mport os import bianl # 假设 bianl 包含 main\_window 和 product\_table from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QTableWidget, QTableWidgetItem, QComboBox, QFileDialog, QFrame, QGroupBox, QHeaderView, QDateEdit, QMessageBox) from PyQt5.QtCore import Qt import common\_usage # 假设存在 from openpyxl import Workbook # 假设存在 import product\_confirm\_qianzhi # 假设存在 import traceback # 导入 traceback 模块 def handle\_confirm\_product(): print("--- handle\_confirm\_product function started ---") # 诊断信息：函数开始执行 try: # 包裹整个函数体以捕获任何未预料的异常 total\_rows = bianl.product\_table.rowCount() print(f"Total rows in table: {total\_rows}") # 诊断信息：总行数 action\_taken = False # 标记是否有实际操作发生 for row in range(total\_rows): print(f"\nProcessing row {row}...") # 诊断信息：开始处理行 if row == total\_rows - 1: print(f"Skipping last row ({row}).") # 诊断信息：跳过最后一行 continue # --- 获取表格项 --- product\_number\_item = bianl.product\_table.item(row, 1) product\_name\_item = bianl.product\_table.item(row, 2) device\_position\_item = bianl.product\_table.item(row, 4) product\_number = product\_number\_item.text().strip() if product\_number\_item else "" product\_name = product\_name\_item.text().strip() if product\_name\_item else "" device\_position = device\_position\_item.text().strip() if device\_position\_item else "" print(f"Row {row} data: PN='{product\_number}', Name='{product\_name}', Pos='{device\_position}'") # 诊断信息：行数据 # --- 检查当前行是否为空行（除了序号列） --- # 使用已经获取并处理过的变量 if not (product\_number or product\_name or device\_position): print(f"Row {row} is considered empty, skipping.") # 诊断信息：跳过空行 # 如果当前行是空行，则跳过处理 continue # --- 检查必填项是否填写完整 --- # 使用已经获取并处理过的变量 if not (product\_number and product\_name and device\_position): QMessageBox.warning(bianl.main\_window, "警告", f"第 {row + 1} 行存在未填写的必填项！ (产品编号, 产品名称, 设备工位)") print(f"Row {row} has missing required fields. Function will return.") # 诊断信息：必填项缺失 return # 发现不完整行，停止处理 # --- 获取和处理行状态 --- try: current\_status = product\_confirm\_qianzhi.get\_status(row) print(f"Row {row} status: {current\_status}") # 诊断信息：行状态 except Exception as e: print(f"Error getting status for row {row}: {e}\n{traceback.format\_exc()}") # 诊断信息：获取状态出错 QMessageBox.critical(bianl.main\_window, "错误", f"获取第 {row + 1} 行状态时出错: {e}") return # 获取状态失败，停止处理 # --- 根据状态执行操作 --- if current\_status == "start": print(f"Row {row} is in 'start' status. Processing as new product.") # 诊断信息：处理新产品 action\_taken = True # 标记有操作尝试 # 注意：确保 get\_input\_must\_var 不会引发未捕获的异常 try: product\_confirm\_qianzhi.get\_input\_must\_var(row) # 假设此函数从界面获取数据并存到 product\_confirm\_qianzhi 模块的变量中 except Exception as e: print(f"Error in get\_input\_must\_var for row {row}: {e}\n{traceback.format\_exc()}") QMessageBox.critical(bianl.main\_window, "错误", f"处理第 {row + 1} 行输入时出错: {e}") return # 检查产品是否存在 try: exists = product\_confirm\_qianzhi.check\_existing\_product( product\_number, product\_name, device\_position) except Exception as e: print(f"Error in check\_existing\_product for row {row}: {e}\n{traceback.format\_exc()}") # 诊断信息：检查存在性出错 QMessageBox.critical(bianl.main\_window, "错误", f"检查第 {row + 1} 行产品是否存在时出错: {e}") return # 检查存在性失败，停止处理 if exists: print(f"Row {row} product already exists.") # 诊断信息：产品已存在 QMessageBox.warning(bianl.main\_window, "提示", f"第 {row + 1} 行产品已存在！") # 这里使用 continue 而不是 return，因为可能需要处理其他行 continue # 继续处理下一行 # 保存新产品 try: print(f"Saving new product for row {row}...") # 诊断信息：尝试保存 product\_confirm\_qianzhi.save\_new\_product() # 假设此函数使用 get\_input\_must\_var 获取的数据 print(f"New product for row {row} saved successfully.") # 诊断信息：保存成功 product\_confirm\_qianzhi.update\_status(row, "view") # 更新状态 print(f"Row {row} status updated to 'view'.") # 诊断信息：状态更新 except Exception as e: print(f"Error saving new product for row {row}: {e}\n{traceback.format\_exc()}") # 诊断信息：保存出错 QMessageBox.critical(bianl.main\_window, "错误", f"保存第 {row + 1} 行新产品时发生错误: {e}") return # 保存失败，停止处理 elif current\_status == "edit": print(f"Row {row} is in 'edit' status. Processing as existing product.") # 诊断信息：处理编辑 action\_taken = True # 标记有操作尝试 # --- 获取旧数据（需要确保这些变量在 product\_confirm\_qianzhi 中被正确设置和获取）--- # 注意：这种依赖全局/模块级变量来存储“旧值”的方式可能不稳定， # 更好的方式是在进入编辑状态时就存储好该行的原始数据。 # 这里假设 product\_confirm\_qianzhi 能正确提供旧值 try: # 确保这些变量在调用 handle\_confirm\_product 前，或在 get\_status('edit') 时被填充 # 如果 curr\_row\_\* 不是 QLineEdit，而是普通字符串，就不需要 .text().strip() # 这里假设它们是 QLineEdit 或类似对象，如果不是，请直接使用变量 old\_product\_number = getattr(product\_confirm\_qianzhi, 'curr\_row\_product\_number', None) old\_product\_name = getattr(product\_confirm\_qianzhi, 'curr\_row\_product\_name', None) old\_device\_position = getattr(product\_confirm\_qianzhi, 'curr\_row\_device\_position', None) # 安全地获取文本值 old\_pn\_text = old\_product\_number.text().strip() if hasattr(old\_product\_number, 'text') else str(old\_product\_number or '') old\_name\_text = old\_product\_name.text().strip() if hasattr(old\_product\_name, 'text') else str(old\_product\_name or '') old\_pos\_text = old\_device\_position.text().strip() if hasattr(old\_device\_position, 'text') else str(old\_device\_position or '') print(f"Row {row} old values: PN='{old\_pn\_text}', Name='{old\_name\_text}', Pos='{old\_pos\_text}'") # 诊断信息：旧值 except AttributeError as e: print(f"Error accessing old value attributes in product\_confirm\_qianzhi for row {row}: {e}") QMessageBox.critical(bianl.main\_window, "错误", f"获取第 {row + 1} 行旧数据时出错: {e}. 请确保编辑状态被正确初始化。") return except Exception as e: print(f"Error getting old values for row {row}: {e}\n{traceback.format\_exc()}") QMessageBox.critical(bianl.main\_window, "错误", f"获取第 {row + 1} 行旧数据时发生意外错误: {e}") return # --- 比较新旧数据 --- if (old\_pn\_text != product\_number or old\_name\_text != product\_name or old\_pos\_text != device\_position): print(f"Row {row} data has changed. Attempting update.") # 诊断信息：数据已更改 # 更新现有产品 try: print(f"Updating existing product for row {row}...") # 诊断信息：尝试更新 product\_confirm\_qianzhi.update\_existing\_product(row) # 假设此函数使用表格当前行的数据 print(f"Existing product for row {row} updated successfully.") # 诊断信息：更新成功 product\_confirm\_qianzhi.update\_status(row, "view") print(f"Row {row} status updated to 'view'.") # 诊断信息：状态更新 except Exception as e: print(f"Error updating existing product for row {row}: {e}\n{traceback.format\_exc()}") # 诊断信息：更新出错 QMessageBox.critical(bianl.main\_window, "错误", f"更新第 {row + 1} 行产品信息时发生错误: {e}") return # 更新失败，停止处理 else: print(f"Row {row} data has not changed. No update needed.") # 诊断信息：数据未更改 else: # 如果状态不是 "start" 或 "edit"，打印出来 print(f"Row {row} has unhandled status '{current\_status}'. Skipping actions for this row.") # 诊断信息：未处理的状态 # --- 更新所有行的状态为 "view" （如果循环正常完成）--- # 这个逻辑可能需要调整：是否应该只更新被成功处理的行？ # 当前实现是，只要函数没有因 return 提前退出，所有行（除最后一行）都会被设为 view。 print("\nUpdating all processed rows status to 'view'...") # 诊断信息：开始最终状态更新 final\_update\_count = 0 for row in range(total\_rows - 1): try: # 检查行是否应该被更新状态（例如，跳过完全空行？） # 这里简单地更新所有行 product\_confirm\_qianzhi.update\_status(row, "view") final\_update\_count += 1 except Exception as e: # 记录错误但可能不中断，以便尝试更新其他行 print(f"Error updating status to 'view' for row {row}: {e}\n{traceback.format\_exc()}") QMessageBox.warning(bianl.main\_window, "警告", f"最终更新第 {row + 1} 行状态为 'view' 时出错: {e}") print(f"Finished updating status for {final\_update\_count} rows.") # 诊断信息：结束最终状态更新 if not action\_taken: print("No specific actions (save/update) were triggered during the process.") # 可以选择在这里给用户一个提示，比如 "没有检测到需要保存或更新的更改。" # QMessageBox.information(bianl.main\_window, "提示", "没有检测到需要保存或更新的更改。") except Exception as e: # --- 捕获任何未被内部 try...except 处理的异常 --- error\_message = f"An unexpected error occurred in handle\_confirm\_product: {e}" print(f"{error\_message}\n{traceback.format\_exc()}") # 打印详细的回溯信息到控制台 QMessageBox.critical(bianl.main\_window, "严重错误", f"{error\_message}\n\n请查看控制台输出获取详细信息。") finally: print("--- handle\_confirm\_product function finished ---") # 诊断信息：函数结束