Golf

Release 0.0

February 19, 2016

Golf is a fast, simple and lightweight micro-web framework for Go. It comes with powerful features and has no dependencies other than the Go Standard Library.

CHAPTER 1

Installation

```
go get github.com/dinever/golf
```

CHAPTER 2

Hello World

Here is a simple "Hello World!" application using Golf:

```
package main

import "github.com/dinever/golf"

func helloWorldHandler(ctx *Golf.Context) {
    ctx.Write("Hello World!")
}

func main() {
    app := Golf.New()
    app.Get("/", helloWorldHandler)
    app.Run(":9000")
}
```

CHAPTER 3

Documentation

3.1 Quickstart

3.1.1 A Simple Application

To get started with a simple Golf application, create a file named `app.go`:

```
package main

import "github.com/dinever/golf"

func helloWorldHandler(ctx *Golf.Context) {
    ctx.Write("Hello World!")
}

func main() {
    app := Golf.New()
    app.Get("/", helloWorldHandler)
    app.Run(":9000")
}
```

And run the application by using the go compiler:

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

Then you should be able to see the result if you take a look at http://localhost:9000.

3.1.2 Routing

Routing in Golf is easy. Here are some examples:

```
App.Get("/", HomeHandler)
App.Get("/p/:page/?", PageHandler)
App.Post("/comment/:id/", CommentHandler)
```

In the above example, we added 3 routing rules. They are:

- HTTP **GET** method on the path `/`. If user visits **http://example.com/**, the request will be passed to `HomeHandler`.
- HTTP **GET** method on the path `/p/:page/?`. Every request with a path matching this routing rule, e.g. `/p/1/`, `/p/101/` or `/p/12345` will be passed to `PageHandler`, and what user inputed in the `page` field

will be stored in `ctx.Params`. Please notice that the last question mark `?` in the routing rule means that the previous `/` is optional, and both `/p/123/` and `/p/123` can be matched.

- HTTP **POST** method on the path `/comment/:id/`. This is similar to the previous one. The only difference is that this one matches a `POST` method.

The parameters will be passed in `ctx.Params`, and can be get by this:

```
p, ok := ctx.Params["page"]
```

Other than **GET** and **POST**, you can use methods `App.Put` and `App.Delete` to handle **PUT** and **DELETE** requests.

3.1.3 Static Files

Static files, like stylesheets and javascript files can be handled easily by calling `App.Static`:

```
App.Static("/static/", "static")
App.Static("/static/", "resources")
```

The first argument is the URL path for static files, the second argument is the path of the folder holding static files on your filesystem. Please notice that Golf supports serving multiple folders into one URL root PATH, which means that if you set Golf like the above example, when user requested `/static/style.css`, Golf will firstly lookup the file `style.css` in the folder `static`, and continue looking up the same file in the folder `resources` if it is not found in the previous folder.

3.1.4 Rendering Templates

If you are familiar with the Django/Jinja2 style template syntax, it may not comfortable for you to get in touch with the Go style template, especially the template inheritence part. However, using Golf you can handle it like a charm. Let's take a look at an example:

`main.go`:

```
package main

import (
  "github.com/dinever/golf"
)

func homeHandler(ctx *Golf.Context) {
  data := map[string]interface{}{
    "Title": "Hello World",
  }
  ctx.Loader("template").Render("index.html", data)
}

func main() {
  App := Golf.New()
  App.View.SetTemplateLoader("template", "templates/")
  App.Get("/", homeHandler)
  App.Run(":9000")
}
```

`templates/index.html`:

```
<h1>{{ .Title }}</h1>
```

The first step is to indicating a template loader for `App.View`. The first argument is the name of the template loader, the second argument is the path of the template folder. Golf allows you to indicating multiple template loaders, e.g., one for the templates of admin panel and another one for the templates of front-end.

After the template loader is set, you can render templates inside the loader by calling `ctx.Loader("loader_name").Render("file_name", data)`. Indicating a template loader before calling Render is necessary, otherwise Golf can not find out the template file.

3.1.5 Redirection

Simple call `ctx.Redirect` with the path as an argument. By default Golf set the status code to 301. If you want a 302 Redirection, please manually set the header as 302 after calling `ctx.Redirect`.

```
ctx.Redirect("/foo")
```

3.1.6 Error Handling

You can use `Golf.Error` to set handlers for different type of errors like the following:

```
App.Error(401, unauthorizedHandler)
App.Error(404, notFoundHandler)
```

To raise an error inside context, please use the method `Context.Abort`:

```
func homeHandler(ctx *Golf.Context) {
  user, e := getUser()
  if e != nil {
    ctx.Abort(401)
    return
  }
  ctx.Loader("template").Render("index.html", data)
}
```

Golf will find the corresponding error handler in the error handler map. If the error handler for this status code is not set, golf calls `App.DefaultErrorHandler`.

You can also indicate a `DefaultErrorHandler` by yourself.

```
App.DefaultErrorHandler = errorHandler
```

CHAPTER 4

LICENSE

Apache License