

PR #39014 完整报告

sgl-project/sglang

[Router] Count open HTTP exchanges until their response body finishes

合并时间: 2026-09-16 15:04

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39014>

[Router] 在途 HTTP 交换计数: 随响应体结束才释放 (PR #39014)

一、执行摘要

本 PR 为 sgl-router 新增了一个进程内在途计数 `server::inflight`: 在边缘中间件 `count_requests` 上入场计数, 并把 RAII 守卫搬进响应体, 直到正文写完 (或被客户端放弃导致 body 被 drop) 才释放。它测的是 `axum::serve(...).with_graceful_shutdown(...)` 真正等待的量, 而不是 `ActiveLoadRegistry::inflight_count` 那种只覆盖被代理请求的口径。改动对线上流量完全透明, 是 router 优雅关闭 drain 三段式 (#39014 → #39015 → #39016) 的第 1 块, 并夹带一行测试 fixture 修复让 `main` 上 `pd_protocol_binding.rs` 的断编译恢复。

二、功能与动机

动机可以浓缩为 PR body 里的一句判断:

"This is the number `axum::serve(...).with_graceful_shutdown(...)` actually waits on, and nothing currently reports it."

现有的最近似指标 `ActiveLoadRegistry::inflight_count` 回答的是另一个问题 —— 它数 proxied requests, 所以有两处结构性盲区:

1. 非代理路由上的交换 (`/metrics`、`/readyz`、`/healthz`、404) 完全不在它的视野里;
2. 更关键的是 流式补全的正文窗口: chat completion 立刻回 header、随后写数分钟, 而计数器在 handler 退出时释放, 于是整个 drain 窗口都读出 0。

作者对后果的描述是: 只报 `active_load` 的心跳会对一个「几分钟后将被 SIGKILL 且仍有在途工作」的 pod 说 "0 in flight"。因此本 PR 先只造「仪器」、不造「政策」: 不改 wire、不改响应、也不接消费方, 等 #39016 的 drain 来读它。

另外, PR 顺带修复 `main` 的断编译: fixture (#39006) 与 `sampling_overrides` 字段 (#39000) 在同一窗口落地, 各自对当时的 `main` 是绿的, 于是没有任何 merge 报冲突, 结果是 `cargo test` 与 `cargo clippy --all-targets` 在 `main` 及所有从它切出的 router 分支上失败。

三、实现拆解

1. 计数原语 (`src/server/inflight.rs`, 新增 +178): `InflightHttp` 内含一个 `AtomicUsize`, `count()` 以 `Relaxed` 读取 (注释论证: 诊断量纲, 读者与增量竞争只会晚一个 tick, 不会读到撕裂值)。`enter()` 返回 `InflightGuard`, `Drop` 中 `fetch_sub` —— 选择守卫而非显式

increment/decrement, 是为了覆盖「响应构造前请求 future 就被取消」的情形。

TrackedBody 包装 axum::body::Body 并持有守卫, poll_frame 利用 Body / InflightGuard 均为 Unpin 的事实做普通字段借用转发 (无需 pin-project), is_end_stream / size_hint 原样透传。对外仅暴露 track_body(body, guard)。

- 边缘接入 (src/server/app.rs) : count_requests 是唯一能看到每条路由上每次交换的位置。改动为两处: record_ingress 之后 let inflight = ctx.inflight_http.enter();; 结尾由返回 resp 改为 resp.map(|body| track_body(body, inflight))。函数注释写明了理由 —— 「a streaming completion has barely started when this function returns」。
- 上下文接线 (src/server/app_context.rs、src/server/mod.rs) : AppContext 新增 pub inflight_http: Arc<InflightHttp> (注释直接对标 active_load 只覆盖被代理子集), 在 with_active_load 与 #[cfg(test)] stub() 两处都初始化, 保证测试框架下可用; mod.rs 增加 pub mod inflight;。
- 依赖与锁文件 (Cargo.toml、Cargo.lock) : 显式新增 http-body = "1", 注释说明 Frame / SizeHint 不经 axum::body 重导出, 因此实现 body 包装器必须直接命名该 crate; 锁文件相应加一行。
- 测试配套 (源码与测试同步改动) : 单元层面 4 个测试覆盖「守卫 drop 释放」「流式 body 保持计数到最后一帧」「被放弃的 body 释放计数」「普通 body 原样透传」; 集成层面在 tests/proxy/graceful_shutdown.rs 新增 wait_for_inflight_http 轮询助手 (消除客户端读完最后一字节与服务端守卫 drop 之间的竞态), 以及两个走真实 build_router + axum::serve 栈的测试: inflight_http_counts_a_streaming_response_until_its_body_fini shes (header 到手、正文还有 ~480 ms 时断言为 1) 与 inflight_http_returns_to_zero_ after_non_proxied_routes (遍历 5 个非代理路径后回到 0)。
- 夹带的编译修复: tests/proxy/pd_protocol_binding.rs 的 config() 补 sampling_overrides: Default::default(), 与仪表逻辑无关, 是让本 PR 以及所有 router 分支恢复绿灯的前提。

关注面	文件	关键符号
计数语义	src/server/inflight.rs	InflightHttp::enter、InflightGuard::drop、track_body、TrackedBody::poll_frame
接线点	src/server/app.rs	count_requests
共享状态	src/server/app_context.rs	AppContext.inflight_http
验证	tests/proxy/graceful_shutdown.rs	wait_for_inflight_http、两条 inflight_http_* 测试
解锁	tests/proxy/pd_protocol_binding.rs	config()

experimental/sgl-router/src/server/inflight.rs

全新核心模块：定义 `InflightHttp` (`AtomicUsize` 诊断 gauge)、`InflightGuard` (`Drop` 释放，兼容请求被取消)、`TrackedBody` / `track_body` (把计数附着在响应体上，直到最后一帧或被 `drop`)。还内嵌 4 个单元测试锁定释放语义，是整条 drain 链路的度量地基。

```
// experimental/sgl-router/src/server/inflight.rs (新增)
// 计数对象：一个进程内原子计数器，外加 RAII 守卫与响应体包装器。
// 语义边界：这里数的是 axum 优雅关闭真正等待的量 —— 所有路由上仍然打开的
// HTTP 交换，而不是 ActiveLoadRegistry 只覆盖的「被代理请求」子集。

#[derive(Debug, Default)]
pub struct InflightHttp {
    open: AtomicUsize,
}

impl InflightHttp {
    pub fn new() -> Arc<Self> {
        Arc::new(Self::default())
    }

    /// 只读快照。Relaxed 足够：这是诊断量纲，读者与增量竞争时最多晚一个 tick
    /// 看到新值，绝不会读到撕裂值。
    pub fn count(&self) -> usize {
        self.open.load(Ordering::Relaxed)
    }

    /// 进入一次交换：计数 +1 并返回守卫，守卫 drop 时自动 -1。
    /// 用守卫而不是显式的 increment/decrement 对，是因为请求 future 可能在
    /// 响应构造完成前就被取消（客户端提前断开），那时没有地方调用 decrement。
    pub fn enter(self: &Arc<Self>) -> InflightGuard {
        self.open.fetch_add(1, Ordering::Relaxed);
        InflightGuard(Arc::clone(self))
    }
}

pub struct InflightGuard(Arc<InflightHttp>);

impl Drop for InflightGuard {
    fn drop(&mut self) {
        self.0.open.fetch_sub(1, Ordering::Relaxed);
    }
}

/// 本 PR 的关键设计：计数附着在响应体上，而不是 handler 的 future 上。
/// 流式补全 (SSE) 在 header 就绪时就返回 handler，正文还要继续写很久；
/// 若在 handler 返回处释放计数，整段 drain 等待窗口都会读出 0。
struct TrackedBody {
    inner: Body,
```

```

    _guard: InflightGuard, // 持有即计数，随 body 一起 drop
}

/// 包装响应体，让交换在「客户端还在被写入」的整个期间保持被计数。
pub fn track_body(body: Body, guard: InflightGuard) -> Body {
    Body::new(TrackedBody { inner: body, _guard: guard })
}

impl HttpBody for TrackedBody {
    type Data = Bytes;
    type Error = axum::Error;

    fn poll_frame(
        self: Pin<&mut Self>,
        cx: &mut Context<'_,>,
    ) -> Poll<Option<Result<Frame<Self::Data>, Self::Error>>> {
        // Body 与 InflightGuard 都是 Unpin，直接字段借用转发即可，
        // 不需要 pin-project，也没有任何数据拷贝。
        Pin::new(&mut self.get_mut().inner).poll_frame(cx)
    }

    // 以下两个方法纯透传，保证包装层对下游不可见。
    fn is_end_stream(&self) -> bool {
        self.inner.is_end_stream()
    }

    fn size_hint(&self) -> SizeHint {
        self.inner.size_hint()
    }
}

```

experimental/sgl-router/src/server/app.rs

唯一接入点：count_requests 中间件在 record_ingress 后 enter() 计数，并把守卫通过 resp.map(|body| track_body(body, inflight)) 搬进响应体，替换原来的直接 resp 返回。这一行是本 PR 全部语义的承载处，改动虽小但决定了计数是否覆盖流式响应。

```

// experimental/sgl-router/src/server/app.rs —— build_router 上的边缘中间件
// 它同时承担两类职责：原有的 requests_total / responses_total 入口 - 出口计数，
// 以及新增的 inflight_http 在途交换计数（唯一能看到「每条路由、每次交换」的位置）。

```

```

async fn count_requests(State(ctx): State<Arc<AppContext>>, req: Request, next: Next) ->
Response {
    let method = req.method().as_str().to_owned();
    // route 取匹配到的模板而非原始 URI，避免标签基数爆炸。
    let route = req
        .extensions()
        .get:::<MatchedPath>()
        .map(|m| m.as_str().to_owned())
        .unwrap_or_else(|| "unmatched".to_owned());
}

```

```

// 原逻辑：入口侧指标 requests_total{route, method}
ctx.metrics.record_ingress(&route, &method);

// 新增：在任何 handler 之前入场计数，守卫先留在本函数局部变量里。
let inflight = ctx.inflight_http.enter();

let resp = next.run(req).await;

// 原逻辑：出口侧指标 responses_total{route, method, status_code}
ctx.metrics
    .record_response(&route, &method, resp.status().as_u16());

// 决定性的一行：把守卫从中间件作用域搬到响应体上。
// 本函数在此处返回时，流式补全的正文往往才刚开始写 —— 若在此 drop 守卫，
// 计数会在整个 drain 等待窗口里错误地读出 0。
resp.map(|body| crate::server::inflight::track_body(body, inflight))
}

```

experimental/sgl-router/src/server/app_context.rs

把 `inflight_http: Arc<InflightHttp>` 注入共享上下文，注释明确它与 `active_load` 的职责差异（后者只覆盖被代理子集）；在 `with_active_load` 与 `#[cfg(test)] stub()` 两处构造点都完成初始化，保证测试框架下计数可用。

```

// experimental/sgl-router/src/server/app_context.rs
// 新增字段：所有路由共享的进程内在途交换计数。
// 注释点明与 active_load 的区别 —— 后者只看得见被代理的那部分请求，
// 而这里覆盖非代理路由与流式正文仍在写入的窗口。
pub struct AppContext {
    // ... (tokenizers / proxy / registry / policies / metrics / engine_load 等原有字段)
    pub kv_metrics: Option<KvIndexMetrics>,
    /// Open HTTP exchanges, on every route. What axum's graceful shutdown
    /// waits on — `active_load` sees only the proxied subset.
    pub inflight_http: Arc<InflightHttp>,
    ready: AtomicBool,
}

// 生产路径：with_active_load 内构造，跟随 AppContext 生命周期。
// let metrics = MetricsRegistry::new();
// active_load.attach_metrics(Arc::clone(&metrics));
// self { /* ... */ inflight_http: InflightHttp::new(), ready: AtomicBool::new(false) }

// 测试路径：stub() 同样要初始化，否则基于 AppContext 的中间件测试无法计数。
// （本次同时把 ModelConfig 的 sampling_overrides 补进 stub，与 pd_protocol_binding
// 的 fixture 修复同源，都是 #39000 引入字段后的编译适配。）

```

五、评论区精华

本 PR 的讨论非常稀薄：只有 1 条 review (sherlockwu, **APPROVED**)，正文为空，没有任何 inline 评论。因此「讨论」实际以 PR body 与源码文档注释的形式完成，以下几条最具价值

:

"`ActiveLoadRegistry::inflight_count` is the closest thing and answers a different question — it counts proxied requests, so it sees neither a request on a non-proxied route nor, more importantly, a streaming completion whose handler has already returned." —— 与既有指标划清职责边界，是整段设计的立论。

"a client that disconnects mid-stream must not ratchet the gauge up for the life of the process" —— 解释了为什么必须用 RAII 守卫、为什么释放点选在 body 而不是 handler。

"Verified non-vacuous: releasing the guard at handler exit instead of on body completion fails `inflight_http_counts_a_streaming_response_until_its_body_finishes` with `left: 0, right: 1`." —— 作者主动提供反向验证，弥补了 review 讨论不足带来的可信度缺口。

"The fixture (#39006) and the `sampling_overrides` field (#39000) landed in the same window, each green against the `main` it was tested on, so no merge reported a conflict." —— 交代了 `main` 断编译的协作成因，也解释了为什么这行修复会被夹带进来。

由于缺少独立评审意见，本 PR 的语义正确性实际上由「作者自述 + 自证测试」承担，这一点在后续 #39016 落地时应被再次核对。

六、风险与影响

技术风险

1. 暂无消费方，语义未被真实使用锁定：`inflight_http` 目前只是一个内部 `AppContext` 字段，既未在 `/metrics` 暴露，也无任何心跳读取。「什么算在途」的最终定义权在 #39016，本 PR 的测试只能证明当前假设自洽，不能证明对 `drain` 足够。
2. layer 顺序与 body 缓冲：守卫的释放时机取决于 `TrackedBody` 在 `tower/axum` 层栈中的位置。`Cargo.toml` 同时启用了 `tower-http` 的 `compression-gzip`、`timeout`、`request-id` 等层，若某一层将响应体整体缓冲而非流式转发，计数窗口会被拉长到该层生命周期，从而影响未来 `drain` 的时长判断；现有集成测试覆盖了真实 `build_router` 栈，但没有覆盖 `gzip / timeout` 组合。
3. 全局热路径新增一层 body 包装：每个响应的每一帧都经过 `TrackedBody::poll_frame`。实现是零拷贝的字段转发、`Unpin` 下无需 `pin` 投影，开销极小，但属全局路径，后续改动需保持这一轻量实现。
4. 计数口径包含非代理流量：`/metrics`、`/readyz`、404 都会被计入。这是刻意设计（它们同样是 `axum drain` 等待的交换），但意味着一个被长期持有的 `/metrics` scrape 或慢客户端会延长未来的下线时间，这是有意权衡，需要在 #39016 的文档中写清。
5. 夹带的修复带来归因耦合：`pd_protocol_binding.rs` 的一行修复与仪表无关，混在同一 `commit` 中会让 `bisect / blame` 看到不相关改动；风险很低（纯测试 fixture），但提醒了多 PR 并行落地时的协作风险。

影响范围

维度	影响
终端用户	无可见变化，wire 与响应内容均未改
运维	首次拥有覆盖全部路由的真实在途口径，是 shutdown drain 心跳的唯一可信数据源
可观测性	尚未导出为 Prometheus 指标， <code>/metrics</code> 上看不到这条 gauge
团队	解锁 main 上 router 分支的 <code>cargo test / cargo clippy --all-targets</code> ，并为 #39015 / #39016 解除前置依赖

七、关联脉络

本 PR 属于 router 优雅关闭 drain 三段式的第 1 块，PR body 给出的 stack 是：

- 1/3 #39014（本 PR）—— 在途计数仪表；
- 2/3 #39015 `[Router] Add the shutdown-drain configuration surface`—— drain 的可调参数、启动校验与告警，改的是 `src/config/` 与 `tests/proxy/graceful_shutdown.rs`；
- 3/3 #39016—— drain 本体与 k8s 证明（原 #39017 只测 drain 行为，已并入并关闭）。

横向看，本 PR 与 #39000 / #39001 采样契约系列在同窗口落地：#39000 引入 `ModelConfig.sampling_overrides`，与 #39006 的 fixture 形成一次「各自绿、合并红」的断编译，本 PR 的一行修复正是为这一段历史收尾。此外 #39004（按 worker 解析 wire protocol）同样改动了 `tests/proxy/pd_protocol_binding.rs` 中注册 `WireProtocol::H2c` 混合 worker 的 fixture，说明该文件近期是多人交叉修改的热区。

放到更长的演进线上看，router 近期一系列 PR 都在补齐「可解释的运行状态」：#39004 明确每个 worker 的转发协议、#39001 按请求强制采样契约并给出拒绝码与指标、#39111 / #39110 把 KV 分层缓存路由做成可观测面板与 e2e 覆盖。本 PR 补上的则是 停机过程中的状态可见性——从「能不能优雅停下」迈向「停机时能否说清在等什么」。