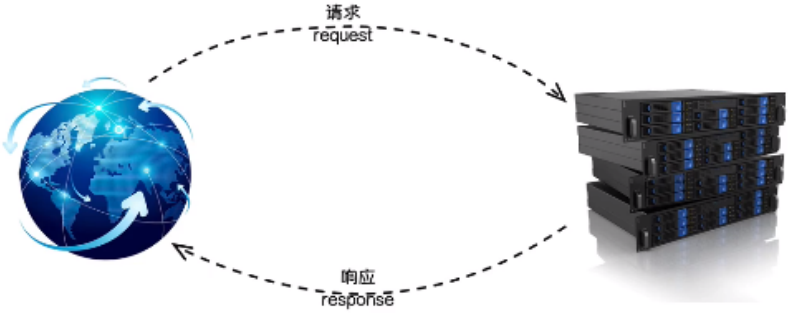
1.

Gin框架系列教程



2.

参考老师博客地址:

https://www.liwenzhou.com/posts/Go/Gin_framework/

gin文档:

https://gin-gonic.com/zh-cn/docs/

参考写gin的网站

3.

在项目下直接输入安装命令即可:

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

```
package main

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

func sayHello(c *gin.Context){
 c.JSON(200,gin.H{ //code:200
 // c.JSON: 返回JSON格式的数据
  "message":"Hello golang!",
  })
}

func main() {
  // 创建一个默认的路由引擎
```

```
 r := gin.Default() //返回默认的路由引擎

 // GET：请求方式；/hello：请求的路径
 // 当客户端以GET方法请求/hello路径时，会执行后面的匿名函数
 //指定用户使用GET请求访问/hello时,执行sayHello这个函数
 r.GET("/hello", sayHello)

 // 启动HTTP服务，默认在0.0.0.0:8080启动服务
 r.Run()
}
```

```
D:\Gopro\src\studybro1>go build

D:\Gopro\src\studybro1>gin_victory01.exe
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery m
iddleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in produ
ction.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)
```

运行

修改端口后的代码:

```
package main

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

func sayHello(c *gin.Context){
 c.JSON(200,gin.H{ //code:200
 // c.JSON：返回JSON格式的数据
  "message":"Hello golang!",
  })
}

func main() {
 // 创建一个默认的路由引擎
 r := gin.Default() //返回默认的路由引擎
```

```
17
18  // GET：请求方式；/hello：请求的路径
19  // 当客户端以GET方法请求/hello路径时，会执行后面的匿名函数
20  //指定用户使用GET请求访问/hello时,执行sayHello这个函数
21  r.GET("/hello", sayHello)
22
23  // 启动HTTP服务，默认在0.0.0.0:8080启动服务
24  r.Run(":9090") //修改端口:9090
25 }
```

编译后,运行

```
D:\Gopro\src\studybro1>go build

D:\Gopro\src\studybro1>gin_victory01.exe
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middlewar
e already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /hello                    --> main.sayHello (3 handlers)
[GIN-debug] Listening and serving HTTP on :9090
```

浏览器显示的结果:

{"message":"Hello golang!"}

另一种形式的程序,第一个Gin示例

```
1  package main
2
3  import (
4   "github.com/gin-gonic/gin"
5  )
6
7  func main() {
8   // 创建一个默认的路由引擎
9   r := gin.Default()
10   // GET：请求方式；/hello：请求的路径
11   // 当客户端以GET方法请求/hello路径时，会执行后面的匿名函数
```

```
 r.GET("/hello", func(c *gin.Context) {
 // c.JSON：返回JSON格式的数据
 c.JSON(200, gin.H{
 "message": "Hello world!",
 })
 })
 // 启动HTTP服务，默认在0.0.0.0:8080启动服务
 r.Run()
}
```

4.

下面看一下:

RESTful API:只是一种规范,注意

```
GET用来获取资源
POST用来新建资源
PUT用来更新资源
DELETE用来删除资源。
```

5.

示例:

```
//(1)
//package main
//
//import (
// "fmt"
//)
//
//func main() {
// fmt.Println("hello")
//}

//(2)
//package main
//
//import (
// "github.com/gin-gonic/gin"
//)
//
//func main() {
```

```
// // 创建一个默认的路由引擎
// r := gin.Default() //返回默认的路由引擎
//
// // GET：请求方式；/hello：请求的路径
// // 当客户端以GET方法请求/hello路径时，会执行后面的匿名函数
// r.GET("/hello", func(c *gin.Context) {
// // c.JSON：返回JSON格式的数据
// c.JSON(200, gin.H{
// "message": "Hello world!",
// })
// })
// // 启动HTTP服务，默认在0.0.0.0:8080启动服务
// r.Run()
//}

//(3)
package main

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

func sayHello(c *gin.Context){
 c.JSON(200,gin.H{ //code:200
 // c.JSON：返回JSON格式的数据
 "message":"Hello golang!",
 })
}

func main() {
 // 创建一个默认的路由引擎
 r := gin.Default() //返回默认的路由引擎

 // GET：请求方式；/hello：请求的路径
 // 当客户端以GET方法请求/hello路径时，会执行后面的匿名函数
 //指定用户使用GET请求访问/hello时,执行sayHello这个函数
 r.GET("/hello", sayHello)

 r.GET("/book", sayHello)
 r.GET("/create_book", sayHello)
```

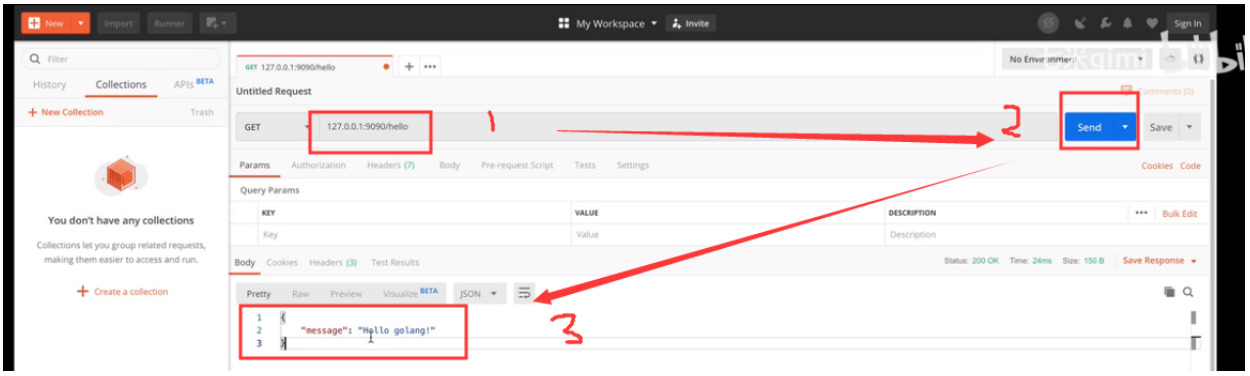
```
60  r.GET("/update_book", sayHello)
61  r.GET("/shanchu_book", sayHello)
62
63
64  r.GET("/books", sayHello)
65  r.POST("/books", sayHello)
66  r.PUT("/books", sayHello)
67  r.DELETE("/books", sayHello)
68
69  // 启动HTTP服务，默认在0.0.0.0:8080启动服务
70  r.Run(":9090") //修改端口:9090
71 }
```

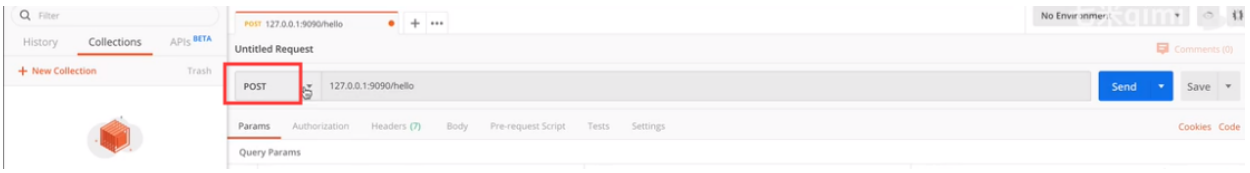
6.

压力测试:

https://www.getpostman.com/

```
1  开发RESTful API的时候我们通常使用Postman来作为客户端的测试工具。
```





我们可以修改请求的方式:

get post put delete

7.

```go
package main

import (
 "github.com/gin-gonic/gin"
 "net/http"
)

func sayHello(c *gin.Context){
 c.JSON(200,gin.H{ //code:200
  // c.JSON：返回JSON格式的数据
  "message":"Hello golang!",
  })
 }

 func main() {
  // 创建一个默认的路由引擎
  r := gin.Default() //返回默认的路由引擎

  // GET：请求方式；/hello：请求的路径
  // 当客户端以GET方法请求/hello路径时，会执行后面的匿名函数
  //指定用户使用GET请求访问/hello时,执行sayHello这个函数
  r.GET("/hello", sayHello)

  //r.GET("/book", sayHello)
  //r.GET("/create_book", sayHello)
  //r.GET("/update_book", sayHello)
  //r.GET("/shanchu_book", sayHello)


  r.GET("/books", func(c *gin.Context) {
  c.JSON(200, gin.H{
  "method":"GET",
  //"message": "Hello golang!",
  })
  })
  r.POST("/books", func(c *gin.Context) {
  c.JSON(http.StatusOK,gin.H{
  "method":"POST",
  })
```

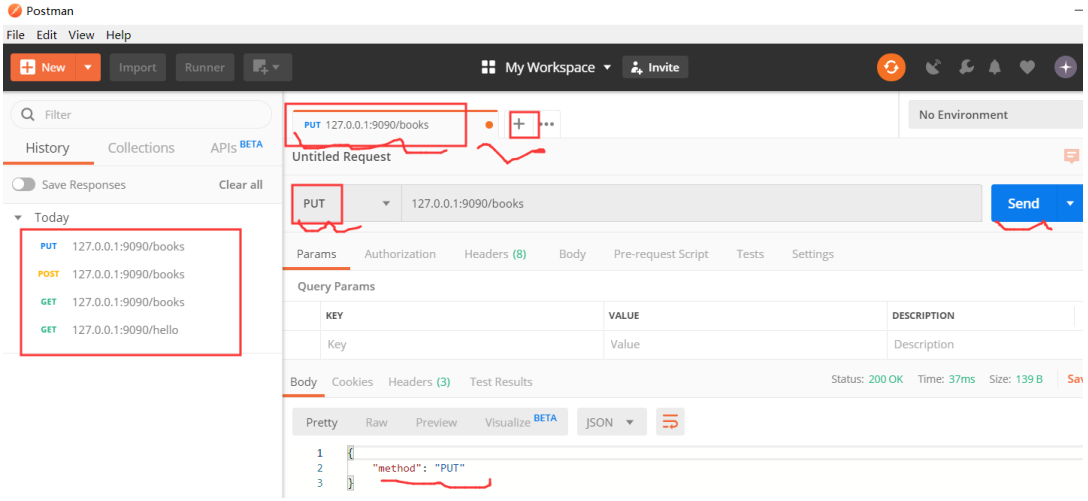
```
40  })
41  r.PUT("/books", func(c *gin.Context) {
42  c.JSON(http.StatusOK,gin.H{
43  "method":"PUT",
44  })
45  })
46
47  r.DELETE("/books", func(c *gin.Context) {
48  c.JSON(http.StatusOK,gin.H{
49  "method":"DELETE",
50  })
51  })
52
53  // 启动HTTP服务，默认在0.0.0.0:8080启动服务
54  r.Run(":9090") //修改端口:9090
55 }
```

运行下面,先编译,再执行(需要注意一下路径):

```
Terminal:  Local  +
D:\Gopro\src\studybro1>go build

D:\Gopro\src\studybro1>gin_victory01.exe
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /hello                    --> main.sayHello (3 handlers)
[GIN-debug] GET    /books                    --> main.main.func1 (3 handlers)
[GIN-debug] POST   /books                    --> main.main.func2 (3 handlers)
[GIN-debug] PUT    /books                    --> main.main.func3 (3 handlers)
[GIN-debug] DELETE /books                    --> main.main.func4 (3 handlers)
[GIN-debug] Listening and serving HTTP on :9090
[GIN] 2019/11/21 - 10:25:22 |?[97;42m 200 ?[0m|     22.9422ms |       127.0.0.1 |?[97;44m GET     ?[0m /hello
[GIN] 2019/11/21 - 10:25:28 |?[97;42m 200 ?[0m|            0s |       127.0.0.1 |?[97;44m GET     ?[0m /books
[GIN] 2019/11/21 - 10:25:34 |?[97;42m 200 ?[0m|            0s |       127.0.0.1 |?[97;46m POST    ?[0m /books
[GIN] 2019/11/21 - 10:25:38 |?[97;42m 200 ?[0m|     13.9621ms |       127.0.0.1 |?[90;43m PUT     ?[0m /books
```

测试:

postman需要先注册,才能够使用


Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
No Environment
Filter
History
Collections
APIs BETA
Save Responses
Clear all
Today
PUT 127.0.0.1:9090/books
POST 127.0.0.1:9090/books
GET 127.0.0.1:9090/books
GET 127.0.0.1:9090/hello
PUT 127.0.0.1:9090/books
Untitled Request
PUT
127.0.0.1:9090/books
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Query Params
KEY
VALUE
DESCRIPTION
Key
Value
Description
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 37ms
Size: 139 B
Pretty
Raw
Preview
Visualize BETA
JSON
1 {
2 "method": "PUT"
3 }