

# 评论模块讲解文档

## 一、数据库与实体类设计

### 1.1 核心实体表设计

#### 1.1.1 Comment实体表

属性名	属性含义	是否为空	备注
id	评论ID	否	自增主键
userId	用户ID	否	外键关联User表
scenicId	景点ID	否	外键关联ScenicSpot表
content	评论内容	否	-
rating	评分	是	1-5星评分
likes	点赞数	是	默认为0
createTime	创建时间	是	自动生成
userNickname	用户昵称	是	非持久化字段
userAvatar	用户头像	是	非持久化字段
liked	当前用户是否点赞	是	非持久化字段
scenicName	景点名称	是	非持久化字段

#### 1.1.2 CommentLike实体表

属性名	属性含义	是否为空	备注
id	点赞ID	否	自增主键
userId	用户ID	否	外键关联User表

属性名	属性含义	是否为空	备注
commentId	评论ID	否	外键关联Comment表
createTime	创建时间	是	自动生成

## 1.2 核心属性说明

- 关联关系：评论与用户、景点之间建立多对一关系，通过userId和scenicId进行关联
- 点赞机制：通过CommentLike表记录用户点赞行为，实现点赞/取消点赞功能
- 非持久化字段：userNickname、userAvatar、liked、scenicName等字段为非持久化字段，用于前端展示
- 评分系统：支持1-5星评分，用于表达用户对景点的满意度
- 数据完整性：删除评论时会检查权限，确保只有评论作者或管理员可以删除

# 二、各功能详细讲解

## 2.1 评论列表展示功能

### 2.1.1 功能概述

评论列表展示功能允许用户查看某个景点的所有评论，包括评论内容、评分、点赞数和评论者信息，支持分页查询，并根据用户登录状态显示不同的操作选项。

### 2.1.2 详细讲解功能实现流程

前端通过CommentList.vue组件实现评论列表展示。当用户访问景点详情页面时，组件通过onMounted钩子调用fetchComments()方法，该方法向后端发送GET请求到/comment/page接口，传递scenicId、currentPage和pageSize参数。后端CommentController的getCommentsByPage方法接收请求并调用commentService.getCommentsByPage方法进行分页查询，然后通过批量查询用户信息、景点信息和点赞状态，丰富评论数据后返回给前端。前端接收数据后渲染评论列表，显示用户头像、昵称、评分、评论内容和点赞数，并根据用户权限显示删除按钮。

### 2.1.3 关键代码讲解

1. 代码位置：

【springboot/src/main/java/org/example/springboot/service/CommentService.java】

```
public Page<Comment> getCommentsByPage(Long scenicId, String scenicName, String userName, String content, Integer currentPage, Integer size) {  
    LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>()  
    ();
```

```

// 如果提供了景点ID, 直接使用ID查询
if (scenicId != null) {
    queryWrapper.eq(Comment::getScenicId, scenicId);
}
// 如果提供了景点名称, 先查询景点ID, 再使用ID查询评论
else if (StringUtils.isNotBlank(scenicName)) {
    List<Long> scenicIds = getScenicIdsByName(scenicName);
    if (scenicIds.isEmpty()) {
        // 如果没有找到匹配的景点, 返回空结果
        return new Page<>(currentPage, size);
    }
    queryWrapper.in(Comment::getScenicId, scenicIds);
}

// 其他条件查询...

queryWrapper.orderByDesc(Comment::getCreateTime);
return commentMapper.selectPage(new Page<>(currentPage, size),
    queryWrapper);
}

```

这段代码实现了评论的分页查询功能。它首先根据传入的条件构建查询条件, 如果提供了景点ID, 则直接按ID查询; 如果提供了景点名称, 则先查询对应的景点ID, 再按这些ID查询评论。同样的逻辑也应用于用户名和评论内容的查询。最后按创建时间降序排列, 返回分页结果。

## 2. 代码位置:

**【springboot/src/main/java/org/example/springboot/controller/CommentController.java】**

```

@Operation(summary = "分页查询评论")
@GetMapping("/page")
public Result<?> getCommentsByPage(
    @RequestParam(required = false) Long scenicId,
    @RequestParam(required = false) String scenicName,
    @RequestParam(required = false) String userName,
    @RequestParam(required = false) String content,
    @RequestParam(defaultValue = "1") Integer currentPage,
    @RequestParam(defaultValue = "10") Integer size) {
    Page<Comment> page = commentService.getCommentsByPage(scenicId,
        scenicName, userName, content, currentPage, size);
    // 批量查用户
    List<Long> userIds =
        page.getRecords().stream().map(Comment::getUserId).distinct().toList();
    List<User> users = userService getUsersByIds(userIds);

    // 处理用户信息
    for (Comment c : page.getRecords()) {
        users.stream()

```

```

        .filter(u -> u.getId().equals(c.getUserId()))
        .findFirst()
        .ifPresent(u -> {
            c.setUserNickname(u.getNickname());
            c.setUserAvatar(u.getAvatar());
        });
    }

    // 批量查询景点信息和点赞状态...

    return Result.success(page);
}

```

这段代码是控制器层的评论分页查询实现。它调用服务层方法获取分页数据后，进行了三次数据丰富：1) 批量查询用户信息，填充用户昵称和头像；2) 批量查询景点信息，填充景点名称；3) 批量查询点赞状态，标记当前用户是否已点赞。这种批量查询的方式比循环单次查询更高效。

### 3. 代码位置：【vue3/src/views/frontend/comment/CommentList.vue】

```

const fetchComments = async () => {
    await request.get('/comment/page', {
        scenicId: scenicId.value,
        currentPage: currentPage.value,
        size: pageSize.value
    }, {
        onSuccess: (res) => {
            comments.value = res.records || []
            total.value = res.total || 0
        }
    })
}

```

这段代码是前端获取评论列表的实现。它通过request工具向后端发送GET请求，传递景点ID和分页参数，然后将返回的记录和总数保存到响应式变量中，供模板渲染使用。

## 2.1.4 功能实现细节

- **分页机制**：使用MyBatis-Plus的Page对象实现分页，前端通过el-pagination组件控制页码
- **数据丰富**：通过批量查询用户、景点和点赞信息，减少数据库查询次数
- **条件过滤**：支持按景点ID/名称、用户名/昵称、评论内容进行过滤
- **排序规则**：评论默认按创建时间降序排列，显示最新评论
- **权限控制**：根据用户身份显示不同操作按钮，如删除按钮只对评论作者和管理员可见

## 2.2 发表评论功能

### 2.2.1 功能概述

发表评论功能允许已登录用户对景点进行评价，包括评分和文字评论，评论成功后会立即显示在评论列表中，并与用户账号关联。

### 2.2.2 详细讲解功能实现流程

用户在前端填写评论表单，包括评分和评论内容，点击"发布"按钮后，前端通过 `formRef.value.validate()` 方法验证表单数据的有效性，验证通过后调用 `submitComment()` 方法，该方法通过 `request.post('/comment/add', {...})` 向后端发送POST请求，传递评论内容、评分和景点ID。后端 `CommentController` 的 `addComment` 方法接收请求，从JWT令牌中获取当前用户ID，设置到 `comment` 对象中，然后调用 `commentService.addComment` 方法将评论保存到数据库。保存成功后，前端会清空表单并重新加载评论列表，显示最新评论。

### 2.2.3 关键代码讲解

1. 代码位置：【`vue3/src/views/frontend/comment/CommentList.vue`】

```
const submitComment = () => {
  formRef.value.validate(async (valid) => {
    if (valid) {
      submitLoading.value = true
      try {
        await request.post('/comment/add', {
          ...form,
          scenicId: scenicId.value
        }, {
          successMsg: '评论成功',
          onSuccess: () => {
            form.content = ''
            form.rating = 5
            fetchComments()
          }
        })
      } finally {
        submitLoading.value = false
      }
    }
  })
}
```

这段代码实现了前端提交评论的功能。它首先验证表单数据，验证通过后设置加载状态并发送POST请求，请求成功后清空表单并重新加载评论列表。整个过程中使用了 `try-finally` 结构确保无论请求成功与否都会重置加载状态。

## 2. 代码位置:

【springboot/src/main/java/org/example/springboot/controller/CommentController.java】

```
@Operation(summary = "添加评论")
@PostMapping("/add")
public Result<?> addComment(@RequestBody Comment comment) {
    // 获取当前用户ID
    comment.setUserId(JwtTokenUtils.getCurrentUser().getId());
    commentService.addComment(comment);
    return Result.success("评论成功");
}
```

这段代码是控制器层的添加评论实现。它从JWT令牌中获取当前用户ID，设置到评论对象中，然后调用服务层方法保存评论。这种方式确保了评论与当前登录用户关联，防止伪造用户ID。

## 3. 代码位置:

【springboot/src/main/java/org/example/springboot/service/CommentService.java】

```
public void addComment(Comment comment) {
    if (commentMapper.insert(comment) <= 0) throw new
    ServiceException("评论失败");
}
```

这段代码是服务层的添加评论实现。它直接调用MyBatis-Plus的insert方法将评论保存到数据库，并通过返回值检查操作是否成功，如果失败则抛出异常。

### 2.2.4 功能实现细节

- **表单验证:** 前端使用Element Plus的表单验证功能，确保评论内容不为空且在2-200字之间
- **评分系统:** 使用el-rate组件实现1-5星评分，默认为5星
- **用户认证:** 通过JWT令牌获取当前用户ID，确保评论与用户关联
- **异常处理:** 服务层通过ServiceException处理评论失败的情况，前端通过try-catch捕获异常
- **用户体验:** 提交成功后自动清空表单并刷新评论列表，显示最新评论

## 2.3 评论点赞功能

### 2.3.1 功能概述

评论点赞功能允许已登录用户对评论进行点赞或取消点赞，系统会记录用户的点赞状态，并实时更新评论的点赞数，同一用户对同一评论只能点赞一次。

### 2.3.2 详细讲解功能实现流程

用户点击评论下方的点赞按钮时，前端调用`likeComment(item)`方法，该方法首先检查用户是否已登录，如未登录则提示用户登录，已登录则向后端发送PUT请求到`/comment/like/{id}`接口。后端`CommentController`的`toggleLike`方法接收请求并调用`commentLikeService.toggleLike`方法，该方法首先检查用户是否已点赞该评论，如未点赞则添加点赞记录并增加评论点赞数，如已点赞则删除点赞记录并减少评论点赞数，最后返回当前点赞状态。前端根据返回的点赞状态更新UI显示，包括点赞图标的样式和点赞数。

### 2.3.3 关键代码讲解

#### 1. 代码位置：

【springboot/src/main/java/org/example/springboot/service/CommentLikeService.java】

```
@Transactional
public boolean toggleLike(Long commentId) {
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("用户未登录");
    }

    // 检查评论是否存在
    Comment comment = commentMapper.selectById(commentId);
    if (comment == null) {
        throw new ServiceException("评论不存在");
    }

    // 检查是否已点赞
    LambdaQueryWrapper<CommentLike> queryWrapper = new
    LambdaQueryWrapper<>();
    queryWrapper.eq(CommentLike::getUserId, currentUser.getId())
        .eq(CommentLike::getCommentId, commentId);
    CommentLike like = commentLikeMapper.selectOne(queryWrapper);

    if (like == null) {
        // 未点赞，添加点赞
        like = new CommentLike();
        like.setUserId(currentUser.getId());
        like.setCommentId(commentId);
        commentLikeMapper.insert(like);
    }
}
```

```

// 增加评论点赞数
Integer currentLikes = comment.getLikes();
comment.setLikes(currentLikes == null ? 1 : currentLikes + 1);
commentMapper.updateById(comment);
return true;
} else {
// 已点赞，取消点赞
commentLikeMapper.deleteById(like.getId());

// 减少评论点赞数
Integer currentLikes = comment.getLikes();
comment.setLikes(currentLikes == null || currentLikes <= 0 ? 0
: currentLikes - 1);
commentMapper.updateById(comment);
return false;
}
}
}

```

这段代码实现了评论点赞/取消点赞的核心逻辑。它首先验证用户登录状态和评论存在性，然后查询用户是否已点赞该评论。如果未点赞，则创建点赞记录并增加评论点赞数；如果已点赞，则删除点赞记录并减少评论点赞数。整个过程使用@Transactional注解确保数据一致性。

## 2. 代码位置：【vue3/src/views/frontend/comment/CommentList.vue】

```

const likeComment = async (item) => {
  if (!isLoggedIn.value) {
    ElMessage.warning('请先登录')
    return
  }

  try {
    await request.put(`/comment/like/${item.id}`, null, {
      showDefaultMsg: false,
      onSuccess: (isLiked) => {
        if (isLiked) {
          item.liked = true
          item.likes = (item.likes || 0) + 1
          ElMessage.success('点赞成功')
        } else {
          item.liked = false
          item.likes = Math.max(0, (item.likes || 0) - 1)
          ElMessage.success('取消点赞成功')
        }
      }
    })
  } catch (error) {
    console.error('点赞操作失败:', error)
  }
}

```

这段代码是前端实现点赞功能的核心。它首先检查用户是否已登录，然后发送PUT请求到后端。根据返回的点赞状态，更新评论的liked属性和likes数量，并显示相应的成功提示。注意这里使用了Math.max(0, ...)确保点赞数不会为负数。

### 3. 代码位置:

【springboot/src/main/java/org/example/springboot/controller/CommentController.java】

```
@Operation(summary = "点赞/取消点赞评论")
@PutMapping("/like/{id}")
public Result<?> toggleLike(@PathVariable Long id) {
    boolean isLiked = commentLikeService.toggleLike(id);
    return Result.success(isLiked ? "点赞成功" : "取消点赞成功", isLiked);
}
```

这段代码是控制器层的点赞接口实现。它调用服务层的toggleLike方法，并根据返回的布尔值构建不同的成功消息，同时将点赞状态作为数据返回给前端。

## 2.3.4 功能实现细节

- **点赞状态记录**: 通过CommentLike表记录用户点赞行为，确保同一用户对同一评论只能点赞一次
- **事务管理**: 使用@Transactional注解确保点赞记录和评论点赞数的一致性
- **批量查询点赞状态**: 在评论列表查询时，通过batchCheckLiked方法批量查询当前用户的点赞状态，提高性能
- **前端状态管理**: 点赞/取消点赞后，前端立即更新UI状态，提供即时反馈
- **用户体验**: 未登录用户点击点赞按钮会收到登录提示，已点赞的评论显示不同的按钮样式

## 2.4 评论删除功能

### 2.4.1 功能概述

评论删除功能允许评论作者或管理员删除评论，删除前会进行权限验证，确保只有有权限的用户才能删除评论，删除后会自动刷新评论列表。

### 2.4.2 详细讲解功能实现流程

当用户点击评论下方的删除按钮时，前端首先通过canDelete(item)方法验证用户是否有权删除该评论（评论作者或管理员），然后调用deleteComment(item)方法，该方法通过ElMessageBox.confirm显示确认对话框，用户确认后向后端发送DELETE请求到/comment/delete/{id}接口。后端CommentController的deleteComment方法接收请求，从JWT令牌中获取当前用户ID和角色，然后调用commentService.deleteComment方法，该方法首先验证评论是否存在，然后检查当前用户是否有权限删除（评论作者或管理员），最后执行删除操作。删除成功后，前端重新加载评论列表，更新UI显示。

## 2.4.3 关键代码讲解

### 1. 代码位置:

【springboot/src/main/java/org/example/springboot/service/CommentService.java】

```
public void deleteComment(Long id, Long userId, boolean isAdmin) {
    Comment comment = commentMapper.selectById(id);
    if (comment == null) throw new ServiceException("评论不存在");
    if (!isAdmin && !comment.getUserId().equals(userId)) throw new
ServiceException("无权删除");
    if (commentMapper.deleteById(id) <= 0) throw new
ServiceException("删除失败");
}
```

这段代码实现了评论删除的核心逻辑。它首先查询评论是否存在，然后验证当前用户是否有权删除（是评论作者或管理员），最后执行删除操作。如果任何一步失败，都会抛出带有具体错误信息的ServiceException。

### 2. 代码位置:

【springboot/src/main/java/org/example/springboot/controller/CommentController.java】

```
@Operation(summary = "删除评论")
@DeleteMapping("/delete/{id}")
public Result<> deleteComment(@PathVariable Long id) {
    var user = JwtTokenUtils.getCurrentUser();
    boolean isAdmin = user != null &&
"ADMIN".equals(user.getRoleCode());
    commentService.deleteComment(id, user.getId(), isAdmin);
    return Result.success("删除成功");
}
```

这段代码是控制器层的删除评论实现。它从JWT令牌中获取当前用户信息，判断用户是否为管理员，然后调用服务层方法执行删除操作，并返回成功消息。

### 3. 代码位置: 【vue3/src/views/frontend/comment/CommentList.vue】

```
const canDelete = (item) => {
    return isAdmin.value || item.userId === userId.value
}

const deleteComment = (item) => {
    ElMessageBox.confirm('确定要删除该评论吗?', '提示', {
        confirmButtonText: '确定',
        cancelButtonText: '取消',
        type: 'warning'
    }).then(async () => {
```

```
    await request.delete(`/comment/delete/${item.id}`, {
      successMsg: '删除成功',
      onSuccess: () => fetchComments()
    })
  }).catch(() => {})
}
```

这段代码实现了前端的评论删除功能。`canDelete`方法判断当前用户是否有权限删除评论（是管理员或评论作者），`deleteComment`方法显示确认对话框，用户确认后发送删除请求，并在成功时重新加载评论列表。

#### 2.4.4 功能实现细节

- **权限验证**：前后端都实现了权限验证，确保只有评论作者或管理员可以删除评论
- **用户体验**：删除前显示确认对话框，防止误操作；删除成功后自动刷新评论列表
- **错误处理**：服务层通过`ServiceException`处理各种错误情况，如评论不存在、无权删除等
- **按钮显示控制**：前端通过`canDelete`方法控制删除按钮的显示，只对有权限的用户显示
- **数据一致性**：删除评论后，相关的点赞记录由数据库外键约束自动处理

## 2.5 评论管理功能

### 2.5.1 功能概述

评论管理功能是一个后台管理功能，允许管理员查看、搜索和删除系统中的所有评论，支持按景点名称、用户名/昵称和评论内容进行多条件搜索，提供分页浏览和批量操作功能。

### 2.5.2 详细讲解功能实现流程

管理员访问后台评论管理页面时，前端通过`onMounted`钩子调用`fetchComments()`方法，该方法向后端发送GET请求到`/comment/page`接口，不传递特定的`scenicId`参数，获取所有评论的分页数据。管理员可以通过搜索表单输入景点名称、用户名/昵称和评论内容进行搜索，点击搜索按钮后调用`handleSearch()`方法，该方法重置页码并重新加载评论列表。管理员可以点击评论行的删除按钮删除评论，删除过程与前台相同，但管理员可以删除任何评论，无需权限验证。

### 2.5.3 关键代码讲解

1. 代码位置：【`vue3/src/views/backend/comment/CommentList.vue`】

```
const fetchComments = async () => {
  loading.value = true
  try {
    // 构建查询参数
    const params = {
```

```

        currentPage: currentPage.value,
        size: pageSize.value
    }

    if (searchForm.scenicName) params.scenicName =
searchForm.scenicName
    if (searchForm.userName) params.userName = searchForm.userName
    if (searchForm.content) params.content = searchForm.content

    await request.get('/comment/page', params, {
        showDefaultMsg: false,
        onSuccess: (res) => {
            tableData.value = res.records || []
            total.value = res.total || 0
        }
    })
} catch (error) {
    console.error('获取评论列表失败:', error)
} finally {
    loading.value = false
}
}

```

这段代码实现了后台评论管理页面的数据加载功能。它根据搜索表单的值构建查询参数，发送GET请求获取评论列表，并在成功时更新表格数据和总数。整个过程使用loading状态控制加载动画，并使用try-catch-finally结构处理异常情况。

## 2. 代码位置：【vue3/src/views/backend/comment/CommentList.vue】

```

const handleSearch = () => {
    currentPage.value = 1
    fetchComments()
}

const resetSearch = () => {
    searchForm.scenicName = ''
    searchForm.userName = ''
    searchForm.content = ''
    currentPage.value = 1
    fetchComments()
}

```

这段代码实现了搜索和重置功能。handleSearch方法重置页码并重新加载评论列表，resetSearch方法清空搜索表单的所有字段，重置页码并重新加载评论列表。

## 3. 代码位置：【vue3/src/views/backend/comment/CommentList.vue】

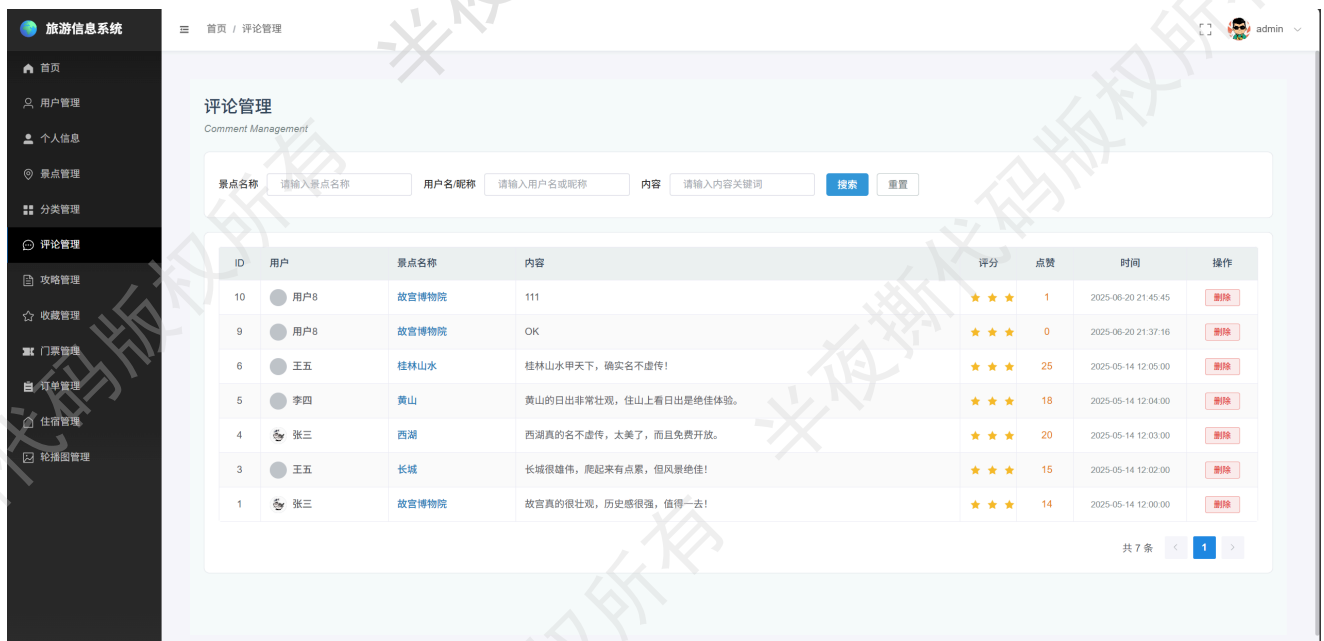
```
const deleteComment = (row) => {
  ElMessageBox.confirm('确定要删除该评论吗?', '提示', {
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
  }).then(async () => {
    await request.delete(`/comment/delete/${row.id}`, {
      successMsg: '删除成功',
      onSuccess: () => fetchComments()
    })
  }).catch(() => {})
}
```

这段代码实现了后台删除评论的功能。它显示确认对话框，用户确认后发送删除请求，并在成功时重新加载评论列表。与前台不同的是，后台管理员可以删除任何评论，无需权限验证。

## 2.5.4 功能实现细节

- **多条件搜索：**支持按景点名称、用户名/昵称和评论内容进行组合搜索
- **分页控制：**使用Element Plus的el-pagination组件实现分页，支持页码跳转
- **表格展示：**使用el-table组件展示评论数据，包括用户信息、景点名称、评论内容、评分、点赞数和操作按钮
- **数据加载优化：**使用loading状态控制加载动画，提升用户体验
- **错误处理：**使用try-catch结构捕获并处理异常情况，确保页面不会因错误而崩溃

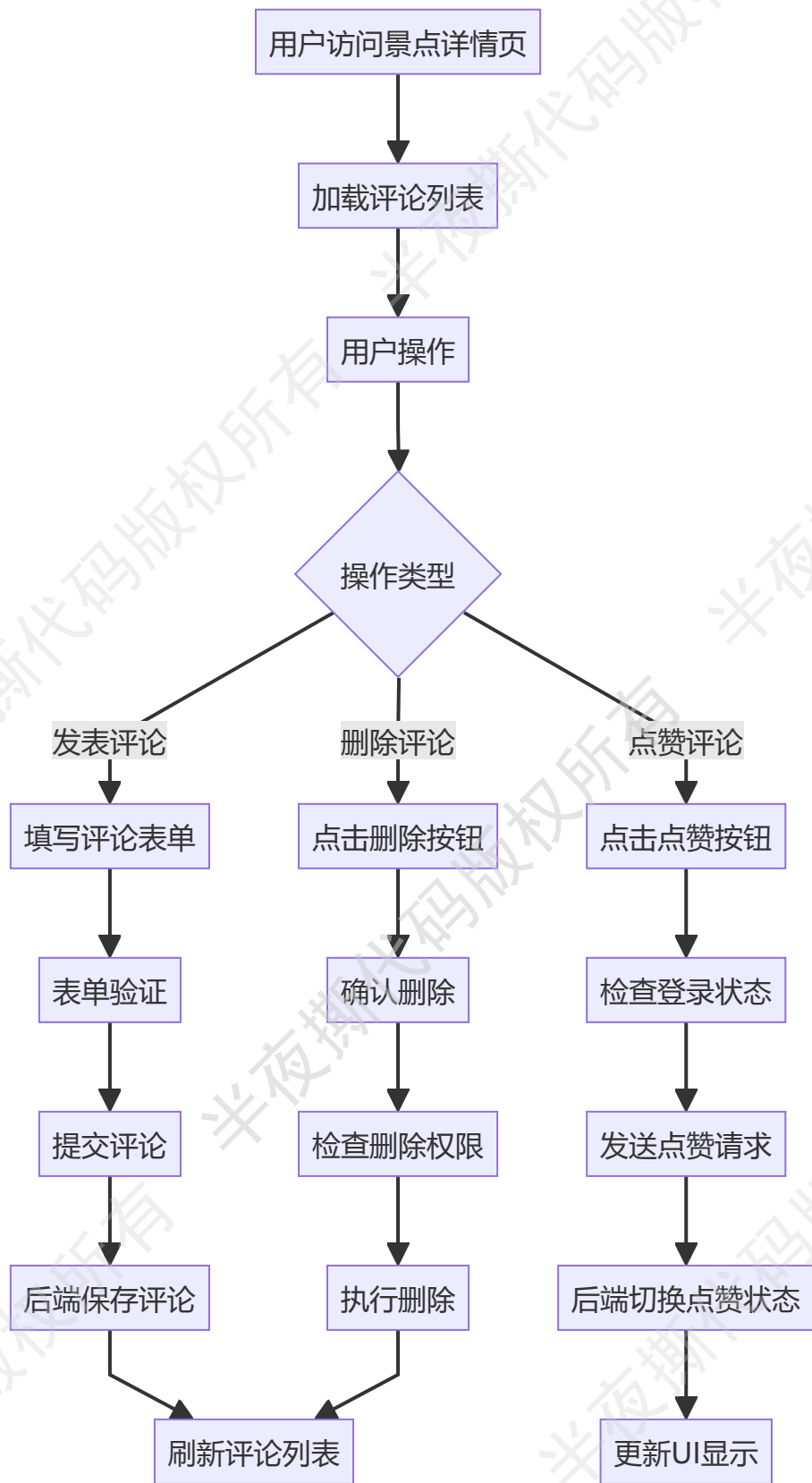
## 2.5.5 【功能截图】



## 三、总结

### 3.1 评论功能实现

评论模块实现了用户对景点的评价功能，通过Comment实体记录评论内容、评分和关联信息。前端CommentList.vue组件通过request.get('/comment/page')获取分页评论数据，后端CommentController的getCommentsByPage方法使用LambdaQueryWrapper构建多条件查询，并通过批量查询用户信息、景点信息和点赞状态丰富评论数据。用户可以通过评论表单发表评论，前端通过formRef.validate()验证数据有效性，然后通过request.post('/comment/add')提交评论，后端通过JwtTokenUtils.getCurrentUser()获取当前用户ID并关联到评论。评论成功后前端会清空表单并刷新评论列表。



### 3.2 点赞功能实现

点赞功能通过CommentLike实体记录用户点赞行为，实现一人一赞的机制。用户点击点赞按钮时，前端通过likeComment()方法发送request.put('/comment/like/\${id}')请求，后端CommentLikeService的toggleLike()方法首先查询用户是否已点赞，未点赞则创建点赞记录并增加评论点赞数，已点赞则删除点赞记录并减少评论点赞数。整个过程使用@Transactional注解确保数据一致性。前端根据返回的点赞状态更新UI显示，包括点赞图标

样式和点赞数。在评论列表加载时，通过`batchCheckLiked()`方法批量查询当前用户的点赞状态，提高性能。

### 3.3 评论管理功能实现

评论管理功能面向系统管理员，通过后台`CommentList.vue`组件实现。管理界面提供了多条件搜索功能，支持按景点名称、用户名/昵称和评论内容过滤评论，并支持分页浏览。管理员可以查看所有评论的详细信息，包括用户信息、景点名称、评论内容、评分和点赞数。删除功能通过`deleteComment()`方法实现，管理员可以删除任何评论，无需权限验证。后端在处理删除请求时，会检查用户角色，管理员可以越权删除任何评论，普通用户只能删除自己的评论。整个管理界面采用`Element Plus`组件库构建，提供了良好的用户体验。

总的来说，评论模块通过前后端分离架构，实现了评论的发布、展示、点赞和管理功能，为用户提供了表达意见和交流的平台，也为管理员提供了内容监管的工具。系统设计考虑了数据安全性、用户体验和性能优化，通过批量查询、事务管理和权限控制等机制，确保了功能的稳定和高效运行。