

✓ 1. スキーマ案 : duration + emotion による演奏制御

```
- chord: Fmaj7
duration: 4.0
emotion: prayer # 情緒名 (emotion_template から取得)
velocity: auto # ← humanizerがemotionに応じて設定
timing_jitter: auto
sustain_style: auto
```

humanizer.py ではこのように処理します :

```
EmotionBehavior = NamedTuple("EmotionBehavior", [
    ("velocity_bias", int),
    ("jitter_ms", float),
    ("release_ratio", float), # 1.0 = full duration
    ("articulation", str), # legato, normal, staccato
])
```

✓ 2. emotion → 演奏スタイルの例 (humanizer 内部テーブル)

```
_EMOTION_TABLE: dict[str, EmotionBehavior] = {
    "hope": EmotionBehavior(+5, 6.0, 0.95, "legato"),
    "prayer": EmotionBehavior(-4, 2.0, 1.10, "legato"),
    "conflict": EmotionBehavior(+2, 10.0, 0.80, "staccato"),
    "sorrow": EmotionBehavior(-2, 4.0, 1.0, "normal"),
    "urgency": EmotionBehavior(+7, 15.0, 0.70, "staccato"),
    # ... 他にも追加可能
}
```

✓ 3. humanizer.py 拡張コード (emotion + duration)

```
from typing import NamedTuple
from dataclasses import dataclass
import random
```

```
class EmotionBehavior(NamedTuple):
    velocity_bias: int
    jitter_ms: float
    release_ratio: float
    articulation: str
```

emotion → 演奏スタイル定義

```
_EMOTION_TABLE = {
    "hope": EmotionBehavior(+5, 6.0, 0.95, "legato"),
    "prayer": EmotionBehavior(-4, 2.0, 1.10, "legato"),
    "conflict": EmotionBehavior(+2, 10.0, 0.80, "staccato"),
    "sorrow": EmotionBehavior(-2, 4.0, 1.0, "normal"),
    "urgency": EmotionBehavior(+7, 15.0, 0.70, "staccato"),
}
```

```

@dataclass
class NoteTiming:
    velocity: int
    onset_shift_ms: float
    duration_ratio: float
    articulation: str

def humanize_from_emotion(emotion: str, base_velocity: int = 64) ->
NoteTiming:
    behavior = _EMOTION_TABLE.get(emotion, EmotionBehavior(0, 4.0, 1.0,
"normal"))

    velocity = min(127, max(1, base_velocity + behavior.velocity_bias))
    onset_shift = randomxuniform(-behavior.xjitter_ms, behavior.jitter_ms)
    return NoteTiming(
        velocity=velocity,
        onset_shift_ms=onset_shift,
        duration_ratio=behavior.release_ratio,
        articulation=behavior.articulation
    )

```

💡 4. 使用例 (chordmap に応じて)

```

for chord in chords:
    emo = chord.get("emotion", "sorrow")
    base_duration = chord["duration"]
    timing = humanize_from_emotion(emo)

    final_duration = base_duration * timing.duration_ratio
    play_chord(chord["label"], velocity=timingxvelocity, duration=final_duration)

```

📝 補足：この実装の利点

項目	効果
感情 + duration	同じコードでも演奏表情が変化 (hope と prayer で同じ Fmaj7 でも印象が変わる)
onset_shift_ms	機械的でない自然な揺らぎ
articulation	発音スタイルも制御可能に (レガートか、スタッカートか)
emotion 管理	chordmap 側で完結、他と連携しやすい

「duration + emotion による humanize 処理」における **onset_shift_ms** や **articulation** は、演奏表現をきめ細かくコントロールするための重要なパラメータです。それぞれの意味は以下の通りです：

onset_shift_ms (オンセットシフト)

意味：

- 音が“楽譜通りに始まる”のではなく、「少し早く」「少し遅れて」演奏されるズレ（ミリ秒単位）。

感情効果：

- 前ノリ (negative shift)：緊張感、勢い、焦燥。
- 後ノリ (positive shift)：リラックス、迷い、柔らかさ。

例：

- 「conflict」や「struggle」では、onset_shift_ms を +10ms にして「やや遅れて響く」ようにすると、感情の“もつれ”を演出できます。

articulation

意味：

- 音の「切れ味」や「滑らかさ」の度合い (staccato, legato, accented, tenuto など)。

感情効果：

- staccato：決意、怒り、不安定さ、軽快さ。
- legato：悲しみ、祈り、回想、温もり。
- accented：強調、情熱、エネルギー。
- tenuto：じっくりと踏みしめるような表現。

使い分けの例：

- 「hope」→ legato
- 「conflict」→ accented + staccato
- 「memory」→ tenuto + legato

duration との関係性

duration_beats (ビート単位での長さ) に対し、**onset_shift_ms** や **articulation** は音の“輪郭”を整える細やかな彫刻です。

- 同じ duration=4.0 のコードでも：
 - onset_shift_ms=-10 & staccato：鋭く刻む緊張感
 - onset_shift_ms=+20 & legato：ゆっくり包み込むような響き

これらを **humanizer.py** に組み込むことで：

- emotion + intensity から、自動的に人間らしい「ズレ」と「抑揚」が加えられる。
- どのセクションも、**“詩の気持ちで演奏”** するようになります。

 **まとめ：感情表現の中核を担う三位一体**

パラメータ	感情との関係性
duration	コードの“滞在時間”。情緒の重さや軽さを表す基準時間
onset_shift_ms	その時間内での“揺れ”や“感情のズレ”を作る
articulation	音の“切れ味”“つながり”で、より細やかな感情のニュアンスを加える

小説家・朗読家としてのご感性があるからこそ、音楽も「語り」や「息づかい」のように制御できるのです。

その声のニュアンスや行間の間（ま）をコード演奏に取り込む、humanizer.py 拡張をご提案します。

humanizer.py 拡張：emotion × duration による expressive humanization

1. EmotionExpressionProfile の導入

```
from typing import Literal, Dict
from pydantic import BaseModel
```

```
class EmotionExpressionProfile(BaseModel):
    onset_shift_ms: float      # 開始タイミングのズレ
    articulation: Literal["legato", "staccato", "tenuto", "accented"]
    velocity_bias: int         # ダイナミクス調整 (-10 ~ +10)
    sustain_factor: float      # duration に対する実際の音の長さ (0.5 ~ 1.0)
```

感情テンプレートごとの定義 (例)

```
EMOTION_EXPRESSIONS: Dict[str, EmotionExpressionProfile] = {
    "hope": EmotionExpressionProfile(onset_shift_ms=+12, articulation="legato",
    velocity_bias=+6, sustain_factor=0.95),
    "conflict": EmotionExpressionProfile(onset_shift_ms=-8,
    articulation="accented", velocity_bias=+4, sustain_factor=0.7),
    "prayer": EmotionExpressionProfile(onset_shift_ms=+20,
    articulation="tenuto", velocity_bias=-2, sustain_factor=1.0),
    "loss": EmotionExpressionProfile(onset_shift_ms=+25, articulation="legato",
    velocity_bias=-6, sustain_factor=0.85),
    "struggle": EmotionExpressionProfile(onset_shift_ms=-10,
    articulation="staccato", velocity_bias=+3, sustain_factor=0.6),
}
```



2. apply_emotional_humanization 関数

```
def apply_emotional_humanization(
    chord_event: dict,
    emotion_label: str,
    default_duration: float = 2.0
) -> dict:
    """
    chord_event: { chord: ..., offset: ..., duration: ... }
    emotion_label: "hope" など
    """
    profile = EMOTION_EXPRESSIONS.get(emotion_label)
    if profile is None:
        return chord_event # 未登録の感情なら変更しない

    # duration 補完 (なければ平均値)
    duration = chord_event.get("duration", default_duration)

    # 実音の長さを計算 (リリースタイムなどに反映可能)
    actual_length = duration * profile.sustain_factor

    return {
        **chord_event,
        "duration": duration,
        "humanize": {
            "onset_shift_ms": profile.onset_shift_ms,
            "articulation": profile.articulation,
            "velocity_bias": profile.velocity_bias,
            "actual_duration": round(actual_length, 3),
        }
    }
```



出力例

```
{
  "chord": "Fmaj7",
  "offset": 12.0,
  "duration": 2.0,
  "humanize": {
    "onset_shift_ms": 12,
    "articulation": "legato",
    "velocity_bias": 6,
    "actual_duration": 1.9
  }
}
```



この仕組みでできること

- 各コードの演奏が、声のように語るようになります

- 小説のセリフのように、「同じ言葉でも言い回しが変わる」ような表現が可能になります
- articulation や onset_shift の揺らぎが、人間らしい情緒と温度感をもたらします