Pulsar消息队列

一、简介

Apache Pulsar 是 Apache 软件基金会的顶级项目，是下一代云原生分布式消息流平台，集消息、存储、轻量化函数式计算为一体，采用计算与存储分离架构设计，支持多租户、持久化存储、多机房跨区域数据复制，具有强一致性、高吞吐、低延时及高可扩展性等流数据存储特性

1.1 Apache Pulsar的功能与特性:

- **多租户支持**

Pulsar支持多租户，每个租户可以单独设置认证机制、存储配额、隔离策略等。

- **高吞吐与低延迟**

Pulsar通过分区Topic和计算存储分离的设计，实现了高吞吐和低延迟的消息传递。

- **持久化存储**

支持数据持久化存储，确保消息不丢失。同时，支持数据分层式存储，可将数据从热/冷存储卸载到冷/长期存储（oss/hdfs）。

- **多集群部署与数据复制**

原生支持多集群部署，集群间支持无缝的数据复制（Geo-replication），可以实现跨地域的消息传递和消费。

- **高扩展性**

可无缝扩展至上百万个Topic，支持多种topic订阅模式（独占/共享/故障转移）。

- **多语言客户端**

支持多种编程语言客户端，如Java、Go、Python和C++等。

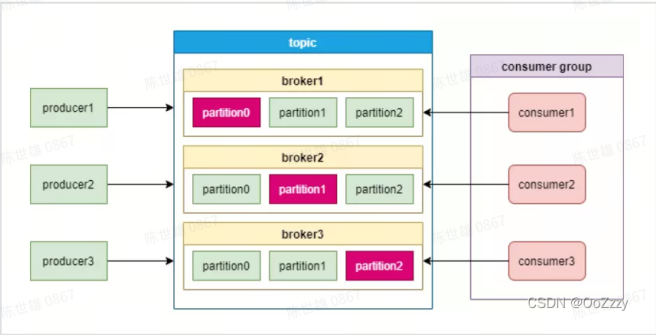
1.2 对Kafka、Pulsar、RocketMQ、RabbitMQ、NSQ这几个消息队列对比

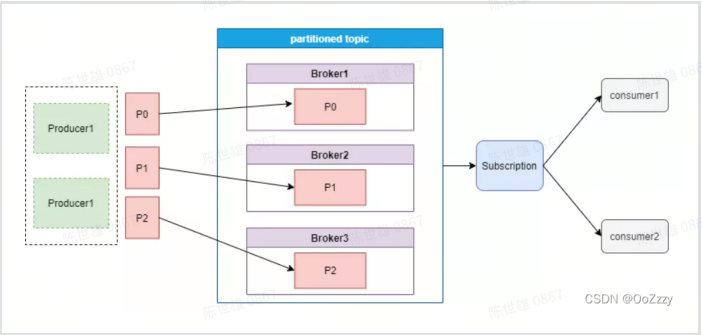
分类	功能项	Kafka	Pulsar	RocketMQ	RabbitMQ	NSQ
功能	消费推拉模式	pull	push	pull	push	push
	延迟队列	✗	✓	✓	✓	✓
	死信队列	✗	✓	✓	✓	✗
	优先级队列	✗	✗	✗	✓	✗
	消息回溯	✓	✓	✓	✗	✗
	消息持久化	✓	✓	✓	✓	✓
	消息确认机制	Offset	Offset+单条	Offset	单条	单条
	消息TTL	✗	✓	✓	✓	✗
	多租户隔离	✗	✓	✗	✗	✗
	消息顺序性	分区有序	流模式有序	消费者加锁	✗	✗
	消息查询	✗	✓	✓	✓	✗
	消费模式	流模式	流模式+队列模式	广播模式+集群模式	队列模式	队列模式
	消息可靠性	request.required.acks	Ack Quorum Size(Qa)	与Kafka类似	镜像模式	mem-queue-size
性能	单机吞吐量	605MB/s	605MB/s	类似Kafka	38MB/s	?
	消息延迟	5 ms	5ms	ms 级	微秒级	?
	支持主题数	几十到几百	上百万个	几百到几千	Thousands	?
运维	高可用	分布式架构	分布式架构	主从构架	主从架构	分布式架构
	跨地域容灾	-	✓	-	-	-
	集群扩容	增加节点，通过复制数据均衡	增加节点，通过新增分片负载均衡	增加节点	增加节点	增加节点
	使用成本	1494 元 / 月，500G SSD，40MB/s	2.00 元 / 百万次调用	公测中	公测中	需要自行部署

1.3 Kafka &Pulsar

- 1) 模型概念

- Kafka: producer – topic – consumer group – consumer
- Pulsar: producer – topic -subsciption- consumer





- 2) 消息消费模式
  - Kafka: 主要集中在流(Stream) 模式, 对单个partition是独占消费, 没有共享(Queue)的消费模式
  - Pulsar: 提供了统一的消息模型和API； 流(Stream) 模式 – 独占和故障切换订阅方式 ; 队列(Queue)模式 – 共享订阅的方式

- 3) 消息确认(ack)
  - Kafka: 使用偏移量 offset
  - Pulsar: 使用专门的cursor管理. 累积确认和kafka效果一样; 提供单条或选择性确认
- 4) 消息保留:
  - Kafka: 根据设置的保留期来删除消息, 有可能消息没被消费, 过期后被删除, 不支持TTL
  - Pulsar: 消息只有被所有订阅消费后才会删除, 不会丢失数据,. 也运行设置保留期, 保留被消费的数据 . 支持TTL

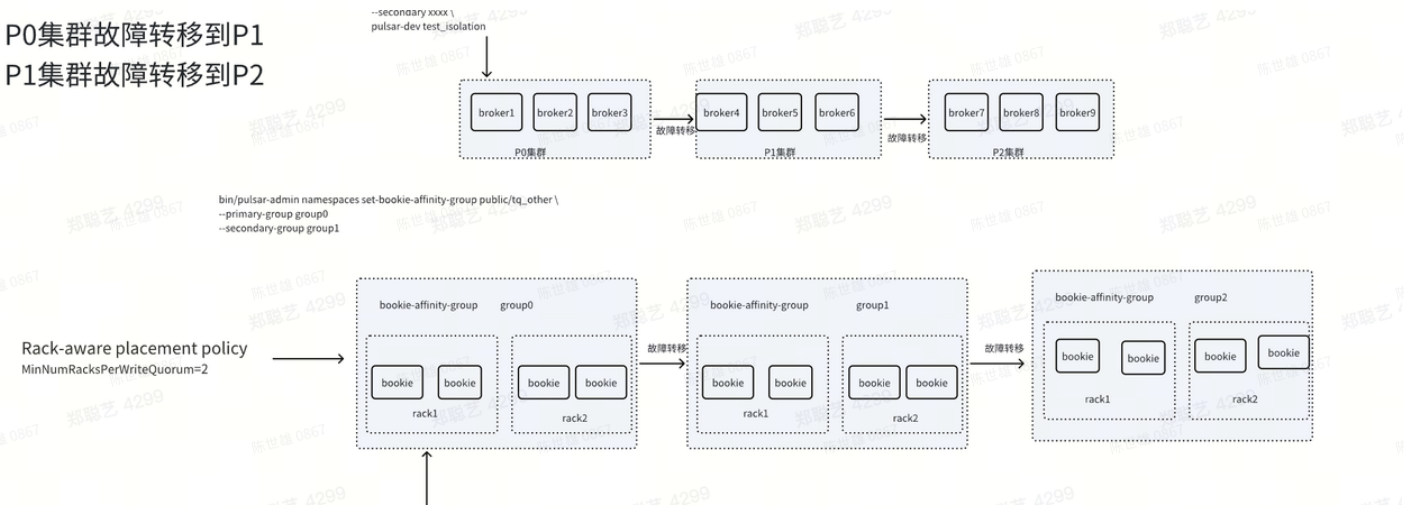
二、目前集群情况

2.1 集群情况

集群	地址	集群情况	说明
测试环境	pulsar://pulsar1.test.hbmonitor.com:6650,pulsar2.test.hbmonitor.com:6650,pulsar3.test.hbmonitor.com:6650		

p0 线上环境	待创建		P0: 要求稳定,以及数据延迟(高峰cpu平均负载不能超40%);
p1 线上环境	pulsar://n1.pulsar1.taqucloud.com:6650,n2.pulsar1.taqucloud.com:6650		P1: 要求稳定,以及数据延迟(高峰cpu负载不能超过50%) [最大不可用时长5分钟内];
p2 线上环境	pulsar://n1.pulsar2.taqucloud.com:6650,n2.pulsar2.taqucloud.com:6650,n3.pulsar2.taqucloud.com:6650,n4.pulsar2.taqucloud.com:6650,n5.pulsar2.taqucloud.com:6650,n6.pulsar2.taqucloud.com:6650,n7.pulsar2.taqucloud.com:6650,n8.pulsar2.taqucloud.com:6650	晚上高峰147万/qps Cpu 40%左右	P2: 稳定性要求不高, (高峰cpu负载不能超过70%) [最大不可用时长20分钟内];

2.2 集群故障转移



2.3 topic监控

https://op-apo.internal.taqu.cn/dashboard/grafana/d/KXZkeC9Iz/pulsar-topicxing-neng?refresh=1m

op-apo.internal.taqu.cn

三、Topics（主题）

和Kafka的发布订阅系统一样，Pulsar 中的 topic 是带有名称的通道，用来从 producer 到 consumer 传输消息。Topic 的名称是符合良好结构的 URL。

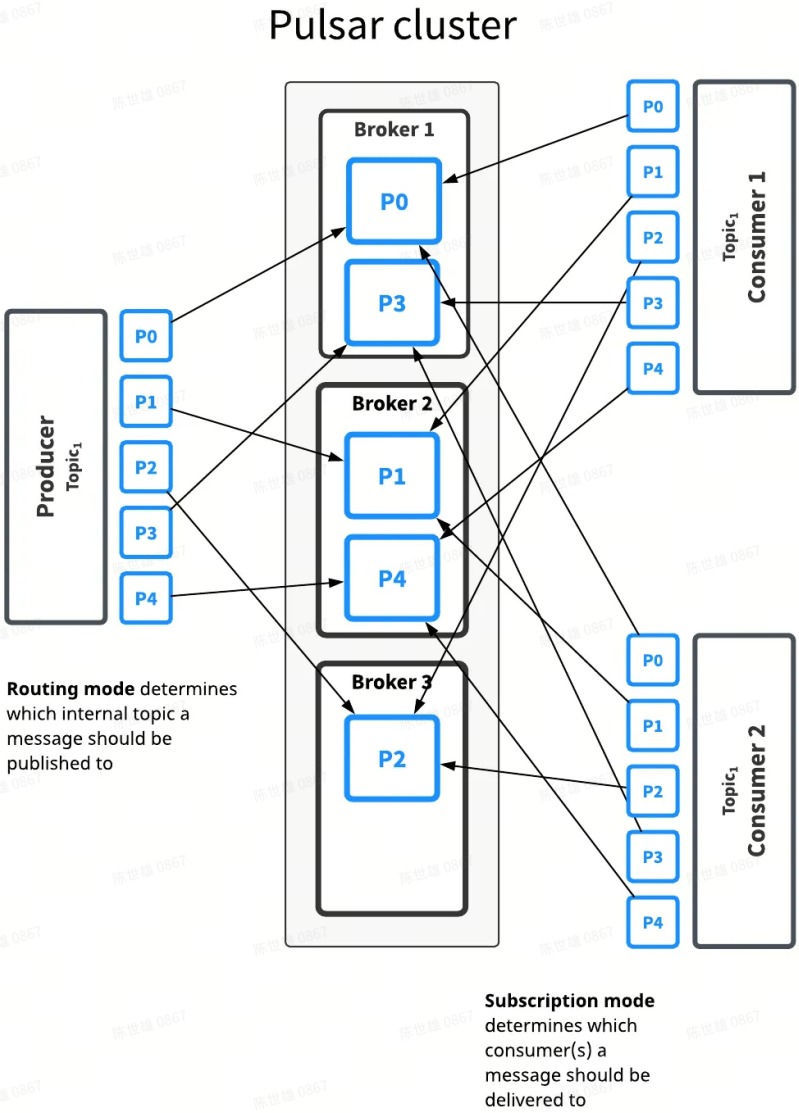
```
{persistent|non-persistent}://tenant/namespace/topic
```

Topic name component	Description
persistent / non-persistent	定义了 topic 类型，Pulsar 支持两种不同 topic：持久和非持久（默认是持久类型，如果你没有指明类型，topic 将会是持久类型）。持久 topic 的所有消息都会保存在硬盘上（这意味着多块硬盘，除非是单机模式的 broker），反之，非持久 topic 的数据不会存储到硬盘上。
tenant	实例中 topic 的租户。tenant 是 Pulsar 多租户的基本要素。可以被跨集群的传播。
namespace	topic 的管理单元，相关 topic 组的管理机制。大多数的 topic 配置在 namespace 层面生效。每个 tenant 可以有多个 namespace。
topic	主题名称的最后组成部分，topic 的名称很自由，没有什么特殊的含义。

3.1 Partitioned topics（分区主题）

普通主题仅由单个 broker 提供服务，这限制了主题的最大吞吐量。分区主题是由多个 broker 处理的一种特殊类型的主题，因此允许更高的吞吐量。

分区的主题实际上实现为 N 个内部主题，其中 N 是分区的数量。当将消息发布到分区主题时，每个消息都被路由到几个 broker 中的一个。分区在 broker 间的分布由 Pulsar 自动处理。



如上图，Topic1 主题有 5 个分区（P0 到 P4），划分在 3 个 broker 上。因为分区比 broker 多，前两个 broker 分别处理两个分区，而第三个 broker 只处理一个分区（同样，Pulsar 自动处理分区的分布）。

此主题的消息将广播给两个消费者。路由模式决定将每个消息发布到哪个分区，而订阅模式决定将哪些消息发送到哪个消费者。

分区的数量可以在创建主题时指定。

3.2 Routing modes（路由模式）

当发布消息到分区 topic，你必须要指定路由模式。路由模式决定了每条消息被发布到的分区（其实是内部主题），Pulsar中的路由模式由MessageRoutingMode枚举类型定义，其选项说明如下：

下面是三种默认可用的路由模式：

Mode	Description
RoundRobinPartition	message 无 key 则轮询，有 key 则 hash(key) 指定分区。（默认模式）
SinglePartition	message 无 key，producer 将会随机选择一个分区，把所有的消息发往该分区。如果 message 指定了 key，分区的 producer 会把 key 做 hash，然后分配消息到指定的分区。
CustomPartition	使用自定义消息路由实现，可以决定特定的消息进入指定的分区。

3.3 Ordering guarantee（顺序保证）

消息的顺序性主要取决于消息的路由模型(MessageRoutingMode)与消息的key。如果消息指定了Key，则无论是RoundRobinPartition还是SinglePartition，相同的key的消息都会发送到相同的分区：

Ordering guarantee	Description	Routing Mode and Key
Per-key-partition（按 key 分区）	具有相同 key 的所有消息将被按顺序放置在同一个分区中。	使用 SinglePartition 或 RoundRobinPartition 模式，Key 由每个消息提供。
Per-producer（按 producer）	来自同一生产者的所有消息将是有序的。	使用 SinglePartition 模式，并且没有为每个消息提供 Key。

3.4 Hashing scheme（哈希方案）

HashingScheme 是一个 enum，表示在选择要为特定消息使用的分区时可用的标准哈希函数集。有两种类型的标准哈希函数可用：JavaStringHash 和 Murmur3_32Hash。生产者的默认哈希函数是 JavaStringHash。

```
Producer<String> producer = pulsarClient.newProducer(Schema.STRING)
    .enableBatching(false)
    .topic(Constants.testTopic)
    .hashingScheme(HashingScheme.JavaStringHash)
    .create()
```

3.5 persistent/Non-persistent topics（持久/非持久主题）

默认情况下， Pulsar 会保存所有没确认的消息到 BookKeeper 中。持久 Topic 的消息在 Broker 重启或者 Consumer 出现问题时保存下来。

除了持久 Topic ， Pulsar 也支持非持久 Topic 。这些 Topic 的消息只存在于内存中，不会存储到磁盘。

因为 Broker 不会对消息进行持久化存储，当 Producer 将消息发送到 Broker 时， Broker 可以立即将 ack 返回给 Producer ，所以非持久 Topic 的消息传递会比持久 Topic 的消息传递更快一些。相对

的，当 Broker 因为一些原因宕机、重启后，非持久 Topic 的消息都会消失，订阅者将无法收到这些消息。

consumer 的默认订阅模式为 Durable。我们可以通过更改 consumer 的配置将订阅模式更改为 NonDurable。

持久订阅代码示例

```
Consumer<byte[]> consumer = pulsarClient.newConsumer()
                .topic("persistent://tenant/namespace/topic")
                .subscriptionName("my-sub")
                .subscriptionMode(SubscriptionMode.Durable)
                .subscribe();
```

非持久订阅代码示例

```
Consumer<byte[]> consumer = pulsarClient.newConsumer()
                .topic("persistent://tenant/namespace/topic")
                .subscriptionName("my-sub")
                .subscriptionMode(SubscriptionMode.NonDurable)
                .subscribe();
```

3.6 Dead letter topic（死信主题）

死信主题允许你在用户无法成功消费某些消息时使用新消息。在这种机制中，无法使用的消息存储在单独的主题中，称为死信主题。你可以决定如何处理死信主题中的消息。

下面的例子展示了如何在 Java 客户端中使用默认的死信主题：

```
Consumer<byte[]> consumer = pulsarClient.newConsumer(Schema.BYTES)
    .topic("persistent://tenant/namespace/topic")
    .subscriptionName("my-sub")
    .subscriptionType(SubscriptionType.Shared)
    .deadLetterPolicy(DeadLetterPolicy.builder()
          .maxRedeliverCount(maxRedeliveryCount)
          .deadLetterTopic("persistent://my-property/my-ns/dead-letter-custom-topic-my-subscription-custom-DLQ")// 死信队列的topic name
          .build())
    .subscribe();
```

默认的死信主题格式：

```
<topicname>-<subscriptionname>-DLQ
```

> 死信主题依赖于消息的重新投递。由于确认超时或否认确认，消息将被重新发送。如果要对消息使用否定确认，请确保在确认超时之前对其进行否定确认。
> 目前，在共享和 Key_Shared 订阅模式下启用了死信主题。

3.7 Retry letter topic（重试主题）

对于许多在线业务系统，由于业务逻辑处理中出现异常，消息会被重复消费。若要配置重新消费失败消息的延迟时间，你可以配置生产者将消息发送到业务主题和重试主题，并在消费者上启用自动重试。当在消费者上启用自动重试时，如果消息没有被消费，则消息将存储在重试主题中，因此消费者在指定的延迟时间后将自动接收来自重试主题的失败消息。

默认情况下，不启用自动重试功能。你可以将 enableRetry 设置为 true，以启用消费者的自动重试。

下面来看个如何使用从重试主题来消费消息的示例：

```
Consumer<byte[]> consumer = pulsarClient.newConsumer(Schema.BYTES)
    .topic(topic)
    .subscriptionName("my-sub")
    .subscriptionType(SubscriptionType.Shared)
    .enableRetry(true)
    .receiverQueueSize(100)
    .deadLetterPolicy(DeadLetterPolicy.builder()
            .maxRedeliverCount(maxRedeliveryCount)
            .retryLetterTopic("persistent://tenant/namespace/topic-Retry")
            .build())
    .subscriptionInitialPosition(SubscriptionInitialPosition.Earliest)
    .subscribe();
```

3.8 Transaction topic（事务）

通过事务API 实现跨topic消息生产和确认的原子性操作，通过这个功能，Producer可以确保一条消息同时发送到多个 Topic，要么这些消息都发送成功，在所有 Topic 上都可以被消费，要么所有消息都不能被消费。这个功能也允许在一个事务操作中对多个 Topic 上的消息进行 ACK 确认，从而实现端到端的Exactly-once 语义。

```
 PulsarClient pulsarClient = PulsarClient.builder()
           .enableTransaction(true).build();

//3. 基于客户端构建生产着对象
producer = pulsarClient.newProducer).topic(txnTopic1)
                  .sendTimeout(1, TimeUnit.SECONDS).create();

 //4. 基于客户端构建消费者对象
Consumer<String> consumer = pulsarClient.newConsumer()
              topic.topic(topic)
              .subscriptionName("my-sub")
              .subscriptionType(SubscriptionType.Shared)
              .subscribe();

 //2. 开启事务的支持
Transaction transaction = pulsarClient.newTransaction()
                                .withTransactionTimeout(1, TimeUnit.SECONDS)
                                .build().get();
//5. 消费数据
Message<String> message = consumer.receive();
consumer.acknowledge(message.getMessageId(), transaction);

//6. 发送数据生产
messageBuilder = producer.newMessage(transaction).value(val).sendAsync();

txn.commit().get();

```

3.9 Deliver topic（延迟队列）

在 Pulsar 中使用延迟消息，可以精确指定延迟投递的时间，有 deliverAfter 和 deliverAt 两种方式:

- 定时消息(deliverAfter)：消息在发送至服务端后，实际业务并不希望消费端马上收到这条消息，而是推迟到某个时间点被消费，这类消息统称为定时消息。

```
producer.newMessage()
    .deliverAfter(long time, TimeUnit unit)
    .send();
```

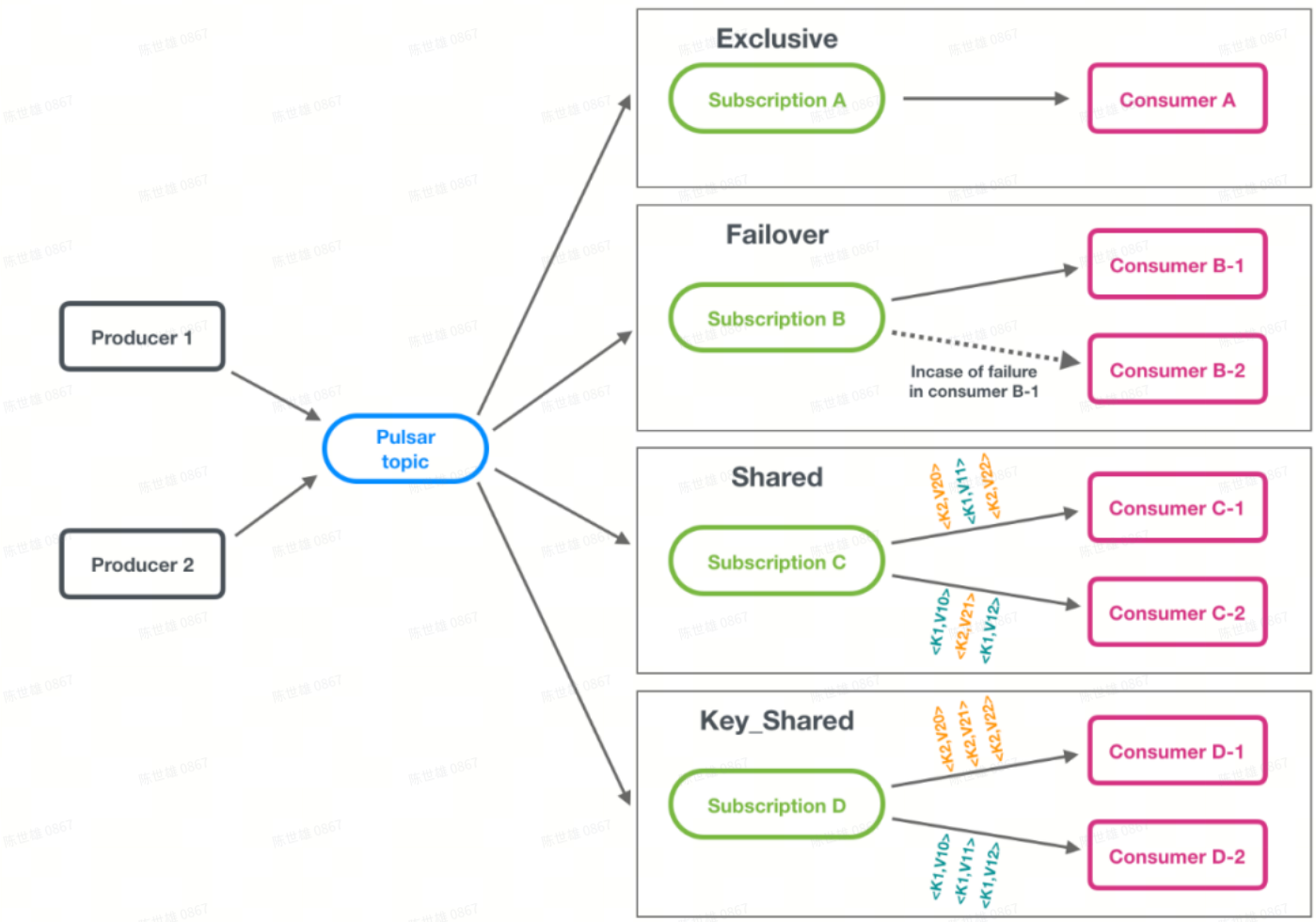
- 延时消息(deliverAt)：消息在发送至服务端后，实际业务并不希望消费端马上收到这条消息，而是推迟一段时间后再被消费，这类消息统称为延时消息

```
producer.newMessage()
    .deliverAt(long timestamp)
    .send();
```

> 定时和延时消息不支持 batch 模式（批量发送），batch 模式会引起消息堆积，保险起见，请在创建 producer 的时候把 enableBatch 参数设为 false。
> 定时和延时消息的消费模式仅支持使用 Shared 模式进行消费，否则会失去定时或延时效果（Key-shared 也不支持）。

四、四种订阅模式

支持多种订阅模式，总共有四种，分别是独占(exclusive)订阅、共享(shared)订阅、故障转移(failover)订阅、键(key_shared)共享



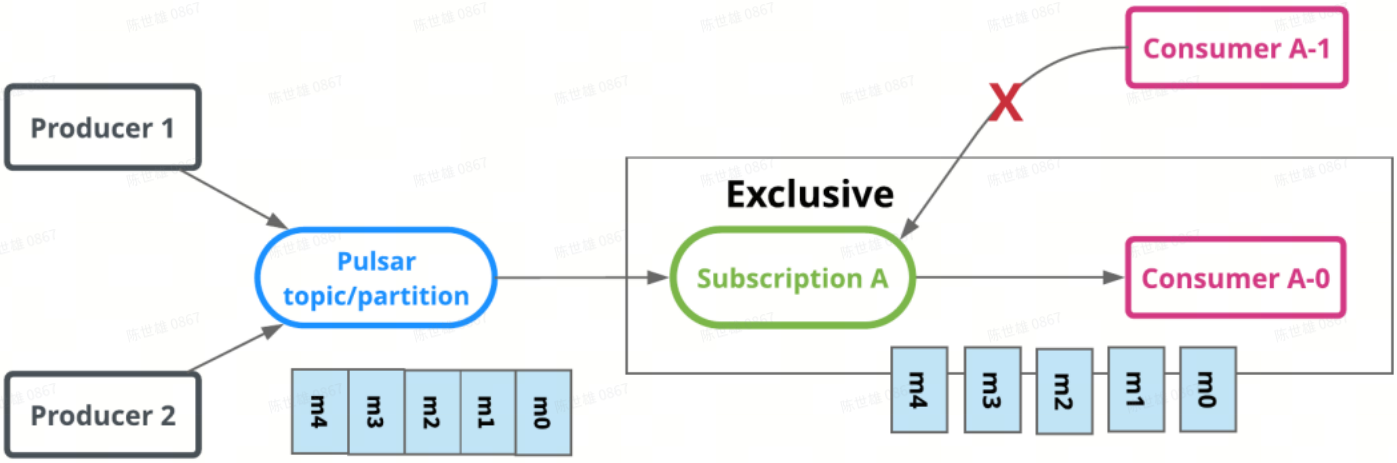
例子：

```
Consumer<byte[]> consumer = client.newConsumer().
```

```
        topic("persistent://tenant/namespace/topic").
        subscriptionName("my-sub")
        .subscriptionInitialPosition(SubscriptionInitialPosition.Latest)
        .subscriptionType(SubscriptionType.Shared)
        .subscribe();
```

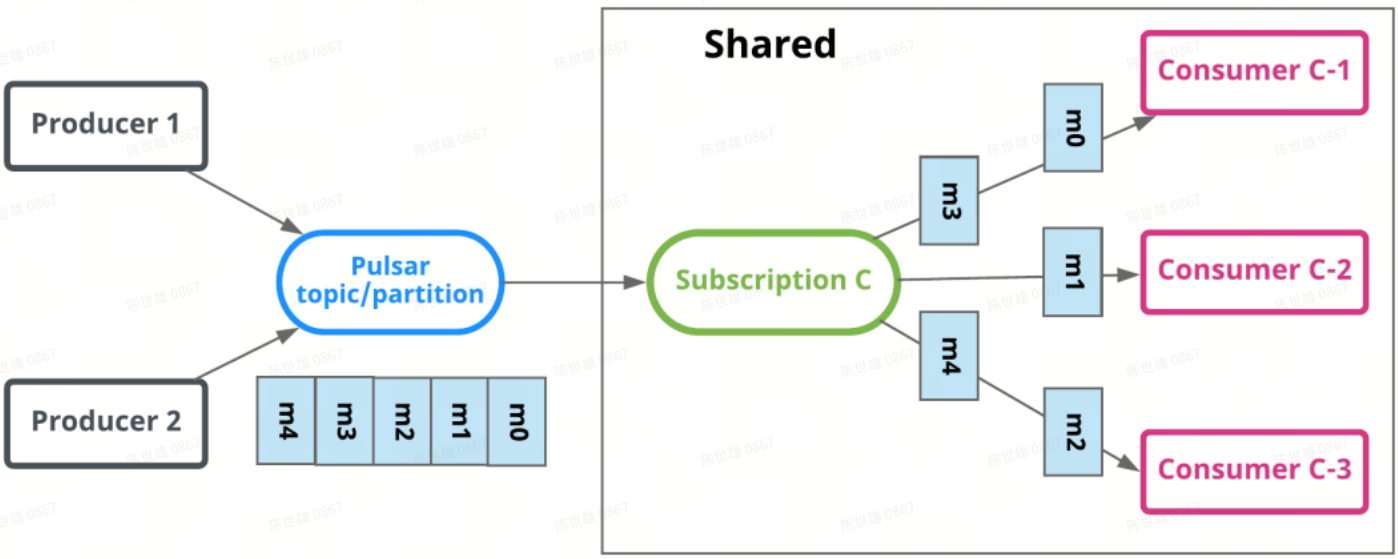
4.1 独占模式（Exclusive）

同时只有一个消费者可以启动并消费数据；通过 `SubscriptionName` 标明是同一个消费者），适用范围较小。



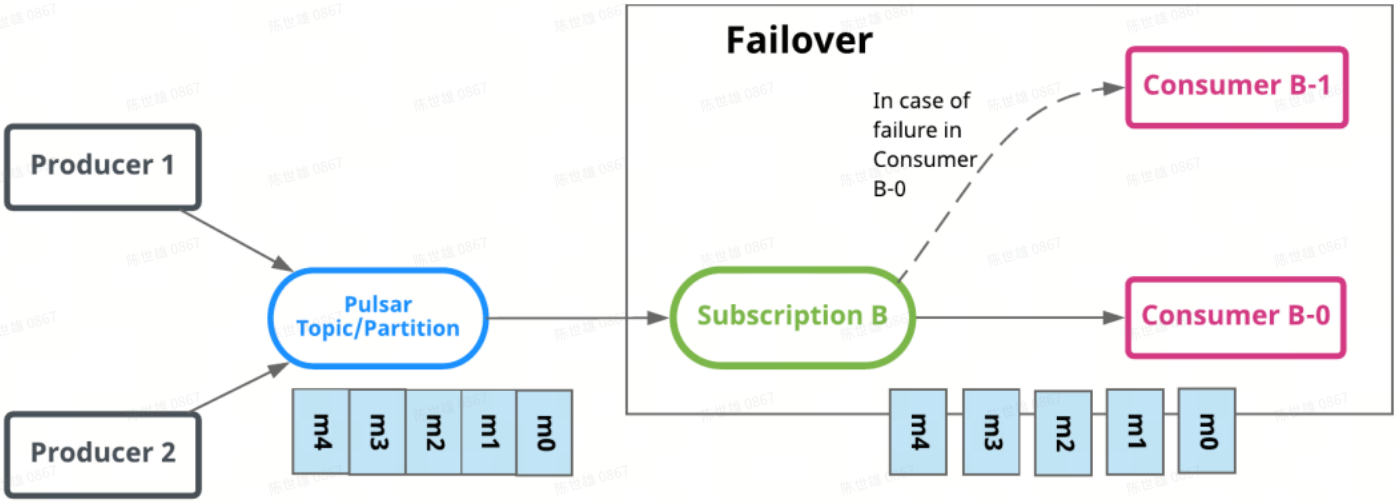
4.2 共享（Shared）

可以有 N 个消费者同时运行，消息按照 round-robin 轮询投递到每个 consumer 中；当某个 consumer 宕机没有 ack 时，该消息将会被投递给其他消费者。这种消费模式可以提高消费能力，但消息无法做到有序。



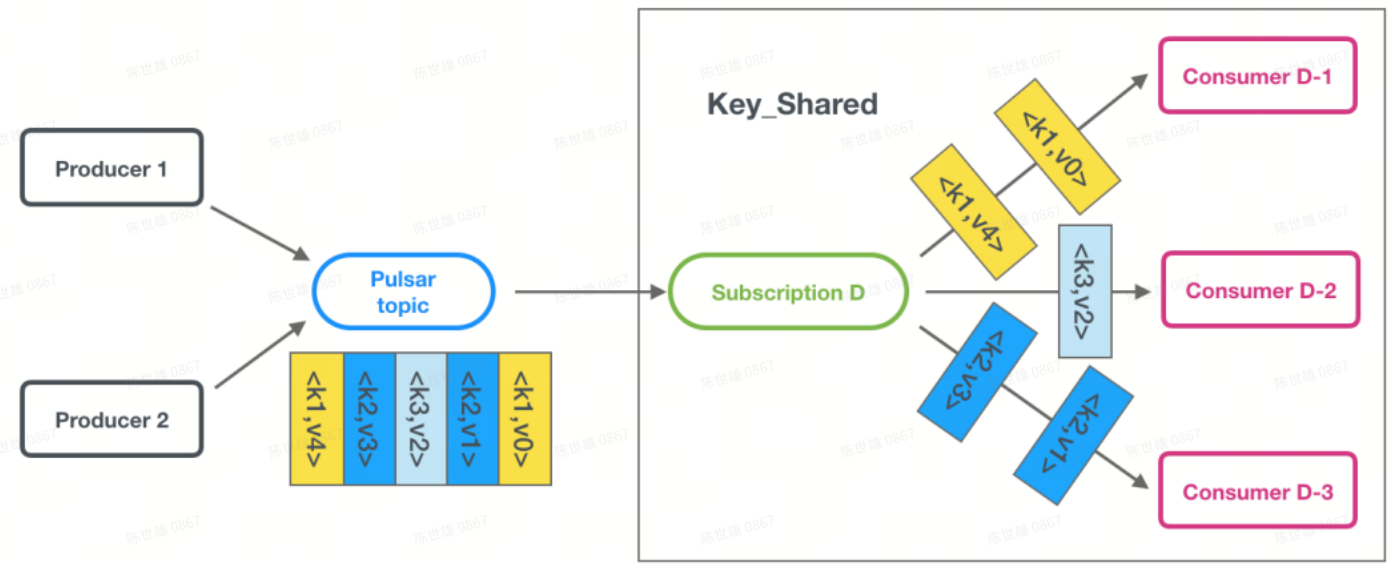
4.3 故障转移（Failover）

故障转移模式在独占模式基础之上可以同时启动多个 consumer，一旦一个 consumer 挂掉之后其余的可以快速顶上，但也只有一个 consumer 可以消费；部分场景可用。



4.4 键共享（KeyShared）

基于共享模式；相当于对同一个topic中的消息进行分组，同一分组内的消息只能被同一个消费者有序消费。



五、核心概念

5.1 租户

Pulsar 的云原生架构支持多租户，每个租户下还支持多 Namespace（命名空间）

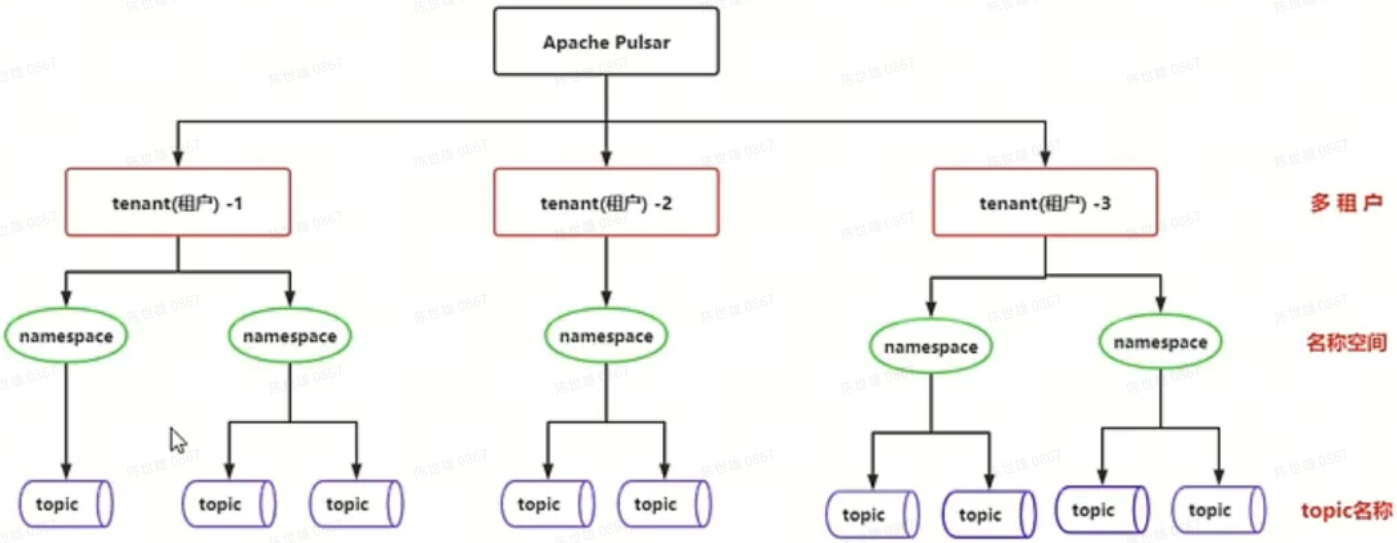
- Tenant（租户）和 Namespace（命名空间）是 Pulsar 支持多租户的两个核心概念。

- 在租户级别，Pulsar 为特定的租户预留合适的存储空间、应用授权和认证机制。
- 在命名空间级别，Pulsar 有一系列的配置策略（Policy）,包括存储配额、流控、消息过期策略和命名空间之间的隔离策略。

Pulsar 的多租户性质主要体现在 Topic ，结构如下：

```
persistent://tenant/namespace/topic
```

租户、命名空间、topic 更直观的关系可以看下图：



5.2 Messages（消息）

Component	Description
Value / data payload	消息携带的数据，所有 Pulsar 的消息携带原始 bytes，但是消息数据也需要遵循数据 schemas。
Key	消息可以被 Key 打标签。这可以对 topic 压缩之类的事情起作用。
Properties	可选的，用户定义属性的 key/value map。
Producer name	生产消息的 producer 的名称（producer 被自动赋予默认名称，但你也可以自己指定。）
Sequence ID	在 topic 中，每个 Pulsar 消息属于一个有序的序列。消息的 sequence ID 是它在序列中的次序。
Publish time	消息发布的时间戳
Event time	可选的时间戳，应用可以附在消息上，代表某个事件发生的时间，例如，消息被处理时。如果没有明确的设置，那么 event time 为0。
TypedMessageBuilder	它用于构造消息。您可以使用TypedMessageBuilder设置消息属性，比如消息键、消息值。设置TypedMessageBuilder时，将键设置为字符串。如果您将键设置为其他类型，例如，AVRO对象，则键将作为字节发送，并且很难从消费者处取回AVRO对象。

5.3 Producers（生产者）

生产者是关联到 topic 的程序，它发布消息到 Pulsar 的 broker 上。

5.3.1 Send modes（发送模式）

producer 可以以同步或者异步的方式发布消息到 broker，pulsar提供以下三种消息发送的接口：

Mode	Description
同步发送（sned）	每发送一条消息时，producer将会等待来自broker的对于这条消息的ack。如果producer没有收到该消息的ack，就会认为这条消息发送失败
异步发送（send async）	生产者将消息放到一个阻塞队列中后，无需等待立即返回。之后client在后台将阻塞队列中的消息发送给broker。如果该阻塞队列已满时，继续向该阻塞队列发送消息会抛出异常（PulsarClientException.ProducerQueueIsFullError）。阻塞队列的最大大小可以进行配置
send timeout	sendTimeout(int sendTimeout, TimeUnit unit)允许在发送的时候设置一个timeout，如果超过这个时间消息还没有被确认，将会抛出一个错误。 注意：如果我们将timeout的时间设置为 0，例如: setTimeout（0，TimeUnit.SECONDS），意味着timeout的时间是无限大。默认时间是：30s

5.3.2 Compression（压缩）

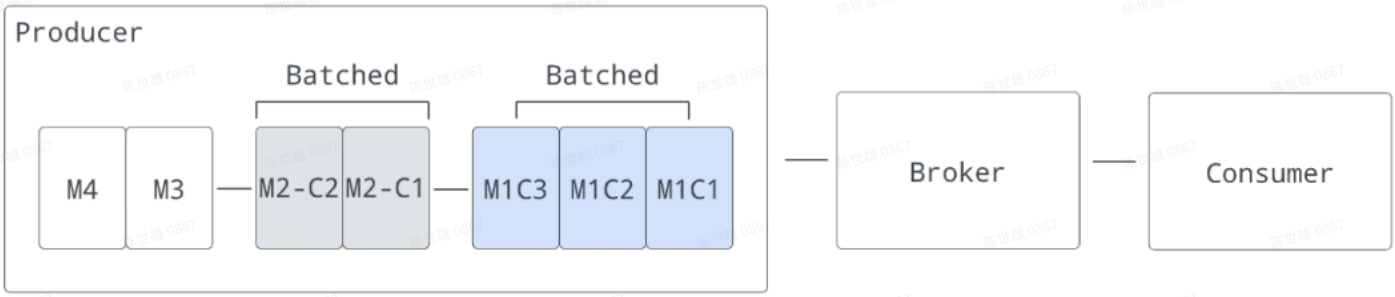
Pulsar目前支持LZ4、ZLIB、ZSTD、SNAPP四种压缩机制，可以通过如下代码指定压缩算法：

```
client.newProducer()
    .topic("persistent://tenant/namespace/topic")
    .compressionType(CompressionType.LZ4)
    .create();
```

5.3.3 Batching（批处理）

Pulsar支持批量消息发送。如果开启批量发送，消息发送者会将多条消息累积到一个批次中进行一次发送。一个批次的消息大小由最大的消息条数+最大的发送延迟两个参数共同决定（参考kafka中的batch.size、linger.ms）,如果开启了批处理，backlogSize表示的一个批次中消息的条数。

批处理示例图如下：



> Pulsar会把一个批次作为一个整体存储到Broker中，消费者接到一个批次后再解绑成一条一条的消息。但即使开启批处理，但调度类消息(设置了deliverAt或者deliverAfter)会单独一条消息进行发送。

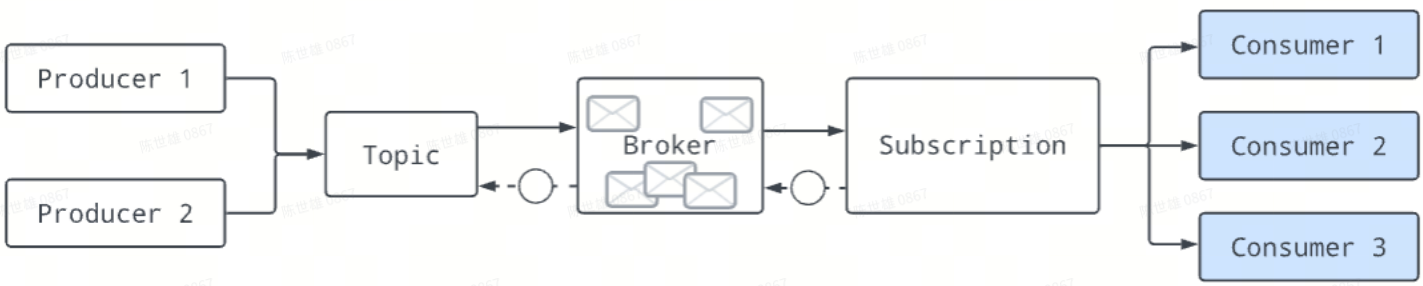
```
Producer<byte[]> BatchProducer = client.newProducer()
                                    .topic("persistent://tenant/namespace/topic")
                                    .batchingMaxMessages(10)
                                    .batchingMaxPublishDelay(10, TimeUnit.MINUTES)
                                    .enableBatching(true).create();

```

5.3.4 Chunking（分块）

- 批处理和分块不能同时启用。要启用分块，必须提前禁用批处理。
- Chunking 只支持持久化的主题。
- Chunking 仅支持 exclusive 和 failover 订阅模式。

5.4 Consumers（消费者）

消费者，通过订阅主题，从而从Broker端接受消息，消息发送，消息消费的核心示意图如下：



5.4.1 Receive modes（接收模式）

消息可以通过同步或者异步的方式从 broker 接收。

Mode	Description
同步接收	同步接收将会阻塞，直到消息可用。
异步接收	异步接收立即返回 future 值，例如 java 中的 CompletableFuture，一旦新消息可用，它即刻完成。

5.4.2 Listeners（监听）

客户端类库提供了它们对于 consumer 的监听实现。当从Broker中收到消息后，将调用消费监听器，从而触发业务代码的执行。举一个 Java 客户端的例子，它提供了 MessageListener 接口。在这个接口中，一旦接受到新的消息，received 方法将被调用。

5.4.3 Acknowledgement（确认）

消费者在成功消费完一条消息后需要告知Broker已成功消费，俗称ACK确认信息。**然后这条消息会被持久化存储，并且在所有订阅组都成功确认后才会删除这条消息。**如果希望Broker继续存储已被所有订阅确认的消息，则需要设置消息的**持久策略**。

如果发送端启用了批处理，则Pulsar可以引入批索引确认机制，避免一个批次的消息重复下发给消费者。

在Pulsar中，确认一条消息有如下两种方式：

- 单条消息独立确认。消费者每一条消息都会发送ACK给Broker，消费端通过调用consumer.acknowledge(msg)对单条消息进行确认。
- 累积确认，消费者只确认接收到的最后一条消息。消费端通过调用consumer.acknowledgeCumulative(msg)进行累积消息确认

> 累积消息确认不能用于 shared 订阅模式，因为 shared 订阅为同一个订阅引入了多个消费者。

六、规范

Pulsar 命名规范