失物模块讲解文档

一、数据库与实体类设计

1.1 核心实体表设计

1.1.1 lost_item表

失物信息表用于存储用户发布的失物信息。

属性名	属性含义	是否为空	备注
id	失物ID	否	主键，自增
title	标题	否	失物标题
description	描述	是	失物详细描述
category_id	分类ID	是	关联物品分类表
lost_place	丢失地点	是	物品丢失的具体位置
lost_time	丢失时间	是	物品丢失的具体时间
contact_name	联系人姓名	是	联系人姓名
contact_phone	联系电话	是	联系电话
images	图片	是	多张图片路径，用逗号分隔
user_id	发布用户ID	否	关联用户表
status	状态	否	0待认领,1已认领,2已关闭
create_time	创建时间	否	默认为当前时间
update_time	更新时间	否	默认为当前时间，自动更新

1.1.2 claim_application表

认领申请表用于记录用户对失物或招领信息的认领申请。

属性名	属性含义	是否为空	备注
id	申请ID	否	主键，自增
item_id	物品ID	否	失物或招领物品ID
item_type	物品类型	否	0招领信息,1失物信息
user_id	申请人ID	否	申请人用户ID
description	申请说明	是	认领申请的描述
status	状态	否	0待审核,1已通过,2已拒绝
create_time	创建时间	否	默认为当前时间
update_time	更新时间	否	默认为当前时间，自动更新
audit_user_id	审核人ID	是	审核人的用户ID
audit_time	审核时间	是	审核的时间
audit_remark	审核备注	是	审核意见或备注

1.2 核心属性说明

LostItem实体类

LostItem实体类对应失物信息表，在SpringBoot后端中定义。除了数据库表中的字段外，还包含以下非数据库字段：

- categoryName: 分类名称，用于前端显示，通过categoryId关联获取
- username: 用户名，用于前端显示，通过userId关联获取

```
@Data
@TableName("lost_item")
@Schema(description = "失物信息实体类")
public class LostItem {
    @TableId(type = IdType.AUTO)
    @Schema(description = "失物ID")
    private Long id;

    @Schema(description = "标题")
```

```
    private String title;

    // 其他字段省略...

    // 额外字段，用于前端展示
    @TableField(exist = false)
    @Schema(description = "分类名称")
    private String categoryName;

    @TableField(exist = false)
    @Schema(description = "用户名")
    private String username;
}
```

ClaimApplication实体类

ClaimApplication实体类对应认领申请表，包含多个非数据库字段用于关联信息展示。

二、各功能详细讲解

2.1 失物信息发布功能

2.1.1 功能概述

失物信息发布功能允许用户填写失物的详细信息（包括标题、分类、丢失地点和时间、物品描述、图片和联系方式）并提交到系统，使其他用户可以查看并认领。发布失物信息后，系统会将其初始状态设置为"待认领"。

2.1.2 详细讲解功能实现流程

失物发布流程：用户在前端填写表单数据后，点击"发布"按钮，前端通过表单验证后使用request工具向后端`/lost-item`接口发送POST请求，后端Controller层的save方法接收数据并调用LostItemService的addLostItem方法处理。在addLostItem方法中，会验证用户登录状态、验证表单数据完整性、设置默认状态为"待认领"，最后调用lostItemMapper的insert方法将数据写入数据库。发布成功后，前端弹窗提示用户并询问是否返回列表页面。

2.1.3 关键代码讲解

1. 前端发布页面表单提交代码 【vue3/src/views/frontend/lost/publish.vue】

```
const submitForm = async () => {
  if (!userStore.userInfo || !userStore.userInfo.id) {
    ElMessage.warning('请先登录后再发布失物信息')
    router.push('/login')
    return
  }

  await formRef.value.validate(async (valid) => {
```

```
      if (!valid) {
        return
      }

      // 处理图片路径
      if (fileList.value.length > 0) {
        formData.images = fileList.value.map(file => file.raw).join(',')
      } else {
        formData.images = ''
      }

      submitting.value = true

      try {
        // 提交失物信息
        await request.post('/lost-item', {
          ...formData,
          userId: userStore.userInfo.id,
          status: 0 // 待认领状态
        }, {
          successMsg: '发布成功',
          onSuccess: () => {
            ElMessageBox.confirm(
              '失物信息发布成功，是否返回列表？',
              '提示',
              {
                confirmButtonText: '返回列表',
                cancelButtonText: '继续发布',
                type: 'success'
              }
            ).then(() => {
              router.push('/lost')
            }).catch(() => {
              resetForm()
            })
          }
        })
      } catch (error) {
        console.error('发布失物信息失败:', error)
      } finally {
        submitting.value = false
      }
    })
  }
```

这段代码首先检查用户是否登录，然后验证表单数据，处理图片路径，最后向后端发送请求。发布成功后会弹出确认框询问用户是否返回列表。

2. 后端处理失物信息发布
【springboot/src/main/java/org/example/springboot/service/LostItemService.java】

```java
@Transactional(rollbackFor = Exception.class)
public void addLostItem(LostItem lostItem) {
    // 获取当前登录用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("未登录或登录已过期");
    }

    // 检查用户状态
    userService.checkUserCanOperate(currentUser);

    // 完整的数据验证
    validateLostItemData(lostItem);

    // 设置用户ID和默认状态
    lostItem.setUserId(currentUser.getId());
    lostItem.setStatus(0); // 待认领状态

    // 验证分类是否存在
    if (lostItem.getCategoryId() != null) {
        itemCategoryService.getCategoryById(lostItem.getCategoryId());
    }

    if (lostItemMapper.insert(lostItem) <= 0) {
        throw new ServiceException("添加失物信息失败");
    }
}
```

这段代码首先验证用户登录状态和权限，然后对提交数据进行验证，设置默认的用户ID和状态，最后将数据插入数据库。整个操作使用事务保证数据一致性，出现异常时会回滚所有操作。

2.1.4 功能实现细节

失物发布功能包含以下细节处理：

- **表单验证**：前端使用Element Plus的表单验证功能对用户输入进行校验，包括必填项验证、长度限制和格式验证。
- **图片上传**：支持多图片上传，限制文件大小和类型，将图片上传到服务器后返回路径存储。
- **用户身份验证**：后端通过JWT令牌获取当前登录用户身份，防止未登录用户或非法用户提交数据。
- **数据完整性校验**：后端对必填字段进行二次验证，确保数据完整有效。

- **事务处理**：使用Spring的@Transactional注解确保数据操作的原子性。

2.1.5 【功能截图占位】

2.2 失物信息查询与展示功能

2.2.1 功能概述

失物查询与展示功能包括列表展示和详情查看两部分。列表页支持按关键词、分类和状态进行筛选查询，以卡片形式展示失物信息。详情页展示失物的所有信息，包括图片、联系方式和认领选项。

2.2.2 详细讲解功能实现流程

列表查询流程：用户打开失物列表页面，前端组件初始化时调用fetchLostItems方法向后端`/lost-item/page`接口发送GET请求，请求参数包括搜索条件、分页参数，后端Controller层的page方法接收请求并调用LostItemService的queryByPage方法处理。queryByPage方法构建查询条件，调用Mapper层查询数据库，并填充关联信息（如分类名称、用户名）。前端接收到数据后，以卡片列表形式展示。

详情查看流程：用户点击卡片查看详情，前端路由跳转到详情页面，组件初始化时调用fetchLostItemDetail方法向后端`/lost-item/{id}`接口发送GET请求，后端Controller层的getById方法接收请求并调用LostItemService的getItemById方法获取详细信息。前端接收到数据后，展示失物的完整信息。

2.2.3 关键代码讲解

1. 前端列表查询代码 【vue3/src/views/frontend/lost/index.vue】

```
// 获取失物列表
const fetchLostItems = async () => {
  loading.value = true
  try {
    await request.get('/lost-item/page', {
      title: searchForm.title,
      categoryId: searchForm.categoryId,
      status: 0, // 只显示待认领的
      currentPage: currentPage.value,
      size: pageSize.value
    }, {
      showDefaultMsg: false,
      onSuccess: (res) => {
        lostItems.value = res.records || []
        total.value = res.total || 0
      }
    })
  } catch (error) {
    console.error('获取失物信息列表失败:', error)
  } finally {
```

```
        loading.value = false
    }
}
```

这段代码发送GET请求获取失物列表，包含搜索条件和分页参数，并在成功回调中更新数据状态。

2. 后端分页查询处理
【springboot/src/main/java/org/example/springboot/service/LostItemService.java】

```
public Page<LostItem> queryByPage(Integer currentPage, Integer size,
String title,
                                  Long categoryId, Integer status, Long
userId) {
    LambdaQueryWrapper<LostItem> queryWrapper = new
LambdaQueryWrapper<>();

    // 添加查询条件
    if (StringUtils.isNotBlank(title)) {
        queryWrapper.like(LostItem::getTitle, title);
    }

    if (categoryId != null) {
        queryWrapper.eq(LostItem::getCategoryId, categoryId);
    }

    if (status != null) {
        queryWrapper.eq(LostItem::getStatus, status);
    }

    if (userId != null) {
        queryWrapper.eq(LostItem::getUserId, userId);
    }

    // 按创建时间降序排序
    queryWrapper.orderByDesc(LostItem::getCreateTime);

    // 执行查询
    Page<LostItem> page = lostItemMapper.selectPage(new Page<>
(currentPage, size), queryWrapper);

    // 填充关联信息
    for (LostItem item : page.getRecords()) {
        fillInfo(item);
    }

    return page;
}
```

这段代码构建查询条件，执行分页查询，并为每条记录填充关联信息（如分类名称、用户名）。

3. 关联信息填充方法
【springboot/src/main/java/org/example/springboot/service/LostItemService.java】

```
private void fillInfo(LostItem lostItem) {
    // 填充分类名称
    if (lostItem.getCategoryId() != null) {
        ItemCategory category =
itemCategoryMapper.selectById(lostItem.getCategoryId());
        if (category != null) {
            lostItem.setCategoryName(category.getName());
        }
    }

    // 填充用户名
    if (lostItem.getUserId() != null) {
        User user = userMapper.selectById(lostItem.getUserId());
        if (user != null) {
            lostItem.setUsername(user.getUsername());
        }
    }
}
```

这个方法通过ID查询关联表，获取分类名称和用户名等信息，填充到失物对象中，便于前端展示。

2.2.4 功能实现细节

失物查询与展示功能包含以下细节处理：

- **多条件查询**：支持按标题关键词、分类和状态进行组合查询。
- **分页处理**：前后端结合实现分页功能，减轻数据库压力。
- **数据缓存**：前端使用Vue的响应式系统缓存查询结果，减少不必要的请求。
- **关联数据处理**：后端查询时自动填充关联表的信息，前端无需多次请求。
- **图片处理**：自动处理图片路径，支持多图片展示。
- **UI交互优化**：使用Element Plus的卡片组件展示列表，提供良好的视觉体验。

2.2.5 【功能截图占位】

2.3 失物信息编辑功能

2.3.1 功能概述

失物编辑功能允许用户（发布者或管理员）修改已发布的失物信息，包括标题、分类、丢失地点和时间、物品描述、图片和联系方式等。同时也可以更新失物的状态（待认领、已认领、已关闭）。

2.3.2 详细讲解功能实现流程

编辑流程：用户点击编辑按钮，前端路由跳转到编辑页面，组件初始化时调用后端`/lost-item/{id}`接口获取当前失物信息填充表单。用户修改信息后提交表单，前端验证数据有效性，然后调用request工具向后端`/lost-item/{id}`接口发送PUT请求。后端Controller层的update方法接收请求并调用LostItemService的updateLostItem方法更新数据。后端会验证用户权限，确保只有发布者或管理员可以编辑信息。更新成功后，前端提示用户并返回上一页面。

2.3.3 关键代码讲解

1. 前端编辑表单提交代码 【vue3/src/views/frontend/lost/edit.vue】

```
const submitForm = async () => {
  await formRef.value.validate(async (valid) => {
    if (!valid) return

    submitting.value = true

    try {
      // 处理图片数据
      let imagesList = []
      if (existingImages.value.length > 0) {
        imagesList = imagesList.concat(existingImages.value)
      }
      if (newImageList.value.length > 0) {
        imagesList = imagesList.concat(newImageList.value)
      }
      formData.value.images = imagesList.join(',')

      // 发送更新请求
      await request.put(`/lost-item/${formData.value.id}`,
formData.value, {
        successMsg: '更新成功',
        onSuccess: () => {
          ElMessageBox.confirm(
            '失物信息更新成功，是否返回详情页？',
            '提示',
            {
```

```
            confirmButtonText: '返回详情',
            cancelButtonText: '继续编辑',
            type: 'success'
          }
        ).then(() => {
          router.push(`/lost/detail/${formData.value.id}`)
        }).catch(() => {})
      }
    })
  } catch (error) {
    console.error('更新失物信息失败:', error)
  } finally {
    submitting.value = false
  }
 })
}
```

这段代码首先验证表单数据，然后处理图片路径（合并已有图片和新上传图片），最后向后端发送更新请求。更新成功后会弹出确认框询问用户是否返回详情页。

2. 后端处理失物信息更新

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

```
@Transactional(rollbackFor = Exception.class)
public void updateLostItem(LostItem lostItem) {
    // 检查物品是否存在
    LostItem existingItem =
lostItemMapper.selectById(lostItem.getId());
    if (existingItem == null) {
        throw new ServiceException("失物信息不存在");
    }

    // 获取当前登录用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("未登录或登录已过期");
    }

    // 权限检查：只有管理员或物品发布者可以修改
    boolean isAdmin = "ADMIN".equals(currentUser.getRoleCode());
    boolean isOwner =
existingItem.getUserId().equals(currentUser.getId());

    if (!isAdmin && !isOwner) {
        throw new ServiceException("无权限修改此失物信息");
    }

    // 数据验证
```

```
        validateLostItemData(lostItem);

        // 保留原始创建者和创建时间
        lostItem.setUserId(existingItem.getUserId());
        lostItem.setCreateTime(existingItem.getCreateTime());

        // 更新数据
        if (lostItemMapper.updateById(lostItem) <= 0) {
            throw new ServiceException("更新失物信息失败");
        }
    }
```

这段代码首先检查物品是否存在，然后验证用户权限（只有管理员或发布者可以修改），接着验证表单数据，最后更新数据库记录。整个操作使用事务保证数据一致性。

2.3.4 功能实现细节

失物编辑功能包含以下细节处理：

- **权限控制**：后端验证用户身份，确保只有发布者或管理员可以编辑信息。
- **表单预填充**：编辑页面加载时自动获取并填充现有失物信息。
- **图片管理**：支持维护现有图片和上传新图片，可以删除不需要的图片。
- **表单验证**：前后端双重验证确保数据有效性。
- **状态管理**：可以更新失物状态（待认领、已认领、已关闭）。
- **原始信息保留**：更新时保留原始的创建者ID和创建时间。

2.3.5 【功能截图占位】

2.4 失物状态管理功能

2.4.1 功能概述

失物状态管理功能允许用户（发布者或管理员）更改失物的状态，包括"待认领"、"已认领"和"已关闭"。状态变更会触发通知机制，通知相关用户。同时，系统也会自动处理长时间未认领的失物，将其标记为过期状态。

2.4.2 详细讲解功能实现流程

状态更新流程：用户在失物详情页或后台管理页面点击状态更新按钮，前端调用request工具向后端`/lost-item/{id}/status`接口发送PUT请求，传递新的状态值。后端Controller层接收请求并调用ItemStatusService的updateLostItemStatus方法处理。该方法验证用户权限、检查状态转换是否合法，然后更新数据库并发送状态变更通知。

自动过期处理流程：系统定时任务（ItemStatusTask）定期调用ItemStatusService的processExpiredItems方法，该方法查询超过指定时间（如30天）未被认领的失物信息，将其状态更新为过期，并发送通知给发布者。

2.4.3 关键代码讲解

1. 前端状态更新代码 【vue3/src/views/backend/lost/index.vue】

```
// 更改失物信息状态
const handleStatusChange = async (row, status) => {
  try {
    const statusMap = {
      0: '待认领',
      1: '已认领',
      2: '已关闭'
    }

    await request.put(`/lost-item/${row.id}/status`, { status }, {
      successMsg: `已将状态更新为"${statusMap[status]}"`,
      onSuccess: () => {
        fetchLostItems()
      }
    })
  } catch (error) {
    console.error('更新状态失败:', error)
  }
}
```

这段代码向后端发送请求更新失物状态，成功后刷新列表数据。

2. 后端状态更新处理
【springboot/src/main/java/org/example/springboot/service/ItemStatusService.java】

```
@Transactional(rollbackFor = Exception.class)
public void updateLostItemStatus(Long itemId, Integer newStatus, String
reason) {
    LostItem lostItem = lostItemMapper.selectById(itemId);
    if (lostItem == null) {
        throw new ServiceException("失物信息不存在");
    }

    // 检查权限
    checkUpdatePermission(lostItem.getUserId());

    // 验证状态转换
    ItemStatus currentStatus =
ItemStatus.fromValue(lostItem.getStatus());
    ItemStatus targetStatus = ItemStatus.fromValue(newStatus);

    if (!currentStatus.canTransitionTo(targetStatus)) {
        throw new
ServiceException(currentStatus.getTransitionAdvice(targetStatus));
```

```
    }

    // 更新状态
    Integer oldStatus = lostItem.getStatus();
    lostItem.setStatus(newStatus);

    lostItemMapper.updateById(lostItem);

    // 发送状态变更通知
    sendStatusChangeNotification(lostItem.getUserId(),
lostItem.getTitle(),
                                 currentStatus.getDescription(),
targetStatus.getDescription(), itemId);

    log.info("失物状态更新成功: itemId={}, oldStatus={}, newStatus={},
reason={}",
            itemId, oldStatus, newStatus, reason);
}
```

这段代码首先检查物品存在性和用户权限，然后验证状态转换的合法性，最后更新状态并发送通知。整个操作使用事务保证数据一致性。

3. 失物过期处理
【springboot/src/main/java/org/example/springboot/service/ItemStatusService.java】

```
@Transactional(rollbackFor = Exception.class)
public void processExpiredItems(int expireDays) {
    LocalDateTime expireTime =
LocalDateTime.now().minusDays(expireDays);

    // 处理过期失物
    LambdaQueryWrapper<LostItem> lostQueryWrapper = new
LambdaQueryWrapper<>();
    lostQueryWrapper.eq(LostItem::getStatus,
ItemStatus.PENDING.getValue())
                    .lt(LostItem::getCreateTime, expireTime);

    List<LostItem> expiredLostItems =
lostItemMapper.selectList(lostQueryWrapper);
    for (LostItem item : expiredLostItems) {
        item.setStatus(ItemStatus.EXPIRED.getValue());
        lostItemMapper.updateById(item);

        // 发送过期通知
        sendStatusChangeNotification(item.getUserId(), item.getTitle(),
                                     ItemStatus.PENDING.getDescription(),
                                     ItemStatus.EXPIRED.getDescription(),
item.getId());
```

```
        }

        // 处理过期招领（代码略）

        log.info("处理过期物品完成：失物{}个，招领{}个",
    expiredLostItems.size(), expiredFoundItems.size());
    }
```

这段代码查询超过指定天数（expireDays）未被认领的失物信息，将其状态更新为过期，并发送通知给发布者。

2.4.4 功能实现细节

失物状态管理功能包含以下细节处理：

- **状态转换规则**：定义了状态转换的合法路径，例如"待认领"可以转换为"已认领"或"已关闭"，但"已认领"不能转回"待认领"。
- **权限控制**：只有物品发布者或管理员可以更改状态。
- **通知机制**：状态变更时自动发送通知给相关用户。
- **自动过期处理**：定时任务自动处理长时间未认领的物品。
- **事务管理**：使用Spring的@Transactional注解确保数据操作的原子性。
- **日志记录**：详细记录状态变更操作，便于追踪和审计。

2.4.5 与其它模块交互

失物状态管理功能与认领申请模块和通知模块有密切交互：

- **认领申请**：当认领申请被批准时，会自动更新失物状态为"已认领"。
- **通知模块**：状态变更时会通过通知模块发送消息给相关用户。

2.5 失物认领申请功能

2.5.1 功能概述

失物认领申请功能允许用户对他人发布的失物信息提交认领申请。申请需包含认领说明，系统会通知失物发布者进行审核。发布者可以通过或拒绝申请，审核结果会通知申请者。

2.5.2 详细讲解功能实现流程

认领申请流程：用户在失物详情页点击"申请认领"按钮，填写认领说明并提交。前端调用request工具向后端`/claim-application`接口发送POST请求。后端Controller层接收请求并调用ClaimApplicationService的add方法处理。该方法验证用户身份、检查物品状态、验证申请数据，然后保存申请记录并发送通知给失物发布者。

申请审核流程：失物发布者在申请列表页面查看申请详情，点击"通过"或"拒绝"按钮进行审核。前端调用request工具向后端/claim-application/{id}/audit接口发送PUT请求，包含审核结果和备注。后端Controller层接收请求并调用ClaimApplicationService的audit方法处理。该方法验证用户权限、更新申请状态，如果通过则同时更新失物状态为"已认领"并拒绝其他申请，最后发送审核结果通知给申请者。

2.5.3 关键代码讲解

1. 认领申请提交代码
   【springboot/src/main/java/org/example/springboot/service/ClaimApplicationService.java】

```
@Transactional(rollbackFor = Exception.class)
public void add(ClaimApplication claimApplication) {
    // 获取当前用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("用户未登录");
    }

    // 设置申请人ID
    claimApplication.setUserId(currentUser.getId());

    // 检查失物信息
    if (claimApplication.getItemType() == 1) {
        // 失物信息
        LostItem lostItem =
lostItemMapper.selectById(claimApplication.getItemId());
        if (lostItem == null) {
            throw new ServiceException("失物信息不存在");
        }

        // 检查物品状态是否为待认领
        if (lostItem.getStatus() != 0) {
            throw new ServiceException("该物品不可认领，可能已被认领或已关
闭");
        }

        // 检查是否是自己发布的物品
        if (lostItem.getUserId().equals(currentUser.getId())) {
            throw new ServiceException("不能认领自己发布的物品");
        }
    }

    // 检查是否已经申请过该物品
    LambdaQueryWrapper<ClaimApplication> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper.eq(ClaimApplication::getItemId,
claimApplication.getItemId())
```

```
                .eq(ClaimApplication::getItemType,
claimApplication.getItemType())
                .eq(ClaimApplication::getUserId, currentUser.getId())
                .in(ClaimApplication::getStatus, 0, 1);  // 待审核或已通过
的申请

    if (claimApplicationMapper.selectCount(queryWrapper) > 0) {
        throw new ServiceException("您已经申请过该物品，请勿重复申请");
    }

    // 设置初始状态为待审核
    claimApplication.setStatus(0);
    claimApplication.setCreateTime(LocalDateTime.now());
    claimApplication.setUpdateTime(LocalDateTime.now());

    // 保存申请记录
    if (claimApplicationMapper.insert(claimApplication) <= 0) {
        throw new ServiceException("申请提交失败");
    }

    // 获取申请人信息
    User applicant = userMapper.selectById(currentUser.getId());

    // 发送申请通知
    sendApplicationNotification(claimApplication, applicant);
}
```

这段代码处理认领申请的提交，包括验证用户身份、检查物品状态、验证申请有效性，最后保存申请记录并发送通知。

2. 申请审核处理代码
【springboot/src/main/java/org/example/springboot/service/ClaimApplicationService.java】

```
@Transactional(rollbackFor = Exception.class)
public void audit(Long id, Integer status, String remark) {
    // 获取当前登录用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("未登录或登录已过期");
    }

    // 查询申请记录
    ClaimApplication claimApplication =
claimApplicationMapper.selectById(id);
    if (claimApplication == null) {
        throw new ServiceException("申请记录不存在");
    }
```

```
        // 检查申请状态，只能审核待审核的申请
        if (claimApplication.getStatus() != 0) {
            throw new ServiceException("该申请已处理，无需重复审核");
        }

        // 检查审核权限
        boolean hasPermission = false;

        if (claimApplication.getItemType() == 1) {
            // 失物信息，检查当前用户是否是发布者
            LostItem lostItem =
lostItemMapper.selectById(claimApplication.getItemId());
            if (lostItem != null) {
                // 物品发布者或管理员可以审核
                boolean isOwner =
lostItem.getUserId().equals(currentUser.getId());
                boolean isAdmin =
"ADMIN".equals(currentUser.getRoleCode());
                hasPermission = isOwner || isAdmin;
            }
        }

        if (!hasPermission) {
            throw new ServiceException("无权限审核此申请");
        }

        // 更新申请状态
        claimApplication.setStatus(status);
        claimApplication.setAuditUserId(currentUser.getId());
        claimApplication.setAuditTime(LocalDateTime.now());
        claimApplication.setAuditRemark(remark);
        claimApplication.setUpdateTime(LocalDateTime.now());

        claimApplicationMapper.updateById(claimApplication);

        // 如果审核通过，更新物品状态为已认领
        if (status == 1) {
            if (claimApplication.getItemType() == 1) {
                // 失物信息
                LostItem lostItem =
lostItemMapper.selectById(claimApplication.getItemId());
                if (lostItem != null) {
                    // 再次检查物品状态，防止并发问题
                    if (lostItem.getStatus() != 0) {
                        throw new ServiceException("该物品已被认领或已关闭，无法
继续处理");
                    }

                    lostItem.setStatus(1); // 设置为已认领
                    lostItemMapper.updateById(lostItem);
```

```
                // 拒绝其他申请
                rejectOtherApplications(claimApplication.getItemId(),
claimApplication.getItemType(), id);
            }
        }
    }

    // 发送审核结果通知
    sendAuditResultNotification(claimApplication, status == 1, remark);
}
```

这段代码处理认领申请的审核，包括验证用户权限、更新申请状态，如果通过则同时更新失物状态并拒绝其他申请，最后发送审核结果通知。

2.5.4 功能实现细节

失物认领申请功能包含以下细节处理：

- **申请限制**：同一用户不能重复申请同一物品，不能申请自己发布的物品。
- **状态检查**：只有"待认领"状态的物品才能申请认领。
- **权限控制**：只有物品发布者或管理员可以审核申请。
- **并发处理**：审核时再次检查物品状态，防止并发问题。
- **关联更新**：审核通过时自动更新物品状态，并拒绝其他申请。
- **通知机制**：申请提交和审核结果都会发送通知给相关用户。
- **事务管理**：使用Spring的@Transactional注解确保数据操作的原子性。

2.5.5 【功能截图占位】

2.6 失物通知功能

2.6.1 功能概述

失物通知功能为系统中的失物相关操作提供消息通知服务，包括状态变更通知、认领申请通知和审核结果通知。通知会保存在系统中，用户可以查看所有收到的通知。

2.6.2 详细讲解功能实现流程

通知发送流程：当系统中发生需要通知的事件（如状态变更、认领申请、审核结果）时，相关服务调用NotificationService的方法发送通知。NotificationService创建通知记录并保存到数据库。用户可以通过前端界面查看所有通知。

2.6.3 关键代码讲解

1. 通知发送代码
   【springboot/src/main/java/org/example/springboot/service/NotificationService.java】

```
@Transactional(rollbackFor = Exception.class)
public void sendNotification(Long userId, String title, String content,
                             Notification.NotificationType type, Long
relatedId) {
    try {
        Notification notification = new Notification();
        notification.setUserId(userId);
        notification.setTitle(title);
        notification.setContent(content);
        notification.setType(type.getValue());
        notification.setRelatedId(relatedId);
        notification.setIsRead(0); // 未读
        notification.setCreateTime(LocalDateTime.now());

        notificationMapper.insert(notification);
        log.info("通知发送成功: userId={}, title={}", userId, title);
    } catch (Exception e) {
        log.error("通知发送失败: userId={}, title={}, error={}", userId,
title, e.getMessage(), e);
        // 通知发送失败不应该影响主业务流程，所以这里只记录日志
    }
}

// 发送状态变更通知
public void sendStatusChangeNotification(Long userId, String itemTitle,
                                         String oldStatus, String
newStatus, Long itemId) {
    String title = "物品状态变更";
    String content = String.format("您发布的物品：%s，状态已从 %s 变更为
%s。",
                                   itemTitle, oldStatus, newStatus);
    sendNotification(userId, title, content,
                     Notification.NotificationType.SYSTEM, itemId);
}
```

这段代码实现了通知的发送功能，包括创建通知记录并保存到数据库。特别注意的是，通知发送失败不会影响主业务流程，只记录日志。

2.6.4 功能实现细节

失物通知功能包含以下细节处理：

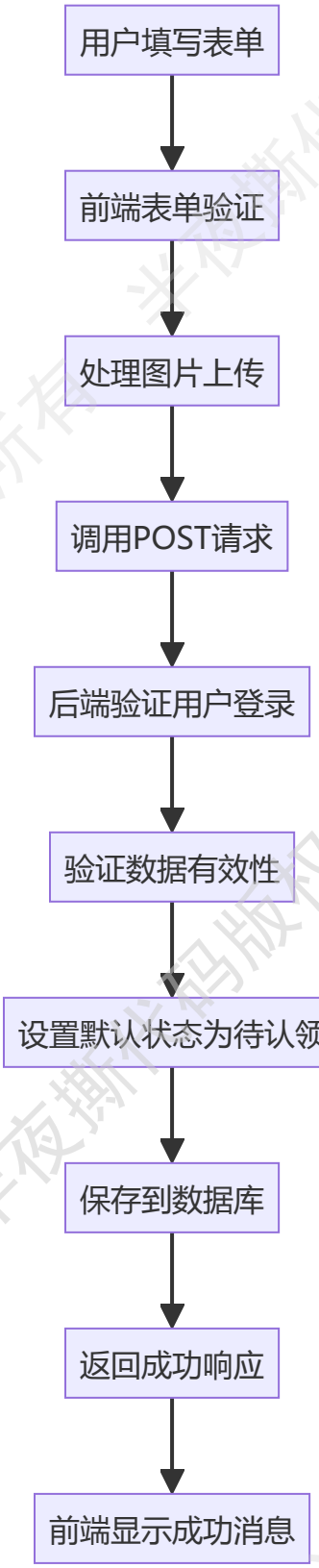
- **通知类型**：支持系统消息、申请消息和审核消息等多种类型。
- **容错处理**：通知发送失败不影响主业务流程，保证系统稳定性。
- **未读标记**：新通知默认标记为未读，用户查看后可标记为已读。
- **关联信息**：通知包含相关的物品或申请ID，便于用户快速查看详情。
- **时间记录**：保存通知创建时间，便于用户按时间顺序查看通知。

2.6.5 【功能截图占位】

三、总结

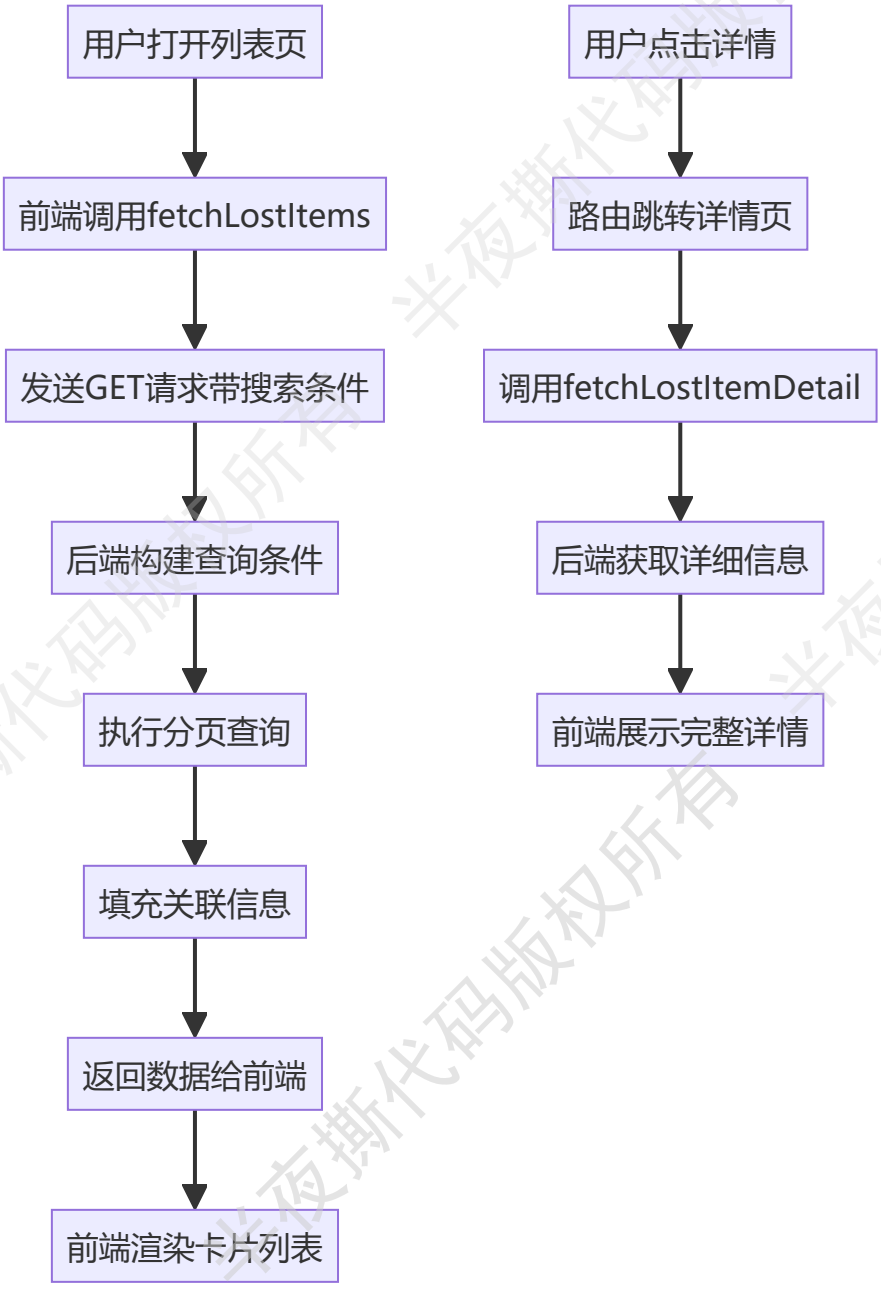
3.1 失物发布功能实现

失物发布功能通过前端的submitForm方法实现，该方法验证表单数据后调用request.post("/lost-item", formData)发送请求。后端LostItemService的addLostItem方法处理请求，首先通过JwtTokenUtils.getCurrentUser()获取当前用户，验证用户状态，对数据进行完整验证，设置默认待认领状态，并通过lostItemMapper.insert(lostItem)将数据保存到数据库。整个过程使用@Transactional注解确保数据一致性，任何异常都会导致事务回滚。



3.2 失物查询与展示功能实现

失物查询与展示功能通过前端的fetchLostItems方法实现，该方法向后端发送GET请求获取数据。后端LostItemService的queryByPage方法处理请求，构建查询条件，执行分页查询，并通过fillInfo方法填充分类名称和用户名等关联信息。前端以卡片形式展示查询结果，支持图片预览、详情查看等操作。详情页通过fetchLostItemDetail方法获取完整信息，展示物品的所有详细数据，并提供认领、编辑等操作选项。



3.3 失物编辑功能实现

失物编辑功能允许用户修改已发布的失物信息，前端通过submitForm方法向后端发送PUT请求更新数据。后端LostItemService的updateLostItem方法处理请求，首先验证用户权限（只有管理员或发布者可以编辑），然后验证数据有效性，保留原始创建者和创建时间，最后更新数据库记录。编辑页面支持图片管理，可维护现有图片和上传新图片，并且可以更新失物状态。

3.4 失物状态管理功能实现

失物状态管理功能通过ItemStatusService实现，支持手动更新状态和自动处理过期物品。手动更新通过updateLostItemStatus方法实现，该方法验证用户权限，检查状态转换合法性，更新数据库并发送通知。自动过期处理通过定时任务调用processExpiredItems方法实现，查询超过指定天数未认领的物品，将其标记为过期并发送通知。状态变更会触发NotificationService发送通知给相关用户。

3.5 失物认领申请功能实现

失物认领申请功能通过ClaimApplicationService实现，包括申请提交和申请审核两部分。用户可以申请认领他人发布的失物，系统验证申请有效性并通知物品发布者。发布者可以审核申请，通过或拒绝，审核结果会通知申请者。申请通过时会自动更新物品状态为"已认领"，并拒绝其他申请。整个过程有完善的权限控制和通知机制，确保操作的安全性和用户体验。

3.6 失物通知功能实现

失物通知功能通过NotificationService实现，为系统中的各种操作提供消息通知服务。包括状态变更通知、认领申请通知和审核结果通知等多种类型。通知记录保存在数据库中，用户可以查看所有收到的通知。特别注意的是，通知发送失败不会影响主业务流程，只记录日志，保证系统稳定性。通知功能增强了用户体验，使用户能够及时了解相关物品的状态变化。