南開大學
Nankai University

计算机与网络空间安全学院

互联网数据库开发大作业

实现文档

姓名：杨鑫 孟笑朵

学号：2011028 2010349

专业：信息安全计算机科学与技术

2023 年 2 月 14 日

目录

1 任务分工

- 杨鑫：参与前期讨论、参与数据库设计、完成前后台模板的收集、完成前端主页部分数据的展示、完成主页除文章页面的其他页面开发与调试、完成后台除文章管理页面的其他页面的开发与调试，完成部署文档，需求文档。
- 孟笑朵：参与前期讨论、参与数据库设计、完成前后台模板的收集、完成前端主页部分数据的展示、完成主页文章页面的开发与调试、完成后台文章管理页面的开发与调试，完成设计文档，项目展示 PPT。
- 两人共同完成实现文档，用户手册。

2 提交记录

本次小组因为只有两个人开发所以日程管理工具使用微信 + 腾讯会议进行代替，同时本小组使用 Git 工具进行代码管理，在仓库中建立主分支，然后杨鑫维护 ydevelop 分支，孟笑朵维护 mdevelop 分支，两个分支各自开发不影响主分支代码，然后到一定时间再进行 git merge 到主分支。接下来一次展示克隆以及 Git 提交记录截图：

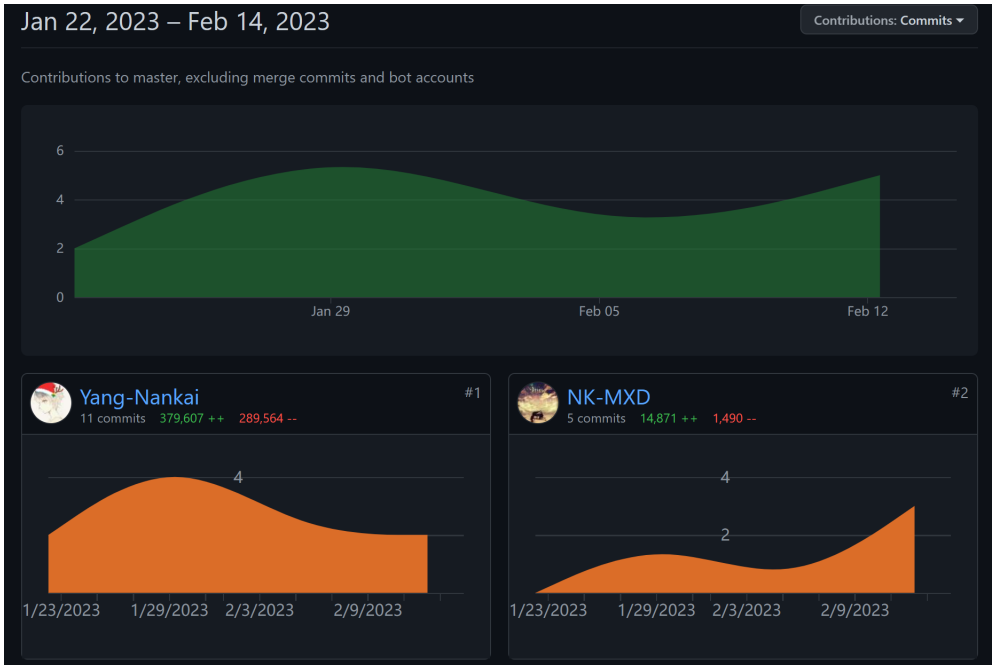

图 2.1: Git 提交记录图

3 主要代码以及目录展示

3.1 整体展示

整体代码目录展示，如图所示：

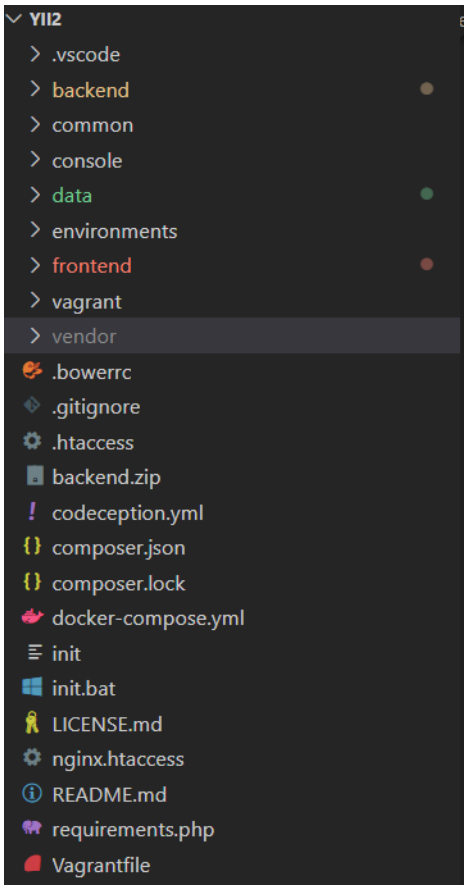

图 3.2: 整体代码目录展示

- backend: 后端独立应用模块
  - assets/ 包含应用程序资产，如 JavaScript 和 CSS JavaScript and CSS
  - config/ 包含后端配置
  - controllers/ 包含 Web 控制器类
  - models/ 包含后端特定的模型类 classes
  - runtime/ 包含运行时生成的文件
  - tests/ 包含后端应用程序的测试
  - views/ 包含 Web 应用程序的视图文件
  - web/ 包含条目脚本和 web 资源
- common: 公共文件模块 (配置，类)
  - config/ 包含共享配置
  - mail/ 包含电子邮件的视图文件
  - models/ 包含后端和前端中使用的模型类
  - tests/ 包含公共类的测试
- console：控制台文件模块

  - config/ 包含控制台配置
  - controllers/ 包含控制台控制器（命令）
  - migrations/ 包含数据库迁移
  - models/ 包含控制台特定的模型类
  - runtime/ 包含运行时生成的文件
- runtime：缓存文件模块 (linux 部署需要这个模块可写)
- frontend：包含前端配置
  - assets/ 包含应用程序资产，如 JavaScript 和 CSS
  - config/ 包含前端配置
  - controllers/ 包含 Web 控制器类
  - models/ 包含前端特定的模型类
  - runtime/ 包含运行时生成的文件
  - tests/ 包含前端应用程序的测试
  - views/ 包含 Web 应用程序的视图文件
  - web/ 包含条目脚本和 web 资源
- vendor：包含依赖第三方软件包
- environments：包含基于环境的覆盖
- data：包含小组成员个人作业与团队作业以及部署需要的数据库部署 sql 文件

3.2 前台展示

前台主要代码如下：

- controllers/：控制器
  - SiteController.php 系统前台首页控制器
  - UserCommentController.php 系统留言面板控制器
- models/：模型
  - CommentForm.php 评论模型
  - ContactForm.php 联系我们模型
  - EntryForm.php 入口模型
  - LoginForm.php 登录模型
  - PasswordResetRequestForm.php 密码重置请求模型
  - ResendVerificationEmailForm.php 重新发送验证码模型
  - ResetPasswordForm.php 重置密码模型
  - SignupForm.php 注册模型

  - User.php 用户模型
  - UserComment.php 用户评论模型
  - UserCommentSearch.php 用户评论搜索模型
  - VerifyEmailForm.php 验证邮箱模型
- views/：视图
  - layouts/newMain.php 前台使用的布局模板
  - site/about.php 关于我们页面
  - site/blog.php 博客页面
  - site/blog-detail.php 博客细节页面
  - site/comment.php 留言页面
  - site/contact.php 联系我们页面
  - site/entry.php 入口页面
  - site/error.php 报错页面
  - site/index.php 主页面
  - site/login.php 登录页面
  - site/personal.php 个人作业界面
  - site/signup.php 注册页面
  - site/team.php 团队作业页面
  - site/requestPasswordResertToken.php 重置密码请求页面
  - site/resendVerificationEmail.php 重发验证邮箱页面
  - site/resetPassword.php 重置密码页面
  - user-comment/index.php 留言板页面

3.2.1 前台模板调试

选择使用 absolute 模板，按照老师课上教授的模板使用方法，static 文件夹放在 frontend/web 目录下，newMain.php 放在 frontend/views/layouts 目录下，AppAsset 放在 frontend/assets 目录下然后 frontend 目录下所有的 controllers 类中都加上 public $layout = 'newMain'，如果某个页面没有使用 newMain 模板，则在相应控制器中具体对应这个页面的 action 中指定使用的模板，方法为 $this->layout="newMain"。

```
class UserCommentController extends Controller
{
    // 设置默认布局
    public $layout = 'newMain';
    ...
}
```

布局主要包括通用 js、css 文件的引入，页头和页脚的样式，导航栏和面包板菜单的样式等。主要实现效果如图所示

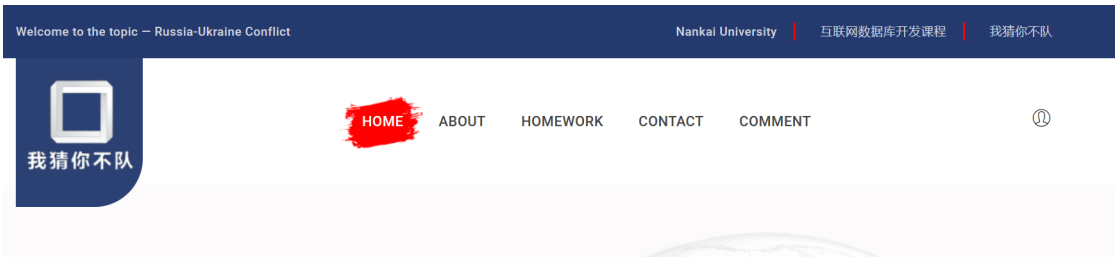

图 3.3: 默认布局页头效果图

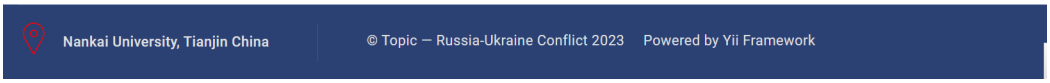

图 3.4: 默认布局页脚效果图

3.2.2 前台主页模块

第一屏使用 CSS 中的 animation 产生动态效果，同时设置往下更多按钮，点击后可以自动跳转到下面内容。

图 3.5: 第一屏效果图

其中 CSS 样式规定为:

```
.banner_keyframes_animation {
  -webkit-animation-duration: 3s;
  animation-duration: 3s;
  -webkit-animation-timing-function: cubic-bezier(0.54, 0.085, 0.5, 0.92);
  animation-timing-function: cubic-bezier(0.54, 0.085, 0.5, 0.92);
  -webkit-animation-name: animateUpDown;
```

```
    animation-name: animateUpDown;
    -webkit-animation-iteration-count: infinite;
    animation-iteration-count: infinite;
    -webkit-transition: all 0.4s cubic-bezier(0.645, 0.045, 0.355, 1);
    transition: all 0.4s cubic-bezier(0.645, 0.045, 0.355, 1);
  }
```

同时对于往下阅读更多使用了如下 JS 中的 window.scrollTo 函数进行实现。另外俄乌冲突重要事件使用了纵轴时间线的样式，如图：

图 3.6: 重要事件时间线

另外对于轮播图展示以及新闻博客展示采用的方式是嵌套 html，使用 js 中的 jQuery 以及 swiper 插件，将从数据库中取出的文章信息以轮播方式展示在系统主页界面。同时左下角还有一个回到顶部的按钮，采用的是 jQuery 中的 scrollup 插件，点击后回到主页的顶部。

3.2.3 前台留言模块

使用 Gii 生成 CRUD 代码，该模块仅仅是作为 SiteController 的一个工具，再 SiteController 中通过判断是否登录然后才能进入留言页面进行查看，然后再利用 UserComment 模型进行查看状态为 1 即为显示评论的评论，然后进行获取数据以及分页显示。

首先创建 comment 的页面，这里分页使用 LinkPager 工具中的 widget，如下：

```
<?php echo LinkPager::widget([
    'pagination' => $pages,
    'nextPageLabel' => false,
    'prevPageLabel' => false,
    'options' => ['class' => 'pagination']
    ]);
?>
```

同时将从控制器传过来的数据通过 foreach 依次显示在页面上，同时对于数据库上的时间戳还需要将其转为年月日-时分秒的形式，如下：

```
<?php
foreach($models as $model):
    ?>
    <div class="container comment  m-1 p-0">
        <div class="ml-3 border-left ">
            <div class="container ml-2">
                <span><?php echo Html::encode($model->username);?></span>
                <span class="summary-text small">
                    <?php
                        $datetime = new \DateTime();
                        $datetime->setTimestamp($model->created_at);
                        echo $datetime->format('Y-m-d H:i:s');
                    ?>
                </span>
            </div>
            <div class="messageText ml-2 container m-2"><?php echo Html::encode($model->commen
        </div>
    </div>
    <?php
endforeach;
?>
```

然后在 siteController 中新增 comment 操作，针对于没有登陆的需要跳转到登录，针对于已经登录的就从 UserComment 模型中取得数据，并通过 Pagination 类进行分页处理，然后将经过分页处理后的数据通过模型传到 comment 页面进行渲染，如下：

```
public function actionComment()
{
    if (Yii::$app->user->isGuest) {
        echo "<script language=\"JavaScript\">alert(\" 您还未登录，请登录!\");</script>";
        $model = new LoginForm();
        if ($model->load(Yii::$app->request->post()) && $model->login()) {
            return $this->goBack();
        }
        $model->password = '';

        return $this->render('login', [
            'model' => $model,
        ]);
```

```
14      } else {
15          $comments = UserComment::find()->where(['status' =>1 ]);
16          $countComments = clone $comments;
17          // VarDumper::dump($countComments->count());
18          $pages = new Pagination([
19              'totalCount' => $comments->count(),
20              'pageSize' => 10,
21          ]);
22
23          $user_comment = new CommentForm();
24          if ($user_comment->load(Yii::$app->request->post()) && $user_comment->comment()) {
25              return $this->refresh();
26          }
27          //获取数据
28          $models = $comments->offset($pages->offset)->limit($pages->limit)->all();
29
30          // VarDumper::dump($comments);
31          return $this->render('comment', [
32              'models' => $models,
33              'pages' => $pages,
34              'comment' => $user_comment,
35          ]);
36      }
37  }
```

最后实现的效果如下:

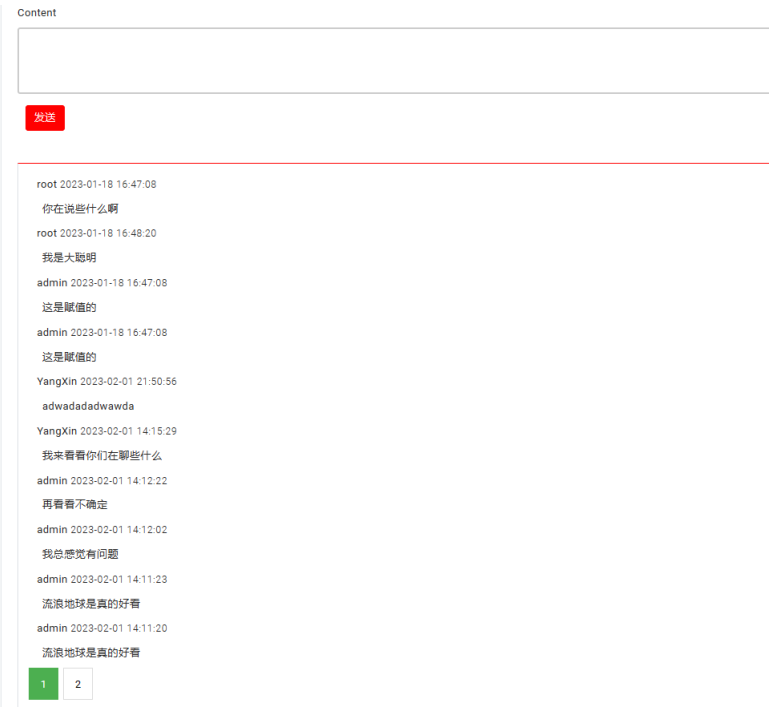

图 3.7: 留言面板

3.2.4 前台文章展示

这部分包括展示文章页面以及对应的从数据库中调用相应的文章数据，如下所示为我们设计的文章详情页面，点击如下页面位置进入文章详情页:

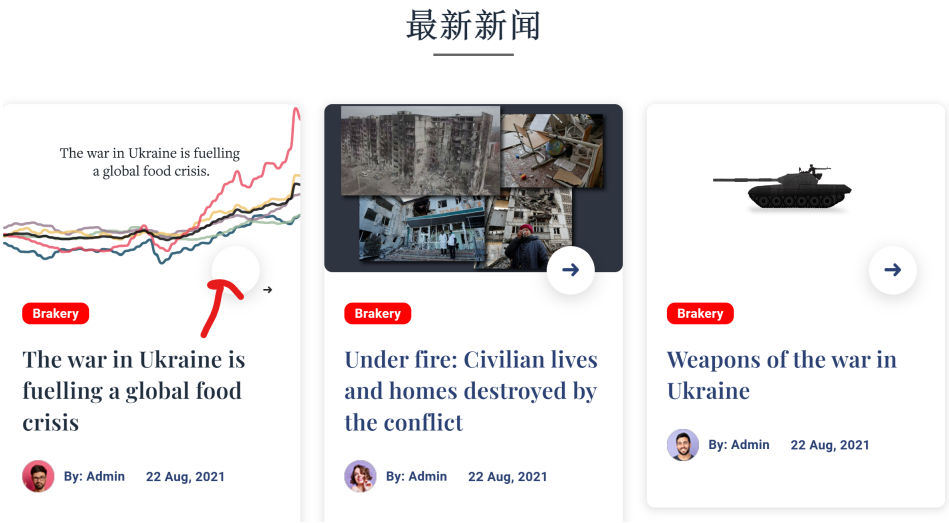

图 3.8: 文章详情页面的展示

图 3.9: 文章详情页面的展示

其中文章留言板块的设计如下所示:

```
iv class="comments_form">
  <div class="comments_title">
      <h2>Leave A Comment</h2>
  </div>
  <div class="comments_form_inner">
      <form action="#">
          <div class="row">
              <div class="col-lg-6 ">
                  <div class="comments_form_input">
                      <input class="border" placeholder="Name *" type="text">
                  </div>
              </div>
              <div class="col-lg-6 ">
                  <div class="comments_form_input">
                      <input class="border" placeholder="Email *" type="text">
                  </div>
              </div>
              <div class="col-12">
                  <div class="comments_form_input">
                      <input class="border" placeholder="Subject (Optinal)"
                          type="text">
                  </div>
              </div>
```

```
                <div class="col-12">
                    <div class="comments_form_input">
                        <textarea class="border" placeholder="Message"></textarea>
                    </div>
                </div>
            </div>
            <button class="btn btn-link" type="submit">Post A Comment</button>
        </form>
    </div>
div>
```

3.3 后台展示

后台主要代码如下:

- controllers/: 控制器
  - BaseController.php 系统后台通用控制器
  - AdminController.php 系统管理员控制器
  - CommentController.php 系统留言控制器
  - PermissionController.php 系统权限控制器
  - PublicController.php 系统公共主页控制器
  - RoleController.php 系统角色控制器
  - RuleController.php 系统规则控制器
  - SystemLogController.php 系统日志控制器
  - UserController.php 系统用户控制器
- models/: 模型
  - UserComment.php 用户评论模型
  - Admin.php 管理员模型
  - Permission.php 全新啊模型
  - LoginForm.php 登录模型
  - Role.php 角色模型
  - Rule.php 规则模型
  - SystemLog.php 系统日志模型
  - User.php 用户模型
- views/: 视图
  - admin/index.php 管理员管理页面
  - comment/index.php 留言管理页面

- permission/index.php 权限管理页面
- public/console.php 主页控制台页面
- public/error.php 报错页面
- public/iframe.php 框架布局页面
- public/login.php 后台登录页面
- role/index.php 角色管理页面
- rule/index.php 规则管理页面
- system-log/index.php 系统日志管理页面
- user/index.php 用户管理

3.3.1 后台模板调试

本后台模板选择使用基于 Layui 开发的 Layui-Admin 模板，如下图，使用方法和前台模板类似，但是需要注意的是需要在模板的 index 文件中找到嵌套 html 的位置，换成配置模板使用的 <?= $content ?> 标签，实现后模板的效果如下：

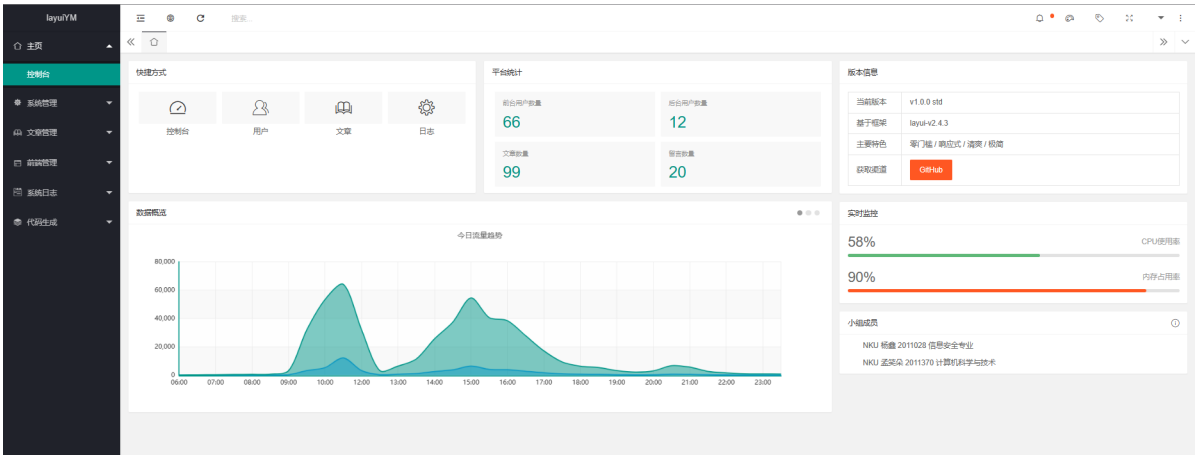

图 3.10: 后台模板效果

3.3.2 首页可视化展示模块

如上图，在后端首页可视化系统的各项数据信息，如文章数、后端用户数、评论数、前台用户数等，使用 layui-admin 前端框架，其中这些数据均通过静态资源，加载 json 文件进行数据加载：

```
<?php
$js = <<<JS
    layui.config({
        base: 'static/layuiadmin/' //静态资源所在路径
    }).extend({
        index: 'lib/index' //主入口模块
    }).use(['index', 'console']);
JS;
```

```
$this->registerJs($js);
?>
```

或者直接通过 Yii2 框架中的模型获得相关的信息，通过 find 和 count 得到数量：

```
<li class="layui-col-xs6">
    <a href="javascript:;" onclick="layer.tips('不跳转', this, {tips: 3});" class="layadmin-bac
        <h3> 留言数量 </h3>
        <p><cite>
            <?php
                $comment = new UserComment();
                $sum = $comment->find()->count();
                echo $sum;
            ?>
        </cite></p>
    </a>
</li>
```

通过上述方法，使用嵌套 php 代码的方式从数据表 model 中查询数据库信息，然后传入 js 中的变量，并通过变量对图表进行数值的设置，以达到可视化显示在后台主页的效果。

3.3.3 后台文章管理

对于文章的管理分为三个部分:

1. 对文章本身进行管理: 包括文章的查找, 添加文章, 修改文章, 删除文章
2. 对文章的分类进行管理: 包括分类的添加, 分类的删除
3. 文章本身进行管理: 对文章评论进行管理: 包括评论的查找, 评论的增删改查操作

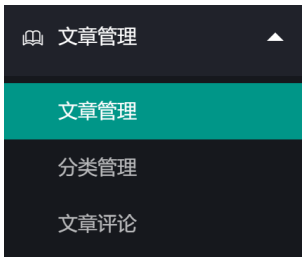

图 3.11: 后台文章管理的部分

后台文章的管理需要用到 YmArticle 类, 该类中需要提供 actionIndex 接口, 该接口既能够对于 Ajax 的动态请求命令做出反应也要能对静态请求命令做出反应, 如下所示编写对应的接口, 可以发现对应 Ajax 动态请求命令, 返回的数据为 json 类型的数据格式;

```
blic function actionIndex()

  $searchModel = new YmArticleSearch();
```

```
if (Yii::$app->request->isAjax){
    Yii::info(" 这里进行了一次 js 搜索");
    $article = (new YmArticle())->find();
    $count = $article->count();
    $page = Yii::$app->request->get('page',1);
    $limit = Yii::$app->request->get('limit',10);
    $author = Yii::$app->request->get('author');
    $title = Yii::$app->request->get('title');
    $data = $article->offset(($page-1)*$limit)-
    >limit($limit)>asArray()->all();
    if ($author){
        $article->where(['like','author',$author]);
    }
    if ($title){
        $article->where(['like','title',$title]);
    }
    $data = $article->offset(($page-1)*$limit)-
    >limit($limit)>asArray()->all();
    // 添加操作
    foreach ($data as &$item){
        $item['viewUrl'] = Url::to(['view','id'=>$item['id']]);/查看按钮
        $item['updateUrl'] =Url::to(['update','id'=>$item['id']]);//更新按钮
        $item['destroyUrl'] =Url::to(['destroy','id'=>$item['id']]);
        // $item['roles'] = Yii::$app-
        >authManager>getRolesByUser($item['id']);
    }
    return $this->asJson([
        'code' => 0,
        'msg' => '请求成功',
        'count' => $count,
        'data' => $data
    ]);
}
else{
    $dataProvider = $searchModel->search($this->request>queryParams);
}

return $this->render('index', [
    'searchModel' => $searchModel,
    'dataProvider' => $dataProvider,
]);

```

针对上述返回的 json 类型的数据格式可以使用 Layui 中特定的模板结构，将对应的数据以表格的形式较为美观的展现出来，如下所示为使用 layui 模板的展现数据的方式，以及对应的使用 js 进行搜索的命令，具体而言，对于文章的搜索，只需要将填写的查找信息传入 json 中，就能利用已有的 actionIndex 接口对对应参数进行查找:

图 3.12: 文章管理界面展示

图 3.13: 文章管理界面展示代码核心逻辑

如下为搜索模块的实现方式:

```
1  //搜索
2  $("#searchBtn").click(function() {
3      var data = {
4          title:$("#title").val(),
5          author:$("#author").val(),
6      }
7      dataTable.reload({
8          where:data,
```

```
            page:{curr:1}
        })
    })
```

如下文章添加界面的设计

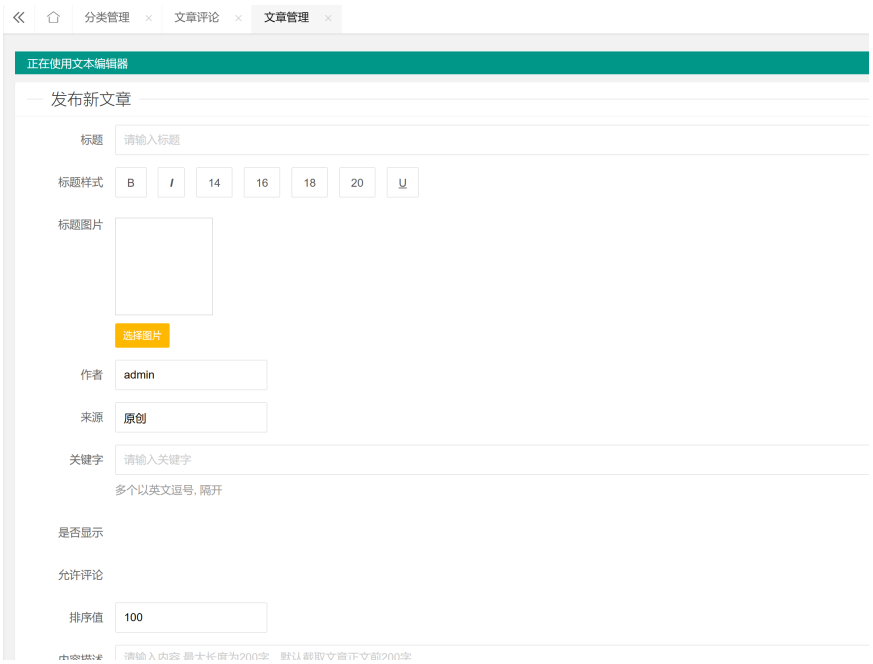

图 3.14: 文章发布界面

```
//搜索
$("#searchBtn").click(function() {
    var data = {
        title:$("#title").val(),
        author:$("#author").val(),
    }
    dataTable.reload({
        where:data,
        page:{curr:1}
    })
})
```

对于文章的评论也是同样的设计模式, 先在 Controller 中定义好对应的 Ajax 动态请求的 actionIndex 接口, 然后将获得的 json 文件进行解析渲染

3.3.4 其他模块生成

这里系统管理模块、前端管理模块、系统日志模块的代码逻辑基本一致，这里以系统管理中的用户管理为例，效果图如下:

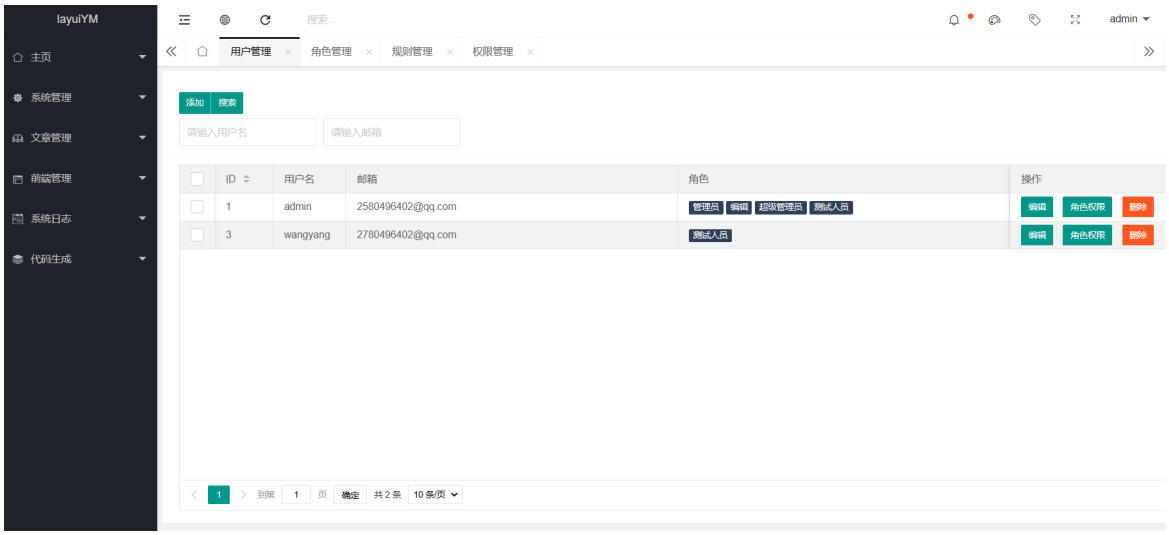

图 3.15: 用户管理效果

可以看到中间主体以表格的形式展示了后台用户的信息比如 ID、用户名、邮箱、角色和操作，首先需要注意的是每个模块均有权限，如果当前用户没有进入该页面的权限，则显示无授权访问，这里的代码逻辑的实现其实是在进入每一个页面之前就进行判断的，因此在 Yii2 框架中使用 beforeAction 最合适，而每一个控制器都需要进行判断，因此需要放在 BaseController 中因为每一个控制器都要去继承它，具体代码如下：

```
1  public function beforeAction($action)
2  {
3      if (!parent::beforeAction($action)){
4          return false;
5      }
6      //检查权限
7      $controller = $action->controller->id;
8      $action = $action->id;
9
10     //保存操作日志
11     if ($controller=='public'){
12         self::saveAdminLog();
13         return true;
14     }
15
16     if (\Yii::$app->user->can($controller.'/*')){
17         self::saveAdminLog();
18         return true;
19     }
20     if (\Yii::$app->user->can($controller.'/'.$action)){
21         self::saveAdminLog();
```

```
            return true;
        }
        if (\Yii::$app->request->isAjax){
            \Yii::$app->response->format = Response::FORMAT_JSON;
            \Yii::$app->response->data = ['code' => 1, 'msg' => '未授权访问'];
            return false;
        }
        throw new UnauthorizedHttpException('未授权访问');
        self::saveAdminLog();
        return true;
    }
```

同时回到当前管理员用户管理模块，分页机制主要是使用 asJson 的形式进行返回数据从而来实现分页技术，另外使用了 Ajax 技术，从而使页面不是全部刷新而是部分刷新，同时对于每一条数据都新增更新、删除、修改、查看、权限等链接从而使得可以对单个管理员用户进行修改，代码如下：

```
public function actionIndex()
{
    if (Yii::$app->request->isAjax){
        $admin = (new Admin())->find();
        $count = $admin->count();
        $page = Yii::$app->request->get('page',1);
        $limit = Yii::$app->request->get('limit',10);
        $username = Yii::$app->request->get('username');
        $email = Yii::$app->request->get('email');
        if ($username){
            $admin->where(['like','username',$username]);
        }
        if ($email){
            $admin->where(['like','email',$email]);
        }
        $data = $admin->offset(($page-1)*$limit)->limit($limit)->asArray()->all();
        foreach ($data as &$item){
            $item['updateUrl'] = Url::to(['update','id'=>$item['id']]);
            $item['destroyUrl'] = Url::to(['destroy','id'=>$item['id']]);
            $item['assignUrl'] = Url::to(['assign','id'=>$item['id']]);
            $item['infoUrl'] = Url::to(['info','id'=>$item['id']]);
            $item['roles'] = Yii::$app->authManager->getRolesByUser($item['id']);
        }
        return $this->asJson([
            'code' => 0,
            'msg' => '请求成功',
```

```
            'count' => $count,
            'data' => $data
        ]);
    }
    return $this->render('index');
}
```

同时其他的操作也是如此，本系统后台模板共实现了类似 10 个 MVC 模型。下面是显示为用户分配角色权限，这里涉及到数据库中多个表的查询与显示，如图：

图 3.16: 为用户分配角色权限

需要指出的是，角色管理模块、权限管理模块、规则管理模块、前端用户管理模块、前端留言管理模块、系统日志模块均是这样的方式进行实现的，由于代码的复用性，因此这里不再进行说明。

3.3.5 代码生成模块

gii 是 yii 中的一个扩展模块，是一个快速开发的好工具；通过 gii 自动生成代码，把一些通用的代码交给程序去生成，很大程度上减少开发者的时间成本。gii 模块可以通过配置 “yii/base/Application::modules” 属性开启它。yii2 是一个快速开发的框架，其中 gii 扩展不得不说是一个很大的助力，通过 gii 自动生成代码，把一些通用的代码交给程序去生成，很大程度上减少开发者的时间成本。为了能够使开发人员进行快速开发，我们将 Gii 模块添加进入了后端菜单中，这里需要将 Gii 进行开启，同时使用 Url::to 跳转到 Gii 的地址，代码如下：

```
<li data-name="gii" class="layui-nav-item">
    <a href="javascript:;" lay-tips=" 代码生成" lay-direction="2">
        <i class="layui-icon layui-icon-template-1"></i>
        <cite> 代码生成 </cite>
    </a>
    <dl class="layui-nav-child">
```

```
            <dd data-name="console">
                <a lay-href="<?php echo \yii\helpers\Url::to(['/gii']); ?>"> 代码生成 </a>
            </dd>
        </dl>
    </li>
```

实现的效果如图：

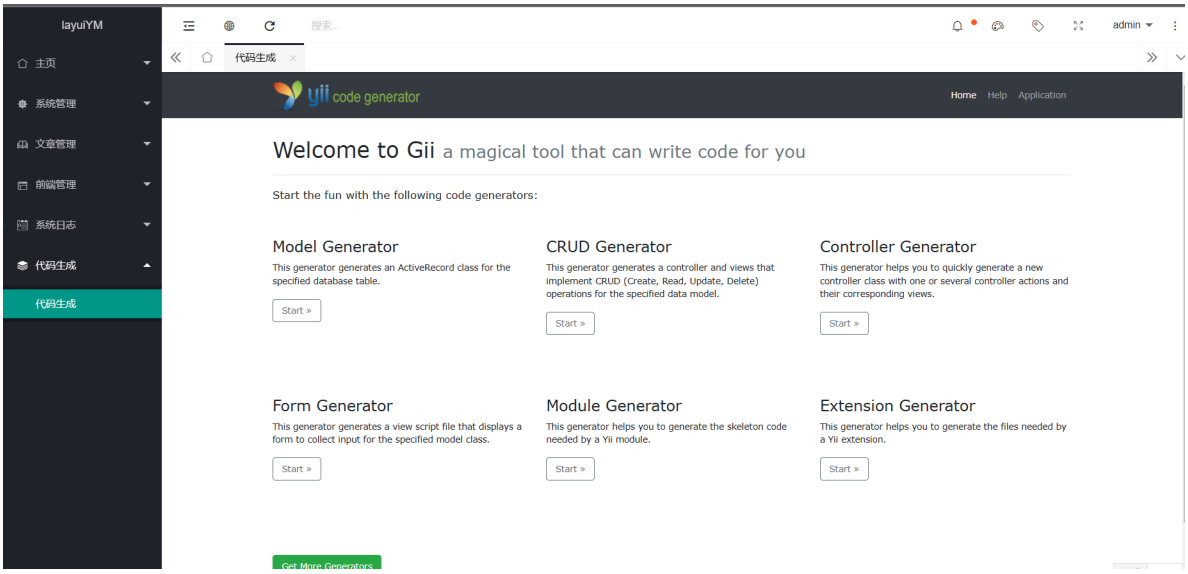

图 3.17: 代码生成模块