景点收藏模块讲解文档

一、数据库与实体类设计

1.1 核心实体表设计

1.1.1 ScenicCollection实体表

属性名	属性含义	是否为空	备注
id	收藏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
createTime	创建时间	是	自动生成

1.2 核心属性说明

- **唯一约束**：表中设置了(user_id, scenic_id)的唯一索引，确保同一用户不能重复收藏同一景点
- **外键关联**：userId和scenicId分别与用户表和景点表建立外键关系，保证数据一致性
- **非持久化字段**：实体类中包含scenicInfo非持久化字段，用于前端展示关联的景点详细信息
- **索引优化**：为userId和scenicId单独创建了索引，提高查询效率
- **时间记录**：自动记录收藏创建时间，便于按时间排序展示

二、各功能详细讲解

2.1 景点收藏功能

2.1.1 功能概述

景点收藏功能允许登录用户将感兴趣的景点添加到个人收藏列表中，方便后续查看和管理，是用户个性化体验的重要组成部分。

2.1.2 详细讲解功能实现流程

用户在景点详情页面看到感兴趣的景点时，可以点击页面上方的"收藏"按钮进行收藏操作。前端首先通过checkCollectionStatus方法检查该景点是否已被当前用户收藏，根据结果显示不同的按钮状态（已收藏/未收藏）。当用户点击收藏按钮时，前端调用handleCollection方法，该方法首先检查用户是否已登录，未登录则弹出提示框引导用户登录。如果用户已登录，则根据当前收藏状态执行不同的操作：如果已收藏，则发送DELETE请求到/scenic-collection/{scenicId}接口取消收藏；如果未收藏，则发送POST请求到/scenic-collection/{scenicId}接口添加收藏。后端ScenicCollectionController接收请求，调用scenicCollectionService的相应方法处理收藏操作。添加收藏时，后端会验证用户登录状态、景点是否存在以及是否已收藏，通过验证后将收藏记录保存到数据库。取消收藏时，后端会验证用户登录状态，然后删除对应的收藏记录。操作成功后，前端更新收藏按钮状态，提供即时反馈。

2.1.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/scenic/ScenicDetail.vue】

```
// 收藏/取消收藏操作
const handleCollection = async () => {
  // 检查登录状态
  if (!isLoggedIn.value) {
    ElMessageBox.confirm('收藏功能需要登录，是否前往登录页面？', '提示', {
      confirmButtonText: '去登录',
      cancelButtonText: '取消',
      type: 'info'
    }).then(() => {
      router.push({
        path: '/login',
        query: { redirect: route.fullPath }
      })
    }).catch(() => {})
    return
  }

  const scenicId = scenic.value.id
  if (!scenicId) return

  collectionLoading.value = true
  try {
    if (isCollected.value) {
      // 取消收藏
      try {
        await request.delete(`/scenic-collection/${scenicId}`, {
          successMsg: '取消收藏成功'
        })
        // 手动更新状态
        isCollected.value = false
      } catch (error) {
```

```
            console.error('取消收藏失败:', error)
            ElMessage.error('取消收藏失败，请稍后重试')
          }
        } else {
          // 添加收藏
          try {
            await request.post(`/scenic-collection/${scenicId}`, null, {
              successMsg: '收藏成功'
            })
            // 手动更新状态
            isCollected.value = true
          } catch (error) {
            console.error('添加收藏失败:', error)
            ElMessage.error('添加收藏失败，请稍后重试')
          }
        }
      } catch (error) {
        console.error('操作收藏失败:', error)
        ElMessage.error('操作失败，请稍后重试')
      } finally {
        collectionLoading.value = false
      }
    }
```

代码说明：handleCollection方法负责处理用户的收藏/取消收藏操作。首先检查用户是否已登录，未登录则引导用户前往登录页面。然后根据当前收藏状态发送不同的请求：已收藏状态下发送DELETE请求取消收藏，未收藏状态下发送POST请求添加收藏。请求成功后手动更新前端状态，确保界面及时反映最新的收藏状态。整个过程有完善的错误处理和加载状态管理，提升用户体验。

2. 代码位置：
   【springboot/src/main/java/org/example/springboot/service/ScenicCollectionService.java】

```
/**
 * 添加景点收藏
 */
public void addCollection(Long scenicId) {
    // 获取当前用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("用户未登录");
    }

    // 检查景点是否存在
    ScenicSpot scenicSpot = scenicSpotMapper.selectById(scenicId);
    if (scenicSpot == null) {
        throw new ServiceException("景点不存在");
    }
```

```
        // 检查是否已收藏
        LambdaQueryWrapper<ScenicCollection> queryWrapper = new
    LambdaQueryWrapper<>();
        queryWrapper.eq(ScenicCollection::getUserId, currentUser.getId())
                    .eq(ScenicCollection::getScenicId, scenicId);
        if (scenicCollectionMapper.selectOne(queryWrapper) != null) {
            throw new ServiceException("已收藏该景点");
        }

        // 添加收藏
        ScenicCollection collection = new ScenicCollection();
        collection.setUserId(currentUser.getId());
        collection.setScenicId(scenicId);
        collection.setCreateTime(LocalDateTime.now());

        if (scenicCollectionMapper.insert(collection) <= 0) {
            throw new ServiceException("收藏失败");
        }
    }
```

代码说明：addCollection方法负责处理添加景点收藏的业务逻辑。首先获取当前登录用户，验证用户登录状态；然后检查要收藏的景点是否存在；接着检查用户是否已收藏该景点，避免重复收藏；最后创建收藏记录并保存到数据库。整个过程包含多项业务校验，确保数据的有效性和一致性，并通过异常机制处理各种错误情况。

2.1.4 功能实现细节

- **登录状态检查**：收藏操作需要用户登录，未登录用户会被引导到登录页面
- **重复收藏检查**：系统会检查用户是否已收藏该景点，避免重复收藏
- **即时反馈**：操作成功或失败都有明确的提示信息，提升用户体验
- **按钮状态管理**：根据收藏状态显示不同的按钮样式和文字，直观反映当前状态
- **加载状态**：操作过程中显示加载状态，避免用户重复点击
- **异常处理**：完善的异常处理机制，确保系统稳定性

2.2 景点收藏列表查看功能

2.2.1 功能概述

景点收藏列表查看功能允许用户查看自己收藏的所有景点，支持分页浏览和取消收藏操作，帮助用户管理个人收藏内容。

2.2.2 详细讲解功能实现流程

用户点击个人中心的"我的收藏"菜单项，进入收藏管理页面。页面加载时，前端通过onMounted钩子调用fetchScenicCollections方法，该方法向后端发送GET请求到/scenic-collection/user接口，传递用户ID、当前页码和每页数量参数。后端ScenicCollectionController的getUserCollections方法接收请求，调用scenicCollectionService.getUserCollections方法查询用户的收藏记录。该方法首先验证用户身份，然后查询数据库获取收藏记录，并通过fillScenicInfo方法填充关联的景点信息。查询结果以分页形式返回给前端，前端接收数据后，通过el-card组件将每个收藏的景点以卡片形式展示，包括景点图片、名称、位置、价格等信息。用户可以点击景点卡片查看详情，也可以点击"取消收藏"按钮取消收藏。点击"取消收藏"按钮时，前端显示确认对话框，用户确认后调用handleCancelCollection方法，向后端发送DELETE请求到/scenic-collection/{scenicId}接口。后端处理取消收藏逻辑，操作成功后前端刷新收藏列表。

2.2.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/collection/MyCollection.vue】

```
// 获取用户收藏的景点
const fetchScenicCollections = async () => {
  loading.value = true
  try {
    await request.get('/scenic-collection/user', {
      currentPage: currentPage.value,
      size: pageSize.value,
      userId: userStore.userInfo.id
    }, {
      onSuccess: (data) => {
        scenicCollections.value = data.records || []
        total.value = data.total || 0
      }
    })
  } catch (error) {
    console.error('获取收藏景点失败:', error)
  } finally {
    loading.value = false
  }
}

// 取消收藏
const handleCancelCollection = (scenicId) => {
  ElMessageBox.confirm('确认取消收藏该景点?', '提示', {
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
  }).then(async () => {
    try {
      await request.delete(`/scenic-collection/${scenicId}`, {}, {
        successMsg: '取消收藏成功',
```

```
          onSuccess: () => {
            // 刷新列表
            fetchScenicCollections()
          }
        })
      } catch (error) {
        console.error('取消收藏失败:', error)
      }
    }).catch(() => {
      // 用户取消操作
    })
}
```

代码说明：fetchScenicCollections方法负责获取用户收藏的景点列表，通过request.get发送请求到后端接口，请求成功后将返回的数据保存到scenicCollections响应式变量中。handleCancelCollection方法处理取消收藏操作，首先显示确认对话框，用户确认后发送DELETE请求到后端接口，操作成功后刷新列表。这两个方法共同实现了收藏列表的查看和管理功能。

2. 代码位置：
【springboot/src/main/java/org/example/springboot/service/ScenicCollectionService.java】

```
/**
 * 查询用户收藏的景点列表（分页）
 */
public Page<ScenicCollection> getUserCollections(Long userId, Integer
currentPage, Integer size) {
    if (userId == null) {
        // 获取当前用户
        User currentUser = JwtTokenUtils.getCurrentUser();
        if (currentUser == null) {
            throw new ServiceException("用户未登录");
        }
        userId = currentUser.getId();
    }

    // 查询收藏记录
    LambdaQueryWrapper<ScenicCollection> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper.eq(ScenicCollection::getUserId, userId)
                .orderByDesc(ScenicCollection::getCreateTime);

    Page<ScenicCollection> page = scenicCollectionMapper.selectPage(new
Page<>(currentPage, size), queryWrapper);

    // 填充景点信息
    fillScenicInfo(page.getRecords());
```

```
        return page;
    }

    /**
     * 填充景点信息
     */
    private void fillScenicInfo(List<ScenicCollection> collections) {
        if (collections == null || collections.isEmpty()) {
            return;
        }

        // 提取景点ID
        List<Long> scenicIds = collections.stream()
                .map(ScenicCollection::getScenicId)
                .collect(Collectors.toList());

        // 批量查询景点信息
        List<ScenicSpot> scenicSpots =
    scenicSpotService.getScenicSpotsByIds(scenicIds);

        // 转换为Map便于查找
        Map<Long, ScenicSpot> scenicSpotMap = scenicSpots.stream()
                .collect(Collectors.toMap(ScenicSpot::getId, spot ->
    spot));

        // 填充景点信息
        collections.forEach(collection -> {
            if (scenicSpotMap.containsKey(collection.getScenicId())) {

     collection.setScenicInfo(scenicSpotMap.get(collection.getScenicId()));
            }
        });
    }
```

代码说明：getUserCollections方法负责查询用户收藏的景点列表，支持分页查询。如果未提供userId，则使用当前登录用户的ID。查询结果按收藏时间降序排序，确保最新收藏的景点显示在前面。查询完成后，调用fillScenicInfo方法填充关联的景点信息，该方法通过批量查询优化了性能，避免了N+1查询问题。这种设计既满足了功能需求，又保证了查询效率。

2.2.4 功能实现细节

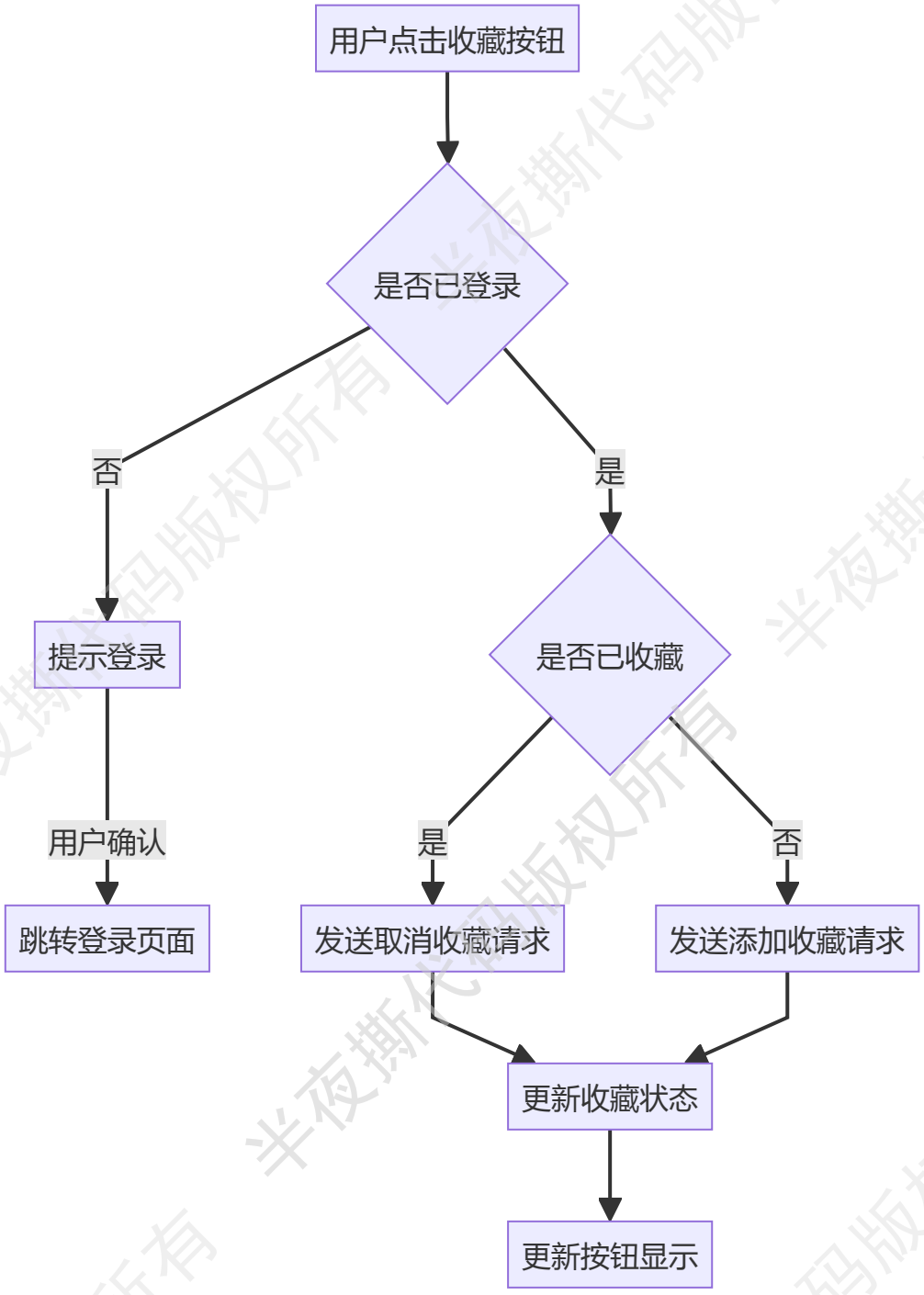
- **分页查询**：支持分页查询收藏列表，提高大数据量下的查询效率
- **时间排序**：按收藏时间降序排序，最新收藏的景点显示在前面
- **数据关联**：自动填充关联的景点信息，避免前端多次请求
- **批量查询优化**：使用批量查询景点信息，避免N+1查询问题
- **卡片式布局**：采用卡片式布局展示收藏的景点，视觉效果好
- **响应式设计**：适应不同屏幕尺寸，提供良好的移动端体验

- **空数据处理**：当没有收藏景点时，显示友好的空状态提示

三、总结

3.1 景点收藏功能实现

景点收藏功能通过前端ScenicDetail.vue组件中的handleCollection方法实现，该方法根据当前收藏状态发送不同的请求：未收藏时发送POST请求添加收藏，已收藏时发送DELETE请求取消收藏。后端ScenicCollectionService.addCollection方法处理添加收藏逻辑，包括验证用户登录状态、检查景点是否存在、避免重复收藏等。整个功能设计注重用户体验，提供了即时反馈和友好的交互，同时通过完善的异常处理确保系统稳定性。



3.2 景点收藏列表查看功能实现

景点收藏列表查看功能通过MyCollection.vue组件实现，该组件在加载时调用fetchScenicCollections方法获取用户收藏的景点列表。后端ScenicCollectionService.getUserCollections方法查询用户的收藏记录，并通过fillScenicInfo方法填充关联的景点信息。前端以卡片形式展示收藏的景点，支持查看详情和取消收藏操作。取消收藏通过handleCancelCollection方法实现，该方法发送DELETE请求到后端接口，操作成功后刷新列表。整个功能设计注重数据关联和用户体验，通过批量查询优化了性能，避免了N+1查询问题。