PR #33894 完整报告

sgl-project/sglang

refactor error responses into shared utils::response helpers

合并时间：2026-08-13 16:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33894

执行摘要

PR #33894 将 Rust API server 中原生（/generate）与 OpenAI（/v1/chat/completions、/v1/completions、/v1/models）的错误响应逻辑统一到新建的 utils::response 共享模块：error_value 构造原生 JSON body，error_response 统一 unary / streaming 塑造规则，sse_error_response 生成流内错误帧。各模块私有的 pre_submit_error、openai_error_response、streaming_error 被删除，替换为 native_error / openai_error 两个薄包装。这是一次零行为变更的重构，错误响应形状继续与 Python 侧对齐，并由迁移后的单元测试固定，合入者 sherlockwu 的评论直接决定了薄包装的最终形态。

功能与动机

PR body 是空模板，动机来自提交信息：'Unify the native and OpenAI error paths on one error_response / error_value / sse_error_response set in utils::response'。此前错误响应的塑造逻辑分散在三处：frame.rs 的 error_value、submit.rs 的 pre_submit_error + sse_error_response、openai.rs 的 openai_error_response + streaming_error，各自重复实现同一套「unary → 状态码 + JSON body；streaming → 200 + SSE 错误帧 + [DONE]」的规则，新增 endpoint 时极易走样。重构的核心判断是：wire shape（body 的 JSON 形状）是协议自有的，归各 endpoint 拥有；而 shaping（怎么把错误变成 HTTP 响应）是通用的，统一下沉到共享模块。PD bootstrap registry 的纯文本 body 是协议自有，刻意不参与统一。

实现拆解

1. 新建共享模块 utils::response：response.rs（+94 行）定义 error_value、error_response、sse_error_response 三个原语，并在 utils.rs 注册 pub mod response。模块注释明确划分职责：原生 API 用 error_value 构造 body，OpenAI 用自己的 error_payload，两者都经由 error_response 出网。附带的 error_responses_match_python_shape 测试把 Python parity 契约（unary 形态与 streaming 形态）固化为可执行断言。
2. 原生路径迁移：frame.rs 删除私有 error_value；submit.rs 删除 pre_submit_error、sse_error_response 及测试（净删 80 行），TM inbox 关闭时的 503 改调 native_api::native_error；native_api.rs 新增薄包装 native_error(code, message, stream)，generate 入口的 JSON 解析失败、batch 校验失败、prefetch 失败三条错误路径全部切换。
3. OpenAI 路径迁移：openai.rs 的 error_payload 公开化且 message 泛化为 impl Into<String>，删除 openai_error_response 与 streaming_error，openai_error 扩展

stream 参数后委托 error_response。completions.rs / chat.rs / models.rs 数十处 pre-submit 调用点显式补 false；流内错误帧（truncated、EgressItem::Error、abort_status）从 streaming_error 改为直接构造 error_payload(...).to_string(), StatusCode::from_u16(...).unwrap_or(500) 回退逻辑保持原样。

4. 测试配套：test_utils.rs 的 openai_error_response_covers_unary_and_sse 改用新 openai_error 签名，验证 unary 的 JSON 形状（type / param / code 字段）与 streaming 的 200 + SSE 帧 + [DONE] 形态；原 submit.rs 中的 parity 测试等价迁移到 response.rs。

rust/sglang-server/src/utils/response.rs

本 PR 的核心新增文件，统一承载原生与 OpenAI 共用的错误响应塑造逻辑（unary → JSON，streaming → 200 + SSE 错误帧 + [DONE]），并内置 Python parity 测试。

```
//! utils/response.rs —— 共享的 HTTP 错误响应塑造逻辑。
//!
//! 两类东西刻意分开：WIRE SHAPES（JSON body 形状）仍归各 endpoint 所有，
//! 这里只负责「怎么把错误变成 HTTP 响应」。原生 API 用 `error_value`
//! 构造 `{"error": {message, code}}`，OpenAI 前端用自己的 `error_payload`，
//! 两者都经由 `error_response` 出网；PD bootstrap registry 的纯文本 body
//! 是协议自有的，刻意不在此统一。

use std::convert::Infallible;

use axum::{
    Json,
    http::StatusCode,
    response::{IntoResponse, Response, sse::{Event, Sse}},
};

/// 原生错误 body：统一的 `{"error": {...}}` 对象，不是裸文本 ——
/// 裸文本会让按 JSON 解析的客户端直接报错。
pub fn error_value(code: u16, message: &str) -> serde_json::Value {
    serde_json::json!({ "error": { "message": message, "code": code } })
}

/// 按客户端已承诺的形态出错误：
/// unary → 状态码 + JSON body；streaming → 200 + 一个 SSE 错误帧 + `[DONE]`，
/// 因为客户端已经在读流（Python 端也在 `stream_results()` 里流内应答）。
/// body 由调用者决定形状：原生 `error_value` 或 OpenAI `error_payload`。
pub fn error_response(code: StatusCode, body: serde_json::Value, stream: bool) -> Response {
    if !stream {
        return (code, Json(body)).into_response();
    }
    sse_error_response(body)
}

/// 200 SSE 响应：一个错误帧 + `[DONE]`。
/// 原生 API 与 OpenAI 前端共用；状态码折进 body（响应状态已是 200）。
```

```
pub fn sse_error_response(body: serde_json::Value) -> Response {
    let frames = [body.to_string(), "[DONE]".to_string()];
    Sse::new(futures::stream::iter(
        frames.map(|data| Ok::<_, Infallible>(Event::default().data(data))),
    ))
    .into_response()
}

#[cfg(test)]
mod tests {
    use super::*;

    /// Python-parity 钉子（对齐 `http_server.generate_request`）：
    /// unary 错误是 4xx/5xx + JSON `{"error": ...}`；
    /// streaming 错误是 200 + 一个 SSE 错误帧 + `[DONE]`。
    #[tokio::test]
    async fn error_responses_match_python_shape() {
        // unary 分支：状态码透传，body 为 JSON 对象
        let unary = error_response(StatusCode::BAD_REQUEST, error_value(400, "bad input"),
        false);
        assert_eq!(unary.status(), StatusCode::BAD_REQUEST);
        let body = axum::body::to_bytes(unary.into_body(), 64 * 1024).await.unwrap();
        let v: serde_json::Value = serde_json::from_slice(&body).expect("JSON body");
        assert_eq!(v["error"]["message"], "bad input");
        assert_eq!(v["error"]["code"], 400);

        // streaming 分支：HTTP 本身是 200，状态码以 "code" 字段内嵌
        let streamed = error_response(StatusCode::BAD_REQUEST, error_value(400, "bad input"),
        true);
        assert_eq!(streamed.status(), StatusCode::OK, "the stream itself is 200");
        let body = axum::body::to_bytes(streamed.into_body(), 64 * 1024).await.unwrap();
        let text = String::from_utf8(body.to_vec()).unwrap();
        assert!(text.contains(r#""code":400"#), "carries the status in-band: {text}");
        assert!(text.trim_end().ends_with("data: [DONE]"), "terminated: {text}");
    }
}
```

rust/sglang-server/src/api_server/openai.rs

OpenAI 错误路径的收口点：error_payload 公开化并泛化参数，openai_error 扩展 stream 参数后委托共享 error_response，删除 openai_error_response 与 streaming_error。

```
// api_server/openai.rs —— OpenAI 前端的错误 body 与薄包装。
// `error_payload` 是协议自有的 body 形状；`openai_error` 只负责把
// 状态码、body、stream 标志一次性交给共享的 `utils::response::error_response`。

/// OpenAI 错误 payload：`type` 是 SDK 面对的错误类别
///（`AuthenticationError` / `InternalServerError` / `BadRequestError`），
/// `code` 携带 HTTP 状态 —— 对齐 Python OpenAI 前端的输出形状。
pub(super) fn error_payload(code: StatusCode, message: impl Into<String>) -> serde_json::
```

```
Value {
    let message = message.into();
    let error_type = if code == StatusCode::UNAUTHORIZED {
        "AuthenticationError"
    } else if code.is_server_error() {
        "InternalServerError"
    } else {
        "BadRequestError"
    };
    serde_json::json!({
        "error": {
            "object": "error",
            "message": message,
            "type": error_type,
            "param": null,
            "code": code.as_u16(),
        }
    })
}

/// OpenAI 错误响应薄包装：调用点只需写一次状态码，
/// body 拼装与 unary / streaming 分支全部下沉到共享的 `error_response`。
pub(super) fn openai_error(code: StatusCode, message: impl Into<String>, stream: bool) ->
Response {
    error_response(code, error_payload(code, message), stream)
}

// api_server/native_api.rs 中对应的原生侧薄包装，语义完全对称：
// body 用原生 `error_value`，其余逻辑同样委托 `error_response`。
// pub(super) fn native_error(code: StatusCode, message: &str, stream: bool) -> Response {
// error_response(code, error_value(code.as_u16(), message), stream)
// }
```

评论区精华

> Can we have a thin wrapper (one native, one OpenAI) so the status code don't need to be typed twice per call site?

sherlockwu 在 chat.rs 的 diff 上指出：初始版本里每个调用点要把同一个状态码传给 error_response 和 error_payload 两次，代码冗余且易错；建议原生与 OpenAI 各做一个薄包装，把 code + message + stream 一次性收口。该建议被完整落实——native_error 与 openai_error 成为本 PR 对外的主要 API 形态，第二个提交 'address comment' 即针对此评论。这是本 PR 唯一一条 review 评论，却直接决定了最终代码结构。

风险与影响

- 调用点密集变更：openai_error 从「恒 unary」变为显式 stream 参数，completions.rs / chat.rs / models.rs 约数十处调用点手工补 false，遗漏或误传会改变错误响应形态（unary 4xx ↔ 200 + SSE）。当前调用点均为 pre-submit 校验场景，与旧语义一致，测试

覆盖了双分支但并非逐点覆盖。

- 流式错误构造内联：streaming_error 删除后，其 StatusCode::from_u16 + unwrap_or(500) 逻辑内联进 completions.rs 与 chat.rs 的两个 streaming 事件循环，后续调整流内错误帧格式需同步改两处。
- 模块循环引用：submit.rs 依赖 native_api::native_error，native_api 又依赖 submit::submit；Rust 允许模块级循环，但增加了耦合，后续拆分 api_server 时需注意。
- 兼容性：对客户端无 wire 变化，error_payload 泛化为 impl Into<String> 属纯内部签名变化；error_value 从 frame.rs 迁到 utils 后为 pub，可见性扩大但语义未变。

影响范围覆盖所有原生与 OpenAI 端点的错误路径，但由于是零行为变更，对用户和下游客户端透明；对团队的主要收益是错误契约集中化，后续演进（如统一 error object 字段、调整流内错误帧格式）只需改 response.rs 加各 endpoint 的 body 构造器。

关联脉络

历史 PR 中未发现直接修改 rust/sglang-server 错误路径或同功能线的条目，本 PR 属于 Rust 前端「单一错误契约」建设的一部分。它与 Python 端 http_server.generate_request / OpenAI 前端的 parity 契约是跨语言约束：测试名 error_responses_match_python_shape 显式声明了这一点，未来 Python 侧若调整流内错误应答（stream_results()），Rust 侧相应改动应落在 response.rs 及其测试上。