# VMS2.0介绍

## 背景介绍

VMS经历了0.x、1.x版本的演进；从0.x支持简单的消息发送/接收功能和消息资源申请/管理服务；到1.x版本支持完善的消息服务功能，以及各类在线运维、服务治理和监控告警等平台级别服务，性能和可靠性方面也有大幅度的提升；目前1.x版本已经大规模投入到生产环境，为各类业务域提供了稳定可靠的消息服务；

目前线上核心系统都迁移到1.x版本的客户端，该版本支持两套消息引擎(kafka和rabbitmq)以适用不同的业务场景，当时kafka\_v0.8版本在可靠性方面还存在一些瑕疵，很难应对核心业务的需求，对于重要业务场景还是采用rabbitmq来支持；对于流量比较大,落盘性能要求高的场景使用kafka引擎来支撑。但在实际生产中管理/维护两套引擎难度比较大，一方面各套引擎对技术人员要求都比较高，要完全掌握各技术细节需要花费较大的时间和人力成本；另外随着最近kafka版本不断更新，在可靠性和可用性方面都做了很多改进，我们从各个角度对比了两套消息系统的特性（[Angelia消息引擎选型调研](http://wiki.corp.vipshop.com/pages/viewpage.action?pageId=385942541)），从各个维度对比来看，选定一套消息引擎kafka能够满足目前的各类业务对消息的需求，可靠性方面也能极大满足。

最新版本原生kafka在功能上还未能支持较全面的消息系统特性，但提供了消息引擎core，在其基础上可以方便的扩展出更丰富的消息特性以更好的支撑各类业务场景，并提供非常高的吞吐性能，对日益增长的业务流量会带来特别高的收益。

下面从vms2.0会带来的功能特性来加以介绍：

## 功能点展示

### 可靠性和并发性能提升([基于Kafka实现消息可靠的ack机制](http://wiki.corp.vipshop.com/pages/viewpage.action?pageId=397345662))

原生客户端现状：

* Kafka自动提交offset模式下(**enable.auto.commit=true**)，在业务消费逻辑异常时极易产生消息丢失；
* 采用手动提交模式下(**enable.auto.commit=false**)，业务实现者需要处理各类异常情况，以保证正确提交进度；同时一旦破坏了原生客户端线程模型，极易产生消费进度覆盖而引起消息丢失的情况；
* 原生客户端的线程模型仅支持单partition被单线程消费的模式，对于partition积压情况很难进行并行消费；

解决方案：

* Vms2.0客户端中禁用自动提交模式，由vms的commit线程来进行手动提交消费进度；从用户角度来看都是自动提交方式，用户指定提交间隔时间来设定提交频度；
* Vms2.0中采用滑动窗口来管理所有pull进来的消息，按照滑动窗口的最小offset位置进行进度的提交；
* 滑动窗口内的消息可以并发的被消费线程来处理，很好避免多线程消费时进度处理不当导致消息丢失问题 (进度覆盖造成消息的丢失)；同时也支持顺序消费的场景。
* 用户实现callback函数通过状态返回值和内部提交线程进行交互， 业务实现者无需关注消费进度复杂的处理逻辑；

### 重试队列和死信队列支持

原生客户端现状：不支持该特性

解决方案：

* 重试队列：

队列消费过程中一旦有消息异常引起堵塞，会提高业务逻辑处理的复杂度(是否本地重试，如何缓存本地，如果crash如何rebalance等等一系列麻烦要处理)，随着这些异常处理会较大的影响系统的吞吐能力；

同时需要考虑重试的时间间隔问题，业务处理异通常由于下游业务调用失败引起，而这些依赖的服务一般不会立即恢复，采用一定延时后再消费，成功概率相对较高；

重试队列还有一种机制就是当前服务实例可能某种原因消费不了该消息(比如网络问题)，通过重试机制可以failover到剩余的服务实例进行消费；

* 死信队列：

某些消息可能协议问题，重试满最大次数后，不应该继续耗用系统资源，有可能该消息就是错误的数据，为避免陷入重试的死循环，采用达到最大重试次数后转为死信消息保存到特殊队列，供后期审计和问题排查使用；

具体逻辑**：**

用户消费失败情况下，该消息会被内置producer根据重试的次数发向不同延时的topic(内置20个延时等级的topic)并在消息扩展属性中携带了真实发往的topic信息(此处就是RETRY.[groupid]构成的重试队列名称)；在vms的sdk消费端默认会订阅该重试topic来达到上述目的； 对于发送异常的消息(概率较小)，本地会进行存储并本地重试以解决此类问题；

消息被重试最大次数(目前20次，默认重试延迟累计约4个小时，当然用户可指定每次延迟的时间[20\*5s~20\*2h]的范围)后，消息会被发往死信队列进行保存。

### 支持消息查询功能

原生客户端现状(AdminClient)：不支持该特性

解决方案：1. adminclient中新增消息查询接口，支持offset和timestamp的查询方式；

2. console界面支持查询UI方便用户实时查询需求

具体逻辑：

1. *Offset查询方式：*通过构建kafka支持的FetchRequest协议来动态拉取对应的消息内容；
2. *Timestamp查询方式：*首先通过kafka的ListOffsetRequest来获取对应时间点的offset位置，再通过offset查询方式获取对应的消息集；

### 支持消息回溯功能

原生客户端现状(Consumer端)：支持单实例的回溯方式(指定offset或最大/最小offset)，代码级别的回溯；缺乏按时间timestamp回溯接口。

解决方案：对原生回溯接口进行扩展， 可通过配置下发的方式， 通知到所有消费实例进行相关回溯操作；

具体逻辑：

1. 扩展按时间timestamp方式进行回溯的接口，用户指定查询时间消费进度回滚至对应进度继续消费；
2. 客户端监听配置中心回溯属性值变化，当获得属性更新时，调用回溯接口执行相关动作(平台级的下发逻辑)，可覆盖所有运行实例；

# 支持消息过滤机制

原生客户端现状(Consumer端)：不支持该特性

解决方案及相关逻辑：

1. 复用相同topic处理不同业务处理时， 业务可通过订阅不同routekey来消费各自的消息，类似rabbitmq的direct路由模式；
2. 客户端进行消息过滤，通过模式匹配的方式匹配routekey来判断是否需要投递给业务方；
3. 需注意流量的开销，过滤发生在客户端，意味着网络流量存在一定的浪费，从目前业务域的吞吐来看，暂无需考虑此类问题；
4. 客户端的性能和延迟问题，客户端采用批量的拉取方式，空消费批量拉消息(1k)的单实例性能可达到(30w+tps)因此，目前能够满足现状。

# 支持组内广播机制([基于kafka1.0实现组内广播功能设计](http://wiki.corp.vipshop.com/pages/viewpage.action?pageId=467602208))

原生客户端现状(Consumer端)：不支持该特性（仅支持组间广播）。

解决方案及相关逻辑：

1. 原生客户端中，同一groupid的多实例消费时，原生客户端采用的RangeAssignor方式来分配partition， 各实例分配的partition是互斥的不能重叠， 各自负责分配的partition的消息消费和进度提交；
2. 要实现同groupid下多实例的广播消费，需实现新的partition assignor逻辑来支持广播的分配方式，所有的实例需分配到全部partitions；同时进度提交上需要每个实例单独维护，我们采用类似rocketmq的做法采用本地文件进行存储；

# 支持消息灰度机制([Angelia灰度流量1.0](http://wiki.corp.vipshop.com/pages/viewpage.action?pageId=416944571))

原生客户端现状(Consumer端)：不支持该特性。

解决方案及相关逻辑：

1. 未灰度的消费端实例在消费消息时， 通过开关判断是否开启了灰度， 若开启了灰度按灰度比例对消费的消息进行过滤处理， 选中灰度的消息会转发至灰度的topic队列；
2. 灰度的消费实例，启动时只订阅灰度topic，此时只消费灰度topic内的消息达到灰度的目的；
3. 所有灰度控制都通过配置中心的属性下发来实现。

# 支持延时消息发送([基于kafka实现延时队列的设计](http://wiki.corp.vipshop.com/pages/viewpage.action?pageId=384635267))

原生客户端现状(Producer端)：不支持该特性

解决方案：

1. Kafka服务端实现定时扫描固定延时topic(20个级别的topic)的服务，每个级别的topic都有自身固定的延时等级，当判断延时符合投递条件时发送到真实的topic队列；
2. Vms客户端在发送延时消息时，根据用户指定的延时等级选择不同延时topic进行消息发送，消息的header信息中携带真实要发的topic属性供service调用转发消息；

具体逻辑：

1. Service需要解决时间精度问题，目前采用服务端appendlog的timestamp来进行判断；另外消息积压时延时消息精度的保证处理，目前采用delay queue来处理到期要发的消息，以后可能考虑timewheel的方式来优化。
2. Service端要处理好各类异常情况以及消费进度提交的问题；
3. 客户端发送逻辑需考虑不同延时选择不同topic发送以及设置真实topic属性值。

# 支持事务性消息发送

原生客户端现状(Producer端)：支持事务消息发送。1.支持消息的原子性发布；2.支持发送端的消息幂等性，避免网络故障引起消息的重复存储。

解决方案及相关逻辑：vms2.0客户端增加相关事务发送接口，来支持事务性消息；

# 支持配置中心的元数据管理

解决方案和相关逻辑(VMS sdk)：变更zookeeper元数据管理方式， 切换至配置中心来进行vms的元数据管理服务， 并实时监控相关配置信息的下发；

# 支持mercury-metrics的监控指标监控

解决方案和相关逻辑(VMS sdk)： 以前采用hermes-metrics进行指标的统计和输出， vms2.0将切换到mercury-metrics的监控指标埋点；

# Vesta资源申请管理平台UI全面升级

1. 用户资源申请及后期维护管理以及不同环境的配置问题，在逻辑上进行了重新的划分和梳理，方便用户更容易找到本系统相关资源的统计和实时监控信息；
2. 进行了权限的管理机制；
3. 元数据线上线下的同步以及域信息的管理工作；
4. 监控指标更进一步做了细化和采集，采集指标维度更丰富，方便用户实时排查和发现问题；
5. 移除了以前过度依赖配置中心实现的消息管理的相关功能点，统一归纳到angelia平台上，为后期维护和扩展新功能提供了便捷；
6. 移除了对rabbitmq资源的管理，目前angelia平台只支持对kafka引擎的支持；

# 服务端（broker）监控指标采集后端服务改进

1. 增加对kafka1.0+更多维度监控指标的采集；
2. 替换hemers组件为mercury-metrics组件进行数据上报功能；
3. 移除rabbitmq指标采集相关模块；
4. 移除zookeeper的依赖逻辑(新增集群或channel,queue资源变化等动态监测和对应服务启停工作)， 采用配置中心+redis集群的方式来实现；

## 未来规划

# 完善mercury统一监控入口，为应用域提供vms资源的专项监控页面，提供详尽的监控指标输出

1. 为应用域监控提供全面的指标信息输出；可以浏览到与vms相关的各类资源信息及配置信息
2. 查看与channel或queue相关的消息类查询信息以及trace等信息资源

# 完善客户端trace埋点和多维度查询服务埋点

1. Trace信息的上下游传递，上游获取服务traceid和span信息存储消息header，下游消费线程消费时恢复相关trace和span信息至thread context环境保证调用链完整；
2. 消息层面trace信息和多维度信息的协议制定，以及输出采集模式确立；

# 完善消息服务质量提升相关流处理平台建立

1. 埋点信息全集群的归集和实时计算平台的处理，生成为多维度查询的ES索引服务；
2. 消息层面trace和审计相关的流计算处理和存储，提供查询service供用户可实时检索相关消息的消费情况； 以及消息审计结果输出和告警通知等事项；