PR #32342 完整报告

sgl-project/sglang

sglang rust server egress message

合并时间：2026-07-30 08:16

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32342

执行摘要

本 PR 新增了 Rust 服务器的 egress（响应）消息层，包括完成原因类型体系、采样参数验证增强和正则 admission 缓存。它是 Rust 服务器消息层拆分工作的一部分，为后续替换 Python TokenizerManager 的响应处理路径奠定基础。设计注重向前兼容性和安全性，但目前缺少单元测试，风险集中在重叠游标假设和锁竞争上。

功能与动机

Rust 服务器正在逐步替代 Python TokenizerManager 的功能，以减少 Python 调度器的负载。PR body 指出本 PR 是从 #29799 拆分而来，并依赖 #32240。核心动机是让 Rust 端独立处理 egress 方向的序列化和协议封装，包括流式响应帧、完成原因分类和采样参数验证。这样做的目的是将 per-request 的 CPU 密集型工作从调度器移出，提升整体吞吐。

实现拆解

1. 出方向消息类型（egress.rs）：定义 EgressSink（基于 mpsc 的 per-request 回传通道）、EgressItem（Four events: Frame, Done, Control, Error）、帧标签常量和列式批解码辅助函数。这些类型构成了响应管线的基础，后续的 detok shard 将 egress 帧写入 EgressSink，API 处理器通过 EgressSource 接收。
2. 完成原因类型（finish_reason.rs）：实现 FinishReason 枚举，与 Python FinishReasonDict 精确对应。通过 serde(tag = "type") 实现基于 type 字段的标签联合；Unknown 分支保留原始 JSON Map，保证 Python 侧新增完成原因不会导致 Rust 解析整体失败。提供 matched() 和 abort_status() 方法供其他组件使用。
3. 采样参数管线（sampling.rs）：补全 SamplingParams 的 normalize() 和 verify() 方法，完整复制 Python 的 __post_init__ → normalize → verify 管线。引入 defaulted! 宏简化字段默认值管理，并增加 stop_strs、stop_regex 的长度和数量硬限制（MAX_STOP_COUNT: 32, MAX_STOP_REGEX_LEN: 256, MAX_STOP_REGEX_COUNT: 32）。
4. 正则 admission 缓存（regex.rs）：添加基于 HashMap 的全局缓存，避免重复解析已验证的正则表达式。缓存容量 512（与 CPython re._MAXCACHE 一致），满时全量清空。RegexPattern::build 首先查询缓存，命中则直接返回已缓存的 bound。
5. 模块注册（message.rs）：添加 mod egress; mod finish_reason; 使新模块参与编译。

rust/sglang-server/src/message/egress.rs

新增 egress 模块核心文件，定义 EgressSink、EgressItem、帧标签和列式批解码辅助函数，是响应管线的基石。

```
//! Egress 方向的核心数据类型 —— 响应回传通道与帧编码。

use bytes::Bytes;
use tokio::sync::mpsc;
use crate::error::Error;

/// 每个请求的回传通道，API 处理器从中 drain 响应。
/// 当前仅支持 Local(mpsc)，未来可扩展为远程传输。
#[derive(Clone, Debug)]
pub enum EgressSink {
    Local(mpsc::Sender<EgressItem>),
}

/// try_send 失败的原因。
#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub enum SinkError {
    Full, // 客户端反压
    Closed, // 客户端已断开
}

impl EgressSink {
    /// 非阻塞发送。Full 和 Closed 均为流终止状态，调用方据此区分日志等级。
    pub fn try_send(&self, item: EgressItem) -> Result<(), SinkError> {
        match self {
            EgressSink::Local(tx) => tx.try_send(item).map_err(|e| match e {
                mpsc::error::TrySendError::Full(_) => SinkError::Full,
                mpsc::error::TrySendError::Closed(_) => SinkError::Closed,
            }),
        }
    }
}

/// API 处理器在 egress 流上收到的四种事件。
#[derive(Debug)]
pub enum EgressItem {
    Frame(ChunkEvent), // 流式中间帧
    Done(ChunkEvent), // 最终完成帧
    Control(Bytes), // 控制请求的透传结果
    Error(Error), // 终端错误
}

// 帧标签常量 —— 用于 egress ring 协议。
pub const EGRESS_TAG_RESULT: u8 = 1; // 单一控制结果
pub const EGRESS_TAG_BATCH: u8 = 2; // 解码 batch
pub const EGRESS_TAG_ERROR: u8 = 3; // 单请求错误
```

rust/sglang-server/src/message/finish_reason.rs

新增完成原因模块，精确对应 Python FinishReasonDict，提供向前兼容的 Unknown 分支和 abort_status 辅助方法。

```
//! FinishReason 枚举 —— 精确映射 Python `FinishReasonDict`。
//! 未知类型被保留为 raw map，向前兼容。

use serde::{Serialize, Deserialize};

/// stop 匹配的值：单个 token id、字符串或多 token 序列。
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize)]
#[serde(untagged)]
pub enum Matched {
    Token(i64),
    Str(String),
    Tokens(Vec<i64>),
}

/// 已知的完成类型，通过 JSON 字段 `type` 区分。
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize)]
#[serde(tag = "type", rename_all = "lowercase")]
pub enum FinishKind {
    Stop {
        #[serde(default)]
        matched: Option<Matched>,
    },
    Length {
        #[serde(default)]
        length: Option<u64>,
    },
    // Abort 被 Box 包装以减少 ChunkEvent 的大小（它很少出现）。
    Abort(Box<AbortReason>),
}

/// Abort 附带的详细信息。
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize)]
pub struct AbortReason {
    pub message: Option<String>,
    pub status_code: Option<u16>,
    pub err_type: Option<String>,
}

/// 完成原因对外表示。
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize)]
#[serde(untagged)]
pub enum FinishReason {
    Known(FinishKind),
    // 未知类型保留原始 JSON Map，保证 batch 解析不会因新增完成类型而整体失败。
```

```
    Unknown(serde_json::Map<String, serde_json::Value>),
}

impl FinishReason {
    /// 如果原因是 stop，返回匹配到的 stop 值。
    pub fn matched(&self) -> Option<&Matched> {
        match self {
            FinishReason::Known(FinishKind::Stop { matched }) => matched.as_ref(),
            _ => None,
        }
    }

    /// 如果原因是有状态码的 abort，返回 (code, message)。
    pub fn abort_status(&self) -> Option<(u16, &str)> {
        match self {
            FinishReason::Known(FinishKind::Abort(a)) => {
                Some((a.status_code?, a.message.as_deref().unwrap_or("request aborted")))
            }
            _ => None,
        }
    }
}
```

评论区精华

> reviewer mrain: take_f32/take_i32 中 [cv..cv + 4*l] 与 [ci..ci + 4*l] 可能重叠，导致未定义行为。 作者 rainj-me: 你说得对，但是这是 hot path，调用方必须保证游标不重叠，否则视为 malformed 帧拒绝。
>
> reviewer mrain: SinkError 没有实现 Display，当前是否用于日志？ 作者 rainj-me: 目前不记录这个错误，以后需要时再添加。
>
> reviewer mrain: 既然最终所有 ChunkEvent 都被实现，直接解析 egress 消息为 Vec<ChunkEvent> 可能更清晰。 （作者未直接回应，建议暂未采纳。）

风险与影响

- 缺少测试覆盖：整个 PR 未包含 Rust 单元测试，egress 解析的正确性仅靠 Python 侧的集成测试保障。批解码边界条件、填充字节对齐等关键路径存在盲区。
- 重叠游标假设：take_f32/take_i32 依赖调用方保证游标区间不重叠，若调用方出错将导致静默解析失败（返回 None），整个 batch 被丢弃。
- 全局锁竞争：正则 admission 缓存使用 Mutex<HashMap>，在多线程 ingress 场景下可能引入锁竞争；全量清空策略可能降低命中率。
- 行为变更：SamplingParams 明确定义拒绝 n > 1，与 Python 端行为一致，但若已有请求使用 n > 1 将收到 400 错误。

影响范围：核心消息路径，但当前仅Rust服务器启用时生效。对用户透明，所有接口保持兼容。后续开发必须遵循本 PR 定义的数据类型和协议约定。

关联脉络

本 PR 是 Rust 服务器消息层拆分工作的一部分，与 #32240（依赖）、#32242（request 消息）、#32343（sampling 消息）属于同一系列。这些 PR 共同构建了完整的 Rust 消息处理管线，最终目标是用 Rust 完全替换 Python TokenizerManager。开发者可沿着 #32240 → #32242 → #32342 → #32343 的顺序阅读，以理解完整的设计演进。