PR #34269 完整报告

sgl-project/sglang

config: state the bag contract as what resolution produced, and the skill rule that goes with it

合并时间：2026-08-15 15:40

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34269

执行摘要

- 一句话：重写 bag 契约测试与 skill 规则，为 step-12 配置重构铺路
- 推荐动作：值得精读。测试部分展示了如何写出不空洞的契约测试：独立 reference（never-published sibling）、raw-differs guard（只比较 resolution 确实写入且偏离默认值的叶子）、passthrough 与 faithfulness 拆分、CI retry 与进程状态泄漏的交互处理，这套方法论可复用到其他配置 / 状态系统的契约测试。SKILL.md 的“has published gate 是刻意设计”与“debt means a decision, not automatically a bag read”两处决策也值得留意。建议同时关注未解决的 KT factory 评论在 step-12 落地时的处置，并与 #34094、#34913 联动阅读。

功能与动机

PR body 明确指出：test_bag_values_match_server_args asserted bag == field，这在今天成立只是因为构造时原地解析；一旦 step-12 让记录保持 raw，该断言会“对每个 resolution 填充的字段按设计失败”。因此需要把契约改写为“bag 承载 resolution 产生的结果”这一能够在翻转后存活的表述，并用测试把“有效值最终只存在于 bag”这一信号钉死。同时 SKILL.md 原有规则“除非字段是运行时改写的，否则不要重写 parameter 读取”对对象正确、对字段不正确：server_args.page_size 在 runner 持有的构造函数里一旦记录保持 raw，将读到 raw 的 pre-resolution 值，需要把这种情形显式命名为债务并给出处置规则。

实现拆解

1. 重写契约测试（test/registered/unit/test_runtime_context_config_bags.py）：将 test_bag_values_match_server_args 改写为 test_the_bags_carry_what_resolution_produced。测试通过 _resolve_published_and_sibling 构造同一份 raw 输入的两次独立解析：第一次 publish 到 runtime context，第二次解析出从未发布过的 sibling 作为 reference，断言 bag 叶子 == reference 叶子，即“bag 等于 resolution 产生什么”，避免 resolved_server_args_dict() 那种读回 vars(server_args) 的自复制式比较。采样只保留 resolution 在两种 CI 设备形态（CUDA host 与 CPU-only runner）上都会写入的四个叶子（attention_backend、page_size、chunked_prefill_size、mem_fraction_static），每个叶子带 raw-differs guard（assertIsNot(default, MISSING) + assertNotEqual(reference, default)），保证比较的是 resolution 确实写过、且偏离 dataclass 默认值的有效值。

2. 保留 tripwire 并拆分 passthrough 冒烟：bag.page_size == sa.page_size 一行保留在 faithfulness 测试末尾并标注为 step-12 tripwire；原测试中 host、hicache_ratio、moe_runner_backend、model_path 等 identity 叶子被拆分到 test_passthrough_leaves_project_into_their_namespaces，其 docstring 明确只声明“publish 把未改动的字段投影进各命名空间”，不声称 resolution faithfulness，避免稀释信号。
3. 双解析的过程状态管理：_resolve_published_and_sibling 创建临时 mini Llama 配置目录（走真实 resolution 管线，绕开 dummy-model 边界提前返回的捷径），快照 os.environ 与沿 MRO 遍历收集的 EnvField._set_to_none 标志，第一次 resolve + publish 后通过 restore_process_state 恢复进程状态，再 resolve 出 sibling；addCleanup 注册临时目录清理与状态恢复。
4. 禁用 CI retry 以保护双解析语义：覆写 TestConfigBags._callTestMethod 直接调用 unittest.TestCase._callTestMethod。原因是 CustomTestCase 在 CI 会重试一次，而 addCleanup 只在最后一次尝试后运行，重试会带着第一次尝试泄漏的进程状态重新进入双解析用例，破坏 sibling 基于纯净快照的假设；该模式与既有 dual-resolve harness（test_resolution_is_reproducible、test_supplied_instance_exposure_ratchet）保持一致。
5. 更新 SKILL.md 规则（.claude/skills/sglang-runtime-context/SKILL.md）：whole-object 规则增加“字段是 resolution 填充的、且调用方运行在已 publish 的进程中”这一 step-12 债务分支；明确“债务意味着做决定，而不是自动改成 bag 读取”，列出 bag 读取、runner stamp、构造函数参数三种 disposition，并保留 per-instance 边界豁免（多 Engine 站点不得因 resolution 填充字段而变成进程级 bag 读取）；补充 linear_attn_backends 作为 per-runner 选择应保持 bag 之外的例子；明确两种刻意保持参数形式的形状（resolution 管线以 resolved_view 调用的 helper、契约是“从交给你的 record 构建 X”的 factory）。
6. 测试与配套说明：无生产源码、schema 或部署配置改动；测试依赖 test_resolution_is_reproducible（reproducibility 契约）与 test_supplied_instance_exposure_ratchet.py（新读取写法会失败）作为配套 guard。

关键文件：

- test/registered/unit/test_runtime_context_config_bags.py（模块 配置契约；类别 test；类型 test-coverage；符号 _callTestMethod, test_the_bags_carry_what_resolution_produced, test_passthrough_leaves_project_into_their_namespaces, _resolve_published_and_sibling）：本 PR 核心：把 bag 契约从“bag == 已发布字段”改写为“bag == resolution 产生的结果”，引入独立 sibling 双解析、raw-differs guard、passthrough 拆分与禁用 CI retry 四个关键设计，是 step-12 重构的翻转检测器。
- .claude/skills/sglang-runtime-context/SKILL.md（模块 技能文档；类别 docs；类型 documentation）：把 whole-object 传递规则从“除非运行时改写否则保持参数读取”扩展出 step-12 债务分支，给出 disposition 决策框架（bag/runner stamp/ 构造函数参数）与 per-instance 豁免，直接影响 AI 辅助开发对配置读取的改写行为。

关键符号：_callTestMethod, test_the_bags_carry_what_resolution_produced, test_passthrough_leaves_project_into_their_namespaces, _resolve_published_and_sibling, restore_process_state, resolve

关键源码片段

test/registered/unit/test_runtime_context_config_bags.py

本 PR 核心：把 bag 契约从“bag == 已发布字段”改写为“bag == resolution 产生的结果”，引入独立 sibling 双解析、raw-differs guard、passthrough 拆分与禁用 CI retry 四个关键设计，是 step-12 重构的翻转检测器。

```
class TestConfigBags(CustomTestCase):
    def _callTestMethod(self, method):
        # CustomTestCase 在 CI 会重试一次，但 addCleanup 只在最后一次尝试之后运行；
        # 双解析用例若重试，会带着第一次尝试泄漏的进程状态重新进入，
        # 正好破坏 sibling 辅助函数依赖的“纯净快照”前提，所以直接跳过重试。
        unittest.TestCase._callTestMethod(self, method)

    def test_the_bags_carry_what_resolution_produced(self):
        """bag 契约：每个采样叶子都必须是 resolution 产生（resolve 后）的值。

        dummy-model 捷径无法满足两个要求：记录必须走完整 resolution 管线
        （dummy 路径在 dummy-model 边界提前返回，采样叶子仍是 raw，raw==raw
        会空洞通过）；reference 必须是同一份 raw 输入的独立解析 —— 一个从未
        publish 过的 sibling，在恢复第一次解析可能写下的进程状态后再解析，
        断言才是 "bag == resolution 产生什么"，而非 "bag == publish 复制来源"。
        Reproducibility（test_resolution_is_reproducible）授权 sibling 充当
        pipeline 输出的替身；raw-differs guard 保证每次相等比较的都是 resolution
        确实写入过的值。step-12 落地后记录保持 raw，届时本断言会为每个
        resolution 写入的叶子开始失败 —— 这正是 bag 成为有效值唯一归属的信号。
        """
        import dataclasses

        sa, reference = self._resolve_published_and_sibling()
        defaults = {f.name: f.default for f in dataclasses.fields(ServerArgs)}
        # 只采样 resolution 在两种 CI 设备形态（CUDA host 与 CPU-only runner）
        # 上都会写入、且起点为 None 默认值的叶子。
        sampled = (
            (lambda: rc.get_exec().kernel.attention_backend, "attention_backend"),
            (lambda: rc.get_schedule().page_size, "page_size"),
            (lambda: rc.get_schedule().chunked_prefill_size, "chunked_prefill_size"),
            (lambda: rc.get_schedule().mem_fraction_static, "mem_fraction_static"),
        )
        for accessor, leaf in sampled:
            with self.subTest(leaf=leaf):
                # raw-differs guard：仍停留在默认值（或无默认值）的叶子证明不了任何事。
                self.assertIsNot(defaults[leaf], dataclasses.MISSING)
                self.assertNotEqual(getattr(reference, leaf), defaults[leaf])
                self.assertEqual(accessor(), getattr(reference, leaf))
        # tripwire：今天 record 与 bag 一致；step-12 翻转后这一行会先失败，
        # 标志 bag 成为有效值的唯一归属。
        self.assertEqual(rc.get_schedule().page_size, sa.page_size)
```

```
def _resolve_published_and_sibling(self):
    """在同一个纯净进程状态下把同一份 raw 输入解析两次。

    第一次解析后 publish；恢复进程状态；第二次解析出从未发布过的 sibling
    作为独立 reference。resolution 可能写入 os.environ 之外的状态
    （multimodal transport 的 sticky env、EnvField 描述符 set() 翻转的
    _set_to_none），所以快照要覆盖两者；沿 MRO 遍历以避免漏掉基类字段。
    """
    import json
    import os
    import shutil
    import tempfile

    from sglang.srt.environ import EnvField, envs

    config_dir = tempfile.mkdtemp(prefix="bag_contract_")
    self.addCleanup(shutil.rmtree, config_dir, ignore_errors=True)
    with open(os.path.join(config_dir, "config.json"), "w") as handle:
        json.dump({...真实 mini Llama 配置字段...}, handle)

    # 快照进程状态：os.environ 之外还有 EnvField 描述符的 _set_to_none 标志。
    env_fields = {}
    for klass in reversed(type(envs).__mro__):
        for name, field in vars(klass).items():
            if isinstance(field, EnvField):
                env_fields[name] = field
    environ_before = dict(os.environ)
    none_flags_before = {
        name: field._set_to_none for name, field in env_fields.items()
    }

    def restore_process_state():
        os.environ.clear()
        os.environ.update(environ_before)
        for name, was_none in none_flags_before.items():
            getattr(type(envs), name)._set_to_none = was_none

    self.addCleanup(restore_process_state)

    def resolve():
        return ServerArgs(model_path=config_dir, device="cuda", random_seed=42)

    sa = resolve()
    rc.publish(sa, role="scheduler")
    restore_process_state()
    return sa, resolve()
```

评论区精华

Codex（P2）：resolved_server_args_dict() 不读 namespace bags，只是拷贝 vars(self.server_args)，所以新断言仍然是 bag == sa.field 的变体，没有验证"what resolution produced"。 ch-wan：对——reference 必须是同一 raw 输入的独立解析，一个从未 publish 的 sibling；reproducibility（#34094）授权它作为 pipeline 输出的替身。

Codex（P2）：两个实例都用 model_path="dummy"，resolution 管线在 dummy-model 边界提前返回，page_size 仍为 None，raw==raw 会空洞通过。 ch-wan：改为真实 mini config 走完整管线，raw-differs guard 要求采样叶子偏离 dataclass 默认值。

ch-wan（自查）：model_path 没有 dataclass default，f.default is dataclasses.MISSING，任何路径都 != MISSING，计数阈值可被单个真实写入的叶子满足，guard 比看起来弱。 ch-wan：阈值改为 per-leaf guard，采样只保留 resolution 确实写入的叶子。

ch-wan（自查）：host、hicache_ratio、moe_runner_backend 在该 mini 配置上不被 resolution 移动，bag == sibling 只证明投影，稀释 faithfulness 信号。 ch-wan：拆分 test_passthrough_leaves_project_into_their_namespaces，docstring 只声明投影。

Codex（P2）：SKILL.md 规则限定"已 publish 的进程"会漏掉非 publishing 消费面，如 DetokenizerManager 读取 resolution 填充的 tokenizer_path 等。 ch-wan：ratchet 已按"reads a field resolution writes"覆盖、不按 publish 判定；"has published" gate 是刻意设计，阻止在 bag 不存在的进程里把读取改成 bag 读取。

Codex（P2）：create_kt_config_from_server_args 作为 parameter-form factory 例外在 KT offloading 下不安全——get_server_args() 拿到 raw record 后 chunked_prefill_size 为 None，KT wrapper 会用错误 chunk size。 本 PR 讨论中未见作者公开回应，文档最终仍保留该例外，属遗留疑虑。

- reference 必须独立于被发布实例，resolved_server_args_dict 不够 (testing): 改为独立解析的 never-published sibling，并在两次解析间恢复进程状态，断言变成"bag == what resolution produces"。
- dummy-model 路径下 raw==raw 空洞通过 (testing): 改用真实 mini Llama config 走完整 resolution 管线，并通过 raw-differs guard 保证采样叶子偏离 dataclass 默认值。
- model_path 无 dataclass 默认值造成 MISSING freebie (correctness): 数值阈值移除，改为 per-leaf guard；采样只保留 resolution 在两种 CI 设备形态上都写入的 4 个叶子，构造输入永不入样。
- passthrough 叶子稀释 faithfulness 信号 (testing): 拆分 test_passthrough_leaves_project_into_their_namespaces，docstring 明确只声明投影，不声称 resolution faithfulness。
- SKILL.md 规则措辞的四点问题 (documentation): 全部重写：debt 读作"pick a disposition"，补 per-mode attention 对与 gpu_id 非 bag 处置案例，显式豁免 per-instance 边界。
- CI retry 与泄漏进程状态的交互 (testing): 覆写 _callTestMethod 直接调用 unittest.TestCase._callTestMethod 禁用重试，与其他 dual-resolve harness 一致。
- 非 publishing 消费者的 resolution 债务是否被规则覆盖 (design): 作者回应：ratchet 按"reads a field resolution writes"覆盖、不按 publish 判定，无洞；skill 规则的 "has

published" gate 是刻意设计，阻止 bag 不存在的进程改为 bag 读取。

- KT config factory 参数形式例外在 raw record 下不安全 (correctness): 本 PR 讨论中未见作者公开回应，最终文档仍保留该 factory 例外，属遗留疑虑，需在 step-12 落地时处置。

风险与影响

- 风险：
  1. 进程状态快照依赖手工维护：_resolve_published_and_sibling 手工快照 os.environ 与沿 MRO 遍历的 EnvField._set_to_none，若未来新增或重命名 EnvField 而测试未同步，存在测试间污染或误报风险。
  2. 禁用 CI retry 降低容错：_callTestMethod 跳过 CustomTestCase 的 CI 重试，该用例在 CI 上若偶发失败将直接暴露；这是作者基于双解析语义的刻意取舍，但仍需关注其稳定性。
  3. KT factory 例外遗留疑虑：Codex 指出 step-12 后 create_kt_config_from_server_args 若收到 raw record，chunked_prefill_size 会变 None，导致 KT wrapper 初始化错误；文档仍将其列为刻意保持参数形式的 factory，落地时可能成为隐患。
  4. tripwire 与未来重构强耦合：bag.page_size == sa.page_size 一行在 step-12 落地后会按设计失败，若团队遗忘其语义可能误判为回归。
  5. CI 观察：PR 的 Extra CI run（#31872546294）标记为失败，材料未给出失败原因。
  6. 无生产代码变更，运行时回归风险为零。 - 影响：对用户无运行时影响（纯测试与文档变更）；对团队的影响集中在开发流程与配置系统演进：SKILL.md 是 Claude Code 的 skill，直接约束 AI 辅助编码时对 supplied-instance 读取的改写规则；contract 测试成为 step-12（records stay raw）重构的翻转检测器，落地时每个 resolution 写入的叶子都会触发预期失败，提示开发者把有效值迁移到 bag。该 PR 确立的双解析 harness 模式（独立 sibling + 过程状态恢复 + 禁用 retry）已与 test_resolution_is_reproducible、test_supplied_instance_exposure_ratchet 形成同族体系，影响后续所有配置契约测试的写法。 - 风险标记：测试与未来重构强耦合，CI 重试被禁用，进程状态快照手工维护，KT factory 例外未决

关联脉络

- PR #34913 [CI] Move the static ratchets back to CPU unit tests: 同族 ratchet/ 契约测试体系：本 PR 引用的 test_supplied_instance_exposure_ratchet.py 与该 PR 涉及的静态 ratchet 检查同属配置系统契约防线，且两者都改动 .claude/skills/sglang-runtime-context/SKILL.md。
- PR #34094 test_resolution_is_reproducible（讨论中引用的 reproducibility 契约）：本 PR 中"独立 sibling 可充当 pipeline 输出替身"的合法性完全依赖 resolution 可复现契约；review 讨论中作者明确引用该测试作为依据。