PR #51144 完整报告

vllm-project/vllm

[Rust Frontend] Support dynamic tools from developer messages

合并时间：2026-08-11 09:30

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51144

执行摘要

Rust 前端（vllm-chat / vllm-server）将工具处理统一收口到新增的 ResolvedToolContext：请求级工具与 developer 消息中声明的动态工具在入口合并为 effective_tools，tool_choice 默认与校验、parser 激活、xgrammar structural-tag 生成、模板渲染全部共享这份已解析状态，修复了「动态工具到达部分渲染器但不进入下游工具处理」的割裂问题。共变更 28 个文件（+686/-258），cargo nextest 625 项通过，已由 njhill 批准合入。

功能与动机

PR body 明确指出：

> The Rust frontend already carries message-scoped function tools on developer messages, but tool choice resolution, parser activation, and structural-tag generation only used request-level tools. Dynamic-only declarations therefore reached some renderers without becoming available to downstream tool handling.

即仅通过 developer 消息声明工具（例如把 vendor 的 system.tools 形状映射为带空内容的 developer 消息）时，工具虽然被部分渲染器渲染出来，但 tool_choice 解析、parser 激活与结构约束生成都看不到它们，导致动态工具不可用。本 PR 的目标就是让整个工具处理链路消费同一份「已解析」的工具状态。

实现拆解

1. 数据模型收口（rust/src/chat/src/request.rs） - 新增 ResolvedToolContext，字段：initial_tools（请求级、渲染在对话前）、effective_tools（请求级与消息级的有序并集）、tool_choice、parallel_tool_calls。 - ChatRequest 原 tools/tool_choice/parallel_tool_calls 三个公开字段合并为单个 tool_context，并新增 tools()、initial_tools()、tool_choice()、parallel_tool_calls() 访问器。 - ChatMessage::declared_tools() 抽取 developer 消息上的非空工具声明。
2. 解析与校验逻辑（request.rs 的 ResolvedToolContext::new） - 按「请求级 → 消息顺序」拼接 effective_tools，用 HashSet 检测重名，返回新增的 Error::DuplicateToolName。 - 未显式指定 tool_choice 时按 effective_tools 是否为空推导 None/Auto；这使「只有动态工具」的请求自动激活工具解析。 - 统一校验 Auto/Required 必须有可用工具（Error::ToolChoiceRequiresTools）、具名 Function 必须命中并集（Error::ToolChoiceFunctionNotFound）。

3. 消费方适配 - output/default/structural_tag.rs：apply_structural_tag_constraint 与 structural_tag_tool_choice 改用 request.tools()，动态工具也能生成结构标签。 - renderer/hf/mod.rs：模板 tools 变量改用 request.tools()（effective_tools），同时保留 developer 消息自身的 tools 字段。 - renderer/inkling/mod.rs：rendered_tools 删除手工拼接逻辑，直接取 request.tools()。 - DeepSeek V3.2/V4、Kimi K3 编码路径改用新访问器，行为保持不变。
4. 服务入口装配 - server/.../chat_completions/convert.rs：prepare_chat_request 中构造 ResolvedToolContext::new 并映射为 chat_submit_error；convert_tool_choice 签名改为非 Option，显式拒绝 AllowedTools 并列出工具名。 - server/.../tokenize/types.rs：Tokenize 入口同步接入。 - server/.../chat_completions/types.rs：删除 normalize() 中 tool_choice 默认注入，以及 validate_chat_cross_parameters 中 5 项 tool_choice 校验，全部下沉到 resolver。
5. 测试配套 - resolver 单元测试：动态工具默认 Auto、initial 与消息顺序保持、重名拒绝、具名校验。 - structural-tag 新增 required_dynamic_tool_builds_structural_tag_from_effective_tools 与 required_initial_and_dynamic_tools_build_one_structural_tag。 - Kimi K3 golden fixture dynamic_only_tool_declare；DeepSeek V3.2/V4 developer tools fixture（含 renders_developer_tools_like_hf_python 对齐测试）；HF 渲染器 dynamic_tools_are_exposed_as_effective_template_tools。 - cargo nextest run -p vllm-chat -p vllm-server -E 'not test(roundtrip_nemotron_v3)'：625 passed / 1 skipped；clippy、fmt、git diff --check 均通过。

rust/src/chat/src/request.rs

核心数据模型变更：新增 ResolvedToolContext 统一解析请求级与消息级工具，ChatRequest 三字段合并为 tool_context，并新增访问器与错误类型。

```
/// 解析请求级工具与消息级工具，生成统一的工具上下文。
///
/// 顺序约定：请求级工具在前，developer 消息中声明的动态工具按消息
/// 顺序拼接在后；同名工具直接报错而不是静默覆盖，避免下游 parser /
/// structural-tag 出现二义性。
pub fn new(
    messages: &[ChatMessage],
    initial_tools: Vec<ChatTool>,
    requested_tool_choice: Option<ChatToolChoice>,
    parallel_tool_calls: bool,
) -> Result<Self> {
    // 先统计动态工具数量，一次性分配容量，避免逐个 push 时反复扩容
    let dynamic_tool_count = messages
        .iter()
        .filter_map(ChatMessage::declared_tools)
        .map(<[ChatTool]>::len)
        .sum::<usize>();
    let mut effective_tools = Vec::with_capacity(initial_tools.len() + dynamic_tool_count);
    let mut names = HashSet::with_capacity(effective_tools.capacity());

    // 用 HashSet 记录已出现名字：请求级与消息级工具统一去重，重复即报错
```

```
        for tool in initial_tools
            .iter()
            .chain(messages.iter().filter_map(ChatMessage::declared_tools).flatten())
        {
            if !names.insert(tool.name.clone()) {
                return Err(Error::DuplicateToolName {
                    name: tool.name.clone(),
                });
            }
            effective_tools.push(tool.clone());
        }

        // 调用方未显式指定 tool_choice 时按工具并集推导：
        // 没有任何可用工具则视为 None，否则默认 Auto（动态工具因此自动激活解析）
        let tool_choice = requested_tool_choice.unwrap_or({
            if effective_tools.is_empty() {
                ChatToolChoice::None
            } else {
                ChatToolChoice::Auto
            }
        });

        // 统一校验：Auto / Required 必须有可用的 effective_tools；
        // 具名 Function 必须命中并集中的工具名
        match &tool_choice {
            ChatToolChoice::Auto | ChatToolChoice::Required if effective_tools.is_empty() => {
                return Err(Error::ToolChoiceRequiresTools);
            }
            ChatToolChoice::Function { name } if !names.contains(name) => {
                return Err(Error::ToolChoiceFunctionNotFound { name: name.clone() });
            }
            ChatToolChoice::None
            | ChatToolChoice::Auto
            | ChatToolChoice::Required
            | ChatToolChoice::Function { .. } => {}
        }

        Ok(Self {
            initial_tools,
            effective_tools,
            tool_choice,
            parallel_tool_calls,
        })
    }
```

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

Chat Completions 入口在 prepare_chat_request 中装配 ResolvedToolContext，convert_tool_choice 签名收紧并对 AllowedTools 显式报错，是动态工具进入服务端的入口。

```
/// 把 OpenAI 的 tool_choice 形状转换为 vllm-chat 内部表示。
///
/// 与旧版本不同，这里不再接收 `Option<&ToolChoice>`：缺省推导
/// （无工具为 None、有工具为 Auto）已下沉到 `ResolvedToolContext::new`，
/// 本函数只负责形状转换与显式报错。
fn convert_tool_choice(tool_choice: &ToolChoice) -> Result<ChatToolChoice, ApiError> {
    match tool_choice {
        ToolChoice::Value(ToolChoiceValue::Auto) => Ok(ChatToolChoice::Auto),
        ToolChoice::Value(ToolChoiceValue::None) => Ok(ChatToolChoice::None),
        ToolChoice::Value(ToolChoiceValue::Required) => Ok(ChatToolChoice::Required),
        // 仅支持 function 类型；其他 tool_type 落入兜底分支报错
        ToolChoice::Function { tool_type, function } if tool_type == "function" => {
            Ok(ChatToolChoice::Function {
                name: function.name.clone(),
            })
        }
        // 显式拒绝 allowed_tools，并在报错信息中列出引用到的工具名，
        // 方便调用方定位；此前该分支会掉进兜底报错，语义不变但信息更清晰
        ToolChoice::AllowedTools { tools, .. } => bail_invalid_request!(
            "allowed_tools tool_choice is not supported yet: {}.",
            tools.iter().map(ToolReference::identifier).join(", ")
        ),
        _ => bail_invalid_request!("tool_choice={:?} is not supported yet.", tool_choice),
    }
}
```

评论区精华

该 PR 没有代码行级 review 评论，讨论主要来自自动化流程与 maintainer 批准：

> chatgpt-codex-connector[bot]（@codex review）："Didn't find any major issues."
>
> mergify[bot]：pre-commit 失败提示，作者按提示 pre-commit run --all-files 修复后重跑 CI（Buildkite #82690、#83220）通过。
>
> claude[bot]："This pull request is from a fork — automated review is disabled."，随后 njhill 直接 APPROVED 并合入。

设计取舍（来自 PR body 与实现）：系统消息 DTO 保持 content-only，vendor system.tools 形状可映射为带空内容的 developer 消息；AllowedTools 在入口显式拒绝并列出工具名，而不是依赖后续校验兜底。

风险与影响

1. 内部 API 破坏性变更：ChatRequest 删除 tools/tool_choice/parallel_tool_calls 三个公开字段，workspace 内调用点已全量更新，但 vllm-chat/vllm-server crate 外部使用者需迁移到 tool_context。
2. 默认语义变化：无 request-level 工具且未指定 tool_choice 时，resolved 值由原 Auto 变为 None（convert.rs 测试断言同步调整；更符合 Python 语义）。

3. 错误路径收紧：重名工具从静默接受变为 DuplicateToolName 400 错误，是新增失败路径；types.rs 中原 AllowedTools 校验实为死代码，删除无回归风险。
4. 影响范围：限于 Rust chat/server，Python 侧与其它后端不受影响；测试覆盖充分，网络依赖的 Nemotron tokenizer 测试被排除。

关联脉络

- #51178 [Rust Frontend][gRPC] Add explicit data-parallel rank routing：同属 Rust 前端服务化演进线，涉及 rust/server 路由与状态装配，与本 PR 入口层改造区域重叠，可视为同一支团队对 Rust 前端的持续收敛。
- #51235 [Rust Frontend] Upgrade MiniJinja：同为 rust/chat 渲染器仓库内的持续演进，本 PR 的 HF 模板渲染改动与其同域。
- #51654 Fix chat completion 500 on non-object JSON bodies：同为 OpenAI chat completion 入口的校验与错误处理演进，本 PR 将工具校验从 validator 下沉到 resolver/convert，延续了入口层健壮性改进方向。
- #51296 [Bugfix] Align deepseek v4 parser thinking default with tokenizer：DeepSeek 前端行为对齐 Python 的系列工作；本 PR 的 fixture 名 renders_developer_tools_like_hf_python 即为对齐 HF Python 渲染输出而设。