

PR #1947 完整报告

radixark/miles

Refuse a duplicate call id with 409 instead of running it twice

合并时间: 2026-09-04 13:25

原文链接: <http://prhub.com.cn/radixark/miles/pull/1947>

执行摘要

- 一句话: RPC 调用存储拒绝重复 call_id, 返回 409 而非重复执行工作。
- 推荐动作: 该 PR 值得精读。虽然源码变更量小 (+12/-2 行核心代码), 但它修复了 RPC 调用存储的一个重要正确性问题——重复 call_id 导致重复执行。变更方案设计简洁: 在最底层 (store) 抛出语义化异常, 在中间层 (core) 转换为 HTTP 响应, 符合关注点分离原则。测试覆盖极为全面 (从 store 单元测试到 154 行 e2e 测试), 测试用例设计有学习价值, 特别是并发去重和 in-flight 去重的场景。该 PR 是 RPC 子系统规范化进程中的一个关键节点, 与 #1948 (boot uuid 绑定)、#1952 (过期清理) 等 PR 共同完善了调用生命周期管理。

功能与动机

call_id 由客户端生成, 重试提交会以相同 call_id 到达服务端。之前 CallStore.begin() 不做去重检查, 直接覆盖已有记录, 导致同一调用被重复执行。commit message 明确指出: "The call id is generated by the client, so a resubmit is the shape a retried submit would take. The store reserves the id for as long as it retains the outcome, and a second submit under a known id is rejected rather than starting the work again."

实现拆解

1. 新增 DuplicateCallError 异常 (miles/utils/workers/rpc/server/store.py): 在 CallStore 类上方定义轻量异常类, 作为重复调用的语义化错误信号。
2. 修改 CallStore.begin() 加入重复检查 (miles/utils/workers/rpc/server/store.py): 在 _purge_expired() 之后、插入新记录之前, 检查 call_id in self._records, 若已存在则抛出 DuplicateCallError。这确保无论旧调用处于 pending 还是 finished 状态, 重复提交都会被拒绝。
3. 修改 RpcServer.submit_call() 捕获并转换异常 (miles/utils/workers/rpc/server/core.py): 导入 DuplicateCallError, 将 self._store.begin() 调用包裹在 try/except 中, 捕获后通过已有的 reject() 辅助函数返回 HTTP 409, reason 为 "duplicate_call"。
4. 补充 store 层单元测试 (tests/fast/utils/workers/rpc/server/test_store.py): 新增 test_duplicate_pending_call_rejected 和 test_duplicate_finished_call_rejected, 验证 pending 和 finished 状态下的重复 call_id 均被拒绝。

5. 补充 app 层集成测试 (tests/fast/utils/workers/rpc/server/test_app.py) : 新增 TestDuplicateCalls 类和两个 TestProtocolErrors 中的测试方法, 覆盖相同 payload 重提交、不同 payload 重提交、跨方法重用 call_id 三种场景。
6. 新增 e2e 端到端去重测试 (tests/fast/utils/workers/e2e/test_dedup.py) : 全新文件, 通过 TestDuplicateSubmits、TestCallIdIdentity、TestNoRetryAfterAmbiguousSubmit 三个测试类, 覆盖完全相同重提交、in-flight 重复、并发重复 (10 个并发请求只有 1 个被接受)、冲突后原始调用仍可轮询、不同 payload/method/key 顺序的重复拒绝、以及 503/丢包不重试等边界场景。

关键文件:

- miles/utils/workers/rpc/server/store.py (模块 调用存储; 类别 source; 类型 core-logic ; 符号 DuplicateCallError, CallStore.begin) : 核心逻辑变更: CallStore.begin() 新增重复 call_id 检查并引入 DuplicateCallError 异常
- miles/utils/workers/rpc/server/core.py (模块 RPC 服务器; 类别 source; 类型 dependency-wiring; 符号 RpcServer.submit_call) : 依赖接线: RpcServer.submit_call() 捕获 DuplicateCallError 并转换为 HTTP 409 响应
- tests/fast/utils/workers/e2e/test_dedup.py (模块 去重测试; 类别 test; 类型 test-coverage; 符号 TestDuplicateSubmits, TestCallIdIdentity, TestNoRetryAfterAmbiguousSubmit) : 全新 e2e 测试文件, 覆盖重复提交去重的完整场景矩阵

关键符号: CallStore.begin, RpcServer.submit_call, DuplicateCallError

关键源码片段

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

核心逻辑变更: CallStore.begin() 新增重复 call_id 检查并引入 DuplicateCallError 异常

```
class DuplicateCallError(Exception):
    pass

class CallStore:
    def __init__(self, *, retrieved_ttl_seconds: float = RETRIEVED_TTL_SECONDS) -> None:
        self._retrieved_ttl_seconds = retrieved_ttl_seconds
        self._records: dict[str, _CallRecord] = {}

    def begin(self, *, call_id: str) -> None:
        self._purge_expired()

        # 拒绝重复 call_id: 如果该 ID 已存在于记录中 (无论 pending 还是 finished) ,
        # 则拒绝开始新的工作, 防止重试提交导致相同工作被执行两次。
        if call_id in self._records:
            raise DuplicateCallError(f"call {call_id} already submitted")

        self._records[call_id] = _CallRecord(finished_event=asyncio.Event())
        log_structured(
```

```

        logger.debug, tag="rpc", op="call_store", phase="accept",
        call=call_id, tracked=len(self._records)
    )

```

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

依赖接线: RpcServer.submit_call() 捕获 DuplicateCallError 并转换为 HTTP 409 响应

```

def submit_call(self, *, method_name: str, request: SubmitRequest) -> SubmitResponse:
    def reject(*, status_code: int, reason: str, detail: str) -> NoReturn:
        log_structured(
            logger.warning, tag="rpc", op="submit", phase="reject",
            reason=reason, method=method_name, call=request.call_id, error=detail,
        )
        raise HTTPException(status_code=status_code, detail=detail)

    spec = self._specs.get(method_name)
    if spec is None:
        reject(status_code=404, reason="unknown_method", detail=f"unknown rpc method {method_name!r}")

    try:
        kwargs = spec.serializer.decode_query(request.query)
    except ValidationError as e:
        reject(status_code=400, reason="invalid_query", detail=str(e))

    try:
        self._store.begin(call_id=request.call_id)
    except DuplicateCallError as e:
        # call_id 由客户端生成, 重复提交代表重试;
        # 拒绝重复执行, 返回 HTTP 409 Conflict。
        reject(status_code=409, reason="duplicate_call", detail=str(e))

    self._executor.start(
        spec=spec, kwargs=kwargs, call_id=request.call_id,
        finish=functools.partial(self._store.finish, call_id=request.call_id),
    )

    return SubmitResponse()

```

tests/fast/utils/workers/e2e/test_dedup.py

全新 e2e 测试文件, 覆盖重复提交去重的完整场景矩阵

```

class TestDuplicateSubmits:
    async def test_identical_resubmit_returns_409(self, raw, handle, tag) -> None:
        """Resubmitting the same call id and payload fails loudly."""
        first = await _submit(raw, method="demo_count_sync", call_id=tag, query={"tag": tag})
        assert first.json() == {"status": "submitted"}
        await _poll(raw, call_id=tag)
        second = await _submit(raw, method="demo_count_sync", call_id=tag, query={"tag": tag})

```

```

assert second.status_code == 409
assert "already submitted" in second.json()["detail"]
# 关键验证：即使收到 409，原始调用仍只执行一次
assert await handle.report_counter(tag=tag) == 1

async def test_concurrent_duplicates_have_one_acceptance(self, raw, handle, tag) -> None:
    """Concurrent duplicate submits produce one acceptance and nine conflicts."""
    responses = await asyncio.gather(
        *[_submit(raw, method="demo_count_sync", call_id=tag, query={"tag": tag}) for _ in
          range(10)]
    )
    await _poll(raw, call_id=tag)
    # 10 个并发请求中只有 1 个被接受，其余 9 个返回 409
    assert [r.status_code for r in responses].count(200) == 1
    assert [r.status_code for r in responses].count(409) == 9
    assert await handle.report_counter(tag=tag) == 1

async def test_in_flight_duplicate_returns_409_immediately(self, raw, handle, tag) -> None:
    """A duplicate in-flight call fails without waiting for completion."""
    await _submit(raw, method="demo_block_sync", call_id=tag, query={"tag": tag})
    await asyncio.sleep(0.05)
    # 原始调用尚未完成，重复提交应立即返回 409 而非阻塞等待
    duplicate = await asyncio.wait_for(
        _submit(raw, method="demo_block_sync", call_id=tag, query={"tag": tag}),
        timeout=1.0,
    )
    assert duplicate.status_code == 409
    await handle.release(tag=tag)
    assert (await _poll(raw, call_id=tag, timeout=2.0))["status"] == "success"

```

评论区精华

Review 由 guapisolo 审批，状态为 APPROVED。审评论论指出："PR #1947 rejects duplicate call IDs before execution while keeping the original call pollable; no actionable issues found." 审批是在 safe-merge checkpoint 上下文中完成的。PR 本身无 review 评论线程，变更逻辑清晰，无争议点。

- PR 审批 (other): 审批通过，无待解决事项

风险与影响

- 风险:

1. 回归风险 (低)：变更仅在 CallStore.begin() 中增加了一次字典查找和条件判断，对正常路径无影响。已有 _purge_expired() 确保过期记录被清理，不会误拒合法的新 call_id。
2. 兼容性风险 (中)：所有之前依赖重复提交行为的客户端逻辑都会收到 409 而非静默成功。需要确认上游调用方（如 worker_handle 的重试逻辑）能正确处理 409。从 test_dedup.py 中 TestNoRetryAfterAmbiguousSubmit 的测试可见，客户端已经设计为不对 ambiguous 结果重试。

3. 并发安全性（低）：begin() 操作在 RPC 服务器的同步路径中执行（submit_call 是同步方法），在 asyncio 单线程模型下不存在竞态条件。test_concurrent_duplicates_have_one_acceptance 测试验证了并发场景下的正确性。
4. 存储层完整性（低）：DuplicateCallError 仅在 begin() 中抛出，finish() 中已有的 RuntimeError("finished twice") 检查不受影响，两层防御各自独立。

- 影响：

1. 用户体验：调用方在提交重复请求时会立即收到 409 错误和明确的 "already submitted" 详情信息，而非静默执行重复工作。
2. 系统正确性：消除了同一调用被重复执行的 bug，对于推理引擎等有副作用的工作尤为重要（参考近期 PR 中 report_counter 测试验证了执行次数为 1）。
3. 团队影响：这是一个 RPC 子系统的基础正确性修复，影响所有通过 RPC 服务器提交调用的模块。测试覆盖从底层 store 到端到端的完整链路，为后续重构提供了安全网。
4. 性能影响：几乎无影响，仅增加一次字典成员检查（O(1)）。 - 风险标记：客户端行为假设，并发去重边界

关联脉络

- PR #1952 Expire finished calls that were never retrieved: 同属 CallStore 生命周期管理：#1952 为已完成但未检索的调用添加自动过期，本 PR 为相同 call_id 的重复提交添加拒绝机制。两者共同完善了调用记录的生命周期控制。
- PR #1948 Pin a client to one server process with a boot uuid: 同属 RPC 子系统可靠性改进：#1948 通过 boot uuid 防止服务器重启后客户端状态丢失，本 PR 通过去重防止重复执行。两者互补，分别处理连接级和调用级的幂等性问题。
- PR #1950 Support non-finite floats across the wire: 同属 RPC wire 协议层改进：#1950 修复序列化问题，本 PR 修复调用去重问题。均为 RPC 子系统的基础质量提升。