Chapter 16 패키지

날짜	@November 24, 2021
발표자	김윤서

16.1 패키지

패키지: Go언어에서 코드를 묶는 가장 큰 단위

함수로 코드블록, 구조체로 데이터, 패키지로 함수와 구조체와 그외 코드를 묶는다. main 패키지는 프로그램 시작점을 포함한 패키지, 프로그램은 main 패키지(필수 요소) 하나와 여러 외부 패키지(선택 요소)로 구성된다.

16.1.1 main 패키지

프로그램이 실행되면 (대부분) 운영체제는 프로그램을 메모리로 옮긴다. (= 로드) 그런 다음 프로젝트 시작점부터 한 줄씩 코드를 실행한다. 시작점 → main() 함수 → main 패키지

16.1.2 그외 패키지

한 프로그램은 main 패키지 외에 다수의 다른 패키지를 포함할 수 있다. 각 기능을 제공하는 패키지를 가져다 쓰면 된다. 표준 입출력은 fmt 패키지, 암호화 기능은 crypto 패키지, 네트워크 기능은 net 패키지를 임포트해 사용한다.

16.1.3 유용한 패키지 찾기

대부분의 기능들은 이미 패키지로 배포되고 있다. 필요한 기능을 담은 패키지를 조합해 빠르게 원하는 서비스를 구현하는 것이 최선의 길이다. 새로 만들기 전에 먼저 표준 패키지에서 같은 기능을 제공하는지 찾아봐야 한다.

- https://golang.org/pkg : 표준 패키지 목록

만약 표준 패키지에서 제공하지 않는 기능이라면 공개된 외부 패키지에서 원하는 기능을 제공하는지 찾아보는 게 좋다.

- https://github.com/avelino/awesome-go : Go 언어에서 많이 사용되는 패키지

16.2 패키지 사용하기

패키지 사용: import 예약어 + 원하는 패키지 경로를 따옴표로 묶어서

다른 패키지를 가져오면 해당 패키지에서 외부로 노출하는 함수, 구조체, 변수, 상수 등 사용 가능, 외부 노출 여부는 변수명, 함수명 구조체명의 첫 글자가 대문자인지 소문자인지로 구분. 패키지명은 가져오는 패키지 경로의 가장 마지막 폴더명

16.2.1 임포트하기

```
import "fmt"
```

```
import (
  "fmt"
  "os"
)
```

- 소괄호로 여러 패키지 임포트

16.2.2 패키지 멤버에 접근하기

패키지명 + 점 . 연산자 → 제공하는 함수, 구조체 등에 접근 가능

```
fmt.Println("Hello World")
```

16.2.3 경로가 있는 패키지 사용하기

```
package main

import (
  "fmt"
  "math/rand"
)

func main() {
  fmt.Println(rand.Int())
}
```

출력문

```
5577006791947779410
```

- 경로가 있는 패키지 접근 → 마지막 폴더명인 rand만 사용

16.2.4 겹치는 패키지 문제 별칭으로 풀기

만약 패키지명이 겹치면 별칭으로 구별

```
import (
  "text/template"
  "html/template"
)
```

마지막 폴더명(패키지명)이 같다

```
import (
  "text/template"
  htemplate "html/template"
)
```

별칭은 패키지명 앞에 쓴다

```
template.New("foo").Parse(`{{define "T"}}Hello`)
htemplate.New("foo").Parse(`{{define "T"}}Hello`)
```

16.2.5 사용하지 않는 패키지 포함하기

패키지를 가져오면 반드시 사용해야 한다. 패키지 임포트 후 사용하지 않으면 에러 발생. 패키지를 직접 사용하지 않지만 부가효과를 얻고자 임포트하는 경우에는 밑줄 _을 패키지명 앞에 붙여준다.

```
import (
  "database/sql"
  _ "github.com/mattn/go-sqlite3"
)
```

16.2.6 패키지 설치하기

import로 패키지 포함 → go build를 통해서 빌드할 때 해당 패키지를 찾아서 포함 → 실행 파일 생성

Go 언어에서 import된 패키지 찾는 방법

1. Go 언어에서 기본 제공하는 패키지는 Go 설치 경로에서 찾는다. Go 설치 시 기본 패키지들까지 같이 설치한다.
2. 깃허브와 같은 외부 저장소에 저장된 패키지의 경우 다운받아서 GOPATH\pkg 폴더에 설치한다. Go 모듈에 정의된 패키지 버전에 맞게 다운로드한다.
3. 현재 모듈 아래 위치한 패키지인지 검사한다.

16.3 Go 모듈

Go 모듈: Go 패키지들을 모아놓은 Go 프로젝트 단위

Go 1.16 버전부터 Go 모듈 사용이 기본이 됐다. 모든 Go 코드는 Go 모듈 아래 있어야 한다.

go build를 하려면 반드시 Go 모듈 루트 폴더에 go.mod 파일이 있어야 한다. go.mod 파일은 모듈 이름, Go 버전, 필요한 외부 패키지 등이 명시되어 있다. go build → go.mod + 외부 저장소 패키지 버전 정보를 담고 있는 go.sum 파일을 통해 외부 패키지와 모듈 내 패키지를 합쳐서 실행 파일을 만든다.

```
go mod init [패키지명]
```

To Do: Go 모듈을 만들고 외부 패키지 활용하기

```
// Ch16/usepkg/custompkg/custompkg.go
package custompkg

import "fmt"

func PrintCustom() {
  fmt.Println("This is custom package!")
}
```

```
// Ch16/usepkg/usepkg.go
package main

import (
  "fmt"
  "usepkg/custompkg"

  "github.com/guptarohit/asciigraph"
  "github.com/tuckersGo/musthaveGo/ch16/expkg"
)

func main() {
  custompkg.PrintCustom()
  expkg.PrintSample()

  data := []float64{3, 4, 5, 6, 9, 7, 5, 8, 5, 10, 2, 7, 2, 5, 6}
  graph := asciigraph.Plot(data)
  fmt.Println(graph)
}
```

- Go 모듈을 userpkg로 만들어서 Go 모듈 아래 패키지 import할 때 "usepkg/custompkg"로 작성함

```
// Ch16/usepkg/go.mod
module usepkg

go 1.17

require (
  github.com/guptarohit/asciigraph v0.5.2
  github.com/tuckersGo/musthaveGo/ch16/expkg v0.0.0-20210809125204-68bca0d80b54
)
```

```
// Ch16/usepkg/go.sum
github.com/guptarohit/asciigraph v0.5.2 h1:aG4kATuuyHQMdTi89KKVIRIcDSIHrsKIozo/UsUE5AM=
github.com/guptarohit/asciigraph v0.5.2/go.mod h1:dYl5wwK4gNsnFf9Zp+l06rFiDZ5YtXM6x7SRWZ3KGag=
github.com/tuckersGo/musthaveGo/ch16/expkg v0.0.0-20210809125204-68bca0d80b54 h1:+cJXdzhQq58OpxkjE1PCV2FODLLdTIMBbApQP9GE4d8=
github.com/tuckersGo/musthaveGo/ch16/expkg v0.0.0-20210809125204-68bca0d80b54/go.mod h1:o12FpIqEJes/Y7CWE9BJemI9VUTQBsH7t3wYlDCw3Fw=
```

- go mod tidy 명령: Go 모듈에 필요한 패키지를 찾아서 다운로드, 필요한 패키지 정보를 go.mod 파일과 go.sum 파일에 적는다.

출력문

```
This is custom package!
This is Github expkg Sample
 10.00 ┤       ╭╮
  9.00 ┤   ╭╮  ╭╯│
  8.00 ┤   ╭╯╰╮╭╯│
  7.00 ┤   │ ╰╯╰╮│
  6.00 ┤  ╭╯    ╰╮
  5.00 ┤ ╭╯      ╰╮
  4.00 ┤╭╯        ╰
  3.00 ┼╯
  2.00 ┤
```

- Go 모듈, 기본 패키지, 모듈 내부 패키지, 외부 저장소 패키지들을 조합한 프로그램
- 다운받은 외부 패키지들은 GOPATH/pkg/mod 폴더에 버전별로 저장되어 있다. 이미 다운 받은 패키지들은 다른 모듈에서 사용되더라도 같은 버전이라면 다시 사용하게 된다.

16.4 패키지명과 패키지 외부 공개

패키지명은 쉽고 간단하게 이름지을 것, 모든 문자를 소문자로 할 것을 권장. 패키지 전역으로 선언된 첫 글자가 대문자로 시작되는 모든 변수, 상수, 타입, 함수, 메서드, 구조체, 구조체의 필드는 패키지 외부로 공개.

To Do: 외부 공개 비공개 알아보기

```
// Ch16/ex16.2/publicpkg/publicpkg.go
package publicpkg

import "fmt"

const (
  PI = 3.1415
  pi = 3.141516
)

var ScreenSize int = 1080
var screenHeight int

func PublicFunc() {
  const MyConst = 100
  fmt.Println("This is a public function", MyConst)
}

func privateFunc() {
  fmt.Println("This is a private function")
}

type MyInt int
type myString string

type MyStruct struct {
  Age  int
  name string
}

func (m MyStruct) PublicMethod() {
  fmt.Println("This is a public method")
}
```

```
func (m MyStruct) privateMethod() {
  fmt.Println("This is a private method")
}

type myPrivateStruct struct {
  Age  int
  name string
}

func (m myPrivateStruct) PrivateMethod() {
  fmt.Println("This is a private method")
}
```

```
// Ch16/ex16.2/ex16.2.go
package main

import (
  "fmt"

  "Ch16/ex16.2/publicpkg"
)

func main() {
  fmt.Println("PI:", publicpkg.PI)
  publicpkg.PublicFunc()

  var myint publicpkg.MyInt = 10
  fmt.Println("myint:", myint)

  var mystruct = publicpkg.MyStruct{Age: 18}
  fmt.Println("mystruct:", mystruct)
}
```

출력문

```
PI: 3.1415
This is a public function 100
myint: 10
mystruct: {18 }
```

- publicpkg의 pi 변수, privateFunc() 함수, MyStruct구조체의 name 필드는 접근 불가

16.5 패키지 초기화

패키지를 임포트하면 컴파일러는 패키지 내 전역 변수를 초기화한다. 그런 다음 패키지에 init() 함수가 있다면 호출해 패키지를 초기화한다. init() 함수는 반드시 입력 매개변수가 없고 반환값도 없는 함수여야 한다. 만약 어떤 패키지의 초기화 함수인 init() 함수 기능만 사용하기 원할 경우 밑줄 _을 이용해서 임포트한다

```
import (
  "database/sql"
  _ "github.com/mattn/go-sqlite3"
)
```

To Do

```
// Ch16/ex16.3/exinit/exinit.go
package exinit

import "fmt"

var (
  a = c + b
  b = f()
  c = f()
```

```
  d = 3
)

func init() {
  d++
  fmt.Println("init function", d)
}

func f() int {
  d++
  fmt.Println("f() d:", d)
  return d
}

func PrintD() {
  fmt.Println("d:", d)
}
```

```
// Ch16/ex16.3/ex16.3.go
package main

import (
  "Ch16/ex16.3/exinit"
  "fmt"
)

func main() {
  fmt.Println("main function")
  exinit.PrintD()
}
```

출력문

```
f() d: 4
f() d: 5
init function 6
main function
d: 6
```

- a = c + b : c와 b 변수가 초기화된 후 초기화
- b = f() : d++ → 4
- c = f() : d++ → 5
- a = 4 + 5 → 9
- 모든 전역 변수들 초기화 후 init() 함수 호출 : d++ → 6
- exinit 패키지 초기화 종료 → main() 함수 호출
- PrintD() : 6

연습문제

1. ScreenX 변수, ScreenY 변수, ColorDepth 상수, ResizeScreen() 함수
2.

답

f() d: 4

f() d: 5

init function a:9 b:4 c:4 d:5

과정

a = b + f() → 4 + 5 → 9

b = c → 4

c = f() → 4

d = 3 → 4 → 5