Gin + Gorm 实战： 一小时完成一个简单的问答社区后端服务

问答社区是一种常见的社交化应用，允许用户发布问题、回答问题并相互交流。随着互联网的发展，问答社区已经成为人们获取知识和分享经验的重要平台。

本文将介绍如何使用 Gin 和 Gorm 构建一个简单的问答社区。本社区包含以下功能：

* 用户注册和登录
* 问题发布和回答
* 问题列表和详情
* 答案列表和详情
* 用户信息和回答列表

数据库设计

一共有users、questions、answers三个表，如下所示：

```
CREATE DATABASE IF NOT EXISTS qasys DEFAULT CHARSET utf8mb4 COLLATE utf8mb4_unicode_ci;

create table qasys.users
(
    id          bigint unsigned auto_increment primary key,
    username    varchar(255) not null comment '用户名',
    password    varchar(255) not null comment '密码',
    email       varchar(255) not null comment '邮箱',
    created_at datetime      null,
    updated_at datetime      null,
    deleted_at datetime      null,
    constraint email unique (email),
    constraint username unique (username)
);
```

```
create table qasys.questions
(
    id          bigint unsigned auto_increment primary key,
    user_id     int             not null,
    title       varchar(255) not null comment '标题',
    content     text            not null comment '问题详情',
    created_at datetime      null,
    updated_at datetime      null,
    deleted_at datetime      null
);

create index questions_user_id_index on qasys.questions (user_id);


create table qasys.answers
(
    id          bigint unsigned auto_increment primary key,
    question_id int       not null comment '问题id',
    user_id     int       not null comment '用户id',
    content     text      not null comment '答案',
    created_at  datetime null,
    updated_at  datetime null,
    deleted_at  datetime null
);

create index answers_question_id_index on qasys.answers (question_id);
create index answers_user_id_index on qasys.answers (user_id);

```


准备环境

1. 确保您的环境已经安装了 Go 和 一个mysql服务，并把上面的表导入mysql。
2. 安装一个脚手架sponge(集成了gin+gorm)，支持在windows、mac、linux环境下，点击查看 [安装sponge说明](http://cxyroad.com/"https://github.com/zhufuyi/sponge/blob/main/assets/install-cn.md")。
3. 安装完成后打开终端，启动sponge UI界面服务：


```
sponge run
```

```

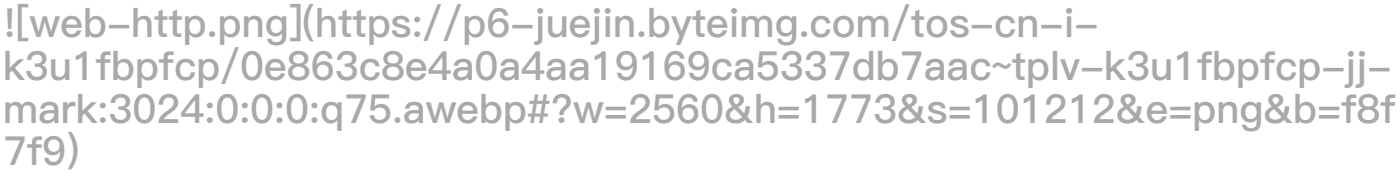
在浏览器访问 `http://localhost:24631`，进入sponge生成代码的UI界面。

### 创建问答社区服务

进入sponge的UI界面：

1. 点击左边菜单栏【SQL】-->【创建web服务】；
2. 选择数据库`mysql`，填写`数据库dsn`，然后点击按钮`获取表名`，选择表名(可多选)；
3. 填写其他参数，鼠标放在问号`?`位置可以查看参数说明；

填写完参数后，点击按钮`下载代码`生成web服务完整项目代码，如下图所示：



这是创建的web服务代码目录，已经包含了users, questions, answers三个表的CRUD api所有代码，包含了Gin和Gorm的初始化和配置代码，开箱即用。

```
.
├── cmd
│   └── qa
│   ├── initial
│   └── main.go
├── configs
├── deployments
│   ├── binary
│   ├── docker-compose
│   └── kubernetes
├── docs
├── internal
│   ├── cache
```
```

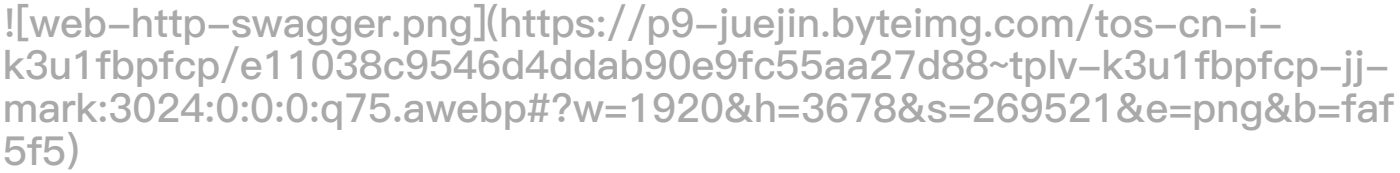
```
│   ├── config
│   ├── dao
│   ├── ecode
│   ├── handler
│   ├── model
│   ├── routers
│   ├── server
│   └── types
└── scripts
```

解压代码文件，打开终端，切换到web服务代码目录，执行命令：

```
# 生成swagger文档
make docs

# 编译和运行服务
make run
```

在浏览器打开 [http://localhost:8080/swagger/index.html](http://cxyroad.com/ "http://localhost:8080/swagger/index.html")，可以在页面上进行增删改查api测试，如下图所示：



从上图中可以看到，使用sponge生成服务已经完成了大部分api了，还有注册登录api、鉴权还没实现，接下完成未实现的功能。

添加注册登录api

1. 定义请求参数和返回结果结构体

进入目录`internal/types`，打开文件`users_types.go`，添加注册和登录的请求和返回结构体代码：

```
// RegisterRequest login request params
type RegisterRequest struct {
Email    string `json:"email" binding:"email"`    // 邮件
Username string `json:"username" binding:"min=2"` // 用户名
Password string `json:"password" binding:"min=6"` // 密码
}

// RegisterRespond data
type RegisterRespond struct {
Code int    `json:"code"` // return code
Msg  string `json:"msg"`  // return information description
Data struct {
ID uint64 `json:"id"`
} `json:"data"` // return data
}

// LoginRequest login request params
type LoginRequest struct {
Username string `json:"username" binding:"min=2"` // 用户名
Password string `json:"password" binding:"min=6"` // 密码
}

// LoginRespond data
type LoginRespond struct {
Code int    `json:"code"` // return code
Msg  string `json:"msg"`  // return information description
Data struct {
ID uint64 `json:"id"`
Token string `json:"token"`
} `json:"data"` // return data
}
```

2. 定义错误码

进入目录`internal/ecode`，打开文件`users_http.go`，添加两行行代码，定义注册和登录错误码：

```
var (
usersNO       = 49
usersName     = ”users”
usersBaseCode = errcode.HCode(usersNO)

// ...
ErrRegisterUsers       = errcode.NewError(usersBaseCode+10, ”注册失败
”)
ErrLoginUsers          = errcode.NewError(usersBaseCode+11, ”登录失败”)
// for each error code added, add +1 to the previous error code
)

```

3. 定义handler函数

进入目录`internal/handler`，打开文件`users.go`，添加注册和登录方法，并填写swagger注解：

```
// Register 注册
// @Summary 注册
// @Description register
// @Tags auth
// @accept json
// @Produce json
// @Param data body types.RegisterRequest true ”login information”
// @Success 200 {object} types.RegisterRespond{}
// @Router /api/v1/auth/register [post]
func (h *usersHandler) Register(c *gin.Context) {

}

// Login 登录
// @Summary 登录
// @Description login
// @Tags auth
// @accept json
// @Produce json
```

```
// @Param data body types.LoginRequest true "login information"
// @Success 200 {object} types.LoginRespond{}
// @Router /api/v1/teacher/login [post]
func (h *usersHandler) Login(c *gin.Context) {

}
```

然后把Register和Login方法添加到UsersHandler接口：

```
type UsersHandler interface {
// ...
Register(c *gin.Context)
Login(c *gin.Context)
}
```

4. 注册路由

进入目录`internal/routers`，打开文件`users.go`，把Register和Login路由注册进来：

```
func noAuthUsersRouter(group *gin.RouterGroup) {
h := handler.NewUsersHandler()
group.POST("/auth/register", h.Register)
group.POST("/auth/login", h.Login)
}
```

然后把`noAuthUsersRouter`函数添加到`routers.go`的注册路由函数下，如下所示：

```
func registerRouters(r *gin.Engine, groupPath string, routerFns
```

```
[]func(*gin.RouterGroup), handlers ...gin.HandlerFunc) {
rg := r.Group(groupPath, handlers...)

noAuthUsersRouter(rg)

for _, fn := range routerFns {
fn(rg)
}
}
```

5. 编写业务逻辑代码

进入目录`internal/handler`，打开文件`users.go`，编写注册和登录的业务逻辑代码：

```
func (h *usersHandler) Register(c *gin.Context) {
req := &types.RegisterRequest{}
err := c.ShouldBindJSON(req)
if err != nil {
logger.Warn("ShouldBindJSON error: ", logger.Err(err),
middleware.GCtxRequestIDField(c))
response.Error(c, ecode.InvalidParams)
return
}
ctx := middleware.WrapCtx(c)

password, err := gocrypto.HashAndSaltPassword(req.Password)
if err != nil {
logger.Error("gocrypto.HashAndSaltPassword error", logger.Err(err),
middleware.CtxRequestIDField(ctx))
response.Output(c, ecode.InternalServerError.ToHTTPCode())
return
}

users := &model.Users{
Username: req.Username,
Password: password,
Email:    req.Email,
}
```

```
err = h.iDao.Create(ctx, users)
if err != nil {
logger.Error("Create error", logger.Err(err), logger.Any("form", req),
middleware.GCtxRequestIDField(c))
response.Output(c, ecode.InternalServerError.ToHTTPCode())
return
}
response.Success(c, gin.H{"id": users.ID})
}

func (h *usersHandler) Login(c *gin.Context) {
req := &types.LoginRequest{}
err := c.ShouldBindJSON(req)
if err != nil {
logger.Warn("ShouldBindJSON error: ", logger.Err(err),
middleware.GCtxRequestIDField(c))
response.Error(c, ecode.InvalidParams)
return
}
ctx := middleware.WrapCtx(c)

condition := &query.Conditions{
Columns: []query.Column{
{
Name:  "username",
Exp:   "=",
Value: req.Username,
},
},
}
user, err := h.iDao.GetByCondition(ctx, condition)
if err != nil {
if errors.Is(err, model.ErrRecordNotFound) {
logger.Warn("Login not found", logger.Err(err), logger.Any("form", req),
middleware.GCtxRequestIDField(c))
response.Error(c, ecode.ErrLoginUsers)
} else {
logger.Error("Login error", logger.Err(err), logger.Any("form", req),
middleware.GCtxRequestIDField(c))
response.Output(c, ecode.InternalServerError.ToHTTPCode())
}
return
}

// 验证密码
if !gocrypto.VerifyPassword(req.Password, user.Password) {
logger.Warn("password error", middleware.CtxRequestIDField(ctx))
response.Error(c, ecode.ErrLoginUsers)
```

```
}

// 生成token
token, err := jwt.GenerateToken(utils.Uint64ToStr(user.ID),
user.Username)
if err != nil {
logger.Error("jwt.GenerateToken error", logger.Err(err),
middleware.CtxRequestIDField(ctx))
response.Output(c, ecode.InternalServerError.ToHTTPCode())
}

// TODO: 存储token到缓存

response.Success(c, gin.H{
"id":    user.ID,
"token": token,
})
}
```

6. 开启api鉴权

有了注册和登录api，其他api需要添加jwt鉴权，sponge生成的所有api默认是没有加入jwt鉴权的，只需开启即可，进入目录`internal/routers`，分别打开`questions.go`,`answers.go`,`users.go`代码，把默认注释代码去掉`//`反注释，表示下面所有路由都开启jwt鉴权，如下所示：

```
group.Use(middleware.Auth())
```

然后在需要鉴权的api的swagger注解中添加下面说明，这样在swagger页面请求api时，请求头会带上token，后端会从请求获取token值并验证token是否有效。

```
// @Security BearerAuth
```

7. 测试api

编写完业务逻辑代码后，在终端执行命令：

```
# 生成swagger文档
make docs

# 编译和运行服务
make run
```

在浏览器刷新[http://localhost:8080/swagger/index.html](http://cxyroad.com/”http://localhost:8080/swagger/index.html”)，在页面上可以看到注册和登录api，在页面测试注册和登录api，获取到token之后，把`Bearer token`填写到Authorize处，测试其他api是否可以正常调用。

8. 部署

默认支持部署到服务器、docker、k8s三种方式：

方式一：部署服务到远程linux服务

```
# 如果需要更新服务，再次执行此命令
make deploy-binary USER=root PWD=123456 IP=192.168.1.10
```

方式二：部署到docker

```
# 构建镜像
```

```
make image-build REPO_HOST=myRepo.com TAG=1.0

# 推送镜像，这里参数REPO_HOST和TAG，与构建镜像的参数
REPO_HOST和TAG一样。
make image-push REPO_HOST=myRepo.com TAG=1.0

# 复制 deployments/docker-compose 目录下的文件到目标服务器，修改镜
像地址后启动服务
docker-compose up -d
```

方式三： 部署到k8s

```
# 构建镜像
make image-build REPO_HOST=myRepo.com TAG=1.0

# 推送镜像，这里参数REPO_HOST和TAG，与构建镜像的参数
REPO_HOST和TAG一样。
make image-push REPO_HOST=myRepo.com TAG=1.0

# 复制 deployments/kubernetes 目录下的文件到目标服务器，修改镜像地址
后，按顺序执行脚本
kubectl apply -f ./*namespace.yml
kubectl apply -f ./
```

总结

sponge集成了Gin 和 Gorm 强大的web后端服务开发框架，可以帮助开发者快速且轻松构建 RESTful API 服务，这是[sponge github地址](http://cxyroad.com/ "https://github.com/zhufuyi/sponge")。

本文介绍了如何使用 Gin 和 Gorm 构建一个简单的问答社区，问答社区包含了一些基本功能，可以作为基础扩展到更复杂的应用。

原文链接: https://juejin.cn/post/7367674099877396507