PR #44382 完整报告

vllm-project/vllm

[Rust Frontend] Add /abort_requests endpoint

合并时间：2026-06-17 14:40

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44382

执行摘要

本 PR 为 Rust 前端新增 /abort_requests 端点，允许通过 POST 请求取消正在进行的任务，是 Rust 前端管理 API 路标的一部分。实现包括路由注册、请求处理函数和完整测试覆盖。

功能与动机

为 Rust 前端添加 /abort_requests 端点，作为 RL / admin / lifecycle APIs 路标的一部分（Issue #44280）。该端点接受包含 request_ids 的 JSON 请求体，返回 200 OK，与 Python 前端的 /abort_requests 接口保持一致。

实现拆解

1. 新增处理模块：创建 abort_requests.rs，定义 AbortRequestsRequest 结构体（request_ids: Option<Vec<String>>）和 abort_requests 异步函数，解析请求体并调用 state.chat.abort(&request_ids)。
2. 路由注册：在 routes.rs 中添加 mod abort_requests 声明，并在 build_router_with_options 的 dev_mode 代码块中注册 POST /abort_requests 路由。
3. 测试覆盖：在 tests.rs 中新增四个测试用例，验证正常请求、缺失字段、畸形 JSON 和空 ID 列表等场景，确保端点的正确性和错误处理符合预期。

rust/src/server/src/routes/abort_requests.rs

核心新增文件，实现 /abort_requests 端点的主要逻辑

```
use std::sync::Arc;
use axum::Json;
use axum::extract::State;
use axum::extract::rejection::JsonRejection;
use axum::http::StatusCode;
use serde::Deserialize;

use crate::error::ApiError;
use crate::state::AppState;
use crate::utils::utility_call_error;

/// 请求体结构：request_ids 为可选字符串列表
#[derive(Debug, Deserialize)]
pub(crate) struct AbortRequestsRequest {
    request_ids: Option<Vec<String>>,
```

```
}

/// POST /abort_requests 处理函数
pub async fn abort_requests(
    State(state): State<Arc<AppState>>,
    body: Result<Json<AbortRequestsRequest>, JsonRejection>,
) -> Result<StatusCode, ApiError> {
    // 尝试解析 JSON，若失败则返回 400
    let Json(body) = body.map_err(|error| ApiError::json_parse_error(error.body_text()))?;
    // 提取 request_ids，若缺失则返回 400 并指出参数名
    let request_ids = body.request_ids.ok_or_else(|| {
        ApiError::invalid_request(
            "Missing 'request_ids' in request body".to_string(),
            Some("request_ids"),
        )
    })?;
    // 调用底层 abort 方法，映射错误
    state
        .chat
        .abort(&request_ids)
        .await
        .map_err(|error| utility_call_error("abort_requests", error))?;
    // 成功返回 200 OK 无 body
    Ok(StatusCode::OK)
}
```

rust/src/server/src/routes.rs

注册 /abort_requests 路由，使其在 dev_mode 下可用

```
// 在 routes.rs 中添加以下内容：

mod abort_requests; // 新增模块声明

// 在 build_router_with_options 函数的 dev_mode 代码块中注册路由：
if dev_mode_enabled {
    router = router
        .route("/reset_prefix_cache", post(cache::reset_prefix_cache))
        .route("/reset_mm_cache", post(cache::reset_mm_cache))
        .route("/reset_encoder_cache", post(cache::reset_encoder_cache))
        .route("/collective_rpc", post(collective_rpc::collective_rpc))
        .route("/abort_requests", post(abort_requests::abort_requests)) // 新增路由
        .route("/sleep", post(sleep::sleep))
        .route("/wake_up", post(sleep::wake_up))
        .route("/is_sleeping", get(sleep::is_sleeping))
        .route("/pause", post(pause::pause))
        .route("/resume", post(pause::resume))
        .route("/is_paused", get(pause::is_paused))
        .route("/server_info", get(server_info::server_info));
}
```

rust/src/server/src/routes/tests.rs

添加四个测试用例，验证端点的正确性和错误处理

```
#[tokio::test(flavor = "multi_thread", worker_threads = 2)]
#[serial]
/// 测试正常请求：发送有效 request_ids 应返回 200 OK 且 body 为空
async fn abort_requests_route_returns_ok_for_well_formed_body() {
    let (app, engine_task) =
        test_admin_app_with_engine_script(l_dealer, _pushl boxed_test_future(async move {})).
        await;

    let response = app
        .clone()
        .call(
            Request::builder()
                .method("POST")
                .uri("/abort_requests")
                .header("content-type", "application/json")
                .body(Body::from(r#"{"request_ids":["req-1","req-2"]}"#))
                .expect("build request"),
        )
        .await
        .expect("call app");

    let status = response.status();
    let body = to_bytes(response.into_body(), usize::MAX).await.expect("read body");
    assert_eq!(status, StatusCode::OK, "{}", String::from_utf8_lossy(&body));
    assert!(body.is_empty());
    engine_task.abort_and_join().await;
}
```

评论区精华

BugenZhao 在评论中指出了关键的设计问题：

> “PR 混淆了 external 和 internal request IDs。Python /abort_requests 接受外部可见的 request ID，并通过 AsyncLLM 的 OutputProcessor.external_req_ids 映射到内部 ID 后发送给 engine core。而在 Rust 中，Llm 随机生成内部 ID 并存储原始值到 external_req_id，但 EngineCoreClient::abort 只查找内部 ID。因此该路由会盲目返回 200 而不实际取消请求。”

经讨论，作者在另一个 PR 中建立了外部到内部的映射，该 PR 得以合并。

风险与影响

- ID 映射风险：若映射未正确实现，/abort_requests 可能返回 200 但实际无操作。该风险已在配套 PR 中修复。
- 可用性：端点仅在 dev_mode 下注册，不影响生产环境。

- 测试覆盖：四个测试用例覆盖了主要场景，但缺失对实际映射逻辑的集成测试（依赖另一 PR）。

关联脉络

本 PR 直接关联 Issue #44280（Rust 前端管理 API 路标），并依赖一个未编号的 PR 实现外部 / 内部 ID 映射。功能完成后，Rust 前端将具备与 Python 前端一致的请求取消能力，为后续管理 API 的扩展奠定了基础。