

# NSQ消息队列

作者：少林之巅

# 目录

1. NSQ介绍

2. NSQ应用场景

3. NSQ原理剖析

4. NSQ使用

# NSQ介绍

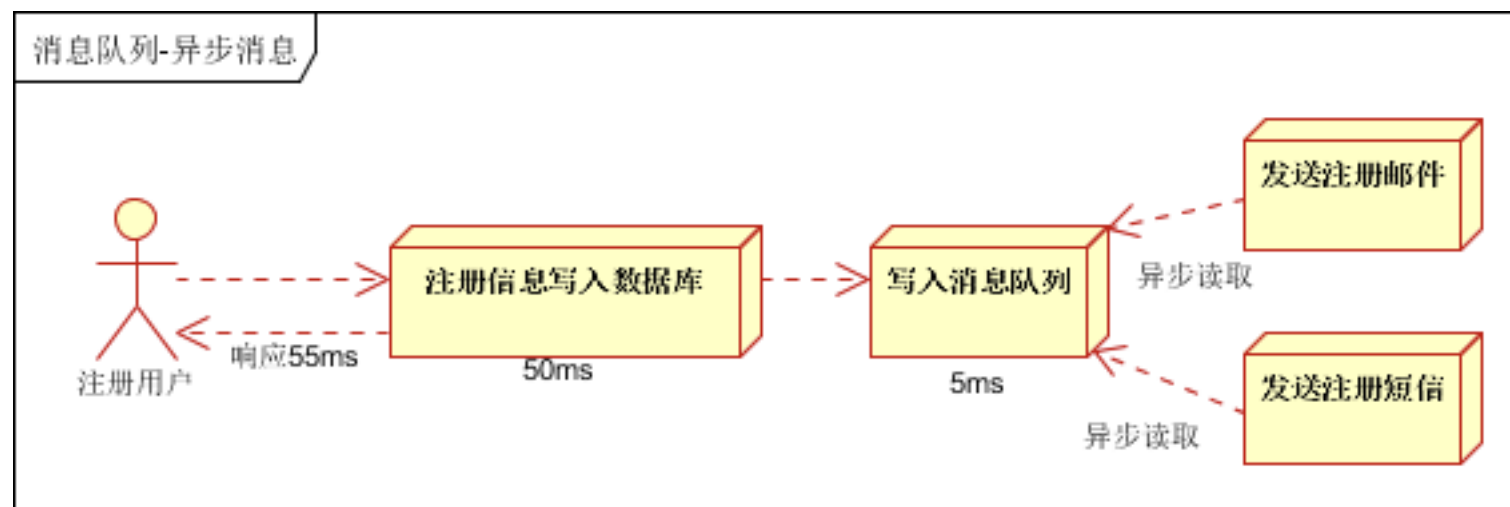
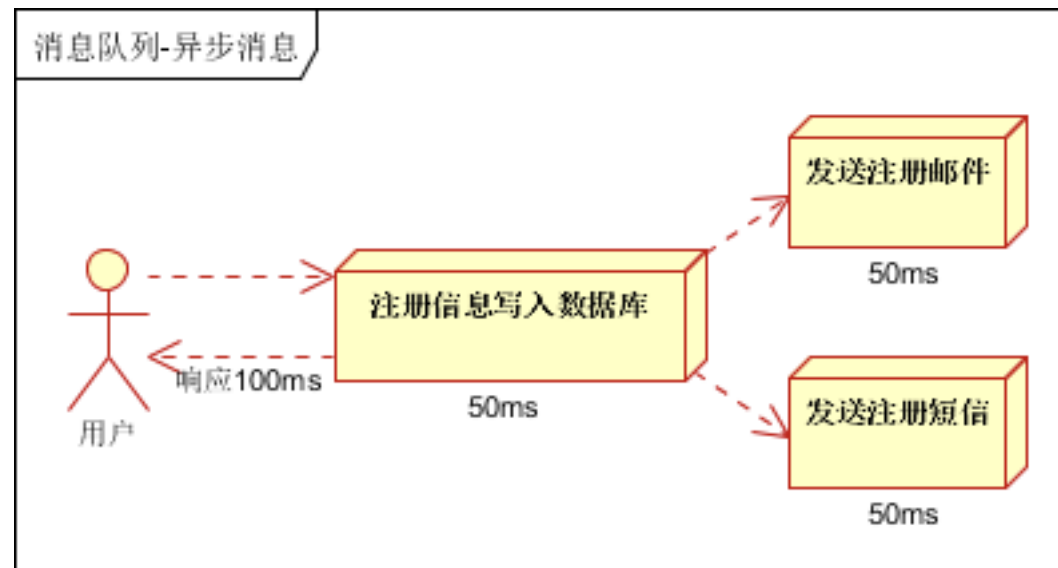
## 1. 简介

- A. NSQ是Go语言编写的，开源的内存分布式消息队列中间件
- B. 可以大规模地处理每天数以十亿计级别的消息
- C. 分布式和去中心化拓扑结构，无单点故障
- D. Github地址：<https://github.com/nsqio/nsq>

# NSQ介绍

## 2. NSQ应用场景

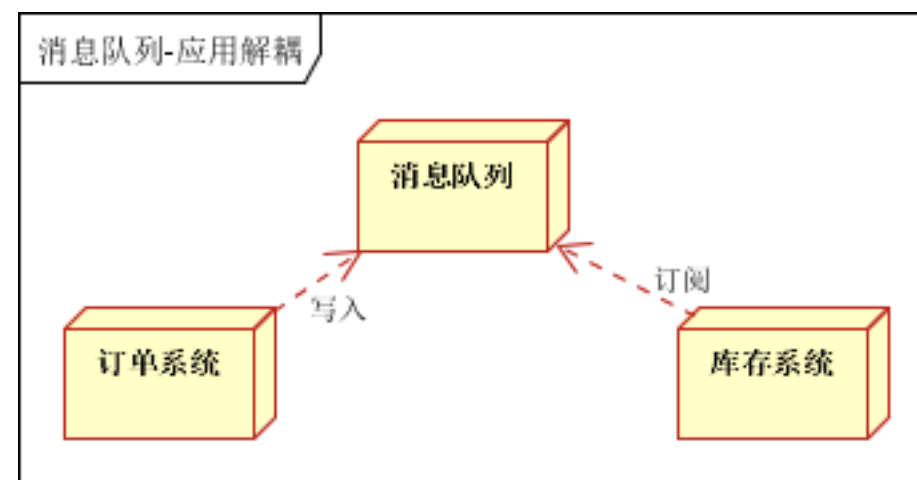
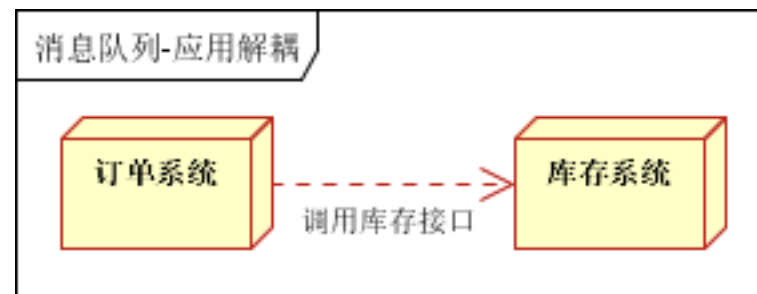
1. 异步处理, 把非关键流程异步化, 提高系统的响应时间和健壮性



# NSQ介绍

## 2. NSQ应用场景

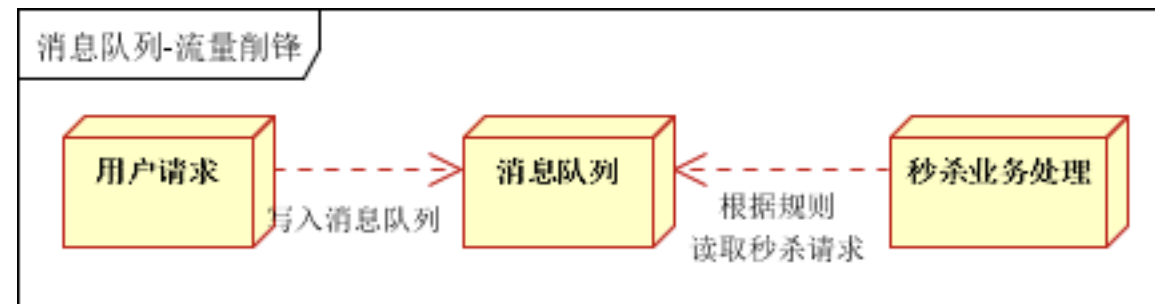
### 2. 应用解耦,通过消息队列,



# NSQ介绍

## 2. NSQ应用场景

### 3. 流量削峰



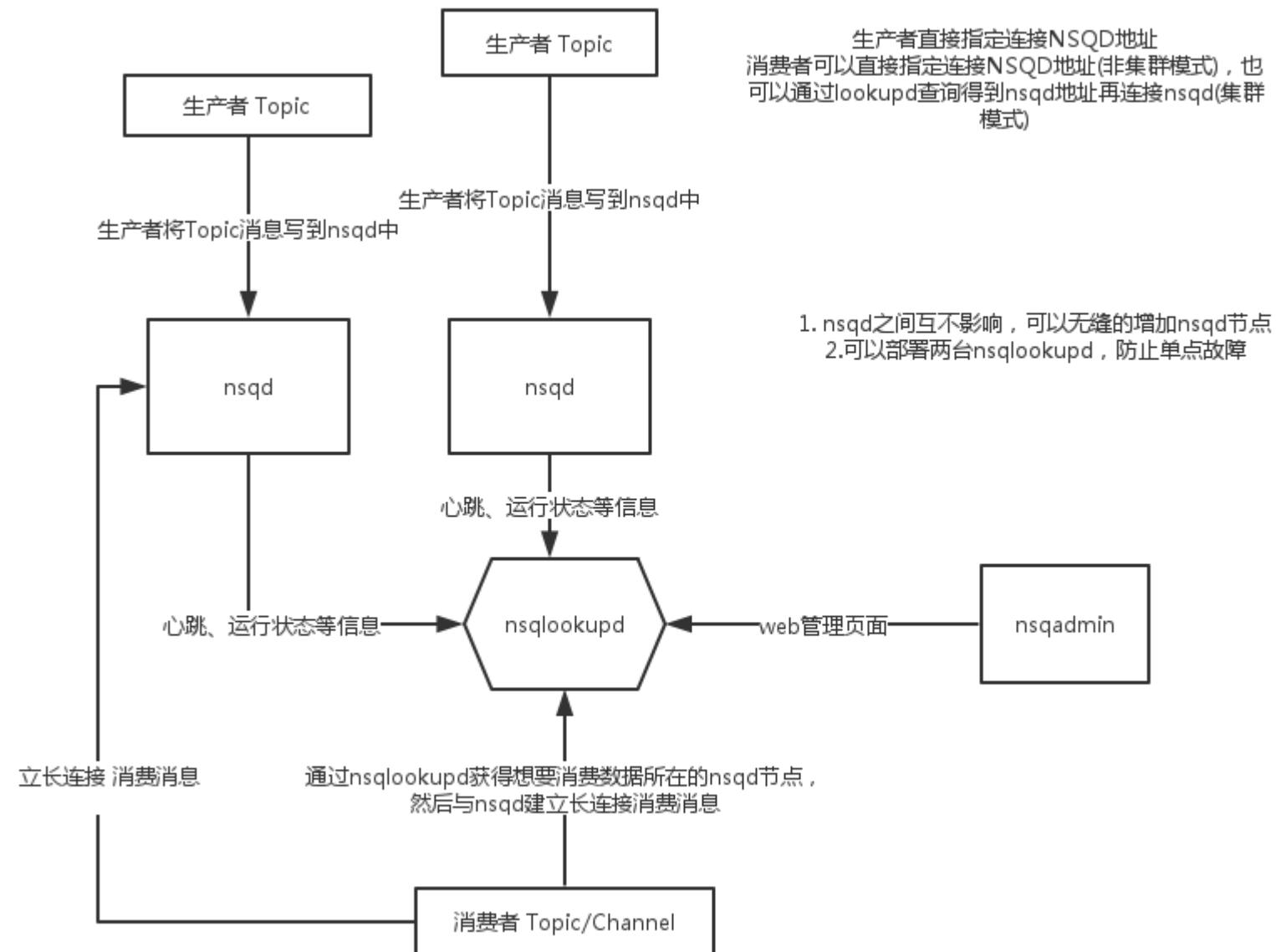
# NSQis架构和原理剖析

## 3. NSQ组件介绍

- A. nsqd, 负责消息接收、保存以及发送消息给消费者的进程
- B. nsqlookupd, 负责维护所有nsqd的状态, 提供服务发现的进程
- C. nsqadmin, 是一个web管理平台, 实时监控集群以及执行各种管理任务

# NSQ架构和原理剖析

## 4. NSQ架构介绍





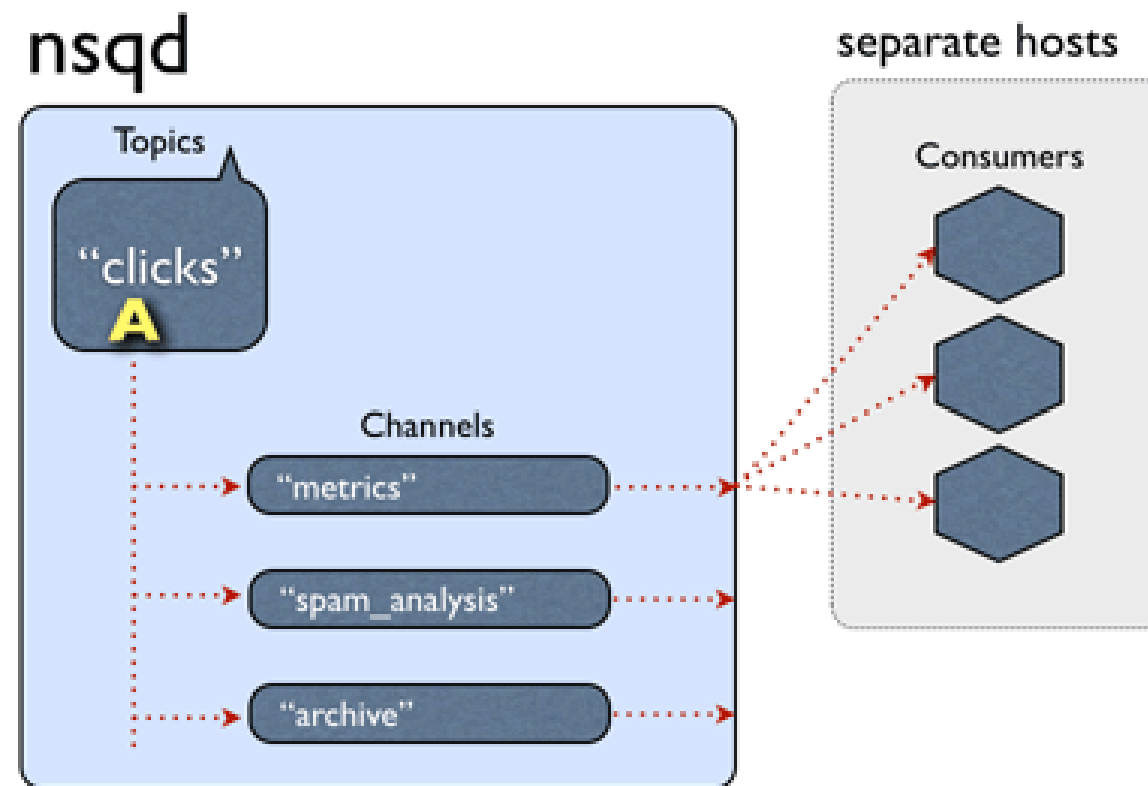
# NSQis架构和原理剖析

## 5. NSQ架构介绍

- A. Topic概念, 对应一个具体的队列。比如订单数据放到order\_queue这个topic
- B. Channel概念, 每个消费者对应一个channel。实现消息可重复消费。

## NSQ架构和原理剖析

### 6. NSQ架构介绍



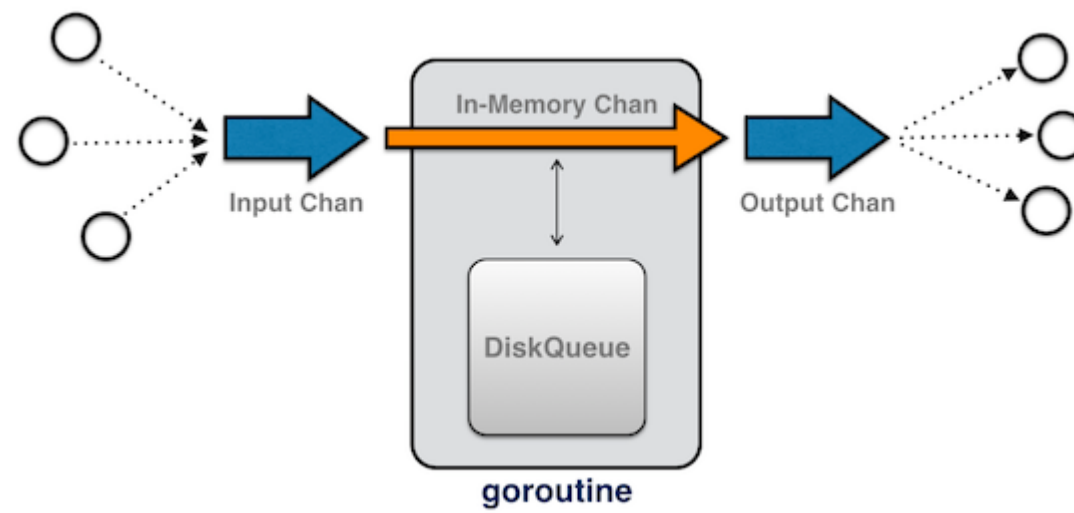
## NSQ架构和原理剖析

### 7. NSQ特性介绍

- A. 消息默认不持久化，可以配置成持久化
- B. 每条消息至少传递一次
- C. 消息不保证有序



## NSQ架构和原理剖析


### 8. NSQ接收和发送消息流程



# NSQ使用

## 9. NSQ搭建

  安全 | <https://nsq.io/deployment/installing.html>

 NSQ v1.0.0-compatible Documentation Co

OVERVIEW

Quick Start

Features & Guarantees

FAQ

Performance

Design

Internals

COMPONENTS

nsqd

nsqlookupd

nsqadmin

utilities

CLIENTS

## INSTALLING

### Binary Releases

Pre-built binaries for linux, darwin, freebsd and windows are available for download:

Current Stable Release: **v1.0.0-compatible**

- [nsq-1.0.0-compatible.darwin-amd64.go1.8.tar.gz](#)
- [nsq-1.0.0-compatible.linux-amd64.go1.8.tar.gz](#)
- [nsq-1.0.0-compatible.freebsd-amd64.go1.8.tar.gz](#)
- [nsq-1.0.0-compatible.windows-amd64.go1.8.tar.gz](#)

### Older Stable Releases

- [nsq-0.3.8.darwin-amd64.go1.6.2.tar.gz](#)
- [nsq-0.3.8.linux-amd64.go1.6.2.tar.gz](#)
- [nsq-0.3.8.freebsd-amd64.go1.6.2.tar.gz](#)
- [nsq-0.3.8.windows-amd64.go1.6.2.tar.gz](#)
- [nsq-0.3.7.darwin-amd64.go1.6.tar.gz](#)

## NSQ使用

### 10. 生成者代码示例

A. `go get github.com/nsqio/go-nsq`

B. `import "github.com/nsqio/go-nsq"`

```

package main
import (
    "bufio"
    "fmt"
    "os"
    "strings"
    "github.com/nsqio/go-nsq"
)
var producer *nsq.Producer
//入口函数
func main() {
    //nsq的地址
    nsqAddress := "127.0.0.1:4150"
    err := initProducer(nsqAddress)
    if err != nil {
        fmt.Printf("init producer failed, err:%v\n", err)
        return
    }
    //读取控制台输入
    reader := bufio.NewReader(os.Stdin)
    for {
        data, err := reader.ReadString('\n')
        if err != nil {
            fmt.Printf("read string failed, err:%v\n", err)
            continue
        }
        data = strings.TrimSpace(data)
        if data == "stop" {
            break
        }
        err = producer.Publish("order_queue", []byte(data))
        if err != nil {
            fmt.Printf("publish message failed, err:%v\n", err)
            continue
        }
        fmt.Printf("publish data:%s succ\n", data)
    }
}
// 初始化生产者
func initProducer(str string) error {
    var err error
    config := nsq.NewConfig()
    producer, err = nsq.NewProducer(str, config)
    if err != nil {
        return err
    }
    return nil
}

```

## NSQ使用

### 10. 消费者代码示例



```

package main
import (
    "fmt"
    "os"
    "syscall"
    "time"

    "os/signal"

    "github.com/nsqio/go-nsq"
)
// 消费者
type Consumer struct {
}
//处理消息
func (*Consumer) HandleMessage(msg *nsq.Message) error {
    fmt.Println("receive", msg.NSQDAddress, "message:", string(msg.Body))
    return nil
}
// 主函数
func main() {
    err := initConsumer("order_queue", "first", "127.0.0.1:4161")
    if err != nil {
        fmt.Printf("init consumer failed, err:%v\n", err)
        return
    }
    c := make(chan os.Signal)
    signal.Notify(c, syscall.SIGINT)
    <-c
}
//初始化消费者
func initConsumer(topic string, channel string, address string) error {
    cfg := nsq.NewConfig()
    cfg.LookupdPollInterval = 15 * time.Second //设置服务发现的轮询时间
    c, err := nsq.NewConsumer(topic, channel, cfg) // 新建一个消费者
    if err != nil {
        return err
    }

    consumer := &Consumer{}
    c.AddHandler(consumer) // 添加消费者接口

    //建立NSQLookupd连接
    if err := c.ConnectToNSQLookupd(address); err != nil {
        return err
    }
    return nil
}

```