Go: i tipi di dati passati per riferimento o per valore

Il linguaggio di programmazione Go, noto anche come Golang, è ampiamente utilizzato per la sua semplicità, efficienza e performance. Una delle caratteristiche chiave di Go è il modo in cui gestisce il passaggio di dati alle funzioni: alcuni tipi di dati vengono passati per valore, mentre altri vengono passati per riferimento. In questo articolo, esploreremo i concetti di passaggio per valore e passaggio per riferimento in Go e i tipi di dati associati a ciascuna modalità.

Passaggio per valore

Nel passaggio per valore, viene effettuata una copia dei dati e questa copia viene passata alla funzione. In altre parole, le modifiche apportate alla copia all'interno della funzione non influenzeranno la variabile originale al di fuori della funzione. In Go, i seguenti tipi di dati sono passati per valore:

Tipi di dati semplici

I tipi di dati semplici, come gli interi (int), i float (float64), le stringhe (string) e i booleani (bool), sono passati per valore. Questo significa che quando passi una variabile di uno di questi tipi a una funzione, la funzione riceverà una copia della variabile, non il riferimento alla variabile originale.

```
package main

import "fmt"

func modificaInt(numero int) {
    numero = 42
```

```
}

func main() {
    x := 10
    modificaInt(x)
    fmt.Println(x) // Stampa 10, poiché x è passato
per valore
}
```

Array

Gli array in Go vengono passati per valore. Anche se sono più complessi rispetto a int o string, quando passi un array a una funzione, questa riceverà una copia dell'array originale.

```
package main

import "fmt"

func modificaArray(arr [3]int) {
    arr[0] = 42
}

func main() {
    a := [3]int{1, 2, 3}
    modificaArray(a)
    fmt.Println(a) // Stampa [1 2 3], poiché l'array
è passato per valore
}
```

Struct

Le struct in Go vengono passate per valore, proprio come gli array e gli altri tipi di dati complessi. Quando passi una struct a una funzione, questa riceve una copia della struct originale.

```
package main

import "fmt"

type Persona struct {
    Nome    string
    Email     string
}

func modificaStruct(p Persona) {
    p.Nome = "Mario"
    p.Email = "mario@site.tld"
}

func main() {
    personaOriginale := Persona{"Luca",
"luca@site.tld"}
    modificaStruct(personaOriginale)
    fmt.Println(personaOriginale) // Stampa {Luca
luca@site.tld}, poiché la struct è passata per valore
}
```

Passaggio per riferimento

Nel passaggio per riferimento, viene passato un puntatore o una referenza alla variabile originale. Ciò significa che qualsiasi modifica apportata alla variabile all'interno della funzione influenzerà direttamente la variabile originale. In Go, i seguenti tipi di dati sono passati per riferimento:

Slice

Gli slice in Go sono strutture dati flessibili e sono passati per riferimento. Quando passi un slice a una funzione, la funzione riceve un riferimento al slice originale.

```
package main

import "fmt"

func modificaSlice(slice []int) {
    slice[0] = 42
}

func main() {
    s := []int{1, 2, 3}
    modificaSlice(s)
    fmt.Println(s) // Stampa [42 2 3], poiché il
slice è passato per riferimento
}
```

Map

Anche le map in Go sono passate per riferimento. Quando passi una map a una funzione, questa riceve un riferimento alla map originale.

```
package main

import "fmt"

func modificaMappa(mappa map[string]int) {
    mappa["chiave"] = 42
}
```

```
func main() {
    m := map[string]int{"chiave": 1}
    modificaMappa(m)
    fmt.Println(m) // Stampa map[chiave:42], poiché
la mappa è passata per riferimento
}
```

Puntatori

Puoi anche passare esplicitamente un puntatore a una variabile in Go, consentendo al chiamante e alla funzione di accedere e modificare la stessa variabile attraverso il puntatore.

```
package main

import "fmt"

func modificaConPuntatore(p *int) {
    *p = 42
}

func main() {
    x := 10
    modificaConPuntatore(&x)
    fmt.Println(x) // Stampa 42, poiché è stato
passato un puntatore
}
```

In conclusione, in Go il passaggio di dati alle funzioni può avvenire per valore o per riferimento, a seconda del tipo di dato. Comprendere la differenza tra passaggio per valore e passaggio per riferimento è essenziale per scrivere codice Go efficace ed evitare comportamenti

inaspettati. Utilizzando questa conoscenza, puoi selezionare il tipo di passaggio appropriato per i tuoi dati in base alle tue esigenze di programmazione.