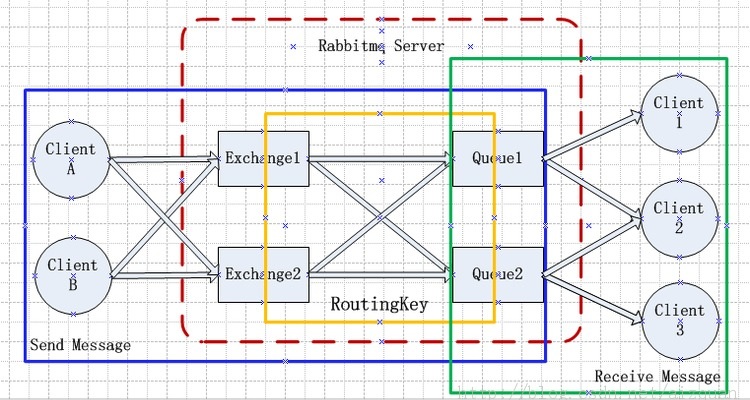
* RabbitMQ简介

RabbitMQ是当前最主流的消息中间件之一。其基于AMQP（Advanced Message Queuing Protocol）高级消息队列协议开发。主要用于组件之间的解耦，主要特征是面向消息、队列、路由（包括点对点和发布/订阅）、可靠性、安全。

RabbitMQ服务端由Erlang语言编写，用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。

* 架构图



* 基本概念

1. RabbitMQ Server

服务端，又称broker server。它的角色就是维护一条从Producer到Consumer的路线，保证数据能够按照指定的方式进行传输。

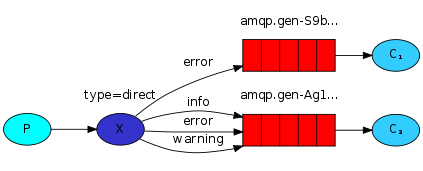
1. Producer

生产者，消息的发送方。决定把消息发送到具体的Exchange。

1. Consumer

消费端，消息的接收方，根据RoutingKey来绑定Queue和Exchange。需要注册到RabbitMQ Server。

1. Exchange



交换器，图中X部分。连接生产者与队列之间的角色。生产者将消息发送到exchange后，再有exchange路由到具体的队列。

1. Queue

Queue（队列）是RabbitMQ的内部对象，用于存储消息，可持久化消息。多个消费者可订阅同一个队列，按照路由规则可指定路由到具体的消费者。

1. RoutingKey

路由键。生产者在将消息发送给Exchange的时候，一般会指定一个routing key，来指定这个消息的路由规则。而这个routing key需要与Exchange Type及binding key联合使用才能最终生效。

1. Connection

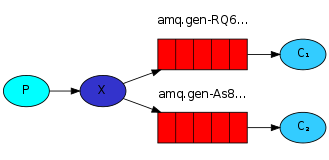
一个TCP的连接。Producer和Consumer都是通过TCP连接到RabbitMQ Server。

1. Channel

虚拟连接。它建立在上述的TCP连接中。数据流动都是在Channel中进行的。也就是说，一般情况是程序起始建立TCP连接，第二步就是建立这个Channel。

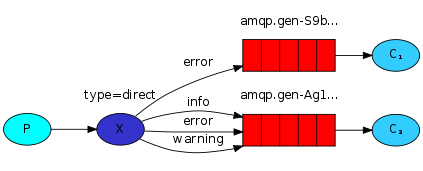
* 路由类型（Exchange Types）

1. Fanout



它会把消息发送到所有与exchange绑定的queue中，类似于将消息广播。上图中生产者发送到exchange的消息，都会路由到队列中，并最终由消费端消费。

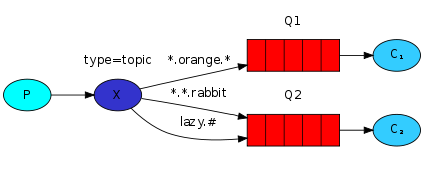
1. Direct



根据exchange与queue绑定的bindingKey，消息将发送到routingKey与bingdingKey一致的queue中。

如上图中生产者的routingKey为warning，则只会发送到queue2，并被C2消费。如routingKey为error，则会发送到queue1和queue2，并被C1、C2消费。

1. Topic



与Direct类似，区别是可结合通配符进行路由。它有两种特殊字符“\*”与“#”，用于做模糊匹配，其中“\*”用于匹配一个单词，“#”用于匹配多个单词

如上图中生产者的routingKey为quick.orange.rabbit，则会发送到queue1和queue2，并被C1、C2消费。如routingKey为lazy.brown.fox，则只会发送到queue2，并被C2消费。

1. Headers

根据发送的消息内容中的headers属性进行匹配。在绑定Queue与Exchange时指定一组键值对，当消息发送到Exchange时，RabbitMQ会取到该消息的headers（也是一个键值对的形式），对比其中的键值对是否完全匹配Queue与Exchange绑定时指定的键值对；如果完全匹配则消息会路由到该Queue，否则不会路由到该Queue。

* 如何使用

Rabbitmq建议每个业务，使用同一个connection，但是用不同的channel。建议每个不同的业务都使用自己的一套生产者、消费者、channel、exchange、exchangeType、queue、routingkey。

1. 定义具体的业务配置类

在前文介绍了，一个完整的消息架构需要以下部分组成：生产者、消费者、channel、exchange、exchangeType、queue、routingkey。而定义配置类的目的，就是为了维护一套通信规则，将它们都连接起来。当应用程序启动时，会自动将生产者和消费者的channel注册到Rabbitmq服务端。具体代码可参考代码DemoConfig.java。

1. 定义属于自己的委托类

相当于你在消费端需要执行的业务处理，都放在这个委托类的方法中处理。具体代码可参考代码DemoDelegate.java。

1. 引入自己配置的RabbitmqTemplate，并发送消息

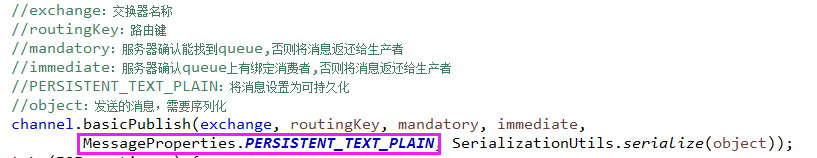
利用@Autowired和@Qualifier指定自己配置的RabbitmqTemplate，再调用send方法进行消息的发送。

* 持久化

RabbitMQ默认不持久化，这意味着一旦服务器宕机或重启，队列以及消息都会丢失。为了避免这种情况发生，RabbitMQ提供了持久化的机制。使用消息持久化，需要设置以下几点：

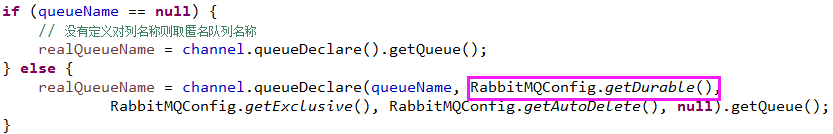
1. 消息的持久化

在生产者发送消息时，设置消息的属性为“PERSISTENT\_TEXT\_PLAIN”，可使消息持久化。如下图代码：



1. queue的持久化

在定义队列时，需要将队列设置为“durable”，来使队列可持久化。如：



1. exchange的持久化

在定义exchange时，需要将队列设置为“durable”，来使交换机可持久化。如：

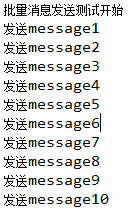


* 验证持久化

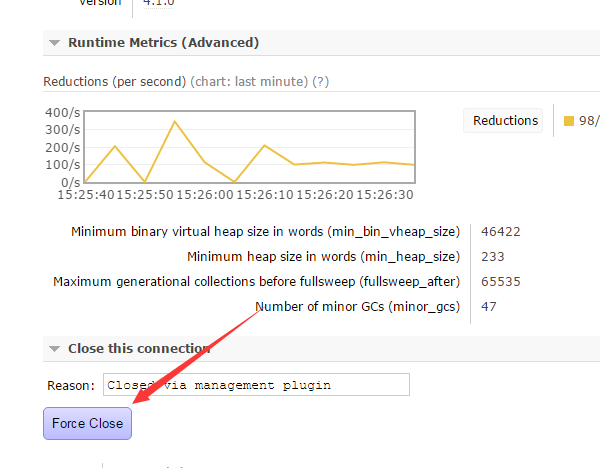
设置了消息持久化是否就意味着消息就不会丢失呢，带着这个问题我在本地环境验证了一下。有以下场景：

1. 生产者发送消息，消费者在消费消息的过程中（还没消费完毕），服务器宕机或链接断开。

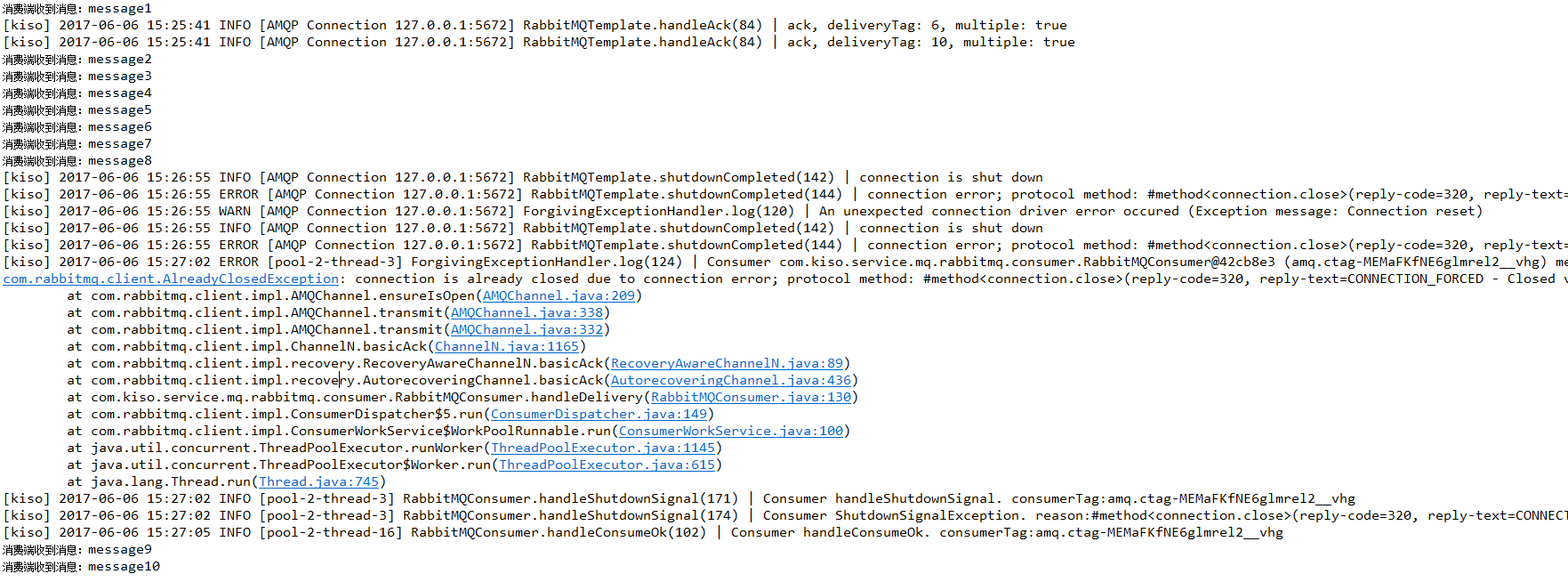
首先，模拟批量发送10条消息，并在消费者设置每条消息等待10秒再进行处理。



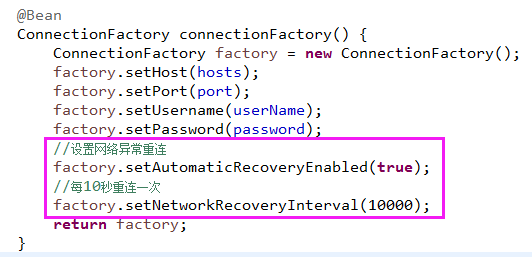
在消费消息的过程中断开rabbitmq连接：



系统会自动重连，当连上后会继续消费未消费完的消息：



通过测试发现，当rabbitmq服务端断开连接后，客户端会实现了自动重连，自动重连是通过以下的配置实现的：



另外，为确保未消费的消息，在重新连接后继续消费，这里用到了rabbitmq的confirm机制：

生产者将channel设置成confirm模式后，将发送消息到对应的队列，消费者收到消息并消费完毕后，需要回ack一个确认标识给生产者，告诉生产者已经收到消息并处理完毕了，可以接着发下一条的消息。实际上可以理解为给生产者的一个回调。

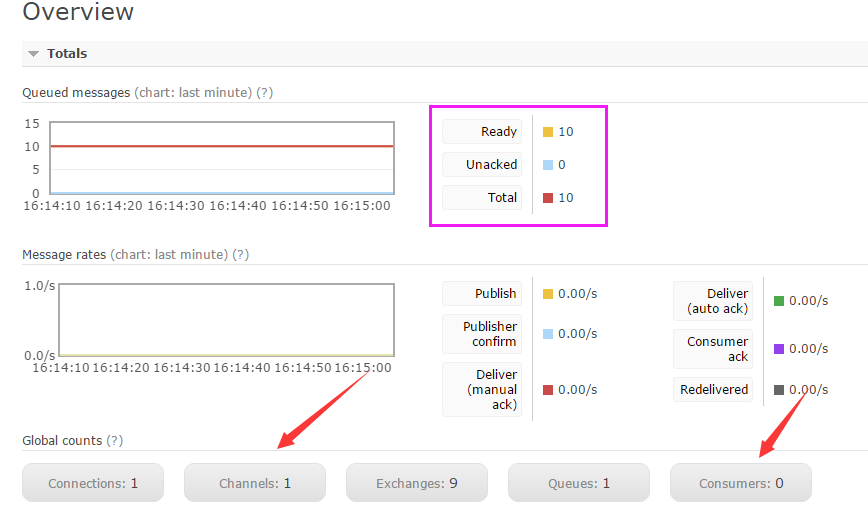
如果RabbitMQ因为自身内部错误导致消息丢失，就会发送一条nack消息，生产者同样可以在回调方法中处理该nack消息。对于nack的消息，rabbitMQ会重新放进队列。

1. 消费者在消费消息的时候，出现业务异常。

我们同样批量发10条消息，处理每条消息时抛出一个异常，如下图日志：



再查看消息的情况：



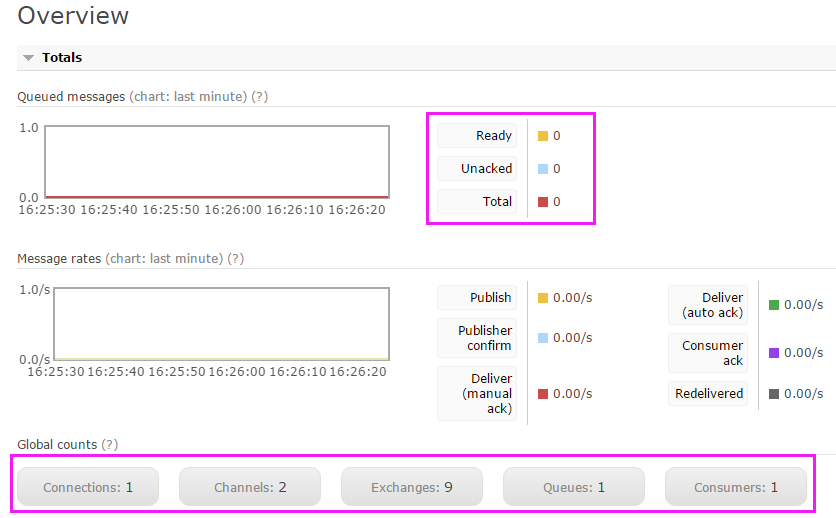
事实上，当消费第一条消息的时候，由于抛出了一个异常，根据日志显示，rabbitmq将消费者的channel关掉了，只剩下一个channel，consumer变为了0个。也就是说当出现了业务异常，会停掉消费者。但由于做了持久化设置，所有的10条消息都会保存在磁盘上，并不会丢失。

如果我们修复这个异常后，重新启动我们应用程序的服务端，再看看效果：



先停掉应用程序，connection，channel，consumer都变为0了。Ready状态的消息还有10条。

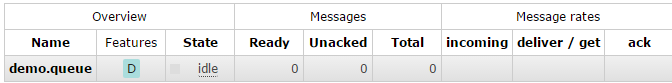
再启动应用程序：



发现connection，channel，consumer都恢复正常了。而消息也陆续被消费。

1. 服务端宕机情况下，生产者发送消息。

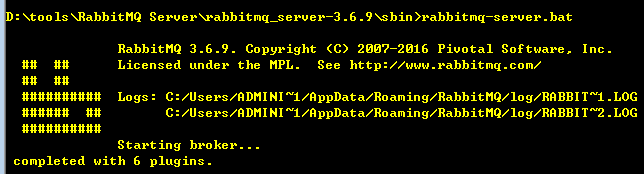




上图显示exchange，queue已设置持久化。现在停掉服务端：

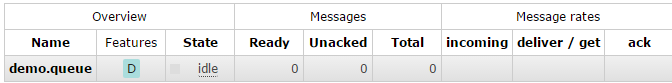


发送消息，然后重新启动服务端。



再查看exchange，queue的情况：





实验证明，如果rabbitmq服务器宕机后，所有后面发送的消息都会丢失，这种情况是无法持久化的，所以只能重新发送。更加保险的做法是，将Rabbitmq服务器做集群并且加入镜像队列（待续）。

* 集群+镜像队列

待续

* 压力测试

处理器：Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz 4核

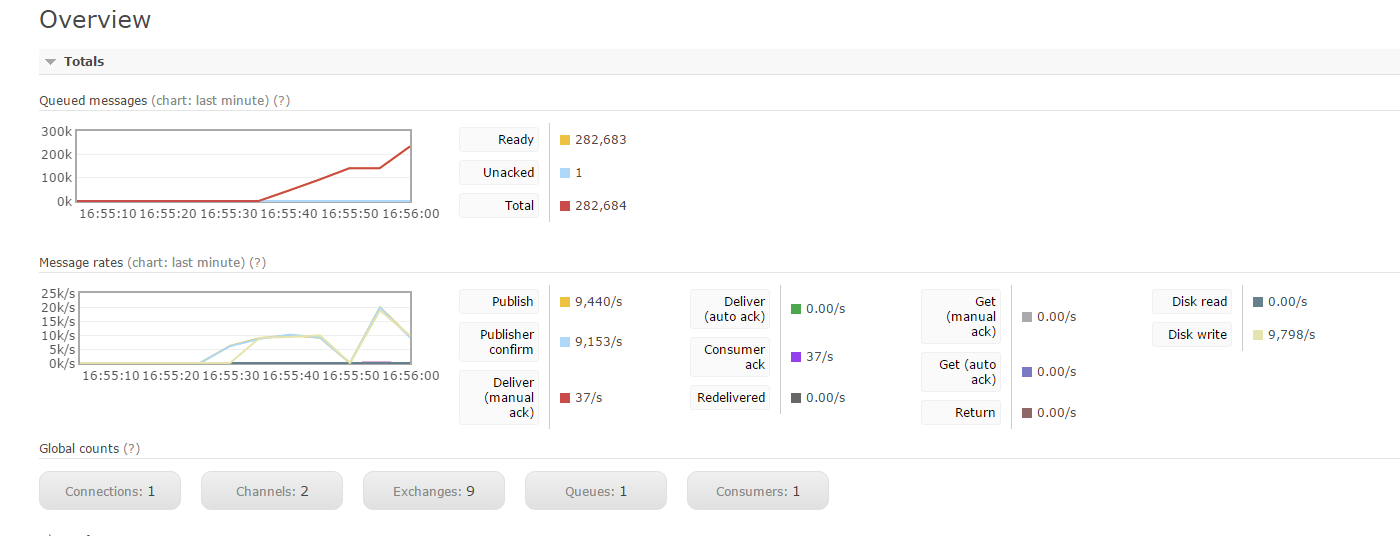
内存：16G金士顿 DDR3 1600MHz

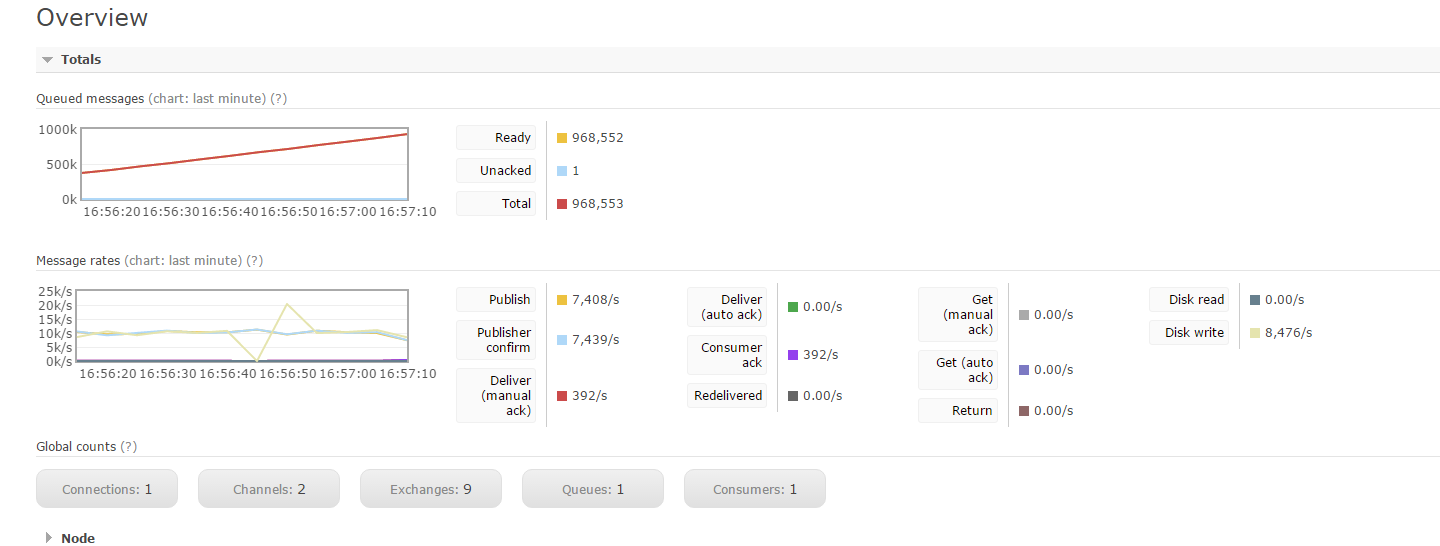
硬盘：金士顿 SUV400S37（240G/固态硬盘）

操作系统：Windows 7 旗舰版 64位

本地测试100万条消息发送的情况。

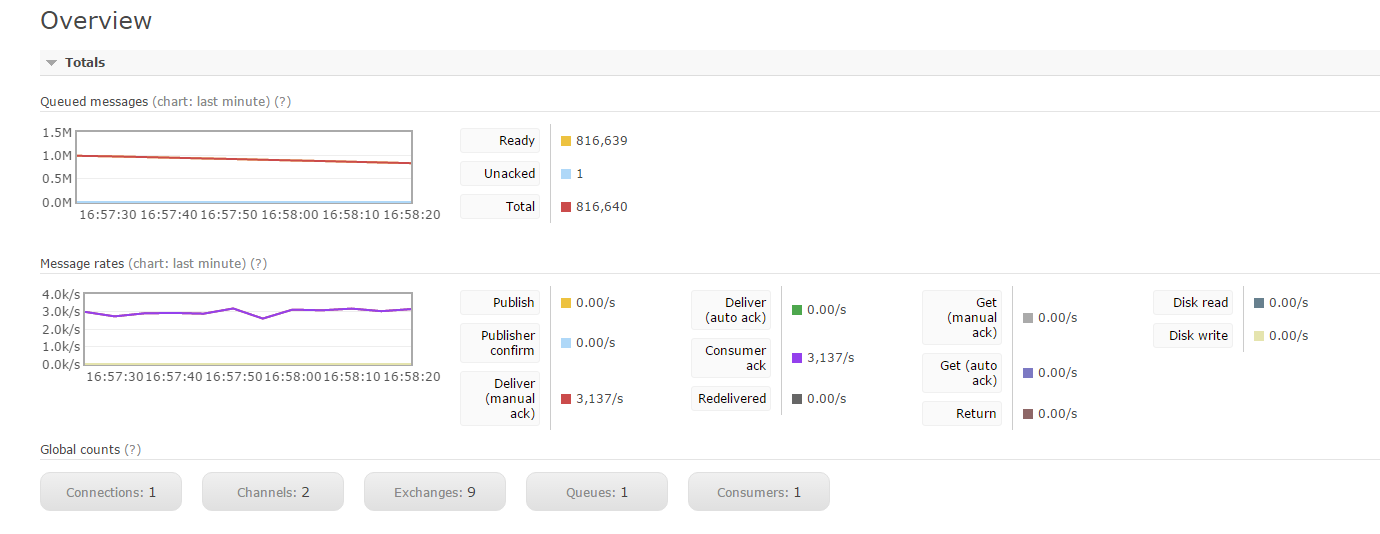
发送中：





有图中的监控可以看出，每秒rabbitmq服务端每秒可以接收7000+ 到9000+的消息。

发送完毕，消费中：



有图中的监控可以看出，消费者回ack大概能达到3000次/秒.