🔧 Gemini CLI - GUI 深度定制 API 完整参考

基于 `@google/gemini-cli-core` v0.30.0 源码分析 · 提取全部可编程接口：

📦 核心包信息

NPM @google/gemini-cli-core

GitHub https://github.com/google-gemini/gemini-cli/tree/main/packages/core

Latest 0.28.0

Nightly 0.30.0

🚀 安装命令

```
# 稳定版
npm install @google/gemini-cli-core
# 预览版 / Nightly
npm install @google/gemini-cli-core@nightly
```

> 💡 官方持续发布，支持直接 `import` 使用，无需克隆 Monorepo。

Core 核心 | Tools 工具 | Events 事件 | Hooks 钩子 | Config 配置

🏗️ 架构总览 - Monorepo 包结构

@google/gemini-cli

CLI 入口 + Ink (React) TUI 渲染层
依赖 core 包

@google/gemini-cli-core ⭐

核心库 - GUI 定制的主要目标
所有业务逻辑、工具、事件、配置

a2a-server

Agent-to-Agent 协议服务器

vscode-ide-companion

VS Code IDE 集成伴侣

> 💡 **GUI 定制核心策略**: 替换 `packages/cli` 的 Ink TUI 层，直接引用 `@google/gemini-cli-core` 提供的所有 API，构建自己的前端（Electron / Web / Tauri）。

> ⚠️ **关键依赖**: Core 基于 `@google/genai` SDK，认证支持 OAuth、API Key、Vertex AI 三种模式。

📦 Core 包完整导出清单 (index.ts)

以下是 `packages/core/src/index.ts` 的全部 export 分类，均可通过 `import { ... } from '@google/gemini-cli-core'` 引用：

分类	模块路径	包含内容

Config	config/config, config/memory, config/models, config/constants, config/storage	Config 类, ConfigParameters, HierarchicalMemory, 模型常量, Storage
Core	core/client, core/contentGenerator, core/geminiChat, core/turn, core/prompts, core/tokenLimits, core/geminiRequest, core/coreToolScheduler	GeminiClient, ContentGenerator, GeminiChat, Turn, GeminiEventType
Tools	tools/tools, tools/tool-registry, tools/tool-names, tools/read-file, tools/ls, tools/grep, tools/glob, tools/edit, tools/write-file, tools/shell, tools/web-fetch, tools/web-search, tools/memoryTool, tools/read-many-files, tools/mcp-tool, tools/write-todos	全部内置工具定义 + ToolRegistry + ToolBuilder 接口
Events	utils/events	CoreEvent 枚举, CoreEventEmitter, 所有 Payload 类型
Hooks	hooks/index, hooks/types	HookSystem, HookEventName, HookDefinition, HookOutput
Scheduler	scheduler/scheduler, scheduler/types, scheduler/tool-executor	ToolCall 状态机, ToolCallRequestInfo, ToolCallResponseInfo
Services	services/fileDiscoveryService, services/gitService, services/chatRecordingService, services/fileSystemService, services/contextManager, services/shellExecutionService	文件发现、Git、会话录制、上下文管理等
MessageBus	confirmation-bus/message-bus, confirmation-bus/types	MessageBus, ToolConfirmation 请求/响应
Output	output/types, output/json-formatter, output/stream-json-formatter	OutputFormat, JsonStreamEvent, JsonOutput
Policy	policy/types, policy/policy-engine, policy/config	PolicyEngine, ApprovalMode, PolicyDecision
Auth	code_assist/oauth2, code_assist/setup, code_assist/types, code_assist/telemetry	OAuth2 流程, Code Assist 集成, 用户层级
MCP	tools/mcp-client, mcp/oauth-provider, mcp/oauth-token-storage	MCP 客户端, OAuth Provider, Token 存储
Agents	agents/types, agents/agentLoader, agents/local-executor	AgentDefinition, AgentLoader, SubagentTool
Utilities	utils/fetch, utils/paths, utils/headless, utils/errors, utils/gitUtils, utils/checkpointUtils, utils/events, utils/browser 等 30+ 模块	工具函数集合

Config 类 - 全局配置中心

`new Config(params: ConfigParameters)` Class

全局配置对象，几乎所有核心功能都依赖于它。GUI 初始化时必须构造此对象。

ConfigParameters 关键参数：

参数	类型	说明
`sessionId`	string	会话唯一标识
`model`	string	模型名称 (如 gemini-2.5-pro, gemini-2.5-flash)
`targetDir`	string	目标项目目录

cwd	string	当前工作目录
debugMode	boolean	调试模式
approvalMode	ApprovalMode	工具审批模式 (AUTO/CONFIRM/YOLO)
mcpServers	Record<string, MCPServerConfig>	MCP 服务器配置映射
interactive	boolean	是否交互模式
question	string?	Headless 模式下的初始问题
checkpointing	boolean	启用检查点保存
includeDirectories	string[]	额外包含的目录
output	OutputSettings	输出格式 (TEXT/JSON/STREAM_JSON)
maxSessionTurns	number	最大会话轮次
enableHooks	boolean	启用 Hook 系统
hooks	HookDefinition[]	用户级 Hook 定义
enableAgents	boolean	启用 Agent 子系统
agents	AgentSettings	Agent 覆盖配置
policyEngineConfig	PolicyEngineConfig	策略引擎配置
plan	boolean	Plan 模式
shellExecutionConfig	ShellExecutionConfig	Shell 执行配置
proxy	string?	代理服务器
accessibility	AccessibilitySettings	无障碍设置
telemetry	TelemetrySettings	遥测设置
contextFileName	string | string[]	上下文文件名 (默认 GEMINI.md)
eventEmitter	EventEmitter	外部事件发射器
toolOutputMasking	ToolOutputMaskingConfig	工具输出掩码配置

Config 核心 Getter 方法

```
// 模型相关
config.getModel(): string
config.getActiveModel(): string
config.setModel(model: string): void

// 工具注册表
config.getToolRegistry(): ToolRegistry
config.getContentGenerator(): ContentGenerator
config.getGeminiClient(): GeminiClient

// 服务
config.getFileDiscoveryService(): FileDiscoveryService
config.getGitService(): GitService
config.getContextManager(): ContextManager

// 路径与环境
```

```
config.getProjectRoot(): string
config.getSessionId(): string
config.getProxy(): string | undefined
config.getCwd(): string

// 策略与审批
config.getPolicyEngine(): PolicyEngine
config.getMessageBus(): MessageBus
config.getHookSystem(): HookSystem
config.getApprovalMode(): ApprovalMode
```

ApprovalMode 枚举

```
enum ApprovalMode {
  ALWAYS_CONFIRM,    // 所有工具调用都需确认
  AUTO_APPROVE,      // 自动批准（基于策略）
  YOLO               // 全自动，不确认
}
```

CustomTheme - 自定义主题 (GUI 直接可用)

```
interface CustomTheme {
  type: 'custom';
  name: string;
  text?: { primary?, secondary?, link?, accent?, response? };
  background?: { primary?, diff?: { added?, removed? } };
  border?: { default?, focused? };
  ui?: { comment?, symbol?, gradient?: string[] };
  status?: { error?, success?, warning? };
}
```

ContentGenerator - 内容生成接口

interface ContentGenerator Interface

```
interface ContentGenerator {
  generateContent(
    request: GenerateContentParameters,
    userPromptId: string
  ): Promise<GenerateContentResponse>;

  generateContentStream(
    request: GenerateContentParameters,
    userPromptId: string
  ): Promise<AsyncGenerator<GenerateContentResponse>>;

  countTokens(
    request: CountTokensParameters
  ): Promise<CountTokensResponse>;

  embedContent(
    request: EmbedContentParameters
  ): Promise<EmbedContentResponse>;

  userTier?: UserTierId;
```

```
  userTierName?: string;
}
```

通过 `createContentGenerator()` 工厂函数创建，支持 OAuth / API Key / Vertex AI 三种认证。

AuthType 枚举

Enum

```
enum AuthType {
  LOGIN_WITH_GOOGLE = 'oauth-personal',
  USE_GEMINI        = 'gemini-api-key',
  USE_VERTEX_AI     = 'vertex-ai',
  LEGACY_CLOUD_SHELL= 'cloud-shell',
  COMPUTE_ADC       = 'compute-default-credentials',
}
```

💬 GeminiChat - 会话管理核心

class GeminiChat

Class

```
class GeminiChat {
  constructor(
    config: Config,
    systemInstruction: string,
    tools: Tool[],
    history: Content[],
    resumedSessionData?: ResumedSessionData,
    onModelChanged?: (modelId: string) => Promise<Tool[]>
  )

  // 核心方法
  setSystemInstruction(sysInstr: string): void

  async sendMessageStream(
    modelConfigKey: ModelConfigKey,
    message: PartListUnion,
    prompt_id: string,
    signal: AbortSignal,
    displayContent?: PartListUnion
  ): Promise<AsyncGenerator<StreamEvent>>

  // 上下文管理
  getHistory(): Content[]
  clearHistory(): void
  compressHistory(): Promise<ChatCompressionInfo>
  addToolResponse(completed: CompletedToolCall[]): void
}
```

StreamEvent 联合类型

Type

```
type StreamEvent =
  | { type: 'chunk';  value: GenerateContentResponse }
```

```
  | { type: 'retry' }
  | { type: 'agent_execution_stopped'; reason: string }
  | { type: 'agent_execution_blocked'; reason: string }
```

🔄 Turn 类 - Agentic Loop 事件流 (GUI 渲染核心)

> 🎯 **这是 GUI 最重要的 API**：Turn.run() 返回的 AsyncGenerator 是驱动整个 UI 渲染的事件源。

`Turn.run()` async* AsyncGenerator

```
async *run(
  modelConfigKey: ModelConfigKey,
  req: PartListUnion,
  signal: AbortSignal,
  displayContent?: PartListUnion
): AsyncGenerator<ServerGeminiStreamEvent>
```

`GeminiEventType` 枚举 - 全部 17 种事件 Enum

事件类型	Value	说明	GUI 用途
Content	{ value: string, traceId? }	模型文本输出（流式）	✅ 实时渲染 Markdown
Thought	{ value: ThoughtSummary, traceId? }	模型思考过程	✅ 折叠显示推理链
ToolCallRequest	{ value: ToolCallRequestInfo }	工具调用请求	✅ 显示工具调用卡片
ToolCallResponse	{ value: ToolCallResponseInfo }	工具执行结果	✅ 显示执行结果
ToolCallConfirmation	{ value: { request, details } }	需用户确认	✅ 弹出确认对话框
UserCancelled	{}	用户取消	停止渲染
Error	{ value: { error: StructuredError } }	错误	✅ 显示错误提示
ChatCompressed	{ value: ChatCompressionInfo }	上下文压缩	通知条
Finished	{ value: { reason, usageMetadata } }	回合完成	✅ Token 统计面板
MaxSessionTurns	{}	达到最大轮次	警告弹窗
LoopDetected	{}	检测到循环	警告提示
Citation	{ value: string }	引用来源	✅ 引用链接渲染
Retry	{}	正在重试	Loading 状态
ContextWindowWillOverflow	{ estimatedRequestTokenCount, remainingTokenCount }	上下文即将溢出	容量警告条
InvalidStream	{}	无效流	错误处理

ModelInfo	{ value: string }	模型信息	状态栏更新
AgentExecutionStopped	{ reason, systemMessage? }	Agent 停止	停止动画

📤 输出格式 - Headless / Stream JSON

JsonStreamEvent 类型 (stream-json 模式) Type

```
enum JsonStreamEventType {
  INIT        = 'init',        // { session_id, model }
  MESSAGE     = 'message',     // { role, content, delta? }
  TOOL_USE    = 'tool_use',    // { tool_name, tool_id, parameters }
  TOOL_RESULT = 'tool_result', // { tool_id, status, output?, error? }
  ERROR       = 'error',       // { severity, message }
  RESULT      = 'result',      // { status, stats? }
}

// 使用: gemini -p "prompt" --output-format stream-json
// 输出 NDJSON (每行一个 JSON 事件)
```

💡 **GUI 最佳实践**: 如果你的 GUI 通过子进程调用 gemini-cli，使用 `--output-format stream-json` 直接解析 NDJSON 流。如果直接嵌入 core 库，则使用 Turn 的 AsyncGenerator。

🛠️ 内置工具完整列表

工具名	类	功能	Kind
read_file	ReadFileTool	读取文件内容	ReadOnly
read_many_files	ReadManyFilesTool	批量读取多个文件	ReadOnly
ls	LSTool	列出目录内容	ReadOnly
glob	GlobTool	文件模式匹配搜索	ReadOnly
grep / ripgrep	GrepTool / RipGrepTool	文件内容搜索	ReadOnly
replace	EditTool	编辑/替换文件内容 (基于 diff)	Write
write_file	WriteFileTool	创建/覆写文件	Write
shell	ShellTool	执行 Shell 命令	Dangerous
web_fetch	WebFetchTool	获取网页内容	Network
google_web_search	WebSearchTool	Google 搜索	Network
save_memory	MemoryTool	保存记忆到 GEMINI.md	Write
write_todos	WriteTodosTool	写入 TODO 列表	Write

ask_user	AskUserTool	向用户提问 (GUI 关键)	Interactive
enter_plan_mode	EnterPlanModeTool	进入计划模式	Other
exit_plan_mode	ExitPlanModeTool	退出计划模式	Other
activate_skill	ActivateSkillTool	激活技能	Other

📋 ToolRegistry - 工具注册表

class ToolRegistry

Class

```
class ToolRegistry {
  registerTool(tool: AnyDeclarativeTool): void
  unregisterTool(name: string): void
  getTool(name: string): AnyDeclarativeTool | undefined
  getActiveTools(): Map<string, AnyDeclarativeTool>
  getAllTools(): Map<string, AnyDeclarativeTool>
  getToolSchemas(modelId?: string): FunctionDeclaration[]
  sortTools(): void
  isActive(name: string): boolean
  getMessageBus(): MessageBus
}
```

> 💡 **自定义工具**: 继承 `DeclarativeTool<TParams, TResult>` 或 `BaseDeclarativeTool` 来创建自定义工具并注册到 ToolRegistry。

ToolBuilder / ToolInvocation 接口

Interface

```
interface ToolBuilder<TParams, TResult> {
  name: string;                // 内部名称
  displayName: string;         // 显示名称 → GUI 使用
  description: string;         // 工具描述
  kind: Kind;                   // ReadOnly | Write | Dangerous | Network | Other
  isOutputMarkdown: boolean;   // 输出是否为 Markdown
  canUpdateOutput: boolean;    // 是否支持流式输出
  getSchema(modelId?): FunctionDeclaration;
  build(params): ToolInvocation;
}

interface ToolInvocation<TParams, TResult> {
  params: TParams;
  getDescription(): string;              // → GUI 显示操作描述
  toolLocations(): ToolLocation[];       // → GUI 显示受影响路径
  shouldConfirmExecute(signal): Promise; // → GUI 弹出确认框
  execute(signal, updateOutput?): Promise<TResult>;
}
```

⏱️ Scheduler - 工具调用调度器

ToolCall 状态机 (GUI 渲染工具调用卡片的数据源) Type

```
type ToolCall =
  | { status: 'validating';        request, tool, invocation }
  | { status: 'scheduled';         request, tool, invocation }
  | { status: 'executing';         request, tool, invocation, liveOutput?, pid? }
  | { status: 'awaiting_approval'; request, tool, invocation, confirmationDetails }
  | { status: 'success';           request, tool, response, invocation, durationMs? }
  | { status: 'error';             request, response, durationMs? }
  | { status: 'cancelled';         request, response, tool, invocation }

// 关键数据结构
interface ToolCallRequestInfo {
  callId: string;
  name: string;
  args: Record<string, unknown>;
  isClientInitiated: boolean;
  prompt_id: string;
  traceId?: string;
  parentCallId?: string;
  schedulerId?: string;
}

interface ToolCallResponseInfo {
  callId: string;
  responseParts: Part[];
  resultDisplay: ToolResultDisplay | undefined;
  error: Error | undefined;
  errorType: ToolErrorType | undefined;
  contentLength?: number;
  data?: Record<string, unknown>;
}
```

Scheduler 回调 Handlers

```
type ConfirmHandler = (toolCall: WaitingToolCall) => Promise<ToolConfirmationOutcome>
type OutputUpdateHandler = (callId: string, chunk: string | AnsiOutput) => void
type AllToolCallsCompleteHandler = (completed: CompletedToolCall[]) => Promise<void>
type ToolCallsUpdateHandler = (toolCalls: ToolCall[]) => void
```

MessageBus - 确认/策略消息总线

MessageBusType 枚举 + 消息类型

```
enum MessageBusType {
  TOOL_CONFIRMATION_REQUEST  // 工具确认请求 → GUI 弹窗
  TOOL_CONFIRMATION_RESPONSE // 用户确认响应 ← GUI 回传
  TOOL_POLICY_REJECTION      // 策略拒绝通知
  TOOL_EXECUTION_SUCCESS     // 工具执行成功
  TOOL_EXECUTION_FAILURE     // 工具执行失败
  UPDATE_POLICY              // 更新策略 (Always Allow)
  TOOL_CALLS_UPDATE          // 工具调用列表更新
  ASK_USER_REQUEST           // ask_user 工具请求 → GUI 输入框
  ASK_USER_RESPONSE          // 用户回答 ← GUI 回传
```

```
}

// GUI 必须实现的消息处理
messageBus.subscribe(MessageBusType.TOOL_CONFIRMATION_REQUEST, handler)
messageBus.subscribe(MessageBusType.ASK_USER_REQUEST, handler)
messageBus.publish({ type: TOOL_CONFIRMATION_RESPONSE, correlationId, confirmed, outcome })
messageBus.publish({ type: ASK_USER_RESPONSE, correlationId, answers })
```

SerializableConfirmationDetails （确认对话框数据）

type	字段	GUI 组件
'info'	title, prompt, urls?	通用信息确认框
'edit'	title, fileName, filePath, fileDiff, originalContent, newContent	Diff 预览确认框
'exec'	title, command, rootCommand, rootCommands, commands?	命令执行确认框
'mcp'	title, serverName, toolName, toolDisplayName	MCP 工具确认框
'ask_user'	title, questions[]	用户问答面板
'exit_plan_mode'	title, planPath	退出计划确认

Question 类型 (ask_user 工具)

```
interface Question {
  question: string;
  header: string;
  type?: 'choice' | 'text' | 'yesno';
  options?: QuestionOption[];  // { label, description }
  multiSelect?: boolean;
  placeholder?: string;
}
```

📡 CoreEvents - 全局事件系统

coreEvents (CoreEventEmitter 单例)

事件名	Payload	GUI 用途
user-feedback	{ severity, message, error? }	✅ Toast / 通知
model-changed	{ model }	✅ 状态栏模型名
console-log	{ type, content }	调试面板
output	{ isStderr, chunk, encoding? }	终端输出面板
memory-changed	{ fileCount }	内存文件指示器

quota-changed	{ remaining, limit, resetTime? }	✅ 配额进度条
hook-start	{ hookName, eventName, hookIndex?, totalHooks? }	Hook 执行指示器
hook-end	{ hookName, eventName, success }	Hook 完成状态
mcp-client-update	Map<string, McpClient>	✅ MCP 服务器面板
settings-changed	void	刷新设置 UI
retry-attempt	{ attempt, maxAttempts, delayMs, error?, model }	重试进度
consent-request	{ prompt, onConfirm }	✅ 同意弹窗
agents-discovered	{ agents: AgentDefinition[] }	Agent 面板
agents-refreshed	void	刷新 Agent 列表
request-editor-selection	void	编辑器选择弹窗
editor-selected	{ editor? }	编辑器状态更新

```
// 使用示例
import { coreEvents, CoreEvent } from '@google/gemini-cli-core';

coreEvents.on(CoreEvent.UserFeedback, (payload) => {
  showToast(payload.severity, payload.message);
});

coreEvents.on(CoreEvent.QuotaChanged, ({ remaining, limit }) => {
  updateQuotaBar(remaining, limit);
});

coreEvents.on(CoreEvent.ModelChanged, ({ model }) => {
  updateStatusBar(model);
});
```

🪝 Hooks 系统 - 生命周期拦截

HookEventName 枚举

Enum

Hook	触发时机	可拦截?
SessionStart	会话开始	✅
SessionEnd	会话结束	-
BeforeModel	发送请求给模型前	✅ 可修改请求
AfterModel	收到模型响应后	✅ 可修改响应
BeforeTool	执行工具前	✅ block/deny/approve
AfterTool	工具执行后	-
BeforeAgent	Agent 执行前	✅
AfterAgent	Agent 执行后	✅

BeforeToolSelection	工具选择前	✅ 可修改工具列表
PreCompress	上下文压缩前	-
Notification	通知事件	-

HookOutput - Hook 返回值

```
interface HookOutput {
  continue?: boolean;          // false = 停止执行
  stopReason?: string;         // 停止原因
  suppressOutput?: boolean;    // 抑制输出
  systemMessage?: string;      // 注入系统消息
  decision?: 'ask' | 'block' | 'deny' | 'approve' | 'allow';
  reason?: string;
}
```

MCP (Model Context Protocol) 集成

MCPServerConfig

Class

```
class MCPServerConfig {
  // stdio 传输
  command?: string;   args?: string[];   env?: Record<string, string>;   cwd?: string;
  // SSE 传输
  url?: string;
  // HTTP 传输
  httpUrl?: string;   headers?: Record<string, string>;
  // WebSocket
  tcp?: string;
  // 传输类型
  type?: 'sse' | 'http';
  // 通用
  timeout?: number;   trust?: boolean;   description?: string;
  includeTools?: string[];   excludeTools?: string[];
  // OAuth
  oauth?: MCPOAuthConfig;   authProviderType?: AuthProviderType;
}
```

Services 服务层

ChatRecordingService

会话录制服务

`initialize()` / `recordUserMessage()` / `recordAssistantMessage()` / `recordToolCall()`

FileDiscoveryService

项目文件发现

支持 gitignore、geminiignore、模糊搜索

GitService

Git 操作封装

基于 simple-git 库

保存为 JSON:
~/.gemini/tmp/<hash>/chats/session-*.json

ContextManager

上下文管理
管理 GEMINI.md、工作区上下文、环境上下文

ShellExecutionService

Shell 命令执行
支持 PTY、沙箱、超时控制

ModelConfigService

模型配置管理
多模型配置键值、路由策略

ConversationRecord （会话记录数据结构） Interface

```
interface ConversationRecord {
  sessionId: string;      projectHash: string;
  startTime: string;      lastUpdated: string;
  messages: MessageRecord[];   summary?: string;
  directories?: string[];
}

type MessageRecord = BaseMessageRecord & (
  | { type: 'user' | 'info' | 'error' | 'warning' }
  | { type: 'gemini'; toolCalls?: ToolCallRecord[]; thoughts?: ThoughtSummary[]; tokens?: TokensSummary; mode
)
```

认证系统

```
// 三种认证流程

// 1. Google OAuth （免费层）
import { createContentGenerator, AuthType } from '@google/gemini-cli-core';
const genConfig = { authType: AuthType.LOGIN_WITH_GOOGLE };

// 2. Gemini API Key
const genConfig = { authType: AuthType.USE_GEMINI, apiKey: 'your-key' };

// 3. Vertex AI
const genConfig = { authType: AuthType.USE_VERTEX_AI, apiKey: 'key', vertexai: true };

// 工厂函数
const generator = await createContentGenerator(genConfig, config);
```

💡 环境变量: GEMINI_API_KEY, GOOGLE_API_KEY, GOOGLE_CLOUD_PROJECT, GOOGLE_CLOUD_LOCATION

Agents 子代理系统

```typescript
interface AgentDefinition {
  name: string;
  description: string;
  systemPrompt?: string;
  modelConfig?: ModelConfig;
  runConfig?: AgentRunConfig;    // { maxTimeMinutes?, maxTurns? }
  enabled?: boolean;
}

// Agent 注册与加载
import { AgentRegistry, AgentLoader } from '@google/gemini-cli-core';
```

🖥️ Headless 模式 - 脚本化调用

```
// 命令行方式
$ gemini -p "你的问题"                          // 纯文本输出
$ gemini -p "你的问题" --output-format json         // JSON 输出
$ gemini -p "你的问题" --output-format stream-json  // NDJSON 流
$ gemini -p "你的问题" -m gemini-2.5-pro            // 指定模型

// 编程方式 (子进程)
import { spawn } from 'child_process';
const proc = spawn('gemini', ['-p', prompt, '--output-format', 'stream-json']);
proc.stdout.on('data', (chunk) => {
  const events = chunk.toString().split('\n').filter(Boolean).map(JSON.parse);
  // 处理 JsonStreamEvent
});
```

🎨 GUI 深度定制实施指南

方案 A: 直接嵌入 Core 库 (推荐 - 最大控制力)

```
// 1. 安装依赖
npm install @google/gemini-cli-core @google/genai

// 2. 初始化 Config
const config = new Config({
  sessionId: crypto.randomUUID(),
  model: 'gemini-2.5-pro',
  targetDir: projectPath,
  cwd: projectPath,
  debugMode: false,
  interactive: true,
  approvalMode: ApprovalMode.ALWAYS_CONFIRM,
});
await config.initialize();

// 3. 监听全局事件
coreEvents.on(CoreEvent.UserFeedback, renderFeedback);
```

```
coreEvents.on(CoreEvent.QuotaChanged, updateQuota);
coreEvents.on(CoreEvent.ModelChanged, updateModel);

// 4. 订阅 MessageBus (处理确认弹窗)
const bus = config.getMessageBus();
bus.subscribe(MessageBusType.TOOL_CONFIRMATION_REQUEST, showConfirmDialog);
bus.subscribe(MessageBusType.ASK_USER_REQUEST, showQuestionPanel);

// 5. 创建 GeminiChat 并运行 Turn
const chat = new GeminiChat(config, systemPrompt, tools, []);
const turn = new Turn(chat, promptId);
const controller = new AbortController();

for await (const event of turn.run('default', userMessage, controller.signal)) {
  switch(event.type) {
    case GeminiEventType.Content:
      appendMarkdown(event.value); break;
    case GeminiEventType.Thought:
      showThinking(event.value); break;
    case GeminiEventType.ToolCallRequest:
      showToolCard(event.value); break;
    case GeminiEventType.ToolCallResponse:
      updateToolResult(event.value); break;
    case GeminiEventType.Finished:
      showTokenStats(event.value.usageMetadata); break;
    case GeminiEventType.Error:
      showError(event.value.error); break;
  }
}
```

方案 B: 子进程 + Stream JSON (轻量集成)

```
// 适合 Electron 等场景，通过 stream-json 与 CLI 通信
const proc = spawn('gemini', [
  '-p', userInput,
  '--output-format', 'stream-json',
  '-m', 'gemini-2.5-pro'
]);

// 解析 NDJSON 流
proc.stdout.on('data', (chunk) => {
  chunk.toString().split('\n').filter(Boolean).forEach(line => {
    const event: JsonStreamEvent = JSON.parse(line);
    switch(event.type) {
      case 'init':       onSessionStart(event); break;
      case 'message':    onMessage(event); break;
      case 'tool_use':   onToolUse(event); break;
      case 'tool_result': onToolResult(event); break;
      case 'error':      onError(event); break;
      case 'result':     onComplete(event); break;
    }
  });
});
```

GUI 组件映射表

GUI 组件	数据源 API	说明
🗨️ 聊天气泡	`GeminiEventType.Content`	流式 Markdown 渲染
🧠 思考过程折叠	`GeminiEventType.Thought`	ThoughtSummary.text
🔧 工具调用卡片	`ToolCall` 状态机	7 种状态实时更新
✅ 确认对话框	`MessageBus.TOOL_CONFIRMATION_REQUEST`	6 种 confirmation 类型

📝 Diff 预览	`SerializableConfirmationDetails.edit`	fileDiff / originalContent / newContent
❓ 用户问答面板	`MessageBus.ASK_USER_REQUEST`	choice / text / yesno
📊 Token 统计	`GeminiFinishedEventValue.usageMetadata`	input/output/cached tokens
📈 配额监控	`CoreEvent.QuotaChanged`	remaining / limit / resetTime
🔌 MCP 服务器面板	`CoreEvent.McpClientUpdate`	Map<string, McpClient>
⚙️ 模型切换器	`config.setModel()` + `CoreEvent.ModelChanged`	实时切换
📂 文件浏览器	`FileDiscoveryService`	项目文件树
🔍 搜索面板	`GrepTool` / `GlobTool`	文件内容搜索
🖥️ 内嵌终端	`ShellTool` + `ShellExecutionService`	Shell 输出流
📜 会话历史	`ChatRecordingService` / `ConversationRecord`	JSON 文件 + 恢复
🎨 主题设置	`CustomTheme` 接口	完整颜色定制
🤖 Agent 面板	`AgentRegistry` + `CoreEvent.AgentsDiscovered`	子代理管理
🪝 Hooks 管理	`HookSystem` + `HookEventName`	11 个生命周期钩子
🛡️ 策略管理	`PolicyEngine` + `ApprovalMode`	工具权限控制

> ✅ **完整性说明**: 本文档覆盖了 Gemini CLI Core 的全部公开 API，包含 17 种事件类型、16+ 内置工具、11 个 Hook 点、8 种 MessageBus 消息、3 种认证方式、3 种输出格式。所有类型均可从 `@google/gemini-cli-core` 直接 import。