```
import os
import tkinter as tk
from tkinter import ttk, filedialog
import pandas as pd
from openpyxl import Workbook
from openpyxl.styles import PatternFill
import ctypes  # 添加Windows API支持
from openpyxl.utils import get_column_letter
from openpyxl.styles import Font

class FileTreeExporter:
    def __init__(self):
        # Windows应用模型ID设置（必须放在Tk()初始化前）
        if os.name == 'nt':
            ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID('FileTreeExporter.1.0')

        self.window = tk.Tk()
        self.window.title("文件树导出工具")
        self.window.geometry("600x400")
        self.window.configure(bg='#1E1E1E')

        # 最终版图标设置（同时尝试.png和.ico）
        icon_paths = ['图标.ico', '图标.png']
        for path in icon_paths:
            try:
                self.window.iconbitmap(path)
                break
            except Exception as e:
                print(f"图标加载失败 ({path}): {str(e)}")

        # 全新暗色主题配置
        self.style = ttk.Style()
        self.style.theme_use('alt')

        # 全局颜色定义
        self.style.configure('.',
            background='#2D2D2D',
            foreground='#E0F2FE',
            fieldbackground='#3A3A3A',
            bordercolor='#4A90D6',
            lightcolor='#2D2D2D',
            darkcolor='#1E1E1E',
            troughcolor='#3A3A3A'
        )

        # 浅蓝色按钮样式
        self.style.configure('Blue.TButton',
```

```
                background='#5D8AA8',
                foreground='white',
                font=('微软雅黑', 10),
                borderwidth=1,
                focusthickness=3,
                focuscolor='#5D8AA8'
        )
        self.style.map('Blue.TButton',
                background=[('active', '#6C9DBF'), ('pressed', '#4A7794')],
                bordercolor=[('active', '#7AB4D6')]
        )

        # 复选框样式
        self.style.configure('TCheckbutton',
                indicatorbackground='#3A3A3A',
                indicatorcolor='#5D8AA8'
        )
        self.style.map('TCheckbutton',
                indicatorcolor=[('selected', '#5D8AA8'), ('!selected', '#3A3A3A')]
        )

        # 输入框样式
        self.style.configure('TEntry',
                fieldbackground='#3A3A3A',
                insertcolor='#E0F2FE'
        )

        # 初始化变量
        self.target_path = tk.StringVar()
        self.include_subfolders = tk.BooleanVar(value=True)
        self.include_extension = tk.BooleanVar(value=True)
        self.create_widgets()

def create_widgets(self):
        # 主容器框架
        main_frame = ttk.Frame(self.window)
        main_frame.pack(padx=20, pady=20, fill='both', expand=True)

        # 目录选择部分
        dir_frame = ttk.Frame(main_frame)
        dir_frame.pack(fill='x', pady=10)

        ttk.Label(dir_frame, text="目标目录:").pack(side='left', padx=5)
        ttk.Entry(dir_frame, textvariable=self.target_path, width=45).pack(side='left', padx=5)
        ttk.Button(dir_frame, text="浏览", command=self.select_directory, style='Blue.TButton').pack(side='left', padx=5)

        # 选项部分
```

```
        option_frame = ttk.Frame(main_frame)
        option_frame.pack(fill='x', pady=15)

        ttk.Checkbutton(option_frame, text="包含子文件夹", variable=self.include_subfolders).pack(side='top', anchor='w', pady=5)
        ttk.Checkbutton(option_frame, text="包含文件扩展名", variable=self.include_extension).pack(side='top', anchor='w', pady=5)

        # 导出按钮
        ttk.Button(main_frame, text="导出到Excel", style='Blue.TButton',
                   command=self.export_tree).pack(pady=20)

        # 状态栏
        self.status_label = ttk.Label(main_frame, foreground="#E0F2FE")
        self.status_label.pack(side='bottom', pady=10)

def select_directory(self):
        directory = filedialog.askdirectory()
        if directory:
                self.target_path.set(directory)
                self.status_label.config(text=f"已选目录: {directory}", foreground="green")

def add_entry(self, data, full_path):
        """添加文件条目到数据集（过滤文件夹）"""
        if os.path.isdir(full_path):
                return
        filename = os.path.basename(full_path)
        name, ext = os.path.splitext(filename)
        data.append((
                name,
                "文件",   # 固定类型为文件
                full_path,
                ext.lower() if ext else "",
                os.path.relpath(full_path, self.target_path.get())
        ))

def export_tree(self):
        if not self.target_path.get():
                self.status_label.config(text="错误：请先选择目录", foreground="red")
                return

        try:
                # 收集数据（保持第一次定义的数据收集逻辑）
                data = []
                if self.include_subfolders.get():
                        for root, dirs, files in os.walk(self.target_path.get()):
                                for name in files:
                                        path = os.path.join(root, name)
                                        self.add_entry(data, path)
```

```
else:
        for item in os.listdir(self.target_path.get()):
                path = os.path.join(self.target_path.get(), item)
                if os.path.isfile(path):
                        self.add_entry(data, path)

# 保持第一次定义的DataFrame创建和保存对话框逻辑
save_path = filedialog.asksaveasfilename(
        defaultextension='.xlsx',
        filetypes=[('Excel 文件', '*.xlsx'), ('All Files', '*.*')]
)
if not save_path:
        return

# 创建明细数据表（添加缺失的代码）
df_detail = pd.DataFrame(data, columns=['名称', '类型', '路径', '后缀', '相对路径'])

# 新增大纲数据透视表结构
base_path = self.target_path.get()   # 添加缺失的base_path定义
max_depth = 0
outline_data = []

# 首次遍历获取最大深度
for item in data:
        rel_path = os.path.relpath(item[2], base_path)
        path_parts = rel_path.split(os.sep) if rel_path != '.' else []
        max_depth = max(max_depth, len(path_parts))

# 创建透视表数据结构
for item in data:
        full_path = item[2]
        rel_path = os.path.relpath(full_path, base_path)
        path_parts = rel_path.split(os.sep) if rel_path != '.' else []

        row = {'完整路径': full_path, '类型': '文件'}
        for depth in range(max_depth):
                col_name = f'层级{depth+1}'
                row[col_name] = path_parts[depth] if depth < len(path_parts) else ''
        outline_data.append(row)

# 生成透视表DataFrame
df_outline = pd.DataFrame(outline_data)
df_outline = df_outline.reindex(columns=['完整路径', '类型'] + [f'层级{i+1}' for i in range(max_depth)])

# 修改Excel写入部分
with pd.ExcelWriter(save_path, engine='openpyxl') as writer:
        df_detail.to_excel(writer, sheet_name='文件明细', index=False)
```

```
# 写入透视表格式的大纲
df_outline.to_excel(writer, sheet_name='文件大纲', index=False)
outline_sheet = writer.sheets['文件大纲']

# 设置合并单元格逻辑
for col in range(3, max_depth+3):  # 从第3列开始是层级列
	col_letter = get_column_letter(col)
	prev_value = None
	merge_start = 2

	for row_num in range(2, len(df_outline)+2):
		current_value = outline_sheet[f'{col_letter}{row_num}'].value
		if current_value == prev_value and current_value != '':
			continue
		if merge_start < row_num - 1:
			outline_sheet.merge_cells(f"{col_letter}{merge_start}:{col_letter}{row_num-1}")
		merge_start = row_num
		prev_value = current_value

# 设置列样式
for col in range(3, max_depth+3):
	col_letter = get_column_letter(col)
	outline_sheet.column_dimensions[col_letter].width = 30
	for cell in outline_sheet[col_letter]:
		cell.alignment = Alignment(horizontal='center', vertical='center')

# 获取大纲工作表对象
outline_sheet = writer.sheets['文件大纲']

# 合并相同层级单元格（仅处理层级列）
for col in range(1, max_depth+1):
	prev_parent = None
	merge_start = 2
	col_letter = get_column_letter(col)

	rows = list(outline_sheet.iter_rows(min_row=2, values_only=True))
	for idx, row in enumerate(rows, 2):
		current_parent = os.path.dirname(outline_data[idx-2]['完整路径'])  # 获取实际父路径
		if current_parent != prev_parent:
			if prev_parent is not None and idx > merge_start:
				outline_sheet.merge_cells(f"{col_letter}{merge_start}:{col_letter}{idx-1}")
				# 设置居中对齐
				for row_num in range(merge_start, idx):
					cell = outline_sheet.cell(row=row_num, column=col)
					cell.alignment = Alignment(horizontal='center', vertical='center')
			merge_start = idx
```

```
                                prev_parent = current_parent

                        # 处理最后一批合并
                        if prev_parent is not None and len(rows) >= merge_start:
                                outline_sheet.merge_cells(f"{col_letter}{merge_start}:{col_letter}{len(rows)+1}")
                                for row_num in range(merge_start, len(rows)+2):
                                        cell = outline_sheet.cell(row=row_num, column=col)
                                        cell.alignment = Alignment(horizontal='center', vertical='center')

                # 设置列宽和标题样式（仅层级列）
                for col in range(1, max_depth+1):
                        col_letter = get_column_letter(col)   # 使用已导入的get_column_letter
                        outline_sheet.column_dimensions[col_letter].width = 30
                        cell = outline_sheet.cell(row=1, column=col)
                        cell.fill = PatternFill(fgColor="555555", fill_type="solid")
                        cell.font = Font(bold=True, color="FFFFFF")   # 使用已导入的Font

                # 获取工作表对象进行格式调整
                workbook = writer.book
                outline_sheet = writer.sheets['文件大纲']

                # 合并相同层级项
                prev_row = 1
                current_value = ""
                for idx, row in enumerate(df_outline.itertuples(), 1):
                        if row.名称 == current_value:
                                continue
                        if idx - prev_row > 1:
                                outline_sheet.merge_cells(f"A{prev_row}:A{idx-1}")
                                outline_sheet.merge_cells(f"B{prev_row}:B{idx-1}")
                                outline_sheet.merge_cells(f"C{prev_row}:C{idx-1}")
                        current_value = row.名称
                        prev_row = idx

                # 设置列宽
                outline_sheet.column_dimensions['A'].width = 10   # 层级列
                outline_sheet.column_dimensions['B'].width = 35   # 名称列
                outline_sheet.column_dimensions['C'].width = 15   # 类型列
                outline_sheet.column_dimensions['D'].width = 50   # 路径列

                # 设置列宽自适应
                for sheet in writer.sheets.values():
                        sheet.column_dimensions['A'].width = 30   # 名称列
                        sheet.column_dimensions['C'].width = 50   # 路径列
                        sheet.column_dimensions['D'].width = 15   # 后缀列

# 修改Excel格式设置部分
```

```
                        for sheet in writer.sheets.values():
                                sheet.sheet_properties.tabColor = "003366"   # 设置sheet标签颜色
                                # 添加标题行背景色
                                for cell in sheet[1]:
                                        cell.fill = PatternFill(fgColor="555555", fill_type="solid")

                        self.status_label.config(text=f"导出成功: {save_path}", foreground="green")

                except Exception as e:
                        self.status_label.config(text=f"错误: {str(e)}", foreground="red")

class ACCENT(ctypes.c_int):
        ACCENT_ENABLE_BLURBEHIND = 3

class WindowCompositionAttribute(ctypes.Structure):
        _fields_ = [
                ("Attribute", ctypes.c_int),
                ("Data", ctypes.POINTER(ctypes.c_ulong)),
                ("SizeOfData", ctypes.c_ulong)
        ]

if __name__ == "__main__":
        app = FileTreeExporter()
        app.window.mainloop()
```