CH8 套件

8-1 前言

8-1-1何謂套件

- 本章中，我們要說明Go 語言如何將程式碼組織成套件 (package)
- 各位會學到如何透過套件隱藏或顯示 Go 語言中的各種物件，比如結構、介面、函式等等
- 到目前為止，我們寫的程式在規模跟複雜度上都很有限，大部份程式都只包含在一個main.go檔案中，而該檔案屬於單一套件 main (若你現在還不太懂什麼是 main 套件，也不必太擔心，本章稍後就會說明 package main的重要性)

- 然而，當你在一個開發團隊中工作時，Go 程式就很可能不再只限於單一檔案；這時的程式碼數量通常會變得龐大，並涉及多重檔案跟函式庫，你甚至得跟多位團隊成員合作開發程式

- 要是沒有將程式碼拆解成較小而且易於管理的單元，程式寫起來就會綁手綁腳

- Go 語言可以將相關的程式概念『模組化』和轉成套件，藉以解決上述大規模程式碼的複雜性問題

- 事實上，Go 語言自身的標準函式庫就是套件的最佳典範，目的是要克服相同的問題

- 目前為止，你已經見識過和用過許多內建套件了，像是fmtstrconv、error 等等。

- 這裡我們就拿 GO 語言的strings 套件舉個例。顧名思義，這個套件收集了各種操作字串用的函式，而且就只包含字串處理相關功能。因此 Go 語言開發人員需要應付字串時，就能從這個套件找到需要的東西，不需要再自己重新定 義

- 套件概念有助於將程式碼組織為模組化的單元

- 首先把一群具備單一目的的程式碼擺在一個套件中，接著把程式碼依其功能分割成個別的.go檔案，並依其功能命名，可以確保每個檔案中的各個函式只會用於特定功能

8-1-2 運用套件的好處

1. 易於維護：模組能將程式碼分割成小單元，檔案名稱也會反映各自的功能，使程式碼維護跟修改起來更容易

2. 可重複利用：重複利用具有以下優點：
    1. 沿用既有套件可以減少專案的開發時間與成本
    2. 既有套件已經經過多次的測試，因而能提升程式品質，減少潛在臭蟲
    3. 隨著套件數的增長，你可以更迅速的規畫未來的專案

3. 模組化: 模組化可以寫出更容易維護,可重複使用的程式碼，在某種程度上和可重複利用是相近的概念

8-2 使用套件

8-2-1 何謂套件

- Go語言追求的”別重複你自己”(Don’t Repeat Yourself, DRY)法則，也就是同樣的程式碼不該寫兩次
- DRY法則的第一步，就是將程式碼重構成一個個函式
- 可是如果你有成千上百個常用的函式呢 ?你如何記住這些函式?

- 有些函式可能具有類似的性質，或許某人會定義一個檔案為一個套件，取名為string.go專門負責處理字串的函式

- 這樣確實可以解決一些問題，但要是字串越來越多，單一檔案就會塞滿一堆函式，而且開發出的每一個應用程式都包含這些程式碼

- 這樣的架構當然難以維護，然而go語言允許將好幾個檔案共享一個套件，也就是你可以將同一個套件的程式碼拆分成更易維護的小單元，而每個單元應以該單元的功能命名

- Go語言並不在意套件內有多少檔案；只要同資料的所有檔案都被宣告為某某套件，他們就都可以透過該套件來存取

- 實際上，這些檔案會收集在電腦的同一個資料夾底下，而該資料夾名稱就是前述檔案所宣告的套件名稱

- 也就是說，一個套件實際上是系統中的一個目錄，它會收集屬於該套件的所有檔案

8-2-2 套件的命名

- 可以將套件名稱當成套件的自帶說明文件，所以應該簡明易懂，通常用簡單的名詞即可

- 良好的套件名稱應該符合：
  - 全小寫，不要大小寫混雜，不應該有底線
  - 簡潔
  - 使用非複數名詞
  - 避免太通用的名稱

- Go語言也鼓勵套件名稱縮寫，例如：
  - strconv(string conversion, 字串轉換)
  - regexp (regular expression, 正規表達式)
  - sync (synchronization, 同步運算)
  - os(operating system, 作業系統)

8-2-3 套件的宣告

- 每一個go語言檔案的開頭一定式套件宣告，指出該檔案屬於的套件名稱：

  package 套件名稱

- 舉例來說, 標準函式庫中的strings套件底下含有的go原始碼檔案(這裡僅列出一部份)，都會在開頭宣告說它們屬於strings套件：

```go
// 本程式僅為示範用途，無法執行

// https://golang.org/src/strings/builder.go
package strings
import (
    "unicode/utf8"
    "unsafe"
)
type Builder struct {
    addr *Builder // of receiver, to detect copies by value
    buf  []byte
}

// https://golang.org/src/strings/compare.go
package strings
func Compare(a, b string) int {
    if a == b {
        return 0
    }
    if a < b {
        return -1
    }
    return +1
}

```

```go
// https://golang.org/src/strings/replace.go
package strings
import (
    "io"
    "sync"
)
type Replacer struct {
    once   sync.Once // guards buildOnce method
    r      replacer
    oldnew []string
}
```

- 所有定義在strings套件內的的這些go原始碼檔案，其函式, 型別及變數都能統一透過匯入strings套件來存取

- 就算這些檔案分散在多個檔案之間，他們仍是同一個套件，而且從內部來說，所有檔案也可自由存取彼此的內容，例如於其他檔案的函式或型別定義

- 不過，對於套件之外的程式，套件必須決定有哪些東西是可被外部”看見”

- 下面就來看看go語言套件的匯出機制是如何運作的

8-2-4 將套件的功能匯出

- 在go語言中 , 套件的變數, 常數, 型別跟函式等如果有匯出(exported), 就可以被套件外的程式存取 , 反之沒有匯出就只能在套件內使用

- Go語言採用非常簡單的方式來決定某功能是否匯出 : 以英文大寫字母開頭的東西就是有匯出的 , 反之則是未匯出

- go語言沒有所謂的存取修飾符, 例如public或private這類關鍵字 , 而單純以功能名稱的首字大小寫來決定他是否匯出

- 注意 : 匯出必要的部分就好

- 來看一段範例：

```go
package main

import (
    "fmt"
    "strings"
)

func main() {
    str := "found me"
    if strings.Contains(str, "found") {
        fmt.Println("value found in str")
    }
}
```

- 這段程式碼匯入了strings套件 , 並呼叫其函式Contains()

- strings.Contains()會檢查str變數 , 看其中是否包含 “found” 字串 , 有的話就回傳true, 使if敘述印出訊息 :

```
value found in str
```

- 來看看strings.go的原始碼Contains()的定義 :

```
func Contains(s, substr string) bool {
    return Index(s, substr) >= 0
}
```

註 : Contain()會檢查參數s的字串是否包含substr , 它使用的Index()函式會尋找substr在s中的索引 , 找不到會傳回-1, 因此傳回0以上代表存在

- 而在strings套件中，還有另一個函式explode(), 同樣位於原始檔strings.go中，其功能是把字串根據特定的分隔字串拆成切片 (注意它是小寫開頭，代表沒有匯出的功能)

```
func explode(s string, n int) []string {
    l := utf8.RuneCountInString(s)
    if n < 0 || n > l {
        n = l
    }
    a := make([]string, n)
    for i := 0; i < n-1; i++ {
        ch, size := utf8.DecodeRuneInString(s)
        a[i] = s[:size]
        s = s[size:]
        if ch == utf8.RuneError {
            a[i] = string(utf8.RuneError)
        }
    }
    if n > 0 {
        a[n-1] = s
    }
    return a
}
```

- 現在我們來嘗試從自己的程式呼叫strings.explode(), 看看會發生甚麼事:

```go
package main

import (
    "fmt"
    "strings"
)

func main() {
    str := "found me"
    slc := strings.explode(str, 3)
    fmt.Println(slc)
}
```

- 會顯示以下錯誤, 指出你試圖存取未匯出的名稱：

```
# main
cannot refer to unexported name strings.explode Build
process exiting with code : 2 signal:  null
```

8-2 管理套件

- 現在我們了解了套件的本質和用途，也看到套件可以由多個檔案組成，並介紹了go語言命名套件的習慣和匯出功能的方法

- 但在開始打造自己的套件之前，我們需要理解go編輯器會取哪裡尋找我們應用程式所引用的套件

8-2-1 GOROOT

- Go編輯器必須知道如何找到我們用import匯入的套件原始檔
- 對於go語言的標準函式庫，編輯器會利用環境變數$GOROOT去尋找他們
- $GOROOT其實就是go語言在你電腦的安裝路徑，例如C:\Program Files\Go
- 若要檢視go環境變數，請在主控台輸入以下指令：

```
PS D:\git\Golang> go env
```

- 你會看到一連串環境變數，當中可以找到$GOROOT:

```
set GOPROXY=https://proxy.golang.org,direct
set GOROOT=C:\Program Files\Go
set GOSUMDB=sum.golang.org
set GOTMPDIR=
set GOTOOLDIR=C:\Program Files\Go\pkg\tool\windows_amd64
set GOVCS=
```

8-3-2 GOPATH

- $GOPATH通常指向使用者家目錄下的Go目錄，例如C:\Users\使用者名稱\go
- 在$GOPATH底下則會有三個子目錄：bin, pkg, src
- 當你執行go install命令時，go語言會把編譯好的二進位執行檔放在bin目錄
- 至於pkg目錄除了用來放編譯過的套件以外，也會於其mod子目錄底下存放你用go get下載的第三方套件(本章後面會說明)
- 在Go 1.11 版之前, 使用者的所有專案和套件都得置於$GOPATH/src底下，但新的Go Modules功能解除了這種限制

8-2-4 Go Modules

- 從Go 1.11 後, 新的Go Modules功能取代了$GOPATH, 這個功能從go 1.16 後也預設啟用

- 模組(module)代表一系列套件的集合，而模組路徑(module path)會被用來協助go語言尋找你的套件，這樣就不用依賴$GOPATH來放置套件

- 若你有使用到自訂或外部套件，你必須使用以下指令在專案目錄的根目錄建立一個go.mod檔案：`go mod init 模組名稱`

- 這會建立類似如下的go.mod檔案：

  module 模組名稱

  go 1.16

- go.mod內會標明此模組需要的最低go語言版本，好讓其他人可以檢查相容性

- 至於模組名稱，不需要跟專案名稱或資料夾同名，後面會再詳談

- 當你對程式加入或移除套件時，你應該在專案目錄下輸入以下命令來重整go.mod內容：

  ```
  go mod tidy
  ```

- Go Modules其實不只用來找套件而已，也可以用於套件版本控管，可以去官方文件取得更詳細的說明

補充 : GO111MODULE

- 你可以使用另一個環境變數GO111MODULE來控制go module是否啟用 :
  - GO111MODULE=off : 關閉go module, 會去$GOPATH\src找套件
  - GO111MODULE=on : 預設啟用 , 透過go module找套件
  - GO111MODULE=auto : 若專案目錄或父目錄有go.mod就啟用go module,否則去$GOPSTH找套件
- 設定方式 : go env –w GO111MODULE=on

練習：建立一個能計算形狀面積的套件

- 在第七章曾經寫過一個可以計算可種形狀面積的程式

- 這次練習的目的是將所有跟形狀面積計算有關的程式碼(介面/結構/函式)移到稱為shape的套件中，並改寫main套件主程式(至於area子目錄下)來匯入shape套件 (main()函式中的程式碼則維持不變)

- 這是本次練習的檔案結構：

Exercise08.01\

area\

main.go

shape\

shape.go

go.mod

shape 套件

- shape套件內只有一個檔案shape.go
- shape.go的檔案內容和第7章的練習中main()以外的部分相同，但這次欲匯出的介面/結構/函式必須用大寫開頭
- 同時注意，以下所有結構的方法仍維持小寫開頭，因為我們不用匯出這些功能給使用者

```go
// Package Shape implements types and methods of various shapes
package shape

import "fmt"

type Shape interface {
    area() float64
    name() string
}

type Triangle struct {
    Base   float64
    Height float64
}

type Rectangle struct {
    Length float64
    Width  float64
}

type Square struct {
    Side float64
}

func PrintShapeDetails(shapes ...Shape) {
    for _, item := range shapes {
        fmt.Printf("%s的面積: %.2f\n", item.name(), item.area())
    }
}

```

```go
func (t Triangle) area() float64 {
    return (t.Base * t.Height) / 2
}

func (t Triangle) name() string {
    return "三角形"
}

func (r Rectangle) area() float64 {
    return r.Length * r.Width
}

func (r Rectangle) name() string {
    return "長方形"
}

func (s Square) area() float64 {
    return s.Side * s.Side
}

func (s Square) name() string {
    return "正方形"
}
```

建立 go.mod

- 現在要替專案Exersise08.01建立go.mod檔，以便設定模組路徑，進而使main套件能正確匯入shape套件

- 請在專案的根目錄位置執行以下指令：

```
PS D:\git\Golang\ch8\Exercise08.01> go mod init Exercise08.01
go: creating new go.mod: module Exercise08.01
go: to add module requirements and sums:
        go mod tidy
```

- 模組名稱可以自訂，在此我們沿用專案的名稱Eercise08.01
- 同時go語言也提示，你可以用go mod tidy 重整go.mod
- 現在Eercise08.01資料夾底下會多出一個go.mod檔：

main套件

- 最後我們來撰寫mian套件，而main.go位於Eercise08.01\area子目錄下

- 這裡的重點是使用模組路徑來匯入shape套件:

```go
package main //main套件(主程式)

import "Exercise08.01/shape" //以模組路徑匯入套件

func main() {
    t := shape.Triangle{Base: 15.5, Height: 20.1}
    r := shape.Rectangle{Length: 20, Width: 10}
    s := shape.Square{Side: 10}
    shape.PrintShapeDetails(t, r, s)
}
```

執行結果

```
PS D:\git\Golang\ch8\Exercise08.01\area> go run .
三角形的面積: 155.78
長方形的面積: 200.00
正方形的面積: 100.00
```

將main套件編譯程執行檔

- 最後來將main套件編譯成一個可執行的二進位檔(注意要在area底下執行命令) :

```
PS D:\git\Golang\ch8\Exercise08.01\area> go build
```

- 這樣go語言編譯器會在area底下產生一個名為area的執行檔 :

```
PS D:\git\Golang\ch8\Exercise08.01> .\area\area.exe
三角形的面積: 155.78
長方形的面積: 200.00
正方形的面積: 100.00
```

補充

- 若將剛剛的go build指令改成go install, go語言會將執行檔放到$GOPATH/bin下

8-3-5 下載第三方模組檔案

用go get下載第三方模組或套件

- 下面我們要使用的套件為go官方提供的範例模組example(https://github.com/golang/example), 當中有一個stringutil套件, 內含一個可反轉字串的套件

- 若你打開上面的連結 , 會看到下面寫著安裝說明 :

  go get golang.org/x/example/hello

- go get後面寫的網址 , 代表下載對象是golang.org/x/example這個模組以下的hello套件

Go語言的模組套件路徑

Go 語言的模組與套件路徑，一般由以下部分組成：

- **儲存庫根路徑 (repository root path)**：例如 golang.org/x/example 或 github.com/golang/example。
- 如果模組不是定義在儲存庫的根路徑，那麼它會有**模組子目錄 (module subdirectory)**。以上例來說，example 模組就位於其儲存庫根路徑內。
- 要是模組有第二版以上的發行版本，模組路徑也應該包含版本號，如 ...example/sub/v2。
- 對於模組底下的套件，其路徑就是模組路徑加上套件路徑，如 ...example/hello。

可以看到在此儲存庫對應的就是 Github 網站的路徑。而這個 example 模組比較特別，它有註冊在 Go 語言的官方套件目錄網站 https://pkg.go.dev/，因此可用 golang.org/x/<模組名稱> 的路徑存取。稍後我們會看到，不是所有模組和套件都能使用這樣的路徑。

- 我們撰寫的主程式如下：

```go
package main

import (
    "fmt"

    "golang.org/x/example/stringutil" //匯入第三方套件
)

func main() {
    //呼叫套件來反轉字串
    fmt.Println(stringutil.Revese("!selpmaxe oG , olleH"))
}
```

- 接著在專案路徑下執行go mod init 套件名稱

```
PS D:\git\Golang\ch8\Example8-3-5> go mod init Example8.03
go: creating new go.mod: module Example8.03
go: to add module requirements and sums:
        go mod tidy
```

- 第二步是下載套件，在此依照提示用go get下載:

```
PS D:\git\Golang\ch8\Example8-3-5> go get golang.org/x/example/stringutil
go: downloading golang.org/x/example v0.0.0-20220412213650-2e68773dfca0
go: added golang.org/x/example v0.0.0-20220412213650-2e68773dfca0
```

- 這時你到系統中的$GOPATH\pkg\mod底下就會發現有以下資料夾出現, 即為你剛下載的模組

- 用go get下載模組後，go語言會自動更新go.mod的內容

- 現在回來檢查專案本身的go.mod 會看到require.....那行(沒有的話試著用go mod tidy重整一下)

- 這麼一來，專案執行時就能順利於$GOPATH/pkg/mod存取第三方套件，下面是執行結果：

```
PS D:\git\Golang\ch8\Example8-3-5> go run .
Hello, Go examples!
```

用go mod tidy 整理/更新go.mod

- 下面再來看一個例子，這回要修該剛剛的範例，改用另一個位置不同，但功能一模一樣的第三方套件https://github.com/ozgio/strutil：

```go
package main

import (
    "fmt"

    "github.com/ozgio/strutil"
)

func main() {
    fmt.Println(strutil.Reverse("!selpmaxe oG ,olleH"))
    //呼叫套件功能來反轉字串
}
```

- 為了使用這個套件，同樣可以用go get下載它

- 不過這裡可以試另一種方式：先如上加入套件路徑，再用go mod tidy 重新整理go.mod ，這會使go語言自動搜尋並下載指定的套件：

```
PS D:\git\Golang\ch8\Example8-3-5> go mod tidy
go: finding module for package github.com/ozgio/strutil
go: downloading github.com/ozgio/strutil v0.4.0
go: found github.com/ozgio/strutil in github.com/ozgio/strutil v0.4.0
```

- 以下是重新執行的結果：

```
PS D:\git\Golang\ch8\Example8-3-5> go run .
Hello, Go examples!
```

- 從以上範例可以發現，你不僅可以替自己的程式打造套件，也可以發佈到網路上供人使用

- 模組管理就先到這裡，下面我們來看看更多go語言中和運用套件有關的功能

8-4 套件的呼叫與執行

8-4-1 套件別名

- go語言允許你替套件賦予別名，你需要這麼做的理由可能如下：
  - 其他人的套件名稱不易理解，為了在程式中闡明用途，該用別稱稱呼之
  - 套件名稱可能過長，於是用別名簡化
  - 某些場合中，就算兩個套件的路徑完全不同，套件名稱卻撞名了，這時就可以用別名區隔

- 套件別名語法非常簡單，只需要將別名放在import敘述的套件名稱前面即可：

```
import   別名   套件
```

- 以下是套件別名運用的實例 (印出Hello, Go技術者們!)

```go
package main

import (
    f "fmt"
)

func main() {
    f.Println("Hello, Go 技術者們!")
}
```

8-4-2 init()函式

- go語言中的套件基本分成兩種：可執行和不可執行的
- main套件是個特殊套件，也是可以執行的套件
- mian 套建裡一定要有一個main()函式, 這就是go在go run/go build等指令會尋找並執行的對象

- 不過，任何套件檔案(包括main)還可以定義一個特殊的函式init(), 它可以用來替套件設置初始狀態或初始值

- 下面式幾個運用init()的例子：
  - 設置資料庫物件和連線
  - 初始化套件變數
  - 建立檔案
  - 載入設定組態
  - 驗證或修復程式狀態

- 對於一個套件檔案，go語言會以下面的順序呼叫init()和main()：

1. 匯入的外部套件的套件層級最先初始化
2. 接著套件自身的套件層級會初始化
3. 呼叫外部套件的init()
4. 呼叫套件本身的init()
5. 如果執行的檔案是main套件，最後會呼叫套件本身的main()函式

- 以下是一個示範init()與main()執行順序的簡單例子：

```go
package main

import (
    "fmt"
)

var name = "Gopher"

func init() {
    fmt.Println("哈囉, ", name)
}

func main() {
    fmt.Println("哈囉, main()函式")
}
```

- 執行結果：

```
哈囉，  Gopher
哈囉，main()函式
```

- 以上輸出結果證明，套件層級的變數宣告會最先被執行(字串變數name), 接著是init(), 最後是main()

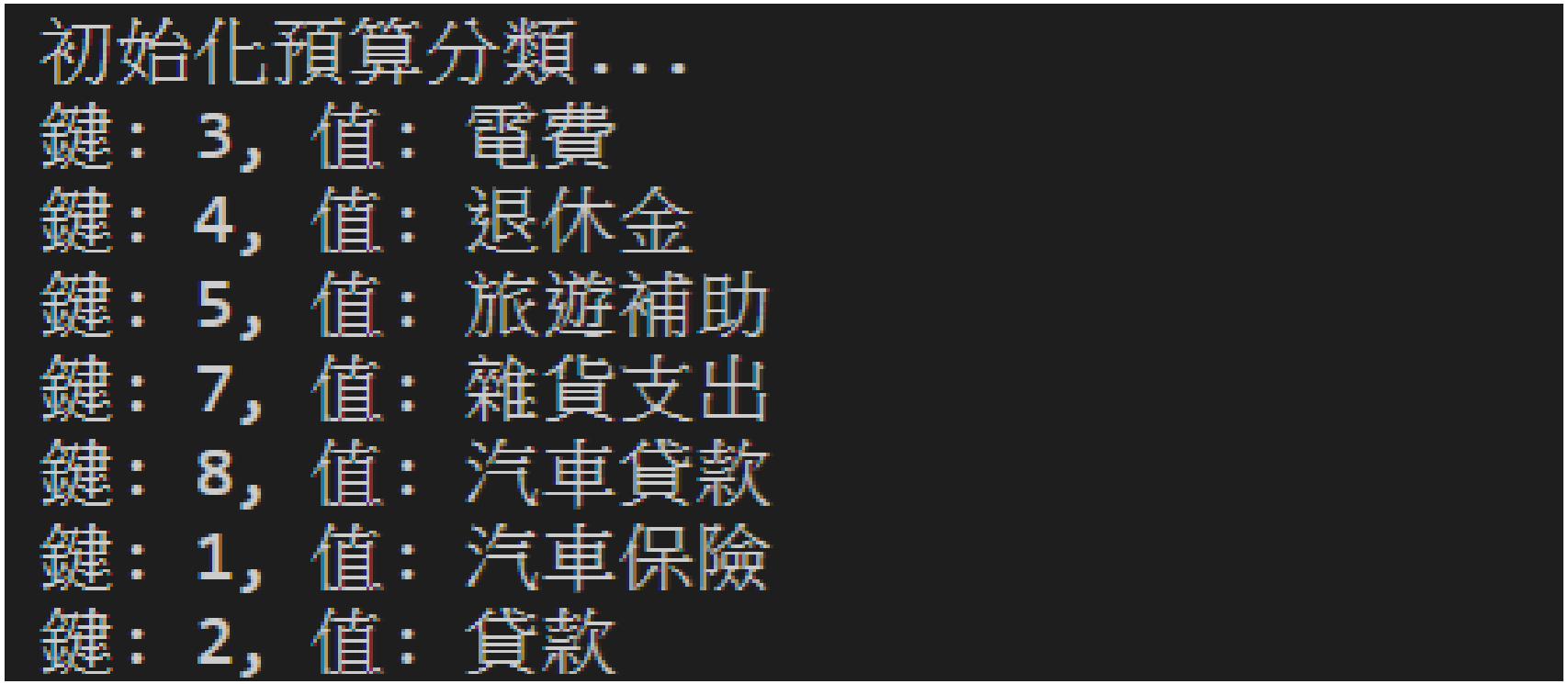
- 注意init()不能有參數或回傳值

練習：載入預算分類

- 撰寫一支程式，在執行main()函式之前之一系列的支出預算分類存入一個map集合，以便讓main()只負責印出map內容

```go
package main

import "fmt"

var budgetCategories = make(map[int]string)

func init() {
    fmt.Println("初始化預算分類...")
    budgetCategories[1] = "汽車保險"
    budgetCategories[2] = "貸款"
    budgetCategories[3] = "電費"
    budgetCategories[4] = "退休金"
    budgetCategories[5] = "旅遊補助"
    budgetCategories[7] = "雜貨支出"
    budgetCategories[8] = "汽車貸款"
}

func main() {
    for k, v := range budgetCategories {
        fmt.Printf("鍵: %d, 值: %s\n", k, v)
    }
}
```

執行結果：


初始化預算分類...
鍵： 3, 值： 電費
鍵： 4, 值： 退休金
鍵： 5, 值： 旅遊補助
鍵： 7, 值： 雜貨支出
鍵： 8, 值： 汽車貸款
鍵： 1, 值： 汽車保險
鍵： 2, 值： 貸款

8-4-3 執行多個init函式

- 在一個套件裡也可以有多個init(), 這樣就可以將初始化動作也模組化，讓程式更容易維護

- 舉例來說，假設你需要設置不同的檔案和資料庫連線，還需要修復環境的狀態以利程式執行，只用一個main()就會太過繁複，維護與除錯時也會難以區分各部分的功能

- 套件內有多重init()時，其執行順序是由上往下依序執行

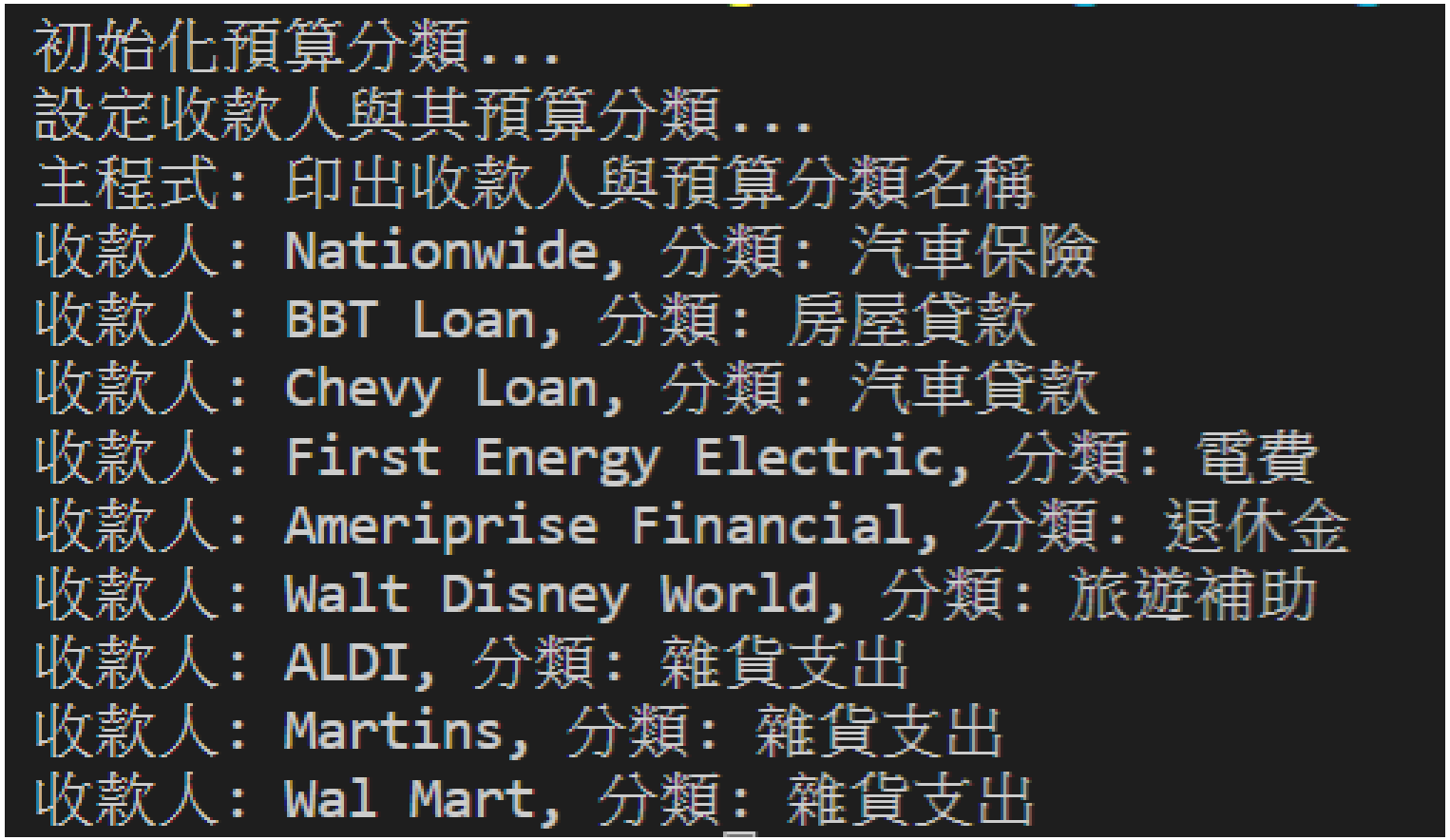
練習：將收款方與預算分類做配對

- 延伸前一個練習，將收款方資料對應到應到支出預算分類，最後將收款方與其預算分類印出

```go
package main

import "fmt"

var budgetCategories = make(map[int]string)
var payeeToCategory = make(map[string]int)

func init() {
    fmt.Println("初始化預算分類...")
    budgetCategories[1] = "汽車保險"
    budgetCategories[2] = "房屋貸款"
    budgetCategories[3] = "電費"
    budgetCategories[4] = "退休金"
    budgetCategories[5] = "旅遊補助"
    budgetCategories[7] = "雜貨支出"
    budgetCategories[8] = "汽車貸款"
}

```

```go
func init() {
    fmt.Println("設定收款人與其預算分類...")
    payeeToCategory["Nationwide"] = 1
    payeeToCategory["BBT Loan"] = 2
    payeeToCategory["First Energy Electric"] = 3
    payeeToCategory["Ameriprise Financial"] = 4
    payeeToCategory["Walt Disney World"] = 5
    payeeToCategory["ALDI"] = 7
    payeeToCategory["Martins"] = 7
    payeeToCategory["Wal Mart"] = 7
    payeeToCategory["Chevy Loan"] = 8
}

func main() {
    fmt.Println("主程式: 印出收款人與預算分類名稱")
    for k, v := range payeeToCategory {
        fmt.Printf("收款人: %s, 分類: %s\n", k, budgetCategories[v])
    }
}
```

執行結果：


初始化預算分類...
設定收款人與其預算分類...
主程式：印出收款人與預算分類名稱
收款人：Nationwide，分類：汽車保險
收款人：BBT Loan，分類：房屋貸款
收款人：Chevy Loan，分類：汽車貸款
收款人：First Energy Electric，分類：電費
收款人：Ameriprise Financial，分類：退休金
收款人：Walt Disney World，分類：旅遊補助
收款人：ALDI，分類：雜貨支出
收款人：Martins，分類：雜貨支出
收款人：Wal Mart，分類：雜貨支出

本章結束