
客渠技术月刊
vol.3
Technical Journal
经验总结 Kafka消费堆积引发的UIOC
技术分享 JAVA8特性之Stream
微 创 新 基于h5的可配置可嵌入多终端销售工具的设计和开发方案
技术前沿 去哪儿网业务大规模容器化最佳实践

目录

经验总结：Kafka 消费堆积引发的 UIOC

MCS-CAS-EXP(客户接入服务系统-快递服务)系统 UICO	系统级别：S　异常时间：2022-03-07 17:54:00--2022-03-07 22:21:00　责任组织：O 线应用支持组/中间件产品组　责任人：(866319)谢晓超/(193937)李剑芳
影响	影响范围:部分用户的清单运费、路由、回告、寄件码数据等推送延迟 (如拼多多、唯品会等) 影响内容:唯品会：影响约 200 单唯品会寄件码数据回传，清单和路由数据回传出现延迟。 拼多多：实操回告数据推送速率出现抖动。 故障现象:1、kfk 消费组消费拼多多回告数据出现积压告警 2、客户反馈唯品会寄件码出现推送
处理过程	17:54:00 异常开始 02:00:00 shiva-oms-sws 系统容灾演练,导致 exp 消费者未消费 sws 的清单运费数据 16:50:00 用户反馈得物、抖音清单运费数据缺失 16:55:00 研发定位异常原因是 kfk 主题未正常消费 16:59:00 上报领导，申请串行重启 sync 模块，消费 sws 系统清单运费 kfk 主题 17:01:00 sync 模块串行重启完毕，未消费的 sws 清单运费数据消费完毕，合计消费 1.2 亿数据量 17:26:00 观察系统处理正常，未出现业务告警 18:07:00 拼多多回告卡夫卡主题出现消费积压，开始分析定位 18:18:00 启动 UIOC 18:19:00 中间件定位到消费积压原因为：Kafka 触发 Rebalance，导致消费效率降低 18:51:00 降低唯品会消费组消费线程数，降低推送频率 19:21:00 扩容 exp-push 从 15 扩容为 32 个 19:26:00 停止推送唯品会和通用客户失败的路由重推送任务 20:33:00 扩容 extgw 生产 8 个节点，容灾 8 个节点 20:42:00 停止清单费用推送失败重推送任务 20:46:00 恢复消费推送唯品会相关数据 kfk 线程数

	21:09:00 逐步开启路由推送失败表重推送任务 21:54:00 开启剩余关闭的重推送任务 22:02:00 观察无消费积压、关闭 UIOC
原因分析	推送大量清单费用数据，消费者从 kfk top 取数据推送给唯品会等大客户，由于短时间推送大量数据加上唯品会对清单运费推送接口进行限流，导致推送成功率降低，推送耗时增加，推送完成一批数据后消费者再从 kfk 获取数据进行下次推送，由于整体推送效率降低，导致消费者无法在指定的时间之内完成消息的消费（多主题使用同一个消费者），引起 kafka 不断 Rebalance 引起消费积压
改进措施	1. 优化 kfk 告警规则 2. 优化线程池大小并按照线程池进行监控 3. 中间件修复容灾切换，部分消费者未正常消费的问题 4. 不同模块不能共用同一个消费组
经验教训	缺乏对于 kfk reblance 场景处理的应急处理方法，将可行执行方案整理成可操作文档。

经验总结：加密机异常引发的 UIOC

MCS-CAS-EXP(客户接入服务系统-快递服务)系统 UICO	系统级别：S 异常时间：2022-03-07 10:18:00--2022-03-07 14:35:00 责任组织：M 线应用支持组/顺丰科技技术集团网络安全部办公安全组 责任人：(01187872)张浙栋/(01110629)陈勇
影响	影响范围:影响部分外部大客户下单超时 影响内容:3-7 号全天下单量 19322712，异常时间段(10:20-14:35)下单量 5779583。 2-28 号全天下单量 16224740，在对应异常时间段(10:20-14:35)下单量 4913068。 2-21 号全天下单量 16014842，在对应异常时间段(10:20-14:35)下单量 4936535。 2-14 号全天下单量 15449663，在对应异常时间段(10:20-14:35)下单量 4809942。 订单量对比前三周全天和异常同时间段没有下降，没有实际业务影响。 故障现象:1. 有业务请求失败告警和接口响应超时报警（接口成功率 98.38%）。 2. 部分外部大客户反馈顺丰接口有少量请求超时
处理过程	10:18 异常开始 10:20 接到接口响应超时报警，外部大客户反馈下单接口有少量超时并上报异常 10:25 收到接口告警恢复通知，检查原因中 10:40 又收到大量接口超时告警，启动 UIOC

	10:43 统计 ordercreate 模块日志有大量调用 PSDS 接口超时现象 10:51 确认与 PSDS 接口无关系，继续分析后端缓慢原因 11:14 决策后对后端 ordercreate 服务进行节点扩容 11:51 扩容后问题未解决，再分析推测是调用加密机服务异常，安排研发同步准备增加打印耗时日志的版本帮助确定后端缓慢问题 11:54 确认一台容灾加密机连接数被 PMP 系统打满 12:15 决策切流重启 ordercreate 模块尝试释放连接 12:43 通知 PMP 系统先停止应用降低加密机压力，验证还有异常，马上安排加密机厂商去机房现场支持。 13:31 强行重启加密机服务，验证依然没有修复，连接数仍然被打满。同步在等加密机厂商到达机房。 13:41 决策从 ordercreate 配置文件剔除异常的加密机节点 13:43 屏蔽 iis 系统请求，开墙修复 PMP 调用加密机问题 14:01 配置文件剔除异常加密机节点后，先切小流量验证发现后端异常超时增多，回滚配置，回切流量到正常的环境先恢复业务，后打 dump 后确认异常增多是加密机引起 14:13 重新核对配置文件修改正确，并重启应用，切流后验证应用核心下单功能恢复正常 14:36 排查其他非核心功能配置文件还有 2 个模块使用该异常加密机，并全部剔除这个异常加密机节点，验证应用恢复 16:01 继续观察应用无异常，关闭 UIOC
原因分析	pmp 系统突发大量请求调用加密机，加密机集群中的一台服务器连接数达到单台加密机并发瓶颈，新的请求出现堵塞，导致 exp 系统的 ordercreate 模块调用加密机接口部分出现失败，影响唯品会和拼多多部分下单失败。 暂停 PMP 加密机调用后，加密机持续异常无法释放连接，重启加密机无效。
改进措施	业务侧增加加密机耗时异常路径追踪，发现加密机异常时及时处理。 梳理加密机集群清单，优化集群分配 优化加密机定位方法(监控，日志，厂商) 应用加密机日志目录正确性排查 加密机客户端 SDK 以及服务器版本升级

经验教训	加密机问题定位时间过长，加密机设备对于问题定位提供信息过少。

经验总结：DB 异常引发的 UIOC

ESG-OCS2-ODP(在线丰发平台)系统 UICO	系统级别：A　异常时间：2022-02-20 17:38:00--2022-02-20 17:50:00　责任组织：在线客服产品研发组/服务运维研发组　责任人：(01375725)张凡/(01387123)李海泉
影响	影响范围:部分用户 影响内容:在线客服系统，人工、机器人无法正常服务。 业务影响： 1、异常时间段业务影响 83.17%，计算数据：1-异常当天业务 1786/正常业务 10613； 2、全天业务影响 1.07%，计算数据：1-异常当天业务 304712/正常业务 308001； 故障现象:1、在线客服系统、人工、机器人等系统模块业务监控告警，环比各业务下降； 2、核心模块 odp-session 模块线程告警；
处理过程	17:38 异常开始 17:38 值班人员，确认系统告警问题，联系运维确认告警异常。 17:46 重启腾讯云 nginx 模块应用 17:46 重启 odp-session 模块，各业务恢复正常 17:47 启动 UIOC 17:47 确认及分析应用 http 线程告警原因 17:50 各业务用户验证回复正常 18:04 关闭 uioc
原因分析	因 pis 系统 hbase 容灾集群一台机器异常，引发应用集群的均衡器短时间服务出现较大延迟，影响到 pis 对外提供的接口 http://api-market.int.sfcloud.local:1080/inc-pis-core/queryPromiseExtP 接口异常，导致 odp 系统调用 PIS 的 odp-session 模块线程耗光，在线客服出现异常，无法正常服务。
改进措施	pis 系统 hbase 容灾集群一台机器异常问题跟进 ODP 系统健壮性问题解决：在线系统方法块级别的熔断（例如方法块级别针对 pis，oms，sisp 的 http req 做管控） 当前采用阿里系官方原生的 sentinel 组件可以行管控，需要我们研发自己开发一部分代码。 针对云原生方案： 短期：通过整合原生 sentinel 组件接入，预计 2.28 日完成 长期：通过平台完整对接（接口级，方法块级都接入），预计 3 月中下旬完成

经验教训	本次 uioc，主要是因为外围系统异常，自身系统受影响，没有做到熔断机制，受制于外围系统，需要 odp 系统做改善。 1、系统调用接口没有熔断机制； 2、高峰核心系统健壮性问题；

经验总结：nginx 变更引发的 UIOC

MCS-CAS-API(客户接入服务系统-统一接入平台)UICO	系统级别：S 异常时间：2022-01-12 11:00:00--2022-01-12 16:15:00 责任组织：中间件产品组 责任人：(01383326)韩武君
影响	影响范围:部分外部大客户 影响内容:异常当天 1-12 号全天下单量 19112144 ,异常时间段(11:00-16:15)下单量 7248122，上一周 1-5 号全天下单量 17119823 ,异常时间段(11:00-16:15)下单量 6725234， 上上周 12-29 号全天下单量 16093041 ,异常时间段(11:00-16:15)下单量 6082496， 上上上周 12-22 号全天下单量 17002348 ,异常时间段(11:00-16:15)下单量 6476635， 异常当天对比前 3 周同时间段业务量没有下降，没有实际业务影响。 故障现象:部分外部大客户反馈下单报 SSL 握手失败错误
处理过程	11:00 异常开始 15:42 接到部分大客户报障下单报 SSL 握手失败错误，启动 UIOC 15:45 查看 BGP 的 nginx 的日志发现很多 ssl 握手失败报错 15:50 查看 nginx 的 error 日志发现从 11:00 开始 ssl 握手失败开始频繁出现 16:05 决策移除 BGP 的其中两个节点，操作移除第 1 个节点之后异常没有消除 16:10 查看 BGP nginx 的配置文件在 11:00 被修改过，并且 nginx 进程重新 reload 过配置文件。对比备份文件发现当前配置文件缺失部分 ssl 加密套件算法 16:13 决策修改 nginx 配置，把缺失的加密套件算法加上，并重载 nginx 配置完成,查看日志 ssl 握手失败报错消失 16:15 用户反馈 https 下单恢复正常，观察无新增异常，关闭 UIOC
原因分析	中间件开发人员在测试环境更新模板的时候错误更新到了生产环境，导致大客户的 BGP 负载丢失部分 ssl 加密套件算法，引起部分用户下单时候出现 ssl 握手失败。
改进措施	检视 bgp 环境的 nginx 配置文件模板，增加缺失部分配置 在 BGP 的 nginx 日志上面配置上 ssl 握手错误日志监控

	nginx 配置文件模板管理存放在管理平台
经验教训	生产环境变更要走 CAB 评审

技术分享：JAVA8 特性之 Stream

分享人介绍

姓名：万毅

工号：380183

部门：客户体验研发部

小组：在线客服产品研发组

负责在线客服组系统架构升级和综合治理，不断提升和钻研JAVA技术栈，斜杠青年，喜欢研究最新技术和行业风口。

学习Stream的好处和优势

·无存储：

流并不存储值；

流的元素源自数据源（可能是某个数据结构、生成函数或/O通道等等），通过一系列计算步骤得到；

·函数式风格：

对流的操作会产生一个结果，但流的数据源不会被修改；

·惰性求值：

多数流操作（包括过滤、映射、排序以及去重）都可以以惰性方式实现。

这使得我们可以用一遍遍历完成整个流水线操作，并可以用短路操作提供更高效的实现；

·无需上界：

不少问题都可以被表达为无限流（infinite stream）：

用户不停地读取流直到满意的结果出现为止（比如说，枚举 完美数 这个操作可以被表达为在所有整数上进行过滤）；

集合是有限的，但流可以表达为无线流；

·代码简练：

对于一些collection的迭代处理操作，使用 stream 编写可以十分简洁，如果使用传统的 collection 迭代操作，代码可能十分啰嗦，可读性也会比较糟糕；

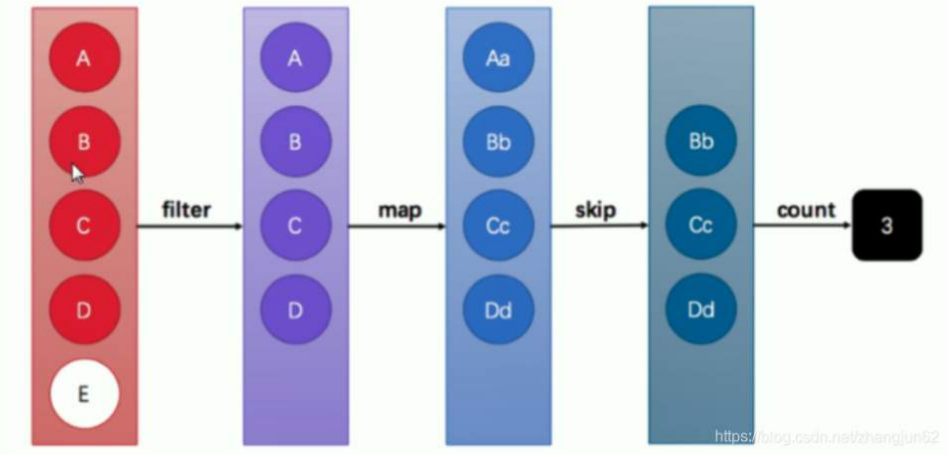
stream 和 iterator 迭代的效率比较

整体思想（抽象）

整体来看，流式思想类似于工厂车间的“生产流水线”。

视图理解（逐步）

当需要对多个元素进行操作（特别是多步操作）的时候，考虑到性能及便利性，应该首先拼好一个“模型”步骤方案，然后再按照方案去执行它。



Stream流是什么（偏理论，书面解释）

图中展示了过滤、映射、跳过、计数等多步操作，这是一种集合元素的处理方案，而方案就是一种“函数模型”。**图中的每一个方框都是一个“流”**，调用指定的方法，可以从一个流模型转换为另一个流模型。而最右侧的数字3是最终结果。

这里的 filter 、 map 、 skip 都是在对函数模型进行操作，**集合元素并没有真正被处理**。只有当终结方法 count执行的时候，整个模型才会按照指定策略执行操作。而这得益于Lambda的延迟执行特性

“Stream流”其实是一个集合元素的函数模型，它并不是集合，也不是数据结构，其本身并不存储任何元素（或其地址值）。

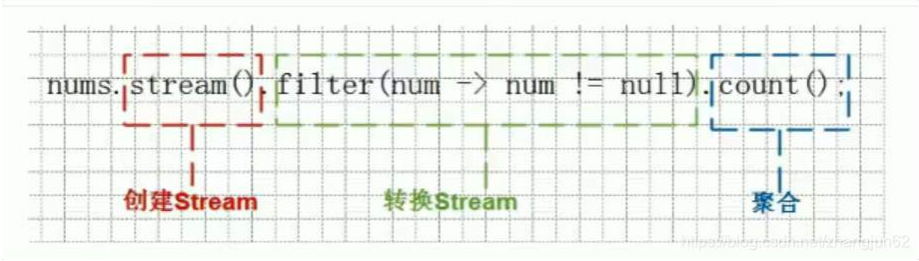
Stream流是什么（通俗易懂）

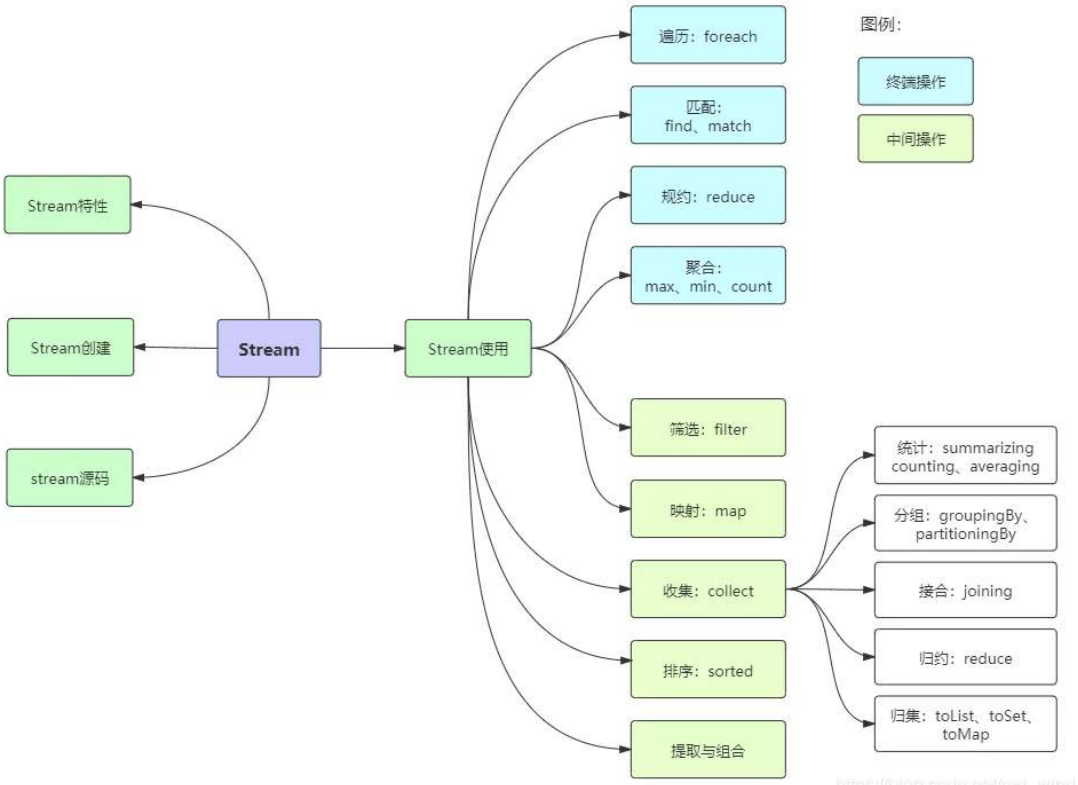
Stream`将要处理的元素集合看作一种流，在流的过程中，借助`Stream API`对流中的元素进行操作，比如：筛选、排序、聚合等。

`Stream`可以由数组或集合创建，对流的操作分为两种：
1. **中间操作**，每次返回一个新的流，可以有多个。
2. **终端操作**，每个流只能进行一次终端操作，终端操作结束后流无法再次使用。终端操作会产生一个新的集合或值。

`Stream`有几个特性
1. stream**不存储数据**，而是按照特定的规则对数据进行计算，一般会输出结果。
2. stream**不会改变数据源**，通常情况下会产生一个新的集合或一个值。
3. stream**具有延迟执行特性**，只有调用终端操作时，中间操作才会执行。

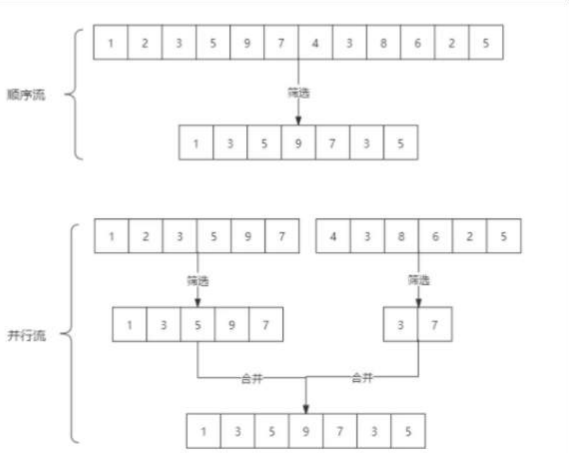
常用方法


nums.stream().filter(num -> num != null).count();
创建Stream
转换Stream
聚合


Stream特性
Stream创建
Stream
stream源码
Stream使用
遍历：foreach
匹配：
find、match
规约：reduce
聚合：
max、min、count
筛选：filter
映射：map
收集：collect
排序：sorted
提取与组合
统计：summarizing
counting、averaging
分组：groupingBy、
partitioningBy
接合：joining
归约：reduce
归集：toList、toSet、
toMap
图例：
终端操作
中间操作

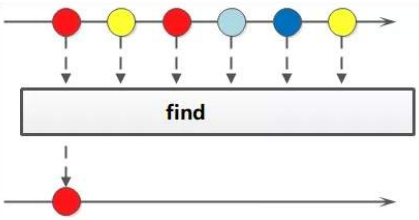
stream和parallelStream

`stream`是**顺序流**，由主线程按顺序对流执行操作，而`parallelStream`是**并行流**，内部以多线程并行执行的方式对流进行操作，但前提是流中的数据处理没有顺序要求。例如筛选集合中的奇数，两者的处理不同之处



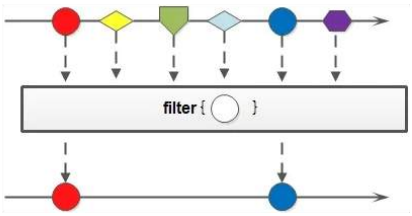
遍历/匹配（foreach/find/match）

Stream`也是支持类似集合的遍历和匹配元素的，只是`Stream`中的元素是以`Optional`类型存在的。`Stream`的遍历、匹配非常简单。



筛选（filter）

筛选，是按照一定的规则校验流中的元素，将符合条件的元素提取到新的流中的操作。

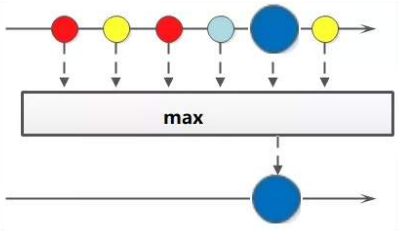


案例一：筛选出Integer集合中大于7的元素，并打印出来

案例二：筛选员工中工资高于8000的人，并形成新的集合。 形成新集合依赖collect（收集）

聚合（ max/min/count)

max、 min、 count这些字眼你一定不陌生，没错，在mysql中我们常用它们进行数据统计。Java stream中也引入了这些概念和用法，极大地方便了我们对集合、数组的数据统计工作。



案例一：获取String集合中最长的元素。

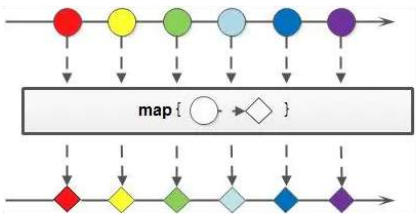
案例二：获取Integer集合中的最大值。

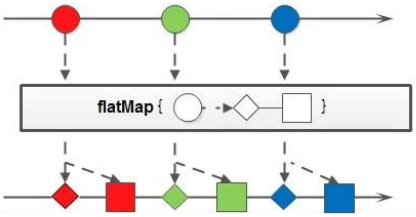
案例三：获取员工工资最高的人。

案例四：计算Integer集合中大于6的元素的个数。

映射(map/flatMap)

- 映射，可以将一个流的元素按照一定的映射规则映射到另一个流中。分为map和flatMap：
- map：接收一个函数作为参数，该函数会被应用到每个元素上，并将其映射成一个新的元素。
- flatMap：接收一个函数作为参数，将流中的每个值都换成另一个流，然后把所有流连接成一个流。





案例一：英文字符串数组的元素全部改为大写。整数数组每个元素+3。

案例二：将员工的薪资全部增加1000。

案例三：将两个字符数组合并成一个新的字符数组。

归约(reduce)

- 归约，也称缩减，顾名思义，是把一个流缩减成一个值，能实现对集合求和、求乘积和求最值操作。

案例一：求Integer集合的元素之和、乘积和最大值。

案例二：求所有员工的工资之和和最高工资。

收集(collect)

collect，收集，可以说是内容最繁多、功能最丰富的部分了。从字面上去理解，就是把一个流收集起来，最终可以是收集成一个值也可以收集成一个新的集合。collect主要依赖java.util.stream.Collectors类内置的静态方法。

- **归集(toList/toSet/toMap)**

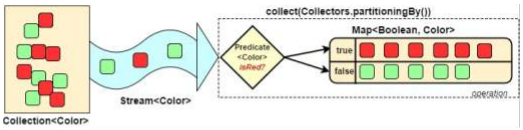
因为流不存储数据，那么在流中的数据完成处理后，需要将流中的数据重新归集到新的集合里。toList、toSet和toMap比较常用，另外还有toCollection、toConcurrentMap等复杂一些的用法。

- **统计(count/averaging)**

Collectors提供了一系列用于数据统计的静态方法：

  - 计数：count
  - 平均值：averagingInt、averagingLong、averagingDouble
  - 最值：maxBy、minBy
  - 求和：summingInt、summingLong、summingDouble
  - 统计以上所有：summarizingInt、summarizingLong、summarizingDouble

- **分组(partitioningBy/groupingBy)**



  - 分区：将stream按条件分为两个Map，比如员工按薪资是否高于8000分为两部分。
  - 分组：将集合分为多个Map，比如员工按性别分组。有单级分组和多级分组。

- **接合(joining)**

joining可以将stream中的元素用特定的连接符（没有的话，则直接连接）连接成一个字符串。

- **归约(reducing)**

Collectors类提供的reducing方法，相比于stream本身的reduce方法，增加了对自定义归约的支持。

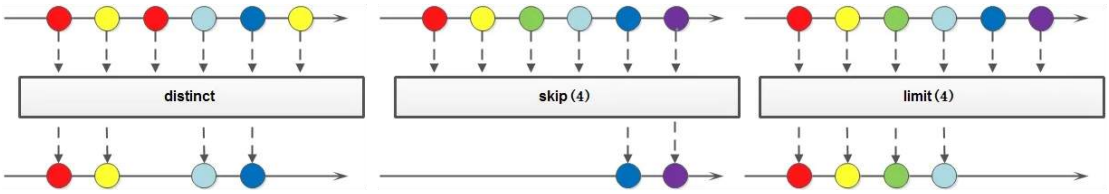
排序(sorted)

sorted，中间操作。有两种排序：

- sorted()：自然排序，流中元素需实现Comparable接口
- sorted(Comparator com)：Comparator排序器自定义排序

案例：将员工按工资由高到低（工资一样则按年龄由大到小）排序

提取/组合(distinct/skip/limit)



流也可以进行合并、去重、限制、跳过等操作。

技术分享：JAVA 锁机制

分享人介绍

姓名：卢镜强

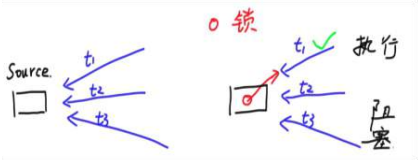
工号：01394891

部门：客户体验研发部

小组：呼叫中心产品研发组

参与顺丰热线和外呼相关系统后端研发，对于java锁，多线程，线程池等有深刻理解。热爱运动，特别是篮球。

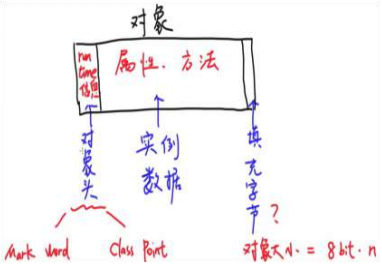
锁机制-什么是锁？



思考：
1、什么是锁？
2、锁在jvm虚拟机中运行时候的作用范围？

如何解决？
1、基于object的悲观锁
2、基于CAS的乐观锁

悲观锁-对象、对象头结构



锁状态	25bit		4bit	1bit	2bit
	23bit	2bit		是否偏向锁	锁标志位
无锁	对象的HashCode		分代年龄	0	01
偏向锁	线程ID	Epoch	分代年龄	1	01
轻量级锁	指向栈中锁记录的指针				00
重量级锁	指向重量级锁的指针				10
GC标记	空				11

Mark Word图

对象结构：
对象头
实例数据
填充字节

对象头结构：
Mark Word
Class Pointter

Synchronized关键字

悲观锁-Synchronized实现原理

通过代码看：monitorenter和monitorexit

monitor：常被翻译为监视器或管程。一个线程进入了monitor，那么其他线程只能等待，只有当这个线程退出，其他线程才有机会进入。

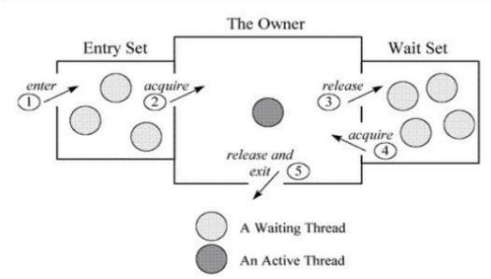


图2.3 Monitor

模拟流程：

1. Entry Set中聚集了一些想要进入Monitor的线程，它们处于**waiting**状态。
2. 假设某个名为A线程成功进入了Monitor，那么它就处于**active**状态。
3. 此时A线程执行途中，遇到一个判断条件，需要它暂时让出执行权，那么它将进入wait set，状态也被标记为**waiting**。
4. 这时entry set中的其他线程就有机会进入Monitor，假设一个线程B成功进入并且顺利完成，那么它可以通过notify的形式来唤醒wait set中的线程A，让线程A再次进入Monitor，执行完成后便退出。

悲观锁-Synchronized锁升级

锁状态	25bit		4bit	1bit	2bit
	23bit	2bit		是否偏向锁	锁标志位
无锁	对象的HashCode		分代年龄	0	01
偏向锁	线程ID	Epoch	分代年龄	1	01
轻量级锁	指向栈中锁记录的指针				00
重量级锁	指向重量级锁的指针				10
GC标记	空				11

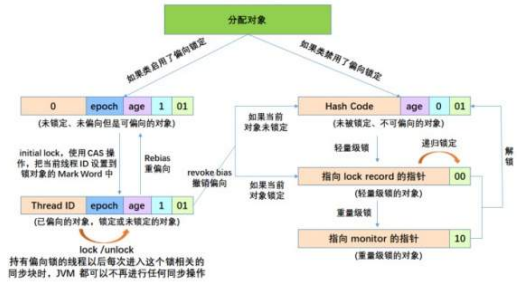


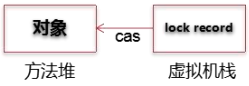
图2.4 对象锁的状态变化

无锁：一是不会出现多线程环境，或者多线程环境也不会出现资源竞争；二是不使用操作系统原语对系统资源锁定，可使用cas；

偏向锁：假如一个对象被加锁，实际运行时候，只有一条线程获取这个对象锁；

轻量级锁：当一个线程获取某一个对象的锁时，假如看到锁标志位00，那么就知道它是轻量级锁；线程会在虚拟机栈区开辟“Lock Record”的空间，实现**线程和对象锁的绑定**；

重量级锁：需要使用monitor来对线程进行控制，使用同步原语来锁定资源



方法堆　　虚拟机栈

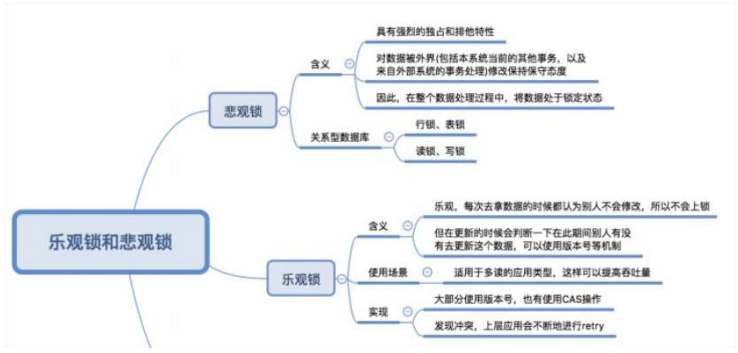
乐观锁-含义/区别

“悲观锁”（Pessimistic Concurrency Control）

权衡悲观锁利弊

一些情况下，同步代码块执行的耗时远远小于线程切换的耗时，用悲观锁就显得不划算。

区别



乐观锁-CAS机制了解

通俗例子

我们现在假设有一间更衣室，房间门上挂着一块牌子，正面是0，反面是1，这块牌子代表房间是否被占用的状态。当显示0的时候，房间为空，谁都可以进入，当显示1时，则代表有人正在使用。在上面这个比喻里，房间就是共享资源，号码牌就是一把乐观锁，人就是线程。

假设此时A和B这两条线程都看到了牌子上显示的是0，于是争抢着去使用房间。但是A线程抢先获得了时间片，他第一个冲进房间并将这块牌子的状态改为1，此时B线程才冲过来，但是发现牌子上的状态已经被改为1，不过B线程没有放弃，不断回来看看牌子变回0了没。

```C++
int cas(long *addr, long oldValue, long newValue)
{
    /* Executes atomically. */
    if(*addr != old)
        return 0;
    *addr = new;
    return 1;
}
```

CAS函数

思考：CAS分为compare和swap两步操作，多线程下难以保证原子性？

乐观锁-CAS机制原理

是否存在漏洞？

因为看上去这个CAS函数本身没有进行任何同步措施，似乎还是存在线程不安全的问题。比如A线程看到牌子的状态是0，伸手去翻的一瞬间，很有可能B线程突然抢到时间片，将牌子翻成了1，但是线程A不知情，也将牌子翻到了1，这就出现了线程安全问题，AB线程同时获得了资源，好比两个人进入了更衣室，非常尴尬。

如何实现原子性？

那么，如何实现CAS的原子性呢？所幸的是，各种不同架构的CPU都提供了指令级的CAS原子操作，比如在x86架构下，通过cmpxchg指令支持CAS，在ARM下，通过LL/SC来实现CAS。也就是说，既然CPU已经原生地支持了CAS，那么上层进行调用即可。现在，除了通过操作系统的同步原语（比如mutex）来有锁地实现线程同步（悲观），通过CAS的方式我们能实现另一种无锁的同步机制（乐观）。

乐观锁-java编程

假设有一个简单的需求，你需要使用3条线程，将一个值，从0累加到1000，你该怎么做?

```
public static Integer num = 0;

    public static void main(String[] args) throws InterruptedException {
        for (int i  = 0; i < 3; i++) {
            Thread t = new Thread(new Runnable() {
                @Override
                public void run() {
                    while (num < 1000) {
                        num++;
                        System.out.println(Thread.currentThread().getName() +
":" + num);
                    }
                }
            });
            t.start();
        }
    }
```

重中之中-AQS

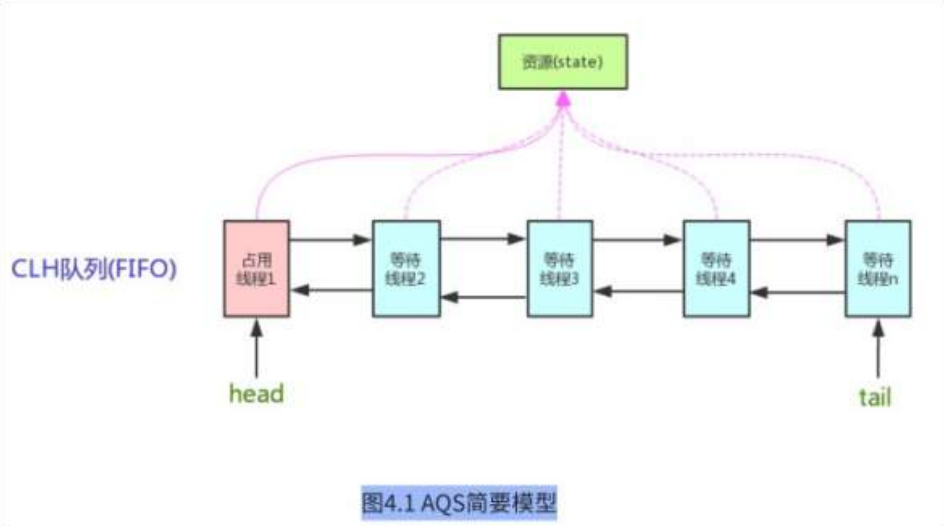


图4.1 AQS简要模型

思考：AQS如何设计？

1）如何锁定共享资源？ -->共享资源标志位

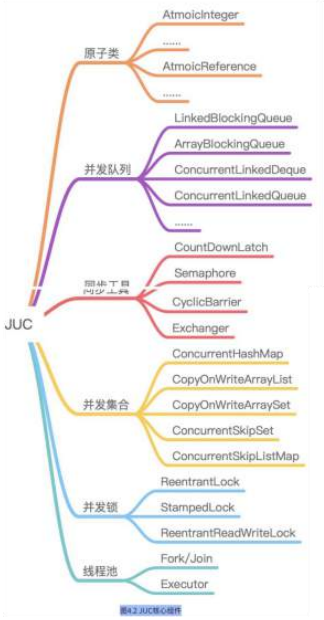
2）拒绝其他线程如何设计？ --> 双向队列

锁机制-juc核心组件

总结

基于乐观锁的CAS机制以及AOS阻塞队列，促进上层产生多种多样无锁编程的思路，以及诞生各种好用的工具。这些通过CAS来实现同步的工具，由于不会锁定资源，而且当线程需要修改共享资源的对象时，总是乐观认为对象状态没有被其他线程修改过，每次自己都会区compare状态值。

了解完CAS机制+AQS阻塞队列，可以尝试去领悟JUC核心组件啦～



微创新：一种基于 h5 的可配置可嵌入多终端销售工具的设计和开发方案

赛道：研发工具类
成员：裴玲玲　黄金玉　苏德仁　金天璋　董知灵　范锐　戴金森　于昕

简介

本文基于 html5 和低代码技术，提出了一种可配置可嵌入多终端的销售工具。在减少开发成本的同时，该销售工具能够更灵活地满足企业的销售管理需求。

背景

一个灵活、实用、真正适合企业现状的销售工具能够最大化地助力企业管理销售终端，提高销售业绩。目前市场上广泛使用的销售工具主要包括销售管理系统和销售管理软件，但是，pc 端网站不能随时随地打开使用，app 需要注册安装才能操作，因此，无论哪一种，其获取和使用都不够便捷。此外，现有销售工具页面中包含的表单项较多，且表单项固定，页面内容的调整需要前后端人员共同配合，开发成本大，可配置性不强，在适配不同企业的销售需求方面也有所不足。

痛点

1.现有销售工具的获取和使用均不够便捷；

2.现有销售工具的表单内容固定，配置性较差，且开发成本大；

3.现有销售工具大多不支持销售人员辅助客户下单。

主要解决方案

针对以上痛点，本方案提出了一种基于 html5 的可配置可嵌入多终端销售工具的设计和开发方案，技术实现方案及流程图如下：

一 技术实现方案

1.销售工具在具体应用中嵌入方案的设计

销售工具可嵌入到多个终端的不同应用中，在具体应用中，销售工具可以以不同的身份嵌入，例如微应用、数据看板、业务助手、openAPI 等。需要根据不同应用的接入方式定义不同的配置信息，同时，需要考虑如何实现配置信息的通用定义。

2.销售工具在在具体应用的鉴权处理

将销售工具部署到具体应用时，可配置用户权限。部署完成后，在应用中会根据用户权限，初始化展示销售工具入口。

3.销售工具与具体应用的数据交互方案设计

应用自身提供的 API 一般可以获取到当前设备信息、应用基本信息、用户基本信息，以及图片、视频等资源信息，销售工具需要按照一定的规则去使用 API，以获取这些信息。此外，还需要考虑在具体应用中，销售工具自身接口的调用。

4.销售工具与具体应用的消息同步方案设计

需要考虑销售工具的订单状态改变时，在具体应用中信息同步通知的实现。例如，在丰声中，数字商店销售工具的消息会通过业务助手工具推送给销售人员。

5.表单模板的动态化配置方案设计

首先，需要新建模板；其次，需要定义模板及模板配置规则；然后，需要对各个表单项配置具体行为；此外，还需要对各个表单项的样式进行设计；最后，部署该模板，便可以在销售工具中直接应用。

6.销售工具功能模块设计

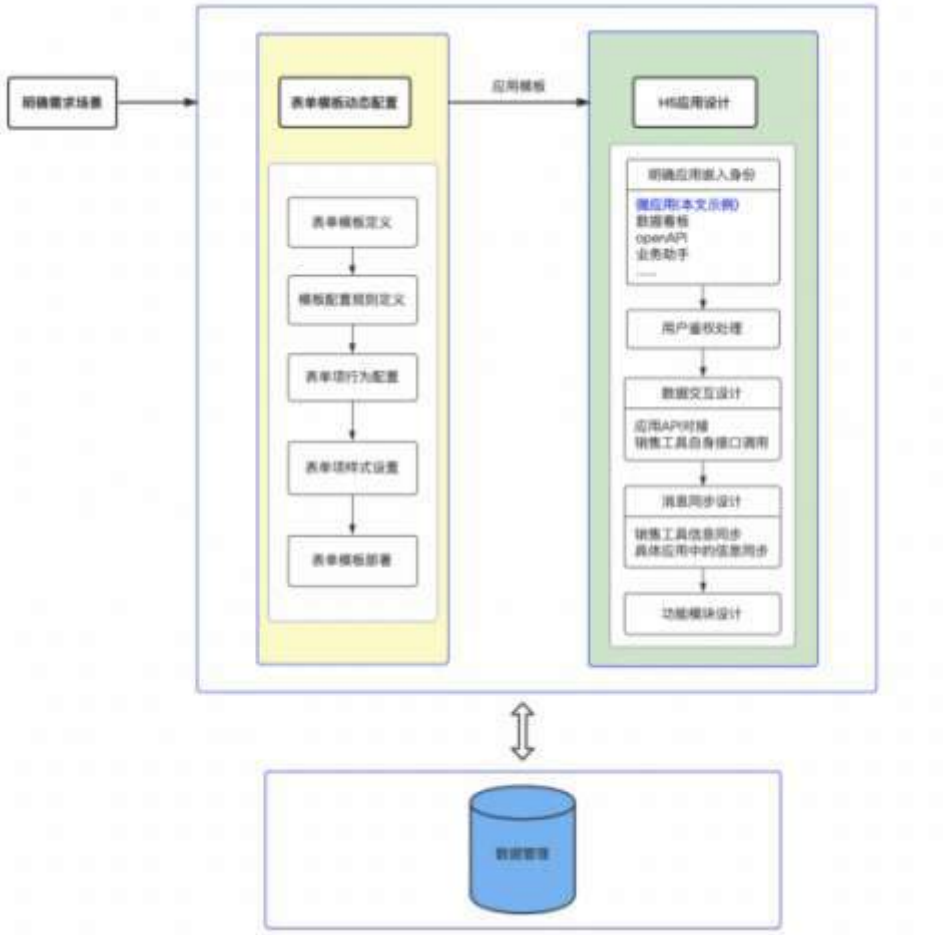
(1)销售推广。销售人员可以选择对应商品生成推广码链接或二维码，此链接或二维码包含销售人员工号、姓名及推荐的商品信息。销售人员把此链接或二维码发送给客户，客户通过此链接或二维码访问，由客户自主进行商品浏览或下单。

(2)创建订单。在销售工具首页，提供下单入口。下单时，销售人员选择产品并输入客户手机号后，可选择该客户所能购买的产品规格和产品周期，选择完成后，可以直接生成订单，也可以进行调价或添加定制计费项后生成订单。创建成功后，客户在客户侧系统进行支付，完成产品交易。

(3)订单列表。展示销售人员创建的所有订单，包含待支付订单、交易成功的订单，和已关闭状态的订单。

(4)订单详情。销售人员查看代客户创建订单的详情信息。

二 技术实现流程图



应用示例

1.数字商店销售工具在【丰声】中的应用

将销售工具部署到【丰声】时，可配置用户权限。部署完成后，在丰声-微应用中会根据对应权限，初始化展示销售工具入口，点击可进入销售工具首页。

2. 创建订单

首先，选择所要购买的产品，然后填写并校验客户手机号。产品名称和客户手机号校验通过后，会展示标准计费模块，可以选择对应的产品规格和产品周期，需要注意的是，产品规格和产品周期是一一对应的关系，只有选择完产品规格，才可以选择产品周期。产品规格和使用周期确定后，页面中会展示订单总价，在只有标准计费项，且不调价时，订单总价等于产品标准费用。此外，销售人员可以在标准计费的基础上进行调价，调价后金额不能低于

标准费用的 50%，否则不能成功创建订单。调价校验通过后，订单总价等于调价后的金额。如下图所示：


2:49
顺丰数字商店-...
选择产品
销果裹
客户手机号
13667212139
合同方式
用户协议
标准合同
客户名称
非必填
业务区
非必填
推荐人
01398521
标准计费
选择规格
高阶版
选择时间
1个月(30天)
是否调价
金额：￥39.00
调价后金额(元)
15
不得少于标准计费金额的50%
定制计费
订单生成后，客户可在数字商店中看到此笔支付订单
总价：￥0.00
生成订单

1. 表单模板的动态配置

（1）新建模板：选择模板的应用平台

（2）定义模板及模板配置规则

首先，需要设计组成模板的各个内容区块，以及内容区块中所包含的表单项；然后，需要定义每个表单项的配置规则，例如表单的布局、是否禁用、是否校验、是否显示清空按钮等。


组件
区块
请输入内容
组件库
热门
文本
表单
表单容器
表单输入
表单多行输入
表单选择
表单单选
表单多选
大纲树
可搜索组件名称 / ID
普通容器
标题
表单容器
EVT
内容插槽
表单选择
表单电话
表单日期
表单输入
表单开关
表单选择
表单选择
表单单选
表单开关

属性 样式

表单选择

组件ID id5

基础属性

* 字段绑定 formSelect

表单提交数据的 Key，页面内需保证唯一，低代码中可根据该字段，控制当前组件的赋值、校验等操作

字段标题 选择产品

显示标题

选中值

占位符 请选择产品

选项列表

+

高级属性

布局方式 水平

是否必填

是否显示必填...

必填校验错... 该项为必填项

（3）表单项行为配置

创新点

该销售工具支持在多个终端的不同应用中嵌入，获取和操作较为便捷；

该销售工具支持表单内容的动态配置，开发成本低、灵活性强；

该销售工具支持销售人员直接下单，提升了用户的下单体验；

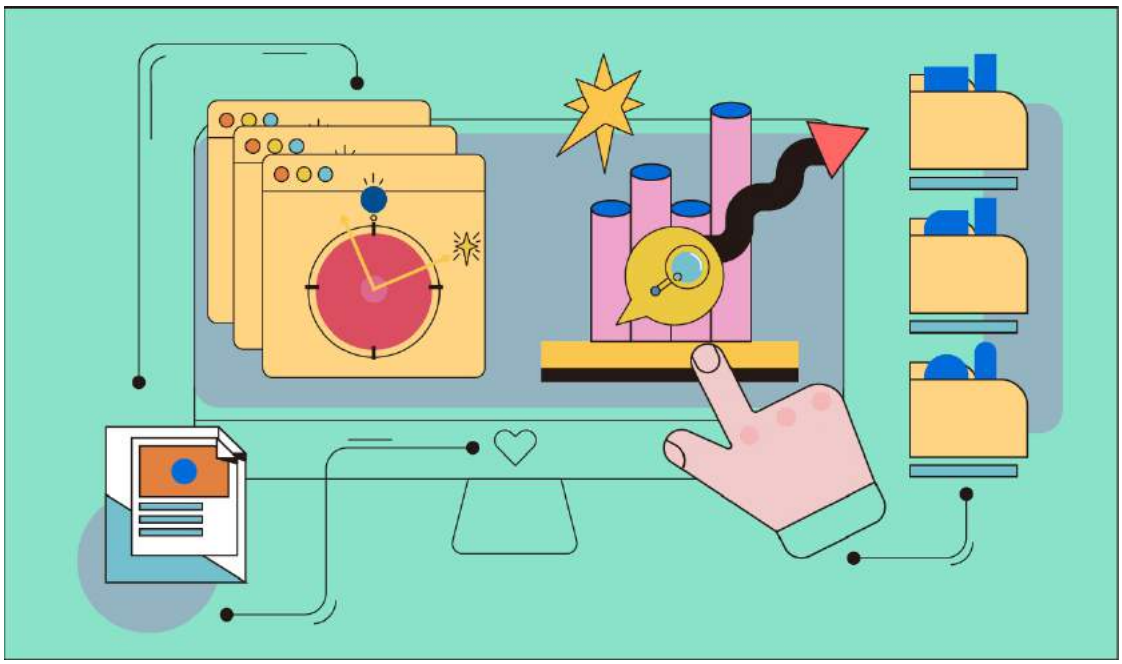
该销售工具支持销售人员对产品进行调价或增加定制计费项，费用调整更灵活。

带来影响

该销售工具的下单、查单、生成推荐链接等功能已应用于顺丰科技数字商店销售工具，且已部署到丰声微应用中表单动态配置功能（低代码），组内在启动研发中。

微创新：强互动类营销创意体验前端解决方案

赛道：研发项目类
成员：康妮　甘仲欢　戴金森　王任悦　廖翊伽　杜娟　陈健芬　李振



简介

使用数据与场景结合以故事化镜头叙事的复杂动画交互的形式，提升用户体验，加强用户联系的同时为平台起拉新裂变效果

背景

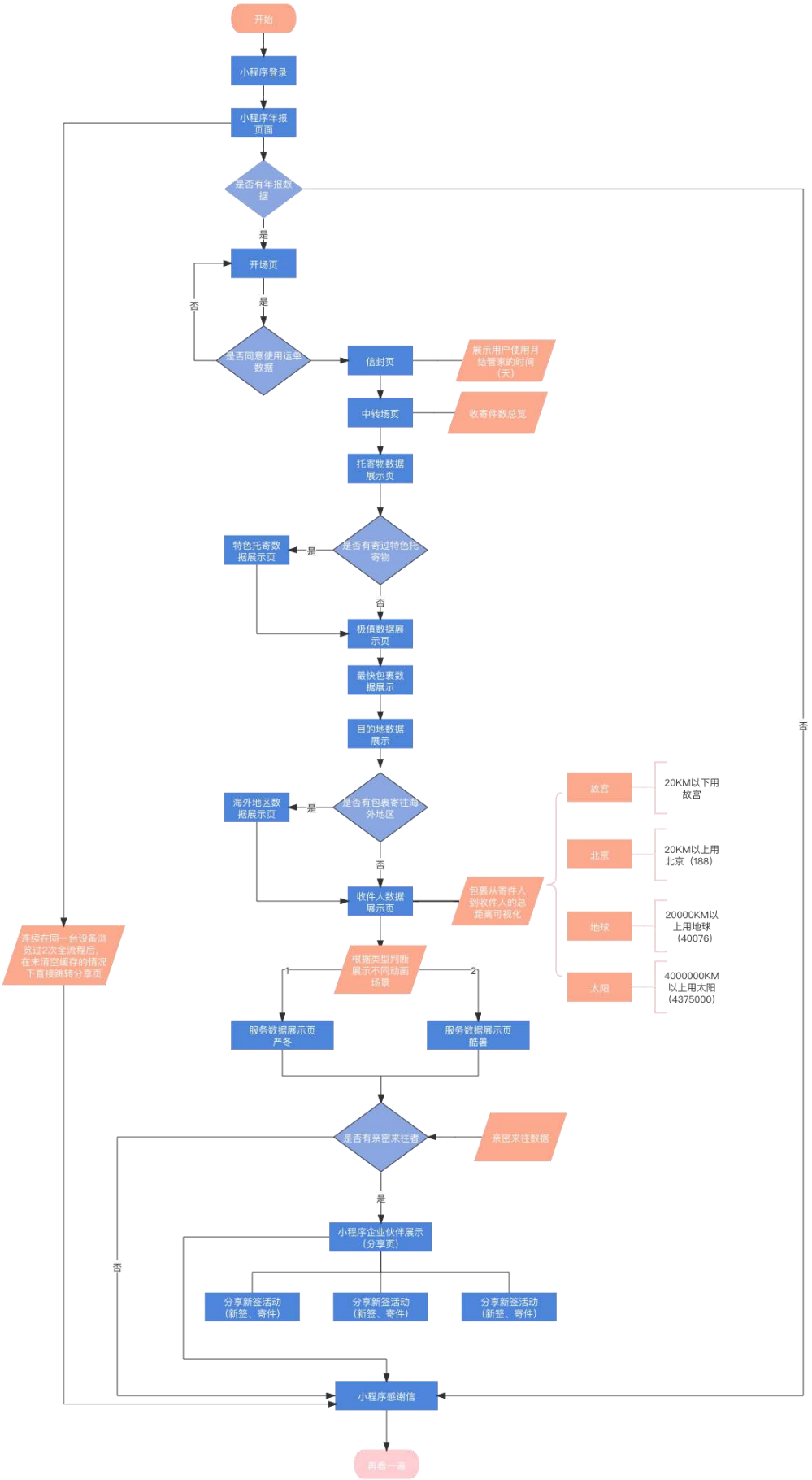
随着智能终端设备的多样化及业务的快速发展，动画被广泛需要并应用在各类业务场景中，一个高质量的动画在丰富用户体验、提升业务价值方面至关重要。然而，目前痛点是常见的 html5 动画大多使用图片作为背景图，通过 css3 实现简单的动画效果，场景不够丰富；或者没有将场景与数据较好地结合，动画交互性不强，用户体验较差；复杂动画往往伴随着资源体积大，加载慢，影响用户观感。

基于以上问题，本文提出结合场景与数据的动画设计方案，通过运镜的方式实现动画中数据与场景的强关联。在丰富动画交互体验的同时，唤起用户回忆、增强品牌形象，拉动业务增长，较好地实现了动画赋能的效果。此外，通过资源自由组合的预加载的方式，减少了用户的等待时长，增强了动画的交互体验。

解决方案

产品侧：

以故事性形式讲述一个包裹的全生命周期，从包裹寄出地至目的地整个运输过程中经过的地点、接触的人或物，每到一个站点或场景下，角色讲述相应的年度回顾内容。流程图如下：


开始
小程序登录
小程序年报页面
是否有年报数据
是
开场页
是
否
是否同意使用运单数据
信封页
展示用户使用月结管家的时间（天）
中转场页
收寄件数总览
托寄物数据展示页
是否有寄过特色托寄物
是
特色托寄数据展示页
否
极值数据展示页
最快包裹数据展示
目的地数据展示
是否有包裹寄往海外地区
是
海外地区数据展示页
否
收件人数据展示页
包裹从寄件人到收件人的总距离可视化
故宫
20KM以下用故宫
北京
20KM以上用北京（188）
地球
20000KM以上用地球（40076）
太阳
4000000KM以上用太阳（4375000）
否
连续在同一台设备浏览过2次全流程后，在未清空缓存的情况下直接跳转分享页
根据类型判断展示不同动画场景
1
2
服务数据展示页 严冬
服务数据展示页 酷暑
是否有亲密来往者
亲密来往数据
是
小程序企业伙伴展示（分享页）
否
分享新签活动（新签、寄件）
分享新签活动（新签、寄件）
分享新签活动（新签、寄件）
小程序感谢信
再看一遍

技术侧：动画交互以运镜的方式对多场景动画进行呈现。

1、多场景、多动效动画的设计

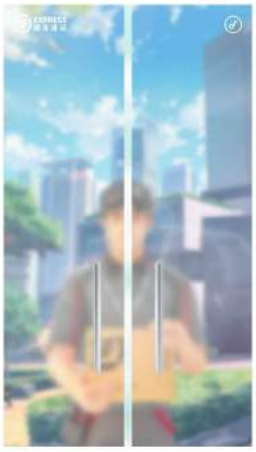
物流数据以包裹为主角开始的奇妙旅程，动画使用“运镜”技巧把包裹从寄出地至目的地整个运输过程中经过的地点、接触的人或物，每到一个站点或场景下，讲诉相应的回顾内容。

包裹发出：

数据视觉视化包裹中转场：

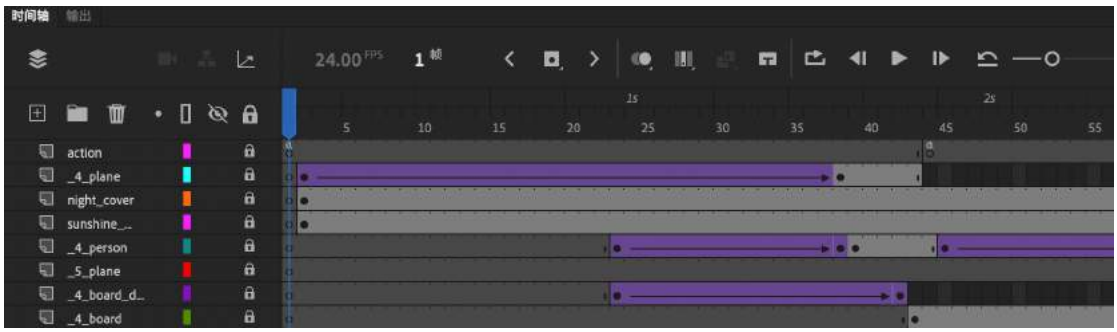
物流数据可视化多场景：

动态展示极端天气下小哥派件服务：





基于以上复杂动画可以使用 Adobe Animate cc + 代码相结合，快速实现

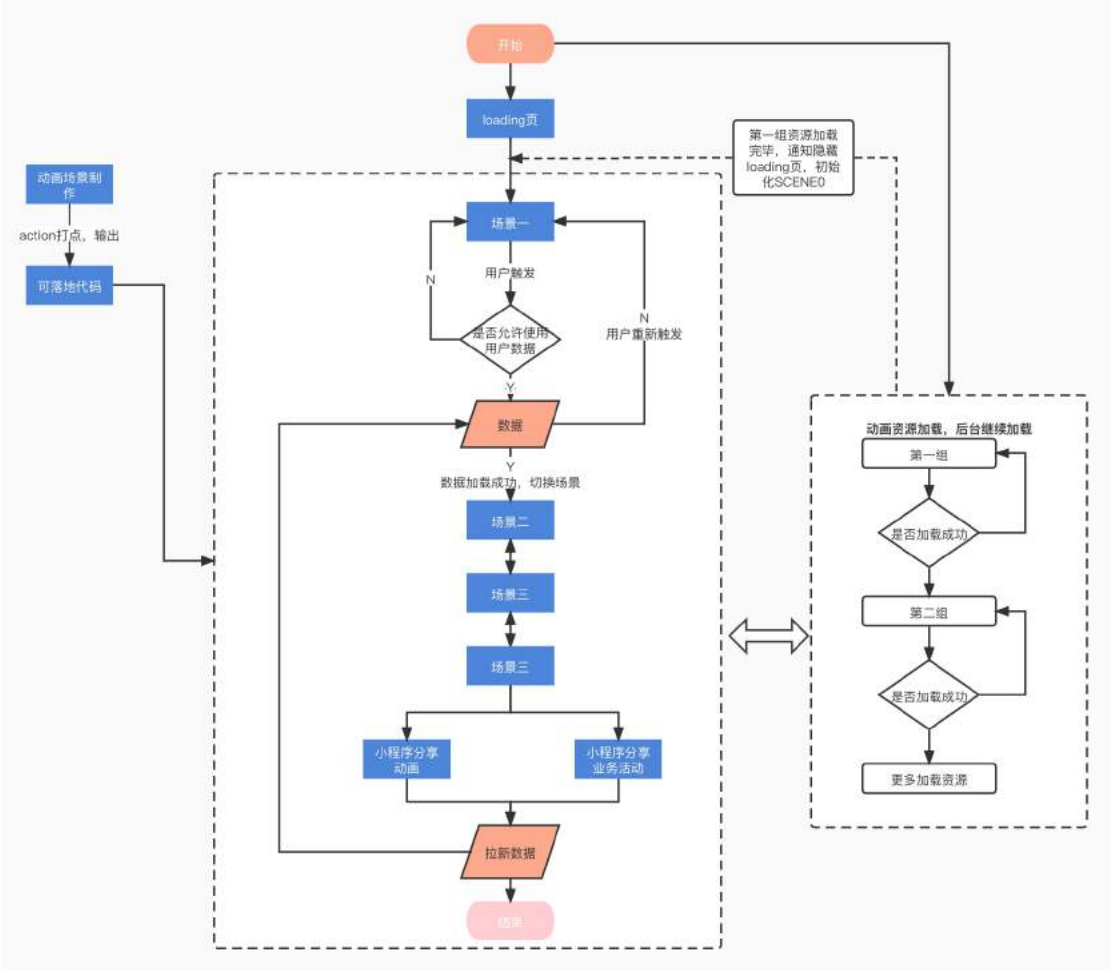



```
(function (cjs, an) {
var p;
var lib={};var ss={};var img={};
lib.ssMetadata = [
        {name:"report_p3_atlas_1", frames: [[784,0
        {name:"report_p3_atlas_2", frames: [[0,0,9
        {name:"report_p3_atlas_3", frames: [[0,824
        {name:"report_p3_atlas_4", frames: [[0,0,1
        {name:"report_p3_atlas_5", frames: [[0,856
        {name:"report_p3_atlas_6", frames: [[752,8
        {name:"report_p3_atlas_7", frames: [[0,529
];
```

2、动画资源预加载及重新加载方案的设计

使用

createjs.



LoadQueue 方法封装预加载资源公共方法，支持自由组合资源加载，在完成第一组资源加载后，用户即可体验动画，后台继续加载其他资源直至资源加载完毕。

3、场景切换时动画衔接方案的设计

（1）场景切换平滑顺畅

```
createjs.Tween.get(this.curScene.view)
  .to({ alpha: 0.2 }, 400)
  .call(() => {
    if (this.curScene) this.curScene.toRemove();
    this.createNewScene(v);
    this.refreshScene();
  });
```

（2）当前场景包含多个叙事场景时，即一个 SCENE 对象多个叙事场景，在叙事场景的出场、循环帧分别打点，如下代码

```
        if (
            this.view.currentFrame === 43 ||
```

```
            this.view.currentFrame === 171 ||
            this.view.currentFrame === 299
        ) {
            this.view.stop();
        }
```

通过用户触发再控制动画演出，如代码：this.view.gotoAndPlay();

（3）当前场景含有主场景和子场景，数据控制主场景以及子场景，如下图：

3-3-1

3-3-2

3-3-3

当用户数据没有 3-3-2 的数据时，当前的主场景和子场景都需要通过跳帧的方式划到下一页，同时需要注意调整需要先把 3-3-1 的动画 this.view.play()出场的最后一帧 this.view.stop()，然后再跳帧 this.view.play(3-3-2 的入场帧)，细腻地控制动画场景切换；

（4）多个场景

根据数据来决定动画场景，但是有些元素又是共有的，那么同一个场景中可以设计不同的 SCENE 动画，比如上述描述的动态展示极端天气下小哥上门派件，可如下方式实现：


```
if (v === 5 && reportData) {
  sceneName = "SCENEWINTER";
  if (reportData.temperatureType === 2) sceneName = "SCENESUMMER";
}
if (!window["config"][libName]) return;
const page = new window["config"][libName][sceneName]();
```

4、动画赋能具体方案的设计

基于小程序分享的功能，除了分享动画之外，还加入分享业务活动，分享人和被分享人都有对应的奖励，最后把基于业务数据的动画互动，促进业务数据，形成闭环。

创新点

1、高效实现复杂动画动效，故事化镜头叙事，使得动画更加吸引力；

2、动画输出可落地代码，研发直接使用；

3、关键帧、逻辑帧 action 打点控制动画演出；

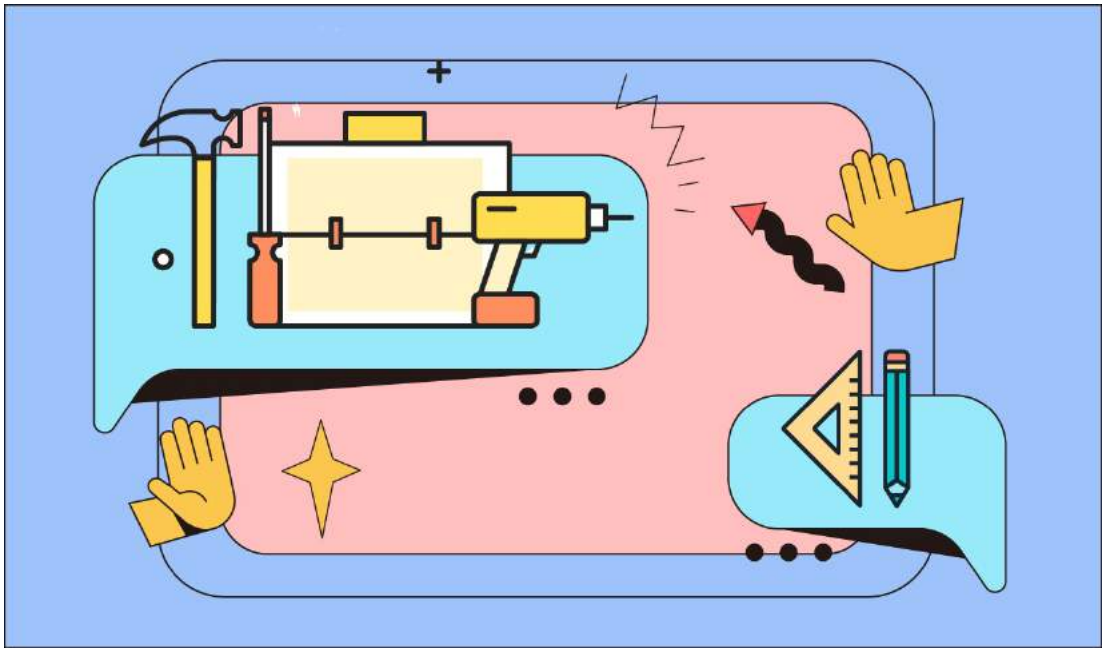
4、视觉及动效呈现细腻有品质感，操作体验舒适，动画最后结合业务实现拉新；

业务影响

目前已经应用在年报回顾项目，以包裹的运输过程的每一个场景来讲述年报回顾内容，动画采用镜头实现，动画细腻有吸引力，用户观感强。

微创新：基于海量业务日志的系统实时监控

赛道：研发工具类
成员：刘志永　魏维　曾云辉　刘鹏　刘斯杰　张鑫　孙珺毅



简介

基于目前涉及上下游系统众多且流量访问巨大的情况，在现有实时日志系统的基础上构建系统实时监控，从而提高线上生产问题定位与分析能力。

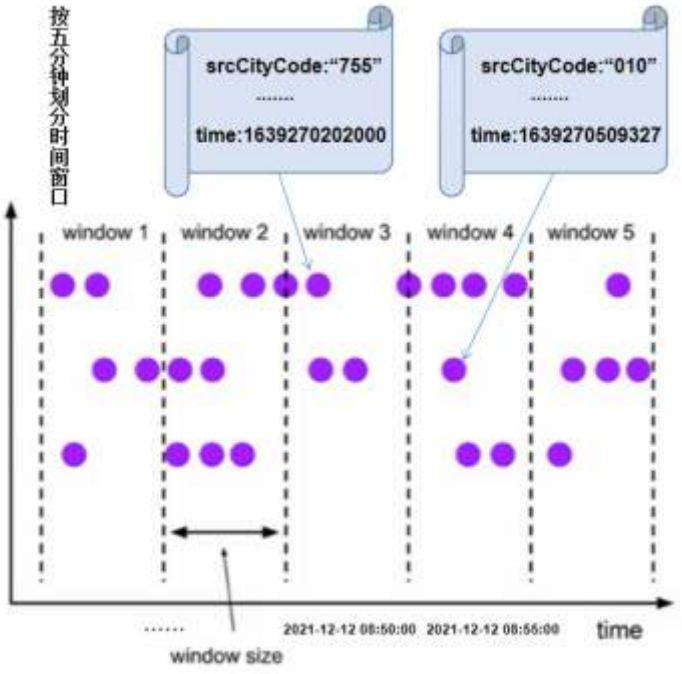
背景

产品目录系统目前对接的上下游系统众多，并且系统所提供的产品推荐、服务推荐等核心接口每日具有极高的访问量，因此提供 P90、P95、P99 等服务响应时间指标来衡量系统接口性能，并统计关键接口的调用量是十分有必要的。另一方面，产品目录系统的推荐功能与一线

小哥的收派息息相关，因此通过构建实时监控可以有效统计目标产品、服务以及各管控点的调用量，从而为系统的平稳运行提供保障。

解决方案

作为一个成熟的流处理框架，Flink 所带有的时间窗口以及水位线机制十分适合构建实时监控。通过设置不同尺度的时间窗口，将产品目录系统的日志数据依照自身时间戳大小落入不同的窗口中，之后通过提取关键属性维度与聚合计算等操作可以很方便的自定义统计各类指标。



通过对业务日志的映射和聚合等操作，按时间窗口维度对外实时输出监控指标信息，并存入 MySQL 数据库中以便后续可视化展示。

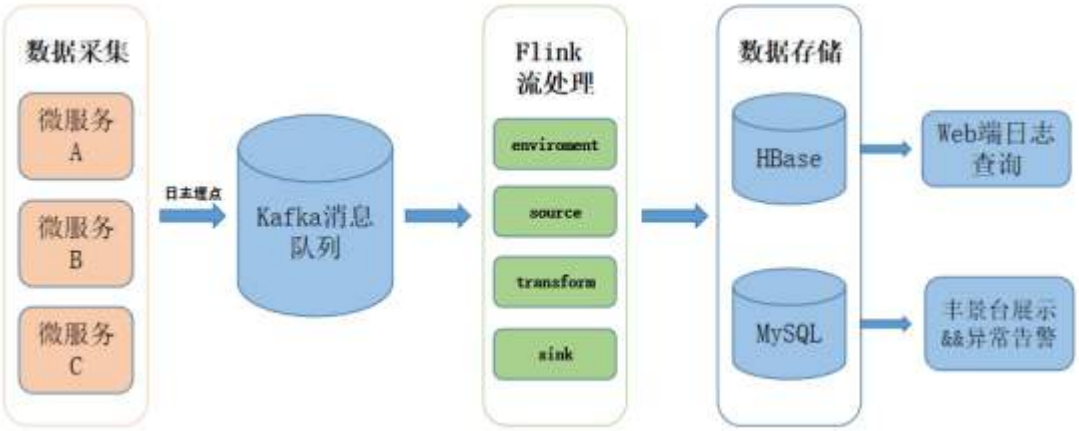
```
public static void main(String[] args) throws Exception {
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.setStreamTimeCharacteristic(TimeCharacteristic.IngestionTime);
    env.enableCheckpointing(60000L);
    env.getCheckpointConfig().enableExternalizedCheckpoints(CheckpointConfig.ExternalizedCheckpointCleanup.RETAIN_ON_CANCELLATION);
    env.getCheckpointConfig().setMaxConcurrentCheckpoints(1);
    env.getCheckpointConfig().setMinPauseBetweenCheckpoints(500L);
    env.getCheckpointConfig().setCheckpointTimeout(30000L);
    env.setStateBackend(new FsStateBackend("hdfs:///data/flink/checkpoints/event"));
    // 设置kafka的基本信息
    final ParameterTool myProp = ParameterTool.fromPropertiesFile(Resources.getResourceAsStream(CommonConst.MY_PROP_PATH));
    String topic = ConfigurationManager.getProperty(Constants.KAFKA_TOPICS);
    Properties prop = new Properties();
    prop.setProperty(Constants.KAFKA_BOOTSTRAP_SERVERS, myProp.get(Constants.KAFKA_BOOTSTRAP_SERVERS));
    prop.setProperty(Constants.KAFKA_GROUP_ID,"HCS_FREE_FORE_EXIT");
    prop.setProperty(Constants.KAFKA_ZK_CONNECT,myProp.get(Constants.KAFKA_ZK_CONNECT));

    FlinkKafkaConsumer08<String> stringFlinkKafkaConsumer08 = new FlinkKafkaConsumer08<>(topic, new SimpleStringSchema(), prop);
    stringFlinkKafkaConsumer08.setStartFromLatest();

    DataStreamSource<String> test = (DataStreamSource<String>) env.addSource(stringFlinkKafkaConsumer08).name("FREE_FORE_VAS_LOG");
    // 统计每日流量
    test
            .flatMap(new LineSplitter()).name("LineSplitter")
            .keyBy(0)
            .timeWindow(Time.minutes(Long.valueOf(myProp.get(Constants.TIME_INTERVAL))))
            .process(new RecommendCountProcess())
            .writeUsingOutputFormat(new KafkaMysqlFormat()).name("countFormat");

    // 统计接口耗时
    test
            .flatMap(new CostTimeSplitter()).name("CostTimeSplitter")
            .keyBy(0)
            .timeWindow(Time.minutes(Long.valueOf(myProp.get(Constants.TIME_INTERVAL))))
            .process(new CostTimeComputeProcess())
            .writeUsingOutputFormat(new CostTimeFormat()).name("CostTimeFormat");
    env.execute();
}
```

产品目录系统基于实时业务日志构建的监控系统架构如下所示：



创新点

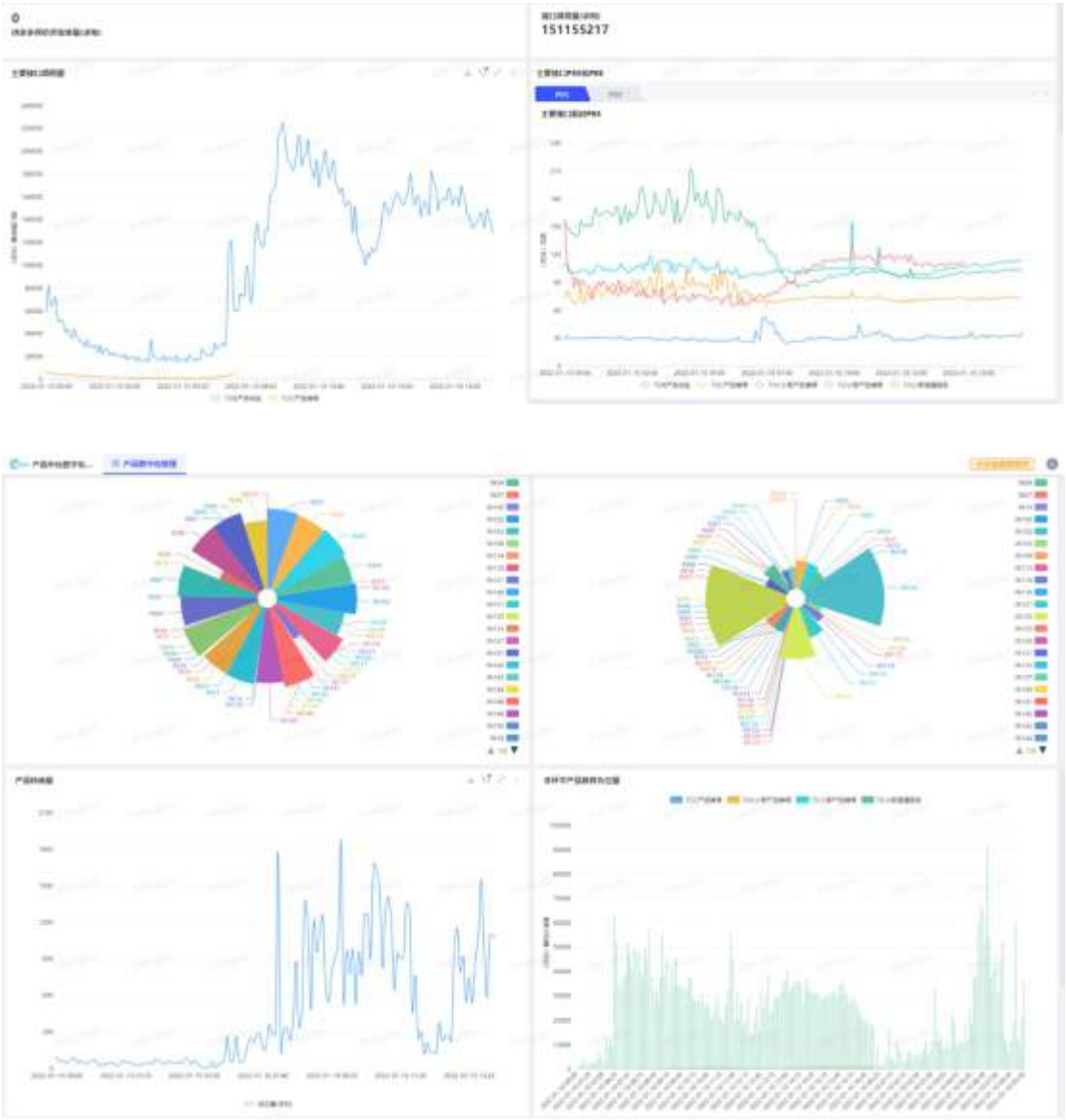
基于实际业务场景，提供实时系统监控，提高线上分析能力

通过引入可配置的方式自定义各类监控指标，增强实时数据数据的灵活性

业务影响

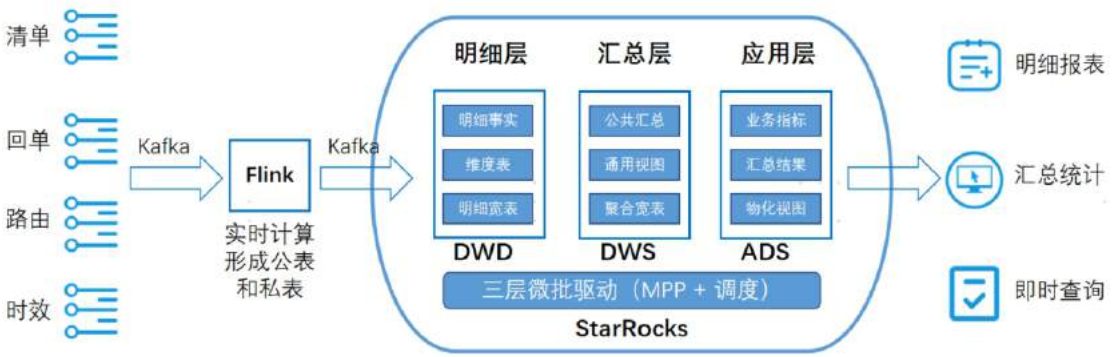
对线上生产进行监控，提供实时告警

通过可视化数据助力生产问题分析



微创新: Flink+StarRocks，专项客服极速新体验

赛道：研发项目类
成员：严向东　刘浩　董念　曾庆东　谢行　宋宇辰　孙林果　黄松



简介

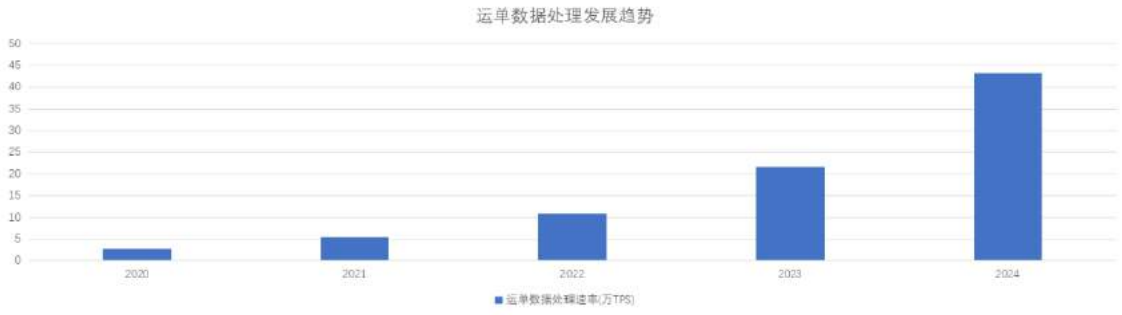
应用 Flink 高吞吐低延迟高性能技术特性，结合新一代 StarRocks 极速全场景 MPP 数据库高速读写性能，升级专项客服业务数据计算速度和处理能力，助力公司战略客户经营。

背景

当前，专项客服存在 2 个比较严峻的问题：

1.专项客服平台对接了清单、回单、路由、PIS 承诺时效等 Kafka 海量运单数据，尤其是 2020 双十一高峰日运单量相对平常剧增近 7 倍，消费出现积压（最长积压达 5 小时），严重影响专项客服人员正常业务开展。

2.专项客服数据一式两份，一份存 Oralce，一份存 BDP 数仓，两边数据存在同步差异，丰景台报表数据要滞后客户端 2~3 小时。未来，专项客服平台大客户运单数量依旧会逐年维持高速增长。平台数据运算速度和吞吐能力亟待快速提升！



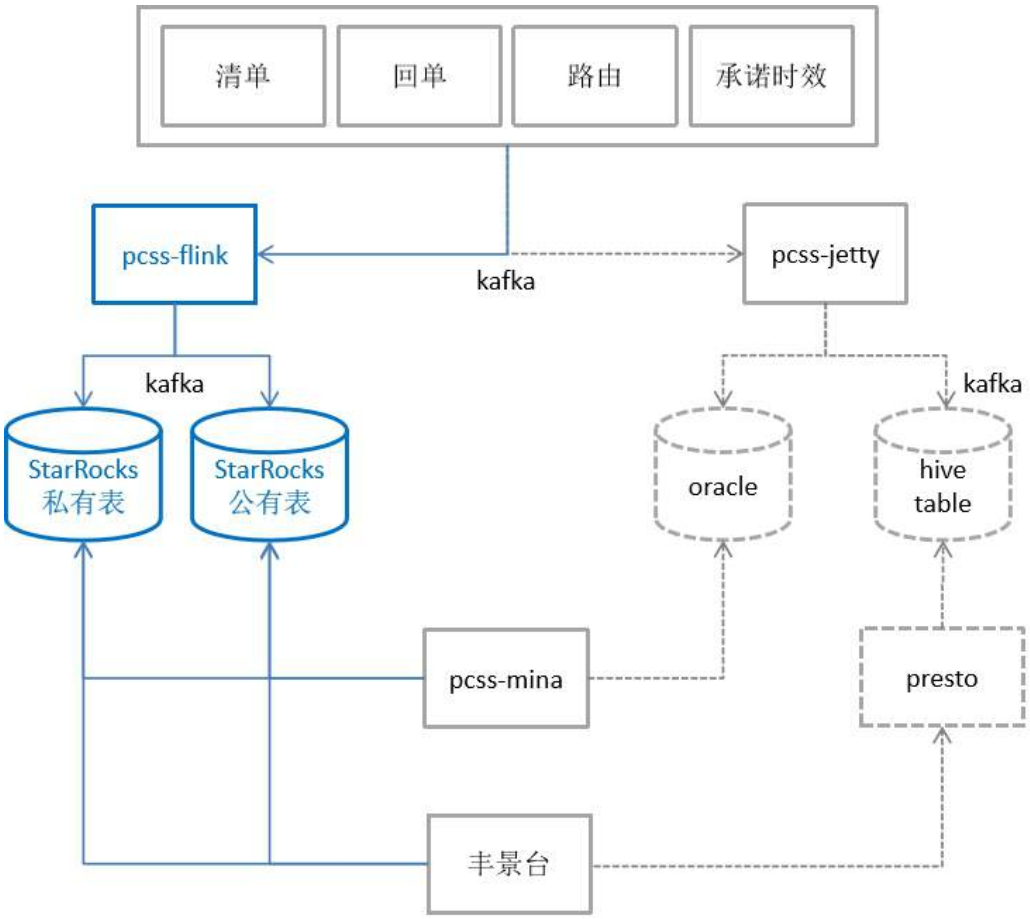
解决方案

PCSS 原有架构基础上进行迭代优化

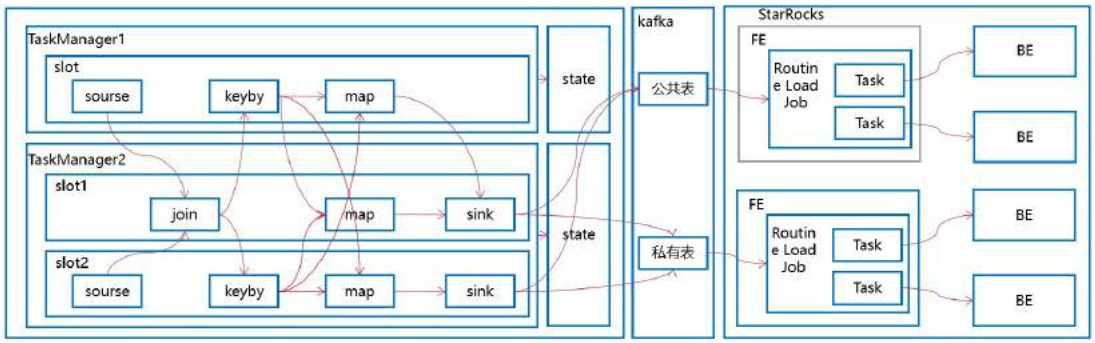
a.引入 StarRocks(也称作 DorisDB)大数据存储,整合 oracle 和 hive 分别落库为 StarRocks 统一存储，规避两处数据不同步的问题

b.分离出公有表数据（存储全网清单、回单、路由和承诺时效原始无加工字段），可开放共享给中心内外其他系统接入

c.引入 Flink 实时计算替代传统 Java 应用，提升数据运算吞吐能力，保证运算低延迟，提升系统稳定性和可扩展性



备注：蓝色为新增组件，虚线部分被替代组件



FLINK深度调优

1. join的使用,state的共享
2. state的应用，state持久化策略，保存时间策略
3. 每个算子环节的并行度，tm数量，slot数量的精心考量
4. kafka版本，分区

StarRocks推动社区发展

1. 推动解决kafka导数性能问题
2. 推动解决了当kafka消费情况下，删除表崩溃问题，开源版已经修复补丁包
3. 推动解决全量查询时BE节点OOM问题
4. 解决重启BE的routine load时，报too many tasks问题
5. 推动社区版支持kafka下发字段局部更新

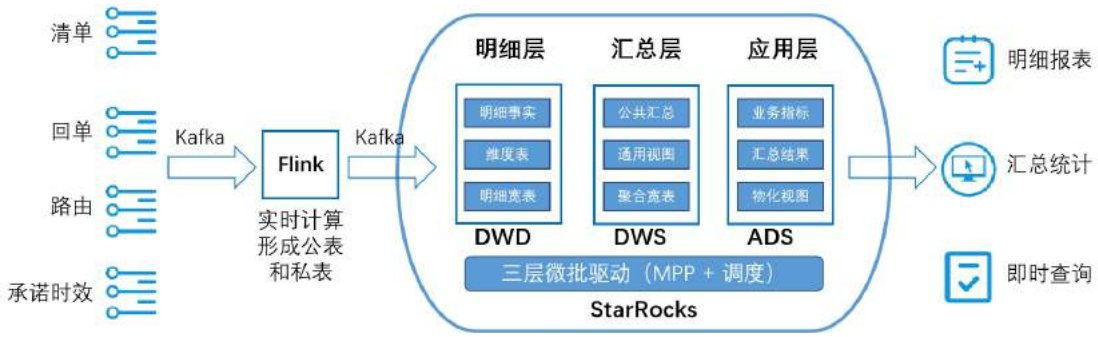
充分利用 StarRocks MPP 存储特性：

a. StarRocks 支持多种数据分析场景，支持多种数据模型(明细模型、聚合模型、更新模型)，多种导入方式（批量和实时），可整合和接入多种现有系统(Spark、Flink、Hive、ElasticSearch)。后续可根据实际业务场景扩展满足各种自定义统计需求。

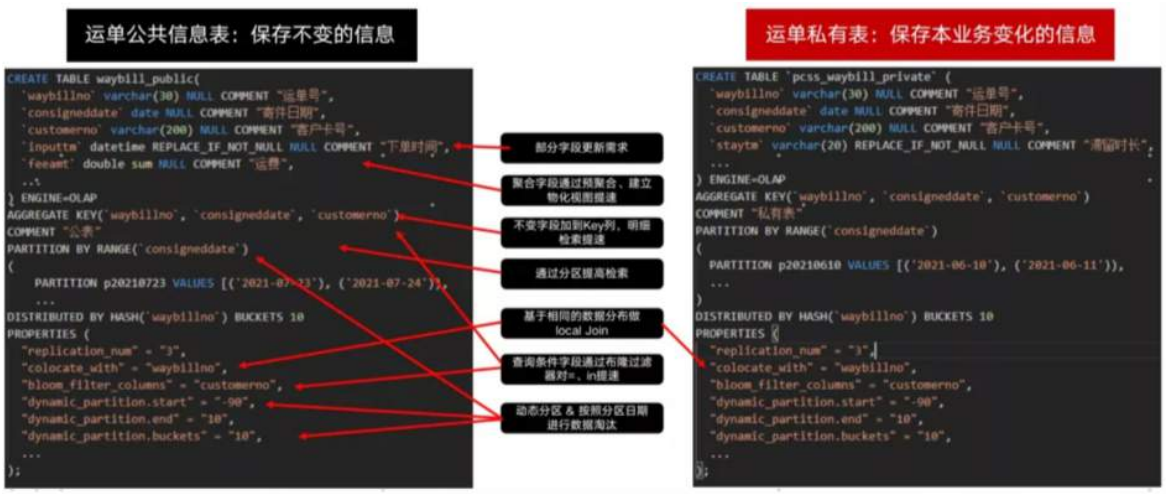
b. StarRocks 兼容 MySQL 协议，可使用 MySQL 客户端和常用 BI 工具对接 StarRocks 来进行数据分析，研发测试运维可快速无障碍上手。

c. StarRocks 的分布式架构，对数据表进行水平划分并以多副本存储。集群规模可以灵活伸缩，能够支持 10PB 级别的数据分析；支持 MPP 框架，并行加速计算；支持多副本，具有弹性容错能力。可满足业务持续快速增长的需求。

d. StarRocks 支持关系模型，使用严格的数据类型和列式存储引擎，通过编码和压缩技术，降低读写放大；使用向量化执行方式，充分挖掘多核 CPU 的并行计算能力，读写查询性能显著。



e.StarRocks 表的设计体现：



1）采用 StarRocks 聚合表模型建表、同时支持明细表和物化视图。

2）按照寄件日期创建分区，按照运单号创建分桶，提升数据维护管理和检索效率。

3）通过 replace_if_not_null 支持部分字段更新，提升更新效率。

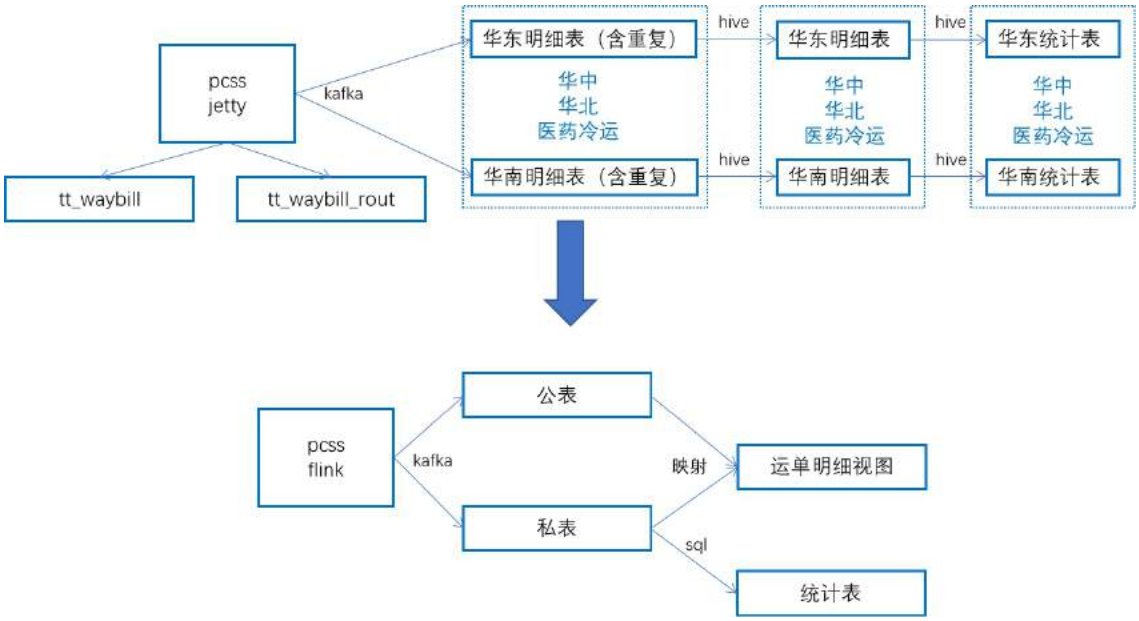
4）变化不频繁字段加到 key 列，并两个表冗余，提高查询效率。两表按照 Collocate Join 提升 Join 效率。

5）查询条件增加布隆过滤器索引，提升检索效率。

方案论证及其落地过程

创新点

创新点一：整个系统数据处理流程逻辑去繁化简；统一了客户端应用和丰景台数据源；既避免专项客服人员在各区部库之间来回切换，又防止了不同数据源数据不同步。

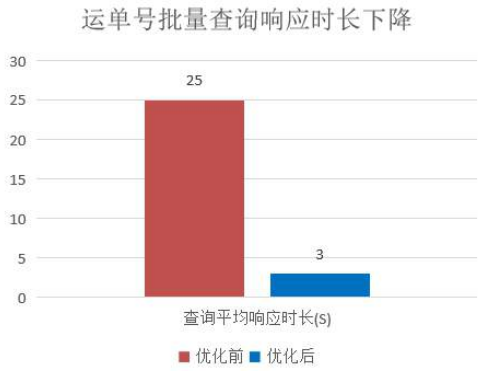


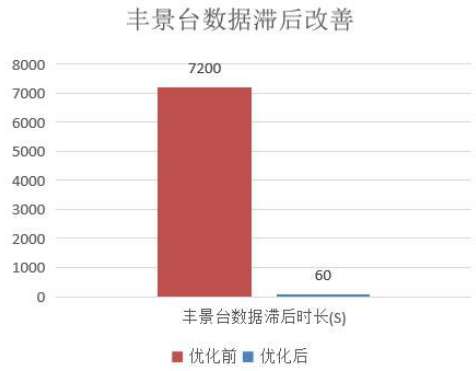
创新点二：顺丰科技 Flink + StarRocks 首个应用落地实践，支持起全日 15 亿+，8.2W TPS 大规模数据运算写入。

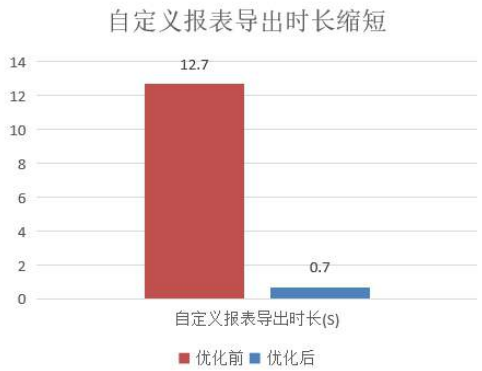
创新点三：借助 Flink 的 State(Rocksdb)本地缓存和 Redis 缓存技术相结合，避免对数据库查询操作，大幅提升应用吞吐能力和数据实时性。

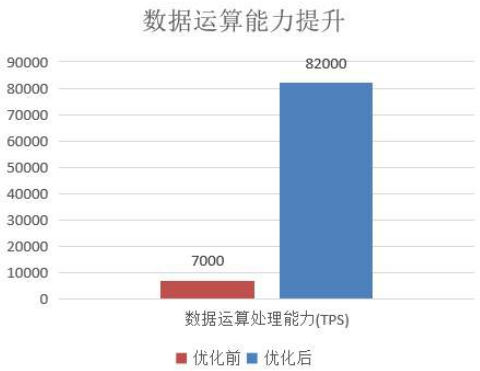
业务影响

影响一：用整体性能得到明细提升：以下应用各项指标均得到显著改善（数据来源生产压测结果）。









影响二：该创新优化方案实施上线后，可直接满足 2 年内业务增量的的运算与读写，并且未来 3~5 年也只需进行简单的水平扩容即可满足业务增长。

影响三：推动 StarRocks 开源社区优化，配合推动开源社区提升 kafka 导数性能（PCSS 当前数据库配置，可提升至 20W TPS 写入），支持 kafka 下发字段局部更新；提升了顺丰科技在开源社区的影响力！

技术前沿：去哪儿网业务大规模容器化最佳实践

作者：邹晟、陈靖贤
原文链接：https://mp.weixin.qq.com/s/P4Ro-OoPAvXjkurrUSRHnw
标签：云原生 Kubernetes

邹晟

去哪儿网基础架构团队高级 DevOps 工程师，现主要负责 CI/CD 平台开发与维护，云原生技术研究与实现。同时也是 KubeSphere Talented Speaker。

陈靖贤

去哪儿网 DevOps 产品经理，目前主要负责在去哪儿传播 DevOps 文化，调查、导入和开发流程、工具、平台的最佳实践，帮助公司以更快的速度交付软件，降低风险，降低运营成本。

近几年随着云原生技术的成熟，Qunar 为了实现整个技术体系的演进，Qunar 在 2021 年向云原生迈出了第一步 -- 容器化 。落地过程包括了价值评估、基础设施建设，CI/CD 流程改造、中间件的适配、可观测性工具、应用自动化迁移等。迁移过程涉及到了 3000 个应用左右，涉及千人级研发团队，是一个难度非常大的系统工程。这篇文章会讲述迁移过程涉及到的 CI/CD 模型改造、自动化应用容器化改造等最佳实践。

背景

容器化落地前的业务痛点

在容器化落地之前，我们经常会听到来自不同角色的各种各样的抱怨与吐槽：

- 领导层的声音：服务器资源和维护成本太高了，我们必须降低成本！
- OPS 的声音：有一台宿主故障了，需要把上边的服务赶快恢复！
- QA 的声音：测试环境好好的，为啥上线失败了？
- 研发人员的声音：业务高峰来啦，资源不够用啦！赶快去扩容！哎！扩容太慢了。

造成这些问题的因素主要包含：资源利用率、服务无法自愈、测试环境与线上环境不一致，运行环境缺少弹性能力。

企业所面临的商业环境是复杂多变的，**当今环境的竞争异常激烈，而且不再是大鱼吃小鱼，而是快鱼吃慢鱼**。通用电气前 CEO Jack Welch 曾经说过：“如果外部变化的速度超过内部变化的速度，终结就不远了”。我们的企业如何能在这样的环境中存活发展呢？各个企业都在不断的探索和实践中前行。今年来，云原生技术日趋成熟，已经被广大企业广泛接受并采用。云原生技术可以帮助企业降低成本，加快业务迭代，对赋能企业的产业升级提供强有力的技术支撑。容器化作为云原生技术之一，成为 Qunar 拥抱云原生进行技术升级的重要一环。

容器化落地过程的挑战与应对

全司范围内进行容器化全面落地并不是一件容易的事，我们面临了重重困难，但是我们为了最终目标的达成，想尽一切办法去一一化解。

首先，涉及部门多：容器化迁移涉及 OPS、基础架构、数据组等基础设施部门以及 20+业务部门。这么多的部门达成对目标的一致认同，并保持行动的协调一致是非常不容容易的。好在我们这个项目得到了公司高层的认可，并将此作为 2021 年的企业级目标。在企业级目标的引领下，各个部门协同一致，通力配合保障了项目的成功。

其次， 改造范围大：由于历史原因，我们的服务多是有状态的。从中间件、发布系统到网络、存储、日志收集、监控告警以及业务服务本身等等各个环节对机器名、IP、本地存储等状态有着强依赖。而容器本身是无状态的，这就意味着我们需要从基础设施、发布、运维的工具、平台进行整体改造。针对这重重问题，我们进行了一一列举，逐个击破，最终满足容器化迁移的条件。

再次，业务迁移成本高：本次迁移涉及应用数量大概有 3000 多个，迁移过程需要对应用进行升级改造、测试回归及上线，这个过程将花费大量人力。如何降低业务的迁移成本呢？我们支持将大部分的适配工作在中间件层进行统一支持，然后支持在业务代码中进行中间件的自动升级，自动进行容器化迁移，通过持续交付流水线对迁移过程中的变更进行自动化的测试与验证，经过容器与虚机灰度混部观察等手段大大降低了业务人工迁移的成本。

最后，学习成本高：我们的研发团队有千人级的规模，对于新技术的引入与升级，研发同学需要花费额外的成本进行学习和使用。为了降低大家的学习成本，我们通过平台工具屏蔽差异性操作以及技术细节，通过可视化配置、引导式操作，优化持续交付流程等方式来降低业务的学习成本。

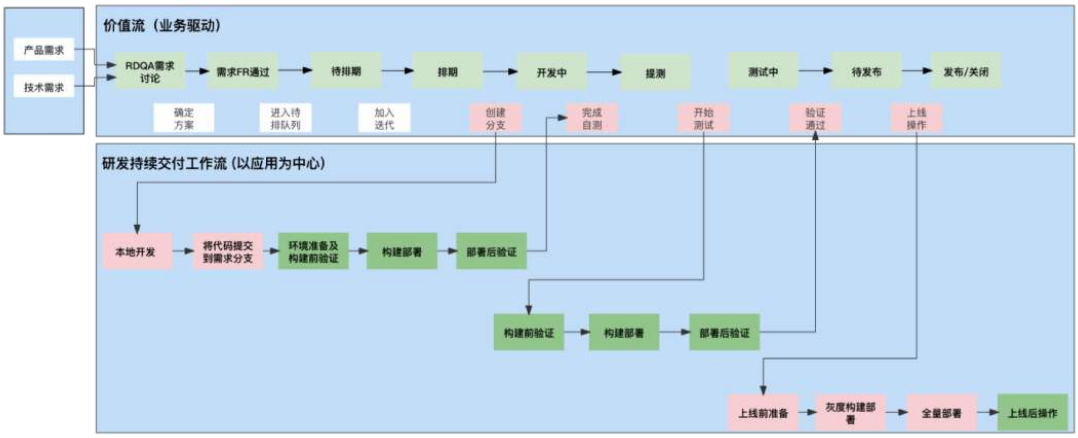
容器化后的收益

经过 2021 年一年时间，我们完成了容器化基础设施建设，工具平台的升级改造以及 90%应用的容器化迁移(应用总数 3000+)。从效果数据来看，容器化虚拟比例从 1：17 提升到 1：30， 资源利用率提升 76%；宿主运维时间之前以天为单位，容器化以后变成了分钟级，运维效率提升了 400 倍；由于容器化启动时间的缩短以及部署策略的优化，应用的交付速度提升 40%；K8s 集群提供了服务自愈能力，应用运行过程中平均自愈次数达到 2000 次/月；另外，容器化落地也为公司进行下一步云原生技术的深入推广与落地奠定了基础。

持续交付

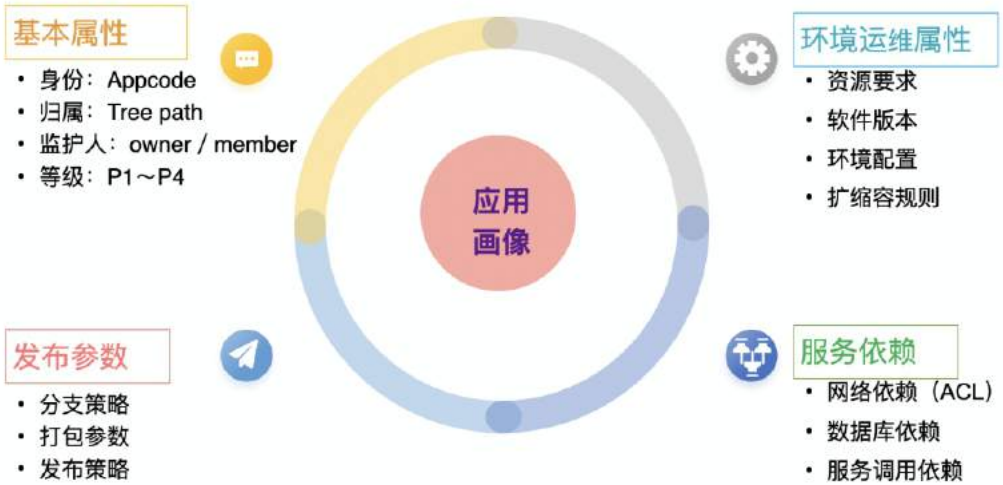
项目研发流程

Qunar 采用了业务驱动的价值流与以应用为中心的持续交付工作流的双流模型。企业以业务价值交付为目标，在交付过程中，各个阶段的交付物会在多个角色中的流转，在流转过程中难免会出现不同程度的浪费。为了有效对价值交付的效率进行有效度量与优化提升，我们不仅仅需要关注开发过程中的效率提升，还需要关注开发前的效率提升。我们将持续交付工作流的流转与价值流的流动进行联通，希望可以实现从项目域、开发域、测试域到运维域的流程自动流转的。但是在容器化之前，由于环境不一致、各阶段配置的不一致性等原因，导致交付过程中，交付流程在各阶段间流转时不可避免的存在人工干预的情况。容器化之后，我们参照云原生的 OAM 模型，对应用进行了规范化定义，建立应用画像，统一术语，消除数据孤岛，使流程可以顺畅高速流转。



应用画像

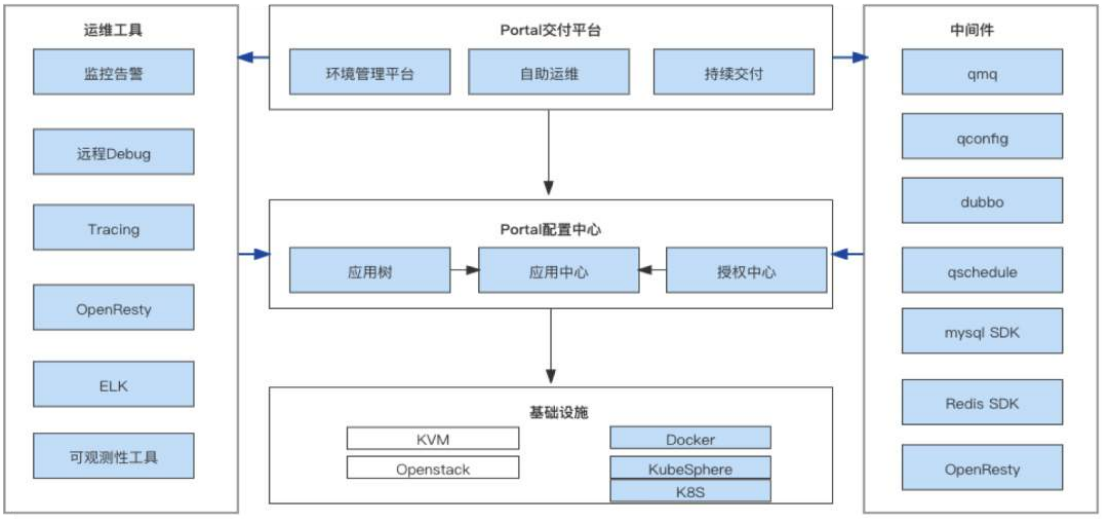
针对上面项目的流程流转，最重要的一个连接器就是 App code，因此我们也针对它做了抽象，画像定义：



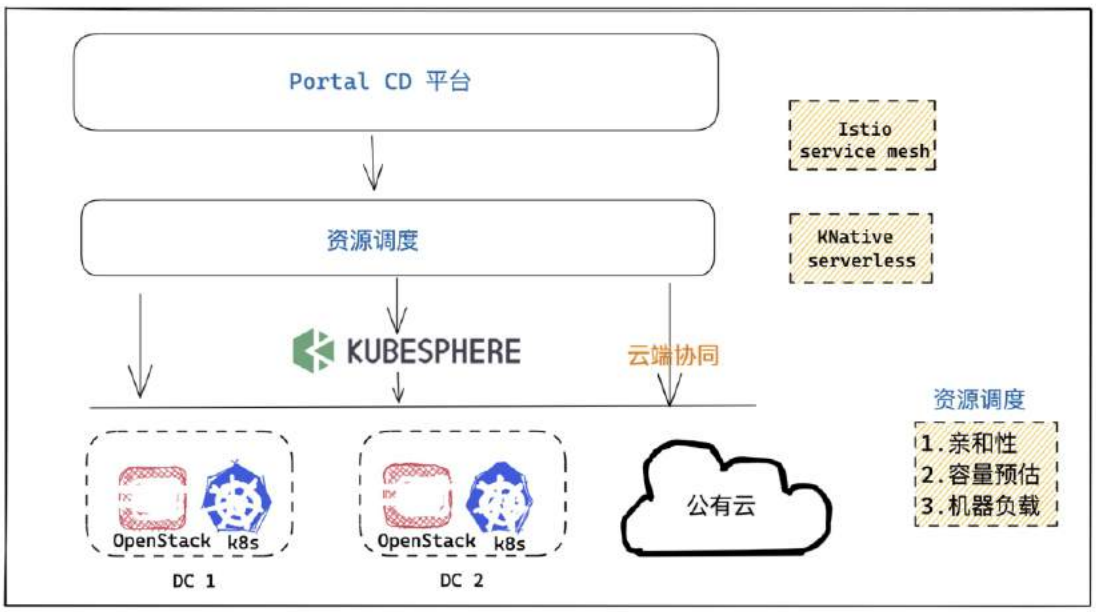
开发人员在面对开发、测试和生产等复杂的环境时、需要编写和维护多分应用部署配置文件；运维人员需要理解和对接不同的平台，管理差异巨大的运维能力和运维流程。

参照云原生中的开发应用模型原则，我们通过建立应用画像，指定应用的标准化定义。我们开发和运维人员通过标准的应用描述进行协作，轻松实现应用的“一键部署”、“模块化运维”，无须纠结于服务的开通配置和接入工作，提升应用交付与运维的效率和体验。

借助应用的规范化，将应用平台进行统一化与规范化。我们将原来的以资源为中心，转变为以应用为中心。平台架构如下：



多云协同



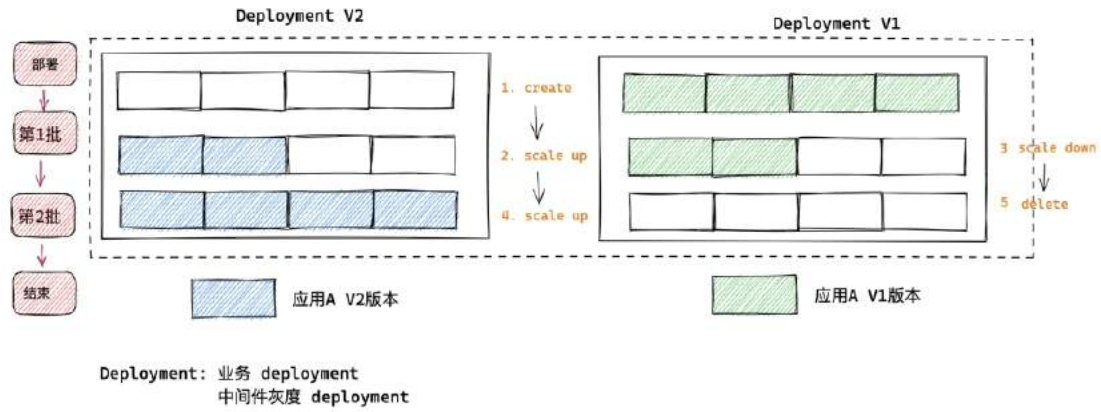
基础设施资源层：底层资源既支持 KVM 也支持容器，这些资源都是跨机房部署。同时为了让底层资源更具弹性，我们对接了公有云。

平台层：基于 K8s、KubeSphere 多集群管理、Service Mesh、Serverless 等云原生技术来提高技术先进性和技术的架构演进。

资源调度：

- 会考虑节点的亲和性：机器配置， 普通磁盘、SSD， 千兆网卡、万兆网卡。
- 容量预估：发布前预计算，如果资源不足，禁止发布。
- 机器负载：尽可能让集群所有节点负载均衡。

双 Deployment 发布



优点：

1. 降低了操作复杂度， 操作只包含 create, scale, delete。而单个 Deployment 更新过程可能会有更新过程失败，卡在中间状态， 升级和回滚都无法进行的时候。
2. 支持分批操作， 发布流程更可控。
3. 更新 Deployment label 变为可能。

缺点：

- 需要记录和控制 Deployment 状态，操作相对于单 Deployment 会相对复杂一些。

业务应用自动迁移方案

这次容器化需要迁移 3000+ 的应用，为了减少开发测试人员的迁移成本，我们提供了一个自动升级 Java SDK、自动迁移容器的方案。

自动迁移方案：

KVM 回收

为了快速腾出资源到 K8s 集群，我们会在规定期限内通知应用的 owner 回收机器，如果超过规定时间(7 天)，遗留的 KVM 资源回被强制回收。

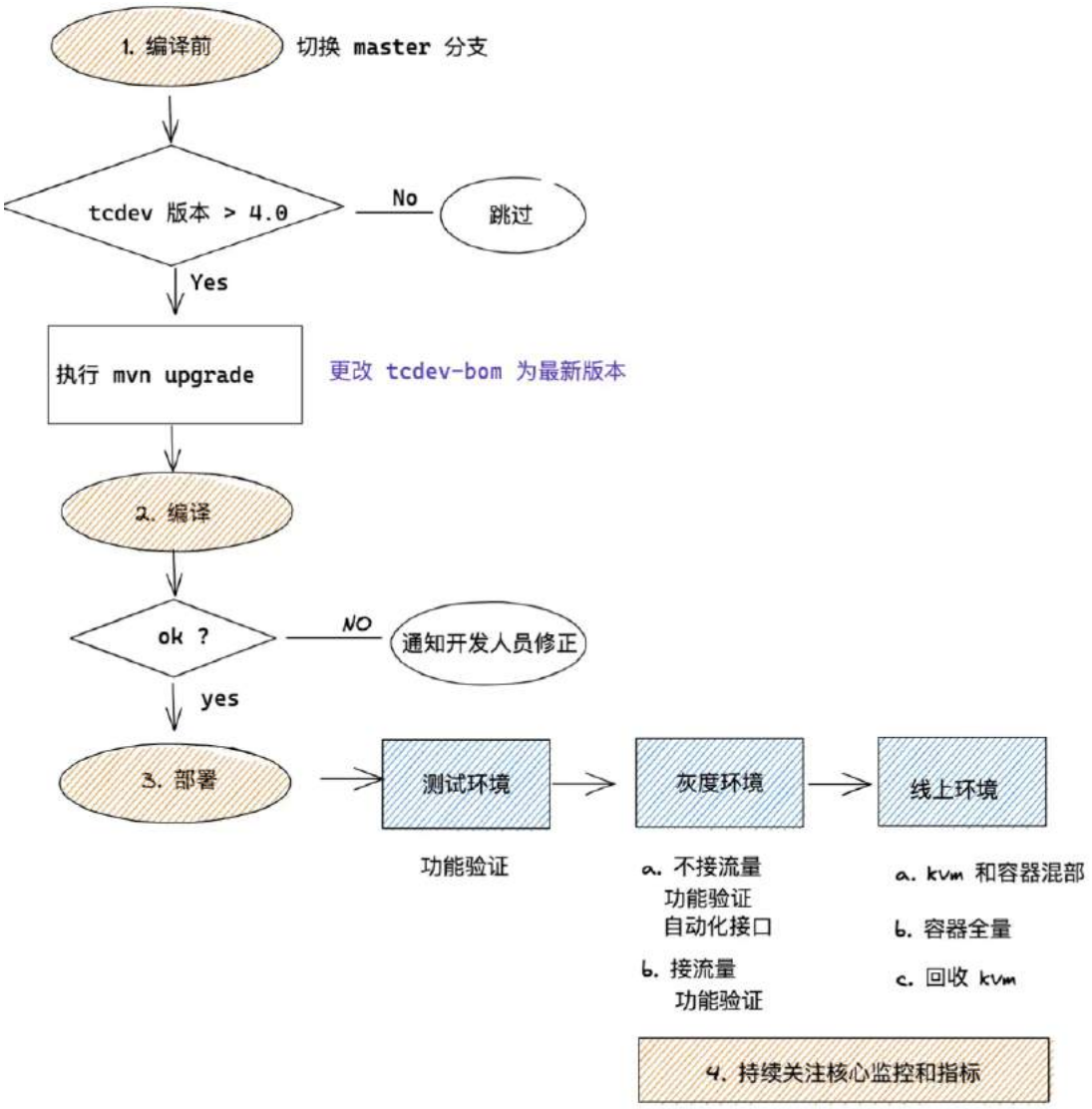
自动升级 SDK 流程

tcdev bom 是公司统一管理二方包和三方包依赖的公共基础组建，绝大多数的 Java 应用都会使用这个组建，因此我们的升级 SDK 方案就是在编译过程自动检测并升级 tcdev 版本。

升级 SDK 的前置条件

前提	说明
二方包、三方包统一管理	Super POM, tcdev-bom：核心组件统一管理、升级
质量门禁	静态代码检查，版本兼容性校验， 做好上线前的质量控制
自动验证组件升级后的兼容性	通过批量跑自动化测试，对比 master 分支和升级后的分支
发布的超管权限	自动升级、灰度、上线

升级步骤



事后复盘的时候，很多业务同学也都反馈这个自助升级迁移的方式为他们节省了大量时间，价值非常明显，得到了大家的认可。

总结

在云原生转型的过程中，如何让业务更顺畅的享受到云原生的红利是非常有挑战的，希望这篇文章能给刚步入云原生的同学带来一些启发。云原生的路上，我们一起共勉！

技术前沿: 无服务器系统的设计模式

作者： Tridib Bolar 译者： 张卫滨

原文链接：

https://www.infoq.com/articles/design-patterns-for-serverless-systems/
标签：前言科技 设计模式 架构

在软件架构和应用设计领域，设计模式是基本的构建块之一。设计模式的概念是由 Christopher Alexander 在上世纪 70 年代末提出来的（The Timeless Way of

Building, 1979 以及 A Pattern Language — Towns, Buildings, Construction, 1977）：

每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心。通过这种方式，我们可以无数次地使用那些已有的解决方案，而无需重复相同的工作。—— Alexander et al

随后，这个概念被软件社区所采用，从而产生了应用于软件设计领域的不同种类的设计模式。

面向对象的设计模式是一个抽象工具，用来设计遵循 OOP 方式的代码级别的构建块。Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides（Gangs of Four - GoF）合作撰写的图书 Design Patterns Elements of Reusable Object-Oriented Software（中译本名为《设计模式：可复用面向对象软件的基础》由机械工业出版社出版——译者注）为开发人员提供了面向对象设计领域的一个指导手册。这本书于 1994 年首次出版，从那时起，设计模式就成为了软件设计的一个组成部分。

设计模式也适用于组织。一个大型的组织确实就像一台庞大的机器，它有很多的齿轮、管道、过滤器、马达等等。在数字时代，我们正在试图将人脑数字化，因此将企业机器进行数字化并不是什么了不起的事情。将企业的某一组成部分或者某一区域实现数字化是不够的。实际上，要操控一个企业，就必须要集成其所有不同的组成部分。企业和解决方案架构师在尝试使用模式来解决日常的集成场景。这个过程是真正敏捷的。每天，来自世界各个角落的思想家们都在解决问题，并发明新的企业集成模式。在这里，我想要提及该领域的两位大师，Martin Fowler 和 Gregor Hohpe。

高层管理人员在不断追逐新的技术趋势，每天都有新的数字产品变种问世。商业人士都想方设法在这个数字海洋中获取最大的利益，所以有必要对遗留系统进行现代化改造，也就是所谓的数字化转型。在这个领域中，像 Ian Cartwright、Rob Horn、James Lewis 这样的研究人员也基于他们多年的迁移经验，在最近的 Patterns of Legacy Displacement 文章中提出了一些模式。

在这个快速变更的时代，敏捷性是成功的关键。弹性、持续交付、更快的上市时间、高效开发等等，这些都是推动系统向微服务架构转移的力量。但与此同时，并不是所有的场景都适合微服务。为了帮助我们理解这个边界在哪里，微服务模式的作者 Chris Richardson 为不同的使用场景提出了大量的微服务模式。

除了我上面提到的这些之外，还有更多的模式类别。事实上，关于企业系统架构和软件的模式有大量的文献。这意味着，架构师们需要明智地选择该如何满足他们的要求。

进入无服务器的领域

到目前为止，我们已经讨论了针对不同需求和架构的不同类型的模式，但是我们忽略了一个重要的场景，也就是无服务器的系统。在当前的技术范围内，无服务器是最重要和最有活力的方式之一，尤其是在 IaaS 和云计算领域。

无服务器平台可以分为两大类，分别是函数即服务（Function as a Service，FaaS）和后端即服务（Backend as a Service，BaaS）。FaaS 模式允许客户建立、部署、运行和管理他们的应用，而无需管理底层的基础设施。与之不同的是，BaaS 提供在线服务，通过云的方式处理特定的任务，比如认证、存储管理、通知、消息等等。

所有面向无服务器计算的服务都属于 FaaS 这一类别（比如 AWS Lambda、Google Cloud Function、Google Run、Apache OpenWhisk），而其他的无服务器服务则可以归为 BaaS，比如无服务器存储（AWS DynamoDB、AWS S3、Google Cloud Storage）、无服务器工作流（AWS Step Function）、无服务器消息（AWS SNS、AWS SQS、Google PubSub）等等。

无服务器这个术语非常具有吸引力，但是它可能会有一定的误导性。真的有服务能够无需服务器就能存在吗？在云供应商提供的所有无服务器组件的背后，隐藏着一个很简单的魔法：在这些组件幕后，都有一个服务器。云提供商负责管理物理机和 / 或虚拟服务器的可扩展性（自动扩展）、可调用性、并发、网络等，同时还会为终端用户提供一个接口来配置它们，包括像自定义运行时、环境变量、版本、安全库、并发、读 / 写容量等。

如果我们专注于使用无服务器方式实现一个架构的话，那么随之而来的是一些基本的、高层次的问题。

- 使用无服务器构建块设计一个系统时，首选的架构风格是什么？
- 我们的应用要采取纯粹的无服务器方式，还是采用混合方式？
- 我们该在哪些用例中采用无服务器方式呢？
- 在实现无服务器应用的时候，有哪些可重用的架构构建块或模式呢？

在本文剩余的内容中，我将会阐述上述四个问题的答案。

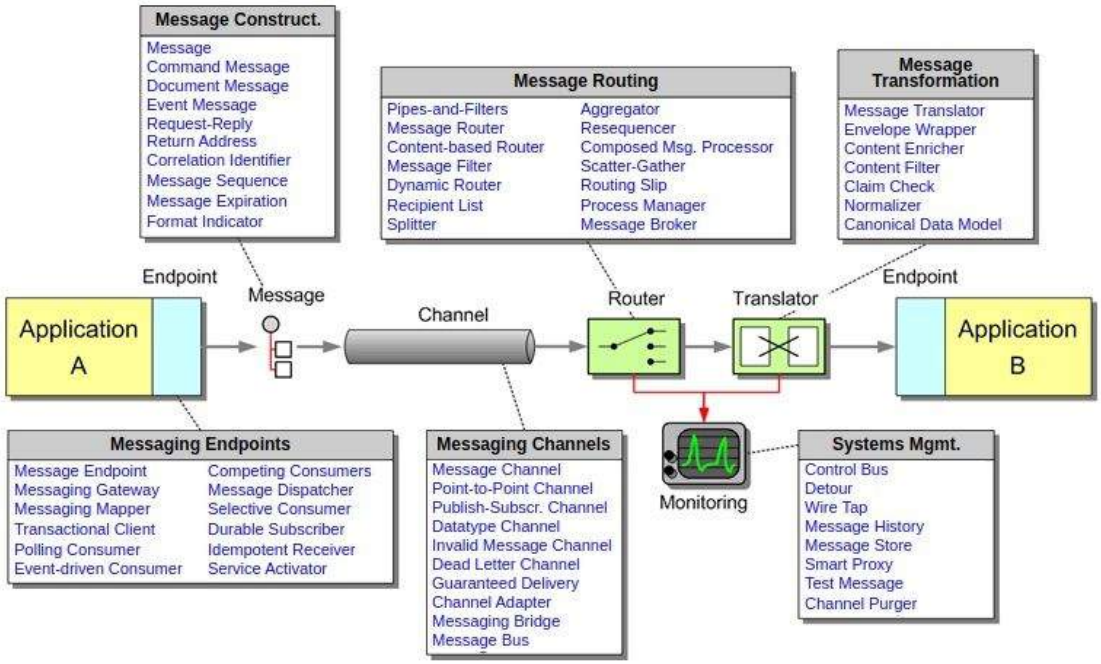
无服务器模式

在技术领域，无服务器模式相对比较新，而且正处于快速发展之中。它所涉及的不同方面，包括运行机制、适用性、使用场景、使用模式、实现模式等，每一步都在不断发生着变化。不仅如此，随着云供应商不断发明新的无服务器产品，同样的微服务模式可以通过各种方式来实现，它们的价格和性能也各不相同。在世界范围内，软件工程师都在从不同的视角出发，使用不同的方式在思考。因此，到目前为止，尚未形成构建无服务器系统的通用方式。

在 API Days 澳大利亚会议上，来自亚马逊云科技的解决方案架构师 Cassandra Bonner 做了一个关于 Lambda 无服务器服务的五个主要使用模式的演讲。她从需求的角度定义了这五个模式：

管道和过滤器（Pipes and filters）

这些模式并不是无服务器架构所特有的。实际上，它们是分布式系统模式的一个子集，比如由 Gregor Hohpe 和 Bobby Woolf 总结整理的 65 个消息模式，它们代表了这种模式最广泛的集合。



我撰写本文的目的是在 AWS 云环境中按照无服务器的方式实现管道（Pipe）和过滤器（Filter）模式。我将会讨论一些可供选择的实现方式以及它们各自的优势和劣势。在实现过程中，可重用性是我要考虑的一个具体的方面。

无服务器架构的管道和过滤器模式

在敏捷编程中，以及对微服务友好的环境中，设计和编码的方式已经与单体时代不同了。敏捷和微服务开发者不再把所有的逻辑放到一个功能单元中，而是倾向于更加细粒度的服务和任务，遵循单一职责原则（single responsibility principle，SRP）。有了这一点，开发人员就可以将复杂的功能分解成一系列可独立管理的任务。每个任务会从客户端获取一些输入，然后消费这些输入以执行其特定的职责，并生成一些输出，这些输出会转移到下一个任务中。根据这一原则，多个任务构成了一个任务链。每个任务都将输入数据转换成所需的输出，而这些输出又会作为下一个任务的输入。这些转换器（transformer）传统上被称为过滤器，而将数据从一个过滤器传递到另一个过滤器的连接器（connector）被称为管道。

管道和过滤器一个非常常见的用法是这样的：当客户端的请求到达服务器的时候，请求载荷必须要经历一个过滤和认证的过程。当请求被处理的时候，可能会有新的流量进来，在执行业务逻辑之前，系统必须要执行一些通用的任务，比如解密、认证、校验并从请求载荷中移除重复的消息或事件。
另外一个场景就是在电子商务应用中将商品添加到购物车的过程。在这种情况下，任务链可能会包含如下的任务：检查商品的可用性、计算价格、添加折扣、更新购物车总数等。对于其中的每个步骤，我们都可以编写一个过滤器，然后使用管道将它们全部连接起来。
实现这种模式最简单的方式就是使用 lambda 函数。我们知道，有两种调用 AWS 服务的方式，也就是同步方式或异步方式。在同步场景中，lambda 运行函数并等待，直到发起调用的 lambda 接收到被调用 lambda 的响应为止，而在异步的情况中，不需要等待。AWS 支持回调方法和 future 对象来异步接收响应。在这里，管道的角色就由内部网络来扮演。

在这种直接的 lambda 到 lambda 的调用中，不管是同步还是异步，都有可能出现节流的情况。当请求的流入速度超过了函数的扩展能力，并且函数已经到了最大的并发水平（默认是 1000），或者 lambda 的实例数量达到了配置的预留并发限制，所有额外的请求都会因为节流错误（状态码为 429）而失败。为了处理这种情况，我们需要在两个 lambda 之间添加一些中间存储，这样能够临时存储无法立即处理的请求并实现针对被节流消息的重试机制，一旦有 lambda 实例可用，它就会获取这些消息并开始对其进行处理。
我们可以通过使用 AWS 的简单队列服务（Simple Queue Service，SQS）来实现这一点，如下图所示。每个 lambda 过滤器处理一个事件并将其推送到队列中。在这种设计中，Lambda 可以从 SQS 轮询多个事件，并作为一个批次进行处理，这也可以提高性能和降低成本。

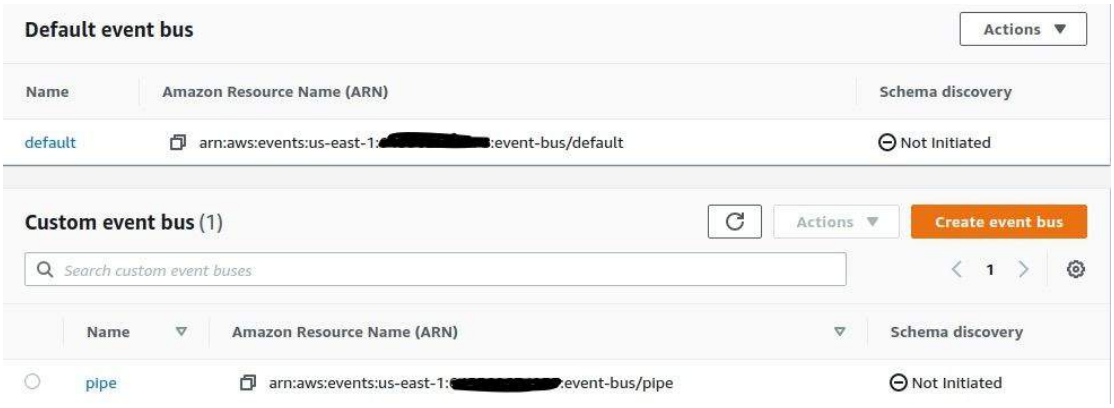
这种方式可以减少节流的风险，但是并不能完全避免。这里有一些可配置的参数，我们可以使用它们来平衡节流。除此之外，我们还可以为 lambda 实现一个死信队列（Dead Letter Queue，DLQ）来处理被节流的事件 / 消息，并能够防止这些消息丢失。有一篇很好的文章题为“在数据项目中组合使用 SQS 和 Lambda 的经验教训”，读者可以通过它来了解解决该问题的关键参数。
在下一节中，我将会构建一个通用的、可重用的解决方案，该方案会用到另外一个适用于无服务器事件处理的 AWS 组件，即 Amazon EventBridge，我会实现管道和过滤器设计模式。

在无服务器架构中实现管道和过滤器模式

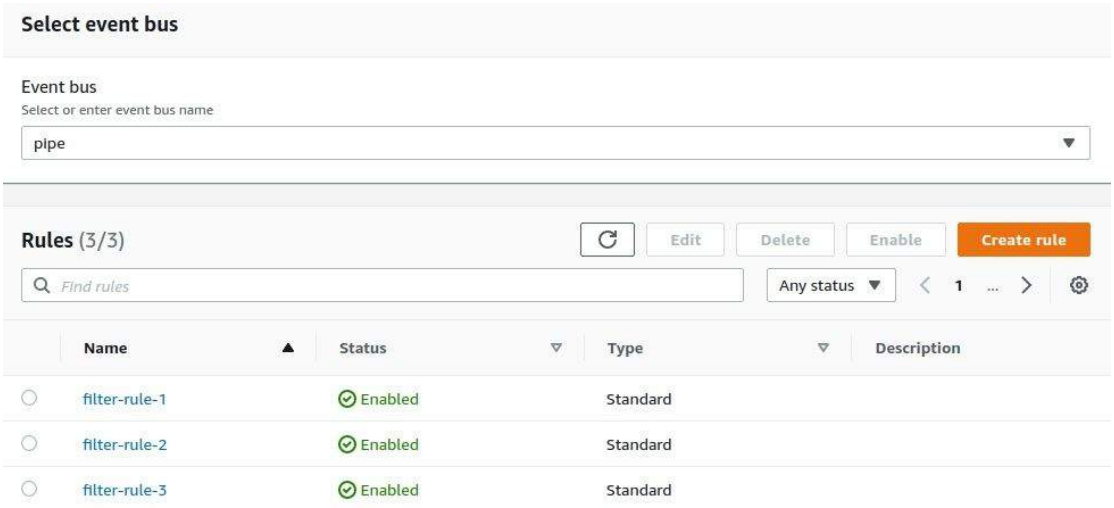
Amazon EventBridge 是一个无服务器事件总线，它可以利用从你的应用程序、集成的软件即服务（SaaS）应用程序和 AWS 服务中产生的事件，从而能够更容易地构建大规模的事件驱动应用。

在了解它如何运行之前，我们需要理解一些与 AWS EventBridge 相关的术语。

事件总线是 EventBridge 的关键组件之一。事件总线接收来自不同源的事件 / 消息，并将它们与一组定义的规则相匹配。EventBridge 有一个默认的事件总线，但用户也可以创建自己的事件总线。在这个 POC 中，我创建了一个名为“pipe”的事件总线。



规则（Rule）必须要与特定事件总线关联。在这个 POC 中，我为三个不同的过滤器创建了三个规则，如下图所示。



对于每个规则来讲，事件模式和目标是两个非常基本的配置。

事件模式是一个条件。它与自己所匹配的事件具有相同的结构。如果传入的事件具有相匹配的模式，那么规则就会被激活，并将传入的事件传递给目标（目的地）。目标是一个资源或端点，EventBridge 能够将事件发送给它。对于特定的模式，我们可以设置多个目标。

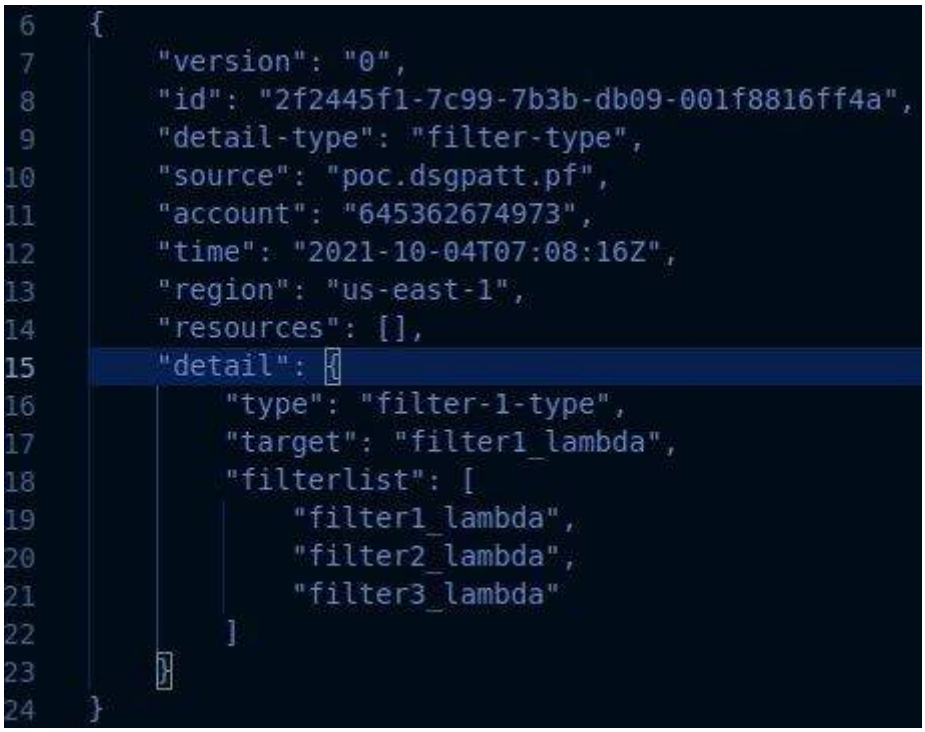
在我们的例子中，我将 lambda 名设置为模式中的 detail.target，一旦 lambda 名称匹配，目标 lambda 就会被触发。

注意：detail.target 是一个 json 字段。目标是事件的一个可配置的端点 / 目的地。

在事件流中，可以执行的不同步骤如下所示：

1. 源生成一个事件（它必须遵循事件源生成器和 event bridge 规则创建者所定义的模式）。

   为了测试我们的实现，我使用了如下的事件：



2. 基于测试事件的具体 detail.target 值，会有一个规则匹配并执行。在我们的场景中，这将会导致事件 / 消息会路由到与规则关联的目标 lambda 上，即 filter1_lambda。
3. 目标 lambda 完成其任务，并将事件目标（detail.target）替换为 detail.filterlist json 列表中的下一个 lambda，也就是 filter2_lambda。
4. 目标 lambda 随后调用 lambda 层的工具函数 next_filter()。
5. next_filter() 函数负责构建最终的事件并将其放到 event bridge 中。
6. 基于新的目标值（即 filter2_lambda），另外一条规则能够被匹配，从而会调用一个单独的过滤器 lambda。

```
{
  "Detail": {
    "target": ["filter2_lambda"]
  }
}
```

7. 在完成所有的任务之后，终端过滤器会将消息发送给下一个非过滤器的目的地。在本 POC 中，终端过滤器是 filter3_lambda。这个 lambda 不再调用 next_filter 函数，而是调用 DynamoDb API，将数据保存到 DynamoDb 的表中。

正如我们所看到的，借助 EventBridge 的模式匹配路由功能，我们可以用单一的事件总线来实现管道和过滤器模式，即便链中的某个后继阶段依然在忙于处理前一个事件，链中的其他阶段都可以自由地开始处理下一个事件，从而提高整体效率。

如上图所示，事件最初会到 filter1_lambda 中，因为客户端事件的 detail.target 属性与目标为 filter1_lambda 的 filter-rule1 事件模式相匹配。执行完成后，filter1_lambda 将事件的 detail.target 设置为下一个 lambda，即 filter2_lambda，并将修改后的事件发回给事件总线。由于 detail.target 的值是 filter2_lambda，所以 filter-rule2 就会被触发，如此反复。通过这个递归过程，所有的过滤器都会被执行。最后一个过滤器可以调用一些其他资源，而非调用 next_filter() 工具层。在上面的实现中，每个 lambda 共同的重要任务之一就是将事件目标（detail.target）修改成 filterlist 中的下一个 lambda。为了完成这个任务，我们使用了 lambda 层（lambda layer）。

lambda 层是 lambda 的一个特性，它可以帮助开发者从 lambda 代码中提取通用功能或库，并将其放入一个层中。这个层可以作为一个工具式的代码块，实际的 lambda 代码可以在这个层上面执行。Lambda 可以根据需要重用该层的通用功能和 / 或库。

AWS 文档这样说：

Lambda 层是一个包含额外代码的归档文件，如库、依赖，甚至是自定义运行时。

对于这个 POC 来讲，我写了一个工具层，它导出了 next_filter 函数。Lambda 过滤器使用这个函数从 filterlist 中推断出下一个过滤器的名字。相关的代码片段在本文末尾的附录中给出。

整个 POC 代码以及 AWS 云开发工具包（AWS Cloud Development Kit，CDK）的基础设施代码可以在 github 仓库中找到。

总结

模式是软件设计领域中最有用、最有效的工具之一。为了以标准的方式解决常见的设计问题，我们可以使用合适的设计模式。模式就像一个设计插件。在技术方面，无服务器是一个快速增长的领域，所有的云计算供应商都在定期推出新托管的无服务器服务。因此，要决定一个合适的无服务器管理服务的技术栈是很困难的。在这篇文章中，我讨论了如何使用不同的 AWS 无服务器托管服务，以无服务器的方式完成一种设计模式的不同实现方法。

附录

next_filter 的代码片段：

```
module.exports.next_filter = (async function (event) {
var i = event.detail.filterlist.indexOf(event.detail.target);
if (event.detail.filterlist.length === i + 1) {
return null;
} else {
event.detail.target = event.detail.filterlist[i + 1];
var finalEvent = {
"Source": event.source,
"EventBusName": "mypipe",
"DetailType": event["detail-type"],
"Time": new Date(),
"Detail": JSON.stringify(event.detail, null, 2)
}
var Entries = [];
Entries.push(finalEvent);
var entry = { "Entries": Entries };
var result = await eventbridge.putEvents(entry).promise();
return result;   }
```

```
});
```

参考资料

Lambda SQS 扩展

（https://aws.amazon.com/cn/premiumsupport/knowledge-center/lambda-sqs-scaling/）

SQS 消息的短轮询和长轮询

（https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-short-and-long-polling.html#sqs-long-polling）

节流

（https://docs.aws.amazon.com/lambda/latest/operatorguide/throttling.html）

在数据项目中组合使用 SQS 和 Lambda 的经验教训

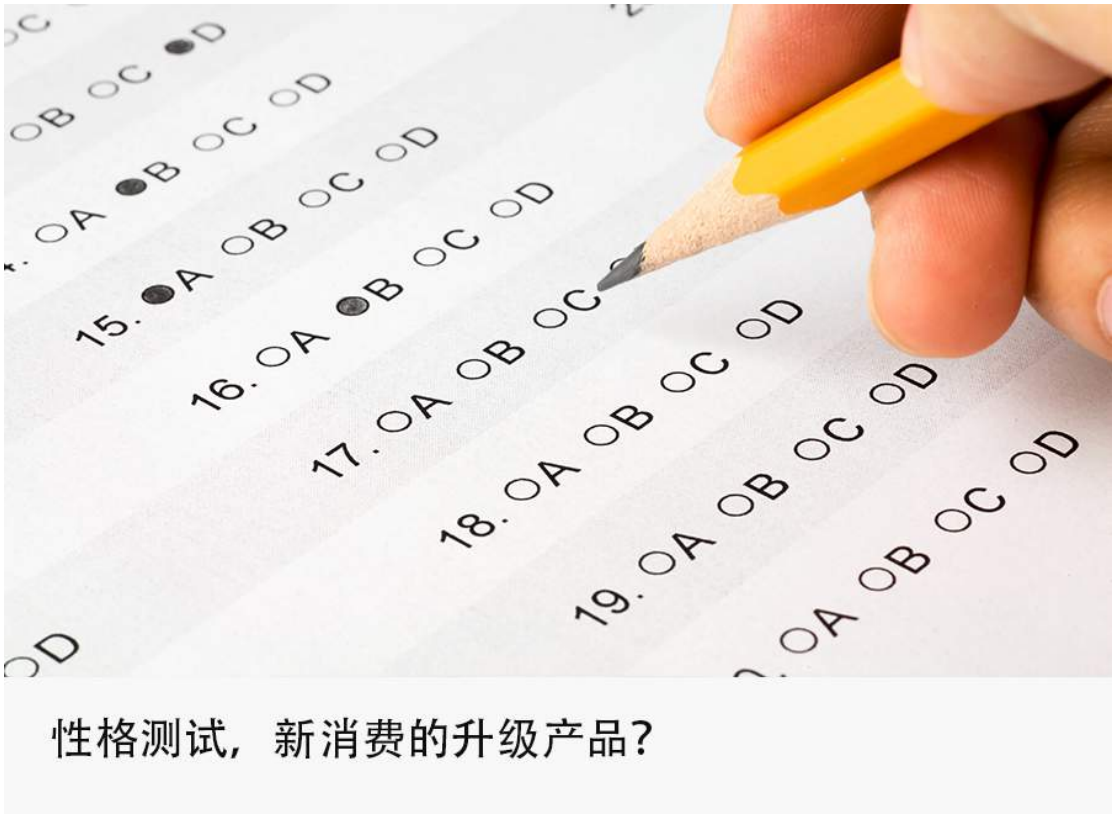
（https://data.solita.fi/lessons-learned-from-combining-sqs-and-lambda-in-a-data-project/）

作者简介：

Tridib Bolar 在印度加尔各答工作，是一家 IT 公司的云计算解决方案架构师。他已经在编程领域工作了 18 年以上。他主要从事 AWS 平台相关的工作，同时也在探索 GCP。除了是云计算无服务器模式的支持者之外，他也是物联网技术的爱好者。

技术前沿：爆火的性格测试，催生年入千万的低成本生意

作者：乔雪
原文：https://mp.weixin.qq.com/s/kNeoVmzDY6Yd27sY7Fpwug
标签：前沿科技 新商业



有句话常说，性格决定命运，在短视频当道和知识付费盛行的当下，互联网甚至让性格测试成为了一门爆火的生意。

在抖音、快手、知乎、小红书等平台上，有关于“性格测试”相关的内容，在热榜上几乎从未缺位，它所映射的，更是当代年轻人生存的矛盾体：他们一方面想要躺平，

在口头上拒绝“职业规划”，另一面，却热衷探索自己的生活和职业的轨迹，沉迷用测试指导生活。

测试性格的背后，是打工人们对自己所处困境的疑惑，但在求解的过程中，往往会在半路遭遇到不同的生意们的“援手”。

01

性格测试，新晋流量密码和社交货币

我是天秤座，你是什么星座？星座曾是社交中最频繁的谈资，而如今，风向变了，今天的年轻人们拥有了新的社交黑话，社交简介上的“INTP，ENTJ”，或者是“我是 INTJ，想要结识 INFP 小可爱，对 Fi、Ne 展现出的天马行空非常上头，感觉 INFP 个个都是小天才和诗人。”

以 MBTI 为首的性格测试突然成为了新的流量密码，在短视频、B 站、小红书等年轻人聚集的平台上，MBTI、九型人格、DESC 等性格测试都有不错的流量，豆瓣“人格气质心理”小组已经有 26 万人聚集于此，每天热衷于讨论性格类型；而甚至很多卖货博主也在想尽办法蹭上性格测试为自己带流量。

小红书博主旅行作家“悄悄”发的第一篇笔记就是关于 MBTI 的，短时间内就获得了 3000 多的点赞，之后的很多视频都有过万点赞，十多万的阅读，这些数据让她感觉到这个领域火了，之后“悄悄”也顺其自然地成为了一位研究 MBTI 的专业博主，很多心理、助眠、图书的商家都找上门来想要合作。

在一款 95 后喜爱的社交 APP 上，用户需要选择自己的 MBTI 类型，如果你还不清楚自己到底属于哪一类，它还会引导你去测试，然后为你推荐适合的性格用户进行交友匹配，MBTI 自带“现代科学”的属性，它变成了超越生辰属相、紫微塔罗和星座更潮流的社交货币。

一位最近迷上了 MBTI 的大学生陈芝告诉 Tech 星球，测试结果让她非常信服，她也在用这个方法分析身边的人，“前男友和我一样都是 infp，所以相处起来会很累，而现在的男朋友是 enfj，很配，感觉是和我最搭的性格。”

MBTI 到底是什么？MBTI 全名为迈尔斯 - 布里格斯类型指标（Myers-Briggs Type Indicator），是基于卡尔·荣格（Carl Jung）的心理学理论发展出来的人格测验，荣格将人体现世界四种心理功能划分为两种维度——实感 (S)/ 直觉 (N)、思考 (T)/ 情感 (F)，每种功能又有两种体现——内向 (I)/ 外向 (E)，共计八种心理类型。MBTI 的创始人又在此基础上增加了一个维度：判断 (J)/ 感知 (P)，将人格划分成了如 INFP、INFJ、ESFP、ENTJ 等 16 种。

因为 MBTI 性格分析的火爆，2021 年底，韩国为此推出一档社交观察类综艺《MBTI Inside》，节目将 16 种人格类型的人聚集在一起，观察其碰撞出的化学反应，节目播出后收视异常火爆，或者说因为蹭上了 MBTI 的热点才让综艺变得热门。Disney+

首部原创韩剧，在塑造人物上，直接加入 MBTI 元素，为不同人物设计专属 16 型性格，再次推动 MBTI 的火热。

甚至连虚拟的小说人物都没有逃过被测试的热潮：《红楼梦》里贾宝玉个性鲜明，思维发散，爱讲故事，是 ENFP 型；林黛玉浪漫多情，善于自我表达，是 INFP 型……而且，MBTI 不仅仅是娱乐的专属，这个测试全世界每年有超过 200 万人参加，美国有大约 200 个联邦机构给员工测过 MBTI。美国亚利桑那州立大学甚至求职宣传手册上写道：“MBTI 人格类型测试能帮助你找到更好的工作”。性格测试的火热，不仅仅推动着人们在社交领域的分享与标榜欲望，它还孕育出商业的种子。

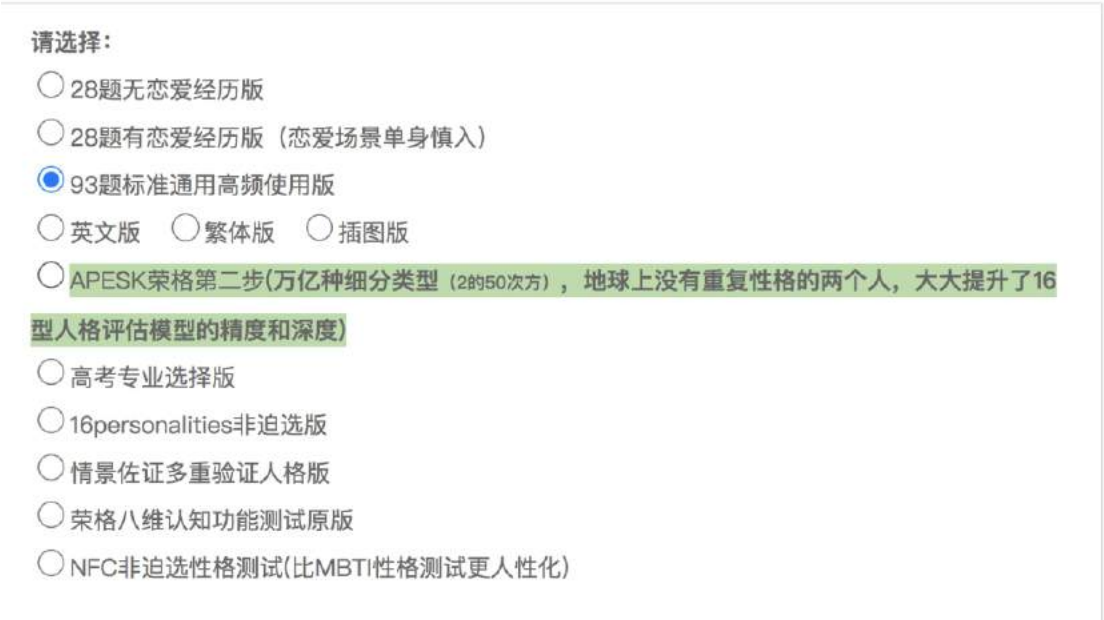
02

性格测试商业化：年入千万的低成本生意

性格测试，不再局限在简单的测试范畴，它已发展为一门生意。

牛津大学文学系副教授米尔维·艾姆蕾（Merve Emre）在 2018 年提到，《财富》榜 100 强公司有 89 家在不同阶段对员工进行性格评估，性格测试产业已经形成约 20 亿美元的市场。拥有 MBTI 正版版权的 CCP 公司，每年能创造千万美金的收益。

而这套商业模式，很快就在国内生根发芽。在测试类型上，市面上各类名目的测试琳琅满目，呈现出多样和复杂性，以 MBTI 为例，就有 28 题、93 题等更多的版本以供选择，为了方便信息检索的便利，有很多商家利用信息差，兜售各种类型、各种版本的性格测试，以满足测试这一需求。



性格测试正成为一种商品，甚至某种程度上也在变成快消品。淘宝上定价在 5 元到 100 元不等，但测试本身不是目的，这背后的结果以及分析才是更一步的需求。

肖涵告诉 Tech 星球，自己在某网站测试出结果后，为了获得进一步的解读，又购买了 58 元的深度解析版本，为了对自己的心理状况和背后原因进一步剖析，又在抖音

上找到一家心理工作室获得更深度的咨询。肖涵介绍说，该工作室依据心理咨询师的等级不同定价，每小时的定价在 300-1200 元之间，“分析得很准的，但具体有什么帮助，其实很虚的东西。”

而在以量取胜，一锤子买卖的贩卖性格测试后，另一种生意模式客单价更高，粘性更强。

以性格测试为噱头和卖点的培训也已经形成一张成熟的网络，Tech 星球发现，在某一 DISC 的测试页面上，想要进行免费测试，需要首先关注公众号，测试完后只出现一个简单的结果，但如果想针对自己性格获得更专业的解析，则可以线下免费一对一的导师分析，因此也形成了完整的链路，巧妙地指向线下培训机构。

在线下的该培训机构里，又会先对学员进行一个更简单的色彩心理学测试（红、黄、蓝、绿四种性格），值得一提的是，该培训老师并没有心理学相关背景，在此之前是 k12 学科培训老师转型而来。

培训老师会针对测试者性格短板，刻意向职场和人际关系方向引导，“你知道董明珠吗，你觉得她是什么类型的人？她和你一样是大红型人格，我有一个学员在她公司做直播，董总是对事不对人的类型，如果你做错了她会狠狠地批评你，但是上播前，她也会细心问候每个工作人员，所以你这种大红的人一定要学会蓝色的一面。”

除了向名人靠拢，该老师的话术还激发出打工人们普遍渴望升职加薪的潜在需求，“我接触过一个字节的学员，她之前自我表达和心理素质都特别差，但是通过学习，现在已经升级为管理层了，手下带十多人的团队。”老师继续发动心理攻势，“难道你想一辈子做打工人吗，不可能一直做基层吧。”

根据学员编号，Tech 星球推测出，该老师一年间就有 200 多名左右的学员，以《DISC 性格沟通科学》这门课为例，售价为 5688 元，但课程常常捆绑销售，需要配合其他几个课程一起授课，据该老师介绍的平均过万客单价来算，该机构 4-5 名老师，一年可以达到千万营收。

性格测试的另一巧妙之处在于，它把测试结果和职业与就业强关联。一种类型往往会指向很多种职业分析，因此更多的公司将其用于面试、培训等各个方面，比如业内盛传的“华为每 20 个面试者就有 17 个败在性格测试”、“PwC 喜欢目光长远、有计划且喜欢创新的人”、“德勤喜欢诚信且以团队为核心的人”等。

为了提升通过率，也有招聘产业在盯上这门生意，Tech 星球咨询相关猎头时，该猎头表示“包过，有内部性格测试题库，有专业的老师辅导。”

03

性格测试的背后：伪科学还是真需求？

用一套测量的结果，去评判一个人的性格，靠谱吗？

事实是，MBTI 测试并没有现代心理科学的学术基础，它甚至连一套最为基础的心理学都算不上，作为 MBTI 的开创者，也就是创造出我们一说到性格就“内向 / 外向”的二维划分的心理学大师荣格，他也曾明确地警示过“性格类型”只是他观察到的粗略倾向，而不是严格的分类。

一个基础的 MBTI 测试结果大概是这样的：

“INTP 人格类型相当稀少，仅占人口的百分之三，因为’寻常普通’最使他们不快。INTP 以自己的创造力和发明的才能，独特的视角和充沛的智慧为傲。INTP 常成为哲学家，建筑师，和充满想象的教授，历史上的很多科学发现都要归功于他们。”

甚至在测试的底部还为该类型列举出了代表人物：牛顿、爱因斯坦，以及比尔盖茨都属于 INTP。

国家二级心理咨询师凌宇告诉 Tech 星球，这里的结论牵涉到 2 个心理学效应，一方面，名人效应让测试者更愿意相信结果的可信度，也放大了该测试的结果；另一方面，是巴纳姆效应在发挥作用，这指的是“当人们用一些普通、含糊不清、广泛的形容词来描述一个人的时候，人们往往很容易就接受这些描述，并认为描述中所说的就是自己。”

而现代心理学早就淘汰了这一套分类方法，即便是拥有 MBTI 正版版权的 CCP 公司，其董事会中有三位顶尖的心理学家，他们自己也从来没有在学术研究中使用过 MBTI。

凌宇还告诉 Tech 星球，专业的心理学维度的测试有大量的数据和实验作为测试基础，且在不断的优化和迭代，而且一旦有心理学专业人士做测试后，都需要有细化解析和针对测试人关键事件的分析，每个人的结果都是千人千面的，而这种仅仅把人格划分为十几种，并没有科学性。

此外，重测率也是心理测验的一大指标，据统计，MBTI 在 1200 名测试者中，重测率为 57%，仅有 17% 的人承认它的解释力。而合格的心理学测试，至少要达到 50%。

凌宇所在机构的所有咨询师都不提供相关服务，他还认为，这类测试可以当作消遣，但别太认真，如果真的有精神上的困扰，一定要寻求专业人士的帮助而不是网上的测试。

即便是这样，依然无法阻挡人们愿意在各大社交网站上分享，MBTI 拥有积极心理学的影子，很少对人下负面的结论，因此也增加了分享欲，很多测试者会主动寻找测试结果中与自己性格相符的部分，进而对测试模型给于高度评价。

研究两年多 MBTI 的小红书博主旅行作家“悄悄”告诉 Tech 星球，MBTI 其实是一种工具，它作为一套理论不用全部相信，但作为一面“镜子”是有价值的，能教会我们如何重视和了解自己，而每一种性格都可以从低阶进阶到高阶，完善短板，发挥天赋，避免资源错位。而在此之前，国内这方面教育的确存在缺失，MBTI 弥补了这个深层次需求。

心理学可以看作是消费升级的一种，随着物质生活的逐步满足，精神消费的需求开始浮出水面，并不断增多。据相关数据，国内精神心理科的执业医师仅 4.5 万名左右，而精神问题占到了人口的 10% 以上。这也从侧面反映出，心理消费高度不足的供给端和需求端的刚需，也为性格测试的火爆提供了滋养土壤。

这还在深层次地反应出，国内的心理健康赛道还有待发展，但就像荣格所说的：人格类型不是静止的，是随着时间不断‘旋转’的”。要知道，每个人的性格都是独一无二的，千人千面，而性格测试却只有 16 面。

欢迎投稿

客渠技术月刊编辑部

扫描二维码立即加入群聊