PR #1954 完整报告

THUDM/slime

[coding_agent_rl] middleware: shutdown_session drains in-flight handl···

合并时间：2026-05-27 16:02

原文链接：http://prhub.com.cn/THUDM/slime/pull/1954

执行摘要

- 一句话：修复 SWE RL rollout 关闭时的竞态崩溃
- 推荐动作：该 PR 值得精读，尤其对于需要在异步服务中实现优雅关闭的场景。其设计模式（模块级 inflight 追踪 + 墓碑 + 多层 drain）可以复用。建议关注 _handle_request 中的 try/finally 结构，确保异常安全。此外，考虑在未来将 settle_sec 和 flush_cache 的超时参数化，便于调优。

功能与动机

当 SWE rollout 样本返回后，slime 的训练循环会立即调用 release_memory_occupation，其内部通过 is_fully_idle() 进行严格断言。然而刚关闭的 sandbox 中可能仍有滞留的 claude-cli 请求位于 sglang 管道中（running_batch / chunked_req / result_queue / hicache writes），导致断言失败并 SIGQUIT 调度器子进程。每个多步 SWE RL 运行在第一个 rollout 步之后的 offload 时都会崩溃。

实现拆解

1. middleware.py 模块级状态：在模块作用域添加 _inflight: dict[str, set[asyncio.Task]] 和 _closed: set[str]。_inflight 用于追踪每个 sid 正在运行的 _handle_request 任务；_closed 作为永久墓碑标记已关闭的 sid。它们位于 Session 类外部，从而在 pop_session_split 销毁 Session 后仍可访问。
2. _handle_request 集成追踪与拒绝：在 _handle_request 入口处检查 sid 是否已被关闭，如果是则直接返回 503。将原来的同步处理逻辑包裹在 try/finally 块中，try 块开始前将当前任务注册到 _inflight，finally 块中移除。这确保了即使异步任务异常退出，追踪也会正确清理。
3. shutdown_session 函数：新增异步函数，执行三层 drain 流程：
   - 将 sid 加入 _closed 集合，后续请求立即被拒绝
   - 等待 _inflight[sid] 中所有正在运行的任务完成（通过 asyncio.wait）
   - 调用 sglang 的 /flush_cache 接口，利用其内置的 idle 等待机制确保管道清空
4. generate.py finally 重排：将 shutdown_session 移到 pop_session_split 之前调用，确保在提取训练数据前所有滞留请求已被清理。

关键文件：

- examples/coding_agent_rl/middleware.py（模块 中间件；类别 source；类型 core-logic；符号 _inflight, _closed, _handle_request, shutdown_session）：核心实现，新增 shutdown_session 函数以及模块级 _inflight 和 _closed 状态，修改 _handle_request 以支持任务追踪和拒绝已关闭的请求。
- examples/coding_agent_rl/generate.py（模块 生成器；类别 source；类型 core-logic；符号 generate）：修改 generate 函数的 finally 块，在 pop_session_split 前调用 shutdown_session，确保顺序正确。

关键符号：shutdown_session, _handle_request, generate

关键源码片段

examples/coding_agent_rl/middleware.py

核心实现，新增 shutdown_session 函数以及模块级 _inflight 和 _closed 状态，修改 _handle_request 以支持任务追踪和拒绝已关闭的请求。

```
# 模块级状态，独立于 Session 生命周期
_inflight: dict[str, set[asyncio.Task]] = {}
_closed: set[str] = set()

async def shutdown_session(sid: str, *, wait_timeout: float = 5.0) -> None:
    """Tombstone sid (late requests 503) and drain in-flight local handlers
    (cancel fires /abort_request to sglang). Does NOT wait for sglang idle --
    sglang_engine.release_memory_occupation already calls flush_cache() with
    60×1s polling. Idempotent."""
    _closed.add(sid)
    tasks = [t for t in _inflight.get(sid, set()) if not t.done()]
    if tasks:
        _, pending = await asyncio.wait(tasks, timeout=wait_timeout,
                                         return_when=asyncio.ALL_COMPLETED)
        for t in pending:
            t.cancel() # /abort_request fires on cancellation
        await asyncio.gather(*pending, return_exceptions=True)

async def _handle_request(request: web.Request) -> web.StreamResponse:
    body = await request.json()
    sid = request.headers["Authorization"].removeprefix("Bearer ").strip()
    if sid in _closed: # session drained; refuse stragglers
        return web.Response(status=503, text="session closed")
    app = request.app
    s = app["store"].setdefault(sid, Session())
    task = asyncio.current_task()
    _inflight.setdefault(sid, set()).add(task)
    try:
        async with s.lock:
            target, is_sub, kind = _select_chain(s, body)
            ideal_ids = _build_prompt(target, body, kind, app["tokenizer"])
            output_ids, finish = await _generate(target, ideal_ids, s, body, app)
```

```
            blocks, stop, did = _build_reply(target, output_ids, finish, app)
            if did and not is_sub:
                _start_sub_chain(s, did)
            in_tok, out_tok = len(ideal_ids), len(output_ids)
            return await _stream_response(request, blocks, stop, in_tok, out_tok)
    finally:
        _inflight.get(sid, set()).discard(task)
```

examples/coding_agent_rl/generate.py

修改 generate 函数的 finally 块，在 pop_session_split 前调用 shutdown_session，确保顺序正确。

```
# generate() 函数末尾的 finally 块，之前是直接 pop_session_split
finally:
    # Close the sid before next train step's release_memory_occupation;
    # stragglers from this trajectory would otherwise race its idle assert.
    await middleware.shutdown_session(session_id)
    middleware.pop_session_split(state.store, session_id) # idempotent
```

评论区精华

该 PR 没有产生 review 评论，由 zhuzilin 直接批准。PR body 中包含了详细的背景分析和设计决策说明，核心讨论隐含在 PR 描述中：

- 为什么选择模块级状态而非 Session 内状态：因为 pop_session_split 会销毁 Session 对象，而 shutdown_session 需要在其之后仍能访问状态。
- 为什么需要三层 drain：单靠 settle sleep 仍存在竞态，结合 sglang 原生的 flush_cache idle 等待更可靠。
- 为什么 _closed 不随 pop_session_split 清除：因为 sid 在每个 rollout 步是唯一的，集合仅线性增长，不影响性能。
- 暂无高价值评论线程

风险与影响

- 风险：
  1. 模块级状态泄露：_inflight 和 _closed 在进程生命周期内持续存在，如果 sid 数量极大，可能导致内存占用过高。但 PR 作者指出 sid 是每个 rollout 步唯一的，且仅线性增长，风险可控。
  2. shutdown_session 超时：wait_timeout 默认 5 秒，如果任务未能在此期间完成，剩余任务将被取消。理论上可能导致部分请求未完成，但最终 sglang 的 release_memory_occupation 会通过 flush_cache 进行更彻底的等待。
  3. 依赖 sglang 行为：flush_cache 的实现依赖于 sglang 版本，如果未来版本改变了 idle 检查逻辑，可能导致本修复失效。
  4. 仅影响示例代码：所有修改均在 examples/coding_agent_rl 目录内，不影响 slime 核心库的稳定性。 - 影响：影响范围：仅限 examples/coding_agent_rl 示例，不影响 slime 核心模块或其他示例。用户影响：之前 SWE RL 训练在每一步 rollout 后都会崩溃，修复

后可以稳定运行多个训练步。系统影响：无，变更仅在 middleware 层添加状态管理，不修改 sglang 后端。向后兼容：完全兼容，新增函数 shutdown_session 是可选的，现有使用 pop_session_split 的代码仍能正常工作。

- 风险标记：内存泄露风险，外部依赖行为变化

关联脉络

- PR #1957 Minor refactor for coding agent rl logic and remove SWE_LIST_TRAJECTORY: 同为 coding_agent_rl 示例的改进，涉及 generate.py 和 middleware 逻辑重构，与本 PR 存在交互（shutdown_session 的调用位置）。
- PR #1956 Add slime/agent/ and move sandbox impl inside: 将 sandbox 从示例迁移至核心模块，与本 PR 同属 agentic 功能线，可能影响关闭流程。
- PR #1923 [examples] add coding_agent_rl: agent-in-sandbox RL minimal demo: coding_agent_rl 示例的初始实现，本 PR 修复了该示例中引入的竞态问题。