提案ドキュメント ー“言葉と歌の文芸プロジェクト” 次世代 ChordMap設計と演奏制御スクリプト

はじめに

本ドキュメントでは、以下のコンセプトを基に、ChordMapチームおよびスクリプトを統合・ブラッシュアップした最終提案を示します。

- **音楽×文学の融合**：歌詞の感情・構成美を最優先し、和声・リズム・演奏表現を設計
- **MUSIC21基盤**：音階・コード候補生成にmusic21を活用
- **感情ドリブン演奏制御**：emotion＋durationによるHumanizerロジックをスクリプト内に組み込み

主要要素

1. **ChordMap Emotions YAML** (既存提供)
   各セクションのメタ情報 (order, length_in_measures, tonic, mode, musical_intent) と、コード進行 (label, duration_beats, emotion, humanize) がすべて記述済。
2. **スクリプト群**
   - **generate_chordmap_section** (pdf①)
   - **EmotionHumanizer**拡張 (pdf②,スニペット)
   - **ChordMap自動生成スクリプト** (pdf③)
   - **音域フィルタリング / Voicing最適化** (pdf④)

これらを一貫して動作させるためのワークフローとコード設計を以下にまとめます。

1. ChordMap Emotions YAML構造の最終仕様

既に提供済みの chordmap_emotions.yaml を基礎としつつ、以下のフィールド構造を確定します。
(global_settingsおよびproject_titleは省略可)

1.1. セクション定義

```
sections:
  <SectionName>:
    order: <int>                  # セクションの登場順
    length_in_measures: <float>    # 小節数
    tonic: <str>                  # 主音 (例: D)
    mode: <str>                   # モード (例: dorian)
    musical_intent:
      emotion: <str>              # 感情タグ (例: quiet_pain_and_nascent_strength)
      intensity: <str>            # 感情強度 (low/medium/high)
    part_settings:                # 任意 (パートごとの細かいヒント)
      <instrument>_style_keyword: <str>
      <instrument>_velocity:       <int>
      <instrument>_apply_pedal:    <bool>
    part_specific_hints:          # 任意 (歌詞終端の小節位置など)
      lyrics_end_actual_beats_from_section_start: <float>
    chord_progression:
      - label: <str>              # コード名 (例: Dm7)
        duration_beats: <float>      # 小節内の継続拍数 (例: 4.0)
        emotion: <str>              # 同セクションの emotion と一致可
```

```
    nuance: <str> (任意)          # 追加テンションや装飾の説明
    humanize:
      actual_duration: <float>   # release時まで含めた実音長（beats）
      onset_shift_ms: <float>    # 人間らしい開始タイミングシフト
      articulation: <str>        # legato/staccato/tenuto/accented
      velocity_bias: <int>       # -10〜+10 のMIDIベロシティ偏差
    voicing:                     # 任意（voice-leading後の具体的構成音）
      - <str> (例: D3)
      - <str> (例: A3)
      - ...
    midi_program: <int> (任意)    # DAW統合用MIDI音色番号
    channel: <int> (任意)
    adjusted_start_beat: <float>  # 小節端でないコードの開始位置（beats）
```

- 各フィールドは**必須 or 任意**を明示し、**Pydanticモデル**でバリデーション可能にする。
- 各パート（guitar/bass/drums/pianoなど）の part_settings は簡易化し、必要に応じて専用loaderで扱う。

2. スクリプト設計 ― フル自動化エンジンの流れ

以下では、トップレベルから順に主要関数・尤度・I/Oフローを説明します。

2.1. 全体ワークフロー図

```
入力:
  chordmap_emotions.yaml  ←– 1: 人手で編集/レビュー済み
  (必要時) emotion_templates.json

  ┌──────────────────────────────────────────────┐
| ① load_chordmap_emotions |  ═══> Pydantic で構造検証
  └──────────────────────────────────────────────┘
            | sections 構造体 (dict)
            ▼
  ┌──────────────────────────────────────────────────────────────┐
| ② for each section in sections: |
|   generate_section_data()       |
  └──────────────────────────────────────────────────────────────┘
            |
            ├── 2-a: get_scale_pitches(tonic, mode)  (music21)
            ├── 2-b: get_tensions_for_emotion(emotion) (internal map)
            ├── 2-c: apply_tensions_to_chord(label, tensions) (chord_voicer)
            ├── 2-d: filter_voicing_pitches(notes, max_notes=5)
            ├── 2-e: apply_emotional_humanization(humanize_params)
(humanizer)
            └── 2-f: assemble chord_event dict

            ▼ 完成した `section_data`
  ┌──────────────────────────────────────────────────────────────────────────────┐
| ③ 全 sections をまとめた出力 chordmap_humanized.yaml |
```

```
└──────────────────────────────────────────────────────────────────────
──────────┘

(オプション)
└── ④ `modular_composer.py` で MIDI や DAW データを生成
```

2.2. ① load_chordmap_emotions

```
from pathlib import Path
from pydantic import BaseModel, ValidationError
import yaml

class ChordEvent(BaseModel):
    label: str
    duration_beats: float
    emotion: str
    nuance: str | None = None
    humanize: dict
    voicing: list[str] | None = None
    midi_program: int | None = None
    channel: int | None = None
    adjusted_start_beat: float | None = None

class SectionMeta(BaseModel):
    order: int
    length_in_measures: float
    tonic: str
    mode: str
    musical_intent: dict  # emotion, intensity を含む
    part_settings: dict | None
    part_specific_hints: dict | None
    chord_progression: list[ChordEvent]

class ChordMapEmotions(BaseModel):
    sections: dict[str, SectionMeta]



def load_chordmap_emotions(path: str) -> ChordMapEmotions:
    with open(path, 'r', encoding='utf-8') as f:
        data = yaml.safe_load(f)
    try:
        return ChordMapEmotions(**data)
    except ValidationError as e:
        raise RuntimeError(f"ChordMapEmotions ValidationError: {e}")
```

- **目的**：YAML から内部 Pydantic モデルへ変換し、フォーマットエラーを即検出する。
- **出力**：ChordMapEmotions オブジェクト → 各 sections は SectionMeta としてアクセス可能。

2.3. ② generate_section_data

各セクションごとに以下の手順を実行し、最終的な chord_event 情報を完成させます。

2-a: get_scale_pitches(tonic, mode)

```
from music21 import scale, pitch

def get_scale_pitches(tonic: str, mode_name: str) -> list[str]:
    mode_map = {
        'ionian': scale.MajorScale,
        'aeolian': scale.MinorScale,
        'dorian': scale.DorianScale,
        'phrygian': scale.PhrygianScale,
        'lydian': scale.LydianScale,
        'mixolydian': scale.MixolydianScale,
        'locrian': scale.LocrianScale
    }
    scale_class = mode_map.get(mode_name.lower())
    if not scale_class:
        raise ValueError(f"Unsupported mode: {mode_name}")
    s = scale_class(tonic)
    return [str(p) for p in s.getPitches(f'{tonic}3', f'{tonic}5')]
```

- **目的**：指定モードと主音に基づくスケール音（octave付き）を取得。

2-b: get_tensions_for_emotion(emotion)

```
# 予め emotion に対応するテンション候補を定義した dict をロード
EMOTION_TENSION_MAP = {
    'quiet_pain_and_nascent_strength': ['m7', 'add9'],
    'deep_regret_gratitude_and_realization': ['m7', 'add11', 'M7'],
    'acceptance_of_love_and_pain_hopeful_belief': ['add9', 'M7', '13'],
    # ... 残りセクションも同様にマッピング
}

def get_tensions_for_emotion(emotion: str) -> list[str]:
    return EMOTION_TENSION_MAP.get(emotion, [])
```

- **目的**：歌詞感情タグから、そのセクションで用いるべきコードテンションのリストを返却。

2-c: apply_tensions_to_chord(label, tensions)

```
from chord_voicer import apply_tensions  # 内部で music21 チャンク

def apply_tensions_to_chord(base_chord: str, scale: str, tonic: str,
desired_tensions: list[str]) -> dict:
    # apply_tensions は { 'notes': [...], 'label': 'Dm7add9', ... } を返す想定
    return apply_tensions(
        base_chord=base_chord,
        scale=scale,
        tonic=tonic,
```

```
        desired_tensions=desired_tensions
    )
```

- **目的**：generate_voicing_for_chord 等ではなく、指定テンションをコードに加える。
- **出力**：Voiced構成音情報を含む辞書 (notes, label, その他)。

2-d: filter_voicing_pitches(notes, max_notes=5)

```
from music21 import pitch as m21_pitch

def filter_voicing_pitches(pitches: list[str], max_notes: int = 5) -> list[str]:
    p_objs = [m21_pitch.Pitch(p) for p in pitches]
    # 音域 C3–C6 に収める
    for p in p_objs:
        while p.midi < 48:  p.octave += 1
        while p.midi > 84:  p.octave -= 1
    # 重複削除して先頭から max_notes 件を返却
    names = list(dict.fromkeys(p.nameWithOctave for p in p_objs))
    return names[:max_notes]
```

- **目的**：DAWやMIDIでディソナンスを避け、最大音数を制限。

2-e: apply_emotional_humanization(humanize_params)

```
import random

# Pydanticモデル例 (humanizer.py から引用)
from pydantic import BaseModel

class EmotionBehavior(BaseModel):
    onset_shift_ms: float
    articulation: str       # legato, staccato, tenuto, accented
    velocity_bias: int      # -10〜+10
    release_ratio: float    # 0.5〜1.2 (duration × release_ratio が実音長)

# 予め emotion → EmotionBehavior を定義
EMOTION_EXPRESSIONS = {
    'quiet_pain_and_nascent_strength': EmotionBehavior(onset_shift_ms=+12,
articulation='legato', velocity_bias=+6, release_ratio=0.95),
    'deep_regret_gratitude_and_realization':
EmotionBehavior(onset_shift_ms=-10, articulation='staccato',
velocity_bias=+3, release_ratio=0.60),
    'acceptance_of_love_and_pain_hopeful_belief':
EmotionBehavior(onset_shift_ms=-10, articulation='staccato',
velocity_bias=+3, release_ratio=0.60),
    # ... 以降各 emotion ごとに定義
}

def apply_emotional_humanization(base_duration: float, emotion: str) -> dict:
    behavior = EMOTION_EXPRESSIONS.get(emotion)
    if not behavior:
        return {
```

```
            'actual_duration': base_duration,
            'onset_shift_ms': 0.0,
            'articulation': 'normal',
            'velocity_bias': 0
        }
    actual_dur = base_duration * behavior.release_ratio
    onset_shift = random.uniform(-behavior.onset_shift_ms,
behavior.onset_shift_ms)
    return {
        'actual_duration': actual_dur,
        'onset_shift_ms': onset_shift,
        'articulation': behavior.articulation,
        'velocity_bias': behavior.velocity_bias
    }
```

- **目的**：duration_beats を release_ratio に従い変化させ、タイミングやアーティキュレーションを付加。
- **出力**：humanize 辞書。

2-f: assemble chord_event dict

```
# すでに base_data = { 'label': <str>, 'duration_beats': <float>, 'emotion': <str>,
'nuance': <str> } があるとする
voiced_info = apply_tensions_to_chord(base_data['label'], mode, tonic,
tensions)
filtered_notes = filter_voicing_pitches(voiced_info.get('notes', []))
hparams = apply_emotional_humanization(base_data['duration_beats'],
base_data['emotion'])

chord_event = {
    'label': voiced_info.get('label', base_data['label']),
    'duration_beats': base_data['duration_beats'],
    'emotion': base_data['emotion'],
    'nuance': base_data.get('nuance'),
    'humanize': hparams,
    'voicing': filtered_notes,
    'midi_program': None,  # 必要ならテンプレートから設定
    'channel': None,
    'adjusted_start_beat': base_data.get('adjusted_start_beat')
}
```

- **目的**：Voicing・Humanize情報を組み込んだ最終的な chord_event を生成。

2.4. ③ 全Sectionsまとめ → YAML出力

```
import yaml

def save_chordmap_yaml(chordmap: dict, out_path: str):
    with open(out_path, 'w', encoding='utf-8') as f:
        yaml.dump(chordmap, f, sort_keys=False, allow_unicode=True)
```

```
def process_chordmap_with_full_emotion(input_path: str, output_path: str):
    # 1) chordmap_emotions.yaml を読み込む
    cme = load_chordmap_emotions(input_path)

    output_map = {'sections': {}}
    for sec_name, sec_meta in cme.sections.items():
        section_output = {
            'order': sec_meta.order,
            'length_in_measures': sec_meta.length_in_measures,
            'tonic': sec_meta.tonic,
            'mode': sec_meta.mode,
            'musical_intent': sec_meta.musical_intent,
            'part_settings': sec_meta.part_settings,
            'part_specific_hints': sec_meta.part_specific_hints,
            'chord_progression': []
        }
        # get_scale_pitches
        scale_notes = get_scale_pitches(sec_meta.tonic, sec_meta.mode)
        # get tensions
        tensions = get_tensions_for_emotion(sec_meta.musical_intent['emotion'])

        for chord_entry in sec_meta.chord_progression:
            # base_data として Pydantic モデルから dict
            base_data = chord_entry.dict()
            voiced = apply_tensions_to_chord(
                base_data['label'], sec_meta.mode, sec_meta.tonic, tensions
            )
            filtered_notes = filter_voicing_pitches(voiced.get('notes', []))
            hparams = apply_emotional_humanization(
                base_data['duration_beats'], base_data['emotion']
            )
            event_out = {
                'label': voiced.get('label', base_data['label']),
                'duration_beats': base_data['duration_beats'],
                'emotion': base_data['emotion'],
                'nuance': base_data.get('nuance'),
                'humanize': hparams,
                'voicing': filtered_notes,
                'midi_program': base_data.get('midi_program'),
                'channel': base_data.get('channel'),
                'adjusted_start_beat': base_data.get('adjusted_start_beat')
            }
            section_output['chord_progression'].append(event_out)
```

```
        output_map['sections'][sec_name] = section_output

    save_chordmap_yaml(output_map, output_path)
```

- **呼び出し例**：
  python process_chordmap.py chordmap_emotions.yaml chordmap_emotions_humanized.yaml
-
- 出力されたYAMLは、**すべてのセクションで Humanize・Voicing が自動補完された最終版**となる。

3. スクリプト統合後の期待効果・活用方法

1. **一貫性の担保**
   - コード ⇒ テンション ⇒ Humanize ⇒ Voicing を一つの流れで自動生成。
   - YAML定義を手で修正 → 再実行し、瞬時に反映したMIDI/DAWファイルを出力可能。
2. **感情ドリブンの演奏表現**
   - “duration_beats” を人間の感情に合わせて自動的に変化させ、単にランダムではない**感情ベースの揺らぎ**を実現。
   - “articulation” や “onset_shift_ms” も emotion に左右されるため、体温を感じる演奏になる。
3. **コードテンションと文学的情趣の融合**
   - 小説家／詩人としてのHarusanのセンスを、ChodMap YAML で忠実にコントロール可能。
   - “D Dorian の m7add9” というような刺激的な和声進行が、歌詞のドラマとぴったり同期。

活用シーン例

- **YouTube AudioBook**：朗読に重ねる演奏BGMをMIDIから生成し、情感を増幅。
- **楽譜・各パート譜自動生成**：MIDIから楽譜を吐き出し、そのまま演奏者に共有。
- **DAW連携**：plugins用MIDIトラックとして、Cubase/Logic/Studio Oneで再生。
- **新曲制作のプロトタイプ**：コード進行と感情設計をYAMLでブラッシュアップしながら “即興的” にアイデアを試せる。

4. 今後の拡張アイデア／検討事項

1. **動的EmotionIntensity補正**
   - 歌詞のキーワード出現頻度をNLPで解析し、自動で intensity をスケール（0-1）に連動
   - 例：サビの盛り上がりで intensity を高くするロジック ⇒ “Chorus 1” で動的に表現を変える
2. **YAML/TOML編集UI**
   - Webベースの簡易エディタを作成し、プレビュー付きでChordMapを可視化・編集可能にする
   - 共同作業時のダブルエディット競合を防ぐ

3. **AIコード候補生成連携**
    - GPTや小規模AIモデルを呼び出し、詩の内容から自動で雰囲気に合うコード候補を提案
    - “Verse 1” に対して「D Dorian → Dm7add9 → Am7add9 → Gm7add11」など複数案を生成
4. **スウィング／リズムジェネレーター細分化**
    - pattern_type をより細かく分け、Jazz Swing・ラテン系・ブギウギなど多彩なスタイルを追加
    - 各パラメータをYAMLで微調整しながら、最適なリズムパターンを選定
5. **完全DAWバウンス自動化**
    - render_to_wav() 関数を加え、内部でVST音源 (e.g. Pianoteq, BFD3) を叩いてWAV化まで実行
    - スクリプト実行だけで最終マスタリング用オーディオファイルを生成

5. まとめ

- **ChordMap Emotions YAML** に感情・テンション・Humanize情報が一元化されることで、楽曲構成と演奏表現が強固に連携します。
- **スクリプト全体** はPydanticとmusic21を核に設計され、構造的な安全性と拡張性を担保。
- **今後の拡張** も見据えつつ、まずは現状のYAMLをベースにフル自動化し、Harusanの小説的感性と音楽の結合を具現化しましょう。

添付：最終スクリプト一覧

1. chordmap_emotions.yaml
2. process_chordmap.py
3. humanizer.py (EmotionExpressionProfile含む拡張)
4. chord_voicer.py (apply_tensions & filter_voicing含む)
5. generate_chordmap_section.py (必要に応じて)

上記を適切なプロジェクト構造で配置し、実行環境 (Python3.8+、music21, pydantic, PyYAML) を整えれば、最初のフルオートChordMap生成が完成します。
ぜひお試しいただき、ご意見や要望をお寄せください。