链滴

棘皮动物的演化史

作者：VoidCup

原文链接：https://ld246.com/article/1564478884360

来源网站：链滴

问题

- 消息中间件产生的背景？
  - 为什么需要消息中间件, 有什么作用？
  - 应用于什么场景？
- HTTP协议同步接口调用失败了怎么做？
- 消息队列异步通讯与同步通讯区别
  - 什么是异步通讯, 什么是同步通讯？
  - 两者有何区别？
- JMS消息通讯模型有哪些？
  - JMS有什么通讯模型？
  - 如何保证JMS可靠消息？
- ActiveMQ服务端宕机了, 消息会丢失吗？
  - 消息会丢失吗？

    - 参考
      - 拉钩程序员Alex --同步架构和异步架构的区别及消息队列
      - Mr_Smile2014 --消息队列(Message Queue)产生的背景, 常用的技术
      - TourJiang --消息中间件及ActiveMQ介绍
      - QING --JMS消息传送机制
      - itcast_cn --消息中间件之ActiveMQ原理及使用教程

- weixin_34133829 --JMS 消息的可靠性机制
    - 阮一峰的网络日志 --RESTful API 最佳实践

消息中间件

消息中间件产生的背景

1.1面向对象的中间件

1.1.1 什么是面向对象的中间件

为了解决**数据集成和系统整合**等问题, CORBA(Common Object Request Broker Architecture,公共象请求代理体系结构), DCOM(分布式组件对象模型,分布式组件对象模式), RMI(Remote ethod Invocation,远程方法调用) 等RPC (Remote Procedure Call 远程过程调用)中间件技术也应运生,在性能、健壮性、可扩展性上都存在着诸多缺点。分布式对象调用, 如CORBA, RMI 和 DCOM, 提了一种通讯机制, 透明地在异构的分布式计算环境中传递对象请求, 这些对象可以位于本地或远程机器. 通过在对象与对象之间提供一种统一的接口, 使对象之间的调用和数据共享不在关心对象的位置, 实现言及所驻留的操作系统, 这个接口就是面向对象的中间件

- dubbo

1.1.2 面向对象的中间件的局限性

- 同步通信: 客户发出调用后，必须等待服务对象完成处理并返回结果后才能继续执行
- 客户和服务对象的生命周期紧密耦合:客户进程和服务对象进程都必须正常运行，如果由于服务对象溃或网络故障导致客户的请求不可达，客户会接收到异常。用户依赖于异构的其他接口服务

而基于消息的异步处理模型采用非阻塞的调用特性，发送者将消息发送给消息服务器，消息服务器在适的时候再将消息转发给接收者；发送和接收是异步的，发送者无需等待，二者的生命周期也可以不相同，而且发送者可以将消息间接传给多个接收者，大大提高了程序的性能、可扩展性及健壮性，这得异步处理模型在分布应用上比起同步处理模型更具有吸引力

1.2 消息中间件

1.2.1 消息中间件的作用

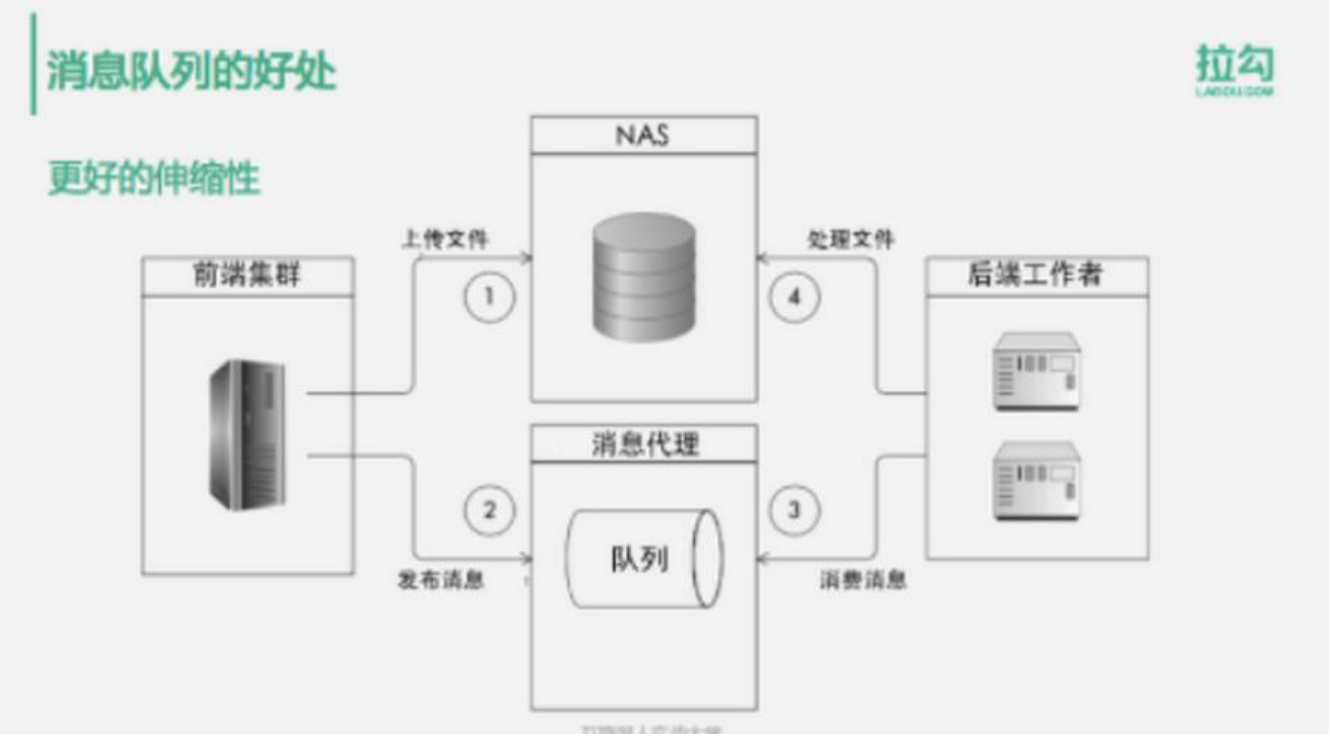
1.2.1.1 异步处理

对一些比较耗时的操作，可以把处理过程通过消息队列进行异步处理。这样做可以推迟耗时操作的处，使耗时操作异步化，而不必阻塞客户端的程序，客户端的程序在得到处理结果之前就可以继续执行从而提高客户端程序的处理性能。

1.2.1.2 易伸缩

耗时的任务可以通过分布式消息队列，向多台消费者服务器并行发送消息，然后在很多台消费者服务上并行处理消息，也就是说可以在多台物理服务器上运行消费者。那么当负载上升的时候，可以很容

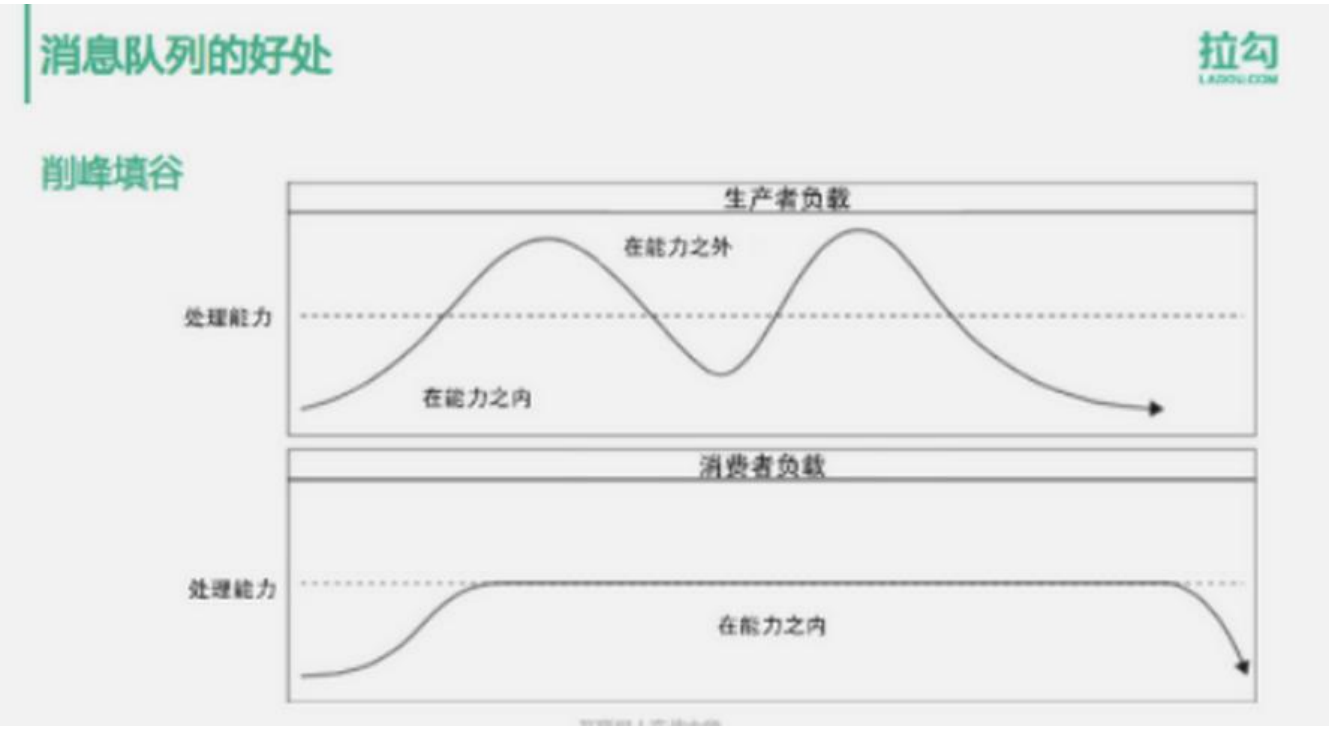
地添加更多的机器成为消费者。



- 消费者服务能更方便的集群

1.2.1.3 平衡流量峰值, 削峰填谷

使用消息队列，即便是访问流量持续的增长，系统依然可以持续地接收请求。这种情况下，虽然生产发布消息的速度比消费者消费消息的速度快，但是可以持续的将消息纳入到消息队列中，用消息队列为消息的缓冲，因此短时间内，发布者不会受到消费处理能力的影响。



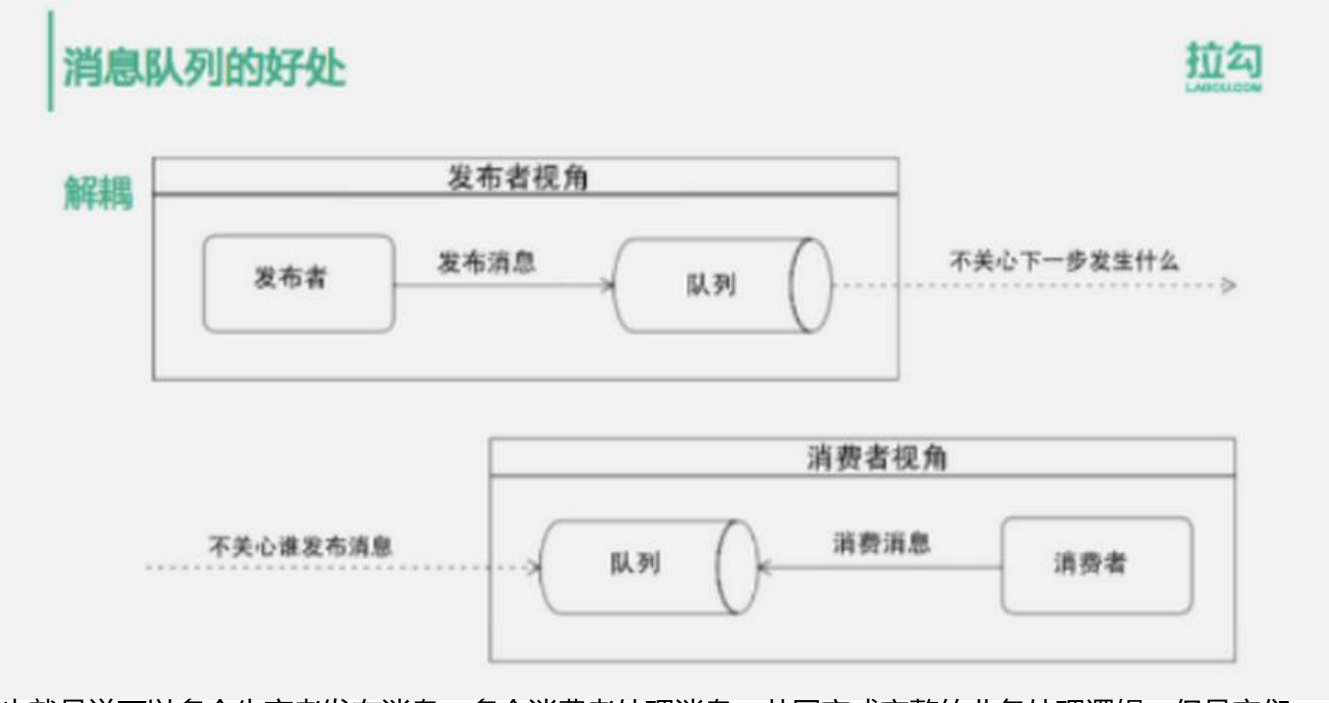
- 生产高峰时期通过消息队列最为缓存也能使得消费者在一个最佳的处理状态处理消息.

1.2.1.4 失败隔离和自我修复

因为发布者不直接依赖消费者，所以分布式消息队列可以将消费者系统产生的错误异常与生产者系统离开来，生产者不受消费者失败的影响。 当在消息消费过程中出现处理逻辑失败的时候，这个错误只影响到消费者自身，而不会传递给消息的生产者，也就是应用程序可以按照原来的处理逻辑继续执行.

所以，这也就意味着在任何时候都可以对后端的服务器执行维护和发布操作。可以重启、添加或删除务器，而不影响生产者的可用性，这样简化了部署和服务器管理的难度。

1.2.1.5 可以使生产者和消费者之间解耦合



也就是说可以多个生产者发布消息，多个消费者处理消息，共同完成完整的业务处理逻辑，但是它们不需要直接的交互调用，没有代码的依赖耦合。在传统的同步调用中，调用者代码必须要依赖被调用的代码，也就是生产者代码必须要依赖消费者的处理逻辑代码，代码需要直接的耦合，而使用消息队，这两部分的代码不需要进行任何的耦合。

- 生产者不依赖消费者, 消费者也不依赖生产者

1.2.2 应用场景

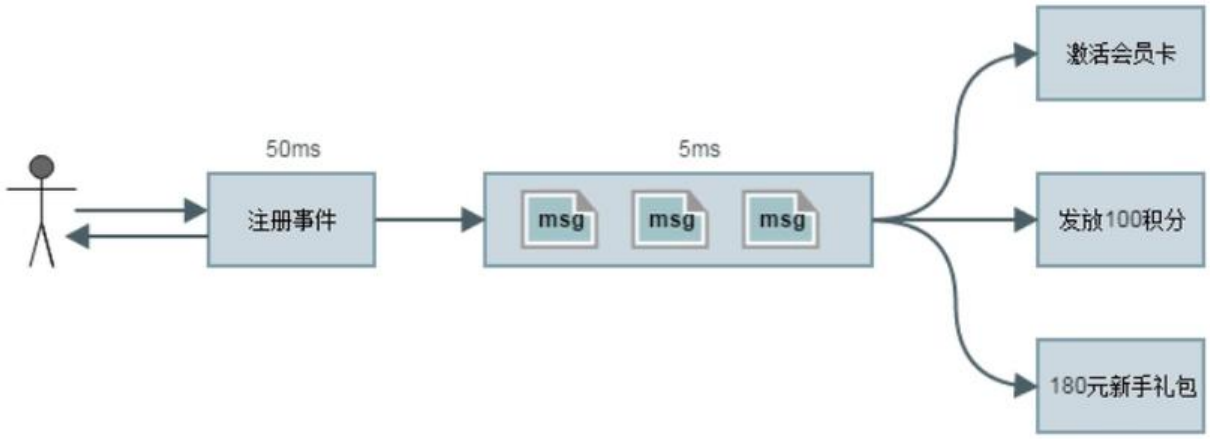
1.2.2.1 异步应用

场景说明:新用户注册发放100积分,180元新手大礼包,激活会员卡，传统的做法有两种:串行方式,并行式。

- 串行方式

原文链接：棘皮动物的演化史

● 使用消息队列

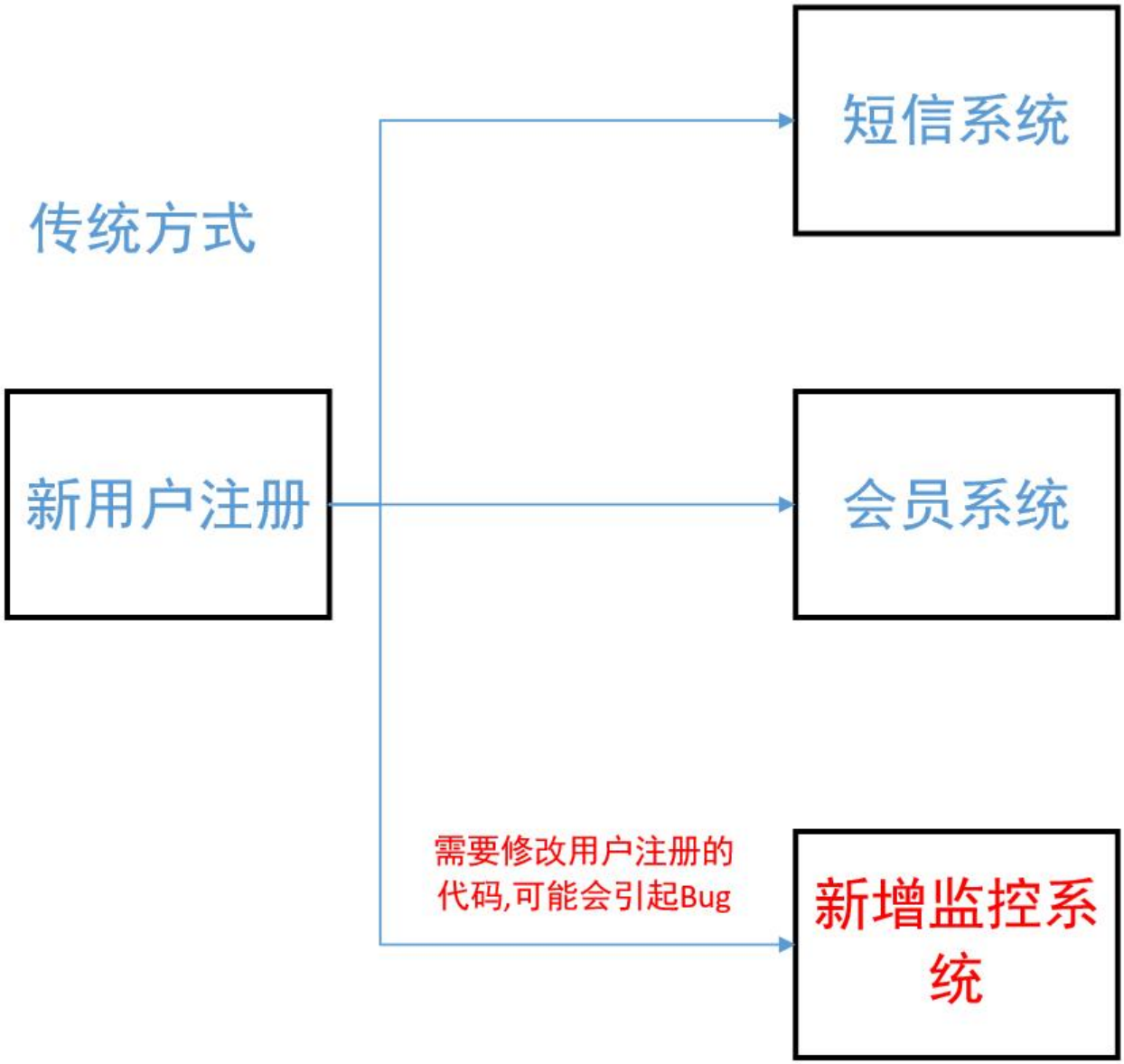


● 同步处理情况下都会涉及到非主业务的其他操作, 而注册的主流程不应该受其他事件影响; 通过息队列的方式, 可以把后续的处理流程进行异步处理可以大大提高响应速度;

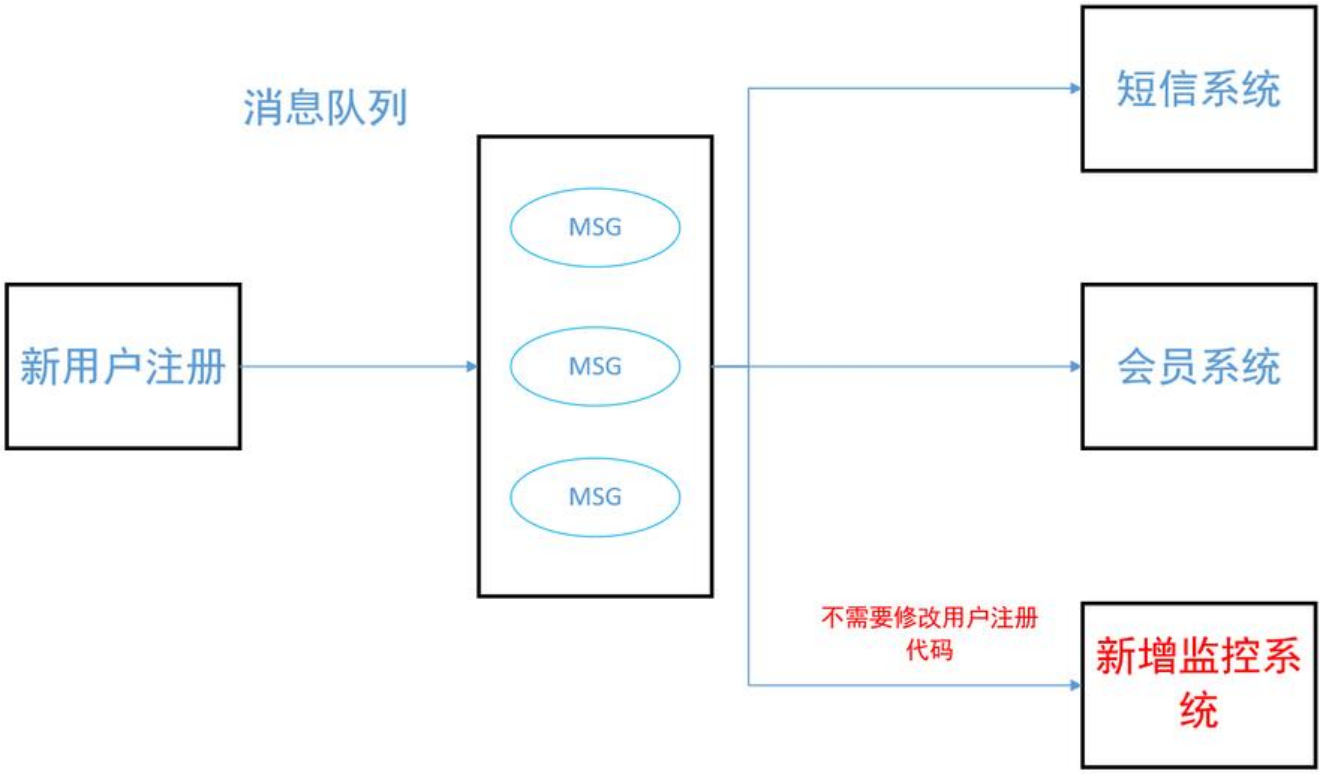
1.2.2.2 解耦应用

比如新用户注册，如果用传统同步调用的方式，那么发邮件、发短信、写数据库、通知关联系统这些码会和用户注册代码直接耦合起来，整个代码看起来就是完成用户注册逻辑后，后面必然跟着发邮件发短信这些代码。如果要新增一个功能，比如将监控用户注册情况，将注册信息发送到业务监控系统就必须要修改前面的代码，至少增加一行代码，发送注册信息到监控系统，我们知道，任何代码的修都可能会引起bug

● 传统方式



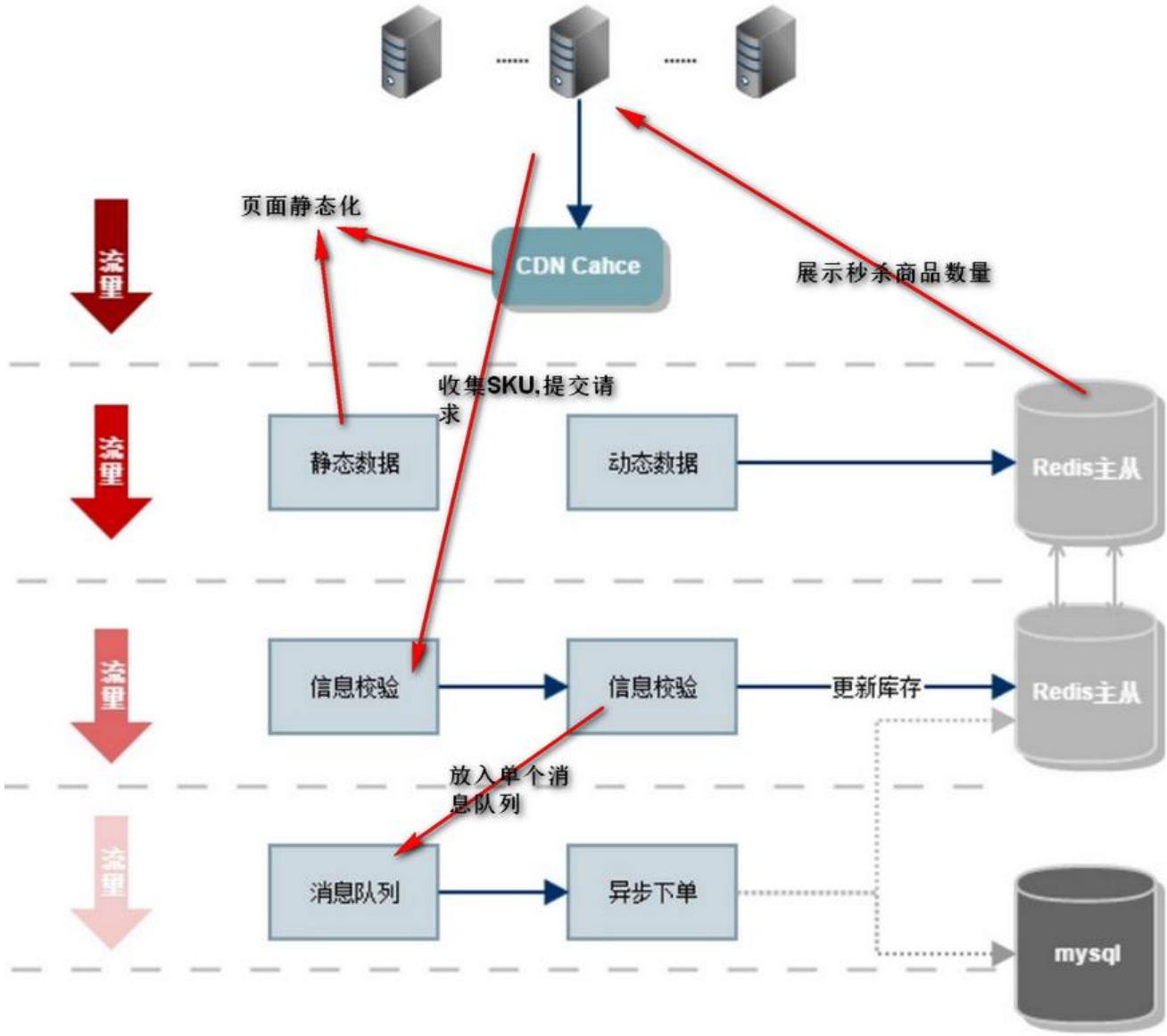
- 使用消息队列



1.2.2.3 秒杀方案

场景说明:秒杀活动对稀缺或者特价的商品进行定时定量售卖，吸引成大量的消费者进行抢购，但又只少部分消费者可以下单成功。
因此，秒杀活动将在较短时间内产生比平时大数十倍，上百倍的页面访问流量和下单请求流量。

- 秒杀前：用户不断刷新商品详情页，页面请求达到瞬时峰值。
- 秒杀开始：用户点击秒杀按钮，下单请求达到瞬时峰值。
- 秒杀后：一部分成功下单的用户不断刷新订单或者产生退单操作，大部分用户继续刷新商品详情页待退单机会



● 秒杀前，用户不断刷新商品详情页，造成大量的页面请求。所以，我们需要把秒杀商品详情页普通的商品详情页分开。对于秒杀商品详情页尽量将能静态化的元素静态化处理，除了秒杀按钮需要务端进行动态判断，其他的静态数据可以缓存在浏览器和CDN 上。这样，秒杀前刷新页面导致的流进入服务端的流量只有很小的一部分。

● 利用读写分离 Redis 缓存拦截流量(活动未开始时拦截大部分动态数据请求)

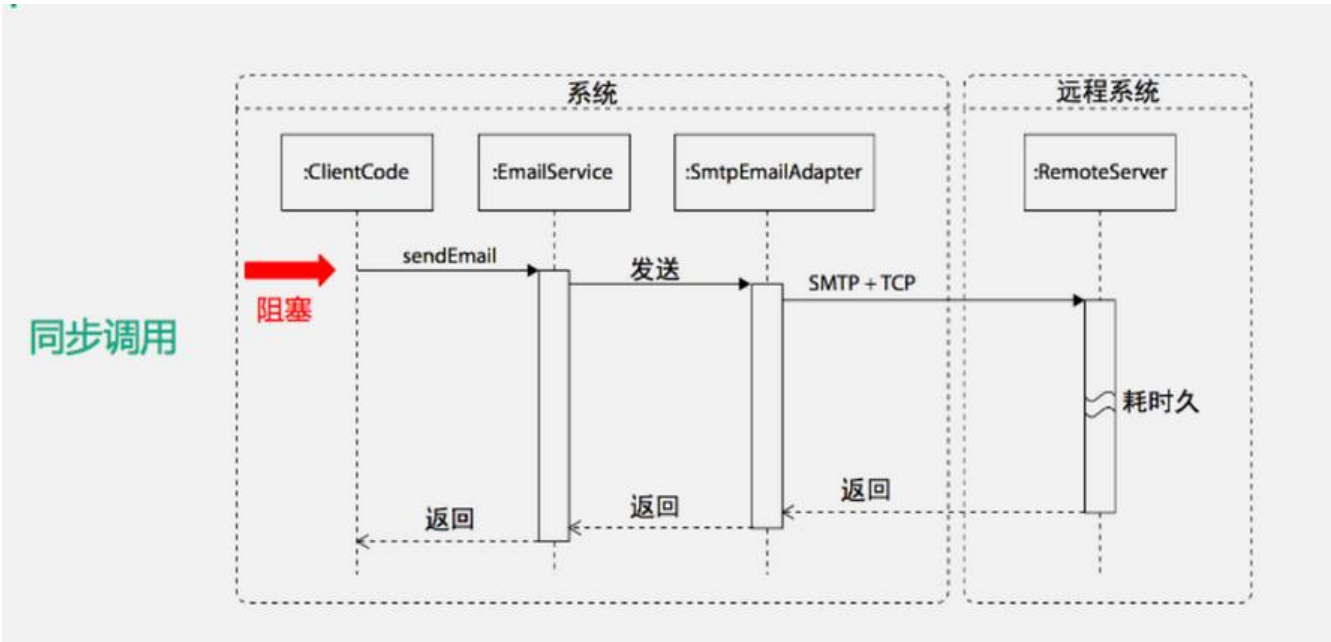
● 成功参与下单后，进入下层服务，开始进行订单信息校验，库存扣量。为了避免直接访问数据，我们使用主从版 Redis 来进行库存扣量

● 如果还有大量并发的请求则利用消息队列组件，当秒杀服务将订单信息写入消息队列后，即可为下单完成，避免直接操作数据库。

同步&异步

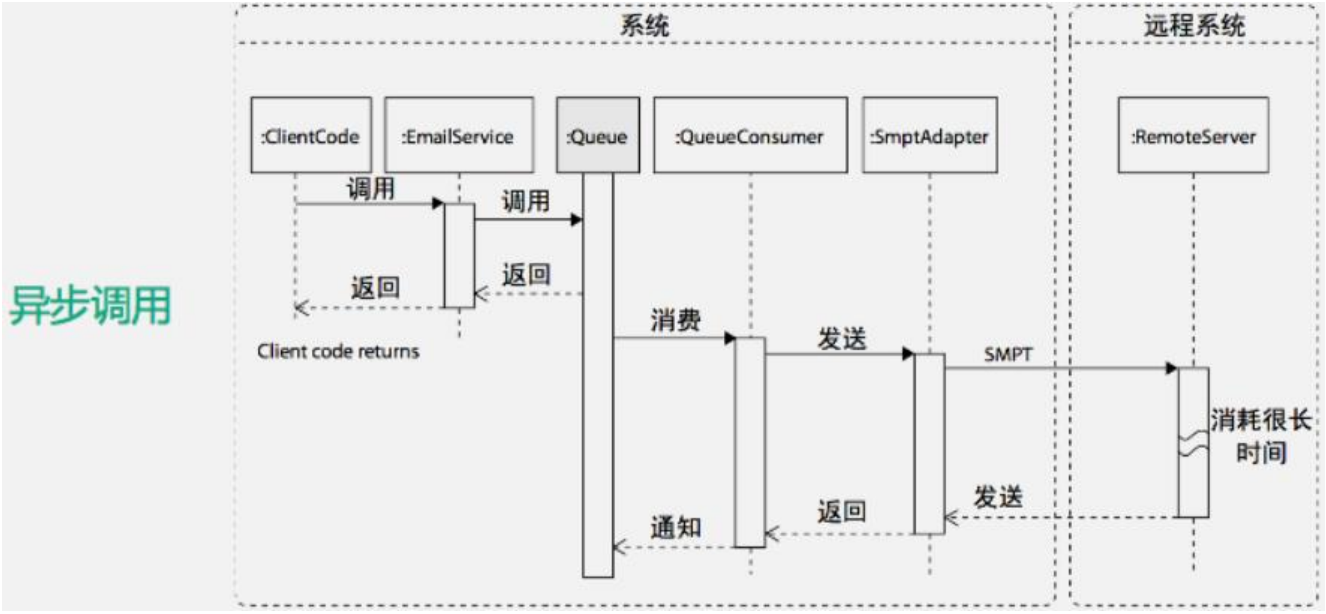
1. 同步调用

是指从请求的发起一直到最终的处理完成期间, 请求调用方一直在同步阻塞等待调用的处理完成.



● 同步调用: 此时整个过程处于阻塞状态, 直到返回状态(代码块结束)

2. 异步调用



● 异步调用过程: 用户只是调用某个接口服务(EmailServer), 接口服务会调用请求发送给消息队列后立返回, 用户不在阻塞等待; 消息队列通过发布一条消息通知消费者消费消息, 消费者进行后续操作; 此时用程序的调用和业务逻辑真正的发送邮件操作不是同步的;

3.异步架构的主要组成部分

● 使用异步架构的主要手段, 就是通过消息队列构建的: 消息的生产者只是把消息发送到消息队列, 而息消费者只是从消息队列中拿出消息, 然后进行业务逻辑处理; 此时消息生产者和消息消费者是异步的;

● 异步架构的三种角色:

● 消息的生产者: 只在乎在什么时刻创建一个合法消息并发送到消息队列

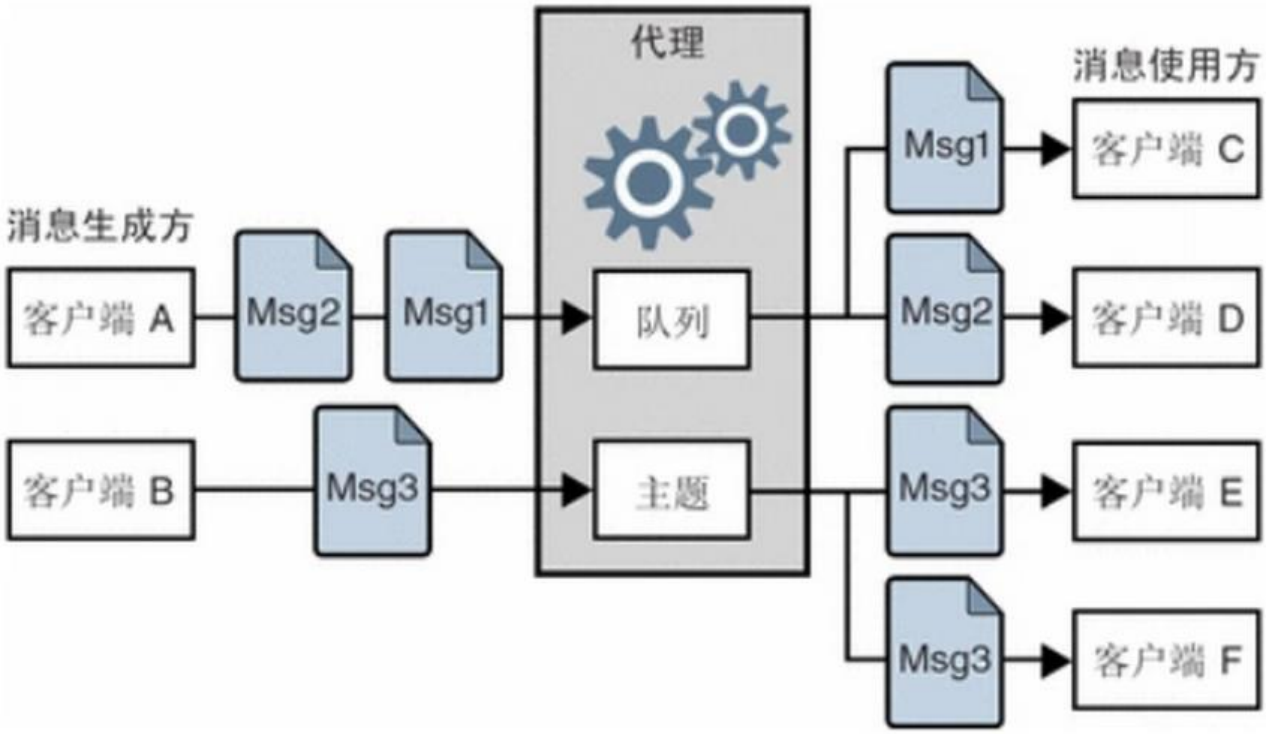
● 消息队列: 消息队列是是消息发送的目的地和发给消费者的一个缓冲, 可以使用共享文件,关系数库, NOSQL系统,最主要的还是分布式消息队列服务器来实现;

● 消息的消费者: 消息的消费者只在乎从消息队列中拿出消息消费消息

JMS(Java Message Service)

JMS即Java消息服务（Java Message Service）应用程序接口，是一个Java平台中关于面向消息中间（MOM）的API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。Java消息务是一个与具体平台无关的API，绝大多数MOM提供商都对JMS提供支持。

1. JMS通讯模型有哪些



● 点对点模式(P2P)：客户端使用此模式向队列目的地发送一条消息，只有一个接收者能够从该目的获得该消息。访问该目的地的其他任何接收者都不能获得该消息。

- 类似于RocketMQ 集群模式

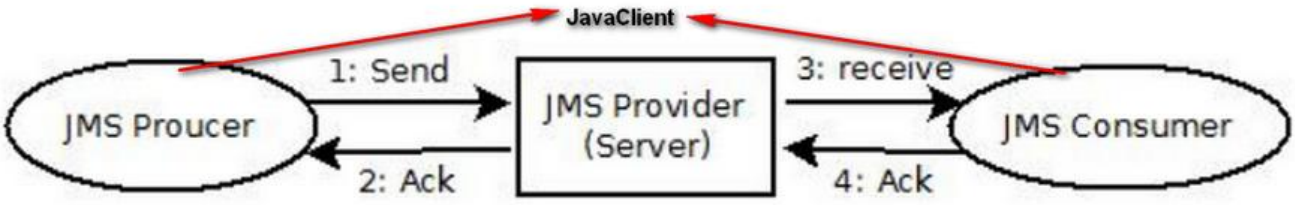
● 发布/订阅模式(publish-subscribe)。客户端使用此广播模式向主题目的地发送一条消息，任意数的使用方订户都可以从该目的地检索此消息。每个订户都获得此消息的一个副本。

- 类似于RocketMQ 订阅模式

2. JMS如何保证消息可靠

2.1 消息可靠？

1. 消息是否已经成功发送到消息中间件
2. 消息是否有丢失的情况 消息是否已经被消费成功



● 消息可靠要保证 JavaClient 和 JMS Service 的消息传输过程中不出问题

● 消息可靠还要保证 JMS消息的持久化是完整无误

2.2 保证传输

- JMS-Proucer 没有成功的把消息发送给 JMS-Provider;
  - JMS中的ACK MODE对producer并没有任何意义
  - 此时需要生产者自己解决了

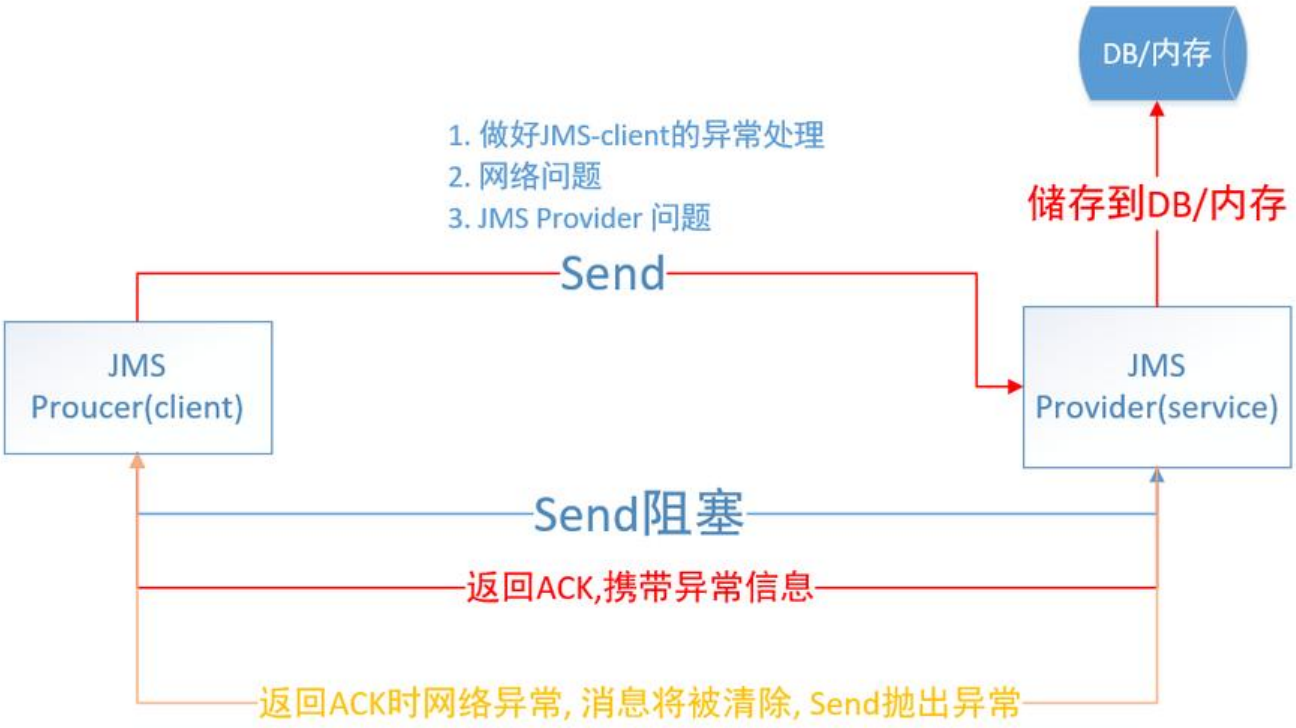
1: 消息生产者中send会阻塞操作(网络IO阻塞),如果网络异常,将导致消息发送失败,此时生产者需要在码设计上做好异常检测或者重发机制;其中网络异常主要体现在"网络中断"或者JMS Provider失效.

- 处理异常, 做好重发;网络异常主要体现在网络中断或者JMS Provider 失效

2: JMS Provider接收到消息之后,将会保存在内存中(或者DB中),此后并立即向此Producer发送"ACK信号,如果JMS Provider在存储上遇到无法解决的异常(比如磁盘故障,嵌入式DB异常),那么"ACK"响应会携带异常信息(message);如果"ACK"发送中网络IO异常导致无法发送,将会导致"此消息"被移除,此时end方法也将从阻塞中返回并抛出异常.因为网络IO异常,通常TCP链接的双端都能及时的感知到.

ACK发送正常,指的是TCP连接上"ACK"数据已经通过网络发送给了Client,这个过程中没有发生异常,此也意味着当前JMS消息被持久存储,producer的send方法也正确返回.此后如果Client端在处理ACK信过程中出现问题(几乎不可能),那么JMS Provider也将认为此消息已经被正确发送.需要提醒的是,在Clie t端:消息的send和ACK在IO流上不是一个连续的过程,在不同的确认时机中,ACK信息是可以是同步的(s nd方法阻塞,知道收到ACK,比如持久化类型的消息),有些时异步获取的(比如事务中的消息,或者非持久消息),不过JMS中的ACK MODE对producer并没有任何意义..

- JMS Provider 接受到消息后, 保存信息到内存或DB; 储存过程发生异常, ACK 响应将会携带异常信息, 返回给 JMS Proucer, 再如果网络中断, 那么此消息会被移除, 并且Send方法也会从阻塞中返回抛出异常



- JMS-Proucer 成功的把消息发送给 JMS-Provider

● JMS-多种签收方式:

● 自动签收: 生产者向队列发送消息后，只要消费者监听了消息队列，消费者将立刻获得消息不管消费者是否成功取得消息，过程是否抛出异常导致消费者无法获得消息，都不会触发重试机制。点：没有事务机制，没有补偿机制

● 事物签收: 对于生产者而言，生产者要想向消息队列发送消息，必须提交事务。对于消费者言，如果消费没有提交事务，则默认表示没有消费，会触发重试机制。双方事务提交

● 手动签收: 需要消费者手动签收，如果消费者没有进行签收，则默认消息没有被消费。 单方务提交

3: 消息消费者receive消息,这个操作在Queue中是consumer主动"拉取"(监听),在Topic中是Provider送;当消费者接收到消息之后,可以立即确认(AUTO),然后再去执行与消息有关的业务;也可以"延迟"确认.

4: 当消息被消费者接收到之后,JMS Provider将消息添加到此消费者的"待确认消息"列表,如果JMS Proider在此消费者的IO通道中阻塞返回,但却没有收到ACK,那么将导致此消息重发.如果ACK正常接收到,么JMS Provider将会把此消息从持久存储设备删除.

● 通过三种签收方式来保证消息在 JMS-Provider消费的可靠

如下为JMS中ACK MODE:

1. AUTO_ACKNOWLEDGE:自动确认,当Consumer客户端(通常是指Consumer所属的Session)在收消息之后,即准备确认消息,通常在receive()方法返回之前,或者在messageListener.onMessage()正常行之后,向Provider发送ACK指令..所谓AUTO_ACK,就是确认的时机有Session"择机"实施;开发者无法扰.

2. DUPS_OK_ACKNOWLEDGE: 允许延迟批量确认,重点在"批量",AUTO_ACK通常是确认一条消息事实上在不同的JMS实现中,它仍然可以像DUPS一样确认多条消息)..当消费者接收到消息之后,并不是即确认,而是"滞留"一段时间之后才会确认(批量确认);这种方式直观上是提升了client端消费数据的速度优先接触消息),但这种模式下,需要消费者能够接受"消息重发"的情况,比如当Consumer客户端失效重启那些尚未确认但已经被"touch"(消费)过的消息有可能会重复接受到;对于JMS Provider而言,如果它"待"一定时间后,仍未收到"确认",将会重发消息.,这种模式,对性能的提升是不明确的,因为较晚的确认消息意味着JMS Provider需要更长时间的保留"待确认"消息列表..究竟多少条消息作为"批量"的边界,有具的JMS实现者决定.

2. CLIENT_ACKNOWLEDGE: 客户端确认,需要要求消息消费者明确的调用message.acknowledge()法来确认此消息的接收成功,无论何种原因(未捕获异常),如果此方法未被调用,那么此消息将会被重发.种模式下,允许消费者自己选择时机确认消息,通常使用在消息编组(group)的情况下:将一系列消息作为个group接收,当group中最后一个消息确认成功后,那么JMS Provider就认为此组消息全部接收成功(需确认组的最后一条消息即可,那么JMS Provider会认为此前的其他消息也接收正常).

2.3 消息的储存

2.3.1 非持久化消息储存 -- 消息的不可靠

正常情况下，非持久化消息是存储在内存中的，持久化消息是存储在文件中的。能够存储的最大消息据在${ActiveMQ_HOME}/conf/activemq.xml文件中的systemUsage节点SystemUsage配置设置一些系统内存和硬盘容量

```
<systemUsage>
   <systemUsage>
```

```
        <memoryUsage>
          //该子标记设置整个ActiveMQ节点的“可用内存限制”。这个值不能超过ActiveMQ本身设
的最大内存大小。其中的
          percentOfJvmHeap属性表示百分比。占用70%的堆内存
          <memoryUsage percentOfJvmHeap="70" />
        </memoryUsage>
        <storeUsage>
          //该标记设置整个ActiveMQ节点，用于存储“持久化消息”的“可用磁盘空间”。该子标记
limit属性必须要进行设置
          <storeUsage limit="100 gb"/>
        </storeUsage>
        <tempUsage>
          //一旦ActiveMQ服务节点存储的消息达到了memoryUsage的限制，非持久化消息就会被转
到 temp store区域，虽然
          我们说过非持久化消息不进行持久化存储，但是ActiveMQ为了防止“数据洪峰”出现时非持
化消息大量堆积致使内存耗
          尽的情况出现，还是会将非持久化消息写入到磁盘的临时区域——temp store。这个子标记
是为了设置这个temp
          store区域的“可用磁盘空间限制”
          <tempUsage limit="50 gb"/>
        </tempUsage>
    </systemUsage>
</systemUsage>
```

- 当非持久化消息堆积到一定程度时, 也就是内存超过指定的阈值时, ActiveMQ会将内存中的非持久消息写入临时文件;
- 当服务器重启之后非持久化的临时文件也会删除

2.3.2 消息的持久化 -- 消息的可靠

- 在JMS-Prodiver 成功接收到消息后, 可以通过以下持久化方式把 消息持久化;
  - KahaDB 储存(默认方式)
  - JDBC储存
  - Memory存储
  - LevelDB储存
  - JDBC With ActiveMQ Journal

ActiveMQ服务端宕机了, 消息会丢失吗 ?

- 不会, KahaDB 作为 ActiveMQ的默认持久化储存方式

HTTP协议同步接口调用失败了怎么做 ?

- 一直阻塞直调用的接口返回结果
  - 查看状态码:
    - 1xx：相关信息
    - 2xx：操作成功
    - 3xx：重定向

- 4xx：客户端错误
    - 5xx：服务器错误

2.2 2xxx状态码

200状态码表示操作成功，但是不同的方法可以返回更精确的状态码。

- GET: 200 OK
- POST: 201 Created
- PUT: 200 OK
- PATCH: 200 OK
- DELETE: 204 No Content

2.3 3xx 状态码

API 用不到301状态码（永久重定向）和302状态码（暂时重定向，307也是这个含义），因为它们可由应用级别返回，浏览器会直接跳转，API 级别可以不考虑这两种情况。

API 用到的3xx状态码，主要是303 See Other，表示参考另一个 URL。它与302和307的含义一样，是"暂时重定向"，区别在于302和307用于GET请求，而303用于POST、PUT和DELETE请求。收到30以后，浏览器不会自动跳转，而会让用户自己决定下一步怎么办。下面是一个例子。

```
HTTP/1.1 303 See Other
Location: /api/orders/12345
```

2.4 4xx 状态码

4xx状态码表示客户端错误，主要有下面几种。

400 Bad Request：服务器不理解客户端的请求，未做任何处理。

401 Unauthorized：用户未提供身份验证凭据，或者没有通过身份验证。

403 Forbidden：用户通过了身份验证，但是不具有访问资源所需的权限。

404 Not Found：所请求的资源不存在，或不可用。

405 Method Not Allowed：用户已经通过身份验证，但是所用的 HTTP 方法不在他的权限之内。

410 Gone：所请求的资源已从这个地址转移，不再可用。

415 Unsupported Media Type：客户端要求的返回格式不支持。比如，API 只能返回 JSON 格式，是客户端要求返回 XML 格式。

422 Unprocessable Entity ：客户端上传的附件无法处理，导致请求失败。

429 Too Many Requests：客户端的请求次数超过限额。

2.5 5xx 状态码

5xx状态码表示服务端错误。一般来说，API 不会向用户透露服务器的详细信息，所以只要两个状态

就够了。

500 Internal Server Error：客户端请求有效，服务器处理时发生了意外。

503 Service Unavailable：服务器无法处理请求，一般用于网站维护状态。