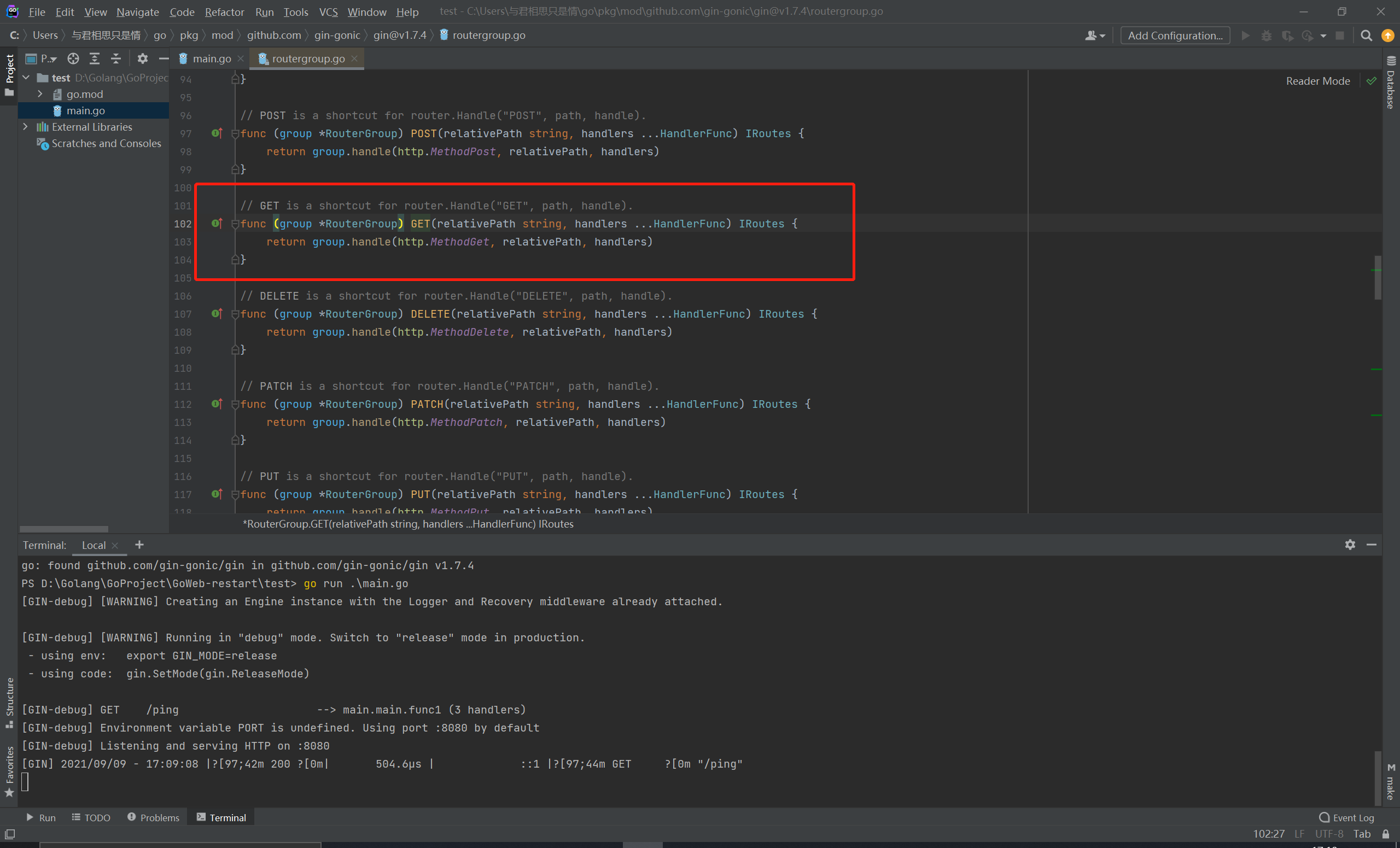
**GoWeb基础**

* **Gin基础**

1. **gin基本框架**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  //创建一个gin默认的Engine实例  r := gin.Default()    r.GET("/ping", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "msg": "pong",  })  })    //默认端口8080,http://localhost:8080/...  r.Run()  //设定端口1234  r.Run("1234")  } |

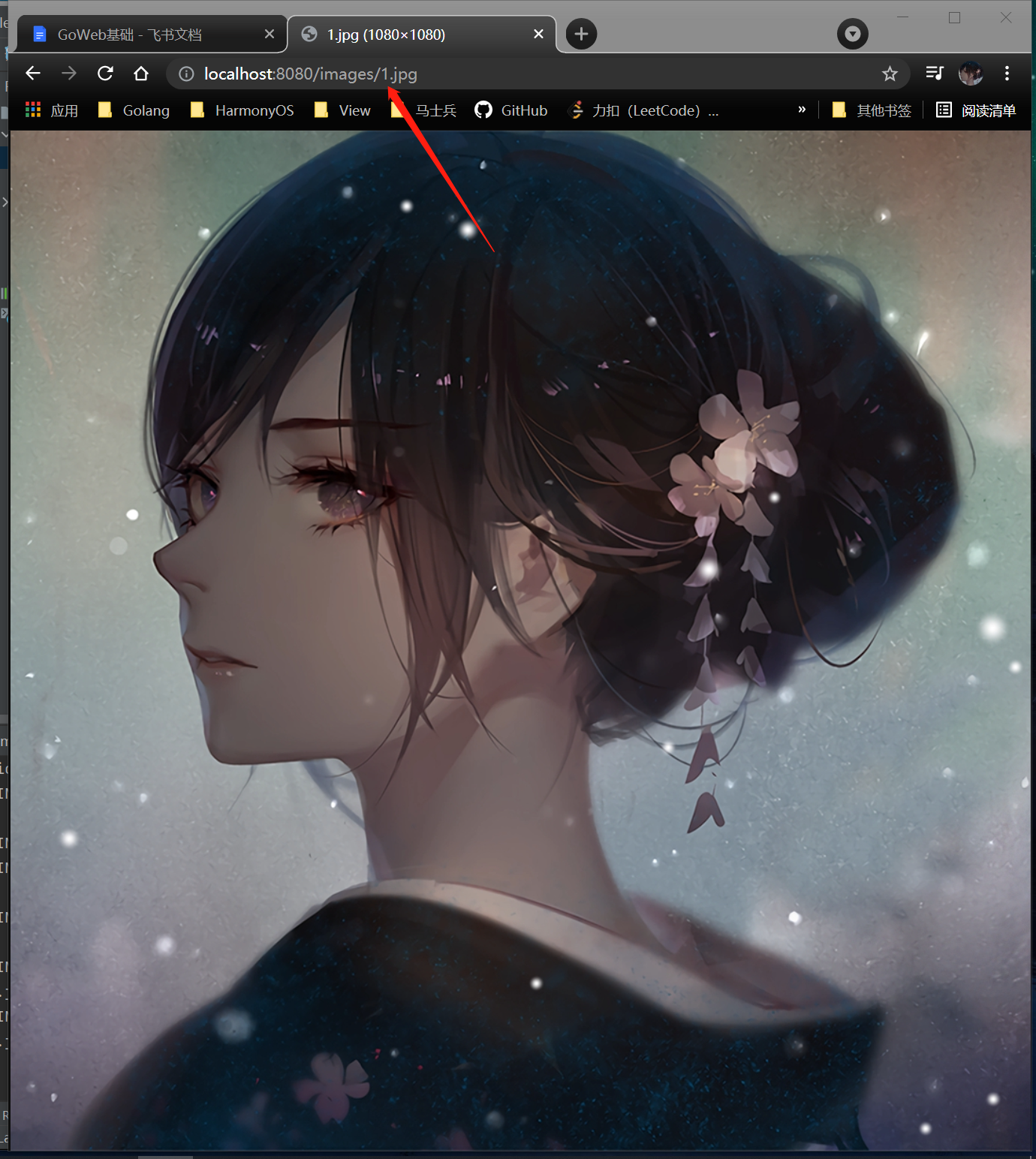


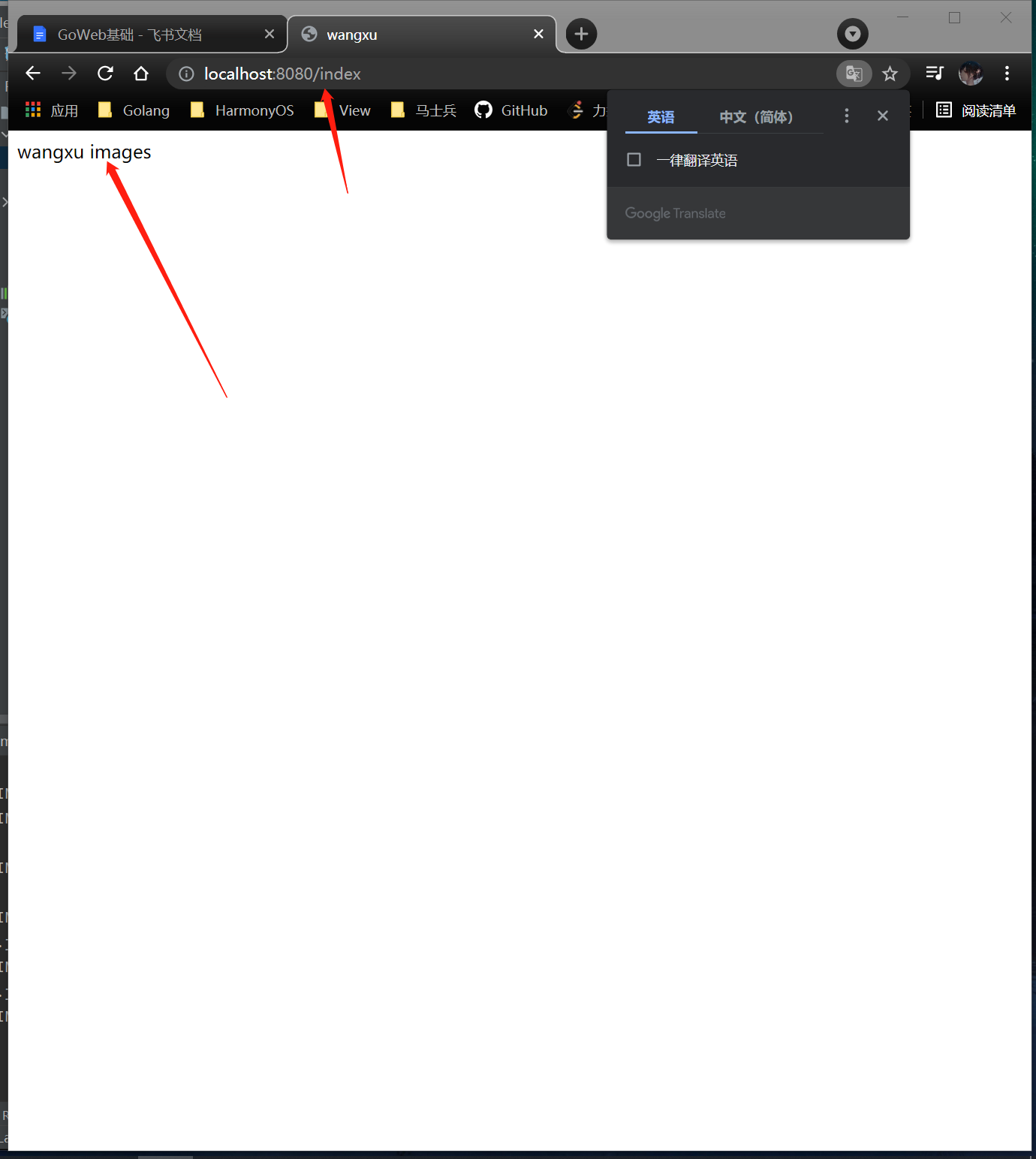
1. **多种请求类型**

|  |
| --- |
| Go  import (  "fmt"  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()    //GET 获取所有的文章信息  r.GET("/ping", func(c \*gin.Context) {  c.String(http.*StatusOK*, "GET")  })  //POST 创建一篇新文章  r.POST("/ping", func(c \*gin.Context) {  c.String(http.*StatusOK*, "POST")  })  //PUT 修改一篇文章  r.PUT("/ping/:id", func(c \*gin.Context) {  c.String(http.*StatusOK*, fmt.Sprintf("PUT id:%s", c.Param("id")))  })  //DELETE 删除一篇文章  r.DELETE("/ping", func(c \*gin.Context) {  c.String(http.*StatusOK*, "DELETE")  })    //Any 匹配所有的请求方法  r.Any("/users", func(c \*gin.Context) {  c.String(http.*StatusOK*, "ANY")  })  r.Run()  } |

1. **加载静态文件**

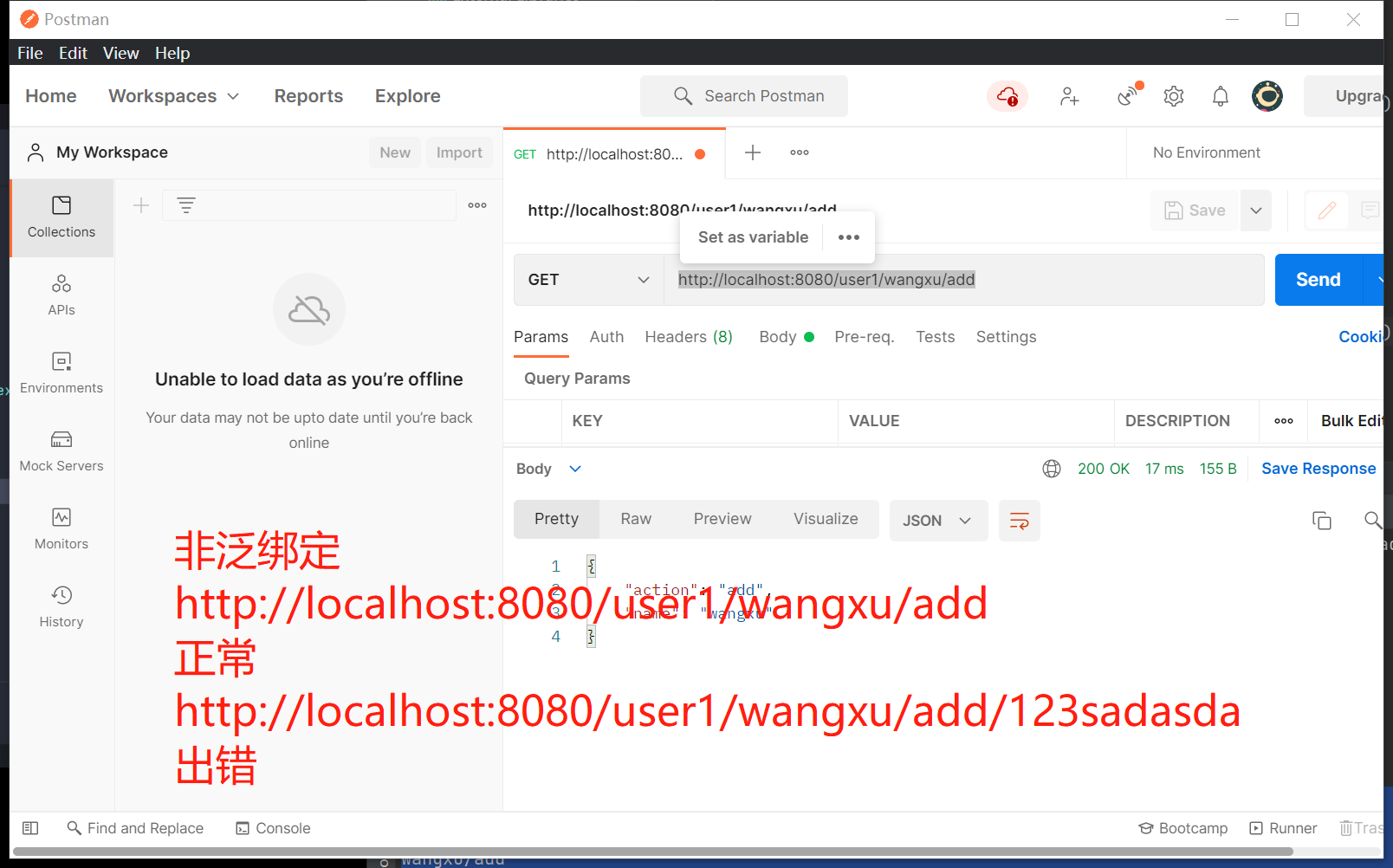
|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()  //加载静态文件  r.Static("./images", "./images")    // StaticFS works just like `Static()` but a custom `http.FileSystem` can be used instead.  // Gin by default user: gin.Dir()  r.StaticFS("/static", http.Dir("./static"))    //加载单独的静态文件  r.StaticFile("/index", "index.html")  r.Run()  } |

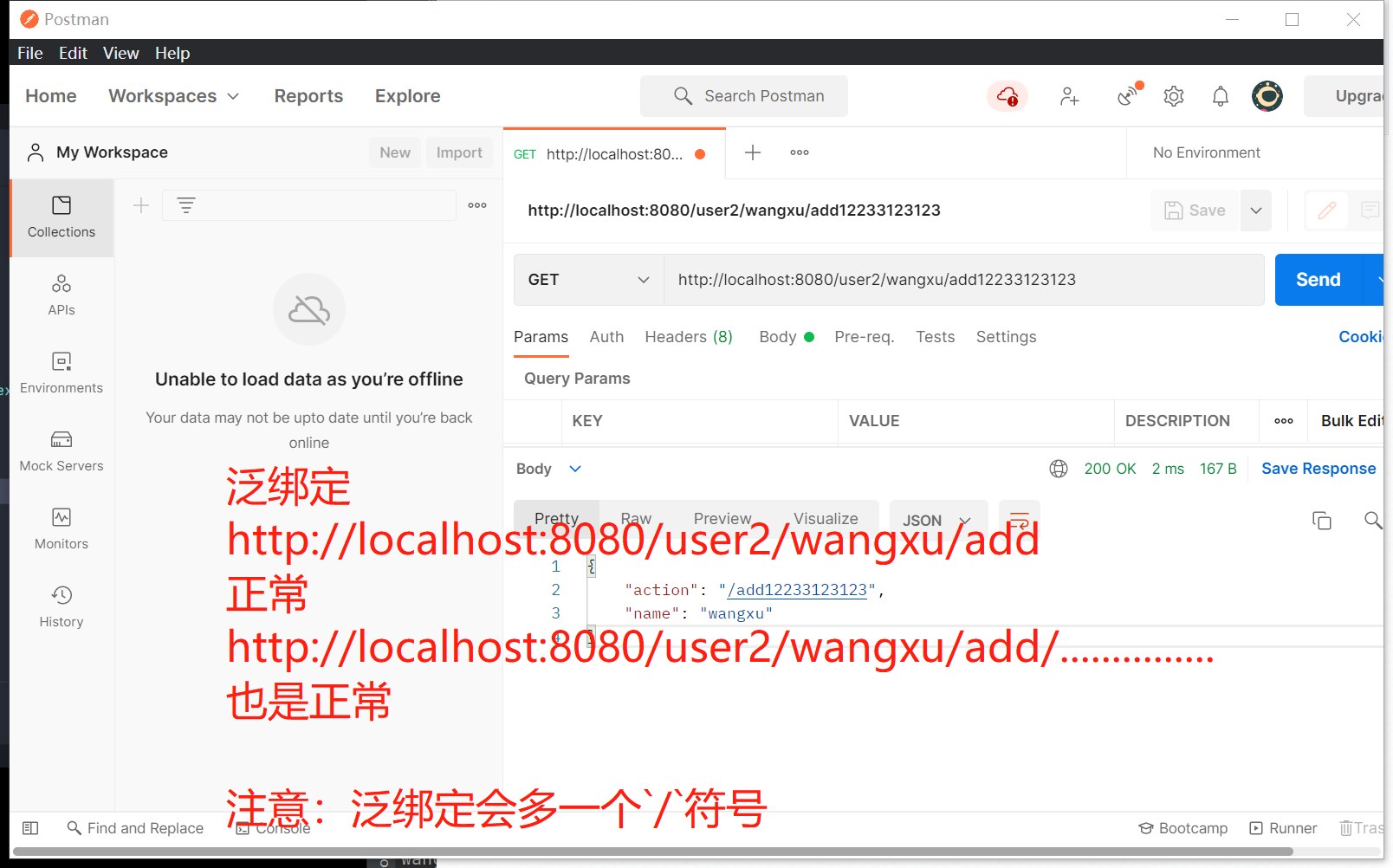




1. **泛绑定**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()  //不是泛绑定  r.GET("/user1/:name/:action", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "name": c.Param("name"),//c.Param() 取参数  "action": c.Param("action"),  })  })    //泛绑定  r.GET("/user2/:name/\*action", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "name": c.Param("name"),  "action": c.Param("action"),  })  })    r.Run()  } |





1. **路由分组**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func getHandler(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "msg": "获取文章列表",  })  }    func postHandler(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "msg": "新建文章",  })  }    func deleteHandler(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "msg": "删除文章",  "id": c.Param("id"),  })  }    func main() {  r := gin.Default()    //常规使用  //r.GET("/posts", getHandler) //获取文章列表  //r.POST("/posts", postHandler) //新建一篇文章  //r.DELETE("/posts/:id", deleteHandler) //删除序号为id的一篇文章    //路由分组  p := r.Group("/posts")  {  p.GET("", getHandler)  p.POST("", postHandler)  p.DELETE("/:id", deleteHandler)  }  r.Run()  } |

* **参数获取**

1. **Parameters in path（路径中的参数）**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()  r.GET("/posts", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "msg": "获取所有的文章列表",  })  })    r.GET("/posts/:id", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "msg": "获取序号为id的文章",  "id": c.Param("id"),//获取路径中的参数id  })  })    r.Run()  } |

1. **Querystring parameters（查询字符串参数）**
2. **Get Query Array 和 Query Map（获取查询数组和查询映射）**

|  |
| --- |
| Swift  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()    r.GET("/welcome", welcomeHanlder)  r.GET("/array", arrayHanlder)  r.GET("/map", mapHandler)  r.Run()  }    //询字符串参数（Querystring parameters）  func welcomeHanlder(c \*gin.Context) {  //firstname lastname  firstName := c.DefaultQuery("firstname", "wangxu")  //lastnName := c.DefaultQuery("lastname", "wangxu")  lastnName := c.Query("lastname")  c.JSON(http.*StatusOK*, gin.H{  "firstname": firstName,  "lastname": lastnName,  })  }    //获取查询数组（Get Query Array）  func arrayHanlder(c \*gin.Context) {  ids := c.QueryArray("ids")  c.JSON(http.*StatusOK*, gin.H{  "ids": ids,  })  }    //获取查询映射（Query Map）  func mapHandler(c \*gin.Context) {  m := c.QueryMap("user")  c.JSON(http.*StatusOK*, gin.H{  "map": m,  })  } |

1. **Post Form**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()    r.POST("/post", postHandler)  r.Run()  }    func postHandler(c \*gin.Context) {  message := c.PostForm("message")  name := c.DefaultPostForm("name", "wangxu")  c.JSON(http.*StatusOK*, gin.H{  "message": message,  "name": name,  })  } |

1. **PostFormMap和PostFormArray**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()    r.POST("/post", postHandler)  r.POST("/array", arrayHandler)  r.POST("/map", mapHandler)  r.Run()  }    func mapHandler(c \*gin.Context) {  u := c.PostFormMap("user")  c.JSON(http.*StatusOK*, gin.H{  "data": u,  })  }    func arrayHandler(c \*gin.Context) {  arr := c.PostFormArray("list")  c.JSON(http.*StatusOK*, gin.H{  "data": arr,  })  }    func postHandler(c \*gin.Context) {  message := c.PostForm("message")  name := c.DefaultPostForm("name", "wangxu")  c.JSON(http.*StatusOK*, gin.H{  "message": message,  "name": name,  })  } |

1. **query与post form**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()    r.POST("/user/:id", func(c \*gin.Context) {  //取路径中的参数  id := c.Param("id")  //取post中的参数  name := c.PostForm("name")  age := c.PostForm("age")  c.JSON(http.*StatusOK*, gin.H{  "id": id,  "name": name,  "age": age,  })  })    r.Run()  } |

* **数据绑定**

1. **should bind(注意和mustBind的区别)**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    //required意味着必须包含被required修饰的数据  //form意味着必须有个form后面的key，如：id，name。  //lte小于等于，其余简写自行百度  type User struct {  ID string `form:"id" binding:"required"`  Name string `form:"name" binding:"required"`  }    func main() {  r := gin.Default()    r.GET("/user", func(c \*gin.Context) {  var u User  if err := c.ShouldBindQuery(&u); err != nil {  c.JSON(http.*StatusOK*, gin.H{  "code": 200,  "msg": err.Error(),  })  return  }  c.JSON(http.*StatusOK*, gin.H{  "msg": 0,  "data": u,  })  })    r.Run()  } |

1. **bind json**
2. **bind form**

|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )  //bind json  type User struct {  ID string `json:"id" binding:"required"`  Name string `json:"name" binding:"required"`  }  /\*  //bind form  type User struct {  ID string `form:"id" binding:"required"`  Name string `form:"name" binding:"required"`  }  \*/  func main() {  r := gin.Default()    r.POST("/user", func(c \*gin.Context) {  var u User  if err := c.ShouldBindJSON(&u); err != nil {  c.JSON(http.*StatusOK*, gin.H{  "code": 200,  "err": err.Error(),  })  return  }  c.JSON(http.*StatusOK*, gin.H{  "code": 0,  "data": u,  })  })    r.Run()  } |

1. **Only Bind Query String**

|  |
| --- |
| Go  package main    import "github.com/gin-gonic/gin"    type User struct {  ID string `form:"id" binding:"required,uuid"`  UserName string `form:"username" binding:"required,min=3"`  PassWord string `form:"password" binding:"required,min=3"`  }    func main() {  r := gin.Default()  r.POST("user", func(c \*gin.Context) {  var user User  if err := c.ShouldBindQuery(&user); err != nil {  c.JSON(200, gin.H{  "Code": 200,  "Msg": err.Error(),  })  return  }  c.JSON(200, gin.H{  "Code": 0,  "ID": user.ID,  })  })  r.Run()  } |

1. **Bind Query or Post Data**

|  |
| --- |
| Go  package main    import "github.com/gin-gonic/gin"    type User struct {  // application/json json  // application/x-www-form-urlencoded form  ID string `form:"id" binding:"required"`  }    func main() {  r := gin.Default()  r.POST("user", func(c \*gin.Context) {  var user User  if err := c.ShouldBindJSON(&user); err != nil {  c.JSON(200, gin.H{  "Code": 200,  "Msg": err.Error(),  })  return  }  c.JSON(200, gin.H{  "Code": 0,  "ID": user.ID,  })  })  r.Run()  } |

1. **Bind Uri**

|  |
| --- |
| Go  package main    import "github.com/gin-gonic/gin"    type User struct {  ID string `uri:"id" binding:"required,uuid"`  }    func main() {  r := gin.Default()  //http://localhost:8080/user/1234  r.POST("/user/:id", func(c \*gin.Context) {  var user User  if err := c.ShouldBindUri(&user); err != nil {  c.JSON(200, gin.H{  "Code": 200,  "Msg": err.Error(),  })  return  }  c.JSON(200, gin.H{  "Code": 0,  "ID": user.ID,  })  })  r.Run()  } |

1. **Bind header**

|  |
| --- |
| Go  package main    import "github.com/gin-gonic/gin"    type Header struct {  Referer string `header:"Referer" binding:"required"`  }    func main() {  r := gin.Default()  r.GET("/header", func(c \*gin.Context) {  var h Header  if err := c.ShouldBindHeader(&h); err != nil {  c.JSON(200, gin.H{  "Code": 200,  "Msg": err.Error(),  })  return  }  c.JSON(200, gin.H{  "Code": 0,  "ID": h.Referer,  })  })  r.Run()  } |

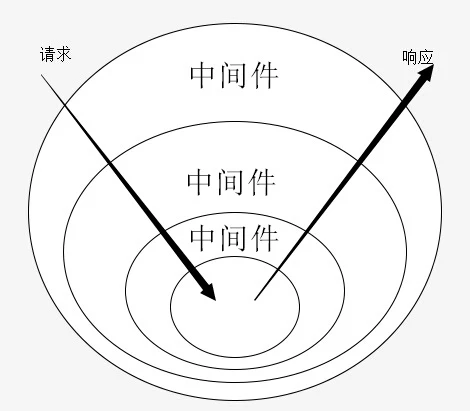
* **小练习**

//登录注册

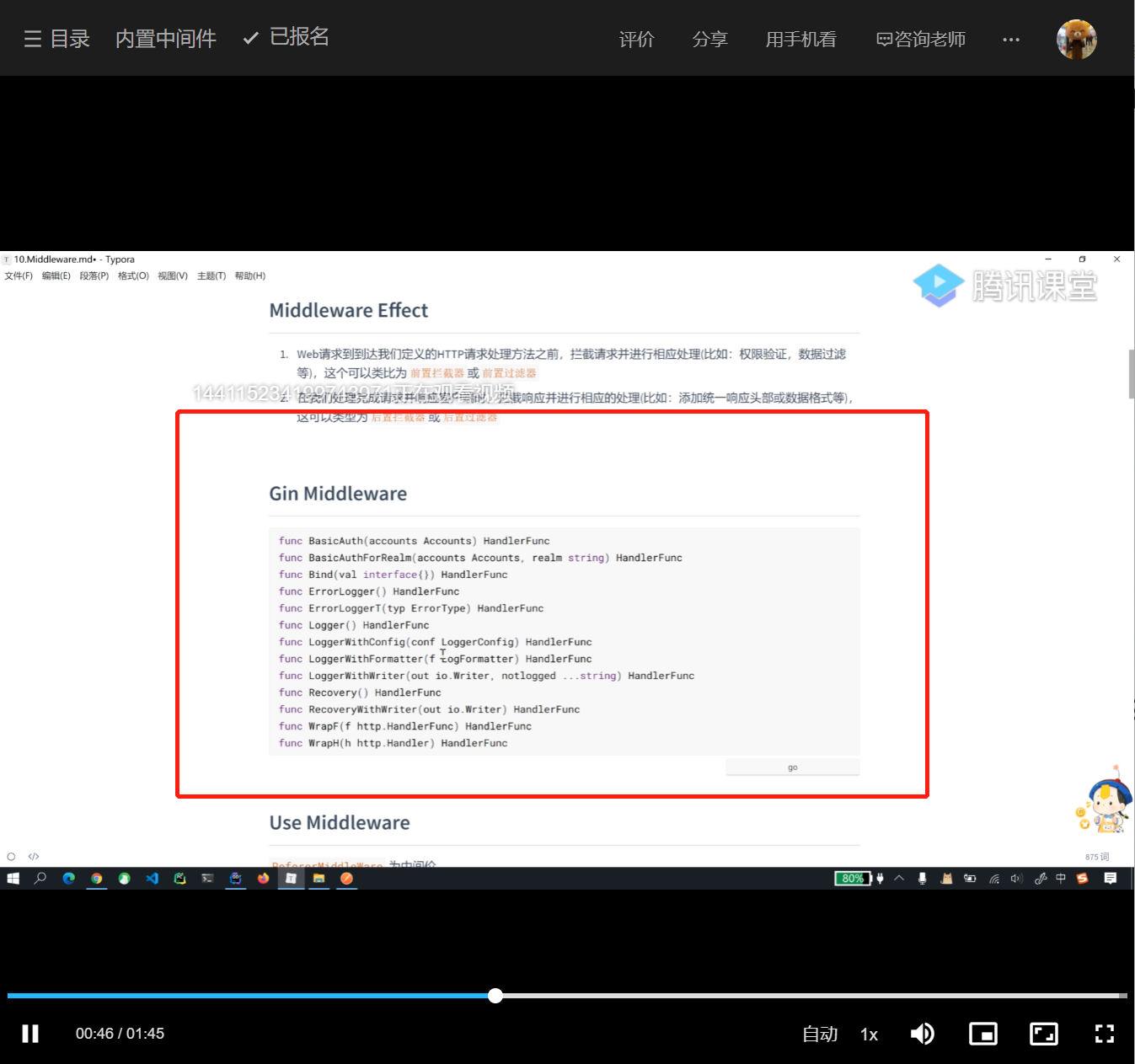
|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    type Login struct {  UserName string `json:"username" binding:"required,min=3"`  PassWord string `json:"password" binding:"required,len=8"`  RePassWord string `json:"repassword" binding:"required,len=8"`  }    type Register struct {  UserName string `json:"username" binding:"required,min=3"`  PassWord string `json:"password" binding:"required,len=8"`  Age int `json:"age" binding:"required,gte=1,lte=140"`  Sex int `json:"sex" binding:"required"`  Email string `json:"email" binding:"required,email"`  }    func main() {  r := gin.Default()    r.POST("/login", loginHandler)  r.POST("/register", registerHandler)  r.Run()  }    func registerHandler(c \*gin.Context) {  var r Register  if err := c.ShouldBindJSON(&r); err != nil {  c.JSON(http.*StatusOK*, gin.H{  "code": 200,  "err": err.Error(),  })  return  }  c.JSON(http.*StatusOK*, gin.H{  "code": 0,  "msg": "register succ",  "data": r.UserName,  })  }    func loginHandler(c \*gin.Context) {  var l Login  if err := c.ShouldBindJSON(&l); err != nil {  c.JSON(http.*StatusOK*, gin.H{  "code": 200,  "err": err.Error(),  })  return  }  c.JSON(http.*StatusOK*, gin.H{  "code": 0,  "data": l.UserName,  "msg": "login succ",  })  } |

* **中间件**

1. **洋葱模型**



1. **内置中间件**



1. **中间件的使用**

|  |
| --- |
| Go  package main    import (  "fmt"  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()  //全局调用  //r.Use(loginAuth)    r.GET("/ping", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "code": 0,  "msg": "pong",  })  })    r.POST("/login", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "data": "token",  })  })    r.POST("/register", func(c \*gin.Context) {    })    ////登录保护  //r.GET("user/:id", func(c \*gin.Context) {  //  //})  //  //r.PUT("/user/:id", func(c \*gin.Context) {  //  //})    //-->middleware-->core-->...  user := r.Group("/user", loginAuth)  {  user.GET("/:id", func(c \*gin.Context) {  fmt.Println("获取用户详情接口，需要登录保护")  c.JSON(http.*StatusOK*, gin.H{  "msg": "获取用户详情接口，需要登录保护",  })  })    user.PUT("/:id", func(c \*gin.Context) {  fmt.Println("更新用户详情接口，需要登录保护")  c.JSON(http.*StatusOK*, gin.H{  "msg": "更新用户详情接口，需要登录保护",  })  })  }    r.Run()  }    func loginAuth(c \*gin.Context) {  fmt.Println("我是登录保护中间件")  } |

1. **next函数**

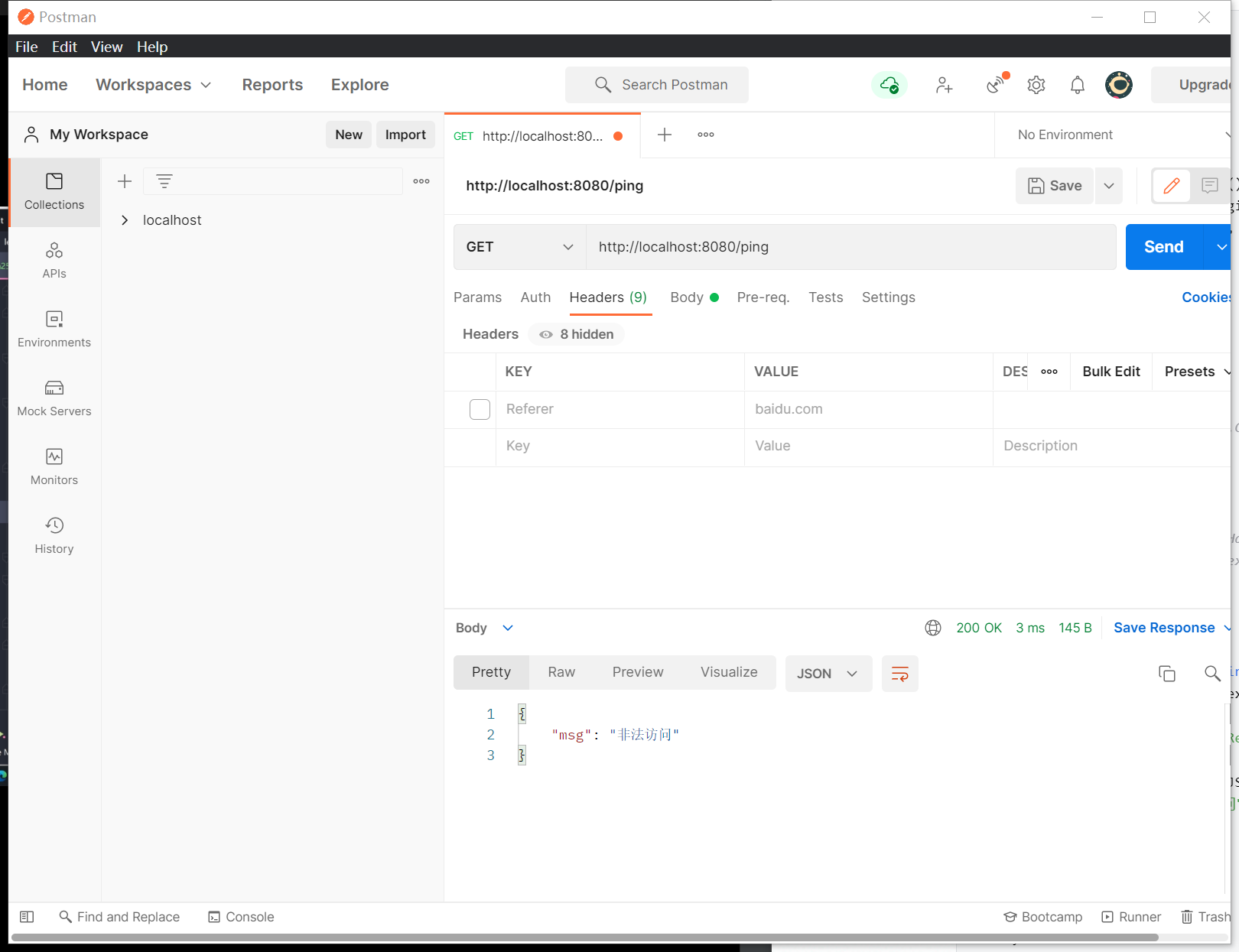
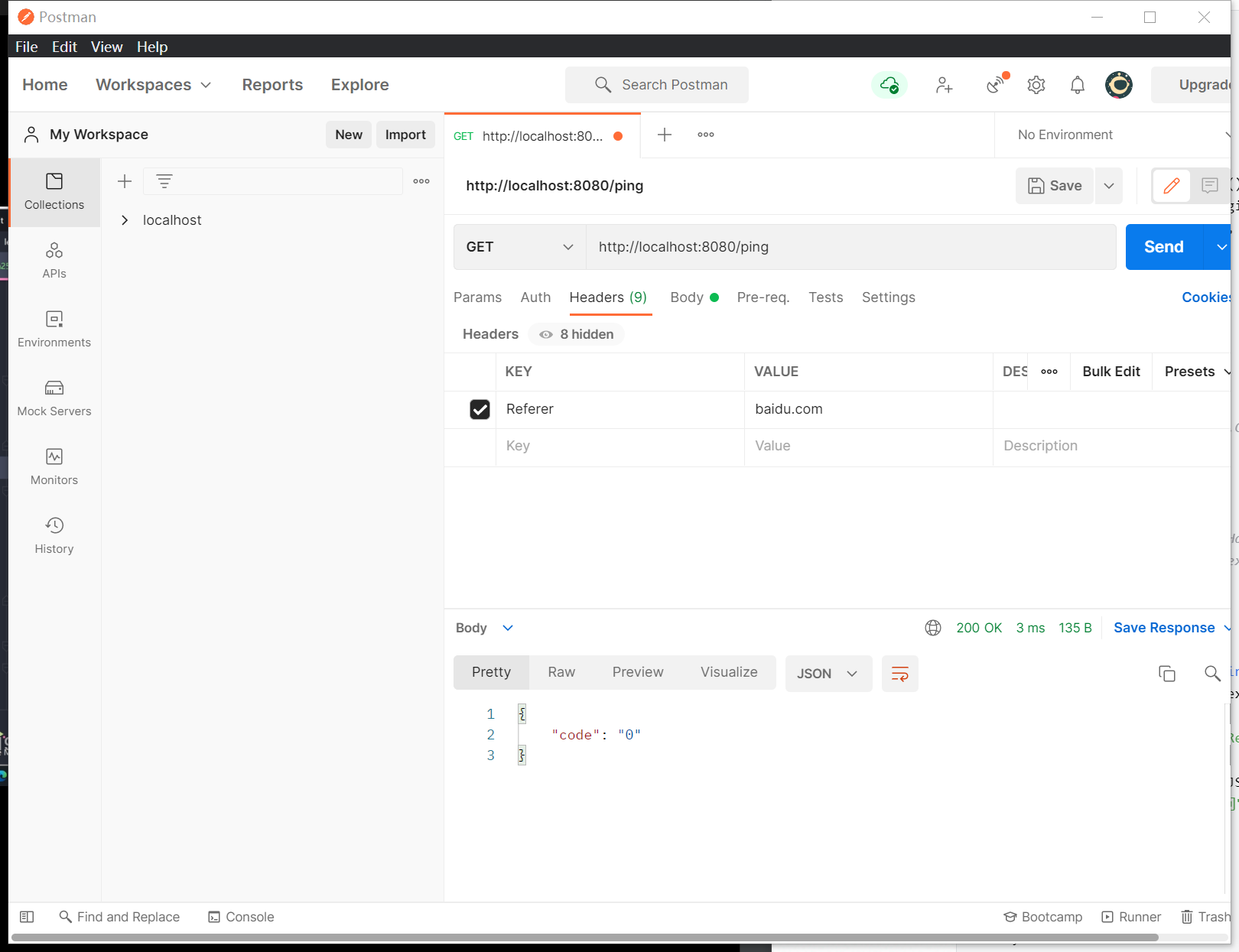
该函数顾名思义就是程序挂起来继续向下走，然后执行完成下面的函数，最后会反过来执行该中间件。

next函数对于单个中间件，程序再执行到next前一行，会暂时跳过下面得部分，再整体执行完之后，再继续执行之前next函数剩下得部分。如果是对于多个中间件，那么就是栈得先进后出原则，先执行next函数得中间件会再最后被执行。

|  |
| --- |
| Go  package main    import (  "log"  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()  r.Use(myMiddleware1)  r.GET("/ping", func(c \*gin.Context) {  k := c.GetInt("key")  c.Set("key", k+2000)  log.Println("will enter func myMiddleware()")  c.JSON(http.*StatusOK*, gin.H{  "key": c.GetInt("key"),  })  })  r.Run()  }    func myMiddleware1(c \*gin.Context) {  log.Println("middleware in")  c.Set("key", 1000)  log.Println("before func Next()")  c.Next()  log.Println("func Next after")  log.Println("middlerware done")  } |

1. **自定义中间件(custom middleware)**

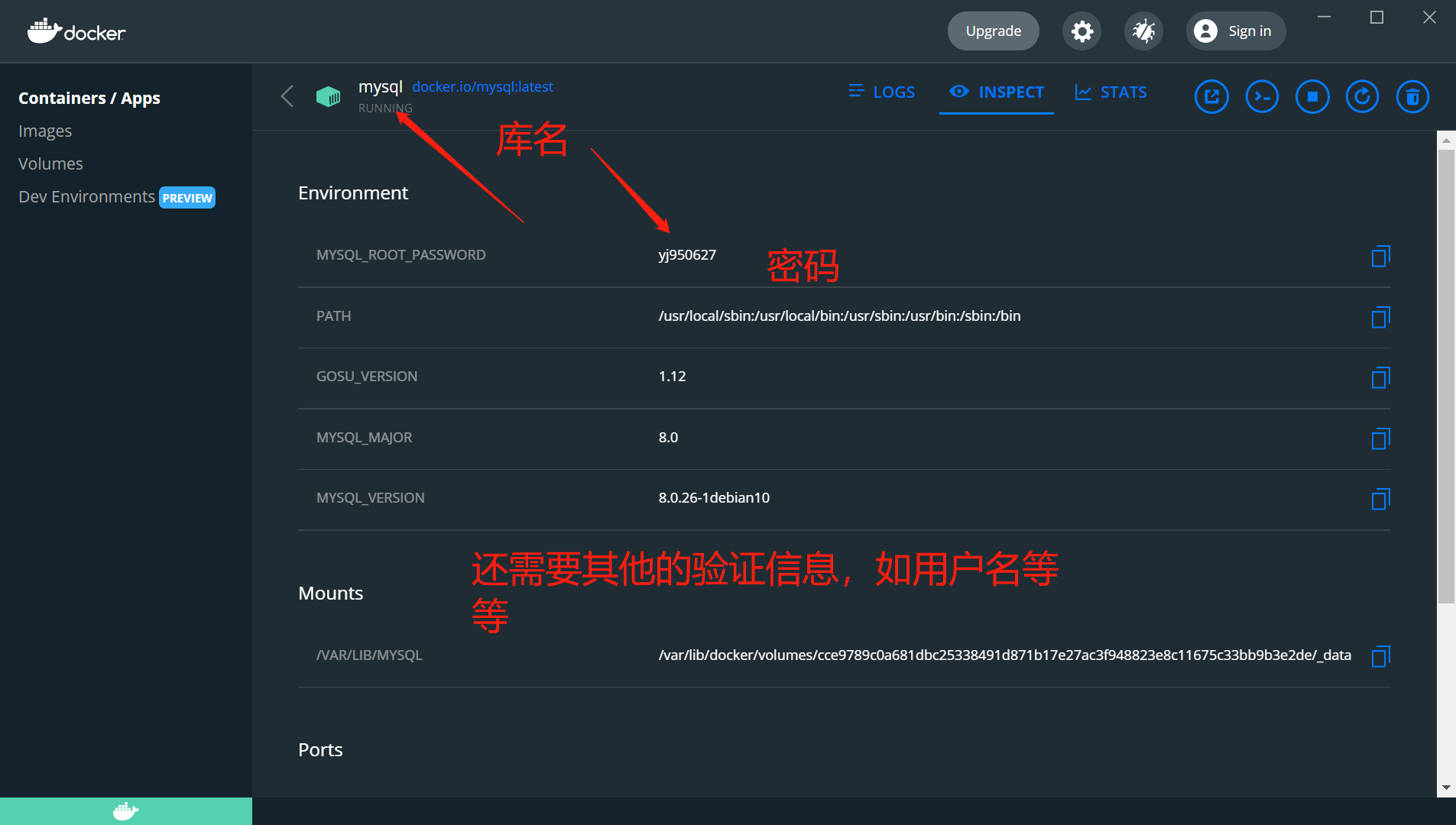
|  |
| --- |
| Go  package main    import (  "net/http"    "github.com/gin-gonic/gin"  )    func main() {  r := gin.Default()  //r.Use(middlerware1)  //r.Use(middlerware2())    r.Use(RefererMiddleware())  r.GET("/ping", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "code": "0",  })  })  r.Run()  }    ////自定义中间件  ////第一种方式  //func middlerware1(c \*gin.Context) {  //  //}  //  ////第二种方式：闭包  //func middlerware2() gin.HandlerFunc {  // return func(c \*gin.Context) {  // //域名 \*.baidu.com  // }  //}    func RefererMiddleware() gin.HandlerFunc {  return func(c \*gin.Context) {  //取referer  ref := c.GetHeader("Referer")  if ref == "" {  c.AbortWithStatusJSON(http.*StatusOK*, gin.H{  "msg": "非法访问",  })  return  }  c.Next()  }  } |



* **Mysql－ｓｑｌ**

1. **链接MySQL**

|  |
| --- |
| Go  package main    import (  "database/sql"  "fmt"    \_ "github.com/go-sql-driver/mysql"  )    func main() {  db, err := sql.Open("mysql", "root:yj950627@tcp(localhost:3306)/mysql")  //验证  if err != nil {  fmt.Println("validate failed")  panic(err)  }  defer db.Close()  //尝试建立连接  if err := db.Ping(); err != nil {  fmt.Println("connect to database failed")  panic(err)  }  db.SetMaxOpenConns(10) //设置最大链接数  db.SetMaxIdleConns(10) //设置最大闲置链接数  fmt.Println("connect to database succ")    } |



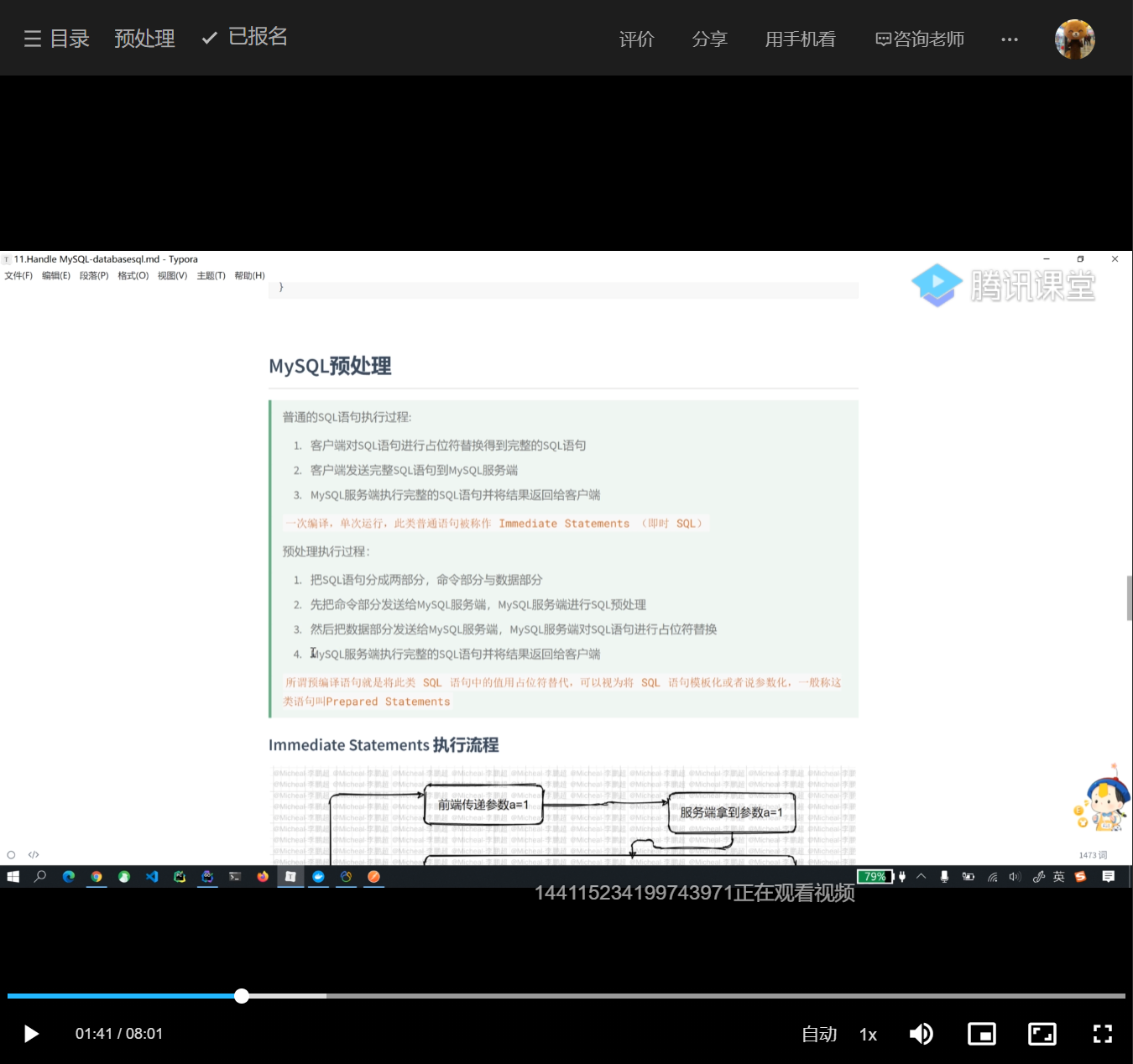
1. **封装链接**

|  |
| --- |
| Go  package main    import (  "database/sql"  "fmt"    \_ "github.com/go-sql-driver/mysql"  )    var db \*sql.DB    func initializeDatabase() (err error) {  dsn := "root:yj950627@tcp(localhost:3306)/mysql?charset=utf8mb4&parseTime=True&loc=Local"  db, err = sql.Open("mysql", dsn)  if err != nil {  panic(err)  return  }  if err = db.Ping(); err != nil {  return  }  return nil  }    func main() {  if err := initializeDatabase(); err != nil {  panic(err)  }  db.SetMaxOpenConns(10) //设置最大链接数  db.SetMaxIdleConns(10) //设置最大闲置链接数  fmt.Println("connect to database succ")    } |

1. **查询单行数据**
2. **查询多行数据**
3. **删除数据**

|  |
| --- |
| Go  package main    import (  "database/sql"  "fmt"  "net/http"    "github.com/gin-gonic/gin"    \_ "github.com/go-sql-driver/mysql"  )    type User struct {  ID int  Name string  Age int  }    var db \*sql.DB    //查询单行的数据  func queryStringRow() (u User) {  sqlStr := "select \* from student where id = ?"  //查询单行  if err := db.QueryRow(sqlStr, 1).Scan(&u.ID, &u.Name, &u.Age); err != nil {  fmt.Println(err.Error())  return  }  fmt.Println("user data:", u)  return  }    //查询多行数据  func queryMultiRow() []User {  sqlStr := "select \* from student"  rows, err := db.Query(sqlStr)  if err != nil {  fmt.Println(err)  return nil  }  defer rows.Close()  users := make([]User, 0)  for rows.Next() {  var u User  err := rows.Scan(&u.ID, &u.Name, &u.Age)  if err != nil {  fmt.Println(err)  return nil  }  users = append(users, u)  }  return users  }    //更新数据  func updateRow() {  sqlStr := "update student set name = ? where id =?"  result, err := db.Exec(sqlStr, "华夏", 3)  if err != nil {  fmt.Println(err)  return  }  n, err := result.RowsAffected()  if err != nil {  fmt.Println("get rows affected failed", err)  return  }  fmt.Println("update succ,effected row", n)    }    //删除数据  func deleteRow() {  sqlStr := "delete from student where id =?"  result, err := db.Exec(sqlStr, 3)  if err != nil {  fmt.Println(err)  return  }  n, err := result.RowsAffected()  if err != nil {  fmt.Println("get rows affected failed ", err)  return  }  fmt.Println("get rows affected succ ", n)    }    //插入数据  func insertRow() {  sqlStr := "insert into student(name,age) values(?,?)"  result, err := db.Exec(sqlStr, "中国", 5000)  if err != nil {  fmt.Println("insert data failed", err)  return  }    id, err := result.LastInsertId()  if err != nil {  fmt.Println("get last insert id failed", err)  return  }  fmt.Println("inset succ", id)  }    func initializeDatabase() (err error) {  dsn := "root:yj950627@tcp(localhost:3306)/student?charset=utf8mb4&parseTime=True&loc=Local"  db, err = sql.Open("mysql", dsn)  if err != nil {  panic(err)  return  }  if err = db.Ping(); err != nil {  return  }  return nil  }    func main() {  if err := initializeDatabase(); err != nil {  panic(err)  }  db.SetMaxOpenConns(10) //设置最大链接数  db.SetMaxIdleConns(10) //设置最大闲置链接数  fmt.Println("connect to database succ")    r := gin.Default()  //查询用户  r.GET("/ping", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "data": queryStringRow(),  })  })  //查询多行数据  r.GET("/users", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "data": queryMultiRow(),  })  })  //更新数据  updateRow()  r.GET("/update", func(c \*gin.Context) {  c.JSON(http.*StatusOK*, gin.H{  "data": queryMultiRow(),  })  })  //删除数据  deleteRow()  //插入数据  insertRow()  r.Run()    } |

1. **预处理**



1. **go实现mysql预处理**

|  |
| --- |
| Go  package main    import (  "database/sql"  "fmt"    "github.com/gin-gonic/gin"    \_ "github.com/go-sql-driver/mysql"  )    type User struct {  ID int  Name string  Age int  }  type userForm struct {  Name string `form:"name" binding:"required"`  }    var db \*sql.DB    //查询多行数据  func queryMultiRow(name string) []User {  sqlStr := "select \* from student where name = ?"  //语句部分  stmt, err := db.Prepare(sqlStr) //发送给mysql服务端，之后编译这个模板语句  if err != nil {  fmt.Println(err)  return nil  }  defer stmt.Close()  //数据部分  rows, \_ := stmt.Query(name) //将参数拼接再模板上执行，这里不会再次编译  defer rows.Close()  users := make([]User, 0)  for rows.Next() {  var u User  err := rows.Scan(&u.ID, &u.Name, &u.Age)  if err != nil {  fmt.Println(err)  return nil  }  users = append(users, u)  }  return users  }    func insertRow() {  sqlStr := "insert into student(name,age) value(?,?)"  stmt, err := db.Prepare(sqlStr)  if err != nil {  fmt.Println(err)  return  }  defer stmt.Close()  \_, err = stmt.Exec("wx1", 31)  if err != nil {  fmt.Println("insert failed1", err)  return  }    \_, err = stmt.Exec("wx2", 32)  if err != nil {  fmt.Println("insert failed2", err)  return  }  fmt.Println("insert succ")  }    func initializeDatabase() (err error) {  dsn := "root:yj950627@tcp(localhost:3306)/student?charset=utf8mb4&parseTime=True&loc=Local"  db, err = sql.Open("mysql", dsn)  if err != nil {  panic(err)  return  }  if err = db.Ping(); err != nil {  return  }  return nil  }    func main() {  if err := initializeDatabase(); err != nil {  panic(err)  }  db.SetMaxOpenConns(10) //设置最大链接数  db.SetMaxIdleConns(10) //设置最大闲置链接数  fmt.Println("connect to database succ")  //预处理，插入数据  insertRow()  r := gin.Default()  r.GET("/users", func(c \*gin.Context) {  var u userForm  if err := c.ShouldBind(&u); err != nil {  c.JSON(200, gin.H{  "err": err.Error(),  })  return  }  //预处理，请求多行数据  data := queryMultiRow(u.Name)  c.JSON(200, gin.H{  "data": data,  })  })  r.Run()  } |

* **Mysql-sqlx**

1. **使用sqlx链接MySQL**

|  |
| --- |
| Go  package main    import (  "fmt"    \_ "github.com/go-sql-driver/mysql"  "github.com/jmoiron/sqlx"  \_ "github.com/jmoiron/sqlx"  )    var db \*sqlx.DB    func initializeDatabase() (err error) {  dsn := "root:yj950627@tcp(localhost:3306)/student?charset=utf8mb4&parseTime=True&loc=Local"  db, err = sqlx.Connect("mysql", dsn)  if err != nil {  panic(err)  return  }  db.SetMaxOpenConns(10) //最大链接数  db.SetMaxIdleConns(10) //最大闲置链接数  fmt.Println("connect 2 database succ")    return nil  }    func main() {  if err := initializeDatabase(); err != nil {  panic(err)  }  }    /\*  //resource code  // Connect to a database and verify with a ping.  func Connect(driverName, dataSourceName string) (\*DB, error) {  db, err := Open(driverName, dataSourceName)  if err != nil {  return nil, err  }  err = db.Ping()  if err != nil {  db.Close()  return nil, err  }  return db, nil  }\*/ |

1. **使用sqlx查询单条数据**
2. **使用sqlx查询多条数据**
3. **使用sqlx更新数据**
4. **使用sqlx插入数据**
5. **使用sqlx删除数据**

|  |
| --- |
| Go  package main    import (  "fmt"    \_ "github.com/go-sql-driver/mysql"  "github.com/jmoiron/sqlx"  \_ "github.com/jmoiron/sqlx"  )    type User struct {  ID int  Name string  Age int  }    var db \*sqlx.DB    func initializeDatabase() (err error) {  dsn := "root:yj950627@tcp(localhost:3306)/student?charset=utf8mb4&parseTime=True&loc=Local"  db, err = sqlx.Connect("mysql", dsn)  if err != nil {  panic(err)  return  }  db.SetMaxOpenConns(10) //最大链接数  db.SetMaxIdleConns(10) //最大闲置链接数  fmt.Println("connect 2 database succ")    return nil  }    //查询单行数据  func queryRow() {  sqlStr := "select \* from student where id = ?"  var u User  if err := db.Get(&u, sqlStr, 6); err != nil {  fmt.Println("query failed", err)  return  }  fmt.Println(u)  }    //查询多行数据  func queryMultiRow() []User {  sqlStr := "select \* from student"  var users []User  if err := db.Select(&users, sqlStr); err != nil {  fmt.Println(err)  return nil  }  fmt.Println(users)  return users    }    //更新数据  func updateRow() {  sqlStr := "update student set age = ? where id =?"  res, err := db.Exec(sqlStr, 55, 1)  if err != nil {  fmt.Println(err)  return  }  n, err := res.RowsAffected()  if err != nil {  fmt.Println(err)  return  }  fmt.Println("update succ", n)  }    //插入数据  func insertRow() {  sqlStr := "insert into student (name,age) values(?,?)"  res, err := db.Exec(sqlStr, "wwh", 25)  if err != nil {  fmt.Println("insert failed ", err)  return  }  id, err := res.LastInsertId()  if err != nil {  fmt.Println("get last insert id failed", err)  return  }  fmt.Println("insert succ", id)  }    //删除数据  func deleteRow() {  sqlStr := "delete from student where id = ?"  res, err := db.Exec(sqlStr, 8)  if err != nil {  fmt.Println("delete failed", err)  return  }  n, err := res.RowsAffected()  if err != nil {  fmt.Println("get rows affected failed", err)  return  }  fmt.Println("delete succ", n)  }    func main() {  if err := initializeDatabase(); err != nil {  panic(err)  }  ////查询单行数据  queryRow()  //查询多行数据  queryMultiRow()  //更新数据  updateRow()  //插入数据  insertRow()  //删除数据  deleteRow()  } |

1. **sqlx named query**
2. **sqlx named exec**

|  |
| --- |
| Go  package main    import (  "fmt"    \_ "github.com/go-sql-driver/mysql"  "github.com/jmoiron/sqlx"  \_ "github.com/jmoiron/sqlx"  )    type User struct {  ID int  Name string  Age int  }    var db \*sqlx.DB    func initializeDatabase() (err error) {  dsn := "root:yj950627@tcp(localhost:3306)/student?charset=utf8mb4&parseTime=True&loc=Local"  db, err = sqlx.Connect("mysql", dsn)  if err != nil {  panic(err)  return  }  db.SetMaxOpenConns(10) //最大链接数  db.SetMaxIdleConns(10) //最大闲置链接数  fmt.Println("connect 2 database succ")    return nil  }    //查询获取数据库中年龄为25的全部用户信息  func selectNameQuery() {  condition := User{  Age: 25,  }  sqlStr := "select \* from student where age = :age"  rows, err := db.NamedQuery(sqlStr, condition)  if err != nil {  fmt.Println("named query failed ", err)  return  }  defer rows.Close()  for rows.Next() {  var u User  if err := rows.StructScan(&u); err != nil {  fmt.Println("structsacn failed", err)  continue  }  fmt.Println(u)  }  }    //批量插入  func batchInsert() {  users := []User{  {Name: "a", Age: 1},  {Name: "b", Age: 2},  {Name: "c", Age: 3},  }  sqlStr := "insert into student (name,age) values(:name,:age )"  \_, err := db.NamedExec(sqlStr, users)  if err != nil {  fmt.Println(err)  return  }  fmt.Println("批量插入成功")  }    func main() {  if err := initializeDatabase(); err != nil {  panic(err)  }  selectNameQuery()  batchInsert()  } |

* **redis**

1. **使用go-redis链接**

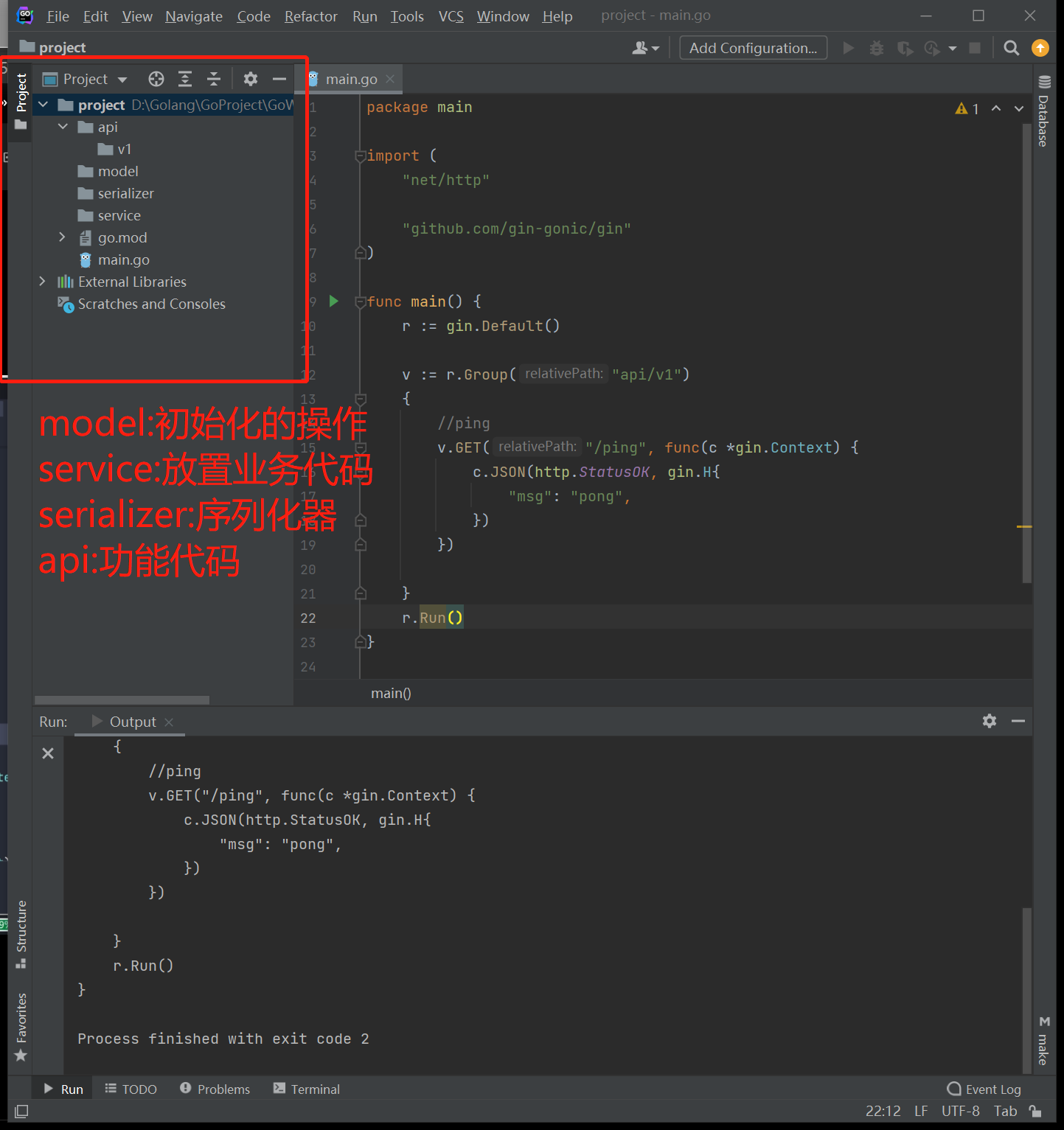
|  |
| --- |
| Go  package main    import (  "context"  "fmt"    "github.com/go-redis/redis/v8"  \_ "github.com/go-redis/redis/v8"  \_ "github.com/go-sql-driver/mysql"  \_ "github.com/jmoiron/sqlx"  )    var ctx = context.Background()  var rdb \*redis.Client    func initializeRedisClient() (err error) {  rdb = redis.NewClient(&redis.Options{  Addr: "localhost:6379",  Password: "",  DB: 0,  //连接池大小  PoolSize: 100,  })  \_, err = rdb.Ping(ctx).Result()  return  }    func main() {  if err := initializeRedisClient(); err != nil {  panic(err)  }  fmt.Println("connect 2 redis database succ")  } |

1. **Redis-go-redis基本使用**

|  |
| --- |
| Go  package main    import (  "context"  "fmt"    "github.com/go-redis/redis/v8"  \_ "github.com/go-redis/redis/v8"  \_ "github.com/go-sql-driver/mysql"  \_ "github.com/jmoiron/sqlx"  )    var ctx = context.Background()  var rdb \*redis.Client    func initializeRedisClient() (err error) {  rdb = redis.NewClient(&redis.Options{  Addr: "localhost:6379",  Password: "",  DB: 0,  //连接池大小  PoolSize: 100,  })  \_, err = rdb.Ping(ctx).Result()  return  }    func main() {  if err := initializeRedisClient(); err != nil {  panic(err)  }  fmt.Println("connect 2 redis database succ")    //set  err := rdb.Set(ctx, "name", "value1", 0).Err()  if err != nil {  panic(err)  }  //get  val, err := rdb.Get(ctx, "name").Result()  if err != nil {  panic(err)  }  fmt.Println("name", val)  } |

* **综合项目练习**

1. **项目结构**



1. **工程源码**

D:\Golang\GoProject\GoWeb-restart