链滴

gin 框架自定义日志输出，自定义 gin 中间件扩展 Logger

作者：450370050

原文链接：https://ld246.com/article/1531819038419

来源网站：链滴

gin框架是款高性能的GoWeb框架，可以快速开发部署api服务。在使用过程中我们需要记录各种各的日志，下面介绍下我们怎么自定义日志记录格式或扩展日志。

gin简单剖析

api服务创建

```
package main

import "github.com/gin-gonic/gin"

func main() {
    r := gin.Default()
    r.GET("/ping", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "pong",
        })
    })
    r.Run() // listen and serve on 0.0.0.0:8080
}
```

以上是github中官方介绍的一个简单的demo，三步创建了一个api服务

1. gin.Default 获取到一个Engine实例
2. engine.GET() 添加一个Get请求的路由逻辑
3. engine.Run() 启动服务

gin.Default 剖析

```
func New() *Engine {
  debugPrintWARNINGNew()
  engine := &Engine{
    RouterGroup: RouterGroup{
      Handlers: nil,
      basePath: "/",
      root:     true,
    },
    FuncMap:                template.FuncMap{},
    RedirectTrailingSlash:  true,
    RedirectFixedPath:      false,
    HandleMethodNotAllowed: false,
    ForwardedByClientIP:    true,
    AppEngine:              defaultAppEngine,
    UseRawPath:             false,
    UnescapePathValues:     true,
    trees:                  make(methodTrees, 0, 9),
    delims:                 render.Delims{"{{", "}}"},
  }
  engine.RouterGroup.engine = engine
  engine.pool.New = func() interface{} {
    return engine.allocateContext()
```

```
    }
    return engine
}


// Default returns an Engine instance with the Logger and Recovery middleware already attac
ed.
func Default() *Engine {
    engine := New()
    engine.Use(Logger(), Recovery())
    return engine
}
```

gin.Default 通过New创建了Engine实例， 并Use了 Logger Recovery两个HandlerFunc中间件。释也介绍了

默认返回一个引擎实例，其中包含日志记录器和崩溃恢复中间件。

那么我们是不是可以通过自己的Logger中间件来记录日志？ 写到这里发现官方文档其实是有介绍的少走弯路还是先看文档额~~不过直接看源码也没坏处，哈哈

Using middleware

```go
func main() {
        // Creates a router without any middleware by default
        r := gin.New()

        // Global middleware
        // Logger middleware will write the logs to gin.DefaultWriter even if you set with GIN_MODE=release.
        // By default gin.DefaultWriter = os.Stdout
        r.Use(gin.Logger())

        // Recovery middleware recovers from any panics and writes a 500 if there was one.
        r.Use(gin.Recovery())

        // Per route middleware, you can add as many as you desire.
        r.GET("/benchmark", MyBenchLogger(), benchEndpoint)

        // Authorization group
        // authorized := r.Group("/", AuthRequired())
        // exactly the same as:
        authorized := r.Group("/")
        // per group middleware! in this case we use the custom created
        // AuthRequired() middleware just in the "authorized" group.
        authorized.Use(AuthRequired())
        {
                authorized.POST("/login", loginEndpoint)
                authorized.POST("/submit", submitEndpoint)
                authorized.POST("/read", readEndpoint)

                // nested group
                testing := authorized.Group("testing")
                testing.GET("/analytics", analyticsEndpoint)
        }

        // Listen and serve on 0.0.0.0:8080
        r.Run(":8080")
}
```

日志中间件

```
func (c *Context) Next() {
  c.index++
  s := int8(len(c.handlers))
  for ; c.index < s; c.index++ {
    c.handlers[c.index](c)
  }
}
```

gin通过循环当前的中间件处理链Handler，逐个调用中间件。

自定义日志中间件

1. 日志记录（logrus）

logrus是第三方包，github上比较活跃，已经实现了很多常用功能，日常开发中用到的较多。

2. 日志分割（rotatelogs）

logrus没有提供日志切分，go-file-rotatelogs可以实现linux logratate的大多数功能。

使用logrus的hook来加载 github.com/lestrrat/go-file-rotatelogs 模块.每次当我们写入日志的时，logrus都会调用go-file-rotatelogs 来判断日志是否要进行切分...

```
package api

import (
  "github.com/gin-gonic/gin"
  "os"
  "fmt"
  "github.com/sirupsen/logrus"
  "github.com/lestrrat/go-file-rotatelogs"
  "time"
  "github.com/rifflock/lfshook"
)

func Logger() gin.HandlerFunc {
  logClient := logrus.New()

  //禁止logrus的输出
  src, err := os.OpenFile(os.DevNull, os.O_APPEND|os.O_WRONLY, os.ModeAppend)
  if err!= nil{
    fmt.Println("err", err)
  }
  logClient.Out = src
  logClient.SetLevel(logrus.DebugLevel)
  apiLogPath := "api.log"
  logWriter, err := rotatelogs.New(
    apiLogPath+".%Y-%m-%d-%H-%M.log",
    rotatelogs.WithLinkName(apiLogPath), // 生成软链，指向最新日志文件
    rotatelogs.WithMaxAge(7*24*time.Hour), // 文件最大保存时间
    rotatelogs.WithRotationTime(24*time.Hour), // 日志切割时间间隔
  )
  writeMap := lfshook.WriterMap{
```

原文链接：gin 框架自定义日志输出，自定义 gin 中间件扩展 Logger

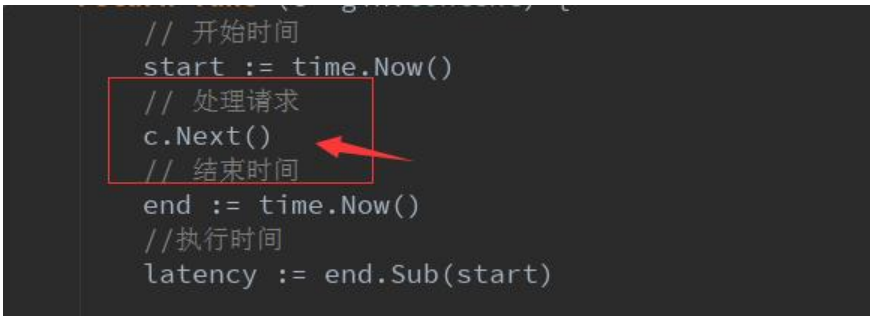
```
        logrus.InfoLevel:  logWriter,
        logrus.FatalLevel: logWriter,
    }
    lfHook := lfshook.NewHook(writeMap, &logrus.JSONFormatter{})
    logClient.AddHook(lfHook)


    return func (c *gin.Context) {
        // 开始时间
        start := time.Now()
        // 处理请求
        c.Next()
        // 结束时间
        end := time.Now()
        //执行时间
        latency := end.Sub(start)

        path := c.Request.URL.Path

        clientIP := c.ClientIP()
        method := c.Request.Method
        statusCode := c.Writer.Status()
        logClient.Infof("| %3d | %13v | %15s | %s  %s |",
            statusCode,
            latency,
            clientIP,
            method, path,
        )
    }
}
```

注:




```
// Next should be used only inside middleware.
// It executes the pending handlers in the chain inside the calling handler.
// See example in GitHub.
func (c *Context) Next() {
    c.index++
    s := int8(len(c.handlers))
    for ; c.index < s; c.index++ {
        c.handlers[c.index](c)
    }
}
```

这里面为了计算程序的执行时长， 我们又调用了一次c.Next()，让后面的中间件执行，这样日志中间拿到了程序其它所有中间件执行的总时长，认为是程序的处理时间。

c.Next循环的时候都是同一个Context指针上下文，index下标为共享的数据，我们可以通过调用一次cNext让log中间件插一脚。