Lesson 02: Table 資料結構

歡迎來到第二課！今天我們要學習 Lua 中最強大、最靈活的資料結構：Table。如果說變數像是單獨的盒子，那麼 Table 就像是一個萬能的收納櫃，裡面可以有很多隔間，每個隔間都能放不同的東西。

什麼是 Table？

Table 的概念

想像你有一個多功能收納櫃：

- 有些格子是按順序排列的（像書架上的書）
- 有些格子有特殊的標籤（像抽屜上的標籤寫著「文具」、「文件」）
- 你可以隨時新增格子、移除格子、或者改變格子裡的東西

這個收納櫃就是 Lua 的 Table，它可以同時當作：

1. **陣列**（Array）：有順序的清單
2. **字典**（Dictionary）：鍵值對應的資料
3. **物件**（Object）：組織相關資料的結構

最棒的是，同一個 Table 可以混合使用這些功能！

Table 作為陣列

建立陣列

陣列就像是一排有順序的盒子，每個盒子都有編號：

```lua
-- 建立水果清單
fruits = {"蘋果", "香蕉", "橘子", "葡萄"}

-- 建立數字陣列
numbers = {10, 20, 30, 40, 50}

-- 建立混合型別的陣列
mixed = {"小明", 25, true, "學生"}
```

重要提醒：Lua 的陣列索引從 1 開始，不是 0！

存取陣列元素

使用中括號和數字來存取元素：

```lua
fruits = {"蘋果", "香蕉", "橘子", "葡萄"}

print(fruits[1])    -- 蘋果（第一個元素）
print(fruits[2])    -- 香蕉（第二個元素）
print(fruits[4])    -- 葡萄（第四個元素）
print(fruits[10])   -- nil（不存在的元素）
```

修改陣列元素

```lua
fruits = {"蘋果", "香蕉", "橘子"}

-- 修改現有元素
fruits[2] = "鳳梨"
print(fruits[2])    -- 鳳梨

-- 新增元素
fruits[4] = "草莓"
print(fruits[4])    -- 草莓

-- 顯示整個陣列
for i = 1, 4 do
    print(i, fruits[i])
end
```

陣列長度

使用 # 運算子取得陣列長度：

```lua
fruits = {"蘋果", "香蕉", "橘子", "葡萄"}
print(#fruits)      -- 4

-- 在最後新增元素
fruits[#fruits + 1] = "檸檬"
print(#fruits)      -- 5
```

陣列操作函式

Lua 提供了一些實用的函式來操作陣列：

```lua
items = {"A", "B", "C"}

-- table.insert: 插入元素
table.insert(items, "D")            -- 在最後插入
table.insert(items, 2, "X")         -- 在位置 2 插入
-- 現在 items = {"A", "X", "B", "C", "D"}

-- table.remove: 移除元素
table.remove(items)                  -- 移除最後一個
table.remove(items, 2)               -- 移除位置 2 的元素
-- 現在 items = {"A", "B", "C"}

-- table.concat: 把陣列元素連接成字串
text = table.concat(items, ", ")
print(text)          -- A, B, C

-- table.sort: 排序
numbers = {5, 2, 8, 1, 9}
table.sort(numbers)
-- 現在 numbers = {1, 2, 5, 8, 9}
```

Table 作為字典

建立字典

字典使用「鍵值對」來儲存資料，就像真正的字典一樣:

```lua
-- 學生資料
student = {
    name = "小明",
    age = 20,
    grade = "大二",
    is_graduate = false
}

-- 商品資料
product = {
    id = "P001",
    title = "Lua 程式設計書",
    price = 450,
    category = "程式設計"
}
```

存取字典資料

有兩種方式可以存取字典的資料：

```
student = {
    name = "小明",
    age = 20,
    major = "資訊工程"
}

-- 方法一：點記法（推薦，較簡潔）
print(student.name)      -- 小明
print(student.age)       -- 20

-- 方法二：中括號記法（較靈活）
print(student["name"])   -- 小明
print(student["age"])    -- 20

-- 動態鍵名時必須使用中括號記法
key = "major"
print(student[key])      -- 資訊工程
```

修改和新增字典資料

```
student = {name = "小明", age = 20}

-- 修改現有資料
student.age = 21
student["name"] = "小華"

-- 新增資料
student.email = "hello.kitty@example.com"
student["phone"] = "0912345678"

print(student.name)      -- 小華
print(student.age)       -- 21
print(student.email)     -- hello.kitty@example.com
```

刪除字典資料

```
student = {
    name = "小明",
    age = 20,
    email = "ming@example.com",
```

```
    temp_note = "暫時資料"
}

-- 設定為 nil 就是刪除
student.temp_note = nil

-- 檢查是否存在
if student.temp_note == nil then
    print("temp_note 已被刪除")
end
```

遍歷 Table

遍歷陣列

```
fruits = {"蘋果", "香蕉", "橘子", "葡萄"}

-- 方法一：使用數字索引
for i = 1, #fruits do
    print(i, fruits[i])
end

-- 方法二：使用 ipairs（推薦用於陣列）
for index, value in ipairs(fruits) do
    print(index, value)
end
```

遍歷字典

```
student = {
    name = "小明",
    age = 20,
    major = "資訊工程",
    gpa = 3.8
}

-- 使用 pairs 遍歷所有鍵值對
for key, value in pairs(student) do
    print(key, "=", value)
end
-- 輸出：
-- name = 小明
-- age = 20
```

```lua
-- major = 資訊工程
-- gpa = 3.8
```

巢狀 Table

Table 裡面可以包含其他 Table，就像收納盒裡面還有更小的收納盒：

二維陣列

```lua
-- 九九乘法表
multiplication_table = {
    {1, 2, 3, 4, 5},
    {2, 4, 6, 8, 10},
    {3, 6, 9, 12, 15}
}

-- 存取二維陣列
print(multiplication_table[2][3])   -- 6

-- 遍歷二維陣列
for i = 1, #multiplication_table do
    for j = 1, #multiplication_table[i] do
        print(string.format("%d × %d = %d", i, j,
multiplication_table[i][j]))
    end
end
```

複雜的資料結構

```lua
-- 班級資料
class = {
    name = "資工一甲",
    teacher = "王老師",
    students = {
        {name = "小明", age = 19, scores = {85, 92, 78}},
        {name = "小華", age = 20, scores = {90, 88, 95}},
        {name = "小美", age = 19, scores = {88, 85, 90}}
    },
    schedule = {
        monday = {"數學", "英文", "程式設計"},
        tuesday = {"物理", "化學", "體育"},
        wednesday = {"程式設計", "數學", "英文"}
    }
```

```lua
}

-- 存取複雜資料
print(class.name)                          -- 資工一甲
print(class.students[1].name)              -- 小明
print(class.students[1].scores[1])         -- 85
print(class.schedule.monday[1])            -- 數學
```

實用範例

範例 ：通訊錄管理

```lua
-- 建立通訊錄
contacts = {
    {name = "小明", phone = "0912345678", email = "ming@example.com"},
    {name = "小華", phone = "0987654321", email = "hua@example.com"},
    {name = "小美", phone = "0911111111", email = "mei@example.com"}
}

-- 新增聯絡人
function add_contact(name, phone, email)
    table.insert(contacts, {
        name = name,
        phone = phone,
        email = email
    })
end

-- 查詢聯絡人
function find_contact(name)
    for i, contact in ipairs(contacts) do
        if contact.name == name then
            return contact
        end
    end
    return nil
end

-- 使用範例
add_contact("小李", "0922222222", "li@example.com")

local found = find_contact("小明")
if found then
```

```
    print("找到了：" .. found.name .. ", 電話：" .. found.phone)
end
```

常見錯誤和注意事項

1. 索引從 1 開始

```
-- 錯誤示範
fruits = {"蘋果", "香蕉", "橘子"}
print(fruits[0])    -- nil（不是第一個元素）

-- 正確做法
print(fruits[1])    -- 蘋果（第一個元素）
```

2. nil 會「打洞」

```
-- 設定陣列元素為 nil 會造成「洞」
items = {"A", "B", "C", "D"}
items[2] = nil

print(#items)       -- 可能是 4，也可能是 1（不確定）

-- 使用 table.remove 比較安全
items = {"A", "B", "C", "D"}
table.remove(items, 2)
print(#items)       -- 確定是 3
```

3. 鍵值的型別

```
-- 不同型別的鍵是不同的
t = {}
t["1"] = "字串鍵"
t[1] = "數字鍵"

print(t["1"])      -- 字串鍵
print(t[1])        -- 數字鍵
```

4. Table 比較

```lua
-- Table 比較的是引用，不是內容
t1 = {1, 2, 3}
t2 = {1, 2, 3}
t3 = t1

print(t1 == t2)      -- false (不同的 Table)
print(t1 == t3)      -- true (同一個 Table)
```

練習時間

基礎練習

1. **建立陣列**:
    - 建立一個包含你喜歡的 5 種食物的陣列
    - 印出第 3 個食物
    - 新增一種新食物到陣列末尾
2. **建立字典**:
    - 建立一個描述你自己的 Table（姓名、年齡、興趣等）
    - 修改其中一個值
    - 新增一個新的欄位
3. **遍歷練習**:
    - 遍歷你建立的陣列，印出所有食物
    - 遍歷你的個人資料 Table，印出所有鍵值對

grandMA2 燈光群組管理實戰應用

Table 在燈光控制中是無敵的工具！讓我們看看如何用 Table 來管理舞台上的所有燈光:

舞台燈光分區管理

```lua
-- Plugin：舞台燈光分區控制系統
function Start()
    -- 定義舞台燈光群組
    local stage_lights = {
        front = {1, 2, 3, 4},          -- 前場燈光編號
        back = {5, 6, 7, 8},           -- 後場燈光編號
        side_left = {9, 10, 11},       -- 左側燈光
        side_right = {12, 13, 14},     -- 右側燈光
```

```
        effects = {15, 16, 17, 18}     -- 效果燈組
    }

    -- 燈光預設設定
    local light_config = {
        fade_time = 2,                 -- 淡入淡出時間
        default_intensity = 80,        -- 預設亮度
        warmup_time = 0.5              -- 預熱時間
    }

    gma.echo("=== 舞台燈光分區控制系統啟動 ===")

    -- 依序點亮各區域
    ControlLightGroup(stage_lights.front, 100, light_config.fade_time,
"前場")
    gma.sleep(2)

    ControlLightGroup(stage_lights.back, 60, light_config.fade_time, "後
場")
    gma.sleep(2)

    ControlLightGroup(stage_lights.side_left, 40,
light_config.fade_time, "左側")
    gma.sleep(1)

    ControlLightGroup(stage_lights.side_right, 40,
light_config.fade_time, "右側")
    gma.sleep(2)

    -- 效果燈閃爍
    EffectStrobe(stage_lights.effects, 3, 0.2)
end

-- 控制燈光群組的函式
function ControlLightGroup(lights, intensity, fade_time, group_name)
    gma.echo("控制 " .. group_name .. " 燈光，亮度：" .. intensity .. "%")

    for _, light_id in ipairs(lights) do
        local command = "Fixture " .. light_id .. " at " .. intensity
        if fade_time > 0 then
            command = command .. " fade " .. fade_time
        end
        gma.cmd(command)
    end
end
```

```lua
-- 效果燈閃爍函式
function EffectStrobe(lights, times, interval)
    gma.echo("效果燈閃爍 " .. times .. " 次")

    for i = 1, times do
        -- 全亮
        for _, light_id in ipairs(lights) do
            gma.cmd("Fixture " .. light_id .. " at 100")
        end
        gma.sleep(interval)

        -- 關閉
        for _, light_id in ipairs(lights) do
            gma.cmd("Fixture " .. light_id .. " at 0")
        end
        gma.sleep(interval)
    end
end
```

進階應用：燈光場景管理系統

```lua
-- 定義多個場景的燈光設定
local lighting_scenes = {
    opening = {
        name = "開場",
        lights = {
            {group = {1, 2, 3, 4}, intensity = 100, color = "white"},
            {group = {5, 6, 7, 8}, intensity = 50, color = "blue"},
            {group = {15, 16}, intensity = 80, color = "red"}
        },
        fade_time = 3
    },

    romantic = {
        name = "浪漫場景",
        lights = {
            {group = {1, 3}, intensity = 30, color = "warm_white"},
            {group = {9, 10, 11, 12, 13, 14}, intensity = 20, color =
"pink"},
            {group = {15, 16, 17, 18}, intensity = 0, color = "off"}
        },
        fade_time = 5
    },
```

```
    dramatic = {
        name = "戲劇場景",
        lights = {
            {group = {2, 4}, intensity = 90, color = "white"},
            {group = {5, 6, 7, 8}, intensity = 10, color = "deep_blue"},
            {group = {15, 16, 17, 18}, intensity = 100, color = "red"}
        },
        fade_time = 1.5
    }
}

function PlayScene(scene_name)
    local scene = lighting_scenes[scene_name]
    if not scene then
        gma.echo("錯誤：找不到場景 " .. scene_name)
        return
    end

    gma.echo("播放場景：" .. scene.name)

    -- 執行場景中的每個燈光設定
    for _, light_setup in ipairs(scene.lights) do
        for _, fixture_id in ipairs(light_setup.group) do
            local cmd = string.format("Fixture %d at %d", fixture_id,
light_setup.intensity)

            if light_setup.color ~= "off" then
                cmd = cmd .. " " .. light_setup.color
            end

            if scene.fade_time > 0 then
                cmd = cmd .. " fade " .. scene.fade_time
            end

            gma.cmd(cmd)
        end
    end
end

function Start()
    gma.echo("=== 燈光場景管理系統 ===")

    -- 播放開場場景
    PlayScene("opening")
```

```
    gma.sleep(5)

    -- 切換到浪漫場景
    PlayScene("romantic")
    gma.sleep(5)

    -- 切換到戲劇場景
    PlayScene("dramatic")
end
```

為什麼 Table 在燈光控制中如此重要？

1. **組織燈光群組**：把相關的燈光組織在一起

```
local wash_lights = {1, 2, 3, 4, 5}       -- 洗牆燈
local spot_lights = {10, 11, 12, 13}      -- 聚光燈
local moving_lights = {20, 21, 22, 23}    -- 搖頭燈
```

2. **儲存設定參數**：像 Recast Preset plugin 的 config 設定

```
local config = {
    showBackupPopup = true,
    useSelection = nil,
    cueOnly = nil,
    unlock = true,
    telnetTimeout = 0.5
}
```

3. **管理複雜資料**：DMX 位址、顏色資料、效果參數

```
local fixture_data = {
    {id = 1, dmx_start = 1, type = "LED_PAR", channels = 7},
    {id = 2, dmx_start = 8, type = "MOVING_HEAD", channels = 16},
    {id = 3, dmx_start = 24, type = "WASH", channels = 4}
}
```

實際應用：智慧燈光預設管理器

```
-- Plugin 名稱：Smart Lighting Preset Manager
function Start()
    -- 燈光預設資料庫
    local presets = {
        {name = "全場亮", fixtures = "1 thru 50", intensity = 100, color
= "white"},
```

```
        {name = "暖光", fixtures = "1 thru 20", intensity = 70, color =
"warm_white"},
        {name = "冷光", fixtures = "21 thru 40", intensity = 70, color =
"cool_white"},
        {name = "彩光", fixtures = "41 thru 50", intensity = 80, color =
"rainbow"}
    }

    -- 顯示可用預設
    gma.echo("=== 可用燈光預設 ===")
    for i, preset in ipairs(presets) do
        gma.echo(i .. ". " .. preset.name)
    end

    -- 讓使用者選擇
    local choice = gma.textinput("選擇預設", "請輸入數字 (1-" .. #presets
.. ")")
    local preset_index = tonumber(choice)

    if preset_index and preset_index >= 1 and preset_index <= #presets
then
        local selected_preset = presets[preset_index]

        gma.echo("執行預設：" .. selected_preset.name)

        local cmd = "Fixture " .. selected_preset.fixtures ..
                    " at " .. selected_preset.intensity

        if selected_preset.color then
            cmd = cmd .. " " .. selected_preset.color
        end

        gma.cmd(cmd)
        gma.echo("預設執行完成！")
    else
        gma.echo("無效的選擇")
    end
end

function Cleanup()
    gma.echo("智慧燈光預設管理器已結束")
end

return Start, Cleanup
```

Table 操作在燈光控制中的應用

Table 操作	燈光控制應用	實例
陣列遍歷	批次控制燈光群組	`for _, light in ipairs(group)`
字典查詢	場景參數查找	`scenes["romantic"].fade_time`
資料插入	動態新增燈光到群組	`table.insert(group, new_light)`
巢狀結構	複雜場景設定	`scenes.act1.scene2.lighting`

小結

Table 是 Lua 的核心，掌握了 Table，你就掌握了 Lua 程式設計的精髓。在燈光控制領域，Table 更是不可或缺：

1. **燈光群組管理**：用陣列組織相關燈具
2. **設定檔管理**：用字典儲存 plugin 參數
3. **場景資料庫**：用複雜結構管理多個場景
4. **批次操作**：用 Table 遍歷實現批次控制

無論是儲存設定檔、管理燈光群組、或是建立複雜的場景系統，Table 都是你最好的朋友。現在你不只懂得資料結構，更懂得如何用它們來創造令人驚豔的燈光效果！