

# § 14 PoC 3 — 巷の実コードに頻出する「ガチ動的」Python 20 イディオムの Isolate bit-exact + 速度比較

項目	値
対象 spec	spec_section_14_draft.md v0.1, § 14.9 PoC 計画 PoC 3 系
実施日	2026-06-04
環境	WSL Ubuntu 24.04 / Rust 1.93.1 / CPython 3.12.3 / PyO3 0.22.6
目標	metaclass / <code>__init_subclass__</code> / <code>type()</code> / descriptor / <code>__getattr__</code> / <code>__setattr__</code> / <code>__call__</code> / functools / dataclass / enum / generator / contextmanager / eval-exec という実コードに普通に出る動的イディオム 20 件を、Deterministic Python Isolate で (1) bit-exact 検証、(2) <code>python3</code> との end-to-end 速度比較
正しさ結果	☑20/20 PASS(stdout SHA-256 完全一致)
速度結果	☑Isolate / python3 = <b>median 0.793</b> × (Isolate が ~21% 速い、全 20 件で一貫)

## 0. 結論

- 正しさ: PoC 1( Any 5)・PoC 2( `*args/**kwargs` + 動的 `getattr/setattr` + monkey 15)に続き、PoC 3 では「巷の実コードでそのまま見る」レベルの動的構文 20 件 —— メタクラス登録・`__init_subclass__`・`type()` 動的クラス生成・記述子( `__set_name__` )・`__getattr__` / `__setattr__` 介入・`__call__`・singledispatch・dataclass `__post_init__`・enum・ジェネレータ委譲・context manager・eval/exec —— が Python = Rust+CPython-Isolate で SHA-256 完全一致。累計 40/40。
- 速度: 同一サンプルの `launch-to-exit` 実時間を `python3 <file>` と Isolate バイナリで比較すると、Isolate は全 20 件で **0.73~0.81**×(中央値 0.793×、~21% 速い)。これは「Python の実行が速い」のではなく(両者は同一 CPython バイトコードを走らせるので実行コストは定義上同一)、プロセス起動経路が軽いことに由来する正直な差。後述 § 3.1 参照。

## 1. 検証カテゴリと結果(bit-exact)

### A. クラス機構 / メタクラス(5 件)

#	sample	検証イディオム	SHA(冒頭)	バイト
01	01_metaclass_registry.py	metaclass <code>__new__</code> でサブクラスを registry 自動登録(Django/Flask 風)	7d9240c8cddb0bba...	26
02	02_init_subclass_registry.py	<code>__init_subclass__</code> (route=...) でルート登録	99a1c35dafb8651f...	20
03	03_type_dynamic_class.py	<code>type(name, bases, ns)</code> で実行時にクラス生成	82f0ae64af86245c...	31
04	04_abc_abstractmethod.py	<code>abc.ABC</code> + <code>@abstractmethod</code> + 抽象生成ガード	a22aa18a91131451...	40
05	05_slots_repr.py	<code>__slots__</code> + <code>__repr__</code> + <code>__add__</code> + slots ガード	c5c0ec5db354fff8...	52

### B. 属性 / 記述子プロトコル(5 件)

#	sample	検証イディオム	SHA	バイト
06	06_property_validation.py	<code>@property</code> getter/setter + 検証(model 風)	04158b904c33dfa6...	63
07	07_descriptor_set_name.py	型付き記述子 + <code>__set_name__</code> (ORM field 風)	49a95f6525d03425...	32
08	08_getattr_lazy_proxy.py	<code>__getattr__</code> 遅延プロキシ(config 動的属性)	3ebc30eb58d03ad0...	34
09	09_setattr_frozen.py	<code>__setattr__</code> 介入で frozen object	2ac36a35ab733c6e...	37
10	10_callable_memoizer.py	<code>__call__</code> オブジェクトを memoize 関数として( <code>@Memoize</code> で fib)	eaf8500a242b7c8c...	58

## C. functools / collections / dataclass / enum(7 件)

#	sample	検証イディオム	SHA	バイト
11	11_functools_partial.py	functools.partial 部分適用 + partial of partial	e63d13ecc8032924...	30
12	12 singledispatch.py	functools.singledispatch 型ディスパッチ汎関数	14eef95127b3f5e6...	34
13	13_wraps_counter.py	functools.wraps + クロー ジャ呼出カウンタ	439d23b18ac480bf...	59
14	14_namedtuple.py	collections.namedtuple + _replace / _asdict	a36a67176d88230e...	96
15	15_dataclass_postinit.py	@dataclass(order=True) + __post_init__ + field(default_factory)	66affeb86c3918a6...	155
16	16_enum_functional.py	Enum + auto() + メソッド + 関数型 API	3b269ae5ab9cd46b...	79
17	17_defaultdict_counter.py	defaultdict + Counter.most_common 語頻 度	8da960082a535fb5...	91

## D. 制御フロー動的(generator / context manager / eval-exec)(3 件)

#	sample	検証イディオム	SHA	バイト
18	18_generator_yield_from.py	yield from 再帰 flatten + send() パイプライン	0799a4bdec1dab91...	44
19	19_contextmanager.py	contextlib.contextmanager + ネスト	d2a24f41adea50a9...	97
20	20_eval_exec_namespace.py	exec(src, g, ns) で関数定義 + eval(expr, g, l) (§13 DEFER_HARD)	ce08d3d0c4209853...	55

## 2. 実行結果(bit-exact ログ)

```
=== build runner ===
  Compiling poc3_realworld v0.1.0
    Finished `release` profile [optimized] target(s) in 8.04s
binary: 338568 bytes
```

```

=== run + bit-exact compare (20 samples) ===
01_metaclass_registry      PASS  sha=7d9240c8cddb0bba... (26B)
02_init_subclass_registry  PASS  sha=99a1c35dafb8651f... (20B)
03_type_dynamic_class      PASS  sha=82f0ae64af86245c... (31B)
04_abc_abstractmethod      PASS  sha=a22aa18a91131451... (40B)
05_slots_repr              PASS  sha=c5c0ec5db354fff8... (52B)
06_property_validation     PASS  sha=04158b904c33dfa6... (63B)
07_descriptor_set_name     PASS  sha=49a95f6525d03425... (32B)
08_getattr_lazy_proxy      PASS  sha=3ebc30eb58d03ad0... (34B)
09_setattr_frozen          PASS  sha=2ac36a35ab733c6e... (37B)
10_callable_memoizer       PASS  sha=eaf8500a242b7c8c... (58B)
11_functools_partial       PASS  sha=e63d13ecc8032924... (30B)
12 singledispatch          PASS  sha=14eef95127b3f5e6... (34B)
13_wraps_counter           PASS  sha=439d23b18ac480bf... (59B)
14_namedtuple              PASS  sha=a36a67176d88230e... (96B)
15_dataclass_postinit      PASS  sha=66affeb86c3918a6... (155B)
16_enum_functional         PASS  sha=3b269ae5ab9cd46b... (79B)
17_defaultdict_counter     PASS  sha=8da960082a535fb5... (91B)
18_generator_yield_from    PASS  sha=0799a4bdec1dab91... (44B)
19_contextmanager          PASS  sha=d2a24f41adea50a9... (97B)
20_eval_exec_namespace     PASS  sha=ce08d3d0c4209853... (55B)

=== summary: PASS=20 / 20 ===

```

### 3. 速度比較(end-to-end wall time)

計測方法: 各サンプルにつき `python3 <file>` と Isolate バイナリを **warmup 3 回(破棄)**→計測 30 回、各回の launch-to-exit を `date +%s%N` で取り、**中央値**を採用。両者とも完全なプロセス起動 + インタプリタ初期化を含む。

sample	python3(ms)	isolate(ms)	iso/py
01_metaclass_registry	19.94	15.94	0.799
02_init_subclass_registry	19.87	15.74	0.792
03_type_dynamic_class	19.97	16.03	0.803
04_abc_abstractmethod	19.97	15.75	0.789
05_slots_repr	20.07	15.69	0.782
06_property_validation	19.91	15.85	0.796
07_descriptor_set_name	19.82	15.80	0.797
08_getattr_lazy_proxy	19.82	15.65	0.789
09_setattr_frozen	19.60	15.85	0.809
10_callable_memoizer	19.82	15.82	0.798
11_functools_partial	19.94	15.90	0.797
12 singledispatch	28.10	21.60	0.769
13_wraps_counter	19.82	15.89	0.802
14_namedtuple	19.95	15.78	0.791
15_dataclass_postinit	34.43	25.04	0.727
16_enum_functional	21.22	16.61	0.783
17_defaultdict_counter	20.59	16.12	0.783
18_generator_yield_from	19.79	15.98	0.808

19_contextmanager	20.43	16.03	0.785
20_eval_exec_namespace	19.94	15.83	0.794

```
=== aggregate over 20 samples ===
python3 total median-sum : 423.0 ms
isolate total median-sum : 332.9 ms
overall ratio (sum iso/py): 0.787
median per-sample ratio   : 0.793
```

### 3.1 速度差の正直な解釈 ★重要

- ・「Python が速くなった」のではない。Isolate もユーザコードは **埋め込み CPython 3.12 のバイトコードインタプリタ**でそのまま実行する。`python3` と Isolate は **同一のオペコードを同一の VM** で走らせるので、コード実行そのもののコストは**定義上同一**(だから bit-exact が成り立つ)。
- ・差は「**起動経路**」。`python3 <file>` の CLI は、`site` 初期化・`sys.argv` /path セットアップ・`__main__` モジュール機構・ファイルの `import` 機構経由ロード等の**付帯処理**を持つ。埋め込みランナーは `Py_Initialize` → `PyRun_String(code)` の**最短経路**で走り、ストリップ + LTO + `panic=abort` の薄い 338 KB バイナリ。差分はほぼこの **launcher オーバーヘッド**(全件で ~4ms 一定)。
- ・したがって速度比較の正しい主張は「**Hybrid Isolate を挟んでも遅くならない。むしろ端から端の実時間では ~21% 速い**」。実行が重いサンプル(#12 singledispatch、#15 dataclass = import が重い)では絶対時間が伸びるが、比率は他と同水準(0.73~0.77×)で、起動経路差が支配的という解釈と整合する。
- ・**注意点(over-claim 回避)**: この ~21% はあくまで **short-lived な一発実行**の話。サーバ常駐や長時間バッチのように起動コストが償却される使い方では、両者の差は実行時間に埋もれて **~1.0× に漸近**する見込み(本 PoC では未計測、PoC 4+ 候補)。

## 4. 重要な観察

1. 「**巷の実コード**」レベルでも **1件もズレない**: メタクラス登録・記述子・`__getattr__` / `__setattr__` 介入・dataclass・enum・generator・context manager・eval/exec まで、業務 Python に普通に出るイディオムを 20 並べても **20/20 bit-exact**。PoC 2 の「reject カテゴリーが動く」に続き、**reject でない普通の動的コードも当然そのまま動くことを確認**。
2. **eval/exec(#20)が素直に通る**: §13 で DEFER\_HARD だった `exec(src, globals, ns)` による関数定義と `eval(expr, g, l)` の式評価が、Isolate(= 同一プロセス CPython)で完全一致。動的コード生成は「Rust で再実装」ではなく「CPython にそのまま委譲」が正しい、という §14 の方針を再確認。
3. **import が重いサンプルが速度の天井を作る**: dataclasses (#15)・`functools.singledispatch` (#12)は `stdlib import` が起動時間を押し上げる(34ms / 28ms)。これは `python3` / Isolate 双方に等しく効くため比率は不変。**動的機能の複雑さではなく import グラフが startup コストの主因**という、Hybrid 設計に効く知見。

4. バイナリは PoC 1/2 と同一 338,568 バイト: runner が完全同型なので当然だが、サンプル数・難易度が増えても同梱コストは固定を 3 度目の確認。
5. 速度比較は「遅くならない」を実証する形で機能: Hybrid Isolate の懸念は通常「ネイティブを挟む分のオーバーヘッド」だが、実測は逆に launcher 経路が軽くなって ~21% 速い。少なくとも「Isolate を挟むと遅い」という反論は short-lived 実行に関しては成立しない。

## 5. 配布物

```
engine/poc3_realworld/
├─ Cargo.toml                pyo3 0.22 + auto-initialize
├─ src/main.rs               汎用 Isolate runner(PoC 1/2 と同型)
├─ samples/                  20 件(A:01-05 / B:06-10 / C:11-17 / D:
18-20)
|   ├─ 01_metaclass_registry.py    metaclass __new__ registry
|   ├─ 02_init_subclass_registry.py __init_subclass__ (route=)
|   ├─ 03_type_dynamic_class.py     type(name, bases, ns)
|   ├─ 04_abc_abstractmethod.py     abc.ABC + abstractmethod
|   ├─ 05_slots_repr.py             __slots__ + __repr__ + __add__
|   ├─ 06_property_validation.py     @property getter/setter
|   ├─ 07_descriptor_set_name.py     descriptor + __set_name__
|   ├─ 08_getattr_lazy_proxy.py      __getattr__ proxy
|   ├─ 09_setattr_frozen.py          __setattr__ frozen guard
|   ├─ 10_callable_memoizer.py       __call__ memoizer (@Memoize fib)
|   ├─ 11_functools_partial.py       functools.partial
|   ├─ 12 singledispatch.py          functools.singledispatch
|   ├─ 13_wraps_counter.py           functools.wraps + counter
|   ├─ 14_namedtuple.py              namedtuple _replace/_asdict
|   ├─ 15_dataclass_postinit.py      @dataclass __post_init__
|   ├─ 16_enum_functional.py         Enum + auto() + 関数型 API
|   ├─ 17_defaultdict_counter.py     defaultdict + Counter
|   ├─ 18_generator_yield_from.py    yield from + send()
|   ├─ 19_contextmanager.py          contextlib.contextmanager
|   └─ 20_eval_exec_namespace.py     eval / exec + namespace
├─ run_bit_exact.sh             ビルド → 20件実行 → SHA-256 比較
├─ run_speed.sh                 warmup 3 + reps 30、python3 vs Isolate 中
央値
└─ target/release/poc3_isolate    338,568 B Isolate runner
```

## 6. § 14.9 PoC ロードマップ 進捗

Phase	内容	状態
☑PoC 0	最小実証	完了
☑PoC 1	Any 5 sample	完了
☑PoC 2		完了

Phase	内容	状態
	<code>*args/**kwargs</code> + 動的 getattr/setattr + monkey 15 sample	
☒PoC 3	巷の実コード動的イディオム 20 sample + 速度比較	完了(本書)
PoC 4	startup 償却の検証(常駐 / バッチ での $\sim 1.0\times$ 漸近)、musl-libm pin、cross-platform(x86_64 / aarch64 / WASI)	次
PoC 5	RustPython 版(Light tier)で同一 20 sample 実行、差分計測	

PoC 3 完了。巷の実コードに普通に出る動的イディオム 20 件(metaclass / 記述子 / `__getattr__` / `__setattr__` / `functools` / `dataclass` / `enum` / `generator` / `contextmanager` / `eval-exec`)が、Hybrid Isolate 内で 20/20 bit-exact。累計 40/40。さらに end-to-end 速度比較で Isolate は python3 の  $0.793\times$ ( $\sim 21\%$  速い、起動経路差由来・全件一貫)。「Isolate を挟むと遅い」は short-lived 実行では不成立。PoC 4(startup 償却 / cross-platform)へ進行可。