门票预订模块讲解文档

一、数据库与实体类设计

1.1 核心实体表设计

1.1.1 Ticket实体表

属性名	属性含义	是否为空	备注
id	门票ID	否	自增主键
scenicId	景点ID	否	外键关联ScenicSpot表
ticketName	门票名称	否	-
price	门票价格	否	BigDecimal类型
discountPrice	优惠价格	是	BigDecimal类型
ticketType	门票类型	是	成人票/儿童票/学生票等
validPeriod	有效期	是	-
description	描述	是	-
stock	库存	是	-
status	状态	是	1-可预订, 0-不可预订
createTime	创建时间	是	自动生成
updateTime	更新时间	是	自动更新

1.1.2 TicketOrder实体表

属性名	属性含义	是否为空	备注
id	订单ID	否	自增主键

属性名	属性含义	是否为空	备注
orderNo	订单编号	否	唯一标识
userId	用户ID	否	外键关联User表
ticketId	门票ID	否	外键关联Ticket表
quantity	购买数量	否	-
visitorName	游客姓名	否	-
visitorPhone	游客手机号	否	-
idCard	身份证号	是	-
visitDate	游玩日期	否	LocalDate类型
totalAmount	订单总金额	否	BigDecimal类型
status	订单状态	是	0-待支付, 1-已支付, 2-已取消, 3-已退款, 4-已完成
paymentTime	支付时间	是	-
paymentMethod	支付方式	是	WECHAT/ALIPAY/BANK_CARD
createTime	创建时间	是	自动生成
updateTime	更新时间	是	自动更新
ticketName	门票名称	是	非持久化字段
scenicName	景点名称	是	非持久化字段

属性名	属性含义	是否为空	备注
username	用户名	是	非持久化字段

1.2 核心属性说明

- **价格精度**：使用BigDecimal类型存储价格，确保金额计算精确性
- **优惠价格**：支持设置优惠价格，实现门票促销功能
- **库存管理**：通过stock字段记录门票库存，支持库存扣减和恢复
- **订单状态**：通过status字段记录订单状态，支持完整的订单生命周期管理
- **支付方式**：支持多种支付方式，如微信、支付宝、银行卡等
- **游玩日期**：记录用户选择的游玩日期，便于景点安排和管理
- **非持久化字段**：多个非持久化字段用于前端展示，减少查询次数

二、各功能详细讲解

2.1 门票列表展示功能

2.1.1 功能概述

门票列表展示功能允许用户浏览系统中的所有可预订门票，支持按名称、类型和景点进行搜索，每个门票以卡片形式展示，包含名称、价格、类型、有效期等信息，用户可以点击卡片进入预订页面。

2.1.2 详细讲解功能实现流程

当用户访问门票列表页面时，前端通过onMounted钩子调用fetchTickets()方法，该方法向后端发送GET请求到/ticket/page接口，传递搜索条件和分页参数。后端TicketController的getTicketsByPage方法接收请求并调用ticketService.getTicketsByPage方法进行分页查询，只返回状态为可预订的门票。查询结果包含门票的基本信息，如名称、价格、类型、有效期等。前端接收数据后以网格布局渲染门票卡片，每张卡片显示门票名称、价格（优惠价格和原价）、类型、有效期等信息，并支持点击跳转到预订页面。

2.1.3 关键代码讲解

1. 代码位置：
   【springboot/src/main/java/org/example/springboot/service/TicketService.java】

```
public Page<Ticket> getTicketsByPage(String ticketName, String
ticketType, Long scenicId, Integer currentPage, Integer size) {
    LambdaQueryWrapper<Ticket> queryWrapper = new LambdaQueryWrapper<>
();
```

```
        // 添加查询条件
        if (StringUtils.isNotBlank(ticketName)) {
            queryWrapper.like(Ticket::getTicketName, ticketName);
        }
        if (StringUtils.isNotBlank(ticketType)) {
            queryWrapper.eq(Ticket::getTicketType, ticketType);
        }
        if (scenicId != null) {
            queryWrapper.eq(Ticket::getScenicId, scenicId);
        }

        // 只查询可预订的门票
        queryWrapper.eq(Ticket::getStatus, 1);

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

        return ticketMapper.selectPage(new Page<>(currentPage, size),
    queryWrapper);
    }
```

这段代码实现了门票的分页查询功能。它首先根据传入的条件构建查询条件，如果提供了门票名称，则使用like进行模糊查询；如果提供了门票类型，则精确匹配该类型；如果提供了景点ID，则精确匹配该景点的门票。然后，它通过status字段筛选出可预订的门票，并按创建时间降序排列，确保最新添加的门票显示在前面。最后，使用MyBatis-Plus的分页功能执行查询并返回结果。

2. 代码位置：【vue3/src/views/frontend/ticket/index.vue】

```
const fetchTickets = async () => {
  loading.value = true
  try {
    await request.get('/ticket/page', {
      ticketName: searchForm.ticketName,
      ticketType: searchForm.ticketType,
      scenicId: searchForm.scenicId,
      currentPage: currentPage.value,
      size: pageSize.value
    }, {
      showDefaultMsg: false,
      onSuccess: (res) => {
        ticketList.value = res.records||[]
        total.value = res.total||0
      }
    })
  } catch (error) {
    console.error('获取门票列表失败:', error)
  } finally {
```

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

这段代码是前端获取门票列表的实现。它通过request工具向后端发送GET请求，传递搜索条件和分页参数，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。整个过程使用loading状态控制加载动画，并使用try-finally结构确保无论请求成功与否都会重置加载状态。

2.1.4 功能实现细节

- **搜索功能**：支持按门票名称、类型和景点进行多条件搜索
- **分页控制**：使用el-pagination组件实现分页，支持页码跳转
- **价格显示**：如果存在优惠价格，同时显示优惠价格和原价，否则只显示原价
- **状态过滤**：只显示状态为可预订的门票，避免用户预订不可用门票
- **响应式设计**：使用网格布局实现响应式设计，自动适应不同屏幕尺寸
- **景点选择器**：提供景点选择器，支持按景点筛选门票

2.2 门票预订功能

2.2.1 功能概述

门票预订功能允许用户选择门票、填写游客信息并提交订单，系统会自动计算订单金额、生成订单号并扣减门票库存，用户可以在预订页面查看门票详细信息，包括价格、类型、有效期等。

2.2.2 详细讲解功能实现流程

当用户点击门票卡片时，前端通过router.push跳转到门票预订页面，传递门票ID作为路由参数。预订页面booking.vue组件通过onMounted钩子调用fetchTicketDetail()方法，该方法向后端发送GET请求到/ticket/{id}接口获取门票详情。后端TicketController的getTicketById方法接收请求并调用ticketService.getTicketById方法获取门票信息。前端接收数据后渲染门票信息卡片，并显示预订表单，用户填写游玩日期、购买数量、游客信息等。用户提交表单时，前端会验证表单数据的有效性，然后调用submitBooking方法向后端发送POST请求到/order接口创建订单。后端TicketOrderController的createOrder方法接收请求，调用ticketOrderService.createOrder方法处理订单创建逻辑，包括生成订单号、计算订单金额、扣减门票库存等。创建成功后，前端显示支付对话框，用户可以选择支付方式并完成支付。

2.2.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/ticket/booking.vue】

```
const submitBooking = async () => {
  if (!userStore.isLoggedIn) {
    ElMessage.warning('请先登录')
    router.push('/login?redirect=' +
encodeURIComponent(route.fullPath))
    return
  }

  bookingFormRef.value.validate(async (valid) => {
    if (valid) {
      loading.value = true
      try {
        await request.post('/order', bookingForm, {
          successMsg: '订单创建成功',
          onSuccess: (res) => {
            createdOrder.value = res
            payDialogVisible.value = true
          }
        })
      } catch (error) {
        console.error('创建订单失败:', error)
      } finally {
        loading.value = false
      }
    }
  })
}
```

这段代码实现了前端提交预订表单的功能。它首先检查用户是否已登录，如未登录则提示并跳转到登录页面。然后，它验证表单数据的有效性，验证通过后发送POST请求创建订单。请求成功后，会显示支付对话框，让用户选择支付方式。整个过程使用loading状态控制加载动画，并使用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/service/TicketOrderService.java】

```
@Transactional
public TicketOrder createOrder(TicketOrder order) {
    // 获取当前登录用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("用户未登录");
    }
    order.setUserId(currentUser.getId());
```

```
        // 检查门票是否存在且可预订
        Ticket ticket = ticketMapper.selectById(order.getTicketId());
        if (ticket == null) {
            throw new ServiceException("门票不存在");
        }
        if (ticket.getStatus() != 1) {
            throw new ServiceException("该门票暂不可预订");
        }

        // 检查库存是否足够
        if (ticket.getStock() < order.getQuantity()) {
            throw new ServiceException("门票库存不足");
        }

        // 生成订单号
        order.setOrderNo(generateOrderNo());

        // 设置订单状态为待支付
        order.setStatus(0);

        // 计算订单总金额
        order.setTotalAmount(ticket.getDiscountPrice() != null ?

     ticket.getDiscountPrice().multiply(java.math.BigDecimal.valueOf(order.
    getQuantity())) :

     ticket.getPrice().multiply(java.math.BigDecimal.valueOf(order.getQuant
    ity())));

        // 保存订单
        ticketOrderMapper.insert(order);

        // 扣减库存
        ticketService.updateTicketStock(order.getTicketId(),
    order.getQuantity());

        // 发送订单创建消息
        sendOrderMessage(order, "create");

        return order;
    }
```

这段代码实现了后端创建订单的核心逻辑。它首先验证用户登录状态和门票可用性，然后检查库存是否充足。接着，它生成订单号、设置订单状态为待支付、计算订单总金额，并将订单保存到数据库。保存成功后，它调用ticketService.updateTicketStock方法扣减门票库存，并发送订单创建消息。整个过程使用@Transactional注解确保事务一致性，如果任何步骤失败，都会回滚整个事务。

3. 代码位置：
【springboot/src/main/java/org/example/springboot/service/TicketService.java】

```
@Transactional
public void updateTicketStock(Long ticketId, Integer quantity) {
    Ticket ticket = ticketMapper.selectById(ticketId);
    if (ticket == null) {
        throw new ServiceException("门票不存在");
    }

    // 检查库存是否充足
    if (ticket.getStock() < quantity) {
        throw new ServiceException("门票库存不足");
    }

    // 更新库存
    ticket.setStock(ticket.getStock() - quantity);
    ticketMapper.updateById(ticket);
}
```

这段代码实现了更新门票库存的功能。它首先查询门票信息，然后检查库存是否充足，最后更新库存并保存到数据库。这种设计确保了库存操作的安全性，防止库存出现负数的情况。同样，它也使用@Transactional注解确保事务一致性。

2.2.4 功能实现细节

- **表单验证**：使用Element Plus的表单验证功能，确保游客信息的完整性和有效性
- **日期限制**：禁用今天之前的日期，确保用户只能选择有效的游玩日期
- **数量限制**：根据门票库存限制购买数量，防止超卖
- **价格计算**：自动计算订单总金额，优先使用优惠价格
- **用户验证**：检查用户登录状态，未登录用户会被引导到登录页面
- **库存管理**：创建订单时自动扣减库存，取消订单时自动恢复库存
- **事务管理**：使用@Transactional注解确保订单创建和库存更新的一致性

2.3 订单支付功能

2.3.1 功能概述

订单支付功能允许用户选择支付方式并完成订单支付，支持微信支付、支付宝支付和银行卡支付等多种支付方式，系统会根据用户选择的支付方式提供相应的支付界面，并在支付成功后更新订单状态。

2.3.2 详细讲解功能实现流程

创建订单成功后，前端显示支付对话框，用户可以选择支付方式。如果选择支付宝支付，点击"去支付"按钮会跳转到支付宝支付页面；如果选择其他支付方式，点击"确认已支付"按钮会直接调用支付接口。前端通过confirmPayment方法或goToAlipay方法处理支付请求，向后端发送POST请求到/order/{id}/pay接口，传递支付方式参数。后端TicketOrderController的payOrder方法接收请求，调用ticketOrderService.payOrder方法处理支付逻辑，包括验证订单状态、更新订单状态、记录支付时间和支付方式等。支付成功后，前端会显示支付成功页面，并提供查看订单和继续浏览门票的选项。

2.3.3 关键代码讲解

1. 代码位置：【vue3/src/views/frontend/ticket/booking.vue】

```
const confirmPayment = async () => {
  if (!paymentMethod.value) {
    ElMessage.warning('请选择支付方式')
    return
  }

  loading.value = true
  try {
    await request.post(`/order/${createdOrder.value.id}/pay`, null, {
      params: {
        paymentMethod: paymentMethod.value
      },
      successMsg: '支付成功',
      onSuccess: () => {
        payDialogVisible.value = false
        router.push('/orders')
      }
    })
  } catch (error) {
    console.error('支付订单失败:', error)
  } finally {
    loading.value = false
  }
}

const goToAlipay = () => {
  if (createdOrder.value && createdOrder.value.id) {
    router.push(`/payment/alipay/${createdOrder.value.id}`)
  } else {
    ElMessage.error('订单信息错误')
  }
}
```

这段代码实现了前端的支付功能。confirmPayment方法处理微信支付和银行卡支付，它首先检查用户是否已选择支付方式，然后发送POST请求到支付接口，请求成功后关闭支付对话框并跳转到订单列表页面。goToAlipay方法处理支付宝支付，它直接跳转到支付宝支付页面，该页面会加载支付宝支付表单。

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

```
public String createAlipayForm(Long orderId) {
    // 获取订单详情
    TicketOrder order = ticketOrderService.getOrderDetail(orderId);
    if (order == null) {
        throw new ServiceException("订单不存在");
    }

    // 创建支付宝客户端
    AlipayClient alipayClient = new DefaultAlipayClient(
            alipayConfig.getGateway(),
            alipayConfig.getAppId(),
            alipayConfig.getPrivateKey(),
            alipayConfig.getFormat(),
            alipayConfig.getCharset(),
            alipayConfig.getPublicKey(),
            alipayConfig.getSignType());

    // 创建支付请求
    AlipayTradePagePayRequest request = new
AlipayTradePagePayRequest();
    // 设置回调地址
    request.setReturnUrl(alipayConfig.getReturnUrl());
    request.setNotifyUrl(alipayConfig.getNotifyUrl());

    // 创建支付模型
    AlipayTradePagePayModel model = new AlipayTradePagePayModel();
    model.setOutTradeNo(order.getOrderNo());
    model.setTotalAmount(order.getTotalAmount().toString());
    model.setSubject("门票预订-" + order.getTicketName());
    model.setProductCode("FAST_INSTANT_TRADE_PAY");

    request.setBizModel(model);

    try {
        // 生成表单
        return alipayClient.pageExecute(request).getBody();
    } catch (AlipayApiException e) {
        LOGGER.error("生成支付宝支付表单失败: {}", e.getMessage());
        throw new ServiceException("生成支付宝支付表单失败");
    }
```

```
    }
```

这段代码实现了生成支付宝支付表单的功能。它首先获取订单详情，然后创建支付宝客户端和支付请求，设置回调地址和支付参数，最后生成支付表单并返回。这个表单会在前端页面中加载，用户可以直接在页面上完成支付宝支付。

3. 代码位置：
   【springboot/src/main/java/org/example/springboot/service/TicketOrderService.java】

```
@Transactional
public void payOrder(Long orderId, String paymentMethod) {
    // 获取当前登录用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("用户未登录");
    }

    // 获取订单信息
    TicketOrder order = ticketOrderMapper.selectById(orderId);
    if (order == null) {
        throw new ServiceException("订单不存在");
    }

    // 验证订单所属
    if (!order.getUserId().equals(currentUser.getId()) &&
!"ADMIN".equals(currentUser.getRoleCode())) {
        throw new ServiceException("无权操作此订单");
    }

    // 检查订单状态，只有待支付的订单可以支付
    if (order.getStatus() != 0) {
        throw new ServiceException("订单状态错误，无法支付");
    }

    // 更新订单状态为已支付
    order.setStatus(1);
    order.setPaymentTime(LocalDateTime.now());
    order.setPaymentMethod(paymentMethod);
    ticketOrderMapper.updateById(order);

    // 发送订单支付消息
    sendOrderMessage(order, "pay");
}
```

这段代码实现了支付订单的功能。它首先验证用户登录状态和订单所属权，然后检查订单状态是否为待支付，最后更新订单状态为已支付，并记录支付时间和支付方式。整个过程使用@Transactional注解确保事务一致性。

2.3.4 功能实现细节

- **多种支付方式**：支持微信支付、支付宝支付和银行卡支付等多种支付方式
- **支付宝集成**：使用支付宝SDK生成支付表单，实现在线支付功能
- **状态验证**：只有待支付状态的订单才能进行支付操作
- **权限控制**：验证用户是否有权操作该订单，防止越权操作
- **支付记录**：记录支付时间和支付方式，便于后续查询和统计
- **支付结果页**：显示支付结果页面，提供查看订单和继续浏览的选项

2.4 订单管理功能

2.4.1 功能概述

订单管理功能允许用户查看、支付和取消自己的订单，同时也允许管理员查看和管理所有订单，包括退款、完成等操作。系统提供了订单列表和订单详情页面，用户可以查看订单状态、支付信息、游客信息等详细内容。

2.4.2 详细讲解功能实现流程

用户点击"我的订单"菜单项时，前端通过router.push跳转到订单列表页面。该页面通过onMounted钩子调用fetchOrders()方法，向后端发送GET请求到/order/user接口获取当前用户的订单列表。后端TicketOrderController的getUserOrders方法接收请求，调用ticketOrderService.getUserOrders方法查询指定用户的订单，并通过fillOrderDetails方法填充订单关联信息（门票名称、景点名称等）。前端接收数据后以表格形式展示，包括订单号、门票名称、游玩日期、数量、金额、状态等信息，并根据订单状态提供不同的操作按钮。对于待支付订单，用户可以点击"支付"按钮继续支付，或点击"取消"按钮取消订单；对于已支付订单，用户可以查看订单详情。用户点击"取消"按钮时，前端调用cancelOrder()方法，向后端发送POST请求到/order/{id}/cancel接口。后端处理取消逻辑，包括验证订单所属、检查订单状态、更新订单状态为已取消、恢复门票库存。管理员可以访问后台订单管理页面，查看所有订单并进行更多操作，如退款处理。管理员点击"退款"按钮时，前端调用refundOrder()方法，向后端发送POST请求到/order/{id}/refund接口，后端处理退款逻辑，包括验证管理员权限、检查订单状态、更新订单状态为已退款、恢复门票库存。

2.4.3 关键代码讲解

1. 代码位置：
   【springboot/src/main/java/org/example/springboot/service/TicketOrderService.java】

```
public Page<TicketOrder> getUserOrders(Long userId, Integer
currentPage, Integer size) {
    if (userId == null) {
        // 获取当前登录用户
        User currentUser = JwtTokenUtils.getCurrentUser();
```

```
            if (currentUser == null) {
                throw new ServiceException("用户未登录");
            }
            userId = currentUser.getId();
        }

        // 查询用户订单
        LambdaQueryWrapper<TicketOrder> queryWrapper = new
LambdaQueryWrapper<>();
        queryWrapper.eq(TicketOrder::getUserId, userId);
        queryWrapper.orderByDesc(TicketOrder::getCreateTime);

        Page<TicketOrder> page = ticketOrderMapper.selectPage(new Page<>
(currentPage, size), queryWrapper);

        // 填充订单关联信息
        for (TicketOrder order : page.getRecords()) {
            fillOrderDetails(order);
        }

        return page;
    }

    private void fillOrderDetails(TicketOrder order) {
        // 填充门票信息
        Ticket ticket = ticketMapper.selectById(order.getTicketId());
        if (ticket != null) {
            order.setTicketName(ticket.getTicketName());

            // 填充景点信息
            ScenicSpot scenicSpot =
scenicSpotMapper.selectById(ticket.getScenicId());
            if (scenicSpot != null) {
                order.setScenicName(scenicSpot.getName());
            }
        }

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

这段代码实现了获取用户订单列表的功能。它首先获取当前登录用户ID，然后查询该用户的所有订单，并按创建时间降序排列。查询结果返回前，它会调用fillOrderDetails方法填充订单关联信息，包括门票名称、景点名称和用户信息，这样前端就可以直接显示这些信息，无需再次查询。

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

```
@Transactional
public void cancelOrder(Long orderId) {
    // 获取当前登录用户
    User currentUser = JwtTokenUtils.getCurrentUser();
    if (currentUser == null) {
        throw new ServiceException("用户未登录");
    }

    // 获取订单信息
    TicketOrder order = ticketOrderMapper.selectById(orderId);
    if (order == null) {
        throw new ServiceException("订单不存在");
    }

    // 验证订单所属
    if (!order.getUserId().equals(currentUser.getId()) &&
!"ADMIN".equals(currentUser.getRoleCode())) {
        throw new ServiceException("无权操作此订单");
    }

    // 检查订单状态，只有待支付的订单可以取消
    if (order.getStatus() != 0) {
        throw new ServiceException("只有待支付的订单可以取消");
    }

    // 更新订单状态为已取消
    order.setStatus(2);
    ticketOrderMapper.updateById(order);

    // 恢复门票库存
    ticketService.restoreTicketStock(order.getTicketId(),
order.getQuantity());

    // 发送订单取消消息
    sendOrderMessage(order, "cancel");
}
```

代码说明：cancelOrder方法使用@Transactional注解确保事务完整性，首先验证用户登录状态和订单所属权限，然后检查订单状态是否为待支付，最后更新订单状态为已取消并恢复门票库存。该方法包含多项业务校验，确保订单取消操作的合法性和数据一致性。

2.4.4 功能实现细节

- **权限控制**：普通用户只能查看和操作自己的订单，管理员可以查看和操作所有订单
- **状态流转**：订单状态从待支付→已支付→已完成，或待支付→已取消，已支付→已退款
- **库存管理**：取消订单或退款时自动恢复门票库存，确保库存数据准确性
- **关联数据填充**：查询订单时自动填充关联信息，避免前端多次请求
- **条件筛选**：支持按订单号、游客姓名、手机号和订单状态筛选订单
- **分页展示**：使用MyBatis-Plus的分页插件实现订单数据分页查询
- **事务管理**：使用@Transactional注解确保订单状态更新和库存操作在同一事务中

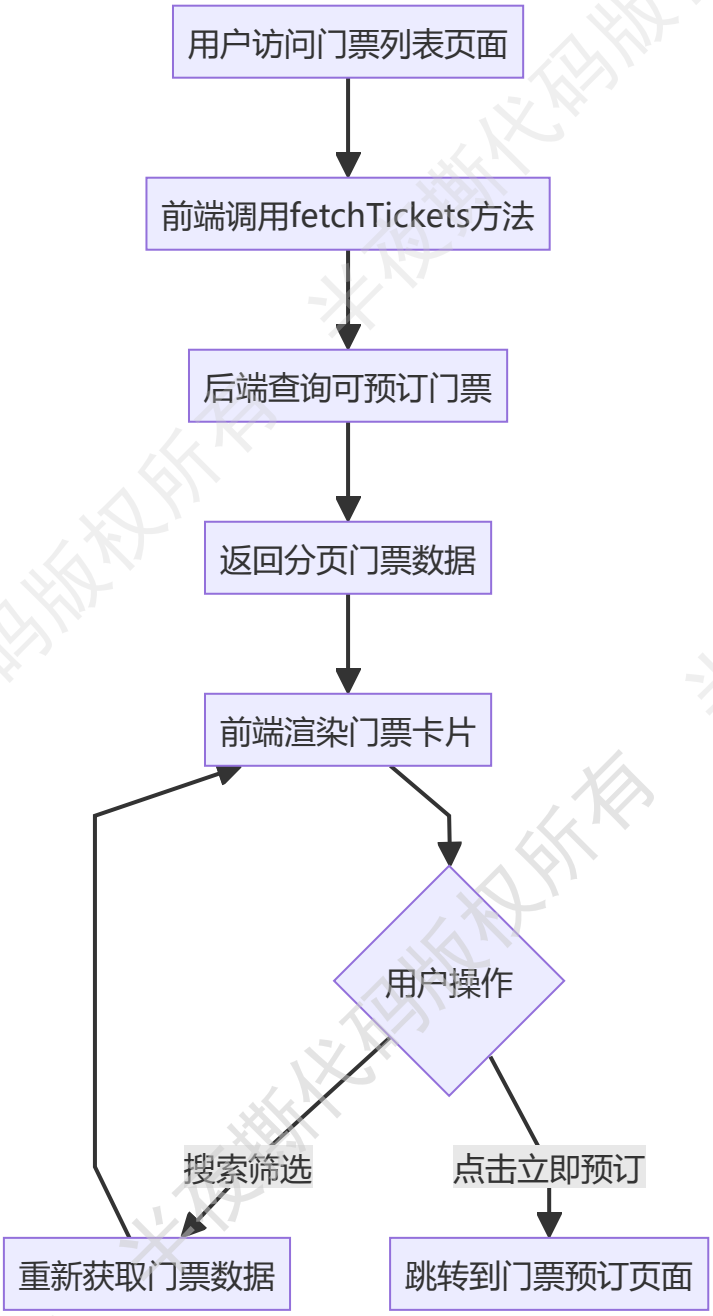
三、总结

3.1 门票列表浏览功能实现

门票列表浏览功能通过前端index.vue组件实现，该组件在加载时调用fetchTickets()方法获取可预订门票列表。后端TicketService.getTicketsByPage方法使用LambdaQueryWrapper构建查询条件，支持按名称模糊匹配、类型精确匹配和景点ID筛选，并通过status=1筛选出可预订门票。前端以卡片形
我是Claude 3.7 Sonnet模型。我将继续编写门票预订模块讲解文档。

3.1 门票列表浏览功能实现

门票列表浏览功能通过前端index.vue组件实现，该组件在加载时调用fetchTickets()方法获取可预订门票列表。后端TicketService.getTicketsByPage方法使用LambdaQueryWrapper构建查询条件，支持按名称模糊匹配、类型精确匹配和景点ID筛选，并通过status=1筛选出可预订门票。前端以卡片形式展示门票信息，包括名称、类型、价格和有效期，并根据是否有优惠价格动态调整显示样式。用户可通过搜索栏筛选门票，点击"立即预订"按钮跳转到预订页面。整个流程设计简洁明了，提供了良好的用户体验。



3.2 门票预订功能实现

门票预订功能通过booking.vue组件实现，该组件首先调用fetchTicketDetail()方法获取门票详情，然后展示预订表单供用户填写。用户填写游玩日期、购买数量和游客信息后，点击"提交订单"按钮触发submitBooking()方法，该方法验证表单数据并向后端发送创建订单请求。后端TicketOrderService.createOrder方法执行多项业务校验，包括检查用户登录状态、门票可用性和库存，然后生成订单号、计算总金额、保存订单并扣减库存。整个过程采用事务管理确保数据一致性，特别是订单创建与库存扣减的原子性操作，有效防止超卖问题。

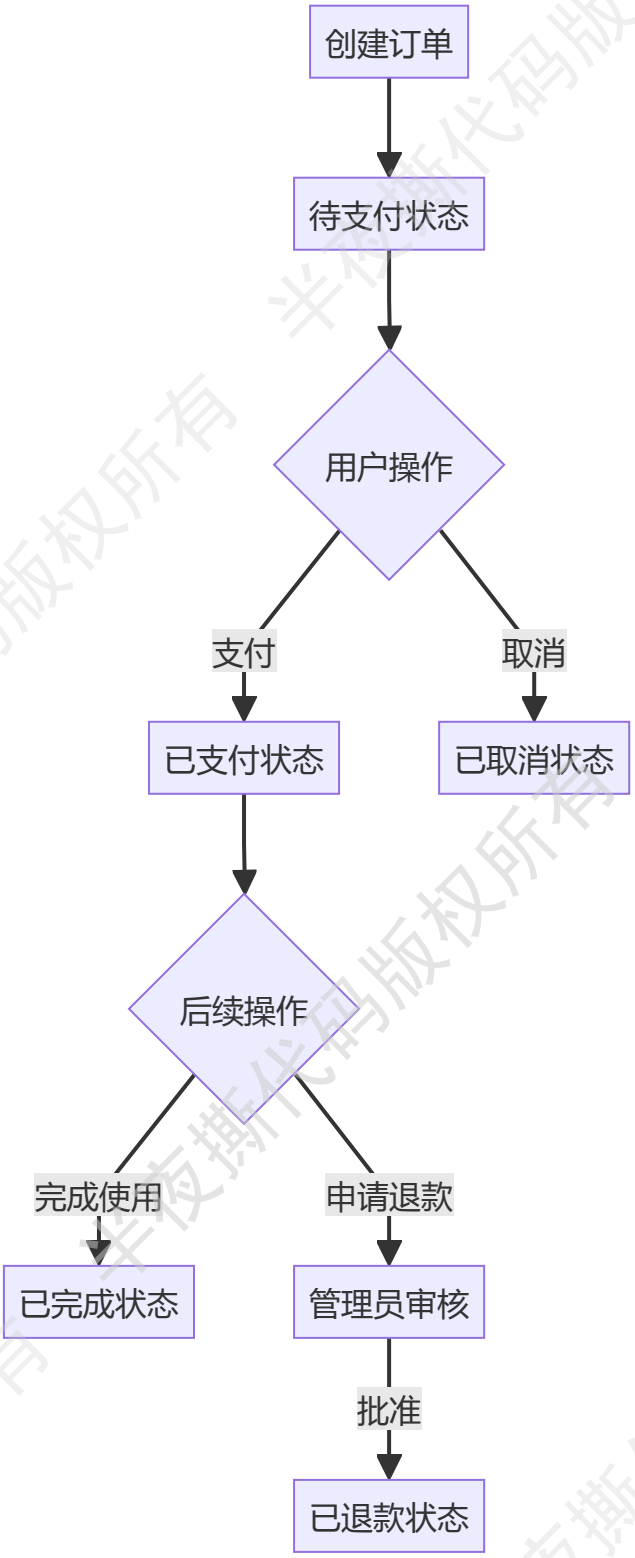
3.3 订单支付功能实现

订单支付功能支持支付宝支付和模拟支付两种方式。用户创建订单后，系统显示支付对话框，用户选择支付方式后进行支付。支付宝支付通过AlipayService.createAlipayForm方法生成支付表单，设置订单号、金额和商品名称等参数，用户提交表单后跳转到支付宝页面完成支付。支付成功后，系统通过同步回调和异步通知更新订单状态。模拟支付则通过

TicketOrderService.payOrder方法直接更新订单状态为已支付，记录支付方式和时间。支付完成后，前端跳转到支付结果页面显示相应信息，整个支付流程设计合理，确保了交易的安全性和可靠性。

3.4 订单管理功能实现

订单管理功能区分普通用户和管理员权限，普通用户只能查看和操作自己的订单，管理员可以查看和操作所有订单。用户可以取消待支付订单，管理员可以对已支付订单进行退款处理。订单取消和退款操作都会自动恢复门票库存，确保库存数据准确性。系统通过fillOrderDetails方法自动填充订单关联信息，提高了前端展示效率。订单状态流转清晰，从待支付→已支付→已完成，或待支付→已取消，已支付→已退款，每个状态变更都有严格的业务校验，确保操作的合法性和数据一致性。



总结来看，门票预订模块实现了完整的在线购票流程，从门票浏览、预订下单到支付完成，再到订单管理，各环节设计合理，数据处理安全可靠。系统特别注重数据一致性和业务校验，采用事务管理确保关键操作的原子性，有效防止了超卖等常见问题。前端界面设计简洁直观，表单验证完善，提供了良好的用户体验。支付功能的实现考虑了实际应用场景，支持多种支付方式，并有完善的支付结果处理机制。整个模块设计符合电子商务系统的基本要求，具有较高的实用性和可靠性。