Go 小 工 具 分 享

目录页

目录
CONTENTS

目录页

Caddy

21945

Caddy是一款采用Golang编写的全平台的Web服务器

配置

特性

一、易于配置

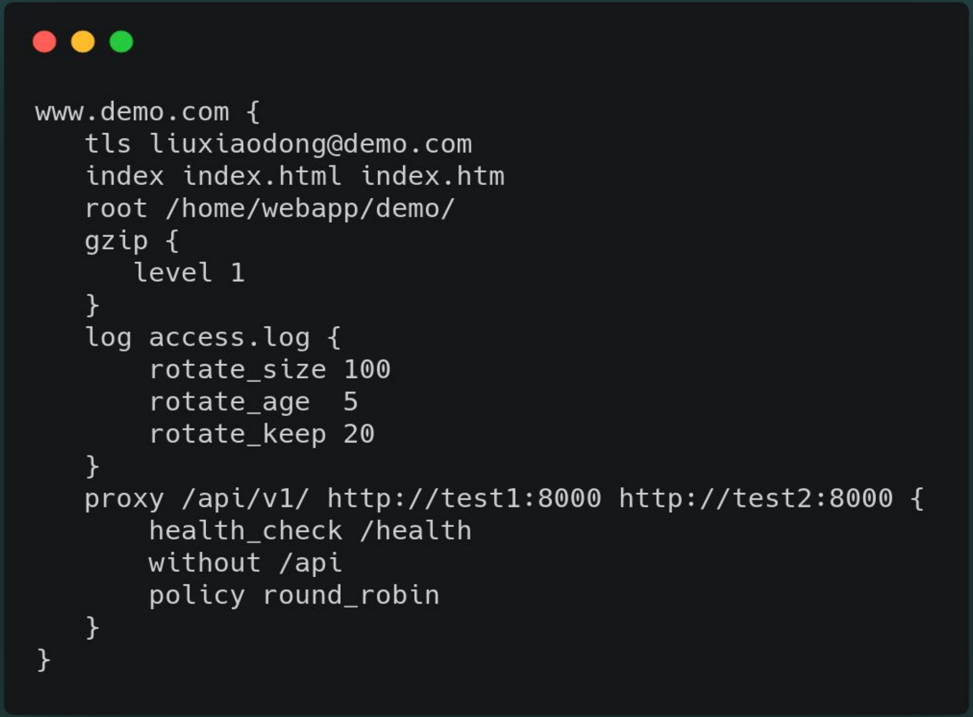
二、自动https支持(Let's Encrypt)

三、默认支持Http2

四、多Host支持

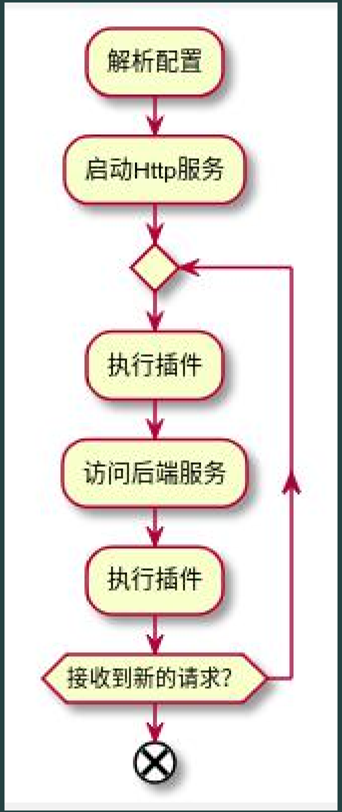
五、插件丰富，易于拓展

配置

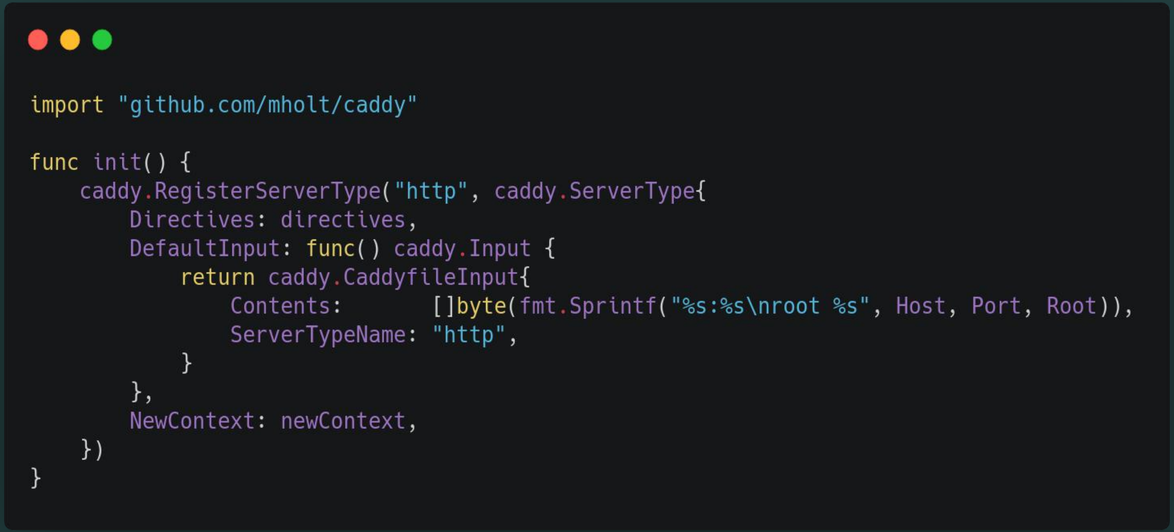


```
www.demo.com {
   tls liuxiaodong@demo.com
   index index.html index.htm
   root /home/webapp/demo/
   gzip {
      level 1
   }
   log access.log {
       rotate_size 100
       rotate_age  5
       rotate_keep 20
   }
   proxy /api/v1/ http://test1:8000 http://test2:8000 {
       health_check /health
       without /api
       policy round_robin
   }
}
```

原理

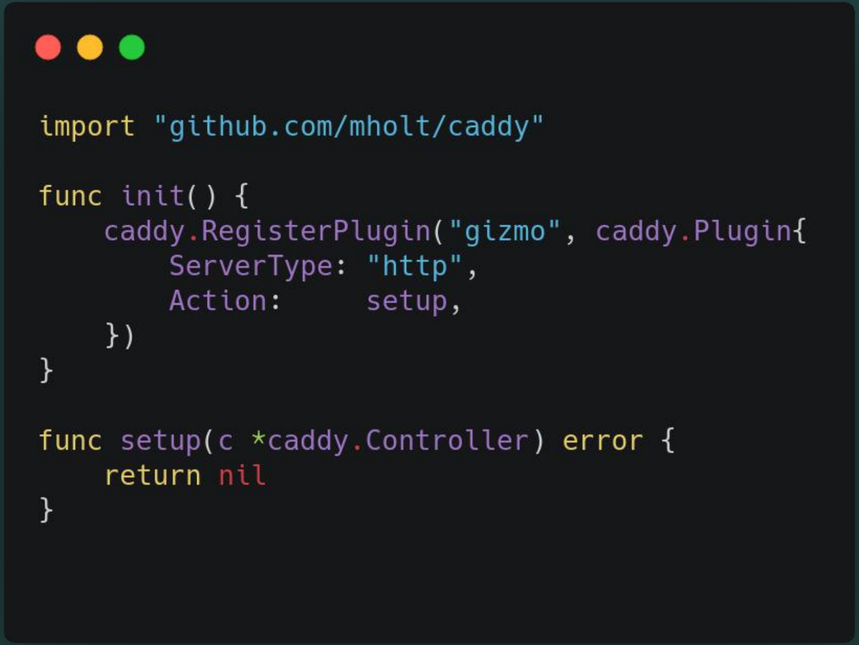


插件（Server）



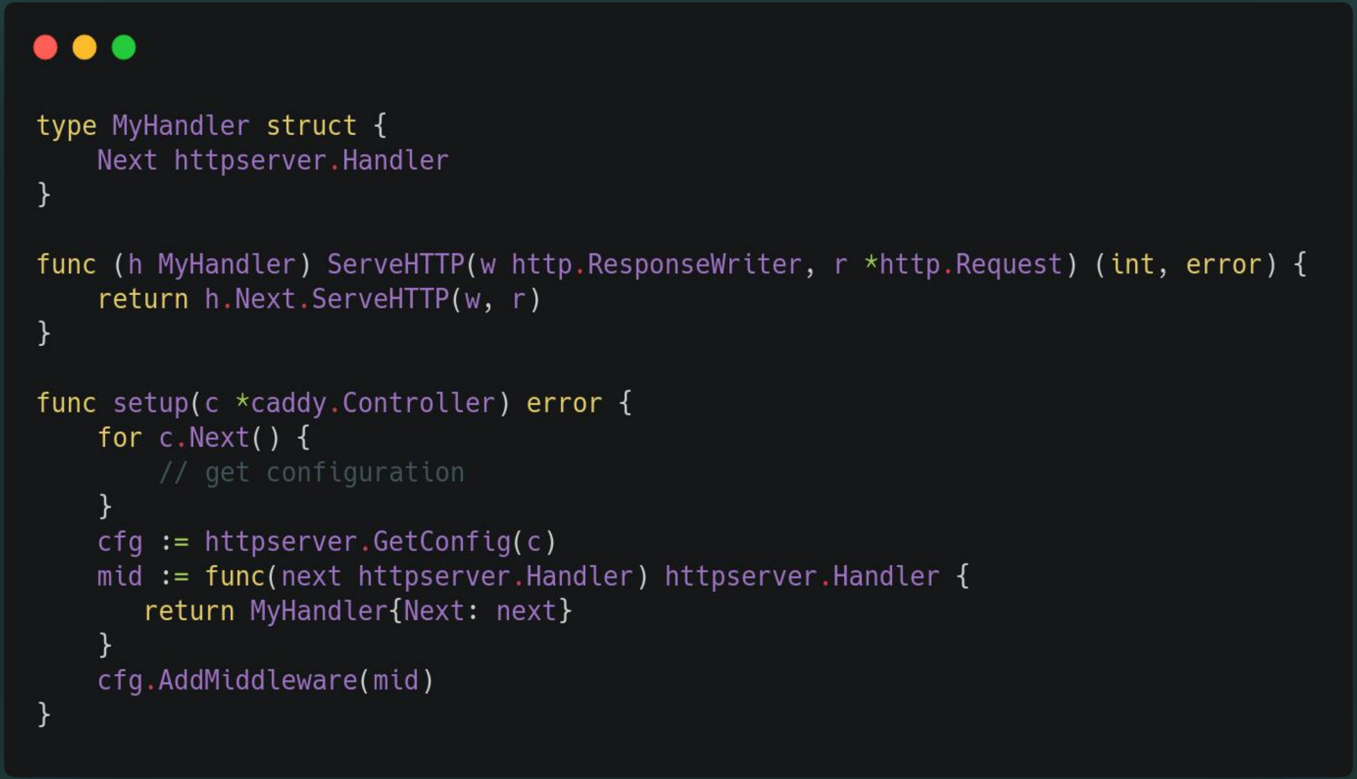
```go
import "github.com/mholt/caddy"

func init() {
    caddy.RegisterServerType("http", caddy.ServerType{
        Directives: directives,
        DefaultInput: func() caddy.Input {
            return caddy.CaddyfileInput{
                Contents:       []byte(fmt.Sprintf("%s:%s\nroot %s", Host, Port, Root)),
                ServerTypeName: "http",
            }
        },
        NewContext: newContext,
    })
}
```

插件（Directive）



```go
import "github.com/mholt/caddy"

func init() {
    caddy.RegisterPlugin("gizmo", caddy.Plugin{
        ServerType: "http",
        Action:     setup,
    })
}

func setup(c *caddy.Controller) error {
    return nil
}
```

插件（Middleware）



```go
type MyHandler struct {
    Next httpserver.Handler
}

func (h MyHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) (int, error) {
    return h.Next.ServeHTTP(w, r)
}

func setup(c *caddy.Controller) error {
    for c.Next() {
        // get configuration
    }
    cfg := httpserver.GetConfig(c)
    mid := func(next httpserver.Handler) httpserver.Handler {
       return MyHandler{Next: next}
    }
    cfg.AddMiddleware(mid)
}
```

目录页

Traefik^{22467}

traefik是一款采用Golang编写的CloudNative网关

配置

特性

一、https支持(Let's Encrypt)

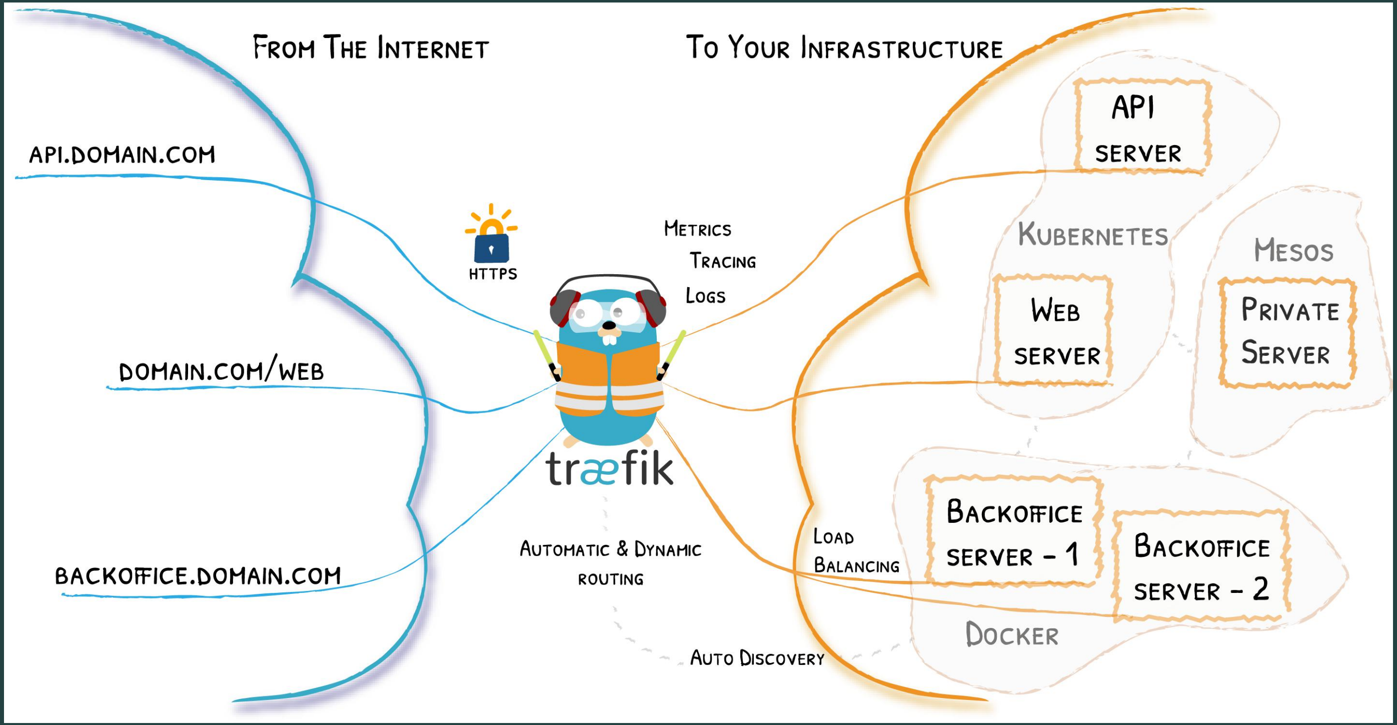
二、支持多种配置源（文件、etcd、k8s、consul等）

三、性能监测（Prometheus，Datadog，Rest，InfluxDB）

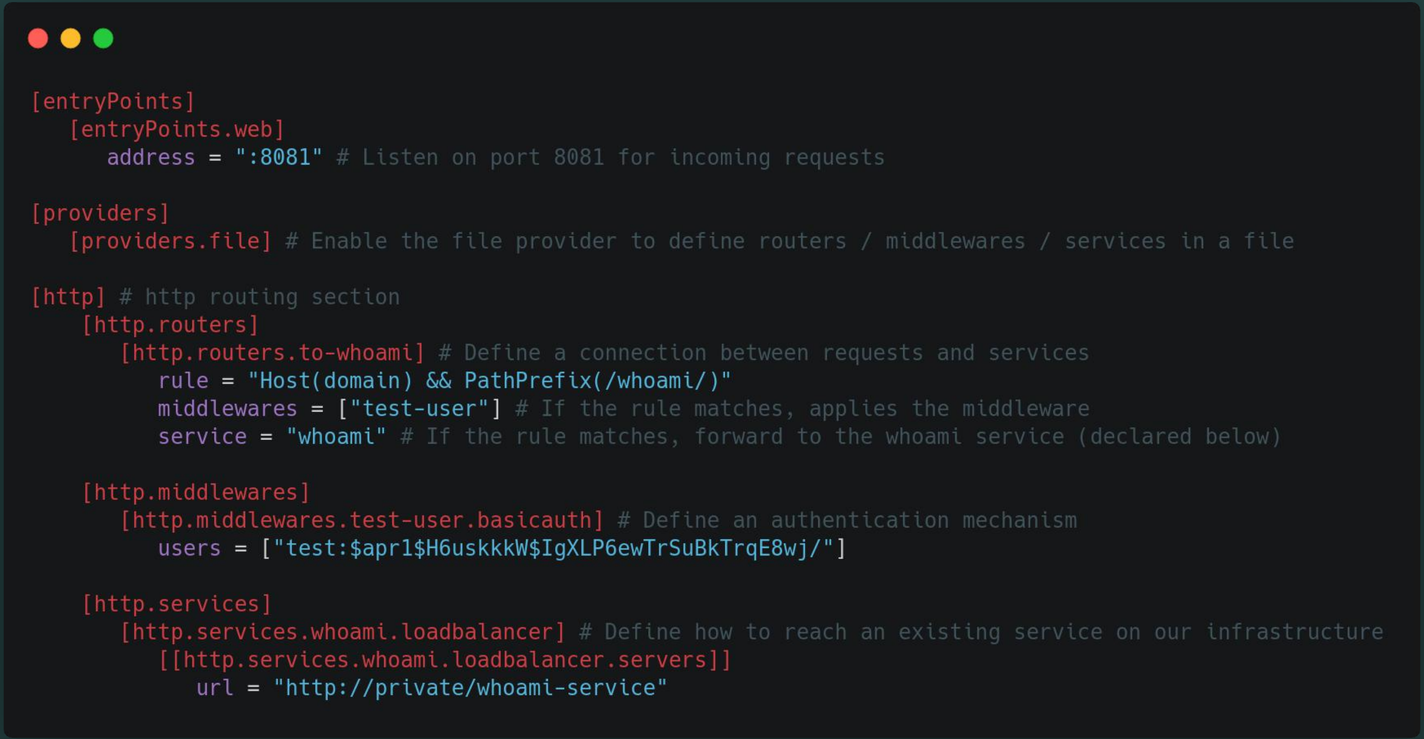
四、拥有多种负载均衡策略

五、插件丰富，易于拓展

配置

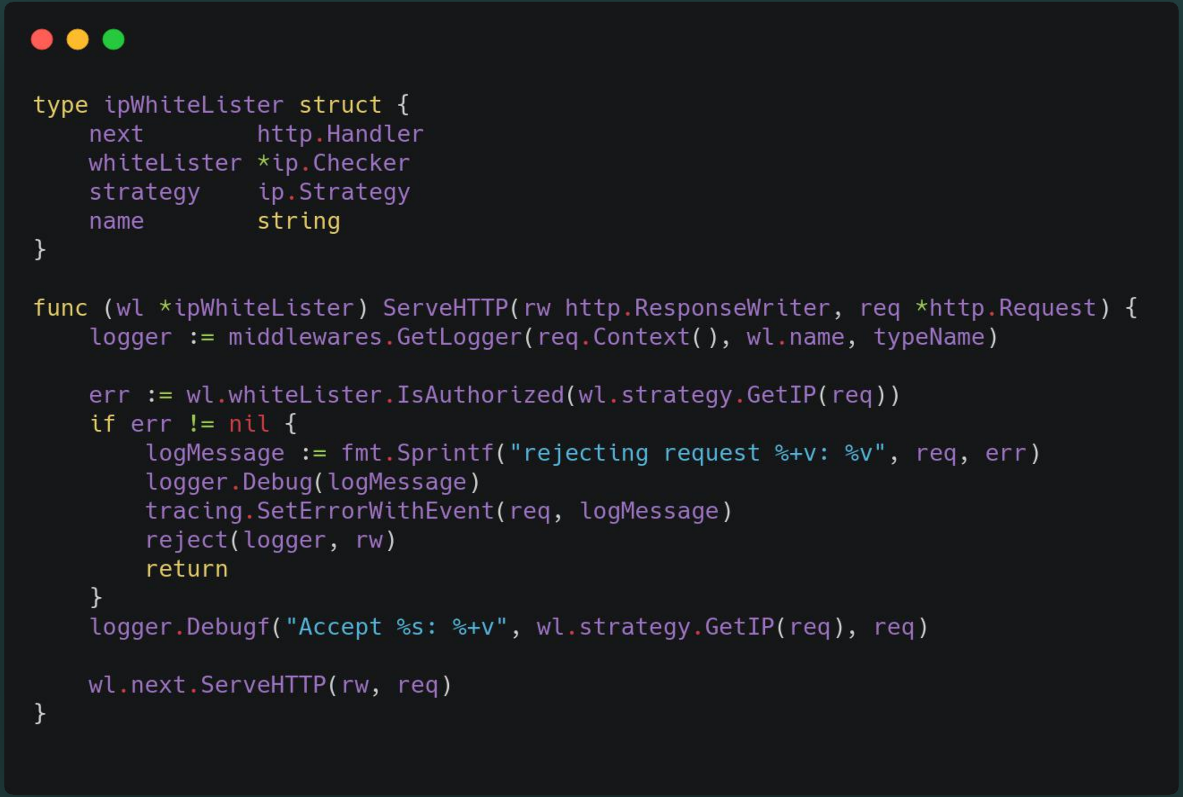

From The Internet
To Your Infrastructure
API.DOMAIN.COM
DOMAIN.COM/WEB
BACKOFFICE.DOMAIN.COM
HTTPS
Metrics
Tracing
Logs
træfik
Automatic & Dynamic
routing
Auto Discovery
Load
Balancing
API
SERVER
Kubernetes
Web
SERVER
Mesos
Private
Server
Backoffice
SERVER - 1
Backoffice
SERVER - 2
Docker

配置



```
[entryPoints]
  [entryPoints.web]
    address = ":8081" # Listen on port 8081 for incoming requests

[providers]
  [providers.file] # Enable the file provider to define routers / middlewares / services in a file

[http] # http routing section
    [http.routers]
      [http.routers.to-whoami] # Define a connection between requests and services
        rule = "Host(domain) && PathPrefix(/whoami/)"
        middlewares = ["test-user"] # If the rule matches, applies the middleware
        service = "whoami" # If the rule matches, forward to the whoami service (declared below)

    [http.middlewares]
      [http.middlewares.test-user.basicauth] # Define an authentication mechanism
        users = ["test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/"]

    [http.services]
      [http.services.whoami.loadbalancer] # Define how to reach an existing service on our infrastructure
        [[http.services.whoami.loadbalancer.servers]]
          url = "http://private/whoami-service"
```

插件



```go
type ipWhiteLister struct {
    next        http.Handler
    whiteLister *ip.Checker
    strategy    ip.Strategy
    name        string
}

func (wl *ipWhiteLister) ServeHTTP(rw http.ResponseWriter, req *http.Request) {
    logger := middlewares.GetLogger(req.Context(), wl.name, typeName)

    err := wl.whiteLister.IsAuthorized(wl.strategy.GetIP(req))
    if err != nil {
        logMessage := fmt.Sprintf("rejecting request %+v: %v", req, err)
        logger.Debug(logMessage)
        tracing.SetErrorWithEvent(req, logMessage)
        reject(logger, rw)
        return
    }
    logger.Debugf("Accept %s: %+v", wl.strategy.GetIP(req), req)

    wl.next.ServeHTTP(rw, req)
}
```

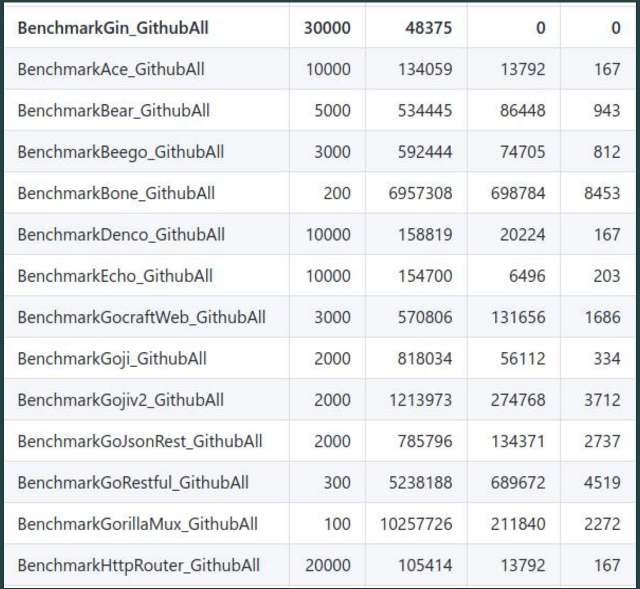
目录页

Gin

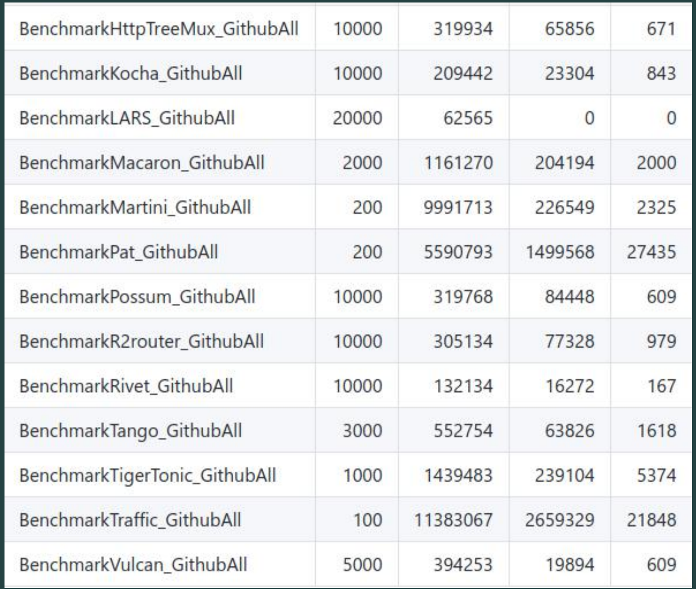
27740

gin是一款golang web框架、采用httprouter作为路由器，性能非常好

性能比较



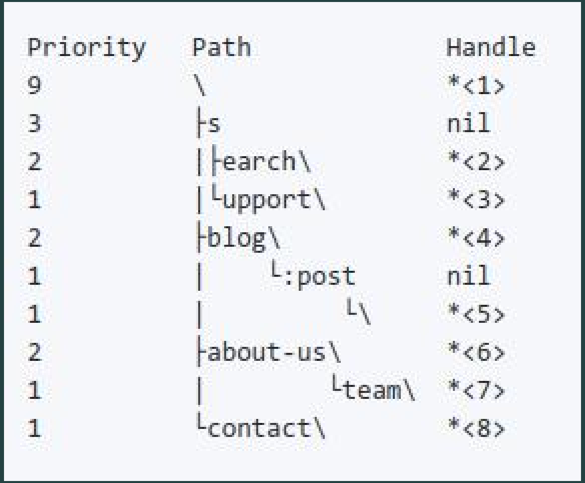
BenchmarkGin_GithubAll	**30000**	**48375**	**0**	**0**
BenchmarkAce_GithubAll	10000	134059	13792	167
BenchmarkBear_GithubAll	5000	534445	86448	943
BenchmarkBeego_GithubAll	3000	592444	74705	812
BenchmarkBone_GithubAll	200	6957308	698784	8453
BenchmarkDenco_GithubAll	10000	158819	20224	167
BenchmarkEcho_GithubAll	10000	154700	6496	203
BenchmarkGocraftWeb_GithubAll	3000	570806	131656	1686
BenchmarkGoji_GithubAll	2000	818034	56112	334
BenchmarkGojiv2_GithubAll	2000	1213973	274768	3712
BenchmarkGoJsonRest_GithubAll	2000	785796	134371	2737
BenchmarkGoRestful_GithubAll	300	5238188	689672	4519
BenchmarkGorillaMux_GithubAll	100	10257726	211840	2272
BenchmarkHttpRouter_GithubAll	20000	105414	13792	167



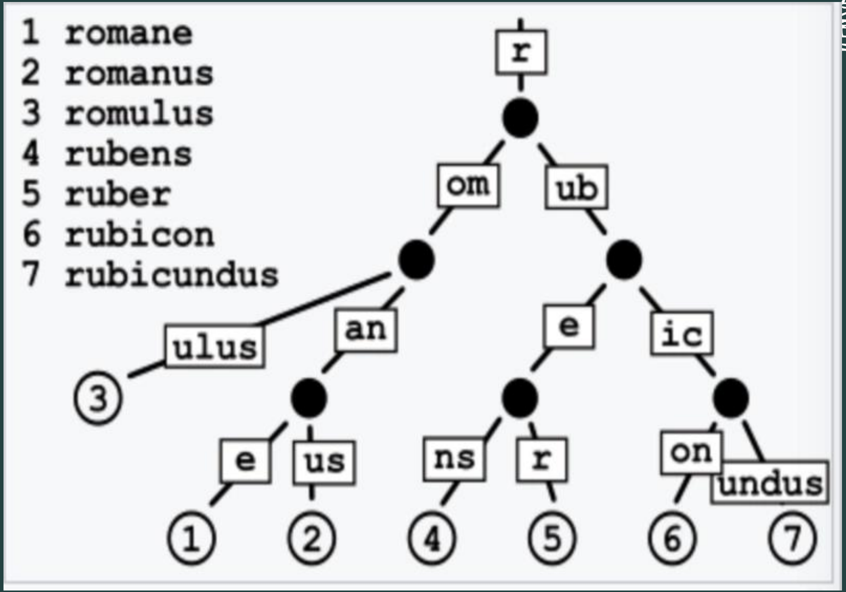
BenchmarkHttpTreeMux_GithubAll	10000	319934	65856	671
BenchmarkKocha_GithubAll	10000	209442	23304	843
BenchmarkLARS_GithubAll	20000	62565	0	0
BenchmarkMacaron_GithubAll	2000	1161270	204194	2000
BenchmarkMartini_GithubAll	200	9991713	226549	2325
BenchmarkPat_GithubAll	200	5590793	1499568	27435
BenchmarkPossum_GithubAll	10000	319768	84448	609
BenchmarkR2router_GithubAll	10000	305134	77328	979
BenchmarkRivet_GithubAll	10000	132134	16272	167
BenchmarkTango_GithubAll	3000	552754	63826	1618
BenchmarkTigerTonic_GithubAll	1000	1439483	239104	5374
BenchmarkTraffic_GithubAll	100	11383067	2659329	21848
BenchmarkVulcan_GithubAll	5000	394253	19894	609

(1): Total Repetitions achieved in constant time, higher means more confident result
(2): Single Repetition Duration (ns/op), lower is better
(3): Heap Memory (B/op), lower is better
(4): Average Allocations per Repetition (allocs/op), lower is better

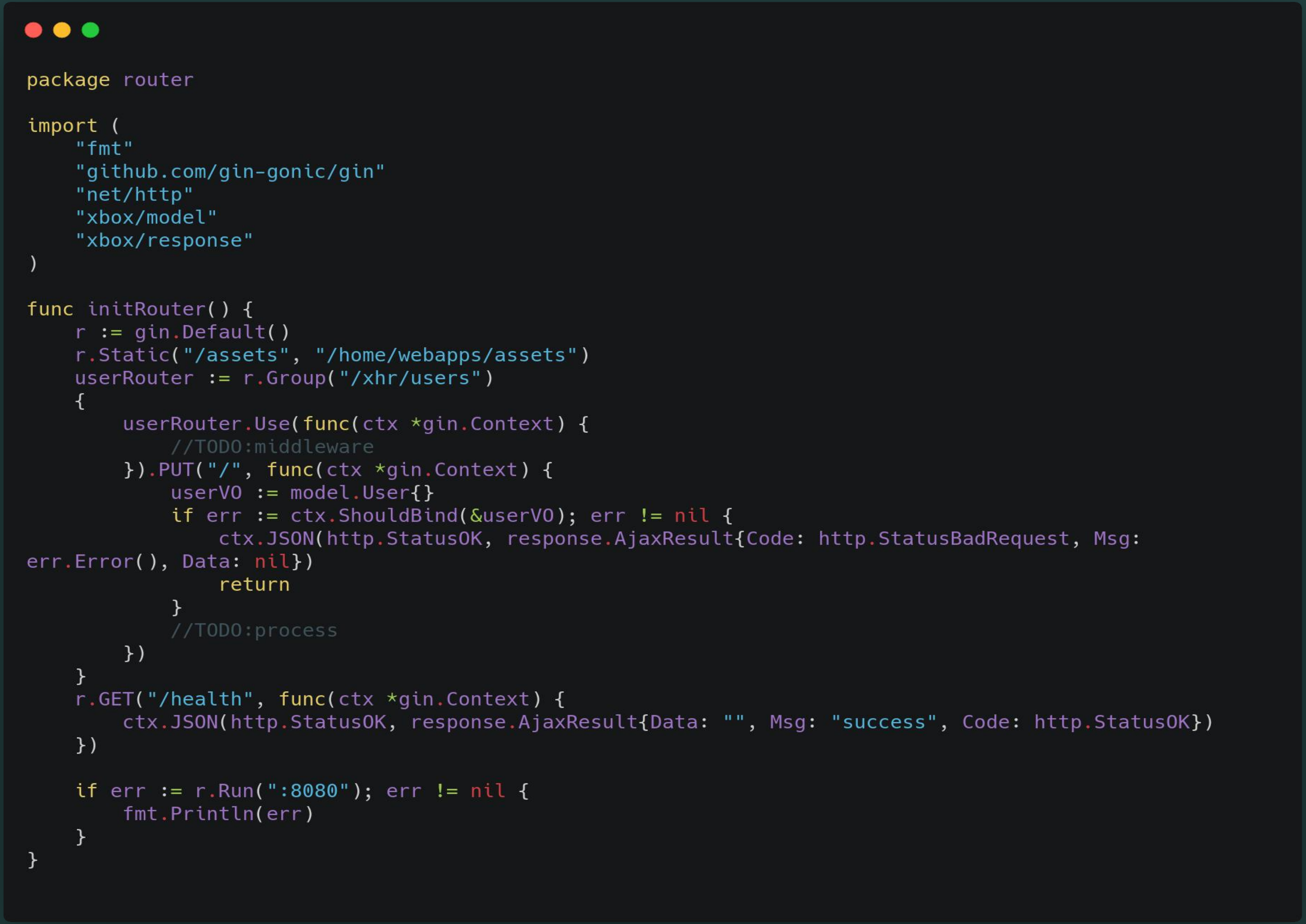
基数树



```
Priority   Path              Handle
9          \                 *<1>
3          ├s                nil
2          |├earch\          *<2>
1          |└upport\         *<3>
2          ├blog\            *<4>
1          |    └:post       nil
1          |         └\      *<5>
2          ├about-us\        *<6>
1          |        └team\   *<7>
1          └contact\         *<8>
```

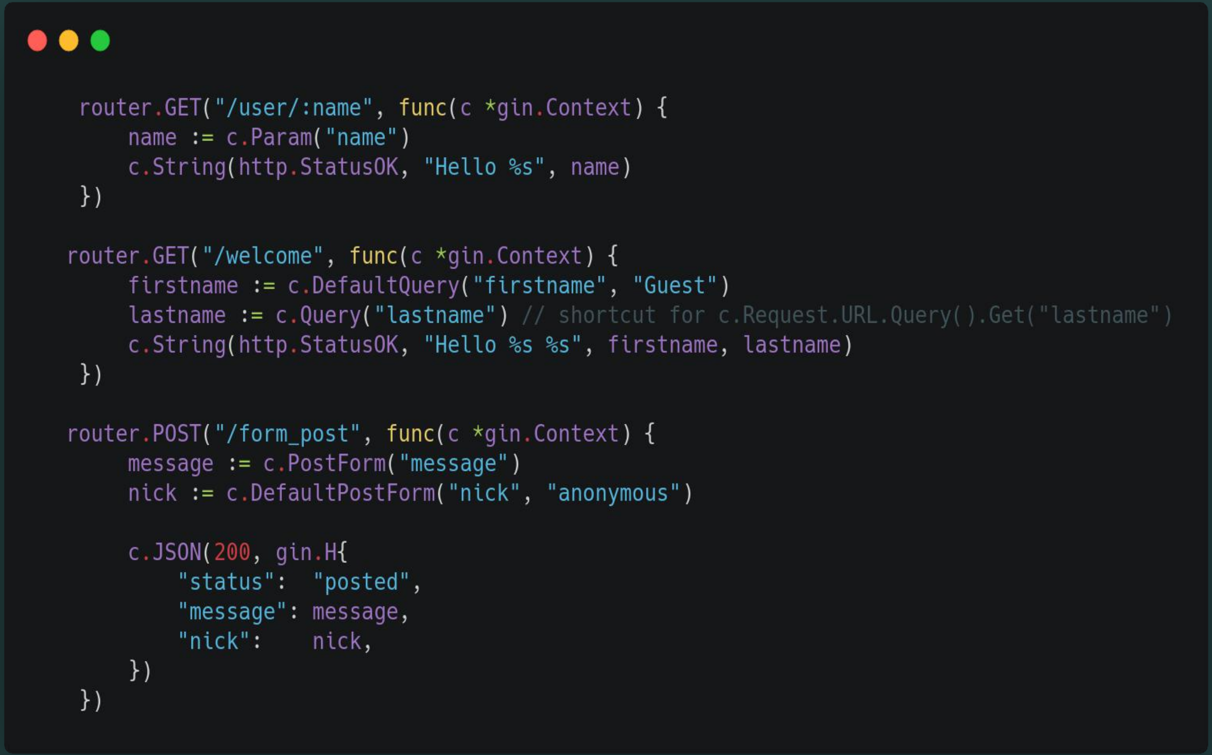


示例



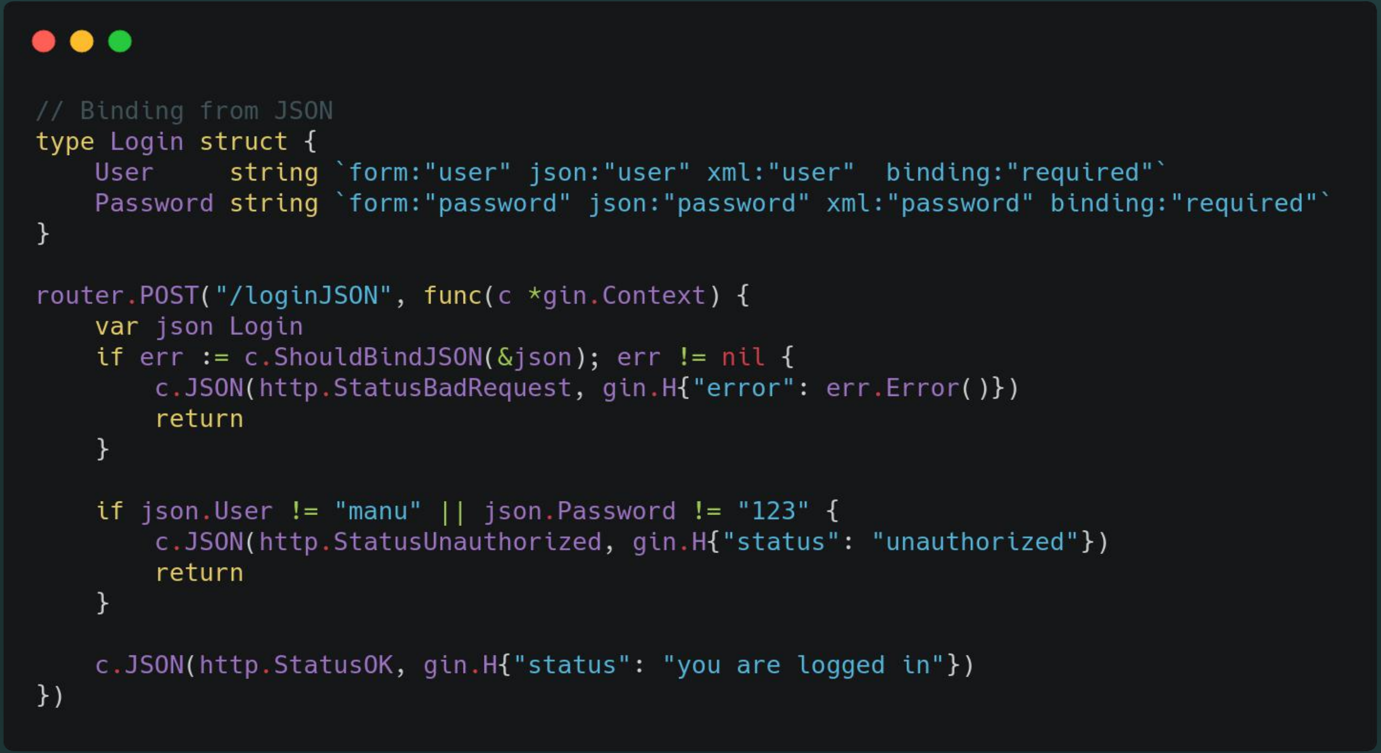
```go
package router

import (
    "fmt"
    "github.com/gin-gonic/gin"
    "net/http"
    "xbox/model"
    "xbox/response"
)

func initRouter() {
    r := gin.Default()
    r.Static("/assets", "/home/webapps/assets")
    userRouter := r.Group("/xhr/users")
    {
        userRouter.Use(func(ctx *gin.Context) {
            //TODO:middleware
        }).PUT("/", func(ctx *gin.Context) {
            userVO := model.User{}
            if err := ctx.ShouldBind(&userVO); err != nil {
                ctx.JSON(http.StatusOK, response.AjaxResult{Code: http.StatusBadRequest, Msg:
err.Error(), Data: nil})
                return
            }
            //TODO:process
        })
    }
    r.GET("/health", func(ctx *gin.Context) {
        ctx.JSON(http.StatusOK, response.AjaxResult{Data: "", Msg: "success", Code: http.StatusOK})
    })

    if err := r.Run(":8080"); err != nil {
        fmt.Println(err)
    }
}
```

示例



```go
 router.GET("/user/:name", func(c *gin.Context) {
     name := c.Param("name")
     c.String(http.StatusOK, "Hello %s", name)
 })

router.GET("/welcome", func(c *gin.Context) {
     firstname := c.DefaultQuery("firstname", "Guest")
     lastname := c.Query("lastname") // shortcut for c.Request.URL.Query().Get("lastname")
     c.String(http.StatusOK, "Hello %s %s", firstname, lastname)
 })

router.POST("/form_post", func(c *gin.Context) {
     message := c.PostForm("message")
     nick := c.DefaultPostForm("nick", "anonymous")

     c.JSON(200, gin.H{
         "status":  "posted",
         "message": message,
         "nick":    nick,
     })
 })
```

示例



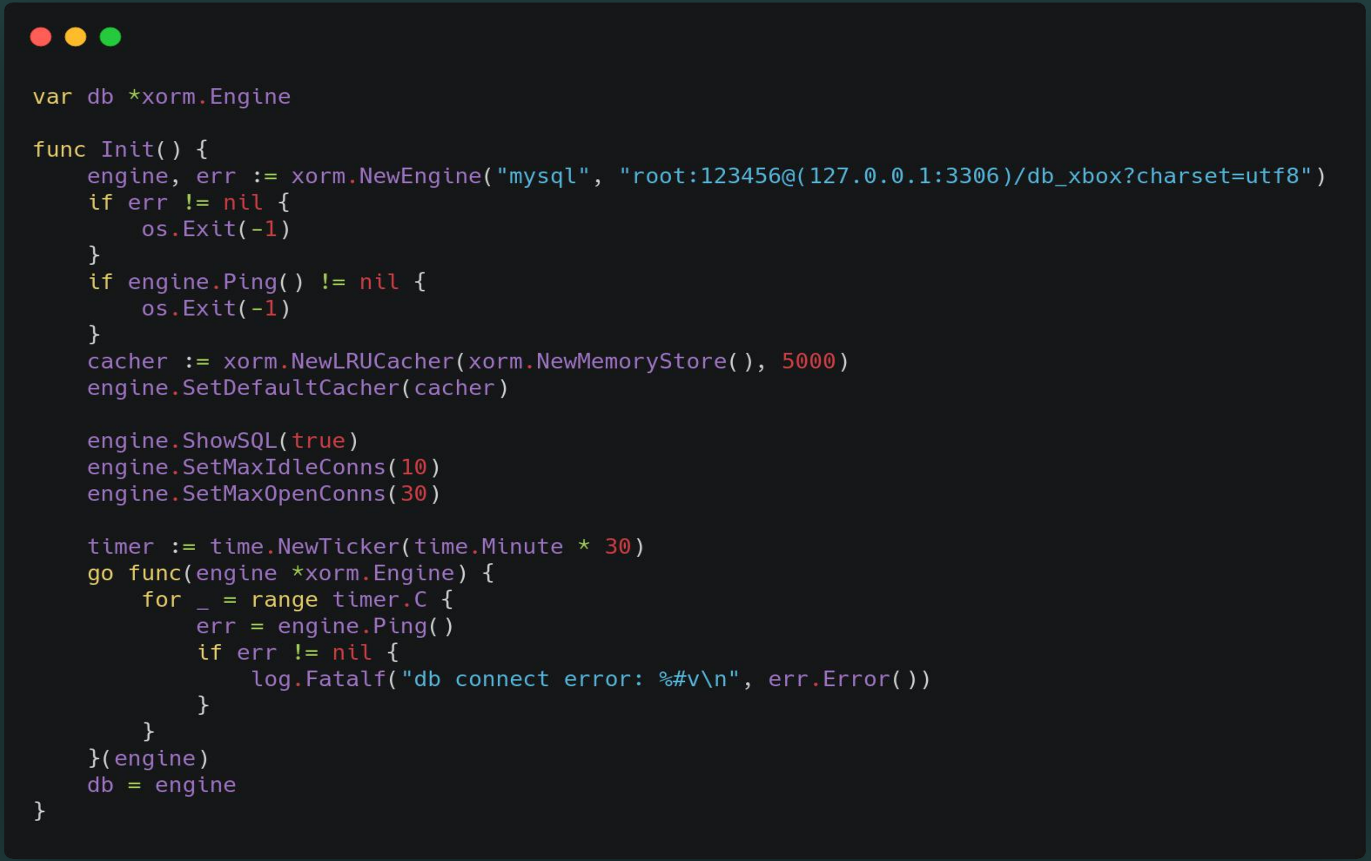
```go
// Binding from JSON
type Login struct {
    User     string `form:"user" json:"user" xml:"user"  binding:"required"`
    Password string `form:"password" json:"password" xml:"password" binding:"required"`
}

router.POST("/loginJSON", func(c *gin.Context) {
    var json Login
    if err := c.ShouldBindJSON(&json); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    if json.User != "manu" || json.Password != "123" {
        c.JSON(http.StatusUnauthorized, gin.H{"status": "unauthorized"})
        return
    }

    c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
})
```

目录页

Xorm

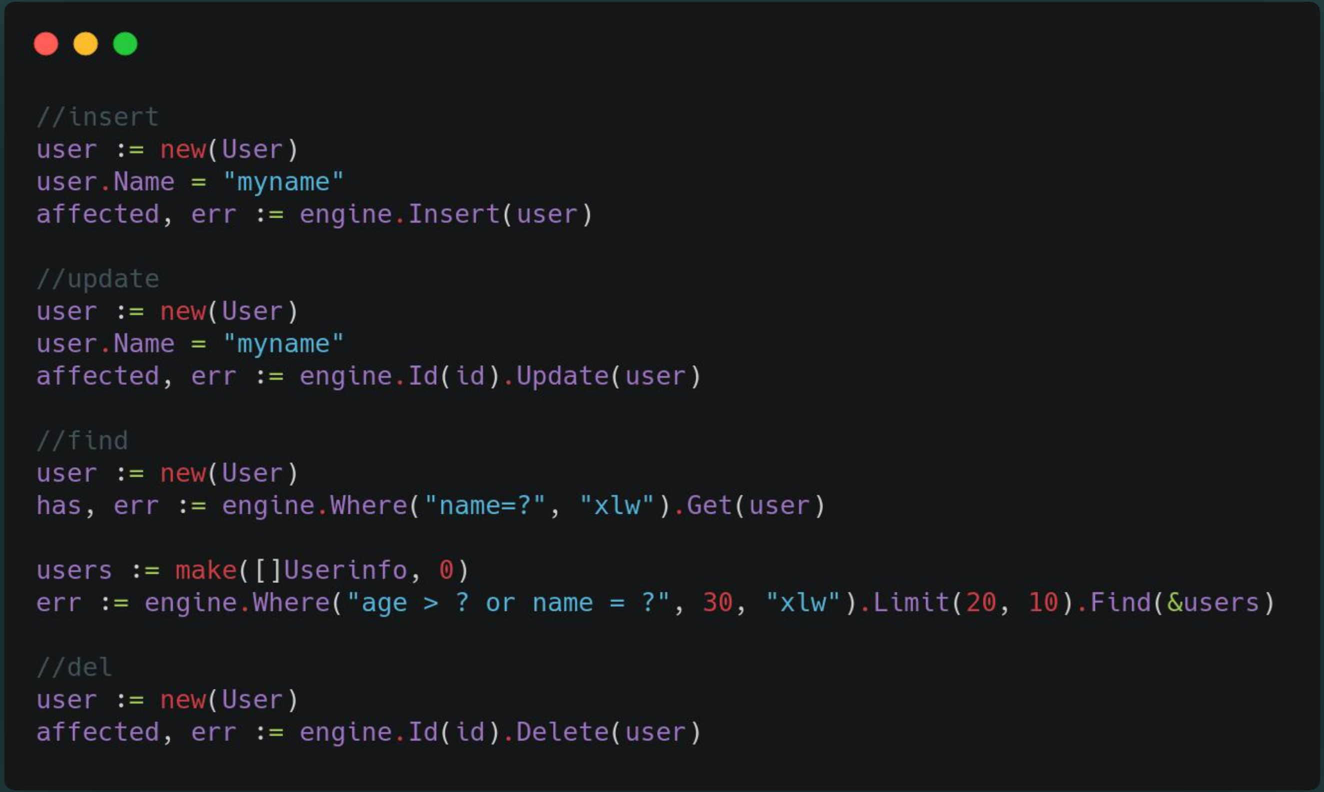
xorm是一个简单而强大的Go语言ORM库. 通过它可以使数据库操作非常简便。

配置



```go
var db *xorm.Engine

func Init() {
    engine, err := xorm.NewEngine("mysql", "root:123456@(127.0.0.1:3306)/db_xbox?charset=utf8")
    if err != nil {
        os.Exit(-1)
    }
    if engine.Ping() != nil {
        os.Exit(-1)
    }
    cacher := xorm.NewLRUCacher(xorm.NewMemoryStore(), 5000)
    engine.SetDefaultCacher(cacher)

    engine.ShowSQL(true)
    engine.SetMaxIdleConns(10)
    engine.SetMaxOpenConns(30)

    timer := time.NewTicker(time.Minute * 30)
    go func(engine *xorm.Engine) {
        for _ = range timer.C {
            err = engine.Ping()
            if err != nil {
                log.Fatalf("db connect error: %#v\n", err.Error())
            }
        }
    }(engine)
    db = engine
}
```

示例

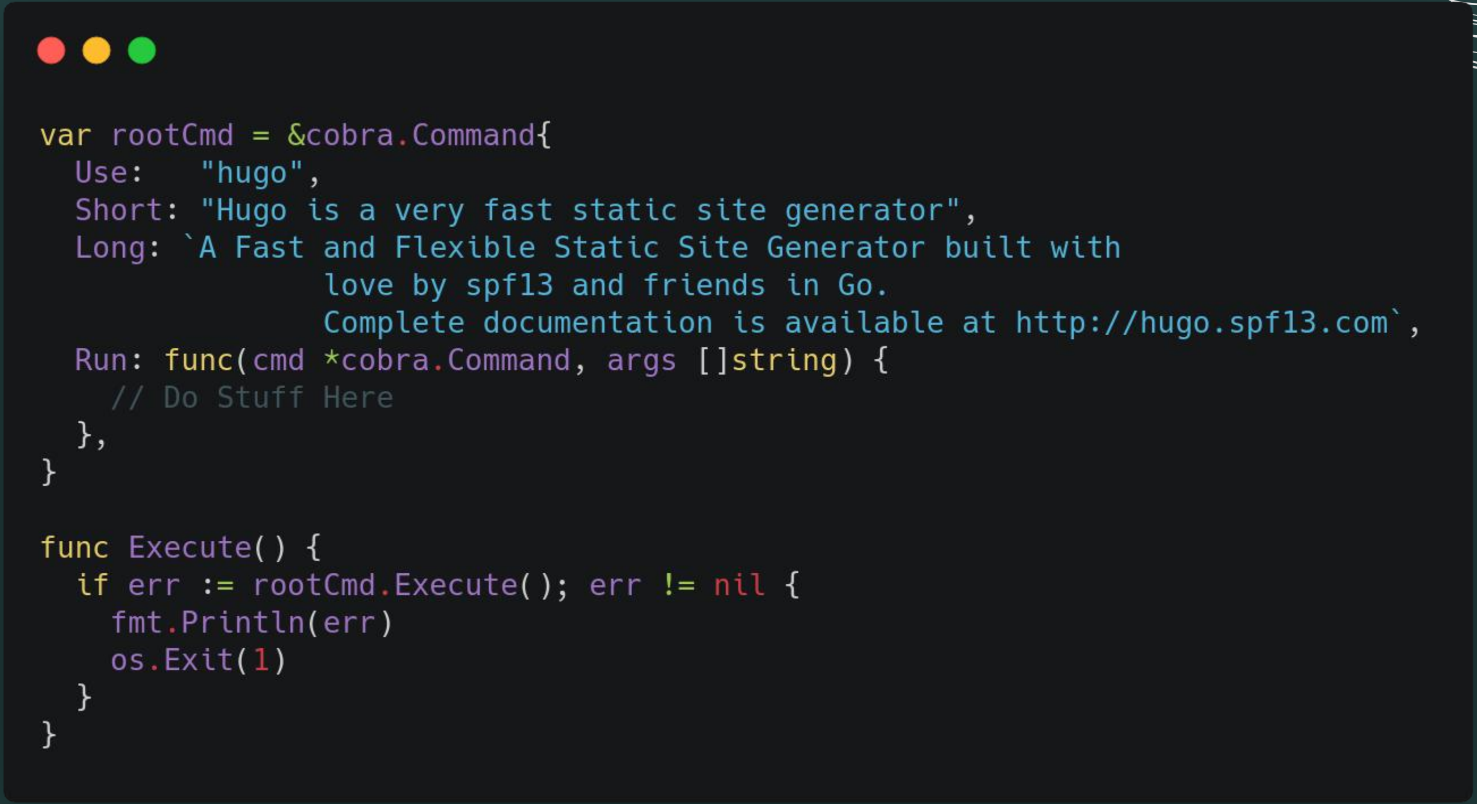


```
//insert
user := new(User)
user.Name = "myname"
affected, err := engine.Insert(user)

//update
user := new(User)
user.Name = "myname"
affected, err := engine.Id(id).Update(user)

//find
user := new(User)
has, err := engine.Where("name=?", "xlw").Get(user)

users := make([]Userinfo, 0)
err := engine.Where("age > ? or name = ?", 30, "xlw").Limit(20, 10).Find(&users)

//del
user := new(User)
affected, err := engine.Id(id).Delete(user)
```

目录页

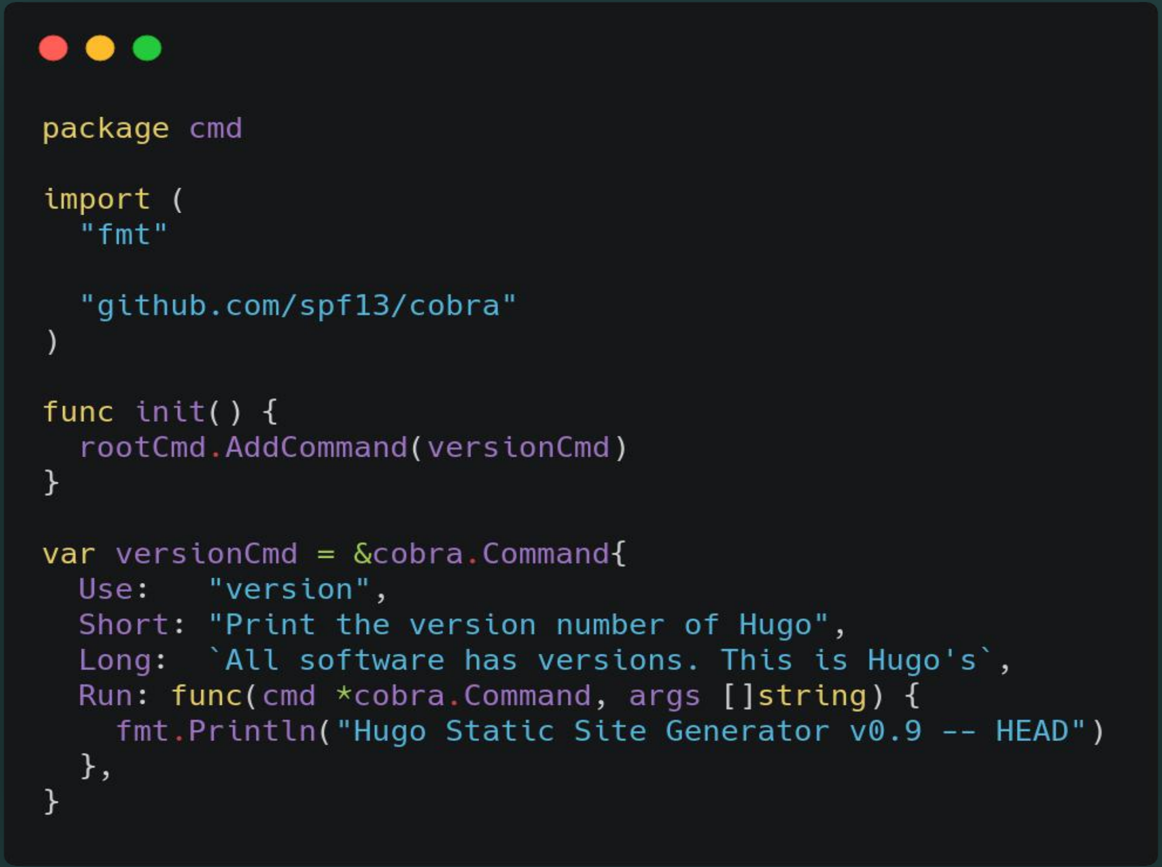
Cobra

配置



```go
var rootCmd = &cobra.Command{
  Use:   "hugo",
  Short: "Hugo is a very fast static site generator",
  Long: `A Fast and Flexible Static Site Generator built with
                love by spf13 and friends in Go.
                Complete documentation is available at http://hugo.spf13.com`,
  Run: func(cmd *cobra.Command, args []string) {
    // Do Stuff Here
  },
}

func Execute() {
  if err := rootCmd.Execute(); err != nil {
    fmt.Println(err)
    os.Exit(1)
  }
}
```

配置



```go
package cmd

import (
  "fmt"

  "github.com/spf13/cobra"
)

func init() {
  rootCmd.AddCommand(versionCmd)
}

var versionCmd = &cobra.Command{
  Use:   "version",
  Short: "Print the version number of Hugo",
  Long:  `All software has versions. This is Hugo's`,
  Run: func(cmd *cobra.Command, args []string) {
    fmt.Println("Hugo Static Site Generator v0.9 -- HEAD")
  },
}
```

Logrus

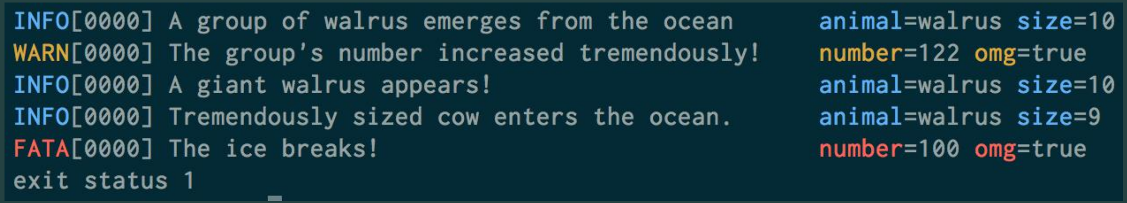
结构化的日志框架，logrus鼓励通过Field机制进行精细化的、结构化的日志记录，而不是通过冗长的消息来记录日志

配置

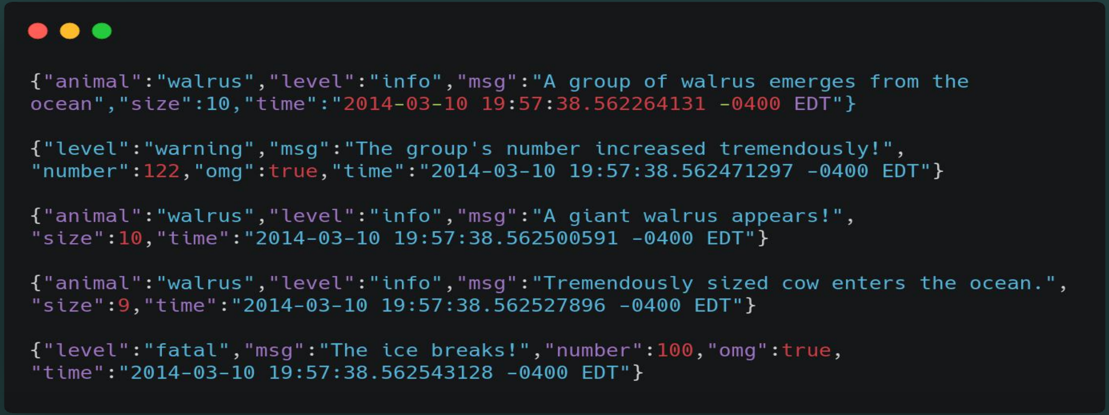
特性

一、Field机制，精细化控制日志
二、Hook机制，通过hook输出到多种源
三、完全兼容官方的日志格式

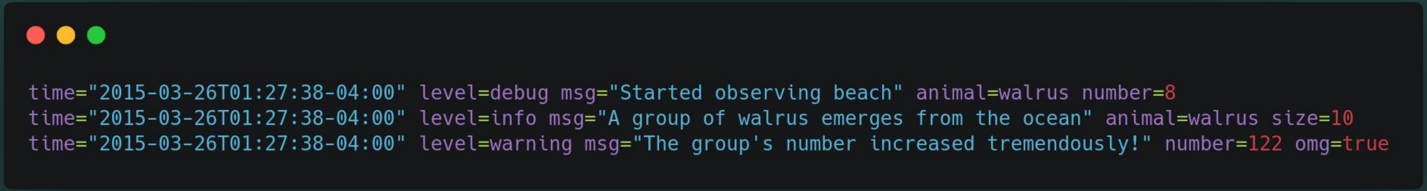
配置



```
INFO[0000] A group of walrus emerges from the ocean      animal=walrus size=10
WARN[0000] The group's number increased tremendously!    number=122 omg=true
INFO[0000] A giant walrus appears!                       animal=walrus size=10
INFO[0000] Tremendously sized cow enters the ocean.      animal=walrus size=9
FATA[0000] The ice breaks!                               number=100 omg=true
exit status 1
```

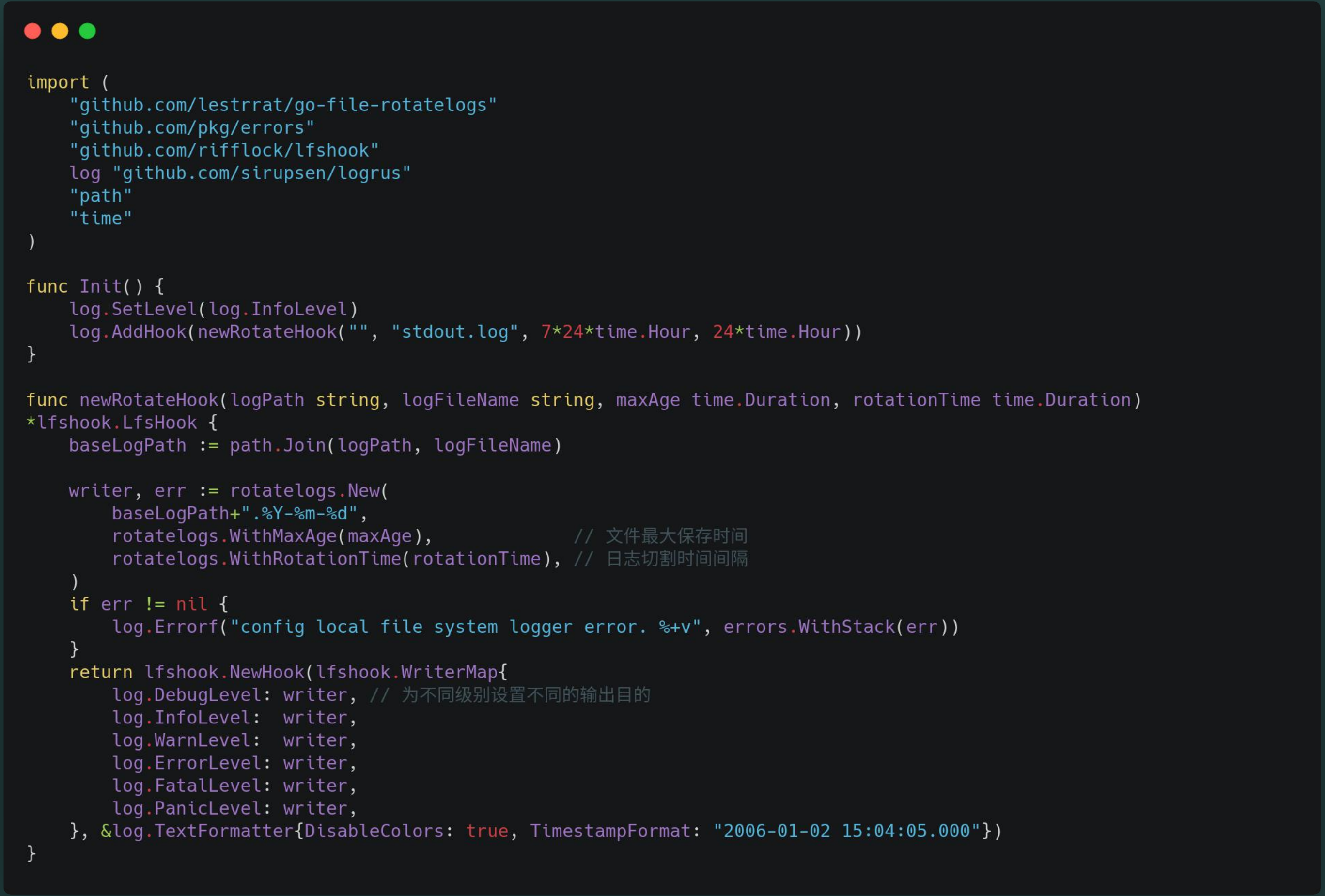


```
{"animal":"walrus","level":"info","msg":"A group of walrus emerges from the
ocean","size":10,"time":"2014-03-10 19:57:38.562264131 -0400 EDT"}

{"level":"warning","msg":"The group's number increased tremendously!",
"number":122,"omg":true,"time":"2014-03-10 19:57:38.562471297 -0400 EDT"}

{"animal":"walrus","level":"info","msg":"A giant walrus appears!",
"size":10,"time":"2014-03-10 19:57:38.562500591 -0400 EDT"}

{"animal":"walrus","level":"info","msg":"Tremendously sized cow enters the ocean.",
"size":9,"time":"2014-03-10 19:57:38.562527896 -0400 EDT"}

{"level":"fatal","msg":"The ice breaks!","number":100,"omg":true,
"time":"2014-03-10 19:57:38.562543128 -0400 EDT"}
```



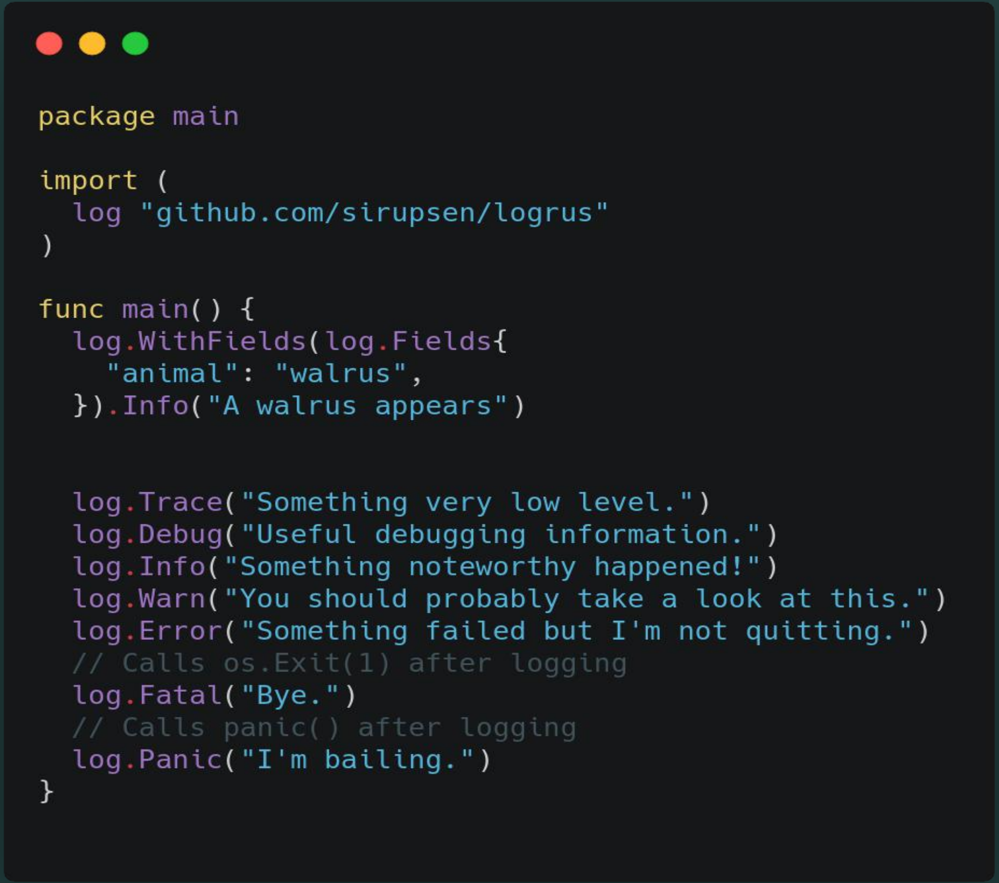
```
time="2015-03-26T01:27:38-04:00" level=debug msg="Started observing beach" animal=walrus number=8
time="2015-03-26T01:27:38-04:00" level=info msg="A group of walrus emerges from the ocean" animal=walrus size=10
time="2015-03-26T01:27:38-04:00" level=warning msg="The group's number increased tremendously!" number=122 omg=true
```

配置

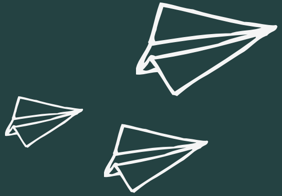


```go
import (
    "github.com/lestrrat/go-file-rotatelogs"
    "github.com/pkg/errors"
    "github.com/rifflock/lfshook"
    log "github.com/sirupsen/logrus"
    "path"
    "time"
)

func Init() {
    log.SetLevel(log.InfoLevel)
    log.AddHook(newRotateHook("", "stdout.log", 7*24*time.Hour, 24*time.Hour))
}

func newRotateHook(logPath string, logFileName string, maxAge time.Duration, rotationTime time.Duration)
*lfshook.LfsHook {
    baseLogPath := path.Join(logPath, logFileName)

    writer, err := rotatelogs.New(
        baseLogPath+".%Y-%m-%d",
        rotatelogs.WithMaxAge(maxAge),             // 文件最大保存时间
        rotatelogs.WithRotationTime(rotationTime), // 日志切割时间间隔
    )
    if err != nil {
        log.Errorf("config local file system logger error. %+v", errors.WithStack(err))
    }
    return lfshook.NewHook(lfshook.WriterMap{
        log.DebugLevel: writer, // 为不同级别设置不同的输出目的
        log.InfoLevel:  writer,
        log.WarnLevel:  writer,
        log.ErrorLevel: writer,
        log.FatalLevel: writer,
        log.PanicLevel: writer,
    }, &log.TextFormatter{DisableColors: true, TimestampFormat: "2006-01-02 15:04:05.000"})
}
```

使用



```go
package main

import (
  log "github.com/sirupsen/logrus"
)

func main() {
  log.WithFields(log.Fields{
    "animal": "walrus",
  }).Info("A walrus appears")


  log.Trace("Something very low level.")
  log.Debug("Useful debugging information.")
  log.Info("Something noteworthy happened!")
  log.Warn("You should probably take a look at this.")
  log.Error("Something failed but I'm not quitting.")
  // Calls os.Exit(1) after logging
  log.Fatal("Bye.")
  // Calls panic() after logging
  log.Panic("I'm bailing.")
}
```



感谢聆听