通道逻辑架构设计：

1. **单通道数据模型**

Channel { id: int // 通道 ID (1-24) isEnabled: bool // 是否启用 audioParams { // 音频参数 volume: float frequency: float gain: float } filterParams { // 滤波器参数 type: enum // 滤波器类型 settings: dict // 具体设置 } status { // 通道状态 solo: bool mute: bool } }

2.状态管理策略

- **状态管理策略**
- **独立状态**
- 每个通道维护自己的状态
- 无需考虑通道间的状态同步
- 状态更新只影响单个通道

3.数据访问模式

- **按需加载**
- 点击通道按钮时加载对应通道数据
- 保持当前活跃通道数据在内存中
- 非活跃通道数据可以释放

4.缓存策略

活跃通道缓存 ↓ 最近使用的通道数据 (LRU Cache) ↓ 数据库存储

5.**界面交互流程**

[通道按钮点击] → [加载通道数据] → [显示过滤器界面] ↓ [参数调整] → [更新单通道] → [保存更改] → [更新 UI]

核心组件

1.ChannelManager

- **ChannelManager**
- 管理单个通道的所有操作
- 维护通道数据的生命周期
- 处理通道数据的加载和保存

2.FilterWidget

- **FilterWidget**
- 显示和控制单个通道的过滤器界面

- 处理参数调整
- 实时更新显示

3.**DataCache**

- **DataCache**
- 简单的 LRU 缓存
- 只缓存当前使用的通道数据
- 自动清理非活跃数据

数据流向

[用户界面] ↕ [ChannelManager] ←→ [DataCache] ↕ [数据库存储]

项目文件组织结构：

project/ ├── database/ | ├── **init**.py | ├── db_manager.py # 数据库管理（保持现有） | └── models.py # 数据模型（扩展现有） | ├── widgets/ | ├── **init**.py | ├── audio_filter_widget.py # 主界面（重构） | ├── Ui_widget.py # Qt Designer 生成的 UI（保持现有） | | | ├── components/ # 新增：组件目录 | | ├── **init**.py | | ├── filter_table.py # 滤波器表格组件 | | └── filter_controls.py # 滤波器控制组件 | | | ├── controllers/ # 新增：控制器目录 | | ├── **init**.py | | ├── channel_controller.py # 通道控制器 | | └── filter_controller.py # 滤波器控制器 | | | ├── models/ # 新增：模型目录 | | ├── **init**.py | | ├── channel_model.py # 通道数据模型 | | └── filter_model.py # 滤波器数据模型 | | | └── utils/ # 新增：工具目录 | ├── **init**.py | ├── cache.py # 缓存管理 | └── state.py # 状态管理 | └── data/ # 保持现有 └── database.db