MAVENIR™
Trust the Future

Golang Training Pointers

24/01/2023

Golang: Interfaces

Overview

- An Interface is an abstract type.
- Interface describes all the methods of a method set and provides the signatures for each method

Syntax:

```
type interface_name interface{

// Method signatures

}
```

Golang: Interface

Overview

```
type myinterface interface{



// Methods

fun1() int

fun2() float64

}
```

Golang: Interface

Important Points

- The zero value of the interface is nil.
- When an interface contains zero methods, such types of interface is known as the empty interface. So, all the types implement the empty interface.

Syntax: ***interface{}***

- **Type assertion**
- **Type Switch**

Golang: Interface

Multiple Interfaces

- It is not allowed to create same name methods in two or more interfaces
- If you try to do so, then your program will panic

Golang: Interface

Embeding Interface

- Go interface fully supports embedding
- An interface can embed other interfaces or an interface can embed other interface's method signatures in it

```
type interface_name1 interface {
    Method1()
}
type interface_name2 interface {

    Method2()
}
```

Golang: Interface

Embedding Interface

```
type interface_name1 interface {

    Method1()
}

type interface_name2 interface {

    Method2()
}
```

Golang: Interfaces

Adding Items

> Polymorphism

Golang: Goroutine

Overview

- A Goroutine is a function or method which executes independently
- Goroutine can be compared as a light weighted thread
- Every program contains at least a single Goroutine and that Goroutine is known as the **main Goroutine**
- All the Goroutines are working under the main Goroutines
- if the main Goroutine terminated, then all the goroutine present in the program also terminated

Golang: Goroutine

Delete

Syntax

```
func name(){

// statements

}


// using go keyword as the

// prefix of your function call

go name()
```

Golang: Goroutine

Anonymous Goroutine

- Goroutine with an anonymous function can be created
- Anonymous Goroutine simply by using go keyword as a prefix

Golang: Goroutine

Sync package

- Goroutine can synchronize use sync.Mutex //Smillar to posix mutex
- Goroutines can wait for each other using Wait Groups
- Critical sections using mutexes

Golang: Channels

Overview

- Channel are used to share data between goroutines
- Channels act as a pipe between the goroutines
- Channels guarantees a synchronous exchange.
- Data type should be specified at the time of declaration of a channel
- Values and pointers of built-in, named, struct, and reference types can be shared across channel
- Only one goroutine has access to a data item at any given time
- Hence data races cannot occur, by design

Golang: Channel

Types

- Channels are of 2 types
  - Unbuffered channel
    - Unbuffered channel will have only one item, this is used for synchronous communication
  - Buffered channels
    - Buffer channels will have specified number of channles, and are used for Asynchronous communication

Golang: Log

Overview

- The standard library package log provides a basic infrastructure for log management.
- Logs can be providing code tracing, profiling, and analytics
- log.SetPrefix("LOG: ")
- log.SetFlags(log.Ldate | log.Lmicroseconds | log.Llongfile)
- log.Println("init started")
- log.Fatalln("fatal message")
- log.Panicln("panic message")

Golang: Files and Directories

Overview

- OS Package provides functionality to work with files
  - Os.Create
  - _, err := os.Stat("test")
  - os.IsNotExist(err)
  - os.Rename(oldName, newName)
  - Os.Open
  - Io.Copy
  - os.Remove

Golang: Files and Directories

File stat

- fileStat.Name()
- fileStat.Size()
- fileStat.Mode()
- fileStat.ModTime()
- fileStat.IsDir()
- Os.Truncate
- File permission (os.O_RDWR|os.O_APPEND|os.O_CREATE)
- Os.Chmod