MAVENIR™
Trust the Future

Golang Training

24/01/2023

Golang Overview

Go was designed in Google in 2007

Open Sourced in 2009

- Why Go?
  - Why one more new programing language?
  - What is the Purpose?
  - How it is better or Different?
- Features of Go
  - Statically typed
  - Garbage Collection
  - Go standard library
  - Concurrency Support
  - Type Safety
  - Rich open-source package

Go Installation

- What you need to run Go Programs?
  - Go Compiler
    https://go.dev/doc/install
  - Editor
    https://code.visualstudio.com/download
- Quick trail
  - Go playground:
    https://go.dev/play/
- Go standard library:
  - https://pkg.go.dev/std
- Documentation
  - https://go.dev/doc/
- Quick Help
  - https://go.dev/doc/effective_go

MAVENIR

Golang vs Other Languages

- Golang Vs C++
  - https://github.com/mail2sada/GolangTraining/blob/main/GolangTrainingMaterial.xlsx
- Golang vs JAVA
  - https://github.com/mail2sada/GolangTraining/blob/main/GolangTrainingMaterial.xlsx

MAVENIR

Hello world with Go…

- Go source files have .go extension
- Every source file is part of a package
- Execution of go begins with main package.
- Function main() is the entry point of the code.
- Importing packages
- Hello word..
  - https://github.com/mail2sada/GolangTraining/blob/main/Day1/Day1Src/helloworld.go

Go Code Structure

- Go code is always packaged in package
- Go code begins its execution from main function in main package
- Package name is specified using package keyword
  - Syntax `package "package-name"`
- Package can be imported using import key word
  - Syntax `Import "package-name"`

Packages and Modules

- Package is a way to reuse the code
- Package is way to group code
  - Eg. math package having functions related with maths
- Every go source(.go) belongs to a package
  - Go source code will have first line as package, indicates source code in the belonging to package
  - Specified with key word package <package_name>

Packages types

- Package are of 2 types
  - Executable package
  - Utility Package
- Executable package
  - Package main is the only executable package.
- Utility Package
  - Any package other than main package is a utility package

Modules

- Module is go support for dependency management
- Module is a collection of related packages with **go.mod** at its root
- Modules are managed by go.mod file
- Module dependency is also managed by go.sum file
  - Contains cryptographic hash of bits of all project's dependent modules

go.mod

- Go.mod is a module dependency file
  - Import path of the module at the top
  - The version of go with which the module is created
  - Direct dependencies of the module.
- Below command can be used to create a module
  - go mod init {module_import_path}

go.sum

> Lists down the checksum of direct and indirect dependency required along with the version

> **go.sum** file is used to validate the checksum of each direct and indirect dependency.

Importing modules

- Modules are imported using import path
  - go mod init {module_import_path}
- Eg. go mod init github.com/learn
  - export GO111MODULE=auto
  - go mod init "github.com/mail2sada/gotraining"

go.mod

```
1   module github.com/mail2sada/gotraining
2
3   go 1.19
4
```

Importing modules

```
go.mod
1   module github.com/mail2sada/gotraining
2
3   go 1.19
4
```

- Import path of the module at the top
  - module github.com/mail2sada/gotraining
- Version of go with which the module was created
  - go 1.19

go.sum

go.sum

```
github.com/google/uuid v1.0.0 h1:b4Gk+7WdP/d3HZH8EJsZpvV7EtDOgaZLtnaNGIu1adA=
github.com/google/uuid v1.0.0/go.mod h1:TIyPZe4MgqvfeYDBFedMoGGpEw/LqOeaOT+nhxU+yHo=
github.com/pborman/uuid v1.2.1 h1:+ZZIw58t/ozdjRaXh/3awHfmWRbzYxJoAdNJxe/3pvw=
github.com/pborman/uuid v1.2.1/go.mod h1:X/NO0urCmaxf9VXbdlT7C2Yzkj2IKimNn4k+gtPdI/k=
```

Package:Some Info

- Creating user defined packages
- Importing package
- Exported and UnExported methods
- Nested packages
- Alias importing...
- Init functions
- Using external packages (go get)
- Blank Identifier in import
- Importing from external module
- Performing go get on our module...

MAVENIR

Package:Order of Execution

- The program starts with the main package
- All imported packages in the source files of the main package are initialized
- Then global variables declaration in these packages is initialized
- After this, init() function is run in these packages
- Global variables in the main package are initialized
- Init() in the main package is run if present
- main function in main package is run.
- go mod help

Package:Naming convention

- Underscore in the package name
- Camel casing or any kind of mixed caps

Package:Naming convention

- Underscore in the package name
- Camel casing or any kind of mixed caps