## SQL

我们不是第一次接触了 所以我想以一种面向使用场景的方式来进行记录

你其实不需要知道什么是 DDL DML这些 也不需要知道如何create a database 这些并不能帮助你了解和认识我们数据库

这里我们推荐你使用navicat进行数据库可视化的创建和管理 在vscode中是使用sql notebook编写sql语句

## 登录和注册

我们规定接口格式如下

/login/?username=admin&password=123456  
/register/?username=admin&password=123456

首先我们应该明白,对于像密码格式检查比如只允许字母+数组 不能存在特殊字符 像这种应该在客户端实现 这样的好处在于可以更加快速检测错误 还可以减轻服务端的压力.

包括像对于password之类的重要信息进行加密 都可以将其放到客户端 这里应该使用非对称的加密算法 这样即使数据库的数据被他人盗窃,用户的密码也不会丢失.

### 表的设计

我们可以很自然的设计出这样的表格式.

CREATE TABLE `users` (
  
 `id` int(0) NOT NULL AUTO\_INCREMENT,
  
 `username` varchar(32) CHARACTER NOT NULL,
  
 `password` varchar(32) CHARACTER NOT NULL,
  
 PRIMARY KEY (`id`) USING BTREE
  
)

### 注册功能的实现

// 这里部分的知识是web开发的也就顺带着写一下吧  
func Register(c \*gin.Context) {  
 // 获取query的参数  
 username := c.Query("username")  
 password := c.Query("password")  
  
 // 查询数据库中是否存在该用户  
 var user model.User  
 dao.DB.Where("username = ?", username).First(&user)  
 if user.Id != 0 {  
 c.JSON(200, gin.H{  
 "code": "1",  
 "message": "该用户名已被使用",  
 })  
 return  
 } else {  
 dao.DB.Create(&model.User{Username: username, Password: password})  
 c.JSON(200, gin.H{  
 "code": "0",  
 "message": "注册成功",  
 })  
 return  
 }  
}

这里的逻辑是很简单的 先利用username查询数据库中是否存在数据 如果没有则插入数据 注册成功 反之返回注册失败 该用户名已被使用

### 登录

一开始我是这样写的

func Login(c \*gin.Context) {  
 // 获取query的参数  
 username := c.Query("username")  
 password := c.Query("password")  
 // 查询数据库中是否存在该用户  
 var user model.User  
 dao.DB.Where("username = ? and password = ?", username, password).First(&user)   
 if user.Id != 0 {  
 c.JSON(200, gin.H{  
 "code": "0",  
 "message": "登录成功",  
 })  
 return  
 } else {  
 c.JSON(200, gin.H{  
 "code": "1",  
 "message": "用户名或密码错误",  
 })  
 return  
 }  
}

func Login(c \*gin.Context) {  
 // 获取query的参数  
 username := c.Query("username")  
 password := c.Query("password")  
 // 查询数据库中是否存在该用户  
 var user model.User  
 dao.DB.Where("username = ?", username).First(&user)  
 if user.Id != 0 {  
 if user.Password == password {  
 c.JSON(200, gin.H{  
 "code": "0",  
 "message": "登录成功",  
 })  
 return  
 } else {  
 c.JSON(200, gin.H{  
 "code": "1",  
 "message": "密码错误",  
 })  
 return  
 }  
 } else {  
 c.JSON(200, gin.H{  
 "code": "1",  
 "message": "用户不存在",  
 })  
 return  
 }  
}

我个人觉得后者更好,首先两种方法都只查询了一次数据库,性能上相差不大.后者有更加详细的错误提示,可以引导用户进行正确的操作.

第一种用户就无法知道自己是密码还是账号错误了,就不知道是该检查账号是否正确还是选择忘记密码.

## 视频列表

视频列表格式如下

CREATE TABLE `videos` (
  
 `id` int(0) NOT NULL AUTO\_INCREMENT,
  
 `likes` int(0) UNSIGNED NOT NULL DEFAULT 0,
  
 `comments` int(0) UNSIGNED NOT NULL DEFAULT 0,
  
 `url` varchar(255) NOT NULL,
  
 `title` varchar(255) NOT NULL,
  
 PRIMARY KEY (`id`) USING BTREE
  
)

我们的视频数据有很多可能是上万条或者更多,但是用户肯定是不需要这么多的,我们每次只发送5条数据给用户即可

我这里是根据id来进行排序 你也可以添加比如时间字段 然后按照时间的先后顺序进行发送

dao.DB.Where("id > ?", last\_time\_id).Limit(5).Find(&videos)

## 用户视频列表

这里很明显是一个多表结构了 准确讲是一对多,可能一个用户有多条视频,也可能一条都没有

这里我们需要重新定义我们的表结构 当然了 其实只需要修改视频表就够了

CREATE TABLE `videos` (
  
 `id` int(0) NOT NULL AUTO\_INCREMENT,
  
 `user\_id` int(0) UNSIGNED NOT NULL DEFAULT 0,
  
 `likes` int(0) UNSIGNED NOT NULL DEFAULT 0,
  
 `comments` int(0) UNSIGNED NOT NULL DEFAULT 0,
  
 `url` varchar(255) NOT NULL,
  
 `title` varchar(255) NOT NULL,
  
 PRIMARY KEY (`id`) USING BTREE
  
)

其实也只是多加了一个user\_id字段,但是就是这一个字段,就可以让用户表和视频表产生关联

比如投稿接口/publish?userid&title 实现如下

func Publish(c \*gin.Context) {
  
 //获取参数
  
 title := c.Query("title")
  
 userid := c.Query("userid")
  
  
 //插入数据库
  
 dao.DB.Table("videos").Create(&model.Video{
  
 Title: title,
  
 UserID: userid,
  
 URL: "http://www.baidu.com",
  
 Likes: 0,
  
 Comments: 0,
  
 })
  
 c.JSON(200, gin.H{
  
 "code": "0",
  
 "message": "发布成功",
  
 })
  
}

## 评论列表

我们的数据库的cell是平行的关系 但是我们的评论是嵌套关系(因为存在回复该条评论的评论 不过一般不会出现更深层级的评论了)

你可能会说 用一个字符列表去存储评论不就好了 但是你要知道在关系型数据库中 每个单元的长度是固定的 如果你的二级评论很多 就会出现问题

{  
 "status\_code": 0,  
 "status\_msg": "string",  
 "comment\_list": [  
 {  
 "id": 0,  
 "user": {  
 "id": 0,  
 "name": "string",  
 "follow\_count": 0,  
 "follower\_count": 0,  
 "is\_follow": true  
 },  
 "content": "string",  
 "create\_date": "string"  
 }  
 ]  
}