

PR #56988 完整报告

vllm-project/vllm

[Bugfix][Responses] Clean up MCP tool sessions once before closing them

合并时间: 2026-09-16 14:07

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56988>

执行摘要

- 一句话: MCP 会话清理回调仅注册一次, 修复重复清理与关闭后调用
- 推荐动作: 值得精读, 尤其是 AsyncExitStack 回调注册时机与去重模式: 将遍历全部会话的清理回调从循环内移到循环外、用布尔标志控制注册一次, 是处理 " 多次初始化、单次清理 " 任务的简洁范例。配套测试的参数化矩阵 (空 / 单 / 未使用 / 首 / 末 / 全部) 与部分初始化失败用例也值得借鉴。对使用 Responses API + MCP 工具的团队, 建议合入后关注 MCP 服务端日志中清理请求是否恢复正常频率。

功能与动机

PR body 明确指出: "With multiple MCP tool sessions, both Responses contexts register the same cleanup callback once per session. Each callback visits all called tools. This can send duplicate cleanup requests to an open session and attempt another call on a session that has already closed." 即多个 MCP 会话共存时, 每个会话都注册一次遍历全部 `called_tools` 的清理回调, 造成清理请求重复发送, 且可能在会话关闭后才发起调用。PR 关联 issue #25697 (gpt-oss 流式工具调用报错), 并说明本变更只解决其中 `cleanup` 顺序问题, 不修复独立的跨任务 `cancel-scope` 错误。

实现拆解

1. 变更入口与根因定位: 问题位于 `vllm/entrypoints/openai/responses/context.py` 中 `ConversationContext.init_tool_sessions` (第 560 行附近) 与 `HarmonyContext.init_tool_sessions` (第 861 行附近)。两处实现都在 `for tool_name in self.available_tools` 循环体内调用 `exit_stack.push_async_exit(self.cleanup_session)`, 每个新建会话都压入一个清理回调。
2. 核心逻辑改造: 引入局部布尔标志 `initialized_session = False`, 在循环开始前初始化; 每当真正打开新会话 (即 `tool_name not in self._tool_sessions` 且 `new_session` 成功返回) 时置为 `True`; 循环结束后仅在 `initialized_session` 为真时注册一次 `cleanup_session`。这样既保证有会话需要清理时回调必被注册, 又避免重复注册。由于 `cleanup_session` 内部通过 `asyncio.gather` 遍历 `self.called_tools` 并发清理所有用过的会话, 单次注册即可覆盖全部会话, 且 `AsyncExitStack` 按 LIFO 顺序执行退出回调, `cleanup_session` 会在各会话关闭之前运行。
3. 测试配套: `tests/entrypoints/openai/responses/test_serving_responses.py` 新增 `tool_context` fixture (参数化为 `ParsableContext` 与 `HarmonyContext`, 用 `__new__` 绕过

__init__ 仅铺设最小状态)，以及两个异步测试：test_mcp_sessions_cleaned_once_before_close 用 6 组 available_tools / called_tools 参数组合断言每个被调用会话恰好收到一次 cleanup_session、cleanup 事件先于 close 事件、close 顺序符合 LIFO；test_mcp_partial_initialization_closes_opened_session 模拟第二个会话初始化抛 RuntimeError，验证已打开的会话仍会被关闭且不会触发清理。

4. 配置与部署配套：无额外配置或部署改动；cleanup_session 本体逻辑未变，仍通过 contextlib.suppress(Exception) 容忍清理失败。

关键文件：

- vllm/entrypoints/openai/responses/context.py（模块 响应上下文；类别 source；类型 core-logic；符号 init_tool_sessions, cleanup_session）：核心修复文件：ConversationContext 与 HarmonyContext 的 init_tool_sessions 均改为循环外单次注册 cleanup_session，消除重复清理与关闭后调用。
- tests/entrypoints/openai/responses/test_serving_responses.py（模块 回归测试；类别 test；类型 test-coverage；符号 tool_context, test_mcp_sessions_cleaned_once_before_close, new_session, test_mcp_partial_initialization_closes_opened_session）：新增生命周期回归测试：覆盖 ParsableContext/HarmonyContext 两种实现、6 组工具使用组合、重复初始化与部分初始化失败场景，是验证修复正确性的主要依据。

关键符号：init_tool_sessions, cleanup_session,
test_mcp_sessions_cleaned_once_before_close,
test_mcp_partial_initialization_closes_opened_session

关键源码片段

vllm/entrypoints/openai/responses/context.py

核心修复文件：ConversationContext 与 HarmonyContext 的 init_tool_sessions 均改为循环外单次注册 cleanup_session，消除重复清理与关闭后调用。

```
async def init_tool_sessions(
    self,
    tool_server: ToolServer | None,
    exit_stack: AsyncExitStack,
    request_id: str,
    mcp_tools: dict[str, Mcp],
):
    """为请求涉及的 MCP 工具打开会话，并在循环结束后只注册一次统一清理回调。"""
    if tool_server:
        initialized_session = False # 记录本轮是否真正打开了新会话
        for tool_name in self.available_tools:
            if tool_name in self._tool_sessions:
                continue # 重复初始化时复用已有会话，不再重复注册清理

            tool_type = _map_tool_name_to_tool_type(tool_name)
            headers = (
                mcp_tools[tool_type].headers if tool_type in mcp_tools else None
            )
```

```

        tool_session = await exit_stack.enter_async_context(
            tool_server.new_session(tool_name, request_id, headers)
        )
        self._tool_sessions[tool_name] = tool_session
        initialized_session = True
    if initialized_session:
        # cleanup_session 内部会遍历 self.called_tools 清理所有已调用工具，
        # 因此这里只需注册一次；原来在循环内注册会导致重复清理请求，
        # 甚至在会话关闭后仍尝试调用。
        exit_stack.push_async_exit(self.cleanup_session)

# 说明：AsyncExitStack 按 LIFO 顺序执行退出回调，cleanup_session 在
# 会话进入上下文之后才注册，因此它会先于各 session 的关闭逻辑执行，
# 保证清理请求发送时会话仍然存活。

async def cleanup_session(self, *args, **kwargs) -> None:
    """作为 __aexit__ 协程使用：对所有被调用过的工具会话发送清理请求。"""

    async def cleanup_tool_session(tool_session):
        # 本地 Tool（如 LocalShell）不走 MCP cleanup_session 协议，直接跳过
        if not isinstance(tool_session, Tool):
            logger.info(
                "Cleaning up tool session for %s", tool_session._client_info
            )
            with contextlib.suppress(Exception):
                await tool_session.call_tool("cleanup_session", {})

    await asyncio.gather(
        *(
            cleanup_tool_session(self._tool_sessions[tool])
            for tool in self.called_tools
        )
    )

```

tests/entrypoints/openai/responses/test_serving_responses.py

新增生命周期回归测试：覆盖 ParsableContext/HarmonyContext 两种实现、6 组工具使用组合、重复初始化与部分初始化失败场景，是验证修复正确性的主要依据。

```

# tool_context fixture：在 ParsableContext / HarmonyContext 两个实现上
# 跑同一套生命周期断言，验证修复对两条代码路径都生效。
@pytest.fixture(params=[ParsableContext, HarmonyContext])
def tool_context(self, request):
    context = request.param.__new__(request.param) # 跳过 __init__，只铺设最小状态
    context.available_tools = ["browser", "python"]
    context._tool_sessions = {}
    context.called_tools = set()
    return context

```

```

@asynccontextmanager
async def new_session(name, *_args):
    # 模拟 ToolServer.new_session: 记录 cleanup 与 close 事件的先后顺序
    session = MagicMock()
    session.call_tool = AsyncMock(
        side_effect=lambda _: events.append(("cleanup", name))
    )
    sessions[name] = session
    try:
        yield session
    finally:
        events.append(("close", name))

# 连续两次初始化, 验证重复初始化不会重复注册清理回调
async with AsyncExitStack() as stack:
    await tool_context.init_tool_sessions(tool_server, stack, "req", {})
    await tool_context.init_tool_sessions(tool_server, stack, "req", {})
    tool_context.called_tools.update(called_tools)

for name, session in sessions.items():
    if name in called_tools:
        # 每个被调用过的会话只收到一次 cleanup_session, 且先于 close 发生
        session.call_tool.assert_awaited_once_with("cleanup_session", {})
        assert events.index(("cleanup", name)) < events.index(("close", name))
    else:
        session.call_tool.assert_not_awaited()

# AsyncExitStack 按 LIFO 顺序关闭会话: 最后打开的会话最先关闭
assert [name for event, name in events if event == "close"] == list(
    reversed(available_tools)
)

```

评论区精华

该 PR 来自 fork, 审查过程较精简: [claude\[bot\]](#) 自动回复说明 fork 仓库默认禁用自动化 review, 需维护者手动触发; 维护者 [chaunceyjiang](#) 直接批准并回复 "Thanks~", 未提出额外修改意见。CI 流程中第一次 `/ci run` 因分支落后上游 main 43 个 commit 被拒绝, 作者通过 merge main 同步后重新触发 Buildkite CI #89248, 最终通过。整体没有出现实质性技术争论, 核心设计 (循环外单次注册) 在 PR body 中已有充分论证。

- 暂无高价值评论线程

风险与影响

- 风险:
 1. 清理失败被静默吞掉: `cleanup_session` 内 `with contextlib.suppress(Exception)` 会吞掉所有 `call_tool("cleanup_session", {})` 异常且无日志输出, MCP 服务端清理失败时难

以观测。

2. 并发清理无超时上限：asyncio.gather 并发执行全部清理调用，若某个 MCP 服务的清理请求长时间挂起，会阻塞 AsyncExitStack.__aexit__，拖慢请求结束流程。
3. 双类重复逻辑的维护成本：ConversationContext 与 HarmonyContext 各有一份几乎相同的 init_tool_sessions 实现，本 PR 需要同时修改两处；未来若只改一处，另一条路径会回归。
4. 行为依赖 AsyncExitStack 语义：修复正确性依赖 LIFO 退出顺序（cleanup 先于 session close），若未来有其他代码向同一 stack 压入更晚注册的退出回调，顺序假设可能被破坏。
 - 影响：影响范围集中在 Responses API + MCP 工具调用路径：修复后每个请求对每个被调用过的 MCP 会话只发送一次 cleanup_session 请求，且保证在会话关闭前发送，避免 MCP 服务端收到重复清理或对已关闭会话的无效调用。对使用本地工具（LocalShell、CodeInterpreter）的用户无影响，因为 cleanup_session 内部有 isinstance(tool_session, Tool) 判断跳过非 MCP 会话。代码改动量小（源码 +7/-1），风险低，但属于资源生命周期正确性修复，对依赖 MCP 工具的线上服务有实际收益。
 - 风险标记：清理异常被静默吞掉，并发清理无超时控制，双类重复逻辑需同步维护

关联脉络

- PR #25790 [gpt-oss] disable tool server initialization if no tool in request: 同属 issue #25697 修复系列的前置改动：请求无工具时跳过 tool server 初始化，本 PR 则解决有多个工具时的清理顺序问题，两者互补。
- PR #52521 [Bugfix][Frontend] Keep vLLM MCP x-session-id over client headers: 同一 MCP 工具会话生命周期治理线：该 PR 处理 x-session-id 被客户端 header 覆盖的问题，本 PR 处理清理回调重复注册问题，均影响 MCP 会话的正确创建与销毁。