PR #2496 完整报告

radixark/miles

feat(ci): add authorized comment-to-label gateway

合并时间：2026-08-20 14:46

原文链接：http://prhub.com.cn/radixark/miles/pull/2496

执行摘要

- 一句话：新增 PR 评论命令网关，授权加 CI 标签与重跑任务
- 推荐动作：值得精读，尤其关注三点：① COMMAND_REGISTRY 作为 handler、policy key、capability 的唯一事实来源，把评论到动作的映射压缩到一处；② capability 隔离的 GitHub App token 铸造逻辑（issues 与 actions 互斥）；③ 精确 default-deny ACL 与严格 JSON 解析（_strict_object 拒绝重复键）的组合，防止策略文件被绕过。对 CI 维护者，tests/ci/test/test_comment_ci_command.py 展示了如何用 FakeAPI 断言不应发生的调用，是同类网关测试的范例。对普通贡献者，阅读 docs/ci/01-label.md 的 Manage CI from PR comments 一节即可。

功能与动机

Miles 的 CI 是 label 驱动的，但没有仓库写权限的贡献者无法从 PR 讨论中请求一个已批准的 CI 标签；旧的 target-string 路由器把每条评论都当作标签操作，新增非标签命令需要同步修改 workflow、解析器、策略、分发器和 token 选择。维护者需要一个可审查的访问层，委托固定命令而不授予仓库级变更权限，正如 PR body 所述："Workflow owners need one reviewable access layer that delegates fixed commands without granting repository-wide mutation rights."

实现拆解

1. 命令解析与注册表（新增 .github/workflows/scripts/comment_ci_command.py，约 870 行）：把评论解析成精确命令，定义 AddLabel、ClearLabels、RerunFailedCI 类型，COMMAND_REGISTRY 静态注册表将命令与 handler、policy key、token capability 绑定为唯一事实来源；_strict_object 与 _reject_json_constant 提供严格 JSON 解析，拒绝重复键和非标准数字，防止策略文件被绕过；_validate_permissions 与 _validate_user_ids 只接受 write/admin，无关评论在读取策略前即退出（capability = none），不会调用 GitHub API 或铸造 token。
2. ACL 策略（新增 .github/workflows/policies/comment-command-access.json）：version 2 的精确 default-deny ACL。add_label_access 组（仓库 write/admin 或显式 user_ids）可执行 add_label；repo_write_access 组（仅 live write/admin）可执行 clear_labels 和 rerun_failed_ci。allowed_labels 白名单列出 20 个 run-ci-* 标签与 bypass-fastfail，新增 KNOWN_LABELS 条目不会自动暴露为可评论标签。

3. Workflow 编排（新增 .github/workflows/comment-ci-command.yml，97 行）：监听 issue_comment.created，默认关闭（vars.CI_COMMAND_APP_ENABLED == 'true' 才运行）。第一阶段用 github.token 做 preflight 授权并输出 capability（none/issues/actions）；issues 能力时用 GitHub App token（permission-issues: write、permission-pull-requests: read）执行加 / 清标签；actions 能力走独立 job，用 permission-actions: write 铸造 token 后重跑失败任务，并以 concurrency: queue: max 保证同一 PR 的重跑串行、不替换旧请求。两次 checkout 都使用稀疏模式，只取 handler 与 policy 文件，且以 github.sha（基础分支）为准，避免 PR 内容污染信任边界。
4. 测试（新增 tests/ci/test/test_comment_ci_command.py，1422 行，144 个用例）：用 FakeAPI 模拟 GitHub API，覆盖权限校验、fork head、user_ids 精确授权、未知评论早退、clear-labels 只清持久控制标签、rerun 绑定当前 PR head 等工作流契约；通过 register_cpu_ci 挂到 stage-a-cpu CPU 套件，保证在常规 CI 中执行。
5. 文档（更新 docs/ci/01-label.md）：新增 "Manage CI from PR comments" 小节，说明命令的精确语法（整条评论只能有一条命令，允许多余首尾空白、不允许参数 / 正文 / 第二条命令）、default-deny 语义、user_ids 只授权加标签、maintain 角色按 legacy write 处理等。

关键文件：

- tests/ci/test/test_comment_ci_command.py（模块 CI 命令；类别 test；类型 test-coverage；符号 load_module, FakeAPI, init, get_pull）：新增 1422 行测试（144 个用例），用 FakeAPI 完整模拟 GitHub API 调用序列，覆盖授权、拒绝、fork、user_ids、并发契约，是本次变更行为契约的锁定者。
- .github/workflows/scripts/comment_ci_command.py（模块 CI 命令；类别 infra；类型 infrastructure；符号 CommentCommandError, AddLabel, ClearLabels, RerunFailedCI）：命令网关的核心实现（870 行）：负责严格解析评论、校验策略、按 COMMAND_REGISTRY 路由到 AddLabel/ClearLabels/RerunFailedCI handler，并输出 capability 供 workflow 铸造不同权限的 token。
- .github/workflows/comment-ci-command.yml（模块 CI 网关；类别 infra；类型 infrastructure）：新增 workflow 编排：issue_comment 触发、默认禁用、preflight 授权后按 capability 分叉，issues 与 actions 使用互斥的 App token 范围；rerun 以 concurrency queue:max 保证 per-PR 串行。
- .github/workflows/policies/comment-command-access.json（模块 CI 策略；类别 infra；类型 infrastructure）：精确 default-deny ACL：add_label_access/repo_write_access 两组权限与 allowed_labels 白名单，是“谁能做什么”的唯一数据来源。
- docs/ci/01-label.md（模块 CI 文档；类别 docs；类型 documentation）：面向贡献者与维护者的使用说明与安全语义文档，新增 Manage CI from PR comments 一节。

关键符号：load_json, _strict_object, _reject_json_constant, _validate_permissions, _validate_user_ids, AddLabel, ClearLabels, RerunFailedCI, FakeAPI.get_pull, FakeAPI.get_permission, FakeAPI.add_label, FakeAPI.remove_label, FakeAPI.list_workflow_runs, FakeAPI.rerun_failed_jobs, FakeAPI.list_pulls_for_head

关键源码片段

tests/ci/test/test_comment_ci_command.py

新增 1422 行测试（144 个用例），用 FakeAPI 完整模拟 GitHub API 调用序列，覆盖授权、拒绝、fork、user_ids、并发契约，是本次变更行为契约的锁定者。

```
import importlib.util
import json
import urllib.error
import urllib.parse
from pathlib import Path

import pytest
from tests.ci.ci_register import register_cpu_ci
from tests.ci.labels import KNOWN_LABELS

register_cpu_ci(est_time=1, suite="stage-a-cpu", labels=[])

ROOT = Path(__file__).parents[3]
HANDLER_PATH = ROOT / ".github/workflows/scripts/comment_ci_command.py"
POLICY_PATH = ROOT / ".github/workflows/policies/comment-command-access.json"
WORKFLOW_PATH = ROOT / ".github/workflows/comment-ci-command.yml"


def load_module(name, path):
    # 直接按文件路径加载脚本，避免测试与真实 workflow 使用不同入口。
    spec = importlib.util.spec_from_file_location(name, path)
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)
    return module


HANDLER = load_module("comment_ci_command", HANDLER_PATH)
ACTOR_ID = 1234 # 固定测试参与者数字 ID，对应 GitHub 的稳定 user.id。
HEAD_SHA = "a" * 40
HEAD_REF = "feature/test"
WRITE_PERMISSIONS = frozenset({"write", "admin"})


# FakeAPI 记录全部 API 调用，让测试既能断言结果，也能断言“不应发生”的调用，
例如无关评论绝不应触发 get_permission 或加标签。
class FakeAPI:
    def __init__(self, pull, *, permission="write", permission_actor_id=ACTOR_ID):
        self.pull = pull
        self.permission = {
            "permission": permission,
            "user": {"id": permission_actor_id},
        }
        self.calls = []
```

```
        self.get_calls = []
        self.permission_calls = []
        self.add_calls = []
        self.remove_calls = []
        # 为每个可重跑 workflow 准备独立的运行记录，便于验证 rerun 只命中当前 head。
        self.workflow_runs = {workflow_file: [] for workflow_file, _ in HANDLER.RERUN_
        WORKFLOWS}
        self.list_run_calls = []
        self.rerun_calls = []
        self.list_pull_calls = []
        self.head_pulls = [pull]

    def get_pull(self, pull_number):
        self.calls.append(("get_pull", pull_number))
        self.get_calls.append(pull_number)
        return self.pull

    def get_permission(self, actor_login):
        self.calls.append(("get_permission", actor_login))
        self.permission_calls.append(actor_login)
        return self.permission

    def add_label(self, pull_number, label):
        self.calls.append(("add_label", pull_number, label))
        self.add_calls.append((pull_number, label))
        return [*self.pull["labels"], {"name": label}]

    def remove_label(self, pull_number, label):
        self.calls.append(("remove_label", pull_number, label))
        self.remove_calls.append((pull_number, label))
        self.pull["labels"] = [item for item in self.pull["labels"] if item["name"] != label]
        return self.pull["labels"]

    def list_workflow_runs(self, workflow_file, head_sha):
        self.calls.append(("list_workflow_runs", workflow_file, head_sha))
        self.list_run_calls.append((workflow_file, head_sha))
        return self.workflow_runs[workflow_file]

    def rerun_failed_jobs(self, run_id):
        self.calls.append(("rerun_failed_jobs", run_id))
        self.rerun_calls.append(run_id)

    def list_pulls_for_head(self, owner_login, head_ref):
        # 用于把 head ref 解析回当前 open PR，保证 rerun 只作用在精确 head 上。
        self.calls.append(("list_pulls_for_head", owner_login, head_ref))
        self.list_pull_calls.append((owner_login, head_ref))
        return self.head_pulls
```

评论区精华

该 PR 没有产生任何 review 评论，唯一的审核者 yushengsu-thu 直接 APPROVED，说明核心安全设计在作者自述与内部复核后获得认可。设计权衡主要写在 PR body 中：

- 参考 SGLang 的 slash-command-handler.yml，但 Miles 改为使用稳定数字 ID、精确 default-deny 资源、且不 checkout PR 代码（信任边界落在基础分支）。
- user_ids 只授予 add-label 能力；/clear-labels 与 /rerun-failed-ci 始终要求 live write/admin，rerun 还绑定当前 PR head。
- 重跑评论按 PR 串行化，queue: max 保留旧的 pending 请求而不是替换，避免并发重跑互相取消。 结论：作者自称经独立复核无 P0/P1/P2 问题，审核者批准，评审焦点集中在 COMMAND_REGISTRY 单一来源、comment-command-access.json 的精确默认拒绝、workflow 的可信检出与互斥 token 范围。
- 暂无高价值评论线程

风险与影响

- 风险：
  1. 信任边界依赖基础分支：workflow 以 github.sha 稀疏检出 handler 与 policy，PR 无法注入，但拥有基础分支写权限者仍可改动 comment_ci_command.py 扩大权限或外发 App token；该模型等价于仓库写权限信任，需在 App 注册与 rollout 时确认事件仅来自默认分支。
  2. 令牌泄漏面：App token 按 capability 互斥铸造，但 issues 分支的 token 会通过环境变量传递给脚本，若开启 workflow 的 step debug，CI_COMMAND_API_TOKEN 可能进入日志；建议关闭调试并最小化 App 权限（pull-requests 仅 read）。
  3. 行为回归风险：clear-labels 只删除 CLEAR_EXACT_LABELS 中两个持久控制标签，rerun-failed-ci 只重跑三个 PR workflow 且绑定 head SHA；若未来新增 workflow 未同步到 RERUN_WORKFLOWS，重跑会静默遗漏。
  4. 排队限制：GitHub 对 queue: max 并发保留最多 100 个待执行 job，同一 PR 重跑频繁时可能挤占全局配额并导致 5 分钟 timeout。
  5. ACL 手工维护：comment-command-access.json 的 allowed_labels 与 KNOWN_LABELS 需要人工同步，漏加会导致贡献者请求被默认拒绝（安全方向正确，但体验会受影响）。
- 影响：
  - 贡献者：可在 PR 评论里 /run-ci-short 等请求 CI 标签，免去找维护者打标签的等待；无写权限者仍然不能清标签或重跑。
  - 维护者：通过 /clear-labels、/rerun-failed-ci 可自助处理持久控制标签和失败重跑；通过 JSON user_ids 可逐个授信外部协作者，无需授予仓库写权限。
  - 系统：新增仓库级 GitHub App 与两组配置变量 / 密钥；workflow 默认禁用，配置完成前不影响现有 label 驱动 CI。
  - 团队：CI 交互从 label 路由演进为命令网关，后续扩命令（如 test-case）只需在注册表加一项，不必跨文件同步；影响范围集中在 CI/ 权限边界，不涉及训练、rollout、

dashboard 等核心模块。

- 风险标记：默认禁用需 App 配置，token 能力互斥依赖 workflow 逻辑，真实 GitHub App 集成未验证，重跑排队受 100 上限约束，ACL 手工维护

关联脉络

- PR #2651 docs: fix stale paths, flags, env vars and metric names: 同属 CI/ 文档治理线，持续校准 docs/ 与真实 CI 行为；间接关联，未改相同文件。
- PR #2654 docs: name the Slack channel #miles-rl: 修改了 docs/ci/contributor-guide.md，同属 docs/ci 目录维护；间接关联。