Innocentre, Hong Kong
16MAY2018

GO PRESENTATION OF AWESOMENESS

Alpha WONG

Lalamove

Backend engineer

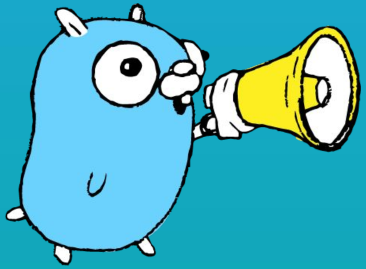
< 1 Year Gopher

Credit

Alan Wong
Benjamin Po
Desmond Ho
Eugene Tokariev
Jack Tang
Jay Pun
Reynaldi Wijaya
Samuel Kwok

Acknowledgement

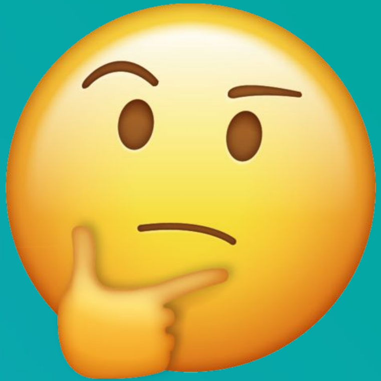
Mikael Knutsson



Today's (glorious) blather.

SECTION ONE

Why not Beego ?



Consideration

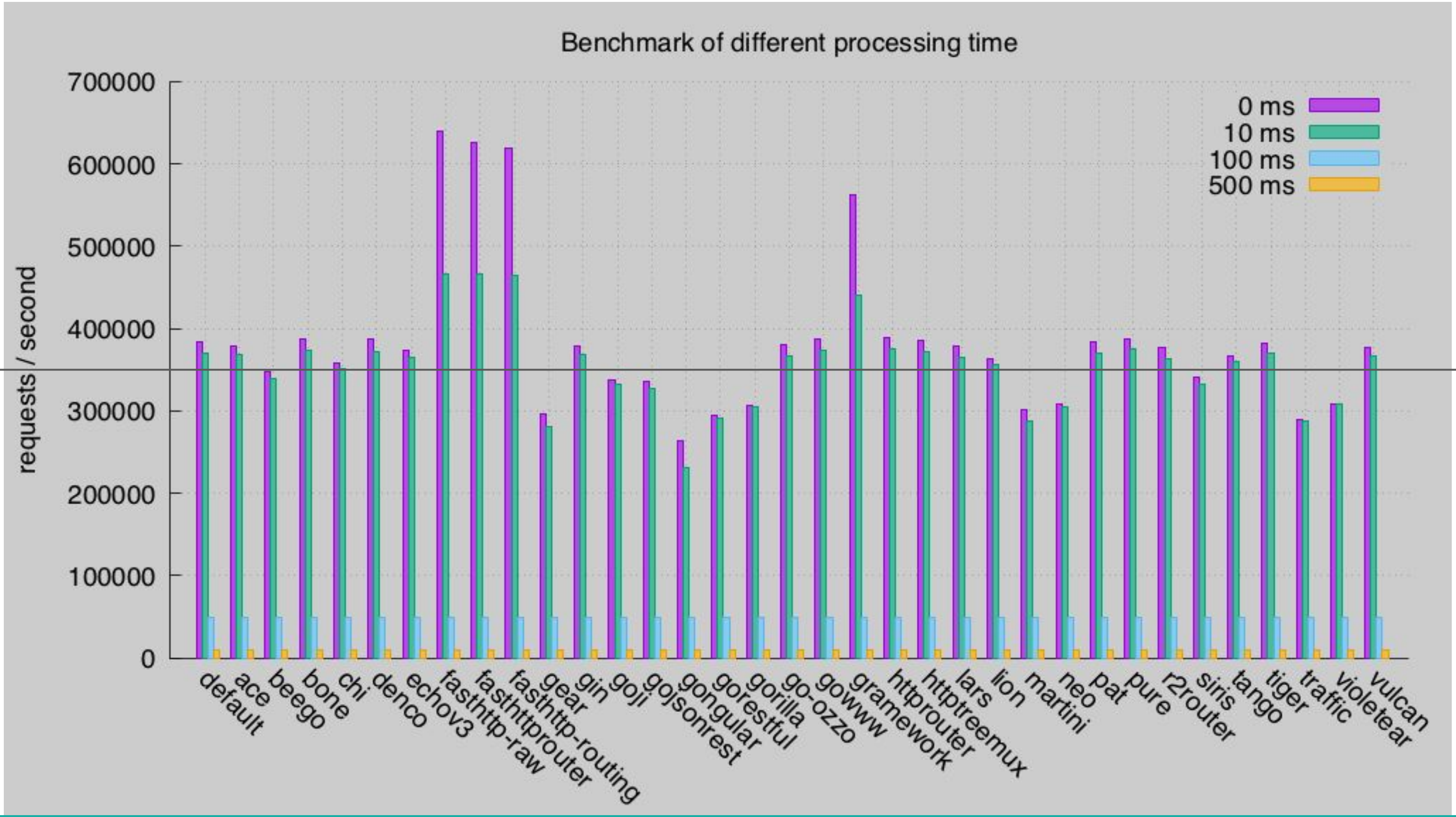
- Performance
- Learning curve
- Maintainability
- Reliable
- Productive
- Community support

Consideration

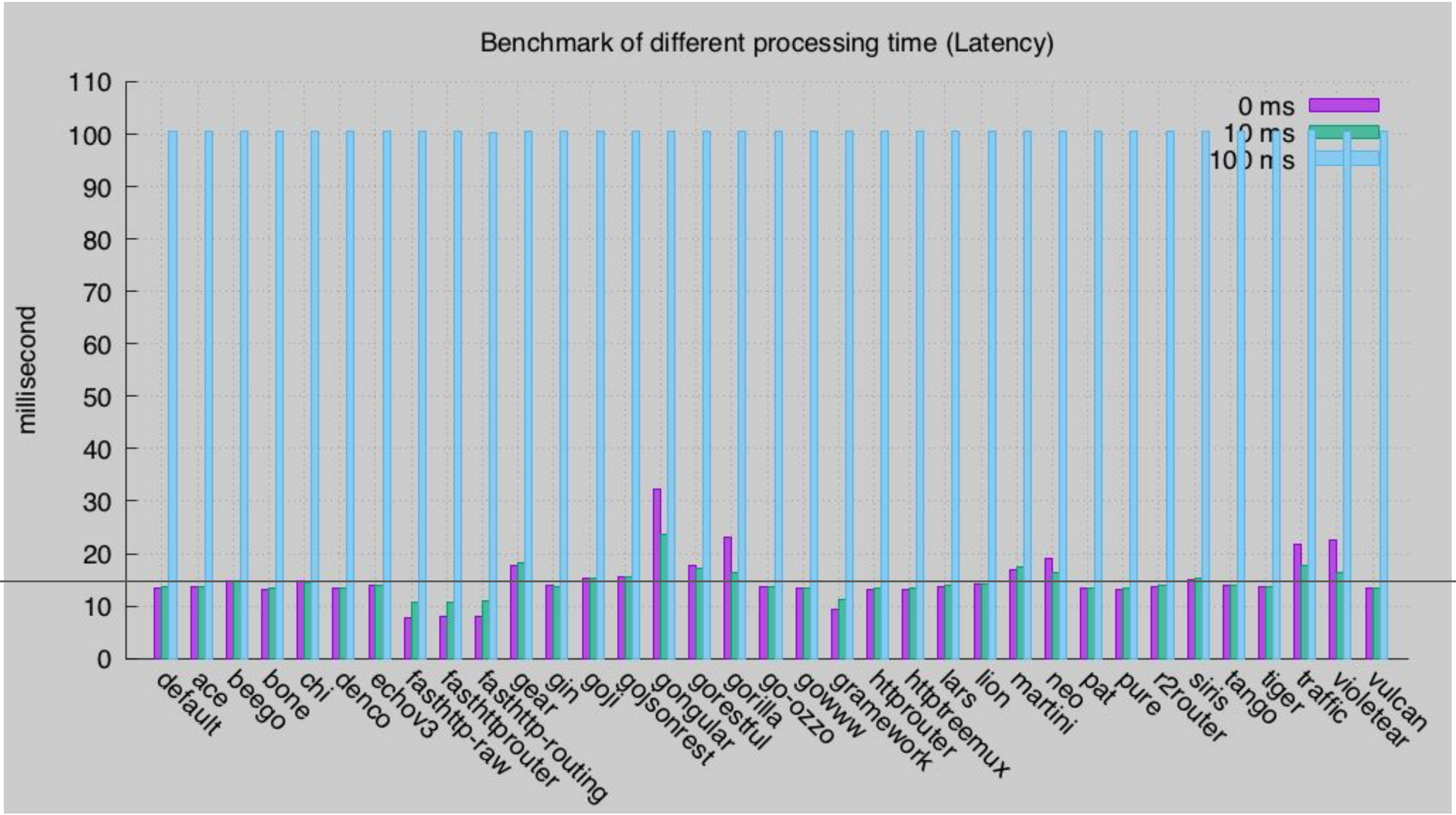
Performance

(5000 concurrency clients)

Performance (higher is better)

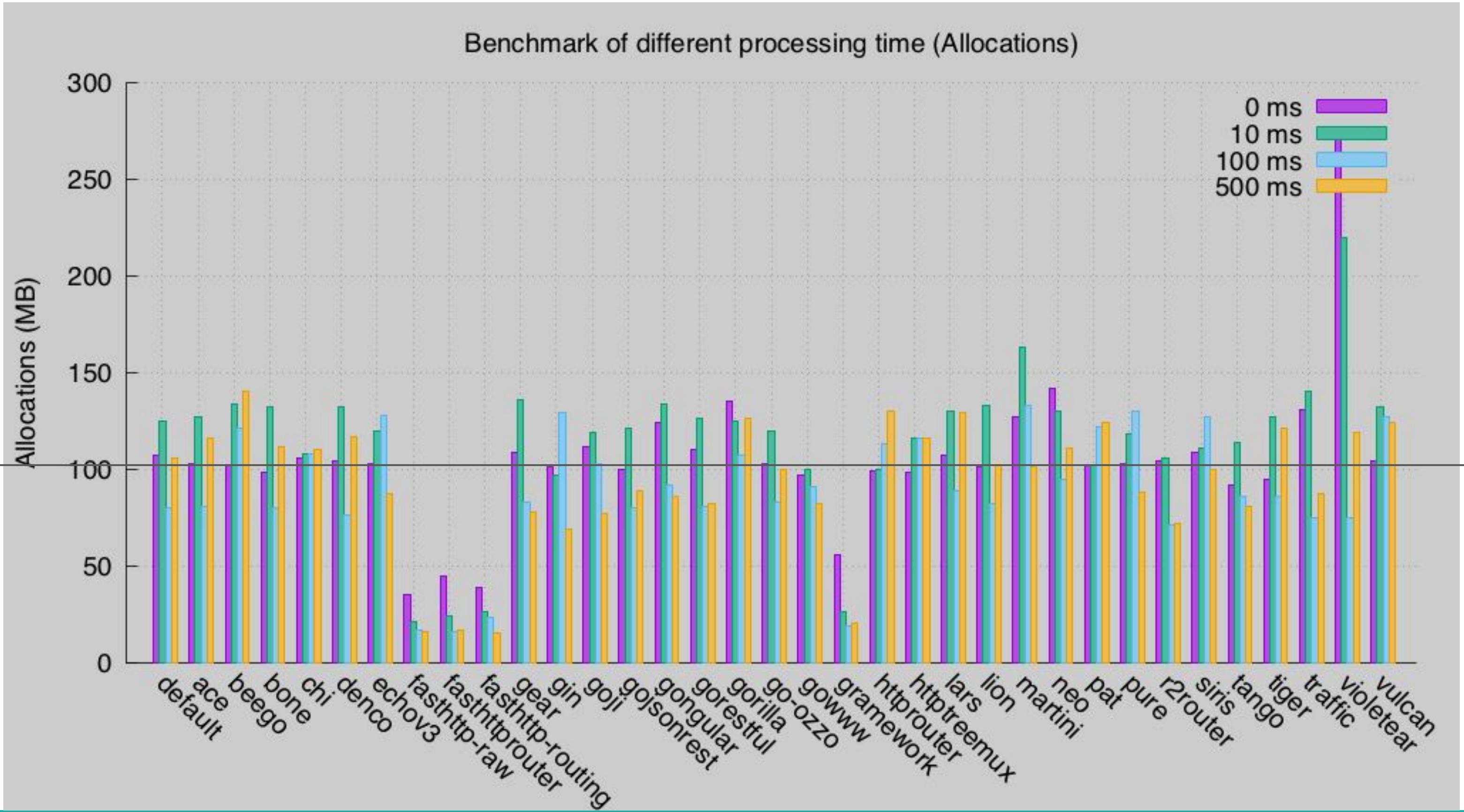


Processing time (smaller is better)



https://github.com/smallnest/go-web-framework-benchmark

Heap allocations (smaller is better)



https://github.com/smallnest/go-web-framework-benchmark

Consideration

Learning curve

Syntax

You can define a config as following

```
appname = beepkg
httpaddr = "127.0.0.1"
httpport = 9090

runmode ="dev"
autorender = false
recoverpanic = false
viewspath = "myview"

[dev]
httpport = 8080
[prod]
httpport = 8088
[test]
httpport = 8888

// config.go
beego.AppConfig.Int("dev::httpport")
```

Each config feature should be easy to understand.

Syntax

You can define a controller as following

```
// Orders
type OrdersController struct {
    beego.Controller
}

// @router /v1/orders/:id [get]
func (this *CMSController) Get() {
    c:=this.Ctx.Input.Param(":id")
    ...
    this.Data["json"] = c
    this.ServeJSON()
}
```

Each route feature should be easy to understand.

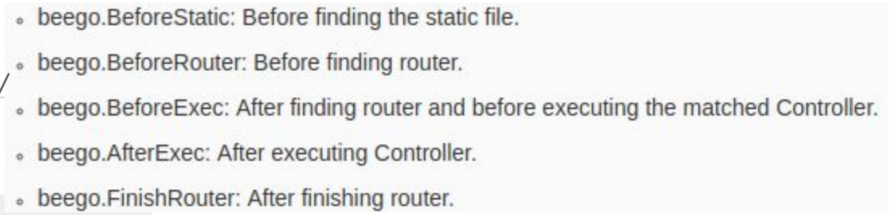
Syntax

You can define a callback style route as following

```
ns := beego.NewNamespace("/v1",
    beego.NSCond(func(ctx *context.Context) bool {
        if ctx.Input.Domain() == "api.beego.me" {
            return true
        }
        return false
    }),
    beego.NSBefore(auth),
    beego.NSGet("/notallowed", func(ctx *context.Context) {
        ctx.Output.Body([]byte("notAllowed"))
    }),
    beego.NSRouter("/version", &AdminController{}, "get:ShowAPIVersion"),
    beego.NSRouter("/changepassword", &UserController{}),
    beego.NSNamespace("/shop",
        beego.NSBefore(sentry),
        beego.NSGet("/:id", func(ctx *context.Context) {
            ctx.Output.Body([]byte("notAllowed"))
        }),
    ),
    beego.NSNamespace("/cms",
        beego.NSInclude(
            &controllers.MainController{},
            &controllers.CMSController{},
            &controllers.BlockController{},
        ),
    ),
)
beego.AddNamespace(ns) //register namespace
```

Each route feature should be easy to understand.

Syntax



- beego.BeforeStatic: Before finding the static file.
- beego.BeforeRouter: Before finding router.
- beego.BeforeExec: After finding router and before executing the matched Controller.
- beego.AfterExec: After executing Controller.
- beego.FinishRouter: After finishing router.

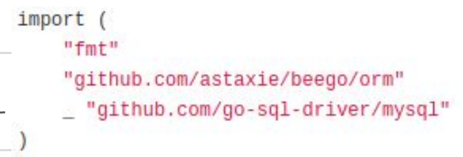
You can define a middleware as following

```
func AddFootPrintMiddleware() {
	var foodPrintMiddleware = func(ctx *context.Context) {
		footPrint := uuid.NewV4().String()

		dump, _ := httputil.DumpRequest(ctx.Request, true)

		log.Printf(
			"footPrint: %s request: %v",
			 footPrint,
			 string(dump),
		)

		nativeCtx := httpContext.WithValue(
			ctx.Request.Context(),
			"Foot print",
			footPrint
		)

		ctx.Request = ctx.Request.WithContext(nativeCtx)
	}

	beego.InsertFilter("/v1/*", beego.BeforeRouter, foodPrintMiddleware)
}
```

Each middleware feature should be easy to understand.

Syntax

You can define a ORM as following



```
import (
	"fmt"
	"github.com/astaxie/beego/orm"
	_ "github.com/go-sql-driver/mysql"
)
```

```
...
func init() {
    orm.RegisterDriver("mysql", orm.DRMySQL)
    orm.RegisterDataBase(
        "Default",
        "mysql",
        "root:root@/orm_test?charset=utf8",
    )
}

func main() {
    o := orm.NewOrm()

    // Using default, you can use other database
    o.Using("default")

    profile := new(Profile)
    profile.Age = 30
}
```

Each ORM feature should be easy to understand.

Syntax

Waiting !!! Beego also support raw SQL

```
func init() {
    orm.RegisterDriver("mysql", orm.DRMySQL)
    orm.RegisterDataBase(
        "Default",
        "mysql",
        "root:root@/orm_test?charset=utf8",
    )
}

func main() {
    o := orm.NewOrm()

    // Using default, you can use other database
    o.Using("default")

    var c customer
    err := o.Raw("SELECT id, name FROM Customer WHERE id = ?", 1).QueryRow(&c)

    res, err := o.Raw("UPDATE Customer SET name = ?", "your").Exec()
    if err == nil {
       num, _ := res.RowsAffected()
       fmt.Println("mysql row affected nums: ", num)
    }
}
```

Each SQL feature should be easy to understand.

Syntax

You can define struct as following

```
type (
	Customer struct {
		Id       int64     `json:"id" orm:"auto"`
		Name     string    `json:"name"`
		Password string    `json:"password"`
		Created  time.Time `orm:"auto_now_add;type(datetime)"`
		Updated  time.Time `orm:"auto_now;type(datetime)"`
	}

	Account struct {
		Id       int64     `json:"id" orm:"auto"`
		Active   bool      `json:"active"`
		Customer *Customer `json:"customer,omitempty" orm:"rel(fk)"`
		Created  time.Time `orm:"auto_now_add;type(datetime)"`
		Updated  time.Time `orm:"auto_now;type(datetime)"`
	}
)
```

Each struct feature should be easy to understand.

Syntax

You can define a test case for handler as following

```
func TestGetTokenSuccess(t *testing.T) {
	...
	r, _ := http.NewRequest("POST", "/v1/route", strings.NewReader(jsonString))
	r.Header.Set("Content-Type", "application/json")
	w := httptest.NewRecorder()
	beego.BeeApp.Handlers.ServeHTTP(w, r)

	assert.Equal(t, http.StatusOK, w.Code)
	var actual types.TokenResponse

	json.NewDecoder(w.Body).Decode(&actual)
	assert.Equal(t, types.TokenResponse{
		Token: "hello-world",
		Error: "",
	}, actual)
	...
}
```

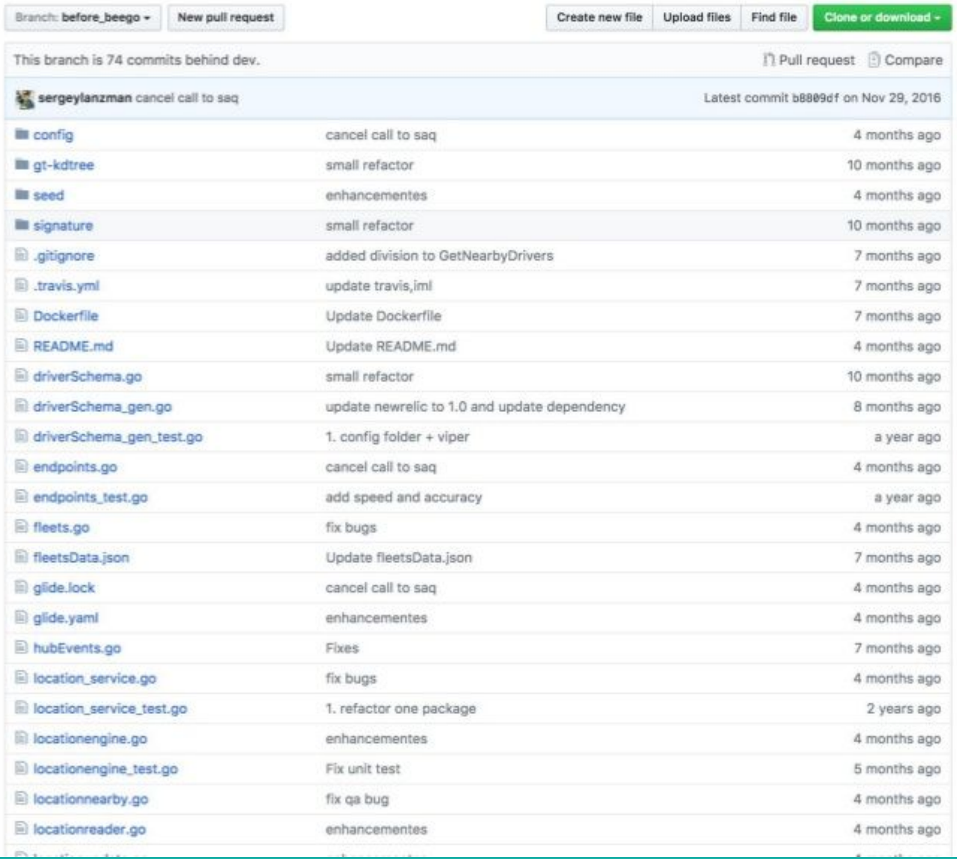
Each test Case feature should be easy to understand.

Consideration

Maintainability

Project struct

Before BeeGo



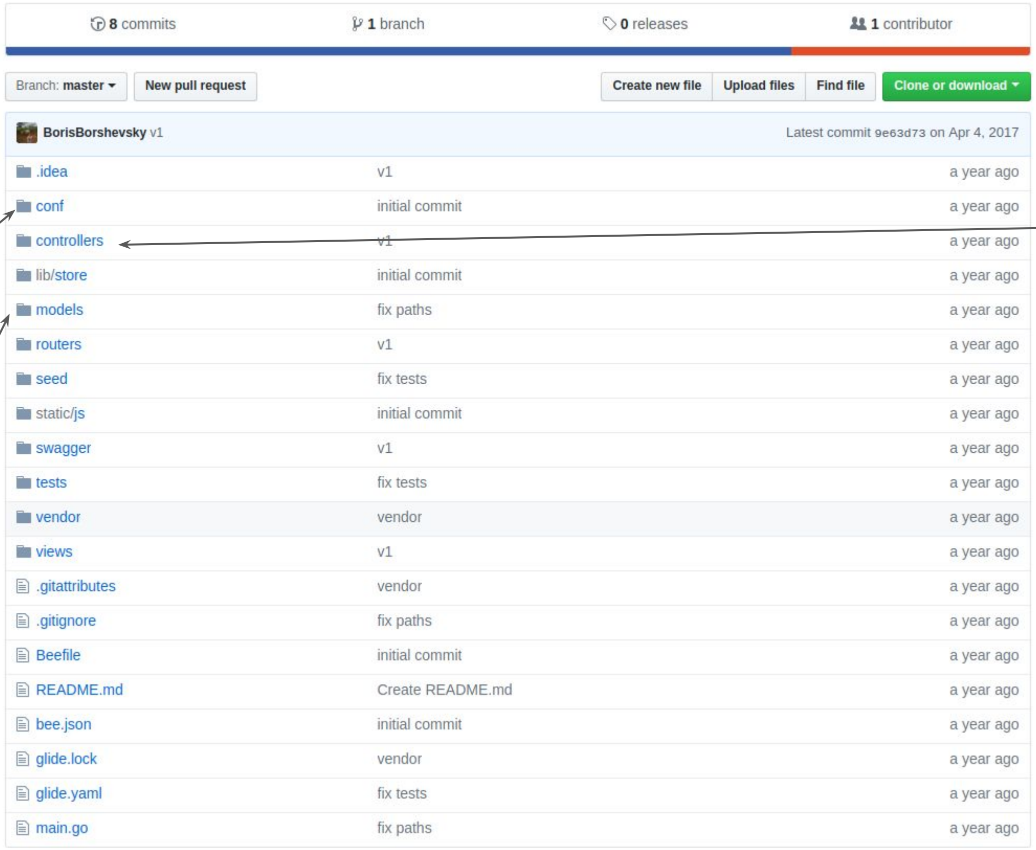
This branch is 74 commits behind dev.

sergeylanzman cancel call to saq — Latest commit b8809df on Nov 29, 2016

config	cancel call to saq	4 months ago
gt-kdtree	small refactor	10 months ago
seed	enhancementes	4 months ago
signature	small refactor	10 months ago
.gitignore	added division to GetNearbyDrivers	7 months ago
.travis.yml	update travis,iml	7 months ago
Dockerfile	Update Dockerfile	7 months ago
README.md	Update README.md	4 months ago
driverSchema.go	small refactor	10 months ago
driverSchema_gen.go	update newrelic to 1.0 and update dependency	8 months ago
driverSchema_gen_test.go	1. config folder + viper	a year ago
endpoints.go	cancel call to saq	4 months ago
endpoints_test.go	add speed and accuracy	a year ago
fleets.go	fix bugs	4 months ago
fleetsData.json	Update fleetsData.json	7 months ago
glide.lock	cancel call to saq	4 months ago
glide.yaml	enhancementes	4 months ago
hubEvents.go	Fixes	7 months ago
location_service.go	fix bugs	4 months ago
location_service_test.go	1. refactor one package	2 years ago
locationengine.go	enhancementes	4 months ago
locationengine_test.go	Fix unit test	5 months ago
locationnearby.go	fix qa bug	4 months ago
locationreader.go	enhancementes	4 months ago

Gett

Project struct (const.)



8 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

BorisBorshevsky v1 — Latest commit 9e63d73 on Apr 4, 2017

.idea	v1	a year ago
conf	initial commit	a year ago
controllers	v1	a year ago
lib/store	initial commit	a year ago
models	fix paths	a year ago
routers	v1	a year ago
seed	fix tests	a year ago
static/js	initial commit	a year ago
swagger	v1	a year ago
tests	fix tests	a year ago
vendor	vendor	a year ago
views	v1	a year ago
.gitattributes	vendor	a year ago
.gitignore	fix paths	a year ago
Beefile	initial commit	a year ago
README.md	Create README.md	a year ago
bee.json	initial commit	a year ago
glide.lock	vendor	a year ago
glide.yaml	fix tests	a year ago
main.go	fix paths	a year ago

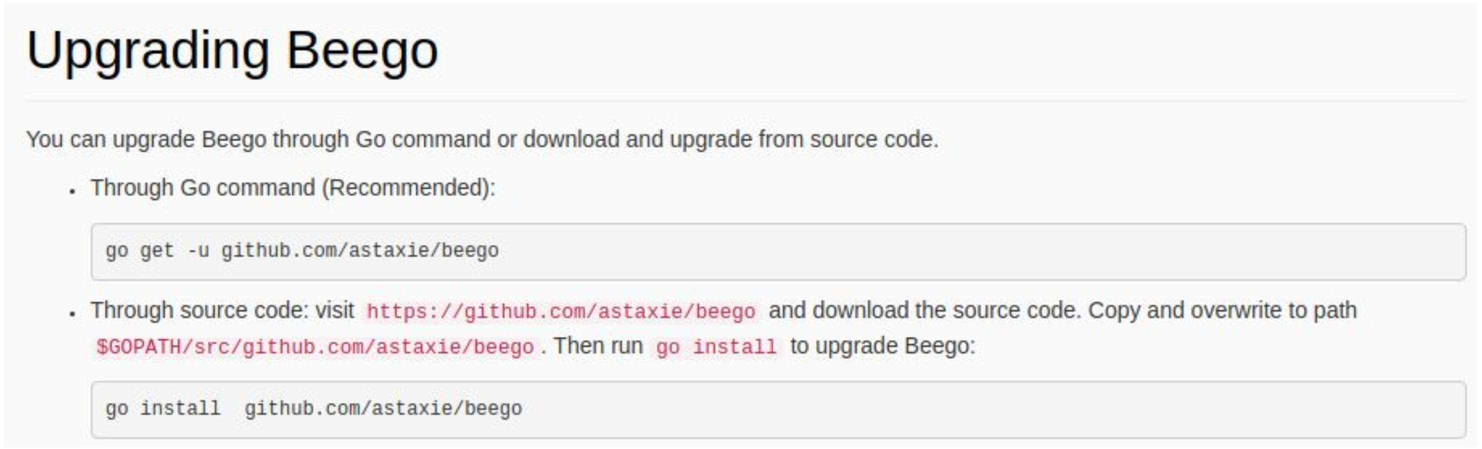
Store handler

Store config

Store model

https://www.slideshare.net/BorisBorshevsky/beego-meetup-442017-gett

Upgrading



Upgrading Beego

You can upgrade Beego through Go command or download and upgrade from source code.

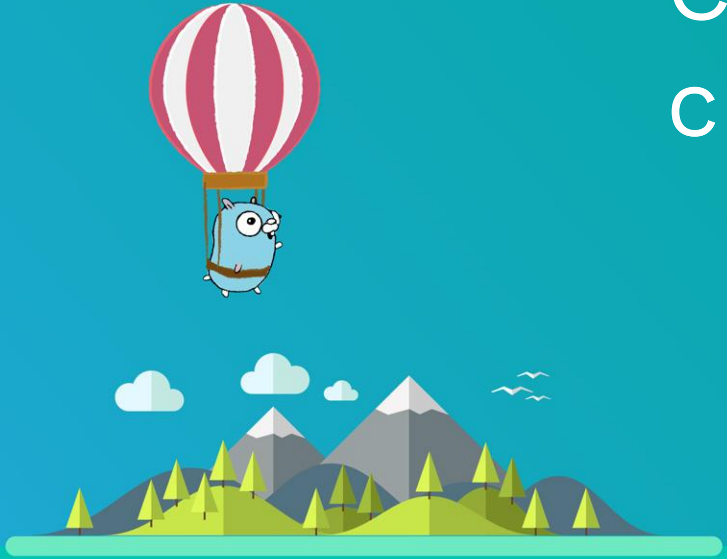
- Through Go command (Recommended):

```
go get -u github.com/astaxie/beego
```

- Through source code: visit `https://github.com/astaxie/beego` and download the source code. Copy and overwrite to path `$GOPATH/src/github.com/astaxie/beego` . Then run `go install` to upgrade Beego:

```
go install  github.com/astaxie/beego
```

“Clear is better than clever.”



ROB PIKE

WORKING WITHIN A COMPANY

Keep it simple, stupid

Beego introduce a structure for engineer to follow, a suite for them

Consideration

Reliable

Star user

By astaxie (https://github.com/astaxie)

Our well-known customers
HUAWEI 华为企业云
乐视云
360 安全中心
有道 youdao
Weico
新浪微博
JD.COM 京东
Bmob
ArdanStudios
OPERA 欧朋
美团网 meituan.com
土豆网 Tudou.com 每个人都是生活的导演
联众游戏
17173.com
淘宝网 Taobao.com
SNDA 盛大网络
Tencent Games
ZALORA
微影时代 微票儿
lianjia 链家 .com

Consideration

Productive

Beego/Bee (https://github.com/beego/bee)

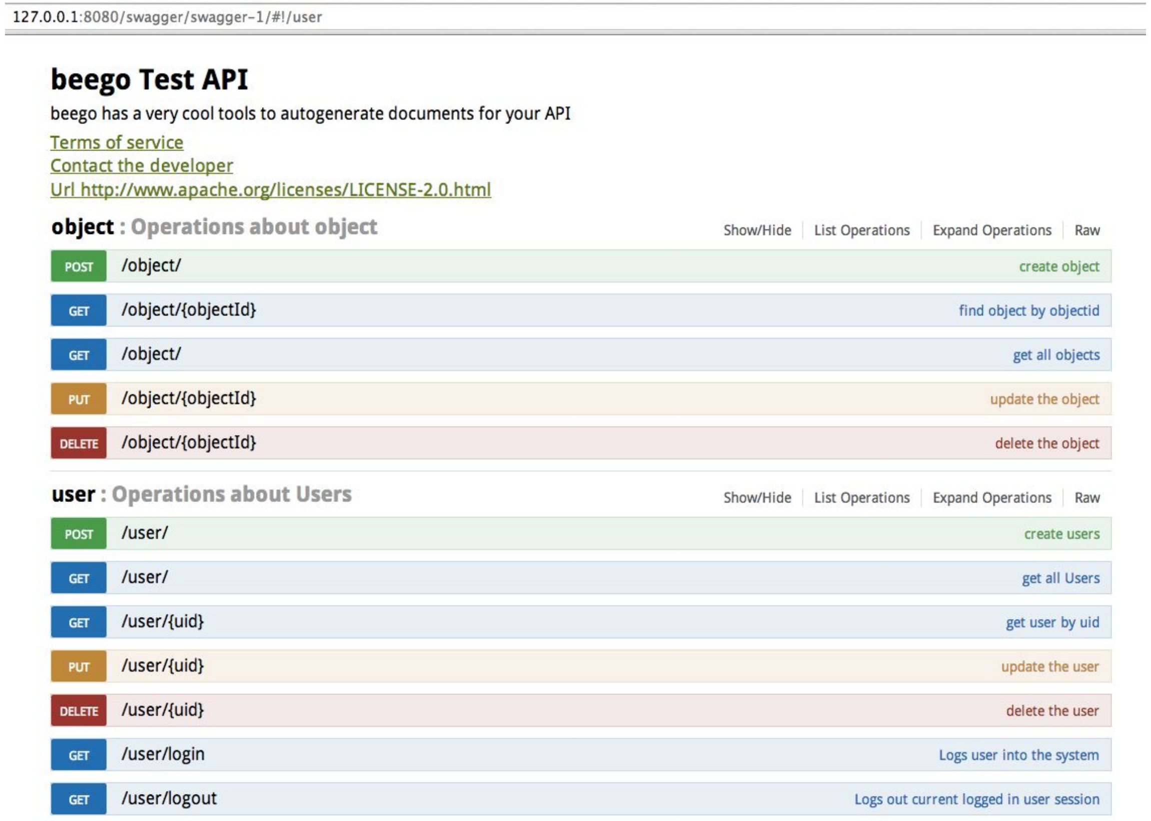
Bee run it !!

```
    version     Prints the current Bee version
    migrate     Runs database migrations
    api         Creates a Beego API application
    bale        Transforms non-Go files to Go source files
    fix         Fixes your application by making it compatible with newer versions of Beego
    dlv         Start a debugging session using Delve
    dockerize   Generates a Dockerfile for your Beego application
    generate    Source code generator
    hprose      Creates an RPC application based on Hprose and Beego frameworks
    new         Creates a Beego application
    pack        Compresses a Beego application into a single file
    rs          Run customized scripts
    run         Run the application by starting a local development server
```

Each Bee feature should be easy to understand.

Bee

1. Hot reload
2. Swagger
3. Init project
4. Dockerize
5. Run script
6. More


127.0.0.1:8080/swagger/swagger-1/#!/user

beego Test API

beego has a very cool tools to autogenerate documents for your API

Terms of service
Contact the developer
Url http://www.apache.org/licenses/LICENSE-2.0.html

object : Operations about object — Show/Hide | List Operations | Expand Operations | Raw

POST	/object/	create object
GET	/object/{objectId}	find object by objectid
GET	/object/	get all objects
PUT	/object/{objectId}	update the object
DELETE	/object/{objectId}	delete the object

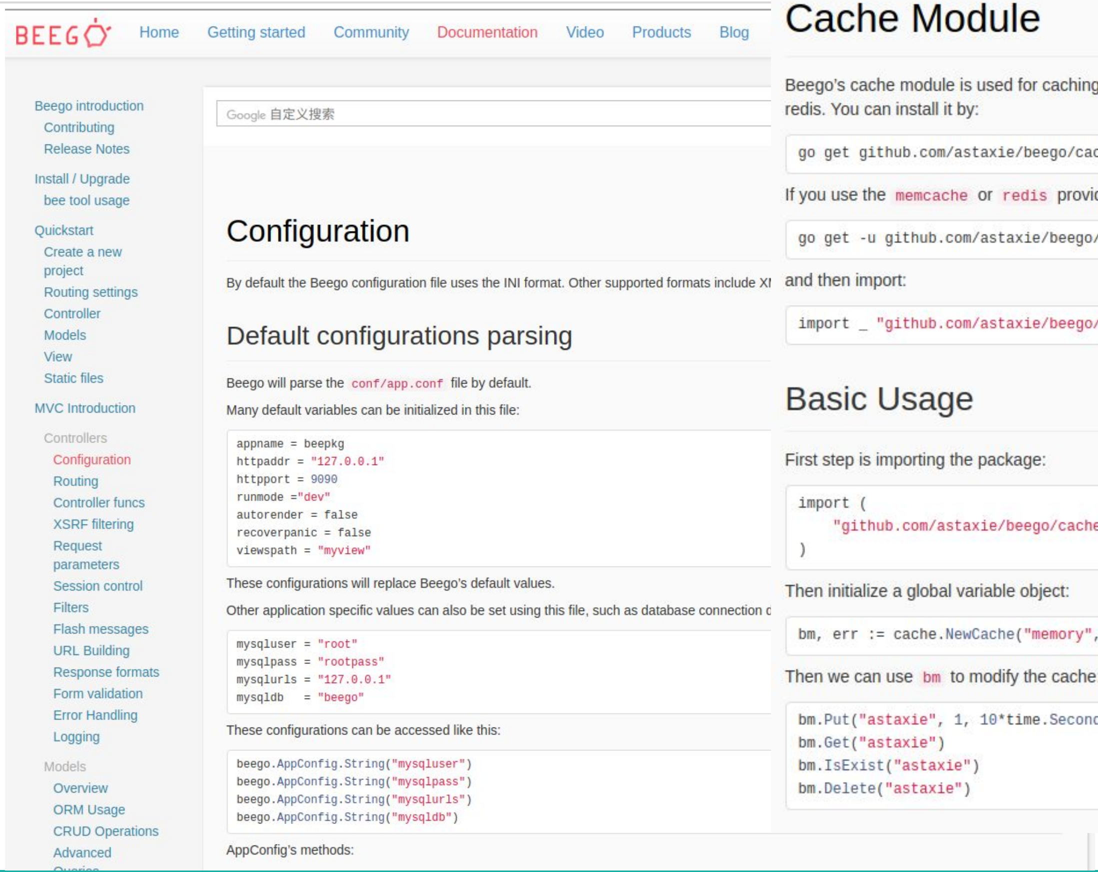
user : Operations about Users — Show/Hide | List Operations | Expand Operations | Raw

POST	/user/	create users
GET	/user/	get all Users
GET	/user/{uid}	get user by uid
PUT	/user/{uid}	update the user
DELETE	/user/{uid}	delete the user
GET	/user/login	Logs user into the system
GET	/user/logout	Logs out current logged in user session

Consideration

Community support

Rich documentation (https://beego.me/docs/intro/)



Configuration

By default the Beego configuration file uses the INI format. Other supported formats include X

Default configurations parsing

Beego will parse the `conf/app.conf` file by default.

Many default variables can be initialized in this file:

```
appname = beepkg
httpaddr = "127.0.0.1"
httpport = 9090
runmode ="dev"
autorender = false
recoverpanic = false
viewspath = "myview"
```

These configurations will replace Beego's default values.

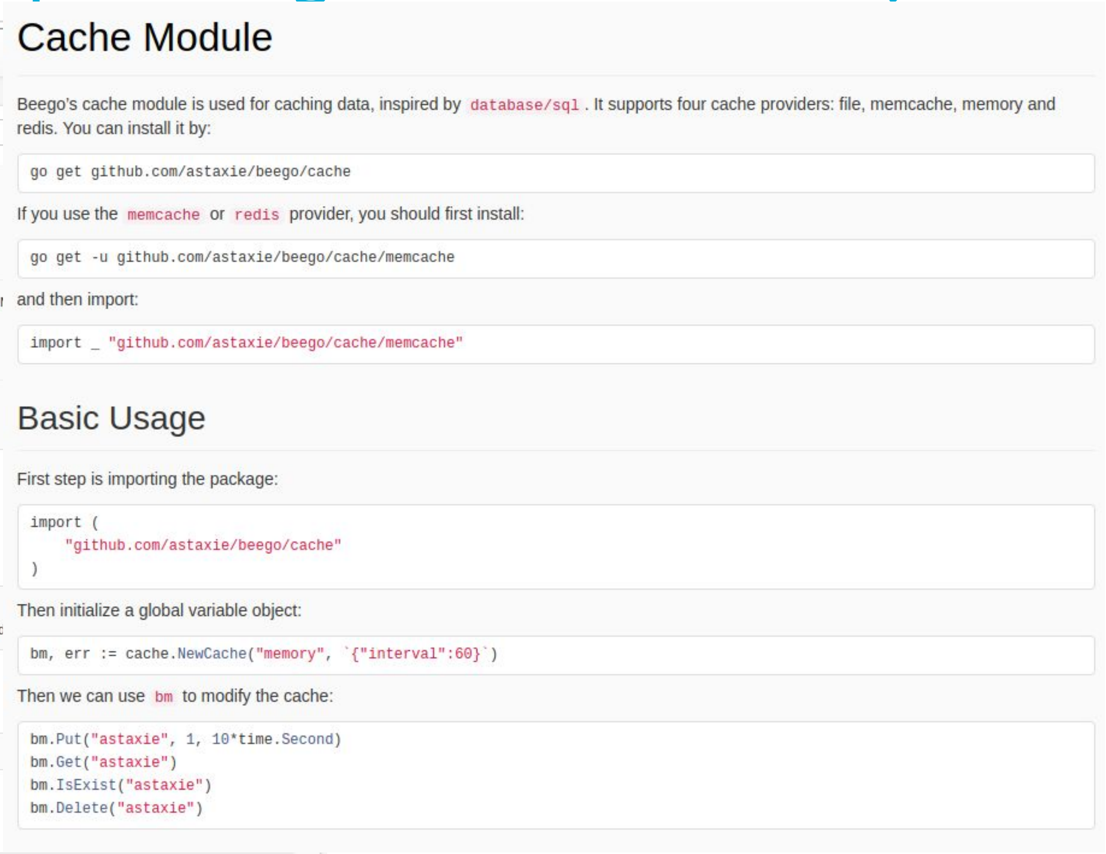
Other application specific values can also be set using this file, such as database connection d

```
mysqluser = "root"
mysqlpass = "rootpass"
mysqlurls = "127.0.0.1"
mysqldb   = "beego"
```

These configurations can be accessed like this:

```
beego.AppConfig.String("mysqluser")
beego.AppConfig.String("mysqlpass")
beego.AppConfig.String("mysqlurls")
beego.AppConfig.String("mysqldb")
```

AppConfig's methods:



Cache Module

Beego's cache module is used for caching data, inspired by `database/sql` . It supports four cache providers: file, memcache, memory and redis. You can install it by:

```
go get github.com/astaxie/beego/cache
```

If you use the `memcache` or `redis` provider, you should first install:

```
go get -u github.com/astaxie/beego/cache/memcache
```

and then import:

```
import _ "github.com/astaxie/beego/cache/memcache"
```

Basic Usage

First step is importing the package:

```
import (
    "github.com/astaxie/beego/cache"
)
```

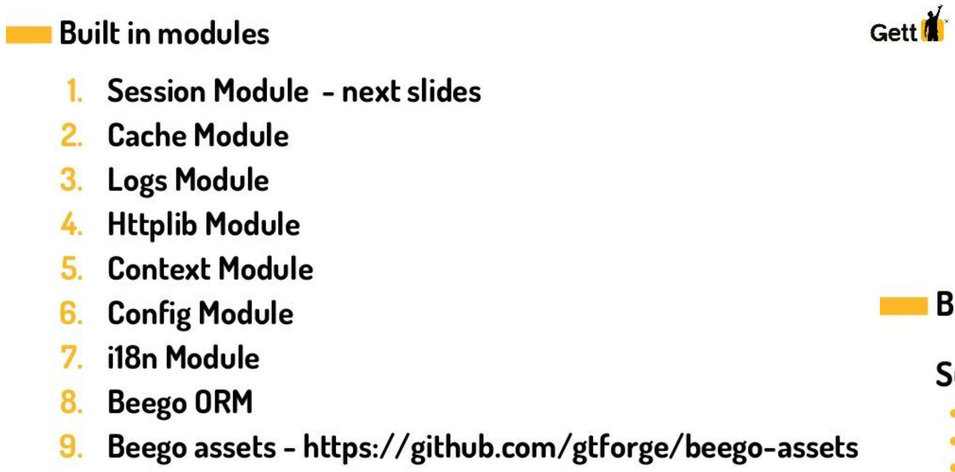
Then initialize a global variable object:

```
bm, err := cache.NewCache("memory", `{"interval":60}`)
```

Then we can use `bm` to modify the cache:

```
bm.Put("astaxie", 1, 10*time.Second)
bm.Get("astaxie")
bm.IsExist("astaxie")
bm.Delete("astaxie")
```

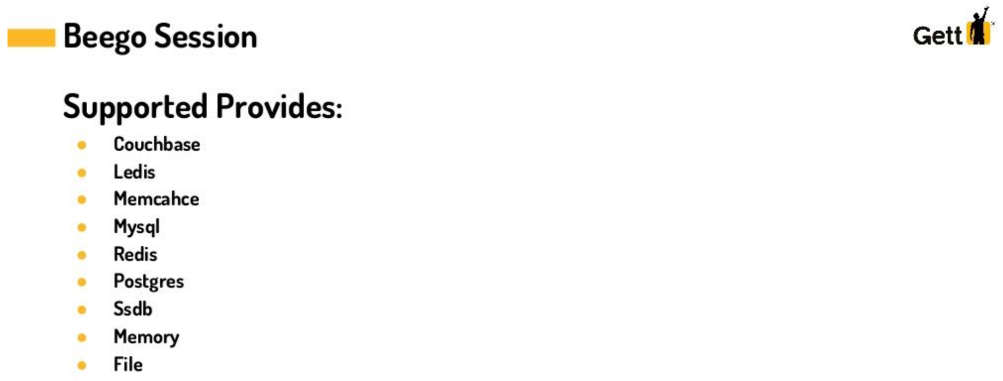
Rich built in module



Built in modules

Gett

1. Session Module - next slides
2. Cache Module
3. Logs Module
4. Httplib Module
5. Context Module
6. Config Module
7. i18n Module
8. Beego ORM
9. Beego assets - https://github.com/gtforge/beego-assets



Beego Session

Gett

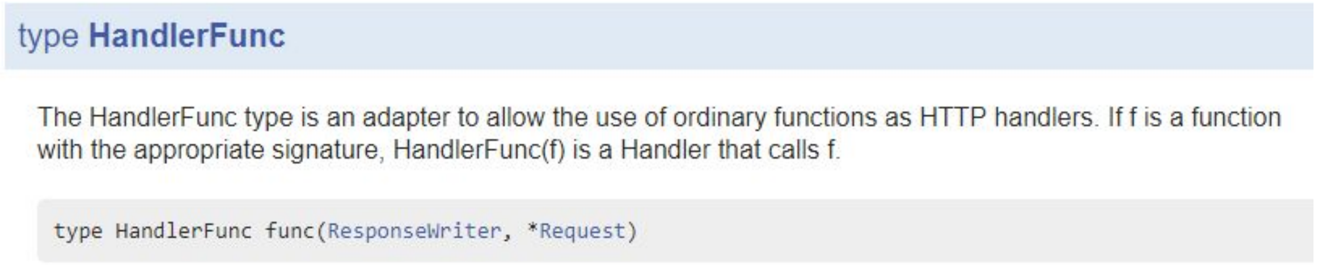
Supported Provides:

- Couchbase
- Ledis
- Memcahce
- Mysql
- Redis
- Postgres
- Ssdb
- Memory
- File

Bonus

Why framework is matter

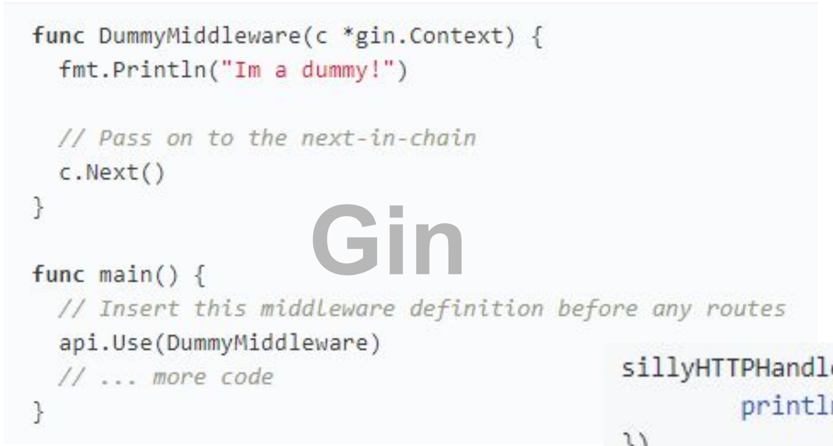
net/http



type HandlerFunc

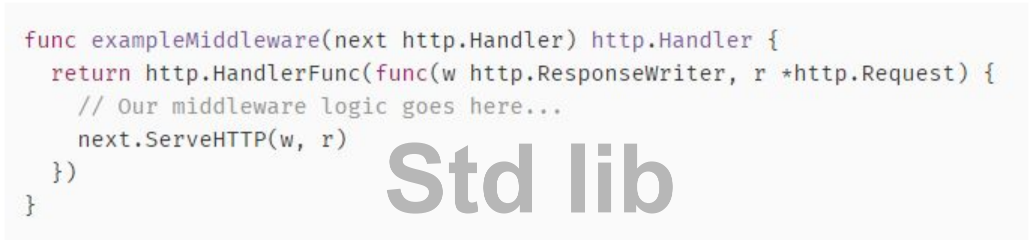
The HandlerFunc type is an adapter to allow the use of ordinary functions as HTTP handlers. If f is a function with the appropriate signature, HandlerFunc(f) is a Handler that calls f.

```
type HandlerFunc func(ResponseWriter, *Request)
```



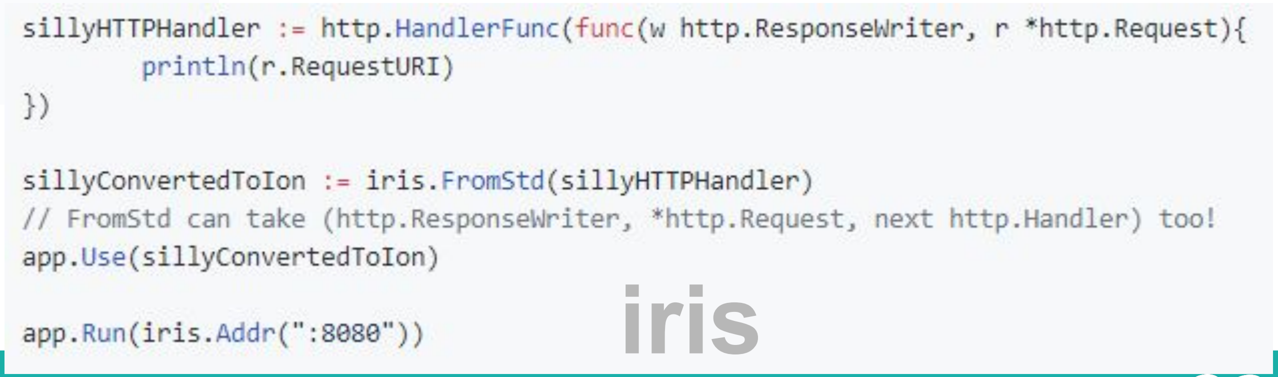
```go
func DummyMiddleware(c *gin.Context) {
  fmt.Println("Im a dummy!")

  // Pass on to the next-in-chain
  c.Next()
}

func main() {
  // Insert this middleware definition before any routes
  api.Use(DummyMiddleware)
  // ... more code
}
```

Gin



```go
func exampleMiddleware(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    // Our middleware logic goes here...
    next.ServeHTTP(w, r)
  })
}
```

Std lib



```go
sillyHTTPHandler := http.HandlerFunc(func(w http.ResponseWriter, r *http.Request){
        println(r.RequestURI)
})

sillyConvertedToIon := iris.FromStd(sillyHTTPHandler)
// FromStd can take (http.ResponseWriter, *http.Request, next http.Handler) too!
app.Use(sillyConvertedToIon)

app.Run(iris.Addr(":8080"))
```

iris

net/http

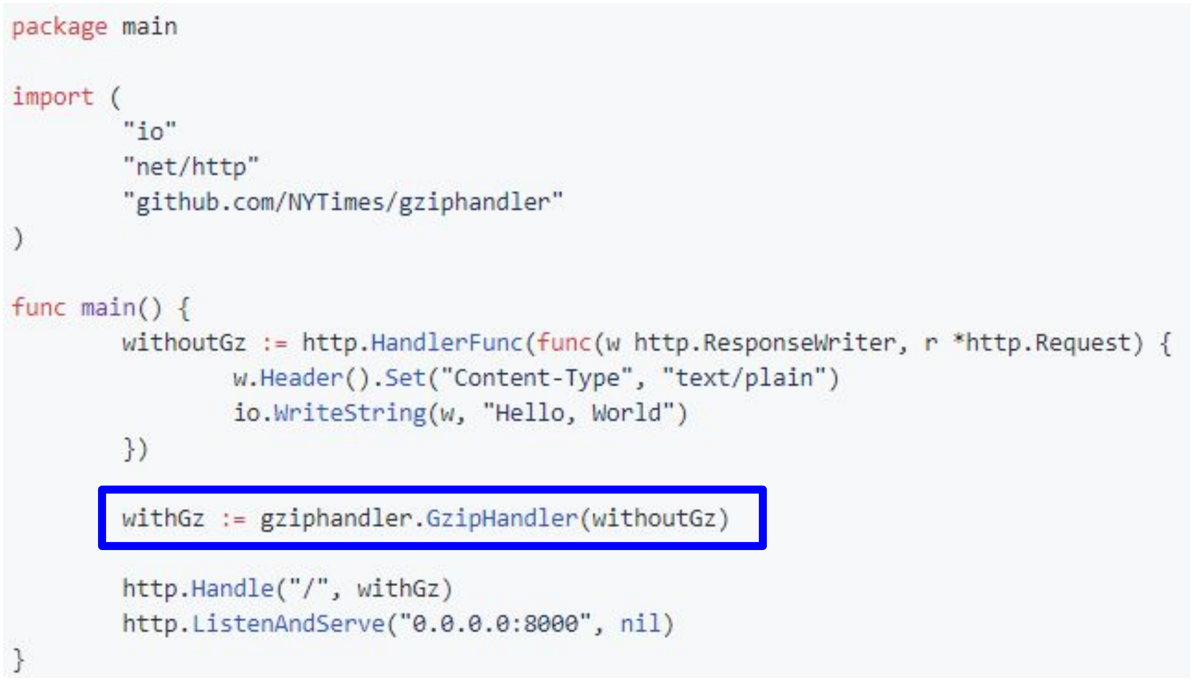
THEY ARE DIFFERENCE !!

Interface

$\left\{ \begin{array}{l} \text{Each framework cannot be used to other (excluding iris)} \end{array} \right.$

WHAT IF I FOUND AN AWESOME MIDDLEWARE

NYTimes/gziphandler (https://github.com/NYTimes/gziphandler)



```go
package main

import (
	"io"
	"net/http"
	"github.com/NYTimes/gziphandler"
)

func main() {
	withoutGz := http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
		w.Header().Set("Content-Type", "text/plain")
		io.WriteString(w, "Hello, World")
	})

	withGz := gziphandler.GzipHandler(withoutGz)

	http.Handle("/", withGz)
	http.ListenAndServe("0.0.0.0:8000", nil)
}
```

It cannot be apply to either Gin or Beego

WHAT WE BELIEVE

Quotes

“

Murphy's Law

If there are two or more ways to do something, and one of those ways can result in a catastrophe, then someone will do it.

Final stage

DEMO

Q&A

Thanks all of my workmate