PR #36240 完整报告

sgl-project/sglang

[CI] Stop the config ratchets re-parsing the package on every scan

合并时间：2026-08-25 09:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36240

执行摘要

- 一句话：8 个配置棘轮扫描测试加预过滤与缓存，CI 提速数倍
- 推荐动作：值得精读。核心价值不在功能交付，而在两个可复用模式：基于“字面标识符必然出现于源码文本”的预过滤，以及“输出等价断言 + 负对照注入”的验证方法。对 CI 维护者和静态分析型测试的作者很有借鉴意义；对只关心推理服务的工程师则优先级中等，因为不涉及任何运行时路径。实现中有两点值得学习：刻意不做过度收紧的边界判断，以及用负对照证明性能优化没有牺牲查全率。

功能与动机

base-a-test-cpu 分片 7 在 run 32786070189 上 15 分钟超时；调查发现这不是单个文件的问题，而是家族模式：config / runtime-context ratchets 每个都要遍历整个 python/sglang 包（3467 个文件、1.31 MB AST），有的还重复多次，累计 507 秒几乎占满一个分片的预算。同时 ast.parse 是绝对大头（2.75 秒 warm / 7.04 秒 cold），而 read_text 全部文件只要 0.085 秒，所以任何守卫必须放在 parse 之前。

实现拆解

1. 瓶颈定位：对 8 个 ratchet 逐个计时，确认反复 ast.parse 全包是主因（server_args.py 约 10745 行、24 ms/ 次，在 6 个独立位置被解析；runtime_context.py 在单个测试里被解析 559 次；_hook_declared_fields 被调用 1968 次）；单次全量 parse 需 2.75 秒（热）到 7.04 秒（冷），而逐文件读取仅 0.085 秒。
2. 源码预过滤：每个扫描都用“字面标识符”匹配 AST 节点（例如 func.id == "get_server_args"、ClassDef.name == "ServerArgs"、参数名 server_args），标识符出现在 AST 中必然逐字出现在源码文本中。因此在 ast.parse 之前先读文本并做 if "<token>" not in source: continue，跳过无 token 的文件。真实命中率很低：get_server_args 27/3467、configured_ 41/3467、_late_resolution 4/3467、server_args 599/3467。涉及 test_global_config_read_ratchet.py、test_chain_read_ratchet.py、test_supplied_instance_exposure_ratchet.py、test_resolution_reads_no_bag.py、test_server_args_namespaces.py、test_launch_path_reads_configured_sizes.py、test_publish_precedes_bag_reads.py。
3. 记忆化：对“同一文件被多次解析”的场景引入 functools.lru_cache：test_model_config_reads_resolved_input.py 新增 _parsed(path) 与 _declared_resolution_fields(path)，让 _hook_declarations 及各解析点复用同一棵树；

test_resolution_reads_no_bag.py 将 _registered_entries() 缓存并抽出 _functions_in(path) 供 _reaches_a_bag 复用函数表；test_supplied_instance_exposure_ratchet.py 用类属性 _READS_CACHE 缓存 _supplied_instance_reads 的结果，避免每个测试方法都重扫全包。

4. 刻意保留的边界：不把 token 收紧成 ".server_args" 点号子串，因为 model_runner . server_args . pp_size 是合法 Python，点号子串过滤会漏掉；同时验证了当前代码不存在 id()/is 比较和 AST 原地修改，证明共享解析树安全。
5. 等价性验证：未改动任何断言与豁免列表。两套 harness 全部通过：新旧实现并排运行且输出完全一致（ALL EQUIVALENCE ASSERTIONS PASSED）；再复制包并注入 10 个覆盖各类拼写的违规模块作负对照（NEGATIVE CONTROL PASSED），确认加速没有引入漏报。test_server_args_namespaces.py 的 sites 保持 1990，> 1500 的空洞守卫不受影响。

关键文件：

- test/registered/unit/server_args/test_model_config_reads_resolved_input.py（模块 模型配置；类别 test；类型 performance；符号 _parsed, _declared_resolution_fields）：改动最大的文件：新增 _parsed 与 _declared_resolution_fields 两个 lru_cache 辅助函数，把全部内联 ast.parse 收敛到 _parsed，并让 _hook_declarations 复用抽取后的字段集合；本文件本地扫描 9.69 秒降到 0.34 秒（28.8x）。
- test/registered/unit/server_args/test_resolution_reads_no_bag.py（模块 解析逻辑；类别 test；类型 performance；符号 _registered_entries, _functions_in, _reaches_a_bag）：新增 _functions_in 缓存、_registered_entries 改为 lru_cache，并对 run_post_process_pass 调用点扫描先做 token 预过滤再 parse；本文件 13.63 秒降到 0.56 秒（24.5x），同时大幅减少同时驻留的 AST 数量。
- test/registered/unit/test_supplied_instance_exposure_ratchet.py（模块 参数暴露；类别 test；类型 performance；符号 _late_resolution_written_fields, _supplied_instance_reads, _override_written_fields）：给三个大 census 循环加 token 预过滤（_late_resolution、server_args、record_config_updates 与 get_context + override 的组合判断），并为 _supplied_instance_reads 加入类级缓存；7.4x / 7.0x 加速。
- test/registered/unit/test_chain_read_ratchet.py（模块 链式读取；类别 test；类型 performance；符号 _declared_by_keyword, _subtrees_with_their_own_record, _chain_reads）：三处扫描（_declared_by_keyword、_subtrees_with_their_own_record、_chain_reads）全部加 token 预过滤，避免对不含目标标识符的文件 parse；4.9x 加速。
- test/registered/unit/test_global_config_read_ratchet.py（模块 全局配置；类别 test；类型 performance；符号 _field_reads, test_the_call_sites_match_the_documented_set, test_the_scanned_accessors_are_never_import_renamed）：_field_reads、call-site 匹配和 import-rename 检查三个循环都先查 get_server_args / configured_ 再 parse，显著降低全包解析次数；14.7x 加速。
- test/registered/unit/test_server_args_namespaces.py（模块 命名空间；类别 test；类型 performance；符号 test_no_module_shadows_a_bag_accessor, test_the_readers_agree_with_the_namespace_metadata）：通过 accessors 名称列表对两个测试循环做预过滤，避免对不含相关标识符的文件 parse；1.7x 加速。

- test/registered/unit/test_launch_path_reads_configured_sizes.py（模块 启动路径；类别 test；类型 performance）：本文件增加 5 行预过滤，同样属于家族修复的一部分；2.8x 加速。
- test/registered/unit/test_publish_precedes_bag_reads.py（模块 发布时序；类别 test；类型 performance）：增加 3 行 token 预过滤，7.0x 加速；是 publish-before-read 时序 ratchet 的配套优化。

关键符号：_parsed, _declared_resolution_fields, _registered_entries, _functions_in, _reaches_a_bag, _supplied_instance_reads, _late_resolution_written_fields, _override_written_fields, _declared_by_keyword, _subtrees_with_their_own_record, _chain_reads, _field_reads

关键源码片段

test/registered/unit/server_args/test_model_config_reads_resolved_input.py

改动最大的文件：新增 _parsed 与 _declared_resolution_fields 两个 lru_cache 辅助函数，把全部内联 ast.parse 收敛到 _parsed，并让 _hook_declarations 复用抽取后的字段集合；本文件本地扫描 9.69 秒降到 0.34 秒（28.8x）。

```
# 核心思想：扫描的匹配谓词都以"字面标识符"命中 AST 节点，
# 而标识符出现在 AST 中意味着源码文本里必然存在该字面 token。
# 因此对没有该 token 的文件可以安全跳过 ast.parse —— 解析本身约
# 2.75 秒（热）/ 7.04 秒（冷），而 read_text 全文只要 0.085 秒。

import ast
import functools
import pathlib

# lru_cache 化解析：同一文件在同一测试里可能被 parse 数百次
#  （runtime_context.py 1777 行曾被 parse 559 次；server_args.py 10745 行、
# 24 ms/ 次 在 6 个独立位置重复解析）。输入相同，缓存 AST 等价；
# 作者已验证代码中没有对 AST 节点的 id()/is 比较，也没有就地修改树。
@functools.lru_cache(maxsize=None)
def _parsed(path):
    return ast.parse(path.read_text(encoding="utf-8-sig"))

# 派生集合也缓存：_hook_declarations 原本对每个声明调用点都重新遍历被
# import 模块的整棵树，抽成独立函数后所有调用点共享第一次计算的结果。
@functools.lru_cache(maxsize=None)
def _declared_resolution_fields(path):
    fields = set()
    for node in ast.walk(_parsed(path)):
        if (
            isinstance(node, ast.Call)
            and isinstance(node.func, ast.Name)
            and node.func.id == "declare_resolution"
        ):
            fields |= {kw.arg for kw in node.keywords if kw.arg}
```

```
    return frozenset(fields)

# 使用示例：定位 server_args.py 中 _handle_model_specific_adjustments 处理器，
# 通过命中缓存只真正 parse 一次，后面所有测试共享同一棵树。
def _registry_collection_is_after_the_build():
    tree = _parsed(_SRT / "server_args.py")
    handler = next(
        node for node in ast.walk(tree)
        if isinstance(node, ast.FunctionDef)
        and node.name == "_handle_model_specific_adjustments"
    )
    # … 后续 collect/build 行号比较逻辑省略
```

test/registered/unit/server_args/test_resolution_reads_no_bag.py

新增 _functions_in 缓存、_registered_entries 改为 lru_cache，并对 run_post_process_pass 调用点扫描先做 token 预过滤再 parse；本文件 13.63 秒降到 0.56 秒（24.5x），同时大幅减少同时驻留的 AST 数量。

```
# 注册项枚举缓存：_registered_entries 原本在每个测试方法里对全部 1664 个
# srt 文件重复执行注册表 + 调用点扫描，现在整个枚举只计算一次。
@functools.lru_cache(maxsize=None)
def _registered_entries():
    entries = set()
    … # 遍历注册表取得 pass / override 提供者，省略中间细节
    return entries

# 预过滤示例：静态读入一次源码，先检查调用点 token。
# run_post_process_pass 在真实树上命中极少，绝大多数文件没有该标识符，
# 不需要 parse；而 read_text 全部文件仅 0.085 秒，几乎免费。
sources = {
    path: path.read_text(encoding="utf-8-sig")
    for path in sorted(_SRT.rglob("*.py"))
}
for path, source in sources.items():
    if "run_post_process_pass" not in source:
        continue
    try:
        tree = ast.parse(source)
    except SyntaxError:
        continue
    # … 收集 by_value 调用点，省略中间细节

# 模块内函数表缓存：_reaches_a_bag 需要对多个入口做递归 call graph 遍历，
# 每个入口都重新 parse 同一文件；抽成 _functions_in 后函数表只建一次。
@functools.lru_cache(maxsize=None)
def _functions_in(path):
    tree = ast.parse(path.read_text(encoding="utf-8-sig"))
    return {
        node.name: node
```

```
        for node in ast.walk(tree)
        if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef))
    }

# 递归入口复用缓存函数表，seen 集合防止成环。
def _reaches_a_bag(path, entry):
    functions = _functions_in(path)
    seen = set()
    def walk(name):
        ... # 递归跟随函数调用，命中 bag accessor 即返回 True，省略细节
    return walk(entry)
```

评论区精华

该 PR 没有形成独立 review 评论，实质讨论都在 PR body 的 “For reviewers — three judgment calls” 与验证章节里。作者主动给出三个判断点：其一，不收紧为 ".server_args" 子串，因为 model_runner . server_args . pp_size 是合法 Python；其二，lru_cache 共享 AST 是安全的，因为代码中不存在 id()/is 节点比较和树突变；其三，CI 端到端收益不会直接映射到本地扫描加速，因为每个文件还有约 10 秒的解释器启动与 import torch / import sglang 固定开销。GitHub 侧只有机器人反复提示 “Rebase Required Before Re-run”，作者 rebase 后手动执行 /rerun-test，机器人确认 7 个测试全部通过，无遗留疑虑。

- 设计决策：不做 .server_args 点号子串收紧 (design): 保留通用 token 预过滤（server_args 命中 599/3467），用等价性断言与负对照支撑安全性；未收紧成点号子串。
- 等价性验证：identical-output 与 negative control (testing): 两条验证均通过，断言未放开，test_server_args_namespaces 的 sites 保持 1990，> 1500 守卫不受影响。
- CI rebase 导致自动 rerun 未触发 (other): rebase 并重新 push 后手动 rerun，机器人报告 7 个任务全部通过；无遗留问题。

风险与影响

- 风险：技术风险集中在三个不变量上：1）token 预过滤依赖“匹配谓词即字面标识符”的假设，未来若出现正则匹配或符号表解析这类非字面匹配，预过滤可能产生假阴性；当前负对照覆盖了所有已知拼写，包括 ctx . get_server_args() 与 model_runner . server_args 的空格点号形式，但无法防止未来漂移。2）lru_cache 缓存 AST 后，后续维护者若在遍历中做节点改写，会把修改泄漏到下一次调用；作者只验证了当前快照。3）_READS_CACHE 这类类属性缓存与测试类全局共享，若以后 census 依赖不同包路径或上下文，缓存会陈旧。此外本 PR 只优化扫描段，CI 单文件还有约 10 秒解释器启动与 import 固定成本，不能完全根治分片超时，需与 #36241 这类启动开销削减配合。
- 影响：用户与运行时零影响：全部改动都在 test/registered/unit 下，未触碰 sglang.srt 生产代码。系统侧：base-a-test-cpu 的扫描段从 111.7 秒降到 17.7 秒，8 个测试各自提速 1.7x-28.8x，显著降低分片超时概率，test_resolution_reads_no_bag 的驻留 AST 从 1664 棵降到几棵，RSS 约 860 MB 降至微量。团队侧：为所有静态 census 类测试提供了通用优化模板（先读文本判断是否 parse、lru_cache 记忆化、双 harness 等价验证），后续扩展 ratchet 时可直接套用。

- 风险标记：字面 token 预过滤依赖匹配谓词逐字性，AST 缓存假设树不可变，仅缓解扫描段耗时，固定启动开销仍在

关联脉络

- PR #36235 [CI] Stop the resolution ratchets re-parsing the whole package: 同一 perf/config-ratchet-scans 系列的前置 / 并行 PR，针对 resolution ratchets 做同样优化，文件范围与本 PR 高度重合（如 test_resolution_reads_no_bag 等）。
- PR #36241 [CI] Cut repeated tokenizer loads, serial subprocesses and a double scan: 同一 CI 开销削减系列，处理 CPU 测试的重复加载与串行子进程开销；与本 PR 的 split 优化互补，共同降低 base-a-test-cpu 的分片耗时。
- PR #36242 [CI] Register test_dflash_logits at its real cost: 同一 CI 时长治理脉络中对 test_dflash_logits 的计费修正，反映团队在系统性校准 CPU CI 测试成本，与本 PR 属同一工作流。