CLASS

基于gin框架的前后端交互项目　　创作人：张政余　　前端支持：田贺元

一、总体逻辑

1.基于gin框架，通过GET和POST请求实现前后端数据交互

2.各结构体对象根据特殊标识相关联，实现一一对应

3.cookie鉴权，对某些路由请求进行权限控制

二、MySQL数据库结构：

```
//students表单

CREATE TABLE `students` (
`ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
`StuName` VARCHAR(20) NOT NULL,
`StuPassword` VARCHAR(20) NOT NULL,
`Phone` VARCHAR(20) DEFAULT '无',
`Age` INT(11) DEFAULT '0',
`Gender` varchar(255) DEFAULT '未知',
`Class` varchar(255) DEFAULT '未知',
`SecQue` varchar(255) NOT NULL,
`SecAns` varchar(255) NOT NULL,PRIMARY KEY(`ID`)
)ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4;

//测试数据
    stu := student.Student{
        //ID:          1,
        StuName:     "小明",
        StuPassword: "123456",
        Phone:       "13896764180",
        Age:         18,
        Gender:      "男",
        Class:       "9班",
        SecQue:      "你的大学是什么？",
        SecAns:      "重邮",
    }
    Gorm.Db.Create(&stu)
```

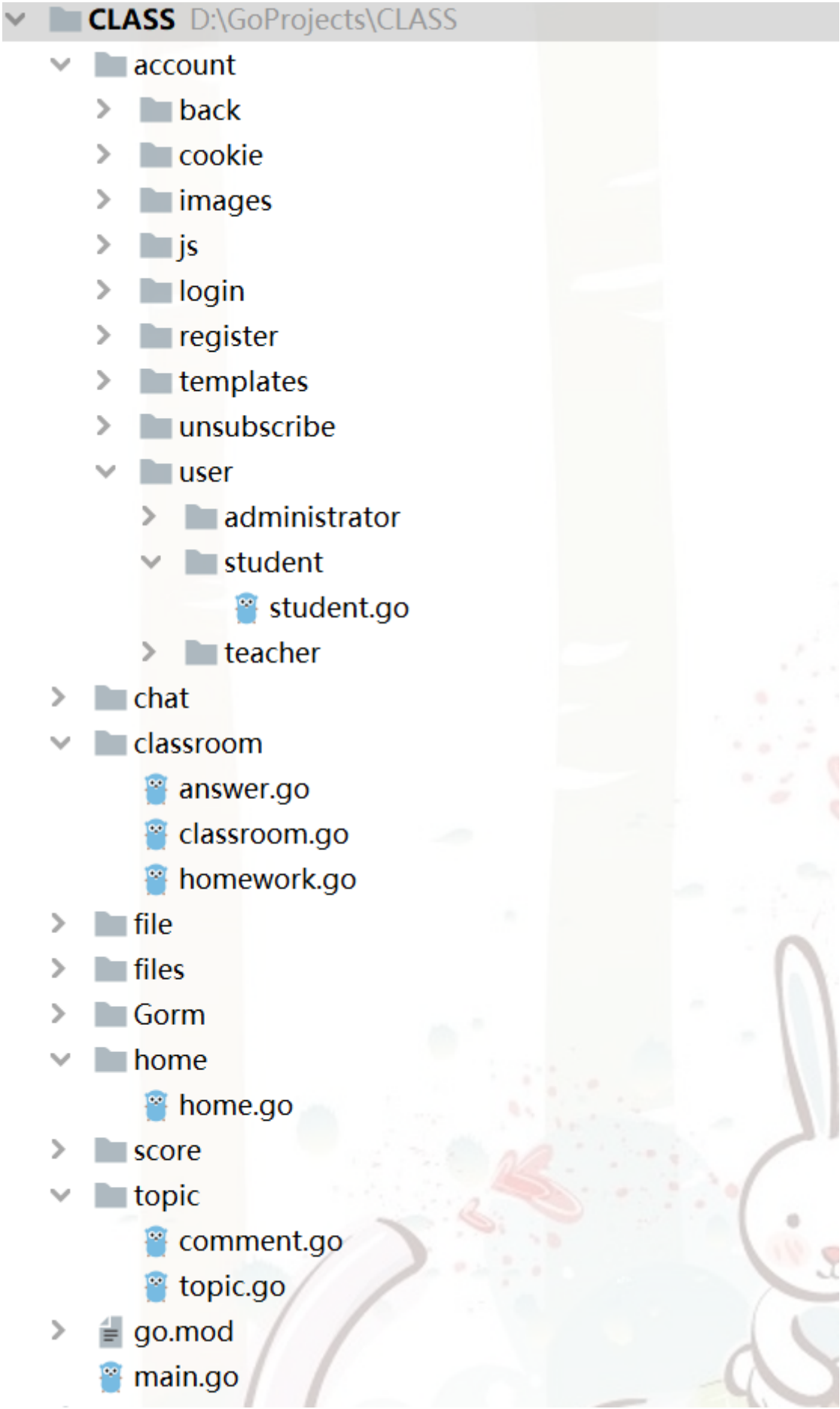
```
//topic

CREATE TABLE `Topics` (
`ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT KEY,
`Poster` VARCHAR(20) DEFAULT '未知',
`Title` VARCHAR(20) NOT NULL,
`content` text,
`PostTime` DATETIME,
`PointNum` BIGINT unsigned DEFAULT '0'
```

```
)ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4;


//comment

CREATE TABLE `Comments` (
`ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT KEY,
`TopicId` SMALLINT UNSIGNED,
`Observer` varchar(45) DEFAULT '未知',
`Comment` text,
`CommentTime` DATETIME,
`PointNum` BIGINT UNSIGNED DEFAULT '0'
)ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4;

//设置外键
ALTER TABLE 表名 ADD CONSTRAINT 外键名 FOREIGN KEY(列名) REFERENCES 主表名 (列名);

ALTER TABLE Comments ADD CONSTRAINT TopicId FOREIGN KEY(TopicId) REFERENCES
Topics (ID);

//成绩模块
CREATE TABLE `Scores` (
`ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
`ExaminationName` VARCHAR(20) NOT NULL,
`ExaminationNumber` INT(11) DEFAULT '999',
`StuName` VARCHAR(20) NOT NULL,
`Class` varchar(255) DEFAULT '未知',
`Phone` VARCHAR(20) DEFAULT '无',
`ChineseScore` INT(11) DEFAULT '0',
`MathScore` INT(11) DEFAULT '0',
`EnglishScore` INT(11) DEFAULT '0',
`TotalScore` INT(11) DEFAULT '0',PRIMARY KEY(`ID`)
)ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4;
```

```
//作业
CREATE TABLE `Homeworks` (
`ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT KEY,
`ClassMark` VARCHAR(20) NOT NULL DEFAULT '未知',
`Poster` VARCHAR(20) DEFAULT '未知',
`Title` VARCHAR(20) NOT NULL,
`Content` text,
`PostTime` DATETIME,
`Answer` VARCHAR(225) DEFAULT '略'
)ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4;

//回答
CREATE TABLE `Answers` (
`ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT KEY,
`Student` VARCHAR(20) DEFAULT '未知',
`HomeworkId` SMALLINT UNSIGNED NOT NULL,
`TheAnswer` text,
`AnswerTime` DATETIME
)ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4;
```

```
//课堂
CREATE TABLE `ClassRooms` (
`ID` SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT KEY,
`ClassMark` VARCHAR(20) NOT NULL DEFAULT '未知',
`Teacher` VARCHAR(20) DEFAULT '未知',
`Describe` VARCHAR(225) DEFAULT '无',
`CreateTime` DATETIME,
`SignNum` BIGINT UNSIGNED DEFAULT '0'
)ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4;
```

三、项目结构


CLASS D:\GoProjects\CLASS
account
back
cookie
images
js
login
register
templates
unsubscribe
user
administrator
student
student.go
teacher
chat
classroom
answer.go
classroom.go
homework.go
file
files
Gorm
home
home.go
score
topic
comment.go
topic.go
go.mod
main.go

四、后端建设

1.account账户部分

1)登录：

```
var User *student.Student

//初始化user
func CreateUser() {
    User = new(student.Student)
    User.StuName = "游客"
}

//LoginCheck 登录账号密码验证
func LoginCheck(Phone string, password string) (err error) {
    err = Gorm.Db.Where("Phone=?", Phone).Take(User).Error
    if err != nil {
        err = errors.New("该用户不存在！")
        fmt.Println(err)
        return err
    } else if User.StuPassword != password {
        err = errors.New("密码错误！")
        fmt.Println(err)
        return err
    } else {
        //token,_:=SetToken(username)
        //fmt.Println("token:",token)
        fmt.Println("登录成功！")
    }
    return
}

//GETLogin 登录的GET请求
func GETLogin(c *gin.Context) {
    c.HTML(http.StatusOK, "login.html", nil)
}

//POSTLogin 登录的POST请求
func POSTLogin(c *gin.Context) {
    //获取form参数
    Phone := c.PostForm("Phone")
    Password := c.PostForm("password")
    err := LoginCheck(Phone, Password) //检查用户名和密码是否正确
    if err != nil {
        c.String(200, "结果：", err)
    } else {
        //登录成功，设置中间件
        c.SetCookie("key_cookie", Phone, 3600, "/",
"http://127.0.0.1:9090/CLASS", false, false)
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/home")
    }
}
```

2）注册：

```
//RegisterCheck 注册查验
func RegisterCheck(User *student.Student) (err error) {
    //检查输入是否合法
    //这部分的合法性检验可以前端控制实现
    if User.StuName == "" {
        fmt.Println("姓名不能为空")
        return errors.New("姓名不能为空")
    } else if User.Phone == "" {
        fmt.Println("电话号码不能为空")
        return errors.New("电话号码不能为空")
    } else if User.StuPassword == "" {
        fmt.Println("密码不能为空")
        return errors.New("密码不能为空")
    }
    //检查用户是否已经存在
    err = Gorm.Db.Where("Phone=?", User.Phone).Take(User).Error
    if errors.Is(err, gorm.ErrRecordNotFound) {
        err = Gorm.Db.Save(User).Error
        if err != nil {
            fmt.Println(err)
            return
        }
    } else {
        fmt.Println("电话号码已被使用")
        return errors.New("电话号码已被使用")
    }
    return
}

//GETRegister 注册GET
func GETRegister(c *gin.Context) {
    c.HTML(http.StatusOK, "register.html", nil)
}

//POSTRegister 注册POST
func POSTRegister(c *gin.Context) {
    U := new(student.Student)
    c.Bind(U)
    err := RegisterCheck(U)
    if err == nil {
        c.HTML(http.StatusOK, "login.html", nil)
    } else { //若存在，则不可以注册
        c.HTML(http.StatusOK, "register.html", gin.H{
            "outcome": err,
        })
    }
}
```

3）账号注销：

```
//只能在成功登录的情况下才允许注销账号

//GETUnsubscribe 注销的get请求
func GETUnsubscribe(c *gin.Context) {
```

```
    c.HTML(http.StatusOK, "unsubscribe.html", gin.H{"SecQue":
login.User.SecQue})
}

//POSTUnsubscribe 注销的POST请求
func POSTUnsubscribe(c *gin.Context) {
    StuName := c.PostForm("StuName")
    StuPassword := c.PostForm("StuPassword")
    Phone := c.PostForm("Phone")
    SecAns := c.PostForm("SecAns")
    if StuName != login.User.StuName {
        fmt.Println("姓名错误！")
    } else if StuPassword != login.User.StuPassword {
        fmt.Println("密码错误！")
    } else if Phone != login.User.Phone {
        fmt.Println("电话号码有误！")
    } else if SecAns != login.User.SecAns {
        fmt.Println("密保错误！")
    } else {
        U := new(student.Student)
        err := Gorm.Db.Where("Phone=?", Phone).Delete(U)
        fmt.Println("注销err", err)
        if err == nil {
            c.JSON(http.StatusOK, gin.H{"结果：": "注销失败！"})
            return
        }
        c.JSON(http.StatusOK, gin.H{"结果：": "感谢您的使用！"})
    }
}
```

4）找回密码：

```
//BackCheck 密保找回
func BackCheck(StuName string, Phone string) (err error) {
    err = Gorm.Db.Where("Phone=?", Phone).Take(login.User).Error
    if err != nil {
        fmt.Println(err)
        errors.New("手机号不存在！")
        return err
    } else if login.User.StuName != StuName {
        fmt.Println(login.User.StuName)
        err = errors.New("学生姓名错误！")
        fmt.Println(err)
        return err
    }
    return nil
}

//GETback 找回密码的GET请求
func GETback(c *gin.Context) {
    c.HTML(http.StatusOK, "back1.html", nil)
}

//POSTback1 找回密码的POST请求1
func POSTback1(c *gin.Context) {
    StuName := c.PostForm("StuName")
    Phone := c.PostForm("Phone")
```

```
    err := BackCheck(StuName, Phone)
    if err != nil {
        c.JSON(200, gin.H{"结果：": err})
        return
    } else {
        c.HTML(http.StatusOK, "back2.html", gin.H{"SecQue": login.User.SecQue})
    }
    return
}

//POSTback2 找回密码的POST请求2
func POSTback2(c *gin.Context) {
    SecAns := c.PostForm("SecAns")
    fmt.Println("SecAns:", SecAns)
    if SecAns != login.User.SecAns {
        c.JSON(200, gin.H{"结果：": errors.New("密保答案错误！")})
        return
    } else {
        c.JSON(http.StatusOK, gin.H{"所有信息": login.User})
    }
}
```

5）cookie鉴权：

```
//定义中间件，用cookie记住登录状态
func AuthMiddleWare() gin.HandlerFunc {
   return func(c *gin.Context) {
      //获取客户端cookie并校验
      cookie, err := c.Cookie("key_cookie")
      if err == nil {
         if cookie == login.User.Phone {
            c.Next()
            return
         }
      }
      //fmt.Println("cookie err:",err)
      //异常，重新登录
      c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/login")
      c.Abort() //若验证不通过，不再调用后续函数处理
      return
   }
}
```

2.主页面

```
//GETHome 主页面的GET请求
func GETHome(c *gin.Context) {
    classRooms, err := classroom.GetClassRoom()
    if err != nil {
        c.JSON(404, gin.H{"结果:历史课堂获取失败！err:": err})
    }
    Students, err := student.GetStudents()
    if err != nil {
```

```
        c.JSON(404, gin.H{"结果:用户获取失败! err:": err})
    }
    Scores, err := score.GetScore()
    if err != nil {
        c.JSON(404, gin.H{"结果:成绩获取失败! err:": err})
    }
    topics, err := topic.GetTopic()
    if err != nil {
        c.JSON(404, gin.H{"结果:话题获取失败! err:": err})
    }
    files, err := file.GetAllFile("./files")
    if err != nil {
        fmt.Println("文件获取失败err: ", err)
    }
    c.HTML(200, "home.html", gin.H{
        "classRooms": classRooms,
        "Students":   Students,
        "Scores":     Scores,
        "Topics":     topics,
        "username":   login.User.StuName,
        "files":      files,
    })
}

func OUT(c *gin.Context) {
    login.User = new(student.Student)
    c.Redirect(http.StatusMovedPermanently, "http://127.0.0.1:9090/CLASS/login")
}
```

3.成绩

```
//分数构成
type score struct {
    ID                int    `gorm:"column:ID" form:"ID" json:"ID,omitempty"`
    ExaminationName   string `gorm:"column:ExaminationName"
form:"ExaminationName" json:"ExaminationName,omitempty"`
    ExaminationNumber int    `gorm:"column:ExaminationNumber"
form:"ExaminationNumber" json:"ExaminationNumber,omitempty"`
    StuName           string `gorm:"column:StuName" form:"StuName"
json:"StuName,omitempty"`
    Class             string `gorm:"column:Class" form:"Class"
json:"Class,omitempty"`
    Phone             string `gorm:"column:Phone" form:"Phone"
json:"Phone,omitempty"`
    ChineseScore      int    `gorm:"column:ChineseScore" form:"ChineseScore"
json:"ChineseScore,omitempty"`
    MathScore         int    `gorm:"column:MathScore" form:"MathScore"
json:"MathScore,omitempty"`
    EnglishScore      int    `gorm:"column:EnglishScore" form:"EnglishScore"
json:"EnglishScore,omitempty"`
    TotalScore        int    `gorm:"column:TotalScore" form:"TotalScore"
json:"TotalScore,omitempty"`
}

//CreateScore 新增成绩
func CreateScore(S *score) (err error) {
```

```
    err = Gorm.Db.Save(S).Error
    if err != nil {
        fmt.Println("新增成绩失败err：", err)
        return err
    }
    return nil
}

//deleteScore 删除成绩
func deleteScore(ID int) (err error) {
    err = Gorm.Db.Where("ID=?", ID).Delete(&score{}).Error
    if err != nil {
        fmt.Println("成绩删除失败err：", err)
        return err
    }
    return nil
}

//GetScore 获取成绩
func GetScore() (scores []score, err error) {
    err = Gorm.Db.Find(&scores).Error
    if err != nil {
        fmt.Println("成绩查询失败err：", err)
        return nil, err
    }
    return scores, nil
}

//AddScore 成绩发布页面的GET请求
func AddScore(c *gin.Context) {
    c.HTML(200, "score.html", gin.H{"username": login.User.StuName})
}

//POStScore 成绩添加的POST请求
func POStScore(c *gin.Context) {
    Score := new(score)
    err := c.Bind(Score)
    if err != nil {
        fmt.Println("成绩绑定失败! err：", err)
        return
    }
    if Score.StuName == " " || Score.ExaminationName == " " || Score.Class == "
" || Score.ExaminationNumber == 0 {
        fmt.Println("所填数据不合法")
        c.JSON(http.StatusOK, gin.H{"结果：": "添加失败！", "错误：": "所填数据不合
法"})
        return
    }
    err = CreateScore(Score)
    if err == nil {
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/home")
    } else {
        c.JSON(http.StatusOK, gin.H{"结果：": "添加失败！", "错误：": err})
    }

}
```

```
//Deletescore 删除分数的POST请求
func Deletescore(c *gin.Context) {
    ScoreId := c.Param("ScoreId")
    ID, _ := strconv.Atoi(ScoreId)
    err := deleteScore(ID)
    if err != nil {
        fmt.Println("成绩删除失败! err: ", err)
        return
    }
    c.Redirect(http.StatusMovedPermanently, "http://127.0.0.1:9090/CLASS/home")

}
```

4.话题

1) topic

```
//话题
type topic struct {
    ID       int       `gorm:"column:ID" form:"ID" json:"ID,omitempty"`
    Poster   string    `gorm:"column:Poster" form:"Poster"
json:"Poster,omitempty"`       //发布者
    Title    string    `gorm:"column:Title" form:"Title" json:"Title,omitempty"`
         //话题标题
    Content  string    `gorm:"column:Content" form:"Content"
json:"Content,omitempty"`    //话题内容
    PostTime time.Time `gorm:"column:PostTime" form:"PostTime" json:"PostTime"`
          //发布时间
    PointNum int       `gorm:"column:PointNum" form:"PointNum"
json:"PointNum,omitempty"` //点赞
}

//新增话题
func CreateTopic(T *topic) (err error) {
    err = Gorm.Db.Save(T).Error
    if err != nil {
        fmt.Println("新增话题失败err: ", err)
        return err
    }
    return nil
}

//删除话题
func deleteTopic(ID int) (err error) {
    err = Gorm.Db.Where("ID=?", ID).Delete(&topic{}).Error
    if err != nil {
        fmt.Println("删除失败err: ", err)
        return err
    }
    return nil
}

// PointTopic 话题点赞
func PointTopic(ID int) (err error) {
    err = Gorm.Db.Model(&topic{}).Where("ID=?", ID).Update("PointNum",
gorm.Expr("PointNum + 1")).Error
```

```
    if err != nil {
        fmt.Println("点赞失败err", err)
        return err
    }
    return nil
}

// GetTopic 获取话题
func GetTopic() (topics []topic, err error) {
    err = Gorm.Db.Find(&topics).Error
    if err != nil {
        fmt.Println("话题查询失败err：", err)
        return nil, err
    }
    return topics, nil
}

// GETTopic 发布话题的GET请求
func GETTopic(c *gin.Context) {
    c.HTML(http.StatusOK, "topic.html", gin.H{"username": login.User.StuName})
}

//POSTTopic 发布话题的POST请求
func POSTTopic(c *gin.Context) {
    T := new(topic)
    err := c.Bind(T)
    if err != nil {
        fmt.Println("话题数据绑定失败err：", err)
        return
    }
    T.Poster = login.User.StuName
    T.PostTime = time.Now()
    err = CreateTopic(T)
    if err != nil {
        c.JSON(http.StatusBadGateway, gin.H{"结果：": "发布失败！"})
    }
    //发布成功，返回主页面
    c.Redirect(http.StatusMovedPermanently, "http://127.0.0.1:9090/CLASS/home")
}

//ONETopic 单个话题的详情页面
func ONETopic(c *gin.Context) {
    Topic := new(topic)
    ID := c.Param("ID") //获取话题ID
    id, _ := strconv.Atoi(ID)
    err := Gorm.Db.Where("ID=?", id).First(Topic).Error
    if err == nil {
        Comments, err := GetComment(id)
        if err != nil {
            fmt.Println("评论查询失败err", err)
        }
        c.HTML(200, "onetopic.html", gin.H{
            "Title":    Topic.Title,
            "Content":  Topic.Content,
            "PostTime": Topic.PostTime,
            "PointNum": Topic.PointNum,
            "username": login.User.StuName,
            "ID":       Topic.ID,
```

```
            "Comments": Comments,
        })
    } else {
        fmt.Println(err)
        c.JSON(http.StatusOK, gin.H{"结果：": "查询失败！"})
    }
}

//DeleteTopic 删除单个话题
func DeleteTopic(c *gin.Context) {
    ID := c.Param("ID")
    id, err := strconv.Atoi(ID)
    if err != nil {
        fmt.Println("转换 ID err：", err)
    }
    err = deleteTopic(id)
    if err != nil {
        fmt.Println("删除话题失败err:", err)
        return
    } else {
        //c.JSON(http.StatusOK, gin.H{"结果：": "删除成功！"})
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/home") //删除成功，返回主页面
    }
}

//PointT 话题点赞
func PointT(c *gin.Context) {
    ID := c.Param("ID")
    id, err := strconv.Atoi(ID)
    if err != nil {
        fmt.Println("点赞 ID 转换 err:", err)
        return
    }
    err = PointTopic(id)
    if err != nil {
        fmt.Println("点赞失败 MySQL err:", err)
        return
    }
    c.Redirect(http.StatusMovedPermanently, "http://127.0.0.1:9090/CLASS/home")
//点赞成功，返回主页面
}
```

2) comment

```
//评论
type comment struct {
    ID          int        ` gorm:"column:ID" form:"ID" json:"ID,omitempty"`
    TopicId     int        `gorm:"column:TopicId" form:"TopicId"
json:"TopicId,omitempty"`    //外键与Topic绑定标识
    Observer    string     `gorm:"column:Observer" form:"Observer"
json:"Observer,omitempty"` //评论者
    Comment     string     `gorm:"column:Comment" form:"Comment"
json:"Comment,omitempty"`    //评论
    CommentTime time.Time `gorm:"column:CommentTime" form:"CommentTime"
json:"CommentTime"`  //评论时间
```

```
    PointNum    int       `gorm:"column:PointNum" form:"PointNum"
json:"PointNum,omitempty"` //点赞数
}

// CreateComment 新增评论
func CreateComment(T *comment) (err error) {
    err = Gorm.Db.Save(T).Error
    if err != nil {
        fmt.Println("新增话题失败 err: ", err)
        return err
    }
    return nil
}

//deleteComment 删除评论
func deleteComment(ID int) (err error) {
    err = Gorm.Db.Where("ID=?", ID).Delete(&comment{}).Error
    if err != nil {
        fmt.Println("删除评论失败err: ", err)
        return err
    }
    return nil
}

// PointComment 评论点赞
func PointComment(ID int) (err error) {
    err = Gorm.Db.Model(&comment{}).Where("ID=?", ID).Update("PointNum",
gorm.Expr("PointNum + 1")).Error
    if err != nil {
        fmt.Println("评论点赞失败err", err)
        return err
    }
    return nil
}

// GetComment 获取话题对应评论
func GetComment(id int) (comments []comment, err error) {
    err = Gorm.Db.Where("TopicId = ? ", id).Find(&comments).Error
    if err != nil {
        fmt.Println("评论查询失败err: ", err)
        return nil, err
    }
    return comments, nil
}

//POSTComment 发布评论的POST请求
func POSTComment(c *gin.Context) {
    Comment := new(comment)
    err := c.Bind(Comment)
    if err != nil {
        fmt.Println("评论数据绑定失败err: ", err)
        return
    }
    Comment.Observer = login.User.StuName
    ID := c.Param("ID") //获取Topic的ID
    id, _ := strconv.Atoi(ID)
    Comment.TopicId = id
    Comment.CommentTime = time.Now()
```

```
    err = CreateComment(Comment)
    if err != nil {
        c.JSON(http.StatusOK, gin.H{"结果：": "发布失败！"})
    }
    c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/onetopic/"+ID)
}

//DeleteComment 删除评论的POST请求
func DeleteComment(c *gin.Context) {
    ID := c.Param("commentID") //获取评论ID
    id, _ := strconv.Atoi(ID)
    err := deleteComment(id)
    if err != nil {
        fmt.Println("删除话题失败err:", err)
        return
    } else {
        //c.JSON(http.StatusOK, gin.H{"结果：": "删除成功！"})
        TopicId := c.Param("ID")
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/onetopic/"+TopicId)
    }
}

//PointC 评论点赞的POST请求
func PointC(c *gin.Context) {
    ID := c.Param("commentID")
    id, err := strconv.Atoi(ID)
    if err != nil {
        fmt.Println("点赞 评论ID 转换失败 err:", err)
        return
    }
    err = PointComment(id)
    if err != nil {
        fmt.Println("点赞失败MySQL err:", err)
        return
    }
    TopicId := c.Param("ID")
    c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/onetopic/"+TopicId)
}
```

5.课堂

1）classroom

```
//课堂
type Classroom struct {
    ID         int       `gorm:"column:ID" json:"ID,omitempty" form:"ID"`
    ClassMark  string    `gorm:"column:ClassMark" json:"ClassMark,omitempty"
form:"ClassMark"`
    Teacher    string    `gorm:"column:Teacher" json:"Teacher,omitempty"
form:"Teacher"`
    Describe   string    `gorm:"column:Describe" json:"Describe,omitempty"
form:"Describe"`
```

```
    SignNum    int       `gorm:"column:SignNum" json:"SignNum,omitempty"
form:"SignNum"`
    CreateTime time.Time `gorm:"column:CreateTime" json:"CreateTime,omitempty"
form:"CreateTime"`
}

//GetClassRoom 获取课堂记录
func GetClassRoom() (classRooms []Classroom, err error) {
    err = Gorm.Db.Find(&classRooms).Error
    if err != nil {
        fmt.Println("课堂查询失败err: ", err)
        return nil, err
    }
    return classRooms, nil
}

//oneclass 获取一个课堂
func oneclass(ClassMark string) (calssroom Classroom, err error) {
    err = Gorm.Db.Where("ClassMark=?", ClassMark).First(&calssroom).Error
    if err != nil {
        fmt.Println("单个课堂获取失败err:", err)
        return calssroom, err
    }
    return calssroom, nil
}

//deleteclassroom 删除课堂
func deleteclassroom(ClassMark string) (err error) {
    err = Gorm.Db.Where("ClassMark=?", ClassMark).Delete(&Classroom{}).Error
    if err != nil {
        fmt.Println("删除失败err: ", err)
        return err
    }
    return nil
}

//Signin 课堂签到
func Signin(ClassMark string) (err error) {
    err = Gorm.Db.Model(&Classroom{}).Where("ClassMark=?",
ClassMark).Update("SignNum", gorm.Expr("SignNum + 1")).Error
    if err != nil {
        fmt.Println("签到失败err", err)
        return err
    }
    return nil
}

//生成随机字符串，作为课堂暗号
func randomString(len int) string {
    bytes := make([]byte, len)
    for i := 0; i < len; i++ {
        bytes[i] = byte(65 + rand.Intn(25))
    }
    return string(bytes)
}

//GETClassroom 单个课堂页面
func GETClassroom(c *gin.Context) {
```

```
    ClassMark := c.Param("ClassMark")
    classroom, err := oneclass(ClassMark)
    if err != nil {
        c.JSON(404, gin.H{"结果:单个获取失败! err:": err})
    }
    homeworks, err := GetHomework(ClassMark)
    if err != nil {
        c.JSON(404, gin.H{"结果:作业获取失败! err:": err})
    }
    c.HTML(200, "classroom.html", gin.H{
        "homeworks": homeworks,
        "Teacher":   classroom.Teacher,
        "Describe":  classroom.Describe,
        "ClassMark": classroom.ClassMark,
        "SignNum":   classroom.SignNum,
    })
}

//Create 创建课堂的GET请求
func Create(c *gin.Context) {
    c.HTML(200, "createclassroom.html", nil)
}

//CreateClassroom 创建课堂的POST请求
func CreateClassroom(c *gin.Context) {
    rand.Seed(time.Now().UnixNano())
    classRoom := new(Classroom)
    err := c.Bind(classRoom)
    if err != nil {
        fmt.Println("课堂数据绑定失败err：", err)
        return
    }
    classRoom.ClassMark=randomString(5)
    classRoom.CreateTime = time.Now()
    err = Gorm.Db.Save(classRoom).Error
    if err != nil {
        fmt.Println("新增课堂失败err：", err)
        return
    }
    c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/classroom/"+classRoom.ClassMark)

}

//EnterClassroom 加入课堂
func EnterClassroom(c *gin.Context) {
    ClassMark := c.PostForm("ClassMark")
    c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/classroom/"+ClassMark)
}

//SignIn 课堂签到
func SignIn(c *gin.Context) {
    ClassMark := c.Param("ClassMark")
    err := Signin(ClassMark)
    if err != nil {
        fmt.Println("签到失败MySQL err:", err)
        return
```

```
    }
    c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/classroom/"+ClassMark)
}
```

2) homework

```
type Homework struct {
    ID        int       `form:"ID" gorm:"column:ID" json:"ID,omitempty"`
    ClassMark string    `gorm:"column:ClassMark" json:"ClassMark,omitempty"
form:"ClassMark"`
    Poster    string    `form:"Poster" gorm:"column:Poster"
json:"Poster,omitempty"`
    Title     string    `form:"Title" gorm:"column:Title"
json:"Title,omitempty"`
    Content   string    `form:"Content" gorm:"column:Content"
json:"Content,omitempty"`
    Answer    string    `form:"Answer" gorm:"column:Answer"
json:"Answer,omitempty"`
    PostTime  time.Time `form:"PostTime" gorm:"column:PostTime" json:"PostTime"`
}

//CreateHomework 发布作业
func CreateHomework(H *Homework) (err error) {
    err = Gorm.Db.Save(H).Error
    if err != nil {
        fmt.Println("发布作业失败err: ", err)
        return err
    }
    return nil
}

//deleteHomework 删除作业
func deleteHomework(ID int) (err error) {
    err = Gorm.Db.Where("ID=?", ID).Delete(&Homework{}).Error
    if err != nil {
        fmt.Println("作业删除失败err: ", err)
        return err
    }
    return nil
}

//GetHomework 获取作业
func GetHomework(ClassMark string) (HomeworkS []Homework, err error) {
    err = Gorm.Db.Where("ClassMark=?", ClassMark).Find(&HomeworkS).Error
    if err != nil {
        fmt.Println("作业查询失败err: ", err)
        return nil, err
    }
    return HomeworkS, nil
}

//GETHomework GETTopic
func GETHomework(c *gin.Context) {
    c.HTML(http.StatusOK, "homework.html", gin.H{
        "username":  login.User.StuName,
```

```
        "ClassMark": c.Param("ClassMark"),
    })
}

//POSTHomework 发布作业
func POSTHomework(c *gin.Context) {
    Poster := login.User.StuName
    H := new(Homework)
    err := c.Bind(H)
    if err != nil {
        fmt.Println("文章数据绑定失败err：", err)
        return
    }
    H.Poster = Poster
    H.PostTime = time.Now()
    H.ClassMark = c.Param("ClassMark")
    err = CreateHomework(H)
    if err == nil {
        //c.HTML(http.StatusOK, "home.html", nil)
        //c.JSON(http.StatusOK, gin.H{"outcome:": "ok"})
        ClassMark := c.Param("ClassMark")
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/classroom/"+ClassMark)
    } else {
        c.JSON(http.StatusBadGateway, gin.H{"结果：": "发布失败！"})
    }
}

//ONEHomework 单个作业详情
func ONEHomework(c *gin.Context) {
    homework := new(Homework)
    ID := c.Param("ID")
    id, _ := strconv.Atoi(ID)
    err := Gorm.Db.Where("ID=?", id).First(homework).Error
    if err == nil {
        Answers, err := GetAnswer(id)
        if err != nil {
            fmt.Println("评论回答失败err", err)
        }
        c.HTML(200, "onehomework.html", gin.H{
            "Title":     homework.Title,
            "Content":   homework.Content,
            "PostTime":  homework.PostTime,
            "username":  login.User.StuName,
            "ID":        homework.ID,
            "Answers":   Answers,
            "ClassMark": homework.ClassMark,
        })
    } else {
        fmt.Println(err)
        c.JSON(http.StatusOK, gin.H{"结果：": "编辑失败！"})
    }
}

//DeleteHomework 删除作业
func DeleteHomework(c *gin.Context) {
    ID := c.Param("ID")
    id, err := strconv.Atoi(ID)
```

```
    if err != nil {
        fmt.Println("转换IDerr：", err)
    }
    err = deleteHomework(id)
    if err != nil {
        fmt.Println("删除回答失败err:", err)
        return
    } else {
        //c.JSON(http.StatusOK, gin.H{"结果：": "删除成功！"})
        ClassMark := c.Param("ClassMark")
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/classroom/"+ClassMark)
    }
}
```

3) answer

```
type answer struct {
    ID         int       `gorm:"column:ID" form:"ID" json:"ID,omitempty"`
    HomeworkId int       `gorm:"column:HomeworkId" form:"HomeworkId"
json:"HomeworkId,omitempty"`
    TheAnswer  string    `gorm:"column:TheAnswer" form:"TheAnswer"
json:"TheAnswer,omitempty"`
    Student    string    `gorm:"column:Student" form:"Student"
json:"Student,omitempty"`
    AnswerTime time.Time `gorm:"column:AnswerTime" form:"AnswerTime"
json:"AnswerTime"`
}

//CreateAnswer 新增回答
func CreateAnswer(A *answer) (err error) {
    err = Gorm.Db.Save(A).Error
    if err != nil {
        fmt.Println("新增回答失败err：", err)
        return err
    }
    return nil
}

//deleteAnswer 删除回答
func deleteAnswer(ID int) (err error) {
    err = Gorm.Db.Where("ID=?", ID).Delete(&answer{}).Error
    if err != nil {
        fmt.Println("删除回答失败err：", err)
        return err
    }
    return nil
}

//GetAnswer 获取回答
func GetAnswer(id int) (answers []answer, err error) {
    err = Gorm.Db.Where("HomeworkId = ? ", id).Find(&answers).Error
    if err != nil {
        fmt.Println("获取回答失败err：", err)
        return nil, err
    }
```

```
    return answers, nil
}

//POSTAnswer 发布回答的POST请求
func POSTAnswer(c *gin.Context) {
    Answer := new(answer)
    err := c.Bind(Answer)
    if err != nil {
        fmt.Println("评论数据绑定失败err：", err)
        return
    }
    Answer.Student = login.User.StuName
    ID := c.Param("ID")
    id, _ := strconv.Atoi(ID)
    Answer.HomeworkId = id
    Answer.AnswerTime = time.Now()
    err = CreateAnswer(Answer)
    if err == nil {
        //c.HTML(http.StatusOK, "home.html", nil)
        //c.JSON(http.StatusOK, gin.H{"结果：": "评论成功！"})
        ClassMark := c.Param("ClassMark")
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/classroom/"+ClassMark+"/onehomework/"+ID)
    } else {
        c.JSON(http.StatusOK, gin.H{"结果：": "提交失败！"})
    }
}

//DeleteAnswer 删除回答
func DeleteAnswer(c *gin.Context) {
    ID := c.Param("answerID")
    id, _ := strconv.Atoi(ID)
    err := deleteAnswer(id)
    if err != nil {
        fmt.Println("删除话题失败err:", err)
        return
    } else {
        HomeworkId := c.Param("ID")
        ClassMark := c.Param("ClassMark")
        //c.JSON(http.StatusOK, gin.H{"结果：": "删除成功！"})
        c.Redirect(http.StatusMovedPermanently,
"http://127.0.0.1:9090/CLASS/classroom/"+ClassMark+"/onehomework/"+HomeworkId)
    }
}
```

4)聊天功能

(这部分是借鉴的，还未搞懂)

```
//connection部分
package chat

import (
    "encoding/json"
    "fmt"
    "github.com/gorilla/websocket"
```

```
    "net/http"
)

type connection struct {
    ws   *websocket.Conn
    sc   chan []byte
    data *Data
}

var wu = &websocket.Upgrader{ReadBufferSize: 512,
    WriteBufferSize: 512, CheckOrigin: func(r *http.Request) bool { return true
}}

func myws(w http.ResponseWriter, r *http.Request) {
    ws, err := wu.Upgrade(w, r, nil)
    if err != nil {
        return
    }
    c := &connection{sc: make(chan []byte, 256), ws: ws, data: &Data{}}
    h.r <- c
    go c.writer()
    c.reader()
    defer func() {
        c.data.Type = "logout"
        user_list = del(user_list, c.data.User)
        c.data.UserList = user_list
        c.data.Content = c.data.User
        data_b, _ := json.Marshal(c.data)
        h.b <- data_b
        h.r <- c
    }()
}

func (c *connection) writer() {
    for message := range c.sc {
        c.ws.WriteMessage(websocket.TextMessage, message)
    }
    c.ws.Close()
}

var user_list = []string{}

func (c *connection) reader() {
    for {
        _, message, err := c.ws.ReadMessage()
        if err != nil {
            h.r <- c
            break
        }
        json.Unmarshal(message, &c.data)
        switch c.data.Type {
        case "login":
            c.data.User = c.data.Content
            c.data.From = c.data.User
            user_list = append(user_list, c.data.User)
            c.data.UserList = user_list
            data_b, _ := json.Marshal(c.data)
            h.b <- data_b
```

```
        case "user":
            c.data.Type = "user"
            data_b, _ := json.Marshal(c.data)
            h.b <- data_b
        case "logout":
            c.data.Type = "logout"
            user_list = del(user_list, c.data.User)
            data_b, _ := json.Marshal(c.data)
            h.b <- data_b
            h.r <- c
        default:
            fmt.Print("========default================")
        }
    }
}

func del(slice []string, user string) []string {
    count := len(slice)
    if count == 0 {
        return slice
    }
    if count == 1 && slice[0] == user {
        return []string{}
    }
    var n_slice = []string{}
    for i := range slice {
        if slice[i] == user && i == count {
            return slice[:count]
        } else if slice[i] == user {
            n_slice = append(slice[:i], slice[i+1:]...)
            break
        }
    }
    fmt.Println(n_slice)
    return n_slice
}

//data部分
package chat

type Data struct {
    Ip       string   `json:"ip"`
    User     string   `json:"user"`
    From     string   `json:"from"`
    Type     string   `json:"type"`
    Content  string   `json:"content"`
    UserList []string `json:"user_list"`
}

//hub部分
package chat

import "encoding/json"

var h = hub{
    c: make(map[*connection]bool),
    u: make(chan *connection),
    b: make(chan []byte),
```

```
    r: make(chan *connection),
}

type hub struct {
    c map[*connection]bool
    b chan []byte
    r chan *connection
    u chan *connection
}

func (h *hub) run() {
    for {
        select {
        case c := <-h.r:
            h.c[c] = true
            c.data.Ip = c.ws.RemoteAddr().String()
            c.data.Type = "handshake"
            c.data.UserList = user_list
            data_b, _ := json.Marshal(c.data)
            c.sc <- data_b
        case c := <-h.u:
            if _, ok := h.c[c]; ok {
                delete(h.c, c)
                close(c.sc)
            }
        case data := <-h.b:
            for c := range h.c {
                select {
                case c.sc <- data:
                default:
                    delete(h.c, c)
                    close(c.sc)
                }
            }
        }
    }
}
```