大型电商分布式系统实践 第6周

DATAGURU专业数据分析社区

上节内容回顾

为提升性能，mysql的取舍

1. 主从同步，master与slave之间数据存在延时同步，一致性由强一致性变为最终一致性

2. 分库分表，牺牲了查询的灵活性，必须带上分库分表所依赖的关键属性，牺牲了诸如外键、多表关联查询等RDBMS的传统特性.

3. 系统扩展复杂，数据库库、表路由规则的变更，数据迁移的成本高.

4. 业务拆分后，原先一个库中的表，可能被拆分到多个库中，使得原本简单的事务控制发展为分布式事务.

。。。。。。。。。。

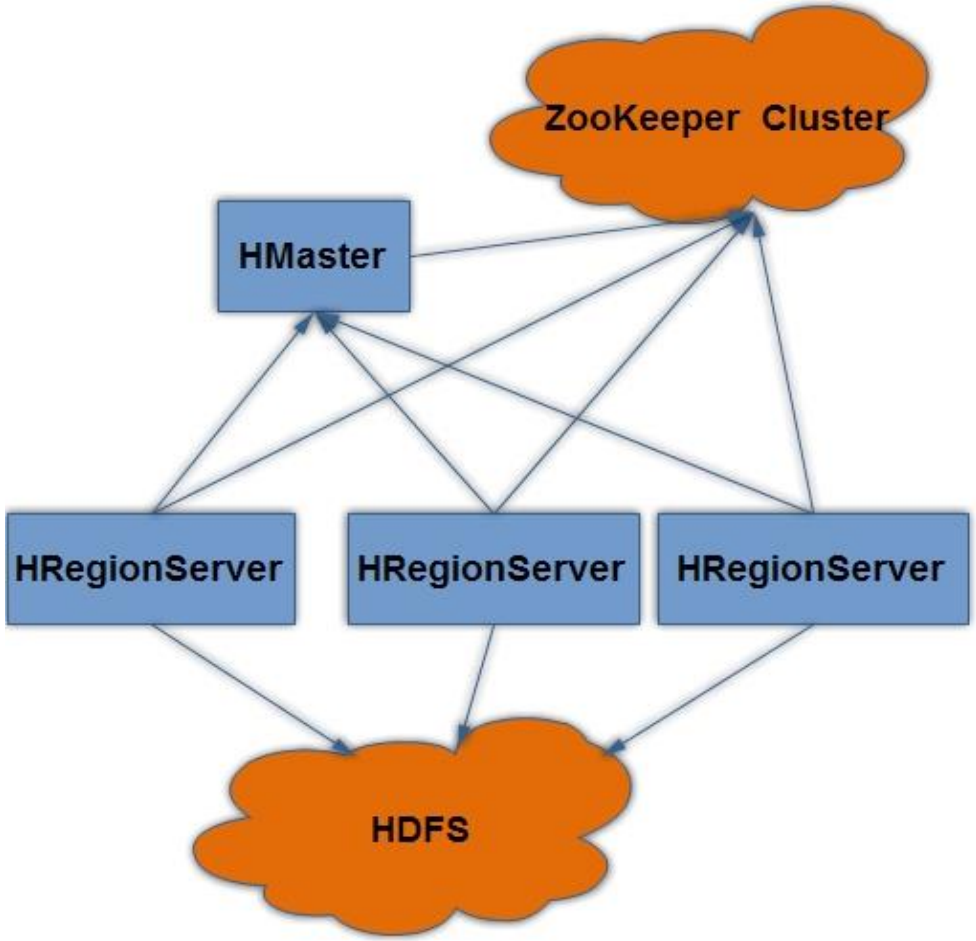
持久化存储--hbase

HBase是Apache Hadoop项目下的一个子项目，它以google BigTable为原型，设计实现了高可靠性、高可扩展性、实时读写的列存储数据库，它的本质实际上是一张稀疏的大表，用来存储粗粒度的结构化数据，并且，能够通过简单的增加节点来实现系统的线性扩展。HBase运行在分布式文件系统HDFS之上，利用它可以在廉价的PC Server上搭建起大规模结构化存储集群。HBase的数据以表的形式来进行组织，每个表由行列组成，与传统的关系型数据库不同的是，HBase每个列属于一个特定的列族，通过行和列来确定一个存储单元，而每个存储单元又可以有多个版本，通过时间戳来标识。

rowkey	column-family1			column-family2		column-family3
	column1	column2	column3	column1	column2	column1
key1	…	…	…	…	…	…
key2	…	…	…	…	…	…
key3	…	…	…	…	…	…

持久化存储—hbase的架构

HBase集群中通常包含两种角色，HMaster和HRegionServer，当表随着记录条数的增加而不断变大后，将会分裂成一个个region，每个region可以由[startkey,endkey)来表示，它包含一个startkey到endkey的半闭区间。一个HRegionServer可以管理多个region，并由HMaster来负责HRegionServer的调度以及集群状态的监管。由于region可分散由不同的HRegionServer来管理，因此理论上再大的表都可以通过集群来处理。



持久化存储—hbase的api使用

持久化存储—rowkey的设计

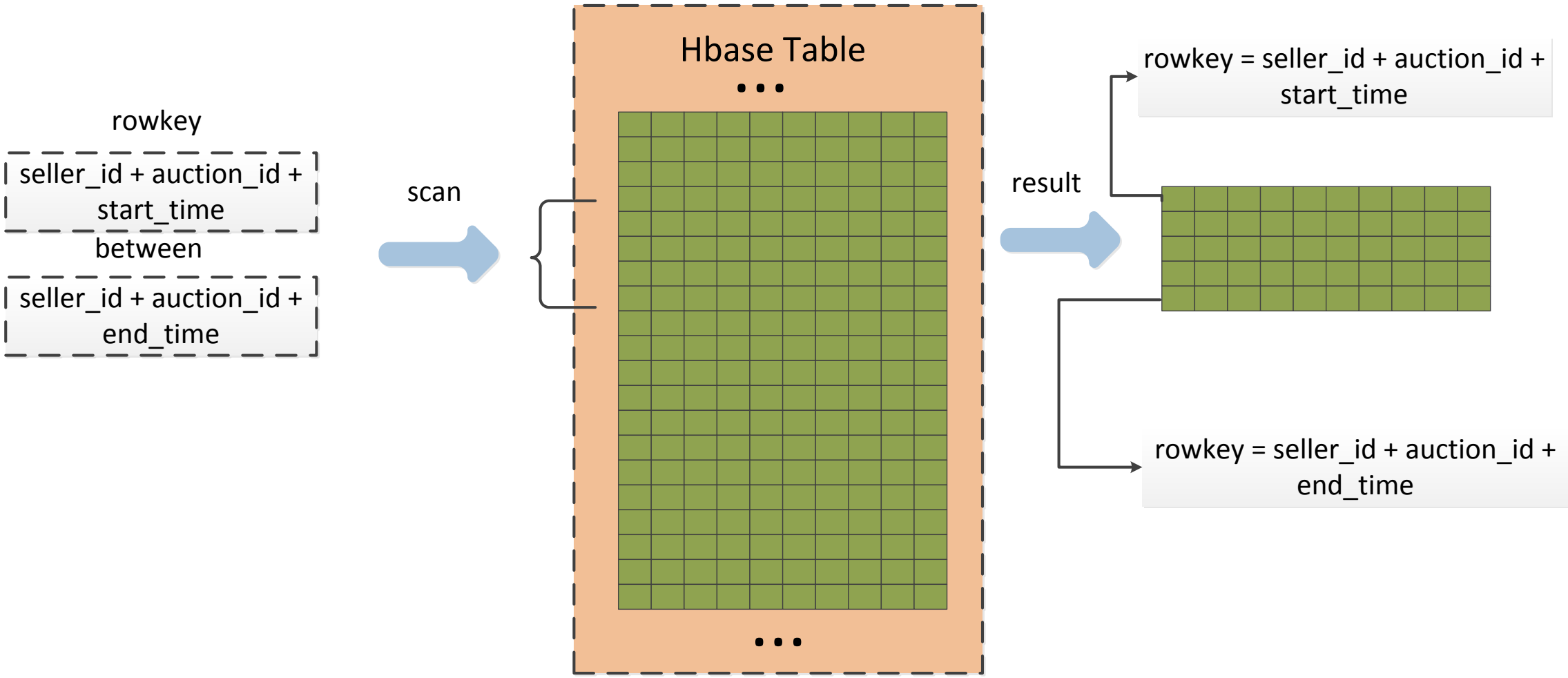
举例来说，假设使用HBase来存储用户的订单信息，我们可能会通过这样几个维度来记录订单的信息，包括购买用户的id、交易时间、商品id、商品名称、交易金额、卖家id等等，假设需要从卖家维度来查看某商品已售出的订单，并且按照下单时间区间来进行查询，那么，订单表可以这样设计：

rowkey：seller_id + auction_id + create_time

列族：order_info(auction_title,price,user_id)

使用卖家id+商品id+交易时间作为表的rowkey，列族为order，该列族包含三列，即商品标题、价格、购买者id，由于HBase的行是按照rowkey来排序的，这样，通过rowkey进行范围查询，可以缩小\scan的范围。

持久化存储—rowkey的设计



持久化存储—rowkey的设计

假设需要从购买者维度来进行订单数据的查询，展现用户购买过的商品，并且按照购买时间进行查询分页，那么，rowkey的设计又不同了：

rowkey：user_id + create_time

列族：order_info(auction_id,auction_title,price,seller_id)

这样，通过买家id+交易时间区间，便能够查询出用户在某个时间范围内购买所产生的订单。

持久化存储—二级索引表

有些时候，我们即需要从卖家维度来查询商品售出情况，又需要从买家维度来查询商品购买情况，类似的多条件复杂查询关系型数据库能够很好的支持，但是对于HBase来说，实现起来并不是那么的容易，基本的解决思路就是建立一张二级索引表，将查询条件设计成二级索引表的rowkey，而存储的数据则是数据表的rowkey，这样，就可以一定程度上的实现多个条件的查询。但是二级索引表也会引入一系列的问题，多表的插入将降低数据写入的性能，并且，由于多表之间无事务保障，可能会带来数据一致性的问题。

持久化存储—hbase的使用场景和限制

与传统的关系型数据库相比，HBase有更好的伸缩能力，更适合于海量数据的存储和处理，并且，由于多个region server的存在，使得HBase能够多个节点同时写入，显著提高了写入性能，并且是可扩展的。但是，HBase本身能够支持的查询维度有限，难以支持复杂查询，如group by、order by、join等等，这些特点使得它的应用场景受到了限制。当然，这也并非是不可弥补的硬伤，通过后面章节所介绍的搜索引擎，构建索引，可以在一定程度上解决HBase复杂条件组合查询的问题。

持久化存储—redis

redis是一个高性能的key-value数据库，与其他很多key-value数据库不同之处在于，redis不仅支持简单的键值对类型的存储，它还支持其他的一系列丰富的数据存储结构，包括strings、hashs、lists、sets、sorted sets等等，并在这些数据结构类型上定义了一套强大的API。通过定义不同的存储结构，redis可以很轻易完成很多其他key-value数据难以完成的任务，如排序、去重等等。

持久化存储—redis的api使用

持久化存储—redis的使用场景

相较于传统的关系型数据库，redis有更好的读写吞吐能力，能够支撑更高的并发数，而相较于其他的key-value类型的数据库，redis能够提供更为丰富的数据类型的支持，能够更灵活的满足业务需求。redis能够高效率的实现诸如排序取topN、访问计数器、队列系统、数据排重等等业务需求，并且，通过将服务器设置为cache-only，还能够提供高性能的缓存服务，相较于memcache来说，在性能差别不大的情况下，它能够支持更为丰富的数据类型。

消息系统

在分布式系统中，消息系统的应用十分广泛，消息可以作为应用间通信的一种方式，消息被保存在队列中，直到被接收者取出，由于消息发送者不需要同步等待消息接收者的响应，消息的异步接收降低了系统集成的耦合度，提升了分布式系统协作的效率，使得系统能够更快的响应用户，提供更高的吞吐，当系统处于峰值压力时，分布式消息队列还能够作为缓冲，削峰填谷，缓解集群的压力，避免整个系统被压垮。

开源的消息系统有很多，包括apache的ActiveMQ，apache的Kafka，RabbitMQ，memcacheQ等等。。。

消息系统--JMS

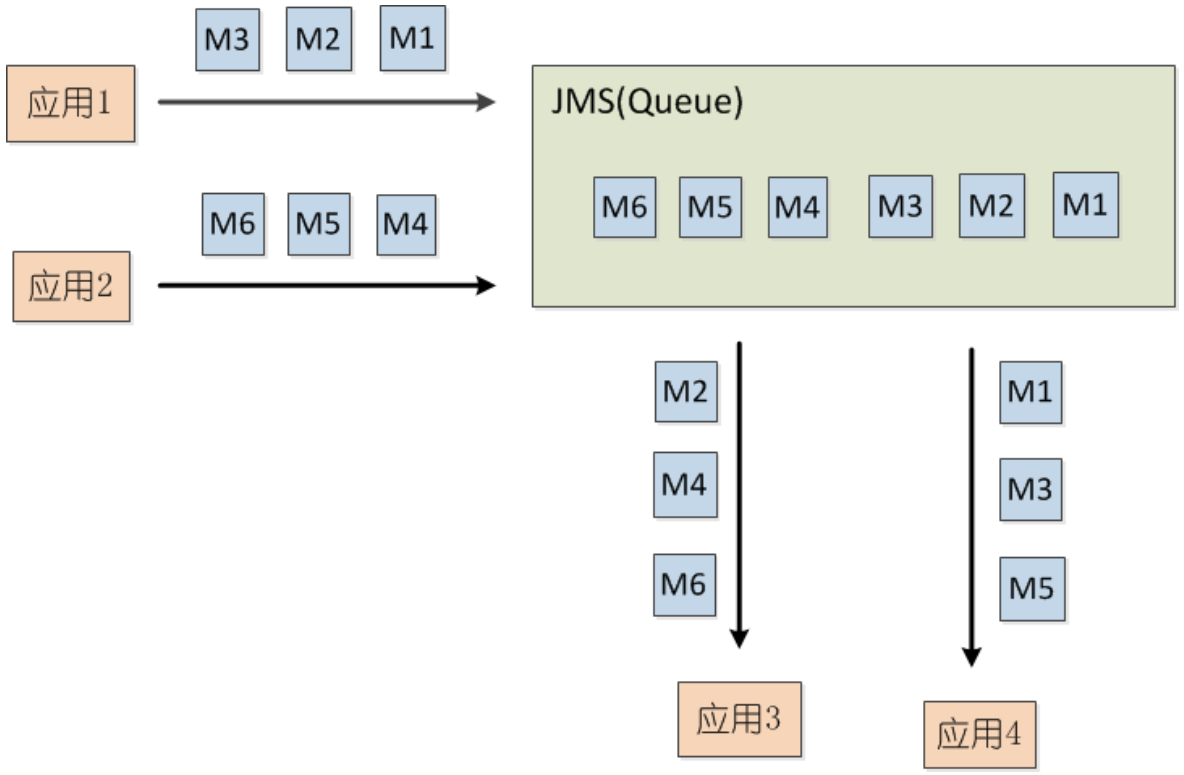
JMS(Java Message Service，即Java消息服务)是J2EE提出消息服务规范，它是一组java应用程序接口，它提供消息的创建、消息的发送、消息接收、消息读取等等一系列服务。JMS定义了一组公共应用程序接口和相应的语法，类似于java数据库的统一访问接口JDBC，它是一种与厂商无关的API，使得java程序能够很好的与不同厂商的消息组件进行通信。

JMS支持的消息类型包括：简单文本(TextMessage)、可序列化的对象(ObjectMessage)、键值对(MapMessage)、字节流(BytesMessage)、流(StreamMessage)，以及无有效负载的消息 (Message)等等。消息的发送是异步的，因此，消息的发布者发送完消息之后，不需要等待消息接收者立即响应，这样便提高了分布式系统协作的效率。

消息系统—JMS消息模型

JMS支持两种消息发送和接收模型，一种称为Point-to-Point(P2P)模型，即采用点对点的方式发送消息，P2P模型是基于queue(队列)的，消息生产者发送消息到队列，消息消费者从队列中接收消息，队列的存在，使得消息的异步传输称为可能，P2P模型在点对点的情况下进行消息传递时采用。另一种为pub/sub(Publish/Subscribe，即发布/订阅)模型，发布/订阅模型定义了如何向一个内容节点发布和订阅消息，这个内容节点称为topic(主题)，主题可以认为是消息传递的中介，消息发布者将消息发布到某个主题，而消息订阅者则从主题订阅消息，主题使得消息的订阅者与消息的发布者互相保持独立，不需要进行接触即可保证消息的传递，发布/订阅模型在消息的一对多广播时采用。

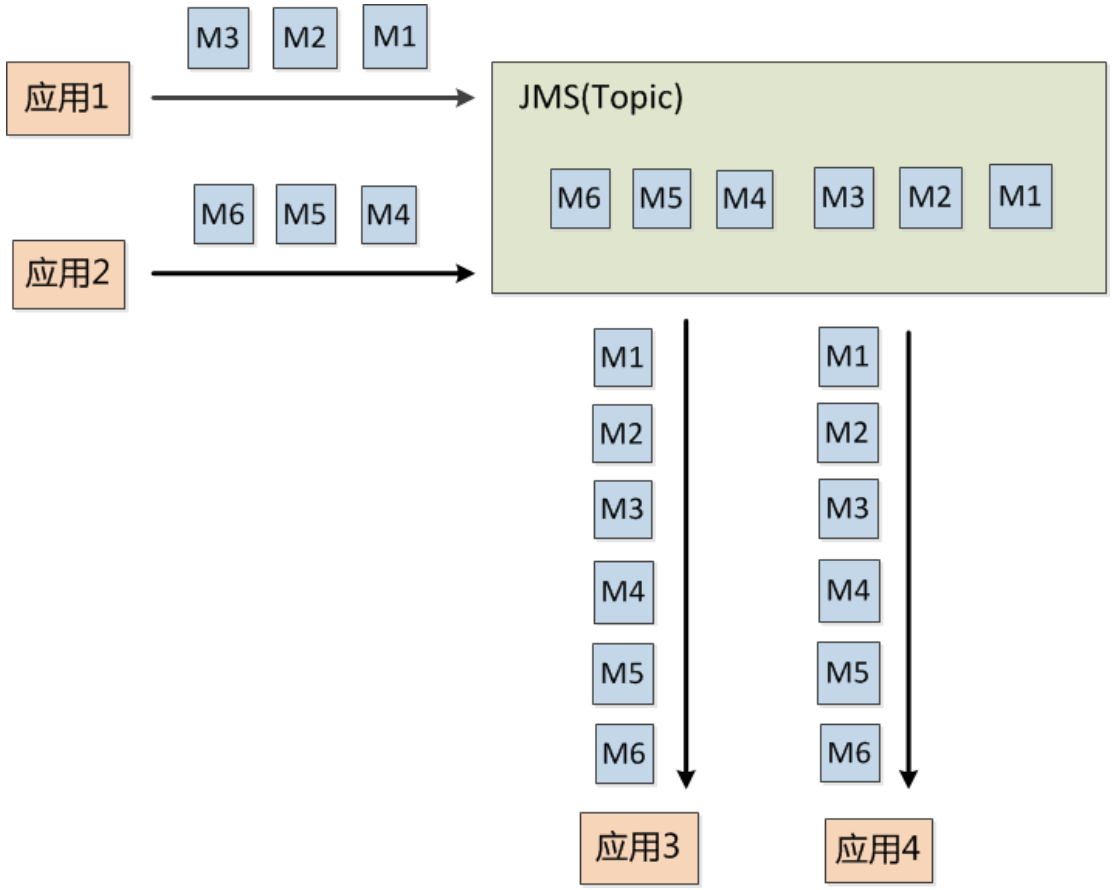
消息系统—点对点模型



Queue模型

多个消息的生产者和消息的消费者都可以注册到同一个消息队列，当消息的生产者发送一条消息之后，只有其中一个消息消费者会接收到消息生产者所发送的消息，而不是所有的消息消费者都会收到该消息。

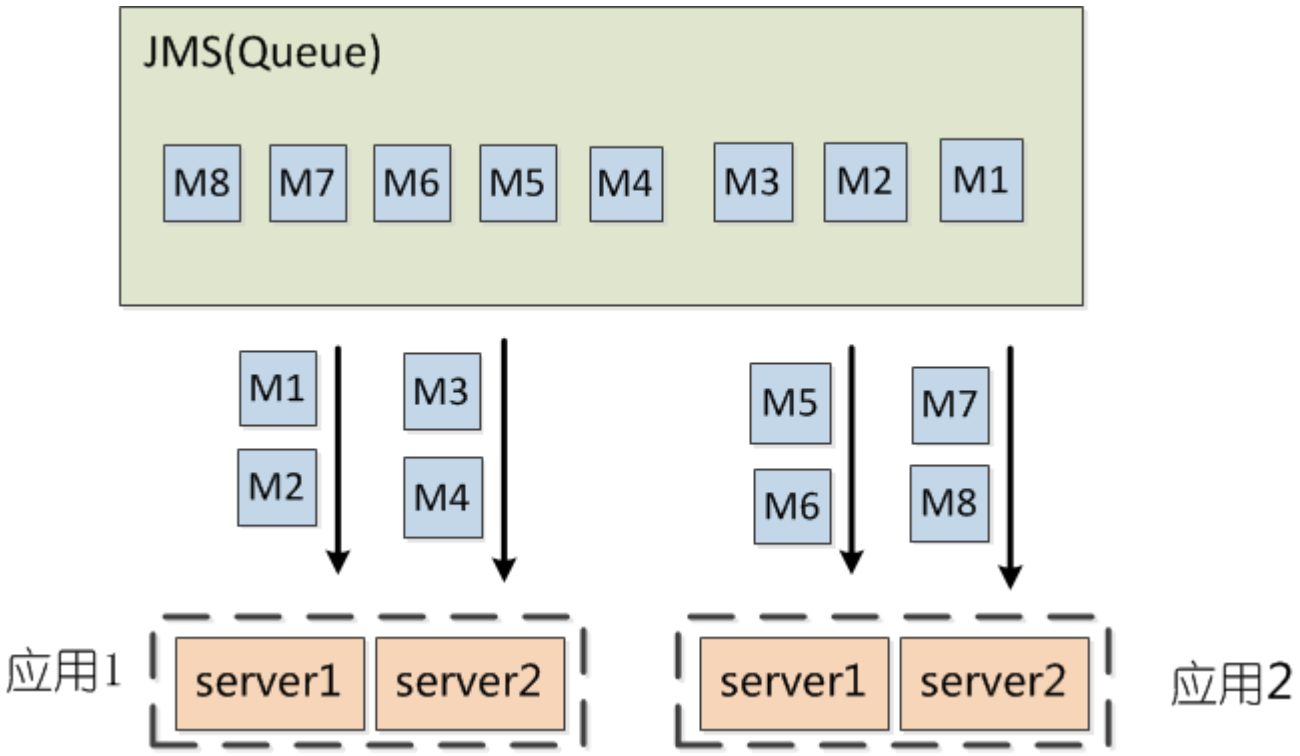
消息系统—发布/订阅模型



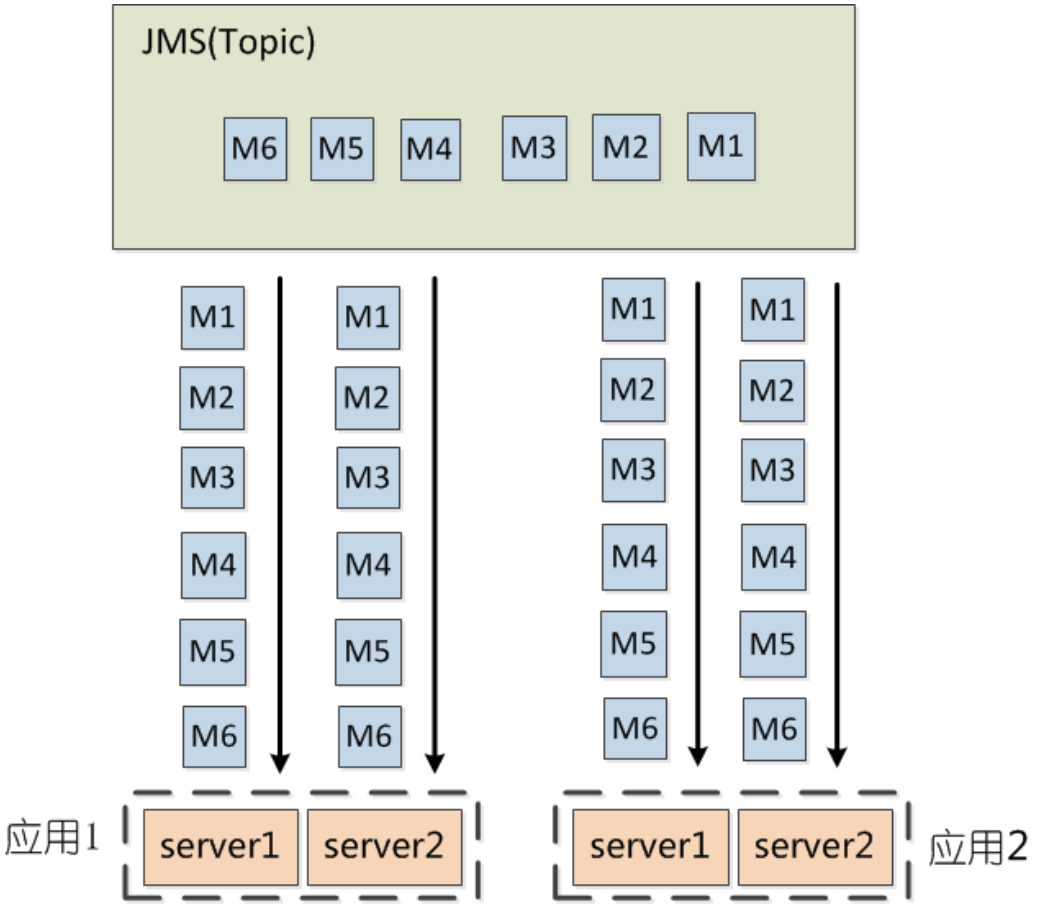
Topic模型

对于发布/订阅的消息传输模型来说，消息的发布者需将消息投递给topic，而消息的订阅者则需要在相应的topic进行注册，以便接收相应topic的消息，与点对点消息传输模型不同的是，消息发布者的消息将被自动发送给所有订阅了该topic的消息订阅者。当消息订阅者某段时间由于某种原因断开了与消息发布者的连接时，这个时间段内的消息将会丢失，除非将消息的订阅模式设置为持久订阅(durable subscription)，这时，消息的发布者将会为消息的订阅者保留这部分时间所产生的消息，当消息的订阅者重新连接消息发布者时，消息订阅者仍然可以获得这部分消息，而不至于这部分消息丢失。

消息系统—JMS模型的限制

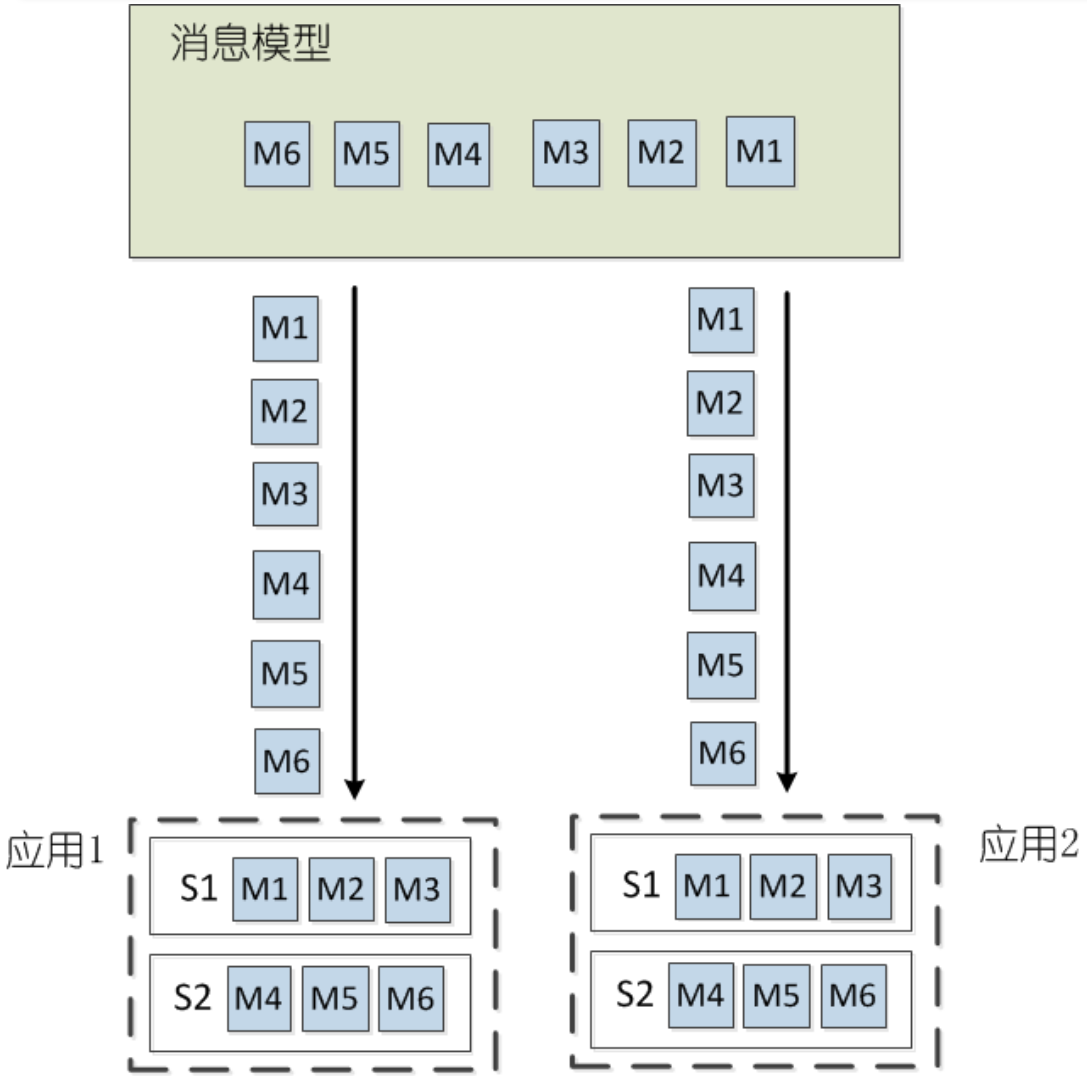


应用扩展后的Queue模型



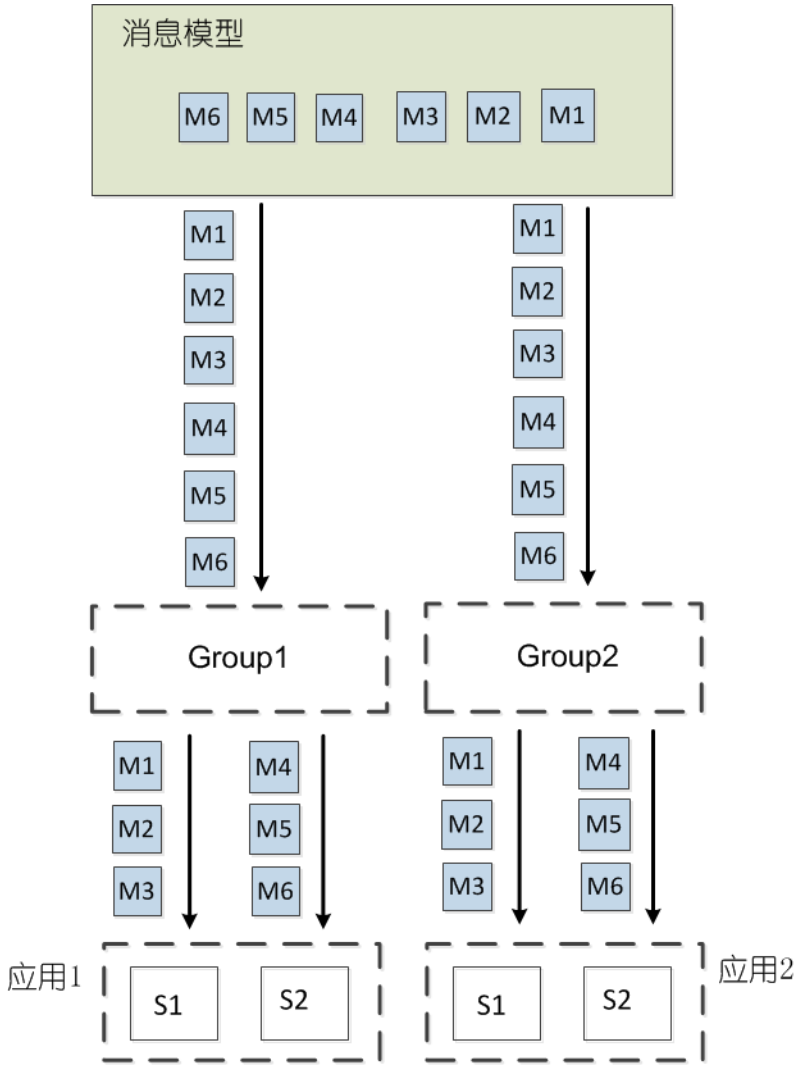
应用扩展后的Topic模型

消息系统—我们需要的消息模型



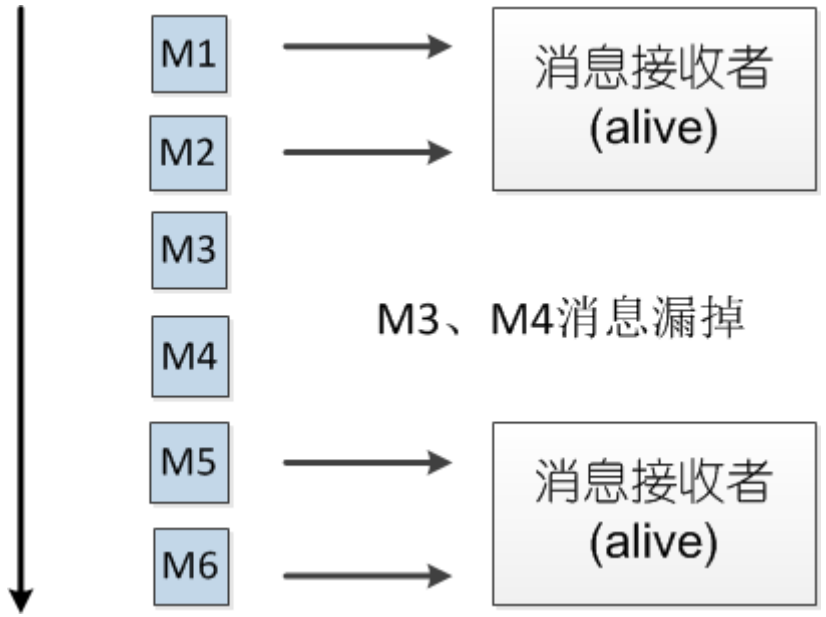
大型分布式系统对于消息系统的需求：
1. 消息发送和消息接收都是集群
2. 同一个消息的接收方可能有多台机器甚至是多个集群来进行消息的处理
3. 不同集群对于同一条消息的处理都不能相互干扰

消息系统—我们需要的消息模型

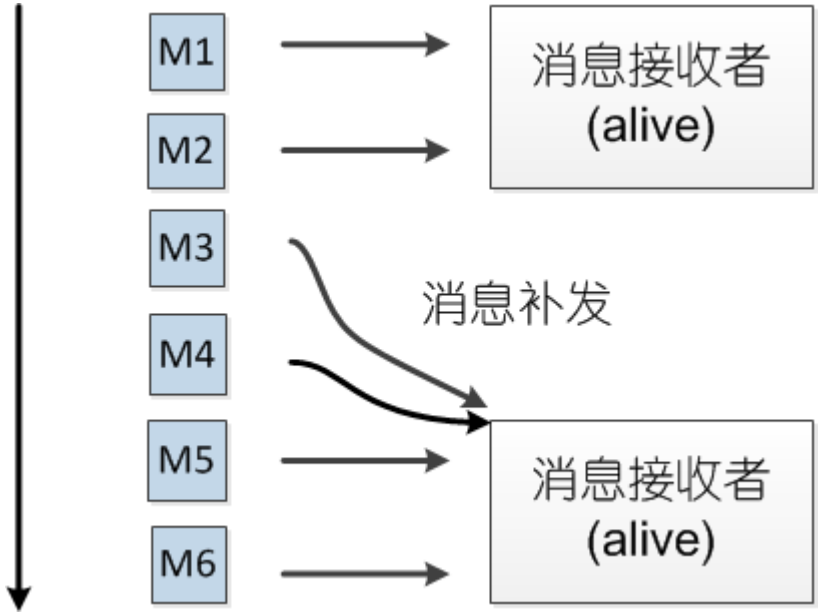


具体来说，我们可以把集群和集群之间对消息的消费当做topic模型来处理，而集群内部的各个具体应用实例对消息的消费当做queue模型来处理，我们可以引入groupid，用这个id来标识不同的集群，而集群内的各个应用实例的连接使用同样的groupid ，当服务器进行调度时，根据groupid进行连接分组，在不同的groupid之间保障消息的独立投送，而拥有同样groupid的连接则共同消费这些信息。这个策略分两级来进行处理，把topic模型和queue模型的特点结合起来使用，从而达到多个不同的集群进行消息订阅的目的。当然，抛弃JMS意味着我们可能需要自己实现连接的管理、消息创建、消息发送、消息接收、消息读取等一系列接口。

消息系统—持久订阅和非持久订阅



非持久订阅



持久订阅

消息系统—如何保障消息的一致性

业务处理代码：

```
function(){
//业务操作
//调用服务，将数据写入数据库
//发送消息
}
```

1. 业务操作在前，发送消息在后，如果业务失败还行，如果业务成功，此时系统宕机，消息则发送失败
2. 如果业务成功，应用也正常，此时消息系统宕机，消息没收到，也会导致消息收不到

消息系统—如何保障消息的一致性

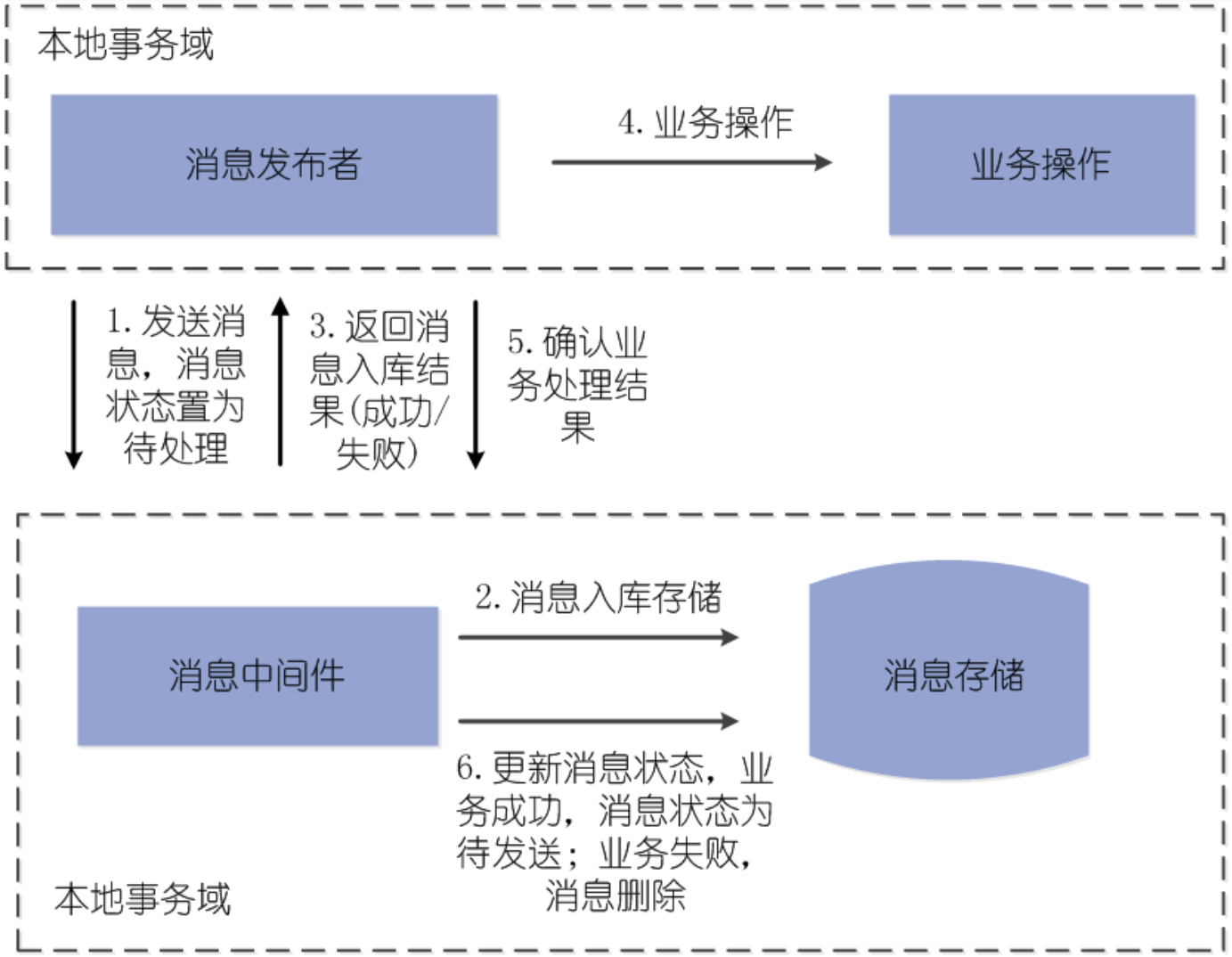
另外一种业务代码的写法：

```
function(){
//发送消息
//业务操作
//调用服务或者写数据库
}
```

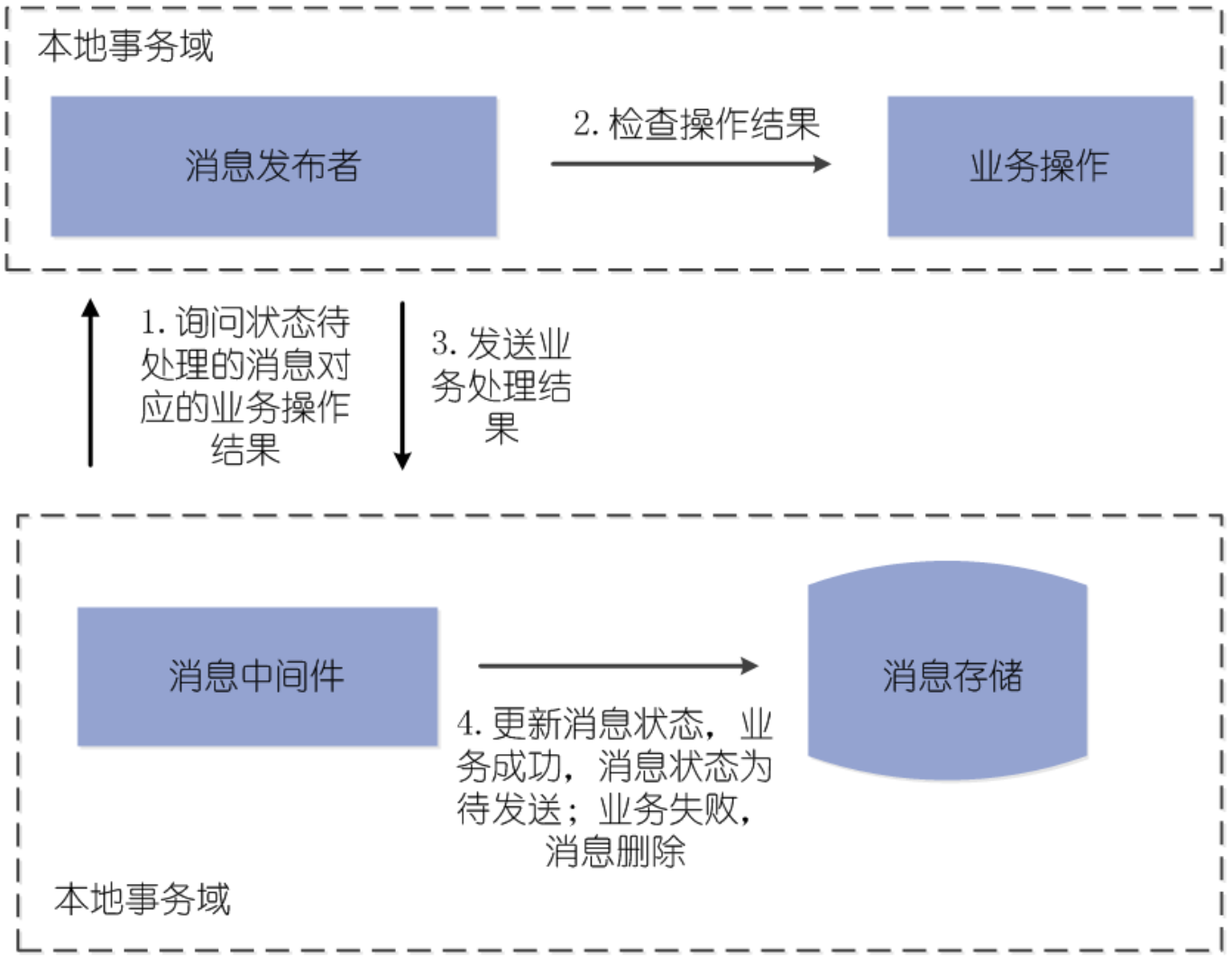
这种方式更不靠谱，业务如果失败，消息却已经发出。

真正的生产环境，第一种做法丢失消息的比例相对来说是很低的，但是，对于必须保证一致性的场景下，比如跟交易相关的业务操作，这两种方案都不能接受。

消息系统—如何保障消息的一致性



消息系统—如何保障消息的一致性



Thanks

FAQ时间