Goland之框架beego

一、MVC思想

```
M-------> Model：模型
V-------> View：视图
C-------> Controller：控制器
```

二、beego之路由

2.1、静态路由

- beego.Get("自定义路径","处理器函数（或者处理器）")
- 请求方式：GET、POST等

```
// 导入beego的包
import (
    "fmt"

    "github.com/astaxie/beego/context" // 导入beego的包

    "github.com/astaxie/beego" // 导入beego的包
)

func main() {

    // 路由绑定函数，  路径，处理器函数（处理器）
    // 以GET方式请求，通过绑定函数处理
    beego.Get("/", func(ctx *context.Context) {

        // 用户数据的获取
        name := ctx.Input.Query("name")

        // 给用户响应数据
        ctx.Output.Context.WriteString(fmt.Sprintf("你输入的名字是：%s", name))
    })

    // 路由绑定函数，  路径，处理器函数（处理器）
    // 以POST方式请求，通过绑定函数处理
    beego.Post("/", func(ctx *context.Context) {
        // 用户数据的获取
        name := ctx.Input.Query("name")

        // 给用户响应数据
        ctx.Output.Context.WriteString(fmt.Sprintf("(POST)你输入的名字是：%s",
name))
    })

    // Any函数任意请求都可以处理
    beego.Any("/any", func(ctx *context.Context) {
        // 用户数据的获取
```

```
        name := ctx.Input.Query("name")

        // 给用户响应数据,ctx.Input.Method() 获取请求方式
        ctx.Output.Context.WriteString(fmt.Sprintf("你输入的名字是：%s", name))
        ctx.Output.Context.WriteString(fmt.Sprintf("请求方式是：%s",
ctx.Input.Method()))
    })

    // 启动beego
    beego.Run()
}
```

2.2、动态路由（正则路由）

- 匹配 /数字/ 格式的路由。并且把匹配到的值放入:id参数中
  - beego.Get("/:id(\d+)/", func(ctx *context.Context))
- 匹配任意的格式
  - Get("/any/:context/",func)
- 匹配一个文件
  - beego.Get("/file/*",func())
  - http://localhost:8080/file/xxx.aaa
- 匹配ext
  - beego.Get("/file/*", func()
  - http://localhost:8080/file/xxx.aaa

```
// 导入beego的包
import (
    "github.com/astaxie/beego"
    "github.com/astaxie/beego/context"
)

func main() {

    // 正则路由
    // 匹配 /数字/ 格式的路由。并且把匹配到的值放入:id参数中
    beego.Get("/:id(\\d+)/", func(ctx *context.Context) {
        ctx.WriteString("匹配")
    })

    // 匹配任意的格式
    beego.Get("/any/:context/", func(ctx *context.Context) {
        ctx.WriteString("匹配context")
    })

    // 匹配一个文件，http://localhost:8080/file/xxx.aaa
    beego.Get("/file/*", func(ctx *context.Context) {
        ctx.WriteString("匹配file")
    })

    // 匹配ext，http://localhost:8080/ext/xxx.aaa
    beego.Get("/ext/*.*", func(ctx *context.Context) {
        ctx.WriteString("匹配ext")
    })

    // 启动beego
```

```
    beego.Run()
}
```

2.3、url参数获取（少用，不建议用。不方便做权限控制）

```
package main

// 导入beego的包
import (
    "fmt"

    "github.com/astaxie/beego"
    "github.com/astaxie/beego/context"
)

func main() {

    // 正则路由
    // 匹配 /数字/ 格式的路由。并且把匹配到的值放入:id参数中
    beego.Get("/:id(\\d+)/", func(ctx *context.Context) {
        id := ctx.Input.Param(":id")
        ctx.WriteString(fmt.Sprintf("匹配的id：%s", id))
    })
    beego.Get("/:name(\\w+)/", func(ctx *context.Context) {
        name := ctx.Input.Param(":name")
        ctx.WriteString(fmt.Sprintf("匹配的name：%s", name))
    })

    // 匹配任意的格式
    beego.Get("/any/:context/", func(ctx *context.Context) {
        context := ctx.Input.Param(":id")
        ctx.WriteString(fmt.Sprintf("匹配的context：%s", context))
    })

    // 匹配一个文件，http://localhost:8080/file/xxx.aaa
    beego.Get("/file/*", func(ctx *context.Context) {
        // * 怎么表示呢？用splat
        splat := ctx.Input.Param(":splat")
        ctx.WriteString(fmt.Sprintf("匹配的file：%s", splat))
    })

    // 匹配ext，http://localhost:8080/ext/xxx.aaa
    beego.Get("/ext/*.*", func(ctx *context.Context) {
        // 匹配 . 前面的*用 path
        // 匹配 . 后面的用 ext
        path := ctx.Input.Param(":path")
        ext := ctx.Input.Param(":ext")
        ctx.WriteString(fmt.Sprintf("匹配ext前：%s,后：%s", path, ext))
    })

    // 启动beego
    beego.Run()
}
```

三、beego控制器

3.1、Restful风格的控制器

3.1.1、Restful风格的控制器（API格式）

```
    a、一切皆资源 ----> url
    b、通过动作来表示对资源的操作类型  http method ---->  post 创建，Get 或者查询，
Delete 删除，Put 更新
```

```
package main

// 导入beego的包
import (
    "github.com/astaxie/beego"
)

// 定义控制器
type HomeController struct {
    beego.Controller
}

// 1、可以处理Get请求
func (c *HomeController) Get() {
    c.Ctx.WriteString("Get")
}

// 2、可以处理Post请求
func (c *HomeController) Post() {
    c.Ctx.WriteString("Post")
}

// 3、可以处理Delete请求
func (c *HomeController) Delete() {
    c.Ctx.WriteString("Delete")
}

// 4、可以处理Put请求
func (c *HomeController) Put() {
    c.Ctx.WriteString("Put")
}

func main() {

    // 通过路由将URL跟控制器绑定
    beego.Router("/home", &HomeController{})

    beego.Run()
}
```

3.2、自定义匹配控制器

- 自定义路由规则，用分号切割路由规则
    - beego.Router("/task", &TaskContorller{}, "get,head:Query;post:Add;put:Modify")

```
package main

import "github.com/astaxie/beego"

// 自定义控制器&路由规则
type TaskContorller struct {
    beego.Controller
}

// 添加任务
func (t *TaskContorller) Add() {
    t.Ctx.WriteString("Add")
}

// 查询任务
func (t *TaskContorller) Query() {
    t.Ctx.WriteString("Query")
}

// 删除任务
func (t *TaskContorller) Del() {
    t.Ctx.WriteString("Del")
}

// 修改任务
func (t *TaskContorller) Modify() {
    t.Ctx.WriteString("Modify")
}

func main() {

    // 自定义路由规则，用分号切割路由规则
    beego.Router("/task", &TaskContorller{},
"get,head:Query;post:Add;put:Modify")
    // 启动
    beego.Run()
}
```

3.3、自动匹配路由，用的最多

```
package main

import "github.com/astaxie/beego"

// 自定义控制器&路由规则
type TaskContorller struct {
    beego.Controller
}

// 添加任务
func (t *TaskContorller) Add() {
    t.Ctx.WriteString("Add")
}

// 查询任务
func (t *TaskContorller) Query() {
    t.Ctx.WriteString("Query")
```

```
}

// 删除任务
func (t *TaskContorller) Del() {
    t.Ctx.WriteString("Del")
}

// 修改任务
func (t *TaskContorller) Modify() {
    t.Ctx.WriteString("Modify")
}

func main() {

    // 自动路由，用的最多
    // url 控制 controller/action(方法)
    // add ----> Add方法
    beego.AutoRouter(&TaskContorller{})
    // 启动
    beego.Run()
}
```

四、获取url的数据于相应

```
package main

import (
    "encoding/json"
    "fmt"

    "github.com/astaxie/beego"
)

// 定义一个结构体，用来接收数据
type InputForm struct {
    Name     string
    Password string `form:"password"` // 定义标签就可以在url传递小写的参数了
}

type InputContorller struct {
    beego.Controller
}

// 下面获取URL数据的函数放着这里即可

func main() {

    beego.AutoRouter(&InputContorller{})
    beego.Run()
}
```

4.1、获取数据

4.1.1、从GET请求获取数据

```
// 从Get请求拿数据 （http://localhost:8080//inputcontorller/QueryParams/?Name=12）
func (c *InputContorller) QueryParams() {
    // 获取url的数据:
    /*
        方式一：不常用，最基础的方式
        c.Ctx.Request.ParseForm()
        fmt.Println(c.Ctx.Request.Form)

        方式二：
        fmt.Println(c.Ctx.Input.Query("name"))

        方式三：不常用，知道就行
        var name string
        c.Ctx.Input.Bind(&name, "name")
        fmt.Println(name)

        方式四：
        fmt.Println(c.GetString("name"))
    */

    // 方式五：用的多。属性名跟URL传递的参数必须大小写一致（上面结构体的Name）。小写的话得定义标签，看上面结构体吧
    var form InputForm
    c.ParseForm(&form)
    fmt.Println(form)

    //
    c.Ctx.WriteString("")
}
```

4.1.2、从POST请求获取数据

```
// 从Post请求拿数据 （http://localhost:8080//inputcontorller/form/?Name=12）
func (c *InputContorller) Form() {
    /*
        方式一：
        c.Ctx.Request.ParseForm()
        fmt.Println(c.Ctx.Request.Form)
    */

    // 方式二：
    c.Ctx.Request.ParseForm()
    fmt.Println(c.Ctx.Request.PostForm)
    // 方式三：
    fmt.Println(c.GetString("name"))

    // 方式四：用的多。属性名跟URL传递的参数必须大小写一致（上面结构体的Name）。小写的话得定义标签，看上面结构体吧
    var form InputForm
    c.ParseForm(&form)
    fmt.Println(form)

    //
    c.Ctx.WriteString("")
}
```

4.1.3、beego上传文件

```
// 上传文件（http://localhost:8080//inputcontorller/file）
func (c *InputContorller) File() {
    // 方式一：使用Request 对象
    // c.Ctx.Request.FormFile("name")

    // 方式二：实际上就是封装了c.Ctx.Request.FormFile
    // c.GetFile("name")

    // 方式三：上传文件到当前路径upload目录下
    c.SaveToFile("img", "./upload/a.jpg")

    c.Ctx.WriteString("")
}
```

4.1.4、json

```
// json
func (c *InputContorller) Json() {
    // 一定要调用CopyBody，否则拿不到数据
    c.Ctx.Input.CopyBody(10 * 1024 * 1024) // 读取内容，后面会写到conf配置文件中
    var m map[string]interface{}

    json.Unmarshal(c.Ctx.Input.RequestBody, &m)     // 把json数据解析到 map中
    fmt.Println(string(c.Ctx.Input.RequestBody), m) // 解析数据
    c.Ctx.WriteString("")
}
```

4.1.5、beego设置普通cooking

```
func (c *InputContorller) Cookie() {
    // 方式一：
    cookie, err := c.Ctx.Request.Cookie("name")
    if err == nil {
        fmt.Println(cookie)
    }
    c.Ctx.WriteString("")

    // 方式二：
    c.Ctx.Input.Cookie("name")

    // 方式三：
    c.Ctx.GetCookie("name")

    // 设置cookie方式一：
    // http.SetCookie()

    // 设置cookie方式二：
    c.Ctx.SetCookie("name", "xxx")
}
```

4.1.6、一种安全的Cookie,就是有个签名的cooking

```
// 一种安全的Cookie,就是有个签名的cooking
func (c *InputContorller) SecureCookie() {
    // 获取Cooking
    c.Ctx.GetSecureCookie(cookieKey, "test") // ==  s.GetSecureCookie()

    // 三个参数对应 secretname,cookiename，cookievalue
    c.Ctx.SetSecureCookie(cookieKey, "test", "vvv") // == c.SetSecureCookie()

    // 给客户端一个相应
    c.Ctx.WriteString("")
}
```

4.1.7、获取请求信息

或者用这个：c.Ctx.Input.Method()---------->c.Ctx.Input.*(很多)

```
// 判断请求方式,获取请求信息
func (c *InputContorller) Header() {
    fmt.Printf("请求方式是：%s\n", c.Ctx.Request.Method)
    fmt.Printf("请求URL是：%s\n", c.Ctx.Request.URL)
    fmt.Printf("请求头是：%s\n", c.Ctx.Request.Header)

    // 给客户端一个相应
    c.Ctx.WriteString("")
}
```

五、响应方式

```
package main

import (
    "encoding/xml"

    "github.com/astaxie/beego"
)

type OutputController struct {
    beego.Controller
}

// 响应方式一（http://localhost:8080/output/cstring）。    调试用的比较多
func (c *OutputController) CString() {
    c.Ctx.WriteString("控制器响应方式一！")
}

// 响应方式二 （http://localhost:8080/output/outputbody）。调试用的比较多
func (c *OutputController) OutputBody() {
    c.Ctx.Output.Body([]byte("asdfghjkl"))
}

// 响应方式三 （响应到模板）  --- 常用
func (c *OutputController) Tpl() {
    // 指定模板名称即可，就会渲染出模板的数据来了
    c.TplName = "output.html"
```

```
}

// 响应方式四、五 （响应json、Yaml格式的数据）  --- 常用
func (c *OutputController) Json() {
    c.Data["json"] = map[string]string{"a": "xxx", "b": "yyy"}
    c.ServeJSON()
}

func (c *OutputController) Yaml() {
    c.Data["yaml"] = map[string]string{"a": "xxx", "b": "yyy"}
    c.ServeYAML()
}

// 方式六、XML格式响应数据
type User struct {
    Name string
    Addr string
}

func (c *OutputController) Xml() {
    c.Data["xml"] = struct {
        XMLName xml.Name `xml:"root"`
        User    User     `xml:"user"`
    }{User: User{Name: "kk", Addr: "aa"}}
    c.ServeXML()
}

// 方式七、重定向
func (c *OutputController) Redir() {
    c.Redirect("www.baidu.com", 302)

}

/*
    其他方式
    c.StopRun()    // 停止不往下执行
    c.Abort("404") // 抛出一个错误
*/
func main() {

    // 自动路由
    beego.AutoRouter(&OutputController{})

    // 启动beego
    beego.Run()
}
```

六、Beego之模板

6.1、Beego模板语法

- 指定要渲染的模板文件名称(在views文件夹中)
- 注意：若无任何响应，则加载控制器名称/函数名.tpl文件显示
- 传递数据用c.data[],传递，可传递的数据有很多，json、结构体、yaml、map等
- 默认只支持html、tpl两种后缀结尾

6.1.1、模板的传参c.data[]

```
package main

import "github.com/astaxie/beego"

type HomeController struct {
    beego.Controller
}

// 指定要渲染的模板文件名称(在views文件夹中)
// 注意：若无任何响应，则加载控制器名称/函数名.tpl文件显示
func (c *HomeController) Index() {
    // 传递数据用c.data[],传递，可传递的数据有很多，json、结构体、yaml、map等
    c.Data["name"] = "kk"
    c.Data["sex"] = true
    c.Data["sco"] = []int{1, 2, 3, 4, 5, 6}
    c.Data["user"] = map[int]string{1: "kk", 2: "jj"}
    c.TplName = "index.html" // 默认只支持html、tpl两种后缀结尾
}

func main() {

    // 自动路由
    beego.AutoRouter(&HomeController{})

    // 启动beego
    beego.Run()
}
```

6.1.2、模板的语法使用

```
<body>
    <!-- 简单渲染 -->
    <h1>{{ .name }}</h1>

    <!-- 添加判断 -->
    <h2>{{ if .sex }}男{{ else }}女{{ end }}</h2>

    <!-- 循环遍历方式一 -->
    <h3>
        {{ range .sco }}
        {{ . }}|
        {{ end }}
    </h3>

    <!-- 循环遍历方式二：变量重新赋值。$name 定义变量，$name使用 -->
    <h3>
        {{ range $index,$value := .sco }}
        {{ $index }} = {{ $value }} <br>
        {{ end }}
    </h3>

    <!-- 遍历map -->
    <h4>
        {{ range .user }}
        {{ . }}
```

```
        {{ end }}
    </h4>

    <h4>
        {{ range $key,$value := .user }}
        {{ $key }} = {{ $value }} <br>
        {{ end }}
    </h4>
</body>
```

6.2、beego自带模板函数

```
date （ymdHis）
dateformat(2006 01 02 15 04 05)
config.
```

6.3、beego自定义模板函数

6.3.1、模板函数的定义

```
package main

import (
    "strings"

    "github.com/astaxie/beego"
)

type HomeController struct {
    beego.Controller
}

// 指定要渲染的模板文件名称(在views文件夹中)
// 注意：若无任何响应，则加载控制器名称/函数名.tpl文件显示
func (c *HomeController) Index() {
    // 传递数据用c.data[],传递，可传递的数据有很多，json、结构体、yaml、map等
    c.Data["name"] = "kk"
    c.Data["sex"] = true
    c.Data["sco"] = []int{1, 2, 3, 4, 5, 6}
    c.Data["user"] = map[int]string{1: "kk", 2: "jj"}
    c.TplName = "index.html" // 默认只支持html、tpl两种后缀结尾
}

func main() {

    // 自定义模板函数，参数：自定义函数名称(调用的时候的名称)、自定义匿名函数(函数中实现功
能)、
    beego.AddFuncMap("lower", func(in string) string {
        // 大写转小写
        return strings.ToLower(in)
    })

    // 自动路由
    beego.AutoRouter(&HomeController{})

    // 启动beego
```

```
    beego.Run()
}
```

6.3.2、模板函数的使用与自定义模板的使用

```
<!-- 模板定义define,"自定义模板名称"，页面内的模板 -->
    {{ define "tzh"}}
    <!-- 模板函数调用方式2 -->
    这是一个简单的模板：{{ .|lower }}
    {{ end }}

    <!-- 自定义模板的使用()
         template "自定义模板name"
         多次调用!
    -->
    {{ template "tzh" .name }}
    {{ template "tzh" .name }}
    {{ template "tzh" .name }}
```

七、beego配置文件

- app.conf

```
# 应用配置
AddName=tzh666

# 运行模式设置,通过环境变量的方式获取，如果没设置环境变量RunMode，那么使用默认的模式(||后面的)
RunMode=${RunMode||prod}

# 开发模式，需要更改才能启动开发模式
[dev]

# 生产者模式，默认使用的模式
[prod]

# WEB文件夹配置
AutoRender=true
ViewsPath=views
StaticDir=statuc

# 监听的端口，ip配置
HTTPPort=${HTTPPort||9999}
HTTPAddr=192.168.1.1

include "mysql.conf"
```

- mysql.conf

```
# 配置MySQL相关信息，使用的时候 mysql::keyname
[mysqld]
MYSQL_HOST=192.168.1.107
MYSQL_PORT=3306
MYSQL_USER=goland
MYSQLPASSWORD=goland@2020
```

八、Beego之orm

8.1、orm使用步骤

```
beego之orm
    1、导入包
        "github.com/astaxie/beego/orm"
    2、注册驱动
        orm.RegisterDriver("mysql", orm.DRMySQL)
    3、注册数据库
        orm.RegisterDataBase("default", "mysql", dsn)
    4、定义数据模型 model
        type User struct {
            ID     int
            Name   string
            Gender bool
            Tel    string
            Heigth float32
        }
    5、注册数据模型
        orm.RegisterModel(new(User))
    6、操作
        同步表结构
            orm.RunCommand()  // go run .\main.go orm syncdb
            orm.RunSyncdb("default", true, true) // default 是注册数据库时候的别名
参数二为是否开启创建表  参数三是否更新表
        数据的操作：CRUD
```

8.2、orm代码实现

```
package main

import (
    "fmt"
    "time"

    "github.com/astaxie/beego/orm"

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

type UserModel struct {
    ID        int    `orm:"column(id);pk;auto;"` // 指定ID在数据库中的字段名是id(默认
是i_d),且是主键自增长
    Name      string `orm:"description(姓名)"`     // description 描述
    Gender    bool
    Tel       string
```

```
    Heigth    float32
    AAA       string `orm:"column(description);size(1024);default(aaa)"` // 设置
数据库字段名为description,默认 是aaa
    BBB       string
    CreatedAt *time.Time `orm:"auto_now_add"` // 插入数据的时间插入当前时间
    UpdateAt  *time.Time `orm:"auto_now"`     // 更改时间的时候,更改为当前时间
    DeleteAt  *time.Time `orm:"null"`         // 允许为空,beego默认字段名不能为空
}

// 把UserModel改名为user,再映射到数据库中
func (m *UserModel) TableName() string {
    return "user"
}

// 联合索引
func (m *UserModel) TableIndex() [][]string {
    return [][]string{
        {"AAA"},
        {"BBB"},
        {"BBB", "AAA"},
        {"BBB", "Name"},
        {"BBB", "Name", "AAA"},
    }
}

// 数据库信息
const (
    dbUser     = "root"
    dbPassWord = "123456"
    dbHost     = "192.168.1.208"
    dbPort     = 3306
    dbName     = "testorm"
)

func main() {
    // 注册驱动,默认beego已经注册了,可以省略不写
    orm.RegisterDriver("mysql", orm.DRMySQL)

    dsn := fmt.Sprintf("%s:%s@tcp(%s:%d)/%s?
charset=utf8mb4&loc=PRC&parseTime=true",
        dbUser, dbPassWord, dbHost, dbPort, dbName)

    // 注册数据库。别名、驱动名,dsn信息
    orm.RegisterDataBase("default", "mysql", dsn)

    // 注册模型
    orm.RegisterModel(new(UserModel))

    // 同步表结构
    // orm.RunCommand()
    orm.RunSyncdb("default", true, true)
}
```

8.3、orm之增加数据

```
    // 增加数据
    brithday, _ := time.Parse("2006-01-02", "1997-11-05")
```

```
    user := &User{
        Name:     "kk",
        Password: "123456",
        Gender:   true,
        Tel:      "15213659546",
        Brithday: &brithday,
    }
    // 数据库连接
    ormer := orm.NewOrm()
    // 插入数据
    id, err := ormer.Insert(user)
    fmt.Println(id, err)
```

8.4、orm之读取数据

```
    // 连接数据库
    ormer := orm.NewOrm()

    // 查找数据
    user := &User{ID: 1}
    err := ormer.Read(user)
    fmt.Println(user, err)

    user = &User{Name: "kk", Password: "123456"}
    err = ormer.Read(user)
    fmt.Println(user, err)
```

8.4、orm之更新数据

```
// 更新数据
user = &User{ID: 1}
err = ormer.Read(user)
fmt.Println(user, err)
user.Name = "hh"
ormer.Update(user)
```

8.5、orm之删除数据

```
// 删除数据
line, err := ormer.Delete(&User{ID: 2})
fmt.Println(line, err)
```

8.6、orm之查找数据集

```
    // 连接数据库
    ormer := orm.NewOrm()

    // 数据查找,使用到查询结果集,可以传递表名、结构体名 ormer.QueryTable("user")
    queryset := ormer.QueryTable(&User{})

    // 统计数据量
    count, err := queryset.Count()
    fmt.Println(count, err)

    // 获取所有数据
```

```
    var users []*User
    queryset.All(&users)

    // 条件查询 列表__条件,对象
    fmt.Println(queryset.Filter("name__iexact", "kk").Count())  // 等于
    fmt.Println(queryset.Exclude("name__iexact", "kk").Count()) // 不等于

    fmt.Println(queryset.Filter("name__contains", "kk").Count())           //
like
    fmt.Println(queryset.Filter("name__startswith", "kk").Count())         //
name 查询以kk开头的
    fmt.Println(queryset.Filter("name__endswith", "k").Count())            //
name 查询以kk结尾的
    fmt.Println(queryset.Filter("id__in", []int{1, 2, 3, 4, 5}).Count())   // in
    fmt.Println(queryset.Filter("id__gt", 5).Count())                      // >
大于
    fmt.Println(queryset.Filter("id__gt", 5).Filter("id__lt", 20).Count()) // 大
于且小于20

    fmt.Println(queryset.Limit(3).Offset(2).All(&users))      // 分页,All查询所有
数据
    fmt.Println(queryset.Limit(3).Offset(2).One(&users))      // 分页,One查询第一
条数据
    fmt.Println(queryset.OrderBy("Name", "-Tel").All(&users)) // 排序，默认升序，降
序"-Name"

    // name like '%kk%' and (tel like '152' or tel like '158') 拼接 where 条件
    cond := orm.NewCondition()
    condTel := orm.NewCondition()
    condTel = condTel.Or("tel__startswith", "152").Or("tel__startswith", "158")
// tel like '152' or tel like '158'
    cond = cond.And("name__contains", "kk").AndCond(condTel)
 // name like '%kk%' and
    fmt.Println(queryset.SetCond(cond).All(&users))
```

8.7、批量删除、更新

```
// 批量更新
queryset.Filter("id__gt", 10).Update(orm.Params{
    "name":   "xkk",
    "heigth": orm.ColValue(orm.ColAdd, 10), // 对原有的字段上 +10
})

// 批量删除
queryset.Filter("id__lt", 10).Delete()
```

8.9、原生SQL

```
db, err := orm.GetDB("default") // 别名
db.QueryRow().Scan()
db.Exec()
```