PR #21254 完整报告

sgl-project/sglang

feat: add OpenTelemetry tracing to DiffGenerator

合并时间：2026-04-24 00:25

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/21254

执行摘要

- 一句话：为 DiffGenerator 添加 OpenTelemetry 追踪基础设施
- 推荐动作：此 PR 值得精读，尤其关注：1）如何将 OTel 追踪集成到多进程架构中（worker 进程独立初始化）；2）通过上下文管理器封装复用追踪逻辑；3）跨分解角色传播追踪上下文的序列化方案；4）轻量级进程内 OTLP 收集器的实现（替代 Docker 依赖）。

功能与动机

PR 描述指出需要为 diffusion/multimodal generation 子系统启用 OpenTelemetry 追踪，以实现与 SRT 运行时的功能对等。下游消费者（如 Dynamo）需要 diffusion 路径中的追踪支持，因此希望尽早落地基础架构。

实现拆解

1. 服务端配置与进程级初始化 - 修改 python/sglang/multimodal_gen/runtime/server_args.py，新增 --enable-trace 和 --otlp-traces-endpointCLI 标志，镜像 SRT 的配置。 - 在 DiffGenerator.from_server_args() 中初始化 OTel 导出器，服务名 sglang-diffusion，线程标签 DiffGenerator。 - GPU worker 进程（gpu_worker.py）中为每个 worker 初始化追踪，线程标签 DiffWorker_rank{N}。 - launch_server.py 将追踪配置传递给 worker 进程。
2. 请求级追踪上下文 - 在 python/sglang/multimodal_gen/runtime/entrypoints/utils.py 的 prepare_request 中，当追踪启用时创建 TraceReqContext，并链接外部 W3C 追踪头。 - python/sglang/multimodal_gen/runtime/schedule_batch.py 的 Req 类新增 trace_ctx 字段，默认使用无操作的 TraceNullContext。 - 在 OpenAI 兼容的 image/video API 端点（image_api.py、video_api.py）中提取 traceparent/tracestate 请求头。
3. 跨度（Span）插装 - 引入 python/sglang/multimodal_gen/runtime/utils/trace_wrapper.py，提供 DiffStage 命名常量（scheduler_dispatch、gpu_forward）和上下文管理器 trace_req、trace_slice，简化跨度生命周期管理。 - managers/scheduler.py 中 scheduler_dispatch 跨度包裹 forward 调度。 - managers/gpu_worker.py 中 gpu_forward 跨度包裹 pipeline 前向调用。 - diffusion_generator.py 的 finally 块中调用 trace_req_finish() 关闭跨度。 - 所有导入延迟，避免追踪禁用时的依赖膨胀。
4. 分解角色间的追踪上下文传播 - 修改 python/sglang/multimodal_gen/runtime/disaggregation/scheduler_mixin.py，在 extract_transfer_fields 中将 TraceReqContext 序列化为 JSON 安全的 _trace_state（W3C 载体）通过标量字段传递，接收方在

_build_disagg_req 中通过 __setstate__ 重建上下文。 - 新增 _disagg_trace_dispatch 辅助函数包裹执行前向，确保跨度在分解角色中正确嵌套。

5. 测试覆盖 - 新增 python/sglang/test/otel_collector.py：轻量级进程内 OTLP 收集器，替代 Docker-based otel-collector，用于测试断言。 - 新增 python/sglang/multimodal_gen/test/unit/test_disagg_trace.py：单元测试验证分解追踪传播的序列化 / 反序列化契约。 - 新增 python/sglang/multimodal_gen/test/server/test_tracing.py：集成测试，启动带追踪的扩散服务器，发送请求并验证期望跨度被导出。 - 修改 test/registered/observability/test_tracing.py：移除重复的收集器实现，导入共享的 LightweightOtlpCollector。

代码示例：trace_wrapper.py 核心逻辑

代码示例：分解标量字段中的追踪状态序列化

已知限制（后续跟进）

- trace_req_finish() 尚未附加跨度属性（模型名、生成参数、延迟分解）。
- 无 /set_trace_level 端点用于动态追踪详细级别控制。
- Mesh API 未插装。

关键文件：

- python/sglang/multimodal_gen/runtime/utils/trace_wrapper.py（模块 追踪工具；类别 source；类型 core-logic；符号 DiffStageConfig, DiffStage, trace_req, trace_slice）：新增核心文件，定义扩散追踪的上下文管理器、命名阶段常量，统一了追踪调用方式。
- python/sglang/test/otel_collector.py（模块 测试工具；类别 test；类型 test-coverage；符号 Span, LightweightOtlpCollector, _try_grpc_server, TraceServicer）：新增进程内轻量级 OTLP 收集器，替代 Docker 依赖，用于测试断言。
- python/sglang/multimodal_gen/runtime/disaggregation/scheduler_mixin.py（模块 分解调度；类别 source；类型 dependency-wiring；符号 _disagg_trace_dispatch）：修改分解调度器 mixin，实现追踪上下文在编码器 / 去噪器 / 解码器角色间的 JSON 安全传播。
- python/sglang/multimodal_gen/test/server/test_tracing.py（模块 追踪测试；类别 test；类型 test-coverage；符号 tracing_env, _generate_image, _wait_for_spans, test_spans_exported）：新增集成测试，启动扩散服务器并验证跨度导出，验证端到端追踪。
- python/sglang/multimodal_gen/test/unit/test_disagg_trace.py（模块 分解追踪测试；类别 test；类型 test-coverage；符号 _enable_minimal_otel, _traceparent_from, _roundtrip_scalar_fields, TestDisaggTracePropagation）：新增单元测试，验证分解场景下追踪上下文的序列化 / 反序列化契约。
- test/registered/observability/test_tracing.py（模块 追踪测试；类别 test；类型 test-coverage；符号 Span, LightweightOtlpCollector, init, _try_grpc_server）：修改已有追踪测试，将 LightweightOtlpCollector 提取到公共工具，减少重复。
- python/sglang/multimodal_gen/runtime/entrypoints/openai/image_api.py（模块 API 入口；类别 source；类型 entrypoint）：修改入口点，提取 W3C 追踪头部，创建 TraceReqContext。

- python/sglang/multimodal_gen/runtime/managers/scheduler.py（模块 调度器；类别 source；类型 dependency-wiring）：修改调度器，添加 scheduler_dispatch 跨度并传递追踪上下文。
- python/sglang/multimodal_gen/runtime/entrypoints/utils.py（模块 请求处理；类别 source；类型 dependency-wiring）：添加 prepare_request 函数中的追踪上下文创建逻辑。
- python/sglang/multimodal_gen/runtime/entrypoints/diffusion_generator.py（模块 生成器；类别 source；类型 dependency-wiring）：修改生成器，在 finally 块中关闭追踪，集成 trace_req 上下文管理器。
- python/sglang/multimodal_gen/runtime/managers/gpu_worker.py（模块 GPU worker；类别 source；类型 dependency-wiring）：修改 GPU worker，添加 gpu_forward 跨度，包裹 pipeline 前向调用。

关键符号：DiffStageConfig.init, DiffStage.SCHEDULER_DISPATCH, DiffStage.GPU_FORWARD, trace_req, trace_slice, LightweightOtlpCollector.init, LightweightOtlpCollector._try_grpc_server, LightweightOtlpCollector._handle_trace_request, SchedulerDisaggMixin._disagg_trace_dispatch, extract_transfer_fields, _build_disagg_req, prepare_request, test_spans_exported, test_spans_without_traceparent, test_batch_requests, TestDisaggTracePropagation.test_tracing_disabled_omits_trace_state, TestDisaggTracePropagation.test_tracing_enabled_state_roundtrip, TestDisaggTracePropagation.test_build_disagg_req_installs_rebuilt_ctx, TestDisaggTracePropagation.test_build_disagg_req_falls_back_when_tracing_off

关键源码片段

python/sglang/multimodal_gen/runtime/disaggregation/scheduler_mixin.py

修改分解调度器 mixin，实现追踪上下文在编码器 / 去噪器 / 解码器角色间的 JSON 安全传播。

```
# 导出追踪状态到标量字段
trace_ctx = getattr(req, "trace_ctx", None)
if trace_ctx is not None and getattr(trace_ctx, "tracing_enable", False):
    try:
        trace_state = trace_ctx.__getstate__()
        if trace_state and trace_state.get("tracing_enable"):
            scalar_fields["_trace_state"] = trace_state
    except Exception as e:
        logger.debug("Failed to export trace state: %s", e)

# 接收方重建追踪上下文
trace_state = scalar_fields.pop("_trace_state", None)
if trace_state is not None:
    try:
        rebuilt_ctx = object.__new__(TraceReqContext)
        rebuilt_ctx.__setstate__(trace_state)
        req.trace_ctx = rebuilt_ctx
    except Exception as e:
```

```
            logger.debug("Failed to rebuild trace context: %s", e)

    # 使用 _disagg_trace_dispatch 包装前向执行
    with self._disagg_trace_dispatch(req):
        self.worker.execute_forward([req], return_req=True)
```

评论区精华

1. 将手动 span 操作重构为上下文管理器：

sufeng-buaa: "Could we implement this using a context manager and put the related implementations in one place, for example in python/sglang/multimodal_gen/runtime/utils/trace_wrapper.py? Then this can be rewritten as with trace_slice(req.trace_ctx, DiffStag.GPU_FORWARD):" 作者接受并实现，创建了 trace_wrapper.py 和 DiffStageConfig/DiffStage。

2. 定义 DiffStageConfig 为冻结数据类而非枚举：

sufeng-buaa: "I think defining it this way is better, since we wouldn't need to pass the level every time." 建议使用冻结数据类，作者采纳。

3. 避免延迟导入：

mickqian: "please avoid lazy imports as it introduce runtime overhead and hide possible exceptions." 作者移除所有延迟导入。

4. 测试集成与 CI：

sufeng-buaa: "I recently submitted a PR to integrate tracing into the ci(#21740)... After my PR is merged, you can add test cases directly and start up the sglang server for real testing." 作者等待后添加了集成测试，并将 LightweightOtlpCollector 提取到公共工具。

5. 关于是否拆分为两个 PR 的讨论：

mickqian: "if this is not related to sglang-diffusion, please split into two PRs" 针对 test/registered/observability/test_tracing.py 的修改。最终保留在一个 PR 中。

- 使用上下文管理器重构 Span 操作 (design): 作者采纳建议，创建了 trace_req 和 trace_slice 上下文管理器，以及 DiffStageConfig 数据类。
- DiffStageConfig 的定义方式 (design): 作者采纳，将枚举改为数据类 DiffStageConfig。
- 避免延迟导入 (performance): 作者移除所有延迟导入，改为模块级别导入。
- 测试集成与 CI (testing): 作者等待 PR 合并后添加了集成测试，并提取公共收集器。
- 是否拆分 PR (other): 作者和 sufeng-buaa 均认为提取公共基础设施有助于减少重复，最终保留在一个 PR 中。

风险与影响

- 风险：

    - 默认关闭，零运行时影响：--enable-trace 为 opt-in，且导入使用延迟加载（最终移除），禁用时无任何追踪代码执行。
    - 分解序列化风险：_trace_state 序列化 / 反序列化有异常保护，测试覆盖了禁用和启用场景以及 JSON 往返。
    - 测试基础设施共享：LightweightOtlpCollector 从 test/registered/observability/test_tracing.py 提取至 python/sglang/test/otel_collector.py，原有测试导入更新，风险低。
    - 跨进程跨度关联：通过 W3C traceparent 传递，保证正确性依赖于 OTel SDK 的一致性。
- 影响：
    - 用户影响：启用追踪后，diffusion 请求将产生 OTel 跨度，可被标准 OTel 收集器消费，便于调试和监控。默认关闭，无影响。
    - 系统影响：新增依赖（opentelemetry-api 和 opentelemetry-sdk）仅在启用追踪时加载。测试框架新增共享收集器，减少依赖。
    - 团队影响：为后续细粒度跨度属性、动态追踪级别控制提供了基础。
    - 风险标记：新增追踪基础设施，跨进程序列化，测试覆盖依赖 CI，默认关闭零风险

关联脉络

- PR #21740 integrate tracing into the ci: 此 PR 提供测试框架，作者后续添加的集成测试依赖于该 PR 的 CI 基础设施。