# PR #36622 完整报告

sgl-project/sglang

config: the record is not an object that gets passed around

合并时间：2026-08-28 03:57

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36622

## 执行摘要

- 一句话：清理 ServerArgs 传递链，消除死参数与非字段属性
- 推荐动作：值得精读。重点看三点：一是两个 AST ratchet 测试的设计（如何用约 80 行代码锁定整个包的配置契约，并内置 ' 扫描自身失效 ' 的自检，防止测试变成摆设）；二是 override_server_args 的权衡记录——PR body 详细记录了两种被否定方案各自的问题（handler 进程级副作用 vs 直接消费者读到 dummy 值），是 ' 为什么当前形状是对的 ' 的极佳范例；三是级联删除脚本误删活参数的事故复盘，对任何做机械性大范围重构的人都有警示意义。建议结合同系列 gc-p1 至 gc-p4 一起阅读，才能看到 ' 声明 / 投影 / 读取 ' 三层契约的全貌。

## 功能与动机

PR body 明确这是五连 PR 的第 5 个，叠加在 raw-input ServerArgs 工作（#36250–#36255）之上，按 ' 必须如何被阅读 ' 分组。核心论点是：函数不应接收它从未读取的记录——'A server_args parameter that the body never names keeps a reference to the whole record alive across a call boundary, and it reads as an invitation: the next person to need one value takes it off the parameter that is already there'，且删除会级联：调用者只是为了转发才持有记录。同时 ' 记录不增长投影看不见的属性 '：moe_ep_size 仅被日志读取、grpc_worker_threads 是环境变量派生结果却游离在投影之外、model_config 缓存因公开命名迫使只读保护开例外，这三点积累正是因为命名空间覆盖、投影和读取 ratchet 都只走字段。

## 实现拆解

1. 级联删除从未读取的 server_args 形参（推进到不动点）：从 release_req → retract_all → 调度调用点、build_kv_host_pool → build_kv_only_group → build_hicache_draft_sidecars 等链路逐个删除函数体从未引用的 server_args 参数，初查 7 个、级联后共清理 13 个模块级函数签名。涉及 schedule_batch.py（release_req、retract_all、retract_decode）、scheduler.py（update_running_batch、pause_generation 调用点）、hybrid_pool_assembler.py（build_kv_host_pool、build_kv_only_group、build_hybrid_swa_group 等 KV host pool 构建族）、load_model_utils.py（load_kv_cache_scales）、tokenizer_manager.py（get_processor_wrapper）等。类方法中具有契约意义的参数保留（BaseKVManager.init、RadixCacheCpp.init、插件钩子 SRTPlatform.apply_server_args_defaults /

CustomSpecAlgo.handle_server_args、StackStrategy.build /_MiniMaxSparseStrategy.build 系列），仅对 _DetailSinglePassGatherer.init、TokenizerMetricsCollector.init、RayDataParallelController.launch_tensor_parallel_group 三个不明确的方法豁免。同时移除 schedule_batch.py 中对 ServerArgs 的直接 import，并修正 elastic_ep.py 经由它间接 re-import 的问题。

2. 修复发布顺序并记录一次误删事故：init_multi_tokenizer 原先从共享内存反序列化记录后、在 publish 之前断言 server_args.api_key，现改为先发布、再断言 get_serving().api_key（两者之间无写入，顺序可交换）。级联脚本曾按行删除参数，误删 load_kv_cache_scales 一行中 model、kv_cache_dtype 两个活参数，通过 ruff --select F821 与 origin/main 对比发现 4 个未定义名并恢复。
3. 收紧记录契约（非字段属性清零）：删除仅被日志引用的 moe_ep_size（日志改读 cfg.dwdp_size）；把环境变量派生的 grpc_worker_threads 提升为正式字段（Arg(no_cli=True)、NS("serving")），由 _handle_deprecated_args 经 _declare 声明，校验改读投影视图 cfg；get_model_config() 缓存由公开命名 model_config 改为私有 _model_config，从而撤销只读保护中 _CACHE_SLOTS 的命名例外；resolve_once 中设置 _resolution_failed 由 object.__setattr__改为普通赋值。
4. 两条防回归契约测试：新增 test_dead_server_args_parameter_ratchet.py（AST 扫描全包模块级函数，基线锁定为 0，并内置 scanned > 50 的扫描有效性自检）与 test_no_public_non_field_slot.py（AST 扫描 ServerArgs 类内所有 self 写入，要求公开属性必为 dataclass 字段，下划线私有名除外）。
5. 有意保留的例外：RuntimeContext.override_server_args 的 ' 声明 + 写回 ' 双重写保持不变，因为两种替代方案都被论证为更差——构造器传值会触发 handler 的进程级副作用（configure_media_url_security、DG_* 环境变量写入、model_path 触发网络拉取）；只声明不写回则 7 个注册测试文件会读到 dummy 默认值。PR body 完整记录了这两次被否定的重写及其理由。

关键文件：

- test/registered/unit/test_dead_server_args_parameter_ratchet.py（模块 配置护栏；类别 test；类型 test-coverage；符号 _dead_parameters, TestNoDeadServerArgsParameter, test_no_module_level_function_takes_a_record_it_ignores）：新增的防回归测试核心：用 AST 扫描整个 sglang 包，锁定 ' 模块级函数不得携带从未引用的 server_args 参数 ' 为基线 0，并内置 scanned > 50 的自检防止扫描本身失效。这是本 PR 契约主张的可执行化，也是后续所有配置清理的护栏。
- test/registered/unit/server_args/test_no_public_non_field_slot.py（模块 配置契约；类别 test；类型 test-coverage；符号 _self_written_attributes, TestNoPublicNonFieldSlot, test_every_public_attribute_is_a_field）：契约测试的另一半：AST 扫描 ServerArgs 类内所有 self 写入（含 object.__setattr__两种写法），要求每个公开属性都必须是 dataclass 字段；这正是本 PR 清理 moe_ep_size、grpc_worker_threads、model_config 三个历史漏网的依据。
- python/sglang/srt/managers/schedule_batch.py（模块 批调度；类别 source；类型 core-logic；符号 retract_decode, release_req, retract_all）：调度核心路径：release_req、retract_all、retract_decode 三个函数删除 server_args 形参，OOM 回收

与 decode 分离路径的签名变化以这里为源头，scheduler.py 调用点同步联动。

- python/sglang/srt/server_args.py（模块 配置记录；类别 source；类型 core-logic；符号 grpc_worker_threads, _handle_deprecated_args, get_model_config, resolve_once）：配置记录契约的核心文件：新增 grpc_worker_threads 正式字段、_handle_deprecated_args 改用 _declare 声明、删除 moe_ep_size、model_config 缓存改名 _model_config、resolve_once 的 setattr 简化，所有 ' 非字段属性归位 ' 的改动都落在这里。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 update_running_batch, pause_generation, get_internal_state）：调度器的调用点联动：update_running_batch 中 batch.retract_decode(self.server_args) 改为无参调用，pause_generation 中 retract_all 调用移除实参，get_internal_state 因 model_config 改名不再需要 pop 该字段。
- python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py（模块 KV 池组装；类别 source；类型 core-logic；符号 build_kv_host_pool, build_kv_only_group, build_hybrid_swa_group, build_full_draft_pools）：KV host pool 构建链的批量签名清理：build_kv_host_pool、build_kv_only_group、build_hybrid_swa_group、build_full_draft_pools 等多个函数删除未使用的 server_args 形参，是级联删除覆盖最广的一个文件。

关键符号：retract_decode, release_req, retract_all, build_kv_host_pool, build_kv_only_group, build_hybrid_swa_group, load_kv_cache_scales, get_processor_wrapper, _handle_deprecated_args, get_model_config, resolve_once, init_multi_tokenizer, _dead_parameters, _self_written_attributes

## 关键源码片段

### test/registered/unit/test_dead_server_args_parameter_ratchet.py

新增的防回归测试核心：用 AST 扫描整个 sglang 包，锁定 ' 模块级函数不得携带从未引用的 server_args 参数 ' 为基线 0，并内置 scanned > 50 的自检防止扫描本身失效。这是本 PR 契约主张的可执行化，也是后续所有配置清理的护栏。

```
# 一个函数不接收它从未读取的记录。
#
# 一个 `server_args` 参数如果函数体内从未引用，就会让整个记录跨调用边界存活，
# 并像一个邀请：下一位需要某个值的人会直接从现成参数上取，而不是思考这个值
# 应该来自哪里。删掉一个通常暴露下一个——调用者只是为了把它传下去才持有记录。
#
# 类方法豁免：基类、重写、或某个策略实现是为其契约携带参数的，单看任何一个
# 方法体都不是证据。本测试只扫描模块级函数。

import ast
import pathlib
import unittest

import sglang
from sglang.test.ci.ci_register import register_cpu_ci
```

```
from sglang.test.test_utils import CustomTestCase

register_cpu_ci(est_time=6, suite="base-a-test-cpu")

_PACKAGE_ROOT = pathlib.Path(next(iter(sglang.__path__)))

# 解析管线负责构建记录，因此那里的参数是主语而非乘客；
# `multimodal_gen` 有另一个同名类，不在本契约内。
_EXCLUDED = ("srt/arg_groups", "srt/server_args.py", "multimodal_gen")

_BASELINE = 0 # 模块级函数中死参数的允许数量，契约要求恒为 0


def _dead_parameters():
    """遍历包内所有 .py 文件，用 AST 找出带 server_args 参数却从不引用它的函数。"""
    found = []
    scanned = 0
    for path in sorted(_PACKAGE_ROOT.rglob("*.py")):
        rel = path.relative_to(_PACKAGE_ROOT).as_posix()
        if rel.startswith(_EXCLUDED):
            continue
        source = path.read_text(encoding="utf-8-sig")
        if "server_args" not in source:
            continue
        scanned += 1
        try:
            tree = ast.parse(source)
        except SyntaxError:
            continue
        for node in tree.body:
            if not isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef)):
                continue
            # 收集位置参数与 keyword-only 参数名
            taken = [a.arg for a in node.args.args] + [
                a.arg for a in node.args.kwonlyargs
            ]
            if "server_args" not in taken:
                continue
            # 检查函数体内是否真的以名字引用过该参数（排除 def 本身）
            named = any(
                isinstance(inner, ast.Name) and inner.id == "server_args"
                for inner in ast.walk(node)
                if inner is not node
            )
            if not named:
                found.append(f"{rel}:{node.lineno} {node.name}")
    return found, scanned

class TestNoDeadServerArgsParameter(CustomTestCase):
```

```
    def test_no_module_level_function_takes_a_record_it_ignores(self):
        found, scanned = _dead_parameters()
        # 先验证扫描没坏：如果全包只有不到 50 个文件提到 server_args，
        # 说明扫描失效而不是树是干净的
        self.assertGreater(
            scanned,
            50,
            f"only {scanned} files mention server_args; the scan is broken, not "
            "the tree",
        )
        self.assertEqual(
            _BASELINE,
            len(found),
            "these functions take `server_args` and never name it; drop the "
            "parameter and the argument at every call site, then check whether "
            f"the caller still needs its own: {found}",
        )


if __name__ == "__main__":
    unittest.main()
```

## test/registered/unit/server_args/test_no_public_non_field_slot.py

契约测试的另一半：AST 扫描 ServerArgs 类内所有 self 写入（含 object.__setattr__两种写法），要求每个公开属性都必须是 dataclass 字段；这正是本 PR 清理 moe_ep_size、grpc_worker_threads、model_config 三个历史漏网的依据。

```
# 记录不增长投影看不见的属性。
#
# 一个公开命名但不是 dataclass 字段的属性，对这里的所有护栏都不可见：
# 命名空间覆盖检查走字段、投影走字段、读取 ratchet 也只看字段读取。
# 历史上积累的三个漏网之鱼就是这样来的——`ModelConfig` 缓存、只有日志行
# 读取的 `moe_ep_size`、以及由一个入口跨边界读取的环境派生 `grpc_worker_threads`。
#
# 下划线开头的名字是记录自己的簿记，可以保留：只读保护的写权限判定就是按
# 这个拼写分类的，私有名本来就已脱离配置层。

import ast
import dataclasses
import pathlib
import unittest

import sglang
from sglang.srt.server_args import ServerArgs
from sglang.test.ci.ci_register import register_cpu_ci
from sglang.test.test_utils import CustomTestCase

register_cpu_ci(est_time=4, suite="base-a-test-cpu")
```

```
def _self_written_attributes() -> set:
    """返回 `ServerArgs` 写在自己身上的属性名，兼容两种拼写方式。"""
    source = (
        pathlib.Path(next(iter(sglang.__path__))) / "srt" / "server_args.py"
    ).read_text(encoding="utf-8-sig")
    tree = ast.parse(source)
    cls = next(
        node
        for node in tree.body
        if isinstance(node, ast.ClassDef) and node.name == "ServerArgs"
    )
    written = set()
    for node in ast.walk(cls):
        # 普通赋值：self.xxx = ...
        if isinstance(node, ast.Assign):
            for target in node.targets:
                if (
                    isinstance(target, ast.Attribute)
                    and isinstance(target.value, ast.Name)
                    and target.value.id == "self"
                ):
                    written.add(target.attr)
        # 显式 setattr：object.__setattr__(self, "xxx", ...)
        if (
            isinstance(node, ast.Call)
            and getattr(node.func, "attr", None) == "__setattr__"
            and getattr(getattr(node.func, "value", None), "id", None) == "object"
            and len(node.args) >= 2
            and isinstance(node.args[1], ast.Constant)
        ):
            written.add(node.args[1].value)
    return written


class TestNoPublicNonFieldSlot(CustomTestCase):
    def test_every_public_attribute_is_a_field(self):
        written = _self_written_attributes()
        # 先自检扫描有效性：至少应发现 5 处 self 写入，否则扫描坏了
        self.assertGreater(
            len(written),
            5,
            f"only {len(written)} self-writes found; the scan is broken, not the "
            "record",
        )
        fields = {field.name for field in dataclasses.fields(ServerArgs)}
        # 公开（无下划线前缀）但非字段的写入即违约
        stray = sorted(
            name for name in written if not name.startswith("_") and name not in fields
        )
```

```
        self.assertEqual(
            [],
            stray,
            "these are written on the record under a public name but are not "
            "fields, so the projection cannot see them and no other guard "
            "watches them: make each a field, or give it the leading underscore "
            f"that says it is the record's own bookkeeping: {stray}",
        )


if __name__ == "__main__":
    unittest.main()
```

## python/sglang/srt/server_args.py

配置记录契约的核心文件：新增 grpc_worker_threads 正式字段、_handle_deprecated_args 改用 _declare 声明、删除 moe_ep_size、model_config 缓存改名 _model_config、resolve_once 的 setattr 简化，所有 ' 非字段属性归位 ' 的改动都落在这里。

```
# server_args.py —— ServerArgs 记录契约收紧示例

# 1) 环境变量派生值改为正式字段：原来挂在记录上的非字段属性对投影、命名空间
# 覆盖检查、读取 ratchet 全部不可见，因此必须成为字段；不暴露 CLI 参数。
# Env-only (SGLANG_GRPC_WORKER_THREADS); a field so the projection sees it.
grpc_worker_threads: A[Optional[int], Arg(no_cli=True), NS("serving")] = None


# 2) handler 内不再直接写 `self.grpc_worker_threads = ...`，而是经 `_declare` 声明，
# 让投影（resolved_dict）与读取 ratchet 都能看到该值。
def _handle_deprecated_args(self):
    ...
    self._declare(
        "_handle_deprecated_args",
        grpc_worker_threads=envs.SGLANG_GRPC_WORKER_THREADS.get(),
    )
    ...
    # 校验改读投影视图 cfg 而不是记录本身：`None` 表示环境变量未设置，跳过校验
    if cfg.grpc_worker_threads is not None and cfg.grpc_worker_threads < 1:
        raise ValueError(
            "SGLANG_GRPC_WORKER_THREADS "
            f"({cfg.grpc_worker_threads}) must be >= 1"
        )


# 3) `get_model_config()` 的缓存由公开命名 `model_config` 改为私有 `_model_config`：
# 公开命名迫使只读保护为它开 `_CACHE_SLOTS` 例外，改私有后按下划线拼写
# 就天然属于记录自身簿记，例外随之撤销。
def get_model_config(self):
    cfg = resolving_view(self)
    from sglang.srt.configs.model_config import ModelConfig
```

```
memo = getattr(self, "_model_config", None)
...
```

## 评论区精华

Codex 自动审查（chatgpt-codex-connector[bot]）在 runtime_context.py 提出两条 P2 建议，均与 PR body 中 ' 有意保留的例外 ' 直接对应：

1. Revalidate divergent values before publishing：override_server_args 在解析完成后再声明调用方原值，会绕过对应校验。例如 override_server_args(grpc_port=50051, grpc_worker_threads=0).install() 先正常校验环境派生值（通常为 4），随后声明 0 并发布，绕开 _handle_deprecated_args 的 grpc_worker_threads >= 1 约束。
2. Preserve override values for direct record consumers：若只把覆盖值声明进 stash 而不写回记录，直接读记录字段的代码会拿到 dummy 默认值。具体案例是 test_real_path_loads_table 覆盖 speculative_dspark_sps_table_path，而 build_sps_cost_table 直接读 server_args.speculative_dspark_sps_table_path，会得到 None 而返回未初始化表。 两条建议在合并时均未修改——PR 作者已在 body 中详细论证当前 ' 声明 + 写回 ' 是三种方案里唯一没有问题的形态，并把 ' 关闭它需要什么 ' 记录为待办。

- override_server_args 晚声明绕过数值校验 (correctness): PR body 第 2 节已完整记录该权衡并有意保留现状（'The current shape has neither problem'），' 关闭它需要什么 ' 被记录为待办；合并时未修改。
- 只声明不写回会让直接读记录字段的消费者拿到 dummy 值 (correctness): 这正是 PR 保留 declare + write 双重写的原因：声明让投影与 bags 看到值，写回让 handed record 携带值；两种被否定的重写方案及其各自问题已在 PR body 中记录。

## 风险与影响

- 风险：
  1. 调度核心路径签名变更：retract_decode、release_req、retract_all 是 OOM 回收与 decode 分离（decode disaggregation）路径的关键函数，scheduler.py 的 update_running_batch 调用点同步修改，回归风险集中于此；好在调用点与形参是机械联动删除，AST ratchet 测试可覆盖新增死参数，但无法验证语义。
  2. 配置校验可被绕过：override_server_args 的晚声明机制可发布未经验证的值（如 grpc_worker_threads=0），Codex 已指出，属于已记录的已知缺口，未来若有人依赖该校验应重新收紧。
  3. 机械重写误删风险：级联删除脚本曾按行删除导致 load_kv_cache_scales 两个活参数被误删，说明按行编辑的自动化重写有真实事故记录；本次靠 ruff F821 对比基线救回，后续类似清理应把该检查纳入流程。
  4. 契约测试覆盖有限：两个 AST ratchet 只扫描模块级函数和 ServerArgs 自身的 self 写入，类方法被显式豁免（契约原因），_DetailSinglePassGatherer 等三个类方法留下长尾；non-field 属性的读取方没有对应的 ' 读取 ratchet'，未来仍可能在类内部积累。 - 影响：对调用方：13 个函数签名变化，所有调用点同步移除实参，涉及调度器、KV pool 组装、模型加载、tokenizer 多处模块，属于破坏性 API 变更但仅在仓库内部。对

系统：ServerArgs 记录契约收紧——所有写入记录的值要么是 dataclass 字段（投影可见）要么是下划线私有名，配置读取 / 校验 / 投影三条通道首次对齐。对团队：新增两个廉价的 AST 静态测试作为配置架构的护栏，后续任何新增 ' 死参数 ' 或 ' 公开非字段属性 ' 都会直接 CI 失败，降低了配置体系后续演进（override、arg group、投影）的维护成本。用户在功能上无感知。 - 风险标记：调度核心路径签名变更，配置校验可被绕过，机械重写曾误删活参数，契约测试覆盖有限

## 关联脉络

- PR #36623 CI vehicle PR for the gc-p1..p5 series: PR body 说明整个五连 PR 系列的 CI 跑在独立的 vehicle PR 上，其分支位于本 PR 之上一个占位 commit。#36622 的合并依赖该 PR 提供 CI 验证。
- PR #36676 Refactor server_args constants and layout: 同期对 server_args.py 的布局与常量组织重构，与本 PR 同属 ServerArgs 配置体系整理，改动同一文件、同一配置模块。
- PR #36681 Move server args config parser under utils: 将配置解析器移入 utils 目录，与 raw-input ServerArgs 工作线同源，后续维护同一批 server_args 相关测试文件（test/registered/unit/server_args/）。