# 简介

消息队列使用消息将应用程序连接起来，这些消息通过像RabbitMQ这样的服务在应用程序之间进行路由。

AMQP（Advanced Message Queuing Protocol 高级消息队列协议）是一个开放标准，为面向消息的中间件设计，实现组件之间的解耦，消息的生产者和消费者不需要知道彼此的存在。它的主要特点是面向消息，队列，路由（点对点和发布/订阅），可靠性和安全。

RabbitMQ实现了AMQP

# 客户端

ConectionFactory，Connection，Channel是RabbitMQ的客户端API对外提供的基本对象。

ConectionFactory 是Connection 的工厂。

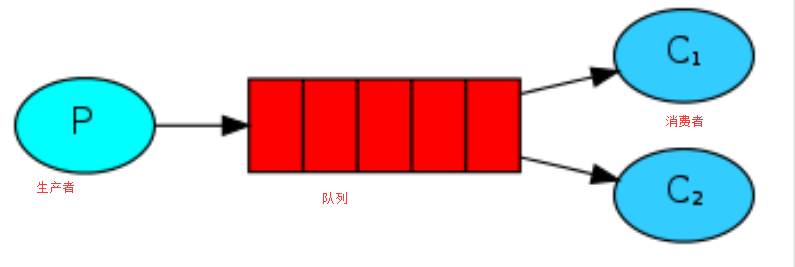
Connection是与RabbitMQ的Socket连接，封装了AMQP的协议部分。

Channel是与RabbitMQ交互的一个重要接口，大部分业务操作是在Channel中完成，包括定义Queue，定义Exchange，绑定Queue和Exchange，发布消息等。

# RabbitMQ内部

## Queue

队列是RabbitMQ的内部对象，用来存储消息。RabbitMQ所有消息都只能存在Queue中。



### 消息ACK

实际应用中，消息传递到消费者后会从队列中移除消息，但是消费者发生意外（例如宕机）没有处理消息，消息就被队列移除造成消息丢失的情况。为了避免这种情况，可以要求消费者在接收消息后发送一个回执（ACK）给RabbitMQ，RabbitMQ在收到回执后才会将消息移除。

这种模式要求消费者必须在接收到每个消息后发送回执，否则会产生严重BUG：消息队列中消息不移除，消费者会重复消费相同消息。

### 消息持久化

如果希望及时在RabbitMQ服务重启的情况下，也不会丢失消息，可以将Queue和exchange都设置为可持久化（durable）的，这样即使rabbitmq重启，queue和exchange对象也不会消失。

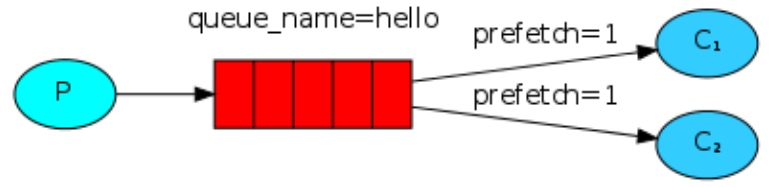
在消息对象的属性上设置deliverymode=2，能够让消息持久化

如果要对小概率的消息丢失（比如rabbitmq收到了生产者消息，但是还没来得及持久化rabbitmq就宕机了）进行管理，就需要用到事务

### Prefetch count

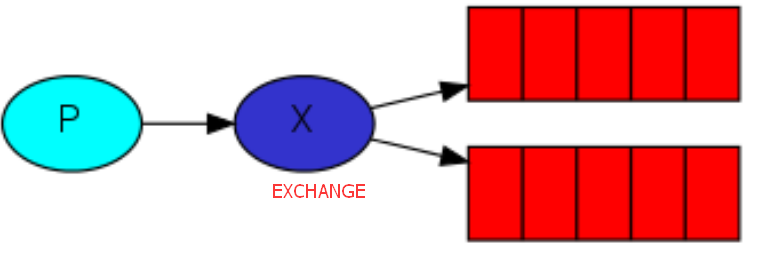
当一个queue上有多个消费者的时候，queue会将消息逐个分发给消费者（1个消息只会发送给一个消费者进行处理，而不是将1个消息复制多份给每个消费者处理），每次分发的消息个数通过prefetch count来设置

设置prefetch count设置queue每次发送给消费者的消息数，例如设置prefetch count=1，则queue每次发送个每个消费者一条消息，消费者处理完后，queue会再给该消费者发送一条消息：



## Exchange

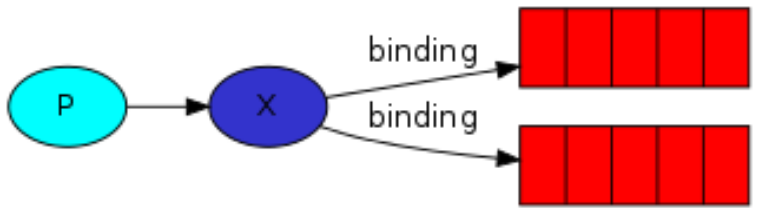
生产者不是将消息直接传送到queue中，而是传送给exchange(交换器)，由exchange将消息路由到一个或多个queue中：



### Binding

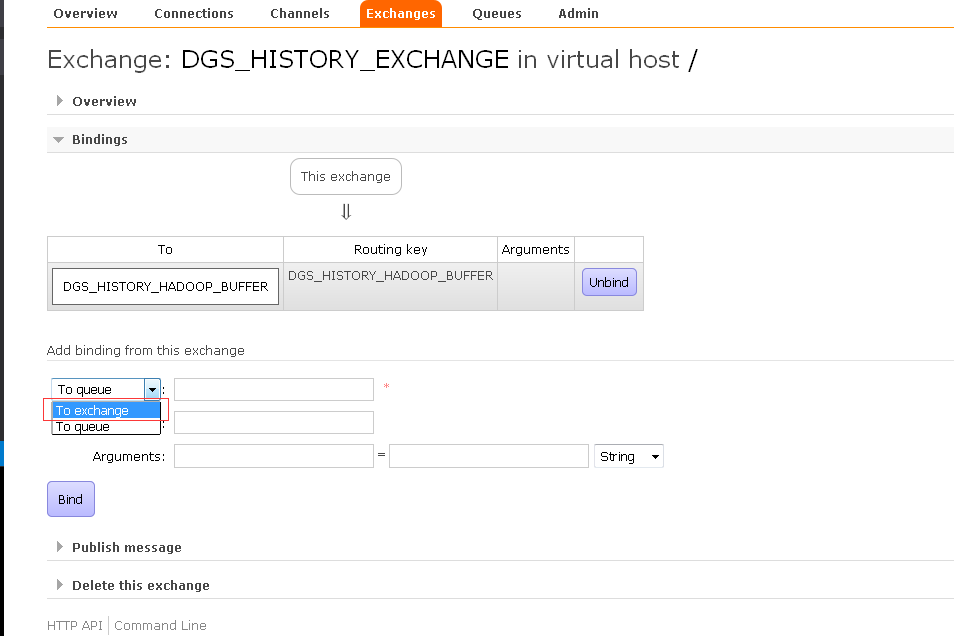
RabbitMQ通过Binding将Exchange和Queue关联起来

Exchange和binding是多对多的关系，同一个queue和一个exchange也可以binding多次（指定不同的binding key）。

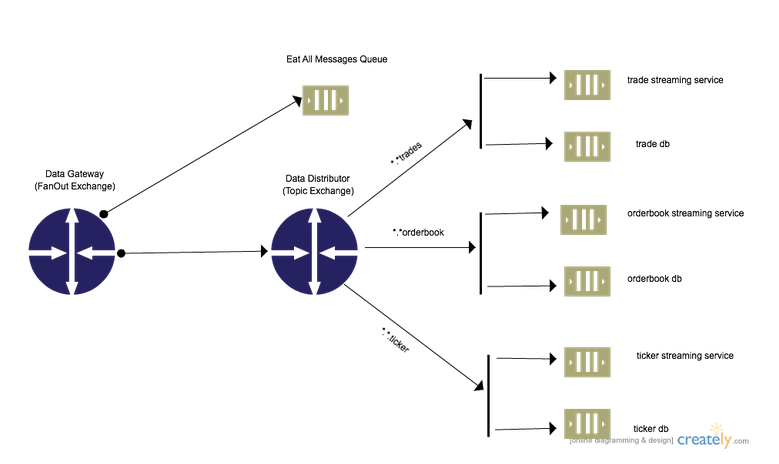


实际上绑定也可以发生在Exchange和Exchange之间：

控制界面



拓扑图：



详细：

http://skillachie.com/2014/06/27/rabbitmq-exchange-to-exchange-bindings-ampq/

### Binding key

使用binding时，一般会指定binding key。

### Routing key

生产者在将消息发送给exhange时，一般会指定routig key，用来匹配路由规则。Routing key的长度限制在255字节内。

### Exchange Types

Binding key，Routing key和Exchange Types联合作用，决定将消息路由到哪一个队列中。·

路由策略是，exchange会使用消息的routing key和自己绑定的所有binding key进行匹配，匹配规则由Exchange Types决定，如果符合匹配规则，那么消息就会被发送到所有符合匹配的queue

如果消息没有匹配到任何binding上，消息就会被丢弃

常用的有fanout,direct,topic,headers四种

#### fanout

和bingding key与routing key无关，在这种模式下，Exchange会把自己受到的所有消息发送到与它绑定的所有Queue

#### direct

exchange会把消息路由到那些binding key和routing key完全匹配的queue

例如exchange与queue1的binding key是“a”，那么当消息的routing key是“a”的数据到达exchange后，会被路由到queue1上。

#### topic

exchange会将bingding key和routing key按指定的规则进行匹配（类似于正则匹配）。

1. routing key和binding key都使用的是”.”分隔的字符串
2. binding key中使用两种特殊字符 “\*”和“#”进行模糊匹配，其中”\*”匹配一个单词；“#”用于匹配0个或多个单词

例如binding key=\*.lazy.#

那么routing key=”abc.lazy.def.ghi”就符合匹配规则

#### headers

这种模式下，exchange不依赖routing key和binding key来进行消息路由，而是根据消息的headers属性进行匹配。

当消息发送到Exchange时，RabbitMQ会取到改消息的headers（一个键值对），对比其中的键值对是否完全匹配Queue和Exchange绑定是指定的键值对；如果完全匹配则消息会路由到该queue，否则就不路由到该queue。

这种模式较少使用。

## 消息属性

AMQP为每个消息定义了14种属性，消息属性随着消息一起发送。常用消息属性如下：

* delivery\_mode : 标记消息是持久性消息还是瞬态信息。值2表示是持久消息
* content\_type : 用来描述MIME的类型。如把其类型设定为JSON；
* reply\_to : 用于命名一个回调Queue，用于接收PRC响应；
* correlation\_id : 用于与相关联的请求的RPC响应.

## 远程过程调用(RPC)

### 默认Exchange

接收应答消息需要有专门的Queue，它不需要被绑定到Exchange上，此时MQ就会自动将Queue绑定到一个默认Exchange上：



详细信息：



可以看到这个Exchange不能被删除，默认是绑定到了所有的queue，但是这个exchange不能被显式绑定或显式接触绑定，也不能用ExchangeDeclare函数来显式声明（Exchange名称为“”），routing key就是对应的queue的名称。

### 原理

MQ的设计，本身对消息的处理就是异步的。生产者将消息发送给MQ后不知道消息的处理结果甚至不知道消息是否能够被处理。

实际应用中，是可以做到同步处理的：

1. 生产者发送消息时，在消息的属性（MessageProperties，在AMQP协议中定义了14种属性，随着消息一起发送）中设置两个值：replyTo（一个Queue名称，告诉处理结果发送到哪个消息队列）和correlationId（这个消息的标识号，rabbitmq服务处理完消息后将此属性返还，生产者根据这个id了解哪个消息被处理）
2. 服务端接收消息并处理
3. 服务端处理消息后，生成一条应答消息到replyTo指定的queue，同时带上之前的correlationId属性
4. 客户端此前已经订阅了replyTo指定的queue，收到服务应答消息后根据correlationId属性判断哪条请求消息被执行。

# 实际使用

MQ是对字符大小写敏感的

## 性能

根据测试， 512字节的包大小，非事务下读写达到1W3包/s，事务下读写达到3000包/s

## 持久化

默认当rabbitmq的内存占用了机器内存的25%以上，就会进行数据持久化；当内存下降，就会加载持久化的数据。

Exchange和queue如果不设置durable=true，即采用非持久化，那么这两个对象可能会自动被删除（例如服务重启），数据也会丢失

消息持久化，设置消息属性deliverymode=2。队列持久化由队列控制，但是队列中消息是否持久化由消息持久化控制，也就是说，如果重启之前那个queue里面还有没有发出去的消息的话，重启之后那队列里面是不是还存在原来的消息，这个就要取决于发送者在发送消息时对消息的设置了。

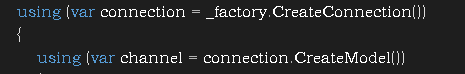
<http://blog.csdn.net/hanruikai/article/details/42778921>

## 数据交互相关

如果没有提到设置，就是采用的默认设置

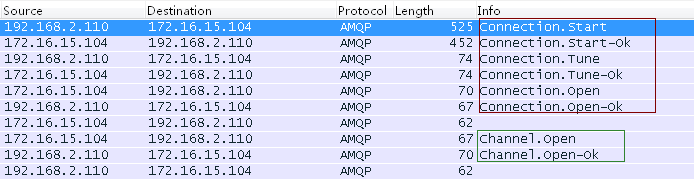
### 开启Channel

* 代码



创建连接channel

* 报文



红框开启了Connection；绿框是开启了channel；前后几个长度62的包是AMQP的心跳包

* MQ监视



Connection



Channel

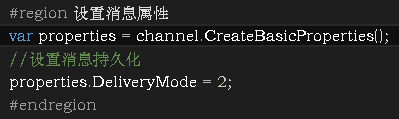
可以看到channel的属性中 Perfetch是空，也就是默认这个channel接收的效率不做限制。

MQ服务会自动关闭一定时间不活跃的channel对象

### 基本写数据到服务

在创建的channel上公布数据：

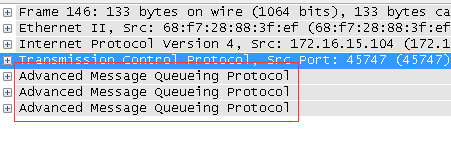
* 代码





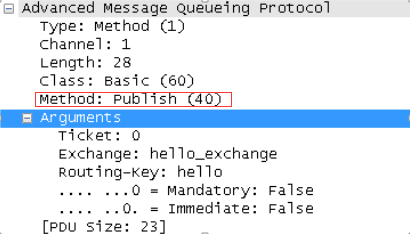
* 报文





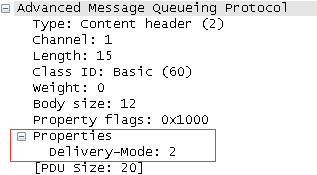
一个报文封装成3个AMQP数据项，type分别是：Method，header和body（数据）

* 第一个数据项method：



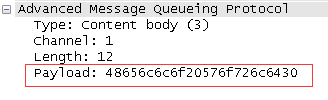
publish表示是发布数据，Arguments也在这个数据项中，其中包括exchange和routingkey

* 第二个数据项header：



数据项包含消息属性propperties

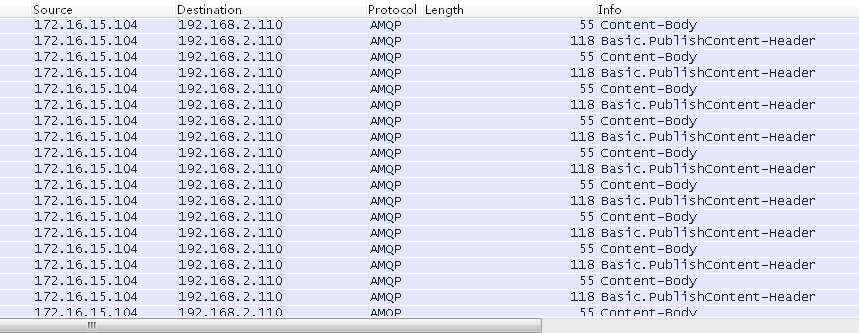
* 第三个数据项body：



payload是实际的数据

### 可靠写数据

默认情况下，RabbitMQ服务器不会针对接收到的数据做任何确认，例如：



发送方没有收到RabbitMQ的任何确认消息。

这种发送数据的方式实际是不可靠的，因为上层实际是将数据推送到MQ客户端缓存中，MQ客户端再将数据提交到Rabbitmq服务器。这个过程中，任何网络上的问题都会导致数据丢失，因为针对消息没有任何确认机制。

可以使用重量级的事务保证数据可靠，也可以使用轻量级的确认机制，来保证数据可靠。

#### Confirm

* API相关接收

定义在接口IModel上，即通道对象上



1.在channel上调用方法ConfirmSelect方法，可以设置这个Channel在推送数据到MQ服务时，需要进行消息确认：



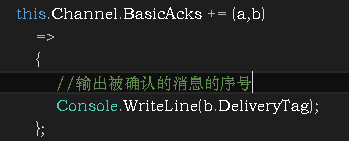
使用



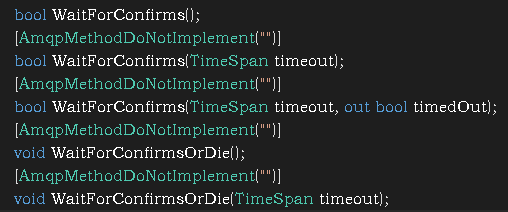
2.在channel上订阅事件BasicAcks后，会在收到MQ服务发送的ACK消息后触发这个事件绑定的方法



使用

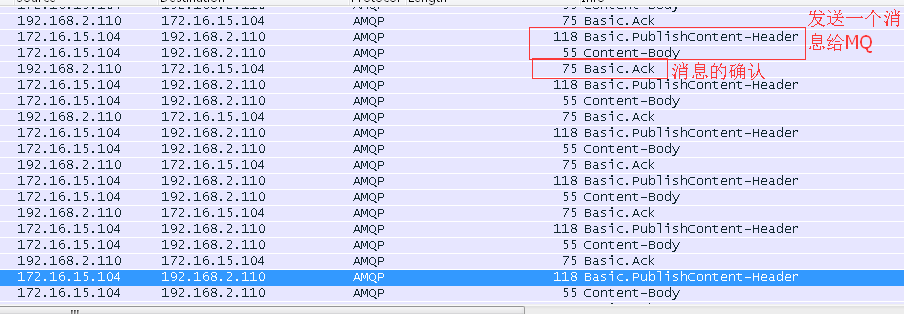


3.在channel上调用方法WaitForConfirms或者WaitForConfirmsOrDie，会阻塞等待消息的确认（任何一个的ack）

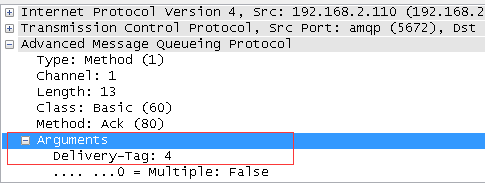


如果之前没有调用ConfirmSelect方法要求Channel使用ack，则调用这些操作会出错

* 报文



确认消息详细



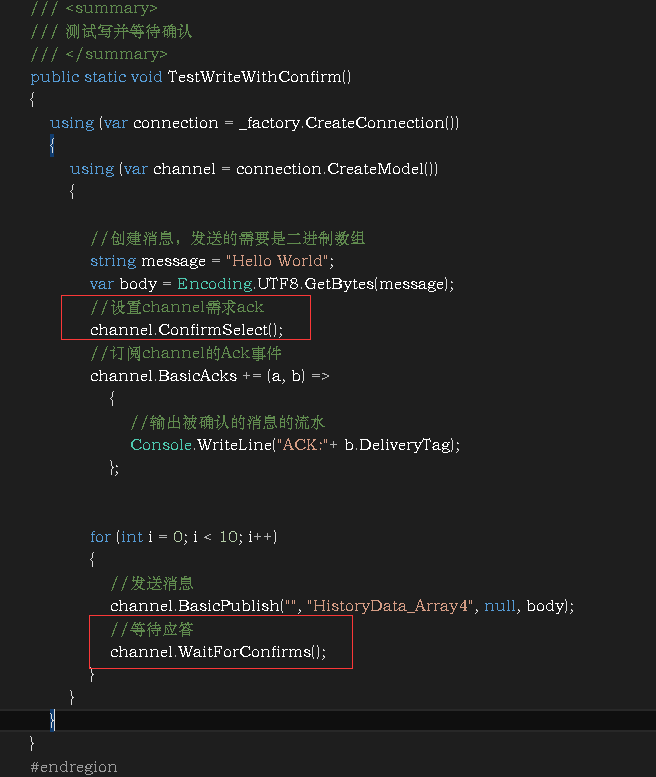
确认消息是针对Delivery-Tag的。Delivery-Tag由BasicPublish方法在发送消息时底层赋值，从1开始递增，值的有效性和对应Channel的生命周期相同。即如果新建一个Channel，这个值从新从1开始递增,以此来将Ack与发送出去的数据进行一一对应。

或者使用更准确的方式，IModel的NextPublishSeqNo属性



这个属性能够获取到下一次调用Publish类方法，发送数据时使用的DeliveryTag

使用代码示例：



相关介绍：

<http://rianjs.net/2013/12/publisher-confirms-with-rabbitmq-and-c-sharp>

<http://www.jianshu.com/p/4112d78a8753>

<http://www.jianshu.com/p/6579e48d18ae>

### 从服务读数据

从服务器读取数据分为两种方式：

一种是主动从MQ服务器获取



一种是对MQ服务器进行订阅，被动的等待MQ服务器推送数据。



#### 订阅MQ数据

* API介绍

需要进行两个操作，首先是指明预处理的数据量，然后是在通道上绑定消费者并声明是否自动应答，API：



声明预处理数据量，也就是在Customer一旦被绑定到指定的Channel，MQ就会立刻推送prefetchCount个数据给Customrer



进行Customer的绑定

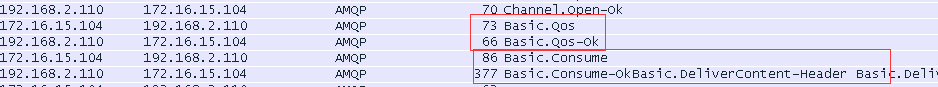
注意：如果noAck给的是true，在这个通道上的消息，不需要发进行消息的ACK，即MQ向这个Customer推送消息的时候，不用等待ACK应答，这会导致队列一旦有数据，就会向Customer推送。

* 状况1：
  + 代码



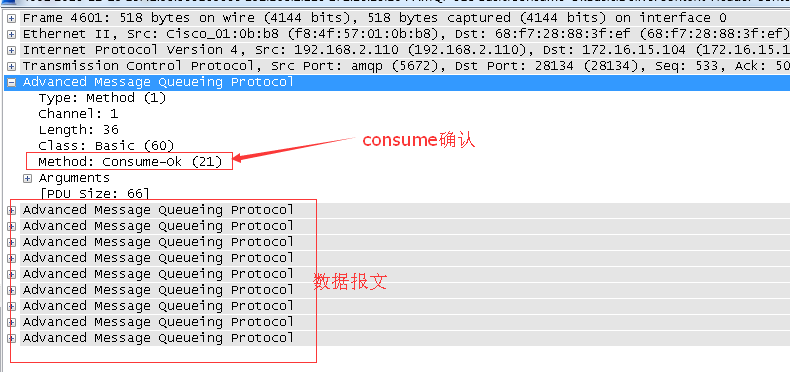
第一步表示预处理数据量是3个，第二步表示绑定消费者，并且这个消费者需要ACK

* + 报文

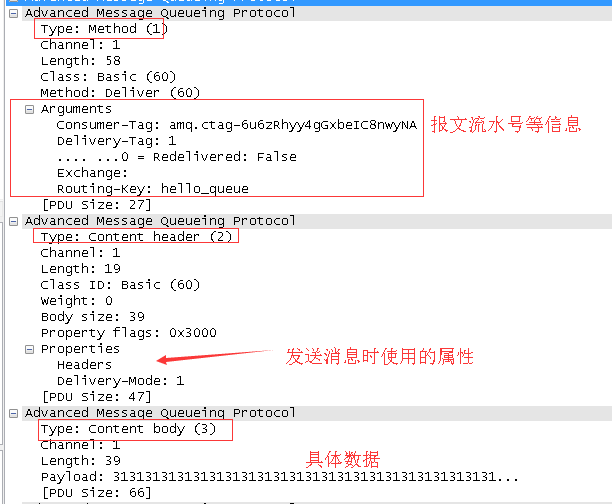


在ChannelOpen了之后，分别进行了2次的数据交互，分别由BasicQos和BasicConsume操作触发。

注意看最后一次 Basic.Cusume-ok的报文中，除了对Cusume的确认外，还携带了一定数量的数据报文：



数据报文格式是每3包是一个完整数据，分别是：Method，Header和Body



如果报文的消息是空，那么第三包数据 Content body 可能没有

* 状况2：
  + 代码

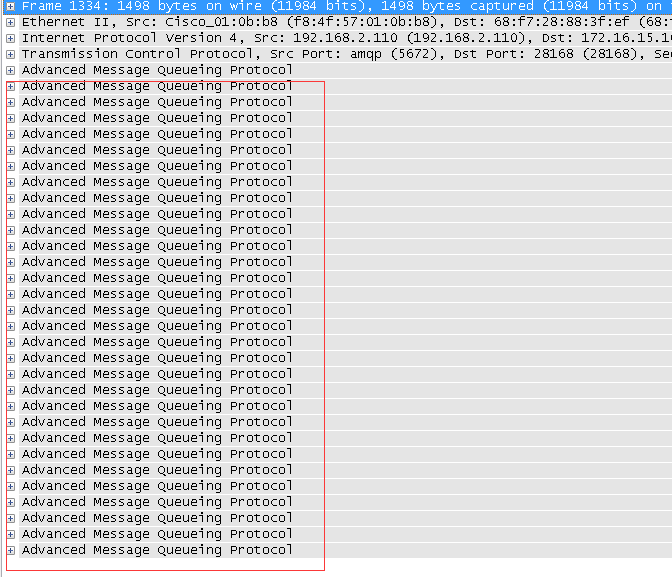


表示这个消费者的消息不需要ACK确认

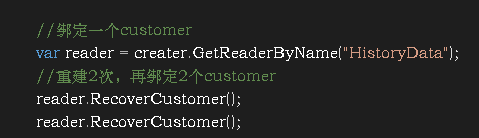
* + 报文

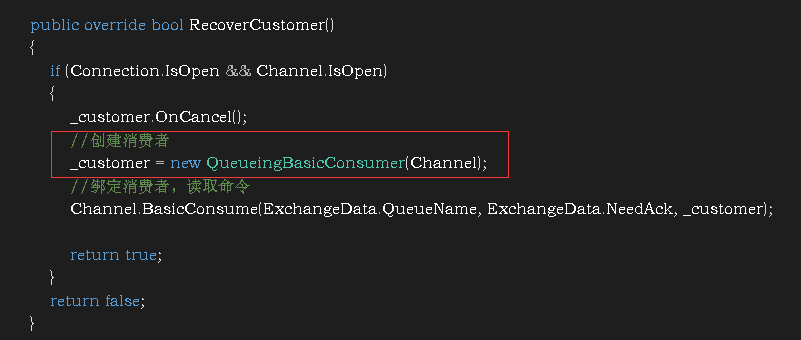


在对consume进行确认的报文中，携带了大量的信息数据，远远超过BasicQos中指明的3。并且后续还有报文



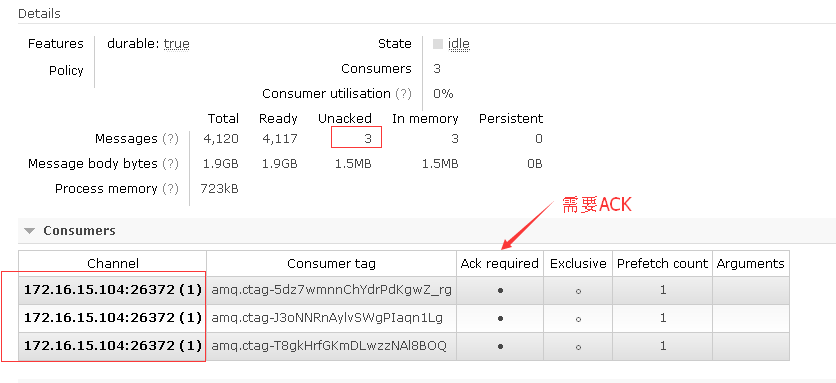
* 状况3，绑定多个custom
  + 代码





创建一个新的消费者并进行绑定，上述代码等于是在1个channel上创建了3个custom对象，并且每个对象设置的Perfetch count是1

* + 控制台



1.可以看到3个消费者都是出于Ack required状态，即在声明消费者时，使用的需要Ack的模式，也就是



2.现在MQ的unacked是3，即MQ为每个消费对象都发送了1个消息并等待应答。即MQ主动推送数据是针对Customer对象，而不是channel。但是整体的消息是针对channel的，因为消息的流水号（DeliveryTag）还是递增了，即如果从第三个对象获取数据，那么数据的DeliverTag是3。

3.也就是说，如果前2个customer对象不被使用，并且当前channel不释放，那么永远有2个unack的消息被保存（推送到了前2个customer，DeliveryTag分别是1和2），这2个消息在MQ上一直处于unacked状态。

4.因为API本身不提供customer对象的销毁函数，所以如果创建多个customer对象但是实际上只需要使用其中1个，那么可以公用它们的queue（实际不要这样使用，可能出现异常 SharedQueue closed）

##### 结论

如果是通过consum订阅了MQ服务器的消息，在consum协议报文交互文成后， MQ会立刻主动推送数据给消费者，消息会在consum的本地缓存中进行存储，等待上层处理。

在同一个channel上，如果创建了多个consum对象，那么每个consum对象都必须要被使用，否则推送到这个consum对象上的数据就会被浪费。

推送的具体数据量，以及后续怎么获得新的数据，查看 Perfetch Count相关。

#### 主动获取MQ数据

* API

主动获取MQ数据非常简单，只需要在IModel上调用一个API即可获取一个数据：

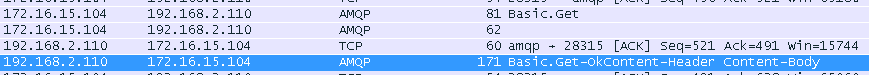


第二个参数表示是否在获取消息的同时立刻发送Ack

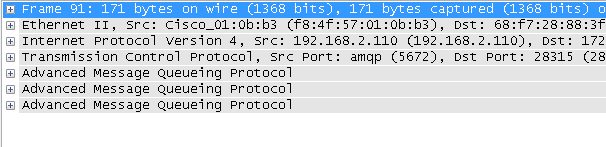
* 代码



* 报文



在Channel创建后，调用BasicGet后的报文交互，每一次MQ服务都会响应一个数据包

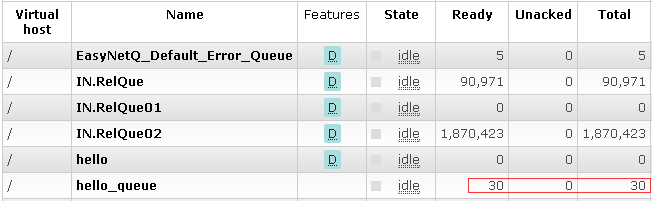


#### ACK相关

默认客户端从MQ读取的每一个消息都是需要通知MQ进行ACK的（除非在BasicConsum操作中设置关闭Ack）。

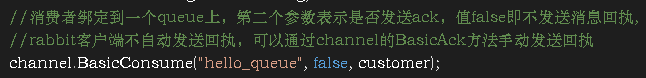
##### 不对消息进行ACK

* 发布消息



监控可以看到队列中有30个待处理的消息

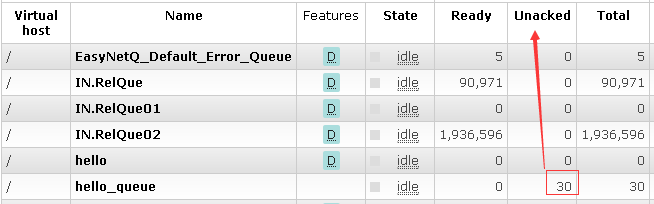
* 创建消费者，但是消费者不需要Ack



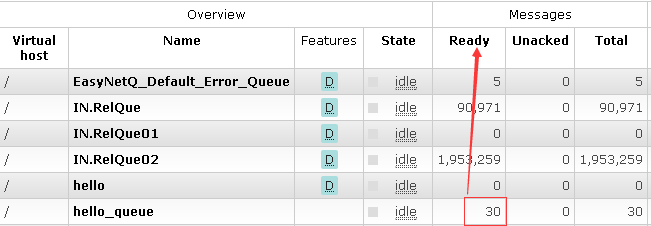
或者主动获取数据时不立刻Ack



它消费了全部消息，但是都没有发送ACK：



* 消费者停止



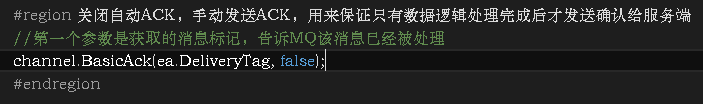
消费者（channel）关闭后，所有unacked的消息又成为了ready待处理的状态，当消费者重启后，这些消息又会被消费者重新处理。

rabbitmq如果察觉到一个客户端连接断开，那么那个客户端连接处理的未ACK消息会重新ready，等待处理。

因此需要消费者开启ACK，只有rabbitmq收到了对应消息的ACK后，才会将发送出去的消息移除；否则消息将一直存在。

经测试，关闭默认ACK，消费者对信息的获取速率更高

##### 手动ACK



IModel的BasicAck方法可以在关闭了自动ACK后，用在每个消息的逻辑处理最后调用，确保在消息被处理完成后，才通知MQ移除消息

实际应用中，这种应用较广泛

##### 手动拒绝



#### Perfetch count相关

从应用上看，这个而设置仅和订阅MQ消息的接收模式相关，这个属性在BasicConsum操作中完成，这个设置对于主动获取消息的接收模式没有作用。

如果是订阅了MQ消息，MQ服务会主动推送给消费者的，只要consum报文交互完成，数据推送立即开始。

如果在不进行Perfetch count设置的情况下，可以理解为perfetch count这个值是无限大。

就使用的现象来看：

1. 当执行了绑定消费者到queue的操作后，MQ会同时立刻发送perfetch count个消息给消费者；如果不设置Perfetch count，那么queue对一个消费者分发数据就无限制，每一次都发送大量的数据到客户端由客户端缓存，直到queue中消息耗尽
2. 如果消费者关闭了自动ACK，则此时MQ会在发送了Perfetch count个报文后停止继续发送数据给消费者；

如果开启了自动ACK，会持续发送直到queue中不再有消息。

1. 在关闭了自动ACK情况，每次消费者发送一个有效的ACK，都会从MQ获取一个消息。

因此从使用现象上看，推测在MQ服务内部是这样使用这个数据：

1. queueu会为每个消费者记录一个unack数量（即发送了消息给消费者但是消费者没有反馈ACK的计数）
2. 只要queue中有消息，queue是会主动向每个消费者分发自己的消息的，每次分发最多perfetch count-unack个消息

所以在初次绑定消费者时，MQ主动推送了perfetch count个消息，同时记录了unack=perfetch count；此后消费者发送一个有效ACK让unack数量减1，因此MQ会立刻发送perfetch count-unack（即1个）消息给消费者

##### 接收数据（设置Perfetch count）

* 代码

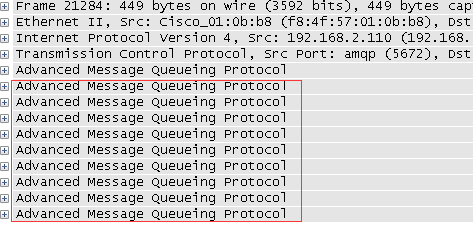




设置Perfetch count数量为3

* 报文





可以看到服务只在应答的同时发送了9个数据项（共3个消息），没有更多的数据被发送到客户端。

此后每次发送确认，都可以从服务获取一个消息：

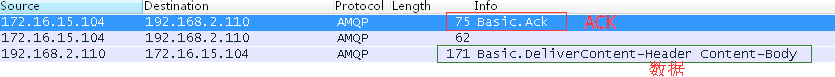
* 代码



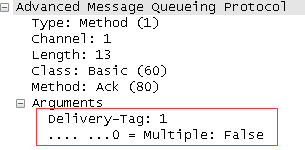


第一个参数是消息的标记，第二个参数作用未知（设置true和false使用没有区别）

* 报文



* ACK：



只包含一个method项，method是ack，红框标记的就是BasicAck方法的两个个参数。

* 数据包



服务回复了一包数据，也是method，header，body三个项

##### 关于ACK

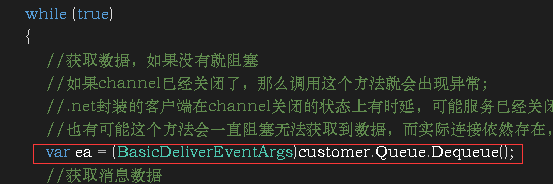
* 代码：



设置消费者channel每次只接受一个包



设置消费者的自动ack为false，即不会自动应答



循环读取消息，但是实际上只有第一次调用能够成功，后面的调用会被阻塞，因为消费了消息后，本地没有消息缓存

## 远程过程调用

远程过程调用就是要求消费者在消费完成之后，针对消费的消息发送一个确认给生产者，这个确认不是通信级别的MQ的ACK，而是逻辑级别的确认

MQ的Ack确认可以由MQ自己完成，但是远程过程调用的应答中，所有的Queue的绑定，应答的发送，消息ID的获取都需要上层逻辑手动完成。

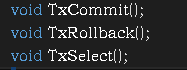
## 事务

API介绍：

定义在IModel接口上



方法



分别是提交，回滚和开始

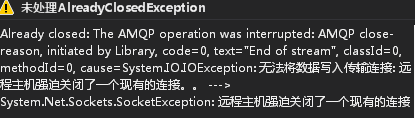
## 使用RabbitMQ.Client.dll注释事项

* 客户端和mqserver的每一个连接只能用于一个用途，例如一个连接进行了数据的写，此后就只能继续做写操作，而不能做读。可以采用每次操作都创建一个新channel的方式（存疑）
* 客户端和mqserver的连接断开后无明显症状，包括所有的操作调用都不会发生异常，但是实际却不起作用。可能是客户端API封装问题。

例如，打断点调试一段时间后，抓包发现客户端主动断开了与MQ的连接：



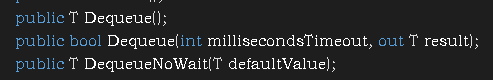
此后再调用MQ的方法就会抛出异常



* Mqserver默认对每个连接的心跳是10分钟，如果10分钟连接上没有发生数据交互，那么连接就回收
* .net客户端对于MQ连接的最大问题是状态更新不及时，一个连接即使已经被销毁了，客户端可能也不会有任何提示，调用也不会出现异常。所以解决办法是自己绑定一个心跳queue，隔一段时间就自己给自己发送一个心跳数据
* 当使用ExchangeDeclare或QueueDeclare函数去声明一个队列时，如果队列已经存在，则不会有任何操作；但是如果使用的参数（例如Exchange的Type）和现有的不符，就会报错

### 使用rabbitmq获取数据

Rabbitmq读取数据的方式有3种：



第一种方式在没有收到数据时会持续阻塞

第二种在收到数据时，如果没有数据会最多等待指定的时间，如果没有就获取默认值并返回false

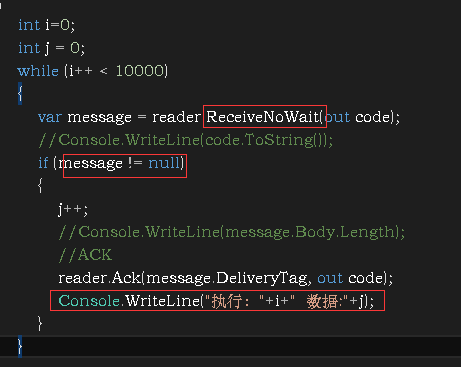
第三种在收到数据时，不会有任何阻塞，如果没有数据就返回null。

实际使用中，

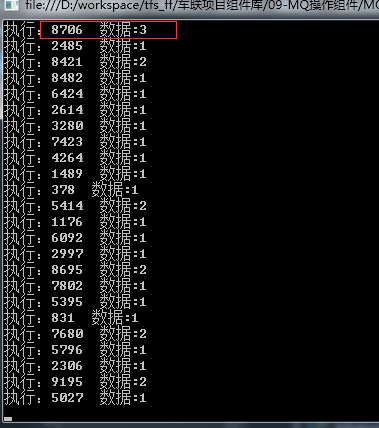
一．在非自动ACK的情况下

发现：

1.如果是使用的第三种无阻塞接收数据方式，那么有很大可能会获取到null数据，因为总是会需要等待信息被MQ服务推送到本地缓存；在本地缓存没有数据情况下（即消息在网络传输过程中），得到的都是null。



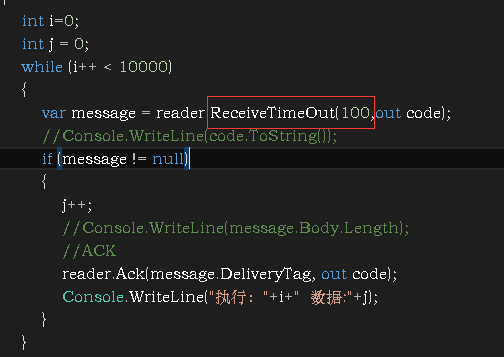
结果：

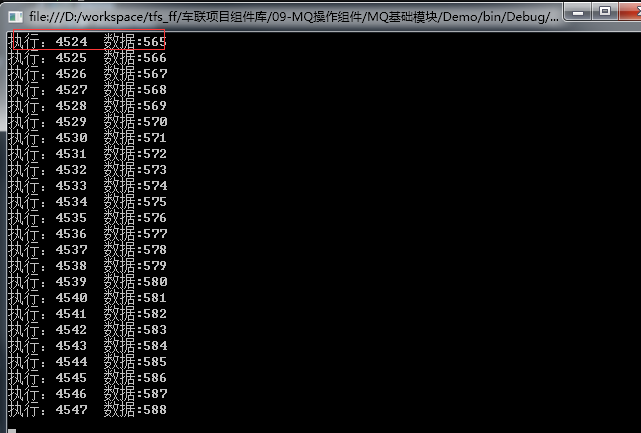


即执行8000多次，可能只有3次能得到数据。

2.使用第二种方式，阻塞一段时间：

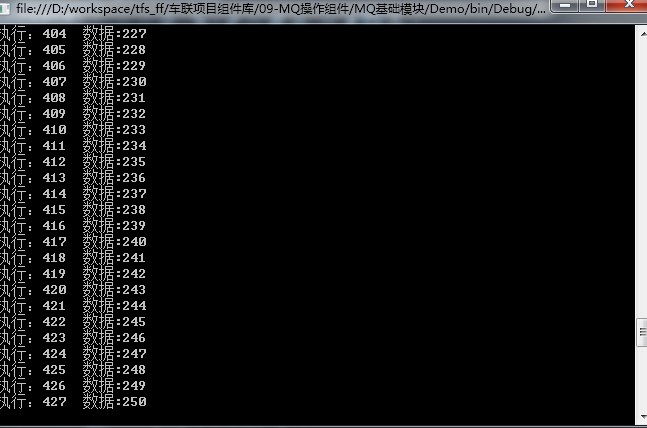
等待100毫秒





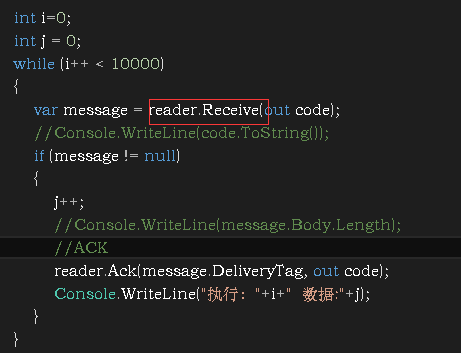
从日志可以看到执行过程时间变长，但是取得数据的概率变大。

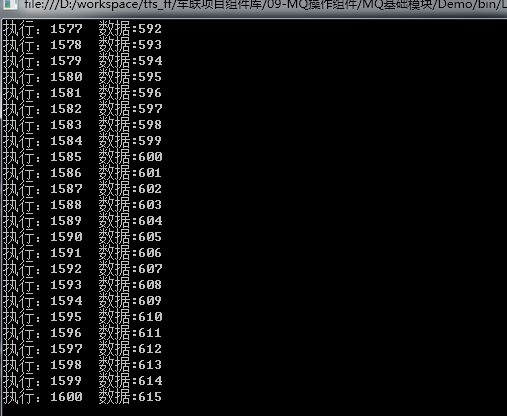
等待3000毫秒



执行的时间会花费更长，但是获取数据的概率会更大。

1. 如果使用第一种方式，阻塞等待数据

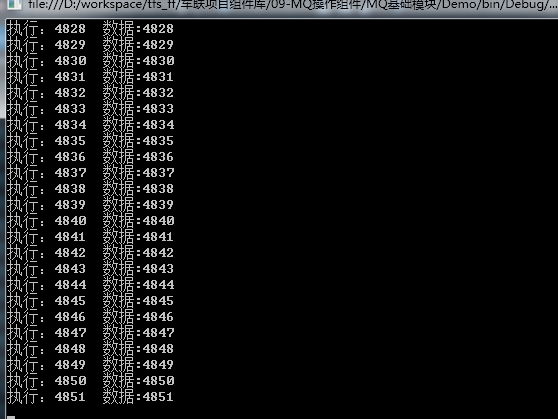




也有一定的概率获取到null对象。

二．如果是采用自动ACK的模式

使用任何一种获取数据的方式，都会很快获取到数据：



数据获取非常迅速而且总是能够获取到数据（但是通过实际测试，在数据量非常大的时候，可能会出现获取到的总是null的情况）

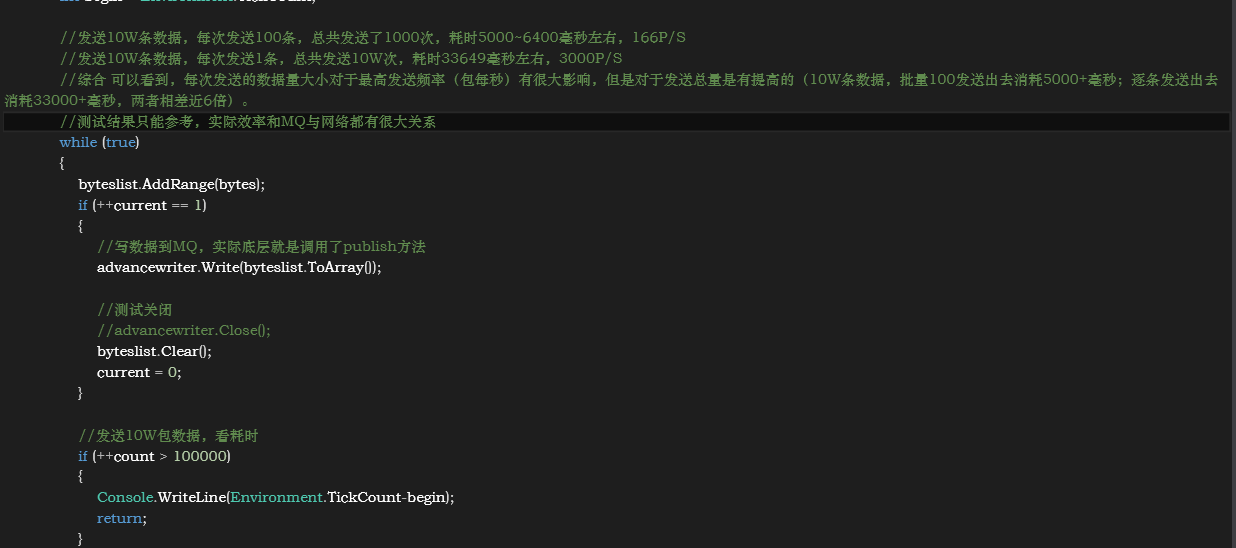
结论：

对于非自动ACK的情况，比较实用的是使用超时获取数据的方式，因为可以给足够的时间让数据从MQ传送给客户端，获取null对象的概率相对较低。

对于自动ACK的情况，在MQ有数据时，客户端总是能够获取到有效数据，但因为对MQ底层的不了解，猜测可能是本身对.NET的类库支持不够好。所以导致各种BUG，例如上面说的得到的数据总是null的情况。

### 数据的发送效率

MQ发送数据时，可以批量数据使用一个publish发送，提高效率：



# 其他

## 相关介绍

<http://www.diggerplus.org/archives/3110>

## 效率

<http://www.diggerplus.org/archives/3136>

## 第三方交换模板 Consistent Hash Exchange

### 作用

每一个Exchange有一个调度算法，算法根据消息的Routing key来进行算法计算，然后将消息均衡分发到Exchange上的所有的队列上。

能达到的效果：

1. 每个消费队列上都会有消息，消息量大小由消费队列指定权重（实际也就是binding key），当队列添加或删除后会自动重新分配。
2. 保证同一性，任何具有相同routing key的消息总是会被发布到同一个队列上，只要队列情况不发生变化，那么这个情况会一直保持；除非队列添加或 删除，导致了消息重新分配。

需要注意的是，因为这个模板的Exchange使用时可以不显式声明Queue，Queue是有任何一个消费者连接到Exchange上后，Exchange自动生成并绑定的（队列名称随机，每个消费者一个Queue）。

因此如果在没有任何消费者的情况下，所有发送到这个Exchange上的消息都会丢失（因为没有Queue）

也可以显式声明队列名称并绑定。

# 异常相关

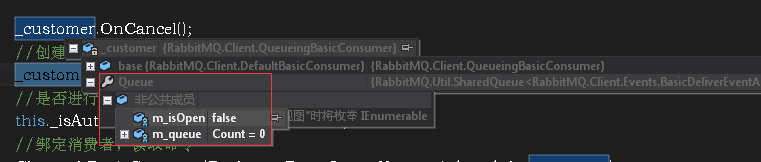
对于MQ的API，如果任何和网络通信相关的操作调用发生了异常，都有可能导致连接的关闭。

## SharedQueue Closed

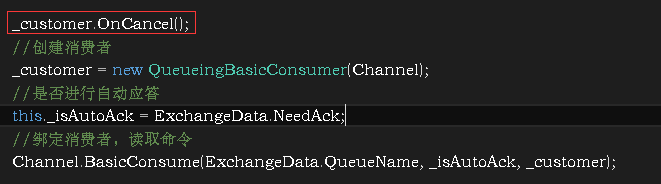
这个异常是在调试时出现，在接收数据时程序如果长时间进行逻辑操作（例如断点），而导致一定时间没有继续从MQ接收数据，那么在之后几次接收时，会在QueueingBasicConsumer对象的Receive方法报出。



查看QueueingBasicConsumer对象的状态：

 看到Queue对象状态是false，count是0。这两个状态都是protected的，因此无法从外部获取这些状态并进行判断，只能通过try catch捕获异常

之后如果channel是工作状态，那么在channel上重新绑定新的QueueingBasicConsumer对象即可：



但是这样一来，如果是在需要ACK的情况下，原来的在这个customer对象上的queue中的数据就会被锁死，MQ会一直记录这些数据是unacked的，直到channel被销毁才恢复。

这个异常可能是由于和MQ的Channel销毁导致的

## 取得的数据总是null

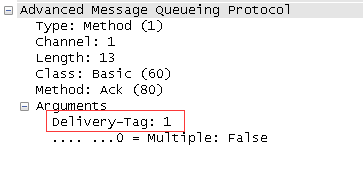
从rabbitmq获取的数据总是null，此时重新恢复customer或者重建channel

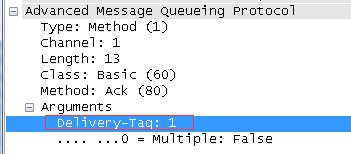
## 重复Ack

对于同一个报文，如果对它的DeliveryTag进行了多次ACK，那么就会出错，此时可能会导致channel的关闭。



两次ACK：

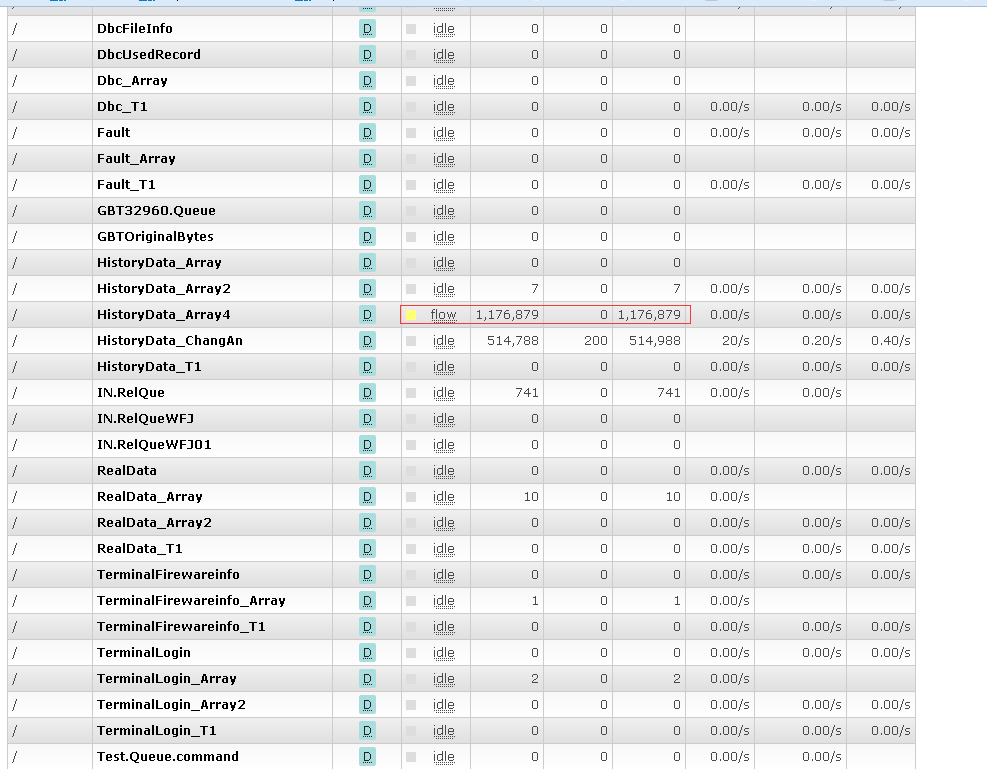




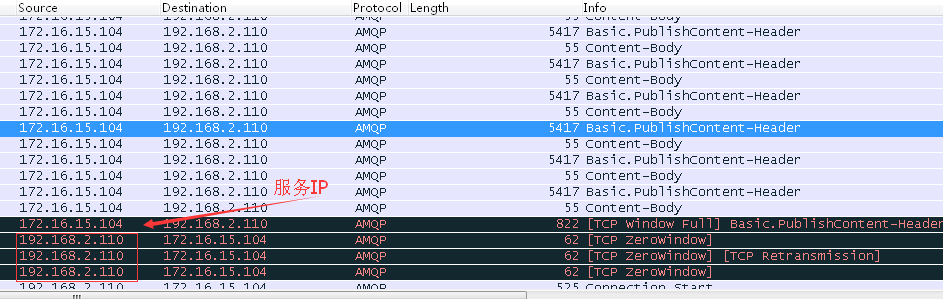
Tag都是1，之后Channel关闭了（由消费者发起）

## RabbitMQ中数据堆积

RabbitMQ中一个管道的数据不能堆积太多，否则会导致向这个管道写数据时效率降低，甚至服务端TCP的窗口大小为0导致连接断开



数据堆积在MQ内存中。



服务端提示滑动窗口满

# 经验相关