阿里云开发者社区　阿里文娱技术

为业务量身打造
技术与AI助力增长

阿里文娱 用户 & 内容运营平台 技术实践

六大模块
用户触达 / 会员营销 / 数据工程
内容理解 / 内容管理 / 内容分发

藏经阁

阿里云开发者电子书系列

阿里巴巴文娱技术
钉钉交流群

阿里巴巴文娱技术公众号

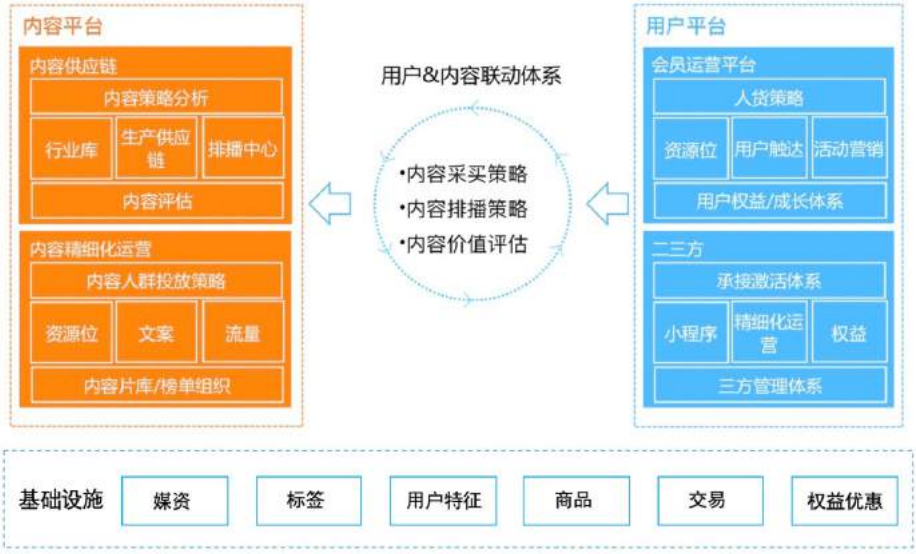
阿里云开发者“藏经阁”
海量免费电子书下载

序

作为一家由技术驱动的在线视频平台，优酷最重要的两大业务系列就是用户运营和内容运营，如何降本提效，用大数据和 AI 驱动用户增长和内容智能？我们挑选了最具代表性的场景和技术能力，分享给大家。

用户运营平台的目标是支撑优酷用户和会员增长。其中，会员增长的核心是：建设基于数据洞察的会员增长引擎，实现潜力付费用户挖掘、关键时刻付费转化提升、付费用户留存率提升，并结合用户心智制定价格策略，实现会员持续增长。用户增长的核心是：建设（阿里生态）二三方平台及统一触达平台，二三方平台主要与阿里巴巴经济体进行深度融合，通过业务和技术的合作，获取大量的优质新用户；统一触达平台通过可视化编排、实时任务调度及触达链路的打通，支撑优酷平台与用户统一触达和高频连接。

内容平台主要围绕内容全链路上的供给、运营、分发业务进行能力建设。其中，内容供给的核心是：建立数字化、智能化的评估机制和信息化制片管理机制，提升内容生产的质量和效率。同时从用户视角出发，建立贯穿版权、评估、运营、自制工作室等各团队的内容策略机制，实现供需匹配。内容分发和运营的核心是：建立内容综合价值体系，实现各品类从播前、播中的评级和资源配置拉通，并基于内容价值实现资源合理分配，提升整体内容的宣分发效率。



业务的快速发展离不开技术的底层支撑，而技术架构的发展始终围绕着降低开发门槛和提升开发效率这两点。本章围绕用户运营和内容运营两大平台，从领域驱动设计、FaaS 平台、Serverless、服务编排等技术入手，讲述产品技术团队在降本增效方向上做出的最佳实践和探索性尝试。

希望这些介绍能够帮助读者在技术领域拓展新思路，在自己的业务领域也能不拘一格，大胆创新。

阿里文娱优酷技术中心研究员　心石

2020 年 7 月 6 日

目录

用户运营平台

用户拉新与触达系统

优酷用户触达平台技术揭秘

作者 | 阿里文娱技术专家　曾辰

一、前言

优酷在用户运营场景上，沉淀了丰富的用户运营及营销触达策略。为了保障用户的持续活跃，当用户进入产品生命周期的后半段（预流失 / 流失状态）就需要精准洞察用户，在恰当的时机，通过正确的渠道，向用户推送个性化的内容达到召回、转化及促活的目的。但当我们深挖这些用户触达运营场景时，发现如下问题：

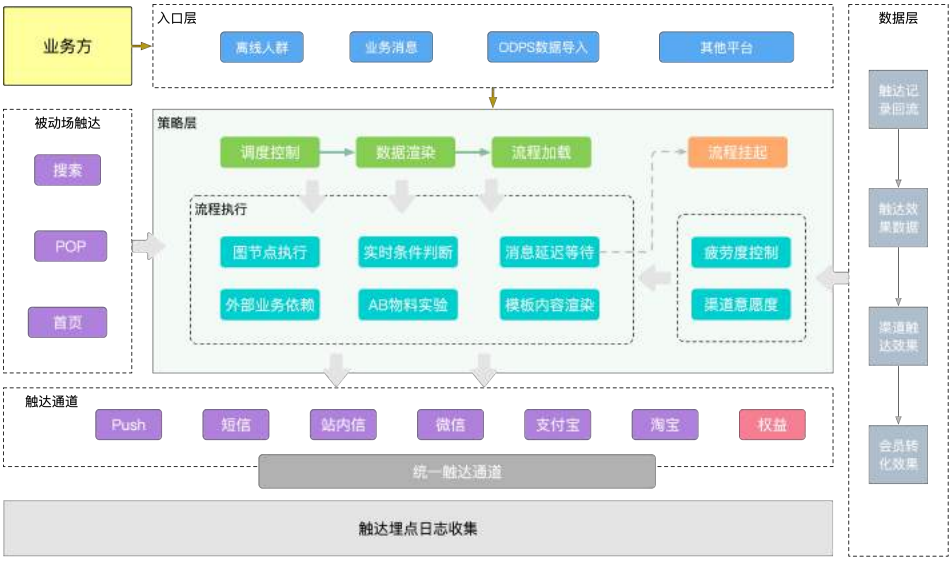
1）触达渠道建设成本高，PUSH、站内信、短信、微信公众号、支付宝生活号等触达场景接入各业务时，存在重复建设接入的问题；

2）过分关注单一触达方式在局部场景的应用，缺乏整体视角和统一机制，包括渠道间的联动及频次控制；

3）数据分析及沉淀能力薄弱，有单点数据，但缺少整体链路数据，同时触达策略无法复用等。

基于以上问题，我们打造了优酷统一触达平台，它是基于用户行为分析和洞察一站式触达平台，打通了优酷业务域内人群、素材、渠道、触达策略全链路，支持触达 AB 实验迭代能力，提供全链路效果数据。

触达平台目前整体架构大图及业务能力如下，除了支持主动触达矩阵外，还支持搜索、小喇叭等非标资源位的触达。除此之外，结合入口人群、策略编排及执行、数据回流构成了整个触达平台的链路。



二、平台架构实现

从平台架构大图中可以看到，架构核心主要包括入口层、策略层、通道层。人群及事件作为平台入口层，接入各类业务的能力，它控制了平台整体的起始调度执行；策略层是整个平台的中枢，提供触达策略的制定及执行；通道层整合了各个底层触达通道，它的核心是物料和数据源的能力。三层架构自上而下，共同组成了触达核心链路，下面针对各层的主要模块进行拆解。

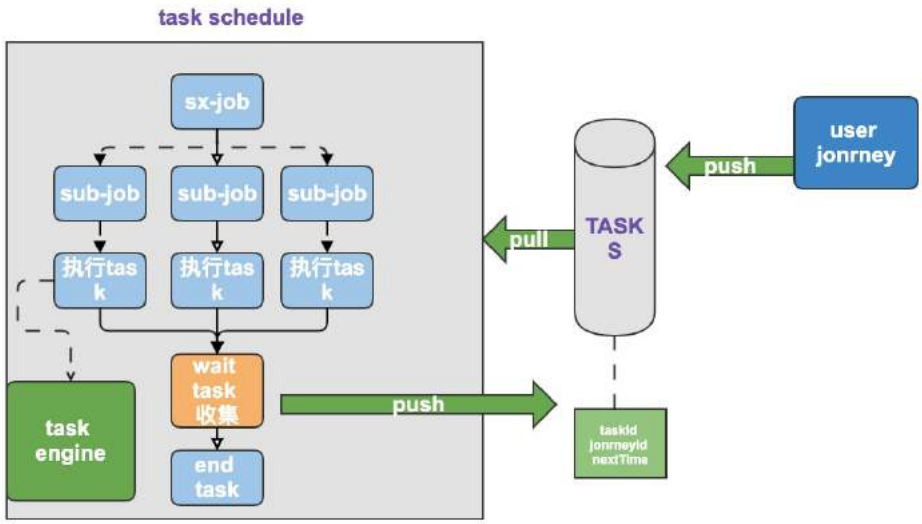
（一）人群及事件

统一触达平台作为触达业务的承接方，需要对接众多的业务触达需求，所以对平台的适配能力提出了很大的要求。而作为入口节点，承担了主要的职责。目前我们主要提供离线人群、业务事件适配，离线人群主要针对用户运营等场景，而事件主要用于实时系统通知及外部业务对接的触达能力。

离线人群

离线人群是通过人群标签圈选，通过数据接口导入人群数据，从而进行逐一的触

达。离线人群处理最主要的挑战，是如何保障人群发送效率，在尽可能短的时间内完成发送。基于这个目标，我们采用阿里集团分布式任务调度服务的分布式调度能力，基于大人群分片，多层下溯，充分利用分布式的能力。

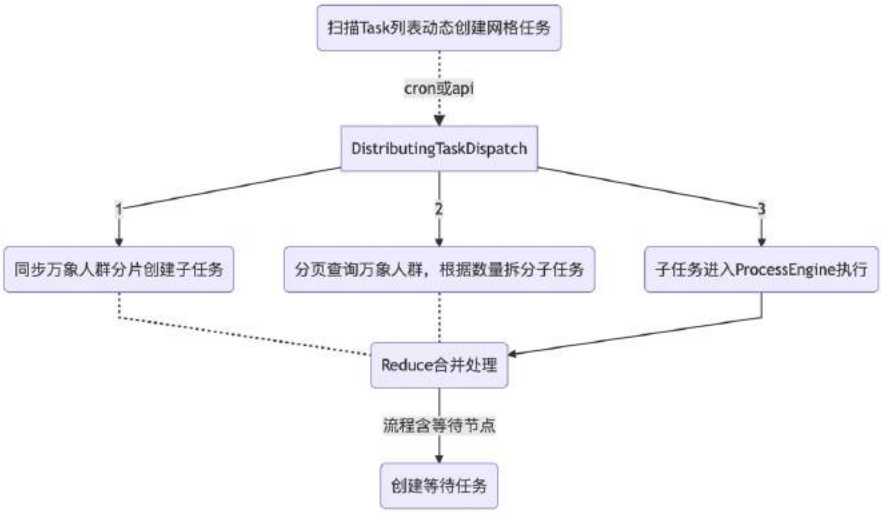


调度引擎也是基于任务调度的引擎。根据任务类型主要分为流程任务、等待任务；流程任务主要针对离线人群流程入口节点的调度，而等待任务是流程任务因在等待节点中断流程执行，在之后恢复执行的任务。目前支持多种调度策略，定时、立即发送，单次、周期任务，最终转化成调度表达式。流程任务的整体调度流程分 3 层：

1）获取人群类型、人群数量以及分页等基本信息；

2）分页获取人群用户列表，并完善用户账户信息；

3）根据配置细分成更小的人群分片，分片信息暂存 nosql。



人群任务及事件任务都支持流程等待，因为人群策略实现上略有差异，所以等待流程的实现也略有差异，但整体一致。任务暂存至 nosql 数据库，以任务需恢复时间时刻（分钟）为 key，等待任务的调度需要每分钟扫描以当前时刻为 key 的等待流程。

通过分布式的能力，能非常高效的完成人群的调度，但也因此引入了其他的问题，内存与线程等问题，最终，我们在发送效率及服务稳定性之间找到了一个平衡点。

1）人群规模不可控，经常会有上亿的人群场景，人群信息预加载内存后，短时间对内存产生了极大的开销。为了解决这个问题，我们引入了 nosql 进行了人群信息的暂存，在子任务执行时，再预加载人群数据；

2）人群任务触发时，执行引擎线程都是满负载运行；最初我们考虑运行效率，在触达渠道节点又引入了一层异步线程，但事与愿违，异步线程池的等待任务队列在短时间打满，并又占用大量内存，最终取消了异步发送逻辑。另一方面，除了渠道服务，触达的物料素材可能会依赖于外部数据源，在高并发服务调用下，对下流业务服务产生了极大的压力。为此，我们引入匀速器模式，针对各个业务提供的预设 qps 执行调用。

消息事件

消息事件的实时性、业务数据能力，正好是离线人群的补充。消息事件也是触达平台对外部业务提供的较为便捷的对接入口。同时，我们通过消息事件将会员触达系统、人群导入等内部的链路拉通。通过事件方式接入，消息的触达时机由业务侧控制，触达平台业务通过等待节点对消息发送时机做干预。

为了能支持各类业务方，也为了减少业务方的对接成本，触达平台对接入消息不做任何协议约束；完全依赖触达平台的事件适配能力。业务方只需要提供消息 topic 及 tag，以及消息体样例。为了解决消息的适配，触达平台的事件管理主要提供了以下能力：

1）消息过滤器，通过 MVEL 表达式提取目标消息；

2）目标字段映射，规范触达内部的消息格式；

3）占位符及参数表达式，消息数据加工。

通过以上 3 个能力，基本解决了消息统一适配的问题。通过 MVEL 表达式，可以灵活的支持各类业务需求的配置。总结下，消息事件的处理就是如下的流程：

上面提到，离线人群的缺点是无法携带业务内容，这对一些千人千面的触达需求不能很好的支持。而业务消息在一些营销场景下也不能很好的支持，基于此我们引入了 ODPS 数据导入的能力。它有离线人群的业务特性，又有事件消息的业务能力，而且技术实现上统一将导入转化成消息事件，使其与消息处理逻辑保持一致。目前导入是针对一些特有的业务场景，使用频次会较低，但其自由度较高。导入人群实现方案结合了离线人群以及事件人群的能力，能在满足业务诉求的同时能有较高的发送效率。

（二）策略执行引擎

策略即触达业务逻辑，作为统一触达平台，我们希望通过编排来替代原先需要硬编码的业务逻辑。触达平台通过一种有向无环图的形式，来形象的表述这种策略。当然上面讲的人群及事件、以及后续会着重解析的物料模块都是策略的一部分。但这一部分，主要来分析下整体的图形策略编排、以及策略执行引擎。

首先，需要明确节点、边的定义，通过对触达关键链路的拆解，明确了下如下节点和边的搭建规范。

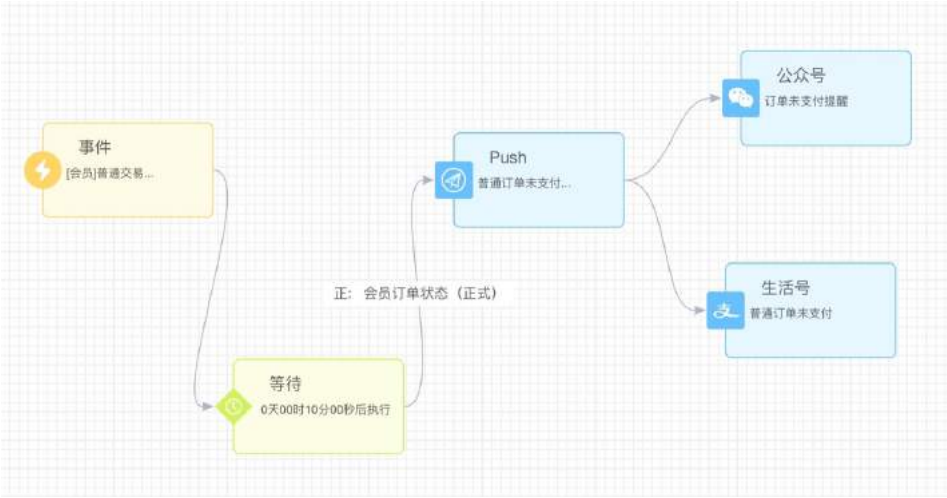
节点定义：

名称	类型
ENTRY_NODE	进入节点
ACTIVITY_NODE	渠道节点
FLOW_NODE	流程节点

边定义：

名称	类型
BLANK_EDGE	空边
CONDITION_EDGE	条件边

按照规范，我们产出了如下触达策略。实现层面，我们用 node，edge 两个列表进行存储，通过首尾相连，组成一个 Graph。节点以及条件边都有对应类型的控制器实现，通过解析后的 Graph，依次执行对应控制器。



在执行阶段，执行引擎根据策略编排规范，逐一执行各个节点。当然，前置数据预处理、调度逻辑已经在人群、事件入口已完成。执行引擎执行顺序如下：

1）接收人群分区任务、事件消息任务以及等待任务；

2）获取流程对应节点流程图；

3）执行上下文数据预处理，并进行执行引擎数据收集；

4）定位首个执行节点，等待任务需要从上一次终端节点下一个节点开始执行；

5）执行首个节点对应控制器；

6）节点执行结束，获取后续节点列表，并逐一执行；

7）节点中断或所有节点执行完成，更新相关计数，退出执行引擎。

执行引擎的职责比较明确，需要保证触达任务高效稳定的执行。主动触达场景对执行引擎的执行效率没那么大的要求，只是针对大人群的触达，因为耗时较久，目前主要的瓶颈还是下流服务。对执行引擎主要性能要求，来源于被动触达场景，因为面向 C 端，对 RT 及 RT 稳定性有很大的要求。这块将是后续执行引擎优化的重点，同时对任务间的资源隔离也是需要着重考虑的点。

（三）物料及数据源

触达平台的物料，泛指渠道节点依赖的素材。物料基于渠道模板配置，且支持事件、自定义数据源的占位符渲染能力。模板能力相对简单，但在物料模块实现上需要对模板进行对应的支持，所以模板的实现应该较为的通用，便于后续模板的扩展及改动。物料的渲染是对预设占位符的变量能力的替换，从而实现不同用户触达消息的个性化能力。目前我们已支持事件占位符、数据源占位符。

1）事件占位符只针对事件流程，可以对事件消息体内容进行逻辑加工处理，产出我们需要的占位符变量；

2）数据源占位符相对通用，不局限于流程类型，对定义的数据源结果进行二次加工产出需要的占位符变量。这个的数据源一般是一个外部的服务，需要预先确认服务，方法，以及入参出参的映射。

数据源难点及实现

物料渲染阶段，相对较为复杂的是数据源占位符的渲染，有以下难点：

1. 数据源入参不足

- 事件、导入人群，因人群本身就携带了业务参数，使用相对方便，一般数据源入参都能映射到人群消息变量上；离线人群本身只携带了用户 id，所以在很多业务场景使用数据源能力会受限，非常受到限制；
- 实现方案

a）任务配置环节，通过与其他平台的交互，引入对应的业务入参，业务参数在任务级别是静态的，通过数据源调用在实现消息个性化；

b）通过 ODPS 业务数据表导入，人群信息同时携带业务信息；借助 ODPS 的开发能力以及数据动态更新能力，能覆盖各类负载的触达需求。

2. 复杂业务配置受限

- 数据源能力目前仅支持在物料占位符上，一个数据源配置一个服务调用，对需要多个服务组合调用场景并不能很好的支持；目前需要通过硬编码的形式再产

出一个新的服务

- 实现方案

a）将数据源的能力扩展至前置节点中，通过串行编排，实现数据源链式执行的能力；

b）支持数据源之间的联动，通过上下文传递数据源占位符等信息，作为后续节点的输入。

3. 高并发场景

- 离线人群使用阿里分布式调度服务 MapReduce 模型，事件人群直接是对消息队列的消息消费。在流程执行中，对物料模块会产生极大的压力，数据源服务提供方的 RT 参差不齐
- 实现方案

a）基于入口节点的高并发执行场景，数据源不适合异步化，异步执行在高并发场景时会使数据源任务场景大量的堆积，最终会产生大量的内存压力；

b）权衡整体的平台负载及触达性能，我们采用了数据源的同步调用，针对多数据源场景进行并发调用以提高整体的执行效率。

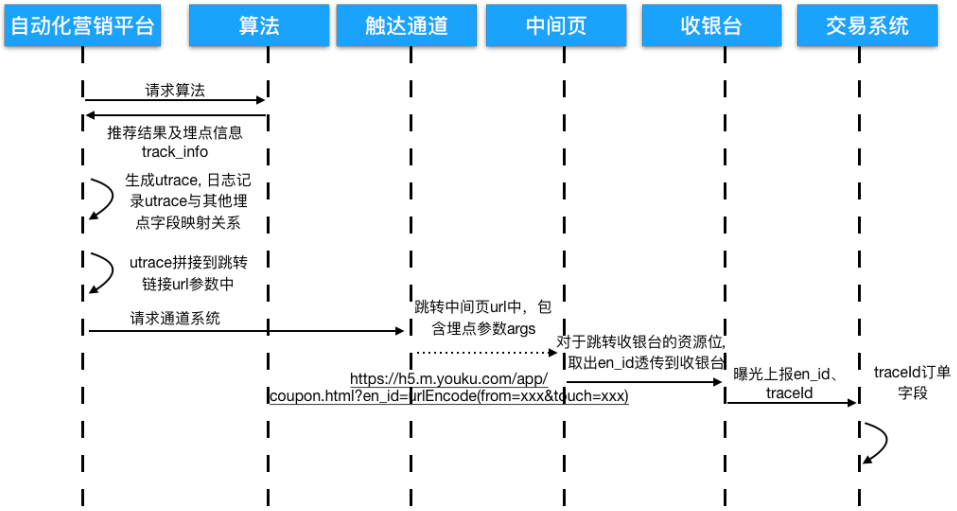
三、数据能力

基于触达平台化的背景，必须提供一套可复用的数据能力，来适配未来各类的业务的数据需求。在明确了触达转化链路上的曝光、点击、转化等各个指标项后，我们通过统一的埋点，拉通整体的数据回流链路，真正提供平台化的数据能力。

数据回流

触达平台定义 utrace 用于关联整个触达营销业务链路，一个 utrace 可以追溯到触达营销计划，触达渠道，内容素材，实验分桶，算法相关信息，以及其他统计维度的字段。在执行投放任务时，自动化营销平台基于上述字段生成 utrace。最终，通过 utrace，将渠道侧的曝光、点击数据，收银台的到访，下单进行统一串联，关联触达计划、实验分桶等。

中间页和支付埋点透传



四、总结及展望

目前，优酷统一触达平台能力基本完善，通过人群圈选、事件消息对接各类触达业务的能力；通过策略编排，结合实时条件边、数据源自定义等能力，能够承接复杂触达编排业务场景。在系统性能方面，目前平台承接了大量业务事件消息链路，执行引擎能在毫秒级别完成策略的执行，真正做到实时触达；同时每日执行的人群任务，能在小时级别内完成千万级别人群的触达，同时支持多任务并发执行，真正做到高效触达。在系统稳定方面，为了避免任务间相互影响，我们通过执行线程预分配进行隔离，保证各业务链路的稳定性。同时利用限流的匀速器模式，保证资源及外部数据源的平稳执行，减少流量尖刺等带来的系统异常。

未来，我们在技术层面对平台的建设，主要是两个方向：

第一，建设全域触达矩阵，拉通主动触达场景的编排能力。主动触达不同被动触

达，因为直接面向 C 端用户，需要有更高的系统稳定性，以及更短的响应时间，这对平台策略执行引擎提出了更大的要求

第二，支持状态跃迁的策略能力。需要引入状态机机制，对用户状态进行生命周期式的管理，这对平台的架构会是一个很大的挑战。

优酷 DSP 广告投放系统实践

作者 | 阿里文娱技术专家　鸿雁

随着 RTB 网络在线展现广告交易模式的兴起，各大公司都纷纷搭建自己的 DSP（Demand-Side Platform）广告投放系统进行获客。优酷在近几年，也搭建了 DSP 系统，并且在持续迭代。在这一过程中，经历哪些技术探索？趟过哪些坑？有怎样的技术方案沉淀？下面我将从技术视角一一分享出来，希望对大家有启发。

一、业务目标

DSP 的核心目标就是用户增长，通过广告拉新和召回，提升用户留存和活跃度。从月活跃度的维度，我们把用户进行分层：新用户、低活跃用户、中活跃用户、高活跃用户和流失用户。

具体活动度的指标，可以根据业务的具体情况进行定义。人群划分好之后，就可以针对各分层人群进行召回承接。我们的重点是要召回当月新用户、低活用户、中活用户、流失用户。

高活用户：即使不采取召回，也会主动来；流失用户 (30 天没来过的用户) ：召回难度最大。因为有可能用户已经卸载或者转移到其他 APP。从数据上看，流失用户的转化 ROI 也是最低的。所以我们的召回策略就是要把召回阶段前置，在低活和中活这两个阶段就开始进行召回干预。因为这个阶段用户获取成本相对较低，ROI 比较高，让这部分用户不断触达召回，提升留存，把他们往高活上去迁移 (当然这部的用户体验也要跟的上，让用户形成使用习惯和心智) 。

二、系统架构

明确了业务目标，接下来介绍系统的架构，直接上图：

下面从渠道能力、策略能力、算法能力、实时监测，数据等几个维度来介绍下 DSP 系统的实践与思考。

三、渠道能力

召回获客首先要有强大的渠道能力做支撑，这样才能获取和覆盖到足够多的我们想要的用户，本文只涉及到支持 RTB 竞价广告相关的渠道，市面上有很多 ADX，比如百度 ADX、网易、广点通、tanx 等等，那么这么多渠道，我们也不可能全部接入，流量压力也是可想而知的。

首先要思考和评估我们要重点接入哪些渠道，哪些渠道的获客体量大，哪些渠道的成本低，每个渠道的用户分层是怎样的，我们需要对渠道有一个渠道质量的洞察。

可以从以下几个指标进行洞察：

媒体信息	下发流量uv	优酷覆盖uv	多媒体重合度	可触达占比	点击率	转化率

需要分析：

1）媒体下发流量中有多少是安装优酷的用户，用户分层占比情况；

2）多个媒体之间的用户重合度，避免一个用户在多个媒体的重复曝光，可做跨媒体的曝光频次控制；

3）每日实际最多可触达多少优酷用户，并分析是优酷用户但是没有被触达的原因是什么，是什么环节漏掉了这个用户的触达；

4）媒体的点击率如何，转化率（对应优酷的换端率、播放率）如何，预算分配等等，然后做出最佳的媒体组合进行投放，最大化 ROI；

5）另外提下网盟渠道的优化，比如广点通。

广点通渠道我们主要对接的是网盟，广点通网盟可能包含上万家的 APP 媒体资源位，每个媒体的效果和表现都不一样。这给广告主带来的优化难度很大，而且广点通的流量下发应该是根据（竞价成功 pv/ 流量下发 pv）的占比来下发的，比例越大，流量下发的越多，这就带来一个问题，如果要获取精准用户，势必参与竞价的 pv 会减少，占比降低，最终导致流量下发降低，覆盖不到更多的用户，需要进行权衡。

四、策略能力

在 DSP 广告的投放过程中会涉及到很多的投放策略，下面介绍几个比较通用的投放策略。

1）频次控制

这里说的频次控制包括：参与竞价频次控制、展现频次控制、点击频次控制、换端频次控制等几个维度。

一般说频次控制主要是说展现频次控制，优酷又增加了三个维度的频次控制。

参与竞价频次控制：这个是分媒体的，在有些媒体效果会很好，有些媒体则不需要，由于有些媒体参与竞价和展现之间的延迟时间比较长，导致一直没有展现，这个时候展现控制就失效了。

用户可以一直参与竞价，然后集中或者分散开去展现，这种情况展现控制就失效了，有可能一个用户要展现上百次都有可能。

点击频次控制：对于当天点击过广告的用户，则不在购买，可以防止疲劳。

换端控制：对于当天唤起过优酷的用户，不在继续购买，由于用户主动或者被动已经来过优酷了，就没必要在进行购买了。

下面举例说明下不同媒体频次控制的次数，不同媒体的效果差的也比较大，尤其是网盟渠道会和头部媒体渠道差别比较大

2）素材赛马策略

在投放素材的时候，一个广告计划可能会配置多套素材进行投放，如果没有赛马机制的话就会随机投放。

赛马机制主要解决的是把点击率高的素材给到尽可能多的曝光，点击率低的素材减少曝光或者不曝光。

这个非常类似推荐系统里面的两个经典问题：EE 问题（从探索到应用）和冷启动问题。

下面介绍一个简单的赛马逻辑实现，具体策略如下：

假定投放方案 A，下面有广告位 M，符合 M 条件的素材 X 套（X>=2）

素材 CTR = 素材点击数 / 该素材展示数

冷启动过程：

第一：在 15min 时间范围内，广告位 M 符合投放方案 A 流量对于 X 套素材流量平均分配。

第二：计算上个 15min 内 X 套素材的 CTR 结果 [CTR1,CTR2,…,CTRx]，并进行排序，假定 CTR 最高的为素材 N。

第三：在下一个投放范围内（15min）范围内，对于广告位 M 符合投放方案 A 竞价返回素材时：

a）对于素材 N 分配 80% 的流量

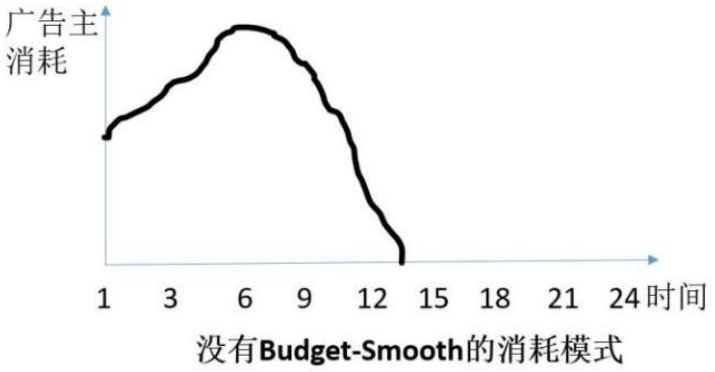
b）对于剩下的 X−1 套素材，平均分配剩余 20% 流量

第四：投放中重复步骤 2 和步骤 3 的判断。

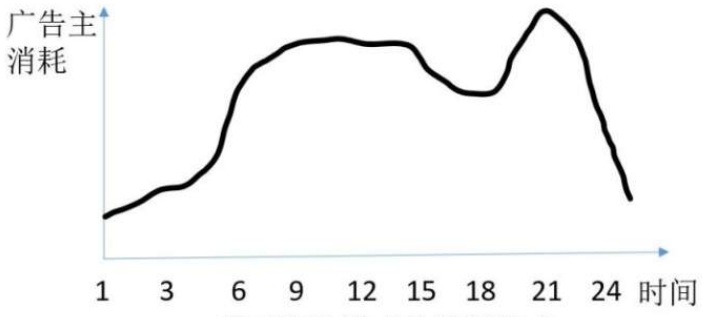
3）平滑投放（budget smooth）

平滑投放是保证广告主在预算一定的情况下，能平滑的消耗预算，而不是在开始几分钟之内全部花完。

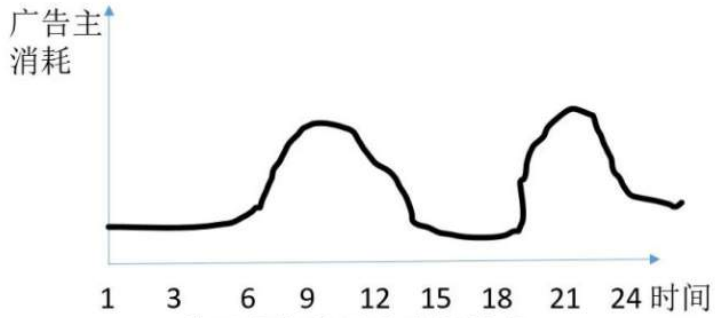
平滑投放可以让广告主触达更多的人，更多时间段的用户群，下面是几种消耗模式，借网络上的几张图：



没有Budget-Smooth的消耗模式



基于流量模式的消耗模式



基于优化效果的消耗模式

之前我们采用的是基于流量模式的平滑投放，会有一个问题就是流量大的时候分配更多的预算，但是转化 ROI 不一定好，所以目前我们采用的是第三种方式，基于效果的平滑投放，这里的效果主要指的是点击 CTR 效果，比如中午大家午休，刷手机比较多，效果比较好，那我们就多分配预算在中午这段时间进行投放。

4）广告重定向策略（retargeting）

“重定向”是广告领域的一个术语，又被称作“访客找回广告”，“在营销广告”，“回头客广告”等等，是帮助广告主有效获取目标受众的一种重要方式，比如在消费者购物的过程中，常常会因为这样或那样的原因，最终没有完成转化。重定向广告将商品展示到此类未完成转化的消费者面前，将其带回到相关网页。由于展示的商品往往也是消费者最想要的，所以重定向广告的转化率比一般广告要更高。

最近X-X天在优酷看过剧(第N集）（时间由新到旧筛选）	视频观看时长占比在(0,10%]	返回当前集（第N）视频信息
	视频观看时长占比在(10%,65%]	返回当前集（第N+1）视频信息
	视频观看时长占比在(65%,100%]	按照观看时间顺序寻找下个剧集内容

5）图片视觉优化策略

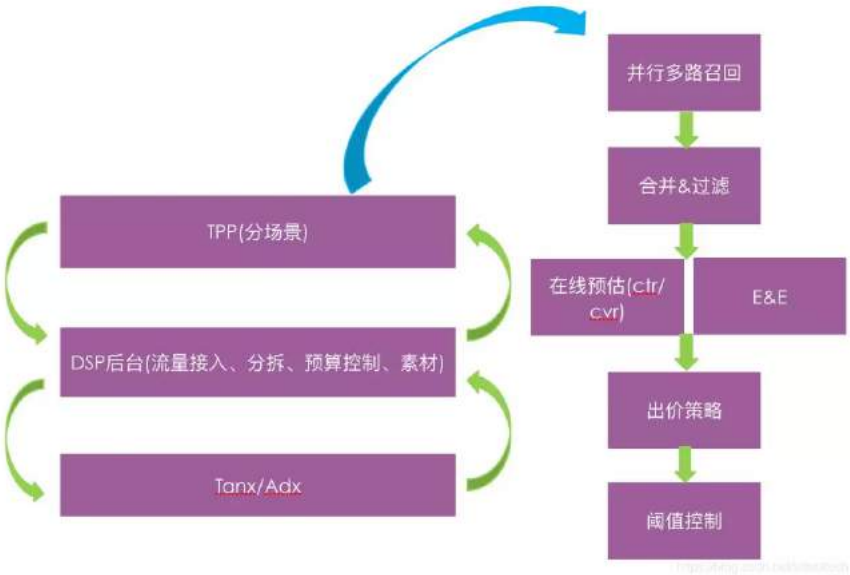
为了给用户更好的体验以及更好的视觉冲击力，提升用户的点击欲望，我们对投放的图片素材进行了二次加工，取得了不错的效果。

测试中发现不同渠道的视觉不同，不能统一设计，所以我们进行了分渠道对比视觉设计实验，比如在某 feed 流中：

通过工程合图能力，我们快速实现并上线了几十套模板，线上进行 AB 实验，测试点击率效果，选择优秀的模板长期投放。

五、算法能力

接入算法能力主要为了实现个性化的站外广告推荐，目前接入算的架构：



1) 并行多路召回

基于播放记录召回：会根据用户的站内播放行为进行召回，按时间进行衰减；

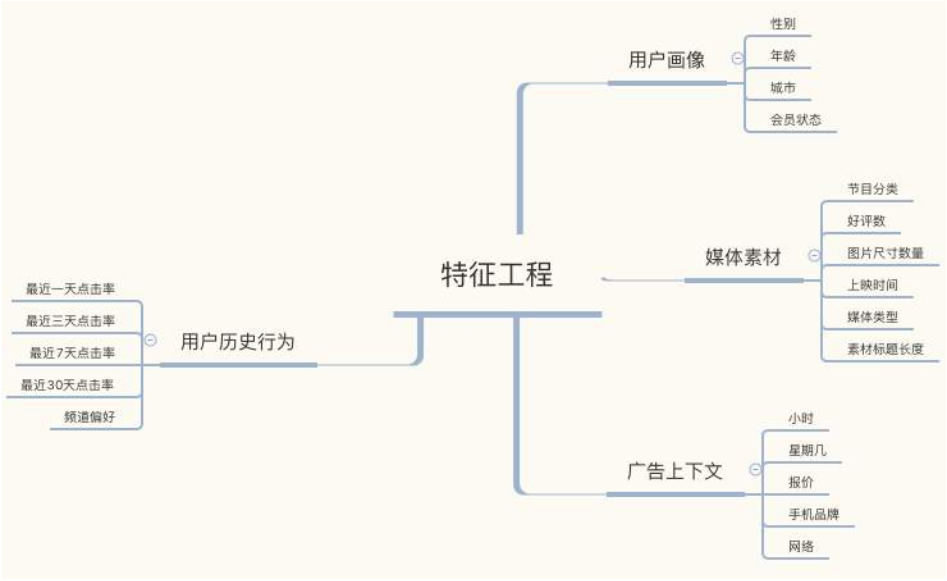
基于热点节目召回：比如最近新上的新热剧等；

基于用户偏好召回：比如明星偏好，剧集偏好等；

基于站内行为召回：比如收藏行为，点击浏览行为，搜索行为等。

2）分渠道模型

由于我们接入了很多渠道，每个渠道的特征表现都不太一样，需要分渠道进行针对性的模型训练。简单的特征如下：



六、数据 & 监控能力

完善的数据分析系统对 DSP 的投放非常重要，通过数据分析，优化转化漏斗，降低成本。

（一）转化漏斗

首先需要定义业务的转化漏斗，见下图：

下面逐步说下这几个漏斗阶段我们可以干预的策略。

1）参与竞价

参与竞价的量，主要受限于用户的识别能力，如果根据设备信息可以准确识别到是优酷的用户，那么参与竞价的量可以大大提升。

2) 竞价成功

竞价成功的量，主要有这几个方面的受限，第一个是出价，出价过低导致竞价失败率高，第二个是接口性能，目前 ADX 要求是 100ms 内返回，这就要求 dsp 需要不断的优化接口的响应时间，可以很明显的提升竞价成功的量。

3）展现

展现的量，比如标题长度，曝光的次数等等。另一个是媒体自身的原因了，可能媒体本身有些复杂的频控逻辑导致展现不成功。

4）点击

这个第一是和算法的推荐能力的准确性有关系，推荐的越准确越容易点击，另一个是和图片和标题的吸引程度以及视觉体验有关系，比如一个猎奇的短视频，可以大大激发用户的点击欲望，还有就是刚才说的视觉图片合图的优化；

5）唤起

换端率，主要受限于识别用户是否安装了优酷 APP，用户是否是误点击，或者不像打开观看，时机不对？或者还有个原因就是系统启动太慢，有广告，提前退出了；

6）播放

播放率，主要受限于播放页的性能和前贴广告，如果前贴广告太长，用户等不及就退出了，没有产生有效的播放，再有就是网络状态 4G 或 Wi-Fi 等等；

7）次日留存

主要和产品的用户体验有关系了。

（二）全方位监控

广告系统的交互各个环节都需要进行详细的监控，才能做到有问题及时发现。

比如突然的消耗上升或者降低，点击率的下降，参与竞价量的下降等等，都需要进行详细的监控，避免资损的发生。目前建立的监控体系指标有如下：

维度	分钟级监控项
广告计划	参与竞价、竞价成功、点击、展现、唤起、消耗 …
媒体	参与竞价、竞价成功、点击、展现、唤起、消耗 …
大盘	总花费、竞价总pv、竞价成功总pv、曝光pv、点击pv、换端uv、换端成本，cpm …
其他	错误码、流量下发总量、rt区间分布、状态码..

七、总结

最后在总结下整个 dsp 的建设过程要考虑的点：

1）渠道建设，流量接入；

2）投放策略沉淀；

3）算法模型发挥推荐优势；

4）完善的监控体系。

当然本篇文章由于篇幅有限，还没有涉及到 DSP 的另一块核心，流量价值预估和出价算法，这块后续会有单独的篇幅去介绍，优酷的这套体系上线后在拉新和召回上发挥了重要的作用，为用户增长打下了坚实的基础。我们还在不断的建设和完善中，非常欢迎计算广告领域的专家大牛加入。

会员运营系统

优酷会员观影特权 – 技术保障之路

作者 | 阿里文娱技术专家　不晚

一、背景介绍

在互联网高速发展的大环境下，各大视频网站的付费会员数量一直在高速增长。增长的原因即得益于用户付费意识和观看习惯，也与网站的内容深耕、观看体验、会员生态体系建设息息相关。调查结果显示，优质的内容依然是网络视频用户付费的第一驱动因素，第二驱动因素则是观看体验，包括但不限于：起播速度、起播成功率、播放流畅度、1080P 清晰度、免广告、内容抢先看等。因此各大视频网站之间除了持续依靠内容抢占市场，各自也在深耕细作，打造全方位的优质观看体验留存用户。

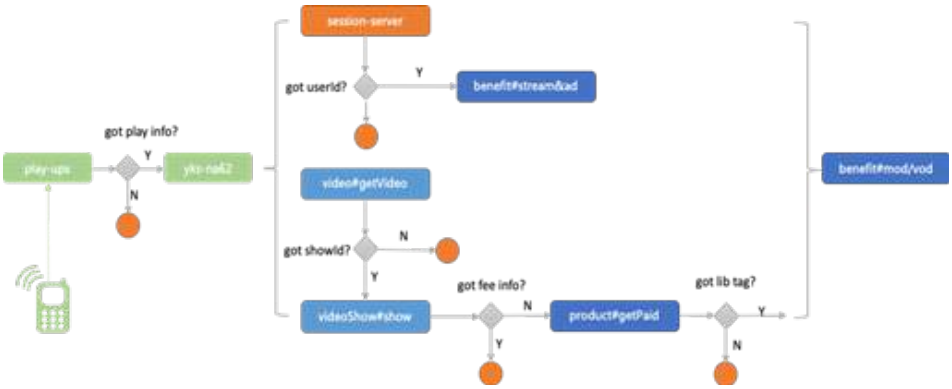
在优酷付费会员体系中，优质的观看体验同样非常重要，尤其是“1080P 清晰度”、“免广告”、“内容抢先看”这些付费会员专属的观影特权体验；因此围绕付费会员观影特权体验，下面的内容笔者将逐一展开，给大家分享极致权益体验的技术保障实践。

二、极致权益体验的技术挑战

在话题展开之前，我们需要知道，体验其实是用户在“使用”产品或服务过程中的全部感受，包括用户的情感、喜好、认知印象等。极致的体验，就是让用户的这些感受处于最满意状态。在网络视频观看的场景中，用户的操作路径一般依次为：进入播放的权益验证（仅试看或广告等待或直接观看）、可能发生的购买行为、观看中的体验与感受、观看后的内容评价等。对于付费会员来讲，最在意的其实是进入播放的权

益验证结果，亦或者是购买行为发生之后的权益到账情况。极致的观影特权体验，就是要保证每次权益验证的稳定性、准确性、快速性；每次权益发放的正确性、及时性等。下面将重点分析权益验证的场景，后面统称为验权场景。

在播放场景中，所有用户点击影片，起播阶段都会执行验权，这个阶段请求到服务端的流量每秒大概有几十万，并且为了保证起播的速度和成功率，所有请求必须是在秒级内响应给客户端的。一旦出现响应时间过长或者请求无法响应的情况，都将给用户带来非常糟糕播放体验；尤其是付费会员的观影特权体验，可能会导致付费会员无法跳广告、无法切换 1080P 清晰度、观看不了 VIP 专享的影片等等。另外，从起播验权的关键链路节点图中（图一）可以看到，验权链路有八个关键节点，每个节点的稳定性、响应时间、计算结果都会影响付费会员的特权体验。如图中的 SessionServer 节点，若起播请求携带的登录 token 在当前节点换取登录用户 ID 超时或换取失败，那后面的验权流程就会直接截断，对正常已经是付费会员的用户会判断为没有权益，从而出现只能试看影片或等待播广告的情况。由此可见，面对高并发的流量请求、快速的响应时间、复杂的服务链路，如果要把权益体验做到极致，链路的响应时间、稳定性、准确率必须是最高要求，技术上更是面临巨大的挑战。



图一：影片起播验权（关键节点路径）

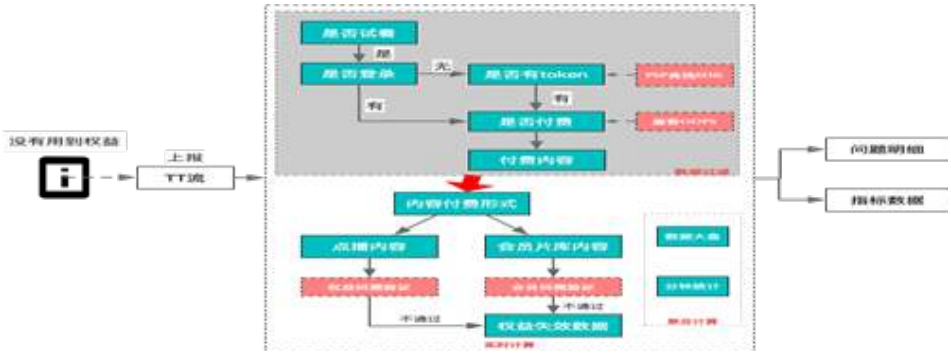
三、极致权益体验的技术保障

前面提到了播放验权场景的权益体验技术挑战，主要是链路的响应时间、稳定性和准确率的要求。下面将从技术方案探索到执行落地展开诉述。

我们做任何事情都会有一个起因，而“优化”的起因是需要进行监控、调查或对比的。在播放验权的场景中，首先要做的是准确找到问题指标，然后利用指标进行多维度分析，最后找到问题点，持续改进优化。

1. 问题指标

针对付费会员的观影特权体验，最合适的指标应该是权益失效受影响的人数和次数。要获得这个指标的准确数字，首先第一步是从数字上要定义清楚怎么样算权益失效，最准确的场景应该是确认用户在问题时候有权益但没有享受到算一次权益失效。按照这个思路，通过收集播放端所有未能使用到权益的上报数据，权益端提供回溯验权的能力，利用 Blink 平台的实时计算能力进行回溯验权，确认用户的真实权益情况，生成权益失效数据；计算方案详见图二。最后对已确认的问题数据，通过区分场景、设计指标，聚合计算出各个维度的权益失效指标数据。



图二：权益失效指标（计算方案）

2. 改进落地

在已经有了权益失效指标数据的前提下，通过逐步深入，持续分析，最终发现影响权益失效的因素有两个：

因素一：登录验签失败率。

通过细化分析，定位出登录验签失败率高的主要原因有两个：接口超时严重和签名验证错误率高。接口超时严重是因为登录签名验证涉及非常复杂的加密解密计算，并且这计算全部在账号服务端进行，几十万高并发流量下，账号服务端的计算资源急剧消耗，从而导致接口性能降低，响应严重超时。解决这个问题的核心思路就是去中心化，将复杂计算对资源的消耗从账号服务端剥离出来。账号服务端提供 Client 计算资源包，接口调用方在本地节点进行验签计算，Client 内部远程接口的调用只是简单查询，大大降低了账号服务端的计算压力，解决了接口超时的问题。登录签名验证错误率高的原因主要集中在 Token 过期和 Token 无效两个维度，通过优化 token 安全时间范围、发布 SDK 统一管理签名逻辑，签名验证错误高的问题直接降了一大半。优化效果见图三。

图三：登录验签成功率

因素二：链路关键节点接口成功率。主要是三个关键链路节点，权益节点、播放节点、商品节点。

1）权益节点

权益节点承载着所有用户的权益数据，用户能不能跳广告、付费内容是否可看，这些都依赖权益节点验权结果的准确性。每秒几十万的验权流量请求，要求结果控制在毫秒内返回，这就必须保证接口非常稳定，接口响应时间轻微抖动都会导致验权失败。在权益节点中，验权接口准确率一直保持在 99.99% 以上，验权结果准确性是没有问题的，主要是接口抖动超时问题。定位出的主要原因是集群机器频繁的超时 YGC。经过各种猜测、分析、排除、验证后才定位出根源所在，原因是配置中心往集群机器同步数据，由于每次同步内容过大（已超过 10M），导致集群机器每次接收到同步数据都会反序列化出一个超大对象。这个超大对象存活寿命非常短，在 JVM 新生代里就被回收了，由于回收的对象较大从而导致 YGC 时间过长，进而影响了验权接口性能，导致接口响应时间出现抖动。

找到了 YGC 时间长的根源，问题解决就变成了如何避免超大对象的生成。通过优化配置中心同步数据方案，将数据同步拆分成全量同步和增量同步。增量同步不会生成大对象，对 YGC 完全没有影响；只有全量同步会反序列化出一个超大对象，但这个只在应用启动时执行一次。另外，全量同步的数据会一直常驻 JVM 内存作为缓存数据使用，也不会被 YGC 回收。问题按照这个思路落地解决，验权接口抖动超时的问题基本没有了，接口成功率也维持在了 99.99% 以上。

2）播放节点

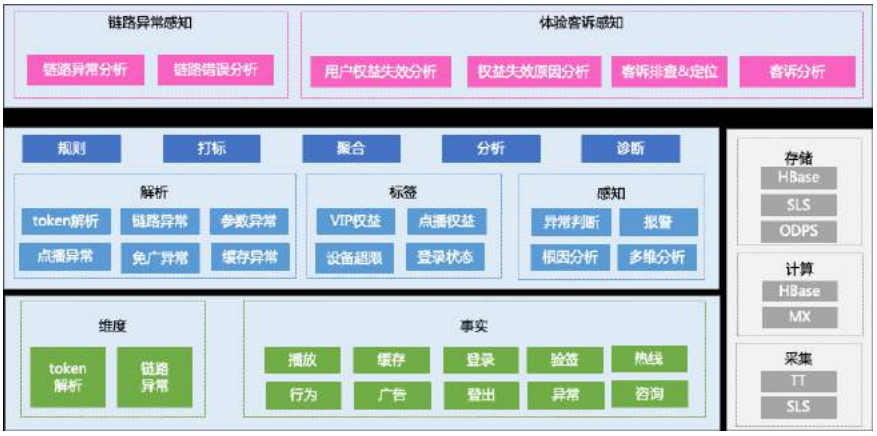
播放节点负责串联各个中间节点的处理结果，任何一个中间节点出现接口超时或接口不可用，都会影响播放节点的性能，甚至是拖垮整个播放节点。因此针对播放节点的稳定性保障，主要是内部多维度的性能优化，包括梳理所有外部依赖，下线了一些不必要的接口调用；解决发布抖动问题，增加启动预热；优化验权接口调用逻辑，如果验权接口超时未返回结果，直接默认验权通过，宁可给没权益的用户一次免费使用权益的机会，也不能让有权益的用户出现一次权益失效。

3）商品节点

商品节点负责所有内容的付费属性管理，并且需要将这些付费信息准确同步给媒资节点。如果商品节点和媒资节点的数据同步出现不一致，就可能导致付费内容免费看的情况。另外商品节点提供的内容付费信息查询接口，也必须保证接口的稳定性，如果出现接口超时，就会获取不到内容的付费属性，可能导致已购买内容无法观看的情况。因此正对商品节点，主要是提高接口稳定性和数据准确率，具备包括增加缓存一致性巡检和补偿任务，提高数据服务的准确性；原 HTTP 接口有签名验证，耗时比较长，提供 HSF 接口替换原有 HTTP 接口；原 HTTP 接口没有同机房策略，切换 HSF 接口开启同机房策略，决绝跨机房接口调用响应时间长的问题。

3. 自动化跟进

经过全方位的人工排查定位、改进优化落地，播放链路所有节点接口成功率已提升到了 99.99%，同时增加业务兜底逻辑，权益失效人数指标对比下降了 99.9%。从全量用户来看，这个指标对应的权益体验指数已经相当不错了；但各种偶然突发性的问题还是会影响用户的权益体验，因此通过搭建了一套实时问题自动分析计算体系（图四），将用户在线发生的问题以小蜜机器人的方式实时反馈，能够给用户精准的问题解答，从而给用户最好的权益体验感知。



图四：实时问题自动分析（计算框架）

四、总结

播放验权场景的观影特权体验只是优酷会员极致权益体验很小的一部分，后面还有缓存场景、直播场景、投屏场景等等。每个场景都有各自复杂的技术链路和业务逻辑，优化的方法也不是固定的流程，需要具体问题具体分析。后面笔者将继续极致权益体验的技术探索之路，用技术的手段来不断提升付费会员的专属权益体验。

优酷会员营销系统技术实践

作者 | 阿里文娱高级开发工程师　臻龙

一、背景介绍

随着在线视频行业数十年的发展，各家的会员业务，尤其是会员规模都已进入成熟期，呈现饱和状态。会员营销，不论是针对现有市场的精细化营销，还是针对下沉市场的定向营销，都需要更加针对性的打法。在这复杂需求背后，是强大营销系统的支撑。下面，我将分享优酷营销系统的技术实践经验。

二、面临的挑战

1. 场景复杂多变

会员营销的场景复杂多变，具体归纳总结是四类：折扣，买赠，领取，发放，支撑起优酷站内，（阿里生态）二三方，优酷 OTT 等多个业务方的营销需求。

营销场景一：折扣类。在商品维度，可以是简单的普通活动的折扣，也可以是可选优惠券与普通普通折扣，支付渠道与普通活动折扣的叠加折扣。订单维度，可以是单个订单的折扣优惠，也可以是多个子订单的组合折扣优惠；

营销场景二：买赠类。赠送的权益，有虚拟的会员权益，也有阿里生态二三方合作方的联合会员权益，比如 88VIP，当然也可以是实物类的商品；

营销场景三：领取类。不同的业务需求方，会根据自身业务诉求和业务目标，配置不同营销领取能力的营销策略。因业务而异，因目标而变换；

营销场景四：发放类。符合业务逻辑，主动给用户发放一定权益。业务逻辑也呈现多样化，比如预约影片，可以给用户发放观影体验特权；观影过程中，满足观影时

长，发放用户成长值等等。

从以上四类场景出发，每年每季度都会出现新的玩法创新。比如，折扣的叠加，折扣买赠类的叠加、多个折扣能力互斥、折扣并领取的组合式营销等等。反馈到后端的营销系统上，不能一味的定制化开发，而是需要提供给运营同学，灵活的编排优惠的能力，并且实现复杂业务需求组合，快速拿到实验结果数据，帮助运营及时调整策略，最终实现业务价值的最大化。

2. 系统要求高可用，低延时

除了单独的系统能力，营销系统还担负着与交易系统核心节点的优惠实现的使命，高可用，低延时的系统接口必不可少。如果优惠透出与下单场景的系统接口性能差，优惠透出问题凸显，会影响用户的下单意愿和交易的订单与支付；如果接口耗时大，会带来交易渲染与下单的价格不一致，用户无法得到看到的优惠，引发严重的客诉行为，更甚会造成很大的法务风险。

业务的特殊性，要求营销系统业务核心接口，必须高可用，高性能，低延时，稳定性才能保证用户良好的营销体验。

三、优酷会员营销系统的架构实现

1. 活动规则化，QL 语言实现

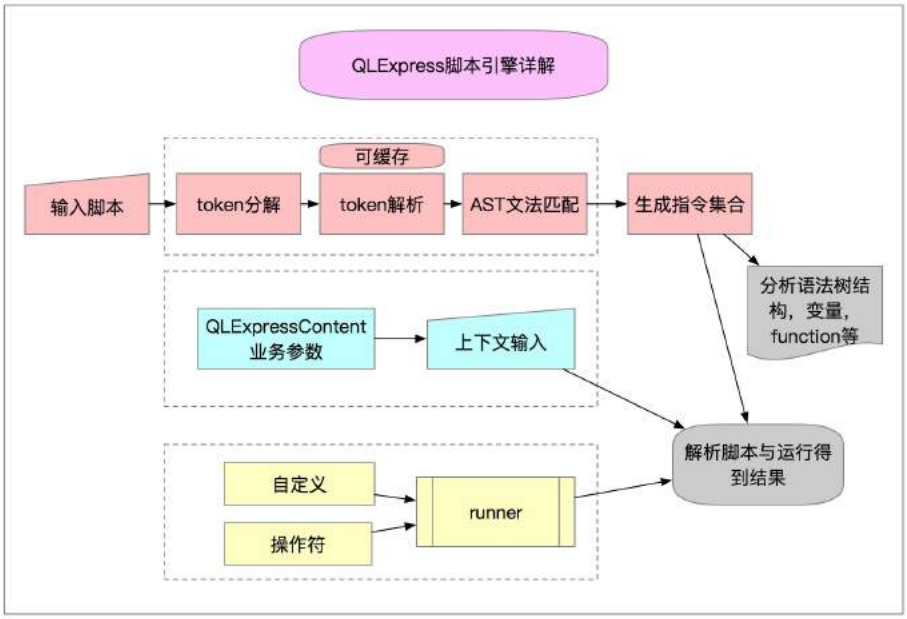
活动规则化，即将业务条件抽象成业务规则，依据业务逻辑对流程进行编排。

营销活动的业务规则抽象，即符合一个流程中所有节点中的规则条件，才能符合该活动。基于对业务的划分，最常见的规则有人群规则，会员身份规则，次数规则，商品规则，库存规则等等。

同一个活动模型下，通过对活动最小单元的规则选取和流程的编排，可以扩展出适用于不同业务场景的优惠活动，真正实现灵活快速的支撑业务。

如何实现业务规则抽象能力？如何将规则的实现侵入性降到最低？优酷会员营销系统采用阿里集团推荐的 QL 语言，一个轻量级的类 java 语法规则引擎，嵌入式规则引擎更加灵活。

该技术选型支持标准的 JAVA 语法，自定义操作符号重载，支持函数定义、宏定义，弱脚本语言，使用灵活，整个运算过程通过 ThreadLocal 来保证线程安全，资源消耗少，不会 new 新的对象，是使用缓存池技术实现。



QLExpress 脚本引擎详解

2. 统一营销架构升级，核心引擎实现

上节的营销系统，能满足 90% 以上的营销需求。接下来的难点是单一优惠形式向复杂的组合优惠形式的转变。单一优惠形式问题凸显，无法支持优惠叠加，优惠价格体系薄弱，核心引擎比较弱，复杂场景难以支撑，维护成本太高。

基于以上的问题点，我们作出了架构升级以及核心营销节点的优化。以新的优惠券能力扩展为契机，抽象出统一营销的能力，实现各种优惠能力的统一。在统一营销的管理下，实现各种优惠能力的互斥与叠加，通过核心引擎的价格等优惠计算，最终输出给交易系统。

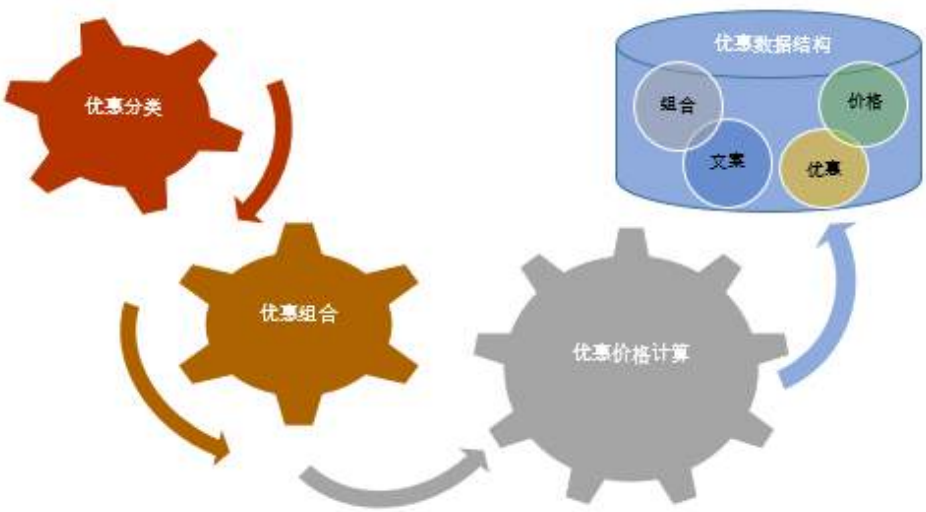
有了这样一个统一营销架构之后，不仅仅是针对优惠券的叠加优惠能力，更加可以灵活的扩展增加其他优惠能力，最终数据合理的数据结构，能够描述多种优惠能力的组合，给用户带来更加丰富的营销体验。

统一营销架构示意图

一个复杂的，具备多种优惠能力的统一营销系统，其核心引擎是怎样实现的呢?引擎核心实现如下：

- 优惠请求上下文，优惠请求信息获取
- 优惠模型定义，不同优惠类型的模型定义
- 优惠分类引擎，根据优惠类型进行数据分类，业务规则生成优惠组合序列

- 优惠计算上下文，将优惠组合序列的计算上下文透传給计算引擎
- 优惠计算引擎，对优惠组合序列，进行顺序的价格计算，并支持多币种
- 优惠计算结果，针对不同的优惠组合序列，输出不同的计算结果，生成以组合、文案、优惠、价格等信息组合的优惠数据结构
- 计算结果排序，多重维度排序后，输出优惠信息



核心引擎实现示意图

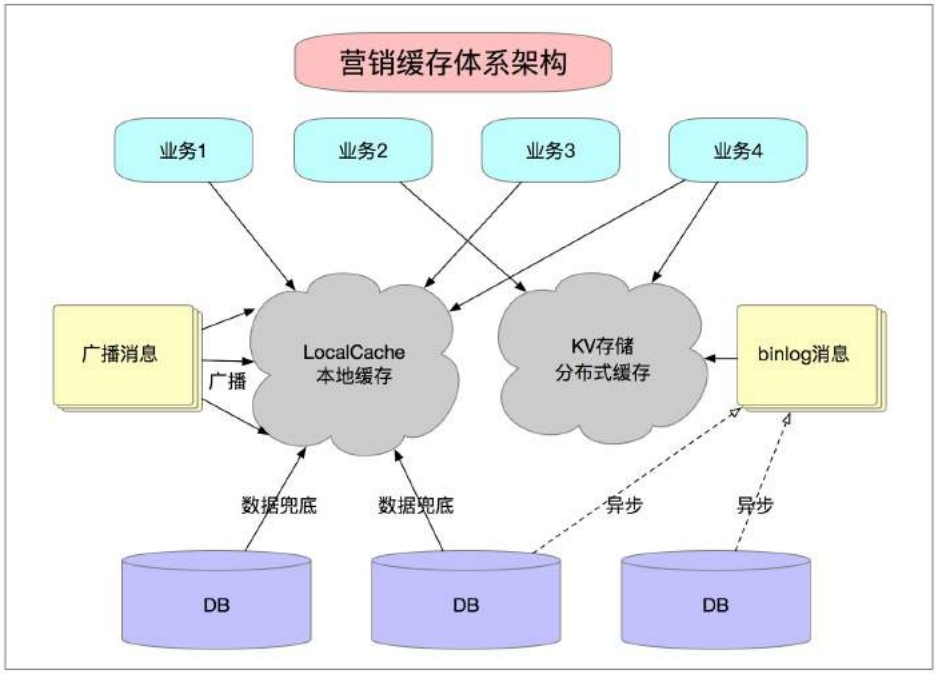
3. 服务如何做到低延时，高可用

3.1 多缓存存储实现

采用 LocalCache 本地缓存与 KV 存储的分布式缓存相结合的方式。

首先将活动基础数据，规则数据，收获数据等数据量小，且访问非常频繁的活动数据，缓存到本地缓存中。分布式集群中每个 JVM 都有缓存，需要有实时的广播消息监听，感知活动数据的更新，实现增量的更新机制。同时我们也完善了全量数据的更新机制，保证全量数据的一致性。

其次将用户参与活动次数等用户维度数据，进行分布式缓存的存储并落盘。作为DB 的数据缓存，提供给分布式集群查询校验次数的快速高性能的查询服务，减少对DB 的查询依赖。DB 的实时变更的 binlog 消息提供给 KV 存储做准实时数据同步。



营销缓存体系架构示意图

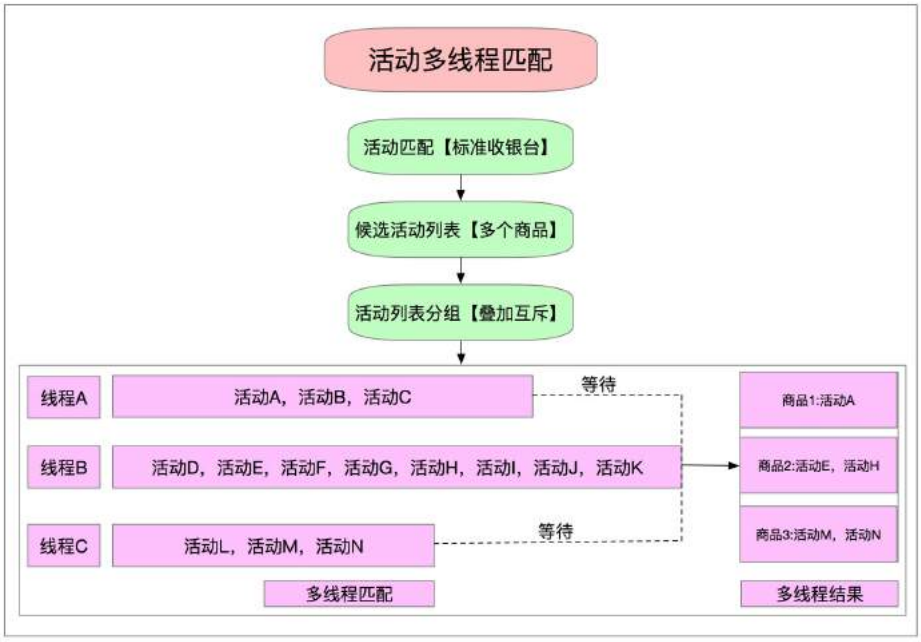
3.2 多线程匹配实现

以交易营销核心节点为细分，将渲染，下单，发放收获等核心节点分类，各自业务节点都采用多线程实现，依据业务类型不同，设置各自的线程池大小，并合理规定线程任务队列，多余线程的超时自动销毁，保证系统最优化。

多线程实现使用 ExecutorService、Callable、Future 实现有返回结果。执行Callable 任务，获取到一个 Future 的对象。在该 Future 对象上调用 get 就可以获取到 Ca 任务返回的 Object。值得注意的是 get 方法是阻塞的，直到等待全部线程返回结果。

线程池该设置多少个线程合适，如何设置才能使得系统性能最优，在这里不做详细展开，可以搜索相关资料学习。其中线程池核心参数如下：

- corePoolSize：线程池核心线程数量
- maximumPoolSize：线程池最大线程数
- keepAliveTime：线程数大于核心线程数时，多余的空闲线程存活的最长时间
- unit：单位时间
- workQueue：任务队列，用来储存等待执行任务的队列
- threadFactory：线程工厂，用来创建线程，一般默认即可



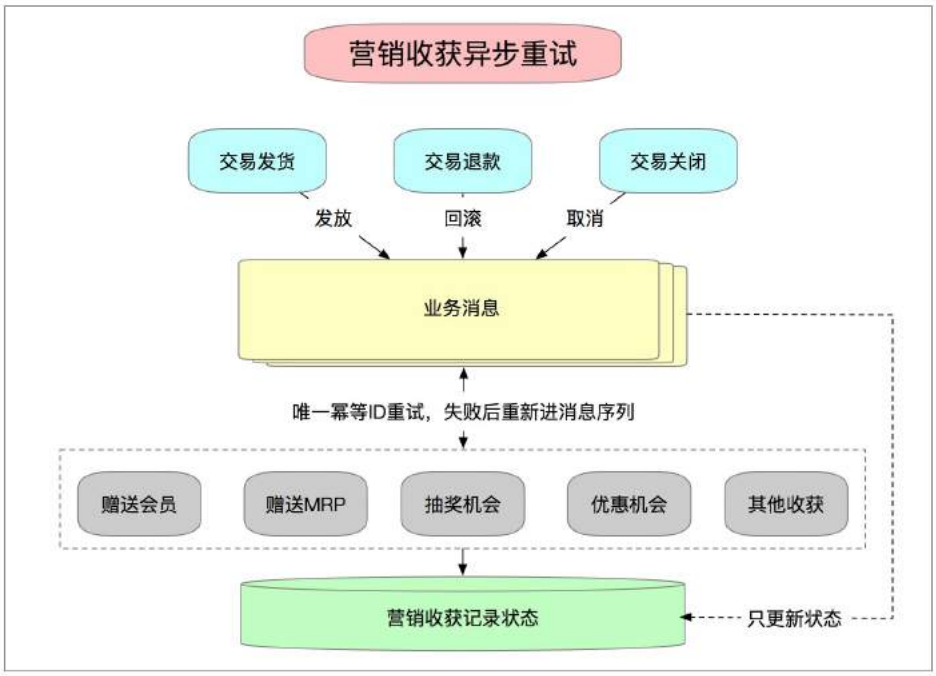
活动多线程匹配示意图

3.3 异步化收获实现

优惠活动收获的发放是与交易系统的支付成功调用紧密联系在一起。支付的成功在交易发放会员权益的同时告知营销发放阿里生态二三方赠送权益。由于二三方权益

接口性能不一，为了实现高性能接口服务能力，采用异步化发放权益 + 重试失败的机制实现。

交易支付成功后，通知营销发放权益，发放通知进入业务消息，营销平台订阅消息，将权益发放出去，针对失败的处理告知消息 later，为下一次重试做准备，并设定重试次数限制。以幂等 ID 为标识，保证权益发放唯一性。



收获异步重试示意图

3.4 分布式事务实现

复杂的业务逻辑，高并发的业务场景，尤其是交易营销类业务节点，要保证业务的一致性，缺少不了锁的实现。

交易营销的下单流程，针对限制次数的高价值优惠能力，要保证业务处理的运行中用户数据的一致性，需要对业务节点加锁。活动的下单节点采用的是分布式锁，通

过业务规则确定锁的唯一标识，同时是可重入锁。锁的实现注意了业务加锁与业务解锁，并设置了锁的超时时间，保证锁高可用。

为了保证业务流程的一致性，还有库存锁的实现。了解 MySQL 的读者都清楚，MySQL 是插件式存储引擎结构，分为两层：一层是 MySQL Server 层；另一层是存储引擎层。我们采用 QUEUE_ON_PK 在 Server 层排队，减少 kernel_mutex 在并发场景下对资源的竞争；依赖 DB 的行级锁，避免使用乐观锁在高并发场景下导致业务的失败，提升了业务处理成功率。

四、总结建议

以上是营销平台如何高效支撑营销需求的实践方案，总结下来有主要两点：

一是系统架构设计要通用，灵活，扩展，才能高效支撑业务多变；

系统架构设计要充分考虑通用性和扩展性，尤其是支撑复杂多变的业务系统。在设计系统的同时考虑到未来业务的方向，反复推敲系统架构的合理性。至于灵活性，对系统模块进行详细的拆解与划分，灵活运用适合业务的设计模式，做到可复用。

二是复杂的业务逻辑背景下，高可用低延时的服务设计要多维度合理化实现；

简单的业务实现考虑合适的设计模式，复杂的业务实现考虑整体的性能优化实现。缓存，多线程，协程，异步，重试，锁实现，分布式最终一致性等等要结合业务处理本身，而不是硬怼上去，更多的是考虑其实现的合理性。

数据工程技术

一文看懂阿里文娱大数据 OLAP 选型

作者 | 阿里文娱技术专家　布鸽

一、摘要

随着各行各业信息技术的发展，数据分析对业务的提升功不可没。数据处理大致可以分为两类：联机事务处理 OLTP（Online transaction processing）、联机分析处理 OLAP（Online Analytical Processing）。而大数据 OLAP 引擎也是百花齐放，使得业内各数据平台的架构设计各有不同，本文主要是对 OLAP 的技术选型做一些探究和总结。

二、背景

大多数公司内部的数据平台是业务主导，服务于业务、聚焦于业务的，产品本身即为数据分析、数据运营等各个数据受众所沉淀下来的业务分析流程。比如视频行业内有纵向的内容宣发数据平台、用户运营数据平台、播放体验数据平台等，皆是对不同业务知识的抽象、沉淀、总结。如此一来产品对业务友好，符合各需求方的使用习惯，专注业务数据。然而这样一来各个数据平台混杂，弊端也暴露出来一部分：

1. 产品维护成本提升：数据产品越来越多，各平台的需求迭代等都需要各个同学负责；
2. 技术成本未收敛：各个业务之间是有很多共性的，绝大多数产品聚焦于业务模式下的固化查询（报表类需求），技术方案类似但选型不一致，如预计算未形成范式、kv 存储选型多。数据平台的价值触及天花板，各自为政，数据发

挥了各业务域下的价值却未向上突破天花板；

3. 仍有部分数据分析师的特定报表散落各地未统一收口。

讲真，这篇文章解决不了这么大的问题，只希望从技术选型上能带来一些启发式的思考。一方面，希望弥补现阶段数据产品，对交互式分析支持力度不够的盲区；另一方面，希望对现阶段的数据产品从技术方案上分类，整合背后的选型逻辑，形成统一的技术框架和产品出口，以此达到增加业务覆盖度、提升数据产品开发效率、控制成本的目的。

引用 SQL on Hadoop 文档中的概念来阐述，可以根据用户查询时延将其做下分类：

- Batch

Batch SQL 的查询时间通常在分钟，小时级别，一般用于复杂的 ETL 处理，数据挖掘，高级分析。最典型的系统是 Hive(ODPS)、Spark-SQL；

- Interactive

如 Impala 和 Drill 提供的交互式分析能力——在 Hadoop 规模的数据集上做传统 BI 和分析。在实现上通常采用 MPP 架构，比如 Presto（ADS 集成），Impala，Drill，HAWQ(Seahawks)，Greenplum(HybridDB for psql)；

- Operational

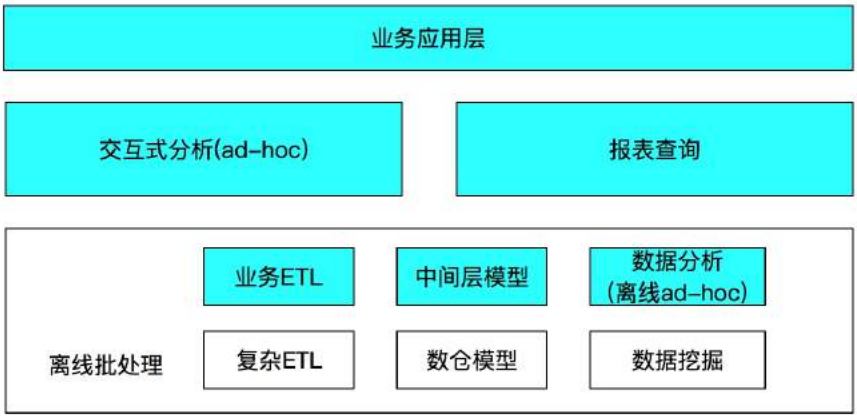
高并发、低延迟的查询，在 OLAP 中常作为存储预计算结果集的 kv 数据库如

Hbase，以及传统的 OLTP 都属于这个范畴，牺牲部分灵活性来换取较高的查询效率。

三、技术选型

上面的分类是从数据规模、灵活性、查询延时三个方面的协调。由此，需要明确数据需求所属的范畴，是复杂的 ETL 任务还是 ad-hoc 查询？还是报表、实时大盘这种高并发、低延迟的查询？

将数据需求放到整体大数据场景中，从技术而非业务的分类来看，如下图所示（蓝色部分）：



- 离线批处理

复杂 ETL 主要是指任务较重的 ETL，抽取、转换、加载产出明细层、数仓模型；而业务 ETL 指产出汇总层、中间层模型。并且相对于较重的数据挖掘查询，将部分要求延时低且数据规模中等、计算复杂低的归为离线 ad-hoc。实际上这里界限并不明显，但稍后会较之做是否加速的选取，所以这里做一个区分。

- 交互式分析

相对于批处理动辄分钟级、小时级的延时，需要提升至秒级、亚秒级、分钟级的

查询，同时还要求不失离线批处理查询的灵活度。

- 报表查询

指大部分的报表类数据产品，查询模式比较固定，但是作为大盘、报表类需求，要求高并发、低延时的查询。

所以有了市面上对 OLAP 从查询类型上的划分：离线批处理、即席查询 (ad-hoc)、固化查询。这里给出几个大类并列举代表性的产品，说明其背后原理逻辑，不深入给出具体的功能、性能差异上的评测。

1. 离线批处理引擎

离线批处理引擎主要用于复杂的 ETL、构建数仓、数据挖掘等对延时要求不高，但灵活性最大的处理引擎，典型的代表如 Hive(ODPS)，Spark。这类引擎典型的优点就是吞吐量大，扩展性好，容错性好；缺点就是低效，适合规模大、逻辑复杂任务。

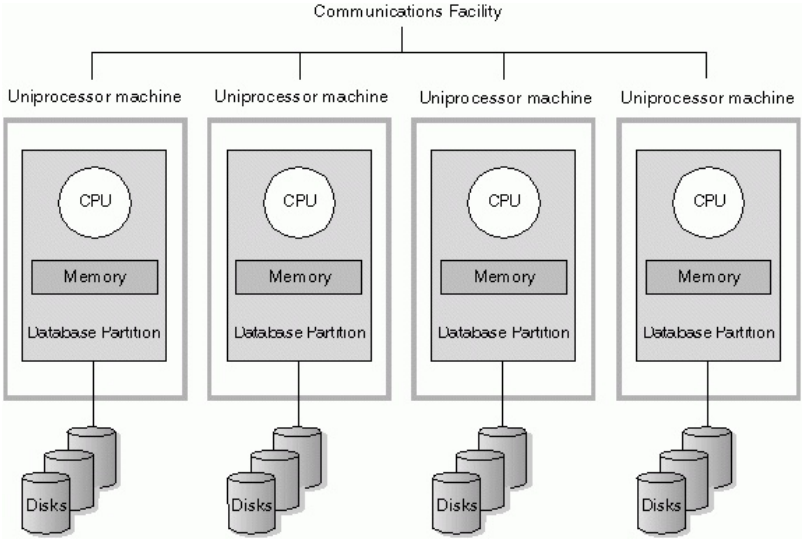
它的逻辑不难理解，随着 MapReduce 的发表和衍生技术的出现，不论是 Hadoop MapReduce 还是 Spark 等工具，共同思想都是对数据文件切片成独立的 Task 做并行计算，其中一些算子通过 shuffle 实现，就要落地中间结果或者缓存数据集，通过 HDFS 备份以及计算的中间结果进行容错，再加上任务调度等问题使得系统整体效率慢。但是扩展性好，理论上的扩展瓶颈只在元数据节点，这也使得更大吞吐量的批处理成为可能，所以离线数仓构建、大型的 ETL 最适合不过。

2. MPP

MPP 架构早于 Hadoop 的出现便大规模应用于企业的数仓建设，Hadoop 也一度被认为是 MPP 的替代方案。MPP 即大规模并行处理（Massively Parallel Processor），每个节点都有独立的磁盘存储系统和内存系统，业务数据根据数据库模型和应用特点划分到各个节点上，每台数据节点通过网络彼此协同计算，作为整体提供数据库服务。有完全的可伸缩性、高可用、高性能、优秀的性价比、资源共享等优势。有

很好的数据量和灵活性支持，但是对响应时间是没有保证的。有很多存储模型，行存、列存、行列混存以节省内存加速查询。当数据量和计算复杂度增加后，响应时间会变慢，从秒级到分钟级，甚至小时级都有可能。

它最早出现是为了解决关系型数据库扩展性差的问题，消除共享存储。



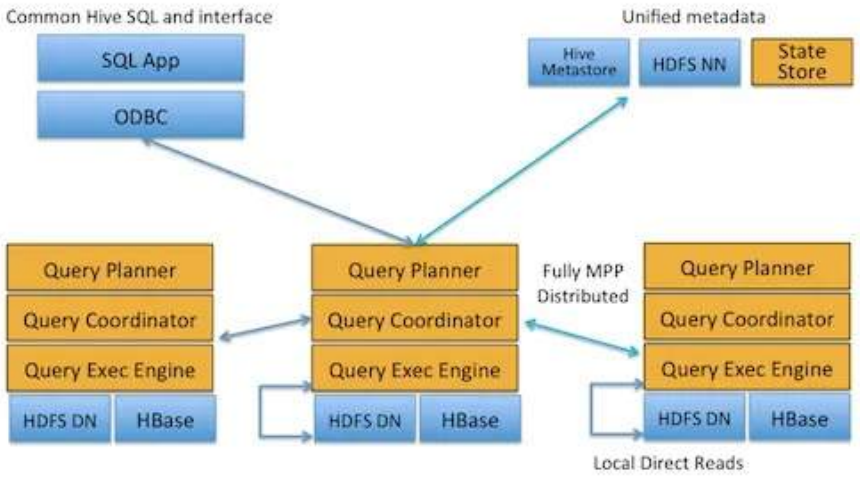
在当时它的扩展性是受人瞩目的，然后到后期被诟病的也是它的扩展性。市面上 MPP 架构的集群规模并不会太大，由于短板效应，存储或者计算都受限与最慢的节点，再考虑硬件的损耗，维护成本居高不下。

尽管它早于 Hadoop 生态，并一度被怀疑已被 Hadoop 替代，但它的存在仍然弥补了一些缺憾，并且给了 Hadoop 生态很多启发式的思考。相比于 HDFS+MAPREDUCE 架构的低效（Hive 也属此类，仅将 SQL 翻译成 MR 也没有执行优化），适合于大部分的交互式分析的场景，规模小于批处理，查询一样灵活，但是响应要求比批处理高。这类产品诸如 Teradata、Greenplum(HybridDB for psql)、HANA、ClickHouse 等。

3. MPP on Hadoop

上面的 MPP 架构都是包括存储的，虽然市面上定义 MPP 架构说的只是计算模型，不考虑存储，但这里还是单独将它拆分出来了，也只是为了细分产品。Hadoop 最大的优势就是其生态，总有前辈们发现了上述问题考虑 MPP 能否和 Hadoop 结合以弥补各自的缺点。

对于 Greenplum 等产品，不重复其缺点，还需要将数据同步到自己的存储，给带来额外的问题是同步成本。如果说 Greenplum 是历史原因，还有一些产品可能是出于其他理解也可能是考虑商业模式，虽兼容 Hadoop 但不属于其生态，在走向产品化的途中一路高歌。但仍有大部分的企业用户渴望 MPP On Hadoop，如 Presto、Impala、HAWQ 等产品，它们在 HDFS 上层提供执行引擎以及优化，提供并行计算能力，有更低的查询延时，代价就是伸缩性和稳定性差。架构类似 MPP：



它们同样适用于交互式分析、即席查询场景。

4. 预计算

预计算系统（Druid/Kylin 等）则在入库时对数据进行预聚合，通过 KV 存储结果

集。进一步牺牲灵活性换取性能，以实现对超大数据集的秒级响应。

类似的，很多情况下没直接用这些产品但是间接实现了预计算的也算在这一类，通常这种方式会结合流 / 批计算引擎以及 KV 存储综合使用，如 HIVE/FLINK 计算结果集后写入 HBase/ 阿里云表格存储等 KV 存储。这一类就是灵活性差，数据需求变更后会影响数据模型。若支持多维度自由组合需要计算 Cube，就要面临膨胀等问题，是预计算 + 查询时计算还是全走预计算都要进行取舍。然而得益与分布式 NoSQL 引擎的发展，其高并发、低延时的特性带来了很好的收益，适用于报表查询场景，是高并发、低延时查询的不二选择。

5. 其他引擎

这类引擎相比 MPP 系统，思路很难一概而论。基本是在入库时创建索引，基于各自的存储模型进行优化，查询时做并行计算，单论常规的多维度聚合计算效率和 MPP 在伯仲之间，但是功能细节上需要慎重考虑。这里讨论两个，一个是 Elasticsearch，一个是 Histore。

1) Elasticsearch

在写入时将数据转换为倒排索引，采用 Scatter-Gather 计算模型，在搜索类查询上能做到毫秒级、秒级响应。但是对于扫描聚合为主的查询，随着处理数据量的增加，响应时间也会退化到亚秒级、分钟级。对去重、join 支持并不友好，分析函数丰富但与传统语义差异较大。

所谓成也萧何、败也萧何，采用 Scatter-Gather 计算模型相比 MR、MPP 来说，效率提高计算成本也低，但是这种计算模型往往采用估算算法代替全数据量计算，有两面性，一定要提前评估。但 ES 发力 OLAP 也是社区将来的发展方向，拭目以待。适用于检索、交互式分析场景。

还有另一个优点单独说明下：在交互式分析中，有一类需求使数据模型变动较大的。如优酷技术侧（主要以播放体验为主，如卡顿等）日志的数据分析，经常会新

增埋点字段以验证线上数据，比如端上临时需要看 HTTPS 协议对延时的影响，这类需求可能只做为验证或者短期内分析有效，需要数据模型快速迭代，这时选择无 Schema 的数据引擎无疑是对开发效率有极大帮助的，所以选择了 ES。如果采用预计算架构，那么新增、删除维度需要反复回刷数据，还要重新构建 Cube，灵活性就差了一大截，另外这类需求的用户只有少量开发，不会有高并发的查询。

优点：

1）采用 Scatter-Gather 计算模型取得很好的性能，聚合计算、去重计算效率较高；
2）无 schema，扩展十分灵活；
3）支持复杂查询、地理位置等分析查询函数；
4）支持全文检索，倒排索引 query 效率高。

缺点：

1）SQL 支持不够完善 (6.3)，性能差，DSL 门槛较高；
2）join 支持不友好，需要提前 shuffle；
3）对于基数大的维度聚合时，buckets 过多导致效率低下，只有一个 reduce 节点，数据量大时有单点瓶颈；
4）HLL++ 去重非精确；
5）不支持联合排序。

2）HiStore

自研知识网格 + 列存，官方介绍比较全也比较细，还有实践例子：https://yq.aliyun.com/articles/159558

四、其他因素

上面的分类以及选型基本是比较概述性的，范围比较广。实际选型的时候，一般

会先界定技术范围，然后针对每个分类的不同引擎、竞品再做更细的调研。这种情况下，可以依次做更细粒度的思考：

1. 数据规模：数据量级多大，一次查询 scan 的数据规模（百亿、十亿）；
2. 实时性：是否要求实时写入、实时可见；
3. 查询类型：即席查询、还是固化查询；
4. 查询延时：查询响应延时要求、是否需要高并发 (MPP 架构基本不要考虑并发，水位就那些，可能一次查询就打满，讨论并发没有意义)；
5. 写入吞吐。需要支持的写入吞吐是多少（对于预建索引的引擎，需要考虑其优化方式，要和实时可见行做折中）；
6. 查询模式。这一步往往和模型的设计同时进行，是否宽表、是否需要 join、是否时序数据等；
7. 精确度。是否要求 100% 精确，尤其是基数计算，一些引擎可能不支持精确去重，这点儿可以和查询延时做权衡；
8. 产品其他功能性需求。这个根据实际情况评估优先级了，要考虑哪些功能对其可用，如租户、安全等等。

揭秘！阿里文娱数据服务平台发展史

作者 | 阿里文娱技术专家　布鸽

一、背景

近些年，随着双 11、618 等营销活动的常态化，优酷对内部的数据分析能力提出更高要求。主要体现在以下三方面：

1. 实时性：传统的离线数据分析已无法满足强实时性的数据分析需求。在面向直播的数据大屏中，需要实时计算在线人数、CDN 带宽水位、直播体验（错误、卡顿等）等大盘数据指标，需要全网的客户端日志以及个别服务端日志，无疑对数据的实时性提出了更高的挑战；
2. 灵活性：除已经固化的业务报表外，新上线的活动、研发为了优化某一个模块所依赖的数据分析，都需要灵活、个性化的维度；
3. 平台化：尽管依赖阿里集团的数据生态体系，但 Case By Case 的业务开发仍旧无法满足实时大屏需求，开发、维护成本增加，如何快速支撑实时大屏的流式计算成为数据团队要解决的核心问题。

面对如此三个方面挑战，优酷数据团队首先解决了数据实时性的问题，并在过程中沉淀出了面向实时、离线的多维度聚合统计分析类场景，提供模型搭建、数据计算、数据可视化的一站式数据服务平台。

二、前身：实时多维度聚合计算

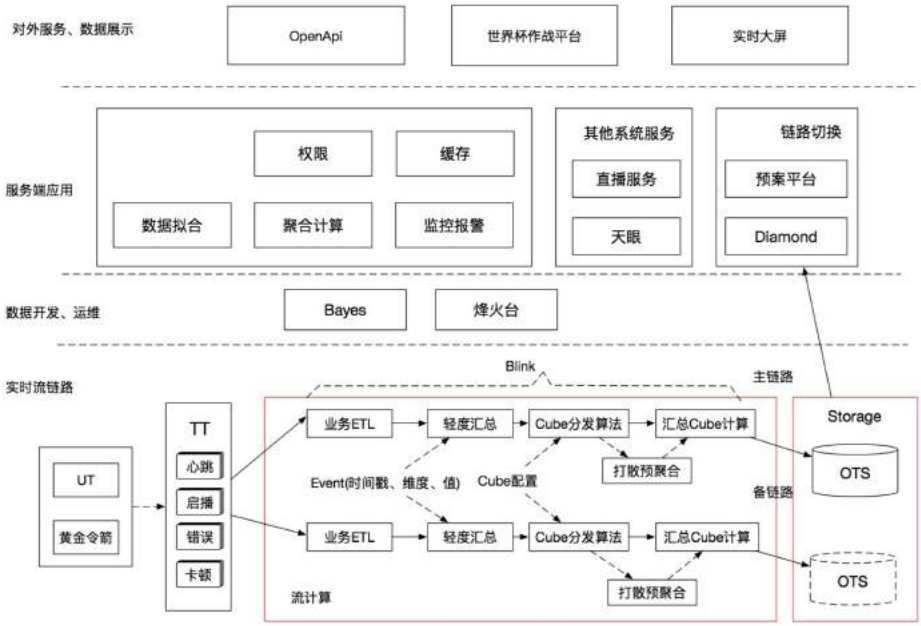
每年的双 11，除阿里集团的双 11 媒体大屏外，每个 BU 的大促、活动、战役都有自己的实时大屏，优酷也不例外，面对的主要挑战如下：

1. 技术挑战：实时大屏都对数据有着非常高的要求，同时面临着高吞吐、低延

时、零差错、高稳定等多方面的挑战。仅世界杯期间直播相关日志的实时流计算处理峰值就达到 1 千万条 /s，处理的总数据量高达百亿，期间实时大屏的稳定性保持在 3 个 9 的水平，并且要求分钟级的数据延时，涉及多维度分析；

2. 业务挑战：在面向直播的数据大屏中，为了实时计算在线人数、CDN 带宽水位、直播体验（错误、卡顿等）等大盘数据指标，需要全网的客户端日志以及个别服务端日志。另外为了让技术同学更全面的分析流量，增加了很多实时的数据维度，比如理论带宽降级策略需要端、版本、清晰度等，再比如播放体验卡顿相关需要运营商、省份城市、网络制式等维度。这些数据监控了当日直播的方方面面，也是活动应急决策的重要依据。

3. 整体架构：



1）埋点日志采集系统：阿里自主研发的无线端 APP、H5 埋点采集工具，并且定义了一套客户端的埋点规范；

2）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；

3）Blink：Blink 是阿里基于 Apache Flink 自研的实时流计算引擎，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；

4）OTS: 表格存储（Tablestore）是阿里云自研的 NoSQL 多模型数据库，提供海量结构化数据存储以及快速的查询和分析服务。表格存储的分布式存储和强大的索引引擎能够支持 PB 级存储、千万 TPS 以及毫秒级延迟的服务能力。

为了满足性能要求，优酷采用预聚合的技术架构。依托于实时流计算引擎，对端上采集上来的日志做实时 ETL、数据立方体预计算并落地 KV 存储对外提供毫秒级响应的数据查询。分别从稳定性、实时性、开发效率提升上做了以下工作：

1. 稳定性

面对如此庞大数据量的实时计算，稳定性压倒一切，为了保障实时大屏的稳定性，做了一系列的任务优化、蓄洪压测、主备链路等工作。

1）任务优化

首先是任务的优化，众所周知，在流式计算 shuffle 中，当下游处理能力不足时，会通知上游停止发送数据，从而避免数据丢失。这就造成了任务的反压，使得实时流计算的吞吐量一直上不去，这时候就要对任务整理分析性能瓶颈并作出优化。

我们遇到了三个造成反压的问题，一个是 IP 解析的 UDF 问题，一个是数据倾斜，一个是 KV 存储的写入瓶颈。

a）任务上线做蓄洪压测时发现，source 节点资源 (cpu/mem) 并未打满，却出现了数据延时。并且并行度已经达到了 TT 分区的上限，无法通过简单的增加并行来解决。经分析，由于 Flink 中一个 Vertex 是多个 Operator 的结合，进一步拆分 Operator 后发现是解析 IP 的 UDF 出现了性能问题，修复后 source 节点吞吐量大幅提升；

b）数据倾斜是大数据处理中老生常谈的问题了，在优酷的业务场景下，在做聚合统计的时候，很多时候是按 CDN 域名、运营商、版本等维度统计，不可避免出现热点问题，尤其是 CDN 域名基本都集中在头部一两个上，长尾却有很多。造成的现象就是该节点增加并行度毫无收益，并且资源大量空闲，in_queue（用来缓存需要计算的数据）却一直被打满，造成反压。定位该问题的思路就是进入明细看每个 TaskExecutor 的 tps，发现除了头部的几个外，其他节点 tps 和 in_queue 都是 0，所以这里有明显的数据倾斜。解决该问题的思路是先对散列字段做 HASH 取模分桶，先做桶内的局部聚合，再做全局聚合；

c）为了保证更高的实时性，用的是纯流式计算，并非 window 的方式聚合，这使得输出的 tps 要高的多。发现 Sink 节点也出现了性能瓶颈，并且增加并行度已无效，为了提高写出速度，通过调整 Sink 的 batchSize（根据 rowkey 去重的 buffer）参数来优化输出。

2）蓄洪压测

蓄洪压测也是依托于 Blink 提供的模拟上游压力的能力，通过回放 TT 的历史日志来生成影子作业，完成链路的压测。并且在压测中进行任务优化并应用于原实时任务。在战役前共组织了三次大数据压测，完成了几十个作业的压测配置优化，压测的峰值 TPS 达到了 3 千万。

3）主备链路

为了解决 OTS 分区自动裂变（类似 HBase 的自动分区，当出现某个 Region 的写入热点时，为了提高吞吐会对该 Region 进行分区裂变以提高吞吐，但是会导致服务的短暂不可用），以及流计算高峰期存在 Failover 的风险，对实时链路做了不同集群的双链路备份，以及存储不同集群实例的备份。并且在服务端通过预案平台做查询请求的无缝切换，以此来保证链路的可用性。此外，由于主备链路计算数据立方体，维度较多，对于核心的大屏数据指标，单独隔离了兜底的实时计算任务，保证即使主备链路都失败也能保证核心业务的平稳运行。

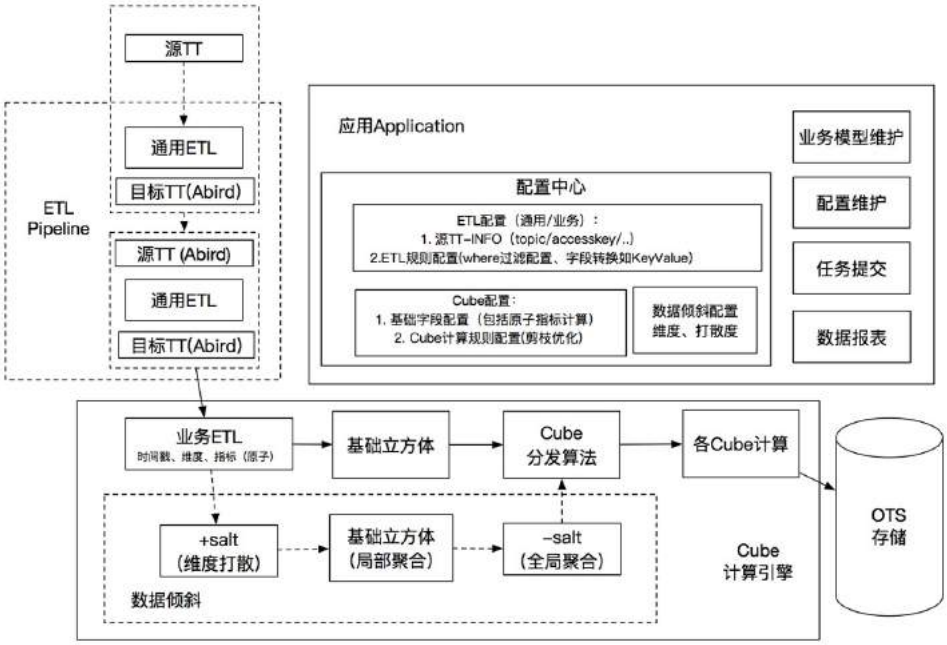
2. 实时性

在实时大屏的业务中，尤其是技术侧的大屏，涉及到直播作战的实时决策，对数据实时性要求较高。为了保证实时性，弃用 window 的计算方式，采用纯流式计算（中间结果通过 state 保存，并流式输出到存储中）。依托于 Blink 强大的实时计算能力，为了提高吞吐，读写 State 开启了 MiniBatch，基于事件消息来触发微批 State 的更新操作。

3. 开发效率

从前几次的实时计算开发经验来看，one by one 的业务开发无法满足日益增长的实时大屏需求，开发、维护成本逐渐增加。如何快速支撑实时大屏的流式计算成为数据中台团队要解决的核心问题。

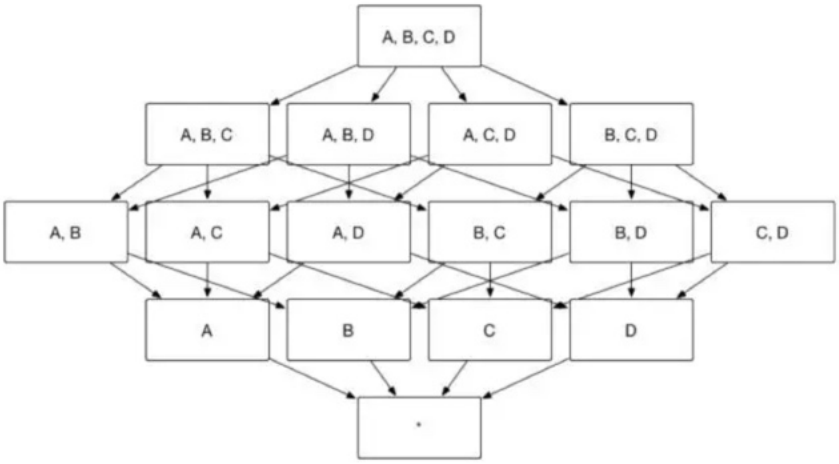
在实时数据统计的场景中，大部分都是根据某些维度进行去重、求和、算记录数、求最大值、求最小值、求平均值、算排行榜等聚合计算，另外还会涉及到实时多表 join、静态维表关联等。对业务模型抽象后，逐步形成了配置化的实时计算开发框架。



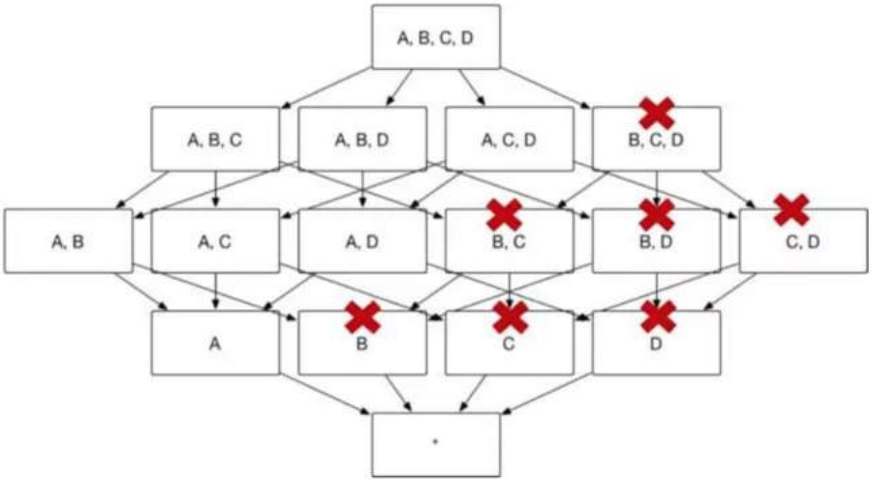
核心思路是经过 ETL-Pipeline 将数据处理成时序数据模型（指标、维度、时间戳），针对生产出来的时序数据，再进行数据立方体的配置化计算（同时对数据立方体做业务剪枝，优化成冰上立方体从而降低计算成本），将聚合计算的结果按统一的协议格式写入 kv 存储中。这样，实时任务的接入就只需关注业务 ETL，同时需要聚合计算的维度即可，不同业务按同一套协议规范写入 kv 存储中，并通过 biz_code 加以区分，服务端通过 biz_code 面向指标、维度查询不同业务的数据。

数据立方体

假如将计算某指标在 [A,B,C,D] 四个维度的排列组合，那么所计算的数据立方体长这个样子：

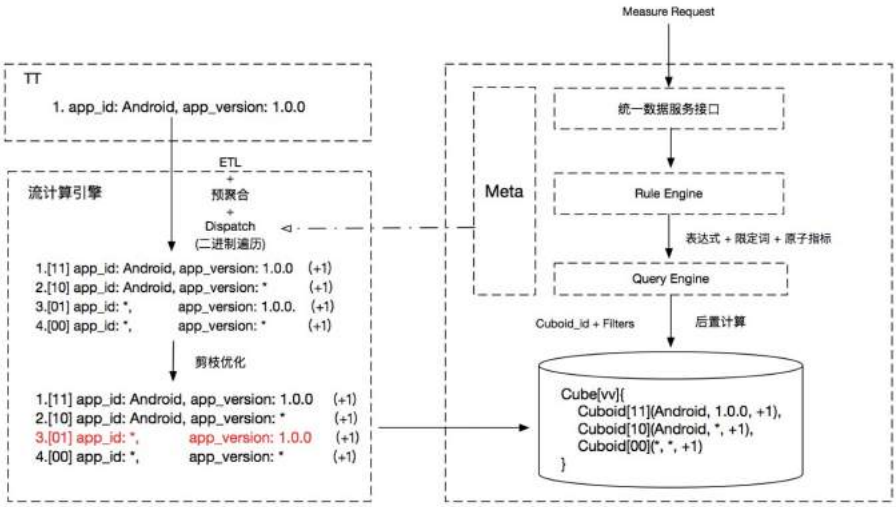


如上所示是一个完全立方体，不难算出若维度为 n，则总的立方体单体个数为 2^n 个，其存储的数据量总大小为各个维度基数的乘积。如果数据立方体中所有方体都预先计算，所需存储空间可能爆炸，特别是当立方体包含许多维时，会造成维灾难。尤其是在实时计算中，翻倍的数据量级会大大影响流计算吞吐量，所以根据业务的实际需要会对立方体做必要维度、层级维度、维度组等裁剪。这些完全通过配置化来实现。剪枝后的数据立方体：



那么如何实现实时计算中数据立方体的构建呢？实现的算法并不复杂，通过 udtf 来实现单条记录的 dispatch，对分发后的数据做聚合统计：如计算 [A,B] 的数据立方体，分发后的结果为 [A,],[A,B],[,B],[,]。内部通过遍历二进制数组来做到所有维度的不重不漏，并在此基础上进行过滤，实现数据立方体的裁剪。

原理图示：



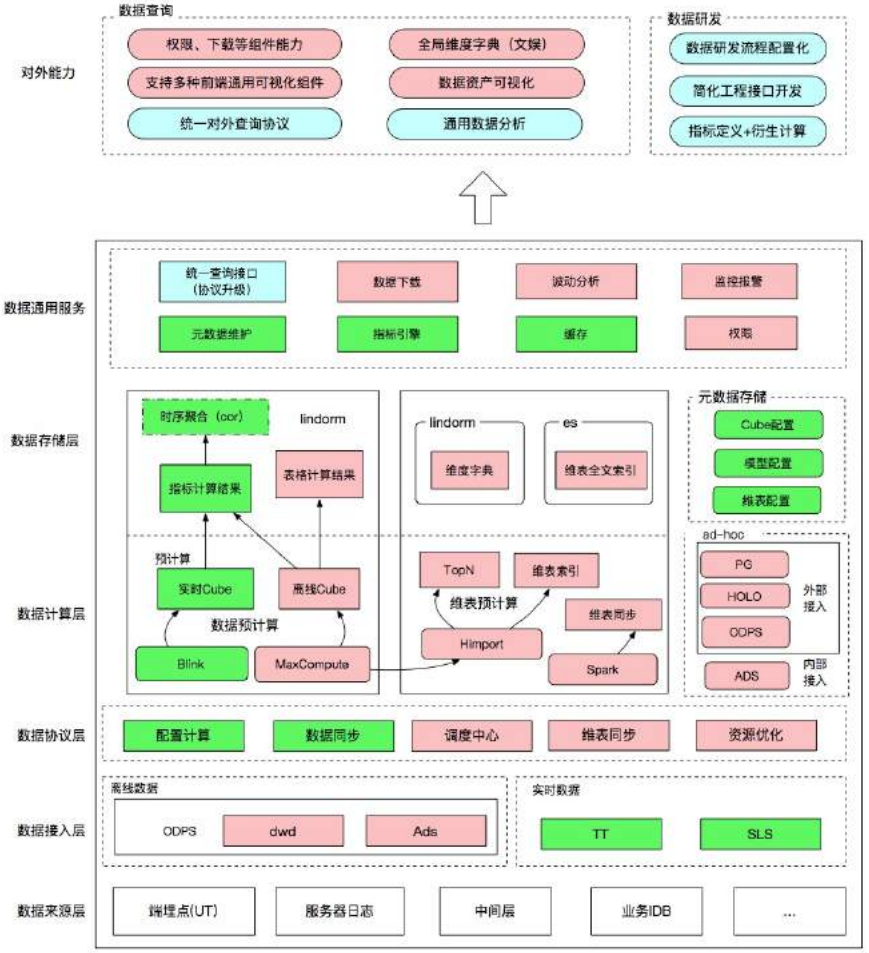
三、数据服务平台

随着上述问题的解决，业务开发的效率逐渐提升，业务也向着更灵活的方式演变。随着业务的逐渐增长，Blink 实时任务数逐步增加，又衍生出了一系列的挑战，如任务的后续维护、元数据管理、降低成本、数据膨胀等等。

为了更好的把优酷的实时、离线数据赋能给更多的业务方使用，我们通过架构升级、底层逻辑封装等方式来突破现有瓶颈。旨在打通文娱数据中间层的模型搭建、OLAP 数据引擎、以及工程侧元数据管理和数据查询服务，结合前端通用可视化组件，提供集数据研发、数据展示一体的文娱数据平台。面向两类用户提供服务，一类

是数据研发同学（技术、产品、数据分析），可通过数据服务平台构建多维度分析的实时、离线数据模型；另一类是数据使用方（运营、产品、数据分析、技术），如运营可通过图表查看数据或者技术同学通过统一接口获取数据（HSF/HTTP）。

以下为系统架构图：



从整体架构设计可以看出，为了解决 Blink 实时任务不断增加维护成本升高的问题，我们把数据模型集中管理，在平台上定义数据模型，并打通 Blink 和 ODPS 提

交实时 / 离线任务，同时提供维表查询检索的能力，将文娱数据集中起来为业务提供数据服务。

在降低计算成本的目标上，数据服务平台做了两件事，一个是计算脚本模版化，面向不同的业务场景提供不同参数配置、生成不同的计算脚本，比如实时计算是否启用 window 模式等；另一个是由数据服务平台提供指标的后置计算能力，如常见的点击率指标，ctr= 点击 pv/ 曝光 pv，由于双流 join 需要消耗额外的计算资源，并且行为日志量级较大，更加剧这个问题。通过数据服务拆解两个数据模型，并创建衍生指标配置点击率的表达式，即可查询。

另一个问题要解决的就是预计算不适合的场景，比如维度较多，维度基数较大的部分场景；AD-HOC 场景等。由于数据服务面向上层使用方已定义了一套面向多维度聚合计算的查询协议，在常规的 OLAP 引擎上也可以做 SQL 语义的转换，针对这种场景可以定义数据模型并由数据服务做建表、数据同步等操作，目前仅支持阿里集团内的 ODPS 等数据源。

四、未来规划

在现有的数据服务平台上，我们逐渐收纳了标准定义的各类数据指标，如内容消费分发的曝光、点击、CTR、转化率等，又如播放体验类的 vv、成功率、卡顿率、秒开率等等，这些数据一方面服务于数据分析场景做报表的可视化展示，一方面提供给其他系统应用做策略等业务使用。在收口此类标准化数据后，未来就可以在数据之上提供更多自助式分析工具、监控报警、日报月报等可插拔的功能组件，以满足不同的业务需要，抽象此类公共能力。另一方面，在资源成本方面，尤其是在预计算模块，可以增加数据立方体的诊断优化，持续优化降低成本，以防止租户内资源使用过度等问题。

五、结语

做技术的同学除了解决业务问题本身以外，要有敏锐的嗅觉和产品思维发现业务

问题背后的深层逻辑，比如数据服务平台最初的目标仅是为了解决实时多维度计算的问题，然而成本、开发效率、数据维护等潜在的问题浮现而出的时候，这就要求技术同学要走在业务之前，逐步抽象问题，通过平台化或者其他手段来提出解决方案，当业务能够尽快落地实现的时候，也能推动业务本身尽快进入新一轮的迭代周期，充分发挥技术价值。

内容运营平台

内容智能平台

机器如何为内容体检？基于视听 AI 的内容创作理解

作者 | 阿里文娱算法专家　梵生

一、视觉 AI 的内容创作理解的背景与机遇

视听盛会，剧集、综艺、短视频等都是娱乐行业的主流载体，而高品质的长视频（剧集、综艺）是内容行业提升用户粘性的关键，也是娱乐行业的必争之地。但是，剧集、综艺等长视频面临严峻的问题：剧综的拍摄、剪辑投入巨大，制作周期长，但目前行业很难在播前甚至制作早期进行质量评价或品控。高投入和高不确定性的质量评估体系形成了主要矛盾。

大数据与人工智能已经在各个行业大展身手，而海量的视频数据、用户观看数据，已经为人工智能算法提供了肥沃的土壤；视频、音频、文本等非结构化数据，天然符合人工智能（深度学习）算法擅长的领域。因此，使用 AI 技术来对视听介质进行全方位解构，并利用海量数据、发掘内容创作规律，辅助内容质量评判是 AI 技术落地的一大领域，也是一片 AI 应用的蓝海。

二、内容创作理解的体系—成片体检

成片体检是我们使用 AI 算法对内容创作质量进行量化的尝试。类比人的体检，成片体检主要指利用视听 AI 技术，计算出能够反映内容创作质量的各个维度指标，并根据不同类型的内容，分别计算出优质内容在各个维度指标上的最佳取值或区间，形成内容的健康标准。我们通过 AI 算法，对待检测内容在各个维度上与健康内容进行比对，给出相应维度的预警、实现成片质量的体检报告辅助剪辑优化。我们期望做

到的就是类比医学中的高精密仪器，全方位、准确地数字化扫描整个内容，进而赋能整个内容行业。

成片体检的整体框架如图 1 所示。整个框架可以分为指标层、指标提取算法、融合层、基础模型层。

1）指标层是依赖于内容创作体系所总结归纳出来的，可用以量化内容创作的计算指标。为了从原始视频媒介得到这些指标，我们需要自下而上分别建立基础算法层，算法融合层和指标提取层；

2）基础算法层指对原始视频介质的解析，学术上属于典型的视频理解与视频解构。基础算法包括典型人物检测、人物识别、人物重拾、场景识别、动作行为识别，也包括镜头切分识别、表情识别、情绪识别、景别识别、背景音乐情绪识别等内容行业特别关注的基础模型。基础模型往往得到一些视频基础元素级别的结果，需要经过模型融合层的相关模型，才能形成具有内容意义的中间结果；

3）模型融合层包括角色轨迹识别、故事场景切换，角色情绪发展模型等。指标提取层则直接根据融合层结果或者基础模型结果，结合用户播放、评论数据筛选出的优质内容，计算出内容的健康标准。比如，一部正常的电视剧，不同番位的出镜占比、故事线的占比是怎样的，一般用怎样的镜头时长，怎样的景别占比；角色交互的复杂度指标如何等等。

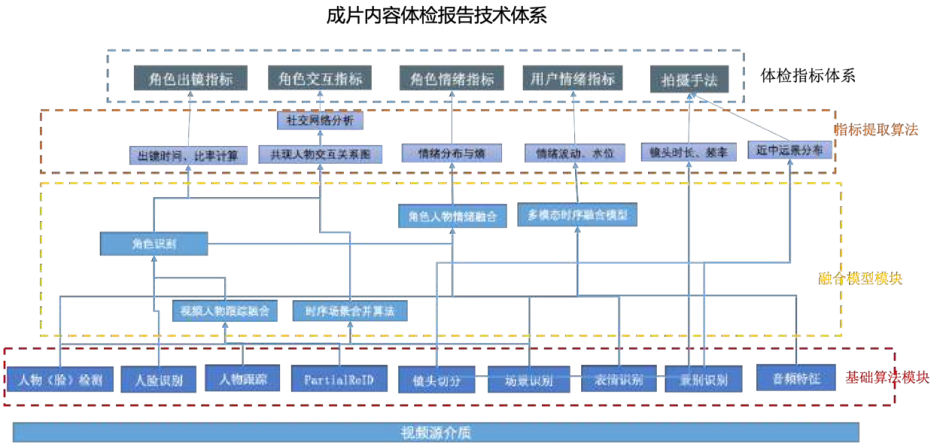


图 1　成片内容体检报告技术体系

三、成片体检体系下的视觉技术探索

为了支撑成片体检体系搭建，我们在音视频基础模型上进行了一些探索。整个体检指标体系所需要的基础模型涉及几乎视频理解领域所有的问题，包括但不限于人物（人脸）检测、识别、跟踪，人物重拾，人物动作识别，人物表情识别等等。为了充分发挥阿里巴巴整个集团的技术优势，我们在部分模型上选取了集团或其他团队的模型，比如人脸识别、动作识别等等；同时，针对内容行业的视频介质特点，我们团队也自研了针对内容视频的定制优化的人物匹配框架，视频情感计算，剧集场景识别等基础模型，并在各自领域的接近或者超越业内最佳性能。下面着重介绍人物匹配框架与观影情绪模拟这两部分工作。

1. 剧综人物匹配框架

成片体检的许多指标需要依赖于准而全的视频人物角色识别。内容行业的视频是多机位、多角度拍摄剪辑而成；同时，根据拍摄需求，人物妆容、衣着都有着较大差异，这就导致了人物识别问题有别于传统的人脸识别或者行人检测。主要体现在：

1）多机位的拍摄导致的是非配合式的人脸识别，在侧脸、背面、远景下，人脸

识别尚无太好的解决方案；

2）镜头切换与剪辑有别于监控场景，打破了视频内在逻辑，使得检测跟踪的作用有限；

3）根据创作需要，经常交替出现人物的全身、半身画面。上述特点会导致大量侧脸、背面、远景、半身、全身角度下人物的丢失。无法满足我们“准”、“全”的要求。

针对上述问题，我们设计了剧综人物匹配框架。如图 2，我们把剧集内的人物“准全”的识别，拆解为镜头内和跨镜头的问题进行分析。在同一镜头内，我们复用成熟的检测与跟踪，那么跨镜头则需要人物重拾。对于长时的多姿态，则需要利用时空、人脸人体、上下文等整体信息进行人物匹配，这就类似于多维信息下人物检索问题。人物重拾的特征作为基础特征层，被多维信息人物检索使用。

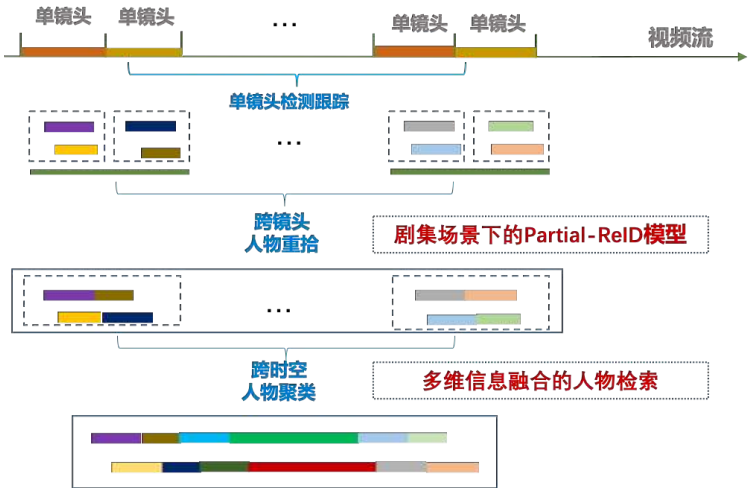


图 2　剧集场景下的人物匹配算法框架

在基础人物重拾模型上，我们提出了适用于影视行业的 DramaReID 剧集 ReID 数据集和 ESA-ReID 模型。DramaReID 数据集来自于我们海量的剧综视频数据，覆盖了上万个人物的全身、半身等视角，是目前业内已知的最大的 partial reid 数据集。针对刚才提到的人物重拾问题特点，我们提出了 Entropy based Semantic

Alignment Re-ID 模型，模型的整体架构如图 3 所示。类似与传统的 re-ID 模型，我们使用 ResNet50 作为主干特征提取网络，随后，针对全半身比对时需要的语义级别的特征和比对，我们分别引入了语义分割多任务，基于语义分割不确定性的置信度模块和基于置信度的动态比对模块，以解决剧综场景下全半身人物重拾问题，得到的人物形象表征也为后续多维人物检索做准备。

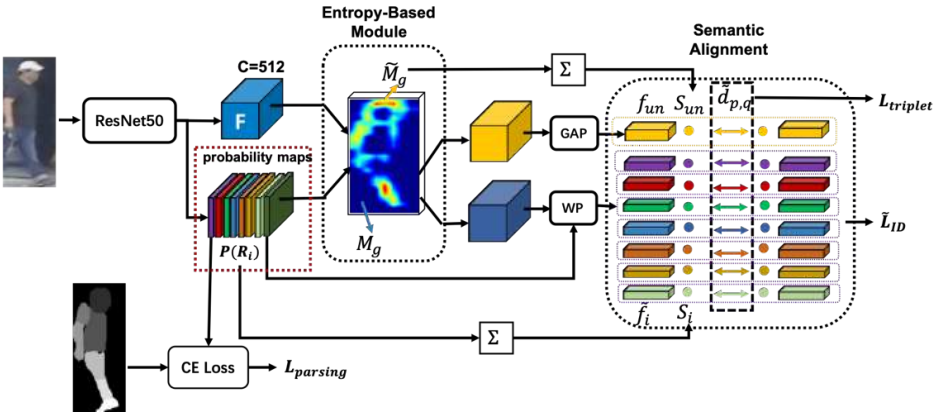


图 3　ESA-ReID 模型架构图

具体地，我们使用成熟的语义分割模型，对待识别的人物进行人体部件语义分割，该分割结果作为监督信号，来训练我们的语义分割支路。语义分割支路得到人体各个部位的分割区域及其概率后，会经过基于熵的不确定性计算模块，来获得人体高确定性和不确定的掩模区域。我们的方法是业内首个利用熵来度量人物重拾任务中的语义分割的不确定的。通过度量不确定性，一方面减弱语义分割的误差对模型性能的影响，另一方面，不确定性高的区域，正好对应了人体缺失或被遮挡的部位，可以用于人物相似度计算。通过基于熵的不确定模块得到确定性和不确定性掩模后，我们可以得到人体各个部件的特征，以及对应的不确定性。在计算待匹配的两个人物的相似度时，就能够通过各个部件一一比对，并用其对应的不确定性来做权重进行计算。一方缺失的部件，其不确定性高，进而权重变低甚至为 0，这样相似度就取决于待比较的两个人物共同出现的身体部位的视觉特征。

整体上看，我们的 ESA Re-ID 方法是端到端的模型，在 inference 阶段不依赖任何第三方模型；同时，我们引入的基于熵的度量，极大程度降低了语义分割支路的误差，并在语义部位级别进行了对齐比对。我们的模型在业内公开的数据集，如 Market1501，DukeMTMC 等达到了 SOTA 水平，在 Partial-ReID，PartialILID 等 partial 测试集上，大幅超越了 SOTA。在我们自建的 Drama ReID 数据集上，我们的方法也和业内主流的 SOTA 方法进行了比较，性能上均有巨大提升。具体可见我们后续将要公开的论文。

人物重拾的特征目前无法解决剧集中长时场景下，人物变装变形象的问题。该问题可以定义为多维信息的人物检索问题。多维信息包括人脸特征、人体特征、场景特征等等，业内也有学术论文进行了相关的研究工作。目前，我们设计了一种无监督的，基于人脸、人体多维特征长时跨镜头层次聚类的方法。具体图 4。整体思路是，我们期望在时域局部使用人物重拾特征进行人物合并，而在全时域使用人物与人脸特征进行合并，这样综合人脸和人体重拾特征的层次聚类，在聚类的纯度、精度都有巨大提升。

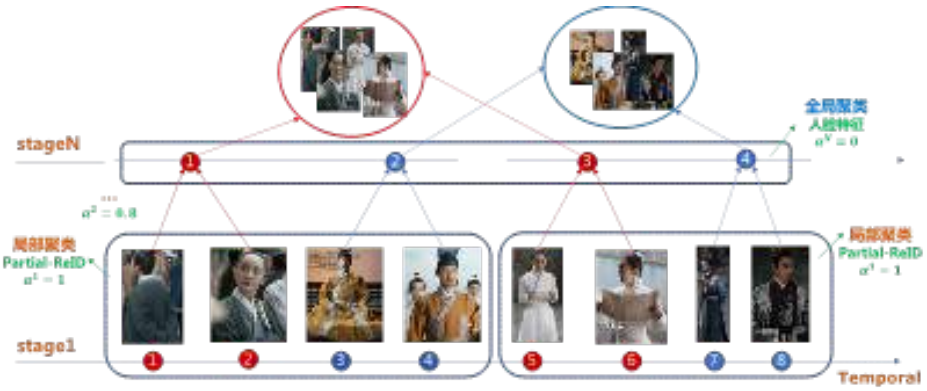


图 4　人物层次聚类示意图

在使用了上述的人物匹配框架之后，我们的剧综人物的准确率、召回率都有 10% 以上的提升，不仅为成片体检提供了准确的人物类数据与指标，还为优酷的“只看他“业务提供了算法支撑，提升了人物召回率，降低审核成本。

2. 观影情绪模拟

共情是内容拉动观众的核心，预测内容能给观众带来的情感体验是内容体检的另一个重要且直观的指标，能够在内容播放前就预测观众的观看的结果，比如情绪高点、低谷，或者平局的情绪高点的时长占比等，将对视频优化有重要指导意义。直接通过视频内容来预测观众的情感状态是音视频和情感计算交叉领域问题。在情感计算领域，除了使用典型的 7 类情感之外，学术界会使用 Valence 和 Arousal 二维情绪模型，来细粒度全面描述人的情感状态。Arousal 可以理解为是情绪的强度，范围为（−1，1），1 表示最强，如激动，−1 表示最弱，比如睡着的状态。Valence 表示情绪的正负（−1，1），1 表示正向，−1 表示负向。那么任何情感状态均可以使用在 Valence 和 Arousal 的坐标系中表示。另一方面，视频表征领域，可以利用视频的场景、人物、行为姿态、背景音乐等多个维度共同表征视频特点。那么上述观影情绪模拟就是建立上述视频表征到情感状态的映射函数。

基于学术界的相关研究和已有的开源数据集，我们提出了基于多维视频表征的情绪预测模型，模型的输入是连续的剧综片段，输出预测的用户逐时情绪 Valence 和 Arousal 值。

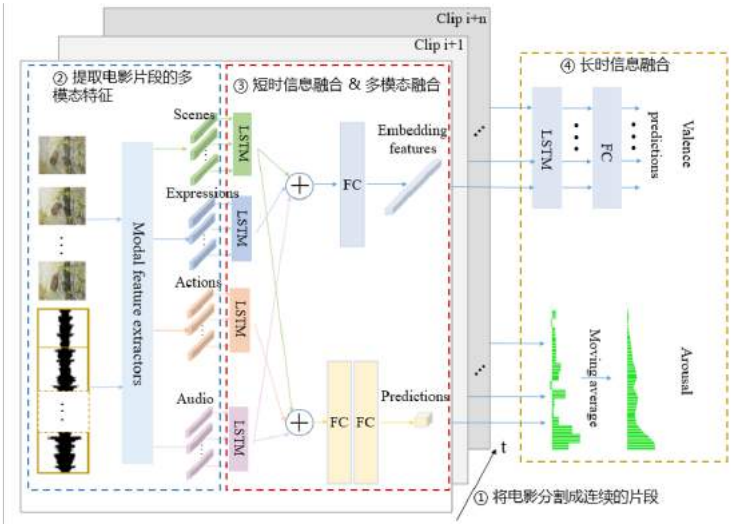


图 5　观影情绪模拟模型结构图

模型的整体结构如图 5 所示：首先，将整个视频分成连续的固定长度的片段，随后，对每个片段进行基础特征提取。在基础特征上，我们使用了分别提取了场景特征，人物表情特征，人物行为特征和音频特征。具体地、场景特征使用了基于 Places365 数据集 pretrained 的 Vgg6 作为特征提取器，提取每帧的场景表征；人物表情特征则使用了我们自研的人物帧级别表情模型作为特征提取器，逐帧提取该帧图片的人物特征；行为特征使用了 OpenPose 的预训练主干网络，音频特征使用了基于梅尔倒谱和 Vggish 的特征提取器提取音频帧特征。在得到各个模态的逐帧表征后，我们引入了长短时融合机制，以反映情绪随时间具有依赖性的特点，并兼顾长期趋势和短期波动。在短时特征融合上，我们将每帧的各个模态特征，分别送入各自的 LSTM 层，得到各个模态在该视频段落的最终表征。经过 LSTM 之后的多模态特征经过合并后，再次送入第二层 LSTM，该层 LSTM 的输入是相邻视频段的融合后的模态表征，输出的是每个视频段的 Valence 或 Arousal 值。第一层 LSTM 是短时时序融合，第二层 LSTM 则是长时时序融合。考虑到 Valence 和 Arousal 存在一定差异，我们对两者分别进行建模。人的情绪强度往往具有更强的平滑性，而 Valence 则可以随片段快速转变。因此我们对 Arousal 部分进行了滑动平均处理，得到最终的 Arousal 结果。我们的模型在开源的多媒体情感计算数据集上，在 MSE，PCC 等指标均超过了业内的 SOTA 水平。模型具体实现和数据测评见我们公开的论文

（https://arxiv.org/abs/1909.01763）。

情感模拟反映了用户对内容的真实感受，我们使用模型的结果和真实线上视频的收视数据进行了对比，发现了惊人的一致性。这就充分证明了模型的使用价值。图 6 是在电影《我不是药神》的 case 和《长安十二时辰》收视曲线和 Arousal 曲线的比对结果。可以看到，在《我不是药神》的 Valence 预测中，我们的情感 Valence 正确的反映了电影前喜后悲的情感趋势。图 7 是《长安十二时辰》的情绪 Arousal 预测和收视曲线比对，发现情绪高潮点和低点对应了收视高点和低点，这进一步证实了用户情绪模拟的巨大业务价值。

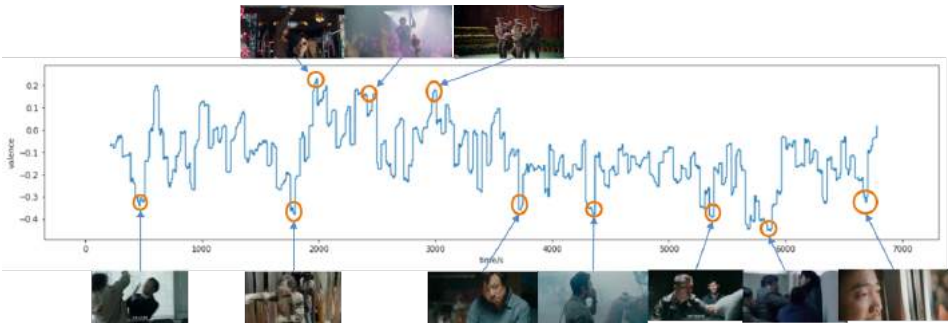


图 6 《我不是药神》Valence 预测结果示意图

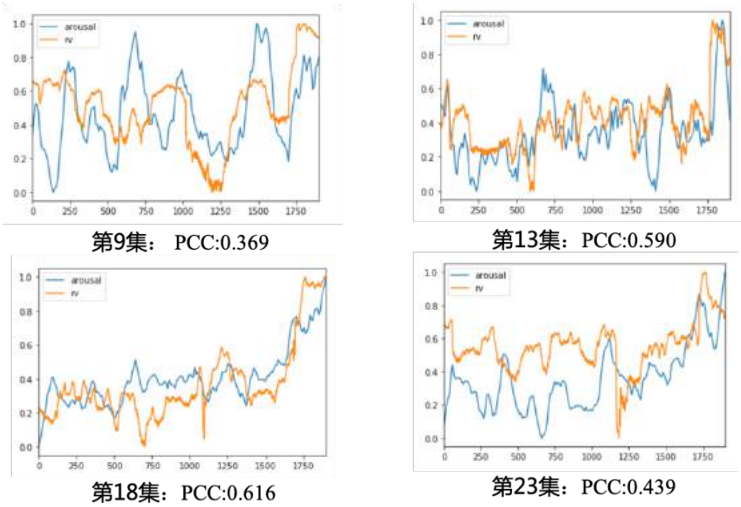


图 7 长安十二时辰收视曲线与 Arousal 预测对比

四、成片体检的应用与未来

目前，我们已经建立了基本的基于人物和情感的体检体系，并根据全网头部剧集，建立了各个题材相应的“健康标准“。覆盖了主要的剧综播前评估与优化。在人物侧，通过人物识别得到的故事线、人物出镜指标，帮助我们发现了前期热播的剧集在第一集主人公故事线缺失的预警，并得到片方认可和修改。我们的情绪模拟曲线，全面在覆盖优酷的自制综艺、剧集、网大。其中，通过网大开放平台对业内

透出的体检能力，能够为网大片方检测成片的高潮低谷，和相对业内优质内容的水位参考，为平台带来了大量的签约与合作，不少内容经过体检和优化后成为了网大爆款，如图 8。

图 8　开放平台内容辅助优化效果喜报

未来，整个成片体检将会更加深入和精细化。从应用角度看，我们将继续扩展体检维度，同时深入题材特有的细粒度体检指标，形成题材定制化体检能力。在整个视觉 AI 技术上，围绕成片体检，我们将继续在多模态人物检索，多模态情感计算，人物交互片段检测与关系属性识别等理解视频剧综内容所面临的特有的问题上深入研究，持续向文娱行业输出算法成果与能力。

视频内容理解核心技术解密 —— Partial re-ID 在成片体检中的技术实践

作者 | 阿里文娱高级算法工程师　朔衣

引言

人物重拾（Person Re-identification，简称为 re-ID）是一项在现实世界非常具有挑战性的任务，它旨在利用视觉算法模型匹配出不同视角的不同摄像头下的相同人物。无处不在的遮挡，复杂的背景，光照的变化等都使得这个问题困难重重。而目前大部分的开源数据集，如 Market1501，DukeMTMC 等都是在监控视频下采集的行人数据，这些数据集的人物形象大多是直立全身的固定姿态，目前的一些主流的算法模型对于这些数据集都有很好的性能优化和提升。然而，基于内容视频数据进行的人物 re-ID 的研究工作却是屈指可数。区别于传统的监控视频，内容视频中有大量的人工剪辑和镜头切换，大量多角度和多机位的镜头拍摄，这些都使得内容视频中的人物形象存在不同程度的遮挡，人体区域的丢失，角度，姿态以及大小的变化。

我们可以通过如图 1 的一个具体的示例看下。在图 1（a）中，A 和 B 是用来描述同一场景的不同角度和机位的镜头。而镜头 A 中和镜头 B 中的人物匹配问题可以看做是一个针对全身人体和半身人体的匹配问题。在图 1（b）中，镜头 A 中的人物形象具有可识别的正脸，此时，如果我们能够匹配出镜头 B 中的同一人物，那么我们最终就能识别出那些在某些镜头中仅仅只是背面或者侧面的人物角色信息。这有助于我们提高视频内容分析指标的准确性和有效性。

(a) (b)

图 1　内容视频中的人物匹配

综上我们可以看出，内容视频中的 re-ID 算法要解决的一个主要问题就是遮挡或者部分人物形象的匹配，学术界通常称为 partial re-ID 或者 occluded re-ID。目前行业内对于这类问题的主要解决思路一种是通过将局部区域特征重建为全局特征来实现隐式的对齐操作，另一种则是利用人体空间区域划分，来实现相同区域特征的显式对齐操作。但是现有这些方法的问题一方面是训练数据集跟内容视频的数据差异较大，另一方面是粗粒度的空间区域划分并不能适应内容视频中复杂的人体姿态。因此，我们考虑从数据集和算法模型两个方面进行内容视频中的 re-ID 算法优化。

二、数据集构建

图 2 是开源数据集和剧集视频数据的一个对比，从图中我们发现，剧集视频中的人物姿态更加多变，分辨率较高，并且伴随着大量的遮挡或人体部分丢失，而开源数据集中的人物姿态较为固定，较多是直立全身的图像。因此，在目前的开源数据集上训练的算法模型很难直接应用于内容视频之中。基于此，我们直接构建了一个剧集场景下的大型 re-ID 数据集。

图 2　数据集对比

整个数据集的构建流程主要分为以下几步：

1. 筛选剧集：根据时间和热度选取大约 500 部剧集，每部选择 10-20 集
2. 抽帧检测：对每集视频抽帧检测，利用人体检测模型获取人体框图片。
3. 数据分组：对检测好的人体框图片，按照 show 进行分组，以减少标注工作量。
4. 数据标注：对分组的数据集进行人物 ID 的标注，每个 ID 大约标注 30 张左右。

最终得到我们剧集场景下的 Drama-ReID 数据集，整个数据集大约有 1W 的 ID 数，包含 38W 左右的图片数量。是目前行业内最大的剧集场景的 re-ID 数据集。

三、算法模型

我们的整体算法模型框架如图 3 所示。基础网络部分使用预训练的 resnet50，为了获得更大的特征图，我们会将 backbone 的最后一个卷积层的 stride 设置为 1。在 backbone 之后有三个主要的模块：人体语义分割，信息熵度量模块，语义对齐匹

配。整个网络结构是 end-to-end 训练。

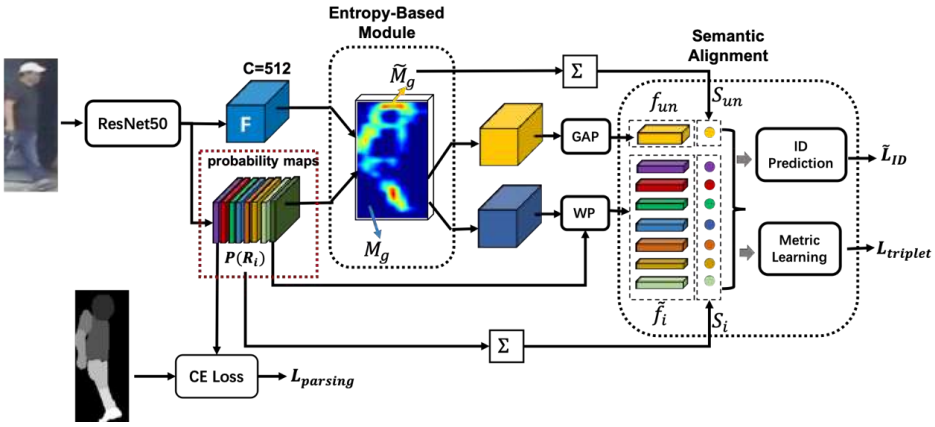


图 3　网络结构

1. 人体语义分割

不同于现有的一些算法那样对人体区域进行一个空间区域上的划分，我们利用人体语义分割进行人体语义区域的划分，如图 4 所示：

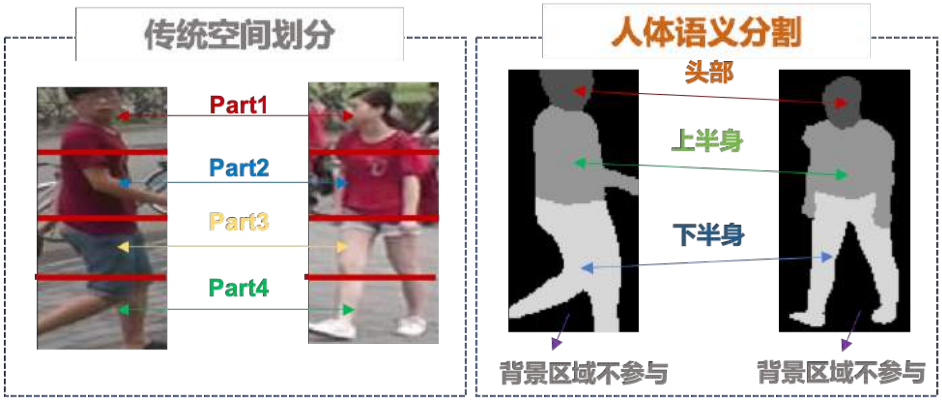


图 4　人体语义分割

一方面，我们可以利用人体语义区域划分来实现语义级别的特征对齐，另一方

面，我们可以去掉背景区域特征，防止部分复杂背景的图像对人物匹配造成影响。同时，我们也并没有像现有一些算法中那样利用一个单独的语义分割模型来提取人体语义区域，而是使用一个多任务学习的人体语义分割，它的好处一是可以减少模型的复杂度和计算量，另一个则是通过增加语义分割的监督 loss，可以有效提高基础特征的空间表征能力。

2. 信息熵度量模块

多任务的人体语义分割可以帮助我们提取人体语义区域，但同时我们也需要考虑语义分割错误的情况。错误的语义分割会带来错误的特征对齐，从而导致人物匹配错误。考虑到如果某个区域的分割概率越高，这个区域被分割正确的可能性就越高，我们通过计算分割概率的信息熵来度量这种分割的不确定性。计算公式如下：

$$E_g = -\sum_{i=0}^{P} p(R_i|g) log p(R_i|g)$$

如果越小，表示模型对于这部分区域分割正确的把握就越高。通过这种方式，我们可以计算出特征图上每个点的信息熵，然后通过设定一个合适的信息熵阈值，我们将整个人体区域划分为了高熵和低熵区域。

整个的信息熵度量模块如图 5 所示：

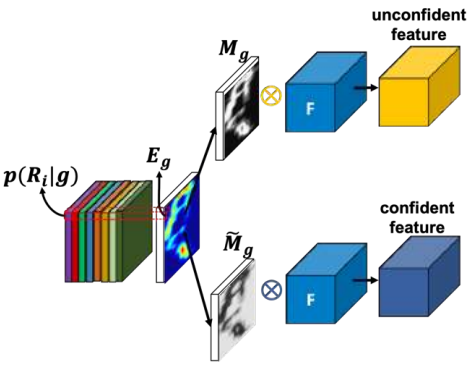


图 5　信息熵度量

高熵区域是语义分割不确定性高的部分，它们可能无法被正确的划分到某个的语义区域上，我们直接提取它们的全局特征。而往往，同一个人物的高熵部分具有某种独特性（例如某个人物的特殊的帽子），这种独特少有的元素很难被分割正确，但它却是我们进行人物匹配的一个重要依据，而我们的高熵区域的全局特征就是在表征这类人物所独有的元素。

低熵区域是语义分割不确定性低的部分，它们往往是能够轻易分割正确的部分，往往跟人体的区域结构具有强关联性。我们通过一个 entropy attention map 来增强语义分割中确定性高的表征，而抑制语义分割中确定性低的表征。一方面增强了稳定的语义部件的表达，另一方面减少了错误对齐的可能。

同时，在模型训练过程中，整体的信息熵会随着分割 loss 的降低而降低。在训练的初始阶段，模型不能很好进行语义分割，导致信息熵都偏高，大部分的区域都会被划分到高熵区域，此时人物比对的特征以全局特征为主。随着训练的进行，模型的语义分割能力会提高，信息熵会降低，大部分的区域都会被划分到低熵区域，此时人物比对的特征以语义特征为主。这像是模型在特征选择上的一种自我对抗学习，在训练的过程中动态的选择高熵的全局特征和低熵的局部特征。

3. 语义对齐匹配

在获得了高熵的全局特征和低熵的语义特征之后，我们会首先根据下面的公式计算其各自的重要性分数，对于高熵区域来说，如果它所包含的人体区域特征越多，它的重要性分数会越高。而对于低熵区域来说，如果某个人体语义区域不可见，被遮挡或者不显著，则它的重要性分数会低。而语义对齐匹配的如下图所示：

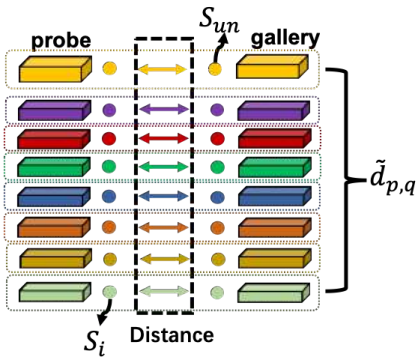


图 6　语义对齐匹配

根据各自的重要性分数和其对应的特征比对距离，可以计算最终人物匹配的整体距离。

$$d_{p,q} = \frac{\sum_{i=1}^{N-1} S_i^p \cdot S_i^q \cdot D(f_i^p, f_i^q) + S_{un}^p \cdot S_{un}^q \cdot D(f_{un}^p, f_{un}^q)}{\sum_{i=1}^{N-1} S_i^p \cdot S_i^q + S_{un}^p \cdot S_{un}^q}$$

此时，我们就实现了人体区域的动态对齐匹配。

四、结果分析

我们在开源数据集上进行了我们的模型一些 SOTA 模型的效果对比，结果如下：

Table 3. Results on Market-1501 and DukeMTMC-reID

Method	Market-1501		DukeMTMC-reID	
	rank1	mAP	rank1	mAP
PCB(ECCV2018)[22]	93.8	81.6	83.3	69.2
SPREID(CVPR2018)([10]	92.5	81.3	84.4	71.0
HPM(AAAI2019)[3]	94.2	82.2	86.6	74.3
Auto-REID(ICCV2019)[16]	94.5	85.1	**88.5**	75.1
DSR(CVPR2018)[7]	83.5	64.2	-	-
SFR[8]	93.0	81.0	84.8	71.2
VPM(CVPR2019)[21]	93.0	80.8	83.6	73.6
PGFA(ICCV2019)[14]	92.4	77.3	82.6	65.5
STNReID[13]	93.8	84.9	-	-
Baseline	93.2	82.5	84.9	75.4
ESA-ReID(ours)	**94.5**	**86.3**	87.7	**77.6**

图 7　Holistic 数据集对比

Method	Partial-REID		Partial-ILIDS	
	rank1	rank3	rank1	rank3
MTRC[11]	23.7	27.3	17.7	26.1
AMC+SWM(ICCV2015)[27]	37.3	46.0	21.0	32.8
DSR(CVPR2018)[7]	50.7	70.0	58.8	67.2
SFR[8]	56.9	78.5	63.9	74.8
STNReID[13]	66.7	80.3	54.6	71.3
VPM(CVPR2019)[21]	67.7	81.9	65.5	74.5
PGFA(ICCV2019)[14]	68.0	80.0	69.1	80.9
Baseline	55.0	76.1	56.2	66.3
ESA-ReID(ours)	**78.6**	**84.3**	**73.6**	**82.4**

图 8　Partial 数据集对比

图 7 是全身数据集上的对比结果，图 8 是 Partial 数据集的对比结果。

同时我们也在构建的 Drama-ReID 数据集上进行了测试对比，结果如下：

Table 1. Results on Drama-ReID dataset

Method	Rank1	mAP	PR-AUC
DSR(CVPR2018)[7]	82.5	73.0	0.47
SFR[8]	86.9	76.3	0.60
PCB(ECCV2018)[22]	93.8	84.5	0.79
VPM(CVPR2019)[21]	94.5	86.7	0.81
Baseline	93.2	83.6	0.78
ESA-ReID(ours)	**96.7**	**91.3**	**0.86**

图 9　Drama 数据集对比

五、成片体检应用案例

通过加入 Partial re-ID 特征我们可以获取更加准全的视频人物数据，这些数据目前主要应用于成片体检中各项指标的计算，例如人物出镜，人物交互，故事线等，同时我们可以根据以上这些指标对视频内容进行量化的剪辑优化或者内容评估。以下是《冰糖炖雪梨》的一些 case 应用。

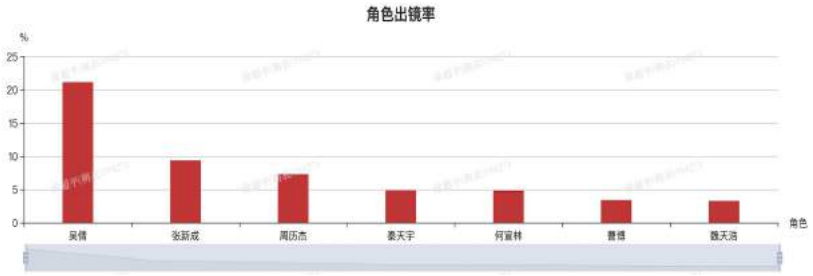


图 10　角色出镜率

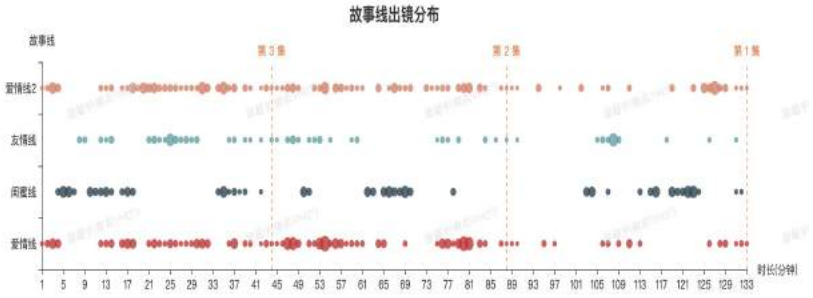


图 11　故事线分布

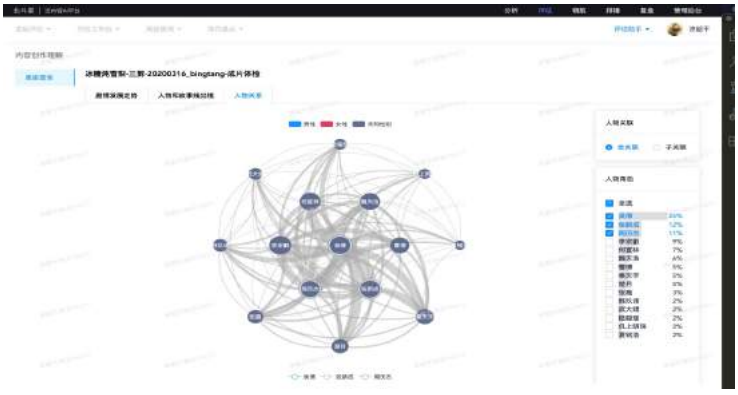


图 12　角色社交网络关系

文娱内容流量管理的关键技术——多任务保量优化算法实践

作者 | 阿里文娱高级算法工程师　雷航

本文章已被 KDD2020 录用：

Hang Lei, Yin Zhao, and Longjun Cai. 2020. Multi-objective Optimization for Guaranteed Delivery in Video Service Platform. In Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'20), August 23–27, 2020, Virtual Event, CA, USA. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403352

一、业务背景

保量策略对于视频内容来说，是一种很重要的投放策略。新热视频内容都需要增加自身的曝光资源来达到播放量最大化，而各场景（首页、频道页等）的总体资源有限且每个抽屉坑位的日曝光资源有限，因此各内容的曝光资源分配存在竞争问题。另外，不同场景之间相互独立，每个场景根据自身的目标进行效率和体验上的优化，但是场景与场景之间流量协同无法通过优化单一场景来完成。

图 1　剧集频道页和首页

为内容分配曝光量涉及到关于曝光和点击建模问题，以及内容的未来点击量预测问题。内容曝光、点击和播放等构成了一个复杂的非线性混沌系统，不仅取决于内容质量本身，也取决于内容更新时间、更新策略和用户点击习惯等。传统的统计预测模型无法阐述外部环境的各种干扰因素以及系统的混沌特性，即无法从机理上描述系统本质。针对此问题，我们首先通过分析新热内容的历史曝光点击日志，使用常微分方程建立了新热内容曝光敏感模型，即 pv-click-ctr 模型（简称 P2C 模型）。在 P2C 模型基础上，结合各场景和抽屉的曝光资源约束，给出一种曝光资源约束下的多目标优化保量框架与算法。

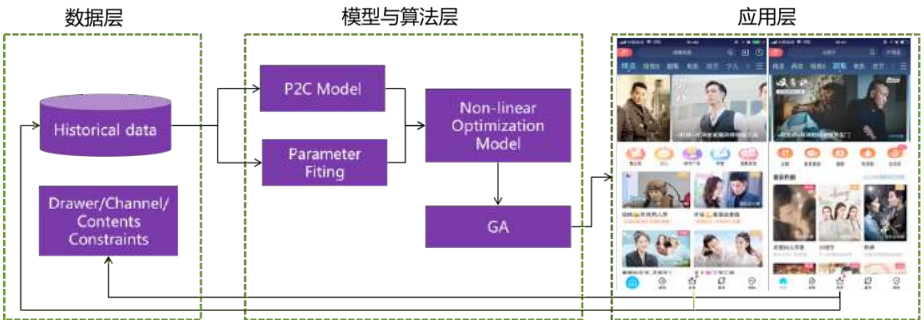


图 2　保量策略框架

二、内容曝光敏感度模型

通常情况下，点击 PV（click）随曝光 PV 增大而增大，即高曝光带来高点击。但是，内容消费者数量有限，给同一个消费者针对单一内容重复曝光并不会带来更多的点击量。这种点击“饱和”现象可从内容的历史曝光点击日志观察得到。受此现象启发，我们根据内容曝光 PV 和点击 PV 历史数据特点，建立一种能够描述内容点击量随曝光量变化趋势的常微分方程（Ordinary Differential Equation, ODE）模型，即 pv-click-ctr (P2C) 模型，整体结构如图 3 所示。

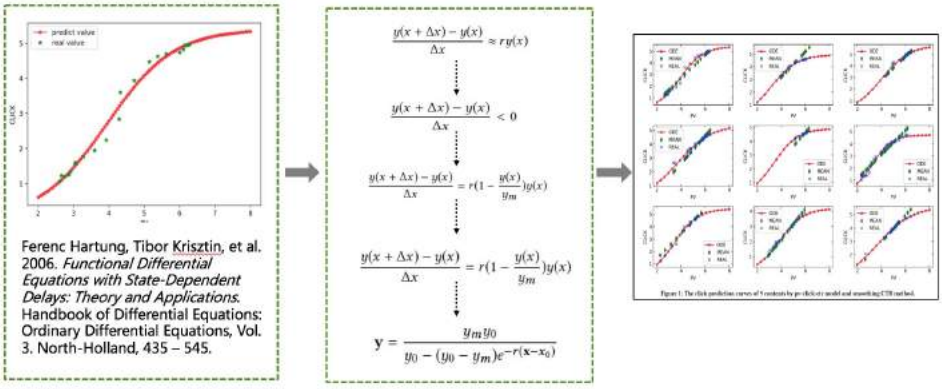


图 3　P2C 模型整体架构图

一个内容由于自身因素和外部环境的限制，对应的点击量存在最大值或饱和值 ym。当给定一个曝光量 x 时，存在唯一的点击量 y 和饱和度 k。对于一个点击量 y，饱和度 k 定义为当前点击量和饱和值的差距与饱和值的比值，即

$$k = \frac{y_m - y(x)}{y_m} \tag{1}$$

对于任意一个内容，随着 pv 的增大，click 饱和度减小，且单位 pv 带来的 click 增量（简称 click 增量）与当前 click 比值呈下降趋势。也就是说，click 增量与饱和度存在正相关关系，可用下式表示。

$$\frac{y(x+\Delta x) - y(x)}{\Delta x} = r(1 - \frac{y(x)}{y_m})y(x) \tag{2}$$

其中，r 为正相关系数。根据式 (2)，可以得到 click 随 pv 增长的常微分方程模型。

$$\frac{\mathbf{dy}}{\mathbf{dx}} = r(1 - \frac{\mathbf{y}}{y_m})\mathbf{y} \tag{3}$$

对式 (3) 分离变量后两端进行积分，可以得到

$$\mathbf{y} = \frac{y_m y_0}{y_0 - (y_0 - y_m)e^{-r(\mathbf{x}-x_0)}} \tag{4}$$

其中，x0 和 y0 分别为初始 pv 和 click。

对于式子 (4) 中的参数 r 和 ym，可采用最小二乘法拟合。这里首先需要对历史 pv 和 click 数据以及参数进行过滤和预处理。

（a）样本点过滤原则。分别在日历史 pv 和 click 数据序列选取最大递增子序列；

（b）参数预处理。由于点击量饱和值 ym 的数量级通常很大，而相关系数 r 数

量级通常很小，为了避免“大数吃小数”的现象，分别对这两个参数进行数据变换，即：ym → log10(ym)，r → er；

（c）样本点预处理。为了避免最小二乘法在拟合参数时陷入局部最优，分别对历史样本（click 值 y，pv 值 x）进行数据变换，即：x → log10(x)，y → log10(y)。经过参数拟合过程，可得到单一内容 pv-click 函数关系。进而可进行 pv-click-ctr 预测，这里可采用有限差分的数值解法预测，也可将数据点代入式子 (4) 预测。

三、保量模型 & 算法

基于上一节建立的 P2C 模型，本节任务是在各场景和抽屉曝光资源有限的情况下，给出每个内容近似最优的曝光量。整体方案流程如下图：

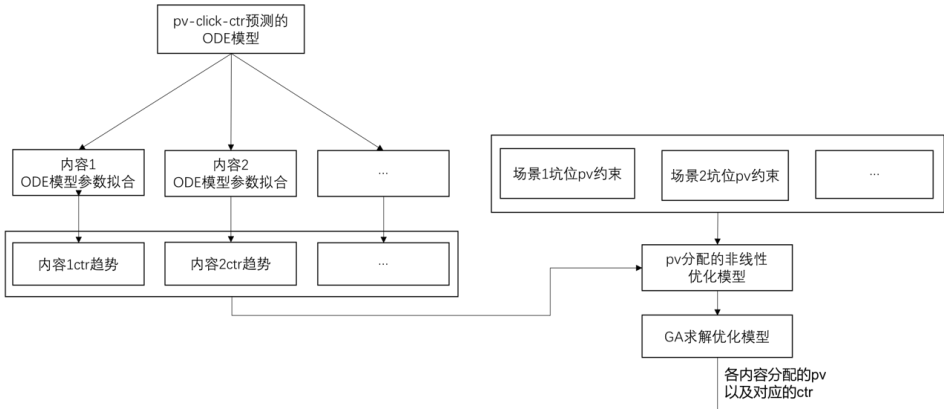


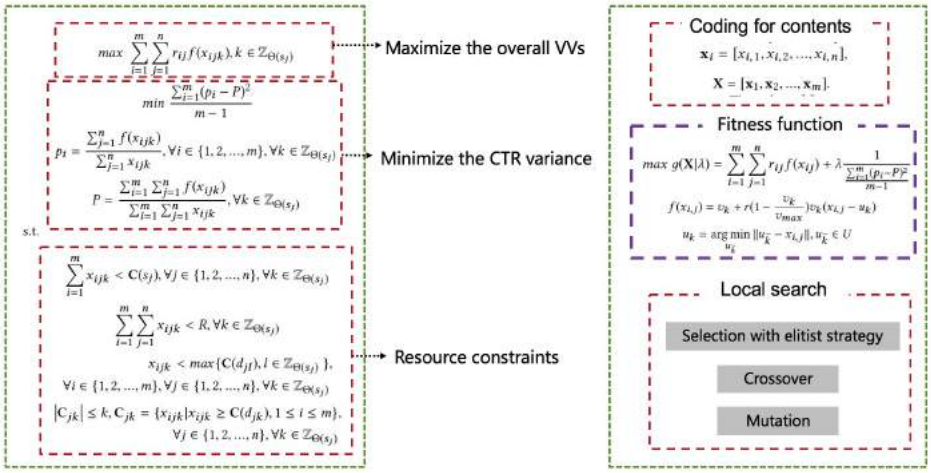
图 4　保量模型整体架构图

第一，基于 pv-click-ctr 预测的常微分方程 (ODE) 模型，针对内容池中每个内容，采用最小二乘拟合 ODE 中的两个参数：click 饱和值 ym 和 click 随 pv 的固有增长率 r。从而给出每个内容 pv-click 函数关系；

第二，基于给定的优化目标和约束条件，可建立 pv 分配的多目标非线性优化模型。在将业务问题抽象为数学模型之前，有必要对模型中的符号进行说明，如下所示：

表1: 保量模型符号说明

变量	说明
xijk	内容 i 在抽屉 j 中的坑位 k 分配到的 pv 量
R	所有抽屉的 pv 总量
rij	内容 i 在抽屉 j 中 click 与 vv 的正相关系数
C(Sj)	抽屉 j 对应的总曝光 pv
f(•)	P2C 模型预测函数



上述模型的优化目标包含两个：多场景 vv 最大化，内容池内容 ctr 方差最小。需要注意的是，这里的 ctr 方差最小是曝光公平的一种形式化描述，用以平衡“过曝光”和“欠曝光”。约束条件分别表示了场景、抽屉、坑位和内容的曝光 PV 约束。由于目标函数我们采用数值方法求解，使得上述优化模型无法运用传统的基于梯度的算法求解。而进化算法提供了一种解决方案，这里选取遗传算法 (GA) 求解。需要说明的是，GA 中的适应值函数计算采用了 P2C 模型。

四、实验结果

我们选取多个新热内容，分别给出 P2C 模型的预测效果以及保量模型的离线效果。这里的评估指标是均方根误差 (RMSE) 和绝对误差百分比 (APE)。分别采用 P2C 模型和平滑 ctr 方法 * 预测新热内容的点击量。从表中可以看出 P2C 模型可以

有效预测点击量，在 RMSE 方面优于平滑 ctr 方法。

* Xuerui Wang, Wei Li, Ying Cui, Ruofei Zhang, and Jianchang Mao. 2011. Click through rate estimation for rare events in online advertising. In Online multimedia advertising: Techniques and technologies. IGI Global, 1–12.

Table 3: Comparison between pv-click-ctr model and smoothing CTR method on online data

content index	♯ days	RMSE	
		pv-click-ctr model	smoothing CTR
1	15	0.0752	0.40144
2	19	0.13875	0.18118
3	27	0.12788	0.24273
4	10	0.04989	0.20967
5	32	0.15736	0.29446
6	16	0.11209	0.22749
7	24	0.13558	0.13774
8	21	0.10277	0.18685
9	15	0.04226	0.09936

Table 5: Comparison of online data and GA results

content index	vv			pv		
	REAL	GA	APE	REAL	GA	APE
1	50079	48123	3.907%	1575906	1475113	6.396%
2	28762	28077	2.382%	2190789	2257795	3.059%
3	18213	16139	11.388%	652690	620341	4.956%
4	56581	56727	0.258%	1457541	1327932	8.892%
5	38978	38492	1.246%	2379646	2152768	9.534%

线上实验部分，我们建立了分桶实验。基准桶采用人工策略保量；实验桶采用本文提出的策略，实验过程中关注和对比基准桶和实验桶每日投放效果（CTR 方差、策略在场景上的整体 CTR 等）。以下给出 30 天和 7 周的保量效果数据，与人工策略结果对比发现，保量策略在 CTR 方差和场景整体 CTR 方面均有不同程度的提升。特别地，在 CTR 方差方面，保量策略效果非常明显，平均相对提升 +50%。

Table 7: A/B test result during 30 days for optimization strategy and manual strategy

day index	var		CTR		
	manual	GA	manual	GA	%Lift
1	0.0333%	0.0166%	2.35%	3.03%	+28.94%
2	0.0301%	0.0259%	2.26%	2.82%	+24.90%
3	0.0492%	0.0261%	2.70%	2.88%	+6.82%
4	0.0544%	0.0432%	2.69%	3.48%	+29.33%
5	0.0347%	0.0201%	2.52%	2.96%	+17.74%
6	0.0447%	0.0238%	2.35%	2.99%	+27.53%
7	0.0369%	0.0198%	2.50%	5.35%	+114.43%
8	0.0423%	0.0266%	2.61%	3.32%	+26.91%
9	0.0439%	0.0332%	2.63%	4.36%	+65.70%
10	0.0570%	0.0469%	2.96%	4.09%	+38.14%
11	0.0604%	0.0575%	2.72%	3.32%	+22.32%
12	0.0669%	0.0154%	2.69%	2.82%	+4.98%
13	0.0319%	0.0058%	2.00%	3.12%	+56.18%
14	0.0593%	0.0073%	2.50%	2.58%	+3.50%
15	0.0566%	0.0031%	2.14%	2.54%	+18.70%
16	0.0636%	0.0269%	2.72%	2.91%	+7.27%
17	0.0564%	0.0015%	2.58%	2.82%	+9.12%
18	0.0460%	0.0025%	2.35%	2.65%	+13.01%
19	0.0212%	0.0067%	1.72%	2.47%	+43.92%
20	0.0433%	0.0173%	2.31%	2.40%	+3.90%
21	0.0637%	0.0458%	2.10%	2.80%	+33.35%
22	0.0365%	0.0182%	2.13%	2.68%	+26.13%
23	0.0584%	0.0168%	2.61%	2.73%	+4.40%
24	0.0428%	0.0261%	2.20%	2.92%	+32.70%
25	0.0761%	0.0585%	2.97%	3.44%	+15.53%
26	0.0529%	0.0361%	2.58%	4.80%	+85.80%
27	0.0578%	0.0531%	2.62%	4.39%	+67.42%
28	0.0800%	0.0300%	3.10%	4.63%	+49.65%
29	0.0552%	0.0264%	2.57%	4.62%	+79.41%
30	0.0737%	0.0214%	2.95%	3.82%	+29.60%

Table 8: A/B test result during 7 weeks for optimization strategy and manual strategy

week index	var (%Reduce)	CTR (%Lift)
1	+79.21%	+13.96%
2	+74.80%	+20.93%
3	+78.93%	+45.23%
4	+52.14%	+18.41%
5	+66.98%	+39.29%
6	+22.63%	+60.59%
7	+33.60%	+51.59%
Average	+58.33%	+35.72%

五、总结 & 展望

内容保量策略旨在解决流量资源有限与需求过多之间的矛盾，为各内容提供一种优化的曝光量建议，使得各场景的曝光资源能够产生更大价值。本文针对新热内容的多场景 VV 保量需求，提出了一种资源约束下的保量模型和算法框架，此框架整体由预测和优化两阶段构成。我们在部分场景进行了离线测试及分桶实验，实验结果反映了本文策略的可行性和有效性。未来需要持续探索和完善的有很多方面，如 PUV 保量、保量冷启动问题等。

数据如何反哺宣推与制作？面向文娱的舆情挖掘实践

作者 | 阿里文娱算法专家　李穆

一、背景

近年来，随着文娱产业的不断发展，剧本、弹幕、影评等内容相关的文本数据得以沉淀下来，利用自然语言处理技术对这些文本数据进行挖掘与分析对用户理解和内容理解有着重要的意义。阿里文娱北斗星平台（阿里文娱大脑）是一个泛内容大数据智能分析平台，基于多种形式的内容相关海量数据和 AI 技术，为内容评估、内容制作、市场宣推等环节的决策赋能。本文主要以面向内容行业的用户舆情挖掘、主创画像等文娱相关典型场景为切入点，介绍舆情挖掘技术在北斗星平台中的应用情况。

二、用户舆情挖掘

通常在内容播出后，用户往往会在站内和站外平台以弹幕和评论等方式对内容进行评价和反馈，这些用户舆情直接反映了用户对内容的喜好程度，挖掘用户舆情中的观点和用户的褒贬倾向对内容的评估、宣推等多个环节有着巨大的价值。北斗星对用户舆情的挖掘主要包括句子级别情感分析、细粒度情感分析、观点聚合三个层次，下面分别展开介绍。

1. 句子级别的情感分析

情感分析是舆情分析领域中常用的方法之一，传统的短文本情感分析通过训练句子级别的分类模型，针对用户的每条评论返回用户的情感极性，例如针对一条用户评论“这部剧太好看了!”模型返回“正向”。最终通过对所有用户评论返回的结果即可统计总体的正向、负向以及中性评论的比例，从而获得用户对当前内容的认可程度。近年来，以 BERT 为代表的动态语言模型已经在多个自然语言处理任务上刷新了纪

录，基于 BERT（Bidirectional Encoder Representation from Transformers）的短文本分类模型已经被证明效果要由于 textCNN、textRNN、fastText 等传统神经网络模型，因此我们选择使用基于 BERT 的分类模型来实现句子级别的情感分类。

在互联网环境下，用户对内容的表达方式多种多样，诸如“辣眼睛”“五毛特效”“给导演寄刀片”等新奇而形象的搭配层出不穷，这也给情感分析增加了难度，因此我们基于自己的文娱语料预训练了一个 BERT-base 模型，然后在精标的情感分类数据集上进行 finetune，比直接使用 google 的 BERT-base 要高 3% 左右，这也说明了使用垂直领域相关语料进行预训练的必要性。除此之外，基于大量的用户评论带有评分信息（1 星 -5 星），为了使 BERT 在预训练阶段捕捉的信息能够更加贴合舆情挖掘的场景，我们尝试将 BERT 预训练目标之一的 NSP（Next Sentence Prediction）替换为用户评论对应的评分预测，下游的舆情相关任务相较于直接使用 NSP 会有接近 2% 的提升。最终我们的句子级别情感分类在测试集上准确率达到 92%。为了满足实际应用，我们对训练出的 BERT 模型使用知识蒸馏（knowledge distillation）技术进行压缩以满足实际应用的性能需求，最终蒸馏后得到的模型准确率为 90.7%，从实用效果来看，蒸馏后的模型对文娱领域的大多数特有表达都具备了较好的适应性。

然而，句子级别的情感分类在实际应用中存在一定的局限性：

特效还是不错的，剧情就比较狗血了。

考虑上面的例子，这条评论中包含了两个评价对象（特效 和 剧情）且当前用户对它们的情感极性并不一致（特效 - 正向，剧情 - 负向），由于用户经常在一条评论中讨论内容的多个维度，这就使得句子级别的单一情感分类结果不足以刻画用户在一条评论中提及的多个评价对象。

2. 细粒度情感分析

针对句子级别情感分析的局限性，我们通过细粒度情感分析（aspect-based

sentiment analysis）来解决，具体问题定义如下：

给定一个句子 S，S 中包含若干评价对象（aspect），需要模型输出每一个评价对象在当前句子 S 上下文中对应的情感极性 y。

由于在实际应用场景下，往往评价对象不是事先定义好的封闭集合，所以首先需要把评价对象识别出来，然后再基于当前句子中的上下文针对每一个评价对象 A 进行分类。 这种两阶段 pipeline 的方式存在两个缺点，一是存在错误传递问题，即如果第一步评价对象识别错误就会影响第二步的结果；二是需要两个模型，存在一定的性能问题。因此我们使用联合标注策略将细粒度情感分析任务转化为序列标注任务，使用一个模型同时解决评价对象的识别与评价对象的情感分类两个目标。

具体来说，如上面的例子所示，给定一句评论，模型需要针对句子中的每个字输出其对应的 label，其中每个 label 对应的第一个字符表示当前这个字在评价对象（aspect）或评价词（opinion word）中的相对位置，B 表示起始位置，I 表示非起始位置，O 表示不属于评价对象或评价词，第二个字符为 A 或者 P，分别代表评价对象和评价词，第三个字符中，“+”表示正向，“−”表示负向。当模型完成 label sequence 的识别后，我们就可以基于每个位置对应的 label 将评价对象以及其对应的情感极性解析出来。

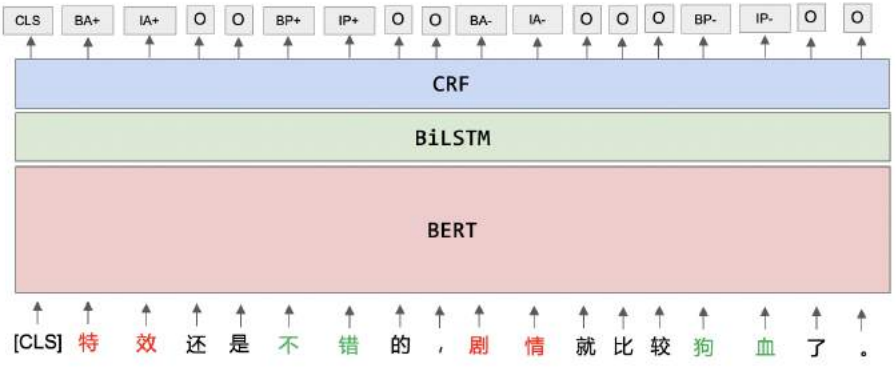


图 1　基于联合标注策略的细粒度情感分析

我们使用的序列标注模型如图 1 所示，首先最下层的输入是一条评论，使用 BERT 对其进行编码，BERT 上面是 Bi-LSTM 层，最上面使用 CRF 层对不同 label 之间的状态转移进行约束。我们还是使用基于文娱语料预训练的 BERT，由于 finetune 阶段和上层的 Bi-LSTM 和 CRF 对学习率的需求不一样，为了使 Bi-LSTM 和 CRF 能够充分学习，我们对它们使用比底层 BERT 更大的学习率（layer-wise learning rate），相比于设置相同的学习率会有接近 1% 的提升。

图 2　北斗星系统中的细粒度情感分析效果图

图 2 是细粒度情感分析在北斗星系统中的应用情况，左侧是《长安十二时辰》这部剧对应的用户舆情中属于“制作”这一类别的评价对象的情感极性聚合结果，右侧是属于“剧情”类别的聚合结果，可以发现用户普遍对“画面”、“片尾曲”、“镜头”等视觉音效相关的评价对象比较满意，而对剧情则负向反馈较多。

3. 观点聚合

通过细粒度情感分析可以聚合用户对各个评价对象的喜好程度，但是为了增加结果的可解释性，我们进一步实现了观点聚合功能。

观点定义为 < 评价对象，评价词 > 二元组，由于在细粒度情感分析的任务中，我们已经识别出了句子中所有的评价对象和评价词，因此我们可以将观点挖掘转化成判断评价对象与评价词是否匹配的二分类任务。还是以上面的评论为例，共计有下列

四个候选观点二元组，我们的模型需要识别出 < 特效，不错 > < 剧情，狗血 > 为正样本，而 < 特效，狗血 >< 剧情，不错 > 为负样本。

特 效 还 是 不 错 的 ， 剧 情 就 比 较 狗 血 了 。

候选观点二元组	是否为正确观点
特效 不错	√
特效 狗血	×
剧情 狗血	√
剧情 不错	×

我们将底层 BERT 表征得到的每个候选二元组的 encoder 向量拼接之后作为分类模型的输入进行二分类，识别为正样本的二元组被认为是一个有效观点。具体实现过程中，我们利用评价对象和评价词的情感极性一致性过滤掉了大量候选二元组，提高了效率，另外基于近义词字典对部分评价词含义接近的二元组进行合并（例如 < 演技 不错 > 与 < 演技 好 > 会被合并），最终将相同的观点合并后按照频率排序即可到用户的 top N 观点，这些用户观点是内容宣推的重要素材来源之一。

三、主创画像

由于主创与角色的契合程度与内容质量息息相关，因此主创画像对于内容评估和制作都有着巨大的价值。我们主要使用标签结合基础指标的形式构建主创的画像。其中基础指标主要包括历史作品表现、热度、商业价值等维度，由于北斗星已经构建了成熟的艺人榜单体系来量化主创的人气、历史作品表现等维度，主创画像的核心在于主创标签的挖掘，下面主要介绍主创标签挖掘方法。

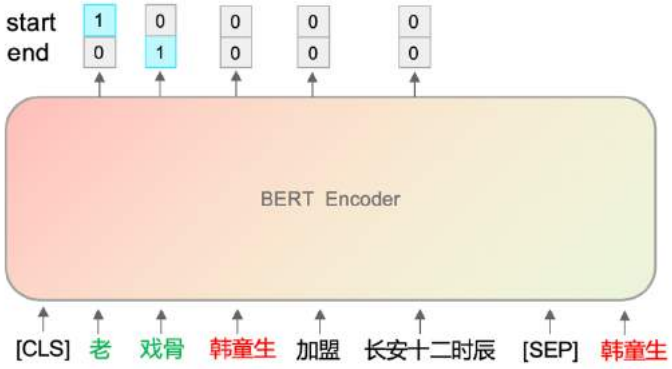


图 3　演员标签抽取模型

以演员为例，首先我们基于演员榜单去舆情库中匹配与当前演员相关的舆情，相应的，每条舆情中会包含当前的演员名，我们将舆情文本与当前演员名拼接成一个句子对作为输入，使用基于 BERT 的机器阅读理解（Machine Reading Comprehension）框架来识别舆情文本中描述当前演员的标签。标签可能是一个词也可能是多个词组成的短语，为了更好地识别标签的边界，我们使用片段（span）检测的的方式，每一个词对应的输出是两个二分类，分别判断当前的词是否为一个标签的开头（start）和结尾（end）。在 inference 阶段，如果 start label 和 end label 的距离小于某个阈值，则认为两者之间的 span 属于一个候选标签，在同一个距离窗口内，我们选择分类概率之和最大的 start label 与 end label 之间的 span 作为描述当前演员的一个标签。经过模型识别后，我们获得了大量 < 演员 , 标签 > 的二元组，针对每个演员，综合考虑该演员对应每个标签的频次和标签的稀缺性后可以得出描述该演员的 top N 标签，这些标签反映了演员的特质以及他们在网友心目中的主流印象。

目前基于标签的主创画像功能已经服务于北斗星平台的多个应用场景。下面简要介绍其在智能选角场景中的应用。

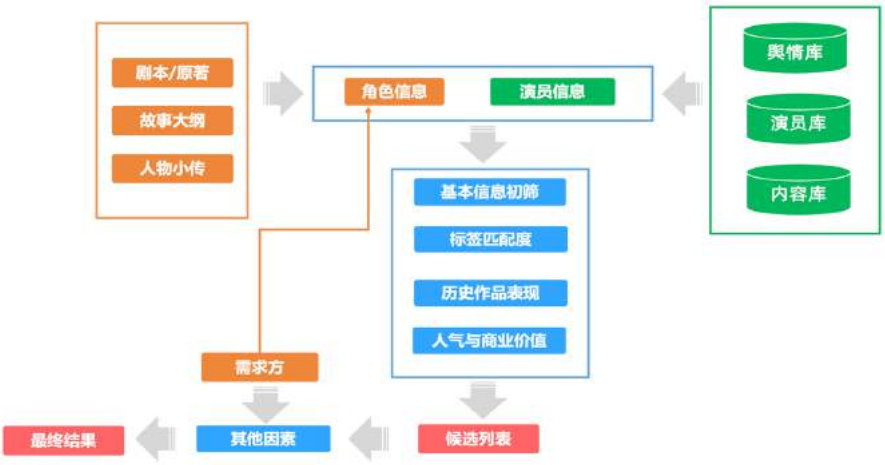


图 4　北斗星智能选角流程图

图 4 是北斗星智能选角的基本流程，主要可以分为以下几个步骤：

1. 首先是角色画像的建立，我们使用一系列标签来描述角色画像，角色标签主要有两个来源，一是业务方提供，二是利用标签识别模型从剧本、原著、故事大纲、人物小传等文本内容中抽取，角色标签抽取模型与主创标签识别模型基本一致。
2. 从演员库中初步筛选出符合基本条件的演员，基本条件通常包括性别、年龄、身高等。
3. 基于上面的标签识别模型从全网舆情挖掘每个演员身上的标签。
4. 基于演员与角色的标签匹配度、演员同题材历史作品表现以及当前时间窗口的人气等指标生成候选列表返回给业务方，最终由业务方综合考虑成本、演员档期等其他因素完成最终的选角。

北斗星的智能选角已经在《长安十二时辰》《白夜重生》等项目中得到了应用并取得了良好的效果。目前智能选角方案主要还是以辅助为主，基于大数据的结果更客观，数据的支撑使得每个维度都有可解释性，可以提供选角的视野更广，从全网的演员库中反馈的结果，可能比单个选角导演短时间内想到的候选人更全面些，避免遗漏。

四、总结与展望

内容宣发与制作是内容产业的重要环节，通过舆情挖掘相关技术获取用户反馈以及主创画像可以为内容宣发与制作提供有效的抓手，未来我们将继续应用自然语言处理相关技术在内容理解和用户理解等问题上持续进行探索和尝试。随着近年来以 BERT 为代表的的动态预训练语言模型的迅速发展，pretrain 加 finetune 已经成为了自然语言处理的新范式，如何针对文娱领域特有的典型应用场景设计预训练目标以及如何更好地利用知识图谱等先验知识将成为未来值得探索的方向。

媒资与素材管理

阿里文娱如何存储、管理泛内容数据?

作者 | 阿里文娱高级开发工程师　至德

用户在优酷或者其他互联网 App 上看到的文字、图片、视频等，都可以被称为内容，那么这些内容是如何被生产、管理和组织的？本文将简单介绍阿里文娱是如何利用网状关系组织泛内容，以及如何构建泛内容的网状关系。

一、泛内容存储管理的挑战

1. 数据规模大：在阿里文娱内，泛内容实体类型多，实体数据规模庞大，如何高效存储和管理；
2. 兼顾内容生产和分发：泛内容数据管理方式，既要提供快捷的内容生产模式，又能在分发链路上提效；
3. 扩展能力要求：泛内容实体类型会随着业务发展而不断增加，运营的新玩法也层出不穷，泛内容的存储管理方式，要能够适应和支撑未来的业务发展，必须要具有很强的业务扩展能力。

二、什么是泛内容的网状关系

标签是被广泛应用于内容管理的一种方式，内容生产链路上，我们会将内容通过机器学习算法或人工标注的方式打上相关标签。通过这些标签，将内容连接并组织成一张网。如图 1 所示，以视频为例，独立的视频通过标签被连接成了一张网络。

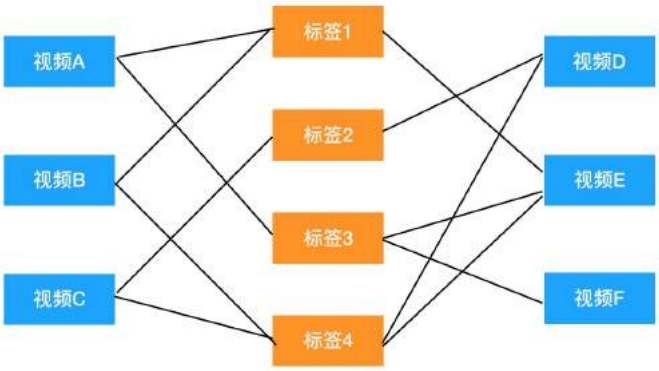


图 1　视频通过标签连接成网

有了内容的连接关系，内容的组织方式也有了更多样的玩法，运营同学也拥有了更丰富的运营工具进行内容分发。如图 2 所示，我们可以根据运营需求，将标签 1 升级为话题，标签 2 升级为榜单。运营便可以将视频 A、B、E 组织成为一个话题，将视频 C、D 组织成为一个榜单，用于前台运营活动。

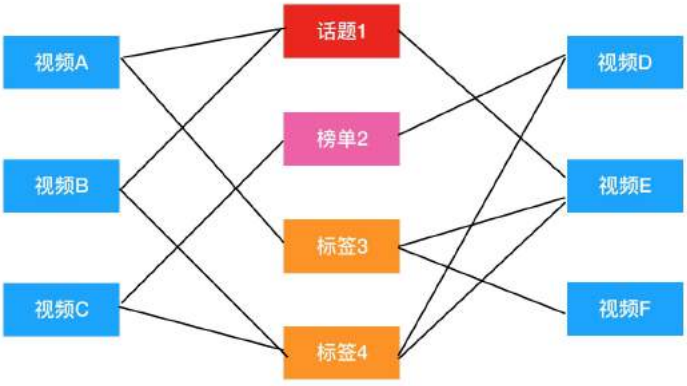


图 2　标签升级为话题和榜单

如图 3 所示，左图为沉浸式视频播放页，可以通过点击左下角话题标签跳转至右图的话题详情页，详情页列表中则通过 Feeds 流形式展现出该话题下所有视频。

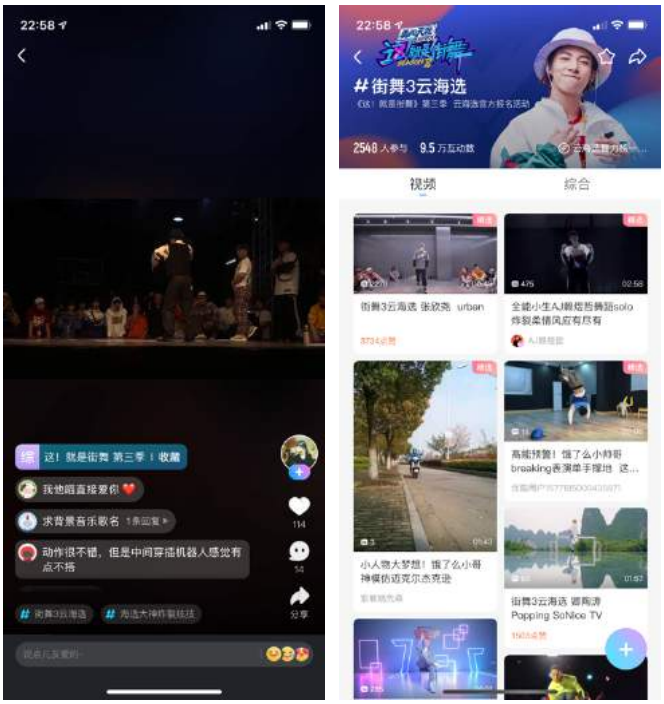


图 3　优酷内标签话题引导形式

泛内容网状关系背后的支撑技术是统一标签服务，其包括标签结构及打标结果的定义、基于媒资平台的核心标签服务、以及面向运营的标签管理工具和打标工具，下面的篇幅中将依次进行介绍。

三、标签结构及打标结果的设计

用户或运营在为内容打标时，为了提高打标效率，往往会先选择一个分类标签，然后再针对这个分类进行打标。传统设计中，标签结构往往被描述为一颗树，只能描述标签间的父子关系，例如图 4 所示的五层标签树。传统的标签结构在描述这种场景时，主要有两个问题：

1. 前三层的标签是父子关系，但是第四层和第三层之间是属性关系，两种关系应区别对待；

2. 第五层和第四层之间是属性值的关系，也应区别对待。

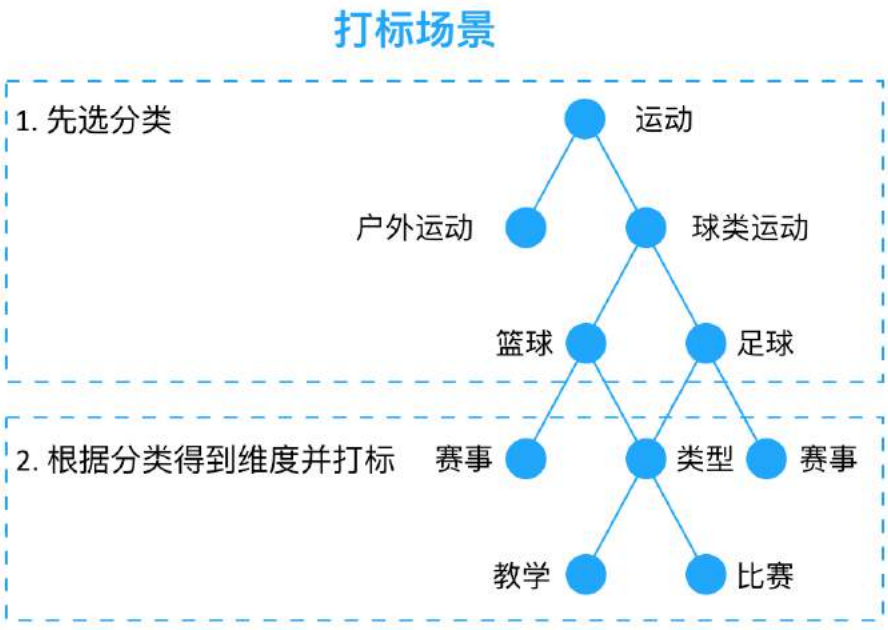


图 4 传统标签结构设计

统一标签服务的标签结构对上述两种关系进行了抽象，如图 5 所示。

首先，引入了标签类型的概念，每一个标签类型是个森林，例如图中紫色方框标表示的分类、赛事、技巧标签类型。对于简单的标签场景，例如内容标签、质量标签等场景，只需要使用标签类型即可支持场景。

其次，对于复杂场景，引入了子标签类型和标签分组的概念，图中橙色箭头表示运动分类标签关联了赛事和技巧两个子标签类型，当一个内容被标记为运动或其子分类时，这些内容都可以继续打赛事和技巧两类标签。图中绿色方框表示用篮球、足球对赛事和技巧类型的标签进行了可重叠的分组。

这一套标签结构表述能力十分强大，支撑了目前泛内容的所有标签。

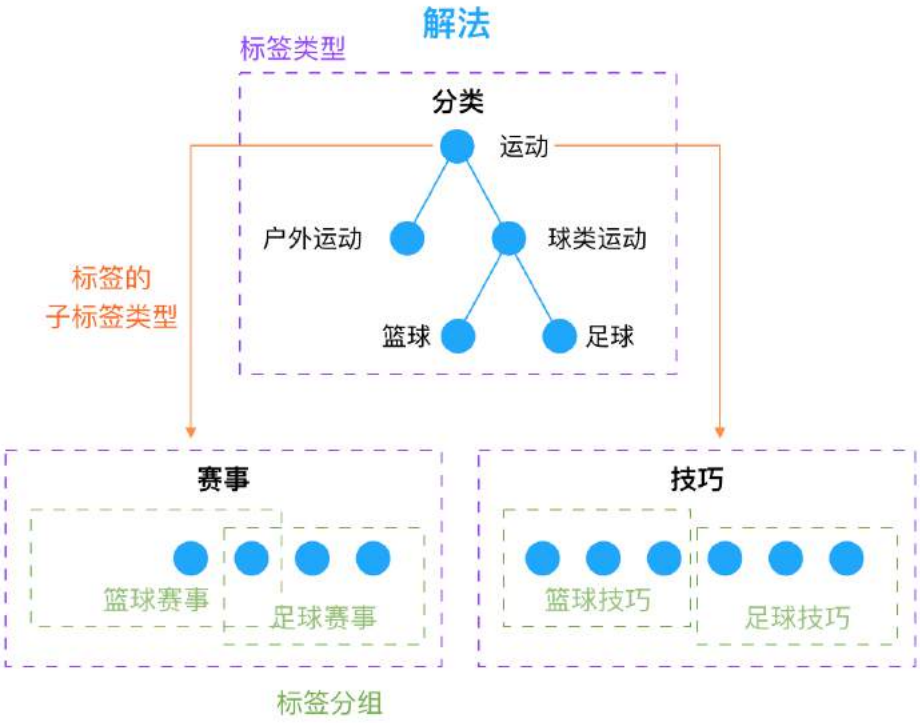


图 5　统一标签服务的标签结构

上述标签结构描述的标签为枚举型标签，除枚举型外，打标结果中还支持保存开放型标签和关联型标签。其他标签类型的方案设计暂且不在本文赘述。

四、核心标签服务

所有的原始数据，标签库和打标结果，都保存在媒资平台，从理论上说，媒资平台已经具备了内容的打标能力。然而这个读写能力较为原始，业务方使用起来多有不便，所以需要一个系统对原始能力针对标签业务进行业务封装，为业务方提供好用的标签服务，这就是核心标签服务，如图 6 所示。

其主要功能包括，对标签按路径进行展开，支持按照标签来源、按照标签类型、按照操作来源进行打标，提供了打标结果横竖转换的能力，同时还具备权限控制和流

量控制能力。

为了支持运营和算法不断地对标签体系进行迭代，核心标签服务中还提供标签体系的 ABTest 能力。

图 6　核心标签服务

五、标签管理工具和打标工具

标签管理工具基于媒资内容管理平台搭建，为运营提供标签体系的查看、审核、新增、修改、下线、删除、批量导入导出等功能。

由于打标流程中涉及机审、人审不同的打标方式，还涉及质量类标签、安全类标签、业务类标签等多种标签类型的打标需求，业务上需要使用打标工作流系统实现流程编排，如图 7 举例所示 (仅为流程编排示意，非实际业务流程)。通过打标工作流系统，可以按照业务需求实现复杂的打标流程管理。

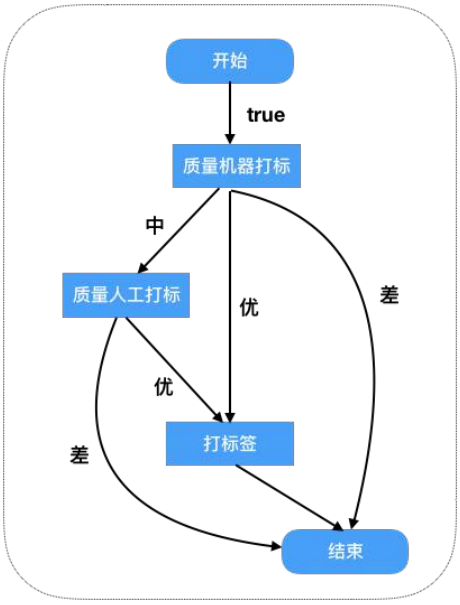


图 7　举例打标流程 (非实际业务流程)

六、小结和展望

目前，泛内容的网状关系有效组织和管理了阿里文娱内海量的长短视频、节目、人物、角色等泛内容实体，在生产和分发侧，都便捷的支撑了算法和运营需求，并且不断的为运营同学扩展提供了话题、榜片单等多种运营工具用于不同的内容分发场景，产生了极大的业务价值。

未来，泛内容的网状关系也将在数据和算法方面进行深度挖掘，通过算法推动网状关系演进，并且在算法打标、算法推荐等场景，利用网状关系提升算法效果，最终更好地服务用户。

内容分发提效

让需求开发飞起来！阿里文娱 FaaS 平台的实现与落地

作者 | 阿里文娱高级研发工程师 墨洵 阿里文娱研发工程师　武升

函数即服务（FaaS）作为云计算 2.0 时代重要的发展方向，能够从工程效率、可靠性、性能、成本等方面给开发者带来巨大的价值，尤其是能够极大的提升研发效率。因此，拥抱 FaaS 成为开发者关心的重要技术领域。本文将介绍阿里文娱的函数计算平台的设计思想与关键技术难点，并结合文娱业务介绍函数计算的落地实践经验。

一、背景

优酷内容分发业务涵盖了优酷主客的首页、频道页、二级页等不同场景下的内容分发，服务端之前采用传统的 Java 应用结合阿里集团中间件的开发模式，一直是产品评审、API 设计、前后端联调、前后端发版等节奏。然而，随着端上内容的多样化，产品需求迭代的加速，传统的服务端架构开发模式已显得力不从心，我们虽然沉淀出一套通用框架，但受限于开发模式的本质并没有变化，业务开发的灵活性与开发成本依然很高。总结起来，面临的挑战主要是：API 依赖数据源多，业务需求变化快，前后端联调成本大等。

随着 Serverless 技术的发展，FaaS 的相关实践探索都在阿里内部逐渐多起来，我们思考了 FaaS 的特点和面临的挑战，希望通过 FaaS 技术的引入，把一系列基础能力沉淀下来，在此之上，通过 FaaS 来承接上层业务逻辑，阿里巴巴文娱优酷 FaaS 平台应运而生。

二、平台设计与技术难点

1. 设计目标

希望实现一个通用的函数计算平台，在这个平台上，开发者直接通过编写、运行和管理一个或多个函数对外提供服务，允许通过微服务、HTTP 接口、事件源触发等多种方式调用函数。同时，函数的开发及发布应该是秒级生效，且无需重启宿主应用的，这样就可以克服传统 Java 应用发布部署的时间成本，极大的减轻开发者在代码开发之外的时间成本，同时可以快速回滚。

FaaS 平台应该提供函数式应用的运行环境，应该支持轻量级脚本语言编写函数。我们首选 Groovy 语言主要是考虑了 Groovy 的代码简洁，同时可以访问 Java 的原生的类和对象。

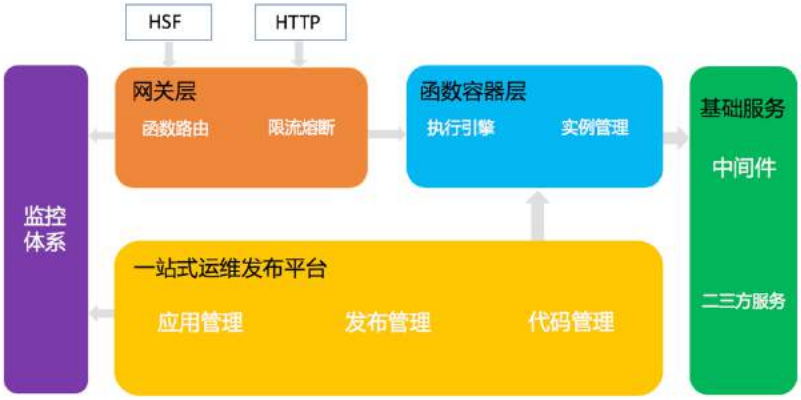
FaaS 可以根据实际的访问情况进行函数实例的动态加载和资源分配。

总结起来，在 FaaS 平台上运行的函数应该是一个短小、离散、可复用的代码块，我们希望它有以下几个特点：

1）生命周期短，支持快速发布部署
2）非守护进程（不需要长时间运行，按需加载）
3）不提供长连接服务
4）无状态
5）可重用现有服务或第三方资源（重点，FaaS 应该建立在完善的基础服务上）
6）毫秒级执行时间

2. 平台整体设计

FaaS 平台的整体核心架构主要由网关、运行时容器、一站式运维发布平台、基础服务等组成：



网关层主要负责接受函数调用请求，通过函数的唯一标识及函数的集群信息分发函数调用到对应集群的机器环境中执行。

函数容器层是整个系统的核心，主要通过函数执行引擎进行实例的调用执行，同时负责函数实例的生命周期管理，包括按需加载、代码预热、实例卸载回收等工作。

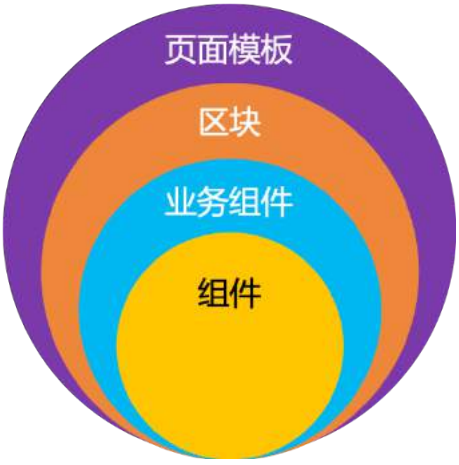
一站式发布运维平台（FaaS Platform）是面向开发者的主要操作平台，开发者在平台上进行函数编写、版本提交发布、回滚、监控运维等一系列工作。整个监控体系打通了集团的基础服务监控体系，，可以提供实时大盘，集群性能等基本监控指标的查询功能。

整个 FaaS 平台建立在集团中间件以及优酷内容分发依赖的各基础服务之上，通过良好的封装向开发者提供简洁的服务调用方式，同时函数本身的执行都是运行在互相隔离的环境中，通过统一的函数实例管理，进行函数的调度、执行监控、动态管理等。

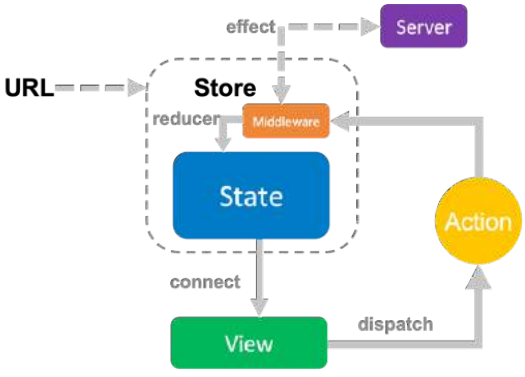
整体技术栈服务端容器层主要是采用 Java 实现，结合集团中间件完成整个容器层的主要功能。

前端主要基于 React 框架和 Dva 状态管理框架实现，当然，在实际开发过程中我们选择了蚂蚁金服的 Bigfish 框架和 Odin 脚手架。React 提供了组件化的概念，

这意味着我们开发的组件可以像 HTML 基本 DOM 元素一样不断被复用。为了实现组件的复用化和研发效率的提升，Bigfish 在 Web 页面上进行了分层设计，细粒度从大到小依次为页面模板 -> 区块 -> 业务组件 -> 组件。Odin 脚手架是优酷推出一款面向中后台业务系统的前端开发脚手架，集成了 Bigfish 的框架，支持以配置化的方式构建网站路由，使得开发者不需要关注过多底层细节，可以快速上手实现业务逻辑和页面构建。

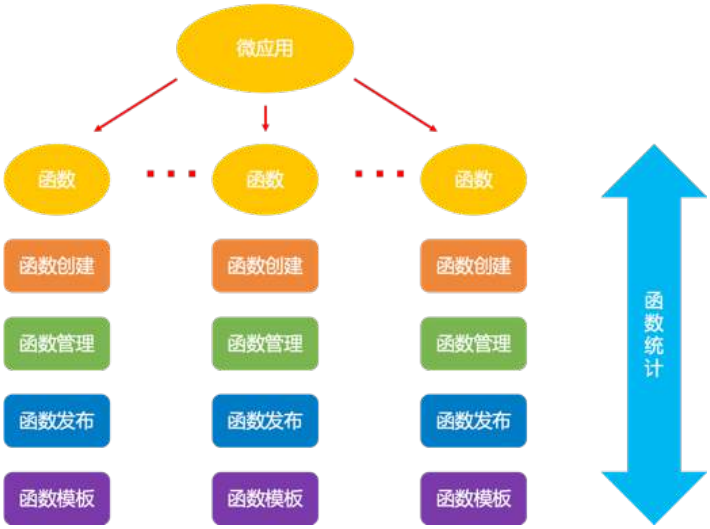


类似于服务端侧的 MVC 分层模式，前端在实现业务逻辑和数据通信时也有对应的封层设计模式，来实现组件的状态管理。经历了从 Flux -> Redux -> Dva 的衍变，状态管理机制对复杂业务带来的益处正在不变突出。Dva 的完整数据流图如下：



State 是负责保存整个应用状态，View 是 React 组件构成的视图层，Action 是描述事件的对象。connect 方法是绑定 State 到 View 的函数，使得 View 层的组件可以动态监听 State 中的属性，同时可以通过 dispatch 方法负责将 Action 发送至 State 触发状态改变。触发状态改变有两种类型的函数：effect 函数和 reducer 函数。前者会与服务端进行数据通信，可以处理异步动作；后者处理同步动作，并直接更新 State。

FaaS Platform 前端主要分为函数创建、函数管理、函数发布、函数模板和应用统计五个模块。在 FaaS Platform 系统中，函数是对外可被调度的最小单元，而应用是划分机器资源的最小单位，所以我们设定应用与函数存在一对多的映射关系。



1）函数创建模块

函数创建模块主要提供添加函数的功能。一个完整函数必须包括函数名称、函数标识、函数类型、函数所属应用及应用下所属分类等基本信息；同时类似于 mtop 网关，我们提供对于函数入参、响应业务结果、响应业务错误码的配置页面，用于自动生成函数调用入参表单和函数接口文档。函数的英文标识唯一确定一个函数，不可重复。

2）函数管理模块

函数管理模块主要提供函数的 CRUD 操作和函数的在线编写功能。在本页面我们可以快速进行复杂条件的函数查询和函数基本信息和状态的编辑。同时我们提供函数编写的在线 Web IDE，支持文件增删、代码编写、自动保存、函数提交、函数调试、日志打印等功能。

3）函数发布模块

函数发布模块主要提供函数提交历史的查询和执行函数发布的功能。我们像传统 Java 应用支持引入二三方依赖，但不同于传统的 Java 应用发布，FaaS Platform 系统中的函数发布可以实现秒级发布。目前函数发布已经支持函数回滚发布和函数分批次发布，从部署环节实现对复杂多变业务需求的快速响应。

4）函数模板模块

函数模板模块主要提供函数模板的 CRUD 操作和函数的在线编写功能。结合实际的业务场景，我们首先提供一些基础的内置模板，方便函数的快速初始化。同时对于某一个业务问题的完整解决方案，我们允许该函数保存为自定义的函数模板。函数模板的 Web IDE 同样支持函数模板的在线编写、调试、自动保存等功能。

5）应用统计模块

由于函数隶属于应用从而具备机器资源，我们计划提供应用统计模块以应用为拆分进行函数上线状态、发布版本的数据统计；同时我们也基于函数日志提供函数调用情况（调用量、成功率、响应时间）的统计分析和监控。关于具备的细节，我们正在逐步实现和完善。

3. 主要特性

优酷 FaaS 平台的主要特性是开发接入低成本、函数运行时环境隔离以及运维监控操作的透明化。

1）开发接入低成本

FaaS 平台通过一站式的云端开发平台，使用户可以直接面向业务逻辑的开发，而无需关注基础服务及中间件的依赖，平台本身提供完善的基础能力封装，包括：快捷开发能力，中间件快速接入能力，数据存储快速接入能力，基础能力封装直接调用等。

业务逻辑开发模式轻量化、无应用化，发布回滚秒级生效，极大的减轻了传统服务端开发过程的繁琐流程，将开发者的精力更多的集中于核心业务逻辑的开发。

同时提供如下的简洁易于操作的开发部署流程设计，减轻开发者开发部署的时间成本。

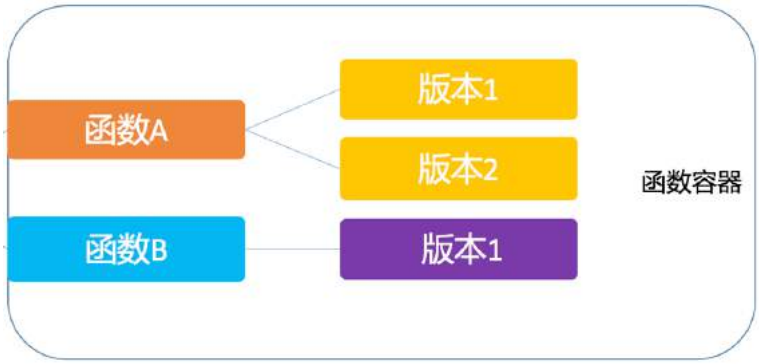
FaaS 平台上的函数除了开发成本低，调用者接入的方式也比较简单。我们同时提供了中心化和去中心化两种使用方式，不管去中心化还是中心化使用方式，函数代码的编写、调试、发布均在一站式运维发布平台上完成。在中心化接入方式下，我们通过统一的函数服务集群提供对外服务，允许调用者通过统一的函数调用接口以 HSF 服务或者 HTTP 接口调用函数，而函数代码的执行完全在我们的函数服务集群上，开发者无需自己申请应用。

对于去中心化接入方式，开发者如果想调用函数平台上的 FaaS 函数，可以引入我们提供的 SDK，此时，函数的执行完全在调用者应用的本地进程里，FaaS 平台只提供函数的开发发布功能。

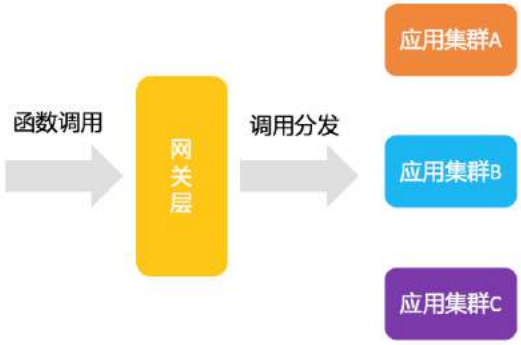
2) 运行时环境的隔离

运行时环境的隔离分为两个层次，一个层次是函数容器内部函数实例之间的隔离；另外一个层次是不同函数本身就运行在不同的虚拟应用集群上，集群与集群之间的隔离性。

函数容器内部函数实例的隔离指的是在 FaaS 平台上编写的 Groovy 函数运行在统一的 JVM 进程中，每个函数在开发的过程中都会生成多个版本，而不同函数之间、同一函数的不同版本之间在运行时的环境都是相互隔离，互不干扰的。



函数运行集群的隔离性主要是根据函数的访问量、函数的服务特点（长尾服务还是通用服务）等特性，在函数创建之初就将函数绑定在不同的虚拟应用上，而不同的应用会运行在不同的机器集群上，函数在被调用时，网关层可以根据函数的应用将函数的调用分发到不同的集群上执行，保证函数之间物理隔离。



3）运维监控的透明化

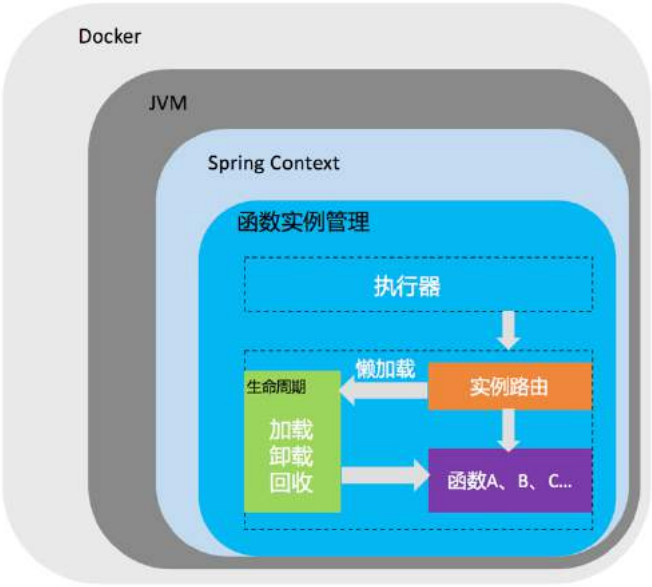
FaaS 平台的函数都能在平台上直接进行监控运维操作，我们通过在函数执行流程上收集函数的执行日志，并将日志实时上报到集团监控服务，可以在平台上实时监控函数运行。

4. 技术难点

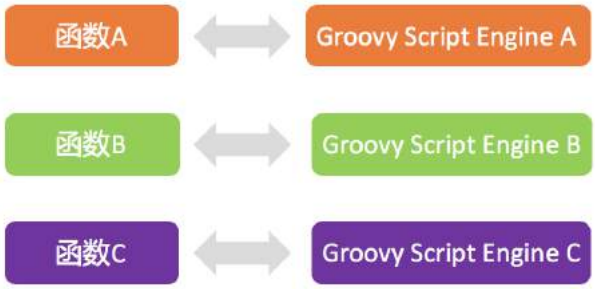
1）函数执行引擎设计

函数执行引擎是整个 FaaS 的核心部分，负责函数实例的加载、预热、调度执行、卸载等生命周期管理。FaaS 的函数目前支持 Groovy 语言，选择 Groovy 主要是由于 JVM 提供的运行时环境天然支持 Groovy 语言的运行。FaaS 平台上每个函数都具有一个自己独立的代码版本库，每次提交都将生成递增的版本，执行引擎加载函数实例时会从版本库中加载当前最新版本的代码，通过初始化、预编译等操作生成函数的实例放到实例池中，由于每个函数都有唯一标识，因此，当调用某个具体的函数时，执行引擎会从实例池中取出对应实例加载执行。整个流程如下图所示：

由于函数实例都存在于同一个 JVM 进程中，并且不同于服务，函数的粒度更小，因此函数的生命周期需要严格控制，不然大量函数加载到内存中，有可能出现内存占用过大的问题。同时兼顾 SDK 调用方式，防止多个函数常驻内存将宿主应用的内存耗尽。所以目前采用了懒加载机制，按需加载函数实例到内存中，过期自动回收，有助于释放内存提高内存利用率。



每个 Groovy 函数对应一个 Groovy 的解释器环境 GroovyEngine, 不同的函数之间相互独立，每个函数在加载到内存的过程中都分别独立的进行预编译，初始化等流程，防止不同函数之间相互干扰，同时为二三方 JAR 包加载提供隔离的环境，防止出现不同函数之间的类加载器相互影响的情况。

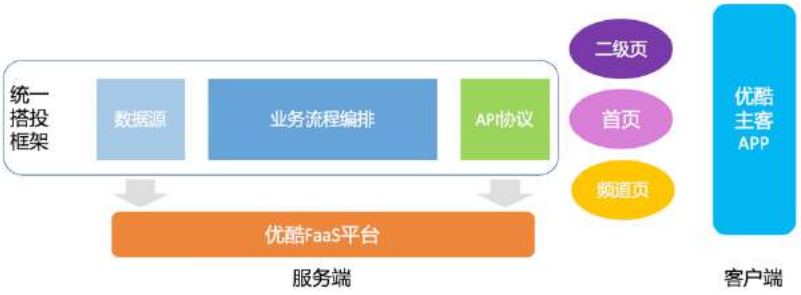


2）二三方 JAR 包加载能力

FaaS 平台提供二三方 JAR 包的加载能力，允许在不重启整个底层容器的情况下，加载函数自己的二三方依赖，我们通过实现 Groovy 二三方 JAR 包加载能力的 Classloader, 实现了函数与函数之间、函数不同版本之间的二三方依赖加载能力。FaaS 平台的 Classloader 体系：

三、FaaS 平台的落地探索

结合目前阿里文娱业务的特点，即大多以内容分发为主，以首页、二级页等业务来看，内容分发具有运营坑位多、需求变化快、数据源多等特点，传统的 Java 服务端开发方式，前后端联调以及后端开发部署都逐渐成了影响迭代效率的重要瓶颈，以往都是服务端开发在客户端发版前发布线上，发布耗时长，回滚成本高，因此通过引入 FaaS，希望提高服务端开发的灵活性，让开发者更多的面向业务逻辑而不是花较大量的时间在服务的部署维护上面。



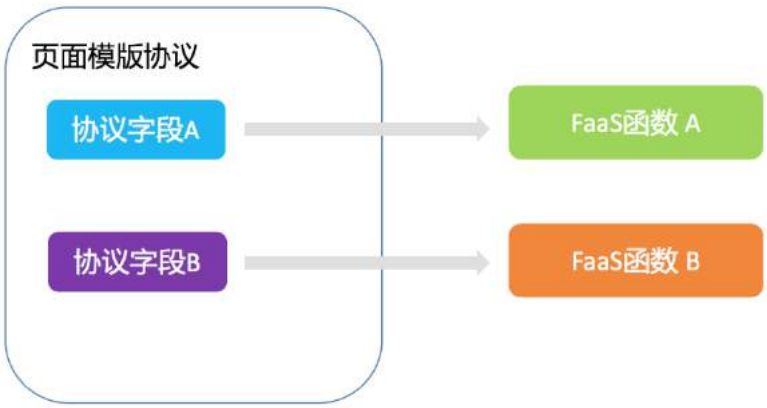
优酷内部的内容分发目前主要在统一的内容搭建投放框架之上开发，这套框架是一套流程编排的框架，通过流程编排，从不同数据源获取内容，通过业务逻辑处理，最终通过模版字段映射输出 API 内容。目前 FaaS 主要应用在数据源及模版字段映射阶段。数据源即原始数据接口的封装，通过数据源获取实际业务需要的原始数据，比如媒资节目视频、节目专题数据、用户关注等业务数据；模版字段映射主要通过编写 Java 的函数根据实际业务逻辑生成字段内容。以往的开发模式下，如果业务逻辑有变化，需要变更然后发布 Java 应用才能生效，采用 FaaS 开发之后，只需要发布对应的 FaaS 函数即可，由于 FaaS 函数的发布是秒级，因此极大的提高了迭代效率。

1. 统一的数据源封装

我们使用 FaaS 实现数据源接口的封装，当有新的数据接口需要接入时，直接

在 FaaS 平台上通过编写函数实现，可以做到在本地 Java 应用不发布的情况下，直接上线新数据源。对于新业务接口的快速接入具有重要意义。同时这些数据源可以被重用，因此在多人协作的模式下，通过复用函数实现的数据源极大的减少了重复开发量。

2. FaaS 函数处理 API 协议模版字段映射



我们扩展了搭投框架，通过 Faas 的 SDK，服务端接口的模版解析阶段除了能

解析普通的 Java 函数，也可以支持解析 FaaS 函数，这类函数的代码不是通过原生 Java 代码编写，而是在 Faas 平台上用 Groovy 代码编写而成，这类函数的特点是编写、更新、发布均不需要重新部署哥伦布业务应用，只需要在 Faas 平台上操作函数即可。字段逻辑的修改可以完全不用重启 Java 应用，快速应对迭代变更。每个函数都有独立的生命周期和发布流程，不同函数的发布变更之间相互隔离。当有字段逻辑的变化时，可以完全不重启本地 Java 应用，直接通过函数的秒级发布来完成，极大提高了迭代效率。

四、总结与展望

目前优酷内容分发相关业务已经陆续引入 FaaS 能力，在 FaaS 的助力下，迭代效率提升。但是平台整体上还处于刚刚起步阶段，也是我们 Serverless 实践的初步尝试。后续我们希望在以下几个方面继续探索 FaaS 平台的技术与落地：

1）支持更多编程语言的运行时环境，以及更友好的云端 IDE 开发体验；
2）优化函数运行集群的资源调度策略，合理分配函数执行需要的资源，支持动态扩缩容；
3）结合内容分发业务的特点，寻找更多业务的切入点，通过 FaaS 进一步提升现有技术架构的灵活性和迭代效率。

一文详解领域驱动设计：是什么、为什么、怎么做？

作者 | 阿里文娱技术专家　战獒

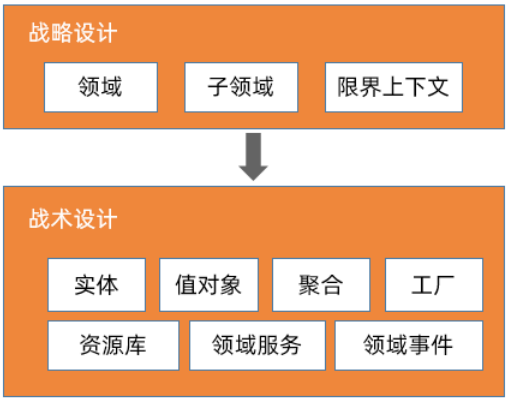
一、什么是领域驱动设计

领域驱动设计的概念是 2004 年 Evic Evans 在他的著作《Domain-Driven Design : Tackling Complexity in the Heart of Software》（中文译名：领域驱动设计：软件核心复杂性应对之道）中提出的，从领域驱动设计提出距今已经有 15 年的时间，为什么最近才开始在中国的互联网圈大行其道？似乎一夜之间大家都在谈论，那么领域驱动设计到底帮我们解决了什么问题？带着这些疑问，一起来看下阿里巴巴文娱是如何实践领域驱动设计的。

二、领域驱动设计大行其道的必然原因

软件系统从来都不是凭空而来，而是以软件的形式解决特定的问题。当我们面临现实世界的复杂问题时，如何以软件的形式落地？领域驱动设计是一套方法论，指导我们将复杂问题进行拆分、拆分出各个子系统间的关联以及是如何运转的，帮助我们解决大型的复杂系统在落地中遇到的问题。

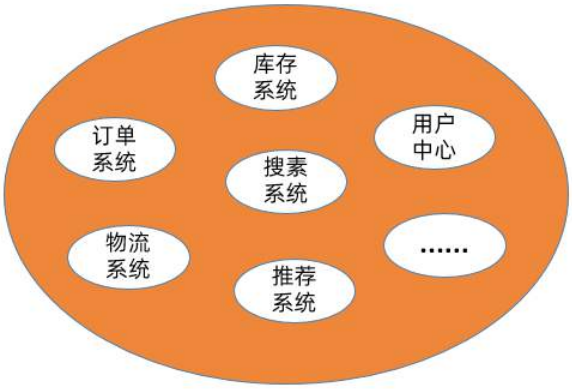
Evic Evans 在著作中将软件系统的设计分为 2 个部分：战略设计和战术设计。在战略设计层面提出了域、子域、限界上下文等重要概念；在战术设计层面提出了实体、值对象、领域服务、领域事件、聚合、工厂、资源库等重要概念。如图所示：



图一　战略设计与战术设计

战略设计部分指导我们如何拆分一个复杂的系统，战术部分指导我们对于拆分出来的单个子系统如何进行落地，在落地过程中应该遵循哪些原则。

以大家熟知的电子商务系统举例，早期的电商系统因为业务相对简单，用户量和团队规模也较小，一个单体应用就可以搞定，随着容量上升可以将单体应用进行横向扩容，比如早期的淘宝就是这样做的。拆分过程中我们可以把电商系统这个单体应用拆分成订单子系统、库存子系统、物流子系统、搜索推荐子系统等等，如图二所示：



图二　电商系统微服务划分

领域驱动设计在战略层面上的域、子域、限界上下文的划分思想和微服务的划分不谋而合。域对应一个问题空间，也就是上例中的电商系统；子域是把域这个大的问题空间拆分成若干个小的更容易解决的问题空间，也就是单体应用向微服务演进过程中划分出来的各个子系统；限界上下文是解决方案空间，每个子域对应一个或多个解决方案空间。微服务的划分是也是将一个大的问题拆分成若干个小的问题，每一个小的问题用一个或多个微服务来解决。

对于大多数开发同学来说都没有机会接触系统的划分，这些工作一般是公司的技术领导层与架构师来做的，普通的开发同学日常工作中接触到的只是某一个具体微服务或微服务中某一个模块的落地，那是不是说领域驱动设计对于普通开发同学来说就没有用了？当然不是这样，领域驱动设计中的战术设计部分就是指导我们如果落地一个系统才可以使系统具备高可扩展性、高可读性。

所有的系统最终都要以代码的形式落地，而落地的工作都是由普通的开发同学来做的，系统是否具备高可扩展性、高可读性直接影响了整个团队的效率。

三、传统分层架构存在的问题

对于大多数开发同学来说，大部分时间都花在落地一个个微服务上，下面我们来看阿里文娱是如何结合领域驱动设计的思想将微服务进行战术落地的。

目前笔者接触过的微服务大多数都是分层架构并且在 Service 层与 Manager 层实现具体的业务逻辑，使用 DO、DTO、BO、VO 等进行数据传输，数据和行为基本完全隔离。这种分层结构图三是阿里巴巴 Java 开发手册中的标准分层结构。

该规范中定义了各层的职责，其中最重要的两层 Service 层和 Manager 层是这样规范的（以下两层解释摘抄自《阿里巴巴 Java 开发手册》）：

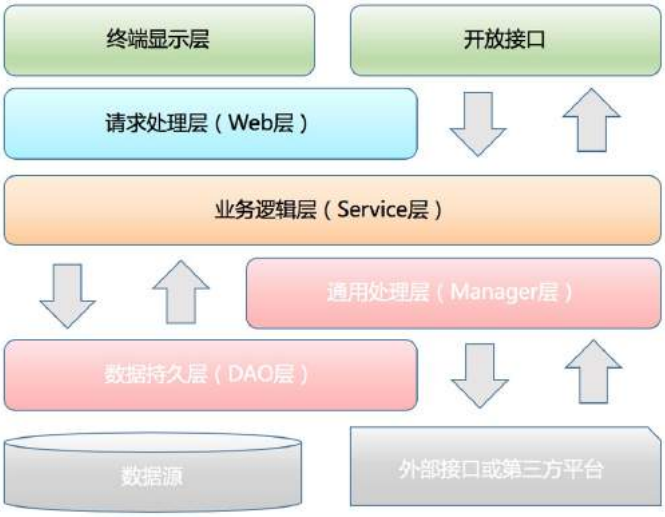
Service 层：相对具体的业务逻辑服务层

Manager 层：通用业务处理层，它有如下特征：

1）对第三方平台封装的层，预处理返回结果及转化异常信息；

2）对 Service 层通用能力的下沉，如缓存方案、中间件通用处理；

3）与 DAO 层交互，对多个 DAO 的组合复用



图三　传统的分层结构

阿里文娱早期的项目分层也基本都采用这种架构形式。上面的分层并没有问题，但是这种分层架构采用的是包的形式进行的层与层的隔离，需要每一位开发同学理解并且自觉遵守以上规范，但是在实际工作中我们发现很多同学对 Service 层和 Manager 层的区别并不是特别的清楚，即使清楚的同学大部分也并没有完全遵守手册中的规范，这种现象导致 Manager 层除了沉底一些通用能力以外和 Service 层并没有什么本质区别。

在实际的业务代码中 Service 层和 Manager 层都充斥了大量的第三方依赖，对系统的稳定性有很大的影响。每依赖一个第三方服务都要各种异常问题，这些异常处理的代码往往会和业务代码混在一起，当这种代码多了以后会使代码的可读性非常差。

阿里文娱业务的复杂度提升很快，业务迭代速度也很快，Service 层和 Manager 层代码量迅速膨胀，业务逻辑变得越来越复杂。在这种业务场景下，大文娱引入了领域驱动设计并设计了一套完整的领域驱动模型评估与演进的解决方案来辅助开发同学将领域驱动设计的思想真正的落地。

四、文娱领域驱动设计实践

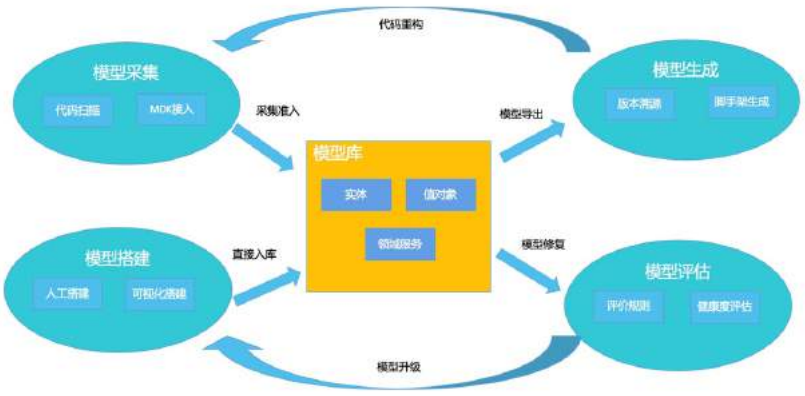
领域驱动设计的关键在于识别业务的模型，而模型又是会随着业务的发展而演进的，对于新的业务来说能效平台提供了业务模型分析的功能，开发同学可以在能效平台设计并搭建自己的领域模型，搭建出来后能效平台可以评估领域模型设计的是否合理，如果模型设计合理则可以基于以上设计的模型符合领域模型规范的代码。对于已有应用，能效平台设计了一套领域注解并以 SDK 的形式提供出去：

第一步：开发同学按照领域设计的原则对业务代码进行分析并打上注解；

第二步：能效平台可自动扫描该项目并收集该项目中的领域模型；

第三步：模型收集后，开发同学可以在能效平台改进业务模型并重新按照领域模型的规范生成代码。

完整流程如下图所示：



图四　领域模型的生命周期

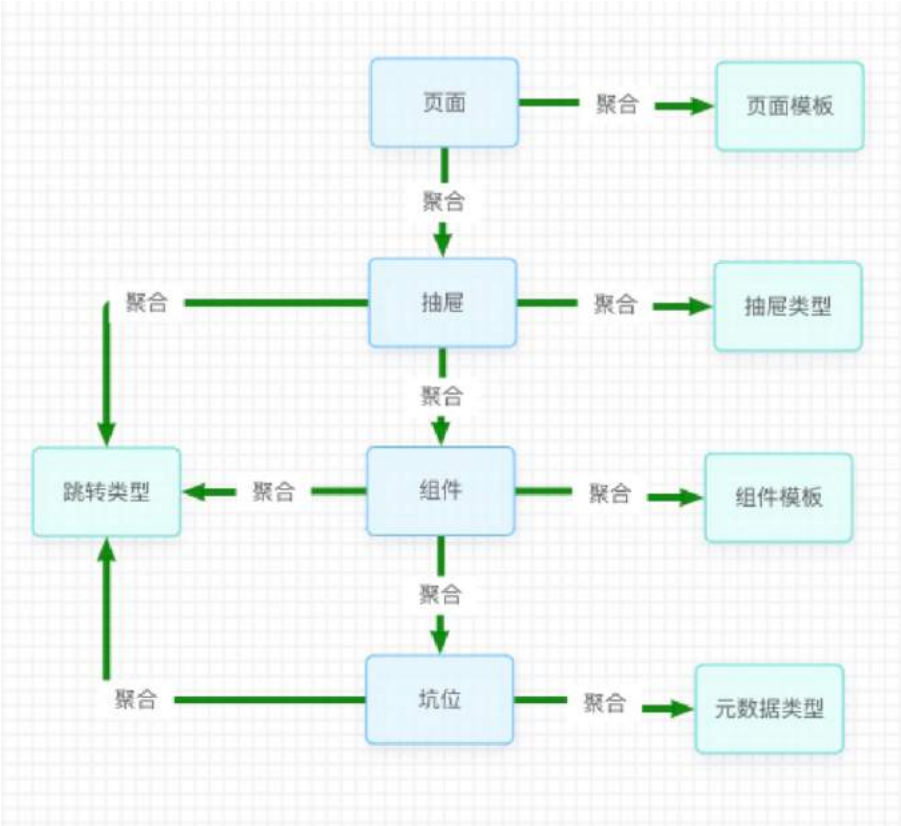
1. 模型采集

对于已有的准备重构的应用，我们设计了一套领域模型的注解，开发同学可以将注解加到对应的类、属性、方法上。当系统是按数据模型落地而不是按领域模型的方式落地时，可以先找到系统的数据模型，然后在能效平台对数据模型进行组织生成领域模型。

图五　领域模型注解

2. 模型搭建

对于新应用或者已经进行完模型采集的应用，开发同学可以在能效平台进行模型的搭建和修改，如图六所示。



图六　领域模型

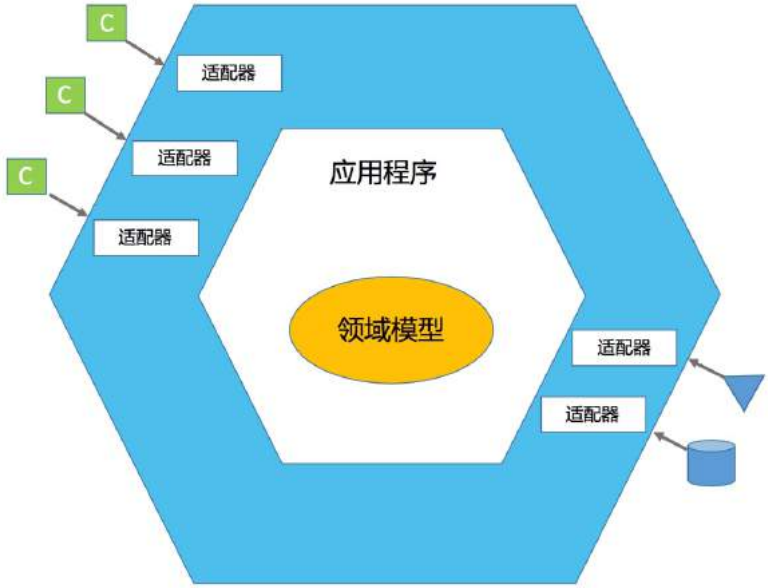
3. 健康度评估

对于已经搭建完的模型能效平台，根据领域驱动设计的规范创建了一套完整的校验规则，模型搭建完成在生成脚手架之前会根据校验规则进行打分，当打分通过时可以将模型生成脚手架。

图七　模型校验

4. 脚手架生成

当模型搭建完毕并且校验通过后可以将模型生成脚手架，其代码结构是按照六边形架构的标准生成的，六边形架构也成为端口与适配器架构，该架构的思想是将内部核心的领域逻辑与外界依赖进行隔离，这里的依赖是指所有对其他微服务的依赖、http 的依赖、数据库依赖、缓存依赖、消息中间件依赖等等，所有的这些依赖都通过适配器进行转换成应用可理解可识别的最小化信息。在实际的项目中，每种依赖都要考虑各种异常情况并进行处理，而这些处理实际上并不数据领域逻辑，却耦合到了业务代码里，当这种依赖多了对系统的稳定性会产生很大的影响，传统的分层架构虽然也会让我们将自身的领域逻辑和依赖进行分离，在阿里巴巴规范手册中提到所有的依赖都应该放到 Manager 层，但是这种规范是非常容易被打破的。六边形架构从应用分层上让我们更容易去遵守这样的规范。



图八　六边形架构

根据六边形架构的指导思想，在实际的应用分层中一般划分为四层，分别是：

1）**用户接口层**：负责用户展现相关的逻辑

2）**应用层**：负责对一个用例进行流程编排（将接口用例分成若干个步骤，但是不负责每步的具体实施）

3）**领域层**：负责实现核心的领域逻辑即业务逻辑（负责实现具体的业务逻辑）

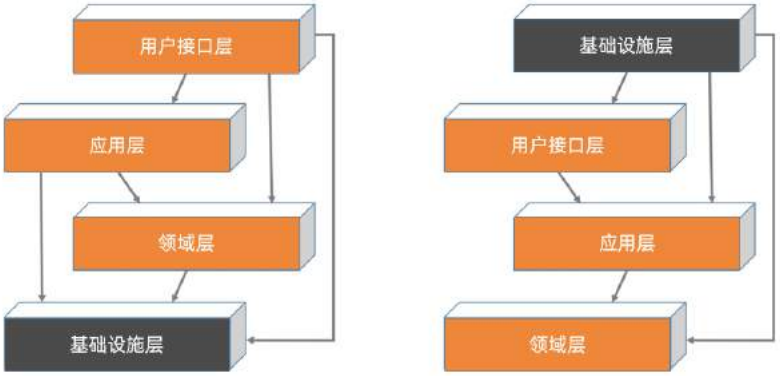
4）**基础设施层**：所有依赖的具体实现

但是从应用架构的角度看，层级组织形式可以分为两种：

1）传统分层架构，如图九左侧

这种分层架构是 Evic Evans 在《Domain-Driven Design：Tackling Complexity in the Heart of Software》中提出的，其中用户接口层、应用层、领域层可直接依赖基础设施层，与图三的传统架构并无本质区别，因为所有层都直接依赖了基

础设施层。这种方式需要强制开发同学将所有的依赖进行下沉，随着时间的推移这种规范非常容易被打破。



图九：层依赖关系

2）依赖倒置的分层架构，如图九右侧

这种分层架构是依赖倒置的分层架构，特点是：1）基础设计层可直接依赖其他三层，反之则不行；2）用户接口层、应用层、领域层如果要使用基础设施层中的能力，只能通过 IOC 的方式进行依赖注入，这也遵从了面向对象编程中的依赖倒置原则。当开发同学要在以上三层中直接引用第三方依赖时，是找不到具体的类信息的，也就是不能 import。同时这种方式对单元测试的规范也可以起到很大的作用，当我们编写单元测试时可以为领域层注入一个测试运行时的依赖，这样应用运行单元测试可以不依赖下游服务，在代码层面上也更加规范。

五、总结

经典的三层或多层架构虽然是目前最普遍的架构，但是在隔离方面做得并不够好。在业务架构选型时要结合自身业务特点，而不能千篇一律的选择某一种业务架构，合适的业务架构可以延长项目的生命周期，降低项目的重构频率，最终达到降低人力成本的目的。

阿里工程师教你 3 分钟实现数据源编排和接入

作者 | 阿里文娱高级开发工程师　天甘　慕理

一、背景

你一定也深有同感，在开发业务需求时，1/3 是时间都花费在了前期准备工作上。比如，调研数据来自哪里，查看几条真实数据及其结构，然后开始写代码过滤出所需要的数据，并将数据组装成期望的结构……经过漫长的过程后，才能进入正题。今天我将介绍，如何通过元数据中心，帮你直接提升开发效率。

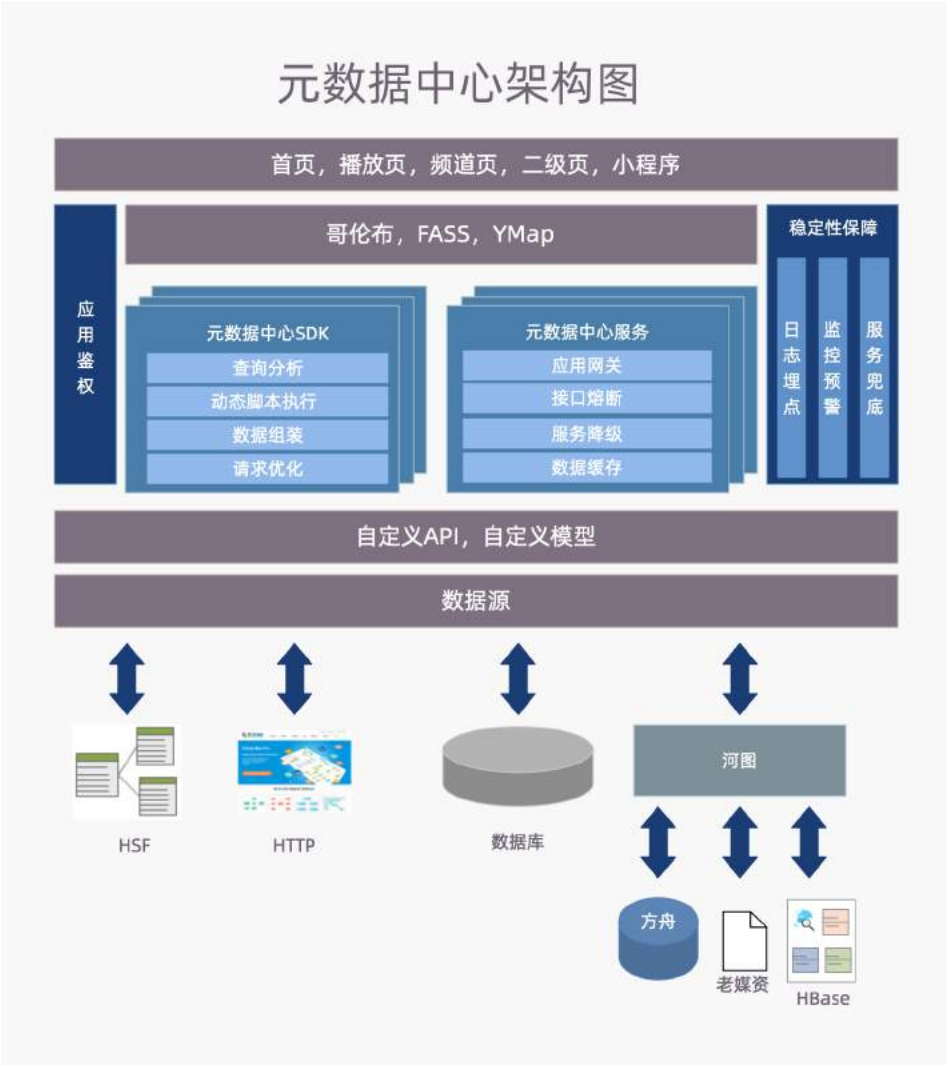
什么是元数据中心？它有两个主要作用：沉淀数据源广场、自定义接口。

1）数据源广场：通过关键字搜索出基础平台服务相关接口，可手动快速调用查看接口；

2）自定义接口：根据业务自定义所需字段 (字段名、类型)，元数据中心 SDK 还将接口调用这一系列逻辑抽象出来，沉淀了“接口调用引擎”，业务开发只需要将平台下发的接口标示传入引擎中，即可完成接口接入，由 hardcode 转为配置化。

除以上两点，元数据中心还建立了统一监控、熔断能力，采用多线程池，保证接口调用的效率及稳定性。

二、架构



元数据中心有两个核心模块：

1）数据源，对基础平台服务（像需求中的 ABCD 系统）接口做标准化调用；

2）自定义接口，为业务开发做接口编排，包括入参和出参的定义。

1. 数据源

数据源模块是将基础平台服务接口做了标准化调用，与其他平台标准化不同的是，基础平台服务不需要实现 java 接口，这样降低了基础服务接入。目前元数据中心暂时支持 HTTP 及 RPC 接口，后面我们会逐步支持分布式数据库数据源，分布式缓存数据源及 mock 数据源。数据源模块采用以下技术来保证调用的稳定：

1）数据源模块采用多线程池技术，不但可以并发调用，保证接口调用的稳定，还可以根据接口调用情况动态调整所使用的线程池，保证在某个接口性能较差情况下不影响其他接口的调用；

2）使用了泛化调用技术，泛化调用技术避免了引入二方包导致包冲突的问题；

3）数据源模块还支持分批调用，很多诸如 queryByIdList 这种查询，肯定需要对查询 ID 做 count 限制，但是有时候业务所需要的单次查询数多于限制数，那么就需要分批并发调用多次，然后再 merge 结果；

4）利用 ThreadLocal 技术元数据中心还支持对接口调用的超时时间做动态的调整，有些业务为了获取到数据可以容忍较长的 RT，有些业务对 RT 比较敏感，所以可以根据业务不同对超时时间做个性化设置。

2. 业务自定义接口

1）Schema 的定义

通常在业务开发时，接口主要解决两件事：1）对接底层的数据源，2）用几个数据源的结果组合成一个业务接口暴露给上层的业务。因此，在设计 Schema 时考虑了以上两点，将 Schema 分为调用和返回，根据不同的需求配置调用哪些数据源，并配置这些数据源的依赖关系，并且配置接口需要以什么样的方式返回，可直接透出数据源的返回结果，对结果进行简单维护，也可以自己定义结果，并指定取值的来源，对结果的类型和结构进行控制和干预。

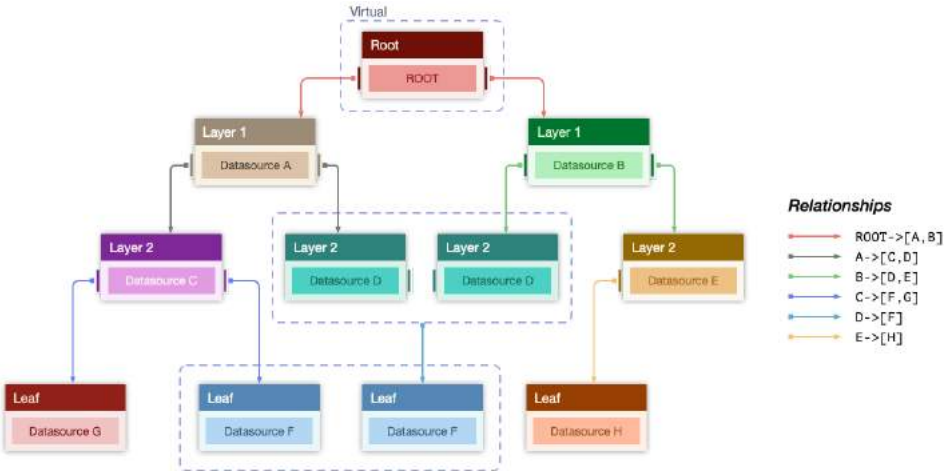
2）Schema 执行引擎

第一：简述执行过程

首先解决依赖关系，然后进行数据源调用，调用后再根据返回配置来处理。这里如果返回值配置的是一个表达式，那么引擎直接执行这个表达式来作为中间结果，并根据特殊配置对中间就结果进行处理。如果没有指定配置表达式，而是指定了自己需要的哪些字段，引擎会根据这些字段值的表达式分别进行设置，如果这个字段是一个嵌套，引擎对嵌套的 Schema 进行递归处理。

第二：如何解决数据源依赖

有很多使用场景是一个业务接口需要两个数据源的支持。对于依赖，抽象出一个依赖解决器，它会遍历所有的数据源调用，分析调用关系并生成一颗依赖树。在调用时，对树进行层次遍历和先序遍历，保证先执行到无依赖的调用，再处理有依赖的调用。举例，总共需要调用 8 个数据源分别为 A、B、C、D、E、F、G、H，其中 C 依赖 A，D 依赖 A 和 B，E 依赖 B，G 依赖 C，F 依赖 C 和 D，H 依赖 E。下图表示生成的依赖树：



如图所示，其中先按层次遍历 ROOT，但这是一个虚拟的节点不处理，之后调

处理 A 和 B，这里可以并发调用，也可以顺序调用，交给数据源调用层来处理。A 和 B 处理完成后，处理 A 的子节点和 B 的子节点 C、D、F。直到所有节点都处理完毕。

第三：如何映射结果

引擎调用数据源后，需要根据表达式来进行结果映射。如果是需要自定义返回的每一个 key，比如调用视频查询，默认返回的 key 为 id，而需要向业务方返回的是 vid，可配置 key 的值表达式。设计时希望表达式尽量简单，于是设计出了这样一种表达式，比如 vid，可配置为 datasource.video.response.id。

3）动态脚本的执行与优化

第一：为什么选择 Groovy

因为整体技术栈是 Java，因此首选的脚本语言就是基于 JVM，而选择 Groovy 一是因为成熟，同时语法比 Java 简洁，二是因为接入简单（只需一个 jar 包），三是因为本身兼容 Java 的大部分语法，学习成本较低。

第二：Groovy 遇到的问题和解决办法

主要是安全问题和性能问题。安全的问题非常严重，因为脚本是交给用户输入的，所以很容易出现风险代码。系统对 Groovy 的编译器进行了一系列自定义，同时使用沙箱，将代码进行拦截，过滤掉敏感操作，最大化避免风险代码的执行。

性能上，Groovy 的性能远比 Java 慢，于是进行了一番优化，比如将 Groovy 的代码进行预编译和缓存，保证执行时不会发生编译动作，保证执行一个 Groovy 单条脚本不会有太大的性能问题。

第三：最终解决方案

经过测试，发现 Groovy 本身的执行效率还是要低，因为一次请求可能需要执行成百上千次的脚本，一次执行的性能问题不明显，但是多次执行问题很容易就暴露出

来，比原生的 Java 执行仍然有百倍的差距。

这个既然是 Groovy 的问题，并且无解，就想到了是否可以用其它的方式替换 Groovy，于是研究了各种表达式引擎包括 Aviator、FEL、MVEL、JSONPath、Java 动态编译。并且对这些代码进行测试，使用两条语义相同的代码，分别编写了不同脚本的版本，与 Java 原生的代码进行比对。长时间运行后结果如下：

脚本执行千次耗时比对

表达式类型	最大	最小	平均
纯Java	3.234	0.059	0.085
Groovy独立	414.281	21.645	27.833
Groovy合并	319.423	14.002	17.831
Aviator	6.964	0.541	0.685
FEL	659.461	57.152	68.763
Json Path	1.524	0.143	0.178
Java动态编译	1.926	0.124	0.160
MVEL	125.885	1.169	1.549

上图为两个脚本执行 1000 次的耗时，发现使用 Java 动态编译和 JSON-Path，性能比 Groovy 提升了约 150 倍。但是缺点也非常明显：JSON-Path 是能写单行的取值语句，而 Java 语法复杂、不方便写内部类和方法等。考虑到使用场景，Java 和 JSON-Path 满足 95% 以上的场景，可再用 Groovy 写更复杂的脚本。

三、稳定性

稳定大于一切，接口调用稳定性是重要一环，如何保证稳定性呢?

1）有问题时要能及时的报警，

2）为了整个应用的稳定性，需要自动熔断。

1. 监控报警

因为元数据中心 SDK 是去中心化的，接口的调用逻辑发生在业务应用本地，SDK 利用 logback 的 appender 扩展将日志异步发送到日志服务中，再用监控平台采集日志服务的日志，在监控平台中完成监控报警的搭建。元数据中心 SDK 还集成了调用链监控平台，不但可以在业务应用对应的调用链监控平台上看到接口调用情况，还利用了 logback 的 ClassicConverter 扩展，在日志中统一加入调用链唯一 ID, 方便通过该 ID 查看全链路日志，以达到快速定位问题的目的。

2. 熔断降级

如果你的应用有外部依赖，那么保证系统稳定性，熔断是必不可缺的手段。想象一下，某个依赖的 RT 突然变高，如果没有及时熔断，那么你应用的线程池资源很快会被该依赖消耗完，且整个应用将无法响应，造成雪崩，所以熔断的重要性是不言而喻的。元数据中心目前支持三种熔断策略：RT、秒级异常比例、分钟异常数。

四、实践提效

说了这么多，究竟能否为业务开发提效呢。实践是检验真理的唯一标准，接下来让我们一起来看下元数据中心如何与业务结合，为业务开发提效。

1. 与内容分发开发框架集成

目前优酷主 APP 业务基本上是统一的内容分发开发框架开发的，那么与内容分发开发框架集成，是为业务开发提效最好的切入点。内容分发开发框架虽然具有强大

的配置能力，但是它的数据源开发还是面临到处找接口，学习怎么调用，然后引入二方包，hard code，提测，aone 发布一系列漫长而重复的操作。另外一旦有接口需要切换或者业务需要增加字段等变更操作，还需要通过 hard code 方式改动，这样不但开发效率比较低，频繁的发布还增加了系统不稳定性。结合元数据中心的目标，内容分发开发框架集成元数据中心就可以完美解决它的痛点：

- 接口接入不再需要 hard code，通过配置即可完成；
- 接口需要切换或者业务需要增加字段时，不需要 hard code，在元数据中心做切换，做字段的增加，然后通过脚本语言动态的将字段读取出来；
- 无需重复的打印监控日志，也不需要单独做熔断，由元数据中心统一处理。

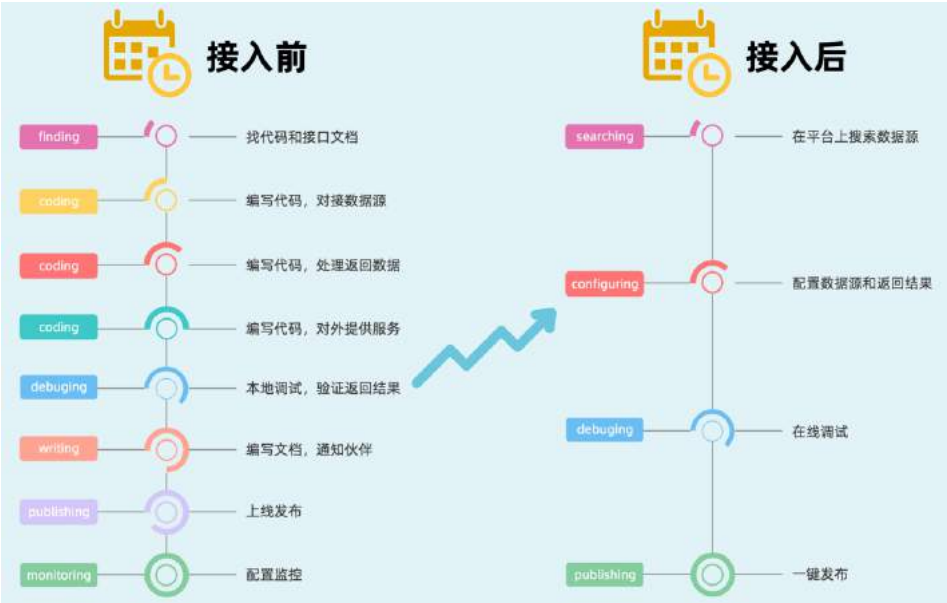
再来看看开发者的反馈情况，最近与业务开发同学了解到，目前开发新组件的需求中有涉及需要读取某些数据，通过元数据中心配置了相关接口和所需字段，且结合开发框架的配置能力，从开发到上线基本没有 hard code, 通过配置快速的上线验证。

2. 与 Faas 集成

众所周知，互联网产品讲究的是快速迭代和试错，业务需要具备快速上线能力，与竞争对手 PK。这就意味着背后的技术研发流程需要更加高效，元数据中心与 Faas 结合就可以实现快速上线的效果。因为业务开发通过元数据中心可以快速接入接口，再结合 Faas 的开箱即用特点，直接编写业务逻辑，保存之后自动完成构建、部署，整个流程简单、高效。足以达到快速上线验证的目的。前优酷小程序业务已经在元数据中心 +Faas 下上线了很多业务。

六、总结与展望

元数据中心平台已经初步上线，并通过与内容分发开发框架集成已服务于优酷 APP 部分业务场景及与 Faas 集成服务于小程序部分业务场景。从业务开发反馈来看，目前已实现了业务在接入数据源这一层由 hard code 方式转变为配置方式，极大提高了开发效率。



虽然平台基础能力已经建设完成，并且开发提效这一目标也已初步实现，但是平台建设还有很长一段路要走，还需要做很多工作，比如提高配置效率，支持更多协议数据源，将平台的价值发挥到最大化。

Serverless 如何做到快速发布？微应用平台技术实践

作者 | 阿里文娱技术专家　嘉若

一、背景

作为开发者，在面对需求变更期间，我们通常的状态是开发 – 自测 – 联调，需要频繁改动代码，即使修改少量代码，也要重启容器来检查执行效果。等待时间少则 3–5 分钟，多则 10 分钟以上，严重加长了非开发时间。

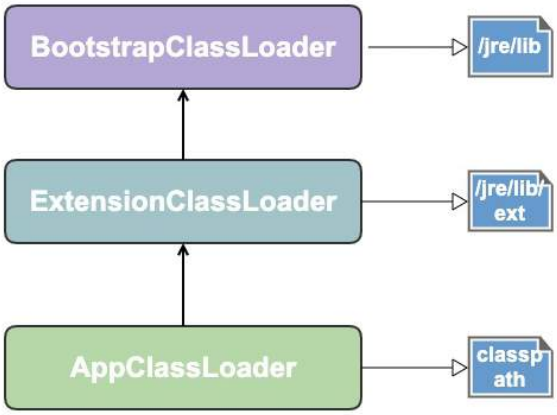
有没有更好的方式来解决这一问题？一种方案是写出没 bug 的代码，这显然很难，因为很多 bug 并非是技术型的，更多的是业务型；另一方案就是采用动态语言，比如 node.js,erlang，但这也就意味着技术选型的单一，以及放弃其他语言的生态。

那么，作为以 java 语言为主的开发者，寻找一种可动态部署的方案，就是非常靠谱的是可行方案。

二、java 动态加载

ClassLoader

说到动态部署，jvm 的类加载机制是一个绕不过去的问题。我们都知道 jvm 采用双亲委派模式，Java 类是通过 Java 虚拟机加载的，某个类的 class 文件在被 classloader 加载后，会生成对应的 Class 对象，之后就可以创建该类的实例。

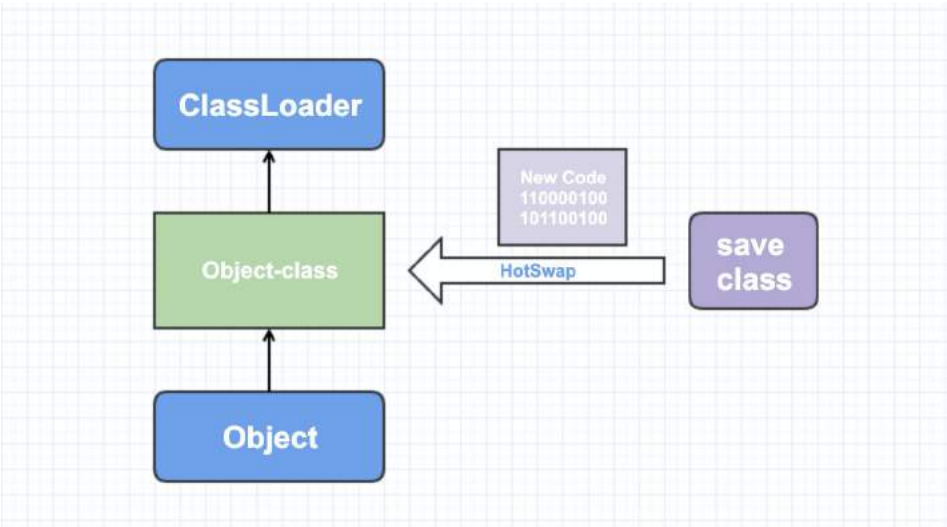


基于双亲委派的模式既能保证 jvm 的安全性，也能基于此更高效的执行 class 转换的字节码指令，加快执行速度。

默认的虚拟机行为只会在启动时加载类，类被加载后就不会被替换和更新，如果要实现动态加载，需要改变 classLoader 的加载行为，令 jvm 监听到 class 文件的更新，重新加载 class 文件，但同时如此也会对 jvm 的安全留下大坑，如此需要十分谨慎。

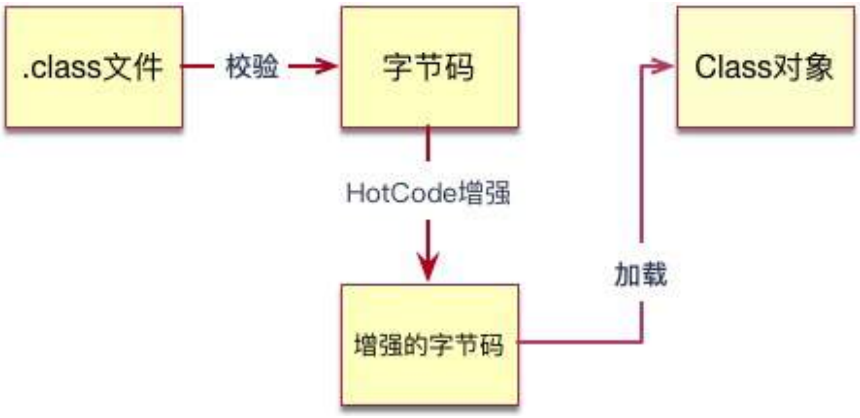
三、HotSwap

在 jdk1.4 期间，jvm 就引入了 hotswap，允许调试者使用同一个类标识来更新类的字节码，避免了类的字节码被修改就重载容器。但这个弊端是仅限于修改方法体的修改——也就是说除了方法体，不能添加方法域，也不能修改其他任何代码。



目前商业软件 JRebel 也做到在极少限制的情况，做到动态更新类的变动，但是这种方案商业应用收费，兼容性方面有待考验。

阿里开发的 hotCode，也可以做到热部署，采用的是遵守 hotswap 规则，同时采用代理的方式支持类字节码的改变，目前已经支持到不限于方法体的改动，可以增加以及修改方法，解决了类级别的频繁修改部署问题，但针对应用级别（变化数量多）的还是需要重启容器来解决。

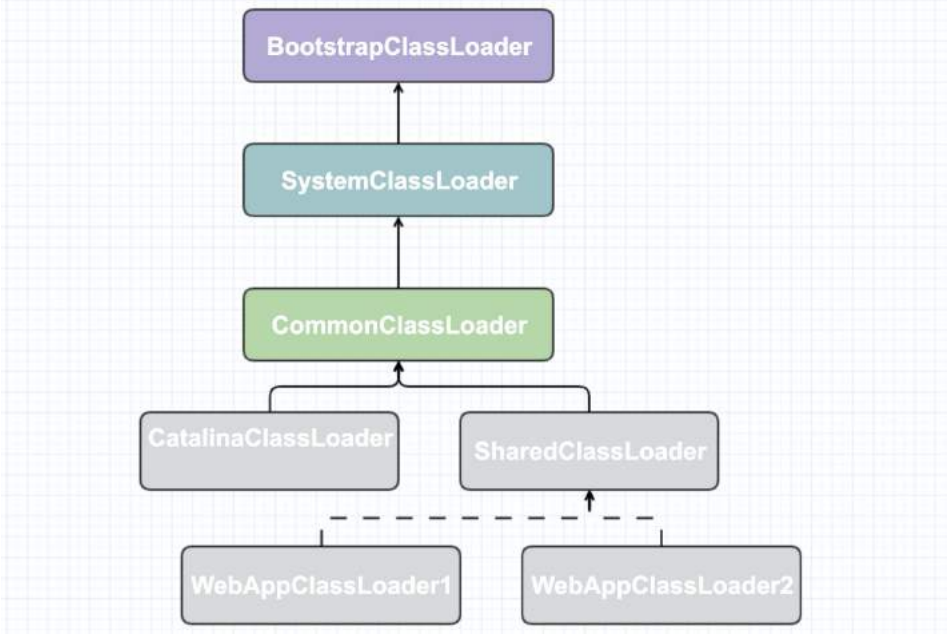


四、多 ClassLoader

基于应用级别的动态加载，我们可以理解为整个应用的类改动比较大 ，采用双亲委派模式，我们就需要重新启动整个应用来解决了。

但 tomcat 给我们了不同的解决方案，tomcat 也是支持动态部署的，tomcat 把 classLoader 重新设计了，也是沿用双亲委派的模式之上，新扩展了自己的 classLoader，借此来管理和分离不同 APP 的加载。

tomcat 启动的时候会有启动一个线程检查加载的类是否发生变化（具体参考 Lifecycle 的实现），如果发生了变化就会把应用的启动的线程停止掉清理，同时把该应用的 WebAPPClassLoader 废弃，再创建一个新的 WebAPPClassLoader 加载 APP。



多 classLoader 的优点：

1）同一进程可以加载不同的 APP，即便有相同的 className；

2）不同 APP 可以共享类库，容器提前加载，无需多次加载；

3）APP 类隔离，减少冲突，同时单个 APP 的重加载不影响其他 APP(但无法内存和 cpu 隔离)。

五、微应用平台的 Class 动态加载

YMAP (优酷微应用平台)，借鉴 serverLess 思想，支持应用秒级部署，快速扩缩容，支撑业务快速灵活变更，让开发者只专注于业务。在热部署方面，我们对比了多种方案。

方案对比

方案	优点	缺点
热部署	快速	局部更新
多 classsLoader	基于 classLoader 业务隔离	实现成本高

因为 YMAP 要做一个平台化的方案，所以不能只限于热部署，还要考虑到应用方的二方库扩展以及多文件的变动，要做到快速，简单，易用，所以我们最终采用了多 classLoader 的方案。

class 卸载

多 classLoader 的只是更好的解决了类隔离的问题，但要实现动态部署的还需要结合 jvm 的 class 卸载的能力，那么怎样才能卸载 class ?

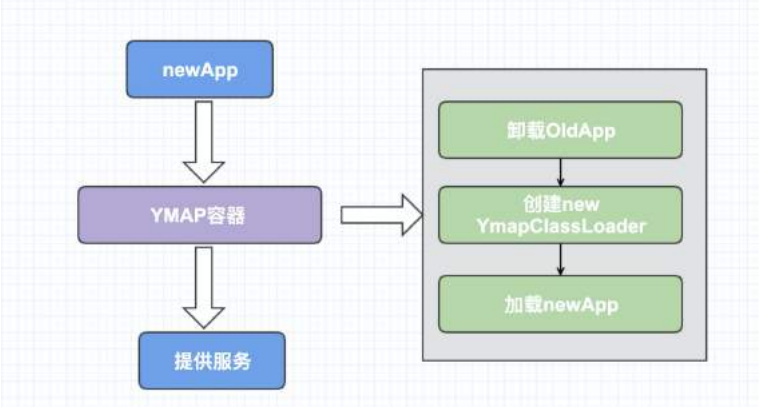
JVM 中的 Class 只有满足以下三个条件，才能被 GC 回收，也就是该 Class 被卸载 (unload):

1）该类所有的实例都已经被 GC。

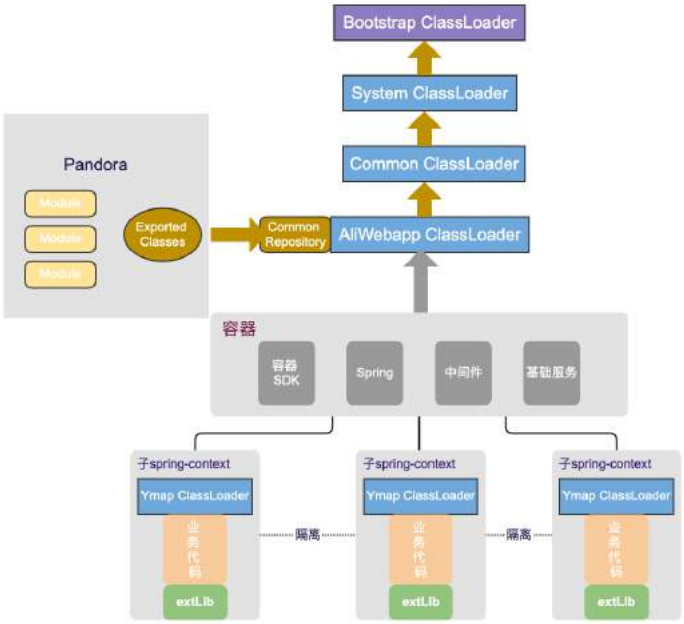
2）加载该类的 ClassLoader 实例已经被 GC。

3）该类的 java.lang.Class 对象没有在任何地方被引用。

如此，基于 YMAP（优酷微应用平台）ClassLoader 的隔离，有新的应用代码部署的时候，采用新老 classLoader 的替换的方式，回收老 APP 的资源，就能做到不启动容器的情况下，做到应用的重新部署。



YMAP（优酷微应用平台）ClassLoader



以上就是 YMAP 的 classLoader 的架构全貌，我们在多 classLoader 基础之上，增加一些更贴合业务的落地方案。

1）原生支持 spring 的全能力，方便开发；

2）中间件（DB，cache）以及基础能力无需重复接入，一站支持，让业务方只关心业务代码；

3）支持二方包自定义扩展，增加业务开发灵活性；

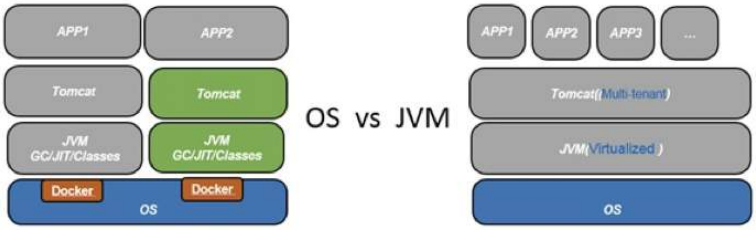
4）支持资源隔离（CPU 和内存隔离），支持混合部署，提升机器利用率；

5）快速动态部署，20s 以内。

资源隔离

Tomcat 的多 APP 部署一直没有没有被推广的原因之一，就是安全性太低了，多个 APP 之间会因为资源划分的问题被相互影响，那么如何解决资源隔离问题?

我们引入了集团 AliJdk 的多租户能力，复用的是 linux 的 cgroup 的能力，这个方案可以让单台机器上的部署的多个 APP 之间资源 (内存、CPU) 相互隔离。



常见的资源隔离问题主要表现在多个应用对资源的争夺，比如 APP1 出现了死循环，导致耗费了整个机器资源的 cpu，其他应用无法继续获得指令集的执行时间。以及常见的内存泄漏，也会导致其他应用出现不可访问的情况。

另外我们做过一个调查，发现多数的长尾应用在 90% 的时间，机器资源都是浪

费的，如果给这类的应用采用硬件分离部署，会过多的浪费主机资源，不如采用多租户的方式，单容器部署多个应用，可以大幅提升资源利用率。

ymap 引入了多租户能力，通过 JVM 的虚拟化 / 资源隔离，以支持容器的多租户，让多个应用可以同时部署在同一个容器而互相不受影响，从而可以更大程度的提高资源利用率，降低单应用的部署成本。

多租户能力的引入，解决了以下痛点：

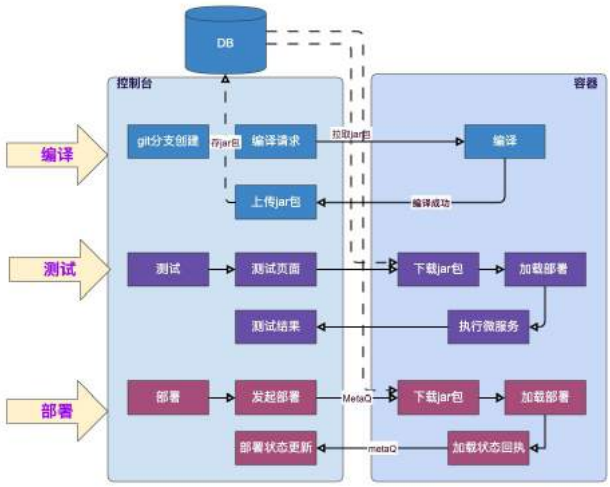
1）资源隔离能力，包括 IO 资源和 CPU 资源隔离，以及 mem 隔离；

2）高密度部署，降低应用部署成本；

3）减少进程间沟通（序列化开销），打通共享能力。

快速部署

YmapClassLoader 帮助我们解决了技术上动态加载能力，同时我们也配套了相应的 APP 打包编译系统，结合 aone 的构建和 git 代码管理能力，方便开发者快速变更和部署，从开发提交代码，到编译部署，再到测试反馈，提供一站式服务，整个过程控制在 30s 以内，方便开发者快速验证，无需耗费过多的时间再部署发布上。



六、优酷微应用平台的生态

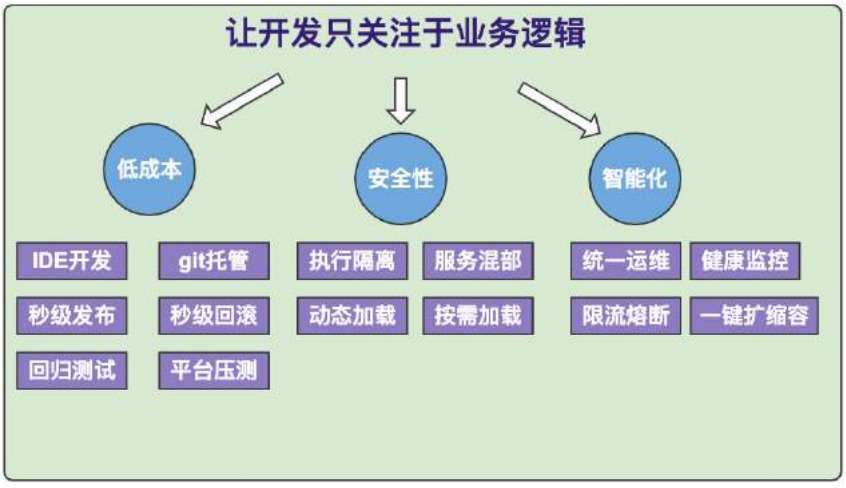
YMAP（优酷微应用平台）的设计初心是希望实现类 serverless 的微服务平台，让开发者只关心专注于业务逻辑：

在设计上，基于低成本、安全、高效为关键目标，为研发赋能；

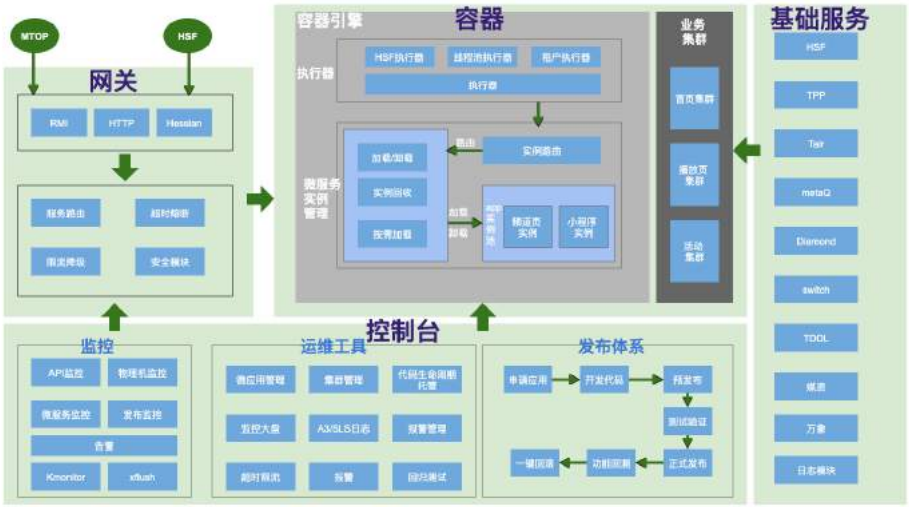
在开发上，支持一站式发布体系，从开发，测试到发布，全部可视化支持；

在容器上，动态部署能力，多租户隔离能力，以及基础服务扩展能力，提升应用部署效率，以及资源利用率；

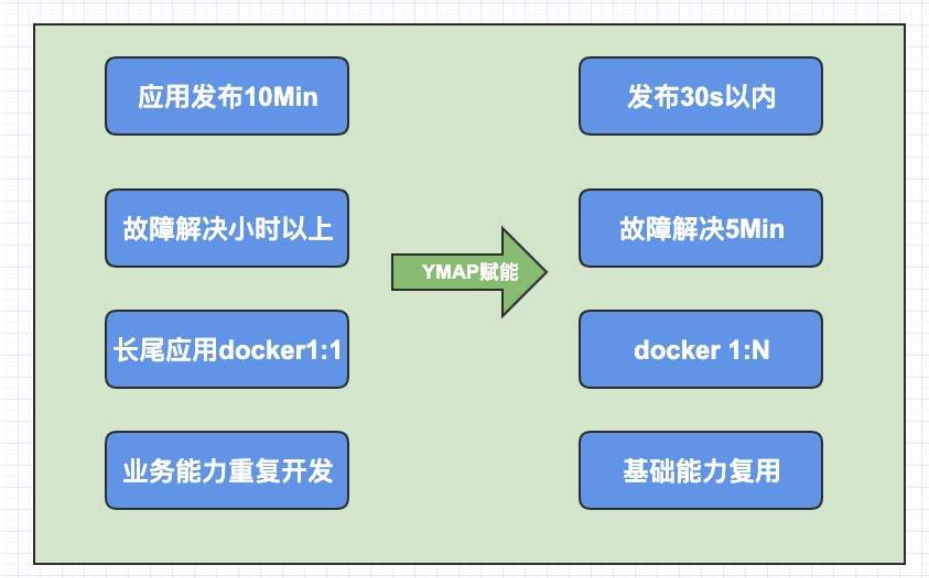
在运维上，帮应用实现智能化运维工作，应用自动接入监控报警，一键快速扩缩容，应用运维大盘以及微服务日志自动云端化可视化等，提升应用的运维效率。



微应用平台服务架构



微应用平台业务效果



YMAP（优酷微应用平台）基于 serverless 的思想，让应用服务化，提供最小逻辑单元，让业务快速开发落地，同时也兼顾资源效率的提升，采用 1：N 的部署方式，进可能的提升主机利用率。从开发赋能到运维赋能，为业务方带来更低的成本，更灵活的应对方式。

阿里巴巴文娱技术
钉钉交流群

阿里巴巴文娱技术公众号

阿里云开发者“藏经阁”
海量免费电子书下载