

# 分布式消息队列

NSQ原理与实现

骆爽 (imluoshuang@gmail.com)

# 为什么使用消息队列

- \* 解耦

- \* 扩展性：生产者、消费者自由增减
- \* 避免单点故障

- \* 冗余

- \* 插入-获取-删除

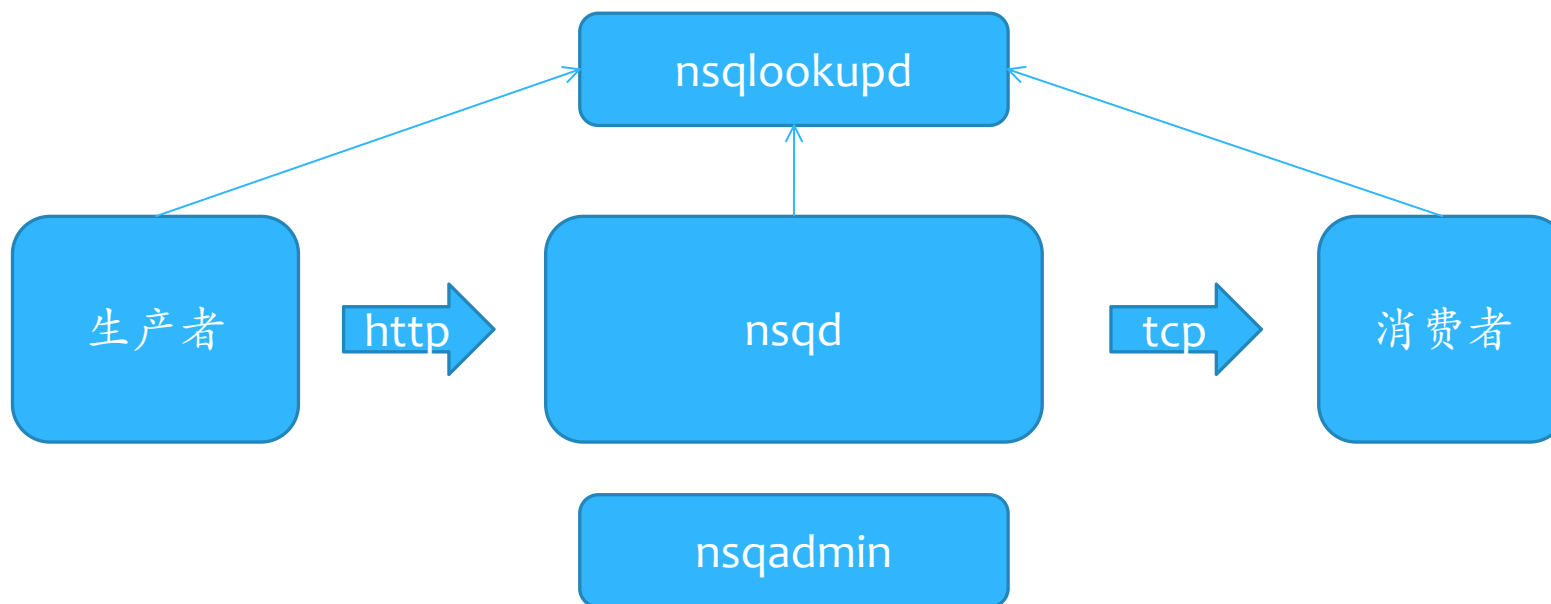
- \* 异步通信

- \* 缓冲

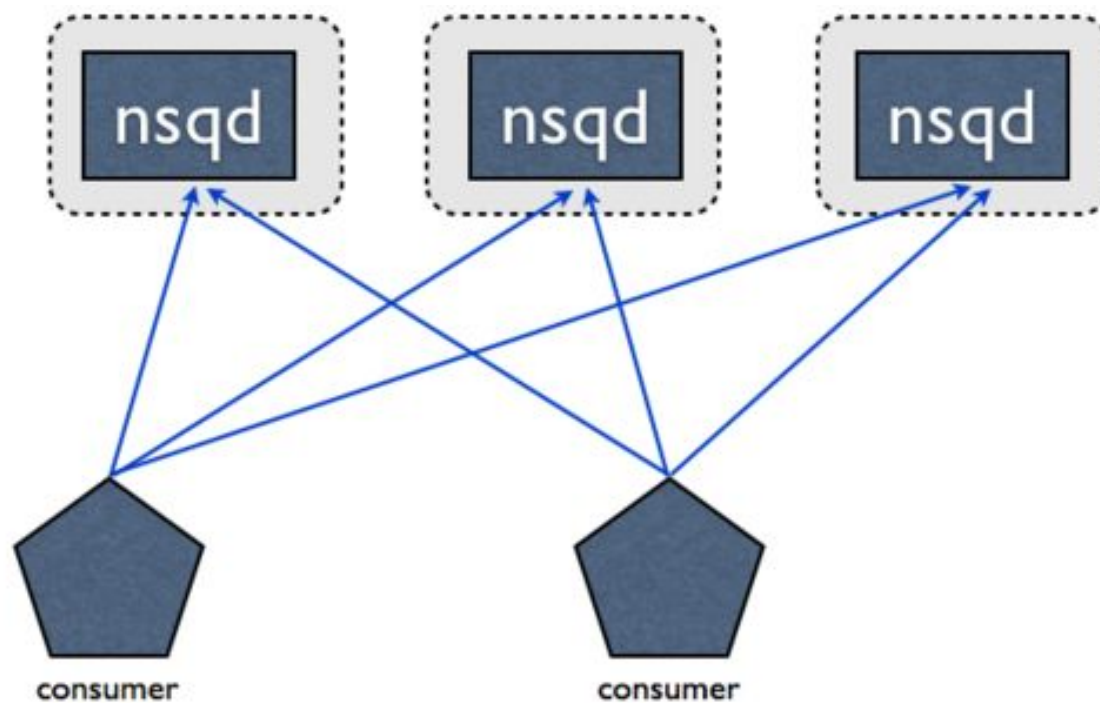
- \* 频率性能统计

# NSQ是什么

- \* Go语言实现
- \* 完全解耦的消息队列



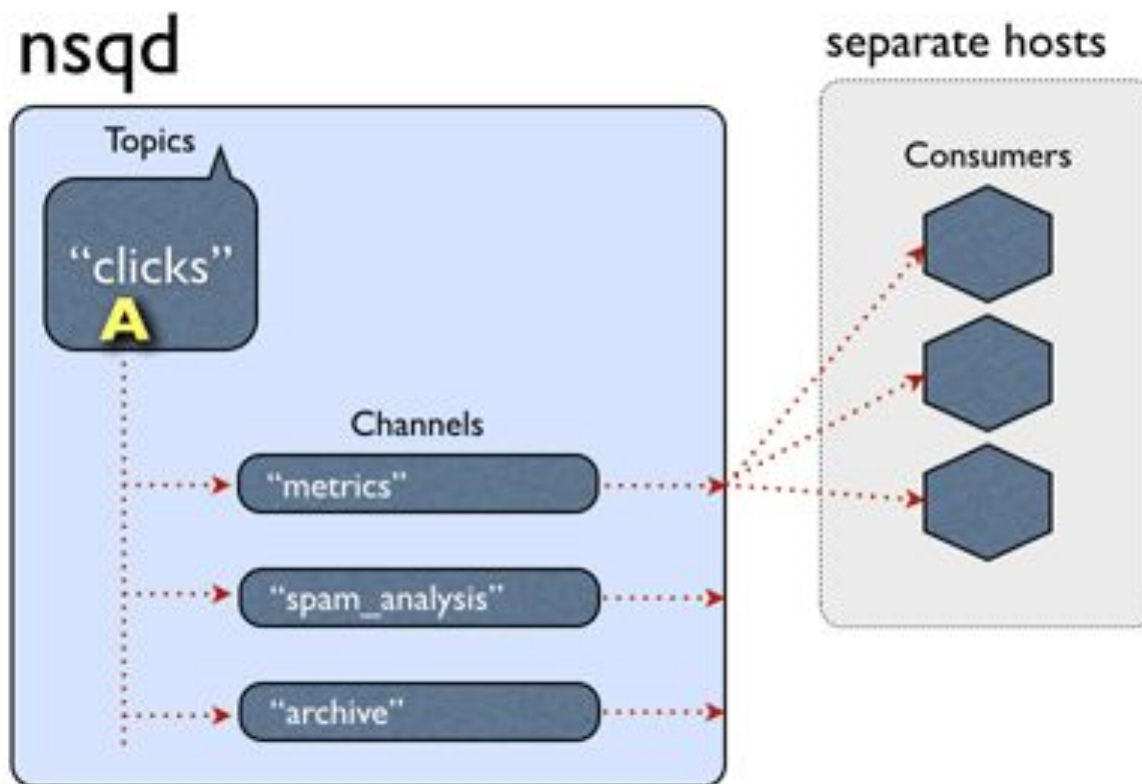
# 避免单点故障



# Go语言的管道

- \* 不要通过共享内存来通信，而应该通过通信来共享内存
- \* 管道：
  - \* `var dbl_chan chan int = make(chan int)`
  - \* `var sgl_chan <- chan string = make(<-chan string)`
  - \* `var buf_chan chan int64 = make(chan int64, 2)`

# NSQD



# 生产者 -> nsqd

- \* HTTP/HTTPS协议，使用POST提交
- \* 发布消息
  - \*\$ curl -d "<msg>" http://127.0.0.1:4151/pub?topic=my\_topic

# nsqd: topic -> channel

- \* 每个topic可以有多个channel，每个channel都会收到topic消息的一个副本
- \* Channel实际对应着消息处理的一种方式
  - \* 例如收到一条监控数据，一方面把它落地到数据库中永久保存，一方面写入缓存供提高查询效率



# nsqd -> 消费者

- \* 消费者使用TCP协议连接到nsqd，每个消费线程仅订阅一个topic的一个channel
- \* 一个消费者进程可以包含多个handler线程，从而单个进程可以处理不同的topic和不同的channel

TIY

# 启动nsqd 消息队列



# 启动nsqd 消息队列

```
[devop@gitlab ~]$ ll
total 4
drwxrwxr-x. 2 devop devop 4096 Sep 25 15:28 nsqdata
[devop@gitlab ~]$ nsqd -data-path=nsqdata/
[nsqd] 2014/09/25 15:28:21.268673 nsqd v0.3.0-alpha (built w/go1.3)
[nsqd] 2014/09/25 15:28:21.268734 ID: 161
[nsqd] 2014/09/25 15:28:21.268759 NSQ: persisting topic/channel metadata to nsqdata/nsqd.161.dat
[nsqd] 2014/09/25 15:28:21.297883 TCP: listening on [::]:4150
[nsqd] 2014/09/25 15:28:21.297917 HTTP: listening on [::]:4151
```

# 启动nsqadmin, 用来观察



# 启动nsqadmin, 用来观察

```
[devop@gitlab ~]$ nsqadmin -nsqd-http-address=127.0.0.1:4151  
[nsqadmin] 2014/09/25 15:29:51.373194 nsqlookupd v0.3.0-alpha (built w/go1.3)  
[nsqadmin] 2014/09/25 15:29:51.390654 HTTP: listening on [::]:4171  
█
```

# 通过HTTP生产一条消息



# 通过HTTP生产一条消息

```
[devop@gitlab ~]$ curl -d "hello world 1" "http://127.0.0.1:4151/pub?topic=test"  
OK[devop@gitlab ~]$
```

\* 返回了OK，表示nsqd收到了这条数据



# 不启动消费者，观察一下消息队列

## Topics

Topic

test

```
[devop@gitlab ~]$ ll
total 4
drwxrwxr-x. 2 devop devop 4096 Sep 25 15:28 nsqdata
[devop@gitlab ~]$ nsqd -data-path=nsqdata/
[nsqd] 2014/09/25 15:28:21.268673 nsqd v0.3.0-alpha (built w/go1.3)
[nsqd] 2014/09/25 15:28:21.268734 ID: 161
[nsqd] 2014/09/25 15:28:21.268759 NSQ: persisting topic/channel metadata to nsqdata/nsqd.161.dat
[nsqd] 2014/09/25 15:28:21.297883 TCP: listening on [::]:4150
[nsqd] 2014/09/25 15:28:21.297917 HTTP: listening on [::]:4151
[nsqd] 2014/09/25 15:32:14.345117 TOPIC(test): created
[nsqd] 2014/09/25 15:32:14.345166 NSQ: persisting topic/channel metadata to nsqdata/nsqd.161.dat
```

Streams / test

Topic: **test**

Empty Queue

Delete Topic

Pause Topic


### Topic Message Queue

NSQd Host	Depth	Memory + Disk	Messages	Channels
<span>x</span> 127.0.0.1:4151	1	1 + 0	1	0
Total:	1	1 + 0	1	0

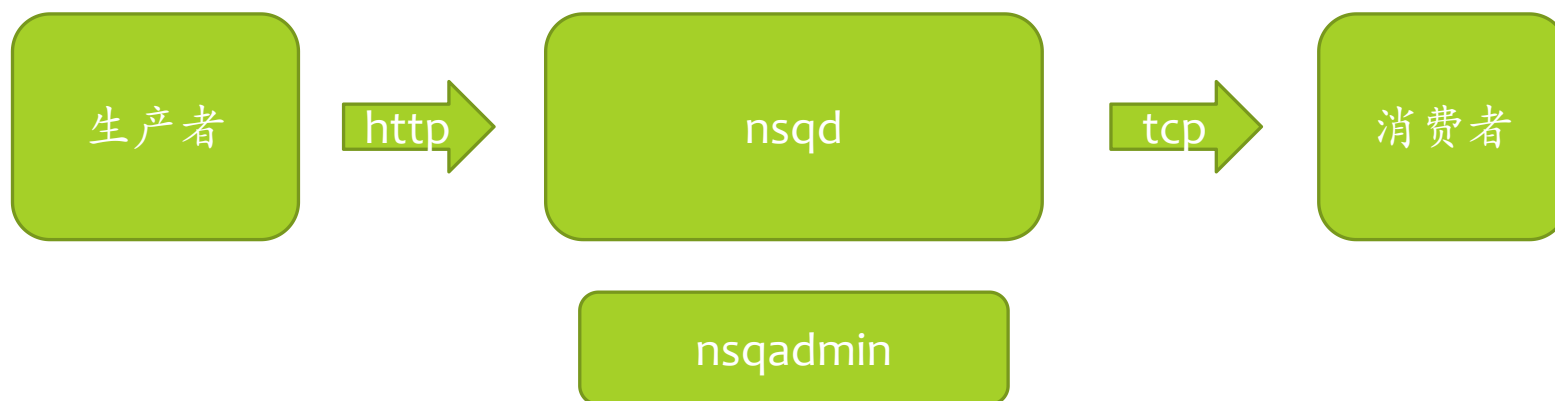
### Channel Message Queues

#### Notice

No channels exist for this topic.  
Messages will queue at the topic until a channel is created.

- 
- \* 我们看到topic自动建立了，内存队列中有1条数据
  - \* 现在这个topic下面没有任何的channel建立
  - \* 上图还可以看出，多个nsqd的实例可以接收同一个topic，这里我们仅有一个nsqd的服务端

# 启动消费者开始消费消息



# nsq\_tail

- \* 这是nsq官方给出的一个消费者实例程序，可以从消息队列指定的topic中消费消息。
- \* 我们必须提供channel的名称，一个消费者线程必须从topic下面具体的channel中消费消息

```
[devop@gitlab ~]$ nsq_tail --help
Usage of nsq_tail:
  -channel="": NSQ channel
  -consumer-opt=: option to passthrough to nsq.Consumer (may be given multiple times)
  -lookupd-http-address=: lookupd HTTP address (may be given multiple times)
  -max-in-flight=200: max number of messages to allow in flight
  -n=0: total messages to show (will wait if starved)
  -nsqd-tcp-address=: nsqd TCP address (may be given multiple times)
  -reader-opt=: (deprecated) use --consumer-opt
  -topic="": NSQ topic
  -version=false: print version string
[devop@gitlab ~]$
```

# 启动消费者开始消费消息

- \* 启动nsq\_tail，指定了刚才的topic名称“test”
- \* channel名称为“tail”
- \* 可以看到立刻收到了刚才发送的消息

```
[devop@gitlab ~]$ nsq_tail -nsqd-tcp-address=127.0.0.1:4150 -topic=test -channel=tail
2014/09/25 15:42:40 INF    1 [test/tail] (127.0.0.1:4150) connecting to nsqd
hello world 1
```

# 再看nsqd

- \* 消费者作为client连接上了，首先IDENTIFY了自身
- \* 根据我们提供的channel名称，topic发现channel不存在，因此新建了tail
- \* 在topic/channel有变动时，nsqd的数据会自动持久化

```
[nsqd] 2014/09/25 15:32:14.345166 NSQ: persisting topic/channel metadata to nsqdata/nsqd.161
[nsqd] 2014/09/25 15:42:40.847546 TCP: new client(127.0.0.1:34466)
[nsqd] 2014/09/25 15:42:40.848058 CLIENT(127.0.0.1:34466): desired protocol magic ' V2'
[nsqd] 2014/09/25 15:42:40.848556 [127.0.0.1:34466] IDENTIFY: {ShortId:gitlab LongId:gitlab
te:false DeflateLevel:6 Snappy:false SampleRate:0 UserAgent:nsq_tail/0.3.0-alpha go-nsqd/1.0.
[nsqd] 2014/09/25 15:42:40.849266 TOPIC(test): new channel(tail)
[nsqd] 2014/09/25 15:42:40.849350 NSQ: persisting topic/channel metadata to nsqdata/nsqd.161
```



# 从nsqadmin观察统计信息

Streams / test

Topic: **test**

[Empty Queue](#) [Delete Topic](#) [Pause Topic](#)

### Topic Message Queue

NSQd Host	Depth	Memory + Disk	Messages	Channels
<span>✗</span> 127.0.0.1:4151	0	0 + 0	1	1
Total:	0	0 + 0	1	1

### Channel Message Queues

Channel	Depth	Memory + Disk	In-Flight	Deferred	Requeued	Timed Out	Messages	Connections
<a href="#">tail</a>	0	0 + 0	0	0	0	0	1	1

- \* 我们的channel在topic下出现了，统计信息表明它处理了一个信息，有一个客户端连接在上面



# 从nsqadmin观察统计信息

Streams / test / tail

Topic: **test**

Channel: **tail**

Empty Queue

Delete Channel

Pause Channel

## Channel

	Message Queues				Statistics			
NSQd Host	Depth	Memory + Disk	In-Flight	Deferred	Requeued	Timed Out	Messages	Connections
<a href="#">127.0.0.1:4151</a>	0	0 + 0	0	0	0	0	1	1
Total:	0	0 + 0	0	0	0	0	1	1

## Client Connections

Client Host	Protocol	Attributes	NSQd Host	In-Flight	Ready Count	Finished	Requeued	Messages	Connected
gitlab:34466	V2 (nsq_tail/0.3.0-alpha go-nsq/1.0.1-alpha)		<a href="#">127.0.0.1:4151</a>	0	200	1	0	1	6m43s

## 再来一个消费者，新增一个channel

```
OK[devop@gitlab ~]$ curl -d "hello world with 2 channel" "http://127.0.0.1:4151/pub?topic=test"
OK[devop@gitlab ~]$
```

```
[devop@gitlab ~]$ nsq_tail -nsqd-tcp-address=127.0.0.1:4150 -topic=test -channel=tail
2014/09/25 15:42:40 INF    1 [test/tail] (127.0.0.1:4150) connecting to nsqd
hello world 1
hello world with 2 channel
```

```
[devop@gitlab ~]$ nsq_tail -nsqd-tcp-address=127.0.0.1:4150 -topic=test -channel=tail2
2014/09/25 17:22:22 INF    1 [test/tail2] (127.0.0.1:4150) connecting to nsqd
hello world with 2 channel
```

\* 两个消费者从两个channel独立收到了消息

## 如果两个消费者连接一个channel呢

```
OK[devop@gitlab ~]$ curl -d "hello world with 1 channel" "http://127.0.0.1:4151/pub?topic=test"
OK[devop@gitlab ~]$
```

```
[devop@gitlab ~]$ nsq_tail -nsqd-tcp-address=127.0.0.1:4150 -topic=test -channel=tail
2014/09/25 17:24:38 INF    1 [test/tail] (127.0.0.1:4150) connecting to nsqd
hello world with 1 channel
```

```
[devop@gitlab ~]$ nsq_tail -nsqd-tcp-address=127.0.0.1:4150 -topic=test -channel=tail
2014/09/25 15:42:40 INF    1 [test/tail] (127.0.0.1:4150) connecting to nsqd
hello world 1
hello world with 2 channel
```

\* 这次仅有一个消费者收到了消息

# 停止消费者，生产一条消息，停掉 nsqd，看看磁盘队列

- \* 在启动nsqd时我们指定了data-path，里面存放了：
  - \* nsqd元数据
  - \* topic拥有一个磁盘队列及其元数据
  - \* 每个topic的每个channel拥有磁盘队列及其元数据

# 看看磁盘队列

- \* 刚才我们停掉了tail2这个channel，因此tail2有两条消息被持久化，tail只有一条，从大小上可以看出来
- \* 元数据保存了每个队列的消息数目和读写位置

```
OK[devop@gitlab nsqdata]$ ll
total 24
-rw-----. 1 devop devop 143 Sep 25 17:32 nsqd.161.dat
-rw-----. 1 devop devop  10 Sep 25 17:32 test.diskqueue.meta.dat
-rw-----. 1 devop devop 107 Sep 25 17:32 test:tail2.diskqueue.000000.dat
-rw-----. 1 devop devop  12 Sep 25 17:32 test:tail2.diskqueue.meta.dat
-rw-----. 1 devop devop  51 Sep 25 17:32 test:tail.diskqueue.000000.dat
-rw-----. 1 devop devop  11 Sep 25 17:32 test:tail.diskqueue.meta.dat
[devop@gitlab nsqdata]$
```

# 一些初步的结论

- \* 消息队列nsqd分为topic和channel两个层次，前者面向生产者，后者面向消费者
- \* 一个topic可以对于多个channel，这些channel分别享有独立的消息副本
- \* 消息的生产使用极为简单的http协议
- \* 消息的消费使用复杂的tcp+私有协议，保证每个消息都会得到处理

深入内部

# 消息如何存储

- \* 消息拥有全局唯一的ID，通过GUID发生器产生。Nsqd提供一个4096缓冲的管道提供它们。
- \* 消息可能会绑定一个客户端，仅当有消费者通过channel得到这个消息时才会绑定
- \* Pri和index用于消息的处理优先级和堆排序

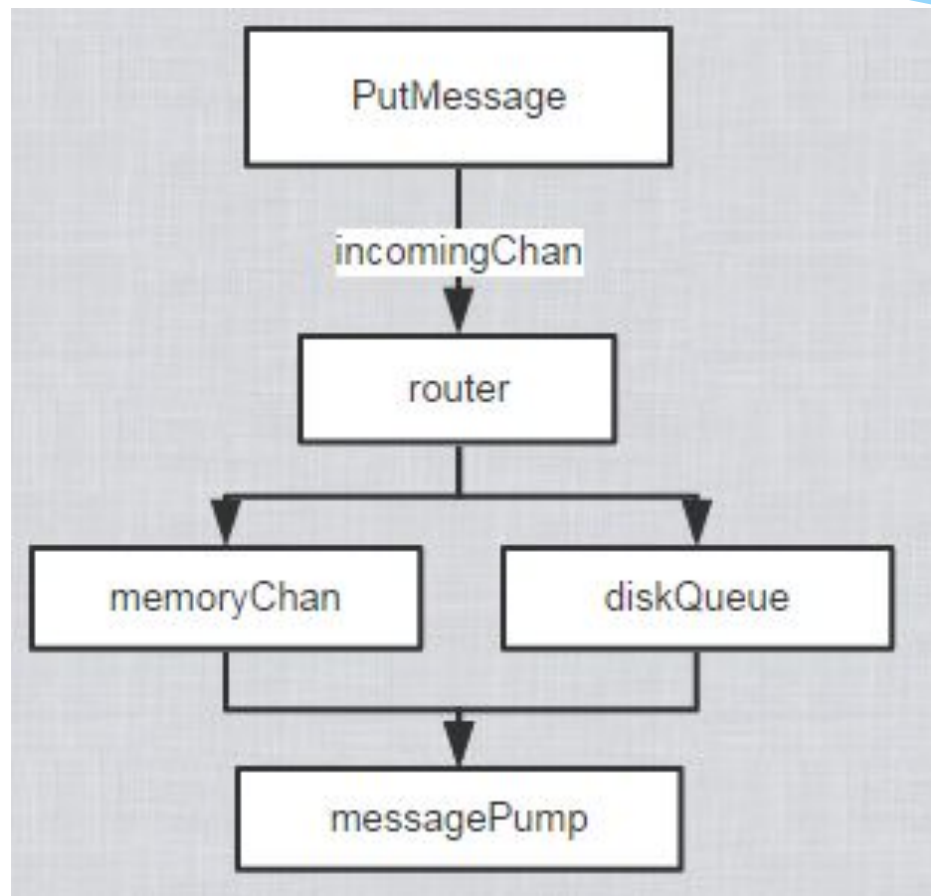
```
type Message struct {  
    ID          MessageID  
    Body        []byte  
    Timestamp   int64  
    Attempts    uint16  
  
    // for in-flight handling  
    deliveryTS  time.Time  
    clientID    int64  
    pri         int64  
    index       int  
}
```



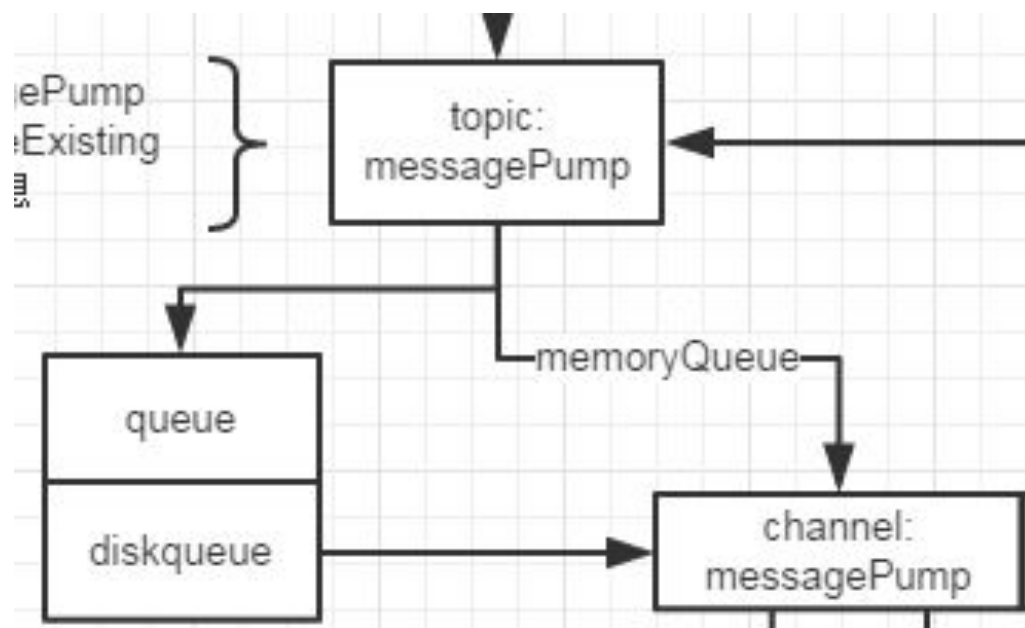
# 消息如何产生

- \* 在新建topic之后，topic对象拥有PutMessage方法来接收消息，它从http请求的body中得到消息体，从nsqd拿到GUID。
- \* 随后message被其router循环放入memoryQueue或diskQueue（当mQ满了）
- \* 对连接到topic的每个channel，topic为其准备一个消息副本，通过channel的PutMessage进行递交。
- \* 仅当一个topic没有任何channel的时候，topic才存储消息到自身的队列。

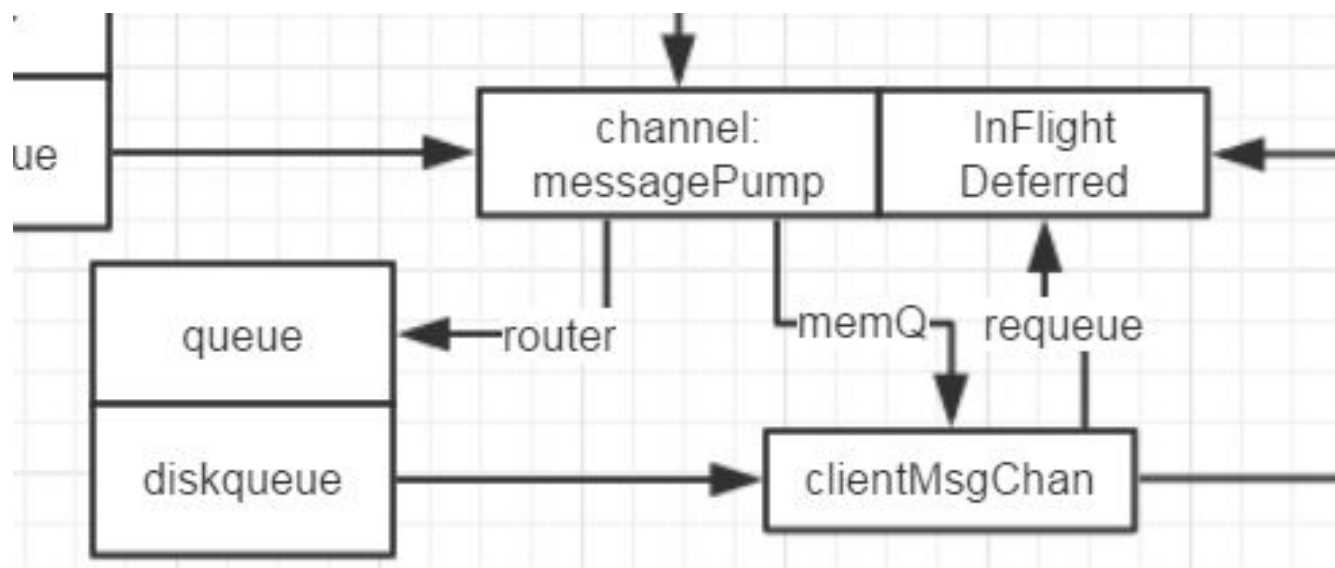
# 消息如何传递



# 消息传递之 topic



# 消息传递之 channel



# Protocol模块

- \* 处理与远端client的交互：发送消息，接收client命令
- \* 解析私有协议后发送给相应的对象处理
- \* Topic：处理PUB, MPUB
- \* Channel：处理SUB, FIN, REQ, TOUCH
- \* Client：远端client在本地的代表，实际起到认证和计数器的作用，处理IDENTIFY, AUTH, SUB, RDY, CLS

# 消息传递之 protocol

- \* messagePump:

- \* 读来自channel的clientMsgChan，通过TCP发送给远端client

- \* IOLoop:

- \* 接受来自远端的命令，发送给topic/channel/本地client对象

# 处理失败的消息处理

- \* 消费者正确消费消息后，回复FIN给nsqd
- \* 消费者处理消息出错时，回复REQ给nsqd
- \* 这两个消息都会被IOLoop转给channel进行消息队列的处理：
  - \* FIN —— 从消息字典和优先级队列中移除
  - \* REQ —— 将消息优先级降低后重新放入incoming队列
  - \* 关于优先级队列可参考《数据结构与算法》小根堆排序部分

# 磁盘队列的实现

- \* 对外的接口：
  - \* Depth() int64
  - \* ReadChan() chan byte[]
  - \* Put(data []byte) error
  - \* Close() error
  - \* Delete() error
  - \* Empty() error



# 源码阅读笔记

- \* 流程图

- \* <http://www.processon.com/view/link/541198050cf28bc1e0c6539e>

- \* 源码和笔记

- \* [https://github.com/seanluo/nsq\\_with\\_note](https://github.com/seanluo/nsq_with_note)

# 暂时未涉及之处

- \* nsqlookupd
- \* 消费者client的鉴权
- \* Go 语言装饰器、包装器设计模式的使用
- \* Go语言GC的优化处理

# References

- \* NSQD源代码: <https://github.com/bitly/nsq>
- \* NSQ官方文档: <http://nsq.io/overview/internals.html>