# 🧩 Hook 開發與使用規範文件

## 1. 🎯 Hook 架構設計目的

Hooks 架構目的是為了提供更具彈性、可擴充、模組化的流程控制，解耦主流程邏輯與擴充行為，如：

* 權限檢查（RBAC）
* 發送通知（Email / Slack / Log）
* 外部服務整合（IAM / 第三方 API）
* 條件式資料處理（自訂欄位、自動標籤）

## 2. ✅ 何時應該使用 Hook？

| 使用情境 | 適合使用 Hook？ | 說明 |
| --- | --- | --- |
| 主流程控制 | ❌ 否 | 建議使用傳統 function 實作 |
| 副作用邏輯 | ✅ 是 | 如：通知、審核、紀錄 |
| 模組化擴充 | ✅ 是 | 插拔 plugin 方式掛載邏輯 |
| 權限動態變動 | ✅ 是 | 不同權限掛不同邏輯 |
| 複雜流程鏈結 | ✅ 是 | 使用 HookGraph 實現 DAG 流程 |

## 3. 🛠 Plugin 註冊樣板

// plugin/my\_plugin.go  
  
package plugin  
  
import (  
 "fmt"  
 "your\_project/internal/hooks"  
)  
  
type MyPlugin struct{}  
  
func (p \*MyPlugin) Name() string {  
 return "MyPlugin"  
}  
  
func (p \*MyPlugin) GetHookNames() []string {  
 return []string{"create\_account"}  
}  
  
func (p \*MyPlugin) RegisterHooks(hm \*hooks.HookManager) {  
 hooks.HookBuilders{  
 {  
 HookName: "create\_account",  
 Description: "Create audit log after account creation",  
 Permissions: "admin",  
 Priority: 20,  
 Handler: p.handleCreateAudit,  
 },  
 }.RegisterHookDefinitions(hm, p.Name())  
}  
  
func (p \*MyPlugin) handleCreateAudit(ctx \*hooks.HookContext) hooks.HookResult {  
 accountID, \_ := ctx.GetEnvData("account\_id")  
 fmt.Println("[MyPlugin] 寫入審計日誌", accountID)  
 return ctx.Success("audit recorded")  
}  
  
func init() {  
 hooks.RegisterPluginType("MyPlugin", func() hooks.Plugin {  
 return &MyPlugin{}  
 })  
}

## 4. 🔍 Plugin 測試樣板

// test/plugin\_myplugin\_test.go  
  
func TestCreateAccountHook(t \*testing.T) {  
 ctx := hooks.NewHookContext("tester", map[string]interface{}{})  
 ctx.SetEnvData("account\_id", "abc123")  
 ctx.SetUserData("permissions", "admin")  
  
 manager := hooks.NewHookManager()  
 plugin := &plugin.MyPlugin{}  
 plugin.RegisterHooks(manager)  
  
 err := manager.Execute("create\_account", ctx, false)  
 assert.Nil(t, err)  
 assert.Equal(t, true, ctx.LastResult().Success)  
}

## 5. 🚀 開發者 Onboarding 指南

### 步驟 1️⃣：理解架構角色

* HookManager：註冊 / 執行 hook，負責調度
* HookContext：儲存執行上下文資料
* HookGraph：進階流程控制圖，支援 DAG 流程
* Plugin：實作模組的實體，每個 plugin 可註冊一個或多個 hook 行為

### 步驟 2️⃣：建立一個 Plugin

cp plugin/sample\_plugin.go plugin/my\_custom\_plugin.go  
vi plugin/my\_custom\_plugin.go

* 改 Name() 名稱為你 plugin 名稱（請用 PascalCase）
* 在 RegisterHooks() 加上你要註冊的 hook 名稱與 handler

### 步驟 3️⃣：將 Plugin 加入載入流程

請確認 hooks.RegisterPluginType() 已註冊至全域註冊機制

func init() {  
 hooks.RegisterPluginType("MyPlugin", func() hooks.Plugin {  
 return &MyPlugin{}  
 })  
}

### 步驟 4️⃣：撰寫對應測試

* 請為每個 Plugin 寫一份 unit test 來驗證其在 hook 執行時是否符合預期行為

## 6. 📚 常見命名規則

* Plugin 名稱：MyPlugin（PascalCase）
* Hook 名稱：snake\_case（如 create\_account, send\_notification）
* Permission 權限：以字串區分，如 admin, user, moderator
* Handler 方法：以 handleXxx(ctx) 命名方式撰寫

## 7. 🧪 Hook 測試建議策略

| 測試類型 | 建議 |
| --- | --- |
| 單元測試 | 每個 plugin 寫獨立測試 |
| 集成測試 | 透過 Process(input) 模擬 hook 鏈 |
| 性能測試 | 對大量連續 hook 呼叫測試 latency |

如需協助撰寫 plugin 或測試模板，可聯絡框架維護者。