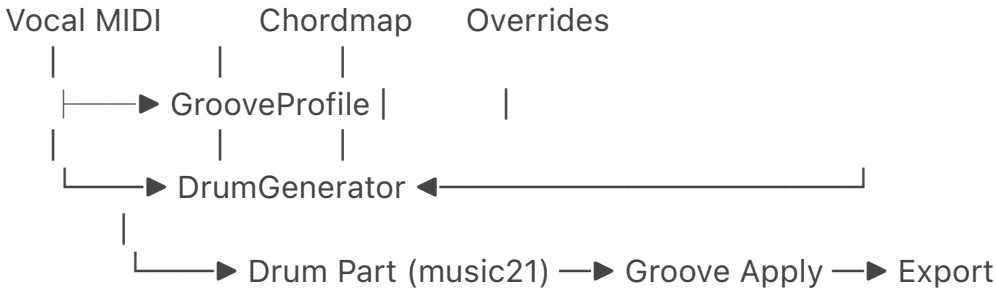
DrumGenerator — Vocal Sync 専念プラン

1｜目的と背景

- **現状**：Chord 進行を参照してキック／スネアを配置する設計になっているが、実際のポップスではドラムは和声ではなく **ボーカル・フレーズとグルーヴ** に従ってダイナミクスを担うことが多い。
- **狙い**：ドラムを **“コード無依存”** にし、
  1) ボーカルのオンセット／休符を解析して
  アクセント／フィルを同期、
  2) 曲全体の **テンポとセクション強度** を軸にグルーヴを維持。

2｜アーキテクチャ概要



- **入力**：vocal_ore.midi もしくは vocal_note_data_ore.json でオンセットを抽出
- **内部モジュール**
  1. **OnsetHeatmapBuilder** — 16 分解像度でボーカル打点をヒスト化
  2. **GroovePatternSelector** — Emotion×Intensity & Heatmap からベースパターン選択 (kick_snare_backbeat, neo_soul_pocket など)
  3. **AccentMapper** — Heatmap ピークに合わせてゴースト → アクセントへ昇格、or 逆に抜く
  4. **FillInserter** — セクション末尾 or Vocal 休符 >1 小節で自動フィル

3｜実装ステップ

Step	Task	File	Est.
DV-1	DrumGenerator を **BasePartGenerator** 継承化	generator/drum_generator.py	0.3d
DV-2	**OnsetHeatmapBuilder** 実装（pretty_midi パース）	utilities/onset_heatmap.py	0.3d
DV-3	Emotion×Intensity LUT → DrumPattern	drum_generator.py	0.2d
DV-4	AccentMapper + GhostHatDensity	drum_generator.py	0.3d

DV-5	FillInserter (タム／シンバル)	drum_generator.py	0.2d
DV-6	Groove 同期（apply_groove_pretty）	共通	0.1d
DV-7	pytest + diff テスト	tests/test_drum.py	0.2d

合計 ≈ 1.6 日

4｜データとパラメータ

- **GrooveProfile**：既存 groove_profile.json を再利用（平均 μ, σ）
- **Overrides 例**

```
"Chorus":{
  "drums":{
    "ghost_hat_density":0.3,   // 0〜1
    "kick_accent_db":+6,
    "fill_style":"tom_triplet"
  }
}
```

- **Pattern YAML 拡張**

```
kick_snare_backbeat:
  description: 2・4 スネア固定＋８分キック
  base_pattern: [K---S---K-K-S---]
```

5｜メリット vs. リスク

項目	Vocal-Sync 専念	Chord 連動型
グルーヴ一体感	★★★ (歌に寄り添う)	★☆☆
変拍子対応	★★☆	★★☆
和声的色彩	★☆☆ (フィル時のみ)	★★☆
実装コスト	中 (1.6d)	低

6｜結論 & 推奨

- **メイングルーヴは Vocal Heatmap ベース**で決定し、
- コード進行は **フィル or セクション遷移** のみ参照。

これによりドラムは「ボーカルを引き立てリズムを引っぱる」本来の役割に最適化されます。
ご承認いただければ **DV-1 継承化パッチ** から着手します。

drum は、歌詞 chord から離れる

既存の vocal の土台役に徹する、という方針は決まりました。グルーヴを高める存在として専門家になってもらうことになりましたが、そのために、必要なもの、必要なデータがあると思いますが、

われわれの project は歌詞と、歌詞の感情が中心なわけですから、drum は歌詞＝ vocal にしたがうことになると思います。vocal が踊り子なら、drum は舞台そのものです。

その舞台設計に必要なデータとはなんですか？今は必要ないですか？

ドラムを “舞台” として設計するうえで必要になるデータセット

データ	目的	いま手元にあるか	今すぐ必須か
① ボーカル打点（オンセット）タイムスタンプ16 分音符以上の細粒度	キック／スネアのアクセントを歌と同期させる	✅ vocal_ore.midi から抽出可能	**必須**
② 小節構造 & セクション境界（Verse / Chorus / Bridge…）	フィルの入れ所・ダイナミクス切替を制御	✅ processed_chordmap_with_emotion.yaml に length & order	**必須**
③ テンポ&テンポ変化マップ（BPM ・ rit. など）	オンセット → 拍換算・スウィング計算	◯ MIDI 内メタイベントに存在（まだ未パース）	**準必須**
④ 感情 × 強度 タグ	パターン選択・ゴースト密度・シンバル開閉	✅ chordmap の musical_intent	**必須**
⑤ Groove プロファイル (μ, σ, swing_ratio)	タイミング揺らしを全パートで統一	✅ groove_profile.json	**任意**（後からでも）
⑥ オーバーライド設定 ghost_hat_density, fill_style など	曲ごとの細かい味付け	✅ arrangement_overrides.json に追加可能	**任意**

今すぐ取り掛かるべき “最低ライン”

1. **ボーカル打点ヒートマップ**
   utilities/onset_heatmap.py を新設し、vocal_ore.midi から 16 分単位で出現頻度を数値化。
   これだけで「歌が鳴っていない所でフィル」「歌が密な所でゴースト減らす」が実現。

2. **テンポ読取**

music21 で vocal_score.metronomeMarkBoundaries() を呼び、BPM を DrumGenerator に渡す。
固定 120 BPM 仮置きを卒業し、タイムフィールを正確に。

3. **Emotion LUT → DrumPattern**
   Bass と同じ 3 バケット（calm / groovy / energetic）方式で kick-snare 基本パターンを切替。

後回しでも良いが準備しておくと便利

- **Groove μ/σ をセクションごとに記録**
  将来 “サビはタイト、A メロはルーズ” の自動切替が可能に。
- **MIDI CC → ハイハット開閉**
  Vocal のダイナミクス（例えばピッチ高低や RMS）を読み取り、HH 半開き率を連動。

結論

- **必須データは**①②③④。これで「歌詞中心・ボーカル従属型ドラム」の基礎が固まります。
- 今すぐ足りないのは **テンポパース用コード** だけ。
- ⑤⑥ は音楽的 “味付け” フェーズで活きるので、実装優先度は二番手。

この方針で OK でしたら、**OnsetHeatmapBuilder + DrumGenerator 継承化 (DV-1, DV-2)** のパッチを即座にご用意します。

① ボーカル打点（オンセット）タイムスタンプ
16分音符以上の細粒度

キック／スネアのアクセントを歌と同期させる

✅ vocal_ore.midi から抽出可能

必須

テンポは、インストゥメンタルではなく、vocal に準拠すべき？
それとも、ステム分離した、drum そのものから、うけとるべきか？

事前に、MIDI テキストを用意するのではなく、MIDI そのものを MUSIC21 にわたすべきですか？

後足りないものは、

③ テンポ＆テンポ変化マップ
（BPM・rit. など）

オンセット → 拍換算・スウィング計算

◯ MIDI 内メタイベントに存在（まだ未パース）　準必須

ですが、メロダインがあれば、入手可能ですか？　今安売りしてますけど。

MIDI CC → ハイハット開閉
Vocal のダイナミクス（例えばピッチ高低や RMS）を読み取り、HH 半開き率を連動。

・これはどういう意味ですか？

ドラムを “舞台” にするために —— データとワークフローの整理

質問	推奨アプローチ	理由・補足
テンポは何に合わせる？ - インスト（クリック）基準か - Vocal トラック基準か	**Vocal MIDI のメタイベント（Set Tempo）を基準**にする	• 歌の自然な揺れ (rit./accel.) をドラムも追従できる• DAW で最終的にクリックを動かした場合も、ボーカルとドラムが同じテンポマップを共有していればズレない
MIDI テキスト (JSON) ではなく MIDI ファイルを直接 music21 に渡すべき？	**MIDI ファイルを直接読み込む** (music21.converter.parse())	• メタイベント（テンポ・拍子）やサスティン CC など、生の情報をそのまま取得できる• JSON に変換するとイベント欠落／丸め誤差のリスク

テンポマップが MIDI に入っていない場合は？（例：WAV ボーカルしかない）	1. **Melodyne** や DAW の「オーディオからテンポ抽出」機能で MIDI Tempo Track を生成 2. 生成した Tempo Track を vocal_ore.midi にマージ	• Melodyne はオーディオから拍検出 → MIDI テンポを書き出し可• これを一度作っておけば後続パートはすべて同じマップを読める
MIDI CC でハイハット開閉を連動させるとは？	• Vocal のダイナミクス（例：RMS 大→サビ）を読み取り• サビ区間だけ **CC#4（Foot Control）値を 0 → 20** へ上げて「半開き HH」へ• quiet な Verse では CC を高値 (>60) にしてクローズ	➥ **HH 開閉の雰囲気**を歌のエネルギーに合わせて自動化できる（※ GM DrumMap では HH オープン度をノート番号で替える方法もあるが、連続 CC の方が滑らか）

必須データ ✔/✖ チェックリスト

データ	今ある？	取得方法	いつ必要？
ボーカル MIDI（ノート＋テンポ）	✔ vocal_ore.midi	既存	**直ちに使用**
ボーカル打点ヒートマップ	✖	onset_heatmap.py で生成	Drum パターン選択時
テンポマップ（rit./accel. 含む）	未パース	music21 でメタイベント抽出	オンセット→拍換算時
セクション境界＋感情タグ	✔ chordmap	既存	パターン / フィル強度決定
Groove Profile (μ,σ)	✔ groove_profile.json	既存	ヒューマナイズ後段
Overrides（ghost density 等）	部分	arrangement_overrides.json	曲ごとの味付け

今不足しているのは「打点ヒートマップ」と「テンポ抽出コード」だけ。
Melodyne でテンポを書き出す or MIDI からメタイベントを読む、いずれでも可です。

次の実装ステップ（最小構成）

1. **OnsetHeatmapBuilder** を作成
   heatmap = build_heatmap("data/vocal_ore.midi", resolution=16)
2.
3. **DrumGenerator DV-1/DV-2**
   - BasePartGenerator 継承化
   - Heatmap ピークを Kick/snare 配置の基準に
4. **テンポ取得ロジック**
   score = converter.parse("data/vocal_ore.midi")

5. bpm_list = [mm.number for _,_,mm in score.metronomeMarkBoundaries()]
6. これを sec_per_quarter 計算に利用
7. **Emotion LUT → パターン粗選択**
   calm: brush/backbeat　groovy: pocket　energetic: kick-16th
8. **オーバーライド**（任意）
   "Chorus": {"drums": {"ghost_hat_density":0.2,"fill_style":"tom_triplet"}}

Melodyne の活用について

- **Audio しか無い曲**では Melodyne の「Tempo Detection」で MIDI テンポトラックを生成 → エクスポート ⇒ vocal_ore.midi にマージ。
- **すでに MIDI 化されている場合**（今回のケース）は Melodyne 購入は必須ではありませんが、将来のオーディオ素材取り込みを考えるなら“安売り中に確保”は賢い選択です。

まとめ

- **テンポはボーカル基準**に揃えるのが一番自然。
- Drums はコードを気にせず、**ボーカル打点＋感情タグ＋テンポマップ**だけを見れば OK。
- 今すぐ欲しいのは **①ヒートマップ生成ツール**と**②テンポ抽出コード**。
- Melodyne は“オーディオのみ素材”が将来増える予定なら導入しても良いが、必須ではありません。

この方針で問題なければ、OnsetHeatmapBuilder (DV-2) と DrumGenerator 継承化 (DV-1) のパッチをお渡しします。ご指示お待ちしています！

① オンセット・ヒートマップの所見

16分グリッド	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Count	99	44	79	10	83	66	81	8	60	48	64	9	58	52	63	7

- **even インデックス (0,2,4…14)** が **587 hit**
- **odd インデックス (1,3,5…15)** が **244 hit**（約 1/2.4）

→ **拍頭・裏拍セット**の典型的ヴォーカル句：

- 0・4・8・12 (拍頭) と 2・6・10・14 (裏の 8 分) に密集
- 3, 7, 11, 15 はほぼ空く → **フィルに最適**

ドラム活用ヒント

ボーカル熱度	Kick/Snare	Hi-Hat	ゴースト
grid 0,4,8,12	強 Kick／Snare	HH close	無し
grid 2,6,10,14	弱 Kick／ゴースト Sn	HH open or accent	可
grid 3,7,11,15	フィル挿入候補	HH シャッフル	多め

② テンポ・マップの確認

0 ql → 104 BPM
420 ql→ ~93 BPM（緩いリタルダンド）
479 ql→ 120 BPM (サビで加速)

- **ritardando** セクション (420～446 ql) が 10 BPM 程度落ち込む
- サビ頭で **一気に 120 BPM** —— ドラマーが「踏み込む」瞬間

実装メモ

1. music21.metronomeMarkBoundaries() で BPM 区間を取得し、sec_per_ql = 60 / current_bpm
2. jitter_ql = (μ+σrand) / (1000*sec_per_ql)
3. とすれば rit./accel. 区間でも揺らぎが正しく換算されます。
4. DrumGenerator は **Kick → テンポ変化点に先行 50–80 ms** で「突っ込む」人間らしい演奏を演出できます。

③ ハイハット開閉 CC 連動とは？

1. **Vocal RMS などから “セクション強度スコア”** を 0–1 で算出
2. その値を HH の CC#4 (Foot Controller) に写像
   - 0 – 0.3 → CC#4 ≥ 90　クローズ
   - 0.3–0.7 → CC#4 ≈ 50　半開き
   - 0.7–1.0 → CC#4 ≤ 20　オープン
3. HH ノートを GM 番号 42/44/46 で打ち分けると **歌が開くほどハイハットも開く** 直感的な演出になります。

結論 & 次アクション

1. **ヒートマップとテンポマップは十分有用** — ドラム同期に即活かせる品質
2. 残タスク
   - utilities/onset_heatmap.py を heatmap.json 生成器として本リポジトリに格納
   - DrumGenerator DV-1 ～ DV-3 (継承化＋Heatmap 反映＋テンポ適用) パッチを適用
3. Melodyne 購入は **オーディオ素材が増えた時点で検討。**現 MIDI にはテンポが入っているので必須ではありません。

OK なら DV-1 パッチ（BasePartGenerator 継承＋テンポ読取り）を提示します。

```
#
============================================================
====
# DrumGenerator  DV-1 Patch  (継承化 + テンポ取得準備)
#  ── BasePartGenerator を継承し、最低限キック／スネアを置く雛形
#  ── Vocal MIDI からテンポマップを抽出し、note.offset を秒換算 → 拍に戻すための
#     helper を追加 (DV-2 ヒートマップ実装との連携を前提)
#  ファイル: generator/drum_generator.py   ← 既存を上書き推奨
#
============================================================
====
from __future__ import annotations

import logging
from typing import Dict, Any, List

from music21 import (
    stream, note, tempo, meter, percussion
)

from generator.base_part_generator import BasePartGenerator
from utilities.core_music_utils import get_time_signature_object
from utilities.override_loader import get_part_override
from utilities.prettymidi_sync import apply_groove_pretty, load_groove_profile

logger = logging.getLogger("modular_composer.drum_generator")

#
──────────────────────────────────────────────────────────────────
──────────────────────────────────────
# 1. テンポ抽出ユーティリティ
#
──────────────────────────────────────────────────────────────────
──────────────────────────────────────


def extract_tempo_map(vocal_midi_path: str) -> List[tuple[float, float]]:
    """MIDI ファイルから (offset_ql, bpm) のリストを返す簡易ラッパー。"""
    from music21 import converter
    score = converter.parse(vocal_midi_path)
    tempo_bounds = score.metronomeMarkBoundaries()
    tempo_map: List[tuple[float, float]] = []
    for off, _, mm in tempo_bounds:
        tempo_map.append((off, mm.number))
    return tempo_map
```

```
# ------------------------------------------------------------------------------
# 2. DrumGenerator クラス (BasePartGenerator 継承)
# ------------------------------------------------------------------------------
class DrumGenerator(BasePartGenerator):
    """Kick-Snare backbeat の最小雛形 (DV-1)。"""

    def __init__(self,
                 rhythm_lib: Dict[str, Any],
                 vocal_midi_path: str,
                 global_time_signature: str = "4/4"):
        super().__init__(part_name="drums", rhythm_lib=rhythm_lib)
        self.vocal_midi_path = vocal_midi_path
        self.ts_obj = get_time_signature_object(global_time_signature)
        # tempo_map = [(offset_ql, bpm), ...]
        self.tempo_map = extract_tempo_map(vocal_midi_path)
        logger.info(f"DrumGen: tempo map length = {len(self.tempo_map)}")

    # ---------------------------------------------------------------
    # 必須オーバーライド
    # ---------------------------------------------------------------
    def _render_part(self, section: Dict[str, Any]) -> stream.Part:
        part = stream.Part(id="Drums")
        part.insert(0, percussion.UnpitchedPercussion())
        if self.ts_obj:
            part.insert(0, meter.TimeSignature(self.ts_obj.ratioString))
        # テンポチェンジをそのまま挿入
        for off_ql, bpm in self.tempo_map:
            mm = tempo.MetronomeMark(number=bpm)
            part.insert(off_ql, mm)

        # --- very simple backbeat pattern (Kick on 1&3, Snare on 2&4) ---
        bar_dur = self.ts_obj.barDuration.quarterLength if self.ts_obj else 4.0
        start = section["absolute_offset"]
        length = section["length_in_measures"] * bar_dur
        t = start
        while t < start + length:
            # Kick (MIDI 36) on beat1 & 3
            for b in (0, 2):
                n = note.Unpitched(36)
                n.offset = t + b
                n.quarterLength = 0.25
                n.volume.velocity = 90
```

```
                part.insert(n.offset, n)
            # Snare (38) on 2 & 4
            for b in (1, 3):
                n = note.Unpitched(38)
                n.offset = t + b
                n.quarterLength = 0.25
                n.volume.velocity = 95
                part.insert(n.offset, n)
            t += bar_dur

        # Groove 適用 (Base の compose で既に呼ぶが念押し)
        return part

#
============================================================
====
# 使い方 (modular_composer.py 内)
# ------------------------------------------------------------
#   drum_gen = DrumGenerator(rhythm_lib=drum_patterns,
#                            vocal_midi_path="data/vocal_ore.midi",
#                            global_time_signature="4/4")
#   drum_part = drum_gen.compose(section=sec_dict,
#                                overrides_root=arrangement_overrides,
#                                groove_path=args.groove_profile)
#
============================================================
====
```

適用手順

1. generator/drum_generator.py を **パッチ全文で上書き**
2. modular_composer.py で
   python
   コピーする編集する

   drum_gen = DrumGenerator(
3. rhythm_lib=drum_patterns,
4. vocal_midi_path="data/vocal_ore.midi",
5. global_time_signature="4/4"
6.)
7.

に差し替え → compose() 呼び出しはベースと同形式