

# PR #52840 完整报告

vllm-project/vllm

[Rust Frontend][gRPC] Add LoRA lifecycle control

合并时间: 2026-08-25 12:13

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/52840>

## 执行摘要

PR #52840 为 Rust 前端 gRPC 控制面补齐 LoRA 生命周期能力: 在 Control 服务上新增「加载 / 卸载 / 列表」三个 RPC, 并让 unary 与 streaming 生成请求可按名字选择已加载的 adapter。核心设计是复用现有 LoraManager 注册表, HTTP 与 gRPC 共享同一套身份、容量与加载顺序语义; 同时把 LoRA 参数校验下沉到 LoraRequest::new, 并新增 LoraDisabledError 统一“引擎未启用 LoRA”的错误路径。合入时保留了一个已知安全缺口: gRPC 的 load\_lora 尚未接入 HTTP 侧的路径前缀校验, 已由作者确认以 follow-up PR 跟进。

## 功能与动机

PR body 明确目标: “Add runtime LoRA lifecycle operations to the Rust frontend's existing gRPC APIs”——在此之前动态 LoRA 管理只有 HTTP 路径, gRPC 客户端需要额外 listener 或独立注册表。设计上强调“Reuse the same LoraManager registry for HTTP and gRPC so adapter identity, duplicate-name handling, capacity checks, and load order remain consistent across transports”, 即两个传输面必须共享同一状态, 避免身份分叉。作者在 review 中还透露该接口面向 Dynamo sidecar 集成场景, 但最终放弃了客户端显式指定 LoRA ID 的做法, 改为服务端统一分配。

## 实现拆解

1. proto 契约扩展: rust/proto/control.proto 新增 LoraAdapter 消息和 LoadLora / UnloadLora / ListLoras 三个 RPC (+30 行), rust/proto/inference.proto 扩展生成请求以支持按名字选 adapter (+3 行), rust/Cargo.lock 同步更新。
2. 注册表与校验集中化: rust/src/server/src/lora.rs 为 LoadLoraError / UnloadLoraError 派生 thiserror::Error, 新增 Disabled / InvalidRequest 变体与 LoraDisabledError; LoraManager::load\_lora 在应用层统一完成重名检测、base model 冲突检测与服务端自增 ID 分配; LoraRequest::new 负责非空名称 / 路径及非零 ID 的传输无关校验。
3. 统一能力门检: rust/src/server/src/state.rs 新增 ensure\_lora\_enabled(), 通过 ready\_response().supports\_lora 判断引擎是否启用 LoRA, load/unload/list 三个入口统一先过门检; 新增 list\_loras() 返回按加载顺序排列的 adapter 快照 (对应 BugenZhao 的 “single place” 提交)。
4. gRPC handler: rust/src/server/src/grpc/control.rs 实现三个 RPC 与错误码映射 (Disabled→FailedPrecondition、InvalidRequest→InvalidArgument、已存在 / 冲突→AlreadyExists、引擎错误→Internal); unload 先校验 name 非空, 再基于 list 快照定位 adapter; grpc/inference.rs 让生成请求按名称解析 adapter 并填入

EngineCoreRequest.lora\_request。

5. HTTP 路由重构: rust/src/server/src/routes/lora.rs 把内联 match 错误映射抽成 load\_lora\_api\_error / unload\_lora\_api\_error, 并删除手写的空名 / 空路径检查, 行为由共享校验接管; 路径前缀校验保留在 HTTP 侧, 未与 gRPC 共享。
6. 测试配套: rust/src/server/src/grpc/tests.rs 新增 reply\_utility\_bool 辅助函数及两个集成测试, 覆盖 load→生成选中 adapter→unload 全链路、重复加载冲突、未启用 LoRA 引擎时 list 失败; routes/tests.rs 将 ready 参数化以支持 LoRA 启停场景。

## rust/src/server/src/grpc/control.rs

gRPC LoRA 生命周期 API 的核心实现: 新增 load\_lora/unload\_lora/list\_loras 三个 RPC, 并将统一错误类型映射为 gRPC Status 码, 是整个 PR 的主入口。

// 将内部 LoRA 请求转换为对外公开的 gRPC 协议对象。

// lora\_int\_id 是 u64, proto 字段是 int64, 这里收敛到 i64::MAX 防止溢出。

```
fn lora_to_proto(adapter: &LoraRequest) -> pb::LoraAdapter {
    pb::LoraAdapter {
        lora_id: adapter.lora_int_id.min(i64::MAX as u64) as i64,
        lora_name: adapter.lora_name.clone(),
        source_path: adapter.lora_path.clone(),
    }
}
```

// 把统一的 LoadLoraError 映射为 gRPC Status 码。

// 明确区分“引擎未启用”“请求不合法”“已存在 / 冲突”“引擎内部错误”四类语义。

```
fn load_lora_status(error: LoadLoraError) -> Status {
    let code = match &error {
        LoadLoraError::Disabled(_) => Code::FailedPrecondition,
        LoadLoraError::InvalidRequest(_) => Code::InvalidArgument,
        LoadLoraError::AlreadyLoaded { .. } | LoadLoraError::BaseModelName { .. } => {
            Code::AlreadyExists
        }
        LoadLoraError::Engine { .. } | LoadLoraError::NotLoaded { .. } => Code::Internal,
    };
    Status::new(code, error.to_report_string())
}
```

// gRPC 控制服务的 LoRA 生命周期入口, 与 HTTP 路由共享同一个 LoraManager。

// 注意: 这里直接把 source\_path 传给 state.load\_lora, 尚未接入 HTTP 侧

// 的 validate\_lora\_path\_access 路径前缀校验, 是已知的待跟进安全缺口。

#[tonic::async\_trait]

```
impl pb::control_server::Control for ControlServiceImpl {
    async fn load_lora(
        &self,
        request: Request<pb::LoadLoraRequest>,
    ) -> Result<Response<pb::LoadLoraResponse>, Status> {
        let request = request.into_inner();
        // 统一走 AppState, 复用 LoraManager 的 ID 分配、重名与容量检查;
        // gRPC 场景 load_inplace 与 is_3d_lora_weight 固定为 false, 与 HTTP 默认一致。
    }
}
```

```

        let adapter = self
            .state
            .load_lora(request.lora_name, request.source_path, false, false)
            .await
            .map_err(load_lora_status)?;
        Ok(Response::new(pb::LoadLoraResponse {
            adapter: Some(lora_to_proto(&adapter)),
        }))
    }
}

```

## rust/src/server/src/lora.rs

注册表与错误模型改造核心：错误枚举派生 thiserror、新增 LoraDisabledError/InvalidRequest 变体、服务端自增 ID 分配与重名检查，是 HTTP/gRPC 行为一致性的基石。

```

// 引擎未启用 LoRA 时所有生命周期操作统一返回这个错误。
#[derive(Debug, Error)]
#[error("engine was not started with LoRA enabled")]
pub(crate) struct LoraDisabledError;

impl LoraManager {
    // 加载 adapter 的核心逻辑：先做名称与 ID 的校验和分配，再下发给引擎。
    pub(crate) async fn load_lora(
        // ... 参数省略
    ) -> Result<LoraRequest, LoadLoraError> {
        // 与已加载 adapter 或 base model 名称冲突时直接拒绝。
        if base_model_names.iter().any(|name| name == &lora_name) {
            return Err(LoadLoraError::BaseModelName { lora_name });
        }

        // 统一在应用层读取现有 ID：非 inplace 重名视为 AlreadyLoaded,
        // inplace 或新名称则复用已有 ID 或自增分配新 ID。
        let requests = self.requests.read().await;
        let existing_lora_int_id = requests.get(&lora_name).map(|request| request.lora_int_id);
        if !load_inplace && existing_lora_int_id.is_some() {
            return Err(LoadLoraError::AlreadyLoaded { lora_name });
        }

        let lora_int_id = existing_lora_int_id
            .unwrap_or_else(|| self.id_counter.fetch_add(1, Ordering::Relaxed) + 1);

        // LoraRequest::new 负责非空名称 / 路径与非零 ID 的传输无关校验,
        // 错误统一映射为 InvalidRequest, 供 HTTP 与 gRPC 共用。
        let lora_request = LoraRequest::new(
            lora_name.clone(),
            lora_int_id,
            lora_path,
            load_inplace,
            // ... 其余参数省略
        )
    }
}

```

```

    )?;
    // ... 后续引擎调用与注册表更新省略
    Ok(lora_request)
}
}

```

## rust/src/server/src/state.rs

新增 list\_loras 与 ensure\_lora\_enabled, 把“引擎未启用 LoRA”检查收敛到单点, 是 BugenZhao 重构提交的核心文件。

```

impl AppState {
    // 引擎启动时未开启 LoRA 支持, 则所有生命周期 API 统一返回错误,
    // 避免各调用点各自判断导致行为不一致。
    fn ensure_lora_enabled(&self) -> Result<(), LoraDisabledError> {
        self.engine_core_client()
            .ready_response()
            .supports_lora
            .then_some(())
            .ok_or(LoraDisabledError)
    }

    // 返回当前注册的 adapter 快照, 列表顺序即加载顺序。
    pub async fn list_loras(&self) -> Result<Vec<LoraRequest>, LoraDisabledError> {
        self.ensure_lora_enabled()?;
        Ok(self.lora_manager.served_lora_requests().await)
    }

    // 加载与卸载入口都先过统一门检, 再委托给 LoraManager。
    pub async fn load_lora(
        &self,
        lora_name: String,
        lora_path: String,
        load_inplace: bool,
        is_3d_lora_weight: bool,
    ) -> Result<LoraRequest, LoadLoraError> {
        self.ensure_lora_enabled()?;
        self.lora_manager
            .load_lora(
                self.engine_core_client(),
                &self.served_model_names,
                lora_name,
                lora_path,
                load_inplace,
                is_3d_lora_weight,
            )
            .await
    }
}

```

## 评论区精华

Our HTTP endpoint doesn't seem to support external LoRA ID... is this something we intentionally want in the gRPC interface? — BugenZhao

作者回复：最初为兼容 Dynamo 内部管理 ID 而加入，但调查后确认 Dynamo 侧边车可以让 vLLM 自己分配 ID，因此移除外部 ID 功能，与 HTTP 行为对齐。

🔗 Severity: HIGH ... gRPC load\_lora handler passes source\_path directly ... bypasses VLLM\_RUNTIME\_LORA\_ALLOWED\_PATH\_PREFIXES ... allowing a gRPC client to load LoRA adapters from arbitrary filesystem paths. — depthfirst-app[bot]

作者承认问题并提议 follow-up：把 `validate_lora_path_access` 从 `routes/lora.rs` 移到 `lora.rs` 供双端共享，已有 draft 实现。该问题在合入时尚未修复。

LGTM! I've just pushed some minor refactors to make error handling path a bit cleaner — BugenZhao (APPROVED)

## 风险与影响

- 安全：gRPC `load_lora` 未执行路径前缀校验与 `canonicalize`，已认证的 gRPC 客户端可加载任意本地路径的 adapter；风险随 gRPC 面暴露而生效，已确认延期修复，须跟踪 follow-up。
- 错误语义差异：HTTP 把 Disabled 映射为 400 `invalid_request`，gRPC 映射为 `FailedPrecondition`；两类客户端感知不同，文档需明确。
- 兼容性：新增 RPC 不破坏现有协议；proto 生成代码与 Cargo.lock 同步更新，内部未发布 API 的外部 ID 字段被移除，需确认无早期消费者。
- 回归面：HTTP 加载路径的手写校验删除后依赖 `LoraRequest::new`，行为略有收紧（如 `name/path` 为空现在统一走 `InvalidRequest`），原有 HTTP LoRA 用例需保持通过（测试结果 340 passed）。

## 关联脉络

该 PR 是 Rust 前端 gRPC 控制面能力扩展的一环，作者明确其服务 Dynamo sidecar 集成场景；合入后应关注其提到的路径校验共享 follow-up PR。与历史安全类 PR #53561（音频文件大小限制全路径强制）主题相近：都是“控制面安全约束在多入口间保持一致”的问题，本 PR 在 gRPC 入口留了同类缺口。