Importar paquetes en Go

Introducción

Habrá ocasiones en las que su código necesitará funciones adicionales fuera de su programa actual. En estos casos, puede usar paquetes para sumar sofisticación a su programa. Un paquete representa todos los archivos en un solo directorio en el disco. Los paquetes pueden definir funciones, tipos e interfaces a los que puede hacer referencia en otros archivos o paquetes de Go.

Este tutorial le servirá como orientación en la instalación, la importación y el solapamiento de paquetes.

Paquetes de la biblioteca estándar

La biblioteca estándar que se incluye con Go es un conjunto de paquetes. Estos paquetes contienen muchos de los bloques fundamentales para escribir software moderno. Por ejemplo, el paquete `fmt` contiene funciones básicas para dar formato a las cadenas e imprimirlas. El paquete `net/http` contiene funciones que permiten que un desarrollador cree servicios web y envíe y obtenga datos a través del protocolo `http`, entre otras posibilidades.

Para aprovechar las funciones de un paquete, deberá acceder a este con una instrucción `import`. Una instrucción `import` consta de la palabra clave `import` y el nombre del paquete.

Como ejemplo, en el archivo random.go del `programa` de Go puede importar el paquete `math/rand` para generar números aleatorios de esta manera:

random.go

```
import "math/rand"
```

Cuando importamos un paquete, hacemos que esté disponible en nuestro programa actual como un espacio de nombres por separado. Esto significa que tendremos que hacer referencia a la función en *notación de punto*, como en `package.function`.

En la práctica, una función del paquete `math/rand` podría parecerse a los siguientes ejemplos:

- `rand.Int()`, que invoca a la función para mostrar un entero al azar.
- `rand.Intn()`, que llama a la función para mostrar un elemento aleatorio desde el `0` hasta el número especificado que se proporcionó.

Crearemos un bucle `for` para mostrar la forma en que invocaremos una función del paquete `math/randm` dentro de nuestro programa `random.go`:

random.go

```
package main

import "math/rand"

func main() {
  for i := 0; i < 10; i++ {
    println(rand.Intn(25))
  }
}
```

Este programa primero importa el paquete `math/rand` en la tercera línea y luego se desplaza a un bucle `for` que se ejecutará 10 veces. Dentro del bucle, el programa imprimirá un entero aleatorio en el rango de `0` a `25`. El número entero `25` se transmite a `rand.Intn()` como su parámetro.

Cuando ejecutemos el programa con `go run random.go`, veremos 10 enteros aleatorios como resultado. Debido a que son aleatorios, es probable que obtenga diferentes enteros cada vez que ejecute el programa. El resultado tendrá un aspecto similar a este:

```
Output
6
12
22
9
6
18
0
15
6
0
```

Los enteros nunca serán menores que 0 o mayores que 24.

Al importar más de un paquete, puede usar `()` para crear un bloque. Al usar un bloque, puede evitar repetir la palabra clave `import` en cada línea. Esto hará que su código se vea más limpio:

random.go

```
import (
  "fmt"
  "math/rand"
)
```

Para aprovechar el paquete adicional, ahora podemos dar formato al resultado e imprimir la iteración en la que se generó cada número aleatorio durante el bucle:

random.go

```
package main

import (
  "fmt"
  "math/rand"
)

func main() {
  for i := 0; i < 10; i++ {
    fmt.Printf("%d) %d\n", i, rand.Intn(25))
  }
}
```

Ahora, cuando ejecutemos nuestro programa, veremos un resultado similar a este:

```
Output
0) 6
1) 12
2) 22
3) 9
4) 6
5) 18
6) 0
7) 15
8) 6
9) 0
```

A través de esta sección, aprendió a importar paquetes y usarlos para escribir un programa más sofisticado. Hasta ahora, solo utilizamos paquetes de la biblioteca estándar. A continuación, veremos la manera de instalar y usar paquetes escritos por otros desarrolladores.

Instalar paquetes

Si bien la biblioteca estándar incluye muchos paquetes fabulosos y útiles, estos están diseñados de forma intencional para* uso general *y no específico. Esto permite que los desarrolladores construyan sus propios paquetes sobre la biblioteca estándar para satisfacer sus propias necesidades específicas.

La cadena de herramientas de Go incluye el comando `go get`. Este comando le permite instalar paquetes de terceros en su entorno de desarrollo local y usarlos en su programa.

Cuando se usa `go get` para instalar paquetes de terceros, es común que la ruta canónica de un paquete haga referencia a este. Esa ruta también puede ser una ruta hacia un proyecto público alojado en un repositorio de códigos como GitHub. En ese sentido, si desea importar el paquete `flect`, usaría la ruta canónica completa:

```
$ go get github.com/gobuffalo/flect
```

La herramienta `go get` encontrará el paquete, en este caso en GitHub, y lo instalará en su `$GOPATH`.

Para este ejemplo, el código se instalaría en este directorio:

```
$GOPATH/src/github.com/gobuffalo/flect
```

Los autores originales a menudo actualizan los paquetes para corregir errores o añadir nuevas funciones. Cuando esto sucede, es posible que quiera usar la versión más reciente de ese paquete para aprovechar las nuevas funciones o las correcciones de errores. Para actualizar un paquete, puede usar el indicador `-u` con el comando `go get`:

```
$ go get -u github.com/gobuffalo/flect
```

Este comando también hará que Go instale el paquete si no se encuentra localmente. Si ya está instalado, Go intentará actualizar el paquete a la última versión.

El comando `go get` siempre permite obtener la última versión disponible del paquete. Sin embargo, puede haber actualizaciones a versiones anteriores del paquete que son aún más recientes de las que está usando y sería útil actualizarlas en su programa. Para obtener esa versión específica del paquete, tendría que usar una herramienta de **administración de paquetes**, como Go Modules.

A partir de Go 1.11, Go Modules se utiliza para administrar la versión del paquete que se intenta importar. El tema de la administración de paquetes está fuera del alcance de este artículo, pero puede obtener más información relacionada en la página de GitHub sobre módulos de Go.

Solapar paquetes importados

Quizá desee cambiar el nombre de un paquete si ya dispone de un paquete local con el mismo nombre que un paquete de terceros que esté usando. Cuando esto suceda, solapar su importación es la mejor manera de manejar el conflicto. Puede modificar los nombres de los paquetes y sus funciones dentro de Go disponiendo un nombre `alias` delante del paquete importado.

La construcción de esta instrucción tiene el siguiente aspecto:

```
import another_name "package"
```

En este ejemplo, modifique el nombre del paquete `fmt` en el archivo del programa `random.go`. Cambiaremos el nombre del paquete de `fmt` a `f` para abreviarlo. Nuestro programa modificado tendrá el siguiente aspecto:

random.go

```
package main

import (
 f "fmt"
  "math/rand"
)

func main() {
  for i := 0; i < 10; i++ {
    f.Printf("%d) %d\n", i, rand.Intn(25))
  }
}
```

Dentro del programa, ahora haremos referencia a la función `Printf` como `f.Printf` en lugar de `fmt.Printf`.

Mientras que en otros lenguajes existe una inclinación por el solapamiento de paquetes para facilitar su uso posterior en los programas, en Go esto no sucede. Por ejemplo, el solapamiento del paquete `fmt` para que pase a ser `f` no sería compatible con la guía de estilo.

Cuando se cambian los nombres de las importaciones para evitar un conflicto de nombres, debe intentar aplicar el cambio a la importación más local o específica del proyecto. Por ejemplo, si tuviera un paquete *local* llamado `strings` y también necesitara importar el paquete del *sistema* llamado `strings`, optaría por cambiar el

nombre su paquete local en lugar del paquete del sistema. Siempre que sea posible, es mejor evitar por completo el conflicto de nombres.

En esta sección, aprendió a asignar un alias a una importación para evitar conflictos con otra importación en nuestro programa. Es necesario recordar que la legibilidad y la claridad son importantes en su programa, por lo que solo debe usar el solapamiento para hacer más legible el código o cuando necesite evitar un conflicto de nombres.

Dar formato a las importaciones

Al dar formato a las importaciones, puede clasificar los paquetes en un orden específico que hará que su código sea más uniforme. Además, esto evitará que se produzcan confirmaciones aleatorias cuando lo único que cambie sea el orden de clasificación de las importaciones. Debido a que la aplicación de formato a las importaciones evitará las confirmaciones aleatorias, esto evitará renovaciones innecesarias y revisiones confusas de código.

La mayoría de los editores darán formato a las importaciones de manera automática o le permitirán configurar su editor para usar `goimports`. Se considera que el uso de `goimports` en su editor es una práctica estándar, ya que intentar mantener manualmente el orden de clasificación de sus importaciones puede ser tedioso y estar sujeto a errores. Además, si se realizan cambios de estilo, `goimports` se actualizará para reflejar estos cambios de estilo. Esto garantiza que usted y cualquier persona que trabaje en su código tengan un estilo uniforme en sus bloques de importación.

A continuación, se muestra el aspecto que puede tener un bloque de importación de ejemplo antes de la aplicación de formato:

```
import (
  "fmt"
  "os"
  "github.com/digital/ocean/godo"
  "github.com/sammy/foo"
  "math/rand"
  "github.com/sammy/bar"
)
```

Ejecutar la herramienta `goimport` (como alternativa, al guardar el archivo , esta se ejecutará en la mayoría de los editores en los que venga instalada), ahora tendrá el siguiente formato:

```
import (
  "fmt"
  "math/rand"
  "os"

  "github.com/sammy/foo"
  "github.com/sammy/bar"

  "github.com/digital/ocean/godo"
)
```

Tenga en cuenta que agrupa primero todos los paquetes de la biblioteca estándar y luego paquetes de terceros junto con líneas en blanco. Esto facilita la lectura y comprensión de los paquetes que se utilizan.

A través de esta sección, aprendió que el uso `goimports` mantendrá todos los bloques de importación en un formato adecuado e impedirá renovaciones de código innecesarias entre desarrolladores que trabajen en los mismos archivos.

Conclusión

Al importar paquetes, se puede invocar funciones que no están integradas en Go. Algunos paquetes son parte de la biblioteca estándar que se instala con Go y se instalarán a través de `go get`.

Usar paquetes nos permite aportar más solidez y capacidad a nuestros programas, ya que aprovechamos el código existente. También podemos crear nuestros propios paquetes para que nosotros y otros programadores lo utilicemos en futuros programas.