Husky — Git Hooks 管理工具

概述

Husky 是 Node.js 生态中最流行的 Git Hooks 管理工具之一，它简化了 `pre-commit`、`commit-msg`、`push` 等钩子的配置和运行流程，让代码质量检查能够自动在提交前执行。

为什么需要 Git Hooks?

在团队协作中，手动遵守代码规范几乎不可能——lint 错误、未格式化、测试失败等问题经常流入仓库。Git Hooks 能在代码提交前自动拦截这些问题，把质量防线前置到客户端而非 CI 阶段。

核心概念

什么是 Git Hooks?

Git 在每个操作完成后会自动查找并执行 `.git/hooks/` 目录下的对应脚本文件。最常见的 Hook 包括：

Hook 名称	触发时机	典型用途
`pre-commit`	执行 `git commit` 时，commit 之前	代码 lint、格式化、单元测试
`commit-msg`	pre-commit 成功后	校验 Commit Message 格式（如 Conventional Commits）

Hook 名称	触发时机	典型用途
`pre-push`	执行 `git push` 之前	全量测试、安全检查
`post-commit`	commit 成功后	通知、日志记录

思考

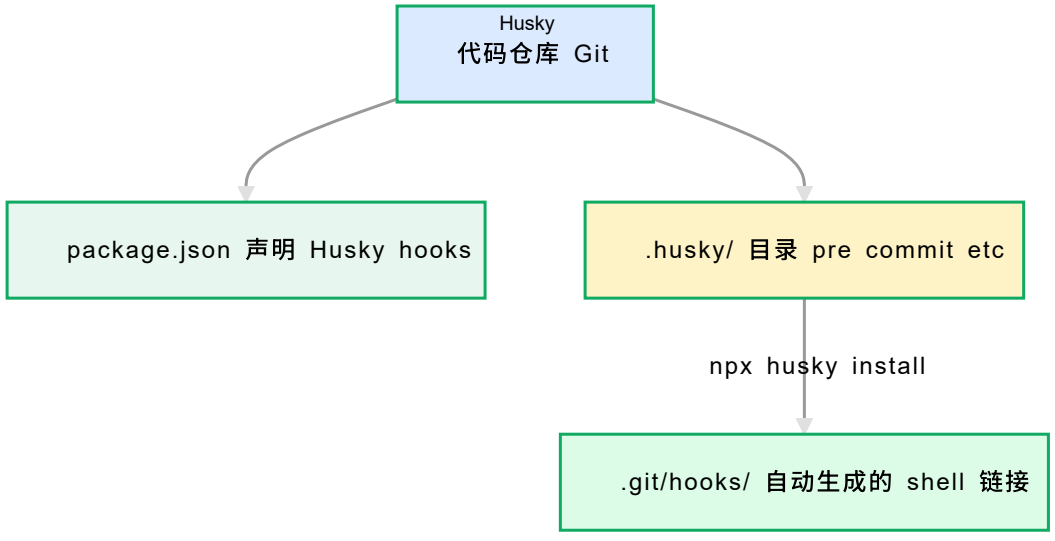
如果某个 Hook 执行返回非零退出码（exit code ≠ 0），会发生什么？

—— Git 会中止当前操作，阻止代码被提交或推送。这就是"拦截"的本质。

关键点：原生 Git Hooks 存在一个痛点——它们存储在 `.git/` 目录下，不会随版本库同步。每个开发者都需要手动配置，容易遗漏或不一致。

Husky 如何解决这个问题？

Husky 的核心理念：**将 Hook 配置写入 `package.json`（即代码仓库本身），通过 `npx husky install` 自动生成 `.git/hooks/` 目录下的脚本。**



这种设计带来的好处：

1. **Hook 配置与代码同源** — 团队成员克隆后只需一条命令即可生效
2. **版本控制** — Hook 脚本的变更可追溯、可 Review
3. **易于维护** — 集中管理，无需分散在各开发者的本地配置中

安装与配置

步骤一：安装

```
# npm
npm install husky --save-dev
```

```
# yarn
yarn add husky --dev


# pnpm
pnpm add husky -D
```

步骤二：初始化

```
# Husky v9+ 使用 pkg-manager 模式
npx husky init
```

这会自动完成两件事：

1. 创建 `.husky/` 目录
2. 生成 `.husky/pre-commit` 示例脚本
3. 在 `package.json` 中添加 `"prepare"` 脚本

```
{
  "scripts": {
    "prepare": "husky"
```

```
  }
}
```

> **为什么用 `prepare` 脚本？**
>
> 当其他开发者执行 `npm install` / `yarn install` / `pnpm install` 时，`prepare` 会自动运行，相当于一次性完成 Husky 初始化。这样就不需要每个人都手动执行 `npx husky init` 了。

步骤三：添加更多 Hook

```
# 创建 commit-msg hook
npx husky add .husky/commit-msg 'npx --no-install commitlint --edit "$1"'

# 创建 pre-push hook
npx husky add .husky/pre-push 'npm test'
```

每个生成的 `.husky/` 目录下的文件都是一个简单的 Shell 脚本：

```
#!/usr/bin/env sh
.nvm/default/node_modules/husky/run.js node "cypress"
```

```
# 或直接执行命令
npm run lint
```

实战集成

集成 Lint + Format

```
# .husky/pre-commit
npx lint-staged
```

配合 `lint-staged` 只处理暂存的文件，大幅加快执行速度：

```
{
  "lint-staged": {
    "*.{js,ts,jsx,tsx}": ["eslint --fix", "prettier --write"],
    "*.json": ["prettier --write"]
  }
}
```

集成 Commit Message 校验

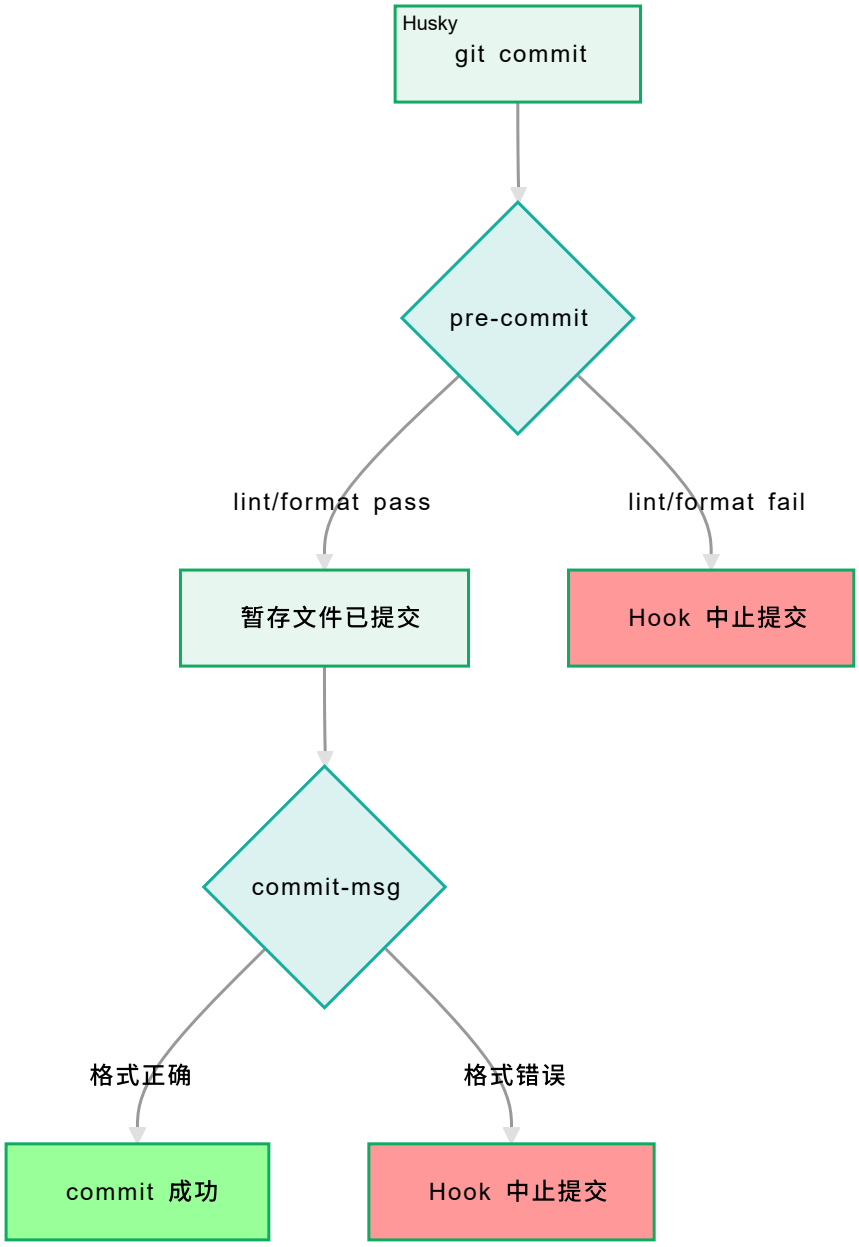
```
# .husky/commit-msg
npx --no-install commitlint --edit "$1"
```

Commitlint 规则示例：

```
// commitlint.config.js
module.exports = {
  extends: ['@commitlint/config-conventional'],
  rules: {
    'type-enum': [2, 'always', [
      'feat', 'fix', 'docs', 'style', 'refactor',
      'test', 'chore', 'perf'
    ]]
  }
};
```

Husky Hooks 执行流程

提交时的 Hook 拦截链路如下：


Husky
git commit
pre-commit
lint/format pass
lint/format fail
暂存文件已提交
Hook 中止提交
commit-msg
格式正确
格式错误
commit 成功
Hook 中止提交

Hook 的退出码机制

每个 Hook 脚本通过 exit code 决定 Git 是否继续——返回 `0` 表示成功放行，返回非 `0` 则中断整个操作。

常见场景与技巧

跳过 Hook（仅个人调试）

```
git commit --no-verify
```

安全警告

`--no-verify` 仅用于紧急调试！不要在 PR 提交时使用，这会绕过所有质量检查。

Conditional Husky

Husky v8+ 支持条件执行，避免在 CI 环境中浪费资源：

```
if [ "$CI" != "true" ]; then
  npx lint-staged
fi
```

Windows 兼容性

Husky v9+ 已内置良好的 Windows 支持，但需注意：

- 确保系统安装了 `sh`（Windows 上的 Git Bash 自带）
- 或在 `.husky/_/husky.sh` 中配置 PowerShell 作为 shell 替代品

大型项目的性能优化

```
{
  "lint-staged": {
    // 按语言分组处理，充分利用并行
    "*.{js,ts}": ["eslint --cache --fix"],
    "*.{css,scss}": ["stylelint --fix"],
    "*.{png,jpg,svg}": ["imagemin"]
  }
}
```

结合 `turbo` 或 `nx` 做增量构建，只重新验证有变化的文件。

与其他方案的对比

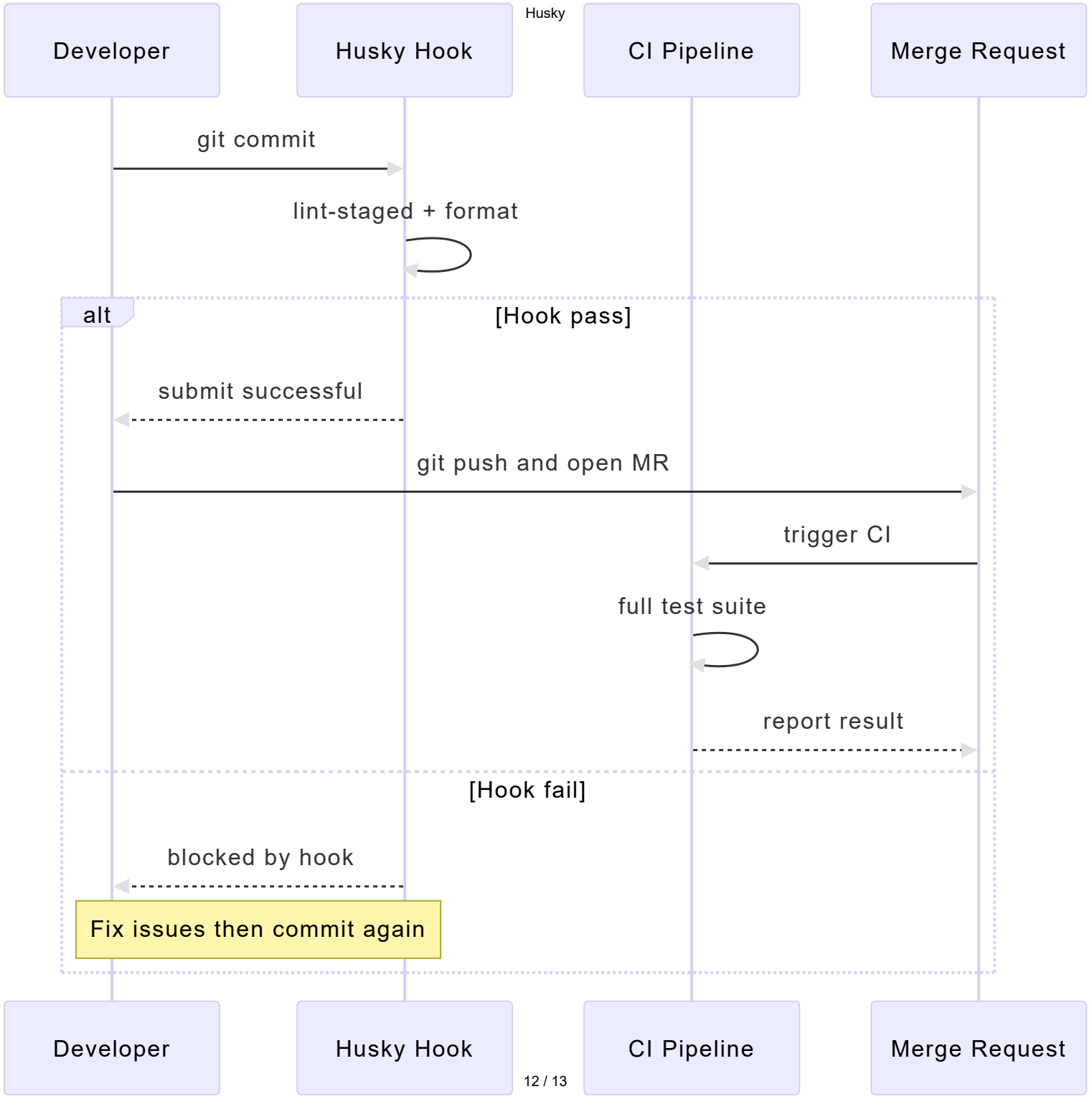
方案	配置方式	跨平台	社区规模	推荐场景
Husky	JS/JSON 声明式	✅ 好	⭐⭐⭐⭐⭐	Node.js / JS 项目首选

方案	配置方式	跨平台	社区规模	推荐场景
Pre-commit (Python)	YAML 配置文件	✅ 好	⭐⭐⭐⭐	多语言混合项目
Ruff + Git Hook	Shell 脚本	✅ 好	⭐⭐⭐	Python 项目快速集成
Rust (cargo-husky)	Cargo 构建脚本	✅ 好	⭐⭐	Rust 项目

选型建议

如果你的项目是 Node.js 技术栈，Husky 是最自然的选择——零额外依赖，与 npm/yarn/pnpm 生态无缝集成。对于 Go/Rust/Python 混合项目，可以考虑 [pre-commit] 框架，它用 YAML 统一管理多种语言的 Hook。

进阶：CI 与 Hooks 的配合策略


Developer
Husky Hook
CI Pipeline
Merge Request
git commit
lint-staged + format
alt
[Hook pass]
submit successful
git push and open MR
trigger CI
full test suite
report result
[Hook fail]
blocked by hook
Fix issues then commit again
Developer
Husky Hook
CI Pipeline
Merge Request

最佳实践：

- **客户端 Hook**（Husky）：快速反馈，只检查暂存文件 + Commit Message 格式
- **服务端 CI**：完整测试套件 + 安全扫描 + 构建验证
- 两者互为补充，前者保体验，后者保安全

关联笔记

- [CLAUDE.md] — Agent 行为配置参考