# prometheus\prometheus\config\config.go

E:\workspace\go\prometheus\prometheus\vendor\github.com\

prometheus\prometheus\config\config.go

## 设置默认metrics

// DefaultScrapeConfig is the default scrape configuration.  
DefaultScrapeConfig = ScrapeConfig{  
 // ScrapeTimeout and ScrapeInterval default to the  
 // configured globals.  
 MetricsPath: "/metrics",  
 Scheme: "http",  
 HonorLabels: false,  
}

# Memory usage of Prometheus client libraries | Robust Perception

https://www.robustperception.io/memory-usage-of-prometheus-client-libraries/

**A common question around Prometheus**[**client libraries**](https://prometheus.io/docs/instrumenting/clientlibs/)**is how much RAM they’ll use on a busy process. There tends to be disbelief when we say it’s the same as an inactive server. Let’s look deeper.**

**The simplest way to test this is a small benchmark:**

**from prometheus\_client import Counter**

**import resource**

**print("Before creating counters: ", resource.getrusage(0).ru\_maxrss)**

**counters = []**

**for i in range(1000):**

**counters.append(Counter("counter{0}".format(i), "help"))**

**print("After creating counters: ", resource.getrusage(0).ru\_maxrss)**

**for i in range(10):**

**for c in counters:**

**c.inc()**

**print("After 10 increments each: ", resource.getrusage(0).ru\_maxrss)**

**for i in range(1000):**

**for c in counters:**

**c.inc()**

**print("After 1000 increments each: ", resource.getrusage(0).ru\_maxrss)**

**When run this produces for me:**

**('Before creating counters: ', 12792)**

**('After creating counters: ', 13844)**

**('After 10 increments each: ', 13844)**

**('After 1000 increments each: ', 13844)**

**So the claim that a busy server is going to use the same amount of RAM as a quiet server is shown to be true.**

**Why is this? Surely there’s buffering going on of all the increments?**

**The answer is no. The counter is just a value that is updated in memory upon an increment. If you were to look at the core of what a client library does, ignoring all the concurrency handling it is simply the constant memory function:**

**def inc(self, amount)**

**self.value += amount**

**Gauges are similarly simple, and Histograms are essentially just a convenient wrapper around a set of Counters; so both Gauges and Histograms are also constant memory. The quantiles in a Summary vary by implementation, it should be bounded in client libraries but if you’re worried use a quantile-less Summary (which is two Counters) or a Histogram instead.**

**If you’re wondering how Prometheus can work off just this single value rather than a stream of buffered events, check out**[**How does a Prometheus Counter work?**](https://www.robustperception.io/how-does-a-prometheus-counter-work/)

**(With the Java client the above claim is for practical purposes correct, but not the full truth. For performance it uses a**[**Striped64**](https://github.com/prometheus/client_java/blob/master/simpleclient/src/main/java/io/prometheus/client/Striped64.java)**, which grows its internal data structures when it encounters contention. However this growth is bounded based on the number of CPUs in the machine, and is thus constant memory.)**

**Want to know more about client library internals?**[***Contact us***](mailto:prometheus@robustperception.io)**.**

[**instrumentation**](https://www.robustperception.io/tag/instrumentation/)**,**[**java**](https://www.robustperception.io/tag/java/)**,**[**prometheus**](https://www.robustperception.io/tag/prometheus/)**,**[**python**](https://www.robustperception.io/tag/python/)

# Analysing Prometheus Memory Usage | Robust Perception

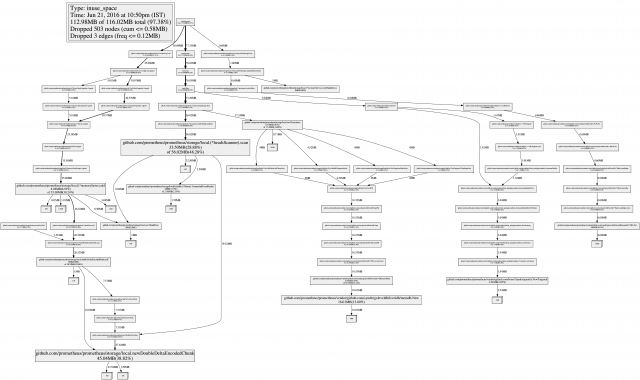
https://www.robustperception.io/analysing-prometheus-memory-usage/

**Ever wondered how**[**Prometheus**](https://prometheus.io/)**is using its memory? Let’s find out!**

**Prometheus is linked with**[**pprof**](https://golang.org/pkg/net/http/pprof/)**, a Go profiling tool that makes it easy to look at CPU and memory usage. To use it against a local Prometheus server to investigate memory usage, ensure you have a working Go install and then run:**

**go tool pprof -svg http://localhost:9090/debug/pprof/heap > heap.svg**

**This will produce a SVG file that you can open in your web browser. Here’s an example from a small Prometheus server:**

**[](http://www.robustperception.io/wp-content/uploads/2016/06/heap.png)**

**local.newDoubleDeltaEncodedChunk in the bottom left here is memory used by samples, and will usually be the biggest memory user. The local.newPersistence subtree covers the metadata database.**

**There are metrics that are useful. process\_resident\_memory\_bytes is the amount of memory the Prometheus process is using from the kernel, while go\_memstats\_alloc\_bytes is how much Go is using from that. A large difference between these two could indicate spiky memory usage, or fragmentation issues.**

[**profiling**](https://www.robustperception.io/tag/profiling/)**,**[**prometheus**](https://www.robustperception.io/tag/prometheus/)

# Prometheus源码分析(一)编译安装及命令行参数说明

- jianyuanPC的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/jianyuanpc/article/details/52561744

目前开源的告警系统不少，选择Prometheus主要因为，它比较轻便、支持复杂的规则运算、规则的动态加载、组件之间的耦合度低（都是通过http协议交互）、而且生态圈完善。

prometheus主要应用于告警业务，常用的组件有prometheus、pushgateway、alertmanager。

* prometheus组件：用于根据所配置的规则进行规则运算，当存在规则触发时，将告警信息发送给alertmanager。
* pushgateway组件：数据网关，prometheus的数据源，
* alertmanager组件：用于发送告警（短信、邮件等）

Prometheus详情请访问以下网址：

<https://prometheus.io/docs/introduction/overview/>

对Prometheus感兴趣的朋友请加入QQ群：70860761 一起探讨

**源码编译及安装**

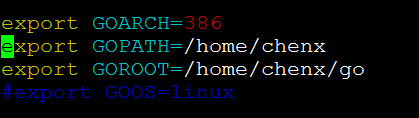
**相关程序下载**

下载安装[**Git**](http://lib.csdn.net/base/git)：yum install -y git

下载[**Go**](http://lib.csdn.net/base/go)   
  
版本：go1.6.2.[**Linux**](http://lib.csdn.net/base/linux)-386.tar.gz，根据系统选择版本   
下载地址：<http://www.golangtc.com/download>

下载依赖：   
yum install -y gcc mercurial

**环境变量设置**



**编译启动**

mkdir -p GOPATH/src/github.com/prometheus

cd GOPATH/src/github.com/prometheus

git clone https://github.com/prometheus/prometheus.git

cd prometheus

// 开始编译

make build

// 编译完成启动prometheus 命令

./prometheus -config.file=your\_config.yml

说明：alertmanager、pushgateway组件源码的编译与Prometheus类型(make build)

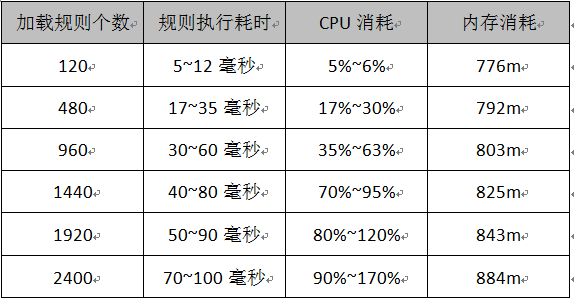
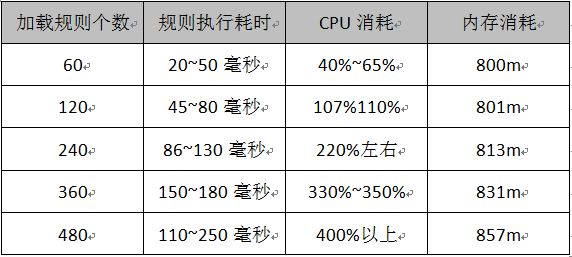
**命令行参数说明**

**prometheus**

1. -version 查看版本信息
2. -config.file “prometheus.yml” 指定加载的配置文件，默认当前路径下prometheus.yml
3. -alertmanager.notification-queue-capacity 10000 告警队列大小，默认值为10000
4. -alertmanager.timeout 10s 通过HTTP接口发送告警到AlertManager的超时时间，默认为10s
5. -alertmanager.url 指定alertmanager的地址
6. -query.max-concurrency 20 最大并发查询连接个数，默认为20
7. -query.timeout 2m0s 查询超时时间，默认2分钟
8. -query.staleness-delta 5m0s 这个参数很重要，当没有scrape到指标是，使用上次(最近一次)指标值进行补，该值设置距上次指标的时间间隔，默认为5分钟，操作5分针就补空值。
9. STORAGE命令参数主要用于设置指标存储的方式，目前prometheus支持opentsdb、influxdb、local三种
10. -web.listen-address “:9090” 指定外部请求的访问地址，默认为本地的9090端口。

**性能测试**

[**测试**](http://lib.csdn.net/base/softwaretest)机器数量：1台（非独立）

* 规则运算不指定维度   
  
* 规则运算指定维度   
  

# Prometheus源码分析(二)配置文件说明

- jianyuanPC的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/jianyuanpc/article/details/52756887

本想直接对Prometheus各个组件进行源码分析，但考虑到源码中与prometheus、alertmanager组件中配置文件(prometheus.yml、alertmanager.yml)有很大的关联，所以这一节主要针对配置文件进行说明。

Prometheus更多功能介绍请访问以下网址：

<https://prometheus.io/docs/introduction/overview/>

**对Prometheus感兴趣的朋友请加入QQ群：70860761 一起探讨**

**配置说明**

**prometheus.yml**

# my global config

global:

scrape\_interval: 15s # By default, scrape targets every 15 seconds.

evaluation\_interval: 15s # By default, scrape targets every 15 seconds.

# scrape\_timeout is set to the global default (10s).

# Attach these labels to any time series or alerts when communicating with

# external systems (federation, remote storage, Alertmanager).

external\_labels:

monitor: 'codelab-monitor'

# Load and evaluate rules in this file every 'evaluation\_interval' seconds.

rule\_files:

# - "first.rules"

# - "second.rules"

- "alert.rules"

# - "record.rules"

# A scrape configuration containing exactly one endpoint to scrape:

# Here it's Prometheus itself.

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: 'windows-test'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 1s

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['192.168.3.1:9090','192.168.3.120:9090']

- job\_name: 'windows-chenx'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 3s

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['192.168.3.1:9091']

参数说明：

* global下的scrape\_interval  
  用于向pushgateway采集数据的频率，上图所示：每隔15秒向pushgateway采集一次指标数据
* global下的evaluation\_interval  
  表示规则计算的频率，上图所示：每隔15秒根据所配置的规则集，进行规则计算
* global下的external\_labels  
  为指标增加额外的维度，可用于区分不同的prometheus,在应用中多个prometheus可以对应一个alertmanager
* rule\_files   
  指定所配置规则文件，文件中每行可表示一个规则
* scrape\_configs下的job\_name  
  指定任务名称，在指标中会增加该维度，表示该指标所属的job
* scrape\_configs下的scrape\_interval  
  覆盖global下的scrape\_interval配置
* static\_configs下的targets  
  指定指标数据源的地址，多个地址之间用逗号隔开

**alertmanager.yml**

global:

# The smarthost and SMTP sender used for mail notifications.

smtp\_smarthost: 'smtp.qq.com:465'

smtp\_from: '447040949@qq.com'

smtp\_auth\_username: '447040949@qq.com'

smtp\_auth\_password: 'nihao206206#'

# The auth token for Hipchat.

hipchat\_auth\_token: '1234556789'

# Alternative host for Hipchat.

hipchat\_url: 'https://hipchat.foobar.org/'

# The directory from which notification templates are read.

templates:

- '/etc/alertmanager/template/\*.tmpl'

# The root route on which each incoming alert enters.

route:

# The labels by which incoming alerts are grouped together. For example,

# multiple alerts coming in for cluster=A and alertname=LatencyHigh would

# be batched into a single group.

group\_by: ['alertname', 'cluster', 'service']

# When a new group of alerts is created by an incoming alert, wait at

# least 'group\_wait' to send the initial notification.

# This way ensures that you get multiple alerts for the same group that start

# firing shortly after another are batched together on the first

# notification.

group\_wait: 30s

# When the first notification was sent, wait 'group\_interval' to send a batch

# of new alerts that started firing for that group.

group\_interval: 5m

# If an alert has successfully been sent, wait 'repeat\_interval' to

# resend them.

repeat\_interval: 3h

# A default receiver

receiver: team-X-mails

# All the above attributes are inherited by all child routes and can

# overwritten on each.

# The child route trees.

routes:

# This routes performs a regular expression match on alert labels to

# catch alerts that are related to a list of services.

- match\_re:

service: ^(foo1|foo2|baz)$

receiver: team-X-mails

# The service has a sub-route for critical alerts, any alerts

# that do not match, i.e. severity != critical, fall-back to the

# parent node and are sent to 'team-X-mails'

routes:

- match:

severity: critical

receiver: team-X-pager

- match:

service: files

receiver: team-Y-mails

routes:

- match:

severity: critical

receiver: team-Y-pager

# This route handles all alerts coming from a database service. If there's

# no team to handle it, it defaults to the DB team.

- match:

service: database

receiver: team-DB-pager

# Also group alerts by affected database.

group\_by: [alertname, cluster, database]

routes:

- match:

owner: team-X

receiver: team-X-pager

- match:

owner: team-Y

receiver: team-Y-pager

# Inhibition rules allow to mute a set of alerts given that another alert is

# firing.

# We use this to mute any warning-level notifications if the same alert is

# already critical.

inhibit\_rules:

- source\_match:

severity: 'critical'

target\_match:

severity: 'warning'

# Apply inhibition if the alertname is the same.

equal: ['alertname', 'cluster', 'service']

receivers:

- name: 'team-X-mails'

webhook\_configs:

- url: 'http://u2.kugou.net:11770/sendRtxByPost'

- name: 'team-X-pager'

email\_configs:

- to: 'team-X+alerts-critical@example.org'

pagerduty\_configs:

- service\_key: <team-X-key>

- name: 'team-Y-mails'

email\_configs:

- to: 'team-Y+alerts@example.org'

- name: 'team-Y-pager'

pagerduty\_configs:

- service\_key: <team-Y-key>

- name: 'team-DB-pager'

pagerduty\_configs:

- service\_key: <team-DB-key>

- name: 'team-X-hipchat'

hipchat\_configs:

- auth\_token: <auth\_token>

room\_id: 85

message\_format: html

notify: true

参数说明

* global  
  smtp\_smarthost、smtp\_from、smtp\_auth\_username、smtp\_auth\_password用于设置smtp邮件的地址及用户信息  
  hipchat\_auth\_token与安全性认证有关
* templates  
  指定告警信息展示的模版
* route  
  group\_by：指定所指定的维度对告警进行分组  
  group\_wait:指定每组告警发送等待的时间  
  group\_interval:指定告警调度的时间间隔  
  repeat\_interval:在连续告警触发的情况下，重复发送告警的时间间隔
* receiver  
  指定告警默认的接受者
* routes  
  match\_re:定义告警接收者的匹配方式  
  service:定义匹配的方式，纬度service值以foo1或foo2或baz开始/结束时表示匹配成功  
  receiver：定义了匹配成功的的情况下的接受者
* inhibit\_rules  
  定义告警的抑制条件，过滤不必要的告警
* receivers  
  定义了具体的接收者，也就是告警具体的方式方式

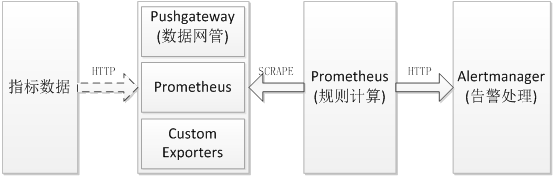
# Prometheus源码分析(三)Prometheus常用服务架构

- jianyuanPC的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/jianyuanpc/article/details/52771791

**对Prometheus感兴趣的朋友请加入QQ群：70860761 一起探讨**

**Prometheus常用服务架构图**



指标的采集有以下三种方式：  
1. 指标数据先存储到第三方组件中(如：Kafka),通过http协议发送到Pushgateway,然后Prometheus周期性的从Pushgateway中获取指标数据。  
2. Prometheus互相可以成为对方的target（目标）,从对方服务中获取指标数据。  
3. 自定义指标数据(CustomExporter),Prometheus将CustomExporter设置为自己的target.

**Pushgateway方式**

在Prometheus的prometheus.yml配置文件中的scrape\_configs标签下添加一下啊配置：

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: 'windows-test'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 1s

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['192.168.3.120:9091']

说明：Pushgateway默认端口为9091

发送指标到Pushgateway

import io.prometheus.client.CollectorRegistry;

import io.prometheus.client.Gauge;

import io.prometheus.client.exporter.PushGateway;

void executeBatchJob() throws Exception {

CollectorRegistry registry = new CollectorRegistry();

Gauge duration = Gauge.build()

.name("my\_batch\_job\_duration\_seconds")

.help("Duration of my batch job in seconds.")

.register(registry);

Gauge.Timer durationTimer = duration.startTimer();

try {

// Your code here.

// This is only added to the registry after success,

// so that a previous success in the Pushgateway is not overwritten on failure.

Gauge lastSuccess = Gauge.build()

.name("my\_batch\_job\_last\_success\_unixtime")

.help("Last time my batch job succeeded, in unixtime.")

.register(registry);

lastSuccess.setToCurrentTime();

} finally {

durationTimer.setDuration();

PushGateway pg = new PushGateway("127.0.0.1:9091");

pg.pushAdd(registry, "my\_batch\_job");

}

}

所依赖Maven包

<dependency>

<groupId>io.prometheus</groupId>

<artifactId>simpleclient</artifactId>

<version>0.0.10</version>

</dependency>

<dependency>

<groupId>io.prometheus</groupId>

<artifactId>simpleclient\_pushgateway</artifactId>

<version>0.0.10</version>

</dependency>

**Prometheus互相采集方式**

与Pushgateway方式类似，将Prometheus的地址(格式IP:端口)加入到指定prometheus.yml配置文件中启动即可。

**自定义方式**

实现方式  
1、自定义的指标收集类都必须到CollectorRegistry进行注册， 指标数据通过CollectorRegistry类的方法或者函数，返回给Prometheus.  
2、CollectorRegistry必须提供register()和unregister()函数，一个指标收集器可以注册多个CollectorRegistry.  
3、客户端库必须是线程安全的.

示例

func NewCollector(program string) \*prometheus.GaugeVec {

buildInfo := prometheus.NewGaugeVec(

prometheus.GaugeOpts{

Namespace: program,

Name: "build\_info",

Help: fmt.Sprintf(

"A metric with a constant '1' value labeled by version, revision, branch, and goversion from which %s was built.",

program,

),

},

[]string{"version", "revision", "branch", "goversion"},

)

buildInfo.WithLabelValues(Version, Revision, Branch, GoVersion).Set(1)

return buildInfo

}

prometheus.MustRegister(version.NewCollector("mysqld\_exporter"))

**告警发送服务接口**

Prometheus根据告警规则配置自动发送告警信息。以HTTP POST的方式发送报警信息,只要告警的状态活跃就会不停的告警，告警默认地址: [http://AlertManagerIP:9093/api/v1/alerts](http://alertmanagerip:9093/api/v1/alerts)。   
告警格式如下：

[

{

"labels": {

"<labelname>": "<labelvalue>",

...

},

"annotations": {

"<labelname>": "<labelvalue>",

},

"startsAt": "<rfc3339>",

"endsAt": "<rfc3339>"

"generatorURL": "<generator\_url>"

},

...

]

说明：  
labels：区分具相同告警实体不同标签类别的告警。  
annotations：表示最新接收没有被标识的告警  
startsAs和endsAt时间戳在省略的情况下，startsAs为Alertmanager当前时间，endsAt只能设置为告警的结束时间。其它情况下配置的超时时间从最近一次接受到告警开始计时。  
generatorURL：唯一标识告警客户端的入口。

# Prometheus源码分析(四)Prometheus启动过程

- jianyuanPC的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/jianyuanpc/article/details/54412773

**对Prometheus感兴趣的朋友请加入QQ群：70860761 一起探讨**

**Prometheus启动过程**

启动入口源码

func Main() int {

if err := parse(os.Args[1:]); err != nil {

log.Error(err)

return 2

}

if cfg.printVersion {

fmt.Fprintln(os.Stdout, version.Print("prometheus"))

return 0

}

log.Infoln("Starting prometheus", version.Info())

log.Infoln("Build context", version.BuildContext())

var reloadables []Reloadable

var (

memStorage = local.NewMemorySeriesStorage(&cfg.storage)

remoteStorage = remote.New(&cfg.remote)

sampleAppender = storage.Fanout{memStorage}

)

if remoteStorage != nil {

sampleAppender = append(sampleAppender, remoteStorage)

reloadables = append(reloadables, remoteStorage)

}

var (

notifier = notifier.New(&cfg.notifier)

targetManager = retrieval.NewTargetManager(sampleAppender)

queryEngine = promql.NewEngine(memStorage, &cfg.queryEngine)

)

ruleManager := rules.NewManager(&rules.ManagerOptions{

SampleAppender: sampleAppender,

Notifier: notifier,

QueryEngine: queryEngine,

ExternalURL: cfg.web.ExternalURL,

})

flags := map[string]string{}

cfg.fs.VisitAll(func(f \*flag.Flag) {

flags[f.Name] = f.Value.String()

})

// 当前版本信息

version := &web.PrometheusVersion{

Version: version.Version,

Revision: version.Revision,

Branch: version.Branch,

BuildUser: version.BuildUser,

BuildDate: version.BuildDate,

GoVersion: version.GoVersion,

}

webHandler := web.New(memStorage, queryEngine, targetManager, ruleManager, version, flags, &cfg.web)

reloadables = append(reloadables, targetManager, ruleManager, webHandler, notifier)

if !reloadConfig(cfg.configFile, reloadables...) {

return 1

}

// Wait for reload or termination signals. Start the handler for SIGHUP as

// early as possible, but ignore it until we are ready to handle reloading

// our config.

hup := make(chan os.Signal)

hupReady := make(chan bool)

signal.Notify(hup, syscall.SIGHUP)

go func() {

<-hupReady

for {

select {

case <-hup:

case <-webHandler.Reload():

}

reloadConfig(cfg.configFile, reloadables...)

}

}()

// Start all components. The order is NOT arbitrary.

if err := memStorage.Start(); err != nil {

log.Errorln("Error opening memory series storage:", err)

return 1

}

defer func() {

if err := memStorage.Stop(); err != nil {

log.Errorln("Error stopping storage:", err)

}

}()

if remoteStorage != nil {

prometheus.MustRegister(remoteStorage)

go remoteStorage.Run()

defer remoteStorage.Stop()

}

// The storage has to be fully initialized before registering.

prometheus.MustRegister(memStorage)

prometheus.MustRegister(notifier)

prometheus.MustRegister(configSuccess)

prometheus.MustRegister(configSuccessTime)

// The notifieris a dependency of the rule manager. It has to be

// started before and torn down afterwards.

go notifier.Run()

defer notifier.Stop()

go ruleManager.Run()

defer ruleManager.Stop()

go targetManager.Run()

defer targetManager.Stop()

// Shutting down the query engine before the rule manager will cause pending queries

// to be canceled and ensures a quick shutdown of the rule manager.

defer queryEngine.Stop()

go webHandler.Run()

// Wait for reload or termination signals.

close(hupReady) // Unblock SIGHUP handler.

term := make(chan os.Signal)

signal.Notify(term, os.Interrupt, syscall.SIGTERM)

select {

case <-term:

log.Warn("Received SIGTERM, exiting gracefully...")

case <-webHandler.Quit():

log.Warn("Received termination request via web service, exiting gracefully...")

case err := <-webHandler.ListenError():

log.Errorln("Error starting web server, exiting gracefully:", err)

}

log.Info("See you next time!")

return 0

}

// Reloadable things can change their internal state to match a new config

// and handle failure gracefully.

type Reloadable interface {

ApplyConfig(\*config.Config) bool

}

// 配置加载处理

func reloadConfig(filename string, rls ...Reloadable) (success bool) {

log.Infof("Loading configuration file %s", filename)

defer func() {

if success {

configSuccess.Set(1)

configSuccessTime.Set(float64(time.Now().Unix()))

} else {

configSuccess.Set(0)

}

}()

conf, err := config.LoadFile(filename)

if err != nil {

log.Errorf("Couldn't load configuration (-config.file=%s): %v", filename, err)

return false

}

success = true

for \_, rl := range rls {

success = success && rl.ApplyConfig(conf)

}

return success

}

* 2行：parse方法解析命令行参数
* 18~25行：根据所解析的命令行参数构造指标存储引擎（本地模式或者远程模式[opentsdb,influxdb…]）。
* 27~38行：构造规则管理服务(ruleManager)，包括规则的执行和告警的发送等；
* 54行：构造Web管理服务
* 56~60行：加载告警规则，并同步到相关服务
* 65~77行：告警规则动态加载处理，以Linux信号量的通知方式实现；
* 81~96行：启动指标存储引擎以及程序退出引擎析构处理；
* 89~101行：注册prometheus组件内部所输出的系统指标；

# cmd\prometheus\main.go

E:\workspace\go\prometheus\prometheus\

cmd\prometheus\main.go

## 判断federate的存储方式

webHandler

入口

webHandler := web.New(&cfg.web)

E:\workspace\go\prometheus\prometheus

\cmd\prometheus\config.go

cfg.fs.StringVar(  
 &cfg.localStorageEngine, "storage.local.engine", "persisted",  
 "Local storage engine. Supported values are: 'persisted' (full local storage with on-disk persistence) and 'none' (no local storage).",  
)

E:\workspace\go\prometheus\prometheus\

cmd\prometheus\main.go

默认为persisted，所以**localStorage**赋值为本地存储

**var** localStorage local.Storage  
**switch** cfg.localStorageEngine {  
**case** "persisted":  
 localStorage = local.NewMemorySeriesStorage(&cfg.storage)  
 sampleAppender = storage.Fanout{localStorage}  
**case** "none":  
 localStorage = &local.NoopStorage{}  
**default**:  
 log.Errorf("Invalid local storage engine %q", cfg.localStorageEngine)  
 **return** 1  
}

处理federate的请求

E:\workspace\go\prometheus\prometheus\

web\web.go

router.Get("/federate", instrh("federate", httputil.CompressionHandler{  
 Handler: http.HandlerFunc(h.federation),  
}))

E:\workspace\go\prometheus\prometheus\

web\federate.go

# storage\remote\write.go

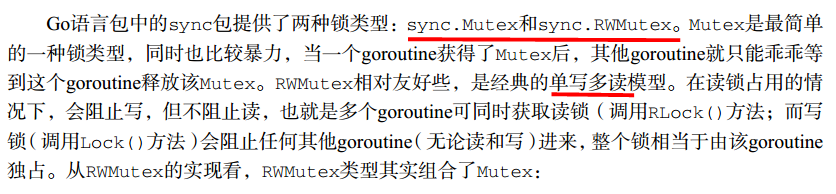
E:\workspace\go\prometheus\prometheus\vendor\github.com\prometheus\prometheus\

storage\remote\write.go

## Writer

// Writer allows queueing samples for remote writes.  
**type** Writer **struct** {  
 mtx sync.RWMutex  
 queues []\*QueueManager  
}

4.8.1 同步锁

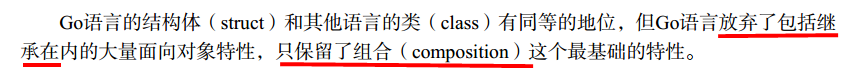


### ApplyConfig

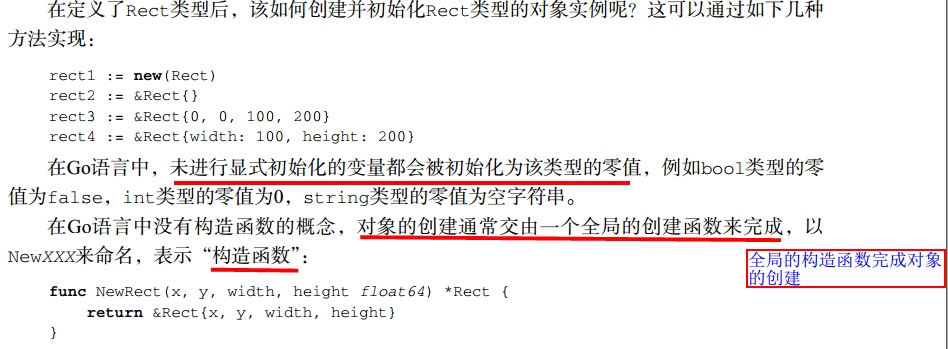
// ApplyConfig updates the state as the new config requires.  
**func** (w \*Writer) ApplyConfig(conf \*config.Config) error {  
 w.mtx.Lock()  
 **defer** w.mtx.Unlock()  
  
 newQueues := []\*QueueManager{}  
 // *TODO: we should only stop & recreate queues which have changes,* // as this can be quite disruptive.  
 **for** i, rwConf := **range** conf.RemoteWriteConfigs {  
 c, err := NewClient(i, &clientConfig{  
 url: rwConf.URL,  
 timeout: rwConf.RemoteTimeout,  
 httpClientConfig: rwConf.HTTPClientConfig,  
 })  
 **if** err != nil {  
 **return** err  
 }  
 newQueues = append(newQueues, NewQueueManager(  
 defaultQueueManagerConfig,  
 conf.GlobalConfig.ExternalLabels,  
 rwConf.WriteRelabelConfigs,  
 c,  
 ))  
 }  
//停止原队列

**for** \_, q := **range** w.queues {  
 q.Stop()  
 }  
//重启新队列

w.queues = newQueues  
 **for** \_, q := **range** w.queues {  
 q.Start()  
 }  
 **return** nil  
}



未进行显式初始化的变量都会被初始化为该类型的零值



# storage\remote\queue\_manager.go

E:\workspace\go\prometheus\prometheus\

storage\remote\queue\_manager.go

## 收集counter

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | succeeded\_samples\_total | Total number of samples successfully sent to remote storage. |
|  | failed\_samples\_total | Total number of samples which failed on send to remote storage. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 配置QueueManagerConfig

// QueueManagerConfig is the configuration for the queue used to write to remote  
// storage.  
**type** QueueManagerConfig **struct** {  
 // Number of samples to buffer per shard before we start dropping them.  
 QueueCapacity int  
 // Max number of shards, i.e. amount of concurrency.  
 MaxShards int  
 // Maximum number of samples per send.  
 MaxSamplesPerSend int  
 // Maximum time sample will wait in buffer.  
 BatchSendDeadline time.Duration  
 // Max number of times to retry a batch on recoverable errors.  
 MaxRetries int  
 // On recoverable errors, backoff exponentially.  
 MinBackoff time.Duration  
 MaxBackoff time.Duration  
}

### 默认配置

这个队列的最大分片是1000，每个分片每秒100个sample，那么一秒就可以发送100\*1000个sample。每一种存储，无论是本地存储还有远端存储，写数据都实现Append方法，remote的也一样，在remote的Append就调用了queue的Append方法。

// defaultQueueManagerConfig is the default remote queue configuration.  
**var** defaultQueueManagerConfig = QueueManagerConfig{  
 // With a maximum of 1000 shards, assuming an average of 100ms remote write  
 // time and 100 samples per batch, we will be able to push 1M samples/s.  
 MaxShards: 1000,  
 MaxSamplesPerSend: 100,  
  
 // By default, buffer 1000 batches, which at 100ms per batch is 1:40mins. At  
 // 1000 shards, this will buffer 100M samples total.  
 QueueCapacity: 100 \* 1000,  
 BatchSendDeadline: 5 \* time.*Second*,  
  
 // Max number of times to retry a batch on recoverable errors.  
 MaxRetries: 10,  
 MinBackoff: 30 \* time.*Millisecond*,  
 MaxBackoff: 100 \* time.*Millisecond*,  
}

## StorageClient

// StorageClient defines an interface for sending a batch of samples to an  
// external timeseries database.  
**type** StorageClient **interface** {  
 // Store stores the given samples in the remote storage.  
 Store(model.Samples) error  
 // Name identifies the remote storage implementation.  
 Name() string  
}

## QueueManager

// QueueManager manages a queue of samples to be sent to the Storage  
// indicated by the provided StorageClient.  
**type** QueueManager **struct** {  
 cfg QueueManagerConfig  
 externalLabels model.LabelSet  
 relabelConfigs []\*config.RelabelConfig  
 client StorageClient  
 queueName string  
 logLimiter \*rate.Limiter  
  
 shardsMtx sync.Mutex  
 shards \*shards  
 numShards int  
 reshardChan **chan** int  
 quit **chan struct**{}  
 wg sync.WaitGroup  
  
 samplesIn, samplesOut, samplesOutDuration ewmaRate  
 integralAccumulator float64  
}