

[架构师]

双11特刊

电商生态的技术创新

2016

11.11



Geekbang® | InfoQ
极客邦科技

Broadview®
www.broadview.com.cn

蘑菇街11.11：

移动流量猛增，如何设计高并发多终端的无线网关



唯品会11.11：

频繁黑匣架构背景下，看唯品会的革命性重构

1号店11.11：

机器排序学习在电商搜索中的实战

魅族11.11：

架构为酒业务当歌，解读电商运维之困

苏宁11.11：

苏宁易购移动端的架构优化实践

架构师双十一特刊：电商生态的技术创新

联系我们

本期主编 徐 川

提供反馈 feedback@cn.infoq.com

流程编辑 丁晓昀

商务合作 sales@cn.infoq.com

发行人 霍泰稳

内容合作 editors@cn.infoq.com



极客邦科技官网

Geekbang.

极客邦科技

Mission

使命

整合全球优质学习资源
帮助技术人和企业成长

Vision

愿景

全球领先的技术人
学习和交流平台



会员制、城市分会、学习活动、全球技术领导力峰会

企
业
成
长

InfoQ

专注中高端技术
人员的技术媒体

EGO NETWORKS

高端技术人员
学习型社交网络

技术专栏、迷你书、顶级技术大会、垂直技术峰会

会员学习计划、小班课直播课、翻转课堂、WorkShop、
训练营、企业内训

技术人成长

StuQ

专注0-5年IT从业
者的职业教育平台

极客邦科技是一家 IT 技术学习服务综合提供商，旗下运营 InfoQ 技术媒体、EGO 社交网络、StuQ 职业教育三大业务品牌，致力于通过整合全球优质学习资源，帮助技术人和企业成长。

了解更多，请访问官网：<http://www.geekbang.org>

卷首语 | 本期主编 徐川 Amos

电商大促不是中国独有，但中国将它发扬光大，创造一个又一个的销售记录。同时，大促所带来的技术挑战更大，但世界上已经没有先行者供我们学习和模仿，只能自己探索。

今年我参加京东双十一的技术备战沟通会，印象比较深刻的是刘海峰老师说的一句话，他说现在大促已经常规化了，面向大促的备战也常规化了。

是的，仅仅在几年前，一场秒杀让服务器宕机是常事，但现在，一秒几十万个订单，也不再让我们惊讶。在这背后，是技术的进步，业务推动着电商技术飞速进化。

在这个进化过程中，分享和交流不是必不可少，但可以缩短独自摸索的时间。

InfoQ 策划的双十一系列内容，希望能促进行业之间交流，提高行业的整体技术水平。从 2014 年到现在，我们策划了 3 期的双十一系列文章，多场线下会议电商大促专场演讲，国内的电商技术，进入了一个学习实践和分享反哺的良性循环。

今年，我们除了关注大促技术架构演进、移动技术、网关、重构，还关注了机器学习、人工智能等等在电商搜索、物流、供应链的应用。

今年双十一之后，马云说，阿里双十一要继续办下去，要办满 100 届，我们也希望国内电商生态里的同仁，能更多的出来分享，从而将大促和电商带往一个更高的境界。



蘑菇街11.11： 移动流量猛增，如何设计高并发多终端的无线网关

作者 薛晨

自从 2011 年蘑菇街上线，蘑菇街一直沿用的是以 PHP 为核心的业务系统架构。但是，随着业务的增长、业务逻辑的复杂化，对技术架构有了更高要求。另外，随着移动互联网的普及，大量用户流量从 PC 端到无线端快速转移，故而移动端架构在保证稳定性的前提下，支持高效开发和迭代显得尤为重要。

而且，电商的大促业务越来越常态

化，双十一作为一年一度的电商大促高峰，更是一年比一年火爆。蘑菇街 2016 双十一，推出买手（红人）购物清单、红人买手直播、实时榜单等新功能。这些新的变化对技术上高并发、高可用的考验自然越来越大。

MWP 无线网关的设计

美丽联合无线平台（Meili Wireless Platform，以下简称 MWP），

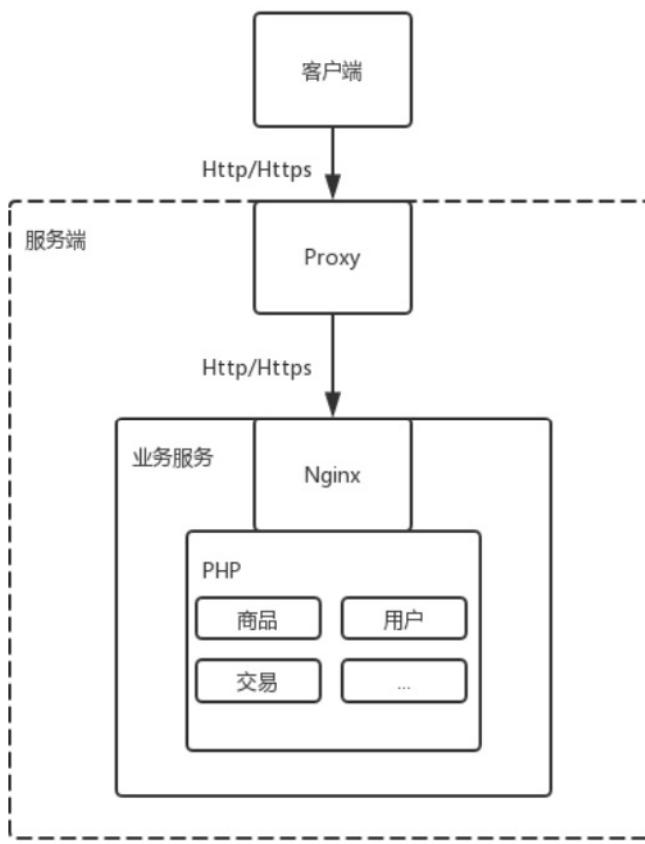


图1

是美丽联合集团为无线端开发的技术平台，它主要由无线网关以及围绕它开发的一系列技术组件和产品组成，是一套覆盖包括 Android、iOS、H5 等各类型无线终端技术组件和服务端通用业务开发框架在内的技术解决方案。

MWP面临的问题和挑战

基于 PHP 的架构是蘑菇街早期使用的，当时开发人员较少，业务逻辑相对简单，在业务迭代过程中更多地将就快速迭代试错，对于业务逻辑之外的系统

优化相对比较少人关注。

图 1 中，客户端发起 HTTP 请求到公网 Proxy，由 Proxy 将请求转发到业务服务器上，所有蘑菇街的业务都集中在一个 PHP 服务中，请求路径非常简单，基于这种架构的开发和运维也非常简单。从代码角度讲，所有业务代码都在一个工程里，相互调用都是简单的内部调用，使用非常方便，从系统角度讲，基于这种简单的系统架构排查定位问题也会降低很多难度，可以快速反应。

上述的架构在早期比较长一段时间都在使用，后面尝试过在服务化下对业务应用做一些简单的隔离，但是没有对架构有根本性的改变。这样的架构在当时的确行之非常有效，但是随着业务和团队成员的增长，越来越多的问题暴露出来。

整体性能较差

从宏观的角度来看，对用户来说，性能上其中一个明显的表现就在于客户端请求的 RT 上。有研究显示，移动端用户在点开一个页面的时候，如果 RT 超过 5 秒，有 74% 的用户将会选择离开页面。在原架构中，客户端请求链路只

支持 HTTP 短连接的方式，短链接的建立和释放会消耗很多时间，特别是移动端用户，在复杂的无线网络环境下，带宽资源非常宝贵，频繁地建立 HTTP 连接，所消耗的资源和时间成本会非常高，基于原有的架构优化的成本会非常高。

存在严重的安全风险

图 1 中的 HTTP 请求链路是一个裸露的链路，架构层面没有机制对请求做任务安全校验，如果黑客修改了请求中的数据，业务也能正常走下去，而业务上如果需要接入这种安全校验，比如支付、交易这种高危业务，则需要自己去额外接入。

开发效率随着业务复杂度和人员的增长快速下降

上文提到老架构中，各个业务都在

一个工程中，开发人员少的时候开发起来会很方便，但是当业务膨胀之后，里面的大部分业务都从之前的一两个人维护转变为一个团队在维护，在局部代码里会同时有多人在并行开发，随之而来的是沟通成本变高，代码变得臃肿，线上故障也随之频发。随着代码臃肿复杂，也给新的业务迭代带来成本的提高，开发的效率急剧下降。

下面我们来看一下 MWP 通过怎样的设计来解决这些问题的。

整体设计

MWP 提供从客户端到服务端的一整套技术解决方案（见图 2），包括如下内容。

- 客户端 SDK，为各客户端接入MWP而

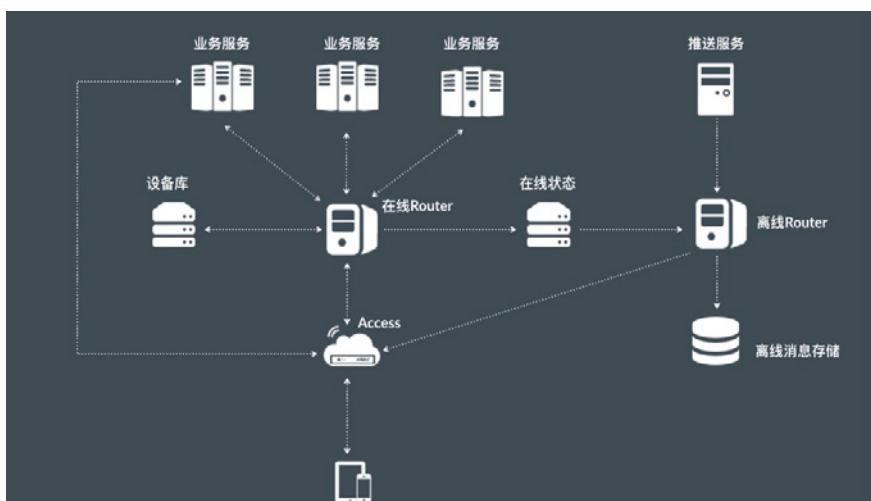


图 2

提供的客户端SDK。

- MWP-Router，主要为服务端应用提供统一的服务暴露方式，并为客户端的请求提供路由分发机制。
- Actionlet框架，为基于MWP的服务端业务应用提供统一的开发框架。
- MWP-DSL，是MWP提供的一套面向业务数据的无线端、前后端分离解决方案。

客户端SDK

MWP-SDK 作为客户端上业务访问后端服务的入口，与服务端的架构相对应，分层上将通用网络层和上层应用层分开解耦，最底层的通用网络层包含建联策略、HttpDNS、自有协议等网络优化需要的各方面，上层应用层包含对 API 请求逻辑、DSL 的封装等，这样的分离，对网络层的长期优化是非常必要的。我们将网络能力整体封装成了标准的 SDK，一方面将整体优化的收益推广给 App 群享受，另一方面也将 Native 通道的能力暴露给 H5。

在应用层，基于 Pipeline 灵活的编排和扩展能力，方便集成了客户端鉴权、离线缓存、防刷、时间纠偏、状态管理、性能上报等功能，实现了不同的网络协议之间实现灵活的切换和降级重

试，横向与其他客户端基础组件打通，比如，与配置中心配合实现配置的准实时下放等。为了解决页面请求过多、接口回调嵌套等问题，实现了 MWP-DSL 的调用方式，将原本客户端对数据的处理逻辑放到服务端的 DSL 层，解放服务端开发，合并客户端请求，减少页面上的网络请求损耗。

应用层的功能扩展和优化都基于网络层的稳定性和安全性上，所以网络层的优化显得尤为重要。由于移动网络的差异化和多样化，使得客户端的网络环境问题依然严峻，我们都或多或少遭遇到了各种域名缓存、内容劫持、用户跨网访问缓慢等问题，网络安全面临考验。MWP 的动态调度集成了 HttpDNS 组件来解决经常遇到的这些问题。动态调度通过策略的下发来控制客户端使用的协议（长链、短链、是否加密等）、端口等，通过策略优先级选择、不同网络环境下策略表的缓存和后台跑马等方式对建联策略进行优化。

在初期架构中，我们只对重要的接口使用 HTTPS，因为传统的 HTTPS 的整个握手流程是非常繁重的，尤其是在复杂的无线网络环境，往往造成建链过慢，甚至超时的情况；但是从安全的角度考虑，又必须对用户数据的传输建立在一

个安全加密的通道之上。为了解决两者的平衡，我们加入了安全网关接入层，接入层基于长链和自有协议进行数据的传输，并通过合并请求、证书预置和优化加密算法实现了一套基于 TLS1.3 的 0-RTT 加密机制。在建链的效率和数据安全上找到平衡，在不牺牲用户体验的基础上，达到了安全传输的目的。另外也正在尝试接入 HTTP2.0 协议，为客户端网络层带来更多的优化。

MWP-Router

MWP-Router（以下简称 Router）是 MWP 的路由层，它提供多种接入方式及 RPC 泛化调用方式，基于 Servlet 3.0 和 Pipeline 机制提供了高性能高可用的路由服务。

作为蘑菇街无线业务的入口，性能和稳定性是最重要的指标。Router 是基于 Servlet 3.0 和 Actor 模型的全异步架构，AsyncContext 和 Event-Loop 充分发挥了现代 cpu 的性能，在隔离各个请求资源的同时，用极小的内存换取了最大的吞吐量，灵活的 Pipeline 机制提供了强大的流程编排能力，结合 RPC 泛化调用提供了一整套标准的 API 服务。

Router 的其他特性如下：

- 通过构建符合协议标准的头部信息

可以方便集成鉴权、防刷、缓存、时间校准及配置准实时下放等特性；

- 管理后台通过精细化的配置来管理 App 和 API，包括路由、安全、流控、权限、别名等；
- 提供定制化的 DSL，客户端开发可以根据自己的业务场景任意组装和处理后端服务的元数据供端上展示使用，包括但不限于 API 聚合、API 依赖分层、数据分段返回等。

在最初的架构中，Router 是基于 HTTP 协议的，重要信息 API 使用 HTTPS。众所周知，在无线网络复杂而恶劣的环境下，数据安全和用户体验很难取得很好的平衡。为了解决以上问题，最大程度保证用户体验，我们增加了网关接入层来管理连接。接入层使用自定义协议和 App 建立长连接，基于我们自己实现的 TLS1.3 0-RTT 机制来保证建连的效率保障数据的安全，配合 session-ticket-reuse、证书预埋、App 加固等机制保证了协议本身的高效稳定及安全性。接入层缓存了部分基于连接的协议数据，对于优化网络 io 的效果也非常明显。另外，我们也接入了 SDPY 协议，并正在尝试接入 HTTP2.0 协议，期间对 Nginx 性能调优、内核参数调优、协议

参数调优等都积累了大量的优化经验，针对当下流行的微信小程序，后续还会接入 WebSocket 协议。

Actionlet框架

Actionlet框架的目的

MWP 为内部调用定义了 API 泛化调用方式，后端的业务应用若需要接入 MWP 需要遵循这种调用方式，所以 MWP 提供了 Actionlet 框架，为业务应用提供接入 MWP 的快速便捷方式，同时也为各业务应用带来一些额外的好处。

- 规范化业务对外输出接口，所有接入MWP的业务都需要按照一定的规则，有统一的输入和输出方式，这也是方便后续对API和应用进行统一管理的前提条件。
- Pipeline等模式隔离开环境和接入方式对业务逻辑的侵入，如果没有Actionlet框架，各业务开发需要关心请求上下文信息，比如 Servlet上下文等，这样可以提高 Actionlet 业务代码在多端接入方式（Android、iOS、H5 等）下的复用。
- 统一的Actionlet框架可以为一些通用的横向逻辑提供统一实现，各

业务开发只要高度关注自己的业务逻辑即可，而不用每个业务都需要接入依赖甚至自己实现这些逻辑，比如用户 Session 的处理就是一个很好的例子。

Actionlet框架的技术挑战

对于一个对外提供服务的业务应用，最关心的应该是服务的输入和输出，而各个不同业务 API 的输入输出又会有很大差异。比如，一个注册接口需要输入用户名、密码和其他用户信息，而返回的是是否注册成功的结果，而一个商品列表页则需要输入商品类型，返回的则是一个商品列表以及商品内部详细信息，甚至对应用户信息的复杂数据结构。在 Java 这种强类型语言中，如何抽象出一种统一 API 的规范，满足各种各样不通的业务，又能为外部提供统一的接口模型？

在 Actionlet 框架的目的里我们提到，业务应用本身应该是关注纯业务代码的实现，但是接入 Actionlet 的业务应用又是面向最终用户的。那么这里就会有一个矛盾，面向最终用户的接口必然会带上环境的上下文，比如，走 Web 请求就会有 Servlet 相关的上下文，甚至 HTTP 的上下文。怎样处理这些上下

文信息，让业务开发能完全关注业务代码的实现，而不用花很多心思在处理请求的上下文上？

在接收到请求时，系统需要处理很多逻辑才会走到业务代码中，比如参数的解析、用户 Session 的校验、API 路由的选择等，这些逻辑串联在一起作为请求处理的前置流程，框架以怎样的方式控制这些流程的执行，又如何支持后续在这些流程中添加或修改？

Actionlet框架的设计

图 3 中，由 MWPBaseService 接收请求，下发给 ActionletExecutor，ActionletExecutor 作为真正的执行器入口。如果要使用整套 Actionlet 的框架，所有的请求需要由 ActionletExecutor 为入口来执行，再经过一连串的 Valve 流程，Valve 可以简单理解是拦截器，实际是阀门配合 Pipeline 做到对流程的控制，最后调用执行具体的业务 Actionlet。

Pipeline和Valve

Valve 是 Pipeline 中

的概念，而这里详细提出来，是因为 Actionlet 的执行流程中很多功能是通过 Valve 来实现的。比如，请求的路由 RouterValve、请求的执行 InvokeValve，都是 Valve。

那我们是如何通过 Valve 来对流程进行定义和控制的呢？其实默认的

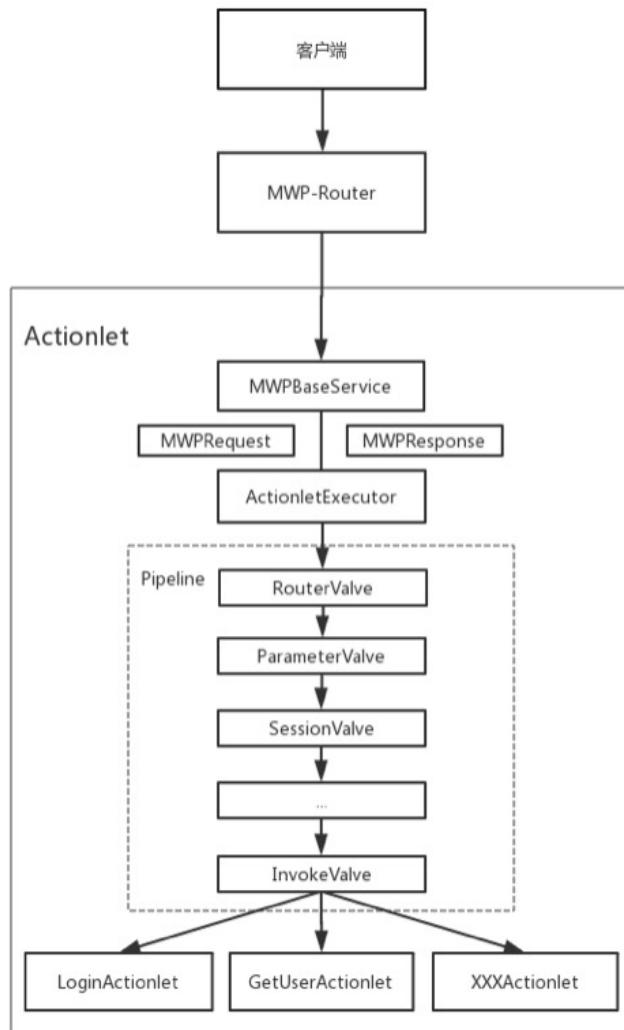


图 3

ActionletExecutor 就是基于 Pipeline 来实现的，它在初始化的时候就预先定义了一组 Valve，在请求进来时依序执行各个 Valve。如上图中，接收到请求后会依次执行 RouterValve、ParameterValve、SessionValve、InvokeValve，而如果后续扩展想改变流程或在流程中加入另外自定义的流程就非常方便了，只要在流程定义的地方修改就可以。

Valve 的排列顺序也是有要求的，因为请求是从第一个 Valve 执行到最后一个，再从最后一个执行到第一个，这是一个责任链模式。但是和拦截器不同，Valve 本身还可以做一定的流程控制，比如直接 breakPipeline，或直接 goto 到某个 Valve。

Actionlet的具体设计

首先，我们先来看一段 Actionlet 的接口定义。

```
public interface Actionlet<P, R> {
    ActionResult<R> execute(P parameter);
}
```

从上面的定义中，我们约束了 Actionlet 的入参 parameter 和返回的接口 ActionResult，强制约束了入参和

返回结果只有一个，业务方可以自由定义自己具体的 Domain 来作为输入和输出，这样做方便使用规约的方式来对外暴露接口，减少要对参数做映射的工作量。而负责在请求 Request 和返回结果的 Response 中，这两个 Domain 将会被序列化和反序列化成 Json 来进行传输。

那么有人可能会有疑问，大部分业务在获得自己业务输入之外，还会需要一些额外的请求信息，比如客户端来源，甚至 HTTP 头等数据，在这么严格的封装之下，如何拿到原始的 ActionRequest 和 ActionResponse 呢？可以通过 Actionlet 的上下文 ActionletContext 来获取，因为目前 Actionlet 都是同步的请求，所以请求的上下文放在 ThreadLocal 中。

上下文的隔离

图 3 中 ActionletExecutor 配合 ActionRequest 和 ActionResponse，就是为了将环境的上下文抽象出来，从而使 Actionlet 能更专注在纯业务代码上。

其中，ActionRequest 的作用就是将环境上下文中的请求给抽象成通用的模型，比如 Servlet 中 ActionRequest 就可以解析 HttpServletRequest 中的参数，从而封装成可以被 Actionlet 直接使用的 Request。而 ActionResponse

就是将 Actionlet 返回的数据结果进行对应环境的输出，比如，Servlet 中 ActionResponse 会将结果进行渲染然后输出给 HttpServletReponse。ActionletExecutor 就会将整个流程串联起来。

因为 Actionlet 的业务逻辑可能会对接多个环境实现，那么就可以针对不同的环境来实现不同的 ActionletExecutor 和相应的 Valve，来达到对环境的隔离。

异步Actionlet的支持

上文提到的业务 Actionlet 都是同步场景下的 Actionlet，在大部分场景下同步 Actionlet 已经满足绝大多数业务的请求，而在很多高并发场

景下，异步 Actionlet 会是更好的选择。Actionlet 框架提供了后续提供异步 Actionlet 的扩展，只需重写现有 Actionlet 调用的方式即可，对代码侵入性也比较小。

MWP-DSL

MWP-DSL 在 MWP 中提供一套 DSL，针对无线端（Android、iOS、H5）中和展现层强相关的业务数据的组装、拼接和转换，集成原有服务端部分 Control 层代码和客户端 View 层的代码，本质上是一套面向业务数据的无线端前后端分离解决方案（见图 4）。

服务端、客户端开发对接场景

客户端没有太多的 Control 逻辑，

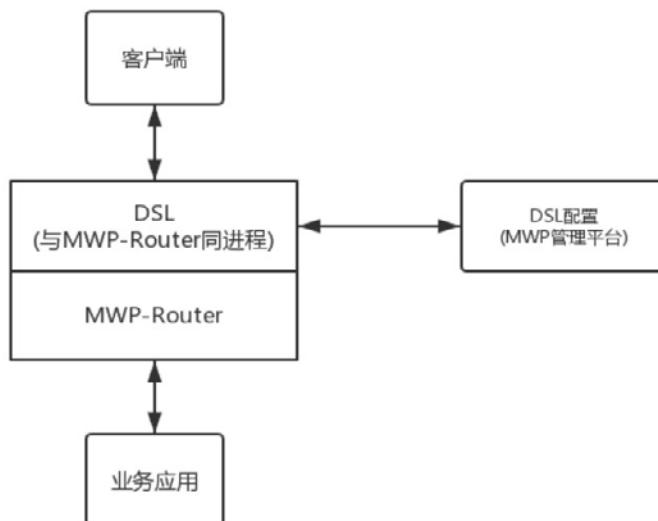


图 4

也没有太深的回调嵌套回调，服务端的 Control 层做了很多直接琐碎的直接关系展现层的数据的组装、拼接和转换。客户端一般只有一个大的 Callback，数据拿来后直接 Mapping，同时整个 Activity 都会依赖这个 Callback。

这种场景下的问题是，客户端没法做到分块加载渲染和 BigPipe，同时依赖一个大的 Callback，如果后台有任何接口超时会等待很久，此外服务端同学不能专注自己的 Module，任何小的需求改动（包括不需要后台提供数据 Schema 无变更的场景）都需要服务端同学参与，并联调。

服务端同学不为客户端的个性化展示需求做适配和拼接，客户端同学需要自己去调用多个不同服务提供方的多个接口，将 Control 层的逻辑已 Callback 嵌套的方式写在客户端。

在这种场景下，客户端同学代码 Callback 嵌套严重难以维护，三端 Control 层代码没法复用，任何业务上微小的改动都需要客户端同学发版。

MWP-DSL目的

- 工程上
 1. 客户端 MVC 强制分离，避免 callback 嵌套，提高客户端代码可维护性。

2. Android、iOS、H5 三端 Control 层逻辑复用。
 3. 客户端 Control 层逻辑变更不依赖发版，控制力更强。
 4. 专人做专事，面向业务数据的无线领域前后端分离方案，后端同学专注 Module 层，客户端同学专注在 View 层和 Control 层。甚至只要业务需求没有底层数据 Schema 的改变，完全不需求服务端同学介入，只要相应客户端同学自己组合下数据接口就好，减少前端联调成本。
 5. 有限的 DSL，提高上手速度，可维护性、安全性等。
- 技术上
 1. MWP-DSL 具备热更新的能力。
 2. 通过 BigPipe 支持分段返回，从而支持客户端诸如 Lazy Load 等，提升客户端用户体验。
 3. DSL 对于 MWP 异步化和并行改造，提升整体接口性能。
 4. 所谓微服务的可能落地方式。

业界现状

Facebook GraphQL 专注于提供面向业务数据的一种新的数据查询和检索方案，关注点在客户端数据查询的易用性，本质上希望客户端直接通过写类似 SQL

的方式（但是比 SQL 更直接，类似于面向数据 JSON）来对后台数据（把后台的一个接口类比于数据库中的一张表）做过滤和查询。

相比之下，本质上 MWP DSL 支持的业务场景更为复杂。

DSL 包含大量的业务逻辑，也就是 if else 和 for。

同时，我们对于元数据的新增和变更比较灵活，而不仅仅是数据的筛选。

所以，和 GraphQL 的异同可以理解为 MapReduce 和 Hive 的区别。

MWP-DSL的挑战

- 业务上
 1. 真实业务场景足够复杂，会出现任意N个MWP接口随机组合和callback情况。
 2. MWP-DSL提供的能力如何即受限又足够，同时易扩展，并且易用，也就是学习接入成本低。
- 技术上（主要是性能、稳定性与安全）
 1. 从轻量级MWP请求转发到多MWP组合并运算带来的系统压力。
 2. 为了做到非阻塞、全异步编码，带来的排查问题、线程模型与调度复杂度的增加。

MWP 特点（高稳定性、高 QPS、低 RT、性能问题）会被放大。

MWP-DSL的解决方案

- MWP-DSL的业务本质（业务模型，见图5）

 1. N个接口任意情况组合。
 2. M个flush到客户端（M>1，即为 BigPipe的情况）。

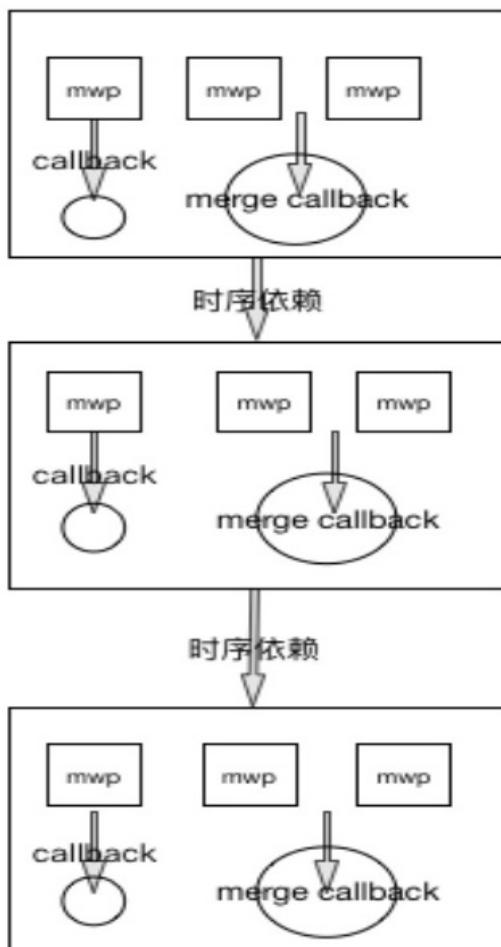


图 5

- 3. T个独立的callback（包含错误的细粒度处理，完全由业务方自己定制）。
- 4. 三种基本原子情况组合（独立、merge、时序依赖）。
- DSL客户端编码框架
 1. 多个flush处理各自的callback。
 2. flushkey和BigPipe解耦。
 3. 多个flushkey相互隔离，更细粒度的错误处理。
- 性能方面
 1. 全异步化与线程调度模型(rxjava、netty eventloop，多callback仍然交给触发线程，避免加锁的并发控制与线程拷贝)。
 2. 高性能Groovy集成（静态编译执行效率与原生java接近、jvm调优与GroovyClassLoader隔离避免GC问题，与perm区无用类爆炸、Groovy版本自身的bug）。
- 稳定性方面
 1. 线程隔离，避免极端情况影响MWP。
 2. DSL接口隔离与容错。
 3. DSL相关开关。
 4. 灰度发布与切流量。
- 安全方面
 1. DSL代码静态扫描，通过白名单和黑名单机制，明确业务方同学用DSL可以做什么和不可以做什么。
 2. DSL接口错误隔离。
 3. DSL接口级别资源控制，比如，限制DSL中的循环次数避免死循环对CPU的消耗，以及对于总体内存的监控与报警。
 4. DSL接口级别性能监控与自动化运维，比如，监控接口rt、自动对异常接口做降级操作等。

MWP上线运行情况

MWP 已经上线运行接近一年，集团蘑菇街业务相关的主要服务都已经从老的系统架构迁移到 MWP 上，目前整体运行非常稳定，对整体服务质量有了很大的提升。

• 性能方面

MWP 目前通过对网络链路做的优化，已经使用长连接的方式替换原来短连接的请求，这样建立连接的资源消耗只在打开或唤醒 App 的时候产生，而不会每次请求都重新建立连接。在实际应用中，客户端平均 RT 时间从原来的 841 毫秒优化到现在的 282 毫秒，优化非常明显。

• 安全方面

MWP 通过收敛请求链路，在网络链

路上做安全校验，防止数据包篡改等安全问题。针对 HTTP 短链方式，MWP 对请求头和数据包进行验签，验签失败的请求直接返回客户端失败信息，而针对长连接，我们自己实现的 TLS1.3 0-RTT 机制，有效保障数据的安全，并在此技术上做了更多优化。

- **开发效率方面**

通过 MWP 的路由分发和 Actionlet 框架，为业务提供快速业务迭代的可能。按照过去的方式，各业务代码杂糅在一起，各业务开发需要考虑请求的上下文信息，比如从移动端过来请求和从 PC 端过来请求的不同处理方式，参数防篡改，以及安全和反垃圾等。而如今接入 MWP 之后，各业务应用天然独立，业务开发只需关注业务逻辑，其他的像网络链路、上下文解析等 MWP 都已经封装处理掉，无需业务关心，有效提升多人协作下的开发效率。

基于MWP的周边生态的建设 支持横向功能的扩展

对于业务系统而言，安全、反垃圾、限流等这些横向的功能是每个业务都需要去考虑和实现的。过去的方式是，每个业务都需要引入一堆的依赖来实现每一个功能，甚至有些功能各个业务都自

己实现一套，工作量复杂又冗余，业务开发的注意力被分散在周边逻辑中而不是聚焦在业务逻辑。而 MWP 已经实现或者接入了这些功能，对于具体业务开发来说只要接入 MWP，默认地或者可以用简单的配置来接入这些功能，非常方便。后续如果有更多的横向功能，只要 MWP 来实现就可以，业务应用只要拿来就用即可。

外部系统接入

对于客户端 App 和服务端应用而言，一些周边系统的功能非常重要，比如一个强大的配置中心提供配置管理，方便地修改客户端配置，你想随时推送最新的启动图到客户端，又或者让客户端网络连接方式在 HTTP 和长连接之间切换。MWP 就为这些系统提供了一个强大的平台，支持了这些系统的接入，如目前支持通过通道准实时推送配置等。

API和应用管理

MWP 提供了配套的管理后台，对 API、DSL、服务端应用和客户端 App 进行管理。在此基础上，用户可以在后台查看 API 的 QPS、RT 这些实时基础数据，方便了解线上运行情况。除此之外，还支持在后台对接口进行简单的测试，以及对客户端权限、接口流控、超时控制

等参数进行配置，并实时生效。

未来展望

目前我们正在使用 Go 重写网关接入层，优化现有的长连接机制。一方面，MWP 客户端和服务端之间的链路主要支持上行请求，这样服务端的一些变更只有在客户端主动请求或拉取的时候才能下发到用户，如果能支持下行通道，不管是对于业务的拓展还是系统机制的优

化都会打来很大好处。另一方面，MWP-Router 是 MWP 系统中的重心节点，如果网关接入层足够强大，后续可以轻松地将 Router 的功能下沉到业务服务，实现去中心化。

MWP 是一个基础平台，随着集团业务的发展，还需要围绕这个平台建立更多更完善的功能和系统，以支撑集团业务更长远的业务发展。

主办方 Geekbang. 极客邦科技 X-NODE 创极无限 战略合作伙伴 拉勾 专注互联网职业机会

12月4日 - 12日 硅谷 和我们一起去硅谷！

ChinaTech Day

中国技术开放日·美国站

邀您同行！

30位技术大牛同行，探访硅谷10家创新科技企业

QR code



1号店11.11： 机器排序学习在电商搜索中的实战

作者 张志浩

背景

1号店的搜索 Ranking Model 一直在朝着精细化方向深化，我们希望在提升用户满意度的同时，也能提升网站的流量转化率。在实践机器排序学习之前，1号店网站的搜索 Ranking Model 已经经历了 4 个阶段：通用排序模型 (Universal Ranking Model)、基于区域的排序模型 (Region-based Ranking Model)、基于品类的排序模型 (Category-based Ranking Model) 和基于用户的排序模型 (User-based Ranking Model)。

在这个过程中，需要对搜索、浏览、

点击、购买，评论等几十个行为特征进行细粒度的分解和重组，这就需要人工去分析用户的搜索行为和流量效率之间的关系，并通过人工调整排序模型来优化效果。在细分了区域和品类之后，为了更有效地提升排序模型的优化效率，我们开始考虑使用机器排序学习算法。

机器排序学习的一般分为两个流程，其中 “*training data -> learning algorithm -> ranking model*” 是一个离线训练过程，包含数据清洗、特征抽取、模型训练和模型优化等环节。而 “*user query -> top-k retrieval -> ranking model -> results page*” 则

是在线应用过程，表示利用离线训练得到的模型进行预估。

机器排序学习有如下优点：

- 人工规则排序是通过构造排序函数，并通过不断的实验确定最佳的参数组合，以此形成排序规则对搜索结果进行排序。但是在数据量较大、排序特征较多的情况下，依靠人工很难充分发现数据中隐藏的信息，而机器排序则可以较好地解决这个问题；
- 机器学习可以基于持续的数据反馈进行自我学习和迭代，不断地挖掘业务价值，对目标问题进行持续优化。

设计原则

从一开始，我们就没有考虑将机器排序学习作为人工规则排序的替代者，

而是致力于将两者作为互为补充的排序模型，不同品类采用不同的排序策略，这也体现在了我们的搜索排序架构上。

图 1 表示目前 1 号店现在生产环境上机器排序学习的框架，包含两个部分：离线训练和在线应用。离线部分主要是根据历史数据以及训练目标，产生一个可用于在线预测的排序模型；而在线部分则是利用离线产生的排序模型，根据在线用户的 Context 完成实际的排序。

在整个机器排序框架中，整个在线排序模块分为 3 层。

第一层称为一排（也称为粗排），主要是根据用户 Query 从索引库中召回所有相关商品，然后依据文本相关性和类目相关性得分，完成第 1 轮商品筛选，形成上层排序模型的候选商品集。

第二层称为二排（也称为精排），

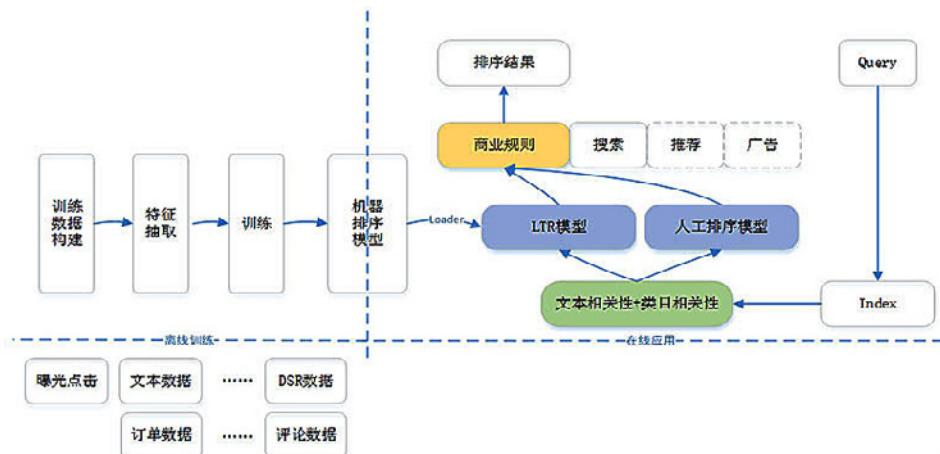


图 1 机器排序学习框架

主要是以一排的候选商品集为基础，应用 LTR 模型或者人工排序模型，完成基于排序特征的重新排序。

第三层称为三排，主要是根据各种商业需求对二排的排序结果进行调整，如类目打散、商品推广等。

在整个机器排序框架中，离线部分需要模型评测环境，而在线部分更需要数据收集模块和 A/B Test 模块。这两点并没有在图 1 的框架中列出，但是在后续的介绍中会给出相应的说明。

机器排序学习离线训练过程

我们选择开源的 RankLib 作为机器排序学习的工具包。在离线训练过程中，我们测试了 RankNet，LambdaMART 和 Random Forests，根据评测结果，我们选择了 LambdaMART（具体采用哪种模型，需要根据实际业务场景和训练结果而定）。该算法是一种有监督学习 (Supervised Learning) 的迭代决策回归树排序算法，目前已经被广泛应用到数据挖掘的诸多领域。

LambdaMART 模型可以分成 Lambda 和 MART 两部分，底层模型训练用的是 MART (Multiple Additive Regression Tree)，也叫 GBDT (Gradient Boosting Decision Tree)，它的核心是每一棵树

学习的是之前所有树结论和的残差，这个残差 + 当前的预测值就能得到真实值。而 Lambda 是 MART 求解过程使用的梯度，其物理含义是一个待排序的文档下一次迭代应该排序的方向和强度。具体 LambdaMART 的算法和工作原理可以参考。

下面我们以 LambdaMART 为基础算法来介绍机器排序学习的整个过程。

训练目标数据

离线训练目标数据的获取有 2 种方法，人工标注和点击日志。两者都是为了构建出 $\langle q, p, r-score \rangle$ 的数值 pair 对，作为机器学习的训练数据集。这里 q 表示用户的查询 query， p 表示通过 q 召回的商品 product， $r-score$ 表示商品 p 在查询 q 条件下的相关性得分。

其中人工标注一般步骤为，根据给定的 query 商品对，判断商品和 query 是否相关，以及相关强度。该强度值可以用数值序列来表示，常见的是用 5 档评分，如 1- 差，2- 一般，3- 好，4- 优秀，5- 完美。人工标注一方面是标注者的主观判断，会受标注者背景、爱好等因素的影响，另一方面，实际查询的 query 和相关商品数量比较多，所以全部靠人工标注工作量大，一般很少采用。

因此，在我们的实践探索中，寻找

获取方便且具有代表性的相关性的度量指标则成为重中之重。经过初期的探索，我们确定以 CTR 为基础，实现了低成本的 query–product 相关性标注（虽然不完美，但在实际工程中切实可行）。具体步骤如下：从用户真实的搜索和点击日志中，挖掘出同一个 query 下，商品的排序位置以及在这个位置上的点击数，如 query_1 有 3 个排好序的商品 a, b 和 c，但是 b 得到了更多的点击，那么 b 的相关性可能好于 a。点击数据隐式地反映了相同 query 下搜索结果相关性的好坏。

在搭建相关性数据的过程中，需要避免“展示位置偏见” position bias 现象，即在搜索结果中，排序靠前的结果被点击的概率会大于排序靠后的结果；在我们的训练模型中，如图 2 所示，第 6 位开始的商品点击数相比前 5 位有明显的下降，所以我们会直接去除搜索结果前 5 个商品的点击数。

同时在实际场景中，搜索日志也通常含有噪音，只有足够多的点击次数才能体现商品相关性的大小。因此为了提升训练效率和训练效果，我们也针对 Click 数少于某个阈值的情况（即少于某个阈值的点击，我们就直接忽略）进行了测试，分别为 2, 3, 4, 5, 6, 7, 8。

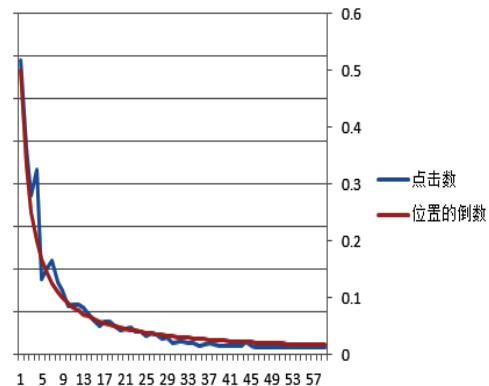


图 2 位置偏见的示意图

经过离线分析，在阈值为 4 的情况下，NDCG 分别有 20%-30% 的提升（这是 NDCG@60 on validation data 和当前线上的 NDCG 相比），效果提升比较明显（见表 1）。

特征抽取

LTR 使用的排序特征（也称为 Feature），和人工规则排序使用的特征基本相同。按照数据特性的差异，我们将其分为 2 大类，如表 1 所示，当然不同类型的数据价值和反映用户意图的强弱也不尽相同（见表 2）。

特征数据分几类或者怎么分类都不是重要问题，这里主要是说明要尽可能抽取多的特征数据，供后续训练使用。同时对于机器排序学习而言，这里的行为特征都需要跟踪到 Query 维度，所以在数据采集的时候，需要预先处理好，这样才能体现 Query 对应的排序影响。

表格 1 Click 数对训练目标的影响

	2	3	4	5	6	7	8
Average Overlap Score	0.2779563	0.2622574	0.2927366	0.2605555	0.3292106	0.2570833	0.2815816
RBO Score	0.3660776	0.3278322	0.3587360	0.3362893	0.3906409	0.3456386	0.3614653
LikeNDCG1	3918.920	4529.955	4345.691	4550.222	3970.641	3555.782	3418.661
LikeNDCG2	203.5713	250.1454	235.3356	244.135	209.4407	217.9441	207.5903
NDCG@60 on training data	0.4734	0.4479	0.502	0.4846	0.4983	0.621	0.643
NDCG@60 on validation data	0.4896	0.4727	0.536	0.4743	0.4928	0.5015	0.4864

表格 2 特征抽取部分样例

Feature类别		Feature项
查询相关性特征	查询相关性	text relevance, category relevance
商品特征	商品静态特征	title, sub-title, attribute (brand, category, size, color, flavor)等
	商品动态特征	price, promotion等
	订单特征	sales volume by day/week/month, GMV by day/week/month等
	行为特征	CTR (list->detail, detail->shopping)
	售后特征	number of review, positive rate of review, complain rate, return rate等
	商家服务特征	DSR, online duration, online instant response等

如果能将订单特征也区分到 Query 维度会更好，但是我们现在还没有做到这一点，以后还会继续实验。

除了商品静态特征以外，商品动态特征、订单特征、行为特征、售后特征、商家服务特征都需要细分到品类和区

域，幸运的是，在人工规则的排序模型阶段，就已经为我们准备好了 profile-based 的特征数据。为了能在人工规则排序和机器排序学习之间共享这些特征数据，我们将这些特征数据存储在基于 HBase 的 Item Feature Repository 数

据库中。目前这些特征数据还是以天为单位进行批处理，而非实时完成。

在得到相关度数据和特征数据后，就可以根据 LambdaMART 训练数据的格式（如下所示），构建完整的训练数据集。每一行代表一个训练数据，项与项之间用空格分隔，其中 <target> 是相关性得分，<qid> 是每个 query 的序号，<feature> 是特征项的编号，<value> 是对应特征项归一化后的数值，<info> 是注释项，不影响训练结果。

```
<target> qid:<qid>
<feature>:<value> <feature>:<value>
... <feature>:<value> # <info>
```

图 3 表示项目中使用的实际训练数据（这里选取了其中 10 个特征作为示例，# 后面可以增加 Query 和商品名称，方便分析时的查看）。

离线训练

LambdaMART 学习过程的主要步骤可以参考，数学推导不是本文的重点：

```
1 0 qid:983 1:0.582 2:0.353 3:0.553 4:0.582 5:0.451 6:1.762 7:0.891 8:0.998 9:0.778 10:0.543 #维达蓝色经典系列3层140g卷筒卫生纸
2 0 qid:983 1:0.583 2:0.422 3:0.553 4:0.583 5:0.452 6:1.758 7:0.902 8:1.012 9:0.765 10:0.545 #维达3层280片卷筒卫生纸*12卷
3 0 qid:983 1:0.582 2:0.413 3:0.552 4:0.583 5:0.439 6:1.689 7:0.901 8:0.986 9:0.782 10:0.476 #维达至有份量系列3层200g
4 2 qid:379 1:0.713 2:0.310 3:0.716 4:0.672 5:1.103 6:0.997 7:0.913 8:0.770 9:0.614 10:0.123 #vinda维达 无芯卫生卷纸 无香 85g*12粒
5 0 qid:379 1:1.021 2:0.322 3:0.709 4:0.657 5:1.112 6:0.879 7:0.924 8:0.775 9:0.324 10:0.132 #心相印mind act upon mind心相印
6 2 qid:379 1:1.032 2:0.345 3:0.804 4:0.649 5:1.004 6:0.645 7:0.935 8:0.679 9:0.527 10:0.127 #清风 卷纸 马蹄莲系列3层100g
7 0 qid:677 1:0.981 2:0.262 3:0.777 4:0.465 5:0.000 6:0.000 7:0.758 8:0.567 9:0.435 10:0.127 #3m/思高 超洁净百洁布3片装
8 1 qid:677 1:0.980 2:0.259 3:0.769 4:0.465 5:0.000 6:0.000 7:0.762 8:0.587 9:0.435 10:0.157 #3m/思高 防刮擦百洁布3片装
9 2 qid:677 1:0.981 2:0.253 3:0.782 4:0.465 5:0.000 6:0.000 7:0.773 8:0.587 9:0.435 10:0.130 #3m/思高 防刮擦海绵百洁布2片装
10 0 qid:672 1:1.311 2:0.724 3:0.000 4:0.443 5:1.000 6:0.217 7:0.773 8:0.593 9:0.378 10:0.654 #妮飘 手帕纸 60包3层10片/包 纸面印花
11 0 qid:672 1:1.311 2:0.723 3:0.000 4:0.443 5:1.000 6:0.218 7:0.765 8:0.582 9:0.412 10:0.655 #彩竹 波斯猫 名典迷你手帕纸(加香)
12 0 qid:672 1:1.312 2:0.645 3:0.000 4:0.553 5:1.000 6:0.223 7:0.767 8:0.576 9:0.412 10:0.652 #清风 清香型纸手帕10包 原生木浆纸巾
13 1 qid:655 1:1.608 2:0.100 3:0.627 4:1.092 5:0.667 6:0.225 7:0.671 8:1.001 9:0.523 10:0.716 #3m/思高 易洁8层耐用抹布
14 0 qid:655 1:1.607 2:0.112 3:0.628 4:1.094 5:0.652 6:0.226 7:0.669 8:0.980 9:0.409 10:0.011 #3m/思高 耐用型天然橡胶手套
15 0 qid:655 1:1.608 2:0.123 3:0.618 4:1.094 5:0.624 6:0.261 7:0.701 8:0.789 9:0.495 10:0.087 #3m/思高 随手粘衣物用56张+15张替换
```

图 3 LambdaMART 训练样本

- 先遍历所有的训练数据，计算每个 pair 互换位置导致的指标变化 deltaNDCG 以及 lambda；
- 创建回归树拟合第一步生成的 lambda，生成一颗叶子节点数为 L 的回归树；
- 对这棵树的每个叶子节点通过预测的 regression lambda 计算每个叶子节点的输出值；
- 更新模型，将当前学习到的回归树加到已有的模型中，用学习率 shrinkage 系数做 regularization。

RankLib 提供了简单的命令行，只需根据实际的需要配置好参数，就可以实现训练模型、保存模型的功能。下面是我们在工程中使用的参数配置示例：

```
java -jar ~/bin/RankLib.jar
-train ~/train_all.csv -gmax 4
-tvs 0.8 -norm zscore -ranker 6
-metric2t NDCG@60 -tree 1000 -leaf
```

Algorithm: LambdaMART

```

set number of trees  $N$ , number of training samples  $m$ , number of leaves per tree  $L$ ,
learning rate  $\eta$ 
for  $i = 0$  to  $m$  do
     $F_0(x_i) = \text{BaseModel}(x_i)$  //If BaseModel is empty, set  $F_0(x_i) = 0$ 
end for
for  $k = 1$  to  $N$  do
    for  $i = 0$  to  $m$  do
         $y_i = \lambda_i$ 
         $w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$ 
    end for
     $\{R_{lk}\}_{l=1}^L$  // Create  $L$  leaf tree on  $\{x_i, y_i\}_{i=1}^m$ 
     $\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$  // Assign leaf values based on Newton step.
     $F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$  // Take step with learning rate  $\eta$ .
end for

```

图 4 LambdaMART 训练过程

```
10 -shrinkage 0.1 -save ~/models/
learned_lambdamart_model.mod
```

参数说明

- `-train`是必须的，表示训练样本所在的文件名；
- `-ranker`是必须的，表示指定的机器学习算法，而6就是LambdaMART；
- `-gmax`是可选的，指定训练目标相关性的最大等级，默认是4，代表5档评分，即 $\{0, 1, 2, 3, 4\}$ ；
- `-tvs`是可选的，设置样本中用于训练的数据比例，即train数据：validation的比是0.8:0.2；

- `-norm`是可选的，指定特征归一化的方法，默认没有归一化，`zscore`代表采用均方误差来归一化；
- `-metric2t`是可选的，训练数据的评测方法，默认是ERR@10；
- `-tree`是可选的，指定lambdamart使用的树的数目，默认是1000；
- `-leaf`是可选的，指定lambdamart每棵树的叶节点数，默认是10；
- `-shrinkage`是可选的，指定lambdamart的学习率，默认是0.1；
- `-save`是可选的，用于保存模型。

上述命令执行结束后会在`-save`指定的目录下产生一个如图5的模型文件，该模型文件也就是可以用于生产环境的机器排序模型。

离线评测

在排序模型训练完成之后，我们需要准备与训练样本相同数据格式的离线测试样本，这个测试样本尽量选取和训练样本不同的数据集。

```
java -jar ~/bin/RankLib.jar
-load ~/models/learned_lambdaMart_
model.mod -test ~/test_samples.csv
-metric2T NDCG@60 -score ~/rerank_
scores.txt
```

这里`-load`对应的参数就是在离线训练中得到的排序模型，`-test`对应的

参数就是测试样本集，`-metric2T`使用和训练过程相同的评价方式，`-score`对应的参数就是保存测试样本集中每个query下商品的得分。

【实验样本】为了离线测试 LTR 的模型效果，选择了两个流量差异较大的类目，暂且称为类目 A 和类目 B，其中类目 A 的流量大约是类目 B 的 40 倍。同时选择不同平台、不同时间段内的数据作为训练样本。

【评测指标】除了使用评价排序效果的 NDCG 以外，出于商业因素的考虑，我们还选择了 4 个评价排序位置变动情况的指标，其中 AverageOverlapScore 和 RBOScore 指标越大表示正常排序和 LTR

```
<ensemble>
  <tree id="1" weight="0.1">
    <split>
      <feature> 10 </feature>
      <threshold> 0.05859375 </threshold>
      <split pos="left">
        <feature> 11 </feature>
        <threshold> 0.015625 </threshold>
        <split pos="left">
          <feature> 15 </feature>
          <threshold> 0.4140625 </threshold>
          <split pos="left">
            <output> -0.8295480608940125 </output>
          </split>
          <split pos="right">
            <output> 0.6033934950828552 </output>
          </split>
        </split>
        <split pos="right">
          <feature> 11 </feature>
          <threshold> 0.5390625 </threshold>
          <split pos="left">
            <output> 0.032923102378845215 </output>
          </split>
          <split pos="right">
            <output> 1.0059690475463867 </output>
          </split>
        </split>
      </split>
    </tree>
  </ensemble>
```

图 5 LambdaMART 训练产生的模型

排序越接近，LikeNDCG1 和 LikeNDCG2 指标越小表示正常排序和 LTR 排序越接近。

可以看出，同样用一周的数据，PC+IOS+Android 的评测效果在多个指标上要好于只用 PC 的效果；而同样在 PC 端，两周数据的评测效果也好于一周；但是这个结论并不是普遍性，对于不同类目，需要具体问题具体分析，进而确定表现较好的模型。

另外，为了验证 LambdaMART 的不同训练参数对预测效率和预测效果的差异，我们也进行了其它 4 组实验，分别

表格 3 类目 A 的离线评测效果

类目	A				
	PC	PC + IOS + ANDROID	PC	PC	PC
训练数据平台	PC	PC + IOS + ANDROID	PC	PC	PC
训练数据时间段	一周	一周	两周	一周	一周
NDCG@60 on training data	0.5617	0.5365	0.5313	0.5253	0.5778
NDCG@60 on validation data	0.5391	0.5411	0.5605	0.5138	0.5372
AverageOverlapScore	0.3535	0.3751	0.3710	0.4254	0.3442
RBOScore	0.4236	0.42823	0.4306	0.4591	0.4005
LikeNDCG1	3001.8343	2997.8116	2957.3804	3591.0862	3108.4491
LikeNDCG2	148.5160	147.1333	141.9098	176.5259	156.8397

是如下：

1. 默认参数设置
2. 在1的基础上调整tvs参数
3. 在2的基础上调整leaf参数
4. 在3的基础上调整tree和shrinkage 参数

4 个模型的排序结果如图 6 所示，其中横坐标 P01 表示线上排序 1-8，P02 表示线上排序 9-16，纵坐标表示这 8 个位置与 LTR 排序下商品位置变动数量。

不同模型的指标对比如下，模型 2 的 4 个指标均为最佳。

综合 LambdaMART 在 Training Data

表格 4 类目 B 的离线评测效果

类目	B				
	PC	PC + IOS + ANDROID	PC	PC	PC + IOS + ANDROID
训练数据平台	PC	PC + IOS + ANDROID	PC	PC	PC + IOS + ANDROID
训练数据时间段	一周	一周	两周	一周	一周
NDCG@60 on training data	0.6834	0.4882	0.6148	0.7887	0.4955
NDCG@60 on validation data	0.5193	0.5356	0.5549	0.5522	0.5042
AverageOverlapScore	0.2273	0.3438	0.3100	0.2207	0.1965
RBOScore	0.2597	0.3636	0.3495	0.2573	0.2344
LikeNDCG1	3166.5950	2765.4566	2022.0802	3202.8392	3142.5835
LikeNDCG2	208.9072	156.8297	110.9762	201.1636	188.5675

和 Validation Data 的迭代次数和训练时间来看，随着参数的增加，训练时间和迭代次数都有增加；但是当去除噪音 Click 的数据后，训练时间和迭代次数有明显的降低。

在线应用

【模型加载】 离线测评时，可以用命令行的方式加载模型，读取数据文件，进而测试模型性能。显然这种方式并不适合在线应用，因此我们需要提取 RankLib 加载模型和在线打分的代码。为了有效进行系统地管理，我们将模型

文件统一存放在 MySQL 表中，并利用字段区分类目和平台等信息。

【在线测试】 为了排除用户固有行为特性的影响，从而更加准确地比较 LTR 排序和人工排序的各项指标，在进行 A/B Test 之前，我们首先进行了一段时间的 AA 测试，从而选出各项指标较为接近的桶，再比较这些桶在 LTR 排序和人工排序之间差异。

将训练后的 LTR 模型按照图 2 的框架应用到生产环境中，在整个测试期间，我们观察了不同时间周期、不同平台数

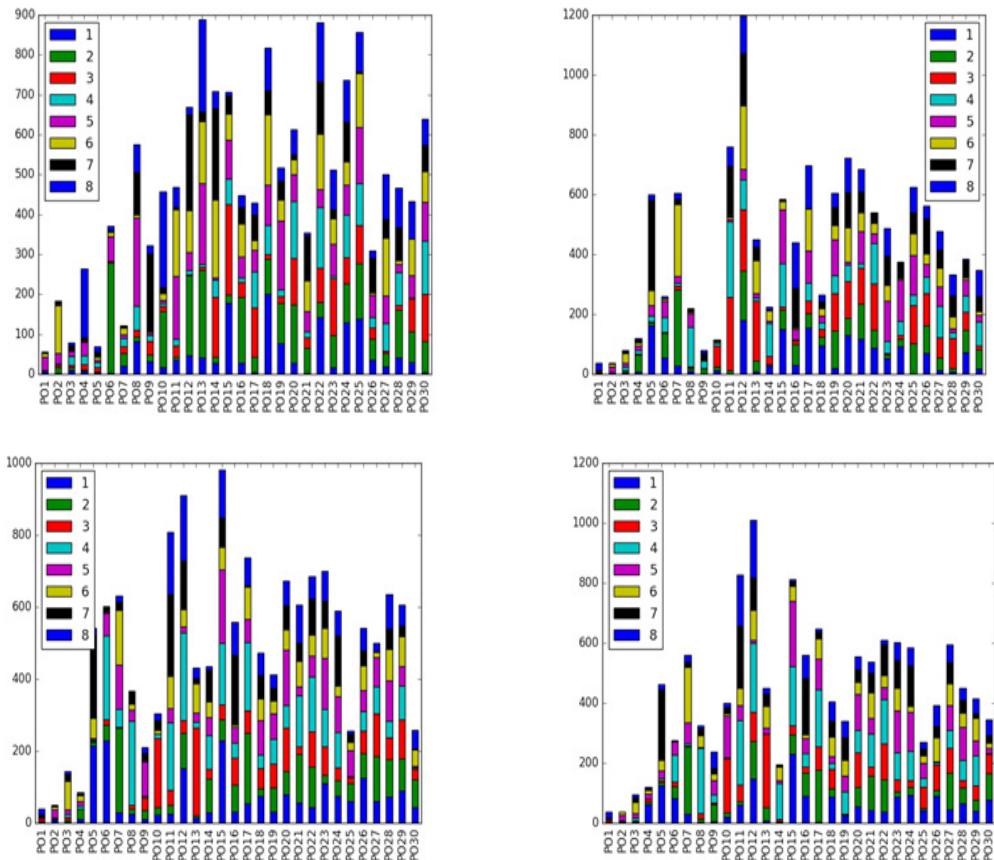


图 6 4个模型的排序结果

据产生的多个模型。总体来讲，LTR 模型对“搜索导航页到商详页的 CTR”在 2016 年 11.11 活动期间有约 8.7% 的提升，并且“订单转化率”在 11.11 活动期间有约 4.5% 的提升。（其中 LTR_1 和 LTR_2 是我们设置的不同流量分桶，用于观测不同分桶对模型的影响）

总结

总的来说，人工规则与机器排序是

紧密结合的。如果排序的量化目标不太明确，则人工规则就更适合。电商搜索不仅要考虑查询相关性，还要考虑销售额和订单转化率，因此电商搜索往往有较多的业务规则。如 eBay 和 Google 在搜索排序方面就结合了人工业务规则，而广告排序一般更依赖于机器排序，因为其优化目标比较明确。所以我们也会在当前成果的基础上，将机器排序学习进一步应用到推荐和广告系统上。

魅族11.11： 架构为酒业务当歌，解读电商运维之困

作者 刘文彬

写在前面

电商是什么？拆开来讲就是电子和商务。之前最关注的是电子，就是我们所说的技术。而商务就是业务。之所以简称为电商，是因为两者非常紧密的相连着。

今年是我在魅族电商业务部的第三年：之前从 Flyme 到官网、论坛等业务我都经历了一遍；电商业务对于我来说是最有挑战的业务之一。在从事电商的运维后，特别是经历了双十一、大型抢购、秒杀活动后，更加真实的清楚了电商的真正含义，得出了之前从未有过的感受。

1. 千万不要低头做技术，学会真正从业务角度去思考。
2. 不要低估技术的力量，技术是可以推动业务的。我们站在技术这一方，与业务是互相影响互相牵引的。
3. 精通你手中的业务，是每一个环节。小到一张优惠券的生成和去向的过程。

当你对手中的业务非常非常熟悉，甚至有一种“我是否能做产品经理了”的错觉——当你熟悉到这种境界，你会发现曾经低头做技术时所发现不了的东西。因为随着对业务理解越来越清晰，



你自然会知道架构中哪些集群是可以合并的，哪些资源是浪费的，哪些环节是需要做扩容的。你可以很好的去做架构优化，随之而来的就是更好的成本控制。你不用再去猜，这里我需不需要加机器，加多少台？这些机器是不是可以撤掉？并且你还可以参与主导业务活动。

这里跟大家说了一个真实存在的故事：有一次市场部打算在某一周做秒杀活动，但是由于他们担心网站抗不住，决定将活动分在五天分别举行。但是基于对业务和技术的熟悉理解，我认为活动完全可以在一天内完成。于是，我从了两方面工作：首先，在活动来临之前，

通过活动类型分析、流量来源分析、产品热度分析、用户习惯分析等等，估算好了大致流量，提前做了架构扩容。其次，热度高和热度一般的活动安排在了在用户最活跃的时间段，合理安排秒杀活动顺序。然后很有信心的告诉市场部同事：“你们要做的这期秒杀活动在一天内完成绝对没有问题，放心去做”。

最后，活动比预期更成功，用户反响非常好。既保证了用户参与度最大化，也保证了网站的稳定。后台同事也夸奖道：没有想到仅用之前 2/3 的机器就扛过这期大活动！这都要得益于将资源整合和优化。所以我认为：技术一

定要懂业务，并且要吃透！

从业务到技术的备战

魅族备战双十一的工作主要分成两个部分：其一为交易前，其二为交易后。

交易前环节：我们首先评估双十一备货方案，然后从优惠力度、所争取到的流量入口资源等级评估，在下线若干非核心功能，例如服务类和保险类业务。同时，与我们的云厂商相关部门沟通合作，以获取到优先级较高的TOP接口。

交易后环节：这部分的重点工作是保障各环节服务稳定性和链路稳定性。双十一订单虽然不在魅族商城生成，但从淘系引来的订单流量量级非常之大，量变引发质变，不能用常规手段处理订单。因此，我们在业务架构上做水平拆分，水平扩容 SLB 集群、使用 DTS，按照多线程作业的方式从淘系接口处抓取订单，在 ERP 系统内先行做订单格式中转，再传递到订单中心 OMS 处做订单处理、推单入仓等操作，随后再分批推送订单数据到物流 WMS 接口并完成订单入仓的操作。

业务难点及对策

整个双十一对于我们来说，难度最大的地方在于保证全链路稳定。因为仓内操作有一部分人工作业，存在效率瓶

颈。如何喂饱仓内操作部分是最大的挑战，我们的订单需要赶在全网大面积订单出库前，在第一时间就能搭上首批物流运能。从出库这个环节倒推，需要每个系统之间的交接耗时尽可能最短，只处理最需要的业务，大量采用多线程作业方式，从而确保百万级订单在淘系入口抓取下来后，能以最快的速度完成订单格式转换、订单处理、订单推仓、出仓回传，每一个环节都需要对现行方案做反复多次的验证，并组织各环节做链路压测而非普通的性能压测，最薄弱的一环将会直接拖慢整个流程的效率。

从业务开始，分析此次双十一我们卖的是不是热销产品、是否有优惠产品、是否有秒杀类产品。接下来再分析流量来源，我们从哪里做了导流、导了多少流量。结合以上两点，我们能大概算出实际到我们平台的流量有多少。有了具体流量，我们就能做集群扩容、系统调优，还能评估出哪些功能可以上线、哪些不能上线。这样就可以很好地提前梳理出，哪些是可能会发生故障的点。

整体架构解读

我们核心业务基于云厂商的服务做了多套集群，后端 RS 独立于每套集群。用 DNSPod 进行分流，分别以地区进行区分，如华东、华南等。RS 搭载

了 OpenResty（后文简称 OR）和 Java，其中 OR 我们与 Lua 语言进行搭配。OR 在处理大并发大流量下，比 Nginx 强大并且灵活很多。并且在性能上远远高于 Nginx，其次能很好的做到限流，所以我们线上基本上取代了 Nginx（见图 1）。

缓存我们使用的 Redis，其中我们自行开发了 zoowatchdog，负载节

点的健康检查和主从切换。当初并没有合适的高可用方案，原版 twmproxy 没有和 ZooKeeper 结合的。那我们如何维护 ZooKeeper 上的 redis 数据呢？结合这些问题，我们自行研发了 zoowatchdog。一方面做监控，一方面维护 ZooKeeper 上的 Redis 信息（见图 2）。

在数据库上，我们实现了分表分库。

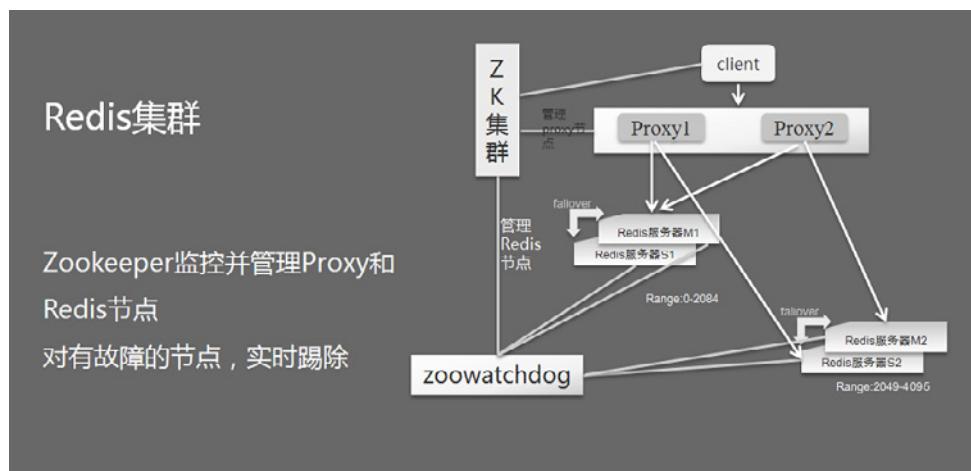
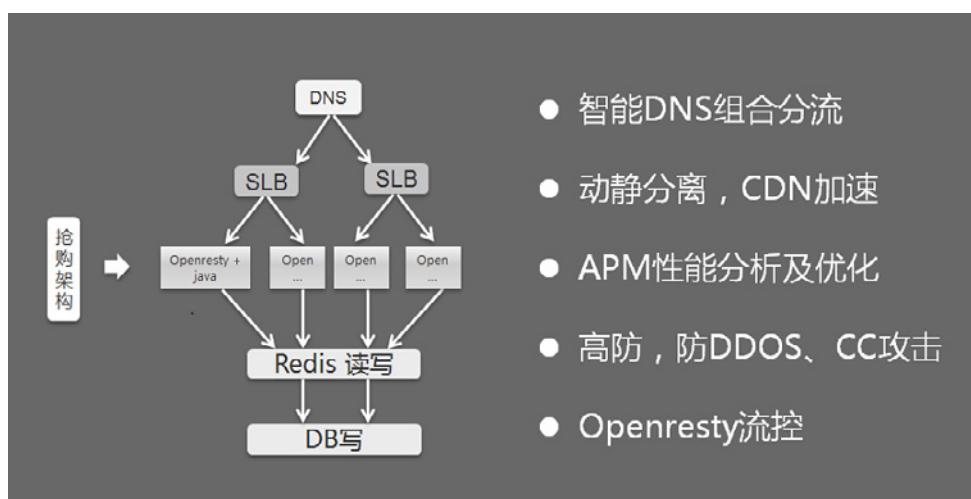


图 1、图 2

有一个主库、多个写库、多个读库（不管读写库都有主备），每个读库仅提供一个业务去读（因为我们业务的读写库压力都非常大，所以我们首先把写库做成多个可写入点。读库分为业务来读，每个读库仅提供一个业务去读，互不影响。）用户调度我们是根据用户 ID，经过取模算法分配到不同的写库、读库上。每个库又分 32 张表，这里则是通过订单 ID 再决定具体读 / 写哪张表，有效地保证了在大并发下的读写效率（见图 3）。

三大技术点护航大促

1. 峰值应对

在监控布局上，我们采用了 Zabbix

和 CAT 两套监控进行。Zabbix 监控硬件、网络系统等。CAT 监控业务，并且结合听云 APM。报警接入我们的微信和钉钉以及短信（工作时间发钉钉和短信，非工作时间发微信和短信）。

活动系统利用阿里云的 ESS 弹出服务可以做到及时扩容。只需提前将镜像准备好，再设定触发规则，如负载达到多少或者定时弹出。

总体而言，流量控制工作依次为流量清洗、限流、限量、防火墙和紧急预案。在限流工作上，通过 OpenResty 和代码层来控制，OpenResty 控制排队数量，根据网站负载来实时调整。代码层控制单个 IP 发起的请求数和下单数。非系统核心代码，许多功能我们都做成了开

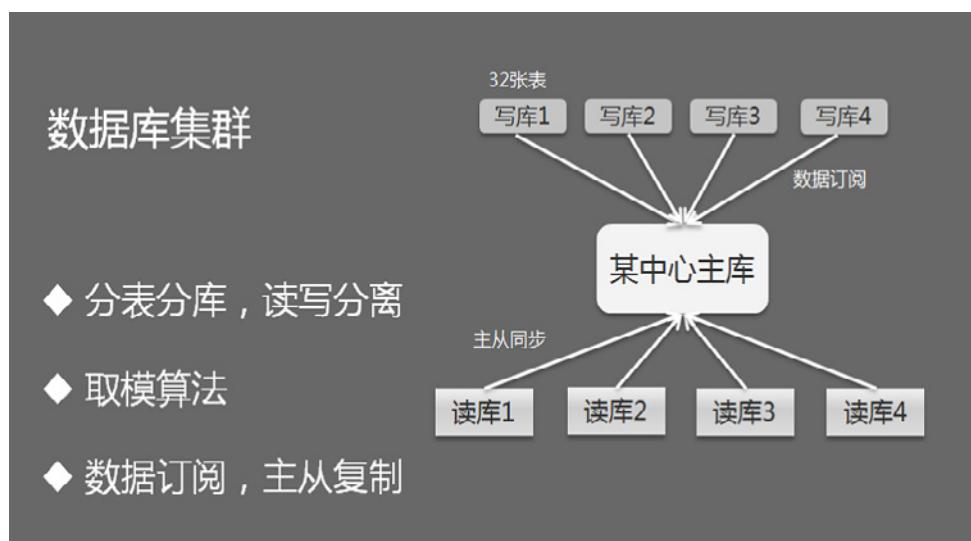


图 3

关。如此，在大并发来临时，我们可以进行降级：根据当前网站负载决定关闭哪些功能。同时我们还准备了紧急预案，当流量突破上面层层抵达系统时如果仍足以击垮网站，那么我们将仅仅保障核心用户的正常使用，非核心用户则会暂时不提供服务。

2. 快速访问

动静分离，动态服务端缓存会话和热点数据等。静态 CDN 加速，并设计了 CDN 容灾架构，以及 CDN 流量划分。并通过听云 APM 做到性能监控和页面打开速度监控。主要做了一下几个优化和措施：

1. 大图片延迟加载；
2. 小图合并，或内嵌，减少请求；
3. 静态资源的压缩、合并；
4. 静态资源非覆盖式发布，持久化缓存（进行中）；
5. 部分数据，异步请求，动态渲染；
6. 访问频率较高的页面，首页、商品详情页等静态化，及 SEO 优化；
7. 提取页面公共静态资源做预加载，预缓存；
8. 商城移动端首页的首屏添加样式的内嵌，保证首屏加载的可用性；
9. 后端 Server 缓存页面（如首页）15 秒，以及 Redis 热点数据缓存；

10. OpenResty 限流，防止恶意请求。

基于以上这些优化，我们的首屏时间控制在 2s 以内。

3. 严治黄牛

黄牛令人痛恶深绝，这是电商行业业内共识。黄牛不仅影响了正常用户的利益，同时对应非正常的请求给网站带来很大压力。黄牛带来的流量可不能小看，完全有可能引起雪崩效应。在普通用户们正常进行网站请求时，黄牛就在疯狂的刷页面、接口；而当活动即将开始时，普通用户也开始狂点鼠标时，黄牛依旧在疯狂请求。那么，如果对于大促活动你没有提前将黄牛流量算入此次活动，很有可能影响活动正常开展。

魅族电商通过自建用户体系、活动分级、系统简单、用户评分、流量过滤等研发，做到了行业领先的防黄牛技术。不仅保护了正常用户的利益，同样也减轻部分网站负载、运营人员的工作量。

自建用户体系： 用户来魅族商城购买产品时，首先需要先建一个账号，这个账号与手机号码绑定。下单时必须是处于登录状态。那么这样的好处是什么呢？身份识别。那么建立了自己的用户体系后该怎么办呢？需要完成数据沉淀和积累，以了解用户的购买习惯、购买动机和用户活跃度。

用户打分：了解用户的购买习惯、购买动机、活跃度后，我们再通过数据分析给这个用户得出一个信用分值。可分为 S、A、B、C、D 共 5 个等级。

活动分级：当要举办一个抢购或者是秒杀活动时，可以设置一个门槛以防止黄牛参加，比如可以假设只有 S 级的用户才能参加。门槛等级根据活动及活动的产品热度来决定。

流量过滤：通过防火墙、OR、程序等措施层层限制请求数、下单数及排队数等，以此来控制黄牛发起的非正常请求（如单 IP 一秒发起 100 次请求，或者绕过前系统，直接请求下单接口等方式）。

系统筛选：通过已经下单成功的订单信息区进行分析，看看哪些用户是黄牛。假设这次下单成功的 10000 单中，有 20 单是下单地址或者姓名类似等（订单信息有批量生成的嫌疑）。

根据魅族的经验来看，自建用户体系 + 用户打分 + 活动分级 + 流量控制 + 系统筛选，这一套组合拳够黄牛吃撑了。

总结

目前业务飞速增长，留给我们研发团队追赶业务需求脚步的时间并不多。所以需要以一种快速反应、快速建立的

态度来应对大并发环境下各种问题，我们没有那么多时间去淡定地构建一个伟大又超凡的架构。我们只能沉着冷静的去构建最适合现状、最能快速扩展和响应的架构，以此为飞速发展的业务提供一个当下最坚固的地基。

我个人加入魅族已经三年有余，可以说是看着魅族从一个青春期的男孩，经历种种冷眼与嘲笑后，迅速成长为一个男人。这一路走来，遇到的问题从来没有少过，也曾让我手足无措、茫然四顾。如今回顾总结，从业务的角度出发，重新去审视技术本身，发现其实没有那么难以抉择和难以舍弃。坚决的去选择最适合当下的，自己用的最舒服的架构吧。

苏宁11.11： 苏宁易购移动端的架构优化实践

作者 苏宁易购移动端技术团队

双11，不仅是广大商家和消费者们的饕餮盛宴，同时也是各家电商公司的技术研发力量的实力体现。2016年苏宁易购双11首小时订单量增287%，移动端占比86%。为了给用户带来更好的体验，苏宁易购移动端团队一直以来追求“极速、稳定、安全”的目标，围绕这一共同目标，我们以小团队作战、敏捷的开发模式，开展了一系列工作，其中包括：

1. 客户端架构解耦：为了保障App性能的稳定，适应业务的快熟发展及模块化，我们完成了客户端架构的改造，实现了系统横向和纵向解耦、动态化、组件化等。
2. 客户端性能优化：为了追求更好的响应性能，同时兼顾移动端安全

性，我们进行了图片渲染、静态资源上的全方面加速，并且对网络链路进行了深度的优化。

3. 监控体系：一切以数据为依据，实现了移动App监控和数据分析系统。

1. 客户端架构的改造

苏宁易购客户端从11年开始做起，随着移动技术的发展，架构也发生了很大的变化。

老架构的问题主要是代码耦合问题：

- 横向代码耦合：业务代码之间没有明显的模块边界，哪里需要用到其它的功能就直接包含一下其它模块业务的头文件，生成所需功能的对象来直接使用，模块之间调用呈网状结构。

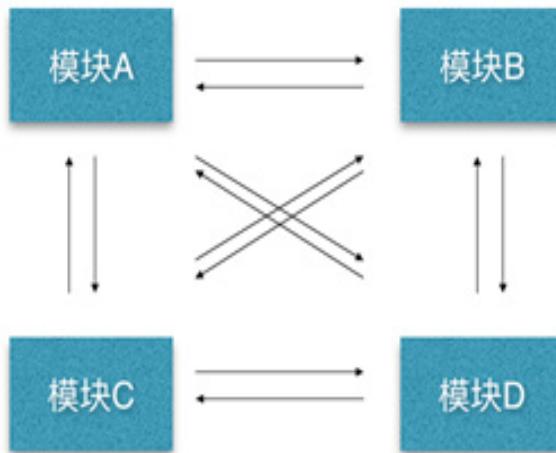


图 1 网状引用结构：简单的逻辑变的复杂且混乱

- 纵向代码耦合：由于没有合理的进行纵向分层，很多基础功能代码里包含了大量的业务功能，业务代码与基础功能代码没有严格的逻辑界线（见图1）。

1.1 客户端分层模型，解决纵向耦合

首先，我们对客户端进行了分层处理。客户端在纵向上划分成三层：业务层、服务层、基础层。在横向上，我们引入了功能模块的概念，将客户端抽象为一个个的模块所组建成的一个整体。分层思想如图 2 所示。

- 基础层：对基础代码做了很多整理，把基础代码里所有的业务代码都删除掉。另外我们自己重新封装了数据库、UI、网络、自定义控件、常用的第三方库、常用的自定

义宏等内容，它们构成了我们的基础层。

- 服务层：在基础层成型以后，我们把一些通用的功能进行了封装，例如：埋点、url管理、网络状态检测、缓存管理等。它们使用基础层来搭建自己的功能，为业务层提供服务。
- 业务层：对所有的业务功能进行模块划分，大致划分为首页、购物车、嗨购、会员、搜索等模块，所有模块都使用统一的真实目录结构，将模块从逻辑上和物理上都归于一个目录下，不同的模块之间从物理上进行目录隔离。

1.2 中介型引用结构，解决横向耦合

解除横向耦合的原理就是将之前直

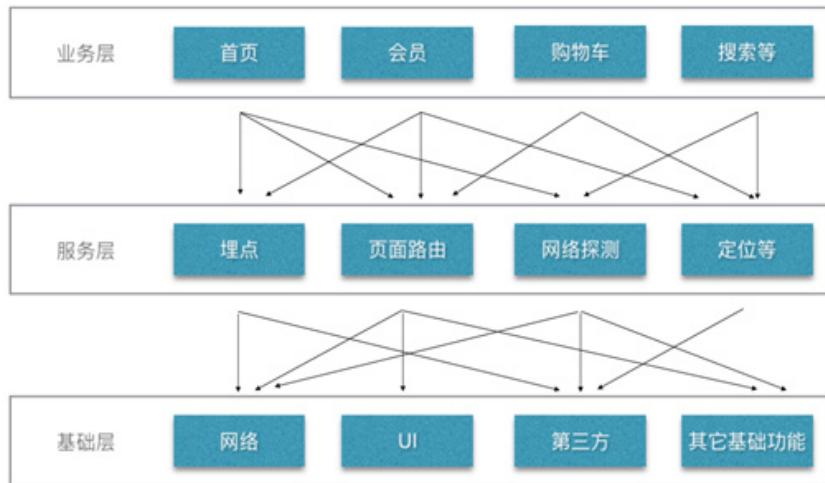


图 2 新架构下客户端分层模型

接的网状引用结构改变为间接的中介型引用结构。加入一个模块管理器，所有的模块把需要提供给其它模块调用的功能，都预先在管理器中进行注册。在使用时，使用者向管理器进行申请，即可以得到对应的功能对象。在这种结构下，模块的插拔对其它功能影响非常小，耦合也非常小，如图 3 所示。

在具体的实现方式上，目前常用的解耦方式大致有 protocol 方式和 url 方式，我们两种都有用到：在对服务层的引用上，我们更多使用 protocol 方式；在对某些业务模块的调用上，我们更多使用 url 方式。

protocol 解耦： protocol 方式是 id 的一种灵活使用，体现了面向接口编程的思想，模块要导出的功能不是一个

具体的头文件，而是用 protocol 来表示功能的接口的 .h 文件。使用者在使用时得到的是一个 id，但可以调用具体的协议方法来完成自己需要的功能。另外，由于得到的是 id，所以功能模块也实现了接口与实现的分离，例如根据不同的情况，被调用者可以通过 id 返回不同的实现对象，而使用者无需做任何变动。目前我们在埋点、定位、分享等服务层上使用的是这种方式的调用。

动态化和 url 解耦： 随着移动端架构的发展，出现了动态化的概念，即是把客户端的部分功能在原生和 H5、Weex、React Native 等实现方式之间进行灵活的切换。这样客户端可以进行动态的发布、降级、切换实现等功能而不用重新发版本。易购客户端里我们主要

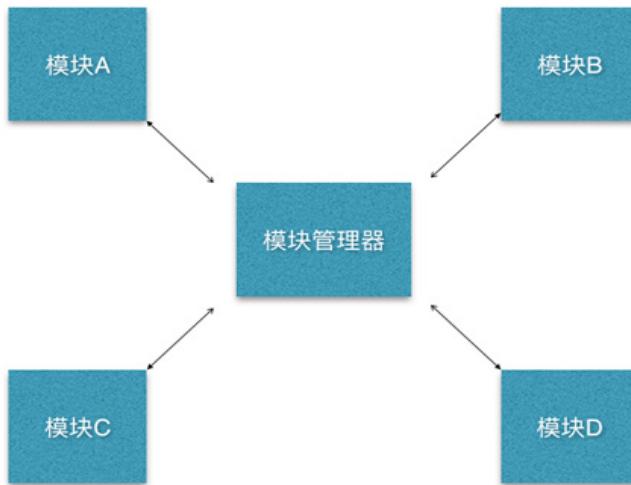


图3 中介型引用结构：调用逻辑清晰

采用了 url 的方式来支持这种动态化。

我们把需要动态化的原生页面按业务线 / 模块 / 页面名的形式定义它的 url，在页面的类对象生成后注册一个 block 到 url 跳转管理器里，这个 block 会在使用者调用页面 url 时被执行，生成一个页面对象进行返回和展示，这样就得到了 url 化的原生页面跳转。同理，H5、Weex 页面在客户端也需要一个原生页面作为容器来加载它们，因此我们把这两个容器做为单独的模块，并且使用来进行跳转。如图 4 所示。

我们在 url 跳转管理器上做了一些操作，它可以下载服务端的配置列表，根据配置列表来改变当前的跳转方向，跳到不同的容器上来进行展示，由此我们也实现了通过服务端的配置来达到客

户端实现切换的目的。

2. H5容器的性能优化

在 H5 容器层面，我们也进行了多项的优化来提升页面的加载速度，并且减少页面的下载数据量。

2.1 静态资源预加载

我们把前端页面进行了动静分离，把一些页面会使用到的静态资源预先下载到客户端。把容器的资源请求进行拦截，将本地已经下好的文件内容返回给它进行展示，这样就省去了即时网络通信的时间，提升了页面的加载性能。

2.2 WebP图片格式的使用

H5 容器的图片我们全部使用 WebP 格式。我们把容器对图片的请求进行了拦截，用 native 代码重新发起一个 WebP 图片的请求，在请求成功后 native

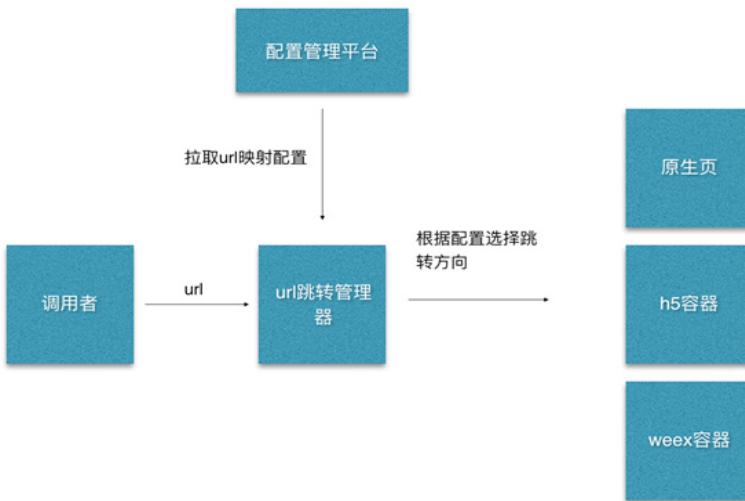


图 4 url 解耦模型

把 WebP 图片转换为 png 的数据返回给 H5 容器进行展示，使用 WebP 后我们页面需要下载的图片数据量减少了一半以上。

2.3 容器渲染性能优化

为了提升容器的渲染性能，并且解决 H5 页面长列表内存不断增大等问题，我们引入了 Weex。Weex 通过将 js 转化为 native 代码渲染，有效的提升了页面的渲染性能，Weex 提供的长列表实现机制通过 native 的单元格重用也很好的解决了长列表问题。我们把 H5 容器和 Weex 容器进行了动态化处理，通过 url 映射可以任意的在两者之间进行转换。

3. 网络链路的优化

移动网络环境与有线网络存在着很

大的差异与区别。因此，移动网络对我们的性能策略优化提出了新的、独特的要求。在我们未作优化前，2G/3G 网络下一个简单的资源（下载用时在 0.1s）请求在网络传输中可能都会耗费很长的时间（1.345s，见图 5）。

另一方面，随着近年来苏宁易购移动端的不断发展和引流，用户数持续递增，网络劫持的问题也逐渐严重起来，比如：页面加载白屏，APP 下端弹出小广告，甚至无法提交订单。保证移动网络下，用户隐私和数据传输安全性也成为了我们这些年来重点工作之一。

为了解决性能与安全两方面的问题与瓶颈，我们采用了以下技术方案：

3.1 使用HTTP/2——链路复用，加速传输

HTTP/2 采用二进制帧格式而非文本



图 5 下载用时只占端到端时间很少的一部分

格式进行传输，突破了请求并发数的限制，能够实现完全的多路复用，使页面的传输效率和建连复用效益最大化。我们在移动端部分促销页面使用了 HTTP/2 传输，在 3G/4G/Wifi 网络下，其效果是令人欣喜的（见图 6）。

3.2 HttpDNS 寻址——DNS 解析安全、精确、快速

HttpDNS 是使用 HTTP 协议向 DNS 服务器的 80 端口进行请求，代替传统的 DNS 协议向 DNS 服务器的 53 端口进行请求，绕开了运营商的 Local DNS，从而避免了使用运营商 Local DNS 造成的劫持和跨网问题。

在实现上，我们会维护一张域名列表，将域名解析值预取到客户端本地的 DNSCache 中，预取优先调用 HttpDNS 接口，如果获取不到数据，则直接从 localDNS 取数据，并设置一个独立的线程作为定时器，根据 TTL 过期时间来检查 domain 是否需要更新。

HttpDNS 为我们带来了安全和性能

上诸多的收益。

3.3 HTTPS代理——保证传输链路的绝对安全

HTTPS 能够给用户带来更安全的网络体验、更好的隐私保护。然而，HTTPS 增加了 TLS 握手环节，再加上应用数据传输需要经过对称加密，对性能提出了更大的挑战。作为一个好的架构，一定要均衡安全和性能两方面，如果让天秤向任何一方倾斜过多，都会影响最终的用户体验。

因此，结合服务端，我们做了很多优化工作：

- 基于链路的优化

建立连接的延迟体现在每个 SSL 连接上，因此尽早完成 SSL 握手是优化工作的重点。对于普通的图片资源和文档请求，我们在 CDN 上完成 SSL 卸载；对于涉及用户信息的受限资源和脚本，我们在内网防火墙上完成 SSL 卸载。

- 基于 SSL 协议的优化

在 SSL 协议优化方面，我们在服务

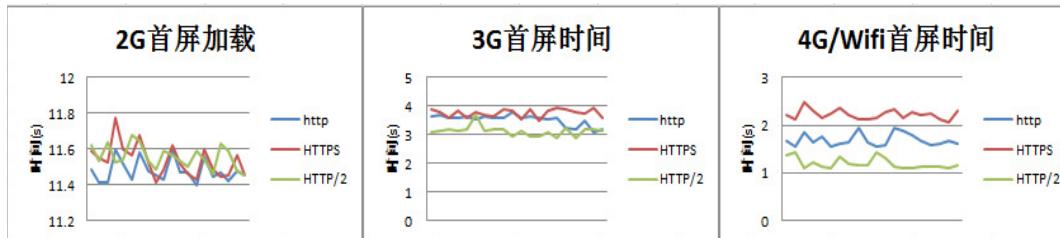


图 6 HTTP/2 加速效果与 HTTP/1.1 对比

端支持 ALPN 协议，使用适合 Forward Secrecy 的加密算法，开启了 False Start，客户端在第二次 SSL 握手的同时就可以发送应用数据，减少了一次 RTT 时间。

另外，我们启用了 Session ID 和 Session Ticket 的会话恢复技术，对同一个用户，在多个连接间共享协商后的安全密钥。并将 Session 信息缓存到 Redis 中，通过一台中心服务器统一管理，解决了集群情况下 Session ID 无法共享的问题。

- 基于证书和加密套件的优化

在证书链优化方面，由于 TCP 初始拥塞窗口的存在，如果证书太长可能会产生额外的往返开销，移动端采用的证书链都是“站点证书 - 中间证书 - 根证书”三级的，同时服务端只发送站点证书和中间证书，根证书部署在客户端，将证书链控制在 3KB 以内。为了避免不必要的证书过期校验，我们在服务端开启了 OCSP Stapling。

在加密套件的选择上，优先选择 ECDHE-RSA-AES128-GCM-SHA256，综合安全、性能和开销，是最优的选择。

当然，我们还实现了 H5 内容压缩与图片的智能适配，针对 HTML、JS、CSS 等格式进行无损压缩；根据终端网络制式智能调整图片分发尺寸，提升终端展现性能。

经过我们以上的改造，从 10 大城市 3 大运营商上采集的数据分析，客户端访问速度提速明显，Android 大概提速一倍，IOS 也有 60% 左右的提高；错误率也明显减少了 80%（见图 7 和图 8）。

4. 移动App性能监控系统

监控是系统的眼睛，尤其是针对线上电商系统。监控能让我们知道系统的运行状态，在我们进行优化时，监控能告诉我们优化的效果，在系统出问题时，也能让我们知道系统发生了什么。可以说，一切都是建立在完善监控的监控体系下的，因此，最后咱们花点时间来聊一聊易购移动端的监控手段。



图 7、图 8 安卓/IOS 的加速效果对比

移动端监控体系既不能只看客户端，也不能只看服务端，必须将客户端、网络，服务端相打通，形成完整端到端的调用链，才能让问题无处遁形(见图9)。

我们开发了专门用于App监控和数据统计分析系统。能够从五个维度：移动应用崩溃、移动应用错误、移动应用请求响应时间、移动应用交互性能、运营商网络响应时间来帮助我们分析问题（见图10）。

5. 结语

路漫漫其修远兮，吾将上下而求索。经过了这些年的努力，客户端的架构升级让易购App更加稳定的运转起来，移动端加速各项技术的应用让用户体验更加的极速和安全，再通过监控系统的全方位24小时实时监控和保障，真正做到了我们的良苦用心，用户的畅快体验。然而，追求极致的道路是没有终点



图 9 苏宁易购移动端端到端监控体系 / 图 10 苏宁易购移动端平均响应时间监控

的，并且前行的过程中将永远有挑战，易购移动端团队将继续努力下去！

本文由苏宁易购移动端技术团队撰写。苏宁云商最早成立的完善技术团队之一。面向苏宁易购数亿用户群体，目前我们拥有一整套完备移动研发能力，在移动框架、性能优化、自动化测试、移动 app 监控及前端容器框架等各个方面已拥有近百位移动技术专家。苏宁易

购双 11 首个小时订单量增 287%，移动端占比 86%；我们致力于整合苏宁线上线下渠道，立足于移动开发技术，扩展移动端优秀产品，为用户提供极致移动体验。苏宁易购移动端技术团队将会在移动端动态化，容器化，服务化等多方面深耕细作，并将会和广大的同行朋友们一起，努力把移动技术越做越好。



唯品会11.11：

频繁黑匣架构背景下，看唯品会的革命性重构

作者 张广平

eBay和唯品会的电商文化缩影

eBay 是一个老牌的互联网公司，是曾经全球最大的交易平台。我有幸在 eBay 中国研发中心工作接近 10 年，曾经在支付平台、电商平台、云平台等不同开发部门工作过。eBay 电商系统设计非常复杂，将系统划分为一个个小模块，每个团队和其中每个人负责一个产品的一个小模块。这种系统结构经过长期演练已经非常严谨、成熟和稳定。

eBay 集中了很多优秀人才，通过参与其中，我个人收获很大，但由于每个

人所负责的模块相对来说比较小，所做的变化被限制在所负责的模块中，很难参与到其它系统模块，而且系统改造比较保守，更追求稳定。再加上地域的原因，中国员工很难参与到核心模块当中。久而久之，对个人发展来说就产生了障碍。

相比 eBay，唯品会有了很大空间，正好碰到公司大发展阶段，业务发展非常快，但电商系统远远跟不上业务发展需要，系统面临大改造。正好把自己以前在 eBay 的工作经验发挥出来，我和

我的团队深入到各个业务团队中，为公司重大项目提供架构设计方案。在一年左右的时间内，主导并完成了公司电商核心系统的架构设计，随后也支持开发团队完成了大部分实施。同时组建公司架构评审委员会，对公司关键项目进行评审。

谈到两个公司的文化，eBay 有着大公司的一切特征，公司有比较严谨的组织和计划，但应对变化的节奏较慢，技术上偏保守，追求稳定，所以一些新技术的采用上比较滞后；而唯品会的特点就是小、快、灵，运营和技术方面随着市场的高速发展快速调整。两年多以前，唯品会的主流系统全部采用 PHP，而现在我们绝大多数系统已经升级到自己研发的基础框架，比如服务化框架 OSP、日志监控 Mercury、配置中心等，这些新的框架已经适用到唯品会的核心系统中，效果非常好。

不能继续做黑匣架构

我非常荣幸能参与到公司的架构评审中去，在没有架构评审之前，我们好多项目甚至没有设计文档，开发人员甚至不清楚怎么做架构设计，开发人员只考虑自己的模块，注重快速上线，满足当前项目的短期需求，不考虑系统的集

成和将来扩展性，系统变成了黑匣子。

我们制定了架构评审流程，抽调各个部门的专家到评审委员会，优化了项目管理流程，把架构评审作为项目开发的一个必需环节。

对于是否做架构评审，我们通常有个筛选标准：看看项目是否对主流程产生影响，考虑到一些关键性的修改对项目的影响，我们有以下几个比较主要的关注点：

- 设计是否满足系统需求，包括功能、性能、兼容性、可靠性等；
- 涉及新技术基础组件的引入；
- 核心业务流程变动；
- 与核心业务系统交互变动；
- 与外部系统交互变动；
- 主要系统的边界划分；
- 是否符合公司制定的架构标准规范；
- 是否符合安全规范；
- 脱离实际情况的容量规划；
- 是否涉及系统重复建设问题。

架构的变动是针对长远的规划，为了更好支持业务，我们需要采取**渐进的发展策略**，不能为了一个理想的模型而牺牲整个系统的开发进程，我们只能在一定的时间范围以内做一些优化设计，使我们的系统尽量变得有序。

对于战略级项目而言，我们需要一个比较长远的思考，在需求阶段就介入架构设计，充分调研需求，了解上下文，在整个系统基础上统筹思考，要尽量考虑未来变化，使我们的系统有比较好的扩展性。

架构评审非常难，**难在如何平衡**，既不能影响业务系统的正常开发，又需要考虑架构的有序性，需要花大量时间，刚开始我自己花了很多时间来研究进来的个项目，我们每周都有大量的新项目出现，这些新项目在开始阶段缺乏相应的文档，给筛选工作带来很大的难度，这就需要我们架构师对系统非常了解。

至于筛选完成后，需要秘书长协助发出通知，安排评审日期，有时候还需要根据项目上线调整日程或者单独安排

评审。后来我们应用架构团队参与进来形成值班制度，减少了对专人的依赖，能够保证正常的运作。如今架构评审已经运转2年多，提前发现了大量的问题，预防了一些比较严重的事故发生。

Venus架构以及核心

Venus 是唯品会自主开发的一整套软件开发框架和体系，覆盖了软件的开发、测试、发布、运维四个阶段的一整套框架、中间件、平台的集合，主要由应用框架、服务框架、服务网关、服务安全中心、数据服务平台、配置中心、消息中心、定时任务平台、自动化测试平台、CI/CD 平台、监控平台等组成，着力于提升效率，降低技术难度，推进标准化，提升应用的性能和可用性，整个设计框架如图 1 所示。



图 1

简单谈谈 Venus 中的开放服务平台（OSP）和 Mercury 的功能，OSP 的主要目标是提供服务化的核心远程调用机制。

1. 契约化的服务接口保证系统间的解耦清晰、干净；
2. 基于Thrift的通信和协议层确保系统的高性能；
3. 服务可以自动注册并被发现，易于部署；
4. 配合配置中心，服务配置可以动态更新；
5. 客户端与治理逻辑的分离使服务接入得到极大简化。

除此之外，OSP 提供了丰富的服务

治理能力，如路由、负载均衡、服务保护和优雅降级等，通过 OSP，有效的实现了流量控制。OSP Proxy 具有软负载的作用，系统不需要硬件负载均衡，可以将服务请求均衡地分配到不同服务主机上。另外 OSP 还可以配置服务的路由，服务请求可以被分配到不同版本的服务中处理，这样很容易实现灰度发布（见图 2）。

Mercury 是唯品会开发的一款分布式日志跟踪系统，它主要分为 3 部分功能，调用链跟踪，监控报警和问题定位，可以给运维和开发人员定位提供信息。可以帮助理解系统行为、用于分析性能问题的工具，该系统让开发者可通过一

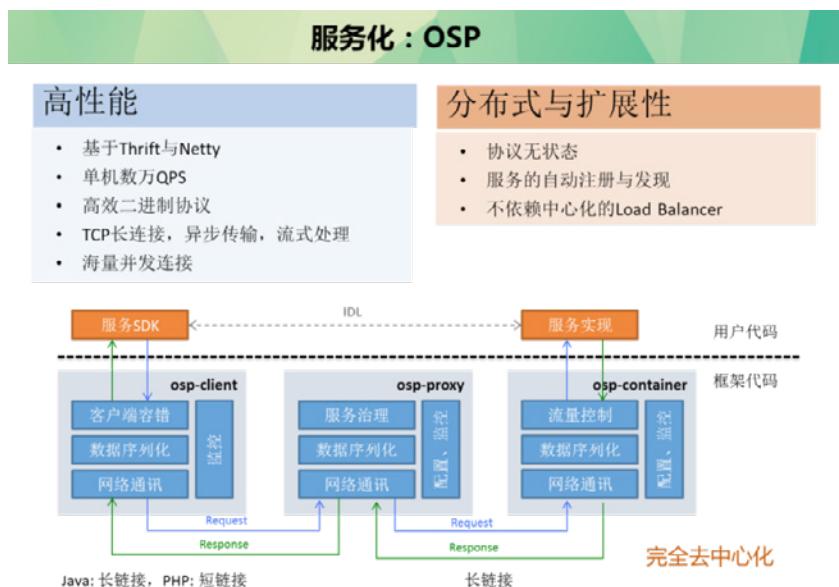


图2

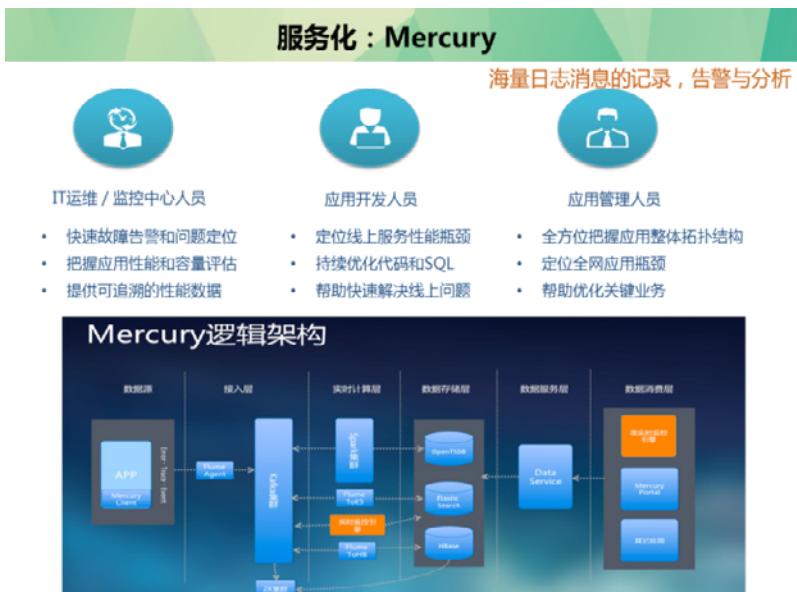


图 3

个 Web 前端轻松的收集和分析数据，例如用户每次请求服务的处理时间等，可方便的监测系统中存在的瓶颈（见图 3）。

开源 VS 自研

在公司开始研发这些基础框架之前，我们也做了大量调研，市场上的确存在一些开源框架，我们调研后发现这些框架存在一些兼容性问题，无法和现有唯品会系统无缝对接，总结一下难处。

- 如果要对接的话，还是需要大量工作。
- 开源方案存在以后持续发展问题，我们的基础架构本身就是一个体系工程，如何把这些独立存在的开源

框架集成在一起？

- 出现问题如何快速解决，是等开源下个版本，还是自己解决？

基于上述这些原因，我们决定自己研发，由于人力有限，我们先从服务化框架开始，研发 OSP，参考了业界比较成熟的方案，Thrift/Protocol buffers/Dubbo 等，最后方案采用 RPC 方式，基于 Thrift 的通信协议，软负载方式，通过服务自动注册和发现机制，引入本地和远程服务代理，通过长连接方式，大大提高了服务的连接速度。通过在我们系统中的实践，系统性能和可靠性得到大幅度提高，目前我们已经准备开源我们的服务框架。

有了统一的基础框架体系，通过代码生成工具生成大量代码，开发人员大部分工作只需要关注和业务系统相关的业务逻辑代码，开发效率得到了比较大的提高，通过OSP服务框架设计服务，数据层有效封装了数据访问，开发人员不需要关注分库分表、读写分离。OSP特有的本地和远程代理机制保证了服务请求的快速处理。

唯品会的革命性重构

之前唯品会的绝大部分系统基于PHP，PHP开发简单，但最大的问题就是性能低下，另外PHP也缺乏体系框架的优化支持，各个团队使用不同的版本，开发和维护成本都比较高，概述这阶段我们出现过的问题：

- 业务系统迫于业务高速发展压力，大部分改动都是为了应付短期需求，系统已经变得相当脆弱，不仅性能低下，浪费大量硬件资源。
- 业务流程混乱，系统间交互逻辑过于复杂，系统间耦合严重，系统边界不清，简单事情复杂化，导致大量的业务分支，无法管理，业务逻辑。

一个很小的改动就需要多个系统一起变更，系统集成困难，服务颗粒度过

大，复用性和扩展性比较差，数据库定义不严谨，数据冗余严重，大量系统间复制，导致系统间数据不一致性频繁发生。

系统维护效率低，运维成本高，故障率高。

那我们如何改造呢？

1. 我们的重构首先基于Venus技术体系，需要Venus高性能、高可靠的系统支持；
2. 针对业务系统的特点，对业务进行了深入分析，通过业务建模，根据业务对象的特有特征对系统做了重新拆分改造，重新系统处理业务的流程，简化系统间的交互，保证系统间的数据传递质量。定义业务的边界和交互方式，优化业务流程。以支撑未来系统为目标，支持更多的业务模式；
3. 服务分层，业务下沉，提高服务的复用性。针对不同层次不同场景不同粒度的访问需要，将服务划分为流程层、聚合层和基础层，层与层之间相互独立，又相互依赖，通过上层对下层的依赖和调用组成一个完整的系统；
4. 服务化解耦，采用OSP服务化体系，基于服务间松耦合、服务内部

- 紧耦合的原则，在考虑系统扩展性、不同模块不同职责基础上将系统划分为若干个独立的服务，服务独立部署。全面服务化，更好的支持服务治理，服务路由，服务降级和服务限流；
5. 系统间通讯增加异步处理，减少同步处理。对一些执行时间比较长的任务而又不需要得到立即响应的案例，通过异步消息机制，通知消息监听者，既提高了系统的响应速度，又改善了系统间的紧耦合情况。分布式异步消息队列服务器可在宕机后确保消息不丢失，从而提高系统可用性、健壮性和扩展性；
 6. 统一框架整合定时任务，时间敏感的通过异步消息机制；
 7. 优化数据访问，统一数据共享标准，明确各数据共享场景及规范，访问量大的数据库做读写分离，数据库有能力支撑时，尽量不要引入缓存，合理利用缓存提高访问性能，访问量和数据量都很大的数据库，通过数据库分库分表，不同业务域数据库做分区隔离，重要数据配置备库；
 8. 接入监控系统，通过系统监控和业务监控发现系统运行问题。

系统的改造力度比较大，重构团队面临巨大压力，通过日夜奋战，完成核心服务设计开发。但又面临新的问题：怎么确保新设计的系统没有问题，如何在不影响系统运行的情况下接入流量？

我们采取渐进的策略，从简单的模块入手逐步接入流量，验证没有问题的情况下，扩展到更多模块，通过这种方式可以尽早发现问题，及时纠正，把影响降到最低。在初期，发现单机 QPS 并没有想象的好，通过分析发现，我们的日志系统产生过多日志，占用了过多系统 IO 资源。有些正常的服务请求被误认为异常，比如返回空数。

闪购模式下如何解决超卖

在电商系统中，超卖问题几乎无法避免。

一方面是技术原因，在高并发情况下，用户涌入抢购，每次抢购请求都需要减库存，一般在减库存前，对库存数做一次查询再做减的动作，多线程情况下数据库无法保证数据库库存为 0 判断。如果贸然采取数据库行锁，性能必然引起比较大的下降。

在库存很宽裕的情况下，不存在超卖，当库存接近 0 的时候，容易出现超卖。所以我们的解决方案就是后台有一

个定时任务，当库存降低到一个阈值时，**根据当前 SKU 剩余库存和订单或者购物车占用的库存数量，重新纠正剩余库存**，在这种情况下，虽然无法防止超卖，但可以降低超卖的发生。

另外一方面超卖是管理流程原因，除了前端售卖，后端库存变化有很多因素，如仓储入库、盘盈、盘亏等环节，需要对库存变化环节流程梳理，加强系统间对账环节，改善系统间数据不一致性发生频率。

挑战大促

唯品会作为一个特卖电商系统，其闪购限时特卖业务特点决定了网站随时都需要处理高并发、大流量的用户请求。大量买家在每次新的品牌档期上线后，大量涌入，抢购商品，造成网站承担大量流量。尤其碰到热门商品，网站并发访问剧增，会造成整个网站负载过重，响应延迟，严重时甚至会出现服务宕机的情况。双 11 销量则是平时的 10 倍以上，将会涌入大量请求，导致资源紧张，我们必须对所需资源做出评估，寻找现有系统存在的瓶颈，并对各个系统作出方案：

- 首先根据引流情况和预计交易量评估各个业务线可能产生的流量。

- 对现有各个业务系统的能力做出正确评估，通过线上线下压测，确定单台应用服务器和数据库服务器的能力。
- 然后计算出所需的应用服务器、数据库服务器及其网络资源等，进行扩容。
- 重点模块进行预演，发现问题及时整改。
- 同时为了确保主营业务流程的正常运转，在流量过大或者出现异常情况下，可通过限流、降级等手段，通过自动熔断机制，关闭主流程以外的服务。
- 做好防刷预案，设置各种维度规则判断，剔除恶意请求。
- 另外监控系统非常重要，通过系统监控和业务监控两个层面，及时发现问题并作出对应措施。
- 为了保证大促前系统稳定，限制代码版本的更迭，保证线上环境的稳定性。

以我们的降级策略为例，电商系统为了保证用户体验，在资源有限的条件下，我们必须保证关键系统的稳定性。通过对不同业务级别定义不同的降级策略，对除核心主流程以外的功能，根据系统压力情况进行有策略的关闭，从而

达到服务降级的目的。

主流程系统包括用户注册、登陆、商品列表、商品浏览、购物车、结算、促销、支付、订单等，同样是商品浏览，我们需要保证在线商品信息始终可以访问，而对于下线的商品信息，我们可以容许在访问容量受限情况下，容许关闭某些页面的访问等；

另外对于由于服务降级导致的有损服务，通过异步补偿机制实现。

在分布式环境下保证数据一致性

一般情况下，我们根据应用的需要，将系统间一些公共数据独立出来，作为独立服务模块维护，系统可以调用独立服务去读写这个独立数据，这种情况下就避免了不一致性。例如电商系统中各个子系统都需要商品数据，我们有了独立的商品服务系统，通过访问商品服务，应用系统可通过访问商品服务获取商品信息。在有些场景下，为了业务需要，需要系统间数据冗余，在一个系统保留另外一个系统的数据进行相关性计算，如果这部分数据没有及时更新会导致两个系统中的冗余数据不一致，这种情况下，可根据数据相关度需要选择同步修改或者异步通知方式修改这部分数据。

举个例子，在支付系统中，涉及到

卡、币、券的扣减，涉及到多个系统的交互。在一笔交易中，如果出现其中一个步骤失败，就需要进入退款流程，回滚多个系统的数据。涉及金额运算实时性和准确性较高，接口都支持幂等，如果使用同步调用，将会出现性能问题。一个比较有效的解决方法就是采用异步消息通知机制，实现系统最终一致性。当出现某个环节失败，发送异步消息，每个相关系统收到消息通知后，做相关退款操作。并采用定时对账机制，保证各个系统之间数据最终一致性。

再一次回头看重构

首先我们架构设计源于业务驱动，我们在做设计之前必须充分了解业务，不了解业务的设计，即使再高明的技术，也没有任何价值，架构无从谈起。我们的设计必须支持现有业务，同时要考虑系统的开放性，易于扩展，同时系统的性能和高可靠性非常关键。最后一点非常重要：**架构必须简单，易于维护**，我们的系统不是一个个独立的艺术品，我们要考虑整个体系的运维，架构师要综合考虑开发、维护、运维成本。

大家都在谈重构，重构其实有不同纬度，有开发人员经常都在做的简单代码重构，有小团队内部做的模块重构，

架构重构则不同，是对业务模块进行重新整合基础上对系统做全面整改，会对关联系统有影响，所以要比较谨慎，多个模块一起参与，制定比较严密的计划，否则做不到真正的架构重构。

通过参与唯品会核心系统重构，感触良多，如对选购线、订单、结算、购物车、促销、支付等模块做整改，团队非常辛苦，深入到业务团队中做调研，整理业务流程，无数的设计会议讨论，设计评审，接入讨论和生产环境监控，修复紧急问题等等，每个环节都很重要。

作为架构师，是产品、开发、测试、运维等环节中重要枢纽，设计方案需要充分考虑各方面因素。在设计评审会议中，我们也发现了一些模块存在过度设计问题，导致系统过于复杂。在设计中，也面临一些艰难选择，我们的设计是为了未来的业务发展，未来很多情况不明朗，如果过于理想化，对现有业务会有比较大的冲击，也会影响实施进程，在业务压力的基础上，为此做了很多折衷方案。

在充分考虑未来系统发展基础上，大模块进行划分，模块内部则着眼于现实，先确定大系统间交互逻辑，未来模块内部可以进一步重构而不影响其它业务模块。在实施策略上，采取渐进的策

略，分阶段实施。

我相信只要我们架构师坚持自己的信念，策略得当，我们的重构一定会达到预期目标，我们会为将唯品会做大做强持续努力，也希望通过本次分享给大家带来一定收获。

更多“双十一”精选文章



腾讯云11.11：

[十分钟内完成弹性伸缩，流量清洗化解 DDoS 攻击](#)



京东11.11：

[大数据构建京东智慧物流系统](#)



京东11.11：

[人工智能在京东供应链的应用](#)



楚楚街11.11：

[特色移动电商的大促技术演进之路](#)



在微信关注 InfoQ



在新浪微博关注 InfoQ

关于 InfoQ

InfoQ 是一家全球性在线新闻 / 社区网站，创立于 2006 年，基于实践者驱动的社区模式建立，目前在全球拥有英、法、中、葡、日五种语言的站点。

软件正在改变世界。促进软件开发领域知识与创新的传播是我们的使命。每月有超过 200 万的技术人员访问 InfoQ 中文站。我们面向 5-8 年工作经验的研发团队领导者、CTO、架构师、项目经理、工程总监和高级软件开发者等中高端技术人群，提供中立的、由技术实践者主导的技术资讯及技术会议，搭建连接中国技术高端社区与国际主流技术社区的桥梁。

2016 年重点关注技术领域



产品

线上品牌专栏

架构、移动、开发语言、前端、容器、大数据、NLP、运维、云计算等。

迷你书

《架构师》、《云生态专刊》、《开源启示录》、《顶尖技术团队访谈录》、以及其他特刊及品牌迷你书。

新媒体

InfoQ 微信公众号、官方微博、垂直技术社群。

直 播



垂直技术公众号和社群

聊聊架构、移动开发前线、大数据杂谈、细说云计算、前端之巅、高效开发运维。

技术大会

QCon 全球软件开发大会
北京站 | 上海站

ArchSummit 全球架构师峰会
北京站 | 深圳站

GMTC 全球移动技术大会

CNUT 全球容器技术大会

ChinaTech Day 中国技术开放日

国内版：北京、上海、苏州、武汉、台北等

海外版：日本、美国、欧洲



InfoQ

国内最好的原创技术社区，一线互联网公司核心技术人员提供优质内容。订阅 InfoQ，看全球互联网技术最佳实践。做技术的不会没听过 QCon，不会不知道 InfoQ 吧？——冯大辉
从事技术工作，或有兴趣了解 IT 技术行业的朋友，都值得订阅。——曹政



关注「InfoQ」回复“大咖说”，看大咖聊职业发展、行业洞察



聊聊架构

以架构之“道”为基础，呈现更多的务实落地的架构内容。

关注「聊聊架构」
和百位架构师共聊架构



细说云计算

探讨云计算的一切，关注云平台架构、网络、存储与分发。这里有干货，也有闲聊。

关注「细说云计算」
回复“群分享”，
看云计算实践干货分享文章



大数据杂谈

专注大数据和机器学习，分享前沿技术，交流深度思考。

关注「大数据杂谈」
回复“微课堂”，看历届
高质量技术分享文章汇总



前端之巅

紧跟前端发展，共享一线技术，不断学习进步，攀登前端之巅。

关注「前端之巅」
回复“京东”，看京东
如何做网站前端监控



移动开发前线

关注移动开发领域最前沿和第一线开发技术，打造技术分享型社群。

关注「移动开发前线」
回复“群分享”，看移动
开发实践干货文章



高效开发运维

常规运维、亦或是崛起的DevOps，探讨如何IT交付实现价值。

关注「高效开发运维」
回复“DevOps”，四篇精品
文章领悟DevOps





架构师“双十一”特刊