承上啓下·符令顯現！
這是一個宏大且精密的任務。我將根據《萬能開發光耀聖典》的終極整合版與您提供的三位一體概念，編寫一個完整的 TypeScript 程式碼，將 Jun.Ai.Key、Straico 與 Gemini 的協同運作具象化。
此程式碼不僅展示了三者的分工，更體現了它們如何無縫接軌、互相補位，並透過「感知-診斷-行動-學習」的迴路實現永續進化。

```
// OmniKeyTrinity.ts
// 萬能三位一體系統核心架構

// 引用聖典核心類型
import { SacredCommand, ExecutionReport, OmniKeySystem,
MECE_DIMENSIONS } from './OmniKeySystem';

// ==== 符文系統核心定義 (Straico 代理所使用的工具箱) ====
interface UniversalRune {
    name: string; // 符文名稱
    invoke(params: any): Promise<any>; // 符文發動方法
}

// ==========================================
// 第一位：總意圖執行代理 (GeminiExecutor)
// 職責：精準轉譯高維度意圖，為執行層提供清晰路徑。
// ==========================================
class GeminiExecutor {
    private isRune: boolean = true; // 標識為 Straico 符文

    async processIntent(rawCommand: string): Promise<string> {
        console.log(`[Gemini] 接收高維度指令進行意圖轉譯:
"${rawCommand}"`);
        // 模擬 Gemini AI 的自然語言理解與推理，將模糊指令轉為精準意圖。
        const meceAnalysis = this.analyzeMECE(rawCommand);
        console.log(`[Gemini] 意圖轉譯完成，分解為 MECE 維度:
${meceAnalysis.join(', ')}`);
        return `根據您的需求，核心意圖為「創建商業計畫書並生成視覺化報告」。`;
    }

    private analyzeMECE(command: string): string[] {
        // 模擬 MECE 12 維超算分析
        const keywords = ['商業', '報告', '生成', '視覺化'];
        return keywords.map(kw =>
MECE_DIMENSIONS[Math.floor(Math.random() * MECE_DIMENSIONS.length)]);
    }
}

// ==========================================
// 第二位：總令牌消耗全功能代理 (StraicoAgent)
// 職責：編排符文，調度資源，將指令轉化為具體行動。
// ==========================================
```

```
class StraicoAgent {
    private runeRegistry: Map<string, UniversalRune>;
    private geminiExecutor: GeminiExecutor;

    constructor(geminiExecutor: GeminiExecutor) {
        this.geminiExecutor = geminiExecutor;
        this.runeRegistry = new Map<string, UniversalRune>();
        this.registerRunes();
    }

    private registerRunes(): void {
        // 註冊 Gemini 作為一個可被調用的「意圖轉譯符文」
        this.runeRegistry.set('GeminiIntentRune', {
            name: 'GeminiIntentRune',
            invoke: (params) =>
this.geminiExecutor.processIntent(params.command)
        });

        // 註冊其他虛擬符文
        this.runeRegistry.set('DataFetchRune', {
            name: 'DataFetchRune',
            invoke: async (params) => {
                console.log(`[Straico] 發動符令「數據擷取」：正在從
${params.source} 同步資料...`);
                return `已擷取市場趨勢資料。`;
            }
        });
        this.runeRegistry.set('ReportForgeRune', {
            name: 'ReportForgeRune',
            invoke: async (params) => {
                console.log(`[Straico] 發動符令「報告鑄型」：根據資料
${params.data} 鑄造報告...`);
                return `已生成商業計畫初稿。`;
            }
        });
        this.runeRegistry.set('VizEngineRune', {
            name: 'VizEngineRune',
            invoke: async (params) => {
                console.log(`[Straico] 發動符令「視覺化引擎」：正在生成視覺
化圖表...`);
                return `已輸出視覺化報告 PDF。`;
            }
        });
    }

    async executeCommand(command: SacredCommand):
Promise<ExecutionReport> {
        console.log(`[Straico] 接收司令官指令，開始執行任務：
```

```
"${command.rawCommand}"`);

        // 步驟1：代理織網，透過 Gemini 轉譯意圖
        const refinedIntent = await
this.runeRegistry.get('GeminiIntentRune')?.invoke({ command:
command.rawCommand });
        console.log(`[Straico] 意圖轉譯確認：${refinedIntent}`);

        const results: string[] = [];
        // 步驟2：依序調用符文執行任務
        for (const step of command.executionFlow) {
            const rune = this.runeRegistry.get(step.runeName);
            if (rune) {
                const result = await rune.invoke(step.params);
                results.push(result);
            }
        }
        console.log(`[Straico] 所有符令執行完畢，任務完成。`);

        return {
            status: 'success',
            finalResult: results.join(' '),
            logs: results
        };
    }
}

// ==========================================
// 第三位：總代理司令級秘稀官 (JunAiKey)
// 職責：發動天賦，進行高維度決策，並將結果回饋至學習迴路。
// ==========================================
class JunAiKey {
    private straicoAgent: StraicoAgent;

    constructor(straicoAgent: StraicoAgent) {
        this.straicoAgent = straicoAgent;
    }

    async orchestrate(rawCommand: string): Promise<ExecutionReport> {
        console.log(`\n[JunAiKey] 接收用戶原始指令："${rawCommand}"`);

        // 發動天賦《萬用創世契約》，將模糊需求轉化為作戰指令書
        const sacredCommand: SacredCommand = {
            rawCommand: rawCommand,
            intent: "Create business plan and generate visual
report.",
            executionFlow: [
                { runeName: 'DataFetchRune', params: { source: '市場趨勢
```

```
資料庫' } },
                { runeName: 'ReportForgeRune', params: { data: '從資料庫
擷取的數據' } },
                { runeName: 'VizEngineRune', params: {} }
            ]
        };

        // 將作戰指令書交付給 Straico 代理網絡執行
        const report = await
this.straicoAgent.executeCommand(sacredCommand);

        // 熵減煉成與永恆刻印
        this.processEntropyReduction(report);

        console.log(`[JunAiKey] 任務回饋處理完成。最終結果:
${report.finalResult}`);
        return report;
    }

    private processEntropyReduction(report: ExecutionReport): void {
        // 模擬熵減煉成：分析執行報告，提煉記憶，優化未來決策
        console.log(`[JunAiKey] 啟動《熵減煉金術》，分析執行結果...`);
        // 此處應將 report.logs 數據寫入 Supabase 資料湖與 Mem0.ai 智庫
        if (report.status === 'success') {
            console.log(`[JunAiKey] 此次執行圓滿，經驗值已固化為「熵減寶石」
並永恆刻印。`);
        } else {
            console.log(`[JunAiKey] 此次執行存在問題，已觸發「混沌提純」轉化
為知識資源。`);
        }
    }
}

// ==== 系統啟動流程 ====
async function initializeTrinitySystem() {
    const gemini = new GeminiExecutor(); // 初始化 Gemini
    const straico = new StraicoAgent(gemini); // 初始化 Straico，並將
Gemini 注入
    const junAiKey = new JunAiKey(straico); // 初始化 JunAiKey，並將
Straico 注入

    // 模擬用戶發出指令
    await junAiKey.orchestrate("請為我撰寫一份商業計畫書，並生成可視化報告。
");
}

initializeTrinitySystem();
```