**开源软件成熟度评测报告-分布式消息中间件**

# 背景

随着互联网技术和金融科技的不断发展，从RPC到Web Service，从SOA的推行再到RESTful以及云计算中PaaS与SaaS的推广，分布式架构在金融企业中得到了广泛应用，消息中间件则在分布式系统之间的通信、集成和整合上发挥了关键作用。分布式消息中间件通过高效、可靠的消息传递机制，降低应用系统之间的耦合性，实现高性能的数据交换，保障了分布式计算网络环境下高可用和一致性。

面对诸多的分布式消息中间件，金融企业面临如何选择并确定适合企业长期发展的相应开源软件。金融行业开源软件研究工作组结合金融企业的实际应用场景，针对主流的开源分布式消息中间件建立评测并开展评测实施，以支撑金融企业选择成熟度高、适合企业需求的开源软件。

# 分布式消息中间件评测模型

分布式消息中间件评测模型基于金融行业开源软件成熟度评测整体模型建立。整体模型充分结合了开源软件的特性、系统工程领域对于软件产品质量的要求以及金融行业对于开源软件的使用需求。**整体模型涵盖开源许可证、行业认可度、产品活力、服务支持、安全性、兼容性、可维护性、可扩展性、功能性、可靠性、易用性、性能效率等12大类评估属性，117个评测指标**。

分布式消息中间件主要用于分布式系统架构中，进行应用系统间的数据传输，已经在各种行业中广泛应用。开源分布式消息中间件种类很多，各有侧重，在引入时需要结合具体的应用场景，选择性价比最高的开源产品。因此，需要对分布式消息中间件进行全面评估，防范开源软件带来风险，为金融企业更好的应用开源软件提供支撑。

**分布式消息中间件评测模型重点围绕金融企业关注的特性和可评测的指标建立，涵盖整体模型的12方面评估属性**，分布式消息中间件评测模型具体详见已发布的《[***金融业开源软件研究评测——分布式消息中间件评测模型***](http://mp.weixin.qq.com/s/I13duPyGbmOZyp6mKtjjWg)》。

# 分布式消息中间件简介

消息中间件具有消息存储、转发、过滤和排队等功能，在分布式环境下扩展进程间的通信，主要用于业务系统解耦、消息异步传递、错峰流控等场景中。除了传统的商业产品如IBM WebSphere MQ、东方通TongLINK/Q之外，开源消息中间件技术近几年发展迅速，常见的有ActiveMQ、RabbitMQ、ZeroMQ、Kafka、MetaQ、RocketMQ等，并已经在许多行业得到广泛应用。

经过前期对金融企业应用情况和业务需求的调研，以及对当前技术发展趋势的考虑，本次我们选取了快速发展的Kafka，以及应用广泛的RabbitMQ作为评测对象，后续也将对其他消息队列中间件开展评测。

## Kafka

Kafka是由LinkedIn公司在2010年12月开源的一种高吞吐量的分布式消息系统，属于Apache软件基金会的顶级子项目之一，目前已被越来越多的开源分布式处理系统集成。Kafka适用于大规模消息处理的应用场景，具有良好的可扩展性和性能优势。与传统消息系统不同，Kafka还被广泛应用于日志聚合、流式数据处理等场景中。

Kafka具有高性能、高可用、分布式的技术特点。Kafka强大的负载均衡和副本策略保证了节点的可靠性和高可用性，支持节点的动态扩展；在设计实现上与传统消息中间件有较大差异，使用文件系统来管理消息的生命周期，能够在常数时间复杂度内提供消息持久化和数据访问，支持消息的批量发送和压缩传输，性能表现优异。由于其并非作为传统MQ设计，没有遵循主流消息服务规范，因此在事务、协议兼容等方面有所欠缺。

## RabbitMQ

2007年，基于AMQP标准的RabbitMQ 1.0版本由Rabbit技术公司发布，2010年被SpringSource收购，2013年并入Pivotal，现由Pivotal Software提供商业支持。RabbitMQ是一种基于Erlang实现AMQP协议的开源消息中间件，它提供了功能强大的消息队列服务，常用于Web服务器快速响应请求，适合跨平台、跨语言的消息传输。

RabbitMQ具有消息可靠传输、灵活路由策略、多协议支持等特点。RabbitMQ具有健壮的消息确认机制、用户角色体系、以及认证和授权管理功能，保障消息可靠传输。灵活的交换器和绑定规则设置提供了强大的消息路由功能，同时支持AMQP、HTTP、STOMP、MQTT等协议。此外，RabbitMQ多节点集群的联合不依赖外部服务，支持服务的高可用，但服务的负载均衡需要使用第三方组件。

# 评测环境

根据实际业务使用情况和开源软件主流的稳定版本，选择待评测的开源软件，如下表中所示。

表1 软件版本信息

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 评测版本 | 0.11.0 | 3.6.10 |
| 开发语言 | Scala、Java | Erlang |
| 使用协议 | TCP层二进制协议 | AMQP |
| 官网地址 | <http://kafka.apache.org/> | <https://www.rabbitmq.com/> |
| 源码地址 | git://git.apache.org/kafka.git | <https://github.com/rabbitmq/rabbitmq-server> |

参考生产环境中业务系统的实际运行环境搭建评测环境，针对Kafka和RabbitMQ各搭建3节点物理服务器的验证环境，每台服务器配置如下表。

表2 测试环境信息

|  |  |
| --- | --- |
| **指标** | **参数** |
| CPU | Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz\*2 |
| 内存 | 128G |
| 操作系统 | CentOS 7.2 |
| 硬盘 | 600G 10Krpm SAS\*2 |
| 网络带宽 | 10Gbps |

# 开源软件许可证对比

开源软件许可证是针对开源软件的授权条款，规定了包括商业用途、分发、修改、专利授权、私用、公开源码、放置许可协议与版权信息、使用网络分发、使用相同协议、声明变更、承担责任、使用商标等12方面的使用要求与限制。用户使用开源软件进行商业化时，需遵守开源软件许可证条款的规定。

经过扫描Kafka和RabbitMQ开源软件源代码，Kafka使用的开源许可证为Apache 2.0，RabbitMQ使用的开源许可证为MPL（Mozilla Public License） 1.1。

表3 开源软件许可证对比

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 使用的许可证 | Apache2.0 | MPL1.1 |
| 允许基于开源项目提供服务 | 是 | 是 |
| 修改源代码是否允许闭源，再发布是否可以采用不同许可证 | 是 | 否 |
| 需声明修改部分版权归属 | 是 | 是 |
| 贡献者使用专利保护源代码 | 不允许 | 不允许 |
| 授权用户使用商标 | 否 | 否 |

Kafka遵循的Apache2.0协议对于开源软件源代码使用的限制较少，允许用户免费使用、修改和发布衍生产品，不会面临版权或专利方面的风险，也无需公开源码。RabbitMQ采用的MPL 1.1协议对于开源软件源代码的使用相对严格，虽然可以免费修改和无偿使用，但修改的源代码必须开源，且要求继续使用MPL 1.1协议。同时，RabbitMQ遵循的MPL 1.1协议要求修改后的代码版权归开源软件的发起者。

结果表明，**RabbitMQ与Kafka相比，存在修改/衍生产品的源代码必须开源，再发布必须遵守原开源许可证、代码版权归属开源软件发起者等问题。因此，企业如需发布包含消息中间件的闭源商业产品，优先选择Kafka。**

# 产品活跃度对比

开源软件产品的活跃度反映开源软件的可持续性和可进化的能力，主要从开源软件的版本发布情况、开源社区情况、软件的关注情况等方面进行分析。

## 软件版本发布：Kafka出现晚，近两年更新较快

开源软件版本发布情况反映了开源软件发展的稳定性和可持续性。模型通过版本号、代码量和发布时间等方面分析开源软件版本发布情况。通过对开源软件的源代码进行分析，得到近两三年Kafka、RabbitMQ版本发布情况如下。

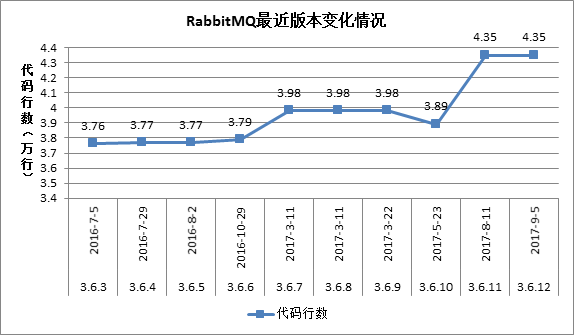
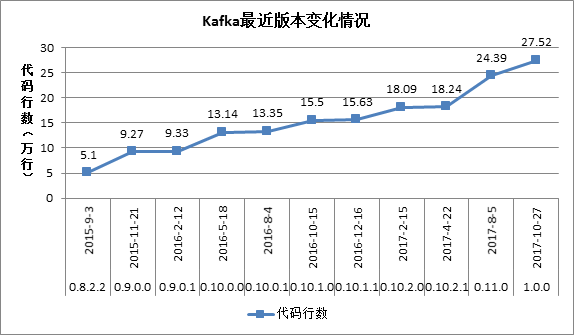


图1 软件版本、代码变化情况

Kafka 1.0.0版本于2017年10月份发布，代码行数相比两年前增长了五倍多，版本迭代比较快。通过对近两年Kafka各个版本进行研究发现，版本更新主要是修复了之前版本的缺陷，优化了业务处理的逻辑，增加了诸如分区自动均衡调整、安全认证、加密传输、事务、EOS（exactly-once semantics）等新特性。

RabbitMQ从2012年发布3.0版本至今，代码行数仅增长30%左右，版本迭代比较稳定。通过对近两年RabbitMQ各个版本进行研究发现，版本更新主要是对服务端、管理插件、MQTT插件、STOMP插件、客户端等的缺陷/漏洞进行修复和功能优化，主体功能更新较少，代码量变化较小。

结果表明，**当前Kafka和RabbitMQ版本发布均比较稳定，Kafka代码增长较快，RabbitMQ代码增长则趋于平稳。**

## 开源社区情况：Kafka贡献者人数是RabbitMQ的近五倍

开源社区情况反映了开源软件的活跃情况，以及未来一段时间内的开源软件发展情况。模型通过开源软件的贡献者、提交量、贡献者等级三个方面分析评估开源软件的社区情况。通过对开源社区情况进行分析，得到Kafka、RabbitMQ开源社区情况如下。

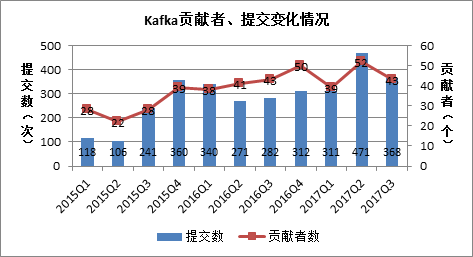
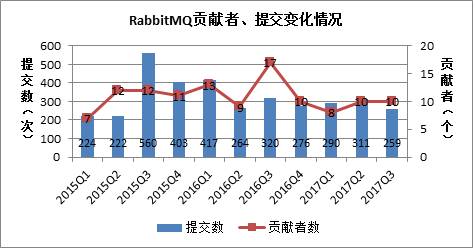
 

图2 软件贡献者、提交数变化情况

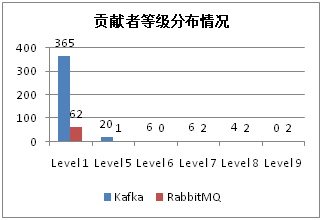


图3 软件贡献者等级分布情况对比

从贡献者数量看，Kafka每个季度贡献者数量是RabbitMQ的两倍到五倍；从提交数量看，Kafka和RabbitMQ每个季度的提交数量差异不大；从贡献者等级分布情况看，Kafka贡献者人数远超RabbitMQ（约六倍）。

结果表明，**Kafka在开源社区的贡献者人数和贡献者等级方面，超过RabbitMQ。在提交量方面，两者均有稳定的提交。**

## 软件关注度：Kafka的关注者和使用者数量超过RabbitMQ

开源软件关注度反映了开源软件的吸引力和实际应用情况。模型通过关注源代码的开发者数量、开发者提出的需求数量和博客论坛书籍数量等分析软件的关注度。通过对软件相关信息进行分析，得到软件关注度信息如下所示。

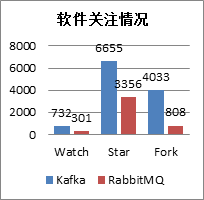
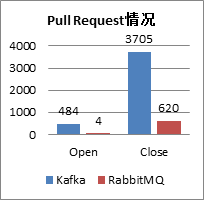
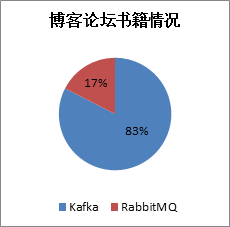
  

图4 软件关注度对比

从软件关注情况看，目前关注Kafka的开发者人数是RabbiMQ的两至三倍。在软件需求数量方面，Kafka是RabbitMQ的六七倍。从博客论坛书籍数量看，Kafka的博客、论坛、书籍等方面的资料数量是RabbitMQ的近五倍。

结果表明，**Kafka相比RabbitMQ来说，有更多的关注者和使用者**。

# 行业认可与服务支持对比

行业认可与服务支持反映开源软件在业界的应用情况和提供专业化服务的情况。通过对应用集成方案和云计算服务两方面进行分析，结果情况如下表所示。

表4 行业认可与服务支持情况

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 应用系统集成 | 支持集成大数据平台（如Hadoop、HDFS、Flume）、流处理平台（如Storm、Spark、Flink）、搜索引擎（如ElasticSearch、Presto、Hive）等 | 支持与开发配置框架（如Spring、Puppet、Docker）等的集成 |
| 云计算服务 | 已有公有云提供基于Kafka项目的消息服务（腾讯云CKafka、百度云Kafka、阿里云Kafka） | 提供RabbitMQ云服务（如CloudAMQP, Google Cloud Platform等） |

结果表明，**Kafka和RabbitMQ已经有了很多商业化实践或应用案例，广泛应用于云计算、大数据等场景中。Kafka在云计算、大数据平台中应用更加广泛。**

# 功能特性对比

消息中间件主要用于分布式应用之间的信息交换，消息的生产者将消息发送至服务端，消息保存在服务端的内存或磁盘上，直至消费者将消息取走。针对消息中间件的功能，主要从服务端、生产者、消费者、以及消息传输等四个方面对Kafka和RabbitMQ的功能进行评测。

## 服务端特性: Kafka具备核心功能，RabbitMQ功能更全面

服务端主要负责消息、队列、集群、安全等方面的管理和应用的服务请求处理等。服务端的技术特性主要包括消息、队列和集群三个方面。

表5 服务端技术特性对比

|  |  |  |  |
| --- | --- | --- | --- |
| **对比项** | | **Kafka** | **RabbitMQ** |
| 消息 | 消息结构 | 基于主题、分区、文件结构 | 基于虚拟主机、交换器、队列结构 |
| 消息持久化 | 支持（持久化到磁盘） | 支持（默认存储在内存，可选持久化到磁盘） |
| 顺序存储 | 支持（分区级别上保证消息写入和读取顺序一致） | 不支持 |
| 消息堆积 | 支持 | 支持 |
| 消息死信 | 不支持 | 支持（消费者不能处理的消息发布到另一个队列，等待重试或人工干预） |
| 消息删除 | 支持（消息到一定时间或一定大小后，无论是否被消费都会删除） | 支持（消费者确认后删除消息，即使是持久化消息） |
| 队列 | 队列定义 | 支持（通过ZooKeeper、生产者定义分区） | 支持（通过Web、生产者、消费者定义） |
| 远程队列定义 | 不支持 | 支持（生产者或消费者未指定队列或队列不存在，节点自动创建队列） |
| 消息路由 | 不支持（通过主题对消息进行分类发送） | 支持（通过多种类型的交换器实现高级消息路由） |
| 集群 | 集群管理 | 支持（通过ZooKeeper管理） | 支持（基于Erlang的分布式架构） |
| 节点类型 | 管理节点（ZooKeeper）、Kafka节点 | Disc节点、RAM节点 |
| 元数据存储 | Zookeeper节点 | Disc节点、RAM节点 |
| 集群模式 | 选举模式（通过ZooKeeper管理） | 普通模式、镜像模式 |
| 权限和访问控制 | 支持（使用ACLs进行访问控制，由Zookeeper保存授权信息） | 支持（用户在虚拟主机的粒度下进行权限控制，必须给用户指定虚拟主机和角色） |
| 同一数据集不同消费者 | 支持（通过Zookeeper和消费者组实现多个消费者消费一份数据） | 支持（通过将数据复制到多个队列中给消费者消费） |

对于Kafka，在消息处理方面，将消息持久化到磁盘，并通过分布式的顺序读写磁盘，提高了消息读写的效率**。**在队列方面，通过主题、分区进行消息的生产和消费，不支持分区的远程定义和消息路由，并且仅能通过消费者组(Group)进行消费。在集群方面，使用Zookeeper对分布式集群进行协调管理，实现节点分区均衡分布、节点选举、协调生产者和消费者、并提供各项配置服务。

对于RabbitMQ，在消息处理方面，提供内存和磁盘两种方式保存消息，使用内存方式处理消息的效率更高。在队列方面，支持生产者和消费者的队列远程定义，并能够通过交换器（Exchange）灵活的配置消息的路由策略。在集群方面，基于Erlang 的分布式特性构建集群，每个节点为对等节点，向客户端提供服务，通常需要第三方负载均衡组件做负载均衡和失效转发。

结果表明，**Kakfa具备消息中间件的核心功能，并进行分布式和消息处理的优化，而RabbitMQ的功能更加全面，提供了更多的功能实现消息的灵活处理**。

## 生产者特性：Kafka和RabbitMQ均具备核心功能

生产者指产生消息的应用，生产者产生消息后，将发送给服务端，服务端采用磁盘或内存等方式保存消息，同时保存消息副本。生产者特性对比如下表所示。

表6 生产者技术特性对比

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 消息生产模型 | Push模型 | Push模型 |
| 消息同步传输 | 支持 | 支持 |
| 消息异步传输 | 支持 | 支持 |
| 批量传输 | 支持 | 不支持 |
| 数据压缩 | 支持（将多条消息批量压缩后发送和存储） | 不支持 |
| 消息确认 | 支持（ack机制，确保数据保存到磁盘） | 支持（confirm机制，确认消息正确到达队列） |
| 消息重试 | 支持 | 支持 |
| 消息有效期 | 不支持 | 支持 |

从表中可以看到，Kafka提供了批量传输和数据压缩功能，批量传输能够一次性传输多条消息，数据压缩能够降低对网络带宽的要求，从而提高消息的处理能力。在传输的可靠性方面，RabbitMQ提供Confirm确认机制，确认生产者客户端的消息正确送达队列；Kafka提供Ack机制，确认数据正确接收并保存到磁盘。

结果表明，**Kafka和RabbitMQ均具备核心功能，但是两者在高级功能上各有侧重。**

## 消费者特性：Kafka使用Pull模型，RabbitMQ使用Push模型

消费者指接收和处理消息的应用，消息发送给消息队列中间件服务端后，转发给消费者，进行其他业务处理。消费者特性对比如下表所示。

表7 消费者技术特性对比

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 消息消费模型 | Pull模型 | Push模型 |
| 数据解压缩 | 支持（支持gzip、snappy、LZ4等压缩格式） | 不支持 |
| 消息过滤 | 不支持 | 支持 |
| 消息重新消费 | 支持（消息保存在磁盘，指定消息的偏移值即可重新消费数据） | 不支持（消费者确认后会删除该条消息） |
| 消息确认 | 支持（消费者自动/手动提交消息消费结果） | 支持（通过消息Ack机制保证消息从队列可靠地到达消费者） |
| 预读取计数 | 不支持（Pull模型，消费者控制消息的读取速度和数量） | 支持（当未确认消息数量达到预取数，将停止在该消息通道上传送更多消息，直到有消息消费被确认） |

在消息消费模型方面，Kafka使用Pull模型，消费者根据需要的消息，从服务端Pull数据；RabbitMQ采用Push模型，消费者与服务端队列保持长连接，队列中有消息则会Push至消费者。

在消息确认方面，Kafka提供消息至少发送一次(“at least once”），确保消息被可靠消费; RabbitMQ提供消息确认机制（Ack），直到消费者显式返回Ack才移除消息，确保消息到达消费者。此外，RabbitMQ还提供消息过滤，预读取计数等功能。

结果表明，**Kafka与RabbitMQ采用不同的消费模型，在消费者获取消息的灵活性和可靠性方面，RabbitMQ提供的功能更加全面。**

## 消息传输特性：Kafka具备核心功能，RabbitMQ功能更全面

消息传输模式主要包括点对点(PTP)模式和发布-订阅(Pub/Sub)模式。在消息传输方面，Kafka和RabbitMQ技术特性对比如下。

表8 消息传输技术特性对比

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 点对点(PTP)模式 | 支持 | 支持 |
| 发布-订阅(Pub/Sub)模式 | 支持 | 支持 |
| 事务 | 支持 | 支持 |
| 消息优先级 | 不支持 | 支持 |
| 延时/定时消息 | 不支持 | 支持 |
| JMS规范支持 | 不支持 | 支持 |

Kafka和RabbitMQ均支持点对点模式和发布-订阅模式，但在实现机制上，Kafka基于消费者组(group)实现，RabbitMQ则是基于AMQP协议。另外，RabbitMQ还支持消息优先级、定时消息和JMS规范等功能。

结果表明，**Kafka能够满足主要的消息传输要求，但RabbitMQ支持更多高级特性。**

# 性能效率

消息中间件的性能效率主要指集群接收和发送消息的吞吐率。吞吐率指标有两个，一是生产者/消费者每秒发送/接收的消息条数，单位为条/秒（Record/s）；二是生产者/消费者每秒发送/接收的消息量，单位为MB/s。我们通过消息的发送吞吐率和接收吞吐率两方面评估消息队列中间件的性能，主要测试场景如下：

（1） 客户端并发数：不断增加生产者/消费者数量，测试发送/接收吞吐率的变化情况；

（2）消息大小：不断增加消息大小，测试发送吞吐率的变化情况；

（3）系统可靠性：通过增加副本数、改变应答模式，测试发送吞吐率的变化情况；

（4） Kafka特性：测试Kafka消息批量发送、数据压缩、分区数等特性对吞吐率的影响。

## 不同生产者数量，Kafka吞吐率均优于RabbitMQ

测试消息发送吞吐率随生产者数量的变化情况。设定消息大小为1000字节，不断增加生产者线程数直至吞吐率不再上升，测试结果如下图所示。

图5 生产者数量增加对吞吐率的影响

测试结果显示，Kafka的发送吞吐率开始呈线性增长，到达峰值后在小范围内波动，峰值约32w条消息每秒。RabbitMQ的发送吞吐率开始持续增长，之后趋于稳定，峰值约12w条消息每秒。当Kafka生产者数量小于分区数时，每个生产者可单独向分区发送消息，因此吞吐率呈直线上升。当生产者多于分区数时，触发各分区间的负载均衡，吞吐率出现波动。RabbitMQ增加生产者，系统并行度增加，吞吐率稳定上升后逐渐达到稳定。

结果表明，**在相同生产者数量的情况下，Kafka发送吞吐率显著高于RabbitMQ，基本在三倍左右。**

## 不同消息大小，Kafka吞吐率均优于RabbitMQ

测试消息发送吞吐率随消息大小的变化情况。保持生产者、消费者等条件不变，令消息大小从100字节不断增加至10000字节，测试结果如下图。

图6 消息大小变化对吞吐率的影响

测试结果显示,无论是Kafka还是RabbitMQ，消息越大，每秒发送的消息条数越少，但每秒传输的数据量越大。这是由于每条消息除了有效负载，还包含一些元数据（Metadata），有效负载越大则元数据占比越小，因此虽然消息条数减少，但每秒传输的数据量增加。

结果表明，**相同消息大小时，Kafka的发送性能超过RabbitMQ近一个数量级，并且Kafka在发送小消息时优势更加明显。**

## 不同副本数和应答模式对Kafka的性能影响相对更小

测试副本数和应答模式对消息发送性能的影响。令副本数从0增加至2，分别测试不同应答模式下吞吐率的变化情况。

图7 副本数和应答模式对吞吐率的影响

测试结果显示,Kafka在等待数据写入主、备分区后确认时，副本数从0增加至2吞吐率下降了7成；另两种应答模式所受影响较小。RabbitMQ在无备份时相对于Kafka性能表现一般，开启镜像模式、副本数增加后吞吐率下降幅度超9成。

结果表明，**Kafka在提升系统可靠性时性能损失更小，相对于RabbitMQ具有更好的副本同步机制。**

## 不同消费者数量，Kafka接收速率优于RabbitMQ

测试消息接收吞吐率随消费者数量的变化情况。设置消息大小为100字节，逐渐增加消费者数量，吞吐率变化情况如下图所示。

图8 Kafka压缩模式对吞吐率的影响

测试结果显示，由于Kafka同一组消费者订阅的数据只能分配给组中的一个成员，由各个成员共同维护消费进度，因此消费者分布在不同组时才可对同一分区的数据并行消费，此时消费吞吐率得到明显提升。对于RabbitMQ，当多个消费者在不同队列订阅同一数据时，需要将消息投递到多个队列中，会增加消息复制的时间。因此，RabbitMQ所有消费者消费同一个队列中的数据时，吞吐率明显比消费不同队列数据时的吞吐率高。

结果表明，**Kafka的消息接收性能要显著优于RabbitMQ。同时，Kafka提供并行消费的能力，可使消费速率随消费者数量的增加而线性增长。**

## Kafka消息批量发送、数据压缩等特性对吞吐率有一定提升

Kafka具有**消息批量发送、数据压缩、主题分区、批量接收**等RabbitMQ不具备的特性，该场景下测试Kafka不同批处理大小、分区数或压缩模式下吞吐率的变化情况。前述性能测试场景中，Kafka采用默认配置，批量发送条数为16384，压缩模式为无压缩，fetch size为1048576字节。分区数推荐设置为节点数的整数倍，测试过程中设置为9。

图9 Kafka每批次发送条数对吞吐率影响 图10 消费者单次接收消息量对性能的影响

图11 Kafka压缩模式对吞吐率的影响 图12 Kafka分区数量对吞吐率影响

测试结果显示，随着批处理大小增加，Kafka吞吐率显著提升，这是因为消息批量发送可以减少服务端的I/O次数和网络传输开销。同时，不断增加消费者单次接收的消息量（Bytes），Kafka接收吞吐率得到进一步提升，建议根据消费者的业务处理需求自主选择消费模式。

Kafka在Snappy、LZ4压缩模式下吞吐率有所提升，GZIP压缩模式相比于无压缩状态性能反而下降。端到端数据压缩在理论上重复数据越多压缩效果越好，网络带宽占用小，但压缩率并不是越高吞吐率越好，提升压缩率会增加压缩和解压的时间开销。

分区数不断增加，Kafka发送吞吐率显著提升后逐渐降低。Kafka的分区结构为系统提供并行消费能力，但当分区数过大时发送吞吐率会显著降低。根据Kafka的负载均衡策略，分区数配置为节点数量的整数倍能够提高资源利用效率。

结果表明，**Kafka消息批量发送和批量接收均能提升吞吐率；压缩模式对吞吐率的影响取决于压缩算法的选择；分区数适当增加能够提升吞吐率，过大则会降低吞吐率。**

经过上述多个场景的性能测试，可以看到**RabbitMQ和Kafka均有不错的性能表现，但Kafka整体上比RabbitMQ具有更高的吞吐率，特别是节点多副本备份、并行消费等场景下Kafka优势十分明显。**

# 安全性对比

消息中间件的安全性主要包括已暴露的安全漏洞情况和安全机制两方面。对Kafka和RabbitMQ的漏洞信息和安全机制进行评测，结果如下表所示。

表9 安全漏洞和安全机制对比

|  |  |  |  |
| --- | --- | --- | --- |
| **对比项** | | **Kafka** | **RabbitMQ** |
| 已暴露的安全漏洞 | | 暂无 | 近三年暴露了14项漏洞（包括一项严重漏洞，两项高危漏洞） |
| 安全机制 | 连接认证 | 支持 | 支持 |
| 认证机制 | 支持SSL和SASL | 支持SSL和SASL |
| 用户体系 | 不支持 | 支持 |
| 加密通信 | 支持 | 支持 |
| 集成外部认证插件 | 支持 | 支持 |

从表中可以看到，Kafka和RabbitMQ在安全机制方面均比较全面。在已暴露的安全漏洞方面，Kafka到目前为止尚未发现已披露漏洞，而RabbitMQ已暴露14个安全漏洞，主要漏洞包括可绕过安全机制获取未授权访问权限、存在跨站脚本漏洞、通过读取日志获取敏感信息、造成拒绝服务等。

结果表明，**从安全机制和已暴露的安全漏洞两方面看，Kafka相对RabbitMQ来说更加安全。**

# 可扩展性对比

消息中间件的可扩展性主要包括节点的水平扩展、动态扩展，以及负载均衡能力的能力。根据对Kafka和RabbitMQ的测试结果，扩展性区别如下表所示。

表10 扩展性对比

|  |  |  |  |
| --- | --- | --- | --- |
| **对比项** | | **Kafka** | **RabbitMQ** |
| 水平扩展 | | 支持 | 支持 |
| 动态扩展 | | 支持 | 不支持 |
| 负载均衡 | 生产者负载均衡 | 支持（通过分区和副本平均分布在各节点上实现） | 支持（由第三方负载均衡组件HAProxy、LVS等进行支持） |
| 消费者负载均衡 | 支持（基于Zookeeper和组管理节点、消费者的动态加入与离开，触发负载均衡） |

在节点扩展方面，Kafka采用分布式架构，通过Zookeeper对节点进行管理，实现节点的动态扩展；RabbitMQ节点的动态扩展则需要第三方组件的支持。在负载均衡方面，Kafka通过分区和副本在节点的均匀分布策略实现生产者客户端的负载均衡，通过为每个消费者均匀分配消费分区和自动触发负载均衡实现消费者客户端的负载均衡，有效的平衡集群节点间的消息负载；RabbitMQ需要采用第三方负载均衡组件，因此扩展节点后需配置节点信息并更新相应的负载均衡策略，在扩展的可用性上弱于Kafka。

结果表明，**Kafka的动态可扩展和负载均衡策略方面均优于RabbitMQ。**

# 可靠性对比

消息中间件的可靠性主要包括节点故障时集群、数据可用，以及消息传输过程中的确认机制和事务支持。根据对Kafka和RabbitMQ的测试结果，区别如下表所示。

表11 可靠性对比

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 节点故障集群可用 | 支持（基于分布式架构） | 支持（基于镜像队列） |
| 节点故障数据可用 | 支持（基于消息持久化和消息副本） | 支持（基于消息副本和可选消息持久化） |
| 消息确认机制 | 支持 | 支持（基于发送者和消费者的消息确认机制） |
| 事务 | 支持 | 支持（基于AMQP事务机制） |

在节点故障时的集群、数据可用方面，Kafka采用副本、选举机制等方式保障节点和数据的高可用；RabbitMQ使用镜像备份机制保证节点和数据的可靠。在消息多副本同步和选举策略上，Kafka比RabbitMQ更加灵活和完善。在消息传输可靠性保障方面，Kafka在0.11.0版本上也增加了对事务的支持，同时提供去重机制（exactly-once），提供了更好的消息传输可靠性；RabbitMQ实现了基于AMQP的事务机制、生产者和消息者的多种消息确认机制。

结果表明，**Kafka和RabbitMQ均能对节点、数据的高可用和消息可靠传输提供保障。**

# 其他特性对比

消息中间件的选择还需要评估可维护性、兼容性和易用性等特性。Kafka和RabbitMQ总体情况对比如下表所示。

表12 其他特性对比

|  |  |  |
| --- | --- | --- |
| **对比项** | **Kafka** | **RabbitMQ** |
| 可维护性 | 良好（代码注释规范，管理、监控、测试工具齐全） | 良好（代码注释规范，管理、监控、测试工具齐全） |
| 兼容性 | 一般（TCP层二进制协议） | 强（支持AMQP、STOMP、MQTT等通讯协议规范） |
| 易用性 | 良好（可配置和集成、第三方插件） | 良好（可配置集成、第三方插件、API接口集成） |

结果表明，**作为广泛应用的消息中间件，Kafka和RabbitMQ在工具和集成等方面均能够提供良好的可维护性和易用性。在兼容性方面，Kafka对现有的消息服务协议的支持较弱，RabbitMQ在协议规范方面具有更好的兼容性。**

# 总结

根据分布式消息中间件评测模型，我们从开源软件许可证、行业认可度、产品活跃情况、服务支持情况、功能、性能、安全性、可扩展性、可靠性、可维护性、兼容性和易用性等12个方面，完成了对开源软件Kafka和RabbitMQ的成熟度评测，总结各项评测结果后对比如下。

表13 Kafka、RabbitMQ对比总结

|  |  |
| --- | --- |
| **评测项** | **评测结果** |
| 开源许可证 | Kafka > RabbitMQ |
| 行业认可度 | Kafka = RabbitMQ |
| 产品活跃度 | Kafka > RabbitMQ |
| 服务支持 | Kafka = RabbitMQ |
| 功能 | Kafka < RabbitMQ |
| 性能 | Kafka > RabbitMQ |
| 安全性 | Kafka > RabbitMQ |
| 可扩展性 | Kafka > RabbitMQ |
| 可靠性 | Kafka = RabbitMQ |
| 其他特性 | Kafka = RabbitMQ |

**总体来说，分布式消息中间件Kafka和RabbitMQ在行业认可、服务支持、可靠性、可维护性、兼容性、易用性等方面各有特色。Kafka在开源许可证、产品活跃度、性能、安全性、可扩展性等方面优于RabbitMQ，Kafka采用的许可证更宽松，活跃度更高，性能远高于RabbitMQ，在安全性和可扩展性方面能够提供更好的保障。Kafka仅在功能上略少于RabbitMQ，但是已经具备了主要的功能。**

**综合上述所有评测结果，建议企业在应用过程中优先选择Kafka。**