PR #33929 完整报告

sgl-project/sglang

[misc] Remove break-graph debug log; reclaim pid-less /dev/shm leaks in CI

合并时间：2026-08-07 12:18

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33929

执行摘要

- 一句话：删除冗余 graph 日志，CI 启动清扫无 pid 的 /dev/shm 泄漏族
- 推荐动作：值得精读 stale_shm_cleanup.py 的清扫逻辑：它展示了“依赖执行时机换取无条件删除”这一设计权衡，以及如何用前缀白名单覆盖无法用 pid 追踪的泄漏族。对于 CI 基础设施维护者，这是一个低风险、高收益的可靠性改进。实现直白，测试与源码同步更新，适合作为 CI 自愈逻辑的参考样例。

功能与动机

PR body 明确指出两个痛点：一是 break-graph debug log 每次捕获每层都打印，在 --log-level debug 下淹没了 CI job 日志（7424 行，约占 50%），且断点位置在 import 时已固定，重复打印无信息增量；二是 nightly job 因 NCCL 创建共享内存段时报 No space left on device (28) 失败，原因是长期运行的 runner 容器在 /dev/shm 中积累了数万个泄漏文件，而既有 cleanup_stale_shm 只回收带 creator pid 的段（sgl_shm_、multi_tokenizer_args_），真正泄漏的族（sglang_loads_、cuda.shm.、nccl-、sem.loky-）都不带 pid，无法按 pid 判断存活。

实现拆解

实现拆解分 4 步：

1. 移除 break-graph debug 日志：在 python/sglang/srt/model_executor/runner_backend_utils/breakable_cuda_graph/breakable_cuda_graph.py 中删除 eager_on_graph wrapper 内的 logger.debug("Break graph due to function: %s", inner.__name__)，并移除不再使用的 logging import 和 logger 全局变量。原因是该日志信息静态、重复且高频，移除后不影响任何功能，仅减少日志噪音。
2. 扩展 CI /dev/shm 清扫规则：在 python/sglang/srt/utils/stale_shm_cleanup.py 中新增 _ORPHAN_PREFIXES 元组，包含 4 个无 pid 标记的泄漏族前缀。调整 _cleanup_stale_shm_impl 的控制流：原先 pid is None 直接跳过，现在改为 elif not entry.name.startswith(_ORPHAN_PREFIXES): continue，即命中孤儿族前缀的条目无条件 unlink。这样做的依据是清扫时机固定在 CI job 启动、killall.py 执行之后，可保证没有任何进程仍引用这些文件。
3. 同步更新测试：在 test/registered/utils/test_stale_shm_cleanup.py 中，将 _unlink_quiet 改为接收完整路径并用 os.unlink(path)（因为孤儿族是普通文件而非

SharedMemory 对象）；新增 _make_raw_file 创建 4096 字节普通文件；新增 test_orphan_family_sweep 验证 4 个族均被清理而未知前缀（unknown_family_file）被保留，并用 skipUnless 限定仅在 CI 环境运行，避免开发机上误删。

4. 配套整理：更新模块 docstring 和函数 docstring，说明两套规则（pid 探活 vs 无条件 unlink）及其安全前提；提交历史经过 6 次迭代，最终将设计简化为 job-start 无条件 unlink，并统一测试清理 helper。

关键文件：

- python/sglang/srt/utils/stale_shm_cleanup.py（模块 SHM 清理；类别 source；类型 core-logic；符号 _ORPHAN_PREFIXES, _cleanup_stale_shm_impl, cleanup_stale_shm）：核心逻辑：新增 _ORPHAN_PREFIXES 无条件清理无 pid 的泄漏族，调整 _cleanup_stale_shm_impl 控制流，是修复 /dev/shm 泄漏的关键。
- test/registered/utils/test_stale_shm_cleanup.py（模块 单元测试；类别 test；类型 test-coverage；符号 _unlink_quiet, _make_raw_file, test_orphan_family_sweep）：新增 _make_raw_file 和 test_orphan_family_sweep，验证孤儿族被清理而未知前缀保留，并将 _unlink_quiet 改为 os.unlink 以支持普通文件。
- python/sglang/srt/model_executor/runner_backend_utils/breakable_cuda_graph/breakable_cuda_graph.py（模块 CUDA 图；类别 source；类型 refactor；符号 eager_on_graph）：删除 per-layer break-graph debug 日志及其相关 logging import，减少 debug 模式下的日志洪水。

关键符号：cleanup_stale_shm, _cleanup_stale_shm_impl, eager_on_graph

关键源码片段

python/sglang/srt/utils/stale_shm_cleanup.py

核心逻辑：新增 _ORPHAN_PREFIXES 无条件清理无 pid 的泄漏族，调整 _cleanup_stale_shm_impl 控制流，是修复 /dev/shm 泄漏的关键。

```
# /dev/shm 中无 pid 标记的 " 孤儿族 " 前缀。这些文件在进程被 SIGKILL 杀时
# 不会走到 Python 的 unlink 路径，因此长期累积。它们不带 pid，无法判断
# 创建者是否存活，只能依赖 " 清扫时机在 CI job 启动且 killall.py 之后 "
# 这一前提来无条件删除。
_ORPHAN_PREFIXES = (
    "sglang_loads_", # managers/load_snapshot.py 的 slot 文件
    "cuda.shm.", # CUDA IPC 共享内存段
    "nccl-", # NCCL communicator 段
    "sem.loky-", # loky/joblib 信号量
)


def _cleanup_stale_shm_impl() -> None:
    # 非 CI 环境直接跳过：pid 探活对其他用户的进程不可靠
    if not _is_in_ci():
        return
    if not _SHM_DIR.is_dir():
```

```
        return

    removed = 0
    freed_bytes = 0
    try:
        entries = list(_SHM_DIR.iterdir())
    except OSError as e:
        logger.warning("cleanup_stale_shm: cannot list %s, skipping: %s", _SHM_DIR, e)
        return

    for entry in entries:
        pid = _creator_pid(entry.name)
        if pid is not None:
            # pid 复用会读成 " 存活 "，因此该逻辑偏向少删（泄漏）而不是误删
            # 活跃段。修改此检查时请保持这个偏向。
            if pid == os.getpid() or _pid_alive(pid):
                continue
        elif not entry.name.startswith(_ORPHAN_PREFIXES):
            # 不带 pid 且不在孤儿族内（如 psm_*），一律跳过
            continue

        try:
            size = entry.stat().st_size
            entry.unlink()
            removed += 1
            freed_bytes += size
        except FileNotFoundError:
            pass # 与其他清理进程竞态
        except OSError as e:
            logger.warning("cleanup_stale_shm: failed to remove %s: %s", entry.name, e)

    if removed:
        logger.info(
            "cleanup_stale_shm: removed %d stale segment(s), freed %.1f MiB",
            removed,
            freed_bytes / (1 << 20),
        )
```

评论区精华

该 PR 没有任何 review 评论，唯一的评论区交互是作者触发的 /tag-and-rerun-ci 命令（issue 评论，用于重新标记 CI）。从 6 个 commit 的演进可以看出设计决策：最初提交 32aa698 尝试在 close 时 unlink load snapshot shm，随后 9d4c84d 简化为 job-start 无条件 unlink，最终 8eed8f1 进一步拆分 if/elif 分支并压缩 docstring。这个由繁到简的过程体现了对“时机保证安全”的信任：只要清扫发生在 killall 之后，就不需要复杂的引用计数或线程安全处理。

- 暂无高价值评论线程

风险与影响

- 风险：主要风险集中在无条件 unlink 的依赖前提上：
- 时序依赖：无条件删除的安全性完全依赖于“清扫在 CI job 启动且 killall.py 之后执行”。如果未来 CI 脚本调整顺序，或有人将 cleanup_stale_shm 移到其他调用点（例如运行中 server 的启动路径），可能误删活跃进程正在使用的共享内存段。当前 _is_in_ci() 门控降低了该风险，但并非绝对隔离。
- 前缀宽匹配：nccl-、cuda.shm. 等前缀较宽泛，在 /dev/shm 中可能匹配到非预期文件。虽然限定 CI 环境且时机正确，但若未来有其他工具使用相同前缀，可能造成误删。
- 测试覆盖局限：test_orphan_family_sweep 用 skipUnless 仅在 CI 环境运行，本地开发环境无法自动验证该路径，回归需依赖 CI。
- 日志删除无风险：break-graph 日志移除不改变任何行为，风险可忽略。
- 影响：影响范围集中在 CI 基础设施和 debug 日志输出：
- 对用户：无运行时功能影响，仅影响 --log-level debug 下的日志量和 CI 作业稳定性。
- 对系统：修复了长期存在的 /dev/shm 泄漏累积问题，避免 nightly/daily CI job 因 tmpfs 耗尽而失败；日志量在 debug 模式下显著减少（约 50%）。
- 对团队：降低 CI 维护成本，减少因环境问题导致的无效重跑；清扫逻辑本身是纯增量改进，不影响现有带 pid 段的清理行为。
- 风险标记：依赖 killall 时序，前缀宽匹配风险，仅 CI 环境生效，测试 CI-only

关联脉络

- 暂无明显关联 PR