**基于gin框架搭建的一个简单的web服务**

刚把go编程基础知识学习完了，学习的时间很短，可能还有的没有完全吸收。不过还是在项目中发现知识，然后在去回顾已学的知识，现在利用gin这个web框架做一个简单的CRUD操作。

**1.Go Web框架的技术选型**

[Top 6 web frameworks for Go as of 2017](https://blog.usejournal.com/top-6-web-frameworks-for-go-as-of-2017-23270e059c4b)，可以看看这个go语言中Web框架的对比和老大的推荐，我选择gin框架作为学习go语言的框架。



image.png

**2.Gin介绍**

[gin框架的中文文档](https://github.com/skybebe/gin-doc-cn)：<https://github.com/skyhee/gin-doc-cn>这个文档相当好，清晰明了解释gin框架的整个用法。下面是gin框架的全部特性：

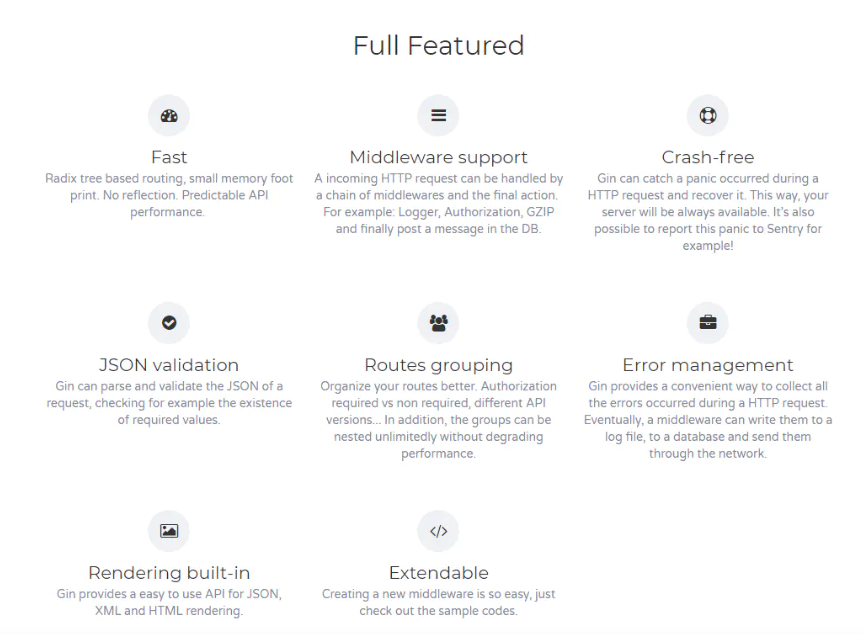


image.png

**3.建立一个web服务实现Hello Word**

首先需要安装ginweb框架和gorm作为ORM层，然后利用go-sql数据库驱动，前提是要配置Gopath环境变量，使用go get安装即可：[gin安装地址](https://github.com/gin-gonic/gin" \t "_blank)：<https://github.com/gin-gonic/gin>

//使用github上的gin托管地址

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

$ go get -u github.com/jinzhu/gorm

$ go get github.com/go-sql-driver/mysql

使用Gin实现Hello world非常简单，创建一个router（路由），然后执行Run方法即可：

package main

import (

"github.com/gin-gonic/gin"

"net/http"

)

func main() {

router:=gin.Default()

router.GET("/", func(c \*gin.Context) {

c.String(http.StatusOK,"Hello World")

})

router.Run(":8080")

}

结果如下：

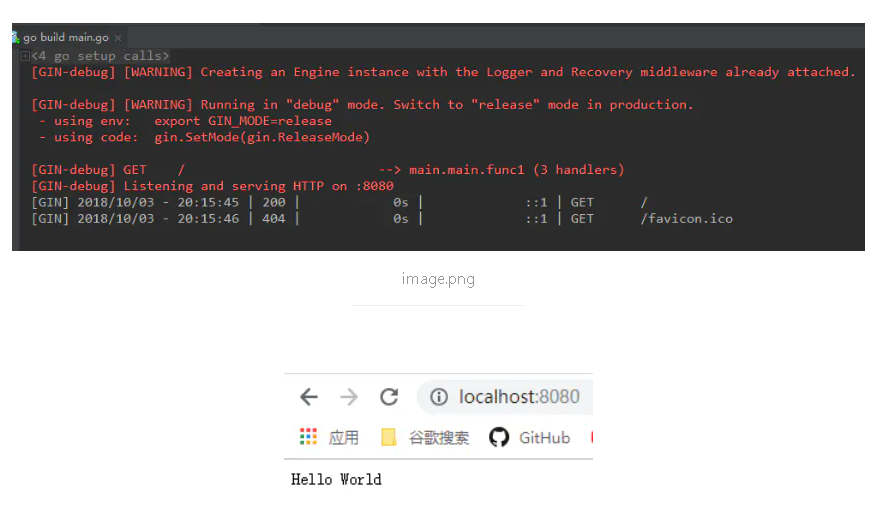


image.pn

是不是比Java方便多了！下面来仔细分析下上面的代码结构：

* 1、router:=gin.Default()：这是默认的服务器。使用gin的Default方法创建一个路由Handler；
* 2、然后通过Http方法绑定路由规则和路由函数。不同于net/http库的路由函数，gin进行了封装，把request和response都封装到了gin.Context的上下文环境中。
* 3、最后启动路由的Run方法监听端口。还可以用http.ListenAndServe(":8080", router)，或者自定义Http服务器配置。

**4.基于database/sql的CURD操作**

**1.建立数据库表**

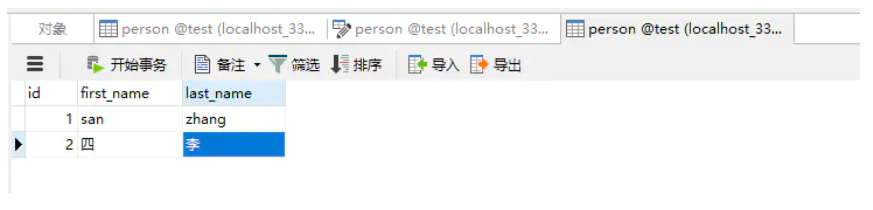


image.png

**2.CURD操作，以查询为例**

参考[人世间](https://gist.github.com/rsj217/26492af115a083876570f003c64df118" \t "_blank)：<https://gist.github.com/rsj217/26492af115a083876570f003c64df118>

package main

import (

"database/sql"

\_ "github.com/go-sql-driver/mysql"

"log"

"net/http"

"github.com/gin-gonic/gin"

)

//利用的是原生的SQL的ORM层

var db \*sql.DB

//定义person类型结构

type Person struct {

Id int `json:"id"`

FirstName string `json:"first\_name"`

LastName string `json:"last\_name"`

}

//定义一个getALL函数用于回去全部的信息

func (p Person) getAll() (persons []Person,err error) {

rows,err:=db.Query("SELECT id, first\_name, last\_name FROM person")

if err!=nil {

return

}

for rows.Next() {

var person Person

//遍历表中所有行的信息

rows.Scan(&person.Id,&person.FirstName,&person.LastName)

//将person添加到persons中

persons=append(persons,person)

}

//最后关闭连接

defer rows.Close()

return

}

func main() {

var err error

db,err =sql.Open("mysql","root:123456@tcp(127.0.0.1:3306)/test?parseTime=true")

//错误检查

if err!=nil{

log.Fatal(err.Error())

}

//推迟数据库连接的关闭

defer db.Close()

err = db.Ping()

if err != nil {

log.Fatal(err.Error())

}

//创建一个路由Handler

router:=gin.Default()

//get方法的查询

router.GET("/person", func(c \*gin.Context) {

p:=Person{}

persons,err:= p.getAll()

if err !=nil {

log.Fatal(err)

}

//H is a shortcut for map[string]interface{}

c.JSON(http.StatusOK,gin.H{

"result":persons,

"count":len(persons),

})

})

router.Run(":8080")

}

**3.测试结果如下**

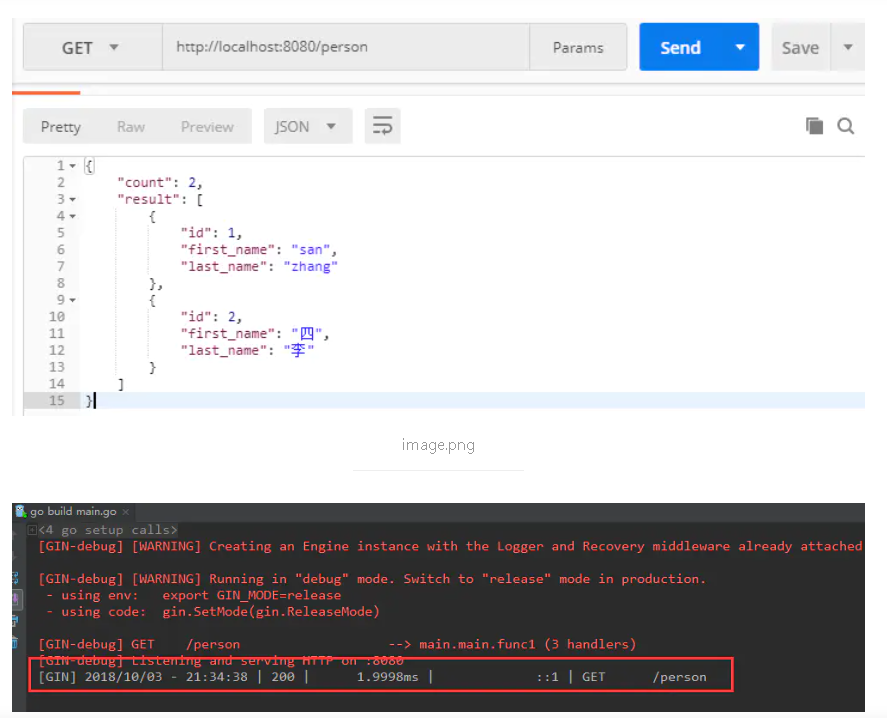


image.png

完整代码如下：

package main

import (

"database/sql"

\_ "github.com/go-sql-driver/mysql"

"log"

"net/http"

"github.com/gin-gonic/gin"

"strconv"

"fmt"

)

var db \*sql.DB

//定义person类型结构

type Person struct {

Id int `json:"id"`

FirstName string `json:"first\_name"`

LastName string `json:"last\_name"`

}

//定义一个getALL函数用于回去全部的信息

func (p Person) getAll() (persons []Person, err error) {

rows, err := db.Query("SELECT id, first\_name, last\_name FROM person")

if err != nil {

return

}

for rows.Next() {

var person Person

//遍历表中所有行的信息

rows.Scan(&person.Id, &person.FirstName, &person.LastName)

//将person添加到persons中

persons = append(persons, person)

}

//最后关闭连接

defer rows.Close()

return

}

//通过id查询

func (p Person) get() (person Person, err error) {

row := db.QueryRow("SELECT id, first\_name, last\_name FROM person WHERE id=?", p.Id)

err = row.Scan(&person.Id, &person.FirstName, &person.LastName)

if err != nil {

return

}

return

}

func (p Person) add() (Id int, err error) {

stmt, err := db.Prepare("INSERT INTO person(first\_name, last\_name) VALUES (?, ?)")

if err != nil {

return

}

//执行插入操作

rs, err := stmt.Exec(p.FirstName, p.LastName)

if err != nil {

return

}

//返回插入的id

id, err := rs.LastInsertId()

if err != nil {

log.Fatalln(err)

}

//将id类型转换

Id = int(id)

defer stmt.Close()

return

}

//通过id删除

func (p Person) del() (rows int, err error) {

stmt, err := db.Prepare("DELETE FROM person WHERE id=?")

if err != nil {

log.Fatalln(err)

}

rs, err := stmt.Exec(p.Id)

if err != nil {

log.Fatalln(err)

}

//删除的行数

row, err := rs.RowsAffected()

if err != nil {

log.Fatalln(err)

}

defer stmt.Close()

rows = int(row)

return

}

func main() {

var err error

db, err = sql.Open("mysql", "root:123456@tcp(127.0.0.1:3306)/test?parseTime=true")

//错误检查

if err != nil {

log.Fatal(err.Error())

}

//推迟数据库连接的关闭

defer db.Close()

err = db.Ping()

if err != nil {

log.Fatal(err.Error())

}

//创建一个路由Handler

router := gin.Default()

//get方法的查询

router.GET("/person", func(c \*gin.Context) {

p := Person{}

persons, err := p.getAll()

if err != nil {

log.Fatal(err)

}

//H is a shortcut for map[string]interface{}

c.JSON(http.StatusOK, gin.H{

"result": persons,

"count": len(persons),

})

})

//利用get方法通过id查询

router.GET("/person/:id", func(c \*gin.Context) {

var result gin.H

//c.Params方法可以获取到/person/:id中的id值

id := c.Param("id")

Id, err := strconv.Atoi(id)

if err != nil {

log.Fatal(err)

}

//定义person结构

p := Person{

Id: Id,

}

person, err := p.get()

if err != nil {

result = gin.H{

"result": nil,

"count": 0,

}

} else {

result = gin.H{

"result": person,

"count": 1,

}

}

c.JSON(http.StatusOK, result)

})

//利用post方法新增数据

router.POST("/person", func(c \*gin.Context) {

var p Person

err := c.Bind(&p)

if err != nil {

log.Fatal(err)

}

Id, err := p.add()

fmt.Print("id=", Id)

name := p.FirstName + " " + p.LastName

c.JSON(http.StatusOK, gin.H{

"message": fmt.Sprintf("%s 插入成功", name),

})

})

//利用DELETE请求方法通过id删除

router.DELETE("/person/:id", func(c \*gin.Context) {

id := c.Param("id")

Id, err := strconv.ParseInt(id, 10, 10)

if err != nil {

log.Fatalln(err)

}

p := Person{Id: int(Id)}

rows, err := p.del()

if err != nil {

log.Fatalln(err)

}

fmt.Println("delete rows ", rows)

c.JSON(http.StatusOK, gin.H{

"message": fmt.Sprintf("Successfully deleted user: %s", id),

})

})

router.Run(":8080")

}

下一篇博客在学习下实际项目中的基于gorm的ORM层的SQL写法，以及规范的项目结构分层！

**参考资料**

<https://godoc.org/github.com/gin-gonic/gin>

<https://gist.github.com/rsj217/26492af115a083876570f003c64df118>