Beego Web框架调研

[liangdong@smzdm.com](mailto:liangdong@smzdm.com)

[安装 2](#_Toc535508159)

[下载依赖 2](#_Toc535508160)

[生成框架 2](#_Toc535508161)

[验证框架 2](#_Toc535508162)

[使用 4](#_Toc535508163)

[Bee工具 5](#_Toc535508164)

[Controller 5](#_Toc535508165)

[配置 5](#_Toc535508166)

[路由 5](#_Toc535508167)

[action 6](#_Toc535508168)

[Xsrf 7](#_Toc535508169)

[获取参数 7](#_Toc535508170)

[Session 8](#_Toc535508171)

[过滤器 9](#_Toc535508172)

[输出json 9](#_Toc535508173)

[输入验证 10](#_Toc535508174)

[错误处理 10](#_Toc535508175)

[日志管理 10](#_Toc535508176)

[Model 11](#_Toc535508177)

[安装 11](#_Toc535508178)

[数据库设置 12](#_Toc535508179)

[注册model 12](#_Toc535508180)

[ORM Object风格的INSERT操作 13](#_Toc535508181)

[Orm Object风格的Read操作 14](#_Toc535508182)

[QuerySelector风格的CURD 15](#_Toc535508183)

[原生SQL风格的CURD 15](#_Toc535508184)

[QueryBuilder风格的CURD 16](#_Toc535508185)

[事务 16](#_Toc535508186)

[View 16](#_Toc535508187)

# 安装

参考文档：<https://beego.me/quickstart>

## 下载依赖

$ go get -u github.com/astaxie/beego

$ go get -u github.com/beego/bee

## 生成框架

比如我最终要把项目上传到<https://github.com/owenliang/beego-demo>，那么就这样生成项目：

bin/bee new github.com/owenliang/beego-demo

框架会生成在$GOPATH/src/github.com/owenliang/beego-demo下面：

-rw-r--r-- 1 liangdong staff 25 1 16 10:10 README.md

drwxr-xr-x 3 liangdong staff 96 1 16 10:05 conf

drwxr-xr-x 3 liangdong staff 96 1 16 10:05 controllers

-rw-r--r-- 1 liangdong staff 129 1 16 10:05 main.go

drwxr-xr-x 2 liangdong staff 64 1 16 10:05 models

drwxr-xr-x 3 liangdong staff 96 1 16 10:05 routers

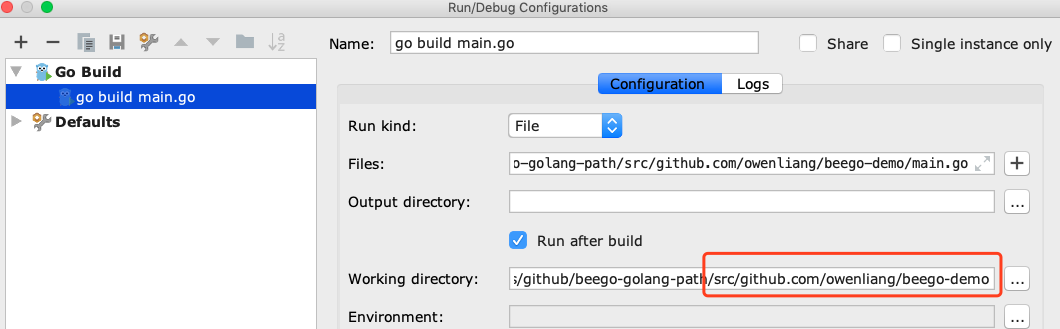
drwxr-xr-x 5 liangdong staff 160 1 16 10:05 static

drwxr-xr-x 3 liangdong staff 96 1 16 10:05 tests

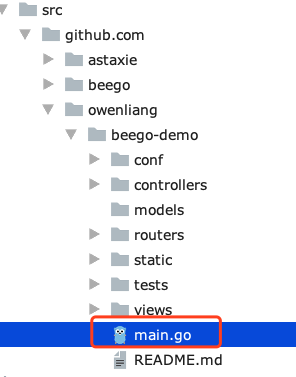
drwxr-xr-x 3 liangdong staff 96 1 16 10:05 views

## 验证框架

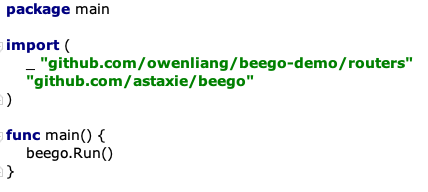
需要配置IDE修改程序的工作目录，因为beego会根据工作目录相对路径查找模板和配置文件。



然后就可以运行main.go了：



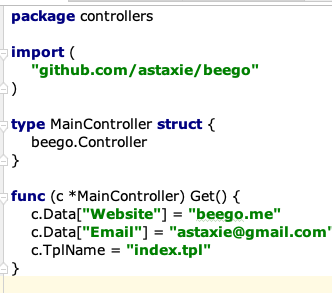
代码很简单，引入了routers模块，启动了beego：



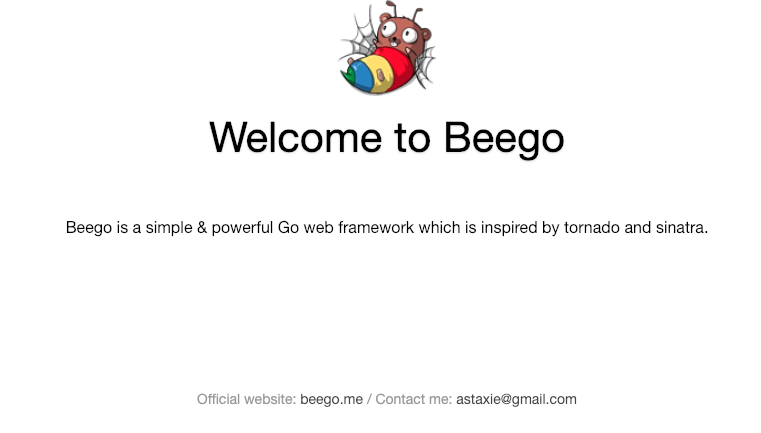
Routers模块会在init回调中注册路由到beego：

。

MainController渲染了一下views/index.tpl模板：



访问localhost:8080，页面正常：



# 使用

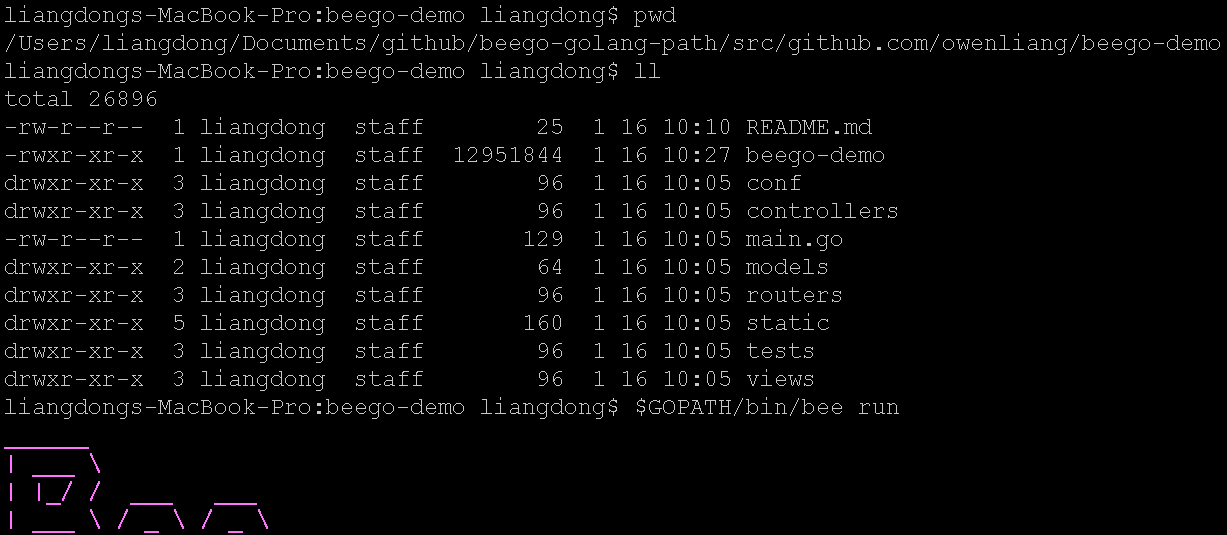
参考文档：<https://beego.me/docs/intro/>

## Bee工具

bee new创建web项目，bee api创建纯api项目。

bee run运行项目可以监听开发阶段的代码变化，实时自动编译，达到脚本语言的开发效率。

Bee run需要进入项目根目录执行：



bee generate用于生成mvc代码，后面遇到再说。

## Controller

### 配置

Conf/app.conf目录下，详细文档：

<https://beego.me/docs/mvc/controller/config.md>

runmode可以切换不同环境的多份配置，而runmode自身可以通过环境变量传递，从而实现不同环境不同配置的需要。

### 路由

详细参考：<https://beego.me/docs/mvc/controller/router.md>

一共3种模式：固定路由、正则路由、自动路由

固定路由，要求URI完全匹配：

beego.Router(**"/"**, &controllers.MainController{})

正则路由，长相有点奇怪，举个栗子：

beego.Router(“/api/:id([0-9]+)“, &controllers.RController{})

controller取参的时候：

this.Ctx.Input.Param(":id")

自动路由，需要把你的controller对象注册上去：

beego.AutoRouter(&controllers.ObjectController{})

beego会反射这个controller对象的方法，生成路由表，比如：

/object/login 调用 ObjectController 中的 Login 方法

/object/logout 调用 ObjectController 中的 Logout 方法

还支持namespace，说白了就是具备某个URI前缀的路由表，我把原先的router改成这样：

**func** init() {  
 ns := beego.NewNamespace(**"/v1"**,  
 beego.NSRouter(**"/"**, &controllers.MainController{}),  
 )  
  
 beego.AddNamespace(ns)  
}

就可以访问localhost:8080/v1了。

### action

我们的controller继承beego基类，有各种操作方法，controller里的数据已经被beego框架填充好了：

**type** MainController **struct** {  
 beego.Controller  
}

根据HTTP request的action的不同，会回调MainController的POST/GET/DELETE等方法，这些方法在beego.Controller中都有留空了实现，我们可以覆写。

**func** (c \*MainController) Get() {  
 c.Data[**"Website"**] = **"beego.me"** c.Data[**"Email"**] = **"astaxie@gmail.com"** c.TplName = **"index.tpl"**}

另外prepare和finish方法是留给我们覆写的，在执行action之前和之后做一些自定义操作，例如：  
**func** (c \*MainController) Prepare() {  
 c.Data[**"Website"**] = **"beego.me"** c.Data[**"Email"**] = **"astaxie@gmail.com"**}  
  
**func** (c \*MainController) Get() {  
 c.TplName = **"index.tpl"**}

### Xsrf

如果想防止跨站提交表单，看一下beego提供的API即可：<https://beego.me/docs/mvc/controller/xsrf.md>

### 获取参数

获取Get/Post参数：<https://beego.me/docs/mvc/controller/params.md>

比如请求：<http://localhost:8080/v1/?arg=xxx>，可以这样获取参数：

**func** (c \*MainController) Get() {  
 v := c.Input().Get(**"arg"**)  
 fmt.Println(v)  
 c.TplName = **"index.tpl"**}

也可以这样：

**func** (c \*MainController) Get() {  
 v := c.GetString(**"arg"**)  
 fmt.Println(v)  
 c.TplName = **"index.tpl"**}

区别就是前者是取原始解析出来的字符串，后者是支持强类型转换到int之类的。

Beego也支持文件表单上传，就不说明了。

### Session

beego支持session存储到redis/mysql之类的。

但是session的k-v最好别用struct作为value，否则序列化和反序列化很麻烦，就用普通int/string就行。

通常我们用redis，需要经历如下操作：

1. 安装beego的providor：

go get -u github.com/astaxie/beego/session/redis

然后在main.go中引入：

import \_ **"github.com/astaxie/beego/session/redis"**

1. 安装redis的driver：

go get github.com/gomodule/redigo

然后在main.go中引入：

\_ **"github.com/gomodule/redigo/redis"**

1. 启动redis：redis-server
2. 配置beego使用redis session：

appname = beego-demo  
httpport = 8080  
runmode = dev  
  
**sessionon = true  
sessionprovider = redis  
sessionproviderconfig=127.0.0.1:6379**

1. 在beego中访问session，统计用户浏览次数：

**func** (c \*MainController) Get() {  
 visit\_times := 0  
  
 times := c.GetSession(**"visit\_times"**)  
 **if** times != nil {  
 visit\_times = times.(int)  
 }  
  
 visit\_times += 1  
 c.SetSession(**"visit\_times"**, visit\_times)  
  
 c.Ctx.WriteString(**"visit\_times="** + strconv.Itoa(visit\_times))  
  
 *// c.TplName = "index.tpl"*}

### 过滤器

beego.InsertFilter(pattern string, position int, filter FilterFunc, params ...bool)

即在URI满足某个pattern规则的时候，会在某个生命期position插入该过滤回调filterFunc。

试了一下，某些position没有回调，用时候再追原因吧。

### 输出json

把结构体赋值给Data的json字段，然后调用ServeJSON就可以返回了：

**type** Resp **struct** {  
 Name string  
 Age int  
}  
  
**func** (c \*MainController) Get() {  
 c.Data[**"json"**] = &Resp{Name: **"owen"**, Age: 20}  
 c.ServeJSON()  
}

### 输入验证

MVC框架大多提供validator这种东西或者model，用来快速校验提交的表单数据是否合法。

Beego提供了一个库来简化验证工作，免得我们一个参数一个参数的去check：<https://beego.me/docs/mvc/controller/validation.md> ，这是需要安装的，等有需要再看。

### 错误处理

MVC框架一般支持运行出错的时候进入通用的error handler/error controller，beego也支持：<https://beego.me/docs/mvc/controller/errors.md> ，用error controller方式更简单合理。

### 日志管理

Beego封装了日志库，我们需要手动配置一波：

**func** init() {  
 *// 配置日志  
  
 // 打印行号* beego.SetLogFuncCall(***true***)  
 *// 配置文件日志(保存到beego.log, 日志最多保留1天, 最高级输出INFO日志)* beego.SetLogger(logs.***AdapterFile***, **`{"filename": "./beego.log", "maxdays": 1, "maxlines": 1, "level": 6}`**)  
 *// 删除命令行日志* beego.BeeLogger.DelLogger(logs.***AdapterConsole***)  
}  
  
**func** main() {  
 beego.Run()  
}

logger库支持多路输出，默认是输出到AdapterConsole，但是我把这一路删了，然后加了一路adpterFile输出到日志文件 。 日志支持按行数/大小/天滚动文件，上面我设置了maxliens:1，所以每打印1行日志，文件就被滚动换掉了：

-r--r----- 1 liangdong staff 1058 1 16 14:18 beego.2019-01-16.001.log

-r--r----- 1 liangdong staff 84 1 16 14:19 beego.2019-01-16.002.log

-rw-rw---- 1 liangdong staff 84 1 16 14:19 beego.log

Maxdays配置那些被归档的日志多久被删除，我设置的1天。

Beego框架本身就会打一些日志，我们程序也可以打：

**func** (c \*MainController) Get() {  
 beego.Info(**"heihei"**)  
 c.Data[**"json"**] = &Resp{Name: **"owen"**, Age: 20}  
 c.ServeJSON()  
}

更多细节参考：

<https://beego.me/docs/mvc/controller/logs.md>

<https://beego.me/docs/module/logs.md>

## Model

### 安装

Beego提供了ORM库，以mysql为例。

先安装mysql的driver：

Go get [github.com/go-sql-driver/mysql](https://github.com/go-sql-driver/mysql)

再安装beego orm：

go get github.com/astaxie/beego/orm

### 数据库设置

在本地建好数据库：create database beego;

在main.go中做注册：

**func** init() {  
 *// 注册driver* orm.RegisterDriver(**"mysql"**, orm.***DRMySQL***)  
 *// 注册数据库( 最后2个数字是最大空闲连接和最大数据库连接数量)* orm.RegisterDataBase(**"default"**, **"mysql"**, **"root:baidu@123@tcp(127.0.0.1:3306)/beego?charset=utf8&loc=Asia%2fShanghai"**, 1, 10)  
}  
  
**func** main() {  
 beego.Run()  
}

具体的mysql schema参考driver文档：<https://github.com/go-sql-driver/mysql>

### 注册model

在models目录下定义model：

**package** models  
  
**type** User **struct** {  
 Id int64  
 Name string **`orm:"size(64);unique"`** *// varchar(64), 唯一索引*}

可以用orm标签，指定各种属性告诉beego每个字段的约束，这个用途主要有2个：

1. 保存model到数据库的时候，可以供beego检查
2. 根据Model初始化table的时候，生成sql语句。

对于2）来说，我们通常会发现golang的web程序会自动创建table，这是通过model->table同步实现的：

**func** init() {  
 *// 注册driver* orm.RegisterDriver(**"mysql"**, orm.***DRMySQL***)  
 *// 注册数据库( 最后2个数字是最大空闲连接和最大数据库连接数量)* orm.RegisterDataBase(**"default"**, **"mysql"**, **"root:baidu@123@tcp(127.0.0.1:3306)/beego?charset=utf8&loc=Asia%2fShanghai"**, 1, 10)  
  
 *// 注册model* orm.RegisterModel(&models.User{})  
 *// 初始化table* orm.RunSyncdb(**"default"**, ***false***, ***false***)  
}

在registerModel后，可以通过runsyncdb做一次建表，底层执行的就是create if not exists，所以model的标签这时候就发挥了作用。

比如size(64)其实就会导致name字段的类型是varchar(64)，因为name是string类型。

查看数据库，user表已经建立：

| user | CREATE TABLE `user` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`name` varchar(64) NOT NULL DEFAULT '',

PRIMARY KEY (`id`),

UNIQUE KEY `name` (`name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

更多细节参考：<https://beego.me/docs/mvc/model/models.md>

### ORM Object风格的INSERT操作

插入：

**func** InsertUser(user \*User) (err error) {  
 *// 每次请求的ormer需要新建, 不能并发使用* orm := orm.NewOrm()  
  
 *// 插入* user.Id, err = orm.Insert(user)  
 **return**}

创建user对象，完成插入：

**func** (c \*MainController) Get() {  
 user := models.User{Name: **"owen"**}  
 **if** err := models.InsertUser(&user); err == nil {  
 beego.Info(**"插入用户:"**, user)  
 c.Ctx.WriteString(**"ok"**)  
 } **else** {  
 c.Ctx.WriteString(err.Error())  
 }  
}

执行1次插入了记录：

mysql> select \* from user;

+----+------+

| id | name |

+----+------+

| 1 | owen |

+----+------+

再执行报错：

Error 1062: Duplicate entry 'owen' for key 'name'

说明name字段根据struct标签定义为unique了。

### Orm Object风格的Read操作

定义方法：

**func** FindUser(user \*User) (err error) {  
 *// 每次请求的ormer需要新建, 不能并发使用* orm := orm.NewOrm()  
  
 err = orm.Read(user)  
 **return**}

然后查找：

**func** (c \*MainController) Get() {  
 *// 查找Id=1的记录* user := models.User{Id: 1}  
 **if** err := models.FindUser(&user); err == nil {  
 c.Ctx.WriteString(**"找到:"** + user.Name)  
 } **else** {  
 c.Ctx.WriteString(**"不存在"**)  
 }  
}

得到结果：

找到:owen

Delete和Update就不演示了，都是ORM对象风格，文档：<https://beego.me/docs/mvc/model/object.md>

### QuerySelector风格的CURD

上述纯ORM object的底层基于query selector构建SQL，这套API风格类似于django，不建议深入了解，文档：<https://beego.me/docs/mvc/model/query.md>

### 原生SQL风格的CURD

类似于这样的：

var users []User

num, err := o.Raw("SELECT id, user\_name FROM user WHERE id = ?", 1).QueryRows(&users)

仍旧支持struct的自动反序列化，我觉得这种就够用，不需要用ORM那些。

### QueryBuilder风格的CURD

避免手写SQL的尴尬，给我们提供链式构造SQL的能力：

qb.Select("user.name",

"profile.age").

From("user").

InnerJoin("profile").On("user.id\_user = profile.fk\_user").

Where("age > ?").

OrderBy("name").Desc().

Limit(10).Offset(0)

比ORM灵活，比裸写SQL清晰，我喜欢这种。

### 事务

很简单，围绕一个ORM()对象展开begin/rollback/commit：

<https://beego.me/docs/mvc/model/transaction.md>

## View

详细文档：<https://beego.me/docs/mvc/view/tutorial.md>，底层用的是Go标准库template，额外做了一些扩展。

下面举一个简单例子：

使用的时候，首先是准备模板数据放到controller的Data里，然后赋值TplName为相对于views目录的路径：

**func** (c \*MainController) Get() {  
 *// 查找Id=1的记录* user := models.User{Id: 1}  
  
 *// 找到记录* **if** err := models.FindUser(&user); err == nil {  
 *// 设置模板数据* c.Data[**"user"**] = &user  
 }  
  
 *// 设置模板文件* c.TplName = **"main.html"**}

然后编写main.html：

{{**template "header.html"** .}}  
 {{**if** .user}}  
 欢迎你，{{.user.Name | htmlquote}}  
 {{**else**}}  
 您没有登录!  
 {{**end**}}  
{{**template "footer.html"** .}}

这里引入了另外2个模板，并且把当前的上下文参数通过.带给了2个嵌套模板。

Go语言动态的部分都要放在{{}}内，具体语法参考文档。

Header.html和footer.html不需要{{define}}也能用，比go template方便点。

Header.html：

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>beego</**title**>  
</**head**>  
<**body**>

Footer.html：

<**script**></**script**>  
</**body**>  
</**html**>