RabbitMQ 的应用场景以及基本原理介绍

1. 背景

RabbitMQ 是一个由 erlang 开发的 AMQP(Advanved Message Queue) 的开源实现。

2. 应用场景

2.1 异步处理

场景说明： 用户注册后, 需要发注册邮件和注册短信 ,传统的做法有两种 1. 串行的方式 ;2. 并行的方式

(1) 串行方式 :将注册信息写入数据库后 ,发送注册邮件 ,再发送注册短信 ,以上三个任务全部完成后才返回给客户端。 这有一个问题是 ,邮件 ,短信并不是必须的 ,它只是一个通知 ,而这种做法让客户端等待没有必要等待的东西 .

(2) 并行方式 :将注册信息写入数据库后 ,发送邮件的同时 ,发送短信 ,以上三个任务完成后 ,返回给客户端 ,并行的方式能提高处理的时间。

假设三个业务节点分别使用　　50ms, 串行方式使用时间　150ms, 并行使用时间　100ms 。虽然并性已经提高的处理时间,但是 ,前面说过 ,邮件和短信对我正常的使用网站没有任何影响，客户端没有必要等着其发送完成才显示注册成功　　,英爱是写入数据库后就返回　.

(3) 消息队列

引入消息队列后,　把发送邮件 ,短信不是必须的业务逻辑异步处理

由此可以看出 ,引入消息队列后,　用户的响应时间就等于写入数据库的时间 +写入消息队列的时间 (可以忽略不计),引入消息队列后处理后 ,响应时间是串行的 3 倍,是并行的 2 倍。

2.2 应用解耦

场景：双 11 是购物狂节 ,用户下单后 ,订单系统需要通知库存系统 ,传统的做法就是订单系统调用库存系统的接口　　.

这种做法有一个缺点　：

当库存系统出现故障时　,订单就会失败。　(这样马云将少赚好多好多钱 ^ ^) 订单系统和库存系统高耦合　.

引入消息队列

订单系统 :用户下单后 ,订单系统完成持久化处理　,将消息写入消息队列 ,返回用户订单下单成功。库存系统　:订阅下单的消息,获取下单消息 ,进行库操作。

就算库存系统出现故障　,消息队列也能保证消息的可靠投递　,不会导致消息丢失 (马云这下高兴了).

流量削峰

流量削峰一般在秒杀活动中应用广泛

场景 :秒杀活动，一般会因为流量过大，导致应用挂掉　,为了解决这个问题，一般在应用前端加入消息队列。

作用 :

1. 可以控制活动人数,　超过此一定阀值的订单直接丢弃　(我为

什么秒杀一次都没有成功过呢　　^^)

2.可以缓解短时间的高流量压垮应用　　(应用程序按自己的最大处理能力获取订单　)

1.用户的请求　,服务器收到之后　,首先写入消息队列　,加入消息队列长度超过最大值　,则直接抛弃用户请求或跳转到错误页面.

2.秒杀业务根据消息队列中的请求信息，再做后续处理　　.

3.系统架构

几个概念说明　：

Broker: 它提供一种传输服务　,它的角色就是维护一条从生产者到消费者的路线，保证数据能按照指定的方式进行传输　　,

Exchange　：消息交换机　,它指定消息按什么规则　,路由到哪个队列。

Queue: 消息的载体 ,每个消息都会被投到一个或多个队列。

Binding: 绑定，它的作用就是把 exchange 和 queue 按照路由规则绑定起来 .

Routing Key: 路由关键字 ,exchange 根据这个关键字进行消息投递。

vhost: 虚拟主机 ,一个 broker 里可以有多个 vhost ，用作不同用户的权限分离。

Producer: 消息生产者 ,就是投递消息的程序 .

Consumer: 消息消费者 ,就是接受消息的程序 .

Channel: 消息通道 ,在客户端的每个连接里 ,可建立多个 channel.

4.任务分发机制

4.1Round-robin dispathching 循环分发

RabbbitMQ 的分发机制非常适合扩展 ,而且它是专门为并发程序设计的 ,如果现在 load 加重 ,那么只需要创建更多的

Consumer 来进行任务处理。

4.2Message acknowledgment 消息确认

为了保证数据不被丢失 ,RabbitMQ 支持消息确认机制 ,为了保证数据能被正确处理而不仅仅是被 Consumer 收到 ,那么我们不能采用 no-ack，而应该是在处理完数据之后发送 ack.

在处理完数据之后发送 ack, 就是告诉 RabbitMQ 数据已经被接收 ,处理完成 ,RabbitMQ 可以安全的删除它了 .

如果 Consumer 退出了但是没有发送 ack, 那么 RabbitMQ 就会把这个 Message 发送到下一个 Consumer , 这样就保证在 Consumer 异常退出情况下数据也不会丢失 .

RabbitMQ 它没有用到超时机制 .RabbitMQ 仅仅通过 Consumer 的连接中断来确认该 Message 并没有正确处理, 也就是说 RabbitMQ 给了 Consumer 足够长的时间做数据处理。

如果忘记 ack, 那么当 Consumer 退出时 ,Mesage 会重新分发 , 然后 RabbitMQ 会占用越来越多的内存 .

5.Message durability 消息持久化

要持久化队列 queue 的持久化需要在声明时指定

durable=True;

这里要注意 ,队列的名字一定要是 Broker 中不存在的 ,不然不能改变此队列的任何属性 .

队列和交换机有一个创建时候指定的标志 durable,durable 的唯一含义就是具有这个标志的队列和交换机会在重启之后重新建立 ,它不表示说在队列中的消息会在重启后恢复

消息持久化包括 3 部分

1. exchange 持久化 ,在声明时指定 durable => true

hannel.ExchangeDeclare(ExchangeName, "direct",

durable: true, autoDelete: false, arguments: null);// 声明消息队列, 且为可持久化的 1

2.queue 持久化 ,在声明时指定 durable => true

channel.QueueDeclare(QueueName, durable: true,

exclusive: false, autoDelete: false, arguments: null);// 声明消息队列, 且为可持久化的 1

3. 消息持久化 ,在投递时指定 delivery_mode => 2(1 是非持久化).

channel.basicPublish("", queueName,

MessageProperties.PERSISTENT_TEXT_PLAIN,

msg.getBytes());　　1

如果 exchange　和 queue　都是持久化的　,那么它们之间的 binding　也是持久化的　,如果 exchange　和 queue　两者之间有一个持久化，一个非持久化　　,则不允许建立绑定　.

注意：一旦创建了队列和交换机　　,就不能修改其标志了　,例如 , 创建了一个　non-durable　的队列 ,然后想把它改变成　durable 的,唯一的办法就是删除这个队列然后重现创建。

6.Fair dispath　公平分发

你可能也注意到了　,分发机制不是那么优雅　,默认状态下,RabbitMQ　将第 n 个 Message　分发给第　n 个 Consumer 。n 是取余后的 ,它不管 Consumer　是否还有　unacked Message ， 只是按照这个默认的机制进行分发　　.

那么如果有个　Consumer　工作比较重 ,那么就会导致有的 Consumer　基本没事可做　,有的 Consumer　却毫无休息的机会　, 那么 ,Rabbit　是如何处理这种问题呢　?

通过 basic.qos　方法设置　prefetch_count=1　, 这样 RabbitMQ

就会使得每个 Consumer 在同一个时间点最多处理一个 Message ， 换句话说 ,在接收到该 Consumer 的 ack 前,它不会将新的 Message 分发给它

channel.basic_qos(prefetch_count=1) 1

注意，这种方法可能会导致 queue 满。当然，这种情况下你可能需要添加更多的 Consumer ， 或者创建更多的 virtualHost 来细化你的设计。

7.分发到多个 Consumer

7.1Exchange

先来温习以下交换机路由的几种类型 ：

Direct Exchange: 直接匹配 ,通过 Exchange 名称 +RountingKey 来发送与接收消息 .

Fanout Exchange: 广播订阅 ,向所有的消费者发布消息 ,但是只有消费者将队列绑定到该路由器才能收到消息 ,忽略 Routing Key.

Topic Exchange ：主题匹配订阅 ,这里的主题指的是 RoutingKey,RoutingKey 可以采用通配符 ,如:*或#, RoutingKey 命名采用 .来分隔多个词 ,只有消息这将队列绑定到该路由器且指定 RoutingKey 符合匹配规则时才能收到消

息;

Headers Exchange: 消息头订阅 ,消息发布前 ,为消息定义一个或多个键值对的消息头 ,然后消费者接收消息同时需要定义类似的键值对请求头 :(如:x-mactch=all 或者 x_match=any) ， 只有请求头与消息头匹配 ,才能接收消息 ,忽略 RoutingKey.

默认的 exchange: 如果用空字符串去声明一个 exchange ,那么系统就会使用 ” amq.direct ” 这个 exchange ， 我们创建一个 queue 时,默认的都会有一个和新建 queue 同名的 routingKey 绑定到这个默认的 exchange 上去

channel.BasicPublish("", "TaskQueue", properties, bytes);1

因为在第一个参数选择了默认的 exchange ， 而我们申明的队列叫 TaskQueue ， 所以默认的, 它在新建一个也叫 TaskQueue 的 routingKey ， 并绑定在默认的 exchange 上, 导致了我们可以在第二个参数 routingKey 中写 TaskQueue ， 这样它就会找到定义的同名的 queue ， 并把消息放进去。

如果有两个接收程序都是用了同一个的 queue 和相同的 routingKey 去绑定 direct exchange 的话, 分发的行为是负

载均衡的，也就是说第一个是程序 1 收到，第二个是程序 2 收到，以此类推。

如果有两个接收程序用了各自的 queue，但使用相同的 routingKey 去绑定 direct exchange 的话，分发的行为是复制的，也就是说每个程序都会收到这个消息的副本。行为相当于 fanout 类型的 exchange 。

下面详细来说 ：

7.2 Bindings 绑定

绑定其实就是关联了 exchange 和 queue，或者这么说:queue 对 exchange 的内容感兴趣 ,exchange 要把它的 Message deliver 到 queue 。

7.3Direct exchange

Driect exchange 的路由算法非常简单 :通过 bindingkey 的完全匹配，可以用下图来说明 .

Exchange 和两个队列绑定在一起 ,Q1 的 bindingkey 是 orange， Q2 的 binding key 是 black 和 green.

当 Producer publish key 是 orange 时 ,exchange 会把它放到 Q1 上 ,如果是 black 或 green 就会到 Q2 上 , 其余的 Message 被丢弃 .

7.4 Multiple bindings

多个 queue 绑定同一个 key 也是可以的 ,对于下图的例子 ,Q1 和 Q2 都绑定了 black, 对于 routing key 是 black 的 Message , 会被 deliver 到 Q1 和 Q2 , 其余的 Message 都会被丢弃 . 7.5 Topic exchange

对于 Message 的 routing_key 是有限制的, 不能使任意的。

格式是以点号 “ . ” 分割的字符表。比如 : ” stock.usd.nyse ”, “ nyse.vmw ”, “ quick.orange.rabbit ”。你可以放任意的 key 在 routing_key 中, 当然最长不能超过 255 bytes 。

对于 routing_key , 有两个特殊字符

*(星号)代表任意一个单词 #(hash)0 个或多个单词

Producer 发送消息时需要设置 routing_key , routing_key 包含三个单词和连个点号 o, 第一个 key 描述了 celerity(灵巧), 第二个是 color(色彩), 第三个是物种 :

在这里我们创建了两个绑定： Q1 的 binding key 是 ”.orange. “Q2 是 “ ..rabbit 和 ”“ lazy.# ”：

Q1 感兴趣所有 orange 颜色的动物 Q2 感兴趣所有 rabbits 和所有的 lazy 的.

例子 :rounting_key 为 “ quick.orange.rabbit ”将会发送到 Q1 和 Q2 中

rounting_key 为 ” lazy.orange.rabbit.hujj.ddd ”会被投递到 Q2 中,#匹配 0 个或多个单词。

8. 消息序列化

RabbitMQ 使用 ProtoBuf 序列化消息 ,它可作为 RabbitMQ 的 Message 的数据格式进行传输 ,由于是结构化的数据 ,这样就极大的方便了 Consumer 的数据高效处理 ,当然也可以使用 XML , 与 XML 相比 ,ProtoBuf 有以下优势 :

1. 简单

2.size 小了 3-10 倍

3. 速度快了 20-100 倍

4. 易于编程

6. 减少了语义的歧义　.

，ProtoBuf　具有速度和空间的优势，使得它现在应用非常广泛