PR #51089 完整报告

vllm-project/vllm

[Feature] Parse request priority from HTTP header

合并时间：2026-08-06 12:10

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51089

执行摘要

- 一句话：新增 X-Vllm-Priority 头覆盖 body 优先级
- 推荐动作：值得快速浏览，适合作为“在双前端同步新增 header 能力”的参考样例。设计上值得借鉴的是将 header 解析收敛到单一静态方法（_get_priority）并由三条 API 复用，以及 Rust 侧用 ctx.priority.or(request.priority) 表达覆盖优先级。若团队关心健壮性，可在此 PR 基础上补 Python 端单测并考虑对非法 header 值返回 400。

功能与动机

issue #51023 提出：运营侧希望在中间代理 / 负载均衡器上通过修改 HTTP header 调整请求优先级，比解析并改写 JSON 请求体简单一个数量级；优先级信息往往对客户端不可见，且可能因运营或业务决策随时变化。issue 还明确要求 header 优先级必须覆盖 body 中的优先级。本 PR 按此语义实现。

实现拆解

1. Python 入口层新增 header 解析：vllm/entrypoints/generate/base/serving.py 新增 PRIORITY_HEADER = "X-Vllm-Priority" 常量与 _get_priority 静态方法，该方法优先读取 header 并尝试 int() 转换，解析失败则回退到 request.priority；这与已有 X-Session-ID 的 header 处理模式保持一致，避免新增依赖。
2. 三类 API 统一接入：vllm/entrypoints/openai/chat_completion/serving.py、vllm/entrypoints/openai/completion/serving.py、vllm/entrypoints/openai/responses/serving.py 中调用 engine_client.generate(...) 时的 priority=request.priority 均替换为 priority=self._get_priority(request, raw_request)，使 header 覆盖规则一次性地作用于 Chat Completions、Completions 与 Responses 三条路径。
3. Rust 前端解析并合并优先级：rust/src/server/src/utils.rs 中 ResolvedRequestContext 新增 priority: Option<i32> 字段，resolve_request_context 从 X-Vllm-Priority 头读取并 trim().parse()；rust/src/server/src/routes/openai/completions/convert.rs 与 rust/src/server/src/routes/openai/chat_completions/convert.rs 中 priority 计算改为 ctx.priority.or(request.priority).unwrap_or(0)，实现与 Python 端一致的“header 优先、body 次之、缺省 0”的语义。
4. 测试与文档配套：两个 Rust 文件各新增单元测试 prepare_completion_request_header_priority_overrides_body、prepare_chat_request_header_priority_overrides_body，验证

body priority=10 被 header -5 覆盖；docs/serving/online_serving/openai_compatible_server.md 补充 X-VIlm-Priority 的取值约束（整数）、覆盖语义、非零值要求启用优先级调度以及 Python 客户端示例。本 PR 未新增 Python 端测试。

关键文件：

- vllm/entrypoints/generate/base/serving.py（模块 入口服务；类别 source；类型 core-logic；符号 _get_priority）：定义 PRIORITY_HEADER 常量和 _get_priority 静态方法，是 Python 入口优先级覆盖的核心实现，后续所有 API 复用。
- rust/src/server/src/utils.rs（模块 请求解析；类别 source；类型 core-logic；符号 ResolvedRequestContext, resolve_request_context）：为 ResolvedRequestContext 新增 priority 字段并在 resolve_request_context 中解析 X-VIlm-Priority，是 Rust 前端优先级来源的统一入口。
- rust/src/server/src/routes/openai/completions/convert.rs（模块 请求转换；类别 source；类型 entrypoint；符号 prepare_completion_request, prepare_completion_request_header_priority_overrides_body）：构造 CompletionRequest 时改用 ctx.priority.or(request.priority) 实现 header 覆盖 body，并新增对应单元测试。
- rust/src/server/src/routes/openai/chat_completions/convert.rs（模块 请求转换；类别 source；类型 entrypoint；符号 prepare_chat_request, prepare_chat_request_header_priority_overrides_body）：与 completions 同步修改，将 header 优先级合并逻辑应用到 Chat Completions 路径，并增加测试。
- vllm/entrypoints/openai/chat_completion/serving.py（模块 聊天服务；类别 source；类型 core-logic；符号 _create_chat_completion）：Chat Completions 服务接入 _get_priority，header 覆盖 body 优先级。
- vllm/entrypoints/openai/completion/serving.py（模块 完成服务；类别 source；类型 core-logic；符号 _create_completion）：Completions 服务接入 _get_priority，保持与 Chat Completions 一致。
- vllm/entrypoints/openai/responses/serving.py（模块 响应服务；类别 source；类型 core-logic；符号 _create_responses）：Responses API 同步接入 header 优先级覆盖逻辑，保证三类 API 行为一致。
- docs/serving/online_serving/openai_compatible_server.md（模块 在线文档；类别 docs；类型 documentation）：补充 X-VIlm-Priority 头的使用说明、覆盖语义与示例，是用户和运维理解该能力的主要文档入口。

关键符号：_get_priority, resolve_request_context, prepare_completion_request, prepare_chat_request

关键源码片段

vllm/entrypoints/generate/base/serving.py

定义 PRIORITY_HEADER 常量和 _get_priority 静态方法，是 Python 入口优先级覆盖的核心实现，后续所有 API 复用。

```
# vllm/entrypoints/generate/base/serving.py
```

```
SESSION_ID_HEADER = "X-Session-ID"
# 新增常量，统一 header 键名，避免散落在各 serving 模块中
PRIORITY_HEADER = "X-VIlm-Priority"


class GenerateBaseServing(...):
    # 省略其他内容 ...

    @staticmethod
    def _get_priority(
        request: ChatCompletionRequest | CompletionRequest | ResponsesRequest,
        raw_request: Request | None,
    ) -> int:
        """优先从 HTTP header 解析优先级，解析失败时回退到请求体中的 priority。

        这样中间代理或负载均衡器只需设置 X-VIlm-Priority 头即可覆盖
        客户端 JSON 里携带的优先级，无需解析整个请求体。
        """
        if raw_request is not None:
            priority = raw_request.headers.get(PRIORITY_HEADER)
            if priority is not None:
                try:
                    # header 值必须是整数，否则抛 ValueError
                    return int(priority)
                except ValueError:
                    # header 值不是合法整数时静默回退到 body 字段
                    pass
        # 最终回退到 body 里的 priority（可能为 None，由调用方决定默认值）
        return request.priority
```

rust/src/server/src/utils.rs

为 ResolvedRequestContext 新增 priority 字段并在 resolve_request_context 中解析 X-VIlm-Priority，是 Rust 前端优先级来源的统一入口。

```
// rust/src/server/src/utils.rs

/// 从 HTTP header 解析出的公共请求上下文，供各路由复用。
#[derive(Debug, Clone, PartialEq, Eq, Default)]
pub struct ResolvedRequestContext {
    pub request_id: String,
    pub data_parallel_rank: Option<u32>,
    pub priority: Option<i32>, // 新增字段：X-VIlm-Priority 解析结果
    pub session_id: Option<String>,
}

/// 提取公共请求元数据：外部 request ID、session ID、优先级，
/// 以及用于引擎路由的 data-parallel-rank。
pub fn resolve_request_context(
    headers: &HeaderMap,
```

```
    request_id: Option<&str>,
) -> ResolvedRequestContext {
    // 仅当 header 缺失或无法解析为 u32 时为 None
    let data_parallel_rank = headers
        .get("X-data-parallel-rank")
        .and_then(|v| v.to_str().ok())
        .and_then(|s| s.trim().parse().ok());

    // 新增：X-VIlm-Priority 解析为 i32，trim 后 parse，失败则为 None
    // 语义与 Python 端"解析失败回退"一致
    let priority = headers
        .get("X-VIlm-Priority")
        .and_then(|value| value.to_str().ok())
        .and_then(|value| value.trim().parse().ok());

    // 请求 ID、session ID 的解析逻辑保持不变 ...
    ResolvedRequestContext {
        request_id,
        data_parallel_rank,
        priority,
        session_id,
    }
}
```

rust/src/server/src/routes/openai/completions/convert.rs

构造 CompletionRequest 时改用 ctx.priority.or(request.priority) 实现 header 覆盖 body，并新增对应单元测试。

```
// rust/src/server/src/routes/openai/completions/convert.rs
// 构造内部请求时，header 优先级（ctx.priority）优先于 body 字段（request.priority），
// 两者都缺失时默认 0。chat_completions/convert.rs 中的 prepare_chat_request 同样处理。
priority: ctx.priority.or(request.priority).unwrap_or(0),

// 对应单元测试：body priority=10 被 header -5 覆盖
#[test]
fn prepare_completion_request_header_priority_overrides_body() {
    let request: CompletionRequest = serde_json::from_value(json!({
        "model": "Qwen/Qwen1.5-0.5B-Chat",
        "prompt": "hello",
        "priority": 10,
    }))
    .expect("parse request");

    let mut headers = HeaderMap::new();
    headers.insert("X-VIlm-Priority", "-5".parse().unwrap());
    let prepared = prepare_completion_request(
        request,
        &served(&["Qwen/Qwen1.5-0.5B-Chat"]),
        request_context(&headers, None),
```

```
        &test_tokenizer(),
    )
    .expect("prepare");
    // 验证 header 覆盖 body 中的 priority
    assert_eq!(prepared.text_request.priority, -5);
}
```

评论区精华

评审过程没有出现实质技术争论：DarkLight1337 与 BugenZhao 均批准，BugenZhao 评论 LGTM；Claude bot 因 PR 来自 fork 而自动评审被禁用，提示维护者可评论 @claude review 手动触发。值得注意的是，PR body 中 Test Plan 部分为空，最终由维护者直接评审放行。

- fork 自动化审核被禁用 (other): 无后续动作，改由维护者人工评审。
- 维护者批准 (other): PR 被合并。

风险与影响

- 风险：
  1. 双栈语义一致性风险：Python 端 _get_priority 在 header 值无法转 int 时回退 body，Rust 端 resolve_request_context 解析失败得到 None 后同样回退 body，两者语义已对齐；但若后续单边修改（例如对非法值报错），三条 API 与两条前端的行为可能分叉。
  2. 非法值静默回退：header 值拼写错误或非整数时会被静默忽略，运维配置错误难以第一时间暴露；可考虑未来对非法值返回 400。
  3. 优先级调度前提：文档明确非零优先级需要服务器启用优先级调度；若未启用，header 设置的非零值可能被忽略或产生与预期不符的排队行为，用户需自行保证配置。
  4. Python 端缺少测试：_get_priority 及三个 serving 文件的替换没有 Python 测试覆盖，回归只能依赖 Rust 端测试和人工验证。
  5. Responses API 联动：Responses 路径同样被修改，但本次没有针对 Responses 的 Rust/Python 定向测试。
- 影响：
  1. 用户侧：代理 / 负载均衡器无需解析和改写整个 JSON body 即可动态调整单请求优先级，对含大附件、图片的请求尤其降低代理开销；header 覆盖语义让客户端无法绕过运营设定的优先级。
  2. 系统侧：优先级最终仍通过既有 engine_client.generate(priority=...) 与调度器交互，调度行为不变；入口层多了一次 header 读取与 int 转换，开销可忽略。
  3. 团队侧：新增一种前端 header 约定，需要同时维护 Python/Rust 两条实现路径的一致性；文档已同步，降低使用门槛。 - 风险标记：Python 端缺少测试覆盖，非法 header 值静默回退，双栈语义需保持一致，非零优先级依赖优先级调度配置

关联脉络

- 暂无明显关联 PR