

PR #1952 完整报告

radixark/miles

Expire finished calls that were never retrieved

合并时间: 2026-09-04 13:31

原文链接: <http://prhub.com.cn/radixark/miles/pull/1952>

执行摘要

- 一句话: 为已完成但未检索的 RPC 调用记录添加 12 小时的自动过期, 解决内存泄漏。
- 推荐动作: 该 PR 是一个清晰、聚焦的 bugfix, 修复了实际存在的内存泄漏问题。实现方案简洁, 测试覆盖全面。值得相关模块 (RPC 服务器、存储) 的维护者精读, 特别是其中关于 TTL 判断逻辑的设计 (`_is_live` 方法)。

功能与动机

PR 提交信息明确指出动机: ‘A record used to live until its result was collected, so a call whose result nobody ever polls for stayed in the store forever.’ 即, 旧的存储机制会导致那些客户端从未轮询 (检索) 结果的已完成调用记录永远留在内存中, 造成长期运行的 RPC 服务内存泄漏。

实现拆解

1. 引入新常量和配置: 在 `miles/utils/workers/rpc/server/store.py` 中新增 `FINISHED_TTL_SECONDS` 常量 (12 小时), 并在 `CallStore.__init__` 方法中增加 `finished_ttl_seconds` 参数, 允许配置该超时时间。
2. 修改记录生命周期判断逻辑: 在 `CallStore` 中新增私有方法 `_is_live(record, now)`, 替代 `_purge_expired` 方法中原有的简单判断。该方法实现了更复杂的过期规则: 未完成的记录永远保留; 已检索的记录根据 `retrieved_ttl` 判断; 未检索的已完成记录根据新的 `finished_ttl` 判断。
3. 更新数据结构: 在 `_CallRecord` 数据类中增加 `finished_at` 字段, 用于在 `finish()` 方法中记录调用完成的时间戳, 这是实现完成时间超时的基础。
4. 完善测试覆盖: 在 `tests/fast/utils/workers/rpc/server/test_store.py` 中新增了大量测试用例 (如 `test_never_retrieved_record_purged_after_finished_ttl`, `test_never_retrieved_records_do_not_accumulate` 等), 全面验证新功能, 包括边界情况 (如失败的结果、检索刚好在截止时间前)。测试辅助函数 `_make_store` 和 `_retrieved_call` 也进行了更新以支持新的 `finished_ttl` 参数。

关键文件:

- `miles/utils/workers/rpc/server/store.py` (模块 RPC 存储; 类别 source; 类型 core-logic; 符号 `CallStore.init`, `CallStore._is_live`, `CallStore.finish`, `_CallRecord`): 核心源码文件, 包含了本次修复的主要逻辑变更: 新增完成时间记录、修改过期判断逻辑。

- tests/fast/utils/workers/rpc/server/test_store.py (模块 测试覆盖; 类别 test; 类型 test-coverage; 符号 _make_store, _retrieved_call, test_never_retrieved_record_purged_after_finished_ttl, test_never_retrieved_records_do_not_accumulate) : 测试文件, 新增了大量测试用例来验证新的过期机制, 是本次修复正确性的重要保障。

关键符号: CallStore.init, CallStore.finish, CallStore._is_live, CallStore._purge_expired

关键源码片段

miles/utils/workers/rpc/server/store.py

核心源码文件, 包含了本次修复的主要逻辑变更: 新增完成时间记录、修改过期判断逻辑。

--- miles/utils/workers/rpc/server/store.py 中的核心逻辑变更 ---

新增常量, 定义已完成记录的默认过期时间 (12 小时)

FINISHED_TTL_SECONDS = 12 * 3600.0

class CallStore:

def __init__(

self,

*,

retrieved_ttl_seconds: float = RETRIEVED_TTL_SECONDS,

finished_ttl_seconds: float = FINISHED_TTL_SECONDS,

) -> None:

self._retrieved_ttl_seconds = retrieved_ttl_seconds

self._finished_ttl_seconds = finished_ttl_seconds # 新增: 存储完成过期时间

self._records: dict[str, _CallRecord] = {}

def finish(self, *, call_id: str, outcome: CallStatusResponse) -> None:

record = self._records[call_id]

if record.outcome is not None:

raise RuntimeError(f"call {call_id} finished twice")

record.outcome = outcome

record.finished_at = time.monotonic() # 新增: 记录完成时的时间戳

record.finished_event.set()

...

def _purge_expired(self) -> None:

now = time.monotonic()

重构过期判断, 委托给新的 _is_live 方法

retained = {call_id: record for call_id, record in self._records.items() if self._is_live(record, now=now)}

...

def _is_live(self, record: _CallRecord, *, now: float) -> bool:

"""判断一条调用记录是否仍然应该保留在内存中。"""

规则 1: 未完成的调用永远保留 (等待结果)

if record.finished_at is None:

```

        return True
# 规则 2: 已检索过的调用, 由检索过期时间 (retrieved_ttl) 决定生命周期
if record.first_retrieved_at is not None:
    return now - record.first_retrieved_at <= self._retrieved_ttl_seconds
# 规则 3: 已完成但未检索的调用, 由完成过期时间 (finished_ttl) 决定生命周期
return now - record.finished_at <= self._finished_ttl_seconds

class _CallRecord:
    finished_event: asyncio.Event
    outcome: CallStatusResponse | None = None
    finished_at: float | None = None # 新增字段
    first_retrieved_at: float | None = None

```

评论区精华

PR 只有审核者 guapisolo 的一条批准评论，其内容指出了本次变更的核心价值：‘fixes the unbounded retention of finished, unpolled RPC calls from #1942 by adding a finished-record TTL’。这确认了本次修改是对之前引入问题的直接修复，且修复被正确地集成到了后续的主干版本中。

- 修复已完成 RPC 调用记录的无界内存保留问题 (correctness): 修复被确认为解决了一个真实的内存泄漏问题，方案（增加 finished TTL）被审核接受。

风险与影响

- 风险：技术风险：风险较低。变更逻辑集中，是增加性修改而非破坏性重构。核心风险点在于默认的 12 小时过期时间是否在所有生产场景下都足够合理，如果客户端有长时间未轮询的合法场景，可能需要调整该值。兼容性风险：CallStore 构造函数新增了带默认值的参数，向后兼容。回归风险：现有测试得到保留并扩展，新测试覆盖了核心逻辑，回归风险可控。
- 影响：对系统影响：提升了 RPC 服务的内存稳定性，防止了因客户端行为（不轮询）导致的内存持续增长，对系统的长期可靠性有积极影响。对用户影响：无直接用户可见的功能变更，但系统将更稳定。对团队影响：为团队解决类似问题提供了模式：即通过多层 TTL 机制管理内存中的数据对象。
- 风险标记：默认超时值需场景适配

关联脉络

- PR #1942（推测）引入或重构 RPC 调用存储机制：审核评论中直接提及 ‘#1942’ 是导致 ‘unbounded retention of finished, unpolled RPC calls’ 问题的源头。本次 PR #1952 是该问题的修复。
- PR #2176（推测）与安全合并检查点相关的 PR: 审核评论中提到 ‘The repair lands before safe-merge checkpoint #2176’，表明 #1952 的修复在更广泛的合并流程中被考虑和验证。