

# 关于 Prometheus Exporter

作者: [someone61489](#)

原文链接: <https://ld246.com/article/1609418091123>

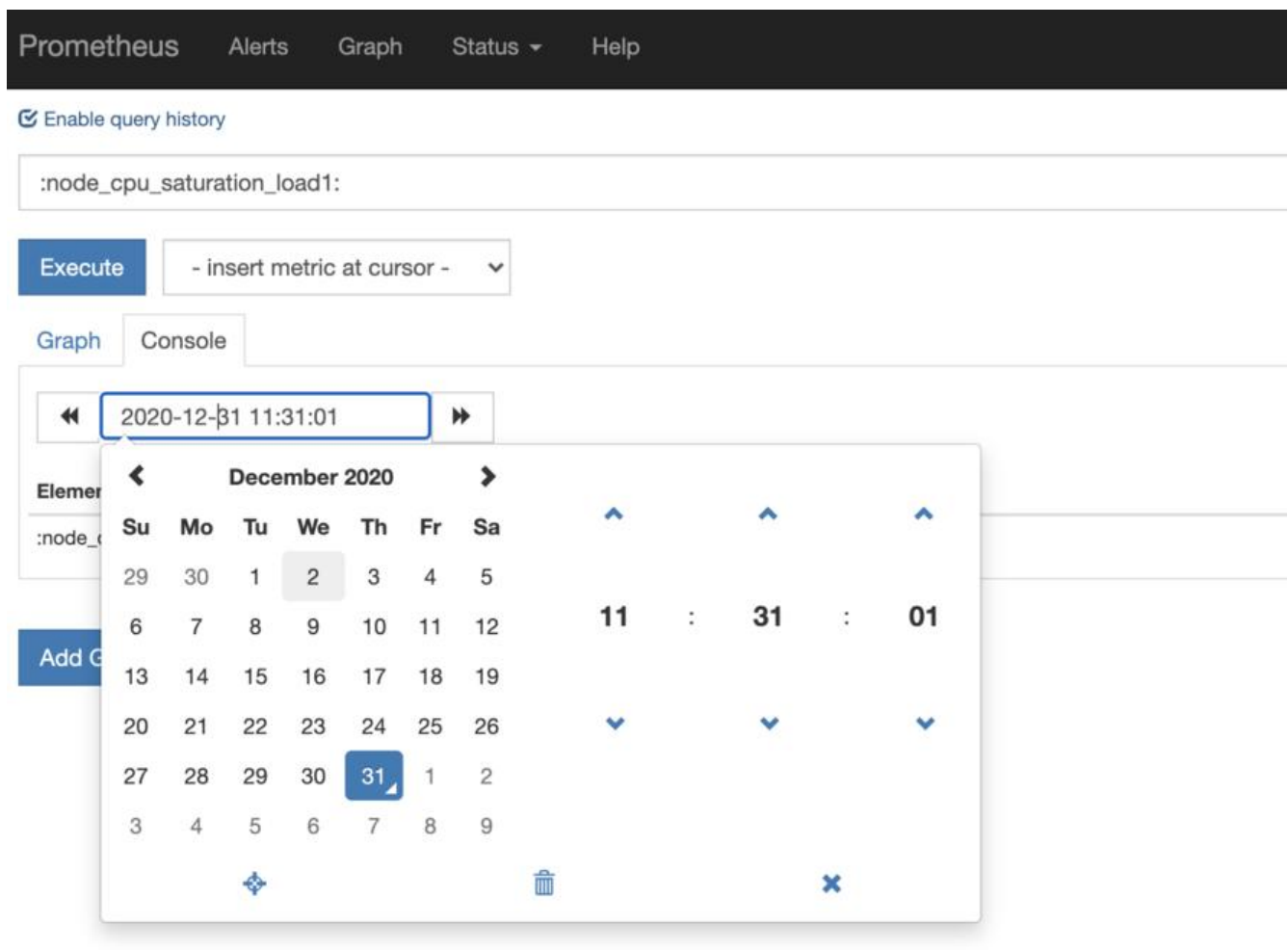
来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

# Prometheus

time series metrics collect toolkit

提供了UI服务，默认端口30313



主要作用就是监控一下我们服务的各项指标，也有许多包已经集成了这个功能，基本上查找一下配置下就可以快速的展示出各项指标了

## 业务指标

但例如 `java_gc`, `go_gc`, `node_request_total` 这种指标对我们来说没有特别大的用处，我们的着力点应在于提取业务指标，各个业务系统数据源不同逻辑不同导致无法使用通用型第三方包来构建我们的指。

这个时候就出现了 `prometheus-exporter` 用来放出自定义的业务指标，等待 `prometheus` 来此处收集。

## prometheus exporter

其实很简单，就是攒一些prometheus数据格式的数据放入 `/metrics` 路由中等待scrape就好

下面使用Go来举例子

```
1 go.mod
2 module github.com/hbyunzai/exporterv2
3
4 go 1.15
5
6 require (
7     github.com/godror/godror v0.22.3
8     github.com/prometheus/client_golang v1.9.0
9     github.com/sirupsen/logrus v1.7.0
10 )
```

这里使用了godror驱动oracle，prometheus/client\_golang 客户端,logrus日志模块

```
34 oracleURL      = flag.String("oracle.url", utils.GetEnvString("YUNZAI_EXPORTER_ORACLE_URL", ""), "Address of the oracle host")
35 oracleUserName = flag.String("oracle.username", utils.GetEnvString("YUNZAI_EXPORTER_ORACLE_USERNAME", ""), "Username of oracle")
36 oraclePassword = flag.String("oracle.password", utils.GetEnvString("YUNZAI_EXPORTER_ORACLE_PASSWORD", ""), "Password of oracle")
37 exporterNamespace = flag.String("exporter.namespace", utils.GetEnvString("YUNZAI_EXPORTER_NAMESPACE", "yunzai"), "Namespace of yunzai exporter")
38 listenAddress    = flag.String("server.listen-address", utils.GetEnvString("YUNZAI_EXPORTER_SERVER_ADDRESS", ":10086"), "Address to listen on for web interface and
39 logDebug        = flag.Bool("exporter.log-debug", utils.GetEnvBool("YUNZAI_EXPORTER_LOG-DEBUG", true), "Log debug mode")
40 longitude       = flag.String("exporter.school-longitude", utils.GetEnvString("YUNZAI_EXPORTER_LONGITUDE", ""), "exporter deploy longitude")
41 latitude        = flag.String("exporter.school-latitude", utils.GetEnvString("YUNZAI_EXPORTER_LATITUDE", ""), "exporter deploy latitude")
42 name            = flag.String("exporter.school-name", utils.GetEnvString("YUNZAI_EXPORTER_SCHOOL_NAME", ""), "exporter deploy school name")
43 lcon            = flag.String("exporter.school-lcon", utils.GetEnvString("YUNZAI_EXPORTER_SCHOOL_ICON", ""), "exporter deploy school lcon http path")
44 gateway         = flag.String("exporter.school-gateway", utils.GetEnvString("YUNZAI_EXPORTER_SCHOOL_GATEWAY", "/backstage"), "gateway path (default /backstage)")
45 logFormatter    = flag.String("exporter.log-formatter", utils.GetEnvString("YUNZAI_EXPORTER_LOG-FORMATTER", "text"), "Log formatter default is text,text or json")
46 metricsOnly     = flag.Bool("exporter.metrics.only", utils.GetEnvBool("YUNZAI_EXPORTER_METRICS_ONLY", false), "only has yunzai exporter metrics")
47 metricsPath     = flag.String("exporter.metrics.path", utils.GetEnvString("YUNZAI_EXPORTER_METRICS_PATH", "/metrics"), "metrics path")
48 tlsClientKeyFile = flag.String("tls-client-key-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_CLIENT_KEY_FILE", ""), "Name of the client key file (including full
49 tlsClientCertFile = flag.String("tls-client-cert-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_CLIENT_CERT_FILE", ""), "Name of the client certificate file (inclu
50 tlsCaCertFile    = flag.String("tls-ca-cert-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_CA_CERT_FILE", ""), "Name of the CA certificate file (including full pa
51 tlsServerKeyFile = flag.String("tls-server-key-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_SERVER_KEY_FILE", ""), "Name of the server key file (including full
52 tlsServerCertFile = flag.String("tls-server-cert-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_SERVER_CERT_FILE", ""), "Name of the server certificate file (inclu
53 skipTLSVerification = flag.Bool("skip-tls-verification", utils.GetEnvBool("YUNZAI_EXPORTER_SKIP_TLS_VERIFICATION", false), "Whether to to skip TLS verification")
```

一些命令行参数

```
// metric only
registry := prometheus.NewRegistry()
if !*metricsOnly {
    registry = prometheus.DefaultRegisterer.(*prometheus.Registry)
}

// exporter
exp := exporter.NewYunzaiExporter(exporter.Option{
    Registry: registry,
    Oracle: database.NewOracleClient(database.BaseClient{
        ClientURL: *oracleURL,
        Username: *oracleUserName,
        Password: *oraclePassword,
    }),
    Namespace: *exporterNamespace,
    SchoolInfo: exporter.SchoolInfo{
        Latitude: *latitude,
        Longitude: *longitude,
        Gateway: *gateway,
        Name: *name,
        Icon: *icon,
    },
    BuildInfo: exporter.BuildInfo{
        Version: BuildVersion,
        CommitSha: BuildCommitSha,
        Date: BuildDate,
    },
    MetricsPath: *metricsPath,
    ClientCertificates: tlsClientCertificates,
    CaCertificates: tlsCaCertificates,
    SkipTLSVerification: *skipTLSVerification,
})
exp.AddTotalScrapes()
exp.AddScrapeDuration()
exp.AddTargetScrapeRequestErrors()
exp.AddMetrics()
exp.RegisterMetrics()
exp.RegisterBuildInfo()
exp.RegisterPosition()
exp.InstallDescriptions()
// HttpServer
exp.AddHTTPService()
exp.AddRootHTTPHandler()
exp.AddHealthHTTPHandler()
exp.AddScrapeHTTPHandler()
exp.AddErrorCounterHTTPHandler()
logrus.Infof("Providing metrics at %s", *listenAddress, *metricsPath)
if *tlsServerCertFile != "" && *tlsServerKeyFile != "" {
    logrus.Debugf("Bind as TLS using cert %s and key %s", *tlsServerCertFile, *tlsServerKeyFile)
    logrus.Fatal(http.ListenAndServeTLS(*listenAddress, *tlsServerCertFile, *tlsServerKeyFile, exp))
} else {
    logrus.Fatal(http.ListenAndServe(*listenAddress, exp))
}
}
```

prometheus注册对象  
不包含go指标的注册对象

自定义 exporter  
需要实现两个 prometheus 接口  
稍后会看

一系列处理

启动 http 服务

main函数主要就是构造exporter对象然后启动http服务



```
45
44 // Exporter exporter
43 type Exporter struct {
42   Option
41 }
40
39 // Option ...
38 type Option struct {
37   mutex *sync.Mutex
36   mux http.ServeMux
35   SchoolInfo SchoolInfo
34   CasPath string
33   Oracle database.DbClient
32   Namespace string
31   MetricsPath string
30   totalScrapes prometheus.Counter
29   scrapeDuration prometheus.Summary
28   targetScrapeRequestErrors prometheus.Counter
27   metricDescriptions map[string]*prometheus.Desc
26   metricMapCounters map[string]MetricHandler
25   metricMapGauges map[string]MetricHandler
24   metricMapSummary map[string]MetricHandler
23   metricMapHistogram map[string]MetricHandler
22   ClientCertificates []tls.Certificate
21   CaCertificates *x509.CertPool
20   SkipTLSVerification bool
19   Registry *prometheus.Registry
18   BuildInfo BuildInfo
17 }
16
15 // NewYunzaiExporter ...
14 func NewYunzaiExporter(options Option) *Exporter {
13   e := &Exporter{
12     options,
11   }
10   e.mutex = &sync.Mutex{}
9   return e
8 }
7
```

exporter 结构体中加入了同步锁，封装好的oracle客户端比较重要的是 **metricDescriptions**和 **metricMap**这几个指标map

作为prometheus的exporter需要实现两个接口,一个是输入描述的 **Describe**一个是收集数据的 **Collect**

```

47 // Collect 实现prometheus接口 放入指标信息
48 func (e *Exporter) Collect(ch chan<- prometheus.Metric) {
49     // 锁住
50     e.mutex.Lock()
51     // 解锁
52     defer e.mutex.Unlock()
53
54     // 总数
55     e.totalScrapes.Inc()
56     // 开始时间
57     startTime := time.Now()
58
59     // 服务可用状态
60     var up float64 = 1
61     var down float64 = 0
62     // 测试链接
63     e.Oracle.Connect()
64     logrus.Debugln("Ping to oracle")
65     canConnect := e.Oracle.Ping()
66     if canConnect {
67         e.RegisterConstMetricGauge(ch, "up", up)
68     } else {
69         e.RegisterConstMetricGauge(ch, "up", down)
70     }
71     e.Oracle.DisConnect()
72
73     // 注册指标
74     for _, metricHandler := range e.metricMapGauges {
75         metricHandler(ch, e)
76     }
77
78     for _, metricHandler := range e.metricMapSummary {
79         metricHandler(ch, e)
80     }
81
82     for _, metricHandler := range e.metricMapCounters {
83         metricHandler(ch, e)
84     }
85
86     for _, metricHandler := range e.metricMapHistogram {
87         metricHandler(ch, e)
88     }
89
90     // 结束时间获取时间间隔
91     took := time.Since(startTime).Seconds()
92     e.scrapeDuration.Observe(took)
93     e.RegisterConstMetricGauge(ch, "exporter_last_scrape_duration_seconds", took)
94 }
95
96 // Describe 注册描述
97 func (e *Exporter) Describe(ch chan<- *prometheus.Desc) {
98     for _, desc := range e.metricDescriptions {
99         ch <- desc
100     }
101     for k := range e.metricMapGauges {
102         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
103     }
104     for k := range e.metricMapCounters {
105         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
106     }
107     for k := range e.metricMapHistogram {
108         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
109     }
110     for k := range e.metricMapSummary {
111         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
112     }
113
114     ch <- e.totalScrapes.Desc()
115     ch <- e.scrapeDuration.Desc()
116     ch <- e.targetScrapeRequestErrors.Desc()
117 }

```

Collect, 循环map然后将数据和字符串传入ch管道中

```
12 // Describe 注册描述
13 func (e *Exporter) Describe(ch chan<- *prometheus.Desc) {
14     for _, desc := range e.metricDescriptions {
15         ch <- desc
16     }
17     for k := range e.metricMapGauges {
18         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
19     }
20     for k := range e.metricMapCounters {
21         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
22     }
23     for k := range e.metricMapHistogram {
24         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
25     }
26     for k := range e.metricMapSummary {
27         ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
28     }
29     ch <- e.totalScrapes.Desc()
30     ch <- e.scrapeDuration.Desc()
31     ch <- e.targetScrapeRequestErrors.Desc()
32 }
33 }
34 }
35 }
36 }
```

Describe相同都是向ch放入构建好的 指针类型对象

接下来看map中到底是什么



```

12 import (
13     "encoding/json"
14     "io/ioutil"
15     "net/http"
16
17     "github.com/prometheus/client_golang/prometheus"
18     "github.com/sirupsen/logrus"
19 )
20
21 var (
22     // MetricGuage ...
23     MetricGuage = map[string]MetricHandler{
24         "user_total":          userTotal,
25         "teacher_total":       teacherTotal,
26         "student_total":       studentTotal,
27         "web_application_total": webApplicationTotal,
28         "mobile_application_total": mobileApplicationTotal,
29         "department_total":    departmentTotal,
30         "role_total":          roleTotal,
31         "login_total":         loginTotal,
32         "online_user_total":   onlineUserTotal,
33         "application_request_perday_total": applicationRequestPerdayTotal,
34     }
35 )
36
37 // userTotal 用户总量
38 func userTotal(ch chan<- prometheus.Metric, e *Exporter) {
39     e.Oracle.Connect()
40     defer e.Oracle.DisConnect()
41     logrus.Debugln("collect user_total mertric")
42     var SQL string = "SELECT COUNT(USER_ID) AS USERTOTAL FROM SPAUTH.BASE_USERS"
43     var USERTOTAL float64
44     if err := e.Oracle.Instance().QueryRow(SQL).Scan(&USERTOTAL); err == nil {
45         e.RegisterConstMetricGauge(ch, "user_total", USERTOTAL)
46     } else {
47         logrus.Fatal(err)
48     }
49 }
50

```

map为string对应的匿名函数,函数类型为MetricHandler

很好理解,将管道传入匿名函数,结构体传入匿名函数,然后利用结构体的oracleClient执行sql构建etric结构体传入ch中即可

## Over

整个流程非常简单,由于公司的exporter不可能开源,所以这里提供redis-exporter地址以供参考

[https://github.com/oliver006/redis\\_exporter](https://github.com/oliver006/redis_exporter)

2020结束,最后一篇简单的参考教程