

A decorative border with intricate floral and scrollwork patterns in a dark brown color, framing the central text.

Golang Gin框架 英文文档

书栈(BookStack.CN)

目 录

致谢

Contents (内容)

Installation

Prerequisite

Quick start

Benchmarks

Gin v1.stable

Build with jsoniter

API Examples

Using GET,POST,PUT,PATCH,DELETE and OPTIONS

Parameters in path

Querystring parameters

Multipart/Urlencoded Form

Another example: query + post form

Upload files

Grouping routes

Blank Gin without middleware by default

Using middleware

How to write log file

Model binding and validation

Custom Validators

Only Bind Query String

Bind Query String or Post Data

Bind HTML checkboxes

Multipart/Urlencoded binding

XML, JSON and YAML rendering

Serving static files

Serving data from reader

HTML rendering

Multitemplate

Redirects

Custom Middleware

Using BasicAuth() middleware

Goroutines inside a middleware

Custom HTTP configuration

Support Let's Encrypt

[Run multiple service using Gin](#)
[Graceful restart or stop](#)
[Build a single binary with templates](#)
[Bind form-data request with custom struct](#)
[Try to bind body into different structs](#)
[http2 server push](#)
[Testing](#)
[Users](#)
[AUTHORS](#)
[Benchmark System](#)
[CHANGELOG](#)
[Contributor Covenant Code of Conduct](#)
[Contributing](#)
[Guide to run Gin under App Engine LOCAL Development Server](#)
[How to use](#)
[How to run this example](#)
[How to generate RSA private key and digital certificate](#)
[Struct level validations](#)
[Gin Default Server](#)

致谢

当前文档《Golang Gin框架英文文档》由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-07-04。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN)，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：<http://www.bookstack.cn/books/gin-en>

书栈官网：<http://www.bookstack.cn>

书栈开源：<https://github.com/TruthHun>

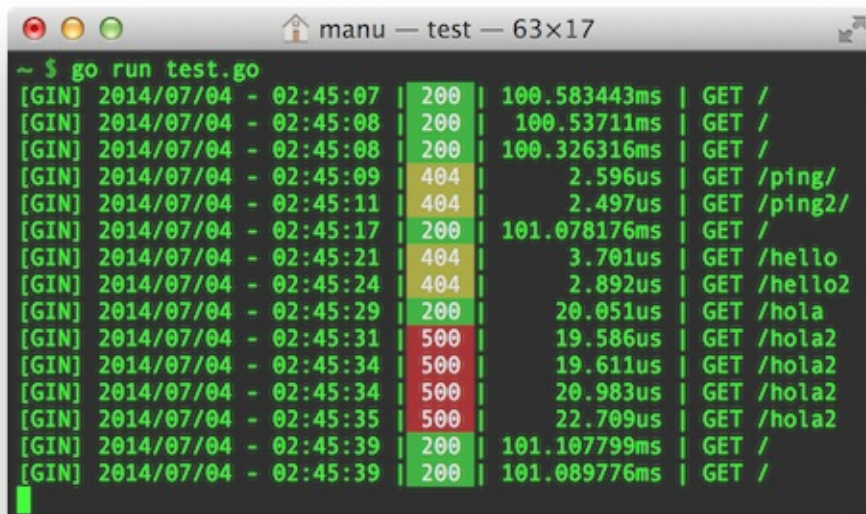
分享，让知识传承更久远！感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

Contents (内容)

Gin Web Framework



Gin is a web framework written in Go (Golang). It features a martini-like API with much better performance, up to 40 times faster thanks to [httprouter](#). If you need performance and good productivity, you will love Gin.



The screenshot shows a terminal window titled "manu — test — 63x17". The command executed is `~ $ go run test.go`. The output displays a series of log entries for the Gin framework, each showing a timestamp, a status code, a response time, and the HTTP method and path. The status codes are color-coded: green for 200, yellow for 404, and red for 500. The response times are in milliseconds (ms) or microseconds (us).

Timestamp	Status	Response Time	Method	Path
2014/07/04 - 02:45:07	200	100.583443ms	GET	/
2014/07/04 - 02:45:08	200	100.53711ms	GET	/
2014/07/04 - 02:45:08	200	100.326316ms	GET	/
2014/07/04 - 02:45:09	404	2.596us	GET	/ping/
2014/07/04 - 02:45:11	404	2.497us	GET	/ping2/
2014/07/04 - 02:45:17	200	101.078176ms	GET	/
2014/07/04 - 02:45:21	404	3.701us	GET	/hello
2014/07/04 - 02:45:24	404	2.892us	GET	/hello2
2014/07/04 - 02:45:29	200	20.051us	GET	/hola
2014/07/04 - 02:45:31	500	19.586us	GET	/hola2
2014/07/04 - 02:45:34	500	19.611us	GET	/hola2
2014/07/04 - 02:45:34	500	20.983us	GET	/hola2
2014/07/04 - 02:45:35	500	22.709us	GET	/hola2
2014/07/04 - 02:45:39	200	101.107799ms	GET	/
2014/07/04 - 02:45:39	200	101.089776ms	GET	/

API Examples

- [Using GET, POST, PUT, PATCH, DELETE and OPTIONS](#)
- [Parameters in path](#)
- [Querystring parameters](#)
- [Multipart/Urlencoded Form](#)
- [Another example: query + post form](#)
- [Upload files](#)
- [Grouping routes](#)
- [Blank Gin without middleware by default](#)
- [Using middleware](#)
- [How to write log file](#)
- [Model binding and validation](#)
- [Custom Validators](#)
- [Only Bind Query String](#)
- [Bind Query String or Post Data](#)
- [Bind HTML checkboxes](#)

- [Multipart/Urlencoded binding](#)
- [XML, JSON and YAML rendering](#)
- [Serving static files](#)
- [Serving data from reader](#)
- [HTML rendering](#)
- [Multitemplate](#)
- [Redirects](#)
- [Custom Middleware](#)
- [Using BasicAuth\(\) middleware](#)
- [Goroutines inside a middleware](#)
- [Custom HTTP configuration](#)
- [Support Let's Encrypt](#)
- [Run multiple service using Gin](#)
- [Graceful restart or stop](#)
- [Build a single binary with templates](#)
- [Bind form-data request with custom struct](#)
- [Try to bind body into different structs](#)
- [http2 server push](#)

Installation

Installation

To install Gin package, you need to install Go and set your Go workspace first.

1. Download and install it:

```
1. $ go get -u github.com/gin-gonic/gin
```

1. Import it in your code:

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

1. (Optional) Import `net/http`. This is required for example if using constants such as `http.StatusOK`.

```
1. import "net/http"
```

Use a vendor tool like [Govendor](#)

1. `go get` govendor

```
1. $ go get github.com/kardianos/govendor
```

1. Create your project folder and `cd` inside

```
1. $ mkdir -p $GOPATH/src/github.com/myusername/project && cd "$_"
```

1. Vendor init your project and add gin

```
1. $ govendor init
2. $ govendor fetch github.com/gin-gonic/gin@v1.2
```

1. Copy a starting template inside your project

```
1. $ curl https://raw.githubusercontent.com/gin-gonic/gin/master/examples/basic/main.go > main.go
```

1. Run your project

```
1. $ go run main.go
```

Prerequisite

Prerequisite

Now Gin requires Go 1.6 or later and Go 1.7 will be required soon.

Quick start

Quick start

1. # assume the following codes in example.go file
2. \$ cat example.go

```
1. package main
2.
3. import "github.com/gin-gonic/gin"
4.
5. func main() {
6.     r := gin.Default()
7.     r.GET("/ping", func(c *gin.Context) {
8.         c.JSON(200, gin.H{
9.             "message": "pong",
10.        })
11.    })
12.    r.Run() // listen and serve on 0.0.0.0:8080
13. }
```

1. # run example.go and visit 0.0.0.0:8080/ping on browser
2. \$ go run example.go

Benchmarks

Benchmarks

Gin uses a custom version of [HttpRouter](#)

[See all benchmarks](#)

Benchmark name	(1)	(2)	(3)	(4)
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

Gin v1.stable

Gin v1. stable

- ☑ Zero allocation router.
- ☑ Still the fastest http router and framework. From routing to writing.
- ☑ Complete suite of unit tests
- ☑ Battle tested
- ☑ API frozen, new releases will not break your code.

Build with jsoniter

Build with jsoniter

Gin use `encoding/json` as default json package but you can change to `jsoniter` by build from other tags.

```
1. $ go build -tags=jsoniter .
```


API Examples

- Using GET, POST, PUT, PATCH, DELETE and OPTIONS
- Parameters in path
- Querystring parameters
- Multipart/Urlencoded Form
- Another example: query + post form
- Upload files
- Grouping routes
- Blank Gin without middleware by default
- Using middleware
- How to write log file
- Model binding and validation
- Custom Validators
- Only Bind Query String
- Bind Query String or Post Data
- Bind HTML checkboxes
- Multipart/Urlencoded binding
- XML, JSON and YAML rendering
- Serving static files
- Serving data from reader
- HTML rendering
- Multitemplate
- Redirects
- Custom Middleware
- Using BasicAuth() middleware
- Goroutines inside a middleware
- Custom HTTP configuration
- Support Let's Encrypt
- Run multiple service using Gin
- Graceful restart or stop
- Build a single binary with templates
- Bind form-data request with custom struct
- Try to bind body into different structs
- http2 server push

Using GET,POST,PUT,PATCH,DELETE and OPTIONS

Using GET, POST, PUT, PATCH, DELETE and OPTIONS

```
1. func main() {
2.     // Disable Console Color
3.     // gin.DisableConsoleColor()
4.
5.     // Creates a gin router with default middleware:
6.     // logger and recovery (crash-free) middleware
7.     router := gin.Default()
8.
9.     router.GET("/someGet", getting)
10.    router.POST("/somePost", posting)
11.    router.PUT("/somePut", putting)
12.    router.DELETE("/someDelete", deleting)
13.    router.PATCH("/somePatch", patching)
14.    router.HEAD("/someHead", head)
15.    router.OPTIONS("/someOptions", options)
16.
17.    // By default it serves on :8080 unless a
18.    // PORT environment variable was defined.
19.    router.Run()
20.    // router.Run(":3000") for a hard coded port
21. }
```

Parameters in path

Parameters in path

```
1. func main() {
2.     router := gin.Default()
3.
4.     // This handler will match /user/john but will not match neither /user/ or
   //user
5.     router.GET("/user/:name", func(c *gin.Context) {
6.         name := c.Param("name")
7.         c.String(http.StatusOK, "Hello %s", name)
8.     })
9.
10.    // However, this one will match /user/john/ and also /user/john/send
11.    // If no other routers match /user/john, it will redirect to /user/john/
12.    router.GET("/user/:name/*action", func(c *gin.Context) {
13.        name := c.Param("name")
14.        action := c.Param("action")
15.        message := name + " is " + action
16.        c.String(http.StatusOK, message)
17.    })
18.
19.    router.Run(":8080")
20. }
```

Querystring parameters

Querystring parameters

```
1. func main() {
2.     router := gin.Default()
3.
4.     // Query string parameters are parsed using the existing underlying request
   object.
5.     // The request responds to a url matching: /welcome?
   firstname=Jane&lastname=Doe
6.     router.GET("/welcome", func(c *gin.Context) {
7.         firstname := c.DefaultQuery("firstname", "Guest")
8.         lastname := c.Query("lastname") // shortcut for
   c.Request.URL.Query().Get("lastname")
9.
10.        c.String(http.StatusOK, "Hello %s %s", firstname, lastname)
11.    })
12.    router.Run(":8080")
13. }
```

Multipart/Urlencoded Form

Multipart/Urlencoded Form

```
1. func main() {
2.     router := gin.Default()
3.
4.     router.POST("/form_post", func(c *gin.Context) {
5.         message := c.PostForm("message")
6.         nick := c.DefaultPostForm("nick", "anonymous")
7.
8.         c.JSON(200, gin.H{
9.             "status": "posted",
10.            "message": message,
11.            "nick":    nick,
12.        })
13.    })
14.    router.Run(":8080")
15. }
```

Another example: query + post form

Another example: query + post form

1. POST /post?id=1234&page=1 HTTP/1.1
2. Content-Type: application/x-www-form-urlencoded
- 3.
4. name=manu&message=this_is_great

```
1. func main() {
2.     router := gin.Default()
3.
4.     router.POST("/post", func(c *gin.Context) {
5.
6.         id := c.Query("id")
7.         page := c.DefaultQuery("page", "0")
8.         name := c.PostForm("name")
9.         message := c.PostForm("message")
10.
11.         fmt.Printf("id: %s; page: %s; name: %s; message: %s", id, page, name,
12.             message)
13.     })
14.     router.Run(":8080")
15. }
```

1. id: 1234; page: 1; name: manu; message: this_is_great

Upload files

Upload files

Single file

References [issue #774](#) and detail [example code](#).

```
1. func main() {
2.     router := gin.Default()
3.     // Set a lower memory limit for multipart forms (default is 32 MiB)
4.     // router.MaxMultipartMemory = 8 << 20 // 8 MiB
5.     router.POST("/upload", func(c *gin.Context) {
6.         // single file
7.         file, _ := c.FormFile("file")
8.         log.Println(file.Filename)
9.
10.        // Upload the file to specific dst.
11.        // c.SaveUploadedFile(file, dst)
12.
13.        c.String(http.StatusOK, fmt.Sprintf("'%s' uploaded!", file.Filename))
14.    })
15.    router.Run(":8080")
16. }
```

How to `curl` :

```
1. curl -X POST http://localhost:8080/upload \
2.     -F "file=@/Users/appleboy/test.zip" \
3.     -H "Content-Type: multipart/form-data"
```

Multiple files

See the detail [example code](#).

```
1. func main() {
2.     router := gin.Default()
3.     // Set a lower memory limit for multipart forms (default is 32 MiB)
4.     // router.MaxMultipartMemory = 8 << 20 // 8 MiB
5.     router.POST("/upload", func(c *gin.Context) {
```

```
6.      // Multipart form
7.      form, _ := c.MultipartForm()
8.      files := form.File["upload[]"]
9.
10.     for _, file := range files {
11.         log.Println(file.Filename)
12.
13.         // Upload the file to specific dst.
14.         // c.SaveUploadedFile(file, dst)
15.     }
16.     c.String(http.StatusOK, fmt.Sprintf("%d files uploaded!", len(files)))
17. })
18. router.Run(":8080")
19. }
```

How to `curl` :

1. `curl -X POST http://localhost:8080/upload \`
2. `-F "upload[]=@/Users/appleboy/test1.zip" \`
3. `-F "upload[]=@/Users/appleboy/test2.zip" \`
4. `-H "Content-Type: multipart/form-data"`

Grouping routes

Grouping routes

```
1. func main() {
2.     router := gin.Default()
3.
4.     // Simple group: v1
5.     v1 := router.Group("/v1")
6.     {
7.         v1.POST("/login", loginEndpoint)
8.         v1.POST("/submit", submitEndpoint)
9.         v1.POST("/read", readEndpoint)
10.    }
11.
12.    // Simple group: v2
13.    v2 := router.Group("/v2")
14.    {
15.        v2.POST("/login", loginEndpoint)
16.        v2.POST("/submit", submitEndpoint)
17.        v2.POST("/read", readEndpoint)
18.    }
19.
20.    router.Run(":8080")
21. }
```

Blank Gin without middleware by default

Blank Gin without middleware by default

Use

```
1. r := gin.New()
```

instead of

```
1. // Default With the Logger and Recovery middleware already attached
2. r := gin.Default()
```

Using middleware

Using middleware

```
1. func main() {
2.     // Creates a router without any middleware by default
3.     r := gin.New()
4.
5.     // Global middleware
6.     // Logger middleware will write the logs to gin.DefaultWriter even if you
   set with GIN_MODE=release.
7.     // By default gin.DefaultWriter = os.Stdout
8.     r.Use(gin.Logger())
9.
10.    // Recovery middleware recovers from any panics and writes a 500 if there
   was one.
11.    r.Use(gin.Recovery())
12.
13.    // Per route middleware, you can add as many as you desire.
14.    r.GET("/benchmark", MyBenchLogger(), benchEndpoint)
15.
16.    // Authorization group
17.    // authorized := r.Group("/", AuthRequired())
18.    // exactly the same as:
19.    authorized := r.Group("/")
20.    // per group middleware! in this case we use the custom created
21.    // AuthRequired() middleware just in the "authorized" group.
22.    authorized.Use(AuthRequired())
23.    {
24.        authorized.POST("/login", loginEndpoint)
25.        authorized.POST("/submit", submitEndpoint)
26.        authorized.POST("/read", readEndpoint)
27.
28.        // nested group
29.        testing := authorized.Group("testing")
30.        testing.GET("/analytics", analyticsEndpoint)
31.    }
32.
33.    // Listen and serve on 0.0.0.0:8080
34.    r.Run(":8080")
}
```

```
35. }
```

How to write log file

How to write log file

```
1. func main() {
2.     // Disable Console Color, you don't need console color when writing the
   logs to file.
3.     gin.DisableConsoleColor()
4.
5.     // Logging to a file.
6.     f, _ := os.Create("gin.log")
7.     gin.DefaultWriter = io.MultiWriter(f)
8.
9.     // Use the following code if you need to write the logs to file and console
   at the same time.
10.    // gin.DefaultWriter = io.MultiWriter(f, os.Stdout)
11.
12.    router := gin.Default()
13.    router.GET("/ping", func(c *gin.Context) {
14.        c.String(200, "pong")
15.    })
16.
17.    router.Run(":8080")
18. }
```

Model binding and validation

Model binding and validation

To bind a request body into a type, use model binding. We currently support binding of JSON, XML and standard form values (foo=bar&boo=baz).

Gin uses [go-playground/validator.v8](#) for validation. Check the full docs on tags usage [here](#).

Note that you need to set the corresponding binding tag on all fields you want to bind. For example, when binding from JSON, set `json:"fieldname"`.

Also, Gin provides two sets of methods for binding:

- **Type** - Must bind
 - **Methods** - `Bind`, `BindJSON`, `BindQuery`
 - **Behavior** - These methods use `MustBindWith` under the hood. If there is a binding error, the request is aborted with `c.AbortWithError(400, err).SetType(ErrorTypeBind)`. This sets the response status code to 400 and the `Content-Type` header is set to `text/plain; charset=utf-8`. Note that if you try to set the response code after this, it will result in a warning `[GIN-debug] [WARNING] Headers were already written. Wanted to override status code 400 with 422`. If you wish to have greater control over the behavior, consider using the `ShouldBind` equivalent method.
- **Type** - Should bind
 - **Methods** - `ShouldBind`, `ShouldBindJSON`, `ShouldBindQuery`
 - **Behavior** - These methods use `ShouldBindWith` under the hood. If there is a binding error, the error is returned and it is the developer's responsibility to handle the request and error appropriately.

When using the Bind-method, Gin tries to infer the binder depending on the Content-Type header. If you are sure what you are binding, you can use `MustBindWith` or `ShouldBindWith`.

You can also specify that specific fields are required. If a field is decorated with `binding:"required"` and has a empty value when binding, an error will be returned.

```
1. // Binding from JSON
2. type Login struct {
3.     User      string `form:"user" json:"user" binding:"required"`
4.     Password string `form:"password" json:"password" binding:"required"`
5. }
6.
7. func main() {
8.     router := gin.Default()
9.
10.    // Example for binding JSON ({"user": "manu", "password": "123"})
11.    router.POST("/loginJSON", func(c *gin.Context) {
12.        var json Login
13.        if err := c.ShouldBindJSON(&json); err == nil {
14.            if json.User == "manu" && json.Password == "123" {
15.                c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
16.            } else {
17.                c.JSON(http.StatusUnauthorized, gin.H{"status":
"unauthorized"})
18.            }
19.        } else {
20.            c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
21.        }
22.    })
23.
24.    // Example for binding a HTML form (user=manu&password=123)
25.    router.POST("/loginForm", func(c *gin.Context) {
26.        var form Login
27.        // This will infer what binder to use depending on the content-type
header.
28.        if err := c.ShouldBind(&form); err == nil {
29.            if form.User == "manu" && form.Password == "123" {
30.                c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
31.            } else {
32.                c.JSON(http.StatusUnauthorized, gin.H{"status":
"unauthorized"})
33.            }
34.        } else {
35.            c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
36.        }
37.    })
38.
39.    // Listen and serve on 0.0.0.0:8080
```

```
40.     router.Run(":8080")
41. }
```

Sample request

```
1. $ curl -v -X POST \
2.   http://localhost:8080/loginJSON \
3.   -H 'content-type: application/json' \
4.   -d '{ "user": "manu" }'
5. > POST /loginJSON HTTP/1.1
6. > Host: localhost:8080
7. > User-Agent: curl/7.51.0
8. > Accept: */*
9. > content-type: application/json
10. > Content-Length: 18
11. >
12. * upload completely sent off: 18 out of 18 bytes
13. < HTTP/1.1 400 Bad Request
14. < Content-Type: application/json; charset=utf-8
15. < Date: Fri, 04 Aug 2017 03:51:31 GMT
16. < Content-Length: 100
17. <
18. {"error":"Key: 'Login.Password' Error:Field validation for 'Password' failed on
    the 'required' tag"}
```

Skip validate

When running the above example using the above the `curl` command, it returns error. Because the example use `binding:"required"` for `Password`. If use `binding:"-"` for `Password`, then it will not return error when running the above example again.

Custom Validators

Custom Validators

It is also possible to register custom validators. See the [example code](#).

```
1. package main
2.
3. import (
4.     "net/http"
5.     "reflect"
6.     "time"
7.
8.     "github.com/gin-gonic/gin"
9.     "github.com/gin-gonic/gin/binding"
10.    "gopkg.in/go-playground/validator.v8"
11. )
12.
13. type Booking struct {
14.     CheckIn  time.Time `form:"check_in" binding:"required,bookabledate"
15.     CheckOut time.Time `form:"check_out" binding:"required,gtfield=CheckIn"
16. }
17.
18. func bookableDate(
19.     v *validator.Validate, topStruct reflect.Value, currentStructOrField
20.     reflect.Value,
21.     field reflect.Value, fieldType reflect.Type, fieldKind reflect.Kind, param
22.     string,
23. ) bool {
24.     if date, ok := field.Interface().(time.Time); ok {
25.         today := time.Now()
26.         if today.Year() > date.Year() || today.YearDay() > date.YearDay() {
27.             return false
28.         }
29.     }
30.     return true
31. }
32.
33. func main() {
```

```

32.     route := gin.Default()
33.
34.     if v, ok := binding.Validator.Engine().(*validator.Validate); ok {
35.         v.RegisterValidation("bookabledate", bookableDate)
36.     }
37.
38.     route.GET("/bookable", getBookable)
39.     route.Run(":8085")
40. }
41.
42. func getBookable(c *gin.Context) {
43.     var b Booking
44.     if err := c.ShouldBindWith(&b, binding.Query); err == nil {
45.         c.JSON(http.StatusOK, gin.H{"message": "Booking dates are valid!"})
46.     } else {
47.         c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
48.     }
49. }

```

```

1. $ curl "localhost:8085/bookable?check_in=2018-04-16&check_out=2018-04-17"
2. {"message":"Booking dates are valid!"}
3.
4. $ curl "localhost:8085/bookable?check_in=2018-03-08&check_out=2018-03-09"
5. {"error":"Key: 'Booking.CheckIn' Error:Field validation for 'CheckIn' failed on
   the 'bookabledate' tag"}

```

Struct level validations can also be registered this way.

See the [struct-lvl-validation example](#) to learn more.

Only Bind Query String

Only Bind Query String

`ShouldBindQuery` function only binds the query params and not the post data. See the [detail information](#).

```
1. package main
2.
3. import (
4.     "log"
5.
6.     "github.com/gin-gonic/gin"
7. )
8.
9. type Person struct {
10.     Name     string `form:"name"`
11.     Address  string `form:"address"`
12. }
13.
14. func main() {
15.     route := gin.Default()
16.     route.Any("/testing", startPage)
17.     route.Run(":8085")
18. }
19.
20. func startPage(c *gin.Context) {
21.     var person Person
22.     if c.ShouldBindQuery(&person) == nil {
23.         log.Println("==== Only Bind By Query String =====")
24.         log.Println(person.Name)
25.         log.Println(person.Address)
26.     }
27.     c.String(200, "Success")
28. }
```

Bind Query String or Post Data

Bind Query String or Post Data

See the [detail information](#).

```
1. package main
2.
3. import "log"
4. import "github.com/gin-gonic/gin"
5. import "time"
6.
7. type Person struct {
8.     Name      string    `form:"name"`
9.     Address   string    `form:"address"`
10.    Birthday  time.Time `form:"birthday" time_format:"2006-01-02" time_utc:"1"`
11. }
12.
13. func main() {
14.     route := gin.Default()
15.     route.GET("/testing", startPage)
16.     route.Run(":8085")
17. }
18.
19. func startPage(c *gin.Context) {
20.     var person Person
21.     // If `GET`, only `Form` binding engine (`query`) used.
22.     // If `POST`, first checks the `content-type` for `JSON` or `XML`, then
    uses `Form` (`form-data`).
23.     // See more at https://github.com/gin-
    gonic/gin/blob/master/binding/binding.go#L48
24.     if c.ShouldBind(&person) == nil {
25.         log.Println(person.Name)
26.         log.Println(person.Address)
27.         log.Println(person.Birthday)
28.     }
29.
30.     c.String(200, "Success")
31. }
```

Test it with:

```
1. $ curl -X GET "localhost:8085/testing?name=appleboy&address=xyz&birthday=1992-03-15"
```

Bind HTML checkboxes

Bind HTML checkboxes

See the [detail information](#)

main.go

```

1.  ...
2.
3.  type myForm struct {
4.      Colors []string `form:"colors[]"`
5.  }
6.
7.  ...
8.
9.  func formHandler(c *gin.Context) {
10.     var fakeForm myForm
11.     c.ShouldBind(&fakeForm)
12.     c.JSON(200, gin.H{"color": fakeForm.Colors})
13.  }
14.
15.  ...

```

form.html

```

1.  <form action="/" method="POST">
2.      <p>Check some colors</p>
3.      <label for="red">Red</label>
4.      <input type="checkbox" name="colors[]" value="red" id="red" />
5.      <label for="green">Green</label>
6.      <input type="checkbox" name="colors[]" value="green" id="green" />
7.      <label for="blue">Blue</label>
8.      <input type="checkbox" name="colors[]" value="blue" id="blue" />
9.      <input type="submit" />
10. </form>

```

result:

```
1.  {"color":["red", "green", "blue"]}
```

Multipart/Urlencoded binding

Multipart/Urlencoded binding

```
1. package main
2.
3. import (
4.     "github.com/gin-gonic/gin"
5. )
6.
7. type LoginForm struct {
8.     User      string `form:"user" binding:"required"`
9.     Password string `form:"password" binding:"required"`
10. }
11.
12. func main() {
13.     router := gin.Default()
14.     router.POST("/login", func(c *gin.Context) {
15.         // you can bind multipart form with explicit binding declaration:
16.         // c.ShouldBindWith(&form, binding.Form)
17.         // or you can simply use autobinding with ShouldBind method:
18.         var form LoginForm
19.         // in this case proper binding will be automatically selected
20.         if c.ShouldBind(&form) == nil {
21.             if form.User == "user" && form.Password == "password" {
22.                 c.JSON(200, gin.H{"status": "you are logged in"})
23.             } else {
24.                 c.JSON(401, gin.H{"status": "unauthorized"})
25.             }
26.         }
27.     })
28.     router.Run(":8080")
29. }
```

Test it with:

```
1. $ curl -v --form user=user --form password=password http://localhost:8080/login
```

XML, JSON and YAML rendering

XML, JSON and YAML rendering

```

1. func main() {
2.     r := gin.Default()
3.
4.     // gin.H is a shortcut for map[string]interface{}
5.     r.GET("/someJSON", func(c *gin.Context) {
6.         c.JSON(http.StatusOK, gin.H{"message": "hey", "status": http.StatusOK})
7.     })
8.
9.     r.GET("/moreJSON", func(c *gin.Context) {
10.        // You also can use a struct
11.        var msg struct {
12.            Name    string `json:"user"`
13.            Message string
14.            Number  int
15.        }
16.        msg.Name = "Lena"
17.        msg.Message = "hey"
18.        msg.Number = 123
19.        // Note that msg.Name becomes "user" in the JSON
20.        // Will output : {"user": "Lena", "Message": "hey", "Number": 123}
21.        c.JSON(http.StatusOK, msg)
22.    })
23.
24.    r.GET("/someXML", func(c *gin.Context) {
25.        c.XML(http.StatusOK, gin.H{"message": "hey", "status": http.StatusOK})
26.    })
27.
28.    r.GET("/someYAML", func(c *gin.Context) {
29.        c.YAML(http.StatusOK, gin.H{"message": "hey", "status": http.StatusOK})
30.    })
31.
32.    // Listen and serve on 0.0.0.0:8080
33.    r.Run(":8080")
34. }

```

SecureJSON

Using SecureJSON to prevent json hijacking. Default prepends to response body if the given struct is array values.

```
"while(1),"
```

```
1. func main() {
2.     r := gin.Default()
3.
4.     // You can also use your own secure json prefix
5.     // r.SecureJsonPrefix("]]'","\n")
6.
7.     r.GET("/someJSON", func(c *gin.Context) {
8.         names := []string{"lena", "austin", "foo"}
9.
10.        // Will output :   while(1);["lena","austin","foo"]
11.        c.SecureJSON(http.StatusOK, names)
12.    })
13.
14.    // Listen and serve on 0.0.0.0:8080
15.    r.Run(":8080")
16. }
```

JSONP

Using JSONP to request data from a server in a different domain. Add callback to response body if the query parameter callback exists.

```
1. func main() {
2.     r := gin.Default()
3.
4.     r.GET("/JSONP?callback=x", func(c *gin.Context) {
5.         data := map[string]interface{}{
6.             "foo": "bar",
7.         }
8.
9.         //callback is x
10.        // Will output :   x({"foo":"bar"})
11.        c.JSONP(http.StatusOK, data)
12.    })
13.
14.    // Listen and serve on 0.0.0.0:8080
15.    r.Run(":8080")
16. }
```

AsciiJSON

Using AsciiJSON to Generates ASCII-only JSON with escaped non-ASCII chracters.

```
1. func main() {
2.     r := gin.Default()
3.
4.     r.GET("/someJSON", func(c *gin.Context) {
5.         data := map[string]interface{}{
6.             "lang": "GO语言",
7.             "tag": "<br>",
8.         }
9.
10.        // will output : {"lang":"GO\u8bed\u8a00","tag":"\u003cbr\u003e"}
11.        c.AscliJSON(http.StatusOK, data)
12.    })
13.
14.    // Listen and serve on 0.0.0.0:8080
15.    r.Run(":8080")
16. }
```

Serving static files

Serving static files

```
1. func main() {  
2.     router := gin.Default()  
3.     router.Static("/assets", "./assets")  
4.     router.StaticFS("/more_static", http.Dir("my_file_system"))  
5.     router.StaticFile("/favicon.ico", "./resources/favicon.ico")  
6.  
7.     // Listen and serve on 0.0.0.0:8080  
8.     router.Run(":8080")  
9. }
```

Serving data from reader

Serving data from reader

```
1. func main() {
2.     router := gin.Default()
3.     router.GET("/someDataFromReader", func(c *gin.Context) {
4.         response, err := http.Get("https://raw.githubusercontent.com/gin-
gonic/logo/master/color.png")
5.         if err != nil || response.StatusCode != http.StatusOK {
6.             c.Status(http.StatusServiceUnavailable)
7.             return
8.         }
9.
10.        reader := response.Body
11.        contentLength := response.ContentLength
12.        contentType := response.Header.Get("Content-Type")
13.
14.        extraHeaders := map[string]string{
15.            "Content-Disposition": `attachment; filename="gopher.png"`,
16.        }
17.
18.        c.DataFromReader(http.StatusOK, contentLength, contentType, reader,
extraHeaders)
19.    })
20.    router.Run(":8080")
21. }
```

HTML rendering

HTML rendering

Using `LoadHTMLGlob()` or `LoadHTMLFiles()`

```

1. func main() {
2.     router := gin.Default()
3.     router.LoadHTMLGlob("templates/*")
4.     //router.LoadHTMLFiles("templates/template1.html",
        "templates/template2.html")
5.     router.GET("/index", func(c *gin.Context) {
6.         c.HTML(http.StatusOK, "index.tmpl", gin.H{
7.             "title": "Main website",
8.         })
9.     })
10.    router.Run(":8080")
11. }
```

templates/index.tmpl

```

1. <html>
2.     <h1>
3.         {{ .title }}
4.     </h1>
5. </html>
```

Using templates with same name in different directories

```

1. func main() {
2.     router := gin.Default()
3.     router.LoadHTMLGlob("templates/**/*.html")
4.     router.GET("/posts/index", func(c *gin.Context) {
5.         c.HTML(http.StatusOK, "posts/index.tmpl", gin.H{
6.             "title": "Posts",
7.         })
8.     })
9.     router.GET("/users/index", func(c *gin.Context) {
10.        c.HTML(http.StatusOK, "users/index.tmpl", gin.H{
11.            "title": "Users",
```

```

12.         })
13.     })
14.     router.Run(":8080")
15. }

```

templates/posts/index.tpl

```

1.  {{ define "posts/index.tpl" }}
2.  <html><h1>
3.      {{ .title }}
4.  </h1>
5.  <p>Using posts/index.tpl</p>
6.  </html>
7.  {{ end }}

```

templates/users/index.tpl

```

1.  {{ define "users/index.tpl" }}
2.  <html><h1>
3.      {{ .title }}
4.  </h1>
5.  <p>Using users/index.tpl</p>
6.  </html>
7.  {{ end }}

```

Custom Template renderer

You can also use your own html template render

```

1.  import "html/template"
2.
3.  func main() {
4.      router := gin.Default()
5.      html := template.Must(template.ParseFiles("file1", "file2"))
6.      router.SetHTMLTemplate(html)
7.      router.Run(":8080")
8.  }

```

Custom Delimiters

You may use custom delims

```

1.     r := gin.Default()
2.     r.Delims("{{", "}}")
3.     r.LoadHTMLGlob("/path/to/templates"))

```

Custom Template Funcs

See the detail [example code](#).

main.go

```

1.  import (
2.      "fmt"
3.      "html/template"
4.      "net/http"
5.      "time"
6.
7.      "github.com/gin-gonic/gin"
8.  )
9.
10. func formatAsDate(t time.Time) string {
11.     year, month, day := t.Date()
12.     return fmt.Sprintf("%d%02d/%02d", year, month, day)
13. }
14.
15. func main() {
16.     router := gin.Default()
17.     router.Delims("{{", "}}")
18.     router.SetFuncMap(template.FuncMap{
19.         "formatAsDate": formatAsDate,
20.     })
21.     router.LoadHTMLFiles("./fixtures/basic/raw.tpl")
22.
23.     router.GET("/raw", func(c *gin.Context) {
24.         c.HTML(http.StatusOK, "raw.tpl", map[string]interface{}{
25.             "now": time.Date(2017, 07, 01, 0, 0, 0, 0, time.UTC),
26.         })
27.     })
28.
29.     router.Run(":8080")
30. }

```

raw.tpl

```
1. Date: {[{.now | formatAsDate}]}
```

Result:

```
1. Date: 2017/07/01
```


Multitemplate

Multitemplate

Gin allow by default use only one `html.Template`. Check [a multitemplate render](#) for using features like go 1.6 `block template` .

Redirects

Redirects

Issuing a HTTP redirect is easy. Both internal and external locations are supported.

```
1. r.GET("/test", func(c *gin.Context) {  
2.     c.Redirect(http.StatusMovedPermanently, "http://www.google.com/")  
3. })
```

Issuing a Router redirect, use `HandleContext` like below.

```
1. r.GET("/test", func(c *gin.Context) {  
2.     c.Request.URL.Path = "/test2"  
3.     r.HandleContext(c)  
4. })  
5. r.GET("/test2", func(c *gin.Context) {  
6.     c.JSON(200, gin.H{"hello": "world"})  
7. })
```

Custom Middleware

Custom Middleware

```
1. func Logger() gin.HandlerFunc {
2.     return func(c *gin.Context) {
3.         t := time.Now()
4.
5.         // Set example variable
6.         c.Set("example", "12345")
7.
8.         // before request
9.
10.        c.Next()
11.
12.        // after request
13.        latency := time.Since(t)
14.        log.Print(latency)
15.
16.        // access the status we are sending
17.        status := c.Writer.Status()
18.        log.Println(status)
19.    }
20. }
21.
22. func main() {
23.     r := gin.New()
24.     r.Use(Logger())
25.
26.     r.GET("/test", func(c *gin.Context) {
27.         example := c.MustGet("example").(string)
28.
29.         // it would print: "12345"
30.         log.Println(example)
31.     })
32.
33.     // Listen and serve on 0.0.0.0:8080
34.     r.Run(":8080")
35. }
```

Using BasicAuth() middleware

Using BasicAuth() middleware

```

1. // simulate some private data
2. var secrets = gin.H{
3.     "foo":    gin.H{"email": "foo@bar.com", "phone": "123433"},
4.     "austin": gin.H{"email": "austin@example.com", "phone": "666"},
5.     "lena":   gin.H{"email": "lena@guapa.com", "phone": "523443"},
6. }
7.
8. func main() {
9.     r := gin.Default()
10.
11.     // Group using gin.BasicAuth() middleware
12.     // gin.Accounts is a shortcut for map[string]string
13.     authorized := r.Group("/admin", gin.BasicAuth(gin.Accounts{
14.         "foo":    "bar",
15.         "austin": "1234",
16.         "lena":   "hello2",
17.         "manu":   "4321",
18.     })))
19.
20.     // /admin/secrets endpoint
21.     // hit "localhost:8080/admin/secrets"
22.     authorized.GET("/secrets", func(c *gin.Context) {
23.         // get user, it was set by the BasicAuth middleware
24.         user := c.MustGet(gin.AuthUserKey).(string)
25.         if secret, ok := secrets[user]; ok {
26.             c.JSON(http.StatusOK, gin.H{"user": user, "secret": secret})
27.         } else {
28.             c.JSON(http.StatusOK, gin.H{"user": user, "secret": "NO SECRET :
(")})
29.         }
30.     })
31.
32.     // Listen and serve on 0.0.0.0:8080
33.     r.Run(":8080")
34. }

```

Goroutines inside a middleware

Goroutines inside a middleware

When starting new Goroutines inside a middleware or handler, you **SHOULD NOT** use the original context inside it, you have to use a read-only copy.

```
1. func main() {
2.     r := gin.Default()
3.
4.     r.GET("/long_async", func(c *gin.Context) {
5.         // create copy to be used inside the goroutine
6.         cCp := c.Copy()
7.         go func() {
8.             // simulate a long task with time.Sleep(). 5 seconds
9.             time.Sleep(5 * time.Second)
10.
11.             // note that you are using the copied context "cCp", IMPORTANT
12.             log.Println("Done! in path " + cCp.Request.URL.Path)
13.         }()
14.     })
15.
16.     r.GET("/long_sync", func(c *gin.Context) {
17.         // simulate a long task with time.Sleep(). 5 seconds
18.         time.Sleep(5 * time.Second)
19.
20.         // since we are NOT using a goroutine, we do not have to copy the
           context
21.         log.Println("Done! in path " + c.Request.URL.Path)
22.     })
23.
24.     // Listen and serve on 0.0.0.0:8080
25.     r.Run(":8080")
26. }
```

Custom HTTP configuration

Custom HTTP configuration

Use `http.ListenAndServe()` directly, like this:

```
1. func main() {  
2.     router := gin.Default()  
3.     http.ListenAndServe(":8080", router)  
4. }
```

or

```
1. func main() {  
2.     router := gin.Default()  
3.  
4.     s := &http.Server{  
5.         Addr:           ":8080",  
6.         Handler:        router,  
7.         ReadTimeout:    10 * time.Second,  
8.         WriteTimeout:   10 * time.Second,  
9.         MaxHeaderBytes: 1 << 20,  
10.    }  
11.    s.ListenAndServe()  
12. }
```

Support Let's Encrypt

Support Let's Encrypt

example for 1-line LetsEncrypt HTTPS servers.

```
1. package main
2.
3. import (
4.     "log"
5.
6.     "github.com/gin-gonic/autotls"
7.     "github.com/gin-gonic/gin"
8. )
9.
10. func main() {
11.     r := gin.Default()
12.
13.     // Ping handler
14.     r.GET("/ping", func(c *gin.Context) {
15.         c.String(200, "pong")
16.     })
17.
18.     log.Fatal(autotls.Run(r, "example1.com", "example2.com"))
19. }
```

example for custom autocert manager.

```
1. package main
2.
3. import (
4.     "log"
5.
6.     "github.com/gin-gonic/autotls"
7.     "github.com/gin-gonic/gin"
8.     "golang.org/x/crypto/acme/autocert"
9. )
10.
11. func main() {
12.     r := gin.Default()
```

```
13.  
14.     // Ping handler  
15.     r.GET("/ping", func(c *gin.Context) {  
16.         c.String(200, "pong")  
17.     })  
18.  
19.     m := autocert.Manager{  
20.         Prompt:      autocert.AcceptTOS,  
21.         HostPolicy: autocert.HostWhitelist("example1.com", "example2.com"),  
22.         Cache:        autocert.DirCache("/var/www/.cache"),  
23.     }  
24.  
25.     log.Fatal(autotls.RunWithManager(r, &m))  
26. }
```


Run multiple service using Gin

Run multiple service using Gin

See the [question](#) and try the following example:

```
1. package main
2.
3. import (
4.     "log"
5.     "net/http"
6.     "time"
7.
8.     "github.com/gin-gonic/gin"
9.     "golang.org/x/sync/errgroup"
10. )
11.
12. var (
13.     g errgroup.Group
14. )
15.
16. func router01() http.Handler {
17.     e := gin.New()
18.     e.Use(gin.Recovery())
19.     e.GET("/", func(c *gin.Context) {
20.         c.JSON(
21.             http.StatusOK,
22.             gin.H{
23.                 "code": http.StatusOK,
24.                 "error": "Welcome server 01",
25.             },
26.         )
27.     })
28.
29.     return e
30. }
31.
32. func router02() http.Handler {
33.     e := gin.New()
34.     e.Use(gin.Recovery())
35.     e.GET("/", func(c *gin.Context) {
```

```
36.         c.JSON(  
37.             http.StatusOK,  
38.             gin.H{  
39.                 "code": http.StatusOK,  
40.                 "error": "Welcome server 02",  
41.             },  
42.         )  
43.     })  
44.  
45.     return e  
46. }  
47.  
48. func main() {  
49.     server01 := &http.Server{  
50.         Addr:         ":8080",  
51.         Handler:      router01(),  
52.         ReadTimeout:  5 * time.Second,  
53.         WriteTimeout: 10 * time.Second,  
54.     }  
55.  
56.     server02 := &http.Server{  
57.         Addr:         ":8081",  
58.         Handler:      router02(),  
59.         ReadTimeout:  5 * time.Second,  
60.         WriteTimeout: 10 * time.Second,  
61.     }  
62.  
63.     g.Go(func() error {  
64.         return server01.ListenAndServe()  
65.     })  
66.  
67.     g.Go(func() error {  
68.         return server02.ListenAndServe()  
69.     })  
70.  
71.     if err := g.Wait(); err != nil {  
72.         log.Fatal(err)  
73.     }  
74. }
```

Graceful restart or stop

Graceful restart or stop

Do you want to graceful restart or stop your web server?

There are some ways this can be done.

We can use [fvbock/endless](#) to replace the default `ListenAndServe`. Refer issue [#296](#) for more details.

```
1. router := gin.Default()
2. router.GET("/", handler)
3. // [...]
4. endless.ListenAndServe(":4242", router)
```

An alternative to endless:

- [manners](#): A polite Go HTTP server that shuts down gracefully.
- [graceful](#): Graceful is a Go package enabling graceful shutdown of an `http.Handler` server.
- [grace](#): Graceful restart & zero downtime deploy for Go servers.

If you are using Go 1.8, you may not need to use this library! Consider using `http.Server`'s built-in `Shutdown()` method for graceful shutdowns. See the full [graceful-shutdown](#) example with gin.

```
1. // +build go1.8
2.
3. package main
4.
5. import (
6.     "context"
7.     "log"
8.     "net/http"
9.     "os"
10.    "os/signal"
11.    "time"
12.
13.    "github.com/gin-gonic/gin"
14. )
15.
```

```
16. func main() {
17.     router := gin.Default()
18.     router.GET("/", func(c *gin.Context) {
19.         time.Sleep(5 * time.Second)
20.         c.String(http.StatusOK, "Welcome Gin Server")
21.     })
22.
23.     srv := &http.Server{
24.         Addr:    ":8080",
25.         Handler: router,
26.     }
27.
28.     go func() {
29.         // service connections
30.         if err := srv.ListenAndServe(); err != nil && err !=
http.ErrServerClosed {
31.             log.Fatalf("listen: %s\n", err)
32.         }
33.     }()
34.
35.     // Wait for interrupt signal to gracefully shutdown the server with
36.     // a timeout of 5 seconds.
37.     quit := make(chan os.Signal)
38.     signal.Notify(quit, os.Interrupt)
39.     <-quit
40.     log.Println("Shutdown Server ...")
41.
42.     ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
43.     defer cancel()
44.     if err := srv.Shutdown(ctx); err != nil {
45.         log.Fatal("Server Shutdown:", err)
46.     }
47.     log.Println("Server exiting")
48. }
```

Build a single binary with templates

Build a single binary with templates

You can build a server into a single binary containing templates by using [go-assets](#).

```

1. func main() {
2.     r := gin.New()
3.
4.     t, err := loadTemplate()
5.     if err != nil {
6.         panic(err)
7.     }
8.     r.SetHTMLTemplate(t)
9.
10.    r.GET("/", func(c *gin.Context) {
11.        c.HTML(http.StatusOK, "/html/index.tpl", nil)
12.    })
13.    r.Run(":8080")
14. }
15.
16. // loadTemplate loads templates embedded by go-assets-builder
17. func loadTemplate() (*template.Template, error) {
18.     t := template.New("")
19.     for name, file := range Assets.Files {
20.         if file.IsDir() || !strings.HasSuffix(name, ".tpl") {
21.             continue
22.         }
23.         h, err := ioutil.ReadAll(file)
24.         if err != nil {
25.             return nil, err
26.         }
27.         t, err = t.New(name).Parse(string(h))
28.         if err != nil {
29.             return nil, err
30.         }
31.     }
32.     return t, nil
33. }

```

See a complete example in the `examples/assets-in-binary` directory.

Bind form-data request with custom struct

Bind form-data request with custom struct

The follow example using custom struct:

```

1. type StructA struct {
2.     FieldA string `form:"field_a"`
3. }
4.
5. type StructB struct {
6.     NestedStruct StructA
7.     FieldB string `form:"field_b"`
8. }
9.
10. type StructC struct {
11.     NestedStructPointer *StructA
12.     FieldC string `form:"field_c"`
13. }
14.
15. type StructD struct {
16.     NestedAnonyStruct struct {
17.         FieldX string `form:"field_x"`
18.     }
19.     FieldD string `form:"field_d"`
20. }
21.
22. func GetDataB(c *gin.Context) {
23.     var b StructB
24.     c.Bind(&b)
25.     c.JSON(200, gin.H{
26.         "a": b.NestedStruct,
27.         "b": b.FieldB,
28.     })
29. }
30.
31. func GetDataC(c *gin.Context) {
32.     var b StructC
33.     c.Bind(&b)
34.     c.JSON(200, gin.H{
35.         "a": b.NestedStructPointer,

```

```

36.         "c": b.FieldC,
37.     })
38. }
39.
40. func GetDataD(c *gin.Context) {
41.     var b StructD
42.     c.Bind(&b)
43.     c.JSON(200, gin.H{
44.         "x": b.NestedAnonyStruct,
45.         "d": b.FieldD,
46.     })
47. }
48.
49. func main() {
50.     r := gin.Default()
51.     r.GET("/getb", GetDataB)
52.     r.GET("/getc", GetDataC)
53.     r.GET("/getd", GetDataD)
54.
55.     r.Run()
56. }

```

Using the command `curl` command result:

```

1. $ curl "http://localhost:8080/getb?field_a=hello&field_b=world"
2. {"a":{"FieldA":"hello"},"b":"world"}
3. $ curl "http://localhost:8080/getc?field_a=hello&field_c=world"
4. {"a":{"FieldA":"hello"},"c":"world"}
5. $ curl "http://localhost:8080/getd?field_x=hello&field_d=world"
6. {"d":"world","x":{"FieldX":"hello"}}

```

NOTE: NOT support the follow style struct:

```

1. type StructX struct {
2.     X struct {} `form:"name_x"` // HERE have form
3. }
4.
5. type StructY struct {
6.     Y StructX `form:"name_y"` // HERE hava form
7. }
8.
9. type StructZ struct {

```



```
10.     Z *StructZ `form:"name_z"` // HERE hava form
11. }
```

In a word, only support nested custom struct which have no `form` now.

Try to bind body into different structs

Try to bind body into different structs

The normal methods for binding request body consumes `c.Request.Body` and they cannot be called multiple times.

```

1. type formA struct {
2.     Foo string `json:"foo" xml:"foo" binding:"required"`
3. }
4.
5. type formB struct {
6.     Bar string `json:"bar" xml:"bar" binding:"required"`
7. }
8.
9. func SomeHandler(c *gin.Context) {
10.    objA := formA{}
11.    objB := formB{}
12.    // This c.ShouldBind consumes c.Request.Body and it cannot be reused.
13.    if errA := c.ShouldBind(&objA); errA == nil {
14.        c.String(http.StatusOK, `the body should be formA`)
15.        // Always an error is occurred by this because c.Request.Body is EOF now.
16.    } else if errB := c.ShouldBind(&objB); errB == nil {
17.        c.String(http.StatusOK, `the body should be formB`)
18.    } else {
19.        ...
20.    }
21. }
```

For this, you can use `c.ShouldBindBodyWith` .

```

1. func SomeHandler(c *gin.Context) {
2.    objA := formA{}
3.    objB := formB{}
4.    // This reads c.Request.Body and stores the result into the context.
5.    if errA := c.ShouldBindBodyWith(&objA, binding.JSON); errA == nil {
6.        c.String(http.StatusOK, `the body should be formA`)
7.        // At this time, it reuses body stored in the context.
8.    } else if errB := c.ShouldBindBodyWith(&objB, binding.JSON); errB == nil {
```

```
9.      c.String(http.StatusOK, `the body should be formB JSON`)
10.     // And it can accepts other formats
11.     } else if errB2 := c.ShouldBindBodyWith(&objB, binding.XML); errB2 == nil {
12.         c.String(http.StatusOK, `the body should be formB XML`)
13.     } else {
14.         ...
15.     }
16. }
```

- `c.ShouldBindBodyWith` stores body into the context before binding. This has a slight impact to performance, so you should not use this method if you are enough to call binding at once.
- This feature is only needed for some formats – `JSON`, `XML`, `MsgPack`, `ProtoBuf`. For other formats, `Query`, `Form`, `FormPost`, `FormMultipart`, can be called by `c.ShouldBind()` multiple times without any damage to performance (See [#1341](#)).

http2 server push

http2 server push

http.Pusher is supported only **go1.8+**. See the [golang blog](#) for detail information.

```
1. package main
2.
3. import (
4.     "html/template"
5.     "log"
6.
7.     "github.com/gin-gonic/gin"
8. )
9.
10. var html = template.Must(template.New("https").Parse(`
11. <html>
12. <head>
13.   <title>Https Test</title>
14.   <script src="/assets/app.js"></script>
15. </head>
16. <body>
17.   <h1 style="color:red;">Welcome, Ginner!</h1>
18. </body>
19. </html>
20. `))
21.
22. func main() {
23.     r := gin.Default()
24.     r.Static("/assets", "./assets")
25.     r.SetHTMLTemplate(html)
26.
27.     r.GET("/", func(c *gin.Context) {
28.         if pusher := c.Writer.Pusher(); pusher != nil {
29.             // use pusher.Push() to do server push
30.             if err := pusher.Push("/assets/app.js", nil); err != nil {
31.                 log.Printf("Failed to push: %v", err)
32.             }
33.         }
34.         c.HTML(200, "https", gin.H{
```

```
35.         "status": "success",
36.     })
37. })
38.
39. // Listen and Server in https://127.0.0.1:8080
40. r.RunTLS(":8080", "./testdata/server.pem", "./testdata/server.key")
41. }
```

Testing

Testing

The `net/http/httptest` package is preferable way for HTTP testing.

```
1. package main
2.
3. func setupRouter() *gin.Engine {
4.     r := gin.Default()
5.     r.GET("/ping", func(c *gin.Context) {
6.         c.String(200, "pong")
7.     })
8.     return r
9. }
10.
11. func main() {
12.     r := setupRouter()
13.     r.Run(":8080")
14. }
```

Test for code example above:

```
1. package main
2.
3. import (
4.     "net/http"
5.     "net/http/httptest"
6.     "testing"
7.
8.     "github.com/stretchr/testify/assert"
9. )
10.
11. func TestPingRoute(t *testing.T) {
12.     router := setupRouter()
13.
14.     w := httptest.NewRecorder()
15.     req, _ := http.NewRequest("GET", "/ping", nil)
16.     router.ServeHTTP(w, req)
17. }
```

```
18.     assert.Equal(t, 200, w.Code)
19.     assert.Equal(t, "pong", w.Body.String())
20. }
```

Users

Users

used by 3.6k projects

Awesome project lists using [Gin](#) web framework.

- [drone](#): Drone is a Continuous Delivery platform built on Docker, written in Go
- [gorush](#): A push notification server written in Go.

AUTHORS

List of all the awesome people working to make Gin the best Web Framework in Go.

gin 0.x series authors

Maintainer: Manu Martinez-Almeida ([@manucorporat](#)), Javier Provecho ([@javierprovecho](#))

People and companies, who have contributed, in alphabetical order.

[@858806258](#) (杰哥)

- Fix typo in example

[@achedeuzot](#) (Klemen Sever)

- Fix newline debug printing

[@adammck](#) (Adam Mckaig)

- Add MIT license

[@AlexanderChen1989](#) (Alexander)

- Typos in README

[@alexanderdidenko](#) (Aleksandr Didenko)

- Add support multipart/form-data

[@alexandernyquist](#) (Alexander Nyquist)

- Using `template.Must` to fix multiple return issue
- ★ Added support for `OPTIONS` verb
- ★ Setting response headers before calling `WriteHeader`
- Improved documentation for model binding
- ★ Added `Content.Redirect()`
- ★ Added tons of Unit tests

[@austinheap](#) (Austin Heap)

- Added travis CI integration

@andredublin (Andre Dublin)

- Fix typo in comment

@bredov (Ludwig Valda Vasquez)

- Fix html templating in debug mode

@bluele (Jun Kimura)

- Fixes code examples in README

@chad-russell

- ★ Support for serializing gin.H into XML

@dickeyxxx (Jeff Dickey)

- Typos in README
- Add example about serving static files

@donileo (Adonis)

- Add NoMethod handler

@dutchcoders (DutchCoders)

- ★ Fix security bug that allows client to spoof ip
- Fix typo. r.HTMLTemplates -> SetHTMLTemplate

@el3ctro- (Joshua Loper)

- Fix typo in example

@ethankan (Ethan Kan)

- Unsigned integers in binding

(Evgeny Persienko)

- Validate sub structures

@frankbille (Frank Bille)

- Add support for HTTP Realm Auth

@fmd (Fareed Dudhia)

- Fix typo. SetHTTPTemplate -> SetHTMLTemplate

@ironiridis (Christopher Harrington)

- Remove old reference

@jammie-stackhouse (Jamie Stackhouse)

- Add more shortcuts for router methods

@jasonrhansen

- Fix spelling and grammar errors in documentation

@JasonSoft (Jason Lee)

- Fix typo in comment

@joiggama (Ignacio Galindo)

- Add utf-8 charset header on renders

@julienschmidt (Julien Schmidt)

- gofmt the code examples

@kelcecil (Kel Cecil)

- Fix readme typo

@kyledinh (Kyle Dinh)

- Adds RunTLS()

@LinusU (Linus Unneback)

- Small fixes in README

@loongmxbt (Saint Asky)

- Fix typo in example

@lucas-clemente (Lucas Clemente)

- ★ work around path.Join removing trailing slashes from routes

@mattn (Yasuhiro Matsumoto)

- Improve color logger

@mdigger (Dmitry Sedykh)

- Fixes Form binding when content-type is x-www-form-urlencoded
- No repeat call `c.Writer.Status()` in `gin.Logger`
- Fixes Content-Type for json render

@mirzac (Mirza Ceric)

- Fix debug printing

@mopemope (Yutaka Matsubara)

- ★ Adds Godep support (Dependencies Manager)
- Fix variadic parameter in the flexible render API
- Fix Corrupted plain render
- Add Pluggable View Renderer Example

@msemenistyi (Mykyta Semenistyi)

- update Readme.md. Add code to String method

@msoedov (Sasha Myasoedov)

- ★ Adds tons of unit tests.

@ngerakines (Nick Gerakines)

- ★ Improves API, `c.GET()` doesn't panic
- Adds `MustGet()` method

@r8k (Rajiv Kilaparti)

- Fix Port usage in README.

@rayrod2030 (Ray Rodriguez)

- Fix typo in example

@rns

- Fix typo in example

@RobAWilkinson (Robert Wilkinson)

- Add example of forms and params

@rogierlommers (Rogier Lommers)

- Add updated static serve example

@se77en (Damon Zhao)

- Improve color logging

@silasb (Silas Baronda)

- Fixing quotes in README

@SkuliOskarsson (Skuli Oskarsson)

- Fixes some texts in README II

@slimmy (Jimmy Pettersson)

- Added messages for required bindings

@smira (Andrey Smirnov)

- Add support for ignored/unexported fields in binding

@superalsrk (SRK.Lyu)

- Update httprouter godeps

@tebeka (Miki Tebeka)

- Use net/http constants instead of numeric values

@techjanitor

- Update context.go reserved IPs

@yosssi (Keiji Yoshida)

- Fix link in README

@yuyabee

- Fixed README

Benchmark System

Benchmark System

VM HOST: DigitalOcean

Machine: 4 CPU, 8 GB RAM. Ubuntu 16.04.2 x64

Date: July 19th, 2017

Go Version: 1.8.3 linux/amd64

Source: [Go HTTP Router Benchmark](#)

Static Routes: 157

1.	Gin:	30512 Bytes
2.		
3.	HttpServeMux:	17344 Bytes
4.	Ace:	30080 Bytes
5.	Bear:	30472 Bytes
6.	Beego:	96408 Bytes
7.	Bone:	37904 Bytes
8.	Denco:	10464 Bytes
9.	Echo:	73680 Bytes
10.	GocraftWeb:	55720 Bytes
11.	Goji:	27200 Bytes
12.	Gojiv2:	104464 Bytes
13.	GoJsonRest:	136472 Bytes
14.	GoRestful:	914904 Bytes
15.	GorillaMux:	675568 Bytes
16.	HttpRouter:	21128 Bytes
17.	HttpTreeMux:	73448 Bytes
18.	Kocha:	115072 Bytes
19.	LARS:	30120 Bytes
20.	Macaron:	37984 Bytes
21.	Martini:	310832 Bytes
22.	Pat:	20464 Bytes
23.	Possum:	91328 Bytes
24.	R2router:	23712 Bytes
25.	Rivet:	23880 Bytes
26.	Tango:	28008 Bytes
27.	TigerTonic:	80368 Bytes

```
28. Traffic:      626480 Bytes
29. Vulcan:      369064 Bytes
```

GithubAPI Routes: 203

```
1. Gin:          52672 Bytes
2.
3. Ace:          48992 Bytes
4. Bear:         161592 Bytes
5. Beego:        147992 Bytes
6. Bone:         97728 Bytes
7. Denco:        36440 Bytes
8. Echo:         95672 Bytes
9. GocraftWeb:   95640 Bytes
10. Goji:         86088 Bytes
11. Gojiv2:       144392 Bytes
12. GoJsonRest:  134648 Bytes
13. GoRestful:   1410760 Bytes
14. GorillaMux:  1509488 Bytes
15. HttpRouter:   37464 Bytes
16. HttpTreeMux: 78800 Bytes
17. Kocha:       785408 Bytes
18. LARS:        49032 Bytes
19. Macaron:     132712 Bytes
20. Martini:     564352 Bytes
21. Pat:         21200 Bytes
22. Possum:      83888 Bytes
23. R2router:    47104 Bytes
24. Rivet:       42840 Bytes
25. Tango:       54584 Bytes
26. TigerTonic:  96384 Bytes
27. Traffic:     1061920 Bytes
28. Vulcan:      465296 Bytes
```

GPlusAPI Routes: 13

```
1. Gin:          3968 Bytes
2.
3. Ace:          3600 Bytes
4. Bear:         7112 Bytes
```

5.	Beego:	10048 Bytes
6.	Bone:	6480 Bytes
7.	Denco:	3256 Bytes
8.	Echo:	9000 Bytes
9.	GocraftWeb:	7496 Bytes
10.	Goji:	2912 Bytes
11.	Gojiv2:	7376 Bytes
12.	GoJsonRest:	11544 Bytes
13.	GoRestful:	88776 Bytes
14.	GorillaMux:	71488 Bytes
15.	HttpRouter:	2712 Bytes
16.	HttpTreeMux:	7440 Bytes
17.	Kocha:	128880 Bytes
18.	LARS:	3640 Bytes
19.	Macaron:	8656 Bytes
20.	Martini:	23936 Bytes
21.	Pat:	1856 Bytes
22.	Possum:	7248 Bytes
23.	R2router:	3928 Bytes
24.	Rivet:	3064 Bytes
25.	Tango:	4912 Bytes
26.	TigerTonic:	9408 Bytes
27.	Traffic:	49472 Bytes
28.	Vulcan:	25496 Bytes

ParseAPI Routes: 26

1.	Gin:	6928 Bytes
2.		
3.	Ace:	6592 Bytes
4.	Bear:	12320 Bytes
5.	Beego:	18960 Bytes
6.	Bone:	11024 Bytes
7.	Denco:	4184 Bytes
8.	Echo:	11168 Bytes
9.	GocraftWeb:	12800 Bytes
10.	Goji:	5232 Bytes
11.	Gojiv2:	14464 Bytes
12.	GoJsonRest:	14216 Bytes
13.	GoRestful:	127368 Bytes
14.	GorillaMux:	123016 Bytes
15.	HttpRouter:	4976 Bytes
16.	HttpTreeMux:	7848 Bytes
17.	Kocha:	181712 Bytes
18.	LARS:	6632 Bytes
19.	Macaron:	13648 Bytes
20.	Martini:	45952 Bytes
21.	Pat:	2560 Bytes
22.	Possum:	9200 Bytes
23.	R2router:	7056 Bytes
24.	Rivet:	5680 Bytes
25.	Tango:	8664 Bytes
26.	TigerTonic:	9840 Bytes
27.	Traffic:	93480 Bytes
28.	Vulcan:	44504 Bytes

Static Routes

1.	BenchmarkGin_StaticAll	50000	34506 ns/op
	0 B/op	0 allocs/op	
2.			
3.	BenchmarkAce_StaticAll	30000	49657 ns/op
	0 B/op	0 allocs/op	
4.			
4.	BenchmarkHttpServeMux_StaticAll	2000	1183737 ns/op

	96 B/op	8 allocs/op		
5.	BenchmarkBeego_StaticAll	5000	412621 ns/op	
	57776 B/op	628 allocs/op		
6.	BenchmarkBear_StaticAll	10000	149242 ns/op	
	20336 B/op	461 allocs/op		
7.	BenchmarkBone_StaticAll	10000	118583 ns/op	
	0 B/op	0 allocs/op		
8.	BenchmarkDenco_StaticAll	100000	13247 ns/op	
	0 B/op	0 allocs/op		
9.	BenchmarkEcho_StaticAll	20000	79914 ns/op	
	5024 B/op	157 allocs/op		
10.	BenchmarkGocraftWeb_StaticAll	10000	211823 ns/op	
	46440 B/op	785 allocs/op		
11.	BenchmarkGoji_StaticAll	10000	109390 ns/op	
	0 B/op	0 allocs/op		
12.	BenchmarkGojiv2_StaticAll	3000	415533 ns/op	
	145696 B/op	1099 allocs/op		
13.	BenchmarkGoJsonRest_StaticAll	5000	364403 ns/op	
	51653 B/op	1727 allocs/op		
14.	BenchmarkGoRestful_StaticAll	500	2578579 ns/op	
	314936 B/op	3144 allocs/op		
15.	BenchmarkGorillaMux_StaticAll	500	2704856 ns/op	
	115648 B/op	1578 allocs/op		
16.	BenchmarkHttpRouter_StaticAll	100000	18541 ns/op	
	0 B/op	0 allocs/op		
17.	BenchmarkHttpTreeMux_StaticAll	100000	22332 ns/op	
	0 B/op	0 allocs/op		
18.	BenchmarkKocha_StaticAll	50000	31176 ns/op	
	0 B/op	0 allocs/op		
19.	BenchmarkLARS_StaticAll	50000	40840 ns/op	
	0 B/op	0 allocs/op		
20.	BenchmarkMacaron_StaticAll	5000	517656 ns/op	
	120576 B/op	1413 allocs/op		
21.	BenchmarkMartini_StaticAll	300	4462289 ns/op	
	125442 B/op	1717 allocs/op		
22.	BenchmarkPat_StaticAll	500	2157275 ns/op	
	533904 B/op	11123 allocs/op		
23.	BenchmarkPossum_StaticAll	10000	254701 ns/op	
	65312 B/op	471 allocs/op		
24.	BenchmarkR2router_StaticAll	10000	133956 ns/op	
	22608 B/op	628 allocs/op		
25.	BenchmarkRivet_StaticAll	30000	46812 ns/op	

	0 B/op	0 allocs/op		
26.	BenchmarkTango_StaticAll	5000	390613 ns/op	
	39225 B/op	1256 allocs/op		
27.	BenchmarkTigerTonic_StaticAll	20000	88060 ns/op	
	7504 B/op	157 allocs/op		
28.	BenchmarkTraffic_StaticAll	500	2910236 ns/op	
	729736 B/op	14287 allocs/op		
29.	BenchmarkVulcan_StaticAll	5000	277366 ns/op	
	15386 B/op	471 allocs/op		

Micro Benchmarks

1.	BenchmarkGin_Param	20000000	113 ns/op	
	0 B/op	0 allocs/op		
2.				
3.	BenchmarkAce_Param	5000000	375 ns/op	
	32 B/op	1 allocs/op		
4.	BenchmarkBear_Param	1000000	1709 ns/op	
	456 B/op	5 allocs/op		
5.	BenchmarkBeego_Param	1000000	2484 ns/op	
	368 B/op	4 allocs/op		
6.	BenchmarkBone_Param	1000000	2391 ns/op	
	688 B/op	5 allocs/op		
7.	BenchmarkDenco_Param	10000000	240 ns/op	
	32 B/op	1 allocs/op		
8.	BenchmarkEcho_Param	5000000	366 ns/op	
	32 B/op	1 allocs/op		
9.	BenchmarkGocraftWeb_Param	1000000	2343 ns/op	
	648 B/op	8 allocs/op		
10.	BenchmarkGoji_Param	1000000	1197 ns/op	
	336 B/op	2 allocs/op		
11.	BenchmarkGojiv2_Param	1000000	2771 ns/op	
	944 B/op	8 allocs/op		
12.	BenchmarkGoJsonRest_Param	1000000	2993 ns/op	
	649 B/op	13 allocs/op		
13.	BenchmarkGoRestful_Param	200000	8860 ns/op	
	2296 B/op	21 allocs/op		
14.	BenchmarkGorillaMux_Param	500000	4461 ns/op	
	1056 B/op	11 allocs/op		
15.	BenchmarkHttpRouter_Param	10000000	175 ns/op	
	32 B/op	1 allocs/op		

16.	BenchmarkHttpTreeMux_Param	1000000	1167 ns/op
	352 B/op	3 allocs/op	
17.	BenchmarkKocha_Param	3000000	429 ns/op
	56 B/op	3 allocs/op	
18.	BenchmarkLARS_Param	10000000	134 ns/op
	0 B/op	0 allocs/op	
19.	BenchmarkMacaron_Param	500000	4635 ns/op
	1056 B/op	10 allocs/op	
20.	BenchmarkMartini_Param	200000	9933 ns/op
	1072 B/op	10 allocs/op	
21.	BenchmarkPat_Param	1000000	2929 ns/op
	648 B/op	12 allocs/op	
22.	BenchmarkPossum_Param	1000000	2503 ns/op
	560 B/op	6 allocs/op	
23.	BenchmarkR2router_Param	1000000	1507 ns/op
	432 B/op	5 allocs/op	
24.	BenchmarkRivet_Param	5000000	297 ns/op
	48 B/op	1 allocs/op	
25.	BenchmarkTango_Param	1000000	1862 ns/op
	248 B/op	8 allocs/op	
26.	BenchmarkTigerTonic_Param	500000	5660 ns/op
	992 B/op	17 allocs/op	
27.	BenchmarkTraffic_Param	200000	8408 ns/op
	1960 B/op	21 allocs/op	
28.	BenchmarkVulcan_Param	2000000	963 ns/op
	98 B/op	3 allocs/op	
29.	BenchmarkAce_Param5	2000000	740 ns/op
	160 B/op	1 allocs/op	
30.	BenchmarkBear_Param5	1000000	2777 ns/op
	501 B/op	5 allocs/op	
31.	BenchmarkBeego_Param5	1000000	3740 ns/op
	368 B/op	4 allocs/op	
32.	BenchmarkBone_Param5	1000000	2950 ns/op
	736 B/op	5 allocs/op	
33.	BenchmarkDenco_Param5	2000000	644 ns/op
	160 B/op	1 allocs/op	
34.	BenchmarkEcho_Param5	3000000	558 ns/op
	32 B/op	1 allocs/op	
35.	BenchmarkGin_Param5	10000000	198 ns/op
	0 B/op	0 allocs/op	
36.	BenchmarkGocraftWeb_Param5	500000	3870 ns/op
	920 B/op	11 allocs/op	

37.	BenchmarkGoji_Param5	1000000	1746 ns/op
	336 B/op	2 allocs/op	
38.	BenchmarkGojiv2_Param5	1000000	3214 ns/op
	1008 B/op	8 allocs/op	
39.	BenchmarkGoJsonRest_Param5	500000	5509 ns/op
	1097 B/op	16 allocs/op	
40.	BenchmarkGoRestful_Param5	200000	11232 ns/op
	2392 B/op	21 allocs/op	
41.	BenchmarkGorillaMux_Param5	300000	7777 ns/op
	1184 B/op	11 allocs/op	
42.	BenchmarkHttpRouter_Param5	3000000	631 ns/op
	160 B/op	1 allocs/op	
43.	BenchmarkHttpTreeMux_Param5	1000000	2800 ns/op
	576 B/op	6 allocs/op	
44.	BenchmarkKocha_Param5	1000000	2053 ns/op
	440 B/op	10 allocs/op	
45.	BenchmarkLARS_Param5	10000000	232 ns/op
	0 B/op	0 allocs/op	
46.	BenchmarkMacaron_Param5	500000	5888 ns/op
	1056 B/op	10 allocs/op	
47.	BenchmarkMartini_Param5	200000	12807 ns/op
	1232 B/op	11 allocs/op	
48.	BenchmarkPat_Param5	300000	7320 ns/op
	964 B/op	32 allocs/op	
49.	BenchmarkPossum_Param5	1000000	2495 ns/op
	560 B/op	6 allocs/op	
50.	BenchmarkR2router_Param5	1000000	1844 ns/op
	432 B/op	5 allocs/op	
51.	BenchmarkRivet_Param5	2000000	935 ns/op
	240 B/op	1 allocs/op	
52.	BenchmarkTango_Param5	1000000	2327 ns/op
	360 B/op	8 allocs/op	
53.	BenchmarkTigerTonic_Param5	100000	18514 ns/op
	2551 B/op	43 allocs/op	
54.	BenchmarkTraffic_Param5	200000	11997 ns/op
	2248 B/op	25 allocs/op	
55.	BenchmarkVulcan_Param5	1000000	1333 ns/op
	98 B/op	3 allocs/op	
56.	BenchmarkAce_Param20	1000000	2031 ns/op
	640 B/op	1 allocs/op	
57.	BenchmarkBear_Param20	200000	7285 ns/op
	1664 B/op	5 allocs/op	

58.	BenchmarkBeego_Param20	300000	6224 ns/op
	368 B/op	4 allocs/op	
59.	BenchmarkBone_Param20	200000	8023 ns/op
	1903 B/op	5 allocs/op	
60.	BenchmarkDenco_Param20	1000000	2262 ns/op
	640 B/op	1 allocs/op	
61.	BenchmarkEcho_Param20	1000000	1387 ns/op
	32 B/op	1 allocs/op	
62.	BenchmarkGin_Param20	3000000	503 ns/op
	0 B/op	0 allocs/op	
63.	BenchmarkGocraftWeb_Param20	100000	14408 ns/op
	3795 B/op	15 allocs/op	
64.	BenchmarkGoji_Param20	500000	5272 ns/op
	1247 B/op	2 allocs/op	
65.	BenchmarkGojiv2_Param20	1000000	4163 ns/op
	1248 B/op	8 allocs/op	
66.	BenchmarkGoJsonRest_Param20	100000	17866 ns/op
	4485 B/op	20 allocs/op	
67.	BenchmarkGoRestful_Param20	100000	21022 ns/op
	4724 B/op	23 allocs/op	
68.	BenchmarkGorillaMux_Param20	100000	17055 ns/op
	3547 B/op	13 allocs/op	
69.	BenchmarkHttpRouter_Param20	1000000	1748 ns/op
	640 B/op	1 allocs/op	
70.	BenchmarkHttpTreeMux_Param20	200000	12246 ns/op
	3196 B/op	10 allocs/op	
71.	BenchmarkKocha_Param20	300000	6861 ns/op
	1808 B/op	27 allocs/op	
72.	BenchmarkLARS_Param20	3000000	526 ns/op
	0 B/op	0 allocs/op	
73.	BenchmarkMacaron_Param20	100000	13069 ns/op
	2906 B/op	12 allocs/op	
74.	BenchmarkMartini_Param20	100000	23602 ns/op
	3597 B/op	13 allocs/op	
75.	BenchmarkPat_Param20	50000	32143 ns/op
	4688 B/op	111 allocs/op	
76.	BenchmarkPossum_Param20	1000000	2396 ns/op
	560 B/op	6 allocs/op	
77.	BenchmarkR2router_Param20	200000	8907 ns/op
	2283 B/op	7 allocs/op	
78.	BenchmarkRivet_Param20	1000000	3280 ns/op
	1024 B/op	1 allocs/op	

79.	BenchmarkTango_Param20	500000	4640 ns/op
	856 B/op	8 allocs/op	
80.	BenchmarkTigerTonic_Param20	20000	67581 ns/op
	10532 B/op	138 allocs/op	
81.	BenchmarkTraffic_Param20	50000	40313 ns/op
	7941 B/op	45 allocs/op	
82.	BenchmarkVulcan_Param20	1000000	2264 ns/op
	98 B/op	3 allocs/op	
83.	BenchmarkAce_ParamWrite	3000000	532 ns/op
	40 B/op	2 allocs/op	
84.	BenchmarkBear_ParamWrite	1000000	1778 ns/op
	456 B/op	5 allocs/op	
85.	BenchmarkBeego_ParamWrite	1000000	2596 ns/op
	376 B/op	5 allocs/op	
86.	BenchmarkBone_ParamWrite	1000000	2519 ns/op
	688 B/op	5 allocs/op	
87.	BenchmarkDenco_ParamWrite	5000000	411 ns/op
	32 B/op	1 allocs/op	
88.	BenchmarkEcho_ParamWrite	2000000	718 ns/op
	40 B/op	2 allocs/op	
89.	BenchmarkGin_ParamWrite	5000000	283 ns/op
	0 B/op	0 allocs/op	
90.	BenchmarkGocraftWeb_ParamWrite	1000000	2561 ns/op
	656 B/op	9 allocs/op	
91.	BenchmarkGoji_ParamWrite	1000000	1378 ns/op
	336 B/op	2 allocs/op	
92.	BenchmarkGojiv2_ParamWrite	1000000	3128 ns/op
	976 B/op	10 allocs/op	
93.	BenchmarkGoJsonRest_ParamWrite	500000	4446 ns/op
	1128 B/op	18 allocs/op	
94.	BenchmarkGoRestful_ParamWrite	200000	10291 ns/op
	2304 B/op	22 allocs/op	
95.	BenchmarkGorillaMux_ParamWrite	500000	5153 ns/op
	1064 B/op	12 allocs/op	
96.	BenchmarkHttpRouter_ParamWrite	5000000	263 ns/op
	32 B/op	1 allocs/op	
97.	BenchmarkHttpTreeMux_ParamWrite	1000000	1351 ns/op
	352 B/op	3 allocs/op	
98.	BenchmarkKocha_ParamWrite	3000000	538 ns/op
	56 B/op	3 allocs/op	
99.	BenchmarkLARS_ParamWrite	5000000	316 ns/op
	0 B/op	0 allocs/op	

100.	BenchmarkMacaron_ParamWrite	500000	5756 ns/op
	1160 B/op	14 allocs/op	
101.	BenchmarkMartini_ParamWrite	200000	13097 ns/op
	1176 B/op	14 allocs/op	
102.	BenchmarkPat_ParamWrite	500000	4954 ns/op
	1072 B/op	17 allocs/op	
103.	BenchmarkPossum_ParamWrite	1000000	2499 ns/op
	560 B/op	6 allocs/op	
104.	BenchmarkR2router_ParamWrite	1000000	1531 ns/op
	432 B/op	5 allocs/op	
105.	BenchmarkRivet_ParamWrite	3000000	570 ns/op
	112 B/op	2 allocs/op	
106.	BenchmarkTango_ParamWrite	2000000	957 ns/op
	136 B/op	4 allocs/op	
107.	BenchmarkTigerTonic_ParamWrite	200000	7025 ns/op
	1424 B/op	23 allocs/op	
108.	BenchmarkTraffic_ParamWrite	200000	10112 ns/op
	2384 B/op	25 allocs/op	
109.	BenchmarkVulcan_ParamWrite	1000000	1006 ns/op
	98 B/op	3 allocs/op	

GitHub

1.	BenchmarkGin_GithubStatic	10000000	156 ns/op
	0 B/op	0 allocs/op	
2.			
3.	BenchmarkAce_GithubStatic	5000000	294 ns/op
	0 B/op	0 allocs/op	
4.	BenchmarkBear_GithubStatic	2000000	893 ns/op
	120 B/op	3 allocs/op	
5.	BenchmarkBeego_GithubStatic	1000000	2491 ns/op
	368 B/op	4 allocs/op	
6.	BenchmarkBone_GithubStatic	50000	25300 ns/op
	2880 B/op	60 allocs/op	
7.	BenchmarkDenco_GithubStatic	20000000	76.0 ns/op
	0 B/op	0 allocs/op	
8.	BenchmarkEcho_GithubStatic	2000000	516 ns/op
	32 B/op	1 allocs/op	
9.	BenchmarkGocraftWeb_GithubStatic	1000000	1448 ns/op
	296 B/op	5 allocs/op	
10.	BenchmarkGoji_GithubStatic	3000000	496 ns/op

	0 B/op	0 allocs/op		
11.	BenchmarkGojiv2_GithubStatic	1000000	2941 ns/op	
	928 B/op	7 allocs/op		
12.	BenchmarkGoRestful_GithubStatic	100000	27256 ns/op	
	3224 B/op	22 allocs/op		
13.	BenchmarkGoJsonRest_GithubStatic	1000000	2196 ns/op	
	329 B/op	11 allocs/op		
14.	BenchmarkGorillaMux_GithubStatic	50000	31617 ns/op	
	736 B/op	10 allocs/op		
15.	BenchmarkHttpRouter_GithubStatic	20000000	88.4 ns/op	
	0 B/op	0 allocs/op		
16.	BenchmarkHttpTreeMux_GithubStatic	10000000	134 ns/op	
	0 B/op	0 allocs/op		
17.	BenchmarkKocha_GithubStatic	20000000	113 ns/op	
	0 B/op	0 allocs/op		
18.	BenchmarkLARS_GithubStatic	10000000	195 ns/op	
	0 B/op	0 allocs/op		
19.	BenchmarkMacaron_GithubStatic	500000	3740 ns/op	
	768 B/op	9 allocs/op		
20.	BenchmarkMartini_GithubStatic	50000	27673 ns/op	
	768 B/op	9 allocs/op		
21.	BenchmarkPat_GithubStatic	100000	19470 ns/op	
	3648 B/op	76 allocs/op		
22.	BenchmarkPossum_GithubStatic	1000000	1729 ns/op	
	416 B/op	3 allocs/op		
23.	BenchmarkR2router_GithubStatic	2000000	879 ns/op	
	144 B/op	4 allocs/op		
24.	BenchmarkRivet_GithubStatic	10000000	231 ns/op	
	0 B/op	0 allocs/op		
25.	BenchmarkTango_GithubStatic	1000000	2325 ns/op	
	248 B/op	8 allocs/op		
26.	BenchmarkTigerTonic_GithubStatic	3000000	610 ns/op	
	48 B/op	1 allocs/op		
27.	BenchmarkTraffic_GithubStatic	20000	62973 ns/op	
	18904 B/op	148 allocs/op		
28.	BenchmarkVulcan_GithubStatic	1000000	1447 ns/op	
	98 B/op	3 allocs/op		
29.	BenchmarkAce_GithubParam	2000000	686 ns/op	
	96 B/op	1 allocs/op		
30.	BenchmarkBear_GithubParam	1000000	2155 ns/op	
	496 B/op	5 allocs/op		
31.	BenchmarkBeego_GithubParam	1000000	2713 ns/op	

	368 B/op	4 allocs/op		
32.	BenchmarkBone_GithubParam	100000	15088 ns/op	
	1760 B/op	18 allocs/op		
33.	BenchmarkDenco_GithubParam	2000000	629 ns/op	
	128 B/op	1 allocs/op		
34.	BenchmarkEcho_GithubParam	2000000	653 ns/op	
	32 B/op	1 allocs/op		
35.	BenchmarkGin_GithubParam	5000000	255 ns/op	
	0 B/op	0 allocs/op		
36.	BenchmarkGocraftWeb_GithubParam	1000000	3145 ns/op	
	712 B/op	9 allocs/op		
37.	BenchmarkGoji_GithubParam	1000000	1916 ns/op	
	336 B/op	2 allocs/op		
38.	BenchmarkGojiv2_GithubParam	1000000	3975 ns/op	
	1024 B/op	10 allocs/op		
39.	BenchmarkGoJsonRest_GithubParam	300000	4134 ns/op	
	713 B/op	14 allocs/op		
40.	BenchmarkGoRestful_GithubParam	50000	30782 ns/op	
	2360 B/op	21 allocs/op		
41.	BenchmarkGorillaMux_GithubParam	100000	17148 ns/op	
	1088 B/op	11 allocs/op		
42.	BenchmarkHttpRouter_GithubParam	3000000	523 ns/op	
	96 B/op	1 allocs/op		
43.	BenchmarkHttpTreeMux_GithubParam	1000000	1671 ns/op	
	384 B/op	4 allocs/op		
44.	BenchmarkKocha_GithubParam	1000000	1021 ns/op	
	128 B/op	5 allocs/op		
45.	BenchmarkLARS_GithubParam	5000000	283 ns/op	
	0 B/op	0 allocs/op		
46.	BenchmarkMacaron_GithubParam	500000	4270 ns/op	
	1056 B/op	10 allocs/op		
47.	BenchmarkMartini_GithubParam	100000	21728 ns/op	
	1152 B/op	11 allocs/op		
48.	BenchmarkPat_GithubParam	200000	11208 ns/op	
	2464 B/op	48 allocs/op		
49.	BenchmarkPossum_GithubParam	1000000	2334 ns/op	
	560 B/op	6 allocs/op		
50.	BenchmarkR2router_GithubParam	1000000	1487 ns/op	
	432 B/op	5 allocs/op		
51.	BenchmarkRivet_GithubParam	2000000	782 ns/op	
	96 B/op	1 allocs/op		
52.	BenchmarkTango_GithubParam	1000000	2653 ns/op	

	344 B/op	8 allocs/op		
53.	BenchmarkTigerTonic_GithubParam	300000	14073 ns/op	
	1440 B/op	24 allocs/op		
54.	BenchmarkTraffic_GithubParam	50000	29164 ns/op	
	5992 B/op	52 allocs/op		
55.	BenchmarkVulcan_GithubParam	1000000	2529 ns/op	
	98 B/op	3 allocs/op		
56.	BenchmarkAce_GithubAll	10000	134059 ns/op	
	13792 B/op	167 allocs/op		
57.	BenchmarkBear_GithubAll	5000	534445 ns/op	
	86448 B/op	943 allocs/op		
58.	BenchmarkBeego_GithubAll	3000	592444 ns/op	
	74705 B/op	812 allocs/op		
59.	BenchmarkBone_GithubAll	200	6957308 ns/op	
	698784 B/op	8453 allocs/op		
60.	BenchmarkDenco_GithubAll	10000	158819 ns/op	
	20224 B/op	167 allocs/op		
61.	BenchmarkEcho_GithubAll	10000	154700 ns/op	
	6496 B/op	203 allocs/op		
62.	BenchmarkGin_GithubAll	30000	48375 ns/op	
	0 B/op	0 allocs/op		
63.	BenchmarkGocraftWeb_GithubAll	3000	570806 ns/op	
	131656 B/op	1686 allocs/op		
64.	BenchmarkGoji_GithubAll	2000	818034 ns/op	
	56112 B/op	334 allocs/op		
65.	BenchmarkGojiv2_GithubAll	2000	1213973 ns/op	
	274768 B/op	3712 allocs/op		
66.	BenchmarkGoJsonRest_GithubAll	2000	785796 ns/op	
	134371 B/op	2737 allocs/op		
67.	BenchmarkGoRestful_GithubAll	300	5238188 ns/op	
	689672 B/op	4519 allocs/op		
68.	BenchmarkGorillaMux_GithubAll	100	10257726 ns/op	
	211840 B/op	2272 allocs/op		
69.	BenchmarkHttpRouter_GithubAll	20000	105414 ns/op	
	13792 B/op	167 allocs/op		
70.	BenchmarkHttpTreeMux_GithubAll	10000	319934 ns/op	
	65856 B/op	671 allocs/op		
71.	BenchmarkKocha_GithubAll	10000	209442 ns/op	
	23304 B/op	843 allocs/op		
72.	BenchmarkLARS_GithubAll	20000	62565 ns/op	
	0 B/op	0 allocs/op		
73.	BenchmarkMacaron_GithubAll	2000	1161270 ns/op	

	204194 B/op	2000 allocs/op	
74.	BenchmarkMartini_GithubAll	200	9991713 ns/op
	226549 B/op	2325 allocs/op	
75.	BenchmarkPat_GithubAll	200	5590793 ns/op
	1499568 B/op	27435 allocs/op	
76.	BenchmarkPossum_GithubAll	10000	319768 ns/op
	84448 B/op	609 allocs/op	
77.	BenchmarkR2router_GithubAll	10000	305134 ns/op
	77328 B/op	979 allocs/op	
78.	BenchmarkRivet_GithubAll	10000	132134 ns/op
	16272 B/op	167 allocs/op	
79.	BenchmarkTango_GithubAll	3000	552754 ns/op
	63826 B/op	1618 allocs/op	
80.	BenchmarkTigerTonic_GithubAll	1000	1439483 ns/op
	239104 B/op	5374 allocs/op	
81.	BenchmarkTraffic_GithubAll	100	11383067 ns/op
	2659329 B/op	21848 allocs/op	
82.	BenchmarkVulcan_GithubAll	5000	394253 ns/op
	19894 B/op	609 allocs/op	

Google+

1.	BenchmarkGin_GPlusStatic	10000000	183 ns/op
	0 B/op	0 allocs/op	
2.			
3.	BenchmarkAce_GPlusStatic	5000000	276 ns/op
	0 B/op	0 allocs/op	
4.	BenchmarkBear_GPlusStatic	2000000	652 ns/op
	104 B/op	3 allocs/op	
5.	BenchmarkBeego_GPlusStatic	1000000	2239 ns/op
	368 B/op	4 allocs/op	
6.	BenchmarkBone_GPlusStatic	5000000	380 ns/op
	32 B/op	1 allocs/op	
7.	BenchmarkDenco_GPlusStatic	30000000	45.8 ns/op
	0 B/op	0 allocs/op	
8.	BenchmarkEcho_GPlusStatic	5000000	338 ns/op
	32 B/op	1 allocs/op	
9.	BenchmarkGocraftWeb_GPlusStatic	1000000	1158 ns/op
	280 B/op	5 allocs/op	
10.	BenchmarkGoji_GPlusStatic	5000000	331 ns/op
	0 B/op	0 allocs/op	

11.	BenchmarkGojiv2_GPlusStatic	1000000	2106 ns/op
	928 B/op	7 allocs/op	
12.	BenchmarkGoJsonRest_GPlusStatic	1000000	1626 ns/op
	329 B/op	11 allocs/op	
13.	BenchmarkGoRestful_GPlusStatic	300000	7598 ns/op
	1976 B/op	20 allocs/op	
14.	BenchmarkGorillaMux_GPlusStatic	1000000	2629 ns/op
	736 B/op	10 allocs/op	
15.	BenchmarkHttpRouter_GPlusStatic	30000000	52.5 ns/op
	0 B/op	0 allocs/op	
16.	BenchmarkHttpTreeMux_GPlusStatic	20000000	85.8 ns/op
	0 B/op	0 allocs/op	
17.	BenchmarkKocha_GPlusStatic	20000000	89.2 ns/op
	0 B/op	0 allocs/op	
18.	BenchmarkLARS_GPlusStatic	10000000	162 ns/op
	0 B/op	0 allocs/op	
19.	BenchmarkMacaron_GPlusStatic	500000	3479 ns/op
	768 B/op	9 allocs/op	
20.	BenchmarkMartini_GPlusStatic	200000	9092 ns/op
	768 B/op	9 allocs/op	
21.	BenchmarkPat_GPlusStatic	3000000	493 ns/op
	96 B/op	2 allocs/op	
22.	BenchmarkPossum_GPlusStatic	1000000	1467 ns/op
	416 B/op	3 allocs/op	
23.	BenchmarkR2router_GPlusStatic	2000000	788 ns/op
	144 B/op	4 allocs/op	
24.	BenchmarkRivet_GPlusStatic	20000000	114 ns/op
	0 B/op	0 allocs/op	
25.	BenchmarkTango_GPlusStatic	1000000	1534 ns/op
	200 B/op	8 allocs/op	
26.	BenchmarkTigerTonic_GPlusStatic	5000000	282 ns/op
	32 B/op	1 allocs/op	
27.	BenchmarkTraffic_GPlusStatic	500000	3798 ns/op
	1192 B/op	15 allocs/op	
28.	BenchmarkVulcan_GPlusStatic	2000000	1125 ns/op
	98 B/op	3 allocs/op	
29.	BenchmarkAce_GPlusParam	3000000	528 ns/op
	64 B/op	1 allocs/op	
30.	BenchmarkBear_GPlusParam	1000000	1570 ns/op
	480 B/op	5 allocs/op	
31.	BenchmarkBeego_GPlusParam	1000000	2369 ns/op
	368 B/op	4 allocs/op	

32.	BenchmarkBone_GPlusParam	1000000	2028 ns/op
	688 B/op	5 allocs/op	
33.	BenchmarkDenco_GPlusParam	5000000	385 ns/op
	64 B/op	1 allocs/op	
34.	BenchmarkEcho_GPlusParam	3000000	441 ns/op
	32 B/op	1 allocs/op	
35.	BenchmarkGin_GPlusParam	10000000	174 ns/op
	0 B/op	0 allocs/op	
36.	BenchmarkGocraftWeb_GPlusParam	1000000	2033 ns/op
	648 B/op	8 allocs/op	
37.	BenchmarkGoji_GPlusParam	1000000	1399 ns/op
	336 B/op	2 allocs/op	
38.	BenchmarkGojiv2_GPlusParam	1000000	2641 ns/op
	944 B/op	8 allocs/op	
39.	BenchmarkGoJsonRest_GPlusParam	1000000	2824 ns/op
	649 B/op	13 allocs/op	
40.	BenchmarkGoRestful_GPlusParam	200000	8875 ns/op
	2296 B/op	21 allocs/op	
41.	BenchmarkGorillaMux_GPlusParam	200000	6291 ns/op
	1056 B/op	11 allocs/op	
42.	BenchmarkHttpRouter_GPlusParam	5000000	316 ns/op
	64 B/op	1 allocs/op	
43.	BenchmarkHttpTreeMux_GPlusParam	1000000	1129 ns/op
	352 B/op	3 allocs/op	
44.	BenchmarkKocha_GPlusParam	3000000	538 ns/op
	56 B/op	3 allocs/op	
45.	BenchmarkLARS_GPlusParam	10000000	198 ns/op
	0 B/op	0 allocs/op	
46.	BenchmarkMacaron_GPlusParam	500000	3554 ns/op
	1056 B/op	10 allocs/op	
47.	BenchmarkMartini_GPlusParam	200000	9831 ns/op
	1072 B/op	10 allocs/op	
48.	BenchmarkPat_GPlusParam	1000000	2706 ns/op
	688 B/op	12 allocs/op	
49.	BenchmarkPossum_GPlusParam	1000000	2297 ns/op
	560 B/op	6 allocs/op	
50.	BenchmarkR2router_GPlusParam	1000000	1318 ns/op
	432 B/op	5 allocs/op	
51.	BenchmarkRivet_GPlusParam	5000000	399 ns/op
	48 B/op	1 allocs/op	
52.	BenchmarkTango_GPlusParam	1000000	2070 ns/op
	264 B/op	8 allocs/op	

53.	BenchmarkTigerTonic_GPlusParam	500000	4853 ns/op
	1056 B/op	17 allocs/op	
54.	BenchmarkTraffic_GPlusParam	200000	8278 ns/op
	1976 B/op	21 allocs/op	
55.	BenchmarkVulcan_GPlusParam	1000000	1243 ns/op
	98 B/op	3 allocs/op	
56.	BenchmarkAce_GPlus2Params	3000000	549 ns/op
	64 B/op	1 allocs/op	
57.	BenchmarkBear_GPlus2Params	1000000	2112 ns/op
	496 B/op	5 allocs/op	
58.	BenchmarkBeego_GPlus2Params	500000	2750 ns/op
	368 B/op	4 allocs/op	
59.	BenchmarkBone_GPlus2Params	300000	7032 ns/op
	1040 B/op	9 allocs/op	
60.	BenchmarkDenco_GPlus2Params	3000000	502 ns/op
	64 B/op	1 allocs/op	
61.	BenchmarkEcho_GPlus2Params	3000000	641 ns/op
	32 B/op	1 allocs/op	
62.	BenchmarkGin_GPlus2Params	5000000	250 ns/op
	0 B/op	0 allocs/op	
63.	BenchmarkGocraftWeb_GPlus2Params	1000000	2681 ns/op
	712 B/op	9 allocs/op	
64.	BenchmarkGoji_GPlus2Params	1000000	1926 ns/op
	336 B/op	2 allocs/op	
65.	BenchmarkGojiv2_GPlus2Params	500000	3996 ns/op
	1024 B/op	11 allocs/op	
66.	BenchmarkGoJsonRest_GPlus2Params	500000	3886 ns/op
	713 B/op	14 allocs/op	
67.	BenchmarkGoRestful_GPlus2Params	200000	10376 ns/op
	2360 B/op	21 allocs/op	
68.	BenchmarkGorillaMux_GPlus2Params	100000	14162 ns/op
	1088 B/op	11 allocs/op	
69.	BenchmarkHttpRouter_GPlus2Params	5000000	336 ns/op
	64 B/op	1 allocs/op	
70.	BenchmarkHttpTreeMux_GPlus2Params	1000000	1523 ns/op
	384 B/op	4 allocs/op	
71.	BenchmarkKocha_GPlus2Params	2000000	970 ns/op
	128 B/op	5 allocs/op	
72.	BenchmarkLARS_GPlus2Params	5000000	238 ns/op
	0 B/op	0 allocs/op	
73.	BenchmarkMacaron_GPlus2Params	500000	4016 ns/op
	1056 B/op	10 allocs/op	

74.	BenchmarkMartini_GPlus2Params	100000	21253 ns/op
	1200 B/op	13 allocs/op	
75.	BenchmarkPat_GPlus2Params	200000	8632 ns/op
	2256 B/op	34 allocs/op	
76.	BenchmarkPossum_GPlus2Params	1000000	2171 ns/op
	560 B/op	6 allocs/op	
77.	BenchmarkR2router_GPlus2Params	1000000	1340 ns/op
	432 B/op	5 allocs/op	
78.	BenchmarkRivet_GPlus2Params	3000000	557 ns/op
	96 B/op	1 allocs/op	
79.	BenchmarkTango_GPlus2Params	1000000	2186 ns/op
	344 B/op	8 allocs/op	
80.	BenchmarkTigerTonic_GPlus2Params	200000	9060 ns/op
	1488 B/op	24 allocs/op	
81.	BenchmarkTraffic_GPlus2Params	100000	20324 ns/op
	3272 B/op	31 allocs/op	
82.	BenchmarkVulcan_GPlus2Params	1000000	2039 ns/op
	98 B/op	3 allocs/op	
83.	BenchmarkAce_GPlusAll	300000	6603 ns/op
	640 B/op	11 allocs/op	
84.	BenchmarkBear_GPlusAll	100000	22363 ns/op
	5488 B/op	61 allocs/op	
85.	BenchmarkBeego_GPlusAll	50000	38757 ns/op
	4784 B/op	52 allocs/op	
86.	BenchmarkBone_GPlusAll	20000	54916 ns/op
	10336 B/op	98 allocs/op	
87.	BenchmarkDenco_GPlusAll	300000	4959 ns/op
	672 B/op	11 allocs/op	
88.	BenchmarkEcho_GPlusAll	200000	6558 ns/op
	416 B/op	13 allocs/op	
89.	BenchmarkGin_GPlusAll	500000	2757 ns/op
	0 B/op	0 allocs/op	
90.	BenchmarkGocraftWeb_GPlusAll	50000	34615 ns/op
	8040 B/op	103 allocs/op	
91.	BenchmarkGoji_GPlusAll	100000	16002 ns/op
	3696 B/op	22 allocs/op	
92.	BenchmarkGojiv2_GPlusAll	50000	35060 ns/op
	12624 B/op	115 allocs/op	
93.	BenchmarkGoJsonRest_GPlusAll	50000	41479 ns/op
	8117 B/op	170 allocs/op	
94.	BenchmarkGoRestful_GPlusAll	10000	131653 ns/op
	32024 B/op	275 allocs/op	

95.	BenchmarkGorillaMux_GPlusAll	10000	101380 ns/op
	13296 B/op	142 allocs/op	
96.	BenchmarkHttpRouter_GPlusAll	500000	3711 ns/op
	640 B/op	11 allocs/op	
97.	BenchmarkHttpTreeMux_GPlusAll	100000	14438 ns/op
	4032 B/op	38 allocs/op	
98.	BenchmarkKocha_GPlusAll	200000	8039 ns/op
	976 B/op	43 allocs/op	
99.	BenchmarkLARS_GPlusAll	500000	2630 ns/op
	0 B/op	0 allocs/op	
100.	BenchmarkMacaron_GPlusAll	30000	51123 ns/op
	13152 B/op	128 allocs/op	
101.	BenchmarkMartini_GPlusAll	10000	176157 ns/op
	14016 B/op	145 allocs/op	
102.	BenchmarkPat_GPlusAll	20000	69911 ns/op
	16576 B/op	298 allocs/op	
103.	BenchmarkPossum_GPlusAll	100000	20716 ns/op
	5408 B/op	39 allocs/op	
104.	BenchmarkR2router_GPlusAll	100000	17463 ns/op
	5040 B/op	63 allocs/op	
105.	BenchmarkRivet_GPlusAll	300000	5142 ns/op
	768 B/op	11 allocs/op	
106.	BenchmarkTango_GPlusAll	50000	27321 ns/op
	3656 B/op	104 allocs/op	
107.	BenchmarkTigerTonic_GPlusAll	20000	77597 ns/op
	14512 B/op	288 allocs/op	
108.	BenchmarkTraffic_GPlusAll	10000	151406 ns/op
	37360 B/op	392 allocs/op	
109.	BenchmarkVulcan_GPlusAll	100000	18555 ns/op
	1274 B/op	39 allocs/op	

Parse.com

1.	BenchmarkGin_ParseStatic	10000000	133 ns/op
	0 B/op	0 allocs/op	
2.			
3.	BenchmarkAce_ParseStatic	5000000	241 ns/op
	0 B/op	0 allocs/op	
4.	BenchmarkBear_ParseStatic	2000000	728 ns/op
	120 B/op	3 allocs/op	
5.	BenchmarkBeego_ParseStatic	1000000	2623 ns/op

	368 B/op	4 allocs/op		
6.	BenchmarkBone_ParseStatic	1000000	1285 ns/op	
	144 B/op	3 allocs/op		
7.	BenchmarkDenco_ParseStatic	30000000	57.8 ns/op	
	0 B/op	0 allocs/op		
8.	BenchmarkEcho_ParseStatic	5000000	342 ns/op	
	32 B/op	1 allocs/op		
9.	BenchmarkGocraftWeb_ParseStatic	1000000	1478 ns/op	
	296 B/op	5 allocs/op		
10.	BenchmarkGoji_ParseStatic	3000000	415 ns/op	
	0 B/op	0 allocs/op		
11.	BenchmarkGojiv2_ParseStatic	1000000	2087 ns/op	
	928 B/op	7 allocs/op		
12.	BenchmarkGoJsonRest_ParseStatic	1000000	1712 ns/op	
	329 B/op	11 allocs/op		
13.	BenchmarkGoRestful_ParseStatic	200000	11072 ns/op	
	3224 B/op	22 allocs/op		
14.	BenchmarkGorillaMux_ParseStatic	500000	4129 ns/op	
	752 B/op	11 allocs/op		
15.	BenchmarkHttpRouter_ParseStatic	30000000	52.4 ns/op	
	0 B/op	0 allocs/op		
16.	BenchmarkHttpTreeMux_ParseStatic	20000000	109 ns/op	
	0 B/op	0 allocs/op		
17.	BenchmarkKocha_ParseStatic	20000000	81.8 ns/op	
	0 B/op	0 allocs/op		
18.	BenchmarkLARS_ParseStatic	10000000	150 ns/op	
	0 B/op	0 allocs/op		
19.	BenchmarkMacaron_ParseStatic	1000000	3288 ns/op	
	768 B/op	9 allocs/op		
20.	BenchmarkMartini_ParseStatic	200000	9110 ns/op	
	768 B/op	9 allocs/op		
21.	BenchmarkPat_ParseStatic	1000000	1135 ns/op	
	240 B/op	5 allocs/op		
22.	BenchmarkPossum_ParseStatic	1000000	1557 ns/op	
	416 B/op	3 allocs/op		
23.	BenchmarkR2router_ParseStatic	2000000	730 ns/op	
	144 B/op	4 allocs/op		
24.	BenchmarkRivet_ParseStatic	10000000	121 ns/op	
	0 B/op	0 allocs/op		
25.	BenchmarkTango_ParseStatic	1000000	1688 ns/op	
	248 B/op	8 allocs/op		
26.	BenchmarkTigerTonic_ParseStatic	3000000	427 ns/op	

	48 B/op	1 allocs/op		
27.	BenchmarkTraffic_ParseStatic	500000	5962 ns/op	
	1816 B/op	20 allocs/op		
28.	BenchmarkVulcan_ParseStatic	2000000	969 ns/op	
	98 B/op	3 allocs/op		
29.	BenchmarkAce_ParseParam	3000000	497 ns/op	
	64 B/op	1 allocs/op		
30.	BenchmarkBear_ParseParam	1000000	1473 ns/op	
	467 B/op	5 allocs/op		
31.	BenchmarkBeego_ParseParam	1000000	2384 ns/op	
	368 B/op	4 allocs/op		
32.	BenchmarkBone_ParseParam	1000000	2513 ns/op	
	768 B/op	6 allocs/op		
33.	BenchmarkDenco_ParseParam	5000000	364 ns/op	
	64 B/op	1 allocs/op		
34.	BenchmarkEcho_ParseParam	5000000	418 ns/op	
	32 B/op	1 allocs/op		
35.	BenchmarkGin_ParseParam	10000000	163 ns/op	
	0 B/op	0 allocs/op		
36.	BenchmarkGocraftWeb_ParseParam	1000000	2361 ns/op	
	664 B/op	8 allocs/op		
37.	BenchmarkGoji_ParseParam	1000000	1590 ns/op	
	336 B/op	2 allocs/op		
38.	BenchmarkGojiv2_ParseParam	1000000	2851 ns/op	
	976 B/op	9 allocs/op		
39.	BenchmarkGoJsonRest_ParseParam	1000000	2965 ns/op	
	649 B/op	13 allocs/op		
40.	BenchmarkGoRestful_ParseParam	200000	12207 ns/op	
	3544 B/op	23 allocs/op		
41.	BenchmarkGorillaMux_ParseParam	500000	5187 ns/op	
	1088 B/op	12 allocs/op		
42.	BenchmarkHttpRouter_ParseParam	5000000	275 ns/op	
	64 B/op	1 allocs/op		
43.	BenchmarkHttpTreeMux_ParseParam	1000000	1108 ns/op	
	352 B/op	3 allocs/op		
44.	BenchmarkKocha_ParseParam	3000000	495 ns/op	
	56 B/op	3 allocs/op		
45.	BenchmarkLARS_ParseParam	10000000	192 ns/op	
	0 B/op	0 allocs/op		
46.	BenchmarkMacaron_ParseParam	500000	4103 ns/op	
	1056 B/op	10 allocs/op		
47.	BenchmarkMartini_ParseParam	200000	9878 ns/op	

	1072 B/op	10 allocs/op		
48.	BenchmarkPat_ParseParam	500000	3657 ns/op	
	1120 B/op	17 allocs/op		
49.	BenchmarkPossum_ParseParam	1000000	2084 ns/op	
	560 B/op	6 allocs/op		
50.	BenchmarkR2router_ParseParam	1000000	1251 ns/op	
	432 B/op	5 allocs/op		
51.	BenchmarkRivet_ParseParam	5000000	335 ns/op	
	48 B/op	1 allocs/op		
52.	BenchmarkTango_ParseParam	1000000	1854 ns/op	
	280 B/op	8 allocs/op		
53.	BenchmarkTigerTonic_ParseParam	500000	4582 ns/op	
	1008 B/op	17 allocs/op		
54.	BenchmarkTraffic_ParseParam	200000	8125 ns/op	
	2248 B/op	23 allocs/op		
55.	BenchmarkVulcan_ParseParam	1000000	1148 ns/op	
	98 B/op	3 allocs/op		
56.	BenchmarkAce_Parse2Params	3000000	539 ns/op	
	64 B/op	1 allocs/op		
57.	BenchmarkBear_Parse2Params	1000000	1778 ns/op	
	496 B/op	5 allocs/op		
58.	BenchmarkBeego_Parse2Params	1000000	2519 ns/op	
	368 B/op	4 allocs/op		
59.	BenchmarkBone_Parse2Params	1000000	2596 ns/op	
	720 B/op	5 allocs/op		
60.	BenchmarkDenco_Parse2Params	3000000	492 ns/op	
	64 B/op	1 allocs/op		
61.	BenchmarkEcho_Parse2Params	3000000	484 ns/op	
	32 B/op	1 allocs/op		
62.	BenchmarkGin_Parse2Params	10000000	193 ns/op	
	0 B/op	0 allocs/op		
63.	BenchmarkGocraftWeb_Parse2Params	1000000	2575 ns/op	
	712 B/op	9 allocs/op		
64.	BenchmarkGoji_Parse2Params	1000000	1373 ns/op	
	336 B/op	2 allocs/op		
65.	BenchmarkGojiv2_Parse2Params	500000	2416 ns/op	
	960 B/op	8 allocs/op		
66.	BenchmarkGoJsonRest_Parse2Params	300000	3452 ns/op	
	713 B/op	14 allocs/op		
67.	BenchmarkGoRestful_Parse2Params	100000	17719 ns/op	
	6008 B/op	25 allocs/op		
68.	BenchmarkGorillaMux_Parse2Params	300000	5102 ns/op	

	1088 B/op	11 allocs/op		
69.	BenchmarkHttpRouter_Parse2Params	5000000	303 ns/op	
	64 B/op	1 allocs/op		
70.	BenchmarkHttpTreeMux_Parse2Params	1000000	1372 ns/op	
	384 B/op	4 allocs/op		
71.	BenchmarkKocha_Parse2Params	2000000	874 ns/op	
	128 B/op	5 allocs/op		
72.	BenchmarkLARS_Parse2Params	10000000	192 ns/op	
	0 B/op	0 allocs/op		
73.	BenchmarkMacaron_Parse2Params	500000	3871 ns/op	
	1056 B/op	10 allocs/op		
74.	BenchmarkMartini_Parse2Params	200000	9954 ns/op	
	1152 B/op	11 allocs/op		
75.	BenchmarkPat_Parse2Params	500000	4194 ns/op	
	832 B/op	17 allocs/op		
76.	BenchmarkPossum_Parse2Params	1000000	2121 ns/op	
	560 B/op	6 allocs/op		
77.	BenchmarkR2router_Parse2Params	1000000	1415 ns/op	
	432 B/op	5 allocs/op		
78.	BenchmarkRivet_Parse2Params	3000000	457 ns/op	
	96 B/op	1 allocs/op		
79.	BenchmarkTango_Parse2Params	1000000	1914 ns/op	
	312 B/op	8 allocs/op		
80.	BenchmarkTigerTonic_Parse2Params	300000	6895 ns/op	
	1408 B/op	24 allocs/op		
81.	BenchmarkTraffic_Parse2Params	200000	8317 ns/op	
	2040 B/op	22 allocs/op		
82.	BenchmarkVulcan_Parse2Params	1000000	1274 ns/op	
	98 B/op	3 allocs/op		
83.	BenchmarkAce_ParseAll	200000	10401 ns/op	
	640 B/op	16 allocs/op		
84.	BenchmarkBear_ParseAll	50000	37743 ns/op	
	8928 B/op	110 allocs/op		
85.	BenchmarkBeego_ParseAll	20000	63193 ns/op	
	9568 B/op	104 allocs/op		
86.	BenchmarkBone_ParseAll	20000	61767 ns/op	
	14160 B/op	131 allocs/op		
87.	BenchmarkDenco_ParseAll	300000	7036 ns/op	
	928 B/op	16 allocs/op		
88.	BenchmarkEcho_ParseAll	200000	11824 ns/op	
	832 B/op	26 allocs/op		
89.	BenchmarkGin_ParseAll	300000	4199 ns/op	

	0 B/op	0 allocs/op		
90.	BenchmarkGocraftWeb_ParseAll	30000	51758 ns/op	
	13728 B/op	181 allocs/op		
91.	BenchmarkGoji_ParseAll	50000	29614 ns/op	
	5376 B/op	32 allocs/op		
92.	BenchmarkGojiv2_ParseAll	20000	68676 ns/op	
	24464 B/op	199 allocs/op		
93.	BenchmarkGoJsonRest_ParseAll	20000	76135 ns/op	
	13866 B/op	321 allocs/op		
94.	BenchmarkGoRestful_ParseAll	5000	389487 ns/op	
	110928 B/op	600 allocs/op		
95.	BenchmarkGorillaMux_ParseAll	10000	221250 ns/op	
	24864 B/op	292 allocs/op		
96.	BenchmarkHttpRouter_ParseAll	200000	6444 ns/op	
	640 B/op	16 allocs/op		
97.	BenchmarkHttpTreeMux_ParseAll	50000	30702 ns/op	
	5728 B/op	51 allocs/op		
98.	BenchmarkKocha_ParseAll	200000	13712 ns/op	
	1112 B/op	54 allocs/op		
99.	BenchmarkLARS_ParseAll	300000	6925 ns/op	
	0 B/op	0 allocs/op		
100.	BenchmarkMacaron_ParseAll	20000	96278 ns/op	
	24576 B/op	250 allocs/op		
101.	BenchmarkMartini_ParseAll	5000	271352 ns/op	
	25072 B/op	253 allocs/op		
102.	BenchmarkPat_ParseAll	20000	74941 ns/op	
	17264 B/op	343 allocs/op		
103.	BenchmarkPossum_ParseAll	50000	39947 ns/op	
	10816 B/op	78 allocs/op		
104.	BenchmarkR2router_ParseAll	50000	42479 ns/op	
	8352 B/op	120 allocs/op		
105.	BenchmarkRivet_ParseAll	200000	7726 ns/op	
	912 B/op	16 allocs/op		
106.	BenchmarkTango_ParseAll	30000	50014 ns/op	
	7168 B/op	208 allocs/op		
107.	BenchmarkTigerTonic_ParseAll	10000	106550 ns/op	
	19728 B/op	379 allocs/op		
108.	BenchmarkTraffic_ParseAll	10000	216037 ns/op	
	57776 B/op	642 allocs/op		
109.	BenchmarkVulcan_ParseAll	50000	34379 ns/op	
	2548 B/op	78 allocs/op		

CHANGELOG

CHANGELOG

Gin 1.2

- [NEW] Switch from godeps to govendor
- [NEW] Add support for Let's Encrypt via gin-gonic/autotls
- [NEW] Improve README examples and add extra at examples folder
- [NEW] Improved support with App Engine
- [NEW] Add custom template delimiters, see #860
- [NEW] Add Template Func Maps, see #962
- [NEW] Add `*context.Handler()`, see #928
- [NEW] Add `*context.GetRawData()`
- [NEW] Add `*context.GetHeader()` (request)
- [NEW] Add `*context.AbortWithStatusJSON()` (JSON content type)
- [NEW] Add `*context.Keys` type cast helpers
- [NEW] Add `*context.ShouldBindWith()`
- [NEW] Add `*context.MustBindWith()`
- [NEW] Add `*engine.SetFuncMap()`
- [DEPRECATE] On next release: `*context.BindWith()`, see #855
- [FIX] Refactor render
- [FIX] Reworked tests
- [FIX] logger now supports cygwin
- [FIX] Use X-Forwarded-For before X-Real-IP
- [FIX] `time.Time` binding (#904)

Gin 1.1.4

- [NEW] Support google appengine for `IsTerminal` func

Gin 1.1.3

- [FIX] Reverted Logger: skip ANSI color commands

Gin 1.1

- [NEW] Implement `QueryArray` and `PostArray` methods
- [NEW] Refactor `GetQuery` and `GetPostForm`

- [NEW] Add contribution guide
- [FIX] Corrected typos in README
- [FIX] Removed additional Iota
- [FIX] Changed imports to gopkg instead of github in README (#733)
- [FIX] Logger: skip ANSI color commands if output is not a tty

Gin 1.0rc2 (...)

- [PERFORMANCE] Fast path for writing Content-Type.
- [PERFORMANCE] Much faster 404 routing
- [PERFORMANCE] Allocation optimizations
- [PERFORMANCE] Faster root tree lookup
- [PERFORMANCE] Zero overhead, String() and JSON() rendering.
- [PERFORMANCE] Faster ClientIP parsing
- [PERFORMANCE] Much faster SSE implementation
- [NEW] Benchmarks suite
- [NEW] Bind validation can be disabled and replaced with custom validators.
- [NEW] More flexible HTML render
- [NEW] Multipart and PostForm bindings
- [NEW] Adds method to return all the registered routes
- [NEW] Context.HandlerName() returns the main handler's name
- [NEW] Adds Error.IsType() helper
- [FIX] Binding multipart form
- [FIX] Integration tests
- [FIX] Crash when binding non struct object in Context.
- [FIX] RunTLS() implementation
- [FIX] Logger() unit tests
- [FIX] Adds SetHTMLTemplate() warning
- [FIX] Context.IsAborted()
- [FIX] More unit tests
- [FIX] JSON, XML, HTML renders accept custom content-types
- [FIX] gin.AbortIndex is unexported
- [FIX] Better approach to avoid directory listing in StaticFS()
- [FIX] Context.ClientIP() always returns the IP with trimmed spaces.
- [FIX] Better warning when running in debug mode.
- [FIX] Google App Engine integration. debugPrint does not use os.Stdout
- [FIX] Fixes integer overflow in error type
- [FIX] Error implements the json.Marshaller interface
- [FIX] MIT license in every file

Gin 1.0rc1 (May 22, 2015)

- [PERFORMANCE] Zero allocation router
- [PERFORMANCE] Faster JSON, XML and text rendering
- [PERFORMANCE] Custom hand optimized `HttpRouter` for Gin
- [PERFORMANCE] Misc code optimizations. Inlining, tail call optimizations
- [NEW] Built-in support for `golang.org/x/net/context`
- [NEW] `Any(path, handler)`. Create a route that matches any path
- [NEW] Refactored rendering pipeline (faster and static typed)
- [NEW] Refactored errors API
- [NEW] `IndentedJSON()` prints pretty JSON
- [NEW] Added `gin.DefaultWriter`
- [NEW] UNIX socket support
- [NEW] `RouterGroup.BasePath` is exposed
- [NEW] JSON validation using `go-validate-yourself` (very powerful options)
- [NEW] Completed suite of unit tests
- [NEW] HTTP streaming with `c.Stream()`
- [NEW] `StaticFile()` creates a router for serving just one file.
- [NEW] `StaticFS()` has an option to disable directory listing.
- [NEW] `StaticFS()` for serving static files through virtual filesystems
- [NEW] Server-Sent Events native support
- [NEW] `WrapF()` and `WrapH()` helpers for wrapping `http.HandlerFunc` and `http.Handler`
- [NEW] Added `LoggerWithWriter()` middleware
- [NEW] Added `RecoveryWithWriter()` middleware
- [NEW] Added `DefaultPostFormValue()`
- [NEW] Added `DefaultFormValue()`
- [NEW] Added `DefaultParamValue()`
- [FIX] `BasicAuth()` when using custom realm
- [FIX] Bug when serving static files in nested routing group
- [FIX] Redirect using built-in `http.Redirect()`
- [FIX] Logger when printing the requested path
- [FIX] Documentation typos
- [FIX] `Context.Engine` renamed to `Context.engine`
- [FIX] Better debugging messages
- [FIX] `ErrorLogger`
- [FIX] Debug HTTP render

- [FIX] Refactored binding and render modules
- [FIX] Refactored Context initialization
- [FIX] Refactored BasicAuth()
- [FIX] NoMethod/NoRoute handlers
- [FIX] Hijacking http
- [FIX] Better support for Google App Engine (using log instead of fmt)

Gin 0.6 (Mar 9, 2015)

- [NEW] Support multipart/form-data
- [NEW] NoMethod handler
- [NEW] Validate sub structures
- [NEW] Support for HTTP Realm Auth
- [FIX] Unsigned integers in binding
- [FIX] Improve color logger

Gin 0.5 (Feb 7, 2015)

- [NEW] Content Negotiation
- [FIX] Solved security bug that allow a client to spoof ip
- [FIX] Fix unexported/ignored fields in binding

Gin 0.4 (Aug 21, 2014)

- [NEW] Development mode
- [NEW] Unit tests
- [NEW] Add Content.Redirect()
- [FIX] Deferring WriteHeader()
- [FIX] Improved documentation for model binding

Gin 0.3 (Jul 18, 2014)

- [PERFORMANCE] Normal log and error log are printed in the same call.
- [PERFORMANCE] Improve performance of NoRouter()
- [PERFORMANCE] Improve context's memory locality, reduce CPU cache faults.
- [NEW] Flexible rendering API
- [NEW] Add Context.File()
- [NEW] Add shortcut RunTLS() for http.ListenAndServeTLS
- [FIX] Rename NotFound404() to NoRoute()
- [FIX] Errors in context are purged

- [FIX] Adds HEAD method in Static file serving
- [FIX] Refactors Static() file serving
- [FIX] Using keyed initialization to fix app-engine integration
- [FIX] Can't unmarshal JSON array, #63
- [FIX] Renaming Context.Req to Context.Request
- [FIX] Check application/x-www-form-urlencoded when parsing form

Gin 0.2b (Jul 08, 2014)

- [PERFORMANCE] Using sync.Pool to allocatio/gc overhead
- [NEW] Travis CI integration
- [NEW] Completely new logger
- [NEW] New API for serving static files. gin.Static()
- [NEW] gin.H() can be serialized into XML
- [NEW] Typed errors. Errors can be typed. Internet/external/custom.
- [NEW] Support for Godeps
- [NEW] Travis/Godocs badges in README
- [NEW] New Bind() and BindWith() methods for parsing request body.
- [NEW] Add Content.Copy()
- [NEW] Add context.LastError()
- [NEW] Add shortcut for OPTIONS HTTP method
- [FIX] Tons of README fixes
- [FIX] Header is written before body
- [FIX] BasicAuth() and changes API a little bit
- [FIX] Recovery() middleware only prints panics
- [FIX] Context.Get() does not panic anymore. Use MustGet() instead.
- [FIX] Multiple http.WriteHeader() in NotFound handlers
- [FIX] Engine.Run() panics if http server can't be setted up
- [FIX] Crash when route path doesn't start with '/'
- [FIX] Do not update header when status code is negative
- [FIX] Setting response headers before calling WriteHeader in context.String()
- [FIX] Add MIT license
- [FIX] Changes behaviour of ErrorLogger() and Logger()

Contributor Covenant Code of Conduct

Contributor Covenant Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at teamgingonic@gmail.com. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the [Contributor Covenant](https://contributor-covenant.org/version/1/4), version 1.4, available at <http://contributor-covenant.org/version/1/4>

Contributing

Contributing

- With issues:
 - Use the search tool before opening a new issue.
 - Please provide source code and commit sha if you found a bug.
 - Review existing issues and provide feedback or react to them.
- With pull requests:
 - Open your pull request against `master`
 - Your pull request should have no more than two commits, if not you should squash them.
 - It should pass all tests in the available continuous integrations systems such as TravisCI.
 - You should add/modify tests to cover your proposed code changes.
 - If your pull request contains a new feature, please document it on the README.

Guide to run Gin under App Engine LOCAL Development Server

Guide to run Gin under App Engine LOCAL Development Server

1. Download, install and setup Go in your computer. (That includes setting your `$GOPATH` .)
2. Download SDK for your platform from [here](https://cloud.google.com/appengine/docs/standard/go/download):
`https://cloud.google.com/appengine/docs/standard/go/download`
3. Download Gin source code using: `$ go get github.com/gin-gonic/gin`
4. Navigate to examples folder: `$ cd $GOPATH/src/github.com/gin-gonic/gin/examples/app-engine/`
5. Run it: `$ dev_appserver.py .` (notice that you have to run this script by Python2)

How to use

Building a single binary containing templates

This is a complete example to create a single binary with the [gin-gonic/gin](#) Web Server with HTML templates.

How to use

Prepare Packages

1. `go get github.com/gin-gonic/gin`
2. `go get github.com/jessevdk/go-assets-builder`

Generate assets.go

1. `go-assets-builder html -o assets.go`

Build the server

1. `go build -o assets-in-binary`

Run

1. `./assets-in-binary`

How to run this example

How to run this example

1. run grpc server

```
1. $ go run grpc/server.go
```

1. run gin server

```
1. $ go run gin/main.go
```

1. use curl command to test it

```
1. $ curl -v 'http://localhost:8052/rest/n/thinkerou'
```

How to generate RSA private key and digital certificate

How to generate RSA private key and digital certificate

1. Install Openssl

Please visit <https://github.com/openssl/openssl> to get pkg and install.

1. Generate RSA private key

```
1. $ mkdir testdata
2. $ openssl genrsa -out ./testdata/server.key 2048
```

1. Generate digital certificate

```
1. $ openssl req -new -x509 -key ./testdata/server.key -out ./testdata/server.pem
   -days 365
```

Struct level validations

Struct level validations

Validations can also be registered at the `struct` level when field level validations

don't make much sense. This can also be used to solve cross-field validation elegantly.

Additionally, it can be combined with tag validations. Struct Level validations run after the structs tag validations.

Example requests

```

1. # Validation errors are generated for struct tags as well as at the struct
   level
2. $ curl -s -X POST http://localhost:8085/user \
3.   -H 'content-type: application/json' \
4.   -d '{}' | jq
5. {
6.   "error": "Key: 'User.Email' Error:Field validation for 'Email' failed on the
   'required' tag\nKey: 'User.FirstName' Error:Field validation for 'FirstName'
   failed on the 'fnameorlname' tag\nKey: 'User.LastName' Error:Field validation
   for 'LastName' failed on the 'fnameorlname' tag",
7.   "message": "User validation failed!"
8. }
9.
10. # Validation fails at the struct level because neither first name nor last name
    are present
11. $ curl -s -X POST http://localhost:8085/user \
12.   -H 'content-type: application/json' \
13.   -d '{"email": "george@vandaley.com"}' | jq
14. {
15.   "error": "Key: 'User.FirstName' Error:Field validation for 'FirstName' failed
   on the 'fnameorlname' tag\nKey: 'User.LastName' Error:Field validation for
   'LastName' failed on the 'fnameorlname' tag",
16.   "message": "User validation failed!"
17. }
18.

```

```
19. # No validation errors when either first name or last name is present
20. $ curl -X POST http://localhost:8085/user \
21.     -H 'content-type: application/json' \
22.     -d '{"fname": "George", "email": "george@vandaley.com"}'
23. {"message": "User validation successful."}
24.
25. $ curl -X POST http://localhost:8085/user \
26.     -H 'content-type: application/json' \
27.     -d '{"lname": "Contanza", "email": "george@vandaley.com"}'
28. {"message": "User validation successful."}
29.
30. $ curl -X POST http://localhost:8085/user \
31.     -H 'content-type: application/json' \
32.     -d '{"fname": "George", "lname": "Costanza", "email":
33.     "george@vandaley.com"}'
33. {"message": "User validation successful."}
```

Useful links

- Validator docs - <https://godoc.org/gopkg.in/go-playground/validator.v8#Validate.RegisterStructValidation>
- Struct level example - https://github.com/go-playground/validator/blob/v8.18.2/examples/struct-level/struct_level.go
- Validator release notes - <https://github.com/go-playground/validator/releases/tag/v8.7>

Gin Default Server

Gin Default Server

This is API experiment for Gin.

```
1. package main
2.
3. import (
4.     "github.com/gin-gonic/gin"
5.     "github.com/gin-gonic/gin/ginS"
6. )
7.
8. func main() {
9.     ginS.GET("/", func(c *gin.Context) { c.String(200, "Hello World") })
10.    ginS.Run()
11. }
```