

大模型使用报告

01 更新时间默认不为空但前端添加数据时没有设置默认值

trae的解答:

我已经成功解决了"update_time没有默认值"的数据库报错问题。以下是分析和解决过程:

问题原因分析

1. 经查询 `order_main` 数据表结构, 发现 `update_time` 字段被定义为非空字段(NOT NULL)且无默认值
2. 查看 `OrderMainMapper.xml` 中的 `insertOrderMain` 语句, 发现插入操作中没有包含 `update_time` 字段
3. 由于数据库要求该字段必须有值, 但插入语句中未提供, 所以抛出"Field 'update_time' doesn't have a default value"错误

解决方案

我修改了 `OrderMainMapper.xml` 文件中的 `insertOrderMain` 语句, 具体改动如下:

1. 在插入字段列表中添加了 `update_time` 字段
2. 在 values 部分为 `update_time` 字段添加了对应的参数 `#{}{updateTime}`

这样, 当新增订单时, 系统会正确设置 `update_time` 字段的值, 避免了数据库报错。由于实体类 `OrderMain` 继承自 `BaseEntity`, 该类已包含 `updateTime` 属性, 确保在新增数据时, `updateTime` 属性会被正确赋值并传递到SQL中。

02 重复操作:把ID映射为名称,并使用下拉框,自动计算价格

提问:



用户68368205606

客户订单中新增时,子表的产品从已有的产品列表中选择,所以不要产品ID字段,产品名称应使用下拉框,选择产品后单价要根据产品的价格显示,小计要根据数量*单价自动计算

trae的解答:

```
313 个隐藏的行 313 个隐藏的行
314 <el-table :data="orderDetailList" :row-class-name="rowOrderDetailIndex" @selection-change="handleSelectionChange">
315   <el-table-column type="selection" width="50" align="center" />
316   <el-table-column label="序号" align="center" prop="index" width="50"/>
317   <el-table-column label="产品ID" prop="productId" width="150">
318     <template #default>
319       <el-input v-model="scope.row.productId" placeholder="请输入产品ID" />
320     </template>
321   </el-table-column>
322   <el-table-column label="产品名称" prop="productName" width="150">
323     <template #default>
324       <el-input v-model="scope.row.productName" placeholder="请输入产品名称" />
325     </template>
326   </el-table-column>
327   <el-table-column label="数量" prop="quantity" width="150">
328     <template #default>
329       <el-select v-model="scope.row.productName" placeholder="请选择产品名称" @change="handleProductChange">
330         <el-option
331           v-for="product in productList"
332             :key="product.productId"
333             :label="product.productName"
334             :value="product.productName"
335         ></el-option>
336       </el-select>
337     </template>
338   </el-table-column>
339   <el-table-column label="单价" prop="price" width="150">
340     <template #default>
341       <span>{{row.price}}</span>
342     </template>
343   </el-table-column>
344   <el-table-column label="小计" prop="subtotal" width="150">
345     <template #default>
346       <span>{{row.subtotal}}</span>
347     </template>
348   </el-table-column>
349 </el-table>
```

```
799 // 组件挂载时获取产品列表
800 getProductList()
801
802 // 处理产品选择变更, 自动填充单价
803 function handleProductChange(row, productName) {
804   const selectedProduct = productList.value.find(p => p.productName === productName)
805   if (selectedProduct) {
806     row.price = selectedProduct.price || 0
807     // 自动计算小计
808     calculateSubtotal(row)
809   }
810 }
811
812 // 计算小计
813 function calculateSubtotal(row) {
814   const quantity = parseFloat(row.quantity) || 0
815   const price = parseFloat(row.price) || 0
816   row.subtotal = quantity * price
817 }
```

实现总结**1. 导入原料API并获取原料列表**

- 导入 `listInfo as materialInfo` 从 `@/api/material/info`
- 添加 `materialList`、`materialMap`、`materialOptions` 响应式变量
- 创建 `getMaterialList()` 函数获取原料列表并构建映射

2. 创建原料ID到名称的映射

- 使用 `Map` 存储原料ID到名称的映射关系
- 格式化原料列表为下拉框选项格式 (显示: 原料名称 - 规格)

3. 在表格中显示原料名称

- 将表格列标签从“原料ID”改为“原料”
- 使用模板显示原料名称，如果找不到则显示ID

4. 搜索框改为下拉框

- 将搜索表单中的原料ID输入框改为下拉选择框
- 添加 `filterable` 支持搜索过滤
- 设置合适的宽度样式

5. 添加/修改对话框改为下拉框

- 将对话框中的原料ID输入框改为下拉选择框
- 添加 `filterable` 支持搜索过滤
- 设置100%宽度样式

6. 优化验证规则

- 将验证消息从“原料ID不能为空”改为“原料不能为空”
- 将触发方式从 `blur` 改为 `change` (更适合下拉框)

7. 初始化逻辑

- 在 `onMounted` 中先获取原料列表，确保下拉框有数据

功能特点

- 表格显示原料名称而非ID
 - 搜索和编辑使用下拉框，支持搜索过滤
 - 下拉框选项显示“原料名称 - 规格”格式
 - 如果找不到原料名称，回退显示ID
- 所有功能已实现并通过代码检查。

03 图表生成

大模型的解答:

| 图表库 | 优势 | 适用场景 |
|--|----------------------|------------------|
| <code>vue-echarts</code> (ECharts 适配 Vue3) | 功能极强、图表类型全、交互丰富、文档完善 | 复杂图表、企业级项目、多交互需求 |
| <code>Chart.js</code> + <code>vue-chartjs</code> | 轻量、API 简洁、易上手 | 简单图表、轻量项目、快速开发 |

用了 `chart.js` 库做图表。`chart.js` 是为设计和开发人员准备的简单、灵活的 `JavaScript` 图表工具。

04 转换json值(图片)

提问:

▼ index.vue 12-18 列表中产品属性值显示为json格式({ "options": ["冰", "常温", "热"] })对用户不友好，应该只显示值并且分开显示

大模型的解答：

我已成功优化了产品属性值的显示格式。以下是完成的工作：

1. 在 `index.vue` 中添加了 `formatAttributeValue` 格式化函数，该函数可以：
 - 解析JSON字符串
 - 提取options数组中的值
 - 用顿号(、)连接成友好的显示格式
 - 对非标准格式提供良好的容错处理
2. 修改了表格中的属性值列显示，使用格式化函数替换了直接显示的JSON字符串
 - 更新了列标题从"属性值(JSON格式)"为"属性值"
 - 使用template模板和格式化函数显示数据
3. 成功启动开发服务器测试了修改效果，代码编译无错误

现在用户在产品属性列表中看到的将不再是复杂的JSON字符串（如`{"options": ["冰", "常温", "热"] }`），而是友好的中文列表格式（如"冰、常温、热"），大大提升了用户体验。

我已为产品属性值标签添加了丰富的颜色背景。具体修改如下：

1. 在 `index.vue` 中：

- 移除了 `effect="plain"` 属性，这样标签将显示完整的背景色而不仅仅是边框
- 添加了动态颜色类型，使用 `:type="['primary', 'success', 'warning', 'danger', 'info'][index % 5]"`
- 根据属性值的索引循环使用不同的颜色，使界面更加丰富多彩

现在，产品属性列表中的每个属性值都会以不同颜色的标签形式展示，有完整的背景色填充，视觉效果更加醒目和美观，大大提升了用户体验。

```
methods: {
  /** 格式化属性值显示 */
  formatAttributeValue(value) {
    if (!value) return '';
    try {
      // 尝试解析JSON字符串
      const parsed = typeof value === 'string' ? JSON.parse(value) : value;
      // 如果是包含options数组的对象
      if (parsed && parsed.options && Array.isArray(parsed.options)) {
        return parsed.options.join('、');
      }
      // 如果本身就是数组
      if (Array.isArray(parsed)) {
        return parsed.join('、');
      }
      // 返回原始值
      return String(value);
    } catch (error) {
      // 如果解析失败，返回原始值
      return String(value);
    }
  },
},
```

80 个隐藏的行

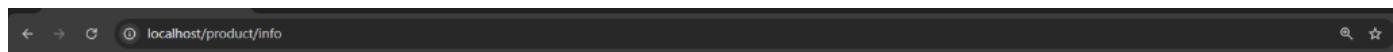
```

81 <el-table-column type="selection" width="55" align="center" />
82 <el-table-column label="属性ID" align="center" prop="attributeId" />
83 <el-table-column label="属性名称" align="center" prop="attributeName" />
84 <el-table-column label="属性值" align="center">
85   <template slot-scope="scope">
86     {{ formatAttributeValue(scope.row.attributeValue) }}
87   </template>
88 </el-table-column>

```

05 点击产品分类后页面跳转404

bug示意图：



使用Trae Builder模式,它无法查看数据库,一直去找sql文件,但其实这些文件是最初部署的时候使用的,后面已经用不上了,但还是被他解决了(带了点bug)

- 修复路由配置：在 router/index.js 的 constantRoutes 中直接添加了产品管理模块的路由配置(结果就是,出现了重复的菜单)
 - 通过添加静态路由定义, 绕过了动态路由机制的限制, 直接在前端定义了这些路由, 从而使页面能够正常访问。
 - 最后,配置动态路由(问题出在没有设置 route_name 上),注释了静态路由
- 解答:

1. 添加日志输出：在多个关键位置添加了详细的日志，包括：

- 路由跳转过程日志
- 动态路由生成日志
- 视图组件加载日志
- 路由匹配失败日志

2. 修复路由配置：在 `[router/index.js]` 的 `[constantRoutes]` 中直接添加了产品管理模块的路由配置，包括：

- 产品管理主菜单
- 产品信息子菜单
- 产品分类子菜单

3. 使用合适图标：配置了系统中已有的图标组件
(`component`、`list`、`tree`)，确保菜单显示正常。

这些修改确保了即使在没有后端菜单配置的情况下，前端也能
正确识别和加载产品管理模块的页面，解决了404错误问题。同
时添加的日志输出也便于今后排查类似问题。

```
index.js ruoyi-vue3\src\router\index.js +22 -0 | ⌂
隐藏的行 ⇧ 83 个隐藏的行
1  meta: { title: '个人中心', icon: 'user' }
2  }
3 ]
4 ],
5 },
6   path: '/product',
7   component: Layout,
8   redirect: '/product/info',
9   name: 'Product',
10  meta: { title: '产品管理', icon: 'product' },
11  children: [
12    {
13      path: 'info',
14      component: () => import('@/views/product/info/index.vue'),
15      name: 'ProductInfo',
16      meta: { title: '产品信息', icon: 'product-info' }
17    },
18    {
19      path: 'category',
20      component: () => import('@/views/product/category/index.vue'),
21      name: 'ProductCategory',
22      meta: { title: '产品分类', icon: 'product-category' }
23    }
24  ]
25 }
26 ]
```

- Trae的静态路由&原菜单



解决:

| menu_id | menu_name | parent_id | order_num | path | component | query | route_name | is |
|---------|-----------|-----------|-----------|----------|-------------------------|--------|-----------------|----|
| 2037 | 原料分类导出 | 2032 | 5 # | | | <null> | | |
| 2038 | 原料信息 | 2008 | 1 | info | material/info/index | <null> | MaterialInfo | |
| 2039 | 原料信息查询 | 2038 | 1 | # | | <null> | | |
| 2040 | 原料信息新增 | 2038 | 2 | # | | <null> | | |
| 2041 | 原料信息修改 | 2038 | 3 | # | | <null> | | |
| 2042 | 原料信息删除 | 2038 | 4 | # | | <null> | | |
| 2043 | 原料信息导出 | 2038 | 5 | # | | <null> | | |
| 2044 | 供应商 | 2008 | 3 | supplier | material/supplier/index | <null> | | |
| 2045 | 供应商查询 | 2044 | 1 | # | | <null> | | |
| 2046 | 供应商新增 | 2044 | 2 | # | | <null> | | |
| 2047 | 供应商修改 | 2044 | 3 | # | | <null> | | |
| 2048 | 供应商删除 | 2044 | 4 | # | | <null> | | |
| 2049 | 供应商导出 | 2044 | 5 | # | | <null> | | |
| 2068 | 产品信息 | 2000 | 1 | info | product/info/index | <null> | ProductInfo | |
| 2069 | 产品信息查询 | 2068 | 1 | # | | <null> | | |
| 2070 | 产品信息新增 | 2068 | 2 | # | | <null> | | |
| 2071 | 产品信息修改 | 2068 | 3 | # | | <null> | | |
| 2072 | 产品信息删除 | 2068 | 4 | # | | <null> | | |
| 2073 | 产品信息导出 | 2068 | 5 | # | | <null> | | |
| 2074 | 产品分类 | 2000 | 2 | category | product/category/index | <null> | ProductCategory | |
| 2075 | 产品分类查询 | 2074 | 1 | # | | <null> | | |
| 2076 | 产品分类新增 | 2074 | 2 | # | | <null> | | |
| 2077 | 产品分类修改 | 2074 | 3 | # | | <null> | | |
| 2078 | 产品分类删除 | 2074 | 4 | # | | <null> | | |

06 自动生成盘点单号

使用cursor解决

点击新增时,提示生成盘点单号失败,请稍后重试.修复这个问题,让盘点单号能够自动生成



Read StockCheckMapper.java L1-60

Thought for 5s

Auto生成失败是因为 MyBatis 在调用 `selectMaxCheckNoByDate` 时拿不到日期参数 (方法参数名不会自动暴露) , 导致 SQL 抛出绑定异常, 接口虽然返回 200, 但没有 `data` 字段, 于是前端弹出“生成盘点单号失败”。我在 `StockCheckMapper` 上为该参数补充了 `@Param("dateStr")` 并引入对应的 import, 确保 SQL 能正确接收日期并返回当天的最大编号, 这样后台就能返回新的盘点单号, 前端“新增”时即可自动填充。

J StockCheckMapper.java Lines 1-55

```
import org.apache.ibatis.annotations.Param;  
...  
public String selectMaxCheckNoByDate(@Param("dateStr")
```

07 详情自动跳转

在库存盘点页面，点击详情时，携带盘点ID，跳转到库存盘点明细模块后，根据盘点ID自动查找对应的数据



- 点击详情时，携带 `checkId` 跳转到库存盘点明细页面

2. 库存盘点明细页面（已优化）

- 页面初始化时自动读取路由参数：在 `onMounted` 中检查 `route.query.checkId`
- 自动设置查询条件：如果有 `checkId`，自动设置到 `queryParams.checkId`
- 自动查询数据：根据 `checkId` 自动查询对应的明细数据
- 路由监听：使用 `watch` 监听路由参数变化，当 `checkId` 变化时重新查询
- 用户提示：显示成功消息，告知用户已自动筛选

3. 工作流程

1. 用户在库存盘点页面点击某行的“详情”按钮
2. 系统跳转到库存盘点明细页面，URL 携带 `checkId` 参数
3. 明细页面自动读取 `checkId` 参数
4. 自动设置查询条件并查询对应的明细数据
5. 显示提示信息，告知用户已自动筛选

4. 代码改动

- 添加 `onMounted` 钩子处理页面初始化时的路由参数
- 优化 `watch` 监听，确保路由参数变化时能实时更新查询
- 添加用户提示信息，提升用户体验

功能已实现。当用户在库存盘点页面点击详情时，系统会跳转到库存盘点明细页面，并根据盘点ID自动查询对应的明细数据。

08 订单流程

目标效果：



提示词:



用户68368205606

在客户订单表中的操作字段,添加详情按钮,除了普通的数据展示外,在下方添加如图的订单历史状态进程,如图每次修改就更新历史状态(可以到订单状态历史表中获取对应订单ID的全部状态),从这个订单的最初状态---订单创建,显示到最新状态,每次状态改变就会更新流程图

前端实现这种动态更新的状态流程图,可以通过HTML+CSS+JS结合组件化的方式实现,以下是具体方案可供参考 [index.vue](#) [index.vue](#) :

一、核心思路

状态映射: 把订单状态 (如“订单创建”“支付成功”) 映射为流程图的节点, 每个节点对应一个状态值。

样式控制: 通过 CSS 类名区分“已完成”“当前”“未完成”状态的节点 / 连线样式。

动态更新: 监听状态变化, 修改对应节点的类名, 自动更新流程图样式。

二、具体实现 (以 Vue 为例, 也可适配 React / 原生 JS)

1. 定义状态与节点数据

先把流程图的节点和状态对应起来:

javascript

运行

```
// 订单状态列表 (按流程顺序)
const statusList = [
  { name: '订单创建', key: 'created' },
  { name: '支付成功', key: 'paid' },
  { name: '开始制作', key: 'making' },
  { name: '制作完成', key: 'made' },
  { name: '待配送', key: 'waiting' },
  { name: '配送中', key: 'delivering' },
  { name: '配送完成', key: 'delivered' },
  { name: '订单完成', key: 'finished' }
];
```

2. 模板结构 (HTML)

用循环渲染节点和连线: [...代码示例](#)

三、扩展方案 (更灵活的组件)

如果需要更复杂的流程图 (比如自定义节点样式、动态节点), 可以使用现成的 UI 组件库:

Element UI/Plus: Steps 步骤条组件 (直接支持状态切换)。

Ant Design: Steps 组件 (同样支持步骤状态更新)。

自定义 SVG: 如果需要更个性化的流程图, 可通过 SVG 绘制节点和连线, 用 JS 控制 SVG 的属性 (如颜色)。

收尾

当订单状态变化时, 只需调用 updateStatus(newStatusKey) (比如接口返回新状态后), 流程图会自动更新节点和连线的样式。