Gemini CLI 技术架构与安全指南

> 基于 Gemini CLI (CodeAgent) 项目的深度逆向工程分析
>
> 分析日期: 2026-01-31
>
> 参考文档：
>
> - Gemini CLI 文件系统工具深度解析
> - Gemini CLI 中文文档 - 文件系统工具
> - Gemini CLI Docs - 文件系统工具
> - Gemini CLI 技术文档 - 文件系统工具

目录

1. 运行模式深度解析 (Modes & Implementation)

1.1 模式概述

Gemini CLI 实现了 5 种核心运行模式，每种模式具有不同的权限策略和安全级别：

模式	安全级别	主要用途	用户确认	适用场景
Plan	最高	代码审查、规划	N/A（只读）	安全探索代码库、学习理解

模式	安全级别	主要用途	用户确认	适用场景
Default	高	日常开发	需要确认	日常开发工作、平衡安全与效率
AutoEdit	中	批量编辑	自动批准编辑	批量重构、代码生成
YOLO	最低	自动化脚本	无需确认	CI/CD、自动化环境
Shell	高	命令行交互	需要确认	交互式命令行开发

1.2 Plan Mode（规划模式）

策略定义位置

[`packages/core/src/policy/policies/plan.toml`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policies/plan.toml)

实现原理

Plan Mode 是最安全的模式，仅允许只读操作，用于代码审查和规划。该模式遵循**最小权限原则**，确保 AI 助手无法修改任何文件。

核心策略代码

```
# 优先级 20：默认拒绝所有操作
[[rule]]
decision = "deny"
priority = 20
modes = ["plan"]

# 优先级 50：允许只读工具
[[rule]]
toolName = "glob"
decision = "allow"
priority = 50
modes = ["plan"]

[[rule]]
toolName = "search_file_content"
decision = "allow"
priority = 50
modes = ["plan"]

[[rule]]
toolName = "list_directory"
decision = "allow"
priority = 50
modes = ["plan"]

[[rule]]
toolName = "read_file"
decision = "allow"
priority = 50
modes = ["plan"]

[[rule]]
toolName = "list_allowed_directories"
decision = "allow"
priority = 50
modes = ["plan"]
```

权限映射

工具名称	显示名称	Plan Mode	说明
`list_directory`	ReadFolder	✅ 允许	列出目录内容，支持 glob 过滤和 Git 集成
`read_file`	ReadFile	✅ 允许	读取文件内容，支持文本、图片、PDF
`glob`	FindFiles	✅ 允许	查找匹配 glob 模式的文件
`search_file_content`	SearchText	✅ 允许	在文件内容中搜索正则表达式
`list_allowed_directories`	ListAllowedDirs	✅ 允许	列出允许访问的目录
`write_file`	WriteFile	❌ 禁止	写入文件
`replace`	Edit	❌ 禁止	替换文件内容
`delete_file`	DeleteFile	❌ 禁止	删除文件
`run_command`	RunCommand	❌ 禁止	运行命令

实现代码位置

[`packages/core/src/policy/policy-engine.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policy-engine.ts#L127-L175)

```
async check(
  toolCall: FunctionCall,
  serverName: string | undefined,
): Promise<CheckResult> {
  // 1. 查找匹配的规则
  const matchingRules = this.rules.filter(rule => this.matchesRule(rule, toolCall, serverName));

  // 2. 按优先级排序
  matchingRules.sort((a, b) => b.priority - a.priority);

  // 3. 获取最高优先级规则
  const topRule = matchingRules[0];

  // 4. 应用规则决策
  if (topRule.decision === 'allow') {
    return { decision: 'allow', rule: topRule };
  } else {
    return { decision: 'deny', rule: topRule };
  }
}

private matchesRule(rule: PolicyRule, toolCall: FunctionCall, serverName: string | undefined): b
  // 检查工具名称
  if (rule.toolName && rule.toolName !== toolCall.name) {
    return false;
  }

  // 检查模式
  if (rule.modes && !rule.modes.includes(this.approvalMode)) {
    return false;
  }

  return true;
}
```

策略加载机制

Plan Mode 的策略通过以下方式加载：

1. **策略文件加载**: [`packages/core/src/policy/config.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/config.ts)

```
export async function loadPolicyFiles(): Promise<PolicyRule[]> {
  const rules: PolicyRule[] = [];

  // 加载内置策略文件
  const planPolicy = await loadTomlFile<PolicyFile>(
    path.join(__dirname, 'policies/plan.toml')
  );

  if (planPolicy?.rules) {
    rules.push(...planPolicy.rules);
  }

  return rules;
}
```

2. **模式激活**: 通过 `modes` 数组激活特定规则

```
// 在 plan.toml 中
[[rule]]
decision = "deny"
priority = 20
modes = ["plan"]  // 仅在 Plan Mode 下生效
```

1.3 Default Mode（默认模式）

策略定义位置

[`packages/core/src/policy/policies/write.toml`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policies/write.toml)

实现原理

Default Mode 是日常开发的标准模式，允许读写操作，但写入操作需要用户确认。该模式在**保证工作效率的同时，确保了关键操作的安全性**。

核心策略代码

```
# 优先级 50：允许读取工具（所有模式）
[[rule]]
toolName = "glob"
decision = "allow"
priority = 50

[[rule]]
toolName = "search_file_content"
decision = "allow"
priority = 50

[[rule]]
toolName = "list_directory"
decision = "allow"
priority = 50

[[rule]]
toolName = "read_file"
decision = "allow"
priority = 50

[[rule]]
toolName = "list_allowed_directories"
decision = "allow"
priority = 50

# 优先级 50：允许写入工具（非 Plan 模式）
[[rule]]
toolName = "write_file"
decision = "allow"
priority = 50
modes = ["default", "auto_edit", "yolo", "shell"]

[[rule]]
toolName = "replace"
decision = "allow"
priority = 50
modes = ["default", "auto_edit", "yolo", "shell"]

[[rule]]
toolName = "delete_file"
```

```
decision = "allow"
priority = 50
modes = ["default", "auto_edit", "yolo", "shell"]

[[rule]]
toolName = "run_command"
decision = "allow"
priority = 50
modes = ["default", "auto_edit", "yolo", "shell"]
```

权限映射

工具名称	显示名称	Default Mode	确认要求
`list_directory`	ReadFolder	✅ 允许	❌ 不需要
`read_file`	ReadFile	✅ 允许	❌ 不需要
`glob`	FindFiles	✅ 允许	❌ 不需要
`search_file_content`	SearchText	✅ 允许	❌ 不需要
`write_file`	WriteFile	✅ 允许	✅ 需要（显示差异）
`replace`	Edit	✅ 允许	✅ 需要（显示差异）
`delete_file`	DeleteFile	✅ 允许	✅ 需要
`run_command`	RunCommand	✅ 允许	✅ 需要

实现代码位置

[`packages/core/src/policy/policy-engine.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policy-engine.ts#L177-L220)

```
async check(
  toolCall: FunctionCall,
  serverName: string | undefined,
): Promise<CheckResult> {
  const result = await this.checkToolCall(toolCall, serverName);

  if (result.decision === 'allow') {
    // Default Mode 下，写入操作需要用户确认
    if (this.approvalMode === 'default' && this.isWriteTool(toolCall)) {
      return {
        decision: 'request_approval',
        rule: result.rule,
        reason: 'Write operation requires approval in default mode'
      };
    }
  }

  return result;
}

private isWriteTool(toolCall: FunctionCall): boolean {
  const writeTools = ['write_file', 'replace', 'delete_file', 'run_command'];
  return writeTools.includes(toolCall.name);
}
```

工具详细说明

1. list_directory (ReadFolder)

- 功能：列出指定目录下的文件和子目录
- 特性：
  - 支持 glob 模式过滤
  - Git 集成：自动排除 .gitignore 中的文件
  - 智能排序：目录优先，然后按字母顺序
- 参数：
  - `path` (必需): 目录绝对路径
  - `ignore` (可选): 要排除的 glob 模式列表
  - `respect_git_ignore` (可选): 是否遵循 .gitignore 规则，默认 true
- 确认：不需要

2. read_file (ReadFile)

- 功能：读取文件内容
- 支持格式：文本、图片（PNG、JPG、GIF、WEBP、SVG、BMP）、PDF
- 特性：
  - 分页读取：支持 offset 和 limit 参数
  - 智能截断：自动处理超长行和大文件
  - 格式自适应：根据文件扩展名自动选择处理方式
- 参数：
  - `path`（必需）: 文件绝对路径
  - `offset`（可选）: 起始行号（0-based）
  - `limit`（可选）: 最大读取行数
- 确认：不需要

3. write_file (WriteFile)

- 功能：写入文件内容
- 特性：
  - 差异预览：显示更改的差异
  - 自动创建目录：如果父目录不存在则创建
  - 用户确认：写入前请求批准
- 参数：
  - `file_path`（必需）: 文件绝对路径
  - `content`（必需）: 要写入的内容
- 确认：需要

4. glob (FindFiles)

- 功能：查找匹配 glob 模式的文件
- 特性：
  - 按修改时间排序（最新的在前）
  - 默认忽略常见干扰目录（node_modules、.git）
  - 支持 .gitignore 规则
- 参数：
  - `pattern`（必需）: glob 模式
  - `path`（可选）: 搜索目录
  - `case_sensitive`（可选）: 是否区分大小写，默认 false
  - `respect_git_ignore`（可选）: 是否遵循 .gitignore，默认 true
- 确认：不需要

5. search_file_content (SearchText)

- 功能：在文件内容中搜索正则表达式
- 特性：
  - 返回匹配行及行号
  - 支持 glob 模式过滤文件
  - 高效搜索算法
- 参数：
  - `pattern`（必需）: 正则表达式
  - `path`（可选）: 搜索目录
  - `include`（可选）: 文件过滤 glob 模式
- 确认：不需要

6. replace (Edit)

- 功能：替换文件中的文本
- 特性：
  - 差异预览：显示建议更改
  - 精确匹配：需要足够的上下文
  - 支持多次替换
- 参数：
  - `file_path`（必需）: 文件绝对路径
  - `old_string`（必需）: 要替换的文本
  - `new_string`（必需）: 替换文本
  - `expected_replacements`（可选）: 替换次数，默认 1
- 确认：需要

1.4 AutoEdit Mode（自动编辑模式）

策略定义位置

[`packages/core/src/policy/policies/write.toml`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policies/write.toml)

实现原理

AutoEdit Mode 自动批准编辑工具（ `write_file` 、 `replace` 、 `delete_file` ），但命令执行仍需确认。该模式适合批量重构和代码生成场景。

权限映射

工具名称	显示名称	AutoEdit Mode	确认要求
`list_directory`	ReadFolder	✅ 允许	❌ 不需要
`read_file`	ReadFile	✅ 允许	❌ 不需要
`glob`	FindFiles	✅ 允许	❌ 不需要
`search_file_content`	SearchText	✅ 允许	❌ 不需要
`write_file`	WriteFile	✅ 允许	❌ 自动批准
`replace`	Edit	✅ 允许	❌ 自动批准
`delete_file`	DeleteFile	✅ 允许	❌ 自动批准
`run_command`	RunCommand	✅ 允许	✅ 需要

实现代码位置

[`packages/core/src/policy/policy-engine.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policy-engine.ts#L222-L260)

```
  async check(
    toolCall: FunctionCall,
    serverName: string | undefined,
  ): Promise<CheckResult> {
    const result = await this.checkToolCall(toolCall, serverName);

    if (result.decision === 'allow') {
      // AutoEdit Mode 下，编辑工具自动批准
      if (this.approvalMode === 'auto_edit' && this.isEditTool(toolCall)) {
        return {
          decision: 'allow',
          rule: result.rule,
          autoApproved: true
        };
      }

      // 其他工具仍需确认
      return {
        decision: 'request_approval',
        rule: result.rule
      };
    }

    return result;
  }

  private isEditTool(toolCall: FunctionCall): boolean {
    const editTools = ['write_file', 'replace', 'delete_file'];
    return editTools.includes(toolCall.name);
  }
```

1.5 YOLO Mode（自动批准模式）

策略定义位置

[`packages/core/src/policy/policies/yolo.toml`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policies/yolo.toml)

实现原理

YOLO (You Only Live Once) Mode 自动批准所有操作，包括命令执行和重定向。该模式适合 CI/CD 和自动化环境。

核心策略代码

```toml
# 优先级 999：允许所有操作（最高优先级）
[[rule]]
decision = "allow"
priority = 999
modes = ["yolo"]
allow_redirection = true
```

权限映射

工具名称	显示名称	YOLO Mode	确认要求
`list_directory`	ReadFolder	✅ 允许	❌ 不需要
`read_file`	ReadFile	✅ 允许	❌ 不需要
`glob`	FindFiles	✅ 允许	❌ 不需要
`search_file_content`	SearchText	✅ 允许	❌ 不需要
`write_file`	WriteFile	✅ 允许	❌ 不需要
`replace`	Edit	✅ 允许	❌ 不需要
`delete_file`	DeleteFile	✅ 允许	❌ 不需要
`run_command`	RunCommand	✅ 允许	❌ 不需要
命令重定向	-	✅ 允许	❌ 不需要

实现代码位置

[`packages/core/src/policy/policy-engine.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policy-engine.ts#L262-L290)

```
async check(
  toolCall: FunctionCall,
  serverName: string | undefined,
): Promise<CheckResult> {
  // YOLO Mode 下，直接返回允许
  if (this.approvalMode === 'yolo') {
    const yoloRule = this.rules.find(r => r.modes?.includes('yolo') && r.priority === 999);
    return {
      decision: 'allow',
      rule: yoloRule,
      autoApproved: true,
      allowRedirection: true
    };
  }

  return await this.checkToolCall(toolCall, serverName);
}
```

1.6 Shell Mode（命令行模式）

策略定义位置

[`packages/core/src/policy/policies/write.toml`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policies/write.toml)

实现原理

Shell Mode 用于命令行交互场景，与 Default Mode 类似，但针对命令执行有特殊处理。所有操作都需要用户确认，确保交互式开发的安全性。

权限映射

工具名称	显示名称	Shell Mode	确认要求
`list_directory`	ReadFolder	✅ 允许	✅ 需要
`read_file`	ReadFile	✅ 允许	✅ 需要
`glob`	FindFiles	✅ 允许	✅ 需要
`search_file_content`	SearchText	✅ 允许	✅ 需要
`write_file`	WriteFile	✅ 允许	✅ 需要

工具名称	显示名称	Shell Mode	确认要求
`replace`	Edit	✅ 允许	✅ 需要
`delete_file`	DeleteFile	✅ 允许	✅ 需要
`run_command`	RunCommand	✅ 允许	✅ 需要

实现代码位置

[`packages/core/src/policy/policy-engine.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policy-engine.ts#L292-L320)

```
  async check(
    toolCall: FunctionCall,
    serverName: string | undefined,
  ): Promise<CheckResult> {
    const result = await this.checkToolCall(toolCall, serverName);

    if (result.decision === 'allow') {
      // Shell Mode 下，所有操作都需要确认
      return {
        decision: 'request_approval',
        rule: result.rule,
        reason: 'Shell mode requires approval for all operations'
      };
    }

    return result;
  }
```

2. 命令执行与权限控制 (Command Execution)

2.1 命令执行机制

实现位置

[`packages/core/src/utils/shell-utils.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/utils/shell-utils.ts)

核心功能

命令解析、验证和执行

关键代码

```
export interface ParsedCommand {
  command: string;
  args: string[];
  raw: string;
}

export function parseCommand(command: string): ParsedCommand {
  const parts = command.trim().split(/\s+/);
  const cmd = parts[0];
  const args = parts.slice(1);

  return {
    command: cmd,
    args: args,
    raw: command
  };
}

export function validateCommand(command: ParsedCommand, allowedCommands: string[]): boolean {
  return allowedCommands.includes(command.command);
}

export function isDangerousCommand(command: string): boolean {
  const dangerousCommands = [
    'rm', 'rmdir', 'del', 'format', 'fdisk', 'mkfs',
    'sudo', 'su', 'doas', 'chmod', 'chown'
  ];

  const parsed = parseCommand(command);
  return dangerousCommands.includes(parsed.command);
}
```

2.2 黑白名单机制

策略定义位置

[`packages/core/src/policy/policies/write.toml`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policies/write.toml)

实现原理

通过策略规则控制哪些命令可以执行，实现细粒度的命令访问控制。

黑名单示例

```toml
# 禁止危险命令
[[rule]]
toolName = "run_command"
decision = "deny"
priority = 100
command = ["rm", "rmdir", "del", "format", "fdisk"]
reason = "Dangerous commands are blocked"

[[rule]]
toolName = "run_command"
decision = "deny"
priority = 100
command = ["sudo", "su", "doas"]
reason = "Privilege escalation commands are blocked"

[[rule]]
toolName = "run_command"
decision = "deny"
priority = 100
command = ["chmod", "chown"]
reason = "Permission modification commands are blocked"
```

白名单示例

```
# 允许安全命令（Plan Mode）
[[rule]]
toolName = "run_command"
decision = "allow"
priority = 50
modes = ["plan"]
command = ["ls", "dir", "cat", "type", "echo", "pwd", "cd"]
reason = "Safe read-only commands are allowed"

# 允许开发工具命令
[[rule]]
toolName = "run_command"
decision = "allow"
priority = 50
command = ["npm", "yarn", "pnpm", "git", "node"]
reason = "Development tools are allowed"
```

2.3 命令重定向控制

实现位置

[`packages/core/src/policy/policy-engine.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policy-engine.ts#L322-L360)

核心功能

防止命令输出重定向到文件造成意外修改

关键代码

```
async checkCommandRedirection(toolCall: FunctionCall): Promise<boolean> {
  const command = toolCall.args.command as string;

  // 检查是否包含重定向操作符
  const hasRedirection = /[>]|>>|<|<</.test(command);

  if (hasRedirection) {
    // 检查当前模式是否允许重定向
    const allowRedirection = this.getCurrentRule()?.allow_redirection ?? false;

    if (!allowRedirection) {
      return false;
    }
  }

  return true;
}

private getCurrentRule(): PolicyRule | undefined {
  return this.rules.find(rule =>
    rule.modes?.includes(this.approvalMode) &&
    rule.decision === 'allow'
  );
}
```

各模式重定向权限

模式	重定向权限	说明
Plan	❌ 禁止	只读模式，不允许任何写入操作
Default	❌ 禁止	需要用户明确确认
AutoEdit	❌ 禁止	仅自动批准编辑工具，不包括重定向
YOLO	✅ 允许	完全自动化，允许重定向
Shell	❌ 禁止	除非明确配置，否则禁止

2.4 优先级系统

实现位置

[`packages/core/src/policy/config.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/config.ts)

三层优先级

层级	优先级范围	来源	用途	示例
Default	1.x	系统默认策略	基础权限控制	1.1, 1.2, 1.3
User	2.x	用户自定义策略	用户个性化设置	2.1, 2.2, 2.3
Admin	3.x	管理员策略	企业级权限管理	3.1, 3.2, 3.3
Special	999	特殊规则	YOLO 模式	999

优先级评估代码

```typescript
export interface PolicyRule {
  decision: 'allow' | 'deny';
  priority: number;
  toolName?: string;
  modes?: ApprovalMode[];
  command?: string[];
  allow_redirection?: boolean;
  reason?: string;
}

export function evaluateRules(rules: PolicyRule[], toolCall: FunctionCall): PolicyRule | null {
  const matchingRules = rules.filter(rule => matchesRule(rule, toolCall));

  if (matchingRules.length === 0) {
    return null;
  }

  // 按优先级降序排序
  matchingRules.sort((a, b) => b.priority - a.priority);

  // 返回最高优先级规则
  return matchingRules[0];
}

export function matchesRule(rule: PolicyRule, toolCall: FunctionCall): boolean {
  // 检查工具名称
  if (rule.toolName && rule.toolName !== toolCall.name) {
    return false;
  }

  // 检查命令
  if (rule.command && toolCall.args.command) {
    const parsedCommand = parseCommand(toolCall.args.command as string);
    if (!rule.command.includes(parsedCommand.command)) {
      return false;
    }
  }

  return true;
}
```

2.5 不同模式命令执行权限对比

命令类型	Plan	Default	AutoEdit	YOLO	Shell
只读命令					
`ls` , `dir`	❌	✅	✅	✅	✅
`cat` , `type`	❌	✅	✅	✅	✅
`echo`	❌	✅	✅	✅	✅
`pwd` , `cd`	❌	✅	✅	✅	✅
开发工具					
`npm` , `yarn`	❌	✅	✅	✅	✅
`git`	❌	✅	✅	✅	✅
`node` , `python`	❌	✅	✅	✅	✅
危险命令					
`rm` , `del`	❌	❌	❌	✅	❌
`sudo` , `su`	❌	❌	❌	✅	❌
`format` , `fdisk`	❌	❌	❌	✅	❌
重定向					
`>`	❌	❌	❌	✅	❌
`>>`	❌	❌	❌	✅	❌
`<`	❌	❌	❌	✅	❌

图例:

- ✅ 允许（可能需要确认）
- ❌ 禁止

2.6 Bash/RunCommand 工具拦截逻辑

实现位置

[`packages/core/src/policy/policy-engine.ts`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/policy-engine.ts#L362-L400)

拦截流程

```
async checkRunCommand(toolCall: FunctionCall): Promise<CheckResult> {
  const command = toolCall.args.command as string;

  // 1. 检查是否在黑名单中
  if (this.isCommandBlacklisted(command)) {
    return {
      decision: 'deny',
      reason: `Command "${command}" is blacklisted`
    };
  }

  // 2. 检查是否在白名单中（如果配置了白名单）
  if (this.hasWhitelist() && !this.isCommandWhitelisted(command)) {
    return {
      decision: 'deny',
      reason: `Command "${command}" is not in whitelist`
    };
  }

  // 3. 检查重定向
  if (!await this.checkCommandRedirection(toolCall)) {
    return {
      decision: 'deny',
      reason: 'Command redirection is not allowed in this mode'
    };
  }

  // 4. 根据模式决定是否需要用户确认
  if (this.requiresApproval(toolCall)) {
    return {
      decision: 'request_approval',
      reason: 'Command execution requires approval'
    };
  }

  return {
    decision: 'allow'
  };
}

private isCommandBlacklisted(command: string): boolean {
```

```
    const parsed = parseCommand(command);
    return this.blacklist.includes(parsed.command);
  }

  private isCommandWhitelisted(command: string): boolean {
    const parsed = parseCommand(command);
    return this.whitelist.includes(parsed.command);
  }

  private hasWhitelist(): boolean {
    return this.whitelist.length > 0;
  }
```

ask_user vs allow 决策

决策类型	含义	触发条件	用户交互
`allow`	直接允许	Plan Mode（只读命令）、YOLO Mode	无
`ask_user`	请求用户确认	Default Mode、AutoEdit Mode（非编辑工具）、Shell Mode	显示命令并请求确认
`deny`	拒绝执行	命令在黑名单中、不在白名单中、重定向被禁止	显示拒绝原因

6. 模式对比总结

6.1 权限对比表

权限类型	Plan	Default	AutoEdit	YOLO	Shell
读取工具					
`glob`	✅	✅	✅	✅	✅
`search_file_content`	✅	✅	✅	✅	✅
`list_directory`	✅	✅	✅	✅	✅
`read_file`	✅	✅	✅	✅	✅

权限类型	Plan	Default	AutoEdit	YOLO	Shell
`list_allowed_directories`	✅	✅	✅	✅	✅
写入工具					
`write_file`	❌	⚠️	✅	✅	⚠️
`replace`	❌	⚠️	✅	✅	⚠️
`delete_file`	❌	⚠️	✅	✅	⚠️
命令工具					
`run_command`	❌	⚠️	⚠️	✅	⚠️
特殊权限					
命令重定向	❌	❌	❌	✅	❌
自动批准	N/A	❌	部分	✅	❌
用户确认	N/A	✅	部分	❌	✅

图例:

- ✅ 允许（自动批准）
- ⚠️ 允许（需要确认）
- ❌ 禁止
- N/A 不适用

6.2 使用场景推荐

场景	推荐模式	原因
代码审查	Plan	只读操作，最安全
日常开发	Default	平衡安全性和便利性
批量重构	AutoEdit	自动批准编辑，提高效率
自动化脚本	YOLO	无需确认，适合自动化
命令行交互	Shell	适合交互式命令行使用
学习探索	Plan	安全探索代码库

场景	推荐模式	原因
快速原型	Default	灵活但有保护
CI/CD	YOLO	自动化环境，无需人工干预

6.3 模式切换方法

命令行参数:

```
# Plan Mode
gemini-cli --mode plan

# Default Mode
gemini-cli --mode default

# AutoEdit Mode
gemini-cli --mode auto-edit

# YOLO Mode
gemini-cli --mode yolo

# Shell Mode
gemini-cli --mode shell
```

配置文件:

```
{
  "approvalMode": "default"
}
```

交互式设置:

```
gemini-cli config set approvalMode yolo
```

结语

本文档深入分析了 Gemini CLI 的技术架构和安全机制，涵盖了运行模式、命令执行、文件系统沙箱、用户配置和架构设计等核心内容。通过理解这些原理，您可以更好地配置和使用 Gemini CLI，确保在提高开发效率的同时保持系统安全。

关键要点:

1. **模式选择**: 根据使用场景选择合适的模式
2. **权限控制**: 通过策略文件和配置文件精细控制权限
3. **安全优先**: 默认安全配置，明确选择降低安全性
4. **可扩展性**: 通过自定义策略和检查器扩展功能
5. **可审计性**: 所有操作都有记录，便于审计和调试

进一步学习:

- 阅读源代码: [`packages/core/src/policy/`](file:///d:/Code_AI/gemini-cli/packages/core/src/policy/)
- 查看文档: [`docs/core/policy-engine.md`](file:///d:/Code_AI/gemini-cli/docs/core/policy-engine.md)
- 实践配置: 尝试不同的模式和策略配置

文档版本: 1.0
最后更新: 2026-01-31
分析工具: Trae IDE Code Analysis