## 项目目录规范

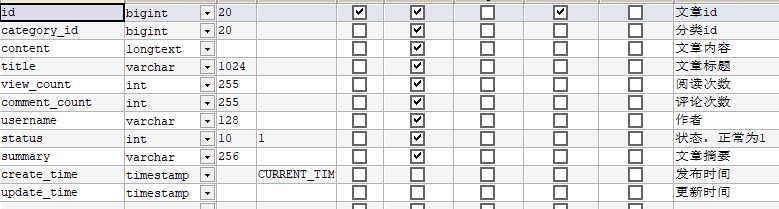


* model：实体
* dao：数据层
* service：业务逻辑
* controller：页面控制相关
* static：css js
* utils：工具
* views：HTML模板
* main.go：入口，定义路由

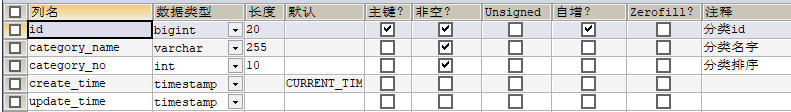
## 需求分析

## 数据库分析

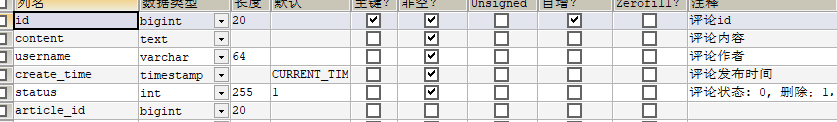
* article文章表



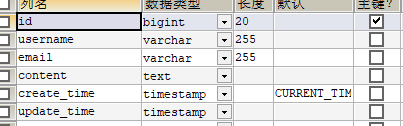
* category：分类表



* 评论表comment



* leave留言表



## 主页的实现

### 实体类

* 分类的结构体
* 文章的结构体（可以不写文章内容，因为是大文本）
* model/category.go

|  |
| --- |
| **package** model  *//`id``category\_name``category\_no``create\_time``update\_time`  // 定义分类结构体* **type** Category **struct** {  CategoryId int64 **`db:"id"`** CategoryName string **`db:"category\_name"`** CategoryNo int **`db:"category\_no"`** } |

* model/article.go

|  |
| --- |
| **package** model  **import "time"** *//``id``category\_id``content``title``view\_count``comment\_count``username``STATUS``summary``create\_time``update\_time``  // 定义文章结构体* **type** ArticleInfo **struct** {  Id int64 **`db:"id"`** CategoryId int64 **`db:"category\_id"`** *// 文章摘要* Summary string **`db:"summary"`** Title string **`db:"title"`** ViewCount uint32 **`db:"view\_count"`** *// 时间* CreateTime time.Time **`db:"create\_time"`** CommentCount uint32 **`db:"comment\_count"`** Username string **`db:"username"`** }  *// 用于文章详情页的实体 // 为了提升效率* **type** ArticleDetail **struct** {  ArticleInfo  *// 文章内容* Content string **`db:"content"`** Category }  *// 用于文章上下页* **type** ArticleRecord **struct** {  ArticleInfo  Category } |

### 数据层

* init()数据库初始化函数
* 分类相关的操作（添加，查询，查1个分类，查多个分类，查所有分类）
* 文章相关的操作（添加文章（投稿），查询所有文章，根据文章id查看内容）
* dao/db/db.go

|  |
| --- |
| **package** db  **import** (  \_ **"github.com/go-sql-driver/mysql"  "github.com/jmoiron/sqlx"** )  **var** (  DB \*sqlx.DB )  *// 初始化* **func** Init(dns string) error {  **var** err error  DB, err = sqlx.Open(**"mysql"**, dns)  **if** err != nil {  **return** err  }  *// 查看是否连成功* err = DB.Ping()  **if** err != nil {  **return** err  }  DB.SetMaxOpenConns(100)  DB.SetMaxIdleConns(16)  **return** nil } |

* dao/db/category.go

|  |
| --- |
| **package** db  **import** (  **"github.com/jmoiron/sqlx"  "go\_5\_blog/model"** )  *// 添加分类* **func** InsertCategory(category \*model.Category) (categoryId int64, err error) {  sqlstr := **"insert into category(category\_name,category\_no) value(?,?)"** result, err := DB.Exec(sqlstr, category.CategoryName, category.CategoryNo)  **if** err != nil {  **return** }  categoryId, err = result.LastInsertId()  **return** }  *// 获取单个分类* **func** GetCategoryById(id int64) (category \*model.Category, err error) {  category = &model.Category{}  sqlstr := **"select id ,category\_name,category\_no from category where id = ?"** err = DB.Get(category, sqlstr, id)  **return** }  *// 获取多个分类* **func** GetCategoryList(categoryIds []int64) (categoryList []\*model.Category, err error) {  sqlstr, args, err := sqlx.In(**"select id,category\_name,category\_no from category where id in(?)"**, categoryIds)  **if** err != nil {  **return** }  *// 查询* err = DB.Select(&categoryList, sqlstr, args...)  **return** }  *// 获取所有分类* **func** GetAllCategoryList() (categoryList []\*model.Category, err error) {  sqlstr := **"select id ,category\_name,category\_no from category order by category\_no asc"** err = DB.Select(&categoryList, sqlstr)  **return** } |

* category\_test.go

|  |
| --- |
| **package** db  **import "testing"  func** init() {  *// parseTime=true 将mysql中时间类型，自动解析为go结构体中的时间类型  // 不加报错* dns := **"root:admin@tcp(localhost:3306)/blogger?parseTime=true"** err := Init(dns)  **if** err != nil {  panic(err)  } }  *// 获取单个分类信息* **func** TestGetCategoryById(t \*testing.T) {  category, err := GetCategoryById(1)  **if** err != nil {  panic(err)  }  t.Logf(**"category:%#v"**, category) }  **func** TestGetCategoryList(t \*testing.T) {  **var** categoryIds []int64  categoryIds = append(categoryIds, 1, 2, 3)  categoryList, err := GetCategoryList(categoryIds)  **if** err != nil {  panic(err)  }  **for** \_, v := **range** categoryList {  t.Logf(**"id:%d category:%#v\n"**, v.CategoryId, v)  } }  **func** TestGetAllCategoryList(t \*testing.T) {  categoryList, err := GetAllCategoryList()  **if** err != nil {  panic(err)  }  **for** \_, v := **range** categoryList {  t.Logf(**"id:%d category:%#v\n"**, v.CategoryId, v)  } } |

* article.go

|  |
| --- |
| **package** db  **import** (  \_ **"github.com/go-sql-driver/mysql"  "go\_5\_blog/model"** )  *// 插入文章* **func** InsertArticle(article \*model.ArticleDetail) (articleId int64, err error) {  *// 加个验证* **if** article == nil {  **return** }  *//``id``category\_id``content``title``view\_count``comment\_count``username``STATUS``summary``create\_time``update\_time``* sqlstr := **`insert into  article(content,summary,title,username,category\_id,view\_count,comment\_count) values(?,?,?,?,?,?,?)`** result, err := DB.Exec(sqlstr, article.Content, article.Summary, article.Title, article.Username, article.ArticleInfo.CategoryId,  article.ArticleInfo.ViewCount, article.ArticleInfo.CommentCount)  **if** err != nil {  **return** }  articleId, err = result.LastInsertId()  **return** }  *// 获取文章列表，做个分页* **func** GetAricleList(pageNum, pageSize int) (articleList []\*model.ArticleInfo, err error) {  **if** pageNum < 0 || pageSize <= 0 {  **return** }  *// 时间降序排序  //``id``category\_id``content``title``view\_count``comment\_count``username``STATUS``summary``create\_time``update\_time``* sqlstr := **`select  id,summary,title,view\_count,create\_time,comment\_count,username,category\_id  from  article  where  status = 1  order by create\_time desc  limit ?,?`** err = DB.Select(&articleList, sqlstr, pageNum, pageSize)  **return** }  *// 根据文章id，查询单个文章* **func** GetArticleDetail(articleId int64) (articleDetail \*model.ArticleDetail, err error) {  **if** articleId < 0 {  **return** }  articleDetail = &model.ArticleDetail{}  sqlstr := **`select  id,summary,title,view\_count,content,create\_time,comment\_count,username,category\_id  from  article  where  id = ?  and  status = 1  `** err = DB.Get(articleDetail, sqlstr, articleId)  **return** }  *// 根据分类id，查询这一类的文章* **func** GetArticleListByCategoryId(categoryId, pageNum, pageSize int) (articleList []\*model.ArticleInfo, err error) {  **if** pageNum < 0 || pageSize <= 0 {  **return** }  sqlstr := **`select  id,summary,title,view\_count,create\_time,comment\_count,username,category\_id  from  article  where  status = 1  and  category\_id = ?  order by create\_time desc  limit ?,?`** err = DB.Select(&articleList, sqlstr, categoryId, pageNum, pageSize)  **return** } |

* article\_test.go

|  |
| --- |
| **package** db  **import** (  **"go\_5\_blog/model"  "testing"  "time"** )  **func** init() {  *// parseTime=true 将mysql中时间类型，自动解析为go结构体中的时间类型  // 不加报错* dns := **"root:admin@tcp(localhost:3306)/blogger?parseTime=true"** err := Init(dns)  **if** err != nil {  panic(err)  } }  *// 测插入文章* **func** TestInsertArticle(t \*testing.T) {  *// 构建对象* article := &model.ArticleDetail{}  article.ArticleInfo.CategoryId = 1  article.ArticleInfo.CommentCount = 0  article.Content = **"abc fdlksafjdlajflk fdjlasjfdkljwa"** article.ArticleInfo.CreateTime = time.Now()  article.ArticleInfo.Title = **"5qi go"** article.ArticleInfo.Username = **"sun"** article.ArticleInfo.Summary = **"abc fd"** article.ArticleInfo.ViewCount = 1  articleId, err := InsertArticle(article)  **if** err != nil {  **return** }  t.Logf(**"articleId : %d\n"**, articleId) }  **func** TestGetAricleList(t \*testing.T) {  articleList, err := GetAricleList(1, 15)  **if** err != nil {  **return** }  t.Logf(**"article : %d\n"**, len(articleList)) }  **func** TestGetArticleDetail(t \*testing.T) {  ar,err := GetArticleDetail(1)  **if** err != nil {  **return** }  t.Logf(**"article %#v\n"**, ar) } |

### 业务逻辑层

* 获取分类列表（用于展现分类云）
* 获取所有文章和对应的分类信息
* 根据分类id，获取该类文章和他们对应的分类信息
* category.go

|  |
| --- |
| **package** service  **import** (  **"go\_5\_blog/dao/db"  "go\_5\_blog/model"** )  *// 获取所有分类* **func** GetALLCategoryList() (categoryList []\*model.Category, err error) {  categoryList, err = db.GetAllCategoryList()  **if** err != nil {  **return** }  **return** } |

* article.go

|  |
| --- |
|  |

### controller层

### 项目入口

## 投稿的实现

### 业务逻辑层

### controller层

## 文章详情页的实现

### 实体类

### 数据层

### 业务逻辑层

### controller层

## 文章评论的实现

### 实体类

### 数据层

### 业务逻辑层

### controller层

## 博客留言的实现

### 实体类

### 数据层

### 业务逻辑层

### controller层