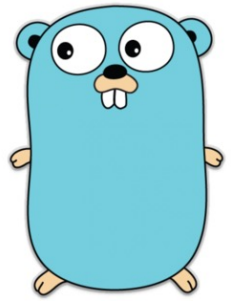
Golang Advanced

By

Prince Khanna

Agenda



Go

- Pointers in Go
- Interfaces in Go?
- Maps
- Creating server in Go
- Rest APIs in Go
- Demo
- Q&A

Pointers in Go

- What is a pointer?
- Pointer to a function
- When to use pointers?

What is Pointer?

A pointer is a variable that stores the **memory address** data referenced by another variable. Pointers have the power to **mutate data** they are pointing to.

To access the address value (*data*) represented by a variable, Go provides the **&** (*ampersand*) operator which is used in front of the variable name.

To find out the value (*data*) a pointer it points to, we need to use * operator, also called **dereferencing operator** which if placed before a pointer variable (*like & operator to get memory address*), it returns the **data** in that memory.

Pointers in Function

- Pointers can be passed to the functions as values and then we can mutate that pointer’s data in function
- For structs function also, we can have pointer as a reference, and this will make the struct object mutable in the function.

When to use and not use pointers?

- When you want to change the value in the caller function as well
- When object is too heavy
- We should avoid using pointers in case of passing value size is very small as it will increase the overhead of garbage collector.

Interfaces in Go

An **interface** is a collection of **method signatures** that a **Type** can implement (*using **methods***). Hence **interface** defines (*not declares*) the behavior of the object (*of the type Type*).

Implementing Interfaces

When a type **implements** an interface, a **variable** of that type can also be represented as the type of an **interface**.

Empty, Embedding and Mutliple Interfaces

- Empty interfaces are used just to denote something
- Every struct implements empty interface

- A struct can also implement multiple interfaces

Type Assertion

We can find out the underlying **dynamic value** of an interface using the syntax **i.(Type)** where i is a variable of type **interface** and Type is a type that implements the **interface**. Go will check if **dynamic type** of i is identical to the Type and return the **dynamic value** is possible.

if Type implements the interface but i does not have a **concrete value** of Type (*because it's nil at the moment*) then Go will panic in runtime.

value, ok := i.(Type)

Pointer vs Value receiver

When a struct has method with pointer receiver, we can not pass value type of the struct, it will give a compile time error for <Method has pointer receiver>

Maps in Golang

A map is like an array except, instead of an integer index, you can have string or any other data types as a key.

var myMap map[keyType]valueType

Concepts in Map

- Creating an empty map
  - m := make(map[keyType]valueType)
- Initializing a map
- Accessing map data
  - value, ok := m[key]
- Length of map
- Delete map element
- Map iteration
  - For loop
- Map with other data types
- Maps are referenced type

Other Concepts in Golang

- new keyword
- Switch statement
- go keyword
- Const keyword

Creating server in Golang

- net/http package
- HandleFunc
- Handler
- Chi Router

Demo

Q&A