

このスクリプトは generate\_chordmap\_section(section\_name, tonic, mode, length) を使って、詩のセクションに合わせたコードマップ（初期案）を生成します。

- get\_scale\_pitches() は mode に応じたスケールピッチを展開。
- 7音構成のスケールに基づいて triad を生成。
- 初期の chordmap には、duration・emotion フィールドを含み、後から apply\_emotional\_humanization() で調整できます。

```
from music21 import key, scale, chord, note
from typing import List, Dict
```

```
def get_scale_pitches(tonic: str, mode: str) -> List[str]:
    """
    指定されたトニックとモードに基づいて、スケール上の音名リストを返します。
    """
    mode_class = {
        'major': scale.MajorScale,
        'minor': scale.MinorScale,
        'dorian': scale.DorianScale,
        'phrygian': scale.PhrygianScale,
        'lydian': scale.LydianScale,
        'mixolydian': scale.MixolydianScale,
        'aeolian': scale.AeolianScale,
        'locrian': scale.LocrianScale,
    }.get(mode.lower())

    if not mode_class:
        raise ValueError(f"Unsupported mode: {mode}")

    sc = mode_class(tonic)
    return [p.nameWithOctave for p in sc.getPitches(tonic + '3', tonic + '5')]


def generate_chordmap_section(section_name: str, tonic: str, mode: str,
length: int) -> Dict:
    """
    与えられたセクション情報から基本的なコードマップ構造を生成。
    """
    scale_pitches = get_scale_pitches(tonic, mode)
    triads = [
        chord.Chord([scale_pitches[i], scale_pitches[(i+2)%7],
scale_pitches[(i+4)%7]])
        for i in range(0, 7)]
```

```
]
```

```
chords = []
for i in range(length):
    ch = triads[i % len(triads)]
    label = ch.root().name + ("m" if ch.quality == "minor" else "")
    chords.append({
        "chord": label,
        "duration": 4, # デフォルト : 1小節分
        "emotion": "unspecified"
    })

return {
    "section": section_name,
    "tonic": tonic,
    "mode": mode,
    "chords": chords
}
```

# テスト例 :

```
if __name__ == "__main__":
    section = generate_chordmap_section("Verse 1", "D", "dorian", 8)
    import yaml
    print(yaml.dump(section, allow_unicode=True))
```

---

## ✓ 機能概要 :

### 1. 感情テンプレート連携

- セクション名から emotion (例 : quiet\_pain\_and\_nascent\_strength) を取得
- 対応するテンプレートからテンションと表現スタイル (rhythm, approach) を読み込み

### 2. コード進行へのテンション自動補完

- スケール内で整合するテンションを選定 (例 : D Dorian なら C → Cadd9)

### 3. humanize 処理の適用

- duration / onset\_shift\_ms / articulation を感情テンプレートに応じて調整
- 例 : hope → staccato 比低め、velocity 高め

```
from pathlib import Path
from chord_voicer import apply_tensions
from humanizer import apply_emotional_humanization
from emotional_map_loader import load_emotional_map,
```

```

get_template_for_section
import yaml

def process_chordmap_with_emotions(chordmap_path: str | Path,
emotional_map_path: str = "emotion_map.cleaned.json") -> dict:
    # chordmap を読み込み
    with open(chordmap_path, "r", encoding="utf-8") as f:
        chordmap = yaml.safe_load(f)

    emap = load_emotional_map(emotional_map_path)

    for section in chordmap["sections"]:
        name = section["name"]
        try:
            tpl = get_template_for_section(name,
map_path=emotional_map_path)
        except KeyError:
            print(f"[warn] Emotion template not found for {name}, skipping...")
            continue

        # テンション補完
        for chord in section["chords"]:
            chord["chord"] = apply_tensions(
                base_chord=chord["chord"],
                scale=section["mode"],
                tonic=section["tonic"],
                desired_tensions=tpl.tensions
            )

        # humanize 処理
        chord.update(apply_emotional_humanization(tpl.rhythm))

    return chordmap




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

# 実行例
if __name__ == "__main__":
    newmap = process_chordmap_with_emotions("chordmap_emotions.yaml",
"emotion_map.cleaned.json")
    save_chordmap_yaml(newmap, "chordmap_emotions_humanized.yaml")

```

---

## ✅ 拡張内容のポイント

項目	説明
 音域フィルタリング	コードの構成音が C3～C6 に収まるよう調整（テンションは1オクターブ下げなど）
 テンション抑制	原則「最大5音以内」に制限（6音以上になったらルート重複やテンションを省略）
 DAW 用情報	midi_program, channel, velocity_bias を chord ごとに付与（emotional_map に準拠）

## 🧠 補助関数（追加）

python

コピーする編集する

```
from music21 import chord as m21_chord, pitch as m21_pitch
```

```
def filter_voicing_pitches(pitches: list[str], max_notes=5) -> list[str]:
```

```
    """5音以内に制限しつつ、音域 C3～C6 に収める"""
```

```
    p_objs = [m21_pitch.Pitch(p) for p in pitches]
```

```
    # 音域外ピッチ補正
```

```
    for p in p_objs:
```

```
        while p.midi < 48: # below C3
```

```
            p.octave += 1
```

```
        while p.midi > 84: # above C6
```

```
            p.octave -= 1
```

```
    # 音の重複を除きながら5音までに制限
```

```
    pitch_names = list(dict.fromkeys(p.nameWithOctave for p in p_objs))
```

```
    return pitch_names[:max_notes]
```

## 🎵 統合テンプレート：フル処理コード

python

コピーする編集する

```
from humanizer import apply_emotional_humanization
from emotional_map_loader import load_emotional_map,
get_template_for_section
from chord_voicer import apply_tensions
import yaml
from pathlib import Path
```

```
def process_chordmap_with_full_emotion(chordmap_path: str | Path,
emotion_path="emotion_map.cleaned.json") -> dict:
```

```
    with open(chordmap_path, "r", encoding="utf-8") as f:
```

```

chordmap = yaml.safe_load(f)

emap = load_emotional_map(emotion_path)

for section in chordmap["sections"]:
    name = section["name"]
    try:
        tpl = get_template_for_section(name, map_path=emotion_path)
    except KeyError:
        continue

    for chord in section["chords"]:
        # テンション補完
        voiced = apply_tensions(
            base_chord=chord["chord"],
            scale=section["mode"],
            tonic=section["tonic"],
            desired_tensions=tpl.tensions
        )
        chord["chord"] = voiced

        # 音域と構成音フィルタ
        chord["notes"] = filter_voicing_pitches(
            pitches=chord×get("notes", []), # apply_tensions内で追加されていれば
            max_notes=5
        )

        # 演奏表現: duration / articulation など
        hparams = apply_emotional_humanization(tpl.rhythm)
        chord.update(hparams)

        # DAW 情報付加
        chord["midi_program"] = tpl.approach # ex: "strings", "piano"
        chord["channel"] = 1
        chord["velocity_bias"] = {"low": -5, "mid": 0, "high":
+5}.get(section.get("musical_intent", {}).get("intensity", "mid"), 0)

    return chordmap

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

```

## 補足

- notes フィールドは "C4", "E4", "Bb4" のように octave を含む pitch 名
- midi\_program は将来的に General MIDI 準拠番号化も可 (例: 0 = piano, 48 = strings)
- テンション抑制は 5 音制限 により段階的に自動

---