PR #43942 完整报告

vllm-project/vllm

[Rust Frontend] Add /server_info to Rust frontend

合并时间：2026-06-03 19:30

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43942

执行摘要

本 PR 为 Rust HTTP 前端新增 /server_info 端点，返回服务器配置和环境变量快照，与 Python 前端功能对齐。端点仅在 dev_mode 启用时开放，并对环境变量进行敏感内容过滤，防止意外泄漏。

功能与动机

PR 正文指出：'Adds /server_info to the Rust HTTP frontend alongside the existing /version route.' 目的是让 Rust 前端也能提供类似 Python 前端的调试信息端点，便于运维和问题排查。

实现拆解

1. 核心数据模块(server_info.rs)：定义 ServerInfoSnapshot 结构体，包含 vllm_config 的文本与 JSON 表示、vllm_env（过滤后可公开的环境变量）和 system_env。通过 render_config_text 将配置序列化为 key=value 文本格式，方便阅读。
2. 路由处理(routes/server_info.rs)：新增 /server_info 路由，解析可选查询参数 config_format（默认 Text），从 AppState 中拉取快照并返回相应格式。
3. 状态集成(state.rs)：在 AppState 中新增 Option<ServerInfoSnapshot> 字段，提供 with_server_info 构建方法和 server_info_response 查询方法。
4. 序列化简化：为 Config、TransportMode 等结构添加 #[derive(Serialize)]，替代手动 format!，使 JSON 输出更稳定且便于维护。
5. 测试覆盖(tests.rs)：新增两个集成测试，验证非 dev_mode 下返回 404，dev_mode 下未设置快照也返回 404。

rust/src/server/src/server_info.rs

核心模块，定义了 ServerInfoSnapshot 结构和所有收集 / 渲染函数，是新增端点的数据支撑。

```
use std::collections::BTreeMap;
use serde_json::{Value, json};
use crate::config::Config;

// 敏感环境变量 key 的通用模式（大小写不敏感）
const SENSITIVE_VLLM_ENV_PATTERNS: &[&str] =
    &["KEY", "SECRET", "TOKEN", "PASSWORD", "CREDENTIAL", "AUTH"];

/// 输出格式：文本或 JSON
```

```rust
#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub(crate) enum ServerInfoConfigFormat {
    Text,
    Json,
}

/// `/server_info` 返回的快照
#[derive(Debug, Clone)]
pub(crate) struct ServerInfoSnapshot {
    vllm_config_text: String,
    vllm_config_json: Value,
    vllm_env: BTreeMap<String, String>,
    system_env: BTreeMap<String, String>,
}

impl ServerInfoSnapshot {
    /// 从配置构造快照
    pub(crate) fn from_config(config: &Config) -> Self {
        let vllm_config_json =
            serde_json::to_value(config).expect("server info value must serialize");
        Self {
            vllm_config_text: render_config_text(&vllm_config_json),
            vllm_config_json,
            vllm_env: collect_vllm_env(),
            system_env: collect_system_env(),
        }
    }

    /// 按格式返回响应体
    pub(crate) fn response(&self, config_format: ServerInfoConfigFormat) -> Value {
        let vllm_config = match config_format {
            ServerInfoConfigFormat::Text => Value::String(self.vllm_config_text.clone()),
            ServerInfoConfigFormat::Json => self.vllm_config_json.clone(),
        };
        json!({ "vllm_config": vllm_config, "vllm_env": self.vllm_env.clone(), "system_env": self.
        system_env.clone() })
    }
}

/// 将配置值渲染为文本行（key=value）
fn render_config_text(config: &Value) -> String {
    match config {
        Value::Object(fields) => fields
            .iter()
            .map(|(key, value)| format!("{key}={}", render_config_text_value(value)))
            .collect::<Vec<_>>()
            .join("\n"),
        _ => render_config_text_value(config),
    }
```

```
}

/// 将单个值渲染为文本
fn render_config_text_value(value: &Value) -> String {
    match value {
        Value::Null => "None".to_string(),
        Value::String(value) => value.clone(),
        _ => value.to_string(),
    }
}

/// 收集所有公开的 VLLM 环境变量（排除敏感 key）
fn collect_vllm_env() -> BTreeMap<String, String> {
    std::env::vars().filter(|(key, _)| is_public_vllm_env_key(key)).collect()
}

/// 判断是否为公开的环境变量 key：以 `VLLM_` 开头且不包含敏感模式
fn is_public_vllm_env_key(key: &str) -> bool {
    let key = key.to_ascii_uppercase();
    key.starts_with("VLLM_") && !SENSITIVE_VLLM_ENV_PATTERNS.iter().any(|p| key.contains(p))
}

/// 收集静态系统环境信息
fn collect_system_env() -> BTreeMap<String, String> {
    BTreeMap::from([
        ("arch".to_string(), std::env::consts::ARCH.to_string()),
        ("family".to_string(), std::env::consts::FAMILY.to_string()),
        ("os".to_string(), std::env::consts::OS.to_string()),
    ])
}
```

rust/src/server/src/routes/server_info.rs

新增路由处理文件，定义端点入口、查询参数解析和格式转换。

```
use std::sync::Arc;
use axum::Json;
use axum::extract::{Query, State};
use axum::http::StatusCode;
use axum::response::{IntoResponse, Response};
use serde::Deserialize;
use crate::server_info::ServerInfoConfigFormat;
use crate::state::AppState;

/// 查询参数中可用的格式
#[derive(Debug, Clone, Copy, Deserialize)]
#[serde(rename_all = "lowercase")]
enum ConfigFormat {
    Text,
    Json,
```

```
}

/// 将查询格式转换为内部格式
impl From<ConfigFormat> for ServerInfoConfigFormat {
    fn from(value: ConfigFormat) -> Self {
        match value {
            ConfigFormat::Text => Self::Text,
            ConfigFormat::Json => Self::Json,
        }
    }
}

/// 默认格式为 Text
fn default_config_format() -> ConfigFormat {
    ConfigFormat::Text
}

/// 查询参数结构
#[derive(Debug, Deserialize)]
pub(crate) struct ServerInfoParams {
    #[serde(default = "default_config_format")]
    config_format: ConfigFormat,
}

/// 处理 `/server_info` 请求
pub async fn server_info(
    State(state): State<Arc<AppState>>,
    Query(params): Query<ServerInfoParams>,
) -> Response {
    match state.server_info_response(params.config_format.into()) {
        Some(response) => Json(response).into_response(),
        None => StatusCode::NOT_FOUND.into_response(),
    }
}
```

评论区精华

> Copilot: 'Filtering out only variables containing "KEY" is not sufficient… Prefer an explicit allowlist.' depthfirst-app: 'Unlike the Python equivalent which uses a curated allowlist, this reads ALL VLLM_* env vars.' BugenZhao: 'What about also moving the construction-related logic into a separate module?' 以及 'I think it would be better to derive Serialize...'

最终 PR 采纳了 dev_mode 门控和扩大 denylist，但未改用 allowlist；分离模块和 Serialize 派生均已实现。

风险与影响

- 安全风险：尽管有 denylist，仍可能漏掉某些敏感环境变量，但 dev_mode 默认关闭，降低了暴露面。建议未来考虑补全 allowlist 机制。
- 影响范围：仅影响 Rust 前端，新增端点对现有功能无干扰。
- 对用户：开发人员可在启用 dev_mode 后通过 /server_info 快速获取配置，方便调试。

关联脉络

本 PR 是 Rust 前端功能扩展系列的一部分，此前已有 #43774（路由扩展钩子）和 #43778（LoRA 端点）。三者共同提升 Rust 前端的可观测性和管理能力，逐步与 Python 前端功能对标。