMQ 通过对上述情况的优化，成功消除了写消息高延迟的情形，并通过了今年双 11 的考验。优化后写消息耗时热力图如下图所示。



优化后 RocketMQ 写消息延迟 99.995% 在 1ms 内，100% 在 100ms 内，如下图所示。



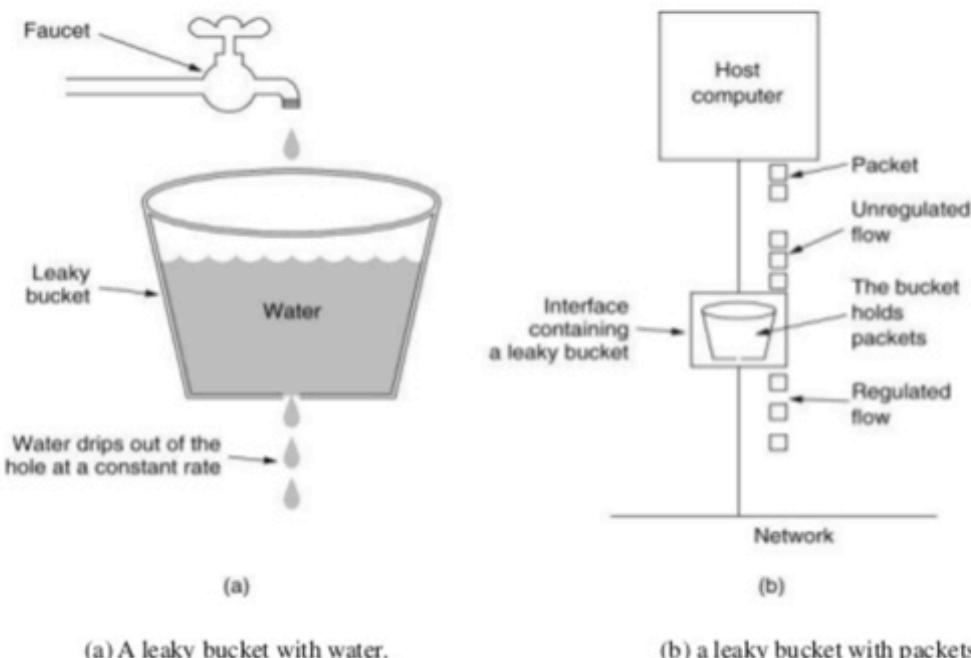
## 3 容量保障三大法宝

他强任他强，清风拂山岗。他横任他横，明月照大江。一九阳真经心法

有了低延迟的优化保障，并不意味着消息引擎就可以高枕无忧。为了给应用带来如丝般顺滑的体验，消息引擎必须进行灵活的容量规划。如何让系统能够在汹涌澎湃的流量洪峰面前谈笑风生？降级、限流、熔断三大法宝便有了用武之地。丢卒保车，以降级、暂停边缘服务、组件为代价保障核心服务的资源，以系统不被突发流量击垮为第一要务。正所谓，他强任他强，清风拂山岗。他横任他横，明月照大江！

从架构的稳定性角度看，在有限资源的情况下，所能提供的单位时间服务能力也是有限的。假如超过承受能力，可能会带来整个服务的停顿，应用的 Crash，进而可能将风险传递给服务调用方造成整个系统的服务能力丧失，进而引发雪崩。另外，根据排队理论，具有延迟的服务随着请求量的不断提升，其平均响应时间也会迅速提升，为了保证服务的 SLA，有必要控制单位时间的请求数量。这就是限流为什么愈发重要的原因。限流这个概念，在学术界又被称为 Traffic Shaping。最早起源于网络通讯领域，典型的有漏桶（leaky bucket）算法和令牌桶（token bucket）算法。

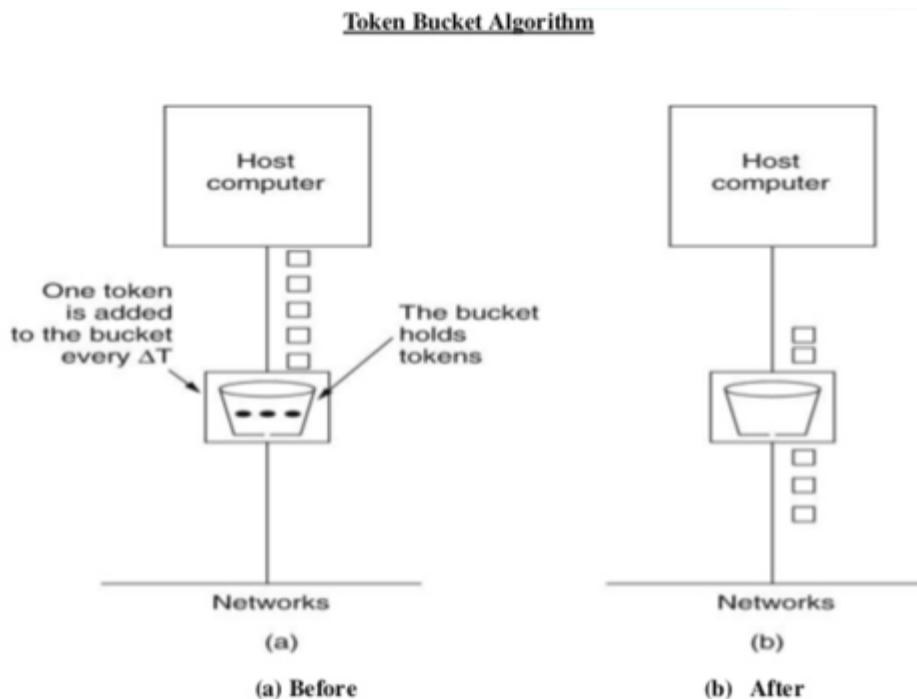
#### The Leaky Bucket Algorithm



漏桶算法基本思路是有一个桶（会漏水），水以恒定速率滴出，上方会有水

滴(请求)进入水桶。如果上方水滴进入速率超过水滴出的速率,那么水桶就会溢出,即请求过载。

令牌桶算法基本思路是同样也有一个桶,令牌以恒定速率放入桶,桶内的令牌数有上限,每个请求会 acquire 一个令牌,如果某个请求来到而桶内没有令牌了,则这个请求是过载的。很显然,令牌桶会存在请求突发激增的问题。



无论是漏桶、令牌桶,抑或其它变种算法,都可以看做是一种控制速度的限流,工程领域如 Guava 里的 RateLimiter, Netty 里的 TrafficShaping 等也都属于此。除此之外,还有一种控制并发的限流模式,如操作系统里的信号量, JDK 里的 Semaphore。

异步解耦,削峰填谷,作为消息引擎的看家本领,Try your best 本身就是其最初的设计初衷( RPC、应用网关、容器等场景下,控制速度应成为流控首选)。但即便如此,一些必要的流控还是需要考量。不过与前面介绍的不同,RocketMQ 中并没有内置 Guava、Netty 等拆箱即用的速度流控组件。而是通过借鉴排队理论,对其中的慢请求进行容错处理。这里的慢请求是指排队等待时间以及服务时间超过某个阈值的请求。对于离线应用场景,容错处理就是利用滑动窗口机制,通过缓慢缩小窗口的手段,来减缓从服务端拉的频率以及消息大小,降低对服务端的影响。而对于那些高频交易,数据复制场景,则采取了快速失败策略,既能预防应用连锁的资源耗尽而引发的应用雪崩,又能有效降低服务端压力,为端到

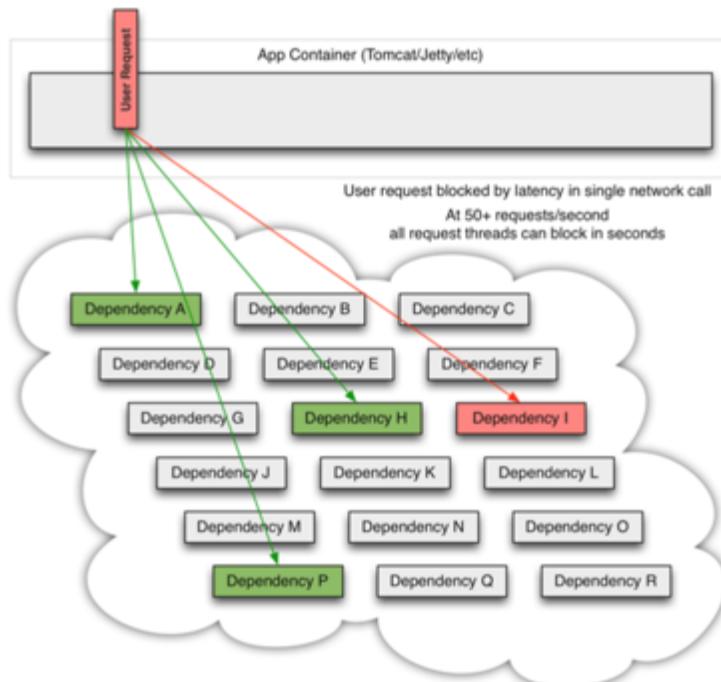
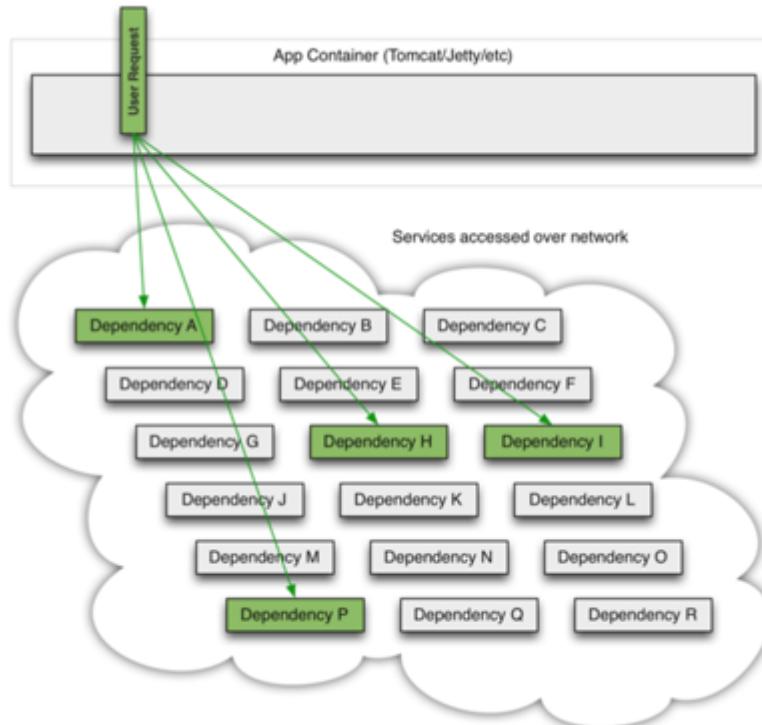
端低延迟带来可靠保障。

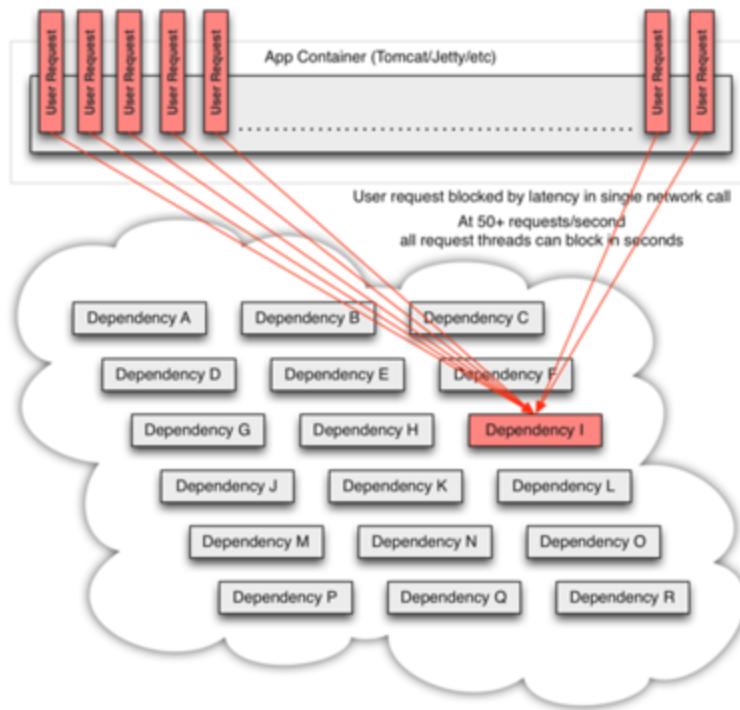
服务降级是一种典型的丢卒保车，二八原则实践。而降级的手段也无外乎关闭，下线等“简单粗暴”的操作。降级目标的选择，更多来自于服务 QoS 的定义。消息引擎早期对于降级的处理主要来自两方面，一方面来自于用户数据的收集，另一方面来自引擎组件的服务 QoS 设定。对于前者，通过运维管控系统推送应用自身 QoS 数据，一般会输出如下表格。而引擎组件的服务 QoS，如服务于消息问题追溯的链路轨迹组件，对于核心功能来说，定级相对较低，可在洪峰到来之前提前关闭。

消息通道	AppName	Topic	是否同步 (中 美同 步) or V同步	是否 单链路	是否 量大 幅 时 大	峰 值 时 刻	日常均值 峰值 TPS	日常均值 峰值 TPS	重要级别 (P1: 不能延迟，不 能丢) P2:可延 迟，不能丢 P3: 可延迟，可以 丢)	应 用 负 责 人	备注
------	---------	-------	----------------------------------	-----------	-------------------------	------------------	-------------------	-------------------	---	-----------------------	----

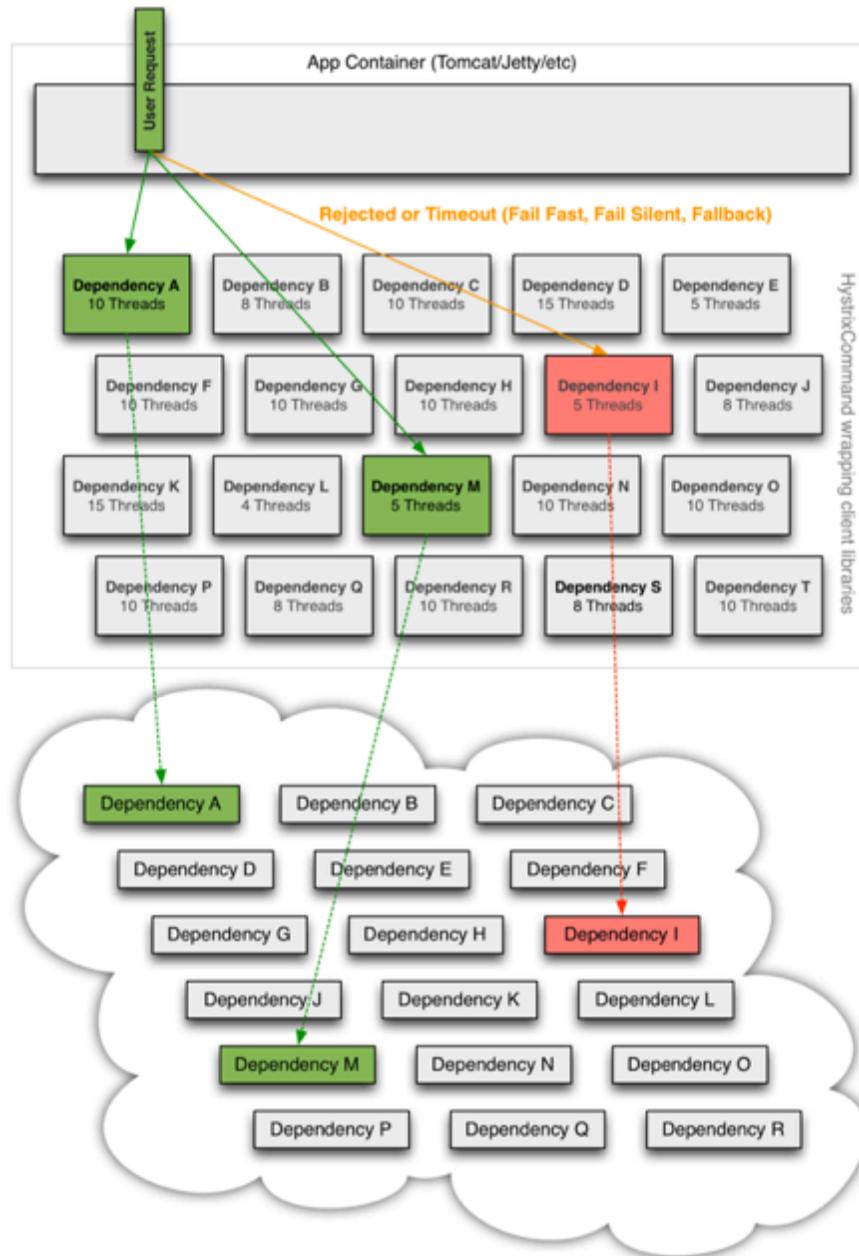
谈到熔断，不得不提经典的电力系统中的保险丝，当负载过大，或者电路发生故障或异常时，电流会不断升高，为防止升高的电流有可能损坏电路中的某些重要器件或贵重器件，烧毁电路甚至造成火灾。保险丝会在电流异常升高到一定的高度和热度的时候，自身熔断切断电流，从而起到保护电路安全运行的作用。

同样，在分布式系统中，如果调用的远程服务或者资源由于某种原因无法使用时，没有这种过载保护，就会导致请求的资源阻塞在服务器上等待从而耗尽系统或者服务器资源。很多时候刚开始可能只是系统出现了局部的、小规模的故障，然而由于种种原因，故障影响的范围越来越大，最终导致了全局性的后果。而这种过载保护就是大家俗称的熔断器(Circuit Breaker)。Netflix 公司为了解决该问题，开源了它们的熔断解决方案 Hystrix。





上述三幅图，描述了系统从初始的健康状态到高并发场景下阻塞在下游的某个关键依赖组件的场景。这种情况很容易诱发雪崩效应。而通过引入 Hystrix 的熔断机制，让应用快速失败，继而能够避免最坏情况的发生。



借鉴 Hystrix 思路，中间件团队自研了一套消息引擎熔断机制。在大促压测备战期间，曾经出现过由于机器硬件设备导致服务不可用。如果采用常规的容错手段，是需要等待 30 秒时间，不可用机器才能从列表里被摘除。但通过这套熔断机制，能在毫秒范围内识别并隔离异常服务。进一步提升了引擎的可用性。

## 4 高可用解决方案

昔之善战者，先为不可胜，以待敌之可胜。不可胜在己，可胜在敌。故善战者，能为不可胜，不能使敌之必可胜。故曰：胜可知，而不可为。  
—孙武

虽然有了容量保障的三大法宝作为依托，但随着消息引擎集群规模的不断上升，到达一定程度后，集群中机器故障的可能性随之提高，严重降低消息的可靠性以及系统的可用性。与此同时，基于多机房部署的集群模式也会引发机房断网，进一步降低消息系统的可用性。为此，阿里中间件（Aliware）重点推出了基于多副本的高可用解决方案，动态识别机器故障、机房断网等灾难场景，实现故障自动恢复；整个恢复过程对用户透明，无需运维人员干预，极大地提升了消息存储的可靠性，保障了整个集群的高可用性。

高可用性几乎是每个分布式系统在设计时必须要考虑的一个重要特性，在遵循 CAP 原则（即：一致性、可用性和分区容错性三者无法在分布式系统中被同时满足，并且最多只能满足其中两个）基础上，业界也提出了一些针对分布式系统通用的高可用解决方案，如下图所示：

**Techniques and Tradeoffs in High-Available Architecture**

	Backups	Master/Slave	Master/Master	2PC	Paxos
<b>Consistency</b>	Weak	Eventual		Strong	
<b>Transactions</b>	No	Full	Local	Full	
<b>Latency</b>	Low			High	
<b>Throughput</b>	High			Low	Medium
<b>Data Loss</b>	Lots	Some		None	
<b>Failover</b>	Down	Read-Only	Read/Write		

其中，行代表了分布式系统中通用的高可用解决方案，包括冷备、Master/Slave、Master/Master、两阶段提交以及基于 Paxos 算法的解决方案；列代表了分布式系统所关心的各项指标，包括数据一致性、事务支持程度、数据延迟、系统吞吐量、数据丢失可能性、故障自动恢复方式。

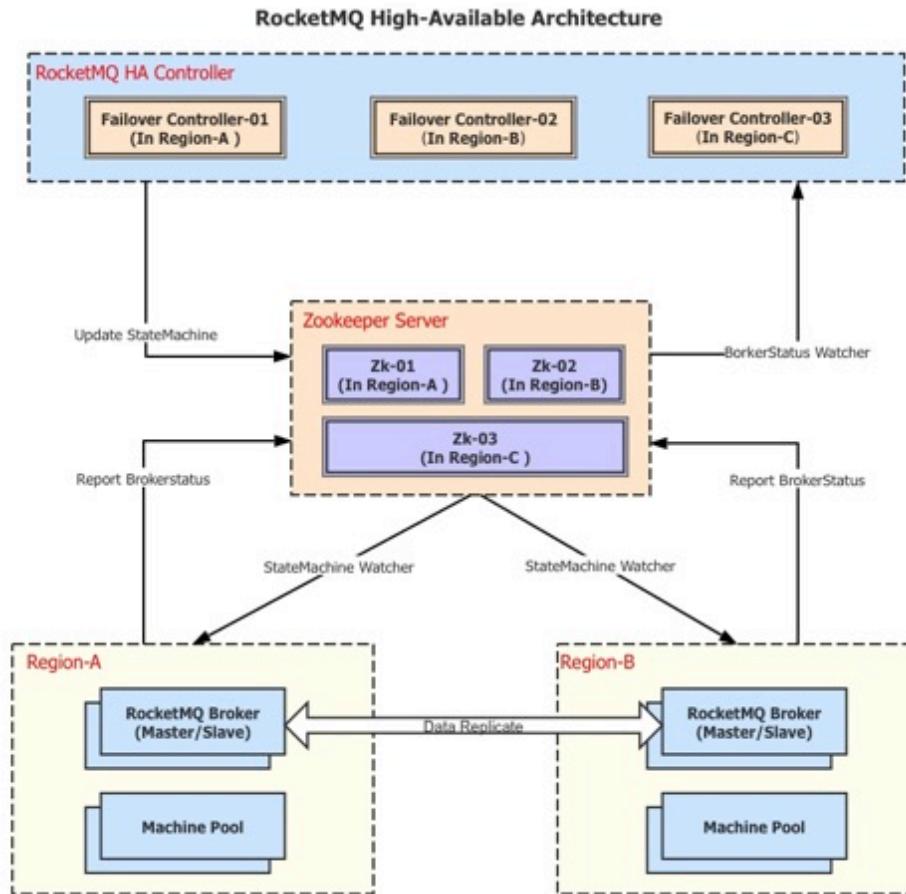
从图中可以看出，不同的解决方案对各项指标的支持程度各有侧重。基于 CAP 原则，很难设计出一种高可用方案能同时够满足所有指标的最优值，以 Master/Slave 为例，一般满足如下几个特性：

- 1) Slave 是 Master 的备份，可以根据数据的重要程度设置 Slave 的个数。  
数据写请求命中 Master，读请求可命中 Master 或者 Slave。
- 2) 写请求命中 Master 之后，数据可通过同步或者异步的方式从 Master 复制到 Slave 上；其中同步复制模式需要保证 Master 和 Slave 均写成功后才反馈给客户端成功；异步复制模式只需要保证 Master 写成功即可反馈给客户端成功。

数据通过同步或者异步方式从 Master 复制到 Slave 上，因此 Master/Slave 结构至少能保证数据的最终一致性；异步复制模式下，数据在 Master 写成功后即可反馈给客户端成功，因此系统拥有较低的延迟和较高的吞吐量，但同时会带来 Master 故障丢数据的可能性；如期望异步复制模式下 Master 故障时数据仍不丢，Slave 只能以 Read-Only 的方式等待 Master 的恢复，即延长了系统的故障恢复时间。相反，Master/Slave 结构中的同步复制模式会以增大数据写入延迟、降低系统吞吐量的代价来保证机器故障时数据不丢，同时降低系统故障恢复时间。

## 5 RocketMQ 高可用架构

RocketMQ 基于原有多机房部署的集群模式，利用分布式锁和通知机制，借助 Controller 组件，设计并实现了 Master/Slave 结构的高可用架构，如下图所示：



其中，Zookeeper 作为分布式调度框架，需要至少在 A、B、C 三个机房部署以保证其高可用，并为 RocketMQ 高可用架构提供如下功能：

- 1) 维护持久节点 ( PERSISTENT )，保存主备状态机；
- 2) 维护临时节点 ( EPHEMERAL )，保存 RocketMQ 的当前状态；
- 3) 当主备状态机、服务端当前状态发生变更时，通知对应的观察者。

RocketMQ 以 Master/Slave 结构实现多机房对等部署，消息的写请求会命中 Master，然后通过同步或者异步方式复制到 Slave 上进行持久化存储；消息的读请求会优先命中 Master，当消息堆积导致磁盘压力大时，读请求转移至 Slave。

RocketMQ 直接与 Zookeeper 进行交互，体现在：

- 1) 以临时节点的方式向 Zookeeper 汇报当前状态；
- 2) 作为观察者监听 Zookeeper 上主备状态机的变更。当发现主备状态机变化时，根据最新的状态机更改当前状态；

RocketMQ HA Controller 是消息引擎高可用架构中降低系统故障恢复时间的无状态组件，在 A、B、C 三个机房分布式部署，其主要职责体现在：

- 1) 作为观察者监听 Zookeeper 上 RocketMQ 当前状态的变更；
- 2) 根据集群的当前状态，控制主备状态机的切换并向 Zookeeper 汇报最新主备状态机。

出于对系统复杂性以及消息引擎本身对 CAP 原则适配的考虑，RocketMQ 高可用架构的设计采用了 Master/Slave 结构，在提供低延迟、高吞吐量消息服务的基础上，采用主备同步复制的方式避免故障时消息的丢失。数据同步过程中，通过维护一个递增的全局唯一 SequenceID 来保证数据强一致。同时引入故障自动恢复机制以降低故障恢复时间，提升系统的可用性。

## 5.1 可用性评估

系统可用性（Availability）是信息工业界用来衡量一个信息系统提供持续服务能力，它表示的是在给定时间区间内系统或者系统某一能力在特定环境中能够正常工作的概率。简单地说，可用性是平均故障间隔时间（MTBF）除以平均故障间隔时间（MTBF）和平均故障修复时间（MTTR）之和所得的结果，即：

$$Availability = \frac{MTBF}{(MTBF + MTTR)}$$

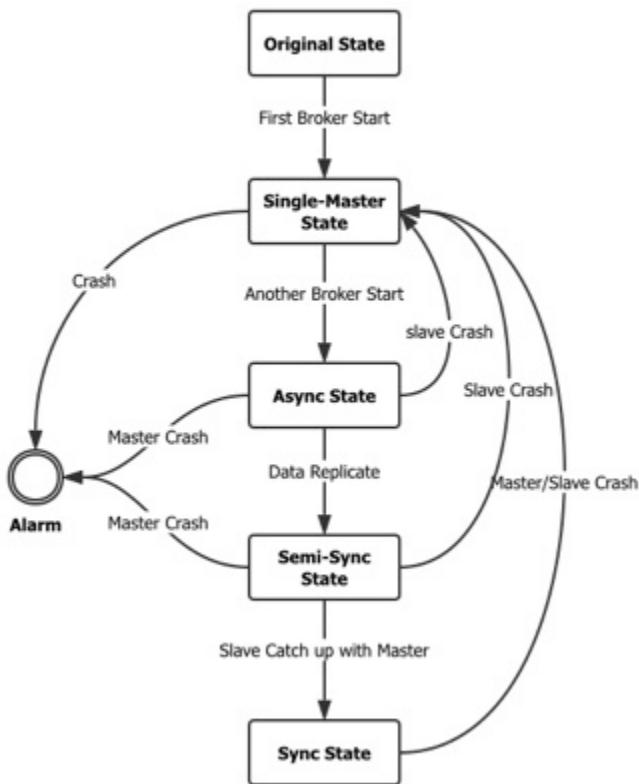
通常业界习惯用 N 个 9 来表征系统可用性，比如 99.9% 代表 3 个 9 的可用性，意味着全年不可用时间在 8.76 小时以内，99.999% 代表 5 个 9 的可用性，意味着全年不可用时间必须保证在 5.26 分钟以内，缺少故障自动恢复机制的系统将很难达到 5 个 9 的高可用性。

## 5.2 RocketMQ 高可用保障

通过可用性计算公式可以看出，要提升系统的可用性，需要在保障系统健壮性以延长平均无故障时间的基础上，进一步加强系统的故障自动恢复能力以缩短平均故障修复时间。RocketMQ 高可用架构设计并实现了 Controller 组件，按照单主状态、异步复制状态、半同步状态以及最终的同步复制状态的有限状态机进行转换。在最终的同步复制状态下，Master 和 Slave 任一节点故障时，其它节点能够在秒级时间内切换到单主状态继续提供服务。相比于之前人工介入重启来恢复服务，RocketMQ 高可用架构赋予了系统故障自动恢复的能力，能极大缩短平均故障恢复时间，提升系统的可用性。

下图描述了 RocketMQ 高可用架构中有限状态机的转换：

RocketMQ High-Available StateMachine Transition



- 1 ) 第一个节点启动后 ,Controller 控制状态机切换为单主状态 , 通知启动节点以 Master 角色提供服务。
- 2 ) 第二个节点启动后 ,Controller 控制状态机切换成异步复制状态。 Master 通过异步方式向 Slave 复制数据。
- 3 ) 当 Slave 的数据即将赶上 Master , Controller 控制状态机切换成半同步状态 , 此时命中 Master 的写请求会被 Hold 住 , 直到 Master 以异步方式向 Slave 复制了所有差异的数据。
- 4 ) 当半同步状态下 Slave 的数据完全赶上 Master 时 , Controller 控制状态机切换成同步复制模式 , Mater 开始以同步方式向 Slave 复制数据。该状态下任一节点出现故障 , 其它节点能够在秒级内切换到单主状态继续提供服务。

Controller 组件控制 RocketMQ 按照单主状态，异步复制状态，半同步状态，同步复制状态的顺序进行状态机切换。中间状态的停留时间与主备之间的数据差异以及网络带宽有关，但最终都会稳定在同步复制状态下。

## 展望

虽然经历了这么多年线上堪比工况的苛刻检验，阿里中间件消息引擎仍然存在着优化空间，如团队正尝试通过优化存储算法、跨语言调用等策略进一步降低消息低延迟存储。面对移动物联网、大数据、VR 等新兴场景，面对席卷全球的开放与商业化生态，团队开始着手打造第 4 代消息引擎，多级协议 QoS，跨网络、跨终端、跨语言支持，面向在线应用更低的响应时间，面向离线应用更高的吞吐，秉持取之于开源，回馈于开源的思想，相信 RocektMQ 朝着更健康的生态发展。

## 参考文献

- [1]Ryan Barrett.  
[http://snarfed.org/transactions\\_across\\_datacenters\\_io.html](http://snarfed.org/transactions_across_datacenters_io.html)
- [2]<http://www.slideshare.net/vimal25792/leaky-bucket-tocken-buckettraffic-shaping>
- [3]<http://systemdesigns.blogspot.com/2015/12/rate-limiter.html>
- [4]Little J D C, Graves S C. Little's law[M]//Building intuition. Springer US, 2008: 81-100.
- [5][https://access.redhat.com/documentation/en-US/Red\\_Hat\\_Enterprise\\_Linux/6/html-single/Performance\\_Tuning\\_Guide/index.html](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/index.html)
- [6]<http://highscalability.com/blog/2012/3/12/google-taming-the-long-latency-tail-when-more-machines-equal.html>
- [7][https://www\\_azul\\_com/files/EnablingJavaInLatencySensitiveEnv\\_DotCMSBootcamp\\_Nashville\\_23Oct20141.pdf](https://www_azul_com/files/EnablingJavaInLatencySensitiveEnv_DotCMSBootcamp_Nashville_23Oct20141.pdf)

## 第四章 电商云化

# 4.1 17.5W 秒级交易峰值下的混合云弹性架构之路

作者：唐三 乐竹 锐晟 潇谦

## 前言

每年的双 11 都是一个全球狂欢的节日，随着每年交易逐年创造奇迹的背后，按照传统的方式，我们的成本也在逐年上升。双 11 当天的秒级交易峰值平时的近 10 多倍，我们要用 3-4 倍的机器去支撑。但大促过后这批机器的资源利用率不高，到次年的双 11 会形成较长时间的低效运行。试想一下，电商交易有大促峰值，而阿里云有售卖 Buffer，如果能充分发挥云计算的弹性能力，让资源可以两边快速腾挪，就可以解决资源浪费的问题了。把我们的交易单元可以部署在云上面，那么大促的时候我们只需要按照压测模型去云上构建一个符合能力的新单元即可，用完马上释放掉，这样无疑是优雅的。专有云+公共云 的混合云弹性架构成为一种自然而然的选择，不但可以资源合理利用，降低成本，同时也锻炼了阿里人的技术能力，为用户提供更优质的服务。

有了架构思路，实现起来似乎也没那么容易。阿里的交易涉及几百个系统，他们之间的依赖错综复杂，如果能够把他们快速的搭建在云上？这次系统之间的依赖如何复杂，如果把他们的容量估算好，快速调整他们的容量水位？这就不得不提到下面的两个秘密武器了：一键建站和弹性容量交付

# 1 一键建站

## 1.1 背景

一键建站是在底层基础设施交付的基础上，快速地在一个空机房部署交易单元，使新机房迅速具备对外提供服务的能力。一键建站的逆过程叫一键下站，即迅速切除单元流量，释放所有单元内应用的物理资源。

从架构的层面看，一键建站的基础是阿里电商体系的异地多活。从运维的角度看，一键建站是运维产品的升华，更是运维效率的核心体现。

一键建站第一次被提出是在 2014 年，但由于系统多，依赖复杂，加上中间件的复杂性，当时新建一个单元需要耗时近 1 个月的时间，更是需要所有单元链路上的运维同学参与。去年，DB 实现了一次完整意义上的一键建站，中间件的建站实现了半自动化，但是应用的建站过程仍需要很多运维同学的支持。今年，一键建站进行了重构，并提出一天（8 小时）一单元的目标，在几乎不用运维同学参与的情况下，顺利支持了 3 个云单元的建站工作，最快一次耗时 6 小时建站。

## 1.2 挑战

今年的双 11 单元架构是三地五单元，一中心四单元，也是第一次遇到同机房两单元。如何控制单元内的链路封闭，单元与单元、单元与中心的同步与可见性，是异地多活的大挑战，也是一键建站的难点。

首先需要明确单元内部署什么。建站需要维护一份知识库，包括单元的数据仓库，中间件，统一接入，以及导购、商品、店铺、交易、会员等一百多个应用。需要知道每个产品的服务器配置，软件配件，容量需求，甚至是应用间链路依赖等相关信息。这份知识库会跟随日常运维操作，调用链路日志等不断更新。同时，一个完整单元不仅仅包含线上环境，还需包含预发环境与小流量(测试环境)，每套环境都有自己的一份知识库。

其次是需要明确部署的每个步骤实现。单元内的每个产品，都需要明确部署的操作细节，以及产品之间的前后依赖。今年，一键建站第一次在云上实施，面对全新的服务器资源(ecs)，全新的网络资源(slb)以及全新的部署方式(docker)

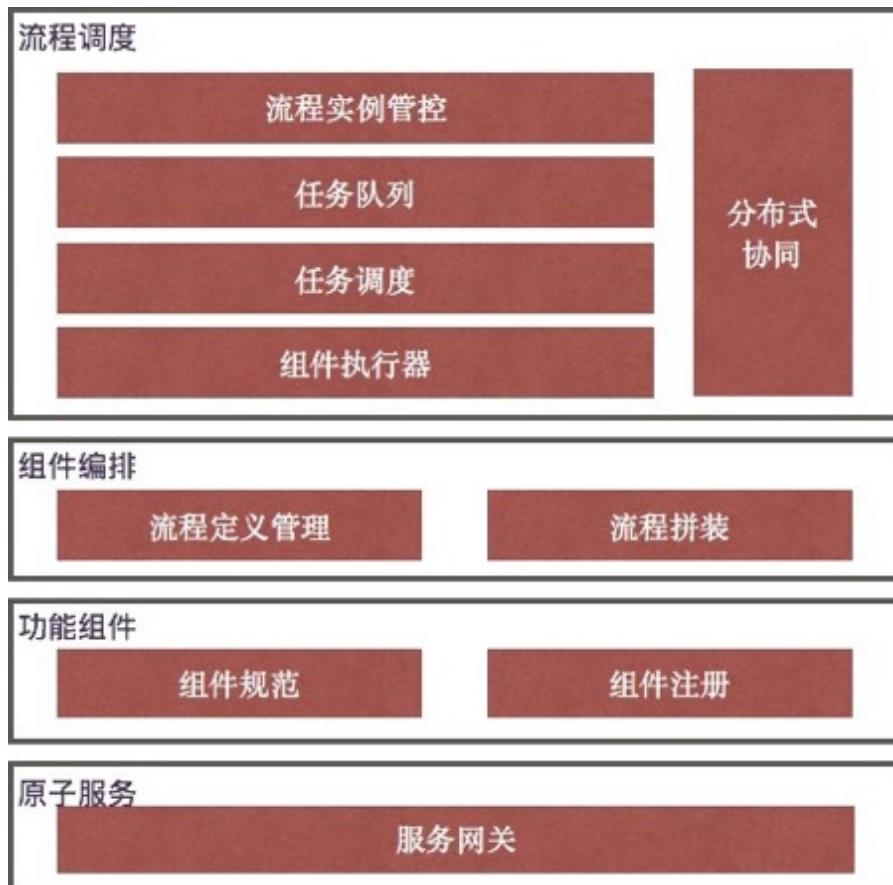
等，每个环节都需要技术落地。由此可见，一键建站是一个庞大的系统，几乎涉及所有的运维产品。

在明确了建站数据与建站步骤的基础上，还需要有一套技术实现能将单元内所有产品相关的近四千个部署步骤串联起来，这就是建站系统。追求建站效率的同时，安全始终要铭记于心。建站的每个步骤，都需要考虑可能出现的突发情况以及应对策略。

### 1.3 技术架构

一键建站是一个体系的构建，是要在原有运维产品的基础上进行升华，将相关产品的原子性服务联动起来，最终凝聚成一个按钮。

一键建站涉及的单元产品种类繁多，相关操作保罗万象，而且变化极为频繁。如果为每类产品写死操作流程，那建站只会疲于代码，即使完成了代码，也只会是一次性的玩物。因此建站需要更多的考虑灵活性，在最终的技术实现上，将系统架构分为四个层次，原子服务、功能组件、组件编排以及流程调度。系统架构如下图：



## 1.4 原子服务

建站平台的能力来源于周边的运维产品，接入相关系统的服务，将最小粒度的一次服务调用称为一个原子操作。服务网关包装一系列原子操作，以提供上层业务调用。

作为唯一的外部系统调用入口，服务网关还需要做统一的日志记录，业务链路跟踪以及故障告警等。

对建站平台的效率要求，很大一部分最终会落在外部系统服务上。最具代表性的是服务器资源申请与 docker 镜像，这两条链路的背后，凝聚着很多人努力的心血。

## 1.5 功能组件

功能组件，是将相关的原子服务进行整合，从而形成的一个个有业务含义的独立功能模块。比如创建服务器、添加帐号、创建 vip、docker upgrade、应用启动、更新 hsf 路由等等，将近百个原子服务最终聚合成三四十个功能组件。组件的实现需要能保证幂等。

功能组件需要遵循一定的规范，从而使得同一组件能被不同的应用，不同的流程复用。

## 1.6 组件编排

组件编排是建站灵活性的核心。建站平台支持在 web 页面动态编排功能组件，从而组装成一个个可以运行的流程。单元内的每类中间件或应用都可能存在部署上的差异，通过服务编排，使每一类产品都能对应到一类流程。

建站需要涉及整套中间件以及一百多个单元应用，这些产品在部署启动上还存在先后链路依赖。去除弱依赖，将单元产品依赖描述成一张无环有向图，每个节点代表一个产品的部署流程。将整张图描述成一个流程，这个流程就是建站流程！

## 1.7 流程调度

流程调度是建站稳定性的有力保障。流程调度负责建站流程的分布式执行，是流程引擎的一个实现，至少需要达到下述几点要求：

1. 高可用。服务器宕机不能影响流程执行；
2. 系统容错。下游系统异常诊断，自动重试；
3. 并发访问控制。流程节点不应该同时在多台服务器被调度；

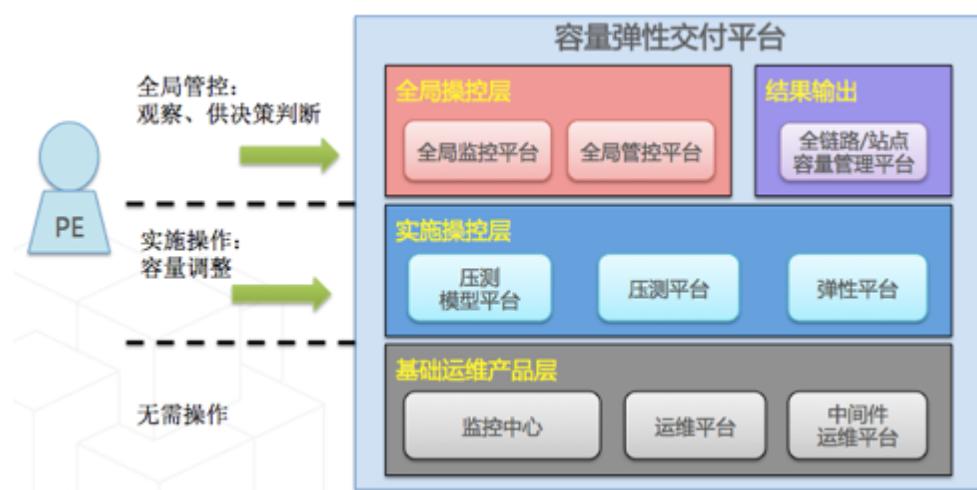
在结构上，流程调度可以分为流程实例管控、任务队列、任务调度、组件执行器与分布式协同组件等。每个节点按照自身的负载情况加载流程实例到本地任务队列，组件执行器负责每个组件的入参注入，出参收集以及反射调用，分布式协同保障同一时刻仅有一个节点在调度流程实例。

一键建站流程是一个包含众多子流程的嵌套流程结构，建站时，流程调度需要同时执行上千个流程。

一键建站只是完成了最小单元的建站工作，如果想让这个单元支撑好大促的流量，还需要对这个单元进行容量评估和扩容，如何快速的评估各个应用的容量并自动扩容呢？这时就需要弹性容量交付出场了。

## 2 弹性容量交付

如下图：



今年弹性技术在实时容量评估算法上作了一定的改良，期初主要出于提升效率，最大程度地降低实施成本，与保障集群稳定性的目的：更加智能，使用在线机器学习实时测算应用性能变化，并可作出简单的故障原因分析，通过算法对各个单元的应用集群进行自然水位拉平。

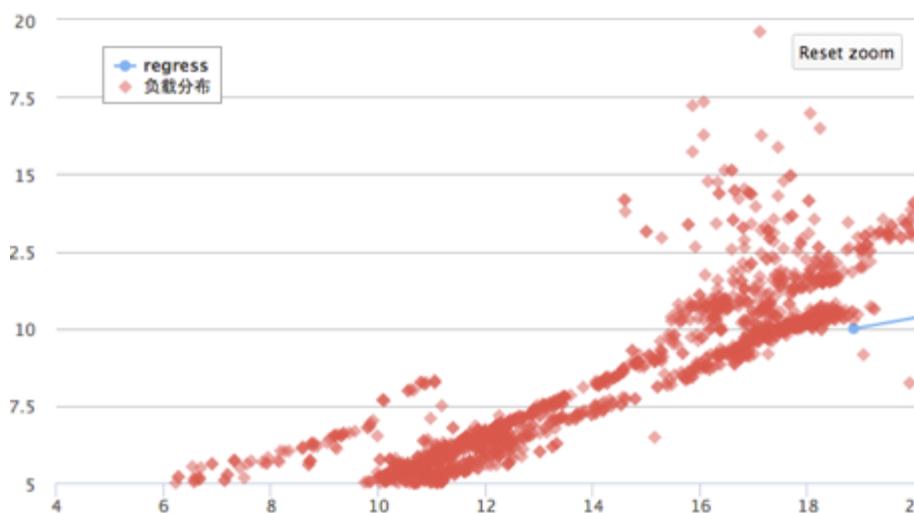
1. 如何用机器在无人介入的情况下，预测应用集群各个单元的性能；需要做很多事情：日常性能变化测量；应用发布性能变化测量；集群中单机的性能变化预测，与目标交易量下会有多少比例机器挂掉的预测，容量问题还是性能问题的判断等等等。

2. 为支持 XX 笔交易的业务目标，需要多少资源；或者说，现有 XX 些资源，最高可以支撑多高的交易量？

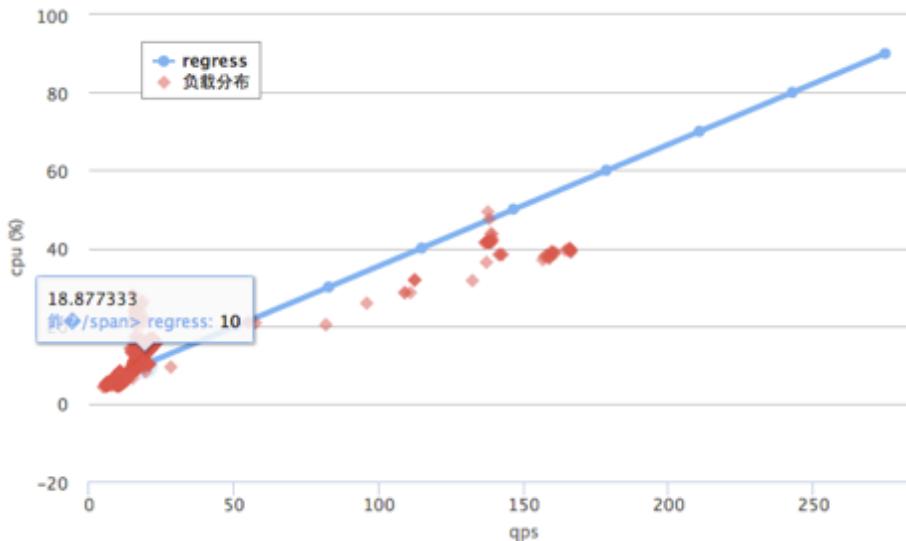
3. 一个应用集群在什么样的物理资源利用率下稳定性与成本会是一个最佳配比？

#### 4. 资源预算。

我们先简单以一个在线 web 服务类应用进行分析，在线电商每天的流量波动与资源利用率是存在一定的关系的（当然也可以换成其它指标进行测算），我们将两项指标叠加，呈散点图形态



现在假设，我们设定资源利用率阀值为 70% 的 cpu 利用率，预测该应用集群的服务能力，我们利用上面呈现的散点图做一次拟合，延长趋势线，呈以下形态：



则求出，该应用极限能力在 X% 的资源利用率下的服务能力大致是 Y.

但实际场景中，情况要复杂得多，在不同压力下，随着物理机的利用率整体饱和度的上升，性能会有一定的损耗，将不同压力下测算的服务能力记录，并作一次回归，预测出目标压力下，大致损耗度，并用刚才计算好的服务能力减去目标压力下的损耗度即可，

哪下一个问题来了，应用集群的资源利用率多少为极限值？这里只是一个假定，每个应用集群的极限能力都不相同；首先前文已经提到，由于各个应用集群布署的物理机坑位不同，有可能超卖，也有可能会与资源占用多的应用布署在同一个物理核内，超线程会带来一定的影响，而一个物理核通常分为两个逻辑核，是否一个物理核的总能力 / 2 则为两项占用该物理核逻辑核上的能力。假定 100% 的资源利用率为满负荷，则两个逻辑核各分 50% 的能力相对合理？

但实际情况是，占用两个逻辑核的应用集群利用率，在容量层次不齐的宏观情况下，有的偏高，有的偏低，这就会出现资源抢占问题。

如何识别某项应用集群合理的资源利用率是多少？我们需要做一些事情，即除了对整个应用集群作上文中讲到的资源测算，还需要对每台单机作能力测算，这里我们随便拟定一个值，如单机负载如果超过 80% 是不可承受的，则我们在整体全链路压测时，会对每台单机做实时的负载预测，看在目标交易量下，多少比例的机器会超过最大的承受能力，该集群的总 qps 会有出现多少比例的损耗。这里假定我们认为不允许有机器出现这样的情况，则当某台机器预测值达到最大

承受能力时，则认为当前集群能力的合理负载应该在多少。

根据上文的描述，我们可以直接拿到测算好的各个应用集群的容量配比进行在线备容即可。通过后续每次的压测，对各个应用集群的预期资源利用率进行逐步逼近，最终达到整体备容目标。

正因为有了以上两个秘密武器，我们在双 11 之前就快速的做好了容量准备，同时双 11 一过，我们立刻对云资源进行一键下站，把资源归还到云的 Buffer 里，对公共云进行售卖。

## 4.2 集团 AliDocker 化双 11 总结

作者：林轩、白慕、潇谦

### 前言

在基础设施方面，今年双 11 最大的变化是支撑双 11 的所有交易核心应用都跑在了 Docker 容器中。几十万 Docker 容器撑起了双 11 交易 17.5 万笔每秒的下单峰值。众所周知 Docker 技术这几年大热，但如果期望阿里这么大体量的应用全部使用 Docker，这可不是一朝一夕就能完成的事情。阿里的应用数量庞大，种类众多，光兼容性的验证没个 1、2 年的时间没人敢把核心应用放上去。因此虽然 Docker 能给研发和运维带来的好处，作为技术人员大家都心领神会，但是想直接去使用，那面对 Docker 浪潮却只能是坐观弄潮者，徒有羡鱼情。

### 1 T4 和 Docker 的融合

所幸的是，这个状况在去年 7 月份有了改变，这事要从阿里内部的容器技术产品 T4 说起。T4 是阿里在 2011 年的时候基于 Linux Container(LXC)开发的容器技术基础设施。从 12 年到 15 年 3 年时间里用的应用越来越多，实际上已经覆盖了电商领域大部分 App。相比 Docker 的模式和理念，T4 其实更适合阿里内部的运维现状。T4 是从阿里内部的资源管理和日常运维中土生土长出来的产品，在诞生的第一天就针对内部基础设施、运维工具甚至是运维习惯做了很多特别的设计。T4 在 LXC 容器的基础上，对容器资源和各种统计的可见性做了很多卓有成效的隔离，使得在容器内部看到的资源就是分配给这个容器的资源，内部看到的负载就是这个容器的负载等等。同时在 LXC 之外还做了容器的磁盘空间配额限制和隔离，容器内看到的那块磁盘大小就是创建容器时分配给他的磁盘配额大小。这样在 CPU、网络、内存、磁盘等资源使用和统计监控上做到了

容器内和物理机上基本没有区别。原来跑在物理机，或者 KVM、Xen 中的应用，能够平滑无感知的迁移到 T4 中。当年从 Xen、Kvm 迁移到 T4 的过程中，很多应用的开发者和 Owner 确实不知道什么时候完成迁移的，可能在某次常规的应用发布中，后台工具系统已经自动做完迁移了。甚至直到去年在和一个应用研发的沟通中，他坚信自己的应用是跑在 KVM 中的，我们到后台查了一下，其实已经在 T4 上了。

我们在去年 5 月份接手了 T4 的整个维护工作，发现 T4 的功能和 Docker 非常相似，但是 T4 和 Docker 相比有一块很大的短板就是没有镜像机制，再加上 T4 这种对人工运维的友好性，在长期使用中就出现了一个问题，就是应用的容器里面遗留了越来越多的不一致，对运维的标准化造成了阻碍。比如有相当一部分应用，在迁移到另外一台物理机上后就没法运行了。因为从第一次上线到存活期间做的各种环境变更、软件升级等，很难严格的记录到发布系统里。另外一个发现是 T4 的技术栈和 Docker 非常相似，都是基于 linux 内核的 cgroup、namespace、chroot、bridge 等机制运作起来的。T4 使用了 LXC，Docker 的执行引擎也支持 LXC，但神奇的是 T4 却能够在阿里内部老版本的 OS 和内核上跑起来。因此直觉告诉我将 T4 和 Docker 结合起来应该是可行的。于是从去年 6 月份开始对这个方向做了探索，终于找到一个恰当的模式，对 Docker 和 T4 都做了一些修改整合后，将两者融合为了一个产品，相当于既让 T4 具备了 Docker 的镜像能力，又让 Docker 具备了 T4 对内部运维体系的友好性，并且能够运行在内部早期的 AliOS5u 和 2.6 内核上。这个产品在内部称为 AliDocker，在去年 8 月份推出了第一个雏形版本。另外这个版本还解决了 Docker 当时很严重的一个问题，就是 Daemon 退出其上所有的容器都会退出，这一点在真正生产环境大规模部署时是无法接受的。Docker 官方直到 1.10 版本才开始部分解决这个问题。我们当时的 Docker 版本从 1.5 一直跟进到后来大规模部署的 1.9 版本，通过 Docker 的 Daemon 管控进程和容器的解耦，Daemon 重启后对之前运行容器的自动识别和重新接管，解决了这个问题。这样 Docker 在阿里内部大规模应用就有了可能。

从这个版本发布到能够替换 T4 大规模部署还走了很长的路，首先 T4 和 Docker 毕竟是两个不同的产品，除了大家耳熟能详的容器机制之外，其实还有很多的细节和特性。为了让应用在迁移中无感知，AliDocker 对原先 T4 容器的细节功能做了全面的兼容，同时对上层的运维系统做了大量改造，使其支持

Docker 场景下的发布和运维模式，从 Docker 镜像构建到分发启停、扩容迁移都做了完备的工具和流程支持。其次在 T4 到 AliDocker 切换的过程中，我们做了 2 者混跑的支持，也就是说同一台物理机上可以同时跑原来的 T4 容器和新的 AliDocker 容器，互不干扰并且能统一运维。因为众多应用的实例是交错部署在众多物理机上的，同一个物理机上往往有十几个不同应用的实例混跑。这种兼容机制就保证了不同应用可以按各自的节奏逐步完成 Docker 化，而不需要在某个时间和空间做一刀切，避免了大规模升级 Docker 的过程中不必要的应用腾挪和迁移。然后在我们将 AliDocker 和 T4 功能完全对齐，从实例级别到应用级别做了足够的灰度后，推送了一个开关，使得从那一刻开始创建新 T4 实例时会自动创建为 AliDocker 实例，从而完成了增量实例的切换。对于存量的 T4 实例，我们选择了一个完整的深圳交易单元，分批次做了批量切换，在切换期间如果发生大的问题，可以把深圳单元的流量全部切换到上海。这一保障要得益于阿里的异地多活灾备架构，对于这类底层基础设施的升级能够提供理想的兜底方案，使我们敢于放开手脚去做，而又能有效的控制风险。所幸在整个升级洗牌的过程中，没有动用到这个大杀器的功能，虽然出了一些小问题但都能及时修复，影响不大，也让我们的系统更加健壮，让各个部门的人对交易核心流量切换到 AliDocker 这件事情更有信心。

## 2 核心应用镜像化

但是仅仅将运行容器从 T4 切换到 Docker 其实对我们带来的改变并不大。Docker 真正的核心价值在于镜像机制，以及镜像机制带来的研发与运维模式的变革。应用镜像化大致来说有 2 种方式，一种是比较保守的方式，镜像中只包含基础环境，容器起来后，再登录到容器中部署应用包。这种方式和原先的 T4 类似，镜像化上不够彻底。为了彻底根治环境不一致的沉疴，从机制上杜绝非标准变更，让每个环境改变都沉淀下来，我们采取了另一种更激进的方式：镜像中除了包含基础环境外，还包含应用程序。应用新版本发布时，直接销毁原有的容器，用新版本的镜像启动新的容器提供服务。这样任何在上一个容器中做的小动作都会随着下一次发布全部清洗掉，如果想要保留下来，就必须固化到应用的 Dockerfile 中。一个应用镜像就代表了应用的所有依赖环境和当前版本。任何

时间任何地点将应用最新镜像拉起，都能得到和线上其他实例一致的服务和行为。我们在推广 AliDocker 的过程中，一直和所有的应用方强调这个理念，也获得了大家的认同。

于是在今年 5 月底，我们成立了专门的项目组，快速推进这个事情，目标是把双 11 流量覆盖的核心应用全部升级为镜像化模式的 Docker 应用。由于我们做到了运行态与 T4 保持一致，改造过程比较顺利，但实际上线中也遇到了许多问题，其中最大的问题就是发布与扩容速度。在镜像化之前，应用新版本发布时只需要分发应用包本身，而应用包一般只有几百 M 的大小；切换到镜像化模式之后，由于镜像包含了一个完整 OS 的 lib 库以及应用依赖的 rpm 包，往往有几个 G 的大小。应用一下感觉发布过程变得非常慢了，有时慢到难以忍受。同时这个变化对分发网络和 Docker 镜像仓库也造成了非常大的压力。在大规模发布或扩容时，很容易把仓库的下载源打挂，发布失败率居高不下，严重影响了整个发布速度和体验。为了解决这个问题，我们成立了紧急攻坚小组，集中精力对发布扩容链条的各个环节做了大力优化。

首先，在存储上，AliDocker 镜像仓库的存储直接使用了阿里云的分布式文件存储产品 OSS。阿里内部的线下开发测试环境和线上正式生产环境在网络上是隔离的，而 OSS 产品在服务端做了线上和线下的打通。AliDocker 除了用 OSS 解决了镜像存储的高可用问题以外，也借助 OSS 的这个特性实现了线下 push 镜像线上 pull 镜像的功能。

其次，在分发架构上，由于阿里在全球各地都有机房，大规模扩容时除了异地机房延迟增加之外，还有可能把长传带宽打满，所以我们在每个地区搭建了镜像 mirror 和超级缓存中心。在节点上下载镜像的单个层时采用了类似 BT(BitTorrent)的 P2P 链式分发技术，P2P 分发中超级缓存中心作为默认回源节点。这样镜像仓库 mirror 加链式分发组成的多级细粒度分布式分发网络大大加快了分发速度，彻底解决了服务端网络和存储的压力。

最后，在发布流程上，我们针对内部发布系统做了多项优化。比如采取了预热模式，线下构建好镜像后就直接异步分发到线上各地域的 mirror 中，分批发布中第一批机器在执行时，第二批机器就提前去 pull 镜像。另外把之前固定的分批发布策略改成了智能滚动分批。每个批次的机器发布执行过程中，往往总有几台长尾机器比较慢，如果等这些最慢的机器都执行完才去执行下一批，在实例

很多的时候会拖慢整个发布的过程。改成智能滚动分批后，在第一批发布完成应用 owner 确认没有问题之后，后面批次的发布不会等待长尾实例结束就会直接执行下一批，这样大大提高了并行度，减少了发布自动执行过程中不必要的等待时间。

在所有这些优化之后，实例最多(近 8 千个实例)，发布最慢的那个应用，整体发布时间缩短到了原来的 1/5，并且发布成功率也大大提高，因分发问题带来的稳定性也随之消失。

### 3 对 Swarm 的定制和优化

今年双 11 的另外一个变化是，大部分的流量都跑在了云上；交易主链路的核心应用在云上和云下都有实例。在 AliDocker 出现之前，云上和非云是两套完全不同的虚拟化机制，云上是 ECS 中直接跑应用，云下是物理机的 T4 容器中跑应用，应用下层的虚拟化机制不同也造成云上和云下只能采用两套不同的运维发布系统，让整个运维体系比较痛苦和臃肿。AliDocker 出现后，将这 2 者统一了起来，云下在物理机上跑 AliDocker 容器，云上在 ECS 上跑 AliDocker 容器，应用及上层运维系统看到的都是 AliDocker 容器。统一的机制带来更简单的实现。我们在这个时间点引入了 Swarm 来做 Docker 容器的管控。Swarm 协议基本兼容 daemon 协议，只在很小的几个点做了改动，可以将一个物理机集群当做一台宿主机来操作，为集群背景下的测试和运维带来了非常大的便捷。我们对 Swarm 做了一些改造，使 Swarm 支持批量创建容器，功能类似后来 Docker1.12 版本的 swarm 模式引进的 replica 概念。因为我们绝大多数应用最小化的部署实体都是对等部署的，多个容器中跑的是完全相同的镜像和配置，因此只是将 Swarm 中一次创建容器的请求处理过程，并行在多个物理机上执行就达到了目的。至于这多个物理机如何选出来，我们现有的资源管理平台有一套自己的调度系统，因此我们修改了 Swarm 的调度部分直接对接到了我们自己的调度系统上，同时对接了我们内部的集中式 ip 资源管理系统。每个容器的 ip 独立于宿主机的 ip，不需要端口映射，就可以直接发布到服务注册中心，供其他应用直接调用。

这个定制化的 Swarm 产品在内部称为为 AliSwarm，除了调度之外的基础功能都与官方 Swarm 相同。但是随着 AliSwarm 接入的节点越来越多，官方 Swarm 的实现部分暴露出了诸多性能问题。比如 Swarm 内部存在系统线程会随连接数增长的问题，当 node 达到一定量时大批量增删 node 很容易造成 Swarm 实例直接 crash；在 node 节点数变多的情况下，总会有那么一部分 node 节点不那么健康，在这种网络频繁时断时续的极端情况下，Swarm 的内部处理存在连接泄露问题，会使系统资源消耗越来越大；每次大规模节点上下线时 node 列表会发生频繁变化刷新，内部识别哪些是新增 node 哪些是删除 node 时做的 diff 操作会大量消耗 cpu；另外对单个容器的查询需要遍历全集群的容器信息，当节点规模在 1W 以上时，这种遍历也会消耗大量 cpu 等等；AliSwarm 在逐渐接入整个电商规模节点的过程中逐步发现这些性能问题，并一一做了解决。官方 Swarm 单实例能够平稳运行的最大集群规模，按之前官方公布的数字是 1000 台宿主机。AliSwarm 在双 11 之前实际线上运行的单实例集群规模已经在 3 万以上，并且 32 核机器 cpu 的 load 在 5 以下，在双 11 建站期间能够支持并发 3 千个以上的实例同时扩容。

节点规模变大之后，每次 swarm 自身发布初始化过程会持续几分钟，系统可用性降低。同时我们有部分业务需要独占的隔离资源池，如果为每个资源池都搭建一套 Swarm 维护成本比较高。因此我们开发了 SwarmProxy 系统，做了基于 TLS 证书管理多个集群的功能。同时每个集群运行 2 个对等的实例，一主一备，同一时间，只有主实例提供服务。主实例发生故障时可以自动切换到热备实例。这样 AliSwarm 自身版本升级时，就不需要等待新实例初始化完成才能提供服务，而是先让旧实例继续提供服务，新实例启动初始化完成后替换原来的热备实例，成为新的热备，然后再做一次主备切换完成升级。这个主备切换过程会根据版本号大小和新版本初始化完成情况自动执行，毫秒级完成。

## 4 阿里中间件 (Aliware) Docker 化

除了交易应用外，今年我们还推进了阿里中间件(Aliware)的 Docker 化，中间件相对于应用来说主要区别是有状态、性能敏感、资源需求差异大、有个性化的运维标准等。中间件的状态经过分析主要分为三类：数据状态、配置状态、

运行时状态。针对数据状态，我们通过挂载 Volume 的方式，让所有需要持久化的数据，全部直接落到宿主机的指定目录中，以保证容器升级或变更后，数据能够不丢失；针对配置状态，我们采用了镜像和配置分离的方式，容器启动时，会根据启动参数或环境变量去配置中心动态获取运行时配置，通过这种方式可以实现在任何环境下均使用相同的镜像，以减少镜像构建和传输的成本；针对运行时状态，我们制定了中间件镜像标准和运行时管控标准，对容器进行操作的时候容器管控平台会按照既定标准和流程进行，以保证整个过程平滑和能够被容器内应用感知并进行相关前置操作。

阿里中间件(Aliware)作为集团内的 P0 级基础服务，对性能的要求近乎苛刻，我们在 Docker 化之初就定下目标，性能损失需要控制在 3~5% 之内。影响中间件性能的因素主要有网络、CPU、内存、磁盘、内核、依赖包等，所以在实施过程中，我们对每一款中间件均进行了详细的性能测试，测试场景涵盖物理机 + Docker、物理机+VM+Docker、物理机+T4，经过测试、内核、网络、容器、中间件等多个团队的通力合作，最终所有中间件 Docker 化后的性能均达到预期，并沉淀出了 Docker 化场景下针对不同性能影响因素的调优方案和策略。

阿里中间件(Aliware)对资源的诉求也比较多样，比如消息中间件对网络、磁盘需求较大；缓存中间件对内存需求较大；数据中间件强依赖 CPU 性能等，针对这种情况我们在资源调度上会均衡各产品需求，让有不同资源诉求的中间件共享宿主机资源，以保证宿主机的资源使用率保持在合理的范围，既节约了成本，又提升了中间件的稳定性和性能表现。在双 11 大促备战期间，我们还对 Docker 化的中间件进行了资源的精细化调整，测试并评估了超线程( HT )带来的影响，并对强依赖 CPU 的中间件进行了调核，以规避资源争抢风险，进一步提高了中间件的稳定性，此外，我们还具备 Docker 化中间件资源动态调整的能力，比如双 11 大促前，我们可以动态为数据中间件增加 CPU 配额、为消息中间件增加磁盘配额。

在阿里集团内部，中间件(Aliware)每一款产品都有自己的个性化运维需求和流程，在这样的背景下，我们以 AliDocker 为推手，完成了中间件统一运维标准和体系的建立，并产出了中间件 CaaS 平台和场景化运维平台，中间件运维效率、资源管理效率均得到了极大提升，双 11 前我们实现了所有中间件的

Docker 化改造，并承担了线上 20%-100%的流量，按计划到财年底我们将实现中间件的 100%Docker 化。

在今年刚刚过去的双 11 中，从 AliDocker 引擎到上层运维部署体系都经历了一次大考，最终交易全链路所有核心应用全部在 AliDocker 容器中圆满完成了 1207 亿成交额的答卷，也证明了 AliDocker 技术体系在电商交易级别的大规模应用中能够担负重任！

# 第五章 业务架构

## 5.1 内容+：打造不一样的双 11

作者：建瓴，神照，随喜，志向，元超

### 1 站上风口的内容经济

内容永远是最优质的流量入口。如果不是，那说明你没做对。

今年是淘宝全面内容化、社区化的元年；今年的双 11，也是一场具有丰富内容、精彩互动的购物狂欢。从必买清单、大咖直播，到 KOL 草地、人群市场，双 11 零点时分经历的淘宝网红经济爆发，都是今年独特而又亮丽的风景线。

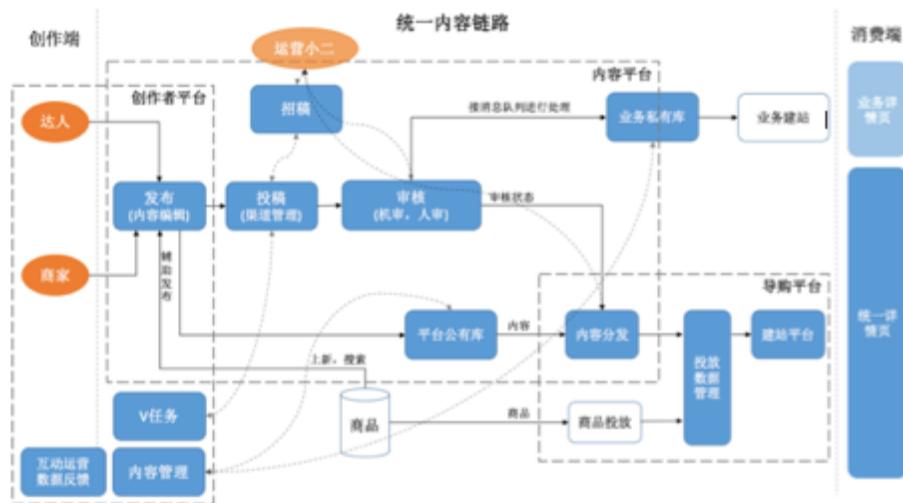


用户越来越注重消费内容的质量，也逐渐愿意为非功利化的优质内容买单；相反，那些关联性的内容，必将越来越被用户排斥。在淘宝，内容化的口号虽然是近一两年才提出来，但处于风口的这头“猪”其实已经养了很多年了：清单、

问答、直播等业务固然给我们带来了更加丰富的互动内容形态，然而商品、店铺详情页上的图片文字，何尝不也属于泛内容的范畴？我们今天说内容化，本质上是内容形态的进化，从过去单一、简陋、缺乏用户亲和力和参与感的内容，向着更丰富、更专业、用户更加喜闻乐见、转化效率更高的形态去进化。猪已肥，风正来。

兵马未动，粮草先行。我们从去年开始着手建设服务于淘系内容业务，从创作沉淀、到管理投放、到前端建站、到数据反馈的一整条闭环链路，在今年新推出的内容型导购产品、以及双11购物狂欢节的各大会场中，都获得了广泛的应用和良好的效果。这条链路包括：

- 创作者平台：服务于达人、商家等的内容创作者，提供认证进驻、内容发布与管理、粉丝互动运营、参与营销任务、权益与成长、效果数据反馈、以及V任务等商业化体系搭建等功能；
- 内容平台：服务于内容业务方，聚焦在核心内容数据链路、统一发布端、统一详情页、内容抓取、内容质量与组织、筛选投放、分佣结算的基础能力的建设；
- 导购平台：服务于淘宝行业和导购产品，对内容、商品进行再加工再组织，赋能各种导购场景的高效搭建。



接下来，我们具体介绍这条内容链路，以及在双11大促期间的应用案例。先从位于这条内容链路核心的内容平台开始说起。

## 2 蜂巢内容平台简介

蜂巢(Beehive)内容平台诞生的背景，是以爱逛街、极有家、iFashion 为代表的一批内容导购产品迅速崛起、获得越来越多用户的青睐。然而业务野蛮生长的同时，也带来了一系列的问题：

- 内容业务的研发需求得不到快速满足，背后是多项目多系统的井状独立部署，每个业务都独立建设内容模型和能力，不同业务之间无法重用，导致开发和维护成本很高；
- 内容与创作者沉淀不足，缺乏规模化，主要原因是缺乏平台化的创作者进驻、管理以及内容创作工具，不同业务场景的内容也没有进行标准化和结构化沉淀；
- 内容的触达场景和影响力有限，缺少分类、打标、检索、推荐等丰富的内容浮现机制，不同业务场景之间的内容无法互通、共享；
- 内容的商业价值有待更充分地发掘，基于内容产生互动等场景下的丰富的多方分佣机制有待探索。

与从某个成熟业务沉淀出来并推而广之的路线不同，蜂巢内容平台与生俱来地就肩负着承载淘系多个不同的内容业务的使命。从着手平台架构设计的第一天开始，研发团队就在对来自不同业务的内容数据进行标准化建模，对内容能力抽象共性的需求，目标就是为了提供通用的服务能力、实现“搭积木式”的快速迭代。

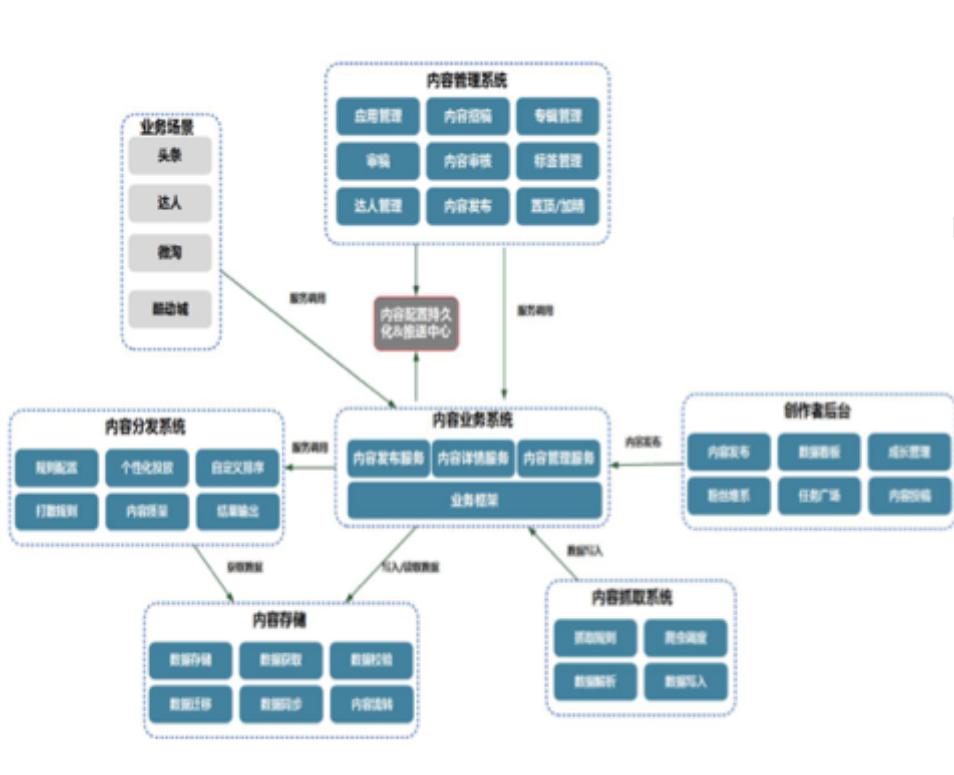


### 蜂巢 ( Beehive ) 寓意

- 形状上，巢房是一个个六角形，结构非常精致，看上去像是一张非常漂亮的网，寓意着内容平台希望将内容统一组织、相互连接、相互分享的目标；
- 功能上，各个巢房都有不同的功能分工，各司其职、灵活高效，寓意着内容平台有着高度可扩展的架构，能够为复杂多变的业务场景提供强有力的支持；

为什么叫“蜂巢”？有如下寓意：

经过一年多时间的建设，经历了从 1.0 到 2.0 的版本升级，目前蜂巢已经成为一个快速构建内容业务的核心技术平台，提供稳定高效的数据服务，灵活可扩展的应用逻辑及商业化能力，满足业务在内容生产与审核、存储与组织、分发与流通、互动营销等方面需求。已经沉淀了数亿内容，支持包括淘宝头条、必买清单、淘宝直播、微淘、和垂直行业等近百个业务、1000 多个线上场景，每天为数千万消费者提供丰富、实用的消费资讯。



## 2.1 赋能内容创作者

双11大促期间，绝大部分内容是由专业的内容创作者生产并提供的，我们称之为达人。达人招募、内容生产、和效果反馈，都在达人平台(daren.taobao.com)上进行。

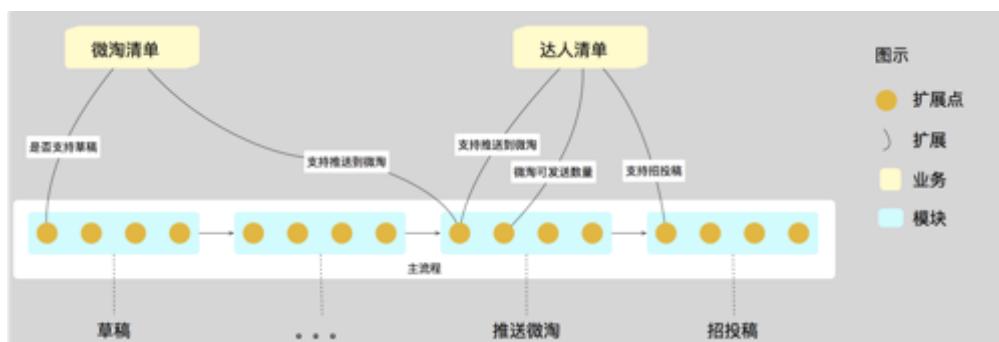


这次大促有很多会场，每个会场背后是不同的业务方。大促准备过程中，每个业务方有各自的诉求，变化也很频繁。面对这种情况，我们把内容体系抽象成了若干可定制化的模块，通过配置和 SPI，让运营小二在后台自主完成业务接入和变更，极大地提高了工作效率。

The screenshot shows the 'Daren Platform' interface. The left sidebar has a dark theme with white text and icons. It includes sections like '粉丝' (Fans), '订阅号' (Subscription), '内容推送' (Content Push), '我的问答' (My Questions), '频道' (Channel), '频道投稿' (Channel Submission), '频道申请' (Channel Application), '任务' (Tasks), '阿里V任务' (Ali V Tasks), '统计' (Statistics), and '内容分析' (Content Analysis). The main content area is titled '首页 > 频道投稿' (Home > Channel Submission). It features a navigation bar with links to '首页', '中国质造', '玩客', '文化中国', '酷动城', '亲宝贝', '全球购', and '有好货-专题'. Below the navigation is a row of circular icons: '图文投稿' (Image and Text Submission), '已投稿列表' (List of Submissions), '头条问答' (Headline Q&A), '头条草稿箱' (Headline Draft Box), '头条数据' (Headline Data), and '草稿箱' (Draft Box). A table titled '已投稿内容' (Published Content) lists one item: '2016-11-17 15:14:02'.

在达人平台，每个业务方表现为一个渠道，渠道内可定制招募流程、达人准入门槛、审核权限、层级和权益，这些都由业务方负责总体运营的小二操作。一个业务方在不同场合对内容要求也可能有差异，这就需要在渠道内配置不同的频道。以“必买清单”为例：日常的“攻略”、“双11预售会场”和“双11清单会场”就是三个不同的频道，有各自不同的门槛、生效时间、以及审核流程。另外，同样是清单，但大促会场和日常活动在标题长度、宝贝数限制上都有不同，所以达人平台也在频道层级提供了定制编辑器的能力。同一渠道下所属频道的配置互相独立，相关运营小二互不干扰。

作为内容编辑器和发布数据链路的提供方，Beehive 平台提供了统一、灵活的内容发布相关组件，以及供业务定制的扩展点机制。当一个业务方需要内容发布相关能力时，只需要定制下相对应发布表单，即可快速实现内容发布需求，而不需要业务方过多地关注底层的内容模型细节。以上述必买清单为例，运营小二可以基于发布组件去配置三个不同的频道，达人也可以把所创作的内容灵活地投稿到这些频道中。

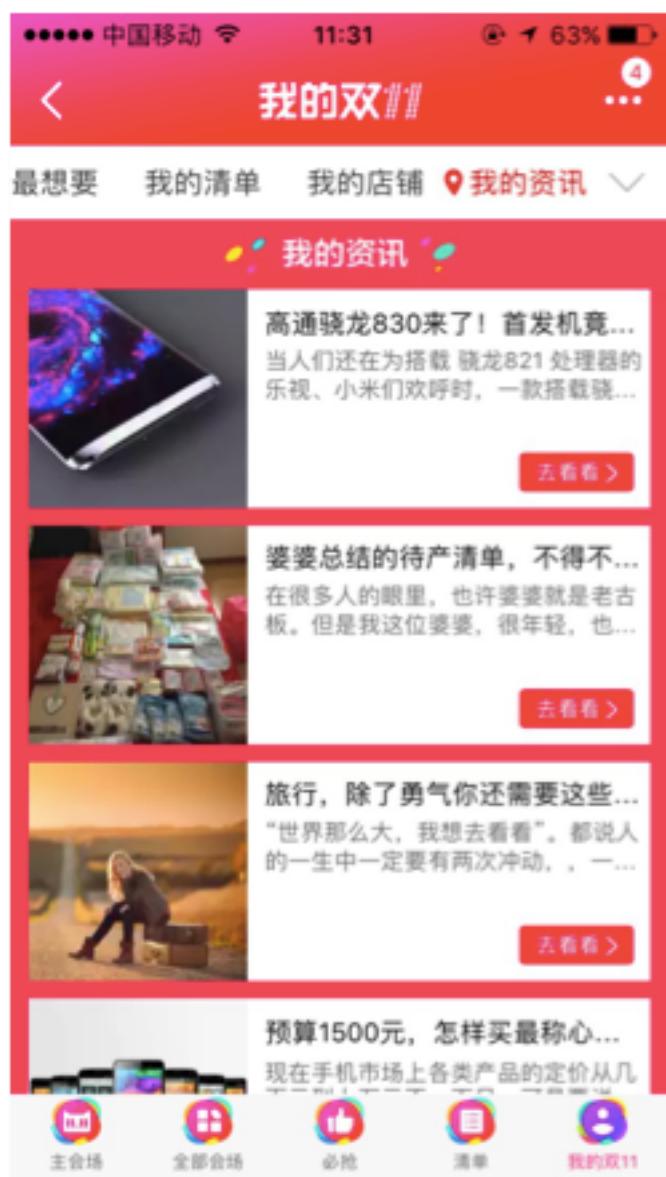


依靠平台化的内容链路和强大的配置能力，在今年双11过程中，团队只用很少的开发资源，就支持了清单、人群、爱逛街、有好货、KOL、和淘宝头条等首屏会场，以及若干行业会场的内容生产，取得了真正赋能创作者、支持业务快速搭建的良好效果。

## 2.2 优质内容的个性化分发

在已经沉淀的近 6 亿内容中，如何去芜存菁、发掘和浮现优质内容，并通过定制化或者个性化的规则机制进行内容的跨业务流通、多场景实时分发，是我们建设内容链路的一个核心能力。

下面就以今年双 11 手机淘宝“我的资讯”模块为例，具体介绍内容质量和分发体系，是如何构建并帮助业务选择并获取到优质的内容数据、最终完成个性化输出。“我的资讯”模块如图：



作为双 11 会场透出的内容，要求是非常高的：一方面要求内容里的商品有一定比例是参加双 11 大促的商品，另一方面也希望内容的图片精美、文字优雅。业务方都希望内容有较好的使用效果，因此我们将内容的质量体系分成了两个方面：

- 内容健康分：也就是内容自身的质量。内容从组成元素上看，有三个基本的部分：文本、图片和商品，我们基于这三个维度，设计了 17 个健康分的指标，以细化内容质量；
- 内容效果分：主要刻画内容被使用的效果，从停留时长、商品点击率、曝光机会等 25 个维度去刻画；对于新发表的内容，为了解决冷启动的问题，我们也通过机器学习的方法，完成了效果预测分，作为内容效果分的一部分，供创作者和业务方参考。

在双 11 我的资讯会场中，产品的要求是：资讯下的商品有半数及以上的双 11 商品，以及对 KOL 达人的内容进行加权。按这个条件初步筛选出来的内容非常多，有数千万条，然而其中大部分作为“资讯”都是不合适的，比如相当一部分的文本描述较短，图片可能带有牛皮癣，另外还有时效性等问题。后来用上了内容健康分中图片、文本等一些维度进行二次过滤，质量提高了很多。

有了质量体系以后，内容的流通和分发就更灵活了，运营可以使用内容分发系统 - 点将台，功能架构图如下：

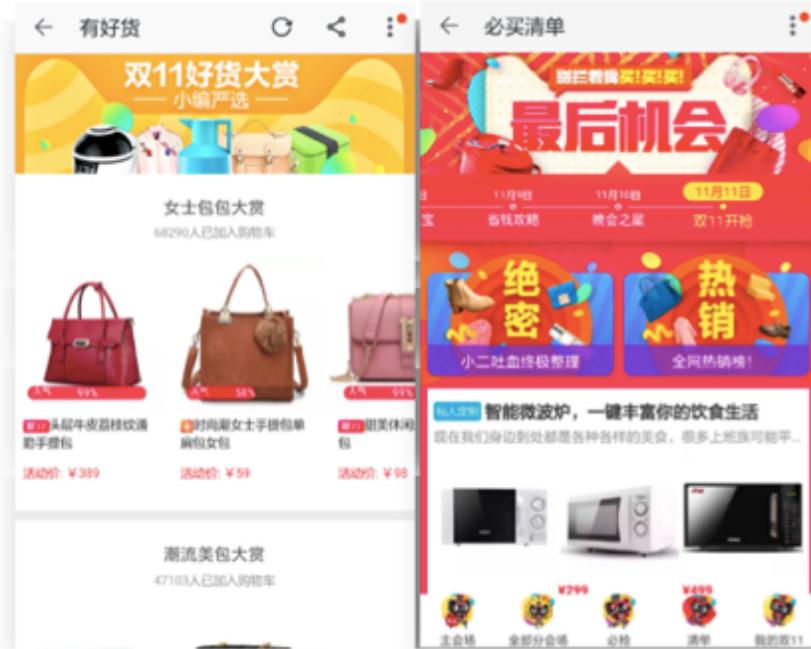


从上图可以看出，一方面，点将台集合了各种内容来源，通过标准化和内容多维度的打标，结合内容质量体系，提供数百个筛选内容的维度，帮助业务方很灵活地选出所需要的优质内容；另一方面，也提供了在线、离线、个性化投放、消息队列等强大的输出能力。正是因为有了这样的能力，今年双 11 “我的资讯”会场，即是通过点将台完成内容的筛选、分发、个性化投放，最终呈现给用户。

### 3 内容消费促进导购升级

“万能”的淘宝，在面对海量消费者强烈的多样性诉求下，势必需要在各个垂直行业以不同的心智去连接消费者；而淘宝导购产品作为建立用户粘性与认知的桥头堡，必然也需要在做好“多样”的同时兼顾“效率”，这就对整个平台提出了很高的要求。所以 2016 年，淘宝在多年导购业务经验积累下，整体抽象出内容/建站/流量/效果等核心的导购产品要素，提出了面向整个导购业务域的平台化升级战略，逐步将传统烟囱型的导购产品升级转型到平台化架构中，以求在提升产品迭代效率与整体导购域稳定性的同时，结合内容化、社区化的元素，在各行业中给用户呈现更酷炫的导购体验。

下面以几个典型的导购产品为例，“有好货”与“必买清单”是大家比较熟知的淘系导购产品。在今年双 11 中，首先它们都依托统一的平台化系统支撑，高效对接到统一的内容生产/分发链路，包括前述的蜂巢内容平台和达人平台；其次，结合淘宝沉淀的一体化营销会场建站能力，让前端团队能够以模块化方式快速完成页面的搭建，让运营团队可以直接通过横向运营工具，快速即时的完成对于导购内容（帖子、商品、店铺等）的筛选、聚合、与投放。



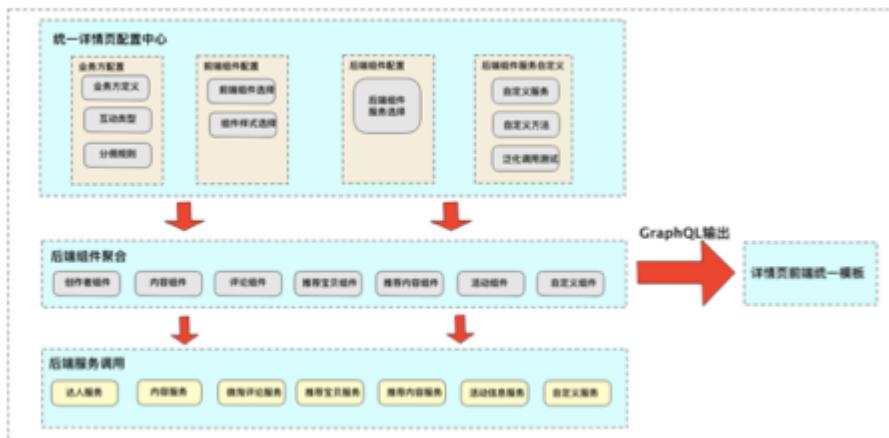
从系统角度，我们站在整个导购业务域的角度做出一套横向整体的稳定性方案，来面对双11零点这样的瞬时脉冲流量，相比于传统的垂直烟囱系统的设计方式，这无疑给容量规划、稳定性预案、统一兜底降级策略的实施，带来了革命性的提升。同时，我们还针对导购入口的个性化所见即所得运营、A/B test 数据化运营支撑、无线端 native 化渲染进行单点突破，基于整个导购业务域产品诉求的抽象造出“轮子”，装上“汽车”，让所有不同体量、不同用户群、不同定位的导购产品同时受益，这一切对于研发效率以及稳定性保障带来的提升，尤其在今年大促中得到了体现。

正是在这样一套贯穿前后的平台化方案支撑下，今年整个淘宝导购产品矩阵成功地将日常状态下的用户心智与粘性，带到了双11当天，让用户简单清晰地完成浏览路径的选择，从而带来更好的点击/成交转化。

用户从导购业务进入，并最终到达内容曝光的阵地：内容详情页。不同业务方对内容详情页一定存在个性化定制的需求，然而经过我们的仔细分析后发现存在比较多的共性。在 Beehive 平台会对详情页的各个组件统一建模，提供了统一的详情页。详情页提供了一套内容相关的组件，业务可以快速基于已有的组件定制一个业务自身相关的详情，而不需要过多的开发代码；当一个业务存在组件

定制需求时，业务只需要根据自身业务定制自己的组件或者服务，并注册到详情配置中心，然后由详情服务统一进行渲染。

在实现上，同一个组件在不同业务中很有可能信息的来源是不同的，比如创作者组件中，有的创作者信息来自达人，有的来自商家。内容平台通过对组件的建模，实现内容详情页在展现形态上的统一，同时也支持了数据的个性化要求。最后，通过 GraphQL 聚合能力来进行数据组装。业务方可以通过参数来决定哪些组件需要在详情页中出现。



面向未来，在我们逐步完成内容导购的数据闭环之后，统一详情页可以帮助我们积累大量的内容、商品消费数据反馈。在提升运营效率的同时，更有价值的是将传统人工的运营经验数字化，畅想真正的“智能化导购运营”来到的那一天，基于消费者、商家、内容生产者长年积累的海量结构化信息，再加上历史上运营经验的积累，将让机器站在整个平台的肩膀上，根据运营核心策略，自动构建出无数个垂直行业的“小有好货”、“小必买清单”、“小人群市场”，这将是我们接下来的目标。

## 4 内容商业化

在一个健康的内容生态系统中，创作者生产内容，业务方选择并使用内容，消费者消费内容，而平台方则需要提供合理的商业化机制，来促进创作者有可持续的动力，源源不断地产生优质内容，促进内容生态体系的整体繁荣。在内容链路上，商业化主要围绕达人身份体系及导购内容进行建设，目前提供的商业化能力包括 CPS 分佣和 V 任务。

## 4.1 CPS 分佣

CPS 分佣依靠阿里妈妈淘客联盟，给内容中包含的商品提供淘客 CPS 链接生成服务，和淘客 CPS 点击行为注册的服务。通过这两个服务，买家通过点击内容中展示的商品下单并完成交易后，内容的创作者、业务方、以及阿里妈妈都可以得到一定比例的佣金，这就是最常见的三方分佣机制。

CPS 分佣能力由蜂巢 Beehive 内容平台提供。不但支持不同的业务方灵活地配置分佣比例，还创新性地引入了内容质量体系，作为动态佣金（奖金）的评判标准，这样更能激发创作者创作更多优质的内容。

## 4.2 V 任务

淘宝在内容展现的过程中大量引入了个性化，从而带来了较大的不确定性：如果仅依赖 CPS 分佣的变现方式，达人不知道自己生产的内容何时会被展现并获得收入，商家也不知道怎样定向找到与宝贝特点吻合的达人去做推广。V 任务作为新一代内容变现平台，很好地解决了供需双方之间的不确定性问题。



在 V 任务中，一个“任务”就是达人和商家间达成的合约，约定达人要生产包含指定宝贝的内容并投放出去。内容的形式很丰富，包括帖子、清单、视频、直

播等；订立合约的形式也很灵活，可以由商家发起达人响应，也可以由达人发起商家响应，甚至可以点对点邀约。这次双11大促中采用比较多的是后两种方式，这主要是因为大促期间，达人在有限的时间段内对自己生产的内容有更大的议价权。

创建完协议后，V任务会利用交易提供的履约能力，保证这份协议的履行。V任务还打通了所有数据链路，以保证签约、履行、反馈、和付款的完整流程，都能在线上自动完成。

达人执行每次任务的效果都会反馈到V任务上的达人指数中，通过大数据的积累，商家可以在此找到最适合推广自己宝贝的达人。而对达人来说，只要完成任务就能收到钱，避免了辛苦生产的内容因为没有被个性化系统选中展示从而没有收入的窘境，增加了确定性。正因为这两点，V任务在达人和商家间都广受欢迎。

通过CPS分佣和V任务等商业化机制，保障并极大地激发了创作者的热情，确保了内容生态体系源源不断有新的优质内容流入，商家的营销任务能得到更有效的执行，消费者也能获取更好的内容体验。

## 5 未来展望

金秋十月的云栖大会上，马老师第一次提出了“五新”的概念：新零售、新制造、新金融、新技术、新能源，在不久的将来将深刻影响全世界人民生活的方方面面。新零售有很多不同的表现形式，个性鲜明、富内容化、强调IP和互动的网红经济，就是新零售业态的一种。而新能源讲的就是越用越有价值的数据，通过历史数据的积累和价值发掘，在未来能够不断地拓展新的业务边界。

在过去的一年多时间里，我们很荣幸能作为平台化、系统性地去储备和使用内容数据这种新能源的第一代矿工，建设了从生产到沉淀再到消费的、端到端的

完整内容链路，并成功经历了这次双 11 的战火洗礼，与各个合作业务一道用内容化打造了一个不一样的双 11。

正如本文开头所说，高质量的内容代表着最优质的流量和关注度；与未来各种新的电商业务形态的结合中，内容将是催化剂。我们期望与各个合作团队一起，让内容大数据这种新能源，释放更大的能量。

## 5.2 双 11 交易核心链路的故事

作者：若海 之魂 屹正

### 1 多元化的玩法，海量的成交背后的核心链路

第八年的双 11 全球狂欢节已经落下帷幕，让人印象深刻的 1207 亿再次创造了历史，双 11 也从此进入了千亿时代。今年双 11 宗旨“全球化、娱乐互动化、无线化、全渠道”在活动中体现的淋漓尽致，从狂欢城、线下 VR 游戏捉猫猫、到双 11 万星璀璨的晚会，让无数人惊叹双 11 成为全民娱乐盛宴。在娱乐狂欢的同时，双 11 回报消费者最直接的就是买买买，“买出新生活、买出新高度”代表了无数剁手党的声音，为了让消费者买的爽、买的值，天猫也是用尽了洪荒之力，从全球精品的低价预售，到狂欢城数十亿的红包、购物券，双 11 当天更是火山红包、密令红包雨下个不停，再到品质商品半价、下午场满 n 免一，让消费者经历一波又一波高潮，停都停不下来。

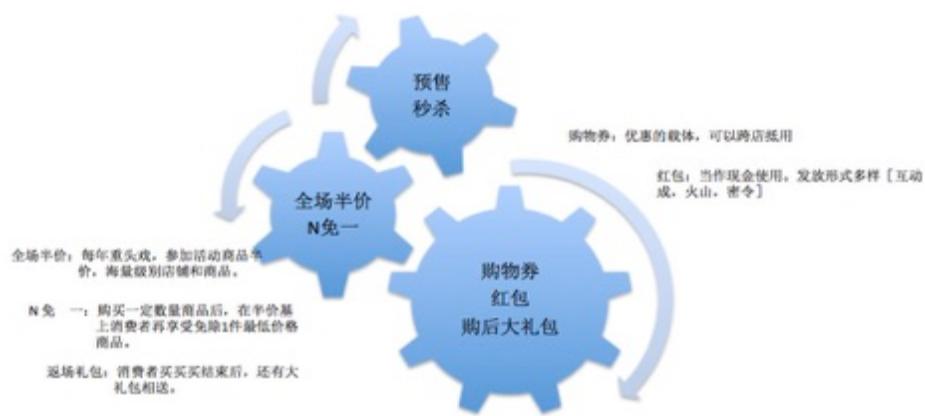
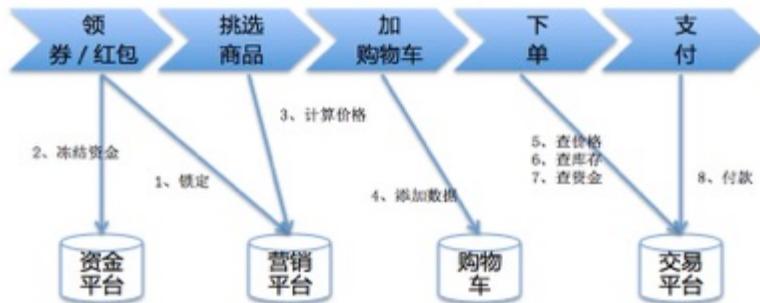


图 1 双 11 多样的玩法

每笔剁手之旅的背后都经过了哪些产品系统的处理，瞬间的剁手快感下面又发生了什么样的信息速递，我们通过一张图来感受。



- 1) 领券、红包，领取券和红包先查询营销卖家是否提供该资源，消费者是否符合领取条件。符合的基础上锁定资源，扣减资源库存，建立用户、资源关系，并通过资金锁定券红包对应的资金。
- 2) 选商品，商品展现时如何呈现商品的价格，不同的活动及用户计算出唯一的价格（人群、终端、渠道、商品等条件）。
- 3) 加购，消费者进行加购的过程中形成类订阅关系，在商品状态发生变更的时候能进行动态更新状态。
- 4) 下单，前面一系列动作到了完成订单的时刻，首先查询商品价格，匹配出当前消费者最优价格，请求库存、限购等来判断是否有商品可售卖，再次查看消费者对应的营销资产可使用与商品并进行优先级排序，确定后创建订单。
- 5) 付款，获取订单上消费者所用的营销资源（券、红包、积分等），调用资金平台进行锁定资金，剩余款项由支付渠道完成（支付宝等），完成后进行营销资源对应的资金进行划转，完成订单支付。

图 2 交易核心链路示意图

令人眼花缭乱的活动，背后是怎样的体系来支撑起这样多元化的玩法，接下来我们一层层的来解构。在每秒 17.5w 笔的高峰下，每笔订单所经的链路众多，如何海量的订单有序、准确的运行，如丝般润滑，需要完整的业务、技术架构和对高峰时刻技术的攻坚。

## 2 核心链路 - 红包系统面临的业务技术挑战及解决之道

### 2.1 红包发放

红包发放需要保证精确的预算控制，即一个预算发出去的红包的总金额不能超过其预算金额。一个预算维护在一条 DB 记录中，对于红包火山等大访问量的活动，这将成为单点热点瓶颈，同时还会导致单表记录过多，数据严重倾斜。

预算控制因与买家无关不需要实现单元化，而用户的红包信息需要实现单元化，发放不单需要考虑预算的扣减，还需要考虑预算扣减后用户侧需要尽快透出红包。因非单元化与单元化数据分布在不同的数据库以及机房中，这会导致跨机房调用，引入了更多不确定性因素。

#### 2.1.1 挑战

如何应对单预算超过几十万的发放 QPS，以及在 2 秒内透出用户获得的红包。为保证精确的预算控制，决定采用数据库的方式来管理预算，而按照之前的常规优化方式，单记录的发放 QPS 上限不超过 4 百。跨机房的网络通信也不能保证 100% 畅通，万一预算扣减成功后用户红包数据写入失败了如何感知并处理。

### 2.1.2 分桶方案

通过分析过往活动数据，我们将预算拆分为多个子预算，然后平均分布到多个数据库中，并根据红包发放请求中的用户 Id 路由到不同的子预算中。在子预算余额不足时，则路由请求至主预算中。多个子预算的扣减并不会完全相等，那么必然有一部分先发生余额不足，所以主预算需要比子预算多分配一些金额。当多个预算的余额都非常少的时候，可以进行子预算的回收动作，避免预算划分的碎片化。

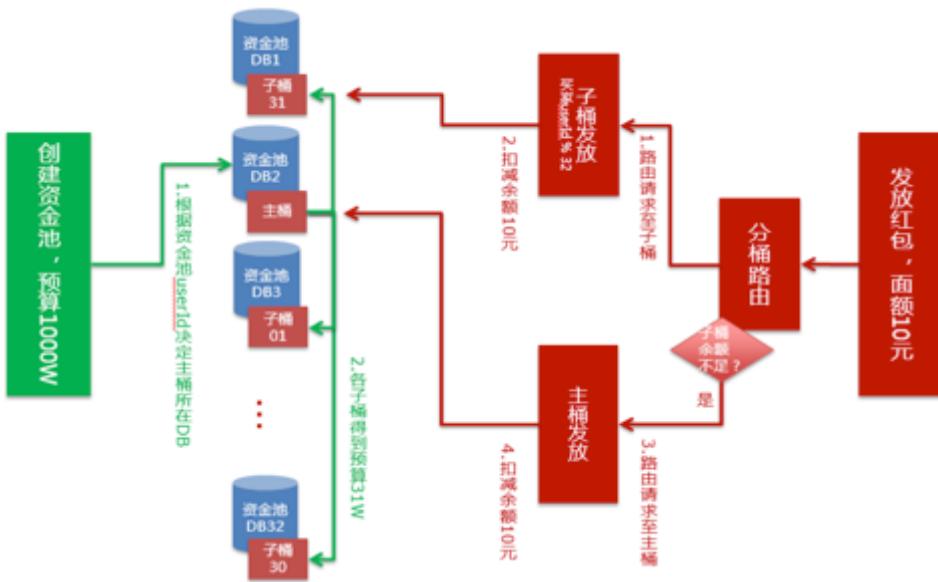


图 3 分桶方案

### 2.1.3 SQL 优化

为了提升单记录的写入性能，我们对写入的 SQL 做了优化。红包发放的场景主要有三条 SQL，两条插入语句和一条更新语句，更新语句是导致热点问题的根源，为了减少更新导致的独占锁，我们使用将三条 SQL 语句放在一起，仅通过一次网络传输就可以到达数据库服务器中。同时为更新语句设置条件以保证更新后的余额大于等于零，这样的写法可以做到扣减之前不用查询预算的余额，仅通过判断 SQL 返回的错误码就能识别是不是余额不足，减少数据库服务器的压力。为更新 SQL 添加 COMMIT\_ON\_SUCCESS 标签，保证事务成功后立即提交当前事务，不用等待客户端获得更新结果后再次发起 COMMIT 请求，减少了 1 次来回的网络交互，同时记录的独占锁可以立即得到释放。为更新 SQL 添加 TARGET\_AFFECT\_ROW 1 标签保证如果满足条件的记录不存在，事务应该失败因不是成功并且影响的行数为零。

通过和 DBA 的沟通，我们还采用了集团 DBA 团队开发的数据库热点补丁，

并且修改了数据库的磁盘写入参数，以 10 个事务为一个写入单位。

#### 2.1.4 效果

采用前述的 SQL 优化方案后，单记录的发放可以稳定在 8 千左右的 QPS，摸高到大约 1 万 QPS。在 32 个库联合压测的过程中，我们发现了因 sequence 分配原因导致无法超过 5 万 QPS，对 sequence 的分配 SQL 采用前述的优化思想优化后，顺利达到接近 30 万的 QPS 的性能。此时数据库与应用的性能都表现都比较平稳。

## 2.2 红包展现

当用户获得红包后，我们需要在多个透出界面中展现用户的红包。这些信息将包括用户一共有多少个红包可用、总计金额多少，以及详细的红包信息。统计这些信息是一件比较消耗数据库性能的事情，而拥有红包的用户是很可能在短时间内多次查询这些信息的。

#### 2.2.1 缓存方案

为了减少数据库的减力，很自然的方案是使用缓存，记住用户前一次查询的结果。而问题的焦点就转变为了如果确定缓存应该失效，我们要在红包的多个生命周期节点里做关联逻辑才能确认缓存的失效，而这样的改动工作量会比较大，同时多次这样的工作很可能是没有意义的，因为用户很可能一次也没有看过他的红包。因为我们每次在进行红包状态的更新时，都会更新 gmt\_modified 字段的值为当前时间，这是一个通用准则，于是我们有了如下的缓存方案：

构造缓存时，采用事务内的一致性读取数据库当前时间做到缓存生成时间，如果该用户没有红包数据，那么缓存生成时间取 2000 年 1 月 1 日。当用户查询红包数据时，我们先执行一次 SQL 查询返回用户对应红包数据的最后更新时间，“然后比较这里返回的时间与缓存生成时间的关系，当用户红包数据的最后更新时间大于或等于缓存生成时间时，判定缓存失效。这样的判断方式非常准确，同时因为可以利用数据库索引且仅返回很少的信息，数据库性能消耗的并不多。

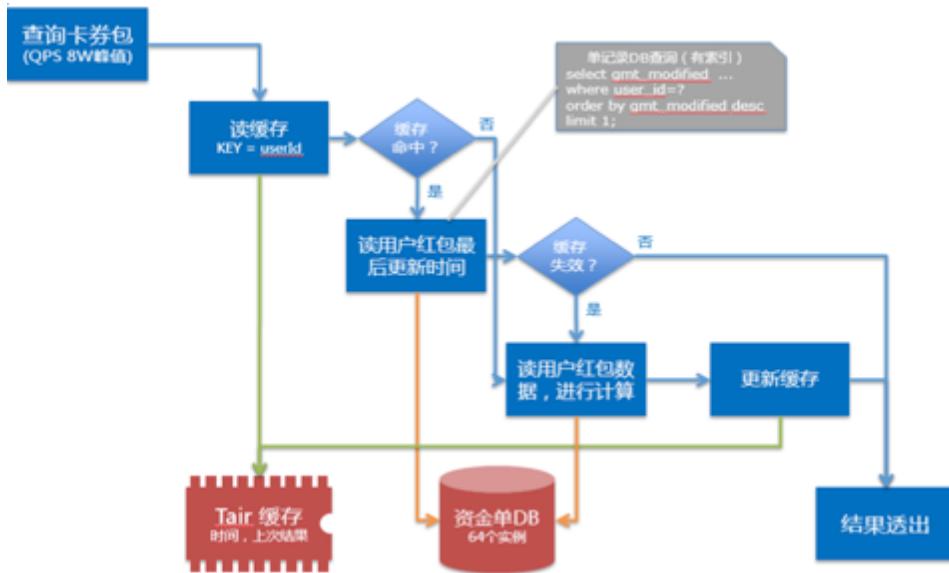


图 4 红包查询方案

## 2.3 红包使用

红包使用的场景需要支撑超过 8 万 QPS 的峰值。业务上的规则是一次下单最多可以 10 个红包，那么对于我们来说，将需要更新 10 个红包的状态，产生 10 条红包的使用流水记录，并且还需要产生至多 10 条红包相关的业务单据。而通常的情况是用户的一次下单行为所涉及的红包会被全额使用。

### 2.3.1 SQL 优化

通过 SQL 分析我们发现在红包使用场景中数据库花费了大量的 CPU 资源进行 SQL 解析，同时一次下单涉及的 SQL 语句也很多，有非常多的网络消耗。而 mysql 并不像 sqlserver 等数据库那样支持预编译，但好消息是 mysql 支持 batch insert 语法。于是我们可以使用 batch insert 的语法来优化插入性能，使用一条 UPDATE SQL 更新多条红包的方式来优化更新性能，然后再将些 SQL 使用前面提到过的合并优化方案合并为一条大的 SQL 语句发送给数据库服务器，减少网络交互。假定本次用户下单时全额使用了 3 个红包，那么可以通过一个 SQL 将三个红包的余额更新为零，同时还使用了金额锁保证红包并没有其它的事务并发更新。同时为更新语句添加 TARGET\_AFFECT\_ROW 3 的标签，如果红包已经被其它订单在并发场景下使用，那么它会使当前事务失败，并且我们可以通过数据库返回的错误码识别这一情况。

### 2.3.2 效果

通过前述的优化措施，在典型的场景下，数据库服务器的负载下降了至少 50%，而且下单的整体响应时间也减少了至少 50%。

## 2.4 红包系统可靠性保障--一致性消息通知 ( hibus )

前面有提到红包的发放只是扣减了非单元化场景的预算，然而我们还需要在预算扣减完成后写入单元化场景的用户红包数据。它们处于不同的数据库，我们需要有机制来保证它们的一致性。除了一致性，还有时效性的需求，因为我们需要在发放成功后 2 秒内就能展现用户的红包数据。普遍的方案使用跨库事务框架来解决问题，但是轻量的跨库事务方案不能做到严格的事务一致性，而严格的跨库事务一致性方案显然是相当重的，它会极大的降低性能。加上我们还有内部业务流程串连的需求，所以便产生了一致性消息通知-hibus。

ibus 的思想是在业务的事务中插入一条消息记录，建立一套消息的订阅与分发系统来支撑消息的处理。因消息记录与业务记录存在于同一个数据库中，可以做到事务一致性。消息记录并不大，并且多个订阅者共享同一条消息记录，因此并不会增加过多的数据库性能消耗。目前我们已经可以做到生成的消息在 1 秒内就可以消耗，因此可以保证用户的红包数据透出时效。

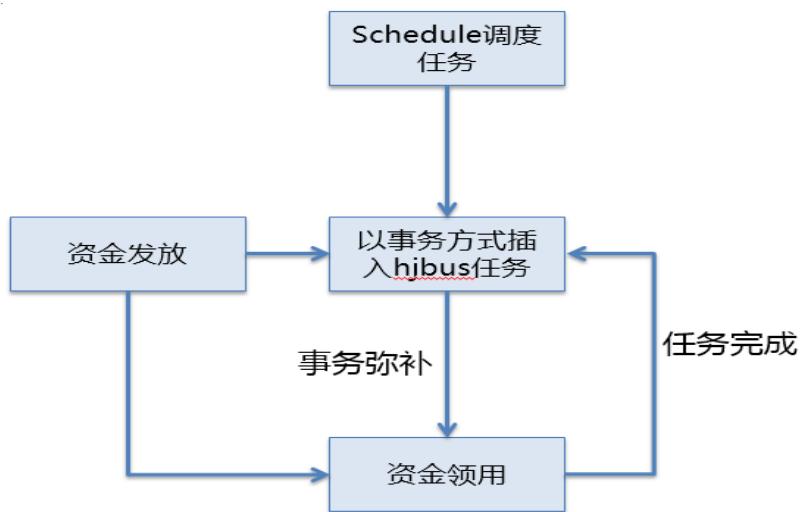


图 5 一致性消息通知-hibus 原理示意

同时通过消息积压的监控，我们可以及时的发现哪些消息的消费出现了问题，出现了什么问题，保障业务各流程的完整性。在一系列的优化工作完成后，我们

的红包系统在双 11 前后都表现的相当稳定。

## 3 核心链路 - 交易系统面临的业务技术挑战以及解决之道

双 11 当天交易下单峰值达到了创纪录的 17.5 万笔每秒，而每次下单都需要完成优惠的计算、红包的使用等一系列的操作，对整个系统的调用量实际远远高于 17.5 万每秒。如此高的并发，加上交易系统在数据上不能出任何差错的特点，对系统的性能和稳定性方面的要求都非常之高。

### 3.1 前置处理，提升性能和稳定性

我们的下单系统需要访问物流系统获取运费模板，并计算运费的价格，在以前的架构中，这需要调用远程的一个系统，由那个系统准备好相应的数据，计算结果后返回。这种方式会把下单的峰值带到下游所有的依赖系统中，也就会要求下游依赖系统具备峰值同等级的能力，这不仅使得整体成本很高，而且下游系统的稳定性也会影响整体下单的稳定性，在依赖关系非常复杂的交易系统中对整体稳定性带来了很大的挑战。

把功能前置后，需要计算运费时，下单系统无需请求一个下游系统，而是直接访问存储着运费模板数据的缓存服务器，并通过前置在下单系统中的运费计算模块，直接在本地计算出运费。这种方式不仅减少了远程服务调用的网络开销，带来了性能的提升，同时也减少了下单系统依赖的下游系统个数，增强了系统的稳定性。

我们对很多下游调用采用了功能前置的架构优化，通过双 11 当天的验证，这种方式在高峰情况下表现突出。

### 3.2 架构升级，提升开发效率和可靠性

双 11 的挑战不仅仅是性能和稳定性，由于参与双 11 的业务团队数量越来越多，业务玩法也越来越丰富，所以直接参与到双 11 业务开发的团队个数和开发人数也越来越多。众多的团队和开发人员，同时在一个平台上开发，怎么提升开发效率也是很关键的一个点，这关系到是否能按时完成需求的开发。

针对多团队协同开发的场景，交易平台去年完成了架构的升级，在新 TMF2 框架（见图 6）下，业务级的代码和平台级的代码进行分离，平台级代码对于交易的能力进行分类、抽象并对外透出，业务方的开发团队能力自助地利用平台提供的能力完成各自业务的定制逻辑的开发，无须依赖平台团队介入，从而大幅度提升整体业务开发效率。



图 6 TMF2 整体架构

TMF2 作为业务与平台分离的业务框架，主要通过两大类模型（能力模型、配置模型），并基于这两大类模型生成的配置数据来贯穿两个业务活动主线（业务配置主线、业务运行主线），详见图 7。

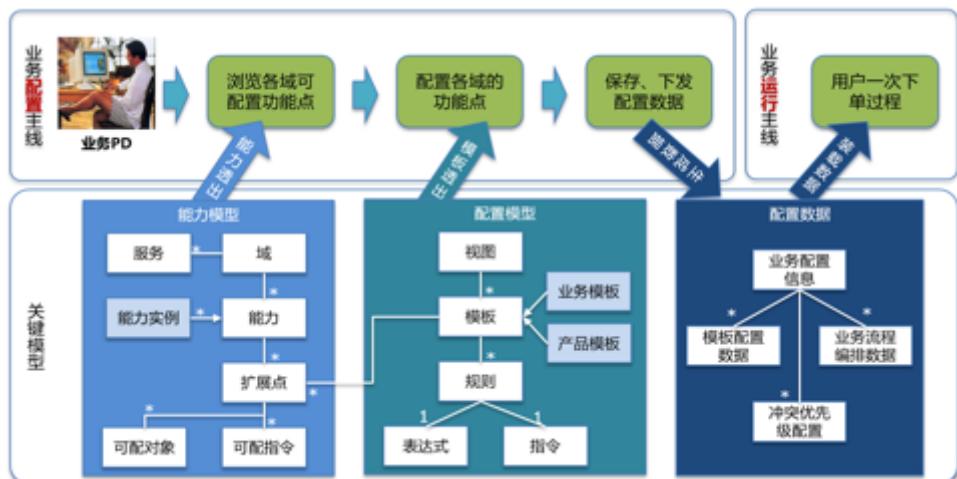


图 7 TMF2 框架模型图

通过对交易建模、抽象和收敛，形成了交易域基础能力层，提供了各交易需要使用的核心能力，采用功能域 - » 能力 - » 扩展点方式进行表达。

例如，在合同订立（下单）环节归纳有十几个功能域，比如优惠、支付、交付、价格、订单、结算、税费等。这些域里的能力又通过扩展点的形式开放给业务方开发做定制，以适应其不同业务场景的需求。

在此基础上还引入了产品的概念，它是包装了多个域的能力，对业务方提供了能满足某种业务功能的能力包，从而使得业务能直接使用，加快业务开发效率。比如预售产品。业务能力和产品，通过业务场景串联，从而形成一个完成的业务解决方案。

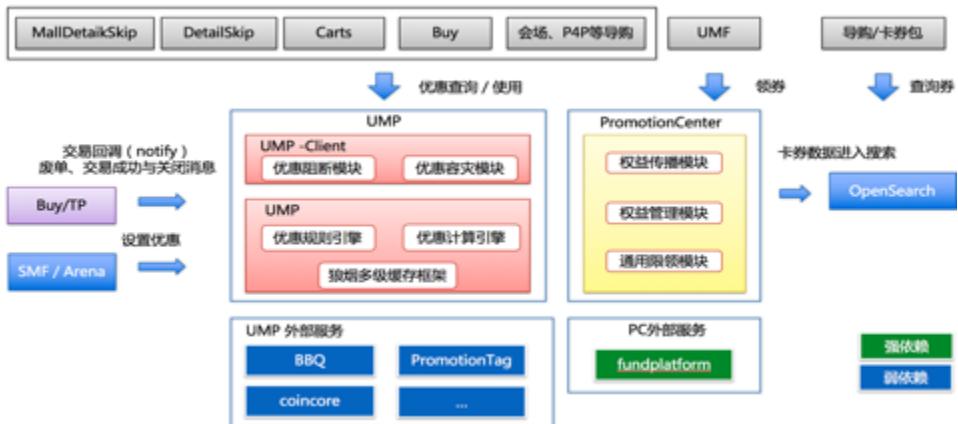
在这个架构中，业务能力、产品和场景属于平台能力，业务方的定制功能都存放于业务包中，这样的架构可以做到业务于平台分离、业务之间隔离的效果。从而使得平台的开发人员和业务的开发人员之间互相不干扰，而不同业务的开发人员之间也互不影响，这极大的提升了多团队同时协作的效率，从而提升了整体的效率。

## 4 核心链路 - 营销系统面临的业务技术挑战以及解决之道

2016 年双 11，营销平台是天猫与淘系优惠系统合并后的第一个双 11，在系统性能、稳定性、数据一致性上遇到了诸多挑战。下文主要是阐述营销平台系统在高性能、数据一致性上遇到的挑战与解决方案的实践，最后再介绍主要的玩法“双 11 购物券”从招商到交易过程中，我们的解决方案。

### 4.1 营销平台双 11 整体架构

营销平台包括 2 个交易核心系统，分别是 UMP（图中红色部分）与 PromotionCenter（图中黄色部分）。



UMP 负责所有优惠计算、优惠在导购与交易链路中的透出，包含三部分：

- (1) UMP 的核心是优惠计算。例如，折扣价是多少，这笔订单能用多少的双 11 购物券，抵扣多少金额；
- (2) 优惠规则引擎负责处理优惠与优惠之间的关系。例如，店铺优惠券能和店铺满减活动叠加使用；
- (3) 狼烟多级缓存框架负责将优惠数据根据功能、热度等因素分配到多级的缓存中，同时屏蔽了不同缓存的实现。

在 2016 年，UMP 整合了天猫优惠系统，将原天猫优惠的所有功能与数据都完整的迁移到了 UMP 中，今年双 11 和去年相比，调用链路、数据流都发生了巨大的变化。PromotionCenter 是权益平台，负责权益的传播、权益的管理。目前主要的权益包括：卡券（店铺优惠券、店铺红包、商品优惠券、天猫购物券、双 11 购物券等）、优酷会员等。

## 4.2 数据一致性实践--通用数据对账平台

双 11 对于营销平台的挑战，先从数据说起。由于营销都是在和钱打交道，钱算的不对，带来的就是损益，所以数据的一致性对于业务正确性来说，尤其重要。

### 4.2.1 数据一致性的挑战和目标

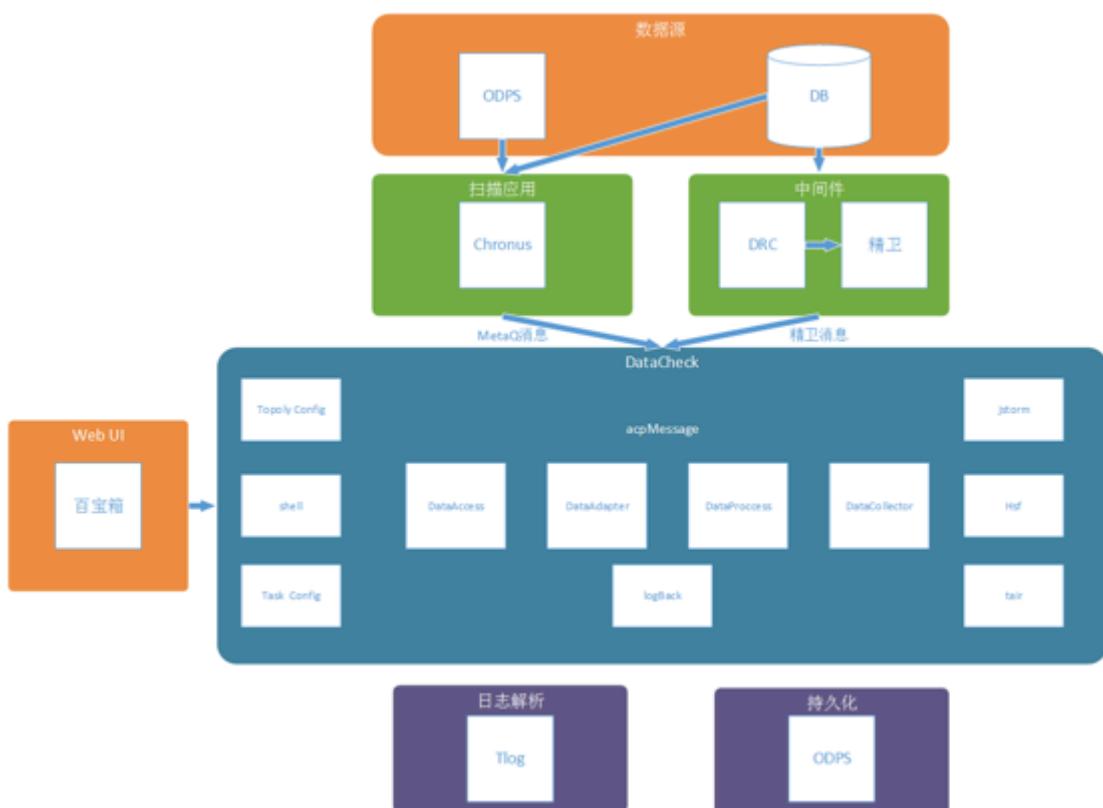
- 1、优惠数据产生与使用过程会经历多种数据源（DB、TAIR、Vsearch 等）

- 2、单元化部署结构导致数据会在多个机房中存储
- 3、优惠与外部如资金，搜索等都有关联，数据链路异常复杂
- 4、缺少主动发现和补偿机制来保障数据的最终正确，可能引发投诉与损

#### 4.2.2 我们建立的通用数据对账平台的目标

- 1、抽取底层模型，形成统一的结构
- 2、支持多种数据源，如 db 与 tair 等，自定义对比方式
- 3、支持增量与全量对账
- 4、业务可配置化，能快速上线，发现线上不一致问题，以及相应处理与报警

通用数据对账平台 ( DataCheck , 简称 DC ) 整体架构



底层基于 JStorm 实时流式计算框架作为运行的基础，上层增加了任务调度管理，数据源、对账脚本管理、监控报警管理等模块。用户可以通过实现简单的对账脚本，就可以完成数据的对账工作。

离线对账，通过 chronus 定时调度程序，扫描 DB 或者定时拉取云梯表的数据，将需要对账的内容组装成 metaq 消息，DataCheck 根据消息内容执行对应的数据对账脚本（数据对账需要预先在百宝箱中配置好），最终的执行结果

实时对账，与离线对账的差异在于触发的机制不同，实时对账通过 DB 的 DRC 消息触发，精卫通过消息触发 DataCheck 进行对账。

#### 4.2.3 与功能相类似的 BCP 的对比

	DataCheck	BCP
多数据源支持	支持	支持
动态脚本支持	支持	支持
云梯离线	支持	不支持
DB 扫表	支持	不支持
可视化后台管理	完善（基于 JStorm）	不完善
动态扩容	完善（基于 JStorm）	不完善
异地多活	完善（基于 JStorm）	不完善
资源隔离	支持	不支持

### 4.3 多级缓存框架实践 —— 狼烟多级缓存框架

在 UMP 整合了天猫优惠系统后，由于天猫卖家中，会有大卖家，如：猫超、当当网、优衣库。每到大促，热点数据成为了我们关注的问题。UMP 由于实时性的要求，是一个重度依赖缓存的系统。如何防止热点数据击穿，提升热点数据的访问效率，我们建立了一个多级缓存框架来解决这个问题 —— 狼烟。

#### 4.3.1 狼烟三级缓存结构

1、预热缓存。

在双 11 大促里我们是可以预测出一些热点数据和必定极热的卖家维度数据。这种促销级的活动在一定的周期内是禁止编辑的，在开始的周期内将预测的

热点数据提前写到本地缓存。在 16 年双 11，狼烟做到了在 3 分钟内完成 7 千多台机器、几个 G 数据的预热。具体实现有堆内、local tair ( 堆外 )。

### 2、热点缓存。

存储在应用中的活跃数据，一些无法预测买家购买行为的数据可以按照周期内 QPS 排序，保证 Top 热点常驻在零点里。我们和 sentinel 团队的子矜合作，提供了初步想法，最后由子矜完成 hotsensor，采用 hotsensor 来防止黑马热点的问题。

### 3、全量数据缓存。

一般采用 tair 实现，支持 ldb 与 mdb。

## 4.3.2 狼烟特性

### 1、统一接口

由于不同缓存有不同的实现，接口差异性较大，在狼烟里封装了统一的 API，业务代码在应用时，无须关注底层实现

### 2、流程控制

狼烟支持在多级缓存获取的流程上，做到细致的控制，如各级缓存的写操作、tair&db 一体化、db 限流等等

### 3、缓存控制

在狼烟里可以做到任意一级缓存的可拆分，可以细粒度到只访问某一级缓存（包括 db，在狼烟里 db 也是统一的一套接口，由应用提供回调）

### 4、预热缓存

狼烟采用蜻蜓进行分发需要预热的数据，采用统一的数据结构包装预热的数据，在单机生成数据（文件）后，提供数据获取接口给到蜻蜓进行分发，分发的流程如下图

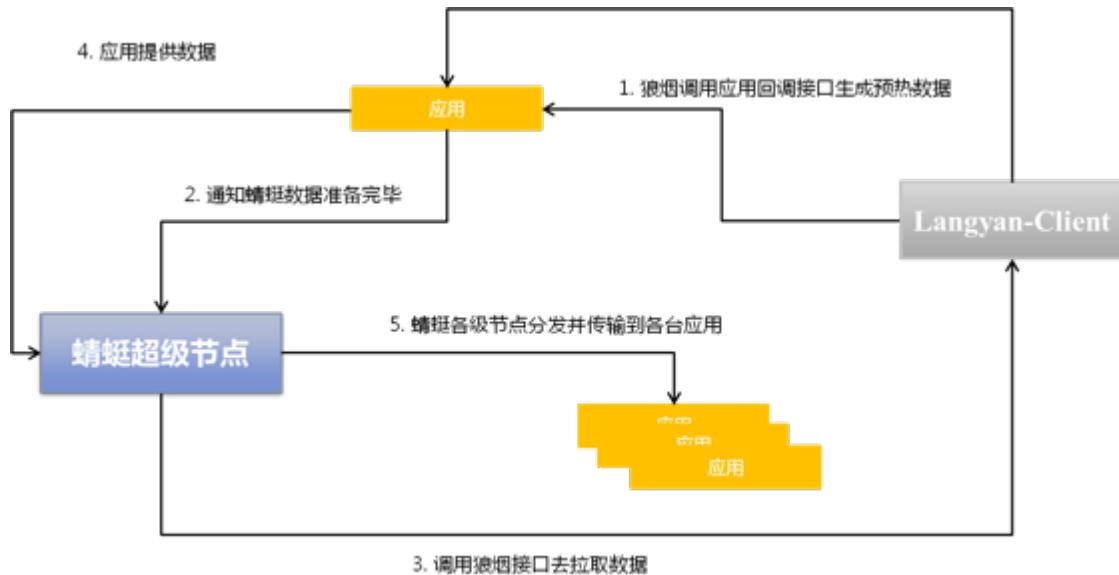
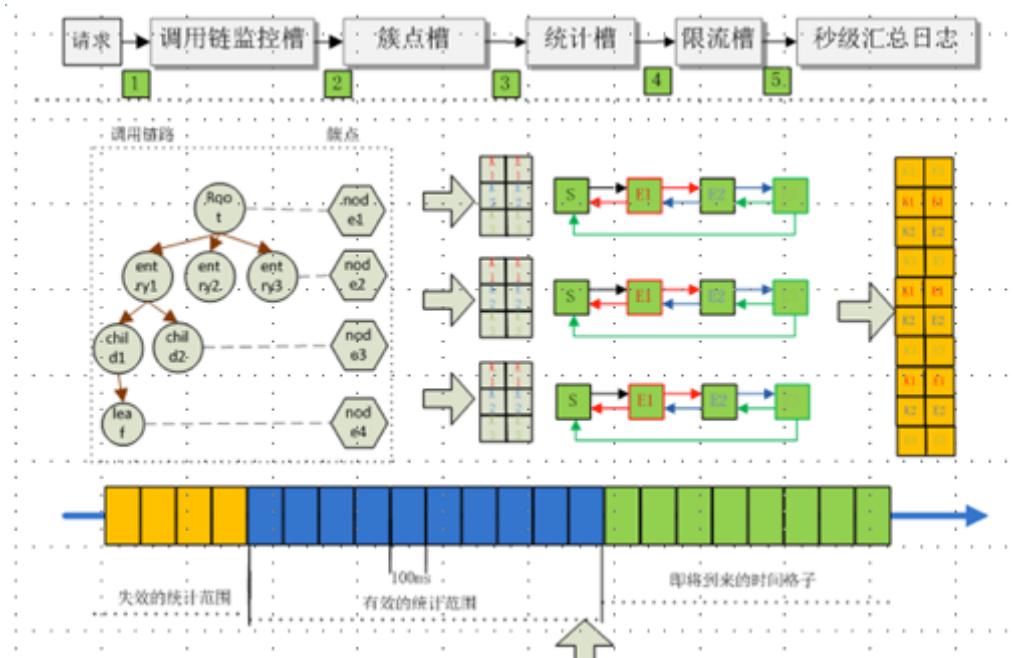


图 8 狼烟分发原理



## 5、热点缓存

**高峰期驱逐策略**：两小时内数据有效，在周期内针对数据访问的 qps 和最近访问一次时间点进行排序，排在最后面的驱逐。热点缓存依赖了 hotsensor 进行热点的计算，hotsensor 针对周期内的单位时间做了一个滑动窗口。采样窗口长度设为 1.5s 或 2h，这个采样窗口又被分割为 20 个格子。通过用一定的算法来统计这 20 个格子的平均滑动窗口累积平均值来进行统计。

计算公式：

$$CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n + 1}$$

## 6、数据处理

在涉及到多级缓存时，未命中的 key 到下一级进行查询，多 key 的情况（譬如常见的：mget、mprefixGet、mprefixGets）需要组装每次的查询结果，去识别到每一个 key。代码量比较多且重复，在 ump 里面充斥着大量的组装返回结果的代码，在应用狼烟后，清理了一大批这样的代码，用两三行去替换之前几十上百行代码，大大增加了代码可读性

### 4.3.3 狼烟在 2016 双 11 表现

#### 1、预热缓存

- ✧ 优惠活动数据，命中率为百分之十几，减少到 Tair 百分之十几的调用量。
- ✧ 卡券规则数据，命中率近乎 100%，应用基本不需要访问 Tair 获取这部分数据

#### 2、热点缓存。

0 点峰值整体命中率为 20% 左右，支持单 key 最高 40 多万的 QPS 访问，没有出现击穿缓存到 DB 的情况。、

## 5 业务中台，电商链路的基石

交易，营销，资金只是业务平台事业部的一部分功能，除此之外部门还有商品，会员，店铺，商户，电子凭证，LBS，数据，评价，规则和内容功能单元。单元之间无缝的衔接构建了阿里集团电商板块业务的全链路业务支撑技术体系，除了耳熟能详的三淘之外，还有很多新兴的业务。利用了平台提供的模块化、可视化配置的技术组件，开放服务和元数据来快速定制独特的商业形态，多个渠道的商业数据又通过 Imap 技术动态归一沉淀，为平台治理和商家多渠道铺货提供便利。

业务平台事业部承接集团“大中台，小前台”战略业务层部分，以支撑业务方快速，低成本创新的能力为目标。然而，业务中台并非部门一己之力来建设，

集团的技术团队都是业务中台能力的提供者，没有团队界限，只有“高效实现业务”的共同目标。按照一致的标准和协议注册到中台，确保模块之间的连通性，避免重复低效建设。随着业务的频繁上线，不断沉淀、打磨、升级能力，技术同学发现自己精心打造的能力被多个业务使用，真是莫大的鼓舞！步入良性循环的运营机制，业务实现的效率越来越高。

双 11 是业务平台事业部难得一次的全员大练兵，我们用实力和勇气证明了自己，然而我们却愿意退居幕后锻铸平台，支撑前端丰富多彩的繁荣生态。今天最好的成绩是明天最低的要求，我们仍然保持追求极致用户体验，性能，稳定的目标，用技术拓展业务边界，为业务方提供简单，可信赖的电商技术基础产品，高效高质量地支持前端业务快速发展和创新，这是我们的愿望，也是我们的使命！

# 5.3 千亿访问量下的开放平台技术

## 揭秘

作者：风胜

### 前言

淘宝开放平台（open.taobao.com）是阿里系统与外部系统通讯的最重要平台，每天承载百亿级的 API 调用，百亿级的消息推送，十亿级的数据同步，经历了 8 年双 11 成倍流量增长的洗礼。本文将为您揭开淘宝开放平台的高性能 API 网关、高可靠消息服务、零漏单数据同步的技术内幕。

## 1 高性能 API 网关

阿里巴巴内部的数据分布在各个独立的业务系统中，如：商品中心、交易平台、用户中心，各个独立系统间通过 HSF（High-speed Service Framework）进行数据交换。如何将这些数据安全可控的开放给外部商家和 ISV，共建繁荣电商数据生态，在这个背景下 API 网关诞生。

### 1.1 总体架构

API 网关采用管道设计模式，处理业务、安全、服务路由和调用等逻辑。为了满足双 11 高并发请求（近百万的峰值 QPS）下的应用场景，网关在架构上做了一些针对性的优化：

1. 元数据读取采用富客户端多级缓存架构，并异步刷新缓存过期数据，该架构能支持千万级 QPS 请求，并能良好的控制机房网络拥塞。

2. 同步调用受限于线程数量，而线程资源宝贵，在 API 网关这类高并发应用场景下，一定比例的 API 超时就会让所有调用的 RT 升高，异步化的引入彻底的隔离 API 之间的影响。网关在 Servlet 线程在进行完 API 调用前置校验后，使用 HSF 或 HTTP NIO client 发起远程服务调用，并结束和回收到该线程。待 HSF 或者 HTTP 请求得到响应后，以事件驱动的方式将远程调用响应结果和 API 请求上下文信息，提交到 TOP 工作线程池，由 TOP 工作线程完成后续的数据处理。最后使用 Jetty Continuation 特性唤起请求将响应结果输出给 ISV，实现请求的全异步化处理。线程模型如图 1 所示。

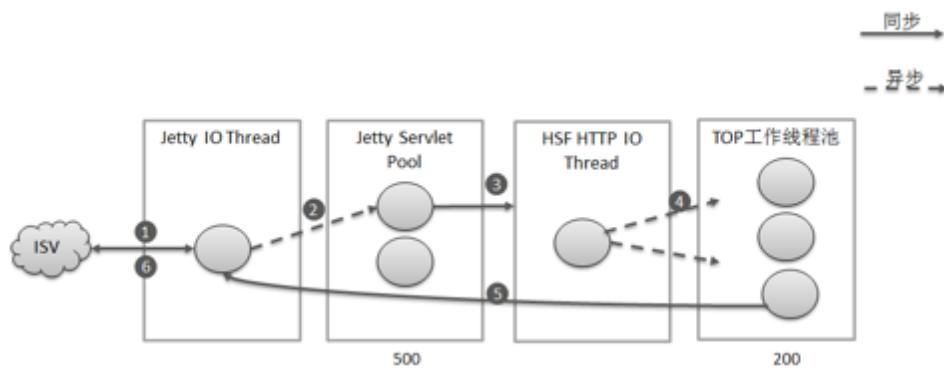


图 1：API 网关全异步化调用模型

## 1.2 多级缓存富客户端

在 API 调用链路中会依赖对元数据的获取，比如需要获取 API 的流控信息、字段等级、类目信息、APP 的密钥、IP 白名单、权限包信息，用户授权信息等等。在双 11 场景下，元数据获取 QPS 高达上千万，如何优化元数据获取的性能是 API 网关的关键点。

千万级 QPS 全部打到 DB 是不可取的，尽管 DB 有做分库分表处理，所以我们在 DB 前面加了一层分布式缓存。然而千万级 QPS 需要近百台缓存服务器，为了节约缓存服务器开销以及减少过多的网络请求，我们在分布式缓存前面加了一层 LRU 规则的本地缓存；为了防止缓存被击穿，我们在本地缓存前面加了一层 BloomFilter。一套基于漏斗模型的元数据读取架构产生（如图 2 所示）。缓存控制中心可以动态推送缓存规则，如数据是否进行缓存、缓存时长、本地缓存

大小。为了解决缓存数据过期时在极端情况下可能出现的并发请求问题，网关会容忍拿到过期的元数据（多数情况对数据时效性要求不高），并提交异步任务更新数据信息。

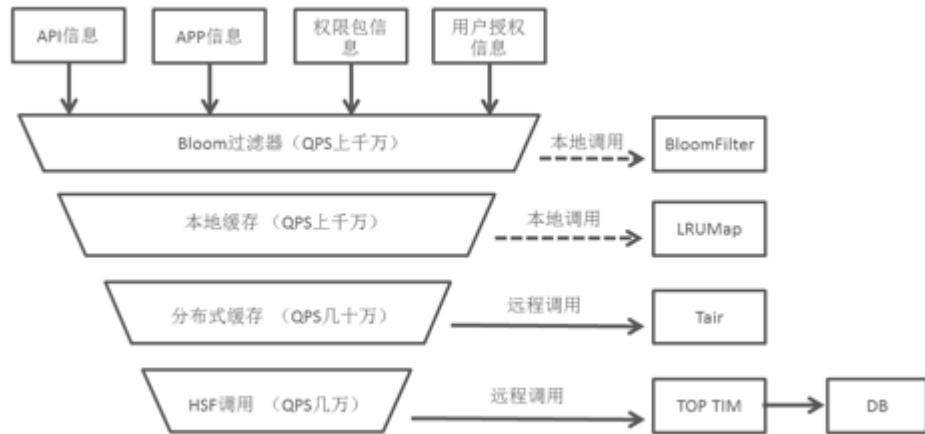


图 2：基于漏斗模型的元数据读取

### 1.3 高性能批量 API 调用

在双 11 高并发的场景下，对商家和 ISV 的系统同样是一个考验，如何提高 ISV 请求 API 的性能，降低请求 RT 和网络消耗同样是一个重要的事情。在 ISV 开发的系统中通常存在这样的逻辑单元，需要调用多个 API 才能完成某项业务（如图 3），在这种串行调用模式下 RT 较长同时多次调用发送较多重复的报文导致网络消耗过多，在弱网环境下表现更加明显。

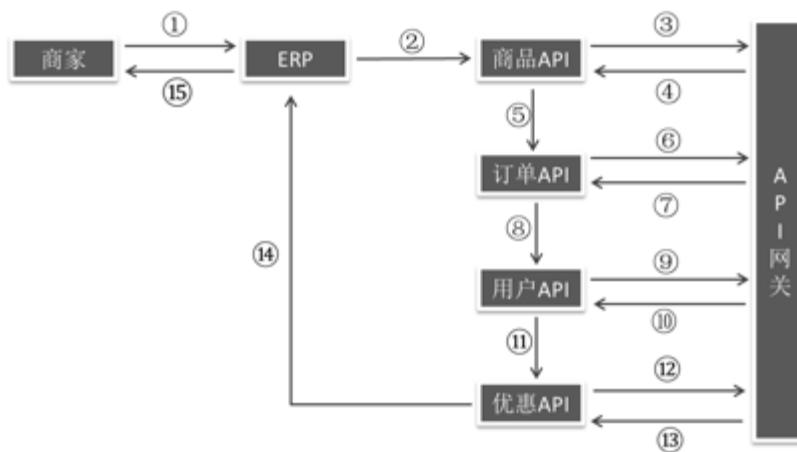


图 3：串行 API 调用处理流程

API 网关提供批量 API 调用模式（如图 4 所示）缓解 ISV 在调用 RT 过高和网络消耗上的痛点。ISV 发起的批量请求会在 TOP SDK 进行合并，并发送到指定的网关；网关接收到请求后在单线程模式下进行公共逻辑计算，计算通过后将调用安装 API 维度拆分，并分别发起异步化远程调用，至此该线程结束并被回收；每个子 API 的远程请求结果返回时会拿到一个线程进行私有逻辑处理，处理结束时会将处理结果缓存并将完成计数器加一；最后完成处理的线程，会将结果进行排序合并和输出。

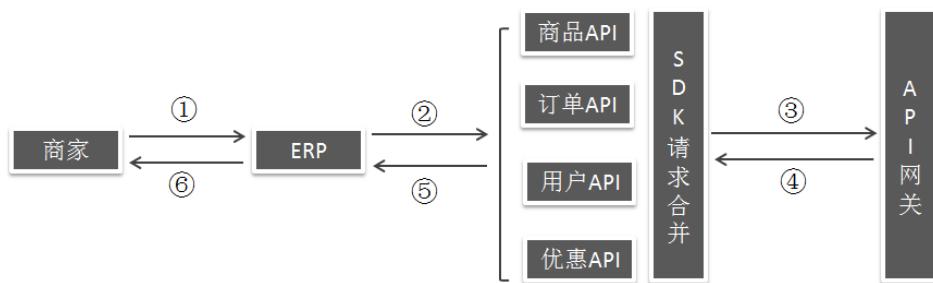


图 4：批量 API 调用处理流程

## 1.4 多维度流量控制

TOP API 网关暴露在互联网环境，日调用量达几百亿。特别是在双 11 场景中，API 调用基数大、调用者众多以及各个 API 的服务能力不一致，为了保证各个 API 能够稳定提供服务，不会被暴涨的请求流量击垮，那么多维度流量控制是 API 网关的一个重要环节。API 网关提供一系列通用的流量控制规则，如 API 每秒流控、API 单日调用量控制、APPKEY 单日调用量控制等。

在双 11 场景中，也会有一些特殊的流量控制场景，比如单个 API 提供的能力有限，例如只能提供 20 万 QPS 的能力而实际的调用需求可能会有 40 万 QPS。在这种场景下怎么去做好流量分配，保证核心业务调用不被限流。TOP API 网关提供了流量分组的策略，比如我们可以把 20 万 QPS 的能力分为 3 个组别，并可以动态去配置和调整每个组别的比例，如：分组 1 占比 50%、如分组 2 占

比 40%、分组 3 占比 10%。我们将核心重要的调用放到分组 1，将实时性要求高的调用放到分组 2，将一些实时性要求不高的调用放到分组 3。通过该模式我们能够让一些核心或者实时性要求高的调用能够较高概率通过流量限制获取到相应数据。同时 TOP API 网关是一个插件化的网关，我们可以编写流控插件并动态部署到网关，在流控插件中我们可以获取到调用上下文信息，通过 Groovy 脚本或简单表达式编写自定义流控规则，以满足双 11 场景中丰富的流控场景。

使用集群流控还是单机流控？单机流控的优势是系统开销较小，但是存在如下短板：

- 1) 集群单机流量分配不均。
- 2) 单日流控计数器在某台服务器挂掉或者重启时比较难处理。
- 3) API QPS 限制小于网关集群机器数量时，单机流控无法配置。基于这些问题，API 网关最开始统一使用集群流控方案，但在双 11 前压测中发现如下一些问题：
  - 4) 单 KEY 热点问题，当单 KEY QPS 超过几十万时，单台缓存服务器 RT 明显增加。
  - 5) 缓存集群 QPS 达到数百万时，服务器投入较高。

针对第一个问题的解法是，将缓存 KEY 进行分片可将请求离散多台缓存服务器。针对第二个问题，API 网关采取了单机+集群流控相结合的解决方案，对于高 QPS API 流控采取单机流控方案，服务端使用 Google ConcurrentLinkedHashMap 缓存计数器，在并发安全的前提下保持了较高的性能，同时能做到 LRU 策略淘汰过期数据。

## 2 高可靠消息服务

有了 API 网关，服务商可以很方便获取淘系数据，但是如何实时获取数据呢？轮询！数据的实时性依赖于应用轮询间隔时间，这种模式，API 调用效率低且浪费机器资源。基于这样的场景，开放平台推出了消息服务技术，提供一个实时的、可靠的、异步双向数据交换通道，大大提高 API 调用效率。目前，整个系统日均处理百亿级消息，可支撑百万级瞬时流量，如丝般顺滑。

## 2.1 总体架构

消息系统从部署上分为三个子系统，路由系统、存储系统以及推送系统。消息数据先存储再推送，保证每条消息至少推送一次。写入与推送分离，发送方不同步等待接收方应答，客户端的任何异常不会影响发送方系统的稳定性。系统模块交互如图 5 所示。

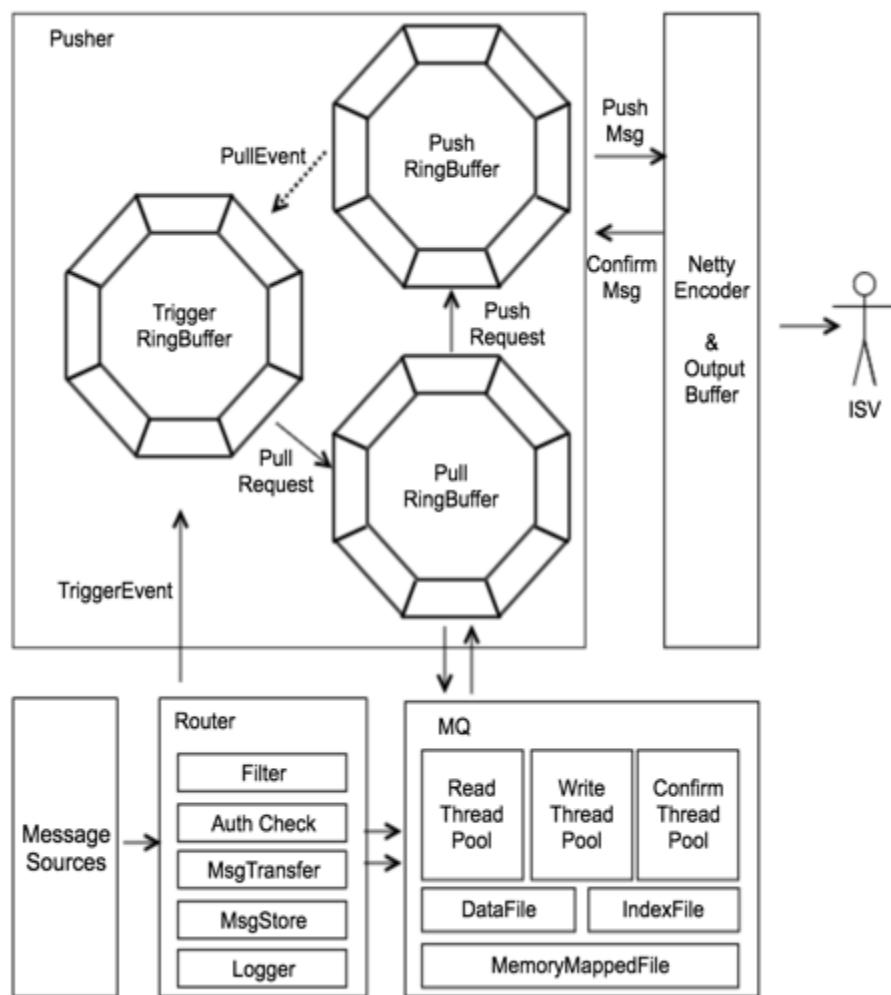


图 5：消息服务总体架构

路由系统，各个处理模块管道化，扩展性强。系统监听主站的交易、商品、物流等变更事件，针对不同业务进行消息过滤、鉴权、转换、存储、日志打点等。系统运行过程记录各个消息的处理状况，通过日志采集器输出给 JStorm 分析集群处理并记录消息轨迹，做到每条消息有迹可循。

存储系统主要用于削峰填谷，基于 BitCask 存储结构和内存映射文件，磁盘完全顺序写入，速度极佳。数据读取基于 FileRegion 零拷贝技术，减少内存拷贝消耗，数据读取速度极快。存储系统部署在多个机房，有一定容灾能力。

推送系统，基于 Disptor 构建事件驱动模型，使用 Netty 作为网络层框架，构建海量连接模型，根据连接吞吐量智能控制流量，降低慢连接对系统的压力；使用 WebSocket 构建长连接通道，延时更低；使用对象池技术，有效降低系统 GC 频率；从消息的触发，到拉取，到发送，到确认，整个过程完全异步，性能极佳。

## 2.2 选择推送还是拉取

在消息系统中，一般有两种消费模式：服务端推送和客户端拉取。本系统主要面向公网的服务器，采用推送模式，有如下优点：

1. 实时性高。从消息的产生到推送，总体平均延时 100 毫秒，最大不超过 200 毫秒。
2. 服务器压力小。相比于拉取模式，每次推送都有数据，避免空轮询消耗资源。
3. 使用简便。使用拉取模式，客户端需要维护消费队列的位置，以及处理多客户端同时消费的并发问题。而在推送模式中，这些事情全部由服务器完成，客户端仅需要启动 SDK 监听消息即可，几乎没有使用门槛。

当然，系统也支持客户端拉取，推送系统会将客户端的拉取请求转换为推送请求，直接返回。推送服务器会据此请求推送相应数据到客户端。即拉取异步化，如果客户端没有新产生的数据，不会返回任何数据，减少客户端的网络消耗。

## 2.3 如何保证低延时推送

在采用推送模式的分布式消息系统中，最核心的指标之一就是推送延时。各个长连接位于不同的推送机器上，那么当消息产生时，该连接所在的机器如何快速感知这个事件？

在本系统中，所有推送机器彼此连接（如图 6 所示），构成一个通知网，其中任意一台机器感知到消息产生事件后，会迅速通知此消息归属的长连接的推送机器，进而将数据快速推送给客户端。而路由系统每收到一条消息，都会通知下游推送系统。上下游系统协调一致，确保消息一触即达。

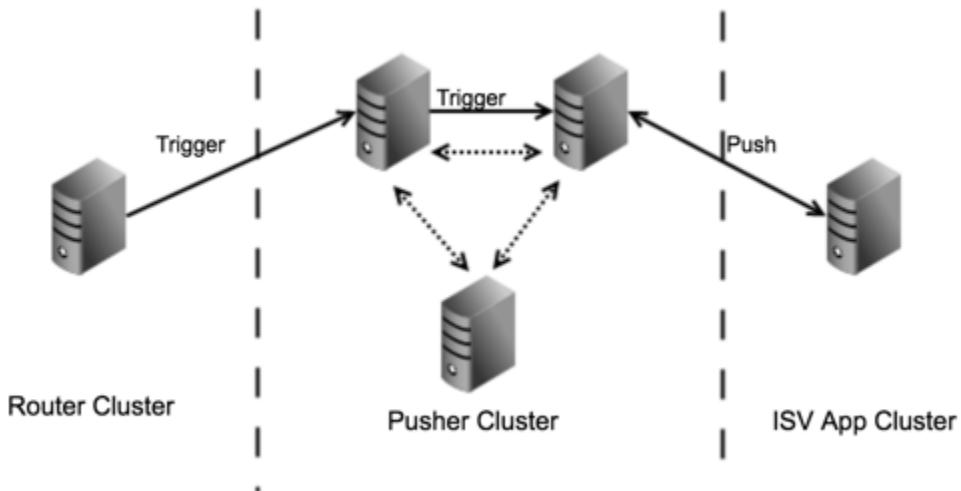


图 6：消息事件触发流程

## 2.4 如何快速确认消息

评估消息系统另外一个核心指标是消息丢失问题。由于面向广大开发者，因此系统必须兼顾各种各样的网络环境问题，开发者能力问题等。为了保证不丢任何一条消息，针对每条推送的消息，都会开启一个事务，从推送开始，到确认结束，如果超时未确认就会重发这条消息，这就是消息确认。

由于公网环境复杂，消息超时时间注定不能太短，如果是内网环境，5 秒足矣，消息事务在内存就能完成。然后在公网环境中，5 秒远远不够，因此需要持久化消息事务。在推送量不大的时候，可以使用数据库记录每条消息的发送记录，使用起来也简单方便。但是当每秒推送量在百万级的时候，使用数据库记录的方式就显得捉襟见肘，即便是分库分表也难以承受如此大的流量。

对于消息推送事务数据，有一个明显特征，99%的数据会在几秒内读写各一次，两次操作完成这条数据就失去了意义。在这种场景，使用数据库本身就不合

理，就像是在数据库中插入一条几乎不会去读的数据。这样没意义的数据放在数据库中，不仅资源浪费，也造成数据库成为系统瓶颈。

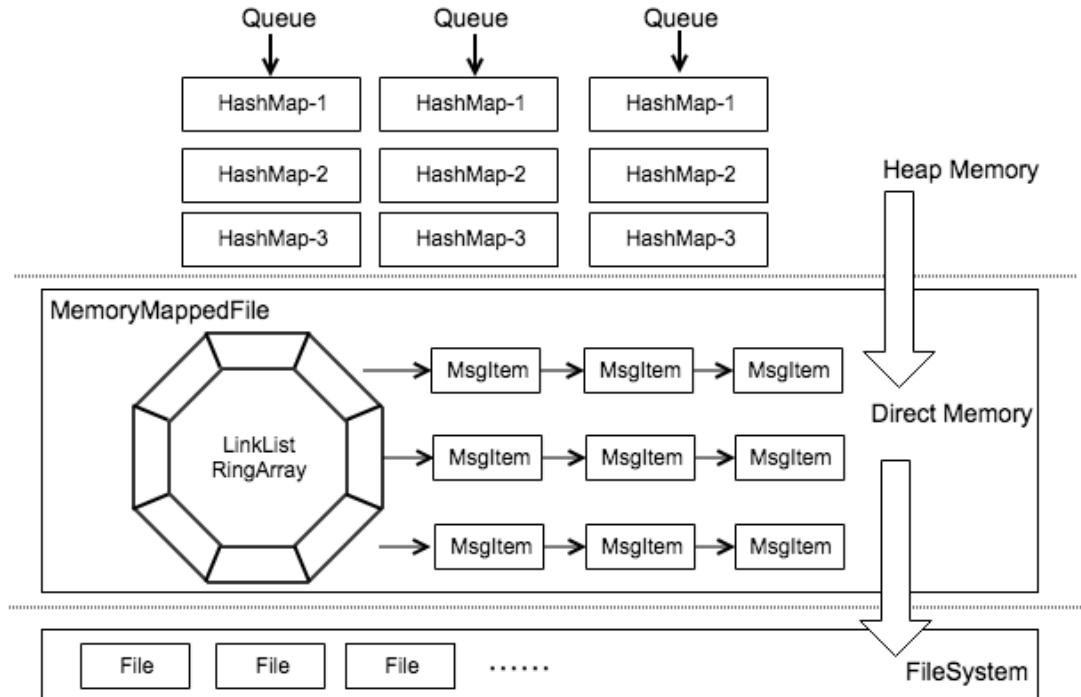


图 7：消息确认流程

如图 7 所示，针对这种场景，本系统在存储子系统使用 **HeapMemory**、**DirectMemory**、**FileSystem** 三级存储结构。为了保护存储系统内存使用情况，**HeapMemory** 存储最近 10 秒发送记录，其余的数据会异步写入内存映射文件中，并写入磁盘。**HeapMemory** 基于时间维度划分成三个 **HashMap**，随着时钟滴答可无锁切换，**DirectMemory** 基于消息队列和时间维度划分成多个链表，形成链表环，最新数据写入指针头链表，末端指针指向的是已经超时的事务所在链表。这里，基于消息队列维护，可以有效隔离各个队列之间的影响；基于时间分片不仅能控制链表长度，也便于扫描超时的事务。

在这种模式下，95%的消息事务会在 **HeapMemory** 内完成，5%的消息会在 **DirectMemory** 完成，极少的消息会涉及磁盘读写，绝大部分消息事务均在内存完成，节省大量服务器资源。

## 3 零漏单数据同步

我们已经有了 API 网关以及可靠的消息服务，但是对外提供服务时，用户在订单数据获取中常常因为经验不足和代码缺陷导致延迟和漏单的现象，于是我们对外提供数据同步的服务。

传统的数据同步技术一般是基于数据库的主备复制完成的。在简单的业务场景下这种方法是可行的，并且已经很多数据库都自带了同步工具。但是在业务复杂度较高或者数据是对外同步的场景下，传统的数据同步工具就很难满足灵活性、安全性的要求了，基于数据的同步技术无法契合复杂的业务场景。

双 11 场景下，数据同步的流量是平常的数十倍，在峰值期间是百倍，而数据同步机器资源不可能逐年成倍增加。保证数据同步写入的平稳的关键在于流量调控及变更合并。

### 3.1 分布式数据一致性保证

在数据同步服务中，我们使用了消息 + 对账任务双重保障机制，消息保障数据同步的实时性，对账任务保障数据同步一致性。以订单数据同步为例，订单在创建及变更过程中都会产生该订单的消息，消息中夹带着订单号。接受到该消息后，对短时间内同一订单的消息做合并，数据同步客户端会拿消息中的订单号请求订单详情，然后写入 DB。消息处理过程保证了订单在创建或者发生了任意变更之后都能在极短的延迟下更新到用户的 DB 中。

对账任务调度体系会同步运行。初始化时每个用户都会生成一个或同步任务，每个任务具有自己的唯一 ID。数据同步客户端存活时每 30 秒发出一次心跳数据，针对同一分组任务的机器的心跳信息将会进行汇总排序，排序结果一般使用 IP 顺序。每台客户端在获取需执行的同步任务列表时，将会根据自身机器在存活机器总和 x 中的顺序 y，取得任务  $ID \% x = y - 1$  的任务列表作为当前客户端的执行任务。执行同步任务时，会从订单中心取出在过去一段时间内发生过变更的订单列表及变更时间，并与用户 DB 中的订单进行一一对比，如果发现订单不存在或者与存储的订单变更时间不一致，则对 DB 中的数据进行更新。

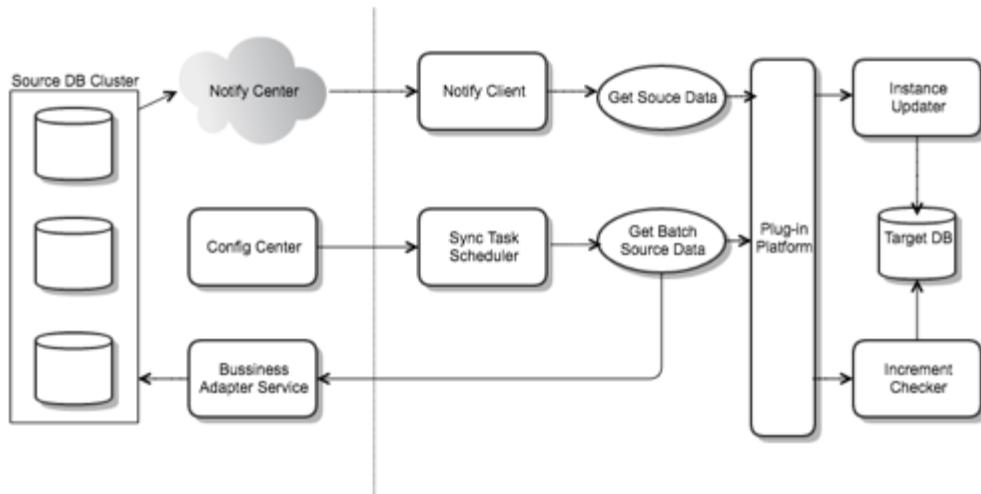


图 8：数据同步服务架构

### 3.2 资源动态调配与隔离

在双 11 场景下如何保证数据同步的高可用，资源调配是重点。最先面临的问题是，如果每台机器都是幂等的对应全体用户，那么光是这些用户身后的 DB 连接数消耗就是很大问题；其次，在淘宝的生态下，卖家用户存在热点，一个热点卖家的订单量可能会是一个普通卖家的数万倍，如果用户之间直接共享机器资源，那么大流量用户将会占用几乎全部的机器资源，小流量用户的数据同步实效会受到很大的影响。

为了解决以上问题，我们引入了分组隔离。数据同步机器自身是一个超大集群，在此之上，我们将机器和用户进行了逻辑集群的划分，同一逻辑集群的机器只服务同一个逻辑集群的用户。在划分逻辑集群时，我们将热点用户从用户池中取出，划分到一批热点用户专属集群中。分组隔离解决了 DB 连接数的问题，在此场景下固定的用户只会有固定的一批机器为他服务，只需要对这批机器分配连接数即可，而另一个好处是，我们可以进行指定逻辑集群的资源倾斜保障大促场景下重点用户的数据同步体验。

数据同步服务大集群的机器来源于三个机房，在划分逻辑集群时，每个逻辑分组集群都是至少由两个以上机房的机器组成，在单个机房宕机的场景下，逻辑集群还会有存活机器，此时消息和任务都会向存活的机器列表进行重新分配，保证该逻辑集群所服务的用户不受影响。在机器发生宕机或者单个逻辑集群的压力增大时，调度程序将会检测到这一情况并且对冗余及空闲机器再次进行逻辑

集群划分，以保证数据同步的正常运行。在集群压力降低或宕机机器恢复一段时间后，调度程序会自动将二次划分的机器回收，或用于其他压力较大的集群。

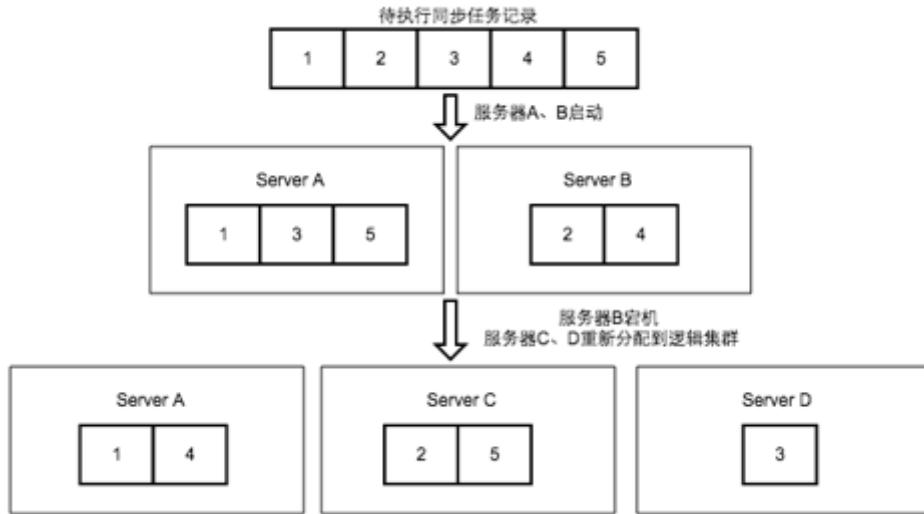


图 9：机器宕机与重分配

### 3.3 通用数据存储模型

订单上存储的数据结构随着业务的发展也在频繁的发生的变化，进行订单数据的同步，需要在上游结构发生变化时，避免对数据同步服务产生影响，同时兼顾用户的读取需求。对此我们设计了应对结构易变数据的大字段存储模型。在订单数据的存储模型中，我们将订单号、卖家昵称、更新时间等需要被当做查询/索引条件的字段抽出独立字段存储，将整个的订单数据结构当成 json 串存入一个大字段中。

名称	类型	是否索引	说明
tid	NUMBER	Y	交易ID
status	VARCHAR	Y	交易状态
type	VARCHAR	Y	交易类型
seller_nick	VARCHAR	Y	卖家昵称
buyer_nick	VARCHAR		买家昵称
created	DATETIME	Y	交易创建时间
modified	DATETIME	Y	交易修改时间
jdp_created	DATETIME	Y	数据同步的创建时间
jdp_modified	DATETIME	Y	数据同步的修改时间
jdp_hashcode	VARCHAR		用作数据校验的字段
jdp_response	MEDIUMTEXT		复杂的交易结构

图 10 订单同步数据存储结构

这样的好处是通过大字段存储做到对上游业务的变化无感知，同时，为了在进行增量数据同步时避免对大字段中的订单详情进行对比，在进行数据同步写入的同时将当前数据的 hashcode 记录存储，这样就将订单数据对比转换成了 hashcode 与 modified 时间对比，提高了更新效率。

### 3.4 如何降低数据写入开销

在双 11 场景下，数据同步的瓶颈一般不在淘宝内部服务，而在外部用户的 DB 性能上。数据同步是以消息的方式保证实时性。在处理非创建消息的时候，我们会使用直接 update + modified 时间判断的更新方式，替换传统的先 select 进行判断之后再进行 update 的做法。这一优化降低了 90% 的 DB 访问量。

传统写法：

```
SELECT * FROM jdp_tb_trade WHERE tid = #tid#;
```

```
UPDATE jdp_tb_trade SET jdp_response = #jdpResponse#, jdp_modified = now() WHERE tid = #tid#
```

优化写法：

```
UPDATE jdp_tb_trade SET jdp_response = #jdpResponse#, jdp_modified = now() WHERE tid = #tid# AND modified < #modified#
```

订单数据存在明显的时间段分布不均的现象，在白天订单成交量较高，对DB的访问量增大，此时不适合做频繁的删除。采用逻辑删除的方式批量更新失效数据，在晚上零点后交易低峰的时候再批量对数据错峰删除，可以有效提升数据同步体验。

## 5.4 智能供应链

作者：昏河

### 前言

在面对天猫双 11 当天 1207 亿商品交易额( GMV )和 6.57 亿物流订单时，供应链作为电子商务企业运作的支柱，是如何有效地组织、协调和优化这个复杂且动态变化的供需网络，从而可以高效地支持 2016 年猫全球购物狂欢节呢？

### 1 传统电子商务企业供应链

在 2015 年前，天猫供应链关注的重点是借助天猫电子商务交易平台，实现供应链交易过程的全程电子化。在这个阶段，天猫利用其供应链将上下游的企业和客户等进行全面的协同和交易撮合，对平台相关的信息流、资金流和物流进行监控和管理。在这种传统的电子商务供应链的支持下，天猫完成了多次双 11，重点打造了其交易系统、库存系统、商品系统和会员系统等一系列传统电子商务支柱系统。并通过一次又一次的双 11 极大地提升了这些系统的稳定性和可靠性，在这些系统的支持下天猫供应链具备了处理频繁变化需求和复杂业务形态的动态供应链雏形，也具备了信息化、数字化、集成化、可视化和自动化等技术特征，但仍然属于传统电子商务供应链，在供应链的控制和管理的智能化水平上还存在着较大的提升空间。

### 2 天猫智能供应链

在 2015 年后，天猫双 11 业务的复杂性和规模日益增加，对其供应链管理的效率和智能化水平也提出了新的要求。2015 年天猫双 11 全球狂欢节涉及了超过 1.6 万个国际品牌，其中 33% 的总买家向国际品牌或商家购物，买家及卖家来自全球 232 个国家及地区。2016 年参与双 11 狂欢的国家和地区共计 235

个，百万店铺线下和线上打通，真正实现了全球狂欢，在背后依赖的正是天猫智能供应链。

在这段时间，阿里巴巴天猫技术团队开始在多个垂直行业的供应链领域进行深耕细致，并引入了运筹优化、机器学习、自然语言处理和图像识别等算法，升级了天猫供应链的控制和管理能力。目前天猫供应链已具备了需求预测、供应链优化、供应链仿真、智能选品和定价等一系列智能服务。

## 2.1 需求预测

天猫作为拥有数亿活跃用户的生态系统服务商，及时掌握顾客的需求变化，对顾客需求做出快速而准确的预测是天猫供应链建设的基础。需求预测是整个市场需求波动的晴雨表，也是供应链运作的源头，需求预测的准确度和灵敏性直接影响到电商的库存策略，物流安排、对顾客订单的交付率准时率、以及个性化推荐策略等。

天猫供应链平台的需求预测包括两个部分，分别为商品需求预测和用户偏好需求预测。商品需求预测是指商品在时间和空间两个维度上的需求预测，关注不同区域不同时间颗粒度上对 SKU 的需求预测，它与采购策略、库存策略和物流策略紧密相关，其使用场景侧重在商品采购批量的确定、库存空间和时间维度的分配、物流资源的订购和安排等环节。高质量的商品需求预测有助于减少商品缺货给商家和平台带来的巨大损失、有助于减少闲置库存产生的资金和人力浪费、更有助于提前准备和安排物流资源以提高物流服务水平。用户偏好需求预测是指对平台上每个消费者个人购买偏好的预测，主要是基于用户在天猫和淘宝等平台上的购买、点击、搜索、收藏和评价等行为信息，预测消费者在当前阶段对商品类目、价格带和规格等关键属性偏好的需求预测，其用途主要是进行商品个性化推荐和展示。

对于商品需求预测，当前天猫供应链平台包括常规需求预测、新品需求预测和包含双 11 期间的活动商品需求预测。当前的预测算法考虑了历史成交、活动数据、节假日和大促日信息、以及商品特性等数据，系统对数千维特征进行构建，并将特征切分为交易、流量、活动、类目和属性等特征集合，建立了多层次的预测模型，在模型中融合了梯度渐近树算法、随机森林算法、以及支持向量机等数十种机器学习算法。需求预测主体算法可以支持商品与算法进行自适应匹配，根据不同的商品信息在每个决策时空内进行商品与不同的融合模型进行快速匹配，

以进行更高精度的预测。同时通过预测数据和真实数据的监控和反馈，算法能够持续地优化预测模型，并基于商品需求的预测质量持续调整模型参数和模型融合方案。

用户偏好需求预测是基于阿里体系内沉淀的大数据，通过深度学习和机器学习等算法，构建了用户画像、商品画像、类目画像和品牌画像等，并建立了用户类目偏好预测模型、用户品牌偏好预测模型、用户商品偏好预测模型、商品间关联预测模型、以及类目关联等数十种预测模型。这些模型在天猫和淘宝平台上众多的购物场景下为用户提供了个性化的商品推荐。在 2016 天猫双 11 期间，天猫多个业务的推荐版块就是采用了这种用户偏好需求预测的算法和模型，为用户提供更精准的个性化购物体验。

## 2.2 供应链优化

传统电商的供应链专注于将其上下游的企业和客户等进行协同和交易撮合，对平台相关的信息流、资金流和物流进行监控和管理，但在供应链管理的智能化和精细化等方面还存在不足，导致了其供应链运作效率和成本控制方面存在着较大的提升空间。

天猫供应链平台以运筹优化算法和人工智能算法作为驱动，针对供应链运作中的补货、库间调拨、以及健康库存等关键业务环节进行了优化。同时针对天猫平台在日常和双 11 期间的业务特点，考虑了不同商品在各区域和时段的需求预测和波动情况、商品的区域营销策略、物流设施的空间拓扑结构、商品库存的历史在架和周转状况等因素，构建了相关业务的数学模型，建立了自动补货、自动调拨和健康库存等优化管理模块，以优化库存资源分配、提高商品在各区域的周转率和在架率。以双 11 前进行的集货仓全国调拨为例，自动调拨算法在综合考虑各区域需求预测的前提下进行了有限库存的全国调拨和库存分配，平衡各区域的供给和需求，优化双 11 期间总体库存周转和在架率。

在营销端，天猫供应链平台也采用了运筹学优化算法，针对商品参与活动的计划和调度问题进行了优化。在综合考虑了类目分配、活动折扣、历史销量、以及库存等业务信息和规则后，建立了混合整数规划的数学模型，应用了分支定界等优化算法高效解决了该问题。在 2016 天猫双 11 期间，天猫部分活动版块就是采用了该数学模型和优化算法对商品在双 11 期间不同活动时间段进行优化调度，以提供消费者更好的购物体验。

## 2.3 供应链仿真

由于电商企业的供应链是一个高度动态，高度变化的复杂系统，其中任何一个环节的行为在很大的概率上依赖于其他环节的表现，仅仅通过解析方法和相关模型还不足以全面地分析和描述电商企业的真实供应链。供应链仿真技术是分析和优化这个复杂系统的重要工具，它不用搭建全景的实体模型，只用实体模型的一部分成本来模拟真实系统，通过建立虚拟模型以测试针对真实供应链系统的各种想法和假设条件，因而可以节省资金和时间投入，更重要的是供应链仿真技术极大地降低了电商企业的决策风险。

天猫智能供应链平台融入了针对电商供应链业务的仿真技术，支持对不同业务线多个业务场景进行仿真分析，将其供应链作为一个整体系统来进行研究和分析。通过模拟每个区域的客户下单、采购补货、预约入库、调拨入库、以及订单送达等多个业务环节，在不同决策场景下对这些环节进行一体化的仿真分析，以确定采购补货和仓间调拨等业务决策的合理性。同时天猫供应链仿真技术与平台上的优化技术紧密结合在一起，一方面通过仿真技术对运筹学获取的供应链运作优化结果予以验证，另一方面，供应链仿真得出的仿真结果可以作为运筹优化的初始解或中间解，进而辅助优化模块更有效地逼近供应链运作决策的最优解。为有效地应对本次双 11，天猫部分自营业务和类自营业务需要调整其补货和调拨业务的关键逻辑和参数，就是通过了天猫供应链仿真技术予以支持和验证，在降低决策成本的前提下提高了供应链业务决策的准确性。

## 2.4 智能选品和定价

品类分析和挖掘是平台和商家生存的关键，也是任何一家做电商的平台需要具备的能力。天猫智能供应链平台采用了深度学习、文本挖掘、图像处理、以及自然语言处理等技术，提供了智能化选品、商品聚类和智能定价的服务能力。

对于新品的选择，供应链平台通过深度学习算法，对不同渠道商品的相关文本和图片进行分析和挖掘，建立了商品本体库、商品特征库、以及相关的商品比对模型，为自营和类自营店铺提供了智能化的新品挖掘方案，进而丰富平台和店铺商品线的深度和宽度。对于既有商品的聚类和汰换，供应链平台将淘系中积累的大数据与天猫业务沉淀下的行业专业知识进行了融合，并通过机器学习算法

和自然语言处理算法建立了季节性预测模型、潜力预测模型、复购周期预测模型和舆情分析模型，进而对不同行业的商品进行聚类和汰换。此外，基于不同行业下商品比对模型和商品聚类，供应链平台应用了机器学习算法和运筹学方法建立了不同商品聚类下的商品价格模型以指导商品定价。这些模型一方面支持了店铺日常的品类规划业务，另一方面也为双 11 大促进行了选品支持。

## 总结

电子商务平台的优势之一是在于能够随时随地、持续大量地收集数据，为电商业务提供及时和可视化的供应链数据，这些高价值的数据为电商企业供应链的智能化提供了土壤。天猫智能供应链平台依托大数据和云计算，覆盖了零售平台从选品、采购、补货、调拨、定价和营销等各个供应链环节，通过运筹学、机器学习、深度学习、文本和图像识别等算法和技术，从整体上优化了供应链运作的效率和成本，通过对跨领域数据和算法的融合，产生了乘法效应，为各业务提供了全供应链的智能解决方案，最大化了其供应链的商业价值。

# 5.5 菜鸟双 11 “十亿级包裹” 之战

作者：兰博

## 前言

每年的双 11 都在刷新物流的世界奇迹，但由于大数据和协同，每次都将看似不可能完成的任务加速完成。以 2013 年-2016 年的一组数据为例，从签收时间看，2013 年双 11 包裹签收过 1 亿用了 9 天，2014 年用了 6 天，到 2015 年提速到了 4 天，今年则进一步提速只用 3.5 天。

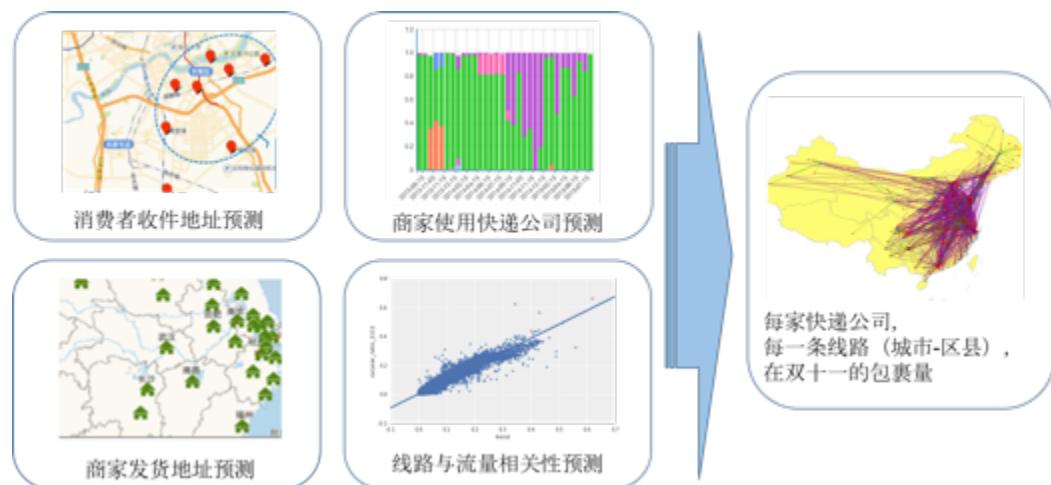
菜鸟作为一家平台型公司，坚持不拥有一辆车、一个快递员，希望通过数据和技术，建设一个社会化协同的物流和供应链公司。在这方面，菜鸟这几年不断探索，在物流全链路上做了大量的尝试和突破，比如通过海量数据精准的预测大促包裹量及流向、通过供应链预测计划合理入库及分仓铺货、以及做到“单未下、货先行”的货品下沉提前打包等。

本文将主要从包裹预测、供应链入库、订单下沉、订单路由调度、电子面单及智能分单，以及在末端小件员方面的一些技术探索实践，通过数据和技术的力量，捍卫这场十亿级包裹的双 11 之战。

## 1 双 11 从包裹预测开始

2015 年双十一当天的物流订单量历史性地达到了 4.67 亿，相比于 2014 年足足增长了 65%。到了 2016 年整个双十一期间的包裹量又将面临爆发性地增长，几近达到 10 亿级的规模。所以，如何采用大数据预测技术从宏观层面来估计 2016 年双十一的包裹总量和并利用微观数据信息得到所有包裹在双十一期间的流量和流向，从而为快递公司提供决策支持，提前规划平稳度过双十一高峰是一个至关重要的技术命题。

包裹总量预测根据今年前台预估的双十一 GMV 总量 整合历史双十一 GMV 数据和物流订单数据，历史上所有大促期间和日常笔单价变化趋势，分析各项宏观因素，并监控集团预售预热期间的销售数据，引入可能存在的不确定性，鲁棒地预测今年包裹总量的增长幅度。最终我们预估得到双十一当日包裹总量 6.8 亿单的数据，达到 97% 的准确率。



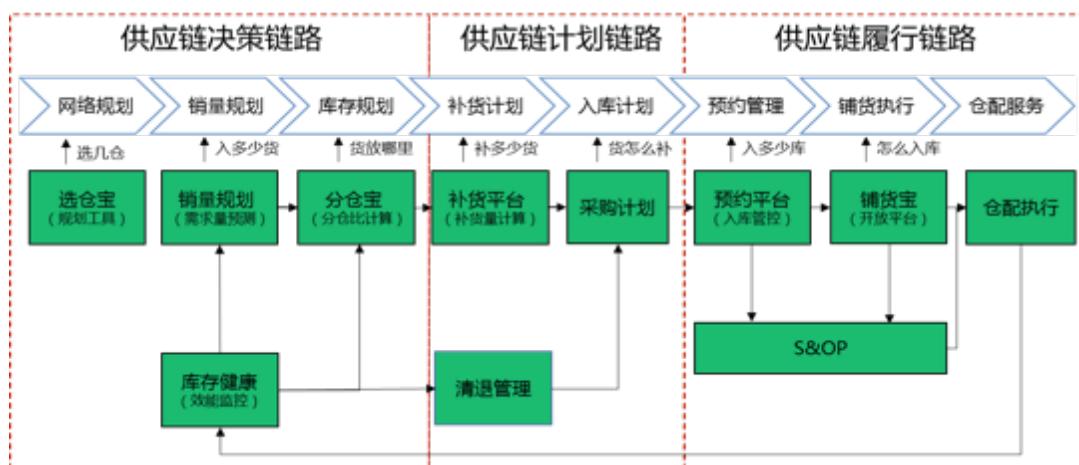
包裹流量流向预测预测的 6.8 亿包裹精确地拆解到不同的快递公司，拆解到全国各个城市，各条线路，并且基于时效预测给出各条线路每天的发货量以及未来双十一期间的到货量可以帮助快递公司提前准备运力，调度人员，精准布局，合理地优化资源从而能够从容应对即将到来的物流高峰。

在这套大数据预测模型中，我们从六百万商家过去 5 年的销售数据中挖掘有效的信息，预测双十一期间商家可能使用的快递公司和对应的发货仓库和城市；根据消费者的历史行为，挖掘预测消费者的双十一期间的收货地址；根据不同快递公司在双十一期间的参与度和能力，预测他们双十一的市场份额。最终，整合所有相关信息建立回归预测模型，我们能够在双十一的前 3 个月就以 80% 的准确率预测得到每家快递公司在全国每一条线路上的包裹量，真正帮助快递公司做到了兵马未动、粮草先行。

## 2 供应链入库计划

在双11的准备阶段，除了如何帮助快递公司做好数据预测，达到提前扩充运力、人员，如何协调数千商家的上亿商品有序进入到数百计的仓库内，也一直是历届双11的头等难题。仓库爆仓、货物集中到仓导致的准备能力难以匹配需求、货物到达不能按时入库，诸如此类问题层出不穷。

2016菜鸟双11入库协同平台提前规划，通过搭建大数据预约平台，串联从预约到完成入库的整个执行流程中的信息，沉淀入库过程中商家、干线、承运商、司机、线路、车辆、货品体积等基础数据，以此根据运筹算法推演，双11备货期间的仓内实际库容及剩余库容、并结合实际资源准备及商家、货品分层，智能推荐预约入库方式，实现多样化统筹预约。最终实现了亿级货物的有序入仓，全程仓库无爆仓，商家无等待。



为了保障资源准备和商家预约需求的一致性，S&OP 在入库端也发挥了极大的作用，系统结合大数据算法预测的出入库量、商家预约量，及历史实际出入库量信息，滚动分析双11前后仓内的预计出入库量，数据提前透传给仓、配、运输系统，多角色协同保障上下游链路的资源准备。

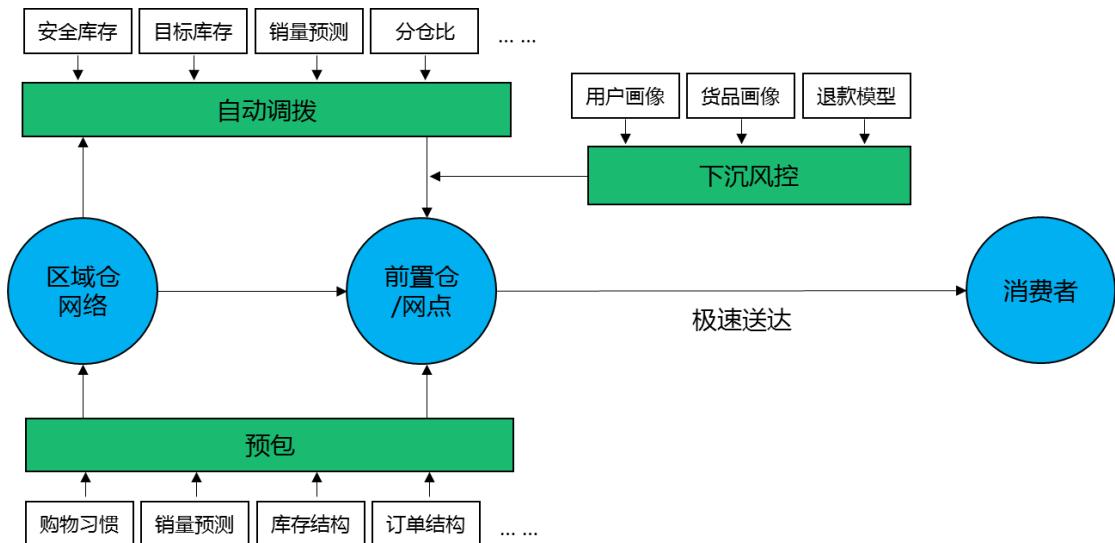
除此之外，菜鸟更进一步做到货物直接铺到离消费者最近，最大化提升消费者的物流体验。结合商家货品粒度的历史销量数据、社会化仓配网络体系、线路时效及成本数据，通过大数据的实时运筹仿真模拟系统，智能推荐最优的货品入仓方案，及仓内库存分布方案。数据直接透传给补货系统，保障货物直接补到离消费者最近的仓库。

打通入库链路的数据，通过选仓分仓推荐、大数据预约平台、S&OP，实现消费者需求、商家预约、仓配资源准备的供需匹配，保障货物有序的实现在仓内的最优分布，是菜鸟让数据发挥价值，实现社会化资源协同的又一利器。

### 3 订单预打包及库存下沉

每逢大促，消费者最关注的问题就是购买的包裹何时送达。除了在入库时让货物直接补到离消费者最近的仓库之外，技术在让货物离消费者更近一点的路上持续献策献力，真正做到“单未下，货先行”。

消费者最可能购买的货品有哪些，哪些货品最可能一起被购买，大促商品及商品组合销量预测，是菜鸟在大促前的必做事项。通过大数据分析和算法模型，结合仓库的商品库存分布、历史发货订单情况，消费者在交易网站的购物车以及收藏夹的数据，预测出商家在双11期间订单结构，仓内会提前对商品进行预包，并随时准备出库，到实际仓库生产作业时，由于进行了预打包，在生产工序将会节省大量时间。



而遍布全国的前置仓及下沉网点是菜鸟仓配网络的重要组成部分，销量预测数据计算后，自动调拨系统也随之启动，依据销量预测结果、各级仓配网络中的库存、缺货、周转、时效、成本等一系列因子，系统会自动给出调拨建议，装车建议，货物以集约化的方式提前下沉至前置仓或网点。

对于下沉可能出现的逆向情况，菜鸟也进行了提前预测，大数据算法下沉风控模型会依据消费者画像、货品画像、退款模型，综合给出退款概率评分，高风险订单不下沉，低风险订单下沉，做到时效和成本的最优平衡。

双11期间的包裹量往往是十倍甚至几十倍的增长，通过大数据算法预测、统筹，提前进行预包、调拨、下沉，可以充分利用双11前几天的黄金时间，极速提高包裹运转效率，提升消费者体验。

## 4 全球智能配送路由

考虑到今年双11恰逢周五，大量订单将在周六日配送，由于核心区域的站点资源有限，再加上双11的单量一般是平时5倍以上，如不将采取有效措施，极有可能会造成末端配送网点过载引发拥堵。

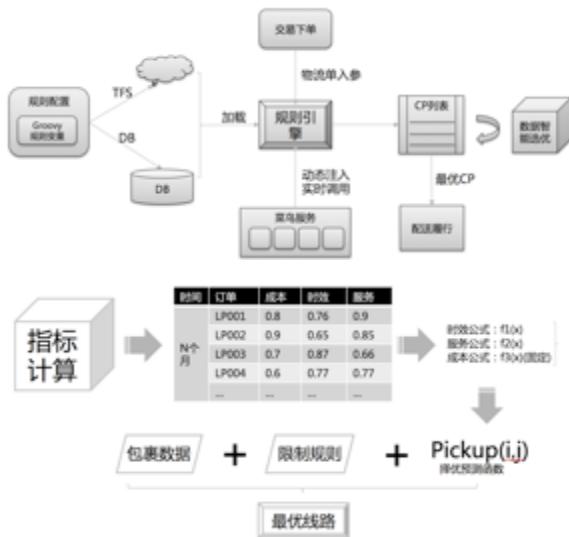
此外，商业办公区域，周末一般很少签收，派送会导致投递资源的浪费，在发货阶段错峰延迟下发此类订单，减少周末高峰期配送压力，可节省因为无法签收带来的重复投递资源。

通过分析高德地理信息库以及日常菜鸟海量配送数据，基于用户的签收周期行为、公用地址的人群分布、POI构词结构等为特征，构建机器学习模型，准确率和办公地址召回率均达到95%以上。基于地址识别技术，菜鸟可以灵活的控制办公地址订单的仓库打包发货的时间，极大降低了“爆仓”的可能。

在国际物流方面，阿里巴巴Aliexpress出口业务和天猫国际进口业务，2016年双11的单量都是去年双11的一倍以上。在阿里巴巴跨境整体单量指数级增长的背后，如何能为每一个包裹选择适合的承运商和线路，让每个包裹的运输在成本、时效和服务上做到最优，是一件商业价值巨大又富有挑战事情。

跨境物流的特点在于距离远、链路长、角色多，这导致每单运费成本较大，物流成本在电商交易整体客单价中占比较大，为了实现优选目标，菜鸟从2015年开始思考如何通过技术手段解决路由问题，最终选择通过人工策略和大数据智能优选的架构方案。

人工策略通过动态规则引擎让运营同学可以根据自身运营经验定制路由策略，根据包裹自身属性做各类限制性过滤；大数据优选是基于历史包裹履行表现做机器学习，从成本、时效、服务质量三个维度为每一个包裹优选最佳履行线路，真正做到智能路由，人工策略和大数据优选大致技术方案如下：



全球智能路由平台 GRC( Global Routing Center )上线以来对业务指标提升明显。在时效方面，通过这套智能路由平台，Aliexpress 在 3~4 个月内成功完成了俄罗斯 Top20 城市从 25 日达到 15 日达的时效提升目标，到欧洲和美国等国家的线路基本也都有 20% 左右的时效提升。成本方面，因为可以根据类目、重量和尺寸优选物流线路，为商家带来的运费节省在 15% 以上；服务方面，平台上线 1 年多之后，通过商家调研和 DSR 评审分析，菜鸟出口物流体验在中国出口电商平台中名列前茅。

## 5 电子面单的应用

面对日趋增长的商家发货量，传统的纸质面单录入及打印的效率低下，并且商家、快递公司、菜鸟间的信息也无法有效互通，不利于物流信息化及基于此的技术优化。在 2014 年，菜鸟推出了电子面单服务，将面单电子化，极大的提升了商家发货效率，并且承载了智能分单等基于大数据的帮助物流公司提升分拨中心、网点等作业效率。

## 2.0 大商家版本



## 2.0 小商家版本



菜鸟电子面单是菜鸟推出的一种在线运单生成、打印与管理的服务，系统能同时提供国内 15 家主流的快递公司、多家落地配公司以及众多的国际物流公司的面单服务，同时我们把菜鸟电子面单服务开放给所有的商家并完成 300 多家主流的 ISV 系统的接入。电子面单系统像一个一个 HUB，通过这个 HUB 商家、菜鸟、物流公司三方系统实现了信息的互联互通。

基于菜鸟电子面单我们能够对商家提供统一的面单服务，商家或 ISV 只需要通过和菜鸟对接就具有同时使用诸多（完成和菜鸟对接的 CP）物流公司电子面单服务的能力。

此外，我们和各大物流公司一起制定了标准电子面单模板，极大的简化与规范了电子面单的模板格式，包括面单尺寸与面单内容格式，同时我们开发了菜鸟电子面单打印工具，制定并开放了菜鸟标准打印规范，确保通过菜鸟电子面单系统打印出来的面单格式统一。

通过电子面单我们可以承载众多的物流服务，助力物流公司，如目前广泛应用的二段码、三段码、在电子面单生成时通过菜鸟大数据智能分单系统实时计算出当前包裹对应的揽件网点、末端分拨中心、末端派件网点以及对应的派件小件员信息，实现面单生成时，包裹全生命周期的路由线路与节点信息全部规划与计算完毕，极大的提升快递操作效率。

## 6 智能分单服务

在过去的大促中，快递企业常有爆仓现象发生，今年双 11，面对破纪录的 6.5 亿件派送包裹，为何却很少看到各种爆仓的媒体报道呢？

包裹在送达消费者手中前，要在分拨中心和网点进行大规模的分拣操作。过去，分拣工通过面单上的地址凭记忆分拣。由于地址量巨大，分拣出错概率高，包裹分发质量难以保证；同时，分拣工需要经过半年的培训才能上岗，在双十一等大促活动，包裹量剧增的情况下，人工分拣会成为效率瓶颈，导致爆仓的各种现象发生。

为了解决该难题，菜鸟算法团队研发了在业内被称为“三段码”的基于大数据的智能分单系统。该系统能够在发货时精确地预测出派件网点和小件员编码，并将编码打印在面单上，指导后续的分拣操作。通过智能分单，分拣工人直接通过面单上的编码进行分拨，分拨准确率可以达到 99.9%，极大提升分拨效率。同时不再依赖专业的分拣工人进行分拣，降低了企业营运成本。目前该系统已经接入国内主流的快递公司，每年可为行业节省成本 6 亿多元。

去年菜鸟推出的“二段码”，帮助分拨中心进行快速分拣，产生了巨大的效应。菜鸟今年在二段码的基础上，推出“三段码”，将预测粒度推进到快递员，帮助网点快速分配包裹给快递员。

菜鸟可以精确的为每个订单分配最佳快递员，并同二段码一样，提前将快递员信息打印在面单上。到达末端网点时，快递员不再通过面单上的详细地址挑选出属于他派送的包裹，而是根据面单上打印的“三段码”，进行快速分拣。

今年双十一，三段码在帮助网点快速分拣、消化包裹上，扮演了极其重要的角色。临时分拣员不再需要繁琐的培训就能快速上手；熟练分拣员也不再需要细读每一个地址，靠人工记忆去分拣。三段码简化了分拣操作，提升了分拣效率，平均每单节省 2.4 秒。今年双十一，包裹在网点的停留时间比去年足足减少一个小时，总计为小件员节省派件时间 16 万个小时。

基于电子面单和智能分单，主流快递公司今年大规模启用自动化分拨设备，大幅提升了核心节点的效率。所以消费者才能不断感知双 11 包裹送的越来越快。



## 7 最后一公里的智能派送

物流末端揽派场景中，每个小件员的揽派范一般是多个小区、学校、工厂、产业园、写字楼等有自然边界范围的 POI 集合，将包裹按照小区、学校、工厂、产业园、写字楼等聚在一起，使得聚在一起的包裹实际行走距离约等于直线距离。

让聚在一起的包裹集中完成揽派，从而大大减少因频繁绕开天然障碍物（主干道、高架桥、河流、小区围墙、山等）而产生的多余路径，因此在物流末端场景中包裹聚合对快递员揽派效率提升明显。

现在，早上快递员满载包裹从网点出发，打开快递员 APP，通过后台包裹聚合算法，告知小件员当前要派送的包裹在小区、学校、写字楼等维度的分布，便于快递员对每一个聚合后的包裹实现批量进行通知、签收、核对、查找小区空自提柜（自提柜聚合）等操作，大大提升快递员的派件任务处理效率，还可以依据聚合后的包裹实时计算出最优的派送路径。

将当天某个时段的预约包裹按照某一小区块（小区，校区，写字楼等）并在一起，构成并单簇，一起分配给同一个快递员，若某个单已被接单但未被揽收，其附近小范围（同一个小区、写字楼、学校等）内若有新订单，则新订单直接追到当前订单的小件员抢单列表里。

通过并单和追单实现包裹的批量揽收，降低快递员揽收包裹的成本，增加快递员的收入，双 11 期间追并单单量占总体揽收单量的 40%。

快递员在派件过程中还可以顺道揽收包裹，在派送过程中，通过派件列表中包裹状态的变化结合快递员的实时位置，系统能判断出快递员当前在哪个小区派件，和接下来要去的隔壁小区，将当前或隔壁小区的寄件包裹直接分配给当前派件员，从而增加快递员的收入。

## 8 结语

每年的双 11，巨大的包裹量，对于菜鸟都是一次冲击和洗礼，磨刀不误砍柴工，菜鸟也一直在各个领域各个环节都深入思考，探索通过数据技术来提升效率优化服务。前文提及到的供应链、订单调度、智能分单等，以及其他未提及到的优化实践，正是通过数据和技术的力量，再一次捍卫了菜鸟双 11 “丝般顺滑” 的技术支撑，给消费者提供了“丝般顺滑”的物流体验。

# 第六章 大数据

# 6.1 双 11 数据大屏背后的实时计算 处理

作者：藏六 黄晓锋 同杰

## 1 双 11 数据大屏的实时计算架构

### 1.1 背景

2016 年的双 11 我们的实时数据直播大屏有三大战场，它们分别是面向媒体的数据大屏、面向商家端的数据大屏、面向阿里巴巴内部业务运营的数据大屏。



每一个直播大屏对数据都有着非常高的精度要求，特别是面向媒体的数据大屏，同时面临着高吞吐、低延时、零差错、高稳定等多方面的挑战。在双 11 的 24 小时中，支付峰值高达 12 万笔/秒，处理的总数据量高达百亿，并且所有数

据是实时对外披露的，所以数据的实时计算不能出现任何差错。除此之外，所有的代码和计算逻辑，都需要封存并随时准备面对监管机构的问询及和检查。

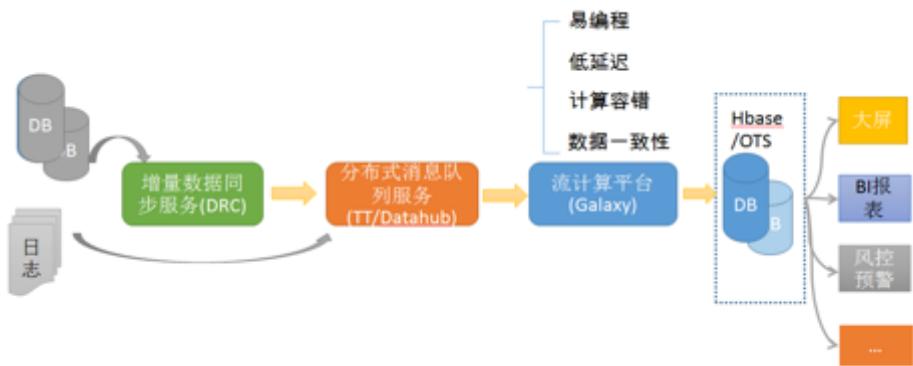


在面向商家的数据直播大屏中，为了实时观察店铺流量情况，需要处理全网的所有流量数据。另外，为了让商家更全面更细地分析流量，增加了很多实时的数据维度，比如我们需要计算每个商家的访客数量、加购数量、热销商品、流量来源、每个商品的访问情况等等。

面向阿里巴巴内部业务运营小二的数据大屏，则提供了最为丰富角度的数据。除了大盘数据之外，还针对不同业务进行了定制，如淘宝、天猫、天猫国际、天猫超市、无线、聚划算、淘海外、飞猪旅行、村淘、AliExpress、卖全球等业务模块，每个业务模块还会进行各维度的交叉分析，比如行业、类目、地域等。这些数据监控了当天活动进展的方方面面，也是当天活动应急响应决策的重要依据。

以上每个直播功能需要实时处理的数据量都是非常庞大的，每秒的总数据量更是高达亿级别，这就对我们的实时计算架构提出了非常高的要求。在面对如此庞大数据的时候，我们的实时处理是如何做高精度、高吞吐、低延时、强保障的呢？

## 1.2 实时计算处理链路整体架构



DRC: DRC(Data replication center)是阿里自主研发的数据流产品，支持异构数据库实时同步、数据记录变更订阅服务。为跨域实时同步、实时增量分发、异地双活、分布式数据库等场景提供产品级的解决方案。

TT: TT(TimeTunnel)是一个高效的、可靠的、可扩展的消息通信平台，基于生产者、消费者和 Topic 模式的消息中间件。

GALAXY: Galaxy 是全球领先的通用流计算平台，支撑阿里绝大部分实时计算任务，支持专有云打包输出。平台易用性高、数据准确性 100%、集群线性扩展、支持容错、支持多租户、资源隔离。双 11 流量暴增情况下，Galaxy 做到毫秒级计算延迟，满足极高稳定性要求，每天处理消息量万亿级。

OTS: 开放结构化数据服务 (Open Table Service, OTS) 是构建在飞天大规模分布式计算系统之上的海量结构化和半结构化数据存储与实时查询的服务，OTS 以数据表的形式组织数据，保证强一致性，提供跨表的事务支持，并提供视图和分页的功能来加速查询。OTS 适用于数据规模大且实时性要求高的应用。

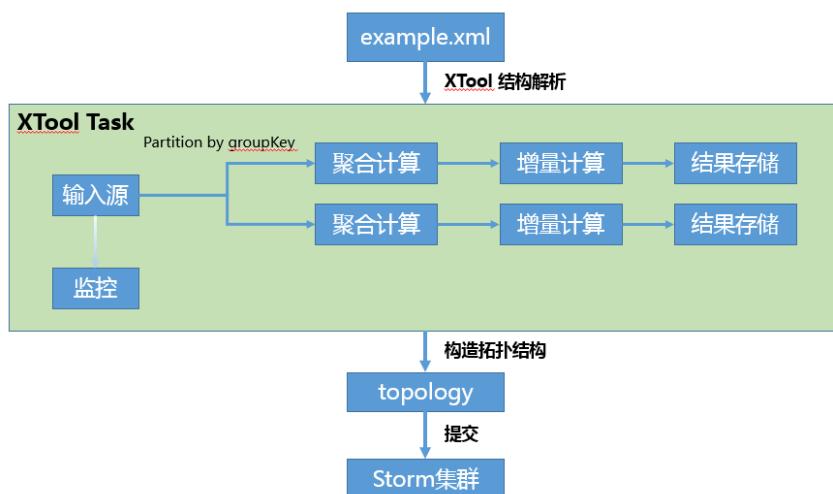
### 1.3 实时计算聚合组件：XTool 介绍

在实时数据统计的场景中，大部分都是根据某些维度进行去重、求和、算记录数、求最大值、求最小值、求平均值、算排行榜等聚合计算，另外还会涉及到实时多表 join、静态维表关联、时间窗口管理等。而 storm 只提供了最底层的计算算子，每一次的聚合操作、关联操作等都是需要代码开发的，效率非常低，并且对于实时计算中 checkpoint、snapshot 的存储和恢复没有标准化的定义。

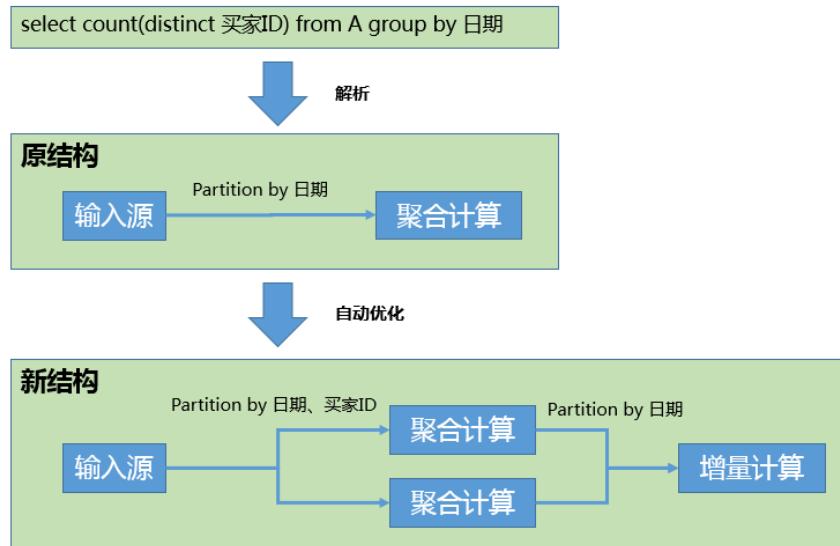
为了提高实时计算开发效率和降低运维成本，我们在 storm 的 trident 语义上，把通用聚合功能封装成了 XTool 组件，提供配置化定义实时计算拓扑，不

需要任何的代码编写。另外，聚合组件还提供了重跑、续跑、`exactly one` 等各种特性，方便修复日常运维中遇到的数据问题。XTool 是一个完整的实时计算解决方案，对接了数据同步中间件（TimeTunnel）、数据储存（HBase）、流计算引擎（storm）等，把这些系统结合起来形成一个通用的实时计算服务架构。

XTool 通过 xml 文件来描述实时任务的拓扑结构，只需要在里面中定义数据源输入、聚合维度、聚合指标、输出信息、任务监控信息等，XTool 会自动解析成对应的聚合 Task，并在 storm 集群中提交实时任务，用户不需要管临时数据如何存储、故障如何恢复、事务信息保障等细节问题。



海量数据实时处理中经常会遇到雪崩、数据倾斜、数据重复等问题，除了通用功能外，XTool 通过输入限流、自动 Hash 分桶、主键唯一化等策略来解决以上的问题。例如在遇到数据倾斜问题时，只需要配置一个标记，XTool 在解析 xml 文件时会自动对 `distinct` 操作进行 Hash 分桶去重，在下层直接进行求和就可以得到去重指标。



高性能是实时计算中赖以生存的特性，XTool 组件为了满足百万/秒甚至千万/秒的 qps 要求，进行了大量的性能和资源调优工作。比如进行指标去重的时候支持精准去重、布隆去重、基数估计去重等模式，在业务需求和资源使用率上取得很好的平衡。另外，XTool 会按照 LRU（最近最少）算法缓存聚合结果在内存中，并把维度的 key 做成布隆集合，当新数据来的时候，可以避免不必要的读库操作。XTool 还充分利用了 localOrShuffle 特性（类似 Hadoop 中的 map 计算本地化），把计算移到数据存储节点上进行，大幅减少数据序列化次数，性能提升非常显著。

目前几乎所有面对商家、媒体、运营小二、高管的实时计算应用都是通过 XTool 来实现的，其经历了几次双 11 的考验，表现出非常高的吞吐量和稳定性保障，功能也越来越丰富。后面计划把 XTool 通过 Apache Beam 来实现，让其不再只能依赖 storm 计算引擎，可以灵活在各个引擎间进行切换，比如 Flink、Apex、Spark 等。

## 1.4 OneService 服务介绍

统一数据服务平台（OneService）是数据技术及产品部-产品技术主导的，以集团数据公共层 One Data 提供上层应用接口依始，提供简单数据查询服务（承接包括公共层所有数据在内多个 BU 简单数据查询服务），复杂数据查询服务（承接集团用户识别（OneID）、用户画像（GProfile）复杂数据查询服务），实时数据推送服务 三大特色数据服务。

1. 简单数据查询服务：定位简单数据查询引擎，通过物理表、逻辑表组合绑定的方式，将具体数据来源屏蔽在引擎内部，用户在平台简单配置来源表等元数据信息，便可以做到不需要依赖任何应用、代码获得接口服务（hsf 服务），目前 SmartDQ 支持接入数据源类型为 HBase/Mysql/Phoenix/OpenSearch；
2. 复杂数据查询服务：复杂数据查询引擎，目前承接了集团用户识别（OneID）、用户画像（GProfile）等复杂数据处理查询；
3. 实时数据推送服务：通过数据推送机制，对外提供高性能、高稳定性的 JsonP/webSocket 接口，提供流量和交易相关的实时数据推送服务。

OneService 还重点解决了服务的高可用问题：

1. 服务本身的高可用：例如多机房部署、HSF 分组隔离等，这里不再展开。
2. 查询链路的快速切换能力：媒体大屏用到的 GMV 等指标，后台一般都会冗余多个计算任务，并写入到多个 HBase 集群上。如果某个集群出现不可用或者数据延迟，需要能秒级切换到另一个集群上。

## 2 大数据整体链路如何应对双 11 的挑战

### 2.1 如何进行实时任务优化

优化工作在实时计算中显得尤为重要，如果吞吐量跟不上的话，也就失去了实时的特性。吞吐量不佳原因非常多，有些是跟系统资源相关，有些跟实现方式相关，以下几点是实时任务优化中经常需要考虑的要素：

1. 独占资源和共享资源的策略：

在一台机器中，共享资源池子是给多个实时任务抢占的，如果一个任务在运行时的 80% 以上的时间都需要去抢资源，这时候就需要考虑给它分配更多的独占资源，避免抢不到 CPU 资源导致吞吐量急剧下降。

2. 合理选择缓存机制，尽量降低读写库次数：

内存读写性能是最高的，根据业务的特性选择不同的缓存机制，让最热和最可能使用的数据放在内存，读写库的次数降下来后，吞吐量自然就上升了。

3. 计算单元合并，降低拓扑层级：

拓扑结构层级越深，性能越差，因为数据在每个节点间传输时，大部分是需要经过序列化和反序列化的，而这个过程非常消耗 CPU 和时间。

#### 4. 内存对象共享，避免字符拷贝：

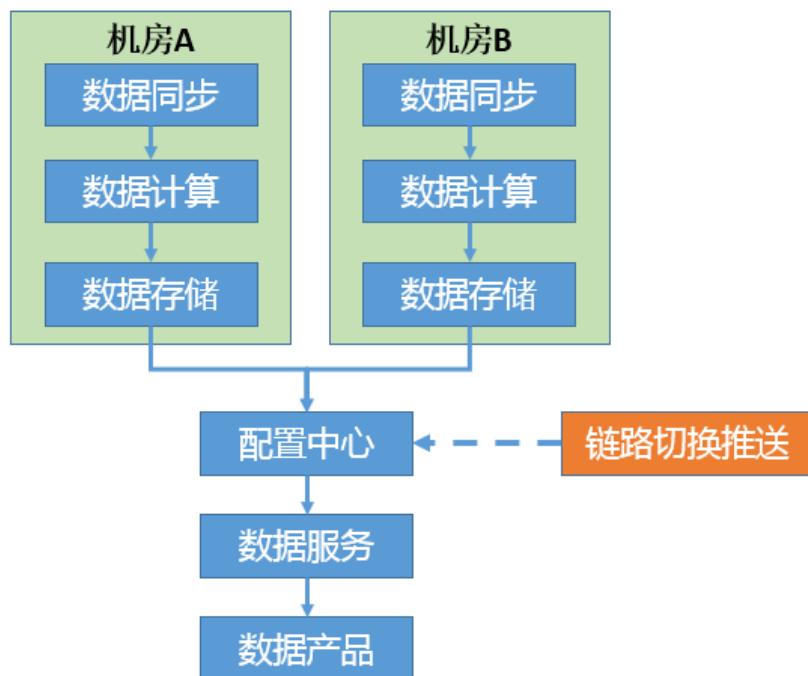
海量数据处理中，大部分对象都是以字符串存在的，在不同线程间合理共享对象，可以大幅降低字符拷贝带来的性能消耗，不过要注意不合理使用带来的内存溢出问题。

#### 5. 高吞吐和低延时间取平衡：

这两个特性是一对矛盾体，当把多个读写库操作或者 ACK 操作合并成一个的时候，可以大幅降低因为网络请求带来的消耗，不过也会导致延时会高一些，在业务上衡量做两者的取舍。

## 2.2 如何进行数据链路保障

实时数据的处理链路非常长( 数据同步->数据计算->数据储存->数据服务 )，每一个环节出现问题，都会导致实时数据停止更新。实时计算属于分布式计算的一种，而单个节点故障是常态的，这种情况在直播大屏中特别明显，因为数据已经不再更新了，所有的客户都会发现数据出现了问题。因此，为了保障实时数据的可用性，需要对整条计算链路都进行多链路搭建，做到多机房容灾，甚至异地容灾。



由于造成链路问题的情况比较多，并且一般不能在秒级里面定位到原因，因此会通过工具比对多条链路计算的结果数据，当某条链路出现问题时，一定会比其他链路计算的值小，并且差异会越来越大。这时候会一键切换到备链路，并且通过推送配置的形式让其秒级生效，所有的接口调用会立刻切到备链路，对直播大屏完全透明，并且用户也感知不到故障的发生。

## 2.3 如何进行压测

在双十一备战中，会对实时链路进行多次压测，主要是模拟双十一的峰值情况，验证系统是否能够正常运行。压测都是在线上环境进行的，这里面分为数据压测和产品压测，数据压测主要是蓄洪压测：类似大坝中把几个小时甚至几天的数据积累下来，并在某个时刻全部放开，达到模拟双十一洪峰流量的情况，这里面的数据属于真实的数据。比如通过把实时作业的订阅数据点位调到几个小时或者几天前，这时候每一批读到的数据是最多的，对实时计算的压力也最大。

产品压测还细分为产品本身的压测和前端页面稳定性测试：

### 1. 产品压测：

通过收集大屏服务端的所有读操作的 url，通过 PAP 压测平台进行压测流量回放，按照 QPS：500 次/每秒 的目标进行压测。在过程中不断的迭代优化服务端的性能，提升大屏应用处理数据的性能。

### 2. 前端页面稳定性：

将大屏页面在浏览器打开，并进行 8 小时到 24 小时的前端稳定性测试。监控大屏前端 js 对客户端浏览器的内存、CPU 等消耗。检测出前端 js 内存泄漏等问题并 fix，提升前端页面的稳定性。

## 3 架构升级，如何更轻松应对未来的双 11

流式计算是对批处理的一个补充，其中最重要的特点就是实时性，而随着业务的增长，实时计算需要处理的数据也会随着增长，因此，吞吐量的提升是实时计算中最大的挑战。除此之外，降低开发成本、提升运维效率也是我们一直追求的目标，让实时计算技术给更多的业务场景提供价值。

我们会通过架构升级、底层计算引擎优化、业务逻辑优化等方式来突破实时

计算的性能瓶颈，力求把实时计算的性能提高一个数量级。另外，后面计划把 XTool 移植到 Apache Beam (Google Dataflow)上面，并尝试底层不同引擎的 Beam Runner ( Flink , Apex , Gearpump 等新开源流计算技术以及阿里巴巴自己的 Galaxy ),封装数据统计的流计算开发产品( 图形化定义流计算拓扑结构 )，让其保持良好的可移植性；同时让流计算开发对 ETL 开发人员更友好，降低流计算的开发成本；加强阿里巴巴在开源社区的影响力。

## 6.2 双 11 背后的大规模数据处理

作者：惠岸 朋春 谦乐

### 1 实时数据总线服务-TT

TimeTunnel ( TT ) 在阿里巴巴集团内部是一个有着超过 6 年历史的实时数据总线服务，它是前台在线业务和后端异步数据处理之间的桥梁。从宏观方面来看，开源界非常著名的 Kafka+Flume 的组合在一定程度上能够提供和 TT 类似的基础功能；不同的是，在阿里巴巴的业务体量和诉求下，我们有比较多的配置管控、资源调度、轨迹校验和血缘识别等方面的工作（图 1）。

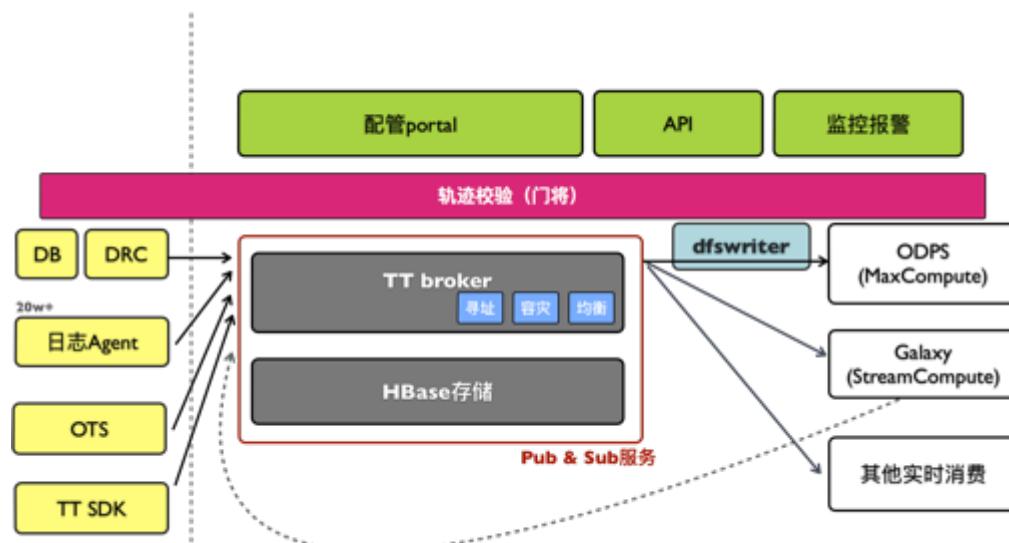


图 1 TimeTunnel 产品架构

#### 1.1 Pub/Sub 服务

通过图 1 我们清楚地看到，TT 的核心部分是一个基于 HBase 做中间存储的 Pub/Sub 服务，它提供了一个能支撑高读写比、大吞吐量和数据不丢的队列服务。除此之外，基于日常运维考虑，我们还支持了按时间 seek 和弹性伸缩的能力。

数据需要在 Pub/Sub “落地”的需求一方面来自于业务上对热点数据多份消费的考虑 ,另一方面一些在线算法方面的应用需要经常性地对数据进行回放训练 ,数据 “落地” 能够比较好地对前后台进行解耦。事实上 ,TT 里最热门的数据 ( 例如天猫交易相关 ) 有超过 100 倍的读写比 ; 而从整体来看 , 仅双 11 当天流出 TT 的数据也比流入的数据多了 3 倍以上。

选择 HBase 作为中间存储的原因是能够成本较低地复用基于 HDFS 的多副本存储能力 , 以及 HBase 自身在提供读写服务时对于热点数据的内存管理能力。图 2 是写入 TT 的数据在 HBase 中的存储模型 , 我们在 broker 层面通过构造合理的 rowkey 来使得同一个分区下的数据可按 rowkey 顺序 scan ; 同时 , 因为在生成 rowkey 的时候我们使用了 broker 上的时间戳作为高位变量 , 因此能很方便地提供按时间 seek 的能力。

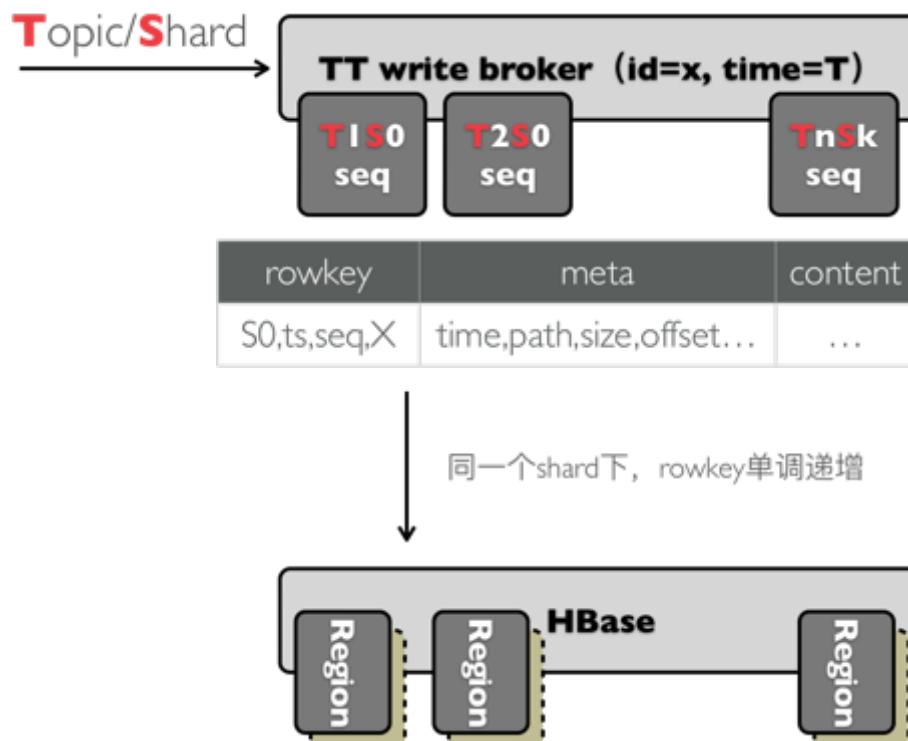


图 2 数据在 HBase 中的存储模型

## 1.2 数据采集

图 1 左侧黄色部分是 TT 的数据采集方案。我们通过以下途径来准实时地收集前台业务产生的增量数据 :

1. 依赖 DRC 实现对 MySQL、OceanBase 以及 Oracle 等前台业务数据

- 库的增量变更进行捕捉解析；
2. 自研的日志 Agent 部署在数十万台的应用服务器上，准实时地捕捉应用日志的变化；
  3. 和其他一些内部主流存储例如 OTS 进行打通；
  4. 用户采用 TT 提供的 SDK 主动写入。

随着集团内重要业务异地多活架构和全球化的发展，数据采集分散在跨越数千甚至上万公里的多个 IDC 中；而与此相反，以 Galaxy、ODPS 为代表的大数据计算服务则需要考虑充分地利用大集中的架构来提升吞吐能力。因此，不可避免地在数据采集过程中需要对数据进行缓冲和压缩以尽可能降低长途链路对于吞吐量的负面影响。

矛盾的是，缓冲意味着前端产生的数据需要在采集端“等待”，也就意味着消费方看到的数据是延迟的。这对于像阿里妈妈这样依赖 TT 做反作弊和实时计费的业务来讲是很难接受的，数据延迟意味着损益，意味着用户体验的显著下降。同样地，压缩也是要消耗采集端的服务器资源的，尤其在双 11 这样的场景下，前台业务对于采集端的功耗尤其敏感。

遗憾的是，世界上从来没有一个只带来好处而没有任何弊端的事物，软件和产品的设计中处处都是折衷和取舍。除了在技术层面将实现细节做到尽可能极致，TT 为了服务这些不同的场景，也提供了一些可配置的参数例如 buffersize、sendthreads、compressLevel 等用来匹配用户对延时、性能以及功耗的不同需求。

### 1.3 轨迹校验

TT 区别于其他类似产品的最大之处，是我们通过技术埋点实现了一套完整的数据轨迹校验的方案——我们称之为“门将”。轨迹校验的目的在于通过监控的手段来保证“数据不丢”，设计得好，甚至可以识别出数据的重复、乱序等情况。

几乎所有类似的产品都宣称自己能做到“数据不丢”，当然也包括配备了“门将”之前的 TT。有意思的是，几乎所有类似的产品都会被“丢数据”这个问题困扰，同样包括 TT。因为我相信我们一定有能力在软件设计以及编码实现方面做到“数据不丢”的承诺，但往往会在一些预期外的异常 case、版本升级或者系统耦合的地方出现这样那样的纰漏，从而导致下游消费方看起来缺失了部分数

据。

以日志采集为例，我们碰到过因为操作系统的限制（请参阅 max\_user\_watches 相关的说明），inotify 没有通知到新文件的产生而发生整个文件漏采集；也碰到过因为软件的 bug 在递归创建子目录的情况下出现了时序问题导致文件漏采集；还碰到过保存在应用服务器上的 checkpoint 文件被意外损坏导致的“丢数据”。这样的案例实在太多，而且防不胜防。

所以，工业界真正的“数据不丢”我认为是有完备的机制能够快速地发现数据丢失，考验的是系统的监控能力。

上文提到过，TT 支撑着阿里妈妈的实时反作弊和点击计费业务；同样地，蚂蚁金服大量涉及资金核对和商户对账的业务也将身家性命托付在 TT 上。这样的业务不允许有任何原因导致的数据正确性问题。

“门将”的核心思路是在采集端往 TT 写入数据的同时，构造恰当的 meta，将数据“链表化”，从而能够在“门将”的校验服务里对数据轨迹进行还原，进而和源头进行校验（图 2）。

仍然以日志采集为例。在采集过程中，我们以 ip+dev+inode+sign 来唯一识别内网上的一个文件，在构造 meta 时记录下当前数据包在原始文件中的 offset 和当前数据包的大小 size，那么对于同一个文件的多个数据包，通过 offset 和 size 就能快速地识别出文件内有没有被重复采集或者遗漏采集。如果在恰当的时间内与这台机器上 ls 命令得到的结果进行比对，就很容易发现有没有文件被漏采集。

## 1.4 小结

所有的技术实现都是业务需求的抽象，这些需求有可能来自于大多数用户需要用到的功能，更有可能来自对上下游业务架构和场景的理解。数据总线服务是一个和业务架构耦合非常密切的基础组件，阿里巴巴集团独特的技术架构、多样性的存储方案和横向平台化的研发模式赋予了 TT 探究更复杂问题的原动力。

在 2016 年双 11 这样一个万众瞩目的时间点，TT 通过前期的软件性能和机房规划上的努力，高峰期单一集群承担了 15GB/s 的写入和 50GB/s 的读取流量，首次做到了对所有业务进行不降级服务。这对于我们、对于搭建在 TT 上的众多业务，都是极大的鼓舞。

## 2 大规模数据流处理技术- Galaxy

每年双 11 除了“折扣”，阿里人关注的另一个焦点，就是面向全世界媒体直播的“实时大屏”（如下图所示）。包括总成交量在内的各项指标，通过数字维度展现了双 11 狂欢节这一是买家，卖家及物流小二共同创造的奇迹！



图 3：双 11 媒体直播大屏

为实现这一大屏，背后需要实时处理海量的、庞大电商系统各个模块产生的交易日志。例如双 11 当天产生的日志量达到了 PB 级别，而每秒处理的峰值更是高达近 1 亿事件！

如此大规模、高吞吐和低延时计算，带来一系列世界级的技术挑战，包括：

**1. 实时编程**：流式的数据处理给业务逻辑的表达和推理带来了很多的复杂性。特别面对不断变化的业务需求，如何帮助用户快速地编写和验证实时计算逻辑是至关重要的。

**2. 低延时**：实时计算强调计算延时和结果的时效性。例如实时大屏对计算延时特别敏感，每年的双 11 都超越前一年更早地达到相同的成交量，系统需要在秒级甚至毫秒级反应出每一笔交易。即使在流量高峰时（双 11 晚 0:00 点）也需要保证延时！

**3. 集群利用率**：为提高资源利用率，我们将不用业务的实时处理逻辑共享一个集群。这样的共享也带来性能隔离的问题，即如何让同一台物理机上的不同逻辑任务不互相干扰。这也是大部分开源框架忽略的重要问题。

**4. 严格容错及数据一致性**：随着应对高吞吐而不断扩大的集群规模，各种软硬件故障都难以避免。如何保证实时计算在任何故障下都能产生准确、一致

的计算结果，不遗漏、重复事件输出，也不引起内部状态的偏差，是另一个重大挑战。

**5. 多样化场景支持**：随着实时决策对业务的价值越来越多，系统还需要支持越来越复杂和多样化的场景，如在线机器学习、结合图计算实现的动态关系网络分析等等。

下文介绍 Galaxy 的重要技术创新，简要描述它们如何帮助应对以上技术挑战。

## 2.1 SQL 与增量计算——复用熟悉的离线思维，自动实现增量（流式）计算

为了简化用户编程，特别是利用原有的离线计算作业快速实现实时计算，Galaxy 允许通过高层描述性语言，如用户熟悉的 SQL 来编写流计算作业。例如下面的例子，通过简单几行 SQL 代码就可以实现过滤、双流关联等业务逻辑。

```

1  create stream table overwrite  s_effect_app_other_page_sys_click
2 曰(
3    client_ip          string
4    ,protocol_version  string
5    ,imei               string
6    ,imsi               string
7    ,brand              string
8  );
9
10 create tmp table overwrite t_effect_user_track_click
11 曰(
12   acookieid         string
13   ,sessionid        string
14   ,ali_trackid      string
15   ,pvtime            string
16   ,user_nick         string
17 );
18
19
20 insert into t_effect_user_track_click
21 曰select
22   t.page
23   ,t.event_id
24   ,t.arg1
25   ,t.arg2
26   ,t.arg3
27   from s_effect_app_other_page_sys_click t
28 曰where ('( t.event_id='1111'
29           and t.page in('Page_DetailController' , 'Page_Shop1'))
30           or t.event_id='1111'
31           or (t.app_key='1111' and t.event_id = '1111') --for iPad ad
32           )
33           and t.app_key in('1111','1112')
34           and t.reserve2 !='-' and t.reserve2 != ''
35           and length(COALESCE(get(t.args, ',', '='), 'clickid'),get(t.args, ',', '='), 'clickId'))>0
36           and COALESCE(get(t.args, ',', '='), 'clickid'),get(t.args, ',', '='), 'clickId')) <> '\$\{clickid\}'
37           and (length(t.long_login_nick)<=100 or  t.long_login_nick is null);
38
39
40 insert into t_effect_user_track_click_alitrack_2 --两者关联
41 曰select
42   t.cookieid
43   ,t.sessionid
44   ,a.ali_trackid
45   ,t.pvtime
46   ,t.clickid
47   from t_effect_user_track_click_noalitrack_1 t
48 曰join t_effect_user_track_click_alitrack_max_1 a
49   on t.clickid=a.clickid;

```

在执行时，由于数据是以流式进入系统的，用户的 SQL 就像数据库视图一样，被自动增量更新，并以一定的频率输出结果，供下游计算和展示。

这一独特的编程设计，不仅帮助用户借助熟悉的离线处理思维表达实时计算逻辑，也因为同样的程序可以在离线系统运行，使得结果的对比变得易如反掌。

## 2.2 高性能优化引擎——实现低延时计算！

用户的 SQL 脚本经过编译优化，生成数据流图，然后运行于 Galaxy 的分布式引擎之上。相比开源数据流引擎，Galaxy 引擎在“阿里巴巴规模”下，面对真实复杂的业务场景做了很多优化。包括自适应的消息打包、自定义序列化、数据行+列压缩、先进的内存管理、和内部缓存队列和线程模型，以及基于下游向上游“反向”传递压力的流控策略等。

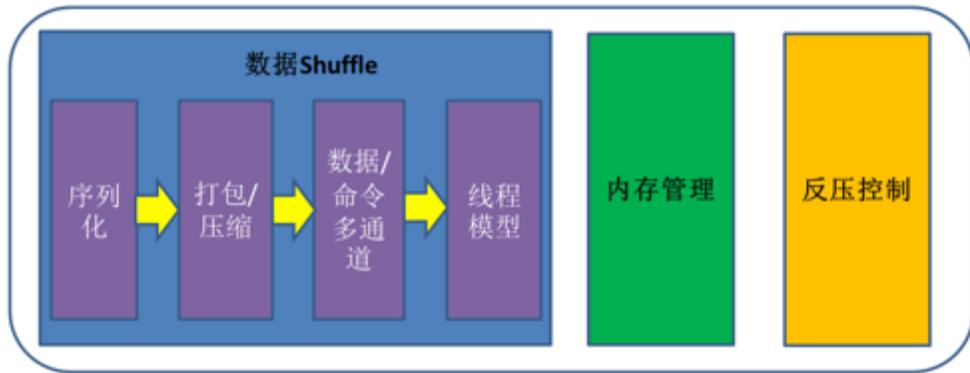


图 4：Galaxy 优化执行流和运行时模块

经过以上一系列的优化，Galaxy 相比去年提升了 6 倍左右的吞吐性能。下图显示了 Galaxy 相比开源系统的性能优势。在面对今年双 11 3 倍于去年的峰值情况下，表现非常稳健。

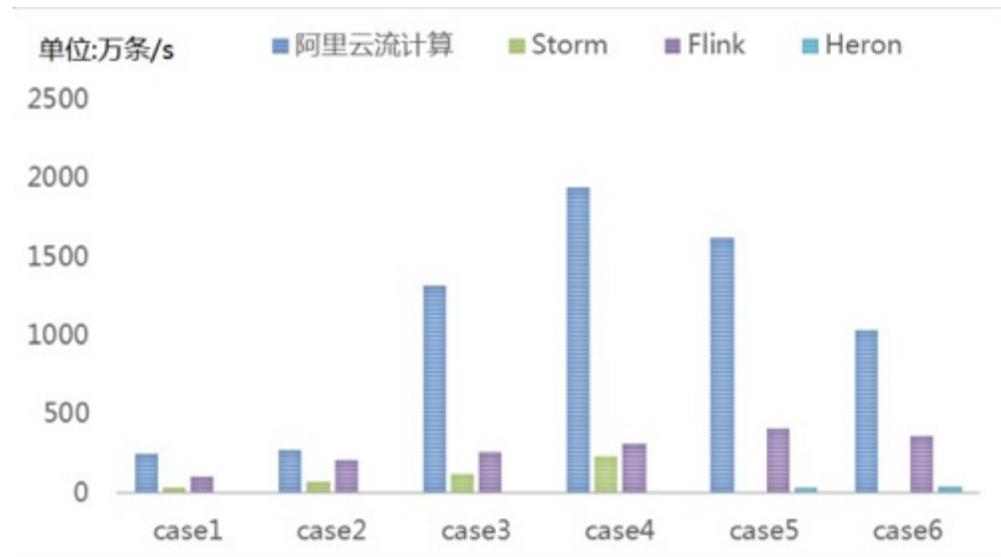


图 5：开源框架性能对比，通过“窗口 WordCount ( 6 组参数 )”基准测试获取。

## 2.3 灵活的资源调度

Galaxy 面对阿里巴巴集团众多业务场景，将不同业务放置于大规模（几千台服务器组成的）共享集群中，以提高资源利用率。另一方面也随之带来了“多租户”环境下的作业资源隔离问题，它直接影响资源的有效利用和作业的计算性能。

经过多年的积累，Galaxy 支持 CPU、内存、网络和磁盘 I/O 等多维度资源的隔离。例如，对于 CPU 的隔离支持灵活的 min-max 策略，既保证了每个作业最基本的资源需求，也使的空闲的资源被最大限度利用。

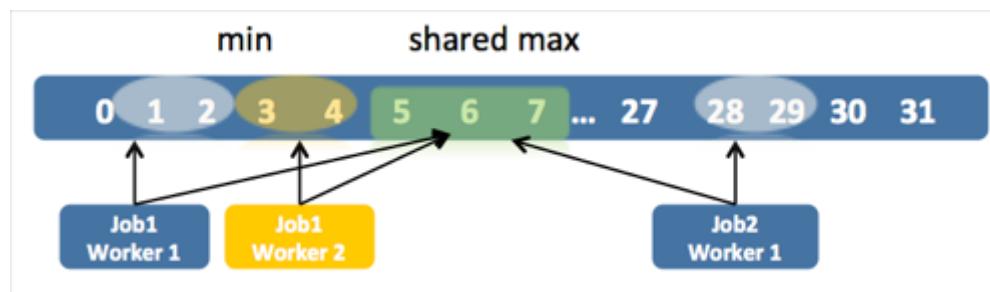


图 6：作业维度的 CPU 资源 min-max 共享模型

在此基础上，Galaxy 的资源调度还支持一定比例的“超卖”、作业优先级调度、动态负载均衡和微作业共享单一物理核等多种机制。对于资源消耗特别大的作业还支持动态按需分配（即资源的弹性分配）。在满足复杂的运维要求和实时计算连续性的同时，实现了高效的资源利用和性能隔离。

## 2.4 容错与状态管理

流计算需要连续处理可能无界的输入和连续产生输出。在长时间运行中，大规模计算集群的各种软件或硬件故障难以避免。由此对于计算和中间结果（如内存状态）的容错就至关重要。为了做到精确的容错和故障恢复，保证结果的准确性。Galaxy 支持多种灵活的容错策略，以在不同计算特性下，权衡容错资源消耗和恢复性能。如基于输入的重新计算、状态检查点（checkpoint），甚至是多副本的状态和计算容错等。

特别是自动的分布式增量检查点功能，系统自动利用内存、本地磁盘和远程存储构成的多级存储，在不影响流计算延时的情况下异步实现了计算状态的持久化。当有故障发生时，保存的状态可以被快速加载。这一切对用户都是无感知的。

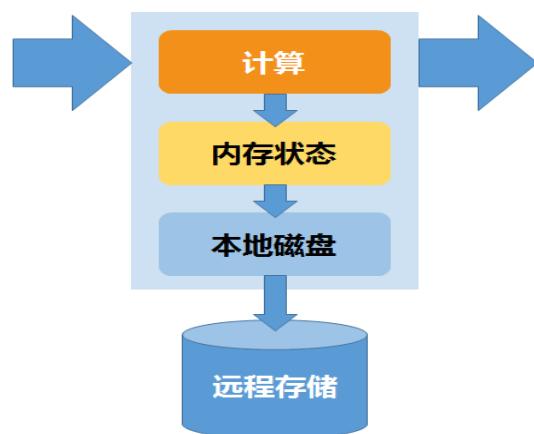


图 7：自动利用多级存储的流计算状态管理

## 2.5 开放可编程 API ( 兼容 Apache Beam )

除了 SQL 这样高层的描述语言和用户自定义逻辑 ( UDF ) , Galaxy 还支持 Apache Beam API , 以提供更为灵活的实时逻辑编程。 Beam 是一个统一开放的大数据应用编程接口 , 可以同时描述离线和在线逻辑 , 最早由 Google 提出。 Beam 提供了功能丰富的编程接口 , 能有效的处理有界、无界、乱序的数据流输入。 下面显示了通过 Beam 实现的流式 WordCount 的例子 :

- 1.指定 Runner(底层计算引擎)创建一个 Pipeline。
- 2.使用 Source 在 Pipeline 上生成一个 PCollection , 输入数据。
- 3.对 PCollection 应用 Transforms 操作 , 比如 wordCount 中的 count 操作。
- 4.对最后的 PCollection 应用 Sink , 输出结果到外部存储中。
- 5.Run Pipeline 到底层的计算引擎中。

使用 Beam 实现 WordCount 代码样例

```
public static class CountWords extends PTransform<PCollection<String>,
    PCollection<KV<String, Long>>> {
    @Override
    public PCollection<KV<String, Long>> apply(PCollection<String> lines) {
        // Convert lines of text into individual words.
        PCollection<String> words = lines.apply(
            ParDo.of(new ExtractWordsFn()));
        // Count the number of times each word occurs.
        PCollection<KV<String, Long>> wordCounts =
            words.apply(Count.<String>perElement());
        return wordCounts;
    }
}
```

借助 Beam , 用户可以利用高性能的 Galaxy 引擎 , 定制面向特定领域的系统交互接口。同时 , Galaxy 今后也将兼容更多生态 ( 如 Spark Streaming 和 Flink Streaming API ) 。

## 2.6 可视化集成开发平台和自动化运维

Galaxy 还提供了“一站式”的集成开发环境——贝叶斯 ( Bayes , <https://data.aliyun.com/product/sc> ) 和自动化运维平台——特斯拉( Tesla )。通过它们，用户可以方便地管理流计算应用的生命周期，包括编程、调试、监控运维，极大地降低了流计算系统的使用门槛。

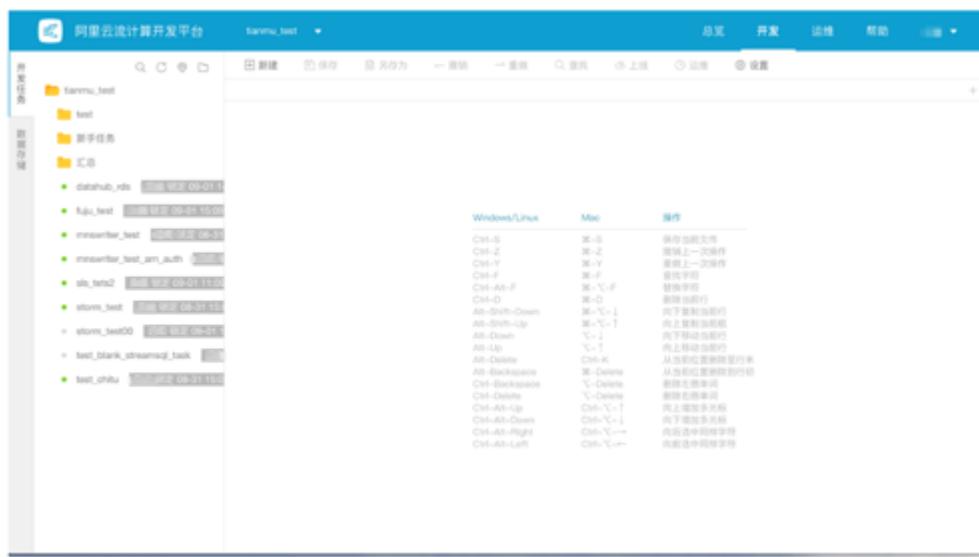


图 8：贝叶斯集成开发环境

## 2.7 双 11 的宝贵工程经验

为保障系统在双 11 平稳支撑业务，在以上功能基础上，我们还总结了完整的全链路保障方法：

- **主备双链路容灾**：利用 Galaxy 对多副本执行的支持，面向双 11 重点媒体大屏等实时业务，实现了跨机房的多链路副本。哪怕是整个机房的故障，都能在秒级自动切换到另一副本上执行，保障了双 11 系统高可用。
- **实时全链路监控**：我们从数据采集、读取、消费、入库各个环节都增加延时指标的埋点，可以清晰地看到整条链路各个阶段的延时，快速分析哪个组件性能瓶颈。另外，针对作业本身运行情况，比如输入吞吐、流量、CPU 和内存消耗，都做了实时分析和展示的系统，能在秒级发现作业的异常。

- **运维诊断工具**：为应对各种应急响应，我们做了一套完整的运维诊断工具用于发现集群热点机器、热点作业。在 Tesla 页面上能快速找到集群的热点机器，通过“机器分析”工具查看这台机器上实时跑的任务，并且能定位到相应的业务和用户。通过“作业分析”工具能自动诊断异常，结合作业的优先级，实现了一键负载均衡、启停、续跑等运维操作。

通过这些保障设施，双 11 当天，即使在发生交换机硬件故障的情况下，面向全球直播的媒体大屏业务并没有受到任何影响！

## 2.8 小结

拥有这些和其它诸多能力，Galaxy 已经具备了相当完善的实时计算能力，也提供了“一站式”的解决方案。今年双 11 当天，Galaxy 处理了 PB 级别数据，处理峰值达到了 1 亿事件每秒，平均处理延迟在毫秒级！除了双 11 媒体大屏，Galaxy 还支撑着阿里巴巴集团内外众多实时业务，包括数据运营、广告营销、搜索个性化、智能客服、物流调度、支付宝、聚划算等。

## 3 大数据计算服务-MaxCompute

每年双 11 都是阿里巴巴从最“前端”到最“后台”所有系统整条链路的一次大考。电商在线系统的浏览和消费产生了大量数据，其数据量是平常的数倍到数十倍。这些数据最终要流到阿里巴巴的大数据计算服务—MaxCompute 上来处理。

MaxCompute 承载了阿里巴巴集团所有的离线计算任务，是集团内部核心大数据平台。截止到目前支撑着每日百万级规模的作业，整个系统拥有数万台机器，单集群规模上万，存储已经到达了 EB 级别，每天有数千位活跃的工程师在平台上做数据处理。

面对如此多的海量数据，首先需要能够低成本的将数据存储下来。MaxCompute 依托背后的飞天分布式操作系统，将大量低成本 PC 服务管理起来。早在 2013 年，我们基于对业务增长速度的判断，发现系统的存储马上就要

“撞墙”了，集群的规模将要应付不了与日俱增的数据量。直到后来成立了 5k 项目组，对技术难点进行了攻坚，将单集群规模扩大到了 5000 台，阿里巴巴也成为了中国首个独立研发拥有大规模通用计算平台技术的公司。

实际上单集群规模到达上万台本身技术挑战非常大，因为规模上来以后对系统设计要求非常高，整个架构不能有单点。但是整个业务规模决定了 1 万台机器是不够的，因此 MaxCompute 抽象出来一个控制层，将分布在各个不同数据中心的多个计算集群统一管理，根据业务特点将不同的业务放在不同的计算集群中，通过跨集群复制，自动将数据在多个集群中同步，使得用户可以把计算引擎当成一个平台。

### 3.1 跨集群复制和全局调度

运行在 MaxCompute 上的业务种类非常多，各个业务部门之间数据也有着错综复杂的依赖关系。如果恰好数据不在同一个地域/机房中，那么就要进行数据的异地读写。比如分析支付宝的数据需要淘宝的数据，支付宝的数据和淘宝的数据并不在同一个机房集群，那就需要跨集群的去读（直读），或者将数据拷贝到本地再读（跨集群复制）。此外由于数据是会被更新的，比如淘宝的数据更新了，这个时候要求支付宝的作业能够读到最新版本的数据。生产任务有各自的基线时间，对处理时间有要求，不能由于互访数据导致任务延时太长。机房之间虽然有几十到上百 G 的直连网络专线，但其他生产业务也对网络带宽有需求，互访数据不能把带宽都占满，需要有网络流量控制。多个任务可能会访问同一份异地数据，再考虑带宽占用的限制，所以访问异地数据不能全部都通过直读异地数据来解决，有的异地数据需要在本地复制一份以供多次任务使用。

为了解决这个问题，MaxCompute 引入了跨集群复制和全局调度机制。MaxCompute 上所有的数据表和分区的元数据引入了版本号，当数据被更新时，其对应的计算集群版本号也会更新。版本更新后，新版本所在的计算集群的数据需要被复制到其他计算集群。但这个复制操作该何时发生，需要考虑多种因素，比如任务完成时效要求，多集群之间的带宽大小等。对这些因素进行全局分析，才能利用动态预先调整，远程读，复制等多种手段做到全局调度。但这一全局分析需要系统运行数据才能进行。MaxCompute 中的元数据、数据血缘关系的分

析，以及整个系统运行过程中产生的数据都会收集到元数据仓库，这样可以利用平台本身的数据分析能力来分析这些数据。这些数据被用来辅助 MaxCompute 平台的工程师做数据化运营，甚至用来帮助系统自身进行优化。

## 3.2 基于历史运行信息的优化

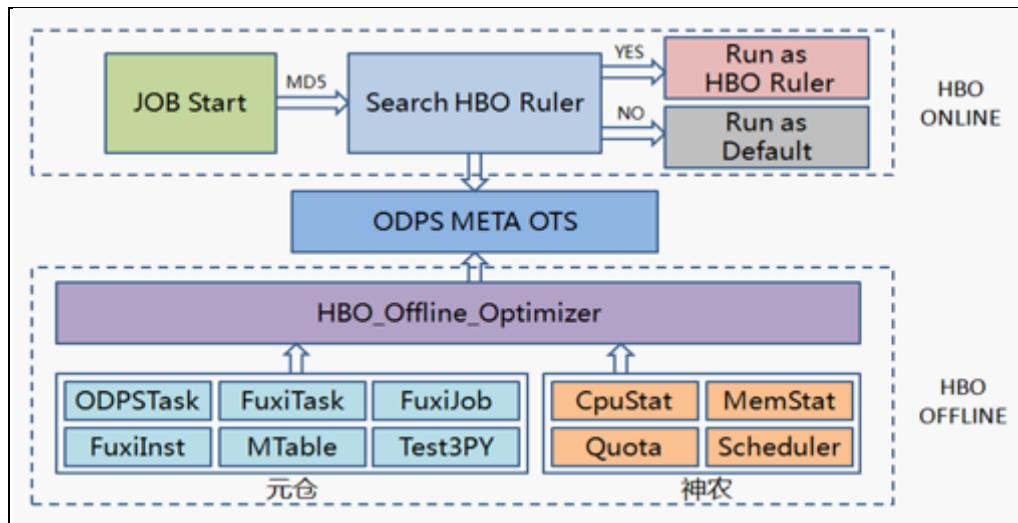
通过对每天运行的作业进行分析，我们发现大部分作业都是重复执行的。这是数据仓库中的一个典型的使用场景：每天产生的新数据被同一套数据处理任务批量重复执行。这样的场景带来了巨大的优化机会。首先每天运行的任务所占用的资源信息会被记录下来，比如运行时占用的 CPU、内存和运行时间。工程师新开发的作业在第一次运行时，申请的 CPU 和内存一般都会和实际占用的 CPU、内存有所差别。如果申请的大于实际占用的，会造成调度的时候为作业多留资源，造成资源浪费，即资源的利用率下降。如果申请的小于实际占用的，会造成一台机器上调度的作业超过了机器能够承载的负荷。这两种资源错配的后果都会降低系统使用效率。最理想的结果是作业申请的资源与实际使用的能够完全匹配。

HBO( History-ed Based Optimization) 基于历史运行信息的优化就是通过收集作业的历史运行记录，根据实际 CPU、内存占用来指导作业合理设置的一种优化手段。它是对集群资源分配的一种优化，概括起来就是根据：任务执行历史+集群状态信息+优化规则，得到最优的作业资源配置。

HBO 包含两部分工作：

- 在线部分(Online)：查找是否存在相应的 hbo 优化计划，如果有，则按照计划进行资源分配并执行
- 离线部分(Offline)：从元数据仓库和神农获取任务的历史执行记录，按照一定的策略生成 hbo 优化计划

下图为 HBO 的流程架构图：



正常情况下，这种基于历史的优化效果非常显著，因为作业总体数据量在天与天之间变化一般不会很大。但到了双 11，由于当天产生的数据量通常是前几天的数倍甚至数十倍，对于一些极限情况需要做特殊处理。比如作业 instance 数会因为处理的数据量增大同步增长而超过单个作业 instance 数量上限。依托 HBO 的工作，可以识别重复的作业、并且能够精准的对单个作业进行设置。利用这个能力，我们可以在节日前先对所有作业做一次分析，比如找出输入表在去年双 11 当天数据量显著增涨的作业，或者找出 instance 数量已经快要接近极限的作业，将他们单个 instance 处理的数据量设大，顺利度过双 11 的考验。以同样的手法可以指导制作针对双 11 的预案，比如调整 CPU、内存的设置、提前发现数据倾斜等等。

# 6.3 突破传统，4k 大屏的沉浸式体验

作者：彦川、小丛

## 前言

能够在 4K 的页面上表演，对设计师和前端开发来说，即是机会也是挑战，我们可以有更大的空间设计宏观的场景，炫酷的转场，让观众感受影院式视觉体验，但是，又必须面对因为画布变大带来的性能问题，以及绞尽脑汁实现很多天马行空的想法。下面是这次双 11 媒体大屏开发中我们的一些设计和思路。

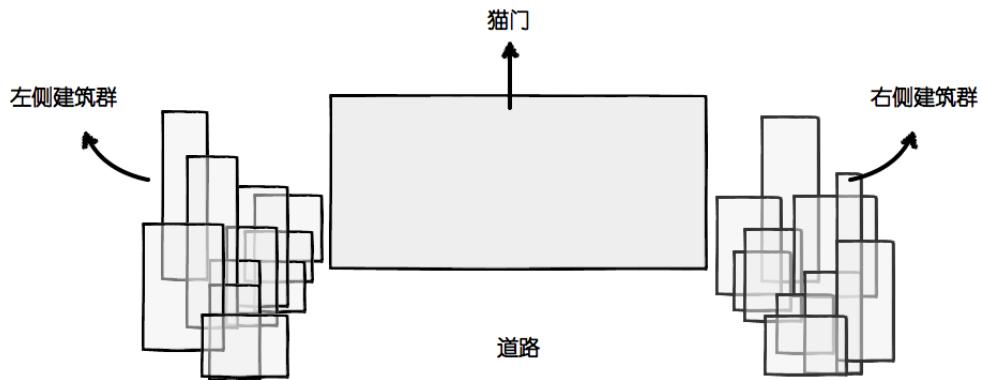
## 1 3D 动感跑道

当逍遥子零点倒数 5 , 4 , 3 , 2 , 1 , 0 ! 激昂音乐奏起，媒体中心大屏幕跳跃出一个动感十足的页面，黄橙橙的 GMV 数字蹭蹭往上长，跳跃的翻牌器下有个不断向前延伸的跑道，两旁错落有致的建筑群，顶部有一簇簇如烟花般四面飞散的点状射线...



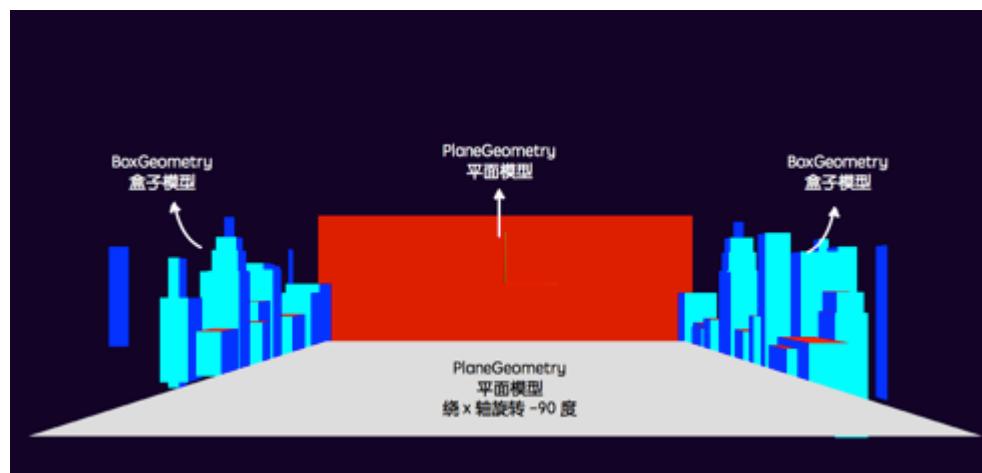
### 1.1 场景设计

场景包括三个部分：红色的猫门，左右两旁的建筑群，底下的炫彩跑道。基本结构如下：

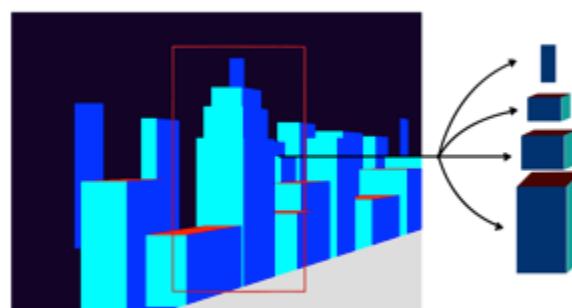


## 1.2 构建 3D 模型

基于 WebGL 技术 , 以其中的几何体构建出一个个 3D 模型 , 为了使得建筑群错落有致 , 逐一调整每幢建筑的位置 ( 调整 position 中的 x,y,z 值 )



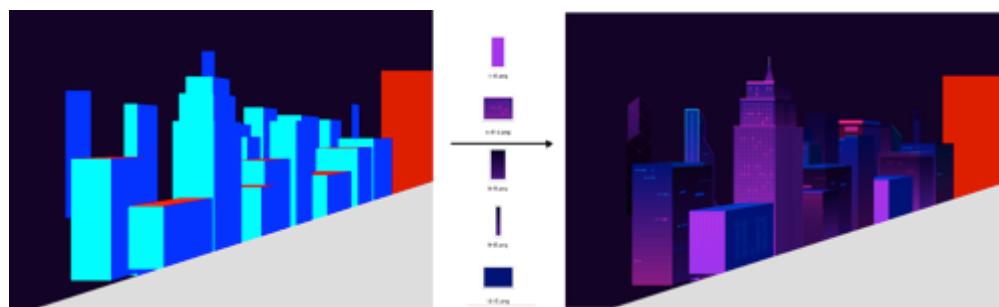
对于不规则的建筑模型 , 采用多种几何体拼凑的形式



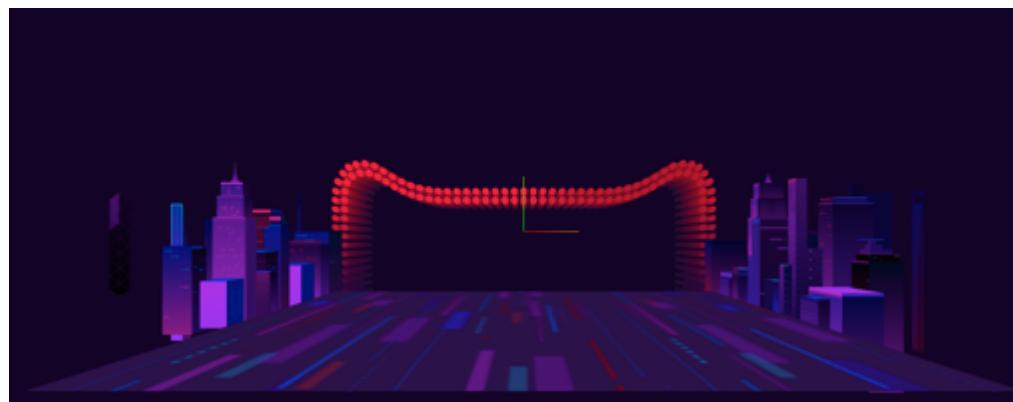
## 1.3 给建筑 “穿上衣服”

现在的建筑还是死气沉沉的几何体 , 给每幢建筑渲染不同的图片材质后 , 便

灵动起来了。



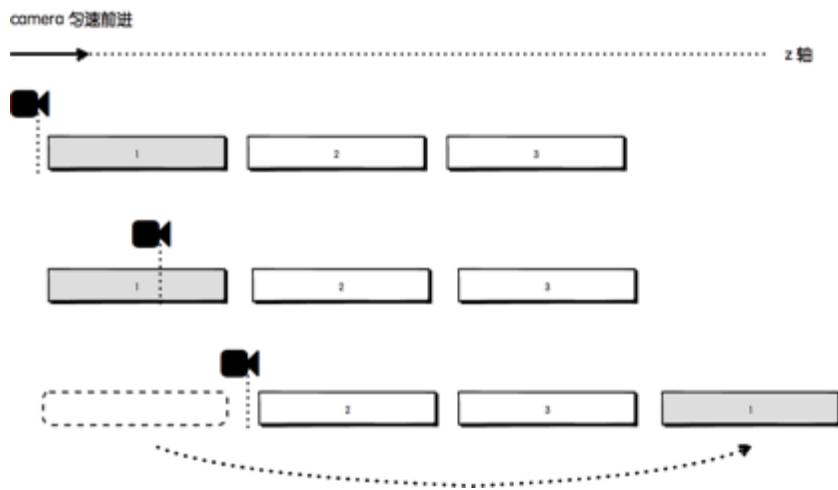
最后给猫门和跑道都渲染上材质，一个完整的场景便构建出来了。



## 1.4 “跑起来吧！”

整体的动效设计是：沿着炫彩跑道不断向前冲刺，迎面而来的建筑群给观众一种强烈的视觉冲击之感。开发思路：

- (1) 猫门 + 跑道 + 左右建筑为一个整体，复制两组，沿着 z 轴依次摆放；
- (2) 摄像机的 z 值在不断前进，即不断累加；
- (3) 当摄像机的 z 值超过一组模型的 z 值时候，重置该组的 z 值，移至末端。以此往复。

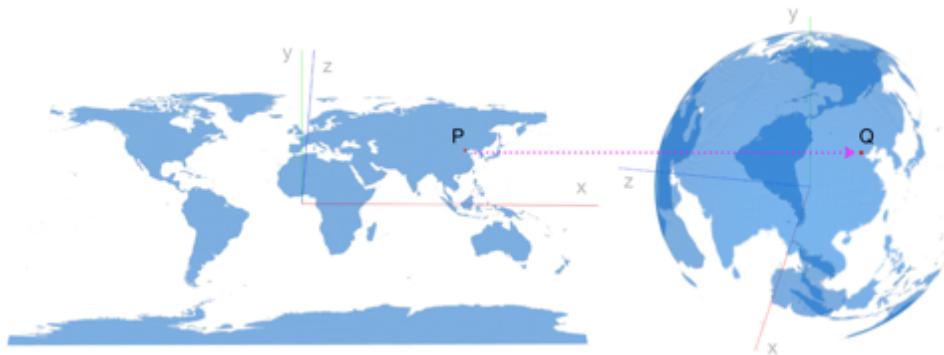


## 2 可形变的地图

地图是展现大数据可视化的最好背景，没有之一。一般情况下，可视化地图场景分为 3D 球面和 2.5D 平面两种，以前两者往往是隔绝开来的，根据展示数据的维度选择一种，比如讲全球贸易时，用球面场景的页面，讲国家贸易时，切换到 2.5D 平面场景的页面。多页面切换严重影响了阅读数据的连贯性，不符合讲故事的风格。因此，我们利用 WebGL 设计了一个可形变的地图，把场景集中在一个页面展示。



在 3D 的世界里，首先有的是点，其次是线和面，不管是平面地图还是球面地图，它们的本质都是点的集合，**如果控制了点，就控制了 3D 世界的变**化。我们的变形地图是一个由无数点组成的对象，通过 GLSL(OpenGL Shading Language)语言，我们可以动态改变地图上点的位置。



举个简单的例子，在平面地图中，北京在世界坐标系中的位置是 P，在球形地图中，北京在相同世界坐标系中的位置是 Q，那么从平面到球面的形变过程，其实就是 P 点到 Q 点的移动，反之亦然。推广到所有组成平面的点，形变的本质就是所有点移动到目的地点的过程。我们分别对所有点的起点(平面点)和终点(球面点)进行线性插值，就可以得到每个点的形变路径，剩下的事情就是加个缓动函数，让点运动起来。

### 3 灵活的飞线

飞线代表着交易，一条飞线的飞出代表一笔订单的完成，给人非常直观的可视化体验，为了从不同维度展现交易，飞线的场景也很多，比如：

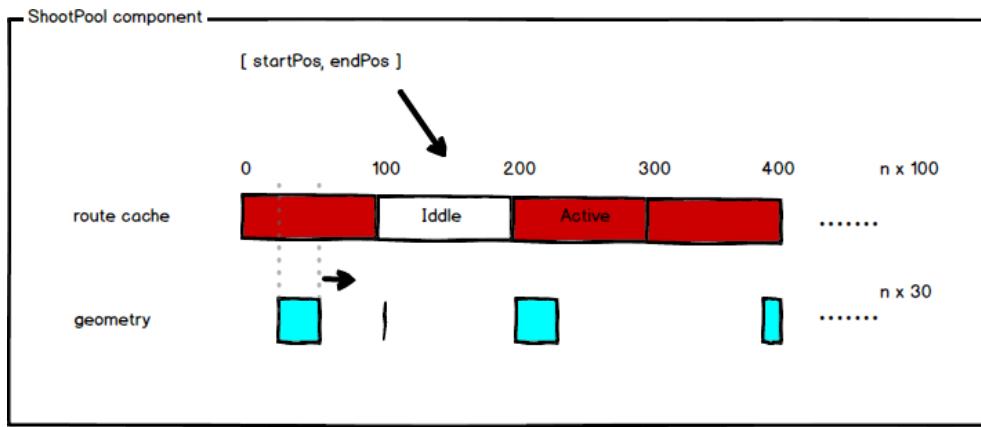
球面场景



平面场景



为了保证 4K 页面下有足够的帧率，以及灵活应对球面与平面的切换，我们对飞线进行了重新设计。首先是把飞线组件从业务中解耦出来，飞线组件不管数据的更新、缓存，只是单纯接受发生交易的起终点经纬度以及展示形式（球面或者平面）等参数，然后绘制一条飞行轨迹；其次，对飞线组件进行了优化，飞线的实现方式很多，可以是点集合不动，让颜色在点集合上流动，或者是事先给点集合赋值颜色，然后移动点，不同的实现方式，性能也会不同。理论和实践得出：所有飞线只用一个几何体，且这个几何体的点数尽可能少对帧率提高有极大的帮助。



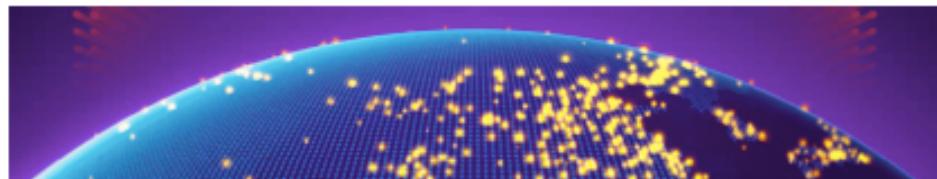
上图是我们设计的飞线组件 ShootPool，发射一条飞线的时候，仅仅需要知道飞线的起点 startPos，和 endPos，ShootPool 的工作机制如下：

- (1) 假设飞线组件最大支持  $n$  条飞线，每条飞线长度为 30 个点，飞行路径为 100 个点，那么几何体的点数为  $n \times 30$ ，路径总长  $n \times 100$ 。
- (2) 当外部调用了 ShootPool 的 run 方法时，组件会根据当前展示形式计算出这条飞线的整个飞行线路，如果这个时候飞线池是不满的，有路径（路径和飞线是对应的）处于 Idle 状态 如图中 100 ~ 199 的点 则将这条 Idle 路径更新，状态改变为 Active。
- (3) 找到当前路径对应的几何体的片段，通过缓动函数，使得当前片段（一条飞线）沿着路径移动，产生飞行效果。
- (4) 当飞线末端到达路径的末端时，设置当前路径为 Idle，等待下一条飞线。

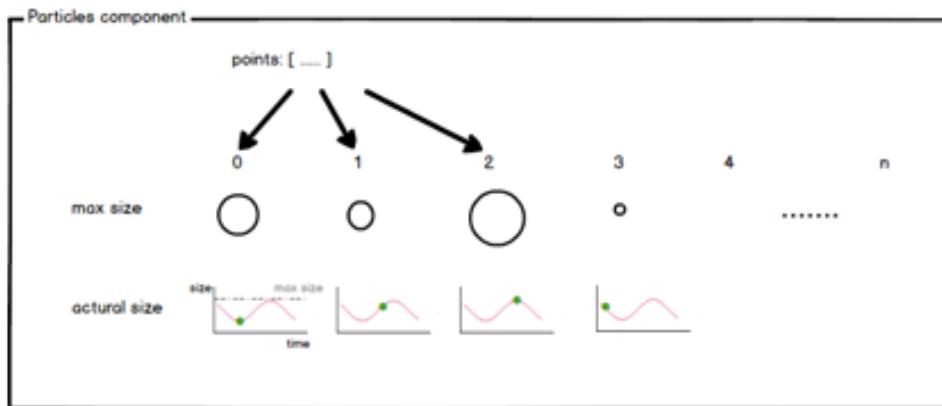
这个场景可以理解为飞机延着航线飞行，每条航线配备一架飞机，因为有钱任性，只要有飞机空着，就可以重新制定航线起飞。

新飞线组件的优势在于：只有一个几何体，几何体上没有冗余的点，可以保证飞线的性能是最优的，另外飞线的颜色和路径都是起飞前重新设定的，面对变化场景时，显得非常灵活。

## 4 会呼吸的热力点



地图上的热力点大小代表着地区的交易量，同时，为了营造一种买买买的热闹气氛，每个点都是在呼吸的。对于重复简单的粒子，一般都采用 WebGL 的粒子系统来实现，因为每个点只是一张图片，而不单独需要几何体，所以，可以展现非常多的点。为了实现呼吸效果，我们依然采用 WebGL 和 GLSL 配合的方式来实现，这样就可以用 GLSL 来动态改变点的大小，思路和飞线的实现方式类似。



`Particles` 组件接收 `points` 数组作为参数，`points` 的每个节点信息用来确定热力点在场景中的位置，以及每个点的 `max size`。组件会根据 `max size` 和时间轴计算出当前热力点的大小（也可以是透明度），然后传递给 GLSL，这样，GPU 在绘制点的时候，就有大小的区分，随着时间推移，大小发生周期性变化（比如 `sin` 函数），从而实现呼吸的效果。

## 5 粒子风格的地球



今年大屏地图的风格是粒子风格，粒子组成的地球，高亮的中国板块，以及高亮的省份，要实现上图的效果，我们用了贴图。为什么不用真正的粒子？如果每个点都用几何体来实现，GPU 需要绘制的顶点数  $n * m$ ， $n$  为粒子数，数量级为万， $m$  为每个几何体的顶点个数，一般为几十个，性能堪忧。如果利用 WebGL 的粒子系统，性能上应该是可以的，GPU 绘制的顶点个数为  $n * 1$ ，可以用 5w 左右的点将世界地图拼出来，见下图。但是粒子的本质是只是一张图片，不是由几何体构成的，所以，我们改变不了每个粒子永远看着摄像机的尴尬，另外，如果设计师如果觉得粒子大小还要小  $x$  倍，那么粒子点数就需要  $5x$  万个，是不是很提心吊胆。

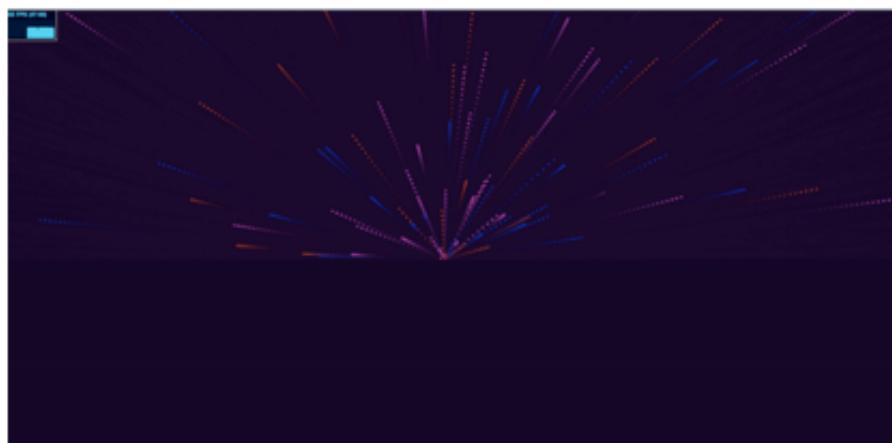


利用贴图的好处在于，可以最大限度地还原设计师的效果，同时几乎不影响性能，性价比相当高。在区域屏省份特写轮换以及国家屏切换时，需要动态改变高亮的区域。要实现这一功能，我们事先准备好一张高亮的地图，通过省份或者国家轮廓的经纬度，将仅包含我们需要的区域用 canvas 绘制出来，制做成新

贴图，然后替换掉老的贴图即可，记住老的贴图必须手动释放内存，不然 GPU 内存会溢出哦。

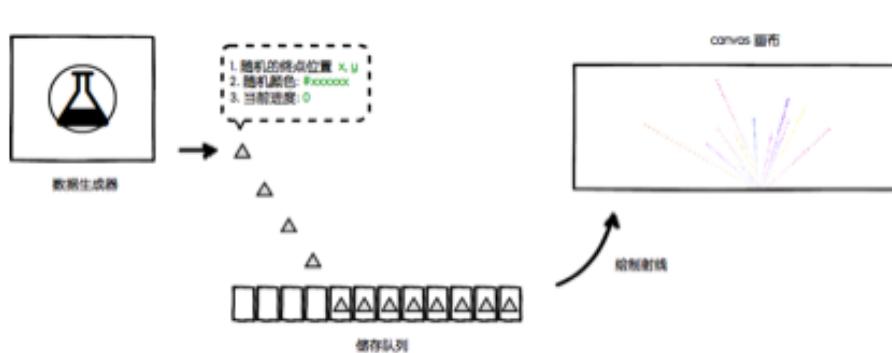
## 6 如烟花般四面飞散点状射线

在这个热闹非凡的节日之中，怎能没有烟花流线来助兴呢？



开发思路：

- (1) 初始化一张与屏幕等宽的 canvas 画布；
- (2) 以中心点为坐标轴原心，射线由原心发出，向四周扩散；
- (3) 不断随机生成终点位置，当射线到达终点位置后，销毁该射线；



射线是由一系列透明度递减的圆点组成。

其中使用了一个小技巧：

- (1) 把画笔的透明度设置为小于 1 , 如 0.85 ; ctx.globalAlpha = 0.85;
- (2) 创建一张虚拟( "虚拟"是指仅在 js 里使用它 ,而不放进 document 中 ) 的 canvas 画布；



每一帧的动画渲染步骤如下：

- (1) 在虚拟画布中存储上一帧的动画状态 ( ctx.drawImage );
- (2) 清空当前画布 ( ctx.clearRect );
- (3) 绘制新的圆点 ( ctx.arc );
- (4) 把虚拟画布中的内容复制回当前画布 ( ctx.drawImage ).

注意此时由于设置了画笔透明度 ,重新绘制在画布中的图像整体透明度将降低 0.15 度。

## 总结

今年的媒体大屏有两个特点：屏多，多达 20 多张屏，屏大，基本所有的屏都是 4K，所以挑战还蛮大的，为了提高开发效率，大部分动效的实现方式遵循简单、高效的原则，所幸视觉还原效果还是符合预期的。文中涉及 3D 可视化技术比较多，其实还有很多 2D 的图表，比如战略屏大地球上的 gmv 柱子也是花费了设计师和开发的很多心思，篇幅有限就不一一叙述。今年是团队第一次参与双 11 媒体大屏的开发，以后的路希望越走越长。

# 第七章 人工智能

# 7.1 基于深度强化学习与自适应在线 学习的搜索和推荐算法研究

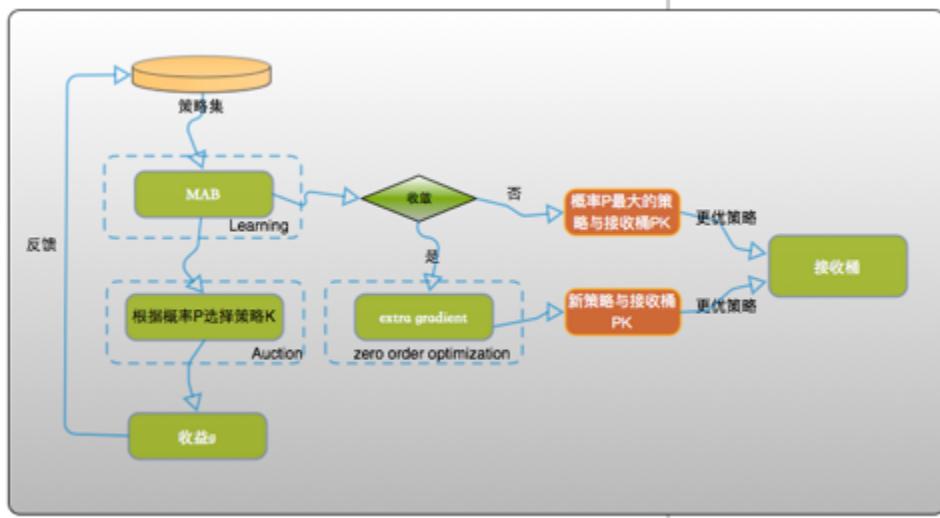
作者：灵培、霹雳、哲予

## 1 搜索算法研究与实践

### 1.1 背景

淘宝的搜索引擎涉及对上亿商品的毫秒级处理响应，而淘宝的用户不仅数量巨大，其行为特点以及对商品的偏好也具有丰富性和多样性。因此，要让搜索引擎对不同特点的用户作出针对性的排序，并以此带动搜索引擎的成交提升，是一个极具挑战性的问题。传统的 Learning to Rank ( LTR ) 方法主要是在商品维度进行学习，根据商品的点击、成交数据构造学习样本，回归出排序权重。LTR 学习的是当前线上已经展示出来商品排序的现象，对已出现的结果集合最好的排序效果，受到了本身排序策略的影响，我们有大量的样本是不可见的，所以 LTR 模型从某种意义上说是解释了过去现象，并不一定真正全局最优的。针对这个问题，有两类的方法，其中一类尝试在离线训练中解决 online 和 offline 不一致的问题，衍生出 Counterfactual Machine Learning 的领域。另外一类就是在线 trial-and-error 进行学习，如 Bandit Learning 和 Reinforcement Learning。

在此之前我们尝试了用多臂老虎机模型 ( Multi-Armed Bandit , MAB ) 来根据用户反馈学习排序策略，结合 exploration 与 exploitation ，收到了较好的效果。



后来更进一步，在原来的基础上引入状态的概念，用马尔可夫决策过程对商品搜索排序问题进行建模，并尝试用深度强化学习的方法来对搜索引擎的排序策略进行实时调控。

实际上，如果把搜索引擎看作智能体（Agent）、把用户看做环境（Environment），则商品的搜索问题可以被视为典型的顺序决策问题。Agent每一次排序策略的选择可以看成一次试错（Trial-and-Error），把用户的反馈，点击成交等作为从环境获得的奖赏。在这种反复不断地试错过程中，Agent将逐步学习到最优的排序策略，最大化累计奖赏。而这种在与环境交互的过程中进行试错的学习，正是强化学习（Reinforcement Learning，RL）的根本思想。

本文接下来的内容将对具体的方案进行详细介绍。

## 1.2 问题建模

马尔可夫决策过程（Markov Decision Process，MDP）是强化学习的最基本理论模型。一般地，MDP可以由一个四元组 $\langle S, A, R, T \rangle$ 表示：(1)  $S$  为状态空间（State Space）；(2)  $A$  为动作空间（Action Space）；(3)  $R: S \times A \times S \rightarrow \mathbb{R}$  为奖赏函数；(4)  $T: S \times A \times S \rightarrow [0,1]$  为环境状态转移函数（State Transition Function）。

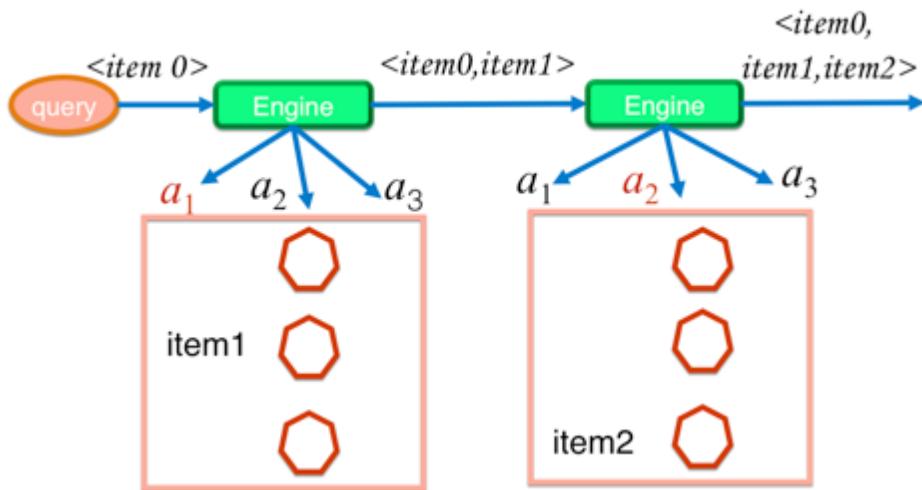
我们的最终目标是用强化学习进行商品搜索排序策略的学习，在实现的过程中，我们一步一步完成了从简单问题到复杂问题的过渡，包括：

1. 基于值表（Tabular）强化学习方法的商品价格档  $T$  变换控制（离散状态、离散动作问题）；

2. 基于值表( Tabular )强化学习方法的商品展示比例控制( 离散状态、离散动作问题 );
3. 基于强化学习值函数估计 ( Value Function Approximation ) 的商品排序策略调控 ( 连续状态、离散动作问题 );
4. 基于强化学习策略估计 ( Policy Approximation ) 的商品排序策略调控 ( 连续状态、连续动作问题 ).

### 1.2.1 状态定义

假设用户在搜索的过程中倾向于点击他感兴趣的的商品，并且较少点击他不感兴趣的的商品。基于这个假设，我们将用户的历史点击行为作为抽取状态特征的数据来源。具体地，在每一个 PV 请求发生时，我们把用户在最近一段时间内点击的商品的特征作为当前 Agent 感知到的状态。当然，在不同的问题中，状态的表示方法会有所不同。例如，在值表强化学习方法中，状态为可枚举的离散变量；在值函数估计和策略估计方法中，状态则表示为特征向量。



### 1.2.2 奖赏函数定义

Agent 给出商品排序，用户根据排序的结果进行的浏览、商品点击或购买等行为都可以看成对 Agent 的排序策略的直接反馈。在第四章中，我们将利用奖赏塑形 ( Reward Shaping ) 方法对奖赏函数的表达进行丰富，提高不同排序策略在反馈信号上的区分度。

## 1.3 算法设计

由于篇幅有限，我们仅对强化学习在搜索中的使用给出 2 个实例。

### 1. Tabular 方法

我们在排序中要引入价格的因素来影响最终展示的价格，若以 GMV 为目标，则简单可以表示为  $cvr * price$ ，同时我们又想控制价格的作用程度，所以目标稍作修改： $cvr * price^t$ ，加入一个变量  $t$  来控制价格的影响。这个  $t$  值的范围很有限，可以用 MAB 或 CMAB 来找到最优解。

我们用强化学习的视角来对这个问题进行抽象，把用户前 2 次点击的商品价格档位（0~7，从低到高）作为状态。这个状态表示的是用户之前点击商品的价格偏好，如果两次都点击 1 档商品，说明用户偏好低价商品，很有可能接下来用户只对低价商品感兴趣，如果这个状态转移分布是稳定的（stationary），那么一个统计模型就可以描述这种规律。而实际上，用户的行为是受我们排序模型的影响的，用户点击 1 档商品也可能是因为当前的排序策略只给用户展示了 1 档商品，并不一定是用户的本质需求。在接下来用户的搜索过程中，我们可以有的选择 1 是只出 1 档商品让用户的需求快速收敛，选择 2 是投放一些附近档位的商品供用户选择，如果用户选择了其他档位的商品，进行了状态的转移，就可能找到一个更好的路径，最终的收益和我们所有的过程中的投放策略都相关。从每个时间点上看，策略可能不是最优的，但全局上可能是最优的。

具体地，当用户进行了搜索后，根据用户的状态  $s$ ，和 Q 表（下图）进行一个 epsilon-greedy 的投放，选择一个动作  $a$ （上文中的价格指数  $t$ ），执行这个  $a$  的排序结果展示给用户，并记录下这次的状态  $s$  与动作  $a$ ，以及用户对这次搜索结果的反馈  $r$ ，从用户的点击与否的反馈，再对 Q 表进行更新。

$Q(s,a)$	a1	a2	...	a10
$s_1$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...	$Q(s_1,a_{10})$
$s_2$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...	$Q(s_2,a_{10})$
...	...	...	...	...
$s_{64}$	$Q(s_{64},a_1)$	$Q(s_{64},a_2)$	...	$Q(s_{64},a_{10})$

根据 Q-Learning 公式进行权重更新。

$$Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha)Q_t(s_t, a_t) + \alpha \left[ r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') \right]$$

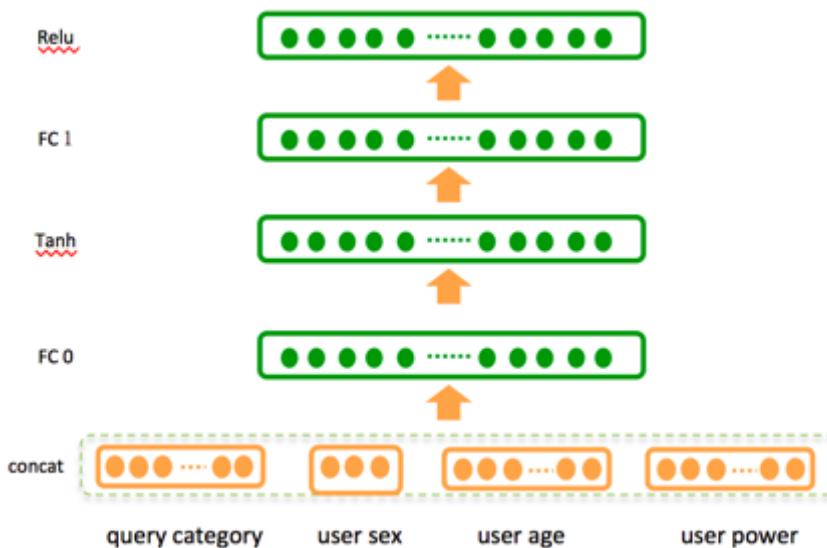
接下来，由于用户点击了某商品，他的状态发生了转移，就找到对应的状态继续进行 epsilon-greedy 的投放。再继续进行学习，直到收敛。

## 2. DDPG 方法

例如一个线性排序模型， $f(x|w) = w^T x, x \in R^m$ ， $x$  是  $m$  维的特征向量，我们学习每个用户状态  $s$  的最优参数  $w$ ，即  $\pi(s) \rightarrow w^*$ 。这种假设需要使用策略估计的方法。策略估计（Policy Approximation）方法是解决连续状态/动作空间问题的有效方法之一。其主要思想是用参数化的函数对策略进行表达，通过优化参数来完成策略的学习。通常，这种参数化的策略函数被称为 Actor。假设我们一共调控  $m$  ( $m \geq 0$ ) 个维度的排序权重，对于任意状态  $s \in S$ ，Actor 对应的输出为

$$\mu_\theta(s) = (\mu_\theta^1(s), \mu_\theta^2(s), \dots, \mu_\theta^m(s))$$

其中， $\theta$  为 Actor 的参数，对于任意  $i$  ( $1 \leq i \leq m$ )， $\mu_\theta^i(s)$  是关于状态的一个函数，代表第  $i$  维的排序权重分，其形式可根据实际情况而定，我们的方案采用深度神经网络作为 Actor 函数。这种方式在不同的状态之间可以通过神经网络来共享一些参数权重。



强化学习的目标是最大化任意状态  $s$  上的长期累积奖赏，根据策略梯度定理，

Actor 函数的参数 $\theta$ 的更新公式可以写为

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}$$

其中， $\nabla_\theta \mu_\theta(s)$ 为 Actor 神经网络在状态 $s$ 上关于 $\theta$ 的梯度， $Q^\mu(s, a)$ 为状态动作对 ( State-Action Pair )  $(s, a)$ 的长期累积奖赏。因为 $s$ 和 $a$ 都是连续的数值，我们采用深度神经网络作为估计器对 $Q^\mu(s, a)$ 进行学习，具体的学习算法可参考深度 Q 学习算法 DQN [1]。

## 1.4 奖赏塑形

我们最初采用的奖赏函数仅基于用户在每一个 PV 中的点击、成交行为反馈来构建。然而，在淘宝主搜这种大规模应用的场景中，我们较难在短时间内观察到不同的排序策略在点击和成交这样的宏观指标上的差别。因此，长期累积奖赏关于不同学习参数的梯度并无明显区别，导致学习算法收敛缓慢。因此，我们有必要在奖赏函数中引入更多的信息，增大不同动作的区分度。

在进行强化学习方案的同时，我们用 Pointwise LTR 进行了一些对比实验，发现 Pointwise LTR 这种直接在商品特征上进行学习的方式在求取策略梯度的时候，能够将不同排序策略更为显著地区分开。参照这个思路，我们将商品的一些属性特征加入到奖赏函数的定义中，通过奖赏塑形 ( Reward Shaping ) 的方法[2, 3]丰富其包含的信息量。

奖赏塑形的思想是在原有的奖赏函数中引入一些先验的知识，加速强化学习算法的收敛。简单地，我们可以将“在状态 $s$ 上选择动作 $a$ ，并转移到状态 $s'$ ”的奖赏值定义为

$$R(s, a, s') = R_0(s, a, s') + \Phi(s)$$

其中， $R_0(s, a, s')$ 为原始定义的奖赏函数， $\Phi(s)$ 为包含先验知识的函数，也被称为势函数 ( Potential Function )。我们可以把势函数 $\Phi(s)$ 理解学习过程中的子目标 ( Local Objective )。根据上面的讨论，我们把每个状态对应 PV 的商品信息纳入 Reward 的定义中，将势函数 $\Phi(s)$ 定义为

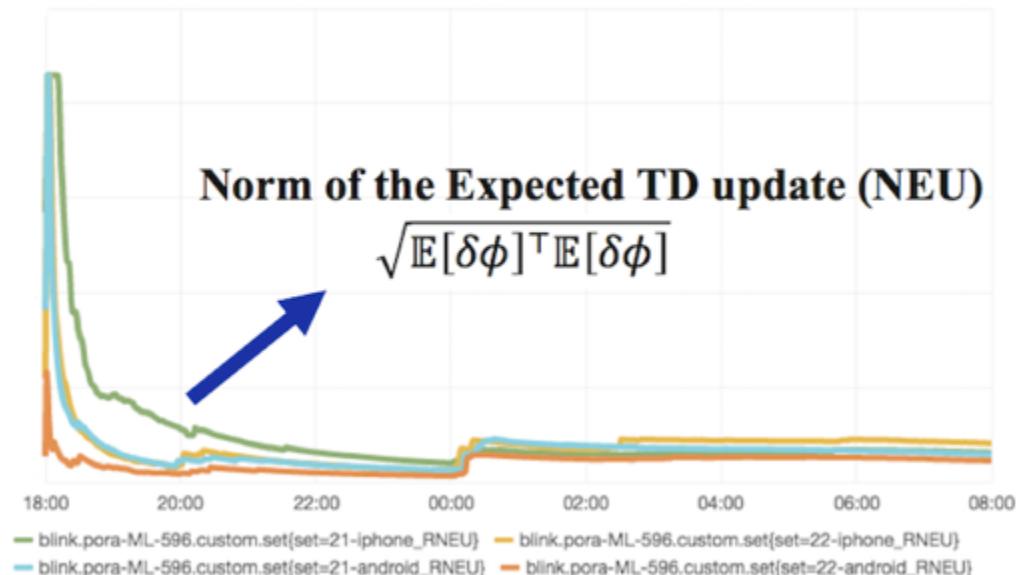
$$\Phi(s) = \sum_{i=1}^K \mathbb{L}(i|\mu_\theta(s))$$

其中， $K$ 为状态 $s$ 对应 PV 中商品的个数， $i$ 表示的第 $i$ 个商品， $\mu_\theta(s)$ 为 Agent 在状态 $s$ 执行的动作， $\mathbb{L}(i|\mu_\theta(s))$ 表示排序策略为 $\mu_\theta$ 时商品的点击 ( 或成交 ) 的似然 ( Likelihood )。因此， $\Phi(s)$ 也就表示在状态 $s$ 上执行动作 $\mu_\theta(s)$ 时，PV 中所有

商品能够被点击（或购买）的似然概率之和。

## 1.5 实验效果

在双 11 期间，我们在无线搜索排序的 21 和 22 号桶对强化学习方案进行了测试。下图展示了我们的算法在学习的过程中的误差 (RNEU) 变化情况，截取的时间范围为 11 月 10 日 18:00 到 11 月 11 日 8:00。



可以看到，从 11 月 10 日 18:00 启动开始，每个桶上的 RNEU 开始逐渐下降。到当天 20:00 之后，下降趋势变得比较缓和，说明学习算法在逐步往最优策略进行逼近。但过了 11 月 11 日 0 点之后，每个桶对应的 RNEU 指标都出现了陡然上升的情况，这是因为 0 点前后用户的行为发生了急剧变化，导致线上数据分布在 0 点以后与 0 点之前产生较大差别。相应地，学习算法获取到新的 reward 信号之后，也会做出适应性地调整。

## 2 推荐算法研究与实践

### 2.1 背景介绍

双 11 主会场是一个很复杂的推荐场景。从推荐的业务形式上看，双 11 主

会场分为三层：分别是楼层、坑位以及具体素材图的推荐。2016 年的双 11 主会场在整体的组织形式上与去年的双 11 主会场类似，但具体业务的构成及组织有较大的不同。首先，可推荐的楼层多于十层，我们需从中挑选数层进行展示，并有可能根据时间段和业务的需求进行调整。因此，展现形式的多变对模型的日志特征学习造成了一定的干扰。其次，坑位的构成为三种会场入口：第一行是行业会场，第二行对应店铺会场，第三行对应是标签会场。最后，在楼层以及坑位都确定之后，我们需要每个的坑位入口上选择具体的素材。2016 年双 11 主会场的素材有两种不同的展现形式，分别是双素材图以及单素材图。双素材图模式能提升用户的点击欲望，增强视觉感官冲击力，但也会对用户的真实点击行为数据造成一定程度的干扰或噪声，甚至对排序的模型产生比较大的偏置。



由于 2016 年双 11 首图宝贝素材总量在百万张且坑位数上百，我们会根据楼层的次序对参与打分的候选集进行配额，根据楼层的实时点击率分配楼层的打分量。在各类业务以及填坑逻辑及调控流量的限制下，推荐结果并不一定能按照原有的打分高低进行展示。因此，我们需要考虑打分宝贝数与工程实现上的平衡关系。由于主会场的 QPS 高达数万，一味地增大打分量是不可取的。为了解决这一问题，我们在初选的 match 召回方式上做了大量的努力，如提升用户的多重兴趣覆盖、增大有效的候选宝贝。

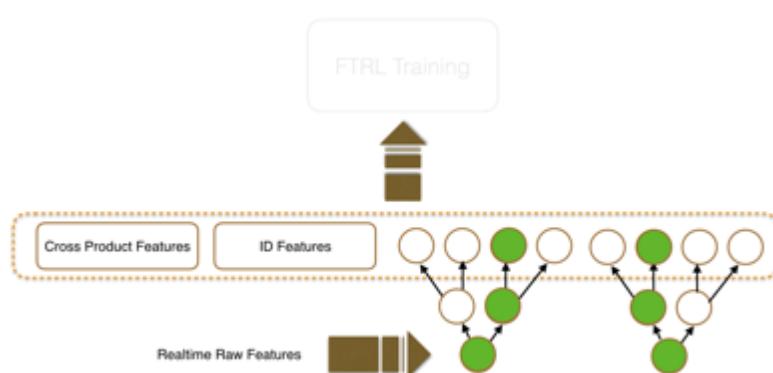
根据在 2015 双 11 的一些经验并结合 2016 年双 11 前期的系统压测情况，在 2016 年双 11 主会场我们采用了素材模型驱动的模式。从个性化推荐算法的

角度来说，我们在 2016 年双 11 主会场尝试了多种新颖的排序模型，并做了严格的效果对比。具体的排序模型涉及 LR、FTRL、GBDT+FTRL 融合模型以及 WIDE&DEEP 模型，同时为了克服 data drift 的波动在日常的首图场景还尝试了 Adaptive-Online-Learning 的算法，以及尝试了强化学习的思路。在后面的章节，会从算法层面逐一阐释。

## 2.2 算法模型

### 2.2.1 GBDT+FTRL 模型

采用非线性模型学习 intermediate feature，作为 ID feature 和 cross feature 的补充，最终输入到线性 model 来做 CTR 预估，最早是由 Facebook 提出的，思路大致如下：采用 raw features（一般是统计类特征）训练出 GBDT 模型，获得的所有树的所有叶子节点就是它能够 generate 出来的特征空间，当每个样本点经过 GBDT 模型的每一个树时，会落到一个叶子节点，即产生了一个中间特征，所有这些中间特征会配合其他 ID 类特征以及人肉交叉的特征一起输入到 LR 模型来做 CTR 预估。显然，GBDT 模型很擅长发掘有区分度的特征，而从根到叶子节点的每一条路径体现了特征组合。对比手工的离散化和特征交叉，模型显然更擅长挖掘出复杂模式，获得更好的效果。我们通过 GBDT 来做特征挖掘，并最终与 FTRL 模型融合的方案如下图：



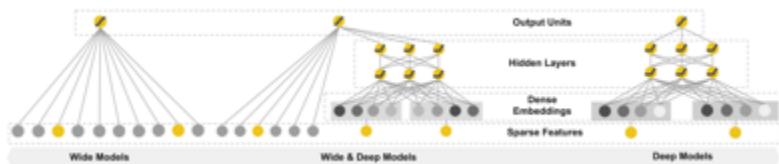
输入到 GBDT 的特征非常关键，这些特征决定了最终产出的中间特征是否有效。我们有一套灵活的特征生成流程，可以方便做各种维度的特征提取以及交叉统计。GBDT+FTRL 中主要用到的特征包含两部分：第一部分是用户/宝贝 ID 与

对方泛化维度交叉统计的特征，包含各种基础行为的次数以及 CTR 等。

第二部分是来自于 match 阶段的一些连续类特征。推荐的 match 阶段负责粗选出一部分跟用户相关的 content，该过程中会有多个模型分出现，例如做 trigger selection 的 model 分，content 的最终 match score 等，这些分数来自于不同离线 model，最终作为 feature 在 online rank model 中，能获得非常好的 ensemble 效果。

## 2.2.2 Wide & Deep Learning 模型

借鉴 Google 今年在深度学习领域的论文《Wide & Deep Learning for Recommender Systems》中所提到的 Wide & Deep Learning 框架（以下简称 WDL），并将其结合基于搜索事业部自研的机器学习平台的在线学习技术，我们研发了一套适用于推荐业务的 WDL 模型算法。下文将会对这一技术进行详述。



WDL 模型的原理框架如上图所示：它将深度神经网络(DNN)网络和逻辑回归(Logistic Regression)模型并置在同一个网络中，并且将离散型特征(Categorical Feature)和连续型特征(Continuous Feature)有机地结合在一起。WDL 模型主要由 wide 侧和 deep 侧组成。Wide 侧通过特征交叉来学习特征间的共现，而 deep 侧通过将具有泛化能力的离散型特征进行特征嵌入(embedding)，和连续型特征一起作为深度神经网络的输入（可以认为是一种特殊的深度神经网络，在网络的最后一层加入了大量的 0/1 节点），从理论上来说，我们可以把 deep 侧看作传统矩阵分解(matrix factorization)的一种泛化实现，值得注意的是特征嵌入的函数是和网络中其他参数通过梯度反向传播共同学习得到。模型的预测值采用如下公式进行计算：

$$P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}_{wide}^T[\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^T a^{(l_f)} + b)$$

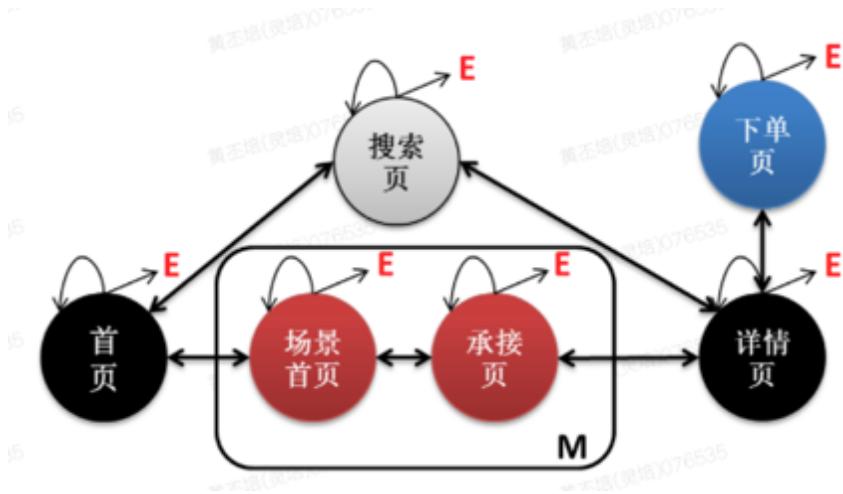
其中，wide 侧和 deep 侧合并在一起计算后验概率  $P(Y=1|x)$ ；在误差反向传播(Backpropagation)的计算过程中时，我们对两个方向同时进行计算。

### 2.2.3 Adaptive-Online-Learning ( 自适应在线学习 )

传统的在线学习模型没有一种机制很好的判断模型应该采用的多长时间的日志进行训练，目前业界的在线学习模型也都是通过经验值的方式来进行数据截断，自适应学习(adaptive learning)的最大优势就在于能够通过自我学习的方法适应业务的多变性。其实现原理在于保留下来每一个时刻开始到现在的数据学习到的模型，然后根据有效的评测指标，计算出各个模型的权重信息，并同时捕捉到数据分布快速变化波动的情况下的用户实时兴趣的细微差别，从而融合出一个最优的模型结果。

### 2.2.4 Reinforcement Learning ( 强化学习 )

相比对每个推荐场景单独进行个性化推荐的策略，基于强化学习框架 ( Reinforcement Learning ) 的推荐系统根据全链路的数据进行整合，同时响应多个异构场景的推荐请求。下图中我们对手机淘宝 ( 天猫 ) 客户端的数据 / 流量通路进行抽象：每个圆圈代表一个独立的手淘场景，E 代表用户在该场景随时离开，箭头代表流量可能地流动方向。



基于以上的数据通路图，我们可以很自然地将全链路多场景的推荐任务理解为一个连续的决策问题：作为一个智能决策者 ( agent )，推荐系统需要持续不断地决定应该为用户推荐怎样的内容 ( 比如，商品、店铺、品牌以及活动 )。强

化学习正是一种对智能决策者进行建模的最佳方式：通过对智能决策者短期状态的变化进行递归式建模，最终引导其渐进式地优化长期目标。

手淘上的推荐场景相当丰富，最具代表性的是一个页面以列表的形式同时推荐多个商品的场景。为了便于读者理解，我们首先介绍单个商品的推荐场景，之后再过渡到多商品的推荐场景。在单商品的推荐场景， $a$  对应的是单个商品。我们的目标是学习在状态  $s$  下采取动作  $a$  所能获得的累积奖励（的期望值）。我们用  $Q(s,a)$  来表示这一期望值。在这种情况下，我们只需要选择一种函数映射关系（如线性函数或神经网络）将  $s$  和  $a$  所代表的向量映射到标量上对目标函数  $Q(s,a)$  进行拟合。

$$Q(s, a) = \mathbb{E}[R|s, a] \quad (1)$$

我们把这一定义延伸到典型的多商品推荐场景。由于文章长度有限，我们下面介绍一种最简单的思路，即假设用户是否会点击单商品的决策是独立的。也就是说，假设用户如果喜欢商品 A，用户不会因为在同一推荐列表中见到了他更喜欢的商品 B 而放弃点击商品 A。在这一假设下，我们对展示每个商品所获得的累积奖励的计算也是独立的。通过一系列的推导，我们可以得到一个对状态  $s$  下商品  $i$  能得到的分数  $f(s,i)$  的递归定义。

$$f(s, i) = I(s_i)[r_i + \gamma \sum_{j \in a_i} f(s_i, j)] \quad (7)$$

通过等式(7)，我们可以迭代计算对无偏估计值进行求解。实际情况中用户必然会因为推荐商品的组合问题产生更复杂的行为，这样一来必然导致累积奖励独立计算的假设不成立。但以此为本，我们可以推导出基于更复杂假设下的计算累积奖励估计量的递归公式。

## 参考文献

- [1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A., Playing atari with deep reinforcement learning. CoRR abs/1312.5602, 2013.
- [2] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In Proceedings of the 16th International Conference on Machine Learning, pages 278–287, 1999
- [3] E. Wiewiora. Potential-based shaping and Q-value initialization are equivalent. Journal of Artificial Intelligence Research, 19(1):205–208, 2003

## 7.2 颠覆传统的电商智能助理 -阿里小蜜技术揭秘

作者：海青

### 1 双 11 的挑战与服务模式的转型

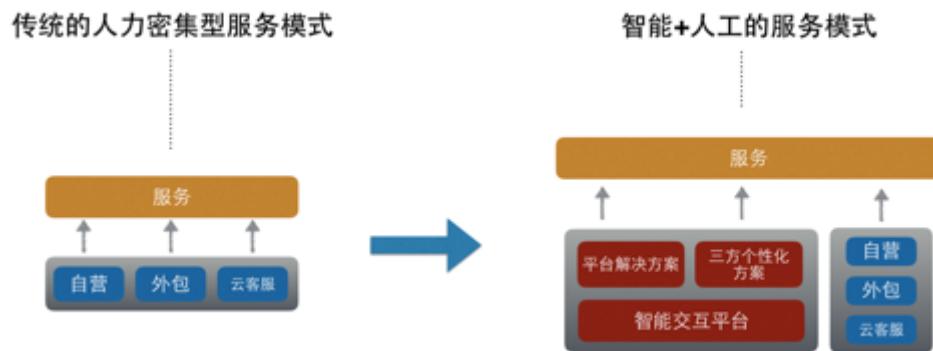
在全球人工智能领域不断发展的今天，包括 Google、Facebook、Microsoft、Amazon、Apple 等互联公司相继推出了自己的智能私人助理和机器人平台，智能人机交互成为各大公司在人工智能战场上激烈竞争的入口级领域。智能人机交互通过拟人化的交互体验逐步在智能客服、任务助理、智能家居、智能硬件、互动聊天等领域发挥巨大的作用和价值。

在 2015 年 7 月，我们阿里也推出了自己的智能私人助理-阿里小蜜，一个围绕着电子商务领域中的服务、导购以及任务助理为核心的智能人机交互产品。通过电子商务领域与智能人机交互领域的结合，提升传统电商领域的能效，带来传统服务行业模式的变化与体验的提升。

#### 1.1 智能人机交互带来服务行业模式的变化

传统的服务行业是个人力密集型的行业，就拿我们阿里巴巴双 11 狂欢节来说，无论是对于阿里直接对外的服务(消费者服务和商家服务)还是淘宝商家的服务都面临着当天服务量的巨大井喷，人力扩容成为每年阿里以及商家巨大的挑战。传统人力密集型服务模式(以自营客服、外包客服和云客服为主的服务模式)亟待被颠覆和改变，围绕着以阿里小蜜产品为核心，通过智能人机交互与人工服务相结合的模式才是未来真正的服务模式。机器通过智能化技术处理掉绝大部分的简单、重复等可识别处理的问题，对于解决不了的问题流向人工，让人提供更

有温度也更加专业的服务。通过智能+人工相结合的模式探索，在今年的双11期间，阿里小蜜整体智能服务量达到643万，其中智能解决率达到95%，智能服务在整个服务量(总服务量=智能服务量+在线人工服务量+电话服务量)占比也达到95%，成为了双11期间服务的绝对主力。



## 1.2 智能人机交互带来服务体验的提升

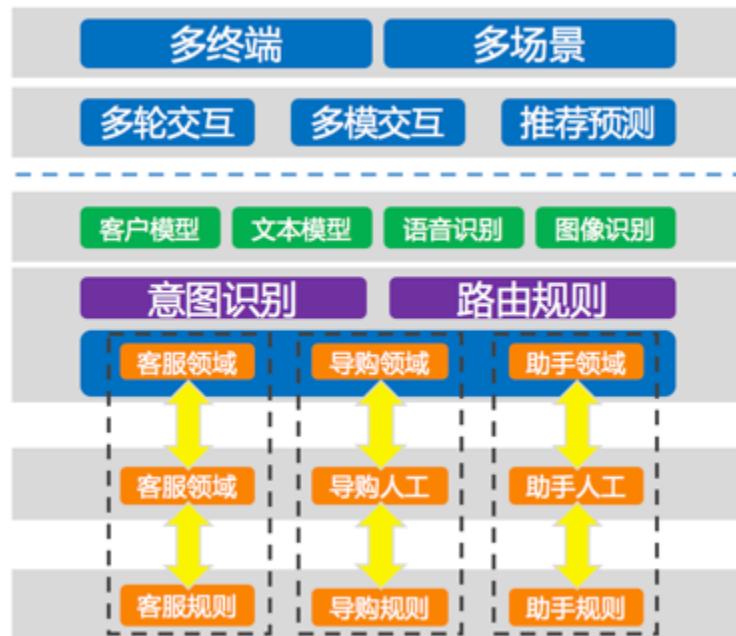
在体验维度，由于机器的运算速度远超于人，因此智能人机交互可以带来提升：智能交互相比人工可以达到急速的秒级体验，而人的服务通常需要一定时间的回复等待。并且在互动领域，智能人机交互也带来了一种新的模式和新的玩法，带来更多价值。

## 2 阿里小蜜及阿里小蜜平台介绍

阿里小蜜是电子商务领域的智能私人助理，基于阿里海量消费和商家数据，结合线上、线下的生活场景需求，以智能+人工的模式提供智能导购、服务、助理的拟人交互业务体验。

1. 在跨终端、多场景领域支持多轮交互、多模交互(文本、语音和图像)和问题推荐预测；
2. 支持多模型(文本模型、客户模型、语音识别和图像识别)识别客户意图；
3. 支持多领域识别和路由分流；

阿里小蜜整体体系图如下：



通过 1 年多阿里小蜜在阿里业务体系内的不断尝试和摸索，今年在电子商务生态圈范围内，在基于千牛的买卖家生态圈、基于钉钉的企业生态圈上，我们将阿里小蜜进行平台化开放，同时赋能给我们商家和企业用户。未来我们期望通过不断的领域数据和技术模型的积累，能够在阿里其他生态圈(例如：阿里云)逐步进行开放，赋能更多电子商务生态圈领域。

阿里小蜜平台结构图如下：



阿里小蜜在各个领域系统示例截图：

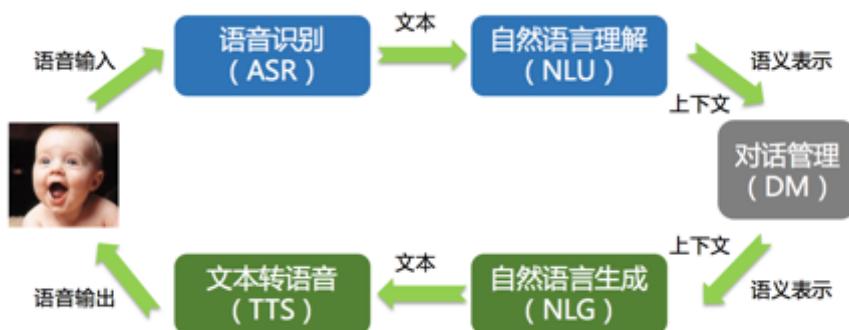


## 3 阿里小蜜技术实践

### 3.1 智能人机交互系统

智能人机交互系统，俗称：chatbot 系统或者 bot 系统，我们核心需要做的事情是理解人类的语言意思，进而给予合理的答案或者 Action。

人机交互基本流程如下：



其核心部分就是 NLU(自然语言理解)，通过对话系统处理后最后通过自然语言生成的方式给出答案。一段语言如何理解对于计算机来说是非常有难度的，例如：“苹果”这个词就具备至少两个含义，一个是水果属性的“苹果”，还有一个是知名互联网公司属性的“苹果”。因此在阿里小蜜这样在电子商务领域的场景中，我们先采用分领域分层分场景的方式进行架构抽象，然后再根据不同的分层和分场景采用不同的机器学习方法进行技术设计。首先我们将对话系统从分成两层：

- 1、意图识别层：识别语言的真实意图，将意图进行分类并进行意图属性抽取。意图决定了后续的领域识别流程，因此意图层是一个结合上下文数据模型与

领域数据模型不断对意图进行明确和推理的过程；

2、问答匹配层：对问题进行匹配识别及生成答案的过程。在阿里小蜜的对话体系中我们按照业务场景进行了3种典型问题类型的划分，并且依据3种类型会采用不同的匹配流程和方法：

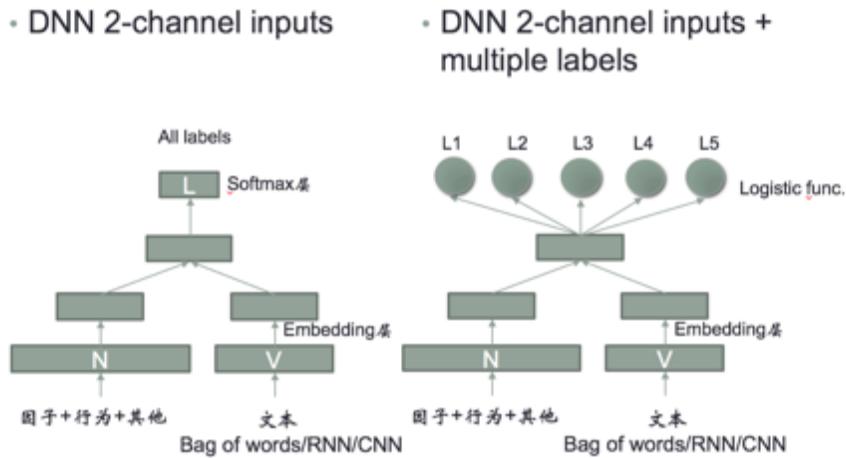
1. 问答型：例如“密码忘记怎么办？”→采用基于知识图谱构建+检索模型匹配方式
2. 任务型：例如“我想订一张明天从杭州到北京的机票”→意图决策+slots filling 的匹配方式
3. 语聊型：例如“我心情不好”→检索模型与Deep Learning相结合的方式

## 3.2 阿里小蜜意图识别的技术方案

通常在意图识别领域我们将其抽象成为机器学习中的分类问题来解决，在阿里小蜜的技术方案中除了传统的文本特征之外，考虑到本身在对话领域中存在语义意图不完整的情况，我们也加入了用实时、离线用户本身的行为及用户本身相关的特征，通过深度学习方案构建模型，对用户意图进行预测。如下图：



在基于深度学习的分类预测模型上，我们有两种具体的选型方案：一种是多分类模型，一种是二分类模型。多分类模型的优点是性能快，但是对于需要扩展分类领域是整个模型需要重新训练；而二分类模型的优点就是扩展领域场景时原来的模型都可以复用，可以平台进行扩展，缺点也很明显需要不断的进行二分，整体的性能上不如多分类好，因此在具体的场景和数据量上可以做不同的选型。整体的基本技术思路就是将行为因子与文本特征分别进行 Embedding 处理，通过向量叠加之后再进行多分类或者二分类处理。这里的文本特征维度可以选择通过传统的 bag of words 的方法，也可使用 Deep Learning 的方法进行向量化。具体如下图：



目前主流的智能匹配技术分为如下 4 种方法：

1. 基于模板匹配(Rule-Based)
2. 基于检索模型(Retrieval Model)
3. 基于统计机器翻译模型(SMT)
4. 基于深度学习模型(Deep Learning)

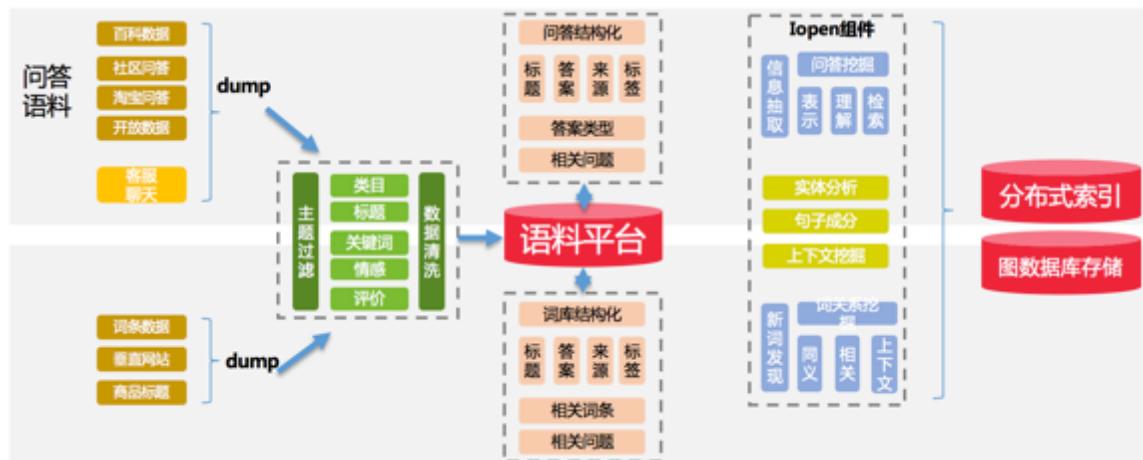
在阿里小蜜的技术场景下，我们采用了基于模板匹配，检索模型以及深度学习模型为基础的方法原型来进行分场景(问答型、任务型、语聊型)的会话系统构建。

### 问答型：基于知识图谱构建+检索模型匹配方式

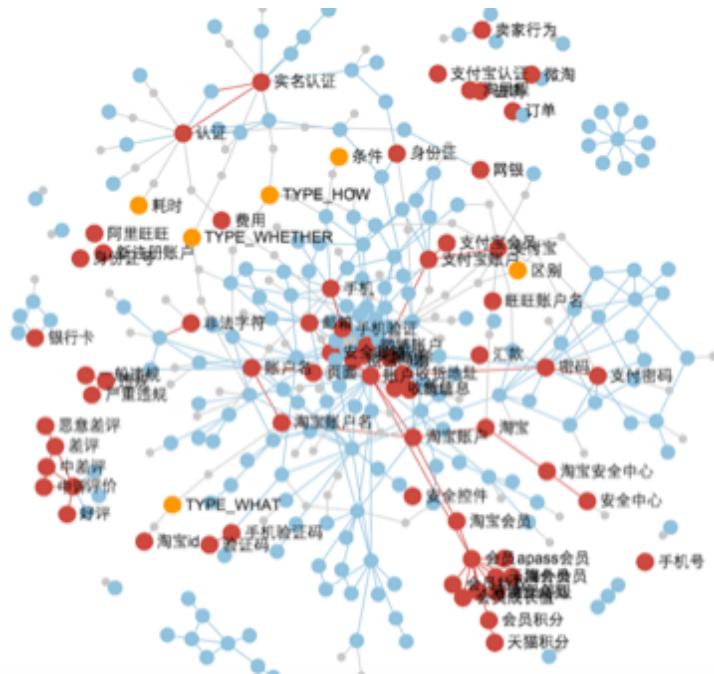
特点：有领域知识的概念，且知识之间的关联性高，并且对精准度要求比较高

基于问答型场景的特点，我们在技术选型上采用了知识图谱构建+检索模型相结合的方式来进行核心匹配模型的设计。

知识图谱的构建我们会从两个角度来进行抽象，一个是实体维度的挖掘，一个是短句维度进行挖掘，通过在淘宝平台上积累的大量属于以及互联网数据，通过主题模型的方式进行挖掘、标注与清洗，再通过预设定好的关系进行实体之间关系的定义最终形成知识图谱。基本的挖掘框架流程如下：



挖掘构建的知识图谱示例如下：



基于知识图谱的匹配模式具备以下几个优点：

1. 在对话结构和流程的设计中支持实体间的上下文会话识别与推理
2. 通常在一般型问答的准确率相对比较高(当然具备推理型场景的需要特殊的设计，会有些复杂)

同样也有明显的缺点：

1. 模型构建初期可能会存在数据的松散和覆盖率问题,导致匹配的覆盖率缺失；

2. 对于知识图谱增量维护相比传统的 QA Pair 对知识的维护上的成本会更大一些；

因此我们在阿里小蜜的问答型设计中，还是融入了传统的基于检索模型的对话匹配。

其在线基本流程分为：

1. 提问预处理：分词、指代消解、纠错等基本文本处理流程；
2. 检索召回：通过检索的方式在候选数据中召回可能的匹配候选数据；
3. 计算：通过 Query 结合上下文模型与候选数据进行计算，通过我们采用文本之间的距离计算方式(余弦相似度、编辑距离)以及分类模型相结合的方式进行计算；
4. 最终根据返回的候选集打分阈值进行最终的产品流程设计。

离线流程分为：

1. 知识数据的索引化；
2. 离线文本模型的构建：例如 Term-Weight 计算等。

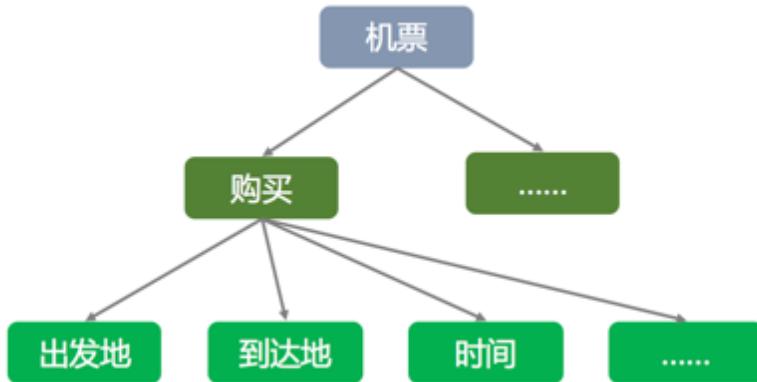
检索模型整体流程如下图：



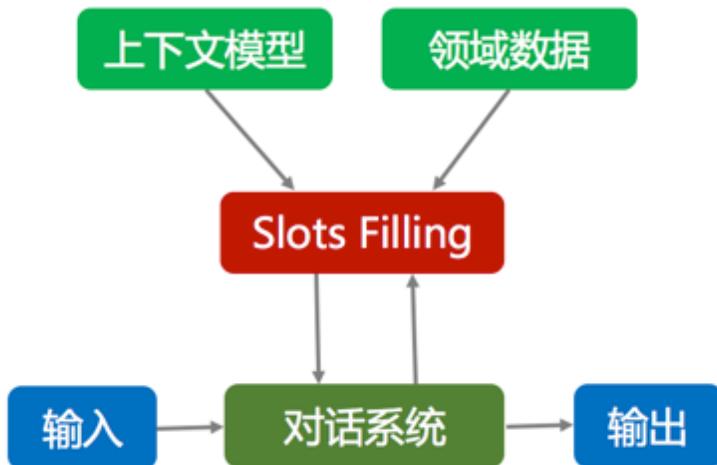
### 任务型：意图决策+slots filling 的匹配方式

特点：有领域知识的概念，每个任务负责独立的业务流程，任务之间相对互斥性强，精准度要求高。基于任务型的特点，在技术选型上，我们采用了意图决策+slot filling 的方式进行会话匹配设计。首先按照任务领域进行本体知识的构

建，例如机票的领域本体知识场景如下：



在问答匹配过程中结合上下文模型和领域数据模型不断在 Query 中进行 slot 属性的提取，并循环进行本体意图树的不断填充和修改，直到必选意图树填充完整后进行输出。如下图：

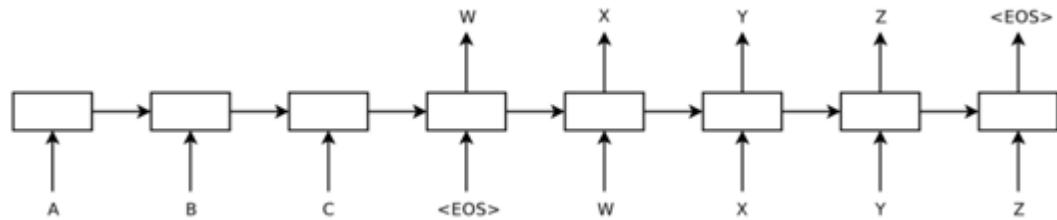


### 语聊型：检索模型与 Deep Learning 相结合的方式

特点：非面向目标，语义意图不明确，通常期待的是语义相关性和渐进性，对准确率要求相对较低。

面向 open domain 的聊天机器人目前无论在学术界还是在工业界都是一大难题，通常在目前这个阶段我们有两种方式来做对话设计：一种是学术界非常火爆的 Deep Learning 生成模型方式，通过 Encoder-Decoder 模型通过 LSTM

的方式进行 Sequence to Sequence 生成，如下图：



一种是 Generation Model(生成模型)：

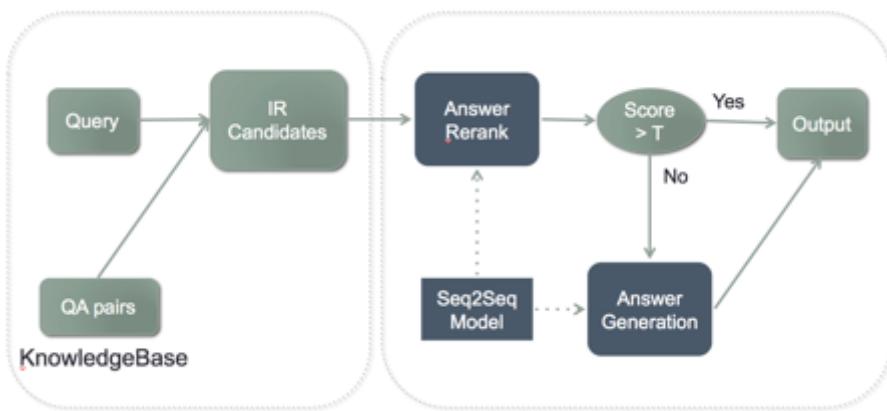
1. 优点：通过深层语义方式进行答案生成，答案不受语料库规模限制；
2. 缺点：模型的可解释性不强，且难以保证一致性和合理性回答。

另外一种方式就是通过传统的检索模型的方式来构建语聊的问答匹配。

Retrieval Model(检索模型)：

1. 优点：答案在预设的语料库中，可控，匹配模型相对简单，可解释性强；
2. 缺点：在一定程度上缺乏一些语义性，且有固定语料库的局限性。

因此在阿里小蜜的聊天引擎中，我们结合了两者各自的优势，将两个模型进行了融合形成了阿里小蜜聊天引擎的核心。先通过传统的检索模型检索出候选集数据，然后通过 Seq2Seq Model 对候选集进行 Rerank，重排序后超过制定的阈值就进行输出，不到阈值就通过 Seq2Seq Model 进行答案生成，整体流程如下图：



## 4 智能交互未来的展望

目前的人工智能领域任然处在弱人工智能阶段，特别是从感知到认知领域需要提升的空间还非常大。智能人机交互在面向目标的领域已经可以与实际工业场景紧密结合并产生巨大价值，随着人工智能技术的不断发展，未来智能人机交互领域的发展还将会有不断的提升，对于未来技术的发展我们值得期待和展望：

1. 数据的不断积累，以及领域知识图谱的不断完善与构建将不断助推智能人机交互的不断提升；
2. 面向任务的垂直细分领域机器人的构建将是之后机器人不断爆发的增长点，open domain 的互动机器人在未来一段时间还需要不断提升与摸索；
3. 随着分布式计算能力的不断提升，深度学习席卷了图像、语音等领域后，在 NLP(自然语言处理)领域将会继续发展，在对话、QA 领域的学术研究将会持续活跃；

在未来随着学术界和工业界的不断结合与积累，期待人工智能电影中的场景早日实现，人人都能拥有自己的智能“小蜜”。

# 7.3 深度学习与自然语言处理在智能语音客服中的应用

作者：余慈

## 前言

95188 电话的支付宝热线目前已经用纯语音交互流程全面代替了传统的按键流程，这个我们称之为“蚁人”的智能语音客服会根据用户的描述判断用户的意图，从而为不同需求的用户提供快速的直达服务，或者直接推送自助解决方案，或者发现是属于紧急问题而直接转给对应业务线的人工客服处理。智能语音客服流程目前日均处理话务占整体话务数的 91%，覆盖上百类业务线以及上千类问题，以超过 70% 的问题识别准确率日均成功为客服分流话务上万通，极大节省了客服人力资源，缩短话务高峰期的用户等待时间，提升了用户体验。在双 11 当天蚁人处理超过 20 万通电话，为双 11 业务提供强有力的支持。

## 1 系统概要

蚁人流程的交互如下：

1. 用户拨打 95188，按 1 进支付宝
2. 系统会提示用户用一句话描述所遇到的问题
3. 用户在电话里描述他（她）想要解决的问题，比如支付宝密码忘记了等
4. 系统会把用户语音输入转成文本，然后调用问题识别模块
5. 对话管理模块（DM）根据识别结果有不同的路径：
  1. 识别出用户要求人工客服，或者需要人工处理的业务类（例如安

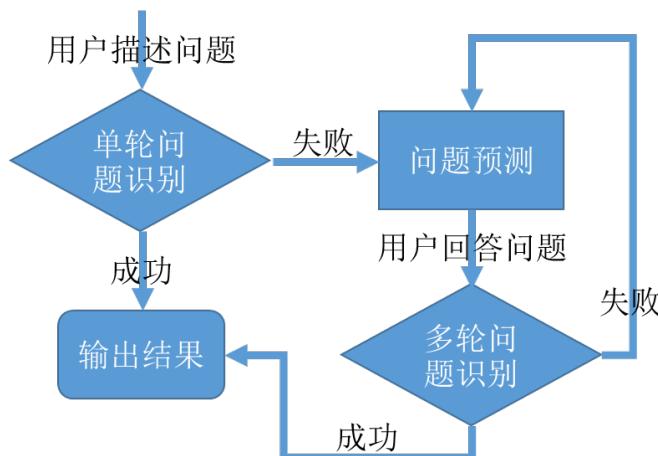
全问题)，就会转到所对应的业务线的客服处理。

2. 识别出的业务可以自助解决，系统就会播放 TTS 给用户：“我想您遇到的问题是 XXXX，请说‘正确’，或者‘错误’”。
3. 识别失败，会进入多轮交互流程，进一步向用户提问并获得回答以帮助问题识别。
6. 系统识别用户在确认阶段的反馈，如果用户肯定了问题，就会推送自助方案到支付宝手机端，并提示用户。如果用户否定，那么会把用户转到人工客服处。

智能语音客服的核心功能是根据与用户的交互信息判断出用户的来电目的，也就是交互步骤#4 中的问题识别模块。问题识别模块允许通过与用户的多轮交互来更准确地判断用户所遇到的业务问题。引入多轮交互流程因为，如果用户只有一次描述问题的机会，下列几个因素的影响往往会导致单轮问题识别无法做出准确的判断：

1. 智能客服的意图判断引擎是基于文本的，而现有的语音转文本技术的字错误率基本上高于 10%，特别是对于一些环境噪音比较大的电话语音数据，ASR 识别错误会比较大地影响单轮问题识别模型的准确率。
2. 用户的表达能力差异化。部份用户难以一次性地准确、完整地描述他(她)的意图，信息量的不足会使得单轮问题识别模型的识别困难。
3. 即在有很人性化的引导语的情况下，现有系统的统计数据证明仍然有相当大比例的用户（约 30%）在面对机器人客服时仍然以“喂”，“你好”等传统对话方式开始交互，而不会按系统引导语的提示来描述意图，导致无效描述。

我们训练了 3 个 DNN，它们之间互相协作来完成整个问题识别流程：



1. 单轮交互问题识别模型：以用户的初始问题描述（如“我的支付宝账号登录不上去了”）为输入执行分类任务，分类目标是 1000 个业务问题。
2. 多轮交互问题识别模型：以与用户的全部对话数据（如“用户：我花呗开通不了啊”，“系统：您是卖家吗”，“用户：是的”）为输入执行分类任务，分类目标与单轮交互问题识别模型相同，也是 1000 个业务问题。
3. 问题预测模型：当问题识别结果的置信分不高于设定的阀值时，系统会认为用户描述信息量不足，问题预测模型就会从问题库里挑选出对分类最有帮助的问题向用户询问，并收集用户的回答用多轮交互问题识别模型来再次判别。

## 2 模型介绍

### 2.1 单轮交互问题识别模型

单轮交互问题识别模型的输入是用户的一句话描述文本（“我的账号被盗了”），输出是该描述所对应的业务分类（“如何解限”）。

我们测试了 feedforward neural network 和 RNN( plain RNN 和 LSTM ) 两种类型的网络，FFNN 的输入为把句子分词后的基于词的 bag of word 向量；Plain RNN 与 LSTM 的输入为基于词的 one-hot 序列。这两种类型网络的各种不同配置测试结果显示在测试集上的准确率 Plain RNN 与 LSTM 会比 FFNN 高约 1.1%，说明用户描述中的词的顺序对业务分类影响不大，因此从性能及训练时间上考虑我们选择了 FFNN。通过多次实验最终确定的结构为：2000 维的输入层 → 500 的线性变换层 → 500 的 sigmoid unit → 1000 的线性变换 → softmax 输出。更多的神经元数量，更多的层数和不同的激活函数，如 RELU 并未产生更好的效果。

支付宝以前的 IVR 流程中有让用户在电话里先描述问题，然后客服接听电话后会打标出该通电话用户所遇到的业务问题。我们取了几个月的数据几千万条，对数据作了一些预处理：分词及词频统计、停用词表过滤、句子长度过滤等。生成的输入所用词表包含按词频排序的 top 2000 个词，扩展词表对准确率没有帮助。清洗完剩下有效描述文本几百万条。

## 2.2 多轮交互问题识别模型

多轮交互问题识别模型的输入是与用户的全部对话内容（如“用户：我花呗开通不了啊”，“系统：您是卖家吗”，“用户：是的”），输出是该段对话所对应的业务分类（“商家怎么开通蚂蚁花呗”）。

对序列建模的自然选择是 LSTM，我们构建了包含 256 个 cell 的 LSTM 网络。把训练数据中，客服与用户的所有话拼接在一起形成一个长句，以词的 one-hot 序列作为 LSTM 的输入。

拉取客服与用户的 380 万通电话录音转成文本，这些电话在接听后被客服人员标注了所对应的业务类别，因此天然就是多轮的训练语料，通过 BPTT 可以训练出 LSTM 网络用于拟合这些实际发生的对话数据所对应的业务类别的分布。

## 2.3 问题预测模型

问题预测模型的输入也是用户的全部对话内容，它的输出是预先定义的问题库中的某一个问题。问题库的数据来源是从几百万通电话录音中提取客服询问用户的问句，通过文本聚类，将同义的问句归到同一类别，每一个问句类别形成一个典型问题，再通过人工 review 的方式筛选出最终的问题库。例如：

- ✓ 余额宝的收支明细吗？
- ✓ 余额宝转出吗？
- ✓ 余额宝转入吗？
- ✓ 大陆公司吗？
- ✓ 大陆个人用户吗？
- ✓ 实名认证吗？

问题预测的目标是挑选一个问题，这个问题如果获得肯定的回答对分类到某个业务帮助最大。用数学建模如下：假设问题库的问题个数为  $N$ ，业务类别总数为  $K$ ，令  $P_i =$  第  $i$  ( $i = 1 \sim N$ ) 个问题是肯定回答的概率， $T_j = P(\text{业务分类为 } j | P_1 P_2 \dots P_N)$  为业务类型为  $j$  的条件概率 ( $j = 1 \sim K$ )，当问题  $i$  变为肯定回答时的信息增益为  $\text{InfoGain}(i) = \text{Entropy}(T) - P_i * \text{Entropy}(T|P_i=1) - (1-P_i)*\text{Entropy}(T|P_i=0)$ ，要挑选出来的问题就是  $i = \text{argmax}(\text{InfoGain}(i))$ 。公式中还有一个问题是如何计算  $T_j = P(\text{业务分类为 } j | P_1 P_2 \dots P_N)$ ，为此我们用了一个多层次神经网络 FFNN 来建模从  $N$  个问题的回答的分布到业务类型的分布的

映射。这个模型的训练数据也是来源于电话录音数据，每一通电话为一个训练数据，与之前的问题库建立流程类似，我们从电话录音文本中提取客服问的问题与用户的回答，转化成  $P_1 P_2 \dots P_N$  向量，加上该通电话的业务类型标注形成了 data-label 的有监督训练语料。多轮流程的交互过程就是模型预测并提出一个问题，用户给出一个回答。根据这个问答，更新对应的问答分布( $P_1 P_2 \dots P_N$ )。根据更新后的输入，重新计算每个问题回答改变后的信息增益，选取信息增益最大的问题，向用户继续提问，直到用户意图足够清晰，多轮交互的 LSTM 网络能够给出高于阀值的分类目标。

## 2.4 迭代优化

在第一版单轮交互问题识别模型上线时识别准确率只有四十几，其主要原因在于训练语料中存在大量的噪声，即众多不同的客服人员对用户描述或电话标注业务类型时存在不少错误或者不一致的情况。这个原因有可能是由于业务熟练程度的原因导致错误标注，或者因为用户的描述本身未包含足够的信息，或表达有误。采用的解决办法是：

1. 用带噪声的数据训练出一个模型
2. 用模型识别一遍训练数据，设定一个阀值找出边界样本
3. 根据业务词表过滤一遍，剩下的再人工检查修正

经过几轮迭代后，识别准确率有很明显的提升。除了训练数据的预处理，我们还在智能客服语音流程里加入了反馈机制：在问题识别完成后会拿识别结果向用户询问系统是否准确地判断出他（她）的问题，用户可以表示肯定或者否定。这部份数据也会在下一个模型迭代周期中成为训练数据的一部份。我们在工程上也建立了数据拉取→清洗→ID 化的自动作流程，形成了数据闭环，使得模型迭代接近全自动化。

## 3 蚁人背后的团队

蚁人项目涉及众多的系统间的交互，CC、MRCP、CSIVR、ASR、TTS、Gateway、DM 等。整个系统的复杂程度很高，它的成功上线离不开众多小伙伴们们的艰苦努力：感谢冷风、圣衣、良穆在和 DM 对接中的工作；小伙伴周蹠在工程上给予的大力支持，智捷及初敏在算法及工程上的各种建设性建议，九清、

弈客、心诗、嫡西、佑助等在蚁人从诞生到落地运营的过程中的巨大努力；感谢坤承在模型训练上给予的大力支持；高杰在项目初期的架构规划上的工作；感谢SPEECH小伙伴们长秦、萧言、燕丹等的鼎力相助，以及众多我无法一一列举的伙伴们。

## 7.4 数据赋能商家背后的黑科技

作者：空望

### 1 背景

马老师曾提到三次技术革命：“第一次技术革命是体能的释放，是让人的力量更大，第二次技术革命是对能源的利用，使得人可以走得更遥远，而这一次技术革命是 IT 时代走向 DT 时代，是真正大脑的释放。我们其实正在进入一个新的能源的时代，这个时代核心资源已经不是石油，而是数据。”。逍遥子也曾经讲过：“我们用大数据赋能了双 11，赋能了我们自己的运营能力。我们还要更上一层楼，利用大数据赋能所有的商家，帮助他们运营好消费者，这样才能让我们在大数据时代践行‘让天下没有难做的生意’的使命。”

新商家事业部自去年 12 月成立以来，数据赋能商家就是重要的方向之一。我们将之前平台沉淀的数据和算法的能力转过来赋能我们为商家提供的工具和平台，这其中包括客户运营平台、千牛、服务市场等等。很多技术在今年的双 11 也起到了非常显著的作用，为商家带来实实在在的收益。下面从客户运营平台、千牛头条、服务市场三个产品给大家分享一下我们如何通过 ML&AI 技术重新定义产品。

### 2 客户运营平台

客户运营平台旨在为商家提供基于大数据和人工智能的客户精细化运营能力。阿里巴巴 2015 年提出客户运营战略方向，带领平台的商家从“流量经营”向“客户运营”转型。客户运营的核心理念有两个方面：其一是精细化，其二是从过于关注短期经营的成交目标向关注客户粘性、客户体验、客户忠诚度的目标转变。客户运营平台提供了“访客运营”和“会员粉丝运营”两大利器，借助大数据和人工智能技术，帮助商家提升客户运营的效率。2016 年双 11，超过 23

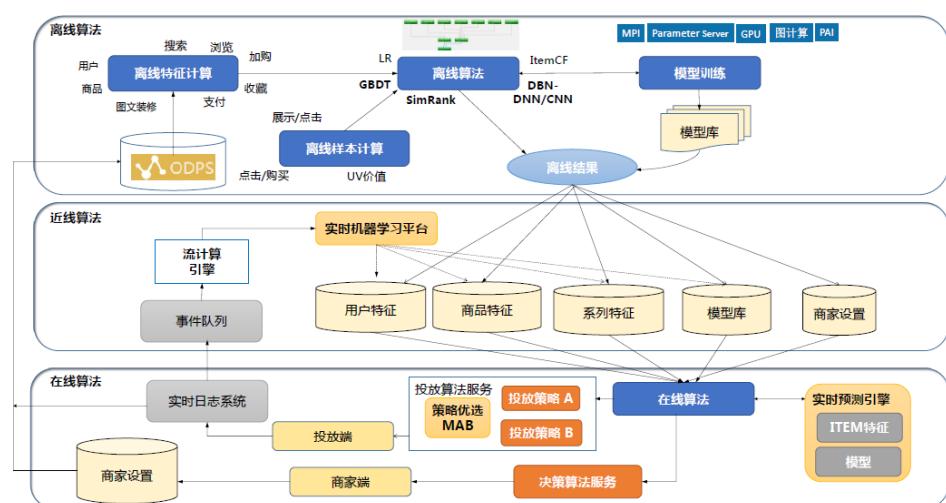
万商家通过客户运营平台实现了店铺的个性化运营和粉丝会员的精准营销，显著提升了成交转化。下面以访客运营为例，介绍 AI 分群引擎的应用。

访客运营通过对访客的细分和精准营销策略来提升转化的效果。如下图所示，是某美妆店铺的访客分群个性化店铺首页策略示例，左侧和右侧分别是针对水乳类和面膜类偏好人群的个性化首页，通过面向细分的人群投放有针对性的内容，可以显著提升客户体验和访客成交转化率。类似的分群运营策略还可以应用在详情、微淘等场景。



访客运营示例：个性化店铺首页

有别于普通的营销平台，阿里巴巴客户运营平台通过 AI 能力，实现了对访客的实时分群，也就是说当客户进入店铺的一瞬间，客户运营平台的 AI 引擎会对用户所属的人群进行实时预测，得到此时此刻的最佳分群结果。下图是 AI 分群引擎架构图：



AI 实时分群的三大特点如下：

1. **行业级别的模型** 同一个特征在不同的行业会有不同的重要性，例如地域属性，对于美妆行业用户是否偏好保湿产品具有较强的预测能力，因为北方干燥，南方潮湿，而对于快消行业，比如用户是否会喜欢吃某种口味的零食，地域属性预测能力就会比较弱。因此同一个特征在不同的行业会有不同的权重。
2. **长期、近期和实时相结合的特征体系** 例如用户对男装、女装、童装三类服饰商品的偏好，既受用户的人口统计学特征，比如性别、年龄段的影响，又受随机实时情境的影响，比如女性客户可能会为男友或家人购买男装，因此当我们按照偏好对客户分群时，既需要考虑长期稳定的属性和行为，又需要结合实时的行为和需求。
3. **店铺分群自适应** 以美妆行业为例，从用户在平台层面上的行为来看，大部分用户可能会偏好美白、保湿功效的产品，而具体到某个品牌，其主营的产品品类和平台总体的品类成交分布很可能有较大偏差，比如一些品牌可能是主打彩妆，另外的一些品牌则主打紧致类的产品。当我们用平台整体数据建立的模型对用户在某个店铺的分群归属进行预测的时候，很可能会出现偏差。为了使得 AI 引擎能够适应店铺多样化的情况，我们从两个角度进行了升级，其一是引入店铺内销量分布作为分群结果的先验，其二是引入了增强学习技术（多臂老虎机 MAB）为每个商家自动调整模型参数。

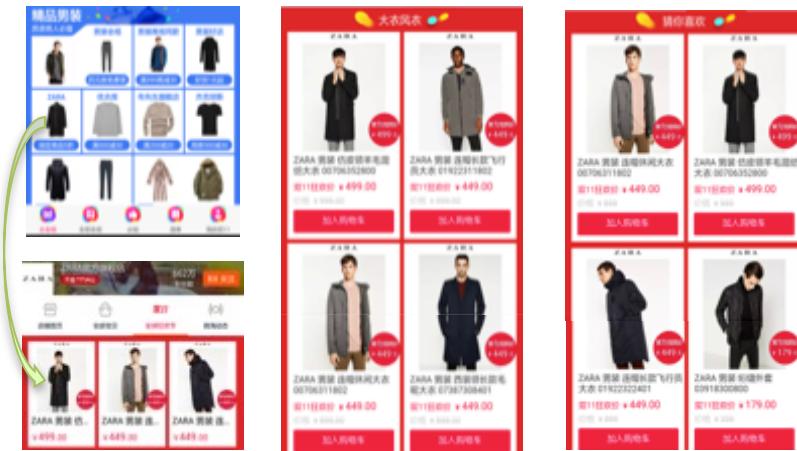
对比于静态的人口统计学分群方法，AI 分群具有实时性高、精准性好、店铺自适应等特点，商家实际使用的效果提升也更加显著。

阿里巴巴作为一个新零售平台，平台上的商家具有很高的多样性。客户运营平台基于增强学习技术的自适应技术，能够使得 AI 分群自动适应每个店铺的策略特点。通过该方法的应用，在一级类目偏好分群的场景下，成交转化率提升超过 10%，同时个性化的渗透率大幅提升 40%。

### 3 双 11 大促店铺承接页个性化技术

今年双 11 阿里巴巴首次实现了全面的全站个性化，包括从会场到大促承接页到店铺再到详情，而大促店铺承接页是连接会场与店铺的桥梁，大促承接页的整体活动氛围和布局由平台确定，商家可以通过页面装修工具来装修商品模块、

营销模块等内容。2016 年双 11 大促承接页首次实现了全面的个性化，实现了显著的成交转化提升。



“所见即所得”，会场个

性化联动

“猜你喜欢”瀑布流

大促承接页个性化属于单个店铺内的个性化，相比于平台级的个性化场景（例如手机淘宝猜你喜欢、有好货等），具有一些特有的挑战。平台级的个性化我们可以想象为把平台的大量商品作为候选商品，形成一个虚拟店铺，用户在平台上的行为都是这个虚拟店铺内的行为；店铺内的个性化，候选商品为单个店铺的商品，用户在单个店铺内的行为是非常稀疏的，必须要考虑借助用户在平台整体的行为来做好店铺内的个性化。面向店铺内个性化的特点，我们的 AI 个性化引擎框架主要包括 matching 框架和 ranking 框架两个部分。

matching 框架解决的是用户偏好意图的覆盖，即基于单店的小数量级商品尽可能全面地匹配用户的偏好和意图，matching 框架还有一个重要的作用是为 ranking 提供输入特征，ranking 框架解决的是个性化商品列表的最优排序问题，它基于历史反馈数据、用户特征、商品特征、用户与商品的交叉特征等训练面向特定业务目标的模型。

#### Scenario Adaptation (Learning-to-rank-based Reranking)

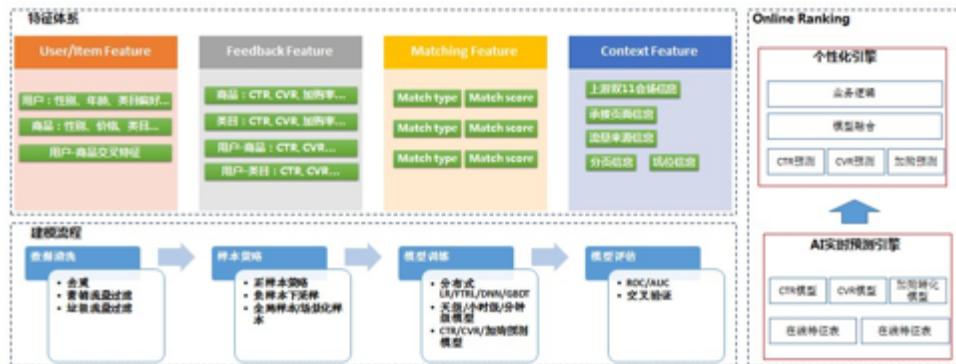
Graph Mining	Hashing-based	Graph Embedding	Semantic-based	Stream Computing
<ul style="list-style-type: none"> <li>• Adsorption</li> <li>• Academic Adar</li> <li>• <u>Simrank</u></li> <li>• <u>Simrank++</u></li> </ul>	<ul style="list-style-type: none"> <li>• <u>Minhash</u></li> <li>• <u>Simhash</u></li> </ul>	<ul style="list-style-type: none"> <li>• LINE</li> <li>• Node2vec</li> <li>• APP(Asymmetric Proximity Preserving Embedding)</li> </ul>	<ul style="list-style-type: none"> <li>• Property matching</li> </ul>	<ul style="list-style-type: none"> <li>• Stream ItemCF</li> <li>• Stream TopN</li> </ul>

在 matching 阶段，我们可插拔的框架支持多种 matching 方法：

1. 基于图挖掘的 matching 方法 包括 adsorption、adar、jacacard、simrank、simrank++以及基于大规模流式计算的增量 ItemCF 方法；
2. 基于索引的方法 包括 minhash、simhash 等 hashing-based 的方法，利用局部一跳信息建立商品索引；
3. 基于 Graph Embedding 的方法 Graph Embedding 也是一种商品索引，与 hashing-based 方法的不同之处在于它可以利用图的全局信息、多跳信息来建模，具有更高的覆盖度；
4. 基于语义匹配的方法 例如基于用户偏好的商品属性与商品本身的属性进行匹配；
5. 基于流式计算的方法 基于流式计算引擎，实时更新商品之间的关联关系，更好地捕捉线上流量的变化
6. 面向场景的适配 基于排序学习技术，可以将上述方法的输出进行面向场景的重新排序和打分，使得 matching 层面向特定场景调优

我们创新地提出了面向电商场景的大规模分布式 Graph Embedding 的算法。商品 Embedding 算法可以将一个现实生活中的商品实体表示成低维空间里的一个向量，使得我们可以仅通过这些向量之间的空间位置关系就能得到商品之间的某些联系。由于我们可以将学习出来的 Embedding 向量作为更上层机器学习任务的输入，这就使得 Embedding 这样的表示学习非常有潜力。在双 11 中，我们设计实现了一种能够保持非对称关系的 Graph Embedding 算法，来对商品进行 Embedding。由于用户对大部分商品（长尾）的点击行为非常稀疏，我们选择将用户的点击序列拼接成商品点击转换图的形式，来试图缓解稀疏性问题以提高商品 Embedding 的质量。另一方面，为了表示商品之间的非对称关系（例如购买手机后可能会对手机壳感兴趣，但反之不成立），我们用两个 Embedding 向量来表示一个商品的不同角色（已被看过和待预测）。我们在构建好的商品图中，对每个商品节点按照带重启动的随机游走进行路径采样，并且仅按照采样的正方向分别对两个 Embedding 向量进行更新。我们基于分布式平台 ODPS Graph 首次实现了亿级别节点、百亿级边的大规模图数据的 Embedding，并且在理论上，我们还证明了基于这种采样方式的 Graph Embedding 算法能够保持原图节点之间的 Rooted PageRank 的相似度关系，相关工作已被人工智能顶

级国际会议 AAAI 2017 接收。



在 ranking 阶段，我们通过亿级特征的大规模稀疏模型，包括 LR、FTRL、DNN ( 深度学习 )，十亿级样本的训练，实现精准的个性化排序。特征的体系包括用户、商品、matching 特征、场景相关的反馈类特征、以及场景化( context )特征，在特征实时性方面我们会结合长期特征、短期特征和实时特征，在追踪线上变化的同时能够保持较高的覆盖度和稳定性；在样本层面，通过日志去噪、样本采样策略、样本构造策略（页面级样本和模块级样本），优化样本构造；在模型层面，通过天级、小时级和实时模型，结合面向不同目标的模型来综合优化业务目标，比如我们会利用多个模型对点击、加购、成交等目标进行建模，并实时结合多个模型来优化最终的业务目标。我们的分布式 GBDT 排序学习算法能够支持不同类型的排序函数，包括 pairwise ranking 和 listwise ranking，能够从样本和特征两个维度对数据进行拆分并进行并行化训练，可以支持百亿级样本。深度学习模型训练基于阿里巴巴集团多机多卡的分布式 GPU 训练平台，可支持大规模亿级稀疏特征、亿级样本的深度学习模型快速训练。我们对深度学习实时预测的性能进行了深入优化，预测性能是普通实现的 10 倍以上。

通过算法细节的不断调优，双 11 承接页个性化效果显著，相比于非个性化页面，个性化页面的成交转化有超过 20% 的提升，带来了十亿级别的成交提升。

## 4 千牛头条技术介绍

### 4.1 产品概述

千牛头条是一个定位于通过内容传播与运营，满足商家内容消费需求的商业媒体平台。为了满足商家获取实时、个性化资讯的需求，同时提高千牛平台的流

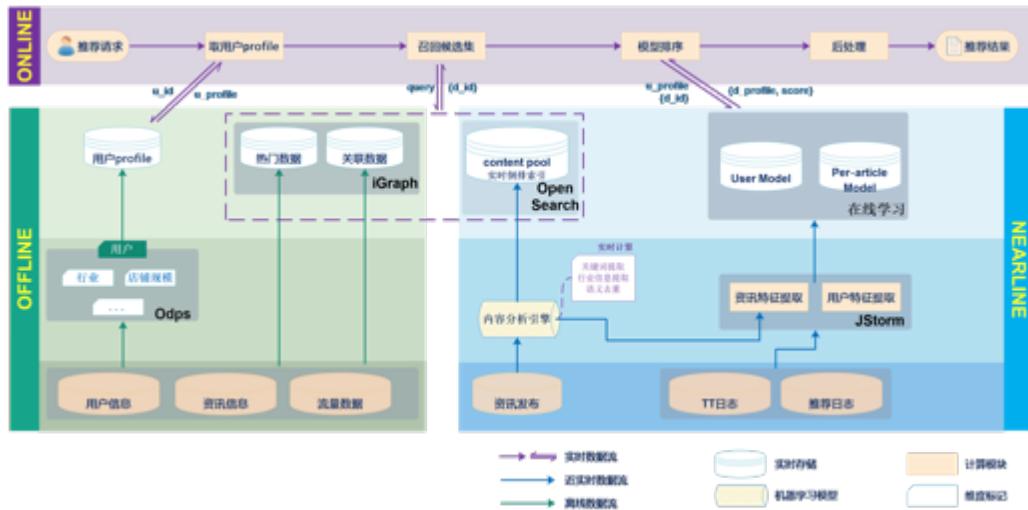
量效率，为千牛头条上线了一套个性化的资讯推荐系统。



千牛头条双 11 和热门频道

目前个性化算法支持了热门 feeds 流、双 11 频道，后续会支持更多的频道和场景。

## 4.2 千年头条推荐整体框架



千牛头条推荐系统总体框架

千牛头条资讯推荐系统的总体框架上图所示，整个框架可分为离线、近实时、实时三部分。

离线部分主要负责用户 Profile 构建以及关联数据挖掘；近实时部分主要包

括新发布资讯的实时分析引擎，模型流式更新引擎；实时部分基于通用的推荐 pipeline，并结合离线、近实时模块的产出结果，完成整套的推荐逻辑。

## 4.3 技术创新点

### 1. 基于期望偏好的用户 profile

为了全方位地刻画用户兴趣，从关键词、类目、行业等维度对用户兴趣进行建模。在计算用户 profile 过程时引入期望偏好，期望偏好是根据行为分布计算的用户预期行为分布，通过用户的期望偏好分布和实际偏好比较得到偏好分，最后用伽马泊松分布对偏好分进行平滑。

### 2. 资讯实时分析引擎

对新发布的资讯进行实时文本分析，包括分词、关键词抽取，通过行业 profile 和 Multi-task 语义向量生成行业标签，并对资讯建立倒排索引，实现实时更新。技术上集成使用 TextRank，Mutual Information, Log Odds Ratio 三种关键词抽取算法实现精准抽取；Multi-task 语义向量模型与传统的 Word2Vector 词向量模型不同，使用词的 meta data 数据，使得模型在给定上下文的情况下，同时学习词的分布和词 meta 信息分布。

### 3. Online Bayesian Logistic Regression 模型

千牛头条用户在百万量级，每天的新资讯相对较少，每篇资讯能够累计较多的用户行为，为此采用了 PerArticle 的模型方法，即针对每篇资讯单独训练一个 Online Bayesian Logistic Regression(BLR)模型，精细化地刻画每篇资讯。相比传统的 LR 模型，BLR 模型认为模型参数本身是有先验分布的，具有更优的泛化能力，上线后 ctr 也有 20% 的提升。

$$p(y=1|x) = \int \sigma(w^T x) q(w|D) dw$$

其中  $\sigma(w^T x)$  就是 LR 模型的预测函数， $q(w|D)$  就是 Bayesian 模型学习的参数分布。

在线 Bayesian 学习算法如下：

- a )  $m_i = 0, q_i = \lambda$ , 每一个参数有一个独立的高斯先验  $\mathcal{N}(m_i, q_i^{-1})$
- b ) 得到一批数据  $(x_j, y_j), j = 1, \dots, n$
- c ) 优化目标函数  $f(w) = \frac{1}{2} \sum_{i=1}^d q_i (w_i - m_i)^2 + \sum_{j=1}^n \log(1 + \exp(-y_j w^T x_j))$

d ) Laplace approximation 求解目标函数，得到模型的均值，方差

$$m_i = \underset{w_i}{\operatorname{argmin}} f(w)$$

$$q_i = q_i + \sum_{j=1}^n x_{ij}^2 p_j (1 - p_j), p_j = (1 + \exp (-w^T x_j))^{-1}$$

#### 4. 高阶泛化统计类特征

模型使用的特征包含用户特征、资讯特征、用户与资讯的交叉特征等，其中用户与资讯的交叉特征是关键特征。传统的做法直接对用户特征和资讯特征进行交叉，这种方法很容易导致特征爆炸，交叉后大量冗余的特征容易导致模型过拟合；为了解决上述问题，采用了一种根据业务经验进行特征交叉，然后对交叉特征计算统计量的方法，该方法可以很好地解决特征组合爆炸问题，同时生成的统计类特征有较好的泛化性。

#### 5. 资讯冷启动

时效性在资讯推荐中至关重要，而新资讯由于缺乏线上反馈导致 CF 等基于行为的推荐算法失效。

从文章维度来看新文章没有用户行为，但是从新文章包含的关键词、主题等维度看，历史上有很多文章也都包含这些特征，因此可以根据新文章包含关键词、主题的历史表现推断新文章的表现。

### 4.4 业务成果

千牛头条经历了两次大的算法升级，核心指标如点击率有了显著的提升，第一次升级增加了基于用户行为的个性化；第二次升级引入文本算法以及在线排序模型。

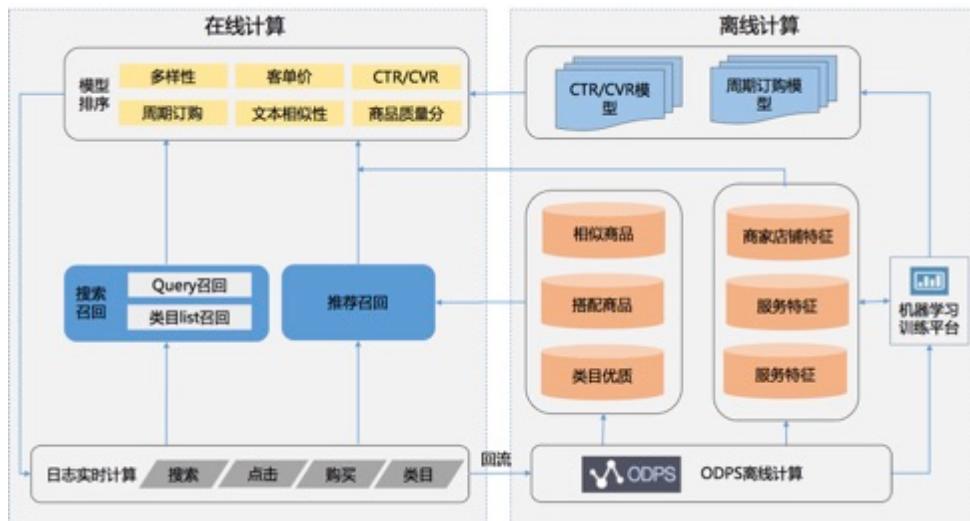
## 5 数据赋能服务市场

### 5.1 背景介绍

服务市场是面向淘系商家提供多样化服务的交易平台，目前覆盖淘系活跃卖家 90%以上。其特点是：用户访问频次低、访问路径短、行为少，订购呈现周

期性。原来的服务市场千人一面，不能很好匹配商家实际需求，导购效率较低。

为解决以上问题，我们设计了服务市场个性化框架（如下图），在个性化搜索和推荐场景中取得了显著的效果。其中搜索点击率提升 10%，空结果率降低 400%；千次展示成交数提升 20%；推荐点击提升 90%，千次展示成交数提升 200%，转化率比服务市场整体高 70%。



个性化导购框架

框架分为在线计算和离线计算，在线计算负责实时的商家行为分析，商品召回，个性化排序；离线部分负责商家/服务特征更新，订购模型训练以及候选商品池计算。



多样化的个性化推荐算法场景

## 5.2 关键技术点

### 1. 实时偏好识别

服务市场的用户访问频次低，识别用户的实时偏好有助于更准确的匹配用户需求。实时偏好包括实时商品偏好和实时类目偏好两个维度，使用时间衰减累计+用户实时访问反馈调整的方式来构建用户实时偏好模型。在实际使用时，根据历史累计的数据选取 TopN 产生实时偏好。

## 2. 匹配召回

服务市场搜索召回面临问题：搜索无结果、搜索结果相关性不高以及搜索结果不够优质。针对上述问题，使用核心词抽取和 query 扩展对原 query 进行语义分析和补充。包括：基于语义 embedding 对搜索词进行自适应分词和向量化表示；为保证核心词与原 query 语义相似，综合语义单元的类目分布熵、与原 query 的邻接熵，与原 query 类目匹配度进行核心词抽取；为了扩大 query 召回，根据用户搜索补充了与原 query 相关的其他搜索核心词。补充后的搜索词大幅降低了搜索无结果率，搜索点击率和转化率也得到明显提升。个性化推荐召回以实时商品偏好，实时类目偏好，近期搜索，历史订购商品为基础，配合相似商品，搭配商品，类目优质商品进行扩大召回，从而构建个性化推荐的优质多样的商品池。

## 3. 模型排序

个性化推荐的模型排序部分负责对召回的商品池结合当前商家店铺和商家行为特征进行个性化排序针。对特定的排序模型组装合适的模型特征（包括单一特征，组合交叉特征，以及 ID 类特征等），然后根据 CTR/CVR 模型生成预测分数；同时结合商家订购服务的特点，以及推荐多样性等策略的考虑，对分数进行重排。

# 7.5 探索基于强化学习的智能推荐之路

作者：朱仙

## 1 背景

随着千人千面个性化推荐技术在手机淘宝的全面应用，推荐场景日均引导成交在整个手淘平台占据着非常重要的比例。用户越来越习惯于逛淘宝来满足日常的休闲或者购物的需求。然而很多时候，用户购买的目的性并不是很明确。一个宝贝的成交，可能会跨越多天，经历较长的决策周期。与此同时，在整个用户的购买决策过程中，他们经常在多个异构的推荐场景间进行跳转，进而影响自身后续的行为序列。

因此，推荐的优化需要从单一场景的直接优化，逐步进阶到多场景的联动优化，这就需要我们将用户的长期行为序列纳入建模，从而实现整个手淘推荐链路的累积收益最大化。在这样的背景目标下，我们先后探索落地了两项阶段性的优化工作：

- 1 ) 基于监督学习的购买决策建模；
- 2 ) 基于强化学习的全链路场景优化；

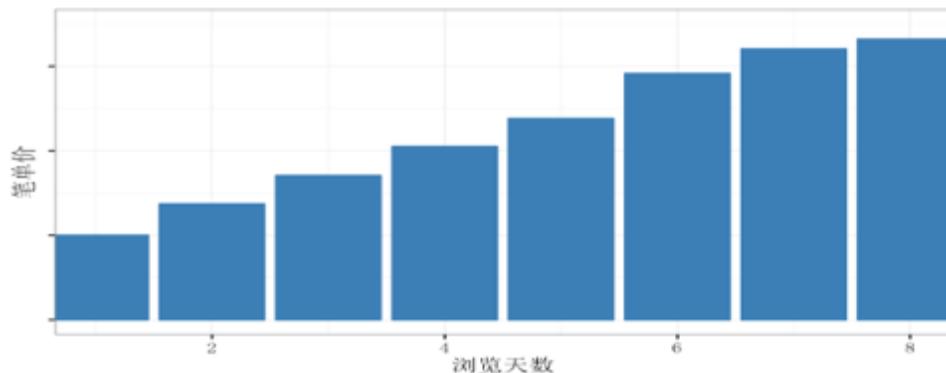
通过这两个阶段的工作，我们初步实现了手淘推荐引擎的合理化与智能化，它能根据用户-系统的联动交互，结合用户当前的状态，给出最优的推荐决策，从而获得长期累积受益的最大化。

## 2 基于监督学习的购买决策

我们首先对已有的单一场景推荐过程进行深入地思考和分析：常见的成交模型仅仅通过用户短期行为对用户当天的成交进行预估，实际上是局部贪心的算法。导致的结果是无法引导高质量的成交，在优化 cvr 时往往会引起笔单价的下降（笔单价低的商品存在 cvr 高的天然属性），即使最终 gmv 提升，也是建立在降低成交质量的代价之上。

### 2.1 建模背景

以首页导购场景猜你喜欢为例，我们对用户的行为与成交日志进行分析。可以发现用户在某一类目或商品上的购买行为存在一个决策的过程，并且这一过程的长度与成交单价强相关，如下图所示：



购买决策过程是一个解释消费者行为的成熟理论，除去购买价格低决策风险小或品牌忠实用户导致快速决策的情况，大部分购买决策过程包括以下几个阶段：

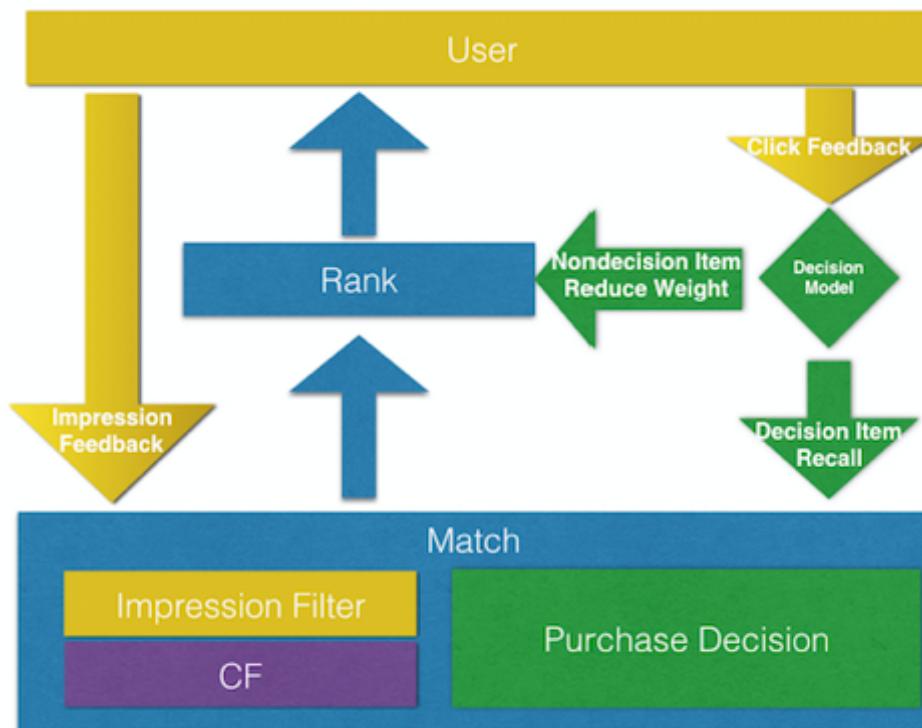
1. 需求产生：用户通过内部以及外部刺激发现现状与理想状态存在的差距，从而激发了购买需求。
2. 信息搜集：用户通过各种渠道（广告、亲友、互联网搜索引擎）学习到相关的行业知识。
3. 候选比较：用户在这一阶段，将根据学到的行业知识聚焦到契合用户需求的几款候选商品并进行评估。

4. 购买决策：完成评估后，用户将会进入最后的决策阶段，根据自身购买力与偏好等信息进行选择，这一环节仍然会受到其他因素(亲友反馈、预算不符等)影响而被打断。

5. 购后反馈：用户购后对商品的体验环节会修正用户对行业的认知。

我们借鉴这样的思路，将用户的购买行为看作多个阶段组成的过程，并结合场景日志对购买决策过程进行建模。我们暂不讨论用户成交后的行为(即购后反馈)，只将购买决策阶段作为整个购买决策过程的终点。

## 2.2 决策建模



如上图所示，以 CF(协同过滤)方式召回的商品会通过小时级曝光日志进行曝光过滤。这一策略在整体上对 ctr 指标有提升，向用户曝光更为丰富多样性的商品，提升用户购买初期信息搜集的体验，但也会过滤用户行为收敛后真正中意的商品，打断用户的购买过程。

因此，我们在 CF 之外增加购买决策阶段商品的召回，不对决策商品进行强制曝光过滤，只是将曝光作为特征加入模型。我们为购买决策阶段进行建模，用户反馈点击通过购买决策模型判别为决策商品或非决策商品：决策商品被重新召回，而非决策商品将会在排序中被降权。

我们收集用户在详情页的行为作为判断用户是否处在购买决策阶段的主要特征，结合全网以及场景的用户偏好，以场景 GMV 为目标进行建模。

$$\begin{aligned}x_{u,i} &= [page_{u,i}, prefer_{u,i}] \\ctr_{u,i} &= g(x_{u,i}) \\cvr_{u,i} &= h(x_{u,i}) \\gmv_{u,i} &= ctr_{u,i} cvr_{u,i} price_{u,i} = g(x_{u,i})h(x_{u,i})price_i\end{aligned}$$

$page_{u,i}$  为用户  $u$  在商品  $i$  详情页的行为，描述了用户对商品的决策行为。 $prefer_{u,i}$  表示用户  $u$  对商品  $i$  的偏好向量。 $g(\cdot)$  与  $h(\cdot)$  是我们结合场景样本与模型拟合得到的  $ctr$  与  $cvr$  预估函数。

与常规 gmv 优化任务所不同的是，我们的  $\langle u, i \rangle$  是全部来自于场景点击召回再次曝光的样本。我们使用 gbdt 作为  $ctr$  与  $cvr$  的拟合函数  $g(\cdot)$ 、 $h(\cdot)$ 。

根据训练的模型，我们可以以预估的 gmv 进行排序，并设置阈值将回归模型转化为二分类模型，从而判断用户是否处在购买决策的阶段。决策商品会被重新召回，而非决策商品会被降权。在此基础上结合商品降价、售罄等利益点在推荐场景中向用户展示，从而促使用户做出最终的购买决定，减少用户的决策时间。

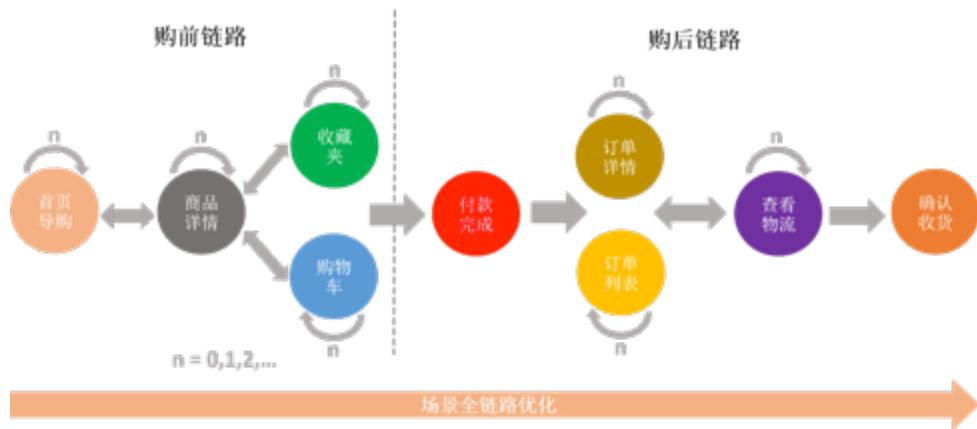
我们的策略在日常进行小流量实验，对比效果在客单价与总成交上都有非常显著的提升；双 11 当天大促全量上线，更是以极少的 pv 占比引导了很大比例的成交。

### 3 基于强化学习的全链路优化

在第一阶段的工作中，我们基于传统监督学习建模的方式，在一定程度上提升了单一场景的引导成交和客单价。但总体上来讲，模型上仍然不够优雅智能，

用户在成交链路上的行为序列特性并没有在模型上体现。

事实上，在电商的个性化推荐中，存在一条典型的交易链路，它能基本刻画一个用户完成一次购买行为所经历的路径环节。比如用户在首页闲逛时可能会对某个商品发生兴趣，然后点击进去查看商品的详情，或加购、或收藏；当用户下定决心购买后，会到达付款完成页，之后又可能会查看订单详情、物流信息等，直到最后确认收货。



这些页面都有相应的推荐场景，在购买前通过推荐辅助促进用户的决策过程；用户完成某个宝贝购买后，全力捕捉用户的购物热情，继续通过合理的推荐让用户继续购买更多的宝贝。整个交易链路的核心就是最大力度的引导用户成交，实现全链路成交的累积最大化。

所以我们就继续考虑能否采用类 MDP 的方式，进行更为合理智能的全链路优化建模，即在用户每一个当前的链路状态，推荐引擎可以依据一定的策略，输出相应的推荐行为，然后根据用户的反馈交互，对策略进行迭代更新，从而最终逐步学习到最优的推荐策略。换句话讲，也就是强化学习的建模优化思想。

然而与搜索等场景不同的是，商品的个性化推荐对用户来讲是一个“Soft Need”，很多时候购物的目的性并不是非常的明确。一个宝贝的成交，可能会跨多天，决策周期较长。并且在购买决策的过程中，用户会经常性的在多个异构的推荐页面场景中进行跳转，情况也较搜索更为复杂。

### 3.1 算法背景

结合推荐场景的语义环境，首先明确强化学习几个核心元素的基本含义。我们将推荐引擎视为 Agent，

S：引擎能感知的当前用户状态空间；

A：商品推荐空间；

R：奖赏函数；一次状态跳转 Agent 能获得的奖赏；

强化学习学到的是一个从环境状态到动作的映射，即行为策略  $\pi: S \rightarrow A$ ，而学习的目标，就是通过与环境的交互过程中，寻找到一个最优策略  $\pi^*$ ，在任意状态 s 下，获得的长期累积奖赏最大化，即：

$$\pi^*(S) = \operatorname{argmax} V^\pi(s)$$

$V^\pi(s)$  为策略对应的状态值函数，表示状态 s 下的折算累积回报：

$$V^\pi(s) = E_\pi[r(s'|s, a) + \gamma V^\pi(s')|s_0 = s]$$

对应的状态-动作值函数为：

$$Q^\pi(s, a) = E_\pi[r(s'|s, a) + \gamma V^\pi(s')|s_0 = s, a_0 = a]$$

即在状态 s 下，采用动作 a，Agent 能获得累积奖赏期望。显然，对于最优的策略  $\pi^*$  对应的状态值函数和动作值函数，有：

$$V^*(s) = \max_a Q^*(s, a)$$

对于经典的强化学习，可以通过寻找最优的状态值函数或动作值函数，学习最优策略  $\pi^*$ 。

### 3.2 算法估计

基于值函数估计的核心思想就是，将状态-动作值函数进行参数化，将大规模的状态动作空间转化为参数空间，在降维的同时增加函数本身的泛化能力。即：

$$Q(s, a; w) = f_w(\phi(s), \psi(a)) \approx Q^*(s, a)$$

我们通过更新参数  $w$  来使得  $Q$  函数逼近最优的  $Q$  值。

其中， $\phi(s)$  为状态  $s$  的特征向量，包含：用户自身维度的一系列特征、当前所处链路场景的特征信息、以及上一跳的 trigger 特征信息等； $\psi(a)$  为商品维度的一系列特征。

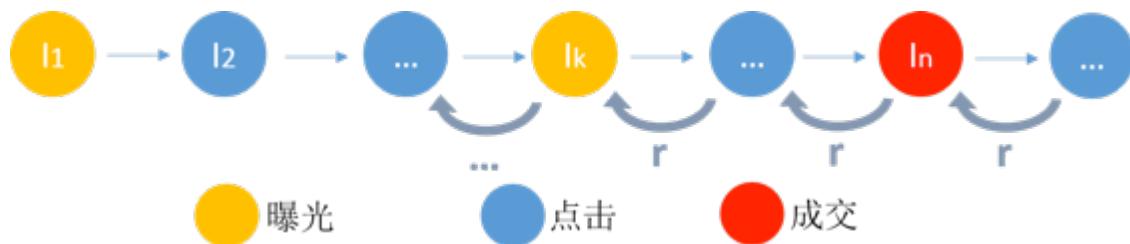
而  $f$  本身是一个回归模型，典型的包括：线性回归，树回归（e.g., rf 或 gbrt 等），以及神经网络的方式。

我们通过 Q-Learning 的方式进行估值的迭代，即：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

其中， $\alpha$  能够减少估计误差造成的影响，类似随机梯度下降，从而最终收敛到最优的  $Q$  值。

在电商的个性化推荐中，用户的购物目的性并不是很明确。一个宝贝的成交，可能会跨多天，决策周期较长。并且在购买决策的过程中，用户经常会在多个异构的推荐页面场景中进行跳转。为了应对这一状况，我们对交易链路多个关键场景中，一定周期内的用户真实曝光、点击、成交等行为，在类目限制的基础上按照时序进行关联，从而可以构建出一个行为决策序列，如下图所示：



我们结合实际的业务需求，对行为序列中单步的奖赏收益可以进行具体的定义。

当我们找到最优的估值函数，就可以依据当前的  $Q$  值计算出最优的动作输出给用户。这种方式称之为 Greedy Policy，即：

$$\pi(S_{t+1}) = \operatorname{argmax}_a Q(S_{t+1}, a)$$

这里可以进一步使用 e-greedy policy 的方式，增加一定的 exploration，会有利于更新  $Q$  值从而获得更好的 Policy。

强化学习的全链路优化策略最终在多个场景取得了非常大的业务指标提升。策略本身具有很好的智能迭代成长能力，同时可以优雅地建模用户长周期的购买决策行为，有效促进了高客单价商品的决策成交。

## 结语

每年的双 11 都是对我们日常算法优化的一次大考，也是我们实验前沿技术的最佳阵地。我们在去年千人千面全面触达终端用户的基础上，进一步探索了更为复杂的多场景全链路联动优化，使推荐本身更加的合理化和智能化。从基于监督学习的购买决策，到基于强化学习的全链路优化，我们稳扎稳打，初步开启了智能推荐的新篇章。

# 7.6 DNN 在搜索场景中的应用

作者：仁重

## 1 背景

搜索排序的特征分大量的使用了 LR , GBDT , SVM 等模型及其变种。我们主要在特征工程，建模的场景，目标采样等方面做了很细致的工作。但这些模型的瓶颈也非常的明显，尽管现在阿里集团内部的 PS 版本 LR 可以支持到 50 亿特征规模，400 亿的样本，但这对于我们来说，看起来依然是不够的，现在上亿的 item 数据，如果直接使用 id 特征的话，和任意特征进行组合后，都会超出 LR 模型的极限规模，对于 GBDT , SVM 等模型的能力则更弱，而我们一直在思考怎么可以突破这种模型的限制，找到更好的特征；另外，及时 LR 模型能支持到上亿规模的特征，在实时预测阶段也是有极大的工程挑战，性能与内存会有非常大的瓶颈。

所以我们第一考虑到的是降维，在降维的基础上，进一步考虑特征的组合。所以 DNN 很自然进入了我们的考虑范围。再考虑的是如果把用户行为序列建模起来，我们希望是用户打开手淘后，先在有好货点了一个商品，再在猜你喜欢点了一个商品，最后进入搜索后会受到之前的行为的影响，当然有很多类似的方法可以间接实现这样的想法。但直接建模的话，LR 这类的模型，很难有能力来支持这类特征，所以很容易就想到了 RNN 模型。

## 2 相关工作

同时前人有很多工作给予了我们提示。Deep Learning over Multi-field Categorical Data 这篇 paper 开始使用 id 类的特征进行 CTR 预估。

Google 也推出 Wide & Deep Learning for Recommender Systems 的 Wide&Deep 模型用于推荐场景。在 FNN 的基础上，又加上了人工的一些特征，让模型可以主动抓住经验中更有用的特征。

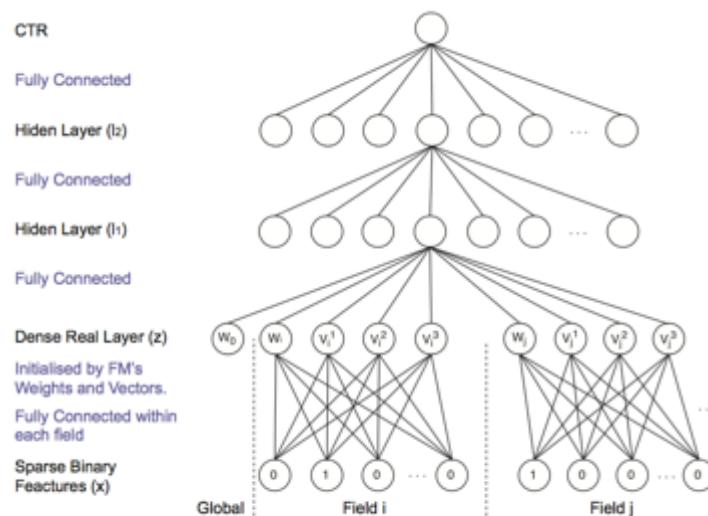
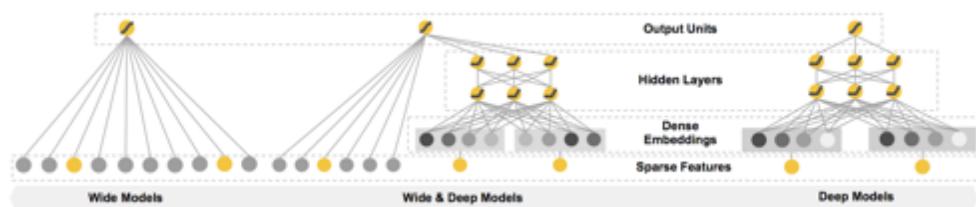


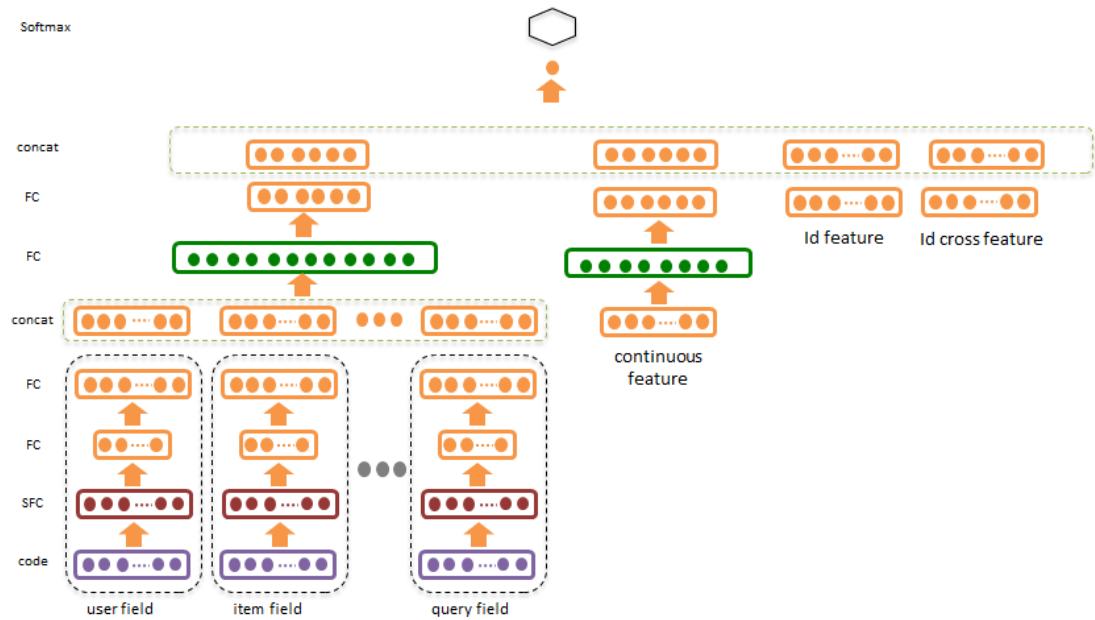
Fig. 1. A 4-layer FNN model structure.

### 3 我们的 Deep Learning 模型

在搜索中，使用了 DNN 进行了尝试了转化率预估模型。转化率预估是搜索应用场景的一个重要问题，转化率预估对应的输入特征包含各个不同域的特征，如用户域，宝贝域，query 域等，各种特征的维度都能高达千万，甚至上亿级别，如何在模型中处理超高维度的特征，成为了一个亟待解决的问题，简单的线性模型在处理高维稀疏特征存在比较好的优势，但是单一的线性模型无法处理特征交叉的问题，比如，我们在转化率预估时并不能单独只考虑宝贝维度的转化率，而更需要考虑用户到宝贝的转化率或者 query 到宝贝的转化率，这种情况下，我们使用单一维度的线性模型就无法解决现有问题，而需要人工构造高阶的组合特征来完成，会消耗巨大的计算量。

大规模 id 特征实时深度神经网络模型，可以处理上亿维度的 id 类输入特征，并通过复杂神经网络结构对不同域的特征(用户，宝贝，query)进行特征组合，解决了单一线性模型无法处理特征交叉的问题，同时也避免了人工构造高阶组合特征的巨大计算量。

深度神经网络通过构造稀疏 id 特征的稠密向量表示，使得模型能有更好的泛化性，同时，为了让模型能更好的拟合大促期间商品特征数据的剧烈变化，在深度网络的最后一层增加商品 id 类特征，id 组合特征和实时的统计量特征，使得整个网络同时兼顾泛化性和实时性的特点。



## wide model

- a. id feature: item\_id, seller\_id , 学习已经出现过的商品，卖家在训练数据上的表现。
- b. id cross feature: user\_id x item\_id , user\_id x seller\_id

连续值统计特征是非常有用的特征，Google 的模型是把 embedding 向量和统计特征放到同一个 DNN 网络中学习，但实验发现这样会削弱统计特征的作用。我们为统计特征专门又组建了一个包含 2 个隐层的网路，并且为了增强非线性效果，激活函数从 RELU 改为 TanH/Sigmoid。

## deep model

- a. 首先需要把离散特征 ( item\_id , item\_tag, user\_id , user\_tag , query\_tag ) embedding 成连续特征。
- b. 将 embedding 后的向量作为 DNN 的输入。考虑到最终线上预测性能的问题，目前我们的 DNN 网络还比较简单，只有 1 到 2 个隐层。

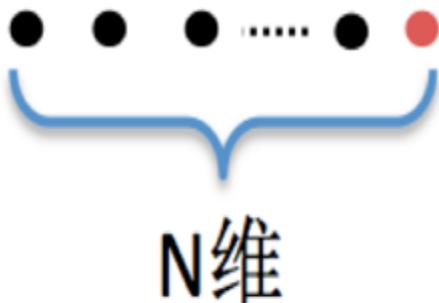
整体模型使用三层全连接层用于 sparse+dense 特征表征学习，再用两层全连接层用于点击/购买与否分类的统一深度学习模型解决方案：

第一层为编码层，包含商品编码，店家编码，类目编码，品牌编码，搜索词编码和用户编码。

在普遍的 CTR 场景中，用户、商品、查询等若干个域的特征维度合计高达几十亿，假设在输入层后直接连接 100 个输出神经元的全连接层，那么这个模型的参数规模将达到千亿规模。直接接入全连接层将导致以下几个问题：1. 各个域都存在冷门的特征，这些冷门的特征将会被热门的特征淹没，基本不起作用，跟全连接层的连接边权值会趋向于 0，冷门的商品只会更冷门。2. 模型的大小将会非常庞大，超过百 G，在训练以及预测中都会出现很多工程上的问题。为了解决上述两个问题，本文引入了紫色编码层，具体分为以下两种编码方式：1. 随机编码 2. 挂靠编码，下面将对以上两种编码方式进行详细的描述。

## 随机编码

假设某一域的输入 ID 类特征的 one-hot 形式最大维度为 N，其 one-hot 示意图则如下所示：



其中黑色为 0，只有红色为 1，该特征表达方式即为 one-hot 形式，在这种表达形式下有两个硬规则：1. 任何两个不同的特征都只有一个元素为 1。2. 没有交叉重叠的红色为 1 的元素。

倘若打破以上两个规则，让 one-hot 变成 six-hot，并且让两两 six-hot 中最多允许有三个为 1 的元素是重叠的，那么对 1w 维的每个 one-hot 特征都可以找到一个随机的 six-hot 特征与其对应，并且可以将这 six-hot 的最高维设置为 500，在这种情况下可以将 1w 维的 one-hot 特征压缩到 500 维，实现 20 倍的特征压缩，如果输入特征是 N 万维，则可以将其分成 N 段，并且在每一段里根据上述寻找到的随机码本进行特征压缩，最后 N 万维的 one-hot 特征可以

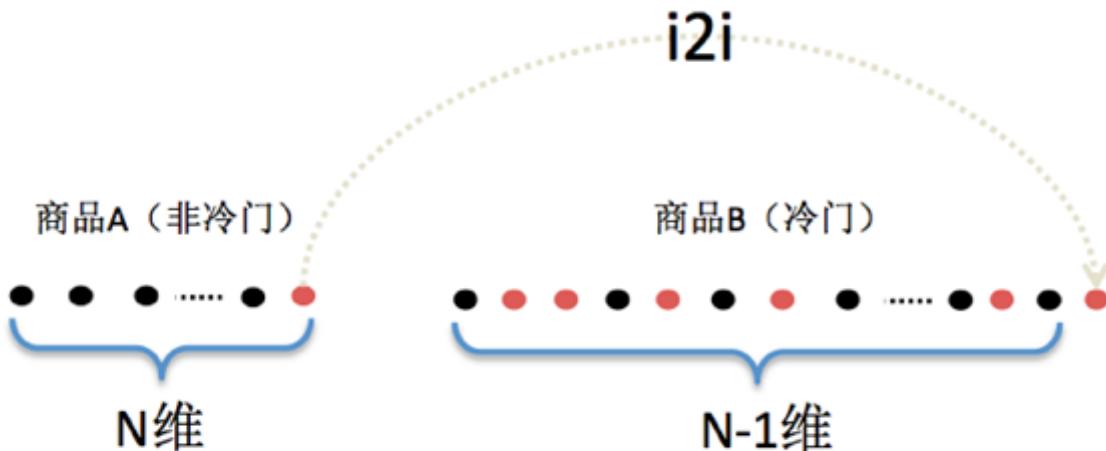
采用以上 six-hot 形式将其压缩到  $N/20$  万维，并且保证两两特征最多只有三个为 1 的元素是重叠的，示意图如下所示：



通过以上任一一种的编码方式，都可以实现模型大小将近 20 倍的压缩，使得百亿规模的模型参数压缩到了几亿维规模，但几亿规模参数的模型前向将会达到秒级，对于几十亿样本的模型训练，以及 CTR 模型的前向来讲将会是一个灾难，接下来将描述如何采用红色稀疏全连接层进行模型前向以及后向的时间压缩。

## 挂靠编码

上述的随机编码对用户域非常适用，但对商品域而言，虽然冷门商品会一定概率跟热门商品重叠一些为 1 的元素共享一些连接边权值，缓解了冷门商品越冷门的问题。但这里并没有利用好相似商品的信息，如何利用相似商品的信息，将冷门的商品与非冷门的相似商品建立共享权值？假设非冷门商品采用 one-hot 编码，冷门商品采用 M-hot 编码，如果冷门商品能通过 i2i 找到对应的热门商品，则该冷门商品共享一维该热门商品的编码，另外 M-1 维编码采用随机编码；否则，直接对 M 维进行随机编码。假设非冷门商品 A 的 one-hot 编码最后一位为 1，冷门商品 B 通过 i2i 找到相似非冷门商品 A，冷门商品 B 采用 six-hot 编码，则其挂靠编码示意图如下：

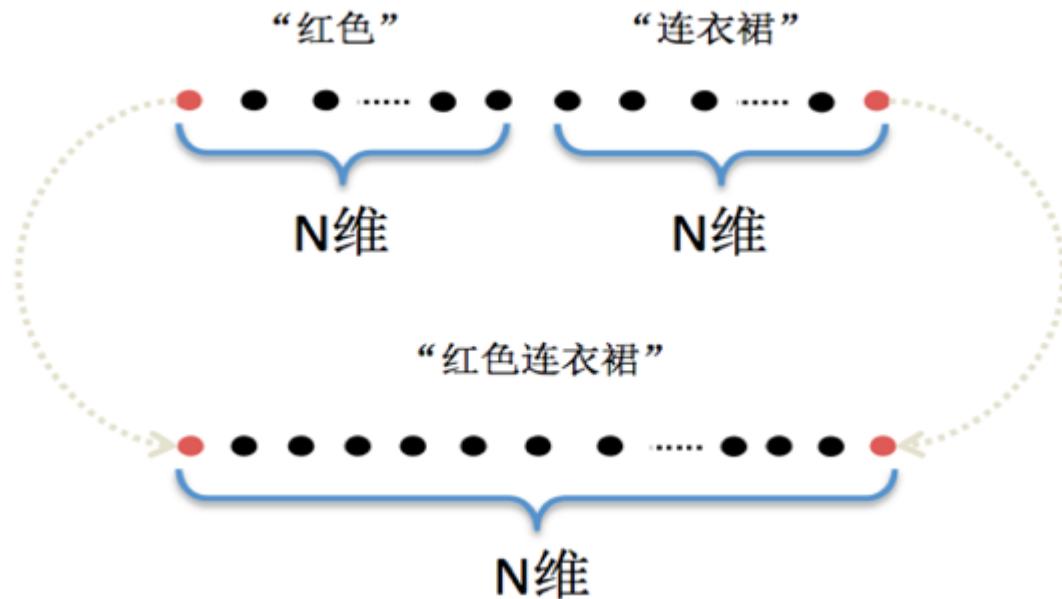


## 分词编码

上述的两种编码很好的解决了用户域与商品域的编码问题，但对查询域还是不够。在对查询域做处理的时候，往常模型往往会对查询短语先进行 ID 化，然后通过近义词合并一些 ID，再经过热门查询词统计来筛选出大概几百 W 的热门查询 ID，然后就会输入到模型中。

在以上的流程中，无法处理有重叠词语的两个查询短语的关系，比如“红色连衣裙”，“红色鞋子”，这两个查询短语都有“红色”这个词语，但是在往常的处理中，这两者并没有任何关系，是独立的两个查询 ID，如此一来可能会丢掉一些用户对某些词语偏好的 pattern。

基于以上观察，可以对查询短语首先进行分词，然后对每个词语进行 one-hot 编码，最后针对每个查询短语进行合并词语编码，也就是每个查询短语元素为 1 的个数是不定长的，它元素 1 的个数是由自身能分成多少个词语来决定的。分词编码的示意图如下：



从第二层到第四层组成了“域间独立”的“行为编码网络”，其中第二层为针对稀疏编码特别优化过的全连接层( Sparse Inner Product Layer )，通过该层将压缩后的编码信息投影到 16 维的低维向量空间中，第三层和第四层均为普通全连接层，其输出维度分别为 16 和 32。“行为编码网络”也可以被看做是针对域信息的二次编码，但是与第一层不同，这部分的最终输出是基于行为数据所训练出来的结果，具有行为上相似的商品或者用户的最终编码更相近的特性。

第五层为 concat 层，其作用是将不同域的信息拼接到一起。

第六层到第八层网络被称为“预测网络”，该部分由三层全连接组成，隐层输出分别为 64,64 和 1。该部分的作用在于综合考虑不同域之间的信息后给出一个最终的排序分数。

最后，Softmax 作为损失函数被用在训练过程中；非线性响应函数被用在每一个全连接之后。

## Online Update

双 11 当天数据分布会发生巨大变化，为了能更好的 fit 实时数据，我们将 WDL 的一部分参数做了在线实时训练。embedding 层由于参数过多，并没有在线训练，其他模型参数都会在线学习更新。

deep 端网络参数和 wide 端参数更新的策略有所不同，wide 端是大规模稀疏特征，为了使训练结果有稀疏性，最好用 FTRL 来做更新。deep 端都是稠密

连续特征，使用的普通的 SGD 来做更新，学习率最好设置小一点。

和离线 Batch training 不同，Online learning 会遇到一些特有的问题：

a. 实时 streaming 样本分布不均匀

现象：线上环境比较复杂，不同来源的日志 qps 和延迟都不同，造成不同时间段样本分布不一样，甚至在短时间段内样本分布异常。比如整体一天下来正负例 1:9，如果某类日志延迟了，短时间可能全是负例，或者全是正例。解决：Pairwise sampling。Pv 日志到了后不立即产出负样本，而是等点击到了后找到关联的 pv，然后把正负样本一起产出，这样的话就能保证正负样本总是 1:9

b. 异步 SGD 更新造成模型不稳定

现象：权重学飘掉(非常大或者非常小)，权重变化太大。解决：mini batch，一批样本梯度累加到一起，更新一次

# 第八章 交互设计

## 8.1 VR 电商购物

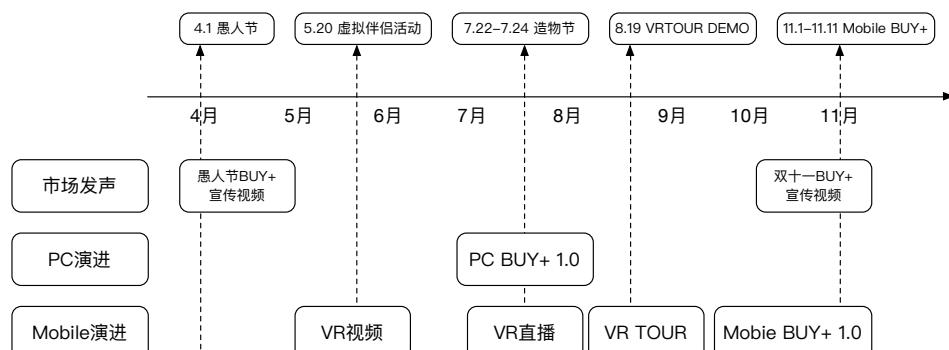
作者：宋五

### 前言

GM LAB 在 2016 年 3 月成立，是一个旨在探索最新电商购物体验的实验室。在探索 VR 购物的过程中，有两个需要核心解决的问题：一个是 VR 购物的产品形态是什么，另一个是 VR 环境下的店铺和商品怎么来。对于这两个问题，我们分别发起了 BUY+ 和 造物神计划 去解决。一直到双 11 结束，基于 BUY+ 探索 VR 购物体验，基于造物神去丰富 VR 素材，都取得了一定的结果。下面就详细介绍一下它们。

### 1 BUY+

BUY+谐音败家，是 GM LAB 启动的一个探索 VR 电商购物体验的计划。BUY+从四月一日开始走入公众视野，中间经过几次洗礼，最终孕育出了双 11 的版本并对外发布。下图是整个淘宝 VR 发展的过程，我们会重点讲一下 Mobile BUY+ 的设计思路以及对未来的思考。



## 1.1 Mobile BUY+

2016年11月1日，Mobile BUY+正式在手机淘宝客户端发布。这个项目目的是在双11全球化的背景下，打造移动端VR购物体验。我们希望做到在购物场景内有限移动，可以对商品进行一定程度的交互，同时完成下单支付这样的基础购物链路。这对我们的技术选型和内容制作都提出了巨大的挑战，下面我们就从这两个方面详细剖析Mobile BUY+。

### 1.1.1 技术选型

技术选型最难的就是VR业界发展的阶段还属于早期，有太多的问题需要解决，有太多的平衡需要选择。

首先，我们选择了Mobile以及Cardboard类的头显，为了让更多的消费者体验到VR，我们放弃了PC，也放弃了GearVR这样的独占平台。这个选择带来了规模的提升，但同时带来了大量中低端机型性能不足的问题。

接着，我们选择了全景视频作为场景方案，全景视频的正反播放作为场景中的移动方案。之所以没有选择3D建模，是因为考虑到高精模的建模成本，以及现阶段Mobile的性能问题。Mobile的交互手段少的客观现状，也很大程度上降低了全景视频方案对于自由度的伤害。更重要的是，全景视频也符合未来云端渲染的技术路径。

最后，我们为了集成进入手机淘宝，放弃了成熟的3D引擎。在手机淘宝引入体积庞大的3D引擎无法接受，为了让更多用户不需要下载新的APP也能体验到BUY+，我们不得不拥抱OpenGL去完成所有功能。

以上的选择给场景可移动，商品有交互，链路能闭环的要求带来了极大的挑战。移动端没有空间定位方案，缺少商品交互手段，VR的购物链路没有人做过。下面介绍一下在这样的选择下我们是怎么实现技术上的3个要求的。

### 1.1.1.1 空间移动



这次我们完成了两个方向的全自由移动，原理是把一个视频拍完以后，再制作一个倒播的视频，只需要在正向走动的时候播正向的视频，逆向走动的时候播放倒播视频。视频播放调用了系统原生的播放器，由于涉及到硬件解码，使得很多品牌的手机表现不一致，导致我们做了大量的适配。我们还碰到的一个问题是正播和倒播的视频切换的时候会出现微微的画面跳格，这个是因为 android 的 Seek 是基于关键帧的，如果将要 Seek 到的画面不在关键帧上就会出现画面跳格，我们通过添加关键帧缓解了这个问题。另外这里有一个有意思的结论，在我们做了大量的试验后，前进后退箭头的位置低于水平视线 15° 的位置时，用户感到最舒适。

### 1.1.1.2 商品交互

商品有交互，首先我们要标定出那些商品是可以做交互的。



BUY+的“人工智能”商品标定

如果在纯 3D 的场景模型里标定工作是非常简单的，但对于一个视频而言，要每时每刻都要标定商品，就非常有难度。假设视频长度 60 秒，每秒有 24 帧，每帧 10 个商品，如果用手工逐一标定商品位置，需要标定 14400 个点，另外在真实情况下每帧出现的商品是不同的，这个工作量肯定是受不了的。我们分析了下面的方案，最终确认了标定方案：

### 1. 图片识别

在视频第一帧对目标物进行标定，并将之作为模板，之后对视频逐帧进行模板匹配，依次标记出商品位置。

但是这个方案经过讨论很快就被否定，因为走到每个位置时看到的物品的面都不一样，这给图像识别带来很大难度。而且很多商品在场景中看上去并不大，对图像识别的要求实在太高，目前的技术储备无法实现。

### 2. 颜色识别

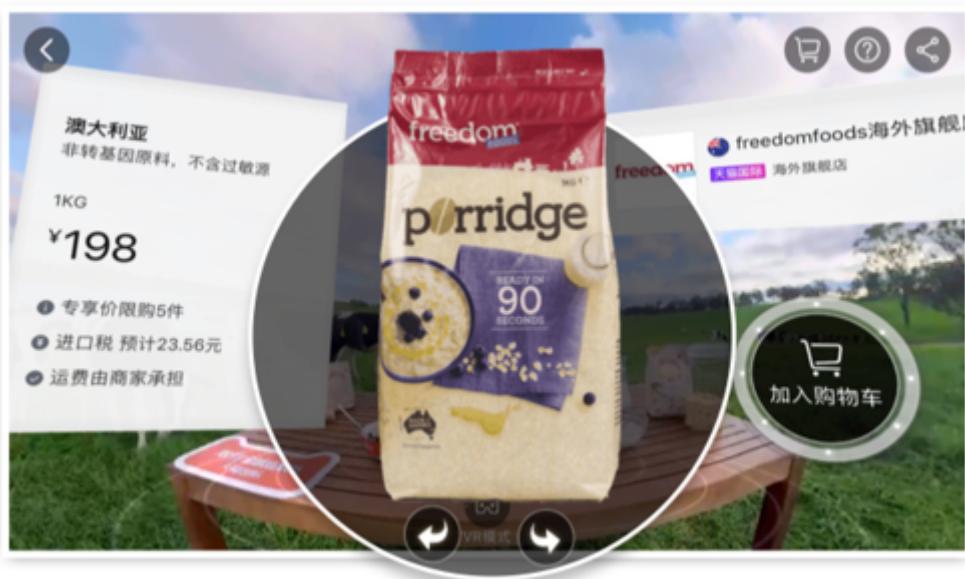
该方案需要将整个全景视频拍两遍，第一遍正常拍摄，第二遍在有商品的位置摆放一个有特殊颜色的物品，然后用颜色的识别来取出商品相应的位置。这里有两个难点，首先两次拍摄的速度必须是一样的，另外需要把场景中特殊颜色的位置全部用图像算法抠出来。我们真实尝试时发现了两个问题：一是如果代替物很小，在整个图片中的像素区域是不够的；二是很容易被其它颜色干扰，比如灯光，而且环境中可能会有与物体颜色一样的东西。经过多次尝试后，发现这个方案也是不可行的，因为现场环境是难以预料的。

### 3. 惯性标定

拍摄初始阶段，以摄像头作为原点建立参考系，分别测量出商品的坐标，并且标定视频的第一帧的数据。然后基于这些初始值以及摄像头的匀速运动计算商品每一帧的坐标。这个方案理论上可行，但实际操作的时候也遇到很大技术困难，首先是匀速运动的问题，虽然可以用轨道车可以解决，但同时也限定了内容的拍摄方案。其次是坐标系转化的问题，测量时一个很小的误差会导致坐标系转化产生很大的误差。最后是每个摄像头的参数不一样的问题，这导致使用不同设备，参数都要重新通过数据去训练。比如，我们尝试去训练完美幻境的一个设备，拿到的参数在其它设备上完全不可用。到此我们的第三种方案也基本可以认为是不可行的。

### 4. 半自动标定

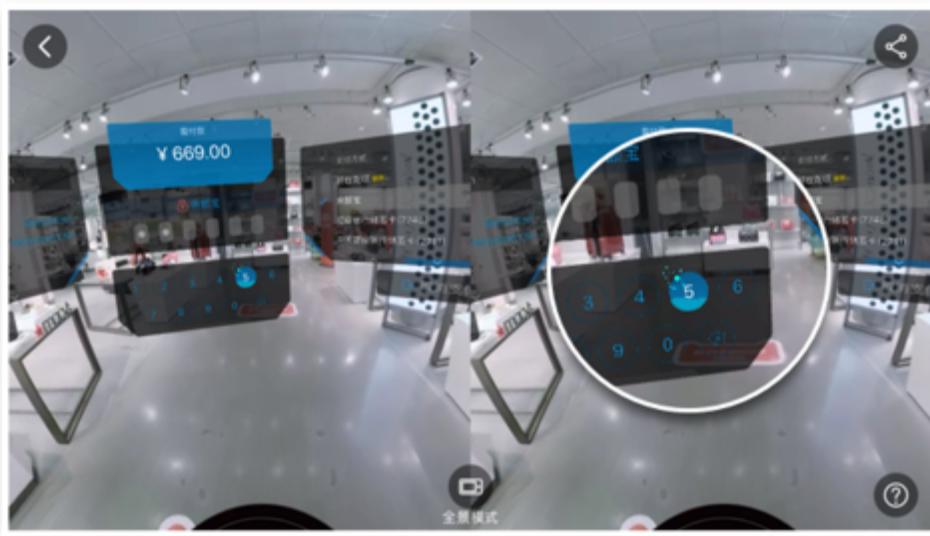
该方案是最终我们采纳的方案。在 PC 上打开视频用鼠标去跟踪每一个商品，记录鼠标的坐标，然后把二维坐标转化成 3 维坐标，这样每个场景的商品标定可以在 10 分钟内完成。但是如果每帧画面的坐标都记录的话，标定的数据会变得非常的大，我们最后决定 100ms 取一次标定数据，用户在这个 100ms 内停下，都会用这个标定数据，成功地把标定数据的大小压缩了 6 倍。



3D 商品的制作方案则是针对商品环拍一圈，每隔一定度数取一张照片，将其压缩为一个文件。当直接使用该文件展示商品时，经常遇到内存溢出问题，因此我们换用了类似视频编解码的方案去展示商品，最终把内存控制在合理的范围内。另外如果要求商品和背景融合完美的话，必须把商品背景扣成透明。我们做

了 150 个商品，每个商品 60 张图片，一共 9000 张图片，工作量非常大。我们通过绿幕的手段，结合一些图片提取的方法让效率变得很高。

### 1.1.1.3 链路闭环



一个完整的交易链路意味着从用户浏览详情开始，进行下单，到最终完成支付的整个流程，其中包含诸如浏览商品模型、选择收货地址、输入密码、选择银行卡等等操作以及各种异常流的提示，手淘一个下单步骤就有上千的测试用例，窥一斑而知全豹，足以见整个交易链路情况之多，交互之繁重。

要在 VR 中做到交易闭环，就必须建立一个强大的 GUI 系统去撑住上述繁重的交互。而业界并未对 GUI 形成标准，整个 VR 行业都在探索中，因此没有太多可借鉴的资源，研发这样的 GUI 系统可以说是很大的挑战。结合 Buy+项目的实际情况，我们认为一个 VR 中的 GUI 系统必须具备以下特征：

#### 1. 丰富的表现力以及低廉的学习成本

一个具有丰富表现力的 UI 系统意味着可以向用户传达更多语义，这个对比一下传统 GUI 与 CUI 的区别，感受就非常明显了。一个 GUI 系统的用户首先是 API 客户，最后才是普通用户，所以必须考虑到 API 客户对该 GUI 框架 API 的接受程度以及学习成本。所以我们最终采取了将 Android 的 View 和 IOS 的 UIView 转为纹理然后在 VR 中渲染的方案。这是个鱼与熊掌兼得的方案，一方面它成功地继承了系统 UI 本身具备的丰富表现力，另一方面使得 API 客户在 Android 与 IOS 平台的开发知识仍然可用。

#### 2. 足够的“VR Style”

第一点有一个弊病，如果严格地按照它的指导去设计该 GUI 框架必然造成 3D 空间 2D 交互的结果。所以必须把平面的 UI 打散，用 3D 的方式组装起来，也就是说需要把一个平面的 Layout 变为一个立体 Layout，只有这样做，该 GUI 框架才具备了“VR Style”化的基因。

### 3. 一个普适性的输入事件框架

一个 GUI 系统的核心功能，无非是输出与输入，输出的是用户看到的各种界面以及动画效果等，而输入则是用户通过不同外设进行的输入。而这块 VR 上面临的问题是输入设备没有统一标准，虽然 Daydream 试图改变这一局面，然而由于该规范在今年才提出，目前尚未形成工业事实上的标准，所以导致我们处在一个很尴尬的时期——未来可能出现主流输入设备，但却不得不面对现在身处 VR 输入设备的乱世。因此要让该 GUI 框架在这一块上具备足够的前瞻性以及防御性，那么必须设计一个普适性的输入事件框架去兼容 Gaze，CardBoard-Trigger，Daydream Controller，Gear-like，甚至是 VR gloves 这样的输入。当然了，鉴于设计这个框架的难度以及 Buy+项目的节奏，我们在双 11 只是完成了对点击、Gaze，CardBoard-Trigger 的支持，后续将会把提到的设备做一个兼容。

### 4. 高性能

对于任何系统，这都是个老生常谈的问题，但在 VR 中性能优化的重要性尤为突出。太低的帧率会导致用户产生明显的晕动症，因此我们在 GUI 这块也做了大量优化，比如像视锥体剪裁，Node 共享内存机制这样的深度优化，以及前文所提到的对 360 度图片的优化。

### 5. 必要的开发工具链

由于该 GUI 框架采用了复用系统 UI 但将其打散重组为 3D 结构的方案，导致这种 3D 的 Layout 复杂度骤然增加，所以必须要有一个在运行时实时调节这种 Layout 的工具来减轻开发过程中的复杂度去提高效率。作为一个工具，必须具有便捷性，因此我们借鉴了 Chrome Inspector 的远程调试协议在 H5 上完成了该工具。

## 1.1.2 内容制作

为了保障店铺和商品的全球化及真实感，我们选了 3 个国家 7 家最有特色的店铺现场实拍。为了让用户感受虚拟世界穿越的过程，我们也拍摄了乘坐标志

性交通工具前往店铺的过程。这些内容拍摄任务重时间紧，所以我们在程序还没有开发完成的情况下就开始了内容的拍摄。

拍摄团队基本上把能用的开发和市场人员全部用上了，在 10 天左右的时间里面完成。不仅拍摄过程辛苦，而且还需要等到拍摄和后期完成才能体验到实际效果，更是巨大的风险。拍摄过程中甚至遇到店铺里面的商品太小看不清楚，而不得不临时调整产品方案的情况。

经过了前期的铺垫，以及最后两个月的全员努力，双 11 的 BUY+完成了。它成为了第一个在线的 VR 购物应用，也是第一个百万人参与的大型 VR 体验。这中间我们收集了大量的数据和用户反馈，帮助我们更好的前进。比如视觉热点数据的分析，可以反向指导实体店铺商品的摆放，分析用户的兴趣点。



随着 VR 行业的发展，未来的 BUY+也会有更多的想象空间。但 BUY+只是一种购物的形态，它的发展依赖整个淘宝的 3D 化进程。缺乏足够的 VR 素材，它只是一个空架子。因此我们启动了造物神计划。

## 2 造物神计划

我们在 2016 年六月发布造物神计划，该计划旨在让 VR 内容的产生更加容

易。我们也借助双 11 的 BUY + 再次强调了造物神计划。目前造物神计划推出了三种不同级别的解决方案。

## 2.1 DIVA

DIVA 英文全称 Digital Interactive Visual Augmentation，我们叫做数字化交互视觉增强，利用现有常规视频和图像标准、结合算法识别与智能学习，形成一套初步成熟稳定的视觉增强技术。卖家和普通用户可以用常用的智能手机，自由拍摄、制作和分享可互动的 3D 内容。

DIVA 支持单轴采集，自动生成 360 度效果。多角度多轴采集，全角度拟合。另外还支持普通视频云端批量转换，iOS/Android/Web 跨平台兼容是它的优势，不过它最大优势是成本大为降低，每个商品 DIVA 成本在几十元到一百元不等。

## 2.2 3D 扫描

3D 扫描技术，扫描结果加入简单的人工干预就可以使用户无死角的看商品的任何细节，并应用于现在的详情。

3D 扫描技术相比 DIVA 有更强的交互能力，更丰富的商品展现维度，但同时价格更昂贵，每个商品要三百元及以上。

## 2.3 光场扫描

传统的 3D 扫描技术只能完成物品的结构、纹理的自动构建，但无法自动化处理材质（可以理解为物体的反光度）。这导致目前的 3D 扫描方案都无法全自动运行，人工添加材质不仅提升了成本，更会出现无法完美还原实物的问题。而通过光场扫描的技术，就可以解决这一问题。

目前这个方案还在探索中。

我们会以 BUY+持续探索，以造物神为核心打造 VR 购物的核心竞争力，带着新的体验步入新的时代。

## 8.2 淘宝直播在双 11 的互动实践

作者：丰火

### 1 简介

2016 年中国的智能手机覆盖率已达 58%，移动网络接入中 4G+Wifi 的占比也接近 90%，这是今年移动直播能迅速爆发的一个重要的前提条件。在当前全民参与全民娱乐的大背景下，移动直播能随时随地的发起和参与的产品特点，也充分满足了用户自我表达塑造个人品牌的需求，加上移动支付的快速普及，愿意为优质内容付费的观众已经成为了大多数，按照这样的链路在直播场景下内容变现的商业模式已经非常清晰，基本上具备流量资源的各大互联网厂商都在今年杀入了移动直播的领域。

另一方面，移动直播作为一个连接用户的平台，实时性极强，借助移动设备随时接入的特性，可切入的场景也更多，双向的交互方式对于包括电商在内的其他业务模式来说也是值得探索的新玩法，所以随着这波浪潮的兴起，我们也快速启动淘宝直播来探索电商+直播的各种可能的方向，经过大半年的探索也得到很好的收获，同时也为今年双 11 直播会场的上线打下了基础。

整个过程对产品和技术上均带来很大的挑战，本文将为大家解析整个过程中所遇到关键问题和解决方案。

#### 1.1 双 11 直播



从互动来看，需要支持多营销场景，典型如：九牛与二虎以及双 11 晚会直播间。九牛与二虎是一场精心设计的 PGC 制作的栏目，九个商家的同时直播，2 位明星随机走访明星各个直播间进行 PK 的玩法，直播间需要提供 9 个房间的实时互通以及切换的能力。双 11 晚会对直播的互动的玩法要求非常高了，除了常规的评论、红包、关注小卡等，还有 AB 剧，红黑 PK，点赞场景化等具体的业务需求外，还需要支持多屏互动。

从规模来看，要支持 50W~100W 的同时观看，同时要具备灵活的切换能力，对权益发放(如红包雨、砸金蛋)引发的 QPS 峰值要保障稳定的服务能力，这些也都需要从业务链路整体来梳理优化。总结一下核心面临的挑战主要有：

#### 1. 稳定性

50W~100W 的同时观看

卡顿率的优化

#### 2. 实时性

播放的时延控制在 <2s (排除人工引入的 60s 时延)

首屏打开时间小于 1s

#### 3. 同步性

互动的实时推送能力

内容和互动的同步

#### 4. 带宽成本

在不降低画质的情况下，节约带宽 30%~40%

下面将分别对这些问题进行分别分析和方案解析。

## 2 技术架构

直播作为一个实时系统，首要的目标是要将从主播端实时采集的音视频内容从网络推送到观众端，并保证整个链路的时延控制在 1s-2s 的范围内，整个流程上时延的优化是贯穿全局的重点。

## 2.1 基础架构



从架构图上可以看出，整个系统主要分为三个部分：

- 主播端：音视频的采集、编码、推流
- 阿里云：转码、截图、水印、录制
- 观众端：音视频数据的下载、解码、播放

## 2.2 架构重构

对直播业务进行了流程优化，使其具有更好的可扩展性和性能，主要体现在以下几方面：

- **开放性**

支持 1688、航旅、天猫系的直播接入，进行单独计费、结算。

- **可配置性**

直播管理流程化，一个流程有若干处理节点组成，不同的个性化接入需求，配置不同的流程。

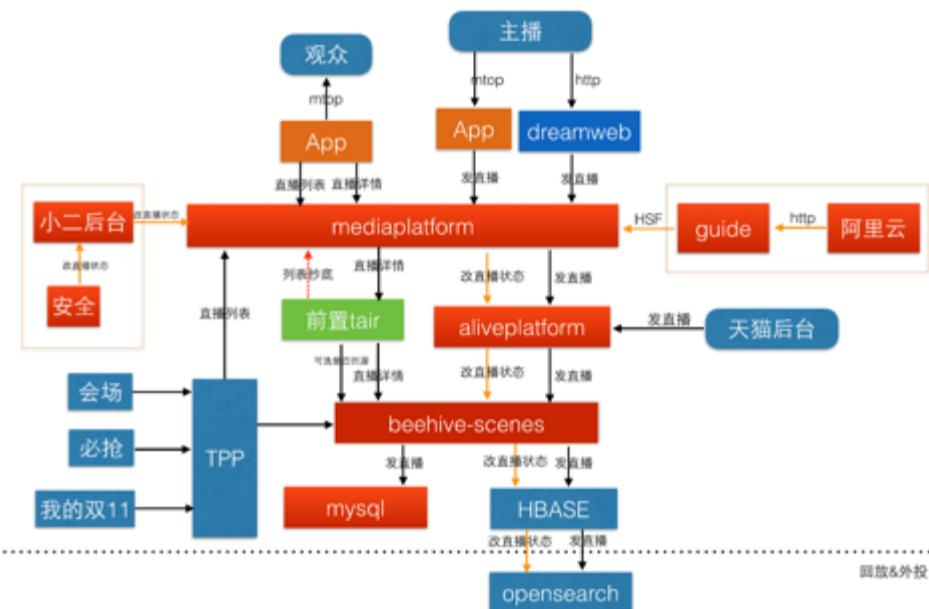
- **可复用性**

每个业务流程中，对通用组件进行复用，不同的接入需求只需新增个性化的

节点就可以快速支持。

- 高性能

优化了业务流程，增加本地缓存和前置缓存，双 11 晚会直播详情接口响应时间从 14ms 降低到 7ms，直播列表页从 114ms 降低为 54ms。

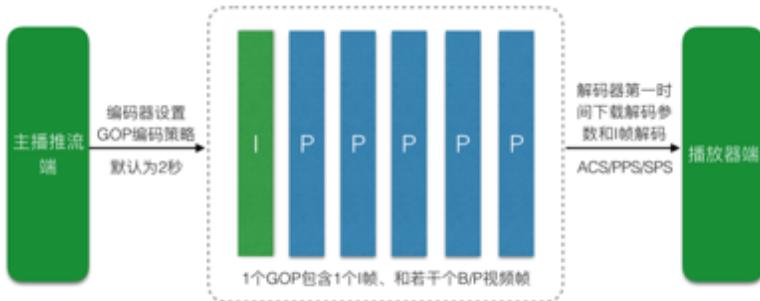


### 3 核心优化

#### 3.1 首屏秒开

首屏秒开是指播放端用户从进入淘宝直播间到出现视频画面的时间在 1 秒钟之内，在极短的时间内呈现直播视频画面、已缩短用户的等待时间；在这 1 秒钟之内需要经历 DNS 解析、TCP 建立连接、RTMP 协议握手、流媒体数据接收/解析、视频解码、YUV 数据渲染等一系列的环节。

推流端视频 H264 编码按照每 2s 一个 I 帧进行编码，然后按照 RTMP 协议打包发送到 CDN 边缘节点服务器，CDN 服务器对视频数据从 I 帧开始缓存；只缓存最后一个 GOP，即后面的 GOP 数据覆盖前面的数据，以保证视频流的实时性；流程如下：



CDN 会在服务器上缓存最后一个 I 帧开始的 GOP 视频数据 ,此视频数据最大为 2s(默认关键帧间隔为 2 秒) 当播放器端 HTTP 连接到 CDN 节点服务器后 ,CDN 服务器会将缓存的 GOP 视频数据全部发送到播放器端。除了优化 GOP 缓冲之外 , 播放段也进行了多项优化 :

- 业务依赖优化

在拉取直播间列表时直接返回播放地址 ,点击进入直播间时优先拉取音视频流来解析播放 ,收到首帧画面后进行后续业务逻辑 ,包括在线人数、评论列表等

- 播放预读缓冲优化

为了判断视频流、音频流的格式 ,播放器一般会预读一部分数据来尝试解码确定确定音视频的编码方式 ,而在淘宝直播中视频 ( H264 ) 、音频 ( AAC ) 编码方式都是确定 ,完全可以避免这样的预读耗时。

双 11 期间最终数据如下 ,其中首屏平均加载 <1000ms

日期	图像首帧平均加载时间 ios	首帧平均加载时间 android
2016/11/03	1162	886
2016/11/04	1187	895
2016/11/05	1164	875
2016/11/06	1160	855
2016/11/07	1156	881
2016/11/08	1201	886
2016/11/09	1226	893
2016/11/10	1100	871
2016/11/11	1239	939
2016/11/12	1142	900
2016/11/13	1126	880
2016/11/14	1156	889
2016/11/15	1076	804
2016/11/16	1083	808
2016/11/17	1084	804
2016/11/18	1067	802

## 3.2 变速播放

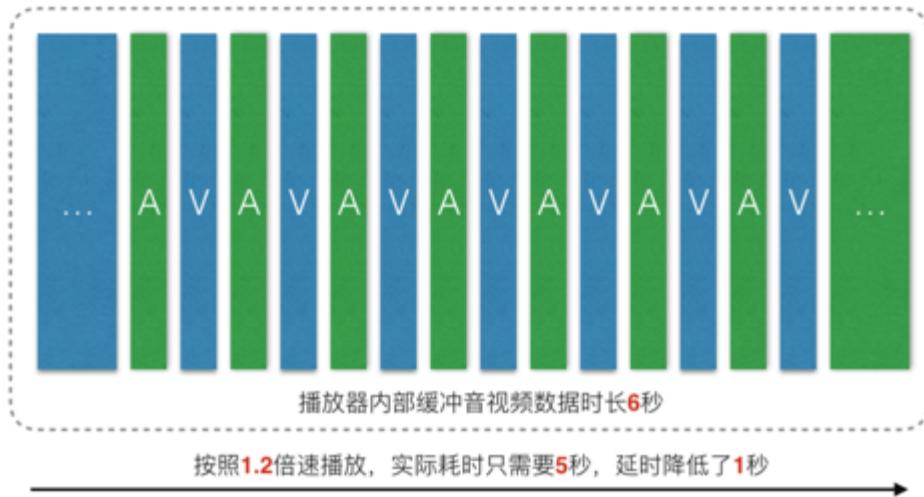
在移动互联网络下，网络发生卡顿、抖动非常普遍情况下会播放器内部缓冲越来越大、整体累计延时也就越来越大，此时需要提高播放速率(如 1.2 倍速~1.5 倍速)来降低累积延时；当播放器内部缓冲区数据越来越低时，需要降低播放速率(如变速到 0.7~0.9 倍速)来等待缓冲数据，避免频繁的卡顿与缓冲等待；

播放器的默认同步机制大多都是视频同步音频的，即以音频为基准时间轴，所以只需要改变音频的播放速率即可实现此目的，视频会自动进行同步；前提是在改变音频播放速率的同时不能降低声音播放体验；

播放器缓冲去大小策略数据如下图：



当网络发生抖动，播放器内部缓冲的音视频数据为 5 秒，此时累计延时就有 5 秒 在正常播放速率(1 倍速)下 播放完 5 秒的音视频数据实际也需要 5 秒，累计延时不会降低，随着累计延时越来越高用户体验也会越来越差；当播放器前次发生卡顿恢复后，播放器下载缓冲的音视频数据达到 6 秒以上时，播放器会开启加速播放到倍速播放，播放速率会在 1.1~1.5 倍速之间浮动，此时缓冲内部的音视频数据会在短时间内播放完毕，如 1.2 倍速播放 6 秒视频数据会在 5 秒内播放完毕，会缩短 1 秒钟的累计延时

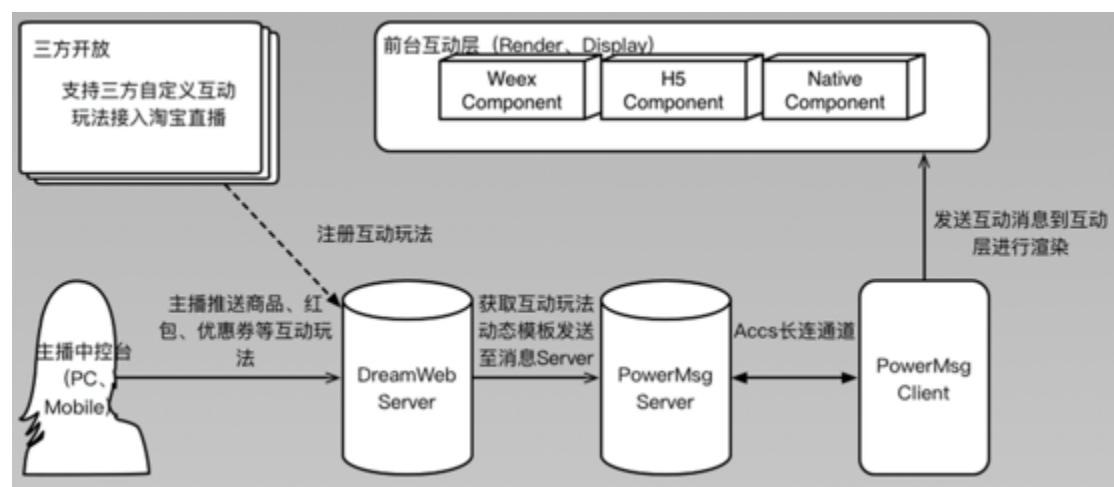


同样道理，当网络发生抖动导致播放器内部缓冲的音视频数据小于 5 秒时，播放器也会将播放速率调慢至 0.8~1.0 之间以使缓冲长度恢复到合理长度以避免卡顿。

## 4 互动玩法

### 4.1 技术框架

互动是直播的关键功能，为了满足丰富的玩法，直播互动玩法的框架必须具备良好的开放性和定制化。一个典型的互动玩法交互的框架图如下：



实现互动玩法的开发和直播间功能完全解耦，承接双 11 期间红包、优惠券、进店小卡、关注小卡等多种玩法



最终的数据包括评论、点赞在内的总互动次数达到 42.5 亿次，互动率超过 20%，提升了整个直播间的互动氛围表现。

## 4.2 帧同步

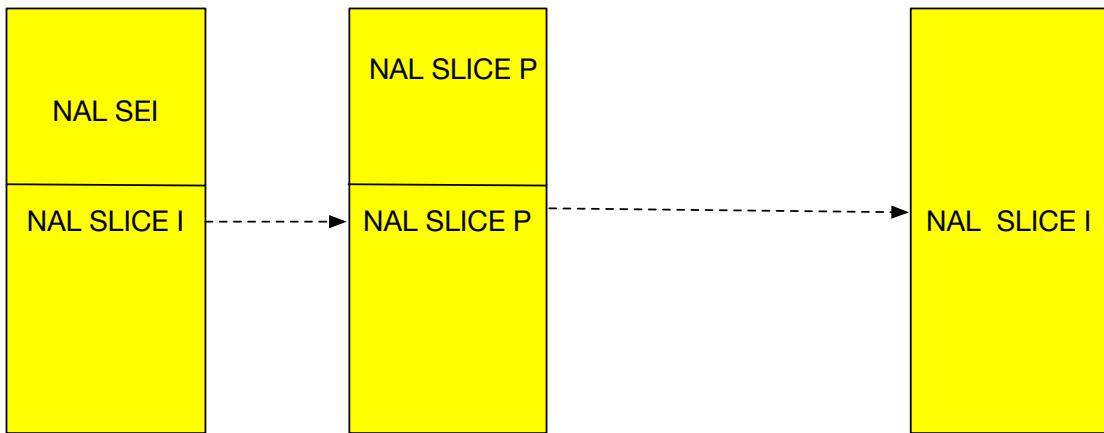
多屏互动的难点在于如何保证来自媒体流（音视频数据）和数据流（互动玩法）在时间上是精准匹配，传统的做法是基于提升消息推送时间和到达率，但在实际场景因政策管制的原因最终播放的媒体流实际上有一定的人为延时（一般是 60s），这样带来的问题：用户看到画面时可能互动消息已经提前出现或者还没有收到，这都会导致互动效果大打折扣。

为了解决这个难题我们提出一套基于视频帧的解决方案：将互动内容作为视频的特殊帧封装到视频流里面。下面详细介绍下方案实现：

在 H264 视频编码标准中，帧是由一个或者多个 NAL 单元组成，而 NAL 单元又分为多种，其中有一些特殊的 NAL 单元其自身可以不携带任何音视频信息，如 SEI\_NAL。H264 标准中，允许 SEI\_NAL 单元由用户按照标准格式自己定义填充数据，且解码器在收到用户自定义的 SEI\_NAL 时，默认不会对其进行处理。因此，我们将互动信息封装成一个 SEI\_NAL，将其打入到视频帧中并由

主播端推送出去，最终播放器配合进行 SEI\_NAL 的解析上报：

NAL\_SEI 帧的构造如下图所示：

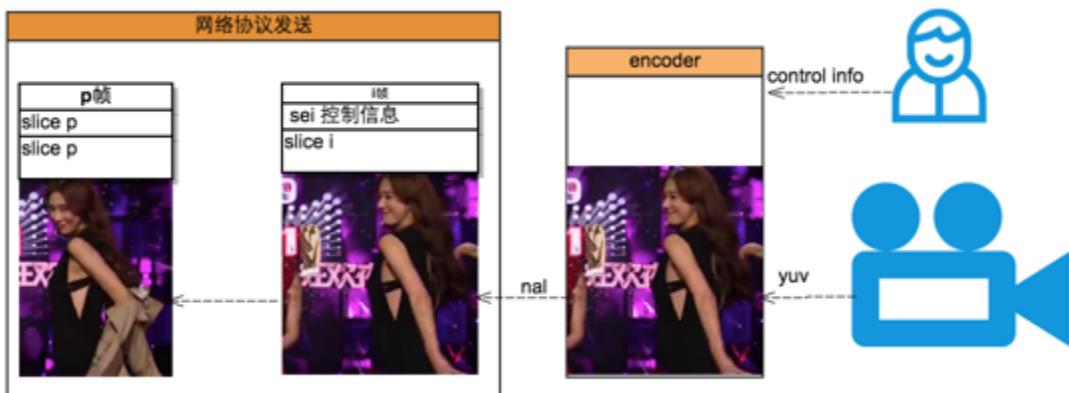


每一次控制信息 SEI 的大小不会超过 255 字节，对视频文件大小的影响很小。

帧同步互动主要包括两部分：

- 摄像机推流时将控制信息写入视频帧

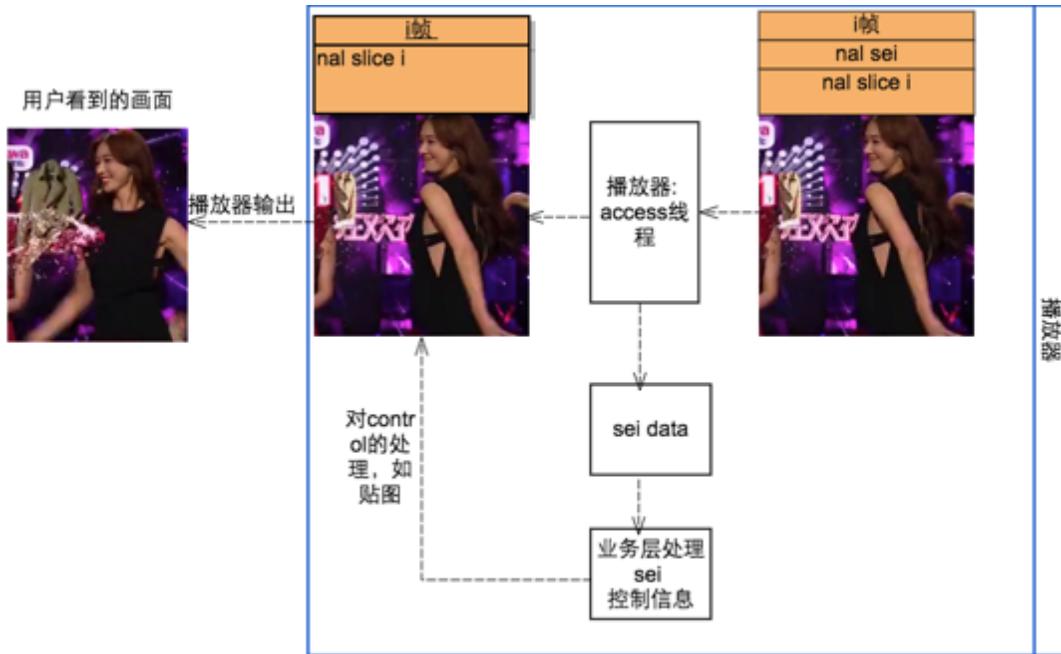
如下图所示，在林志玲扔衣服的时候，摄像机采集到扔衣服这一帧画面，由工作人员手动触发编码器，在当前帧写入控制信息， video 数据和控制数据由推流端发送协议发送到服务器，服务器对此控制信息不做任何处理的下发。由此完成控制信息的写入分发。



- 播放端收到控制信息解析上报

播放器在收到帧之后，会判断帧中是否有控制信息，如果有，则会将控制信息 sei 从视频帧中分离出来上报给业务层，由业务层决定当前控制信息应该做什么操作。比如在用户收到林志玲扔衣服的 sei 时，会上报业务层该 control，由

业务层决定应该在界面显示一件林志玲的衣服。从而达到的效果就是视频里面林志玲在扔衣服，用户的界面上显示出了一件林志玲的衣服。

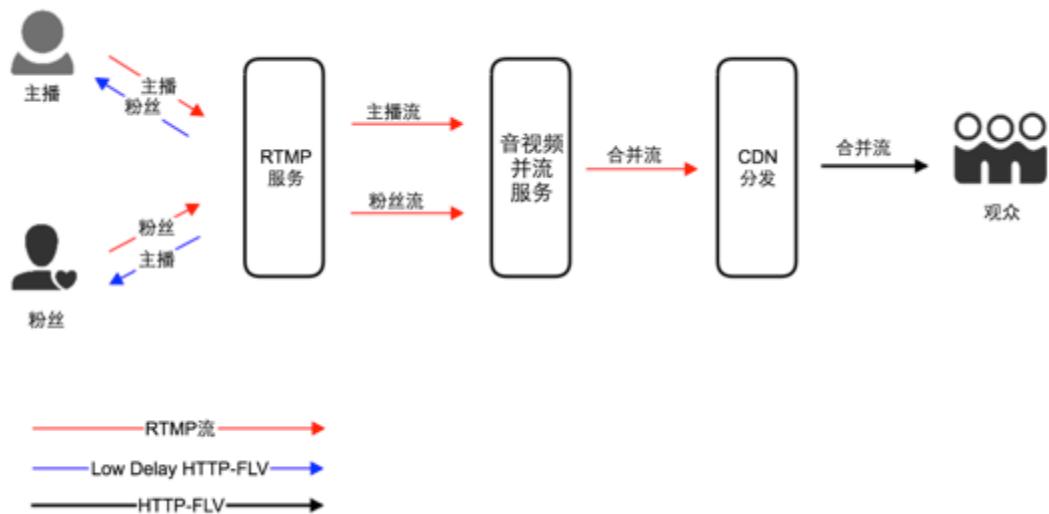


通过基于视频关键帧的互动方案，能够确保用户在视频的同一画面收到一致的互动数据，达到较好的互动体验，同时具备良好的扩展性。

### 4.3 直播连麦

直播的应用场景主要是两个重要的参与角色：主播与观众，而直播应用相比点播等传统应用的最大优势是：主播与观众的实时互动能力能给观众带来的更多的参与感，是除了传统互动方式包括文字，点赞，礼物等方式之外一种重要的互动能力，也是直播领域在未来的竞争重点。

直播互动连麦的流程是指：主播在直播过程中，可以邀请一个或多个观众或者其他主播进入直播间，可以和主播进行双向音视频通话，所有的观众在直播间可以看到他们的互动过程。而引入连麦功能，最大的技术难点：在对现有直播系统架构不做太大调整的情况下，提供低延迟的视频通话体验，同时兼顾未来的功能和架构演变，比如提供多方连麦功能等。经过综合考虑后设计的架构如下：



连麦中主播和连麦粉丝各推一路音视频流到服务器，同时拉取对方的音视频流进行播放，观众则观看连麦双方合并后的流。需要注意的一点是，为了保证连麦双方低延迟，主播和连麦粉丝拉取对方的音视频流均是低延迟流，而观众为了保证流畅度，拉取的是有延迟的流。

方案的优点：

- 1、并流功能在服务端来实现，可以降低手机端的性能压力和带宽压力，同时也更容易扩展连麦人数。
- 2、对现有直播架构改动不大，基本采用和现有直播架构相同的协议体系，开发成本低，稳定性有保证。
- 3、延迟低。实测端到端双方延迟可以达到 500ms，其中服务端延迟平均只有几十毫秒，实测数据打破大家一直以来对 rtmp 协议延迟过大的认识。

连麦的延迟是双方通话的一项重要指标。整个过程的延迟主要有两类：

- 固有延迟，包括采集延迟、数据处理延迟、编码延迟、解码延迟、系统处理延迟、展现延迟等

针对固有延迟，在保证质量的情况下尽量降低部分参数的设置。如在 H264 算法可以采用延迟更低的 baseline (会降低一些视频质量)，采用延迟更低音频算法代替 AAC 等降低固定延时。

- 动态延迟，包括网络传输过程中由于丢包、拥塞等原因引起的延迟。

为了解决网络抖动引发的播放卡顿，在播放端引入一定的缓冲平滑播放过程避免频繁的卡顿。为了能根据网络情况动态确定延迟和流畅的最佳平衡点，我们引入了两个算法来解决：音频变速不变调算法和 jitterbuffer 控制算法。

在解决了上述问题后，最终的基础连麦能力得以完成，在双11造势期间我们也提供“全民连连看”作为直播的一种创新玩法，主打“一个正经的交友视频在线互动游戏”，取得了非常好的效果。连连看最终效果如下：



## 4.4 整体稳定性

互动权益在直播间的推送到达成功率是互动稳定性的最重要的指标，下图是整个直播互动在双 11 期间的数据表现：



为了达到整体稳定性的要求，主要在下面几个方面来进行优化：

- 全链路埋点监控

从消息的推送到互动层的渲染都进行完整的数据埋点，快速定位整体成功率。

- 消息的 push&pull 结合

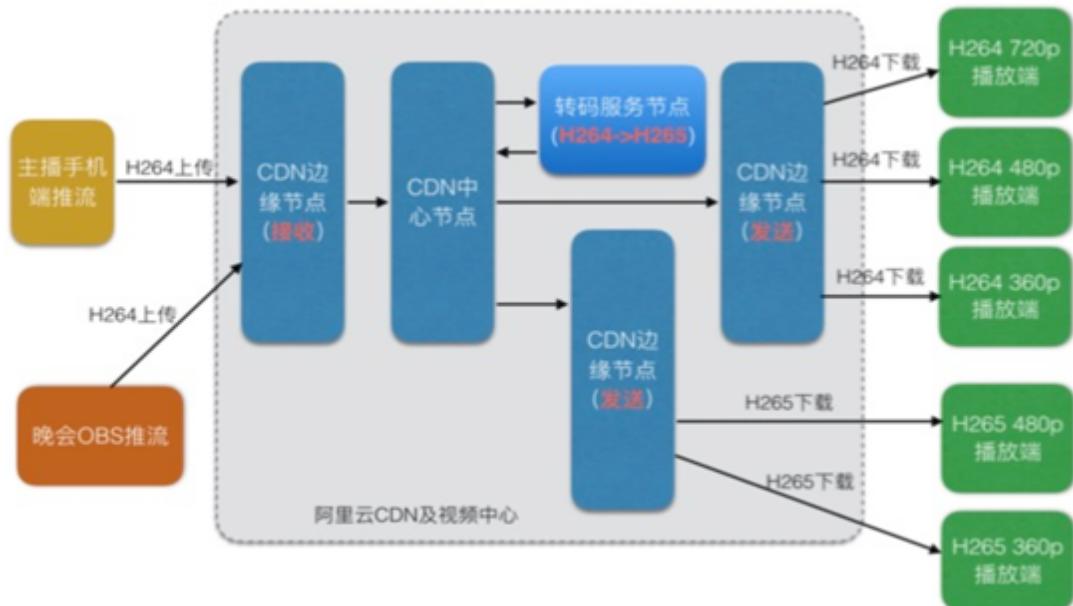
将高频的评论消息和低频的状态消息区别开，用不同的策略来拉取

- H5 互动玩法的优化

除了对“红包雨”页面的占用内存进行优化外，将常用的资源通过 zcache 进行提前推送降低渲染时因资源拉取失败导致互动玩法无法展示

## 5 带宽成本

为了满足带宽节约的目标，在播放端引入 H265 的解码算法，推流端不受影响依然保持 H264 推流，H265 转码由云端转码服务完成，整体架构如下：



双 11 晚会开启 H265 转码后，整体节约带宽 30%左右。

## 总结

今年是淘宝直播在双 11 的首次亮相，整个平台高峰时支持近 50W 的同时在线，同时支持了包括在会场和直播间内丰富多样的互动玩法，表现稳定。未来除了在功能上进一步加强整体运营的能力外，也需要完善全链路的数据监控能力，持续优化核心体验。基于阿里云底层的直播服务共同打造完整移动直播解决方案，提供包括直播 SDK、消息通道、互动玩法等一体化开放平台。

# 8.3 2016 双 11 前端突破

作者：天猫前端团队

## 前言

2016 年天猫前端相比去年有了非常多不同维度的突破，主要可以分为四大类大类：

1. 稳定性、监控
2. 极致的性能优化
3. 业务创新 / 平台建设
4. 技术创新 / 互动

## 1 稳定性、监控

商品到每个用户浏览的每个环节都有监控，尤其在针对消费者体验上的 TES，让前端在消费者真实浏览的过程当中也能够有更进一步的分析在不同环境下消费者实际的体验。以及从服务器 Wormhole 渲染层进行了一系列的稳定性、监控。

### 1.1 Wormhole 双 11 会场稳定性保障

Wormhole 承载了 2016 天猫，淘宝等 BU 的双 11 会场页面渲染职能，整个体系链路横跨了前端开发、存储、渲染、CDN 等不同的领域，在双 11 活动中系统可访问性变得尤为重要。为了确保双 11 会场页面能够在各种极端场景下还能够顺利打开，wormhole 做了非常多的稳定性容灾方案 包括去除单点依赖

HTTPS 防劫持，动态调节 CPU 利用率、数据链路 OSS 依赖容灾以及集群完全无法服务后的终极打底方案等。

### 1.1.1 异地多活模式

Wormhole 是一个 CDN 前置的无状态静态页面渲染服务，从简单和维护成本角度出发，没有接入集团的单元化部署策略，而是采用异地多机房部署的模式，借助 CDN 回源策略实现异地多活。

### 1.1.2 HTTPS 改造防劫持 0-0

为了避免用户访问链路被 IDC 劫持，Wormhole 完成了全 HTTPS 的改造，包括从 CDN 到用户端以及从 CDN 回渲染源站的链路，实现完整的 HTTPS 访问链路，避免中间被劫持，造成用户访问故障。1 源站负载动态调节

在双 11 的场景下，为了保证 0 点时刻用户端能够及时访问到最新的页面（活动发布、价格透出、秒杀等），CDN 的缓存和用户端缓存都要求即时实效，在瞬间产生大量的回源请求，给 Wormhole 渲染源站带来短时间的高负载，在保证渲染实时性的同时，Wormhole 根据当前系统的负载动态的使用历史渲染缓存（本地磁盘、远程 OSS），来降低应用的渲染负载，保障服务的稳定性。

### 1.1.3 数据链路 OSS 依赖容灾

Wormhole 依赖阿里集团的 OSS 作为数据存储服务，包括模块、页面以及投放数据，以尽可能简单的架构来保障服务的可扩展性，提升可维护性。对于 OSS 的依赖，Wormhole 提供了两层稳定性保障：

1. OSS 异地主备部署，避免单点故障；
2. OSS 数据本地内存、磁盘容灾备份，在 OSS 故障异常时，可以直接使用本地缓存数据提供服务，保障整体服务的可用性；

### 1.1.4 页面终极打底方案

为了避免在 Wormhole 渲染服务异常的情况下，我们针对渲染的结果也做了两层的容灾策略，即

1. 渲染结果本地磁盘容灾；
2. 远程 OSS 容灾。

在应用异常时，可以通过本地 Tengine 的容错设置，读取本地磁盘的渲染结果提供服务，或者在极端严重的情况下，通过切换 CDN 回源的方式，将 CDN 整体回源到灾备 OSS，使用历史渲染页面继续提供服务。

## 1.2 TES 技术体验平台

TES 技术体验平台，致力于提供让技术及业务同学都能准确及直接观察、了解及分析业务前台页面状态的能力。从稳定性、性能、体验三个维度来全面掌握业务的运行及使用情况，同时从多维度定义一套符合用户实际体验的指标体系，来描述一个页面的使用运行状态。

### 1.2.1 突破传统

纵观在前端领域，都会存在性能监控及页面监控等多类系统，但存在几个未考虑或不足的问题：

1. 页面加载性能只是页面“好坏”的一个维度，缺乏对页面体验的关注，缺少可以衡量用户真实体验的指标
2. 只能反应单页面问题，无法以业务的视角查看整个业务、整条链路的体验和质量
3. 完全面向开发者，非技术的业务同学无法通过简单的方式去了解自己业务的使用状态
4. 缺少对一个业务从稳定性、性能及体验的综合性可量化评分体系

TES 吸取了基础性能及监控系统的功能，同时提供了体验维度的监控及数据展现，分别从 FPS、CPU、电量使用等可能影响体验的因素进行监控和汇总。提供全局实时监控能力，监控页面的稳定性，如资源加载、接口异常、JS 报错及 crash 等。丰富性能展示维度。

### 1.2.2 业务归一

TES 平台的设计是从业务的角度出发，当页面发布上线后 TES 自动能分析及归纳页面所属的业务，同时从垂直业务及活动进行分类，方便开发及业务同学进行查看对应业务的运行情况。



双 11 期间通过 TES 系统实时发现多处线上图片链接、资源连接以及接口稳定性等问题，同时提供了多端的监控埋点，统计分析接口数据打底以及链路情况等。

### 1.2.3 加载性能

双 11 会场在客户端内以 WEEEX 状态呈现，同时支持 WEB 浏览器访问，从整体可以观察到双 11 所有页面的整体性能的情况，是否达标，同时从在对单个页面有较详细的性能分析。



## 1.2.4 体验性能

目前所有的页面监控对体验都没有比较好的采集以及评估方式，目前 TES 会结合各类影响体验的因素，如 FPS、大图大资源、CPU 执行情况以及耗电量等，同时后续会完善更多合理的指标来完善体验指标的整理，目前以 FPS 数值为主。



# 2 极致的性能优化

从 H5 到 React Native，再到 Weex，阿里在性能、体验优化的道路上尝试了各种方案，从去年双 11 到今年双 11，在底层上我们完成了许多重要优化，保障双 11 的秒开目标。

## 2.1 目标

### 页面加载时间 pageLoad

天猫 Mobile 性能优化的核心衡量指标是 pageLoad，它是一个 webview 实现的类似 JS 的 onload 的事件，在 iOS 下对应 webViewDidFinishLoad，在 android 下对应 onPageFinished。

.onload 和 pageLoad 的差异主要是在 onload 事件里同步执行的 js、加载的资源都会被计入到 pageLoad 内，而 onload 事件里执行代码通常就是特意做了懒加载处理，我们可以在 onload 里包一层 setTimeout 来解决：

```
window.addEventListener('load', ()=>{
  setTimeout(()=>{
    //load assets\data...
  }, 0)
})
```

天猫 pageLoad 目标经历了多个阶段，从 15 年 2 月之前的 2s 到 15 年 5 月的 1.5s，一直到目前的 1s。



## 2.2 首屏可交互时间

pageLoad 和 onload 一样都是一个浏览器实现的通用事件，和 onload 一样事件触发不等于页面真实加载完成。我们需要又一个可以衡量用户真实感受的首屏渲染时间指标，最终选择了基于代码埋点，代码中加入埋点并上报，在模块加载容器上添加 setter 来监听模块加载状态变化的方案尽可能减少对业务的侵入，且不受模块删减、调整影响，模块的大致写法如下：

```
class Module{
  constructor(){
    ...//数据加载 & 渲染
```

```
this.readyState = 'rendered'; //标识该模块已经渲染完成  
...//事件绑定  
this.readyState = 'complete'; //标识该模块已经处于可交互状态  
}  
}
```

## 2.3 体验指标 FPS

除了以上 2 个描述加载速度的指标以外，我们还实现了一个 FPS 指标，用用户衡量页面的滚动流畅性。对于 FPS 的统计方案和业界一致，根据 requestAnimationFrame 之间的时间差值来计算。

因为用户浏览的过程会持续滚动，而埋点上报的数据量有限，我们选择在客户端完成统计计算，上报计算后的统计学指标的方案，包括：最小值、最大值、均值、众数和中位数，而上报的时机选在了每一次页面离开的时候，实现上结合了 visibilitychange 和 pagehide 这 2 个事件，保证覆盖各个移动端环境。

## 2.4 底层优化点

### 2.4.1 HTTP2

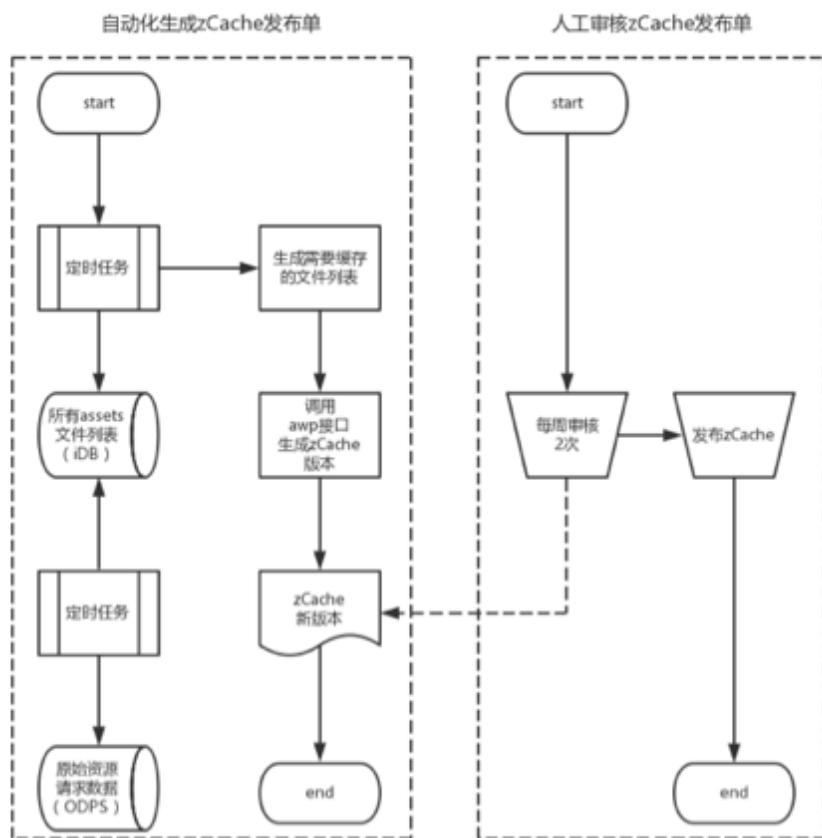
HTTP2 可以带来多路复用、头部压缩、服务端推送能力、请求优先级控制等优点。这一年我们完成了阿里 CDN 和统一接入层的 HTTP2 部署，同时在客户端优化了 HTTP2 的命中率，目前天猫核心域名的 HTTP2 命中率已经达到 97%+，再加上我们完成了大范围的域名收敛，整个页面的域名收敛到了 5 个以内，更好地发挥了 HTTP2 多路复用的优势，最终 HTTP2 带来的性能收益在 250ms+。

## 2.4.2 HTTPDNS

域名解析在移动端是一个非常耗时的过程，且根据网络情况波动很大，手淘猫客对此做了优化：由服务端下发一份域名和 ip 的对应关系，然后客户端直接通过 ip 访问，跳过域名解析的过程，在这套现成技术上，我们完成了天猫核心域名的下发，同时完成域名收敛，保证页面用到的域名均在 HTTPDNS 执行列表之中，最终性能提升 100ms+。

## 2.4.3 自动化的预加载

手淘猫客提供了 2 种预加载方案 zcache 和 packageApp，目前用的主要还是 zcache，因为平台默认提供的是人肉的发布流程，导致线上使用率很低，即使用了也难以坚持长期更新，为此我们开发了一套自动化的预加载策略：按页面访问流量进行排序，自动预加载前 N 个页面用到的资源，大致流程如下：



## 2.5 现阶段成果

### 2.5.1 Weex

在技术上 Weex 整个页面只有一个 bundle.js，所有的数据都异步获取，通过预加载技术将 bundle.js 提前下发到客户端以后，除了数据以外整个加载过程就不再存在任何 assets 加载，双 11 会场中基于 Weex 的会场占比：99%+，整体秒开率在 90% 以上：

- Weex 主会场秒开率
  - 秒开率峰值（00:00）：整体 96.9% (278.8ms)、iOS 99.4% (146.4ms)、Android 93.4% (411.1ms)
  - 秒开率均值：整体 94.4% (367.6ms)、iOS 99.0% (177.0ms)、Android 91.8% (473.3ms)
- Weex 所有会场秒开率
  - 秒开率峰值（00:49）：整体 92.4% (490.7ms)、iOS 97.4% (291.6ms)、Android 87.5% (689.8ms)
  - 秒开率均值：整体 83.9% (651.9ms)、iOS 94.5% (368.0ms)、Android 78.6% (797.4ms)

### 2.5.2 H5

天猫 H5 页面主要分为 2 种模式，一种是静态页面+异步数据，另外一种是服务端同步渲染：

1. 对于第一种模式，运用的页面较少，以天猫首页为例，落地各个优化点以后秒开率达到 90%+，pageLoad 均值在 800ms+。
2. 对于绝大部分服务端同步渲染的页面，目前优化后的 pageLoad 极限在 1.1s 左右，和同步渲染的首屏内容的复杂度正相关，在 1.1s~1.6s 之间波动。

## 3 业务创新 / 平台建设

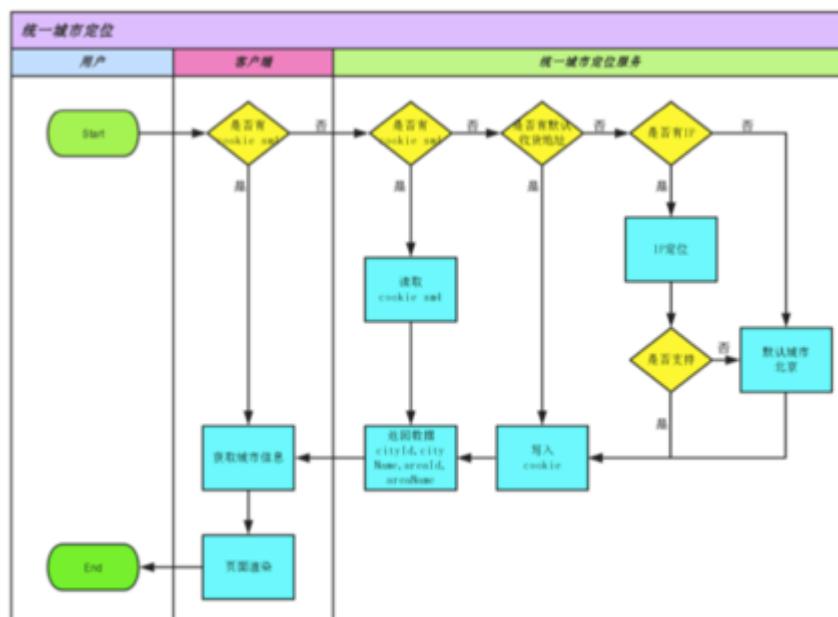
面对日益复杂的业务上透过产品化的方式解决当前的问题，从过去静态化千人一面的货架形式走向千人千面的个性化，商品个性化到楼层个性化，全网单统一投放到区域定投等，结合现有平台方式解决更复杂的业务问题。

### 3.1 区域化服务

各个行业统一区域化逻辑，形成一套通用的可扩展的区域化服务，包括区域优惠、区域库存、区域定投等，而这些能力都依赖城市定位，所以我们要做的第一步就是统一各个行业的定位逻辑，这里的统一包含几个层面：

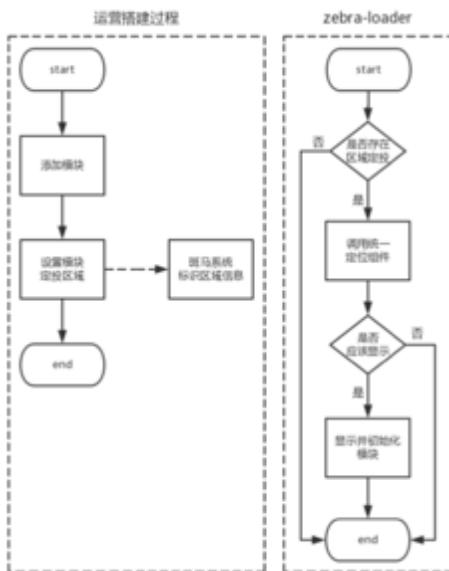
#### 3.1.1 定位逻辑的统一

定位的判断来源主要包括以下几个：用户选择、ip、默认收货地址、gps、打底城市，需要统一这几个判断来源的组合规则和顺序，统一后的规则如下图所示：



### 3.1.2 双 11 业务落地

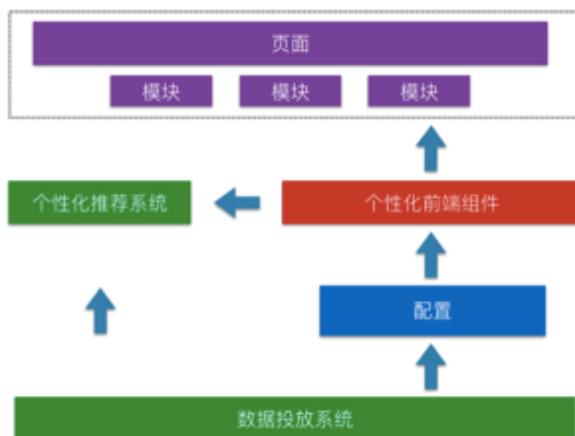
区域化底层能力之上，实现了斑马平台级的模块区域定投功能，运营可以针对任意一个模块开启区域定投，并在双 11 会场中投入使用，业务流程如下：



### 3.2 简单高效的个性化产品方案

一个模块是完整的最小业务粒度单位，通常用一个模块来展现一批相关联的商品，如『国际食品』会场的『早餐拍档』就是一个模块，为用户展现麦片、奶粉等早餐相关商品。

由于是否个性化并非由开发者决定，因此最好的方式就是透过共通的组件来处理数据，增大复用性，大致流程图如下：

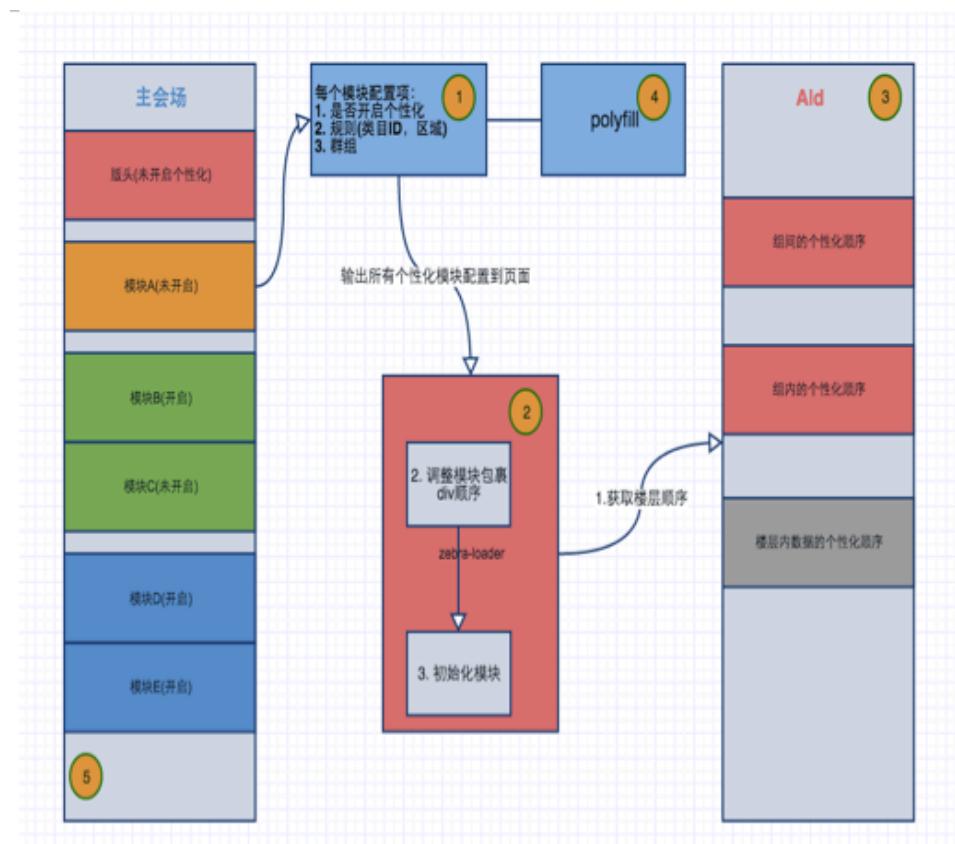


当然，这只是数据流通的基本原理，要实现以上功能，还有以下细节需要处理：

1. 去重问题，同页面使用多个模块但是内容不能重复。
2. 合并请求，优化性能。
3. 容灾打底方案，任何请求都有因为网络抖动、延迟、接口错误所造成挂掉的情况。

### 3.3 模块间个性化

以上模块内个性化效果还是有限，若能在模块顺序个性化能够达到更好的效果，如下图所示：



## 4 技术创新 / 互动

### 4.1 Weex

1754 张双 11 会场页面中( 统计了天猫和淘宝 ),Weex 页面数为 1747 占比 99.6%。手淘 iOS/Android 分别有 83.5%/78.3% 版本 ( UV ) 启用了 Weex 会场 , 手猫 iOS/Android 分别为 91.7%/87.9% 版本 ( UV ) 。 Weex 覆盖了包括主会场、分会场、分分会场、人群会场 等在内几乎所有的双 11 会场业务。

### 4.2 业务支撑

双 11 的会场结构大致为 : 会场框架 ( 框架 + 主会场、全部会场、必抢、清单、我的双 11 ) 、分会场、其他会场 ( 分分会场、人群会场等 ) 。

### 4.3 会场框架

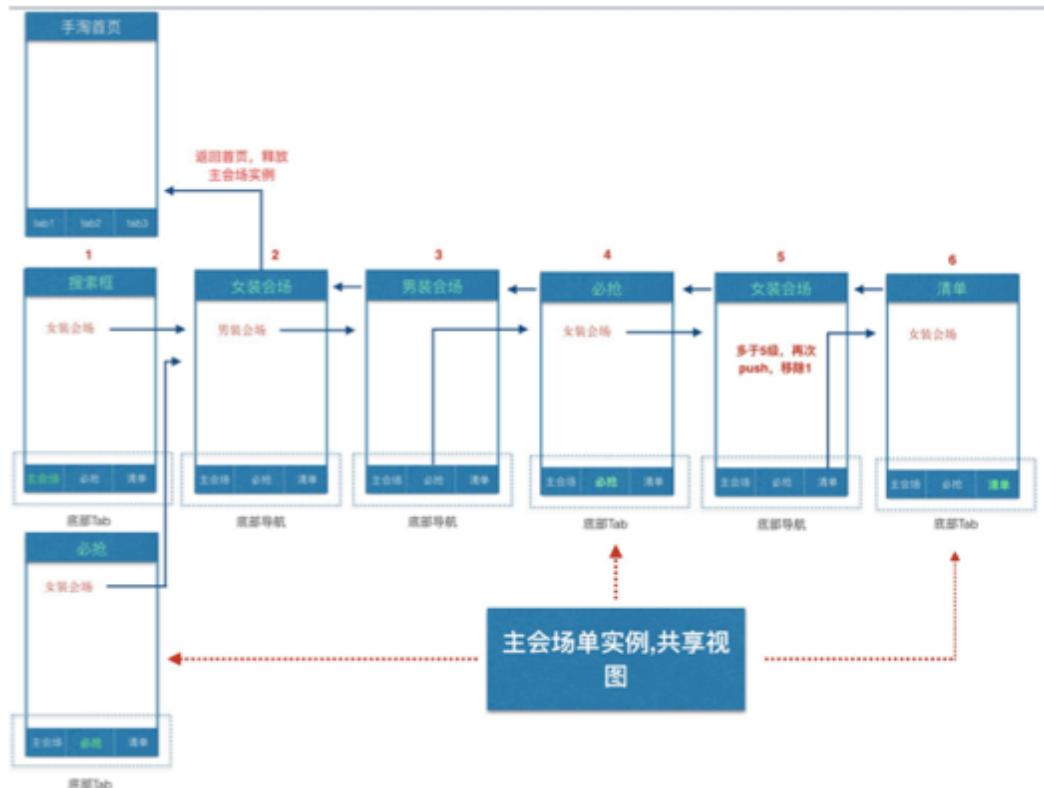
Weex 支撑双 11 业务首要解决的是会场框架及其交互形式 :

1. 交互主流程 :

1. 非 push 方式 ( 框架 Tab 切换 ) : 主会场 - 全部会场 - 必抢 - 清单 - 我的双 11

2. push 方式 : 主会场 - 分会场 - 主会场

2. iOS 考虑到内存开销 , 需严控打开的主分会场 weex 页面 , 定为 n 级可配 , 默认为 5 ; 同时 iOS 会场框架为单实例 , 也是出于内存的考虑 ; Android 连续打开 30 级以上 Weex 页面 , 未见内存异常增长 , 无需特殊方案



会场框架交互

## 4.4 稳定性保障

Weex 的首要挑战就是稳定性，或者说保障 Weex 会场最大限度不降级。

## 4.5 会场压测

### \* 压测场景

1. 5 个 200 坑位的普通会场页面，1 全景图会场页面，1 曝光埋点压测页面，1 直播会场页面

2. 页面中提供链接，能够按顺序进行 push 跳转

### \* 压测方案

1. 主链路（首页->店铺->详情->购物车）做一遍操作，让内存缓存占满，记下内存值 M0

2. 进入 Weex 页面，滑动到底部，滑动到顶部，记下 M1；点击跳转按钮，跳转到下一个页面
  3. 重复步骤 2，让所有的页面进行压栈；全景图->p1p2p3p4->直播->p1p2p3p4->UT
    - \* 压测结果：iOS 通过，Android 通过
1. 测试过程手淘手猫均未出现因为压栈导致的 Crash，稳定性可以保证；
  2. Android 低端机压栈过多会导致体验较差，之后也会引入类似 iOS VC 层级控制；

## 4.6 稳定性数据

2016.11.11，Weex 在手淘中的 Crash 占比情况：

- iOS 1.46% ( TOP7 )
- Android Java Crash 0.85% ( TOP13 )、Native Crash 1.72% ( TOP8 )

# 5 双向互动&AR 跨屏的天猫双 11 晚会前端技术

2016 年的天猫双 11 晚会互动与往年一样是由天猫互动技术团队负责开发，在去年红黑 PK 的经典玩法的基础之上增加双向 AB 剧与 AR 跨屏抢新衣的两个玩法，同时增加了舔屏版和后台揭秘版两个场景。对于前端技术的挑战就是需要让呈现的效果『更实时、更真实』，玩法和场景上的增加也要求开发团队『更高效』。

## 5.1 更实时（时间表）

本次晚会升级了前后端的数据传输方式，以轮训为打底，通过 powermessage 和关键帧技术使数据更轻量准确的实现前后端通讯，来实现复杂的高实时性互动。

1. powermessage

通过 powermessage 技术，建立客户端与后端的长链接，待后端数据有增量变化时，通知前端做出相应的互动状态变更，这样做的好处是减少轮训带来的压力，实时性高。

## 2. 关键帧技术

传统的，我们通过人工切节目表的方案，来实现流内容与前端页面的实时同步，必然会有少许误差，通过关键帧技术，OBS 推流时，在流中增加一段增强信息 SEI，播放器底层捕获后，以通知形式抛给 iOS 和 Android 层，客户端这层获取对应的字符串，组织成 json 通过 argoWebView 提供的 hybird 接口传递给前端，即可实现由流内容触发前端互动操作，达到可控的、精准的实时操作。

## 5.2 更真实（AR）

为了给用户更奇特的用户的体验，实现酷炫的互动营销手法，晚会的一个亮点就是由 AR 技术实现的“明星丢衣服”环节，用户在看电视直播的过程中，明星在电视端丢出一件衣服，主持人会口播提示用户使用手淘或者猫客的摄像头对准屏幕，客户端通过 AR 识别技术进行识别和定位，识别成功后，用户在手机上会看到衣服从电视中浮出，砸碎手机屏幕的特效，效果见：

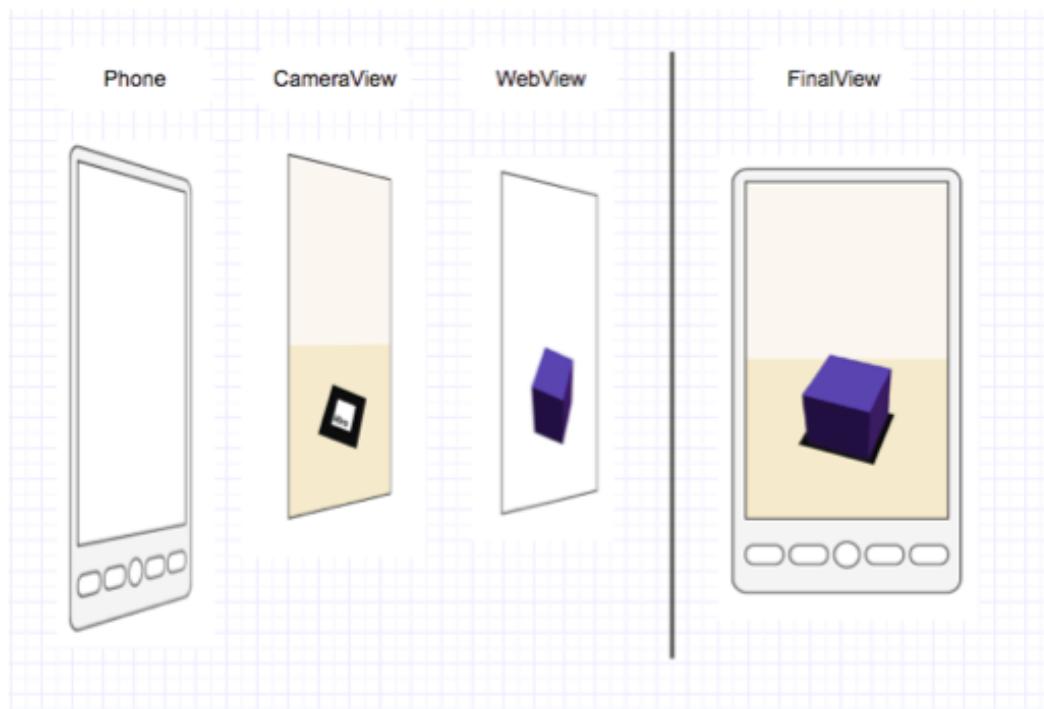


<http://download.taobaocdn.com/freedom/29927/pic/p1b1lk13um1h1p15ti1v99utf1scv4.gif>

### 5.1.1 技术方案

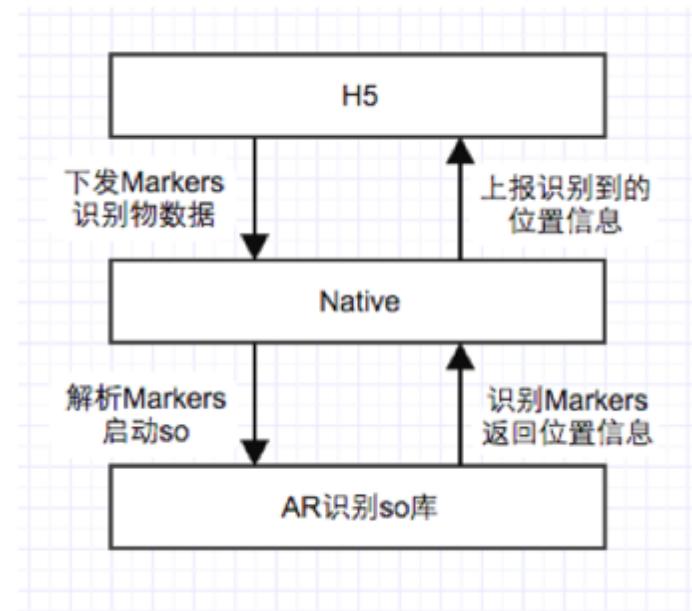
AR 识别功能可以划分为业务层和识别层两部分，这两部分具体实现的功能如下：

1. 业务层：主要包括摄像头数据展示层和 WebView 层。摄像头数据展示层主要负责进行摄像头数据的展示。WebView 层位于整个界面的最上层，负责 Markers ( 识别物 ) 数据的下发以及根据识别层返回的识别物位置信息进行渲染（可以根据前端的需要进行 2D , 3D 以及各种特效的实现）。
2. 识别层：基于 Markers 的位置识别，实时扫描摄像头中的帧数据以判断是否命中识别物，若命中则将识别出的 3D 位置信息返回给上层业务方。



AR 识别方案

### 5.1.2 识别流程



手机天猫 AR 识别流程

## 5.2 更高效（工具）

开场动画、摇一摇、AR 丢衣服等多处动画特效有非常高的要求，素材制作上，给设计师带来挑战的同时，也给前端同学带来不小困难，即酷炫的动画过于复杂，如果按照视觉稿一帧一帧还原的话，需要耗费极大的人工成本，而且一旦动画出现需求变更，对前端来讲简直就是灾难。去年设计师使用的是 Flash 进行开发，前端工程师采用自研的 Flash 插件导出 Hilo 动画，而 2016 年设计师使用 After Effector(AE) 制作动画，我们又通过写 ExtendScript 脚本导出动画数据，优化、解析动画数据后，使用 canvas 来播放动画，形成了新的 AE2Html5 的动画生成工具。

# 6 WebGL 大规模应用，天猫双 11 推出首个 3D 版狂欢城

天猫在每年双 11 预热期都会发布汇集众多品牌、效果酷炫的狂欢城互动页面，2016 年和往年不一样的最大亮点是首次同时推出了 2D 和 3D 两个版本。3D 版狂欢城开启了电商行业首次在大场景下应用 Web3D 技术。

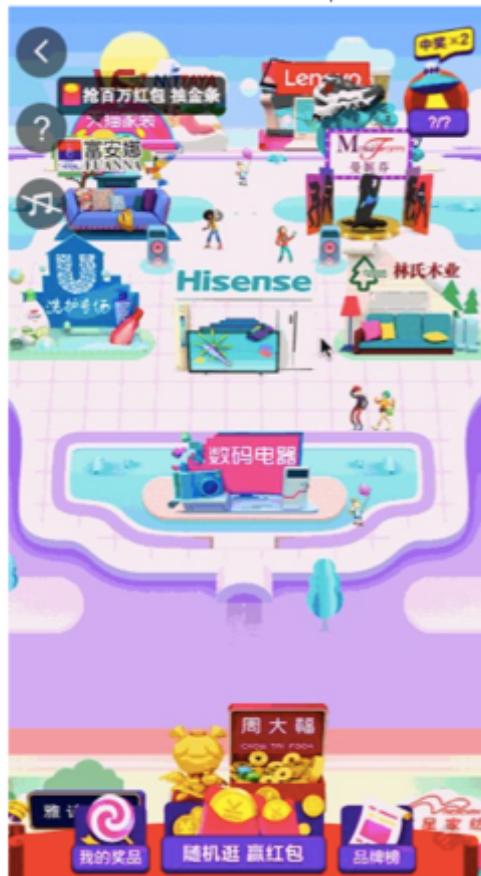
## 6.1 设计方案

在做场景展示选择的时候，我们首先想到的是 3D 巡航场景，和多数第一人称游戏类似，使用第一人称的视角在品牌场景中边逛边玩，但这种方式面临品牌透出和性能挑战。

经过多次简化场景，保留品牌模型，设计师同学提出一种近似滚筒的场景——纸片风格的广告牌方案。这种纸片自带一些 3D 感，实际上我们看到很多 2D 图案，因为有透视效果，当我们审视的角度和设计师设计的视角吻合时，会感觉那是 3D 的。Billboard 在现实生活里也比较常见，比如大型购物地带，繁华路段等。

Billboard 的策略很简单，尽可能正对着用户，正对着人流（流量）。在一些建筑的边界区域，Billboard 能够利用多个面达到尽可能大的曝光。

## 6.2 技术方案



在技术实现上，CSS3 和 Canvas 也能模拟类似的效果。同时，根据手机淘宝和手机天猫当前的 WebView 的环境，CSS3 的支持会比 WebGL 好很多，但是在 CSS3 在处理这种大场景时存在一些无法回避的问题——分割贴图、形状拼接后的“黑边”问题、transform 的白屏黑屏问题。

在底层实现上，WebGL 比 CSS3 更稳定，CSS3 更适合一些小场景。另外，在 3D 表达上，物体的形状和纹理是我们最能直接感受的两个特征，形状就是我们看到的样子，纹理就是表面的附着。除了这些光照，环境（雾、雨），反射，材质都能通过和材质表面的颜色做若干次运算，最终通过算出来的 RGBA 来使物体呈现丰富的外观。在构建复杂 3D 物件形状上，CSS3 会显得捉襟见肘。所以，最后我们选择 WebGL。

# 8.4 Weex 在双 11 会场的大规模应用：业务支撑、稳定性保障和秒开实战

作者：鬼道

## 前言

Native 开发的诸多亮点中，流畅体验和系统调用是最多被提及的。流畅体验体现在页面滚动/动画的流畅性，背后是更好的内存管理和更接近原生的性能；同时又是 Web 的痛点：资源首次下载、长页面内存溢出和滚动性能、动画性能、传统 web 性能(如 JS 执行效率)。Native 有丰富的系统调用能力，而 Web 痛点在于：W3C 标准太慢，有限的设备访问能力，API 兼容性问题较严重，如 Geolocation 在 Android Webview 中可用性很差。

Web 开发同样有诸多亮点，其中最耀眼的当属发布能力和规模协作。Native App 商店审核周期长（尤指 iOS）；应用更新周期长，iOS 稍快大概能达到一周更新率 60%-80%，Android 同样的更新率要 2 周甚至更长。而 Web 在合适的缓存机制下一分钟可达到 99%+。浏览器内核 webkit 提供了相对一致的底层运行环境，html/js/css 控制页面的结构/行为/样式，uri 连接不同的页面，有了这些基础设施，大规模的业务复用和人与人的分工协作变得相对轻松。

同时今天阿里诸多客户端已经面临包大小接近临界值，大促活动页面（H5）体验较差等一系列问题。结合 Native 和 Web 技术亮点，同时又能解决阿里遇到的业务问题，这就是 Weex 诞生的客观环境。

2016.11.11，在 1754 张双 11 会场页面中（统计了天猫和淘宝），Weex 页面数为 1747 占比 99.6%。手淘 iOS/Android 分别有 83.5%/78.3% 版本（UV）启用了 Weex 会场，手猫 iOS/Android 分别为 91.7%/87.9% 版本（UV）。Weex 覆盖了包括主会场、分会场、分分会场、人群会场 等在内几乎所有的双 11 会场业务。

在这样的应用规模下，工作和目标是：

1. 业务支撑，支撑住双 11 需求，这是最基本的要求，详见下文“业务支撑”一节
2. 稳定性保障，Weex 引发的问题第一时间响应并处理，不留到双 11 当天，详见下文“稳定性数据”一节
3. 秒开实战，稳定当先力争高性能，双 11 正式主会场秒开率冲到 97%，所有会场秒开率冲到 93%，详见下文“秒开数据”

2016 双 11 会场的感受可查看原始录屏文件：[WIFI](#) | [4G](#) | [3G](#) | [2G](#) | [无网络](#)。录屏时主会场已经是预加载版本，其中 WIFI 和 4G 效果接近，2G 效果取决于数据的网络请求速度（录屏时数据请求约 3.9s），无网络情况下打底数据来自前一次成功请求。流畅性可查看[该视频](#)，左起为 H5、iOS Weex、Android Weex。

## 1 目标

展开 Weex 双 11 细分目标：

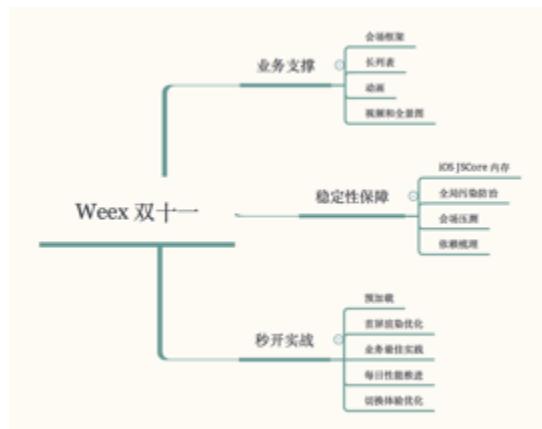


图 - Weex 双 11 细分目标

## 2 业务支撑

支撑住双 11 的业务需求，是 Weex 必须要迈过的坎。

双 11 的会场结构大致为：会场框架（框架 + 主会场、全部会场、必抢、清单、我的双 11）、分会场、其他会场（分分会场、人群会场等）。

### 2.1 会场框架

Weex 支撑双 11 业务首要解决的是会场框架及其交互形式：

1. 交互主流程：
  - a) 非 push 方式（框架 Tab 切换）：主会场 - 全部会场 - 必抢 - 清单 - 我的双 11
  - b) push 方式：主会场 - 分会场 - 主会场
2. iOS 考虑到内存开销，需严控打开的主分会场 weex 页面，定为 n 级可配，默认为 5；同时 iOS 会场框架为单实例，也是出于内存的考虑；Android 连续打开 30 级以上 Weex 页面，未见内存异常增长，无需特殊方案。

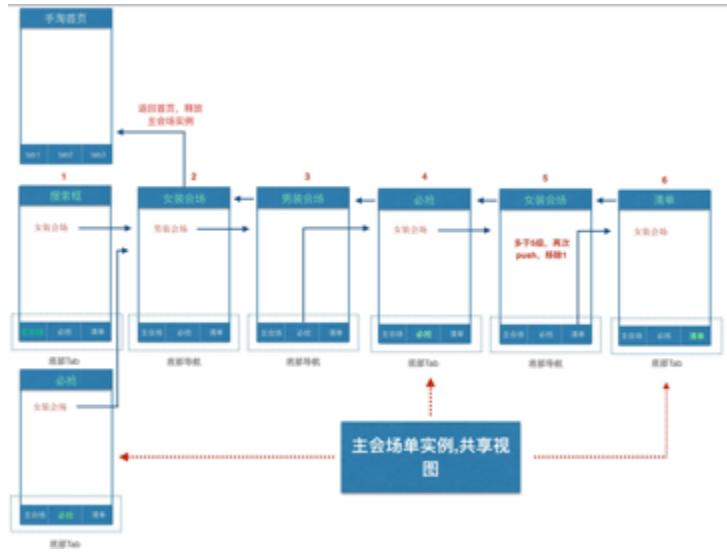


图 - 会场框架交互

Weex 会场框架很好支撑住了双11的复杂交互需求并提供了更好的内存管理。除了会场框架，更多的 Component 和 Module 支撑住了各色各样的双11需求，这里仅列出几个代表：

1. List 组件是几乎所有会场页面的标配，流畅的滚动帧率、高性能的内存复用机制和渲染机制是页面流畅体验的重要基础；
2. Animation 尽管是实验版需求，却支撑住了会场的垂直弹幕、坑位显隐等动画效果，动画效果细腻；
3. Weex 点播/直播组件 和 全景图组件支撑住了更为垂直个性化的业务需求。

## 2.2 组织结构

上节内容也可以看出，参与 Weex 双11 会场涉及多个团队和平台系统：



图 - Weex 双 11 中组织结构

1. 天猫业务：通过斑马（活动页面搭建和发布平台）发布会场页面；
2. 淘宝业务：通过斑马和 AWP （产品页面发布平台）发布会场页面，上层 DSL 使用 Rx（即将开源）；
3. 预加载：提前将会场 js-bundle 下载到客户端，客户端访问 Weex 会场时网络 IO 被拦截到本地文件 IO，从而极大加快了网络加载速度，预加载是这次秒开实战的抓手（注：最核心的工作）；
4. 手淘、手猫客户端，Weex 是客户端的一部分，客户端中其实是 Weex、Native、H5 并存的；
5. Weex SDK、业务模块 :Weex 容器和基础的 Components、Modules，业务模块包括直播/点播组件、全景图组件。

以上也仅涉及到客户端和发布端，背后还有无数的业务后台系统，就不一一列出了。

将 Weex 架构自上而下地展开：

1. Business , Weex 业务层 , Weex 双 11 主战场是手淘和手猫 , 此外还有大量客户端已经启用或接入了 Weex
2. Middleware , Weex 中间件层 , 包括为 Weex 页面提供发布 ( 斑马、 AWP ) 、预加载 ( AWP ) 、客户端接入支持 ( AliWeex ) 、组件库 ( SUI ) 、游戏引擎、图表库等模块 ; 其中斑马、 AWP 、预加载都直接参与了双 11
3. Tool , 工具层
  - i. DevTools , 界面和交互复用了 Webkit Devtools , 支持 elements 、 network 、断点、 console 等
  - ii. Playground , 方便开发者调试 Weex 页面 , 同时也是 Weex example 的聚集地
  - iii. Cli , Weex 命令行工具集
  - iv. 目前仍在建设更多的工具 , 如 weex-pack 支持一键打包成 App

4. DSL
  - i. JS Framework , Weex 最初的 DSL 是基于 Vuejs 1.0 语法子集 ; 目前在社区中在推进基于 Vuejs 2.0 的版本
  - ii. Rx , 基于 reactjs 语法的 Weex DSL( 将于 12 月正式开源 )
5. Engine , 渲染引擎 , 从架构设计上 ,Android/iOS/H5 RenderEngine 是相互独立和平等地位的渲染端 , 这是保持三端一致的基础 , 当然在协议实现层面需要更多的设计、质量保证



图 - Weex 架构

以上就是 Weex 在双 11 中的架构和业务支撑的范围了。

## 3 稳定性保障

Weex 的首要挑战就是稳定性，或者说保障 Weex 会场最大限度不降级。

### 3.1 iOS JSCore 内存治理

8月初(同期双11启动)奥运大促时，手淘iPhone中反复进出会场20+(手猫15+)，会出现crash。奥运大促当天，手淘iOS 1.59% crash次数来自该问题(top 6)，手猫1.94%(top 8)。发现问题的当天成立了攻坚小组，从JS业务代码、JSFM(框架)、iOS渲染、iOS JSCore几个方向同时排查，一周内各方向排查到逐步收敛到：根本原因是Weex页面实例被全局持有(weex runtime只有一份)，进而导致页面退出时内存不被释放，反复进出直至内存爆掉。因而任何可能导致页面实例被全局持有的因素都会触发这个问题：

1. 业务代码中的问题，意外导致的，给出 Lint 工具扫描业务代码，引入了“全局污染治理”(见下一节)
2. JSFM 框架中的问题，如在 destroyInstance 时清理 commonModules 和所有 dependency target；iOS7 下的 Set Polyfill 内存飙升问题
3. iOS 中的问题，通过下发配置控制 VC Push 层级控制；内存警告时的非当前实例销毁策略，加入开关控制；iOS 9.x JSCore 原生 Promise 和 Polyfill 并存时的内存问题

除了建立攻坚团队推进解决该问题，也在造势期前就展开双11会场压测，反复验证该问题，自双11造势期会场开测之后，该问题未再出现。

## 3.2 全局污染治理

在治理 JSCore 内存的过程中，逐步意识到对全局变量管控的必要性。Weex 中多个页面共用 1 个 runtime，单个页面如果写法不规范不仅可能导致内存泄露，更有可能污染全局环境，进而导致所有 Weex 页面无法正常工作。全局污染治理的核心抓手：

1. 严格模式，即 `use strict`，使用严格模式可以将较多常见的 JS 陷阱转化为错误，如：无法再意外创建全局变量、将拼写错转成异常、限制了 eval 的能力 等
2. 冰冻对象，利用 ES5 的 `Object.freeze()`，将 Weex 核心对象和 JS 原生对象“冰冻”住。尝试修改被“冰冻”的对象会抛出错误，一旦“冰冻”无法“解冻”。

## 3.3 跨端依赖梳理

Weex 通过 Adapter 来适配不同客户端的具体实现，诸多通用库，如：网络库、图片库、API 库、H5 容器（Web 组件）、埋点库、配置库 等在不同客户端上版本不一致，因此导致的线上问题将会成为双 11 会场的隐患。为此展开的依赖梳理和同步机制是双 11 稳定性的保障之一。这件事情可能将会长期出现在 Weex 问题清单之中，如何做到上层 Weex SDK

## 3.4 会场压测

### \* 压测场景

1. 5 个 200 坑位的普通会场页面，1 全景图会场页面，1 UT Expose 压测页面，1 直播会场页面

2. 页面中提供链接，能够按顺序进行 push 跳转

### \* 压测方案

1. 主链路（首页->店铺->详情->购物车）做一遍操作，让内存缓存占满，记下内存值 M0
  2. 进入 Weex 页面，滑动到底部，滑动到顶部，记下 M1；点击跳转按钮，跳转到下一个页面
  3. 重复步骤 2，让所有的页面进行压栈；全景图->p1p2p3p4->直播 ->p1p2p3p4->UT
    - \* 压测结果：iOS 通过，Android 通过
1. 测试过程手淘手猫均未出现因为压栈导致的 Crash，稳定性可以保证；
  2. Android 低端机压栈过多会导致体验较差，之后也会引入类似 iOS VC 层级控制；

压测在造势期会场测试阶段展开，在超出真实会场压力的情况下（真实会场 150 坑位上限）尽可能提前嗅探出潜在的 iOS JSCore 内存问题、iOS/Android 异常闪退等细节问题。

## 3.5 稳定性数据

2016.11.11，Weex 在手淘中的 Crash 占比情况：

- \* iOS 1.46% ( TOP7 )
- \* Android Java Crash 0.85% ( TOP13 )、Native Crash 1.72% ( TOP8 )

考虑到会场的业务量级，Weex 的稳定性仍然是不错的。注：单独计算的 Weex Crash 率太小，参考价值不大。

## 4 秒开实战

Weex 秒开率 = ( 加载时间 + 首屏渲染时间 ) < 1s 的比率

其中 **加载时间** 指 Weex js-bundle 的加载时间( 从网络下载或本地加载 )；  
**首屏渲染时间** 指 Weex 页面开始渲染到第 1 个元素 bottom 超出首屏范围的时间。下文提到的“首屏网络时间”为加载时间与首屏渲染时间的和。

从双 11 结果看预加载大幅度提升加载时间，对秒开率的贡献尤其突出；但性能优化是个长期迭代的过程，回头来看优化的抓手是 预加载和首屏渲染优化。

## 4.1 预加载

预加载解决了 1 个问题：

用户访问页面 ( H5/Weex ) 之前，将页面静态资源 ( HTML/JS/CSS/IMG... ) 打包提前下载到客户端；用户访问页面时，将网络 IO 拦截并替换为本地文件 IO；从而实现加载性能的大幅度提升。

日期	App版本	requestType	样本数	networkTime	样本数占比 (本页)
2016-11-11 11	6.1.0	network	1000000	295.97	55.10%
2016-11-11 11	6.1.0	packageApp	1000000	18.48	44.90%

日期	App版本	连接类型	样本数	networkTime	样本数占比 (本页)
2016-11-10	6.1.7	cache	1000000	648.98	1.65%
2016-11-10	6.1.7	http	1000000	921.82	0.10%
2016-11-10	6.1.7	https	1000000	2306.61	2.43%
2016-11-10	6.1.7	null	1000000	0.0093	3.24%
2016-11-10	6.1.7	packageApp	1000000	54.41	60.93%
2016-11-10	6.1.7	spdy	1000000	513.24	0.00%
2016-11-10	6.1.7	spdy_0rtt_acs	1000000	868.03	0.00%
2016-11-10	6.1.7	spdy_0rtt_cdn	1000000	696.47	31.65%

启用预加载后加载时间的变化，粗算一下：手淘 iOS，走网络平均 296ms，走预加载 18ms，网络性能提升约 15 倍；手淘 Android，走网络平均是 696ms，走预加载是 54ms，网络性能提升约 12 倍，但绝对值更大，对 Android 会场秒开贡献更为突出。

2015 年预加载已经在双 11 H5 会场中有较多应用，2016 年预加载升级为一项基础服务，不仅为 WindVane 提供预加载能力，也成为 Weex 秒开的最强外援。

此次双 11 会场共启用 30 个预加载包，总容量超过 20MB，业务需求相对稳定且流量较大的几个页面（会场框架+主会场 等）是独立的包，保证了对整体秒开的贡献，其他分会场均分在剩余的包中。同时主要采用强制更新的策略，即新的资源包（服务端有新发布）未下载到本地就直接读取线上，可以保证业务的实时性。2016.11.11，双 11 会场中 Android 走预加载占比为 59.4%，iOS 为 62.5%，高于平均水平（但还可以更高）。

## 4.2 首屏渲染优化

首屏渲染优化的目标就是尽力缩短首屏的渲染时间，为此在一系列的优化过程中，可以粗分为：DOM 解析优化、UI 渲染优化、分段懒加载。

### 4.2.1 DOM 解析优化

\* Component append 属性定义了 `node` 和 `tree` 2 种渲染模式，`node` 就是逐个节点渲染，`tree` 就是整棵树一起渲染。直观的对比：node | tree。

\* `node` 模式，节点逐个从 js 提交到 native 的，native 侧有个 16ms 间隔的 layout 保证渲染的正确性，这是更接近于 WebKit 的一种解析渲染模式。优势是每一个被解析完的节点都可以立刻显示，同时保证不会长时间阻塞主线程，劣势是可能会造成多次冗余 layout，拉低流畅性。

\* `tree` 模式，整棵树（以当前节点为 root 的整棵树）从 js 提交到 native。优势是只需布局一次，渲染更高效；劣势是如果 tree 过大，就可能会阻塞主线程甚至阻塞渲染。

\* `node` 和 `tree` 可以精细化地控制页面展示的逻辑和颗粒度，典型的实践为首屏以内按 tree 解析，首屏以外按 node 解析。

#### 4.2.2 UI 渲染优化

\* List 组件在 native 分别对应 iOS UITableView 和 Android RecyclerView，这两种 View 构建了 App 的半壁江山，使用它们来封装 list 的好处：

- \* 只会渲染可见区域，减少首屏的渲染消耗
- \* 内存复用，所有滑动到不可见区域的 cell 都会被系统回收，用于渲染下一个 cell
- \* cell 之间天然互相隔离，可以默认以 cell 维度划分并用 tree 的模式解析，提高渲染效率
- \* 拥有原生的交互体验，在 cell 上点击、左滑、右滑、移动排序等交互方式后续可以更方便地支持
- \* 想要达到 60FPS 的体验，一次主线程渲染必须少于 16ms。Weex 中一次渲染需要经过 6 个主要步骤( Build Tree、Compute Style、CSS Layout、Create View、Update Frame、Set View Props )，所以必须在 16ms 内完成这 6 个步骤，现实是任何一步在主线程中都可能超过 16ms，这块。

#### 4.2.3 分段懒加载

\* 除了底层的优化，业务上也通过分段懒加载进一步降低整体渲染时间，对首屏渲染有间接帮助（减少调度）

\* 方案为：会场页面使用 List 进行布局，一个 cell 对应一个模块；页面启动默认加载 6 个模块（少数页面因为首屏模块过多因此特殊处理）；默认往下滑到底触发 loadmore 后再加载 5 个模块；若加载过程中遇到电梯则电梯以下模块全部加载

除了底层的保障，我们也坚持每天产出“性能优化建议”，推进业务性能优化，接下来会有更加方便的工具提供给业务方直接性能调优；如双 11 期间

devtool 中增加了层级检测和告警 ,可以帮助排查深层级导致的 android 低端机 stackoverflow。

## 4.3 秒开数据

由于主会场流量占据了总流量的大部分 ,对其秒开率单列统计。2016.11.11 数据为 :

### 1. 主会场

- 秒开率峰值( 00:00 ) :整体 96.9% ( 278.8ms ) ios 99.4% ( 146.4ms ) Android 93.4% ( 411.1ms )
- 秒开率均值 :整体 94.4% ( 367.6ms ) ios 99.0% ( 177.0ms ) Android 91.8% ( 473.3ms )
- 帧率( FPS ) 红米 Note1s 53、小米 5s 58.5 ;iPhone5c 53.1、iPhone6p 56.9、iPhone7 58 ,帧率数据来自线下采集 ,见[视频](#)( 左起为 H5、iOS Weex、Android Weex ) 。

### 2. 所有会场

- 秒开率峰值( 00:49 ) :整体 92.4% ( 490.7ms ) iOS 97.4% ( 291.6ms ) Android 87.5% ( 689.8ms )
- 秒开率均值 :整体 83.9% ( 651.9ms ) iOS 94.5% ( 368.0ms ) Android 78.6% ( 797.4ms )

## 5 新的起点

Weex 技术委员会在十月中成立了 ,核心解决 Weex 开发过程中的标准化和协同问题。并于 10.26 进行了第一次会议 ,审议的 4 个话题 ( Input focus/blur、750px 实现方案、weex analyze 后续发展、标准化流程草案 ) 经过充分讨论 ,均获得全员投票通过。

鬼道作为第一任组长，接下来半年带领各方推进 Weex 在集团和社区的深度建设，欢迎大家参与 Weex 的共建之中。对于参与 Weex 的各方而言，最直接的影响就是：需要作为 Weex 官方推荐，向集团或（和）外部社区贡献的 Module、Component、工具、平台 等成果 需要通过 Weex 技术委员会检视标准性。要求在 方案设计出来后实现之前 和（或） 实现出来后 这 2 个时间点向 Weex 技术委员会汇报标准性相关细节；这个要求是强制的，目的是保持 Weex 社区的标准化推进。如果你只是为局部业务开发定制化的 Weex 扩展，不涉及标准性，并不会被要求到 Weex 技术委员会汇报。

Weex 任重道远。首先，Weex 不只是 Weex 容器，Weex 业务背后是发布、预加载、AB、线上监控、质量效能度量、数据埋点、业务开发技能转变/升级 等一系列行为的交织，如何减少业务从 H5/Native 转向 Weex 时的“阵痛”，是接下来的攻坚重点；其次，双 11 中遇到的典型案例或问题，会成为下一阶段的工作重点之一；第三，仍有大量业务需求需要开发，为此我们已经启动了 Weex BigBang 项目，按照 WindVane API 的调用频度和业务反馈情况，分批实现 30+ Weex Module/Component，包括常用的 schema 唤起支持、网络类型判断、geolocation、audio、cookie、大图预览、通讯录等；第四，减少新客户端接入 Weex 的成本，目前在尝试的 AliWeex 项目会扩大应用范围并成为客户端接入的标配；最后，跨客户端的底层依赖不同步问题会一直存在下去，需要更好的解法

一一列举了，之后会有 Weex Roadmap 的讨论并且会及时公布出来，欢迎关注。

从“Native 和 Web 融合”开始，先后经历的 Hybrid、React Native、WVC 再到 Weex，这段经历也算挺戏剧性的；未来会是 Weex 吗？答案并不重要，唯有沉醉其中。

## 8.5 双 11 晚会背后的技术

作者：邵雍



回顾 2015 年在鸟巢举行的第一届双 11 晚会，我们可以称之为“全民互动”的晚会。因为不止是现场的几千位观众，全国所有在电视机面前的观众朋友，都可以拿起手机，打开天猫客户端或淘宝客户端，参与到晚会现场的各个明星互动游戏中来，进行红黑押宝，获胜的人，还能抢到一元商品。

而刚刚过去的，在深圳大运中心的 2016 第二届双 11 晚会，更是有了历史性的突破，是一场“双向互动”的晚会。电视机前的观众，可以不只是单向的接

收舞台上的剧情变化，也可以用手机 APP 参与互动，来改变舞台上的剧情发展，支持哪位明星，就一起努力让 TA 赢。对于无法观看电视浙江卫视的用户，还可以打开天猫客户端，观看舔屏版的网络直播并同步参与互动。今年的直播渠道也有很多，手机天猫、手机淘宝、优酷、来疯、今日头条、Youtube，这样全世界的会员和粉丝，都能及时参与到晚会的互动中来，参与一场全球的狂欢盛宴。

这种“双向互动”的玩法，无论在国内还是国外，都是首创。超级碗和《美国偶像》的总制片人、88 届奥斯卡颁奖典礼总导演、天猫双 11 狂欢夜总导演 David Hill 称之为“好莱坞+硅谷”融合的里程碑式的创新。而做为技术的我们，面对这些技术挑战，也是心潮澎湃。



## 1 双向互动怎么玩？

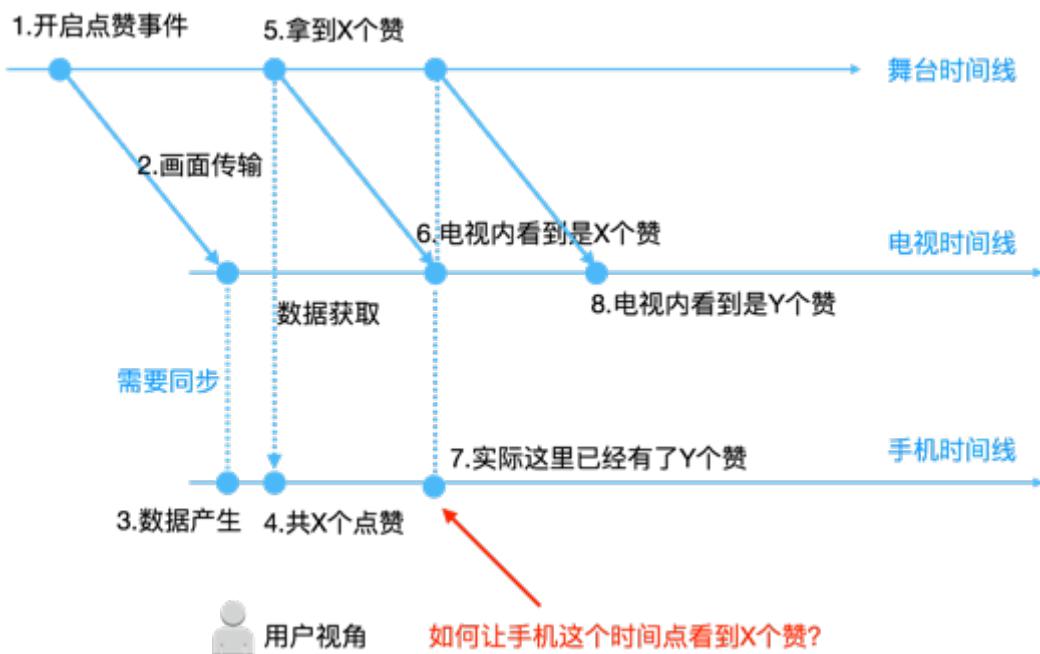
首先说“红黑大 PK”，这个互动去年就有，在主持人说“开启押宝通道”后，用户可以选择的红队或黑队，然后明星开始游戏，当其中一个队伍获胜时，押中队伍的用户有机会，就可以得到一个宝箱，开宝箱会有机会获得指定商品的 1 块钱购买权。今年这个游戏支持了双向互动，在选择完队伍之后，用户还可以

继续努力点赞，为支持的队伍“点赞赢时间”、“点赞得道具”等等，增加胜利的概率。粉丝们终于可以为自己的爱豆，贡献自己的力量了。

其他双向互动玩法还有“AB剧”、“跨屏抢星衣”、“满天星”等等。因为是技术文章，这里不再一一细说，其关键点，都是在“舞台、电视机、手机”三者之间，跨越距离和时间，让用户能有一种“身临其境”的感觉，看电视的同时，也能无缝的参与到我们的互动中来。

## 2 双向互动的难点在哪里？

前面介绍双向互动时说到了“跨越时间”，所以先看看这个时间上的难点在哪里。



如图，可以看到，“开启点赞事件”，首先由舞台发生，经卫星传递画面，60秒后在电视画面出现，此时，用户拿起手机开始点赞。然而，这个点赞数据，开始从0到1的时候，舞台内容其实已经过去了60秒了，这时候舞台只能拿到刚刚开始变化的点赞数据。虽然数据比实际要小，但我们肯定不会去造假夸大，

只能用。所以舞台上的主持人及明星嘉宾，必然是在一个事件发生的 60 秒之后，才能看到数据反馈的（从现场 L 屏看到）。那么关键性的问题来了：现场 L 屏的数据，到了电视里，已经是 60 秒之后了，手机上，怎么同步和电视 L 屏出现一样的数据？

这个问题其实有点烧脑了，我们先从一个简单的问题来说：舞台事件发生的 60 秒后，电视画面出现了相应的“口播”，手机端怎么知道，要“开启点赞事件”了，可以开始从 0 到 1 的计数了？

这个问题，其实我们去年就解决了，我们设计了一个时差同步的专利，大致流程是这样的：

互联网导播点击“开启点赞”按钮

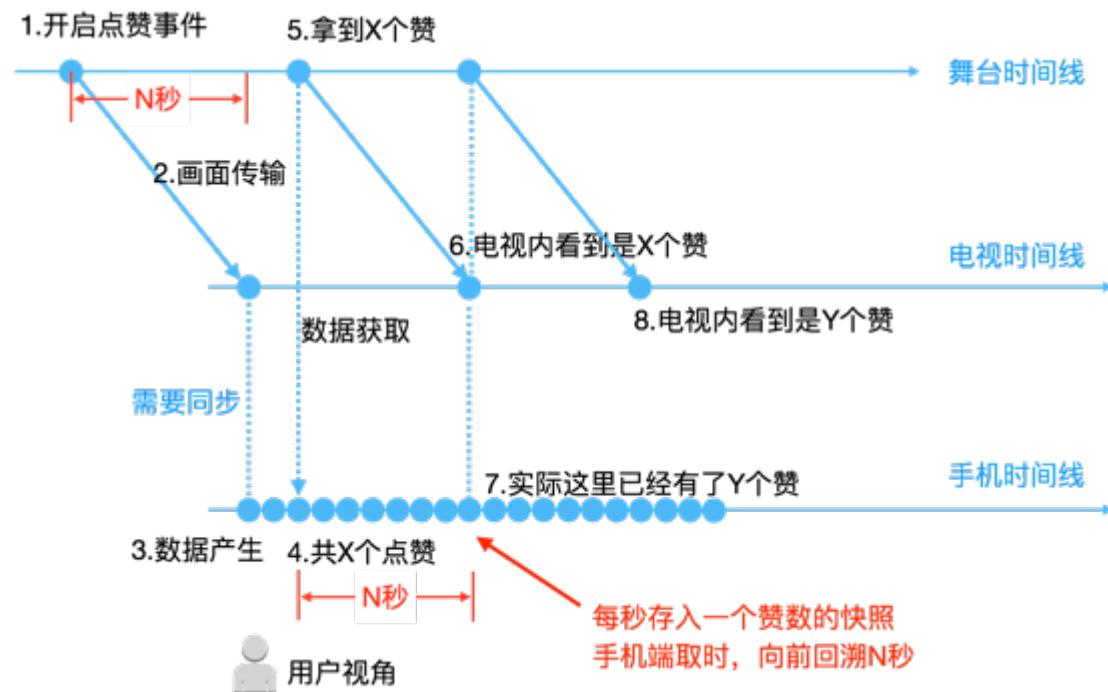
- > 互动后台校准时间，预计电视在 X 分 Y 秒时，才会出现对应的画面
- > 推拉结合预告手机端，在 60 秒内手机端能知道“开启点赞”事件将要发生在 X 分 Y 秒
- > 到了 X 分 Y 秒，手机端，执行相应的事件处理

这样就解决了手机端和电视同步出现一样的内容，然后再回来说，怎么和电视 L 屏出现一样数据。

互联网导播点击“开启点赞”按钮

- > 互动后台校准时间，预计电视在 X 分 Y 秒时，才会出现对应的画面
- > 手机端推拉结合，在 60 秒内，获知“开启点赞”事件发生在 X 分 Y 秒
- > 到了 X 分 Y 秒，手机端，执行相应的事件处理
- > 现场 L 屏开始获取当前时刻的数据（同时将数据持久化起来），数据合成到电视信号，在 X 分 Y 秒，出现在了电视 L 屏
- > X 分 Y 秒，手机端取用的不是当前时刻的数据，而是约 60 秒之前，持久化起来的数据

具体些，将每秒的数据存下来，每毫秒的数据都 hash 到秒，每秒的数据由定时钟写入，然后 L 屏后台获取的是当时的数据，而手机端用户请求的是，60 秒前，存入的数据。这里的 60 秒，只是一个估算的值，具体需要节目卫星通道建立之后，再进行对时校准得出。



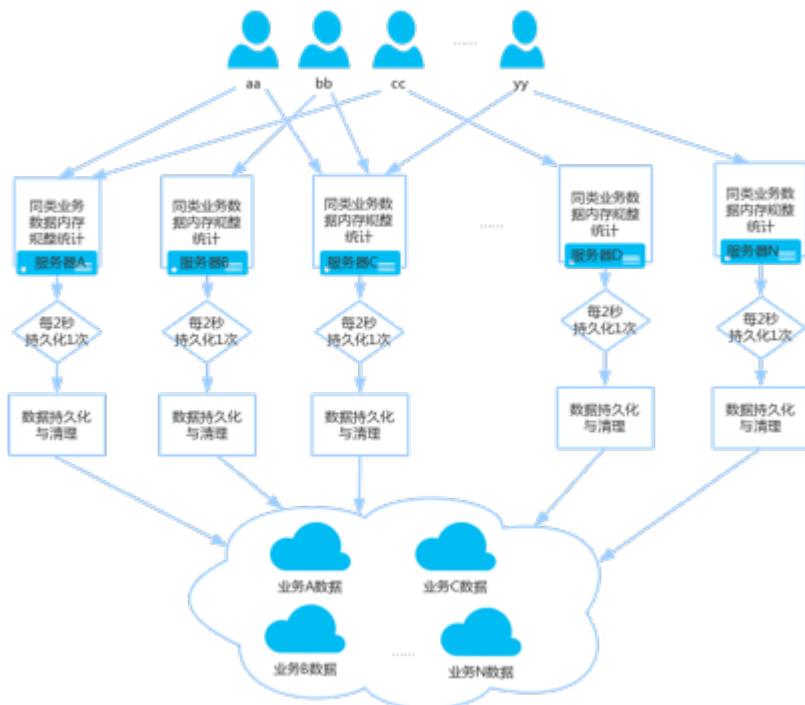
### 3 无时无刻实时点赞排名

点赞，常见于现在比较热的手机主播直播，而在大型电视现场直播节目中出现、使用，也是头一次。前面提到的“双向互动”，主要的互动参与方式：就是简单的“点赞”，一直狂点。很多人技术同学，可能会不以为意，点赞后台不就是个计数器吗？这有什么难的，两行代码就能搞定。

然而我们要面临的是千万级的点赞用户量，和时间非常集中的点赞请求，最终预估会有百亿千亿级别的点赞数。同时，我们要“实时”给出单节目 PV、UV、用户排行榜。

刚接到这个需求的时候也是觉得很棘手。关于单纯的点赞数（PV）功能，我们就做了很多技术选型，DB、缓存，都有难以突破的热点瓶颈，而我们分布式的后端，因为是分布式集群，用纯内存也并不那么放心。

最后，我们的解决方案是两者结合，用内存计数来顶住峰值浏览，但内存中只放入2秒的数据，每2秒会有机制去做持久化，这样就算真的遇到万一，丢了2秒的数据，也关系不大。我们构造了一个新的数据结构，称之为MIT（Memory Increase Tools），对外暴露的能力只有increase，然后内部封装了定时做持久化的逻辑，并且每次持久化都不阻塞其他increase线程。

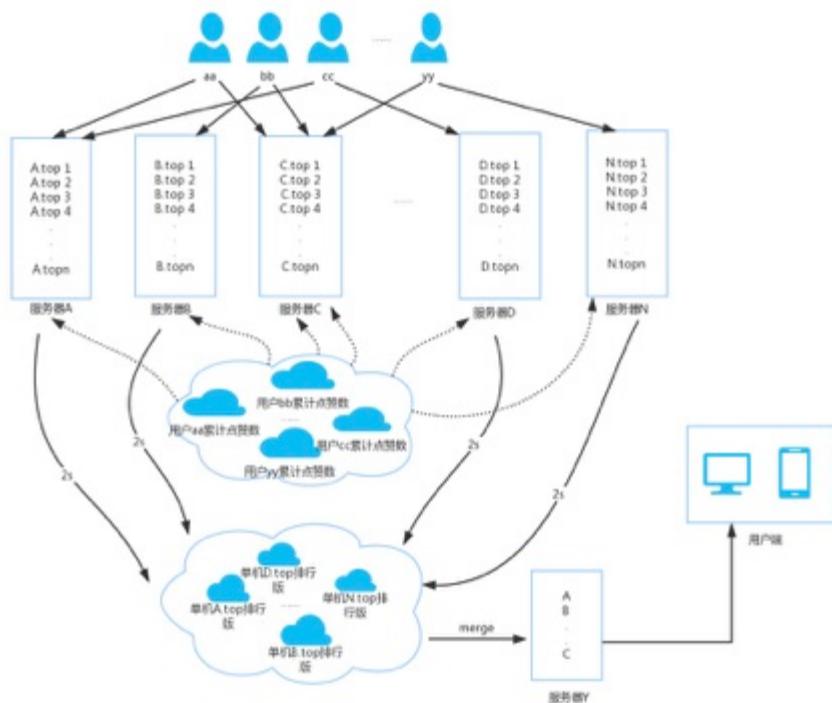


刚刚的MIT设计，也只是解决了PV统计的问题，UV还需要再想办法。传统的UV统计，大多需要在每个用户第一次点赞时，写入保存一个标识，然后每次点赞的时候，判断是不是存在这个标识，不存在时UV加1。

为了避免对于性能极大浪费的情况，特别是对于晚会这种苛刻的场景，我们仔细思考了一下上下文逻辑、意图，通过判断点赞数从0变为了1，就能判断用户是否来过，可以省下一次缓存，使大部分的点赞请求，就只有一次缓存的写入，其他都是内存操作，保证了接近极致的性能。

最后来看实时排行。动态数据的排行榜，最直接能想到的高性能解决方案也就是 redis 的 zadd 了，但这样的话，一来 zadd 的 key 就成为了热点，并发冲突变多、QPS 能力必然受限于单机；二来上千万的数据进行内存 zadd，内存大小、RT 也会暴露问题；三来就是成本，成本问题先不多说，值得说的是，这回钱不一定能解决问题。

果然上帝关上一扇门，又开启了一扇窗。冥思苦想，我们发现，前面 MIT 的思路，恰恰好地，也可以用于排行计算。我们可以用内存，持有一个单机版的排序，然后每 2 秒，刷入到一个缓存，然后定时把所有机器的缓存，合并出一个最终的排行榜。如图：



这里有一个关键点主要注意：持有一个单机版的排序，这里需要有一个在高并发下线程安全、定时刷入时不阻塞其他线程、能自动排序、自动逐出末尾的数据结构。这个数据结构，我们基于跳表也实现了出来，但限于篇幅，后面单独分享。

## 4 尖刺请求也要保成功

前面说过，晚会主打的玩法是“红黑 PK”，在主持人一声令下“开启押宝通道”时，所有用户蜂拥而至，押宝接口将迎来巨大的洪水流量，而这些流

量，我们还不能过早过低限流，因为会直接影响用户体验、互动参与率、甚至客诉。所以我们需要做的是极致的优化！

但是又需要持久化，所以我们限制自己，只能有一次 tair 的写入。最终我们的写法是：

```
oniszTair.versionPrefixPut(pKey(userId), sKey(pkId), target, 2, expire);
```

这里，用的是主子 key 的方式，这样有几个好处：

1、一次主 pkey 查询可以查出所有的押注记录，查询时，也可以节约 tair 网络交互延迟；

2、version=2，确保只有第一次才能写入成功；（第一次写入成功后版本为 1，传入非 1 的数字都会 cas 校验失败，不能写入）

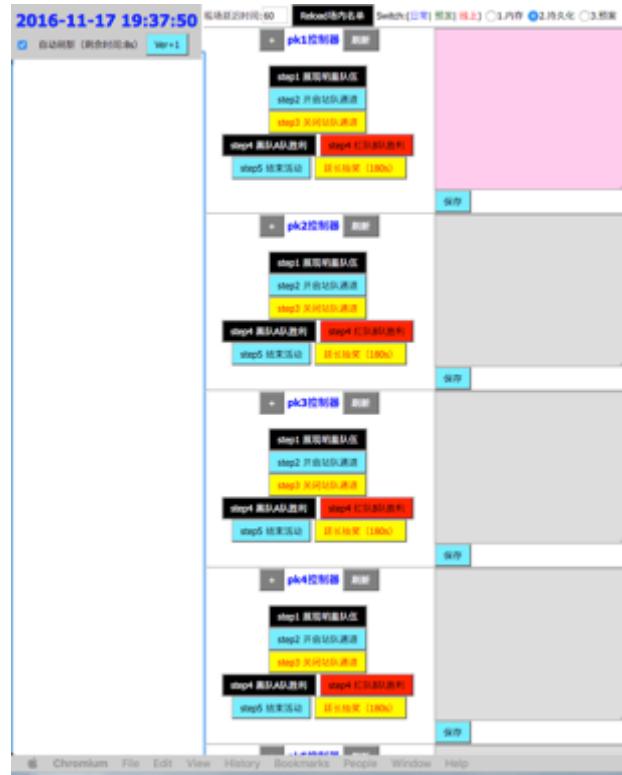
3、写入失败时，不会立即反查一次，而是让前端友好提示，这样在重试时，其实已经错开了峰值；

## 5 互联网导播控制台

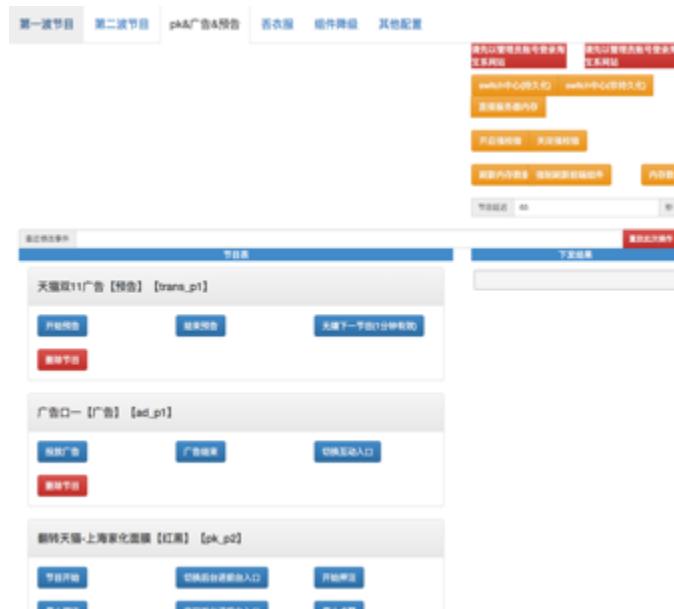
也就是我们的“核按钮”所在。说到这里，有必要说一下“互联网导播”这个职业，这个职业也是双 11 天猫晚会首创，去年才有。类似于电视导播，在导播车盯着十几个摄像机机位画面，将最好的画面切换给观众。互联网导播则是将最合适的互动内容（可以是匹配电视舞台内容或业务场景需要给的内容），切换给成千上万、甚至上亿的手机设备。这个工作会更个性化、更复杂，也更有挑战。



区别于电视导播的控制台，互联网导播使用的是自定义的 H5 页面控制台。2015 年的第一版是这样的，巨丑版：



2016年，我们用上了Bootstrap，漂亮多了。。



然而无论样式是怎样的，这个控制台都需要有如下特点：

- 1、一键式。任何场景，同一时间的指令，导播只需要点一个按钮。（人需要有反应时间，所以一般只来得及点一次。）
- 2、信息要全。要关注的信息，比如输入&输出&反馈，要一目了然，不需要再开新窗口。
- 3、预案灵活。百密也总有一疏，用于修复&纠正的预案，总是要有的。

剩下的，就是等待晚会开始，Rock&Roll！

## 6 跨终端 Web 动画制作工具

这是本次晚会前端的一个技术亮点：产出了通用动画导出工具。由于这是一场时尚的晚会，很多明星大腕，对视觉的要求也会非常高，尤其是对展现给用户的动画特效，更是苛刻的要求。素材制作上，设计师给出的视觉呈现，就给前端同学带来不小困难，比如：酷炫的动画过于复杂，如果按照视觉稿一帧一帧还原的话，需要耗费极大的人工成本，而且一旦动画出现需求调整，对前端开发人员来说，简直就是灾难，时间精力完全耗不起。

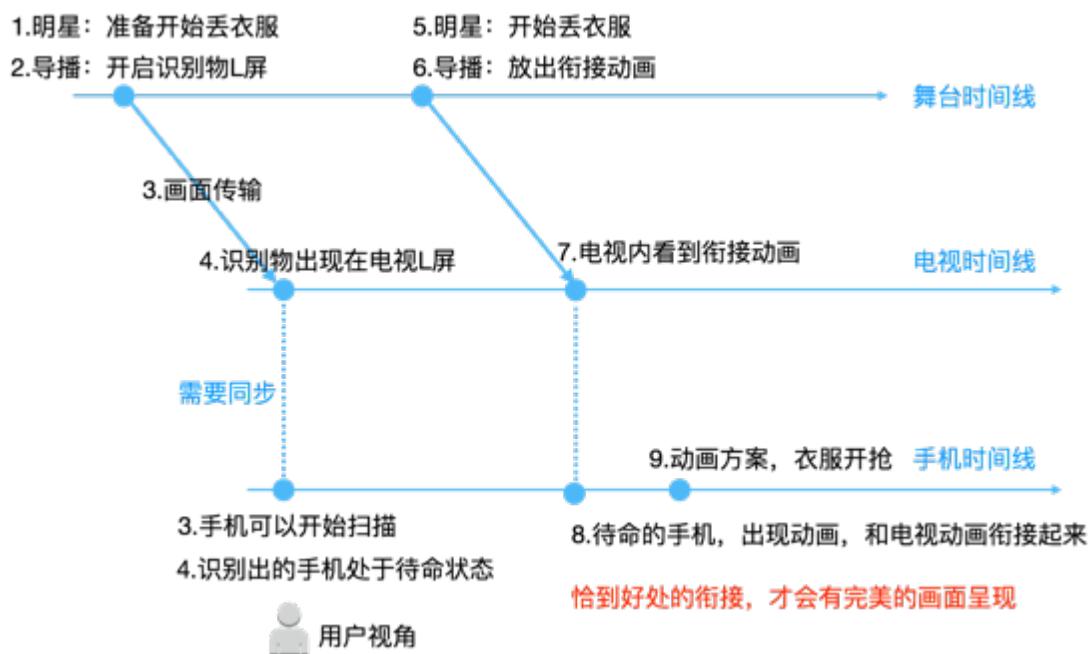
这种情况下，机智的晚会开发同学产出了通用动画导出工具，设计师只需要使用 After Effect(AE) 制作动画，前端就可以通过写 ExtendScript 脚本导出动画数据，优化、解析动画数据后，使用 canvas 来播放动画。

## 7 开启 AR 跨屏抢星衣

“跨屏抢星衣”是今年准备的一个特色环节，导演组安排了刘浩然和林志玲两位大明星，在某个节目中各丢出一件衣服，电视机前的观众，拿起手机可以参与抢“原味星衣”。明星在丢衣服之前，主持人会口播提示用户“使用手淘或者猫客的摄像头对准屏幕”，用户手机对准后，客户端通过 AR 识别技术进行识别和定位，识别成功后，在明星丢出衣服的瞬间，用户在手机上会看到衣服从电视中浮出，砸碎屏幕，到了自己的手机上，效果如图：



做到这点，需要各个环节恰到好处的衔接。逻辑如图：



这个衔接，一方面是明星、电视导播、互联网导播，对于内容 Q 点的约定，另外一方面，是基于前面说过的技术（推拉结合预告手机端），才能让互联网导播的按钮指令，及时的下达到手机。另外，值得一提的是，猫客在 AR 这块，用了

一个使用率较低、视觉融合较难，但是效果却特别好的开源算法（Traditional Template Square Marker），建议可以了解一下，多一种选择总是好的。参看：  
[https://artoolkit.org/documentation/doku.php?id=3\\_Marker\\_Training:marker\\_training](https://artoolkit.org/documentation/doku.php?id=3_Marker_Training:marker_training)

## 感想&展望

双 11 晚会，在技术眼中，就像是码农们进了娱乐圈，连代码都需要写的高大上。要应对各种集中的流量（口播、Q 点、抢）、还要把已经很酷炫的视觉稿还原的更加酷炫。在观众看着电视明星流口水的同时，还能参与互动，给心仪的明星支持，然后拿到礼品。这需要有着如丝般柔顺的体验，用户才会愿意玩。这些特性，在晚会史上都是前无古人的。即便是双 11 天猫晚会本身，在 2016 年也是超越了 2015 年太多的。可以预见的是，明年肯定有更多想不到的玩法，我们不断的创新，只为更 high 更好玩。各种新的技术，新的玩法，新的可能性，只有想不到，没有做不到。

回想起晚会平稳落幕的瞬间，技术人员内心的喜悦，是无法比拟的。本届互联网导播，控制核按钮的“那之手”说：我感觉更加耐操了，以后各种互动都不怕了！





扫码关注“阿里技术”微信公众号  
打开前沿技术的宝藏！