# 代码分析报告

# 一、摘要

## 代码结构概览

service子路径中包含多个文件和类，整体结构如图所示：

service/  
├── ChordHMM.py  
│ ├── \_\_init\_\_  
│ ├── build\_transition\_matrix  
│ ├── generate\_chord\_sequence  
│ ├── predict\_chord  
│ └── viterbi  
├── ChordHMMV2.py  
│ ├── \_\_init\_\_  
│ ├── baum\_welch  
│ ├── generate\_chord\_sequence  
│ ├── predict\_next\_chords  
│ └── viterbi  
├── ChordVectorAnnoy.py  
│ ├── \_\_init\_\_  
│ ├── add\_chord  
│ ├── delete\_chord  
│ ├── search\_chord  
│ └── visualize\_emission\_matrix  
├── ChordVectorFaiss.py  
│ ├── \_\_init\_\_  
│ ├── add\_chord  
│ ├── delete\_chord  
│ ├── search\_chord  
│ └── visualize\_emission\_matrix  
├── MidiInput.py  
│ ├── \_\_init\_\_  
│ ├── \_enable\_keyboard\_mapping  
│ ├── \_initialize\_midi\_input  
│ └── \_on\_key\_press  
├── MatchingScales.py  
│ ├── find\_matching\_scales  
├── markov.py  
│ ├── predict\_chord  
├── numpyMarkov.py  
│ ├── build\_transition\_matrix  
│ ├── predict\_chord  
└── soundNoise.py  
 ├── \_\_init\_\_

这些文件与子目录的不同功能模块对应，实现HMM（隐马尔可夫模型）、音阶匹配、MIDI输入和声音噪声处理等功能。每个文件中包含多个类和函数，实现具体的功能。

### 代码设计风格分析

分析： 1. **命名规范**：大多数变量、函数和类的命名遵循Python的PEP8规范，可读性较强，例如find\_matching\_scales、initialize\_parameters。 2. **一致性**：Python的缩进标准统一为4个空格，代码风格较为一致。使用注释进行代码解释和说明，以便理解和维护。 3. **封装与抽象程度**：在实现ChordHMMV2.py和ChordHMM.py时，通过函数封装功能，使代码更加模块化。例如baum\_welch函数在一个单独的模块中，用于训练模型的参数。

### 关键功能与模块

1. **HMM模型训练与预测（ChordHMMV2.py，ChordHMM.py）**
   * 包含Baum-Welch算法、Viterbi算法、发射矩阵计算等核心功能。
   * 实现和弦序列预测的土地上，支持情绪和风格标签的上下文输入。
2. **声音向量索引（ChordVectorFaiss.py，ChordVectorAnnoy.py）**
   * 使用Faiss或Annoy存储音符向量索引，并实现添加、删除和搜索和弦功能。
   * 通过提取和弦特征向量进行索引和检索。
3. **MIDI输入处理（MidiInput.py）**
   * 初始化MIDI输入或键盘模式，捕捉和处理MIDI音符，并根据音符映射进行处理。

### 潜在问题

1. **资源释放与异常处理**：ChordVectorFaiss.py中的index.search方法未处理可能的Faiss错误。
2. **性能问题**：在索引较大的数据集上执行搜索操作时，可能会导致性能问题。例如，chord\_index的构建和搜索操作。
3. **代码复杂度**：ChordHMMV2.py和ChordHMM.py中的baum\_welch和viterbi算法实现较为复杂，难以阅读和调试。

### 重构建议

1. **函数职责单一**：ChordHMMV2.py和ChordHMM.py中的baum\_welch和viterbi算法可以分解为多个小函数，每个函数职责单一，便于测试和维护。
2. **简化代码逻辑**：通过引入更有意义的变量名、简化控制流转等方式，使代码逻辑更加清晰、易读。例如，将chord\_index替换为chord\_id\_map，将其作为成员变量。
3. **扩展测试覆盖率**：增加异常情况的测试覆盖，例如MIDI输入设备不可用的情况。可以通过单元测试和集成测试进行覆盖。

### 测试情况

* 按照文档中提供的目标子路径service，进行单元测试和集成测试，以验证主要功能。
* 通过pytest等工具进行单元测试，确保每个函数都能正常运行和返回预期结果。

# 二、附录明细

## 基础信息

* 仓库名称：chordPrediction
* 仓库描述：（示例仓库）和弦预测
* 仓库分支：master
* 仓库地址：https://github.com/kinglegendzzh/chordPrediction
* 项目根路径：C:\Users\apple\AppData\Roaming\GitHave\static\repo\chordPrediction
* 分析的目标子路径：service

## 函数信息

### baum\_welch (service/ChordHMMV2.py)

* 行号位置：141-200
* 重要性评分：19.00

**代码片段：**

def baum\_welch(self, max\_iters=10):  
 """  
 Baum-Welch算法，用于训练HMM参数，考虑上下文依赖。  
 :param max\_iters: 最大迭代次数  
 """  
 N = self.n\_states  
 M = self.n\_chords  
  
 sequences = [self.chords\_to\_indices(seq) for seq in self.chord\_sequences]  
  
 for iteration in range(max\_iters):  
 A\_num = np.zeros((N, N))  
 A\_den = np.zeros(N)  
 B\_num = np.zeros((N, M, self.n\_contexts))  
 B\_den = np.zeros((N, self.n\_contexts))  
  
 pi\_sum = np.zeros(N)  
  
 for O, emotions, style in zip(sequences, self.emotion\_labels, self.style\_labels):  
 context\_idx = self.context\_to\_index(emotions, style)  
 if context\_idx == -1:  
 continue # 跳过未知上下文  
  
 T = len(O)  
 alpha = self.forward(O, context\_idx, self.A, self.B, self.pi)  
 beta = self.backward(O, context\_idx, self.A, self.B)  
 xi = np.zeros((T - 1, N, N))  
 gamma = np.zeros((T, N))  
  
 # 计算xi和gamma  
 for t in range(T - 1):  
 denom = np.sum(alpha[t] \* beta[t])  
 for i in range(N):  
 gamma[t, i] = alpha[t, i] \* beta[t, i] / denom  
 b = self.B[:, O[t + 1], context\_idx]  
 xi[t, i, :] = alpha[t, i] \* self.A[i, :] \* b \* beta[t + 1] / denom  
  
 gamma[T - 1] = alpha[T - 1] \* beta[T - 1] / np.sum(alpha[T - 1] \* beta[T - 1])  
  
 # 累积A和B的更新量  
 A\_num += np.sum(xi, axis=0)  
 A\_den += np.sum(gamma[:-1], axis=0)  
 for t in range(T):  
 B\_num[:, O[t], context\_idx] += gamma[t]  
 B\_den[:, context\_idx] += np.sum(gamma, axis=0)  
  
 pi\_sum += gamma[0]  
  
 # 更新模型参数  
 self.A = A\_num / A\_den[:, None]  
 # 防止除零  
 self.A = np.nan\_to\_num(self.A)  
 self.B = B\_num / B\_den[:, None, :]  
 self.B = np.nan\_to\_num(self.B)  
 self.pi = pi\_sum / len(sequences)  
  
 # 防止数值问题，进行归一化  
 self.A /= self.A.sum(axis=1, keepdims=True)  
 self.B /= self.B.sum(axis=1, keepdims=True)  
 self.pi /= self.pi.sum()

* 功能描述： 该函数实现了Baum-Welch算法，用于训练隐马尔可夫模型（HMM）的参数，考虑上下文依赖。通过迭代更新状态转移矩阵A、发射矩阵B和初始状态概率pi，以最大化观察序列的概率。
* 实现流程： 初始化状态转移矩阵A、发射矩阵B和初始状态概率pi。 将和弦序列转换为对应的索引序列，并获取情绪标签和风格标签的上下文索引。 使用前向算法计算每个时间步每个状态的前向概率alpha。 使用后向算法计算贝塔值，用于计算给定观察序列的概率。 计算xi和gamma，用于累积A和B的更新量。 更新模型参数A、B和pi，防止除零和数值问题，进行归一化。 重复上述步骤，直到达到最大迭代次数或参数收敛。
* 引入包： time,pickle,utils.filePath,
* 调用： chords\_to\_indices,zeros,context\_to\_index,forward,backward,nan\_to\_num,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + context\_to\_index: 将情绪标签和风格标签转换为上下文索引。
  + forward: 前向算法用于计算HMM模型在给定观察序列下的状态概率序列，即Alpha矩阵。
  + backward: 后向算法用于计算贝塔值矩阵，该矩阵在隐马尔可夫模型中用于计算给定观察序列的后向概率。

### baum\_welch (service/ChordHMM.py)

* 行号位置：96-146
* 重要性评分：15.10

**代码片段：**

def baum\_welch(self, max\_iters=10):  
 """  
 Baum-Welch算法，用于训练HMM参数。  
 :param max\_iters: 最大迭代次数  
 """  
 N = self.n\_states  
 M = self.n\_chords  
 sequences = [self.chords\_to\_indices(seq) for seq in self.chord\_sequences]  
  
 for iteration in range(max\_iters):  
 A\_num = np.zeros((N, N))  
 A\_den = np.zeros(N)  
 B\_num = np.zeros((N, M))  
 B\_den = np.zeros(N)  
  
 pi\_sum = np.zeros(N)  
  
 for O in sequences:  
 T = len(O)  
 alpha = self.forward(O, self.A, self.B, self.pi)  
 beta = self.backward(O, self.A, self.B)  
 xi = np.zeros((T - 1, N, N))  
 gamma = np.zeros((T, N))  
  
 # 计算xi和gamma  
 for t in range(T - 1):  
 denom = np.sum(alpha[t] \* beta[t])  
 for i in range(N):  
 gamma[t, i] = alpha[t, i] \* beta[t, i] / denom  
 xi[t, i, :] = alpha[t, i] \* self.A[i, :] \* self.B[:, O[t + 1]] \* beta[t + 1] / denom  
  
 gamma[T - 1] = alpha[T - 1] \* beta[T - 1] / np.sum(alpha[T - 1] \* beta[T - 1])  
  
 # 累积A和B的更新量  
 A\_num += np.sum(xi, axis=0)  
 A\_den += np.sum(gamma[:-1], axis=0)  
 for t in range(T):  
 B\_num[:, O[t]] += gamma[t]  
 B\_den += np.sum(gamma, axis=0)  
  
 pi\_sum += gamma[0]  
  
 # 更新模型参数  
 self.A = A\_num / A\_den[:, None]  
 self.B = B\_num / B\_den[:, None]  
 self.pi = pi\_sum / len(sequences)  
  
 # 防止数值问题，进行归一化  
 self.A /= self.A.sum(axis=1, keepdims=True)  
 self.B /= self.B.sum(axis=1, keepdims=True)  
 self.pi /= self.pi.sum()

* 功能描述： 该函数实现了Baum-Welch算法，用于训练隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。通过迭代更新这些参数，使得模型能够更好地拟合给定的和弦序列。
* 实现流程： 初始化模型参数A、B和pi。 将和弦序列转换为对应的索引序列。 进行最大迭代次数的循环： 对于每个观察序列O，计算前向概率alpha和后向概率beta。 计算xi和gamma，用于更新A、B和pi的数值。 累积A、B和pi的更新量。 更新模型参数A、B和pi。 防止数值问题，进行归一化。 返回训练后的模型参数。
* 引入包： pickle,utils.filePath,
* 调用： chords\_to\_indices,zeros,forward,backward,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + forward: 前向算法用于计算HMM模型在给定观察序列下的状态概率序列，即Alpha矩阵。
  + backward: 后向算法用于计算贝塔值矩阵，该矩阵在隐马尔可夫模型中用于计算给定观察序列的后向概率。

### **init** (service/soundNoise.py)

* 行号位置：10-34
* 重要性评分：10.50

**代码片段：**

def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 logging.info('初始化 Fluidsynth 并加载 SoundFont')  
  
 # 检测操作系统并选择合适的音频驱动  
 system = platform.system()  
 if system == "Darwin":  
 driver = "coreaudio" # macOS 使用 coreaudio  
 elif system == "Windows":  
 driver = "dsound" # Windows 使用 DirectSound  
 elif system == "Linux":  
 driver = "alsa" # Linux 使用 ALSA  
 else:  
 driver = "pulseaudio" # 默认选择 pulseaudio  
  
 logging.info(f"Detected OS: {system}, 使用音频驱动: {driver}")  
  
 # 初始化 Fluidsynth，并启动合适的音频驱动  
 self.fs = fluidsynth.Synth()  
 self.fs.start(driver=driver)  
  
 # 加载 SoundFont  
 path = filePath('sounds/FluidR3\_GM/') + 'FluidR3\_GM.sf2'  
 sfid = self.fs.sfload(path)  
 self.fs.program\_select(0, sfid, 0, 0) # 通道 0，选择钢琴音色

* 功能描述： 初始化一个用于处理声音噪声的系统，并设置MIDI输入或键盘映射模式。它首先初始化Pygame和Pygame MIDI模块，然后调用内部函数初始化MIDI输入或启用键盘映射。
* 实现流程： 初始化 Fluidsynth 并加载 SoundFont 检测操作系统并选择合适的音频驱动 初始化 Fluidsynth，并启动合适的音频驱动 加载 SoundFont 选择钢琴音色
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： **init**,info,Synth,start,filePath,sfload,program\_select,
* 内部依赖描述：
  + **init**: 这是一个用于显示和更新音乐和弦的类，支持MIDI功能。它初始化了一个布局，设置了定时器来更新和弦和按钮颜色，并处理MIDI输入。

### delete\_chord (service/ChordVectorFaiss.py)

* 行号位置：117-143
* 重要性评分：8.70

**代码片段：**

def delete\_chord(self, chord\_name):  
 """  
 从索引中删除指定的和弦  
 """  
 # FAISS 的索引不支持直接删除，需要重建索引  
 vectors = []  
 new\_id\_mapping = {}  
 new\_index = 0  
  
 for idx, info in self.id\_mapping.items():  
 if info['chord\_name'] != chord\_name:  
 vector = self.midi\_notes\_to\_vector(info['midi\_notes'])  
 vectors.append(vector)  
 new\_id\_mapping[new\_index] = info  
 new\_index += 1  
  
 # 重置索引和映射  
 self.index = faiss.IndexFlatL2(self.vector\_dim)  
 self.id\_mapping = new\_id\_mapping  
 self.next\_id = new\_index  
  
 # 重新添加剩余的向量  
 if vectors:  
 self.index.add(np.array(vectors, dtype='float32'))  
  
 # 保存索引  
 self.save\_index()

* 功能描述： 该函数用于从索引中删除指定的和弦。它首先将索引中的所有和弦向量和映射信息遍历，跳过指定和弦，然后重建索引并重新添加剩余的向量，最后保存更新后的索引和映射。
* 实现流程： 遍历索引中的所有和弦向量和映射信息。 跳过指定和弦，将剩余的和弦向量和映射信息存储在新的列表和字典中。 重置索引和映射，创建一个新的索引对象。 将剩余的和弦向量添加到新的索引对象中。 保存更新后的索引和映射到指定路径。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： items,midi\_notes\_to\_vector,IndexFlatL2,add,array,save\_index,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。
  + save\_index: 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。

### visualize\_emission\_matrix (service/ChordHMMV2.py)

* 行号位置：288-302
* 重要性评分：8.50

**代码片段：**

def visualize\_emission\_matrix(self):  
 """  
 可视化发射矩阵B（仅针对单个上下文）。  
 """  
 # 选择第一个上下文进行可视化  
 if self.n\_contexts == 0:  
 print("没有可视化的数据。")  
 return  
 plt.figure(figsize=(10, 8))  
 plt.imshow(self.B[:, :, 0], cmap='hot', interpolation='nearest')  
 plt.colorbar()  
 plt.title("Emission Matrix Heatmap (Context 0)")  
 plt.xlabel("Chord Index")  
 plt.ylabel("Hidden State Index")  
 plt.show()

* 功能描述： 该函数用于可视化发射矩阵B（仅针对单个上下文）。它首先检查是否存在上下文数据，如果不存在则打印提示信息并返回。如果存在上下文数据，则使用matplotlib库创建一个热图，展示发射矩阵B的第一个上下文的数据，其中横轴表示音阶索引，纵轴表示隐藏状态索引。
* 实现流程： 检查发射矩阵B的上下文数量是否为0，如果是，则打印提示信息并返回。 创建一个大小为10x8的图形窗口。 使用matplotlib的imshow函数显示发射矩阵B的第一个上下文的数据，使用’hot’颜色映射和’nearest’插值方法。 添加颜色条以显示颜色与数值的对应关系。 设置图形的标题为’发射矩阵热图（上下文0）‘。 设置横轴标签为’音阶索引’，纵轴标签为’隐藏状态索引’。 显示图形窗口。
* 引入包： time,pickle,utils.filePath,
* 调用： figure,imshow,colorbar,title,xlabel,ylabel,show,
* 内部依赖描述：

### add\_chord (service/ChordVectorFaiss.py)

* 行号位置：42-85
* 重要性评分：8.40

**代码片段：**

def add\_chord(self, chord\_name, pressing\_str, force\_insert=True):  
 """  
 添加和弦到索引中  
  
 参数：  
 - chord\_name (str): 和弦名称  
 - pressing\_str (str): 按键字符串，格式如 'C2@12,16,19'  
 - force\_insert (bool): 是否强制插入，当为 False 时，若存在相同记录则不插入  
 """  
 # 如果 force\_insert 为 False，则在插入前检查是否已经存在相同的 chord\_name 和 pressing\_str  
 if not force\_insert:  
 # 遍历 id\_mapping，检查是否存在相同的 chord\_name 和 pressing\_str  
 for idx, info in self.id\_mapping.items():  
 if info['chord\_name'] == chord\_name and info['pressing\_str'] == pressing\_str:  
 # 已经存在相同的记录，不插入  
 print(f"Chord '{chord\_name}' with pressing '{pressing\_str}' already exists. Skipping insertion.")  
 return  
  
 # 解析 pressing\_str，提取 MIDI 音符序列  
 # 格式示例：C2@12,16,19  
 if '@' in pressing\_str:  
 root\_note, notes\_str = pressing\_str.split('@')  
 else:  
 # 处理没有根音的情况  
 notes\_str = pressing\_str  
 midi\_notes = list(map(int, notes\_str.split(',')))  
  
 # 将 MIDI 音符序列转换为向量  
 vector = self.midi\_notes\_to\_vector(midi\_notes)  
  
 # 获取当前索引的总数，作为新向量的 ID  
 vector\_id = self.index.ntotal  
  
 # 添加到索引中  
 self.index.add(np.array([vector]))  
  
 # 更新映射  
 self.id\_mapping[vector\_id] = {  
 'chord\_name': chord\_name,  
 'pressing\_str': pressing\_str,  
 'midi\_notes': midi\_notes  
 }  
  
 self.next\_id = vector\_id + 1

* 功能描述： 该函数用于向索引中添加和弦，并将其与按键字符串和 MIDI 音符序列关联。如果设置了强制插入标志，即使存在相同记录也会插入；否则，会检查是否存在相同记录，如果存在则跳过插入。
* 实现流程： 检查是否设置了强制插入标志，如果未设置则遍历索引中的记录，检查是否存在相同和弦名称和按键字符串的记录。如果存在则跳过插入。 解析按键字符串，提取根音和 MIDI 音符序列。 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。 获取当前索引的总数，作为新向量的 ID。 将新向量添加到索引中。 更新映射，将新向量的 ID 与和弦名称、按键字符串和 MIDI 音符序列关联。 更新下一个可用的 ID。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： items,midi\_notes\_to\_vector,add,array,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### visualize\_transition\_matrix (service/numpyMarkov.py)

* 行号位置：97-108
* 重要性评分：8.20

**代码片段：**

def visualize\_transition\_matrix(self):  
 """  
 可视化概率转移矩阵并输出为图片  
 """  
 plt.figure(figsize=(10, 8))  
 # 使用热图展示转移矩阵  
 plt.imshow(self.transitions, cmap='hot', interpolation='nearest')  
 plt.colorbar() # 添加颜色条  
 plt.title("Transition Matrix Heatmap")  
 plt.xlabel("Next Chord Index")  
 plt.ylabel("Current State Index")  
 plt.show()

* 功能描述： 该函数用于可视化概率转移矩阵，并将结果输出为图片。
* 实现流程： 创建一个大小为10x8的图形窗口。 使用热图展示转移矩阵，颜色映射为’hot’，插值方法为’nearest’。 添加颜色条以显示颜色与数值的对应关系。 设置图形的标题为’Transition Matrix Heatmap’。 设置X轴标签为’Next Chord Index’，Y轴标签为’Current State Index’。 显示图形窗口并输出图片。
* 调用： figure,imshow,colorbar,title,xlabel,ylabel,show,
* 内部依赖描述：

### visualize\_emission\_matrix (service/ChordHMM.py)

* 行号位置：214-224
* 重要性评分：8.10

**代码片段：**

def visualize\_emission\_matrix(self):  
 """  
 可视化发射矩阵B。  
 """  
 plt.figure(figsize=(10, 8))  
 plt.imshow(self.B, cmap='hot', interpolation='nearest')  
 plt.colorbar()  
 plt.title("Emission Matrix Heatmap")  
 plt.xlabel("Chord Index")  
 plt.ylabel("Hidden State Index")  
 plt.show()

* 功能描述： 该函数用于可视化发射矩阵B，通过热图展示发射概率，其中横轴表示音程索引，纵轴表示隐藏状态索引。
* 实现流程： 创建一个大小为10x8的图形窗口。 使用plt.imshow函数以热图形式显示发射矩阵B，颜色映射为’hot’，插值方法为’nearest’。 添加颜色条以显示颜色与发射概率的对应关系。 设置图形标题为’Emission Matrix Heatmap’。 设置横轴标签为’Chord Index’，纵轴标签为’Hidden State Index’。 显示图形窗口以供用户查看。
* 引入包： pickle,utils.filePath,
* 调用： figure,imshow,colorbar,title,xlabel,ylabel,show,
* 内部依赖描述：

### predict\_next\_chords (service/ChordHMMV2.py)

* 行号位置：228-257
* 重要性评分：8.00

**代码片段：**

def predict\_next\_chords(self, current\_sequence, emotions, style, threshold=0.1):  
 """  
 根据当前和弦序列预测下一个和弦，支持情绪标签和风格标签。  
 :param current\_sequence: 当前和弦列表  
 :param emotions: 情绪标签列表  
 :param style: 风格标签  
 :param threshold: 概率阈值，控制预测结果的数量  
 :return: 预测的和弦及其匹配概率的列表  
 """  
 # 将和弦转换为索引  
 O = self.chords\_to\_indices(current\_sequence)  
 # 如果序列中有未知和弦，返回None  
 if -1 in O:  
 print("当前序列包含未知和弦，无法预测。")  
 return None  
 context\_idx = self.context\_to\_index(emotions, style)  
 if context\_idx == -1:  
 print("未知的情绪或风格标签，无法预测。")  
 return None  
 # 获取最可能的隐藏状态序列  
 states = self.viterbi(O, context\_idx)  
 last\_state = states[-1]  
 # 获取可能的下一个和弦及其概率  
 next\_chord\_probs = self.B[last\_state, :, context\_idx]  
 # 过滤概率高于阈值的和弦  
 chord\_indices = np.where(next\_chord\_probs >= threshold)[0]  
 chords\_probs = [(self.index\_chord[i], next\_chord\_probs[i]) for i in chord\_indices]  
 # 按概率降序排序  
 chords\_probs.sort(key=lambda x: x[1], reverse=True)  
 return chords\_probs

* 功能描述： 该函数根据当前和弦序列、情绪标签和风格标签，预测下一个和弦，并返回预测和弦及其匹配概率的列表。它支持通过概率阈值控制预测结果的数量。
* 实现流程： 将当前和弦序列转换为对应的索引序列。 检查序列中是否包含未知和弦，如果包含则返回None。 将情绪标签和风格标签转换为上下文索引。 检查上下文索引是否有效，如果无效则返回None。 使用Viterbi算法获取最可能的隐藏状态序列。 获取当前隐藏状态序列的最后一个状态。 根据最后一个状态和上下文索引获取可能的下一个和弦及其概率。 过滤概率高于阈值的和弦。 将过滤后的和弦及其概率转换为和弦名称和概率的列表。 按概率降序排序预测结果。 返回排序后的预测和弦及其概率的列表。
* 引入包： time,pickle,utils.filePath,
* 调用： chords\_to\_indices,context\_to\_index,viterbi,where,sort,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + context\_to\_index: 将情绪标签和风格标签转换为上下文索引。
  + viterbi: Viterbi算法用于在给定观察序列的情况下，找到最可能的隐藏状态序列。

### viterbi (service/ChordHMMV2.py)

* 行号位置：202-226
* 重要性评分：7.50

**代码片段：**

def viterbi(self, O, context\_index):  
 """  
 Viterbi算法，寻找最可能的隐藏状态序列，考虑上下文。  
 :param O: 观察序列  
 :param context\_index: 上下文索引  
 :return: 最可能的状态序列  
 """  
 T = len(O)  
 N = self.n\_states  
 delta = np.zeros((T, N))  
 psi = np.zeros((T, N), dtype=int)  
  
 delta[0] = self.pi \* self.B[:, O[0], context\_index]  
 for t in range(1, T):  
 for j in range(N):  
 temp = delta[t - 1] \* self.A[:, j]  
 psi[t, j] = np.argmax(temp)  
 delta[t, j] = np.max(temp) \* self.B[j, O[t], context\_index]  
  
 states = np.zeros(T, dtype=int)  
 states[T - 1] = np.argmax(delta[T - 1])  
 for t in range(T - 2, -1, -1):  
 states[t] = psi[t + 1, states[t + 1]]  
  
 return states

* 功能描述： 该函数实现了Viterbi算法，用于在给定观察序列和上下文索引的情况下，寻找最可能的隐藏状态序列。
* 实现流程： 初始化delta和psi数组，用于存储每个时间步和状态的最大概率和前驱状态。 根据初始概率pi和发射概率B，计算第一个时间步的概率delta[0]。 从第二个时间步开始，遍历每个时间步，计算每个状态的最大概率delta[t, j]和前驱状态psi[t, j]。 初始化状态序列states，从最后一个时间步开始，根据psi数组回溯，找到最可能的状态序列。 返回最可能的状态序列。
* 引入包： time,pickle,utils.filePath,
* 调用： zeros,argmax,
* 内部依赖描述：

### viterbi (service/ChordHMM.py)

* 行号位置：148-171
* 重要性评分：7.40

**代码片段：**

def viterbi(self, O):  
 """  
 Viterbi算法，寻找最可能的隐藏状态序列。  
 :param O: 观察序列  
 :return: 最可能的状态序列  
 """  
 T = len(O)  
 N = self.n\_states  
 delta = np.zeros((T, N))  
 psi = np.zeros((T, N), dtype=int)  
  
 delta[0] = self.pi \* self.B[:, O[0]]  
 for t in range(1, T):  
 for j in range(N):  
 temp = delta[t - 1] \* self.A[:, j]  
 psi[t, j] = np.argmax(temp)  
 delta[t, j] = np.max(temp) \* self.B[j, O[t]]  
  
 states = np.zeros(T, dtype=int)  
 states[T - 1] = np.argmax(delta[T - 1])  
 for t in range(T - 2, -1, -1):  
 states[t] = psi[t + 1, states[t + 1]]  
  
 return states

* 功能描述： Viterbi算法用于在给定观察序列的情况下，找到最可能的隐藏状态序列。
* 实现流程： 初始化delta和psi数组，delta用于存储每个时间步每个状态的最大概率，psi用于存储每个时间步每个状态的最大概率对应的前一个状态。 计算初始时间步的概率，即delta[0] = pi \* B[:, O[0]]。 从第二个时间步开始，遍历每个时间步，对于每个状态，计算从上一个时间步到当前状态的最大概率，并更新delta和psi数组。 初始化状态序列数组states，从最后一个时间步开始，根据psi数组回溯，找到每个时间步最可能的状态。 返回最终的状态序列。
* 引入包： pickle,utils.filePath,
* 调用： zeros,argmax,
* 内部依赖描述：

### **init** (service/ChordVectorFaiss.py)

* 行号位置：10-30
* 重要性评分：7.10

**代码片段：**

def \_\_init\_\_(self, vector\_dim=128, index\_path='chord\_index.faiss', mapping\_path='id\_mapping.pkl'):  
 self.vector\_dim = vector\_dim  
 self.index\_path = index\_path  
 self.mapping\_path = mapping\_path  
  
 # 初始化索引  
 self.index = None  
  
 # 用于存储内部 ID 与和弦信息的映射  
 self.id\_mapping = {}  
  
 # 加载已有的索引和映射  
 if os.path.exists(self.index\_path) and os.path.exists(self.mapping\_path):  
 self.index = faiss.read\_index(self.index\_path)  
 with open(self.mapping\_path, 'rb') as f:  
 self.id\_mapping = pickle.load(f)  
 self.next\_id = max(self.id\_mapping.keys()) + 1  
 else:  
 # 使用 L2 距离的平面索引  
 self.index = faiss.IndexFlatL2(self.vector\_dim)  
 self.next\_id = 0

* 功能描述： 该类用于管理一个基于 Faiss 的向量索引，并提供加载和保存索引以及映射的功能。
* 实现流程： 初始化时，设置向量维度、索引路径和映射路径。 检查索引文件和映射文件是否存在，如果存在则加载索引和映射，并设置下一个可用的 ID。 如果文件不存在，则创建一个新的平面索引，并初始化下一个可用的 ID 为 0。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： exists,read\_index,keys,IndexFlatL2,
* 内部依赖描述：

### **init** (service/ChordHMMV2.py)

* 行号位置：11-40
* 重要性评分：7.00

**代码片段：**

def \_\_init\_\_(self, chord\_sequences=None, emotion\_labels=None, style\_labels=None, n\_states=3):  
 """  
 初始化HMM模型，用于和弦序列预测，支持情绪标签和风格标签作为上下文。  
 :param chord\_sequences: 和弦序列的列表，每个序列是一个和弦列表  
 :param emotion\_labels: 每个序列对应的情绪标签列表（列表的列表，支持多个情绪标签）  
 :param style\_labels: 每个序列对应的风格标签列表（单个标签）  
 :param n\_states: HMM的隐藏状态数  
 """  
 self.n\_states = n\_states # 隐藏状态数  
 if chord\_sequences is not None:  
 self.chord\_sequences = chord\_sequences  
 self.emotion\_labels = emotion\_labels  
 self.style\_labels = style\_labels  
 self.chord\_index = self.index\_chords(chord\_sequences) # 为每个和弦分配唯一索引  
 self.n\_chords = len(self.chord\_index) # 不同和弦的数量  
 self.index\_chord = {i: chord for chord, i in self.chord\_index.items()} # 索引到和弦的映射  
 self.contexts = self.build\_contexts(emotion\_labels, style\_labels) # 构建上下文  
 self.context\_index = {ctx: i for i, ctx in enumerate(self.contexts)}  
 self.n\_contexts = len(self.contexts)  
 self.initialize\_parameters() # 初始化HMM参数  
 else:  
 self.chord\_sequences = []  
 self.emotion\_labels = []  
 self.style\_labels = []  
 self.chord\_index = {}  
 self.n\_chords = 0  
 self.index\_chord = {}  
 self.context\_index = {}  
 self.n\_contexts = 0  
 # A, B, pi将在加载模型时被设置

* 功能描述： 该函数用于初始化一个隐马尔可夫模型（HMM），用于和弦序列预测，并支持情绪标签和风格标签作为上下文。它通过为和弦分配唯一索引、构建上下文以及随机初始化HMM参数来完成初始化。
* 实现流程： 为和弦序列中的每个和弦分配一个唯一的索引，同时为未知和弦分配一个特殊的索引。 通过将情绪标签列表和风格标签组合成元组，并将这些元组添加到集合中，构建上下文列表。 随机初始化HMM的参数，包括状态转移矩阵A、发射矩阵B（上下文依赖）和初始状态概率pi。
* 引入包： time,pickle,utils.filePath,
* 调用： index\_chords,items,build\_contexts,initialize\_parameters,
* 内部依赖描述：
  + index\_chords: 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
  + build\_contexts: 该函数用于构建上下文，通过将情绪标签列表和风格标签组合成元组，并将这些元组添加到集合中，最终返回一个包含所有唯一上下文的列表。
  + initialize\_parameters: 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。

### generate\_chord\_sequence (service/ChordHMMV2.py)

* 行号位置：259-286
* 重要性评分：6.80

**代码片段：**

def generate\_chord\_sequence(self, length, emotions, style):  
 """  
 使用训练好的HMM生成和弦序列，支持情绪标签和风格标签。  
 :param length: 要生成的序列长度  
 :param emotions: 情绪标签列表  
 :param style: 风格标签  
 :return: 生成的和弦序列  
 """  
 context\_idx = self.context\_to\_index(emotions, style)  
 if context\_idx == -1:  
 print("未知的情绪或风格标签，无法生成序列。")  
 return None  
  
 states = []  
 observations = []  
  
 state = np.random.choice(self.n\_states, p=self.pi)  
 for \_ in range(length):  
 states.append(state)  
 chord\_probs = self.B[state, :, context\_idx]  
 chord\_probs /= chord\_probs.sum() # 归一化  
 chord = np.random.choice(self.n\_chords, p=chord\_probs)  
 observations.append(chord)  
 state\_probs = self.A[state]  
 state = np.random.choice(self.n\_states, p=state\_probs)  
  
 chords = [self.index\_chord[chord] for chord in observations]  
 return chords

* 功能描述： 该函数使用训练好的HMM模型生成指定长度的和弦序列，并支持情绪和风格标签的输入。
* 实现流程： 将输入的情绪标签和风格标签转换为上下文索引。 检查上下文索引是否有效，如果无效则返回None。 初始化状态和观察序列列表。 随机选择一个初始状态。 根据当前状态和上下文索引计算每个和弦的概率，并选择一个和弦。 根据当前状态的转移概率选择下一个状态。 重复上述步骤直到生成指定长度的和弦序列。 将生成的和弦索引转换为和弦名称，并返回和弦序列。
* 引入包： time,pickle,utils.filePath,
* 调用： context\_to\_index,choice,
* 内部依赖描述：
  + context\_to\_index: 将情绪标签和风格标签转换为上下文索引。

### **init** (service/ChordVectorAnnoy.py)

* 行号位置：9-25
* 重要性评分：6.70

**代码片段：**

def \_\_init\_\_(self, vector\_dim=128, index\_path='chord\_index.ann',  
 mapping\_path='id\_mapping.pkl'):  
 self.vector\_dim = vector\_dim  
 self.index\_path = index\_path  
 self.mapping\_path = mapping\_path  
  
 self.index = AnnoyIndex(self.vector\_dim, 'hamming')  
 self.id\_mapping = {}  
  
 if os.path.exists(self.index\_path) and os.path.exists(self.mapping\_path):  
 self.index.load(self.index\_path)  
 with open(self.mapping\_path, 'rb') as f:  
 self.id\_mapping = pickle.load(f)  
 self.next\_id = max(self.id\_mapping.keys()) + 1  
 self.index.unbuild()  
 else:  
 self.next\_id = 0

* 功能描述： 初始化一个用于存储向量索引的类，支持加载已有的索引和映射文件，或者从头开始创建新的索引和映射。
* 实现流程： 初始化类的属性，包括向量维度、索引路径和映射路径。 检查索引路径和映射路径是否存在。 如果存在，加载索引和映射文件，并记录下一个可用的ID。 如果不存在，初始化索引和映射，并设置下一个可用的ID为0。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： AnnoyIndex,exists,keys,unbuild,
* 内部依赖描述：

### search\_chord (service/ChordVectorFaiss.py)

* 行号位置：95-115
* 重要性评分：6.10

**代码片段：**

def search\_chord(self, midi\_notes, top\_k=5):  
 """  
 根据给定的 MIDI 音符序列搜索最相似的和弦  
 """  
 vector = self.midi\_notes\_to\_vector(midi\_notes)  
 vector = np.array([vector], dtype='float32')  
  
 distances, indices = self.index.search(vector, top\_k)  
  
 results = []  
 for idx, distance in zip(indices[0], distances[0]):  
 if idx == -1:  
 continue  
 chord\_info = self.id\_mapping.get(idx)  
 if chord\_info:  
 results.append({  
 'chord\_name': chord\_info['chord\_name'],  
 'pressing\_str': chord\_info['pressing\_str'],  
 'distance': distance  
 })  
 return results

* 功能描述： 该函数根据给定的 MIDI 音符序列搜索最相似的和弦。
* 实现流程： 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。 将转换后的向量向量化，并设置数据类型为 float32。 使用索引搜索与向量最相似的 top\_k 个和弦。 遍历搜索结果，获取每个和弦的索引和距离。 根据索引从映射表中获取和弦信息。 将和弦名称、按压字符串和距离添加到结果列表中。 返回包含最相似和弦信息的结果列表。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： midi\_notes\_to\_vector,array,search,get,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### **init** (service/MidiInput.py)

* 行号位置：25-35
* 重要性评分：6.10
* 功能描述： 该类初始化一个用于处理声音噪声的系统，并设置MIDI输入或键盘映射模式。它首先初始化Pygame和Pygame MIDI模块，然后调用内部函数初始化MIDI输入或启用键盘映射。
* 实现流程： 初始化Pygame和Pygame MIDI模块。 调用内部函数 \_initialize\_midi\_input 初始化MIDI输入或启用键盘映射。 在 \_initialize\_midi\_input 函数中，打印所有可用的MIDI设备信息。 提示用户选择一个MIDI设备。 如果用户选择设备，则初始化该MIDI设备。 如果用户跳过选择，则启用键盘映射模式。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： **init**,SoundNoise,init,\_initialize\_midi\_input,
* 内部依赖描述：
  + **init**: 这是一个用于显示和更新音乐和弦的类，支持MIDI功能。它初始化了一个布局，设置了定时器来更新和弦和按钮颜色，并处理MIDI输入。
  + \_initialize\_midi\_input: 该函数用于初始化MIDI设备或键盘映射。首先打印所有可用的MIDI设备信息，然后让用户选择一个设备。如果用户选择设备，则初始化该MIDI设备；如果用户跳过选择，则启用键盘映射模式。

### \_on\_key\_press (service/MidiInput.py)

* 行号位置：79-95
* 重要性评分：5.70
* 功能描述： 该函数处理键盘按下事件，当焦点不在 QLineEdit 上时，根据按键字符执行相应的操作。如果按下的是反引号或波浪号，则切换键盘监听开关。如果监听启用且按键字符在音符映射中，则调用\_handle\_note\_press函数处理音符按下事件。
* 实现流程： 获取当前焦点的控件。 如果焦点在 QLineEdit 上，不触发 MIDI 音符映射。 检查按键字符是否为反引号或波浪号，如果是，则切换键盘监听开关，并记录日志。 如果监听启用且按键字符在音符映射中，则将按键字符转换为大写，并检查该音符是否在活动键集合中。 如果音符不在活动键集合中，则将其添加到集合中，并触发按键按下事件。 忽略特殊键（如 Shift 等）的异常情况。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： focusWidget,info,upper,\_handle\_note\_press,
* 内部依赖描述：
  + \_handle\_note\_press: 处理按下音符，如果音符不在活动键集合中，则将其添加到集合中，并触发按键按下事件。

### predict\_chord (service/markov.py)

* 行号位置：32-54
* 重要性评分：5.30
* 功能描述： 该函数用于预测音乐中的下一个和弦。它通过分析当前和弦序列来确定下一个和弦，并计算该和弦的概率。
* 实现流程： 获取当前状态：根据当前和弦序列和马尔科夫链的长度，确定当前状态。 匹配马尔科夫链：检查当前状态是否存在于马尔科夫链中。 选择下一个和弦：如果当前状态存在，从马尔科夫链中选择下一个和弦；否则，从预定义的和弦序列中随机选择。 计算概率：根据选择的和弦在马尔科夫链中的出现次数，计算其概率。 更新状态：将当前状态和选择的和弦更新为新的状态，并获取新的马尔科夫链状态。 返回结果：返回选择的和弦及其概率。
* 引入包： random,
* 调用： choice,get,
* 内部依赖描述：

### predict\_chord (service/numpyMarkov.py)

* 行号位置：73-95
* 重要性评分：5.30
* 功能描述： 该函数根据当前和弦列表预测下一个和弦，并过滤出概率大于阈值的和弦。它首先检查当前和弦的数量是否少于马尔科夫链的阶数，然后将当前和弦状态转换为索引，获取状态转移概率，并过滤和排序概率大于阈值的和弦，最后返回按概率降序排列的和弦及其概率。
* 实现流程： 检查当前和弦的数量是否少于马尔科夫链的阶数，如果少于则抛出异常。 将当前和弦状态转换为对应的索引。 获取当前和弦状态对应的转移概率。 过滤出概率大于阈值的和弦。 按概率降序排序过滤后的和弦。 规范化排序后的和弦概率。 返回按概率降序排列的和弦及其概率。
* 调用： ValueError,state\_to\_index,keys,
* 内部依赖描述：
  + state\_to\_index: 将和弦状态转换为对应的索引，用于状态表示和状态转移的计算。

### \_initialize\_midi\_input (service/MidiInput.py)

* 行号位置：37-47
* 重要性评分：5.10
* 功能描述： 该函数用于初始化MIDI设备或键盘映射。首先打印所有可用的MIDI设备信息，然后让用户选择一个设备。如果用户选择设备，则初始化该MIDI设备；如果用户跳过选择，则启用键盘映射模式。
* 实现流程： 打印所有可用的MIDI设备信息。 让用户选择一个MIDI设备。 如果用户选择设备，则初始化该MIDI设备。 如果用户跳过选择，则启用键盘映射模式。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： \_print\_midi\_devices,\_choose\_device,\_enable\_keyboard\_mapping,\_initialize\_midi\_device,
* 内部依赖描述：
  + \_print\_midi\_devices: 该函数用于打印所有可用的MIDI设备的信息。
  + \_choose\_device: 该函数用于手动选择MIDI设备。用户可以通过输入设备ID来指定设备，如果用户按下回车键，则跳过设备选择，使用默认的键盘映射。
  + \_enable\_keyboard\_mapping: 启用键盘映射模式，记录键盘输入并进行相应的映射处理。
  + \_initialize\_midi\_device: 初始化MIDI设备，尝试通过pygame.midi.Input方法打开指定ID的MIDI输入设备，并设置use\_keyboard\_mapping为False。如果初始化失败，抛出ValueError异常。

### delete\_chord (service/ChordVectorAnnoy.py)

* 行号位置：72-81
* 重要性评分：5.00
* 功能描述： 该函数用于从索引中删除指定的和弦，并更新相关的映射和索引。
* 实现流程： 根据和弦名称查找所有相关的索引ID。 遍历这些索引ID，逐个从索引中移除项，并从映射中删除对应的条目。 在所有项被移除后，重新构建索引并保存更新。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： items,unbuild,remove\_item,build\_index,
* 内部依赖描述：
  + build\_index: 构建索引并保存

### generate\_chord\_sequence (service/ChordHMM.py)

* 行号位置：193-212
* 重要性评分：5.00
* 功能描述： 该函数使用训练好的隐马尔可夫模型（HMM）生成指定长度的和弦序列。
* 实现流程： 初始化状态和观测值列表为空。 随机选择一个初始状态，概率分布由初始状态概率矩阵pi决定。 循环指定长度的次数，每次循环中： 将当前状态添加到状态列表中。 根据当前状态的概率分布矩阵B，随机选择一个和弦作为观测值，并添加到观测值列表中。 根据当前状态的概率分布矩阵A，随机选择下一个状态。 将观测值列表中的索引和弦转换为实际的和弦名称。 返回生成的和弦序列。
* 引入包： pickle,utils.filePath,
* 调用： choice,
* 内部依赖描述：

### predict\_next\_chord (service/ChordHMM.py)

* 行号位置：173-191
* 重要性评分：4.90
* 功能描述： 该函数根据当前和弦序列预测下一个和弦。它首先将和弦序列转换为对应的索引序列，然后使用Viterbi算法找到最可能的隐藏状态序列，最后根据该状态序列预测下一个和弦。
* 实现流程： 将当前和弦序列转换为对应的索引序列。 检查序列中是否包含未知和弦，如果包含则返回None。 使用Viterbi算法找到最可能的隐藏状态序列。 根据最后一个隐藏状态，从转移概率矩阵B中获取下一个和弦的概率分布。 选择概率最高的和弦索引。 将索引转换为对应的和弦并返回。
* 引入包： pickle,utils.filePath,
* 调用： chords\_to\_indices,viterbi,argmax,
* 内部依赖描述：
  + chords\_to\_indices: 将和弦序列转换为对应的索引序列。
  + viterbi: Viterbi算法用于在给定观察序列的情况下，找到最可能的隐藏状态序列。

### **init** (service/ChordHMM.py)

* 行号位置：9-27
* 重要性评分：4.90
* 功能描述： 初始化一个HMM模型，用于和弦序列预测。该模型能够处理和弦序列，并为每个和弦分配一个唯一的索引，同时支持未知和弦的处理。
* 实现流程： 接收和弦序列列表和隐藏状态数作为参数。 初始化隐藏状态数。 如果提供了和弦序列列表，则为每个和弦分配一个唯一的索引，并记录不同和弦的数量。 创建索引到和弦的映射。 初始化HMM模型的参数。 如果未提供和弦序列列表，则初始化为空列表和空映射，并标记参数将在加载模型时设置。
* 引入包： pickle,utils.filePath,
* 调用： index\_chords,items,initialize\_parameters,
* 内部依赖描述：
  + index\_chords: 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
  + initialize\_parameters: 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。

### build\_transition\_matrix (service/numpyMarkov.py)

* 行号位置：31-55
* 重要性评分：4.50
* 功能描述： 该函数根据输入的和弦序列构建概率转移矩阵，通过计算每个状态到下一个状态的转移次数，并将其归一化为概率，从而得到一个表示和弦序列转移概率的矩阵。
* 实现流程： 遍历输入的和弦序列，对于每个和弦序列，计算其马尔科夫链阶数的转移概率。 将当前状态（和弦组合）转换为对应的索引，用于状态表示和状态转移的计算。 获取下一个和弦的索引，处理未知和弦的情况。 确保索引不越界，更新转移矩阵中当前状态到下一个状态的计数。 对转移矩阵的每一行进行归一化处理，将计数转换为概率。 最终得到一个表示和弦序列转移概率的矩阵。
* 调用： state\_to\_index,get,
* 内部依赖描述：
  + state\_to\_index: 将和弦状态转换为对应的索引，用于状态表示和状态转移的计算。

### find\_matching\_scales (service/MatchingScales.py)

* 行号位置：41-64
* 重要性评分：4.40
* 功能描述： 该函数用于找到与输入音符集匹配的音阶。它首先将根音转换为 MIDI 音高类编号，然后合并所有和弦的音符并去重。接着，对于每个音阶，它将其转调到根音，并计算与输入音符集的匹配度。最后，它按匹配度排序并返回匹配的音阶列表。
* 实现流程： 将根音转换为 MIDI 音高类编号。 合并所有和弦的音符并去重，得到输入音符集的音高类集合。 对于每个音阶，将其转调到根音，并计算与输入音符集的匹配度。 按匹配度排序并返回匹配的音阶列表。
* 调用： items,sort,
* 内部依赖描述：

### initialize\_parameters (service/ChordHMMV2.py)

* 行号位置：68-80
* 重要性评分：4.30
* 功能描述： 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B（上下文依赖）和初始状态概率pi。
* 实现流程： 生成一个随机的状态转移矩阵A，其维度为n\_states x n\_states。 对A的每一行进行归一化，确保每一行的和为1。 生成一个随机的发射矩阵B，其维度为n\_states x n\_chords x n\_contexts。 对B的每一行进行归一化，确保每一行的和为1。 生成一个随机的初始状态概率向量pi，其维度为n\_states。 对pi进行归一化，确保其和为1。
* 引入包： time,pickle,utils.filePath,
* 调用： rand,
* 内部依赖描述：

### initialize\_parameters (service/ChordHMM.py)

* 行号位置：40-51
* 重要性评分：4.20
* 功能描述： 该函数用于随机初始化隐马尔可夫模型（HMM）的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。
* 实现流程： 生成一个随机的n\_states x n\_states的矩阵A，并对其进行行归一化，确保每行的和为1。 生成一个随机的n\_states x n\_chords的矩阵B，并对其进行行归一化，确保每行的和为1。 生成一个随机的n\_states维的向量pi，并对其进行归一化，确保其和为1。
* 引入包： pickle,utils.filePath,
* 调用： rand,
* 内部依赖描述：

### import\_data (service/ChordVectorFaiss.py)

* 行号位置：154-165
* 重要性评分：4.20
* 功能描述： 该函数用于从缓存文件中导入和弦数据，并将其添加到索引中，最后保存索引和映射。
* 实现流程： 打开缓存文件进行读取。 逐行读取文件内容，去除行首尾的空白字符。 跳过空行。 将每一行按’::’分割成和弦名称和按弦字符串列表。 将和弦名称和每个按弦字符串传递给add\_chord函数，将其添加到索引中，并转换为MIDI音符序列的One-Hot编码向量。 遍历完所有行后，调用save\_index函数，将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： strip,add\_chord,save\_index,
* 内部依赖描述：
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + save\_index: 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。

### data\_cut (service/ChordVectorAnnoy.py)

* 行号位置：92-103
* 重要性评分：4.20
* 功能描述： 该函数用于从缓存文件中读取和弦数据，并将和弦名称和按弦字符串添加到索引中，最终构建索引。
* 实现流程： 打开缓存文件进行读取。 逐行读取文件内容，去除行首尾的空白字符。 跳过空行。 将每一行按 ‘::’ 分割成和弦名称和按弦字符串列表。 遍历按弦字符串列表，将和弦名称和每个按弦字符串添加到索引中，并转换为 MIDI 音符序列的 One-Hot 编码向量。 调用 build\_index 方法构建索引。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： strip,add\_chord,build\_index,
* 内部依赖描述：
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + build\_index: 构建索引并保存

### **init** (service/numpyMarkov.py)

* 行号位置：6-16
* 重要性评分：4.10
* 功能描述： 该类用于初始化一个马尔科夫链预测器，用于预测和弦序列。它通过为和弦分配索引和构建转移概率矩阵来实现这一功能。
* 实现流程： 初始化预测器，设置马尔科夫链的阶数。 构建和弦索引字典，为每个和弦分配一个唯一的整数索引，并为未知和弦分配一个特殊的索引。 计算唯一和弦的数量。 初始化概率转移矩阵，大小为 (n\_chords^order, n\_chords)。 根据输入的和弦序列构建转移概率矩阵，通过计算每个状态到下一个状态的转移次数，并将其归一化为概率，从而得到一个表示和弦序列转移概率的矩阵。
* 调用： index\_chords,zeros,build\_transition\_matrix,
* 内部依赖描述：
  + index\_chords: 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
  + build\_transition\_matrix: 该函数根据输入的和弦序列构建概率转移矩阵，通过计算每个状态到下一个状态的转移次数，并将其归一化为概率，从而得到一个表示和弦序列转移概率的矩阵。

### \_on\_key\_release (service/MidiInput.py)

* 行号位置：97-107
* 重要性评分：4.10
* 功能描述： 该函数用于处理键盘释放事件，当焦点不在文本输入框时，根据按下的键字符，从预设的键映射中查找对应的音符，并调用内部函数处理音符释放事件。
* 实现流程： 获取当前焦点的控件。 检查焦点控件是否为文本输入框，如果是，则不触发音符释放事件。 尝试获取按下的键字符，并将其转换为大写。 检查键字符是否在预设的键映射中，如果在，则调用内部函数处理音符释放事件，从活动音符列表中移除该音符，并触发音符释放事件。 如果键字符不在键映射中，则忽略该事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： focusWidget,upper,\_handle\_note\_release,
* 内部依赖描述：
  + \_handle\_note\_release: 处理释放音符，当音符被释放时，从活动音符列表中移除该音符，并触发音符释放事件。

### \_process\_midi\_events (service/MidiInput.py)

* 行号位置：142-151
* 重要性评分：4.00
* 功能描述： 该函数用于处理MIDI事件，包括音符按下和释放。它会解析每个事件的状态、音符和力度，并根据状态调用相应的内部函数来处理音符的按下和释放。
* 实现流程： 遍历传入的MIDI事件列表。 解析每个事件的状态、音符和力度。 根据状态判断音符是按下还是释放。 如果音符按下且力度大于0，调用\_handle\_note\_press处理音符按下事件。 如果音符释放或力度为0，调用\_handle\_note\_release处理音符释放事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： info,\_handle\_note\_press,\_handle\_note\_release,
* 内部依赖描述：
  + \_handle\_note\_press: 处理按下音符，如果音符不在活动键集合中，则将其添加到集合中，并触发按键按下事件。
  + \_handle\_note\_release: 处理释放音符，当音符被释放时，从活动音符列表中移除该音符，并触发音符释放事件。

### save\_model (service/ChordHMMV2.py)

* 行号位置：315-333
* 重要性评分：3.90
* 功能描述： 该函数用于将模型参数保存到指定的文件路径。
* 实现流程： 定义一个包含模型所有参数的字典model\_data。 使用pickle模块将model\_data字典以二进制格式写入到指定的文件路径中。 打印保存成功的消息，显示保存的文件路径。
* 引入包： time,pickle,utils.filePath,
* 调用： filePath,dump,
* 内部依赖描述：

### backward (service/ChordHMMV2.py)

* 行号位置：122-139
* 重要性评分：3.80
* 功能描述： 后向算法用于计算贝塔值，该值在隐马尔可夫模型（HMM）中用于计算给定观察序列的概率。
* 实现流程： 确定观察序列O的长度T和状态数N。 初始化贝塔矩阵beta，其最后一行设置为全1，因为从最后一个时间步开始，贝塔值为1。 从倒数第二个时间步开始，逆序遍历到第一个时间步。 对于每个时间步t和状态i，计算贝塔值beta[t, i]，它等于从时间步t+1到T-1的所有状态j的贝塔值beta[t+1, j]乘以状态转移概率A[i, j]和发射概率B[j, O[t+1], context\_index]的乘积之和。 返回计算得到的贝塔矩阵beta。
* 引入包： time,pickle,utils.filePath,
* 调用： zeros,ones,
* 内部依赖描述：

### update\_chord (service/ChordVectorAnnoy.py)

* 行号位置：83-89
* 重要性评分：3.70
* 功能描述： 该函数用于更新指定和弦的 MIDI 音符序列。它首先删除指定和弦，然后添加新的和弦及其按弦字符串，并重新构建索引。
* 实现流程： 删除指定和弦 添加新的和弦及其按弦字符串 将新的和弦转换为 MIDI 音符序列的 One-Hot 编码向量 重新构建索引
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： delete\_chord,add\_chord,build\_index,
* 内部依赖描述：
  + delete\_chord: 该函数用于从索引中删除指定的和弦，并更新相关的映射和索引。
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + build\_index: 构建索引并保存

### update\_chord (service/ChordVectorFaiss.py)

* 行号位置：145-151
* 重要性评分：3.70
* 功能描述： 该函数用于更新指定和弦的 MIDI 音符序列。它首先删除指定和弦，然后添加新的和弦及其按弦字符串，最后保存更新后的索引和映射。
* 实现流程： 删除指定和弦 添加新的和弦及其按弦字符串 保存更新后的索引和映射
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： delete\_chord,add\_chord,save\_index,
* 内部依赖描述：
  + delete\_chord: 该函数用于从索引中删除指定的和弦，并更新相关的映射和索引。
  + add\_chord: 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
  + save\_index: 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。

### save\_model (service/ChordHMM.py)

* 行号位置：237-253
* 重要性评分：3.70
* 功能描述： 该函数用于将模型参数保存到指定的文件路径中。
* 实现流程： 定义一个包含模型参数的字典model\_data。 使用pickle模块将model\_data字典以二进制格式写入到指定的文件路径中。 打印保存成功的消息，显示保存的文件路径。
* 引入包： pickle,utils.filePath,
* 调用： filePath,dump,
* 内部依赖描述：

### \_emit\_key\_press (service/MidiInput.py)

* 行号位置：121-126
* 重要性评分：3.60
* 功能描述： 该函数用于触发按键按下事件，将MIDI音符转换为虚拟键值，并通过信号发出按键按下事件。同时，记录按键按下日志，并调用内部函数处理MIDI输入事件。
* 实现流程： 计算虚拟键值：将传入的MIDI音符减去36，得到虚拟键值。 触发按键按下事件：通过信号发出虚拟键值，通知其他部分按键已按下。 记录日志：使用logging.info记录按键按下事件的日志。 处理MIDI输入事件：调用内部函数on\_midi\_input，传入MIDI音符和127（表示音符按下），处理MIDI输入事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： emit,info,on\_midi\_input,
* 内部依赖描述：
  + on\_midi\_input: 该函数用于处理MIDI输入事件，根据音符是否按下（is\_note\_on）来决定是播放音符还是关闭音符。

### run (service/MidiInput.py)

* 行号位置：135-140
* 重要性评分：3.60
* 功能描述： 该函数用于持续监听MIDI输入事件，并在检测到事件时进行处理。
* 实现流程： 函数会进入一个无限循环，直到self.running为False。 在循环中，首先检查是否启用了键盘映射，如果没有启用并且MIDI输入有事件发生，则读取最多10个MIDI事件。 读取到的MIDI事件会被传递给私有方法self.\_process\_midi\_events进行处理。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： poll,read,\_process\_midi\_events,
* 内部依赖描述：
  + \_process\_midi\_events: 该函数用于处理MIDI事件，包括音符按下和释放。它会解析每个事件的状态、音符和力度，并根据状态调用相应的内部函数来处理音符的按下和释放。

### \_emit\_key\_release (service/MidiInput.py)

* 行号位置：128-133
* 重要性评分：3.60
* 功能描述： 该函数用于触发按键释放事件，处理MIDI音符的释放，并记录日志。
* 实现流程： 计算虚拟键值，通过将输入的音符减去36得到。 触发按键释放事件，通过调用v\_key\_released.emit(virtual\_key)。 记录日志，显示释放的MIDI音符。 调用on\_midi\_input函数，传入音符、0和False，表示音符释放。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： emit,info,on\_midi\_input,
* 内部依赖描述：
  + on\_midi\_input: 该函数用于处理MIDI输入事件，根据音符是否按下（is\_note\_on）来决定是播放音符还是关闭音符。

### search\_chord (service/ChordVectorAnnoy.py)

* 行号位置：55-70
* 重要性评分：3.60
* 功能描述： 该函数根据给定的 MIDI 音符序列搜索最相似的和弦，并返回这些和弦的名称、演奏方式以及与输入序列的距离。
* 实现流程： 将输入的 MIDI 音符序列转换为 One-Hot 编码的向量表示。 使用向量在索引中查找最相似的和弦ID及其距离。 根据找到的ID，从映射表中获取和弦的名称和演奏方式。 将结果以包含和弦名称、演奏方式和距离的字典形式返回。
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： midi\_notes\_to\_vector,get\_nns\_by\_vector,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### backward (service/ChordHMM.py)

* 行号位置：79-94
* 重要性评分：3.60
* 功能描述： 后向算法用于计算贝塔值矩阵，该矩阵在隐马尔可夫模型中用于计算给定观察序列的后向概率。
* 实现流程： 确定观察序列O的长度T和状态转移矩阵A、发射矩阵B的维度N。 初始化贝塔矩阵beta，其形状为(T, N)，并将最后一行设为全1，因为T时刻的后向概率为1。 从倒数第二个时间步开始，逆序遍历到第一个时间步。 对于每个时间步t和每个状态i，计算贝塔值beta[t, i]，它是所有可能的下一个状态j的贝塔值beta[t + 1, j]、状态转移概率A[i, j]和发射概率B[j, O[t + 1]]的乘积之和。 返回计算得到的贝塔矩阵beta。
* 引入包： pickle,utils.filePath,
* 调用： zeros,ones,
* 内部依赖描述：

### build\_index (service/ChordVectorAnnoy.py)

* 行号位置：49-53
* 重要性评分：3.50
* 功能描述： 构建索引并保存
* 实现流程： 调用self.index.build(n\_trees)构建索引，其中n\_trees是树的数量 调用self.index.save(self.index\_path)将构建好的索引保存到指定路径 使用pickle.dump将id\_mapping以二进制格式保存到self.mapping\_path
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： build,save,dump,
* 内部依赖描述：

### add\_chord (service/ChordVectorAnnoy.py)

* 行号位置：37-47
* 重要性评分：3.10
* 功能描述： 该函数用于将和弦名称和按弦字符串添加到索引中，并将其转换为 MIDI 音符序列的 One-Hot 编码向量。
* 实现流程： 解析按弦字符串，提取根音符和音符序列。 将音符序列转换为 MIDI 音符的整数列表。 将 MIDI 音符列表转换为 One-Hot 编码的向量表示。 将向量添加到索引中，并生成一个唯一的 ID。 将和弦名称、按弦字符串和 MIDI 音符列表与该 ID 关联存储。 递增下一个可用的 ID
* 引入包： os,pickle,annoy,utils.filePath,
* 调用： midi\_notes\_to\_vector,add\_item,
* 内部依赖描述：
  + midi\_notes\_to\_vector: 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。

### forward (service/ChordHMMV2.py)

* 行号位置：101-120
* 重要性评分：3.00
* 功能描述： 该函数实现了前向算法，用于计算给定观察序列下的状态序列概率。它通过动态规划的方法，利用状态转移矩阵A、发射矩阵B和初始状态概率pi，计算出每个时间步每个状态的前向概率alpha。
* 实现流程： 确定观察序列O的长度T和状态数N。 初始化Alpha矩阵alpha，其大小为(T, N)，并设置为零。 根据发射矩阵B和初始状态概率pi，计算时间步t=0时每个状态的前向概率alpha[0]。 从时间步t=1开始，遍历每个时间步，对于每个状态j，计算其前向概率alpha[t, j]，该概率由发射概率b和前一时间步所有状态的前向概率与状态转移概率的乘积之和决定。 返回计算得到的Alpha矩阵alpha。
* 引入包： time,pickle,utils.filePath,
* 调用： zeros,
* 内部依赖描述：

### \_initialize\_midi\_device (service/MidiInput.py)

* 行号位置：71-77
* 重要性评分：2.70
* 功能描述： 初始化MIDI设备，尝试通过pygame.midi.Input方法打开指定ID的MIDI输入设备，并设置use\_keyboard\_mapping为False。如果初始化失败，抛出ValueError异常。
* 实现流程： 尝试使用pygame.midi.Input方法初始化MIDI输入设备，传入device\_id作为参数。 如果初始化成功，将self.midi\_input设置为初始化后的MIDI输入设备对象，并将self.use\_keyboard\_mapping设置为False。 如果初始化过程中发生pygame.midi.MidiException异常，捕获该异常并抛出ValueError异常，异常信息包含无法初始化MIDI输入设备的具体原因。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： Input,ValueError,
* 内部依赖描述：

### save\_index (service/ChordVectorFaiss.py)

* 行号位置：87-93
* 重要性评分：2.70
* 功能描述： 该函数用于保存索引和映射。它将索引写入指定路径，并将映射对象序列化后保存到另一个指定路径。
* 实现流程： 使用faiss.write\_index方法将索引对象保存到self.index\_path指定的路径。 打开self.mapping\_path指定的路径，以二进制写模式打开文件。 使用pickle.dump方法将id\_mapping对象序列化并写入文件。 关闭文件以确保数据正确写入磁盘。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： write\_index,dump,
* 内部依赖描述：

### load\_model (service/ChordHMMV2.py)

* 行号位置：335-351
* 重要性评分：2.70
* 功能描述： 该函数用于从指定路径加载模型参数，包括状态转移矩阵A、发射概率矩阵B、初始状态概率向量pi、和一些其他相关数据。
* 实现流程： 打开指定路径下的模型文件。 使用pickle模块加载模型数据。 将加载的数据分别赋值给类的属性A、B、pi等。 打印加载成功的消息。
* 引入包： time,pickle,utils.filePath,
* 调用： filePath,
* 内部依赖描述：

### stop (service/MidiInput.py)

* 行号位置：153-159
* 重要性评分：2.70
* 功能描述： 停止MIDI输入，关闭MIDI输入设备并退出pygame环境。
* 实现流程： 将self.running设置为False，表示停止运行。 如果self.use\_keyboard\_mapping为False，则关闭MIDI输入设备并退出pygame。 退出pygame环境。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： quit,
* 内部依赖描述：

### forward (service/ChordHMM.py)

* 行号位置：61-77
* 重要性评分：2.70
* 功能描述： 前向算法用于计算HMM模型在给定观察序列下的状态概率序列，即Alpha矩阵。
* 实现流程： 初始化Alpha矩阵，长度为观察序列长度T，状态数为N。 设置Alpha矩阵的第一个时间步t=0，根据初始状态概率pi和发射矩阵B计算。 从第二个时间步t=1开始，遍历每个时间步，计算每个状态j在当前时间步的概率，该概率由发射矩阵B和前一个时间步的状态概率以及状态转移矩阵A共同决定。 最终返回Alpha矩阵，其中包含了每个时间步每个状态的概率。
* 引入包： pickle,utils.filePath,
* 调用： zeros,
* 内部依赖描述：

### on\_midi\_input (service/soundNoise.py)

* 行号位置：45-49
* 重要性评分：2.50
* 功能描述： 该函数用于处理MIDI输入事件，根据音符是否按下（is\_note\_on）来决定是播放音符还是关闭音符。
* 实现流程： 检查is\_note\_on参数，判断音符是否按下。 如果is\_note\_on为True，则调用play\_note\_on函数，播放指定音符并设置力度。 如果is\_note\_on为False，则调用play\_note\_off函数，关闭指定音符。
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： play\_note\_on,play\_note\_off,
* 内部依赖描述：
  + play\_note\_on: 该函数用于在音乐播放器上播放指定音符，支持设置音符的力度。
  + play\_note\_off: 关闭指定音符

### \_enable\_keyboard\_mapping (service/MidiInput.py)

* 行号位置：60-64
* 重要性评分：2.50
* 功能描述： 启用键盘映射模式，记录键盘输入并进行相应的映射处理。
* 实现流程： 记录日志，提示使用键盘映射模式。 设置内部标志use\_keyboard\_mapping为True，表示启用键盘映射模式。 启动键盘监听器，开始监听键盘输入事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： info,\_start\_keyboard\_listener,
* 内部依赖描述：
  + \_start\_keyboard\_listener: 该函数用于启动一个键盘事件监听器，监听键盘按键的按下和释放事件，并在事件发生时调用相应的处理函数。

### \_print\_midi\_devices (service/MidiInput.py)

* 行号位置：49-53
* 重要性评分：2.50
* 功能描述： 该函数用于打印所有可用的MIDI设备的信息。
* 实现流程： 使用pygame.midi.get\_count()获取可用的MIDI设备数量。 遍历每个设备，使用pygame.midi.get\_device\_info(i)获取设备信息。 打印每个设备的索引和信息。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： get\_count,get\_device\_info,
* 内部依赖描述：

### load\_model (service/ChordHMM.py)

* 行号位置：255-269
* 重要性评分：2.50
* 功能描述： 该函数用于从指定路径加载模型参数，并将这些参数赋值给类的实例变量。
* 实现流程： 接收一个模型文件的路径作为参数。 使用pickle模块以二进制读取模式打开指定路径下的模型文件。 从文件中加载模型参数，并将其存储在变量model\_data中。 将model\_data中的各个参数（A, B, pi, chord\_index, n\_states, n\_chords, index\_chord）赋值给类的实例变量。 打印一条消息，指示模型已成功加载。
* 引入包： pickle,utils.filePath,
* 调用： filePath,
* 内部依赖描述：

### state\_to\_index (service/numpyMarkov.py)

* 行号位置：57-71
* 重要性评分：2.50
* 功能描述： 将和弦状态转换为对应的索引，用于状态表示和状态转移的计算。
* 实现流程： 初始化索引为0。 遍历当前和弦状态中的每个和弦。 对于每个和弦，获取其对应的索引，如果和弦未知则使用默认的未知和弦索引。 确保和弦索引在有效范围内。 根据和弦的顺序和位置，计算其在状态索引中的权重，并累加到总索引中。 确保最终索引在有效范围内。 返回计算得到的状态索引。
* 调用： get,
* 内部依赖描述：

### \_handle\_note\_press (service/MidiInput.py)

* 行号位置：109-113
* 重要性评分：2.50
* 功能描述： 处理按下音符，如果音符不在活动键集合中，则将其添加到集合中，并触发按键按下事件。
* 实现流程： 检查音符是否在活动键集合中。 如果音符不在集合中，则将其添加到活动键集合中。 触发按键按下事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： add,\_emit\_key\_press,
* 内部依赖描述：
  + \_emit\_key\_press: 该函数用于触发按键按下事件，将MIDI音符转换为虚拟键值，并通过信号发出按键按下事件。同时，记录按键按下日志，并调用内部函数处理MIDI输入事件。

### \_handle\_note\_release (service/MidiInput.py)

* 行号位置：115-119
* 重要性评分：2.50
* 功能描述： 处理释放音符，当音符被释放时，从活动音符列表中移除该音符，并触发音符释放事件。
* 实现流程： 检查音符是否在活动音符列表中。 如果音符在列表中，将其从活动音符列表中移除。 触发音符释放事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： remove,\_emit\_key\_release,
* 内部依赖描述：
  + \_emit\_key\_release: 该函数用于触发按键释放事件，处理MIDI音符的释放，并记录日志。

### build\_contexts (service/ChordHMMV2.py)

* 行号位置：53-66
* 重要性评分：2.40
* 功能描述： 该函数用于构建上下文，通过将情绪标签列表和风格标签组合成元组，并将这些元组添加到集合中，最终返回一个包含所有唯一上下文的列表。
* 实现流程： 接收情绪标签列表的列表和风格标签列表作为输入参数。 遍历情绪标签列表的列表和风格标签列表，使用zip函数将对应位置的元素配对。 将每个情绪标签列表转换为排序后的元组，以便进行哈希操作。 将情绪标签元组和风格标签组合成一个新的元组，作为上下文。 将每个上下文添加到集合中，确保上下文的唯一性。 将集合转换为列表，返回包含所有唯一上下文的列表。
* 引入包： time,pickle,utils.filePath,
* 调用： add,
* 内部依赖描述：

### \_start\_keyboard\_listener (service/MidiInput.py)

* 行号位置：66-69
* 重要性评分：2.40
* 功能描述： 该函数用于启动一个键盘事件监听器，监听键盘按键的按下和释放事件，并在事件发生时调用相应的处理函数。
* 实现流程： 创建一个键盘事件监听器对象，监听按键按下和释放事件。 设置按键按下事件的处理函数为\_on\_key\_press。 设置按键释放事件的处理函数为\_on\_key\_release。 启动监听器，开始监听键盘事件。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： Listener,start,
* 内部依赖描述：

### **init** (service/markov.py)

* 行号位置：7-29
* 重要性评分：2.30
* 功能描述： 该函数用于初始化一个马尔科夫链模型，用于生成和弦序列。它接受一个和弦风格、一个和弦序列列表以及一个可选的阶数（默认为2）。函数首先将所有和弦汇总到一个列表中，然后构建一个马尔科夫链，其中每个状态（由一定数量的和弦组成）对应可能的下一个和弦。最后，函数记住最后一组状态和对应的下一个状态。
* 实现流程： 将所有和弦汇总到一个列表中。 构建马尔科夫链，其中每个状态对应可能的下一个和弦。 记住最后一组状态和对应的下一个状态。 打印马尔可夫链的初始化结果。
* 引入包： random,

### index\_chords (service/numpyMarkov.py)

* 行号位置：18-29
* 重要性评分：2.20
* 功能描述： 该函数用于为和弦序列中的每个和弦分配一个唯一的索引，同时为未知和弦分配一个特殊的索引。
* 实现流程： 初始化一个空集合unique\_chords，用于存储唯一的和弦。 遍历输入的和弦序列sequences，将每个和弦添加到unique\_chords集合中，确保所有和弦都是唯一的。 计算unique\_chords集合的大小，并将其赋值给self.unknown\_chord\_index，作为未知和弦的索引。 使用enumerate函数遍历unique\_chords集合，生成一个字典，将每个和弦映射到一个唯一的索引，并返回该字典。
* 调用： update,
* 内部依赖描述：

### index\_chords (service/ChordHMMV2.py)

* 行号位置：42-51
* 重要性评分：2.00
* 功能描述： 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
* 实现流程： 接收一个包含和弦序列的列表作为输入。 遍历每个和弦序列，将其中的和弦添加到一个集合中，以确保每个和弦都是唯一的。 使用enumerate函数为集合中的每个和弦分配一个唯一的索引，索引从0开始。 返回一个字典，其中键是和弦，值是对应的索引。
* 引入包： time,pickle,utils.filePath,
* 调用： update,
* 内部依赖描述：

### context\_to\_index (service/ChordHMMV2.py)

* 行号位置：90-99
* 重要性评分：2.00
* 功能描述： 将情绪标签和风格标签转换为上下文索引。
* 实现流程： 将情绪标签列表进行排序并转换为元组。 将排序后的情绪元组和风格标签组合成一个上下文元组。 使用上下文元组在self.context\_index字典中查找对应的索引值。 如果找到对应的索引值，则返回该索引值；否则返回-1。
* 引入包： time,pickle,utils.filePath,
* 调用： get,
* 内部依赖描述：

### index\_chords (service/ChordHMM.py)

* 行号位置：29-38
* 重要性评分：2.00
* 功能描述： 该函数用于为和弦序列中的每个唯一和弦分配一个唯一的索引。
* 实现流程： 首先，创建一个空集合unique\_chords来存储唯一的和弦。 遍历输入的和弦序列列表sequences，将每个序列中的和弦添加到unique\_chords集合中，确保每个和弦只出现一次。 使用enumerate函数遍历unique\_chords集合，为每个和弦分配一个从0开始的唯一索引。 返回一个字典，其中键是和弦，值是对应的索引。
* 引入包： pickle,utils.filePath,
* 调用： update,
* 内部依赖描述：

### midi\_notes\_to\_vector (service/ChordVectorFaiss.py)

* 行号位置：32-40
* 重要性评分：1.90
* 功能描述： 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。
* 实现流程： 初始化一个长度为 vector\_dim 的全零向量，数据类型为 float32。 遍历输入的 MIDI 音符序列。 对于每个音符，如果音符值在 0 到 vector\_dim 之间（包括 0 和 vector\_dim），则将向量中对应位置的值设为 1.0。 返回填充好的 One-Hot 编码向量。
* 引入包： os,pickle,faiss,utils.filePath,
* 调用： zeros,
* 内部依赖描述：

### chords\_to\_indices (service/ChordHMMV2.py)

* 行号位置：82-88
* 重要性评分：1.70
* 功能描述： 将和弦序列转换为对应的索引序列。
* 实现流程： 接收一个和弦列表作为输入。 遍历和弦列表中的每个和弦。 使用字典self.chord\_index查找每个和弦对应的索引。如果和弦不在字典中，则返回-1。 将查找得到的索引添加到结果列表中。 返回包含所有和弦索引的结果列表。
* 引入包： time,pickle,utils.filePath,
* 调用： get,
* 内部依赖描述：

### transition\_matrix\_to\_string (service/numpyMarkov.py)

* 行号位置：110-116
* 重要性评分：1.70
* 功能描述： 将概率转移矩阵转换为字符串形式，便于查看和记录。
* 实现流程： 使用np.array2string方法将概率转移矩阵转换为字符串。 设置precision参数为2，确保输出的小数点后有两位。 使用separator参数为’, ’，设置元素之间的分隔符为逗号和空格。 返回转换后的字符串形式的转移矩阵。
* 调用： array2string,
* 内部依赖描述：

### chords\_to\_indices (service/ChordHMM.py)

* 行号位置：53-59
* 重要性评分：1.70
* 功能描述： 将和弦序列转换为对应的索引序列。
* 实现流程： 接收一个和弦列表作为输入。 遍历和弦列表中的每个和弦。 使用self.chord\_index字典查找每个和弦对应的索引。如果和弦不在字典中，则返回-1。 将查找得到的索引添加到结果列表中。 返回包含所有和弦索引的结果列表。
* 引入包： pickle,utils.filePath,
* 调用： get,
* 内部依赖描述：

### \_choose\_device (service/MidiInput.py)

* 行号位置：55-58
* 重要性评分：1.40
* 功能描述： 该函数用于手动选择MIDI设备。用户可以通过输入设备ID来指定设备，如果用户按下回车键，则跳过设备选择，使用默认的键盘映射。
* 实现流程： 提示用户手动指定一个设备ID，或者按下回车键跳过。 读取用户输入。 如果用户输入了设备ID，则将其转换为整数并返回；如果用户按下回车键，则返回None。
* 引入包： pygame,pygame.midi,PyQt5.QtCore,PyQt5.QtWidgets,pynput.keyboard,service.soundNoise,pynput,logging,
* 调用： strip,
* 内部依赖描述：

### play\_note\_off (service/soundNoise.py)

* 行号位置：40-42
* 重要性评分：1.30
* 功能描述： 关闭指定音符
* 实现流程： 接收音符参数 调用fs对象的noteoff方法，关闭指定音符
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： noteoff,
* 内部依赖描述：

### play\_note\_on (service/soundNoise.py)

* 行号位置：36-38
* 重要性评分：1.30
* 功能描述： 该函数用于在音乐播放器上播放指定音符，支持设置音符的力度。
* 实现流程： 接收音符和力度参数。 调用音乐播放器的noteon方法，传入通道0、音符和力度参数，触发音符播放。
* 引入包： platform,fluidsynth,utils.filePath,logging,
* 调用： noteon,
* 内部依赖描述：

### print\_model\_parameters (service/ChordHMMV2.py)

* 行号位置：304-313
* 重要性评分：1.00
* 功能描述： 该函数用于打印隐马尔可夫模型（HMM）的参数，包括状态转移矩阵（A）、发射矩阵（B）和初始状态概率（pi）。
* 实现流程： 打印状态转移矩阵（A） 打印发射矩阵（B） 打印初始状态概率（pi）
* 引入包： time,pickle,utils.filePath,

### print\_model\_parameters (service/ChordHMM.py)

* 行号位置：226-235
* 重要性评分：1.00
* 功能描述： 该函数用于打印隐马尔可夫模型（HMM）的参数，包括状态转移矩阵（A）、发射矩阵（B）和初始状态概率（pi）。
* 实现流程： 打印状态转移矩阵（A） 打印发射矩阵（B） 打印初始状态概率（pi）
* 引入包： pickle,utils.filePath,

### midi\_notes\_to\_vector (service/ChordVectorAnnoy.py)

* 行号位置：27-35
* 重要性评分：0.90
* 功能描述： 将 MIDI 音符序列转换为 One-Hot 编码的向量表示。
* 实现流程： 初始化一个长度为 vector\_dim 的全零向量。 遍历输入的 MIDI 音符序列。 对于每个音符，如果其值在 0 到 vector\_dim 之间（包括 0 和 vector\_dim），则将该音符对应的向量位置设为 1。 返回最终的 One-Hot 编码向量。
* 引入包： os,pickle,annoy,utils.filePath,

### **init** (service/MatchingScales.py)

* 行号位置：36-39
* 重要性评分：0.40
* 功能描述： 初始化一个音乐分析对象，设置根音符、输入音符列表和阈值。
* 实现流程： 接收根音符、输入音符列表和阈值作为参数。 将根音符、输入音符列表和阈值分别赋值给对象的root\_note、input\_notes和threshold属性。

### ChordHMM (service.py)

### ChordHMMV2 (service2.py)

### ChordVectorAnnoy (service.py)

### ChordVectorFaiss (service.py)

### MatchingScales (service.py)

### MidiInput (service.py)

### **init** (service\_\_init\_\_.py)

### markov (service.py)

### numpyMarkov (service.py)

### soundNoise (service.py)