RabbitMQ流量控制机制简单分析

在RabbitMQ中，消息可能被存储在多个不同的队列，消息越早被消费，那么消息经过的队列层次越少，则平均每个消息处理的开销就越小。但若接收消息的速率过快，MQ来不及处理，这些消息就可能进入很深层次的队列，大大增加平均每个消息的处理开销，进一步使得处理新消息和发送旧消息的能力减弱，更多的消息会进入很深的队列，循环往复，整个系统的性能就会极大的降低。另外若接收消息的速率过快还会实现某些进程的mailbox过大，可能会产生很严重的后果。为此，RabbitMQ设计了一套流控机制，本文从以下三个方面去阐述该流控机制是如何工作的。

## 如何开关闸门

RabbitMQ使用TCP长连接进行通讯，接收数据的起点进程为rabbit\_reader。首先分析它的接收loop

|  |
| --- |
| recvloop(Deb, State = #v1{connection\_state = blocked}) ->  mainloop(Deb, State);  recvloop(Deb, State = #v1{sock = Sock, recv\_len = RecvLen, buf\_len = BufLen})  when BufLen < RecvLen ->  ok = rabbit\_net:setopts(Sock, [{active, once}]),  mainloop(Deb, State#v1{pending\_recv = true});  recvloop(Deb, State = #v1{recv\_len = RecvLen, buf = Buf, buf\_len = BufLen}) ->  {Data, Rest} = split\_binary(case Buf of  [B] -> B;  \_ -> list\_to\_binary(lists:reverse(Buf))  end, RecvLen),  recvloop(Deb, handle\_input(State#v1.callback, Data,  State#v1{buf = [Rest],  buf\_len = BufLen - RecvLen})). |

从上面代码可以看出，rabbit\_reader每接收到一个包，就设置套接字属性为{active, onece}，若当前连接被blocked时则不设置{active,once}，这个接收进程就阻塞在receive方法上。通过这种方式来实现闸门的开关。

## 何时关闭闸门

RabbitMQ是用erlang/OTP开发的，一个消息从被接收到被发送给订阅者，必然要在多个进程间的转发，从接收到被消费，一个消息所走过的所有进程自然形成一条消息链，RabbitMQ通过监控这条链上每个节点“mailbox”中未被接收的消息数量，决定何时关闭闸门。实现机制如下所述：

* {{credit\_from,pid}, value}
* {{credit\_to,B}, value}
* {{credit\_from,C}, value}
* {{credit\_to,A}, value}
* {{credit\_from,B}, value}
* {{credit\_to, pid}, value}

如图所示，进程A、B、C连成一条消息链，每个进程字典中有一对关于收发消息的credit值，以进程B为例，｛｛credit\_from, C｝, Value｝，表示能发多少条消息给C，每发一条消息该值减1，当为0时，本进程阻塞住不再往下游进程发消息也不再接收上游的消息；｛｛credit\_to, A｝, Value｝表示再接收多少个消息就向上游进程发增加credit值的消息｛bump\_credit, { self(), Quantity}｝,在上游进程接收到该消息后，就增加｛credit\_from, pid｝值，这样上游进程就能持续发消息。但当上游发送速率高于下游接收速率，credit值会逐渐被耗光这时进程就会被阻塞，阻塞的情况会一直传递到最上游

Rabbit\_reader，这时rabbit\_reader就关闭闸门。

## 何时开启闸门

当上游进程收到来自下游进程的bump\_credit消息时，若此时上游进程处于block状态则解除block状态，开始接收更上游进程的消息，一个个的传导最终能够解除rabbit\_reader的block状态。