PR #30023 完整报告

sgl-project/sglang

[tracing] sglang tracing v2: support exporting tracing data asynchronously

合并时间：2026-08-10 15:23

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30023

执行摘要

- 一句话：异步导出 OTel 追踪：ZMQ 子进程消解热路径开销
- 推荐动作：值得精读 trace_async.py（进程外重放架构、线程安全 socket、TTL 上下文回收）与 trace.py 的 preset_next_span_id 机制。重点关注三个设计点：一是 " 调用方预生成 span ID + exporter 一次性消费 " 如何在不动 OTel start_span API 的前提下保持 span 树一致；二是 flush 时机选择在 process_batch_result 的 CPU/GPU 重叠点而非固定定时器，属于零成本重叠的典型做法；三是环境变量统一收敛到 envs、双轨选择集中到 req_time_stats.init_trace_ctx 的接入方式，把对业务代码的侵入降到最低。

功能与动机

PR body 明确量化了动机：由于 OTel 的线程安全机制和 exporter 线程（GIL 竞争）导致单个 span 导出产生 50–100 µs 开销，当 batch size 很大时，累积的 CPU 开销会超过 GPU 重叠窗口，阻塞下一次请求的调度与 forward 执行。作者给出的 benchmark（8×H20、Qwen3-32B TP8）显示同步 tracing 在 BS=256 时输出吞吐下降 16.0%、TPOT 上升 30.5%、P99 ITL 恶化 1056%，而异步模式把每 span 开销压到几微秒，让开启全量追踪不再以牺牲吞吐为代价。

实现拆解

1. 新增异步追踪核心模块python/sglang/srt/observability/trace_async.py（新增 1036 行）：TraceReqContextAsync 对外暴露与同步 TraceReqContext 一致的接口（trace_slice、trace_req_start、flush 等），内部把操作追加到内存 op 列表，达到 SGLANG_TRACE_ASYNC_FLUSH_THRESHOLD（默认 100）或收到 flush() 时经线程本地 ZMQ PUSH socket 打包发送；_TraceExporterProcess 是每个 worker 进程一个的 daemon，PULL 端按 _context_id 维护上下文缓存并重放 op，slice_start/slice 消息携带调用方预生成的 span ID，导出前调用 TraceCustomIdGenerator.preset_next_span_id() 保证 span 树与同步模式一致，并有 300 秒 TTL 与每 60 秒一轮的 _cleanup_stale 回收陈旧上下文；socket 管理上按 pid 隔离 endpoint（ipc:///tmp/sglang_trace_{pid}.sock）、fork 后加锁重建 Context、SNDHWM/RCVHWM 均设为 10000 防丢消息。
2. 同步链路最小改造python/sglang/srt/observability/trace.py：TraceCustomIdGenerator 增加 preset_next_span_id() 一次性注入机制（threading.local 存储、消费即清空）；trace_set_thread_info() 末尾触发 _on_thread_info_set 回调，供异步模块把线程标签与 TP/DP/PP rank 转发到 exporter；TraceReqContext 新增 flush() 空实现保持接口一致、

copy_for_thread 不再复制 pid 而由 rebuild_thread_context(pid=...) 按需重建、构造函数新增可选 trace_level 参数让 exporter 端重建的上下文与调用方一致；process_tracing_init 在初始化完成后若开启 SGLANG_TRACE_ASYNC 则自动启动 exporter；OTLP 调度参数从 get_int_env_var 迁移到 envs 统一入口。

3. 请求统计层接入python/sglang/srt/observability/req_time_stats.py：init_trace_ctx 依据 is_async_tracing_available() 在 TraceReqContext 与 TraceReqContextAsync 之间选择；__setstate__ 依据序列化状态里的 is_async 字段重建对应类型；新增 flush_trace_batch(reqs) 帮助函数（防御式 getattr 访问，兼容 mock 与未初始化请求）。
4. 调度器 flush 点python/sglang/srt/managers/scheduler.py：process_batch_result 入口处调用 flush_trace_batch(batch.reqs)——此时上一批 GPU forward 仍在途、CPU 空闲，ZMQ 发送可与 GPU 计算重叠，这是性能收益落地的关键位置。
5. 配置、文档与测试配套：environ.py 新增 SGLANG_TRACE_ASYNC（EnvBool，默认 False）与 SGLANG_TRACE_ASYNC_FLUSH_THRESHOLD（EnvInt，默认 100）；docs/docs/references/production_request_trace.mdx 新增 Async Tracing 章节与环境变量调优表，environment_variables.mdx 补充两行变量说明；test/registered/observability/test_tracing.py 新增 TestTraceServerAsync（继承 TestTraceServer，以 SGLANG_TRACE_ASYNC=1 启动服务器，只保留最全面的 test_trace_level_3，其余用例置 None 屏蔽）。

关键文件：

- python/sglang/srt/observability/trace_async.py（模块 可观测性；类别 source；类型 core-logic；符号 TraceReqContextAsync, _TraceExporterProcess, start_trace_exporter, stop_trace_exporter）：本 PR 的核心新模块（新增 1036 行）：TraceReqContextAsync 缓冲 trace 操作、_TraceExporterProcess 守护进程重放并导出 OTel span，内含线程本地 ZMQ socket、fork 重建、span ID 预生成、TTL 清理等关键机制。
- python/sglang/srt/observability/trace.py（模块 可观测性；类别 source；类型 core-logic；符号 TraceCustomIdGenerator, preset_next_span_id, rebuild_thread_context, flush）：同步链路被最小化改造（+64/-12）：TraceCustomIdGenerator.preset_next_span_id() 一次性 span ID 注入机制、_on_thread_info_set 线程信息回调、flush() 空实现、trace_level 参数，并迁移到 envs 统一配置入口，是异步导出与既有同步实现兼容的关键。
- python/sglang/srt/observability/req_time_stats.py（模块 可观测性；类别 source；类型 core-logic；符号 flush_trace_batch, init_trace_ctx, setstate）：异步与同步 trace 上下文的分派点（+42/-9）：init_trace_ctx 按 is_async_tracing_available() 选择 TraceReqContextAsync，__setstate__ 按 is_async 字段重建，新增 flush_trace_batch 供调度器调用。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 process_batch_result）：核心路径注入点（+4/-0）：process_batch_result 入口调用 flush_trace_batch(batch.reqs)，让 ZMQ 发送与下一批 GPU forward 重叠，这是性能收益落地的关键位置。
- python/sglang/srt/environ.py（模块 运行配置；类别 source；类型 configuration；符号 SGLANG_TRACE_ASYNC, SGLANG_TRACE_ASYNC_FLUSH_THRESHOLD）：新增

SGLANG_TRACE_ASYNC（EnvBool，默认 False）与 SGLANG_TRACE_ASYNC_FLUSH_THRESHOLD（EnvInt，默认 100）两个环境变量定义，是整个开关与调优参数的唯一边界。

- test/registered/observability/test_tracing.py（模块 追踪测试；类别 test；类型 test-coverage；符号 TestTraceServerAsync, setUpClass）：新增 TestTraceServerAsync（+34/-0）：以 SGLANG_TRACE_ASYNC=1 启动服务器并复用同步用例框架，只跑最全面的 test_trace_level_3，验证异步导出链路端到端可用。
- docs/docs/references/production_request_trace.mdx（模块 使用文档；类别 other；类型 documentation）：新增 Async Tracing 章节：说明启用方式、工作原理（root span 在调用进程、span ID 预生成、线程信息回调）与两个环境变量的调优表。
- docs/docs/references/environment_variables.mdx（模块 使用文档；类别 other；类型 documentation）：环境变量参考表补充 SGLANG_TRACE_ASYNC 与 SGLANG_TRACE_ASYNC_FLUSH_THRESHOLD 两行说明，保持文档与实现同步。

关键符号：preset_next_span_id, generate_span_id, flush_trace_batch, start_trace_exporter, stop_trace_exporter, maybe_start_trace_exporter, _get_zmq_socket, _forward_thread_info_to_exporter, is_async_tracing_available, _TraceExporterProcess.run, TraceReqContextAsync.flush, init_trace_ctx, process_batch_result

关键源码片段

python/sglang/srt/observability/trace_async.py

本 PR 的核心新模块（新增 1036 行）：TraceReqContextAsync 缓冲 trace 操作、_TraceExporterProcess 守护进程重放并导出 OTel span，内含线程本地 ZMQ socket、fork 重建、span ID 预生成、TTL 清理等关键机制。

```
# python/sglang/srt/observability/trace_async.py
# 线程本地 PUSH socket：每个线程独立连接，避免多线程共享 socket 的并发问题；
# 进程 fork 后（_init_pid != 当前 pid）重建 ZMQ Context，双重检查保证只重建一次。
_socket_lock = threading.Lock()


def _get_zmq_socket():
    """返回线程本地的 ZMQ PUSH socket，fork 后自动重建。"""
    global _zmq_context, _init_pid

    if not _zmq_available or _zmq_endpoint is None:
        return None

    cur_pid = os.getpid()
    if _init_pid != cur_pid:
        with _socket_lock: # 多线程同时检测到 fork 时的竞争保护
            if _init_pid != cur_pid:
                _zmq_context = zmq.Context()
                _init_pid = cur_pid
```

```
            if hasattr(_thread_local, "socket"):
                _thread_local.socket = None

    if not hasattr(_thread_local, "socket") or _thread_local.socket is None:
        from sglang.srt.utils.network import get_zmq_socket

        _thread_local.socket = get_zmq_socket(
            _zmq_context, zmq.PUSH, _zmq_endpoint, bind=False
        )
        # LINGER 保证退出时把积压消息发完；SNDHWM 抬高水位，降低高吞吐下丢消息概率
        _thread_local.socket.setsockopt(zmq.LINGER, 5000)
        _thread_local.socket.setsockopt(zmq.SNDHWM, 10000)

    return _thread_local.socket


def start_trace_exporter(otlp_endpoint, server_name, trace_modules=None):
    """启动 exporter 守护进程；重复调用安全（已有存活进程时直接返回）。"""
    global _exporter_process, _zmq_endpoint, _zmq_context, _init_pid

    if not _zmq_available:
        logger.warning("pyzmq not installed — cannot start async trace exporter")
        return False

    with _exporter_lock: # 防止初始化期多线程并发启动多个 exporter
        if _exporter_process is not None and _exporter_process.is_alive():
            return True

        # 按 pid 隔离 endpoint，避免多进程（scheduler、tokenizer）互踩同一 socket 文件
        zmq_endpoint = f"ipc:///tmp/sglang_trace_{os.getpid()}.sock"
        _zmq_endpoint = zmq_endpoint
        _zmq_context = zmq.Context()
        _init_pid = os.getpid()

        _exporter_process = _TraceExporterProcess(
            otlp_endpoint=otlp_endpoint,
            server_name=server_name,
            zmq_endpoint=zmq_endpoint,
            trace_modules=trace_modules,
        )
        _exporter_process.start()

    # 注册回调：此后本进程内 trace_set_thread_info() 会自动把线程信息转发给 exporter
    import sglang.srt.observability.trace as _tm

    _tm._on_thread_info_set = _forward_thread_info_to_exporter
    return True
```

python/sglang/srt/observability/trace.py

同步链路被最小化改造（+64/-12）：TraceCustomIdGenerator.preset_next_span_id() 一次性 span ID 注入机制、_on_thread_info_set 线程信息回调、flush() 空实现、trace_level 参数，并迁移到 envs 统一配置入口，是异步导出与既有同步实现兼容的关键。

```
# python/sglang/srt/observability/trace.py
# 默认 IdGenerator 在多个 TP scheduler 进程间可能产生重复 trace ID，因此 SGLang 自实现。
# 异步模式新增 " 预设下一条 span ID" 机制：调用方进程预生成 span ID 并随 op 消息发给
# exporter；exporter 在 start_span() 前调用 preset_next_span_id()，让下一次
# generate_span_id() 消费该 ID（一次性），从而保证导出后的 span 树与同步模式完全一致。
class TraceCustomIdGenerator(id_generator.IdGenerator):
    # 线程本地存储，并发调用互不干扰；exporter 进程是单线程的，无需额外加锁
    _preset_local = threading.local()

    def __init__(self):
        super().__init__()
        self.local_random = random.Random()
        self.local_random.seed(time.time())

    def generate_trace_id(self) -> int:
        return self.local_random.getrandbits(64)

    def generate_span_id(self) -> int:
        # 有预设则消费一次并立刻清空，之后回退到随机生成
        preset = getattr(self._preset_local, "span_id", None)
        if preset is not None:
            self._preset_local.span_id = None
            return preset
        return self.local_random.getrandbits(64)

    @classmethod
    def preset_next_span_id(cls, span_id: int):
        """为下一次 start_span() 注入预生成的 span ID（一次性消费）。"""
        cls._preset_local.span_id = span_id
```

python/sglang/srt/observability/req_time_stats.py

异步与同步 trace 上下文的分派点（+42/-9）：init_trace_ctx 按 is_async_tracing_available() 选择 TraceReqContextAsync，__setstate__ 按 is_async 字段重建，新增 flush_trace_batch 供调度器调用。

```
# python/sglang/srt/observability/req_time_stats.py
# 在 CPU/GPU 重叠点主动冲刷一批请求的缓冲 trace 操作：
# ZMQ 发送是 CPU 操作，放在 GPU forward 在途时执行即可 " 免费 " 重叠。
def flush_trace_batch(reqs: List[Any]):
    """Proactively flush buffered trace ops for a batch of requests."""
    if reqs is None or not get_global_tracing_enabled():
        return
    for req in reqs:
        # 防御式访问：测试 mock 或未初始化请求可能没有这些属性
        time_stats = getattr(req, "time_stats", None)
```

```
if time_stats is not None:
    trace_ctx = getattr(time_stats, "trace_ctx", None)
    if trace_ctx is not None:
        trace_ctx.flush()
```

评论区精华

gemini-code-assist[bot] 的高优先级意见集中在并发安全与资源生命周期：_get_zmq_socket 在多线程 + fork 场景下可能竞争重建全局 _zmq_context，exporter 主循环收到非 dict 消息会崩溃并绕过资源清理。这两条在最终代码中分别以 _socket_lock 双重检查、isinstance(msg, dict) 校验 + warning 后 continue 落实。

ShangmingCai 质疑 "Preset span-ID leaks when a slice is filtered out?"，作者回应这是设计使然：每个 slice_start/slice op 都携带新的 preset span ID 并覆盖陈旧值；无 preset 的 op 靠 root_span_carrier 与 last_span_context 完成跨进程链接，消费泄漏 ID 功能上无害。此点以设计澄清收尾，未改代码。

ShangmingCai 另提两点均已修复：一是把 trace_level 传入 exporter 的 init_args 并对 preset 做防御性清理（最终 TraceReqContext.__init__ 新增 trace_level 参数）；二是两次要求不要绕过 environ.py 的 EnvBool 直接读 os.environ（最终统一走 envs.SGLANG_TRACE_ASYNC.get()）。

gemini 还建议用 random.getrandbits(64) 替代 id(self) 作 _context_id 以避免内存地址复用碰撞，以及为 traceparent 解析加 ValueError 防御、__del__ 兼容解释器关闭期；这些点在当前提供的最终代码片段中无法完全确认落实状态，是合并前值得留意的遗留项。

- ZMQ socket 的 fork/ 多线程初始化竞争 (correctness): 最终代码采用 _socket_lock 加双重检查（内层再次判断 pid），线程局部 socket 按需重建，评审意见已落实。
- exporter 主循环健壮性与资源泄漏 (correctness): 最终代码在主循环内先校验 isinstance(msg, dict)，对非法消息 warning 后 continue，避免单条坏消息拖垮 exporter。
- preset span ID 在 slice 被过滤时的泄漏 (design): 判定为设计特性而非 bug，未改代码；但 reviewer 同时建议在 start_span 外围 try/finally 或每 op 开头重置 _preset_local.span_id 作为防御（相关 trace_level 部分作者已修）。
- trace_level 传入 exporter 的 init_args 与 preset 防御性清理 (design): 作者回复 fixed：TraceReqContext.__init__ 新增可选 trace_level 参数，允许显式传入以覆盖全局 level。
- 环境变量读取应走 envs 而非 os.environ (style): 作者两次回复 fixed：最终 process_tracing_init 与 trace_async 均使用 envs.SGLANG_TRACE_ASYNC.get()，exporter 子进程内再 pop 该变量防止递归启动。
- ZMQ HWM 与 stop 超时处理 (performance): 最终代码两端 socket 均设置 HWM 10000；stop 逻辑在 join 后显式检查 is_alive() 再 terminate + join(timeout=2)。
- id(self) 内存复用导致 context ID 碰撞 (correctness): 评审建议合理；但当前提供的最终代码片段未覆盖相关行，无法确认合并前是否全部落实，属于遗留待确认项。

风险与影响

- 风险：

- 核心路径新增调用点：scheduler.process_batch_result 每个迭代都会执行 flush_trace_batch 的遍历（即使 tracing 关闭也至少有一次 get_global_tracing_enabled() 判断），对无 tracing 部署是微小常数开销；开启 tracing 后若 ZMQ 出现背压，flush 阻塞可能重新引入 P99 ITL 抖动（BS=512 下 P99 ITL 仍为 +24.1%）。
- 常驻进程与 IPC 资源生命周期：每个 worker 进程（scheduler、tokenizer 等）各起一个 daemon exporter 与一个 IPC socket 文件（/tmp/sglang_trace_{pid}.sock），进程异常退出或 stop_trace_exporter 清理不及时会遗留 socket 文件与僵尸进程；fork 后 socket 重建是最需要测试覆盖的并发路径。
- 消息丢失：send_pyobj(..., zmq.NOBLOCK) 在 HWM（10000）打满时会直接丢弃消息，高吞吐且 exporter 处理慢于生产速率时 trace 数据不完整（不影响推理正确性，但影响可观测性保真度）。
- 跨进程契约耦合：traceparent 解析（int(tp_parts[1], 16)）缺少 ValueError 防御，畸形上游 header 可能在序列化时抛异常；异步导出依赖 __getstate__/__setstate__ 的字段契约（is_async、last_span_context），协议演进时容易前后端不匹配。
- 双轨维护负担：mooncake、diffusion 仍用同步 TraceReqContext，开启 SGLANG_TRACE_ASYNC 时它们会额外 spawn 一个空转 exporter 进程（PR body 已声明该行为）。
- 测试覆盖有限：TestTraceServerAsync 只跑 test_trace_level_3，未覆盖 exporter 崩溃恢复、IPC 文件清理、并发 flush、TTL 清理等路径。
- 影响：对用户 / 运维：新增 SGLANG_TRACE_ASYNC、SGLANG_TRACE_ASYNC_FLUSH_THRESHOLD 两个环境变量，生产环境可在近似零开销（BS=256 下 -0.05% 吞吐）下开启全量 OTel 追踪，缓解原先 " 开 tracing 掉 16% 吞吐 " 的取舍。

对系统：开启后每个 worker 进程多一个 daemon 子进程与 ZMQ IPC；默认关闭时同步路径完全不变，且异步导出的 span 树与同步模式结构一致（span ID 预生成保证），对下游 trace 分析平台透明。

对团队：observability 模块新增 1000+ 行核心实现与同步 / 异步双轨维护成本；该实现也为后续把 mooncake、diffusion 等模块迁移到异步追踪提供了现成范式与文档。

- 风险标记：核心路径新增调用点，新增常驻子进程与 IPC 资源，ZMQ 高负载下丢消息，默认关闭需显式开启，异步路径测试覆盖有限，跨进程序列化契约耦合

关联脉络

- PR #34095 config: the runner and scheduler read resolved config from the bags: 同样改动 scheduler.py，且与本 PR 一样把配置读取收敛到统一入口；本 PR 的 trace.py 最终以 envs 方式读取 OTLP 参数，与该配置系列方向一致。
- PR #34096 config: the KV-cache configurator reads the bags: 配置袋（统一配置读取）系列的一部分，与本 PR 的 environ.py 改动、trace.py 的 envs 迁移处于同一重构脉络。
- PR #34169 Add skill for the logprob consistency tests: 同为可观测性 / 调试基础设施的沉淀（观测技能文档），与 tracing 双轨建设同属 observability 演进线。