

§ 14 PoC 2 結果報告 — `*args/**kwargs` + 動的 `getattr/setattr` + runtime monkey patching の Isolate bit-exact

項目	値
対象 spec	<code>spec_section_14_draft.md</code> v0.1, § 14.9 PoC 計画 PoC 2
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	§ 13 で <code>DEFER_HARD</code> / <code>NON_LITERAL</code> / <code>PHASE_2_FUTURE</code> に分類されている 3 種の動的構文(任意 <code>*args/**kwargs</code> 、変数引数 <code>getattr/setattr</code> 、runtime monkey patching)を各 5 サンプル、Deterministic Python Isolate で実行し bit-exact 検証
結果	☑15/15 PASS(bit-exact 完全一致)

0. 結論

PoC 1(Any)に続き、PoC 2 でも § 13 で reject されていた 3 種の強い動的構文が Isolate で Python = Rust+CPython-Isolate の SHA-256 完全一致で動作することを 15 件で実機実証。§ 14 主張「動的 Python を Rust で再発明せず、CPython に隔離実行で bit-exact 保つ」が、最も困難視された領域でも成立することを確認しました。

1. 検証カテゴリと結果

A. `*args/**kwargs` 任意展開(5 件)

#	sample	検証構文	SHA(冒頭)	バイト
01	<code>01_args_basic.py</code>	<code>def f(*args) + sum(args)</code>	<code>f6d7a18d69cff172...</code>	20
02	<code>02_kwargs_basic.py</code>	<code>def f(**kwargs) + sorted iter</code>	<code>1f41ddac57b294c1...</code>	38
03	<code>03_mixed_args_kwargs.py</code>	<code>prefix + *args + **kwargs</code>	<code>61987d1210bbb62d...</code>	81
04				18

#	sample	検証構文	SHA(冒頭)	バイト
	04_starargs_unpack.py	呼出側 f(*vals) / f(**dict) / f(*a, **b)	faa665043d9aea03...	
05	05_args_forwarding.py	variadic forwarder def f(*a, **k): return base(*a, **k)	1343057e852a96c0...	9

B. 動的 getattr/setattr/hasattr (第2引数が変数、§ 13 NON_LITERAL)(5 件)

#	sample	検証構文	SHA	バイト
06	06_getattr_variable.py	for name in [...]: getattr(b, name)	881e4253140eb15a...	29
07	07_setattr_variable.py	for name, v in config: setattr(b, name, v)	b03f457d620fca65...	15
08	08_dynamic_attr_chain.py	linked-list walk via getattr(node, "next") 動的 chain	084c799cd551dd1d...	3
09	09_dispatch_table.py	setattr in __init__ + [getattr(h, q) for q in queries]	300538e65a32a39e...	23
10	10_hasattr_dynamic.py	hasattr(b, n) で variable n の存 在チェック	2ff8991d13104c5c...	28

C. Runtime monkey patching(5 件)

#	sample	検証構文	SHA
11	11_class_method_replace.py	Counter.value = faster_value でクラスメソッド差し替え	e2c39a2c6d0b78d6

#	sample	検証構文	SHA
12	12_conditional_rebind.py	impl = fast if cond else slow の動的選択 (両方の分岐実行)	d40536360c7a45e8
13	13_module_attr_patch.py	json.dumps = patched_dumps (stdlib モジュール属性書換)	da75aa47f89954ba
14	14_method_swap.py	Greeter.say 書換 → g.shout() が伝播して別出力	2b2bda44e6ac32fd
15	15_decorator_monkeypatch.py	Service.compute = add_log(Service.compute) (decorator+monkey, 内部で *args/**kwargs 転送)	c79b784b5cc00964

2. 実行結果(出力ログ)

```

=== build runner ===
  Compiling poc2_dynamic v0.1.0
  Finished `release` profile [optimized] target(s) in 7.87s
binary: 338568 bytes

=== run + bit-exact compare (15 samples) ===
01_args_basic          PASS  sha=f6d7a18d69cff172... (20B)
02_kwargs_basic        PASS  sha=1f41ddac57b294c1... (38B)
03_mixed_args_kwargs   PASS  sha=61987d1210bbb62d... (81B)
04_starargs_unpack     PASS  sha=faa665043d9aea03... (18B)
05_args_forwarding     PASS  sha=1343057e852a96c0... (9B)
06_getattr_variable    PASS  sha=881e4253140eb15a... (29B)
07_setattr_variable    PASS  sha=b03f457d620fca65... (15B)
08_dynamic_attr_chain  PASS  sha=084c799cd551dd1d... (3B)
09_dispatch_table      PASS  sha=300538e65a32a39e... (23B)
10_hasattr_dynamic     PASS  sha=2ff8991d13104c5c... (28B)
11_class_method_replace PASS  sha=e2c39a2c6d0b78d6... (27B)
12_conditional_rebind  PASS  sha=d40536360c7a45e8... (10B)
13_module_attr_patch   PASS  sha=da75aa47f89954ba... (45B)
14_method_swap         PASS  sha=2b2bda44e6ac32fd... (14B)
15_decorator_monkeypatch PASS  sha=c79b784b5cc00964... (20B)

=== summary: PASS=15 / 15 ===

```

3. binary 実測

軸	値
Rust binary	338,568 bytes(~331 KB)
依存リンク	libpython3.12.so.1.0 (動的)、libc.so.6、libm.so.6

軸	値
Cargo deps	pyo3 = 0.22 + auto-initialize のみ
build 時間	初回 ~30 秒、増分 7-8 秒
Runner ソース	PoC 1 と同型(20 行強の汎用 Isolate executor)

15 samples が 1 つの binary で全部走る。PoC 1 と同サイズで横展開コストはゼロに近い。

4. 検証された § 14 要素

§ 14 要素	検証
任意 <code>*args</code> の Isolate 委譲	☑ #01, #03, #04, #05
任意 <code>**kwargs</code> の Isolate 委譲	☑ #02, #03, #04, #05, #15
変数引数 <code>getattr</code> / <code>setattr</code> / <code>hasattr</code> の Isolate 委譲	☑ #06 ~ #10
Runtime class method 書換	☑ #11, #14
Runtime stdlib module attribute 書換	☑ #13(json.dumps を実行時差し替え)
条件分岐による関数選択	☑ #12(両分岐実行で結果が変わることまで)
Decorator による monkey patch	☑ #15(<code>*args</code> / <code>**kwargs</code> forwarding と複合)
全 sample で stdout SHA-256 三者一致(Py = Isolate)	☑ 15/15

5. 重要な観察

- § 13 の 3 種 reject カテゴリ全部が Isolate で動く: NON_LITERAL(変数 `getattr/setattr`)、DEFER_HARD(`*args`/`**kwargs`、`eval/exec` は今回入れていないが同じ路線)、PHASE_2_FUTURE(monkey patching) — すべて Isolate に流して 1 件もズレずに bit-exact。
- decorator + monkey patch + kwargs forwarding の複合(#15)まで素直に動く:
「`Service.compute = add_log(Service.compute)`」のように、3 つの動的機能が組み合わさるパターンでも、Isolate は単に CPython にそのまま投げただけなので、複合の複雑さは関知しない。これは「動的の組み合わせ爆発を Isolate に押し込めば管理しなくてよい」という設計上の決定的アドバンテージ。
- stdlib モジュール書換(#13 json.dumps)も問題なし: PyO3 経由で起動した CPython は通常の json モジュールを持ち、ユーザコードからの書換にも普通に追従する。Isolate は別 Python 環境を持たず、同一プロセス内の CPython を使うため stdlib も含めて完全に動く。
- binary は依然 ~338 KB: 同梱バイナリは PoC 1 と変わらず。サンプル数が 3 倍に増えても runner 自体は変わらないため当然だが、「Hybrid Isolate を入れるコストは固定」ということを再確認できる。

5. CPython の `*args/**kwargs` 自体が「Python のまま」決定論的: PYTHONHASHSEED=0 + sorted iter があれば、kwargs dict の順序が定まる。これは spec § 14.3 canonical encoding の sorted-key 規約とも整合する (boundary 通過時に sorted 化すれば確定)。

6. 配布物

```
engine/poc2_dynamic/
├─ Cargo.toml                pyo3 0.22 + auto-initialize
├─ src/main.rs               汎用 Isolate runner (PoC 1 と同型)
├─ samples/
│  ├─ 01_args_basic.py       *args          (DEFER_HARD)
│  ├─ 02_kwargs_basic.py     **kwargs        (DEFER_HARD)
│  ├─ 03_mixed_args_kwargs.py *args/**kwargs (DEFER_HARD)
│  ├─ 04_starargs_unpack.py   call-site unpack
│  ├─ 05_args_forwarding.py   variadic forwarder
│  ├─ 06_getattr_variable.py  getattr(b, name) variable (NON_LITERAL)
│  ├─ 07_setattr_variable.py  setattr(b, name, v) variable
│  ├─ 08_dynamic_attr_chain.py linked-list via getattr
│  ├─ 09_dispatch_table.py    setattr in __init__ + dynamic getattr
│  ├─ 10_hasattr_dynamic.py   hasattr(b, n) variable
│  ├─ 11_class_method_replace.py Class.method = new (PHASE_2_FUTURE)
│  ├─ 12_conditional_rebind.py impl = fast if cond else slow
│  ├─ 13_module_attr_patch.py json.dumps = patched (stdlib monkey)
│  ├─ 14_method_swap.py       Class.dependee = new; caller picks up
│  └─ 15_decorator_monkeypatch.py decorator + monkey + kwargs forwarding
├─ generate_samples.sh        15 sample 一括生成
├─ run_bit_exact.sh           ビルド → 15件実行 → SHA-256 比較
└─ target/release/poc2_isolate 338,568 B Isolate runner
```

7. § 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	<code>*args/**kwargs</code> + 動的 getattr/setattr + monkey patching 各 5 = 15 sample	完了(本書)
PoC 3	partition(関数単位)の自動振り分 け実装、混在サンプル 10 件	次
PoC 4	musl-libm pin、cross-platform 検証(x86_64 / aarch64 / WASI)	
PoC 5	RustPython 版(Light tier)で同一 サンプル実行、差分計測	

PoC 2 完了。§ 13 で reject だった 3 種の強い動的構文(`*args/**kwargs`、変数引数 `getattr/setattr`、`runtime monkey patching`)が、各 5 sample 全件で Hybrid Isolate 内 bit-exact 動作。PoC 1(`Any`)と合わせて 20/20 で § 14 の根幹実証。PoC 3 へ進行可。