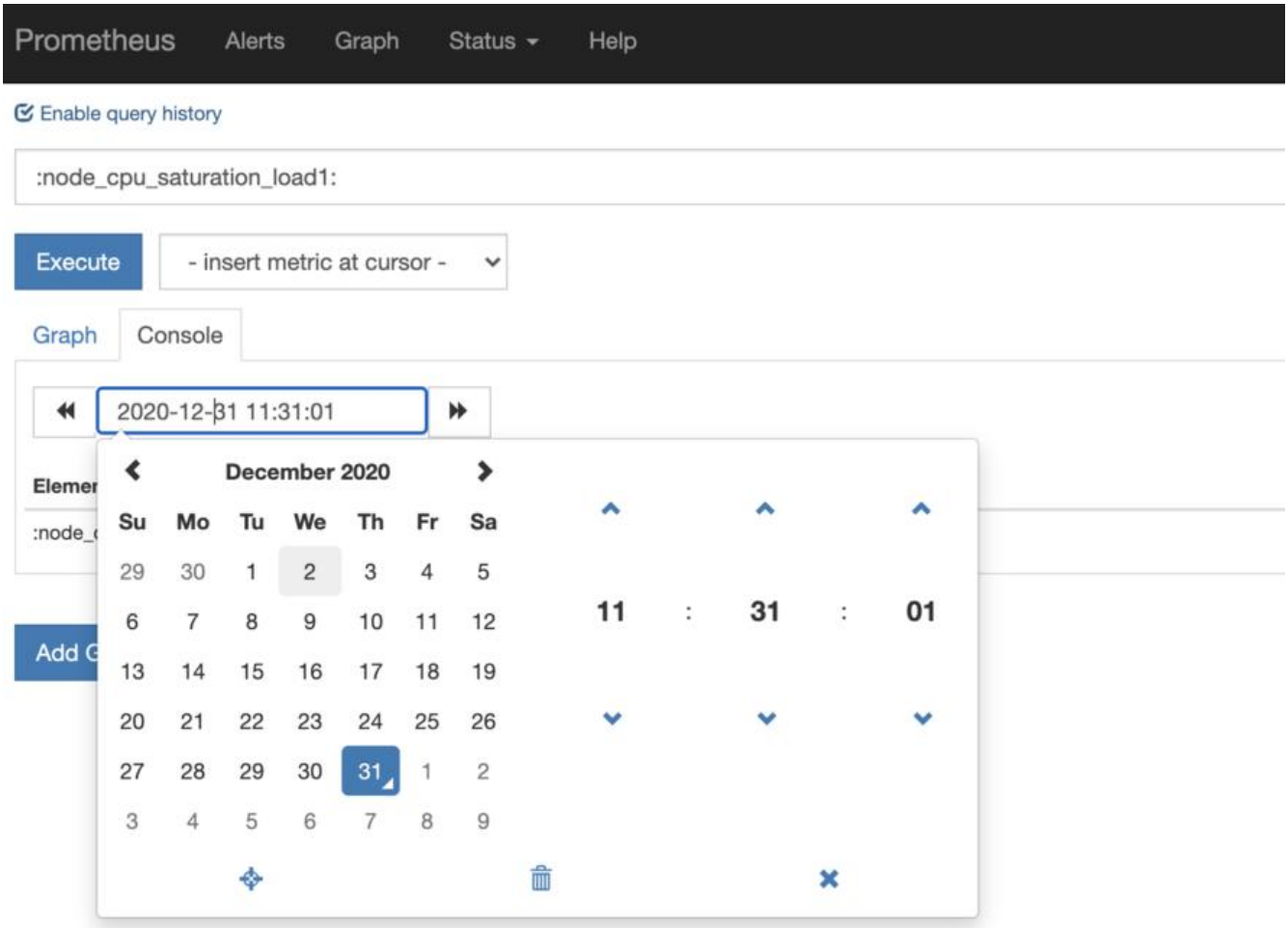
关于 Prometheus Exporter

作者：someone61489
原文链接：https://ld246.com/article/1609418091123
来源网站：链滴

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。

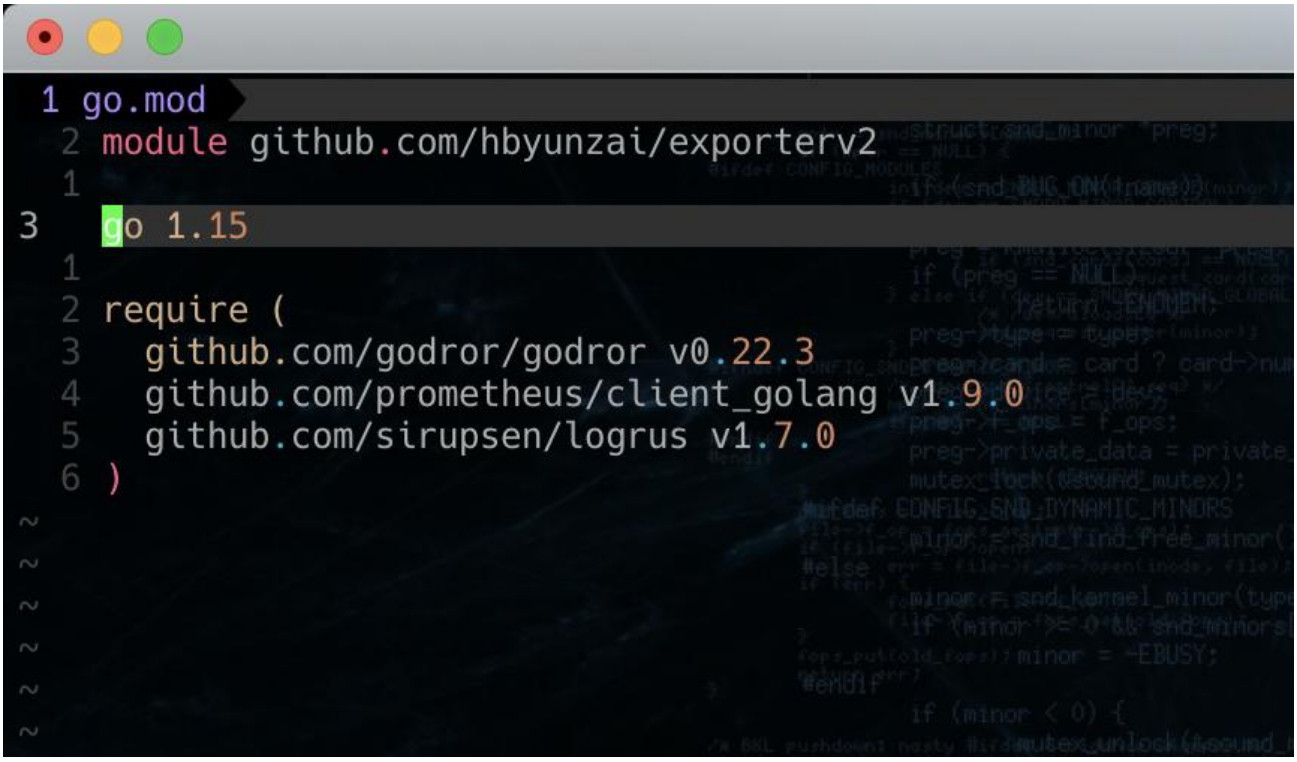
这个时候就出现了 prometheus-exporter用来放出自定义的业务指标，等待prometheus来此处收集。

prometheus exporter

原文链接：关于 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

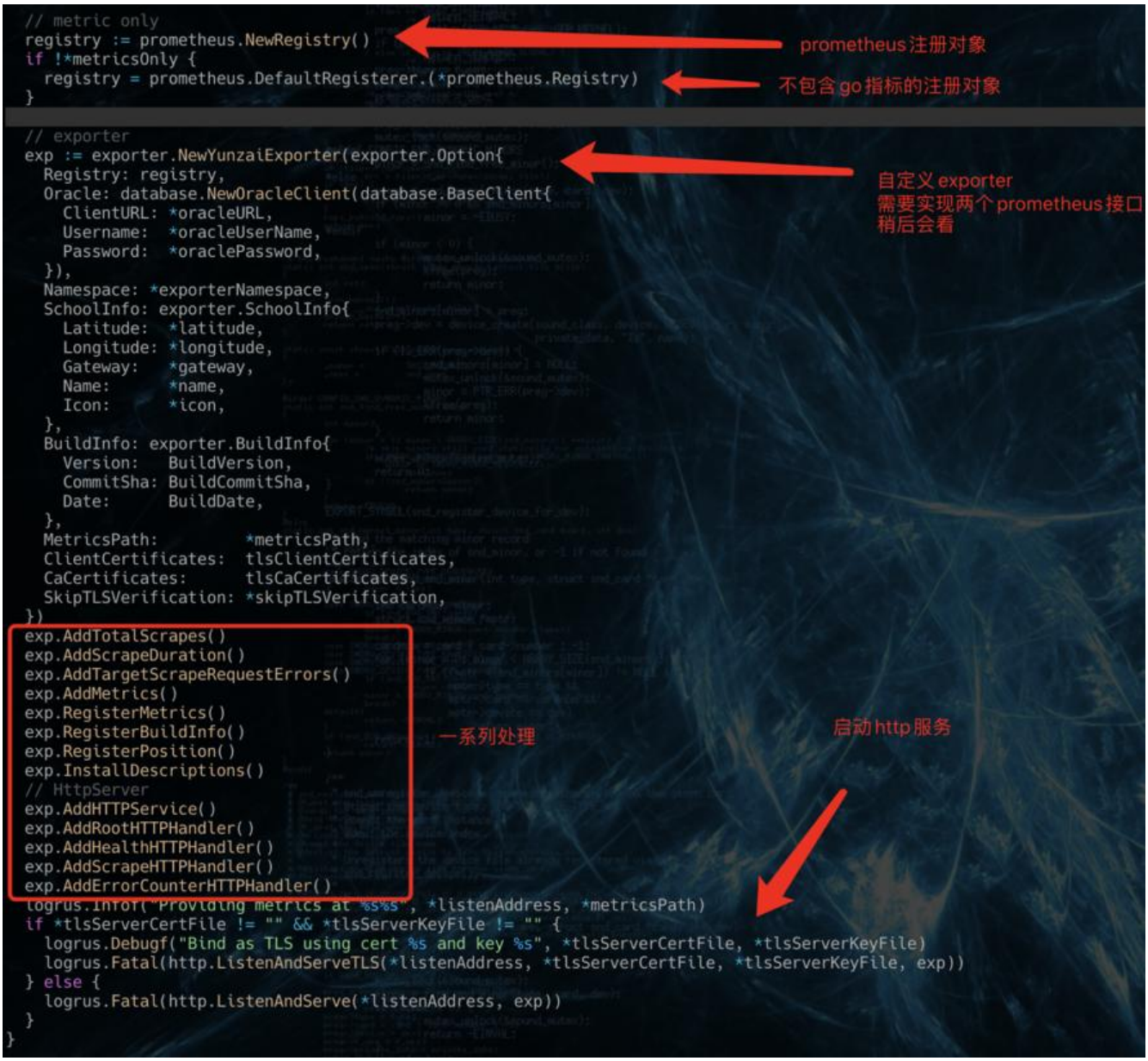


```
module github.com/hbyunzai/exporterv2

go 1.15

require (
    github.com/godror/godror v0.22.3
    github.com/prometheus/client_golang v1.9.0
    github.com/sirupsen/logrus v1.7.0
)
```

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

```
var (
    oracleURL          = flag.String("oracle.url", utils.GetEnvString("YUNZAI_EXPORTER_ORACLE_URL", ""), "Address of the oracle host")
    oracleUserName     = flag.String("oracle.username", utils.GetEnvString("YUNZAI_EXPORTER_ORACLE_USERNAME", ""), "Username of oracle")
    oraclePassword     = flag.String("oracle.password", utils.GetEnvString("YUNZAI_EXPORTER_ORACLE_PASSWORD", ""), "Password of oracle")
    exporterNamespace  = flag.String("exporter.namespace", utils.GetEnvString("YUNZAI_EXPORTER_NAMESPACE", "yunzai"), "Namespace of yunzai exporter")
    listenAddress      = flag.String("server.listen-address", utils.GetEnvString("YUNZAI_EXPORTER_SERVER_ADDRESS", ":10086"), "Address to listen on for web interface ar
    logDebug           = flag.Bool("exporter.log-debug", utils.GetEnvBool("YUNZAI_EXPORTER_LOG-DEBUG", true), "Log debug mode")
    longitude          = flag.String("exporter.school-longitude", utils.GetEnvString("YUNZAI_EXPORTER_LONGITUDE", ""), "exporter deploy longitude")
    latitude           = flag.String("exporter.school-latitude", utils.GetEnvString("YUNZAI_EXPORTER_LATITUDE", ""), "exporter deploy latitude")
    name               = flag.String("exporter.school-name", utils.GetEnvString("YUNZAI_EXPORTER_SCHOOL_NAME", ""), "exporter deploy school name")
    icon               = flag.String("exporter.school-icon", utils.GetEnvString("YUNZAI_EXPORTER_SCHOOL_ICON", ""), "exporter deploy school icon http path")
    gateway            = flag.String("exporter.school-gateway", utils.GetEnvString("YUNZAI_EXPORTER_SCHOOL_GATEWAY", "/backstage"), "gateway path (default /backstage)"
    logFormatter       = flag.String("exporter.log-formmater", utils.GetEnvString("YUNZAI_EXPORTER_LOG-FORMATTER", "text"), "Log formatter default is text,text or json
    metricsOnly        = flag.Bool("exporter.metrics.only", utils.GetEnvBool("YUNZAI_EXPORTER_METRICS_ONLY", false), "only has yunzai exporter metrics")
    metricsPath        = flag.String("exporter.metrics.path", utils.GetEnvString("YUNZAI_EXPORTER_METRICS_PATH", "/metrics"), "metrics path")
    tlsClientKeyFile   = flag.String("tls-client-key-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_CLIENT_KEY_FILE", ""), "Name of the client key file (including full
    tlsClientCertFile  = flag.String("tls-client-cert-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_CLIENT_CERT_FILE", ""), "Name of the client certificate file (inclu
    tlsCaCertFile      = flag.String("tls-ca-cert-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_CA_CERT_FILE", ""), "Name of the CA certificate file (including full pa
    tlsServerKeyFile   = flag.String("tls-server-key-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_SERVER_KEY_FILE", ""), "Name of the server key file (including full
    tlsServerCertFile  = flag.String("tls-server-cert-file", utils.GetEnvString("YUNZAI_EXPORTER_TLS_SERVER_CERT_FILE", ""), "Name of the server certificate file (inclu
    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



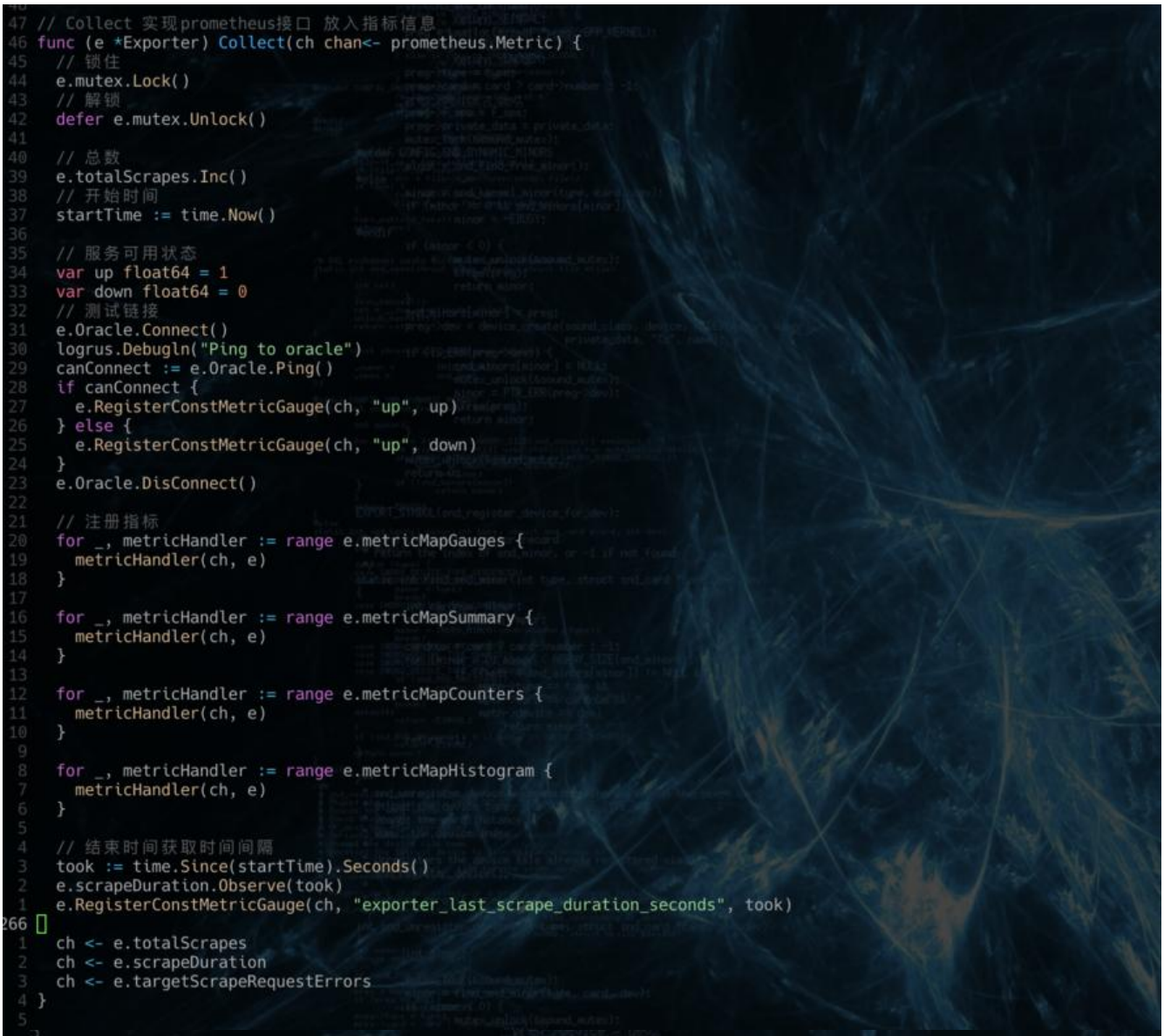
main函数主要就是构造exporter对象然后启动http服务

原文链接：关于 Prometheus Exporter

```go
// Exporter exporter
type Exporter struct {
  Option
}

// Option ...
type Option struct {
  mutex                    *sync.Mutex
  mux                      *http.ServeMux
  SchoolInfo               SchoolInfo
  CasPath                  string
  Oracle                   database.DbClient
  Namespace                string
  MetricsPath              string
  totalScrapes             prometheus.Counter
  scrapeDuration           prometheus.Summary
  targetScrapeRequestErrors prometheus.Counter
  metricDescriptions       map[string]*prometheus.Desc
  metricMapCounters        map[string]MetricHandler
  metricMapGauges          map[string]MetricHandler
  metricMapSummary         map[string]MetricHandler
  metricMapHistogram       map[string]MetricHandler
  ClientCertificates       []tls.Certificate
  CaCertificates           *x509.CertPool
  SkipTLSVerification      bool
  Registry                 *prometheus.Registry
  BuildInfo                BuildInfo
}

// NewYunzaiExporter ...
func NewYunzaiExporter(options Option) *Exporter {
  e := &Exporter{
    options,
  }
  e.mutex = &sync.Mutex{}
  return e
}
```

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 metriMap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



```go
// Collect 实现prometheus接口 放入指标信息
func (e *Exporter) Collect(ch chan<- prometheus.Metric) {
  // 锁住
  e.mutex.Lock()
  // 解锁
  defer e.mutex.Unlock()

  // 总数
  e.totalScrapes.Inc()
  // 开始时间
  startTime := time.Now()

  // 服务可用状态
  var up float64 = 1
  var down float64 = 0
  // 测试链接
  e.Oracle.Connect()
  logrus.Debugln("Ping to oracle")
  canConnect := e.Oracle.Ping()
  if canConnect {
    e.RegisterConstMetricGauge(ch, "up", up)
  } else {
    e.RegisterConstMetricGauge(ch, "up", down)
  }
  e.Oracle.DisConnect()

  // 注册指标
  for _, metricHandler := range e.metricMapGauges {
    metricHandler(ch, e)
  }

  for _, metricHandler := range e.metricMapSummary {
    metricHandler(ch, e)
  }

  for _, metricHandler := range e.metricMapCounters {
    metricHandler(ch, e)
  }

  for _, metricHandler := range e.metricMapHistogram {
    metricHandler(ch, e)
  }

  // 结束时间获取时间间隔
  took := time.Since(startTime).Seconds()
  e.scrapeDuration.Observe(took)
  e.RegisterConstMetricGauge(ch, "exporter_last_scrape_duration_seconds", took)

  ch <- e.totalScrapes
  ch <- e.scrapeDuration
  ch <- e.targetScrapeRequestErrors
}

// Describe 注册描述
func (e *Exporter) Describe(ch chan<- *prometheus.Desc) {
  for _, desc := range e.metricDescriptions {
    ch <- desc
  }
  for k := range e.metricMapGauges {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  for k := range e.metricMapCounters {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  for k := range e.metricMapHistogram {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  for k := range e.metricMapSummary {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  ch <- e.totalScrapes.Desc()
  ch <- e.scrapeDuration.Desc()
  ch <- e.targetScrapeRequestErrors.Desc()
}
```

原文链接：关于 Prometheus Exporter

Collect，循环map然后将数据和字符串传入ch管道中

```go
// Describe 注册描述
func (e *Exporter) Describe(ch chan<- *prometheus.Desc) {
  for _, desc := range e.metricDescriptions {
    ch <- desc
  }
  for k := range e.metricMapGauges {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  for k := range e.metricMapCounters {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  for k := range e.metricMapHistogram {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  for k := range e.metricMapSummary {
    ch <- newMetricDescr(e.Namespace, k, k+" metric", nil)
  }

  ch <- e.totalScrapes.Desc()
  ch <- e.scrapeDuration.Desc()
  ch <- e.targetScrapeRequestErrors.Desc()
}
```

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

```go
import (
  "encoding/json"
  "io/ioutil"
  "net/http"

  "github.com/prometheus/client_golang/prometheus"
  "github.com/sirupsen/logrus"
)

var (
  // MetricGuage ...
  MetricGuage = map[string]MetricHandler{
    "user_total":                    userTotal,
    "teacher_total":                 teacherTotal,
    "student_total":                 studentTotal,
    "web_application_total":         webApplicationTotal,
    "mobile_application_total":      mobileApplicationTotal,
    "department_total":              departmentTotal,
    "role_total":                    roleTotal,
    "login_total":                   loginTotal,
    "online_user_total":             onlineUserTotal,
    "application_request_perday_total": applicationRequestPerdayTotal,
  }
)

// userTotal 用户总量
func userTotal(ch chan<- prometheus.Metric, e *Exporter) {
  e.Oracle.Connect()
  defer e.Oracle.DisConnect()
  logrus.Debugln("collect user_total mertric")
  var SQL string = "SELECT COUNT(USER_ID) AS USERTOTAL FROM SPAUTH.BASE_USERS"
  var USERTOTAL float64
  if err := e.Oracle.Instance().QueryRow(SQL).Scan(&USERTOTAL); err == nil {
    e.RegisterConstMetricGauge(ch, "user_total", USERTOTAL)
  } else {
    logrus.Fatal(err)
  }
}
```

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

2020结束，最后一篇简单的参考教程